Competidor(a): _____

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

Olimpíada Brasileira de Informática

OBI2024

Caderno de Tarefas

Modalidade Programação • Nível 2 • Fase 3

28 de setembro de 2024

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 20 páginas (não contando a folha de rosto), numeradas de 1 a 20. 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.

Brigadeiros

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

Você está na festa de formatura da escola e está quase na hora de servirem um dos doces favoritos dos brasileiros: brigadeiro.

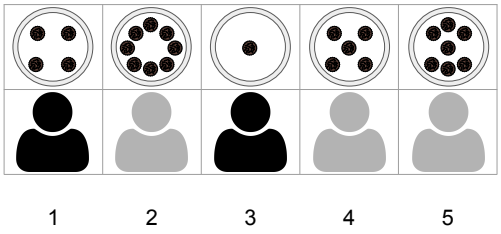
N alunos estão participando da festa e cada um vai comer um dos N pratos com brigadeiros que estão alinhados em cima da mesa do refeitório. Os pratos são numerados de 1 a N da esquerda para a direita. Os alunos estão sentados em frente à mesa na mesma ordem dos brigadeiros e cada um irá comer o prato de brigadeiros à sua frente. Cada prato possui entre 0 e 9 brigadeiros.

O seu grupo de amigos é formado por um subconjunto dos alunos e possui K membros (incluindo você). Seu grupo deseja comer muitos brigadeiros e desenvolveu um plano para que os membros do grupo mudem seus lugares de modo a maximizar a quantidade de brigadeiros que o grupo irá comer no total (ou seja, somando as quantidades que cada membro do grupo irá comer).

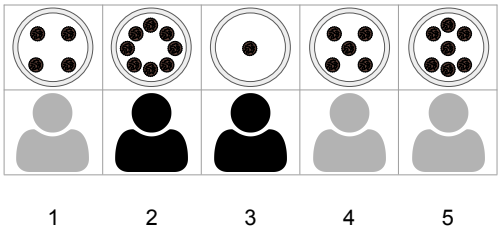
Cada membro do grupo pode pedir para trocar de lugar com um dos alunos sentados nas cadeiras vizinhas à dele (ou seja, os alunos sentados imediatamente à esquerda e à direita dele, se existirem). Os outros alunos não estão prestando atenção nos pratos e, por isso, sempre aceitam as trocas pedidas. Assim, um membro do grupo pode se mover para a esquerda ou para a direita usando várias trocas, sempre com um de seus vizinhos atuais.

Porém, para evitar que os outros alunos descubram o plano, vocês decidiram que somente uma troca pode ser realizada a cada segundo. Vocês sabem que faltam T segundos para que os brigadeiros sejam servidos, e portanto vocês podem fazer no máximo T trocas entre alunos vizinhos.

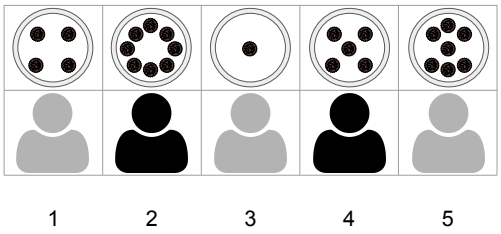
O diagrama abaixo ilustra um exemplo para $N = 5$, $K = 2$ e $T = 2$. Existem 5 pratos com 4, 8, 1, 5 e 7 brigadeiros, da esquerda para a direita. o grupo possui dois amigos que estão inicialmente nas posições 1 e 3, e faltam 2 segundos para os brigadeiros serem servidos.



No primeiro segundo, o membro na posição 1 pode trocar de lugar com o aluno na posição 2:



No último segundo, o membro na posição 3 pode trocar de lugar com o aluno na posição 4:



Desta forma, ao final dos dois segundos, os amigos do grupo estarão nas posições 2 e 4, e portanto irão comer 8 e 5 brigadeiros, respectivamente, totalizando $8+5 = 13$ brigadeiros. É possível verificar que esta é a quantidade máxima de brigadeiros que o grupo pode comer de acordo com as condições do exemplo.

Dados o número de alunos na festa (que também corresponde ao número de pratos), o número de amigos em seu grupo, o número de segundos que faltam para os brigadeiros serem servidos, quantos brigadeiros cada prato possui, e as posições iniciais dos membros do grupo, sua tarefa é determinar o máximo número de brigadeiros que seu grupo consegue comer no total. Observe que pode ser ótimo gastar alguns segundos sem realizar nenhuma troca.

Entrada

A primeira linha da entrada contém três inteiros N, K e T, a quantidade de pratos de brigadeiros, a quantidade de amigos no grupo e quantos segundos restam para os brigadeiros serem servidos, respectivamente.

A segunda linha representa os pratos de brigadeiros e contém N inteiros separados por espaços em branco. O i-ésimo destes inteiros é P_i, a quantidade de brigadeiros no i-ésimo prato.

