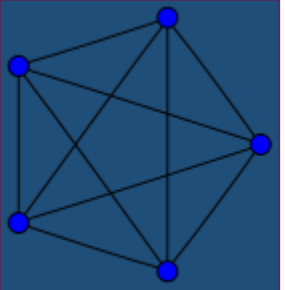
Trabalho prático 2

Desenho de algoritmos

Grupo formado por:

Inês Sousa da Silva, up202008076

Tiago Campos Lourenço,up202004374



Descrição do problema

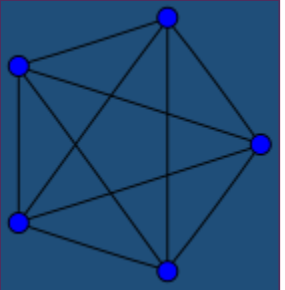
No âmbito da unidade curricular de Desenho de Algoritmos, foi-nos proposta a elaboração de um projeto acerca da logística de uma agência de viagens.

Pretende-se desenvolver uma plataforma digital responsável pela gestão de pedidos para transporte de grupos de pessoas desde de um local origem até um local de destino.

A Empresa dispõe de vários veículos locais, onde cada veículo realiza um trajeto distinto, aportando um diferente número de pessoas, isto é, suponhamos que um veículo A realiza uma viagem do ponto x ao ponto y e que possui uma capacidade limitada a n pessoas.

De entre as várias funcionalidades do sistema é fundamental que a sua implementação permita maximizar a dimensão de um grupo de um local para outro, minimizar o número de transbordos nesse percurso para um grupo que não se separe.

Para grupos que se possam separar, é essencial que na presença de um aumento do número de elementos de um grupo o sistema seja capaz de reajustar o encaminhamento para esse grupo tendo em conta a nova dimensão e que com a informação dos veículos existentes seja capaz de determinar a dimensão máxima do grupo para um percurso definido, entre outras que iremos abordar em pormenor.



Cenário 1-Grupos que não se separam: maximização da dimensão do grupo(1.1)

O objetivo principal deste cenário passa por maximizar a dimensão de um grupo para um dado percurso com origem x e destino y, tendo em conta que o grupo não se pode separar.

Tendo em conta o referido, definimos:

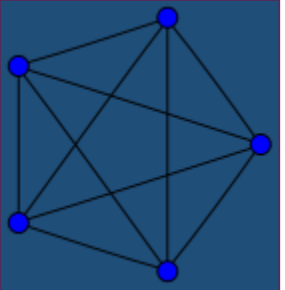
n - número de nós ($n \in \mathbb{N}$)
t - número de arestas /transbordos ($t \in \mathbb{N}$)
start – origem do percurso ($1 \leq start \leq n$)
end – destino do percurso ($1 \leq end \leq n$)

Input:
s – origem da aresta ($\in N$)
e – destino da aresta ($\in N$)
c – capacidade da aresta ($\in N$)

Variável Decisão : $t_{ij} \in \{0,1\}$ -> utilizar/não utilizara aresta de i até j no percurso

Função objetivo : $\max \sum t_{ij} * cij$

Suj. a:
$\forall_i : \Sigma(t_{ij}) \leq 1$, sendo j os nós tal que existe uma aresta de i a j
$\forall t_{jk}; (t_{jk}=1 \Rightarrow \exists_i : t_{ij}=1)$, a menos que j=start
$\exists_i : t_{starti}=1$
$\exists_j : t_{jend}=1$



Cenário 1-Grupos que não se separam: maximização da dimensão do grupo

Para que a otimização do número de elementos do grupo, nomeadamente a sua maximização, fosse realizada de uma forma mais eficiente, baseamos a nossa implementação no algoritmo de Dijkstra, de modo a obtermos uma capacidade(nº elementos) máxima.

Para que este algoritmo pudesse ser utlizado, era necessário um ficheiro de input que continha informação acerca do número de locais e veículos existentes e a respetiva informação de cada veículo(start,end,cap,dur).

