Maratona de Programação FACENS 2026

CADERNO DE PROBLEMAS

Informações gerais

Este caderno contém 11 problemas; as páginas estão numeradas de 1 a 28, não contando esta folha de rosto.

Verifique se o caderno está completo.

1) Sobre os nomes dos programas

Sua solução deve ser chamada arquivo.c, arquivo.cpp, arquivo.py ou arquivo.java, onde "arquivo" é especificado em cada problema. Lembre que em Java o nome da classe principal deve ser igual ao nome do arquivo.

2) Sobre a entrada

A entrada de seu programa deve ser lida da entrada padrão.

A entrada pode ser composta de um ou mais casos de teste, descritos em um número de linhas que depende do problema.

Quando uma linha da entrada contém vários valores, estes são separados por um único espaço em branco; a entrada não contém nenhum outro espaço em branco. Cada linha, incluindo a última, contém exatamente um caractere final-de-linha.

3) Sobre a saída

A saída de seu programa deve ser escrita na saída padrão.

Quando uma linha da saída contém vários valores, estes devem ser separados por um único espaço em branco; a saída não deve conter nenhum outro espaço em branco.

Cada linha, incluindo a última, deve conter exatamente um caractere final-de-linha.

3) Sobre o limite de tempo

O limite de tempo dos exercícios para C, C++ e Python 2/3 é 1 segundo.
O limite de tempo para Java é 5 segundos (sinceramente, Java demora muito para compilar!)

PROBLEMA A: INICIALIZAÇÃO DO REATOR

Arquivo: `reator.[cpp/c/java/py]`

Cor: laranja

Descrição do problema

A estação orbital Nexus utiliza um pequeno reator quântico auxiliar durante o processo de inicialização dos sistemas principais. O reator funciona de maneira bastante simples: a cada ciclo de ativação, sua carga energética aumenta exatamente em uma unidade. Os engenheiros da estação precisam verificar rapidamente qual será o próximo nível de energia do reator após um novo ciclo. Dado o nível atual de energia, determine qual será o valor imediatamente seguinte.

Entrada

A primeira linha da entrada contém um inteiro N, representando a quantidade de casos de teste. Cada caso de teste é composto por uma única linha contendo um inteiro X, representando o nível atual de energia do reator.

Saída

Para cada caso de teste, imprima uma linha contendo o próximo nível de energia do reator.

Restrições

- $1 \leq N \leq 100$
- $1 \leq X \leq 100$

Exemplos de testes

Entrada	**Saída**
5 100 2 35 47 58	101 3 36 48 59

PROBLEMA B: ROTAS DE ENTREGA

Arquivo: `rotas.[cpp/c/java/py]`

Cor: branco polar

Descrição do problema

A TransRota é uma empresa de logística que opera em N cidades do interior do país, numeradas de 1 a N. As cidades são interligadas por estradas com pedágios, e cada estrada pode ser percorrida nos dois sentidos.

O diretor de operações, Zé, precisa planejar rotas especiais: por contrato com alguns municípios, certos carregamentos devem obrigatoriamente passar por uma cidade intermediária de inspeção antes de chegar ao destino, seja para fiscalização sanitária, abastecimento ou descanso dos motoristas.

Dado um conjunto de ordens de entrega, cada uma especifica uma cidade de origem A, uma cidade de destino B e uma cidade de parada obrigatória C. Zé quer saber o menor custo total em pedágios para realizar cada entrega seguindo a rota:

$A \rightarrow C \rightarrow B$

Se não existir caminho possível para alguma ordem, o sistema deve sinalizar que a entrega é impossível.

Entrada

A primeira linha contém um inteiro T, representando a quantidade de casos de teste. Cada caso de teste começa com dois inteiros, N M, onde, N é a quantidade de cidades, M é a quantidade de estradas. As próximas M linhas contêm três inteiros, U V W, indicando uma estrada bidirecional entre as cidades U e V, com custo de pedágio W. Em seguida, uma linha contém um inteiro, Q, representando a quantidade de consultas. As próximas Q linhas contêm três inteiros, A B C, onde, A é a cidade de origem, B é a cidade de destino, C é a cidade de parada obrigatória.

