Competidor(a): _____

Número de inscrição: _____–_____ *(opcional)*

Olimpíada Brasileira de Informática

OBI2023

Caderno de Tarefas

Modalidade Programação • Nível 2 • Fase 3

30 de setembro de 2023

A PROVA TEM DURAÇÃO DE 5 HORAS

Promoção:

Sociedade Brasileira de Computação

Apoio:

Coordenação:

v1.0

Instruções

LEIA ATENTAMENTE ESTAS INSTRUÇÕES ANTES DE INICIAR A PROVA

- Este caderno de tarefas é composto por 14 páginas (não contando a folha de rosto), numeradas de 1 a 14. Verifique se o caderno está completo.
- A prova deve ser feita individualmente.
- É proibido consultar a Internet, livros, anotações ou qualquer outro material durante a prova. É permitida a consulta ao *help* do ambiente de programação se este estiver disponível.
- As tarefas têm o mesmo valor na correção.
- A correção é automatizada, portanto siga atentamente as exigências da tarefa quanto ao formato da entrada e saída de seu programa; em particular, seu programa não deve escrever frases como "Digite o dado de entrada:" ou similares.
- Não implemente nenhum recurso gráfico nas suas soluções (janelas, menus, etc.), nem utilize qualquer rotina para limpar a tela ou posicionar o cursor.
- As tarefas **não** estão necessariamente ordenadas, neste caderno, por ordem de dificuldade; procure resolver primeiro as questões mais fáceis.
- Preste muita atenção no nome dos arquivos fonte indicados nas tarefas. Soluções na linguagem C devem ser arquivos com sufixo *.c*; soluções na linguagem C++ devem ser arquivos com sufixo *.cc* ou *.cpp*; soluções na linguagem Pascal devem ser arquivos com sufixo *.pas*; soluções na linguagem Java devem ser arquivos com sufixo *.java* e a classe principal deve ter o mesmo nome do arquivo fonte; soluções na linguagem Python 3 devem ser arquivos com sufixo *.py*; e soluções na linguagem Javascript devem ter arquivos com sufixo *.js*.
- Na linguagem Java, **não** use o comando *package*, e note que o nome de sua classe principal deve usar somente letras minúsculas (o mesmo nome do arquivo indicado nas tarefas).
- Para tarefas diferentes você pode escolher trabalhar com linguagens diferentes, mas apenas uma solução, em uma única linguagem, deve ser submetida para cada tarefa.
- Ao final da prova, para cada solução que você queira submeter para correção, copie o arquivo fonte para o seu diretório de trabalho ou pen-drive, conforme especificado pelo seu professor.
- Não utilize arquivos para entrada ou saída. Todos os dados devem ser lidos da entrada padrão (normalmente é o teclado) e escritos na saída padrão (normalmente é a tela). Utilize as funções padrão para entrada e saída de dados:
  - em Pascal: *readln, read, writeln, write*;
  - em C: *scanf, getchar, printf, putchar*;
  - em C++: as mesmas de C ou os objetos *cout* e *cin*.
  - em Java: qualquer classe ou função padrão, como por exemplo *Scanner, BufferedReader, BufferedWriter* e *System.out.println*
  - em Python: *read,readline,readlines,input,print,write*
  - em Javascript: *scanf, printf*
- Procure resolver a tarefa de maneira eficiente. Na correção, eficiência também será levada em conta. As soluções serão testadas com outras entradas além das apresentadas como exemplo nas tarefas.

Pirâmide

Autor: Mateus Bezrutchka

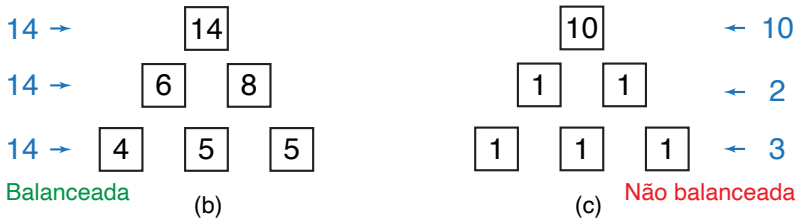
Nome do arquivo: `piramide.c`, `piramide.cpp`, `piramide.java`, `piramide.js` *ou* `piramide.py`

O mais novo lançamento da OBI (*Organização de Brincadeiras Infantis*) é o jogo *Balanceamento de Pirâmide*, que visa explorar as habilidades matemáticas das crianças. O jogo contém seis cubos, todos de mesmo tamanho mas com pesos possivelmente diferentes (os cubos são preenchidos com diferentes volumes de areia). O peso de cada cubo em gramas está escrito em uma de suas faces.

