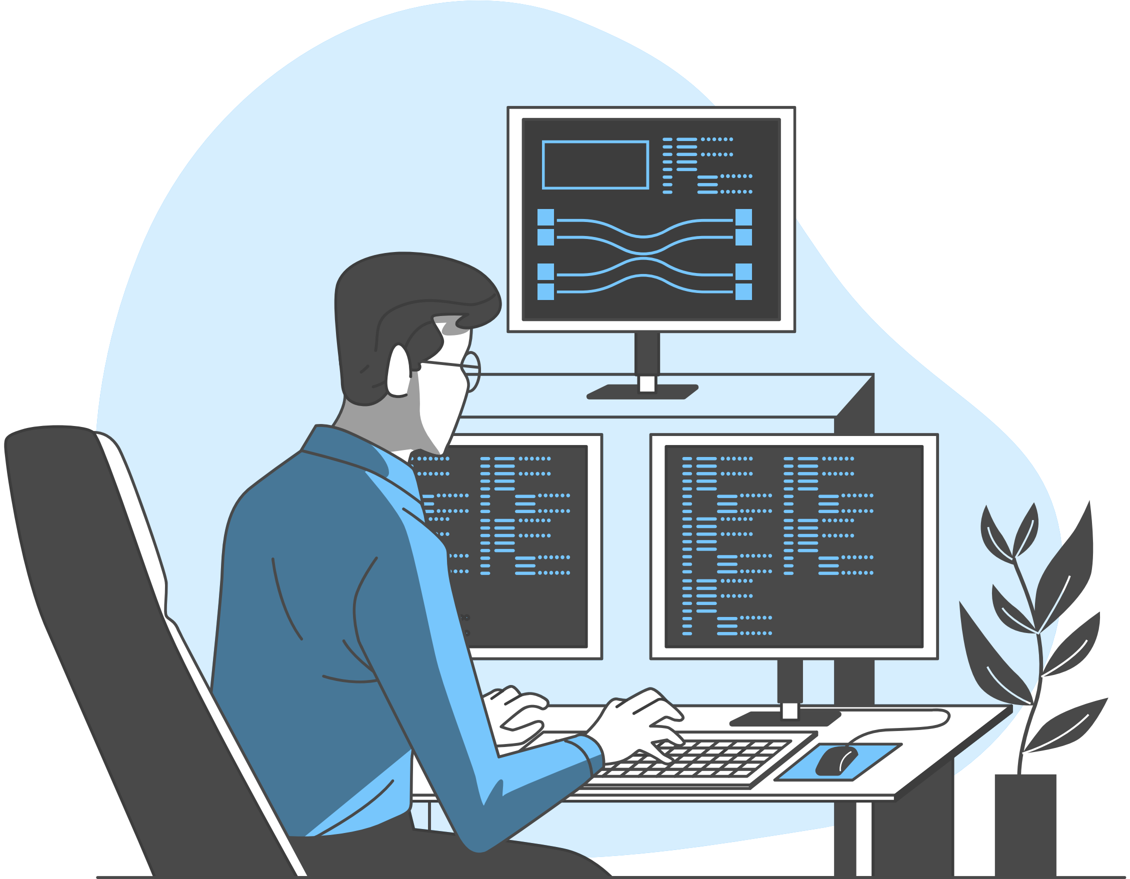
U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO



Agência de Viagens

José Carvalho
Eduardo Correia
Carlos Madaleno

01
Cenário

Grupos que não se separam

1.1

Maximizar a dimensão do grupo
e indicar um qualquer
encaminhamento.

Dados de entrada

V - "Vértices"

E - "Arestas"

e_c - "capacidade da aresta"

Dominio

$$V, E \in \mathbb{N}$$

$$\forall \; e \in E, e_c \in \mathbb{R}^+$$

Dados de saida

Gp - "tamanho do grupo"

Encaminhamento:

Caminho com max(Capacidade)

Variáveis de decisão

Encaminhamento:
Caminho com max(Capacidade)

Restrições

$$\forall \text{ e} \in \text{Encaminhamento},$$
$$\text{e}_c \geq \text{Gp}$$

Função Objetivo

$$\max(\text{Gp})$$

Descrição do Algoritmo

1. Adicionar Primeiro nó do grafo (source) a uma priority queue que ordena os valores pela sua capacidade (sendo que o maior elemento está na primeira posição)
2. Enquanto a priority queue não está vazia :
    a. Ir buscar e remover o primeiro elemento da lista
    b. Para cada nó ligado ao nó atual :
        i. Se o nó ainda não foi visitado:
        ii. Mudar a capacidade da aresta para o mínimo entre a capacidade do nó e a capacidade da aresta atual.
        iii. Guardar no nó ligado o nó fonte.
        iv. Adicionar o nó ligado à priority queue com a capacidade da aresta.
3. Reconstruir o caminho do inicio ao fim através do no source em cada nó.

