Alguns Problemas ligados a Grafos em Maratonas da SBC

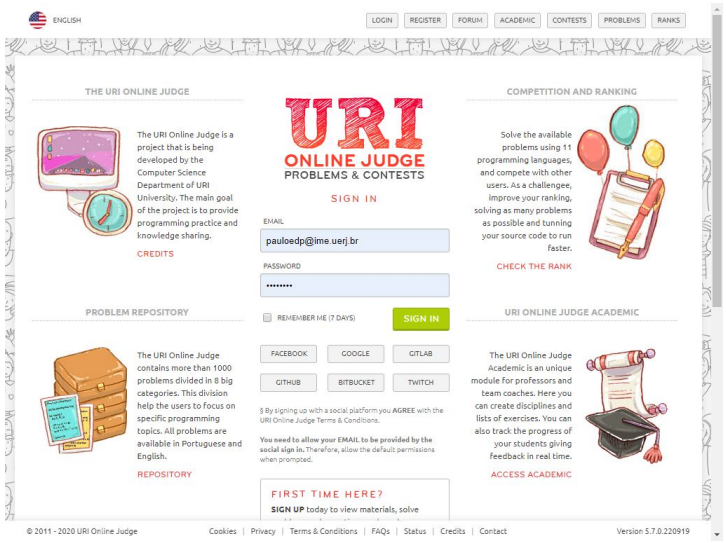
Paulo Eustáquio Duarte Pinto
(pauloedp arroba ime.uerj.br)

setembro/2020

Maratonas de Programação

- Começaram a ser divulgadas no início do século.
- Tornaram-se um instrumento motivador para o estudo de algoritmos e desenvolvimento de software
- Permitem o aperfeiçoamento de habilidades para solução de problemas
- Os resultados passaram a ser usados em seleções em empresas/universidades
- Foram criadas muitas ferramentas de apoio, notadamente os Juízes Online
- Juízes Online de destaque:
  - UVA - Universidade de Valladolid
  - URI - Univ. Reg. Integrada do Alto Uruguai...
  - Codeforces - Iniciativa de rede social russa
  - Hacker Hank - Plataforma de seleção pessoal TI

Juizes Online



Mantêm milhares de problemas cuja solução pode ser testada a qualquer momento

Hospedam competições

Disponibilizam materiais/ferramentas de treinamento

Disponibilizam ferramentas de apoio ao ensino

Problemas sobre grafos nas competições

Nas maratonas da SBC cerca de 2 problemas, em 12 envolvem grafos

Os problemas não são problemas de pesquisa, mas é necessário modelar bem e, muitas vezes, adaptar algoritmos conhecidos.

Problemas, às vezes envolvendo milhões de dados têm que ser resolvidos em 1 segundo, em geral, o que força a busca dos mais eficientes algoritmos. (1 seg ≅ 10^8 instruções)

A escolha do algoritmo é feita observando-se o tamanho da entrada:

Até n = 1.000 pode ser possível usar algoritmo $O(n^2)$

Até n = 100.000 normalmente deve-se usar $O(n \log n)$

Acima de 100.000 normalmente deve-se usar $O(n)$

Problemas sobre grafos nas competições

90% dos alunos usam C++, devido à STL, que tem prontos os tratamentos para as diversas estrutuas de dados

Em C++ existe uma estrutura de dados especial: VECTOR que é um misto de vetor e lista encadeada. Normalmente é a estrutura preferida pelos alunos.

VECTOR são vetores cujo tamanho é automaticamente estendido quando necessário. É como se fossem listas encadeadas que possibilitam acesso por índice em qualquer parte da lista.

VECTOR é, sobretudo, uma estrutura de dados prática, sem estudos teóricos associados.

Problemas em Grafos em Maratonas da SBC

(disponíveis no site do URI)

1931 - Mania de Par
2666 - Imposto Real
2962 - Arte Valiosa
1442 - Desvio de Rua
1391 - Quase o Menor Caminho
2882 - Gasolina
1476 - Caminhão
1490 - Torres que Atacam

1931 - Mania de Par

Contexto: Patricia vai fazer uma viagem onde todas as estradas são bidirecionais e têm sempre um pedágio em cada trecho. Dado o mapa das estradas quer-se saber qual o pedágio mínimo que ela vai pagar, com a restrição de que tem que ser um número par de pedágios.

Entrada: Um caso de teste. Na primeira linha, N e M ($2 \leq N \leq 10^4$, $0 \leq M \leq 50000$). A seguir vêm M linhas, com 3 inteiros C_1, C_2, indicando o par de cidades ligados e G ($\leq 10^4$) o pedágio. Patrícia vai da cidade 0 para a N-1.

Saída: Para cada teste deve ser impresso o pedágio mínimo para um percurso com um número par de pedágios. Se não for possível, imprimir -1.

Exemplo de entrada:

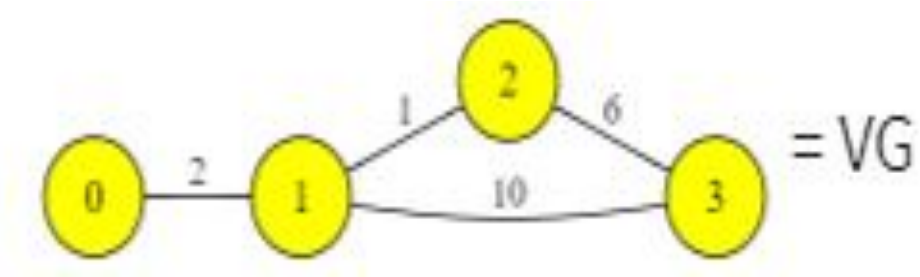
```
4  4
0  1  2
1  2  1
1  3  10
2  3  6
```

Exemplo de saída:

```
12
```

1931 - Mania de Par

O exemplo anterior corresponde ao seguinte grafo:



Distâncias mínimas de 0:

0	1	2	3
0	**19**	**3**	**12**

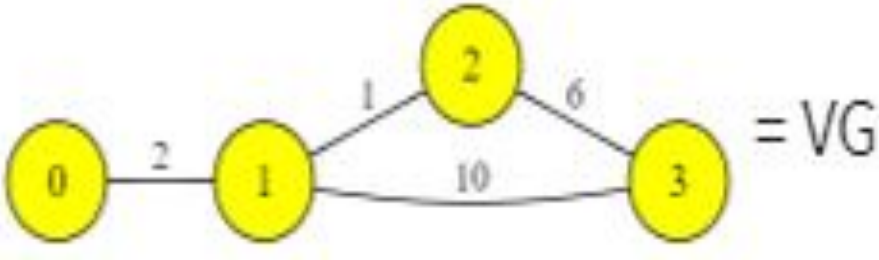
O problema pode ser resolvido de duas maneiras:

a) criando um grafo auxiliar

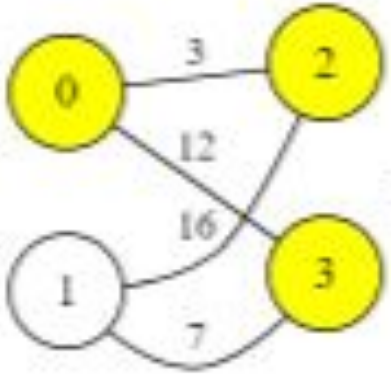
b) Modificando o algoritmo padrão de Dijkstra

1931 - Mania de Par

Primeira solução: Criar um grafo auxiliar e usar Dijkstra nesse grafo. Esse grafo auxiliar tem os mesmos vértices. As arestas refletem os menores caminhos de distância 2 de cada vértice.



= VG



= VG

Se o grafo for representado com matriz de adjacências, a criação é em $O(n^3)$ e Dijkstra em $O(n^2)$.

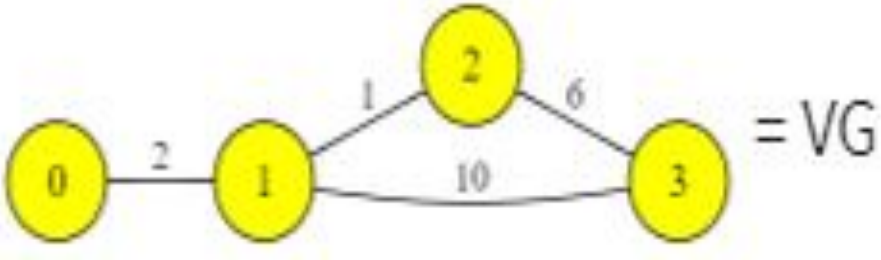
Distâncias mínimas de 0:

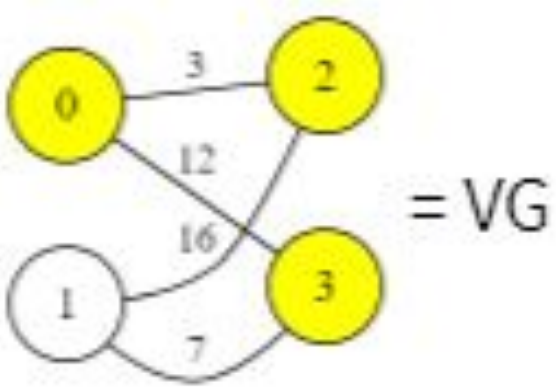
0	1	2	3
0	**19**	**3**	**12**

Pela descrição do problema, o número de arestas é O(n). Se o grafo é representado com listas de adjacências a criação é $O(n^2)$ e Dijkstra em O(n log n).

1931 - Mania de Par

Segunda solução: Modificar Dijkstra de forma a que, quando um vértice é marcado, ao invés de se analisar seus vizinhos, analisam-se os vizinhos dos vizinhos.





Se o grafo for representado com matriz de adjacências, Dijkstra em $O(n^3)$.

Distâncias mínimas de 0:

0	1	2	3
0	19	3	12

Pela descrição do problema, o número de arestas é $O(n)$. Se o grafo é representado com listas de adjacências Dijkstra será $O(n^2)$

2666 - Imposto Real

Contexto: Um reino com cidades $c_1 \ldots c_n$, sendo c_1 a capital, tem um conjunto de estradas estruturados em forma de árvore. O rei mandou recolher os impostos devidos $d_1 .. d_n$, usando uma carruagem de capacidade r. São dadas as distâncias entre cidades interligadas. Cada cidade tem um cofre muito grande. Qual a distância mínima que a carruagem deve percorrer para recolher os impostos?

Entrada: Um único caso de teste descrito em várias linhas. Na primeira vem os inteiros n, r ($2 \leq n \leq 10^4$, $1 \leq r \leq 100$). Na próxima linha vêm n inteiros, os impostos d_i devidos ($0 \leq d_i \leq 100$).Em seguida n-1 descrições das interligações: 3 inteiros A B C, A e B cidades e C a distância entre elas ($2 < A, B \leq n$, $1 \leq C, \leq 100$).

Saída: Um inteiro indicando a distância mínima a ser percorrida.

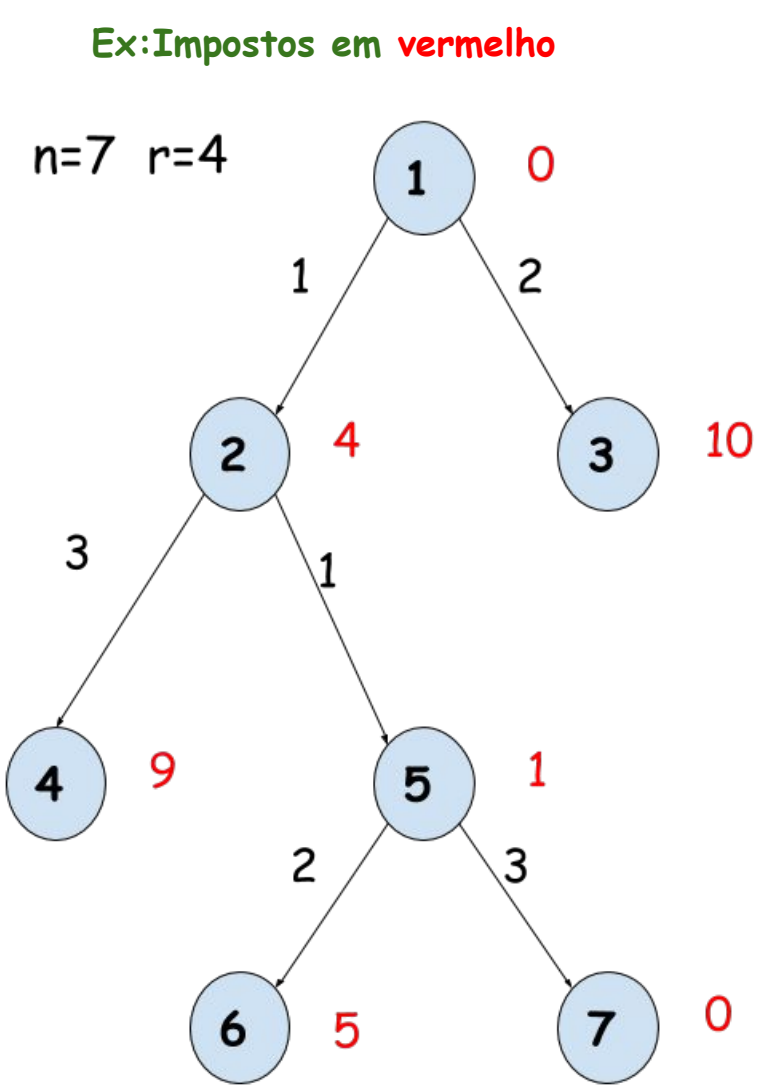
Exemplo de entrada:

```
7 4
0 4 10 9 1 5 0
1 2 1
1 3 2
2 4 3
2 5 1
5 6 2
5 7 3
```

Exemplo de saída:

```
52
```

2666 - Imposto Real



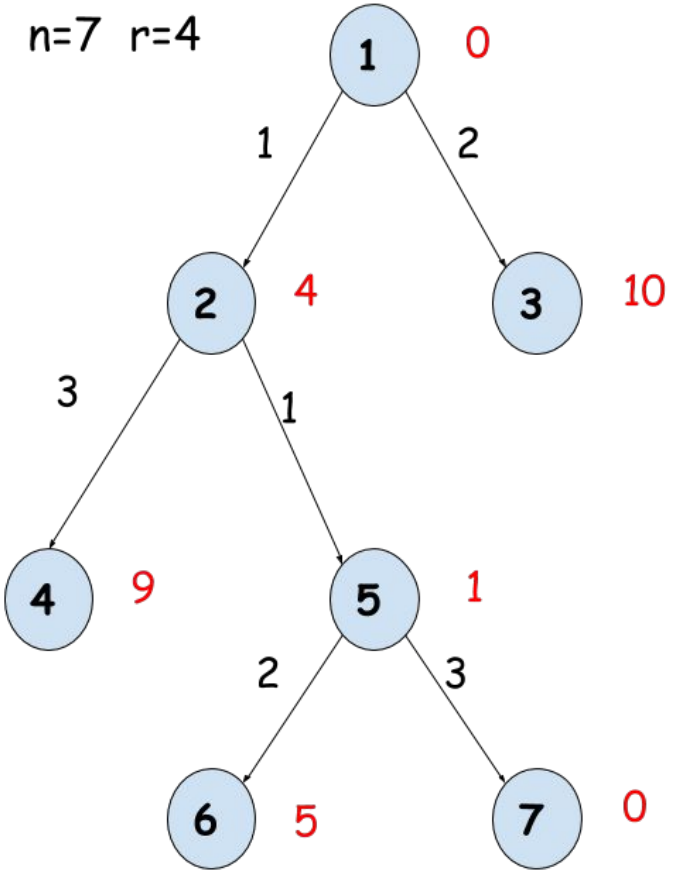
O grafo da rede de estradas é uma árvore T_u enraizada em u=1. Para minimizar a última etapa do processo, entre os nós f e u, em f já devem estar todos os impostos recolhidos na subárvore de f, D(f) pois, assim, no máximo em uma viagem a carruagem viajaria incompleta. Isso vale, então, para cada nó da árvore, o que sugere a seguinte recorrência, onde $S(T_v)$ é o percurso mínimo para a subárvore enraizada em v:

$S(T_v) = 0$, se v é uma folha
$S(T_v) = \sum(S(T_f) + 2*\lceil D(f)/r \rceil * E[u,f])$,

para f filho de v.

