U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

MESTRADO INTEGRADO EM ENGENHARIA INFORMÁTICA E COMPUTAÇÃO

CONCEÇÃO E ANÁLISE DE ALGORITMOS

EcoPonto

RECOLHA SELETIVA DE LIXO

28 de maio de 2019

Turma 7

Cláudia Inês da Costa Martins – up201704136@fe.up.pt

Diogo Rafael Amorim Mendes – up201605360@fe.up.pt

Rita Nunes da Mota – up201703964@fe.up.pt

Índice

1. Descrição do tema a implementar

O tratamento de lixo e resíduos é uma tarefa que todas cidades devem realizar da forma mais eficiente possível. Para transportar lixo e resíduos até às estações de tratamento, as câmaras municipais geralmente mantêm uma frota de camiões especializados que realizam uma determinada rota de recolha, transportando os resíduos até as estações de tratamento. Estas rotas são, geralmente, definidas a priori e realizadas sistematicamente, segundo uma dada frequência.

Nas chamadas “smart cities”, pretende-se realizar tal recolha de uma forma mais inteligente. De facto, os contentores espalhados em diversos pontos da cidade, onde os moradores da vizinhança depositam o lixo, podem não estar suficientemente cheios para justificar o seu esvaziamento pelo camião da recolha, o que tornaria algumas viagens desnecessárias. Com a tecnologia das redes de sensores a desenvolver-se rapidamente, já é possível realizar uma monitorização mais efetiva do nível de acumulação de resíduos em cada contentor distribuído na cidade.

Uma Central de Recolha deseja desenvolver uma aplicação que gera, de forma adaptativa, as rotas de recolha, considerando apenas os contentores com resíduo suficiente que justifique a recolha pelo camião, ou seja, enquanto os contentores não estiverem suficientemente cheios, não serão incluídos na rota dos camiões.

Esta aplicação deve também identificar a rota que minimize o caminho efetuado por cada veículo.

1.1. Primeira Fase

Numa primeira fase, a recolha é realizada por um camião de capacidade ilimitada, que pode efetuar a rota que inclui todos os contentores a necessitarem de ser esvaziados.

Para além da rede de ruas por onde o camião pode circular, e respetivos sentidos, a aplicação também recebe e guarda a localização de todos os contentores e respetivos níveis de acumulação de resíduos.

A melhor rota desde a Central de Recolha, onde ficam estacionados os camiões, até às estações de tratamento, onde deve ser depositado todo o lixo recolhido, é sugerida pela aplicação.

A avaliação da conetividade do grafo é bastante importante, pois a rota só pode ser definida quando existem caminhos que conectem todos os POI'S (Pontos de Interesse) – Central de Recolha, contentores e estação de tratamento. Isto é, todos os POI'S devem estar incluídos no mesmo conexo do grafo. A partir da Central, o camião deve ser capaz de chegar a todos os contentores que é suposto esvaziar e terminar o seu percurso numa estação de tratamento; por fim, deve regressar ao ponto inicial, apesar da aplicação não criar uma rota de regresso é imperial que esta exista.

Algumas vezes, obras nas vias públicas, estradas onde a circulação de camiões seja proibida ou até ruas de pouca largura podem eventualmente tornar certas zonas inacessíveis, o que implica ignorar certas arestas durante o processamento do grafo.

1.2. Segunda Fase

Numa segunda fase, há vários camiões, mas desta vez de capacidade limitada e dedicados à coleta seletiva, ou seja, existem agora diferentes camiões para vários tipos de resíduos em ecopontos (contentor azul – papel, contentor amarelo – plástico e contentor verde – vidro).

Neste caso, a definição de apenas uma rota não será suficiente pois cada camião está especializado para a recolha de um tipo de resíduo. Logo, será necessária a criação de, no mínimo, três rotas para três camiões diferentes.

Como os camiões possuem agora uma capacidade limitada, caso atinjam a sua capacidade máxima, outro camião terá de completar a recolha total do tipo de resíduo em questão. É importante esclarecer que todos os contentores são esvaziados na totalidade por cada camião, logo, caso a quantidade atual de lixo do camião mais a quantidade de resíduos no contentor exceda a capacidade máxima do veículo, outro camião terá de terminar a recolha.

