# **Enunciado dos labs:**

* **Lab1:**

Dada uma matriz de inteiros positivos n por m, estamos interessados em encontrar a maior soma de elementos possível em uma submatriz de tamanho r por s.

### **Entrada**

A entrada tem uma linha com inteiros n, m, r, s separados por espaços, sendo que 1≤n≤1000, 1≤m≤1000, 1≤r≤n e 1≤s≤m. Depois vêm n linhas com m inteiros cada, com os números em cada linha e coluna da matriz.

### **Saída**

O programa deve imprimir o valor da maior soma possível em uma submatriz de tamanho r por s em uma única linha.

* **Lab2:**

Uma loja tem uma fila única que distribui clientes entre os funcionários do atendimento. Um funcionário atende apenas um cliente de cada vez. No início do dia, todos os atendentes estão livres e a fila está vazia. Escreva um programa para calcular informações sobre a espera dos clientes pelo atendimento.

### **Entrada**

A primeira linha contém dois inteiros, C e N, separados por um espaço. C é o número de atendentes da loja e N é o número de clientes que virão naquele dia. C está no intervalo [1,10] e N está no intervalo [1,1000]. Cada uma das N linhas seguintes terão informação sobre um cliente na forma de dois inteiros T e D separados por um espaço. T é o momento, em minutos a partir da abertura da loja, em que o cliente entra na fila e D é o tempo em minutos para atender o cliente. T está no intervalo [0,300] e D está no intervalo [1,10]. As linhas estão ordenadas pelo momento de entrada dos clientes na fila.

### **Saída**

A primeira linha da saída deve informar a espera média ao longo do dia, como ilustrado no exemplo abaixo. A segunda linha deve informar o número de clientes que esperaram na fila por mais de 10 minutos.

* **Lab3:**

Professores frequentemente ouvem desculpas de alunos que não entregaram seus trabalhos. Algumas são relatos de casualidades cotidianas, outras são apenas esfarrapadas. Escreva um programa para escolher as piores desculpas segundo algumas palavras-chave.

### **Entrada**

A entrada para o programa consiste de vários conjuntos de dados. A linha 1 de cada conjunto tem dois inteiros. O primeiro (1 ≤ K ≤ 20) define o número de palavras-chave. O segundo (1 ≤ E ≤ 100) define o número de desculpas no conjunto. As linhas 2 a K + 1 têm uma palavra-chave cada. Todas as palavras-chave terão apenas caracteres alfabéticos minúsculos e não terão brancos.

As linhas K + 2 a K + E + 1 têm uma desculpa cada. Cada desculpa poderá conter qualquer caractere alfanumérico maiúsculo ou minúsculo, espaços ou quaisquer dos sinais de pontuação ".,!? e não excederá 100 caracteres de comprimento. Nenhuma desculpa será formada apenas por espaços.

### **Saída**

Para cada conjunto de entrada, seu programa você deve imprimir as piores desculpas do conjunto. As piores desculpas são aquelas que contêm o maior número de ocorrências de palavras-chave. Uma palavra-chave ocorre em uma desculpa se ela existir na desculpa de forma contígua e se for delimitada pelo início da linha, pelo fim da linha, por um caractere não alfabético ou por um espaço.

Para cada conjunto de entrada, seu programa deve imprimir uma única linha com o número do conjunto imediatamente após a cadeia "Conjunto de desculpas #". As linhas seguintes devem) conter as piores desculpas, uma por linha, exatamente como aparece na entrada. Se houver mais de uma pior desculpa, seu programa deverá imprimi-las na ordem em que aparecem na entrada. Após cada conjunto de saída, seu programa deve imprimir uma linha em branco.