2666 - Imposto Real

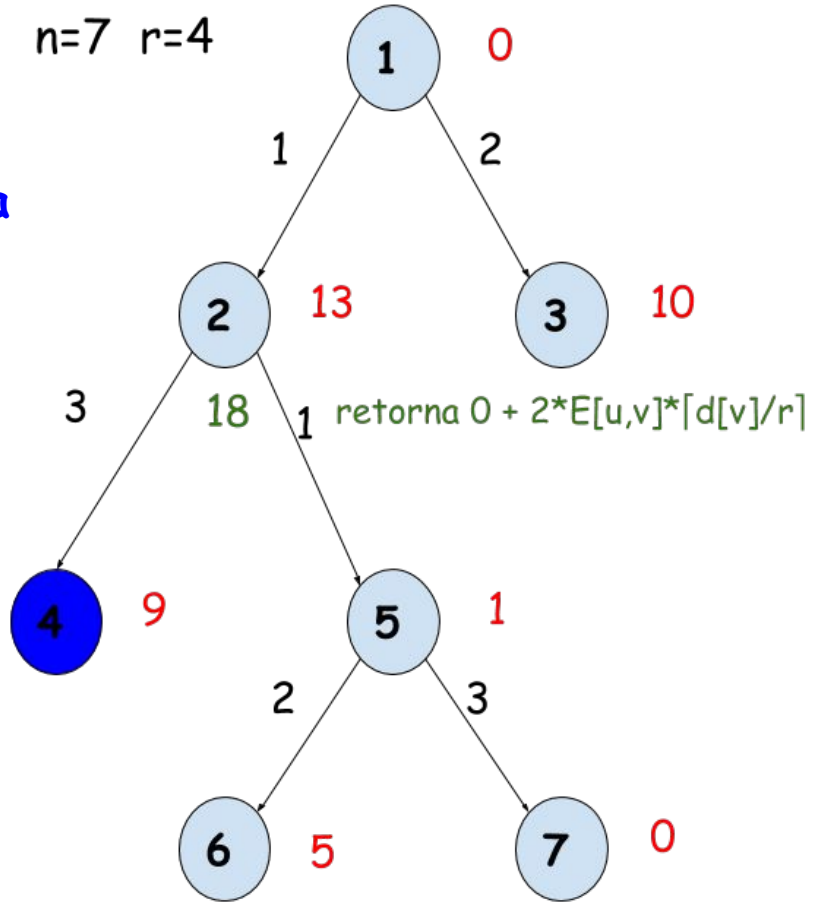
A minimização pode feita com uma Busca em Profundidade na árvore, onde, ao final do processamento de cada nó f, retorna-se $S(T_f)+ 2*\lceil D(f)/r \rceil *E[v,f]$.



2666 - Imposto Real

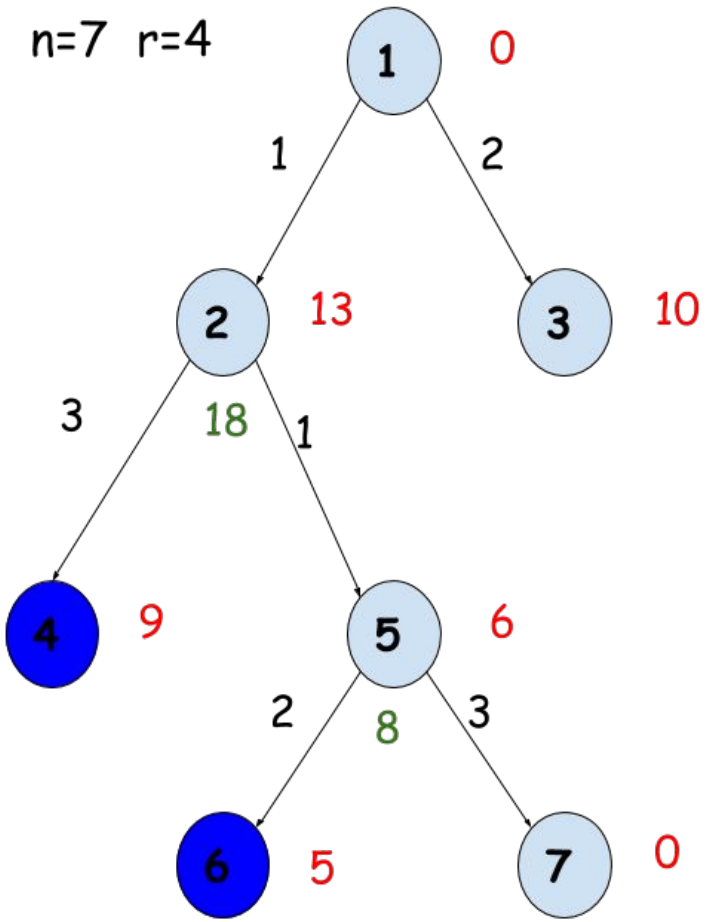
Ao final de BP(2,4):

Exemplo:
Em verde o custo mínimo para a subárvore



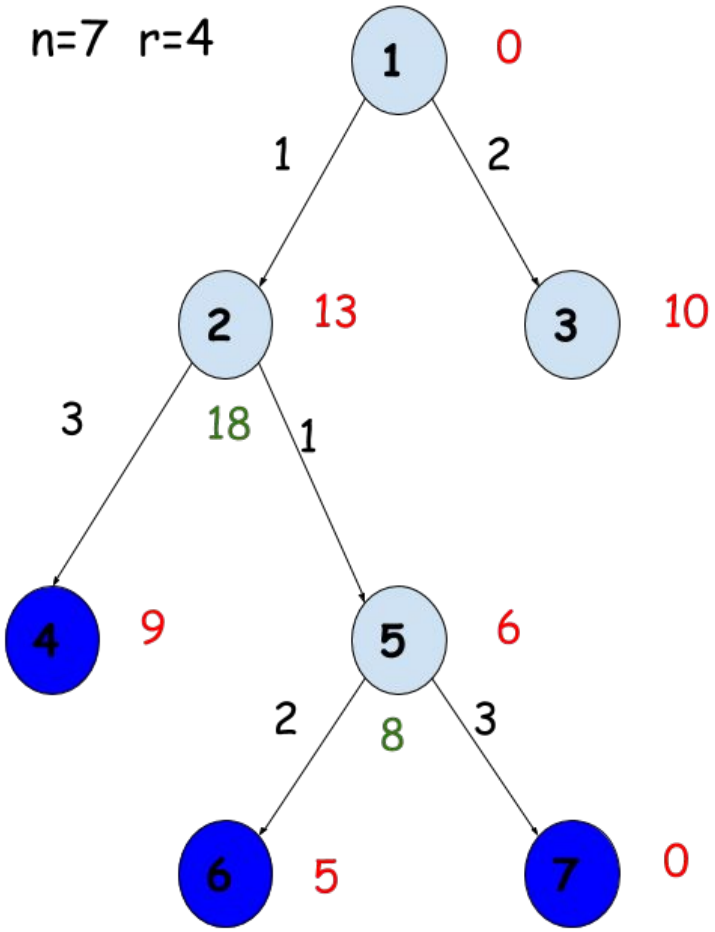
2666 - Imposto Real

Ao final de BP(5,6):



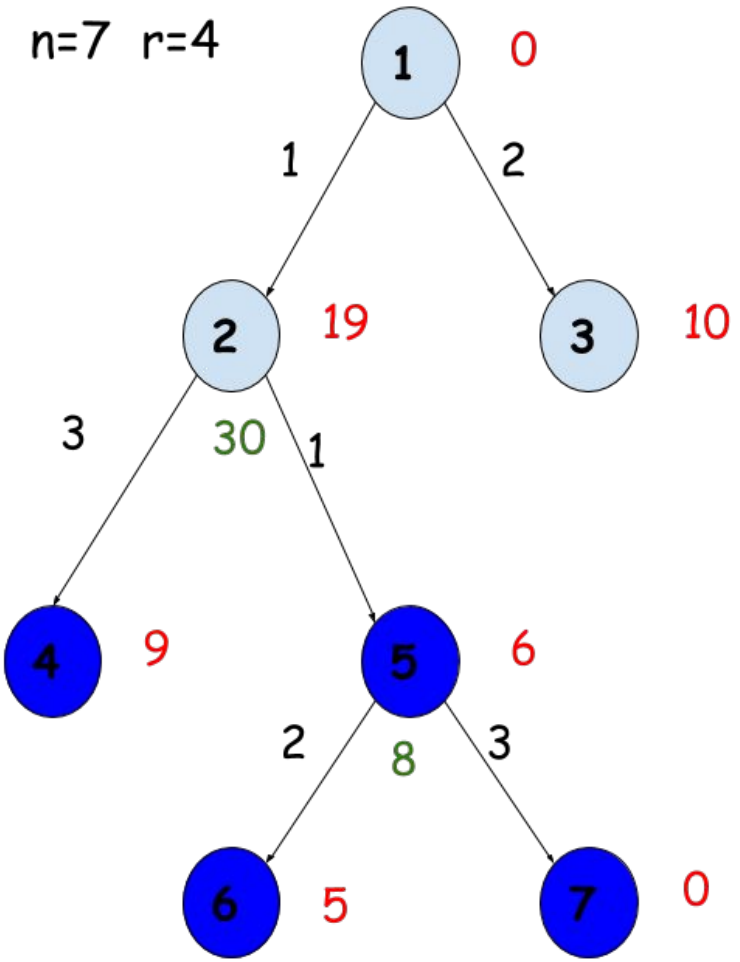
2666 - Imposto Real

Ao final de BP(5,7):



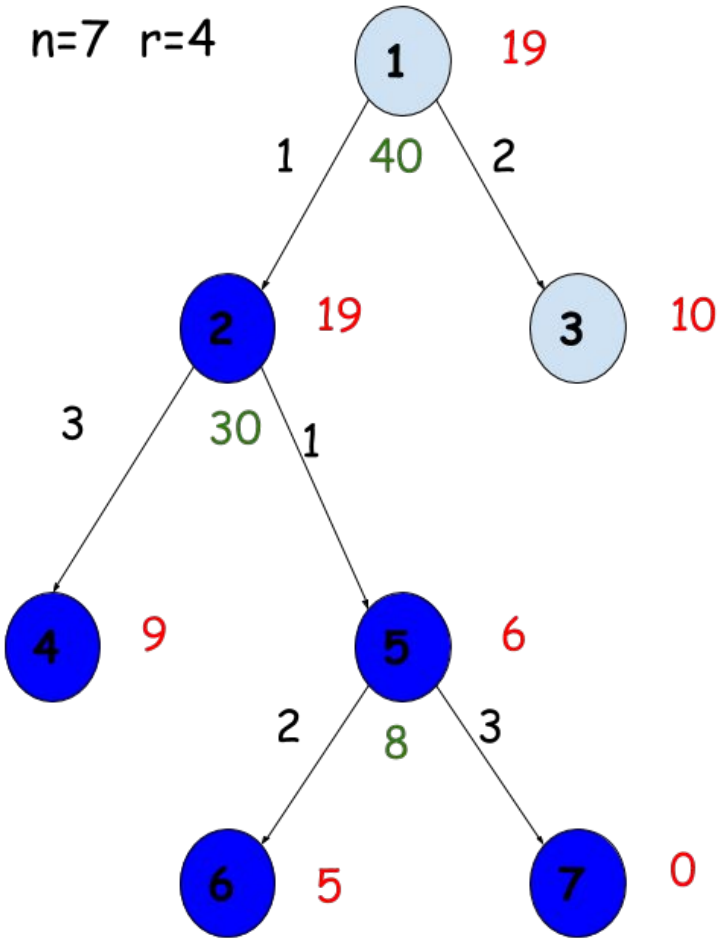
2666 - Imposto Real

Ao final de BP(2,5):



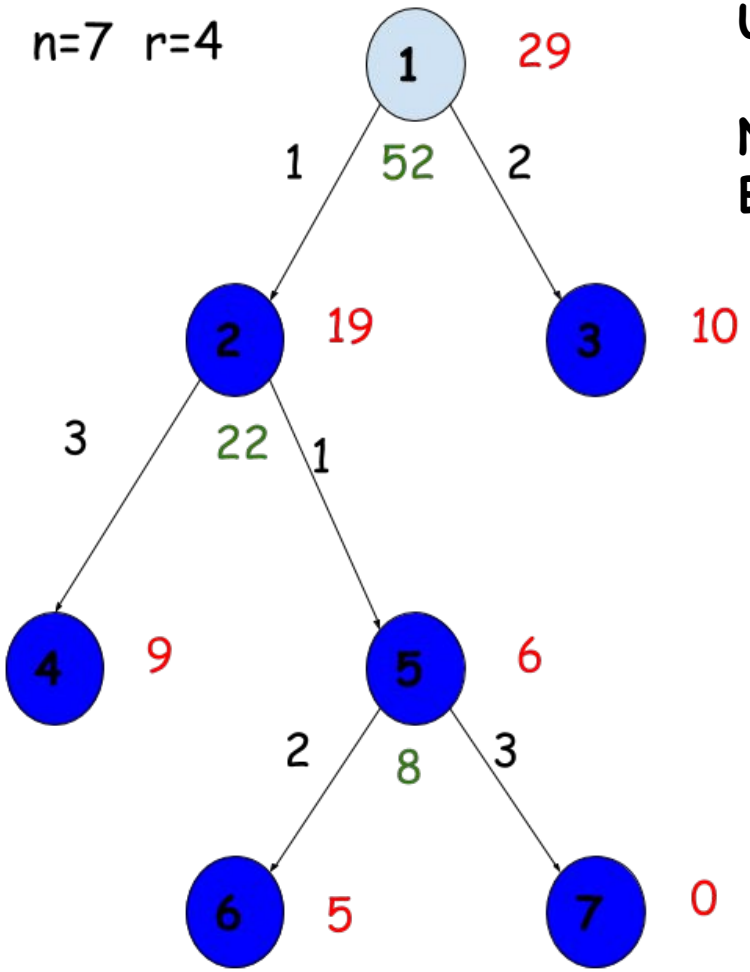
2666 - Imposto Real

Ao final de BP(1,2):



2666 - Imposto Real

Ao final de BP(1,3), já temos a resposta: 52.



