OBI

OLIMPÍADA BRASILEIRA DE INFORMÁTICA

OBI2000 - Curso de Programação
CADERNO DE TAREFAS

14/6/2000 • 8:00 às 13:00

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

a) É proibido consultar livros, anotações ou qualquer outro material durante a prova. É permitido a consulta ao *help* do ambiente de programação se este estiver disponível.
b) É proibido iniciar qualquer comando que não seja um editor ou um compilador, bem como acessar outras área de trabalho que não a sua. Os comandos estão sendo monitorados, qualquer violação implicará na desclassificação sumária do aluno.
c) Todas as tarefas têm o mesmo valor na correção.
d) As tarefas não estão ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões mais fáceis.
e) Preste muita atenção no nome dos arquivos fonte indicados nas tarefas. Soluções na linguagem C devem ser arquivos com sufixo .c; soluções na linguagem C++ devem ser arquivos com sufixo .cc ou .cpp; soluções na linguagem Pascal devem ser arquivos com sufixo .pas.
f) Para problemas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada problema.
g) Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o disquete.
h) Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (normalmente é o teclado) e escritos na saída padrão (normalmente é a tela). Utilize as funções padrão para entrada e saída de dados:
   - em Pascal: *readln*, *read*, *writeln*, *write*;
   - em C: *scanf*, *getchar*, *printf*, *putchar*;
   - em C++: as mesmas de C ou os objetos *cout* e *cin*.
i) Procure resolver o problema de maneira eficiente. Na correção, eficiência também será levada em conta.

Sociedade Brasileira de Computação
http://www.sbc.org.br Email: sbc@sbc.org.br

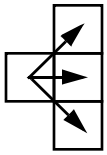
MASP

arquivo fonte: masp.pas, masp.c, masp.cc ou masp.cpp

O MASP (Museu de Arte de São Paulo) tem o melhor acervo de obras de arte da América Latina, sendo reconhecido mundialmente. Além disso, o MASP é diferente de museus tradicionais porque seus quadros não são pendurados em paredes (já que as 'paredes' do MASP são janelas de vidro), mas sim apresentados em cavaletes no meio da sala de exposição. Uma grande exposição está em curso, ocupando todo o salão de exposições.

Considere que:

a) as obras estão organizadas no salão de exposições no formato de uma matriz de *N* linhas por *M* colunas;
b) o visitante pode apenas mover-se da esquerda para a direita, iniciando na coluna 1 (primeira coluna) e terminando na coluna *M* (última coluna);
c) uma *trajetória* de visita é composta de uma seqüência de passos; a cada passo o visitante aprecia uma obra;
d) um passo consiste em mover-se da coluna *i* para a coluna *i*+1 em uma linha adjacente. Ou seja, o visitante pode efetuar um movimento horizontal ou diagonal. Movimentos legais são mostrados na figura abaixo:

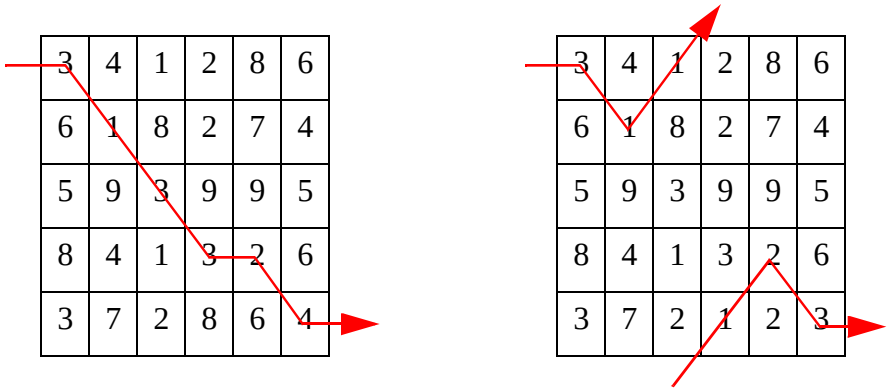
movimentos legais: 

e) para dificultar ainda um pouco mais o problema, a primeira e última linhas (linhas de número 1 e *N*) da matriz são consideradas adjacentes (o que não é possível no MASP!);
f) cada obra tem associada uma *prioridade* de visita. A prioridade é um número inteiro; quanto menor o número, maior a prioridade de que a obra seja apreciada (note que a prioridade pode ser negativa);