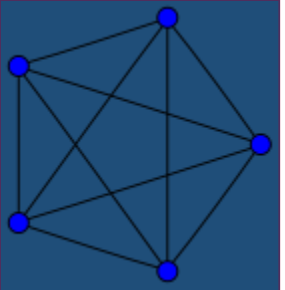
Para auxiliar na preferência de capacidades mais elevadas(objetivo do algoritmo), foi utlizada uma priority queue que armazenava os nós de modo a priorizar os que possuíam uma maior capacidade.

Imaginemos o percurso do nó x até ao nó y(para qualquer x e y pertencentes ao grafo) se o mínimo entre a capacidade do nó x e a aresta de x a y fosse maior que a capacidade de y então a capacidade do nó y era atualizada.

No final o grafo era percorrido do destino para o início selecionando os nós com maior capacidade.

Complexidade temporal : $T(V,E) = O(E*\log|V|)$

Complexidade espacial : $S(V,E) = O(V)$ (memória para a priority queue)



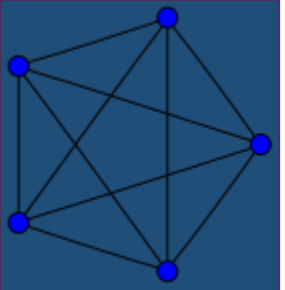
Cenário 1-Grupos que não se separam: maximização da dimensão do grupo

Os resultados obtidos nas tentativas de obter valores cada vez mais otimizados não sofreram muitas variações. No entanto, a adoção de uma priority queue na implementação do algoritmo de Dijkstra, permitiu melhorar a sua eficiência de acordo com os objetivos pretendidos.

Foram testados vários ficheiros de input, nomeadamente um input de tamanho reduzido que permitiu concluir que o valor obtido coincidiu com o esperado.

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

```
1 , 4 , 3 , The maximum capacity of this path is 2
```



Cenário 1-Grupos que não se separam: maximização da dimensão do grupo e minimização do número de transbordos(1.2)

Neste cenário, à semelhança do anterior, pretendia-se maximizar a dimensão de um grupo, acrescentando agora a condição de minimização do número de transbordos.

São apresentadas alternativas(pareto-ótimas), onde um grupo de maior dimensão poder-se-á traduzir em mais transbordos e, pelo contrário, um maior número de transbordos numa menor dimensão.

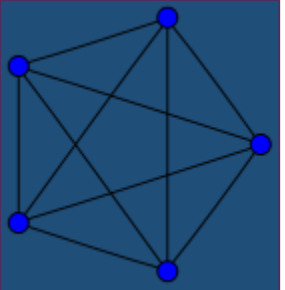
Tendo em conta o referido definimos:

n - número de nós ($n \in \mathbb{N}$)
t - número de arestas /transbordos ($t \in \mathbb{N}$)
start – origem do percurso ($1 \leq start \leq n$)
end – destino do percurso ($1 \leq end \leq n$)

Input:
s – origem da aresta ($\in N$)
e – destino da aresta ($\in N$)
c – capacidade da aresta ($\in N$)

- Variável Decisão : $t_{ij} \in \{0,1\}$ -> utilizar/não utilizara aresta de i até j no percurso
- Função objetivo : $\max_{\min} \frac{\sum t_{ij} * cij}{\Sigma(t_{ij})}$

Suj. a:
$\forall_i : \Sigma(t_{ij}) \leq 1$, sendo j os nós tal que existe uma aresta de i a j
$\forall t_{jk}; (t_{jk}=1 \Rightarrow \exists_i : t_{ij}=1)$, a menos que j=start
$\exists_i : t_{starti}=1$
$\exists_j : t_{jend}=1$



Cenário 1-Grupos que não se separam: maximização da dimensão do grupo e minimização do número de transbordos(1.2)

Para este cenário adotamos o algoritmo de Dijkstra, à semelhança do cenário anterior com o intuito de maximizarmos o número de elementos do grupo. No que respeita à minimização do número de transbordos, utilizamos o BFS(Breath First Search) que devolvia uma estrutura do tipo pair<int,vector<int>> , cujo int correspondia à capacidade do encaminhamento encontrado e vector<int> ao caminho com mínimo overflow.