Saída

Para cada consulta, imprima o menor custo do trajeto $A \rightarrow C \rightarrow B$, ou imprima, **IMPOSSIVEL** caso não exista uma rota válida.

Restrições

- $1 \leq T \leq 10$
- $2 \leq N \leq 300$
- $1 \leq M \leq N(N-1)/2$
- $1 \leq W \leq 10^4$
- $1 \leq Q \leq 10^4$
- $1 \leq U,V,A,B,C \leq N$

Não existem estradas duplicadas, A soma dos valores de N em todos os casos de teste não excede 300.

Exemplos de Testes

Entrada	**Saída**
1 5 6 1 2 10 1 3 5 2 3 2 2 4 8 3 4 4 4 5 3 3 1 4 3 1 5 2 2 5 4	9 16 9

PROBLEMA C: DISTRIBUIÇÃO QUÂNTICA

Arquivo: `quantum.[cpp/c/java/py]`

Cor: prata

Descrição do problema

Uma nave cargueira da Federação transporta M unidades de energia quântica destinadas aos módulos centrais da estação orbital. Os módulos possuem diferentes níveis hierárquicos e o protocolo de distribuição segue regras rígidas de aprovação.

O módulo principal propõe uma forma de dividir as unidades de energia entre todos os módulos ativos (incluindo ele mesmo), e os demais módulos votam para aprovar ou rejeitar a proposta. Se 50% ou mais dos módulos votarem a favor, a distribuição é aceita. Caso contrário, o módulo principal é desativado do sistema e o processo continua com os módulos restantes.

Cada módulo sempre tomará a decisão que maximize sua própria quantidade de energia recebida. Em caso de empate, o módulo prefere aprovar a proposta para evitar instabilidade no sistema. Assumindo que todos os módulos são perfeitamente racionais e inteligentes, determine como a energia será distribuída para que a proposta inicial seja aprovada.

Entrada

A primeira linha contém um inteiro T, representando a quantidade de casos de teste. Cada caso de teste contém dois inteiros:

P M

onde:

- P representa a quantidade de módulos ativos;
- M representa a quantidade de unidades de energia disponíveis.

Saída

Para cada caso de teste, imprima uma única linha contendo P inteiros separados por espaço, representando a quantidade de energia recebida por cada módulo.

Restrições

$1 \leq P \leq M \leq 10^5$

Exemplos de testes

Entrada	Saída
3 1 10 2 3 5 100	10 3 0 98 0 1 0 1

PROBLEMA D: SINCRONIZAÇÃO NEURAL

Arquivo: `neural.[cpp/c/java/py]`

Cor: azul bebe

Descrição do problema

A corporação Atlas desenvolveu uma inteligência artificial experimental capaz de controlar milhares de drones simultaneamente.

O núcleo da IA utiliza seis módulos neurais especiais. Cada módulo fornece melhorias permanentes em quatro parâmetros fundamentais do sistema:

- processamento;
- estabilidade;
- largura de banda;
- consumo energético.

Os módulos disponíveis são:

Módulo	Processamento	Estabilidade	Banda	Energia
Beacon	80	80	40	0
Circuito	0	0	100	60
Altium	0	0	60	100
Ethron	0	40	80	80
Foton	50	100	20	0
Deuteron	100	40	20	10

Entretanto, cada módulo está protegido por um guardião virtual. Para ativar um módulo, o núcleo neural precisa derrotar o guardião correspondente. Cada guardião possui quatro atributos. A eficiência ao enfrentar um guardião é calculada pela soma das diferenças entre os atributos atuais do núcleo neural e os atributos do guardião.

Exemplo:

Núcleo (60, 40, 80, 40) x guardião Beacon (40, 30, 10, 20) = 120
Núcleo (60, 40, 80, 40) x guardião Deuteron (120, 30, 70, 60) = -60

O núcleo neural nunca permitirá uma ativação cuja eficiência seja negativa. Após derrotar um guardião, o núcleo recebe permanentemente os atributos do módulo correspondente.

