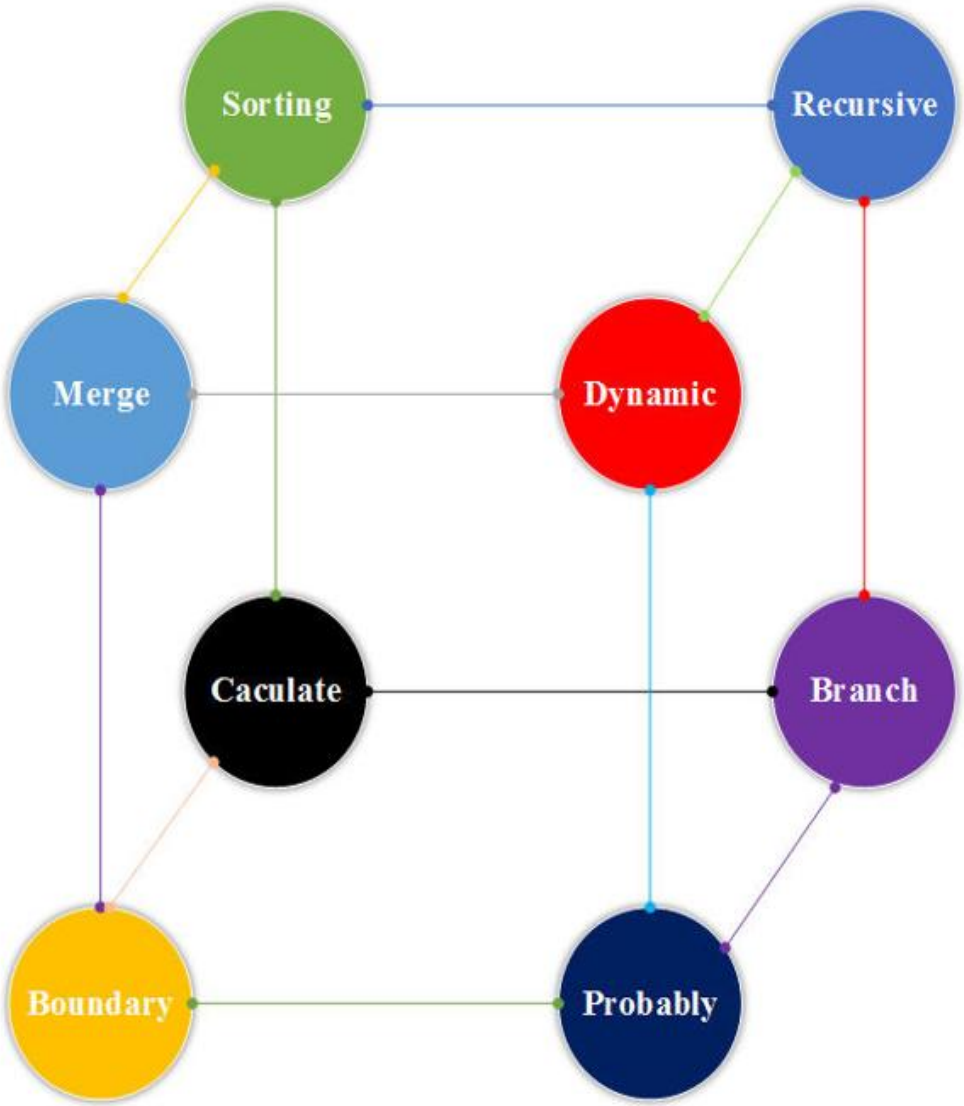
Algoritmos Em JavaScript



Explica algoritmos com belas imagens

Algoritmos Em JavaScript

YANG HU

Aprenda fácil, rápido e bem.

http://br.verejava.com

Copyright © 2021 Yang Hu All rights reserved.

ISBN: 9798595201131

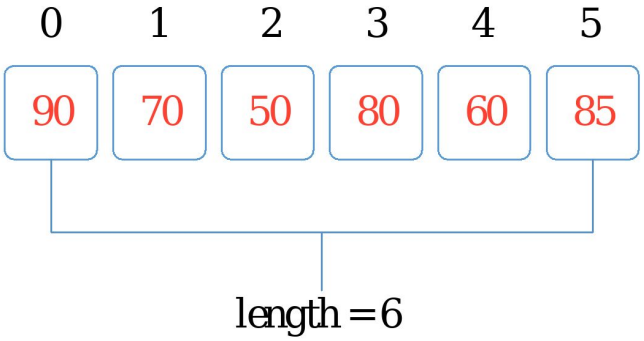
CONTEÚDO

Definição de tabela linear

Tabela Linear (Linear Table): Uma sequência de zero ou mais elementos de dados. Além do primeiro elemento, cada elemento possui apenas um elemento precursor direto e cada elemento possui apenas um elemento sucessor direto, exceto o último. A relação entre os elementos de dados é de um para um. As tabelas lineares podem ser representadas por matrizes unidimensionais.

1. Definir uma matriz unidimensional de pontuações

```
var scores = new Array( 90, 70, 50, 80, 60, 85 );
```



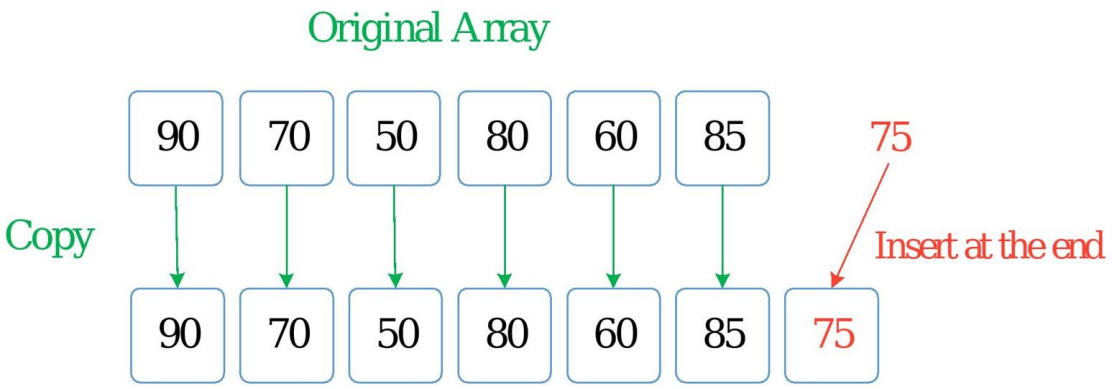
1. Crie um TestOneArray.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript"> var scores = new Array( 90, 70, 50, 80, 60, 85 );
   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }
</script>
```

Resultado:

Anexar tabela linear

1. **Adicione 75 ao final da matriz unidimensional.**



Análise: 1. Primeiro, crie uma matriz temporária (tempArray) maior que o comprimento da matriz original 2. Copie cada valor da matriz para tempArray 3. Atribua 75 ao último índice de tempArray 4. Finalmente, atribua a referência do ponteiro tempArray à matriz original;

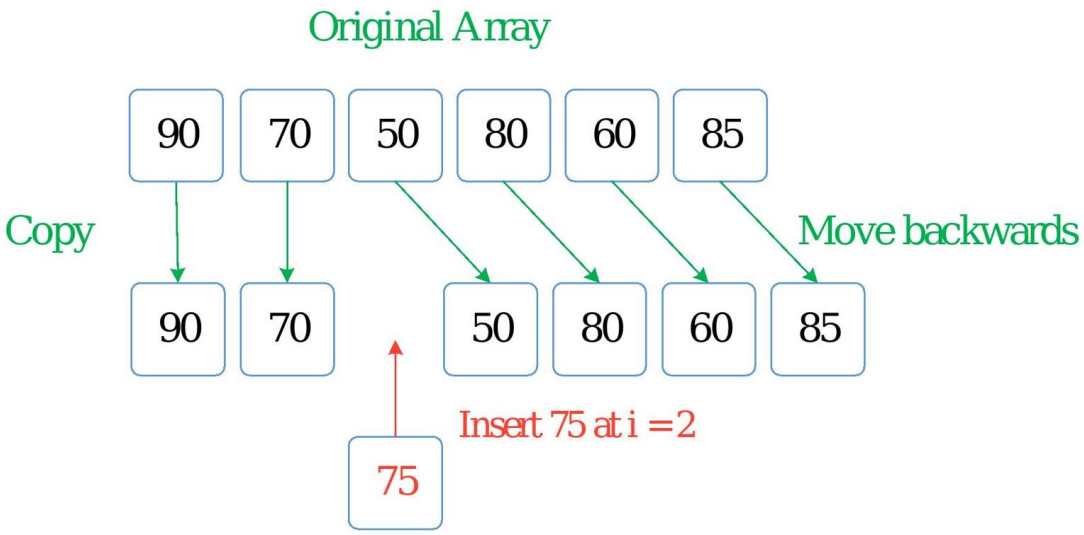
1. Crie um TestOneArrayAppend.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   function append(array, value) {
   var tempArray = new Array(array.length + 1); // Crie um tempArray maior que array
   for (var i = 0; i < array.length; i++) {
   tempArray[i] = array[i]; // Copie o valor da matriz para tempArray }
   tempArray[array.length] = value// Insira o valor para o último índice
return tempArray; }

   //////////////////////testing//////////////////////
   var scores = new Array( 90, 70, 50, 80, 60, 85 );
   scores = append(scores, 75);
   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }
</script>
```

Resultado: 90, 70, 50, 80, 60, 85, 75,

Inserção de mesa linear

1. **Inserir 75 na matriz unidimensional.**



Análise: 1. Primeiro, crie uma matriz temporária tempArray maior que o comprimento original da matriz 2. Do início à posição de inserção Copie cada valor do valor anterior da matriz para tempArray 3. Mova a matriz da posição de inserção para cada valor do último elemento e mova-o de volta para tempArray 4. insira 75 no tempArray.
5. Finalmente, atribua a referência do ponteiro tempArray à matriz original;

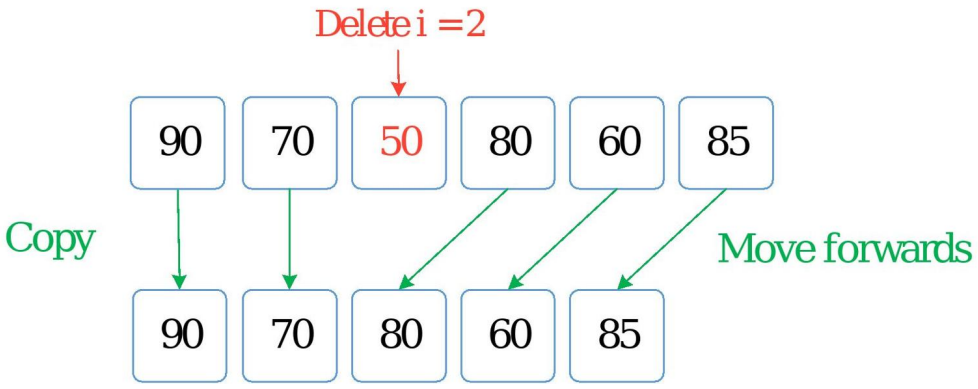
1. Crie um TestOneArrayInsert.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   function insert(array, score, insertIndex) {
   var tempArray = new Array(array.length + 1); for (var i = 0; i <
array.length; i++) {
   if (i < insertIndex) {
   tempArray[i] = array[i]; //  Copie o valor anterior a i = insertIndex para
tempArray } else {
   tempArray[i + 1] = array[i]; //  Copie os elementos restantes para
tempArray }
   }
   tempArray[insertIndex] = score; return tempArray; }

   ////////////////////testing//////////////////
   var scores = new Array( 90, 70, 50, 80, 60, 85 );
   scores = insert(scores, 75, 2); // Insira 75 na posição: index = 2

   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }

</script>
```

Resultado:
90,70,75,50, 80, 60, 85,

Tabela linear Excluir

1. **Exclua o valor do índice = 2 da matriz**



Análise: 1. Crie uma matriz temporária tempArray com tamanho menor que a matriz original.

2. Copie os dados na frente de i = 2 para a frente do tempArray 3. Copie a matriz após i = 2 para tempArray 4. Atribua a referência do ponteiro tempArray à matriz original

1. Crie um TestOneArrayDelete.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   function remove(array, index){
   // crie uma nova matriz, length = scores.length - 1
   var tempArray = new Array(array.length - 1);
   for (var i = 0; i < array.length; i++) {
   if (i < index) // Copie os dados anteriores a i = index para tempArray.
   tempArray[i] = array[i]; if (i > index) // Copie a matriz após i = index
para o final de tempArray tempArray[i - 1] = array[i]; }
   return tempArray; }

   //////////////////////testing//////////////////////
   var scores = new Array( 90, 70, 50, 80, 60, 85 );
   scores = remove(scores, 2); // remova o valor da matriz no índice = 2

   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }

</script>
```

Resultado:

90,70,80,60,85,

Algoritmo de Ordenar de bolhas

Algoritmo de Ordenar de bolhas (Bubble Sorting Algorithm): se arrays [j]> arrays [j + 1] forem trocados. Os elementos restantes repetem esse processo até a Ordenar ser concluída.

Ordenar os seguintes dados de pequeno a grande

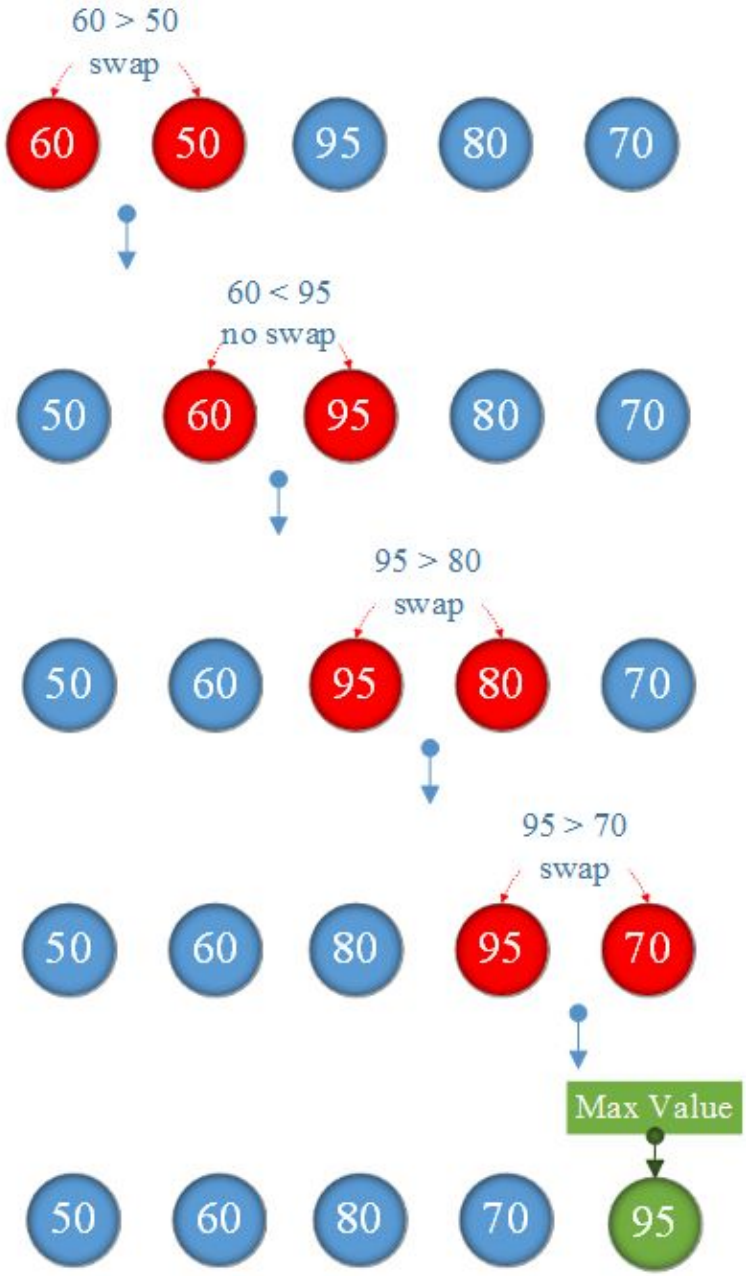
Explicação: permutar(swap), Valor máximo(max value)


Sem Ordenar,

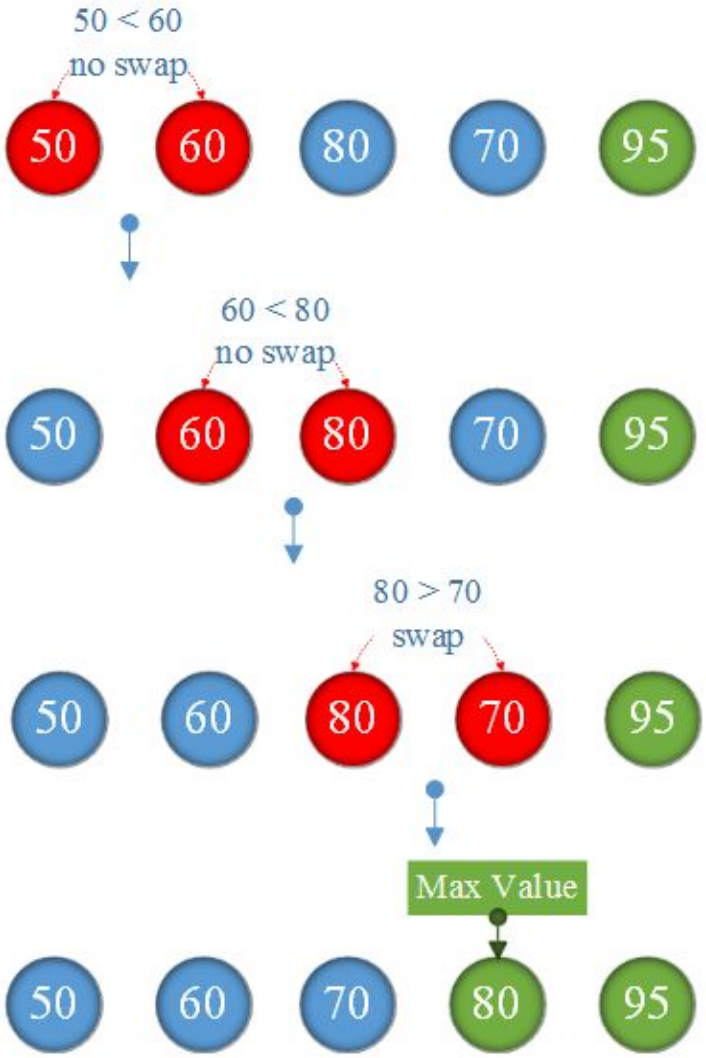

Comparando,


Já Ordenar

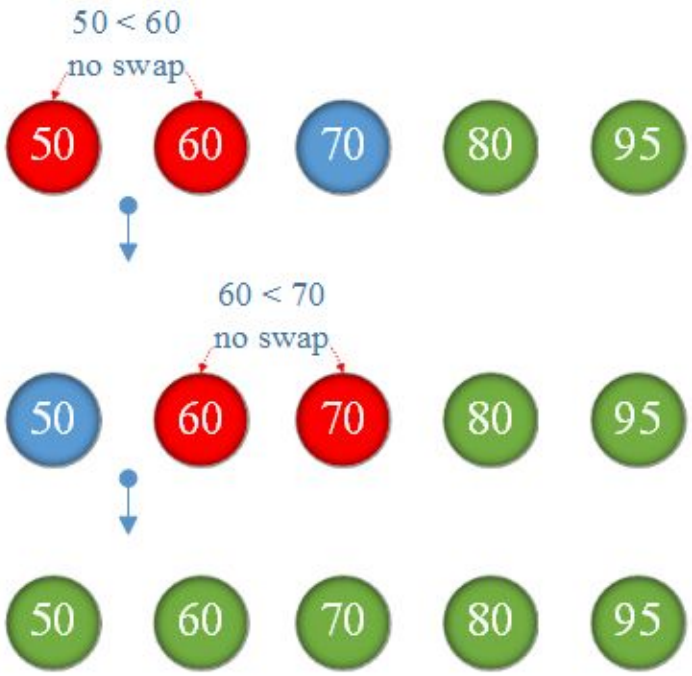
1. **Primeira Ordenar:**



2. Segunda Ordenar:



3. Terceira Ordenar:



Nenhuma troca, então encerre a Ordenar :

1. Crie um TestBubbleSort.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   function sort(arrays) {
   for (var i = 0; i < arrays.length - 1; i++) {
   var isSwap = false; for (var j = 0; j < arrays.length - i - 1; j++) {
   // troca if (arrays[j] > arrays[j + 1]) {
   var flag = arrays[j]; arrays[j] = arrays[j + 1]; arrays[j + 1] = flag;
isSwap = true; }
   }

   if(!isSwap) // Sem troca, pare de classificar {
   break; }
   }
   }

   ////////////////////// test //////////////////
   var scores = [ 90, 70, 50, 80, 60, 85 ];
   sort(scores);
   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }
</script>
```

Resultado:

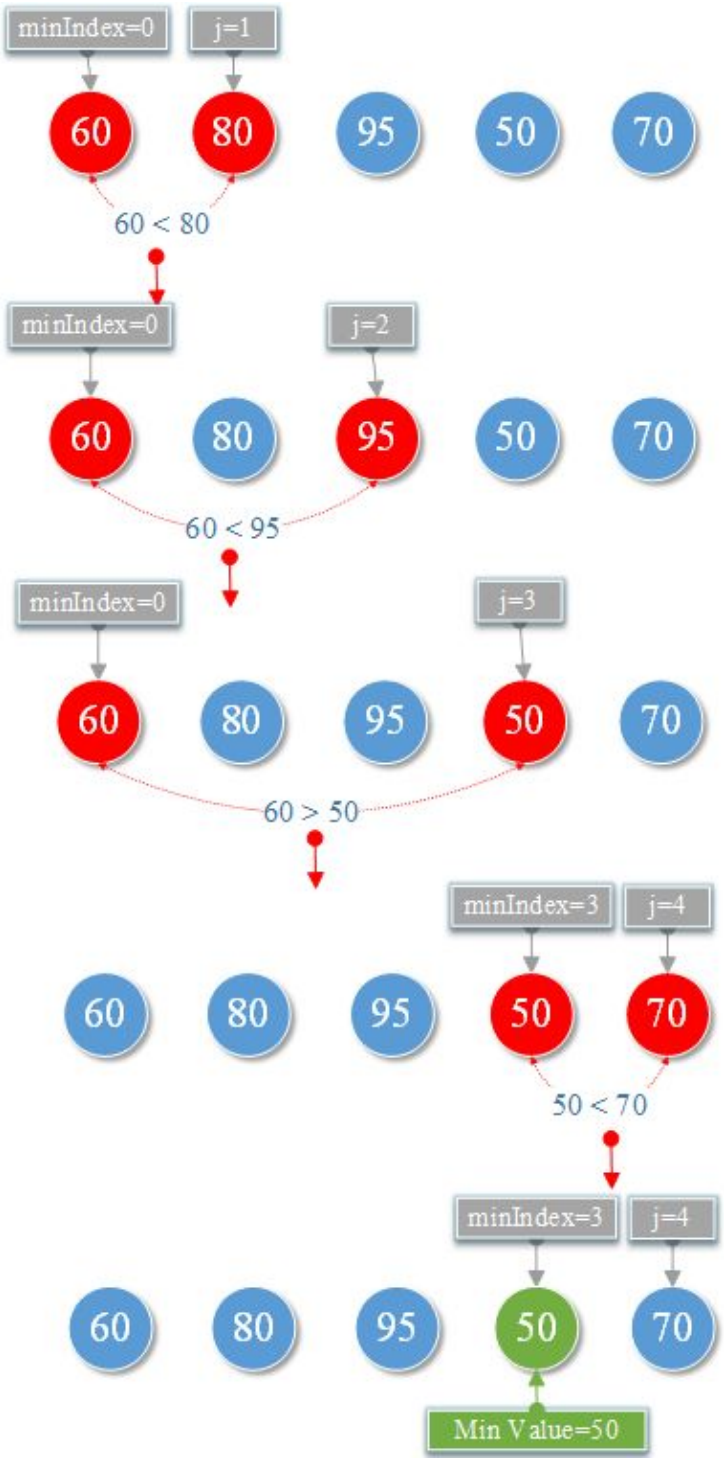
50,60,70,80,95,

Valor Mínimo

Pesquisar o mínimo de sequências inteiras:

1. Ideias algorítmicas

Valor inicial minIndex=0, j=1 Compare arrays[minIndex] com arrays[j] if arrays[minIndex] > arrays[j] then minIndex=j, j++ else j++. continue até o último número, arrays[minIndex] é o valor mínimo.


minIndex=0
j=1
60
80
95
50
70
60 < 80
minIndex=0
j=2
60
80
95
50
70
60 < 95
minIndex=0
j=3
60
80
95
50
70
60 > 50
minIndex=3
j=4
60
80
95
50
70
50 < 70
minIndex=3
j=4
60
80
95
50
70
Min Value=50

1. Crie um TestMinValue.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript"> function min(arrays) {
   var minIndex = 0; for (var j = 1; j < arrays.length; j++) {
   if (arrays[minIndex] > arrays[j]) {
   minIndex = j; }
   }
   return arrays[minIndex]; }

   ////////////////////testing//////////////////

   var scores = [ 60, 80, 95, 50, 70 ];
   var minValue = min(scores);
   document.write("Min Value = " + minValue);
</script>
```

Resultado:
Min Value = 50

Selecione o algoritmo de Ordenar

Selecione o algoritmo de Ordenar (Select Sorting Algorithm):
Classifica uma matriz localizando repetidamente o elemento mínimo de uma parte não classificada e colocando-a no início.

Ordenar os seguintes dados de pequeno a grande

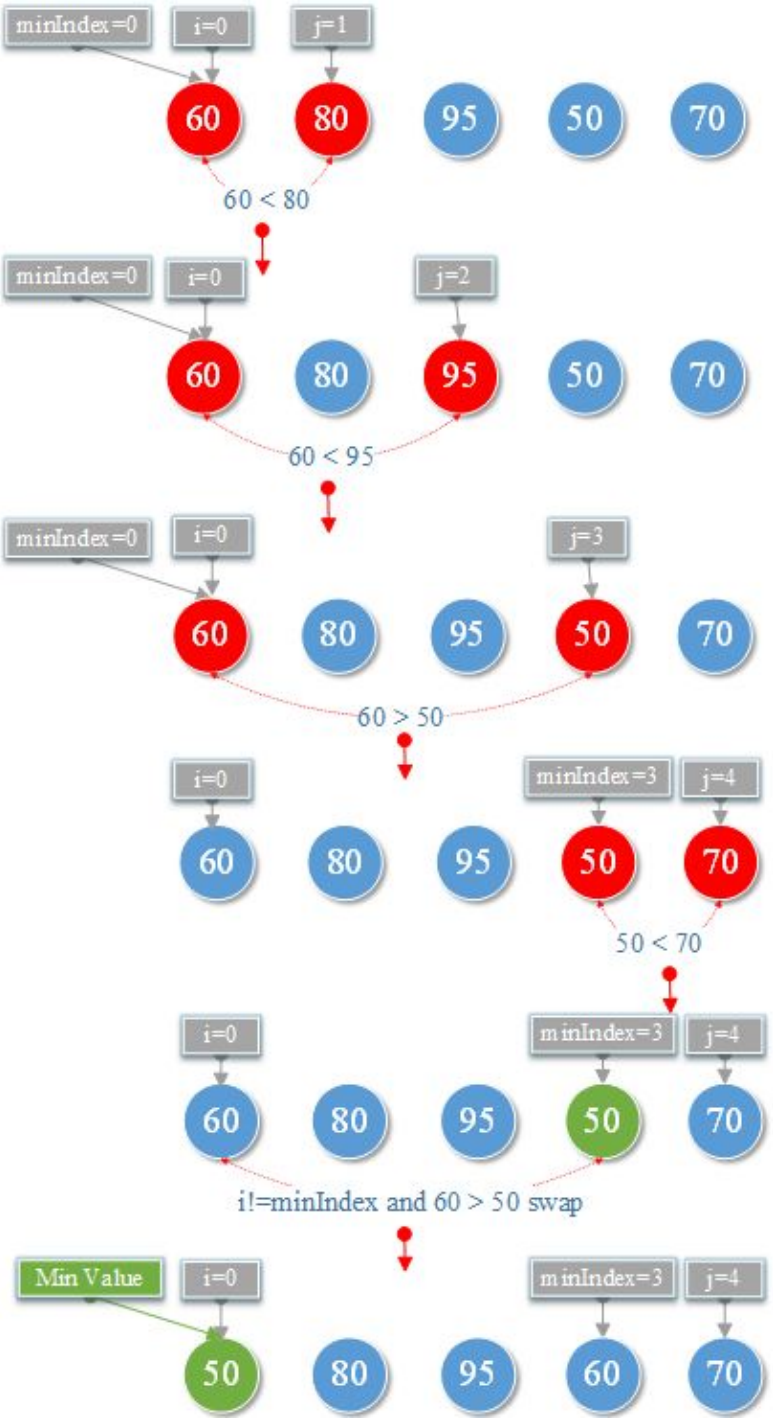
Explicação: permutar(swap), Valor mínimo(min value)


Sem Ordenar,

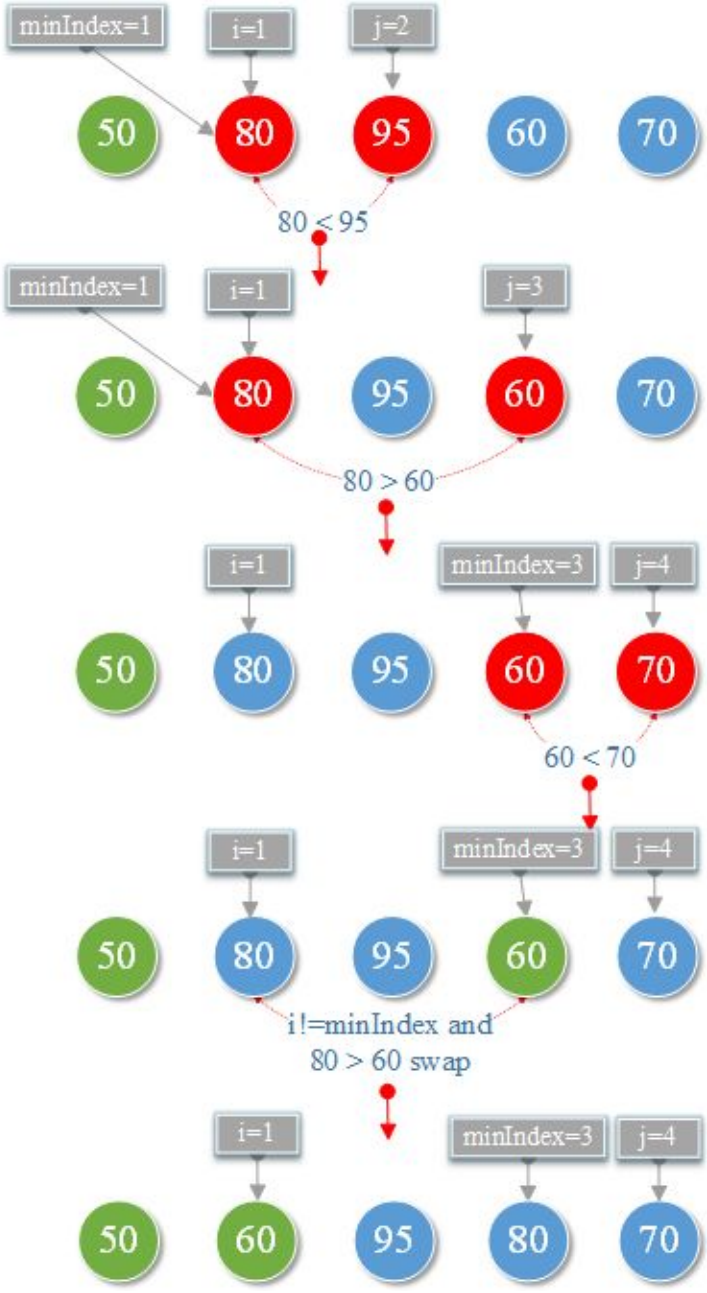

Comparando,


Já Ordenar

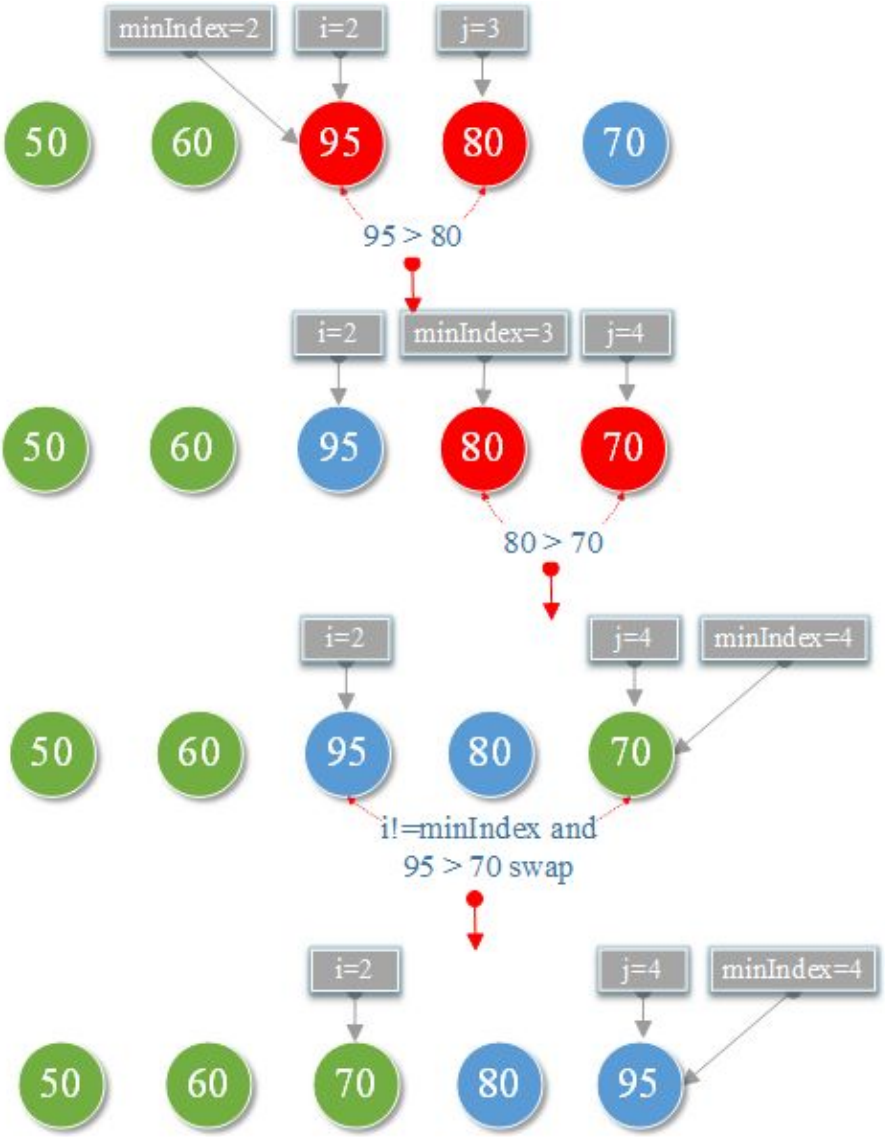
1. **Primeira Ordenar:**



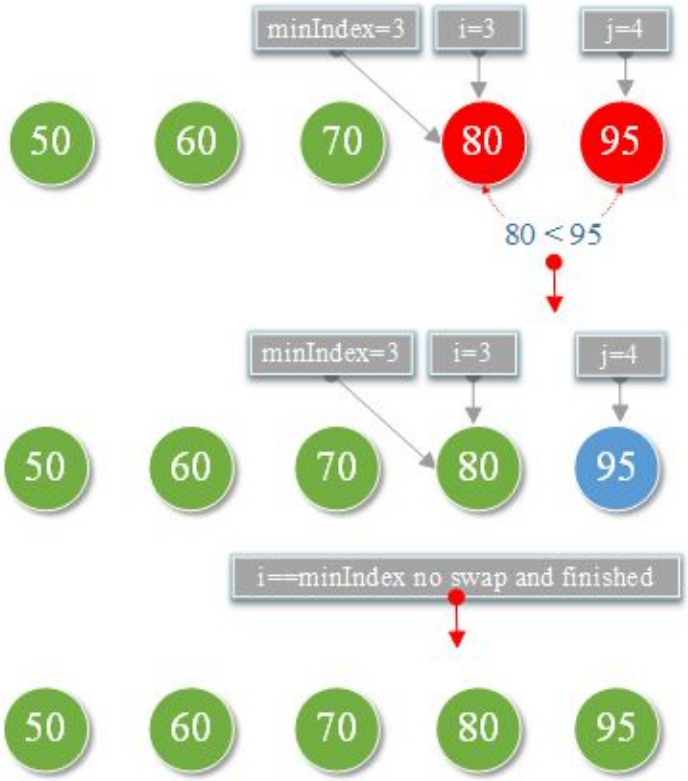
2. Segunda Ordenar:



3. Terceira Ordenar:



4. Quarta Ordenar:



Nenhuma troca, então encerre a Ordenar

1. Crie um TestSelectSort.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   class SelectSort{
   static sort(arrays) {
   var len = arrays.length - 1; var minIndex;// O índice do mínimo
selecionado
   for (var i = 0; i < len; i++) {
   minIndex = i; var minValue = arrays[minIndex]; for (var j = i; j < len;
j++) {
   if (minValue > arrays[j + 1]) {
   minValue = arrays[j + 1]; minIndex = j + 1; }
   }

   // O mínimo de arrays[i] é trocado pelos arrays[minIndex]
   if (i != minIndex){
   var temp = arrays[i]; arrays[i] = arrays[minIndex]; arrays[minIndex] =
temp; }
   }
   }
   }

   //////////////////////testing//////////////////
   var scores = [ 90, 70, 50, 80, 60, 85 ];
   SelectSort.sort(scores);
   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }
</script>
```

Resultado:

50,60,70,80,95,

Inserir algoritmo de Ordenar

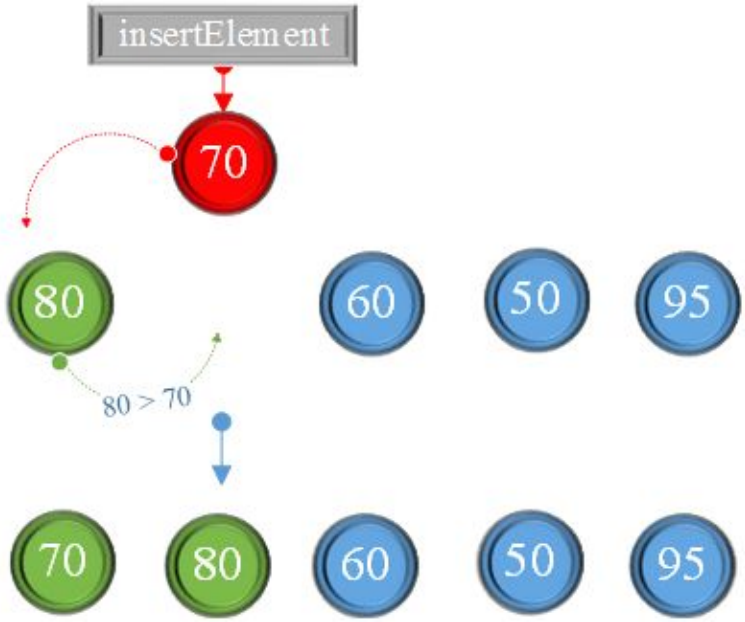
Inserir algoritmo de Ordenar (Insert Sorting Algorithm): Pegue um novo elemento não classificado na matriz, compare-o com o elemento já classificado antes, se o elemento for menor que o elemento classificado, insira um novo elemento na posição correta.

Ordenar os seguintes dados de pequeno a grande

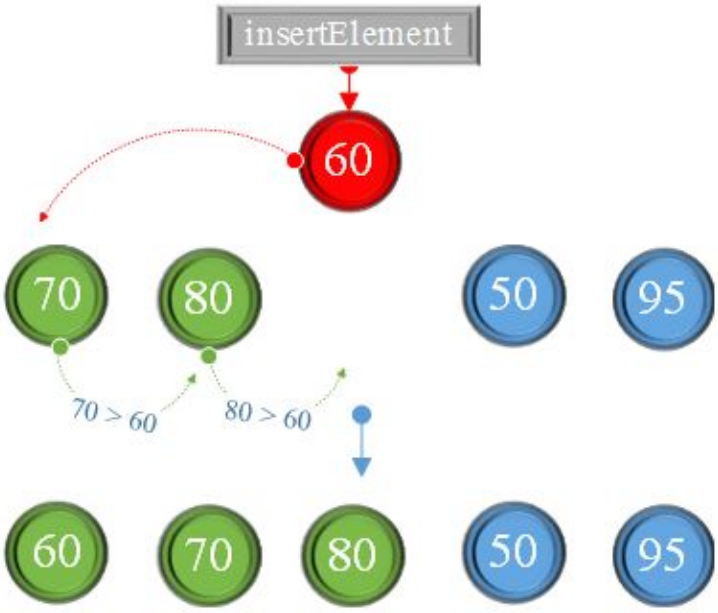
Explicação:

 Sem Ordenar, Comparando, Já Ordenar

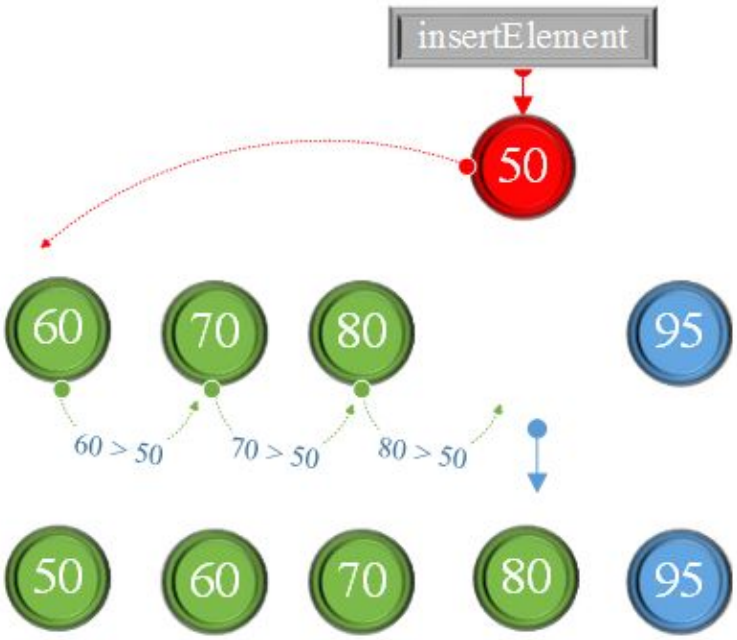
1. **Primeira Ordenar:**



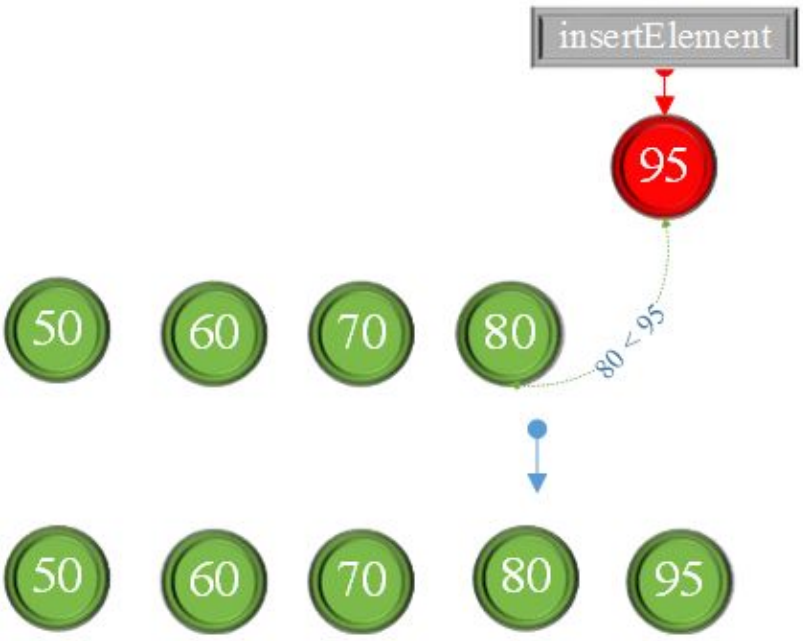
2. **Segunda Ordenar:**



3. Terceira Ordenar:



4 Quarta Ordenar:



1. Crie um TestInsertSort.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   class InsertSort{

   static sort(arrays) {
   for (var i = 0; i < arrays.length; i++) {
   var insertElement = arrays[i];//  Pegue novos elementos não
classificados var insertPosition = i; for (var j = insertPosition - 1; j >= 0;
j--) {
   // insertElement é deslocado para a direita if (insertElement < arrays[j])
{
   arrays[j + 1] = arrays[j]; insertPosition--; }
   }
   arrays[insertPosition] = insertElement; }
   }
   }

   ////////////////////testing//////////////////

   var scores = [ 90, 70, 50, 80, 60, 85 ];
   InsertSort.sort(scores);
   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }
</script>
```

Resultado:

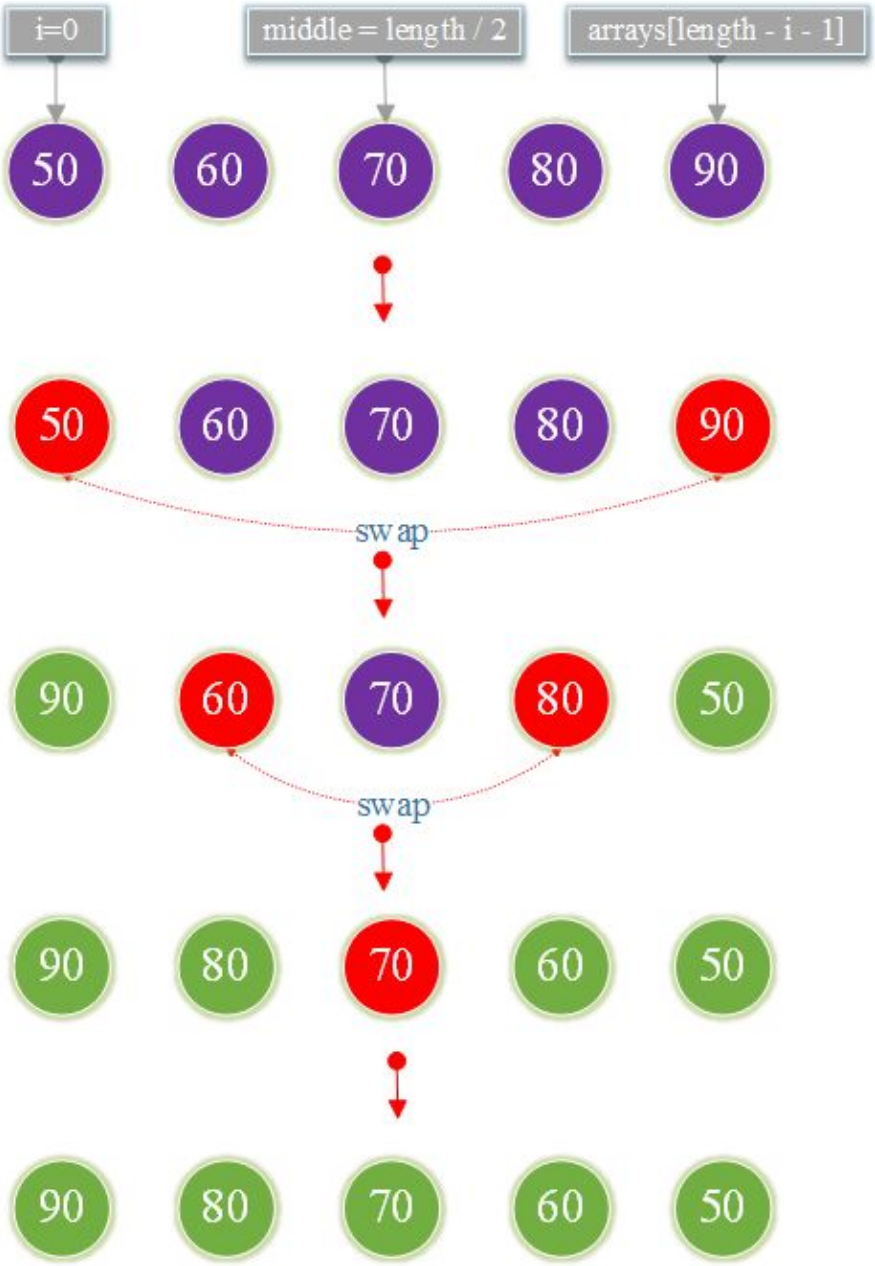
50,60,70,80,95,

Mesa Linear Reversa

Inversão de sequências ordenadas:

1. Ideias algorítmicas

Inicial i = 0 e, em seguida, troque o primeiro elemento pelo último elemento
Repita até o índice do meio i == length / 2.


i=0
middle = length / 2
arrays[length - i - 1]
50
60
70
80
90
50
60
70
80
90
swap
90
60
70
80
50
swap
90
80
70
60
50
90
80
70
60
50

1. Crie um TestReverse.html com o Bloco de notas e abra-o em seu navegador.

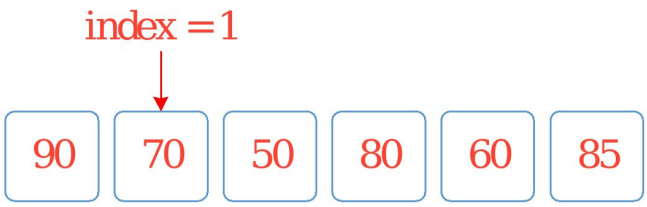
```
<script type="text/javascript"> function reverse(arrays) {
   var length = arrays.length; var middle = length / 2; for (var i = 0; i <=
middle; i++) {
   var temp = arrays[i]; arrays[i] = arrays[length - i - 1]; arrays[length - i -
1] = temp; }
   }

   ////////////////////testing//////////////////

   var scores = [ 50, 60, 70, 80, 90 ];
   reverse(scores);
   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }

</script>
```

Resultado:

90,80,70,60,50,

Pesquisa de tabela linear

1. Digite 70 que deseja pesquisar e depois retorne o índice.



1. Crie um TestOneArraySearch.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   function search(array, value){
   for (var i = 0; i < array.length; i++) {
   if (array[i] == value) {
   return i; }
   }
   return -1; }

   ////////////////////testing////////////////////
   var scores = new Array( 90, 70, 50, 80, 60, 85 ); var value = 70; var
index = search(scores, value);
   if (index > 0) {
   document.write("Found value: " + value + " the index is: " + index);
}else{
   document.write("The value was not found : " + value); }
</script>
```

Resultado:

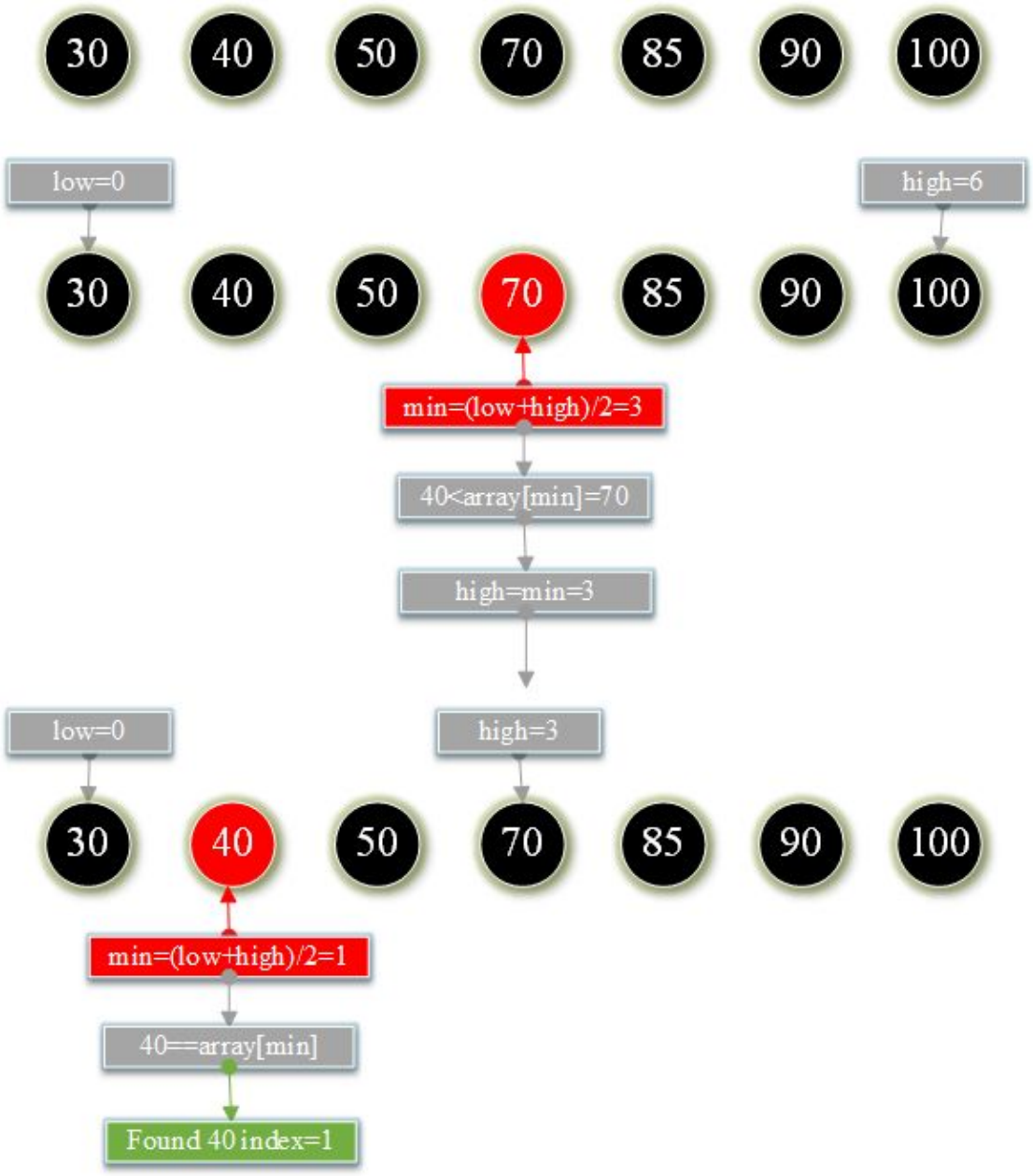
Found value: 70 the index is: 1

Algoritmo de pesquisa de dicotomia

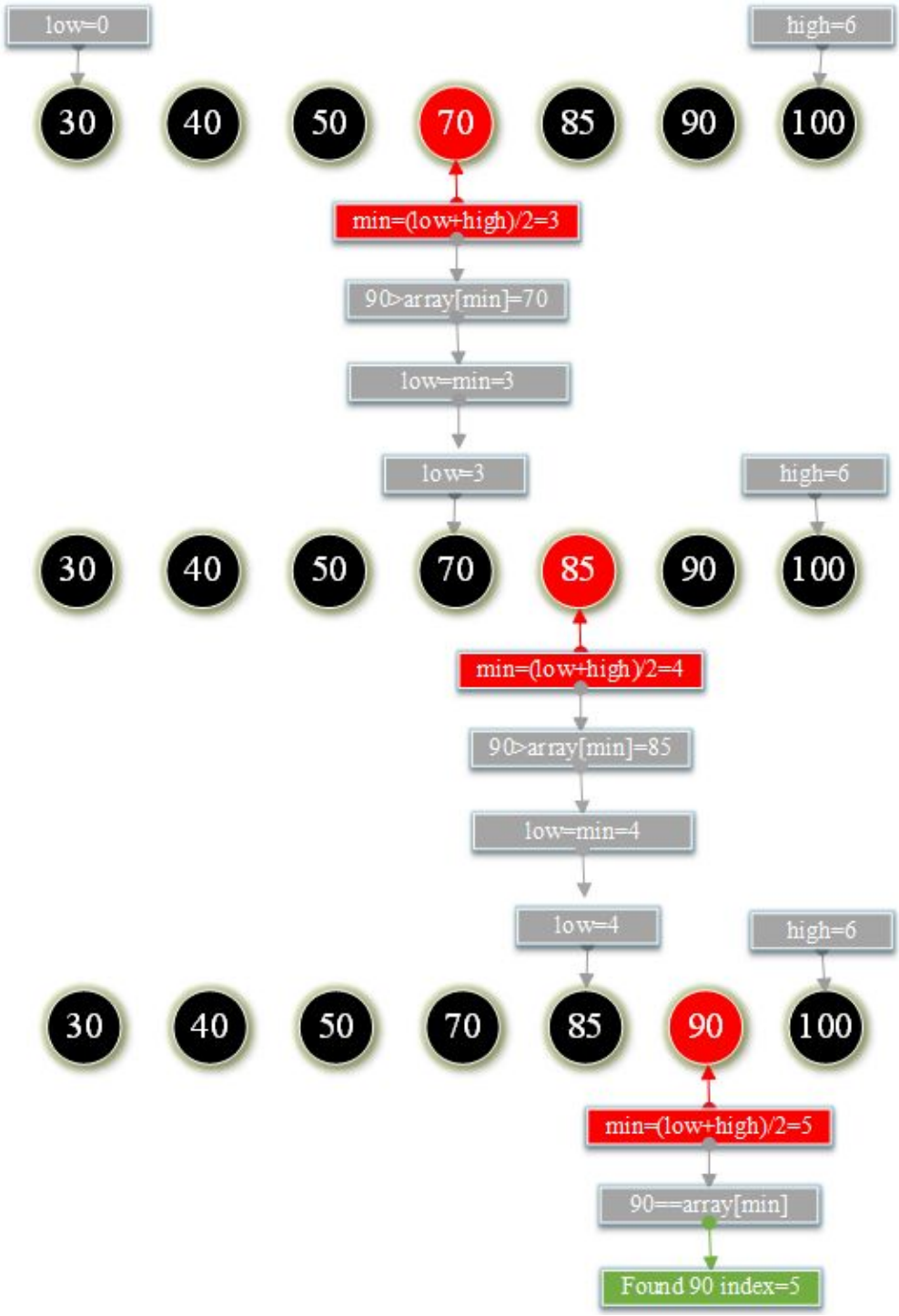
Algoritmo de pesquisa de dicotomia (Dichotomy Search Algorithm): O algoritmo de pesquisa binária, também conhecido como algoritmo de pesquisa binária e algoritmo de pesquisa logarítmica, é um algoritmo de pesquisa para encontrar um elemento específico em uma matriz ordenada. O processo de pesquisa inicia no elemento do meio da matriz. Se o elemento do meio for o elemento a ser encontrado, o processo de pesquisa será encerrado; se um elemento em particular for maior que ou menor que o elemento do meio, ele será encontrado na metade da matriz que é maior ou menor que o elemento do meio, e Comece com o elemento do meio como antes..

1. Inicialize o índice mais baixo low=0, o índice mais alto high=scores.length-1
2. Encontre o searchValue do índice do meio mid=(low+high)/2 scores[mid]
3. Compare a scores[mid] com searchValue E se scores[mid]==searchValue impressão mid index, E se scores[mid]>searchValue que o searchValue será encontrado entre low com mid-1
4. Repita a etapa 3 até encontrar searchValue ou low> = high para finalizar o loop.

Exemplo 1 : Encontre o índice de searchValue = 40 na matriz que foi classificada abaixo.



Exemplo 2 : Encontre o índice de searchValue = 90 na matriz que foi classificada abaixo.



1. Crie um TestBinarySearch.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript"> class BinarySearch{
   static search(arrays, searchValue) {
   var low = 0; //índice mais baixo var high = arrays.length - 1; //índice
mais alto var mid = 0; //índice do meio
   while (low <= high) {
   mid = (low + high) / 2; if (arrays[mid] == searchValue) {
   return mid; } else if (arrays[mid] < searchValue) {
   low = mid + 1; //  continue a encontrar entre mid+1 e high } else if
(arrays[mid] > searchValue) {
   high = mid - 1; //  continue a encontrar entre low e mid-1
   }
   }
   return -1; }
   }

   //////////////////////testing//////////////////////
   var scores = [ 30, 40, 50, 70, 85, 90, 100 ];
   var searchValue = 40; var position = BinarySearch.search(scores,
searchValue); document.write(searchValue + " position:" + position);
   document.write("<br>--------------------------<br>");
   searchValue = 90; position = BinarySearch.search(scores,
searchValue); document.write(searchValue + " position:" + position);
</script>
```

Resultado:

40 position:1

----

90 position:5

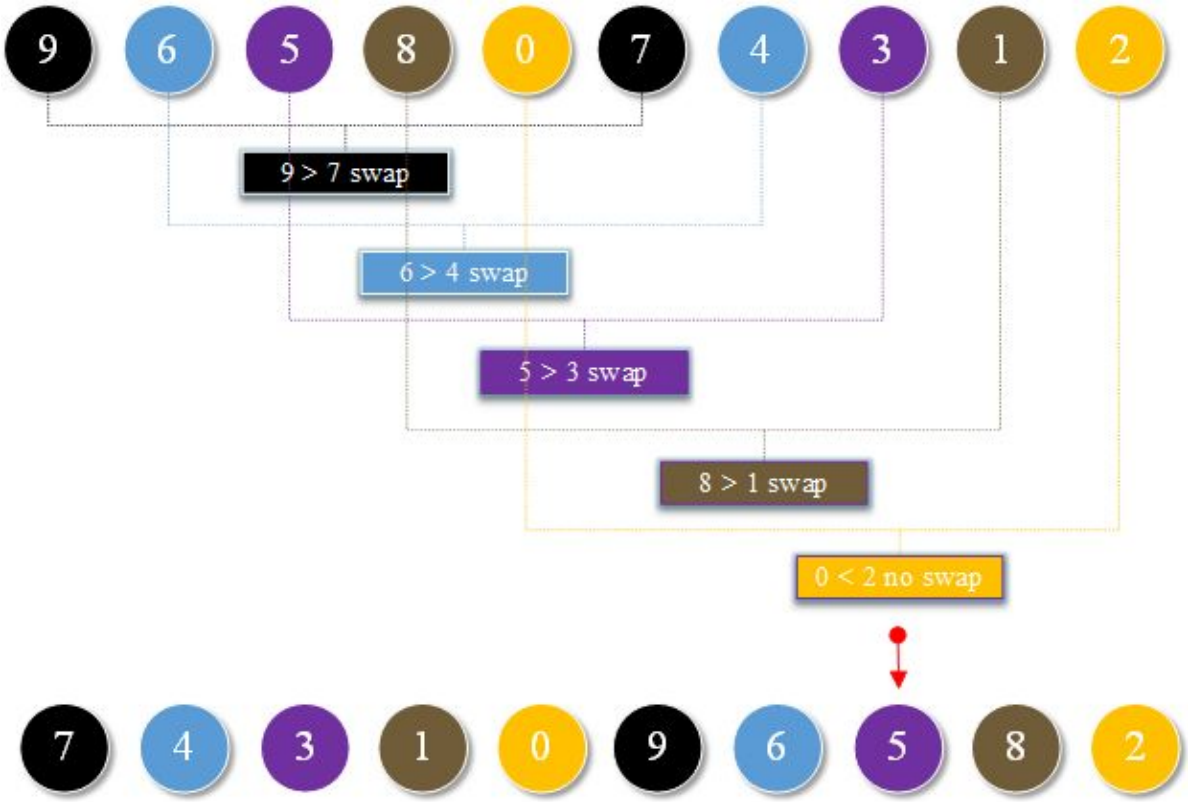
Classificação de cascas

Classificação de cascas (Shell Sorting):
A classificação shell é um algoritmo de classificação altamente eficiente e é baseado no algoritmo de classificação por inserção. Este algoritmo evita grandes deslocamentos como no caso da classificação por inserção, se o valor menor estiver na extrema direita e tiver que ser movido na extrema esquerda.