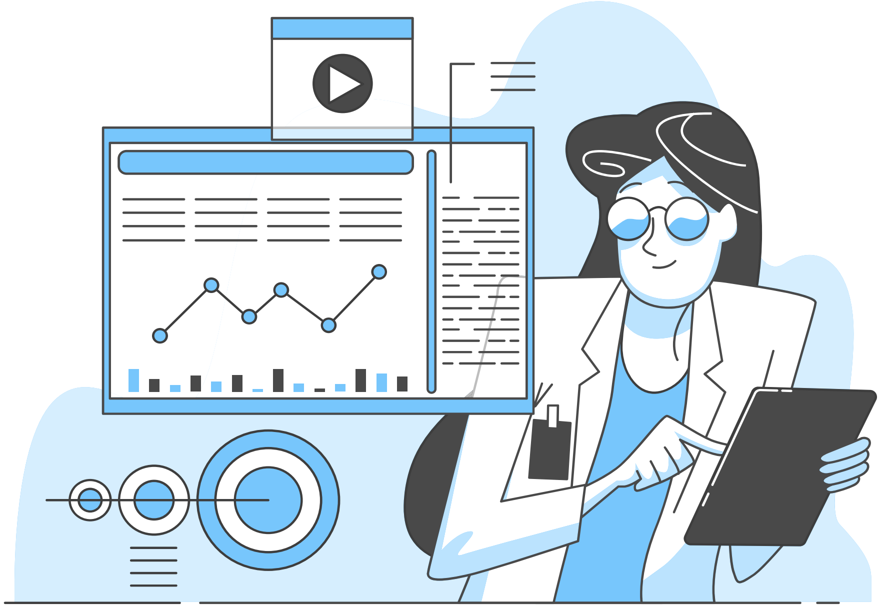
Análise da complexidade

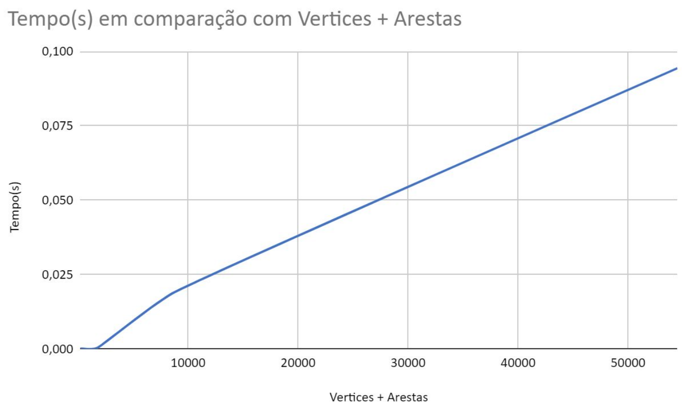
01 Complexidade temporal

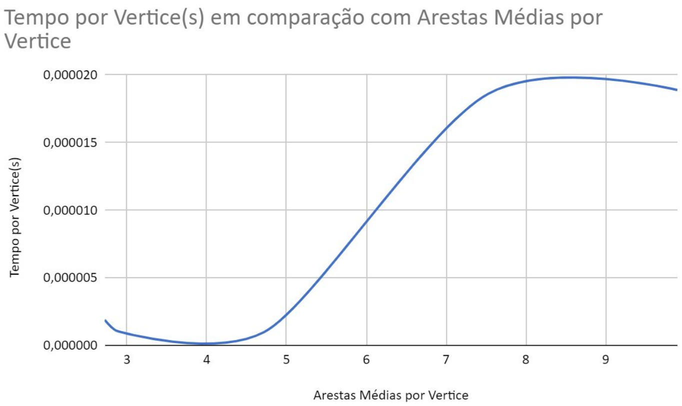
- Tempo para percorrer todos os vertices : O(V+E)
- Tempo necessário para processar um vértice : O(log V)
- Tempo necessário para processar e visitar todos os vértices (complexidade total) : **O((V+E)log V)**

02 Complexidade Espacial

- Espaço para os vértices : O(V)
- Espaço para as arestas : O(E)
- Espaço para as arestas e vértices :O(V + E)
- Espaço para a priority queue (V + E)
- Espaço total : O(2(V + E)) = **O(V+E)**







Avaliação

Empirica

1.2

Maximizar a dimensão do grupo e minimizar o número de transbordos, sem privilegiar um dos critérios relativamente ao outro.