* **Lab4:**

Este trabalho consiste em fazer um programa que manipule uma coleção de contatos. Seu programa deve ser capaz de organizar até 1.000 contatos. O programa deve ser capaz de inserir registros, remover registros, recuperar registros e imprimir todos os registros da coleção. A entrada consiste de um caractere que especifica a operação, seguido de linhas de acordo com cada operação, que pode ser:

i para inserção. As quatro linhas seguintes conterão nome, endereço, telefone e data de aniversário. O nome tem até 50 caracteres, o endereço até 100, o telefone até 12 dígitos e a data de aniversário tem o formato dd/mm/aa.

r para remoção. A linha seguinte tem uma cadeia de caracteres representando um nome. Seu programa deve remover todos os registros que tenham nome exatamente igual ao digitado. O programa deve imprimir o número de registros removidos.

b para busca. A linha seguinte tem uma cadeia de caracteres representando um nome. Seu programa deve imprimir todos os registros que tenham nome contendo alguma parte que é igual ao nome digitado, na ordem em que foram inseridos.

p para impressão. Seu programa deve imprimir todos os registros na agenda, na ordem em que foram inseridos.

f para finalizar o programa.

* **Lab5:**

Em C, um array multidimensional é implementado colocando todos os elementos dele na memória seqüencialmente, por linhas. Os elementos são acessados calculando um deslocamento a partir do primeiro elemento do array.

Outra forma de implementar um array com n dimensões é usando um array unidimensional (vetor) de apontadores para arrays com n-1 dimensões. Essa forma de implementação é usada por várias linguagens de programação (mas não C, como vimos). Um jagged array é um array multidimensional implementado usando apontadores e com elementos de tamanhos diferentes, por exemplo, matrizes triangulares e outras. Escreva um programa para ler uma matriz triangular inferior de números fracionários fornecidos na entrada padrão, armazená-los dinamicamente na memória como um jagged array e calcular os valores normalizados. O programa deve usar a quantidade mínima de memória possível.

### **Entrada:**

A entrada consiste de um inteiro n>0 especificando a ordem da matriz, seguido de (n^2+n)/2 fracionários contendo os elementos na matriz na seqüência:

[0,0]

[1,0] [1,1]

[2,0] [2,1] [2,2]

...

[n-1,0] [n-1,1] ... [n-1,n-1]

### **Saída:**

A saída deve ter n+2 linhas. Primeiro vêem n linhas na forma de uma matriz triangular inferior. Cada valor z na matriz impressa na saída deve ser z=(x-m)/d, onde x é um elemento da matriz, m é a média e d é o desvio-padrão de todos os valores na matriz. Nenhuma entrada terá desvio-padrão igual a zero. Depois vem uma linha em branco e, na linha seguinte, a média e o desvio-padrão. Os valores devem ser impressos com 12 casas decimais. Há um espaço depois do último número em cada linha.

* **Lab6:**

Em uma lista não há uma forma simples que facilite a recuperação dos registros armazenados nos nós: para recuperar o nó na posição i de uma lista é preciso percorrê-la a partir da cabeça, fazendo i acessos a nós.

Em muitas aplicações, as freqüências com que os registros são recuperados não são uniformes. Faz sentido que os registros que são recuperados com maior freqüência sejam colocados mais próximos da cabeça, mas tipicamente tais freqüências não são conhecidas e mudam ao longo do tempo.

Estratégias de permutação podem ser aplicadas para reduzir o número de acessos a nós da lista durante a recuperação de registros em uma lista. Tais estratégias movem o registro que acabou de ser recuperado um certo número de posições em direção ao início da lista, sem modificar a ordem relativa dos demais registros. Listas acompanhadas de alguma estratégia desse tipo foram chamadas de listas auto-organizáveis.

Algumas estratégias de permutação foram propostas na literatura. As mais usadas incluem:

Move-to-front (MTF): quando um registro é recuperado ele é movido para o início da lista, se ele ainda não estiver no início da lista.

Transpose (TR): quando um registro é recuperado ele é trocado de posição com o registro que o precede, se ele ainda não estiver no início da lista.

Count (C): cada registro tem um contador do número de acessos. Quando um registro é recuperado o contador é incrementado e ele é movido para uma posição anterior a todos os registros com contador menor ou igual ao dele.

Por exemplo, suponha que a lista L tenha registros com chaves (1,2,3,4,5) nesta ordem e suponha que a seqüência de requisições para recuperar registros seja (4,2,2,4,3,1,3). Abaixo aparecem as modificações na lista e os custos para cada estratégia. O custo é medido como a soma do número de nós visitados para recuperar o registro com a chave requisitada, sem contar as operações realizadas na reorganização da lista.