A terceira e última linha representa as posições iniciais dos alunos na mesa e contém N inteiros separados por espaços em branco. O i-ésimo destes inteiros, G_i, indica se o aluno sentado na i-ésima cadeira é membro do grupo ou não: $G_i = 1$ se o aluno é membro do grupo, ou $G_i = 0$ caso contrário. É garantido que existem exatamente K valores iguais a 1 nesta linha.

Saída

Seu programa deverá produzir uma única linha contendo somente um inteiro, a quantidade máxima de brigadeiros que seu grupo consegue comer.

Restrições

- $1 \leq N \leq 300$
- $0 \leq P_i \leq 9$ para todo $1 \leq i \leq N$
- $1 \leq K \leq N$
- $0 \leq T \leq 1\,000\,000\,000$
- $G_i = 0$ ou $G_i = 1$ para todo $1 \leq i \leq N$
- Existem exatamente K índices i para os quais $G_i = 1$

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 (0 pontos):** Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (13 pontos):**
  - $N \leq 50$
  - $K = 3$
  - $T \leq 1\,000$
- **Subtarefa 3 (22 pontos):**
  - $N \leq 16$
  - $T \leq 1\,000$
- **Subtarefa 4 (23 pontos):**
  - $N \leq 50$
  - $T \leq 1\,000$
- **Subtarefa 5 (10 pontos):**
  - $N \leq 50$
  - $T \leq 100\,000$
- **Subtarefa 6 (11 pontos):**
  - $N \leq 100$
- **Subtarefa 7 (21 pontos):** Sem restrições adicionais.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
5 2 2 4 8 1 5 7 1 0 1 0 0	13

Explicação do exemplo 1: Este é o exemplo mostrado no enunciado.

Exemplo de entrada 2	**Exemplo de saída 2**
4 2 3 8 9 1 5 0 1 0 1	17

Exemplo de entrada 3	**Exemplo de saída 3**
4 2 2 8 9 1 5 0 1 0 1	14

Explicação do exemplo 3: Neste exemplo, os amigos não tem tempo suficiente para conseguir comer 17 brigadeiros (como no exemplo 2). A melhor opção é que eles continuem em suas posições iniciais e comam $9 + 5 = 14$ brigadeiros.

Exemplo de entrada 4	**Exemplo de saída 4**
15 7 100 7 3 0 8 6 1 9 1 5 8 1 6 3 4 9 1 1 0 0 1 0 1 0 0 0 1 0 1 1 0	53

Construtora

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

Em uma rua existem N prédios adjacentes, o i-ésimo entre eles (da esquerda para a direita) possui a_i andares. Uma construtora acabou de comprar todos os prédios da rua, e fará uma obra para que eles tenham a mesma altura (quantidade de andares), porém, para realizar esta obra ela deve seguir as regras definidas pela Sociedade Brasileira de Construção (SBC).

Segundo as regras da SBC, cada fase da obra só pode aumentar em uma unidade a quantidade de andares de prédios consecutivos que já possuam uma mesma quantidade de andares. Sendo mais específico, para cada fase:

- A construtora deve definir um índice inicial L
- A construtora deve definir um índice final R
- Todos os prédios entre L e R (incluindo L e R) devem possuir a mesma quantidade de andares, digamos a
- Ao final desta fase da obra, todos os prédios entre L e R (incluindo L e R) vão possuir $a + 1$ andares

Por exemplo, suponha que N seja igual a 8, e que as alturas iniciais sejam: $a_1 = 5, a_2 = 4, a_3 = 4, a_4 = 4, a_5 = 5, a_6 = 4, a_7 = 4$ e $a_8 = 7$, perceba que:

- A construtora não pode definir $L = 2$ e $R = 7$ para a primeira fase, pois nem todas as quantidades de andares nesse intervalo de prédios são iguais
- A construtora não pode definir $L = 1$ e $R = 5$ para a primeira fase, pois nem todas as quantidades de andares nesse intervalo de prédios são iguais
- A construtora pode definir $L = 8$ e $R = 8$ para a primeira fase, e a quantidade de andares final seria: $a_1 = 5, a_2 = 4, a_3 = 4, a_4 = 4, a_5 = 5, a_6 = 4, a_7 = 4$ e $a_8 = 8$
- A construtora pode definir $L = 2$ e $R = 4$ para a primeira fase, e a quantidade de andares final seria: $a_1 = 5, a_2 = 5, a_3 = 5, a_4 = 5, a_5 = 5, a_6 = 4, a_7 = 4$ e $a_8 = 7$

Para cada fase da obra, a construtora deve pagar impostos para a prefeitura, portanto, ela deseja completar a tarefa usando a menor quantidade possível de fases. No exemplo mostrado anteriormente a menor quantidade de fases é 4, você consegue enxergar como?

