

---

# CONCEÇÃO E ANÁLISE DE ALGORITMOS

---

Gestão de uma companhia de viagens

[TURMA 2MIEIC05](#)

# Índice

|   |    |
|---|----|
| Descrição do Problema .....                                       | 3  |
| O cliente especifica a origem e destino específicos .....         | 3  |
| O cliente especifica grupos de cidades que pretende visitar ..... | 3  |
| Formalização do Problema .....                                    | 4  |
| Dados de Entrada.....   | 4  |
| Dados de Saída.....   | 5  |
| Restrições.....   | 5  |
| Função Objetivo .....   | 5  |
| Descrição da Solução .....  | 6  |
| Estrutura de Dados .....  | 6  |
| Técnicas de Conceção .....  | 7  |
| Principais Algoritmos .....                                       | 7  |
| Algoritmo de Dijkstra .....                                       | 7  |
| Algoritmo de Floyd-Warshall .....                                 | 10 |
| Casos de Utilização .....   | 12 |
| Classes.....  | 12 |
| Ficheiros.....  | 12 |
| Programa.....   | 12 |
| Principais Dificuldades .....                                     | 13 |
| Conclusões .....  | 14 |
| Referências.....  | 15 |

## Descrição do Problema

Uma agência de viagens pretende implementar um sistema que lhe permita conseguir uma seleção mais criteriosa de uma solução de acordo com as necessidades do seu cliente, procurando sempre minimizar o seu custo.

Esta agência oferece viagens, tendo estas como componentes o voo e a estadia. O problema que pretendemos resolver é obter, para os desejos específicos de cada cliente, a opção de custo menor.

Existem três casos possíveis para as especificações do cliente, e como tal a explicação destes será dividida nas suas respetivas secções do relatório.

### O cliente especifica a origem e destino específicos

Nesta primeira opção, é permitido ao cliente escolher a origem e o destino diretamente, o que torna a operação com os algoritmos desenvolvidos em algo direto – basta encontrar os vértices iniciais e finais e, a partir daí, encontrar o caminho com o menor custo.

### O cliente especifica grupos de cidades que pretende visitar

A segunda opção, um pouco mais complexa que a primeira, permite ao cliente escolher vários destinos e implica que seja o nosso sistema a determinar qual o caminho de menor custo entre todos estes vértices. Para tal, não pode apenas ser reutilizado o algoritmo da primeira opção – afinal, encontrar o caminho mais barato entre cada opção na ordem dada pelo cliente pode não levar ao custo final mais barato.

É necessário desenvolver um algoritmo de Dijkstra modificado, permitindo obter assim um conjunto dos caminhos de menor custo.

# Formalização do Problema

## Dados de Entrada

### **Lendo do ficheiro dos clientes (./assets/Clients.txt):**

clientName – o nome do cliente

id – numero que identifica cada cliente

cellphone – numero de telemóvel do cliente

### **Lendo do ficheiro de cada cidade (./assets/Cities/(numero).txt):**

Hotels – vetor com os hotéis de cada cidade, compostos por:

- hotelName – o nome do hotel
- price – o preço da estadia

numberDestinies – o numero de destinos saindo desta cidade

Destinations – vetor com o ID de cada destino que sai da cidade

### **Lendo do ficheiro de cada viagem (./assets/Trips.txt):**

Id – o ID da viagem

Date1 – a data de início da viagem

Date2 – a data de fim da viagem

departureCity – o nome da cidade da partida

arrivalCity – o nome da cidade de chegada

hotelName – o nome do hotel da estadia

cost – o custo da viagem

distance – a distancia total viajada

### **$G_i = (V_i, E_i)$ - grafo dirigido, composto por:**

V - vértices (que representam cidades, descritas em cima) com:

E - arestas (que representam voos) com:

- w - peso da aresta (representa a distância entre os dois vértices que a delimitam)
- ID - identificador único de uma aresta
- dest  $\in V_i$  - vértice de destino

$S \in V_i$  - vértice inicial (cidade de partida)  $T \subseteq V_i$  - vértices finais (cidade de chegada)

### Dados de Saída

Dados de saída  $G_f = (V_f, E_f)$  grafo dirigido, tendo  $V_f$  e  $E_f$  os mesmos atributos que  $V_i$  e  $E_i$ .

