PUC-Rio
Departamento de Informática
Período: 2015.1
Horário: 2as-feiras e 4as-feiras de 17-19
30 de março de 2015

ESTRUTURAS DE DADOS AVANÇADAS (INF 1010)

1^a Lista de Exercícios

1. **Lista**

   (a) Seja um TAD definido por uma lista circular implementada em um vetor.

      i. Dado que o vetor possui max posições, proponha uma função que determine o número n de elementos a partir do conteúdo dos ponteiros de início H e final T da lista.

      ii. Proponha um algoritmo para determinar o k-ésimo elemento da lista. Analise sua complexidade.

      iii. Proponha um algoritmo para remover o k-ésimo elemento da lista. Qual o número total de operações? E a sua complexidade?

      iv. Proponha um algoritmo para inserir um elemento x logo após o k-ésimo elemento da lista. Analise sua complexidade e o número total de operações.

2. **Hash**

   (a) Quais são as propriedades de uma boa função de dispersão?

   (b) Escreva uma função de dispersão (*hash*) que tenha como chave um *string* com os nomes de paises, de até 32 caracteres. A tabela *hash* deve ter 513 elementos.

   (c) No encadeamento interior, em que as chaves são armazenadas na própria tabela de *hash,* de um exemplo em que duas chaves distintas, x e y, com $h(x)?$ $h(y)$ podem apresentar colisões. Prepare uma explicação para um colega em que você exemplifique com uma tabela *hash* de 11 elementos. Invente uma função de *hash* para tornar sua explicação mais concreta.

   (d) Explique com base no seu exemplo anterior porque ao remover uma chave de uma tabela *hash* você não pode simplesmente apagar a entrada da tabela.

   (e) Mostre, através de um desenho, como ficaria a tabela *hash* de 7 elementos que recebesse as seguintes chaves de busca 7,10,15,14,17,16 (nesta ordem).

   (f) Se ela fosse com encadeamento exterior com a função de dispersão:

   (g) Se ela fosse com encadeamento aberto e com segunda função de dispersão:

(h) A função **insere** no quadro abaixo implementa a inserção de chaves inteiras positivas em uma tabela de dispersão com desempate interno de colisões:

```
#define MAX 8
#define VAZIO (-1)
int hash[MAX];

int insere (int x)
{
int pos, k;

pos = x;
for (k = 0; k < MAX; k++)
{
pos = (pos + k) % MAX;

if (hash[pos] == x) /* Chave duplicada – não insere
*/
return (pos);

if (hash[pos] == VAZIO) /* Posição livre */
{
hash[pos] = x;
return (pos);
}
}
return (-1);
}
```

(i) Supondo a tabela inicialmente preenchida com o valor VAZIO em todas as posições, responda:

(j) Insira as chaves 4, 12, 20, 28, 36, 44, 52 e 60 na tabela.

(k) Qual a função de dispersão H(x, k) implicitamente utilizada no algoritmo?

(l) Trata-se de uma função linear ou quadrática em k?

(m) Em que situações a função retornará um valor menor que 0? Por quê?

(n) Para quais valores de MAX é possível assegurar a varredura integral da tabela?

(o) Quantas colisões existem ao se inserir dados dos n primeiros números naturais em uma tabela de dimensão m, usando as seguintes funções hash:

i. n mod 5
ii. n mod 7
iii. n mod 35
iv. n mod (m/5)
v. n mod m

(p) Em uma tabela hash de tamanho m=12, assumindo uma função de hash h'(x) = x mod m:

(q) Insira as seguintes chaves: 4, 10, 24, 36, 25, 14, 23, 100, 2, 3, 12, 13, Caso haja colisão utilize a técnica de armazenamento interno, utilizando para isso uma função de incremento linear, com passo 2.

(r) Escreva a expressão matemática correspondente.

(s) Esta é uma boa função de Hash? Justifique detalhadamente a sua resposta.

