Universidade Federal de Juiz de Fora

Instituto de Ciências Exatas

Departamento de Ciência da Computação

DCC001
ANÁLISE E PROJETO DE ALGORITMOS

Trabalho Prático

Rafael Terra
Mateus Coutinho Marim
Aleksander Yacovenco
Matheus Soares

Professor - Stênio Soares

Juiz de Fora - MG
24 de abril de 2017

Sumário

Lista de Figuras

Lista de Programas

Lista de Tabelas

1 Introdução

Escrever aqui a introdução do trabalho...

1.1 Considerações iniciais

- Ambiente de desenvolvimento do código fonte: Code Blocks (por exemplo).
- Linguagem utilizada: Linguagem C.
- Ambiente de desenvolvimento da documentação: TeXnicCenter 1 BETA 7.50-Editor de LaTeX.

1.2 Especificação do problema

Você deverá implementar um tipo abstrato de dados TVetor para representar vetores no espaço R^n. Esse tipo abstrato deverá armazenar a dimensão do vetor e suas respectivas componentes. Considere que a dimensão dos vetores será determinada em tempo de execução.

2 Algoritmo e estruturas de dados

Em [1], são apresentadas estruturas de dados...
O código resultante desse processo será apresentado no Programa 1.

```
//Timer
//Inicializa a contagem
void tStartTimer(stopWatch *timer)
{
  QueryPerformanceCounter(&timer->start);
}
//Para a contagem
void tStopTimer(stopWatch *timer)
{
  QueryPerformanceCounter(&timer->stop);
}
//Converte o tempo computado pelo stopWatch para segundos
double tLIToSecs(LARGE_INTEGER *L)
{
  LARGE_INTEGER frequency;
  QueryPerformanceFrequency(&frequency);
  return ((double)L->QuadPart /(double)frequency.QuadPart);
}
//Retorna o numero de segundos passados na contagem
double tGetElapsedTime(stopWatch *timer)
{
  LARGE_INTEGER time;
  time.QuadPart = (timer->stop).QuadPart - (timer->start).QuadPart;
  return tLIToSecs(&time) ;
}
```

Programa 1: Timer

3 Análise de complexidade dos algoritmos

3.1 Análise do Bubble Sort

O bubble sort faz múltiplas passadas em uma lista, em cada passada ele verifica se um par de elementos adjacentes estão em ordem, caso não estejam, a posição deles é trocada de forma que o maior deles fique após o menor, isso é repetido até que não sejam mais necessárias trocas.

O melhor caso do bubble sort é quando a sua entrada é uma lista ordenada, neste caso os elementos já estão em ordem nenhuma troca é efetuada e o algoritmo termina na primeira passada, logo sua complexidade é na ordem de $O(n)$.

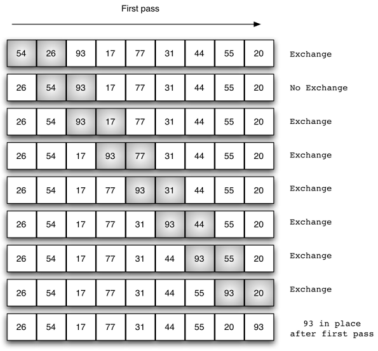

Figura 1: Uma passada do bubble sort.

No pior caso podemos levar em consideração quando os elementos estão ordenados em ordem decrescente, neste caso na iteração 0 o block roda $n-1-0$ vezes, na iteração 1 ele roda $n-1-(n-1)=0$ vezes. Então no total, o bloco roda a quantidade de vezes expressa na Equação 2.

$$O(n) = \sum_{i=0}^{n-1} n - i - 1 = n^2 - \sum_{i=0}^{n-1} i - n = n^2 - n * (n-1)/2 = n^2/2 - n/2 \qquad (1)$$

Dando um pior caso na ordem de $O(n^2)$.

3.2 Análise do Insertion Sort

Figura 2: Insertion sort funciona como um jogo de cartas.

O Insertion sort funciona usando a mesma ideia de quando organizamos cartas nas nossas mãos, para cada carta em nossas mãos verificamos se ela obedece a propriedade de lista ordenada de que a próxima carta tem um valor maior do que a carta atual, caso isso não aconteça, voltamos carta por carta até encontrar a posição que a carta pertence. O melhor caso ocorre quando a lista já está ordenada e todos

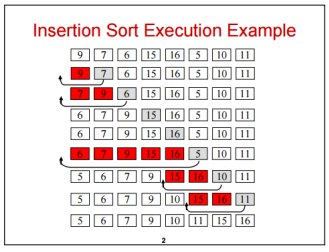

Figura 3: Exemplo de execução.

elementos estão em ordem, neste caso o algoritmo tem uma complexidade na ordem de $O(n)$.