$C_f$  - sequência ordenada de todas as cidades, sendo  $C_f(i)$  o seu  $i$ -ésimo elemento.

Cada um tem os seguintes valores:

- preço – custo associado à viagem
- $P = \{e \in E_i \mid 1 \leq j \leq |P|\}$  - sequência ordenada de arestas a visitar, sendo  $e_j$  o seu  $j$ -ésimo elemento.

### Restrições

- O `numberDestinies` tem de ser igual ao número de destinos saindo de cada cidade
- Todos os inteiros utilizados têm de ser números não-negativos.
- O custo, a distância, e o número de telemóvel têm de ser todos superiores a zero
- `departureCity` e `arrivalCity` têm de ser diferentes  
(acrescentar mais)

### Função Objetivo

**(copiado do relatório do ano passado e adaptado, sugeria alterar, ver se está bem)**

A solução ótima do problema passa por minimizar o número de cidades atravessadas e a distância total percorrida, de modo a que assim se chegue ao destino indicado com o menor custo possível. Logo, a solução ótima passa pela minimização das duas respetivas funções:  $f = |C|$

$$g = \sum_{c \in C} [w(e)] \quad e \in P$$

Tal como referido na descrição do problema, irá ser privilegiada a minimização da função  $f$  sobre a da função

# Descrição da Solução

## Estrutura de Dados

De modo a poder representar a informação e obter os resultados pretendidos da maneira mais rápida e eficiente possível, servimo-nos de grafos, uma estrutura de dados que temos definidos no ficheiro *Graph.h*, tendo sido este fornecido pelos docentes e posteriormente alterado por nós, tendo efetuado bastantes adições e alterações.

No ficheiro em questão temos representadas três classes essenciais ao funcionamento de um grafo – as classes template “Vertex”, “Edge” e “Graph”, respetivamente representando vértices, ligações entre vértices e o grafo em si.

Nas classes em questão, para além da informação importante guardada acerca do funcionamento do grafo, temos também dados importantes para a resolução do problema.

Na classe Vertex, temos:

- Info: o conteúdo do vértice
- Adj: um vector de Edges adjacentes ao Vertex
- Dados auxiliares que servem para determinar o caminho mais rápido, como visited ou dist

Na classe Edge, temos:

- Info: o conteúdo da aresta
- Dest: o vértice de destino
- Weight: o peso quando utilizado para realizar o algoritmo de Dijkstra

Um grafo genérico, em si, contém um vector de nome vertexSet, de apontadores para todos os vértices do grafo. Esta estrutura de dados usa como chave o valor da informação, tal como descrito em cima.

Para uma gestão mais eficiente do trabalho como um todo, servimo-nos da classe “Agencia” onde é criado o grafo de que nos iremos servir ao longo do trabalho. É nesta classe que, para além de interface, métodos de leitura e escrita para ficheiros, temos guardadas as estruturas de dados essenciais para o trabalho – vetores com clientes, trips, cidades, e o grafo.

Quanto ao grafo em si, este é manipulado através de métodos introduzidos na sua classe Graph em Graph.h.

## Técnicas de Conceção

(a fazer)

### Principais Algoritmos

#### Algoritmo de Dijkstra

O Algoritmo de Dijkstra (E.W. Dijkstra) é um dos algoritmos que calcula o caminho de custo mínimo entre vértices de um grafo. Escolhido um vértice como raiz da busca, este algoritmo calcula o custo mínimo deste vértice para todos os demais vértices do grafo. Ele é bastante simples e com um bom nível de performance. Ele não garante, contudo, a exatidão da solução caso haja a presença de arcos com valores negativos.

Este algoritmo parte de uma estimativa inicial para o custo mínimo e vai sucessivamente ajustando esta estimativa. Ele considera que um vértice estará fechado quando já tiver sido obtido um caminho de custo mínimo do vértice tomado como raiz da busca até ele. Caso contrário ele dito estar aberto.

