SCC-203 ALGORITMOS E ESTRUTURAS DE DADOS II
Profa. Graça Nunes – 2º. Sem. 2012

Nome: _____ Nro. USP_____

Gabarito Prova 3 (27/6/2012)

1) (1.5) Associar características (a, ..., f) com o tipos: Árvore-B, Árvore-B*, Arvore-B$^+$, Hashing Extensível. Pode associar mais de uma a um mesmo tipo.

(a) Processamento Sequencial Eficiente — (b, d, f)Árvore-B

(b) Número máximo de *seeks* na busca $\log_k N$ (N= número de chaves)

(c) Prefere redistribuição à divisão durante a inserção — (b, c, d, f) Árvore-B*

(d) Processamento sequencial ineficiente — (a, b, e, f) Árvore-B$^+$

(e) Não precisa armazenar todas as chaves no Índice

(f) Adapta seu tamanho ao número de chaves — (d, f) Hashing Extensível

2) (2.0) Corrigir as afirmações, quando necessário.

(a) Árvores-B estão para memória secundária como AVL estão para RAM: ambas provêm uma maneira de usar árvores de busca binária que funcionam bem para dados dinâmicos.
OK

(b) A altura mínima de uma Árvore-B de ordem m, quando comparada com a altura de uma Árvore-B* de ordem m, é menor.
Falso. A altura mínima da Árvore-B* é menor do que a da Árvore-B, pois ela opta sempre por redistribuir ao invés de dividir na inserção, o que evita o crescimento da altura da árvore.

(c) Uma árvore-B não pode crescer em altura até que esteja 100% cheia.
Falso. A altura cresce durante um splitting, que faz com que 2 páginas fiquem com cerca de 50% de ocupação apenas.

(d) Concatenar ao invés de redistribuir, quando ocorre *underflow* numa página de uma Árvore-B, tem a vantagem de eliminar páginas e deixra a árvore menos propícia a *overflow* em futuras inserções.
Falso. A concatenação de fato elimina páginas, mas deixa páginas mais cheias, com mais chance de overflow.

(e) Numa árvore-B de ordem m, com N chaves, o número mínimo de páginas consultadas durante a busca de uma chave será ~ $\log_m$ N e, no máximo, ~ $\log_{m/2}$ (N/2)
OK

(f) Numa Árvore-B*, toda página, exceto a raiz, tem sempre 2/3 de ocupação.
Falso. Tem no mínimo 2/3 de ocupação, mas pode ter mais.

(g) Considerando as Árvores-B, as melhores páginas para se eleger num sistema de paginação virtual são as mais distantes da raiz.
Falso. É melhor manter na RAM as páginas mais próximas da raiz, que é de onde começa uma busca.

(h) Não é possível, após *overflow* em um *bucket* de um diretório de *hashing* extensível, fazer redistribuição de chaves entre este *bucket* e um *buddy bucket*, já que os endereços dos *buckets* têm que ser compatíveis com as chaves dos mesmos.
OK

3) (1.0) Considere a estrutura em C de uma página de Árvore-B:

```
In C:
struct BTPAGE {
    short   KEYCOUNT;          /* number of keys stored in PAGE */
    char    KEY[MAXKEYS];      /* the actual keys               */
    short   CHILD[MAXKEYS+1]; /* RRNs of children              */
} PAGE;

In Pascal:
TYPE
    BTPAGE = RECORD
                KEYCOUNT: integer;
                KEY     : array[1..MAXKEYS] of char;
                CHILD   : array[1..MAXCHILDREN] of integer
             END;
VAR
    PAGE : BTPAGE;
```

