Maratona Mineira de Programação 2012

Caderno de Problemas

26 de Maio de 2012

Realização

Apoio

Instruções:

- Este caderno contém 11 problemas. As páginas estão numeradas de 1 a 17, não contando esta página de rosto. Verifique se o caderno está completo.
- Em todos os problemas, a entrada de seu programa deve ser lida da *entrada padrão*. A saída deve ser escrita na *saída padrão*.

Problema A. Poodle

Nome do arquivo fonte: `poodle.c`, `poodle.cpp`, ou `poodle.java`

Maria vive perdendo coisas dentro de casa. Desde pequena, tudo em que ela põe a mão desaparece. Isso acontece porque ela é muito desorganizada, deixa tudo espalhado pela casa, tornando humanamente impossível localizar algum objeto no meio de tanta confusão. Ela sempre contou com a ajuda infalível de seu cãozinho Poodle, que consegue localizar seus objetos perdidos. Uma vez ela queria me mostrar a eficiência do seu Poodle. Escondeu propositalmente uma bola em um dos quartos, e gritou: "Poooooooodle!". Então ela disse "Bola" e ele partiu para buscá-la. Ela ficou preocupada porque depois de 30 segundos ele ainda não tinha retornado com a bola. A surpresa foi que logo depois ele apareceu, triufante, carregando 4 bolas!!! A que Maria acabara de esconder e outras 3 de seus filhos, que ela nem se lembrava que existiam, e muito menos onde estavam!

Atualmente Maria faz pós-graduação em Computação. Seu projeto final é uma ferramenta de busca. Em homenagem a seu cãozinho que sempre buscou suas coisas, ela batizou seu projeto de Poodle. A ideia é simples: dada uma palavra, Poodle faz uma busca no disco e retorna todos os documentos que contém a dada palavra. Como no caso real da bola que comentei acima, na maioria das vezes a busca retorna bem mais resultados que o esperado. Os resultados são então exibidos agrupados em páginas. Por exemplo, se a ferramenta for configurada para exibir 10 resultados por página, e a busca retornar 143 resultados, eles serão exibidos em 15 páginas: 14 delas com 10 em cada uma, e a última com os 3 restantes.

A ferramenta já está pronta, e funciona muito bem. Mas Maria teve a feliz ideia de enfeitar o trabalho na tentativa de ganhar mais pontos... ao exibir o resultado de uma busca, Poodle mostra um logotipo com o nome da ferramenta, sendo que os o's de Poodle podem ser dois ou mais, dependendo da quantidade de páginas de resultado. A ideia é que "Poodle" seja escrito com tantas letras quantas forem as páginas de resultado, repetindo o's quando necessário. No exemplo acima, o logotipo seria "Pooooooooooodle", que contém 15 letras.

Naturalmente, se a quantidade de páginas de resultados for inferior a 6, a palavra Poodle não será cortada, o logotipo será Poodle. E, para evitar que o logotipo fique tão grande que nem caiba na tela de resultados, ele será limitado a um máximo de 20 letras, mesmo que a quantidade de páginas de resultado seja superior a 20.

Sua tarefa é ajudar Maria a montar o logotipo.

Entrada

Há vários casos de teste.

Cada caso de teste é uma linha contendo dois números inteiros, N e P, sendo N o número de documentos encontrados pelo Poodle, e P o número de resultados exibidos por página ($1 \leq N \leq 1.000.000$, $1 \leq P \leq 100$). A entrada termina quando $N = P = 0$.

Saída

Para cada caso de teste da entrada seu programa deve produzir uma linha na saída contendo a palavra Poodle, ajustando a quantidade de o's de acordo com as regras descritas no enunciado.

Exemplos

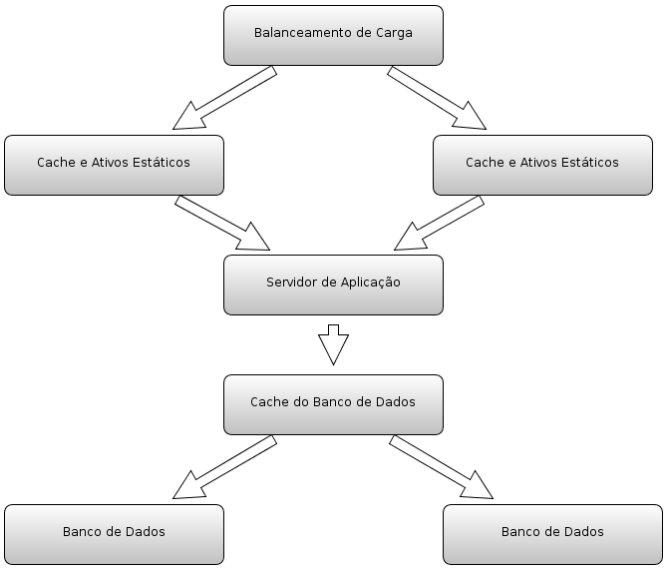
Entrada	Saída
20 4 143 10 42 5 80 3 0 0	Poodle Pooooooooooodle Pooooodle Poooooooooooooooodle

Problema B. Melhorando a Robustez de Aplicações *Web*

Nome do arquivo fonte: `robustez.c`, `robustez.cpp`, ou `robustez.java`

A Web tem crescido a uma taxa assustadoramente alta nos últimos anos, e agora possui mais de 2 bilhões de usuários. Os *Web sites* mais populares têm milhões de usuários. Devido a esses números, criar uma arquitetura para aplicações Web de larga escala, capazes de suportar milhões de usuários concorrentemente, é uma tarefa difícil, mas importante. Por exemplo, duas preocupações que um projetista tem que ter em mente são a latência (tempo de resposta) e a robustez do sistema.