(a)

Em *Balanceamento de Pirâmide*, o objetivo é empilhar os seis cubos de modo a formar uma pirâmide de três andares (com três cubos no primeiro andar, dois no segundo e um no terceiro) tal que os pesos totais de cada andar (ou seja, a soma dos pesos dos cubos no andar) possuam o mesmo valor; nesse caso, dizemos que a pirâmide é *balanceada*. Por exemplo, se os cubos tem pesos (em gramas) 6, 5, 4, 14, 5 e 8, podemos obter uma pirâmide balanceada (com peso total 14 em cada andar) colocando os dois cubos de peso 5 e o cubo de peso 4 no primeiro andar, os cubos de pesos 6 e 8 no segundo andar, e o cubo de peso 14 no terceiro andar (veja a figura b).

Em toda unidade de *Balanceamento de Pirâmide*, deveria ser possível formar uma pirâmide balanceada com os seis cubos dados. Porém, devido a problemas na linha de produção, isso nem sempre é verdade. Recentemente, foi encontrado um exemplar contendo um cubo de peso 10 e cinco cubos de peso 1! Observe que não podemos obter uma pirâmide balanceada com esses cubos (só o cubo de peso 10 é mais pesado do que todos os outros juntos).



Você foi contratado como parte do controle de qualidade da OBI, e sua primeira missão é avaliar se uma unidade do jogo foi produzida corretamente: dados os pesos dos seis cubos, determine se é possível formar uma pirâmide balanceada com os cubos.

Entrada

A única linha da entrada contém seis números inteiros positivos X_i ($1 \leq i \leq 6$), indicando os pesos em gramas de cada um dos seis cubos, separados por espaços.

Saída

Seu programa deverá imprimir uma única linha contendo um único caractere: 'S' (sem aspas), se é possível formar uma pirâmide balanceada com os seis cubos, ou 'N' (sem aspas), se é impossível.

Restrições

- $1 \leq X_i \leq 1\,000$ para $1 \leq i \leq 6$

Informações sobre a pontuação

A tarefa vale 100 pontos.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
`6 5 4 14 5 8`	`S`

Explicação do exemplo 1: Esse exemplo é representado pela figura (b).

Exemplo de entrada 2	**Exemplo de saída 2**
`10 1 1 1 1 1`	`N`

Explicação do exemplo 2: Esse exemplo é representado pela figura (c).

Exemplo de entrada 3	**Exemplo de saída 3**
`10 30 50 70 80 120`	`S`

Explicação do exemplo 3: Podemos colocar os cubos de pesos 10, 30 e 80 no primeiro andar, os cubos de pesos 50 e 70 no segundo andar, e o cubo de peso 120 no terceiro andar, obtendo peso total de 120 por andar.

Exemplo de entrada 4	**Exemplo de saída 4**
`4 8 12 24 20 16`	`N`

Transportes

Autores: André Amaral de Sousa e Rafael Nascimento Soares

Nome do arquivo: `transportes.c`, `transportes.cpp`, `transportes.java`, `transportes.js` *ou* `transportes.py`

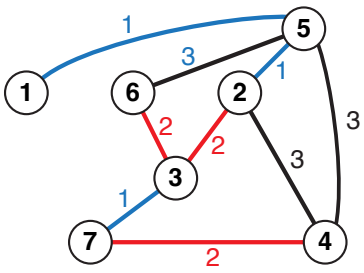
Ênia, a atual rainha da Nlogônia, é fascinada por meios de transporte de todos os tipos. Como consequência, a Nlogônia possui a melhor rede de transporte público do mundo inteiro.

Existem N estações espalhadas por todo o reino, K *sistemas de transporte* (metrô, ônibus, avião, barco, etc) e M ligações diretas entre pares de estações. Cada ligação pertence a um único sistema de transporte entre os K existentes, e pode ser percorrida nas duas direções. Para cada par de estações, existe no máximo uma ligação no total (ou seja, entre todos os sistemas).

