Problema A

Enunciado

Uma escola está organizando um evento e possui **n** alunos, numerados de 1 a ***n***. Cada aluno pode ter um mentor que é um outro aluno, ou pode não ter um mentor, caso em que será considerado um aluno independente. Um aluno ***A*** é considerado o mentor de um aluno ***B*** se uma das seguintes condições for verdadeira:

1. O aluno ***A*** é o mentor imediato do aluno ***B***.
2. O aluno ***B*** tem um mentor ***C*** tal que o aluno ***A*** é o mentor de ***C***.

Não haverá ciclos de mentoria, ou seja, nenhum aluno poderá ser mentor de si mesmo.

Para o evento, os organizadores precisam dividir todos os **n** alunos em grupos. Cada aluno deve pertencer a exatamente um grupo. Além disso, dentro de um único grupo, não pode haver dois alunos ***A*** e ***B*** tais que ***A*** seja o mentor de ***B*** (ou vice-versa).

Qual é o número mínimo de grupos que devem ser formados para garantir que essa condição seja atendida?

Entrada

A primeira linha contém um inteiro ***n*** ($1 \leq n \leq 2000$) — o número de alunos.

As próximas ***n*** linhas contêm os inteiros p_i ($1 \leq p\_i \leq n$ ou $p_i = -1$). Cada indica o mentor imediato do i-ésimo aluno. Se $p_i = -1$, isso significa que o aluno i não possui um mentor.

Entrada	Saída
5 -1 1 2 1 -1	3

Saída

Imprima um único inteiro que denota o número mínimo de grupos que devem ser formados para o evento.

Note

Para o primeiro exemplo, três grupos são suficientes, por exemplo:

- Funcionário 1
- Funcionários 2 e 4
- Funcionários 3 e 5

Observação

Para evitar erros relacionados ao limite de recursão em Python, adicione o seguinte comando na primeira linha do seu código:

```
import sys
sys.setrecursionlimit(3000)
```

Dica

Esta seção oferece apenas uma dica sobre como o problema pode ser resolvido. Não é obrigatório seguir essa abordagem; você é livre para resolver a questão da maneira que achar mais adequada.

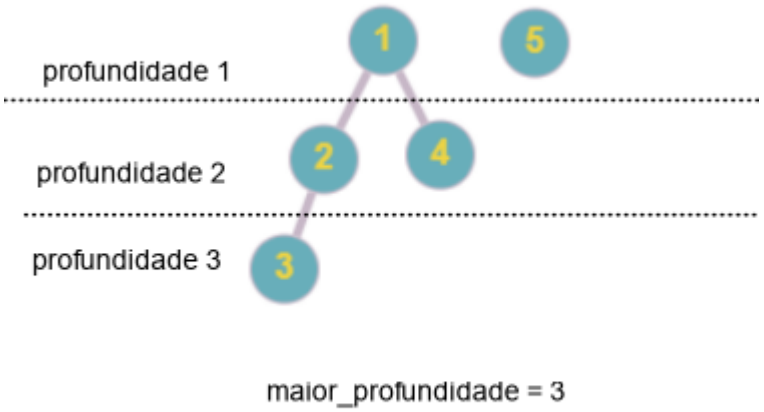
Armazene os mentores (aqueles com `p_i = -1`) em uma lista. Em seguida, execute uma busca em profundidade (DFS) para cada mentor que **retorne a profundidade máxima**. Compare e armazene o maior valor retornado pela DFS como a profundidade máxima. Por exemplo:

```
...
lista_de_mentores = [1, 5]
profundidade_maxima = ...

# Para cada mentor, chame a função DFS que retorna a #profundidade
máxima
for mentor in lista_de_mentores:
    profundidade_maxima = max(profundidade_maxima, dfs(mentor,
grafo))

# Resposta final
```

```
print(profundidade_maxima)
```



Problema B

Enunciado

Em um jogo, há diversas missões secundárias que podem ser feitas pelo jogador. Entretanto, algumas dessas missões só vão poder ser liberadas para o jogador realizá-las após a realização de outra missão. Seu dever é encontrar uma ordem válida em que o jogador pode completar as missões secundárias.

Entrada

A primeira linha da entrada contém dois números inteiros: ***n*** e ***m***. ***n*** é a quantidade de missões secundárias do jogo, e ***m*** é a quantidade de pré-requisitos que existem para as missões no jogo.

