Coletânea de exercícios

8 de agosto de 2017

1 Listas

1 [**?**] Ilustre a inserção das chaves $1, 2, 3, 4, 5, 6$ no início de uma lista de inteiros inicialmente vazia.

2 [**?**] Ilustre a inserção das chaves $6, 5, 4, 3, 2, 1$ no início de uma lista de inteiros inicialmente vazia.

3 [**?**] Quantos apontadores precisam ser atualizados para inserir um nó em uma lista? Não deixe de considerar o caso em que o nó inserido passa a ser o primeiro nó da lista.

4 [**?**] A funcão abaixo deveria imprimir os itens de uma lista de inteiros, mas falha de duas formas. Analise-a e identifique as falhas.

```
typedef struct no {
  int info;
  struct no* prox;
} no;


void imprime(no* cabeca) {

  no* p = cabeca;
  while (p->prox != NULL) {
    printf("%d ",p->info);
    p = p->next;
  }
  printf("\n");
}
```

5 [1] Quantas posições de memória são acessadas para encontrar o elemento na posição k de um vetor de tamanho n? Explique.

6 [1] Quantos nós são acessados na memória para encontrar o elemento na posição k de um lista de tamanho n? Explique.

7 [14, adap. 4.2.5] Qual o número médio de nós acessados ao procurar um elemento numa lista desordenada? E numa lista ordenada?

8 [9, 4.6.3] Escreva uma função para remover de uma lista encadeada todos os elementos que contêm x.

9 [11, 3.2] Implement a routine `movenexttofront(struct node* t)` for a linked list that moves the node following the node pointed to by t to the beginning of the list.

10 [9, 4.6.4] Escreva uma função que remova de uma lista encadeada um nó cujo conteúdo tem valor mínimo.

11 [9, 4.7.3] Escreva uma função que faz uma cópia de uma lista dada.

12 [8, 3.10] Escreva uma função para verificar se duas listas encadeadas simples têm o mesmo conteúdo.

13 [9, 4.7.2] Escreva uma função que copia uma lista encadeada em um vetor.

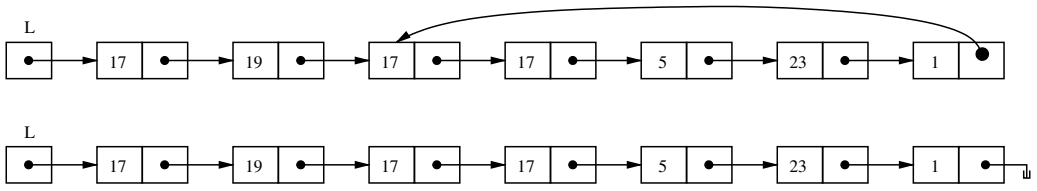
14 [8, 3.6] Remova da lista L_1 os nós cujas posições devem ser encontradas na lista ordenada L_2. Por exemplo, se $L_1 = (A, B, C, D, E)$ e $L_2 = (2, 4, 8)$, então o segundo e o quarto nós devem ser removidos da lista L_1 (o oitavo nó não existe) e, depois da remoção, $L_1 = (A, C, E)$.

15 [2] Considere a definição abaixo para o nó de uma lista encadeada que armazena inteiros.

```
typedef struct node {
  int data;
  struct node* next;
} node;
```

Vamos chamar uma lista de *zuada* se, ao invés de ter um apontador nulo em seu último nó, o último nó apontar para um nó da lista diferente da cabeça.

Escreva uma função em C para receber um apontador para o nó u cabeça de uma lista L e, se a lista for zuada então transformá-la em uma lista encadeada convencional, como ilustrado abaixo. Se a lista não for zuada, a função não deve modificar nada. O retorno da função deve ser 1 se a lista era zuada e 0 caso contrário.



16 Escreva uma função em C para receber um apontador para o nó u cabeça de uma lista L em que cada nó armazena um `char` e dividir a lista em duas outras listas. Na primeira lista devem ficar os $n/2$ primeiros nós de L (truncando a divisão). Na segunda lista devem ficar os nós restantes que contenham um caractere alfabético minúsculo. Os demais nós devem ser removidos da lista e desalocados.

2 Listas duplamente encadeadas

17 [**?**] Quantos apontadores precisam ser atualizados para inserir um nó em uma lista duplamente encadeada? Não deixe de considerar o caso em que o nó inserido passa a ser o primeiro nó da lista ou passa a ser o último nó da lista.

18 [1] Quantos nós de são acessados na memória para encontrar o elemento na posição k de um lista duplamente encadeada de tamanho conhecido n?

3 Listas circulares

19 [9, 4.8.4] Imagine n pessoas dispostas em círculo. Suponha que as pessoas estão enumeradas de 1 a n no sentido horário. Começando com a pessoa de número 1, percorra o círculo no sentido horário e elimine cada m-ésima pessoa enquanto o círculo tiver duas ou mais pessoas. Qual o número do sobrevivente? Escreva e teste uma função que resolva o problema.

20 Considere as definições de tipo abaixo para uma lista duplamente encadeada circular que armazena caracteres.

Escreva uma função em C que recebe apenas um apontador para um nó u de uma lista desse tipo e verifica se a lista armazena exatamente uma seqüência de caracteres da forma 0^n1^n a partir de u, para $n \geq 0$. Isto é, se percorrermos a lista começando em u encontraremos n caracteres 0 seguidos de n caracteres 1 e nada mais.

O retorno da função deve ser 1 se a lista armazena uma seqüência com essa forma, ou 0 caso contrário.

```
typedef struct node {
  char data;
  struct node* next;
  struct node* prev;
} node;
```

4 Listas generalizadas

21 [2] Ilustre a representação das listas generalizadas abaixo.

1. $A = ()$
2. $B = (())$
3. $C = ((), 10, 20, (1, 2, 30))$
4. $D = (A, B, C, 1)$ com compartilhamento
5. $D = (A, B, 1, C)$ com cópia
6. $E = (10, 20, E, E)$

22 Escreva uma função para remover uma lista generalizada recursivamente.

23 [2] Escreva uma função para contar o número de átomos em uma lista generalizada.