(t) Considere um TAD de hash com a interface:

```
typedef struct hash Hash;
Hash* hsh_cria(int n, int hash_func(char*), void free_func(void*));
int hsh_insere(Hash* tabela, void* info, char* chave);
int hsh_remove(Hash* tabela, char* chave);
void* hsh_busca(Hash* tabela, void* chave);
Hash* hsh_libera(Hash* tabela);
void hsh_percorre(Hash* tabela, void callback_func(void*));
```

Considere ainda que, depois de ler uma lista de dados de países, ele tenha armazenado numa tabela hash, tabela_pais, os ponteiros que contem os endereços da estrutura Pais de cada um dos países lidos.

```
typedef struct pais Pais;
struct pais {
char nome[81];
char continente[81];
int exercitos;
char dono[81];
};
```

i. Escreva, como um módulo cliente do TAD que conhece a estrutura Pais, uma função que troque o nome do dono de um país e tenha o seguite protótipo:
int dono_do_pais(char* novo_dono, Hash* tabela_pais, char* nome_do_pais);
o valor de retorno é 1 se teve sucesso e 0, caso contrario. Pode utilizar as funções das biliotecas padrão do C, tipo stdio.h string.h, etc...
Solução

```
int dono_do_pais(char* novo_dono, Hash* tabela_pais, char* nome_do_pais){
Pais* p = (Pais*) hsh_busca(tabela_pais,nome_do_pais);
if (p==NULL) return 0;
strcpy(p->dono, novo_dono);
return 1;
}
```

ii. Escreva uma função de *hash* que possa ser utilizada para armazenar no TAD descrito na Questão 2 os nomes dos países. Considere o protótipo da interface do TAD acima e uma tabela *hash* com 307 elementos.
Solução

```
int hash_func(char* chave) {
int i, k=0;
for (i=0; chave[i]!='\0'; i++)
k+=chave[i];
return k%307;
}
```

(u) Seja $S = \{s_1, s_2, \ldots, s_m\}$ um conjunto muito grande. Assuma que S está particionado em k blocos. Está disponiível um procedimento chamado *which-block* onde $which - block(s_i)$ retorna o número j do bloco onde se encontra o elemento s_i (S pode conter endereços, sendo o CEP o número do bloco). Um subconjunto pequeno T de S é mantido. Implemente as seguintes operações sobre T:

i. $insere(s_i)$
ii. $remove(s_i)$
iii. $remove - bloco(j)$

A operação $remove - bloco(j)$ não precisa remover fisicamente o bloco, basta desconectá-lo da estrutura. Considere que T deve estar inicialmente vazio. Lembre que n e k são muito grandes e é não é possível armazenar uma tabela de tamanho m ou k. Idealmente todas estas operações deveriam ter complexidade $O(log\ k)$.

3. **HEAP**

(a) Seja uma *min-heap* H. Defina o TAD da heap H e escreva um algoritmo que imprima todos os elementos em H cujas chaves são menores que um valor x dado.

(b) Seja uma *min-heap* H. Dado um valor x e um inteiro k, escreva um algoritmo que responda SIM se x é um dos k menores elementos em H, e não caso contrário. Mostre que o número de inspeções (verificação de um nó de H) é no máximo $2 \times (k-1)$.

(c) No vetor mostrado abaixo, mostre os passos do algoritmo O(n) para construção de um *heap* de **mínimo** (*min Heap*). Quantas trocas você precisou fazer?

$$95, 60, 78, 39, 28, 66, 70, 33$$

Solução

Oito elementos: 8/2=4 estão em ordem pois não tem filhos.

--, --, --, --, 28, 66, 70, 33

Insere o 39 e troca com o 33.

__, __, __, 39, 28, 66, 70, 33
__, __, __, 33, 28, 66, 70, 39

Insere o 78 e troca com o 66 (menor filho)

__, __, 78, 33, 28, 66, 70, 39
__, __, 66, 33, 28, 78, 70, 39
Insere o 60 e troca com o 28 (menor filho)

__, 60, 66, 33, 28, 66, 78, 39
__, 28, 66, 33, 60, 78, 70, 39
Insere o 95 e troca com o 28.
95, 28, 66, 33, 60, 78, 70, 39
28, 95, 66, 33, 60, 78, 70, 39