No final eram devolvidas as duas soluções(pareto-ótimas) não priveligiando, um encaminhamento com capacidade máxima e maior número de transbordos, nem um resultado com um número de transbordos maior mas pelo contrário, uma menor capacidade(menor dimensão).

Complexidade temporal : T(ntransbordosMin,ntransbordosMax,V,A)
= O((ntransbordosMax*A)*log|ntransbordosMax*V| +
(ntransbordosMax-ntransbordosMin)*V^E)

Complexidade espacial: T(ncaminhos,ntransbordos,V,A) =
O(ntransbordos*(V+A) + ncaminhos*V)

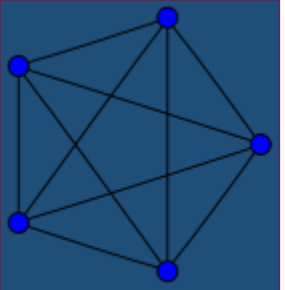
Cenário 1-Grupos que não se separam: maximização da dimensão do grupo e minimização do número de transbordos(1.2)

Os resultados obtidos nas tentativas de obter valores cada vez mais otimizados não sofreram muitas variações.

Foram testados vários ficheiros de input, nomeadamente um input de tamanho reduzido que permitiu concluir que o valor obtido coincidiu com o esperado.

```
11 18
1 2 5 1
1 3 10 5
1 4 5 6
2 5 10 3
3 2 15 7
3 6 20 13
4 7 10 5
5 6 25 8
5 8 10 2
6 4 5 1
6 9 30 2
7 9 5 4
7 10 10 5
9 5 15 6
9 10 5 3
8 11 5 4
9 11 15 1
10 11 10 1
```

```
1 11
Path maximum capacity : 1 , 3 , 6 , 9 , 11 , Capacity: 10
Path minimum overflow : 1 , 2 , 5 , 8 , 11 , Capacity: 5
```



Cenário 2-Grupos que se separam: maximização da dimensão do grupo(2.1)

Neste cenário o principal objetivo, era para uma dimensão de grupo definida, determinar um encaminhamento possível.

Tendo em conta o referido definimos:

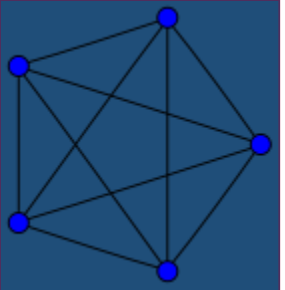
n - número de nós (n ∈ $\mathbb{N}$)
m – dimensão do grupo
t - número de arestas /transbordos (t ∈ $\mathbb{N}$)
start – origem do percurso (1 ≤ start ≤ n)
end – destino do percurso (1 ≤ end ≤ n)

Input:
s – origem da aresta (∈ N)
e – destino da aresta (∈ N)
c – capacidade da aresta (∈ N)

- Variável Decisão : f_{ij} ∈ {0,1} -> fluxo na aresta de i para j
- Função objetivo :

$$\sum f_{ij} = m$$

Suj. a:
$\forall a_{ij}$, $f_{ij} <= c_{ij}$
$\forall_j$: $\Sigma(f_{ij}) = \Sigma(f_{jk})$, exceto se j=dest
f(u,v) = -f(v,u)
((u,v) não pertence E)=> c(u,v)=0
•$\forall a_{jk}$; (t_{jk}=1 => $\exists i: t_{ij}$=1), não se aplicando se j=start
•$\exists_i$:f(start,i)>0
•$\exists_j$:f(j,dest)>0



Cenário 2-Grupos que se separam: maximização da dimensão do grupo

Para que o caminho obtido, dado a sua dimensão, fosse o mais otimizado, baseamos a nossa implementação no algoritmo de Edmonds Karp, de modo a obtermos o caminho mais curto possível.

Para que este algoritmo pudesse ser utlizado era necessário um ficheiro de input que continha informação acerca do número de locais e veículos existentes e a respetiva informação de cada veículo (start, end,cap,dur).