Usar BP(u, v), u pai de v

Na chamada inicial: BP(0,1)

2962 - Arte Valiosa

Contexto: É dada uma sala de museu de dimensões M x N, onde existe uma porta em (0, 0) e um quadro valioso em (M, N). Foram instalados K detectores de movimentos em posições (x_i, y_i) dadas, cada um tendo um raio de ação igual a s_i. Um ladrão quer roubar o quadro valioso. Conseguirá fazer isso sem ser detectado?.

Entrada: Um único caso de teste descrito em K+1 linhas. Na primeira vêm os inteiros M, N, K ($10 \leq M, N \leq 10^4$, $1 \leq K \leq 10^3$). Em seguida K linhas com 3 inteiros, descrevendo sua posição x_i, y_i e seu raio de ação s_i ($0 < x_i < M$, $0 < y_i < N$, $0 < s_i \leq 10^4$).

Saída: Imprimir 'S' se for possível o roubo sem detecção ou 'N', caso contrário.

Exemplo de entrada:

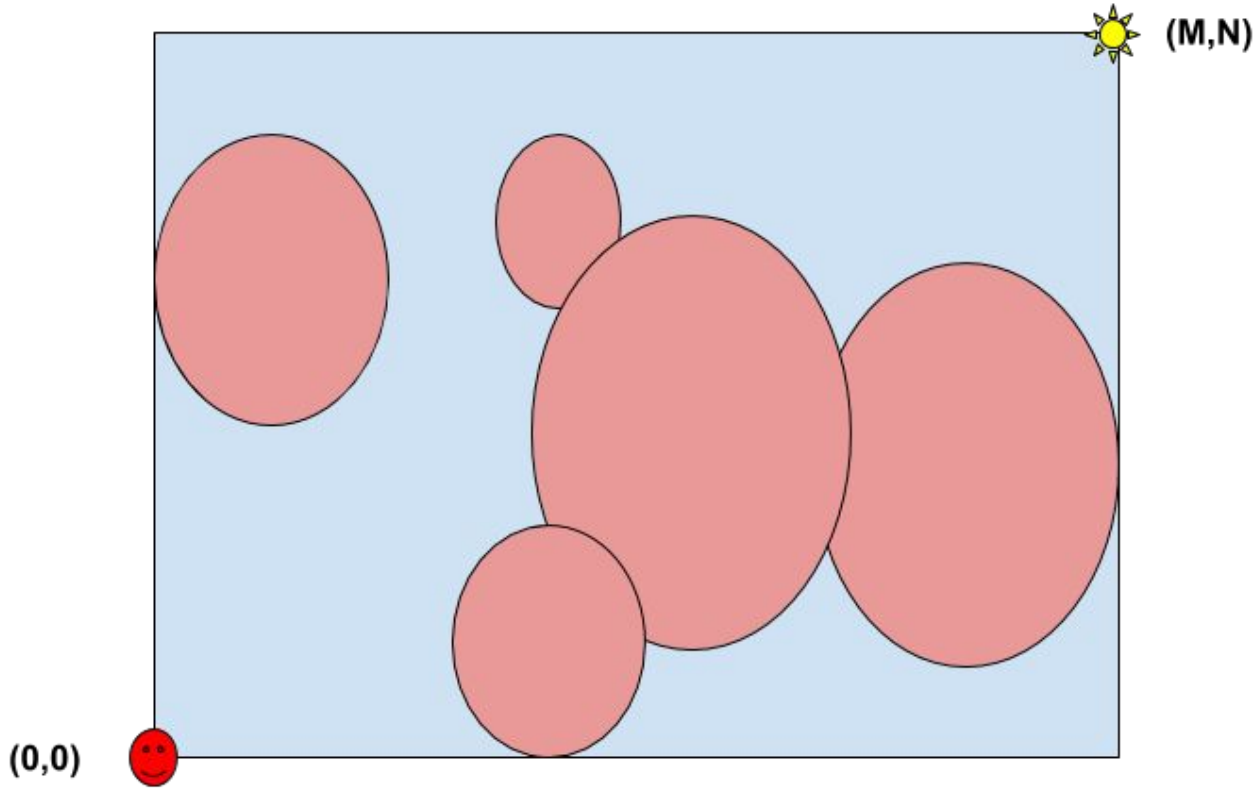
```
10 22 2
4 6 5
6  16  5
```

Exemplo de saída:

```
S
```

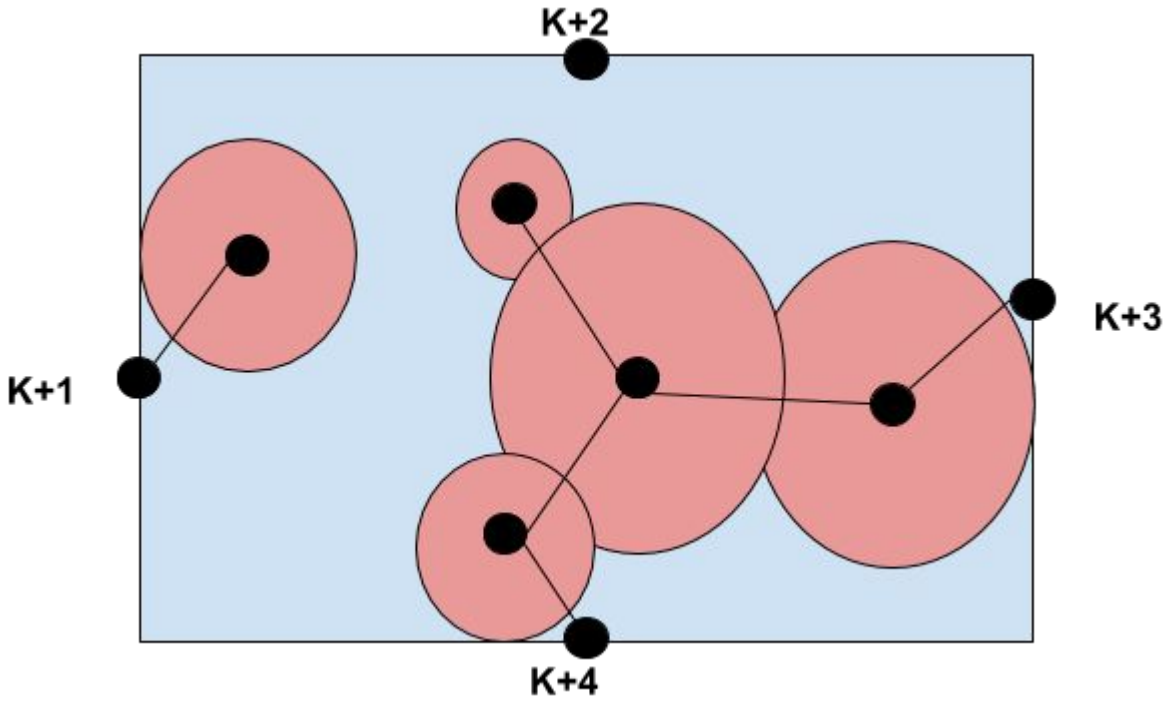
2962 - Arte Valiosa

Visualização



Para resolver o problema, a idéia é verificar se existe uma barreira que separe a porta do quadro valioso

2962 - Arte Valiosa - Modelagem



A modelagem consiste em criar um grafo contendo:

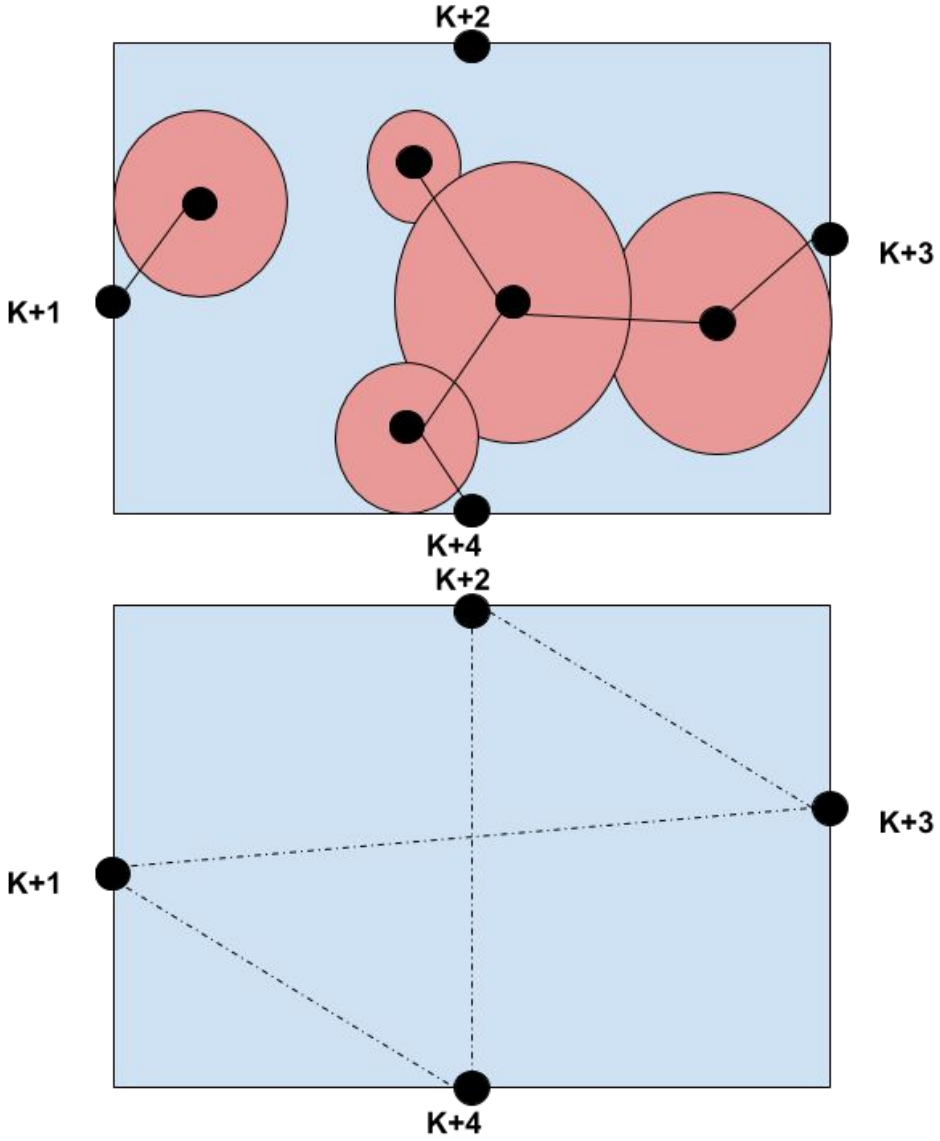
a) vértices 1 a K, um para cada detector

b) arestas entre vértices de dois detectores se seus círculos de ação se interceptarem em pelo menos em 1 ponto

c) vértices K+1 a K+4, um para cada parede da sala

d) arestas entre cada vértice de parede e vértice de detector se o círculo de ação desse detector interceptar a parede em pelo menos 1 ponto

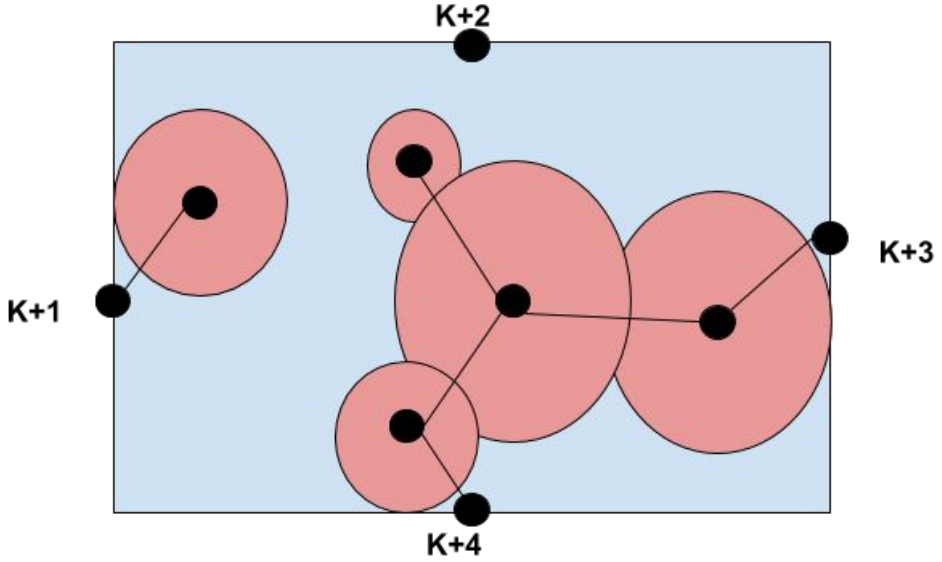
2962 - Arte Valiosa - Solução

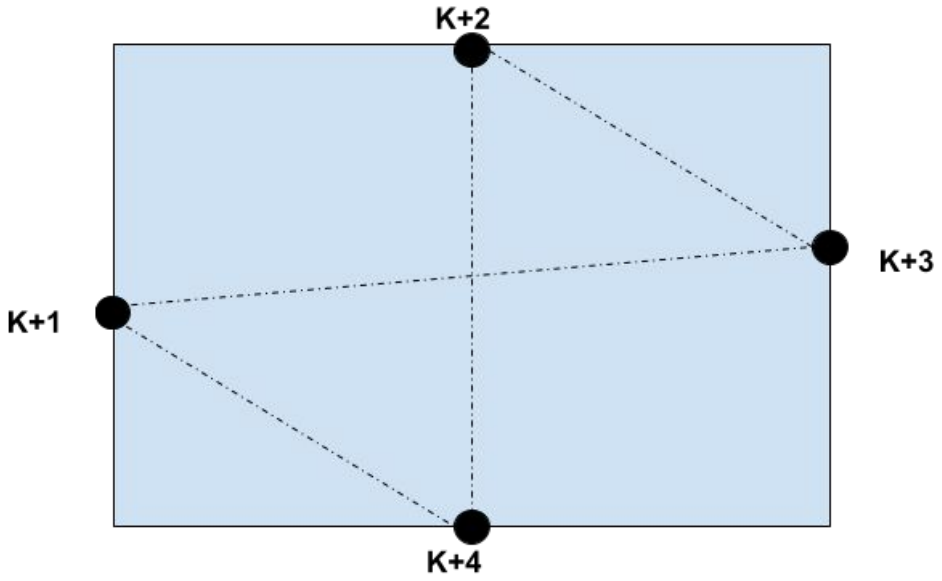


Haverá solução se houver uma das barreiras da figura inferior.

