Coletânea de exercícios

3 de outubro de 2019

1 Apontadores

1 Qual a relação entre variável e apontador?

2 [**?**] No trecho de código abaixo, qual o valor final da variável `i`?

```
int main() {
  int i = 5;
  int *p;
  p = &i;
  *p = 13;
  return 0;
}
```

3 [**?**] No código abaixo, qual será o valor da variável `b` após a execução da função `DefineValor`?

```
void DefineValor(int c, int *e){
  c = 7;
  *e = 11;
}

int main(){
  int b = 1;
  DefineValor(b, &b);
  return 0;
}
```

4 Escreva uma função em C que (i) recebe o tamanho n de um vetor (por valor), um vetor V (por valor) e duas variáveis inteiras min e max (por referência), (ii) encontra os valores mínimo e máximo em V e (iii) retorna esses valores em min e max.

5 Escreva uma função em C que (i) recebe o tamanho n de um vetor (por valor), um vetor V (por valor) e dois apontadores para inteiro $min*$ e $max*$ (por referência), (ii) encontra os valores mínimo e máximo em V e (iii) retorna os endereços desses valores em min e max.

6 [**?**] Explique porque a função abaixo pode levar a erros durante a execução de um programa.

```
int* f() {
  int a;
  return &a;
}
```

7 [**?**, 12.1] Suponha que as seguintes declarações foram realizadas:

```
int v[] = {5, 15, 34, 54, 14, 2, 52, 72};\\
int *p = \&v[1], *q = \&v[5];
```

a. Qual o valor de `*(p+3)`?
b. Qual o valor de `*(q-3)`?
c. Qual o valor de `*(q - p)`?
d. A expressão `p < q` tem valor verdadeiro ou falso?
e. A expressão `*p < *q` tem valor verdadeiro ou falso?

8 [**?**, 12.2] Qual o conteúdo do vetor `v` após a execução do seguinte trecho de código?

```
#define N 10

int v[N] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
int *p = &v[0], *q = &v[N-1], temp;

while (p < q) {
  temp = *p;
  *p++ = *q;
  *q-- = temp;
}
```

2 Alocação dinâmica de memória

9 [**?**] Quais os possíveis resultados da execução da linha de código abaixo?

```
int *a = malloc(sizeof(int)*10);
```

10 [**?**] Escreva uma função que aloca um inteiro e que ou retorna um apontador para ele se a alocação for bem-sucedida ou imprime uma mensagem de erro e retorna NULL se a alocação for mal-sucedida.

11 [**?**] Considere a função abaixo que aloca um inteiro. Explique o que significa a expressão `int** i` na definição de `f`. Mostre como ela deveria ser chamada.

```
int f(int** i) {

  *i = malloc(sizeof(int));
  if (*i)
    return 1;
  else
    return 0;
}
```

12 Escreva uma função em C para alocar um cubo de dimensões $n \times m \times r$ dinamicamente e retornar um apontador para ele. Se não houver memória suficiente, a função deve retornar NULL.

13 Escreva uma função em C para receber um apontador para um cubo alocado dinamicamente e suas dimensões n, m, r e liberar a memória ocupada por ele.

14 Considere o código abaixo. Embora o efeito da 3ª e 4ª linhas seja similar, quais as diferenças entre `A` e `B`?

```
int n;
scanf("%d",&n);
int A[n];
int* B = malloc(sizeof(int)*n);
```

15 [**?**] Escreva uma função em C que recebe uma cadeia s de caracteres e produz uma ou mais cadeias que são o resultado de dividir s nas ocorrências de um caractere c. Por exemplo, se $s = acf;;fgs;get;$ e $c = ;$ então a divisão resulta nas cadeias *acf*, cadeia-vazia, *fgs*, *get* e cadeia-vazia. Se $s = acfget$ então a divisão resulta em *acfget*. Observe que a cadeia de entrada pode ser arbitrariamente longa e que o número de ocorrências de c também é arbitrário.