Dados de entrada

V - "Vértices"

E - "Arestas"

e_c - "capacidade da aresta"

Dominio

$$\mathrm{V,E} \in \mathbb{N}$$

$$\forall\ \mathrm{e} \in E, \mathrm{e_c} \in \mathbb{R}^+$$

Dados de saida

Encaminhamento

Variáveis de decisão

$$\text{Capacidade}_i < \text{Capacidade}_j,$$
$$\text{j} = \text{i+1}$$
$$\text{Transbordo}_i > \text{Transbordo}_j,$$
$$\text{j} = \text{i+1}$$

Encaminhamento:

Caminho com max(Capacidade)

Função Objetivo

$$\max(\text{Capacidade}) \wedge \min(\text{Transbordos})$$
$$\bigvee$$
$$\min(\text{Transbordos}) \wedge \max(\text{Capacidade})$$

Restrições

$$\mathrm{e}_{\text{final}} = \mathrm{e}_{\text{destino}}$$

Descrição do Algoritmo

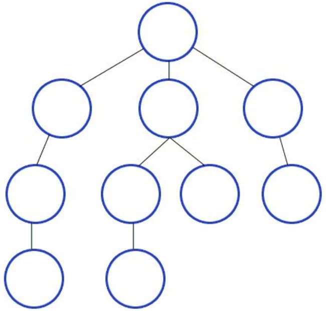
Este algoritmo tem por base a utilização de um DFS para obter todos os caminhos presentes num grafo desde a origem até ao destino.

Para evitar encontrar sempre o mesmo caminho, os caminhos encontrados são removidos/bloqueados para as procuras seguintes.

Seguidamente, e como é ilustrado no slide **13**, os resultados são filtrados conforme o necessário.

*O algoritmo DFS usado é descrito em mais detalhes no slide **20**.

01



Encontrando todos os caminhos da origem ao fim múltiplas instâncias de DFS, fechando caminhos até não haver mais.

02

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

Supondo que estamos a calcular para max (capacidade) o algoritmo procura o valor nos paths resultantes de 01,

03

5

Depois de obter o valor, volta a pesquisar qual o encaminhamento com 5 com capacidade que tem menos transbordos e retorna-o,

5	4

A mesma lógica aplica-se quando o objetivo é priorizar o mínimo de transbordos.

*Os resultados do ponto 2, não são obtidos do gráfico de 1

Análise da complexidade

01

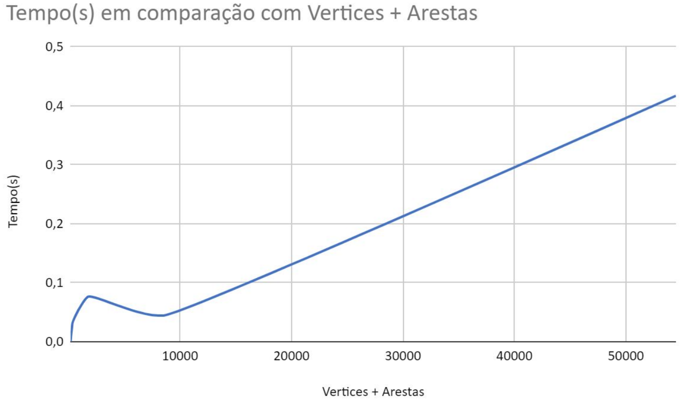
Complexidade temporal

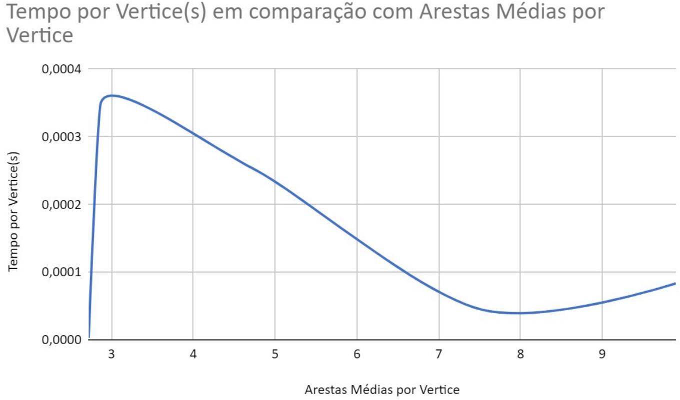
- Percorrer o grafo todo N vezes: **O(N(V+E))**, onde N = max(GroupSize)