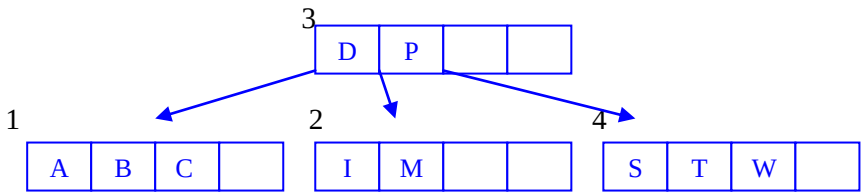
Complete o pseudo-código da função recursiva de busca por uma chave a seguir:

```
FUNCTION: search (RRN,                    //página a ser pesquisada
                  KEY,                   //chave sendo procurada
                  FOUNDRRN,              //página que contém a chave
                  FOUNDPOS)              //posição da chave na página
if _RRN=NIL_____________________ then
    return NOTFOUND   //chave de busca não encontrada
else
  leia página identificada por RRN e armazene em PAGE
  procure KEY em PAGE, fazendo POS igual a posição em
          que KEY ocorre ou deveria ocorrer
  if KEY encontrada then
     FOUNDRRN := _RRN_________
     FOUNDPOS := _POS_________
     return FOUND
    else return (search(PAGE.CHILD[POS], KEY, FOUNDRRN, FOUNDPOS))
    ____________________________________________________
      //a chave de busca não foi encontrada, procura a chave no nó filho
```

4) (2.0) Considere uma **árvore-B** de **ordem 5** (ou seja, cada página tem no máximo 4 chaves). Faça:

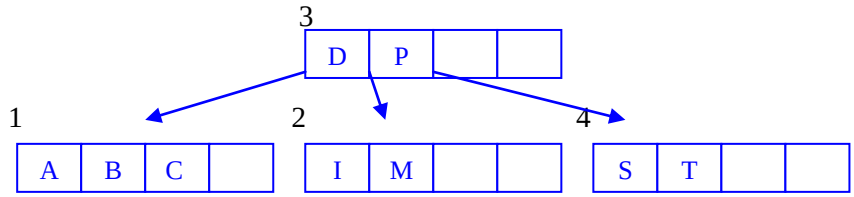
(a) (1.0) **Desenhe** a árvore construída a partir da inserção dos elementos **C, S, D, T, A, M, P, I, B e W** (nesta ordem). **Enumere as páginas com o número RRN**, e indique o que se pergunta a seguir.

Houve:
(a) Divisão de páginas ( ) Sim ( ) Não Qual(is)? _____
(b) Promoção de chaves? ( ) Sim ( ) Não Qual(is)? _____



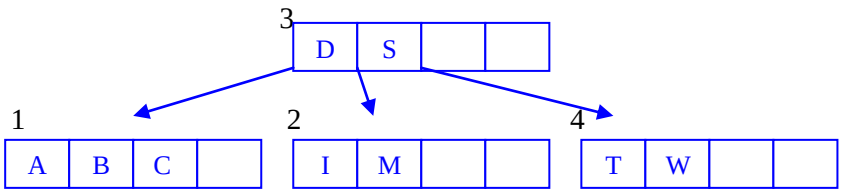
(b) (0.25) Desenhe a árvore após a remoção do elemento W da árvore do item (a).

Houve:
(a) Redistribuição de chaves? ( ) Sim ( ) Não
(b) Concatenação de páginas? ( ) Sim ( ) Não Qual(is)? _____



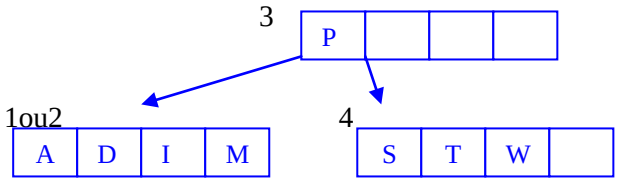
(c) (0.25) Desenhe a árvore após a remoção do elemento P da árvore do item (a).

Houve:
(a) Redistribuição de chaves? ( ) Sim ( ) Não
(b) Concatenação de páginas? ( ) Sim ( ) Não Qual(is)? _____

