7ª Maratona de Programação

Facens 2011

CADERNO DE PROBLEMAS

PROBLEMA A: O PASTOR E O CONSULTOR

Arquivo: `proba.[c/cpp/pas/java]`

Cor: `Branco`

Limite de tempo: `4s`

Descrição do problema

Sendo um programador fanático, você decide adquirir um pouco mais de cultura geral, e começar a ler alguns contos (na internet, é claro!). Logo de cara você se depara com o clássico conto do pastor e do consultor:

"Um pastor cuidava das suas ovelhas, quando avistou no horizonte um possante Jeep vindo em sua direção. Ao chegar perto do pastor, o Jeep parou. Desce um homem muito bem vestido que sem mais nem menos, pergunta:

- Se eu lhe disser quantas ovelhas existem aqui você me daria uma?

O pastor, humildemente acena que sim. Rapidamente o homem saca seu notebook, acessa mapas a partir de satélites, programas de cálculo geométrico e responde:

- Trezentos e trinta e cinco.

O pastor, admirado, acena com a cabeça confirmando o número e permite que o homem escolha uma ovelha. Após escolha criteriosa, o homem prepara-se para partir, mas o pastor o interrompe:

- Se eu adivinhar sua profissão posso ter a ovelha de volta?

- Sim – responde com confiança o homem.

- O senhor é consultor!

Estupefato, o homem devolve a ovelha ao pastor, mas não sem antes perguntar como ele adivinhou.

- Simples: o senhor chegou aqui sem ser chamado, me cobrou o olho da cara para dizer o que eu já sei e não entende nada do meu negócio! Agora por favor devolva meu cachorro!"

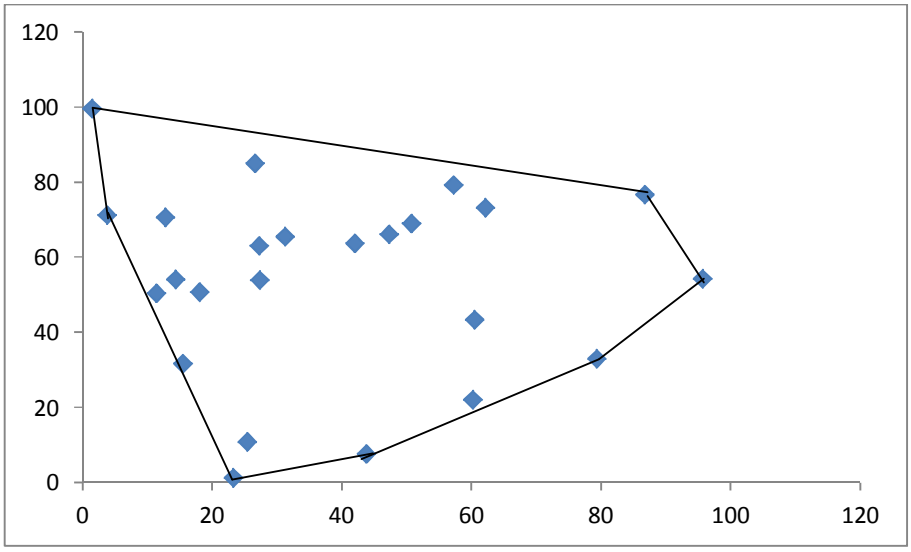

Figura 1 – Exemplo de cerca dadas as posições das ovelhas.

Visivelmente sensibilizado pela oefnsa ao consultor, você decide fazer parte do enredo. Você não quer que o consultor seja um inútil, então pensou em algo mais útil que contar as ovelhas: cercá-las! Sua primeira idéia é criar um programa que, dadas as posições das ovelhas, retorne os pontos que compõe o menor polígono convexo contendo todas as ovelhas. Um exemplo de cerca é apresentado na Figura 1.

Entrada

A entrada é constituída de vários casos de testes. Cada caso de teste começa com um valor N, que corresponde ao número de ovelhas ($3 \leq N \leq 1000$). Em seguida, são informados N pontos (X,Y) indicando as posições das ovelhas ($0 \leq X,Y \leq 5000$). A execução deve ser interrompida quando for encontrado um caso de teste com $N = 0$.

Saída

A saída deverá conter uma linha para cada caso de teste, com as coordenadas dos pontos que formam a cerca. Os pontos devem ser ordenados em ordem crescente (primeiro X depois Y).

Exemplos de testes

Entrada	**Saída**
5 0 10 0 0 10 10 10 0 5 5 10 6 5 60 18 9 87 61 84 50 40 67 30 10 49 61 36 84 53 29 94 4 0 1 0 2 0 3 0 4 0	0,0 0,10 10,0 10,10 6,5 9,87 29,94 60,18 61,84 84,53 0,1 0,2 0,3 0,4

PROBLEMA B: IMPLEMENTAÇÃO DO RAID 5

Arquivo: `probb.[c/cpp/pas/java]`

Cor: `Roxo`

Limite de tempo: `30s`

Descrição do problema

RAID (Redundant Array of Inexpensive Disks - Conjunto Redundante de Discos Econômicos) é um meio de se criar um sub-sistema de armazenamento composto por vários discos individuais com a finalidade de ganhar segurança e desempenho. RAID seriam compostos por dois ou mais discos trabalhando simultaneamente para um mesmo fim.

Há várias implementações possíveis de RAID dividida em alguns modos:

- **RAID 0** - É uma simples concatenação de partições para criar uma grande partição virtual.
- **RAID 1** - é o nível de RAID que implementa o espelhamento de disco, também conhecido como *mirror*.
- **RAID 2** – trabalha com controle de erro ECC
- **RAID 3** – é uma versão simplificada do RAID nível 2.
- **RAID 4** – funciona com dois ou mais discos iguais. Um dos discos guarda a paridade (uma forma de soma de segurança) da informação contida nos discos.
- **RAID 5** – funciona similarmente ao RAID 4, mas supera alguns dos problemas mais comuns sofridos por esse tipo. As informações sobre paridade para os dados do conjunto são distribuídas ao longo de todos os discos do conjunto, ao invés de serem armazenadas num disco dedicado, oferecendo assim mais desempenho que o RAID 4, e, simultaneamente, tolerância a falhas.

