Curso de Tecnologia em Sistemas de Computação
Disciplina: Estrutura de Dados e Algoritmos
AP3 - Segundo semestre de 2015

Nome -
Assinatura -

Observações:

1. Prova sem consulta e sem uso de máquina de calcular.
2. Use caneta para preencher o seu nome e assinar nas folhas de questões e nas folhas de respostas.
3. Você pode usar lápis para responder as questões.
4. Ao final da prova devolva as folhas de questões e as de respostas.
5. Todas as respostas devem ser transcritas nas folhas de respostas. As respostas nas folhas de questões não serão corrigidas.

---

1. Forneça os seguintes conceitos:

   (a) (1,0) Complexidade de pior caso de um algoritmo

   Resposta: Dado um algoritmo A, seja $E = \{E_1, E_2, E_3, \dots\}$ o conjunto de todas as possíveis entradas para A. Além disso, seja t_i o número de passos que A executa ao receber E_i como entrada, para todo $E_i \in E$. A *complexidade de pior caso* pode ser definida como $\max\limits_{E_i \in E}\{t_i\}$.

   (b) (1,0) Árvore B de ordem d

   Resposta: Uma árvore B de ordem d é uma árvore ordenada que é vazia, ou que satisfaz as seguintes condições:

   - A raiz é uma folha ou possui no mínimo dois filhos;
   - Cada página diferente da raiz e das folhas possui no mínimo $d+1$ filhos;
   - Cada página possui no máximo $2d+1$ filhos;
   - Todas as folhas estão no mesmo nível.

   Além disso, dado k o número de elementos de uma página P, então:

   - Se P não é folha, então P possui $k+1$ filhos e, assim, $d \leq k \leq 2d$;
   - Se P é a raiz então $1 \leq k \leq 2d$;
   - Os elementos $s_1, \dots, s_k$ de P estão ordenados;
   - Dados $p_0, \dots, p_k$ os ponteiros para os filhos de P não folha, a árvore é ordenada de modo que:
     - Todos os elementos da página apontada por p_0 são menores que s_1;
     - Todos os elementos da página apontada por p_i são menores que s_{i+1} e maiores que s_i, $i \in \{1, \dots, k-1\}$;
     - Todos os elementos da página apontada por p_k são maiores que s_k;

2. (2,0) Desenvolva um algoritmo que recebe como entrada um vetor V com n elementos distintos (suponha n ímpar) e determina a *mediana* de V.

   A mediana de V é um elemento m tal que existem $(n-1)/2$ elementos menores do que m e $(n-1)/2$ elementos maiores do que m em V.

Resposta: Este é um caso particular do problema de determinação do k-ésimo menor (analogamente maior) elemento de um vetor, onde $k = (n-1)/2$ neste caso. Uma forma simples de encontrar tal elemento é ordenar V e retornar $V[k]$. Este algoritmo possui complexidade $O(n \log n)$ devido à ordenação do vetor, apesar da seleção ser feita em $O(1)$. Este algoritmo é útil se se é desejado efetuar um número grande de seleções no vetor.

Um outro procedimento possível é percorrer o vetor exatamente k vezes buscando o menor elemento em cada iteração i, onde os $i-1$ menores elementos de V estão nas $i-1$ primeiras posições. Isso pode ser feito movendo o menor elemento encontrado para a posição i em cada percurso i e buscando-se o $(i+1)$-ésimo menor elemento dentre as posições de $i+1$ a n. A complexidade deste algoritmo é dado por $O(kn)$ e, assim, $O(n^2)$ quando $k = O(n)$.

Outro algoritmo intuitivo utiliza um heap H de prioridade máxima a partir dos elementos de V e executa $k-1$ remoções. O elemento desejado será o primeiro elemento do heap resultante. Assim, a complexidade deste algoritmo é $O(n + k \log n)$. O termo n segue da criação de H e o termo $k \log n$ segue das $k-1$ remoções em H. Note que se $k = O(n)$, como no caso da mediana, então a complexidade é $O(n \log n)$.