Classifique os seguintes números, de pequeno a grande, por Classificação Shell

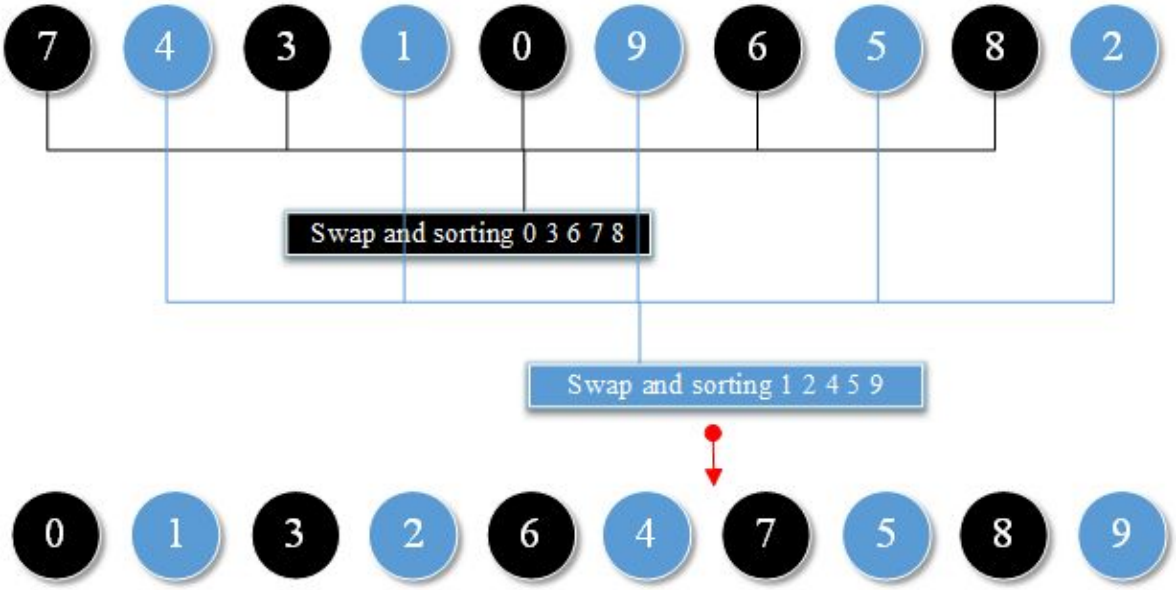
Resultado do algoritmo:
O array é agrupado de acordo com um certo incremento de subscritos, e a inserção de cada grupo é ordenada. Conforme o incremento diminui gradualmente até que o incremento seja 1, todos os dados são agrupados e classificados.

1. **A primeira classificação :**
   gap = array.length / 2 = 5


9 6 5 8 0 7 4 3 1 2
9 > 7 swap
6 > 4 swap
5 > 3 swap
8 > 1 swap
0 < 2 no swap
7 4 3 1 0 9 6 5 8 2

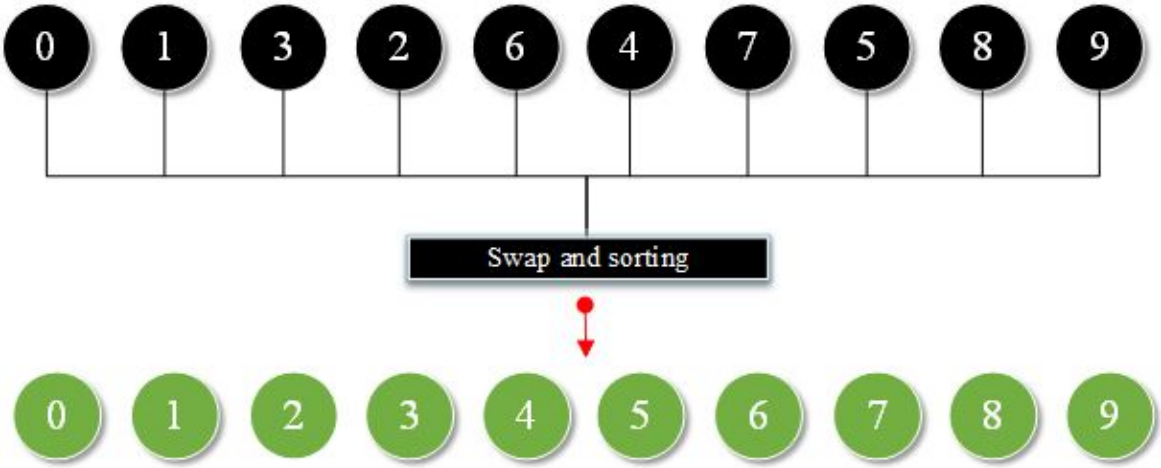
2. A segunda classificação :

gap = 5 / 2 = 2



3. A terceira classificação :

gap = 2 / 2 = 1



1. Crie um TestShellSort.html com o Bloco de notas e abra-o em seu navegador.

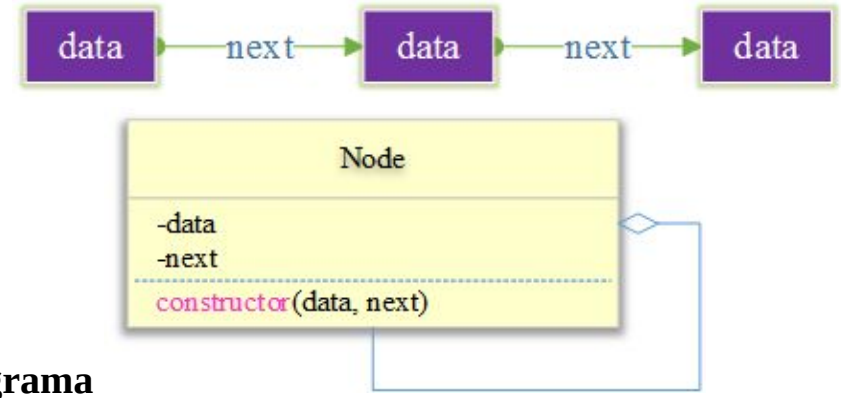
```
<script type="text/javascript">
   function shellSort(array) {
   var middle = parseInt(array.length / 2);
   for (var gap =middle ; gap > 0; gap = parseInt(gap / 2)) {
   for (var i = gap; i < array.length; i++) {
   var j = i;
   while (j - gap >= 0 && array[j] < array[j - gap]) {
   swap(array, j, j - gap);
   j = j - gap;
   }
   }
   }
   }

   function swap(array, a, b) {
   array[a] = array[a] + array[b];
   array[b] = array[a] - array[b];
   array[a] = array[a] - array[b];
   }

   ////////////////////testing//////////////////

   var scores = [ 9, 6, 5, 8, 0, 7, 4, 3, 1, 2 ];

   shellSort(scores);

   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ",");
   }

</script>
```

Resultado:

0,1,2,3,4,5,6,7,8,9,

Lista vinculada única

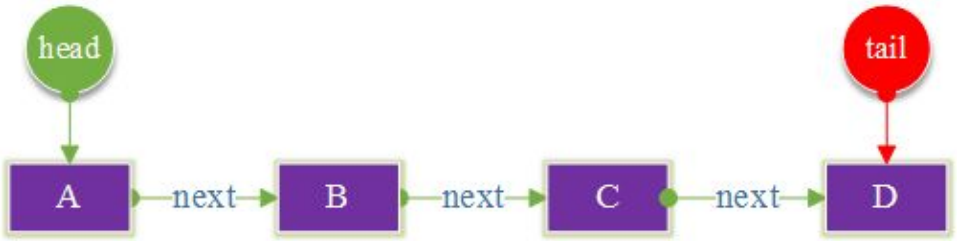
Lista vinculada única (Single Linked List): É uma estrutura de armazenamento encadeado de uma tabela linear, que é conectada por um nó. Cada nó consiste em dados e ponteiro para o próximo nó.



UML Diagrama

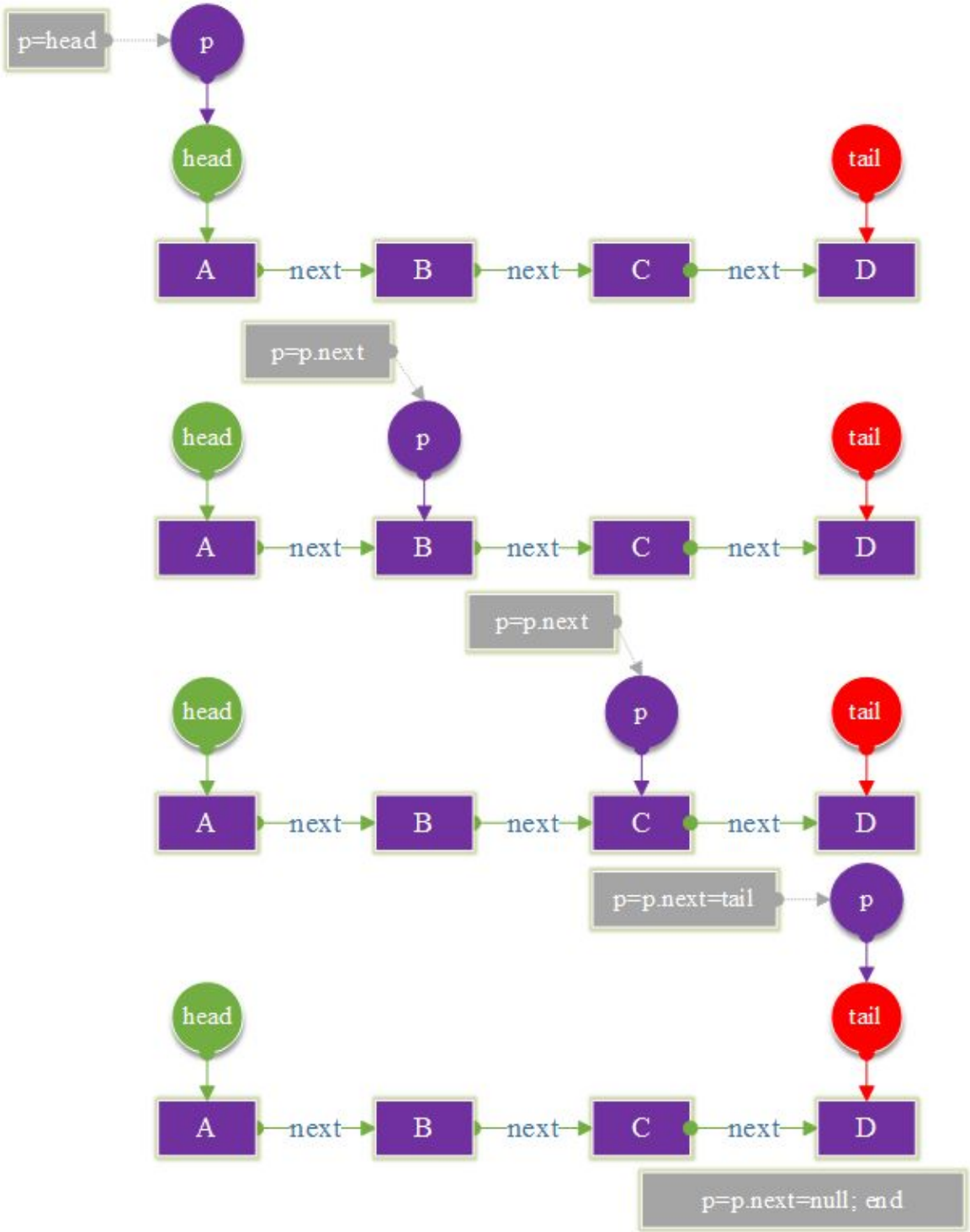
```
class Node{
constructor(data, next){
this.data = data; this.next = next; }

getData(){
return this.data; }
}
```

1. Inicialização de lista vinculada única .
Exemplo: Construir uma lista vinculada única




```
class LinkedList{
init() {
// o primeiro nó chamado cabe?a nó this.head = new Node("A", null);
var nodeB = new Node("B", null); this.head.next = nodeB; var nodeC =
new Node("C", null); nodeB.next = nodeC; // o último nó chamado nó da
cauda this.tail = new Node("D", null); nodeC.next = this.tail; }
}
```

2. Saída de lista vinculada única .



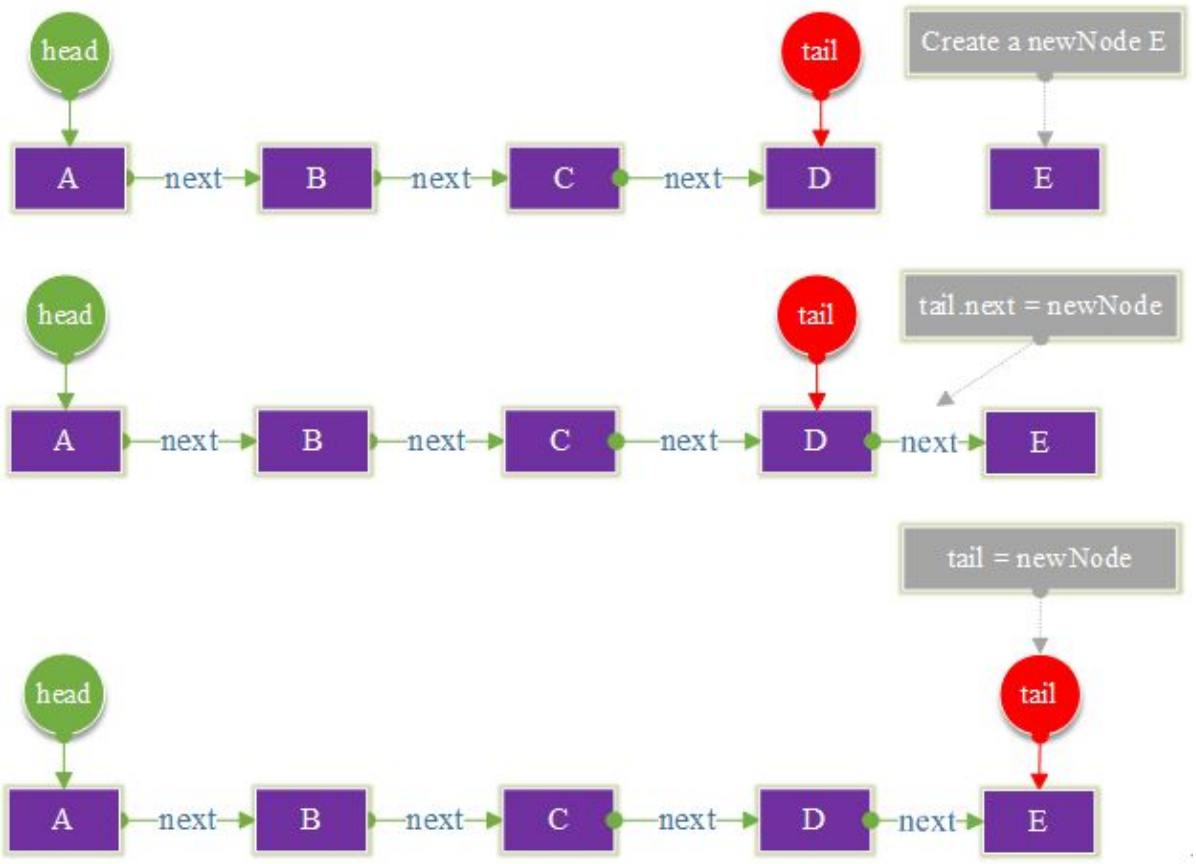
1. Crie um TestSingleLinkList.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   /////////////// Node ////////////////
   class Node{
   constructor(data, next){
   this.data = data; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// LinkedList ////////////////
   class LinkedList{

   init() {
   // o primeiro nó chamado cabe?a nó this.head = new Node("A", null);
   var nodeB = new Node("B", null); this.head.next = nodeB;
   var nodeC = new Node("C", null); nodeB.next = nodeC;
   // o último nó chamado nó da cauda this.tail = new Node("D", null);
nodeC.next = this.tail; }

   getHead(){
   return this.head; }

   }




////////////////// test ////////////////////

   function print(node) {
   var p = node; while (p != null) // Imprimir do início ao fim {
   var data = p.getData(); document.write(data + " -> "); p = p.next; }
```

```
    document.write("End<br><br>"); }

    var linkedList = new LinkedList();
    linkedList.init();
    print(linkedList.getHead());
</script>
```

Resultado: A -> B -> C -> D -> End

3. Anexe um novo nome de nó: E




```
add(newNode) {
    this.tail.next = newNode;
    this.tail = newNode;
}
```

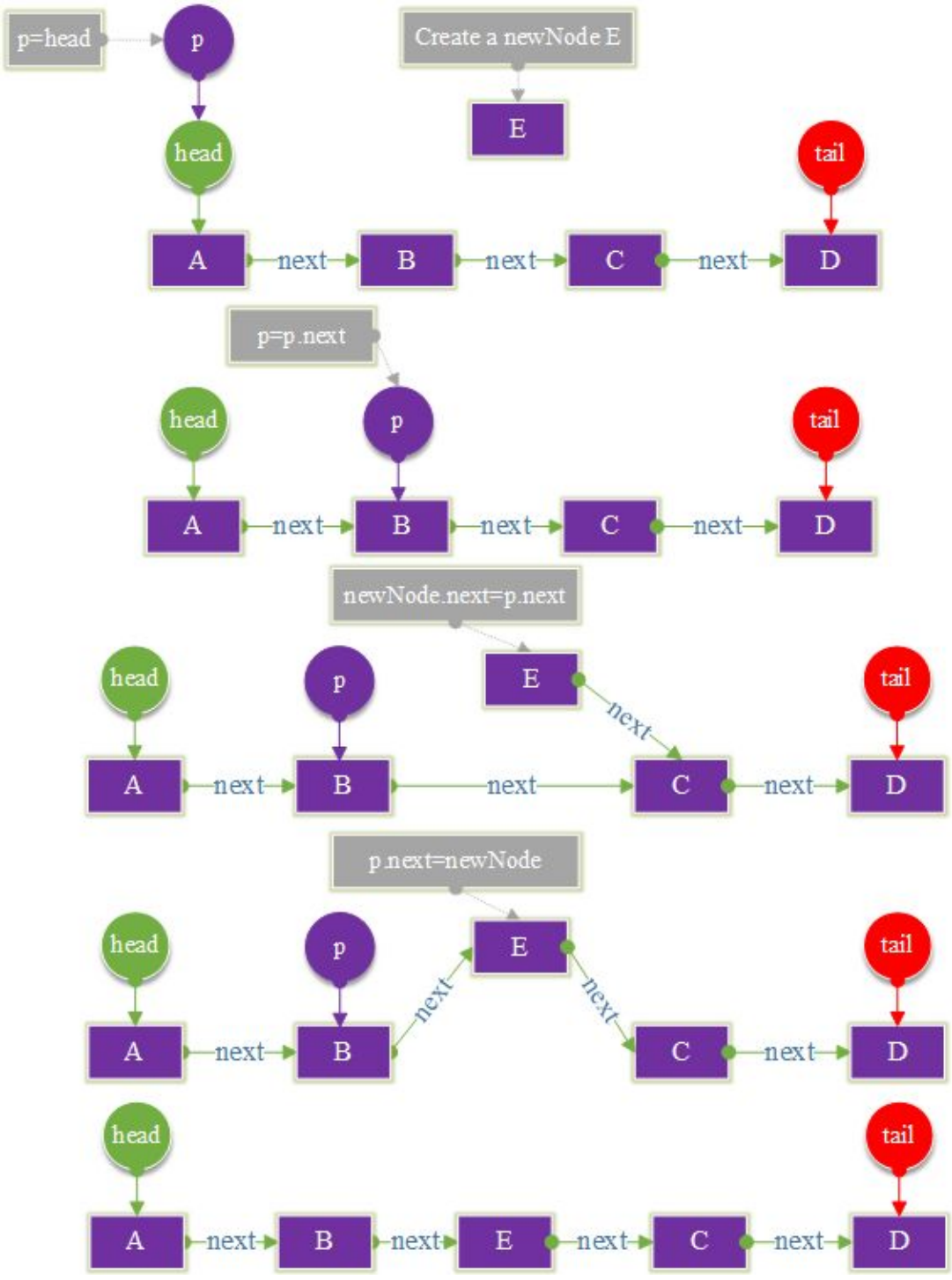
1. **Crie um arquivo: TestSingleLinkList.html**

```
<script type="text/javascript">
   /////////////// Node /////////////////
   class Node{
   constructor(data, next){
   this.data = data; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// LinkedList /////////////////
   class LinkedList{

   init() {
   // o primeiro nó chamado cabe?a nó this.head = new Node("A", null);
   var nodeB = new Node("B", null); this.head.next = nodeB;
   var nodeC = new Node("C", null); nodeB.next = nodeC;
   // o último nó chamado nó da cauda this.tail = new Node("D", null);
nodeC.next = this.tail; }

   add(newNode) {
   this.tail.next = newNode; this.tail = newNode; }

   getHead(){
   return this.head; }
   }


////////////////// test ////////////////////

   function print(node) {
   var p = node; while (p != null) // Imprimir do início ao fim {
   var data = p.getData(); document.write(data + " -> "); p = p.next; }
   document.write("End<br><br>"); }
```

```
    var linkedList = new LinkedList();
    linkedList.init();
    linkedList.add(new Node("E", null)); //  Anexe um novo nó: E

    print(linkedList.getHead());
</script>
```

Resultado: A -> B -> C -> D -> E -> End

3. Insira um nó E na posição 2.



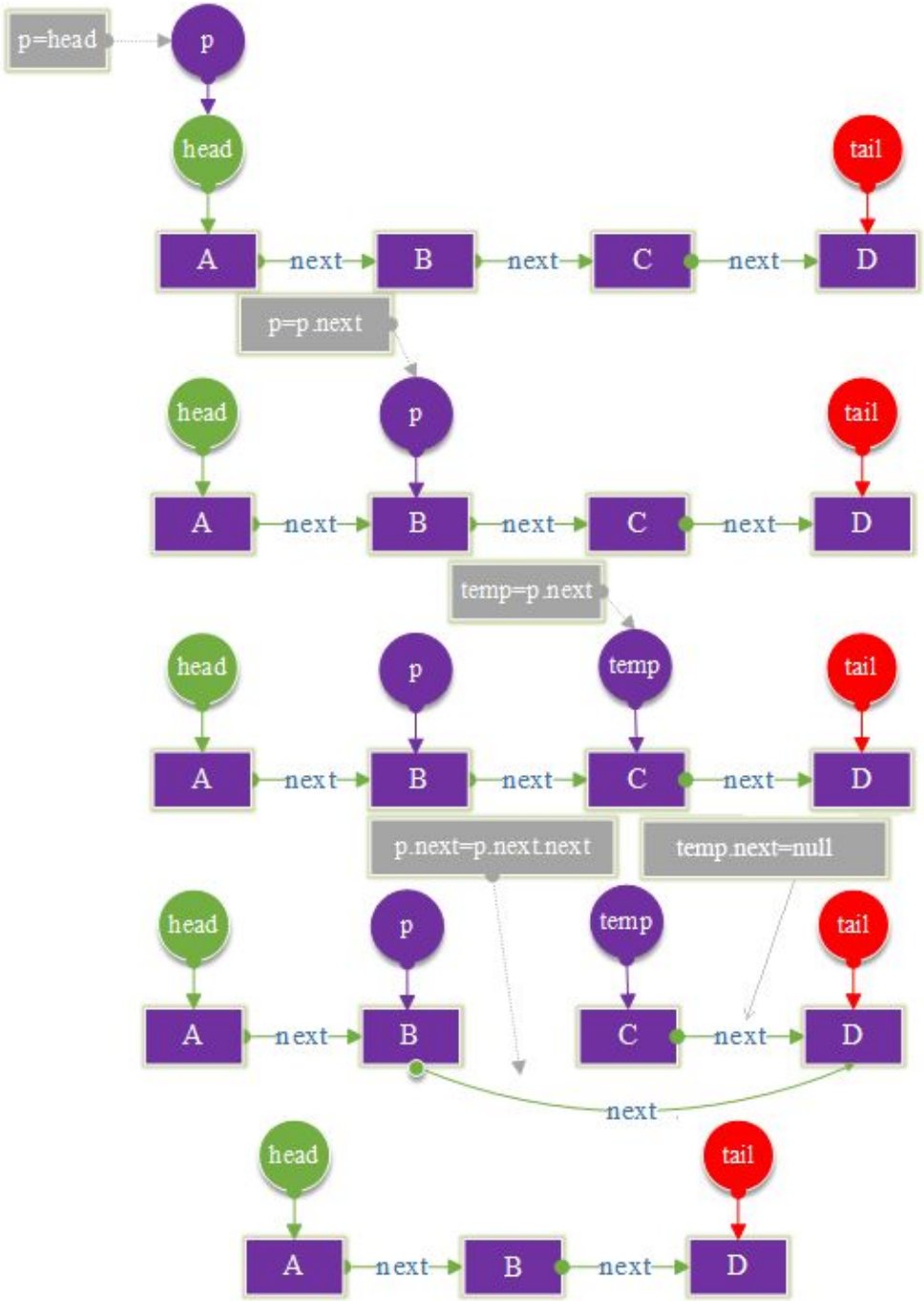
1. Crie um arquivo: TestSingleLinkList.html

```
<script type="text/javascript"> /////////////// Node ////////////////
   class Node{
   constructor(data, next){
   this.data = data; this.next = next; }

   getData(){
   return this.data; }
   }

   ////////////// LinkedList ////////////////
   class LinkedList{

   init() {
   // o primeiro nó chamado cabe?a nó this.head = new Node("A", null);
   var nodeB = new Node("B", null); this.head.next = nodeB;
   var nodeC = new Node("C", null); nodeB.next = nodeC;
   // o último nó chamado nó da cauda this.tail = new Node("D", null);
nodeC.next = this.tail; }

   insert(insertPosition, newNode) {
   var p = this.head; var i = 0; // Mova o nó para a posição de inserção
while (p.next != null && i < insertPosition - 1) {
   p = p.next; i++; }

   newNode.next = p.next; p.next = newNode; }

   getHead(){
   return this.head; }
   }

////////////////// test ////////////////////

   function print(node) {
   var p = node; while (p != null) // Imprimir do início ao fim {
   var data = p.getData(); document.write(data + " -> "); p = p.next; }
   document.write("End<br><br>"); }
```

```
    var linkedList = new LinkedList();
    linkedList.init();
    linkedList.insert(2, new Node("E", null)) //  Insira um novo nó: E

    print(linkedList.getHead());
</script>
```

Resultado: A -> B -> E -> C -> D -> End

4. Exclua o índice = 2 nós.



1. Crie um arquivo: TestSingleLinkList.html

```
<script type="text/javascript"> class Node{
   constructor(data, next){
   this.data = data; this.next = next; }

   getData(){
   return this.data; }
   }
   /////////////// LinkedList ////////////////
   class LinkedList{
   init() {
   this.head = new Node("A", null); // o primeiro nó chamado cabe?a nó
   var nodeB = new Node("B", null); this.head.next = nodeB;
   var nodeC = new Node("C", null); nodeB.next = nodeC;
   this.tail = new Node("D", null); // o último nó chamado nó da cauda
nodeC.next = this.tail; }

   remove(removePosition) {
   var p = this.head; var i = 0; // Mova o nó para a posição do nó anterior
que deseja excluir while (p.next != null && i < removePosition - 1) {
   p = p.next; i++; }
   var temp = p.next; p.next = p.next.next; temp.next = null; }

   getHead(){
   return this.head; }
   }
////////////////// test //////////////////
   function print(node) {
   var p = node; while (p != null) // Imprimir do início ao fim {
   var data = p.getData(); document.write(data + " -> "); p = p.next; }
   document.write("End<br><br>"); }

   var linkedList = new LinkedList(); linkedList.init();
linkedList.remove(2) //  deletar um nó no índice = 2
   print(linkedList.getHead()); </script>
```

Resultado: A -> B -> D -> End

Vantagens da lista de links 1. A lista vinculada é uma estrutura de dados dinâmica que pode aumentar e diminuir em tempo de execução, alocando e desalocando memória. Portanto, não há necessidade de fornecer o tamanho inicial da lista vinculada.

2. A inserção e exclusão de nós são realmente mais fáceis. Ao contrário da matriz aqui, não temos que mudar os elementos após a inserção ou exclusão de um elemento. Na lista vinculada apenas temos que atualizar o endereço presente no próximo ponteiro de um nó.

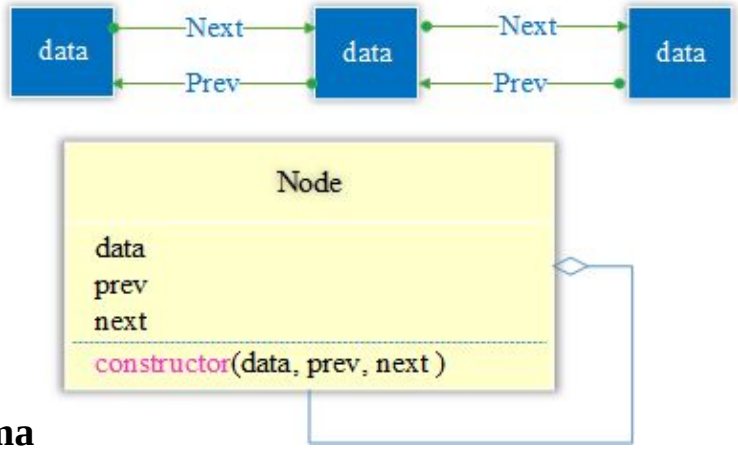
3. Como o tamanho da lista vinculada pode aumentar ou diminuir em tempo de execução, não há desperdício de memória. No caso do array, há muito desperdício de memória, como se declararmos um array de tamanho 10 e armazenarmos apenas 6 elementos nele, então o espaço de 4 elementos será desperdiçado.

Desvantagens da lista de links 1. É necessária mais memória para armazenar elementos na lista encadeada em comparação com a matriz. Porque na lista encadeada cada nó contém um ponteiro e requer memória extra para si mesmo.

2. A travessia de elementos ou nós é difícil na lista vinculada. Não podemos acessar aleatoriamente qualquer elemento como fazemos em array por índice. Por exemplo, se quisermos acessar um nó na posição n, temos que atravessar todos os nós antes dele. Portanto, o tempo necessário para acessar um nó é grande.

Lista duplamente vinculada

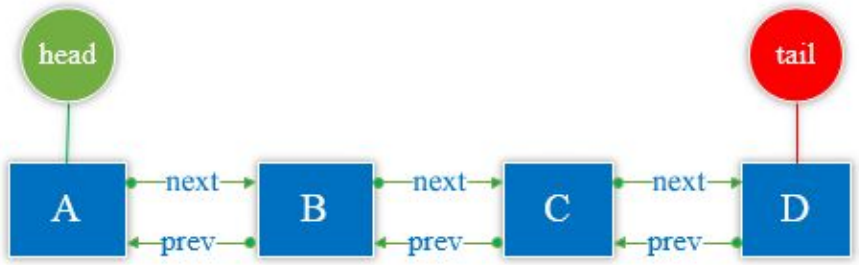
Lista duplamente vinculada (Doubly Linked List): É uma estrutura de armazenamento encadeado de uma tabela linear. É conectado por nós em duas direções. Cada nó consiste em dados, apontando para o nó anterior e apontando para o próximo nó.



UML Diagrama

```
class Node{
constructor(data, prev, next){
this.data = data; this.prev = prev; this.next = next; }

getData(){
return this.data; }
}
```

1. Inicialização de lista duplamente vinculada .
Exemplo: Construir uma lista vinculada




```
   class DoubleLinkList{

   init() {
   this.head = new Node("A"); //o primeiro nó chamado cabe?a nó
this.head.prev = null; this.head.next = null;
   var nodeB = new Node("B"); nodeB.prev = this.head; nodeB.next =
null; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.prev = nodeB; nodeC.next = null;
nodeB.next = nodeC;
   this.tail = new Node("D"); //o último nó chamado nó da cauda
this.tail.prev = nodeC; this.tail.next = null; nodeC.next = this.tail; }
   }
```

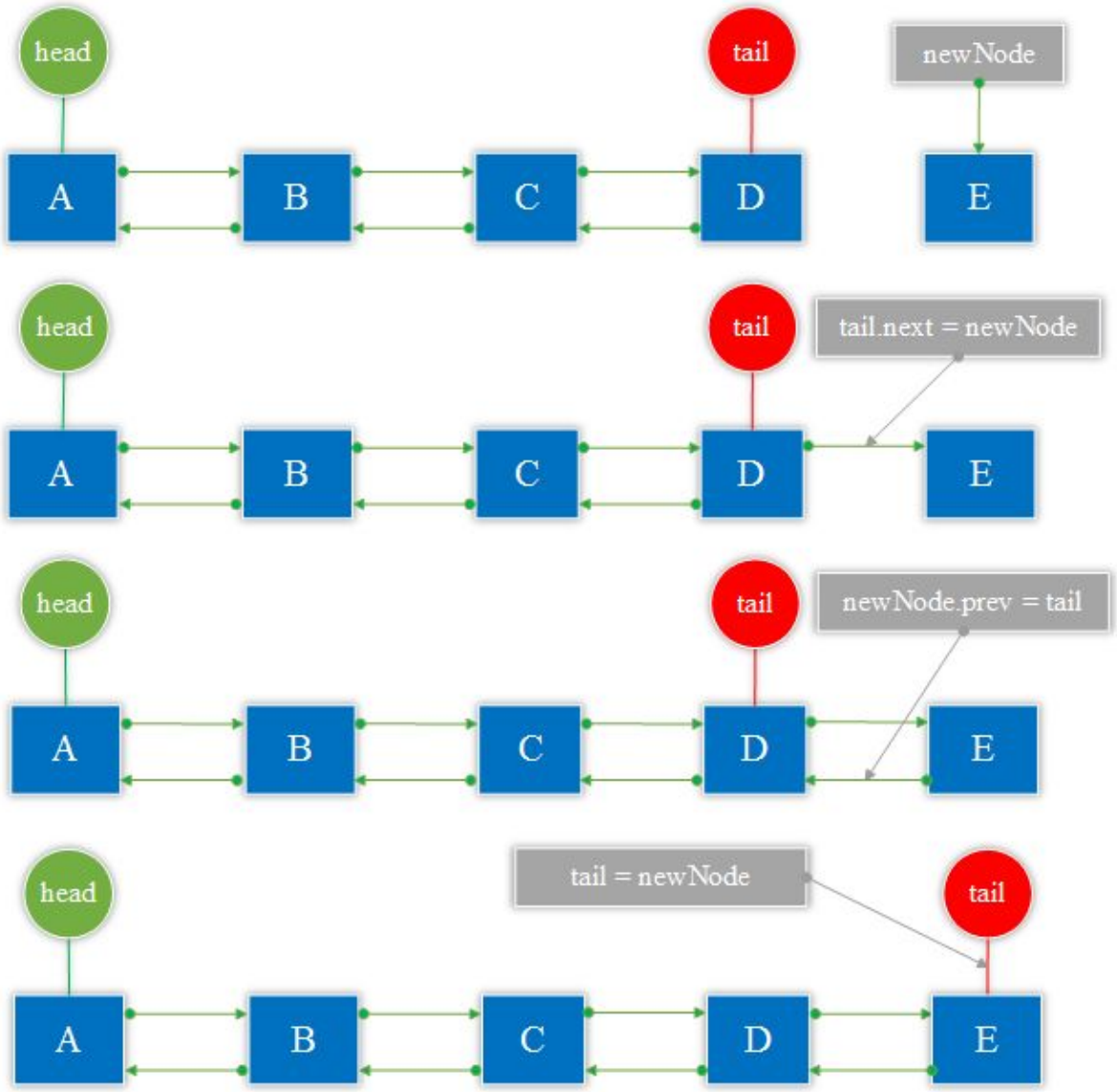
2. Crie um arquivo: TestDoubleLink.html

```
<script type="text/javascript"> /////////////// Node ////////////////
   class Node{
   constructor(data, prev, next){
   this.data = data; this.prev = prev; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// DoubleLinkList ////////////////
   class DoubleLinkList{

   init() {
   this.head = new Node("A"); //o primeiro nó chamado cabe?a nó
this.head.prev = null; this.head.next = null;
   var nodeB = new Node("B"); nodeB.prev = this.head; nodeB.next =
null; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.prev = nodeB; nodeC.next = null;
nodeB.next = nodeC;
   this.tail = new Node("D"); //o último nó chamado nó da cauda
this.tail.prev = nodeC; this.tail.next = null; nodeC.next = this.tail; }

   getHead(){
   return this.head; }
   }

////////////////// test //////////////////

   function print(node) {
   var p = node; var end = null; while (p != null){ // Imprimir do início
ao fim var data = p.getData(); document.write(data + " -> "); end = p; p
= p.next; }
   document.write("End <br><br>");
   p = end; while (p != null){ // Imprimir do final ao início var data =
p.getData(); document.write(data + " -> "); p = p.prev; }
```

```
    document.write("Start<br><br>"); }

    var doubleLinkList = new DoubleLinkList();
    doubleLinkList.init();
    print(doubleLinkList.getHead());
</script>
```

Resultado: A -> B -> C -> D -> End
D -> C -> B -> A -> Start

3. adicione um nó E no final.




```
add(newNode) {
    this.tail.next = newNode;
    newNode.prev = this.tail;
    this.tail = newNode;
}
```

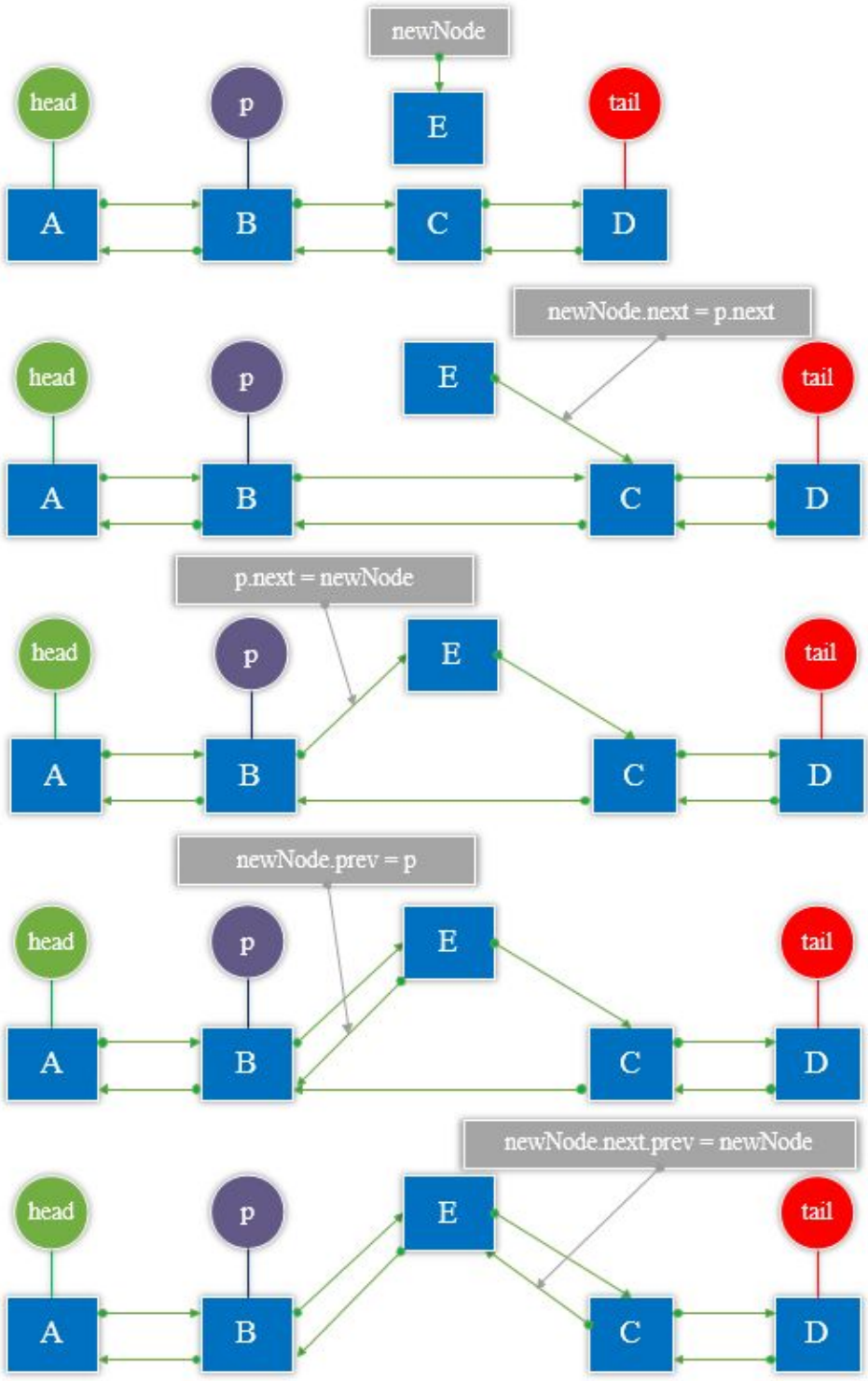
Crie um arquivo: TestDoubleLinkAdd.html

```
<script type="text/javascript"> /////////////// Node /////////////////
   class Node{
   constructor(data, prev, next){
   this.data = data; this.prev = prev; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// DoubleLinkList /////////////////
   class DoubleLinkList{
   init() {
   this.head = new Node("A"); //o primeiro nó chamado cabe?a nó
this.head.prev = null; this.head.next = null;
   var nodeB = new Node("B"); nodeB.prev = this.head; nodeB.next =
null; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.prev = nodeB; nodeC.next = null;
nodeB.next = nodeC;
   this.tail = new Node("D"); //o último nó chamado nó da cauda
this.tail.prev = nodeC; this.tail.next = null; nodeC.next = this.tail; }

   add(newNode) {
   this.tail.next = newNode; newNode.prev = this.tail; this.tail =
newNode; }

   getHead(){
   return this.head; }
   }

////////////////// test ////////////////////
   function print(node) {
   var p = node; var end = null; while (p != null) //Imprimir do início ao
fim {
   var data = p.getData(); document.write(data + " -> "); end = p; p =
p.next; }
```

```
    document.write("End <br><br>");
    p = end; while (p != null) //Imprimir do final ao início {
    var data = p.getData(); document.write(data + " -> "); p = p.prev; }
    document.write("Start<br><br>"); }

    var doubleLinkList = new DoubleLinkList();
    doubleLinkList.init();
    doubleLinkList.add(new Node("E")) //  adicione um nó E

    print(doubleLinkList.getHead()); </script>
```

Resultado: A -> B -> C -> D -> E -> End
E -> D -> C -> B -> A -> Start

3. Insira um nó E na posição 2.



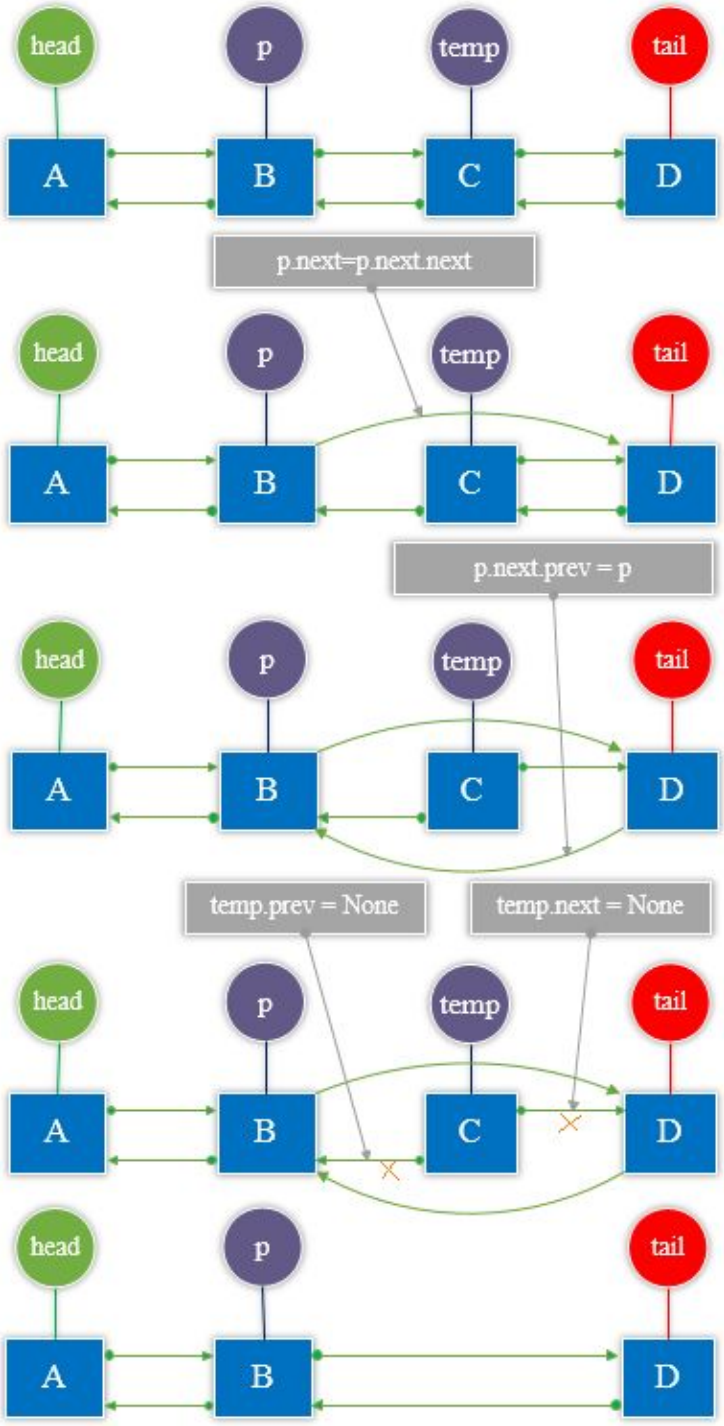
Crie um arquivo: TestDoubleLinkInsert.html

```
<script type="text/javascript">
   /////////////// Node ////////////////
   class Node{

   constructor(data, prev, next){
   this.data = data; this.prev = prev; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// DoubleLink ////////////////
   class DoubleLinkList{

   init() {
   this.head = new Node("A"); //o primeiro nó chamado cabe?a nó
this.head.prev = null; this.head.next = null;
   var nodeB = new Node("B"); nodeB.prev = this.head; nodeB.next =
null; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.prev = nodeB; nodeC.next = null;
nodeB.next = nodeC;
   this.tail = new Node("D"); //o último nó chamado nó da cauda
this.tail.prev = nodeC; this.tail.next = null; nodeC.next = this.tail; }



   insert(insertPosition, newNode) {
   var p = this.head; var i = 0; // Mova o nó para a posição de inserção
while (p.next != null && i < insertPosition-1) {
   p = p.next; i++; }

   newNode.next = p.next; p.next = newNode; newNode.prev = p;
newNode.next.prev = newNode; }

   getHead(){
   return this.head; }
```

```
    }

////////////////// test //////////////////

    function print(node) {
    var p = node; var end = null; while (p != null) //  Imprimir do início
ao fim {
    var data = p.getData(); document.write(data + " -> "); end = p; p =
p.next; }
    document.write("End <br><br>");
    p = end; while (p != null) //  Imprimir do final ao início {
    var data = p.getData(); document.write(data + " -> "); p = p.prev; }
    document.write("Start<br><br>"); }

    var doubleLinkList = new DoubleLinkList();
    doubleLinkList.init();
    doubleLinkList.insert(2, new Node("E")) //  Insira um nó E no índice =
2.

    print(doubleLinkList.getHead());
</script>
```

Resultado: A -> B -> E -> C -> D -> End D -> C -> E -> B -> A -> Start

4. Exclua o índice = 2 nós.



Crie um arquivo: TestDoubleLinkDelete.html

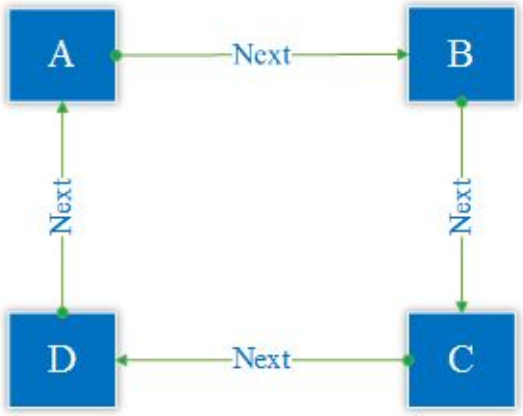
```
<script type="text/javascript">
   ////////////// Node ////////////////
   class Node{

   constructor(data, prev, next){
   this.data = data; this.prev = prev; this.next = next; }

   getData(){
   return this.data; }
   }

   ////////////// DoubleLink ////////////////
   class DoubleLinkList{

   init() {
   this.head = new Node("A"); //o primeiro nó chamado cabe?a nó
this.head.prev = null; this.head.next = null;
   var nodeB = new Node("B"); nodeB.prev = this.head; nodeB.next =
null; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.prev = nodeB; nodeC.next = null;
nodeB.next = nodeC;
   this.tail = new Node("D"); //o último nó chamado nó da cauda
this.tail.prev = nodeC; this.tail.next = null; nodeC.next = this.tail; }



   remove(removePosition) {
   var p = this.head; var i = 0; // Mova o nó para o nó anterior que deseja
excluir while (p.next != null && i < removePosition - 1) {
   p = p.next; i++; }

   var temp = p.next;// Salve o nó que deseja excluir p.next = p.next.next;
p.next.prev = p; temp.next = null;// delete node temp.prev = null;// delete
node }
```

```
   getHead(){
   return this.head; }
   }

////////////////// test ////////////////////

   function print(node) {
   var p = node; var end = null; while (p != null) //  Imprimir do início
ao fim {
   var data = p.getData(); document.write(data + " -> "); end = p; p =
p.next; }
   document.write("End <br><br>");
   p = end; while (p != null) //  Imprimir do final ao início {
   var data = p.getData(); document.write(data + " -> "); p = p.prev; }
   document.write("Start<br><br>"); }

   var doubleLinkList = new DoubleLinkList();
   doubleLinkList.init();
   doubleLinkList.remove(2) //  Exclua o nó em índice = 2.

   print(doubleLinkList.getHead());
</script>
```

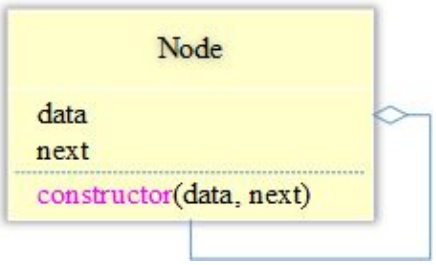
Resultado: A -> B -> D -> End D -> B -> A -> Start

Lista vinculada circular unidirecional

Lista vinculada circular unidirecional (One-way Circular List): É uma estrutura de armazenamento em cadeia de uma tabela linear, que é conectada para formar um anel, e cada nó é composto de dados e um ponteiro para o próximo.

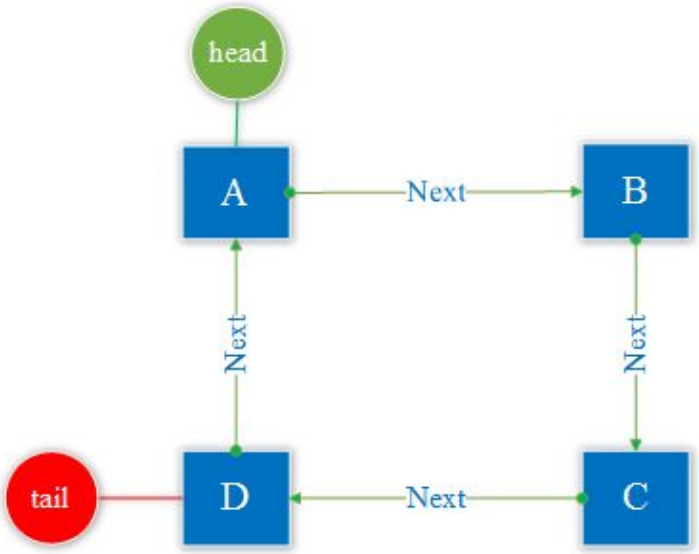


UML Diagrama



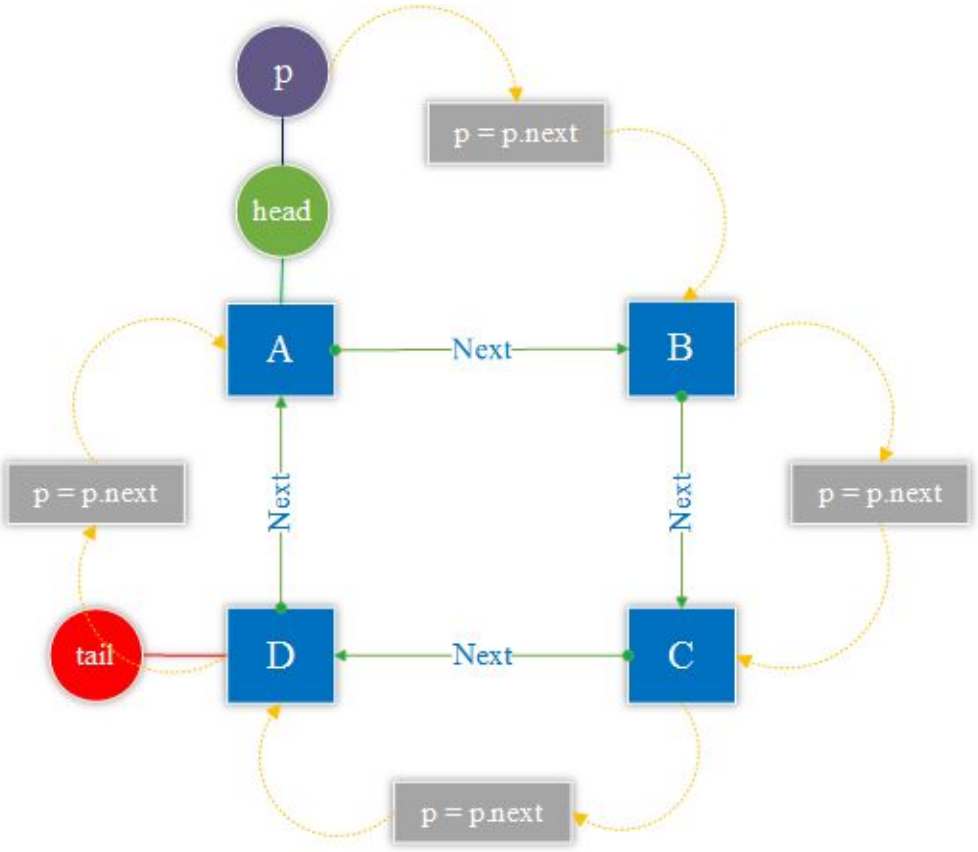

```
class Node{
constructor(data, next){
this.data = data; this.next = next; }

getData(){
return this.data; }
}
```

1. Inicialização de lista vinculada circular unidirecional .