Aqui, vamos nos preocupar com a robustez. Idealmente, uma aplicação Web deve ficar disponível 100% do tempo. Na prática, servidores falham, e é impossível criar uma arquitetura na qual haja uma garantia de uptime de 100%. Porém, é possível chegar arbitrariamente perto disso. Uma aplicação Web é normalmente estruturada em camadas. Cada camada possui alguma responsabilidade específica, e se comunica com as camadas adjacentes. A figura abaixo mostra um exemplo de arquitetura com 5 camadas:



Note que a arquitetura mostrada na figura possui problemas em potencial de robustez. Há apenas um servidor de aplicação. Se ele falhar, então o sistema não será capaz de atender nenhuma requisição que não possa ser servida diretamente do cache na camada 2. Uma maneira de contornar esse problema é adicionar um segundo servidor de aplicação. Assim, mesmo que um deles falhe, ainda seria possível atender as requisições. Há, claro, a chance de que os dois servidores falhem ao mesmo tempo, mas ela é menor que a chance de que um único servidor falhe isoladamente. O mesmo vale para as outras camadas: é sempre possível adicionar mais servidores e melhorar a robustez daquela camada.

O problema, obviamente, é que servidores extras incorrem em custos extras. Os servidores em si possuem um custo alto, e ainda há o custo de manutenção e energia. Assim, ainda que ter, digamos, 400 servidores de aplicação pudesse aumentar muito a robustez do sistema, essa não seria uma solução interessante devido ao custo.

A probabilidade de falha da i-ésima camada, denotada por p_i, é definida como a probabilidade de que todos os servidores naquela camada falhem simultaneamente. Se a probabilidade de falha de um servidor individual naquela camada é f e há n servidores naquela camada, então

$$p_i = f^n.$$

A robustez da camada i, denotada por r_i, é a probabilidade de que a camada i funcione, e é definida como

$$r_i = 1 - p_i = 1 - f^n.$$

A robustez do sistema como um todo, denotada por R, é definida como a probabilidade de que todas as camadas do sistema funcionem, simultaneamente:

$$R = \prod_i r_i.$$

Todos os servidores em uma mesma camada são idênticos. Em particular, eles possuem o mesmo custo, e a mesma probabilidade de falha. Porém, servidores de camadas diferentes podem ser diferentes.

São dados:

- o número de camadas N do sistema;
- o custo c_i de um servidor da i-ésima camada;
- a probabilidade de falha f_i de um servidor da i-ésima camada e
- o custo total máximo B.

Você deve determinar qual é a robustez máxima para o sistema como um todo (R) que é possível obter de forma tal que o custo não ultrapasse B.

Observações

Se uma camada tem zero servidores, então sua robustez é zero.

Entrada

Há vários casos de teste.

Cada caso de teste começa com uma linha contendo dois inteiros N e B, respectivamente o número de camadas no sistema ($1 \leq N \leq 100$) e o custo total máximo ($1 \leq B \leq 1000$) em milhares de reais. Em seguida, há N linhas. A i-ésima dessas linhas possui dois números, c_i e f_i. c_i é um inteiro que representa o custo de um servidor na i-ésima camada em milhares de reais ($1 \leq c_i \leq 200$). f_i é um número de ponto flutuante que representa a probabilidade de falha de um servidor da i-ésima camada ($0 < f_i \leq 1$, o número é dado com 3 casas decimais de precisão).

A entrada termina com $N = B = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo um único número, a robustez R máxima que é possível obter para o sistema descrito sem estourar o custo máximo. Esse valor deve ser impresso com 3 casas decimais de precisão.

Exemplos

Entrada	Saída
3 105 30 0.100 15 0.200 20 0.500 0 0	0.648

Nesse caso, a melhor opção é comprar um servidor para a primeira camada, dois servidores para a segunda camada e dois servidores para a terceira camada, a um custo total de $30 + 15 \cdot 2 + 20 \cdot 2 = 100$. A robustez da primeira camada será $1 - 0.1^1 = 0.9$. A da segunda camada será $1 - 0.2^2 = 0.96$ e a da terceira camada será $1 - 0.5^2 = 0.75$. Portanto, a robustez total é de $0.9 \cdot 0.96 \cdot 0.75 = 0.648$.

Problema C. Investindo no Mercado de Ações I

Nome do arquivo fonte: `acoes1.c`, `acoes1.cpp`, ou `acoes1.java`

João é um dos muitos investidores que vem aumentando sua fortuna nos últimos anos com negociações no mercado de ações. Curiosamente, seu patrimônio cresceu consideravelmente desde que ele resolveu adotar uma estratégia bem particular de investimento.

Considere que João possui N reais para investir e que ele nunca investe mais do que K reais em ações de uma mesma empresa, com o objetivo de diversificar sua carteira e teoricamente reduzir o seu risco. Para tanto, João divide seu capital em partes de no máximo K reais, de acordo com a estratégia descrita a seguir. Inicialmente, se $N > K$, João divide seu capital em duas partes de $\lfloor \frac{N}{2} \rfloor$ e $\lceil \frac{N}{2} \rceil$ reais e continua dividindo cada uma dessas partes de maneira similar, até resultar em partes de no máximo K reais cada. Ao final desse processo, João terá seu capital inicial dividido em E partes e investirá integralmente cada uma delas em ações de uma única empresa, não podendo investir mais de uma parte em uma mesma empresa. Sua tarefa consiste em ajudar João a descobrir em quantas empresas ele irá investir utilizando essa estratégia.

Por exemplo, considere que $N = 18$ e $K = 4$. Após a primeira divisão João terá duas partes de 9 reais. Cada uma dessas partes será dividida, resultando em duas partes de 5 reais e duas partes de 4 reais. As partes de 5 reais são então divididas novamente, resultando em duas partes de 2 reais e duas partes de 3 reais. As partes de 4 reais não precisam mais ser divididas. Logo, todas as 6 partes resultantes (duas de 2 reais, duas de 3 reais e duas de 4 reais) possuem no máximo 4 reais e são utilizadas por João para investir em ações de 6 empresas distintas.

