OBI

OLIMPÍADA BRASILEIRA
DE INFORMÁTICA

OBI2002

Curso de Programação

CADERNO DE TAREFAS

13/7/2002 • 8:00 às 13:00

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

Passeio de bicicleta

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

Em 2001, Arnaldinho foi um dos alunos que representou o seu país na Olimpíada Internacional de Informática (IOI), que ocorreu na cidade de Tampere, na Finlândia. Além de participar da IOI, Arnaldinho queria aproveitar a oportunidade para conhecer aquela cidade. Ao chegar em Tampere, Arnaldinho surpreendeu-se com um fato que talvez poucos saibam: na Finlândia, a bicicleta é o meio de transporte mais utilizado (no verão; no inverno, com temperaturas de até -30^{o}, é impossível andar de bicicleta sem congelar). A IOI estava acontecendo no verão do hemisfério norte e Arnaldinho, que adora andar de bicicleta, decidiu alugar uma para fazer um passeio pelos principais pontos turísticos de Tampere.

Por azar de Arnaldinho, quando ele chegou ao primeiro dos pontos turísticos que ele foi visitar, a correia da bicicleta quebrou. Enquanto visitava este ponto turístico, Arnaldinho decidiu que continuaria o seu passeio mesmo com a correia da bicicleta quebrada, valendo-se das elevações da cidade. Arnaldinho então pegou um mapa com os pontos turísticos de Tampere e as ligações que os conectam. Para felicidade de Arnaldinho, como muitas pessoas passeiam por Tampere de bicicleta, o mapa trazia a altitude dos locais onde se situam todos os pontos turísticos. Além disto, o mapa mencionava que em Tampere, se um ponto turístico A está localizado em um local com altitude maior do que um outro ponto B, e existe ligação direta de A para B, então todo o percurso ao longo desta ligação é em descida. Arnaldinho quer visitar o maior número possível de pontos turísticos sendo que, como a sua bicicleta está estragada, depois de visitar um ponto turístico A ele só pode ir para outro ponto B que tenha altitude menor que A. Além disto, a Finlândia é um país muito civilizado, e Arnaldinho deve respeitar o sentido das ligações entre os pontos turísticos de Tampere.

1. Tarefa

A tua tarefa é ajudar Arnaldinho a encontrar, a partir do mapa com os pontos turísticos de Tampere e as ligações entre estes pontos, o número máximo de pontos turísticos que ele pode visitar, dado o ponto onde ele se encontra e a restrição de que, a partir de um ponto, ele só pode ir para outro com menor altitude.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém três números inteiros *P, L* e *I*, correspondendo, respectivamente, ao número de pontos turísticos, o número de ligações entre eles, e o número do ponto turístico onde Arnaldinho se encontra inicialmente. Os pontos turísticos são numerados seqüencialmente de 1 até *P*. A segunda linha contém *P* números inteiros, correspondendo às altitudes, em metros, dos locais onde se encontram os pontos turísticos de 1 a *P*, respectivamente. Cada uma das *L* linhas seguintes contém dois inteiros *A* e *B*, indicando que há uma ligação direta partindo do ponto turístico *A* até o ponto *B*. O final da entrada é indicado por um conjunto de teste com $P = L = I = 0$.