```
   class SingleCircleLink{

   init() {
   // the first node called head node this.head = new Node("A");
this.head.next = null;
   var nodeB = new Node("B"); nodeB.next = null; this.head.next =
nodeB;
   var nodeC = new Node("C"); nodeC.next = null; nodeB.next = nodeC;
   // the last node called tail node this.tail = new Node("D"); this.tail.next
= this.head; nodeC.next = this.tail; }
   }
```

Navegar na lista de acesso




```
function print(head) {
    var p = head;
    do{
        var data = p.getData();
        document.write(data + " -> ");
        p = p.next;
    }while(p != head);

    var data = p.getData();
    document.write(data + "<br><br>");
}
```

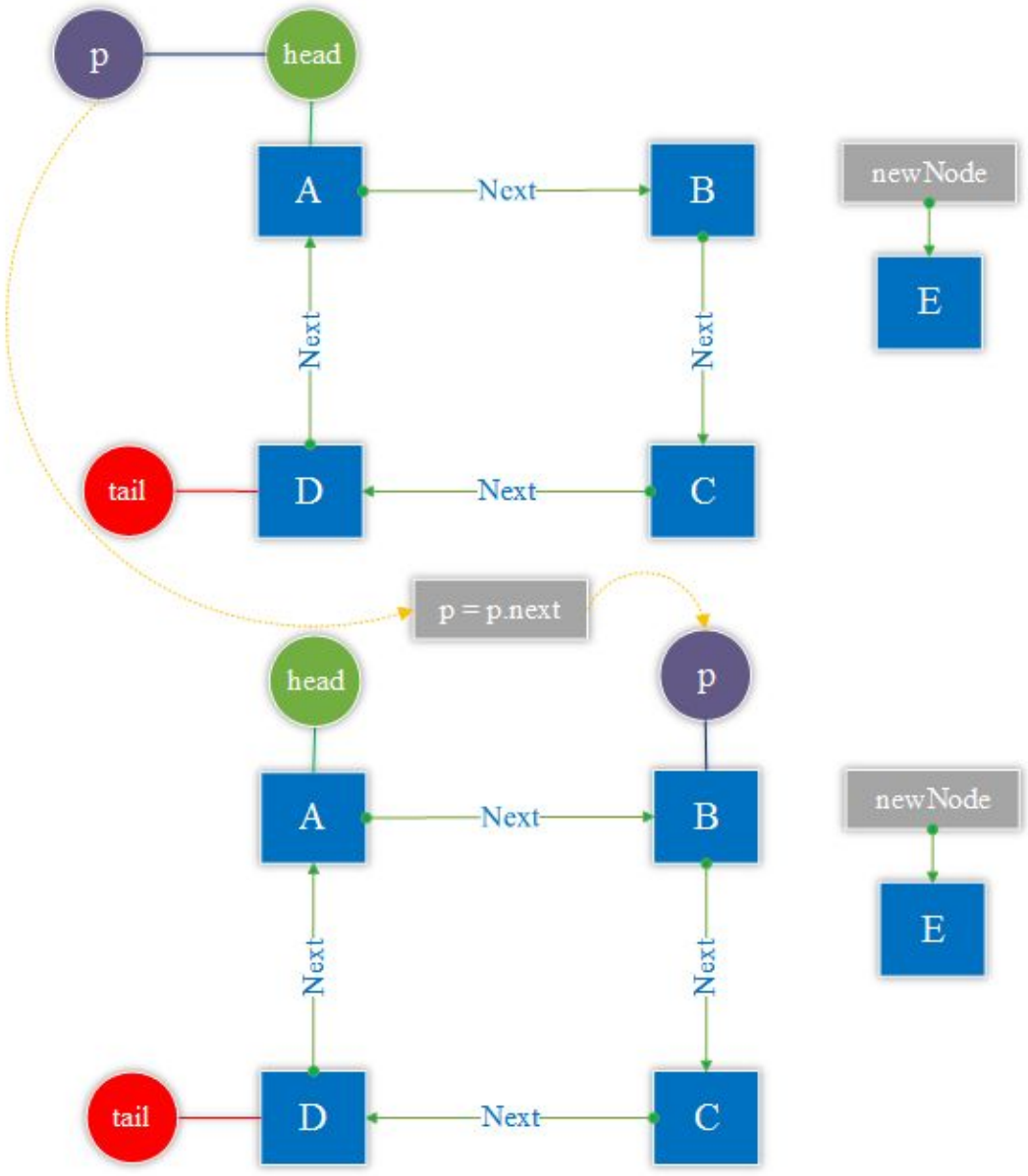
Crie um arquivo: TestSingleCircleLink.html

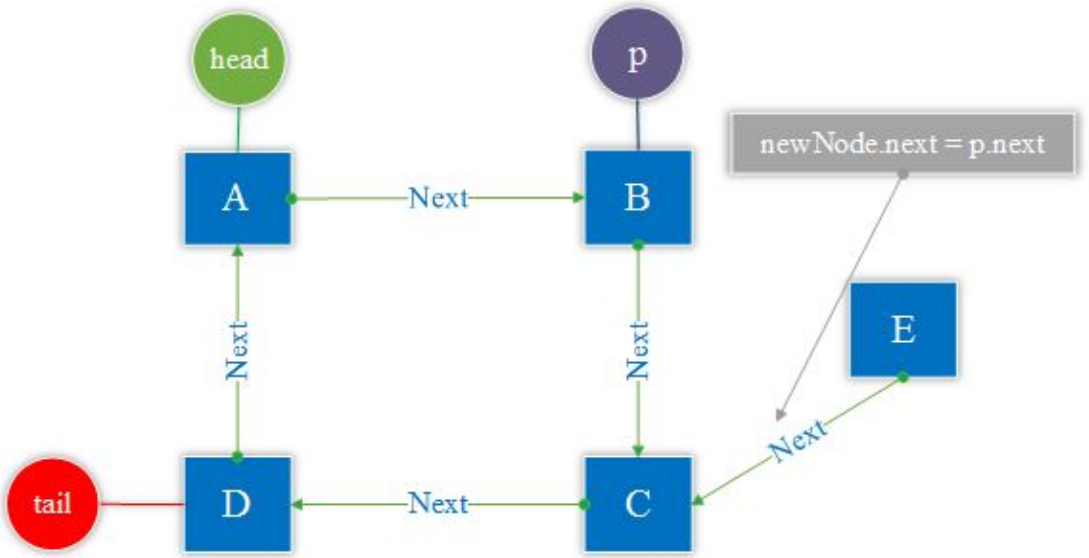
```
<script type="text/javascript">
   /////////////// Node /////////////////
   class Node{
   constructor(data, next){
   this.data = data; this.next = next; }

   getData(){
   return this.data; }
   }


   /////////////// SingleCircleLink /////////////////
   class SingleCircleLink{
   init() {
   // o primeiro nó chamado cabe?a nó this.head = new Node("A");
this.head.next = null;
   var nodeB = new Node("B"); nodeB.next = null; this.head.next =
nodeB;
   var nodeC = new Node("C"); nodeC.next = null; nodeB.next = nodeC;
   //o último nó chamado nó da cauda this.tail = new Node("D");
this.tail.next = this.head; nodeC.next = this.tail; }

   getHead(){
   return this.head; }
   }


////////////////// test ////////////////////

   function print(head) {
   var p = head; do{//Imprimir do início ao fim var data = p.getData();
document.write(data + " -> "); p = p.next; }while(p != head);
   var data = p.getData(); document.write(data + "<br><br>"); }
```

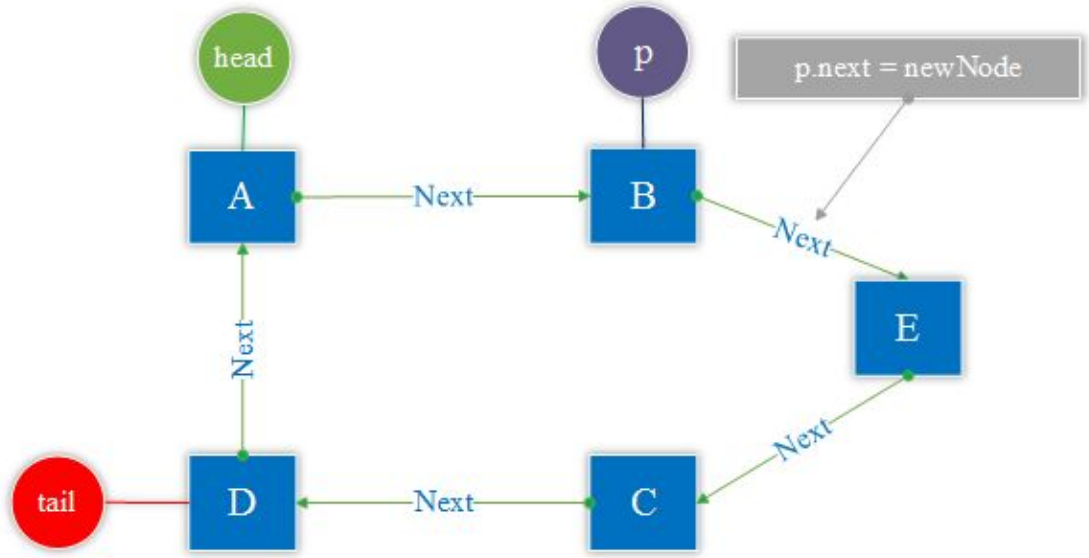
```
    var singleCircleLink = new SingleCircleLink();
    singleCircleLink.init();
    print(singleCircleLink.getHead());
</script>
```

Resultado: A -> B -> C -> D -> A

3. Insira um nó E na posição 2.




head
p
newNode.next = p.next
A
Next
B
E
Next
Next
Next
tail
D
Next
C


head
p
p.next = newNode
A
Next
B
Next
E
Next
Next
tail
D
Next
C

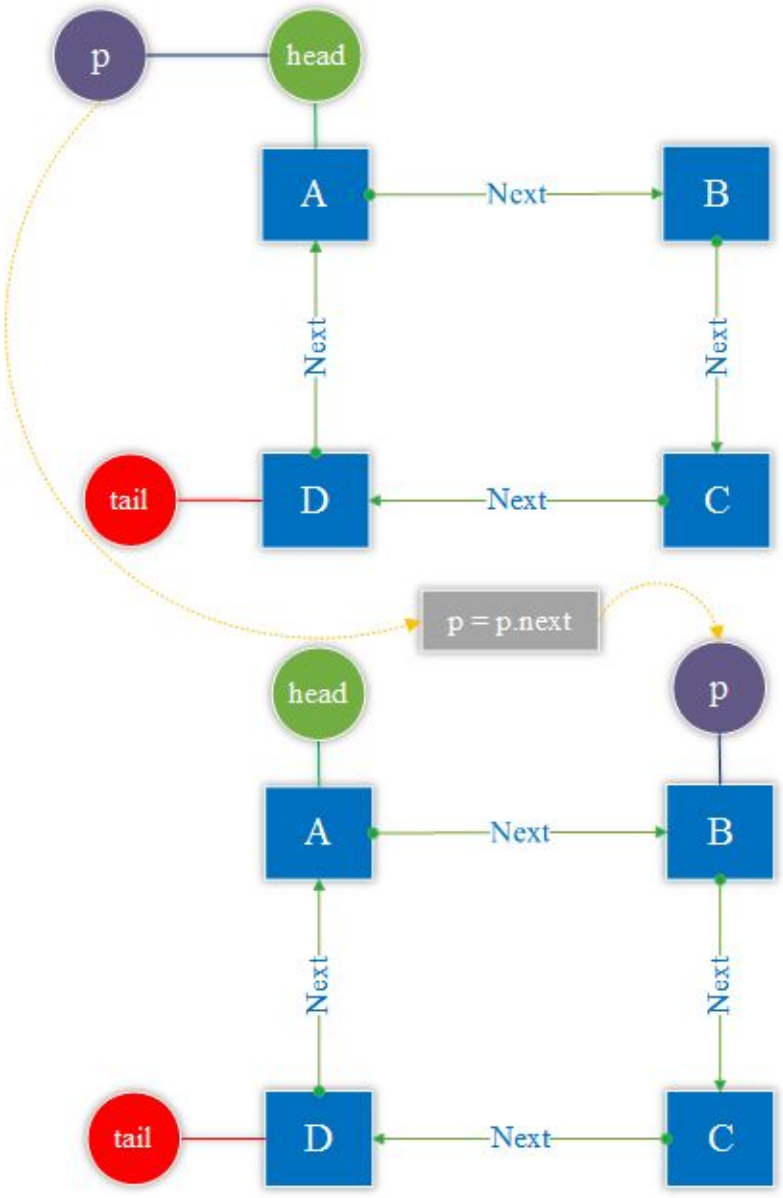
Crie um arquivo: TestSingleCircleLinkInsert.html

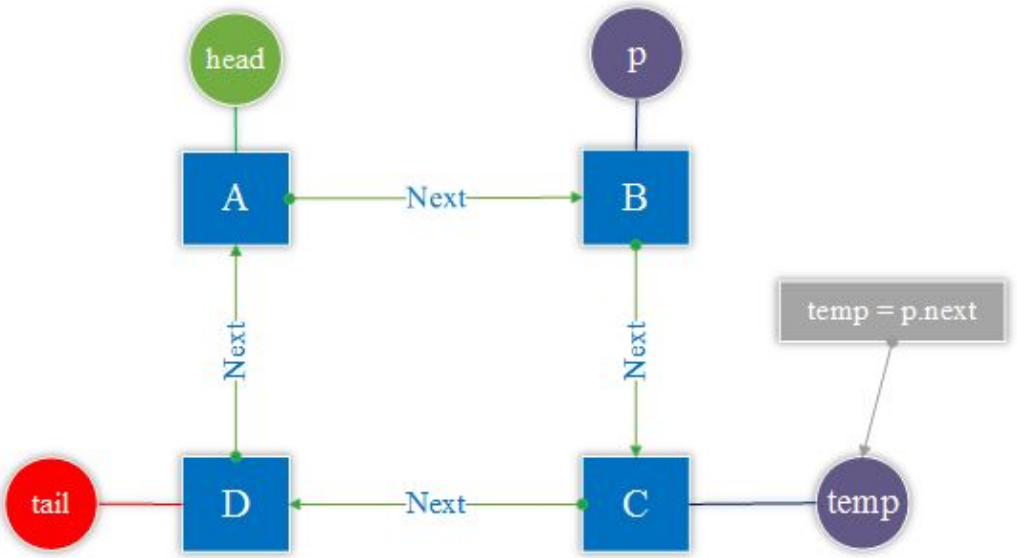
```
<script type="text/javascript"> /////////////// Node /////////////////
   class Node{
   constructor(data, next){
   this.data = data; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// SingleCircleLink /////////////////
   class SingleCircleLink{
   init() {
   this.head = new Node("A"); // o primeiro nó chamado cabe?a nó
this.head.next = null;
   var nodeB = new Node("B"); nodeB.next = null; this.head.next =
nodeB;
   var nodeC = new Node("C"); nodeC.next = null; nodeB.next = nodeC;
   this.tail = new Node("D"); // o último nó chamado nó da cauda
this.tail.next = this.head; nodeC.next = this.tail; }

   insert(insertPosition, newNode) {
   var p = this.head; var i = 0; while (p.next != null && i <
insertPosition - 1) {
   p = p.next; i++; }
   newNode.next = p.next; p.next = newNode; }

   getHead(){
   return this.head; }
   }

////////////////// test //////////////////

   function print(head) {
   var p = head; do{
```

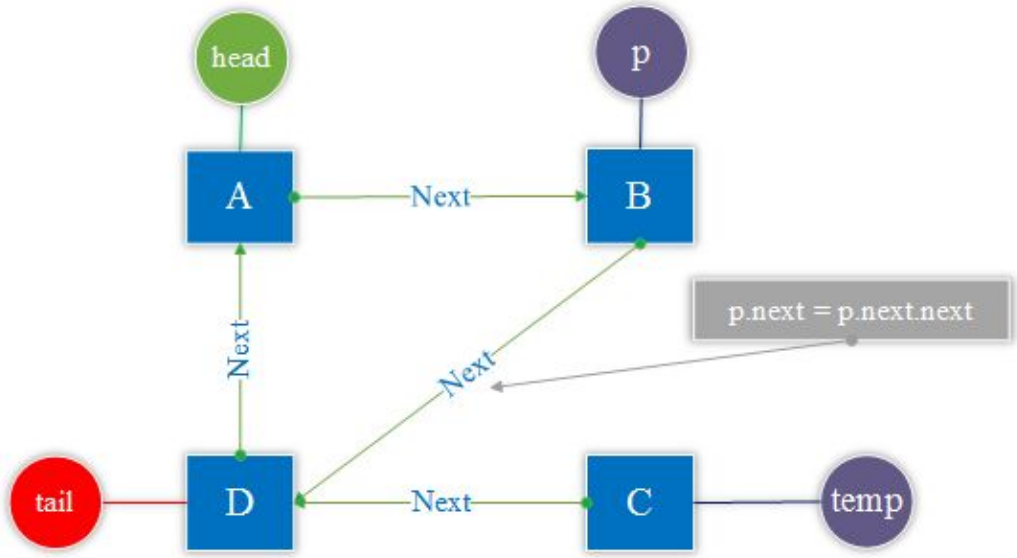
```
   var data = p.getData(); document.write(data + " -> "); p = p.next;
}while(p != head);
   var data = p.getData(); document.write(data + "<br><br>"); }

   var singleCircleLink = new SingleCircleLink();
   singleCircleLink.init();
   singleCircleLink.insert(2, new Node("E")) //  Insira um nó E no índice
= 2.

   print(singleCircleLink.getHead());
</script>
```

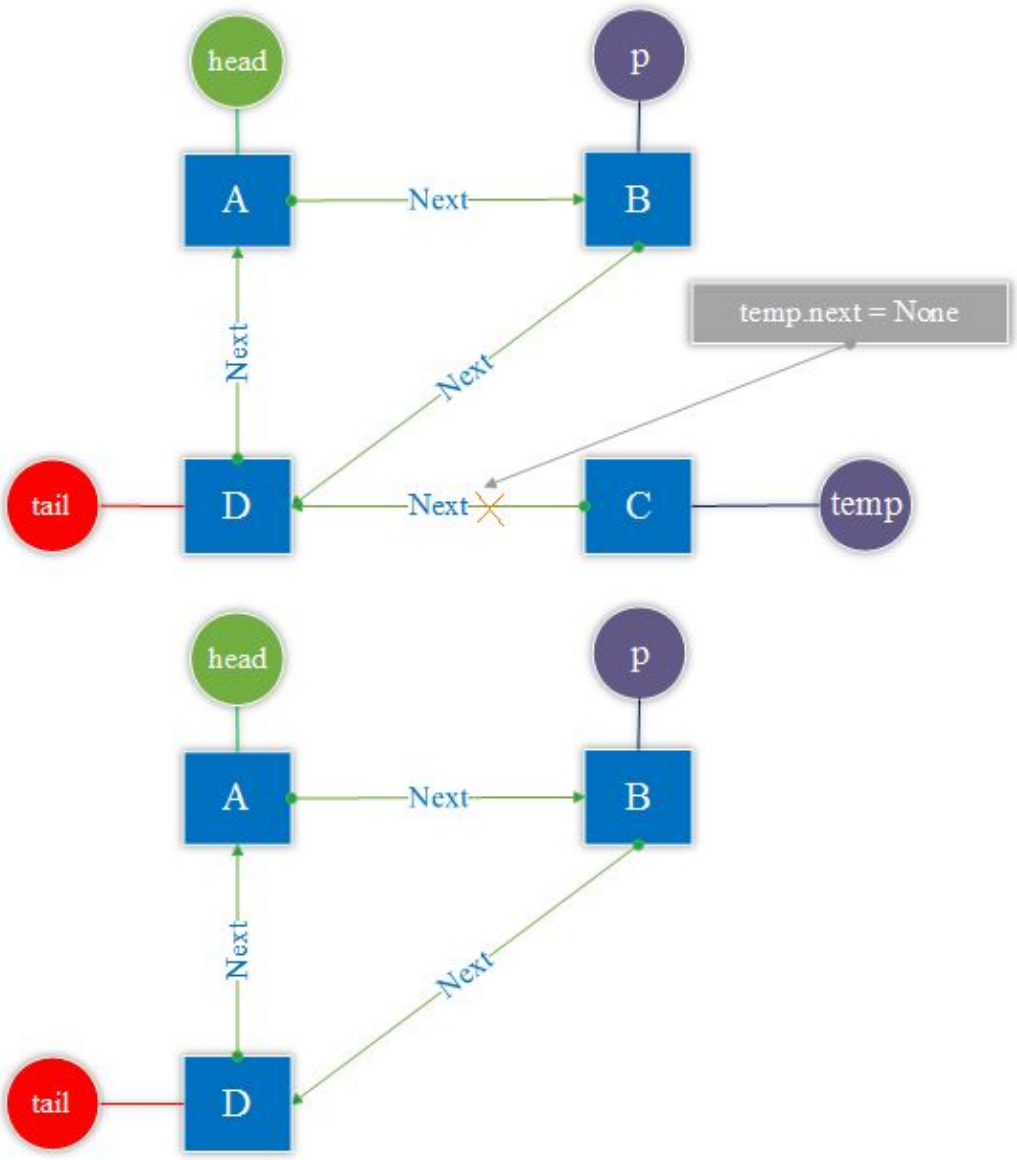
Resultado: A -> B -> E -> C -> D -> A

4. Exclua o índice = 2 nós.




head
p
A
Next
B
Next
Next
temp = p.next
tail
D
Next
C
temp


head
p
A
Next
B
p.next = p.next.next
Next
Next
tail
D
Next
C
temp


head
p
A
Next
B
Next
Next
temp.next = None
tail
D
Next
C
temp
head
p
A
Next
B
Next
Next
tail
D

Crie um arquivo:TestSingleCircleLinkDelete.html

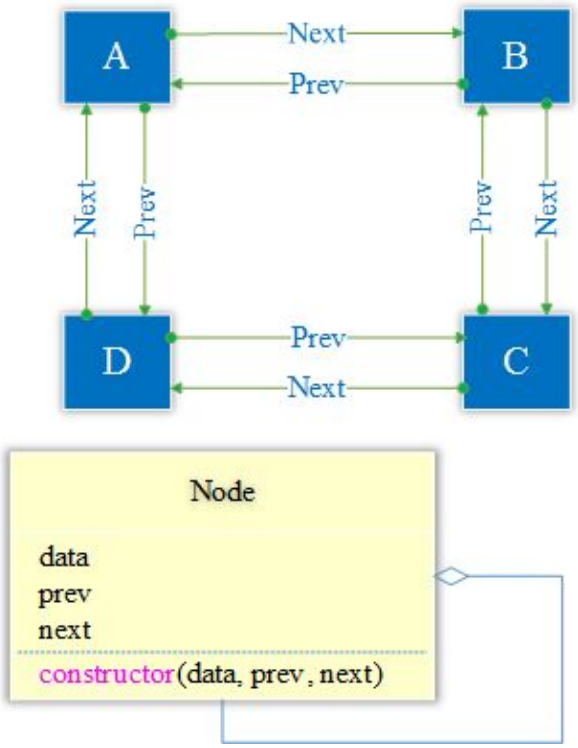
```
<script type="text/javascript"> /////////////// Node /////////////////
   class Node{
   constructor(data, next){
   this.data = data; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// SingleCircleLink /////////////////
   class SingleCircleLink{
   init() {
   this.head = new Node("A"); // o primeiro nó chamado cabe?a nó
this.head.next = null;
   var nodeB = new Node("B"); nodeB.next = null; this.head.next =
nodeB;
   var nodeC = new Node("C"); nodeC.next = null; nodeB.next = nodeC;
   this.tail = new Node("D"); // o último nó chamado nó da cauda
this.tail.next = this.head; nodeC.next = this.tail; }

   remove(removePosition) {
   var p = this.head; var i = 0; while (p.next != null && i <
removePosition - 1) {
   p = p.next; i++; }
   var temp = p.next; p.next = p.next.next; temp.next = null; }

   getHead(){
   return this.head; }
   }

////////////////// test //////////////////

   function print(head) {
   var p = head; do{//Imprimir do início ao fim var data = p.getData();
document.write(data + " -> "); p = p.next; }while(p != head);
   var data = p.getData(); document.write(data + "<br><br>"); }
```

```
    var singleCircleLink = new SingleCircleLink();
    singleCircleLink.init();
    singleCircleLink.remove(2) //  Exclua o nó em índice = 2.

    print(singleCircleLink.getHead());
</script>
```

Resultado: A -> B -> D -> A

Lista vinculada circular de duas vias

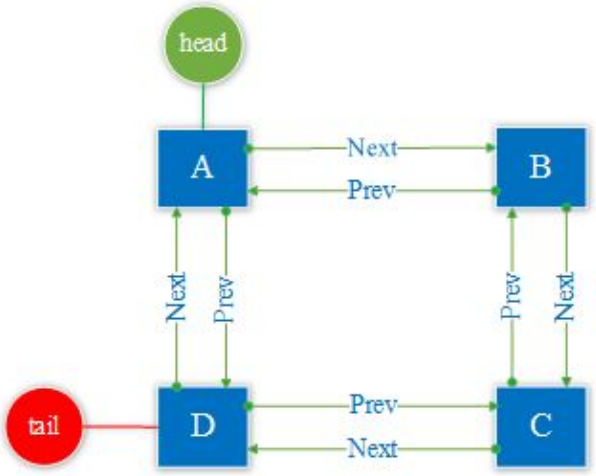
Lista vinculada circular de duas vias (Two-way Circular List): É uma estrutura de armazenamento em cadeia de uma tabela linear. Os nós são conectados em série por duas direções e são conectados para formar um anel. Cada nó é composto de dados, apontando para o nó anterior e apontando para o próximo nó.



UML Diagrama

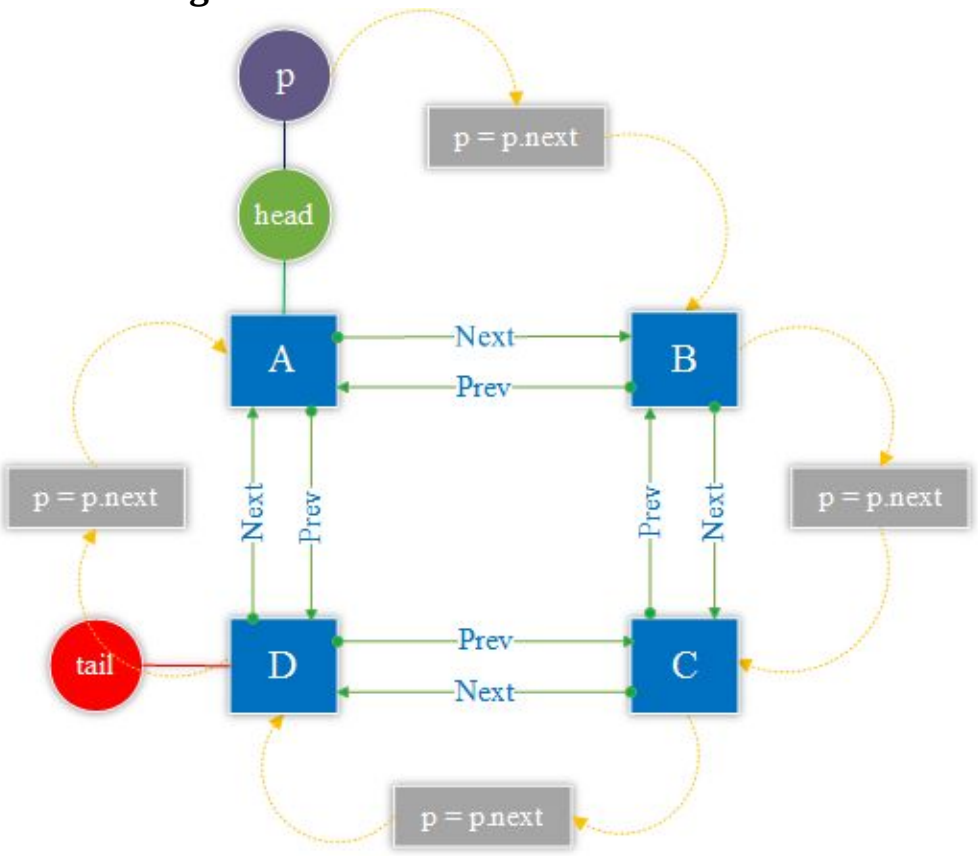
```
class Node{
constructor(data, prev, next){
this.data = data; this.prev = prev; this.next = next; }

getData(){
return this.data; }
}
```

1. Inicialização de lista vinculada circular em dois sentidos .

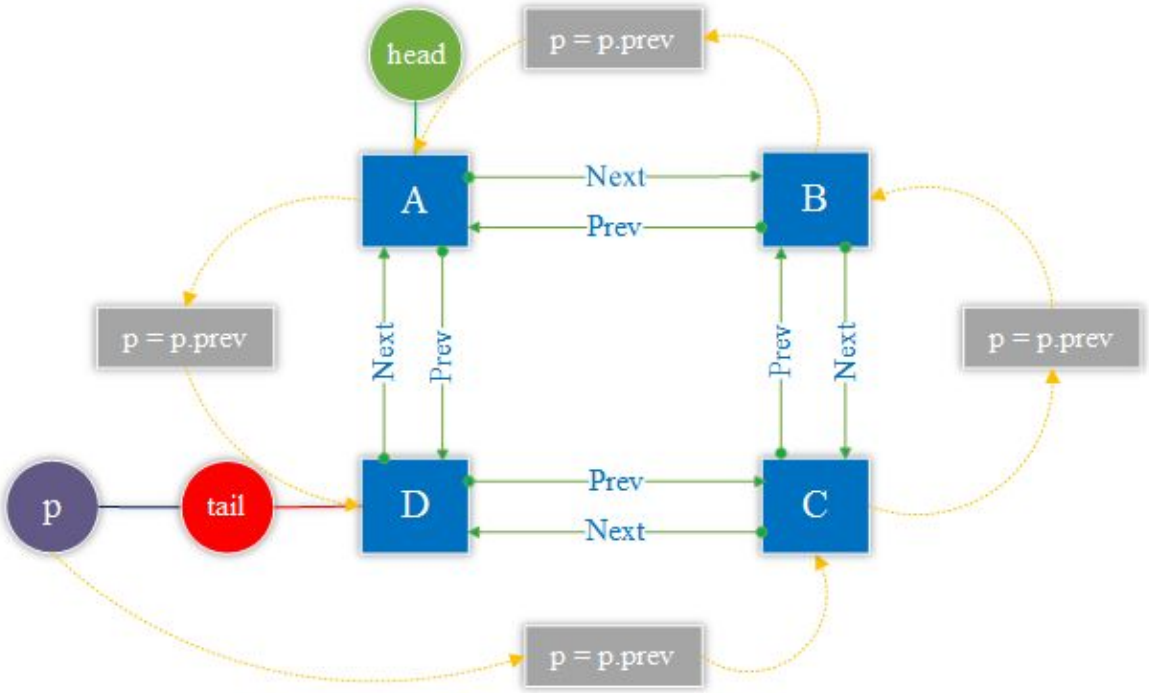



```
   class DoubleCircleLink{
   init() {
   // o primeiro nó chamado cabe?a nó this.head = new Node("A");
this.head.next = null; this.head.prev = null;
   var nodeB = new Node("B"); nodeB.next = null; nodeB.prev =
this.head; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.next = null; nodeC.prev = nodeB;
nodeB.next = nodeC;
   // o último nó chamado nó da cauda this.tail = new Node("D");
this.tail.next = this.head; this.tail.prev = nodeC; nodeC.next = this.tail;
this.head.prev = this.tail; }
   }
```

Navegue na lista ligada: next



Navegue na lista ligada: prev



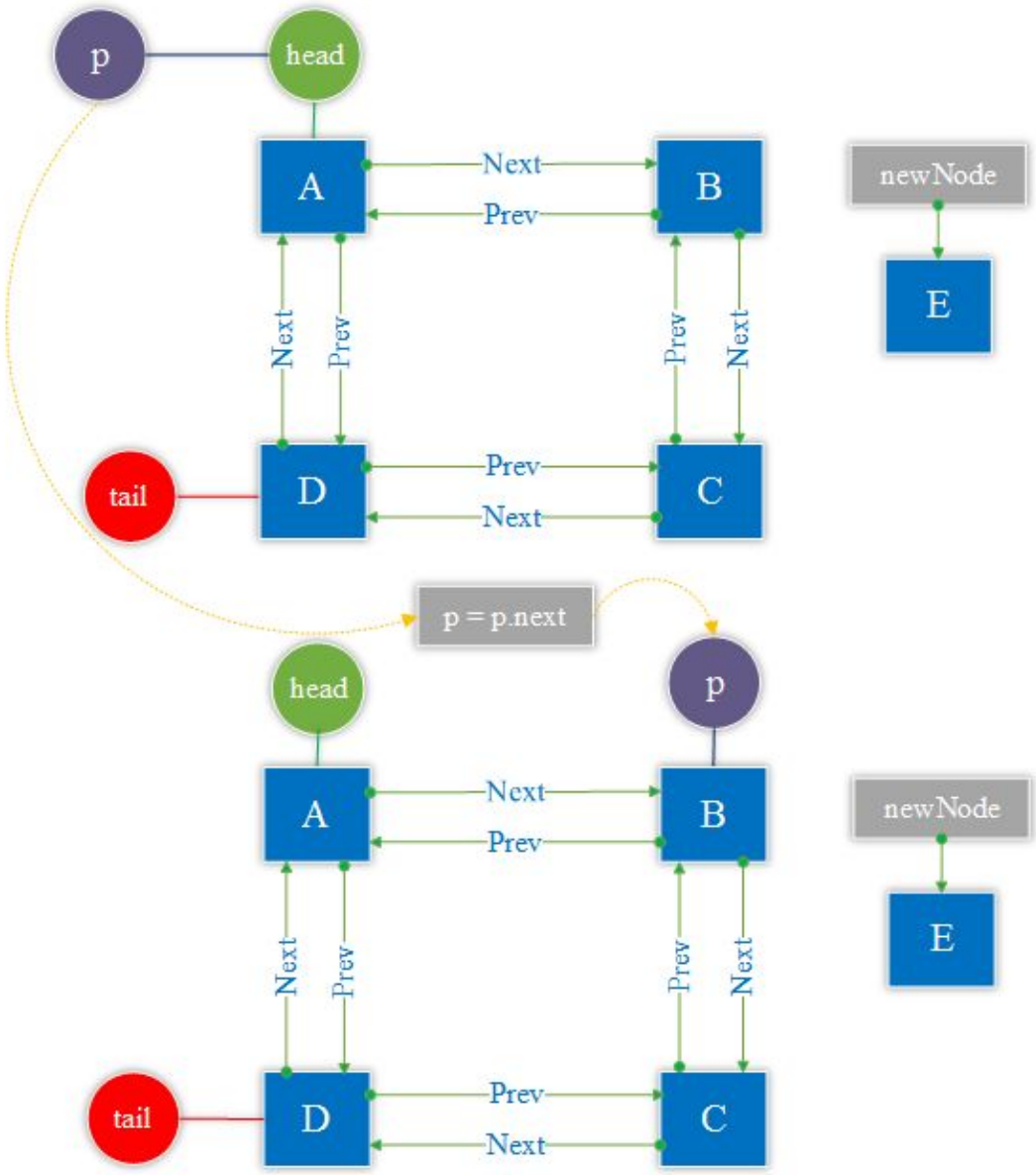
Crie um arquivo: TestDoubleCircleLink.html

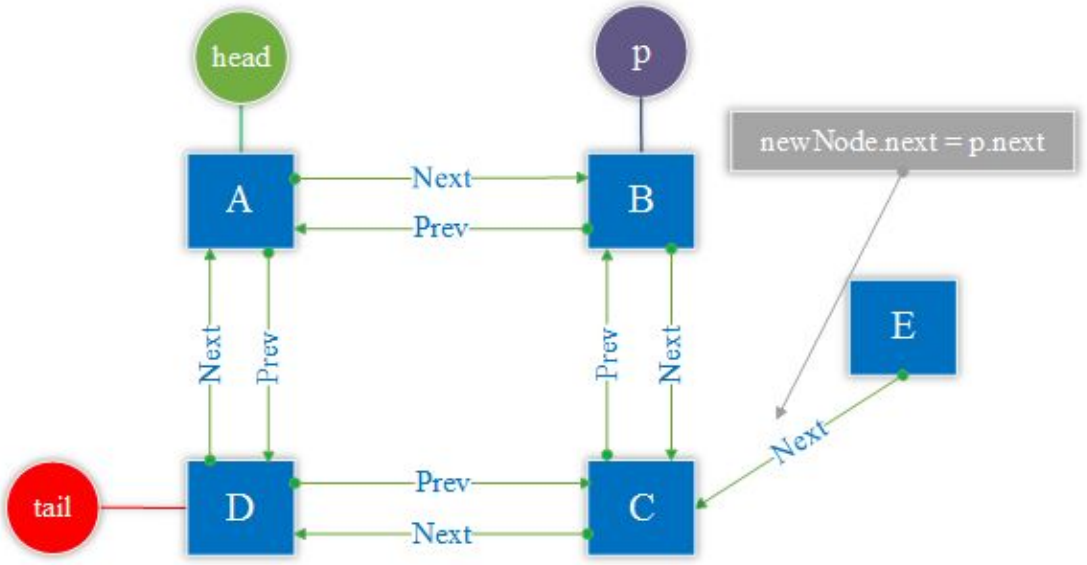
```
<script type="text/javascript">
   /////////////// Node /////////////////
   class Node{
   constructor(data, prev, next){
   this.data = data; this.prev = prev; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// DoubleCircleLink/////////////////
   class DoubleCircleLink{
   init() {
   // o primeiro nó chamado cabe?a nó this.head = new Node("A");
this.head.next = null; this.head.prev = null;
   var nodeB = new Node("B"); nodeB.next = null; nodeB.prev =
this.head; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.next = null; nodeC.prev = nodeB;
nodeB.next = nodeC;
   // o último nó chamado nó da cauda this.tail = new Node("D");
this.tail.next = this.head; this.tail.prev = nodeC; nodeC.next = this.tail;
this.head.prev = this.tail; }


   getHead(){
   return this.head; }

   getTail(){
   return this.tail; }

   }

////////////////// test ////////////////////
   function print(head, tail) {
   var p = head; //Imprimir do início ao fim: next do {
```

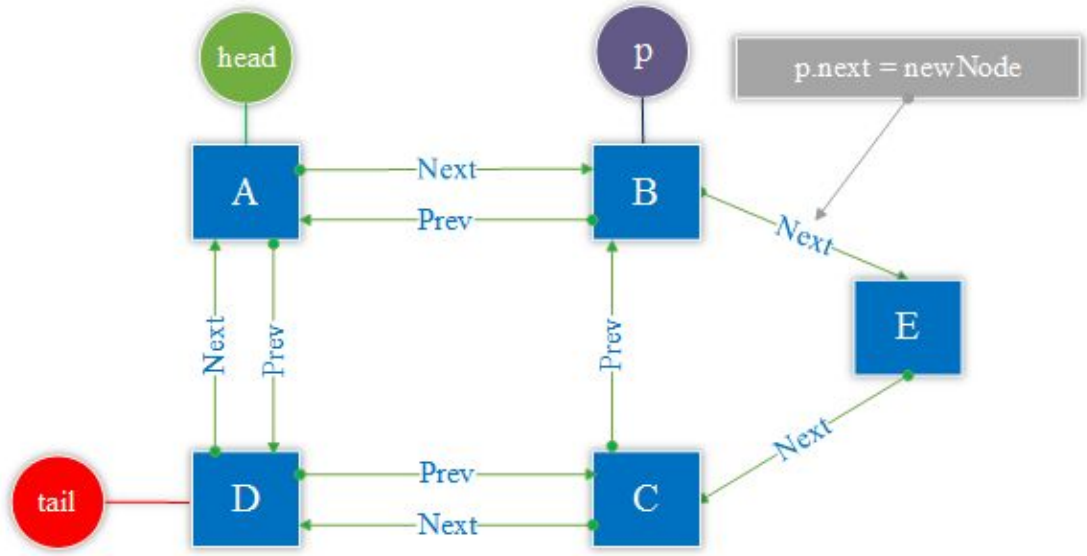
```
    var data = p.getData(); document.write(data + " -> "); p = p.next; }
while (p != head);
    var data = p.getData(); document.write(data + "<br><br>");
    p = tail; //Imprimir do final ao início: prev do {
    data = p.getData(); document.write(data + " -> "); p = p.prev; } while
(p != tail);
    data = p.getData(); document.write(data + "<br><br>"); }

    var doubleCircleLink = new DoubleCircleLink();
    doubleCircleLink.init();
    print(doubleCircleLink.getHead(), doubleCircleLink.getTail());
</script>
```

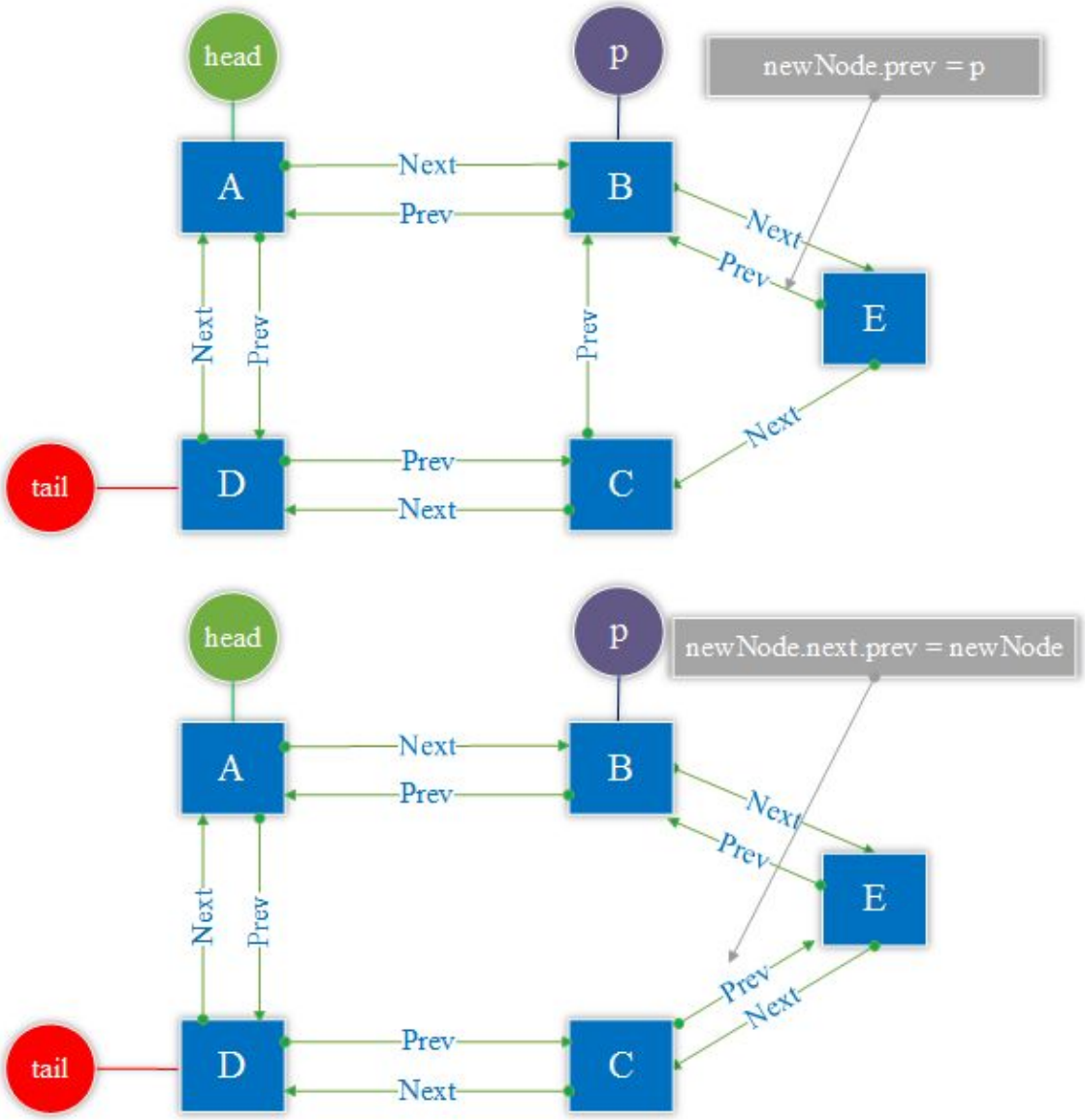
Resultado: A -> B -> C -> D -> A
D -> C -> B -> A -> D

3. Insira um nó E na posição 2.




head
p
newNode.next = p.next
A
Next
Prev
B
Next
Prev
Prev
Next
E
Next
tail
D
Prev
Next
C


head
p
p.next = newNode
A
Next
Prev
B
Next
E
Next
Prev
Prev
Next
tail
D
Prev
Next
C


head
p
newNode.prev = p
A
Next
Prev
B
Next
Prev
E
Next
Prev
Prev
Next
tail
D
Prev
Next
C
head
p
newNode.next.prev = newNode
A
Next
Prev
B
Next
Prev
E
Next
Prev
Prev
Next
tail
D
Prev
Next
C

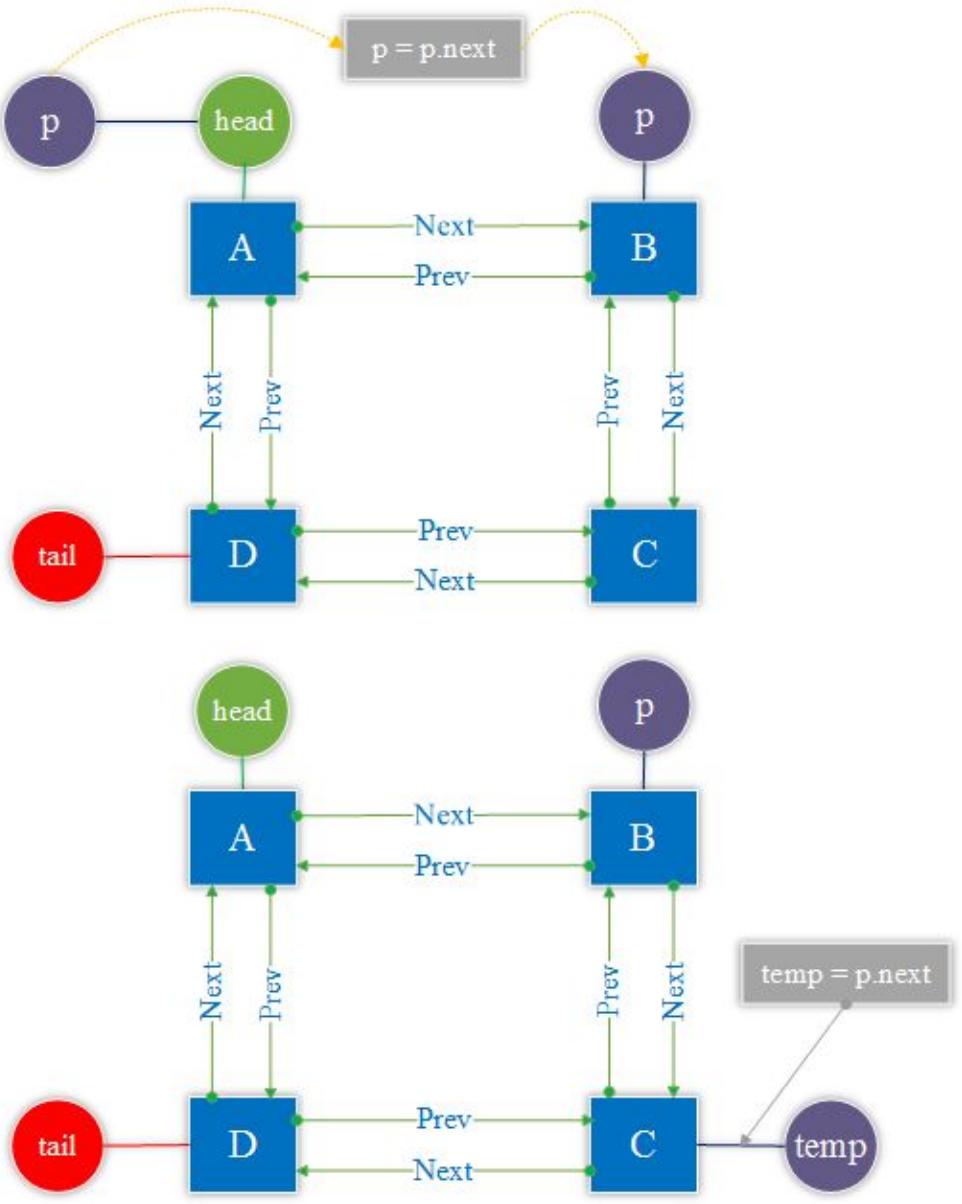
Crie um arquivo: TestDoubleCircleLinkInsert.html

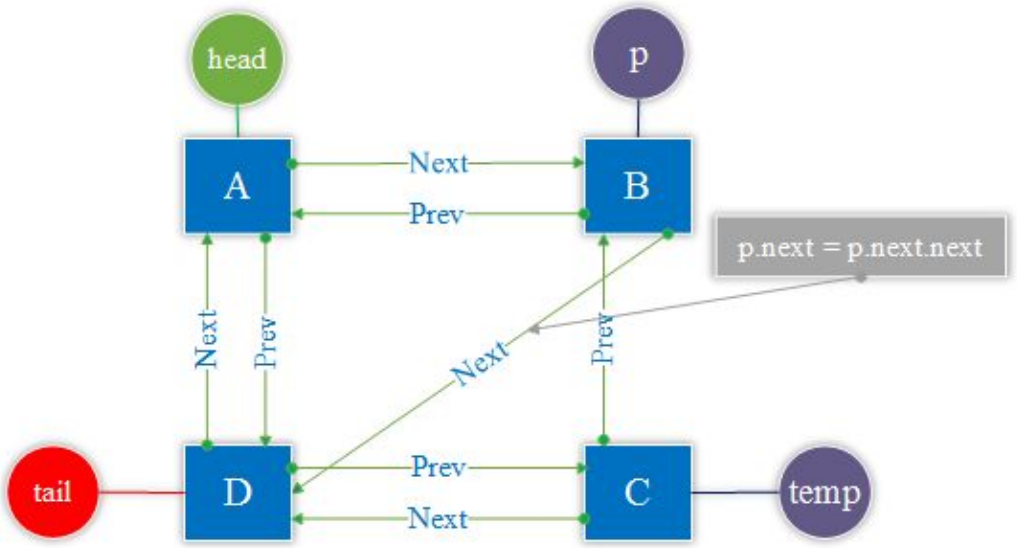
```
<script type="text/javascript">
   /////////////// Node /////////////////
   class Node{
   constructor(data, prev, next){
   this.data = data; this.prev = prev; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// DoubleCircleLink/////////////////
   class DoubleCircleLink{
   init() {
   // o primeiro nó chamado cabe?a nó this.head = new Node("A");
this.head.next = null; this.head.prev = null;
   var nodeB = new Node("B"); nodeB.next = null; nodeB.prev =
this.head; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.next = null; nodeC.prev = nodeB;
nodeB.next = nodeC;
   // o último nó chamado nó da cauda this.tail = new Node("D");
this.tail.next = this.head; this.tail.prev = nodeC; nodeC.next = this.tail;
this.head.prev = this.tail; }

   insert(insertPosition, newNode) {
   var p = this.head; var i = 0; //  Mova o nó para a posição de inserção
while (p.next != null && i < insertPosition - 1) {
   p = p.next; i++; }

   newNode.next = p.next; p.next = newNode; newNode.prev = p;
newNode.next.prev = newNode; }

   getHead(){
   return this.head; }
   getTail(){
   return this.tail; }
```

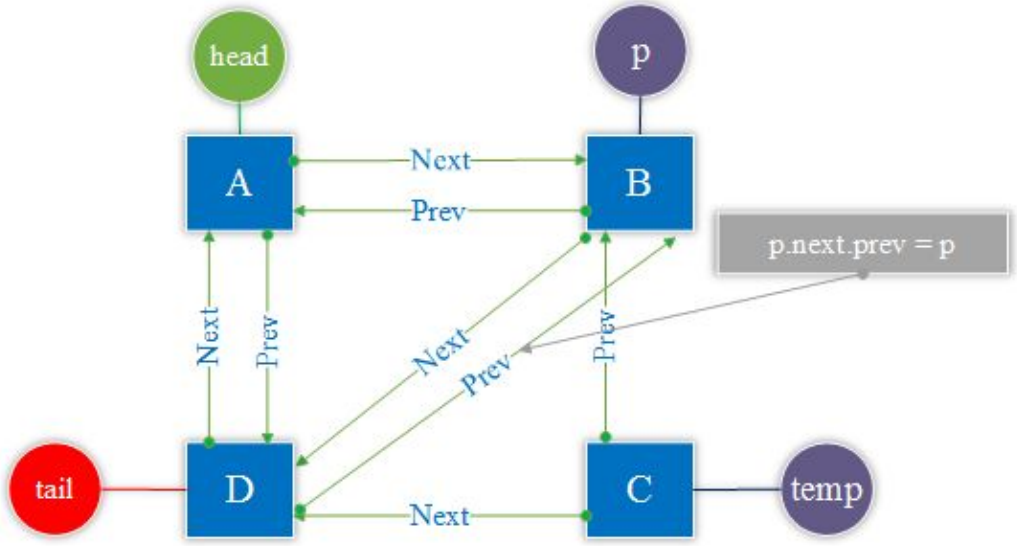
```
    }
////////////////// test //////////////////
    function print(head, tail) {
    var p = head; do {//Imprimir do início ao fim var data = p.getData();
document.write(data + " -> "); p = p.next; } while (p != head);
    var data = p.getData(); document.write(data + "<br><br>");
    p = tail; do {//Imprimir do final ao início data = p.getData();
document.write(data + " -> "); p = p.prev; } while (p != tail);
    data = p.getData(); document.write(data + "<br><br>"); }

    var doubleCircleLink = new DoubleCircleLink();
    doubleCircleLink.init();
    doubleCircleLink.insert(2, new Node("E")) //  Insira um nó E na
posição 2

    print(doubleCircleLink.getHead(), doubleCircleLink.getTail());
</script>
```

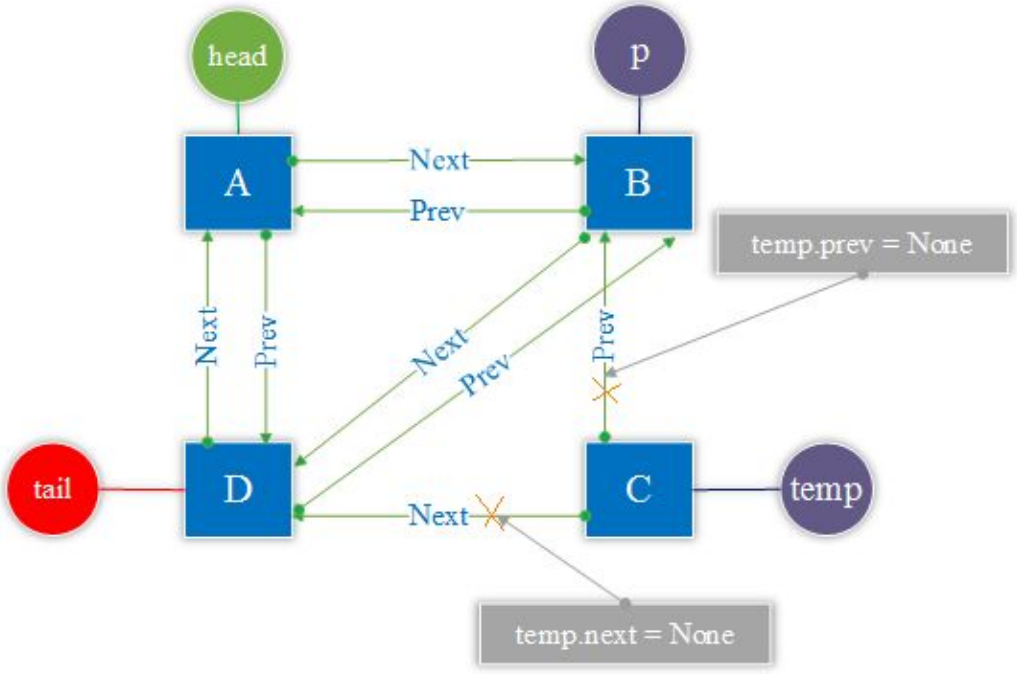
Resultado: A -> B -> E -> C -> D -> A
D -> C -> E -> B -> A -> D

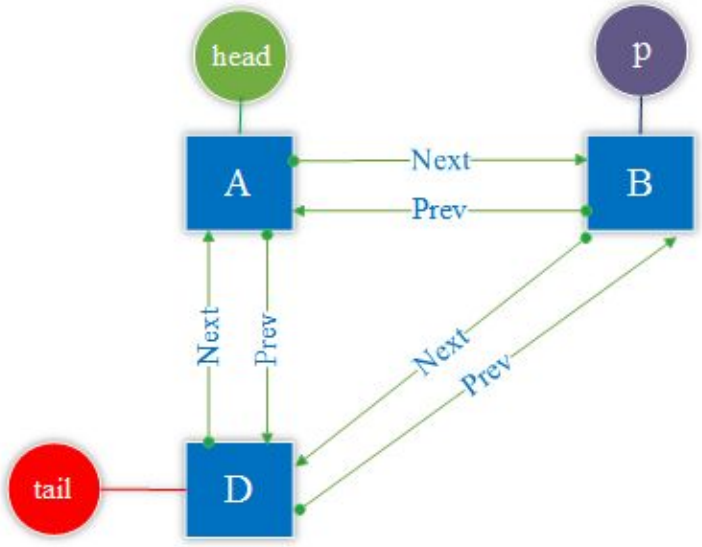
4. Exclua o índice = 2 nós.




head
p
A
Next
B
Prev
p.next = p.next.next
Next
Prev
Next
Prev
tail
D
Prev
C
temp
Next


head
p
A
Next
B
Prev
p.next.prev = p
Next
Prev
Next
Prev
tail
D
C
temp
Next


head
p
A
Next
B
Prev
temp.prev = None
Next
Prev
Next
Prev
tail
D
C
temp
Next
temp.next = None


head
p
A
Next
B
Prev
Next
Prev
Next
Prev
tail
D

Crie um arquivo: TestDoubleCircleLinkDelete.html

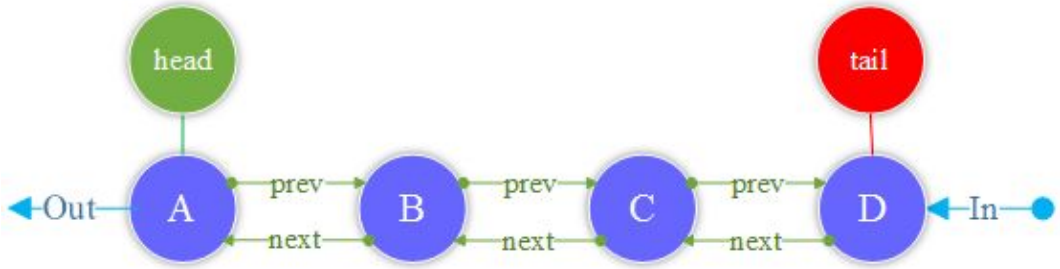
```
<script type="text/javascript">
   /////////////// Node /////////////////
   class Node{
   constructor(data, prev, next){
   this.data = data; this.prev = prev; this.next = next; }

   getData(){
   return this.data; }
   }

   /////////////// DoubleCircleLink/////////////////
   class DoubleCircleLink{
   init() {
   // o primeiro nó chamado cabe?a nó this.head = new Node("A");
this.head.next = null; this.head.prev = null;
   var nodeB = new Node("B"); nodeB.next = null; nodeB.prev =
this.head; this.head.next = nodeB;
   var nodeC = new Node("C"); nodeC.next = null; nodeC.prev = nodeB;
nodeB.next = nodeC;
   //o último nó chamado nó da cauda this.tail = new Node("D");
this.tail.next = this.head; this.tail.prev = nodeC; nodeC.next = this.tail;
this.head.prev = this.tail; }


   remove(removePosition) {
   var p = this.head; var i = 0; while (p.next != null && i <
removePosition - 1) {
   p = p.next; i++; }

   var temp = p.next; p.next = p.next.next; p.next.prev = p; temp.next =
null;//  remover nó temp.prev = null;// remover nó }

   getHead(){
   return this.head; }
   getTail(){
```

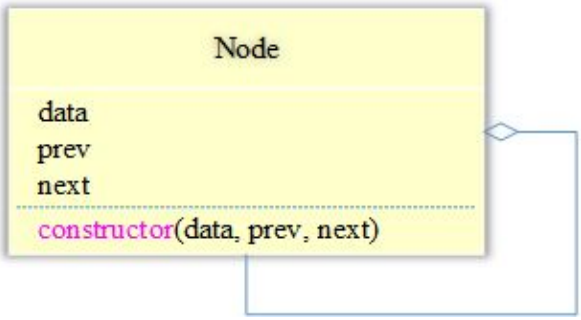
```
    return this.tail; }
    }
////////////////// test //////////////////
    function print(head, tail) {
    var p = head; do {//  Imprimir do início ao fim var data = p.getData();
document.write(data + " -> "); p = p.next; } while (p != head);
    var data = p.getData(); document.write(data + "<br><br>");
    p = tail; do {//  Imprimir do final ao início data = p.getData();
document.write(data + " -> "); p = p.prev; } while (p != tail);
    data = p.getData(); document.write(data + "<br><br>"); }

