OBI

OLIMPÍADA BRASILEIRA
DE INFORMÁTICA

OBI2001

CADERNO DE TAREFAS

19/5/2001 • 13:00 às 18:00

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

1. 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.
2. A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa.
3. Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
4. Todas as tarefas têm o mesmo valor na correção.
5. As tarefas não estão ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões mais fáceis.
6. 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.
7. Para problemas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada problema.
8. Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o seu diretório de trabalho ou disquete, conforme especificado pelo seu professor.
9. 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*.
10. Procure resolver o problema de maneira eficiente. Na correção, eficiência também será levada em conta. As soluções serão testadas com outras entradas além das apresentadas como exemplo nas tarefas.

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

Meteoros

Arquivo fonte: meteoro.c, meteoro.cc, meteoro.cpp ou meteoro.pas

Em noites sem nuvens pode-se muitas vezes observar pontos brilhantes no céu que se deslocam com grande velocidade, e em poucos segundos desaparecem de vista: são as chamadas estrelas cadentes, ou *meteoros.* Meteoros são na verdade partículas de poeira de pequenas dimensões que, ao penetrar na atmosfera terrestre, queimam-se rapidamente (normalmente a uma altura entre 60 e 120 quilômetros). Se os meteoros são suficientemente grandes, podem não queimar-se completamente na atmosfera e dessa forma atingem a superfície terrestre: nesse caso são chamados de *meteoritos*.

Zé Felício é um fazendeiro que adora astronomia e descobriu um portal na Internet que fornece uma lista das posições onde caíram meteoritos. Com base nessa lista, e conhecendo a localização de sua fazenda, Zé Felício deseja saber quantos meteoritos caíram dentro de sua propriedade. Ele precisa de sua ajuda para escrever um programa de computador que faça essa verificação automaticamente.

1. Tarefa

São dados:

- uma lista de pontos no plano cartesiano, onde cada ponto corresponde à posição onde caiu um meteorito;
- as coordenadas de um retângulo que delimita uma fazenda.

As linhas que delimitam a fazenda são paralelas aos eixos cartesianos. Sua tarefa é escrever um programa que determine quantos meteoritos caíram dentro da fazenda (incluindo meteoritos que caíram exatamente sobre as linhas que delimitam a fazenda).

2. Entrada

Seu programa deve ler vários conjuntos de testes. A primeira linha de um conjunto de testes quatro números inteiros X_1, Y_1, X_2 e Y_2, onde (X_1, Y_1) é a coordenada do canto superior esquerdo e (X_2, Y_2) é a coordenada do canto inferior direito do retângulo que delimita a fazenda. A segunda linha contém um inteiro, *N*, que indica o número de meteoritos. Seguem-se *N* linhas, cada uma contendo dois números inteiros *X e Y*, correspondendo às coordenadas de cada meteorito. O final da entrada é indicado por $X_1 = Y_1 = X_2 = Y_2 = 0$.

Exemplo de Entrada

```
2 4 5 1
2
1 2
3 3
2 4 3 2
3
1 1
2 2
3 3
0 0 0 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. A segunda linha deve conter o número de meteoritos que caíram dentro da fazenda. 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

Teste 2
2
```

(esta saída corresponde ao exemplo de entrada acima)

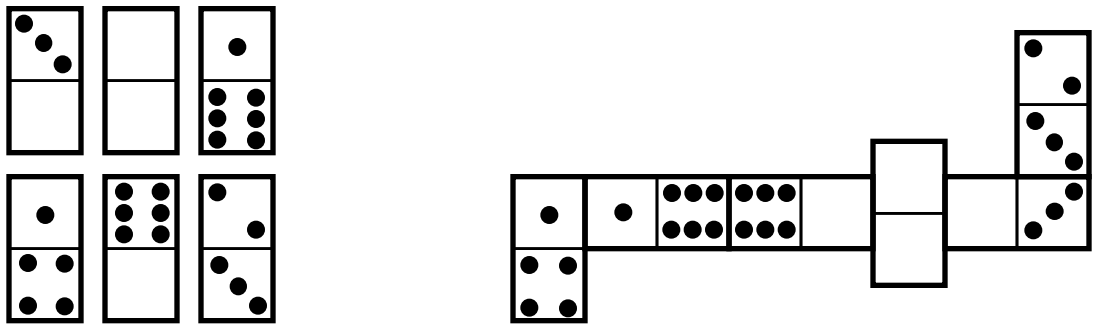
4. Restrições

$0 \leq N \leq 10.000$
$0 \leq X \leq 10.000$
$0 \leq Y \leq 10.000$
$0 \leq X_1 < X_2 \leq 10.000$
$0 \leq Y_2 < Y_1 \leq 10.000$

Dominó

Arquivo fonte: domino.c, domino.cc, domino.cpp ou domino.pas

Todos conhecem o jogo de dominós, em que peças com dois valores devem ser colocadas na mesa em seqüência, de tal forma que os valores de peças imediatamente vizinhas sejam iguais. O objetivo desta tarefa é determinar se é possível colocar todas as peças de um conjunto dado em uma formação válida.