Seu objetivo é determinar a ordem de ativação dos módulos que maximize a soma total das eficiências obtidas. Caso existam múltiplas soluções ótimas, escolha a sequência lexicograficamente menor.

Entrada

A primeira linha contém um inteiro T, representando a quantidade de casos de teste. Cada caso de teste possui 7 linhas contendo 4 inteiros.

- A próxima linha contém os atributos iniciais do núcleo neural.
- As próximas 6 linhas contêm os atributos dos módulos, seguindo exatamente a ordem apresentada na tabela.

Saída

Para cada caso de teste:

- imprima os nomes dos módulos na ordem em que devem ser ativados para maximizar a eficiência total;
- ou imprima:

Precisamos de mais energia para estabilizar o nucleo!

caso não exista uma sequência válida.

Restrições

Todos os valores estarão no intervalo:

[0, 10000]

Exemplos de testes

Entrada	**Saída**
1 80 20 90 10 170 10 0 20 90 200 30 40 10 80 80 200 200 120 40 90 50 150 80 20 60 20 180 10	Beacon Deuteron Ethron Foton Altium Circuito

PROBLEMA E: AUDITORIA DE SENHAS

Arquivo: `senhas.[cpp/c/java/py]`

Cor: verde folha

Descrição do problema

O SecureBank é um banco digital que recentemente sofreu uma tentativa de fraude: criminosos criaram múltiplas contas usando a mesma senha, explorando uma falha no sistema de cadastro para burlar o limite de tentativas de login.

Para conter o problema, a equipe de segurança desenvolveu um módulo de auditoria que recebe as N senhas cadastradas e responde Q investigações. Cada investigação pergunta se a senha do usuário i é idêntica à senha do usuário j, indicando uma possível conta duplicada ou comprometida.

O grande desafio técnico é que as senhas podem possuir centenas de milhares de caracteres. Comparações diretas entre strings tornam-se inviáveis para o volume de consultas exigido pelo sistema. Seu objetivo é responder todas as consultas informadas.

Entrada

A primeira linha contém um inteiro T, representando a quantidade de casos de teste. Cada caso de teste começa com um inteiro, N. Representando a quantidade de senhas cadastradas. As próximas N linhas contêm uma string cada, representando as senhas dos usuários. Em seguida, uma linha contém um inteiro Q, representando a quantidade de consultas. As próximas Q linhas contêm dois inteiros, i j, onde, i representa o índice do primeiro usuário, j representa o índice do segundo usuário. Os índices são numerados a partir de 1.

Saída

Para cada consulta, imprima, **SIM**. Caso as duas senhas sejam idênticas. Caso contrário, imprima **NAO**.

Restrições

- $1 \leq T \leq 10$
- $1 \leq N \leq 10^6$
- 1 ≤ comprimento da senha ≤ 10^5
- $1 \leq Q \leq 10^6$
- $1 \leq i,j \leq N$

As senhas são compostas apenas por letras minúsculas de 'a' até 'z'. A soma dos comprimentos de todas as senhas em um caso de teste não ultrapassa 10^7.

Exemplos de testes

Entrada	Saída
1 4 alpha beta gamma alpha 3 1 4 1 2 2 3	SIM NAO NAO

Explicações

- Consulta 1: senha[1] = "alpha" e senha[4] = "alpha" → SIM
- Consulta 2: senha[1] = "alpha" e senha[2] = "beta" → NAO
- Consulta 3: senha[2] = "beta" e senha[3] = "gamma" → NAO

PROBLEMA F: RUPTURA TEMPORAL

Arquivo: `ruptura.[cpp/c/java/py]`

Cor: vermelho quente

Descrição do problema

Após décadas de pesquisa, a humanidade finalmente conseguiu construir portais temporais estáveis entre cidades. Cada portal consome uma certa quantidade de energia para ser utilizado e gera um nível de instabilidade temporal durante a travessia.

A Federação Cronos deseja enviar um agente da cidade 1 até a cidade N causando a menor instabilidade temporal possível. O agente possui uma bateria com capacidade máxima K unidades de energia. Inicialmente, a bateria começa completamente carregada.