Algoritmo: Seja  $G(V,A)$  um grafo orientado e  $s$  um vértice de  $G$ :

Atribua valor zero à estimativa do custo mínimo do vértice  $s$  (a raiz da busca) e infinito às demais estimativas;

Atribua um valor qualquer aos precedentes (o precedente de um vértice  $t$  é o vértice que precede  $t$  no caminho de custo mínimo de  $s$  para  $t$ );

Enquanto houver algum vértice aberto:

- seja  $k$  um vértice ainda aberto cuja estimativa seja a menor dentre todos os vértices abertos;
- feche o vértice  $k$ ;
- Para todo vértice  $j$  ainda aberto que seja sucessor de  $k$  faça;
- some a estimativa do vértice  $k$  com o custo do arco que une  $k$  a  $j$ ;
- caso esta soma seja melhor que a estimativa anterior para o vértice  $j$ , substitua-a e anote  $k$  como precedente de  $j$ .

A sequência de diagramas seguinte ilustra o seu funcionamento:

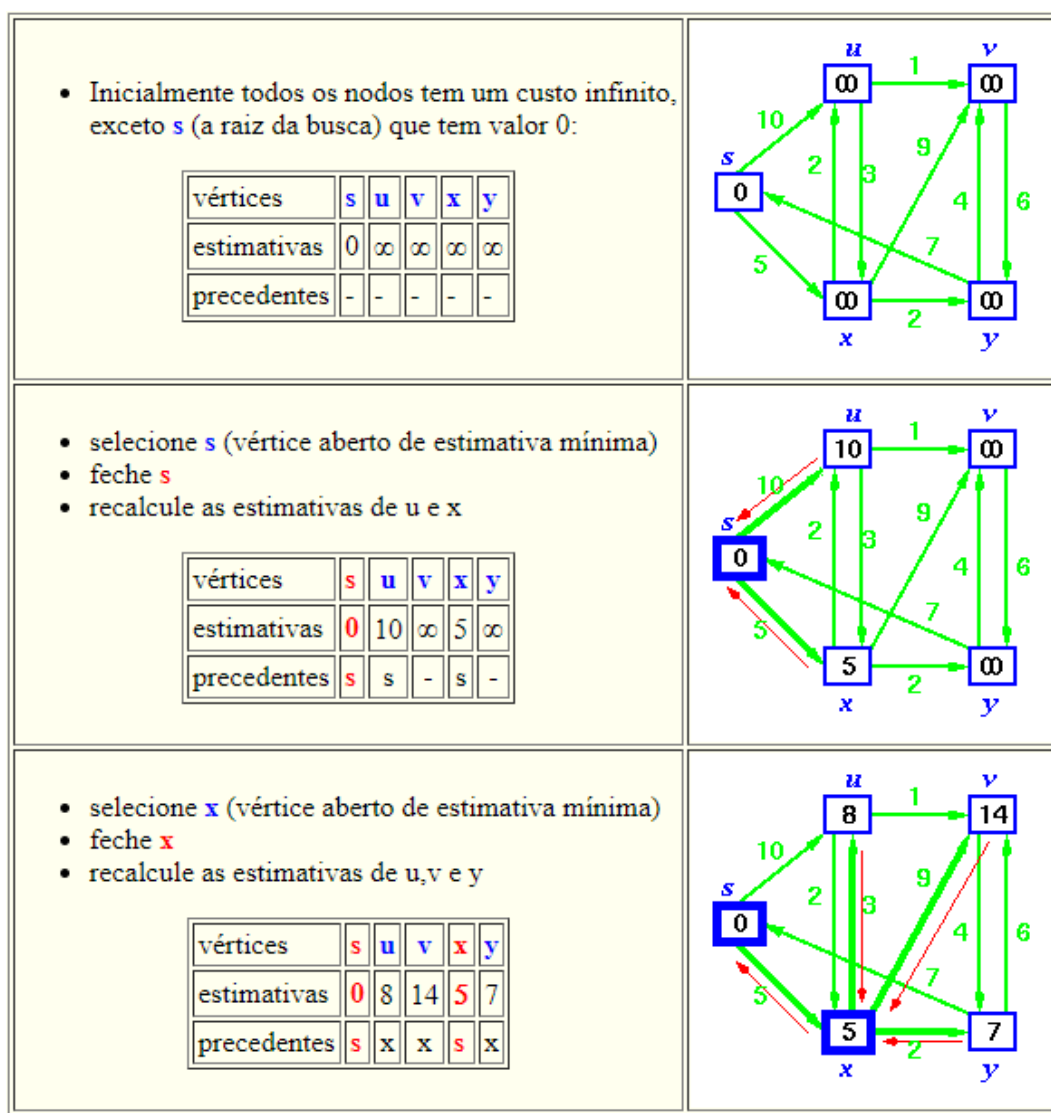


Figura 1 - Demonstração do Algoritmo de Dijkstra (parte 1 de 2)



