Simulado da primeira avaliação

Algoritmos e Estrutura de Dados II (AE43CP)

Prof. Jefferson T. Oliva

Questionário

1. Dado o algoritmo abaixo:

```
int faz_algo(int v[], int n){
01.   int i, j, p, aux;
02.   aux = 0;
03.   for (i = 0; i < n / 2; i++)
04.      aux += v[i] + v[n - i - 1];
05.   for (i = n - 1; i > 0; i--){
06.       for (j = i; j > 0; j--)
07.           if (v[j] < v[j - 1]){
08.               aux = v[j];
09.               v[j] = v[j - 1];
10.               v[j - 1] = aux;
11.           }
12.   }
13.   return aux;
}
```

Faça:

a) - Calcule a quantidade de instruções, que o algoritmo pode executar no **melhor caso**. Para isso, mostre o passo a passo detalhado para o cálculo.

Resolução

- Linha 02: 1
- Linha 03-04: 1 + (n/2) * (7) + 2 = 7n/2 + 3
- Linha 06-11: 1 + i * (4) + 1 = 4i + 2
- Linha 05-12: 2 + (n - 1) (2 + 4i + 2) + 1 = 3 + 4n - 4 + 4 $\sum_{i=1}^{n-1} i$ = -1 + 4n + $2n^2$ - 2n = $2n^2$ + 2n - 1
- Linha 13: 1
- Total: 1 + 7n/2 + 3 + $2n^2$ + 2n - 1 + 1 = $2n^2$ + $\frac{11n}{2}$ + 4

b) - Calcule a quantidade de instruções que o algoritmo pode executar no **pior caso**. Para isso, mostre o passo-a-passo detalhado para o cálculo.

Resolução

- Linha 02: 1
- Linha 03-04: 1 + (n/2) * (7) + 2 = 7n/2 + 3
- Linha 06-11: 1 + i * (4) + 1 = 9i + 2
- Linha 05-12: 2 + (n - 1) (2 + 9i + 2) + 1 = 3 + 4n - 4 + 9 $\sum_{i=1}^{n-1} i$ = $\frac{9n^2}{2} - \frac{9n}{2}$ + 4n -1 = $\frac{9n^2}{2} - \frac{n}{2} - 1$
- Linha 13: 1
- Total: 1 + 7n/2 + 3 + $\frac{9n^2}{2} - \frac{n}{2} - 1 + 1$ = $\frac{9n^2}{2}$ + 3n + 4

Continua na próxima página. . .

c) - Determine a complexidade do algoritmo utilizando uma das seguintes notações: O, Ω ou Θ. Em seguida, interprete (explique) a complexidade do algoritmo de acordo com a notação que você escolheu. Pré-requisito: ter feito o item (a) ou (b).

Resolução: $\Theta(n^2)$, já que ambos os casos (melhor e pior) têm a taxa de crescimento na ordem de n^2.

d) - Determine o espaço extra necessário (em unidades de espaço) para executar o algoritmo. Em seguida, determine a complexidade de espaço utilizando uma das seguintes notações: O, Ω ou Θ.

Resolução: A complexidade de espaço do algoritmo acima é $\Theta(1)$, já são utilizadas 7 unidades de espaço (2 parâmetros, 4 variáveis locais e 1 endereço de retorno)

2. Implemente uma função que inverta uma lista encadeada.

Resolução

```
void inverter(ListaE *l){
   Cell *aux1, *aux2 = NULL;
   if (l1 == NULL){
       aux1 = l->head;
       while (aux1 != NULL){
           l->head = aux1->next;
           aux1->next = aux2;
           aux2 = aux1;
           aux1 = l->head;
       }
       l->head = aux2;
   }
}
```

3. Faça:

a) - Implemente uma função iterativa para fazer a inserção de uma chave (número inteiro) em uma árvore binária de busca. Na árvore binária não podem ser adicionadas chaves repetidas. Exemplo de protótipo de função: *Node* insercao_iterativa(Node *tree, int chave);*

Resolução:

```
Node* insercao_iterativa(int item, Node* tree){
   Node *auxP, *auxF;
   Node *novo = criar(item); // 5 instruções ou Θ(1) ou c (constante)
   if (tree == NULL) // 1 instrução
       tree = novo;
   else{
       auxF = tree; // 1 instrução
       while ((auxF != NULL) && (auxF->item != item)){ // 6 log2(n) + 3 instruções
           auxP = auxF;
           if (item < auxF->item)
               auxF = auxF->left;
           else
               auxF = auxF->right;
       }
       if (auxF == NULL){ // 1 instrução
           if (item < auxP->item) // 1 instrução
               auxP->left = novo; // 1 instrução (tanto faz a atribuição (esta ou do else))
           else
               auxP->right = novo;
       }else
           free(novo);
   }
   return tree; // 1 instrução
}
```

b) - Em que situação ocorre a inserção em uma árvore binária de busca no tempo médio (caso médio)?

Resolução: Quando a árvore está balanceada, ou seja, possui altura na ordem de $\log_2(n)$.

c) - Calcule a quantidade de instruções executadas no algoritmo implementado por você para o caso médio.

Resolução: $6\log_2(n) + 14$ ou $6\log_2(n) + 9 + \Theta(1)$ ou $6\log_2(n) + 9 + c$ (ver comentários no código acima referente à contagem de instruções)

d) - Com base no item (B), qual a complexidade de tempo do algoritmo para o caso médio?