Cada portal conecta duas cidades e possui:

- um custo de energia E;
- uma instabilidade temporal I.

O agente só pode utilizar um portal caso possua energia suficiente. Algumas cidades possuem estações de recarga temporal. Ao chegar em uma dessas cidades, a bateria do agente é instantaneamente recarregada para sua capacidade máxima K. Determine a menor instabilidade temporal total necessária para chegar da cidade 1 até a cidade N.

Entrada

A primeira linha contém um inteiro T, representando a quantidade de casos de teste. Cada caso de teste começa com quatro inteiros:

N M K R

onde:

- N é o número de cidades;
- M é o número de portais;
- K é a capacidade máxima da bateria;
- R é a quantidade de cidades com estação de recarga.

A próxima linha contém R inteiros distintos representando as cidades que possuem estação de recarga. As próximas M linhas contêm quatro inteiros:

U V E I

Indicando um portal bidirecional entre as cidades U e V, que consome E unidades de energia e gera I unidades de instabilidade temporal.

Saída

Para cada caso de teste, imprima:

- a menor instabilidade temporal possível para chegar da cidade 1 até a cidade N;
- ou -1 caso não seja possível realizar a viagem.

Restrições

- 1 ≤ T ≤ 10
- 2 ≤ N ≤ 10^5
- 1 ≤ M ≤ 2×10^5
- 1 ≤ K ≤ 100
- 0 ≤ R ≤ N
- 1 ≤ E ≤ K
- 1 ≤ I ≤ 10^9

A soma de N e M em todos os casos não excede 2×10^5.

Exemplos de testes

Entrada	Saída
1 5 6 5 2 2 4 1 2 3 4 2 3 2 5 3 5 3 7 1 4 5 20 4 5 2 1 2 5 5 50	16

Explicações

Uma das melhores rotas é:

1 → 2 → 3 → 5

- O agente chega na cidade 2 e recarrega completamente.
- A instabilidade total será:

4 + 5 + 7 = 16

PROBLEMA G: REDE ORBITAL

Arquivo: orbital.`[cpp/c/java/py]`

Cor: verde tiffany

Descrição do problema

A Federação Orbital Terrestre mantém uma gigantesca rede de estações espaciais distribuídas ao redor da Terra. Essas estações são utilizadas para transporte de cargas, missões científicas e comunicação global. Cada estação opera em um fuso específico utilizado para sincronizar pousos, decolagens e transmissões. As viagens entre estações são realizadas por aeronaves aeroespaciais que viajam a uma velocidade média de:

1000 quilômetros por hora

Por questões de segurança operacional, nenhuma aeronave pode permanecer em voo contínuo por mais de 12 horas. Caso uma viagem exija mais tempo, a aeronave deverá realizar escalas intermediárias em outras estações orbitais. Cada escala intermediária adiciona exatamente 3 horas ao tempo total da viagem devido aos processos obrigatórios de reabastecimento, manutenção e sincronização orbital. Uma rota entre duas estações pode utilizar múltiplas conexões intermediárias, desde que cada trecho individual respeite o limite máximo de 12 horas de voo contínuo. A Federação deseja verificar se determinadas rotas estão sendo realizadas dentro das normas estabelecidas.

Você deverá determinar se o horário informado para cada viagem é válido com:

- as distâncias entre as estações;
- os fusos orbitais;
- as escalas obrigatórias;
- e o limite máximo de voo contínuo.

Assuma que os tempos de ida e volta entre duas estações são sempre iguais.

Entrada

A entrada contém apenas um caso de teste contendo várias linhas. A primeira linha possui um inteiro A, representando a quantidade de estações orbitais. Cada uma das próximas A linhas contém:

CÓDIGO FUSO

onde:

- CÓDIGO é o identificador da estação orbital;
- FUSO é o deslocamento orbital da estação.

A próxima linha contém um inteiro V, representando a quantidade de rotas aeroespaciais. Cada uma das próximas V linhas contém:

ORIGEM DESTINO DISTANCIA

onde:

- ORIGEM é a estação de saída;
- DESTINO é a estação de chegada;
- DISTANCIA é a distância, em quilômetros, entre as estações.