Uma empresa precisa elaborar o *software* que vai operar em uma controladora de discos com suporte a RAID 5. Para isso, a princípio você deve elaborar um algoritmo simplificado que será utilizado em simulações e testes.

O funcionamento do RAID 5 é o seguinte. Para cada faixa de dados, um disco é usado para armazenar a paridade calculada para aquela faixa, como mostra a figura.

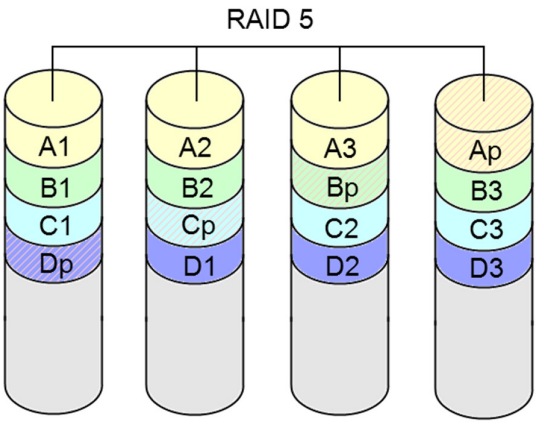

Figura 1: RAID 5

Na primeira faixa de dados (A), o quarto disco é usado para armazenar a paridade .

Na segunda faixa (B), o terceiro disco é usado para paridade, e assim sucessivamente

Para a simulação, cada bloco de dados será composta de 4 bits, e o conjunto será composto de 5 discos.

Para calcular o bloco de paridade, segue o exemplo:

Dados a serem gravados em hexadecimal: A2B7 (cada algarismo corresponde a um bloco de um disco)

Bloco 1: $A_{(16)} = 1010_{(2)}$
Bloco 2: $2_{(16)} = 0010_{(2)}$
Bloco 3: $B_{(16)} = 1011_{(2)}$
Bloco 4: $7_{(16)} = 0111_{(2)}$

Para cada conjunto de bits, a paridade é obtida da seguinte forma. Somam-se os bits. Se o resultado é PAR, a paridade é 0. Se o resultado é ÍMPAR, a paridade é 1.

```
1010
0010
1011
0111
————
0100 ← Bloco de paridade para a faixa específica.
```

$0100_{(2)} = 4_{(16)}$

Portanto, para o conjunto de blocos **A2B7**, o bloco de paridade é **4**.

Bloco p: $4_{(16)} = 0100_{(2)}$

No simulador, serão considerados 5 discos e 16 faixas de gravação. A tabela a seguir demonstra a posição dos blocos de paridade (P) para cada faixa:

Discos	**0**	**1**	**2**	**3**	**4**
Faixa 0					P
Faixa 1				P	
Faixa 2			P		
Faixa 3		P			
Faixa 4	P				
Faixa 5					P
Faixa 6				P	
Faixa 7			P		
Faixa 8		P			
Faixa 9	P				
Faixa 10					P
Faixa 11				P	
Faixa 12			P		
Faixa 13		P			
Faixa 14	P				
Faixa 15					P

Tabela1: Posição dos Blocos de Paridade

Esse simulador aceitara vários comandos de entrada, tais como escrita de dados (W), leitura de dados (R), limpeza de bloco (C), saída (Q) e importação (I) que simula a existência de dados nos discos. Inicialmente, considere que todas as faixas estão livres, isto é, todos os discos estão vazios (preenchidos com 0). Teremos as operações:

Operação de importação (I): recebe o valor dos 5 blocos e grava na próxima faixa livre, marcando a mesma como ocupada, sem se preocupar com paridade. Retorna a posição gravada e o valor de cada bloco na ordem ou 'CHEIO' no caso de todas as faixas estarem ocupadas:

Ex: `Entrada: IA2B27 Saída: 01A2B27`

Operação de gravação (W): recebe o valor dos 4 blocos de dados, calcula a paridade e grava os dados na próxima faixa livre, marcando a mesma como ocupada, nos discos corretos, de acordo com a posição da faixa. Retorna a posição gravada e o valor de cada bloco na ordem ou 'CHEIO' no caso de todas as faixas estarem ocupadas:

Ex: `Entrada: WA2B7 Saída: 02A24B7`

Operação de leitura (R): recebe o número da faixa a ser lida, faz a leitura e verifica se a paridade está correta. Retorna os blocos lidos. Caso a paridade esteja errada, apresenta a palavra 'FALHA' na frente dos dados.

Ex: `Entrada: R02 Saída: A2B7`

Operação de limpeza (C): recebe o número da faixa a ser limpa, zera todos os discos e marca a faixa como livre.

Ex: `Entrada: C01 Saída: 0100000`

Operação de saída (Q): encerra a execução do programa.

Ex: `Entrada: Q Saída:`

Entrada

A entrada é sempre composta de uma letra maiúscula que representa a operação, seguida do parâmetro esperado por essa operação. Para a operação de importação, o parâmetro é composto de 5 dígitos hexadecimais. Para a operação de escrita, o parâmetro é composto de 4 dígitos hexadecimais. Para as operações de leitura e limpeza, o parâmetro é composto de 2 dígitos decimais.