24 [2] A profundidade de um átomo é 0. A profundidade de uma lista generalizada vazia é 0. A profundidade de uma lista generalizada não-vazia é igual à profundidade máxima dentre os elementos dela mais 1. Escreva uma função para calcular a profundidade de uma lista generalizada.

25 [2] Escreva uma função para fazer uma cópia de uma lista generalizada.

26 [2] Escreva uma função para determinar se duas listas generalizadas são iguais.

27 Escreva uma função em C que converte uma lista encadeada simples em uma lista generalizada com os elementos repetidos e consecutivos colocados em sublistas diferentes. Por exemplo, para a lista

$$(aaaabccaadeeee)$$

a saída deve ser

$$((aaaa)(b)(cc)(aa)(d)(eeee)).$$

Defina os tipos que usar.

5 Bit arrays

28 [3] Suponha um vetor de bits implementado como um vetor de `unsigned char` e manipulado pelas macros abaixo.

```
#include <limits.h>

#define bit_set(A,i) ((A)[bit_slot(i)] |= bit_mask(i))
#define bit_clear(A,i) ((A)[bit_slot(i)] &= ~bit_mask(i))
#define bit_test(A,i) ((A)[bit_slot(i)] & bit_mask(i))

#define bit_slot(i) ((i) / CHAR_BIT)
#define bit_nslots(n) ((n) / CHAR_BIT + 1)
#define bit_mask(i) (1 << ((i) % CHAR_BIT))
```

1. Escreva uma linha de código C que cria um vetor para 22 bits inicializados com zeros.
2. Escreva três linhas de código C para setar os bits 7, 12 e 0.
3. Mostre o conteúdo do vetor de bits depois da execução do código nos itens anteriores.

29 [3] Suponha que a macro `bit_mask(i)` foi redefinida como aparece abaixo. Suponha que um vetor para 22 bits foi criado e inicializado com zeros. Depois os bits 7, 12 e 0 foram setados. Mostre o conteúdo do vetor de bits.

```
#define bit_mask(i) (((unsigned char) 128) >> ((i) % CHAR_BIT))
```

30 [2] Escreva um conjunto de macros ou funções para as operações `bit_set`, `bit_clear`, `bit_test` em uma matriz de bits.

6 Filas e Pilhas

31 [6, 9, p. 86] Qual a diferença entre uma pilha e uma fila?

6.1 Filas

32 [12, 4.31] A letter means put and an asterisk means get in the following sequence. Give the sequence of values returned by the get operation when this sequence of operations is performed on an initially empty FIFO queue.

E A S * Y * Q U E * * * S T * * * I O * N * * *

33 Considere uma fila implementada com vetor circular de tamanho 12. Suponha que as seguinter operações foram realizadas nessa ordem: enfileirar(13), enfileirar(19), desenfileirar, enfileirar(23), enfileirar(27), enfileirar(13), enfileirar(19), desenfileirar, desenfileirar, desenfileirar, enfileirar(31), enfileirar(7), enfileirar(2), enfileirar(19), enfileirar(7), enfileirar(2), enfileirar(19),

Ilustre o conteúdo do vetor circular.

34 Considere uma fila implementada com vetor circular de tamanho n. Para cada situação abaixo, explique como calcular em que posição estão o primeiro elemento na fila e o último elemento na fila. Verifique se há conflito para distinguir entre as situações de fila vazia, fila cheia, fila com apenas um elemento e fila com $n - 1$ elementos. Calcule também quantas posições do vetor podem ser efetivamente ocupadas pela lista em cada situação.

1. A fila mantém um índice `inicio` para o primeiro elemento na fila e um índice `fim` para o último elemento na fila.
2. A fila mantém um índice `inicio` para o primeiro elemento na fila e um índice `fim` para a posição imediatamente após o último elemento na fila.
3. A fila mantém um índice `inicio` para o primeiro elemento na fila e um contador `tam` do número de elementos na fila.

35 [10, adap. 3.2, p 2] Chamamos uma fila que permite inserção e remoção em ambas as extremidades de "fila simétrica". Explique como as operações `cria_fila`, `fila_vazia`, `insere_frente`, `insere_final`, `remove_frente` e `remove_final` podem ser implementadas para que a execução faça um número constante de operações.

36 [3] Um sistema de software X usa uma fila para inteiros implementada em C por um vetor circular. Quando foi projetado, o tamanho da fila foi definido como 10.000.000. Posteriormente verificou-se que na maior parte do tempo a fila tem menos que 10.000 elementos e ocasionalmente excede esse tamanho de forma imprevisível. Como o sistema X divide espaço com outras aplicações em um servidor, decidiu-se substituir a fila com vetor circular por uma fila híbrida.

A fila híbrida é composta por um vetor circular de tamanho 10.000, por um índice para a posição do vetor ocupada pelo primeiro elemento da fila, por um contador do número de elementos na fila híbrida e por dois apontadores para a cabeça e para o rabo de uma lista duplamente encadeada sem sentinelas.

Os tipos estão definidos abaixo.

```
typedef struct node {

  int data;
  struct node* next;
  struct node* prev;

} node;
```

```
typedef struct queue {

  int V[10000];
  int first;
  int size;
  node* head;
  node* tail;

} queue;
```

Enquanto for possível, novos elementos na fila são inseridos no vetor circular e quando não for possível inserir no vetor circular, novos elementos na fila são inseridos no início da lista. Quando há uma remoção e a lista não é vazia, um elemento é removido do vetor circular e um outro elemento é movido da lista para o vetor circular.

1. Escreva uma função em C que faz inserção em uma fila híbrida. Sua função deve receber um apontador para `queue` e o inteiro a ser armazenado. Ela deve retornar 1 se for bem sucedida ou 0 caso contrário. Sua função não deve supor a existência de outras funções para manipular vetores circulares ou filas. A fila recebida pela função poderá ter qualquer tamanho e será sempre consistente, isto é, os valores em seus campos estarão sempre corretos.