A função deve alocar e retornar um vetor com as cadeias obtidas. Se não houver memória suficiente, a função deve retornar NULL. O espaço para cada cadeia também deve ser alocado pela função. O espaço alocado não deve ser maior que o necessário.

16 [**?**, 14.4] A função abaixo supostamente cria uma cópia de uma cadeia de caracteres. O que há de errado com a função?

```
char* duplica(const char* p){
  char* q;
  strcpy(q,p);
  return q;
}
```

17 Reescreva a função `duplica` para que ela funcione.

18 [**?**] Considere as duas formas (a) e (b) abaixopara definir uma matriz de inteiros de tamanho $i \times j$. Supondo os comandos sejam bem sucedidos e que tanto um inteiro quanto um apontador ocupem 32 bits, quantos bytes `MA` ocupa e quantos bytes `MB` ocupa?

(a)

```
int MA[i][j];
```

(b)

```
int** MB = malloc(i*sizeof(int*));
for (k=0; k<i; k++)
  MB[k] = malloc(j*sizeof(int));
```

3 Bit arrays

19 [**?**] 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.

20 [**?**] 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))
```

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

4 Listas

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

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

24 [**?**] 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.

25 [**?**] 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");
}
```

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

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

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

29 [**?**, 4.6.3] Escreva uma função para remover de uma lista encadeada todos os elementos que contêm x.

30 [**?**, 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.

31 [**?**, 4.6.4] Escreva uma função que remova de uma lista encadeada um nó cujo conteúdo tem valor mínimo.

32 [**?**, 4.7.3] Escreva uma função que faz uma cópia de uma lista dada.

33 [**?**, 3.10] Escreva uma função para verificar se duas listas encadeadas simples têm o mesmo conteúdo.

34 [**?**, 4.7.2] Escreva uma função que copia uma lista encadeada em um vetor.

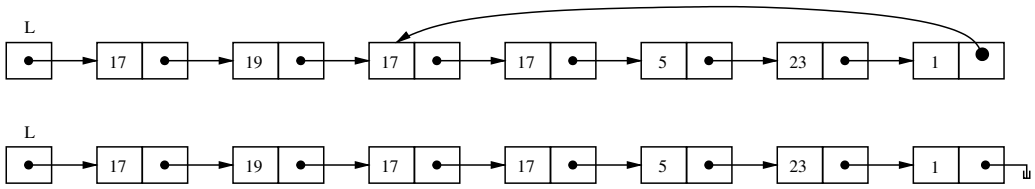
35 [**?**, 3.6] Remova da lista L_1 os nós cujas posições devem ser encontradas na lista ordenada L_2. Por exemplo, se $L_1 = (A, B, C, D, E)$ e $L_2 = (2, 4, 8)$, então o segundo e o quarto nós devem ser removidos da lista L_1 (o oitavo nó não existe) e, depois da remoção, $L_1 = (A, C, E)$.

36 [**?**] 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.



37 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.

5 Listas duplamente encadeadas

38 [**?**] 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.

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

6 Listas circulares

40 [**?**, 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.

41 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;
```

7 Filas e Pilhas

42 [**?**, 9, p. 86] Qual a diferença entre uma pilha e uma fila?

7.1 Filas

43 [**?**, 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 * * *

44 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.

45 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.

46 [**?**, 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.

7.2 Pilhas

47 [**?**, 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 * * *

48 [**?**, 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.

49 [**?**, 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)
```

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

51 [**?**, 11, p. 63] Escreva um algoritmo que converta uma expressão em notação pré-fixa para pós-fixa.

7.3 Pilhas e Filas

52 [**?**, 3.3, p2] Podemos aproveitar uma implementação para fila simétrica para implementar uma fila ou uma pilha. Mostre como isso poderia ser feito.