Entrada

Há vários casos de teste.

Cada caso de teste é descrito em uma única linha contendo dois inteiros N e K, respectivamente o capital inicial de João ($1 \leq N \leq 1.000.000$) em reais e a quantidade máxima de reais ($1 \leq K \leq 1.000.000$) que João pode investir para comprar ações de uma mesma empresa.

A entrada termina com $N = K = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma única linha contendo um único número, a quantidade de empresas E em que João irá investir seu capital.

Exemplos

Entrada	Saída
18 4 5 10 100 1 64 6 0 0	6 1 100 16

Problema D. Investindo no Mercado de Ações II

Nome do arquivo fonte: `acoes2.c`, `acoes2.cpp`, ou `acoes2.java`

José é amigo de João (personagem do Problema C. Investindo no Mercado de Ações I) e também investe no mercado de ações. Porém, a estratégia de investimento utilizada por José é um tanto quanto complexa se comparada à estratégia utilizada por João.

Considere que José possui duas sequências de inteiros $P = (P_1, P_2, \ldots, P_A)$ e $R = (R_1, R_2, \ldots, R_B)$, tal que P é uma sequência de inteiros que representa os preços das últimas A negociações de uma determinada ação e R é uma sequência de inteiros distintos utilizada como referência para decidir se José deve ou não investir nas ações em questão.

A partir da sequência P, José gera uma nova sequência $M = (M_1, M_2, \ldots, M_A)$, em que M_i é igual à mediana dos valores da subsequência $P(1:i) = (P_1, P_2, \ldots, P_i)$, para todo $i = 1, 2, \ldots, A$. A mediana da subsequência $P(1:i)$ é definida como o k-ésimo valor da subsequência $P(1:i)$ ordenada em ordem crescente, em que $k = \lceil \frac{i}{2} \rceil$. Por exemplo, se temos a sequência de preços $P = (5, 3, 2, 1, 4, 2)$, então:

- $M_1 =$ mediana de $P(1:1) =$ mediana de $(5) =$ primeiro valor de $(5) = 5$
- $M_2 =$ mediana de $P(1:2) =$ mediana de $(5, 3) =$ primeiro valor de $(3, 5) = 3$
- $M_3 =$ mediana de $P(1:3) =$ mediana de $(5, 3, 2) =$ segundo valor de $(2, 3, 5) = 3$
- $M_4 =$ mediana de $P(1:4) =$ mediana de $(5, 3, 2, 1) =$ segundo valor de $(1, 2, 3, 5) = 2$
- $M_5 =$ mediana de $P(1:5) =$ mediana de $(5, 3, 2, 1, 4) =$ terceiro valor de $(1, 2, 3, 4, 5) = 3$
- $M_6 =$ mediana de $P(1:6) =$ mediana de $(5, 3, 2, 1, 4, 2) =$ terceiro valor de $(1, 2, 2, 3, 4, 5) = 2$

Assim, se $P = (5, 3, 2, 1, 4, 2)$, temos que $M = (5, 3, 3, 2, 3, 2)$.

Após determinar a sequência M, José a compara com a sequência de referência R, formada por inteiros distintos e obtida através de informações privilegiadas, que representa a sequência ótima de valores para as medianas dos preços das últimas negociações de uma determinada ação. Assim, se as sequências M e R forem iguais, José pode investir tranquilamente tendo a certeza de que consiguirá obter o maior lucro possível em seu investimento.

Como é muito difícil que as sequências M e R sejam iguais, José considera suficiente que elas sejam ao menos bastante similares para que ele invista nas ações. Assim, para determinar o quanto a sequência M se assemelha da sequência R, José deve calcular a maior subsequência comum de M e R.

Uma subsequência X' de uma sequência de inteiros X é obtida a partir da remoção de zero ou mais inteiros de X, com os elementos de X' aparecendo na mesma ordem em que aparecem em X. Por exemplo, se $X = (2, 4, 1, 1, 2, 3)$, então $X' = (2, 1, 1, 3)$ é uma subsequência de X, enquanto $X' = (1, 4)$ e $X' = (2, 1, 5)$ não são subsequências de X. Assim, as subsequências comuns das duas sequências $(5, 3, 3, 2, 4)$ e $(4, 5, 3, 4)$ são $()$, (3), (4), (5), $(5, 3)$, $(5, 4)$ e $(5, 3, 4)$, sendo a última a maior delas, com tamanho 3.

José está com dificuldades para determinar se deve ou não investir nas ações e precisa da sua ajuda. Para tanto, dadas as sequências P e R, você deve calcular o tamanho da maior subsequência comum de M e R, em que M é a sequência de medianas calculada da maneira descrita anteriormente.

Entrada

Há vários casos de teste.

Cada caso de teste é descrito em duas linhas. A primeira linha inicia com um inteiro A ($1 \leq A \leq 100.000$) que representa o tamanho da sequência P, seguido de A inteiros que representam a sequência P. A segunda linha inicia com um inteiro B ($1 \leq B \leq 100.000$) que representa o tamanho da sequência R,

seguido de B inteiros distintos que representam a sequência R. Todos os inteiros das sequências P e R são maiores ou iguais a 0 e menores ou iguais a $1.000.000.000$.

A entrada termina com $A = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma única linha contendo um único inteiro, o tamanho da maior subsequência comum de M e R.

