MC202: Estruturas de Dados

Professores Cid C. de Souza e Hélio Pedrini
Instituto de Computação - UNICAMP – 2º semestre de 2009
Turmas A, B, C e D – 2ª Prova (10/12/2009)

Questão	Valor	Nota
1	2,0	
2	2,0	
3	2,0	
4	2,0	
5	2,0	
Total	10,0	

Nome:

RA: **Turma:**

Instruções:

1. A duração da prova é de 110 minutos.
2. Não é permitida consulta a qualquer material.
3. Somente serão consideradas respostas nos espaços indicados. Use os versos das folhas como rascunho.
4. Celulares, *pagers*, calculadoras e demais dispositivos eletrônicos de comunicação devem permanecer desligados durante toda a prova.
5. **Nenhum aluno será autorizado a deixar a sala de prova antes que tenham decorridos 60 minutos desde o seu início**.
6. **Todo aluno que deixar a sala de prova deverá entregá-la *em definitivo***.
7. A prova pode ser feita a lápis porém, neste caso, eventuais pedidos de revisão de nota poderão ser negados a critério do docente.
8. Todos os códigos devem ser escritos em linguagem `C`.
9. Nas questões que solicitam que seja completada uma função cujo esboço já é fornecido, cada retângulo deverá conter **uma <u>única</u> expressão**, ou então, **um <u>único</u> comando simples** em linguagem `C`, conforme o contexto.
10. Todos os **grafos** mencionados nos enunciados são considerados *simples* (i.e., sem arestas múltiplas) e sem auto-laços.
11. O termo **ordenação** usado nos enunciados refere-se à ordem *monotonamente crescente*.

1. Suponha que se deseja ampliar as funcionalidades da biblioteca para manipulação de *heaps* que foi vista em aula de modo a incluir o procedimento `Remove_Chave_Qualquer(Heap *h,int i)`, que remove do *heap* h o elemento na posição i. Considere que o procedimento só é chamado para valores válidos de i, i.e., entre 0 e tamanho do *heap*-1. Considere ainda que a chave de menor valor se encontra na primeira posição do *heap*.
A solução apresentada abaixo foi proposta por Mané mas, infelizmente, esse procedimento **não** está 100% correto. A implementação feita pelo Mané baseia-se na mesma estrutura de dados usada em aula para definir um *heap*, a qual também é mostrada abaixo. Note que, a título de exemplo, está sendo suposto aqui que as chaves armazenadas no *heap* são números inteiros.

```
#define TAM_MAX 50

typedef struct{
 int vetor[TAM_MAX];
 int tam;
} Heap;
```

```
void Remove_Chave_Qualquer(Heap *h,int i) {
  /* troca o último elemento com aquele da posição i  */
  int aux=(*h).vetor[i];
  (*h).vetor[i]=(*h).vetor[(*h).tam-1];
  (*h).vetor[(*h).tam-1]=aux;
  ((*h).tam)--; /* decrementa o tamanho */
  Desce(h,i); /* desce chave na posição i: visto em aula */
  return;
}
```

O seu objetivo nesta questão é corrigir e explicar ao Mané qual foi o erro cometido por ele. Para tanto, terá que corrigir a função de acordo com os seguintes passos:

(a) Identifique o erro nessa função, apresentando no quadro abaixo um exemplo de remoção de uma chave em uma posição qualquer de um *heap* para o qual ele não funciona.

(b) Reescreva no espaço indicado abaixo o código `C` da função, corrigindo o problema encontrado no item (a). Se achar conveniente, use as funções vistas em aula, cujos protótipos são dados a seguir:

```
void Sobe(Heap *h, int m); /* sobe chave na posição m */
void Desce(Heap *h, int m); /* desce chave na posição m */
void RemoveHeap(Heap *h, int *x); /* retorna e remove o primeiro elemento do heap */
```

(c) Qual a complexidade do algoritmo corrigido que você apresentou no item (b) em função de n (definido como sendo o tamanho do *heap*) ?

2. Para cada tipo de tratamento de colisão indicado abaixo, mostre a tabela de *hash* obtida quando inserimos as chaves `457, 590, 786, 677, 340, 812, 734, 442, 509, 655, 398` e `528` , nessa ordem. Indique ainda o número de colisões ocorridas em cada caso. Considere que a tabela encontra-se inicialmente **vazia** e que a função de *hashing* usada é definida como sendo $f(x) = x \mod 13$, ou seja, o resto da divisão da chave por 13.

(a) encadeamento simples (ou separado): número de colisões = ____

0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

(b) endereçamento aberto com **reespalhamento quadrático**: número de colisões = ____

	chave
0	
1	
2	
3	
4	
5	
6	
7	
8	
9	
10	
11	
12	

(c) encadeamento combinado: número de colisões = ____

Nota: considere que a busca por posições vazias na tabela deve ser feita do final para o início da tabela e que o campo `prox` é inicializado com o valor -1, da mesma forma como foi feito em aula.

	chave	prox
0		
1		
2		
3		
4		
5		
6		
7		
8		
9		
10		
11		
12		

Para este caso de encadeamento combinado, escreva nos quadros indicados abaixo duas chaves dentre aquelas inseridas na tabela que são espalhadas em posições distintas pela função de *hashing* mas que se encontram em uma **mesma** lista:

____ e ____ .

3. Esta questão discute variantes do `BubbleSort` usadas para ordenar vetores definidos como abaixo.

```
#define NMAX 100
typedef int Vetor[NMAX];
```