7. Tarefa

Sua tarefa é escrever um programa que determine a *trajetória ótima* para uma visita ao museu, obedecendo as regras acima. A *trajetória ótima* é aquela que tem a menor soma total das prioridades das obras visitadas. Como exemplo, considere as duas exposições abaixo (a única diferença

entre as duas exposições é a prioridade das obras da última linha):



As trajetórias ótimas para as duas exposições são indicadas na figura. Note que a trajetória ótima para a exposição da direita utiliza a propriedade da adjacência entre a primeira e a última linhas. No caso de haver mais de uma trajetória ótima possível, seu programa deve imprimir a trajetória de menor ordem lexicográfica (veja o formato de Saída, abaixo).

8. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros positivos *N* e *M*, que indicam respectivamente o número de linhas e o número de colunas da matriz. As *N* linhas seguintes contém cada uma *M* números inteiros que representam as prioridades das obras. O final da entrada é indicado por $N = M = 0$.

Exemplo de Entrada

```
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 8 6 4
5 6
3 4 1 2 8 6
6 1 8 2 7 4
5 9 3 9 9 5
8 4 1 3 2 6
3 7 2 1 2 3
2 2
9 10
9 10
0 0
```

9. Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato “Teste *n*”, onde *n* é numerado

a partir de 1. Na segunda linha deve aparecer a trajetória ótima, descrita por uma seqüência de M inteiros (separados por um espaço em branco), representando as linhas da matriz que constituem a trajetória ótima. No caso de haver mais de uma trajetória ótima possível, seu programa deve imprimir a trajetória de menor ordem lexicográfica. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
1 2 3 4 4 5

Teste 2
1 2 1 5 4 5

Teste 3
1 1
```

(esta saída corresponde ao exemplo de entrada acima)

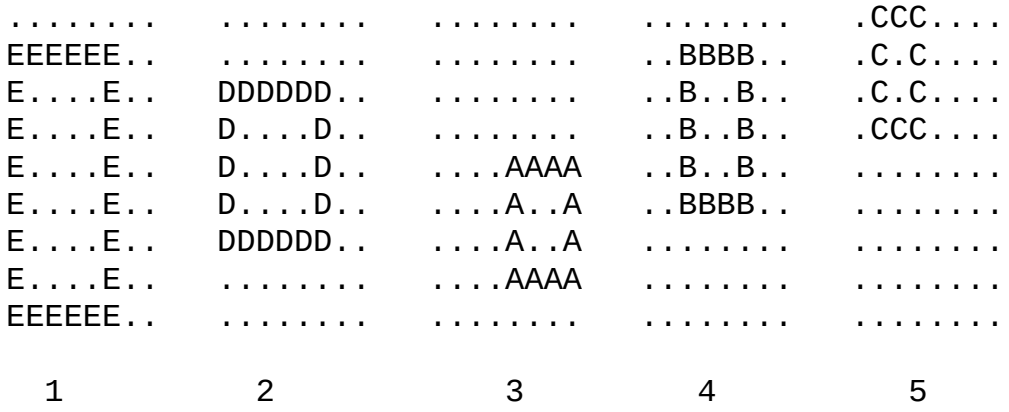
10. Restrições

$0 \leq N \leq 100$ ($N = 0$ apenas para indicar o fim da entrada)
$0 \leq M \leq 100$ ($M = 0$ apenas para indicar o fim da entrada)
$-15000 \leq$ *prioridade* ≤ 15000
$-15000 \leq$ *soma das prioridades de uma trajetória* ≤ 15000

Anéis quadrados

arquivo fonte: anel.pas, anel.c, anel.cc ou anel.cpp

Considere os seguintes anéis quadrados, cada um definido em uma matriz de 8 colunas por 9 linhas:



Agora coloque um anel sobre o outro, começando pelo anel 1 (que fica por baixo de todos) e terminando com o anel 5 (em cima de todos). Portanto, olhando a pilha de cinco anéis pelo lado de cima vemos o seguinte:

```
.CCC....
ECBCBB..
DCBCDB..
DCCC.B..
D.B.ABAA
D.BBBB.A
DDDDAD.A
E...AAAA
EEEEEE..
```

Os anéis são formados por letras maiúsculas, cada anel com uma letra diferente. O caractere ponto (‘.’) é utilizado para representrar espaços vazios. A espessura das paredes do anel é de exatamente um caractere e o comprimento dos lados nunca é menor do que três caracteres. Pelo menos uma parte de cada um dos lados do anel é visível (note que um canto conta como visível para dois lados).

1. Tarefa

Sua tarefa é escrever um programa que, dada a configuração de uma pilha de anéis, determine qual a seqüência de empilhamento (de baixo para cima) que foi utilizada na construção da pilha. No exemplo acima a resposta é EDABC. Se há mais de uma ordem de empilhamento possível, seu programa deve listar todas.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros positivos *X* e *Y* que indicam, respectivamente, a altura e a largura da matriz que contém os anéis. As *X* linhas seguintes contêm *Y* caracteres cada, representando a vista superior da pilha de anéis. O final da entrada é indicado quando $X = Y = 0$.

Exemplo de Entrada

```
9 8
.CCC....
ECBCBB..
DCBCDB..
DCCC.B..
D.B.ABAA
D.BBBB.A
DDDDAD.A
E...AAAA
EEEEEE..
10 10
..AAAAA...
..ACCCA...
..AC.CA...
..AC.CA...
..ACCCA...
..AAAAA...
..........
...BBB....
...B.B....
...BBB....
0 0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir uma lista das letras dos anéis, na ordem em que estes foram empilhados, do mais abaixo para o mais acima. A lista deve ser precedida de uma linha que identifica o conjunto de teste, no formato "Teste *n*", onde *n* é numerado a partir de 1. A lista é composta por uma sequência de letras, identificando anéis, sem espaços em branco entre si. Se há mais de uma possibilidade de empilhamento, liste todas as possibilidades, em qualquer ordem, em linhas sucessivas. Você pode considerar que há sempre ao menos uma ordem de empilhamento válida. O final de uma solução deve ser marcado com uma linha em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
EDABC
```

```
Teste 2
ABC
ACB
BAC
BCA
CAB
CBA
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$5 \leq X \leq 30$
$5 \leq Y \leq 30$

($X = Y = 0$ apenas para indicar o fim da entrada)

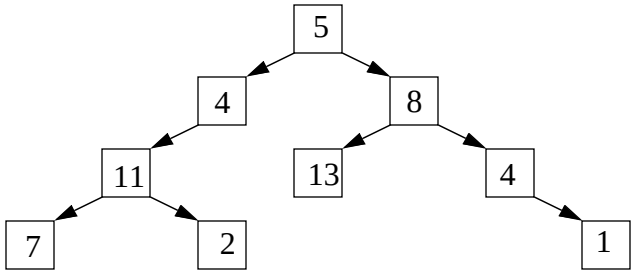
Árvores

arquivo fonte: arvore.pas, arvore.c, arvore.cc ou arvore.cpp

Como todos vocês devem ter percebido durante o curso, árvores são fundamentais em vários ramos (!) da Computação. Este problema envolve a construção e percurso de árvores binárias.

1. Tarefa

Dada uma árvore binária, você deve escrever um programa que produza um percurso “em níveis”. Cada vértice de uma árvore contém um inteiro positivo que identifica o vértice unicamente. Em um percurso em níveis os vértices em um dado nível são visitados da esquerda para a direita, e todos os nós do nível *k* são visitados antes dos nós do nível *k* + 1. Por exemplo, para a árvore abaixo,



um percurso em nível produz 5, 4, 8, 11, 13, 4, 7, 2, 1.

2. Entrada

Na entrada uma árvore binária é especificada como uma seqüência de pares (*v*, *p*) onde *v* é o valor do vértice cujo caminho desde a raiz é descrito pelo vetor *p*. Os elementos de *p* têm valor 1 ou -1, representanto respectivamente um ramo direito ou um ramo esquerdo. O final do vetor *p* é indicado por um elemento de valor 0. Na árvore da figura acima, o vértice que contém 13 é especificado por (13, [1, -1, 0]) e o nó que contém 2 é especificado por (2, [-1, -1, 1, 0]). A raiz é especificada por (5, [0]), onde o vetor vazio representa o caminho da raiz até a própria raiz.