Exemplos

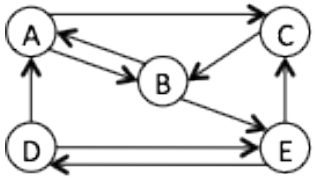
Entrada	Saída
6 5 3 2 1 4 2 5 5 1 2 7 3 4 1 2 3 4 7 7 6 5 4 3 2 1 3 1 1 1 5 2 4 3 5 6 0	3 1 0

Problema E. Atleta mentiroso

Nome do arquivo fonte: `atleta.c`, `atleta.cpp`, ou `atleta.java`

Como parte do treinamento para as Olimpíadas do Rio 2016, uma equipe de atletas brasileiros precisa passar um bom tempo na academia. Dependendo das condições físicas e do papel do atleta na equipe, cada um recebeu uma série de exercícios que devem ser feitos ao longo do dia, com duração variada. A única recomendação comum a todos é que devem fazer seus exercícios sem intervalo: podem entrar na academia quando quiserem, mas depois de entrar, devem fazer todos os exercícios sugeridos pelo treinador, e só então podem sair da academia; e depois de sair não voltam mais. O treinador está desconfiado que um de seus atletas é mentiroso e não está nem aí para o treinamento.

Certo dia perguntou a cada um dos atletas quem ele tinha visto na academia no dia anterior. Arthur disse que tinha visto Bruno e Cesar; Bruno afirma ter visto Arthur e Everton; Cesar disse que viu Bruno; Diego afirma ter visto Arthur e Everton; e Everton disse ter visto Cesar e Diego. A figura abaixo foi feita com essas informações: uma seta de X para Y indica que X afirma ter visto Y.



Sabe-se que enquanto um atleta está na academia, não vê necessariamente todos os que passam por lá durante esse tempo. Mas sempre que dois estiveram presentes na academia simultaneamente, mesmo que por pouco tempo, pelo menos um deles vê o outro. Por exemplo, pelas informações acima é possível concluir que Bruno e Diego não estiverem presentes simultaneamente em nenhum momento, pois nenhum deles viu o outro. Mas houve algum momento em que Arthur e Cesar estavam presentes simultaneamente, já que Arthur viu Cesar (isso é verdade mesmo Cesar não tendo visto Arthur).

Analisando as informações o treinador concluiu que Diego é o mentiroso.

Vejamos como: primeiramente vamos provar que entre Arthur, Bruno, Diego e Everton, um deles mentiu. Suponha, por um momento, que nenhum deles tenha mentido. Já vimos que Bruno e Diego não estiveram na academia simultaneamente, pois não se encontraram (Bruno não viu Diego nem Diego viu Bruno). Mas Arthur esteve com os dois. A figura abaixo mostra duas maneiras disso der ocorrido. Cada traço representa o horário em que o atleta esteve na academia.

Outras maneiras incluem Arthur chegando um pouco antes ou saindo um pouco depois, mas isso não interfere na conclusão a seguir. O fato é que Bruno e Diego estiverem com ele, mas em horários diferentes. Note que é impossível encaixar Everton nessa linha de tempo, pois pelos depoimentos, ele deve encontrar Bruno e Diego, e não encontrar Arthur.

O mesmo ocorre se montarmos uma linha de tempo com Arthur, Bruno e Everton (impossível encaixar Diego), ou com Arthur, Diego e Everton (impossível encaixar Bruno), ou com Bruno, Diego e Everton (impossível encaixar Arthur). Logo, é impossível que todos eles estejam falando a verdade.

Essa situação ocorre mesmo desconsiderando o depoimento de Arthur ou de Everton. Desconsiderando o depoimento de Bruno ocorre situação semelhante com Arthur, Cesar, Diego e Everton. Porém, desconsiderando o depoimento de Diego tudo se encaixa. Logo, Diego é o mentiroso.

O problema é que o número de atletas brasileiros classificados para as olimpíadas está crescendo e fica cada vez mais difícil descobrir o mentiroso. Use seu espírito olímpico para ajudar o treinador fazendo um programa que descobre o atleta mentiroso!

Entrada

Há vários casos de teste.

Cada caso de teste começa com um inteiro N, que representa o número de atletas ($4 \leq N \leq 10$). Em seguida há N linhas, descrevendo quem viu quem. A i-ésima linha começa com um inteiro Q_i, a quantidade de atletas que o i-ésimo atleta viu na academia ($0 \leq Q_i \leq N - 1$), e em seguida, Q_i inteiros distintos, indicando quais são esses atletas. Os atletas são numerados de 1 a N. O primeiro caso de teste do exemplo a seguir é o do enunciado.

A entrada termina com $N = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo o número do atleta mentiroso (haverá no máximo um atleta mentiroso em cada caso de teste, e se houver será possível identificá-lo pelos depoimentos). Ou então a frase "`nenhum mentiroso`" caso os depoimentos sejam coerentes e nenhum mentiroso possa ser identificado a partir deles.

Exemplos

Entrada	Saída
5 2 2 3 2 1 5 1 2 2 1 5 2 3 4 6 2 2 5 2 3 5 2 2 4 1 5 2 3 6 1 1 6 2 2 4 2 1 6 2 5 6 2 2 3 2 1 6 2 3 4 0	4 nenhum mentiroso 5

Problema F. Computação em Nuvem

Nome do arquivo fonte: `nuvem.c`, `nuvem.cpp`, ou `nuvem.java`

O conceito de computação em nuvem tem se tornado muito popular nos últimos tempos. Um dos principais tipos de computação em nuvem é conhecido como IaaS (*Infrastructure as a Service*, em português Infraestrutura como Serviço), onde provedores disponibilizam servidores que podem ser alugados e gerenciados pela Internet.

A Cloud, Inc. é uma empresa que disponibiliza serviços de IaaS. Ela está projetando um novo *data center* para atender a seus clientes. Através de uma pesquisa, eles descobriram que seus clientes, como um todo, precisam de K servidores, cada um dos quais precisa ser capaz de suportar um certo nível de demanda. Se supormos que o custo de um servidor sempre cresce à medida que a demanda que aquele servidor deve suportar cresce, a melhor solução em termos de custo é comprar K servidores que sejam cada um explicitamente montados de forma a atender exatamente à demanda necessária.

