I

Algoritmos de ordenação

5 de Dezembro de 2023

Método de intercalação - Mergesort

- Proposto por Von Neumann em 1945
- Conceito “dividir e conquistar”
- Complexidade de tempo consistente - todos os casos tem tempo O(n log n)
- Preserva a ordem relativa de elementos iguais
- Complexidade de espaço: precisa de espaço de memória adicional proporcional ao array de input

Mergesort - descrição

- Gera uma sequência ordenada a partir de outras duas já ordenadas usando intercalação
- **Dividir**: input é subdividido em partes menores, recursivamente, até ter n vetores de tamanho um
- **Conquistar**: Ordenar cada uma das metades recursivamente aplicando o merge.
- **Combinar**: Juntar recursivamente as metades até chegar no vetor ordenado de tamanho igual ao original

Mergesort - Passo a passo fase de divisão

- Demarcar os índices que definem o começo e o fim do array
- Encontrar o elemento do meio do array calculando (começo+fim)/2
- Dividir o array no meio usando o índice que indica o meio do array (a metade esquerda vai do começo até o índice do meio, a metade direita vai do índice do meio+1 até o fim)
- Aplicar chamada recursiva do mergesort nas metades até que cada subarray contenha apenas um elemento

Mergesort - Passo a passo fase de merge

- Comparar primeiro elemento de cada metade e mover o elemento menor para o array temporário.
- Aplicar o passo 1 para cada elemento na sublista até que o merge seja finalizado.
- Caso as listas não tenham tamanho igual, mover os elementos que sobram para o array temporário
- Substituir os elementos no array original com os elementos ordenados do array temporário

Mergesort - Exemplo

Consideremos o vetor de tamanho 6
[38, 27, 43, 3, 9, 82, 10]

A metade do array será o índice 3 pois 6/2=3. Logo teremos o vetor esquerdo:
[38, 27, 43, 3] e o vetor direito: [9, 82, 10]

Repetimos processo de partição até chegarmos em 6 vetores de tamanho 1, da seguinte forma:
[38] [27] [43] [3] [9] [82] [10]

Mergesort - Exemplo

Na etapa de merge, comparamos o primeiro índice do vetor esquerdo com o vetor direito e vamos alocando a ordenação no vetor temporário, de forma que teremos o seguinte passo a passo:

Combinamos[38]e[27] formando [27, 38].
Combinamos[43]e[3]formando[3, 43].
Combinamos[27, 38] e[3, 43] formando[3, 27, 38, 43].
Combinamos[9]e[82]formando[9, 82].
Combinamos[10]e[9, 82] formando[9, 10, 82]

Mergesort - Exemplo

Assim, teremos as duas metades do vetor ordenadas, então poderemos aplicar um merge final entre as metades [3, 27, 38, 43] e [9, 10, 82] resultando no vetor ordenado

[3, 9, 10, 27, 38, 43, 82]

Mergesort - Código função merge

```java
private static void merge(int[] v, int esq, int meio, int dir) {
    int n1 = meio - esq + 1;//tamanho dos arrays temporarios vai ser alternado a cada recursão
    int n2 = dir - meio;

    //Cria os arrays temporarios
    int[] vEsq = new int[n1];
    int[] vDir = new int[n2];

    //Copiando elementos pros vetores temporarios
    for (int i = 0; i < n1; ++i) {
        vEsq[i] = v[esq + i];
    }
    for (int j = 0; j < n2; ++j) {
        vDir[j] = v[meio + 1 + j];
    }
```

Mergesort - Código função merge

```
    //Aplicando merge nos arrays temporarios
    int i = 0, j = 0;
    int k = esq;
    while (i < n1 && j < n2) {
        if (vEsq[i] <= vDir[j]) {
            v[k] = vEsq[i];
            i++;
        } else {
            v[k] = vDir[j];
            j++;
        }
        k++;
    }

    //Copiar elementos que sobrarem
    while (i < n1) {
        v[k] = vEsq[i];
        i++;
        k++;
    }
    while (j < n2) {
        v[k] = vDir[j];
        j++;
        k++;
    }
}
```

Mergesort - Código função mergeSort

```java
//MergeSort chamada recursiva
private static void mergeSort(int[] v, int esq, int dir) {
    if (esq < dir) {
        int meio = (esq + dir) / 2;

        //Organizar metades esquerda e direita
        mergeSort(v, esq, meio);
        mergeSort(v,  esq: meio + 1, dir);

        //Juntar metades ordenadas usando merge
        merge(v, esq, meio, dir);
    }
}
```

