# Fluxo máximo
# Exercícios

Laboratório de Programação Competitiva - 2020

Pedro Henrique Paiola

# Kill the Werewolf (UVALive - 7896)

- Neste jogo, temos N participantes, dos quais um é o “lobisomem”, e os jogadores devem escolher quem deve ser morto.
- Na primeira rodada, os jogadores não sabem quem é o lobisomem, e então escolhem duas opções de voto.
- Na segunda rodada, o lobisomem se revela, e então os jogadores votam (dentre suas opções). O lobo é o último a votar

# Kill the Werewolf (UVALive - 7896)

- **Objetivo:** supondo que todos os jogadores votam de forma ótima, quantos jogadores podem se revelar como sendo o lobisomem, e ainda assim ganhar o jogo?

# Kill the Werewolf (UVALive - 7896)

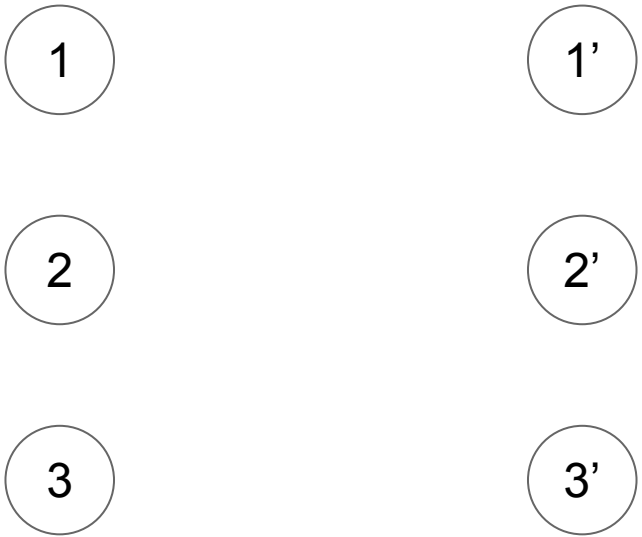
- A primeira observação que podemos fazer é: todos que possuem o lobisomem como opção de voto, irão votar nele.
- Com isso, podemos computar o número X de votos que o lobo receberá.
- Sendo assim, ninguém pode receber mais que X - 1 votos (no empate o lobo ganha)

# Kill the Werewolf (UVALive - 7896)

- Este problema pode ser modelado como um problema de fluxo máximo, onde cada voto é representado por uma unidade de fluxo.

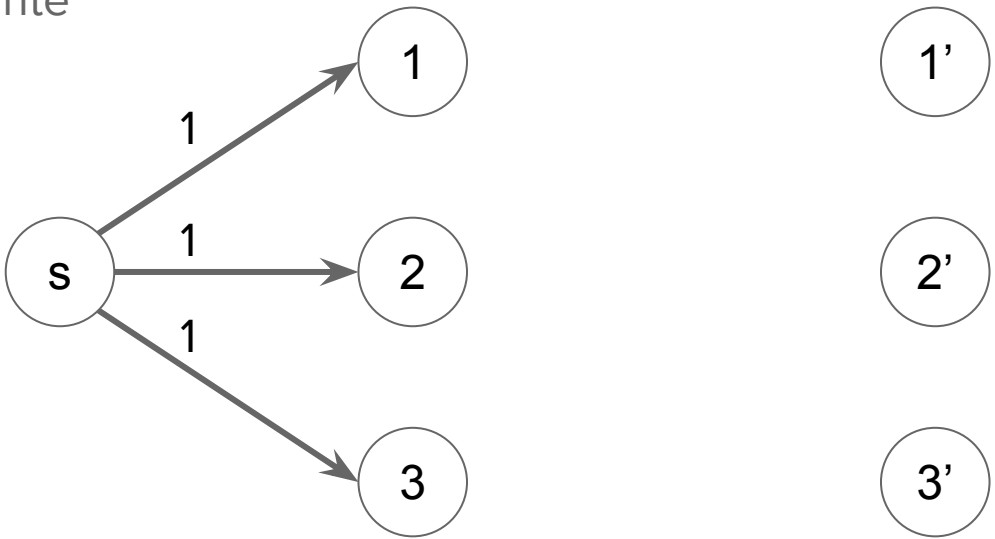
# Kill the Werewolf (UVALive - 7896)

- Cada pessoa será representada por dois vértices, considerando tanto que ela pode votar como também pode ser votada.



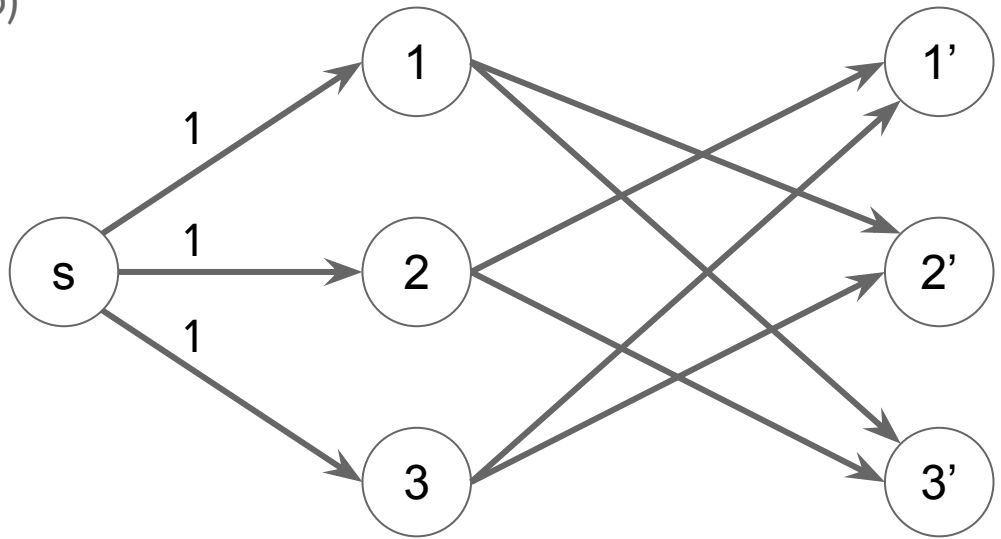
# Kill the Werewolf (UVALive - 7896)

- Como cada pessoa pode votar apenas em uma pessoa, vamos criar uma fonte artificial que fornecerá uma unidade de fluxo (um voto) para cada vértice “votante”



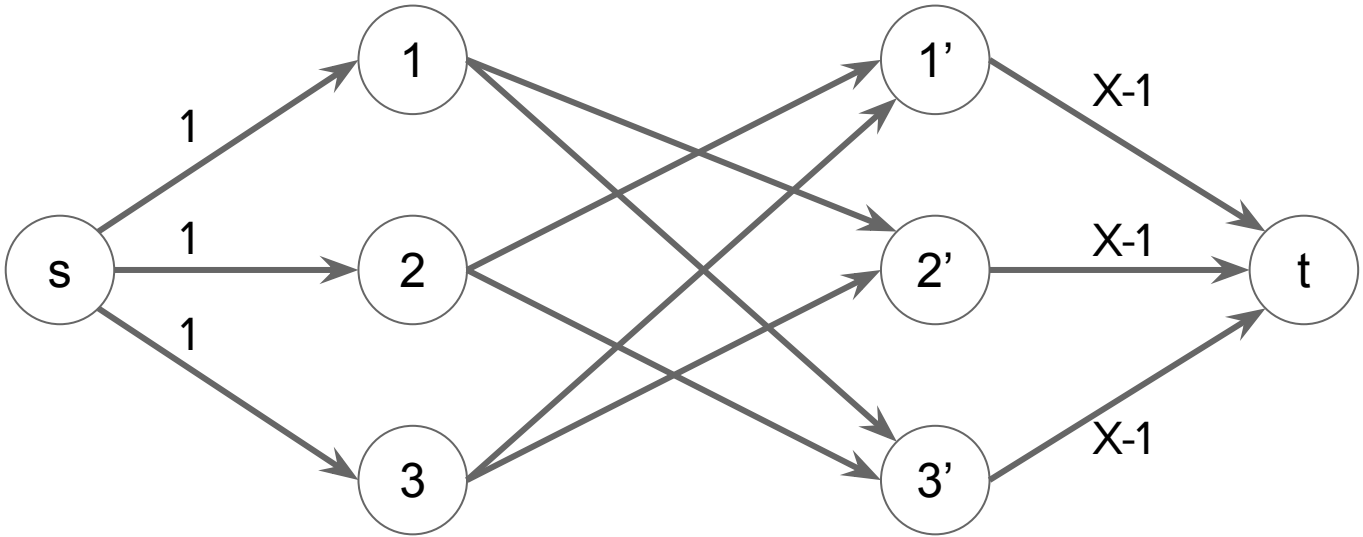
# Kill the Werewolf (UVALive - 7896)

