OBI

OLIMPÍADA BRASILEIRA
DE INFORMÁTICA

CADERNO DE TAREFAS

SEGUNDA FASE • 12/9/99 • 13:00 às 17:00

Instruções:

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. Todas as tarefas têm o mesmo valor na correção.
3. As tarefas não estão ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões mais fáceis.
4. Preste muita atenção no nome dos arquivos de entrada e de saída indicados nas tarefas.
5. Para submeter uma solução, copie o arquivo executável e o arquivo fonte para o seu diretório de trabalho ou disquete, conforme especificado pelo seu professor. Não serão consideradas submissões em que um destes dois arquivos esteja faltando.

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

Problema A

RoboCoffee

(arquivo de entrada: ROBO.IN, arquivo de saída: ROBO.OUT)

Dona Mercedes, a eficiente funcionária responsável pela distribuição de cafezinho na empresa RoboCamp, fabricante de robôs industriais, vai aposentar-se em breve. Para ocupar a função de Dona Mercedes os engenheiros da empresa adaptaram um robô existente, ao qual acoplaram uma máquina de café. Para diminuir os custos de fabricação, o novo robô tem uma operação bem simples, sendo capaz de apenas três movimentos básicos: ficar parado (para que os empregados possam servir-se de café), andar para a frente, e girar sobre o seu eixo (para colocar-se de frente para o seu próximo ponto destino).

A programação do robô é feita através de uma seqüência de N pontos no plano, numerados de 0 a N-1 (o robô desenvolvido não sobe escadas, mas felizmente a empresa ocupa um único pavimento). Cada ponto é determinado por coordenadas inteiras (X,Y). O ponto i é sempre distinto do ponto i+1, para $0 \le i < N$, e o ponto N-1 é distinto do ponto 0 (veja a regra 4 abaixo). O robô move-se a partir do ponto 0, através de todos os pontos dados, observando as seguintes regras:
1) O robô inicia no ponto 0 de frente para o ponto 1;
2) O robô move-se sempre para a frente;
3) Chegando ao ponto i ($0 \le i \le N$-1) o robô gira sobre seu eixo, no sentido do relógio, de um ângulo α ($0^o \le \alpha < 360^o$), de modo a ficar de frente para o ponto (i+1) *mod* N, e faz uma pausa para que os usuários possam servir-se de café;
4) No final, para completar o percurso, o robô movimenta-se do ponto N-1 para o ponto 0, e gira de modo a ficar de frente para o ponto 1.

1. Tarefa

Sua tarefa é escrever um programa que, dada a seqüência de N pontos no plano que corresponde à programação do robô, determina quantas voltas completas sobre o seu eixo o robô perfaz durante seu percurso.

2. Entrada de Dados

O arquivo ROBO.IN contém vários conjuntos de teste. Cada conjunto de teste corresponde a uma programação do robô. A primeira linha de um conjunto de testes contém um inteiro positivo, N, que indica o número de pontos presentes no conjunto de teste. As N linhas seguintes contêm dois inteiros cada, correspondendo ao valor das coordenadas X e Y de um ponto, separados por espaço em branco ($-1000 \le X \le 1000$, $-1000 \le Y \le 1000$).

O final do arquivo de testes é indicado quando $N = 0$ (este último conjunto de teste não é válido e não deve ser processado).

O arquivo ROBO.IN contém ao menos um conjunto de teste que deve ser processado.

Exemplo de Entrada

```
4
1 1
1 0
0 0
0 1
4
2 -3
2 2
-3 3
-2 -1
0
```

3. Saída de Dados

Seu programa deve produzir um arquivo de saída chamado ROBO.OUT. Para cada conjunto de teste do arquivo de entrada seu programa deve produzir três linhas no arquivo de 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 o número de rotações completas efetuadas pelo robô durante seu percurso, encontrado pelo seu programa, precedido de “rotacoes:”. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente (note que não são usados acentos).