Na sequência, há uma linha contendo um inteiro R, representando a quantidade de viagens que devem ser verificadas. Cada uma das próximas R linhas contém:

ORIGEM DESTINO SAIDA CHEGADA

onde:

- ORIGEM é a estação de origem;
- DESTINO é a estação de destino;
- SAIDA é o horário local de partida;
- CHEGADA é o horário local de chegada na estação de destino.

OBS: Um horário de chegada maior que 24 indica que a aeronave chegou no dia seguinte.

Saída

A saída deve conter R linhas. Para cada viagem especificada na entrada, imprima, **SIM,** caso seja possível realizar a rota no tempo estabelecido e dentro das normas da Federação. Caso contrário, imprima, **NAO.**

Restrições

$3 \leq A \leq 20$
$-12 \leq FUSO \leq 12$
$3 \leq V \leq 100$
$100 \leq DISTANCIA \leq 20000$
$1 \leq R \leq 50$
$0 \leq SAIDA \leq 23$
$0 \leq CHEGADA \leq 48$

Todos os identificadores são compostos apenas por letras maiúsculas de A a Z.

Exemplos de testes

Entrada	**Saída**
5 ORB -7 LUN -4 SKY -4 NOV -3 ZEN 10 7 ORB SKY 9500 NOV SKY 2600 ZEN LUN 16000 LUN NOV 7600 ORB LUN 4100 SKY ZEN 11300 NOV ZEN 13300 5 NOV ORB 9 20 NOV ORB 9 19 LUN ZEN 1 31 LUN ZEN 1 38 ZEN SKY 10 26	SIM NAO NAO NAO SIM

PROBLEMA H: O CONVERSOR DE CÓDIGO GRAY

Arquivo: `gray.[cpp/c/java/py]`

Cor: preto

Descrição do problema

Em sistemas de automação industrial e robótica, medir a posição exata de eixos de motores é uma tarefa crítica. Para isso, os engenheiros utilizam sensores que convertem a posição física em números inteiros e os transmitem para uma central de controle.

Se utilizássemos o sistema binário convencional para transmitir esses dados, haveria um problema elétrico grave conhecido como "corrida crítica". Por exemplo, quando o motor passa da posição decimal 3 para a posição 4, a representação binária muda de 0011 para 0100. Repare que, nessa transição, dois bits mudam de valor ao mesmo tempo. Como os circuitos físicos reais possuem pequenas imperfeições e atrasos elétricos, esses bits não mudam exatamente no mesmo milissegundo. Isso pode fazer com que o computador central leia valores intermediários completamente errados por uma fração de tempo, confundindo o sistema.Para resolver esse problema, a indústria utiliza o Código Gray.

A principal propriedade do Código Gray é que, entre dois números inteiros consecutivos, exatamente um único bit altera o seu estado na representação binária (de 0 para 1, ou de 1 para 0). Dessa forma, as transições de estados tornam-se imunes a erros de sincronismo elétrico.

Para determinar qual bit deve ser invertido ao avançar na sequência do Código Gray, aplica-se uma regra baseada na paridade do número de bits ativos (iguais a 1) no estado atual. Se o estado atual possuir uma quantidade par de bits 1, a próxima posição da sequência é obtida invertendo-se apenas o último bit (o bit mais à direita). Caso o estado atual possua uma quantidade ímpar de bits 1, deve-se localizar o bit 1 que está mais à direita e inverter o bit que está imediatamente à esquerda dele. Essa alternância sistemática garante que a sequência avance passando por todas as combinações numéricas possíveis sem nunca repetir um estado e alterando estritamente um bit por vez.

