Aluno(a):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Matrícula:\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| 1a) | 2.0 |  |
| 2a) | 2.0 |  |
| 3a) | 1.0 |  |
| 4a) | 2.0 |  |
| 5a) | 1.0 |  |
| 6a) | 2.0 |  |
|  | 10.0 |  |

1. A prova é individual e sem consulta.
2. A interpretação faz parte da questão. Em caso de dúvida escreva a dúvida e a sua interpretação na resposta.
3. O tempo de prova é 1:45 h.
4. As respostas devem seguir as questões. Caso precise de rascunho use o verso da folha.
5. A prova pode ser feita a lápis.
6. (2.0 pontos) Implemente de forma não recursiva, visitando o menor número possível de nós, uma função que calcule a *largura* de uma árvore de busca binária, definida como a diferença entre o maior e o menor valor dentre os valores de chave na árvore. Por simplicidade, assuma que os valores de chave são inteiros. A função deve ter o seguinte protótipo:

int abb\_largura (Abb\* r)**;**

A função recebe como entrada um apontador para a raiz da árvore e retorna a largura da árvore.

Adote a seguinte estrutura para os nós:

typedef struct \_abb Abb;

struct \_abb {

int chave;

Abb\* esq;

Abb\* dir;

};

Resp:

int abb\_largura (Abb\* r){

Abb\* min, max;

if (r==NULL) return 0;

min = r;

while(min->esq != NULL)

min = min->esq;

max = r;

while(max->dir != NULL)

max = max->dir;

return (max->chave – min->chave);

}

1. (2.0 pontos) Considere uma árvore de busca binária modificada em que cada nó *N* armazena também o menor e o maior valor dentre os valores de chave que ocorrem na sub-árvore com raiz *N*. Por simplicidade, assuma que os valões de chave são inteiros. Adote a seguinte estrutura para os nós:

typedef struct \_abb Abb;

struct \_abb {

int chave;

int min;

int max;

Abb\* esq;

Abb\* dir;

Implemente uma função que receba como entrada uma árvore de busca binária modificada como acima e verifique se a estrutura está *consistente*, ou seja, se os valores *min* e *max* para cada nó *N* de fato armazenam o menor e o maior valor dentre os valores de chave que ocorrem na sub-árvore com raiz *N*. A função deve visitar cada nó apenas 1 vez. A função deve ter o seguinte protótipo:

int abb\_min\_max (Abb\* r)**;**

A função recebe como entrada um apontador para a raiz da árvore e retorna 0, se a estrutura está inconsistente, e 1 se está consistente.

Resp:

int abb\_min\_max(Abb\* r)

{ if (r == NULL) return 1;

if (abb\_min\_max(r->esq))

if (abb\_min\_max(r->dir))

{

int flag;

if (r->esq != NULL)

flag = (r->min == r->esq->min); /\* ver Questão 3 \*/

else flag = (r->min == r->chave);

if (r->dir != NULL)

flag = (r->max == r->dir->max); /\* ver Questão 3 \*/

else flag = (r->max == r->chave);

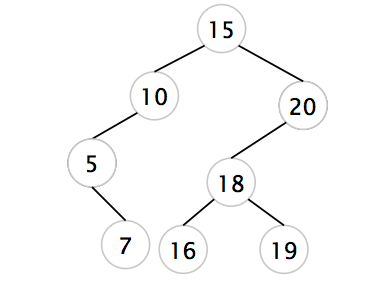
return flag;

};

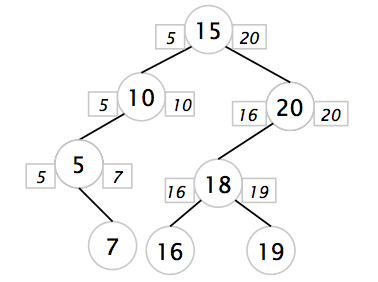
return 0;

}

1. (1.0 ponto) A estrutura de dados introduzida na Questão 2 é redundante? Argumente a favor ou contra esta afirmação, usando a árvore de busca binária abaixo como exemplo e completando os campos *min* e *max* de cada nó.