Como na primeira fase, continuámos a sugerir a rota mais curta, mas na eventualidade de uma rota mais longa com a utilização de um menor número de camiões, a aplicação dará sempre prioridade à rota onde sejam utilizados menos veículos de recolha.

2. Identificação e formalização do problema

2.1. Dados de entrada

- **Tmax** - Taxa de máxima ocupação de um contentor (pronto a recolher).
- **Tipo** - Tipo de lixo a recolher (plástico, vidro, papel).
- **Ci[i]** - Conjunto de camiões da Central de Recolha, onde cada veículo possui uma **cap** – capacidade total do camião.
  O valor de cap é infinito na primeira fase.
- **Gi = (Vi, Ei)** - grafo dirigido pesado, composto por:
  - **V** - Vértices do grafo que correspondem a vários pontos do mapa.
    - Cada vértice possui uma determinada quantidade de lixo **L** e uma quantidade de lixo máxima **Lmax**.
    - A taxa de ocupação **T=L/Lmax** é calculada de maneira a saber se é ou não viável a recolha de lixo no vértice em questão.
    - O conjunto das arestas que partem dos vértices são identificadas por **Adj ⊆ E**.
    - **S ∈ Vi** - vértice inicial (central de camiões).
    - **T ⊆ Vi** - vértices finais (estações de tratamento do lixo).
  - **E** - Arestas do grafo que representam estradas ou outro tipo de vias no mapa.
    - Cada aresta tem um determinado **ID** e é delimitada por dois vértices cuja distância entre eles é guardada em **d**.

2.2. Dados de saída

- **G = (V, A)** – grafo dirigido pesado.
- **Cf[i]** – Conjunto de camiões usados, onde cada veículo possui uma **cap** – capacidade do camião usado e uma sequência de arestas a percorrer que perfaz o percurso **P = {e∈ Ei | 1 ≤ j ≤ |P|**.

2.3. Restrições

O propósito é minimizar a distância percorrida, como tal, algumas restrições foram tidas em conta para a implementação do algoritmo.

- A distância entre a central ou estação de tratamento e um conjunto de contentores é consideravelmente maior do que a distância entre contentores.
- Um veículo tem sempre uma capacidade bastante maior do que a capacidade de qualquer contentor.
- O gráfico é sempre não direcionado (exceto quanto aos vértices da central e da estação de tratamento).

Restrições relacionadas com os dados de entrada:

- cap > 0, pois representa o limite de peso ou volume de cada camião, o que nunca pode ser negativo ou igual a 0.
- L ≥ 0 e Lmax ≥ 0 pelos motivos expressos no ponto anterior.
- 0 < Tmax ≤ 1: como se trata de uma taxa, esta só pode estar contida entre 0 e 1. Tmax nunca pode ser 0 para não possibilitar a recolha de contentores vazios.
- 0 ≤ T ≤ 1 pelos motivos expressos no ponto anterior. Neste caso, T pode adquirir o valor 0, visto que o contentor pode estar vazio.
- d > 0, pois trata-se de uma distância entre dois pontos.
- Todos os vértices incluindo I e T têm obrigatoriamente de pertencer ao mesmo componente fortemente conexo do grafo G.
- Cada aresta A que pertence ao grafo G, tem de ser viável para a passagem de camiões, caso contrário não é adicionada a G.

Restrições relacionadas com os dados de saída:

- cap ≥ L para ser possível a recolha de lixo num determinado vértice. Caso a recolha já tenha ocorrido cap ≥ 0.
- Cf ≤ Ci: o número de camiões usados é forçosamente igual ou inferior ao número de camiões disponíveis.

- Caso Cf < Ci, isso significa que o lixo foi recolhido na totalidade pois não foram utilizados todos os camiões disponíveis. Logo, em cada vértice caso isto aconteça, L=0 e T=0.
- No percurso P, como o camião parte de I (Central de Camiões) é imperial que a primeira aresta de P seja adjacente ao vértice I.
- No percurso P, como o camião termina a viagem em T (Estação de Tratamento) é imperial que a última aresta de P seja adjacente ao vértice T.

