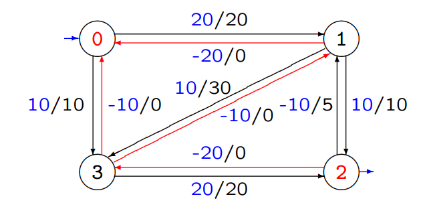
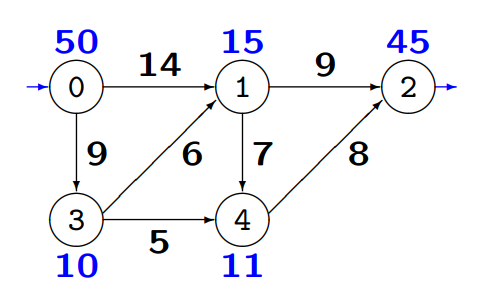
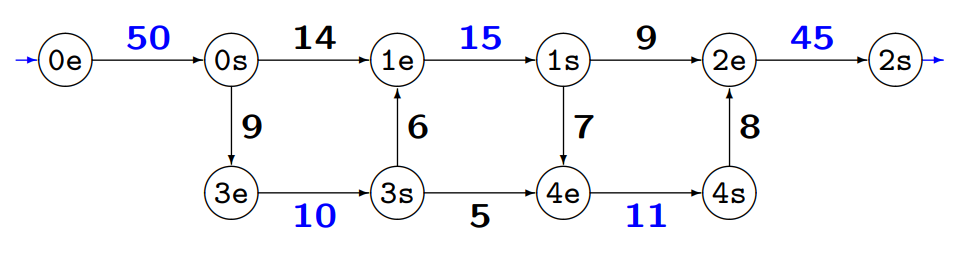
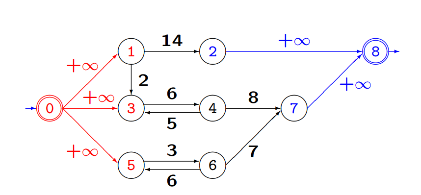
Rescue by rail  
Relatório



# Resolução do Problema



Para a resolução do problema apresentado neste último trabalho, utilizamos o algoritmo de *Edmonds-Karp*. Este algoritmo é uma implementação do método de *Ford-Fulkerson* para calcular o fluxo máximo em uma rede de fluxo. A característica que o distingue é que o caminho de aumento mais curto é usado em cada iteração, o que garante que o cálculo vai terminar.

O algoritmo de *Ford-Fulkerson* é um algoritmo utilizado para resolver problemas de fluxo em rede (*network flow*). O algoritmo é empregue quando se deseja encontrar um fluxo de valor máximo que faça o melhor uso possível das capacidades disponíveis na rede. O método consiste em procurar por um caminho aumentativo e aumentar o fluxo de cada aresta do caminho aumentativo levando em consideração a capacidade residual das mesmas.

Para alem do algoritmo já apresentado é também utilizada a generalização para múltiplas Fonte e a generalização para Pesos nos vértices.

# Complexidade Temporal

Na solução, ao recorremos a generalização para pesos nos vértices e a generalização para múltiplas fontes no grafo, um grafo G = (V, A) passa a ser considerado G’= (1+2V, 2A).

Assim o algoritmo em complexidade temporal fica:  
 - construir a rede de fluxos   
 - inicializar *edges*   
 - *k* x descobrir um caminho   
 - *k* x atualizar o fluxo   
 - *k* x descobrir um caminho

# Complexidade Espacial

- Gráfico com generalização para múltiplas fontes e generalização para pesos nos vértices

- e*dges* (listas ligadas de adjacências)

- Vetor via

- Fila *waiting*

- Vetor Found

- Vetor pathIncr

- Total

# Conclusões

Inicialmente, ao analisarmos o problema, decidimos que a solução para a resolução do problema seria o algoritmo de *Ford-Fulkerson.* Mais tarde concluímos que o algoritmo de *Edmonds-Karp,* com a generalização para várias fontes e para o peso nos nós,era mais adequado para a solução deste problema.

Pensamos, para uma melhor complexidade temporal guardar o fluxo de cada arco dentro do objeto criado. Esta solução traria problemas em termos de complexidade temporal, visto que o algoritmo percorria tudo o que era no, mesmo o que eram *null.* Passamos então a guardar o fluxo dos arcos na respetiva matriz, prejudicando significativamente a complexidade espacial, mas melhorando muito a complexidade temporal como se pode ver nas imagens abaixo.

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente Por fim decidimos abdicar de uma complexidade temporal mais eficiente para uma melhor complexidade espacial ao substituir a matriz de *flow* por um vetor de listas ligadas para cada *edge* com os seus *edges* incidentes.

Uma imagem com texto, captura de ecrã, software, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente   
  
 Sem a matriz *flow* Com a matriz de *flow*

Com vetor de listas ligadas para os *edges* incidentes de cada nó

Pontos fortes são a possibilidade de correr o algoritmo com ponto de escoamento e retornar o valor correto. Apenas o que é guardado na classe é o grafo e tudo o que tem variáveis que são alteradas pelo algoritmo são descartadas depois de ter o resultado.

Uma imagem com padrão, tecido

Descrição gerada automaticamenteDepois da atualização do código não encontramos possíveis melhorias.

Matriz de flow com posições a 0 (sem serem utilizadas)