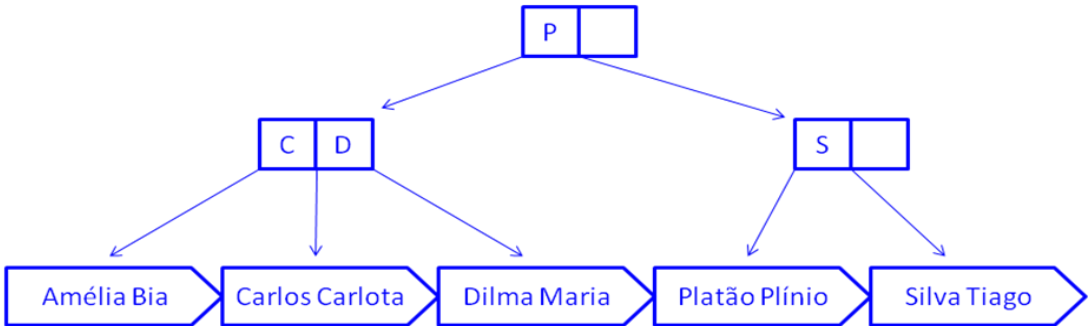


(d) (0.5) Desenhe a árvore após a remoção dos elementos B e C da árvore do item (a).

Houve:
(a) Redistribuição de chaves? ( ) Sim ( ) Não
(b) Concatenação de páginas? ( ) Sim ( ) Não Qual(is)? _____



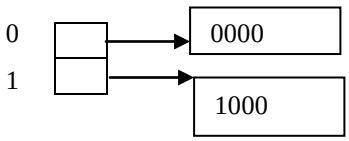
5) (2.0) Desenhe a **árvore-B+ de prefixos simples** a partir da inserção das chaves **Carlos, Silva, Dilma, Tiago, Amélia, Maria, Plínio, Platão, Bia e Cartola** (nesta ordem), assumindo que a páginas árvore têm **ordem 3** (ou seja, cada página tem no máximo 2 prefixos) e os blocos têm no **máximo 3 registros.**



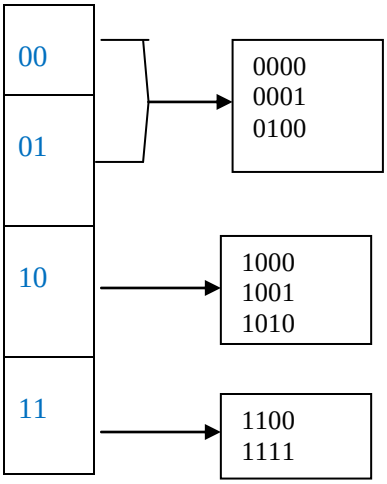
6) (1.5) Considere o **hashing extensível**, com um máximo de **3 elementos por bucket**, e que a função hash gera 4 bits para uma chave. Considerando a configuração inicial abaixo, simule a inserção de chaves que geram os seguintes endereços, desenhando o diretório após a sequência de inserção delimitada por *:

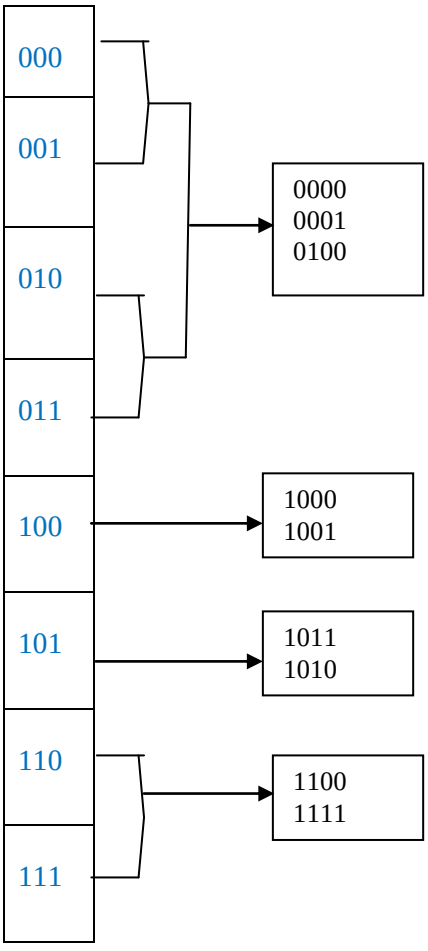
1001, 1010*, 1100, 0001, 0100, 1111*, 1011*

Configuração inicial:



Resp.:


00
01
10
11
0000
0001
0100
1000
1001
1010
1100
1111


000
001
010
011
100
101
110
111
0000
0001
0100
1000
1001
1011
1010
1100
1111