(a) *(b)*

Conjunto de seis peças (a) e uma formação utilizando todas as seis peças (b)

1. Tarefa

É dado um conjuto de peças de dominó. Cada peça tem dois valores *X* e *Y*, com *X* e *Y* variando de 0 a 6 (*X* pode ser igual a *Y*). Sua tarefa é escrever um programa que determine se é possível organizar todas as peças recebidas em seqüência, obedecendo as regras do jogo de dominó.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de testes contém um número inteiro *N* que indica a quantidade de peças do conjunto. As *N* linhas seguintes contêm, cada uma, a descrição de uma peça. Uma peça é descrita por dois inteiros *X* e *Y* ($0 \leq X \leq 6$ e $0 \leq Y \leq 6$) que representam os valores de cada lado da peça. O final da entrada é indicado por $N = 0$.

Exemplo de Entrada

```
3
0 1
2 1
2 1
2
1 1
0 0
6
```

```
3 0
0 0
1 6
4 1
0 6
2 3
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. A segunda linha deve conter a expressão “sim” se for possível organizar todas as peças em uma formação válida ou a expressão “nao” (note a ausência de acento) caso contrário. 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
sim

Teste 2
nao

Teste 3
sim
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

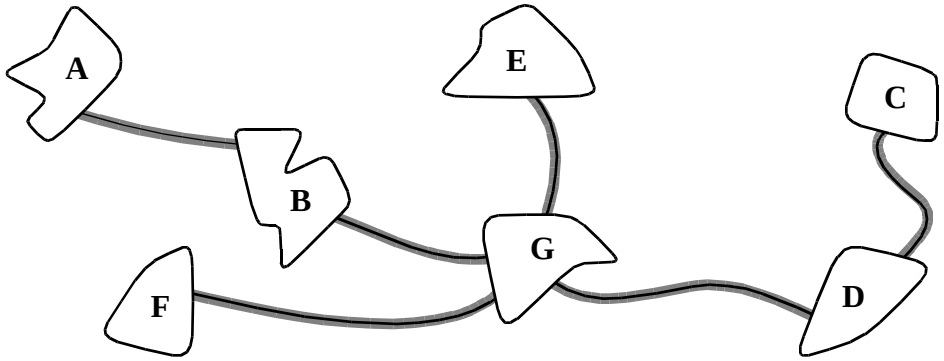
$0 \leq N \leq 100$ (N = 0 apenas para indicar o final da entrada)

Dengue

Arquivo fonte: dengue.c, dengue.cc, dengue.cpp ou dengue.pas

A Costa do Mosquito é um país pequeno mas paradisíaco. Todos os habitantes têm boas moradias, bons empregos, o clima é agradável, e os governantes são justos e incorruptíveis. O sistema de transporte público da Costa do Mosquito é composto de uma rede de linhas de trem. O sistema foi projetado de forma peculiar: existe um único percurso ligando duas quaisquer vilas (esse percurso possivelmente passa por outras vilas). Por exemplo, na figura abaixo, que mostra um trecho do mapa da Costa do Mosquito, há apenas um percurso entre as vilas A e C, passando pelas vilas B, G e D. Uma tarifa fixa de M$ 1,00 é cobrada por cada viagem entre vilas vizinhas; assim, para uma viagem de A a C o usuário gasta M$ 4,00.

Devido a um inesperado surto de dengue, o Ministério da Saúde da Costa do Mosquito resolveu montar um Posto de Vacinação. Para evitar que habitantes gastem muito dinheiro para se deslocar até a vila onde ficará o Posto de Vacinação, o Ministério da Saúde decidiu que este deverá ser instalado em uma vila de forma que o gasto com transporte até o Posto, para os habitantes que gastarem mais, seja o menor possível (para o caso da figura abaixo a vila escolhida seria G).



Sete vilas interligadas por linhas de trem

1. Tarefa

Sua tarefa é escrever um programa que determine uma vila onde deve ser instalado o Posto de Vacinação. Esta vila deve ser tal que o custo com transporte, para os habitantes que tiverem maior custo, seja o menor possível. Note que devido à característica peculiar do sistema viário, ou haverá uma única vila que satisfaz essa restrição, ou haverá duas vilas que a satisfazem. No caso de existirem duas vilas apropriadas, qualquer uma delas serve como solução.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de testes contém um número inteiro N que indica a quantidade de vilas do país. As vilas são numeradas de 1 a N. As N-1 linhas seguintes contêm, cada uma, dois inteiros positivos X e Y que indicam que a vila X tem um caminho que a liga diretamente com a vila Y, sem passar por outras vilas. O final da entrada é indicado por $N = 0$.

Exemplo de Entrada

```
2
1 2
7
1 2
2 5
7 4
7 2
4 6
3 4
1
0 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. A segunda linha deve conter o número da vila na qual deve ser instalado o Posto de Vacinação. 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

Teste 2
7

Teste 3
1
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq N \leq 100$ (N = 0 apenas para indicar o final da entrada)