Porém, ter K configurações diferentes de hardware no *data center* é extremamente problemático para os administradores de sistema. Para simplificar a administração, eles exigem que sejam comprados não mais do que L tipos diferentes de servidor. Um servidor que suporta uma certa demanda c também suporta qualquer demanda menor que c.

Uma solução possível é comprar apenas um tipo de servidor, que seja capaz de atender à maior demanda necessária, pois ele também será capaz de atender a todas as outras demandas. Porém, o custo dessa solução pode ser proibitivo. Considerando que você pode comprar até L tipos diferentes de servidor, provavelmente há uma solução melhor. Por exemplo, suponha que haja 3 clientes, com demandas 3, 7 e 16. Suponha que o custo de um servidor que suporta uma demanda até 3 é R$ 1.500, o custo de um servidor que suporta demanda 7 é R$ 5.500 e o custo de um servidor que suporta demanda 16 é de R$ 19.200. Se você quer comprar até 2 tipos de servidor para atender aos 3 clientes, há quatro opções:

- Comprar três servidores de capacidade 16, a um custo total de R$ 57.600;
- Comprar dois servidores de capacidade 16 e um servidor de capacidade 7, a um custo total de R$ 43.900;
- Comprar dois servidores de capacidade 16 e um servidor de capacidade 3, a um custo total de R$ 39.900;
- Comprar um servidor de capacidade 16 e dois servidores de capacidade 7, a um custo total de R$ 30.200.

Dentre essas opções, a que apresenta o menor custo é a última.

Você receberá uma lista de K demandas requisitadas e o preço de um servidor que suporta cada uma dessas demandas. Determine o menor preço total para comprar K servidores de forma tal que seja possível atingir a demanda requisitada e sejam comprados servidores de no máximo L tipos diferentes.

Observações

- Cada servidor atende a um e apenas um cliente. Um servidor de capacidade 4 **não** pode atender simultaneamente a dois clientes com demanda 2 cada.
- Sejam D_i e D_j duas demandas, e P_i e P_j os preços associados a servidores que sejam capazes de suprir essas demandas. Se $D_i < D_j$, então $P_i \leq P_j$.

Entrada

Há vários casos de teste.

Cada caso de teste começa com uma linha contendo dois inteiros K e L, respectivamente o número de servidores requisitados e o número máximo de tipos de servidor a serem comprados ($1 \leq L \leq K \leq 500$). Em seguida, há K linhas, cada uma das quais contendo dois inteiros D e P, respectivamente a demanda necessária e o menor preço de um servidor que é capaz de atender àquela demanda ($1 \leq D \leq 1000$, $1 \leq P \leq 100.000$). Se houver mais de uma linha com o mesmo valor de D, então essas linhas também terão mesmo valor de P.

A entrada termina com $K = L = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo um inteiro T, que representa o menor custo total que pode ser obtido.

Exemplos

Entrada	Saída
10 3 1 1 2 4 3 5 4 7 5 8 6 12 7 13 8 18 9 19 10 21 0 0	129

A melhor opção para comprar servidores de 3 tipos que atendam às dez demandas requisitadas é comprar:

- 3 servidores que suportam demanda 10, que vão cobrir os casos de demanda 10, 9 e 8;
- 2 servidores que suportam demanda 7, que vão cobrir os casos de demanda 7 e 6;
- 5 servidores que suportam demanda 5, que vão cobrir os demais casos.

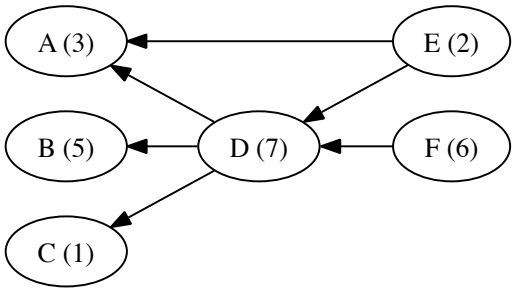
O custo total é $3 \cdot 21 + 2 \cdot 13 + 5 \cdot 8 = 129$.

Problema G. Agendamento de Tarefas

Nome do arquivo fonte: `tarefas.c`, `tarefas.cpp`, ou `tarefas.java`

A construção de um sistema de software complexo pode ser estruturada como um conjunto de *tarefas* logicamente relacionadas, como a análise de requisitos, o projeto da arquitetura do sistema, a implementação dos subcomponentes de software que formarão o sistema final, a escrita de documentação, a validação e outras. Algumas dessas tarefas só podem ser executadas depois que outras tarefas das quais elas dependem tenham sido concluídas, mas várias tarefas podem ser executadas em paralelo desde que elas não dependam umas das outras.

Suponha que sejam conhecidas todas as tarefas necessárias e, para cada tarefa, qual o tempo necessário para execução dela e quais são as outras tarefas das quais ela dependa. Suponha também que um número ilimitado de tarefas possam ser executadas em paralelo (desde que todas as dependências de cada uma delas já tenham sido satisfeitas). Uma pergunta óbvia é qual o menor tempo necessário para completar a construção do sistema de software (i.e., executar todas as tarefas). Considere o exemplo da figura abaixo, onde as tarefas são representadas por letras e uma seta de uma tarefa X para uma tarefa Y indica que X depende de Y (os números entre parênteses indicam o número de dias necessários para executar cada tarefa):



As tarefas A, B e C não possuem nenhuma dependência e podem ser executadas em paralelo. A tarefa D depende dessas três, e as tarefas E e F dependem da tarefa D, então nenhuma outra tarefa pode ser executada até que as tarefas A, B e C sejam concluídas.