02

Complexidade espacial

- Guardar um caminho em forma de stack : O(V+E)
- Guardar group size caminhos : **O(group Size(V))**





Avaliação

Empirica

02
Cenário

Grupos que podem separar-se

2.1

Determinar um encaminhamento para um grupo, dada a sua dimensão

Dados de entrada

$$V : ''vertices''$$
$$E : ''arestas''$$
$$e_c : ''capacidade\ por\ aresta''$$
$$groupSize : ''tamanho\ do\ grupo''$$

Dominio

$$\mathrm{V,E} \in \mathbb{N}$$
$$\forall\ \mathrm{e} \in E, \mathrm{e_c} \in \mathbb{R}^+$$

Dados de saida

Encaminhamentos que levam o grupo ao destino

Variáveis de decisão

$$vetorEncaminhamento:$$
$$\text{"}vetor\ que\ contem\ caminhos$$
$$que\ tem\ o\ destino\ pretendido\text{"}$$
$$Encaminhamento_c:$$
$$\text{"}capacidade\ minima$$
$$das\ arestas\ do\ caminho\text{"}$$

Restrições

$$\sum_{Encaminhamento_c\ \epsilon\ vetorEncaminhamento} >= groupSize$$
$$groupSize > 0$$

Função Objetivo

null

Descrição do Algoritmo

1. Igualar “n” a groupSize
2. Enquanto “n” for maior que 0:
    a. Caminho = EncontrarCaminho()
    b. Se existir um caminho para n pessoas:
        i. Guardar caminho
        ii. Reduzir capacidade do caminho por “n” no grafo
    c. Se não existir, reduzimos n por 1.

Descrição do Algoritmo

EncontrarCaminho():

1. Colocar nó inicial numa stack.
2. Enquanto a stack não estiver vazia :
    a. Buscar o nó no topo da stack.
    b. Se for o nó final, encontrou um caminho, e quebra o loop.
    c. Se não for :
        i. Se existe uma aresta conectada ao nó com capacidade > groupSize.
            1. Adiciona o nó destino à stack.
        ii. Se não existir
            1. Retira um elemento da stack.

3. Retorna o estado atual da stack.

Análise da complexidade

01

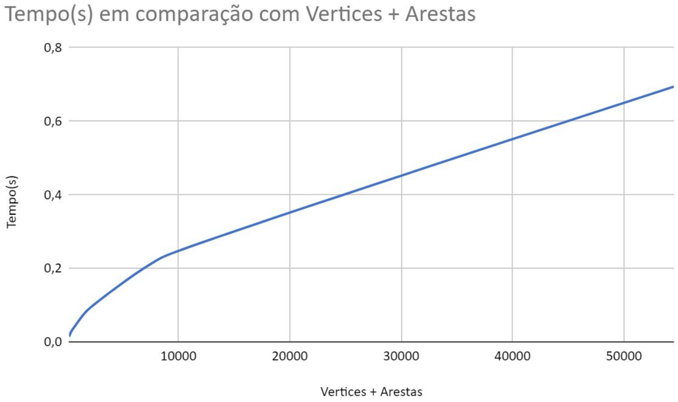
Complexidade temporal

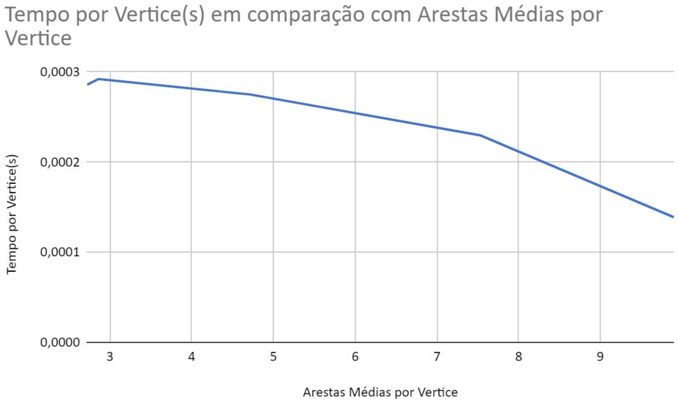
- Percorrer o grafo todo : O(V+E)
- Percorrer o grafo todo group Size vezes : **O(group Size(V+E))**