Dadas a quantidade inicial de andares de cada um dos N prédios, sua tarefa é determinar a quantidade mínima de fases para completar a obra.

Entrada

A primeira linha da entrada contém um único inteiro N, o número de prédios da rua.

A segunda linha contém N inteiros, o i-ésimo entre esses inteiros é a_i, a quantidade de andares do i-ésimo prédio.

Saída

A saída deve conter um único número inteiro, a menor quantidade de fases necessárias para a construtora completar a obra, ou seja, deixar todos os prédios com a mesma altura.

Restrições

- $2 \leq N \leq 100$.
- $1 \leq a_i \leq 100$.

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 (0 pontos):** Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (17 pontos):** $a_i = a_j$ para todo $1 \leq i, j \leq N - 1$, ou seja, todos os $N - 1$ primeiros prédios possuem a mesma altura inicial.
- **Subtarefa 3 (18 pontos):** $1 \leq a_i \leq 2$.
- **Subtarefa 4 (19 pontos):** $1 \leq a_i \leq 3$.
- **Subtarefa 5 (20 pontos):** $a_i > a_j$ para todo $1 \leq i < j \leq N$, ou seja, as alturas iniciais dos prédios são decrescentes da esquerda para a direita.
- **Subtarefa 6 (26 pontos):** Sem restrições adicionais.

Exemplos

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

Explicação do exemplo 1: A construtora define $L = 2$ e $R = 3$ para a primeira fase da obra, ao final as alturas serão: 3 2 2 2. Depois, a construtora define $L = 2$ e $R = 4$ para a segunda fase da obra, ao final as alturas serão: 3 3 3 3, e portanto a obra está finalizada com 2 fases.

Exemplo de entrada 2	**Exemplo de saída 2**
8 5 4 4 4 5 4 4 7	4

Explicação do exemplo 2: Este é o exemplo mostrado no enunciado.

Exemplo de entrada 3	**Exemplo de saída 3**
3 100 100 100	0

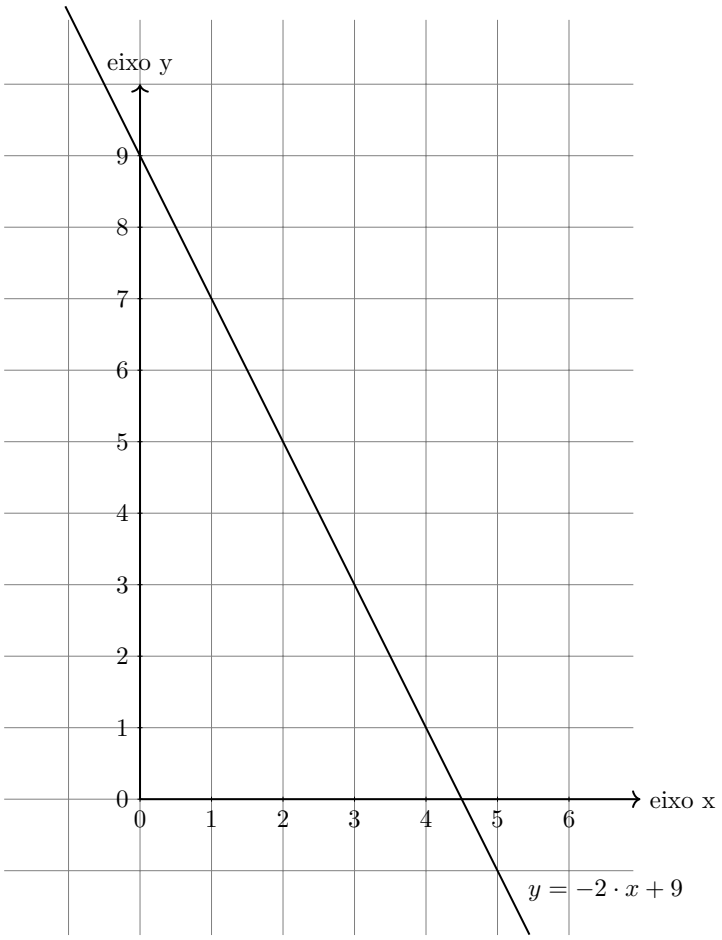
Retas

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

Juan é um artista talentoso que ama criar obras com padrões geométricos, e sua última criação envolve desenhar várias linhas retas em uma imensa tela infinita. Cada linha que ele desenha é representada por uma equação da forma $y = A \cdot x + B$, onde:

- A é a inclinação da linha, indicando o quanto ela sobe ou desce.
- B é o ponto em que a linha cruza o eixo y (a interseção com o eixo vertical).

