Reservas

Considere uma rede de transportes com no máximo duas ligações diretas entre cada par de nós distintos (uma em cada sentido). Para cada ligação, dispomos de informação sobre o número de lugares livres e o preço de cada bilhete.

Pretendemos **processar uma sequência de reservas**. Cada reserva terá a indicação do número de lugares pretendidos e da sequência de nós que define o percurso a efetuar. Esses nós são todos distintos, mas a sequência pode não corresponder a um percurso válido na rede ou pode corresponder a um percurso que já não tem lugares suficientes.

Nesses dois casos, a **reserva não é efetuada** (em nenhum troço do percurso) e será indicado o primeiro problema encontrado no pedido.

Caso contrário, a reserva é efetuada e será indicado o **montante total** a pagar. As reservas são processadas sequencialmente, podendo reduzir a disponibilidade de lugares para as seguintes. Assumimos que os horários das ligações não criam restrições.

Input

Na primeira linha, tem dois inteiros positivos n e r, sendo n o número de nós da rede e r o número de ligações. Seguem-se r linhas que descrevem as ligações. Cada uma tem quatro inteiros: origem, destino, número de lugares inicialmente disponíveis (d) e preço de cada bilhete (isto é, de cada lugar nessa ligação). Os nós são identificados por números consecutivos de 1 a n.

Finalmente, tem o número total t de reservas a processar e t linhas com os seus dados: número de lugares necessários (k), número de nós no percurso (p), e a sequência de nós que o define.

Restrições

$2 \leq n \leq 20000$	número de nós
$2 \leq r \leq 100000$	número de ramos
$0 \leq d \leq 100$	número de lugares disponíveis inicialmente numa certa ligação
$1 \leq t \leq 1000$	número de reservas a processar
$1 \leq k \leq 50$	número de lugares de uma reserva
$2 \leq p \leq 20$	número de nós de um percurso

Para CC1024:

- Todos os casos de teste satisfazem as restrições indicadas. O programa não tem de as verificar.

- Na implementação em Python, deve usar um **dicionário** para registar a informação sobre os ramos da rede (i.e., as ligações). Para cada ramo *(u,v)*, a *chave* é o par *(u,v)* e o *valor* é uma **lista** *[d,b]* em que *d* é o número de lugares que ainda existem e *b* o preço de cada bilhete.
- Note que a reserva numa ligação de um percurso só é concretizada se o percurso for válido e tiver lugares suficientes em todas as ligações.
- Para ler uma linha com vários inteiros, separados por espaços, e os colocar numa lista `x` pode escrever

```
X = list(map(int,input().split()))
```

Output

Para cada reserva, terá uma linha. Se não for possível efetuar a reserva, descreverá o **primeiro problema encontrado** na análise do trajeto da origem para o destino, podendo ser `"(x,y) inexistente"` ou `"Sem lugares suficientes em (x,y)"`, com `x` e `y` substituídos pelos valores correspondentes. Se for possível efetuar a reserva, terá `"Total a pagar: c"`, devendo `c` ser substituído pelo **montante total** a pagar.

Exemplo 1

Input

```
6 7
4 3 9 10
3 5 6 7
5 2 6 2
2 4 3 5
1 2 8 4
5 6 7 4
6 5 2 10
5
2 3 5 2 4
3 4 1 2 4 3
1 4 1 2 4 3
1 3 1 2 4
1 3 1 2 5
```

Output

```
Total a pagar: 14
Sem lugares suficientes em (2,4)
Total a pagar: 19
Sem lugares suficientes em (2,4)
(2,5) inexistente
```

Exemplo 2

Input

```
5 12
4 3 9 10
3 5 6 7
5 2 6 2
1 4 10 7
2 4 3 5
1 2 8 4
4 1 0 20
3 4 15 8
3 1 23 20
2 1 17 10
```

```
5 4 20 10
2 4 3 5
6
8 3 1 4 3
2 3 1 4 3
2 5 1 2 5 4 3
1 2 5 4
2 3 3 1 2
1 2 1 3
```

Output

```
Total a pagar: 136
Sem lugares suficientes em (4,3)
(2,5) inexistente
Total a pagar: 10
Total a pagar: 48
(1,3) inexistente
```

Exemplo 3

Input

```
5 12
4 3 9 10
3 5 6 7
5 2 6 2
1 4 10 7
2 4 3 5
1 2 8 4
4 1 0 20
3 4 15 8
3 1 23 20
2 1 17 10
5 4 20 10
2 4 3 5
4
5 3 1 4 3
2 3 1 4 3
1 2 5 4
1 2 4 1
```

Output

```
Total a pagar: 85
Total a pagar: 34
Total a pagar: 10
Sem lugares suficientes em (4,1)
```