Troca o 95 pelo 33

28, 33, 66, 95, 60, 78, 70, 39
Troca o 95 pelo 39
28, 33, 66, 39, 60, 78, 70, 95

Resposta:
28, 33, 66, 39, 60, 78, 70, 95 trocas = 6

i. Transforme o vetor 30,15,28,60,45,90,10,23 num *heap* de máximo fazendo o mínimo de trocas possível (mostre cada uma das trocas).

ii. Supondo que você crie um *heap* vazio mostre como ele vai sendo construído caso receba os elementos do vetor da questão anterior, um de cada vez. O *heap* final é o mesmo?

iii. Implemente uma função que verifica se a ordem dos elementos de um vetor de ponteiro para estruturas com dados de alunos representa uma fila de prioridade (*heap*), onde a raiz armazena o aluno com a maior nota. A função recebe como parâmetros o número de elementos no vetor e o vetor de ponteiro para o tipo que representa o aluno. A função deve retornar 1 se a ordem dos elementos representa um *heap*; caso contrário, deve retornar zero.

```
struct aluno {
char nome[64];
float nota;
};
typedef struct aluno Aluno;
int heap_max (int n, Aluno** v);
```

(d) Para as questões a seguir, considere o heap com a seguinte estrutura:

```
typedef struct _heap Heap;
struct _heap {
int max; /* tamanho maximo do heap */
int pos; /* proxima posicao disponivel no vetor */
int* info; /* vetor dos elementos do heap */
};
```

i. Escreva uma função em C que retorne o número de elementos maiores que um elemento de valor x de um *heap max*, sendo *heap* e x enviados como parâmetros da função. A função deve examinar o menor número possível de elementos.

ii. Escreva uma função em C que receba como parâmetros um *heap max*, a posição de um elemento do *heap* (no vetor que o representa) e um valor inteiro, e altere a prioridade do elemento para o novo valor, reposicionando-o, caso necessário. Assuma que já foi definida a função

```
void troca (int pai, int pos, int *info).
```

4. **Vetor de Bits**

   Considere a representação de conjuntos por vetores de bits:

   struct set {

   int n; /* número de bytes */

   unsigned char* v; /* vetor */

   };

   typedef struct set Set;

   Implemente uma função que recebe como parâmetros dois conjuntos (A e B) e verifica se o primeiro contém o segundo (A⊃B), isto é, se todos os elementos de B pertencem a A. A função deve retornar 1 se A contém B; caso contrário, deve retornar 0. Note que os conjuntos podem ter dimensões diferentes.

   int contem (Set* A, Set* B);

   Dicas para implementação

   (a) Use a seguinte propriedade: (A⊃B) se e somente se (A∩B) = B

   (b) Não assuma que os vetores representando A e B possuem o mesmo comprimento. Suponha que o vetor representando B possua n elementos e que o vetor representando A possua m elementos; se n>m então todos os elementos B[i], com i>m, devem ser 0.

   (IMPLEMENTAÇÃO NÃO TESTADA)

   BitVector* contem(BitVector* a, BitVector* b)

   {

   int i;

   if (a->max < b->max)

   for (i=a->max; i<b->max; i++)

   if (b->vector[i] != 0) return 0;

   for (i=0; i<b->max; i++)

   if ((a->vector[i] & b->vector[i]) != b->vector[i]) return 0;

   return 1;

   }

5. **P**artição

   (a) Partição dinâmica. Considere o conjunto $\{1, 2, 3, 4, 5, 6\}$. Como ficaria a representação por vetor deste conjunto após a operação de criar-particao-dinamica? Como ficaria este vetor após as operações: união(1,3), união(2,3), união(2,5), união(3,4), união(1,6) se: (a) a operação de união for feita sem critério de tamanho, e (b) com critérios de tamanho.

(b) Considerando a partição (a) do problema anterior, como ficaria a partição depois de uma busca(5) seguido de uma busca(3) com a estratégia de compressão de caminho?

6. **Árvore**

(a) Seja uma arvore binária de busca (ABB), não necessariamente balanceada. Defina o TAD e apresente um algoritmo que imprima todos os elementos na ABB na ordem crescentes das suas chaves na árvore. O seu algoritmo deve executar em $O(n)$, onde n é o número de elementos na árvore.