Resp.:



A estrutura é redundante. De fato, temos que

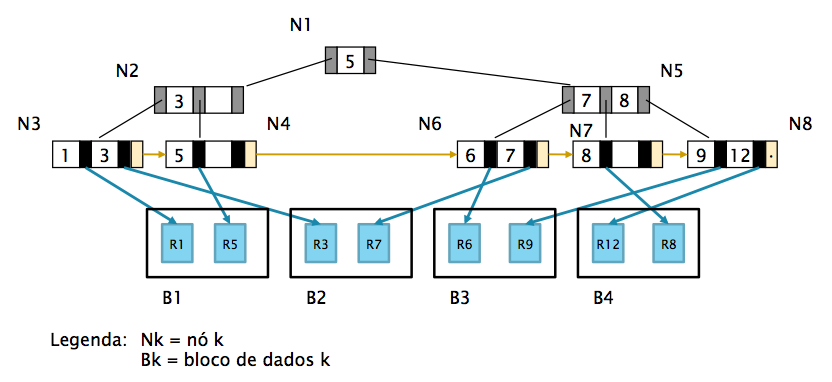
se r->esq não é nulo então r->min = r->esq->min senão r->min = r->chave

se r->dir não é nulo então r->max = r->dir->max senão r->max = r->chave

Logo, se e é um filho à esquerda, não é necessário armazenar e->min, pois e herda o valor de min do pai. Da mesma forma, se d é um filho à direita, não é necessário armazenar d->max, pois d herda o valor de max do pai.

1. (2.0 pontos) Considere a árvore B+ abaixo e um buffer pool capaz de armazenar 4 nós ou blocos de dados, indistintamente. Assuma que o buffer pool é gerenciado de tal forma que os nós ou blocos de dados há mais tempo sem serem usados são retirados do buffer pool, quando necessário.
   1. (1.0 ponto) Assuma que o buffer pool está inicialmente vazio, mas mantém o estado da pesquisa anterior quando uma nova pesquisa é iniciada. Mostre os estados por que passa o buffer pool quando são recuperados os dados associados às seguintes chaves, nesta ordem: 3, 5, 7. Explique sua resposta.
   2. (0.5 ponto) Assuma que o buffer pool está inicialmente vazio. Mostre os estados por que passa o buffer pool quando todas as chaves no intervalo [1,6] são pesquisadas. Use o algoritmo mais eficiente possível. Explique sua resposta.
   3. (0.5 ponto) É possível reorganizar os dados nos blocos de dados para melhorar o desempenho da estrutura? Explique sua resposta.

Em todos os itens, refira-se aos nós e blocos de dados usando os rótulos indicados na figura abaixo.

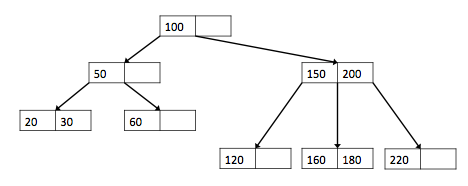


Resp:

|  |  |  |  |
| --- | --- | --- | --- |
| Operação | Buffer Pool | Acesso à memória secundária | Número total de acessos |
|  | (vazio) |  |  |
| (a) Pesquisa da chave 3 | N1  N2 N1  N3 N2 N1  B2 N3 N2 N1 | 1  1  1  1 | 4 |
| (a) Pesquisa da chave 5 | N1 B2 N3 N2  N2 N1 B2 N3  N4 N2 N1 B2  B1 N4 N2 N1 | 0  0  1  1 | 2 |
| (a) Pesquisa da chave 7 | N1 B1 N4 N2  N5 N1 B1 N4  N6 N5 N1 B1  B2 N6 N5 N1 | 0  1  1  1 | 3 |
| (b) Pesquisa no intervalo [1,6]  Acha a primeira chave, 1  Recupera os dados associados a 1  Recupera os dados associados a 3  Recupera N4, o nó seguinte a N3  Recupera os dados associados a 5  Recupera N6, o nó seguinte a N4  Recupera os dados associados a 6 | N1  N2 N1  N3 N2 N1  B1 N3 N2 N1  B2 B1 N3 N2  N4 B2 B1 N3  B1 N4 B2 N3  N6 B1 N4 B2  B3 N6 B1 N4 | 1  1  1  1  1  1  0  1  1 | 8 |

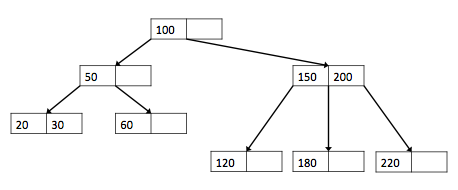
(c) É possível reorganizar os dados nos blocos colocando os registros nos blocos de dados consecutivamente por ordem de chave. No exemplo, apenas os blocos B1 e B2 seriam recuperados para obter os dados das chaves de 1 a 6.