2. Escreva uma função em C que faz remoção em uma fila híbrida. Sua função deve receber um apontador para `queue`. Ela deve devolver o inteiro desenfileirado por referência e deve retornar 1 se for bem sucedida ou 0 caso contrário. Sua função não deve supor a existência de outras funções para manipular vetores circulares ou filas. A fila recebida pela função poderá ter qualquer tamanho e será sempre consistente, isto é, os valores em seus campos estarão sempre corretos.

6.2 Pilhas

37 [12, 4.6] A letter means push and an asterisk means pop in the following sequence. Give the sequence of values returned by the pop operations when this sequence of operations is performed on an initially empty LIFO stack.

E A S * Y * Q U E * * * S T * * * I O * N * * *

38 [4, 2.21, p. 78] É possível manter duas pilhas em um único vetor, se uma delas cresce da primeira posição do vetor, e a outra cresce da última posição. Escreva uma função PUSH(x, S) que insere um elemento x na pilha S, onde S é uma dessas duas pilhas. Inclua todas as verificações para erros nessa sua função.

39 [10, 2.2, p. 2] Escreva uma função que verifique se uma string de entrada é da forma `xCy`, tal que `x` é uma string composta por caracteres A e B e `y` é a string reversa de `x`. Por exemplo, a cadeia ABABBACABBABA é do formato especificado. A string de entrada deve ser lida seqüencialmente.

```
int xCy(char *str)
```

40 Escreva uma função para verificar se uma cadeia uma cadeia composta pelos delimitadores `[]()`{} é bem formada.

41 [6, 11, p. 63] Escreva um algoritmo que converta uma expressão em notação pré-fixa para pós-fixa.

6.3 Pilhas e Filas

42 [10, 3.3, p2] Podemos aproveitar uma implementação para fila simétrica para implementar uma fila ou uma pilha. Mostre como isso poderia ser feito.

43 Mostre como uma pilha pode ser implementada usando uma fila.

44 Mostre como uma fila pode ser implementada usando pilhas.

45 [7, p. 19] What is the final output?

1. Add $\{2, 4, 6, 8\}$ to a stack #1
2. Remove three items from stack, place in queue
3. Remove two items from queue, place in stack #2
4. Remove one item from stack #2, place in queue
5. Remove one item from stack #1, place in stack #2

7 Recursão

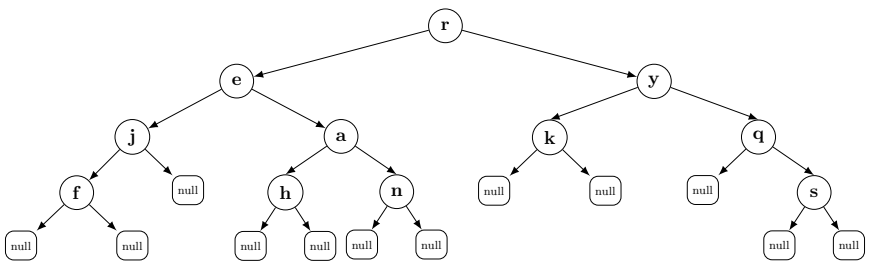
46 [13, 1.1] Responda se é certo ou errado: Todo procedimento recursivo deve incorporar terminações sem chamadas recursivas, caso contrário, ele seria executado um número infinito de vezes.

47 Para todo algoritmo recursivo, existe uma versão não-recursiva? Para todo algoritmo não-recursivo, existe uma versão recursiva? Explique.

8 Árvores

8.1 Representação

48 [3] Ilustre a árvore binária abaixo (a) em representação implícita, (b) em representação filho-irmão e (c) em representação costurada.



49 [3] Suponha que um inteiro usa i bytes e que um apontador usa a bytes. Suponha uma árvore binária com n nós. Cada nó armazena um inteiro.

1. Qual a melhor e a pior relação entre a memória ocupada pelos dados e a memória ocupada pelos apontadores na representação encadeada?
2. Qual a melhor e a pior relação entre a memória ocupada pelos dados e a memória ocupada pelo vetor na representação implícita, supondo que o vetor é o menor possível?

50 Suponha que uma árvore binária está implmentada de duas formas: encadeada e threaded. Em qual das duas implementações um percurso em-ordem deveria ser mais eficiente?

51 [2] Escreva as fórmulas para os filhos e para o pai de um nó supondo que a raiz de uma árvore binária implícita esteja na posição 1 do vetor.

52 [3] Escreva uma função para percorrer uma árvore representada implicitamente em pré-ordem.

53 Escreva uma função para receber uma árvore binária de inteiros em representação seqüencial e construir uma cópia da árvore em representação encadeada. Suponha que na representação seqüencial, posições com valor `UINT_MIN` não são nós da árvore.

54 Escreva uma função para converter uma árvore binária em que cada nó tem apontadores para os filhos esquerdo e direito em uma árvore binária com apontadores para filho e irmão.

8.2 Diversos

55 Escreva uma função para receber uma árvore em representação encadeada e dois índices de nós u e v e fazer v se tornar filho de u, considerando que u não é ancestral de v nem vice-versa e que u não tem filhos.

56 Escreva uma função recursiva para calcular a altura de um nó de árvore binária.

57 Escreva uma função para calcular a profundidade de um nó de árvore binária.

58 Escreva uma função recursiva para calcular o número de descendentes de um nó de árvore binária.

59 Escreva uma função recursiva para verificar se uma árvore binária é cheia.

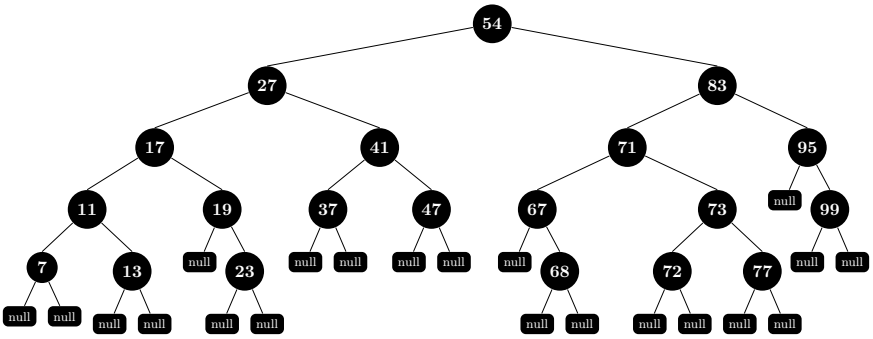
60 Escreva uma função recursiva para verificar se uma árvore binária é completa.