(b) Seja uma ABB. O desbalanceamento de um nó de uma ABB é o módulo da diferença das alturas em as (sub)árvores que têm como raiz os seus filhos esquerdo e direito. Defina o TAD e apresente um algoritmo que calcule o desbalanceamento de todos os nós da ABB. O seu algoritmo deve executar em $O(n)$, onde n é o número de elementos na árvore.

7. (3.0) Considere um TAD de hash com a interface:

```
typedef struct pais Pais;
struct pais {
char nome[81];
char continente[81];
};
int hsh_insere(Hash* tabela, void* info, char* chave);
int hsh_remove(Hash* tabela, char* chave);
void* hsh_busca(Hash* tabela, void* chave);
```

Considere a seguinte lista de países, respectivos continentes:

Lista de pares (país, continente): (India, Asia), (Egito, Africa), (Iran, Asia), (Suecia, Europa), (Peru, America do Sul)

A tabela hash com m(=21) posições, **tabela_pais**, contém ponteiros com os endereços da estrutura Pais de cada um dos países. O endereço na tabela hash é calculado pela função abaixo:

```
int hash_func(char* chave) {
int i, k=0;
for (i=0; chave[i]!='\0'; i++)
k+=chave[i];
return k%21;
}
```

(a) (1.0) Escreva o procedimento *hsh_insere* para armazenar os países, com seus respectivos continentes, na **tabela_pais** descrita acima. Encadeamento interno com passo unitário, ou seja, em caso de colisão tenta-se inserir na primeira posição seguinte livre.

(b) (1.0) Utilizando a função proposta no item anterior, insira a lista de países dada no enunciado.

(c) (1.0) Escreva o procedimento *hsh_remove*. Liste todos os casos que devem ser considerados. Lembre a forma como a colisão é tratada no procedimento *hsh_insere*.

8. (3.0) Seja uma **min 2-heap** implementada utilizando ponteiros (e não um vetor como visto em aula).

   (a) (1.0) Defina um tipo abstrato de dados (struct em C) para ser utilizado na implementação dessa heap.

   (b) (1.0) Escreva o procedimento *a*juste_acima que testa se um elemento é maior que o seu pai e, caso contrário, faz a troca com o pai e itera até que o invariante da heap (pai menor que filhos) seja restabelecido.

   (c) (1.0) Escreva o procedimento *a*juste_abaixo que testa se um elemento é menor que seus filhos, e itera até que o invariante da heap (pai menor que filhos) seja restabelecido.

9. Seja uma arvore binária de busca (ABB), não necessariamente balanceada. Utilizando o TAD abaixo.

   struct elem {

   int chave;

   struct elem* esq;

   struct elem* dir;

   };

   typedef struct elem Elem;

   Elem * Raiz;

   (a) Apresente um algoritmo que imprima todos os elementos na ABB apontada por **Raiz** na ordem crescente das suas chaves na árvore. O seu algoritmo deve executar em $O(n)$, onde n é o número de elementos na árvore.

   (b) Proponha um algoritmo para encontrar a mediana dos elementos na ABB apontada por **Raiz** na ordem crescente das suas chaves na árvore. O seu algoritmo deve executar em $O(n)$, onde n é o número de elementos na árvore. Que informação poderia estar presente no TAD **Elem** que permitiria propor um algoritmo mais eficiente ? Qual seria a complexidade desse algoritmo ?

10. Considere uma grande festa (**Rock ' n Rio**)em que os convidados dançam em pares (incrível, mas isto ainda é comum!). Um grupo observador que deseja entender como os milhares de convidados se dividem, resolveu acessar ao longo da festa os conjuntos de pessoas que dançaram entre si. Isto é, se A dançou com B e A dançou com C, então A, B e C pertencem ao mesmo conjunto. Os acessos aos conjuntos a que pertence cada convidado precisa ser muito eficiente.

(a) Proponha uma estrutura de dados e apresente um algoritmo para que, quando dois convidades estiverem juntos dançando, se possa verificar se ele pertencem a um mesmo conjunto.

(b) Que operação deve ser feita para a atualização dos conjuntos ao se identificar que dois convidados estão dançando juntos? Explique como deve ser feita esta operação para que o acesso seja eficiente.