Relatório CAL

Tema 9 – EatExpress: Entrega de comida

24 de abril de 2020

Turma 1, Grupo 4:

Carlos Lousada, up201806302@fe.up.pt

João Matos, up201703884@fe.up.pt

Pedro Queirós, up201806329@fe.up.pt

Índice

Introdução – descrição do tema

O trabalho consiste em desenvolver uma aplicação que permita a uma empresa de entrega de comida (EatExpress) determinar que percursos os seus estafetas devem utilizar, sendo que estes podem usar diversos meios de transporte, de forma a minimizar o tempo de espera do cliente. Relativamente aos clientes, estes podem escolher um restaurante e o prato que querem. Por fim, o sistema também terá em conta acidentes ou obras que possam inviabilizar certos percursos.

Identificação e formalização do problema

- **Descrição**

Uma utilização típica do programa requer o cálculo do trajeto mais curto entre o estafeta e o restaurante que o cliente requisitou, assim como o cálculo posterior do caminho do restaurante à localização do cliente, formando assim o trajeto completo do estafeta. Desta forma, os dados apresentados abaixo refletem o cálculo de uma das componentes do caminho total. O algoritmo repetir-se-á para a segunda parte do itinerário.

- **Dados de entrada**
  - Grafo com vértices e arestas (que representam, respetivamente, interseções e vias na rede), representado por G(V,E), será armazenado sob a forma de um vetor de vértices

    - Arestas ($e \in E$) contêm:
      - número de ID (*id(e)*)
      - peso (a distância física entre os nós que liga), representado por $distance_{ij}$ ou *weight(e)*
      - campo (booleano) a indicar se a aresta está disponível (no caso de haver obras ou acidentes), representado por $disp_{ij}$ ou *disp(e)*
      - vértice de destino, guardado como um apontador (*dest(e)*)
    - Vértices ($v \in V$) contêm:
      - número de ID (*id(v)*)
      - conteúdo do nó (uma classe com a posição geográfica), representado por *info(v)*
      - conjunto das arestas que dele partem (*adj(v)*)
      - um campo (booleano) auxiliar a indicar se já foi visitado (usado pelos algoritmos), representado por *visited(v)*
      - um campo com a distância ao vértice inicial (*dist(v)*)
      - o índice a usar na fila de prioridade (*queueIndex(v)*)
      - uma etiqueta a dizer se o nó é um cliente, estafeta, restaurante ou nenhum deles (*tag(v)*)

- Um vértice origem (início do percurso), representado por *source*
- Um vértice destino (fim do percurso), representado por *dest*

- **Dados de saída**
  - Percurso mais curto entre a origem e o destino (sob a forma de um vetor de apontadores para vértices), representado por *path*

- **Função objetivo**
  - O mínimo do somatório dos pesos das arestas que ligam os vértices pertencentes ao percurso
    (min (Σ $distance_{ij}$, $i \in path, j \in path$))

- **Restrições**
  - Os números de ID não podem ser negativos (*id(v)* > 0, *id(e)* > 0)
  - Os pesos das arestas têm de ser números positivos (*weight(e)* > 0 ou $distance_{ij}$ > 0, visto que representam distâncias físicas)
  - A distância ao vértice inicial necessita de ser maior que zero (*dist(v)* > 0) de acordo com o que foi mencionado no ponto anterior (só poderá ser zero na origem do caminho)
  - Tem de haver um mínimo de arestas disponível de forma a ser possível a construção de percursos
  - Os vértices de origem e de destino têm de ser distintos (*orig ≠ dest)*
  - Têm de existir arestas entre os vértices que constituem o percurso

- O índice a usar na fila de prioridade não pode ser inferior a zero (uma limitação imposta pela implementação de *MutablePriorityQueue* fornecida)

Perspetiva de solução

Algoritmos em consideração:

- Pesquisa bidirecional usando o algoritmo de Dijkstra
  - O algoritmo teria uma implementação semelhante à usada nas aulas práticas, com algumas diferenças:
    - O percurso seria guardado num vetor de apontadores
    - Quando há arestas indisponíveis (seja porque ocorreu um acidente ou obras), estas são ignoradas
- Pesquisa bidirecional usando o algoritmo A*
  - O algoritmo teria uma implementação semelhante ao de Dijkstra explicado acima, com algumas diferenças:
    - Os vértices têm um campo adicional onde é armazenada uma estimativa da distância desse vértice ao destino
    - Quando são realizadas as verificações aos vértices adjacentes, em vez de apenas comparar a distância de um vértice adjacente à origem com a soma da distância do vértice atual e o peso da aresta, soma-se também o valor estimado e só depois se realiza a comparação
    - Para a estimativa usar-se-ia a distância Euclidiana

Casos de utilização

- Caso em que só há um estafeta com capacidade ilimitada para carga
- Caso em que há vários estafetas (com meios de deslocação possivelmente diferentes), ainda com capacidade ilimitada
- Caso em que há vários estafetas (com meios de deslocação possivelmente diferentes), mas com uma capacidade de carga limitada

Funcionalidades a implementar

- Funcionalidades CRUD (inserir clientes e estafetas, visualizar a sua informação, atualizar a sua informação ou removê-los)
- Realizar uma encomenda (da perspetiva do cliente)
- Atribuir uma encomenda ao “melhor estafeta” (de acordo com vários critérios como proximidade e capacidade de carga)

Conclusão

Em suma, o foco do trabalho envolve o cálculo de dois percursos mais curtos (um do estafeta para o restaurante e outro do restaurante para o cliente). Isto pode ser feito usando os dois algoritmos mencionados anteriormente numa implementação bidirecional, sendo que o algoritmo A* tende a ser mais eficiente e, neste caso, de acordo com a informação fornecida nas aulas teóricas, garante o resultado ótimo.

Relativamente à participação na elaboração do relatório, os três elementos do grupo contribuíram de forma semelhante, seja na construção do texto, na discussão de ideias ou na identificação de possíveis soluções.

Bibliografia

- Slides usados pelo professor durante as aulas teóricas
- Vídeo explicativo sobre o algoritmo A* (https://youtu.be/ySN5Wnu88nE)