UNIVERSIDADE DE SÃO PAULO - ICMC
Departamento de Ciências da Computação
SCC-203 Algoritmos e Estruturas de Dados 2 - 2° Sem /2012
Profa.: Maria Cristina

3ª PROVA (27 de junho)

Aluno: _____

Nº USP: _____

Atenção: favor não destacar as folhas da prova.

1ª Q:	_____
2ª Q.	_____
3ª Q.	_____
4ª Q.	_____
5ª Q.	_____
NOTA:	_____

Questão 1 (2.0)

(a) (1.0) Se uma árvore-B de ordem 11, armazenada em disco, está armazenando 1.330 chaves e está cheia, ou seja, todas as suas páginas estão com o maior número possível de chaves, qual é o número máximo de páginas que a função de busca tem que acessar no disco para localizar uma chave qualquer nessa árvore? **Explique como chegou na resposta**.

Resp.
São 10 chaves por página:
10 chaves no nível da raiz = 10
11 * 10 chaves no nível seguinte = 110
11*11* 10 chaves no nível seguinte = 1210
Total: 1.330
A árvore tem 3 níveis, portanto são 3 acessos no máximo.

(c) (1.0) Considere as seguintes soluções para representar índices: Árvore-B, Arvore-B$^+$, Hashing Extensível. Quais delas admitem processamento sequencial eficiente? Quais admitem acesso aleatório eficiente?

Resp.
Árvore-B – acesso aleatório eficiente
Hashing extensível – acesso aleatório eficiente
Árvore-B+ acesso aleatório e acesso sequencial

Questão 2 (2.5)

Considere uma árvore-B de ordem 5. Faça:

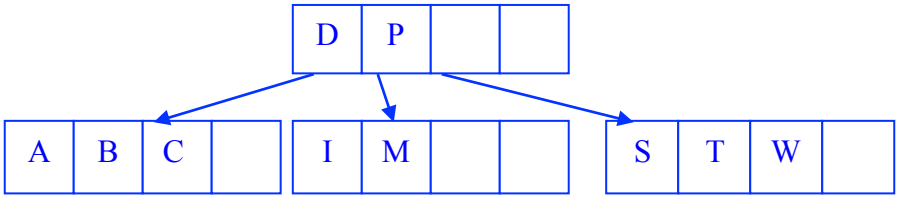
(a) (1.0) Desenhe a árvore construída a partir da inserção dos elementos C, S, D, T, A, M, P, I, B e W (nesta ordem).

(b) (0.5) Desenhe a árvore após a remoção do elemento W da árvore do item (a).

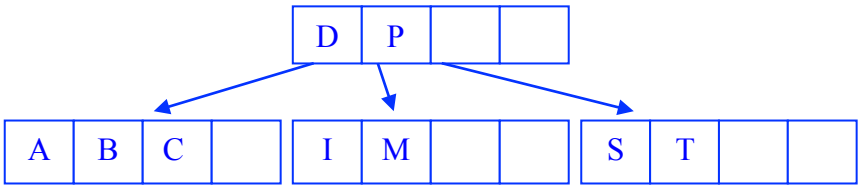
(c) (0.5) Desenhe a árvore após a remoção do elemento P da árvore do item (a).

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

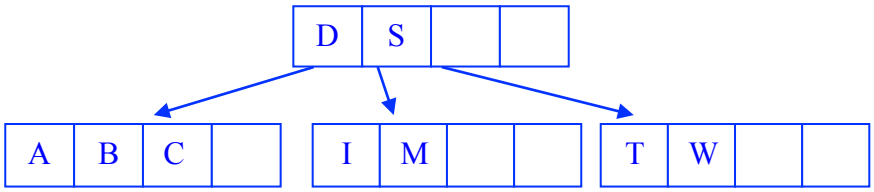
(a) (1.0) Desenhe a árvore construída a partir da inserção dos elementos C, S, D, T, A, M, P, I, B e W (nesta ordem).



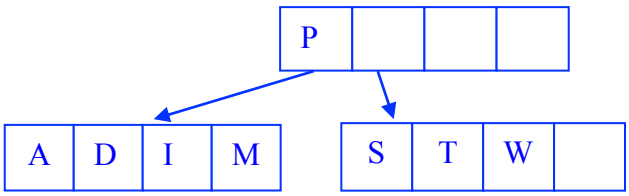
(b) (0.5) Desenhe a árvore após a remoção do elemento W da árvore do item (a).



(c) (0.5) Desenhe a árvore após a remoção do elemento P da árvore do item (a).



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



Questão 3 (2.0)

(a) (1.0) Complete o pseudocódigo do algoritmo recursivo de busca por uma chave em uma árvore B, abaixo.