- Cada pessoa tem duas opções de voto possível, que representaremos por arestas (lembrando que quem pode votar no lobo já votou e não entra no grafo)



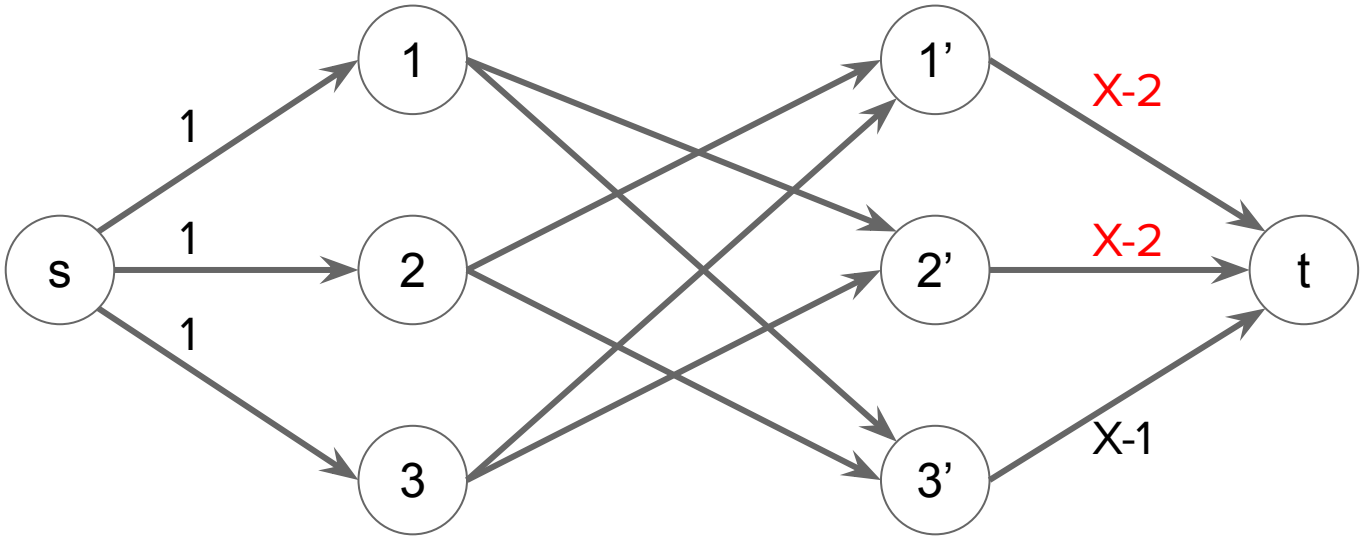
# Kill the Werewolf (UVALive - 7896)

- Por fim, como cada pessoa só pode receber até X - 1 votos, ligaremos os vértices da direita a um sumidouro artificial com capacidade X - 1.



# Kill the Werewolf (UVALive - 7896)

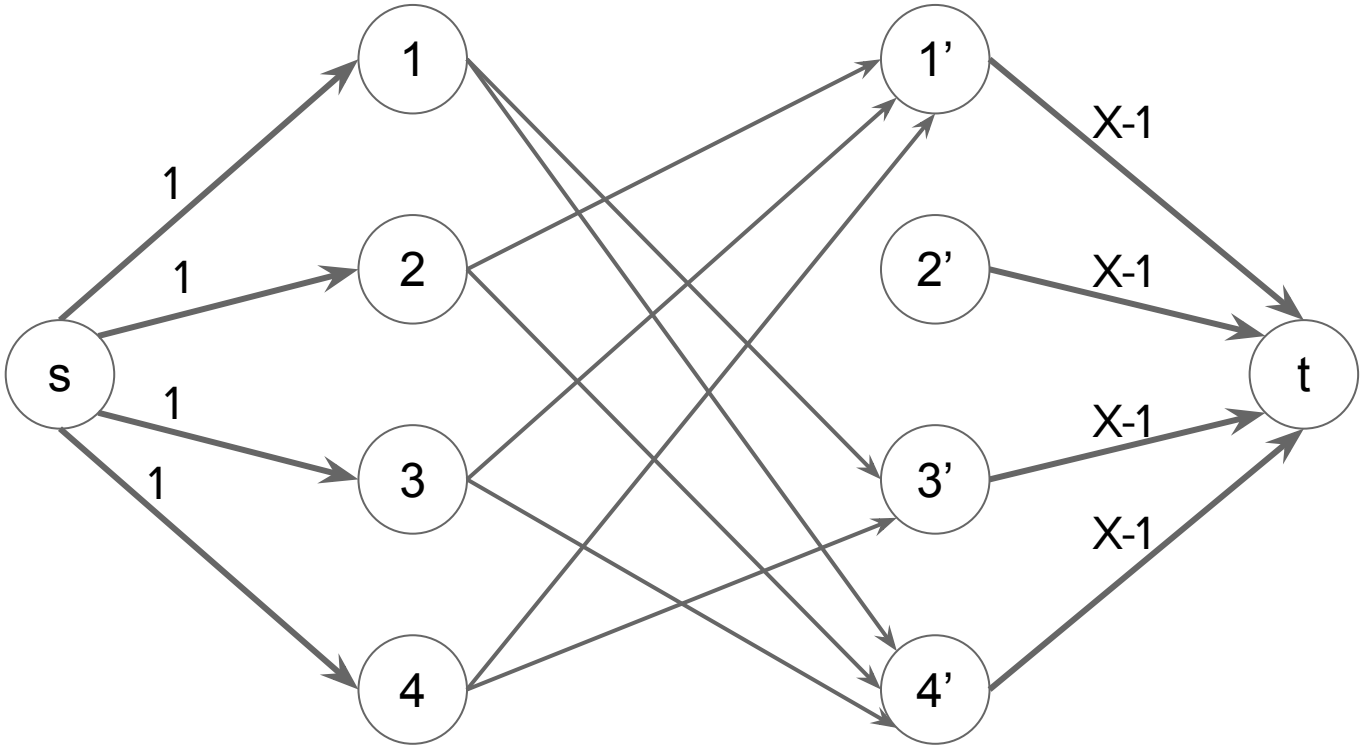
- O lobo não entra nesse grafo diretamente, mas como ele é o último a votar, ele deve priorizar quem recebeu mais votos. Por isso, suas opções de voto só podem receber até X-2 votos.



