acm International Collegiate Programming Contest 2005 IBM event sponsor

Maratona de Programação

Sociedade Brasileira de Computação

10 de setembro de 2005

ACM International Collegiate Programming Contest 2005

South American Regional - Brazilian Round

(Este caderno contém 8 problemas; as páginas são numeradas de 1 a 15)

Sedes:

Centro Universitário do Triângulo – Uberlândia, MG
Faculdades COC – Riberão Preto, SP
Faculdades Integradas Módulo – Caraguatatuba, SP
Fundação Universidade Federal do Rio Grande – Rio Grande, RS
Pontifícia Universidade Católica de Campinas – Campinas, SP
Universidade Anhembi Morumbi – São Paulo, SP
Universidade da Amazônia – Belém, PA
Universidade de Brasília – Brasília, DF
Universidade de Fortaleza – Fortaleza, CE
Universidade do Vale do Itajaí – Itajaí, SC
Universidade do Vale do Paraíba - São José dos Campos, SP
Universidade Estadual do Oeste do Paraná – Cascavel, PR
Universidade Federal de Minas Gerais – Belo Horizonte, MG
Universidade Federal do Maranhão – São Luís, MA
Universidade Federal do Mato Grosso do Sul – Campo Grande, MS
Universidade Federal do Rio de Janeiro – Rio de Janeiro, RJ
Universidade Federal do Rio Grande do Sul – Porto Alegre, RS
Universidade Federal do Sergipe – Aracaju, SE
Universidade Salgado de Oliveira – Juiz de Fora, MG

Problema A
Jogo do Bicho

Arquivo fonte: `bicho.c, bicho.cpp, bicho.java` *ou* `bicho.pas`

Em um país muito distante, as pessoas são viciadas em um jogo de apostas bastante simples. O jogo é baseado em números e é chamado *jogo do bicho*. O nome do jogo deriva do fato que os números são divididos em 25 grupos, dependendo do valor dos dois últimos dígitos (dezenas e unidades), e cada grupo recebe o nome de um animal. Cada grupo é associado a um animal da seguinte forma: o primeiro grupo (burro) consiste nos números 01, 02, 03 e 04; o segundo grupo (águia) é composto dos números 05, 06, 07 e 08; e assim em diante, até o último grupo contendo os números 97, 98, 99 e 00.

As regras do jogo são simples. No momento da aposta, o jogador decide o valor da aposta V e um número N ($0 \leq N \leq 1000000$). Todos os dias, na praça principal da cidade, um número M é sorteado ($0 \leq M \leq 1000000$). O prêmio de cada apostador é calculado da seguinte forma:

- se M e N têm os mesmos quatro últimos dígitos (milhar, centena, dezena e unidade), o apostador recebe $V \times 3000$ (por exemplo, $N = 99301$ e $M = 19301$);
- se M e N têm os mesmos três últimos dígitos (centena, dezena e unidade), o apostador recebe $V \times 500$ (por exemplo, $N = 38944$ e $M = 83944$);
- se M e N têm os mesmos dois últimos dígitos (dezena e unidades), o apostador recebe $V \times 50$ (por exemplo, $N = 111$ e $M = 552211$);
- se M e N têm os dois últimos dígitos no mesmo grupo, correspondendo ao mesmo animal, o apostador recebe $V \times 16$ (por exemplo, $N = 82197$ and $M = 337600$);
- se nenhum dos casos acima ocorrer, o apostador não recebe nada.

Obviamente, o prêmio dado a cada apostador é o máximo possível de acordo com as regras acima. No entanto, não é possível acumular prêmios, de forma que apenas um dos critérios acima deve ser aplicado no cálculo do prêmio. Se um número N ou M com menos de quatro dígitos for apostado ou sorteado, assuma que dígitos 0 devem ser adicionados na frente do número para que se torne de quatro dígitos; por exemplo, 17 corresponde a 0017.

Dado o valor apostado, o número escolhido pelo apostador, e o número sorteado, seu programa deve calcular qual o prêmio que o apostador deve receber.

