Faculdade de Engenharia da Universidade do Porto

ECOPONTO: RECOLHA DE LIXO SELETIVA

Concepção e Análise de Algoritmos 2º ano (2º semestre) -- MIEIC:

Rosaldo José Fernandes Rossetti

Ana Paula Cunha da Rocha

Estudantes & Autores:

Cláudia Marinho, up201404493, up201404493@fe.up.pt

Diogo Serra Duque, up201406274, up201406274@fe.up.pt

João Costa, up201403967, up201403967@fe.up.pt

Turma 2MIEIC03, Grupo F

Abril 2016

Descrição do problema

Neste projeto, propusemo-nos a resolver um problema relevante nos tempos atuais: a recolha de lixo seletiva. A colheita de resíduos para reciclagem é uma tarefa que deve ser executada o mais eficientemente possível, no entanto muitas vezes os camiões de recolha de lixo esvaziam ecopontos quase vazios o que leva a uma perda de tempo desnecessária.

Num contexto das *smart cities*, podemos admitir que é possível saber o conteúdo dos ecopontos e como tal saber se a viagem dos camiões da Central de Recolha até a cada ecoponto compensa ou não. Assim sendo, o objetivo deste projeto é desenvolver uma aplicação que seria então usada pela Central de Recolha que permita determinar as rotas de recolha mais eficientes tendo em conta os conteúdos dos ecopontos (se estes não se encontrarem suficientemente cheios, então os camiões não esvaziam esses ecopontos).

Dados de entrada

Para a aplicação funcionar adequadamente, esta necessita da informação dada pelos seguintes ficheiros:

- **Nodes.txt**: Ficheiro com informação acerca das interseções entre ruas (nós do grafo). Cada linha do ficheiro deve ter a seguinte estrutura: "*id_no;longitude_graus;latitude_graus;longitude_radianos;latitude_radianos"*.
- **Roads.txt**: Ficheiro com informação geral acerca das ruas. Cada linha do ficheiro deve ter a seguinte estrutura: "*id_rua;nome_rua;tem_2_sentidos"*. O parâmetro "tem_2_sentidos" deve ser "True" ou "False", caso seja uma rua de 2 sentidos ou não.
- **Connections.txt**: Ficheiro com informação acerca das ligações entre os nós. Cada linha do ficheiro deve ter a seguinte estrutura: "*id_rua;id_no1;id_no2;"*. O parâmetro id_rua deve corresponder ao id_rua presente em "Roads.txt", e os "id_noX" devem corresponder a "id_no" lidos em "Nodes.txt".
- **Ecopontos.txt**: Ficheiro com informação acerca dos ecopontos. Cada linha do ficheiro deve ter a seguinte estrutura: "*id_no;quantidade_lixo"*. O parâmetro "id_no" representa um nó (lido em "Nodes.txt") onde existe um ecoponto com uma quantidade de lixo "quantidade_lixo*"*, tendo em conta que esta quantidade deverá estar entre 0 e 100 (quilogramas).
- **Trucks.txt**: Ficheiro com informação acerca dos camiões usados na recolha do lixo. Cada linha do ficheiro deve ter a seguinte estrutura: "*nome_camiao;capacidade_maxima;cor"*. O parâmetro "nome_camiao" representa o nome que será associado ao camião, "capacidade_maxima" representa a quantidade máxima de lixo que o camião pode transportar e "cor" é a cor do seu trajeto no GraphViewer caso este tenha de ser usado na recolha. Notar que a capacidade máxima do camião deve ser maior que 100.
- **BlockedRoads.txt:** Ficheiro com informação acerca das ruas bloqueadas. Estas ruas são excluídas das rotas dos camiões. Cada linha do ficheiro deve ter a seguinte estrutura: *"id_no_fonte;id_no_destino"*. A rua que vai ser excluída da rota dos camiões vai do nó fonte ao nó destino. Notar que se a rua tiver dois sentidos, *"id_no_fonte;id_no_destino"* apenas bloqueia um dos sentidos (para também bloquear o sentido contrário é necessário incluir *"id_no_destino;id_no_fonte"* também).
- **Drivers.txt**: Ficheiro com os nomes dos motoristas disponíveis para conduzir um camião. Cada linha do ficheiro tem o nome de um motorista.

