OBI2007

Caderno de Tarefas

Modalidade Programação • Nível 2, Fase 1

17 de Março de 2007

A PROVA TEM DURAÇÃO DE CINCO HORAS

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 8 páginas (não contando esta folha de rosto), numeradas de 1 a 8. Verifique se o caderno está completo.
- A prova deve ser feita individualmente.
- É proibido consultar a Internet, livros, anotações ou qualquer outro material durante a prova. É permitida a consulta ao *help* do ambiente de programação se este estiver disponível.
- As tarefas têm o mesmo valor na correção.
- A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa.
- Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc.), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
- As tarefas **não** estão ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões mais fáceis.
- 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*. Para problemas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada problema.
- 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.
- 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*.
- 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
www.sbc.org.br
Fundação Carlos Chagas
www.fcc.org.br

Chocolate

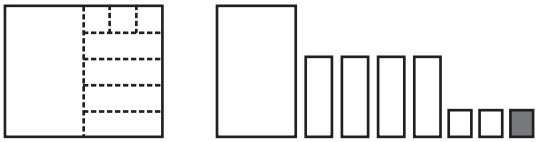
Nome do arquivo fonte: `choc.c`, `choc.cpp`, *ou* `choc.pas`

Juliana é uma famosa doceira reconhecida internacionalmente pelos seus bombons, exportados para todo o mundo. Embora não revele a ninguém as suas receitas, ela já deu entrevistas contando alguns de seus segredos. Sua fábrica de bombons utiliza somente chocolates comprados de um único produtor suíço, que envia barras gigantescas que são cortadas por grandes máquinas.

Dada uma barra grande de chocolate, Juliana realiza divisões sucessivas da barra até obter uma barra que contém a quantidade exata de chocolate para aquela receita. Após cada divisão, ela seleciona um dos pedaços resultantes e armazena os demais para uso futuro. As divisões são determinadas por critérios técnicos relacionados ao tamanho das barras e aos equipamentos disponiveis em um dado momento.

Por exemplo, se ela deseja obter uma barra de 100g de chocolate a partir de uma barra de 3Kg, primeiro ela divide a barra ao meio. Em seguida, um dos pedaços é dividido em cinco partes iguais e por fim, um desses pedaços de 300g é dividido em 3 pedaços, resultando no pedaço de 100g necessário para a receita. Nesse processo, 1 pedaço é utilizado para a receita e 7 pedaços de diferentes tamanhos serão guardados para uso futuro. A figura abaixo ilustra esse cenário.

Barra original



Divisões na barra original e pedaços da barra dividida
O pedaço marcado será utilizado na receita

Tarefa

Dada uma sequência de divisões realizadas por Juliana em uma barra de chocolate, determinar quantos pedaços serão armazenados em estoque para uso futuro.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém um inteiro N que indica o número de divisões feitas na barra de chocolate original ($1 \leq N \leq 1.000$). A linha seguinte contém N inteiros I ($2 \leq I \leq 10$) representando o número de pedaços em que o pedaço atual foi dividido. Sempre que é feita uma divisão, um pedaço é utilizado para a próxima divisão e os demais são separados para serem armazenados em estoque.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha, contendo o número de pedaços de chocolate que serão armazenados em estoque.

Entrada	**Entrada**	**Entrada**
3 2 3 5	5 2 2 2 3 3	7 2 3 4 5 6 7 8
Saída	**Saída**	**Saída**
7	7	28

Repositórios

Nome do arquivo fonte: `repos.c`, `repos.cpp`, *ou* `repos.pas`

Uma das boas práticas ao administrar um conjunto de computadores é manter os aplicativoss sempre atualizados. Entretanto, em uma grande corporação com milhares de aplicativos instalados, a simples verificação do que precisa ser atualizado pode tornar-se uma tarefa bem complicada. Para facilitar isso, alguns fabricantes armazenam todos os aplicativos existentes em grandes bases de dados chamadas repositórios e um programa é responsável por verificar esse repositório e atualizar as versões dos aplicativos.

