Aluno(a):\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ matrícula:\_\_\_\_\_\_\_\_\_\_\_\_

|  |  |  |
| --- | --- | --- |
| 1a) | 2.0 |  |
| 2a) | 2.0 |  |
| 3a) | 3.0 |  |
| 4a) | 3.0 |  |
|  |  |  |

1. A prova é individual e sem consulta.
2. A interpretação faz parte da questão.
3. O tempo de prova é 1:30 h.
4. As respostas devem seguir as questões.
5. Caso precise de rascunho use o verso da folha.
6. Caso parte da resposta esteja no verso, indique claramente este fato.
7. (2.0 pontos) Escreva uma função de dispersão (função *hash*) que receba como argumentos:
   1. uma *string* com a chave a ser indexada;
   2. um inteiro para tratar as colisões, que começa com 0 e avança unitariamente (0, 1, 2, ...)
   3. um inteiro que define o tamanho da tabela *hash*

Esta função deverá retornar um inteiro que identifica a localização da chave na tabela *hash*. Para calcular esse valor utilize todos os caracteres da *string* e que não use apenas a sua soma simples. Quando houver colisão será usada a técnica de endereçamento aberto, assim essa função será reinvocada com o inteiro de colisão incrementado de um a cada colisão. Para tratar a colisão use tentativa linear ou quadrática ou use uma dispersão dupla.

Resp.:

int hash(char \*chave, int colisao, int tam) {

int hash=0;

char \*letra;

for( letra=chave ; \*letra!='\0' ; letra++ ) {

hash += ((int)(\*letra)) \* (67\*(letra-chave+1));

}

hash = hash + (17\*colisao) + (23\*colisao\*colisao);

return hash%tam;

}

1. (2.0 pontos) Os algoritmos abaixo são usados para resolver problemas de tamanho n. Descreva e informe para cada algoritmo sua complexidade no pior caso (O maiúsculo/Ômicron). Tente entender o problema antes de apresentar uma resposta.

a)

for ( i=1; i < n; i \*= 2 ) {

for ( j = n; j > 0; j /= 2 ) {

for ( k = j; k < n; k += 2 ) {

sum += (-j \* k) << i/2;

}

}

}

Resp.:

O tempo de execução do laço mais externo é log2, visto que i dobra a cada passagem. O tempo de execução do laço do meio é semelhante, exceto que j é dividido pela metade, começando em n. O tempo de execução do último laço é n/2, ou seja, O(n). Como os laços estão aninhados, devem-se multiplicar todos eles. Logo, a complexidade desse algoritmo é:

O(n(log n)2)

b)

for( int i = 1; i <= n; i \*= 2 ) {

for( int j = 0; j < i; j++ ) {

for( int k = 0; k < n; k += 2 ) {

sum \*= k;

}

for( int k = 1; k < n; k \*= 2 ) {

sum /= k

}

}

}

Resp.:

Os dois laços mais internos são respectivamente O(n) e O(log n); como estão em paralelo, a complexidade deve ser somada, o que resulta em O(n). O laço mais externo tem complexidade O(log n) visto que os passos dobram a cada passagem. Já o laço do meio é sequencial, ou seja, O(n). Assim a complexidade combinada dos laços é:

O(n2 log n)

1. (3.0 pontos) Considere a representação de partições por florestas, com as otimizações da operação de union através do balanceamento por altura e da operação de find por compressão de caminhos. Caso haja empate, escolha o menor inteiro para raíz.

Começando com uma partição de {1, 2, 3, 4, 5, 6} em singletons, represente a sequência de florestas obtidas sucessivamente pelas operações:

union(1,2), union(3,4), union(5,6), union(3,5), find(6), union(1,3), find(6)

Resp.:

1

2

union(1,2)

3

4

union(3,4)

5

6

union(5,6)

3

4

union(3,5)

5

6

3

4

find(6)

5

6

1

2

union(1,3)

3

4

5

6

1

2

find(6)

3

4

5

6

1. (3.0 pontos) Considere uma tabela de dispersão estendida, parcialmente representada na figura abaixo, onde

i = número de bits de relevância

h = função de hash

H = tabela de hash, ***armazenando apenas ponteiros para os buckets***

B = um bucket arbitrário

jB = informação adicional sobre B

|  |
| --- |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |

i

H

|  |
| --- |
|  |
|  |
|  |

jB

B

* 1. (0,5 ponto) Escreva uma expressão em C cujo valor indica o número de entradas de H que apontam para B.
  2. (0,5 ponto) Seja k o valor da função de hash para uma chave K. Escreva uma expressão em C cujo valor x é tal que H[x] aponta para o bucket onde K deve ser inserida.
  3. (1,0 ponto) Suponha que o bucket B está completo e que existam exatamente 2 entradas de H, H[x] e H[y], apontando para B e que x < y. Ao inserir uma nova chave K em B, haverá overflow de B e um novo bucket B’ será criado; H[x] continuará apontando para B e H[y] apontará para B’.

Sob que condições a criação de B’ não resolverá o problema de overflow? Explique sua resposta.

* 1. (1,0 ponto) Quando a situação em (c) ocorre, explique como a estrutura deve ser atualizada para resolver o problema. Assuma que não há dois registros com a mesma chave (passada como entrada para a função de hash).

Explique qual novo problema poderá ocorrer quando dois registros podem ter a mesma chave (passada como entrada para a função de hash).

Resp.:

a) O número n de entradas de H que apontam para B é dado por n = pow(2.0, (i- jB)).

b) A chave K tal que k=h(K) deve ser inserida no bucket apontado pela entrada H[x],   
onde x = (k % pow(2.0,i)).

c) Seja ***K*** o conjunto de chaves originalmente em B, acrescido da chave K. A criação de B’ não resolverá o problema de overflow quando cada chave L em ***K*** satisfizer a expressão   
(h(L) % pow(2.0,i)) = x. Ou, equivalentemente, quando nenhuma chave L em ***K*** satisfizer a expressão (h(L) % pow(2.0,i)) = y.

d) Quando a situação em (c) ocorre, a tabela de hash H deve ser duplicada, o número de bits de relevância incrementado de 1 e nova tentativa de mover chaves para B’ realizada. Este processo deve ser repetido até seja possível armazenar K.

Quando dois registros podem ter a mesma chave (passada como entrada para a função de hash), então o processo descrito anteriormente pode não ser suficiente para resolver o problema de overflow. Uma estratégia de encadeamento de buckets deverá ser então implementada.

Protótipos e macros que podem ser úteis:

**stdio.h:**

int scanf (char\* formato, ...);

int printf (char\* formato, ...);

FILE\* fopen (char\* nome, char\* modo);

int fclose (FILE\* fp);

int fscanf (FILE\* fp, char\* formato, ...);

int fprintf (FILE\* fp, char\* formato, ...);

char\* fgets(char\* str, int size, FILE\* fp));

int sscanf(char\* str, char\* formato, ...);

**math.h:**

double sqrt (double x);

double pow (double x, double exp);

double cos (double radianos);

double sin (double radianos);

**string.h:**

int strlen (char\* s);

int strcmp (char\* s, char \*t);

char\* strcpy (char\* destino, char\* origem);

char\* strcat (char\* destino, char\* origem);

**stdlib.h:**

void\* malloc (int nbytes);

void free (void\* p);

void qsort (void \*vet, int n, int tam, int (\*comp) (const void\*, const void\*));