Utilizando heap de prioridade máxima também podemos obter outro algoritmo. Construa um heap H com os k primeiros elementos de V, onde h é o maior elemento de H. Para cada um dos $n-k$ elementos restantes, se o mesmo é menor que o valor de h, então reduza a prioridade de h para o valor deste novo elemento e atualize as prioridades de H aplicando o procedimento de descida. Caso contrário, ignore o elemento. A complexidade deste algoritmo é $O\left(k + (n-k)\log k\right)$, onde o termo k segue da construção de H, enquanto $(n-k)\log k$ vem das atualizações do heap. Note que a complexidade desse algoritmo também é $O(n \log n)$ quando $k = O(n)$.

O Algoritmo 1 representa o algoritmo *Quickselect*. Este procedimento segue a mesma ideia do algoritmo de ordenação *Quicksort*, onde é feita uma escolha aleatória de um elemento do vetor, chamado *pivot*, em que dividimos o vetor em duas partes: os elementos menores que o *pivot* e os maiores que o *pivot*. Assim, se o k-ésimo menor elemento é menor que o *pivot*, podemos descartar a segunda parte dos elementos, caso contrário descartamos a primeira parte. O algoritmo reinicia recursivamente com os elementos do vetor resultante até que a primeira e última posições do vetor considerado coincidam, caso em que resta apenas um elemento a ser considerado. A chamada do procedimento é dada por $\textit{Quickselect}(V, 1, n, k)$, onde V é o vetor de entrada com n elementos e k é o elemento desejado. Note que podemos supor que V possui os elementos de 1 a n, correspondendo as posições dos elementos originais de V. A complexidade de pior caso deste algoritmo é $O(n^2)$, dado que podemos escolher o *pivot* de modo que eliminamos apenas um elemento a cada divisão do vetor. Porém, no caso médio esse algoritmo executa $O(n)$ passos.

Algoritmo 1: *Quickselect*$(V,\, l,\, r,\, k)$.

Entrada: Vetor de inteiros V de tamanho $r - l + 1$ e inteiro positivo k.
Saída: Posição correspondente ao k-ésimo menor elemento de V.

$pivot \leftarrow$ valor aleatório entre l e r;
$Valor\_pivot \leftarrow V[pivot]$;
$V[r] \Longleftrightarrow V[pivot]$;
$divisor \leftarrow l$;
para $i \leftarrow l, \ldots, r-1$ **faça**
  se $V[i] < Valor\_pivot$ **então**
    $V[dividor] \Longleftrightarrow V[i]$;
    $divisor \leftarrow divisor + 1$;
$pivot \leftarrow divisor$;
$V[pivot] \Longleftrightarrow V[r]$;
se $V[pivot] = k$ **então**
  Saída: *pivot*
senão
  se $V[pivot] < k$ **então**
    Saída: *Quickselect*$(V,\, pivot + 1,\, r,\, k - (pivot - l + 1))$
  senão
    Saída: *Quickselect*$(V,\, l,\, pivot - 1,\, k)$

---

3. (2,0) Descreva em palavras (informalmente) um algoritmo que execute a seguinte tarefa: Construir uma árvore binária T dados os seus percursos em pré-ordem e em ordem simétrica.