61 O fator de balanceamento de um nó u de uma árvore binária, denotado $u.bf$, é a diferença entre as alturas das subárvores enraizadas no filho da esquerda de u e no filho da direita de u. Escreva uma única função recursiva para calcular o fator de balanceamento de todos os nós de uma árvore binária.

62 O fator de carga de um nó u de uma árvore binária, denotado $u.lf$, é a diferença entre os números de nós nas subárvores enraizadas no filho da esquerda de u e no filho da direita de u. Escreva uma única função recursiva para calcular o fator de carga de todos os nós de uma árvore binária.

8.3 Percursos

63 Suponha que a operação visitar imprime o valor do dado armazenado em um nó. Para a árvore abaixo, mostre quais as seqüências produzidas pelos percursos em profundidade em-ordem, pré-ordem e pós-ordem e em largura.



64 Considere uma árvore T que armazena inteiros distintos, um em cada nó. Considere as ordenações de inteiros produzidas pela visitação pelos percursos em-ordem, pré-ordem e pós-ordem em T. A partir de quais conjuntos dessas ordenações ({pré}, {em}, {pós}, {pré,em}, {pré,pós}, {em,pós}, {pré,em,pós}) é possível reconstruir a árvore sem ambigüidades?

65 Escreva uma função para receber como entrada a ordem produzida por percursos em-ordem e pós-ordem e reconstruir uma árvore explícita.

66 Escreva um percurso em pós-ordem não-recursivo.

9 Buscas

67 Suponha um vetor com $n = 17 \times 10^6$ registros. Suponha que no pior caso uma busca seqüencial realiza $4n+8$ operações, que no pior caso uma busca binária realiza $6(\lfloor \log_2 n \rfloor + 1)$ e que para ordenar um vetor são realizadas $7n\lfloor \log_2 n \rfloor + 1$ operações no pior caso. Quantas buscas de pior caso precisam ser realizadas para que valha a pena ordenar o vetor e usar busca binária ao invés de usar busca seqüencial?

10 Árvores binárias de busca

68 Insira os números na seqüência abaixo, em ordem, em uma árvore binária de busca vazia inicialmente.

`17 26 25 08 01 32 55 48 36 80 50 96 21 93`
Qual a altura da árvore resultante?
Remova os números `96`, `08` e `17`.

69 Mostre a árvore binária de busca formada pela inserção dos números 1 a 15, nesta ordem. Mostre a árvore após a remoção das chaves 6 e 11.

70 [5, 12.2-1] Suppose that we have numbers between 1 and 1000 in a binary search tree, and we want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined?

a. 2, 252, 401, 398, 330, 344, 397, 363.
b. 924, 220, 911, 244, 898, 258, 362, 363.
c. 925, 202, 911, 240, 912, 245, 363.
d. 2, 399, 387, 219, 266, 382, 381, 278, 363.
e. 935, 278, 347, 621, 299, 392, 358, 363.

71 Escreva uma função para retornar a segunda maior chave (2º máximo) em uma árvore binária de busca.

72 [2] Escreva uma função recursiva para verificar se uma árvore é uma árvore binária de busca.

73 [5, 12.2-2] Write recursive versions of TREE-MINIMUM and TREE-MAXIMUM.

74 Escreva uma função para converter uma lista duplamente encadeada de inteiros ordenada em uma árvore binária de busca, sem alocar novos nós.

75 [2] Como você implementaria uma árvore binária de busca com chaves repetidas? Como seria a operação de inserção na sua árvore? Como seria a operação de remoção? Compare sua solução com uma árvore binária de busca convencional, em termos de memória ocupada e número de comparações para realizar uma busca.

76 Escreva uma função recursiva em C que recebe duas árvores binárias de busca e retorna 1 se elas são exatamente iguais ou 0 caso contrário.

77 [2] Escreva uma função para inserir em uma árvore binária de busca com representação costurada.

11 Árvores Balanceadas

78 Insira os números na seqüência abaixo, em ordem, em uma árvore AVL vazia inicialmente.

`17 26 25 08 01 32 55 48 36 80 50 96 21 93`
Qual a altura da árvore resultante?
Remova os números `96`, `08` e `17`.

79 Mostre a árvore AVL formada pela inserção dos números 1 a 15, nesta ordem. Mostre a árvore AVL após a remoção das chaves 6 e 11.

80 Toda árvore binária de busca completa é AVL?

81 A vantagem potencial da implemetação threaded para um percurso em ordem é equivalente, maior ou menor em uma árvore AVL, comparativamente a uma árvore binária de busca?

12 Hashing

82 [1] Considere a inserção das chaves $32, 11, 31, 4, 59, 28, 16, 77, 48$ em uma tabela de hashing de tamanho 13 e uma função calculada pelo método da divisão.

1. Ilustre a tabela resultante para resolução de colisões por encadeamento.
2. Ilustre a tabela resultante para resolução de colisões por sondagem com incremento unitário.
3. Ilustre a tabela resultante para resolução de colisões por sondagem com incremento quadrático com $c_1 = 1$ e $c_2 = 3$.
4. Ilustre a tabela resultante para resolução de colisões por sondagem com hashing duplo.

83 [1] Para cada item da questão anterior, ilustre a remoção das chaves 31 e 77.

84 [1] Suponha que uma tabela de hashing vai armazenar chaves compostas por pares de inteiros (x, y). Que estratégia poderia ser usada para computar uma função de hashing para o par que fosse independente da ordem dos elementos, isto é, $h(x, y) \neq h(y, x)$?