Em termos de algoritmo, a solução é
azer duas buscas em profundidade (ou largura) no grafo criado, cada uma das buscas começando nos vértices K+1 e K+2, respect. e verificar se, em alguma delas, um dos vértices K+3 ou K+4 é alcançado.

2962 - Arte Valiosa - Solução





Observação sobre a criação das arestas:

Existe uma aresta entre o os detectores K_i e K_j se a distância entre suas posições é $\leq s_i+s_j$, ou seja se

$$((x_i-x_j)^2 + (y_i-y_j)^2)^{\frac{1}{2}} \leq s_i+s_j$$

Para evitar problemas de precisão é preferível testar:

$$(x_i-x_j)^2 + (y_i-y_j)^2 \leq (s_i+s_j)^2$$

1442 - Desvio de Rua

Contexto: É dado um digrafo representando o trânsito de uma cidade. Um trecho de rua vai ser bloqueado. Quer-se saber como contornar o efeito do bloqueio, apenas invertendo o fluxo de algumas ruas ou tornando ruas de mão única em ruas de mão dupla, de forma a que se haja caminho entre quaisquer cruzamentos.

Entrada: Vários casos de teste, terminados por fim de arquivo. Cada teste vem em várias linhas. Na primeira, são informados N, M ($1 \leq N \leq 10^3$, $1 \leq M \leq 10^5$), o número de cruzamentos e trechos de rua, respect. A seguir vêm M linhas indicando os trechos de rua. Cada trecho é informado com 3 inteiros A, B ($1 \leq A, B \leq N$) indicando a ligação e T (1 ou 2), indicando o tipo de trânsito: 1 = mão única, 2=mão dupla. O primeiro trecho é o que vai ser bloqueado.

Saída: Para cada teste indicar o que fazer:
'-' nada precisa ser feito
'*' impossível
'1' inverter o sentido do trânsito de algumas ruas de mão única
'2' tornar alguns trechos de mão única em mão dupla.

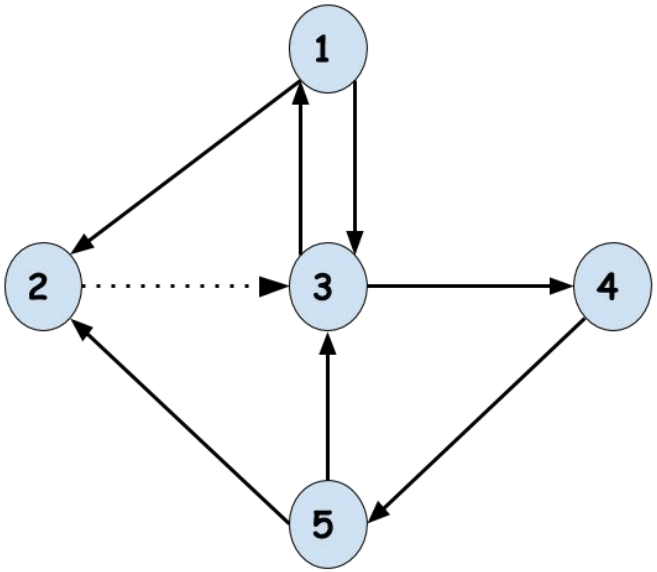
Exemplo de entrada:

```
5 7
2 3 1
1 3 2
1 2 1
3 4 1
4 5 1
5 2 1
5 3 1
```

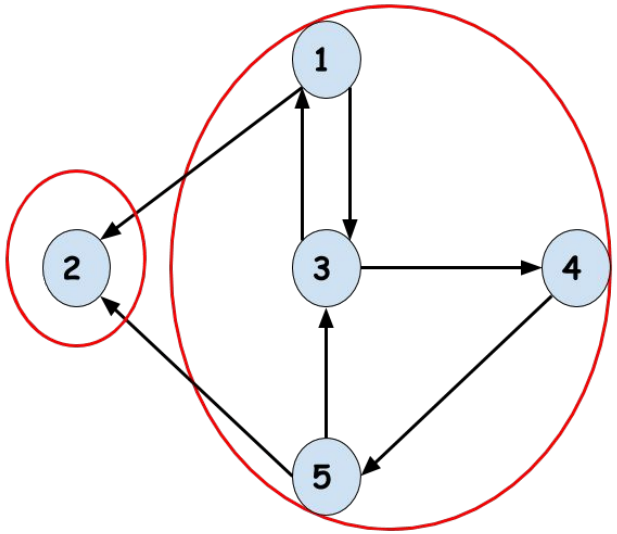
Exemplo de saída:

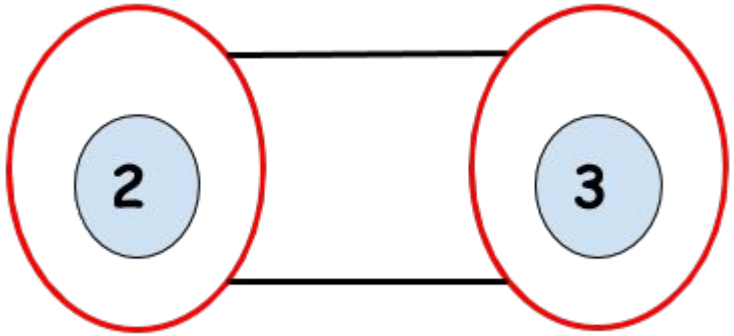
```
1
```

1442 - Desvio de Rua



Deve-se criar um grafo auxiliar que permite analisar certas estruturas do digrafo. O primeiro passo para criar esse grafo auxiliar é fazer a condensação dos componentes fortemente conexos (cfc).



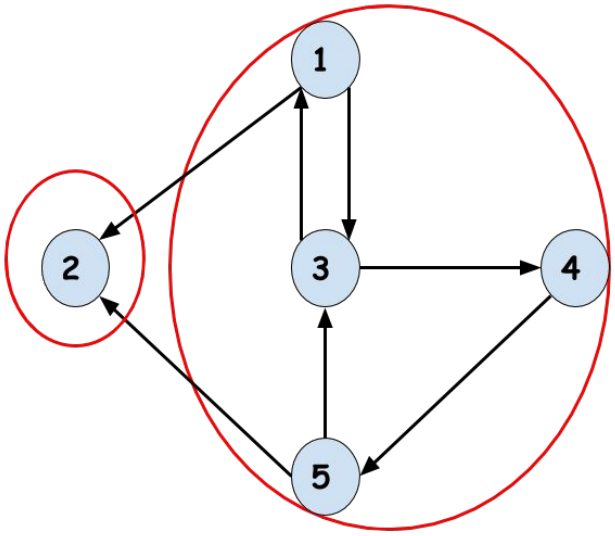


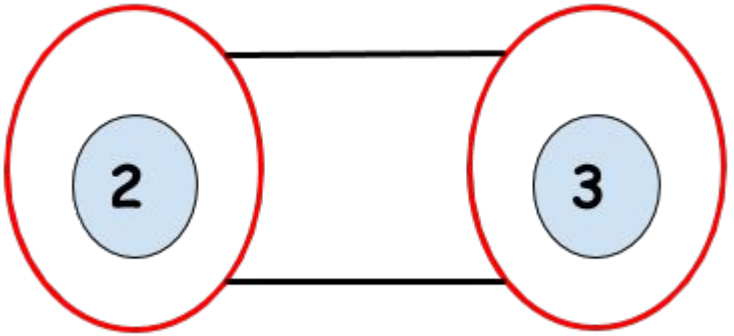
1442 - Desvio de Rua

Como criar o grafo auxiliar:

a) Escolher, para todos os vértices de um cfc um vértice representante que esteja no cfc

b) percorrer as arestas do grafo original:
   i) desprezar as arestas entre vértices de um mesmo componente
   ii) arestas entre vértices de componentes distintos são transformadas para arestas entre representantes
   iii) manter arestas paralelas.

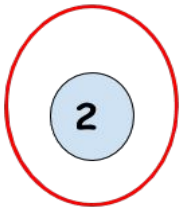




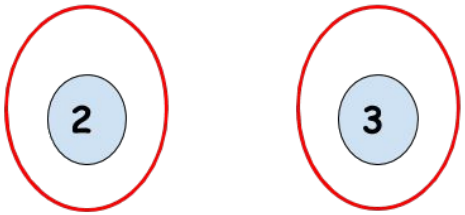
1442 - Desvio de Rua

Análise do grafo auxiliar

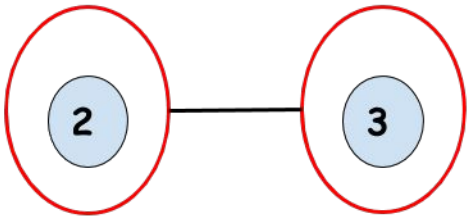
Se o grafo auxiliar tiver só um vértice, nada precisa ser feito.



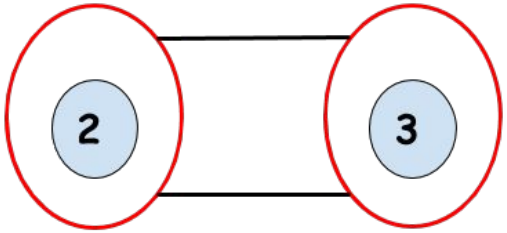
Se o grafo auxiliar for desconexo: não tem solução.



Se o grafo auxiliar tiver ponte (sem arestas paralelas) alguma rua de mão única deve mudar para mão dupla.



Se o grafo auxiliar tiver ponte relativa a arestas paralelas alguma rua de mão única deve ter o sentido invertido.



1442 - Desvio de Rua

Algoritmos envolvidos

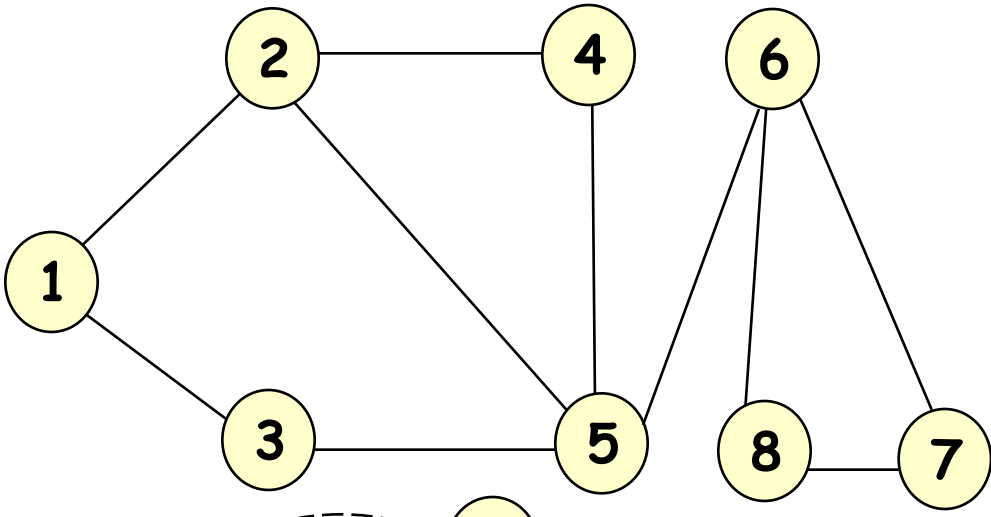
a) Determinação dos cfcs: Algoritmo de Tarjan com determinação de representantes de cada cfc. $O(n^2)$ ou $O(n+m)$,

b) Criação do grafo auxiliar: percorrer o digrafo ignorando arestas do mesmo cfc e, para arestas entre cfcs distintos, criar aresta entre os representantes dos cfcs. $O(n^2)$ ou $O(n+m)$

c) Determinação de pontes: algoritmo análogo ao de determinação de pontos de articulação. $O(n^2)$ ou $O(n+m)$

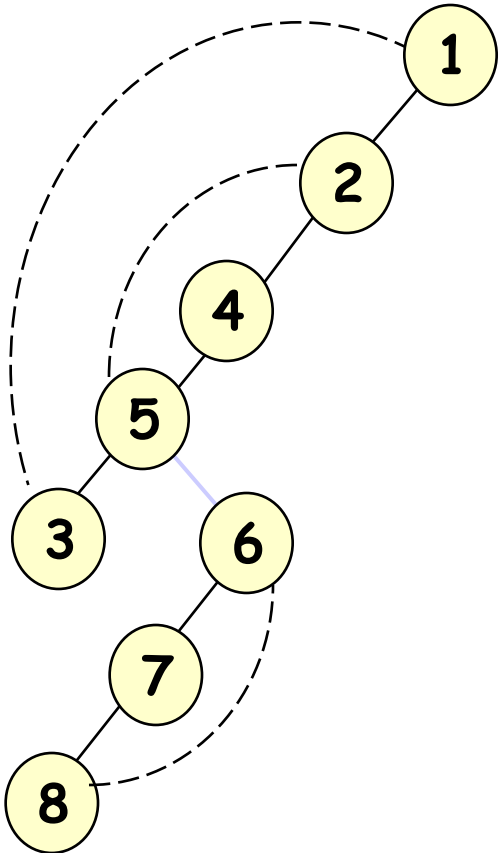
Grafos – Pontes

Ponte: Aresta que, se removida, desconecta o grafo: Ex (5,6)



Idéia de um algoritmo:

Dada uma árvore de profundidade, a aresta (v,w) será ponte quando w é filho de v e não há descendentes de w (incluindo w) com arestas de retorno para ancestrais de w.