As próximas ***m*** linhas consistem em dois inteiros: m_1 e m_2, onde m_1 é a missão que é pré-requisito para liberação da missão m_2.

Saída

Imprima na saída uma ordem válida em que as missões do jogo poderão ser concluídas. Se não existir uma ordem válida (quando há uma dependência circular de missões), imprima "IMPOSSIBLE" na saída.

Observação

Para evitar erros relacionados ao limite de recursão em Python, adicione o seguinte comando na primeira linha do seu código:

```
import sys
sys.setrecursionlimit(100000)
```

Teste

Entrada	Saída
5 3 1 2 3 1 4 5	3 4 1 5 2

Problema C

Durante as férias, o monitor Eduardo decidiu se desconectar do ambiente acadêmico e se aventurar em uma nova jornada. Cansado da rotina escolar, ele resolveu visitar um parque nacional nas montanhas, onde poderia relaxar e, quem sabe, ter novas ideias para suas monitorias. Ao chegar ao parque, Eduardo se viu rodeada por uma densa floresta com árvores variadas, o que despertou seu senso de humor e a fez refletir:

“Por que o pinheiro nunca se perde na floresta?”

“Porque ele sempre acha o caminho de volta com uma pinha!”

Satisfeito com sua própria piada, Eduardo não conseguia parar de pensar nela enquanto procurava o chalé onde iria se hospedar. No entanto, não demorou muito até que ele percebesse que estava perdido. Felizmente, tinha um mapa da região à disposição.

No mapa, existem N áreas numeradas de 1 a N. Eduardo está atualmente na área 1 e o chalé está localizado na área N. Existem também M trilhas que conectam áreas distintas. Cada trilha tem uma descrição que indica o tempo, em minutos, necessário para percorrê-la.

Além disso, como a preservação ambiental é uma prioridade no parque, P das N áreas possuem pinheiros. As áreas 1 e N não possuem pinheiros. Sempre que Eduardo chega a uma área com um pinheiro, ele precisa parar por K segundos e rir da piada que criou. Ele tem T minutos antes do pôr do sol, e Eduardo deseja chegar ao chalé o mais rápido possível (para poder descansar e pensar em novas piadas para suas aulas).

Entrada

A primeira linha contém cinco inteiros: ***N*** ($2 \leq N \leq 30.000$), ***M*** ($0 \leq M \leq 100.000$), ***T*** ($0 \leq T \leq 50.000.000$), ***K*** ($1 \leq K \leq 50.000.000$) e ***P*** ($0 \leq P \leq N - 2$), conforme descrito acima. Em seguida, seguem ***P*** inteiros p_i (), que indicam as áreas com pinheiros. Após isso, seguem ***M*** linhas descrevendo as trilhas, cada uma com três inteiros ***xj***, ***yj*** e ***wj***, indicando que Eduardo pode ir da área ***xj*** para a área ***yj*** em ***wj*** minutos, para $1 \leq j \leq M$ ($xj \neq yj$; $1 \leq xj, yj \leq N$ e $1 \leq wj \leq 100.000$).

Saída

Imprima o menor tempo necessário para Eduardo chegar ao chalé. Se não for possível chegar antes do pôr do sol, imprima "-1" (sem aspas).

Examples

Input	Output
5 7 312 10 2 3 2 1 2 8 4 5 98 3 2 12 5 2 30 5 1 103 3 4 65 2 3 1	10340

Input	Output
4 6 29370 22446 1 3 4 2 32014 2 3 24 2 1 67 4 3 16 1 2 633 2 4 4298	295860

Input	Output

3 2 701 85611 2 2 1 346 3 1 9	-1

.

Dica

Esta seção oferece apenas uma dica sobre como o problema pode ser resolvido. Não é obrigatório seguir essa abordagem; você é livre para resolver a questão da maneira que achar mais adequada.

Para resolver problemas de busca do caminho mais curto em um grafo, uma abordagem eficiente é usar o **algoritmo que vimos em sala de aula**, que pode ser aplicado quando não há ciclos negativos. Neste caso, ao passar por uma área com pinheiros, Eduardo precisará parar e rir por K segundos. Esses "atrasos" devem ser considerados no cálculo do tempo total.