1. (1.0 ponto) Remova sucessivamente as chaves 160 e 150 da árvore 2-3 abaixo. Mostre e comente todos os passos.



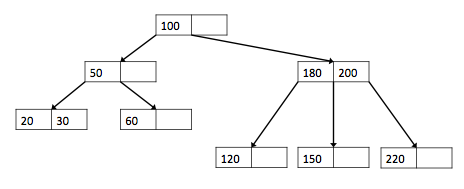
Resp.

*Remoção de 160*

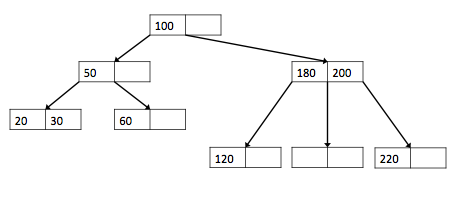


*Remoção de 150*

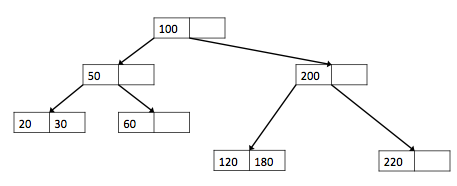
1) Troque 150 com a sua sucessora, 180



2) remova 150



3) Combine nós, rearrumando as chaves:



1. (2.0 pontos) Considere uma família de árvores definida de forma semelhante a árvores B, exceto que:

* Os nós possuem tamanho fixo com 128 bytes
* As chaves possuem tamanho variável, entre 8 bytes e 16 bytes
* Os ponteiros ocupam 4 bytes
* Em cada nó, há um campo a mais, de 4 bytes, indicando o número de chaves que o nó efetivamente armazena
  1. (1.0 ponto) Qual é o maior número de chaves que uma árvore de altura 3 armazena (uma árvore que só tem a raiz possui altura 0, por convenção)? Explique cuidadosamente sua resposta.
  2. (1.0 ponto) Qual é o número mínimo de chaves, estando todos os nós completamente preenchidos, que uma árvore de altura 2 armazena? Explique cuidadosamente sua resposta.

Resp:

a) O maior número de chaves será atingido quando todas as chaves possuírem o menor tamanho. Seja m o número de chaves que podem ser acomodadas em um nó. Cada chave é acompanhada por um ponteiro, exceto a primeira, que é acompanhada por 2 ponteiros. Há ainda 4 bytes adicionais para indicar o número de chaves. Para o caso de todas as chaves terem o menor tamanho, temos:

128 = (4 bytes para número de chaves) +

(4 bytes para o primeiro ponteiro) +

(4 bytes para cada ponteiro + 8 bytes de cada chave) \* m

m = (128-8)/12 = 120/12 = 10 chaves

Logo, cada nó terá 11 filhos.

Portanto, como a árvore possui altura 3, o número máximo de chaves será:

CM = 10 + 11\*10 + 11\*11\*10 + 11\*11\*11\*10 = 10 + 110 + 1.210 + 13.310 = 14.640

b) O menor número de chaves, quando os nós estão completos, será atingido quando todas as chaves possuírem o maior tamanho.

Cada nó completo terá p chaves de tamanho máximo, onde p é dado por:

128 = 4 + 4 + (4 + 16) \* p

p = (128-8)/20 = 120/20 = 6

Logo, cada nó terá 7 filhos.

Portanto, como a árvore possui altura 2, o número de chaves de tamanho máximo em nós completos será:

Cm = 6 + 7\*6 + 7\*7\*6 = 6 + 42 + 294 = 342