    var doubleCircleLink = new DoubleCircleLink();
    doubleCircleLink.init();
    doubleCircleLink.remove(2) //  Exclua um nó E no índice = 2

    print(doubleCircleLink.getHead(), doubleCircleLink.getTail());
</script>
```

Resultado: A -> B -> D -> A
D -> B -> A -> D

Fila

Fila (Queue): Uma fila é uma estrutura linear que segue uma ordem particular na qual as operações são realizadas. O pedido é Primeiro a Entrar, Primeiro a Sair (FIFO).



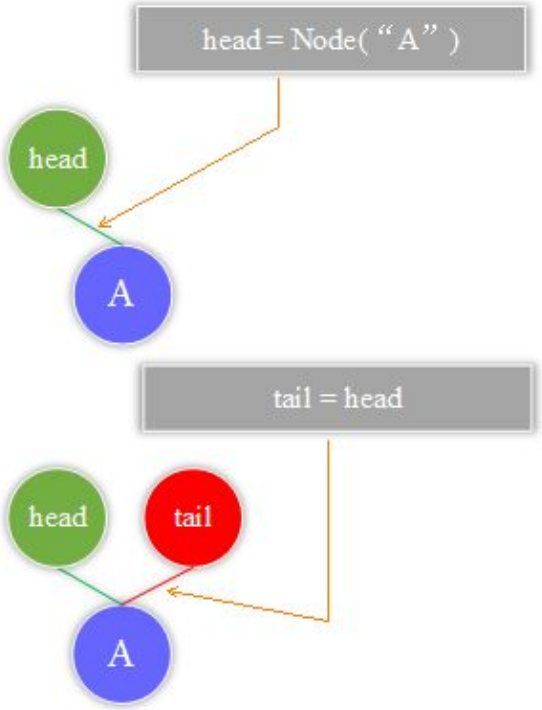


UML Diagrama

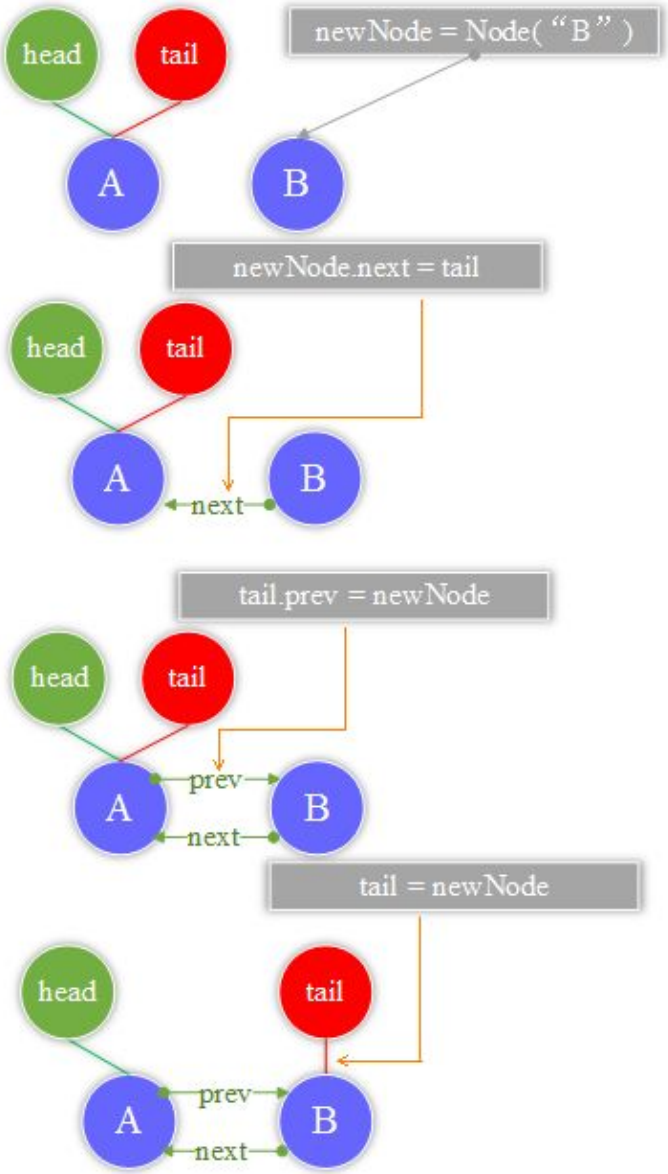
```
class Node{
constructor(data, prev, next){
this.data = data; this.prev = prev; this.next = next; }
getData(){
return this.data; }
}
```

1. Inicialização de fila .

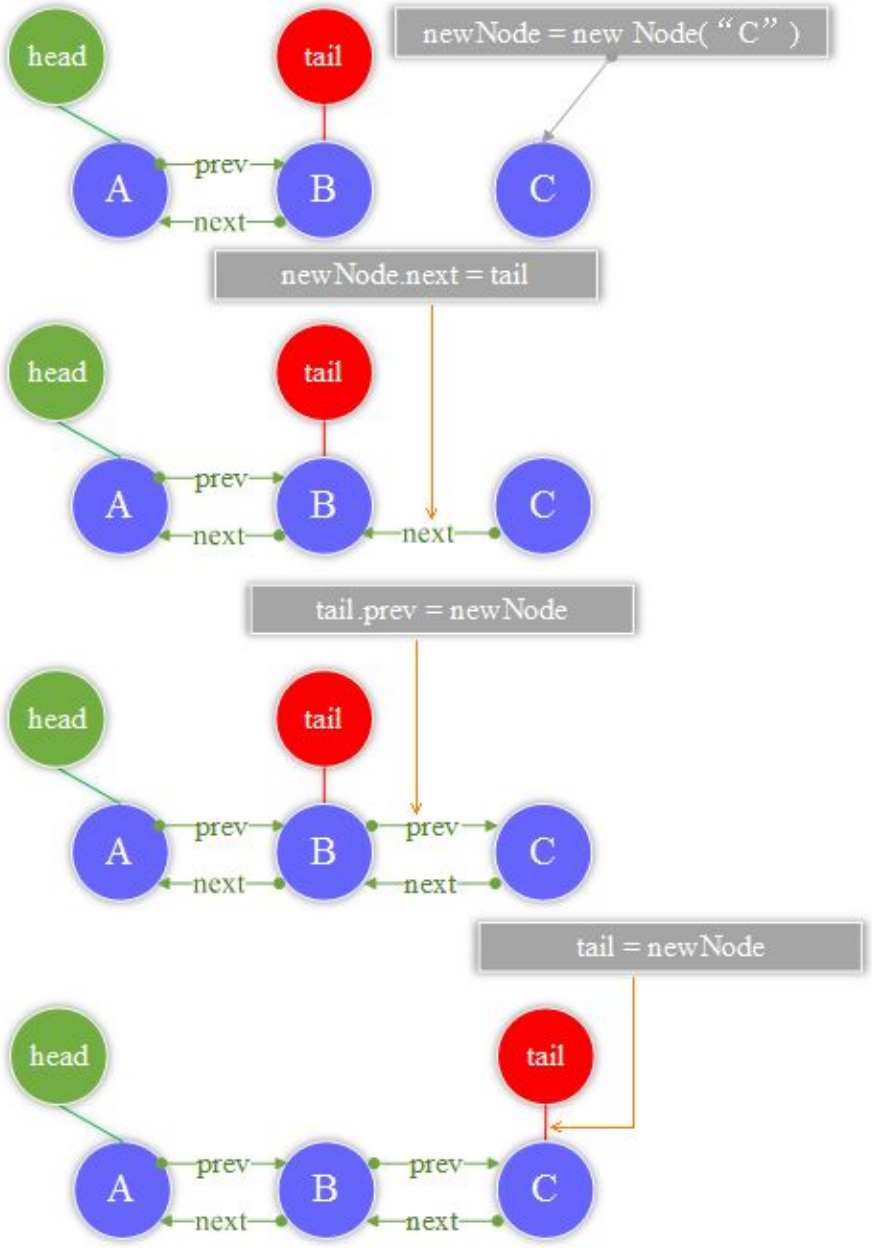
Inserir A



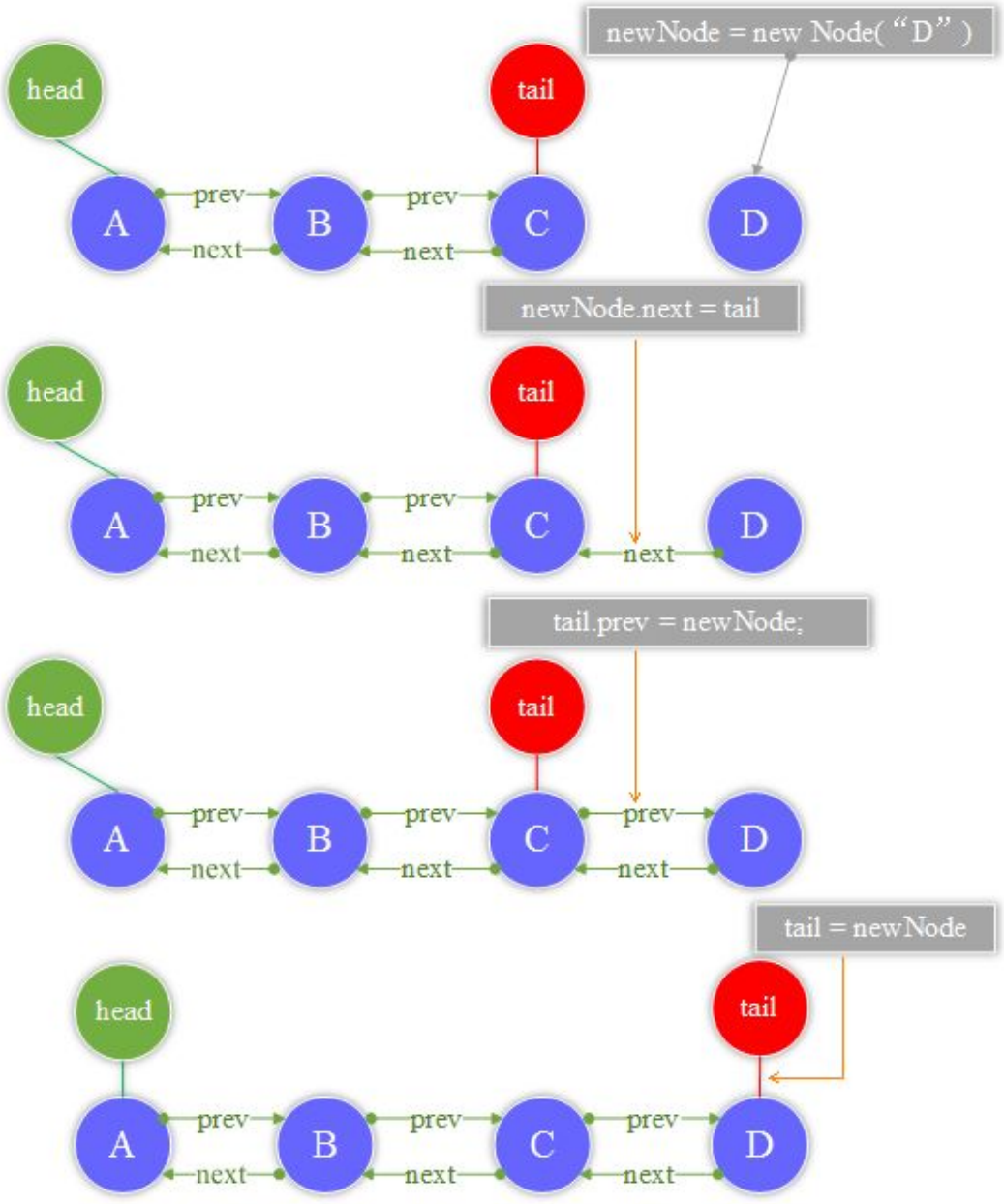
Inserir B


head
tail
newNode = Node(“B”)
A
B
newNode.next = tail
head
tail
A
next
B
tail.prev = newNode
head
tail
prev
A
B
next
tail = newNode
head
tail
prev
A
B
next

Inserir C


head
tail
newNode = new Node(“ C”)
A
prev
B
C
next
newNode.next = tail
head
tail
A
prev
B
C
next
next
tail.prev = newNode
head
tail
A
prev
B
prev
C
next
next
tail = newNode
head
tail
A
prev
B
prev
C
next
next

Inserir D


newNode = new Node("D")
head
tail
A
B
C
D
prev
next
newNode.next = tail
tail.prev = newNode;
tail = newNode

Crie um arquivo: TestQueue.html

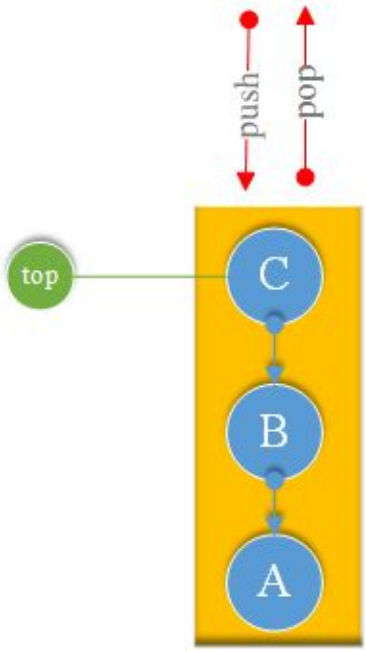
```
<script type="text/javascript"> /////////////// Node ///////////////
   class Node{
   constructor(data, prev, next){
   this.data = data; this.prev = prev; this.next = next; }
   getData(){
   return this.data; }
   }

   ////////////// Queue ///////////////
   class Queue{
   constructor(){
   this.head = null; this.tail = null; this.size = 0; // o tamanho da fila }

   // adicionar elemento à fila offer(element) {
   if (this.head == null) { // se a fila estiver vazia this.head = new
Node(element);// adicionar elemento à cabeça this.tail = this.head; } else
{
   var newNode = new Node(element); newNode.next = this.tail;
this.tail.prev = newNode; this.tail = newNode; }
   this.size++; }

   size() {
   return this.size; }




   // pegue o cabeçote e remova o elemento principal da fila poll() {
   var p = this.head; if (p == null) {
   return null; }
   this.head = this.head.prev; p.next = null; p.prev = null; this.size--;
return p; }
   }

////////////////// test //////////////////
```

```
   function print(queue) {
   document.write("Head "); var node = null; while ((node =
queue.poll())!=null) {
   document.write(node.getData() + " <- "); }
   document.write("Tail <br>"); }

   var queue = new Queue(); queue.offer("A"); queue.offer("B");
queue.offer("C"); queue.offer("D");
    print(queue);
</script>
```

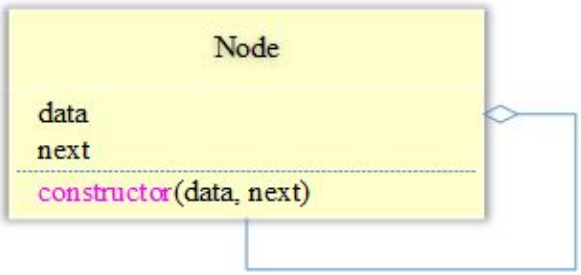
Resultado: Head A <- B <- C <- D <- Tail

Pilha

Pilha (Stack): Stack é uma estrutura de dados linear que segue uma ordem particular na qual as operações são realizadas. O pedido pode ser LIFO (Last In First Out).



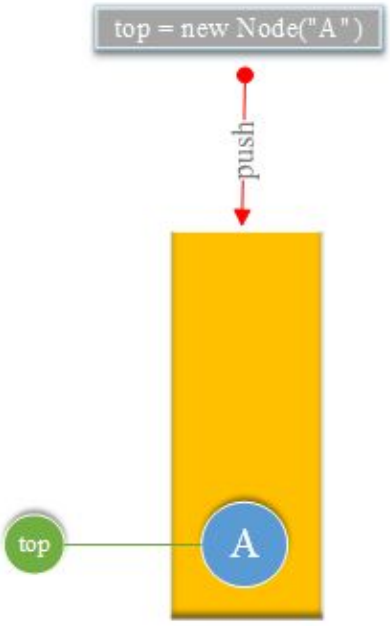
UML Diagrama



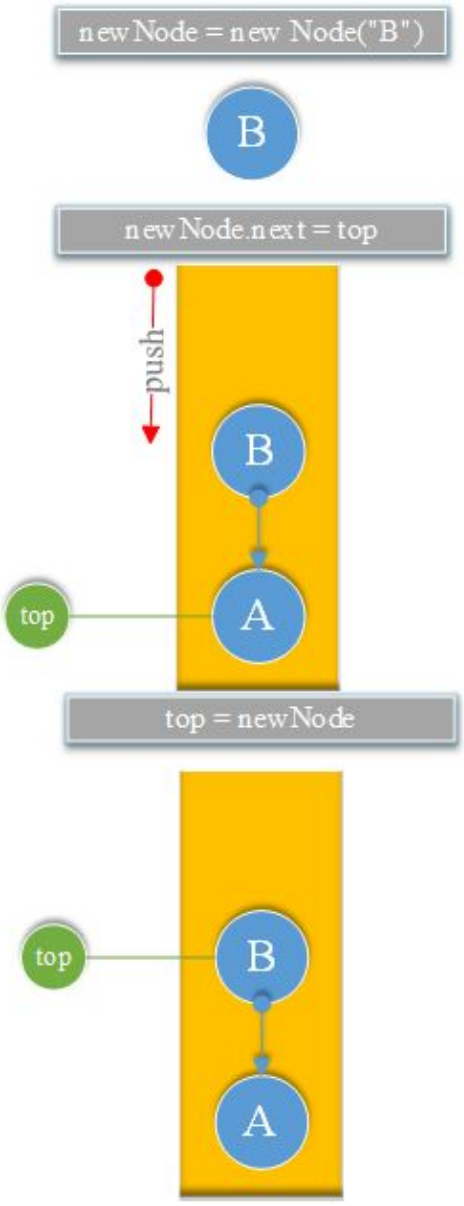

```
class Node{
    constructor(data, next){
        this.data = data;
        this.next = next;
    }
    getData(){
        return this.data;
    }
}
```

1. Inicialização de pilha .

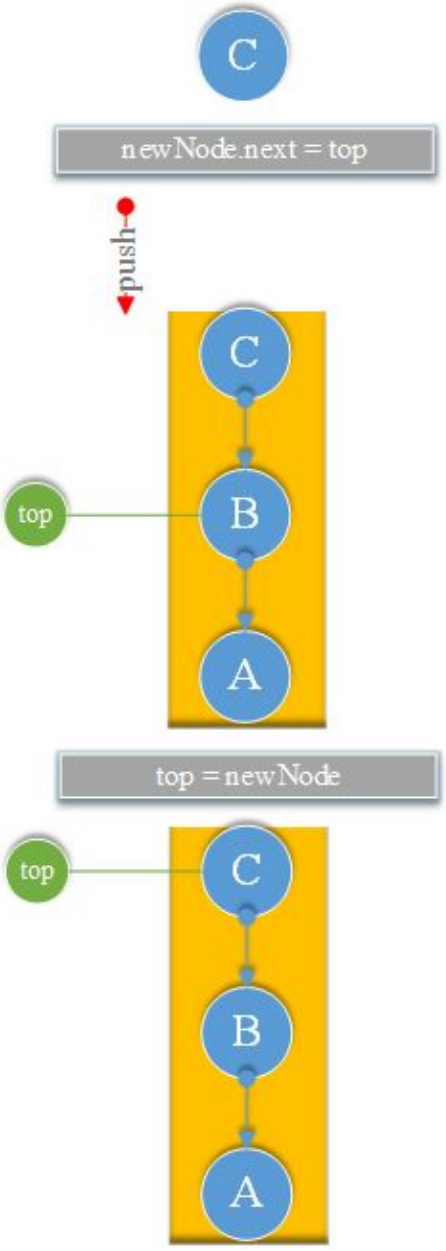
Empurre A para a pilha



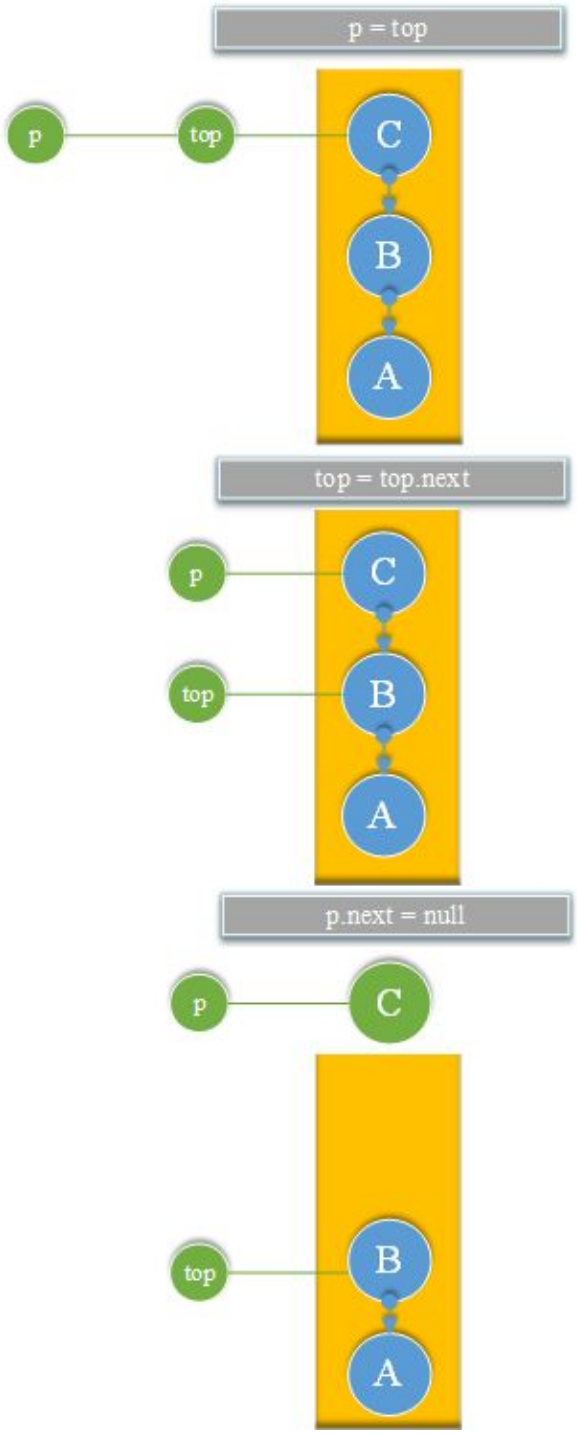
Empurre B para a pilha



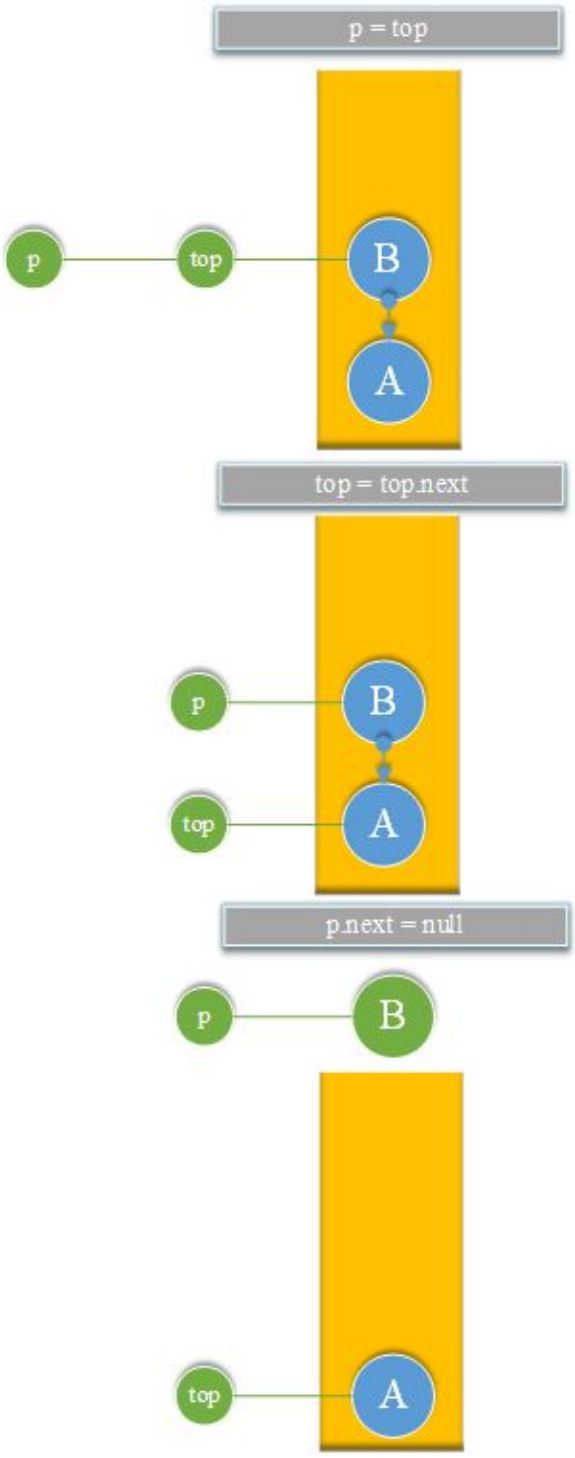
Empurre C para a pilha



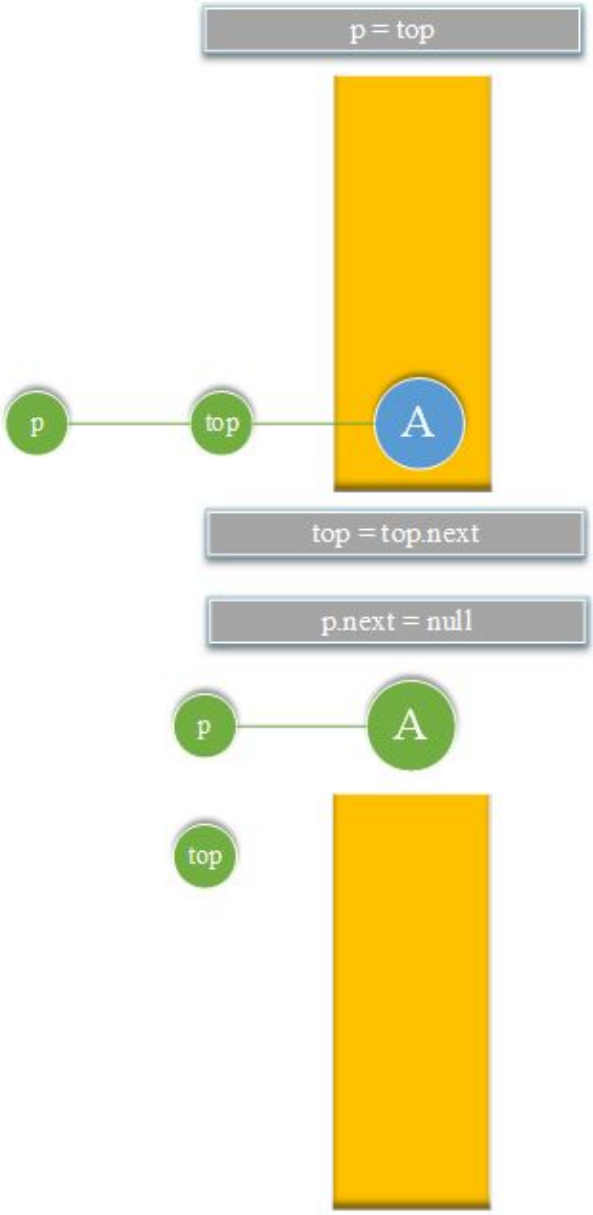
Pop C da pilha:



Pop B da pilha:



Pop A da pilha:



Crie um arquivo: TestStack.html

```
<script type="text/javascript"> ////////////// Node ///////////////
   class Node{
   constructor(data, next){
   this.data = data; this.next = next; }
   getData(){
   return this.data; }
   }

   ////////////// Stack ///////////////
   class Stack {
   constructor(){
   this.top = null; this.size = 0; //  O tamanho da pilha }

   // adicione um elemento ao topo da pilha push(element) {
   if (this.top == null) {
   this.top = new Node(element); } else {
   var newNode = new Node(element); newNode.next = this.top;
this.top = newNode; }
   this.size++; }

   // pegue um elemento e remova-o do topo da pilha pop() {
   if(this.top == null){
   return null; }

   var p = this.top; this.top = this.top.next;// ponteiro superior mover
para baixo p.next = null; this.size--; return p; }

   size() {
   return this.size; }
   }

///////////////// test ////////////////////

   function print(stack) {
   document.write("Top "); var node = null; while ((node =
stack.pop())!=null) {
```

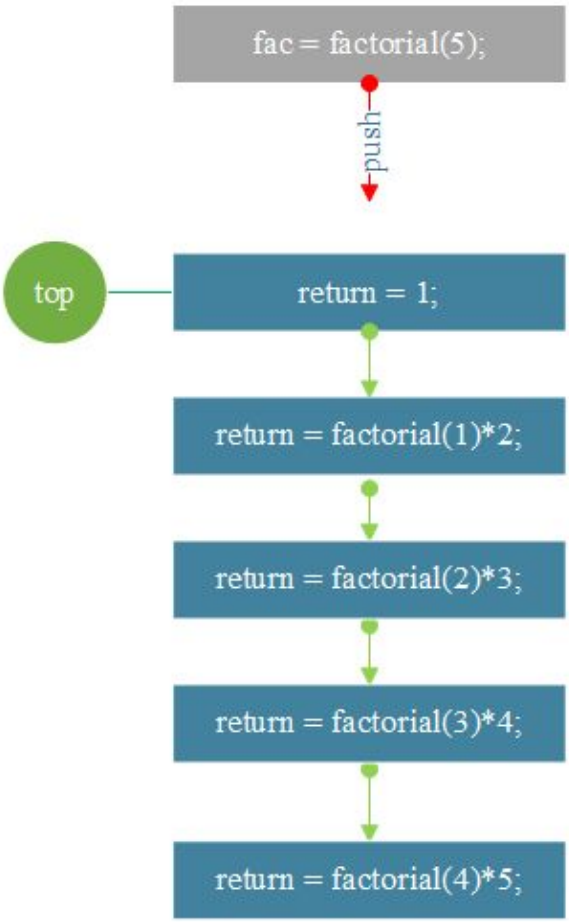
```
    document.write(node.getData() + " -> "); }
    document.write("End <br>"); }

    var stack = new Stack(); stack.push("A"); stack.push("B");
stack.push("C"); stack.push("D"); print(stack);
</script>
```

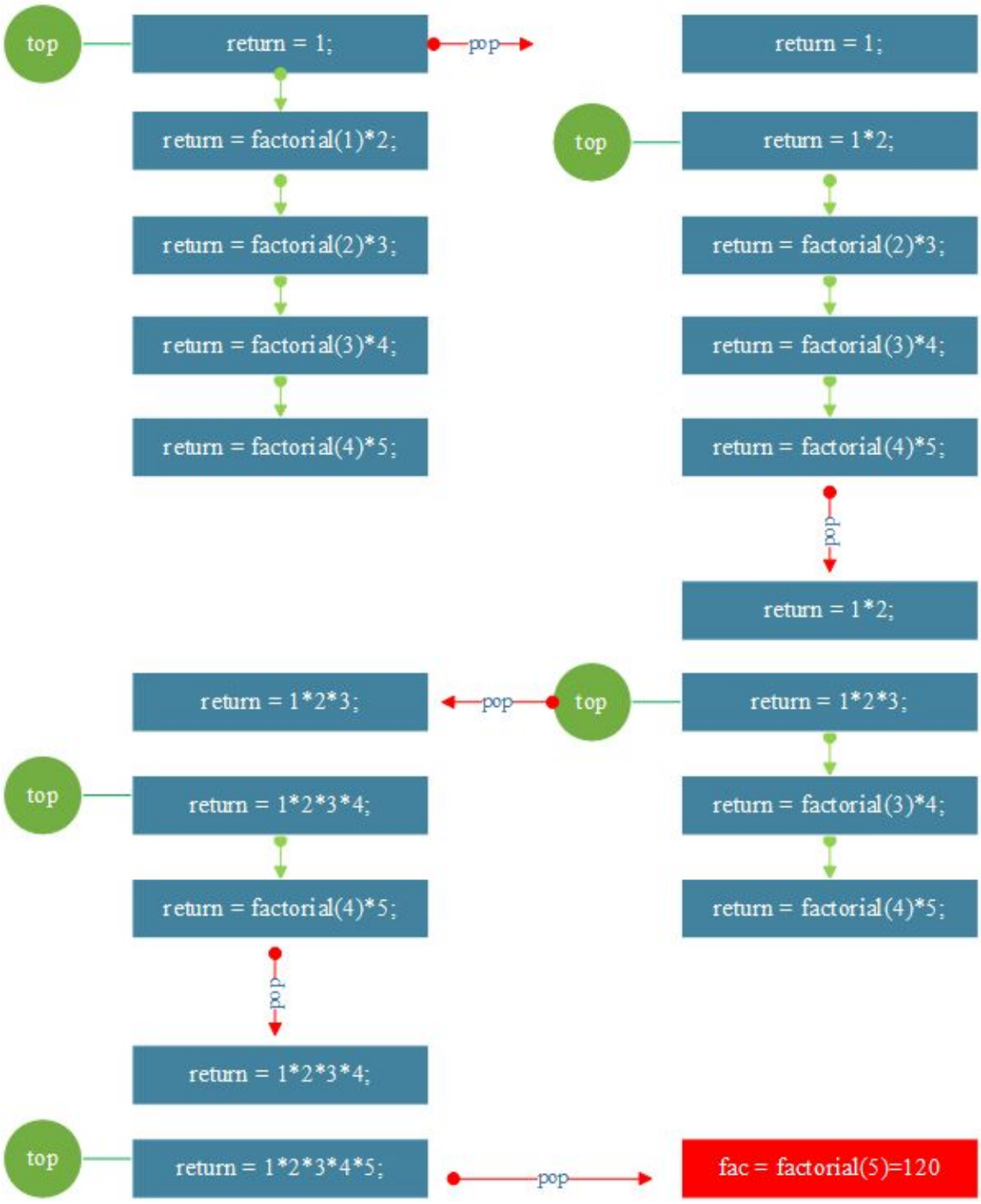
Resultado: Top D -> C -> B -> A -> End

Algoritmo Recursivo

Algoritmo Recursivo (Recursive Algorithm):
A própria função do programa chama o seu próprio progresso até atingir uma determinada condição e passo a passo retorna ao final.

1. Factorial n : n*(n-1)*(n-2) *2*1




top
return = 1;
pop
return = 1;
return = factorial(1)*2;
top
return = 1*2;
return = factorial(2)*3;
return = factorial(2)*3;
return = factorial(3)*4;
return = factorial(3)*4;
return = factorial(4)*5;
return = factorial(4)*5;
pop
return = 1*2;
return = 1*2*3;
pop
top
return = 1*2*3;
top
return = 1*2*3*4;
return = factorial(3)*4;
return = factorial(4)*5;
return = factorial(4)*5;
pop
return = 1*2*3*4;
top
return = 1*2*3*4*5;
pop
fac = factorial(5)=120

Crie um arquivo: TestRecursive.html

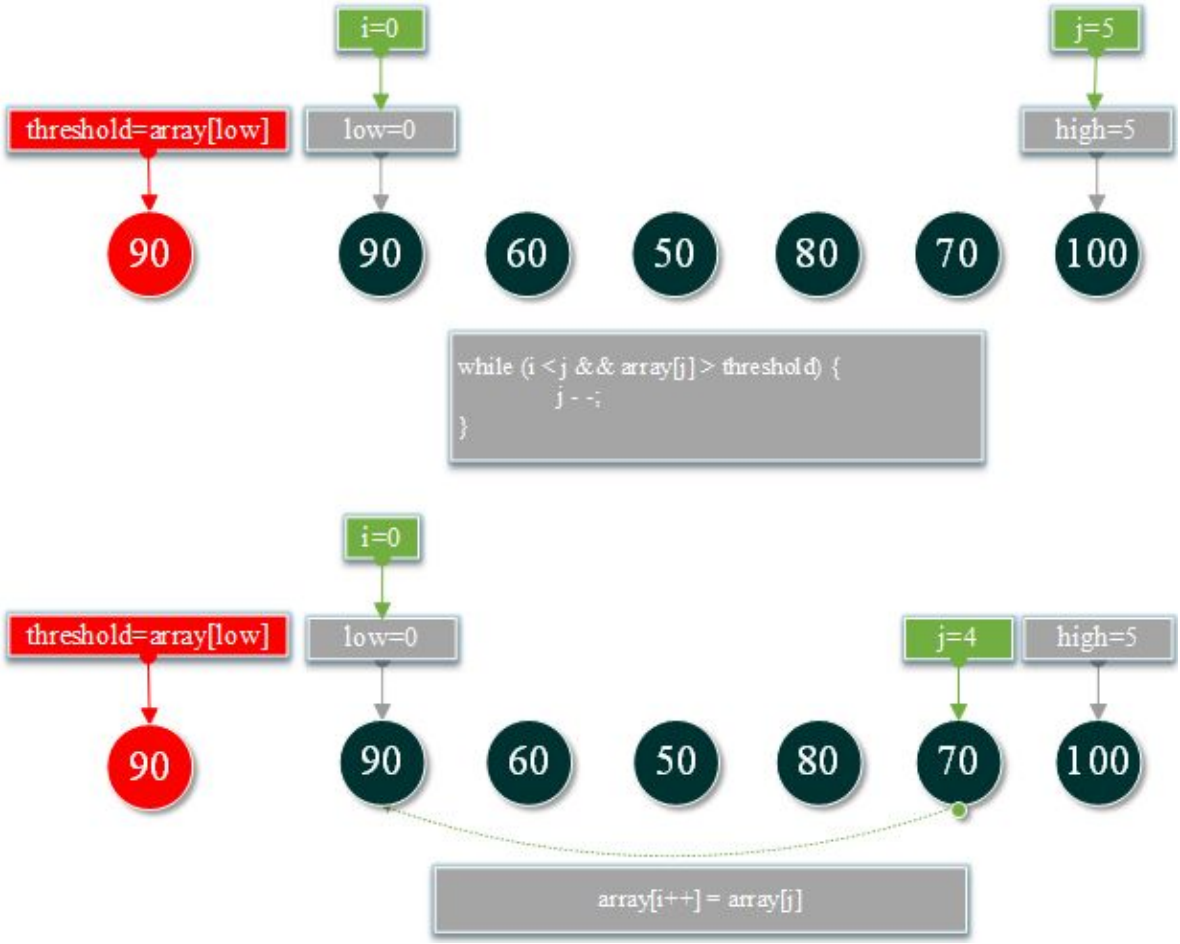
```
<script type="text/javascript"> function factorial(n) {
if (n == 1) {
return 1; } else {
// Invoque-se recursivamente até o final do retorno return factorial(n -
1) * n; }
}

//////////////////////testing//////////////////

var n = 5; var fac = factorial(n); document.write("The factorial of 5 is
:" + fac);
```

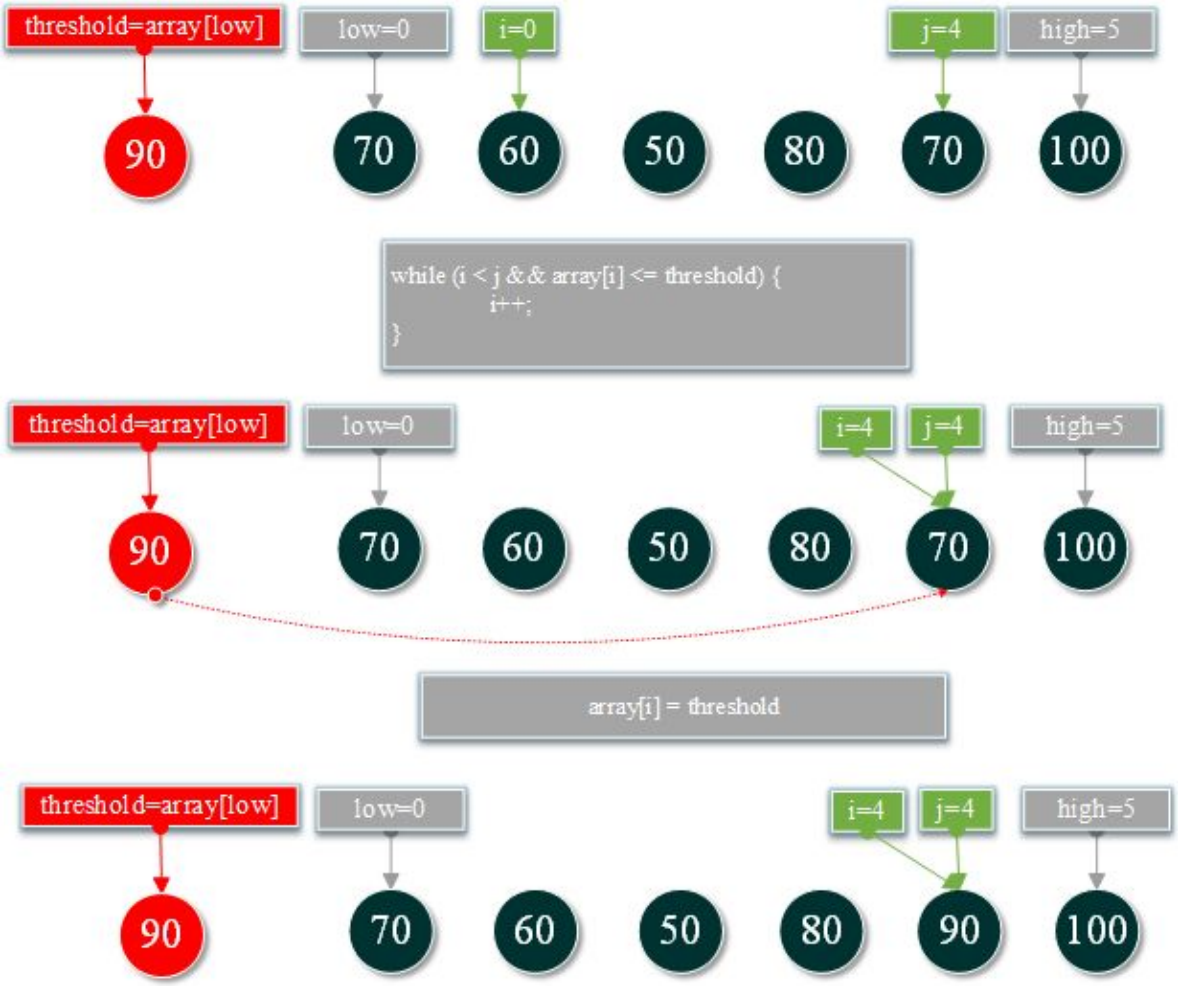
Resultado: The factorial of 5 is :120

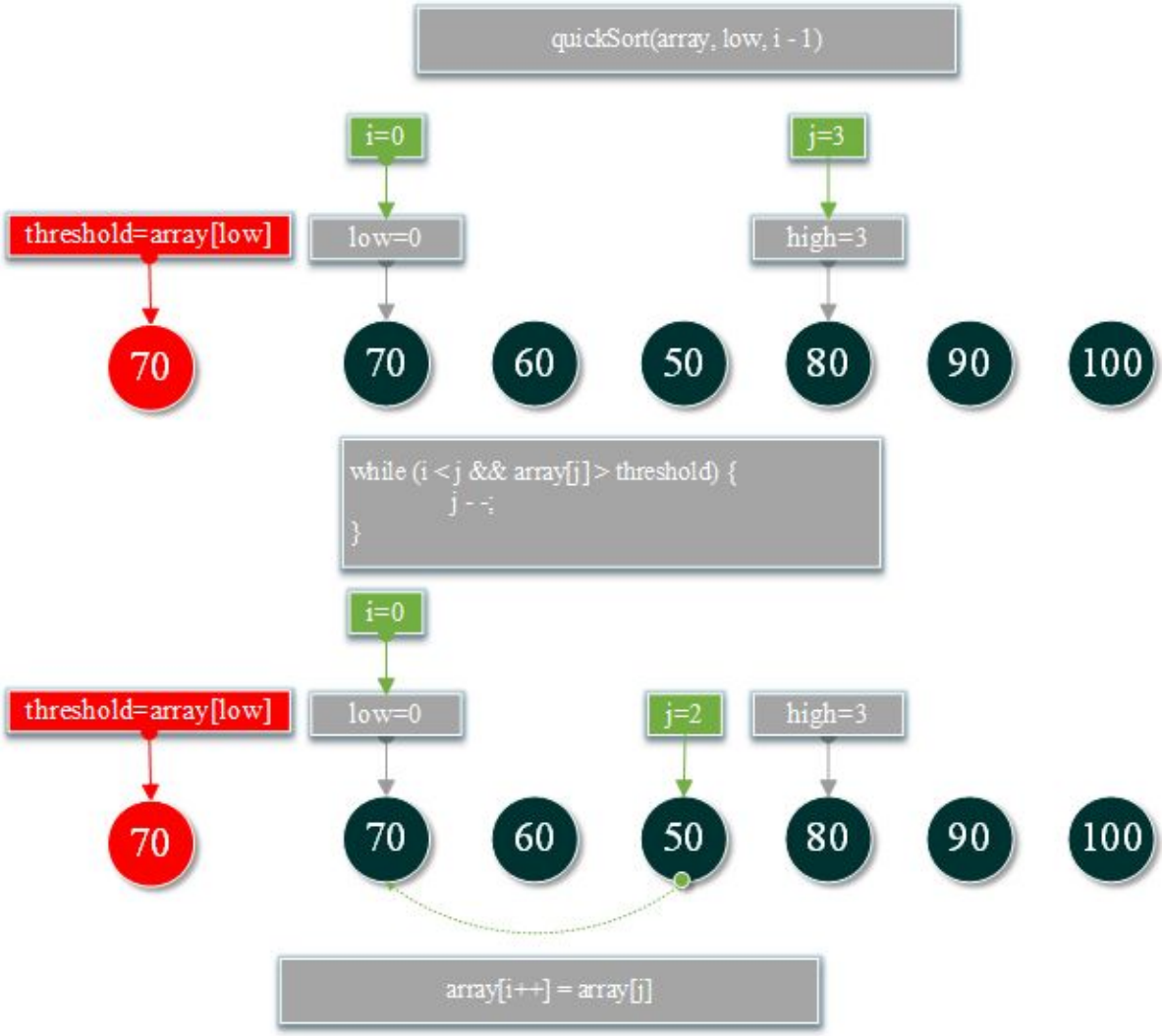
Algoritmo de Ordenar rápida

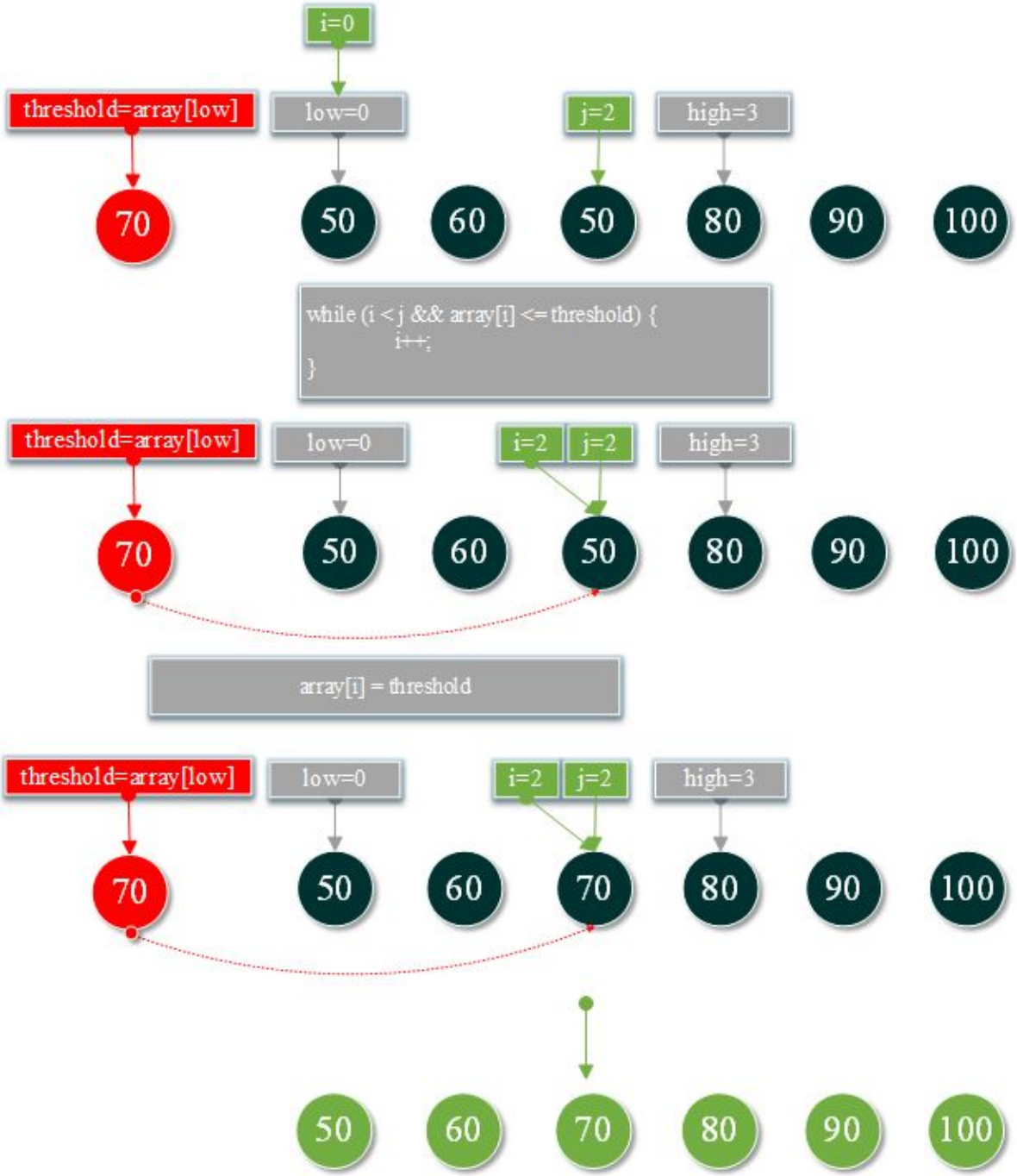
Algoritmo de Ordenar rápida (Quick Sort Algorithm): A Ordenar rápida é um algoritmo de Ordenar popular que, na prática, geralmente é mais rápido em comparação com outros algoritmos de Ordenar. Ele utiliza uma estratégia de dividir e conquistar para classificar rapidamente itens de dados, dividindo uma grande matriz em duas matrizes menores.

1. Classifique os seguintes dados usando a Ordenar rápida {90, 60, 50, 80, 70, 100}




threshold=array[low]
low=0
i=0
j=4
high=5
90
70
60
50
80
70
100
while (i < j && array[i] <= threshold) {
i++;
}
threshold=array[low]
low=0
i=4
j=4
high=5
90
70
60
50
80
70
100
array[i] = threshold
threshold=array[low]
low=0
i=4
j=4
high=5
90
70
60
50
80
90
100


quickSort(array, low, i - 1)
i=0
j=3
threshold=array[low]
low=0
high=3
70
70
60
50
80
90
100
while (i < j && array[j] > threshold) {
j - -;
}
i=0
threshold=array[low]
low=0
j=2
high=3
70
70
60
50
80
90
100
array[i++] = array[j]


i=0
threshold=array[low]
low=0
j=2
high=3
70
50
60
50
80
90
100
while (i < j && array[i] <= threshold) {
i++;
}
threshold=array[low]
low=0
i=2
j=2
high=3
70
50
60
50
80
90
100
array[i] = threshold
threshold=array[low]
low=0
i=2
j=2
high=3
70
50
60
70
80
90
100
50
60
70
80
90
100

Crie um arquivo: TestQuickSort.html

```
<script type="text/javascript"> class QuickSort{
   sort(array) {
   if (array.length > 0) {
   this.quickSort(array, 0, array.length - 1); }
   }

   quickSort(array, low, high) {
   if (low > high) {
   return; }
   var i = low; var j = high; var threshold = array[low]; // Digitalizados alternadamente nos dois extremos da lista while (i < j) {
   // Encontre a primeira posição menor que threshold da direita para a esquerda while (i < j && array[j] > threshold) {
   j--; }
   // Substitua o low por um número menor que o threshold if (i < j) array[i++] = array[j];
   // Encontre a primeira posição maior que threshold da esquerda para a direita while (i < j && array[i] <= threshold) {
   i++; }

   // Substitua o high por um número maior que o threshold if (i < j) array[j--] = array[i]; }
   array[i] = threshold;
   this.quickSort(array, low, i - 1); // esquerda rápida Classificar this.quickSort(array, i + 1, high); // Ordenar rápida à direita }
   }


   //////////////////////testing//////////////////////

   var scores = [ 90, 60, 50, 80, 70, 100 ];
   var quickSort = new QuickSort();
   quickSort.sort(scores);
   for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ","); }
```

```
</script>
```

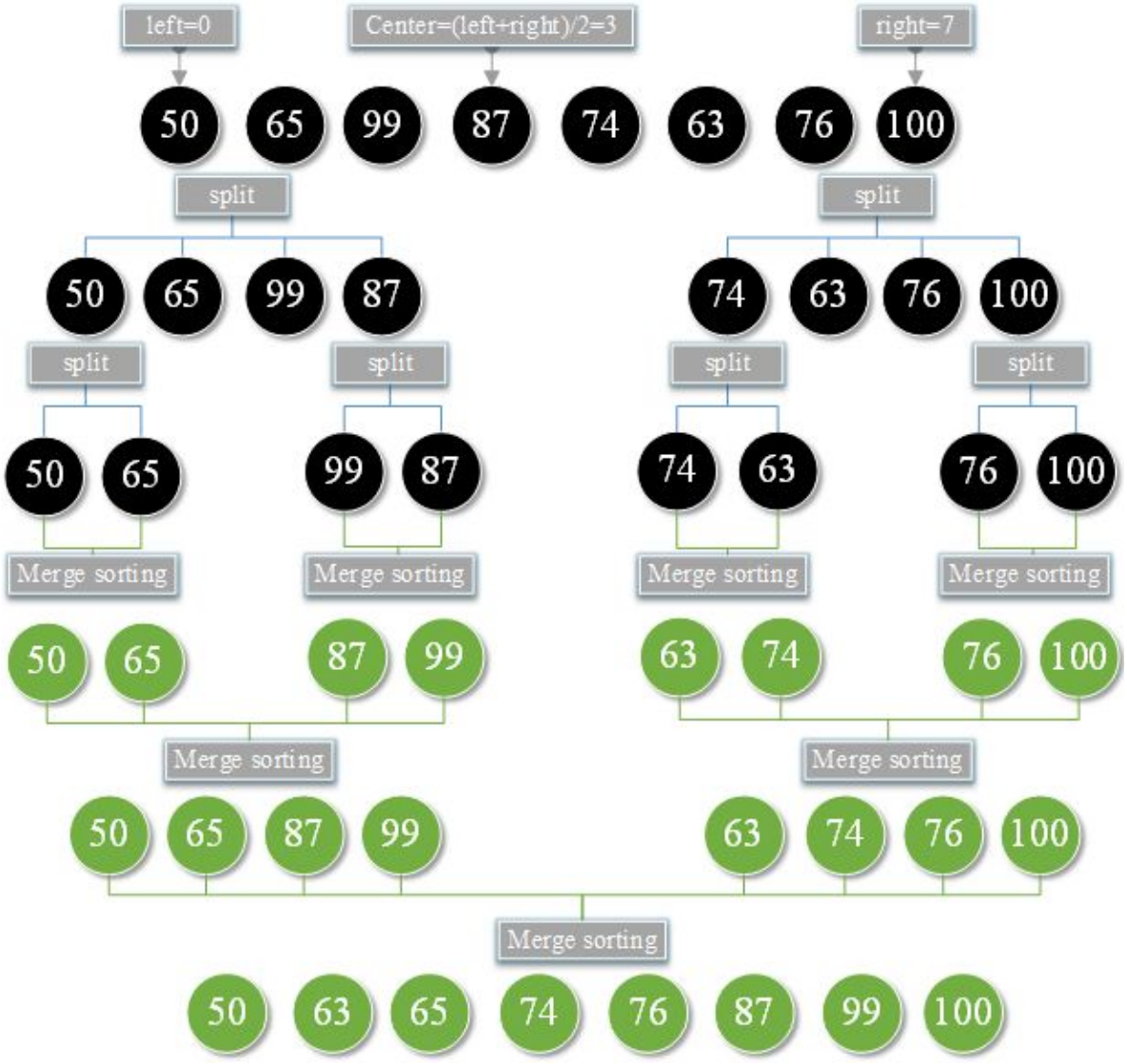
Resultado:

50,60,70,80,90,100

Algoritmo de mesclagem bidirecional

Algoritmo de mesclagem bidirecional (Two-way Merge Algorithm): Os dados da primeira metade e da segunda metade são classificados e as duas sub-listas ordenadas são mescladas em uma lista ordenada, que continua recursiva até o final.

1. Classifique os seguintes dados usando a Ordenar de mesclagem
{50, 65, 99, 87, 74, 63, 76, 100}



Crie um arquivo: TestMergeSort.html

```
<script type="text/javascript"> class MergeSort{
   sort(array) {
   var temp = new Array(array.length); this.mergeSort(array, temp, 0,
array.length - 1); }

   mergeSort(array, temp, left, right) {
   if (left < right) {
   var center = parseInt((left + right) / 2); this.mergeSort(array, temp, left,
center); // Classificar mesclagem à esquerda this.mergeSort(array, temp,
center + 1, right); // Classificar mesclagem à direita this.merge(array,
temp, left, center + 1, right); // Mesclar duas matrizes ordenadas }
   }

   // Combine duas listas ordenadas em uma lista ordenada merge(array,
temp, left, right, rightEndIndex) {
   var leftEndIndex = right - 1; var tempIndex = left; var elementNumber
= rightEndIndex - left + 1;
   while (left <= leftEndIndex && right <= rightEndIndex) {
   if (array[left] <= array[right]) temp[tempIndex++] = array[left++]; else
temp[tempIndex++] = array[right++]; }

   while (left <= leftEndIndex) { // Se houver um elemento à esquerda
temp[tempIndex++] = array[left++]; }

   while (right <= rightEndIndex) { // Se houver elemento à direita
temp[tempIndex++] = array[right++]; }

   for (var i = 0; i < elementNumber; i++) {// Copiar temp para array
array[rightEndIndex] = temp[rightEndIndex]; rightEndIndex--; }
   }
   }

   //////////////////////testing//////////////////

   var scores = [ 50, 65, 99, 87, 74, 63, 76, 100, 92 ];
   var mergeSort = new MergeSort();
```

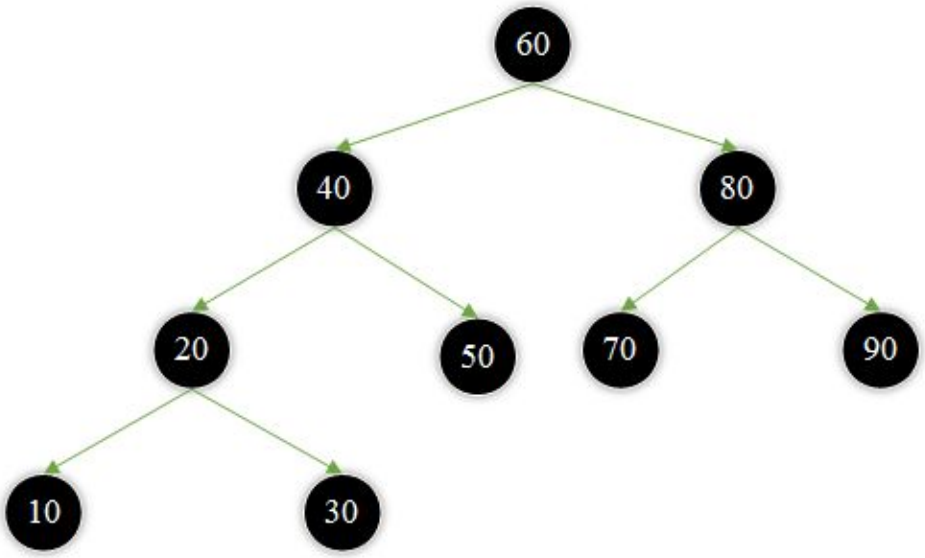
```
    mergeSort.sort(scores);
    for (var i = 0; i < scores.length; i++) {
    document.write(scores[i] + ","); }