# Kill the Werewolf (UVALive - 7896)

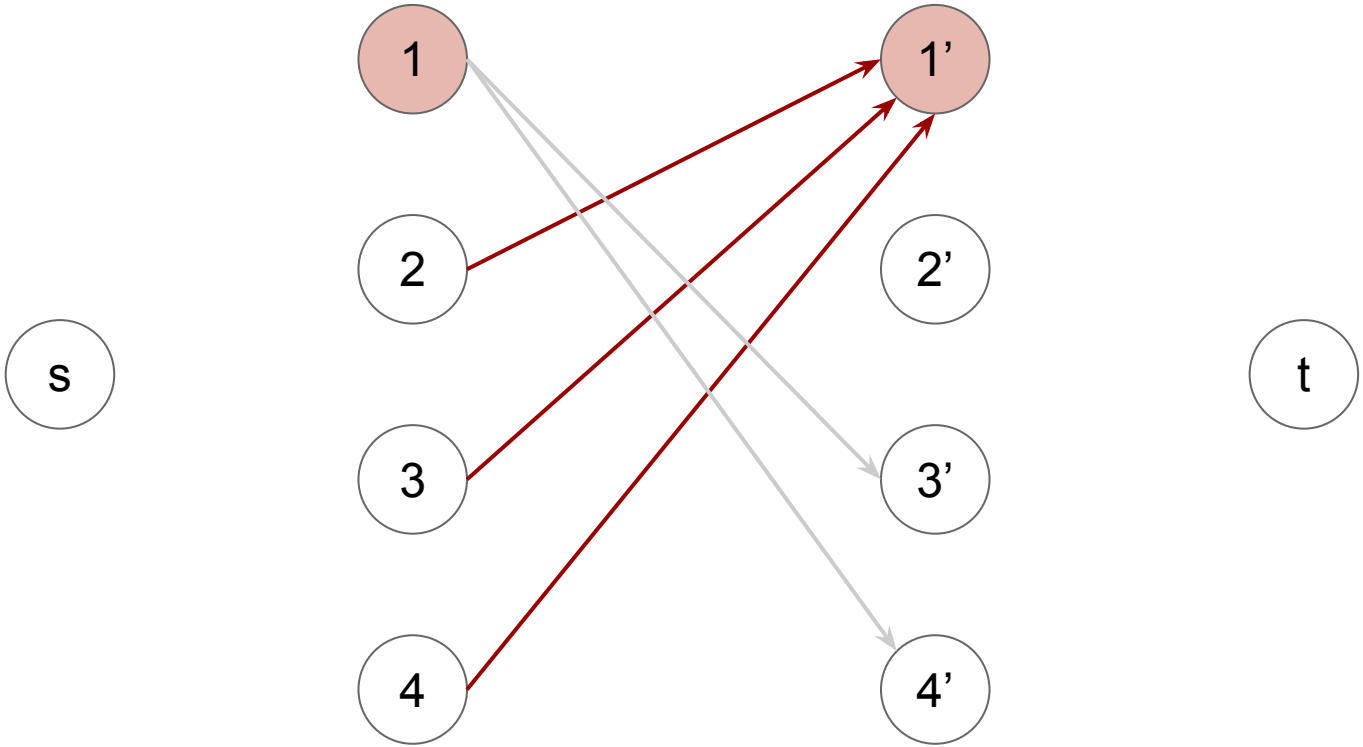
- O lobo recebe X votos, sendo assim, ao aplicarmos um algoritmo de fluxo máximo, nossa resposta deve ser N - X - 1 (o lobo ainda não votou).
- Caso o fluxo seja menor que este valor, então NÃO existe uma combinação de votos possíveis que respeitem as restrições impostas (para garantir a morte do lobo)
- Sendo assim, o lobo sobrevive.