Por exemplo, a figura a seguir mostra a linha $y = -2 \cdot x + 9$:

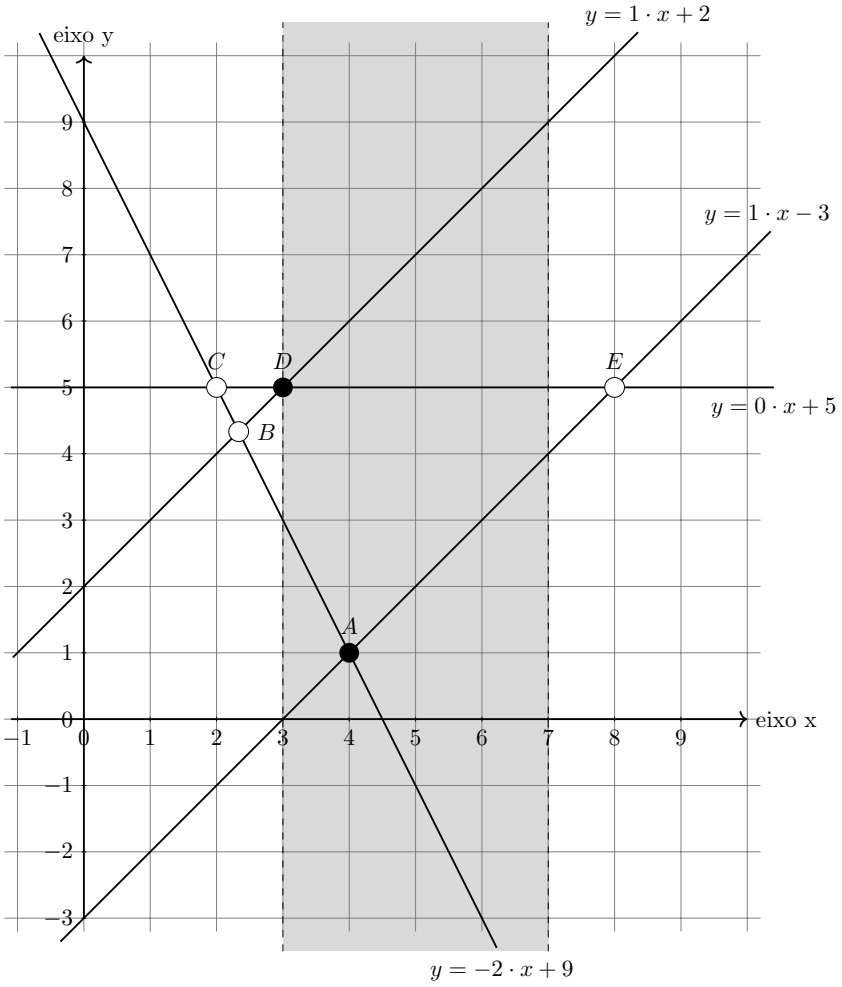


Depois de desenhar N dessas linhas, Juan começou a observar que algumas delas se cruzavam, criando pontos de interseção. Fascinado por esses cruzamentos, ele decidiu focar em uma região específica da sua tela, delimitada entre dois valores no eixo x: X_1 e X_2. Ele quer saber quantas interseções ocorrem dentro desse intervalo de x.

Por exemplo, se Juan desenhou as seguintes $N = 4$ linhas:

- $y = -2 \cdot x + 9$
- $y = 1 \cdot x - 3$
- $y = 1 \cdot x + 2$
- $y = 0 \cdot x + 5$

E deseja saber quantas interseções existem na região entre $X_1 = 3$ e $X_2 = 7$, podemos visualizar na figura a seguir que a resposta é 2:



Perceba que os pontos de interseção A e D estão na região de interesse de Juan, já os pontos C, B e E estão fora dessa região.

Sua tarefa é ajudar Juan a descobrir o número de interseções entre as N linhas que possuem o valor da coordenada x entre X_1 e X_2 (incluindo X_1 e X_2).

Entrada

A primeira linha contém três inteiros N, X_1, X_2, onde N é o número de linhas desenhadas por Juan, X_1 é o limite inferior da coordenada x da região de interesse de Juan, enquanto X_2 é o limite superior dessa região.

As próximas N linhas descrevem as equações das linhas, cada uma contendo dois inteiros A_i e B_i, onde A_i é a inclinação da i-ésima linha, e B_i é o ponto em que a linha cruza o eixo y.

Saída

A saída deve conter um único número inteiro, a quantidade de interseções entre as N linhas que possuem o valor da coordenada x entre X_1 e X_2 (incluindo X_1 e X_2).

Restrições

- $1 \leq N \leq 10^5$
- $-10^9 \leq X_1 \leq X_2 \leq 10^9$

- $-10^9 \leq A_i, B_i \leq 10^9$
- $A_i \neq A_j$ ou $B_i \neq B_j$ para todo $1 \leq i < j \leq N$. Ou seja, não existem duas linhas iguais.