85 [2] Suponha que uma tabela de hashing vai armazenar chaves que são cadeias de caracteres ASCII. Como a função de hashing poderia ser computada?

86 [1] Suponha que você tem uma tabela de hashing de tamanho 101 que está 85% cheia. Para qual tamanho essa tabela deveria ser redimensionada para ficar aproximadamente 30% cheia?

87 [1] Suponha que os nós em cada lista em uma tabela de hashing com encadeamento são mantidos em ordem. Essa mudança vai melhorar o desempenho da tabela de hashing?

88 [1] Porque não deveríamos usar a função $h(k) = k \bmod 2^i$ para algum inteiro $i > 0$?

89 Considere uma tabela de hashing com colisões resolvidas por sondagem com hashing duplo. A tabela armazena apenas chaves inteiras.

Escreva uma função em C que dobra o tamanho de uma tabela como descrita acima, preservando os mesmos mecanismos. A função deve receber dois parâmetros: a tabela e seu tamanho atual. A função deve devolver um apontador para a nova tabela se ela for bem sucedida ou `NULL` caso contrário.

90 Escreva uma função em C que recebe duas tabelas de hashing, A de tamanho m e B de tamanho n, com chaves distintas e constrói uma terceira tabela de hashing $C = A \cup B$ de tamanho t. Suponha que m, n e t são primos e que $t > m + n$. Suponha também que as tabelas usam sondagem linear com hashing dupla com funções h_1 e h_2 já existentes.

13 Filas de prioridades

91 [1] Nas linhas da tabela abaixo temos as operações em uma fila de prioridades. Nas colunas, várias formas de implementar uma fila de prioridades.

Suponha uma fila de prioridades de tamanho n. Preencha as tabelas abaixo indicando o número esperado de comparações e movimentações de chaves que seriam realizadas em cada combinação.

Comparações de chaves	vetor	vetor ordenado	árvore binária de busca	árvore AVL	hashing dupla	heap
inserir						
remover o mínimo						
reduzir uma chave						

Movimentações de chaves	vetor	vetor ordenado	árvore binária de busca	árvore AVL	hashing dupla	heap
inserir						
remover o mínimo						
reduzir uma chave						

14 Heaps

92 Ilustre o heap de máximo construído para as chaves $17, 5, 56, 12, 33, 5, 17, 39, 1$. Ilustre o heap resultante após a remoção do máximo. Ilustre a inserção da chave 99.

93 [9, 10.1.1] Mostre que todo vetor decrescente é um heap de máximo. Mostre que a recíproca não é verdadeira.

94 [9, 10.4.2] Suponha que o vetor $v[1..n]$ é um heap de máximo. O seguinte fragmento de código rearranja o vetor em ordem crescente?

```
for (m = n; m >= 2; m--) {
  int x = v[1];
  for (j = 1; j < m; ++j)
    v[j] = v[j+1];
  v[m] = x;
}
```

95 Escreva uma função que recebe um heap de máximo representado implicitamente e devolve um apontador para a raiz de um heap de máximo com as mesmas chaves inteiras representado explicitamente. Defina os tipos que usar.

96 [9, 10.1.3] Escreva uma função que decide se um vetor v[1..m] é ou não um heap de máximo.

97 Escreva uma função em C com dois parâmetros, um apontador para um heap de mínimo implícito definido pelo tipo abaixo e um inteiro i, que remove a chave na posição i. Ao término da operação a estrutura deve continuar sendo um heap de mínimo.

```
struct heap {
  int *V;
  int max_size; // the length of V.
  int n;  // the length of the heap.
};
```

15 Matrizes esparsas

98 [2] Para a matriz abaixo, construa:

1. A representação como vetor de coordenadas.
2. A representação como lista de listas por linhas.
3. A representação como lista de listas por colunas.
4. A representação CSR.

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 3 & 7 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 2 & 6 & 5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & 0 & 9 & 8 \end{pmatrix}$$

99 [2] Suponha uma matriz de inteiros com dimensões $m \times n$, m' linhas não-vazias e k células não-zero. Suponha que um inteiro ocupa 4 bytes e que um apontador ocupa 8 bytes.

Para as representações abaixo (por linhas), calcule a quantidade de memória necessária para armazenar a matriz.

1. vetor de coordenadas
2. vetor de vetores
3. lista de coordenadas
4. lista de listas
5. CSR

100 [2] Suponha uma matriz de inteiros com dimensões $m \times n$ e k células não-zero representada em uma tabela de hashing com sondagem. Suponha que um inteiro ocupa 4 bytes. Suponha também que o fator-de-carga da tabela seja exatamente 75%.

1. Calcule o valor máximo de k para o qual a implementação ainda ocupa menos memória que a implementação direta de uma matriz $m \times n$.
2. Explique porquê o fator de carga pode não ser exatamente igual a 75% na prática.

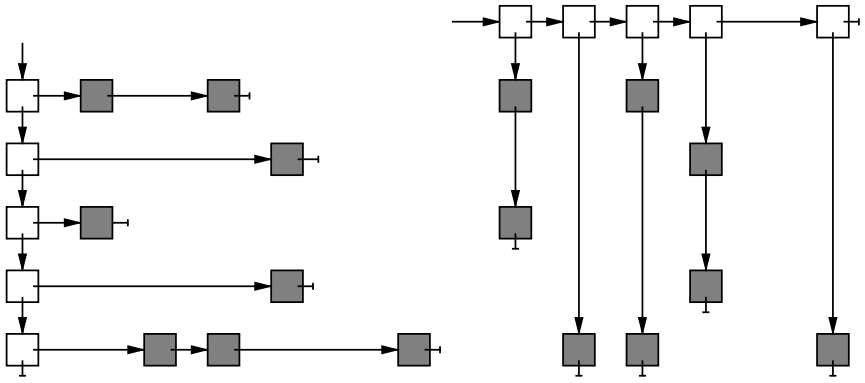
101 [3] Analise as representações vetor de coordenadas, vetor de vetores, lista de coordenadas, lista de listas e CSR quando submetidas às seguintes operações:

1. Alteração do valor de uma célula de 0 para não-zero.

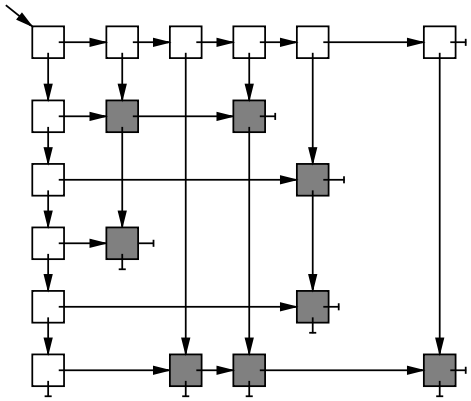
2. Alteração do valor de uma célula de não-zero para 0.

3. Alteração do valor de uma célula de não-zero para outro valor não-zero.

102 [2] Uma lista de listas pode tanto ser uma lista de linhas quanto uma lista de colunas.



Uma *lista ortogonal* combina as duas listas, por linhas e por colunas.



1. Que vantagens a lista ortogonal oferece?

2. Suponha uma matriz de fracionários com dimensões $m \times n$, m' linhas não-vazias, n' colunas não-vazias e k células não-zero. Suponha que um fracionário ocupe 8 bytes e que um apontador ocupe 8 bytes.

   (a) Calcule o número de bytes ocupados por uma matriz implementada por uma lista ortogonal.

   (b) Calcule o valor máximo de k para o qual a implementação ainda ocupa menos memória que uma matriz $m \times n$.

103 [**?**]

Suponha a representação por lista de coordenadas. Essa representação pode ter apenas uma coordenada ao invés de duas em cada nó da lista se codificarmos i, j de forma linearizada como $i * n + j$. Qual o maior valor de n para o qual a lista de coordenadas para uma matriz quadrada $n \times n$ pode ser representada de forma linearizada gastando menos memória que a lista de coordenadas usando pares i, j?

16 Ordenação

104 Quais dos algoritmos vistos em sala são estáveis?

105 Quais dos algoritmos vistos em sala são in-place?

106 [11, adap. 8.2] Qual dos algoritmos vistos em sala é mais rápido quando a entrada está em ordem crescente?

107 [11, adap. 8.3] Qual dos algoritmos vistos em sala é mais rápido quando a entrada está em ordem decrescente?

108 Suponha um conjunto de registros de alunos da Unicamp com campos RA e idade. É possível ordenar os registros por idade e, para uma mesma idade, ordenar por RA, usando apenas o insertion-sort como sub-rotina, sem modificar o algoritmo? É possível ordenar os registros por idade e, para uma mesma idade, ordenar por RA, usando apenas o quicksort como sub-rotina, sem modificar o algoritmo?

109 [2] O algoritmo de ordenação por inserção binária é uma variação do insertion-sort e funciona da seguinte forma. A entrada é um vetor de tamanho n. A busca pela posição para inserir o elemento $A[j]$ usa uma busca binária em $A[0, j-1]$. Depois da busca binária, os elementos maiores que $A[j]$ são deslocados para a direita. Esse algoritmo proporciona uma melhoria de desempenho em relação ao insertion-sort?

110 Qual o comportamento do quicksort quando todas as chaves são iguais?

111 Uma forma eficiente de implementar o quicksort é interromper a recursão quando o subvetor se torna pequeno e usar o insertion-sort. Quando o vetor é suficientemente pequeno o insertion-sort costuma ser mais rápido. Digamos que o tamanho ótimo para interromper o quicksort e executar o insertion-sort em um certo ambiente seja k. Escreva o algoritmo para essa ordenação híbrida.

112 Considere as seguintes situações e os algoritmos insertion-sort, counting-sort, merge-sort, heap-sort, quicksort, Shell-sort e bubblesort. Para cada situação, indique qual ou quais algoritmos seriam mais eficientes, justificando brevemente.

a) Um vetor com n números distintos em ordem crescente.

b) Um vetor com n números distintos em ordem decrescente.

c) Um vetor com n números, sendo $\log_2 n$ números distintos e tais que cada número aparece aproximadamente $n/\log_2 n$ vezes.

d) Um vetor com n números iguais.

e) Um vetor com n números distintos em uma ordem qualquer

113 [2] Explique como o algoritmo de ordenação abaixo funciona.

```
void shake-sort(int A[], int l, int r) {
  int i, aux;

  while (l<r) {
    for (i=l; i<r; i++)
      if (A[i] > A[i+1]) {
        aux = A[i];  A[i] = A[i+1];  A[i+1] = aux;
```

```
      }
    r--;

    for (i=r; i>l; i--)
      if (A[i] < A[i-1]) {
        aux = A[i];  A[i] = A[i-1];  A[i-1] = aux;
      }
    l++;
  }
}
```

114 [2] Explique como o algoritmo de ordenação abaixo funciona.

```
void tree_selection(int* A, int n) {
  int* m1 = malloc(2*n*sizeof(int));
  int* m2 = malloc(2*n*sizeof(int));
  int i,j;

  for (i=n; i<2*n; i++) {
    m1[i] = A[i-n];
    m2[i] = i;
  }

  for (i=n-1; i>0; i--) {
    if (m1[2*i] <= m1[2*i+1]) {
      m1[i] = m1[2*i];
      m2[i] = m2[2*i];
    }
    else {
      m1[i] = m1[2*i+1];
      m2[i] = m2[2*i+1];
    }
  }

  for (j=0; j<n; j++) {
    A[j] = m1[1];
    i = m2[1];
    m1[i] = INT_MAX;

    i = i/2;
    while (i>0) {
      if (m1[2*i] <= m1[2*i+1]) {
        m1[i] = m1[2*i];
        m2[i] = m2[2*i];
      }
      else {
        m1[i] = m1[2*i+1];
        m2[i] = m2[2*i+1];
      }
      i = i/2;
    }
  }
```

```
era uma vez um lobo mau mas chapeuzinho ouviu os conselhos da vovo
```

Figura 1: Um conjunto de cadeias.

```
  free(m1);
  free(m2);
}
```

115 Escreva uma versão iterativa do Mergesort.

116 Escreva uma versão iterativa do Quicksort.

17 Dicionários

117 [2] Ilustre o trie para as cadeias na Fig. 1.