Sorvete

Arquivo fonte: sorvete.c, sorvete.cc, sorvete.cpp ou sorvete.pas

Joãozinho é um menino que costuma ir à praia todos os finais de semana com seus pais. Eles freqüentam sempre a mesma praia, mas cada semana o pai de Joãozinho estaciona o carro em um local diferente ao longo da praia, e instala sua família em um ponto na praia em frente ao carro. Joãozinho é muito comilão, e adora de tomar sorvete na praia. Contudo, alguns dias acontece de nenhum sorveteiro passar pelo local onde eles estão. Intrigado com isto, e não querendo mais ficar sem tomar seu sorvete semanal, Joãozinho foi até a Associação dos Sorveteiros da Praia (ASP), onde ficou sabendo que cada sorveteiro passa o dia percorrendo uma mesma região da praia, indo e voltando. Além disto, cada sorveteiro percorre todos os dias a mesma região. Jõaozinho conseguiu ainda a informação dos pontos de início e fim da região percorrida por cada um dos sorveteiros.

Com base nestes dados, Joãozinho quer descobrir os locais da praia onde o pai dele deve parar o carro, de forma que pelo menos um sorveteiro passe naquele local. Só que o volume de dados é muito grande, e Joãozinho está pensando se seria possível utilizar o computador para ajudá-lo nesta tarefa. No entanto Joãozinho não sabe programar, e está pedindo a sua ajuda.

1. Tarefa

Você deve escrever um programa que leia os dados obtidos pelo Joãozinho e imprima uma lista de intervalos da praia por onde passa pelo menos um sorveteiro.

2. Entrada

Seu programa deve ler vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois inteiros não negativos, *P e S*, que indicam respectivamente o comprimento em metros da praia e o número de sorveteiros. Seguem-se *S* linhas, cada uma contendo dois números inteiros *U* e *V* que descrevem o intervalo de trabalho de cada um dos sorveteiros, em metros contados a partir do início da praia ($U < V$, $0 \leq U \leq P$ e $0 \leq V \leq P$). O final da entrada é indicado por *S=0 e P=0*.

Exemplo de Entrada

```
200 2
0 21
110 180
1000 3
10 400
80 200
400 1000
10 2
1 4
5 6
0 0
```

3. Saída

Para cada conjunto de teste da entrada seu programa deve produzir uma lista dos intervalos da praia que são servidos por pelo menos um sorveteiro. 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. Cada intervalo da lista deve aparecer em uma linha separada, sendo descrito por dois números inteiros *U* e *V*, representando respectivamente o início e o final do intervalo (*U* < *V*). O final da lista de intervalos deve ser indicado por uma linha em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
0 21
110 180

Teste 2
10 1000

Teste 3
1 4
5 6
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq P \leq 10000$ ($P = 0$ apenas para indicar o final da entrada)
$0 \leq S \leq 5000$ ($S = 0$ apenas para indicar o final da entrada)
$0 \leq U < V \leq P$

Calculando

Arquivo fonte: calcula.c, calcula.cc, calcula.cpp ou calcula.pas

A disseminação dos computadores se deve principalmente à capacidade de eles se comportarem como outras máquinas, vindo a substituir muitas destas. Esta flexibilidade é possível porque podemos alterar a funcionalidade de um computador, de modo que ele opere da forma que desejarmos: essa é a base do que chamamos programação.

1. Tarefa

Sua tarefa é escrever um programa que faça com que o computador opere como uma calculadora simples. O seu programa deve ler expressões aritméticas e produzir como saída o valor dessas expressões, como uma calculadora faria. O programa deve implementar apenas um subconjunto reduzido das operações disponíveis em uma calculadora: somas e subtrações.

2. Entrada

A entrada é composta de vários conjuntos de testes. A primeira linha de um conjunto de testes contém um número inteiro m ($1 \leq m \leq 100$), indicando o número de operandos da expressão a ser avaliada. A segunda linha de um conjunto de testes contém a expressão aritmética a ser avaliada, no seguinte formato:

$X_1\ s_1\ X_2\ s_2\ ...\ X_{m-1}\ s_{m-1}\ X_m$

onde

- X_i , $1 \leq i \leq m$, é um *operando* ($0 \leq X_i \leq 100$);
- s_j, $1 \leq j < m$, é um *operador*, representado pelos símbolos ‘+’ ou ‘–’;
- não há espaços em branco entre operandos e operadores.

O final da entrada é indicado pelo valor m=0.

Exemplo de Entrada

```
3
3+7-22
3
5-10-77
10
1+2+3+4+5+6+7+8+9+10
0
```

3. Saída

Para cada conjunto de testes da entrada seu programa deve produzir três linhas. A primeira linha deve conter um identificador da expressão, no formato “Teste *n*”, onde *n* é numerado a partir de 1. Na segunda linha deve aparecer o resultado encontrado pelo seu programa. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigoro-

samente.

Exemplo de Saída

```
Teste 1
-12

Teste 2
-82

Teste 3
55
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$1 \leq m \leq 100$
$0 \leq X_i \leq 100$ para todo $1 \leq i \leq m$