Entrada

A entrada contém vários casos de teste. Cada caso consiste em apenas uma linha, contendo um número real V e dois inteiros N e M, representando respectivamente o valor da aposta com duas casas decimais ($0.01 \leq V \leq 1000.00$), o número escolhido para a aposta ($0 \leq N \leq 1000000$) e o número sorteado ($0 \leq M \leq 1000000$). O final da entrada é indicado por uma linha contendo $V = M = N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada um dos casos de teste seu programa deve imprimir uma linha contendo um número real, com duas casas decimais, representando o valor do prêmio correspondente a aposta dada.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`32.20 32 213929`
`10.50 32 213032`
`2000.00 340000 0`
`520.00 874675 928567`
`10.00 1111 578311`
`0 0 0`
	`515.20`
	`5250.00`
	`6000000.00`
	`0.00`
	`500.00`

Problema B
Curso Universitário

Arquivo fonte: `curso.c`, `curso.cpp`, `curso.java` *ou* `curso.pas`

Há alguns anos atrás, a Universidade de Pinguinhos introduziu um novo sistema flexível de créditos para os alunos ingressantes de cursos de graduação. No novo sistema os alunos podem escolher as disciplinas que desejam cursar em um semestre, com a única restrição de não poderem cursar uma dada disciplina A sem antes terem cursado todas as disciplinas que tiverem sido estabelecidas como pré-requisitos de A. Após alguns semestres o reitor notou que muitos estudantes estavam sendo reprovados em muitas disciplinas, simplesmente porque os estudantes estavam cursando muitas disciplinas por semestre – alguns estudantes chegavam a se matricular em até quinze disciplinas em um semestre. Sendo muito sábio, este ano o reitor introduziu uma regra adicional limitando o número de disciplinas que cada estudante pode cursar por semestre a um certo valor N. Essa regra adicional, no entanto, fez com que os alunos ficassem muito confusos na hora de escolher as disciplinas a serem cursadas em cada semestre.

É aí que você entra na estória. O reitor resolveu disponibilizar um programa de computador para ajudar os alunos a fazerem suas escolhas de disciplinas, e solicitou sua ajuda. Mais precisamente, o reitor quer que o programa sugira as disciplinas a serem cursadas durante o curso da seguinte forma. A cada disciplina é atribuída uma *prioridade*. Se mais do que N disciplinas podem ser cursadas em um determinado semestre (obedecendo ao sistema de pré-requisitos), o programa deve sugerir que o aluno matricule-se nas N disciplinas de maior prioridade. Se N ou menos disciplinas podem ser cursadas em um determinado semestre, o programa deve sugerir que o aluno matricule-se em todas as disciplinas disponíveis.

Portanto, dadas a descrição de pré-requisitos para cada disciplina, a prioridade de cada disciplina, e o número máximo de disciplinas por semestre, seu programa deve calcular o número necessário de semestres para concluir o curso, segundo a sugestão do reitor, e a lista de disciplinas que o aluno deve matricular-se em cada semestre.

Entrada

A entrada contém vários casos de teste. Se uma disciplina não tem qualquer pré-requisito ela é denominada *básica*; caso contrário ela é denominada *avançada*. A primeira linha de um caso de teste contém dois inteiros $1 \leq N \leq 100$ e $1 \leq M \leq 10$ indicando respectivamente o número de disciplinas avançadas do curso e o número máximo de disciplinas que podem ser cursadas por semestre. Cada uma das N linhas seguintes tem o formato

```
STR0 K STR1 STR2 ... STRK
```

onde `STR0` é o nome de uma disciplina avançada, $1 \leq$ `K` ≤ 15 é o número de disciplinas que são pré-requisitos de `STR0`, e `STR1`, `STR2`, ... `STRK` são os nomes das disciplinas que são pré-requisitos de `STR0`. O nome de uma disciplina é uma cadeia com no mínimo um e no máximo sete caracteres alfanuméricos maiúsculos ('A'-'Z' e '0'-'9'). Note que as disciplinas básicas são aquelas que aparecem apenas como pré-requisito de alguma disciplina avançada. Para concluir o curso, um aluno deve cursar (e passar!) todas as disciplinas básicas e avançadas. A prioridade das disciplinas é determinada pela ordem em que elas aparecem pela primeira vez na entrada: a que aparece primeiro tem maior prioridade, e a que aparece por último tem a menor prioridade. Não há circularidade nos pré-requisitos (ou seja, se a disciplina B tem como pré-requisito a

disciplina A então A não tem B como pré-requisito, direta ou indiretamente). O número total de disciplinas é no máximo igual a 200. O final da entrada é indicado por $N = M = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir a saída na seguinte forma. A primeira linha deve conter a frase '`Formatura em S semestres`', onde `S` é o número de semestres necessários para concluir o curso segundo a sugestão do reitor. As `S` linhas seguintes devem conter a descrição das disciplinas a serem cursadas em cada semestre, um semestre por linha, no formato mostrado no exemplo de saída abaixo. Para cada semestre, a lista das disciplinas deve ser dada em ordem lexicográfica.