O pior caso ocorre quando a lista está ordenada de forma decrescente, vamos tentar achar uma fórmula para a quantidade de movimentos necessários.
Iteração 0: 0
Iteração 1: 1
Iteração 2: $1 + 1 = 2$
Iteração 3: $2 + 1 = 3$
.
.
.
Iteração n-1: $\sum_0^{n-1} i = n * (n-1)/2 = n^2/2 - n/2$
Logo, a ordem de complexidade no pior caso é $O(n^2)$.

3.3 Análise do Selection Sort

O selection sort ordena a lista por repetidamente pegar o menor elemento da sublista restante quando se está na posição i e trocar a posição do menor elemento com o da posição da iteração i. O custo para pegar o elemento mínimo em cada iteração i é $n-i$ mesmo que o vetor esteja ordenado, pois não há uma verificação se os elementos já estão em ordem, então vamos ter um custo total dado pela equação 2.

$$O(n) = \sum_{i=0}^{n-1} n - i - 1 = n^2 - \sum_{i=0}^{n-1} i - n = n^2 - n * (n-1)/2 = n^2/2 - n/2 \qquad (2)$$

Logo, a complexidade do algoritmo tanto no melhor, pior e caso médio é de $O(n^2)$.

3.4 Análise do Merge Sort

O merge sort funciona utilizando o paradigma da divisão em conquista, dividindo a lista em dois recursivamente até que ela esteja ordenada, que é o caso em que o tamanho da lista é 1 e depois faz o merge das listas mantendo elas ordenadas.

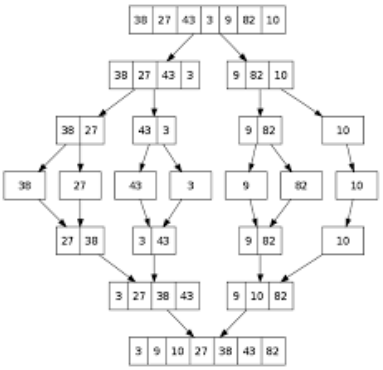

Figura 4: Árvore de execução do merge sort.

A equação de recorrência do merge sort é dada por 3.

$$T(n) = 2T(n/2) + n = c * n * log(n) \qquad (3)$$

Logo, por o comportamento do merge sort não variar de acordo com a natureza da lista, a sua ordem de complexidade em todos os casos é $O(n * log(n))$.

3.5 Análise do Heap Sort

O algoritmo Heap Sort insere todos os elementos (de um vetor não ordenado) em um heap então troca seu primeiro elemento (máximo) com o último (mínimo) e reduz o tamanho da heap por 1 por seu último elemento já está na posição final no vetor ordenado. Depois usamos o procedimento Heapify pois nesse processo podemos ter quebrado a propriedade de heap máxima. Continuamos esse processo até que o tamanho final da heap seja 1.

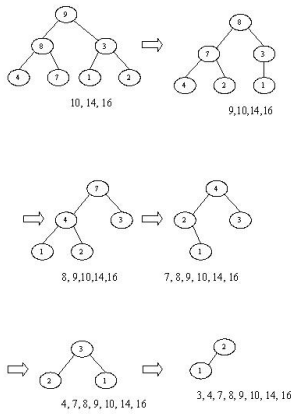

Figura 5: Exemplo do heap sort.

Primeiro nós construímos o max-heap que é um processo de ordem n e depois executamos o algoritmo Heapify que tem custo $log(n)$ por n vezes, logo a equação da complexidade do Heap sort é dada por 4.

$$T(n) = n + n * log(n) \tag{4}$$

Então o pior, melhor e o caso médio do Heap sort é da ordem de $O(n * log(n))$.

4 Testes

Estas estruturas são apresentadas na Figura 5.

5 Conclusão

Neste trabalho foram revistos conceitos sobre...[2].
Muito dos algoritmos são extraídos de:[3].

Referências

[1] Rasmus Pagh. Hash and displace: Efficient evaluation of minimal perfect hash functions. In *Workshop on Algorithms and Data Structures*, pages 49–54, 1999.

[2] Gabriel Torres. Clube do hardware, 2009. `http://clubedohardware.com.br`, visitado em 08/08/2009.

[3] N. Ziviani. *Projeto de Algoritmos: com implementações em Pascal e C*. Cengage Learning (Thomson / Pioneira), São Paulo, 1st edition, 2004.