1391 - Quase o Menor Caminho

Contexto: É dado um digrafo contendo a descrição do mapa de trânsito de uma região: as rotas de trânsito, todas de mão única e com seus tamanhos. Como muitos motoristas usam o GPS para utilizar o caminho mínimo, um motorista quer procurar um caminho alternativo bom para a hora de "rush" entre os pontos **s** e **t** que não passe por nenhuma via que possa estar em caminhos mínimos entre esses pontos.

Entrada: Vários casos de teste. Para cada caso de teste é informado **n**, **m**, **s**, **t** e as **m** interligações, em termos de 3 inteiros (origem, destino, **d** =distância). ($2 \leq n \leq 500$, $1 \leq m \leq 10000$, $0 \leq d \leq 1000$).

Saída: Para cada teste deve ser impresso a distância do caminho alternativo de distância mínima. Se não for possível imprimir -1.

Exemplo de entrada:

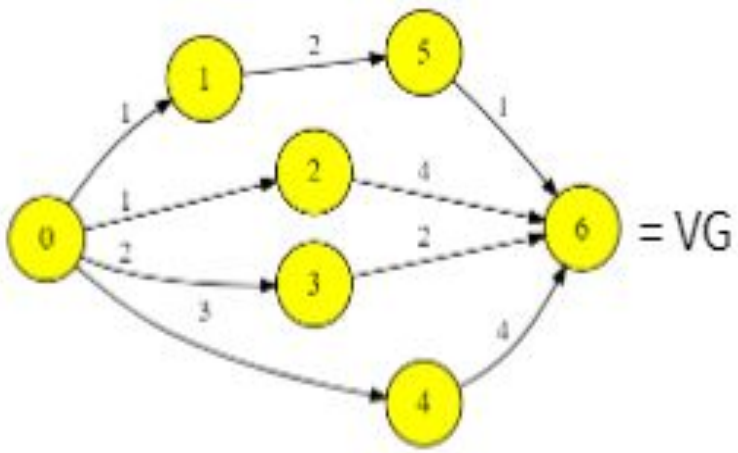
```
7 9 0 6
0 1 1    0 2 1    0 3 2
0 4 3    1 5 2    2 6 4
3 6 2    4 6 4    3 6 1
```

Exemplo de saída:

```
5
```

1391 - Quase o Menor Caminho

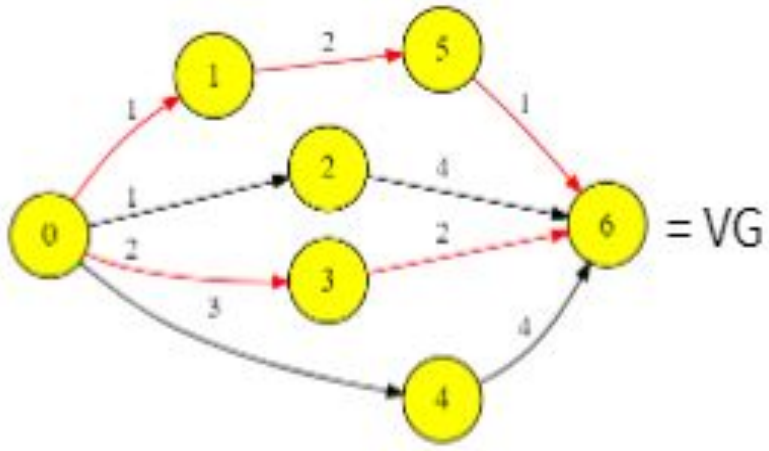
O exemplo anterior corresponde ao seguinte grafo:



Distâncias mínimas de 0:

0	1	2	3	4	5	6
0	**1**	**1**	**2**	**3**	**3**	**4**

A solução consiste em procurar o caminho mínimo usando só as arestas pretas:

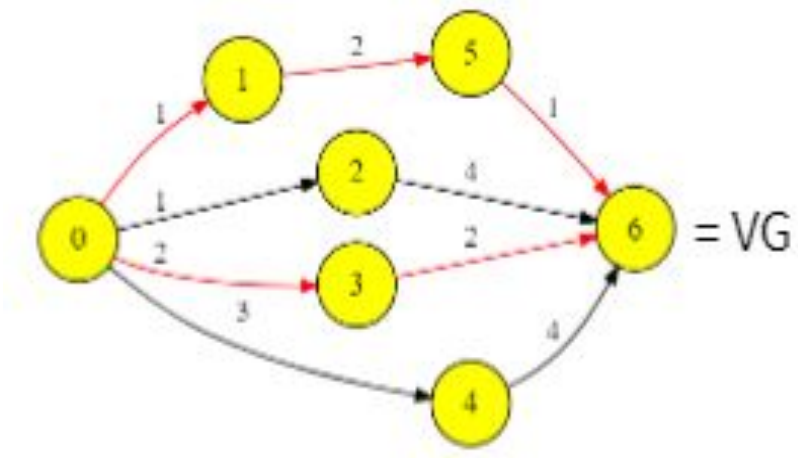


Distâncias mínimas de 0 (sem arestas vermelhas):

0	1	2	3	4	5	6
0	**∞**	**1**	**∞**	**3**	**∞**	**5**

1391 - Quase o Menor Caminho

P: Como encontrar as arestas que podem estar em caminhos mínimos e que devem ser retiradas do grafo?



Distâncias mínimas de 0:

0	1	2	3	4	5	6
0	**1**	**1**	**2**	**3**	**3**	**4**

Distâncias mínimas a 6:

0	1	2	3	4	5	6
4	**3**	**4**	**2**	**4**	**1**	**0**

R: Analisando se cada aresta E[u,v] satisfaz a:

D[t] = D[u] + E[u,v] + D'[v], onde:

D[v] = distância mínima do vértice s ao vértice v

D'[v] = distância mínima do vértice v ao vértice t.

Se E[u,v] satisfaz, deve ser tirada do grafo.

D[v]: pode ser calculado usando o algoritmo de Dijkstra.

D'[v]: pode ser calculado usando o algoritmo de Dijkstra?

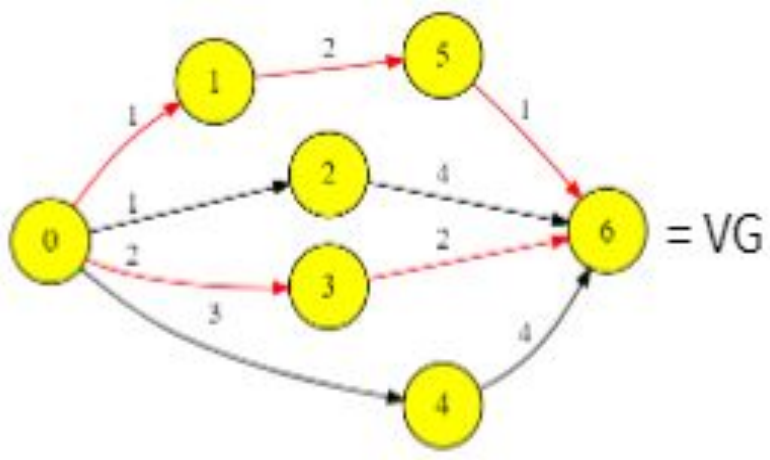
1391 - Quase o Menor Caminho

P: Como encontrar as distâncias mínimas a um vértice t, em um digrafo?

R: Com uma pequena mudança no algoritmo de Dijkstra, considerando, ao invés de arestas de saída de vértices, arestas de entrada.

R: Se o grafo estiver representado com Matriz de Adjacências, essa modificação é muito simples, bastando que a vizinhança de entrada seja procurada na coluna, ao invés da linha.

R: Se o grafo estiver representado com Listas de Adjacências, deve-se representar, para cada vértice as arestas de saída e de entrada.



Distâncias mínimas a 6:

0	1	2	3	4	5	6
4	3	4	2	4	1	0

1391 - Quase o Menor Caminho

Conclusão: para a solução deste problema deve-se:

a) rodar Dijkstra para determinar as distâncias mínimas de **s** aos demais vértices

b) rodar Dijkstra para determinar as distâncias mínimas de todos vértices a **t**

c) fazer uma busca no grafo, criando um grafo auxiliar sem as arestas de caminhos mínimos.

d) rodar Dijkstra no grafo auxiliar para determinar as distâncias mínimas de **s** aos demais vértices.

2882 - Gasolina

Contexto: No fim de uma greve r refinarias devem abastecer rapidamente p postos. São dados os estoques das refinarias, as demandas dos postos, quais refinarias podem atender quais postos e o tempo de atendimento de cada refinaria ao posto. Quer-se saber qual o tempo mínimo para todos os postos estarem abastecidos.

Entrada: Cada tese inicia c/ 3 inteiros numa linha: p, r ($1 \leq p, r \leq 1000$), número de postos e refinarias e np ($1 \leq np \leq 20000$), o número de pares refinaria-posto. Na próxima linha p inteiros, as demandas dos postos; na terceira linha r inteiros, os estoques das refinarias. Nas próximas np linhas, 3 inteiros I, J, T ($1 \leq T \leq 10^6$) número do posto, número da refinaria e tempo de atendimento.

Saída: Para cada teste deve ser impressa o tempo mínimo de atendimento a todos os postos; -1 se não for possível.

Exemplo de entrada:

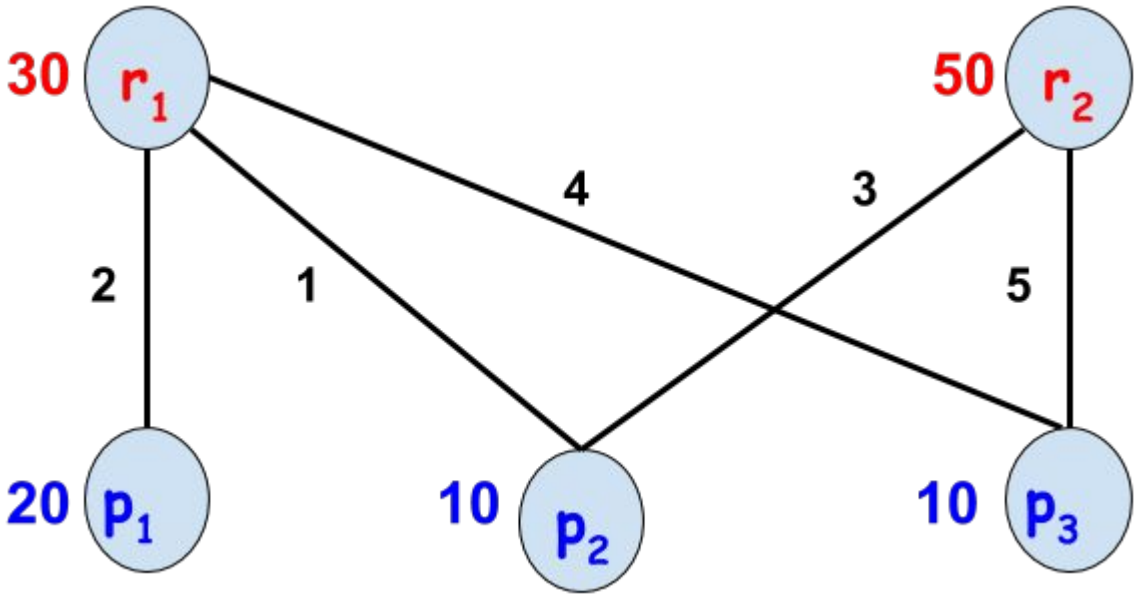
```
3   2   5
20  10  10
30  20
1   1   2
2   1   1
2   2   3
3   1   4
3   2   5
```

Exemplo de saída:

```
4
```

2882 - Gasolina

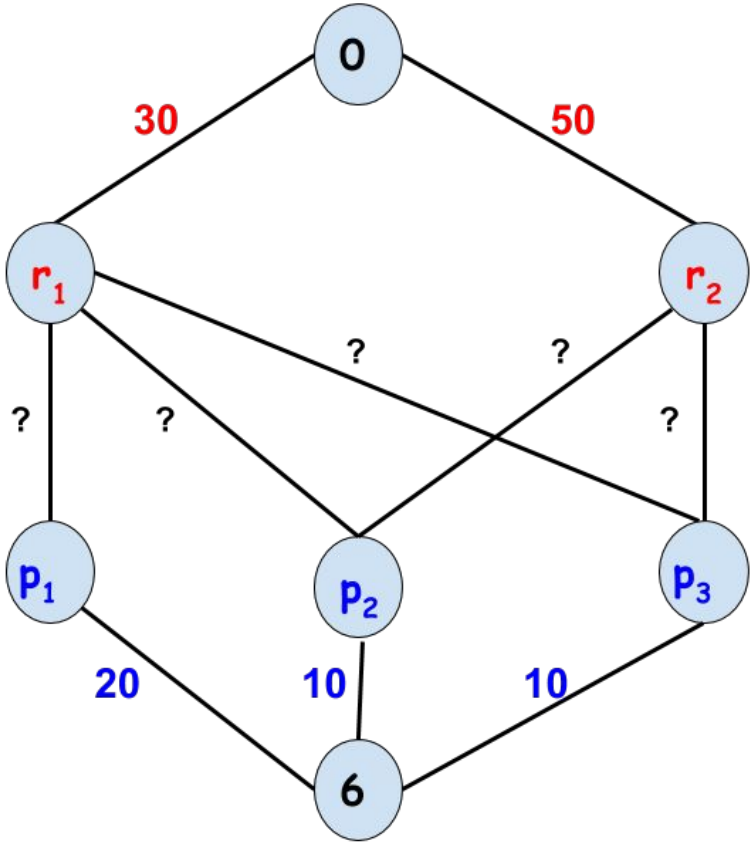
Modelagem do problema:



Uma idéia natural é modelar em termos de fluxo em redes.
Pode-se imaginar a gasolina fluindo das refinarias para os postos.
Então, os tempos devem ser aspectos secundários das arestas.