Para a operação de saída, não há parâmetros

Saída

Para as operações de importação (I) e escrita (W), a saída é composta de 2 dígitos decimais (faixa) e 5 hexadecimais (blocos) ou 'CHEIO' no caso do disco estar cheio. Para a operação de leitura (R) a saída é composta de 4 dígitos hexadecimais e, no caso de erro de paridade, seguido da palavra 'FALHA'. Para a operação de limpeza (C), a saída é composta de 2 dígitos decimais (faixa) e 5 dígitos zeros. Para a operação de saída (Q), não há saídas

Exemplos de testes

Entrada	**Saída**
I12344	0012344
I63207	0163207
IFA3CA	02FA3CA
IA3AAA	03A3AAA
I0BBBB	040BBBB
ICCCC0	05CCCC0
IDDD0D	06DDD0D
IEE2EE	07EE2EE
IF0FFF	08F0FFF
I0ABCD	090ABCD
I8777F	108777F
I98765	1198765
IAB6BC	12AB6BC
IC0DBA	13C0DBA
I11101	1411101
I20222	1520222
I00000	CHEIO
R00	1234
R03	AAAA-FALHA
R05	CCCC
R07	EEEE-FALHA
R09	ABCD
R11	9875-FALHA
C05	0500000
C02	0200000
C07	0700000
R05	0000
R07	0000
WCAC0	02CAAC0
W4359	054359B
WF123	07F1F23
W0201	CHEIO
C10	1000000
W0201	1002013
W1312	CHEIO
R02	CAC0
R05	4359
R07	F123
Q	

PROBLEMA C: EXPRESSÕES REGULARES

Arquivo: `probc.[c/cpp/pas/java]`

Cor: `Laranja`

Limite de tempo: `2s`

Descrição do problema

As linguagens são divididas em classes, onde a classe “universo de todas as linguagens” engloba as “linguagens livres de contexto”, na qual se encontram várias linguagens de programação e possui como subclasse as linguagens regulares. As linguagens regulares pertencem à classe das linguagens mais simples, onde as cadeias podem ser reconhecidas através de algoritmos com pouca complexidade. Nessa classe de linguagens a expressão regular é considerada geradora, pois pode-se dizer como construir as palavras desta classe de linguagem.

Sua equipe foi contratada para implementar um programa que valida palavras baseadas em expressões regulares.

A tabela abaixo lista os metacaracteres que devem ser reconhecidos nas expressões regulares.

Metacaractere	Nome	Função
.	Ponto	Um caractere qualquer. É um curinga
?	Opcional	Pode ocorrer ou não a entidade anterior, para ele é suficiente a ocorrência de um, mas pode ter nenhuma ou várias vezes.
|	Ou	Ou um ou outro
+	Mais	Uma ou mais vezes
{n, m}	Chaves	De n até m vezes
[ ]	Lista	Entre colchetes estabelece-se qual a lista de caracteres permitidos, pode-se utilizar – para estabelecer intervalo, exemplo a-d é igual a: a|b|c|d

Entrada

A entrada é constituída de vários casos de teste. Cada caso possui uma expressão regular e em seguida a quantidade de palavras que serão testadas para aquela linguagem. O programa deverá ser encerrado quando no local da expressão regular for informado o símbolo $.

Saída

A saída será composta da seguinte maneira: para cada palavra de entrada testada será gerada como saída a palavra seguida pelo texto: e um valor valido. ou nao e um valor valido.

Exemplos de testes

Entrada	**Saída**
[a-z] 3 a9 34 AF a|b 4 fa abbbb re 3a [0-9] 3 t4 g 975 $	a9 e um valor valido. 34 nao e um valor valido. AF nao e um valor valido. fa e um valor valido. abbbb e um valor valido. re nao e um valor valido. 3a e um valor valido. t4 e um valor valido. g nao e um valor valido. 975 e um valor valido.

PROBLEMA D: EM CIMA DO MURO

Arquivo: `probd.[c/cpp/pas/java]`

Cor: `Rosa`

Limite de tempo: `2s`

Descrição do problema

Na FACPOS, Faculdade de Politicagem de Sorocaba, os alunos estudam cinco anos como angariar votos do eleitorado com argumentos sinceros. A FACPOS tem um eficiente método de avaliação de conclusão de curso, três candidatos disputam os votos da banca e o vencedor é diplomado.

O problema é que, além de aprovar o aluno, a FACPOS precisa rotular o futuro político: ele deve ser de “esquerda” ou “direita”. Com dificuldades para realizar esta classificação, a direção da FACPOS resolveu pegar os trabalhos que os alunos fizeram durante o curso e classificá-los como sendo de esquerda ou de direita. Sendo assim, ao somar todos os trabalhos de direita ou de esquerda, seria possível classificar o aluno corretamente.

Verificar simplesmente a quantidade maior de trabalhos seria simples, mas a esquerda no Brasil nem sempre é esquerda e a direita nem sempre é direita. Sendo assim, para cada aluno os diretores da FACPOS decidiram estabelecer uma referência, uma linha, que determina o que seria esquerda e o que seria direita.

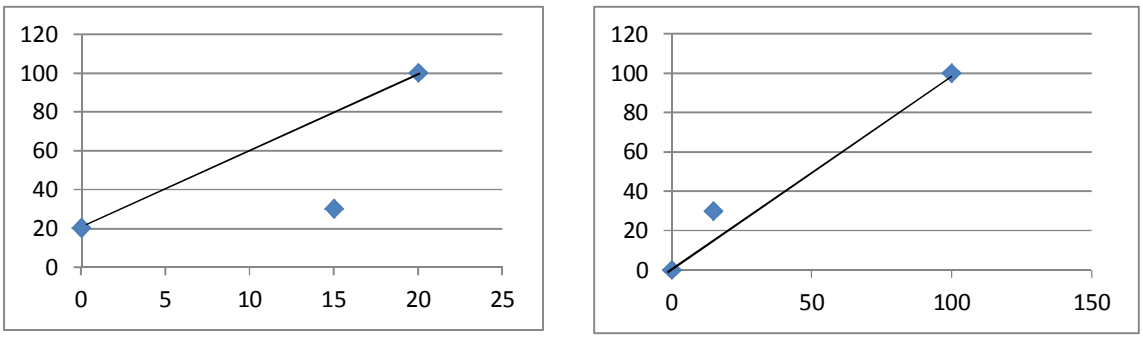

Figura 2 – Exemplo de mesmo candidato com classificações diferentes para referências diferentes.