Exemplo de Entrada

```
4 7 2
100 150 50 200
1 2
2 1
```

```
2 4
4 3
4 2
3 2
3 1
4 7 4
100 50 150 200
1 2
2 1
2 4
4 3
4 2
3 2
3 1
0 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 o número máximo de pontos turísticos que Arnaldinho ainda pode visitar (sem contar o ponto turístico onde ele está inicialmente). 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
3
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

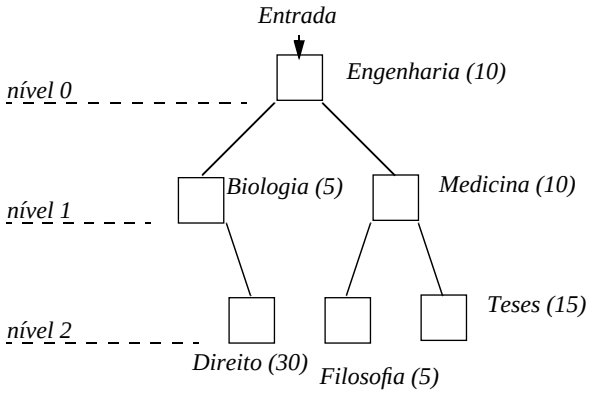
$0 \leq P \leq 150$ ($P = 0$ apenas para indicar o fim da entrada)
$0 \leq L \leq P*(P-1)$ ($L = 0$ apenas para indicar o fim da entrada)
$0 \leq I \leq P$ (I = 0 apenas para indicar o fim da entrada)
$0 \leq H \leq 1000$
$1 \leq A \leq$ P
$1 \leq B \leq$ P
$A \neq B$

Biblioteca Ótima

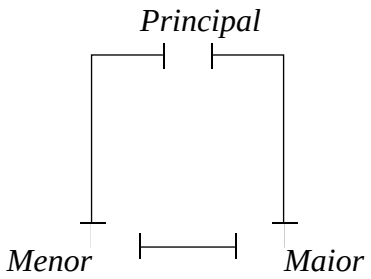
arquivo fonte: biblio.pas, biblio.c, biblio.cc ou biblio.cpp

O arquiteto Otávio B. Ignácio, da firma OBI Arquitetura Ltda, está criando um novo modelo para o projeto arquitetural de bibliotecas. Ele constatou que um visitante poderia ter que andar por toda a biblioteca procurando pela seção à qual pertence o livro desejado caso não houvesse alguma ordem na localização das seções (encontrar o livro dentro da seção é mais fácil porque eles normalmente estão ordenados). O novo modelo procura resolver o problema, de forma que um visiante possa ir mais diretamente para a seção desejada, sem ter que passar por toda a biblioteca.

Pela sua idéia, cada seção está associada a uma única sala e vice-versa. As salas possuem até três portas com os nomes de *Principal*, *Menor* e *Maior*, conforme a figura abaixo, e as portas *Menor* e *Maior* de uma sala estão sempre associadas à porta *Principal* de uma outra sala. Além disso, seu modelo segue a seguinte propriedade: toda seção que se pode chegar através da porta *Menor* de uma determinada sala possui nome necessariamente menor do que o nome da seção atual; da mesma forma, toda seção que se pode chegar através da porta *Maior* possui nome necessariamente maior. A ordem utilizada para comparar os nomes das seções pode ser a mesma ordem utilizada em dicionários (ordem lexicográfica). Note que toda sala tem uma porta *Principal*, mas a existência das portas *Menor* e *Maior* depende da existência de seções adjacentes.



Projeto de uma biblioteca



Configuração de uma sala

Desta forma, quando um visitante entra em uma sala por sua porta *Principal*, ele compara o nome da seção desejada com o nome da seção correspondente à sala. Se o nome da seção desejada for menor, o visitante segue seu caminho pela porta de nome *Menor*; se for maior, segue pela porta de nome *Maior*. Obviamente, se os dois nomes foram iguais, significa que ele encontrou a seção desejada.

Um amigo bibliotecário sugeriu que as seções mais procuradas deveriam ficar mais próximas da entrada principal, de forma a diminuir a distância média necessária para uma pessoa encontrar a seção desejada. No entanto, Otávio sabia que seu modelo de busca restringia a topologia não podendo-se simplesmente colocar as seções mais visitadas próximas à entrada sem considerar sua ordem relativa. Utilizando-se do número de visitas de cada seção em um ano como custo de um nó, ele definiu que o custo total de uma topologia seria dado pela soma total do custo de cada nó multiplicado pelo número de seções intermediárias no caminho desde a entrada da biblioteca até a sala desejada. Este último valor é equivalente ao nível de uma sala, conforme mostrado na figura. Sendo assim o custo da topologia adotada na figura é 115. Substituindo a seção de Biologia pela seção de Direito e fazendo a primeira acessível pela porta *Menor* da segunda geraria uma outra topologia de custo 90, para o mesmo exemplo de biblioteca.

Otávio concluiu que a topologia de custo total mínimo representa a melhor distribuição de seções em uma biblioteca do seu modelo. No entanto, Otávio calculou manualmente este valor para projetos pequenos e não sabe como resolver o problema em projetos maiores.

1. Tarefa

Você foi contratado para desenvolver um programa que calcule o custo total mínimo de uma biblioteca dentre todas as topologias possíveis, dadas as freqüências de acesso às suas seções.

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um número inteiro *N* que indica o número de seções da biblioteca. As seções são identificadas por inteiros de 1 a *N*. A segunda linha do conjunto de teste contém *N* inteiros entre 0 e 100, representando as freqüências de acesso das seções 1, 2, 3,... e *N*, respectivamente. O final da entrada é indicado por *N* = 0.

Exemplo de Entrada