Exemplo de Saída

```
Teste 1
rotacoes: 1

Teste 2
rotacoes: 3
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$1 \leq N \leq 15.000$

Problema B

Restaurante

(arquivo de entrada: REST.IN, arquivo de saída: REST.OUT)

A UNICOMP (Universidade Independente de Computação) possui vários refeitórios que servem seus milhares de alunos e professores. Para melhorar o atendimento a UNICOMP planeja fazer uma reforma nos refeitórios, mas para isso necessita saber qual o número máximo de pessoas que são atendidas simultaneamente em um mesmo refeitório. Para isso a UNICOMP, que possui catracas eletrônicas, coletou os seguintes dados:

- Um vetor *E*, ordenado crescentemente, em que *E[i]* representa o instante de tempo em que a pessoa *i* entrou no restaurante;
- Um vetor *S*, em que *S[i]* representa o instante de tempo em que a pessoa *i* saiu do restaurante.

Os elementos de *E* e *S* são inteiros positivos que indicam o número de segundos transcorridos desde a abertura do restaurante. A entrada e a saída do restaurante se faz por uma única catraca, onde passa apenas uma pessoa por vez, de maneira que os tempos registrados em *E* e *S* são todos distintos.

1. Tarefa

Sua tarefa é escrever um programa que, dados dois vetores de inteiros *E* e *S*, ambos de comprimento igual a *N*, calcula o número máximo de pessoas que estão presentes ao mesmo tempo dentro do restaurante.

2. Entrada de Dados

O arquivo REST.IN contém vários conjuntos de teste. Cada conjunto de teste é composto por três linhas. A primeira linha contém um inteiro positivo, *N*, que indica o comprimento dos vetores *E* e *S*, conforme descrito acima. A segunda linha do conjunto de teste contém os elementos do vetor *E*, separados por espaço em branco, e a terceira linha contém os elementos do vetor *S*, separados por espaço em branco. O final do arquivo de testes é indicado quando $N = 0$ (este último conjunto de testes não é válido e não deve ser processado).

O arquivo REST.IN contém ao menos um conjunto de teste que deve ser processado.

Exemplo de Entrada

```
3
14 67 98
1890 1900 2123
2
200 1800
1543 2324
0
```

3. Saída de Dados

Seu programa deve produzir um arquivo de saída chamado REST.OUT. Para cada conjunto de teste do arquivo de entrada seu programa deve produzir três linhas no arquivo de 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 o número máximo de pessoas presentes simultaneamente no restaurante, encontrado pelo seu programa, precedido por "pessoas:". 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
pessoas: 3

Teste 2
pessoas: 1
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$1 \leq N \leq 5.000$
$1 \leq E[i] \leq 15.000$, $1 \leq i \leq N$
$1 \leq S[i] \leq 15.000$, $1 \leq i \leq N$
$E[i] < E[i+i]$, para $1 \leq i < N$
$E[i] < S[i]$, para $1 \leq i \leq N$
$E[i] \neq S[j]$, para todo par i e j, $1 \leq i \leq N$, $1 \leq j \leq N$
$N = 0$ apenas para indicar o fim do arquivo de entrada

Problema C

Jogo de Búzios

(arquivo de entrada: JOGO.IN, arquivo de saída: JOGO.OUT)

Búzios são pequenas conchas marinhas utilizadas em jogos esotéricos de predição do futuro. O jogo que nos interessa não é esotérico: ele era jogado, apenas por diversão, por jangadeiros de uma bonita praia do Rio de Janeiro. O jogo é jogado por N jangadeiros sentados ao redor de um círculo. Os jangadeiros são numerados de 1 a N de maneira que o jangadeiro (i+1) está sentado à direita do jangadeiro i, para $1 \leq i < N$. O jogador *1* está sentado à direita do jangadeiro N.

No início do jogo cada jangadeiro tem um búzio, exceto o jangadeiro K, que tem dois búzios. Em cada movimento, apenas um jangadeiro participa do jogo (chamado jogador *da vez*). O jogador da vez no primeiro movimento (t =1) é sempre o jangadeiro 1.

O jangadeiro da vez dá um ou dois búzios para o jangadeiro sentado à sua direita. A decisão de dar um ou dois búzios depende do turno: em turnos ímpares (t = 1, 3, 5...) o jangadeiro deve dar um búzio; em turnos pares (t = 2, 4, 6...) o jangadeiro deve dar dois búzios. Se ao terminar seu movimento o jangadeiro ficar sem búzios, ele abandona o jogo.

Nós vamos considerar que o jogo termina quando restar apenas um jangadeiro, que é declarado o vencedor do jogo. Por exemplo, para N=5 e K=3, o jogo termina em 10 movimentos, restando ao final apenas o jangadeiro 5, que é declarado vencedor. Note que é possível um jogo chegar a um estado em que, não importa o número de movimentos jogados, o número de jangadeiros não diminui (por exemplo, para N=7 e K=2), mas não vamos considerar estes casos: os jangadeiros utilizam apenas as combinações de N e K que garantidamente levam a um vencedor.

1. Tarefa

Você deve escrever um programa que, dados N e K, determine de quantos movimentos o jogo foi composto e qual foi o jangadeiro que venceu o jogo.

2. Entrada de Dados

O arquivo JOGO.IN contém vários conjuntos de teste. Cada conjunto de teste é composto por uma única linha, que contém dois inteiros positivos, N e K, correspondendo respectivamente ao número de participantes do jogo e o índice ($1 \leq K \leq N$) do participante que inicia o jogo com dois búzios, conforme descrito acima. O final do arquivo de testes é indicado quando $N = K = 0$ (este último conjunto de testes não é válido e não deve ser processado).

Exemplo de Entrada