Definição: considere as cadeias de caracteres $S_a = a_1a_2...a_m$ e $S_b = b_1b_2 \ldots b_n$. *S_a precede S_b* em ordem lexicográfica se e apenas se S_b é não-vazia e uma das seguintes condições é verdadeira:

- S_a é uma cadeia vazia;
- $a_1 < b_1$ na ordem '$0' < '1' < '2' < \ldots < '9' < 'A' < 'B' < \ldots < 'Z'$;
- $a_1 = b_1$ e a cadeia $a_2a_3 \ldots a_m$ precede a cadeia $b_2b_3 \ldots b_n$.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`2 2`
`B02 3 A01 A02 A03`
`C01 2 B02 B01`
`3 2`
`ARTE2 1 ARTE1`
`PROG3 1 PROG2`
`PROG2 2 MAT1 PROG1`
`0 0`
	`Formatura em 4 semestres`
	`Semestre 1 : A01 A02`
	`Semestre 2 : A03 B01`
	`Semestre 3 : B02`
	`Semestre 4 : C01`
	`Formatura em 4 semestres`
	`Semestre 1 : ARTE1 MAT1`
	`Semestre 2 : ARTE2 PROG1`
	`Semestre 3 : PROG2`
	`Semestre 4 : PROG3`

Problema C
Redução de Pena

Arquivo fonte: `pena.c, pena.cpp, pena.java` *ou* `pena.pas`

O conhecido e temido hacker SideBarCoder finalmente decidiu entregar-se às autoridades, depois de anos infernizando a vida de administradores de sistemas. SideBarCoder nunca causou prejuízos diretamente, mas deixava os administradores furiosos, tendo ficado famoso por deixar recados humorosos em arquivos localizados em sistemas supostamente impenetráveis.

SideBarCoder aceitou a proposta feita publicamente por uma grande empresa, de um ótimo salário, com a condição de revelar a sua verdadeira identidade e trabalhar dali em diante na melhoria da segurança do sistema Linux. No entanto, a Polícia Federal não foi tão condescendente quanto SideBarCoder esperava e prendeu-o no seu primeiro dia de trabalho.

Felizmente o juiz encarregado do caso levou em consideração o fato de SideBarCoder nunca ter causado prejuízo a nenhuma empresa, e decidiu que ele poderia reduzir o período na cadeia se realizasse trabalho voluntário nas escolas de seu bairro (como pintar paredes de salas de aulas, ajudar a cuidar de bebês em creches, etc).

O juiz entregou a SideBarCoder uma lista de tarefas possíveis. Cada tarefa corresponde a uma certa quantidade de *pontos* que podem ser utilizados para reduzir a pena. SideBarCoder deve realizar as tarefas no período de uma semana, de segunda-feira a sexta-feira. Para cada tarefa, o juiz especificou o dia e hora de início e de final da tarefa, e o número de pontos correspondente. O número de pontos atribuído à tarefa não é função de sua duração, mas sim de sua dificuldade. Assim, cuidar de uma classe com vinte crianças por duas horas representa um número maior de pontos do que pintar uma sala de aula, que demora quatro horas.

É lógico que SideBarCoder pensou em utilizar um computador para determinar como conseguir o maior número de pontos possível. No entanto, como pena adicional, o juiz determinou que SideBarCoder não pode chegar perto de um computador antes de cumprir sua pena. Desesperado, SideBarCoder solicitou que você escrevesse um programa para determinar quais tarefas ele deveria escolher.

Para ter os pontos contabilizados, uma tarefa deve ser cumprida integralmente (ou seja, do início ao final do período para ela estabelecido, sem interrupção). No conjunto de tarefas selecionado pelo seu programa não pode haver tarefas conflitantes. Duas tarefas conflitam se ambas devem ser executadas no mesmo dia e existir interseção entre os seus horários. Como todas as tarefas serão realizadas no bairro onde SideBarCoder mora, o tempo de deslocamento de uma tarefa para outra pode ser desconsiderado. Assim, ele pode executar sem conflito duas tarefas A e B tais que o horário de término de A é igual ao horário de início de B.