Move-to-front

Lista inicial L=(1,2,3,4,5)

Requisição = 4. Custo = 4. Lista L=(4,1,2,3,5)

Requisição = 2. Custo = 3. Lista L=(2,4,1,3,5)

Requisição = 2. Custo = 1. Lista L=(2,4,1,3,5)

Requisição = 4. Custo = 2. Lista L=(4,2,1,3,5)

Requisição = 3. Custo = 4. Lista L=(3,4,2,1,5)

Requisição = 1. Custo = 4. Lista L=(1,3,4,2,5)

Requisição = 3. Custo = 2. Lista L=(3,1,4,2,5)

Custo total = 4+3+1+2+4+4+2 = 20.

Transpose

Lista inicial L=(1,2,3,4,5)

Requisição = 4. Custo = 4. Lista L=(1,2,4,3,5)

Requisição = 2. Custo = 2. Lista L=(2,1,4,3,5)

Requisição = 2. Custo = 1. Lista L=(2,1,4,3,5)

Requisição = 4. Custo = 3. Lista L=(2,4,1,3,5)

Requisição = 3. Custo = 4. Lista L=(2,4,3,1,5)

Requisição = 1. Custo = 4. Lista L=(2,4,1,3,5)

Requisição = 3. Custo = 4. Lista L=(2,4,3,1,5)

Custo total = 4+2+1+3+4+4+4 = 22.

Count

Lista inicial L=(1,2,3,4,5). Contador C=(0,0,0,0,0)

Requisição = 4. Custo = 4. Lista L=(4,1,2,3,5). Contador C=(1,0,0,0,0)

Requisição = 2. Custo = 3. Lista L=(2,4,1,3,5). Contador C=(1,1,0,0,0)

Requisição = 2. Custo = 1. Lista L=(2,4,1,3,5). Contador C=(2,1,0,0,0)

Requisição = 4. Custo = 2. Lista L=(4,2,1,3,5). Contador C=(2,2,0,0,0)

Requisição = 3. Custo = 4. Lista L=(4,2,3,1,5). Contador C=(2,2,1,0,0)

Requisição = 1. Custo = 4. Lista L=(4,2,1,3,5). Contador C=(2,2,1,1,0)

Requisição = 3. Custo = 4. Lista L=(3,4,2,1,5). Contador C=(2,2,2,1,0)

Custo total = 4+3+1+2+4+4+4 = 22.

Uma outra estratégia é a move-ahead-k, que move um registro k posições em direção à cabeça depois que ele é acessado. k pode ser definido como um número fixo, como um percentual da distância até a cabeça ou como outra função de distância. Algumas outras estratégias usam combinações dessas que foram listadas.

Neste trabalho as estratégias MTF, TR e C devem ser comparadas. Seu programa deve usar uma lista encadeada. (Em um vetor uma estratégia como essas seria muito custosa porque seriam necessárias muitas movimentações dos dados.)

### **Entrada**

A entrada para o programa são o número inteiro N de chaves na lista, entre 1 e 1000, o número inteiro R de requisições e uma seqüência de R requisições inteiras no intervalo [1,N]. Cada um dos acessos em R deve ser realizado em ordem, para MTF, TR e C, sempre a partir da lista que é iniciada contendo as chaves na ordem 1,2,3,...,N.

### **Saída**

A saída são três inteiros indicando os custos das estratégias MTF, TR e C, respectivamente.

* **Lab7:**

Vamos dizer que um conjunto de registros está em seqüência se a ordem relativa entre eles é importante. Por exemplo, pessoas em uma fila de atendimento formam uma seqüência.

Neste trabalho você deve implementar um vetor dinâmico para armazenar uma seqüência de números inteiros. O vetor dinâmico deve permitir inserção no início e no final da seqüência, remoção do primeiro e do último elementos e recuperação do primeiro e do último elementos.

A seqüência deve ser implementada de forma circular, isto é, o primeiro elemento dela deve poder estar em qualquer posição do vetor. Dessa forma, qualquer uma dessas operações acima poderá ser realizada em tempo constante, não é necessário fazer deslocamento dos dados armazenados no vetor para a esquerda ou para a direita.