   Resposta: Por definição, o percurso em pré-ordem (P_{po}) possui como primeiro elemento a raiz r da árvore T, seguido dos elementos de $T_e(r)$ e dos elementos de $T_d(r)$, onde $T_e(v)$ e $T_d(v)$ denotam as subárvores esquerda e direita de v, respectivamente, para todo elemento $v \in T$. Para saber onde se dá a divisão entre os elementos de $T_e(r)$ e $T_d(r)$ em P_{po}, efetuamos a busca de r no percurso em ordem simétrica (P_{os}). Por definição, todos os elementos que precedem r em P_{os} pertencem a $T_e(r)$ e todos os elementos que sucedem r em P_{os} pertencem a $T_d(r)$. Assim sabemos exatamente que r é a raiz de T e quais os elementos pertencentes a $T_e(r)$ e $T_d(r)$. Podemos aplicar o mesmo procedimento a $T_e(r)$ e a $T_d(r)$ de forma recursiva utilizando os percursos em pré-ordem e ordem simétrica restritos aos elementos de cada subárvore analisada, até que haja apenas um vértice em ambos os percursos. Note que a cada passo ambos os percursos diminuem igualmente em tamanho.

4. (2,0) Dado um heap (lista de prioridades) T com n elementos, onde o elemento de prioridade máxima encontra-se na raiz (primeira posição de T), descrever o algoritmo para aumentar a prioridade de um dado elemento i de T. Suponha que a prioridade de um elemento j qualquer é denotada por $T[j].prio$

   Resposta: O Algoritmo 2 representa o procedimento de aumento de prioridade.

Algoritmo 2: Algoritmo de aumento de prioridade de um elemento em um heap.

Entrada: Vetor T representando uma lista de prioridade máxima, inteiro positivo i que terá a prioridade aumentada para o valor k.

se $k > T[i].prio$ **então**
    $T[i].prio \leftarrow k$;
    $j \leftarrow \lfloor i/2 \rfloor$;
    enquanto $j \geq 1$ e $T[i].prio > T[j].prio$ **faça**
        $T[i] \Longleftrightarrow T[j]$;
        $j \leftarrow \lfloor j/2 \rfloor$;

---

5. Desenhe uma árvore T de altura 4 que satisfaça às seguintes condições, em cada caso.

   (a) (1,0) T é uma árvore AVL e possui um número mínimo de nós (não se esqueça de colocar os valores das chaves dentro de cada nó)

Resposta: A Figura 1 mostra um exemplo de tal árvore. Uma árvore AVL possui número mínimo de nós quando cada nó não folha tem diferença de exatamente 1 entre as alturas de suas subárvores.

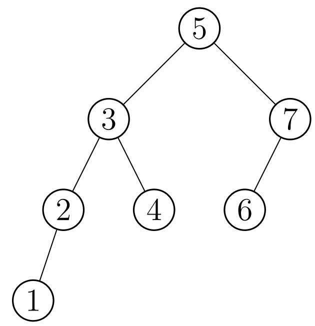

Figura 1: Árvore AVL de altura 4 e com o menor número de nós.

(b) (1,0) T é uma árvore de Huffman e possui um número máximo de nós (não se esqueça de colocar os valores dos pesos dentro de cada nó)

Resposta: A Figura 2 mostra um exemplo de tal árvore, onde as frequências são todas iguais a 1 para os 8 caracteres representados pelas folhas da árvore. Como toda árvore de Huffman é uma árvore estritamente binária, a Figura 2 possui o número máximo de nós Podemos ver que a árvore é formada nível a nível, das folhas até a raiz. Além disso, cada nível é completado antes do nível acima. Isso se deve porque todos os nós em um nível possuem valores menores que os dos níveis acima.

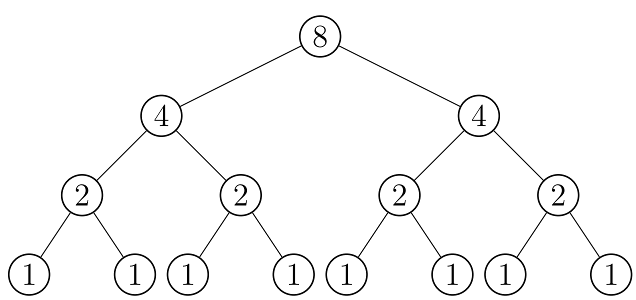

Figura 2: Árvore de Huffman com altura 4 e número máximo de nós.