M.V.Lzr, um administrador de sistemas e rapper nas horas vagas, trabalha em uma empresa que, infelizmente, não utiliza um sistema com repositórios. Para facilitar sua vida, ele decidiu que era a hora de ter o seu próprio sistema e pediu a sua ajuda.

Periodicamente ele varre a Internet em busca das páginas que possam conter os aplicativos e constrói uma lista com as versões dos aplicativos que deseja instalar disponíveis em cada página. Um programa deve verificar então qual a versão de cada programa instalado nos computadores (todos eles possuem os mesmos aplicativos instalados e nas mesmas versões) e instalar todos aqueles que ainda não foram instalados ou cuja versão instalada seja anterior a versão mais recente. Como ele não sabe programar direito, ele pediu sua ajuda.

Tarefa

Dado uma lista de aplicativos instaladas nos computadores da empresa, com suas respectivas versões e uma lista de aplicativos disponíveis na internet que devem ser instalados, determinar quais devem ser instalados e em quais versões.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém dois inteiros C ($1 \leq C \leq 10.000$) e N ($1 \leq N \leq 1.000$) que representam o número total de aplicativos e versões disponíveis na internet e o número total de programas instalados na empresa, respectivamente. As C linhas seguintes possuem dois inteiros cada, P_c ($1 \leq P_c \leq 1.000.000.000$) e V_c ($1 \leq V_c \leq 1.000.000.000$), representando o número do programa e o número da versão instalada nos computadores. Todo aplicativo está instalado uma única vez em cada máquina e em uma única versão. Em seguida, as As N linhas seguintes possuem dois inteiros cada, P_n ($1 \leq P_n \leq 1.000.000.000$) e V_n ($1 \leq V_n \leq 1.000.000.000$), representando o número do programa e o número da versão disponível na internet. Um dado programa pode estar disponível em mais de uma versão na internet.

Saída

Seu programa deve imprimir, na *saída padrão*, diversas linhas, cada uma contendo dois inteiros, P_s e V_s com o número do programa e a versão que deve ser instalada. Em todo caso de teste existe pelo menos um programa que deve ser instalado.

Entrada	**Entrada**	**Entrada**
1 1 5215 1 5215 3 **Saída** 5215 3	3 2 1640 1 2540 4 1870 3 2540 1 1640 4 **Saída** 1640 4	2 5 2000 4 2001 5 2000 1 2001 4 2001 6 2000 2 2000 3 **Saída** 2001 6

Sacoleiro

Nome do arquivo fonte: `saco.c`, `saco.cpp`, *ou* `saco.pas`

Seu amigo sacoleiro pediu sua ajuda num problema que ele está enfrentando. Ele tem um mapa de cidades que ele já conhece e que são interessantes para ele, além das rotas entre as mesmas. Ele pretende fazer uma viagem para comprar presentes para seu filho e para sua filha. O problema é que nem todos os presentes têm o mesmo preço, alguns são obviamente mais caros que os outros, e ele não quer ser injusto dando presentes mais caros para um ou para outro. O objetivo é fazer com que diferença entre a soma dos valores dos presentes seja a menor possível (de preferência que sejam iguais, naturalmente). Há, também, um limite de quanto ele pode gastar na viagem.

O sacoleiro tem um mapa com N cidades e as rotas que as ligam. Além disso, cada cidade pertence ao grupo A ou ao grupo B. No grupo A estão as cidades em que há presentes para o filho, enquanto que no grupo B estão as cidades com presentes para a filha. Sempre que ele pára numa cidade ele pode comprar ou não o presente, **mesmo que ele já tenha estado lá antes**, inclusive pode comprar mais de uma unidade do mesmo presente (enquanto tiver dinheiro disponível, naturalmente). As cidades são numeradas de 0 a $N-1$. O trajeto deve sempre começa na cidade 0. O tamanho do percurso não importa para o sacoleiro. O total disponível de dinheiro para os presentes é T. O sacoleiro não pode terminar a viagem sem ter comprado pelo menos um presente para algum dos filhos.