02

Complexidade espacial

- Guardar um caminho em forma de stack : O(V+E)
- Guardar group size caminhos : **O(group Size(V))**





Avaliação

Empirica

2.2

Algoritmo de Procura de um Desvio

Dados de entrada

V - "Vértices"

E - "Arestas"

C - "Capacidade"

P – "*Passageiros a adicionar*"

Dominio

$V, E, P \epsilon \mathbb{N}$

e_c - "capacidade da aresta"

$\forall$ e $\in E$, $e_c \in \mathbb{R}^+$

Dados de saida

Encaminhamento com possível desvio

Variáveis de decisão

$$Encaminhamento_c : Capacidade\ maxima\ das\ arestas\ do\ caminho$$

$$vetorEncaminhamento : "vetor\ que\ contem\ caminhos\ que\ tem\ o\ destino\ pretendido"$$

$$groupSize : "tamanho\ do\ grupo"$$

Restrições

$$\sum_{Encaminhamento_c\ \epsilon\ vetorEncaminhamento} >= groupSize$$

$$groupSize > 0$$

Função Objetivo

$$max(Passageiros) \wedge min(Transbordos)$$

Descrição do Algoritmo

1. Enquanto 'i' for maior que o tamanho do o vetor de encaminhamentos:
    a. Verificação da capacidade máximo do encaminhamento[i]
    b. Se essa capacidade não foi atingida:
        i. E se for menor que o número de pessoas a adicionar
            1. Atualizar a capacidade do caminho no grafo
            2. groupSize = 0
        ii. Se não:
            1. adicionam-se o número de pessoas possível
            2. groupSize = groupSize - fluxo remanescente
        iii. Se groupSize igual a 0, o ciclo é terminado
2. Se groupSize for negativo:
    a. É mostrado em ecrã o encaminhamento
3. Se for positivo, não se conseguiu alocar todos os passageiros:
    a. Inicia a procura por novo encaminhamento ou desvio

Análise da complexidade

01

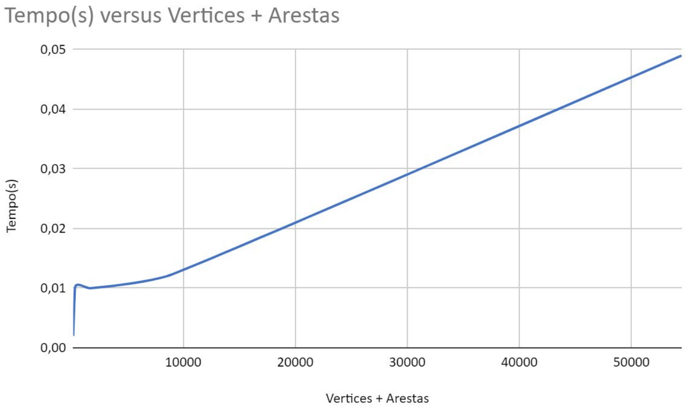
Complexidade temporal

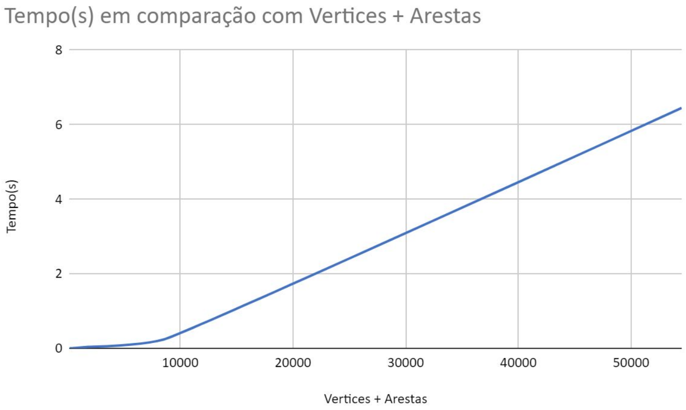
- Procura de fluxo remanescente: **O(Paths(V+E))**
- Atualizar a capacidade do caminho: **O(Paths(V+E))**
- Procura de um desvio: **O(group Size(V+E))**

02

Complexidade espacial

- Guardar um caminho em: **O(V+E)**
- Guardar group size caminhos : **O(groupSize(V))**





Avaliação
Empirica

2.3

Determinar a dimensão máxima do grupo e um encaminhamento.