Assim, é necessário esperar 5 dias até que a tarefa B esteja concluída. Quando isso ocorrer, as tarefas A e C também já estarão concluídas (porque elas foram executadas em paralelo) e então poderemos executar a tarefa D. Ao fim de mais 7 dias necessários para a conclusão da tarefa D, podemos começar a execução das tarefas E e F. A tarefa E estará concluída em 2 dias, mas a tarefa F levará 6 dias para ficar pronta. Logo, o tempo necessário para a execução de todas as tarefas é de $5 + 7 + 6 = 18$ dias.

No exemplo, assumimos que uma tarefa ia ser iniciada tão logo quanto todas as suas dependências estivessem concluídas. Por exemplo, a tarefa A foi iniciada no dia zero e concluída no dia 3, a tarefa D foi iniciada no dia 5 e concluída no dia 12 e a tarefa E (que depende apenas das tarefas A e D) foi iniciada no dia 12 e concluída no dia 14. Porém, nem sempre é necessário iniciar uma tarefa tão logo quanto possível para garantir que o tempo total de execução seja mínimo. Como vimos, o menor tempo de execução para o exemplo dado é de 18 dias. Como a tarefa E leva apenas 2 dias para ser concluída, poderíamos deixar para iniciá-la até o dia 16 sem alterar o prazo total. Da mesma forma, a tarefa A podia ser iniciada até no dia 2, pois ela dura 3 dias e todas as tarefas que dependem dela também dependem direta ou indiretamente da tarefa B, que só será concluída no dia 5. Essa flexibilidade na data de início pode ser interessante por diversos motivos. Por exemplo, se as três tarefas A, B e C forem todas iniciadas no dia 0, então seriam necessárias três equipes diferentes, uma para executar cada tarefa. Porém, se a

data de início da tarefa A for adiada para o dia 2, então as três tarefas poderiam ser executadas por apenas duas equipes: uma que executa a tarefa B, e outra que executa primeiro a tarefa C e depois a tarefa A.

Na empresa onde você trabalha, esse agendamento de tarefas é feito de forma manual. Por muito tempo, esse sistema funcionou bem, mas, agora, os projetos de software estão ficando mais complexos, com um número grande de tarefas, de forma que fica impraticável para um gerente realizar esse agendamento manualmente. Devido a isso, seu chefe pediu que você criasse um programa que, dada uma descrição das tarefas, suas durações e suas dependências, determine o prazo mínimo para conclusão de todas essas tarefas e, para cada tarefa, quais são o primeiro e o último dia no qual a tarefa pode ser iniciada de forma tal que seja possível concluir todas elas dentro desse prazo mínimo.

Entrada

Há vários casos de teste.

Cada caso de teste começa com um inteiro N, que representa o número de tarefas ($0 < N \leq 1000$). Cada uma das N linhas seguintes descreve uma tarefa, e vem no formato `<id> <duracao> <k> <dep 1> <dep 2> ... <dep k>`, onde:

- `<id>` é um inteiro que identifica aquela tarefa (entre 0 e $N-1$, inclusive; tarefas diferentes possuem identificadores diferentes);
- `<duracao>` é um inteiro que representa o número de dias necessário para executar aquela tarefa (entre 1 e 100, inclusive);
- `<k>` é o número de tarefas das quais aquela tarefa depende (entre 0 e $N-1$, inclusive);
- Os k inteiros `<dep i>` representam os identificadores das tarefas das quais essa tarefa depende.

Você pode supor que nenhuma tarefa depende dela mesma, nem direta nem indiretamente.

A entrada termina com $N = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha no formato `Prazo: X dias`, onde `X` é o menor número de dias necessário para concluir todas as tarefas. Em seguida, imprima uma linha para cada tarefa, no formato `Tarefa #I: min=A, max=B`, onde `I` é o identificador da tarefa, `A` é o índice do menor dia no qual essa tarefa pode ser iniciada e `B` é o índice do maior dia no qual essa tarefa pode ser iniciada (os dias são indexados a partir de zero). As tarefas devem ser impressas em ordem crescente de identificador.

Ao fim de cada caso de teste, inclusive o último, imprima uma linha contendo apenas 3 hífens: `---`

Exemplos

Entrada	Saída
6 0 3 0 1 5 0 2 1 0 3 7 3 0 1 2 4 2 2 0 3 5 6 1 3 0	Prazo: 18 dias Tarefa #0: min=0, max=2 Tarefa #1: min=0, max=0 Tarefa #2: min=0, max=4 Tarefa #3: min=5, max=5 Tarefa #4: min=12, max=16 Tarefa #5: min=12, max=12 ---

Problema H. Jogo da Vida

Nome do arquivo fonte: `vida.c`, `vida.cpp`, ou `vida.java`

Você provavelmente já ouviu falar de futebol, um jogo para 22 jogadores. Também provavelmente já ouviu falar de xadrez, que é um jogo para dois jogadores, e de paciência, um jogo para um único jogador. O jogo da vida é um jogo peculiar porque ele é um jogo para... zero jogadores.

O jogo da vida é composto por um tabuleiro 2D infinito, que possui infinitas colunas numeradas $(\ldots, -2, -1, 0, 1, 2, \ldots)$ e infinitas linhas também numeradas $(\ldots, -2, -1, 0, 1, 2, \ldots)$. Em cada posição do tabuleiro, há uma célula, que pode estar em dois estados: viva ou morta. O jogo começa em um dado estado inicial, que é completamente especificado por uma lista das células que estão vivas. A partir daí, há várias etapas. Em cada etapa, uma célula vai "nascer" (passar de morta para viva), morrer ou manter o estado atual, de acordo com os seus vizinhos. Os vizinhos de uma célula são as 8 células adjacentes, incluindo as diagonais. As regras que definem o que acontece com cada célula são:

1. Se uma célula viva possui menos de dois vizinhos vivos, ela morre de solidão.
2. Se uma célula viva possui mais de três vizinhos vivos, ela morre por superpopulação.
3. Uma célula viva que possua dois ou três vizinhos vivos continua viva.
4. Uma célula morta com exatamente três vizinhos vivos se torna uma célula viva.