Tarefa

Escreva um programa que, dadas N cidades, as rotas entre elas e os valores de presentes de cada cidade, retorne qual a diferença mínima possível entre a soma dos presentes do grupo A e a soma dos presentes do grupo B.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém um inteiro N ($2 \leq N \leq 30$) que indica a quantidade de cidades. A segunda linha contém um inteiro T ($10 \leq T \leq 100$) que indica a quantidade de dinheiro que o sacoleiro tem para gastar. As N linhas seguintes contêm a descrição cada cidade. Cada uma dessas linhas tem o formato $XPCKV_0V_1...V_{K-1}$, onde X é um inteiro que representa a cidade (numeradas de 0 a $N-1$); P é um inteiro ($1 \leq P \leq 10$) que indica o valor do presente da cidade X; C é um caracter `A` ou `B`, indicando a que grupo a cidade X pertence; K é um inteiro ($0 \leq K < N$) que indica quantas rotas saem da cidade X; e cada V_i é um inteiro indicando um dos possíveis destinos a partir da cidade X. Note que as rotas **não** são bidirecionais. Uma cidade nunca terá rota para ela mesma e pode-se assumir que $i \neq j \Rightarrow V_i \neq V_j$.

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha com um inteiro representando a menor diferença possível de valores entre os presentes comprados para o grupo A e para o grupo B.

Entrada

```
4
20
0 9 A 2 1 2
1 8 B 1 2
2 7 A 1 3
3 6 B 1 1
```

Saída

```
1
```

Pastas

Nome do arquivo fonte: `pastas.c`, `pastas.cpp`, *ou* `pastas.pas`

Estela é uma secretária dedicada da OBI (Organização Burocrática Internacional), um megaconglomerado empresarial voltado a criação de documentos e preenchimento de formulários. Todo dia ela recebe milhares de pastas suspensas e seu objetivo é organizá-las de uma forma que seja simples recuperar uma pasta do arquivo.

Cada pasta possui uma pequena aba, que fica anexada à pasta e é visível quando a pasta está suspensa em seu arquivo. Todo funcionário fixa a aba em uma das posições especificadas pelo manual de fixação de abas, embora ele possa escolher, ao acaso, qualquer uma das posições descritas no manual. Tais posições são numeradas de 1 até P.

Estela notou que fica consideravelmente mais fácil encontrar as pastas se elas forem arquivadas da seguinte forma: primeiro uma pasta com aba na posição 1, depois uma com aba na posição 2, e assim sucessivamente, até que uma pasta com aba na posição P seja arquivada. Logo após, repete-se o processo, arquivando uma pasta com aba na posição 1. Para Estela, um conjunto de pastas é arquivado de forma perfeita se todas as pastas desse conjunto forem arquivadas da forma descrita anteriormente, ou seja:

- Imediatamente após toda pasta com aba na posição I, $I < P$, existe uma pasta com aba na posição $I + 1$ ou não há nenhuma pasta.
- Imediatamente após toda pasta com aba na posição P, existe uma pasta com aba na posição 1 ou não há nenhuma pasta.
- Todas as pastas do conjunto são armazenadas.

Tarefa

Dado um conjunto de pastas e a posição de suas abas, determinar se é possível arquivar esse conjunto de pastas de forma perfeita.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém dois inteiros P e N que indicam, respectivamente, o número de posições possíveis para se colar as abas ($1 \leq P \leq 1.000$) o número pastas a serem armazenadas ($1 \leq N \leq 1.000.000$). As N linhas seguintes contém um inteiro I ($1 \leq I \leq P$) cada representando a posição onde a aba da I-ésima pasta foi colada.