Para usar os sistemas de transporte, o usuário precisa comprar passagens. Ênia estabeleceu as seguintes regras para estimular o transporte público:

- Cada um dos K sistemas de transporte tem um preço fixo de passagem. Assim, o preço de uma passagem do sistema i é P_i, independente da estação.
- A passagem só deve ser paga quando o usuário entrar no sistema de transporte.
- O usuário só pode entrar em um sistema por vez (por exemplo, para entrar no sistema de avião ele precisa sair do sistema de metrô).
- Após pagar o preço P_i para entrar no sistema i, o usuário pode se locomover o quanto desejar usando ligações do sistema i, contanto que ele não saia desse sistema. Caso o usuário troque de sistema, ele deverá pagar por uma nova passagem de preço P_i para retornar ao sistema i.

Por exemplo, considere a seguinte rede de transportes:



Perceba que há 7 estações e 9 ligações, das quais 3 são do sistema 1, 3 são do sistema 2 e 3 são do sistema 3. Suponha que os preços das passagens sejam $P_1 = 5$, $P_2 = 1$ e $P_3 = 7$ e que o usuário deseja fazer uma viagem da estação 1 para a estação 7. Temos várias opções de trajeto, como por exemplo (as setas indicam a ordem em que visitamos as estações):

- **Viagem 1:** $1 \to 5 \to 6 \to 3 \to 7$
  - A ligação entre as estações 1 e 5 é do tipo 1, logo precisamos pagar $P_1 = 5$ para entrar no sistema 1 pela primeira vez.
  - $5 \to 6$ usa o sistema 3, então precisamos pagar $P_3 = 7$ para entrar no sistema 3 (saindo do sistema 1).
  - $6 \to 3$ usa o sistema 2, então pagamos $P_2 = 1$.

  - $3 \to 7$ usa o sistema 1, então pagamos $P_1 = 5$. Note que precisamos pagar novamente pela passagem do sistema 1 pois estamos reentrando nele.
  - No total, a viagem custa $5 + 7 + 1 + 5 = 18$.

- **Viagem 2:** $1 \to 5 \to 2 \to 4 \to 7$
  - $1 \to 5$ usa o sistema 1, então pagamos $P_1 = 5$.
  - $5 \to 2$ também usa o sistema 1. Como já entramos nesse sistema, não precisamos pagar novamente.
  - $2 \to 4$ usa o sistema 3, então trocamos de sistema e pagamos $P_3 = 7$.
  - $4 \to 7$ usa o sistema 2, então pagamos mais $P_2 = 1$.
  - No total, essa viagem custará $5 + 7 + 1 = 13$.

- **Viagem 3:** $1 \to 5 \to 2 \to 3 \to 7$
  - $1 \to 5$ usa o sistema 1, então pagamos $P_1 = 5$.
  - $5 \to 2$ também usa o sistema 1, então não pagamos nada.
  - $2 \to 3$ usa o sistema 2, então pagamos $P_2 = 1$ para trocar de sistema.
  - $3 \to 7$ usa o sistema 1, então pagamos $P_1 = 5$ para voltar ao sistema 1.
  - No total, essa viagem custará $5 + 1 + 5 = 11$.

Existem outras viagens que começam na estação 1 e terminam na estação 7. Porém, o menor custo total entre todas essas viagens é 11 (correspondente à viagem 3).

Ênia deseja saber o menor custo total para ir da estação A para a estação B, e para isso te contratou. Responda corretamente, ou ela te expulsará da Nlogônia!

Entrada

A primeira linha da entrada contém três inteiros N, M e K, representando o número de estações, o número de ligações entre as estações, e a quantidade de sistemas de transporte, respectivamente.

A segunda linha contém K números inteiros. O i-ésimo desses números é P_i, representando o preço da passagem do sistema de transporte i.

As próximas M linhas contém a descrição das ligações. A j-ésima dessas linhas tem três números V_j, U_j e T_j, representando que há uma ligação entre as estações V_j e U_j usando o sistema T_j.

A última linha da entrada contém dois números inteiros, A e B, representando a estação inicial e a estação final da viagem requisitada pela rainha.