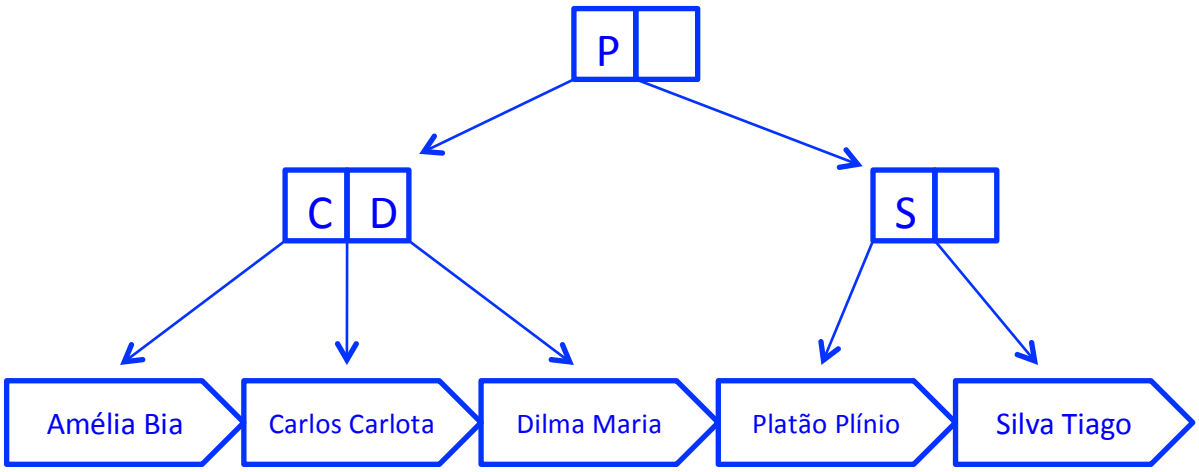
```
struct BTPAGE{
      short KEYCOUNT;       // número de chaves na página
      char   KEY[MAXKEYS];  // vetor de chaves
      short CHILD[MAXKEYS+1]; // vetor de ponteiros para filhos
} PAGE;

FUNCTION: search (RRN,           //página inicial da busca
                  KEY,           //chave sendo procurada
                  FOUND_RRN,     //página que contém a chave
                  FOUND_POS)     //posição da chave na página

if RRN == NIL_____________   then
     return NOT FOUND  //chave de busca não encontrada
else
  PAGE= READ(RRN); //carrega do disco a página no endereço RRN
  POS= Busca(KEY,PAGE); //POS = posição em que KEY ocorre(ria)
  if PAGE.KEY[POS] == KEY then
      FOUND_RRN= RRN_____________
      FOUND_POS= POS__________
      return _FOUND_________________
  else return (search(PAGE.CHILD[POS], KEY, FOUND_RRN,
    FOUND_POS)) ____________________________
       //chave de busca não foi encontrada, procura no nó filho
```

(b) (1.0) Indique como você modificaria o corpo e o cabeçalho da função acima para retornar, como um parâmetro de saída adicional, o número de acessos a disco efetuados na busca pela chave.

```
FUNCTION search (RRN: integer, KEY: char, var found_RRN, found_pos, Nacessos: integer): boolean;

/* found_RRN, found_pos e Nacessos passados por variável */
/* Nacessos igual a 0 na chamada inicial */

var pos: integer;
      page: BTPAGE;
      found_key: Boolean;

Begin
    if RRN == NIL then /* condição de parada para recursão */
          return NOT FOUND
    else
        readpage(RRN, page);
        Nacessos:= Nacessos + 1; /* mais um acesso a disco */
        found:= procura(KEY, page, pos);  /* busca por KEY em PAGE */
                if found then begin
                        found_RRN:= RRN;
                        found_pos:= POS;
                        return (FOUND)
                end
                else    /* desça para o nível inferior, seguindo a referência para o filho */
                        return(search(page.child[pos], KEY, FOUND_RRN,
FOUND_POS, Nacessos))
                end
    end
End;
```

Questão 4 (1.5)

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

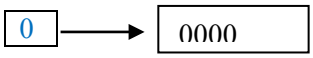


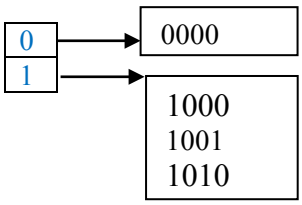
Questão 5 (2.0)

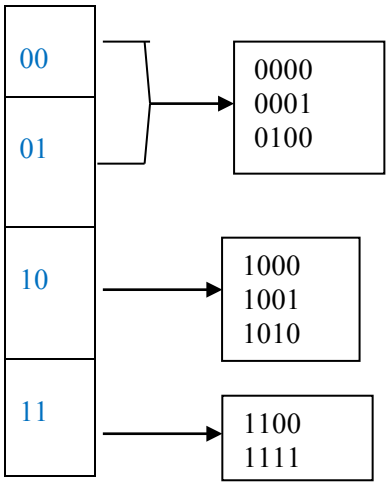
Considere o *hashing* extensível, assumindo que um *bucket* acomoda no máximo 3 elementos, e que a função *hash h(k)* gera endereços de 4 bits para uma chave de entrada *k*. Simule a inserção da sequência de chaves *k1*, ..., *k9*, para as quais são gerados os seguintes endereços, desenhando a configuração do diretório nos pontos indicados por *:

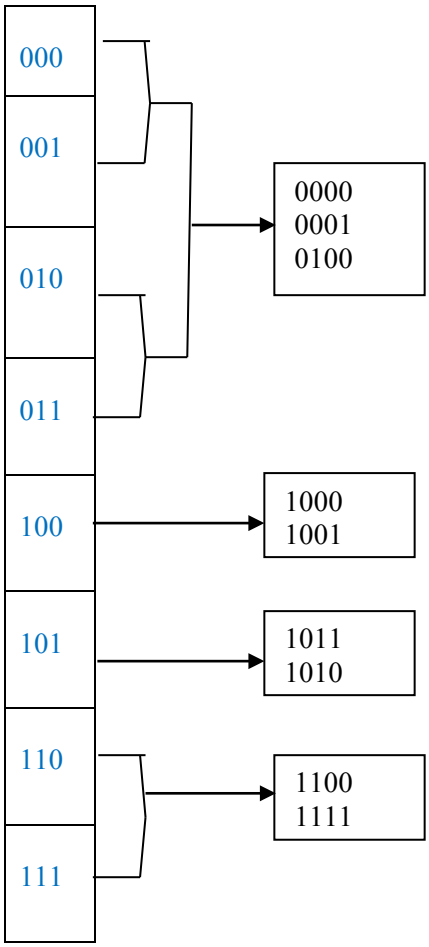
0000*, 1000, 1001, 1010*, 1100, 0001, 0100, 1111*, 1011*

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