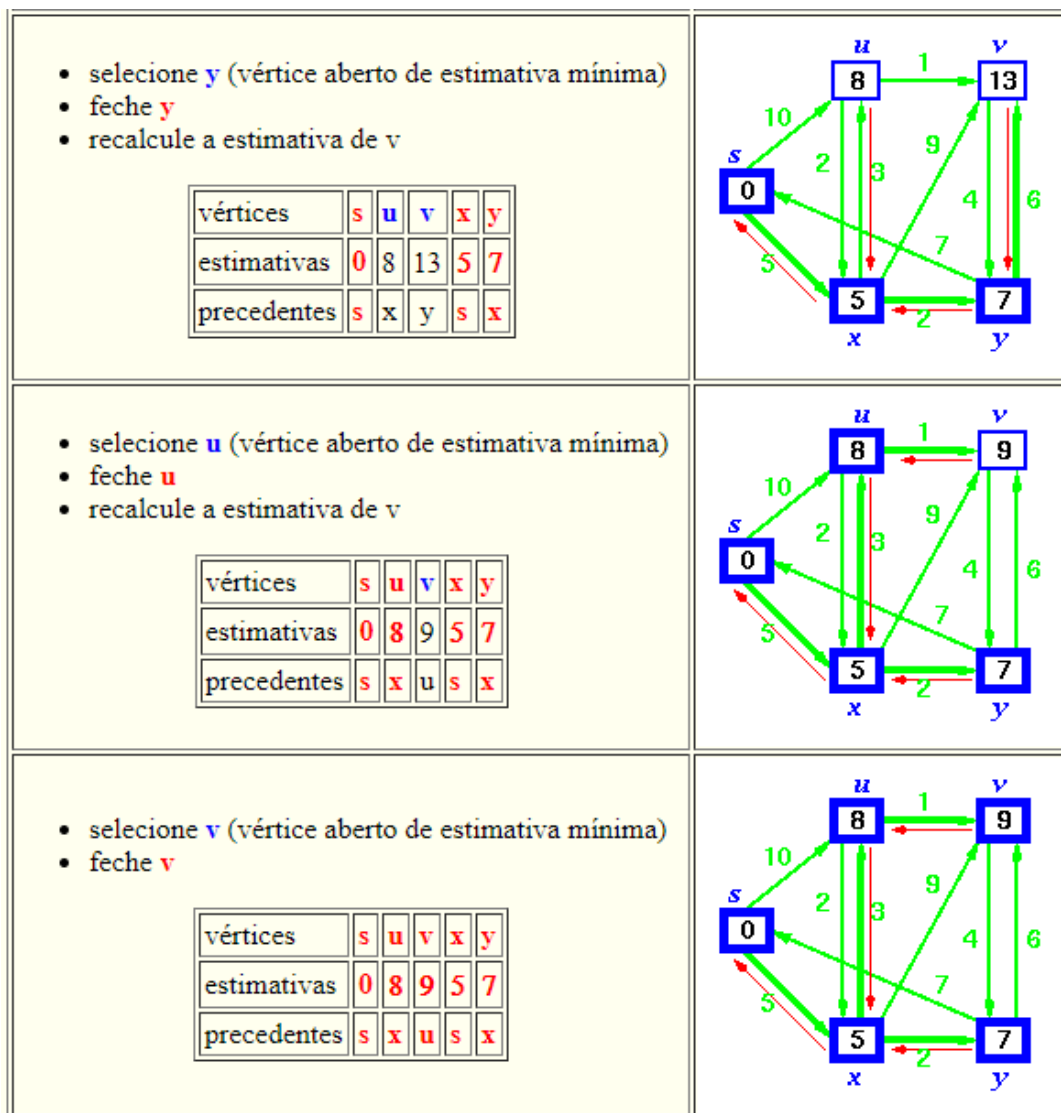


Figura 2 - Demonstração do Algoritmo de Dijkstra (parte 2 de 2)

Quando todos os vértices tiverem sido fechados, os valores obtidos serão os custos mínimos dos caminhos que partem do vértice tomado como raiz da busca até os demais vértices do grafo. O caminho propriamente dito é obtido a partir dos vértices chamados acima de precedentes.