Mergesort - Resultado final

```
Array original:
38 27 43 3 9 82 10
Array ordenado:
3 9 10 27 38 43 82

Process finished with exit code 0
```

HeapSort - Ordenação por comparação

- Inventado em 1964 por J.W.J. Willians
- Usa estrutura de dados chamada “heap” para enxergar o vetor como se este fosse uma árvore binária
- Complexidade O(n log n)
- Não requer uma estrutura de dados auxiliar de tamanho proporcional à entrada

HeapSort - O que é um “heap”

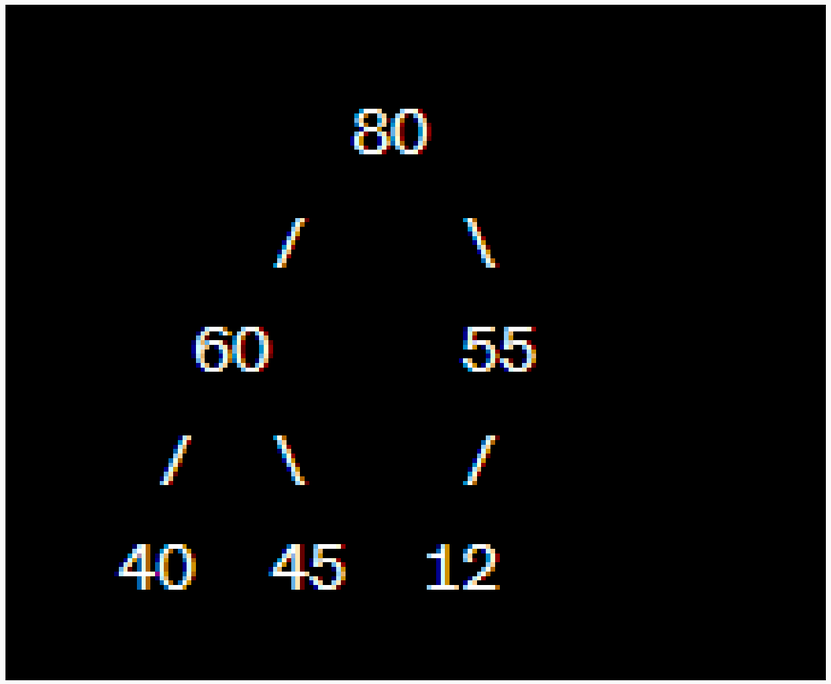
- Heap pode ser pensado como uma fila prioritária (não segue lógica FIFO)
- É uma árvore binária mas não de busca
- Árvore completa ou quase completa da esquerda para direita
- Max heap: para cada nó i com exceção da raiz, o valor de i é menor ou igual ao valor de seu nó pai (nó raiz armazena o maior elemento da árvore)
- Min heap: para cada nó i com exceção da raiz, o valor de i deve ser maior ou igual ao valor de seu nó pai (raiz armazena o menor elemento da árvore)

HeapSort - Relação entre heap e array

- O heap é usado para representar um array
- A transformação de array em heap ocorre da seguinte forma:
- Para um nó de índice i, seu filho da esquerda está no índice 2i+1 e o da direita no índice 2i+2
- Para um nó de índice i, seu pai está no índice (i-1)/2
- Índice zero do vetor é o nó raiz

HeapSort - Relação entre heap e array

- O seguinte vetor [80, 60, 55, 40, 45, 12] gera o seguinte heap:

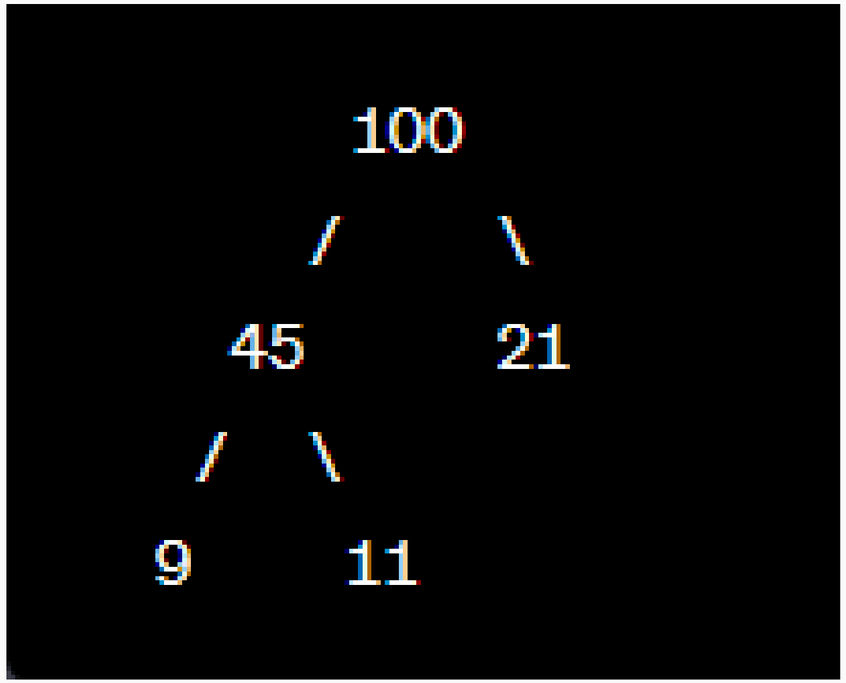


HeapSort - Passo a passo

- “Heapify” – construir o heap a partir do array
- Armazenar o último elemento do vetor em uma variável auxiliar
- Copiar elemento de maior valor (raiz) para a posição final do vetor
- Considerar que a posição v[o] está vazia
- Reorganizar a heap
- Mover o maior filho da raiz para a posição que estava vazia
- Mover a variável auxiliar para sua posição correta no vetor
- Repetir processo até que o vetor esteja ordenado de forma crescente

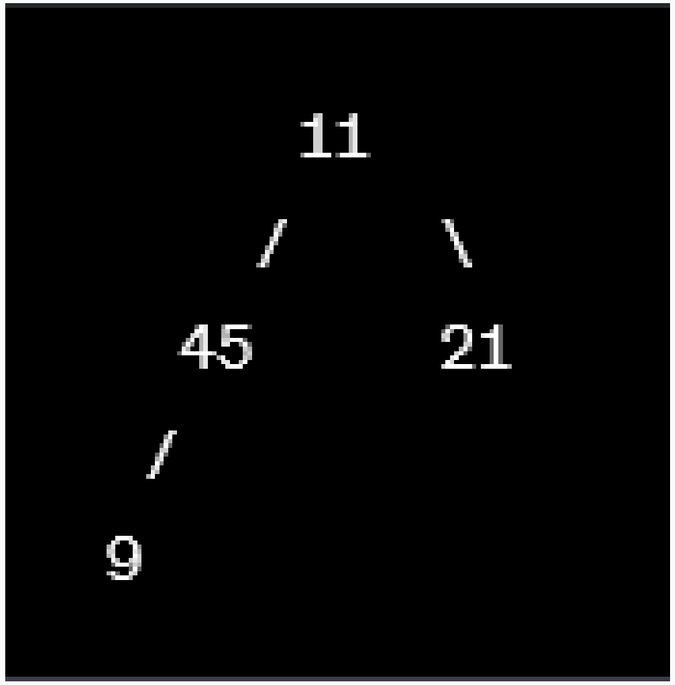
HeapSort - Exemplo

- Considerando o vetor [100, 45, 21, 9, 11] teremos o seguinte heap:



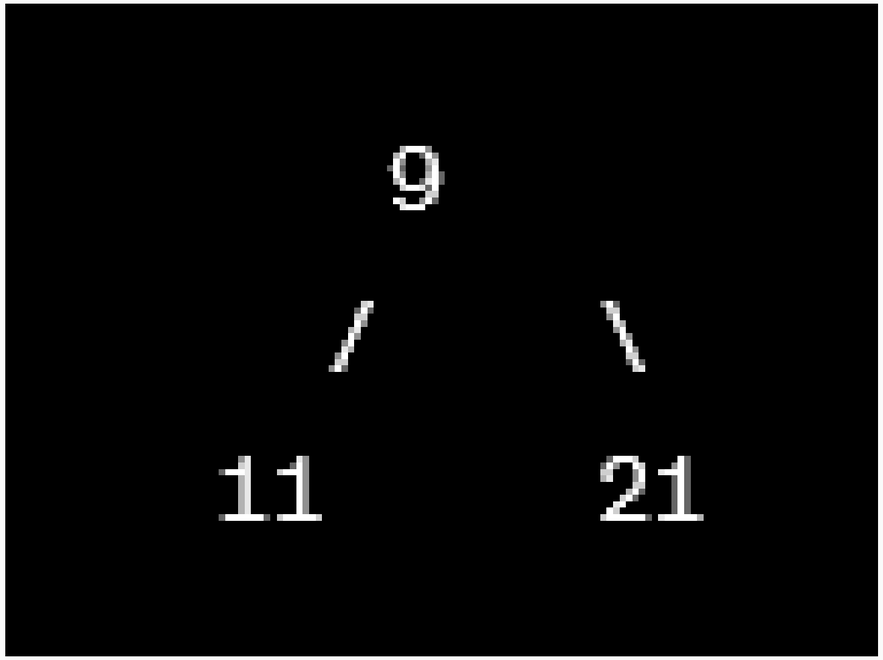
HeapSort - Exemplo

- Armazenamos a raiz 100 na variável auxiliar e reorganizamos o heap



HeapSort - Exemplo

- Movemos 100 para posição correta no vetor (a posição final) e repetimos o processo, removendo 45 e colocando na variável auxiliar, ficando com a seguinte heap:



HeapSort - Exemplo

- 45 então é colocado na posição correta do array, e o processo se repete, até que sobre apenas o número 9 na heap, que será colocado na posição 0 do vetor, resultando em [9, 11, 21, 45, 100].

HeapSort - Algoritmo em Java

```java
public class Heap {
    public void heapSort (int [] v){//função principal
        int tam = v.length;

        //Construçao de uma max heap (raiz é maior elemento)
        for (int i = tam / 2 - 1; i >= 0; i--) {
            heapify(v, tam, i);
        }

        // Extração de elementos da heap
        for (int i = tam - 1; i > 0; i--) {
            // Move a raiz para o fim do vetor usando variavel aux
            int aux = v[0];
            v[0] = v[i];
            v[i] = aux;

            // Reorganização a heap com a função heapify que transforma o array em heap
            heapify(v, i, 0);
        }
    }
```

HeapSort - Algoritmo em Java

```java
public void heapify(int []v, int tam, int i){
    int maior = i; //Maior elemento como raiz
    int esq = 2 * i + 1; // esquerda = 2*i + 1
    int dir = 2 * i + 2; // direita = 2*i + 2

    // Se filho da esquerda for maior que a raiz
    if (esq < tam && v[esq] > v[maior]) {
        maior = esq;
    }

    // Se filho da direita for maior que a raiz
    if (dir < tam && v[dir] > v[maior]) {
        maior = dir;
    }

    // Se o maior não for raiz (diferent de i
    if (maior != i) {
        int aux = v[i];
        v[i] = v[maior];
        v[maior] = aux;

        //Aplicar recursivamente a função heapify na sub arvore afetada
        heapify(v, tam, maior);
    }
}
```

Shellsort - Insertion sort

- Conceitualizado por Donald Shell em 1959
- Usa intervalos para comparar elementos mais distantes no array
- É uma alteração do insertion sort, onde se comparam elementos adjacentes
- Mais rápido que o insertion sort original - permite organização preliminar do vetor
- Coloca elementos muito discrepantes no lugar mais rapidamente

Shellsort - Insertion sort

- Mais eficiente dentre os algoritmos de complexidade quadrática
- Não é necessário alocar memória adicional
- Complexidade depende da sequência de intervalos escolhida
- Sequência original de Shell: N/2 , N/4 , ..., 1

Shellsort - Passo a passo

- Define-se a sequência de intervalos a ser usada
- Divisão da lista original em sublistas com elementos que estão separados pelo intervalo
- Sublistas então são organizadas individualmente
- Iterações repetidas diminuem o intervalo
- Comparação final usando insertion sort (elementos adjacentes)

Shellsort - Exemplo

- Considere o array de tamanho 8 [9, 456, 6, 60, 38, 2, 32, 89]
- 1ª iteração: organizar elementos de distância n/2 (8/2 = 4) gera o seguinte vetor [9, 2, 6, 89, 38, 456, 32, 60]
- 2ª iteração: N/4 (2), resultando no seguinte vetor: [6, 2, 9, 60, 32, 89, 38, 456]
- Iteração final: N/8 (1) Insertion sort até chegar no vetor ordenado [2, 6, 9, 32, 38, 60, 89, 456]

Shellsort - Código em Java

```java
public class Shell {
    void shellSort(int v[]) {
        int tam = v.length;
        //O primeiro for controla o tamanho do intervalo usando o conceito do intervelo de shell original
        for (int intervalo = tam / 2; intervalo > 0; intervalo /= 2) {
            //O segundo for realiza o insertion sort de acordo com o tamanho do intervalo
            for (int i = intervalo; i < tam; i += 1) {
                int aux = v[i];
                int j;
                //Mover elementos maiores que aux para direita
                for (j = i; j >= intervalo && v[j - intervalo] > aux; j -= intervalo) {
                    v[j] = v[j - intervalo];
                }
                v[j] = aux; //Posicionar aux na posição correta dentro do intervalo
            }
        }
    }
}
```

Obrigada por mais um semestre frutífero professora!