Por exemplo, a seqüência circular de chaves inteiras [2,9,5,7] pode estar armazenada de várias formas em um vetor de tamanho 8:

2 9 5 7 \_ \_ \_ \_

\_ \_ 2 9 5 7 \_ \_

\_ \_ \_ \_ 2 9 5 7

5 7 \_ \_ \_ \_ 2 9

9 5 7 \_ \_ \_ \_ 2

Em todas elas, o primeiro elemento da seqüência é o 2 e o último é o 7. Observe que os números não estão ordenados, mas a ordem relativa entre eles deve ser preservada. Eles podem representar o número da conta das tais pessoas na fila de atendimento, por exemplo.

Se um vetor circular contém a seqüência

2 9 5 7 \_ \_ \_ \_

e o número 6 é adicionado ao início da seqüência, então o vetor fica assim:

2 9 5 7 \_ \_ \_ 6

Depois se o número 8 é adicionado ao início da seqüência, o vetor deve ficar assim:

2 9 5 7 \_ \_ 8 6

Se o número 7 é removido do fim da seqüência, o vetor deve ficar assim:

2 9 5 \_ \_ \_ 8 6

E se o número 1 é adicionado ao fim da seqüência, o vetor deve ficar assim:

2 9 5 1 \_ \_ 8 6

A política de redimensionamento do vetor deve ser dobrar quando estiver cheio e reduzir à metade quando estiver 1/4 ocupado. Dessa forma, durante o processamento, o número de posições vazias do vetor não deveria exceder 3n, onde n é o número de posições ocupadas.

Obviamente, quando o vetor for redimensionado, o conteúdo da seqüência e a ordem relativa entre os elementos da seqüência devem ser preservados.

### **Entrada**

A entrada é composta por uma sucessão de comandos, um por linha. Os possíveis comandos estão descritos abaixo.

insert-first inteiro

Insere um inteiro no início da seqüência.

remove-first

Remove o inteiro no início da seqüência. Se a seqüência estiver vazia, não faz nada.

print-first

Imprime o inteiro no início da seqüência. Se a seqüência estiver vazia, não faz nada.

insert-last inteiro

Insere um inteiro no fim da seqüência.

remove-last

Remove o inteiro no fim da seqüência. Se a seqüência estiver vazia, não faz nada.

print-last

Imprime o inteiro no fim da seqüência. Se a seqüência estiver vazia, não faz nada.

is-empty

Imprime "yes" se a seqüência estiver vazia e "no" se não estiver.

exit

Termina o programa.

Nenhuma função deve mover os elementos do vetor para a direita ou para a esquerda.

### **Saída**

A saída deve conter as linhas geradas pelos comandos print-first, print-last e is-empty.

* **Lab8:**

Para reverter um trecho de uma lista encadeada é necessário atualizar todos os apontadores dos nós no trecho, realizando um número de operações proporcional ao número de nós no trecho revertido. Se as operações de reversão forem freqüentes, o custo dessas operações se torna alto.

A lista duplamente encadeada também não permite simplesmente reverter um trecho usando um número constante de operações por que, se fizermos isso e tentarmos percorrer a lista, os apontadores prev e next ficam inconsistentes ao longo da lista. Para resolver o problema temos que reverter de fato o trecho, gastando um número de operações proporcional ao número de nós no trecho revertido.

Nas listas simétricas, que são listas duplamente encadeadas, os apontadores são chamados simplesmente de A e B, sem referência a uma ordem na lista. Para percorrer a lista simétrica é necessário usar informação sobre o nó visitado anteriormente para decidir qual é o próximo nó a ser visitado. Assim, só é possível percorrer a lista a partir da cabeça ou do rabo, não é possível percorrer a partir de um nó qualquer. Uma reversão só precisa testar e ajustar os apontadores que estão entre os limites do intervalo revertido. Nenhum nó interno ao intervalo revertido é atualizado.

Por exemplo considere a lista simétrica abaixo. Na seqüência das figuras acontece uma reversão dos nós nas posições entre 2 e 4 e depois uma reversão entre as posições 4 e 5. Observe que apenas os apontadores nos nós dos extremos da parte revertida são atualizados. Observe também que na terceira lista o caminho para percorrer da cabeça para o rabo passa por apontadores BBAAABB.