(a) No espaço indicado abaixo, preencha o código da implementação **recursiva** do `BubbleSort` que recebe como parâmetros o vetor (u) e o seu tamanho (n). Use as duas colunas do espaço se precisar.

```
void BubbleRec(Vetor u, int n){
```


(b) Uma implementação conhecida do `BubbleSort` tem o nome de `CocktailSort`. Nesta versão do algoritmo, dado um vetor de tamanho n, são executadas $\lfloor \frac{n}{2} \rfloor$ iterações. A i-ésima iteração é composta de dois *passos*. No primeiro passo, usa-se o *método da bolha* para colocar o i-ésimo **máximo** na sua posição final (ordenada). No segundo passo, o *método da bolha* é usado para colocar o i-ésimo **mínimo** na sua posição final (ordenada). Neste contexto, consideramos que as iterações são contadas a partir de *zero* e, assim, o maior/menor valor do vetor original é definido como sendo o seu 0-ésimo máximo/mínimo.

Um exemplo de aplicação do `CocktailSort` a um vetor de 5 elementos é ilustrado abaixo. A seguir mostra-se uma implementação parcial do `CocktailSort`. Preencha os espaços indicados no código de modo a torná-lo correto. Se achar conveniente, você pode usar a rotina `Troca` mostrada abaixo.

iteração	passo	vetor				
(vetor original)	—	4	5	8	2	1
0	1	4	5	2	1	8
0	2	1	4	5	2	8
1	1	1	4	2	5	8
1	2	1	2	4	5	8

```
void Troca(int *a, int *b){
  int c=*a;
  *a=*b; *b=c;
  return;
}
```

```
void CocktailSort(Vetor u, int n) {
    int int i, j= [          ] , esq=0, dir=n-1, k;
    for (i=0; i<=j; i++) { /* faz as iterações buscando o max e o min no subvetor u[esq,...,dir] */
        for (k=esq; [          ] ; k++) /* faz o primeiro passo */
            if ( [          ] ) [          ] ;
            [          ] ;
        for (k=dir; [          ] ; [          ] ) /* faz o segundo passo */
            if ( [          ] ) [          ] ;
            [          ] ;
    } /* final das iterações */
    return;
} /* CocktailSort */
```

4. Considere o texto abaixo. Suponha que cada letra represente um registro em um arquivo armazenado em **memória secundária** (externa). Deseja-se aplicar o método de **intercalação balanceada de** 3 **caminhos** ($f = 3$) para ordenar este arquivo e, para isso, dispõe-se de **6** $(= 2f)$ **fitas**. Sabendo que a memória principal só comporta 3 registros de cada vez ($m = 3$), preencha os quadros abaixo indicando o conteúdo das fitas ao final de cada etapa do processo de intercalação. Lembre-se de identificar **claramente** as separações entre os diferentes blocos nas fitas. Note também que, ao final de cada etapa, só está sendo solicitado que você indique o conteúdo das fitas cujos dados são relevantes naquele momento da execução. Ou seja, preencha o quadro exatamente como o exemplo dado em aula.

U M F E L I Z N A T A L E U M P R O S P E R O A N O N O V O

Arquivo inicial com 30 registros ($n = 30$)

(carga inicial das fitas)

fita 1	
fita 2	
fita 3	

(primeira etapa de intercalação)

fita 4	
fita 5	
fita 6	

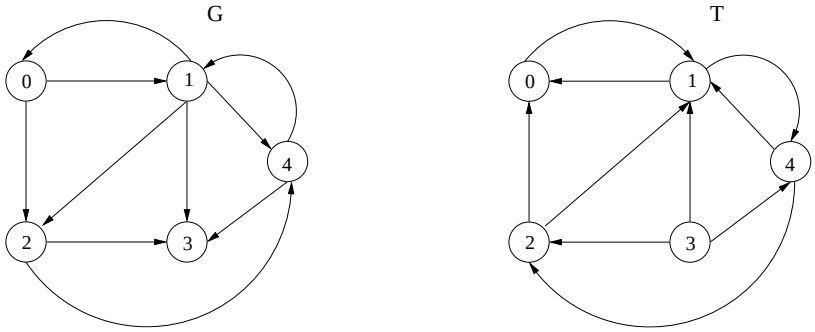
(segunda etapa de intercalação)

fita 1	
fita 2	
fita 3	

(terceira etapa de intercalação)

fita 4	
fita 5	
fita 6	

5. Dado um grafo **direcionado** G, o **grafo transposto** de G é definido como sendo o grafo T obtido de G ao inverter-se a direção de todos os seus arcos, conforme mostrado no exemplo abaixo.



Suponha que o tipo `Grafo` tenha sido definido em uma biblioteca em linguagem `C` conforme mostrado abaixo. Note que os vértices são identificados pelos seus números e, portanto, pela definição de n, vão de 0 (zero) a $(n-1)$.

```
/* definicao das listas de adjacencias */
typedef struct NoAux{
  int vertice;
  struct NoAux *prox;
} ListaAdj, *ApListaAdj;

typedef ApListaAdj *TipoAdj;
```

```
/* define o tipo grafo */
typedef struct{
  int n;       /* n=número de vértices */
  int m;       /* m=número de arcos */
  TipoAdj Adj; /* lista de adjacências */
} Grafo;
```

No quadro abaixo, escreva um código em `C` da função `GrafoTransposto` que constrói o grafo transposto T de um grafo G passado na entrada. Note que o cabeçalho da função já é dado.

```
void GrafoTransposto(Grafo *T, Grafo G) {
```