# Kill the Werewolf (UVALive - 7896)



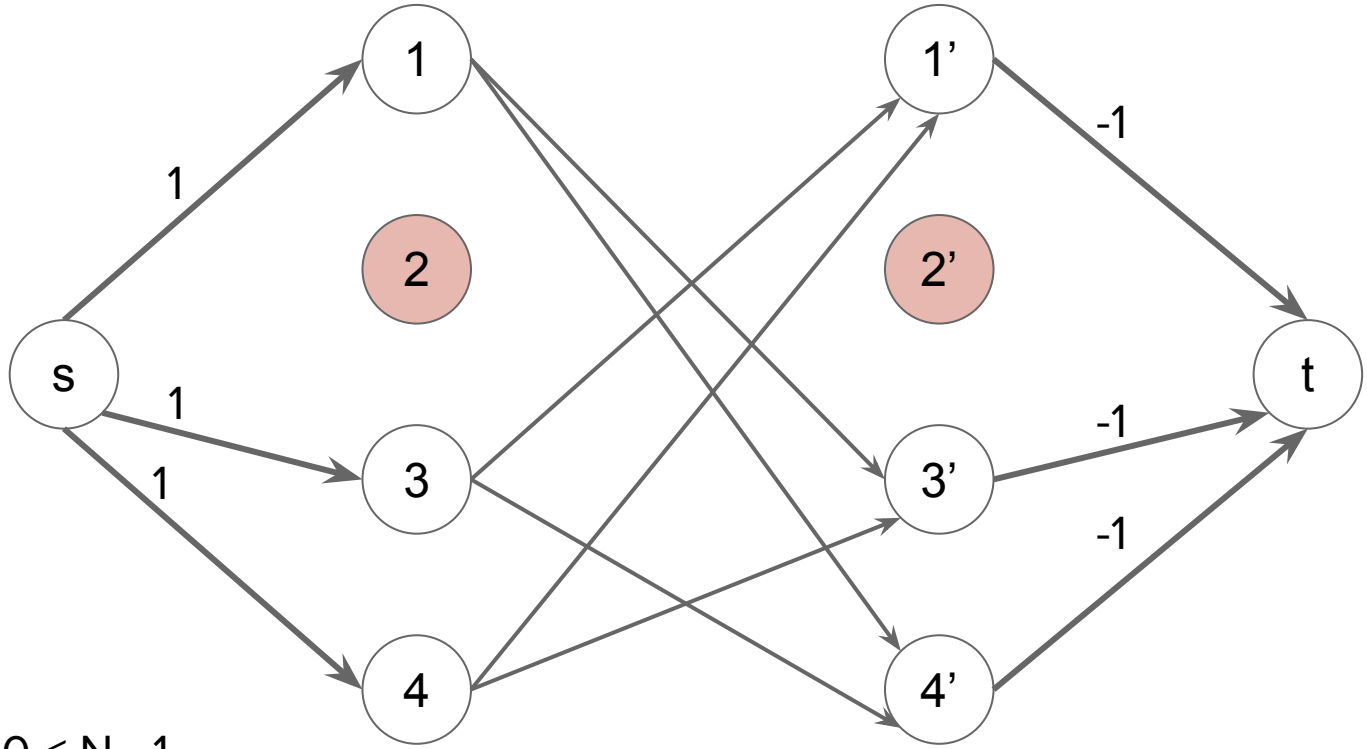

Resp=0

# Kill the Werewolf (UVALive - 7896)



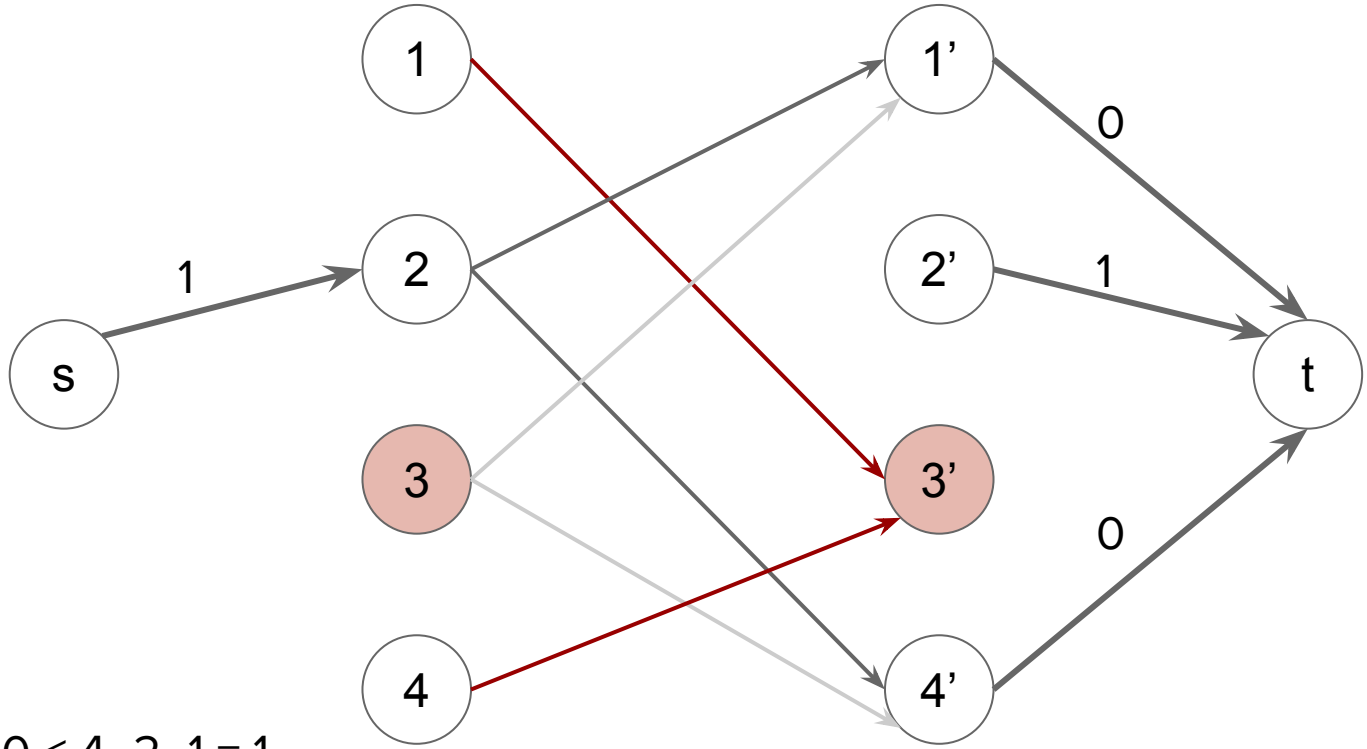

Resp=0

# Kill the Werewolf (UVALive - 7896)



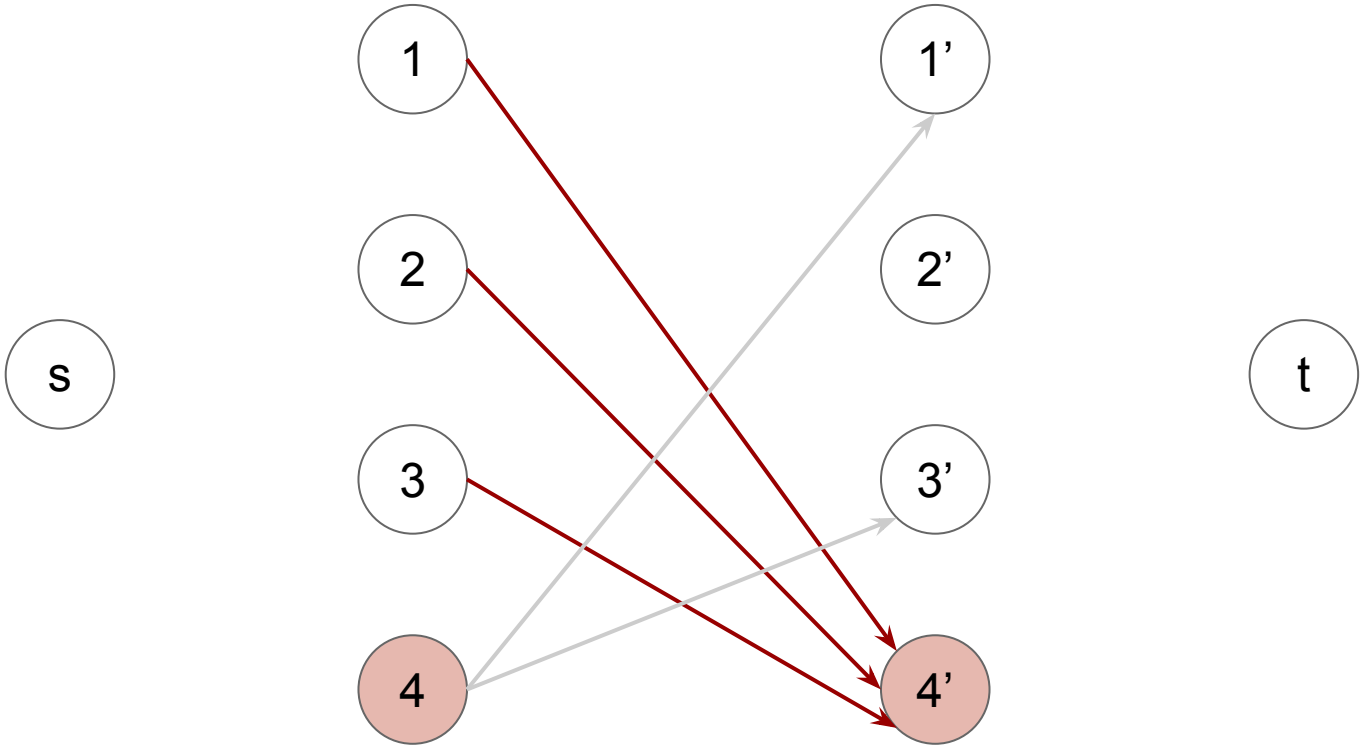

Resp=1  0 < N - 1

# Kill the Werewolf (UVALive - 7896)




Resp=2  0 < 4 -2 -1 = 1

# Kill the Werewolf (UVALive - 7896)




Resp=2

# Collectors Problem (UVA-10779)

- Neste problema, temos um conjunto de amigos trocando figurinhas repetidas, sendo que:
- Bob pode trocar uma figurinha repetida por outra que ele também já possui.
- Os outros amigos só trocam figurinhas repetidas por alguma que eles ainda não possuem.
- Os amigos de Bob só realizam trocas com o Bob.
- **Objetivo:** maximizar o número de diferentes figurinhas que Bob pode conseguir

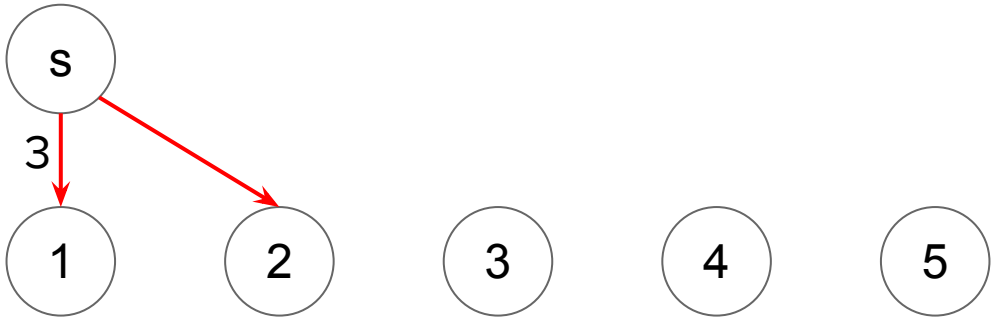
# Collectors Problem (UVA-10779)