NOTA: as linhas dos ficheiros “Roads.txt” e “Connections.txt” devem estar ordenados por “id_rua” de modo a assegurar que cada ligação lida de “Connections.txt” fique na rua devida.

Além da informação fornecida pelos ficheiros, o utilizador também precisa de fornecer a carga mínima dos ecopontos, ou seja, se os ecopontos do mapa tiverem menos lixo que o valor fornecido pelo utilizador então esses ecopontos não são incluídos na rota dos camiões.

Formalização do problema

Dados

Um grafo dirigido pesado, G = <V,E>, que representa um mapa:

- Vértices (V): cada vértice corresponde a um ponto significativo de uma estrada (interseção, curva);
- Arestas (E): cada aresta representa um troço de estrada. O seu peso é a distância percorrida nesse troço.

Os dados mais significativos do nosso projeto são: pontos, arestas, lista de ecopontos, lista de camiões e um vetor de ruas bloqueadas.

Cada ponto é representado por um conjunto de coordenadas e um ID.

Cada aresta tem o seu peso calculado através da distância cartesiana entre os dois vértices que liga.

Na lista de ecopontos, cada ecoponto possuí um id que representa a sua posição no mapa e a quantidade de lixo que contém.

Na lista de camiões, cada camião possuí uma capacidade.

No vetor de ruas bloqueadas, cada rua bloqueada possuí o ID do nó de origem e o ID do nó de destino.

Inputs

O programa tem os seguintes inputs:

- Grafo dirigido, G = <V,E>
- Lista de ecopontos;
- Lista de camiões;
- Vetor de ruas bloqueadas;
- Vetor de motoristas;
- Ponto inicial dos camiões (central);

Outputs

O programa tem os seguintes outputs:

- Caminho de vértices com a viagem a realizar por cada camião
- Todos os ecopontos que foram esvaziados mostram qual foi o camião que o esvaziou (indicando a cor desse camião)