Portanto, além de levar em conta o tempo das trilhas, também será necessário somar o tempo de risada cada vez que Eduardo passar por uma área com pinheiros. Ao calcular o tempo para ir de uma área para outra, verifique se a nova área contém um pinheiro. Se houver, adicione o tempo de risada (K segundos) ao tempo total (isso pode ser realizado na construção do grafo). Depois, continue o algoritmo normalmente.

Problema D

Melina precisa chegar na universidade a tempo de dar a aula de ATAL. Para isso, ela precisa passar por diversos semáforos.

Imagine que os semáforos são numerados de ***1*** a ***n***, e o tempo necessário para ir de um semáforo a outro é um tempo ***t*** minutos. Entretanto, quando a professora chega em um semáforo, ele pode estar fechado no momento, o que faz a professora aumentar em um minuto o tempo de espera naquele semáforo. Imagine que para viajar de um semáforo ***k*** a um semáforo ***j*** leve 4 minutos, e que a professora chegou em ***k*** no momento em que ele está fechado. Assim, ela vai ter que esperar 1 minuto a mais para sair até o próximo semáforo, o que faz com que o tempo de chegada no semáforo ***j*** seja de 5 minutos.

Tendo em vista que você sabe quanto tempo leva entre um semáforo e outro, e em quais minutos ele estará fechado, ajude a professora Melina a saber qual a quantidade mínima de tempo para se chegar no último semáforo (semáforo ***n***), e assim poder entrar na universidade para dar aula.

Entrada

A primeira linha da entrada consiste em dois inteiros: ***n*** e ***m***, a quantidade de semáforos e a quantidade de pares de semáforos que possuem uma rua os interligando.

As próximas ***m*** linhas contém três inteiros: ***u***, ***v***, e ***w***, sendo ***u*** e ***v*** semáforos e ***w*** o tempo em minutos necessário para ir de um a outro.

Logo após, terão ***n*** linhas descrevendo os minutos em que os semáforos estarão fechados. Por exemplo, a primeira linha descreve os minutos em que o primeiro semáforo estará fechado, e assim por diante. Nessas linhas, o primeiro inteiro ***k*** descreve a quantidade de momentos em que o semáforo estará fechado, e os próximos ***k*** inteiros serão esses momentos. Por exemplo, se o primeiro número da linha for 0, então em nenhum momento o semáforo estará fechado. Se o primeiro número da linha for 2, então haverão dois momentos em que o semáforo estará fechado, e esses dois minutos serão escritos logo após esse número na linha.

Output

Imprima a quantidade de tempo que a professora irá necessitar para chegar do primeiro semáforo até o último semáforo. Se a professora não conseguir chegar no último semáforo (por não haver ruas interligando eles), então imprima ***-1***.

Teste

Entrada	Saída
4 6 1 2 2 1 3 3 1 4 8 2 3 4 2 4 5 3 4 3 0 1 3 2 3 4 0	7
3 1 1 2 3 0 1 3 0	-1

Observações

No primeiro teste, a professora tem três caminhos para escolher a partir do primeiro semáforo. Se ela escolher ir pelo semáforo 4 (último semáforo) de uma vez só, ela levará 8 minutos para chegar nele. Se ela escolher ir pelo semáforo 3, ela demora 3 minutos para chegar nele, mas no minuto 3 e 4 o semáforo estará fechado, e ela só poderá sair daquele semáforo no minuto 5, fazendo com que ela chegue no semáforo 4 com 8 minutos. Dessa forma, a melhor opção seria ir pelo semáforo 2, e depois ir para o semáforo 4, pois ela vai chegar em um total de ***2 + 5 = 7*** minutos.

No segundo teste, a professora não tem nenhum caminho para seguir até o semáforo 3, portanto a resposta será -1.

Dica

Em python, utilize uma lista de adjacências para representar o grafo. Utilize uma ***heapq*** para simular uma lista de prioridades.

```
import heapq

heapq.heapify(lista) # Transformar uma lista comum em uma heap
heapq.heappop(heap) # Para retornar o menor item da heap
heapq.heappush(heap, item) # Para adicionar um item na heap
```

Utilize um set para guardar os momentos em que os semáforos estarão fechados, para poder otimizar a verificação se um tempo está dentro dele ou não.