2882 - Gasolina

Modelagem do problema:

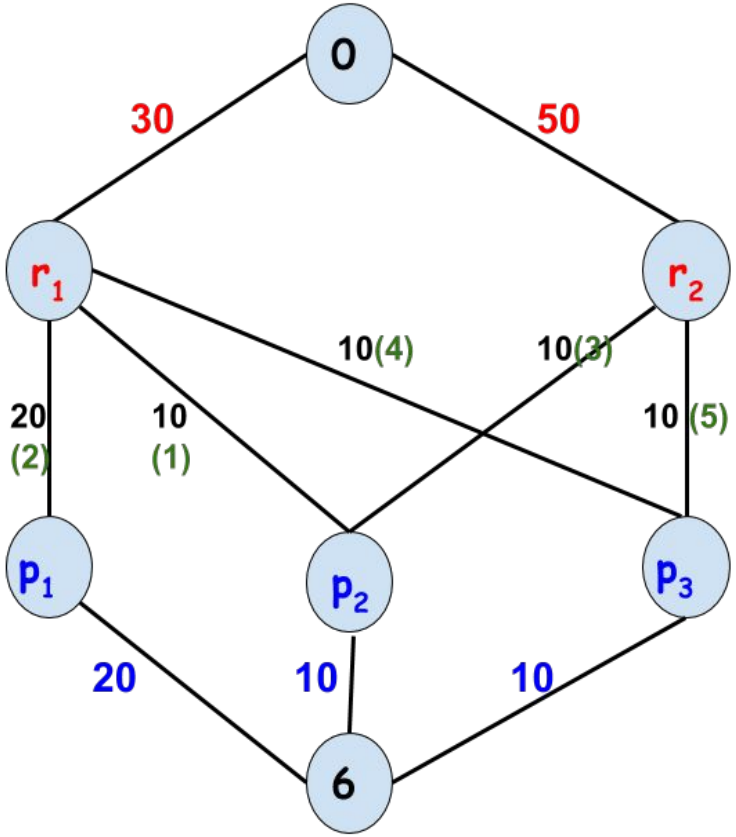


Adiciona-se uma fonte (0) e um sumidouro (6). A fonte liga-se às refinarias com capacidade = estoque de cada refinaria. Os postos ligam-se ao sumidouro com capacidade = demanda de cada posto.

2882 - Gasolina

Modelagem do problema:

Cada refinaria r_i deve ser ligada com o posto p_j que atende através de uma aresta com capacidade = min($e(r_i)$, $d(p_j)$), onde $e(r_i)$ = estoque da refinaria ri e $d(p_j)$ = demanda do posto p_j.



P: Após obter o fluxo máximo, como saber se o problema foi resolvido?

R: Verificando se ele é igual à soma das demandas dos postos.

P: Após obter o fluxo máximo, como saber o tempo mínimo?

R: Verificando na rede a aresta de maior tempo de atendimento.

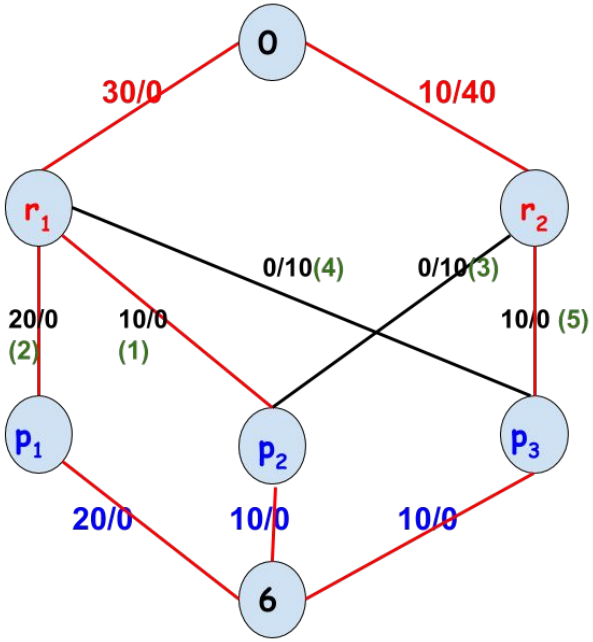
2882 - Gasolina - Modelagem p/ tempo mínimo

Para a solução é preciso descobrir qual o tempo mínimo possível.

Isso pode ser feito por **tentativa**, construindo a rede apenas com arestas cujo tempo de atendimento seja ≤ **tentativa**. A idéia é implementada com eficiência por **Pesquisa Binária na Solução**:

```
PBS():
    Ordenam-se as t arestas (refinaria/posto) pelo tempo de atendimento
    tmax ← t;   tmin ← 1;   dm ← ∑ demandas
    ConstroiG(tmax)
    fm ←  FluxoMax(0, s)
    se (fm < dm):
        não tem solução
    senão:
        enquanto (tmax > tmin):
            tmed ←  (tmax+tmin)/2
            ConstroiG(tmed)
            fm ←  FluxoMax(0, s)
            se (fm < dm):
                tmin ←  tmed+1
            senão:
                tmax ←  tmed
```

2882 - Gasolina - Solução do exemplo

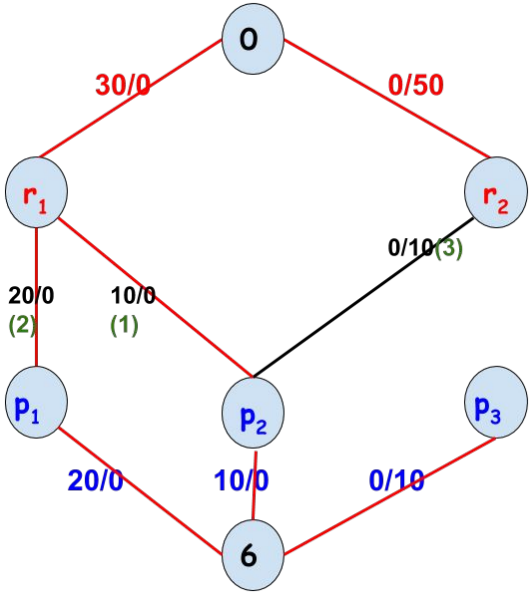


tmax = 5 tmin = 1

ConstroiG(5)

fmax = 40 = dem

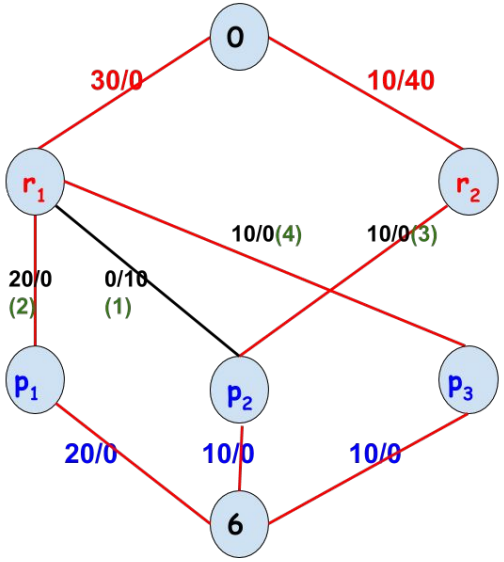
(tem solução)



tmax = 5 timn = 1
tmed = 3

ConstroiG(3)

fmax = 30 < dem
(não soluciona)
tmin = 4



tmax = 5 timn = 4
tmed = 4

ConstroiG(4)

fmax = 40 = dem
(soluciona)
tmax = 4 = tmin

A solução é o tempo de atendimento da aresta 4 = 4

2882 - Gasolina

Complexidade de tempo

O problema tem limite bastante apertado para uso de algoritmos de fluxo em redes, além do que o fluxo pode ter que ser rodado, no pior caso, aproximadamente $\log_2 20000 \cong 15$ vezes. Só passa nos juizes se for usado um algoritmo bem eficiente tal como o de rede de camadas (Dinic), cuja complexidade é $O(V^2E)$.

1476 - Caminhão

Contexto: Uma cidade é feita de ilhas ligadas por pontes, cada uma com limite máximo de peso dado. Uma empresa tem várias sedes em ilhas dadas e fábricas em ilhas também dadas. São dados vários pares (sede, fábrica) e quer-se saber para cada um desses pares qual o máximo peso que um caminhão pode levar da fábrica para a sede.

Entrada: Vários casos de teste, terminados por fim de arquivo. Cada teste vem em várias linhas. Na primeira, são informados N,($1 \leq N \leq 2.10^4$), M, ($1 \leq M \leq 10^5$) e S ($1 \leq S \leq 5.10^4$), o número de ilhas, pontes e consultas, respect. A seguir vêm M linhas indicando as pontes. Cada ponte é informada com 3 inteiros A, B ($1 \leq A, B \leq N$) indicando a ligação e C ($\leq 10^5$), o limite de peso. A seguir vêm S consultas, sendo cada consulta um par de ilhas.

Saída: Para cada consulta indicar o peso máximo que pode ser transportado por um caminhão entre a fábrica e a sede.

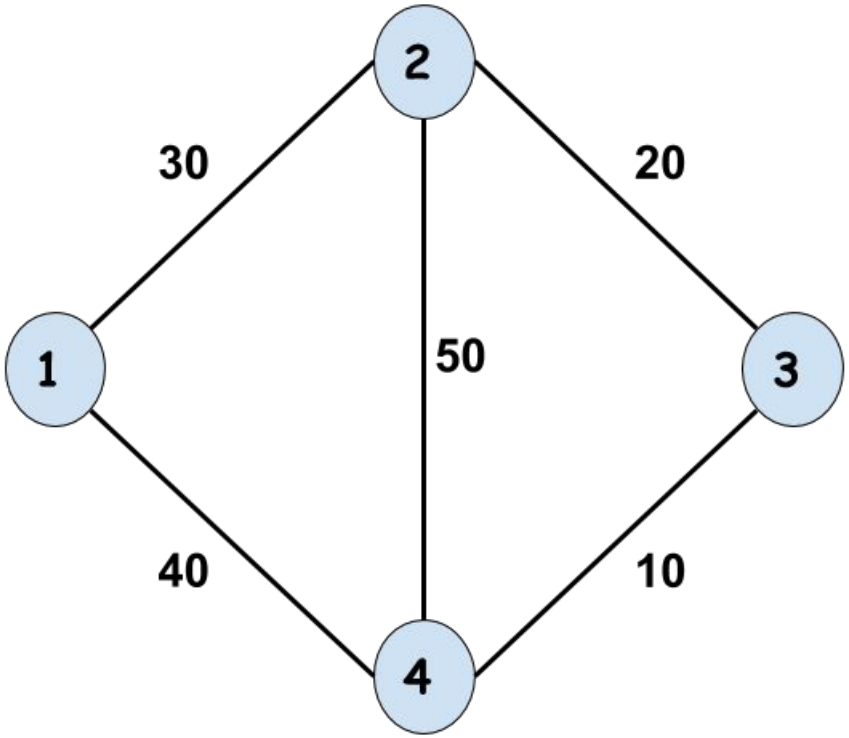
Exemplo de entrada:

```
4 5 3
1 2 30
1 4 40
2 3 20
2 4 50
3 4 10
1 3
1 4
1 2
```

Exemplo de saída:

```
20
40
40
```

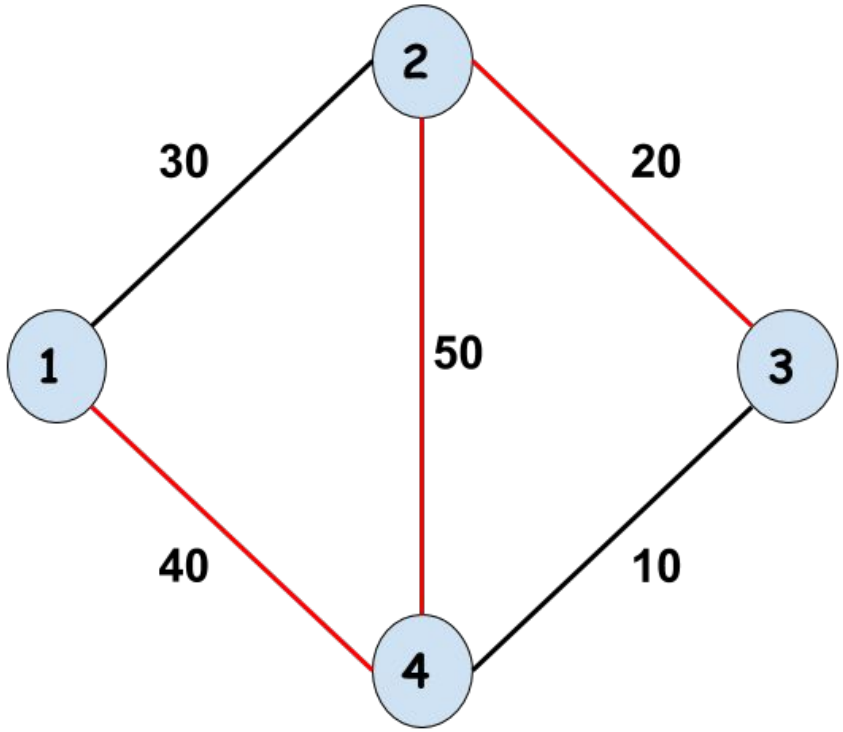
1476 - Caminhão



O máximo para o par (1, 3) é 20.

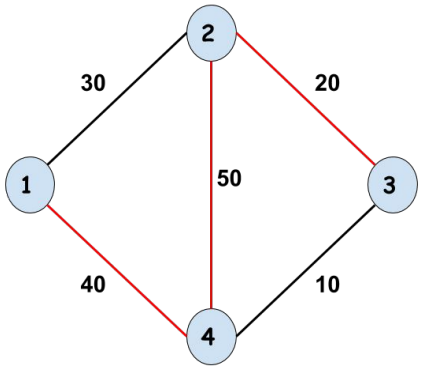
Para o par (1, 2) é 40.