Saída

Seu programa deverá imprimir uma única linha contendo um único inteiro, o menor custo total de uma viagem que começa na estação A e termina na estação B.

Se não for possível fazer uma viagem de A a B usando a rede de transportes, imprima -1.

Restrições

- $2 \leq N \leq 100\,000$
- $1 \leq M \leq 100\,000$
- $1 \leq K \leq 100\,000$

- $1 \leq P_i \leq 1\,000$ para todo $1 \leq i \leq K$
- $1 \leq V_j, U_j \leq N$ para todo $1 \leq j \leq M$
- $V_j \neq U_j$ para todo $1 \leq j \leq M$
- $1 \leq T_j \leq K$ para todo $1 \leq j \leq M$
- Cada par de estações é ligado por no máximo uma ligação no total (ou seja, os pares de estações V_j, U_j são todos distintos)
- $1 \leq A, B \leq N$
- $A \neq B$

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima:

- **Subtarefa 1 (11 pontos):** $K = 1$. *(Veja o exemplo 2.)*
- **Subtarefa 2 (16 pontos):** $K = 2$, $N \leq 100$ e $P_1 = P_2 = 1$. *(Veja o exemplo 3.)*
- **Subtarefa 3 (34 pontos):** $K \leq 10$, $N \leq 10\,000$.
- **Subtarefa 4 (18 pontos):** $K \leq 100$.
- **Subtarefa 5 (21 pontos):** Nenhuma restrição adicional.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima *(elas não precisam ser resolvidas em ordem)*. Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
7 9 3 5 1 7 1 5 1 5 2 1 7 3 1 3 6 2 2 3 2 7 4 2 2 4 3 5 6 3 5 4 3 1 7	11

Explicação do exemplo 1: Este exemplo corresponde ao exemplo mostrado no enunciado.

Exemplo de entrada 2	**Exemplo de saída 2**
6 4 1 50 1 4 1 6 1 1 3 2 1 4 6 1 3 5	-1

Exemplo de entrada 3	**Exemplo de saída 3**
5 5 2 1 1 1 2 1 5 2 1 1 3 2 4 3 1 4 5 2 2 3	2

Oficina Mecânica

Autor: André Amaral de Sousa

Nome do arquivo: `mecanica.c`, `mecanica.cpp`, `mecanica.java`, `mecanica.js` *ou* `mecanica.py`

O dono de uma oficina mecânica te contratou para criar um software de gerenciamento. Com esse software, ele espera não só conseguir controlar os estoques dos produtos utilizados na oficina, mas também auxiliar no trabalho de seus funcionários e tornar a oficina mais eficiente.

Como um(a) excelente programador(a), você já implementou quase todo o software. Falta apenas uma funcionalidade: distribuir os carros a serem consertados entre os mecânicos de modo a garantir alta satisfação dos clientes. Após analisar os dados coletados na oficina sobre clientes passados, você descobriu que o fator mais importante para a satisfação de um cliente é seu *tempo de espera* para ser atendido, ou seja, o tempo em que ele fica esperando até o seu carro começar a ser consertado. Os clientes não se importam com o tempo de conserto de seus respectivos carros, pois eles sabem que os mecânicos fazem um excelente trabalho.

O tempo de conserto varia de acordo com o carro e com o mecânico: cada carro possui um *tempo base* T_i, já determinado durante o orçamento, que mede a quantidade de trabalho necessária para consertar o carro, e cada mecânico possui um *fator de trabalho* F_j, indicando quanto tempo ele demora para completar uma unidade de trabalho. Assim, o tempo necessário para o mecânico j consertar o carro i é $T_i \cdot F_j$. Vale ressaltar que cada mecânico só trabalha em um único carro por vez, e portanto precisa finalizar o conserto de um carro antes de começar a consertar o próximo.

Por exemplo, suponha que tenhamos quatro carros cujos tempos base são 5, 10, 15 e 20, e dois mecânicos cujos fatores de trabalho são 1 e 2. Uma distribuição possível é:

- O mecânico com fator 1 conserta os carros com tempos 20 e 5, nessa ordem.
- O mecânico com fator 2 conserta os carros com tempos 10 e 15, nessa ordem.