Entrada

A entrada contém vários casos de teste. A primeira linha de um caso de teste contém um inteiro $0 \leq N \leq 10000$ que indica o número de tarefas disponibilizadas pelo juiz. Cada uma das N linhas seguintes descreve uma tarefa, no seguinte formato:

```
Codigo Pontos Dia Inicio Final
```

onde

- `Codigo` é um inteiro que identifica univocamente uma tarefa ($1 \leq$ `Codigo` ≤ 10000))

- `Pontos` é um inteiro que indica o número de pontos da tarefa ($1 \leq$ `Pontos` ≤ 50))
- `Dia` é uma cadeia de três caracteres que indica o dia da semana (`Seg, Ter, Qua, Qui, Sex`) da tarefa;
- `Inicio` e `Final` indicam a hora de início e fim da tarefa, no formato *HH:MM* (entre 00:00 e 23:59). O final de uma tarefa ocorre sempre após o seu início e uma tarefa é inteiramente contida em um dia.

O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir seis linhas. A primeira linha deve conter a expressão '`Total de pontos:`', seguida de um espaço, seguida de um inteiro: o número máximo de pontos que SideBarCoder pode acumular. As cinco linhas seguintes da saída devem conter a lista dos pontos a serem conseguidos em cada dia da semana, no formato mostrado no exemplo de saída abaixo.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`3`
`5000 10 Seg 8:00 23:00`
`5001 20 Seg 11:00 12:00`
`5002 15 Seg 23:01 23:30`
`4`
`1977 5 Qua 10:00 10:29`
`1980 10 Qua 10:15 11:15`
`1983 6 Qua 11:00 12:00`
`1000 10 Seg 13:00 22:00`
`0`
	`Total de pontos: 35`
	`Seg: 35`
	`Ter: 0`
	`Qua: 0`
	`Qui: 0`
	`Sex: 0`
	`Total de pontos: 21`
	`Seg: 10`
	`Ter: 0`
	`Qua: 11`
	`Qui: 0`
	`Sex: 0`

Problema D
Cubos Coloridos

Arquivo fonte: `cubos.c`, `cubos.cpp`, `cubos.java` *ou* `cubos.pas`

Crianças adoram brincar com pequenos cubos. Elas passam horas criando 'casas', 'prédios', etc. O irmãozinho de Tomaz acabou de ganhar um conjunto de blocos coloridos no seu aniversário. Cada face de cada cubo é de uma cor.

Como Tomaz é uma criança muito analítica, ele decidiu descobrir quantos "tipos"diferentes de cubos o seu irmãozinho ganhou. Você pode ajudá-lo? Dois cubos são considerados do mesmo *tipo* se for possível rotacionar um deles de forma que as cores nas faces respectivas dos dois blocos sejam iguais.

Entrada

A entrada contém vários casos de teste. A primeira linha do caso de teste contém um inteiro N especificando o número de cubos no conjunto ($1 \leq N \leq 1000$). As próximas $3 \times N$ linhas descrevem os cubos do conjunto. Na descrição as cores serão identificadas pelos números de 0 a 9. A descrição de cada cubo será dada em três linhas mostrando as cores das seis faces do cubo "aberto", no formato dado no exemplo abaixo. No exemplo abaixo, as faces do cubo têm cores de 1 a 6, a face com cor 1 está no lado oposto da face com a cor 3, e a face com cor 2 é vizinha das faces 1, 3, 4 e 6, e está no lado oposto da face com cor 5.

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

O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste seu programa deve imprimir uma linha contendo um único inteiro, correspondente ao número de tipos de cubos no conjunto dado.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
3 0 0 7 2 3 1 0 1 2 3 7 0 3 0 0 2 1 7 2 1 1 1 1 1 1 2 2 2 2 2 2 0	2 2

Problema E
Gerente de Espaço

Arquivo fonte: `espaco.c, espaco.cpp, espaco.java` *ou* `espaco.pas`

É bem verdade que a maioria das pessoas não se importa muito com o que ocorre dentro de um computador, desde que ele execute as tarefas que devem ser desempenhadas. Existem, no entanto, alguns poucos *nerds* que sentem prazer em acompanhar o movimento de bits e bytes dentro da memória do computador.

É para esse público, constituído principalmente de adolescentes, que a multinacional de software ACM (Abstractions of Concrete Machines) deseja desenvolver um sistema que acompanhe e produza um relatório das operações efetuadas em um disco rígido. Um disco rígido é composto de uma seqüência de células atômicas de armazenamento, cada uma de tamanho 1Kb.