</script>
```

Resultado:

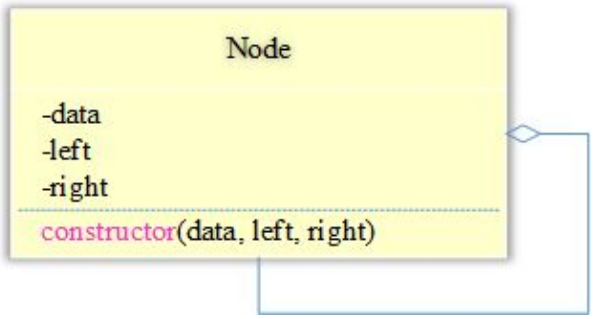
50,63,65,74,76,87,99,100

Árvore de pesquisa binária

Árvore de pesquisa binária (Binary Search Tree): 1. Se a subárvore esquerda de qualquer nó não estiver vazia, o valor de todos os nós na subárvore esquerda será menor que o valor do nó raiz; 2. Se a subárvore direita de qualquer nó não estiver vazia, o valor de todos os nós na subárvore direita será maior que o valor do nó raiz; 3. A subárvore esquerda e a subárvore direita de qualquer nó também são árvores de pesquisa binária.



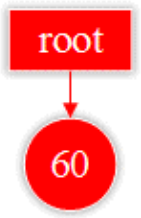
UML Diagrama



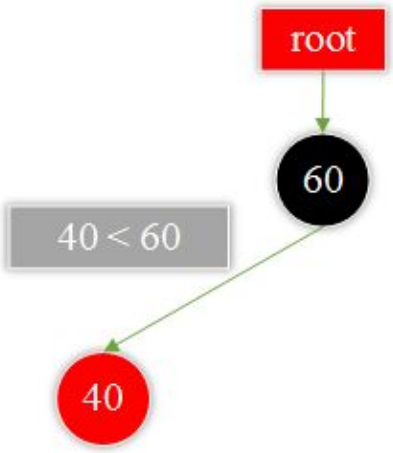

```
class Node{
    constructor(data, left, right){
        this.data = data;
        this.left = left;
        this.right = right;
    }

    getData(){
        return this.data;
    }
}
```

1. **Construir uma árvore de pesquisa binária** Os nós inseridos são comparados a partir do nó raiz e o menor que o nó raiz é comparado com a subárvore esquerda do nó raiz, caso contrário, comparado com a subárvore direita até que a subárvore esquerda esteja vazia ou a subárvore direita esteja vazia, será inserido.

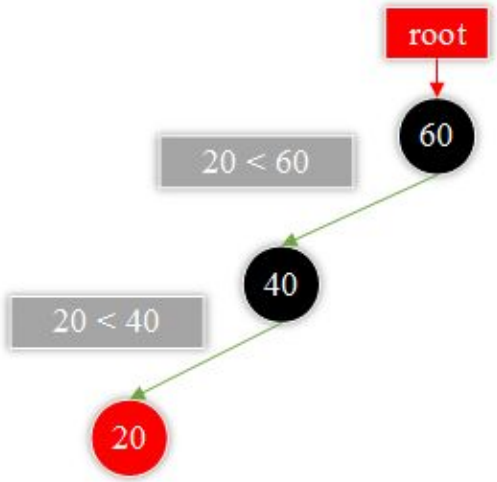
Inserir 60



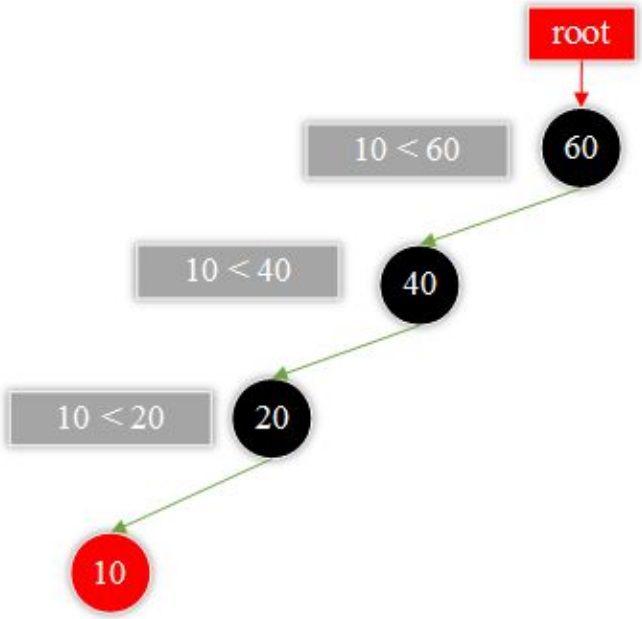
Inserir 40



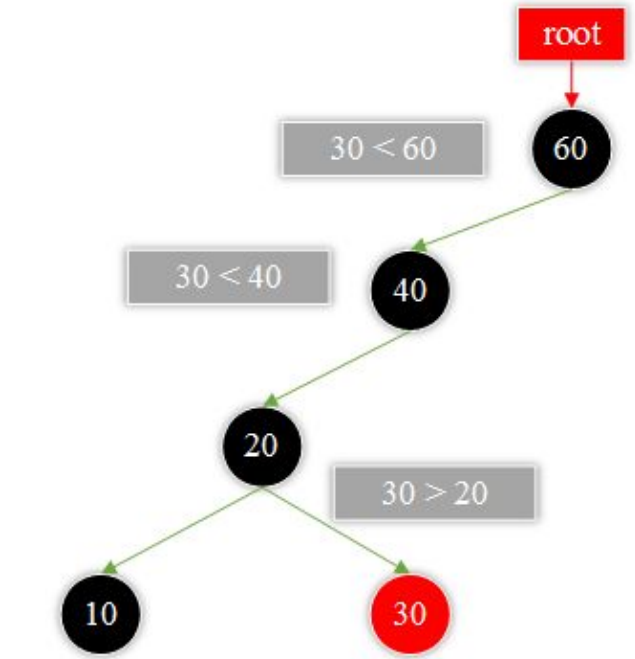
Inserir 20



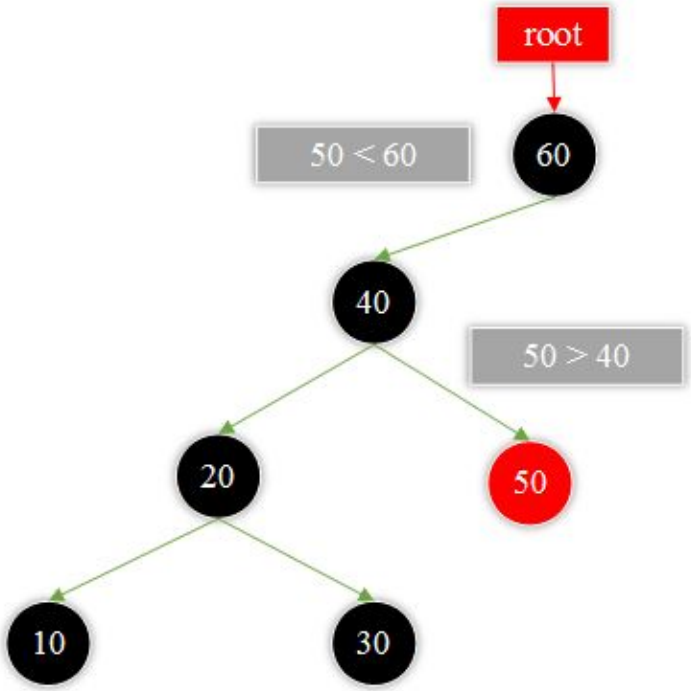
Inserir 10



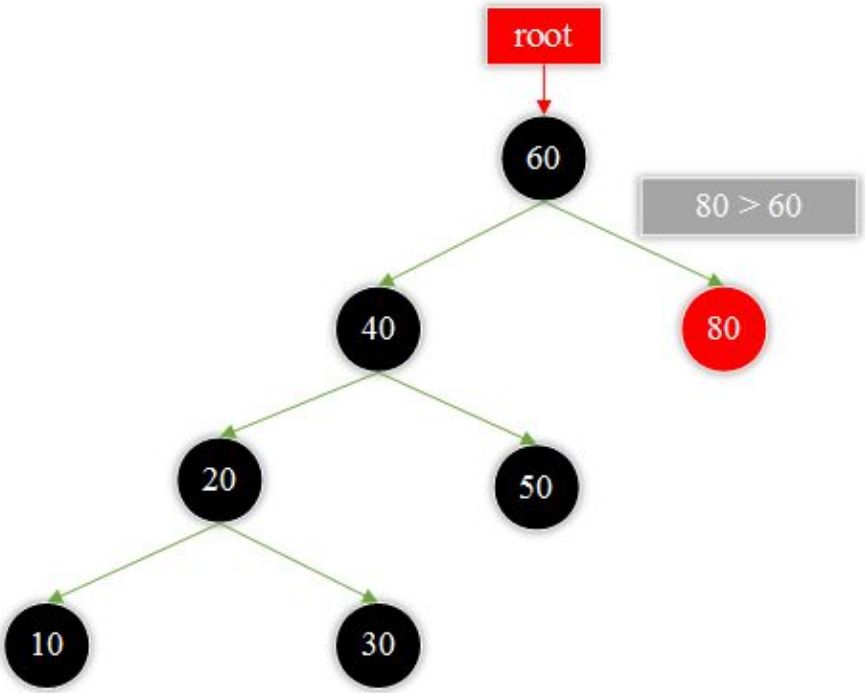
Inserir 30



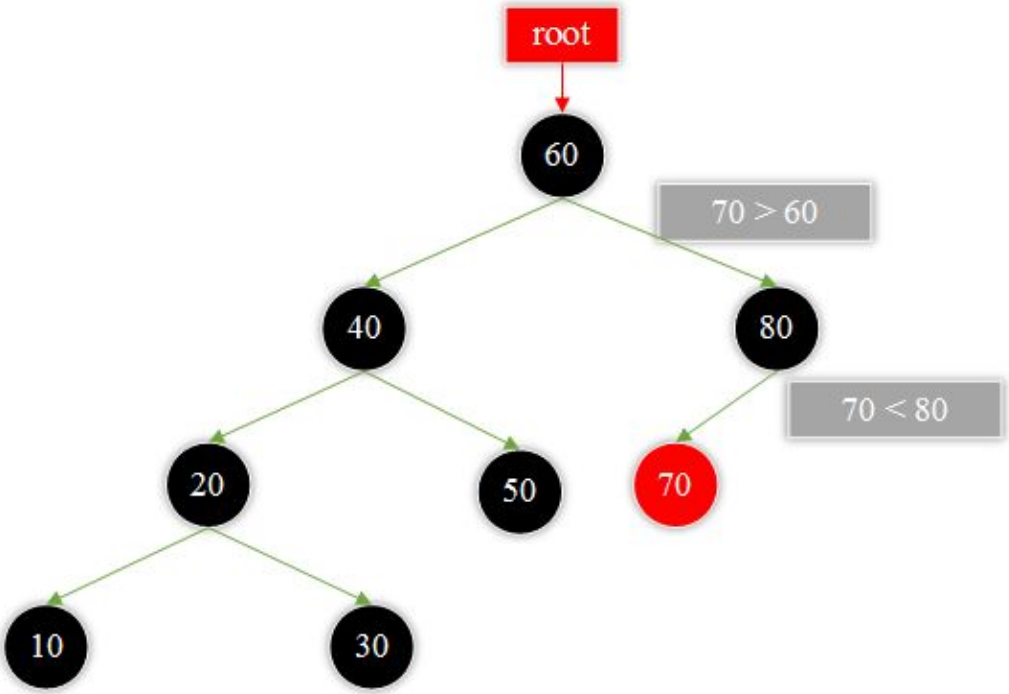
Inserir 50



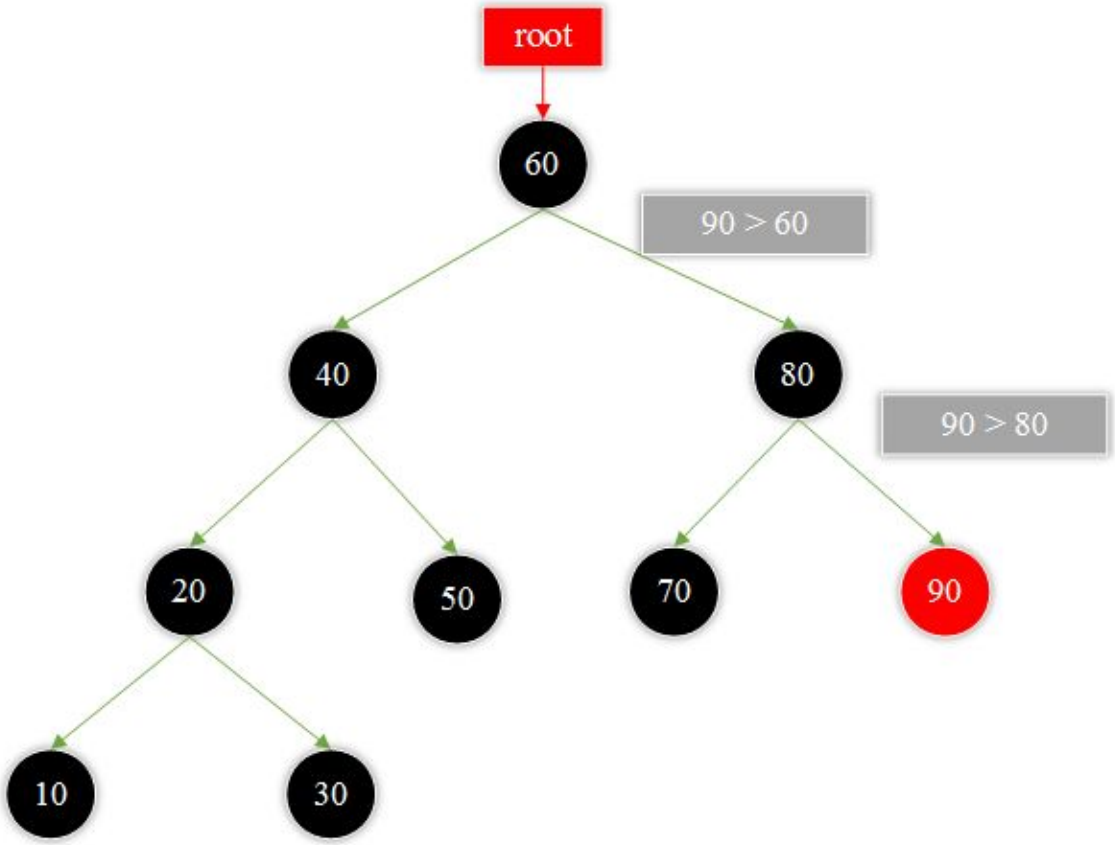
Inserir 80



Inserir 70

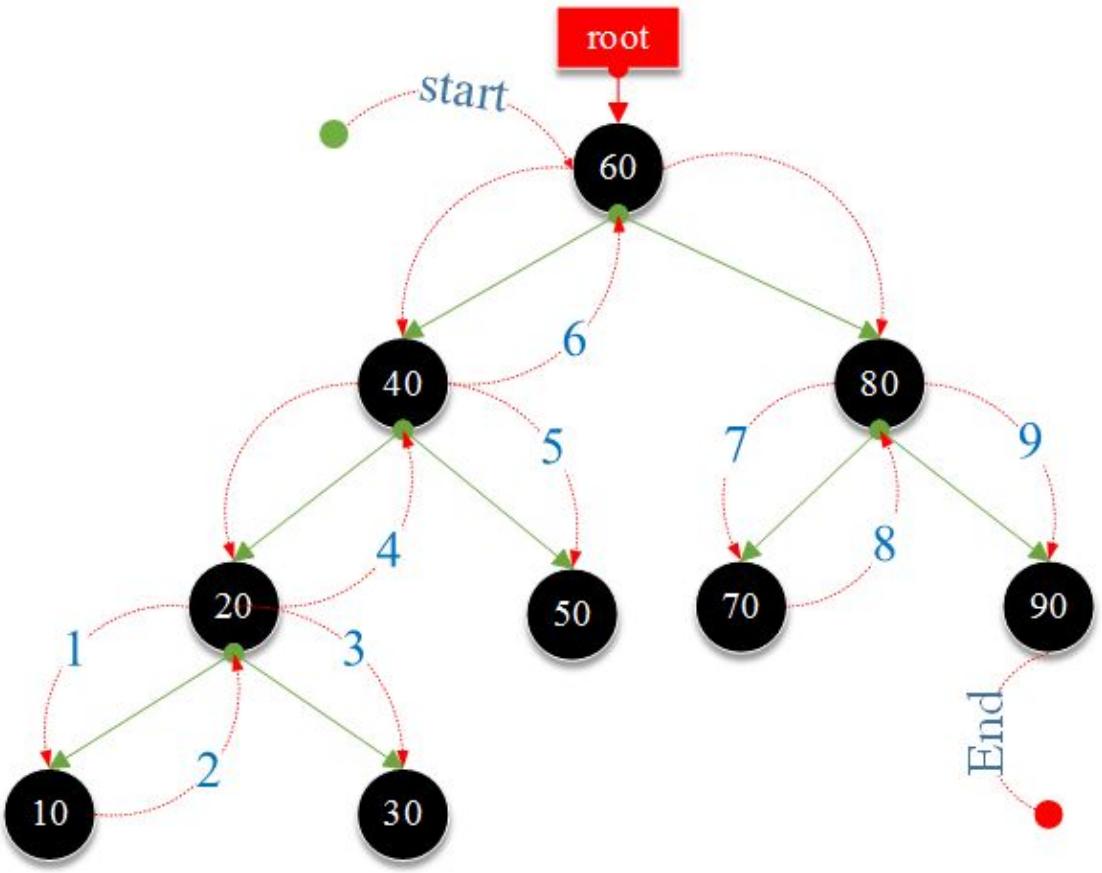


Inserir 90

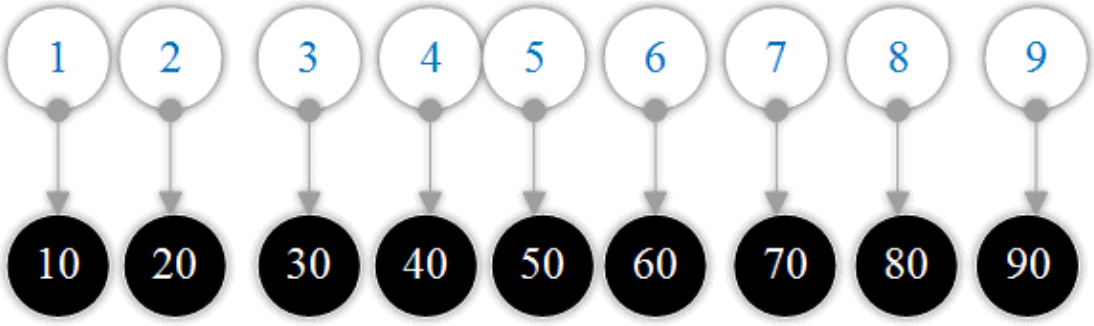


2. á rvore de pesquisa binária In-order traversal

In-order traversal : left subtree -> root node -> right subtree



Resultado:



Crie um arquivo: TestBinarySearchTreeInOrder.html

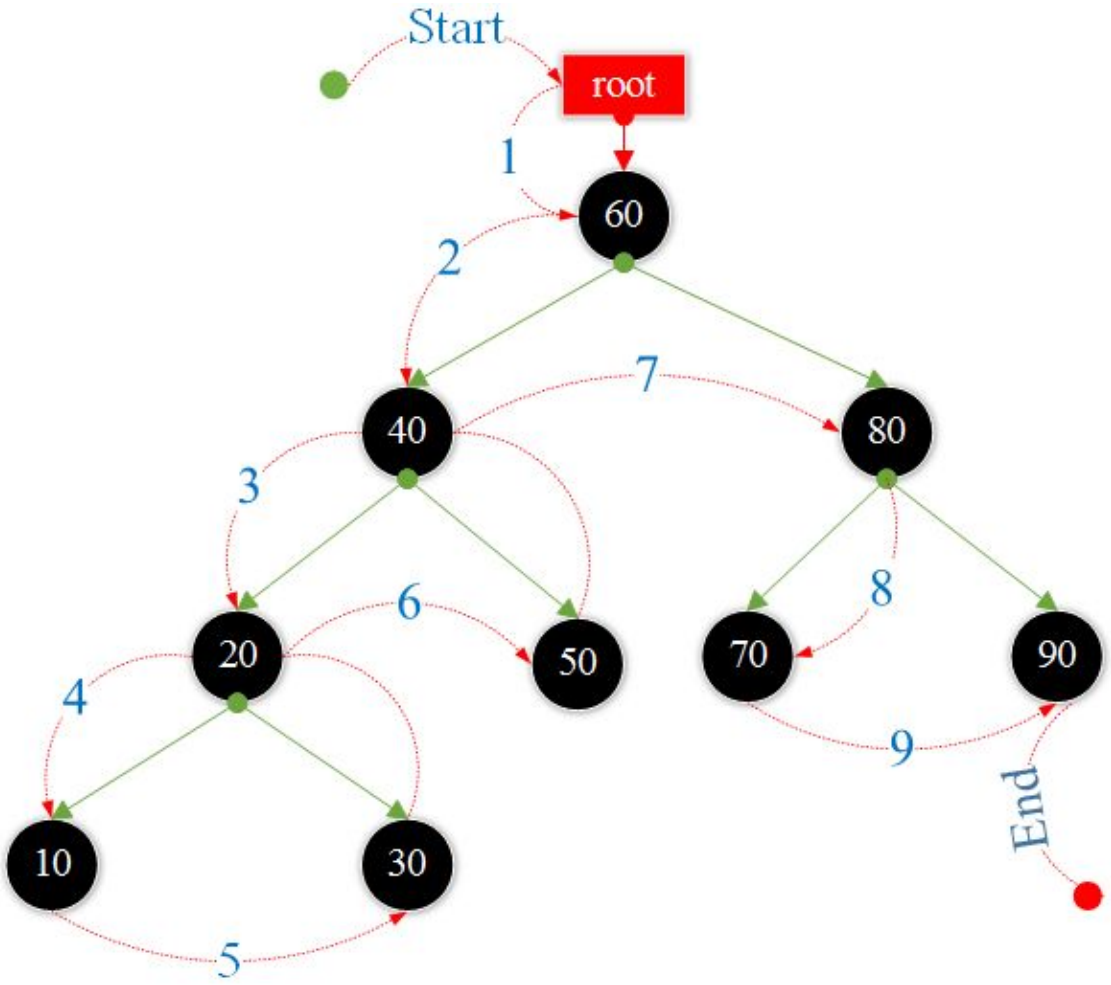
```
<script type="text/javascript"> class BinaryTree {
   getRoot() {
   return this.root; }

   inOrder(root) {
   if (root == null) {
   return; }
   this.inOrder(root.left); // Recursivamente Percorrendo a subárvore
esquerda document.write(root.getData() + ", "); this.inOrder(root.right); //
Recursivamente Percorrendo a subárvore direita }

   insert(node, newData) {
   if (this.root == null) {
   this.root = new Node(newData, null, null); return; }

   var compareValue = newData - node.getData();
   // Subárvore esquerda recursiva, continue a encontrar a posição de
inserção if (compareValue < 0) {
   if (node.left == null) {
   node.left = new Node(newData, null, null); } else {
   this.insert(node.left, newData); }
   } else if (compareValue > 0) {// Subárvore direita recursiva para
encontrar a posição de inserção if (node.right == null) {
   node.right = new Node(newData, null, null); } else {
   this.insert(node.right, newData); }
   }
   }
   }


   class Node{
   constructor(data, left, right){
   this.data = data; this.left = left; this.right = right; }

   getData(){
```

```
    return this.data; }
    }

    //////////////////////testing//////////////////

    var binaryTree=new BinaryTree(); // Construindo uma árvore de
pesquisa binária binaryTree.insert(binaryTree.getRoot(), 60);
binaryTree.insert(binaryTree.getRoot(), 40);
binaryTree.insert(binaryTree.getRoot(), 20);
binaryTree.insert(binaryTree.getRoot(), 10);
binaryTree.insert(binaryTree.getRoot(), 30);
binaryTree.insert(binaryTree.getRoot(), 50);
binaryTree.insert(binaryTree.getRoot(), 80);
binaryTree.insert(binaryTree.getRoot(), 70);
binaryTree.insert(binaryTree.getRoot(), 90); document.write("<br> In-
order traversal binary search tree <br>");
binaryTree.inOrder(binaryTree.getRoot()); </script>
```

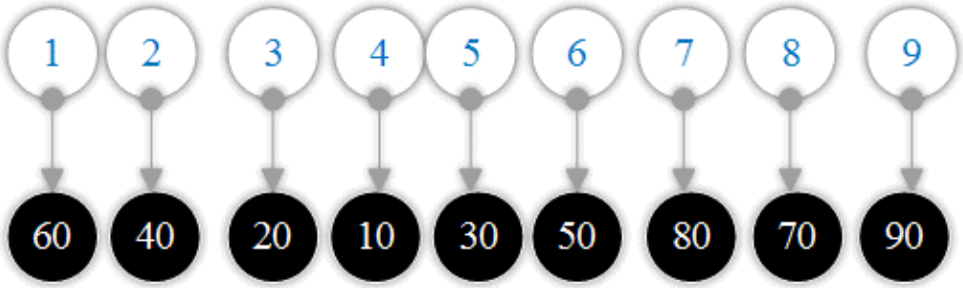
Resultado: In-order traversal binary search tree 10, 20, 30, 40, 50, 60, 70, 80, 90,

3. árvore de pesquisa binária Pre-order traversal

Pre-order traversal : root node -> left subtree -> right subtree



Resultado:



Crie um arquivo: TestBinarySearchTreePreOrder.html

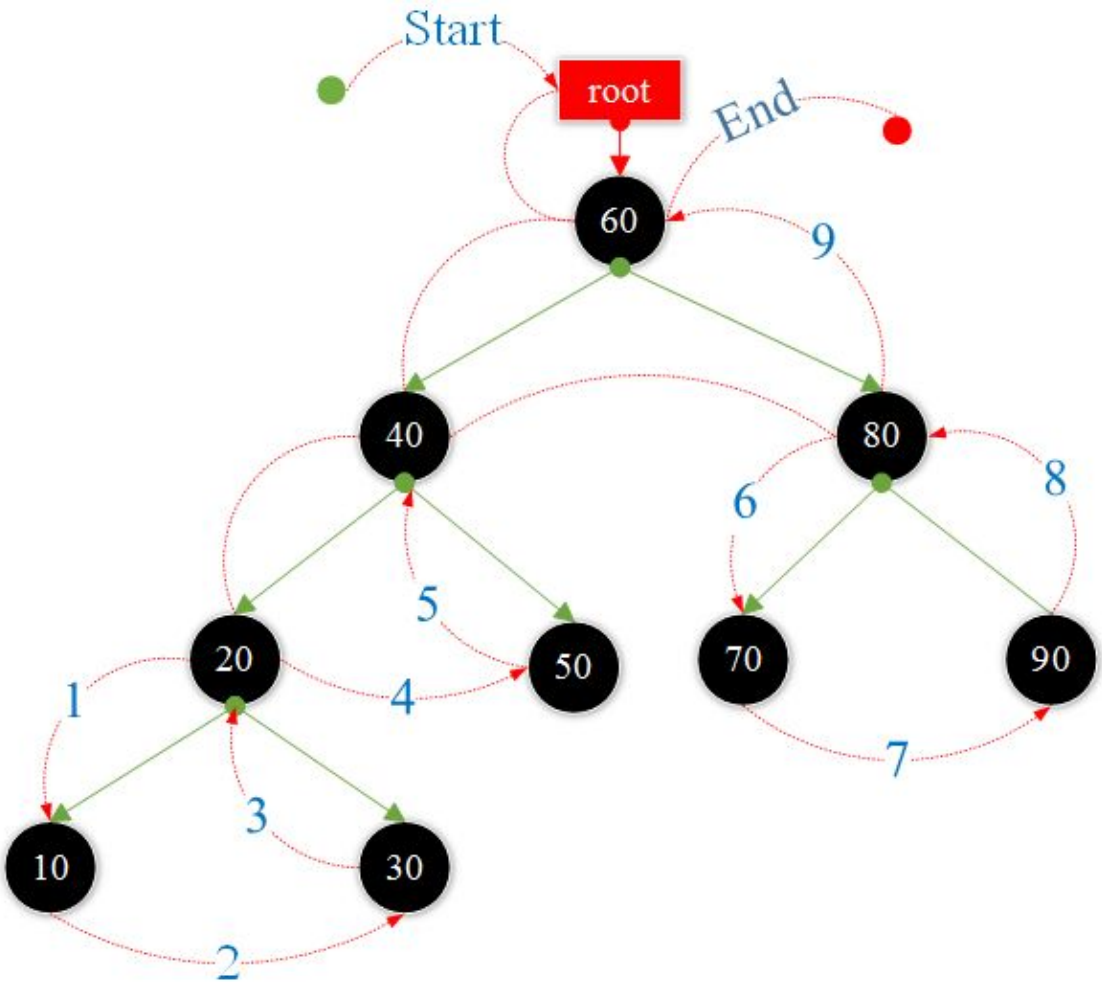
```
<script type="text/javascript"> class BinaryTree {
   getRoot() {
   return this.root; }

   preOrder(root) {
   if (root == null) {
   return; }
   document.write(root.getData() + ", "); this.preOrder(root.left); //
Recursivamente Percorrendo a subárvore esquerda
this.preOrder(root.right); // Recursivamente Percorrendo a subárvore
direita }

   insert(node, newData) {
   if (this.root == null) {
   this.root = new Node(newData, null, null); return; }

   var compareValue = newData - node.getData(); // Subárvore esquerda
recursiva, continue a encontrar a posição de inserção if (compareValue <
0) {
   if (node.left == null) {
   node.left = new Node(newData, null, null); } else {
   this.insert(node.left, newData); }
   } else if (compareValue > 0) {// Subárvore direita recursiva para
encontrar a posição de inserção if (node.right == null) {
   node.right = new Node(newData, null, null); } else {
   this.insert(node.right, newData); }
   }
   }
   }


   class Node{
   constructor(data, left, right){
   this.data = data; this.left = left; this.right = right; }
```

```
   getData(){
   return this.data; }
   }

   //////////////////////testing//////////////////

   var binaryTree=new BinaryTree(); // Construindo uma árvore de
pesquisa binária binaryTree.insert(binaryTree.getRoot(), 60);
binaryTree.insert(binaryTree.getRoot(), 40);
binaryTree.insert(binaryTree.getRoot(), 20);
binaryTree.insert(binaryTree.getRoot(), 10);
binaryTree.insert(binaryTree.getRoot(), 30);
binaryTree.insert(binaryTree.getRoot(), 50);
binaryTree.insert(binaryTree.getRoot(), 80);
binaryTree.insert(binaryTree.getRoot(), 70);
binaryTree.insert(binaryTree.getRoot(), 90);
   document.write("<br> Pre-order traversal binary search tree <br>");
binaryTree.preOrder(binaryTree.getRoot()); </script>
```

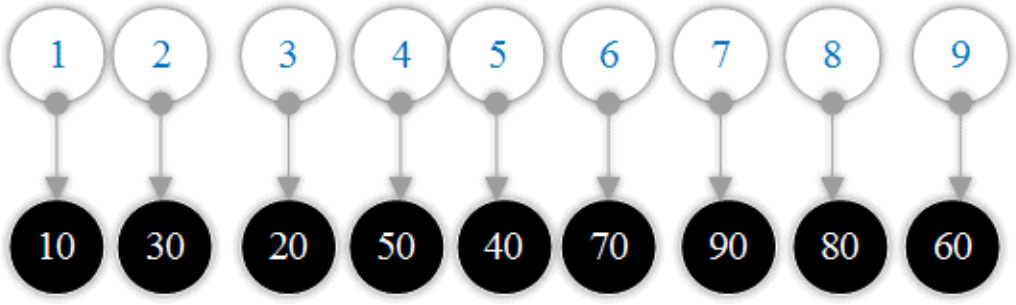
Resultado: Pre-order traversal binary search tree 60, 40, 20, 10, 30, 50, 80, 70, 90,

4. á rvore de pesquisa binária Post-order traversal

Post-order traversal : right subtree -> root node -> left subtree



Resultado:



Crie um arquivo: TestBinarySearchTreePostOrder.html

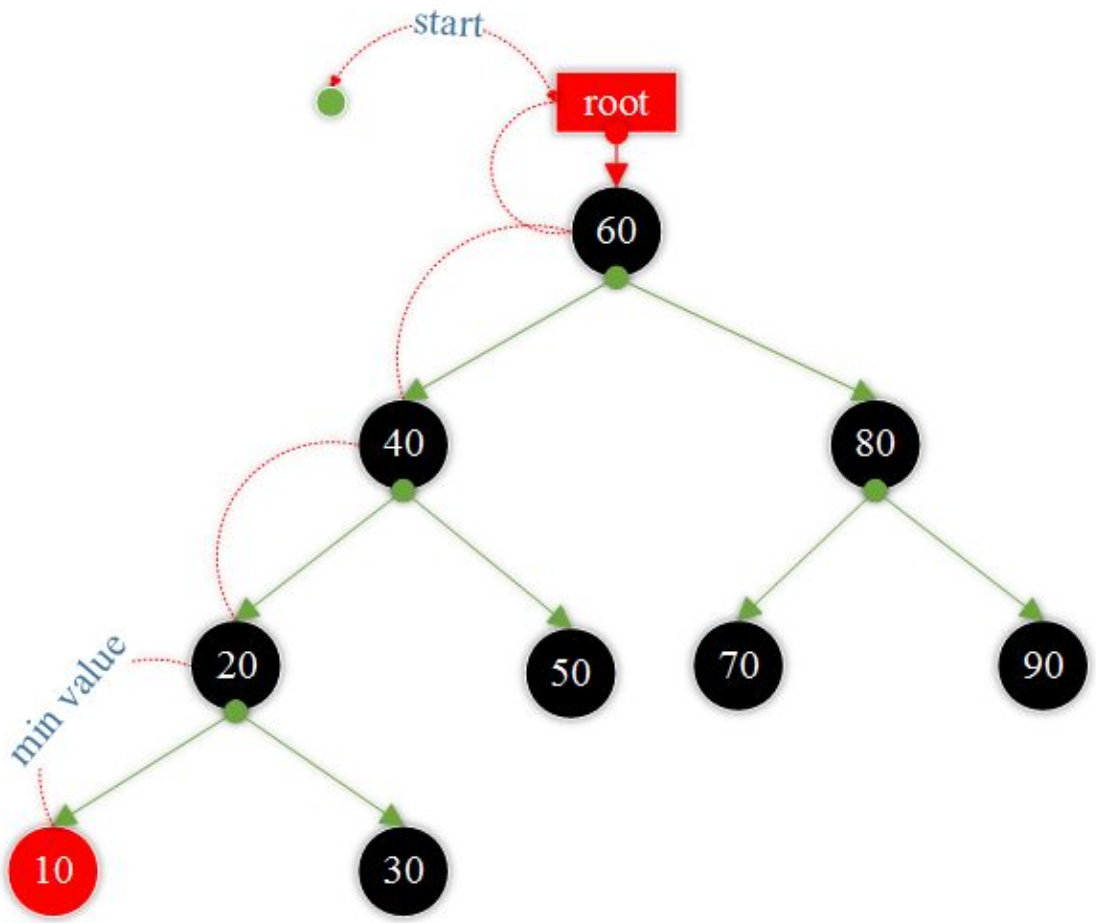
```
<script type="text/javascript"> class BinaryTree {
   getRoot() {
   return this.root; }

   postOrder(root) {
   if (root == null) {
   return; }
   this.postOrder(root.left); // Recursivamente Percorrendo a subárvore
esquerda this.postOrder(root.right); // Recursivamente Percorrendo a
subárvore direita document.write(root.getData() + ", "); }

   insert(node, newData) {
   if (this.root == null) {
   this.root = new Node(newData, null, null); return; }

   var compareValue = newData - node.getData(); // Subárvore esquerda
recursiva, continue a encontrar a posição de inserção if (compareValue <
0) {
   if (node.left == null) {
   node.left = new Node(newData, null, null); } else {
   this.insert(node.left, newData); }
   } else if (compareValue > 0) {// Subárvore direita recursiva para
encontrar a posição de inserção if (node.right == null) {
   node.right = new Node(newData, null, null); } else {
   this.insert(node.right, newData); }
   }
   }
   }


   class Node{
   constructor(data, left, right){
   this.data = data; this.left = left; this.right = right; }

   getData(){
```

```
   return this.data; }
   }

   //////////////////////testing//////////////////

   var binaryTree=new BinaryTree();
   // Constructing a binary search tree
binaryTree.insert(binaryTree.getRoot(), 60);
binaryTree.insert(binaryTree.getRoot(), 40);
binaryTree.insert(binaryTree.getRoot(), 20);
binaryTree.insert(binaryTree.getRoot(), 10);
binaryTree.insert(binaryTree.getRoot(), 30);
binaryTree.insert(binaryTree.getRoot(), 50);
binaryTree.insert(binaryTree.getRoot(), 80);
binaryTree.insert(binaryTree.getRoot(), 70);
binaryTree.insert(binaryTree.getRoot(), 90);
   document.write("<br> Post-order traversal binary search tree <br>");
binaryTree.postOrder(binaryTree.getRoot()); </script>
```

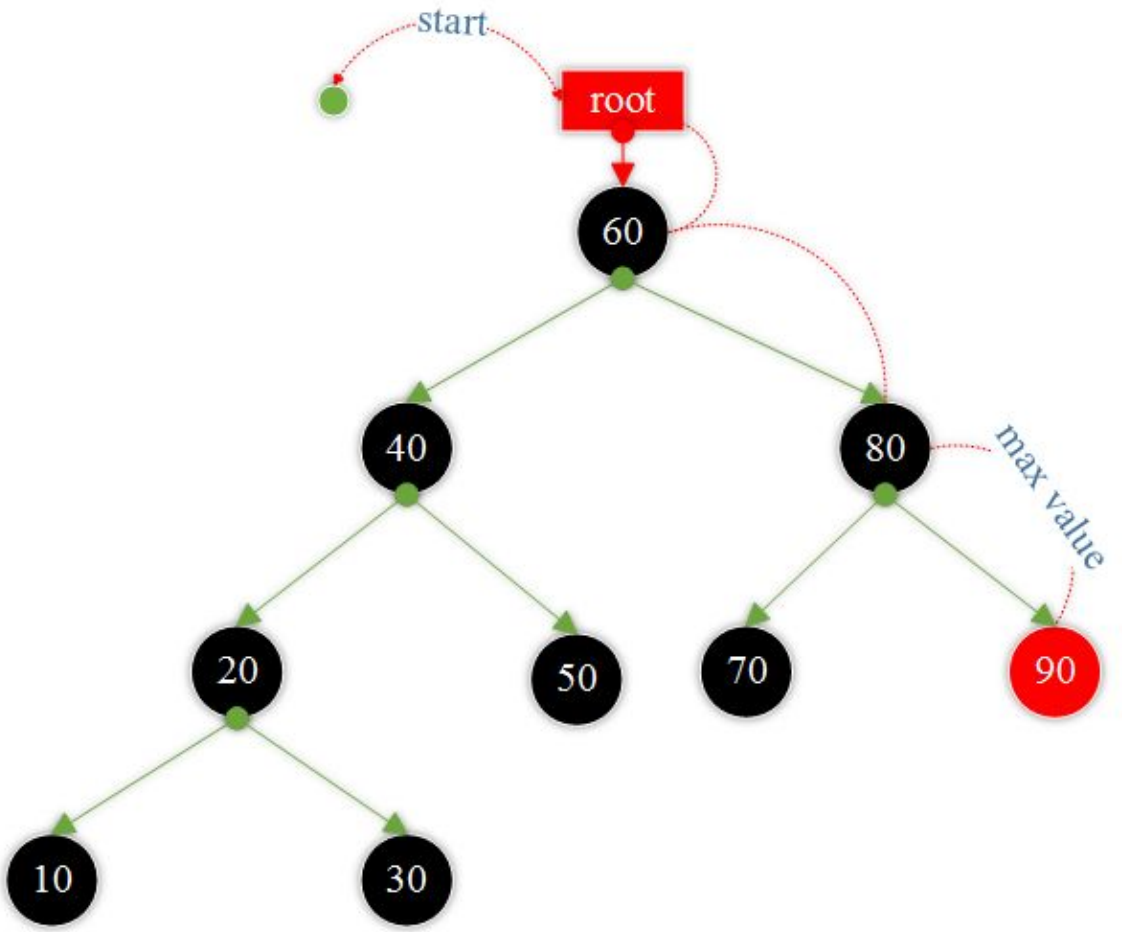
Resultado: Post-order traversal binary search tree 10, 30, 20, 50, 40, 70, 90, 80, 60,

5. á rvore de pesquisa binária Máximo e mínimo

Mínimo: O valor pequeno está no nó filho esquerdo, desde que a recursão atravesse o filho esquerdo até ficar vazio, o nó atual é o nó mínimo.



Máximo: O valor grande está no nó filho certo, desde que o percurso recursivo seja o filho certo até ficar vazio, o nó atual é o maior nó.



Crie um arquivo: TestBinarySearchTreeMaxMinValue.html

```
<script type="text/javascript"> class BinaryTree {
   getRoot() {
   return this.root; }

   // Minimum searchMinValue(node) {
   if (node == null || node.getData() == 0) return null; if (node.left ==
null) {
   return node; }
   // Encontre recursivamente o mínimo na subárvore esquerda return
this.searchMinValue(node.left); }

   // Maximum searchMaxValue(node) {
   if (node == null || node.getData() == 0) return null; if (node.right ==
null) {
   return node; }
   // Encontre recursivamente o valor máximo da subárvore direita return
this.searchMaxValue(node.right); }









   insert(node, newData) {
   if (this.root == null) {
   this.root = new Node(newData, null, null); return; }
```

```
   var compareValue = newData - node.getData(); // Subárvore esquerda recursiva, continue a encontrar a posição de inserção if (compareValue < 0) {
   if (node.left == null) {
   node.left = new Node(newData, null, null); } else {
   this.insert(node.left, newData); }
   } else if (compareValue > 0) {// Subárvore direita recursiva para encontrar a posição de inserção if (node.right == null) {
   node.right = new Node(newData, null, null); } else {
   this.insert(node.right, newData); }
   }
   }
   }

   class Node{
   constructor(data, left, right){
   this.data = data; this.left = left; this.right = right; }

   getData(){
   return this.data; }
   }

   ////////////////////testing//////////////////

   var binaryTree=new BinaryTree();
   // Construindo uma árvore de pesquisa binária
binaryTree.insert(binaryTree.getRoot(), 60);
binaryTree.insert(binaryTree.getRoot(), 40);
binaryTree.insert(binaryTree.getRoot(), 20);
```

```
binaryTree.insert(binaryTree.getRoot(), 10);
binaryTree.insert(binaryTree.getRoot(), 30);
binaryTree.insert(binaryTree.getRoot(), 50);
binaryTree.insert(binaryTree.getRoot(), 80);
binaryTree.insert(binaryTree.getRoot(), 70);
binaryTree.insert(binaryTree.getRoot(), 90);
   document.write("<br> Minimum Value<br> "); var minNode =
binaryTree.searchMinValue(binaryTree.getRoot());
document.write(minNode.getData());
   document.write("<br> Maximum Value<br> "); var maxNode =
binaryTree.searchMaxValue(binaryTree.getRoot());
document.write(maxNode.getData()); </script>
```

Resultado: Minimum Value

10

Maximum Value

90

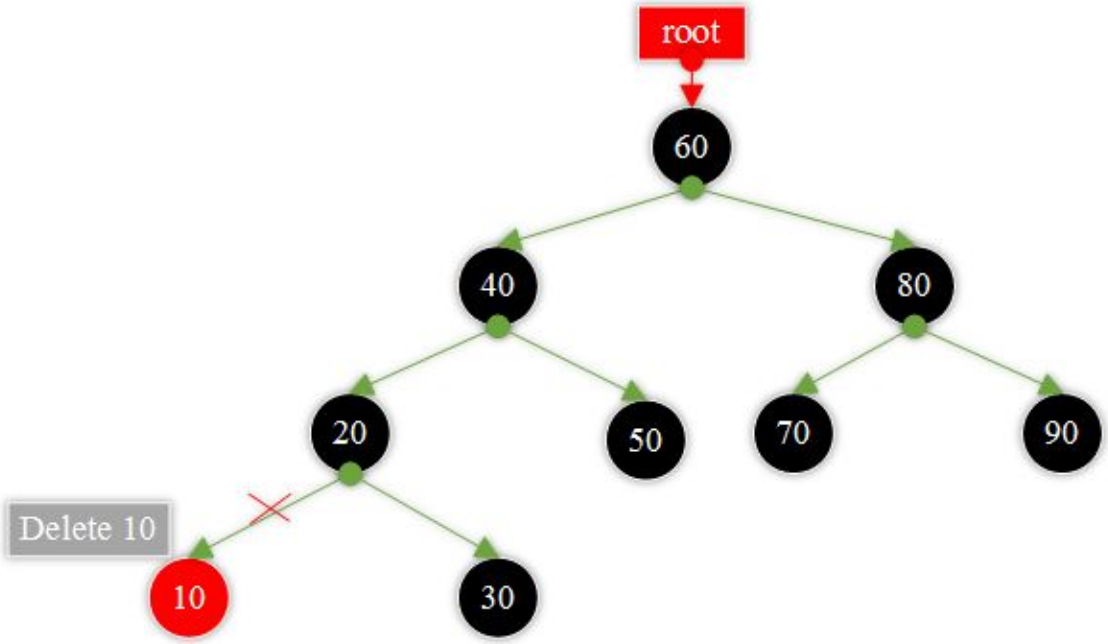
6. árvore de pesquisa binária Delete Node

Casos do nó 3 de exclusão da árvore de pesquisa binária

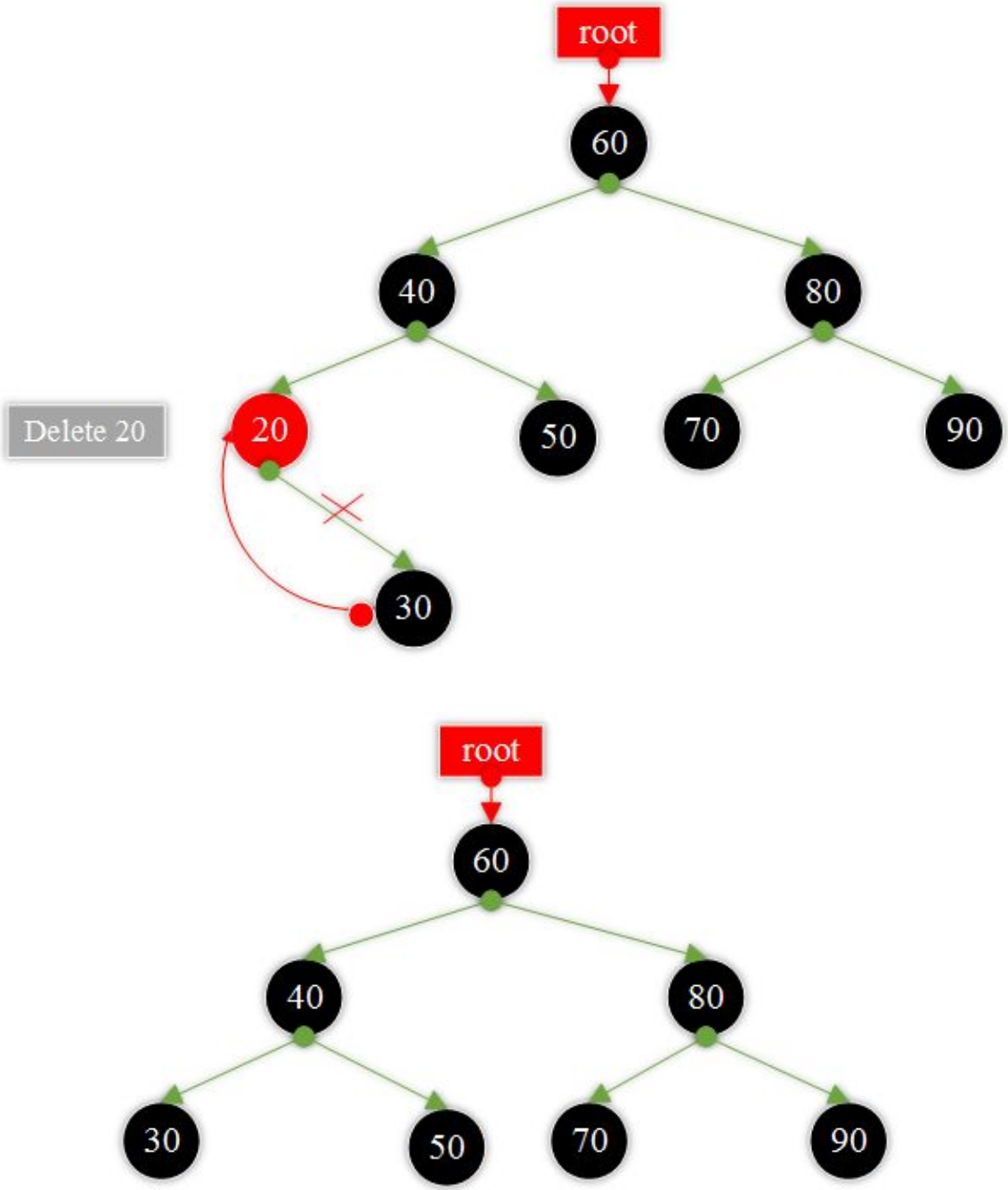
1. Se não houver nó filho, exclua-o diretamente

2. Se houver apenas um nó filho, o nó filho substituirá o nó atual e excluirá o nó atual.
3. Se houver dois nós filhos, substitua o nó atual pelo menor da subárvore direita, porque o menor nó à direita também é maior que o valor à esquerda.

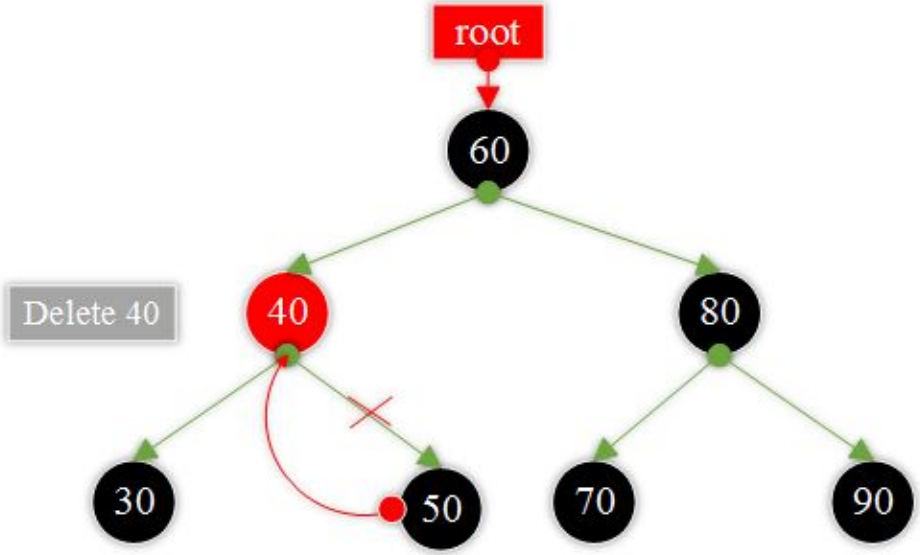
1. Se não houver nó filho, exclua-o diretamente: excluir nó 10

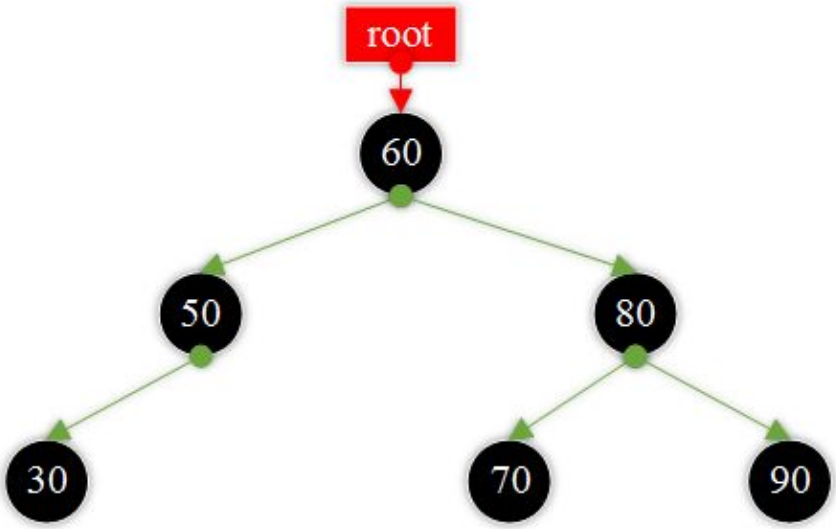


2. Se houver apenas um nó filho, o nó filho substituirá o nó atual e excluirá o nó atual. excluir nó 20



3. Se houver dois nós filhos, substitua o nó atual pelo menor da subárvore direita, porque o menor nó à direita também é maior que o valor à esquerda, excluir nó 40





Crie um arquivo: TestBinarySearchTreeDelete.html

```
<script type="text/javascript"> class Node{
   constructor(data, left, right){
   this.data = data; this.left = left; this.right = right; }

   getData(){
   return this.data; }
   setData(data){
   this.data = data; }
   }

   class BinaryTree {
   getRoot() {
   return this.root; }

   inOrder(root) {
   if (root == null) {
   return; }
   this.inOrder(root.left); // Recursivamente Percorrendo a subárvore
esquerda document.write(root.getData() + ", "); this.inOrder(root.right); //
Recursivamente Percorrendo a subárvore direita }

   searchMinValue(node) {
   if (node == null || node.getData() == 0) return null; if (node.left ==
null) {
   return node; }
   // Encontre recursivamente o mínimo na subárvore esquerda return
this.searchMinValue(node.left); }


   remove(node, newData) {
   if (node == null) return node; var compareValue = newData -
node.getData(); if (compareValue > 0) {
   node.right = this.remove(node.right, newData); } else if (compareValue
< 0) {
```

```
   node.left = this.remove(node.left, newData); } else if (node.left != null
&& node.right != null) {
   // Encontre o nó mínimo da subárvore direita para substituir o nó atual
node.setData(this.searchMinValue(node.right).getData()); node.right =
this.remove(node.right, node.getData()); } else {
   node = (node.left != null) ? node.left : node.right; }
   return node; }

   insert(node, newData) {
   if (this.root == null) {
   this.root = new Node(newData, null, null); return; }

   var compareValue = newData - node.getData(); // Subárvore esquerda
recursiva, continue a encontrar a posição de inserção if (compareValue <
0) {
   if (node.left == null) {
   node.left = new Node(newData, null, null); } else {
   this.insert(node.left, newData); }
   } else if (compareValue > 0) {// Subárvore direita recursiva para
encontrar a posição de inserção if (node.right == null) {
   node.right = new Node(newData, null, null); } else {
   this.insert(node.right, newData); }
   }
   }
   }


   //////////////////////testing//////////////////
   var binaryTree=new BinaryTree(); // Construindo uma árvore de
pesquisa binária binaryTree.insert(binaryTree.getRoot(), 60);
binaryTree.insert(binaryTree.getRoot(), 40);
binaryTree.insert(binaryTree.getRoot(), 20);
binaryTree.insert(binaryTree.getRoot(), 10);
binaryTree.insert(binaryTree.getRoot(), 30);
binaryTree.insert(binaryTree.getRoot(), 50);
binaryTree.insert(binaryTree.getRoot(), 80);
```