Dados de entrada

$$V : \text{"}vertices\text{"}$$
$$E : \text{"}arestas\text{"}$$
$$e_c : \text{"}capacidade\ por\ aresta\text{"}$$

Dominio

$$\mathrm{V,E} \in \mathbb{N}$$
$$\forall\ \mathrm{e} \in E, \mathrm{e_c} \in \mathbb{R}^+$$

Dados de saida

Tamanho máximo do grupo
Encaminhamentos de fluxo que levam o grupo ao destino

Variáveis de decisão

$$shortestPath: \text{"caminho mais curto entre o inicio e o destino"}$$

$$groupSize: \text{"numero de pessoas que passaram pela rede"}$$

Restrições

$$shortestPath_c > 0$$
$$groupSize > 0 => \exists_{shortestPath} \epsilon Graph$$

Função Objetivo

$$\max(\text{groupSize})$$

Descrição do Algoritmo

Enquanto houver caminhos que cheguem ao fim, com capacidade > 0 :

a. Encontrar o caminho mais curto com capacidade > 0 entre o início e o fim.
b. Calcular a capacidade máxima do caminho.
c. Aumentar o fluxo do caminho mais curto pela capacidade máxima do caminho (diminuir a capacidade das arestas pela capacidade máxima).
d. Guardar o caminho e o número de pessoas enviadas.

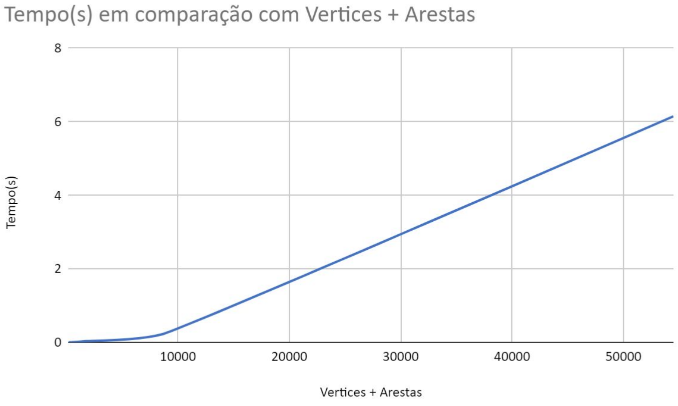
Análise da complexidade

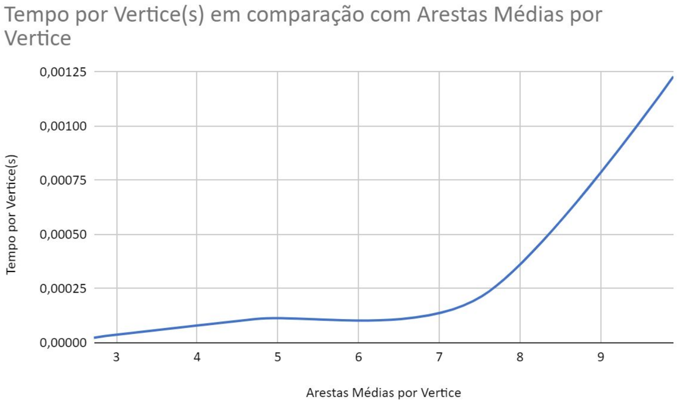
01 Complexidade temporal

- Calcular o caminho mais curto entre dois nós : (V + E)
- Calcular a capacidade máxima do caminho : O (V)
- Aumentar o fluxo do caminho : O(V)
- Repetir até não ser possível enviar mais fluxo : **O(V . E^2)**

02 Complexidade espacial

- Utilização de queue para pesquisa : O (V+E)
- Guardar n caminhos de flow : **O (N(V))**





Avaliação

Empirica

2.4

Partindo de um encaminhamento que constitui um grafo acíclico, determinar quando é que o grupo se reuniria novamente no destino, no mínimo.