Essa distribuição possui os seguintes tempos de espera:

- O carro com tempo base 20 é atendido imediatamente, portanto seu tempo de espera é 0.
- O carro com tempo base 5 precisa esperar o carro com tempo base 20 ser consertado, portanto seu tempo de espera é $20 \cdot 1 = 20$.
- O carro com tempo base 10 é atendido imediatamente, portanto seu tempo de espera é 0.
- O carro com tempo base 15 precisa esperar o carro com tempo base 10 ser consertado, portanto seu tempo de espera é $10 \cdot 2 = 20$.

A soma total dos tempos de espera na distribuição acima é $0 + 20 + 0 + 20 = 40$. Porém, existe uma distribuição (que é ótima) onde a soma total dos tempos de espera é 20 – você consegue encontrá-la?

Dados o número N de carros esperando para serem atendidos, a quantidade M de mecânicos, o tempo base T_i de cada carro, e o fator de trabalho F_j de cada mecânico, determine a menor soma total dos tempos de espera dos carros, que corresponde à melhor forma de distribuir todos os carros entre os mecânicos.

Entrada

A primeira linha da entrada contém dois inteiros N e M representando o número de carros e o número de mecânicos, respectivamente.

A segunda linha contém N números inteiros. O i-ésimo desses números será T_i, representando o tempo base do i-ésimo carro.

A terceira e última linha da entrada contém M números inteiros. O j-ésimo desses números será F_j, representando o fator de trabalho do j-ésimo mecânico.

Saída

Seu programa deverá imprimir uma única linha contendo um único inteiro, a menor soma total dos tempos de espera em uma distribuição de todos os carros entre os mecânicos.

Restrições

- $1 \leq N \leq 100\ 000$
- $1 \leq M \leq 100\ 000$
- $1 \leq T_i \leq 10\ 000$ para todo $1 \leq i \leq N$
- $1 \leq F_j \leq 10\ 000$ para todo $1 \leq j \leq M$

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima:

- **Subtarefa 1 (14 pontos):** $N \leq 1000$ e $M = 1$.
- **Subtarefa 2 (10 pontos):** $T_i = 1$ para todo $1 \leq i \leq N$ e $F_j = 1$ para todo $1 \leq j \leq M$.
- **Subtarefa 3 (16 pontos):** $F_j = 1$ para todo $1 \leq j \leq M$.
- **Subtarefa 4 (36 pontos):** $N \leq 1000$ e $M \leq 1000$.
- **Subtarefa 5 (24 pontos):** Nenhuma restrição adicional.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima *(elas não precisam ser resolvidas em ordem)*. Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**

`4 2`
`5 10 15 20`
`1 2`
	`20`

Explicação do exemplo 1: Este exemplo corresponde ao exemplo mostrado no enunciado.

Exemplo de entrada 2	**Exemplo de saída 2**
4 1 1 1 1 1 3	18

Explicação do exemplo 2: Como há apenas um mecânico, ele consertará todos os carros. E como todos os tempos base são iguais, a ordem dos carros não afeta o resultado. Logo, podemos só consertar os carros em ordem. Os tempos de espera dos quatro carros são então, respectivamente, 0, 3, 6, e 9, para um total de $0 + 3 + 6 + 9 = 18$.

Exemplo de entrada 3	**Exemplo de saída 3**
4 1 15 5 10 20 3	150

Trio de Bonecas

Autor: André Amaral de Sousa

Nome do arquivo: `bonecas.c`, `bonecas.cpp`, `bonecas.java`, `bonecas.js` *ou* `bonecas.py`

Alice é a gerente de produção de uma fábrica de bonecas chinesas. Nessa fábrica, todas as bonecas possuem o mesmo formato, variando apenas de tamanho. As bonecas são vendidas sempre em trios.

Para um trio formado por bonecas de tamanhos A, B e C, com $A \leq B \leq C$, o *balanceamento* do trio é definido como $(A - B)^2$. De acordo com o controle de qualidade da fábrica, idealmente cada trio deveria ter balanceamento 0, ou seja, conter duas bonecas de mesmo tamanho e uma terceira boneca de tamanho maior ou igual a essas duas; porém, isso nem sempre é possível devido a limitações de estoque.

Alice acabou de receber um pedido urgente: o consulado Chinês fará um grande evento no qual pretende distribuir K trios de bonecas, e precisa que Alice envie imediatamente todos os trios. Existem na fábrica N bonecas de diversos tamanhos, ainda não separadas em trios, e Alice não sabe se conseguirá formar K trios ideais.