```
1
5
3
10 10 10
3
5 10 20
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 custo total mínimo para a topologia indicada, calculado 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
0

Teste 2
20

Teste 3
20
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq N \leq 60$ ($N = 0$ apenas para indicar o fim da entrada)
$0 \leq Freqüências \leq 100$

Penta!

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

O técnico Luiz Felipe Scolari (Felipão) foi um dos grandes responsáveis pela conquista do quinto título brasileiro na Copa do Mundo. Contrariando muitos críticos, alterou o esquema tático do time, apostou em jogadores que não tinham garantida uma boa condição física para o torneio, e administrou com um misto de rigor e paternalismo as personalidades muitas vezes difíceis dos jogadores. Estrategista competente, Felipão planejou meticulosa e longamente toda a campanha.

Felipão planejou até mesmo o desfile em carro aberto do Corpo de Bombeiros, na volta da seleção ao país. Tendo recebido a informação que o local mais alto do carro possibilitava apenas a permanência de um número limitado de jogadores, e sabendo que os jogadores iriam querer estar o maior tempo possível no palanque, Felipão determinou, para cada quarteirão do percurso, qual jogador necessariamente deveria estar presente no palanque. No primeiro quarteirão Beletti deveria estar presente no palanque, no segundo quarteirão Cafu, no terceiro quarteirão Denílson, no quarto Edmilson, e assim por diante. No entanto, ele não determinou qual jogador presente ao palanque deveria descer para dar lugar ao novo jogador, quando o palanque estivesse com sua lotação máxima. Como subir e descer do palanque é cansativo (e mesmo um tanto arriscado, considerando a altura), os jogadores decidiram procurar a sua ajuda para descobrir como obedecer às ordens do técnico quanto à permanência do jogador no palanque no momento estipulado, mas de forma a minimizar a troca de jogadores no palanque.

1. Tarefa

Com o dinheiro recebido como prêmio pela conquista da Copa do Mundo os jogadores decidiram contratar os seus serviços. Você deve escrever um programa que determine o número mínimo de *trocas* de jogadores no palanque, conhecendo a lista formulada por Felipão.

Considere que inicialmente o palanque está vazio, e, até chegar à sua lotação máxima, um novo jogador sobe sem que nenhum outro desça (ou seja, não ocorre *troca*). Quando o palanque está com lotação máxima, antes que um jogador possa subir um outro deve descer (caracterizando uma *troca*). Os jogadores sempre desejam permanecer no palanque o maior tempo possível (ou seja, nenhum desce sem que haja necessidade de um novo jogador ascender ao palanque).

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém dois números inteiros N e P, que indicam respectivamente o número de quarteirões do trajeto e o número de jogadores que cabem simultaneamente no palanque do Carro de Bombeiros. Os jogadores são identificados por inteiros de 1 a 23. As N linhas seguintes contêm cada uma um inteiro positivo J, indicando o jogador que deve estar presente no palanque no N-ésimo quarteirão do trajeto. O final da entrada é indicado quando $N = P = 0$.