Saída

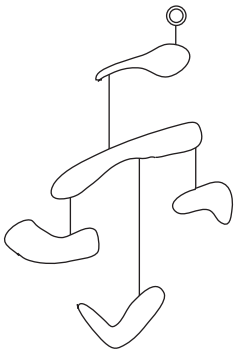
Seu programa deve imprimir, na *saída padrão*, uma única linha, contendo a letra **S** se for possível fazer um arquivamento perfeito ou **N** caso contrário

Entrada	**Entrada**	**Entrada**
2 2 1 2 **Saída** S	3 6 1 2 3 1 2 1 **Saída** N	4 7 1 1 2 2 3 3 4 **Saída** S

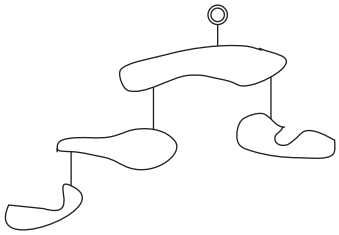
Móbile

Nome do arquivo fonte: `mobile.c`, `mobile.cpp`, *ou* `mobile.pas`

Móbiles são objetos muito populares hoje em dia, sendo encontrados até em berços, para diversão de bebês, mas foram concebidos há muito tempo (em 1931) pelo então jovem artista americano Alexander Calder como esculturas em movimento. Um móbile é uma estrutura composta de peças unidas por fios. O móbile é preso por um fio a uma argola pela qual ele é suspenso, permitindo que a estrutura movimente-se livremente. A argola é presa a uma única peça, chamada de peça-raiz do móbile. A peça-raiz pode ter zero ou mais sub-móbiles pendurados nela, cada sub-móbile sendo composto por uma peça-raiz na qual por sua vez podem estar pendurados zero ou mais sub-móbiles, e assim sucessivamente. Abaixo podemos ver dois exemplos de móbiles:



(a)



(b)

Victor é dono de uma fabrica de móbiles que emprega centenas de artesãos. Cada móbile produzido na fábrica é confeccionado por um artesão, que cria móbiles de acordo com o seu gosto pessoal, utilizando peças de formatos distintos. Entretanto, Victor tem notado que nem todos os seus artesãos possuem a mesma habilidade artística, de forma que às vezes o móbile produzido nem sempre é bem balanceado, segundo a sua concepção. Para Victor, um móbile é bem balanceado se, para cada peça, todos os sub-móbiles pendurados nela são compostos pelo mesmo número de peças. O número de peças de um sub-móbile é determinado contando-se o número de peças que o compõe, incluindo a sua peça-raiz. Note que cada peça do móbile, exceto a peça-raiz, é pendurada em exatamente uma outra peça.

Por exemplo, o móbile da figura (a) acima é um móbile bem balanceado: a peça-raiz possui um único sub-móbile, que por sua vez possui três sub-móbiles, todos com o mesmo número de peças (uma única). Já o móbile da figura (b) é um móbile mal balanceado: a peça-raiz possui dois sub-móbiles, um com o total de duas peças e outro com o total de uma peça.

Tarefa

Dada a descrição de um móbile, você deve escrever um programa para determinar se o móbile está bem balanceado ou não.

Entrada

A entrada contém um único conjunto de testes, que deve ser lido do *dispositivo de entrada padrão* (normalmente o teclado). A primeira linha da entrada contém um inteiro N que indica o número de peças utilizadas no móbile ($1 \leq N \leq 10.000$). As peças são identificadas por inteiros de 1 a N. Cada uma das N linhas seguintes contém dois números inteiros I e J, indicando que a peça de número I está pendurada na peça de número J (a peça raiz está pendurada na argola, que é identificada pelo o número 0).

Saída

Seu programa deve imprimir, na *saída padrão*, uma única linha, contendo a palavra `bem` se o móbile estiver bem balanceado ou `mal` caso esteja mal balanceado. A palavra deve ser escrita com todas as letras em minúsculas.

Entrada	**Entrada**	**Entrada**
3 1 0 2 1 3 2 **Saída** bem	5 1 0 2 1 4 2 3 2 5 2 **Saída** bem	7 2 0 1 2 3 1 4 3 5 4 6 4 7 5 **Saída** mal