Para auxiliar na procura do caminho mais eficiente utilizamos um multimap<int,vector<int>> que seria igual ao resultado da função maxFlow que determinava o fluxo máximo.

Caso o fluxo máximo fosse maior ou igual a dimensão do grupo, retornaria como output um caminho, caso contrário se o fluxo máximo fosse menor que a dimensão do grupo, nenhum caminho seria retornado pois a capacidade total seria superior a todos os possíveis caminhos desde o inicio até ao fim.

Complexidade temporal : O(V*E^2)

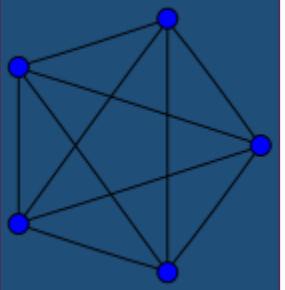
Complexidade espacial: S(V,E) = O(V+E)

Cenário 2-Grupos que se separam: maximização da dimensão do grupo

Os resultados obtidos nas tentativas de obter um caminho cada vez mais otimizados não sofreram muitas variações. Foram testados vários ficheiros de input, nomeadamente um input de tamanho reduzido que permitiu concluir que o caminho obtido coincidiu com o esperado.

```
11 18
1 2 5 1
1 3 10 5
1 4 5 6
2 5 10 3
3 2 15 7
3 6 20 13
4 7 10 5
5 6 25 8
5 8 10 2
6 4 5 1
6 9 30 2
7 9 5 4
7 10 10 5
9 5 15 6
9 10 5 3
8 11 5 4
9 11 15 1
10 11 10 1
```

```
1
1 9
5
5 elements of the group should attend the trip:
1 , 3 , 6 , 9 , with capacity 10
```



Cenário 2-Grupos que se separam : Corrigir um encaminhamento(2.2)

O objetivo deste cenário é, na presença de um aumento do número de elementos de um grupo realizar um reajuste no encaminhamento para esse grupo tendo em conta a nova dimensão.
Tendo em conta o referido é possível definir:

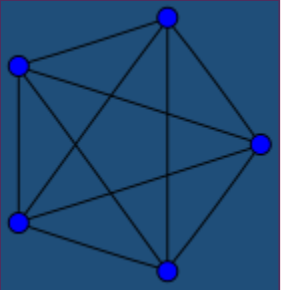
n - número de nós ($n \in \mathbb{N}$)
m – dimensão do grupo
l - número de pessoas a acrescentar
t - número de arestas /transbordos ($t \in \mathbb{N}$)
start – origem do percurso ($1 \leq start \leq n$)
end – destino do percurso ($1 \leq end \leq n$)

Input:
s – origem da aresta ($\in N$)
e – destino da aresta ($\in N$)
c – capacidade da aresta ($\in N$)

- Variável Decisão : $f_{ij} \in \{0,1\}$ -> fluxo na aresta de i a j
- Função objetivo :

$$\sum f_{ij} = m+l$$

Suj. a:
$\forall a_{ij}$, $f_{ij} <= c_{ij}$
$\forall_j$: $\Sigma(f_{ij}) = \Sigma(f_{jk})$, não se aplicando quando j = end
f(u,v) = -f(v,u)
((u,v) não pertence a E)=> c(u,v)=0
$\forall a_{jk}$; ($t_{jk}=1 \Rightarrow \exists_i : t_{ij}=1$) , não se aplicando quando j = start
•$\exists_i$:f(start,i)>0
•$\exists_j$:f(j,end)>0



Cenário 2-Grupos que se separam : Corrigir um encaminhamento(2.2)

Para que a correção do caminho, dado a sua nova dimensão, fosse o mais otimizado, baseamos novamente a nossa implementação no algoritmo Edmonds Karp, de modo a obtermos o caminho mais curto possível.

Para que este algoritmo pudesse ser utlizado era necessário um ficheiro de input que continha informação acerca do número de locais e veículos existentes e a respetiva informação de cada veículo (start, end,cap,dur).

