RELATÓRIO

Concepção e análise de algoritmos

Faculdade de Engenharia da Universidade do Porto

26 de Abril de 2013

Duarte Nuno Pereira Duarte
ei11101@fe.up.pt
Miguel Rui Pereira Marques
ei11099@fe.up.pt
Ruben Fernando Pinto Cordeiro
ei11097@fe.up.pt

Conteúdos

Problema

Contexto

A Bacia Amazónica é uma bacia hidrográfica complexa, formada por vários rios secundários, interligados por rios de calibre menor conhecidos por igarapés. A navegabilidade dos vários canais da bacia hidrográfica Amazónica depende do seu caudal, que varia sazonalmente ao longo do ano. O principal meio de transporte entre as várias populações ribeirinhas na margem desses canais é a navegação fluvial, com ferries denominados por *balsas* a realizarem a maior parte do transporte de mercadorias entre as várias populações.

Descrição

Perante a configuração da rede de transporte, pretende-se implementar uma aplicação que auxilie o transporte de carga fluvial de uma empresa através do cálculo de itinerários óptimos para a realização das suas entregas em diversas populações ribeirinhas.

O fator de custo da entrega depende do peso da encomenda a transportar e do sentido da corrente ao longo do trajeto das balsas: o custo aumenta quando as embarcações navegam contra a corrente, ao passo que diminui no sentido oposto.

Variáveis do problema:

- Configuração da rede hidrográfica;
- Sentido da corrente entre os vários pontos (canais) da rede;
- Estado de navegabilidade de cada canal;
- Capacidade das balsas e das embarcações de apoio;
- Encomendas a realizar;
- Destino de cada encomenda.

Além destas variáveis, existem variações para a optimização das rotas das unidades de transporte:

1. A empresa transportadora possui balsas com a mesma capacidade de carga, sem embarcações de apoio, em número indeterminado;
2. Cada balsa poderá possuir uma embarcação de apoio;
3. O número de balsas do item anterior é limitado;

Requisitos

Funcionalidade

Mediante uma configuração da rede hidrográfica e um conjunto de encomendas, a aplicação deve ser capaz de:

- Gerar itinerários óptimos (com um custo mínimo) desde uma fonte para um conjunto de encomendas associadas a um ou mais destinos, tendo em conta os fatores de custo;
- Determinar quais as encomendas alcançáveis mediante as variações do problema;
- Computar o custo de envio de cada encomenda;
- Simular a sazonalidade rede hidrográfica (rede de transporte), alterando a capacidade dos canais;
- Permitir a visualização dos resultados de forma inteligível (detalhar o percurso) através da *API* disponibilizada;
- Guardar os dados das sessões em ficheiro para futuras execuções.

Casos de utilização

A interface com o utilizador será constituída por vários menus.

Menu inicial:

- Criação de uma bacia hidrográfica;
- Carregamento da configuração de uma bacia hidrográfica.

Menu principal:

- Adicionar uma nova população ribeirinha;
- Adicionar um novo rio;
- Alterar a sazonalidade da bacia;
- Visualizar configuração atual da bacia;
- Gravar a configuração da bacia hidrográfica num ficheiro;
- Criação de um conjunto de encomendas;
- Carregar informação relativa a encomendas anteriores.

Menu relativo às encomendas:

- Criar uma nova encomenda;
- Alterar a fonte da encomenda;
- Alterar a capacidade das balsas;
- Alterar a capacidade das embarcações de apoio;
- Alterar o número de embarcações de apoio;
- Guardar informação relativa à encomenda em ficheiro.

Estrutura

Modelação de dados

A configuração da rede de transporte é representada por um **grafo dirigido**, no qual cada nó (ou vértice) representa uma população ribeirinha e cada aresta representa um canal da bacia hidrográfica. Para cada par de vértices adjacentes (*U, V*), existe uma aresta de *U* para *V* e outra de *V* para *U*, representando o sentido a favor e contra a corrente, respectivamente.