1476 - Caminhão



Dado um par (s, f), a solução consiste em encontrar a menor aresta do caminho de s a f na Árvore Geradora Máxima (AGM).

1476 - Caminhão



Solução 1:
Criar a *AGM* e fazer uma Busca em Profundidade para o par (s, f).

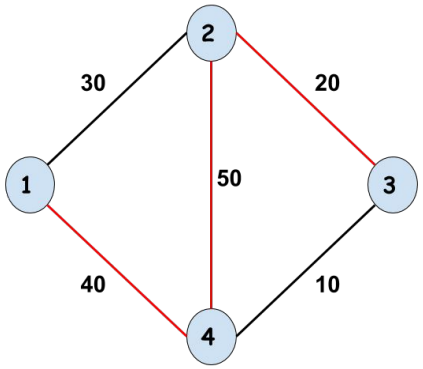
A Solução 1 não serve no contexto da maratona porque o tempo disponível é muito baixo. Cada resposta a uma consulta tem que ser inferior a O(n)

Solução 2:
Usar a técnica de *LCA* (Primeiro Ancestral Comum) na *AGM*, com complexidade O(log n) para cada consulta.

https://www.topcoder.com/community/competitive-programming/tutorials/range-minimum-query-and-lowest-common-ancestor/

1476 - Caminhão

Solução 3:
Prepara resposta às consultas durante a criação da **AGM** com o algoritmo de Kruskal, que usa a técnica Union-Find.

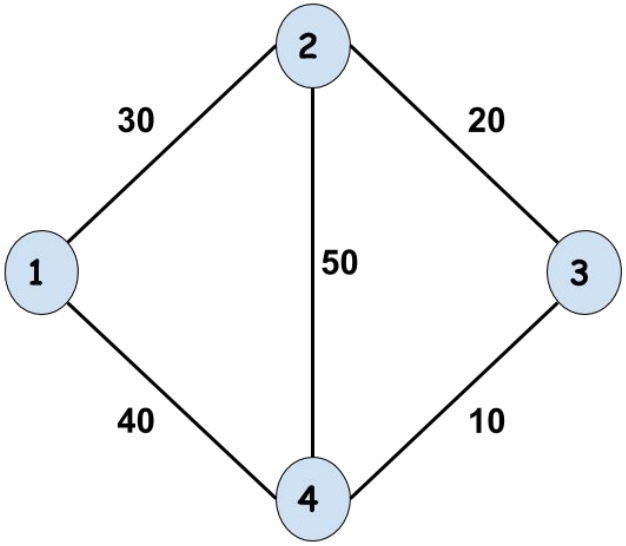


Kruskal:

a) **Inicialmente criar um subconjunto para cada vértice**
b) **Ordenar as arestas de forma não crescente.**
c) **Examinar as arestas da ordenação verificando se cada uma delas pode entrar na AGM.**
d) **Se puder: juntam-se, por Union-Find, os subconjuntos onde estão os vértices dessa aresta.**

a) No início, para cada vértice, cria-se uma lista das consultas de que participa.
b) Quando uma nova aresta é inserida na AGM, a lista de consultas do subconjunto menor é examinada.
c) Se a consulta envolver um vértice de cada um dos subconjuntos (componentes) que vão ser fundidos, a consulta é resolvida e sai da lista. Se não, ela entra na lista de consultas não resolvidas do subconjunto maior.

1476 - Caminhão



Inicialmente:

Arestas ordenadas:
(2, 4, 50) (1, 4, 40) (1, 2, 30), (2,3,20) (3,4,10)

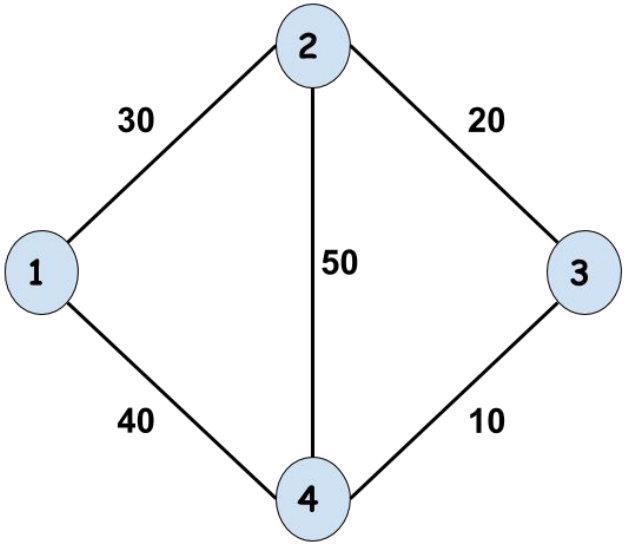
Lista de consultas:

1	1	2	3
2	3		
3	1		
4	2		

Consultas:

1 3
1 4
1 2

1476 - Caminhão



Arestas ordenadas:
(2, 4, 50) (1, 4, 40) (1, 2, 30), (2,3,20) (3,4,10)

Aresta (2, 4, 50) é aceita na AGM, o subconjunto {4} junta-se ao {2], formando {2, 4} e nenhuma consulta da lista 4 é resolvida

Lista de consultas:

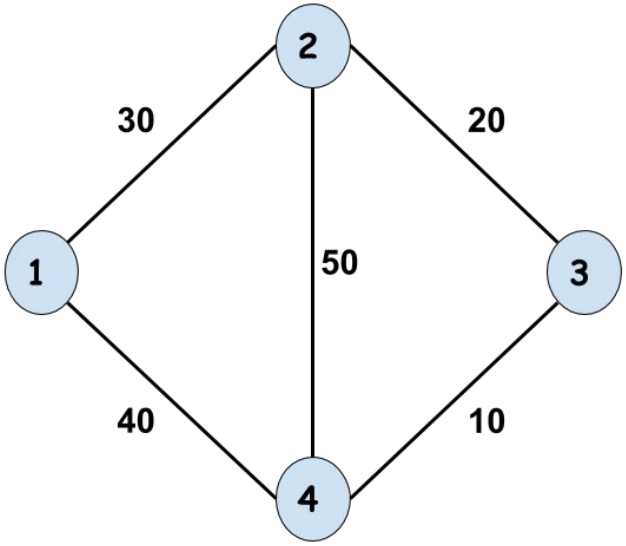
Consultas:

1 3
1 4
1 2

{1}	1	2	3
{2}	3		
{3}	1		
{4}	2		

{1}	1	2	3
{2,4}	3	2	
{3}	1		

1476 - Caminhão



Arestas ordenadas:
(2, 4, 50) (1, 4, 40) (1, 2, 30), (2,3,20) (3,4,10)

Aresta (1, 4, 40) é aceita na AGM, o subconjunto {1} junta-se ao {2,4}, formando {1, 2, 4}. As consultas do subconjunto {2,4} são resolvidas:
Consulta 2: (1,4) = 40
Consulta 3: (1,2) = 40

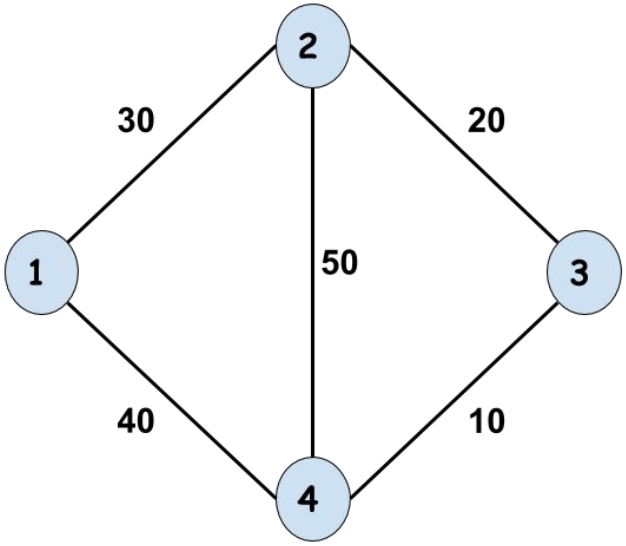
Consultas:
1 3
1 4
1 2

Lista de consultas:

{1}	1	2	3
{2,4}	3	2	
{3}	1		

{1, 2, 4}	1	2	3
{3}	1		

1476 - Caminhão



Arestas ordenadas:
(2, 4, 50) (1, 4, 40) (1, 2, 30), (2,3,20) (3,4,10)

Aresta (1, 2, 30) é rejeitada.

Consultas:

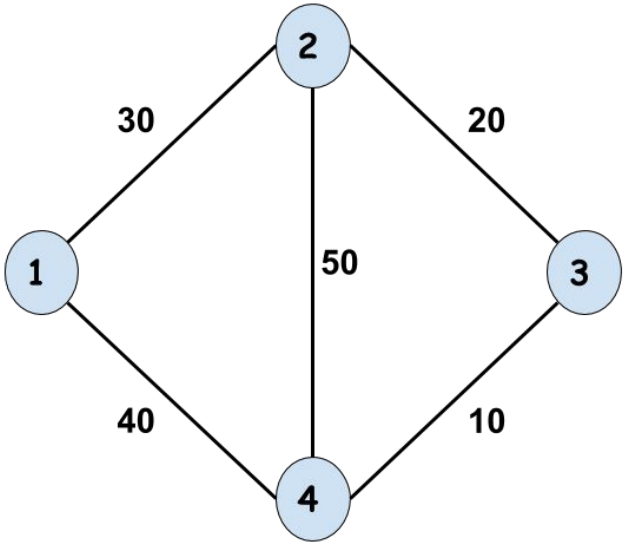
1 3
1 4
1 2

Lista de consultas:

{1}	1	2	3
{2,4}	3	2	
{3}	1		

{1, 2, 4}	1	2	3
{3}	1		

1476 - Caminhão



Arestas ordenadas:
(2, 4, 50) (1, 4, 40) (1, 2, 30), (2,3,20) (3,4,10)

Aresta (2, 3, 20) é aceita na AGM, o subconjunto {3} junta-se ao {1,2,4}, formando {1, 2, 3, 4}. A consulta do subconjunto {3} é resolvida:
Consulta 1: (1,3) = 20

Consultas:

1 3
1 4
1 2

Lista de consultas:

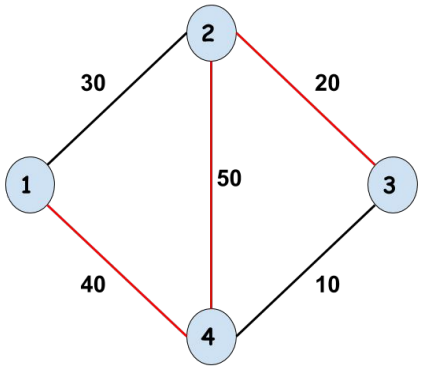
{1,2,4}	1	2	3
{3}	1		

{1,2,3,4}	1	2	3

1476 - Caminhão

Solução 3:

A junção do Union-Find deve ser feita pelo tamanho das listas dos subconjuntos a serem agrupados. O de maior lista passa a ser o pai.



Cada lista de consultas tem tamanho máximo igual a 2S. Quando uma lista de consultas é transferida, ela vai para uma lista que tem, no mínimo seu tamanho. Logo, o máximo de transferências feitas é $\log_2 2S$. Portanto, a manipulação das listas é feita em $O(S \log S)$. O algoritmo como um todo tem complexidade $O(M \log M + S \log S)$.

1490 - Torres que Atacam

Contexto: O problema das Torres Pacíficas consiste em colocar n torres em um tabuleiro n x n, de tal forma que não se ataquem. Nesta variante, existem peões no tabuleiro, de tal forma que eles podem bloquear ataques. Dado um tabuleiro n x n, com alguns peões posicionados, qual o máximo de torres que não se atacam podem ser colocadas?

Entrada: Cada tese começa com o valor n ($1 \leq n \leq 100$). Em seguida vêm n linhas, descrevendo um tabuleiro, onde 'X' indica um peão posicionado e '.' uma posição livre. Os testes terminam por fim de arquivo.

Saída: Para cada teste deve ser impressa a quantidade de torres que podem ser colocadas no tabuleiro, de forma que não se ataquem.

Exemplo de entrada:

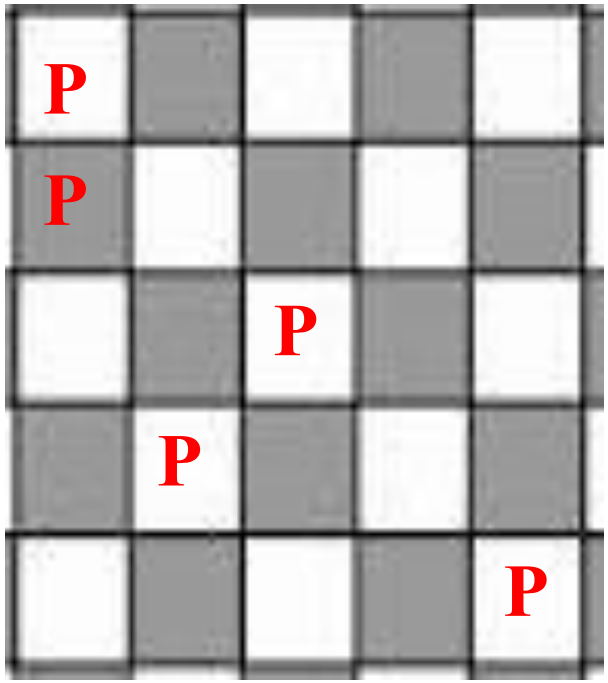
```
5
X....
X....
..X..
.X...
....X
```

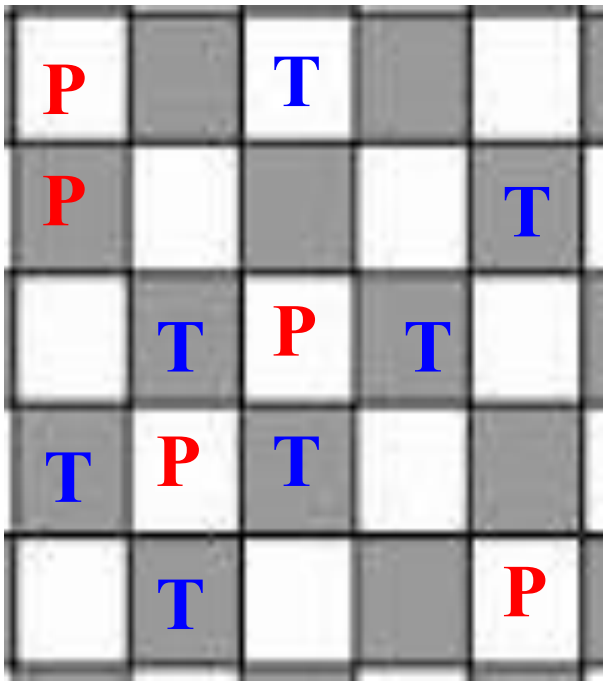
Exemplo de saída:

```
5
```

1490 - Torres que Atacam

Solução do exemplo:



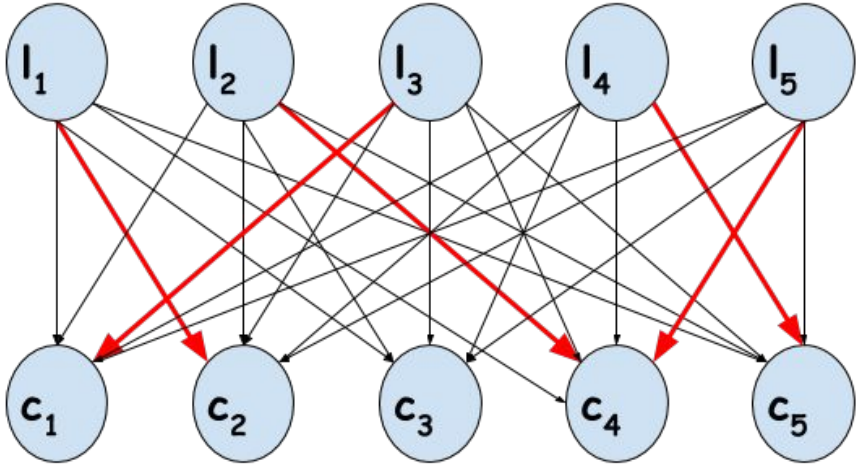


A idéia para a solução é tentar colocar uma torre em cada sequência de casas livres.

