Relatório do projeto de Conceção e Análise de Algoritmos

Trabalho 1

Tema 5: Planeamento de férias

Turma 2 Grupo F:

Joel Márcio Torres Carneiro 201100775 ei11053@fe.up.pt

Daniel Arménio Silva Mendonça 200906506 ei12167@fe.up.pt

Francisco Teixeira Lopes 201106912 ei11056@fe.up.pt

Breve descrição da aplicação:

Uma pessoa pretende planear um circuito turístico em automóvel para as férias em família em regime de *fly and drive*, visitando um conjunto de locais de interesse turístico, começando e terminando numa determinada cidade (com aeroporto próximo).

Para cada local de interesse a visitar, sabe-se a localização e a duração recomendada de cada visita (horas). Existe também uma lista de locais possíveis de alojamento (que podem ser ou não locais de interesse). Está também definido um número máximo de horas por dia que se pretendem gastar em viagem e/ou em visita aos locais de interesse (por exemplo, 10 horas).

Pretende-se gerar um plano de viagem de duração mínima, com a sequência de locais (locais de interesse ou locais de alojamento) e dias (1º dia, 2º dia, etc.).

Numa primeira versão, determinou-se um percurso de duração total mínima, que comece e acabe na cidade especificada e passe pelo menos uma vez em cada local de interesse. Melhorou-se depois o programa para encontrar uma solução ótima de acordo com os objetivos propostos. No caso de não ser viável encontrar a solução ótima num tempo razoável, pode obter uma solução aproximada.

Formalização:

G = (V,A), V é o Set de N locais (todos os locais) a serem visitados. A é o set de caminhos possíveis de percorrer.

D = (C_{ij}) é o custo em horas associado a cada percurso i para j. (duração)

$\mathbf{C_v}$ = (C_j) é o custo em horas da visita à cidade j, se esta ainda não foi visitada.

M é o número máximo de horas diárias disponíveis.

P é o local atual.

$\mathbf{U_d}$ é o número de horas gastas no dia.

N é o número de dias acumulados.

R é uma cidade com aeroporto.

$\mathbf{X_j}$ é 1 se a cidade ainda não foi visitada e é de visita obrigatória, 0 para outros casos.

Dados de entrada:

$C_{a.}$

$C_{v.}$

G

Dados de saída:

Lista dos locais visitados

Objetivo:

Minimizar $\sum_{(d)}[\ \sum_{(i,\ j)} (D_{ij} + D_j)\ i \leftarrow j]$

Restrições:

Assegurar que a primeira viagem é feita a partir de uma cidade com aeroporto.

$(U_d=0, D=0) \rightarrow P_j = R.$

Assegurar que a última posição é um local com aeroporto.

$\sum_{(i, j)} X_j = 0 \rightarrow P_j = R.$

Garantir que não são ultrapassadas o tempo máximo diário disponível.

$M - U_d \geq 0.$

Descrição da solução encontrada:

Para a resolução deste problema o grupo decidiu implementar 6 classes: City, Vertex, Edge, EdgeType, GraphViewer e Graph.

City:

Nesta classe a cidade é "criada". Cada cidade tem um nome, um tempo de visita, se tem aeroporto e/ou alojamento ou não e o tempo de viagem. É possível aceder a cada uma das especificações através de funções criadas na aplicação.

Vertex:

Nesta classe, através das propriedades do grafo, e das funções então criadas, é possível saber-se se uma cidade já foi visitada e assinala-la como tal.

Edge:

Nesta classe é possível saber-se os tempos de viagem de umas cidades para as outras. Desta forma é possível analisar-se qual a cidade mais próxima.

Graph:

Aqui são reunidos todos os resultados das restantes classes e tratados de forma a ser possível obter uma solução do problema.

EdgeType:

Nesta classe são enumerados os tipos de arestas. Existem então dois tipos: arestas com e sem direção.

GraphViewer:

Esta classe trata de representar um grafo criado previamente. Todas as suas funções retornam um booleano a indicar se a sua execução decorreu ou não com sucesso.

Para a obtenção de uma solução foi necessário utilizar algoritmos que nos ajudassem a calcular o pretendido. Desta forma foi usado o algoritmo Floyd-Warshall para calcular o menor caminho entre cidades através de uma matriz recebida como entrada.