Em cada etapa, todas as células atualizam seus estados de acordo com essas regras, simultaneamente.

Dada uma configuração inicial do jogo da vida, calcule e imprima a lista de células vivas após K etapas.

Entrada

Há vários casos de teste.

Cada caso de teste começa com um inteiro N, que representa o número de células vivas na configuração inicial ($0 < N \leq 1000$). Em seguida, há N linhas, cada uma das quais descrevendo uma célula viva. Essas linhas possuem dois inteiros L e C que indicam a linha e a coluna, respectivamente, de uma célula viva ($-1048576 < L, C < 1048576$). Você pode supor que a mesma célula não aparece duas vezes nessa lista. Por fim, a última linha do caso de teste possui um único inteiro K, que representa o número de etapas ($0 < K \leq 100$).

A entrada termina com $N = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo um inteiro C, o número de células vivas após K etapas. Em seguida, imprima C linhas, contendo as coordenadas L e C das células vivas. As células devem ser impressas ordenadas pelo número da linha. Células que possuem o mesmo número de linha devem ser ordenadas pelo número da coluna.

Exemplos

Entrada	Saída
3 -1 0 -1 1 0 0 1 0	4 -1 0 -1 1 0 0 0 1

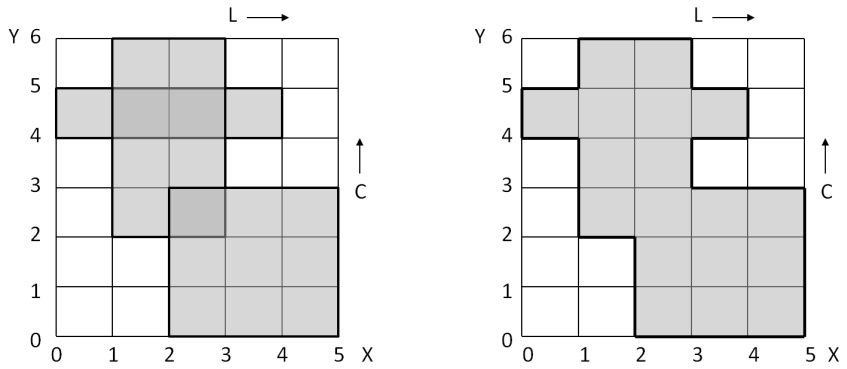
Problema I. Fazenda

Nome do arquivo fonte: `fazenda.c`, `fazenda.cpp`, ou `fazenda.java`

Compadre Roberto teve uma agradável surpresa: recebeu um telefonema de sua irmã dizendo que eles eram os únicos herdeiros das terras dos seus tios. Logo em seguida recebeu um fax com a descrição dos terrenos. Todos eram retangulares, e pela largura e comprimento de cada um concluiu que herdou uma grande área de terra no triângulo mineiro! Não tardou e Compadre Roberto já estava lá para tomar posse dos terrenos.

Chegando lá, uma surpresa nada agradável... os terrenos eram sobrepostos!! Ninguém conseguiu explicar como isso aconteceu, mas boa parte das terras dos seus tios não pertenciam a apenas um, mas a dois ou mais! Isso nunca foi problema para eles, que cuidavam de tudo como uma grande família, sem se preocupar exatamente que pedaço era de quem. Tampouco é problema para Compadre Roberto e sua irmã, que agora são os únicos donos. Mas é frustrante descobrir que o terreno não era tudo o que pensava...

A figura abaixo mostra um exemplo com 3 terrenos. Os terrenos são descritos com quatro números inteiros X, Y, L, C, indicando que o canto sudoeste do terreno (na figura, o canto inferior esquerdo) está na coordenada (X, Y) e ele tem largura L e comprimento C (todos os valores dados em Km). A descrição dos 3 terrenos mostrados é: Terreno A: 1 2 2 4; Terreno B: 2 0 3 3; Terreno C: 0 4 4 1. A figura da esquerda mostra os terrenos, e a da direita o terreno total resultado da sobreposição.

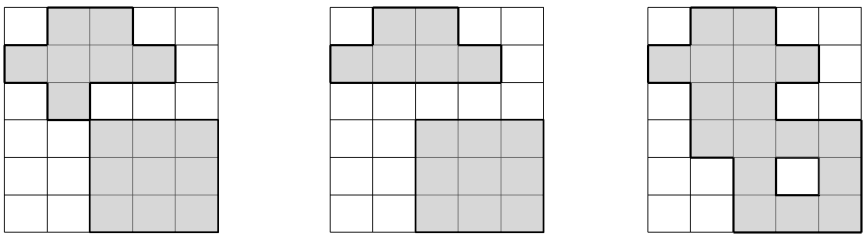


Quando Compadre Roberto recebeu o fax, achava que esses terrenos tinham $2 \times 4 + 3 \times 3 + 4 \times 1 = 21 Km^2$ no total. Agora sabe que são $18 Km^2$...

Ele gostaria de saber quantos Km^2 os terrenos realmente contêm. E com as muitas sobreposições precisa da sua ajuda. Além disso precisa cercar o terreno, então gostaria de saber também quantos Km de cerca são necessários. No exemplo acima a cerca tem $24 Km$ de comprimento.

Observações

A junção dos terrenos nem sempre forma uma região contígua e sem "buracos". Casos como os mostrados na figura abaixo podem acontecer. Então nem sempre será possível cercar o terreno herdado com uma única cerca contígua, pode ser necessário usar mais de uma.