118 [2] Supondo que um trie tenha sido implementado usando um vetor de apontadores para os seus filhos em cada nó, calcule o volume de memória ocupado por ele para as cadeias na Fig. 1.

119 [2] Explique como seria uma tabela de hashing para as chaves na Fig. 1. Quanto espaço de memória seria ocupado por ela?

18 Conjuntos

120 [1] Complete a tabela abaixo com o número aproximado de operações de acesso à memória (leitura e escrita) no pior caso para realizar as operações de conjuntos nas implementações listadas.

	vetor	vetor ordenado	lista	lista ordenada	bit-array
inserir elemento um subconjunto					
remover elemento de subconjunto					
testar continência					
união					
interseção					
listar os elementos de um subconjunto					

121 [2] Escreva uma função em C para computar a interseção de dois subconjuntos representados como um vetor ordenado.

122 [2] Escreva uma função em C para computar a união de dois subconjuntos representados como um vetor ordenado.

123 [2] Suponha que há n pessoas rotuladas entre 1 e $n-1$. Também são dados m pares de pessoas, no formato (i, j) indicando que a pessoa i tem afinidade com a pessoa j. É necessário encontrar o maior grupo de pessoas em que cada uma tenha afinidade com pelo menos uma outra. Escreva um algoritmo para resolver o problema usando conjuntos disjuntos e as operações definidas sobre eles.

19 Grafos

19.1 Representação

124 [2] Represente o grafo da Fig. 2 usando

1. matriz de adjacências.
2. listas de adjacências.
3. vetor de adjacências.

125 [2] Represente o grafo da Fig. 3 usando

1. matriz de adjacências.
2. listas de adjacências.
3. vetor de adjacências.

126 [2] Suponha um grafo com n vértices e m arestas. Suponha que um inteiro ocupa 4 bytes e que um apontador ocupa 8 bytes. Quais os valores de m (em relação a n) para os quais a lista de adjacências é mais econômica no uso de memória que a matriz de adjacências?

127 Proponha uma matriz de adjacências modificada para:

1. Representar grafos em que cada aresta tem um peso e uma cor (números inteiros).
2. Representar grafos que não são simples.
3. Representar grafos que não são simples e com pesos nas arestas.

128 Proponha uma lista de adjacências modificada para representar as mesmas situações do exercício anterior.

129 Suponha que as vizinhanças em um grafo por lista de adjacências são mantidas ordenadas.

1. Há algum ganho na operação para verificar se uma aresta (u, v) pertence ao grafo?
2. Há algum ganho na operação para percorrer a vizinhança de um vértice u do grafo?

130 Suponha que um grafo com n vértices e m arestas seja representado como uma matriz de bits. Isto é, ao invés de usar uma matriz de inteiros usamos uma generalização do bit array para duas dimensões.

1. Quantos bytes são usados para armazenar o grafo?
2. Como fica o desempenho da operação para verificar se uma aresta (u, v) pertence ao grafo?
3. Como fica o desempenho da operação para percorrer a vizinhança de um vértice u do grafo?

131 Suponha que um grafo com n vértices e m arestas seja representado como uma matriz esparsa CSR.

1. Qual o número máximo de arestas para o qual a CSR usa menos memória que a matriz de adjacências?
2. Como fica o desempenho da operação para verificar se uma aresta (u, v) pertence ao grafo?
3. Como fica o desempenho da operação para percorrer a vizinhança de um vértice u do grafo?

132 Suponha que um grafo com n vértices e m arestas foi implementado da seguinte forma: um vetor de tamanho n contém apontadores para tabelas de hashing que armazenam os vizinhos de cada vértice. Ou seja, nessa implementação, ao invés de listas de adjacências temos tabelas de hashing de adjacências. Cada tabela de hashing armazena até $n - 1$ valores, tem tamanho t igual ao primeiro primo maior ou igual a $1.15n$ e resolve suas colisões por sondagem com hashing duplo. Analise esta solução, comparando-a com a lista de adjacências, no que diz respeito à quantidade de memória ocupada e ao número de acessos à memória para realizar as operações (i) verificar se uma aresta (i, j) pertence ao grafo e (ii) percorrer todos os vizinhos de um vértice.

133 Seja $G = (V, E)$ um grafo orientado representado por listas de adjacências. Suponha que queremos encontrar o vértice que tem o maior grau de entrada. Em caso de empate, queremos todos eles. Escreva um algoritmo para resolver esse problema.

134 Seja $G = (V, E)$ um grafo orientado representado por listas de adjacências. Escreva um algoritmo para construir o grafo $G' = (V, E')$, que tem os mesmos vértices que G mas as arestas têm a orientação inversa. Como G' poderia ser usado para resolver o exercício anterior?

19.2 Buscas

135 Execute uma busca em largura no grafo da Fig. 2 a partir do vértice d. Desenhe a árvore busca em largura resultante, com os tamanhos dos caminhos.

136 Execute uma busca em largura no grafo da Fig. 3 a partir do vértice e. Desenhe a árvore busca em largura resultante, com os tamanhos dos caminhos.

137 Execute uma busca em profundidade no grafo da Fig. 2. Desenhe a floresta de busca em profundidade resultante.

138 Execute uma busca em profundidade no grafo da Fig. 3. Desenhe a floresta de busca em profundidade resultante.

139 Escreva uma função em C que recebe um grafo G representado como uma lista de adjacências e devolve um vetor de inteiros C, tal que se dois vértices i e j de G estão no mesmo componente conexo então $Ci] = C[j]$ e se dois vértices r e s de G estão em componentes conexos diferentes então $C[r] \neq C[s]$.

