Algoritmos e Estrutura de Dados II

Exercício Prático
Heapsort e Quicksort

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia

Agenda

Roteiro da Aula

Exercício 1

Utilizando o algoritmo de ordenação **Heapsort**, ordene um vetor composto por 1.000.000, 2.000.000 e 3.000.000 valores. Para cada um, execute os seguintes testes:

- vetor composto por números aleatórios;
- vetor composto por números em ordem crescente;
- vetor composto por números em ordem decrescente.

Exercício 1

Pseudocódigo

Algoritmo 1: ConstroiHeap

Entrada: Vetor $V[i..n-1]$, raiz no nó i, tamanho do vetor n
Saída: Heap no vetor $V[i..n-1]$

```
1  início
2      maior ← i // Inicializa maior como a raiz
3      l ← 2i + 1 // Filho da esquerda
4      r ← 2i + 2 // Filho da direita
5      // Se o filho da esquerda é maior que a raiz
6      se ( l < n AND V[l] > V[maior] ) então
7          maior ← l
8      // Se filho da direita é maior que a raiz
9      se (r < n AND V[r] > V[maior]) então
10         maior ← r
11     // Se maior não é a raiz
12     se (maior ≠ i) então
13         Troca V[i] ↔ V[maior]  // O maior passa a ser a raiz
14         ConstroiHeap(V, maior, n) // Cria o heap na sub-árvore
```

Exercício 1

Solução

```
1  void constroi_heap(int *vetor, int i, int n) {
2      int maior, l, r, aux;
3      maior = i;
4      l = 2*i + 1;
5      r = 2*i + 2;
6      if (l<n && vetor[l]>vetor[maior])
7          maior = l;
8      if ((r<n) && vetor[r]>vetor[maior])
9          maior = r;
10     if (maior!=i) {
11         aux = vetor[i];
12         vetor[i] = vetor[maior];
13         vetor[maior] = aux;
14         constroi_heap(vetor, maior, n);
15     }
16 }
```

Exercício 1

Pseudocódigo

Algoritmo 2: Heapsort

Entrada: Vetor $V[0..n-1]$, tamanho do vetor n
Saída: Vetor V ordenado

```
1  início
2      // Contrói o heap rearranjando o vetor
3      para ( i ← n/2 − 1 decrescendo até i = 0 ) faça
4          ConstroiHeap(V, i, n)
5      // Extrai cada elemento, um por um, do heap
6      para ( i ← n − 1 decrescendo até i = 0) faça
7          // Move a raiz atual para o fim do vetor.
8          Troca V[0] ↔ V[i]
9          // Chama a função para recriar o heap
10         // no vetor reduzido
11         ConstroiHeap(V, 0, i)
```

Exercício 1

Solução

```
1  void heapsort(int n, int *vetor) {
2      int i, aux;
3      for(i = (n/2)-1; i >= 0; i--) {
4          constroi_heap(vetor, i, n);
5      }
6      for(i = n-1; i >= 0; i--) {
7          aux = vetor[0];
8          vetor[0] = vetor[i];
9          vetor[i] = aux;
10         constroi_heap(vetor, 0, i);
11     }
12 }
```

Roteiro da Aula

Exercício 2

Tempo Processamento

- Calcule o o tempo de processamento para ordenar cada um dos vetores do exercício anterior utilizando o algoritmo **Heapsort**.
- Para isso, utilize a função `clock()`. Acesse esse link para entender o funcionamento da função `clock()`.

Exercício 2

Tempo Processamento

- O código a seguir apresenta um exemplo de ordenação utilizando um vetor com valores aleatórios.
- Modifique esse código a fim de obter o tempo de processamento.

Exercício 2

Solução

```c
#define tamanho_vetor 100000
#define valor_max 1000000
#define valor_min 0
int main() {
    int i, *vetor;
    srand(0);
    vetor = malloc(tamanho_vetor*sizeof(int));
    for (i = 0; i < tamanho_vetor; i++)
        vetor[i] = (rand() % valor_max) + valor_min;
    for(i = 0; i < tamanho_vetor; i++)
        printf("%d ", vetor[i]);
    heapsort(tamanho_vetor, vetor);
    printf("\n");
    for(i = 0; i < tamanho_vetor; i++)
        printf("%d ", vetor[i]);
    free (vetor);
    return 0;
}
```

Roteiro da Aula

Exercício 3

Utilizando o algoritmo de ordenação **Quicksort**, ordene um vetor composto por 1.000.000, 2.000.000 e 3.000.000 valores. Para cada um, execute os seguintes testes:

- vetor composto por números aleatórios;
- vetor composto por números em ordem crescente;
- vetor composto por números em ordem decrescente.

Exercício 3

Pseudo-código

Algoritmo 3: QuicksortOrdena

Entrada: Vetor $V[0..n-1]$, tamanho n
Saída: Vetor V ordenado

```
1 início
2     Quicksort(V, 0, n − 1)
```

Algoritmo 4: Quicksort

Entrada: Vetor $V[Esq..Dir]$, Esq, Dir
Saída: Vetor V ordenado

```
1 início
2     (i, j) ← Particiona(V, Esq, Dir)
3     se ( Esq < j ) então
4         Quicksort(V, Esq, j )
5     se ( i < Dir ) então
6         Quicksort(V, i, Dir)
```

Exercício 3

Pseudo-código

Algoritmo 5: Particiona

Entrada: Vetor $V[Esq..Dir]$, Esq, Dir
Saída: Vetor V ordenado

```
1  início
2      i ← Esq; j ← Dir
3      x ← V[(i+j)/2]
4      repita
5          enquanto ( x > V[i] ) faça
6              i ← i + 1
7          enquanto ( x < V[j] ) faça
8              j ← j − 1
9          se ( i ≤ j ) então
10             Trocar V[i] ↔ V[j]
11             i ← i + 1
12             j ← j − 1
13     até (i > j);
14     retorna (i, j)
```

Exercício 3

Solução

```
1  void quicksort(int *a, int left, int right) {
2      int i, j, x, y;
3      i = left; j = right;
4      x = a[(left + right) / 2];
5      while(i <= j) {
6          while(a[i] < x && i < right)
7              i++;
8          while(a[j] > x && j > left)
9              j--;
10         if(i <= j) {
11             y = a[i];
12             a[i] = a[j];
13             a[j] = y;
14             i++; j--;
15         }
16     }
17     if(j > left)
18         quicksort(a, left, j);
19
20     if(i < right)
21         quicksort(a, i, right);
22 }
```

Roteiro da Aula

Exercício 4

Tempo Processamento

Calcule o tempo de processamento para ordenar cada um dos vetores do exercício anterior utilizando o algoritmo **Quicksort**.

Algoritmos e Estrutura de Dados II

Exercício Prático
Heapsort e Quicksort

prof. Frederico Santos de Oliveira

Universidade Federal de Mato Grosso
Instituto de Engenharia