Dados de entrada

$$V : \text{"}vertices\text{"}$$
$$E : \text{"}arestas\text{"}$$
$$e_c : \text{"}capacidade\ por\ aresta\text{"}$$
$$e_d : \text{"}duracao\ da\ aresta\text{"}$$

Dominio

$$\text{V,E} \in \mathbb{N}$$
$$\forall_e \epsilon E, e_c \epsilon \mathbb{R}^+$$
$$\forall_e \epsilon E, e_d \epsilon \mathbb{R}^+$$

Dados de Saida

Tempo mínimo que demora
a percorrer o grafo

Variáveis de decisão

$$finishTime :$$
$$"tempo\ que\ um\ grupo\ demora\ a\ percorrer\ o\ grafo"$$

Restrições

$$finishTime > 0$$

Função Objetivo

$$min(finishTime)$$

Descrição do Algoritmo

1. Calcular os caminhos que permitem o maior grupo chegar ao final (Correr o algoritmo 2.3)
2. Criar um novo grafo apenas com estes caminhos.
3. Calcular o tempo mínimo para a finalização de cada nó :
    a. Colocar o primeiro nó numa queue (com finalização a 0).
    b. Enquanto a queue não estiver vazia :
        i. Remover nó no primeiro elemento da queue
        ii. Para cada aresta ligada a este nó :
            1. Se o nó destino tiver a finalização < finalização do nó fonte + a duração da aresta :
                a. Igualar a sua finalização a finalização do nó fonte + a duração da aresta da ligação.
                b. Se nó destino ainda não estiver na lista de nós visitados :
                    i. Adicionar nó a lista de visitados.
                    ii. Adicionar nó a queue.

Análise da complexidade

01

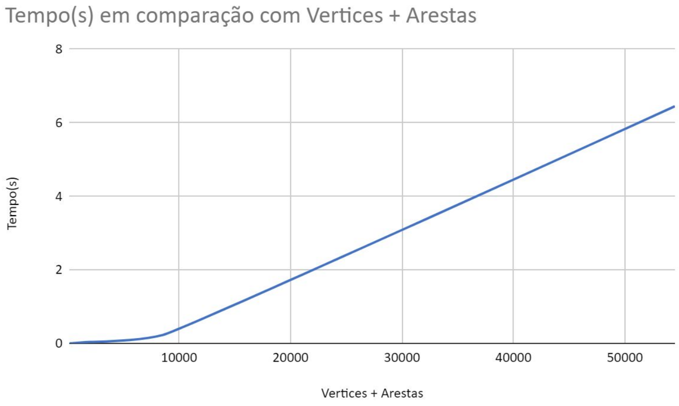
Complexidade temporal

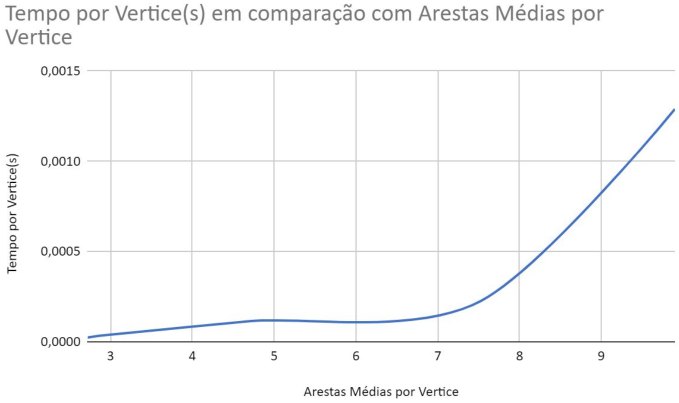
- Correr o algoritmo 2.3 : **O (V . E^2)**
- Criar um novo grafo com os caminhos : O (V+E)
- Percorrer o novo grafo para calcular o tempo de finalização : O(V+E)

02

Complexidade Espacial

- Correr o algoritmo 2.3 : **O (N(V))**
- Criar um novo grafo com estes caminhos : O(N+V)
- Calcular o tempo mínimo de finalização (percorrer o grafo inteiro) : O(N+V)





Avaliação
Empirica

2.5

Indicar o tempo máximo de espera e os locais em que haveria elementos que esperam esse tempo

Dados de entrada

$$V : \text{"}vertices\text{"}$$
$$E : \text{"}arestas\text{"}$$
$$e_c : \text{"}capacidade\ por\ aresta\text{"}$$
$$e_d : \text{"}duracao\ da\ aresta\text{"}$$

Dominio

$$\mathrm{V,E} \in \mathbb{N}$$
$$\forall_e \epsilon E, e_c \epsilon \mathbb{R}^+$$
$$\forall_e \epsilon E, e_d \epsilon \mathbb{R}^+$$

Dados de Saida

$$totalWaitTime$$

Nós em que os grupos esperam

Variáveis de decisão

$$totalWaitTime:$$
$$\text{"}tempo\ que\ um\ grupo$$
$$espera\ na\ totalidade\ a\ percorrer\ o\ grafo\text{"}$$