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 (0 pontos):** Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (16 pontos):** $N = 2$
- **Subtarefa 3 (12 pontos):** $N \leq 1000$
- **Subtarefa 4 (11 pontos):** $A_i = A_j$ para todo $1 \leq i, j \leq N - 1$, ou seja, todas as $N - 1$ primeiras retas possuem a mesma inclinação.
- **Subtarefa 5 (15 pontos):** $X_1 = X_2$
- **Subtarefa 6 (18 pontos):** $X_2 = X_1 + 1$ e $A_i \leq 30$ para $1 \leq i \leq N$.
- **Subtarefa 7 (28 pontos):** Nenhuma restrição adicional.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
4 3 7 -2 9 1 -3 1 2 0 5	2

Explicação do exemplo 1: Este é o exemplo mostrado no enunciado.

Exemplo de entrada 2	**Exemplo de saída 2**
2 2 3 -2 9 1 2	1

Exemplo de entrada 3	**Exemplo de saída 3**
5 100 200 2 213 2 209 2 210 2 410 4 10	3

Exemplo de entrada 4	**Exemplo de saída 4**
4 1 1 15 -5 0 10 -5 15 -20 30	6

Explicação do exemplo 4: As 4 retas têm o mesmo $y = 10$ em $x = 1$ (note que $X_1 = X_2 = 1$), então todos os 6 pares de retas se intersectam. Perceba que o mesmo ponto pode ser contabilizado mais de uma vez caso ele seja a interseção de mais de um par de retas.

Burocracia

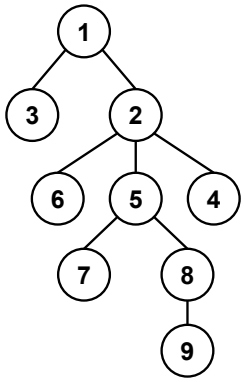
Nome do arquivo: `burocracia.c`, `burocracia.cpp`, `burocracia.java`, `burocracia.js` *ou* `burocracia.py`

O reino de Nlogônia funciona de uma maneira bem interessante. A fim de garantir que todos os nobres estão sendo monitorados, cada um deles possui exatamente um superior direto, de tal forma que o nobre i é subordinado direto do nobre $p[i]$. A única exceção a essa regra é o rei de Nlogônia, que será representado pelo índice 1 e não possui nenhum superior. Note que um nobre pode supervisionar vários outros nobres.

De tempos em tempos, nobres precisam enviar relatórios para algum superior que está a k níveis acima na hierarquia. Por exemplo se o nobre i precisa enviar um relatório para alguém 3 níveis acima, ele será entregue para o nobre $p[p[p[i]]]$. Isso tornou o funcionamento do reino extremamente burocrático, pois dependendo do número de níveis, a comunicação interna do reino se torna prejudicada.

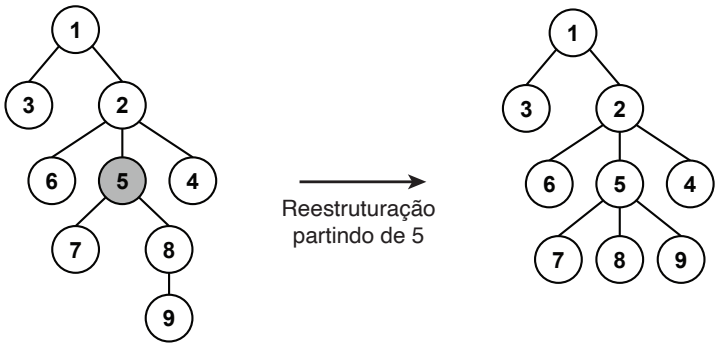
Para remediar esse problema, o rei de Nlogônia fará decretos da seguinte forma: ele escolherá algum nobre v e todos os nobres que estão subordinados a v (direta ou indiretamente) passarão a ser subordinados diretos de v. Em outras palavras para todo u tal que $p[p[\ldots u \ldots]] = v$, $p[u]$ passa a ser v. Dizemos que essa é uma operação de reestruturação partindo de v.

Por exemplo, considere que o reino tem 9 nobres e a estrutura do reino era tal que $p[2] = 1, p[3] = 1, p[4] = 2, p[5] = 2, p[6] = 2, p[7] = 5, p[8] = 5$ e $p[9] = 8$, conforme mostrado na figura abaixo:



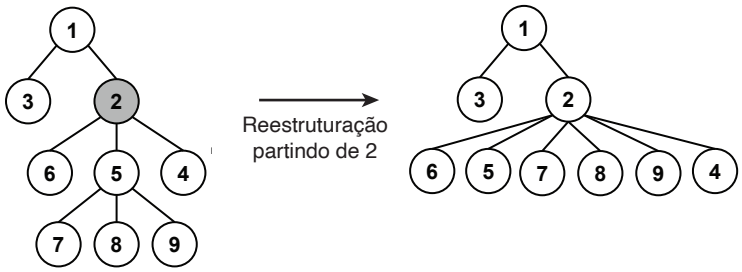
Se neste momento o nobre 9 tiver que entregar um relatório para o superior que está 3 níveis acima, ele deverá entregar ao nobre 2. Mas se o nobre 4 tiver que entregar um relatório para o superior que está 2 níveis acima, ele deverá entregar ao nobre 1.