Devido à urgência do evento, Alice e o consulado fizeram um acordo: o consulado aceitará trios que não sejam ideais; porém, para garantir um bom evento, Alice deve enviar uma caixa com K trios de bonecas cuja soma dos balanceamentos seja a menor possível.

Por exemplo, suponha que Alice possua 7 bonecas com tamanhos 11, 1, 7, 5, 16, 8 e 15, e que o consulado tenha solicitado 2 trios de bonecas. Algumas configurações possíveis para envio são:

- **Configuração 1:** trios (7, 8, 15) e (1, 5, 16).
  - O primeiro trio possui balanceamento $(8 - 7)^2 = 1$.
  - O segundo trio possui balanceamento $(5 - 1)^2 = 16$.
  - O balanceamento total dessa configuração é $1 + 16 = 17$.
- **Configuração 2:** trios (5, 7, 16) e (8, 11, 15).
  - O primeiro trio possui balanceamento $(7 - 5)^2 = 4$.
  - O segundo trio possui balanceamento $(11 - 8)^2 = 9$.
  - O balanceamento total dessa configuração é $4 + 9 = 13$.

Existem outras configurações possíveis, mas a menor soma de balanceamentos possível é 13 (correspondente à configuração 2).

Ajude Alice a calcular a soma de balanceamentos minima em um conjunto com K trios de bonecas.

Entrada

A primeira linha de entrada contém dois inteiros N e K, representando, respectivamente, o número de bonecas e a quantidade de trios solicitados pelo consulado Chinês.

A segunda linha contém N números inteiros. O i-ésimo desses números será T_i, representando o tamanho da i-ésima boneca.

Saída

Seu programa deverá imprimir uma única linha contendo um único inteiro, a menor soma de balanceamentos possível em uma caixa com K trios de bonecas.

Restrições

- $3 \leq N \leq 10\ 000$
- $1 \leq K \leq 3\ 000$
- $3K \leq N$
- $1 \leq T_i \leq 100\ 000$ para todo $1 \leq i \leq N$

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima:

- **Subtarefa 1 (13 pontos):** $1 \leq T_i \leq 2$ para todo $1 \leq i \leq N$.
- **Subtarefa 2 (28 pontos):** $N \leq 30$.
- **Subtarefa 3 (33 pontos):** $N \leq 1\ 000$ e $K \leq 100$.
- **Subtarefa 4 (26 pontos):** Nenhuma restrição adicional.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima *(elas não precisam ser resolvidas em ordem)*. Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
7 2 11 1 7 5 16 8 15	13

Explicação do exemplo 1: Este exemplo corresponde ao exemplo mostrado no enunciado.

Exemplo de entrada 2	**Exemplo de saída 2**
8 2 2 1 2 2 2 1 2 2	0

Fast-Food

Autor: Mateus Bezrutchka

Nome do arquivo: `fastfood.c`, `fastfood.cpp`, `fastfood.java`, `fastfood.js` *ou* `fastfood.py`

Após a descoberta de um tesouro escondido terminar uma maldição milenar, a histórica ilha da Quadradônia está pronta para ser repovoada. A expectativa mundial é de que a ilha se torne um grande ponto turístico, recebendo visitantes de todo o planeta. Como você pode imaginar, a Quadradônia tem esse nome porque as suas ruas formam um grid quadriculado e todos os prédios comerciais estão localizados em intersecções entre duas ruas.

Você foi contratado como programador na *Fast-Food Turbo (FFT)*, uma rede de franquias de restaurantes de comida rápida. De olho nas oportunidades abertas na Quadradônia, a FFT adquiriu N prédios comerciais na ilha, e deseja abrir, nesses prédios, unidades de duas das suas redes mais populares: o restaurante de hambúrgueres quadrados Quaburguer e o restaurante de pizzas quadradas Pizzatro.