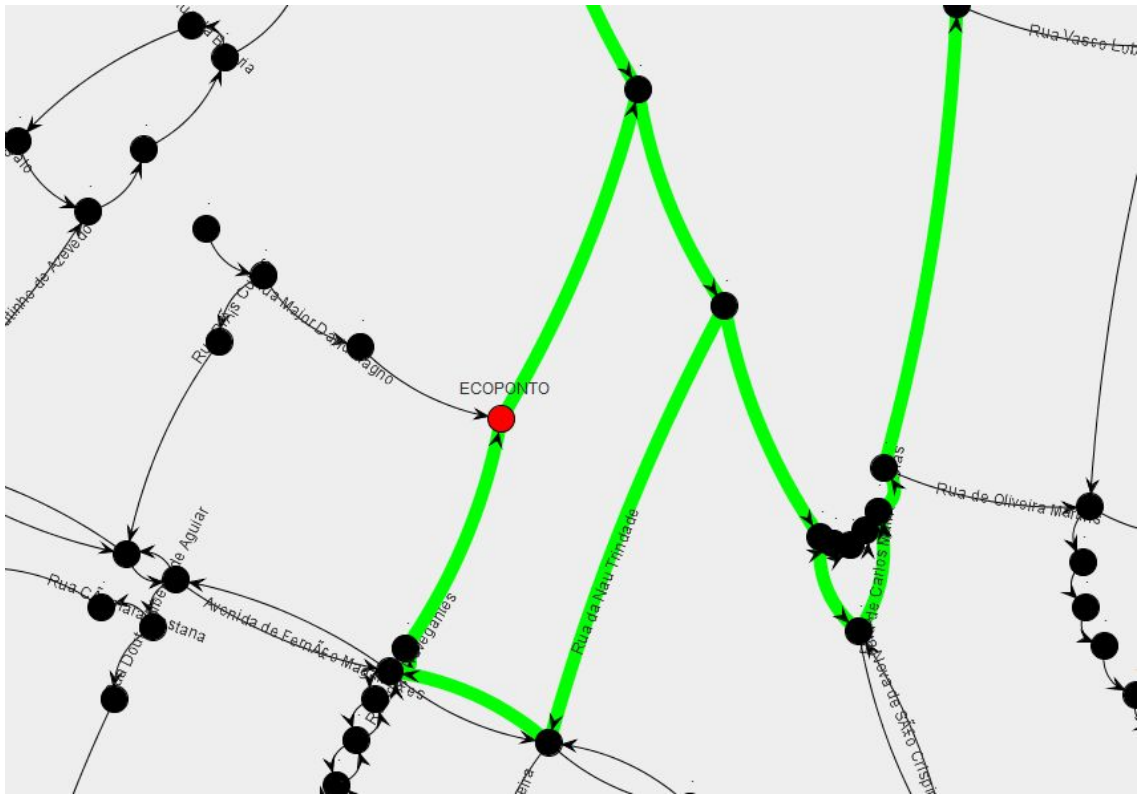
Limites e condições da aplicação

Esta aplicação não apresentará uma solução ótima se não houverem camiões que cheguem para recolher todo o lixo. Se não houverem camiões que cheguem, pode ocorrer que o conjunto de ecopontos recolhidos não seja o melhor.

De forma a apresentar uma solução ao problema o utilizador deve ter também cuidado com a introdução do grafo, sendo que este deve ser conexo, ou seja, deve ser possível percorrer um caminho entre a central dos camiões e todos os ecopontos e entre um ecoponto e todos os restantes.

Além disso, situações irreais como ecopontos com mais lixo do que um camião é capaz de conter não são suportadas.

Pode haver dificuldades na visualização de caminhos devido à complexidade dos trajetos. Isto pode acontecer pois mais que um camião pode passar pelas mesmas ruas ou então pode acontecer uma situação semelhante à retratada em baixo:



Resultados esperados

Devem ser escolhidos o melhor conjunto de camiões para recolher o lixo. Cada camião deve percorrer o melhor conjunto de ecopontos de forma a colectar o máximo de lixo possível. Estes ecopontos devem ser visitados no menor trajeto possível.

Também é possível ver os nomes das maior parte das ruas, as localizações dos ecopontos (que no grafo são representados a vermelho) e a localização da central de camiões (que no grafo é representada a verde). É de notar que nos nomes dos ecopontos é indicado qual foi o camião responsável pela recolha do seu lixo (caso estes tenham sido esvaziados).

Descrição da Solução

Os dados usados neste projeto foram extraídos de um mapa com o auxílio da ferramenta “OSM2TXT Parser”, que criava 3 ficheiros, aos quais demos os nomes “Nodes.txt”, “Roads.txt” e “Connections.txt”.

O programa começa por perguntar se o utilizador quer visualizar o nome das ruas no GraphViewer. Em caso afirmativo, pergunta se prefere que cada rua tenha o seu nome várias vezes escrito, ou se uma vez basta.

Após estas perguntas, os ficheiros anteriormente mencionados são lidos no nosso programa e a informação destes ficheiros é guardada num grafo. Após a leitura dos 3 ficheiros, prossegue-se à leitura de outros 4 ficheiros essenciais ao funcionamento do programa: “Trucks.txt”, “Drivers.txt”, “Ecopontos.txt” e “BlockedRoads.txt”.

Tendo já toda a informação necessária, o grafo que tinha sido criado anteriormente é representado através da ferramenta GraphViewer o mais fielmente possível. Além disso todos os ecopontos e ruas bloqueadas são representados no GraphViewer.

É então feita uma questão ao utilizador caso ele pretenda mudar o valor a localização da central (que é 1011 por padrão) e, se sim, então o utilizador deve introduzir o id da nova central.

De seguida é perguntado ao utilizador se gostaria de adicionar ecopontos ao mapa. Enquanto o utilizador quiser adicionar ecopontos, deverá ir introduzindo o nome da rua onde pretende adicionar um ecoponto e a quantidade de lixo no mesmo.

Por esta altura, já só falta que o utilizador escolha que motoristas ficam com que camiões. Será apresentada uma lista dos motoristas disponíveis para um certo camiao, e o utilizador deve fazer a escolha. Em caso de o nome não ser reconhecido, será feita a pergunta enquanto a situação se verificar.

Depois de recebidos os últimos inputs, as ruas cortadas são retiradas do grafo e todos os ecopontos com menos quantidade de lixo que o valor de carga mínima são excluídos da lista de ecopontos, de forma a que estes não sejam nas rotas dos camiões.