Caso aconteça uma reestruturação partindo do 5, os superiores diretos dos nobres passariam a ser $p[2] = 1, p[3] = 1, p[4] = 2, p[5] = 2, p[6] = 2, p[7] = 5, p[8] = 5$ e $p[9] = 5$, como na figura a seguir:



Se neste momento o nobre 9 tiver que entregar um relatório para o superior que está 3 níveis acima, agora ele deverá entregar ao nobre 1.

Caso aconteça mais uma reestruturação, mas agora partindo do 2, os superiores diretos dos nobres passariam a ser $p[2] = 1, p[3] = 1, p[4] = 2, p[5] = 2, p[6] = 2, p[7] = 2, p[8] = 2$ e $p[9] = 2$, como na figura abaixo:



Se neste momento o nobre 8 tiver que entregar um relatório para o superior que está 1 nível acima, ele deverá entregar ao nobre 2.

Entretanto, esses decretos criaram outro problema: os nobres não sabem mais pra quem eles têm que entregar relatórios! Sendo assim, o Rei de Nlogônia te pediu ajuda. Dadas as operações de reestruturação que acontecem ao longo do tempo, faça um programa que responda aos nobres para quem eles devem entregar relatórios naquele momento, ou seja, qual nobre está k níveis acima na hierarquia.

Entrada

A primeira linha de entrada possui um único inteiro N, o número de nobres no reino de Nlogônia.

A segunda linha de entrada possui $N - 1$ inteiros, $p[2], p[3], \ldots, p[N]$, os superiores diretos de cada nobre.

A terceira linha de entrada possui um único inteiro Q, o número de operações que você deve processar.

As próximas Q linhas da entrada representam as operações que aconteceram no reino de Nlogônia:

- Se o nobre v_j precisa entregar um relatório para um superior k_j níveis acima, a j-ésima dessas linhas será da forma 1 v_j k_j
- Se o rei decidiu que haverá uma reestruturação a partir do nobre v_j, a j-ésima dessas linhas será da forma 2 v_j

Saída

Para cada relatório descrito na entrada, imprima uma linha com um único inteiro: o índice do nobre para o qual será entregue o relatório naquele momento.

Restrições

- $2 \leq N \leq 10^5$
- $1 \leq Q \leq 5 \cdot 10^4$
- $p[i] < i$
- $1 \leq v_j, k_j \leq N$
- Para toda entrega de relatório, é garantido que há um superior k_j níveis acima de v_j

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 (0 pontos):** Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (13 pontos):** $p[i+1] = i$ para todo $1 \leq i < N$. Ou seja, a estrutura dos nobres forma uma linha. *(Veja o exemplo 2.)*
- **Subtarefa 3 (28 pontos):** $N, Q \leq 500$
- **Subtarefa 4 (12 pontos):**
  - $N = 2^m - 1$ para algum inteiro m
  - $p[i] = \lfloor \frac{i}{2} \rfloor$ para todo $2 \leq i \leq N$.
  - Note que $\lfloor x \rfloor$ é definido como o maior inteiro menor ou igual a x
  - Ou seja, a estrutura dos nobres forma uma árvore binária completa. *(Veja o exemplo 3.)*
- **Subtarefa 5 (27 pontos):** $k_j = 1$ para todo $1 \leq j \leq Q$. Ou seja, os nobres só têm que entregar relatórios para superiores diretos.
- **Subtarefa 6 (20 pontos):** Sem restrições adicionais.

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.

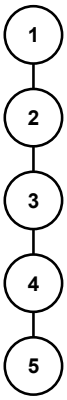
Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
9 1 1 2 2 2 5 5 8 6 1 9 3 1 4 2 2 5 1 9 3 2 2 1 8 1	2 1 1 2

Explicação do exemplo 1: Este é o exemplo mostrado no enunciado.

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

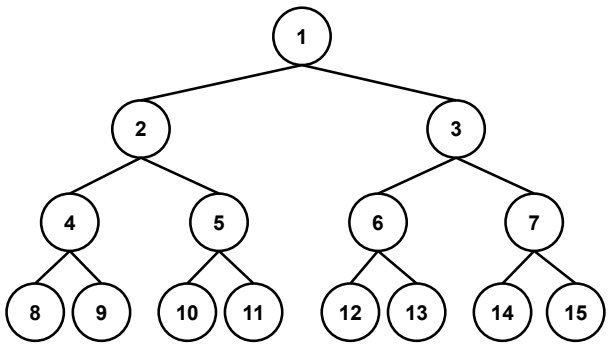
Explicação do exemplo 2: A estrutura inicial dos nobres forma uma linha, conforme a figura a seguir:



Vale ressaltar que esse exemplo satisfaz as restrições da subtarefa 2.