Através destas restrições, podemos assumir que reduzir o número de viagens e veículos usados, resultará numa menor distância percorrida e que o primeiro contentor cujo lixo será recolhido deverá ser o fim de um conjunto de vértices.

2.4. Funções Objetivo

Neste problema, o objetivo principal é diminuir ao máximo a distância percorrida pelos camiões (função g), com prioridade para a minimização do número de camiões usados (função f), recolhendo assim todo o lixo. Assim teremos duas funções onde a nossa solução irá incidir, sendo a minimização de f priorizada em sobre a de g:

$$f = |C|$$

$$g = \sum_{C(i) \in C} [\sum_{A \in P} (d(A))]$$

3. Perspetiva de solução

O problema que nos é apresentado é um típico problema de Routing, e basear-nos-emos nisso para a sua solução.

Em primeiro, para obter o menor caminho a percorrer, iremos transformar o grafo original num contendo apenas vértices que representam os pontos de interesse, ou seja, pontos nos quais o contentor contém uma certa quantidade de lixo que justifique a sua recolha, e as arestas representando a alternativa de caminho mais curto entre os pontos. Para obter estas arestas, iremos utilizar o algoritmo de Dijkstra.

A implementação irá passar pela determinação do ponto de origem e ponto final e aplicar o algoritmo de Dijkstra. Para obter o ponto de origem e ponto final, iremos percorrer os pontos representados por V (pontos com contentores) pela ordem de início do percurso (central de Recolha) e aplicar o algoritmo a vértices adjacentes.

Assim, a solução para a problemática passará por: percorrer o conjunto de Pontos POI e obter o caminho mais curto entre dois pontos adjacentes.

Posteriormente, o algorimto escolhe o melhor conjunto de veículos da central (com a maior capacidade e o tipo de lixo adequado). Esta parte seguirá o algorimto referido que calculará o contentor mais próximo com necessidade de ser recolhido, move o veículo para esse sítio e repete o processo até que não ajam mais contentores ou o veículo esteja cheio.

Finalmente, estando o veículo cheio ou o lixo todo recolhido, o veículo é encaminhado para a estação de tratamento pelo caminho mais próximo até à mesma.

Deste modo, repete-se este ciclo de recolha até que não ajam mais contentores a ser recolhidos ou até que os camiões estejam todos ocupados.

Utilização do Algoritmo de Dijkstra

O algoritmo de Dijkstra é usado para calcular a menor distância entre dois nós num grafo dirigido ou não dirigido com arestas de peso não negativo, em tempo computacional O((|V| + |E|) log |V|). Começa por definir todas as distâncias de todos os vértices para infinito, exceto o de começo que é 0. Este é colocado numa fila de prioridade.

Posteriormente o algoritmo recolhe o conjunto de vértices adjacentes e itera entre eles definindo a distância e colocando-os na fila. Uma vez que todos os vértices tenham sido considerados, o vértice inicial é percorrido e extraído da fila.

Assim que o vértice final tenha sido atingido, o algoritmo termina e o caminho é salvo em cada um dos vértices, senão repete-se o processo para o vértice mais próximo (início da fila).

A implementação do algoritmo de Dijkstra é viável, pois o problema que este relatório tem como objetivo solucionar passa por encontrar o caminho mais curto entre dois pontos num grafo pesado com valores apenas positivos. Como é de natureza greedy irá garantir o melhor caminho entre dois vértices. Assim, o algoritmo torna-se bastante eficiente e fácil de implementar neste problema.

Devido à problemática que iremos solucionar, a nossa implementação passará por fixar o algoritmo a apenas encontrar o melhor caminho entre dois pontos – ponto inicial e final.