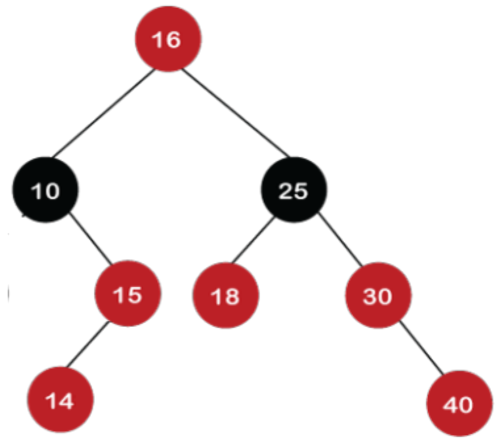
Resolução: $O(\log_2(n))$

e) - Qual a complexidade de espaço para a inserção em árvore binária de busca em um algoritmo recursivo para o caso médio? Como você chegou a tal conclusão?

Resolução: $O(\log_2(n))$, já que em uma árvore de altura $\log_2(n)$ serão realizadas na ordem de $\log_2(n)$ chamadas recursivas.

 Continua na próxima página...

4. Dada a árvore rubro-negra abaixo, onde apenas os nós 10 e 25 estão na cor preta:



a) - Por que a árvore acima está desbalanceada?

Resposta: além da raiz estar vermelha, há duas ocorrências de dois nós vermelhos consecutivos.

b) - Descreva a(s) operação(ões) necessária(s) para tornar a árvore balanceada. Em seguida, mostre (por meio de desenho) como ficaria a árvore rebalanceada. Na árvore resultante, indique quais nós estão na cor vermelha e quais estão na cor preta.

Resposta:

Primeiramente, a raiz deve ser mudada para cor preta.

Em seguida, para o caso dos nós 14 e 15, como nó tio de 14 é preto (nulo) , descendente esquerdo de 15, que é descendente direito de 10, então devemos fazer uma rotação dupla à esquerda. No final, o nó 14 muda para cor preta e passa a ser filho esquerdo de 16. Os nós 10 e 15 possuem a cor vermelha e passam a ser filhos de 10.

Para o último caso que devemos tratar, o nó tio de 40 é vermelho, então, para o balanceamento, basta mudar a cor dos nós pai (30) e tio (18) para preto e o avô (25), para vermelho,

 Continua na próxima página. . .

5. Dada uma árvore B de ordem $N = 4$, responda: (30 pontos)

a) - Uma árvore de altura 4 pode ter, no máximo, quantas páginas? Mostre como você chegou em tal quantia.

Resolução

Cada página de uma árvore B de ordem 4 pode ter até 4 descendentes. Então, para uma árvore de altura 4, a quantidade máxima de páginas é $\sum_{i=0}^{4} 4^i = 341$

b) - Uma árvore de altura 4 pode ter, no máximo, quantas chaves? Mostre como você chegou em tal quantia.

Resolução

De acordo o exercício anterior, uma árvore B de ordem 4 e de altura 4 pode ter, no máximo, 341 páginas. Como cada página nessa árvore pode ter 3 chaves, então, podemos ter, no total, $341 * 3 = 1023$ chaves.

c) - Mostre o porquê do algoritmo busca em uma árvore B de ordem N e com M chaves possui complexidade de tempo na ordem de $O(\log_2(M))$.

Resolução: em uma página, caso aplicada a busca binária, podem ser feitas até $\log_2(N)$ comparações. Caso a chave não seja encontrada, a busca é continuada em uma página filha. O processo é repetido até chegar em uma página folha. Como uma árvore B com M elementos pode possuir altura de $\log_N(M)$, no pior caso, são realizadas $\log_2(N) * \log_N(M) = \log_2(M)$. Logo, a complexidade da busca é na ordem de $O(\log_2(M))$.

 Continua na próxima página. . .

Anexos

- Teorema mestre:
  - Se $f(n) \in O(n^{log_b a-\epsilon})$ para a constante $\epsilon > 0$, então $T(n) \in \Theta(n^{\log_b a})$
  - Se $f(n) \in \Theta(n^{log_b a})$, então $T(n) \in \Theta(n^{\log_b a} \log_2 n)$
  - Se$f(n) \in \Omega(n^{\log_b a+\epsilon})$ para a constante $\epsilon > 0$, e se $af\left(\frac{n}{b}\right) \leq cf(n)$, para a constante $c < 1$ e n suficientemente grande, então $T(n) \in \Theta(f(n))$
- Estruturas de nó de árvores binárias.

```
typedef struct Node Node;

struct Node{
   int item;
   Node *left, *right;
};
```

- Estruturas de dados e protótipos de função para listas encadeadas:

```
typedef struct Cell{
       int item;
       struct Cell *prox;
   }Cell;

typedef struct{
       Cell *cabeca;
   }Lista;

typedef struct{
       Cell *topo;
   }Pilha;

typedef struct{
       Cell *ini, *fim;
   }Fila;

Cell* criar_celula(int chave);

Lista* criar_lista();

Pilha* criar_pilha();

Fila* criar_fila();

void empilhar(Pilha *p, int chave);

int desempilhar(Pilha *p);

void enfileirar(Fila *p, int chave);

int desenfileirar(Fila *p);

int pilha_vazia(Pilha *p);    int fila_vazia(Fila *f);

void liberar_lista(Lista *l);

void liberar_pilha(Pilha *p);

void liberar_fila(Fila *f);
```

 Continua na próxima página. . .

Observações

- Não é necessária a implementação dos protótipos acima.
- O acesso indevido aos dados de pilha de fila acarretará no desconto de 25% da nota.

- Séries

$\sum_{i=1}^{n} 2^i = 2^{n+1} - 1$

$\sum_{i=1}^{n} a^i = \frac{a^{n+1}-1}{a-1}$

$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$