Com a informação acerca dos ecopontos e do ponto de partida dos camiões (central), avaliava-se a conectividade entre os mesmos, de modo a garantir que um camião conseguiria passar por todos os ecopontos partindo do ponto inicial.

Tendo agora validado e obtido toda a informação necessária à aplicação dos algoritmos, entramos num ciclo que só acabará se os camiões se esgotarem, ou se todo o lixo de todos os ecopontos for recolhido. O ciclo tem a seguinte estrutura:

- Verifica-se a quantidade de lixo total que é necessário recolher e encontra-se o camião com menor capacidade capaz de recolher essa quantidade. Caso este não exista, seleciona-se o camião com maior capacidade. (Algoritmo ganancioso)
- Depois de selecionado, este é retirado da lista de camiões (para não ser repetido) e cria-se um circuito de Hamilton para este percorrer. O circuito é criado através de backtracking num grafo simplificado (apenas com os ecopontos e os a central, em que as distâncias entre os todos os pares de vértices foram obtidas com o algoritmo de Dijkstra).

Algoritmos utilizados

Algoritmo de Seleção do Camião

Tendo como dados a lista de camiões e a lista de ecopontos a recolher, o melhor camião é o camião com menor capacidade, que consegue recolher todo o lixo, ou o maior camião no caso de não haver nenhum camião que consiga recolher todo o lixo.

Aplica-se da seguinte forma:

```
for(n = 0 to N)
     if ( C[n] > T )
          return n
return n
```

n: camião
N: número de camiões
C[n]: capacidade de n
T: total de lixo dos ecopontos

Este algoritmo tem um complexidade temporal O(N), e presume que os camiões estão ordenados por ordem de capacidade.

Algoritmo de Seleção dos Ecopontos a Recolher

Tendo como dados os camiões e a lista de ecopontos a recolher, tenta-se dividir os ecopontos pelos camiões de forma a utilizar o minimo de camiões possíveis. Assim este algoritmo não visa encher ao máximo uma camião mas aproveitar ao máximo o espaço conjunto dos camiões. Assim aplica-se um algoritmo ganancioso para encher os camiões.

Aplica-se da seguinte forma:

```
while(T ! = 0)
    n = BestTruck()
         for (e = E to 0)
```

```
if (C[n] > C[e])
    e -> EcoList[n]
    C[n] = C[n] - C[e];
```

n: camião
e: ecoponto
E: número de ecopontos
C[i]: capacidade de i
T: total de lixo dos ecopontos
EcoList[n]: lista de ecopontos a visitar por n

Este algoritmo tem um complexidade temporal O(N*N*E) no pior caso, mas isto só acontece em situações irreais em que os ecopontos tem capacidades próximas dos camiões. No caso geral, a complexidade temporal não excede muitas vezes O(N*E). O algoritmo presume que ecopontos estão ordenados por ordem de capacidade.

Algoritmo de Seleção de Caminhos - (circuito de Euler criado com backtracking)

Tendo como dados um grafo apenas "simplificado", tenta-se criar uma rota otimizada que passe em todos os seus vértices.

Neste grafo "simplificado":

- os vértices são "cópias" dos vértices do grafo original onde existem ecopontos.
- existe uma aresta para cada par possível de pontos, sendo que os pesos das arestas criadas são calculados recorrendo ao algoritmo de Dijkstra.

O algoritmo aplica-se da seguinte forma:

```
backtrackingHamilton(currPath, currWeight, &bestPath, &bestWeight)
    if(currWeight>=bestWeight)
        return
    if(currPath.size()==vertexSet.size())
        bestWeight=currWeight
```

```
            bestPath=currPath
            return
        for( i=0; i<vertexSet.size(); i++)
            currPath.push(vertexSet[i])
            backtrackingHamilton(currPath, currWeight+thisPathWeight,
                                 bestPath, bestWeight)
            currPath.pop()
```

currPath: caminho atual deste “branch”

currWeight: peso atual deste “branch”

bestPath: melhor caminho encontrado até ao momento

bestWeight: peso do caminho encontrado até ao momento

vertexSet: vertices do grafo simplificado

Este algoritmo tem um complexidade temporal O((nVertices-1)!), sendo nVertices o número de vertices no grafo simplificado.