```
3 3
5 3
0 0
```

3. Saída de Dados

Seu programa deve produzir um arquivo de saída chamado JOGO.OUT. Para cada conjunto de teste do arquivo de entrada seu programa deve produzir quatro linhas no arquivo de 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 o número de turnos que compuseram o jogo, precedido de "turnos:". Na terceira linha deve aparecer o índice do jangadeiro que venceu o jogo, precedido de "vencedor:". A quarta linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente.

Exemplo de Saída

```
Teste 1
turnos: 2
vencedor: 3

Teste 2
turnos: 10
vencedor: 5
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$2 \leq N \leq 15.000$
$1 \leq K \leq 15.000$
$N = 0$ e $K = 0$ apenas para indicar o fim do arquivo de entrada

Problema D

Seqüências

(arquivo de entrada: SEQ.IN, arquivo de saída: SEQ.OUT)

Uma seqüência de 0's e 1's é chamada de seqüência-H se é composta por um único 0 ou se é composta por um 1 seguido de duas seqüências-H. Por exemplo, 0, 100 e 10100 são seqüências-H, mas 10, 111 e 10010 não são.

1. Tarefa

Sua tarefa é determinar se uma dada seqüência é ou não uma seqüência-H.

2. Entrada de Dados

O arquivo SEQ.IN contém vários conjuntos de teste. Cada conjunto de teste é composto por uma única linha, que contém a seqüência a ser testada, composta de dígitos 0's e 1's (sem espaços em branco intermediários). O final da seqüência é indicado pelo caractere '#'. O final do arquivo de testes é indicado quando o primeiro caractere da linha de teste é '#' (esta última seqüência de teste não é válida e não deve ser processada).

Exemplo de Entrada

```
0#
10100#
10010#
#
```

3. Saída de Dados

Seu programa deve produzir um arquivo de saída chamado SEQ.OUT. Para cada conjunto de teste do arquivo de entrada seu programa deve produzir três linhas no arquivo de 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 palavra "sim" se a seqüência é uma seqüência-H, ou a palavra "nao" se a seqüência não é uma seqüência-H. A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente (note que não são usados acentos).