Exemplo de Entrada

```
4 2
1
2
3
1
```

```
13 3
10
23
10
11
7
8
23
11
9
10
5
7
11
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 o número mínimo de trocas de jogadores, 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
1

Teste 2
6
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$0 \leq N \leq 10000$ ($N = 0$ apenas para indicar o fim da entrada)
$0 \leq P \leq N$ ($P = 0$ apenas para indicar o fim da entrada)
$1 \leq J \leq 23$

Dominó

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

Um jogo padrão de peças de dominó contém 28 peças, cada uma mostrando dois números de 0 (branco) a 6, com a representação de pontos utilizada normalmente em dados. As 28 peças, que são únicas, consistem das seguintes combinações de números:

Peça	Valores	Peça	Valores	Peça	Valores	Peça	Valores
1	0 | 0	8	1 | 1	15	2 | 3	22	3 | 6
2	0 | 1	9	1 | 2	16	2 | 4	23	4 | 4
3	0 | 2	10	1 | 3	17	2 | 5	24	4 | 5
4	0 | 3	11	1 | 4	18	2 | 6	25	4 | 6
5	0 | 4	12	1 | 5	19	3 | 3	26	5 | 5
6	0 | 5	13	1 | 6	20	3 | 4	27	5 | 6
7	0 | 6	14	2 | 2	21	3 | 5	28	6 | 6

Utilizando todas as peças de um jogo é possível montar uma matriz de 7 x 8 valores. Cada uma dessas matrizes corresponde a pelo menos um *mapa* de dominós. Um mapa consiste de uma matriz 7 x 8 onde os valores são substituídos pelo identificador da peça contendo aqueles valores. Um exemplo de matriz e o mapa correspondente é mostrado abaixo.

Matriz de valores							
6	6	2	6	5	2	4	1
1	3	2	0	1	0	3	4
1	3	2	4	6	6	5	4
1	0	4	3	2	1	1	2
5	1	3	6	0	4	5	5
5	5	4	0	2	6	0	3
6	0	5	3	4	2	0	3

Mapa de peças							
28	28	14	7	17	17	11	11
10	10	14	7	2	2	21	23
8	4	16	25	25	13	21	23
8	4	16	15	15	13	9	9
12	12	22	22	5	5	26	26
27	24	24	3	3	18	1	19
27	6	6	20	20	18	1	19

1. Tarefa

Escreva um programa que analise uma matriz de números e verifique de quantas maneiras ela pode ser montada utilizando um jogo completo de peças de dominó (note que é possível que a matriz não represente um arranjo possível das peças).

2. Entrada

A entrada é composta de vários conjuntos de teste. A primeira linha de um conjunto de teste contém um inteiro *N* que indica o número de conjuntos de teste. Cada conjunto de teste consiste de sete linhas com oito inteiros *D* (entre 0 e 6) cada, representando o padrão observado de valores. Não há linhas separando os conjuntos de teste.

Exemplo de Entrada

```
2
5 4 3 6 5 3 4 6
0 6 0 1 2 3 1 1
3 2 6 5 0 4 2 0
5 3 6 2 3 2 0 6
4 0 4 1 0 0 4 1
```

```
5 2 2 4 4 1 6 5
5 5 3 6 1 2 3 1
4 2 5 2 6 3 5 4
5 0 4 3 1 4 1 1
1 2 3 0 2 2 2 2
1 4 0 1 3 5 6 5
4 0 6 0 3 6 6 5
4 0 1 6 4 0 3 0
6 5 3 6 2 1 5 3
```

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 o número de mapas correspondentes à matriz dada, 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
1

Teste 2
2
```

(esta saída corresponde ao exemplo de entrada acima)

4. Restrições

$1 \leq N \leq 50$
$0 \leq D \leq 6$