A Figura 2 mostra um candidato que fez 15 trabalhos de direita e 30 trabalhos de esquerda, classificado como de esquerda ou direita, de acordo com a referência (reta) dada. A referência, sendo uma reta, pode ser identificada apenas por dois pontos.

A alta cúpula da FACPOS convenceu ardilosamente a alta cúpula da FACENS a criar um programa que, dada a referência e os números de trabalhos dos candidatos, realize a classificação. A FACENS ainda concordou em fazer o

trabalho de graça, pois foi convencida que assim está construindo um país melhor.

Entrada

A entrada é constituída de vários casos de teste. Cada caso possui três pares de coordenadas (X,Y) representando os dois pontos de que definem a reta de referência, e em seguida o número de trabalhos de esquerda e de direita que um aspirante a político realizou no curso ($0 \leq X,Y \leq 5000$). O programa encerra ao encontrar um valor de X negativo.

Saída

A saída será composta de uma única linha que determina se o aluno será um político de esquerda, de direita ou se ele está em cima do muro.

Exemplos de testes

Entrada	**Saída**
0 0 1 1 2 2 0 10 10 10 5 5 0 0 100 100 0 100 -1	Em cima do muro. Direita! Esquerda!

PROBLEMA E: PASSAM AS HORAS

Arquivo: `probe.[c/cpp/pas/java]`

Cor: `Azul Escuro`

Limite de tempo: `2s`

Descrição do problema

Todo mundo um dia já teve um relógio, digital ou análogico, dotado de cronógrafo. O termo cronógrafo remete a aparelhos que conseguem medir uma passagem de tempo. Apenas para curiosidade, cronômetros são cronógrafos que possuem o certificado do COSC, que é um laboratório que executa uma série de testes no relógio para identificar a precisão do cronômetro.

Os relógios analógicos possuem também cronógrafos, exibindo a medida em 3 mostradores diferentes, que exibem décimos de segundo, segundos e minutos.

A FACENS têm trabalhado para desenvolver um relógio analógico, aplicando os conceitos de física e mecânica e para isso precisa de sua ajuda para criar um programa capaz de processar algumas informações relacionadas ao relógio.

Entrada

Inicialmente, é informado um número N ($0 < N \leq 1000$) contendo o número de casos de teste. Para cada caso de teste será informado um caracter C (a, d), indicando o tipo de operação a executar. Quando a operação for igual a 'a', será informado um decimal D ($0 \leq D \leq 7200$) indicando o tempo em segundos que foi medido. Caso a operação seja igual a 'd', serão informado 3 valores M, S e D indicando o ângulo dos 3 ponteiros: ($0 \leq M,S,D \leq 360$), respectivamente o ângulo do ponteiro do mostrador dos minutos, segundos e décimos de segundo.

Saída

Quando a operação selecionada for 'a', a saída deverá exibir o ângulo dos 3 ponteiros na respectiva ordem: minutos, segundos e décimos de segundo. Quando a operação selecionada for 'd', a saída deverá exibir a quantidade em segundos do tempo medido.

Exemplos de testes

Entrada	**Saída**
2 a 147.7 d 12 162 252	12 162 252 147.7

PROBLEMA F: LIMITE DE VELOCIDADE

Arquivo: `probf.[c/cpp/pas/java]`

Cor: `Verde Claro`

Limite de tempo: `2s`

Descrição do problema

O transito no Brasil é um caos em algumas regiões e isso não é nenhuma surpresa, porém é muito difícil entender em que ponto começam as lentidões, se determinados trechos são lentos ou não.

A companhia de engenharia de tráfego apresentou para a prefeitura de Sorocaba uma idéia para medir os trechos de lentidão nas pistas da cidade. O engenheiro responsável pelo projeto propôs escolher as avenidas com maior tráfego de veículos e posicionar um determinado número de radares separados por uma mesma distância, utilizando estas informações para identificar onde estão os trechos de lentidão.

Os dados obtidos dos radares serão enviados a você, que deve preparar uma análise do tráfego de acordo com as entradas e saídas de exemplo.

Uma via Lenta é quando a distância total percorrida com velocidade média decrescente é predominante. Uma via Rápida é quando a distância total percorrida com velocidade média crescente é predominante. Em caso de igualdade das distâncias, a via é Normal.

Entrada

Um inteiro R ($0 \leq R \leq 1000$) contendo o número de radares, um inteiro D ($0 \leq D \leq 1000$) representando a distância em metros entre cada radar e para cada radar um inteiro V ($0 \leq V \leq 200$) contendo a velocidade média dos veículos medida pelo radar. O valor -1 para R identifica que o programa deve ser encerrado.

Saída

Para cada caso de teste, serão apresentadas 3 linhas na saída. A primeira linha irá conter a classificação da via “Lento”, “Normal” ou “Rapido”. A segunda linha apresenta a distância total percorrida com velocidade média decrescente, e a terceira linha apresenta a distância total percorrida com velocidade média crescente.

Exemplos de testes

Entrada	**Saída**
5 50 30 40 50 30 10 4 45 10 20 30 40 5 10 30 40 40 20 50 -1	Normal 100 100 Rapido 0 135 Rapido 10 20

PROBLEMA G: HOUGH LINE

Arquivo: `probg.[c/cpp/pas/java]`

Cor: `Preto`

Limite de tempo: `1s`

Descrição do problema

A transformada de Hough (Hough, 1962) é um método utilizado para detectar em uma imagem digital uma classe de formas geométricas conhecida e que pode ser representada como uma curva paramétrica, como por exemplo retas, círculos e elipses.

