Projeto e Analise de Algoritmos
Prova 2 – 20/10/2005

NOME: _____
MATRICULA: _____ [ ] Regular [ ] Especial
PROFESSORA: Diane Castonguay

JUSTIFIQUE TODAS AS SUAS RESPOSTAS

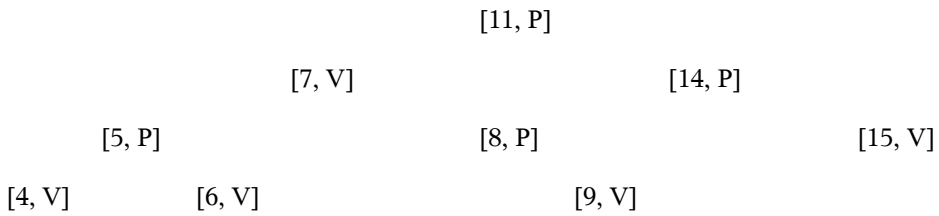
Questão 1 [5 pts] Árvore vermelho-preto

(a) Construir uma árvore vermelho -preto sabendo que:

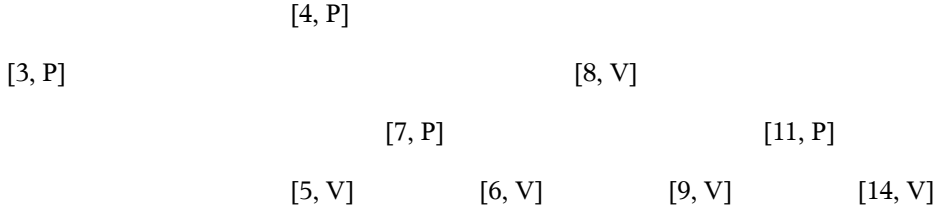
- Os valores das chaves são: 3, 4, 5, 7, 8, 10, 13, 15
- A raiz é o nó com chave 10
- O nó com chave 10 é o pai do nó com chave 7
- Os filhos do nó com chave 7 são os nós com chaves 4 e 8
- Os filhos do nó com chave 4 são os nós com chaves 3 e 5
- O nós com chaves 7 e 13 são vermelhos

Observação: Usar a representação de uma árvore vermelho-preto que omite a sentinela nil[T].

(b) Inserir a chave 3 da árvore vermelho-preto, usando o algoritmo RB-INSERT. Isto é, apresentar a árvore vermelho-preto obtida após a chamada a RB-INSERT(T, z) onde *chave*[z]=3 e T é a árvore vermelho-preto seguinte:



(c) Eliminar a chave 3 da árvore vermelho-preto, usando o algoritmo RB-DELETE. Isto é, apresentar a árvore vermelho-preto obtida após a chamada a RB-DELETE(T, z) onde *chave*[z]=3 e T é a árvore vermelho-preto seguinte:



<u>Questão 2</u> [5 pts] Programação dinâmica

Suponha que você tem uma máquina e um conjunto de n trabalhos a[1], a[2], ..., a[n] para processar nessa máquina. Cada trabalho a[j] tem um tempo de processamento t[j], um lucro p[j] >0 e um prazo final d[j]. A máquina só pode processar um trabalho de cada vez, e o trabalho a[j] deve ser executado ininterruptamente para t[j] unidades de tempo consecutivas. Se o trabalho a[j] for concluído em seu prazo d[j], você recebe um lucro p[j], mas, se ele for completado depois de seu prazo final, você recebe um lucro 0. Queremos achar uma programação (ordem para fazer os trabalhos) que otimiza o lucro.

Dicas:

-- Suponha que os trabalhos são ordenados tais que $d[1] \leq d[2] \leq ... \leq d[n-1] \leq d[n]$

-- Os trabalhos não feitos a tempo serão feitos no final, no decorrer da solução
suponhamos que esses trabalhos não são feitos.

-- As subestruturas ótimas do problema são da forma: para $0 \leq i \leq n$ e $0 \leq T \leq d[n]$
A programação de maior lucro dos trabalhos a[1], ... , a[i] feita antes do tempo T.