- Também iremos modelar este exercício como sendo um problema de fluxo máximo.
- Para exemplificar a modelagem, vamos utilizar o segundo caso de entrada:

3 5
4 1 2 1 1
3 2 2 2
5 1 3 4 4 3

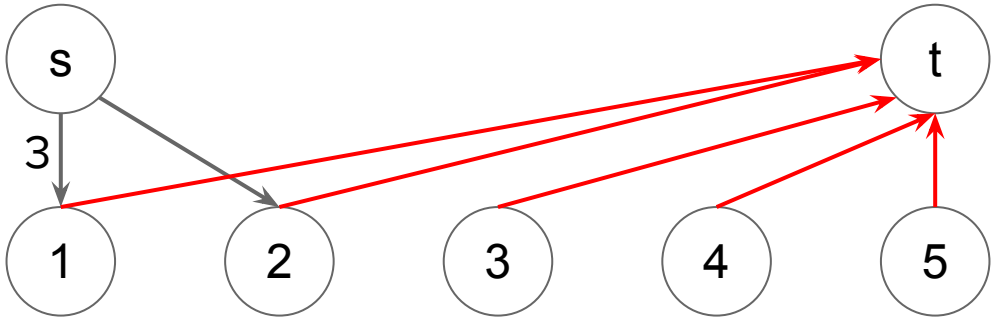
# Collectors Problem (UVA-10779)

- O Bob é o centro do nosso problema, pois todas as trocas partem dele. Vamos construir um vértice para cada tipo de figurinha de Bob. A fonte estará ligada a estes vértices com a quantidade inicial que Bob possui.



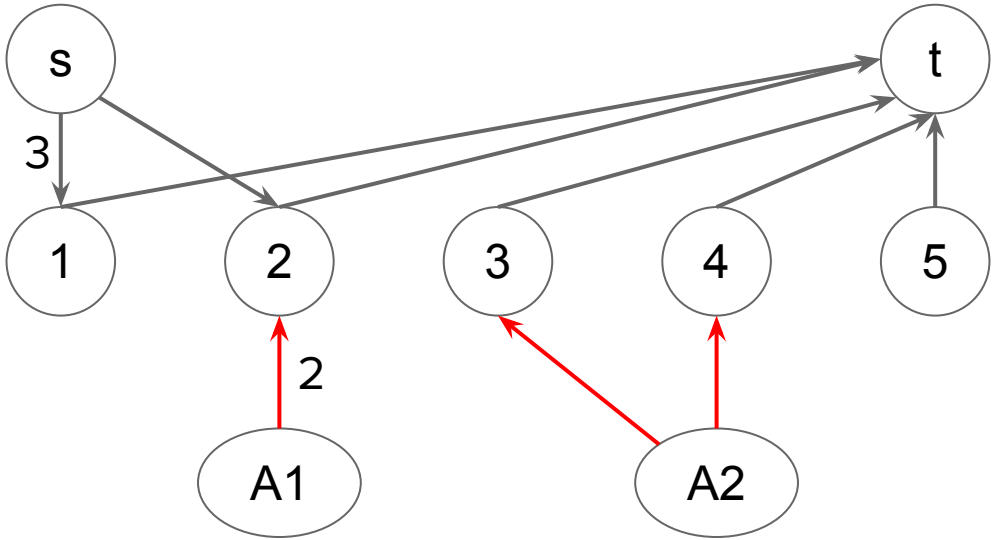
# Collectors Problem (UVA-10779)

- E no final, queremos maximizar o número de diferentes figurinhas. Por isso vamos ligar estes vértices ao sumidouro com capacidade 1 (queremos uma figurinha de cada)



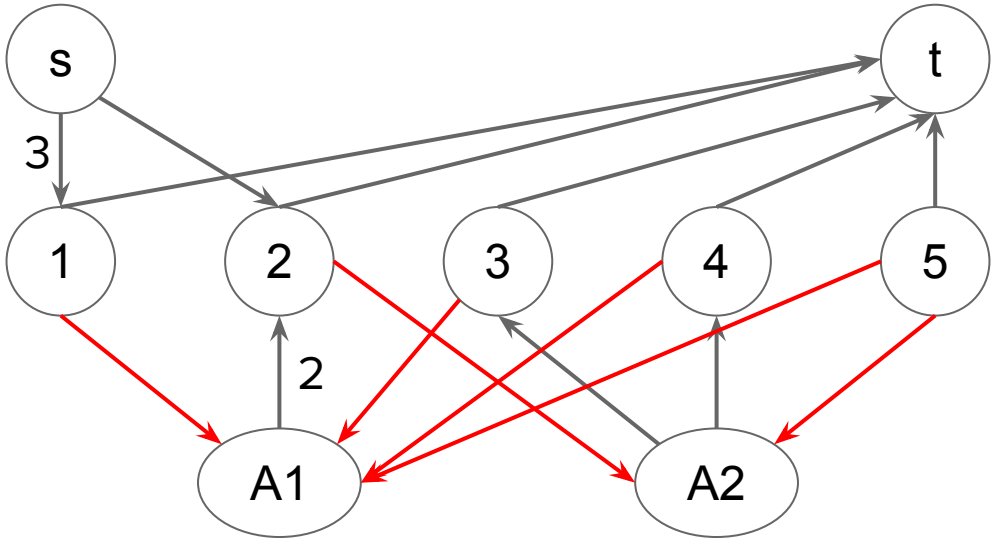
# Collectors Problem (UVA-10779)

- Também criaremos um vértice para cada amigo. Olhando para as figurinhas de cada amigo, se ele tiver mais de uma de um mesmo tipo, ele tem capacidade de trocá-la com Bob



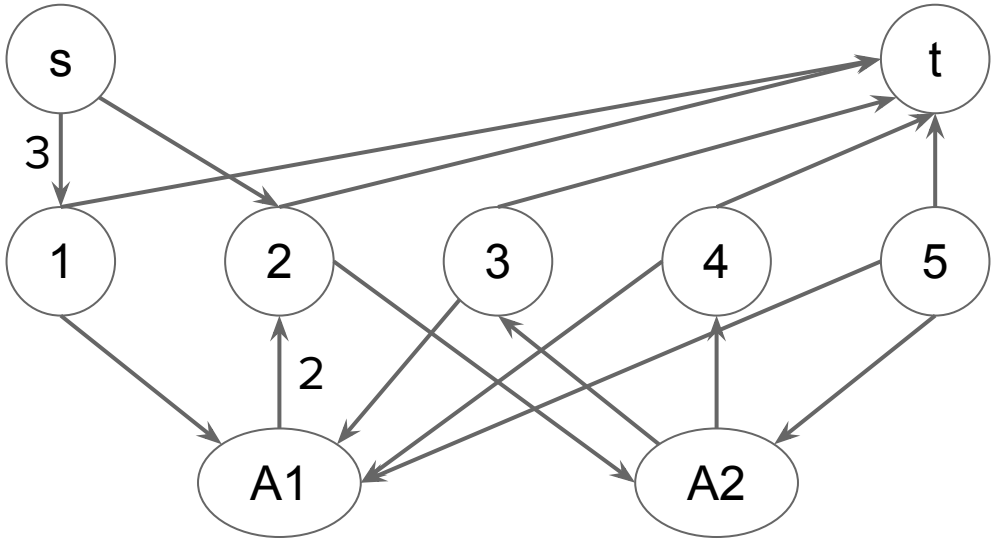
# Collectors Problem (UVA-10779)