Exemplo de Saída

```
Teste 1
sim

Teste 2
sim

Teste 3
nao
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

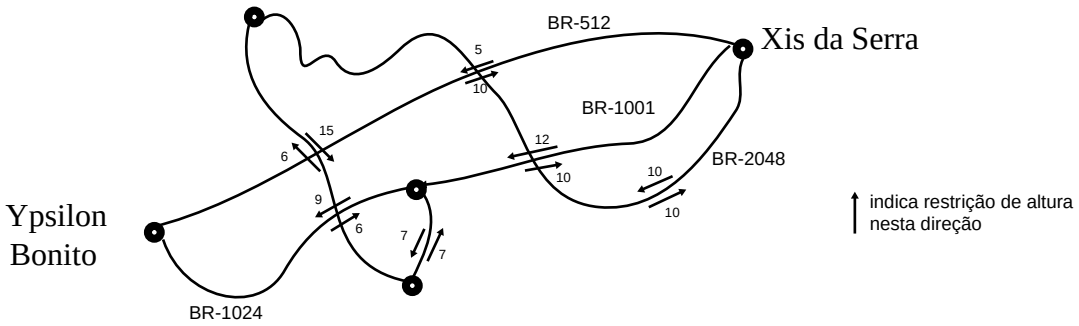
Não há restrição no comprimento da seqüência a ser testada.

Problema E

Carga Pesada

(arquivo de entrada: CARGA.IN, arquivo de saída: CARGA.OUT)

Um dos maiores problemas no transporte de cargas pesadas por rodovias é a altura dos viadutos, pois muitas vezes a carga é mais alta do que o vão do viaduto sob o qual o caminhão deve passar. Considere o mapa abaixo. É possível tranportar uma turbina de hidroelétrica, que em cima do caminhão mede 7 metros de altura, da cidade Xis da Serra para a cidade Ypsilon Bonito?



Neste caso é fácil perceber que a resposta é sim, pois o menor vão encontrado no caminho, se utilizarmos as estradas BR-1001 e BR-1024, é de 9 metros. No caso geral, com dezenas de cidades e estradas, a resposta pode não ser tão evidente.

1. Tarefa

Sua tarefa é determinar, para um dado par de cidades *X* e *Y*, qual a carga mais alta que pode ser transportada de *X* para *Y* por meio rodoviário, conhecendo todas as estradas da região e a altura de todos os túneis e viadutos dessas estradas. Considere que todas as estradas têm limitação de altura e que a interligação das estradas ao redor das cidades não tem restrição de altura.

2. Entrada de Dados

O arquivo CARGA.IN contém vários conjuntos de teste. O número máximo de cidades, *N*, em cada teste é *100*. As cidades são numeradas de 1 a *N*. A primeira linha do conjunto de teste contém dois inteiros positivos X e Y que representam respectivamente as cidades origem e a cidade destino da carga ($1 \le X \le N$, $1 \le Y \le N$). As linhas seguintes contêm cada uma a descrição de uma estrada. Cada descrição é composta por três inteiros *A*, *B* e *C*, representando respectivamente a cidade onde a estrada inicia, a cidade onde a estrada termina e a altura do viaduto ou túnel mais baixo no trajeto de *A* para *B*. O final da descrição das estradas de um teste é indicado por $A = B = C = 0$. Em cada conjunto de teste, sempre há ao menos um caminho de *X* para *Y*. O final do arquivo de testes é indicado quando $X = Y = 0$ (este último conjunto de testes não é válido e não deve ser processado).

Exemplo de Entrada

```
2 4
1 4 5
2 4 12
0 0 0
1 3
1 2 10
1 3 8
2 3 12
3 1 5
0 0 0
0 0
```

3. Saída de Dados

Seu programa deve produzir um arquivo de saída chamado CARGA.OUT. Para cada conjunto de teste do arquivo de entrada seu programa deve produzir três linhas no arquivo de 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 altura máxima da carga, encontrada pelo seu programa, precedida de "altura maxima:". A terceira linha deve ser deixada em branco. A grafia mostrada no Exemplo de Saída, abaixo, deve ser seguida rigorosamente (note que não são usados acentos).

Exemplo de Saída

```
Teste 1
altura maxima: 12

Teste 2
altura maxima: 10
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$1 \leq X \leq 100$
$1 \leq Y \leq 100$
$1 \leq A \leq 100$
$1 \leq B \leq 100$
$1 \leq C \leq 50$
A = 0, B = 0 e C = 0 apenas para indicar o fim da descrição das estradas do mapa
X = 0 e Y = 0 apenas para indicar o fim do arquivo de entrada