```
binaryTree.insert(binaryTree.getRoot(), 70);
binaryTree.insert(binaryTree.getRoot(), 90);
   document.write("<br>delete node is: 10 <br>");
binaryTree.remove(binaryTree.getRoot(), 10); document.write("<br>In-
order traversal binary tree <br>");
binaryTree.inOrder(binaryTree.getRoot());
   document.write("<br>----------------------------------------<br>");
document.write("<br>delete node is: 20<br>");
binaryTree.remove(binaryTree.getRoot(), 20);
   document.write("<br>In-order traversal binary tree<br>");
binaryTree.inOrder(binaryTree.getRoot());
   document.write("<br>----------------------------------------<br>");
document.write("<br>delete node is: 40<br>");
binaryTree.remove(binaryTree.getRoot(), 40);
   document.write("<br>In-order traversal binary tree<br>");
binaryTree.inOrder(binaryTree.getRoot()); </script>
```

Resultado: delete node is: 10
In-order traversal binary tree 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 , ----------
---
delete node is: 20
In-order traversal binary tree 30 , 40 , 50 , 60 , 70 , 80 , 90 , ----------
---
delete node is: 40
In-order traversal binary tree 30 , 50 , 60 , 70 , 80 , 90 ,

Ordenar da pilha

Ordenar da pilha (Heap Sorting): O valor do nó não terminal na árvore binária não é maior que o valor de seus nós filhos esquerdo e direito.

Small top heap : ki <= k2i and ki <= k2i+1
Big top heap :ki >= k2i and ki >= k2i+1

Subscrito do nó pai = (i-1)/2
Subscrito Subnó esquerdo = 2*i+1
Subscrito do subnó direito = 2*i+2

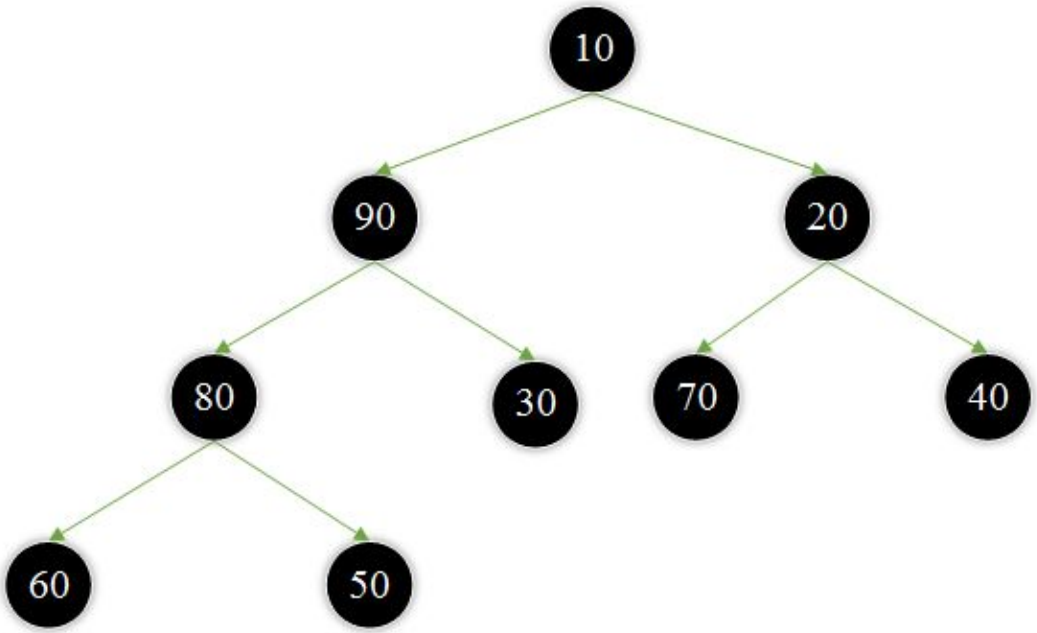
Ordenar da pilha:

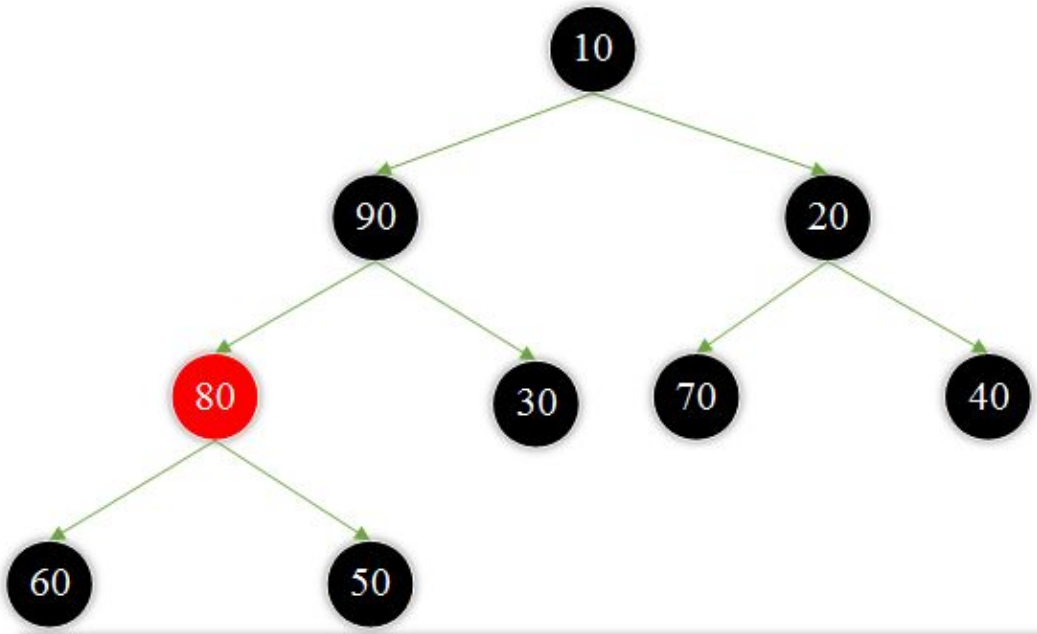
1. Construa uma pilha

2. Após a saída do elemento superior do heap, ajuste de cima para baixo, compare o elemento superior com o nó raiz de suas subárvores esquerda e direita e troque o menor elemento pela parte superior do heap; depois ajuste continuamente até os nós das folhas para obter novo heap.

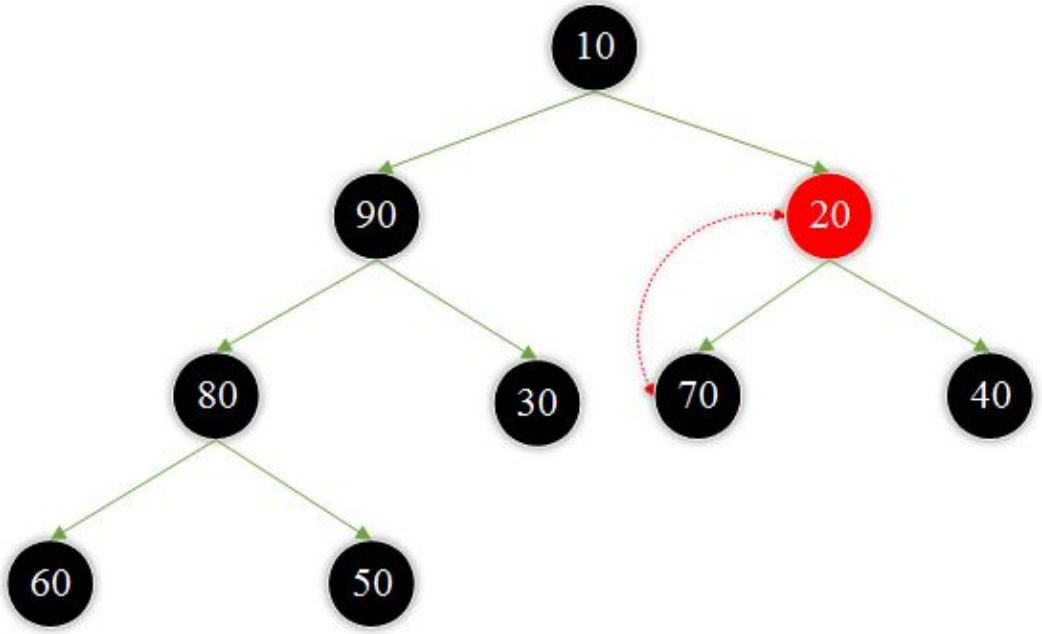
1. {10, 90, 20, 80, 30, 70, 40, 60, 50} construir heap e, em seguida, heap classificar saída.

Inicialize o heap e construa o heap

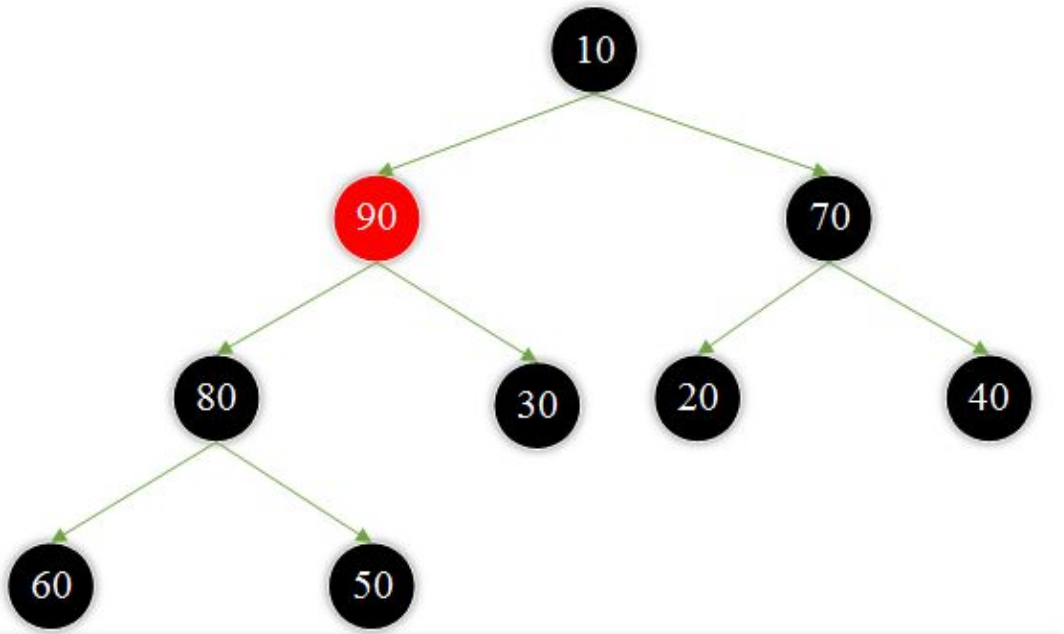

10
90
20
80
30
70
40
60
50



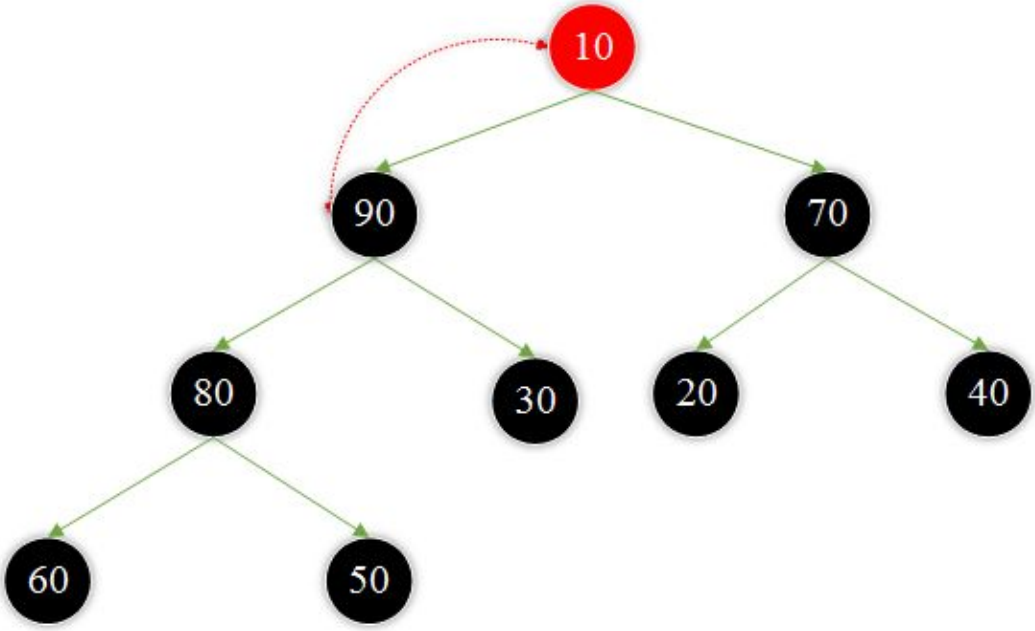
Not Leaf Node = 80 > left = 60 , 80 > right = 50 No need to move



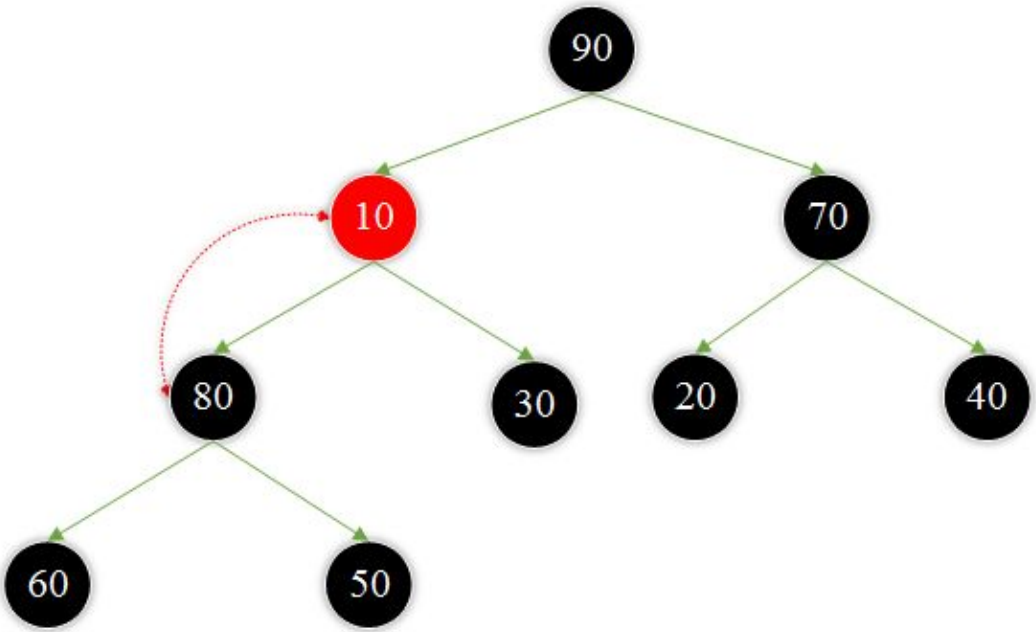
Not Leaf Node = 20 < left = 70 , 70 > right = 40 , 20 swap with 70



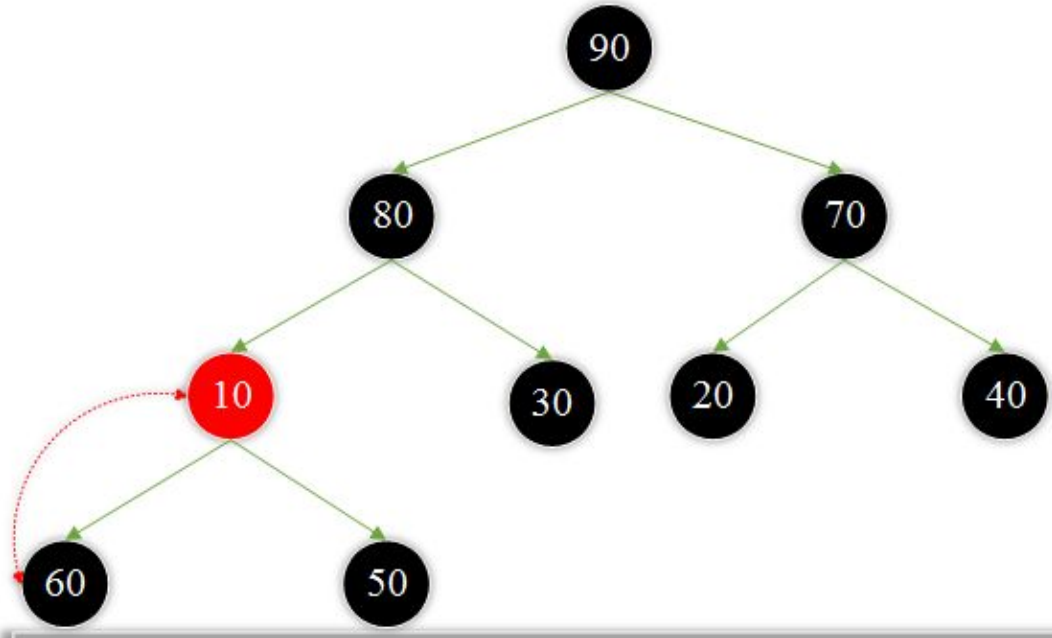
Not Leaf Node = 90 > left = 80 , 80 > right = 30 No need to move



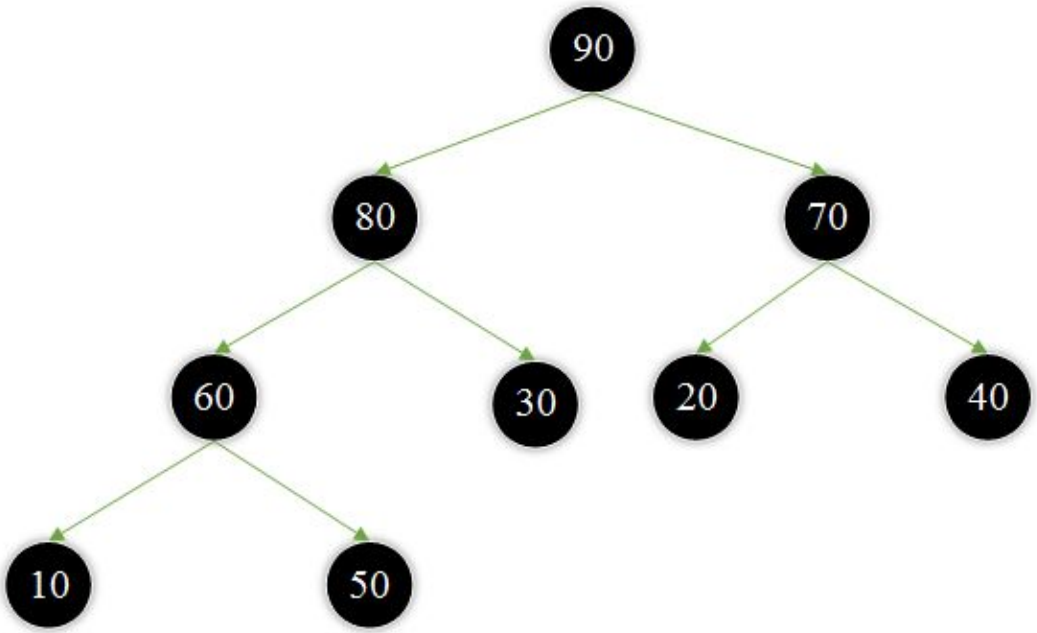
Not Leaf Node = 10 < left = 90 , 90 > right = 70 , 10 swap with 90



Still Not Leaf Node = 10 < left = 80 , 80 > right =30 , 10 swap with 80

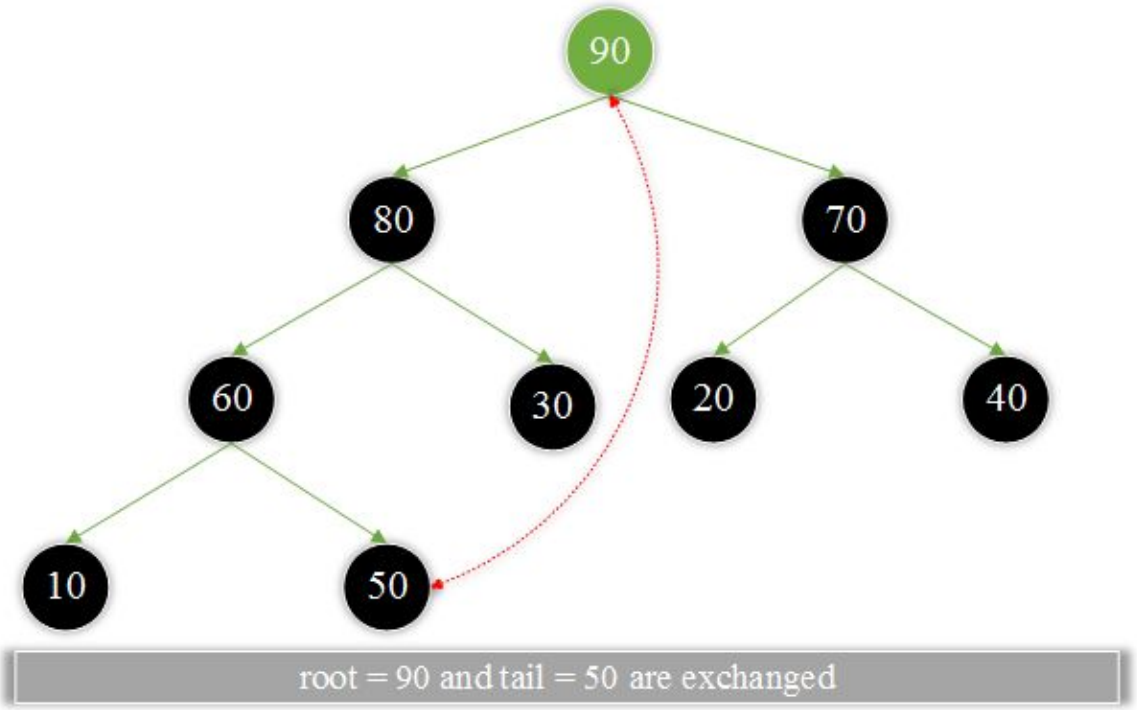


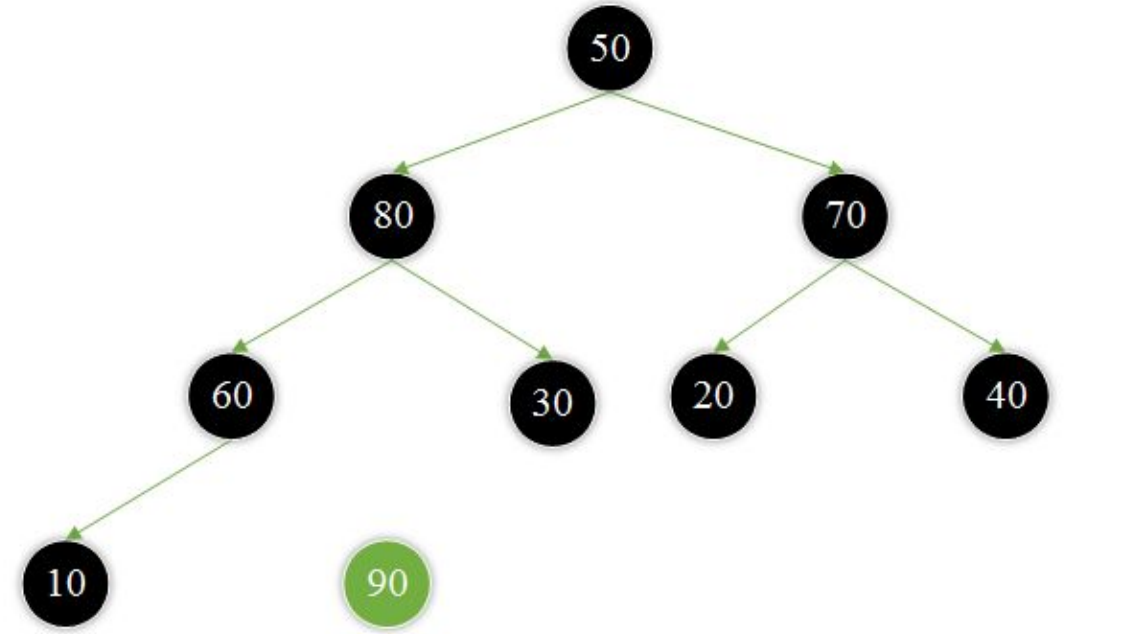
Still Not Leaf Node = 10 < left = 60 , 60 > right =50 , 10 swap with 60



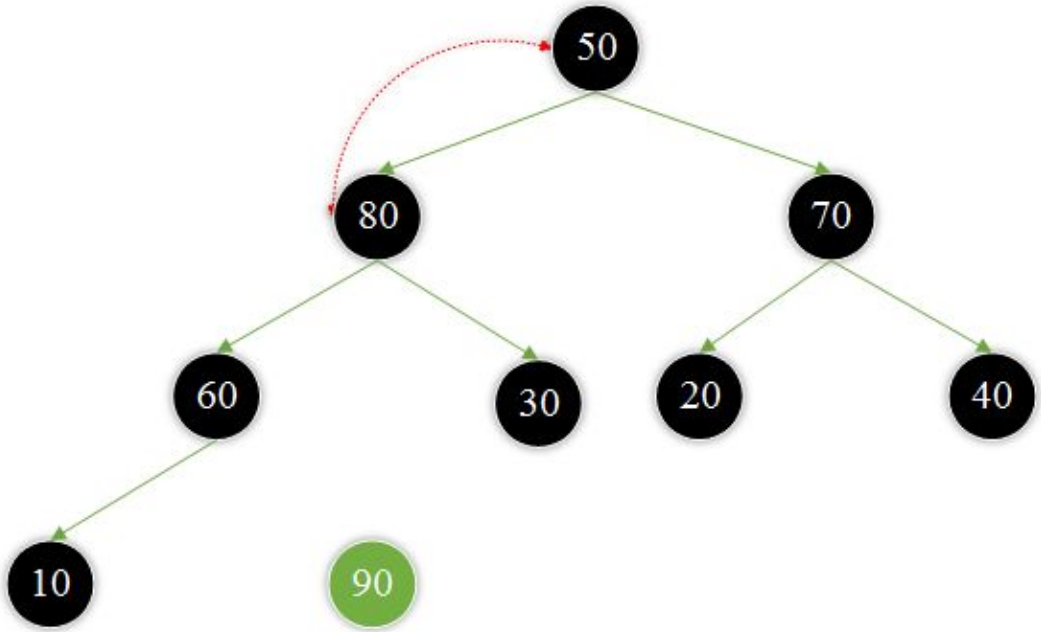
Crie o heap concluído

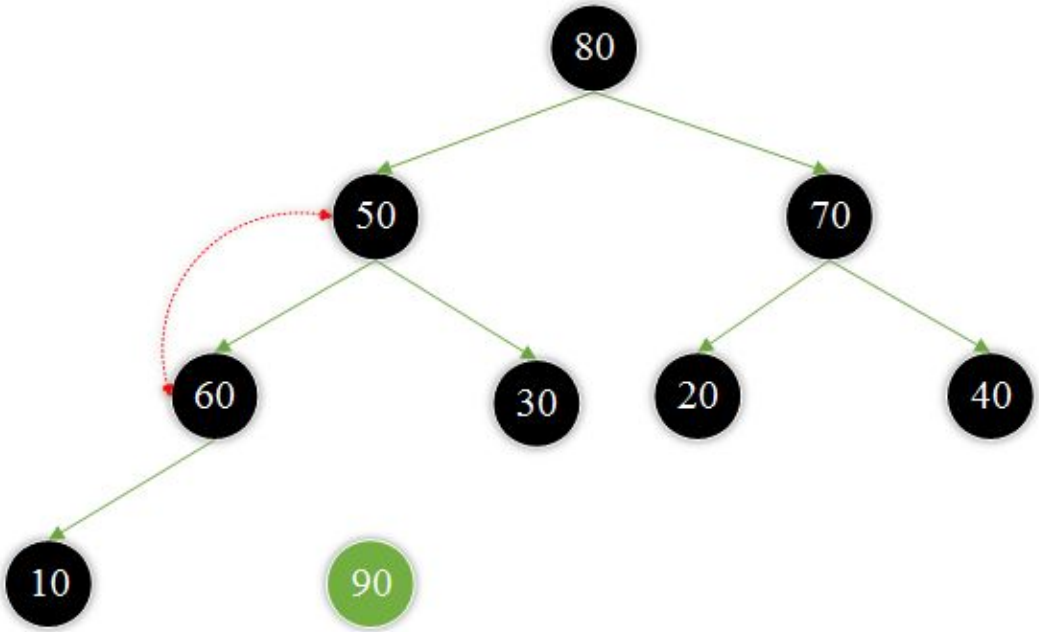
2. Iniciar a Ordenar da pilha

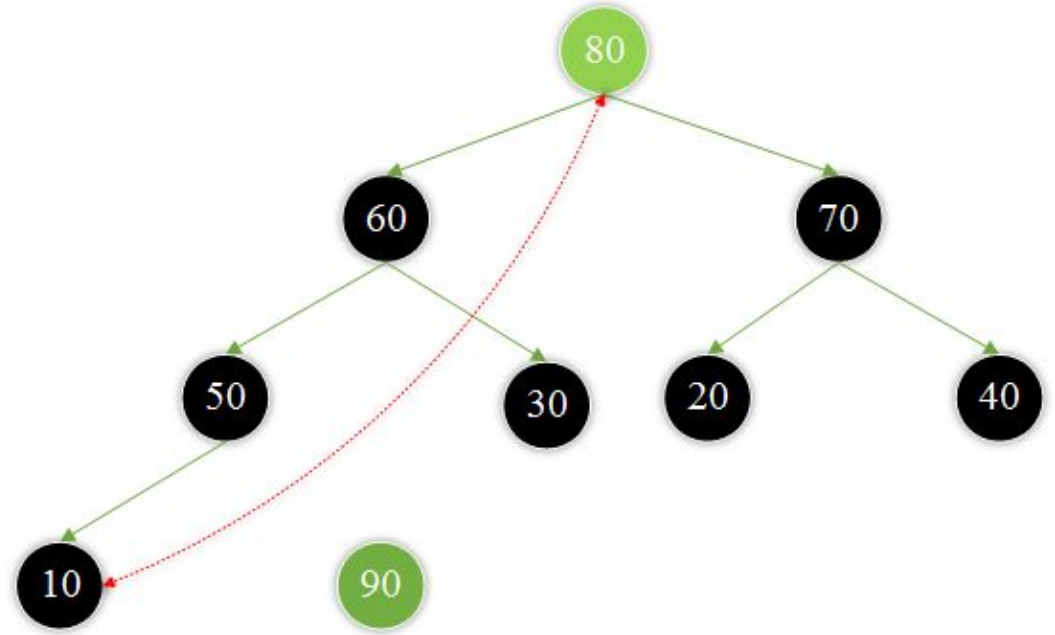




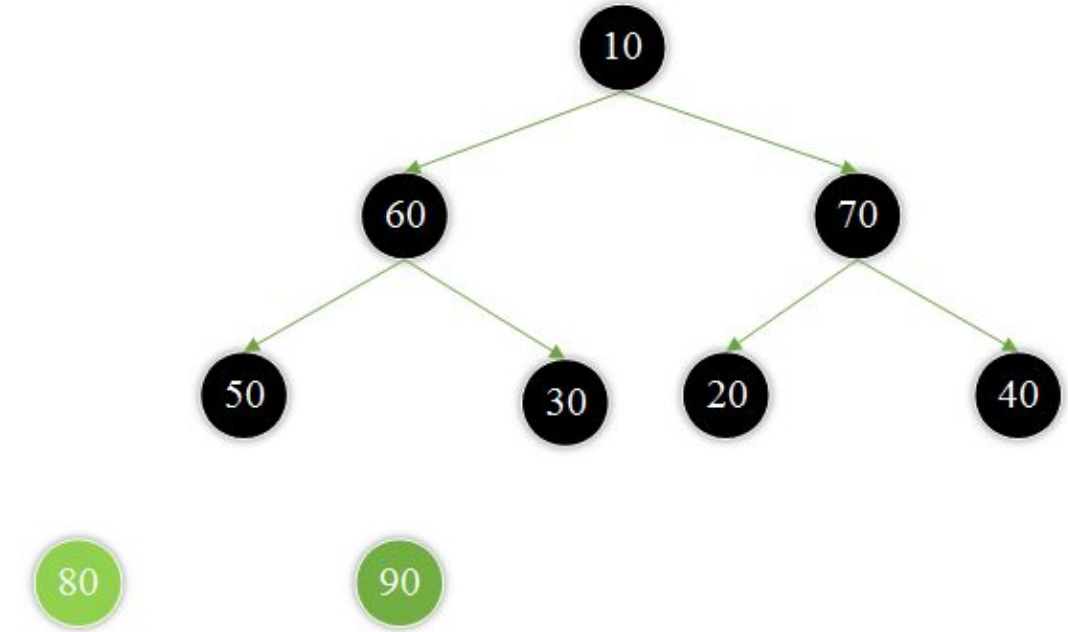
adjust the heap



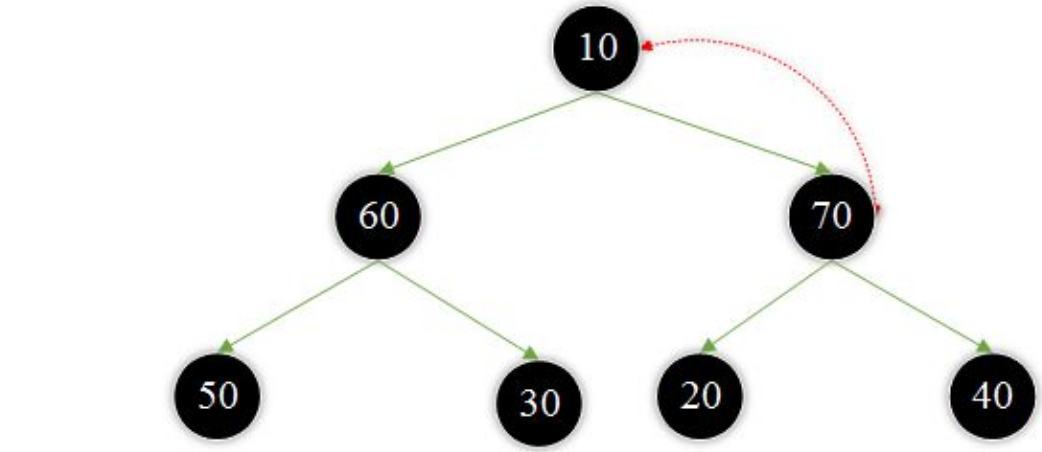


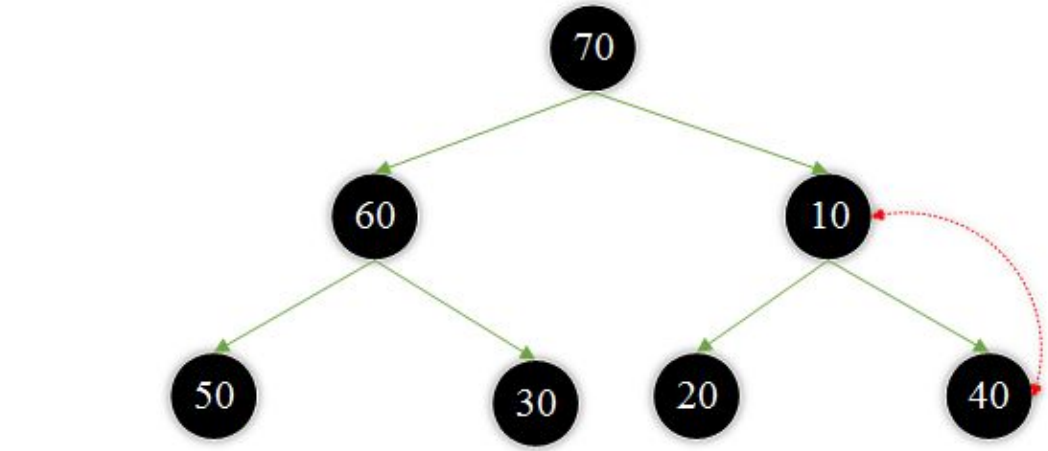


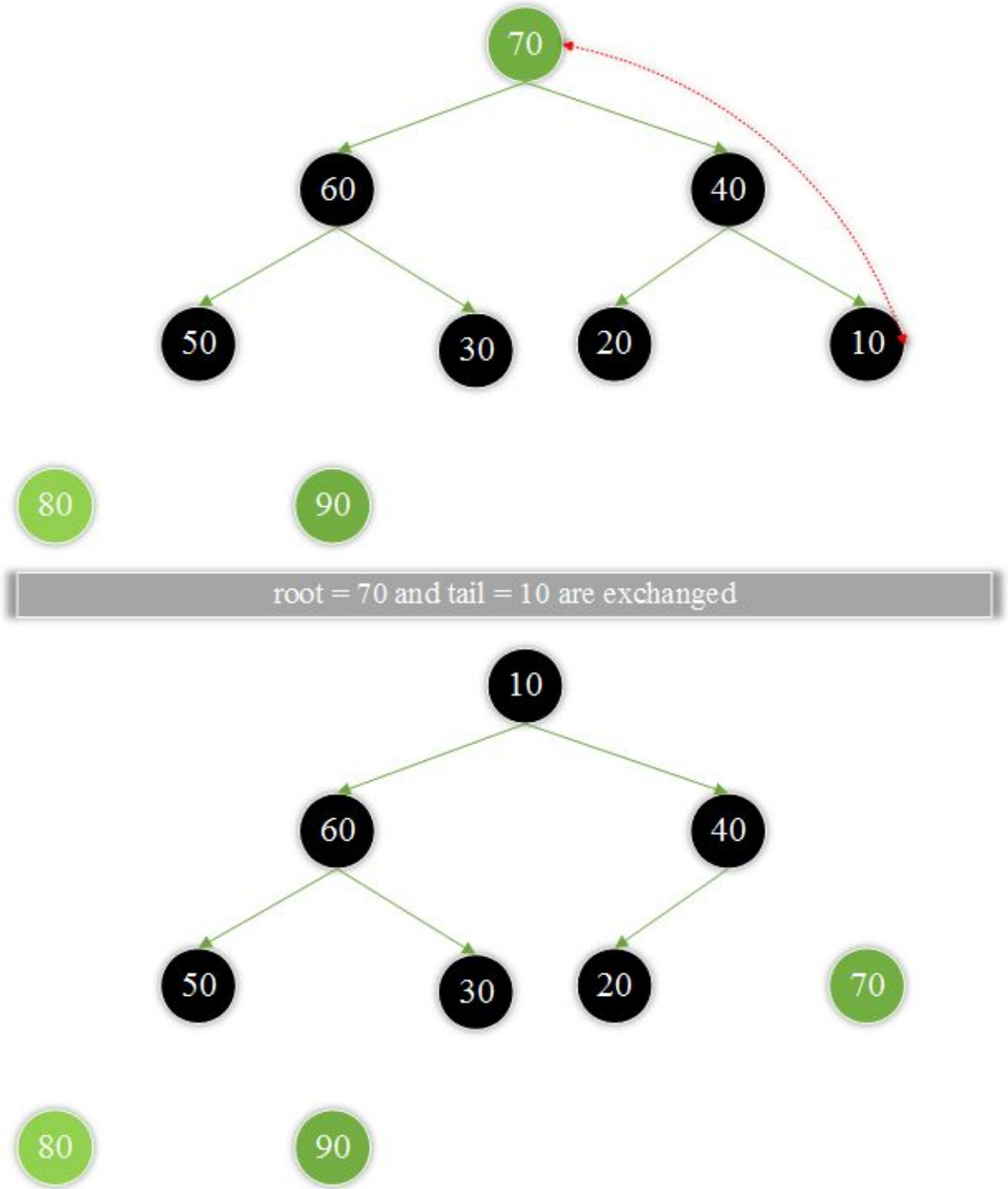
root = 80 and tail = 10 are exchanged



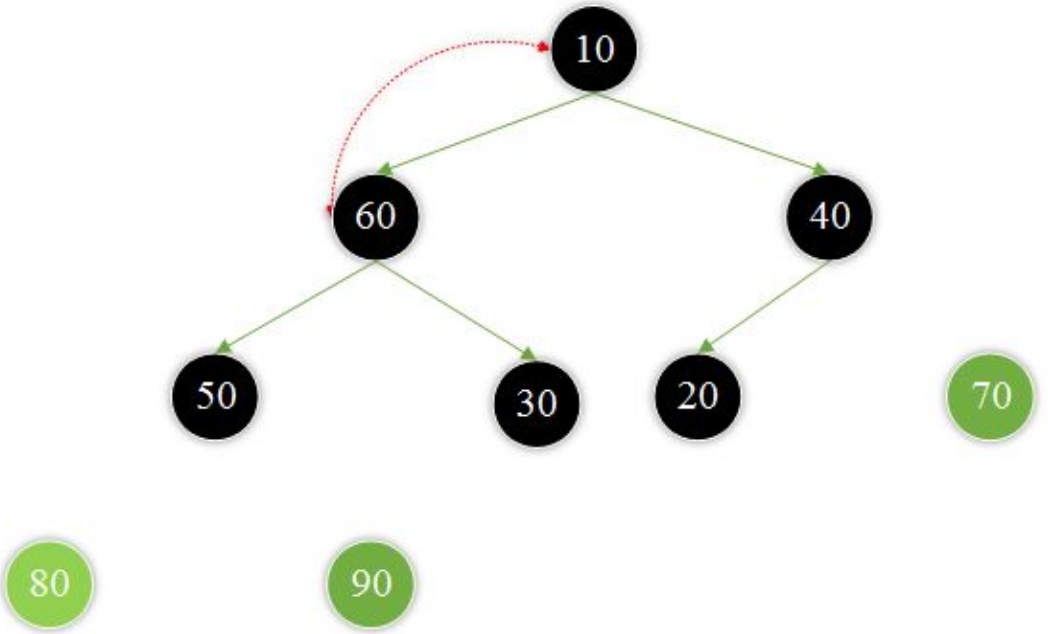
adjust the heap

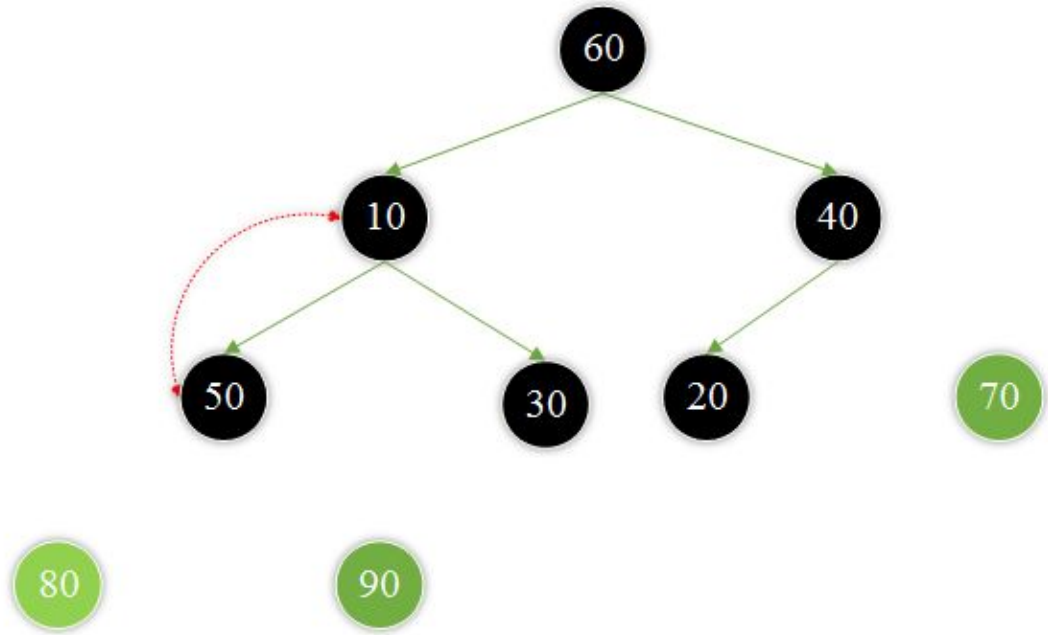


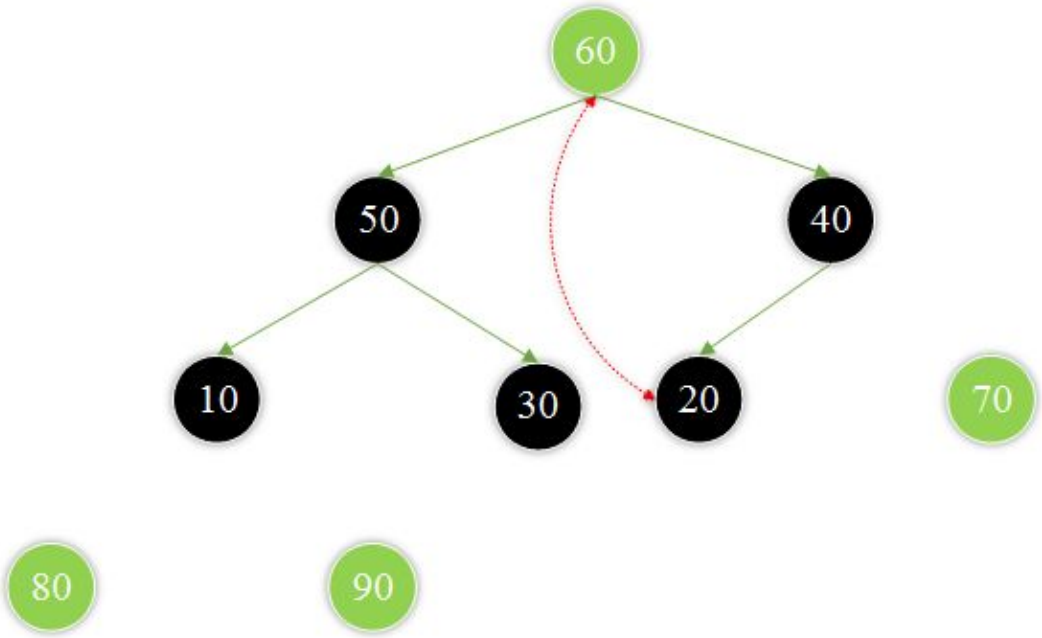



70
60
40
50
30
20
10
80
90
root = 70 and tail = 10 are exchanged
10
60
40
50
30
20
70
80
90

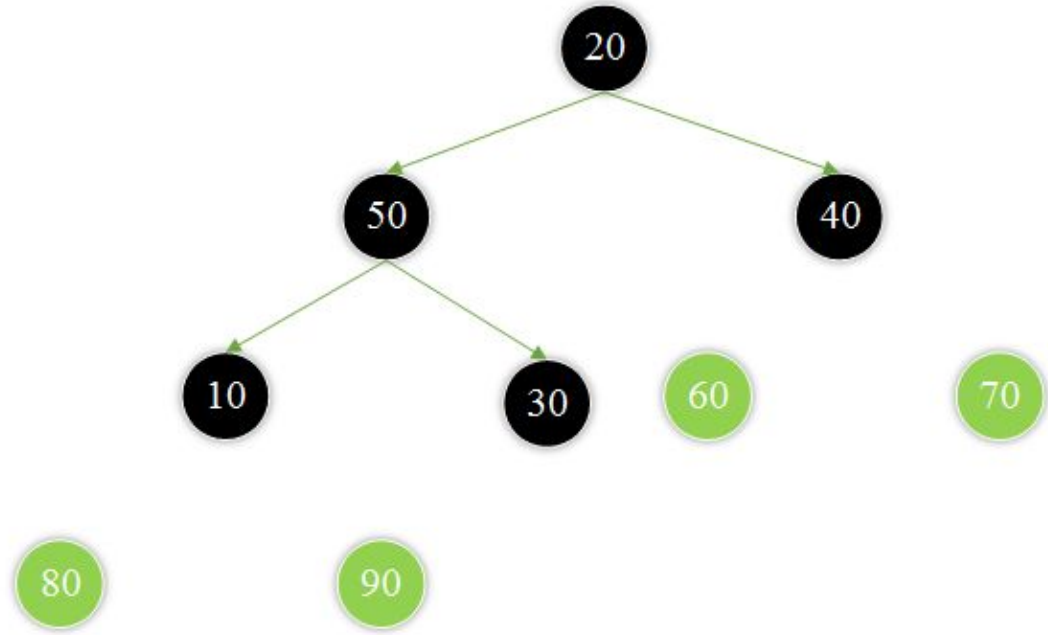
adjust the heap



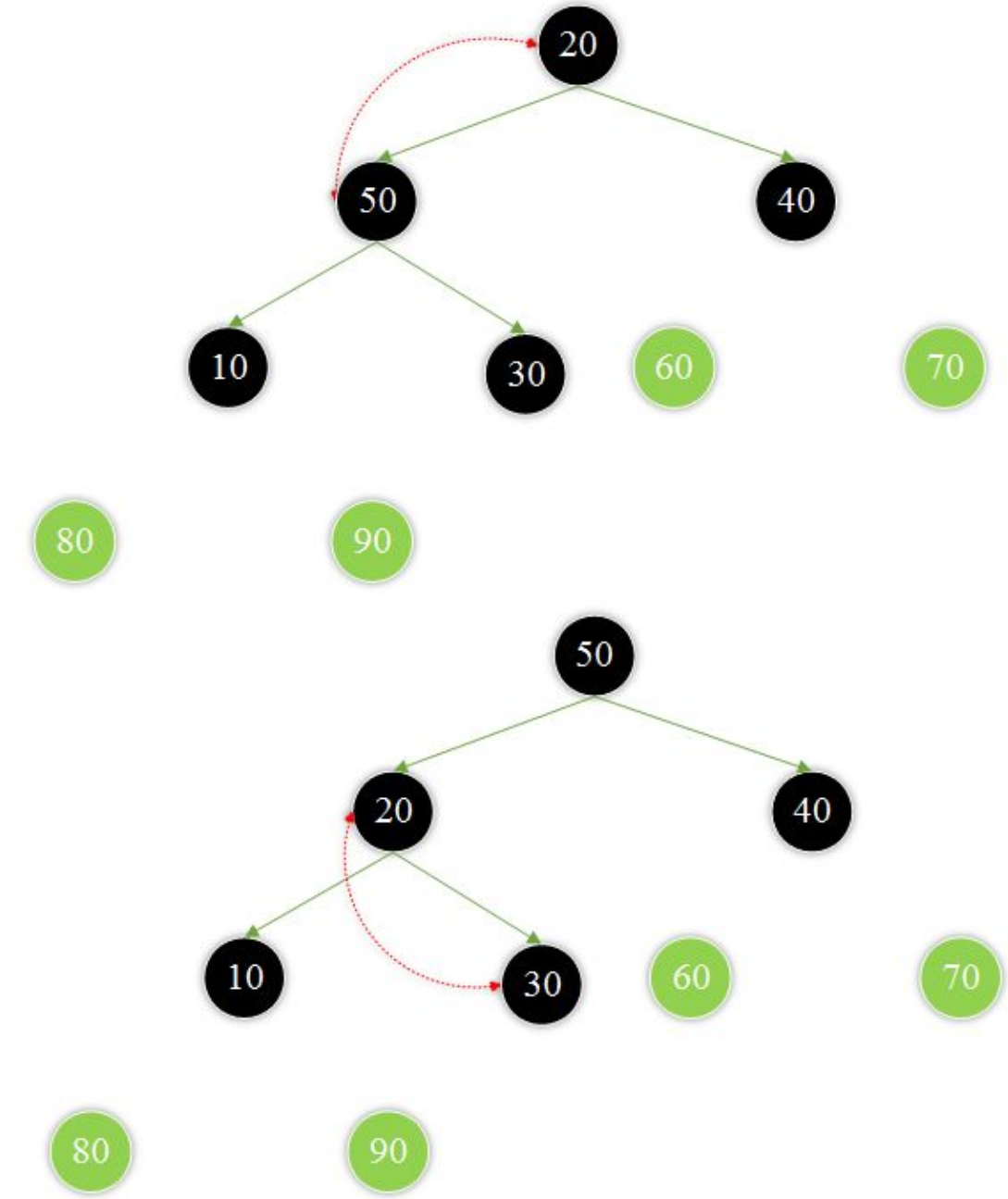


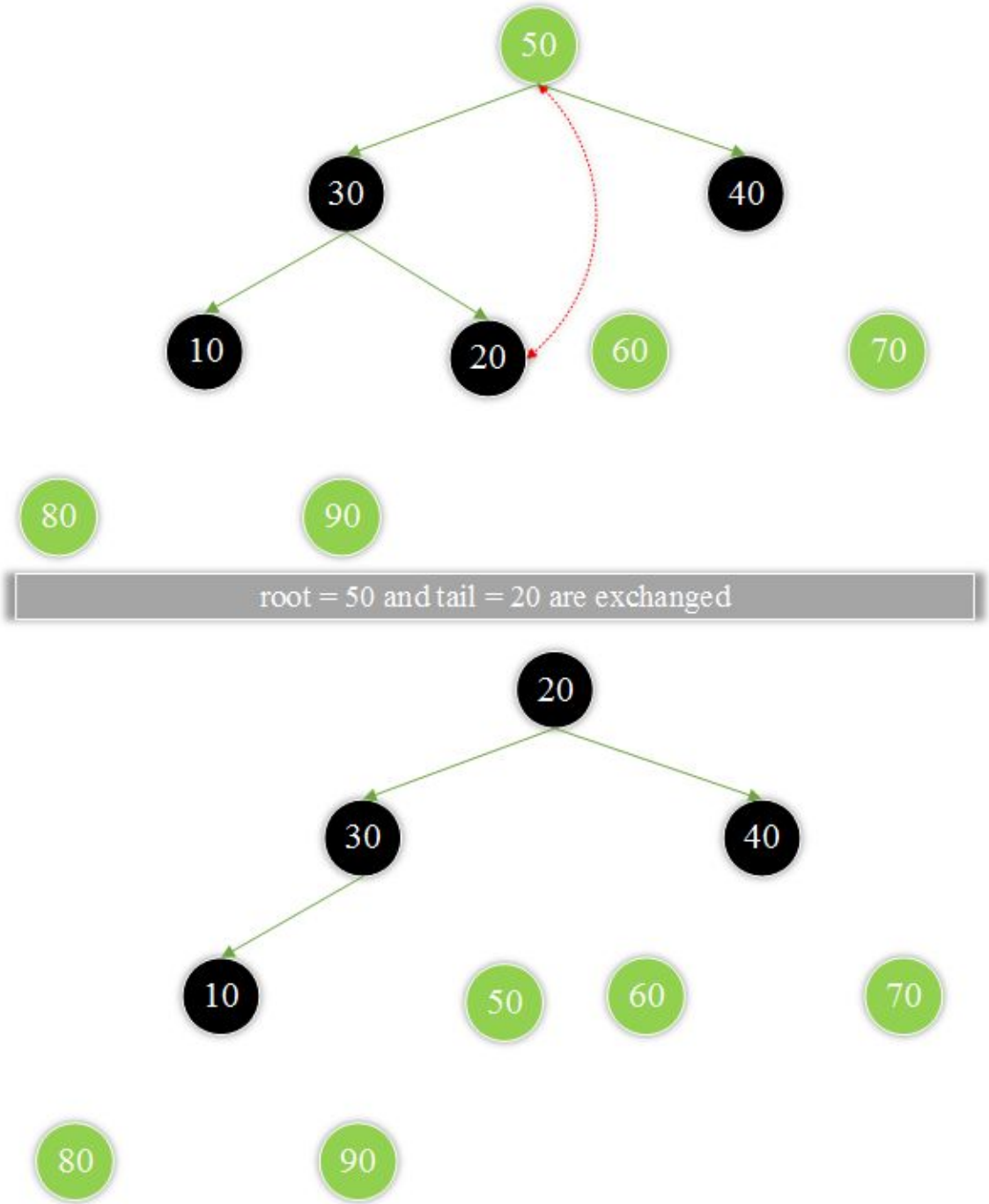


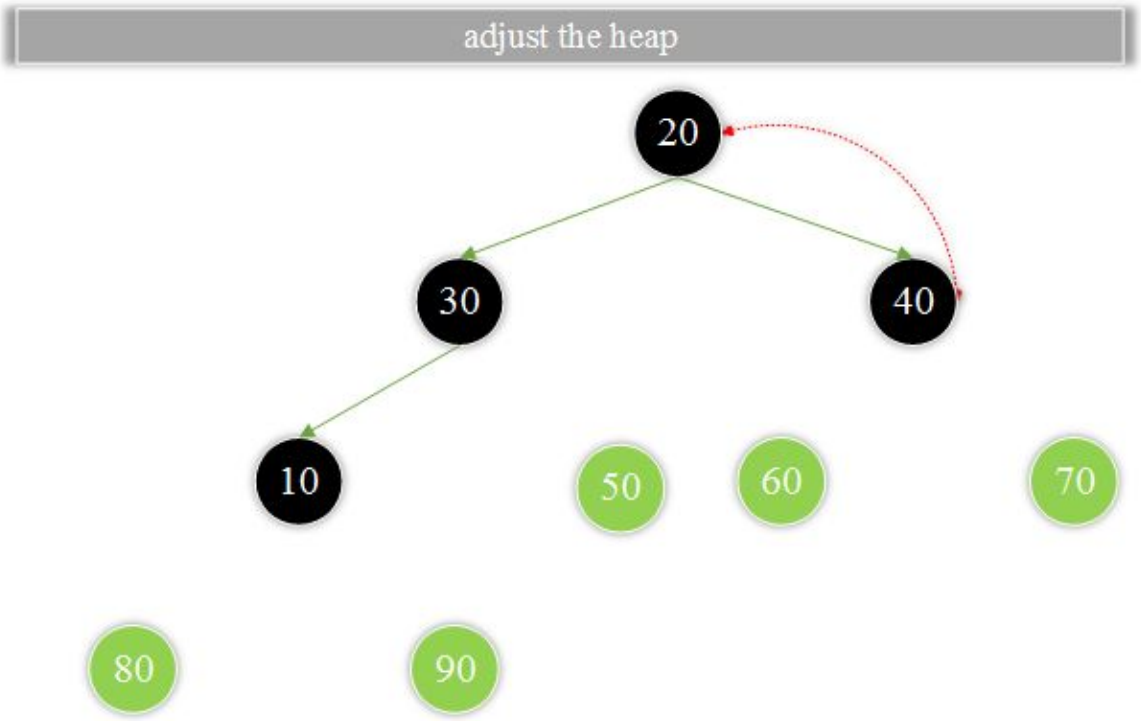
root = 60 and tail = 20 are exchanged

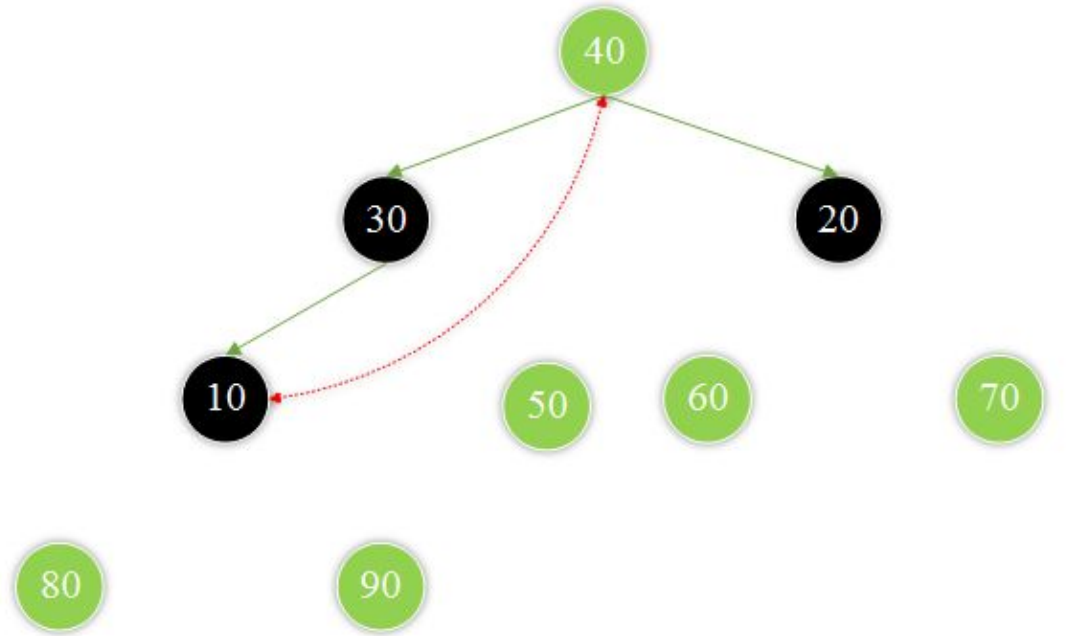


adjust the heap

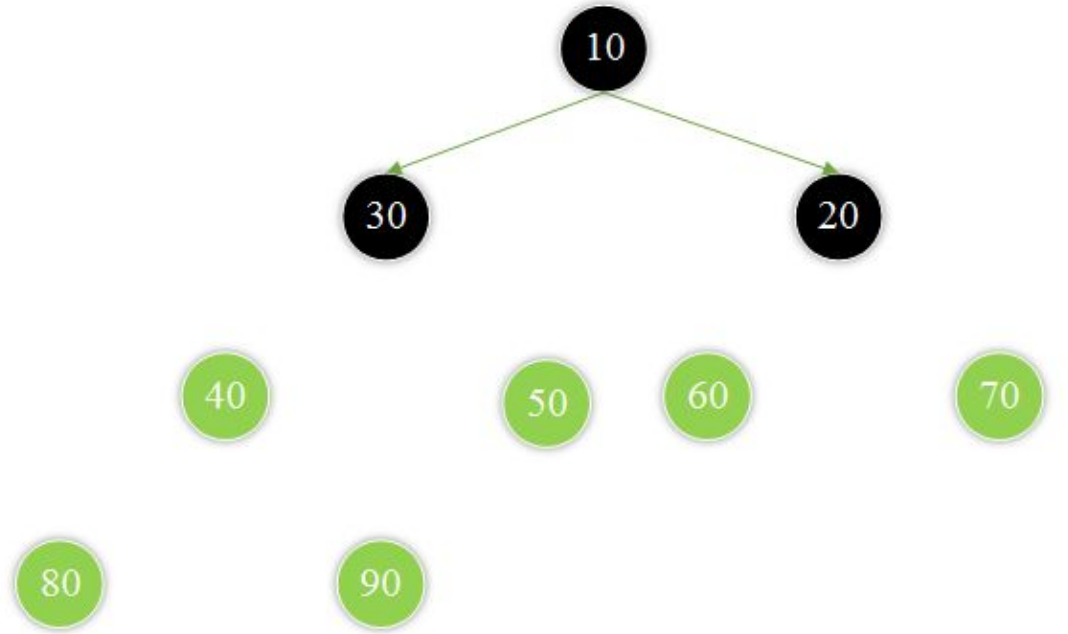


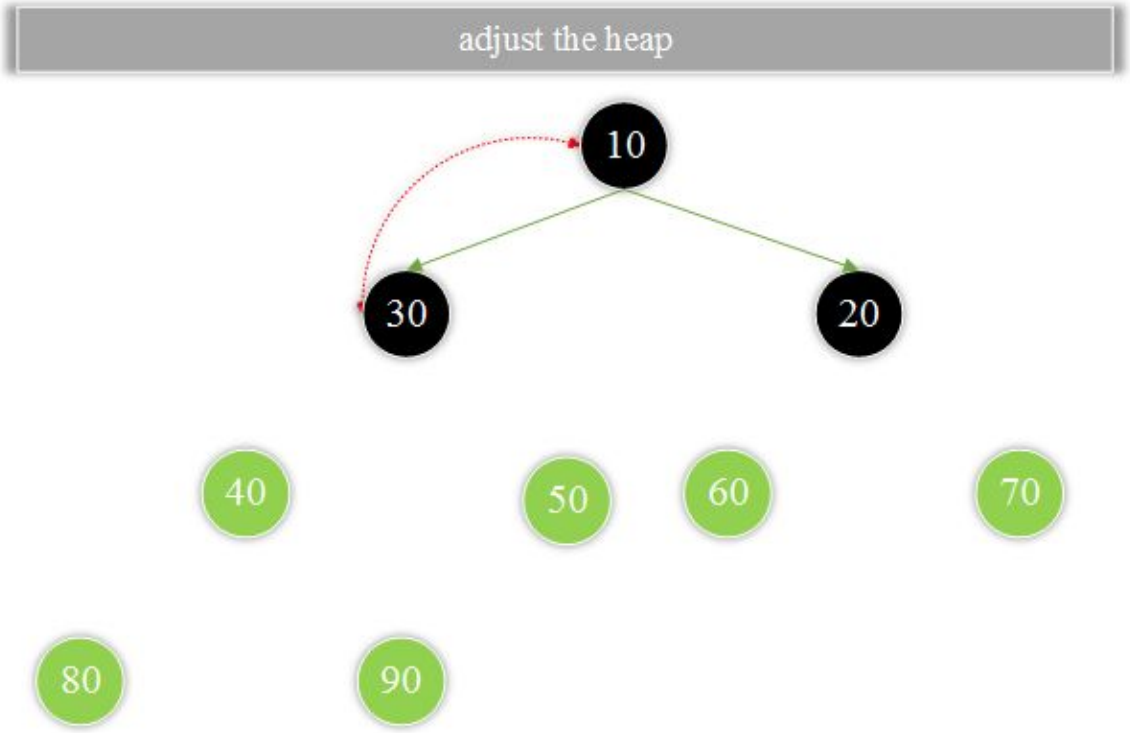

50
30
40
10
20
60
70
80
90
root = 50 and tail = 20 are exchanged
20
30
40
10
50
60
70
80
90

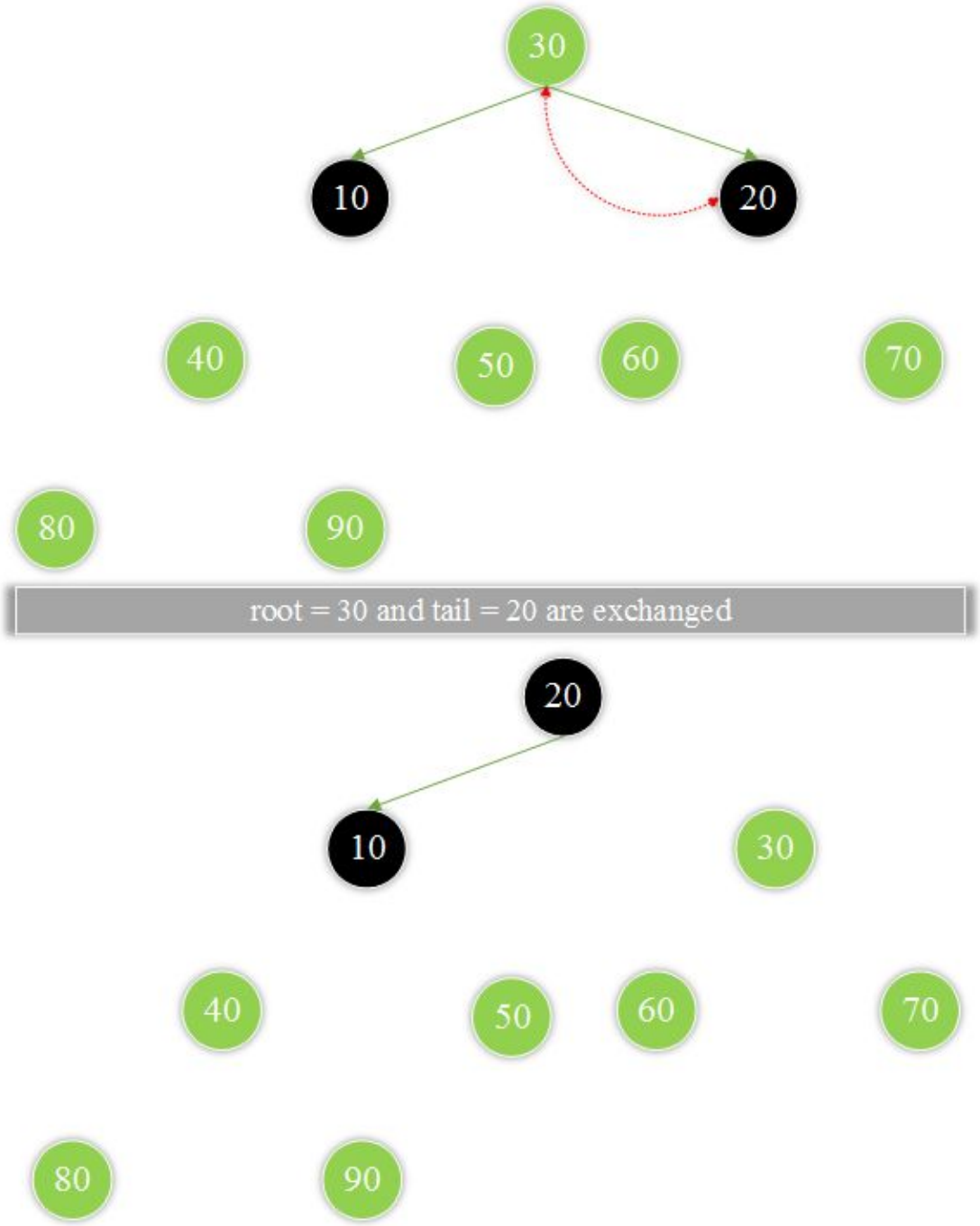

adjust the heap
20
30
40
10
50
60
70
80
90

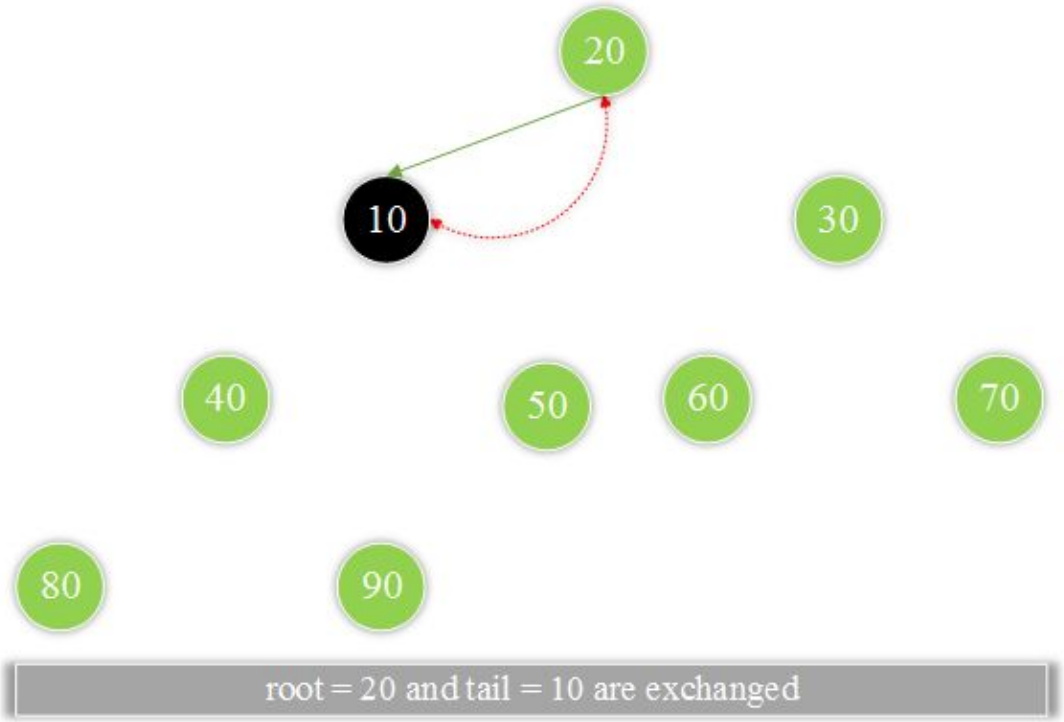


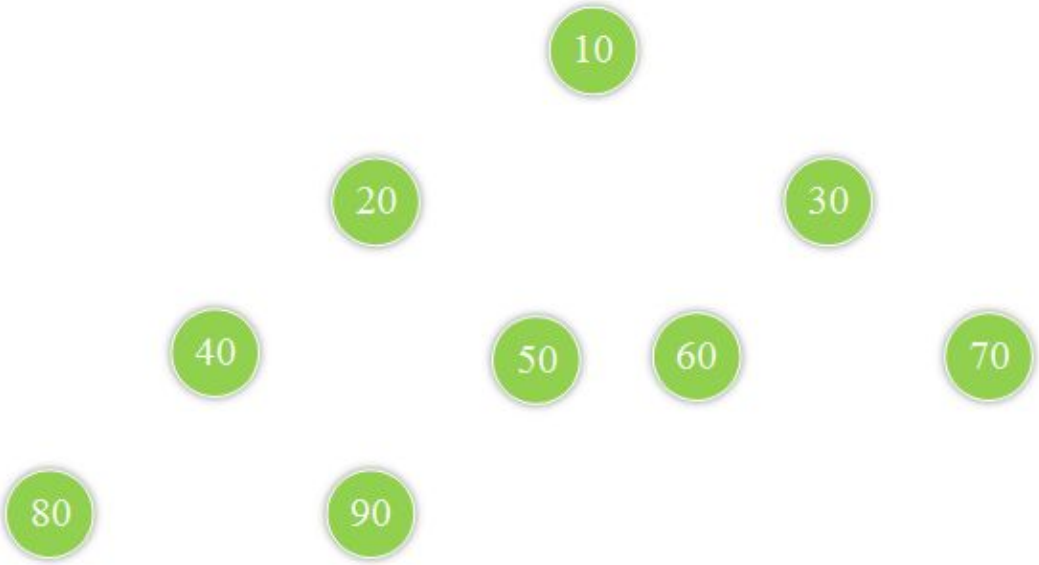
root = 40 and tail = 10 are exchanged




adjust the heap
10
30
20
40
50
60
70
80
90


30
10
20
40
50
60
70
80
90
root = 30 and tail = 20 are exchanged
20
10
30
40
50
60
70
80
90


20
10
30
40
50
60
70
80
90
root = 20 and tail = 10 are exchanged


10
20
30
40
50
60
70
80
90

Resultado


10
20
30
40
50
60
70
80
90

Crie um arquivo: TestBinaryHeapSorting.html