- E caso o amigo não possua um certo tipo de figurinha, ele pode receber uma de Bob durante uma troca. Que tudo isso será feito em uma troca é garantido pela conservação dos fluxos (se A1 recebe 2 figurinhas, ele deve dar 2)

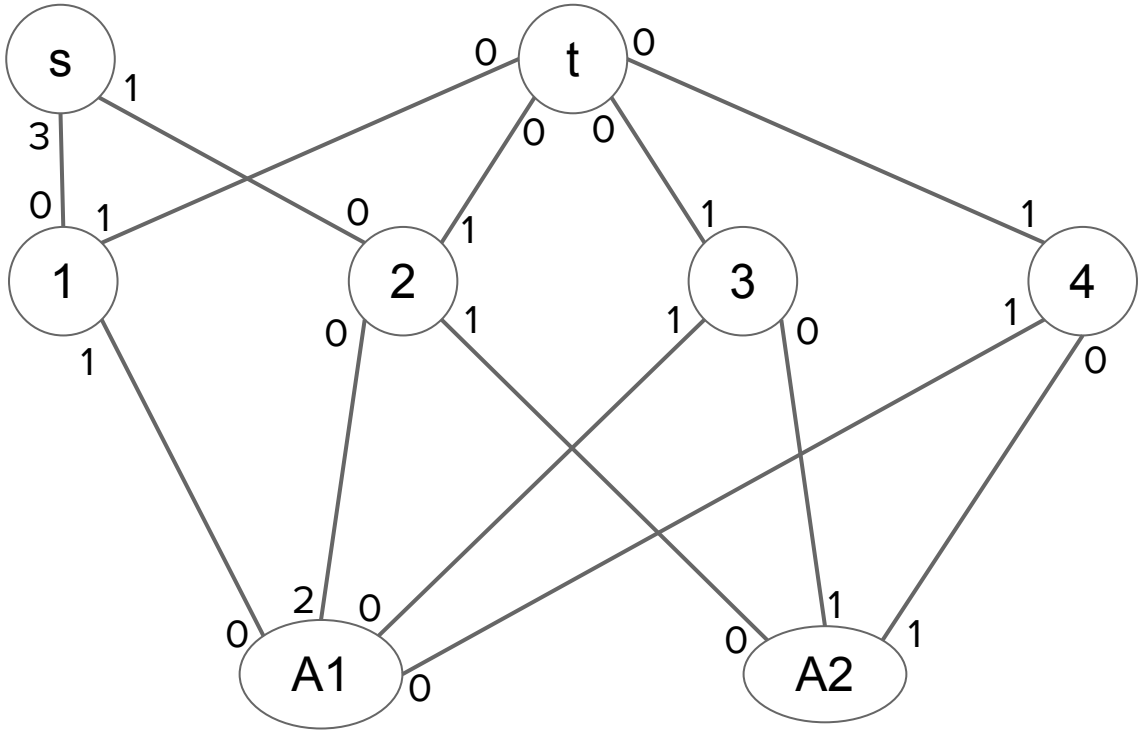


# Collectors Problem (UVA-10779)

- A partir deste grafo, basta aplicar algum algoritmo de fluxo máximo para obtermos nossa resposta.