4. Estruturas de Dados do Programa

Representação do Grafo

A representação do grafo foi concebida a partir da estrutura **Graph** definida no ficheiro “Graph.h”. Esta, é composta por um vetor de vértices (representado por vector<Vertex<Node>> que compõem o grafo e, também estão incluídos métodos que permitem a manipulação do vetor, bem como a inserção e remoção de vértices.

Os vértices do grafo são representados pela classe **Node** definida em “Node.h” que é composta pelo número de identificação (torna cada vértice único no grafo) e pelas suas coordenadas em *x* e em *y*.

Por fim, as arestas do grafo estão representadas na classe **Edge** mais uma vez definida em “Graph.h”, e são compostas pelo respetivo peso de cada aresta, bem como um apontador para o vértice de destino.

Classes Relacionadas com Recolha de Lixo

De forma a conseguirmos desenvolver um programa que fizesse uma recolha seletiva do lixo numa determinada cidade, tivemos que criar uma classe para cada elemento indispensável neste projeto.

Os contentores estão representados pela classe **Container** definida em “Container.h” que guarda o vértice corresponde a um determinado contentor, a capacidade máxima de resíduo do contentor, o tipo de lixo que contém (azul, verde ou amarelo) e uma variável booleano responsável por averiguar se o contentor está cheio ou não.

De seguida, os camiões do lixo foram armazenados na classe **Truck** definida em “Garbage_truck.h”. Cada camião possui um vértice (Node) que indica a localização da central de recolha, ou seja, onde o camião está posicionado antes de iniciar o seu trajeto. Também possui uma variável que guarda a sua capacidade máxima de transporte e o tipo de lixo que recolhe.

Finalmente, as estações de tratamento guardadas na estrutura **TreatmentStation** em “Treatment_station.h” incluem um vértice que indica onde se encontram e uma variável que guarda o tipo de resíduos tratados numa determinada estação.

Graphviewer

Módulo fornecido que tem como principal função a visualização gráfica do grafo processado pelo programa.

Leitura de Mapas

A leitura dos mapas fornecidos no Moodle é feita através da classe **map_creator** definida em “map_creator.h”. Contém variáveis para guardar o grafo, os nós a serem inseridos no grafo, os contentores e as estações de tratamento. Esta estrutura possui funções que guardam de imediato toda a informação lida no grafo.

A função showGraph() é responsável por adicionar cada nó ao grafo. Graficamente, cada contentor é representado por um nó azul, cada contentor de reciclagem por um nó verde, cada aterro sanitário pela cor vermelha, as estações de reciclagem pela magenta e as estações de transferência de resíduo pelo ciano.

5. Conectividade do Grafo

Para avaliar a conectividade do gráfico foi usado o método descrito nas aulas teóricas da cadeira:

1. Pesquisa em profundidade no grafo G determina floresta de expansão, numerando vértices em pós-ordem (ordem inversa de numeração em pré-ordem);
2. Inverter todas as arestas de G (grafo resultante é Gr);
3. Segunda pesquisa em profundidade, em Gr, começando sempre pelo vértice de numeração mais alta ainda não visitado;

6. Casos de Uso Implementados

Quando iniciar o programa, o utilizador deparar-se-á com um menu que terá as seguintes opções:

1. Show map;
2. Get best path;
3. Check connectivity;
4. Exit;

Na primeira opção o utilizador irá visualizar, o grafo com toda a informação recolhida dos ficheiros.

Na segunda opção, é pedido ao utilizador um ponto de partida (central de camiões), o tipo de lixo a recolher, o número de camiões e a sua capacidade. Uma

vez inseridos estas dados, será mostrado ao utilizador os contentores a recolher por cada camião.

Na terceira opção, o utilizador pode fazer a verificação da conetividade do grafo representado através dos dados.

Na quarta opção o utilizador abandona a aplicação.

7. Algoritmo Implementado

O algoritmo que queríamos implementar era o algoritmo Dijkstra que tem como objetivo calcular o melhor caminho entre dois vértices diferentes do grafo.

Foi introduzido em 1956 por Edsger W. Dijkstra.