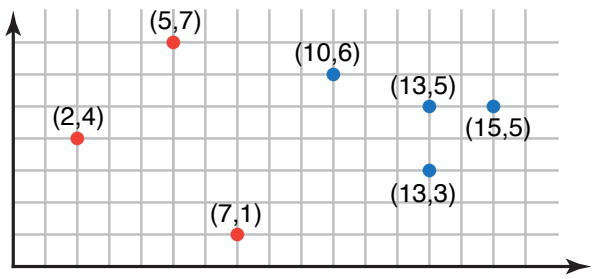
Devido à localização remota da ilha, é muito difícil levar mercadorias para a Quadradônia. Por isso, quando faltam ingredientes em um restaurante da ilha, a solução mais rápida sempre é pedir esses ingredientes a algum outro restaurante da ilha. Se outro restaurante tiver os ingredientes necessários, ele manda um funcionário de carro para fazer a entrega ao restaurante necessitado. Porém, como o Quaburguer e a Pizzatro usam ingredientes bem diferentes, um restaurante só pode fornecer ingredientes para outro restaurante da mesma rede.

Como não existem engarrafamentos na Quadradônia, o tempo que um entregador leva para ir de um prédio a outro é sempre proporcional à *distância quadradônica* entre os prédios, definida como a distância que precisa ser percorrida em ruas da ilha para ir de um prédio a outro. Mais formalmente, se os prédios i e j têm coordenadas (X_i, Y_i) e (X_j, Y_j), respectivamente, a distância quadradônica entre eles é definida como

$$d_{\text{quad}}(i, j) = |X_i - X_j| + |Y_i - Y_j|.$$

A FFT precisa decidir, para cada prédio, de qual rede o restaurante desse prédio fará parte. Como a empresa não consegue prever quais prédios precisarão de ingredientes, os membros do conselho decidiram que eles querem minimizar o pior caso possível, ou seja, a distância d_{quad} máxima entre todos os pares de restaurantes de uma mesma rede.

A figura abaixo ilustra um exemplo para $N = 7$. Observe que, se os prédios 1 e 4 pertencerem à mesma rede, a resposta será no mínimo a distância entre eles, $d_{\text{quad}}(1, 4) = |2 - 13| + |4 - 5| = 12$. Porém, podemos conseguir uma distância máxima igual a 8 se tivermos uma rede com os prédios $(1, 3, 5)$ e a outra com os prédios $(2, 4, 6, 8)$:



Existem outras configurações possíveis, mas 8 é a distância mínima para esse conjunto de prédios.

Sua tarefa é: dadas as coordenadas dos N prédios que a FFT adquiriu, distribua os N prédios entre as duas redes de modo a minimizar a distância quadradônica máxima entre dois restaurantes de uma mesma rede.

Entrada

A primeira linha da entrada contém um único inteiro N, o número de prédios da Quadradônia adquiridos pela FFT.

A segunda linha da entrada contém N inteiros. O i-ésimo desses inteiros, X_i, representa a primeira coordenada do i-ésimo prédio.

A terceira e última linha da entrada contém N inteiros. O i-ésimo desses inteiros, Y_i, representa a segunda coordenada do i-ésimo prédio.

Saída

Seu programa deverá produzir uma única linha contendo um único inteiro, a menor distância quadradônica máxima possível entre dois restaurantes de uma mesma rede.

Restrições

- $1 \leq N \leq 300\,000$
- $1 \leq X_i \leq 500\,000\,000$ para todo $1 \leq i \leq N$
- $1 \leq Y_i \leq 500\,000\,000$ para todo $1 \leq i \leq N$
- Observe que é **possível** que dois prédios tenham as mesmas coordenadas.

Informações sobre a pontuação

A tarefa vale 100 pontos. Estes pontos estão distribuídos em subtarefas, cada uma com suas **restrições adicionais** às definidas acima:

- **Subtarefa 1 (23 pontos):** $N \leq 16$.
- **Subtarefa 2 (35 pontos):** $N \leq 2\,000$.
- **Subtarefa 3 (42 pontos):** Nenhuma restrição adicional.

Seu programa pode resolver corretamente todas ou algumas das subtarefas acima *(elas não precisam ser resolvidas em ordem)*. Sua pontuação final na tarefa é a soma dos pontos de todas as subtarefas resolvidas corretamente por qualquer uma das suas submissões.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
7 2 10 7 13 5 13 15 4 6 1 5 7 3 5	8

Explicação do exemplo 1: Esse exemplo corresponde ao exemplo do enunciado.