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um número inteiro positivo *N* que indica o total de vértices da árvore. As *N* linhas seguintes contêm cada uma a descriçào de um vértice, no formato descrito acima. O final da entrada é indicado quando *N* = 0.

Exemplo de Entrada

```
9
11 -1 -1 0
7 -1 -1 -1 0
8 1 0
```

```
5 0
4 -1 0
13 1 -1 0
2 -1 -1 1 0
1 1 1 1 0
4 1 1 0
3
10 0
20 -1 0
30 1 0
0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas. A primeira linha identifica o conjunto de teste, no formato "Teste *n*", onde *n* é numerado a partir de 1. A segunda linha deve conter os valores dos vértices na ordem do percurso em nível, conforme determinado pelo seu programa. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
5 4 8 11 13 4 7 2 1

Teste 2
10 20 30
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq N \leq 10000$ ($N = 0$ apenas para indicar o fim da entrada)
$0 \leq$ *valor de um vértice* ≤ 10000

Balaio

arquivo fonte: balaio.pas, balaio.c, balaio.cc ou balaio.cpp

Maria mora no interior de Minas Gerais e é especialista em fabricar balaios de junco. Os balaios de Maria são muito bem feitos e têm grande aceitação na região. Cada balaio demora exatamente um dia de trabalho para ser confeccionado: Maria começa a tecer um balaio no início do dia e no final do dia entrega o produto para um cliente. Com a crescente demanda, Maria começou a aceitar pedidos para o futuro: Narciso precisa de um balaio para o dia 10, Coronel Zoio precisa de um para o dia 4, Esmeralda para o dia 6, e assim por diante. Todos os pedidos, com as datas-limite de entrega, estão anotados computador que o filho de Maria comprou. Maria não sabe dizer não, e agora percebeu que aceitou mais pedidos do que vai conseguir produzir, se for considerar as datas-limite impostas pelos seus clientes. Alguém poderia ajudar Maria?

1. Tarefa

Sua tarefa é escrever um programa que determine qual a melhor ordem de entrega dos balaios de Maria, de forma a minimizar a *chateação total* causada por eventuais atrasos na entrega. A medida da *chateação* utiliza um sistema de medição desenvolvido por Maria, por experiência anterior: ela sabe que se atrasar o balaio de Narciso, isso vai causar uma chateação de nível 13; Esmeralda é muito boazinha e, se Maria atrasar a sua entrega, a chateação será nível 0. No entanto, se atrasar o balaio do Coronel Zoio, ela terá uma chateação de nível 125. A *chateação total* é dada pela soma das chateações causadas por todos os atrasos. Considere que Maria trabalha todos os dias, sem descanso, e os dias são numerados sequencialmente a partir de 1.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um número inteiro positivo N, que indica quantidade de pedidos pendentes. A segunda linha contém o vetor de inteiros positivos L, em que $L[i]$ que indica a data-limite para entrega do pedido i ($1 \leq i \leq N$). A terceira linha contém o vetor de inteiros positivos C, em que $C[i]$ indica o nível de chateação ocasionado se o prazo $L[i]$ não for obedecido ($1 \leq i \leq N$). O final da entrada é indicado por $N = 0$.

Exemplo de Entrada

```
3
10 6 4
21 0 125
7
4 2 4 3 1 4 6
10 60 50 40 30 20 10
0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir três linhas na saída. A primeira linha deve conter um identificador do conjunto de teste, no formato “Teste *n*”, onde *n* é numerado

a partir de 1. Na segunda linha devem aparecer os identificadores dos pedidos, na ordem em que os pedidos devem ser entregues, separados por espaços em branco. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
3 2 1

Teste 2
5 2 4 3 6 7 1
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq N \leq 1000000$ ($N = 0$ apenas para indicar o fim da entrada)
$1 \leq L[i] \leq 15000$
$0 \leq C[i] \leq 15000$
$0 \leq$ *chateação total* ≤ 15000