Especificamente, a ACM deseja acompanhar três tipos de operações:

- `insere NOME T`
  insere no disco o arquivo `NOME`, de tamanho `T`. Você pode supor que um arquivo com esse nome não existe ainda no disco. O tamanho `T` de um arquivo é dado na forma XKb, XMb, ou XGb, onde X é um inteiro ($0 < X \leq 1023$). `NOME` é uma cadeia de caracteres com comprimento máximo 10.

- `remove NOME`
  remove o arquivo NOME do disco. Se um arquivo com esse nome não existe, não faz nada;

- `otimiza`
  compacta o disco, deslocando os arquivos existentes na direção do início do disco, eliminando espaços livres entre dois arquivos subseqüentes, e preservando a ordem em que os arquivos aparecem no disco, de modo a deixar um espaço de memória livre no final do disco.

A capacidade de um disco é sempre um número múltiplo de 8Kb. No início, o disco está vazio, ou seja, contém um bloco livre do tamanho da capacidade do disco. Um arquivo é sempre armazenado em um bloco de células de armazenamento contíguas. O arquivo a ser inserido deve ser sempre colocado no início do menor bloco livre cujo tamanho é maior ou igual ao tamanho do arquivo. Se mais de um bloco livre é igualmente adequado, escolha o mais próximo do começo do disco. Caso não seja possível inserir o arquivo por falta de um bloco livre suficientemente grande, deve-se executar automaticamente o comando `otimiza`. Se após a otimização ainda não for possível inserir o arquivo, uma mensagem de erro deve ser produzida. No caso de todas as operações serem executadas sem erro, seu programa deve produzir uma estimativa aproximada do estado final do disco, conforme descrito abaixo.

Lembre que 1Mb corresponde a 1024Kb, enquanto 1Gb corresponde a 1024Mb.

Entrada

A entrada é constituída de vários casos de teste. A primeira linha de um caso de teste contém um único inteiro N indicando o número de operações no disco ($0 < N \leq 10000$). A segunda linha de um caso de teste contém a descrição do tamanho do disco, composta por um inteiro

D ($0 < D \leq 1023$), seguido de um especificador de unidade; o especificador de unidade é uma cadeia de dois caracteres no formato `Kb`, `Mb` ou `Gb`. Cada uma das N linhas seguintes contém a descrição de uma operação no disco (insere, remove ou otimiza, conforme descrito acima). O final da entrada é indicado por $N = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste seu programa deve produzir uma linha na saída. Se todas as operações de inserção forem executadas sem erro, seu programa deve produzir uma linha contendo uma estimativa aproximada do estado do disco, apresentada como se segue. Divida o número de bytes do disco em oito blocos contíguos de mesmo tamanho. Para cada um dos oito blocos seu programa deve verificar a porcentagem P de bytes livres daquele bloco, e apresentar a estimativa do estado final no formato

`[C] [C] [C] [C] [C] [C] [C] [C]`

onde `C` é ' ', '-' ou '#', dependendo se $75 < P \leq 100$, $25 < P \leq 75$ ou $0 \leq P \leq 25$, respectivamente. Caso um arquivo não possa ser inserido por falta de espaço, seu programa deve produzir uma linha contendo a expressão `ERRO: disco cheio`; nesse caso, operações subseqüentes do caso de teste devem ser ignoradas.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`3`
`8Kb`
`insere arq0001 7Kb`
`insere arq0002 3Mb`
`remove arq0001`
`6`
`8Mb`
`insere arq0001 4Mb`
`insere arq0002 1Mb`
`insere arq0003 512Kb`
`remove arq0001`
`remove arq0001`
`insere arq0001 5Mb`
`0`
	`ERRO: disco cheio`
	`[#] [#] [#] [#] [#] [#] [-] [ ]`

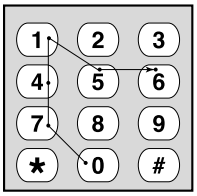
Problema F
Tecle & Some

Arquivo fonte: `tecle.c, tecle.cpp, tecle.java` *ou* `tecle.pas`

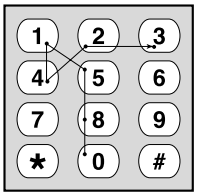
Strike Boy, como o apelido sugere, é um garoto fanático por todo tipo de jogos em computador. Ele está passando as férias em uma ilha paradisíaca, onde computadores não são permitidos. Ele se divertiu por algum tempo com os jogos em seu telefone celular, mas a bateria acabou e não há eletricidade na ilha, de forma que ele parou de jogar. Strike Boy então decidiu inventar um novo passatempo, usando o teclado de seu telefone celular. Neste novo jogo, para dois jogadores, um deles escolhe dois inteiros S e D. O jogador oponente deve então encontrar uma *seqüência de termos* tal que:

- cada termo da seqüência é um número com D dígitos decimais, exceto pelo último termo, que pode ter entre 1 e D dígitos;
- a soma de todos os termos da seqüência é igual a S;
- os dígitos utilizados para formar um termo correspondem às teclas de um teclado padrão de telefone celular ('0' a '9');
- cada dígito é utilizado no máximo uma vez na seqüência;
- o primeiro termo de uma seqüência pode começar com qualquer dígito, mas a ordem dos dígitos da seqüência, quando lidos da esquerda para a direita, é tal que a próxima tecla corresponde sempre a uma tecla imediatamente vizinha da tecla utilizada previamente (na vertical, na horizontal ou na diagonal).

Por exemplo, se $S = 230$ e $D = 3$, há apenas duas soluções possíveis obedecendo as regras do jogo: [074, 156] e [085, 142, 3]. A seqüência [230] não é uma solução porque a tecla '3' não é vizinha da tecla '0'.



Teclado ilustrando as teclas utilizadas para formar a seqüência [074, 156]



Teclado ilustrando as teclas utilizadas para formar a seqüência [085, 142, 3]

Ajude Strike Boy a verificar se as respostas do oponente estão corretas: escreva um programa que, dados S e D, imprima todas as soluções possíveis.

Entrada

A entrada contém vários casos de teste. Cada caso de teste consiste em apenas uma linha, contendo dois inteiros S e D, separados por um espaço, representando a soma desejada e o número de dígitos de cada termo ($0 \leq S \leq 10,000,000,000$ e $1 \leq D \leq 10$). O final da entrada é indicado por $S = D = -1$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir uma resposta. A primeira linha de uma resposta deve conter um identificador do caso de teste, no formato '`#i`', onde '`i`' tem inicialmente o valor 1 e é incrementado a cada caso de teste. Então, se uma solução para o passatempo existe, seu programa deve produzir uma lista das possíveis seqüências de termos. Se mais de uma seqüência é possível, elas devem aparecer em ordem lexicográfica crescente. Cada seqüência de termos deve ser impressa em uma linha, com os termos separados por um espaço em branco. Se não há solução, seu programa deve imprimir uma linha contendo a palavra '`impossivel`' (note ausência de acentuação).

Definição: considere as seqüências $S_a = a_1 a_2 ... a_m$ e $S_b = b_1 b_2 \ldots b_n$. *S_a precede S_b* em ordem lexicográfica se e apenas se S_b é não-vazia e uma das seguintes condições é verdadeira:

- S_a é uma seqüência vazia;
- $a_1 < b_1$;
- $a_1 = b_1$ e a seqüência $a_2 a_3 \ldots a_m$ precede a seqüência $b_2 b_3 \ldots b_n$.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`7 1`
`10 2`
`230 3`
`6311 4`
`2 2`
`-1 -1`
	`#1`
	`0 7`
	`1 2 4`
	`1 4 2`
	`2 1 4`
	`2 4 1`
	`2 5`
	`4 1 2`
	`4 2 1`
	`5 2`
	`7`
	`7 0`
	`#2`
	`impossivel`
	`#3`
	`074 156`
	`085 142 3`
	`#4`
	`0896 5412 3`
	`0986 5321 4`
	`0986 5324 1`
	`0987 5324`
	`#5`
	`2`

Problema G
Regata de Cientistas

Arquivo fonte: `regata.c, regata.cpp, regata.java` *ou* `regata.pas`

Todos os anos, desde 1996, cientistas da computação do mundo todo se encontram para a famosa Regata dos Cientistas. A competição consiste em uma corrida de barcos com obstáculos pelo oceano, onde o objetivo de cada equipe é, partindo de um ponto em comum, alcançar o ponto de chegada sem que nenhum obstáculo seja tocado ou transpassado. Uma equipe que toca ou transpassa um obstáculo é automaticamente desclassificada. A equipe vencedora é aquela que primeiro atinge o ponto de chegada (o ponto de chegada é distinto do ponto de início).