No caso do nosso problema, o objetivo era fixar o algoritmo de forma a encontrar o melhor caminho desde um vértice de origem até ao vértice de destino. É um algoritmo bastante eficiente, pois garante sempre o melhor caminho entre os dois vértices.

Pseudocódigo:

```
# Inicia-se os valores
para todo v ∈ V[G]
     dist[v]← ∞
     prec[v] ← nulo
dist[s] ← 0

Q ← V[G] # Conjunto dos vértices abertos
S ← ø # Conjunto dos vértices fechados

# Relaxamento das arestas
enquanto Q ≠ ø
          u ← extraia-mín(Q)
          S ← S U {u}
          para cada v adjacente a u
               se dist[v] > dist[u] + w(u, v)
                  então dist[v] ← dist[u] + w(u, v)
                        prec[v] ← u
```

8. Análise de complexidade do Algoritmo de Dijkstra

Este algoritmo divide-se em duas fases principais: encontrar o melhor caminho do vértice original ao vértice final e a reconstrução posterior desse percurso através da informação contida nos vértices.

Por sua vez, a primeira fase do algoritmo divide-se também em duas subfases: a primeira consiste na preparação dos vértices para a execução do algoritmo, onde a complexidade temporal desta fase é O(|V|) pois é linear em relação ao número de vértices do grafo; a segunda subfase consiste na pesquisa em si onde a inserção e remoção de vértices é feita auxiliada por uma fila de prioridade em que o tempo de execução é logarítmico na inserção, O(log n), e linear na remoção. Por esta razão, a complexidade temporal da primeira fase do algoritmo é de O((|V| + |E|) * log |V|) pois existe uma reordenação dos vértices na estrutura devido às inserções na fila de prioridade.

A segunda fase do algoritmo resume-se a percorrer todos os vértices que pertencem ao caminho processado, logo é linear relativamente ao número de vértices do grafo.

O algoritmo de Dijkstra tem por fim complexidade temporal de O((|V| + |E|) * log |V|) .

9. Conclusão

Durante a realização deste trabalho, a principal dificuldade foi a gestão do tempo, pois tivemos mais entraves na implementação do que o que contávams.

Tentamos ao máximo colmatar a falta de tempo, e acabámos por conseguir desenvolver todas as classes relativas concretamente ao nosso problema (Truck, TreatmentStation, Container). Também conseguimos ler corretamente os ficheiros fornecidos no Moodle e armazenar corretamente todos os nós, coordenadas, edges e tags neles contidos. A implementação do graph viewer foi bastante trabalhoso e demorada, mas acabamos por mesmo assim conseguir processar o grafo graficamente com a adição de cores de forma a distinguir os vários vértices.

Um dos nossos grandes objetivos era implementar o algoritmo de Dijkstra como referimos na entrega intermédia e cremos que embora não completamente otimizado, conseguimos implementar este algoritmo.

Através deste trabalho, aprofundamos o estudo teórico que temos feito ao longo do semestre acerca dos diferentes algoritmos para calcular caminhos e das técnicas de programação dinâmica.

Cada elemento do grupo realizou diferentes tarefas:

Cláudia Martins – Implementação do relatório intermédio e final, do algoritmo de Dijkstra e da conectividade do grafo, desenho do grafo através da interface do GraphViewer. Implementação da classe map_creator e Graph e contribuição nas classes Truck, TreatmentStation, Container, Garbage_truck e Node.

Diogo Mendes – Esteve envolvido na implementação do relatório intermédio e final. Ajudou a implementar as classes Truck, TreatmentStation, Container, Garbage_truck e Node. Também tentou mas com insucesso implementar o algoritmo Djikstra.

Rita Mota – Contribuição na análise da conetividade do grafo e implementação da classe Graph.

10. Bibliografia

Durante a realização deste trabalho, consultamos diversas fontes a seguir citadas:

- Slides disponibilizados na página do moodle da cadeira;
- Relatórios-exemplo disponibilizados na página do moodle da cadeira;
- https://en.wikipedia.org/wiki/Category:Routing_algorithms;
- https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/
- https://www.geeksforgeeks.org/graph-data-structure-and-algorithms/