A função path_max_flow_dimension_given do cenário 2.1 foi novamente utilizada. A função calcula novamente o caminho mais curto e o fluxo máximo para a dimensão dada, mas de seguida dá-se como input outra dimensão. Caso a diferença entre a dimensão nova e dimensão velha for menor ou igual ao fluxo máximo, não será necessário alterar o encaminhamento porque os elementos do grupo conseguem entrar na ultima viagem listada. Caso contrário será necessário corrigir o caminho.

Complexidade temporal :T(V,E) = O(V*E^2),

Complexidade espacial: S(V,E) = O(V+E)

Cenário 2-Grupos que se separam : Corrigir um encaminhamento(2.2)

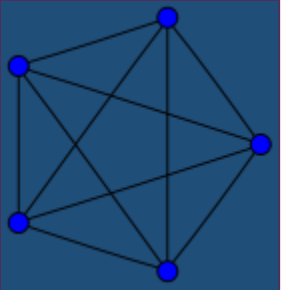
Os resultados obtidos nas tentativas o caminho corrigido mais otimizado não sofreu grandes variações.

Foram testado vários ficheiros de input, nomeadamente um input de tamanho reduzido que permitiu concluir que o valor obtido coincidiu com o esperado.

```
11 18
1 2 5 1
1 3 10 5
1 4 5 6
2 5 10 3
3 2 15 7
3 6 20 13
4 7 10 5
5 6 25 8
5 8 10 2
6 4 5 1
6 9 30 2
7 9 5 4
7 10 10 5
9 5 15 6
9 10 5 3
8 11 5 4
9 11 15 1
10 11 10 1
```

```
2
1 9
4
Este/s é/são o/os caminho/s adequado/s para essa dimensão do grupo:

4 elements of the group should attend the trip:
1 , 3 , 6 , 9 , with capacity 10
Insira agora a nova dimensão do grupo, de modo a corrigir o/os encaminhamento/s anterior/s:

2
Com a nova dimensão de grupo, não é necessário alterar o encaminhamento porque as -2 pessoas novas conseguem entrar na última viagem listada que tem como espaço restante 6.
```

```
1
Este/s é/são o/os caminho/s adequado/s para essa dimensão do grupo:

1 elements of the group should attend the trip:
1 , 3 , 6 , 9 , with capacity 10
Insira agora a nova dimensão do grupo, de modo a corrigir o/os encaminhamento/s anterior/s:

12
10 elements of the group should attend the trip:
1 , 3 , 6 , 9 , with capacity 10
2 elements of the group should attend the trip:
1 , 4 , 7 , 9 , with capacity 5
```



Cenário 2-Grupos que se separam : Determinar a dimensão máxima de um grupo e um encaminhamento(2.3)

Neste cenário, o principal objetivo era maximizar o número de elementos de um grupo e determinar um encaminhamento compatível com essa dimensão. Tratava-se então de um problema de fluxo máximo.

Tendo em conta o referido, é possível definir:

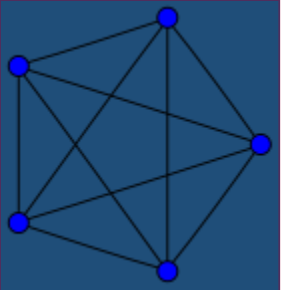
n - número de nós ($n \in \mathbb{N}$)
t - número de arestas /transbordos ($t \in \mathbb{N}$)
start – origem do percurso ($1 \leq start \leq n$)
end – destino do percurso ($1 \leq end \leq n$)

Input:
s – origem da aresta ($\in N$)
e – destino da aresta ($\in N$)
c – capacidade da aresta ($\in N$)

- Variável Decisão : $f_{ij} \in \{0,1\}$ -> fluxo na aresta de i a j
- Função objetivo :

$$\Sigma_{fiend}$$

Suj. a:
$\exists_i$:f(start,i)>0
$\exists_j$:f(j,end)>0
$\forall t_{ij}$, $f_{ij} <= c_{ij}$
$\forall_j$: $\Sigma(f_{ij})=\Sigma(f_{jk})$, excluindo j=end
(u,v) $\notin$ E e por isso c(u,v)=0
f(u,v) = -f(v,u)
$\forall a_{jk}$; (a_{jk}=1 => $\exists_i$:a_{ij}=1), excluindo j= start



Cenário 2-Grupos que se separam : Determinar a dimensão máxima de um grupo e um encaminhamento(2.3)

A abordagem deste cenário, passou pela criação de um multimap<int,vector<int>> associado ao resultado da função maxFlow , que como o próprio nome indica, determina o fluxo máximo para um encaminhamento. O multimap associa o fluxo máximo a um percurso, e sendo que, diferentes percursos podem ter igual fluxo optamos por esta estrutura.