Com essa restrição é possível reverter um trecho da lista delimitado por dois nós quaisquer fazendo apenas um número constante de operações. Isso é, dados dois nós, o número de operações necessárias para a reversão é constante, já que um número fixo de apontadores é atualizado.

A lista simétrica é uma alternativa eficiente nas aplicações em que a operação de reversão é freqüente.

Neste trabalho, implemente uma lista simétrica de double com reversão.

### **Entrada**

Cada linha da entrada contém um comando como descrito a seguir.

- Inserir:

i posição valor

Inserir um novo nó em uma posição específica. O double inserido passa a ocupar aquela posição na lista, contando da cabeça, que está na posição zero. Se não for possível inserir na posição específica, então a inserção deve ser feita no fim da lista. Essa situação acontece, por exemplo, quando a lista tem 5 elementos e pedimos para inserir 3.14 na posição 9. Nesse caso, 3.14 deve ser inserido na posição 5.

- Imprimir:

p

Imprimir os valores na lista a partir da cabeça em uma única linha. Cada elemento da lista deve ser seguido de um espaço. A lista vazia é impressa como uma linha em branco, sem espaços.

- Reverter:

v início término

Reverter o trecho delimitado pelas posições início e término, inclusive.

- Terminar:

t

### **Saída**

A saída deve conter as linhas produzidas pelos comandos 'p'.

* **Lab9:**

Este trabalho consiste em escrever um programa com funções para manipular uma árvore de busca binária para inteiros.

O programa deve criar uma árvore vazia e em seguida deve processá-la de acordo com comandos da entrada-padrão, que podem ser da seguinte forma:

- inserir x

Ao ler o comando inserir x, onde x é um inteiro, o programa deve inserir a chave x na árvore de busca binária. A árvore não deve ter elementos repetidos, então se x já pertence à árvore ele não deve ser inserido de novo.

- buscar x

Ao ler o comando buscar x, onde x é um inteiro, o programa deve buscar a chave x na árvore e imprimir "x pertence" ou "x nao pertence" em uma linha.

- pos-ordem

- em-ordem

- pre-ordem

- largura

Ao ler um desses comandos o programa deve imprimir as chaves na ordem em que forem visitadas por um percurso em profundidade em pós-ordem, em-ordem, em pré-ordem ou por um percurso em largura. As chaves devem ser impressas seguidas por um espaço, em uma única linha. Se a árvore estiver vazia então o programa deve imprimir "vazia" em uma linha.

- terminar

Ao ler esse comando o programa deve desalocar a árvore e terminar.

* **Lab10:**

Este trabalho consiste em escrever um programa com funções para manipular uma árvore de busca binária para inteiros.

O programa deve criar uma árvore vazia e em seguida deve processá-la de acordo com comandos da entrada-padrão, que podem ser da seguinte forma:

- inserir x

Ao ler o comando inserir x, onde x é um inteiro, o programa deve inserir a chave x na árvore de busca binária. A árvore não deve ter elementos repetidos, então se x já pertence à árvore ele não deve ser inserido de novo.

- buscar x

Ao ler o comando buscar x, onde x é um inteiro, o programa deve buscar a chave x na árvore e imprimir "x pertence" ou "x nao pertence" em uma linha.

- pos-ordem

- em-ordem

- pre-ordem

- largura

Ao ler um desses comandos o programa deve imprimir as chaves na ordem em que forem visitadas por um percurso em profundidade em pós-ordem, em-ordem, em pré-ordem ou por um percurso em largura. As chaves devem ser impressas seguidas por um espaço, em uma única linha. Se a árvore estiver vazia então o programa deve imprimir "vazia" em uma linha.

- remover x

Ao ler o comando remover x, onde x é um inteiro, o programa deve remover a chave x da árvore binária de busca, se houver. O programa deve usar sempre o \*sucessor\* de um nó para a substituição de um nó que tenha dois filhos.

- minimo

- maximo

Ao ler um desses comandos o programa deve imprimir a menor chave ou a maior chave na árvore, respectivamente. Se a árvore estiver vazia então o programa deve imprimir "vazia" em uma linha.