Essa transformada realiza um mapeamento entre o espaço cartesiano da imagem e o espaço de parâmetros em que a curva foi definida. Dessa forma, os pontos da curva procurada são concentrados no espaço de parâmetros de acordo com as características que os unem no espaço cartesiano. No caso de retas, um conjunto de pontos colineares no espaço cartesiano gerará uma acumulação em um ponto no espaço de parâmetros. Esse ponto acumulador define os parâmetros da reta que passa por esses pontos cartesianos.

Hough para Retas

Uma possível parametrização para uma reta no plano X-Y é dada pela equação 1:

$$y = ax + b \quad (1)$$

onde os parâmetros a e b indicam, respectivamente, a inclinação da reta com relação ao eixo X e o ponto onde a reta intercepta o eixo Y. Cada reta no plano cartesiano X-Y, pode ser representada por um único ponto no plano A-B (plano de parâmetros), correspondente aos parâmetros dessa reta. A transformada de Hough consiste em calcular, para cada ponto analisado, os parâmetros de todas as possíveis retas que passam por esse ponto. Cada uma dessas retas é calculada de acordo com a equação 2:

$$b = y - ax \quad (2)$$

Com isso, para cada ponto analisado no plano cartesiano, o conjunto de todos os pares de parâmetros calculados formará uma reta no plano de parâmetros. Repetindo esse processo para todos os pontos do plano cartesiano, obteremos várias retas no plano de parâmetros (uma para cada ponto do plano cartesiano). A interseção dessas retas, indicará uma co-linearidade entre pontos no plano cartesiano, sendo que o ponto de interseção corresponderá aos parâmetros a e b da reta que passa por esses pontos co-lineraes.

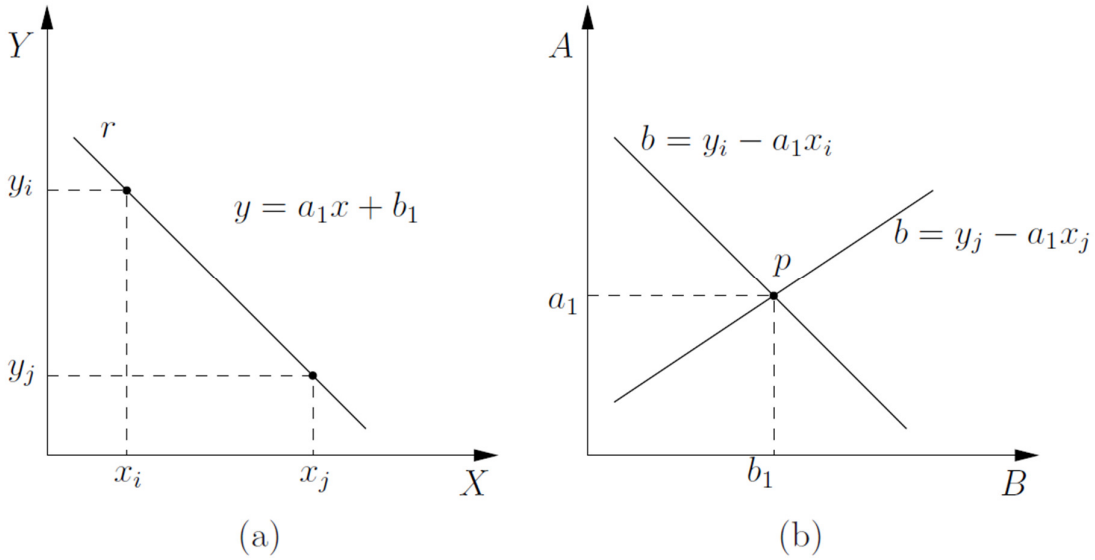

Figura 3 - (a) reta no plano cartesiano (b)interseção das retas no plano de parâmetros

A representação de uma reta em função dos parâmetros a e b mostrada na equação 1 não é adequada para implementar computacionalmente a transformada de Hough, já que esses parâmetros podem assumir infinitos valores. Isso pode ser verificado quando uma reta é paralela ao eixo Y, onde o valor de a tende a infinito. Para solucionar esse problema, foi proposta a utilização de outra representação para a reta por Duda e Hart em 1972. A representação adotada foi a representação normal (coordenada polar), dada pela equação 3:

$$\rho = xcos\theta + ysen\theta \quad (3)$$

A representação normal proporciona um espaço de parâmetros finito, com seus limites conhecidos. Os parâmetros ρ e Θ representam, respectivamente, o comprimento do vetor, que passa pela origem, normal a reta e o ângulo formado entre esse vetor e o eixo X, conforme mostra a figura 2. Utilizando essa representação, não teremos mais a formação de retas no espaço de parâmetros e sim de curvas senoidais. Um conjunto de pontos colineares no plano cartesiano gerará um conjunto de senóides no espaço de parâmetros, que terão um ponto de interseção que identifica a presença da reta que une os pontos no espaço cartesiano.

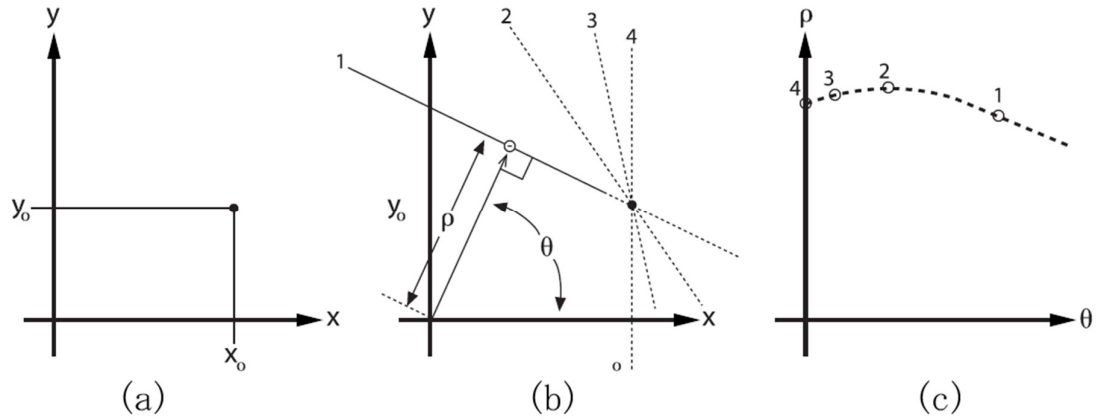

Figura 4 – um ponto no plano cartesiano (a) implica em diversas retas para diferentes ρ e Θ (b). Cada uma dessas retas geram ponto no plano (ρ, Θ) que unidos formam uma curva característica