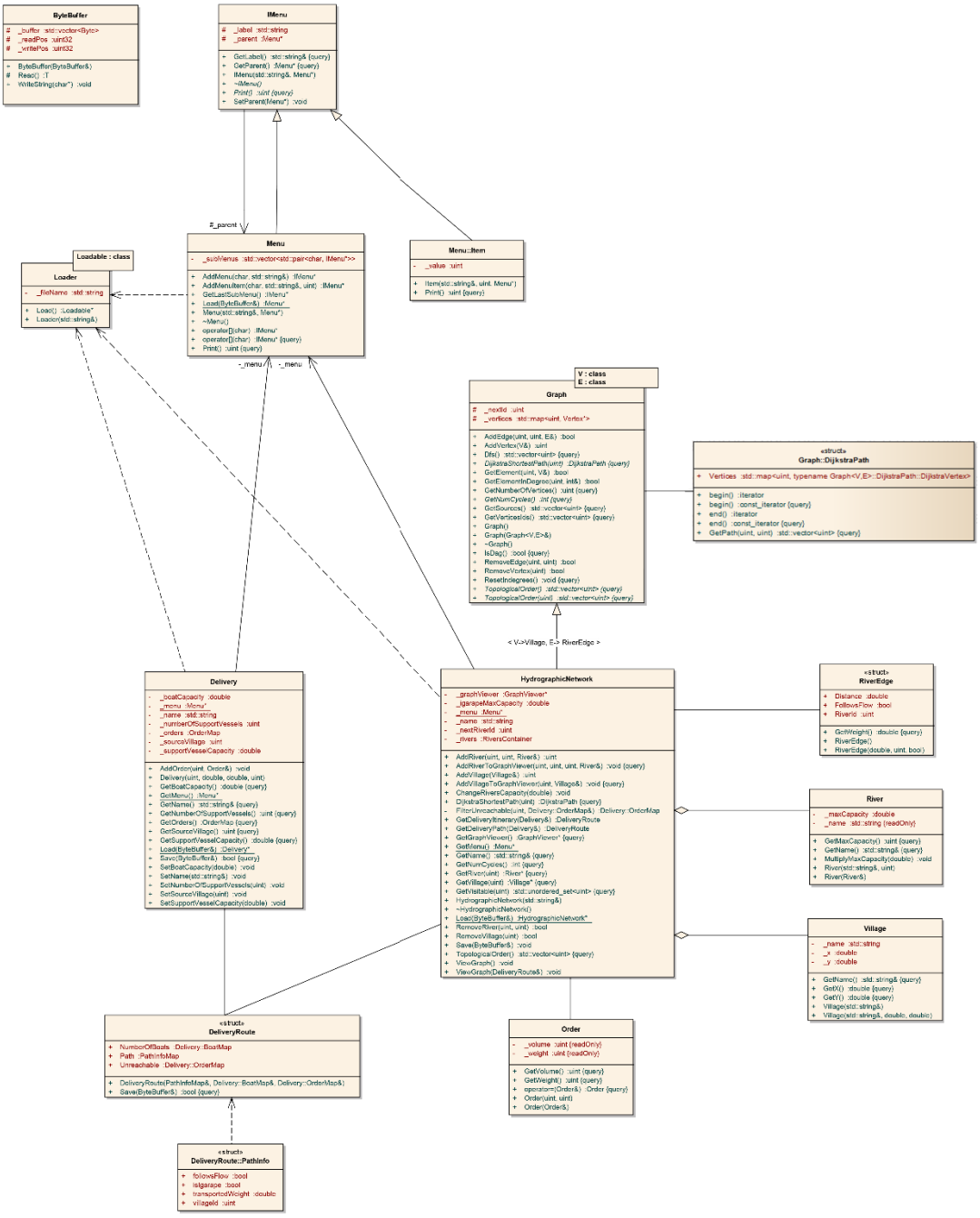
Para cada nó é armazenado o nome da população ribeirinha, assim como as suas coordenadas. Esses dados são encapsulados na classe *Village*.

Para cada aresta é armazenado o nome do rio e a capacidade do mesmo, cujo dados são encapsulados na classe *River*.

A informação gerada pelas funções de cálculo do itinerário é encapsulada na classe *DeliveryRoute*.