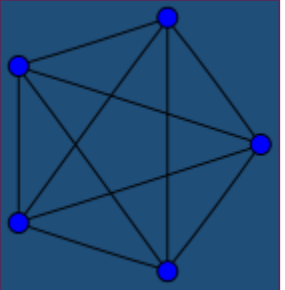
Após termos armazenado o resultado da chamada da função maxflow no multimap, percorríamos a estrutura com um iterador, guardando numa variável max a soma do fluxo máximo de todos os encaminhamentos.

A função path_max_flow era chamada com os parâmetros start e end lidos do input e com a dimensão correspondente à soma e todos os fluxos máximos determinados.

No final o caminho para a dimensão atribuída é gerado e mostrado no output.

Complexidade temporal : O(V*E^2)

Complexidade espacial: S(V,E) = O(V+E)



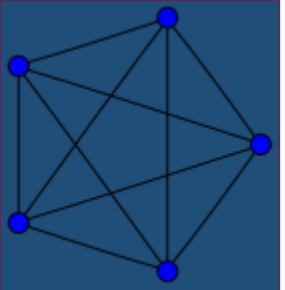
Cenário 2-Grupos que se separam : Determinar a dimensão máxima de um grupo e um encaminhamento(2.3)

Os resultados obtidos nas tentativas de otimizar o caminho e a dimensão máxima do grupo, novamente não sofreram grandes variações.

Foram testados vários inputs, nomeadamente um input de tamanho reduzido que permitiu concluir que o valor obtidos coincidiu com o esperado.

```
11 18
1 2 5 1
1 3 10 5
1 4 5 6
2 5 10 3
3 2 15 7
3 6 20 13
4 7 10 5
5 6 25 8
5 8 10 2
6 4 5 1
6 9 30 2
7 9 5 4
7 10 10 5
9 5 15 6
9 10 5 3
8 11 5 4
9 11 15 1
10 11 10 1
```

```
1 9
Dimensão máxima : 20
The group should be divided by their following trips:
Path 1: 1 , 4 , 7 , 9 ,  with capacity 5
Path 2: 1 , 2 , 5 , 6 , 9 ,  with capacity 5
Path 3: 1 , 3 , 6 , 9 ,  with capacity 10
```



Cenário 2-Grupos que se separam : determinar quando é que o grupo se reuniria novamente no destino (2.4)

No cenário 2.4, dado um encaminhamento para um grupo, cujos elementos se podem separar, era pretendido calcular o tempo, mínimo, onde esses elementos se voltariam a encontrar no destino.

Tendo em conta o referido definimos:

start – a origem do caminho ($1 \leq start \leq n$)
dest – o destino do caminho ($1 \leq end \leq n$)
$d_1, d_2, d_3 \ldots, d_n$ - duração das viagens 1, 2, …, n ($d_i \in \mathbb{N}$)
P - relação de precedência entre viagens

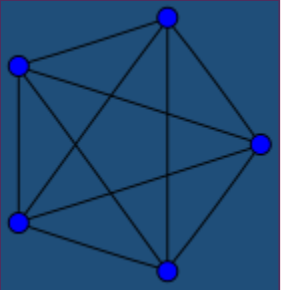
Input:
s – origem da aresta ($\in N$)
e – destino da aresta ($\in N$)
c – capacidade da aresta ($\in N$)

- Variável Decisão :

s: instante em que o grupo se volta a reunir ($s \in \mathbb{N}$)
$m_1, m_2, \ldots, m_n$ – data de início das viagens 1, 2, …, n ($m_i \in \mathbb{N}$)

- Função objetivo : $\min \; s$

Suj. a:
$m_i + d_i \leq m_j, \forall (i,j) \in R$
$m_i + d_i \leq s$, para todo o $i \in$ n viagens



Cenário 2-Grupos que se separam : determinar quando é que o grupo se reuniria novamente no destino (2.4)

Na implementação deste cenário foi utilizado o algoritmo Critical Path Method(CPM), que nos permite determinar o tempo mínimo necessário para que todos os elementos de um grupo se voltem a encontrar no destino .