Sua função deve usar conjuntos disjuntos como estrutura de dados auxiliar para construir V. Assuma que as operações básicas para essa estrutura estão disponíveis. Defina os tipos de dados que usar.

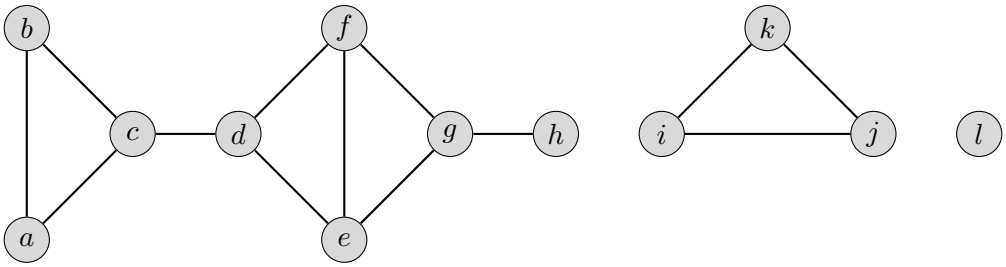


Figura 2: Grafo G_1.

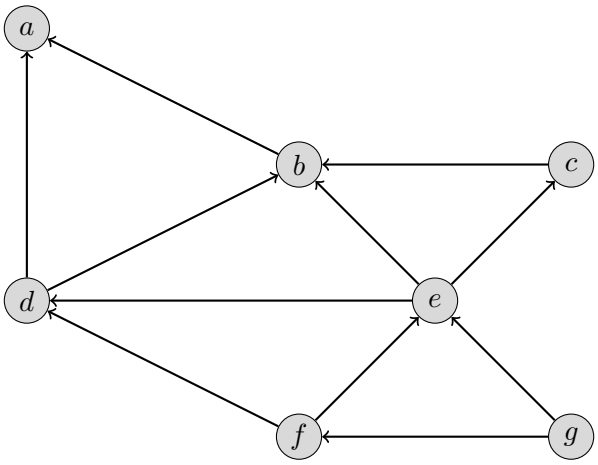


Figura 3: Grafo G_2.

20 Árvores B

140 [5, adap. 18.2-1] Show the results of inserting the keys

$$F, S, Q, K, C, L, H, T, V, W, M, R, N, P, A, B, X, Y, D, Z, E$$

in order into an empty B-tree with degree 2.

141 [**?**, 2.7] Quantas chaves pode conter uma árvore B de ordem m e de altura h?

142 adap. [**?**, 9.5] Suponha que uma árvore B indexa um arquivo não-ordenado com n registros. A altura da árvore é d. Quais os números mínimo e máximo de acessos à memória secundária para:

1. recuperar um registro?
2. adicionar um registro?
3. remover um registro?
4. recuperar todos os registros em ordem?

143 [**?**, 9.11] Discuta os impactos de armazenar os dados juntamente com as chaves dentro da árvore B ou de armazenar os dados em um arquivo separado.

144 [**?**, 3.7] Escreva uma função que imprima o conteúdo de uma árvore B na ordem ascendente.

145 adap. [**?**, 5.7] As árvores B são imunes à ordem dos dados que entram? Construa árvore B de ordem 2 primeiro para a sequência 1, 5, 3, 2, 4, e então para a sequência 1, 2, 3, 4, 5. É melhor inicializar as árvores B com dados ordenados ou com dados em uma ordem aleatória?

146 As definições abaixo são para uma árvore B de grau t, sendo que t é uma constante já definida.

```
typedef struct b_node {

  int keys[2*t-1];              // chaves em ordem crescente
  struct b_node* children[2*t]; // apontadores para os filhos
  char n;                       // número de chaves armazenadas no nó
  char leaf;                    // flag, 1 nas folhas e 0 nos nós internos

} b_node;
```

As definições abaixo são para uma árvore binária de busca.

```
typedef struct bin_node {

  int key;
  struct bin_node* left;
  struct bin_node* right;

} bin_node;
```

Escreva uma função recursiva em C que recebe um apontador do tipo `b_node*` para a raiz de uma árvore B e converte essa árvore B em uma árvore binária de busca com o mesmo conjunto de chaves. O retorno dessa função deve ser um apontador do tipo `bin_node*` para a raiz da árvore binária de busca que foi construída. Assuma que a árvore B dada como entrada está consistente.

21 Dicas

Ex. 97 Uma estratégia de solução possível é mover a chave na última posição do heap para a posição i e reduzir o tamanho do heap. Ao ajustar a nova chave na posição i, ela pode ter que subir ou descer, não é correto supor que ela só pode descer.

Referências

[1] 2012.

[2] 2013.

[3] 2015.

[4] John E. Hopcroft Alfred V. Aho and Jeffrey D. Ullman. *Data Structures and Algorithms*. Addison-Wesley, 1983.

[5] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 3rd edition, 2009.

[6] Vinu V. Das. *Principles of Data Structures using C and C++*. New Age International, 2008.

[7] Desconhecido. http://ocw.mit.edu/courses/civil-and-environmental-engineering/1-00-introduction-to-computers-and-engineering-problem-solving-spring-2012/recitations/MIT1_00S12_REC_12.pdf, 2012.

[8] Adam Drozdek. *Estruturas de dados e algoritmos em C++*. Thomson, 2002.

[9] Paulo Feofiloff. *Algoritmos em linguagem C*. Elsevier, 2009.

[10] Islene Calciolari Garcia. Listas de exercícios, 2007.

[11] Robert Sedgewick. *Algorithms in C*. Addison-Wesley, 1990.

[12] Robert Sedgewick. *Algorithms in C*. Addison-Wesley, 1998.

[13] Jayme L. Szwarcfiter and Lilian Markenzon. *Estrutura de dados e seus algoritmos*. LTC, 2 edition, 1994.

[14] Aaron A. Tenenbaum. *Estruturas de dados usando C*. Makron Books, 1995.