1490 - Torres que Atacam

O problema de Torres que que não se atacam (Torres Pacíficas) em um tabuleiro n x n tem n! soluções distintas quando não existem peões, pois cada permutação de 1 a n é solução válida.

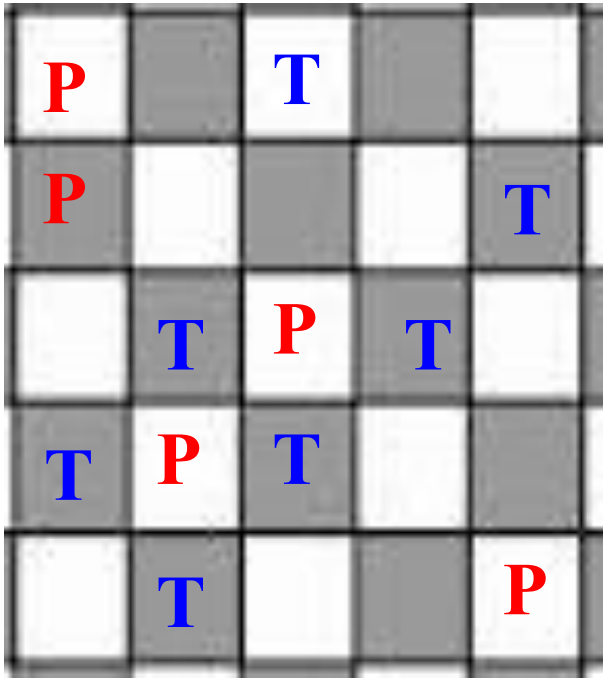
Tem também uma solução baseada em emparelhamento em grafos bipartidos, pois cada emparelhamento máximo também é uma solução.

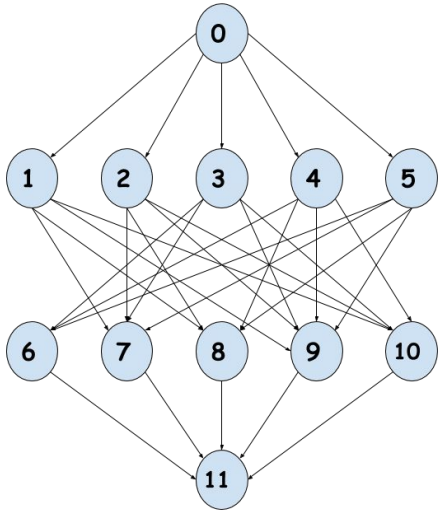


P: A idéia de emparelhamento máximo em grafo bipartido pode ser estendida para o problema atual?

1490 - Torres que Atacam

Solução do exemplo:





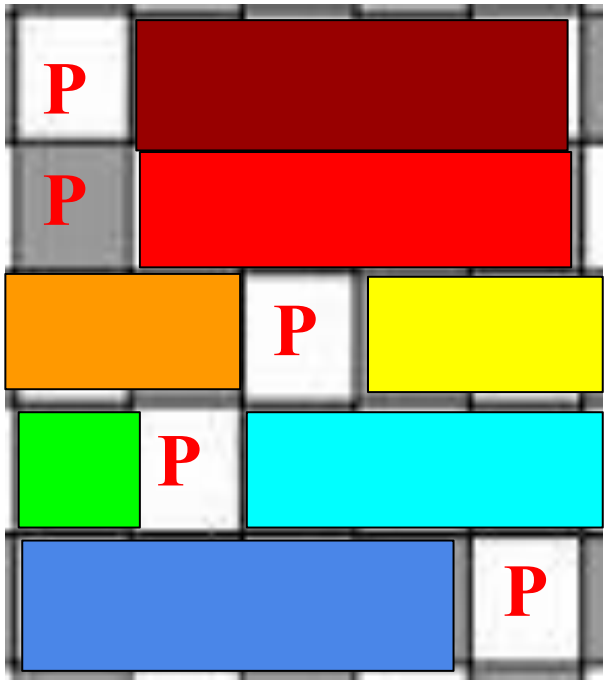
P: A idéia de emparelhamento máximo em grafo bipartido pode ser estendida para o problema atual?

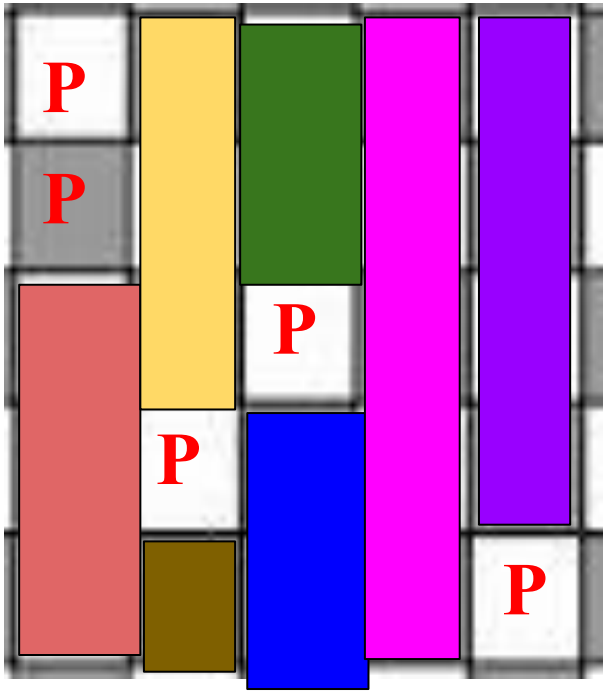
R: Sim, criando o grafo adequado tal que se procure emparelhar uma torre em cada sequência de casas livres.

1490 - Torres que Atacam

Solução do exemplo:

Inicialmente, obter as sequências de casas livres em linhas e colunas.

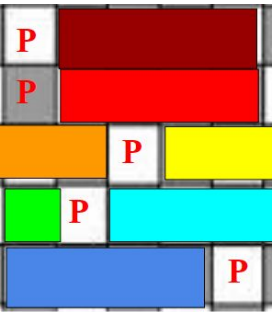


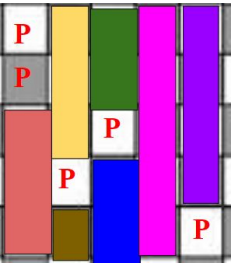


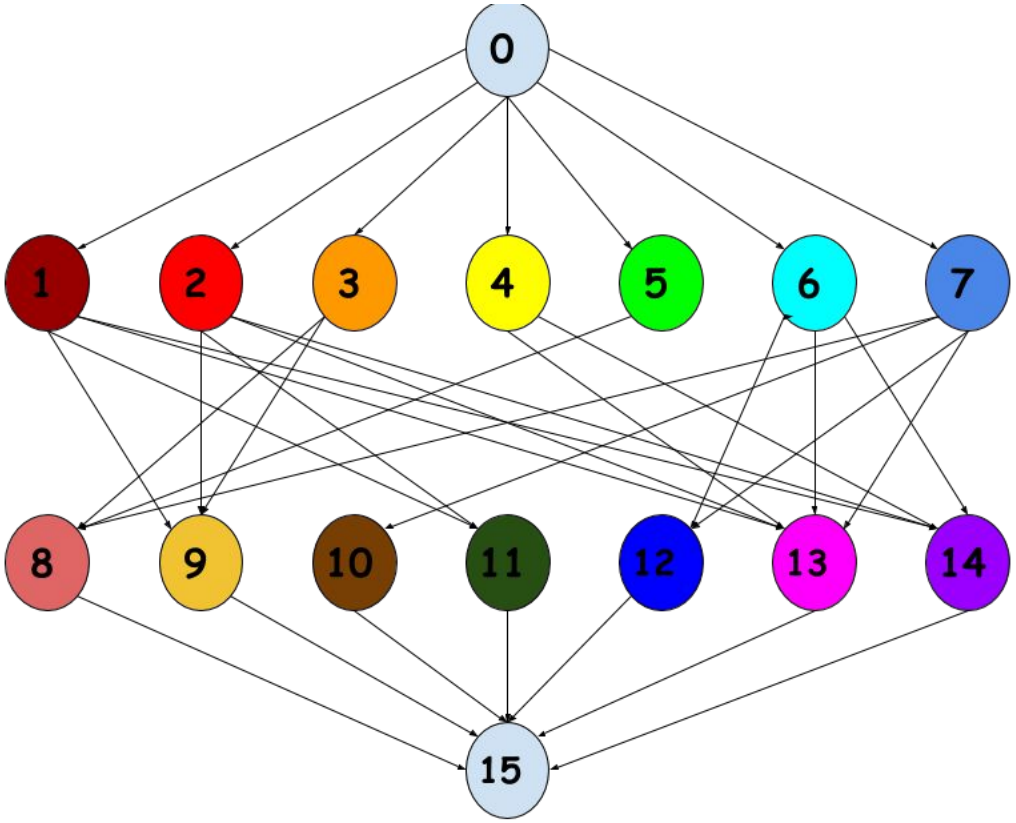
1490 - Torres que Atacam

Solução do exemplo:

Criar um grafo bipartido com um vértice para cada sequência e, posteriormente, uma rede, com todas as arestas tendo peso 1.



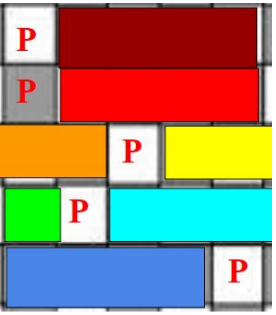


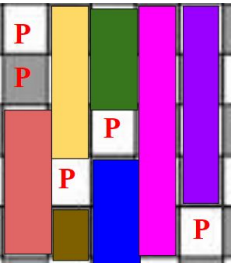


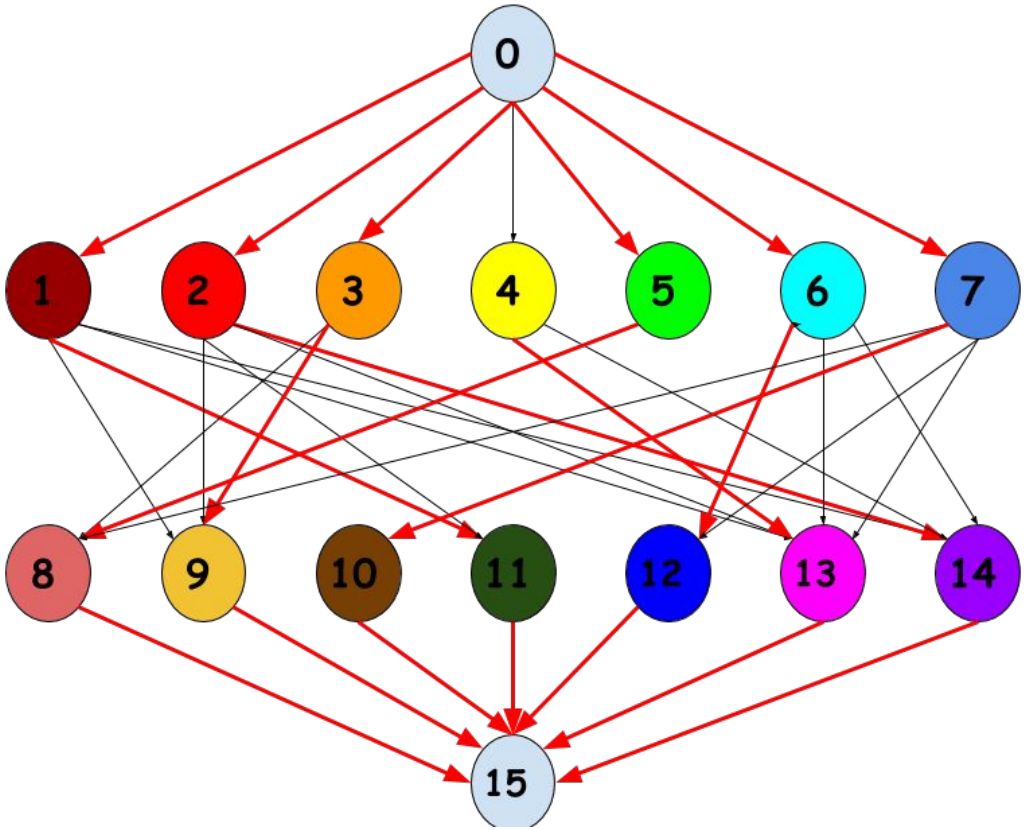
1490 - Torres que Atacam

Solução do exemplo:

Obter o fluxo máximo com um algoritmo eficiente (Dinic).







F = 7

FIM