Tarefa

Considerando que um sistema pra detecção de linhas de uma quadra de basquete retornou várias coordenadas ρ e Θ para as linhas detectadas, seu trabalho é identificar possíveis limites da quadra, ou seja, se duas linhas são paralelas ou se possuem interseção. No caso de possuírem interseção, deve-se definir qual o ponto de interseção das linhas no plano cartesiano (X-Y), mesmo que seja fora da imagem.

Entrada

A entrada é composta de vários casos de testes, sendo que em cada linha existirão as coordenadas ρ (decimal com precisão dupla casas variando de 0.0 a 1000.0) e Θ (0 < Θ < 180) da reta detectada pela transformada de hough (representação normal). Uma entrada é representada por duas linhas, cada uma com os dados ρ e Θ.

Os casos de testes terminam quando as dimensões da matriz forem -1 -1.

Saída

Para cada caso de teste, a saída deverá conter a mensagem ‘Paralelas’, quando as retas forem paralelas ou as coordenadas onde as retas se cruzam: [x, y]. Os valores de x e y são inteiros arredondados sempre para o próximo inteiro subsequente, ou seja, se um dos valores for -1.13 deverá ser arredondado para -2.

Exemplos de Testes

Entrada	**Saída**
10 0 5 0 10 90 3 45 8 50 7 84 2 180 1 90 -1 -1	Paralelas [-6, 10] [5, 7] [-2, 2]

PROBLEMA H: PERFUME

Arquivo: `probh.[c/cpp/pas/java]`

Cor: `Amarelo`

Limite de tempo: `2s`

Descrição do problema

Cerca de 600 AC, com o esgotamento do vedismo na Índia, difundiram-se duas outras concepções religiosas, o budismo e o jainismo.

A palavra jaina vem de jin, vitorioso em sânscrito, e indica *aqueles que obtiveram vitória sobre os desejos mundanos e que tem os sentidos totalmente sob o controle da vontade*. Para atingir essa perfeição, os jainas passavam por um longo treinamento, sendo que o estudo da Ganitanuyoga, ou Matemática, era considerado como um dos exercícios mais nobres e eficazes do mesmo.

Entre os vários temas matemáticos estudados pelos jainas estava a Vikalpa, ou Combinatória. A razão maior da grande atenção que deram à Combinatória era sua concepção atomística do mundo físico.

O **Triângulo de Pascal** muito utilizado em **Análise Combinatória** recebe esse nome devido ao matemático **Blaise Pascal** (1623-1662). A forma mais usual de construir um triângulo de Pascal é na forma de um triângulo isósceles.

Dentre as diversas aplicações estão à possibilidade de se fazer combinações. Como por exemplo, descobrir quantos perfumes de duas fragrâncias é possível fazer se tivermos três fragrâncias disponíveis.

A tarefa da sua equipe é dado o número de fragrâncias disponíveis e a quantidade de frangâncias utilizadas em um perfume, determinar quantos possíveis perfumes diferentes serão criados.

Entrada

Cada entrada será composta por uma linha contendo o número de fragâncias ($0 < n \leq 10$) e a quantidade de fragrâncias utilizadas em cada um ($0 < s \leq 10$).

A execução deverá ser finalizada caso o número de fragrâncias e a quantiadade de fragrâncias sejam zero (0 0).

Saída

Para cada entrada seu programa deverá responder o número de perfumes diferentes que serão criados.

Exemplos de testes

Entrada	**Saída**
1 1 7 3 10 3 10 9 0 0	1 35 120 10

PROBLEMA I: AS DUAS TORRES

Arquivo: `probi.[c/cpp/pas/java]`

Cor: `Marrom`

Limite de tempo: `2s`

Descrição do problema

Xadrez é um jogo de tabuleiro de natureza recreativa e competitiva para dois jogadores, sendo também conhecido como Xadrez Ocidental ou Xadrez Internacional para distingui-lo dos seus predecessores e de outras variantes da atualidade. A forma atual do jogo surgiu no Sudoeste da Europa na segunda metade do Século XV, durante o Renascimento, depois de ter evoluído de suas antigas origens persas e indianas.

Uma das peças mais populares é a torre. Principalmente para os jogadores iniciantes, a tentação de colocar a torre em jogo muito cedo é grande. A torre se move livremente em linhas e colunas, não podendo, entretanto, passar sobre outras peças. A Figura 5 mostra um exemplo de movimentação da torre.

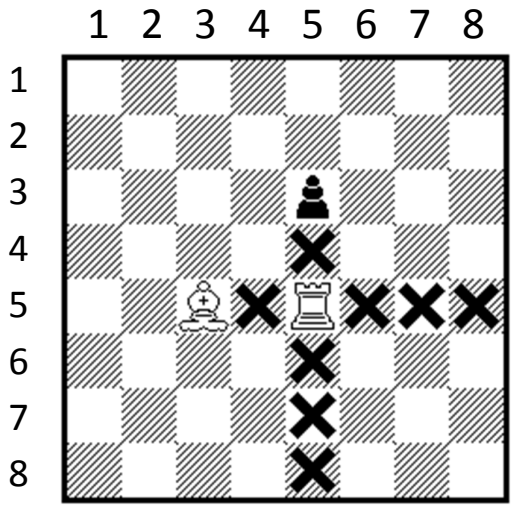

Figura 5 – Exemplo de movimentos possíveis da torre.