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

54 Mostre como uma fila pode ser implementada usando pilhas.

55 [**?**, 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

8 Recursão

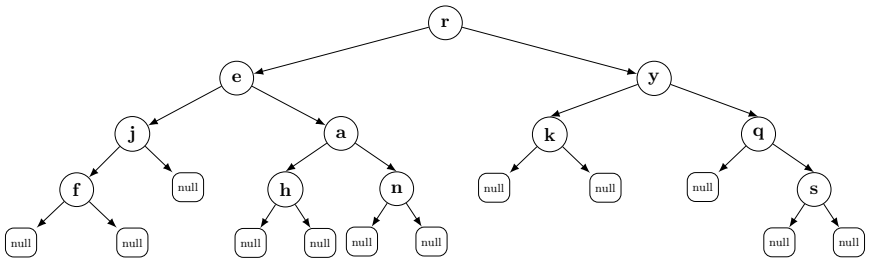
56 [**?**, 1.1] Responda se é certo ou errado: Todo procedimento recursivo deve incorporar terminações sem chamadas recursivas, caso contrário, ele seria executado um número infinito de vezes.

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

9 Árvores

9.1 Representação

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



59 [**?**] 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?

60 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?

61 [**?**] 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.

62 [**?**] Escreva uma função para percorrer uma árvore representada implicitamente em pré-ordem.

63 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.

64 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.

9.2 Diversos

65 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.

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

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

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

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

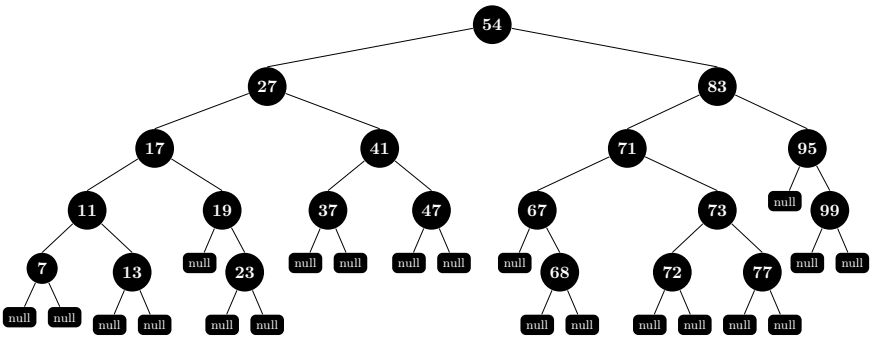
70 Escreva uma função recursiva para verificar se uma árvore binária é completa.

71 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.

72 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.

9.3 Percursos

73 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.



74 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?

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

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

10 Buscas

77 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?

11 Árvores binárias de busca

78 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`.

79 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.

80 [**?**, 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.

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

82 [**?**, 12.2-2] Write recursive versions of TREE-MINIMUM and TREE-MAXIMUM.

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

84 [**?**] 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.

85 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.

86 [**?**] Escreva uma função para inserir em uma árvore binária de busca com representação costurada.

12 Hashing

87 [**?**] 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.

88 [**?**] Para cada item da questão anterior, ilustre a remoção das chaves 31 e 77.

89 [**?**] 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, $h(x, y) \neq h(y, x)$?

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

91 [**?**] Suponha que você tem uma tabela de hashing com função construída pelo método da divisão e de tamanho 101 que está 85% cheia. Para qual tamanho essa tabela deveria ser redimensionada para ficar aproximadamente 30% cheia?

92 [**?**] 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?

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

13 Filas de prioridades

94 [**?**] 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

95 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.

96 [**?**, 10.1.1] Mostre que todo vetor decrescente é um heap de máximo. Mostre que a recíproca não é verdadeira.

97 [**?**, 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;
}
```

98 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.

99 [**?**, 10.1.3] Escreva uma função que decide se um vetor v[1..m] é ou não um heap de máximo.

100 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 Ordenação

101 Quais dos algoritmos vistos em sala são estáveis?

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

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

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

105 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?

106 [**?**] 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?

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

108 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.

109 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

110 [**?**] 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++;
  }
}
```

111 [**?**] 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;
    }
  }

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

112 Escreva uma versão iterativa do Mergesort.

113 Escreva uma versão iterativa do Quicksort.