```
<script type="text/javascript"> class HeapSort {
   constructor(){
   this.array = []; }

   // Inicialize o heap createHeap(array) {
   this.array = array; // Construa uma pilha, (array.length - 1) / 2 varre
metade dos nós com nós filhos for (var i = (array.length - 1) / 2; i >= 0; i-
-) {
   this.adjustHeap(i, array.length - 1); }
   }

   // Ajustar pilha adjustHeap(currentIndex, maxLength) {
   var noLeafValue = this.array[currentIndex];
   //2 * currentIndex + 1 Subscrito atual da subárvore esquerda for (var j
= 2 * currentIndex + 1; j <= maxLength; j = currentIndex * 2 + 1) {
   if (j < maxLength && this.array[j] < this.array[j + 1]) {
   j++; }
   if (noLeafValue >= this.array[j]) {
   break; }
   this.array[currentIndex] = this.array[j]; // Mover para o nó pai
currentIndex = j; }

   this.array[currentIndex] = noLeafValue; }

   heapSort() {
   for (var i = this.array.length - 1; i > 0; i--) {
   var temp = this.array[0]; this.array[0] = this.array[i]; this.array[i] =
temp; this.adjustHeap(0, i - 1); }
   }
   }
   //////////////////////testing//////////////////
   var heapSort = new HeapSort();
   var scores = [ 10, 90, 20, 80, 30, 70, 40, 60, 50 ];
   document.write("Before building a heap : <br>"); for (var i = 0; i <
scores.length; i++) {
   document.write(scores[i] + ", "); }
```

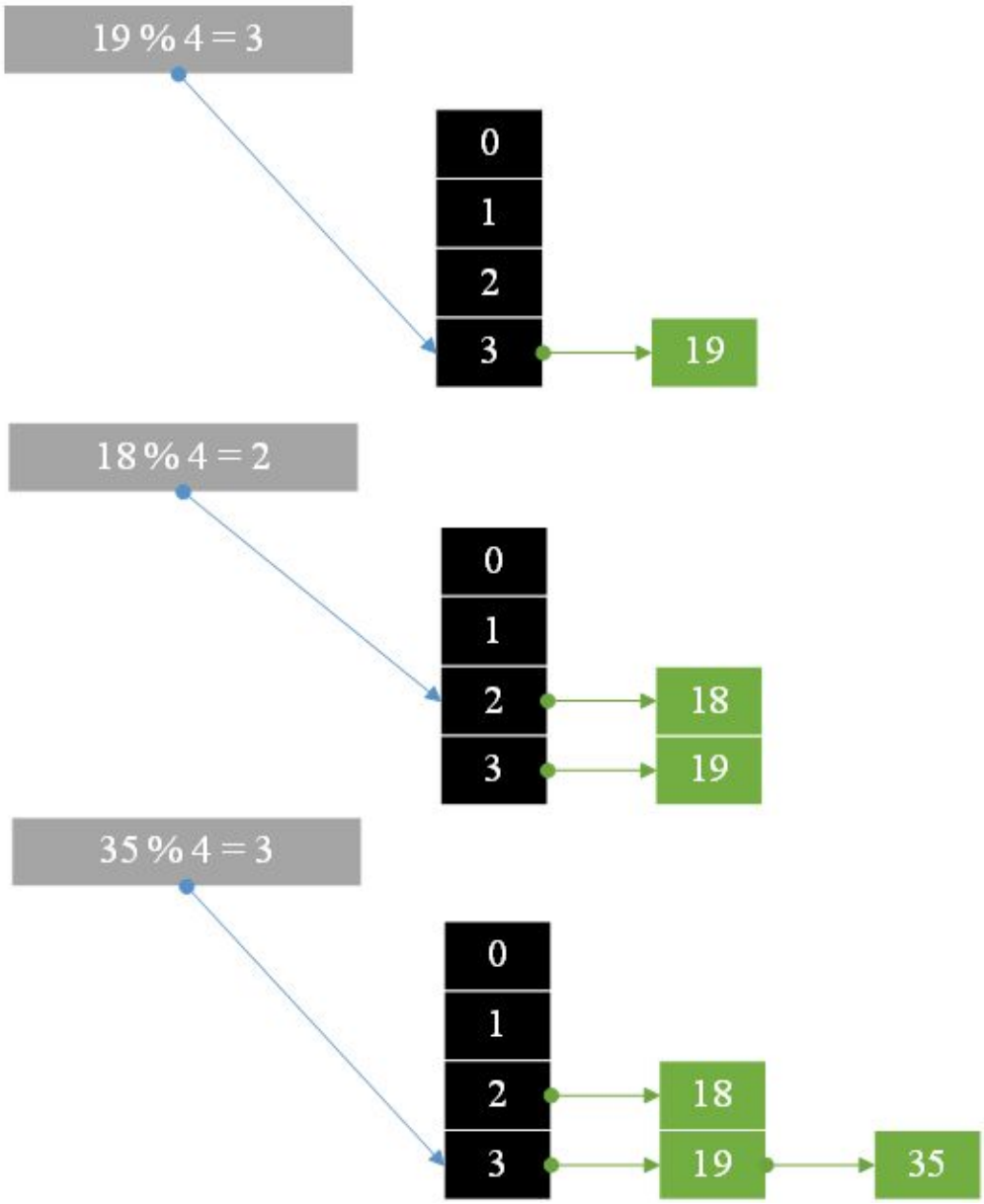
```
   document.write("<br><br>");
   ////////////////////////////////////////////////////////
   document.write("After building a heap : <br>");
heapSort.createHeap(scores); for (var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ", "); }
   document.write("<br><br>");
   ////////////////////////////////////////////////////////
   document.write("After heap sorting : <br>"); heapSort.heapSort(); for
(var i = 0; i < scores.length; i++) {
   document.write(scores[i] + ", "); }
</script>
```

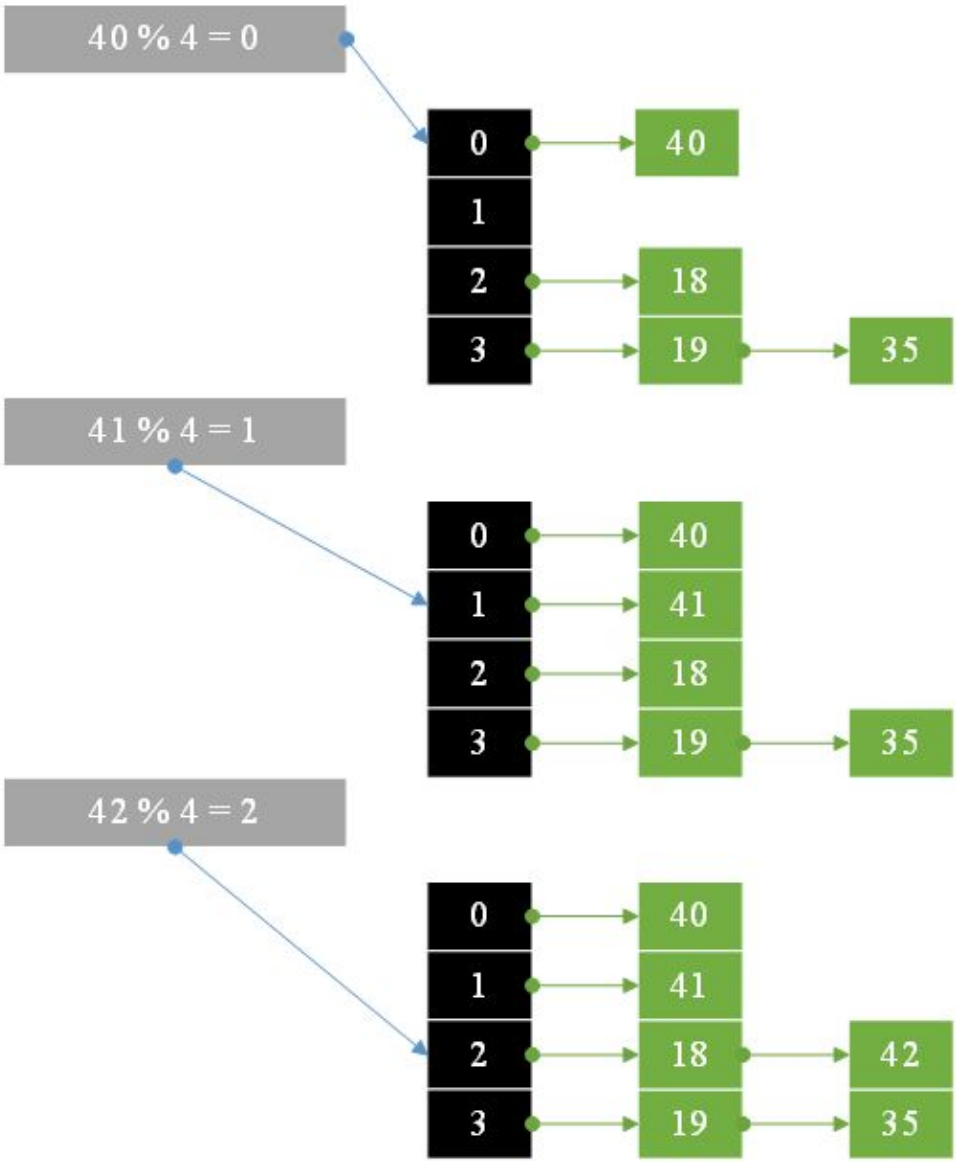
Resultado: Before building a heap : 10 , 90 , 20 , 80 , 30 , 70 , 40 , 60 , 50 ,
After building a heap : 90 , 80 , 70 , 60 , 30 , 20 , 40 , 10 , 50 ,
After heap sorting : 10 , 20 , 30 , 40 , 50 , 60 , 70 , 80 , 90 ,

Tabela de hash

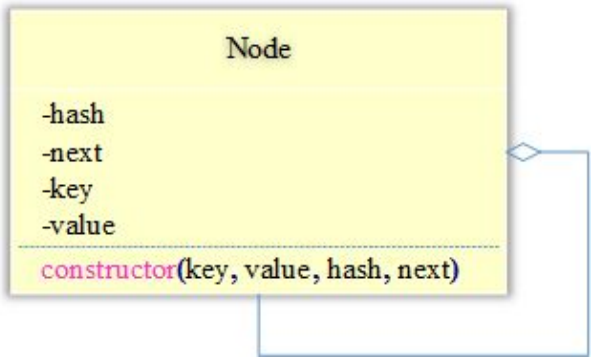
Tabela de hash (Hash Table): Uma tabela de hash é uma estrutura de dados que acessa diretamente um local de armazenamento na memória com base em uma chave. Em outras palavras, ele acessa os registros calculando uma função nos valores-chave, mapeando os dados a serem consultados para um local na tabela, o que acelera a pesquisa. Essa função de mapeamento é chamada de função hash, e a matriz que armazena os registros é chamada de tabela hash. **key => values**.

1. Mapeie {19, 18, 35,40,41,42} para a chave da regra de mapeamento key% 4


19 % 4 = 3
0
1
2
3
19
18 % 4 = 2
0
1
2
3
18
19
35 % 4 = 3
0
1
2
3
18
19
35


40 % 4 = 0
0
1
2
3
40
18
19
35
41 % 4 = 1
0
1
2
3
40
41
18
19
35
42 % 4 = 2
0
1
2
3
40
41
18
19
42
35

2. Hashtable



Crie um arquivo: TestHashtable.html

```
<script type="text/javascript"> class Node {
   constructor(key, value, hash, next){
   this.key = key; this.value = value; this.hash = hash; this.next = next; }

   getKey() {
   return this.key; }
   setKey(key) {
   this.key = key; }

   getValue() {
   return this.value; }
   setValue(value) {
   this.value = value; }

   getHash() {
   return this.hash; }
   setHash(hash) {
   this.hash = hash; }
   }
   class Hashtable {
   constructor(){
   this.capacity = 16; this.table = new Array(this.capacity); this.size = 0; }

   size() {
```

```
    return this.size; }
    isEmpty() {
    return this.size == 0 ? true : false; }

    hash(key) {
    var hash = 0; for (var i = 0; i < key.length; i++) {
    hash += key.charCodeAt(i); }
    return hash % 37; }

    // Calcular o valor do hash de acordo com o algoritmo de hash da chave
hashCode(key) {
    // política de hash para o método do quadrado médio var avg =
this.hash(key) * (Math.pow(5, 0.5) - 1) / 2
    var numeric = avg - Math.floor(avg) return
parseInt(Math.floor(numeric * this.table.length)) }

    get(key) {
    if (key == null) {
    return null; }
    var hash = this.hashCode(key); var node = this.table[hash]; while
(node != null) {// Obter value de acordo com key if (node.getKey() ==
key) {
    return node.getValue(); }
    node = node.next; }
    return null; }

    put(key, value) {
    if (key == null) {
    throw new IllegalArgumentException(); }

    var hash = this.hashCode(key); var newNode = new Node(key, value,
hash, null); var node = table[hash]; while (node != null) {
    if (node.getKey().equals(key)) {
    node.setValue(value); return; }
    node = node.next; }
    newNode.next = this.table[hash]; this.table[hash] = newNode;
this.size++; }
```

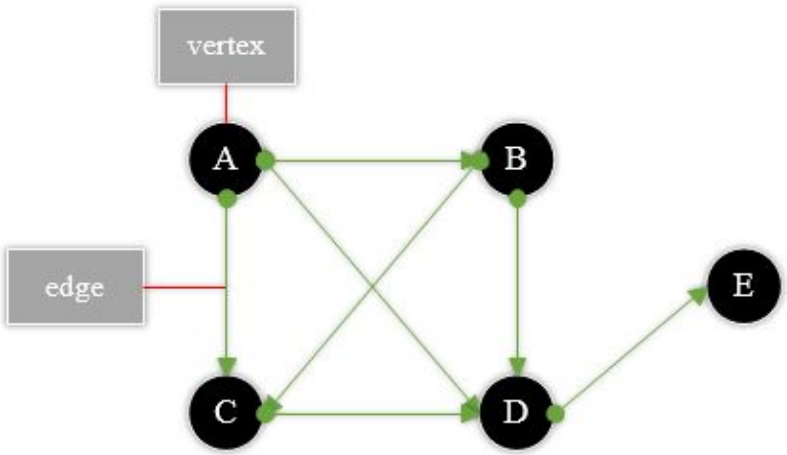
```
    }

    //////////////////////testing//////////////////

    var table = new Hashtable();
    table.put("david", "Good Boy Keep Going"); table.put("grace", "Cute Girl Keep Going");
    document.write("david => " + table.get("david") + "<br>");
document.write("grace => " + table.get("grace") + "<br>");
</script>
```

Resultado: david => Good Boy Keep Going grace => Cute Girl Keep Going

Gráfico direcionado e pesquisa em profundidade

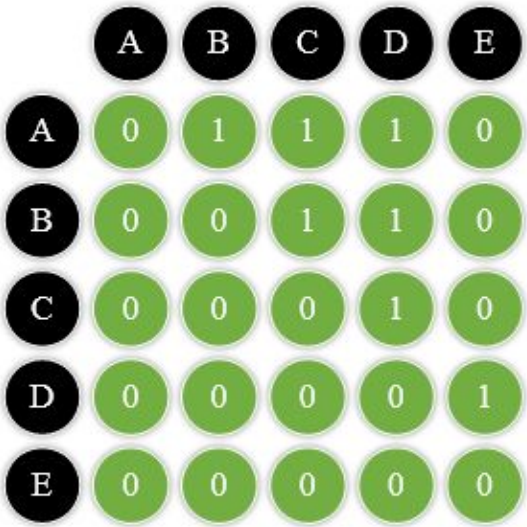
Gráfico direcionado (Directed Graph):
A estrutura de dados é representada por uma matriz de adjacência (ou seja, uma matriz bidimensional) e uma lista de adjacência. Cada nó é chamado de vértice (vertex) e dois nós adjacentes são chamados de arestas (edge).

O gráfico direcionado tem direção : A -> B e B -> A são diferentes



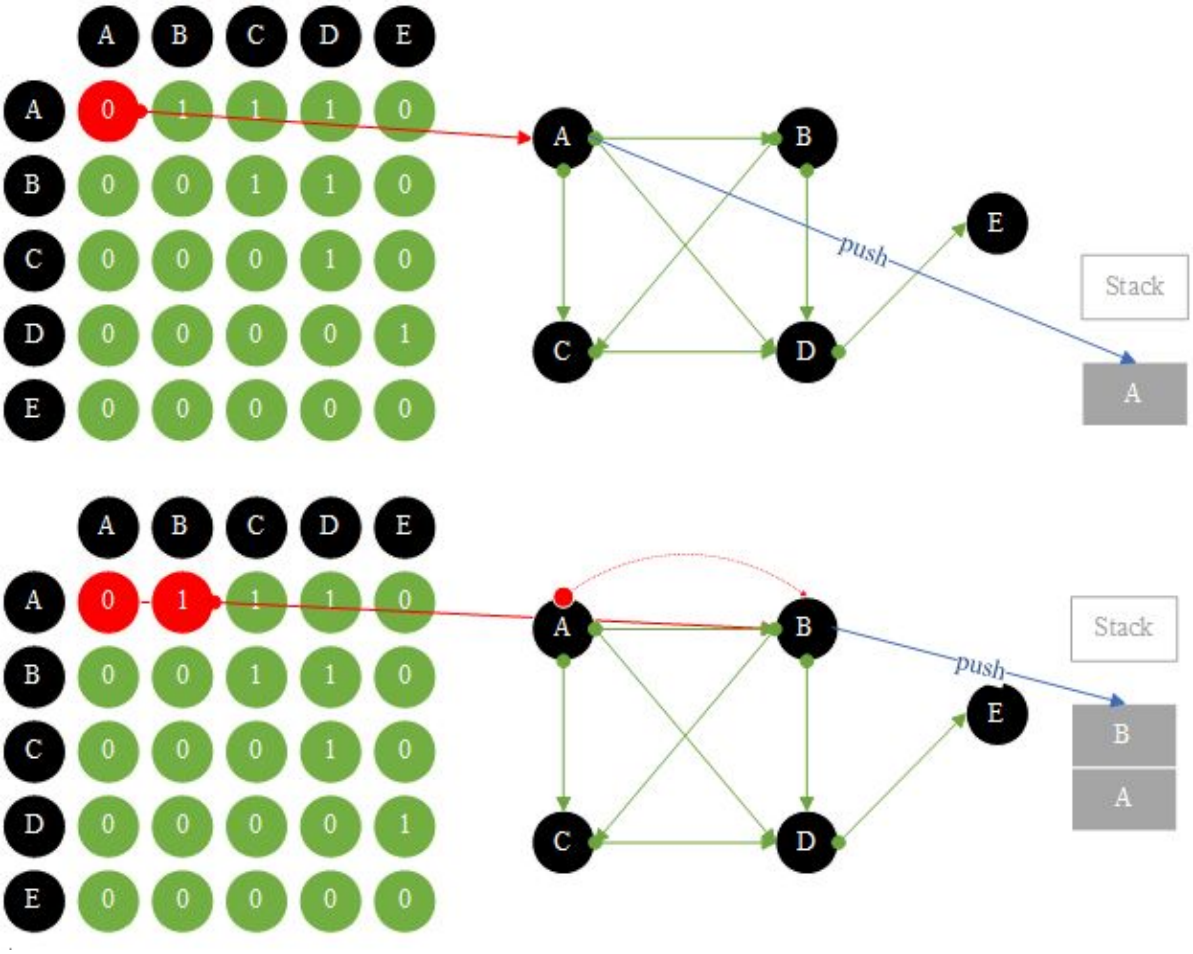
1. matriz de adjacência:

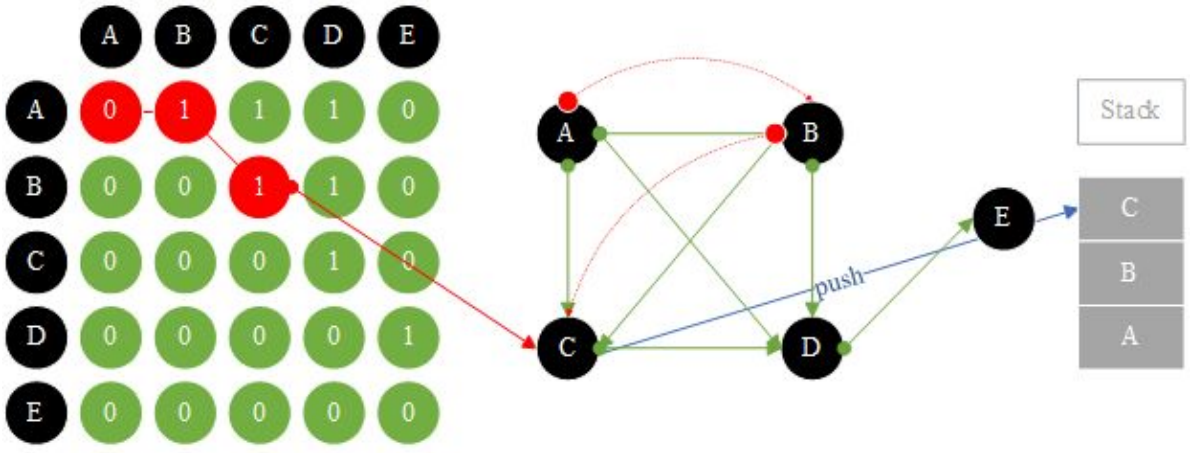
O número total de vértices é um tamanho de matriz bidimensional, se houver aresta de valor 1, caso contrário, aresta é 0.

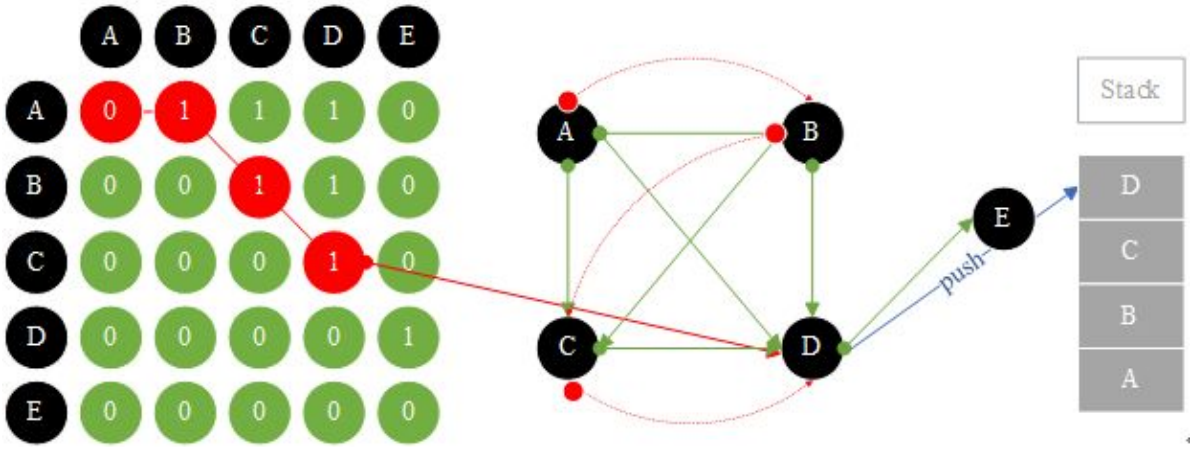

A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0

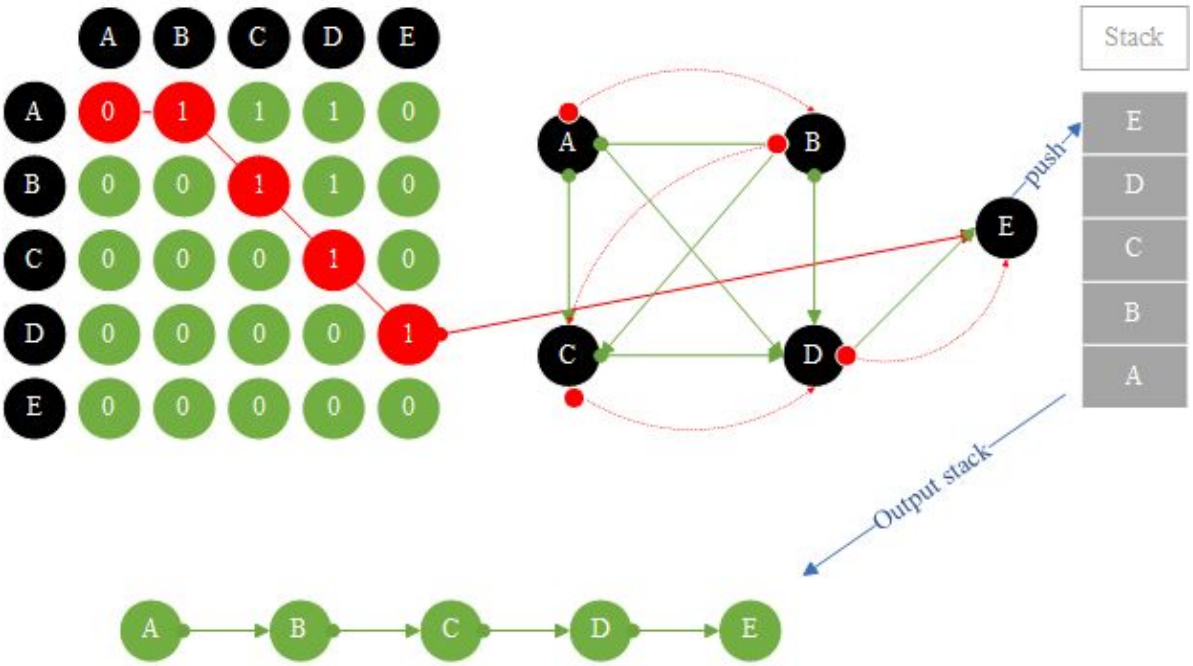
2. Primeira pesquisa de profundidade:

Procure o nó de borda vizinho B de A e, em seguida, encontre o nó vizinho C de B e assim por diante até que todos os nós sejam encontrados A -> B -> C -> D -> E.




A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
push
Stack
C
B
A


A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
push
Stack
D
C
B
A


A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
Stack
E
D
C
B
A
push
Output stack

A
B
C
D
E

Crie um arquivo: TestDirectedGraphDepthFrstSearch.html

```
<script type="text/javascript"> class Vertex {
   constructor(data, visited){
   this.data = data; this.visited = visited; // Você visitou }

   getData() {
   return this.data; }
   setData(data) {
   this.data = data; }

   isVisited() {
   return this.visited; }
   setVisited(visited) {
   this.visited = visited; }
   }
```

```
class Stack {
constructor(){
this.stacks = new Array(); this.top = -1; }

push(element) {
this.top++; this.stacks[this.top] = element; }

pop() {
if(this.top == -1){
return -1; }

var data = this.stacks[this.top]; this.top--; return data; }

peek() {
if(this.top == -1){
return -1; }

var data = this.stacks[this.top]; return data; }

isEmpty() {
if(this.top <= -1){
return true; }
return false; }
}
```

```
class Graph {
constructor(maxVertexSize){
this.maxVertexSize = maxVertexSize;// Tamanho bidimensional da
matriz this.vertexs = new Array(maxVertexSize); this.size = 0; // tamanho
do vértice this.adjacencyMatrix = new Array(maxVertexSize); for (var i
= 0; i < maxVertexSize; i++) {
```

```
   this.adjacencyMatrix[i] = new Array(); for (var j = 0; j <
maxVertexSize; j++) {
   this.adjacencyMatrix[i][j] = 0; }
   }
   this.stack = new Stack();//salva vértices atuais }

   addVertex(data) {
   var vertex = new Vertex(data, false); this.vertexs[this.size++] = vertex;
}

   addEdge(from, to) {
   this.adjacencyMatrix[from][to] = 1; // A -> B e B -> A são diferentes }

   depthFirstSearch() {
   var firstVertex = this.vertexs[0]; // Comece a pesquisar a partir do
primeiro vértice firstVertex.setVisited(true);
document.write(firstVertex.getData()); this.stack.push(0); while
(!this.stack.isEmpty()) {
   var row = this.stack.peek(); // Obter posições de vértice adjacentes que
não foram visitadas var col = this.findAdjacencyUnVisitedVertex(row); if
(col == -1) {
   this.stack.pop(); } else {
   this.vertexs[col].setVisited(true); document.write(" -> " +
this.vertexs[col].getData()); this.stack.push(col); }
   }
   this.clear(); }

   // Obter posições de vértice adjacentes que não foram visitadas
findAdjacencyUnVisitedVertex(row) {
   for (var col = 0; col < this.size; col++) {
   if (this.adjacencyMatrix[row][col] == 1 &&
!this.vertexs[col].isVisited()) {
   return col; }
   }
   return -1; }

   clear() {
```

```
   for (var i = 0; i < this.size; i++) {
   this.vertexs[i].setVisited(false); }
   }

   getAdjacencyMatrix() {
   return this.adjacencyMatrix; }

   getVertexs() {
   return this.vertexs; }
   }

   function printGraph(graph) {
   document.write("Two-dimensional array traversal output vertex edge
and adjacent array : <br>"); document.write("   ");
for (var i = 0; i < graph.getVertexs().length; i++) {
   document.write(graph.getVertexs()[i].getData() +
"    "); }
   document.write("<br>");
   for (var i = 0; i < graph.getAdjacencyMatrix().length; i++) {
   document.write(graph.getVertexs()[i].getData() + " "); for (var j = 0; j
< graph.getAdjacencyMatrix().length; j++) {
   document.write(graph.getAdjacencyMatrix()[i][j] +
"    "); }
   document.write("<br>"); }
   }

   var graph = new Graph(5); graph.addVertex("A");
graph.addVertex("B"); graph.addVertex("C"); graph.addVertex("D");
graph.addVertex("E");
   graph.addEdge(0, 1); graph.addEdge(0, 2); graph.addEdge(0, 3);
graph.addEdge(1, 2); graph.addEdge(1, 3); graph.addEdge(2, 3);
graph.addEdge(3, 4);
   printGraph(graph); document.write("<br>Depth-first search traversal
output : <br>"); graph.depthFirstSearch(); </script>
```

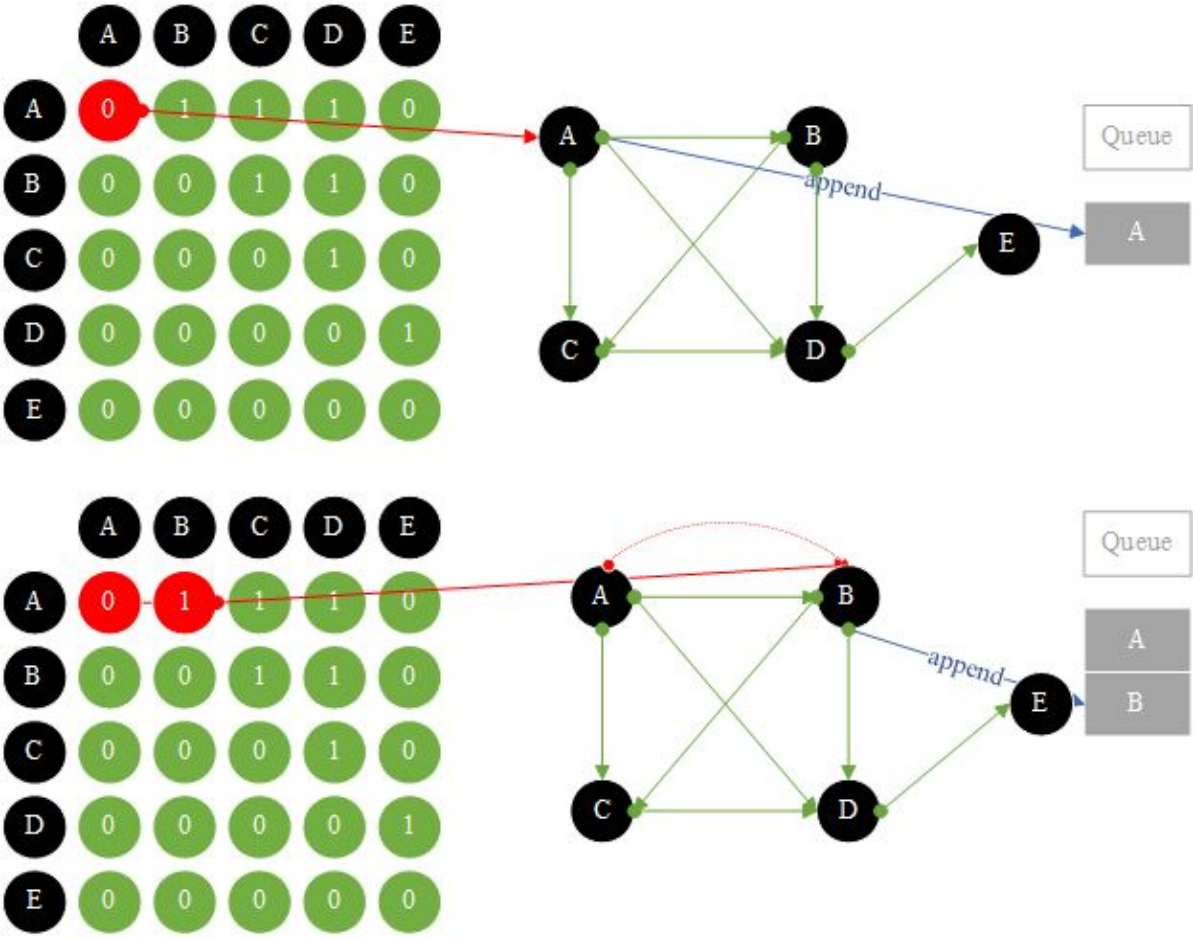
Resultado:

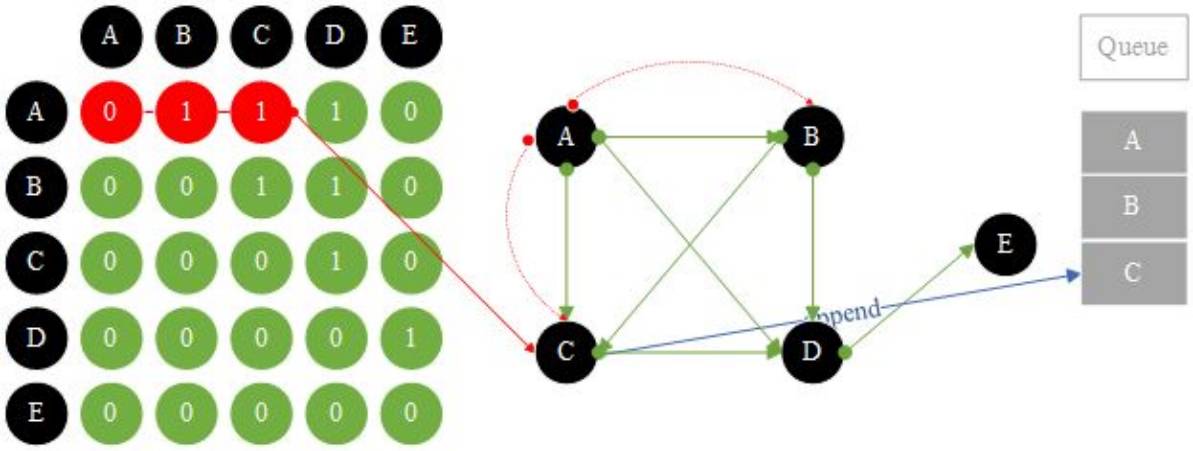
A B C D E

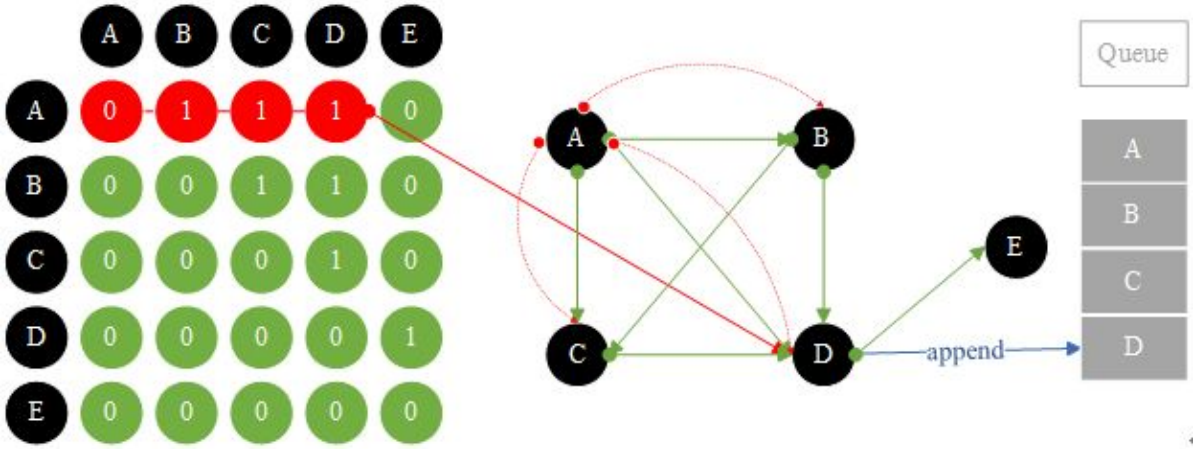
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
Depth-first search traversal output :A -> B -> C -> D -> E

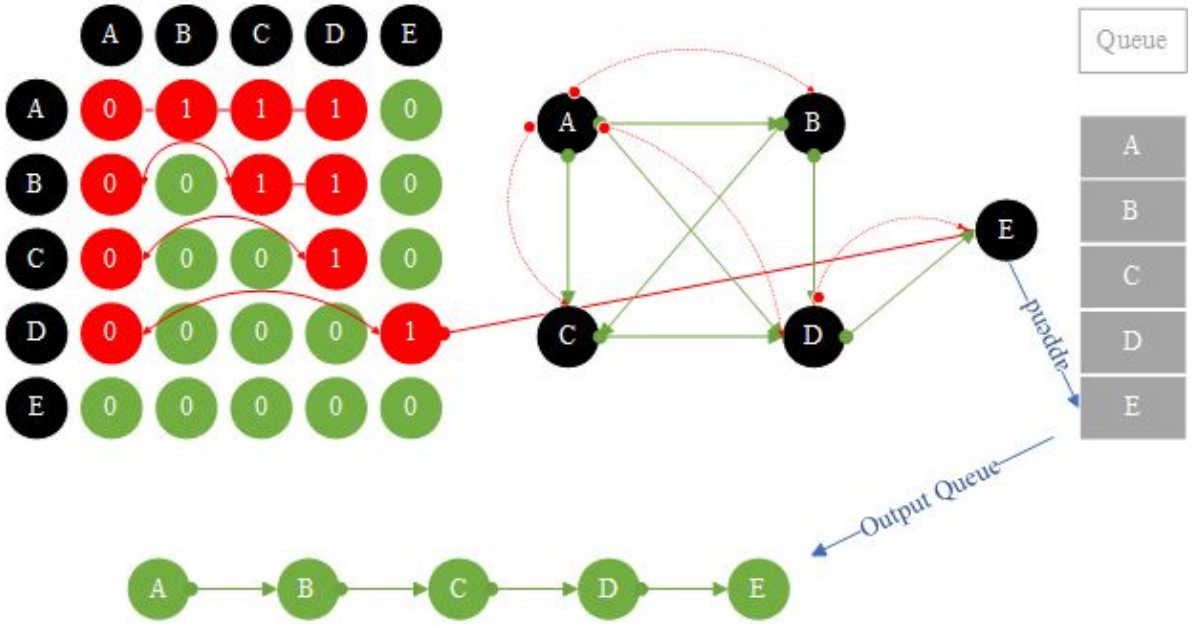
Gráfico dirigido e Primeira pesquisa de largura

Primeira pesquisa de largura (Breadth-First Search): Encontre todos os nós de borda vizinhos B, C, D de A e, em seguida, encontre todos os nós vizinhos A, C, D de B e assim por diante até que todos os nós sejam encontrados A -> B -> C -> D -> E.




A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
Queue
A
B
C
append


A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
Queue
A
B
C
D
append


A B C D E
A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
A
B
C
D
E
Queue
A
B
C
D
E
append
Output Queue
A
B
C
D
E

Crie um arquivo: TestDirectedGraphBreadthFrstSearch.html

```
<script type="text/javascript"> class Vertex {
   constructor(data, visited){
   this.data = data; this.visited = visited; // Você visitou }

   getData() {
   return this.data; }
   setData(data) {
   this.data = data; }

   isVisited() {
   return this.visited; }
   setVisited(visited) {
   this.visited = visited; }
   }
```

```
class Queue {
constructor(){
this.queues = new Array(); }

add(element) {
this.queues[this.queues.length] = element; }

remove() {
if(this.queues.length <= 0){
return -1; }
return this.queues.shift(); }

isEmpty() {
if(this.queues.length <= 0){
return true; }
return false; }
}
```

```
class Graph {
constructor(maxVertexSize){
this.maxVertexSize = maxVertexSize;// Tamanho bidimensional da
matriz this.vertexs = new Array(maxVertexSize); this.size = 0; // tamanho
do vértice this.adjacencyMatrix = new Array(maxVertexSize); for (var i
= 0; i < maxVertexSize; i++) {
this.adjacencyMatrix[i] = new Array(); for (var j = 0; j <
maxVertexSize; j++) {
this.adjacencyMatrix[i][j] = 0; }
}
this.queue = new Queue();//salva vértices atuais }

addVertex(data) {
var vertex = new Vertex(data, false); this.vertexs[this.size] = vertex;
this.size++; }

addEdge(from, to) {
// A -> B e B -> A são diferentes this.adjacencyMatrix[from][to] = 1; }

getAdjacencyMatrix() {
return this.adjacencyMatrix; }

getVertexs() {
return this.vertexs; }









breadthFirstSearch() {
```

```
   // Comece a pesquisar a partir do primeiro vértice var firstVertex =
this.vertexs[0]; firstVertex.setVisited(true);
document.write(firstVertex.getData()); this.queue.add(0);
   var col = 0; while (!this.queue.isEmpty()) {
   var head = this.queue.remove(); // Obter posições de vértice adjacentes
que não foram visitadas col = this.findAdjacencyUnVisitedVertex(head);
   // Passa por todos os vértices conectados ao vértice atual while (col !=
-1) {
   this.vertexs[col].setVisited(true); document.write(" -> " +
this.vertexs[col].getData()); this.queue.add(col); col =
this.findAdjacencyUnVisitedVertex(head); }
   }
   this.clear(); }

   // Obter posições de vértice adjacentes que não foram visitadas
findAdjacencyUnVisitedVertex(row) {
   for (var col = 0; col < this.size; col++) {
   if (this.adjacencyMatrix[row][col] == 1 &&
!this.vertexs[col].isVisited()) {
   return col; }
   }
   return -1; }

   clear() {
   for (var i = 0; i < this.size; i++) {
   this.vertexs[i].setVisited(false); }
   }
   }


   function printGraph(graph) {
   for (var i = 0; i < graph.getVertexs().length; i++) {
   document.write(graph.getVertexs()[i].getData() +
"    "); }
   document.write("<br>");
   for (var i = 0; i < graph.getAdjacencyMatrix().length; i++) {
```

```
   document.write(graph.getVertexs()[i].getData() + " "); for (var j = 0; j
< graph.getAdjacencyMatrix().length; j++) {
   document.write(graph.getAdjacencyMatrix()[i][j] +
"    "); }
   document.write("<br>"); }
   }

   var graph = new Graph(5); graph.addVertex("A");
graph.addVertex("B"); graph.addVertex("C"); graph.addVertex("D");
graph.addVertex("E");
   graph.addEdge(0, 1); graph.addEdge(0, 2); graph.addEdge(0, 3);
graph.addEdge(1, 2); graph.addEdge(1, 3); graph.addEdge(2, 3);
graph.addEdge(3, 4);
   printGraph(graph);
   document.write("\nBreadth-First Search traversal output : <br>");
graph.breadthFirstSearch(); </script>
```

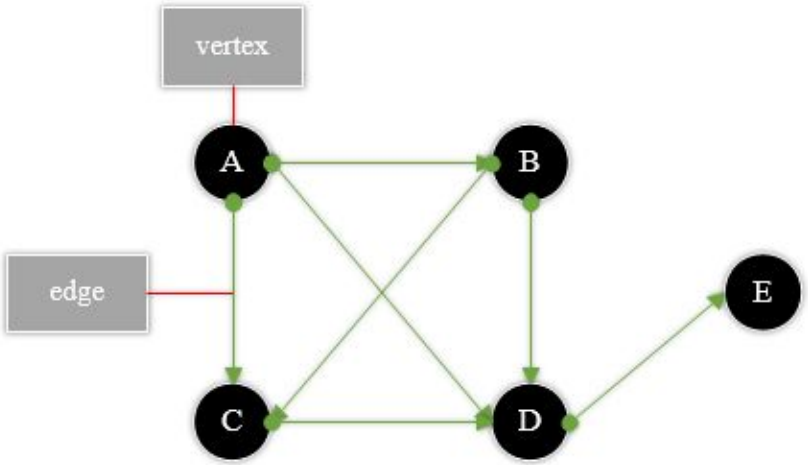
Resultado:

A B C D E

A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0
Breadth-First Search traversal output :A -> B -> C -> D -> E

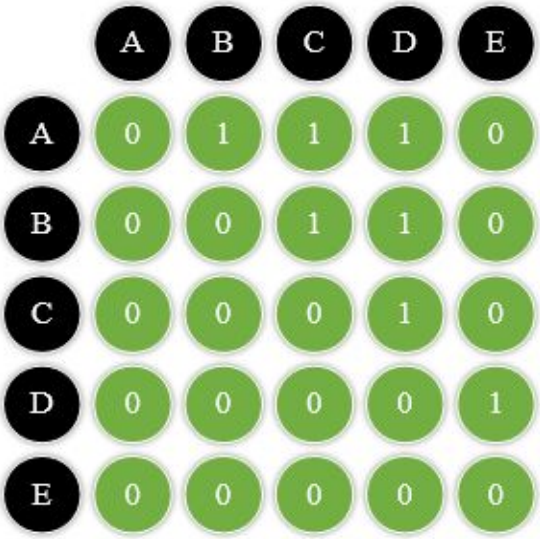
Ordenar topológica

Ordenar topológica (Topological Sorting): Classifique os vértices no gráfico direcionado com a ordem de direção .
O gráfico direcionado tem direção : A -> B e B -> A são diferentes

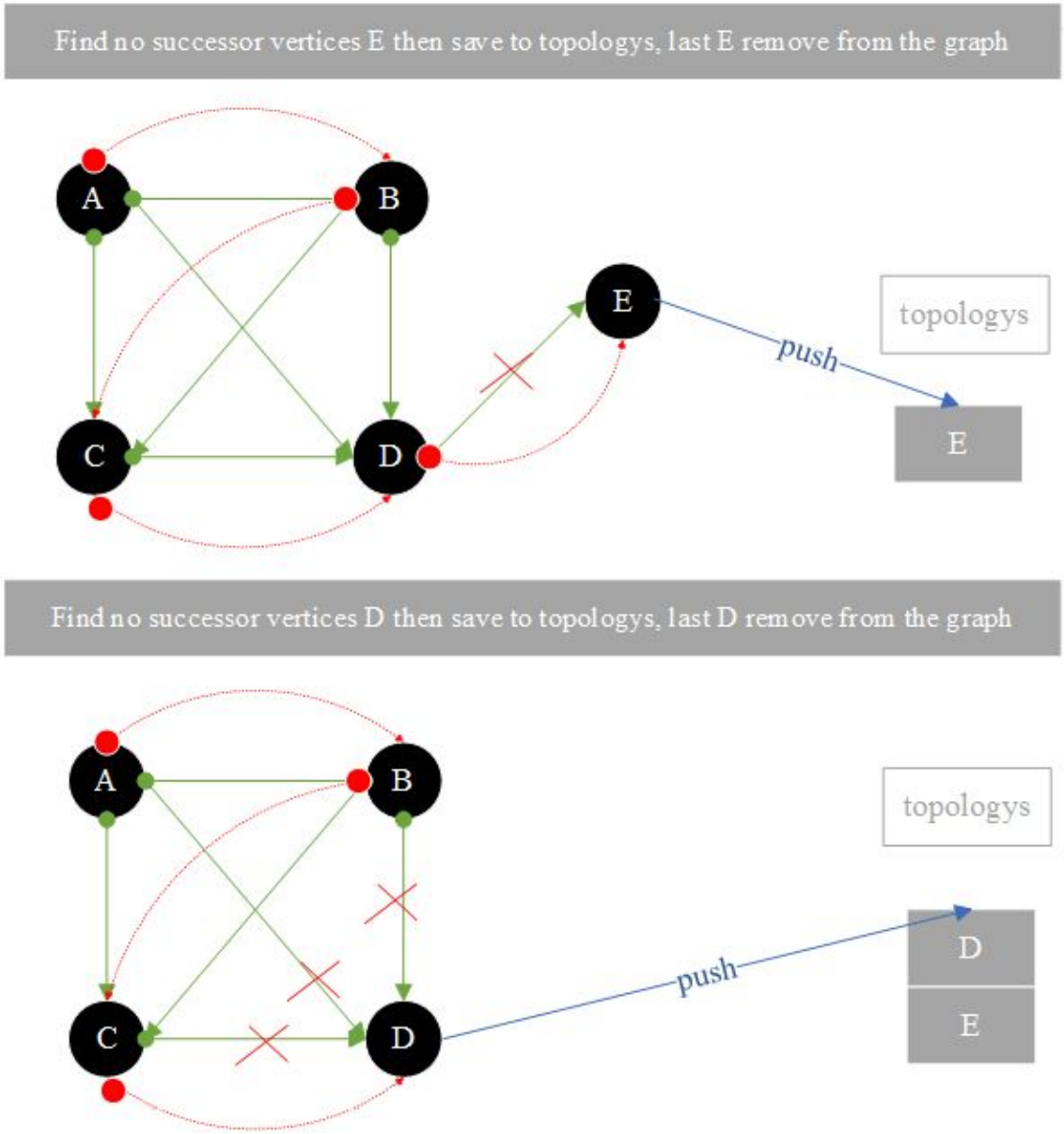


1. matriz de adjacência:

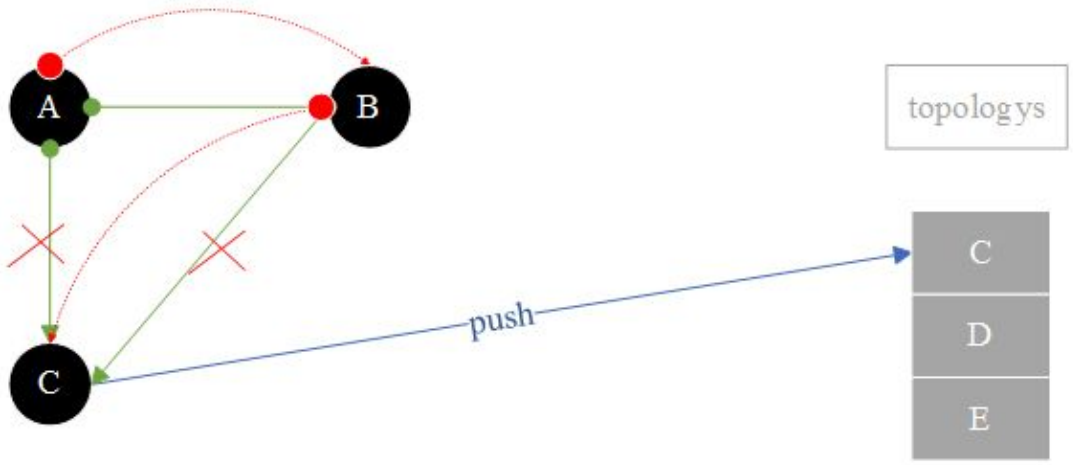
O número total de vértices é um tamanho de matriz bidimensional, se houver aresta de valor 1, caso contrário, aresta é 0.



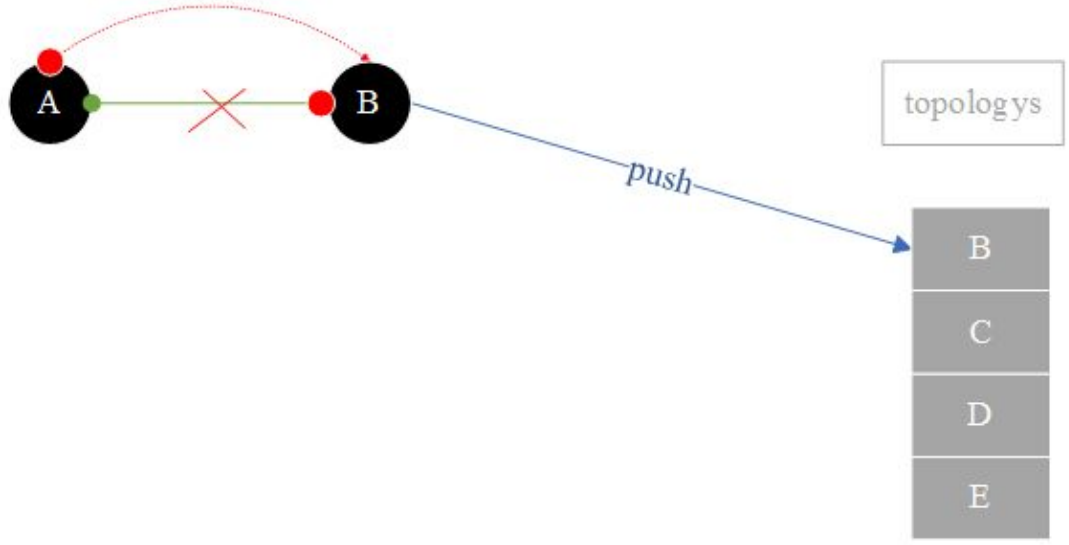
Ordenar topológica a partir do vértice A : A -> B -> C -> D -> E



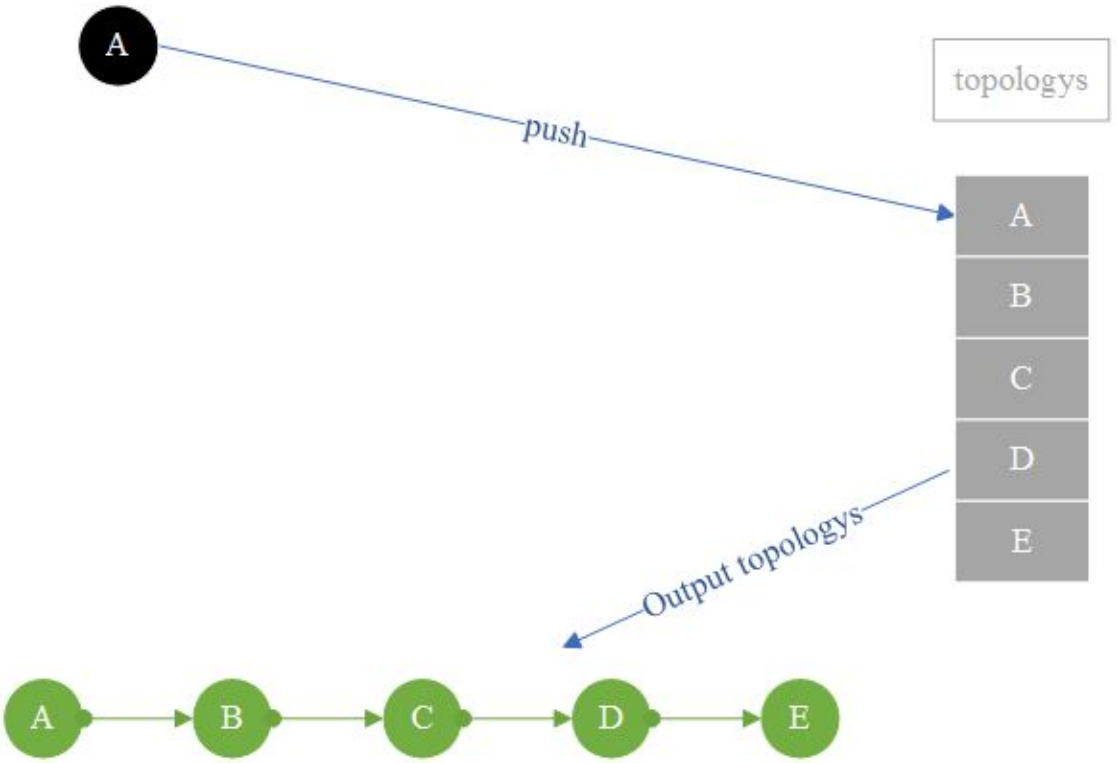
Find no successor vertices C then save to topologys, last C remove from the graph



Find no successor vertices C then save to topologys, last C remove from the graph



Find no successor vertices C then save to topologys, last C remove from the graph



Crie um arquivo: TestTopologicalSorting.html