Exemplo de entrada 3	**Exemplo de saída 3**
15 1 1 2 2 3 3 4 4 5 5 6 6 7 7 6 1 12 2 2 3 1 12 2 1 10 1 2 2 1 10 1	3 1 5 2

Explicação do exemplo 3: A estrutura inicial dos nobres forma uma árvore binária completa, conforme a figura a seguir:



Vale ressaltar que esse exemplo satisfaz as restrições da subtarefa 4.

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

Explicação do exemplo 4: Vale ressaltar que esse exemplo satisfaz as restrições das subtarefas 4 e 5.

Jogo de Pratos

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

Depois de perder tudo no Jogo do Poder, só resta uma coisa para Jonathan: *O Jogo de Pratos*. O objetivo do jogo é muito simples: obter a maior quantidade de pratos possível. Entretanto, para atingir esse objetivo será necessário muito planejamento.

A principal maneira de obter mais pratos no jogo é através de "efeitos". Cada efeito pode ser representado por um par de inteiros (a, b). Se o jogador possui x pratos antes de um efeito entrar em ação, ele passa a possuir $a \cdot x + b$ pratos depois do efeito. Dizemos que a é o fator multiplicativo do efeito e b é o fator aditivo do efeito.

Existem dois tipos de efeitos: feitiços e refeições. Jonathan possui N feitiços e M refeições. Cada feitiço pode ser usado quantas vezes o Jonathan quiser, mas ele só tem mana para usar feitiços K vezes no total. Enquanto isso, refeições podem ser usadas uma vez cada, mas Jonathan pode escolher a ordem que elas são utilizadas. Note que tanto feitiços quanto refeições são efeitos, e portanto funcionam da maneira descrita acima quando ativados.

A fim de obter uma quantidade insana de pratos, Jonathan irá fazer um "combo", combinando vários desses efeitos. Sendo assim, Jonathan irá utilizar todos os feitiços que decidir e logo depois usará todas as refeições em certa ordem.

Por exemplo, se os feitiços disponíveis forem $(2, 0), (3, 0), (1, 2), (2, 1)$, as refeições disponíveis forem $(2, 3), (4, 2), (7, 1)$ e Jonathan pode usar 3 feitiços, ele poderia escolher utilizar os feitiços $4, 2, 4$ e depois usar as refeições $2, 1, 3$. Sendo assim, caso ele começasse com 1 prato, esse seria o processo que sua quantidade de pratos passaria:

- Jonathan começa com 1 prato. Ao usar o feitiço 4 representado por $(2, 1)$ ele fica com $2 \cdot 1 + 1 = 3$ pratos.
- Jonathan está com 3 pratos. Ao usar o feitiço 2 representado por $(3, 0)$ ele fica com $3 \cdot 3 + 0 = 9$ pratos.
- Jonathan está com 9 pratos. Ao usar o feitiço 4 representado por $(2, 1)$ ele fica com $2 \cdot 9 + 1 = 19$ pratos.
- Jonathan está com 19 pratos. Ao usar a refeição 2 representada por $(4, 2)$ ele fica com $4 \cdot 19 + 2 = 78$ pratos.
- Jonathan está com 78 pratos. Ao usar a refeição 1 representada por $(2, 3)$ ele fica com $2 \cdot 78 + 3 = 159$ pratos.
- Jonathan está com 159 pratos. Ao usar a refeição 3 representada por $(7, 1)$ ele fica com $7 \cdot 159 + 1 = 1114$ pratos.

Sendo assim, Jonathan termina com 1114 pratos. Note que essa não necessariamente é a decisão que maximiza a quantidade de pratos.

Como há muitas possibilidades de cumprir sua tarefa, Jonathan pediu sua ajuda! Ele te fará Q perguntas, e em cada uma delas ele quer saber qual é o maior número de pratos alcançável a partir de alguma estratégia válida ao começar com uma quantidade x de pratos. Lembre-se que em uma estratégia válida, Jonathan usará refeições apenas **depois** dos feitiços. Como essa resposta pode ser muito grande, imprima ela módulo $10^9 + 7$.

IMPORTANTE: A resposta não é o maior módulo possível, e sim o módulo do maior número de pratos. Ou seja, o maximização deve ocorrer **antes** de considerar o módulo.

Entrada

A primeira linha de entrada possui 3 inteiros N, M e K, o número de feitiços, o número de refeições e a quantidade de feitiços que Jonathan pode utilizar.