Você foi contratado pela equipe brasileira para desenvolver um programa que calcule o comprimento da menor rota válida possível do ponto de partida ao ponto de chegada.

O oceano é considerado um plano infinito, onde cada obstáculo é localizado em uma posição fixa e representado por um segmento de reta descrito por seus dois extremos (x_1, y_1) e (x_2, y_2). Os barcos são adimensionais (representados como um ponto no plano) e os obstáculos possuem espessura desprezível.

Os obstáculos são dispostos de tal forma que os mesmos não se interceptam. De forma similar, os pontos de início e de chegada da competição não são interceptados por nenhum obstáculo.

Entrada

A entrada é composta por vários casos de teste. A primeira linha de um caso de teste contém cinco números inteiros x_i, y_i, x_f, y_f e n, representando respectivamente as coordenadas do ponto de início (x_i, y_i), as coordenadas do ponto de chegada (x_f, y_f) e a quantidade de obstáculos ($n \leq 150$). Cada uma das n linhas seguintes de um caso de teste contém quatro números inteiros x_1, y_1, x_2 e y_2 que descrevem as coordenadas dos dois extremos de um obstáculo. Considere que as coordenadas x e y de qualquer ponto satisfazem $-5000 \leq x, y \leq 5000$. O final da entrada é representado por uma linha contendo $x_i = y_i = x_f = y_f = n = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste, imprima uma linha contendo o comprimento da menor rota válida possível, arredondado para duas casas decimais.

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**

`0 0 10 0 1`
`5 -1 5 1`
`0 0 10 0 1`
`5 0 5 1`
`0 0 0 0 0`
	`10.20`
	`10.00`

Problema H
Piscina

Arquivo fonte: `piscina.c, piscina.cpp, piscina.java` *ou* `piscina.pas`

O Centro Comunitário decidiu construir uma nova piscina, em tempo para o verão do ano que vem. A nova piscina será retangular, com dimensões X por Y e profundidade Z. A piscina será recoberta com um novo tipo de azulejos cerâmicos, de alta tecnologia, que é produzido em três tamanhos distintos: 5×5, 15×15 e 30×30 (em centímetros). Cada azulejo desses tamanhos custa 2 centavos, 15 centavos e 50 centavos, respectivamente. Os azulejos são de alta qualidade, feitos com um material que não pode ser cortado (ou seja, os azulejos devem ser utilizados inteiros).

A única loja local que vende esse tipo de azulejo tem em estoque uma certa quantidade de azulejos de cada tamanho. Você deve escrever um programa que determine se o estoque de azulejos disponível na loja é suficiente para azulejar toda a piscina. Se o estoque for suficiente, seu programa deve determinar também o número de azulejos de cada tamanho que são necessários para que o custo de azulejar a piscina seja o menor possível.

Os azulejos devem ser usados para recobrir completamente toda a superfície da piscina, sem deixar qualquer espaço sem azulejos, e sem deixar sobras de azulejos transpassando as bordas da piscina.

Entrada

A entrada contém vários casos de teste. Cada caso de teste é composto por duas linhas. A primeira linha contém três números reais X, Y e Z, representando as dimensões e a profundidade da piscina, em metros, com precisão de uma casa decimal ($0 < X, Y \leq 50.0$ e $0 < Z \leq 2.0$). A segunda linha contém três números inteiros P, M e G, representando a quantidade disponível de azulejos de tamanho pequeno, médio e grande ($0 \leq P, M, G \leq 2000000$), respectivamente. O final da entrada é indicado por $X = Y = Z = 0$.

A entrada deve ser lida da entrada padrão.

Saída

Para cada caso de teste da entrada seu programa deve produzir uma linha de saída. Se é possível recobrir completamente a piscina com o estoque disponível, imprima uma linha com três inteiros descrevendo respectivamente as quantidades de azulejos pequenos, médios e grandes para recobrir toda a piscina, com o menor custo possível. Caso contrário, imprima uma linha contendo a palavra '`impossivel`' (note a ausência de acentuação).

A saída deve ser escrita na saída padrão.

Exemplo de entrada	**Saída para o exemplo de entrada**
3.0 4.0 1.0 1000 1000 1000 3.0 3.0 0.9 300 300 300 12.5 12.5 1.6 5000 0 3000 3.0 3.0 1.0 300 300 300 0 0 0	752 0 268 0 0 220 4464 0 2501 impossivel