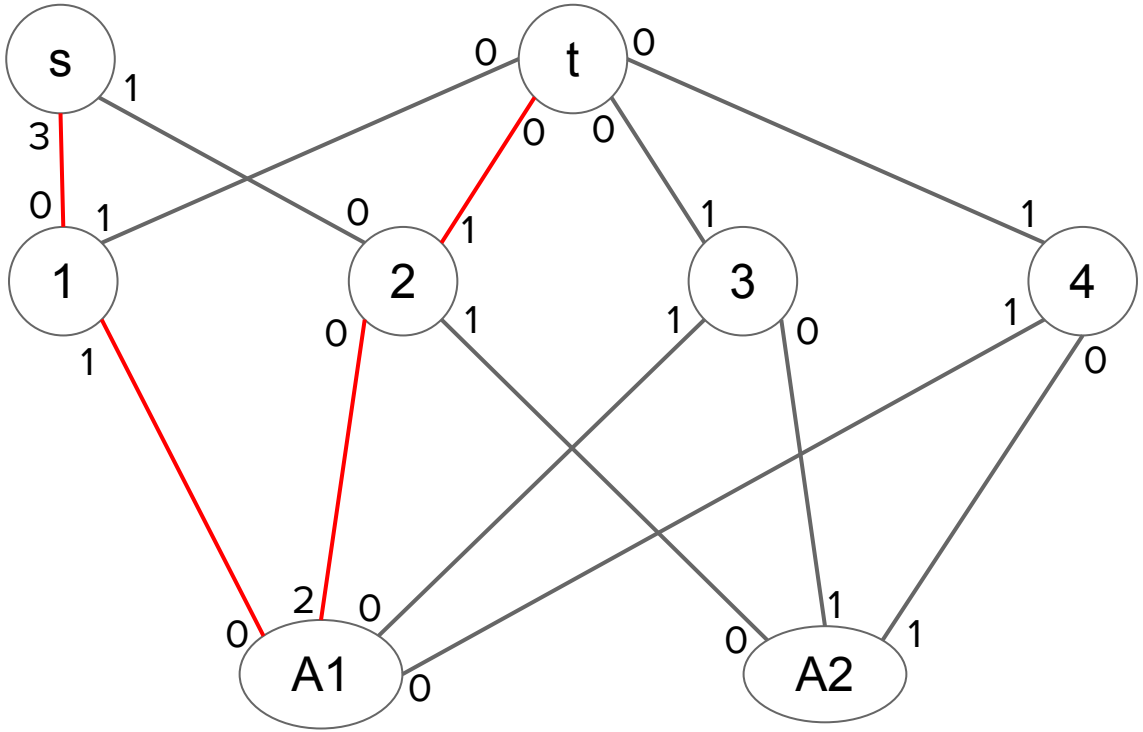


# Collectors Problem (UVA-10779)



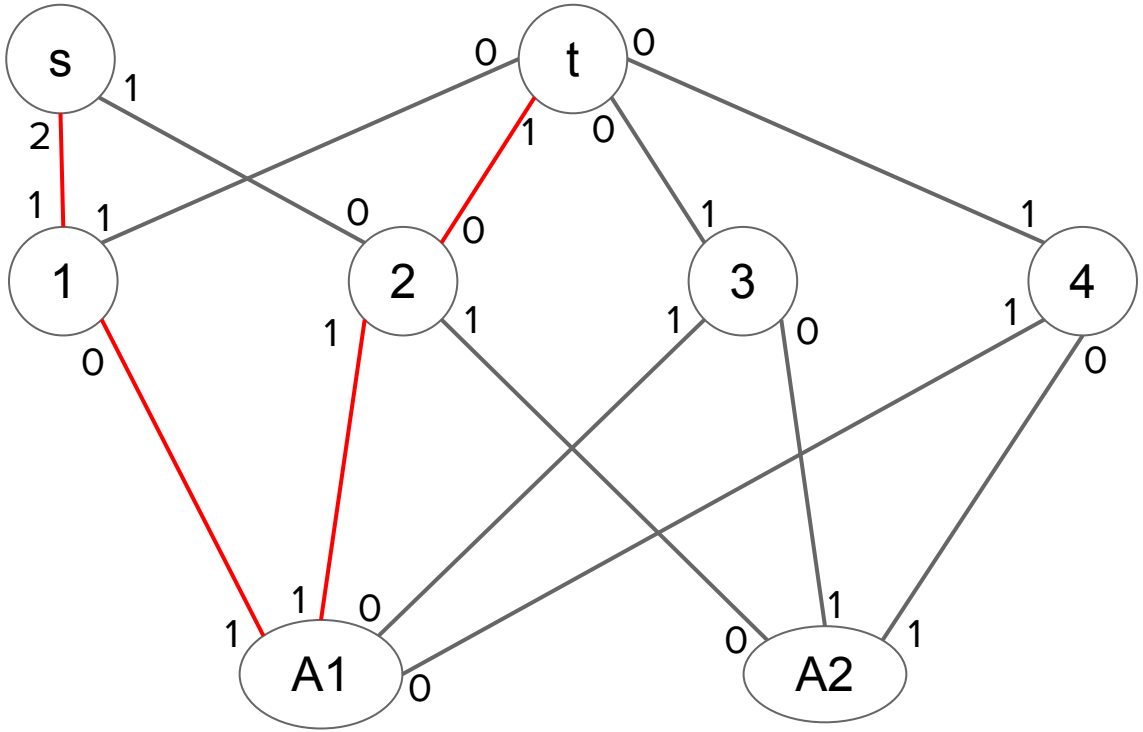

FM = 0

# Collectors Problem (UVA-10779)



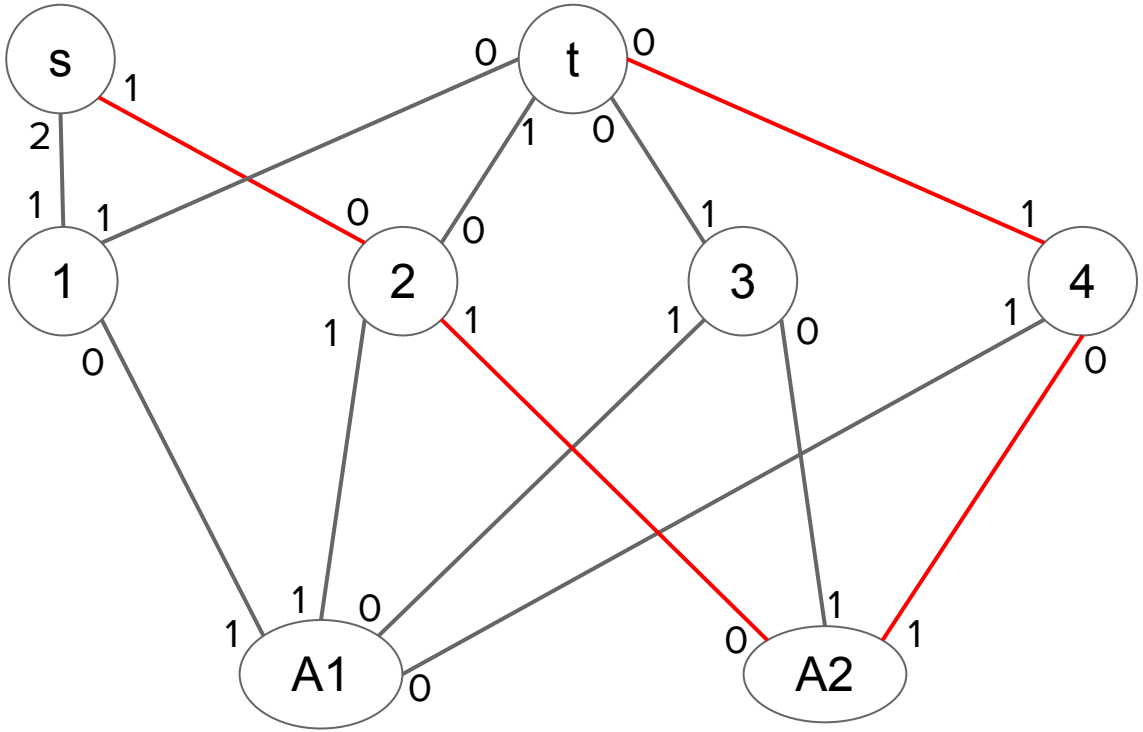

FM = 0

# Collectors Problem (UVA-10779)



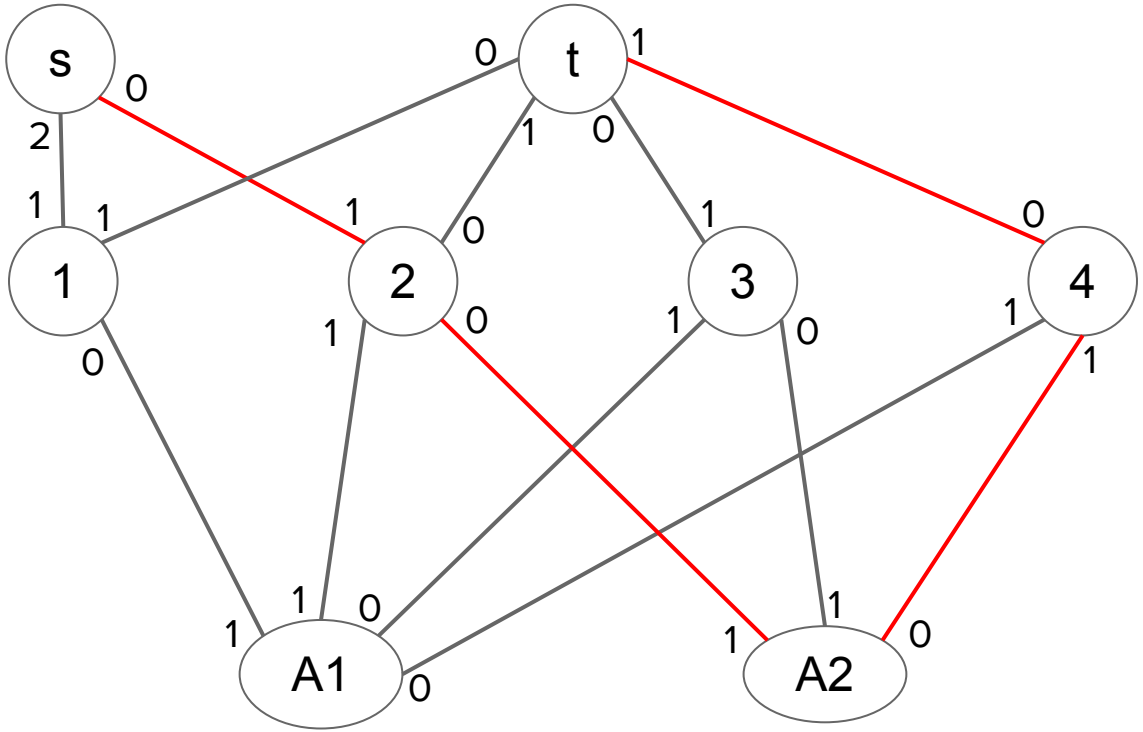

FM = 0