A informação relativa à encomenda a processar (fonte, conjunto de encomendas para cada destino, capacidade das balsas, número e capacidade das embarcações de apoio) é encapsulada na classe *Delivery*.

Diagrama UML



Solução

Introdução

Na sua base, o cálculo do itinerário pode ser generalizado para um problema de *routing*. Para cada destino, será necessário calcular o caminho mais curto da fonte até ao mesmo tendo em conta a função de custo.

Apesar da optimização da carga não estar contemplada nos requisitos da aplicação, será também oportuno saber quantas balsas serão necessárias para transportar cada conjunto de encomendas para os vários destinos.

Dado o contexto do problema foram implementadas duas soluções distintas. Uma delas gera uma rota única que cobre todas as encomendas e todos os destinos, ao passo que a outra gera uma rota por cada destino de entrega. Independentemente do resultados que estas duas soluções geram, os dados de entrada são os mesmos:

- Vértice de partida (ponto a partir do qual vão ser enviadas todas as encomendas);
- Conjunto de encomendas associadas a um ou mais destinos;
- Capacidade das balsas (em volume);
- Número de balsas com embarcações de apoio;
- Capacidade das embarcações de apoio (em volume).

Itinerários Múltiplos

Variáveis:

- $V \leftarrow número\ de\ vértices\ do\ grafo$
- $E \leftarrow número\ de\ arestas\ do\ grafo$
- $N \leftarrow número\ de\ destinos$
- $D \leftarrow número\ de\ encomendas\ por\ destino$

Operações:

1. Determinar quais as encomendas não alcançáveis através do algoritmo da Ordenação topológica (modificado)(*breadth-first search)*, tendo em conta a capacidade das balsas. Destinos separados da fonte por canais cuja capacidade é inferior às capacidades das balsas não são atingíveis.
   Complexidade: $O(|V| + |E|)$

2. A partir das encomendas não alcançáveis por balsas normais, determinar quais as encomendas alcançáveis apenas por balsas com embarcações de apoio, recorrendo novamente ao algoritmo da pesquisa em largura.
   Complexidade: $O(|V| + |E|)$

3. Remover encomendas não alcançáveis no ponto anterior da lista de encomendas.
   Complexidade: $O(|N|)$

4. Calcular o volume, peso total e número de barcos necessários por destino.
   Complexidade: $O(|N| * |D|)$

5. Correr o algoritmo de Dijkstra a partir da fonte para todos os vértices do grafo.
   Complexidade: $O(|E| + \log|V|)$

6. Calcular os caminhos da fonte para cada destino. Complexidade: $O(|N| * \log|V|)$.

Itinerário único

Variáveis:

- $V \leftarrow número\ de\ vértices\ do\ grafo$
- $E \leftarrow número\ de\ arestas\ do\ grafo$
- $N \leftarrow número\ de\ destinos$
- $D \leftarrow número\ de\ encomendas\ por\ destino$

À semelhança do método anterior, inicialmente é feita uma triagem das encomendas de forma a averiguar quais as encomendas cujos destinos são atingíveis por balsas com e/ou sem embarcações de apoio.

Operações:

1. Calcular o peso e volume total das encomendas
   Complexidade: $O(|N| * |D|)$.

2. Fazer a ordenação topológica do grafo de forma a obter uma sequência válida de destinos
   Complexidade: $O(|V| + |E|)$.

3. Correr o algoritmo de *Dijkstra* a partir da fonte para todos os vértices do grafo.
   Complexidade: $O(|E| + \log|V|)$.

4. Determinar o destino com a distância mínima à fonte. Esse destino vai ser atendido em primeiro lugar. Complexidade: $O(|N|)$.

5. Com base na ordenação anterior, calcular o caminho da fonte para o último destino passando pelos intermédios, tendo como critério a sequência de destinos gerados pela ordenação topológica, gerando um itinerário único para todas as encomendas.
   Complexidade: $O(N * [|E| + \log V + |V|^2 + N]) \approx O(N * |V|^2)$

As duas últimas operações são gananciosas: em cada passo procuram maximizar o ganho imediato (neste caso, minimizar o custo) entre cada par de pontos do itinerário.

Algoritmos implementados

Dijkstra

Resolve o problema de cálculo do caminho mais curto a partir de uma fonte única.