A tabela abaixo exemplifica a diferença de comportamento entre o sistema binário convencional e o Código Gray para todas as combinações de 4 bits:

Decimal de Entrada	Binário Convencional	Código Gray (Binário)	Bits alterados no Gray	Decimal de Saída
0	0000	0000	-	0
1	0001	0001	1 bit	1
2	0010	0011	1 bit	3
3	0011	0010	1 bit	2
4	0100	0110	1 bit	6
5	0101	0111	1 bit	7
6	0110	0101	1 bit	5
7	0111	0100	1 bit	4
8	1000	1100	1 bit	12
9	1001	1101	1 bit	13
10	1010	1111	1 bit	15
11	1011	1110	1 bit	14
12	1100	1010	1 bit	10
13	1101	1011	1 bit	11
14	1110	1001	1 bit	9
15	1111	1000	1 bit	8

Sua tarefa é desenvolver o módulo de software do firmware do sensor. O programa receberá uma lista de valores em formato decimal convencional e deverá retornar, para cada um deles, o seu valor equivalente em Código Gray, também representado em formato decimal.

Entrada

A primeira linha da entrada contém um único inteiro N, representando a quantidade de números que serão convertidos.

As N linhas seguintes contêm, cada uma, um número inteiro Vi em formato decimal convencional, representando a informação original do sensor.

Saída

Para cada uma das N linhas de entrada, seu programa deve imprimir uma única linha contendo o valor equivalente daquela posição em Código Gray, representado em formato decimal.

Restrições

- 1 ≤ N ≤ 10^5
- 0 ≤ V < 2^64 (Os valores cabem em um inteiro de 64 bits sem sinal)

Exemplos de testes

Entrada	**Saída**
10 0 1 2 3 4 7 12 100 4294967295 18446744073709551615	0 1 3 2 6 4 10 86 2147483648 9223372036854775808

PROBLEMA I: CHUVA ACUMULADA

Arquivo: `chuva[cpp/c/java/py]`

Cor: lilas

Descrição do problema

O Instituto Nacional de Monitoramento Climático (INMC) opera uma rede de sensores pluviométricos no Vale do Rio Seco, região historicamente vulnerável a secas e enchentes repentinas. Para cada um dos N dias de um período de monitoramento, os sensores registram o volume de chuva em milímetros.

O sistema de alertas do INMC precisa responder rapidamente a três tipos de eventos:

Correção de leitura: sensores defeituosos eventualmente enviam valores incorretos. Quando um erro é detectado, o operador corrige o registro do dia afetado com o valor real medido.

Consulta de acumulado: gestores de barragens consultam o volume total de chuva acumulada em um intervalo de dias para decidir sobre abertura de comportas.

Consulta de pico: a defesa civil consulta o maior volume registrado em um único dia dentro de um intervalo, para avaliar risco de enxurradas repentinas.

Como o número de consultas pode ser muito grande, percorrer todos os dias a cada consulta de acumulado é inviável. O sistema utiliza prefix sum para respondê-la em O(1), atualizando o array de prefixos sempre que uma correção de leitura é recebida. A consulta de pico percorre o intervalo linearmente. Sua tarefa é implementar esse sistema de monitoramento.

Entrada

A primeira linha contém um inteiro T, representando a quantidade de casos de teste. Para cada caso de teste, a primeira linha contém dois inteiros N e Q — o número de dias monitorados e o número de operações. A segunda linha contém N inteiros a_1, a_2, ..., a_n — o volume de chuva inicial de cada dia. As próximas Q linhas contêm cada uma das operações:

- A i v — corrigir o registro do dia i para o valor v
- C L R — consultar o volume acumulado (soma) entre os dias L e R
- M L R — consultar o pico (máximo) diário entre os dias L e R

Saída

Para cada operação do tipo C ou M, imprima o resultado em uma linha. As saídas dos diferentes casos de teste devem ser impressas na mesma ordem em que aparecem na entrada.

Restrições

- $1 \leq T \leq 20$
- $1 \leq N \leq 10^6$
- $1 \leq Q \leq 10^5$
- $0 \leq a_i, v \leq 10^3$
- $1 \leq i \leq N$
- $1 \leq L \leq R \leq N$

Exemplos de testes

Entrada	**Saída**
1 6 6 3 1 4 1 5 9 C 2 5 M 2 5 A 3 10 C 2 5 M 1 6 C 1 6	11 5 17 10 29