```
<script type="text/javascript"> class Vertex {
   constructor(data, visited){
   this.data = data; this.visited = visited; // Você visitou }

   getData() {
   return this.data; }
   setData(data) {
   this.data = data; }

   isVisited() {
   return this.visited; }
   setVisited(visited) {
   this.visited = visited; }
   }
```

```
   class Graph {
   constructor(maxVertexSize){
   this.maxVertexSize = maxVertexSize;// Tamanho bidimensional da
matriz this.vertexs = new Array(maxVertexSize); this.topologys = new
Array(); this.adjacencyMatrix = new Array(maxVertexSize); for (var i =
0; i < maxVertexSize; i++) {
   this.adjacencyMatrix[i] = new Array(); for (var j = 0; j <
maxVertexSize; j++) {
   this.adjacencyMatrix[i][j] = 0; }
   }
   this.size = 0; }

   addVertex(data) {
   var vertex = new Vertex(data, false); this.vertexs[this.size] = vertex;
this.size++; }

   addEdge(from, to) {
   // A -> B e B -> A são diferentes this.adjacencyMatrix[from][to] = 1; }

   topologySort() {
   while (this.size > 0) {
   //Obter nó que nenhum nó sucessor var noSuccessorVertex =
this.getNoSuccessorVertex(); if (noSuccessorVertex == -1) {
   document.write("There is ring in Graph "); return; }
   // Copie o nó excluído para a matriz classificada this.topologys[this.size
- 1] = this.vertexs[noSuccessorVertex]; // Excluir nó que não possui nó
sucessor this.removeVertex(noSuccessorVertex); }
   }




   getNoSuccessorVertex() {
   var existSuccessor = false; for (var row = 0; row < this.size; row++) {
   existSuccessor = false; // Se o nó tiver uma linha fixa, cada coluna terá
um 1, indicando que o nó possui um sucessor, finalizando o loop for (var
col = 0; col < this.size; col++) {
```

```
    if (this.adjacencyMatrix[row][col] == 1) {
    existSuccessor = true; break; }
    }

    if (!existSuccessor) {// Se o nó não tiver sucessor, retorne seu índice
return row; }
    }
    return -1; }

    removeVertex(vertex) {
    if (vertex != this.size - 1) { // Se o vértice é o último elemento, o final //
Os vértices são removidos da matriz de vértices for (var i = vertex; i <
this.size - 1; i++) {
    this.vertexs[i] = this.vertexs[i + 1]; }

    for (var row = vertex; row < this.size - 1; row++) {// Mover uma linha
para cima for (var col = 0; col < this.size - 1; col++) {
    this.adjacencyMatrix[row][col] = this.adjacencyMatrix[row + 1][col]; }
    }

    for (var col = vertex; col < this.size - 1; col++) {// Mova uma linha
para a esquerda for (var row = 0; row < this.size - 1; row++) {
    this.adjacencyMatrix[row][col] = this.adjacencyMatrix[row][col + 1]; }
    }
    }
    this.size--;// Diminuir o número de vértices }



    clear() {
    for (var i = 0; i < this.size; i++) {
    this.vertexs[i].setVisited(false); }
    }

    getAdjacencyMatrix() {
    return this.adjacencyMatrix; }
```

```
   getVertexs() {
   return this.vertexs; }

   getTopologys() {
   return this.topologys; }
   }

   //////////////////////testing//////////////////

   function printGraph(graph) {
   document.write("Two-dimensional array traversal output vertex edge and adjacent array : <br>"); document.write("   ");
for (var i = 0; i < graph.getVertexs().length; i++) {
   document.write(graph.getVertexs()[i].getData() + "    "); }
   document.write("<br>");
   for (var i = 0; i < graph.getAdjacencyMatrix().length; i++) {
   document.write(graph.getVertexs()[i].getData() + " "); for (var j = 0; j < graph.getAdjacencyMatrix().length; j++) {
   document.write(graph.getAdjacencyMatrix()[i][j] + "    "); }
   document.write("<br>"); }
   }



   var graph = new Graph(5); graph.addVertex("A");
graph.addVertex("B"); graph.addVertex("C"); graph.addVertex("D");
graph.addVertex("E");
   graph.addEdge(0, 1); graph.addEdge(0, 2); graph.addEdge(0, 3);
graph.addEdge(1, 2); graph.addEdge(1, 3); graph.addEdge(2, 3);
graph.addEdge(3, 4);
   printGraph(graph);
   document.write("Directed Graph Topological Sorting:<br>");
graph.topologySort();
   for(var i=0;i< graph.getTopologys().length ; i++){
```

```
    document.write(graph.getTopologys()[i].getData()+" -> "); }
</script>
```

Resultado: Two-dimensional array traversal output vertex edge and adjacent array :

A B C D E

A 0 1 1 1 0
B 0 0 1 1 0
C 0 0 0 1 0
D 0 0 0 0 1
E 0 0 0 0 0

Directed Graph Topological Sorting: A -> B -> C -> D -> E ->

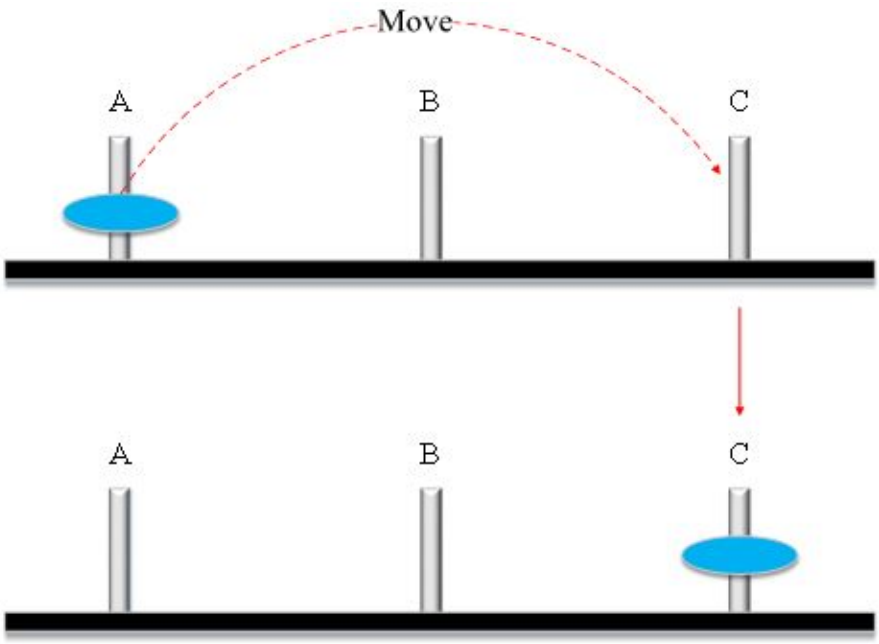
Torres de Hanói
(Towers of Hanoi)

A Torre de Hanói que um francês M. Claus (Lucas) da Tailândia para a França em 1883. Torre de Hanói que é suportada por três pilares de diamante. No início, Deus colocou 64 discos de ouro de cima para baixo no primeiro Pilar. Deus ordenou aos monges que movessem todos os discos de ouro do primeiro para o terceiro Pilar. O princípio de placas grandes sob placas pequenas durante o processo de manuseio. Se apenas uma placa for movida diariamente, a torre será destruída. quando todos os discos são movidos, isso é o fim do mundo.

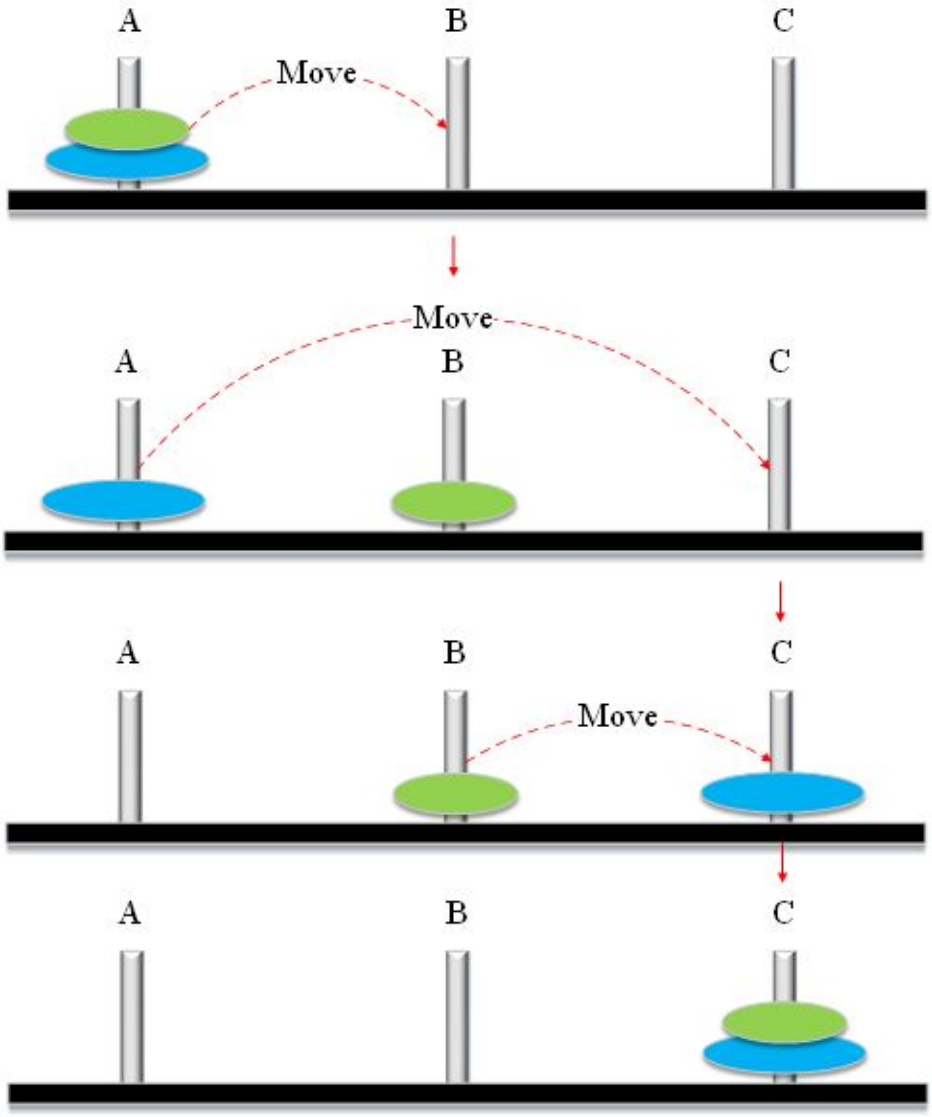
Vamos transformar essa história em um algoritmo:

Marque as três colunas como ABC.
1. Se houver apenas um disco, mova-o diretamente para C (A->C).
2. Quando houver dois discos, use B como auxiliar (A->B, A->C,B->C).
3. Se houver mais de dois discos, use B como um auxiliar (A->B, A->C,B->C), e continuar o processo recursivo.

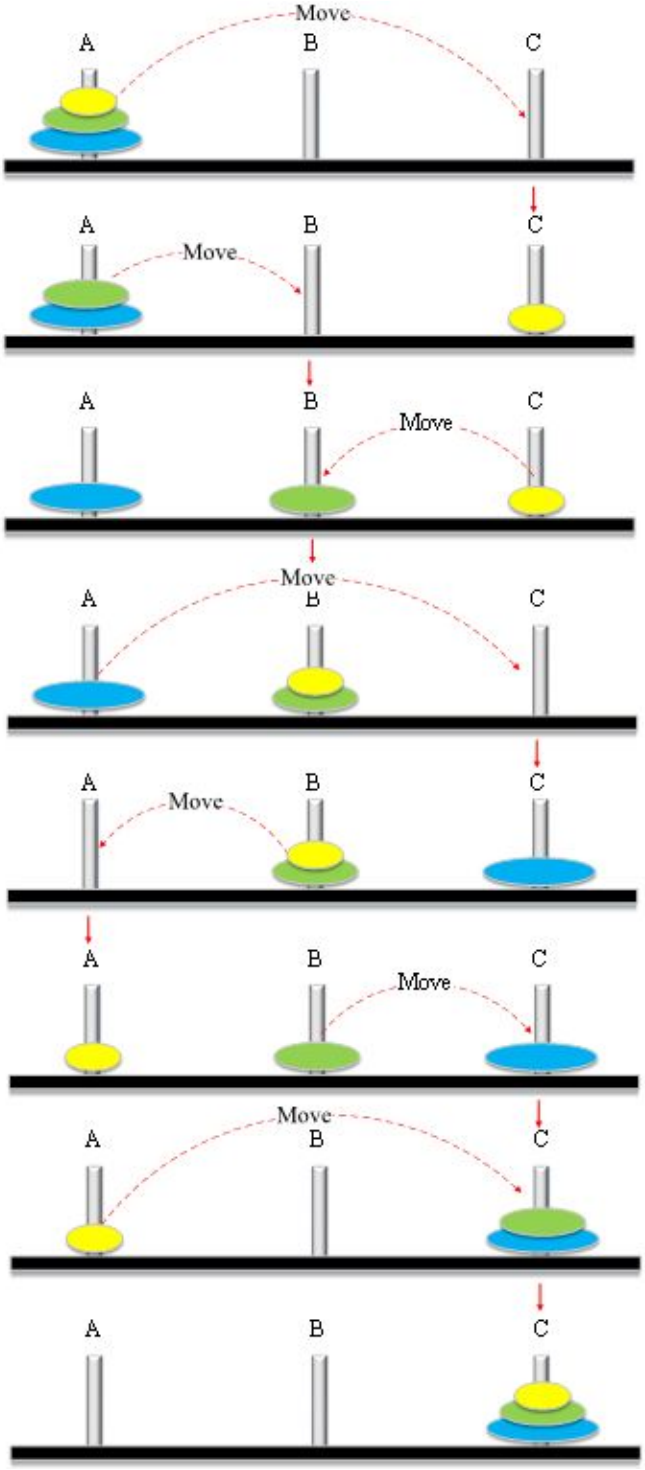
1. Se houver apenas um disco, mova-o diretamente para C (A->C).


Move
A
B
C
A
B
C

2. Quando houver dois discos, use B como auxiliar (A->B, A->C,B->C).



3. Se houver mais de dois discos, use B como auxiliar e continue com o processo recursivo.



1. Crie um TowersofHanoi.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript"> function hanoi(n, A, B, C) {
   if (n == 1) {
   document.write("Move " + n + " "+A + " to " + C + "<br>"); } else {
   hanoi(n - 1, A, C, B); // Mova o n-1º disco de A a C para B
   document.write("Move " + n + " from " + A + " to " + C + "<br>");
hanoi(n - 1, B, A, C); // Mova o n-1º disco de A a C para B
   }
   }
   ////////////////////testing//////////////////
   document.write("Please enter the number of discs 1 <br>"); var n = 1;
hanoi(n, 'A', 'B', 'C');
   document.write("Please enter the number of discs 2 <br>"); var n = 2;
hanoi(n, 'A', 'B', 'C');
   document.write("Please enter the number of discs 3 <br>"); var n = 3;
hanoi(n, 'A', 'B', 'C'); </script>
```

Resultado: Please enter the number of discs :

1

Move 1 A to C

Resultado Executar novamente: Please enter the number of discs :

2

Move 1 A to B

Move 2 from A to C
Move 1 B to C
Resultado Executar novamente: Please enter the number of discs :

3

Move 1 A to C

Move 2 from A to B
Move 1 C to B
Move 3 from A to C
Move 1 B to A Move 2 from B to C
Move 1 A to C

Fibonacci

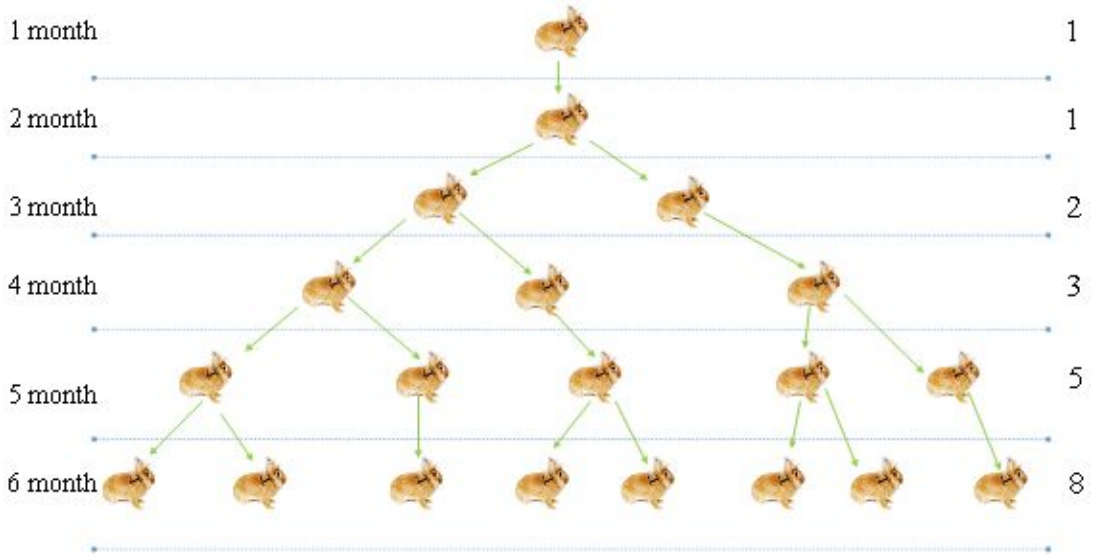
Fibonacci : um matemático europeu nos anos 1200, em seus escritos: "Se houver um coelho Depois de um mês, pode nascer um coelho recém-nascido. No início havia apenas 1 coelho, após um mês ainda 1 coelho. após dois meses, 2 coelhos, e após três meses, há 3 coelhos por exemplo: 1, 1, 2, 3, 5, 8, 13 ...

Definição de Fibonacci:

if n = 0, 1

fn = n if n > 1

fn = fn-1 + fn-2



1. Crie um Fibonacci.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript"> function fibonacci(n) {
   if (n == 1 || n == 2) {
   return 1; } else {
   return fibonacci(n - 1) + fibonacci(n - 2); }
   }

   ////////////////////testing//////////////////
   document.write("Please enter the number of month : 7 <br>"); var
number = 7;
   for (var i = 1; i <= number; i++) {
   document.write(i + " month   " + fibonacci(i) + "<br>"); }
</script>
```

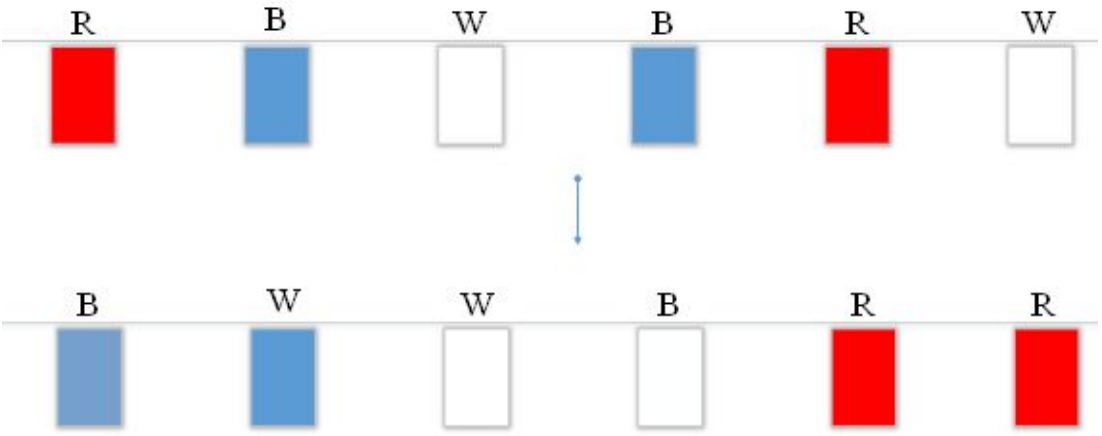
Resultado: Please enter the number of month :

7

1 month 1
2 month 1
3 month 2
4 month 3
5 month 5
6 month 8
7 month 13

Dijkstra

A bandeira tricolor foi hasteada originalmente por E.W. Dijkstra, que usava a bandeira nacional holandesa (Dijkstra é holandesa).
Suponha que haja uma corda com bandeiras vermelha, branca e azul. No início, todas as bandeiras na corda não estão em ordem. Você precisa organizá-los na ordem de azul -> branco -> vermelho. Como movê-los com o mínimo de vezes. você só faz isso na corda e apenas troca duas bandeiras de cada vez.

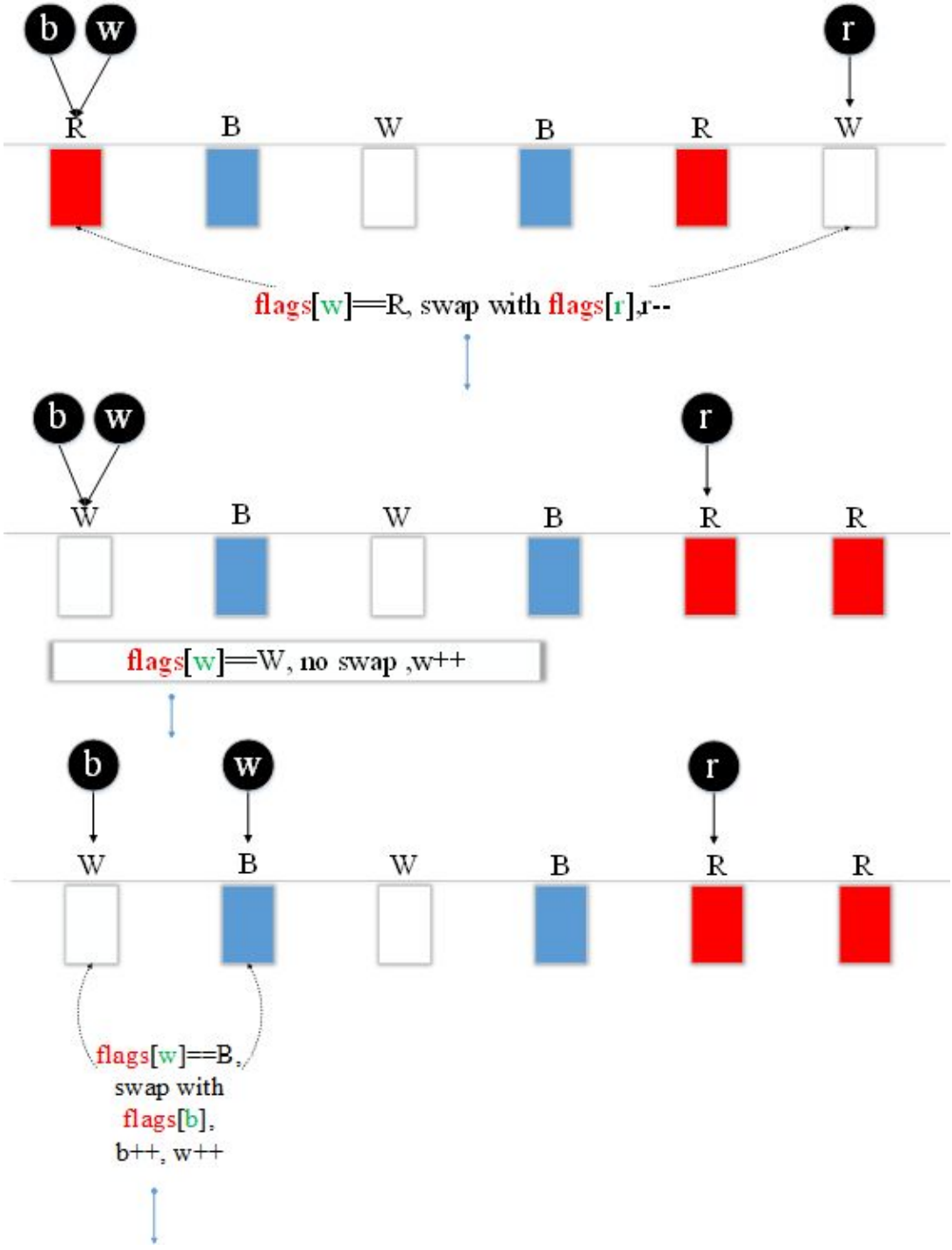


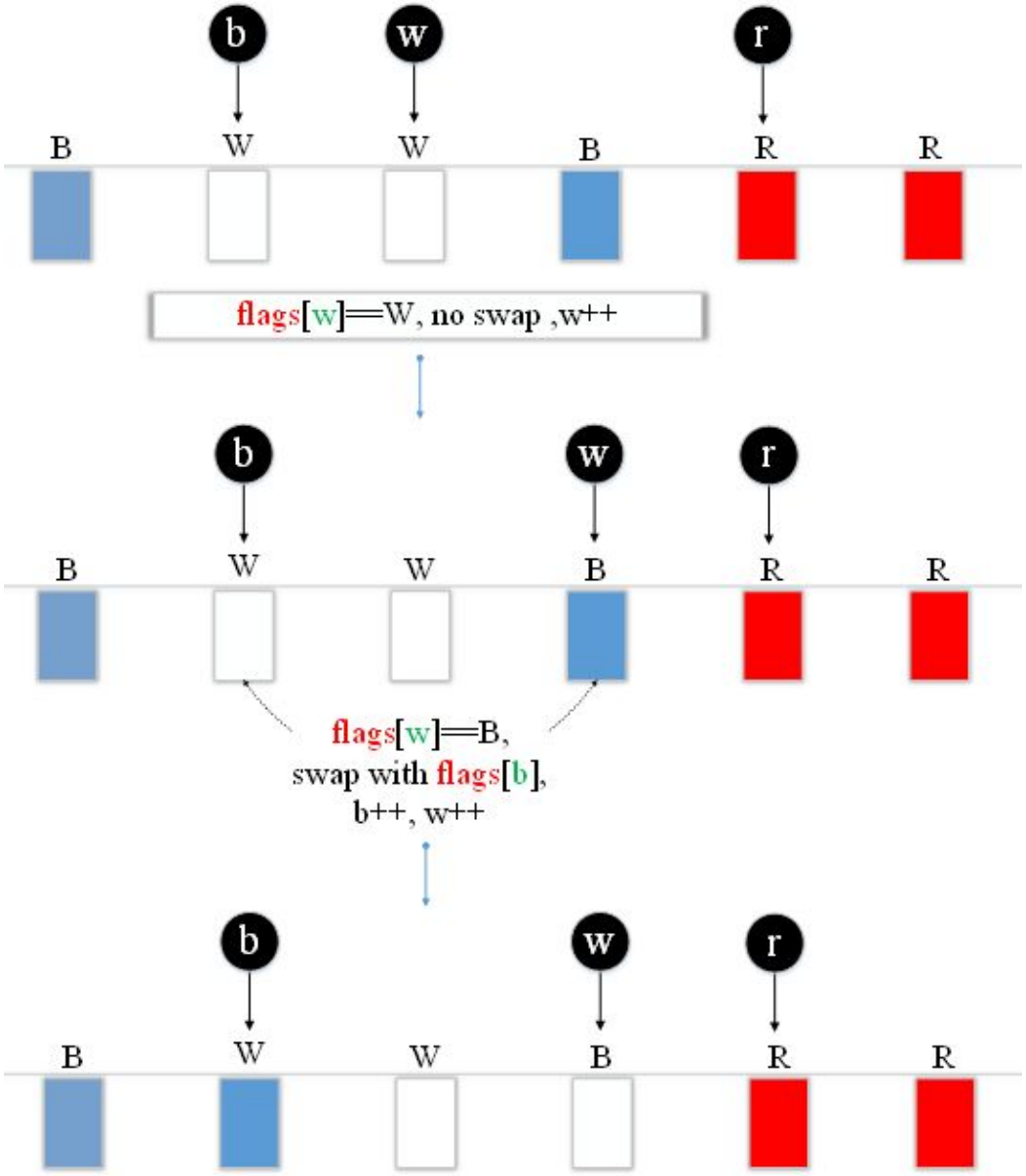
Solução: Use as matrizes de char para armazenar os sinalizadores. Por exemplo, b, w e r indicam a posição das bandeiras **azul**, **branca** e **vermelha**. O início de b e w é 0 da matriz e r está no final da matriz.

(1) Se a posição de w for uma bandeira azul, flags[w] são trocadas por flags[b]. E whiteIndex e b é movido para trás em 1.
(2) Se a posição de w for uma bandeira branca, w com move para trás em 1.
(3) Se a posição de w for uma bandeira vermelha, as flags[w] trocam com as flags[r]. r avança em 1.

No final, as bandeiras na frente de b são todas azuis e as bandeiras atrás de r são todas vermelhas.

Solução Gráfica




b
w
r
B
W
W
B
R
R
flags[w]==W, no swap ,w++
b
w
r
B
W
W
B
R
R
flags[w]==B,
swap with flags[b],
b++, w++
b
w
r
B
W
W
B
R
R

1. Crie um Dijkstra.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript"> function dijkstra(flags) {
   var b = 0, w = 0, r = flags.length - 1; var count = 0; while (w <= r) {
   if (flags[w] == 'W') {
   w++; } else if (flags[w] == 'B') {
   var temp = flags[w]; flags[w] = flags[b]; flags[b] = temp; w++; b++;
count++; } else if (flags[w] == 'R') {
   var m = flags[w]; flags[w] = flags[r]; flags[r] = m; r--; count++; }
   }
   }

   //////////////////////testing//////////////////////

   var flags = [ 'R', 'B', 'W', 'B', 'R', 'W' ];
   dijkstra(flags);
   for (var i = 0; i < flags.length; i++) {
   document.write(flags[i]); }
</script>
```

Resultado:

BBWWRR

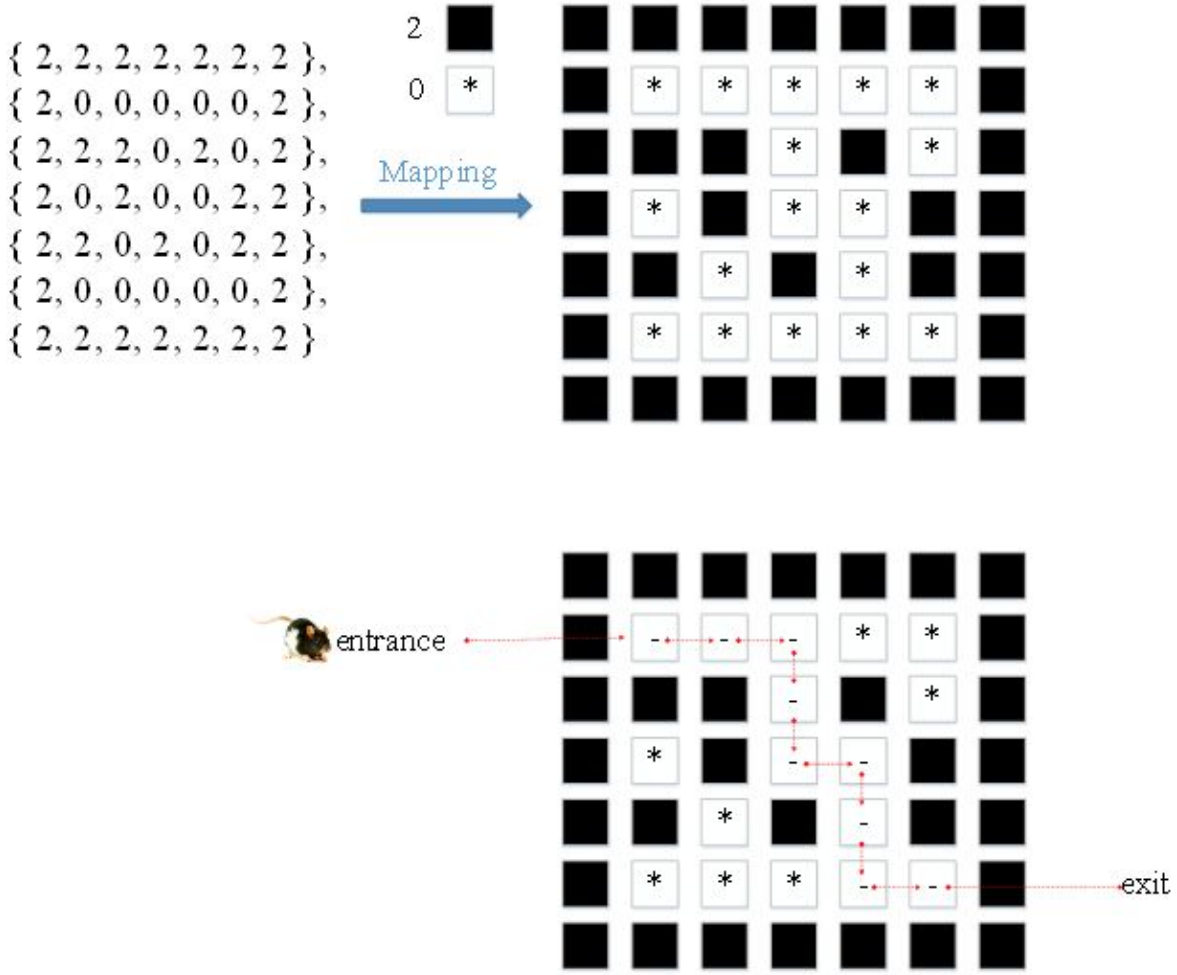
The total exchange count : 4

Labirinto de caminhada do rato

(Mouse Walking Maze) Mouse Walking Maze é um tipo básico de solução recursiva. Usamos 2 para representar a parede em uma matriz bidimensional e usamos 1 para representar o caminho do mouse e tentamos encontrar o caminho da entrada até a saída.

Solução: O mouse se move em quatro direções: para cima, para a esquerda, para baixo e para a direita. se bater na parede, volte e selecione a próxima direção para frente, então teste as quatro direções na matriz até que o mouse alcance a saída.

Solução Gráfica



1. Crie um MouseWalkingMaze.html com o Bloco de notas e abra-o em seu navegador.

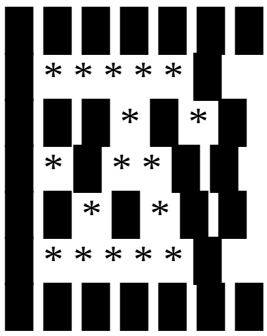
```
<script type="text/javascript">
   var maze = [
   [ 2, 2, 2, 2, 2, 2, 2 ],
   [ 2, 0, 0, 0, 0, 0, 2 ],
   [ 2, 2, 2, 0, 2, 0, 2 ],
   [ 2, 0, 2, 0, 0, 2, 2 ],
   [ 2, 2, 0, 2, 0, 2, 2 ],
   [ 2, 0, 0, 0, 0, 0, 2 ],
   [ 2, 2, 2, 2, 2, 2, 2 ]
   ];

   var startI = 1;
   var startJ = 1;
   var endI = 5;
   var endJ = 5;
   var success = 0;

   // O mouse se move em quatro direções: para cima, para a esquerda,
para baixo e para a direita. se bater na parede, volte e selecione a próxima
direção para frente
   function visit(i, j) {
   maze[i][j] = 1;
   if (i == endI && j == endJ) {
   success = 1;
   }
   if (success != 1 && maze[i][j + 1] == 0)
   visit(i, j + 1);
   if (success != 1 && maze[i + 1][j] == 0)
   visit(i + 1, j);
   if (success != 1 && maze[i][j - 1] == 0)
   visit(i, j - 1);
   if (success != 1 && maze[i - 1][j] == 0)
   visit(i - 1, j);
```

```
if (success != 1)
maze[i][j] = 0;
return success;
}




//////////////////////testing//////////////////
document.write("Maze   <br>");
for (var i = 0; i < 7; i++) {
for (var j = 0; j < 7; j++) {
if (maze[i][j] == 2) {
document.write("   ");
} else {
document.write("*  ");
}
}
document.write("<br>");
}

if (visit(startI, startJ) == 0) {
document.write("No exit found <br>");
} else {
document.write("Maze Path : <br>");
for (var i = 0; i < 7; i++) {
for (var j = 0; j < 7; j++) {
if (maze[i][j] == 2) {
document.write("   ");
} else if (maze[i][j] == 1) {
document.write("-  ");
} else {
document.write("*  ");
}
}
```

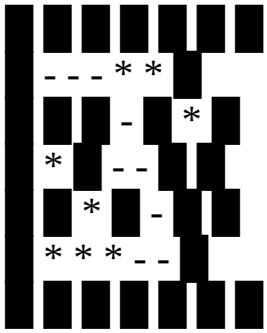
```
    document.write("<br>");
    }
    }
</script>
```

Resultado:

Maze



Maze Path :



Eight Coins

Existem oito moedas com a mesma aparência, uma é uma moeda falsificada e o peso da moeda falsificada é diferente da moeda real, mas não se sabe se a moeda falsificada é mais leve ou mais pesada do que a moeda real. Crie um algoritmo eficiente para detectar esta moeda falsificada.

Solução: .

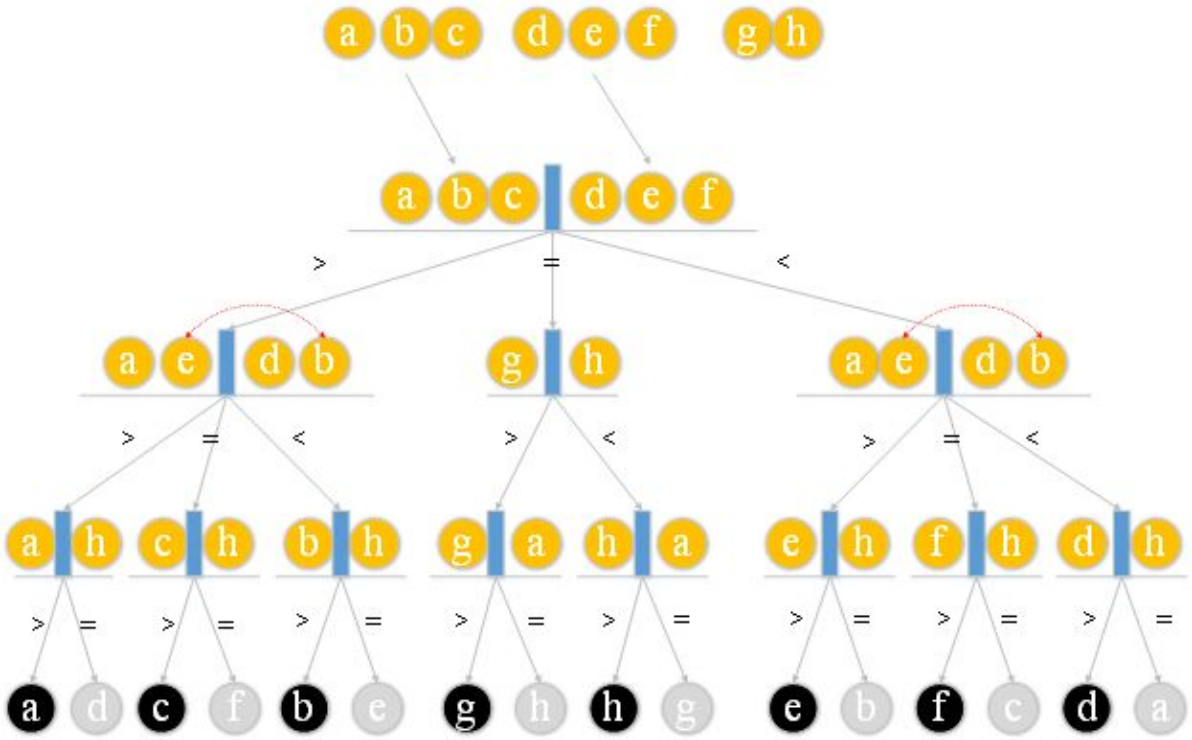
Pegue seis a, b, c, d, e, f de oito moedas e coloque três na balança para comparação. Suponha que a, b, c sejam colocados em um lado e d, e, f sejam colocados no outro lado.

1.a + b + c> d + e + f 2.a + b + c = d + e + f 3.a + b + c <d + e + f

Se a + b + c> d + e + f, há uma moeda falsa em uma das seis moedas e g, h são moedas reais. Neste momento, uma moeda pode ser removida de ambos os lados. Suponha que c e f sejam removidos. Ao mesmo tempo, uma moeda de cada lado é substituída. Suponha que as moedas b e e sejam trocadas, e então a segunda comparação. Existem também três possibilidades:

1. a + e> d + b: a moeda falsa deve ser uma de a, d. contanto que comparemos uma moeda real h com a, podemos encontrar a moeda falsa. Se a> h, a é uma moeda falsificada mais pesada; se a = h, d é uma moeda falsa mais leve.
2. a + e = d + b: a moeda falsa deve ser uma de c, f, e a moeda real h é comparada com c. Se c> h, c é uma moeda falsificada mais pesada; se c = h, então f é uma moeda falsa mais leve.
3. a + e <d + b: um de b ou e é uma moeda falsificada. Use também a moeda real h para comparar com b, se b> h, então b é uma moeda falsificada mais pesada; se b = h, então e é uma moeda falsa mais leve;

Solução Gráfica  **Heavy**  **Light**


a b c d e f g h
a b c d e f
> = <
a e d b
g h
a e d b
> = <
> <
> = <
a h
c h
b h
g a
h a
e h
f h
d h
> = > = > = > = > = > = > = > =
a d c f b e g h h g e b f c d a

1. Crie um EightCoins.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript">
   function compare(coins, i, j, k) {//coin[k] true,coin[i]>coin[j]
   if (coins[i] > coins[k]) //coin[i]>coin[j]&&coin[i]>coin[k] ----->coin[i]
is a heavy counterfeit coin
   document.write("\nCounterfeit currency " + (i + 1) + " is heavier
<br>");
   else //coin[j] is a light counterfeit coin
   document.write("\nCounterfeit currency " + (j + 1) + " is lighter
<br>");
   }

   function eightcoins(coins) {
   if (coins[0] + coins[1] + coins[2] == coins[3] + coins[4] + coins[5]){
//(a+b+c)==(d+e+f)
   if (coins[6] > coins[7]) //g>h?(g>a?g:a):(h>a?h:a)
   compare(coins, 6, 7, 0);
   else //h>g?(h>a?h:a):(g>a?g:a)
   compare(coins, 7, 6, 0);
   } else if (coins[0] + coins[1] + coins[2] > coins[3] + coins[4] +
coins[5]){ //(a+b+c)>(d+e+f)
   if (coins[0] + coins[3] == coins[1] + coins[4]) //(a+e)==(d+b)
   compare(coins, 2, 5, 0);
   else if (coins[0] + coins[3] > coins[1] + coins[4]) //(a+e)>(d+b)
   compare(coins, 0, 4, 1);
   if (coins[0] + coins[3] < coins[1] + coins[4]) //(a+e)<(d+b)
   compare(coins, 1, 3, 0);
   } else if (coins[0] + coins[1] + coins[2] < coins[3] + coins[4] +
coins[5]) {//(a+b+c)<(d+e+f)
   if (coins[0] + coins[3] == coins[1] + coins[4]) //(a+e)>(d+b)
   compare(coins, 5, 2, 0);
   else if (coins[0] + coins[3] > coins[1] + coins[4]) //(a+e)>(d+b)
   compare(coins, 3, 1, 0);
   if (coins[0] + coins[3] < coins[1] + coins[4]) //(a+e)<(d+b)
```

```
    compare(coins, 4, 0, 1);
    }
    }









    //////////////////////testing//////////////////
    var coins = new Array();
    // O peso inicial da moeda é 10
    for (var i = 0; i < 8; i++)
    coins[i] = 10;

    document.write("Enter weight of counterfeit currency (larger or smaller than 10) : 2<br>");
    coins[1] = 2;
    eightcoins(coins);
    for (var i = 0; i < 8; i++)
    document.write(coins[i]+" , ");

    document.write("<br>");

    document.write("Enter weight of counterfeit currency (larger or smaller than 10) : 13<br>");
    coins[3] = 13;
    eightcoins(coins);
    for (var i = 0; i < 8; i++)
    document.write(coins[i]+" , ");
</script>
```

Resultado:

Primeira corrida de novo:
Enter weight of counterfeit currency (larger or smaller than 10) :

2

Counterfeit currency 2 is lighter
10 , 2 , 10 , 10 , 10 , 10 , 10 , 10 ,

Primeira corrida de novo:
Enter weight of counterfeit currency (larger or smaller than 10) :

13

Counterfeit currency 4 is heavier
10 , 10 , 10 , 13 , 10 , 10 , 10 , 10 ,

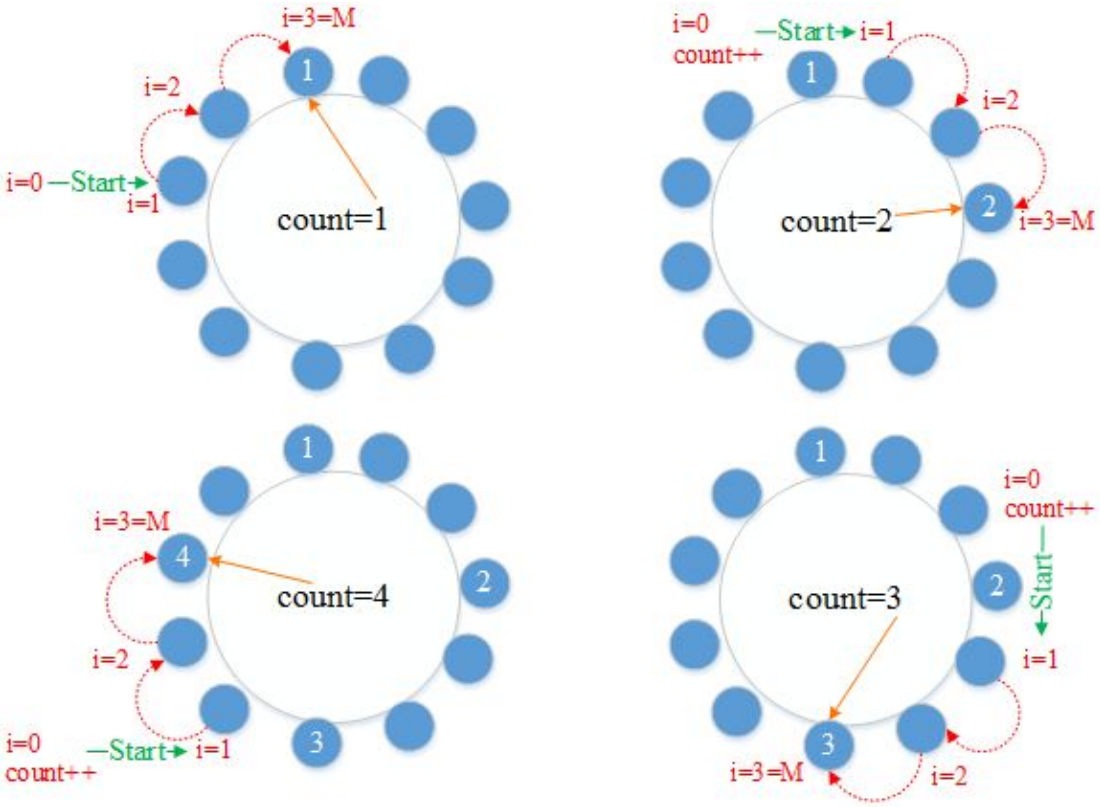
Josephus Problem

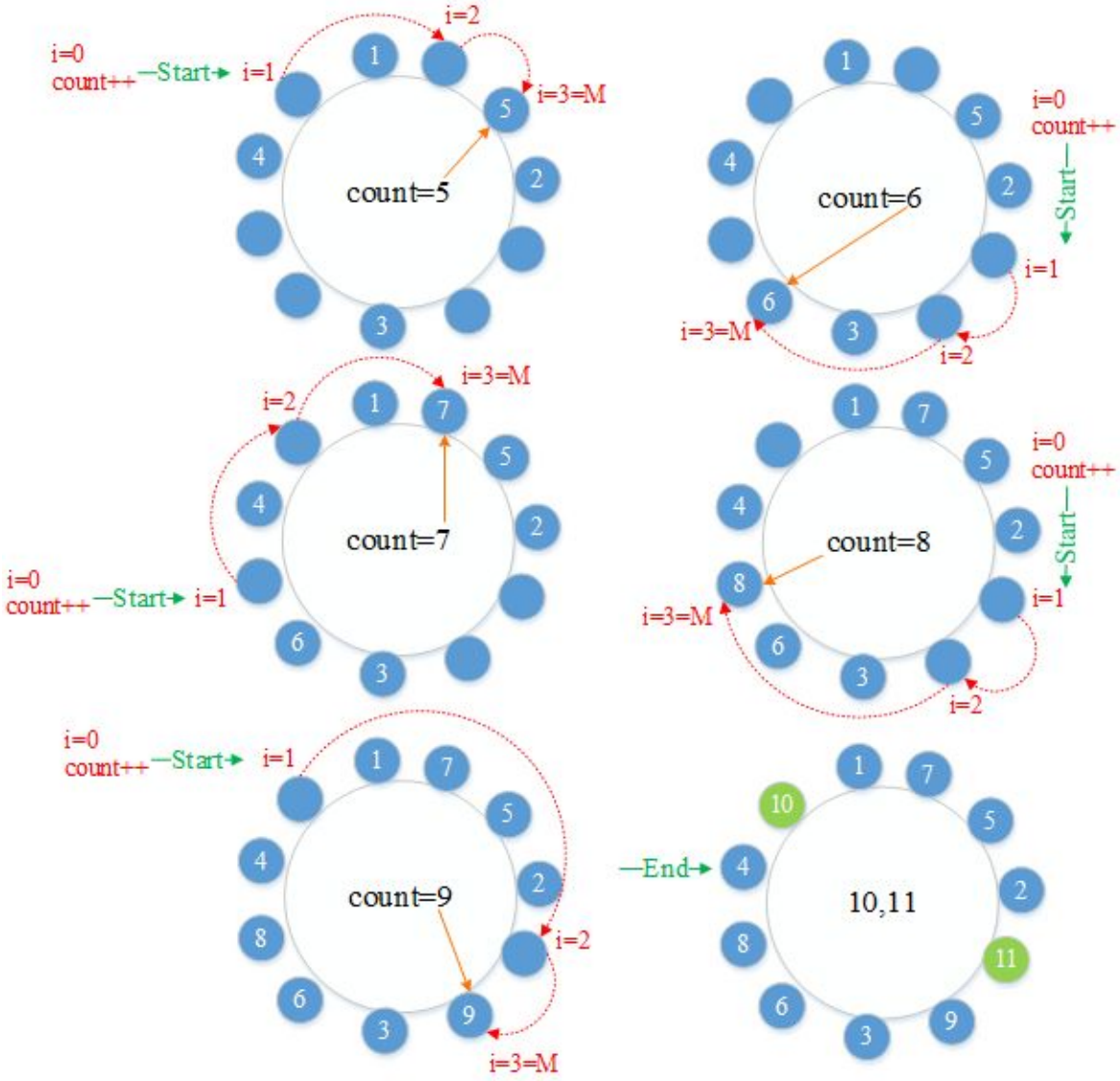
Há 9 judeus escondidos em um buraco com Josefo e seus amigos. Os 9 judeus decidiram morrer ao invés de serem pegos pelo inimigo, então eles decidiram. Em um método suicida, 11 pessoas estão dispostas em um círculo, e a primeira pessoa relata o número. Depois que cada número é relatado à terceira pessoa, a pessoa deve cometer suicídio. Depois, conte novamente a partir do próximo até que todos se suicidem. Mas Josefo e seus amigos não queriam obedecer. Josefo pediu a seus amigos que fingissem obedecer e ele arranjou os amigos consigo mesmo. Na 2ª e 7ª posições escaparam desse jogo da morte.

Solução: Contanto que a matriz seja tratada como um anel. Preencha uma contagem para cada área sem dados, até que a contagem chegue a 11 e, a seguir, liste a matriz do índice 1, você pode saber que cada ordem de suicídio nesta posição é a posição de Joseph. A posição de 11 pessoas é a seguinte: 4 10 1 7 5 2 11 9 3 6 8

Pelo exposto, os dois últimos suicidas ficaram na 2ª e 7ª posições. O anterior Todo mundo morreu, então eles não sabiam que Joseph e seus amigos não seguiam as regras do jogo.

N = 11; // 11 pessoas M = 3; // Depois que cada número é relatado à terceira pessoa, a pessoa deve cometer suicídio.


i=3=M
1
i=2
i=0 —Start→
i=1
count=1
i=0
count++
—Start→ i=1
1
i=2
count=2
2
i=3=M
1
i=3=M
4
count=4
2
i=2
i=0
count++
—Start→ i=1
3
1
i=0
count++
count=3
2
—Start—
i=1
3
i=3=M
i=2


i=0
count++
Start
i=1
i=2
i=3=M
count=5
count=6
i=0
count++
Start
i=1
i=3=M
i=2
count=7
i=3=M
i=2
i=0
count++
Start
i=1
count=8
i=0
count++
Start
i=1
i=3=M
i=2
i=0
count++
Start
i=1
count=9
i=2
i=3=M
End
10,11

1. Crie um Josephus.html com o Bloco de notas e abra-o em seu navegador.

```
<script type="text/javascript"> var N = 11; var M = 3;
   function joseph(man) {
   var count = 1; var i = 0, pos = -1; var alive = 0; while (count <= N) {
   do {
   pos = (pos + 1) % N; // Ring if (man[pos] == 0) i++; if (i == M) {
   i = 0; break; }
   } while (true); man[pos] = count; count++; }

   }

   //////////////////////testing//////////////////
   var man = new Array(); for (var i = 0; i < N; i++) man[i] = 0;
   joseph(man);
   document.write("\nJoseph sequence   <br>"); for (var i = 0; i < N; i++)
document.write(man[i] + " , "); </script>
```

Resultado: Joseph sequence
4 , 10 , 1 , 7 , 5 , 2 , 11 , 9 , 3 , 6 , 8 ,