Como apresentado, o algoritmo de Dijkstra computa apenas um único caminho de custo mínimo entre um dado par de vértices. Para se obter todos os caminhos de custo mínimo entre dois vértices é necessário modificar a forma de anotação dos precedentes. A modificação no passo 3 indicada a seguir é suficiente para permitir o cômputo de todos os caminhos por um processo similar ao descrito acima.

Um modo de implementar o algoritmo de Dijkstra modificado é o que nos foi exemplificado no material disponibilizado pelos docentes, nomeadamente:

## Algoritmo de Dijkstra (modificado)

14

```
1. void Dijkstra(Vertex s) {
2.   for (Vertex v : vertexSet) {v.path = null; v.dist = INFINITY;}
3.   s.dist = 0;
4.   PriorityQueue<Vertex> q = new PriorityQueue<Vertex>();
5.   q.insert(s);
6.   while ( ! q.isEmpty() ) { Não serve versão da biblioteca do Java
7.     Vertex v = q.extractMin(); // remove vértice com dist mínimo
8.     for(Edge e: v.adj) {
9.       Vertex w = e.dest;
10.      if (v.dist + e.weight < w.dist) {
11.        w.dist = v.dist + e.weight;
12.        w.path = v;
13.        if (w.queueIndex == -1) // w ∉ q (ver a seguir)
14.          q.insert(w);
15.      } else
16.        q.decreaseKey(w); Assumindo que as arestas não
                            têm pesos negativos, equivale a
                            testar que w.dist antes da
                            atualização era INFINITY
17.      }
18.    }
19.  }
20. }
```

$O(|E| \log |V|)$

Algoritmos em Grafos: Caminho mais curto • CAL - MIEIC/FEUP, Março de 2011

Esta versão do algoritmo é-nos especialmente útil para a segunda parte do nosso problema, que exigia que encontrássemos vários caminhos de menor custo entre dois pontos.

### Algoritmo de Floyd-Warshall

Como alternativa ao algoritmo de Dijkstra, temos também o algoritmo de Floyd-Warshall, que recebe como entrada uma representante do grafo. Este algoritmo calcula, para cada par de vértices, o menor de todos os caminhos entre os vértices – no nosso caso, o caminho de menor custo. Este algoritmo é mais eficiente para grafos densos, enquanto que o algoritmo de Dijkstra é a melhor escolha para grafos mais esparsos. Mesmo em grafos pouco densos pode ser melhor porque código o é mais simples e utiliza programação dinâmica, mas depende em cada caso. Usa matriz de adjacências com pesos (tendo valor infinito quando não há aresta e valor de 0 quando no eixo da matrix). O valor de um caminho entre dois vértices é a soma dos valores de todas as arestas ao longo desse caminho.

Aqui fica um exemplo deste algoritmo em pseudo-código:

### Algoritmo de Floyd-Warshall

Floyd-Warshall( $G$ ):

Seja  $dist[][]$  uma matriz  $|V| \times |V|$  inicializada com  $\infty$

**Para** cada vértice  $v$  de  $G$  **fazer**:

$dist[v][v] \leftarrow 0$

**Para** todas as arestas  $(u, v)$  de  $G$  **fazer**:

$dist[u][v] \leftarrow peso(u, v)$

**Para**  $k \leftarrow 1$  até  $|V|$  **fazer**:

**Para**  $i \leftarrow 1$  até  $|V|$  **fazer**:

**Para**  $j \leftarrow 1$  até  $|V|$  **fazer**:

**Se**  $dist[i][k] + dist[k][j] < dist[i][j]$  **então**

$dist[i][j] \leftarrow dist[i][k] + dist[k][j]$

- A complexidade é trivialmente  $O(|V|^3)$  - ver os 3 ciclos!
-

# Casos de Utilização

## Classes

**Agencia:** singleton class (instância da mesma limitada a um objeto), a classe criada quando se coloca o projeto a correr e que guarda não só a informação sobre o programa mas também os métodos deste;