Explicação

Array inicial: 3 1 4 1 5 9

Prefix sum: 0 3 4 8 9 14 23

- C 2 5: prefix[5] - prefix[1] = 14 - 3 = 11
- M 2 5: max(1, 4, 1, 5) = 5
- A 3 10: a[3] muda de 4 para 10 (diferença = +6).
    - Array: 3 1 10 1 5 9. Prefix: 0 3 4 14 15 20 29
- C 2 5: prefix[5] - prefix[1] = 20 - 3 = 17
- M 1 6: max(3, 1, 10, 1, 5, 9) = 10
- C 1 6: prefix[6] - prefix[0] = 29 - 0 = 29

PROBLEMA J: SENSORES ABISSAIS

Arquivo: `sensores.[cpp/c/java/py]`

Cor: verde limão

Descrição do problema

As cidades submarinas de Atlântida utilizam sensores acústicos para monitorar atividades sísmicas no fundo do oceano. Com o passar dos anos, os engenheiros perceberam que alguns sensores ocupam posições estratégicas na fronteira externa da rede submarina. Esses sensores serão utilizados para instalar torres principais de monitoramento.

Para simplificar o mapa, os engenheiros representaram os sensores em um plano bidimensional, onde cada ponto indica a posição de um sensor submarino. Seu trabalho é determinar quais sensores pertencem à fronteira externa da rede. Considere que a fronteira externa é formada pelos sensores que compõem o menor contorno convexo capaz de envolver toda a rede submarina.

Entrada

A primeira linha da entrada contém um inteiro T, indicando o número de casos de teste. Cada caso de teste é composto por duas linhas. A primeira linha contém um inteiro N ($3 \leq N \leq 100$), indicando a quantidade de sensores. A segunda linha contém N pares de coordenadas inteiras no formato X,Y separados por espaço, representando as posições dos sensores.

Saída

Para cada caso de teste, seu programa deve exibir em uma linha as coordenadas dos sensores que pertencem à fronteira externa da rede submarina. As coordenadas devem ser separadas por espaço e ordenadas por X e então por Y.

Exemplos de testes

Entrada	**Saída**
1 7 1,10 2,3 10,10 5,5 10,1 1,1 7,2	1,1 1,10 10,1 10,10

PROBLEMA K: O NÚCLEO DE ATLAS

Arquivo: `atlas.[cpp/c/java/py]`

Cor: azul cobalto

Descrição do problema

Após séculos de expansão espacial, a humanidade passou a depender do Núcleo de Atlas, uma gigantesca fonte de energia capaz de alimentar colônias espalhadas pela galáxia.

Cada colônia possui um nível mínimo de estabilidade energética para continuar funcionando. Entretanto, ao fornecer energia para uma colônia, o núcleo sofre oscilações que podem aumentar ou diminuir sua potência atual.

A Federação precisa descobrir qual é a menor potência inicial necessária para que todas as colônias possam ser estabilizadas em alguma ordem. Você recebeu uma lista contendo duas informações para cada colônia:

- Mi: a potência mínima necessária para estabilizar a colônia;
- Di: a variação de potência causada após estabilizá-la.

Uma colônia só pode ser estabilizada caso a potência atual do núcleo seja maior ou igual a Mi. Após estabilizar a colônia, a potência do núcleo passa a ser:

$$P = P + D_i$$

As colônias podem ser estabilizadas em qualquer ordem. Determine a menor potência inicial necessária para estabilizar todas as colônias.

Entrada

A primeira linha contém um inteiro T, representando o número de casos de teste. Cada caso de teste começa com um inteiro N, representando a quantidade de colônias. As próximas N linhas contêm dois inteiros:

Mi Di

Onde:

- Mi é a potência mínima necessária;
- Di é a variação causada após estabilização.

Saída

Para cada caso de teste, imprima a menor potência inicial necessária para estabilizar todas as colônias.

Restrições

- 1 ≤ T ≤ 10
- 1 ≤ N ≤ 2×10^5
- 0 ≤ Mi ≤ 10^9
- -10^9 ≤ Di ≤ 10^9

A soma de N em todos os casos não excede 2 x 10^5

Exemplos de testes

Entrada	**Saída**
2 3 10 5 20 -10 15 3 4 5 -4 7 -2 8 10 20 -5	12 10