Você foi convidado para um duelo de torres. São colocadas duas torres no tabuleiro, uma branca e uma preta, e diversas outras peças que representam apenas obstáculos. O objetivo é cercar e capturar a torre adversária. Não se achando muito bom em xadrez mas acreditando programar muito bem, você decide fazer um programa para tentar vencer este duelo. A primeira tarefa que lhe veio a mente é descobrir qual é o menor número de movimentos necessários para capturar a torre adversária.

Entrada

Cada entrada será composta por uma linha contendo quatro inteiros X, Y, W e Z ($0 < X,Y,Z,W \leq 8$) representando as posições da torre branca e da torre preta. Em seguida, um número de obstáculos N ($0 \leq n \leq 62$). Finalmente, são informadas N linhas com as posições A, B ($0 < A, B \leq 8$) dos obstáculos.

A execução deverá ser finalizada caso o valor X informado seja 0.

Saída

Para cada entrada seu programa deverá responder qual é o número mínimo de movimentos que a torre branca deve realizar para chegar na posição da torre preta, evitando os obstáculos e capturando-a. Caso não seja possível chegar até a torre preta, a saída deve ser zero.

Exemplos de testes

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

PROBLEMA J: CAVERNAS DO RUSAMA

Arquivo: `probj.[c/cpp/pas/java]`

Cor: `Vermelho`

Limite de tempo: `2s`

Descrição do problema

Rusama Lin Baden, o grande chefe da All Caída, organização terrorista responsável pelos atentados ao World Trade Border de 15 de Setembro, foi morto pela tropa de militares Seal em uma operação do exército dos Estados Unidos da Ásia.

A operação durou anos até que o local onde o terrorista estava escondido fosse identificado. Ao contrário do que se esperava, Rusama não estava em uma caverna e sim em uma mansão luxuosa. Dentro da mansão diversos documentos foram encontrados e um deles foi considerado TOP SECRET pela CIA. Após análise do departamento de inteligência da CIA, foi identificado que este documento nada mais era do que uma lista com os complexos de cavernas utilizados pela All Caída. Os analistas ficaram curiosos com o fato de que alguns destes complexos estavam marcados com uma indicação especial. Após visita das unidades militares a dois complexos diferentes (um com a indicação especial) foi possível observar que o material encontrado nos complexos indicados era muito mais valioso (documentos importantes, armamentos pesados, etc).

A CIA trabalhou nestes mapas e conseguiu identificar o padrão responsável pela indicação especial de cada caverna. Ao que parece as cavernas identificadas possuem uma rota que facilitaria o posicionamento de equipes de segurança para proteção do tráfego de materiais entre os locais.

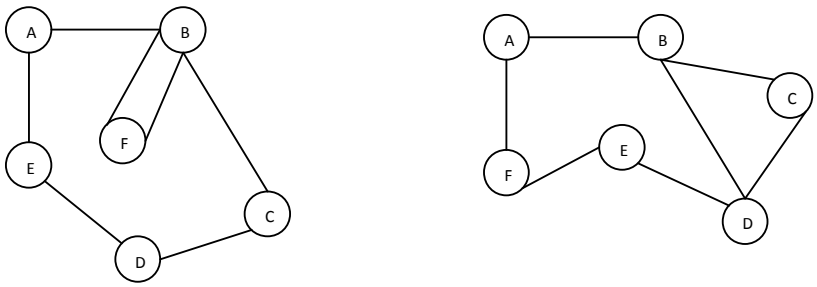

Figura 1 – Duas disposições de cavernas encontradas nos documentos

Na Imagem 1, temos duas disposições de complexos de cavernas distintos, sendo o complexo da esquerda considerado uma caverna que pertence ao padrão identificado pela CIA e o complexo da direita não. O departamento de

inteligência da CIA concluiu que, caso o complexo permitisse que uma pessoa saísse de uma caverna e utilizasse todas as estradas uma única vez, passando por todas as outras cavernas e retornando a cidade de partida, o complexo deveria possuir a marcação especial. Na imagem 2 pode-se notar um destes caminhos partindo da caverna A.

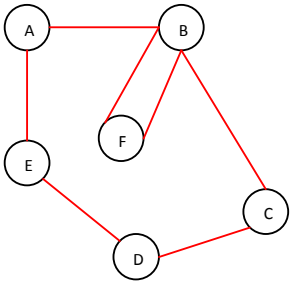

Figura 2 – Rota em um complexo válido

Sua missão é ajudar a CIA nesta tarefa indicando cada um dos complexos importantes.

Entrada

A entrada será composta de um inteiro N (1 < N ≤ 10) indicando o número de cavernas do complexo, um inteiro M (1 < M ≤ 100000) indicando o número de trilhas entre as cavernas. Para cada uma das M trilhas será informado a caverna de origem e a caverna de destino. Todos os caminhos são bidirecionais, porém podem existir mais de uma trilha entre duas cavernas diferentes. A entrada -1 para N encerra os casos de teste.

Saída

A saída será composta por uma linha indicando se o complexo é importante ou não.

Exemplos de testes

Entrada	**Saída**
6 7 A B B C B F C D D E E A F B 6 6 A B B C B D D E E F F A -1	sim nao

PROBLEMA K: MINI GANTT

Arquivo: `probk.[c/cpp/pas/java]`

Cor: `Verde Escuro`

Limite de tempo: `2s`

Descrição do problema

Atualmente, a rapidez com que as mudanças acontecem no ambiente empresarial tem levado um número cada vez maior de empresas a executarem projetos no seu dia-a-dia.