A segunda linha de entrada possui N inteiros: $a_1, a_2, \ldots, a_N$, os fatores multiplicativos dos feitiços.

A terceira linha de entrada possui N inteiros: $b_1, b_2, \ldots, b_N$, os fatores aditivos dos feitiços.

A quarta linha de entrada possui M inteiros: $a'_1, a'_2, \ldots, a'_M$, os fatores multiplicativos das refeições.

A quinta linha de entrada possui M inteiros: $b'_1, b'_2, \ldots, b'_M$, os fatores aditivos das refeições.

A sexta linha de entrada possui um único inteiro Q, o número de perguntas que Jonathan fará para você.

A sétima linha de entrada possui Q inteiros $x_1, x_2, \ldots, x_Q$, a quantidade de pratos inicial para cada pergunta.

Saída

Seu programa deve imprimir Q linhas, cada uma com um único inteiro: a resposta para cada uma das perguntas de Jonathan, módulo $10^9 + 7$.

Restrições

- $1 \leq N \leq 10^5$
- $1 \leq M \leq 10^5$
- $1 \leq K \leq 10^9$
- $1 \leq a_i, a'_i \leq 10^9$
- $0 \leq b_i, b'_i \leq 10^9$
- $a_i + b_i > 1, a'_i + b'_i > 1$, ou seja, nunca haverá um efeito representado por $(1, 0)$
- $1 \leq Q \leq 10^5$
- $1 \leq x_i \leq 10^9$

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 (0 pontos):** Esta subtarefa é composta apenas pelos exemplos mostrados abaixo. Ela não vale pontos, serve apenas para que você verifique se o seu programa imprime o resultado correto para os exemplos.
- **Subtarefa 2 (7 pontos):** $N = M = K = 2$
- **Subtarefa 3 (13 pontos):** $M = 1, Q = 1$, é garantido que a quantidade final de pratos da resposta será menor que $10^9 + 7$, ou seja, a resposta não será grande o suficiente para que seja necessário considerar o módulo.

- **Subtarefa 4 (15 pontos):** $N, Q, K \leq 500, M = 1$.
- **Subtarefa 5 (24 pontos):** $N, Q, K, M \leq 500$.
- **Subtarefa 6 (19 pontos):** $N, Q \leq 3000$
- **Subtarefa 7 (22 pontos):** Sem restrições adicionais.

Exemplos

Exemplo de entrada 1	**Exemplo de saída 1**
2 2 2 2 3 3 2 5 2 5 2 1 1	185

Explicação do exemplo 1: Os feitiços disponíveis são $(2, 3)$ e $(3, 2)$, enquanto as refeições são $(5, 5)$ e $(2, 2)$.
Há duas possíveis soluções ótimas nesse caso:

- Usar os feitiços na ordem $1, 2$ e as refeições na ordem $2, 1$, totalizando $(((1 \cdot 2 + 3) \cdot 3 + 2) \cdot 2 + 2) \cdot 5 + 5 = 185$ pratos.
- Usar os feitiços na ordem $2, 2$ e as refeições na ordem $2, 1$, totalizando $(((1 \cdot 3 + 2) \cdot 3 + 2) \cdot 2 + 2) \cdot 5 + 5 = 185$ pratos.

Note que esse exemplo satisfaz a subtarefa 2.

Exemplo de entrada 2	**Exemplo de saída 2**
5 1 1000000000 1 1 1 1 1 5 4 3 2 1 1 35 4 1 100 200000001 200000002	1 100 200000001 200000002

Explicação do exemplo 2: A resposta para a primeira pergunta consiste em usar o feitiço 1 todas as 10^9 vezes, e logo depois usar a refeição 1. Com essa estratégia, Jonathan consegue 5000000036 pratos.
Note que 5000000036 módulo $10^9 + 7$ é igual a 1, pois o resto da divisão de 5000000036 por $10^9 + 7$ é 1.

Exemplo de entrada 3	**Exemplo de saída 3**
4 3 3 2 3 1 2 0 0 2 1 2 4 7 3 2 1 4 1 2 10000 3	1611 3123 15120099 4635

Explicação do exemplo 3: Este é o exemplo mostrado no enunciado, com algumas perguntas adicionais.

Exemplo de entrada 4	**Exemplo de saída 4**
2 2 2 1000000 2000000 32 57 32 32 5 9 5 1 2 3 4 5	707385849 678713849 650041849 621369849 592697849

Explicação do exemplo 4: Esse exemplo satisfaz a subtarefa 2.

Exemplo de entrada 5	**Exemplo de saída 5**
5 4 1000 807989 325717 452373 639688 388457 524139 933016 888648 755670 919486 647772 75628 922474 745349 379795 828349 203942 120165 5 1 2 3 4 5	843013087 329815085 987387086 644959080 302531074