# Collectors Problem (UVA-10779)



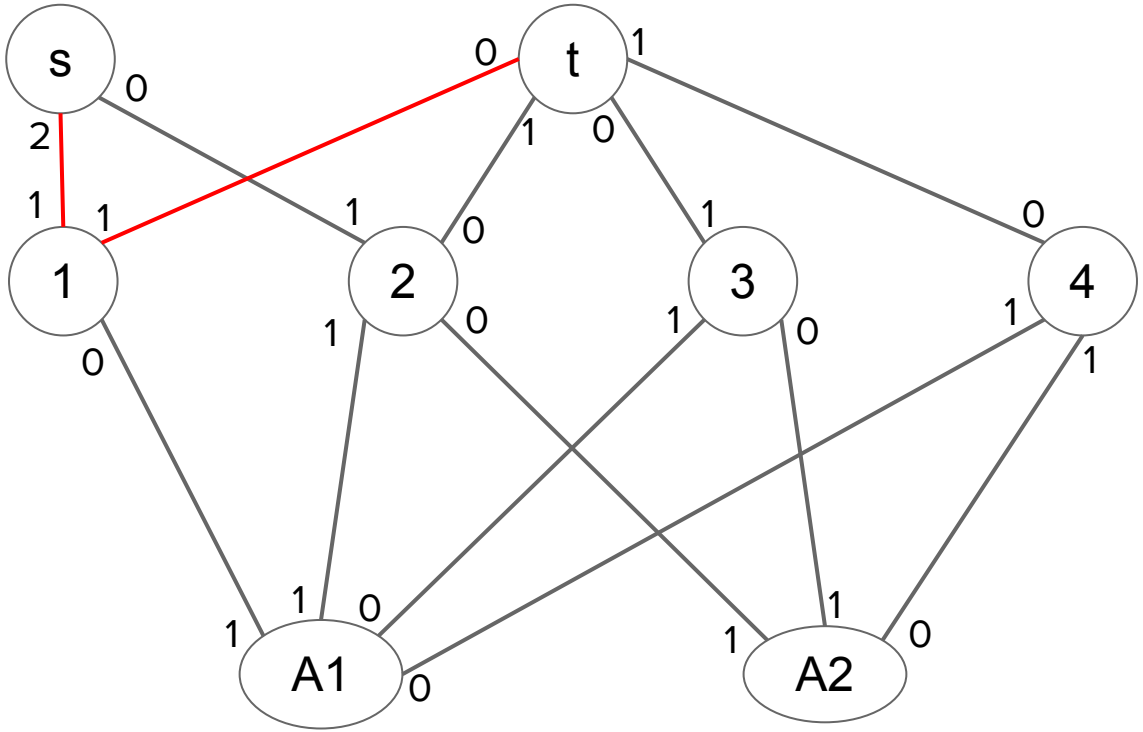

FM = 1

# Collectors Problem (UVA-10779)



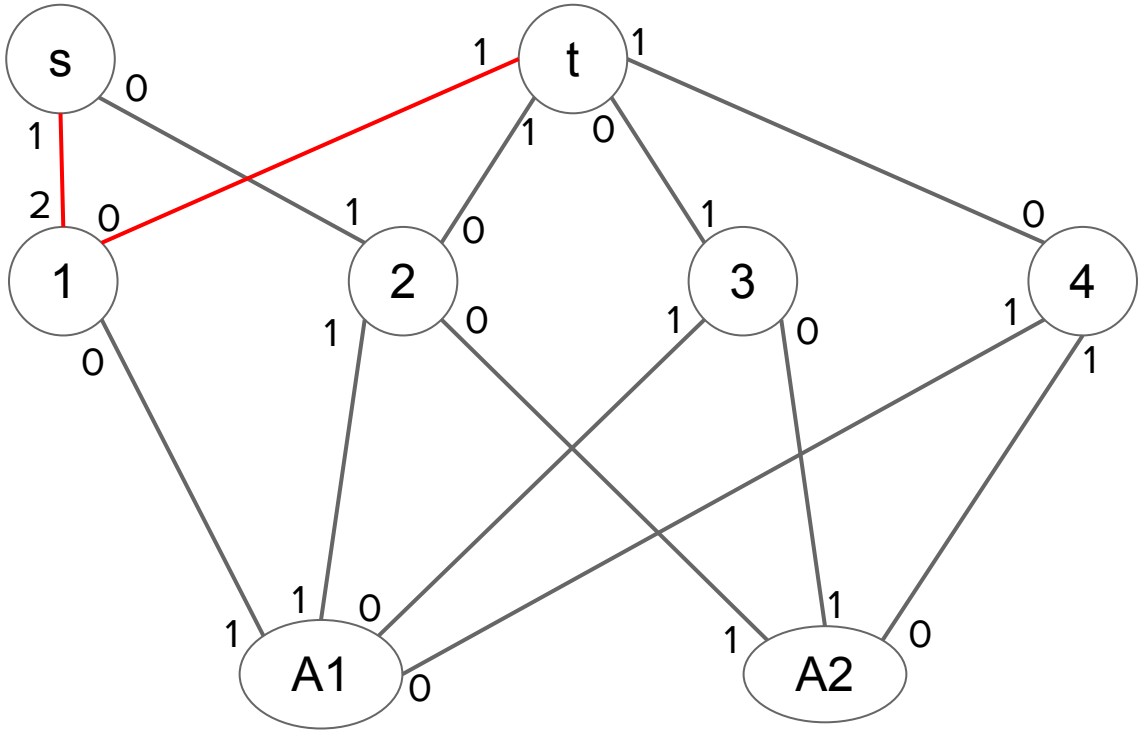

FM = 2

# Collectors Problem (UVA-10779)



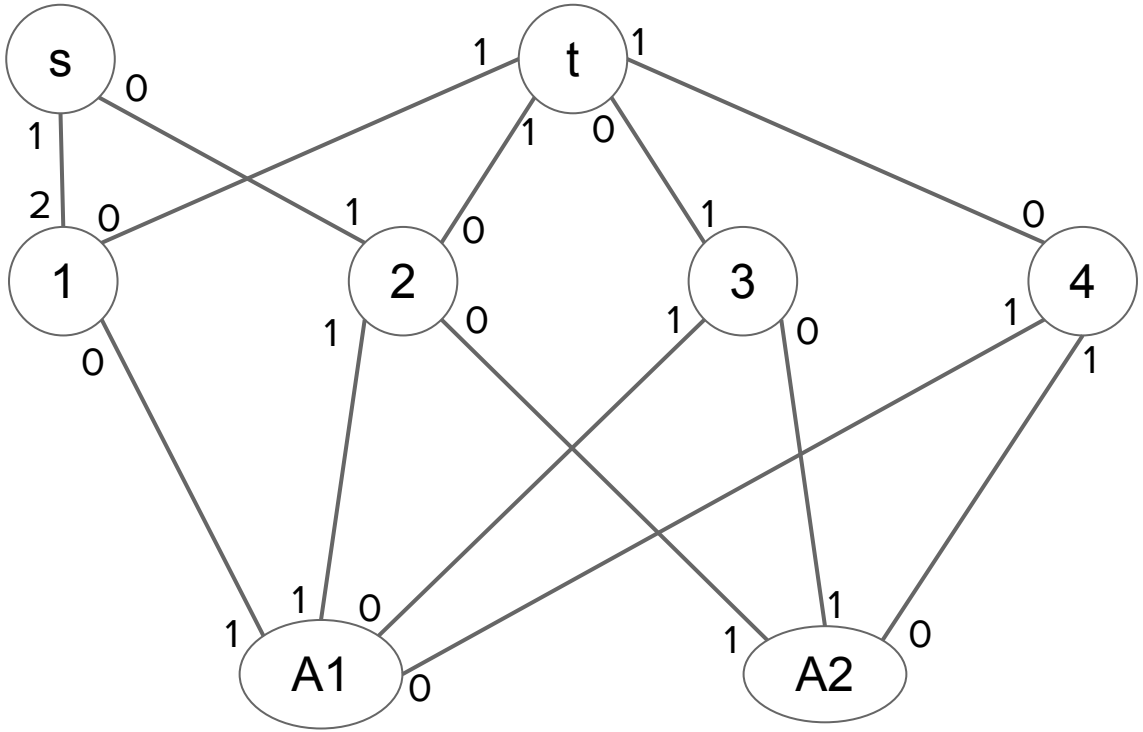

FM = 2

# Collectors Problem (UVA-10779)




FM = 3

# Collectors Problem (UVA-10779)




FM = 3