Para um vértice de origem S do grafo, este algoritmo computa o caminho de custo mínimo entre S e todos os outros vértices. O custo é dado pela função do problema.

Sendo ***C*** o custo entre dois vértices, ***p*** o peso da encomenda, ***f*** o fator de custo da corrente e ***d*** a distância entre dois vértices:

$$C = p * f * d$$

O fator de custo será 1 no sentido da corrente e 2 no sentido contrário à mesma.

Complexidade temporal : a implementação deste algoritmo é feita com base em *heaps* da *STL*[1], pelo que este é executado em tempo $O(|E| * \log|V|)$, sendo *E* o número de arestas e *V* o número de vértices do grafo.

Ordenação topológica (modificado)

Faz a ordenação linear dos vértices do grafo tal que, se existe uma aresta (*V*, *W*) no grafo, então *V* aparece antes de *W*.

Problema: dada a configuração do grafo dirigido, para cada par de vértices adjacentes (*V, U*), existe uma aresta no sentido $V \rightarrow U$ e outra no sentido $U \rightarrow V$. O algoritmo original não funciona para grafos cíclicos.

Solução: arestas cujo sentido seja o contrário da corrente ou cuja capacidade seja inferior à capacidade da embarcação são ignoradas.

Resultado: lista dos vértices por ordem topológica

Complexidade temporal: sendo *V* o número de vértices e *E* o número de arestas do grafo, o algoritmo pode ser executado em tempo $O(|V| + |E|)$ no pior dos casos.

[1] *Standard Template Library*

Pesquisa em largura (modificado)

Dado um vértice de origem *S*, explora-se sistematicamente o grafo descobrindo todos os vértices a que se pode chegar a partir da origem.

Variação adicional: um vértice é visitável a partir de S sse existir pelo menos um caminho composto apenas por arestas cujas capacidades sejam maiores ou iguais à capacidade das balsas.

Solução: poda da pesquisa. A pesquisa em largura é interrompida em ramos que garantidamente não levam a um conjunto de vértices atingíveis, ou seja, em ramos cuja aresta que não suporte a capacidade das balsas.

Complexidade temporal: sendo V o número de vértices e E o número de arestas do grafo, o algoritmo pode ser executado em tempo $O(|V| + |E|)$ no pior dos casos.

Conclusões

Limitações

Apesar da solução para mútiplos itinerários calcular o número de barcos necessários para cada destino, a carga transportada por cada balsa não é maximizada. Tal poderia ser conseguido através de uma aproximação gananciosa ao problema das mochilas múltiplas (*multiple knapsack problem*). No entanto, a maximização da carga por balsa iria aumentar o grau de complexidade do problema, dado que cada balsa poderia transportar encomendas para vários destinos. Seria necessário não só calcular o caminho mais curto da fonte para cada destino, mas também determinar a sequência óptima de destinos de forma a minimizar os custos. Esta operação teria que ser feita para **todas as balsas**.

Existem também situações em que a solução para um itinerário único não gera um percurso óptimo. Dada a natureza gananciosa do algoritmo, este só gera um itinerário tendo em conta a função de custo entre cada par de destinos, não contemplando todos os segmentos alternativos.

A solução passaria por decompor a função de custo f em duas parcelas:

- Uma função $g(x)$ de custo passado, que corresponde à distância conhecida entre o vértice inicial e o atual.
- Uma função $h(x)$ de custo futuro, que corresponde a uma estimativa heurística adimissível da distância entre cada par de vértices.

O desafio iria residir na escolha da função $h(x)$, uma vez que uma estimativa errada para a mesma levará a resultados errados.

Dificuldades encontradas

No âmbito da usabilidade da aplicação, a *API* do *graph viewer* dificultou a representação inteligível dos dados, assim como a interação com o utilizador.

A ambiguidade do enunciado do problema dificultou a identificação das variáveis do mesmo, assim como a definição dos requisitos da aplicação, o que implicou um esforço acrescido de investigação.

Distribuição de tarefas

As tarefas de desenvolvimento da aplicação foram distribuídas de forma equitativa por todos os elementos do grupo.