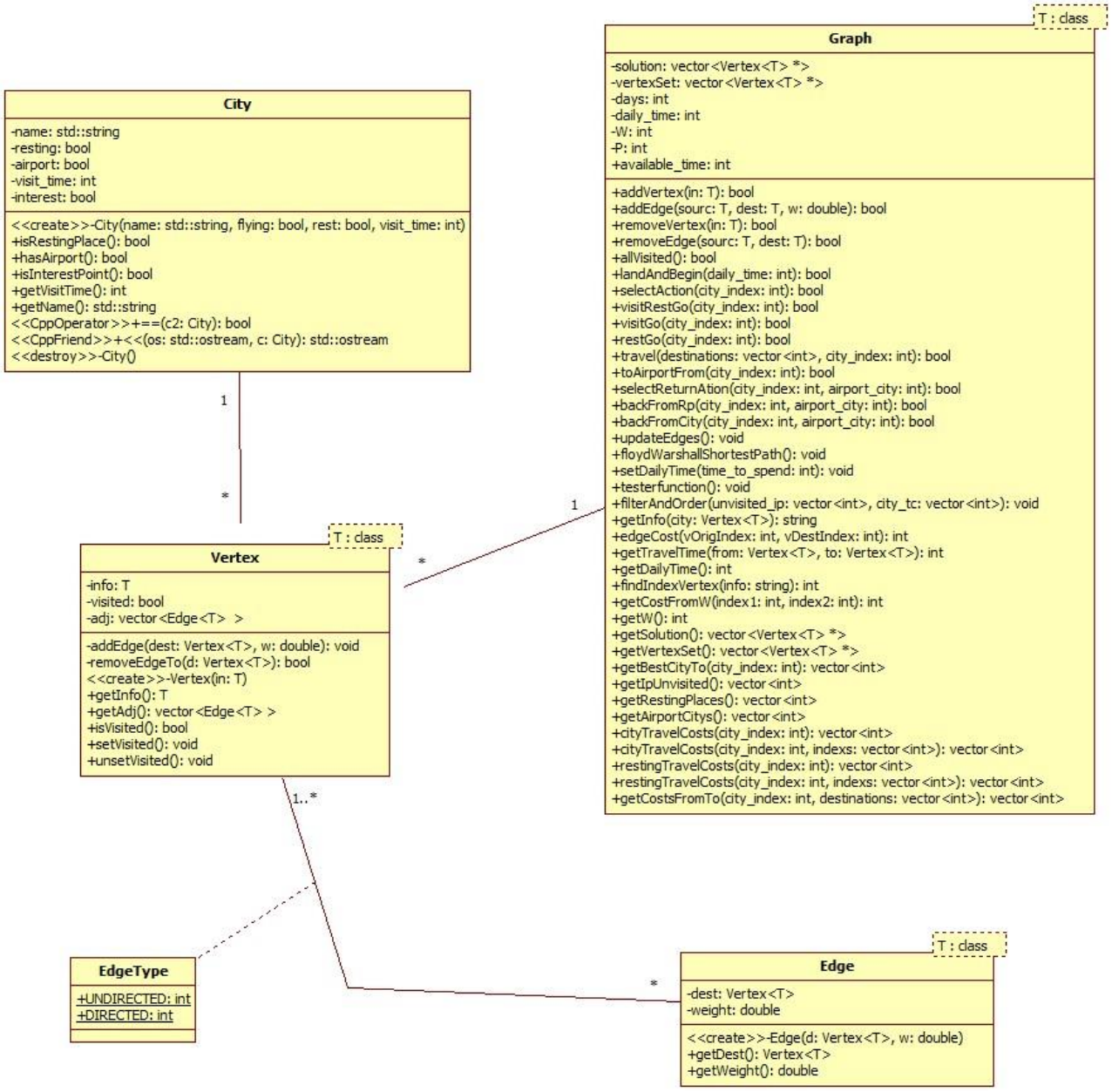
O algoritmo de Floyd-Warshall recebe como entrada uma matriz de adjacência que representa um grafo orientado e valorado. O valor de um caminho entre dois vértices é a soma dos valores de todas as arestas ao longo desse caminho. As arestas do grafo podem ter valores negativos, mas o grafo não pode conter nenhum ciclo de valor negativo. O algoritmo calcula, para cada par de vértices, o menor de todos os caminhos entre os vértices. Por exemplo, o caminho de menor custo. A ordem de complexidade é $\theta(|V|^3)$.

Foi também utilizada uma abordagem greedy para obter uma melhor solução. Um algoritmo greedy é um algoritmo que segue a resolução de problemas heurísticos de forma a fazer a escolha ideal localmente em cada fase, com a esperança de encontrar um valor ótimo global.

Em muitos dos problemas as estratégias gananciosas não produzem soluções ideais sendo então aproximadas. Ainda assim, uma estratégia gananciosa pode produzir soluções localmente ótimas que se aproximam de uma solução ótima global num tempo razoável.

Desta forma, foi necessário a utilização destas abordagens. Para obtermos uma solução em pouco tempo só mesmo utilizando um método ganancioso o torna possível. Queremos que seja relativamente rápida a solução, caso contrário seria demorado encontrar a solução ótima no meio de tantas possíveis pois teriam de ser analisados todos os caminhos. No caso de umas férias pequenas com poucos locais de destino seria viável mas se fosse um número maior de cidades começava a tornar-se muito demorado para que se obtivesse a solução.

Diagrama de Classes UML



Lista de Casos de Utilização Identificados para a Aplicação

1. Criar grafo
2. Criar cidade
3. Criar nós
4. Atribuir custo
5. Atribuir tempo de viagem
6. Atribuir alojamento
7. Atribuir aeroporto
8. Atribuir local de interesse
9. Cancelar viagem que não dê tempo suficiente
10. Eliminar viagem que não dê tempo suficiente

Principais dificuldades encontradas na resolução do trabalho

O desenvolvimento desta aplicação trouxe algumas adversidades na escrita do código e na sua estruturação. Contudo, com alguma reflecção foram ultrapassadas, como por exemplo fazer a divisão de classes e a utilização do graph e das suas propriedades. Após a execução deste projeto ficaram mais clarificadas as noções de graph, edges weight, vertex, entre outros, o que nos trás vantagens para podermos trabalhar de melhor forma em qualquer projeto futuro.