Algoritmos e Estrutura de Dados I

Fabrício Olivetti de França e Paulo Henrique Pisani

13 de março de 2019

Lista de Exercícios 4

Questões Gerais

1. Faça uma adaptação de um algoritmo de ordenação qualquer para que ele remova todas as repetições em uma lista de dados.

Ordenação Simples

1. Mostre que o número de repetições do processo interno do algoritmo *BubbleSort* para que a lista fique ordenada é igual à maior distância que um elemento precisará percorrer de um índice maior para um menor.
2. Como você alteraria o algoritmo *InsertionSort* para se aproveitar de um algoritmo de Busca Binária. Quais ganhos teríamos?
3. Qual o número de comparações feitas pelo *BubbleSort*, *InsertionSort* e *SelectSort* nos seguintes casos:

a. Lista ordenada
b. Lista em ordem inversa
c. Os elementos de índice par são os menores elementos na ordem correta e os de índice ímpar são os maiores elementos na ordem inversa
d. Os elementos até a metade da lista são os menores elementos e estão na ordem correta e os demais elementos são os maiores elementos na ordem inversa
e. Os elementos de índice par são os menores elementos na ordem correta e os de índice ímpar os maiores elementos na ordem correta

Ordenação Eficiente

1. Crie uma versão não-recursiva do *QuickSort*.

2. Modifique o algoritmo *QuickSort* para que, quando a sublista tiver um tamanho menor que n, utilize o algoritmo *InsertionSort*. Determine empiricamente o valor de n.

3. Modifique o algoritmo *QuickSort* para que o *pivot* seja o elemento do meio da sublista, ao invés do primeiro. Em quais situações essa escolha é melhor?

4. Altere todos os algoritmos de ordenação por comparação para que guarde uma contagem da quantidade de comparações efetuadas. Gere diversos arquivos com 10, 100, e 1000 e verifique a média e desvio-padrão do número de comparações feitas por cada algoritmo em cada um dos casos.

5. Ilustre passo a passo a aplicação da função `partition` na *array* `l = {13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11}`.

6. Mostre (informalmente) que a complexidade do algoritmo `partition` é $O(n)$.

7. Altere o algoritmo *QuickSort* para ordenar uma *array* de forma decrescente.

8. Mostre a sequência de operações do *QuickSort* para quando todos os valores são iguais. O que aconteceria se a linha `if (base[j].key < pivot.key)` fosse trocada para `if (base[j].key <= pivot.key)`?

Ordenação Eficiente II

1. Implemente o algoritmo de ordenação *CountingSort* pelos seguintes passos:

   a. Declare um vetor `count` e defina `count[i]` com o número de elementos menores que `x[i]`.
   b. Coloque `x[i]` na posição `count[i]` de um vetor de saída.

   Responda se esse algoritmo é **estável, in-place, online e adaptativo**.

2. Qual a quantidade mínima e máxima de elementos em um *heap* de altura h?

3. Mostre que um *heap* com n elementos tem altura $\lg n$.

4. Mostre que em qualquer sub-árvore de um *max-heap*, a raiz da sub-árvore contém o maior valor de toda a sub-árvore.

5. Onde na *max-heap* o menor elemento pode estar localizado, assumindo que todos os elementos são distintos?

6. Um *array* ordenado é um *min-heap*?

7. Mostre que uma representação de *array* de um *max-heap*, o índice das folhas são: $\lfloor n/2 \rfloor + 1, \lfloor n/2 \rfloor + 2, \ldots, n$.

8. Ilustre a operação `max_heapify(l, 3, 14)` na array `l = {27, 17, 3, 16, 13, 10, 1, 5, 7, 12, 4, 8, 9, 0}`.
9. Escreva a função `min_heapify`.
10. O que acontece se chamar `max_heapify(l, i, n)` quando `l[i]` for maior que seus filhos?
11. O que acontece se chamar `max_heapify(l, i, n)` quando `i > n/2`?
12. Escreva um `max_heapify` iterativo.
13. Ilustre passo a passo o procedimento *HeapSort* na array `l = {5, 13, 2, 25, 7, 17, 20, 8, 4}`.
14. Crie um algoritmo que remove um dado elemento de uma *heap* de tamanho N transformando-a em uma *heap* de tamanho $N - 1$.
15. Ilustre passo a passo o algoritmo *BucketSort* na array `l = {.79, .13, .16, .64, .39, .20, .89, .53, .71, .42}`.
16. Mostre um caso em que o algoritmo *BucketSort* tem complexidade $O(n^2)$.
17. Dados pontos $p_i = (x_i, y_i)$ em uma círculo unitário tal que $0 < x_i^2 + y_i^2 <\leq 1$. Suponha que os pontos estão uniformemente distribuídos. Crie um algoritmo para ordenar n pontos de acordo com a sua distância em relação a origem: $d_i = \sqrt{x_i^2 + y_i^2}$.