- sucessor x

- predecessor x

Ao ler um desses comandos, onde x é um inteiro, o programa deve imprimir o sucessor de x ou o precessor de x, respectivamente. Se x não estiver na árvore ou não tiver sucessor ou predecessor o programa deve imprimir "nao ha" em uma linha.

- buscar-intervalo x y

Ao ler o comando buscar-intervalo x y, onde x e y são inteiros, o programa deve imprimir todas as chaves contidas intervalo [x,y] em ordem crescente, separadas por espaço e em uma linha. Se não houver chaves no intervalo o programa deve imprimir "nenhuma" em uma linha.

- terminar

Ao ler esse comando o programa deve desalocar a árvore e terminar.

* **Lab11:**

Este trabalho consiste em implementar uma tabela de hashing para armazenar cadeias do alfabeto {a,b,...,z,A,B,...,Z,\_}. No máximo 5.000 cadeias de no máximo 250 caracteres estarão armazenadas na tabela ao mesmo tempo.

As colisões devem ser resolvidas usando sondagem com hashing duplo ou encadeamento na própria tabela. Defina o tamanho e a função de hashing para a sua tabela.

Escolha também uma função de transformação de cadeias de caracteres em inteiros. Não importa a forma como você faça para este trabalho, mas algumas funções são muito piores que outras em termos de espalhamento. Um exemplo de uma função que funciona bem é a função de hashing djb2 abaixo.

unsigned long djb2(unsigned char \*str) {

unsigned long hash = 5381;

int c;

while ((c = \*str++))

hash = ((hash << 5) + hash) + c; // hash \* 32 + hash + c

return hash;

}

Cada linha da entrada para seu programa é iniciada por um caractere que pode ser seguido de um espaço e de uma chave. Os caracteres podem ser i para inserir uma chave na tabela, b para buscar uma cadeia na tabela, r para remover uma cadeia da tabela e f para terminar o programa.

Quando uma cadeia é inserida na tabela ela recebe um identificador único. O identificador de cada cadeia é atribuído seqüencialmente a partir de 0, sem reuso. Cada chave deve ser inserida uma única vez.

Para cada busca pela cadeia c na tabela seu programa deve imprimir "c encontrada, id i" ou "nao encontrada" em uma linha, onde i é o identificador que c recebeu no momento em que foi inserida. As outras ações não geram saída.

* **Lab12:**

Implemente o heapsort.

Seu programa deve ter pelo menos uma função para criar um heap e uma função para remover o máximo. Sua implementação de heap deve ser implícita. O heapsort deve ser in-place.

A entrada é uma linha contendo um inteiro n que indica o número de inteiros que serão ordenados seguida de n linhas com um inteiro cada. A saída é uma sucessão dos n inteiros ordenados, cada um seguido por um espaço, em uma única linha.

* **Lab13:**

Este trabalho consiste em implementar uma busca em largura em um grafo não-orientado G. O grafo deve ser representado por um vetor de adjacências.

A primeira linha da entrada tem n, o número de vértices em G. As linhas seguintes representam arestas {i,j} no formato i,j, tais que 1 <= i,j <= n. O fim das arestas é representado por i=0 e j=0. A última linha é um inteiro v, 1 <= v <= n, que é o vértice inicial da busca em largura.

Seu programa deve imprimir os vértices alcançados pela busca em ordem crescente da distância de s. Uma vez que a BFS não impõe qualquer ordem para para percorrer a vizinhança de um vértice, qualquer ordem entre vértices à mesma distância de s será aceita.

* **Lab50:**

A tabela ASCII é uma das formas possíveis de mapear inteiros em símbolos. São 128 códigos, de 0 a 127, sendo que os primeiros 32 e o último são considerados símbolos de controle e os demais são imprimíveis.

Escreva um programa em C para imprimir uma tabela ASCII como a do modelo abaixo. No exemplo algumas linhas foram suprimidas e estão representadas por três pontos. O programa não recebe nenhuma entrada.

+0 +1 +2 +3 +4 +5 +6 +7

+------------------------

32 | ! " # $ % & '

40 | ( ) \* + , - . /

...

120 | x y z { | } ~