Entrada

Há vários casos de teste.

A primeira linha de um caso de teste contém um número inteiro N que é o número de terrenos herdados ($N \leq 20$). As N linhas seguintes contém cada uma a descrição de um terreno no formato $X\ Y\ L\ C$ conforme descrito no enunciado ($0 \leq X, Y \leq 1000$, e $1 \leq L, C \leq 500$).

A entrada termina quando $N = 0$

Saída

Para cada caso de teste da entrada seu programa deve produzir uma linha na saída contendo dois valores A e P, que são a área (em Km^2) e o perímetro (em Km) total da união dos terrenos descritos no caso de teste.

Exemplos

Entrada	Saída
3 1 2 2 4 2 0 3 3 0 4 4 1 5 1 2 1 1 1 2 5 1 2 5 4 2 4 0 1 6 1 0 2 4 4 0 0 1 4 0 3 4 1 3 0 1 4 1 0 2 1 0	18 24 23 36 12 24

Problema J. Macaco Rural

Nome do arquivo fonte: `macaco.c`, `macaco.cpp`, ou `macaco.java`

Você foi contratado como programador para um novo website de compras coletivas chamado *Macaco Rural*. Para se diferenciar dos seus vários concorrentes, esse site planeja oferecer ofertas para *pares* de produtos. Por exemplo, "compre uma bola de futebol e uma camisa oficial da seleção brasileira com 75% de desconto".

O site quer planejar as ofertas do dia para os próximos n dias. Para tanto, há uma lista de $2n$ produtos, com seus respectivos preços (já com os descontos aplicados). Como ofertas muito caras vendem menos, seu chefe quer minimizar o preço da oferta mais cara. Cabe a você agrupar os $2n$ produtos em n pares de forma tal que o custo do par mais caro seja minimizado. O custo de um par é a soma dos custos dos produtos que o compõem.

Entrada

Há vários casos de teste.

Cada caso de teste começa com uma linha que contém um único inteiro N, o número de produtos que serão usados para criar as ofertas ($1 \leq N \leq 2.000.000$, e N é sempre um número par). Em seguida, há uma linha contendo N inteiros $P_1, P_2, \ldots, P_N$, que representam os preços dos N produtos que serão pareados em $N/2$ ofertas ($0 \leq P_i \leq 1.000.000.000$, para todo i).

A entrada termina com $N = 0$, que não deve ser processado.

Saída

Para cada caso de teste, imprima uma linha contendo um único inteiro, que é o maior preço de uma oferta, quando os N produtos são pareados de forma tal a minimizar esse maior preço.

Exemplos

Entrada	Saída
4 1 19 26 17 8 3 9 6 18 14 1 7 8 0	36 19

No primeiro caso de teste, há 3 possibilidades:

- $(1, 19), (17, 26)$, com custos $1 + 19 = 20$ e $17 + 26 = 43$. O maior custo é 43.
- $(1, 26), (17, 19)$, com custos $1 + 26 = 27$ e $17 + 19 = 36$. O maior custo é 36.
- $(1, 17), (19, 26)$, com custos $1 + 17 = 18$ e $19 + 26 = 45$. O maior custo é 45.

Dessas três opções, a que minimiza o maior custo é a segunda, que leva a um custo máximo de 36.

Problema K. Cinefilândia

Nome do arquivo fonte: `cine.c`, `cine.cpp`, ou `cine.java`

Na pacata cidade interiorana de Cinefilândia, o assunto predileto dos moradores é, obviamente, filmes. Para facilitar a conversação, os moradores referem-se aos filmes somente por suas siglas. O filme *The Lord of the Rings*, por exemplo, é tratado simplesmente por LotR.

Como Cinefilândia está entrando para rotas turísticas por causa de suas famosas poças de lama anti-reumáticas, a prefeitura está montando um guia para turistas e contratou você para gerar as siglas dos filmes de forma computadorizada. A regra é simples: o primeiro caractere de cada palavra do nome do filme é usado na sigla na forma maiúscula, exceto para palavras da lista especial. Para estas palavras, se forem a primeira do nome, a palavra é siplesmente ignorada; e, caso não sejam a primeira, tem seu caractere correspondente na forma minúscula.

Entrada

A entrada dos dados inicia-se com uma linha com um inteiro N correspondente ao número de casos de teste. Para cada caso de testes, segue uma linha com os inteiros E e F ($1 \leq E, F \leq 100$), seguida por uma linha com E palavras especiais, com no máximo 10 caracteres cada, e por F linhas com os nomes dos filmes, com no máximo 100 caracteres cada. Palavras, especiais ou não, são formadas por caracteres minúsculos e maiúsculos e dígitos, e podem ser separadas por hífens (-) e (:), além de espaços.

Na entrada, caracteres maiúsculos ou minúsculos são considerados iguais. Isto é, as palavras "harry" e "hArrY" são consideradas iguais.

Saída

Para cada nome de filme, seu programa deve gerar a sigla correspondente, com as primeiras letras de cada palavra. Palavras especiais tem sua letra correspondente na sigla na forma minúscula e as demais palavras tem sua letra correspondente na sigla na forma maiúscula. Além disso, se a primeira palavra do nome for uma palavra especial, então ela deve ser ignorada. Atenção: se as duas primeiras palavras forem especiais, então só a primeira será ignorada.

Todos os casos de teste geram alguma sigla.

Após cada caso de teste (inclusive o último), imprima uma linha em branco.

Exemplos

Entrada	Saída
2 11 3 The and an E ou of De O da dos Das The Lord of The Rings O Senhor Dos Aneis E o Vento Levou 2 3 no in Xisto no Espaco 3 Rambo 2: A missao O imperio contra-ataca	LotR SdA oVL XnE3 R2AM OICA