Competitividade acirrada, margens de lucro estreitas, clientes exigentes e avanços tecnológicos constantes, caracterizam um cenário ideal para a execução de projetos, já que tudo tem que acontecer em um prazo cada vez menor e utilizando menos recursos financeiros.

Gerenciar bem os projetos dentro da empresa, sejam eles o lançamento de um produto, a abertura de uma nova filial, a readequação de um processo produtivo ou a implantação de uma nova tecnologia, tornou-se não apenas um diferencial competitivo para as organizações mas, sobretudo, uma questão de sobrevivência.

Dentro desse contexto, uma empresa contratou a sua equipe para desenvolver um programa que contabilize o número de dias necessários para finalizar um projeto e também os números de dias necessários para concluir cada um de seus agrupamentos de tarefas. Por exemplo, para um projeto que tenha três subgrupos de tarefas, seu programa deve calcular quatro números: um número de dias para o projeto e um número para cada um dos três subgrupos.

Entrada

A entrada é composta de vários casos de teste. Cada caso de teste é composto por várias linhas. Cada linha possui três valores separados por espaço, o primeiro representa o nome da tarefa, o segundo o nome da tarefa predecessora e o terceiro o número de horas que serão gastas para finalizar a

tarefa. A primeira linha é diferente das demais, pois ela representa o projeto. As linhas que representam o projeto e os subgrupos de tarefas têm o número de horas iguais a zero. O número de horas das tarefas pertence ao intervalo [0, 100]. A *string* “EOP” (*End Of Project*) no início de uma linha caracteriza o final de um caso de teste e a *string* “EOI” (*End Of Input*) indica o final dos casos de teste.

Saída

Para cada caso de teste deverão ser apresentados o nome do projeto e o número de horas para finalizá-lo e também os nomes dos subgrupos e seus respectivos números de horas, um por linha e em ordem alfabética conforme exemplos abaixo.

Exemplos de testes

Entrada	**Saída**
Projeto001 * 0 Grupo001 Projeto001 0 Grupo002 Grupo001 0 Tarefa001 Grupo002 74 Tarefa002 Projeto001 46 Tarefa003 Tarefa002 4 Tarefa004 Tarefa003 51 Grupo003 Tarefa002 0 Tarefa005 Grupo003 13 Tarefa006 Grupo003 78 Tarefa007 Grupo003 81 EOP Projeto002 * 0 Grupo001 Projeto002 0 Grupo002 Grupo001 0 Tarefa001 Grupo002 61 Tarefa002 Tarefa001 76 Tarefa003 Grupo002 13 Tarefa004 Tarefa003 68 Grupo003 Grupo002 0 Tarefa005 Grupo003 78 Tarefa006 Grupo003 6 EOP EOI	Grupo001 74 Grupo002 74 Grupo003 81 Projeto001 127 EOP Grupo001 137 Grupo002 137 Grupo003 78 Projeto002 137 EOP

PROBLEMA L: PRECISO DE UM RECURSO

Arquivo: `probl.[c/cpp/pas/java]`

Cor: `Azul Claro`

Limite de tempo: `2s`

Descrição do problema

O planejamento do projeto dentro da gestão de projetos é o processo para quantificar o tempo e orçamento de um projeto. Na fase de planejamento o projeto é refinado, as atividades são definidas, são alocados os recursos, são estimados os custos e são determinadas as opções para atingir os objetivos. A finalidade do planejamento do projeto é criar um plano que permita um gestor acompanhar o progresso e a alocação de sua equipe.

Gestores de projeto começam a enfrentar problemas quando seus recursos precisam ser alocados em mais de um projeto. Um dos problemas enfrentados é a utilização de mais de 100% do tempo de trabalho dos recursos.

Dentro desse contexto, sua equipe foi contratada para desenvolver um programa que calcule, para um determinado período, o número máximo de horas utilizado em um dia de trabalho para cada recurso. Por exemplo, para um recurso alocado em duas tarefas que apresentam intersecção entre seus períodos de execução, as horas de ambas as tarefas devem ser somadas para calcular o total de horas do recurso alocado nos dias de trabalho da intersecção.

Entrada

A entrada é composta de vários casos de teste. Cada caso de teste é composto por várias linhas. Cada linha possui cinco valores separados por espaço, o primeiro representa o nome do recurso, o segundo o nome da tarefa, o terceiro a data de início da tarefa, o quarto a data de término da tarefa e o quinto o número de horas por dia de trabalho. O número de horas pertence ao intervalo [1, 8]. A *string* “EOP” (*End Of Project*) no início de uma linha caracteriza o final de um caso de teste e a *string* “EOI” (*End Of Input*) indica o final dos casos de teste.

Saída

Para cada caso de teste deverá ser apresentado como saída o número máximo de horas alocado em um dia de trabalho para cada recurso, um por linha e em ordem alfabética conforme exemplos abaixo.

Exemplos de testes

Entrada	**Saída**
Recurso001 Tarefa001 28/05/2011 08/06/2011 2 Recurso001 Tarefa002 14/05/2011 06/06/2011 4 Recurso001 Tarefa003 13/05/2011 01/06/2011 2 Recurso001 Tarefa004 24/05/2011 10/06/2011 7 Recurso002 Tarefa001 29/05/2011 02/06/2011 6 Recurso002 Tarefa002 23/05/2011 04/06/2011 4 Recurso002 Tarefa003 21/05/2011 02/06/2011 1 Recurso002 Tarefa004 16/05/2011 25/05/2011 8 Recurso002 Tarefa005 26/05/2011 31/05/2011 4 EOP Recurso001 Tarefa001 28/05/2011 09/08/2011 5 Recurso001 Tarefa002 27/05/2011 04/08/2011 7 EOP EOI	Recurso001: 15 Recurso002: 15 EOP Recurso001: 12 EOP