(a) Exiba as duas possiveis subestruturas ótimas que vem da subestrutura ótima de uma programação dos trabalhos a[1], ... , a[i] feita antes do tempo T quando $t[i] \leq T \leq d[i]$.

(b) Define recursivamente o valor de c[i, T], o lucro máximo de uma programação dos trabalhos a[1], ... , a[i] feita antes do tempo T.

CUIDADO:

- Se $d[i] < T$ então $c[i,T] = c[i, d[i]]$
- $c[0, T] = 0$
- Se $t[i] > T$, $c[i, T] = c[i-1, T]$

(c) Forneça um algoritmo de programação dinâmica para calcular o lucro máximo c[n, d[n]]. Este algoritmo calcula todos os c[i, T].

entradas:

- os vetores a, t, p e d.

saída

- uma matriz c

Observações: n = comprimento[a]
c[1, d[n]] é a quantidade máxima de lucro de uma programação dos trabalhos a[1], ... , a[n].

(d) Forneça um algoritmo PROGRAMAÇÃO para encontrar a programação que obtenha a quantidade máxima de lucro, supondo que todos os tempos de processamento são inteiros de 1 a m. Usar o algoritmo PRAZO, em anexo, que tem como entrada a matriz c e o vetor d, e como saída um vetor b tal que b[i] = 1 se a[i] foi concluido no prazo e b[i] = 0 senão.

entradas:

- a matriz c
- o vetor a

saída:

- um vetor L da ordem para efetuar os trabalhos (entradas: a[i]). É a programação.

Observação: Os trabalhos feitos a tempo vem primeiros na programação, na mesma ordem inicial, e depois vem os outros trabalhos em qualquer ordem.

ALGORITMOS

Inserção nas árvores vermelho-preto

```
RB-INSERT(T, z)
1. y ← nil[T]
2. x ← raiz[T]
3. enquanto x ≠ nil[T] faça
4.                  y ← x
5.                  se chave[z] < chave[x]
6.                              então x ← esquerda[x]
7.                              senão x ← direita[x]
8.                  fim-se
9. fim-enquanto
10. p[z] ← y
11. se y = nil[T]
12.         então raiz[T] ← z
13.         senão se chave[z] < chave[x]
14.                                     então esquerda[y] ← z
15.                                     senão direita[y] ← z
16.                             fim-se
17. fim-se
18. esquerda[z] ← nil[T]
19. direita[z] ← nil[T]
20. cor[z] ← VERMELHO
21. RB-INSERT-FIXUP(T, z)
```

```
RB-INSERT-FIXUP(T, z)
1. enquanto cor[p[z]] = VERMELHO faça
2.         se p[z] = esquerda[p[p[z]]]
3.                  então y ← direita[p[p[z]]]
4.                              se cor[y] = VERMELHO
5.                                      então cor[p[z]] ← PRETO                                 //caso 1
6.                                              cor[y] ← PRETO                                  //caso 1
7.                                              cor[p[p[z]]] ← VERMELHO                         //caso 1
8.                                              z ← p[p[z]]]                                    //caso 1
9.                                      senão   se z = direita[p[z]]
10.                                                     então   z ← p[z]                        //caso 2
11.                                                             LEFT-ROTATE(T, z)               //caso 2
12.                                             fim-se
13.                                             cor[p[z]] ← PRETO                               //caso 3
14.                                             cor[p[p[z]]] ← VERMELHO                         //caso 3
15.                                             RIGHT-ROTATE(T, p[p[z]])                //caso 3
16.                             fim-se
17.                 senão (igual a cláusula então trocando "direita" com "esquerda")
18.         fim-se
19. fim-enquanto
20. cor[raiz[T]] ← PRETO
```

