MAC 5710 - Estruturas de Dados e sua Manipulação

Primeiro semestre de 2001 – Segunda Prova

1. (Valor: 2,0 pontos)

   (a) Em um *heap* de máximo, em que posições pode estar o elemento mínimo?

   Resposta: O elemento de mínimo pode estar em qualquer uma das folhas do *heap*. Isso ocorre já que, em um *heap* de máximo, abaixo de qualquer elemento só podem haver elementos menores que ele. Se o *heap* tem n elementos e está armazenado em um vetor $v[1..n]$, então o elemento mínimo pode estar em qualquer uma das posições de iíndice superior a $\lfloor n/2 \rfloor$. Essas posições correspondem exatamente às folhas do *heap*.

   (b) Quais das seqüências abaixo são *heaps*?

(i)	23	17	14	6	13	10	1	5	7	12
(ii)	20	7	18	5	6	12	15	2	1	4
(iii)	28	27	26	25	24	12	13	14	15	16

   Resposta:

   (i) Não é *heap*, pois o filho direito do 6, que está na posição 9, é o 7, que é maior que ele.

   (ii) É um *heap*: o elemento da posição j é maior que os elementos das posições $2j$ e $2j+1$, para todo j de 1 a 4, e o elemento da posição 5 é maior que o elemento da posição 10.

   (iii) É um *heap* (vale o mesmo que (ii)).

2. (Valor: 3,0 pontos) Suponha que uma expressão aritmética envolvendo os operadores $+$, $-$, $*$ e $/$ esteja armazenada em uma árvore binária da seguinte maneira. Os nós internos da árvore armazenam operadores e as folhas armazenam variáveis. Cada variável é representada por uma letra. Os operandos do operador guardado em um nó interno x são respectivamente as expressão armazenada na subárvore esquerda de x e a expressão armazenada na subárvore direita de x.

   Suponha as seguintes declarações:

```
struct no {
  char info;
  struct no *esq, *dir;
};
double valor (char ch);
```

   A função `valor` recebe uma letra e devolve o valor da variável representada por aquela letra.

   (a) Escreva uma função `avalia` que recebe como parâmetro um apontador para a raiz de uma árvore que armazena uma expressão e devolve o valor da expressão.

   Resposta:

```
double avalia (struct no *r) {
  double x, y;

  if (r->esq == NULL)
    return valor(r->info);
  x = avalia(r->esq);
  y = avalia(r->dir);
  switch (r->info) {
    case '+': return x+y;
    case '-': return x-y;
    case '*': return x*y;
    case '/': return x/y;
  }
}
```

(b) Escreva uma função `imprime` que recebe como parâmetro um apontador para a raiz de uma árvore que armazena uma expressão e imprime a expressão em notação posfixa.

Resposta:

```
void imprime (struct no *r) {
  if (r != NULL) {
    imprime(r->esq);
    imprime(r->dir);
    printf("%c", r->info);
  }
}
```

3. (Valor: 1,5 pontos) Mostre que, em uma árvore rubro-negra, um caminho de um nó x até uma folha abaixo de x tem comprimento no máximo duas vezes o comprimento de qualquer outro caminho de x até uma folha abaixo de x.

   Resposta: Por definição, em uma árvore rubro-negra, o número de nós negros no caminho de um nó x até qualquer uma das folhas abaixo de x é sempre o mesmo. Este número é a chamada altura rubro-negra do nó x. Claramente o comprimento de qualquer caminho do nó x até uma folha abaixo de x é pelo menos a altura rubro-negra de x. Além disso, como todo nó vermelho em uma árvore rubro-negra tem filhos negros, o comprimento de um caminho de x até uma folha abaixo de x tem comprimento no máximo o dobro da altura rubro-negra de x, ou seja, no máximo o dobro do comprimento de qualquer outro caminho de x até uma folha abaixo de x.

4. (Valor: 2,5 pontos) Considere um tipo abstrato de dados que admite as seguintes operações.

   `insere`(x): insere um elemento x;

   `remove`(y): remove um elemento **qualquer**, devolvendo uma cópia dele em y.

   Existem duas possibilidades: o tipo pode permitir duplicatas ou não. Se duplicatas são permitidas, a inserção deve ser feita independente de x já estar presente ou não, e na remoção apenas uma das cópias do elemento escolhido deve ser removida. Se duplicatas não são permitidas, a função `insere` deve verificar se x está presente e inseri-lo apenas se ele não estiver presente.

   (a) Descreva uma estrutura de dados o mais eficiente possível para cada um dos casos.

   Resposta:

   Com duplicatas: uma pilha ou uma fila.

   Sem duplicatas: uma árvore de busca balanceada.

   (b) Qual o custo de cada uma das duas operações, `insere` e `remove`, na sua estrutura de dados? ($O(n^2)$? $O(n \log n)$? $O(n)$? $O(\log n)$? $O(1)$?)

   Resposta:

	`insere`	`remove`
Com duplicatas	$O(1)$	$O(1)$
Sem duplicatas	$O(\log n)$	$O(\log n)$