Restrições

$$totalWaitTime > 0$$

Função Objetivo

$$null$$

Descrição do Algoritmo

1. Correr o algoritmo 2.4.
2. Inserir nos nós as arestas opostas, isto é para cada aresta A->B, inserir uma nova aresta B-A, com as mesmas caracteristicas.
3. Igualar o campo de finalização mais tarde à finalização mais cedo no último nó.
4. Percorrer o grafo com um BFS (da mesma forma que é percorrido no 2.4) do início ao fim, com as novas arestas opostas.
5. Enquanto o grafo é percorrido, o campo de cada nó de finalização mais tarde possível é atualizado, subtraindo a duração da aresta que leva a esse nó (se este valor for menor do que o já estiver lá inserido).
6. O tempo livre é calculado subtraindo o campo de início mais cedo, com o campo de início mais tarde, em todos os nós.

Análise da complexidade

01

Complexidade temporal

- Complexidade do algoritmo 2.4 : **O (V . E^2)**
- Complexidade de calcular o latest start : O (V+E)
- Complexidade de calcular o free time : O (V+E)

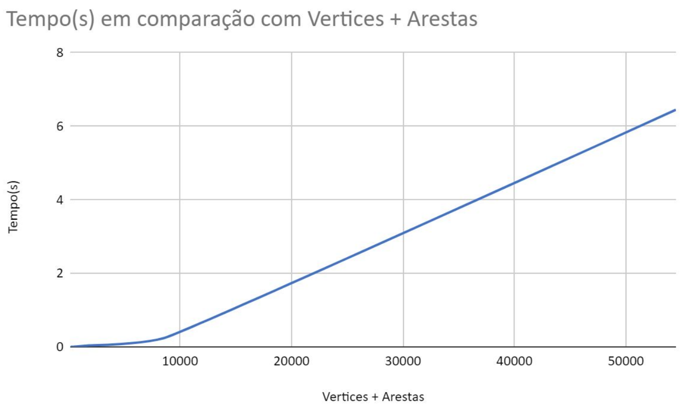
02

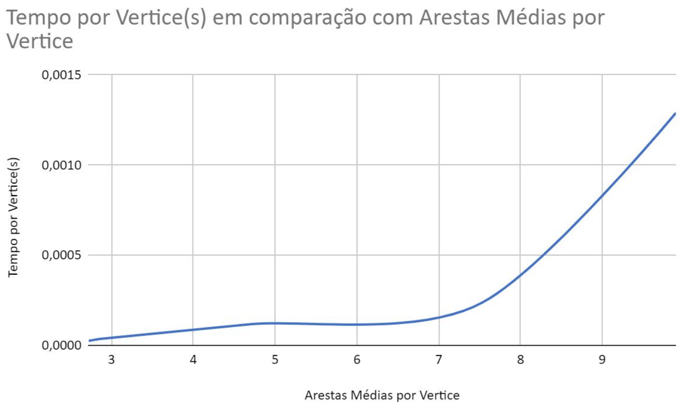
Complexidade Espacial

- Complexidade do algoritmo 2.4 : **O (N(V))**
- Utilizar uma queue para pesquisa : O(N+V)

Avaliação

Empirica





Solução Algorítmica de destaque

Decidimos destacar o algoritmo de **Edmund-Karps,** uma vez que foi o algoritmo que usado para o problema que tivemos mais dificuldade.

Em comparação ao algoritmo Ford-Fulkerson este tem uma melhor complexidade temporal e tem uma implementação concreta.
Assim o algoritmo tem um desempenho melhor quando aplicado em larga escala e permite uma implementação mais uniforme.

Uma aplicação direta do algoritmo pode ser encontrada no cenário **2.3 ,**slides 30 a 35, com principal destaque para o **33** onde tal implementação é explicada em detalhe.

Conclusão

Com a realização deste projeto foi nos possível compreender em primeira pessoa a importância de uma boa análise de cada problema, pois dita a escolha da abordagem a tomar.

Uma análise incorreta ou mais descuidada pode levar a que o algoritmo implementado ou as variáveis de decisão escolhidas afetem o resultado final.

Pondo em causa a viabilidade e fiabilidade da solução, devido a algoritmos pouco eficientes com o escalar dos dados.

Principais dificuldades

Adaptação dos algoritmos dados na aula para os casos específicos presentes no projeto.

Esforço de cada elemento

Cada elemento trabalhou de forma igual, sendo o esforço dividido equitativamente.