CADERNO DE PROBLEMAS

09/05/2007 – 08:00 às 12:00

Leia atentamente estas instruções antes de iniciar a resolução dos problemas. Este caderno é composto de 5 problemas, 2 deles estão descritos em inglês.

1. É proibido utilizar livros, apostilas impressas, manuais ou qualquer outro material que não seja fornecido pela comissão organizadora durante a prova, com exceção de dicionários de inglês. É permitida a consulta do *help* do ambiente de programação se este estiver disponível e do material teórico referente às linguagens de programação (arquivos) gravados no D\ material.
2. A correção das resoluções dos problemas será semi-automatizada, baseada nos resultados obtidos com uma série de execuções dos algoritmos de cada equipe.
3. Siga atentamente as exigências da tarefa quanto ao formato da entrada e saída do seu algoritmo. Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
4. Os problemas não estão ordenados, neste caderno, por ordem de dificuldade. Procure resolver primeiro os problemas mais fáceis.
5. Utilize para nomear os seus arquivos-fonte os nomes indicados nos problemas de acordo com a linguagem de programação utilizada.
6. Cada equipe receberá 1 disquete com o nome da equipe que deve ser utilizado para a entrega dos algoritmos para correção.

Observações:

Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (teclado) e escritos na saída padrão (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`;
- em JavaScript: `prompt`, `document.write`;

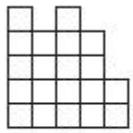
Quando a linguagem escolhida para a resolução dos problemas for o Pascal, recomendamos que não seja utilizada a instrução: `uses newdelay, crt;`

Faculdade de Informática de Presidente Prudente
http://www.unoeste.br/fipp/maratona

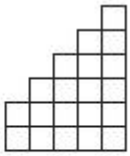
Escada Perfeita (1 – vermelho)

Arquivo fonte: *escada.c*, *escada.cpp*, *escada.pas* ou *escada.html*

Uma construtora, durante a criação de um parque temático, encontrou no terreno um conjunto de várias pilhas de cubos de pedra. Ao invés de pagar pela remoção dos cubos de pedras, um dos arquitetos da empresa achou interessante utilizar as pedras para decoração do parque, determinando que as pedras fossem rearranjadas no formato de "escada". Para isso, os funcionários deveriam mover alguns cubos para formar os degraus das escadas. Só que o arquiteto decidiu que, entre uma pilha e outra de pedras deveria haver exatamente uma pedra de diferença, formando o que ele chamou de escada perfeita. O exemplo abaixo mostra um conjunto de cinco pilhas de pedras encontradas e as cinco pilhas como ficaram após a arrumação em escada perfeita.



Arranjo de pilhas de cubos de pedras encontrado



Escada perfeita construída após o movimento de alguns cubos

Tarefa

Dada uma seqüência de pilhas de cubos de pedras com suas respectivas alturas, você deve determinar o número mínimo de pedras que precisam ser movidas para formar uma escada perfeita com exatamente o mesmo número de pilhas de pedras encontrado inicialmente (ou seja, não devem ser criadas ou eliminadas pilhas de pedras). O degrau mais baixo da escada deve sempre estar do lado esquerdo.

Entrada

A entrada pode conter vários conjuntos de testes, que deve ser lido do dispositivo de entrada padrão (normalmente o teclado). A primeira linha contém um inteiro N que indica o número de pilhas de pedras ($1 \leq N \leq 100$). A segunda linha contém N números inteiros que indicam a quantidade de cubos de pedras em cada uma das pilhas, da esquerda para a direita. O fim da entrada é indicado por $N = 0$.

Exemplo de Entrada

```
5
5 4 5 4 2
6
9 8 7 6 5 4
2
1 5
0
```

Saída

Seu programa deve imprimir, na saída padrão, uma única linha para cada conjunto de teste, contendo um inteiro: o número mínimo de cubos de pedras que devem ser movidos para transformar o conjunto de pilhas em uma escada perfeita, conforme calculado pelo seu programa. Caso não seja possível efetuar a transformação em escada perfeita, imprima como resultado o valor -1.

Exemplo de Saída

```
5
9
-1
```

Cubos Coloridos (2 – amarelo)

Arquivo fonte: *cubos.c*, *cubos.cpp*, *cubos.pas* ou *cubos.html*

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.

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
```

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 Saída

```
2
2
```

Gerente de Espaço (3 – laranja)

Arquivo fonte: *espaco.c*, *espaco.cpp*, *espaco.pas* ou *espaco.html*

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 *X*Kb, *X*Mb, ou *X*Gb, 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.

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
```

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 Saída

```
ERRO: disco cheio
[#][#][#][#][#][#][-][ ]
```

Divide and Conquer (4 – verde)

Arquivo fonte: *divide.c*, *divide.cpp*, *divide.pas* ou *divide.html*

Given two integers M and N, with $1 \leq M \leq N \leq 5000$, you must determine the integer numbers X and Y such that:

A. $M \leq X \leq N$; and

B. Y is the number of divisors of X; and

C. Y is the largest possible; and

D. X is the largest possible.

Input

Your program should process several test cases. Each test case is composed by a single line containing the two integers M and N ($1 \leq M \leq N \leq 5000$). The end of input is indicated by $M = N = 0$.

The input must be read from standard input.

Sample Input

```
1 5
300 500
4500 5000
0 0
```

Output

For each test case in the input your program should print a single line, containing the two integers X and Y, separated by a space character.

The output must be written to standard output.

Sample Output

```
4 3
480 24
4680 48
```

Help! (5 – branco)

Arquivo fonte: *help.c*, *help.cpp*, *help.pas* ou *help.html*

Well, we have to admit: we need your help. This year things have not been running as smoothly as we wished, and we could not finish the contest system software in time. One vital part is missing, and as you know, we need the system working by this afternoon, for the real contest. The missing part is the module that computes the team's scores, given the team's list of submissions.
Please, please, someone help us!

Input

The input contains several test cases. The first line of a test case contains a single integer N indicating the number of submissions in the test ($1 \leq N \leq 300$). Each of the following N lines describe a submission; each of these lines contains a *problem identifier* (a single letter from 'A' to 'Z'), followed by an integer T representing the time in minutes ($0 \leq T \leq 300$), followed by a *judgement* (the word "correct" or the word "incorrect"). The input is in ascending order by time, and there will be at most one "correct" judgement for each problem. The end of input is indicated by $N = 0$.

The input must be read from standard input.

Sample Input

```
3
A 120 incorrect
A 130 incorrect
A 200 incorrect
5
A 100 correct
B 110 incorrect
B 111 correct
C 200 correct
D 300 incorrect
0
```

Output

For each test case in the input your program should output a line containing two integers S and P, separated by a space, where S is the number of distinct problems with a "correct" judgement and P is the time at which each distinct problem is first judged "correct", plus 20 for each "incorrect" submission for a problem that is later judged "correct".

The output must be written to standard output.

Sample Output

```
0 0
3 431
```