ALGORITMOS

Eliminação nas árvores vermelho-preto

```
RB-DELETE(T, z)
1. se esquerda[z] = nil[T] ou direita[z] = nil[T]
2.         então y ← z
3.         senão y ← TREE-SUCESSOR(z)
4. fim-se
5. se esquerda[y] ≠ nil[T]
6.         então x ← esquerda[y]
7.         senão x ← direita[y]
8. fim-se
9. // não verificamos se x ≠ nil[T] para fazer a 10.
10. p[x] ← p[y]
11. se p[y] = nil[T]
12.         então raiz[T] ← x
13.         senão     se y = esquerda[p[y]]
14.                         então esquerda[p[y]] ← x
15.                         senão direita[p[y]] ← x
16.                   fim-se
17. fim-se
18. se y ≠ z
19.         então     chave[z] ← chave[y]
20.                   copiar dados satélite de y em z
21. fim-se
22. se cor[y] = PRETO
23.         então RB-DELETE-FIXUP(T, x)
24. fim-se
25. retorna y
```

```
RB-DELETE-FIXUP(T, x)
1. enquanto x ≠ raiz[T]  e cor[x] = PRETO faça
2.         se x = esquerda[p[x]]
3.                   então w ← direita[p[x]]
4.                         se cor[w] = VERMELHO
5.                               então     cor[w] ← PRETO                          //caso 1
6.                                         cor[p[x]] ← VERMELHO                    //caso 1
7.                                         LEFT-ROTATE(T, p[x])          //caso 1
8.                                         w ← direita[p[x]]                       //caso 1
9.                               fim-se
10.                              se cor[esquerda[w]] = PRETO e cor[direita[w]] = PRETO
11.                                       então cor[w] ← VERMELHO                   //caso 2
12.                                       x ← p[x]                                  //caso 2
13.                                       senão se cor[direita[w]] = PRETO
14.                                                 então     cor[esquerda[w]] ← PRETO    //caso 3
15.                                                           cor[w] ← VERMELHO     //caso 3
16.                                                           RIGHT-ROTATE(T, w)    //caso 3
17.                                                           w ← direita[p[x]]           //caso 3
18.                                                 fim-se
19.                                       cor[w] ← cor[p[x]]                        //caso 4
20.                                       cor[p[x]] ← PRETO                         //caso 4
21.                                       cor[direita[w]] ← PRETO                   //caso 4
22.                                       LEFT-ROTATE(T, p[x])          //caso 4
23.                                       x ← raiz[T]                               //caso 4
24.                              fim-se
25.                   senão (igual a cláusula então trocando "direita" com "esquerda")
26.         fim-se
27. fim-enquanto
28. cor[x] ← PRETO
```

ALGORITMOS

Rotação nas árvores vermelho-preto e algortimos em árvores de pesquisa binária

```
/* Assume que direita[x] ≠ nil[T] e p[raiz[T]] = nil[T] */
LEFT-ROTATE(T, x)
1. y ← direita[x]
2. direita[x] ← esquerda[y]
3. p[esquerda[y]] ← x
4. p[y] ← p[x]
5. se p[x] = nil[T]
6.                  então raiz[T] ← y
7.                  senão se x = esquerda[p[x]]
8.                                    então esquerda[p[x]] ← y
9.                                    senão direita[p[x]] ← y
10.                           fim-se
11. fim-se
12. esquerda[y] ← x
13. p[x] ← y


TREE-SUCESSOR(x)
1. se direita[x] ≠ NIL
2.                  então retorna TREE-MINIMUM(direita[x])
3. fim-se
4. y ← p[x]
5. enquanto y ≠ NIL e x = direita[y] faça
6.                  x ← y
7.                  y ← p[x]
8. fim-enquanto
9. retorna y


TREE-MINIMUM(x)
1. enquanto esquerda[x] ≠ NIL faça
2.                  x ← esquerda[x]
3. fim-enquanto
4. retorna x
```

Questão 2 (d)

```
PRAZO(c, d)
1. n ← comprimento[d]
2. T ← d[n]
3. para i ← n até 1 passo -1 faça
4.          se c[i,T] = c[i–1,T]
5.                  então b[i] ← 0
6.                  senão b[i] ← 1
7.                           T ← T – t[i]
8.          fim-se
9. fim-para
10. devolva b
```