Algoritmo de Pesquisa Exata - (Knuth-Morris-Pratt)

Tendo como dados a lista de ruas e uma string para procurar, utiliza-se o algoritmo de Knuth-Morris-Pratt para cada linha de modo a descobrir se contem a string a pesquisar.

Este algoritmo tem um complexidade temporal O(E), com E sendo endo o número de arestas do grafo. O algoritmo de Knuth-Morris-Pratt, utilizado a cada iteração, possui uma complexidade temporal O(n + m), sende n e m o numero de caracteres das duas strings a comparar.

Algoritmo de Pesquisa Aproximada

Neste trabalho o algoritmo de pesquisa aproximada é usado apenas na pesquisa dos nomes das ruas, ou seja, para obter o nome de uma determinada rua, o

utilizador apenas precisa de introduzir um nome parecido. Por exemplo, para obter “Rua de Ramalho Ortigão”, o utilizador só precisa de introduzir “ramalho ortigao” para obter a rua indicada anteriormente. É de notar que antes de usar este algoritmo, a string introduzida pelo utilizador e os nomes das ruas têm todos os seus caractéres convertidos para maiúsculas (para facilitar a procura).

A função que implementa este algoritmo apenas precisa de receber dois argumentos: um nome de uma rua e a string introduzida pelo utilizador. Este algoritmo depois prossegue ao cálculo do número de inserções, deleções e substituições necessárias para transformar a string introduzida pelo utilizador no nome da rua. Depois esta função é usada num ciclo de tal forma que a string do utilizador seja comparada com todos os nomes das ruas. A string mais parecida é o resultado final.

Este algoritmo tem uma complexidade temporal e espacial similar a $O(|P|.|T|)$, sendo $|P|$ o tamanho da string correspondente ao nome da rua e $|T|$ o tamanho da string introduzida pelo utilizador.

Casos de Utilização

A solução apresentada pode ser utilizada para gerir as rotas de uma frota de camiões do lixo, utilizando dados provenientes de mapas reais.

O utilizador fornece documentos com os dados sobre o mapa, ecopontos e a frota de camiões e programa gera automaticamente os melhores caminhos para cada camião de forma a que a recolha do lixo seja eficiente.

O programa permite ao utilizador adicionar novas ecopontos em ruas especificas utilizando algoritmos de pesquisa exacta e aproximada em strings.

Principais dificuldades

Durante a realização do trabalho foram várias as dificuldades encontradas. Houveram bastantes problemas relacionados com o material fornecido.

Por um lado, o OSM2TXT Parser não funcionava de início e só nas ultimas semanas de desenvolvimento do projecto foi fornecida uma versão funcional.

Por outro lado, no GraphViewer existiam várias limitações, a salientar o facto de o programa só aceitar “ints” como input, o que causava conflitos com os IDs provenientes do OSM2TXT que podiam apresentar valores que excediam a capacidade máxima de um “int”. Existiam também várias funções que não funcionam, como por exemplo defineVertexSize(), defineEdgeThickness() e defineEdgeCurved(). Em termos visuais, o graphViewer não permite uma boa visualizão de caminhos quando existe uma grande densidade de pontos.

Esforço dedicado por cada elemento do grupo

Ao longo do trabalho, todos os elementos deram o seu melhor para a realização do mesmo. De modo a facilitar o desenvolvimento da aplicação, distribuímos diferentes tarefas para cada elemento:

João Costa (up201403967):

- Desenvolvimento do algoritmo de Seleção dos Camiões.
- Desenvolvimento do algoritmo de Seleção dos Ecopontos em função do Camião.
- Desenvolvimento do algoritmo de Construção dos Caminhos.

Diogo Duque (up201406274):

- Conversão da informação dos ficheiros obtidos no “OSM2TXT Parser” para Graph.
- Conversão da informação do Graph para GraphViewer.

Cláudia Marinho (up201404493):

- Correção de bugs e resolução de problemas que ocorreram ao longo do trabalho.
- Implementação das ruas bloqueadas.
- Desenvolvimento do código relacionado com a exclusão de ecopontos das rotas dos camiões

Tudo o resto foi desenvolvido em conjunto, sem divisão especifica de tarefas, pelo que não há discrepâncias no contributo de cada membro para a realização das mesmas.

Referências

- ROSSETTI, R. Slides de CONCEPÇÃO E ANÁLISE DE ALGORITMOS, 2015/2016