Começamos por atualizar o grau de Entrada de todos os nós ou seja quantos veículos passam por o local de um nó x.

De seguida, numa queue armazenamos os nós cujo grau de entrada era nulo.

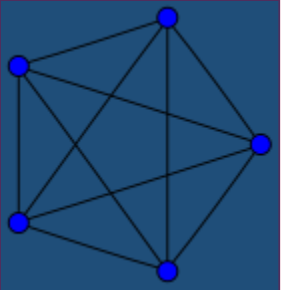
Para determinarmos a duração mínima(variável inicializada a -1) percorremos o conteúdo da queue e só atualizávamos o seu valor se o Early Start do nó removido da queue fosse menor que a duração mínima atual.

Para além disso, percorremos o nós adjacentes ao nó cujo early start era menor que a duração mínima atual e se o Early Start do nó adjacente fosse menor que a soma da duração do ramo e o e que o Early Start do nó origem, atualizávamos o Early start do nó adjacente e decrementávamos o grau de entrada.

Se o grau de entrada do nó adjacente fosse nulo adicionávamos o nó à queue.

No final era retornada a diferença entre o Early start do nó destino e o Early Start do nó origem.

Complexidade temporal : = O(V*E^2 + E) = O(V*E^2)
Complexidade espacial: = O(V+V)=O(V)



Cenário 2-Grupos que se separam : determinar quando é que o grupo se reuniria novamente no destino (2.4)

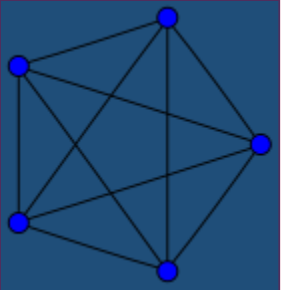
Os resultados obtidos nas tentativas de obter valores cada vez mais otimizados não sofreram muitas variações.

Foram testados vários ficheiros nomeadamente um input de tamanho reduzido que permitiu concluir que o valor obtido coincidiu com o esperado.

```
11 18
1 2 5 1
1 3 10 5
1 4 5 6
2 5 10 3
3 2 15 7
3 6 20 13
4 7 10 5
5 6 25 8
5 8 10 2
6 4 5 1
6 9 30 2
7 9 5 4
7 10 10 5
9 5 15 6
9 10 5 3
8 11 5 4
9 11 15 1
10 11 10 1
```

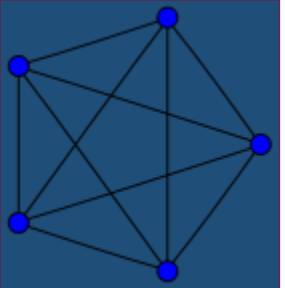
```
1 9
0 grupo reunir-se-à no destino após 20 horas
```

```
2 7
0 grupo reunir-se-à no destino após 12 horas
```



Algoritmos que merecem destaque

Dentro de todos os possíveis algoritmos a escolher, decidimos destacar o algoritmo de Edmonds-Karp, que como visto anteriormente foi o algoritmo mais utilizado e mais eficiente na resolução de grande dos problemas propostos.



Principais dificuldades

Durante a realização do trabalho foram surgindo algumas dificuldades, nomeadamente no que toca à adaptação do algoritmo ao cenário em questão.

Não foi possível concluir o cenário 2.5) devido aos resultados não estarem em concordância como expectável, no entanto, todos os outros cenários foram implementados com sucesso.

No que toca ao esforço, consideramos que foi repartido equitatativamente.