**City:** classe representativa de uma cidade, sendo objetos desta classe nós para o nosso grafo;

**Client:** classe representativa de um cliente, com a informação pessoal deste e as viagens marcadas;

**Date:** classe representativa da data;

**Hotel:** classe representativa de um hotel, guardando a informação e métodos a esta associadas;

**Season:** classe representativa de uma temporada, guardando a informação sobre cada época (nomeadamente, preços);

**Trip:** classe representativa de uma viagem, guardando métodos a esta associados e os voos e hotéis respetivos.

## Ficheiros

**CitiesNames:** guarda nomes de cidades que estão registadas no sistema;

**Clients:** guarda informação dos clientes da agência;

**Trips:** guarda informações sobre as viagens marcadas;

**Pasta “Cities”:** informação sobre todas as cidade disponíveis para viajar, sendo que cada ficheiro representa uma cidade.

## Programa

Primeiramente, o programa lê e guarda a informação dos ficheiros de texto, utilizando-a para criar o grafo. Após isso, é apresentado um menu ao utilizador que lhe permite adicionar ou apagar clientes e viagens, marcar viagens e visualizar todos os percursos de viagens marcadas.

Ao marcar uma viagem, o cliente pode escolher dois locais – a origem e o destino – ou a origem e uma série de destinos por onde visita sem qualquer ordem em específico.

## Principais Dificuldades

Ao longo do desenvolvimento do projeto, a principal dificuldade com que nos deparamos, de longe, foi coordenarmo-nos como grupo. Esta dificuldade atrasou bastante todo o processo de desenvolvimento e entrega do trabalho, já que era frequente os outros membros do grupo ficarem dependentes do trabalho que estava a ser desenvolvido por apenas um e, em vez de poderem trabalhar em paralelo, terem de aguardar (não que tenha havido só um membro do grupo a atrasar o trabalho, isto ocorreu em várias ocasiões com membros diferentes).

Outra dificuldade foi uma impossibilidade de um dos membros que ficou sem acesso a um computador pessoal e, para trabalhar, teve sempre de recorrer aos da FEUP, reduzindo as opções que tínhamos para trabalhar.

Por último, termos tido bastantes projetos em simultâneo com outras cadeiras levou, devido a algumas falhas na gestão de tempo, não termos trabalhado tanto neste projeto quanto desejávamos antes da data de entrega.

## Conclusões

A proposta de trabalho foi eficaz ao exigir vários tipos de estruturação para as viagens, forçando-nos a usar algoritmos diversos dependendo da necessidade para cada situação. Deste modo, ao trabalhar diretamente com os algoritmos e perceber não só o que conseguem e não conseguem fazer, mas também a eficiência com que concluem determinadas tarefas, permite-nos ganhar bastante experiência e conhecimento sobre este meio.

Para além disso, a dinâmica de grupo que foi crescendo ao longo do desenvolvimento do trabalho permitiu que todos melhorássemos as nossas capacidades de comunicação e de coordenação com os restantes membros.

Por fim, podemos concluir afirmando que a aplicação prática dos conteúdos lecionados num trabalho com maior dimensão do que os exercícios das aulas práticas foi, até agora, o método com melhores resultados a fazer cada membro do grupo dominar os temas da unidade curricular.

## Referências

<http://www.inf.ufsc.br/grafos/temas/custo-minimo/dijkstra.html> - explicação do Algoritmo de Dijkstra por docentes do Departamento de Informática e Estatística da Universidade Federal de Santa Catarina

[http://www.dcc.fc.up.pt/~pribeiro/aulas/daa1415/slides/8\\_distancias\\_06122014.pdf](http://www.dcc.fc.up.pt/~pribeiro/aulas/daa1415/slides/8_distancias_06122014.pdf) - explicação do Algoritmo de Floyd-Warshall por Pedro Ribeiro, docente do Departamento de Ciência de Computadores na Faculdade de Ciências da Universidade do Porto

Rosse, Rosaldo, and Ana Paula Rocha. 2015/2016. "Algoritmos Em Grafos: Caminho Mais Curto