Lucas Chaves Lima

Projeto e Análise de Algoritmos
Trabalho Prático em Grafos

Trabalho prático de Projeto e Análise de Algoritmos realizado na Universidade Federal de Minas Gerais.

Universidade Federal de Minas Gerais – UFMG

Departamento de Ciência da Computação

Programa de Pós-Graduação

Orientador: Sebastian Urrutia

Belo Horizonte

Outubro de 2016

Sumário

1 Introdução

O problema de planejamento de rotas para robôs tem como objetivo encontrar uma sequência de rotas das quais um robô pode se mover de um ponto inicial até um ponto final, sem que ele colida com os obstáculos encontrados durante o percurso do cenário. O objetivo deste trabalho prático é a implementação de algoritmos para planejamento de rota para múltiplos robôs. Para isso, foram propostos 3 problemas distintos:

- Problemas de caminhos mínimo de robôs
- Problemas de caminhos mínimo para robôs imãs
- Problemas de fluxo em uma ponte

Existem diversas formas de modelagem para os problemas acima, no entanto, esse trabalho prático tem como objetivo a modelagem e solução utilizando grafos.

2 Descrição dos problemas

2.1 Problema 1

Dado um ambiente de entrada com robôs, obstáculos e pontos de destino, o problema consiste em encontrar uma rota em que cada robôs atinja um dos destinos. Além disso, essa rota deve ser executada no menor tempo possível. Existem N robôs no ambiente, para cada robô k $\in$ (1..N), a sua localização é da forma x_k = (i, j), onde i e j representam a linha e a coluna no ambiente respectivamente, as quais determinam a coordenada da células. Os destinos e obstáculos não sofrem alteração a cada iteração. Por outro lado, a cada iteração os robôs podem se movimentar nas oito direções ao seu redor ou permanecer na mesma posição desde que respeitem as seguintes restrições (R1) ele não se mova para uma posição em que há um obstáculo e (R2) dois robôs não esteja na mesma coordenada (x, y) no mesmo instante. Considerando isso, deve-se levar em consideração de que o ambiente altera-se a cada iteração e deve ser controlado para que não haja conflitos.

2.2 Problema 2

O segundo problema pode ser modelado da mesma forma que foi feita a modelagem do problema 1. No entanto, uma nova restrição deve ser levada em consideração: Quando dois robôs estão adjacentes de forma horizontal ou vertical, existe uma espécie de imã que faz com que os robôs fiquem conectados. Quando os robôs estão conectados, estes podem se mover apenas na mesma direção. Isso nos leva ao problema 2 que consiste em escrever um algoritmo que calcule o caminho que cada robô tem que seguir para que todas as conexões sejam feitas no mesmo instante de tempo. Tendo em conta as restricões R1 e R2.

2.3 Problema 3

Um conjunto de células G formam uma estrutura conectada Norte-Sul (ponte). Cada célula g G representa uma célula que suporta um robô por vez. Uma vez na ponte, cada robô pode-se mover para cima, para baixo e nas diagonais. O objetivo do problema é encontrar o número máximo de robôs que podem transitar pela ponte. O que se resume a um problema de fluxo em grafo.

3 Descrição do projeto

3.1 Visão Geral

Para solucionar os problemas propostos, foram implementadas as seguintes soluções:

- Problema 1 - Cálculo do caminho mínimo utilizando estratégia semelhante a BFS.
- Problema 2 - Alterações mínimas no problema 1 para a resolução desse problema.
- Problema 3 - Cálculo do fluxo máximo sobre a ponte utilizando Ford-Fulkerson

3.2 Principais decisões tomadas

Segue abaixo algumas das principais decisões de projeto:

- Os problemas foram desenvolvidos utilizando a linguagem C++, na versão C++11 2011 Standard.
- Foi utilizado para modelagem um grafo direcionado, onde cada posição do arquivo de entrada é um vértice, e cada um dos vizinhos desse vértice são arestas do vértice.
- Utilização de lista de adjacência para armazenamento de ligações entre vértices.
- Buscas de caminhos mínimos realizadas através de estratégias similares ao BFS.
- Lista de adjacência implementada utilizando map como estrutura de dados.
- Utilização de algoritmos de fluxo máximo (Ford-Fulkerson), para resolução do problema 3.
- Utilização de matriz de adjacência no problema 3
- Problema respeita as restrições R1 e R2 impostas pelos requisitos do projeto
- É realizado atualização da lista de adjacência a cada movimento de robô, em seguida realiza-se o BFS do próximo source utilizando a nova Lista de Adjacência.
- Os executáveis deverão ser chamados da seguinte forma: tp1-problema1 ambiente.txt
- A saída do problema 1 segue o seguinte formato: x1: [(4,1), (3,1), (2,2), (2,3), (2,4)]

3.3 Problema 1 - Encontrando caminhos mínimos para robôs

3.3.1 Processamento dos dados

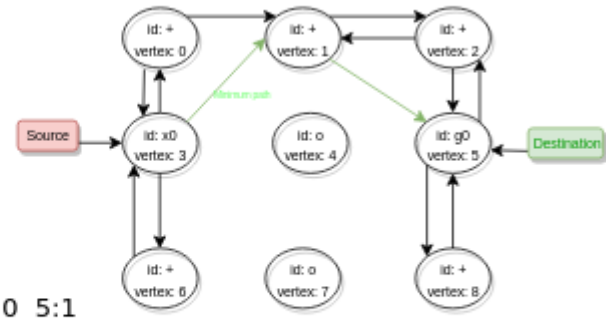
Para realização do processamento dos dados, foi criado uma função readAmbientFile(). Nessa função é feito a leitura dos arquivos de entrada. Foram utilizados grafos para armazenar cada um dos vértices encontrados no arquivo de entrada. Dessa maneira, definimos um novo tipo GraphMap, que consiste de uma map<POS, NODE>. É nessa estrutura que é armazenada o grafo do ambiente processado. Nessa mesma função, criamos estruturas auxiliares para manter um controle de onde está cada um dos robôs e destinos do ambiente, dessa maneira o acesso a esses poderia ser feito de forma mais rápida.

Nessa mesma etapa é realizado o armazenamento de uma queue com os vértices sources (fontes) dos quais serão pontos de partidas do BFS. Dada a entrada a seguir o grafo seria modelado da seguinte maneira. Após a a modelagem em grafos é chamada a função getAdjList, essa função retorna a lista de adjacência do grafo passado como parâmetro. Nesse processo não é adicionado adjacência a arestas obstáculos, e para cada adjacente coloca 1 se é possível alcançar o vértice e 0 caso contrário. Dado os passos vistos acima, um exemplo de processamento inicial dos dados ficaria da seguinte maneira:

Entrada em ambiente.txt

```
+    +    +
x0   o    g0
+    o    +
```

Modelagem em grafos



Lista de adjacência:

```
Vertice:0----> 1:1  3:0
Vertice:1----> 0:1  2:1  3:0  5:1
Vertice:2----> 1:1  5:1
Vertice:3----> 0:1  1:1  6:1
Vertice:5----> 1:1  2:1  8:1
Vertice:6----> 3:0
Vertice:8----> 5:1
```

3.3.2 Realização da busca

Para implementação do cálculo do caminho mínimo foram utilizados os seguintes métodos principais:

- BFS(): Método principal para cálculo dos caminhos mínimos, esse método calcula para cada um dos vértices da sourceFifo o menor caminho até o destino. Para realizar esse cálculo foi utilizado a estratégia Breadth-first search. Intuitivamente, para realizar a busca por esse método você começa pelo vértice raiz e explora todos

os vértices vizinhos. Então, para cada um desses vértices mais próximos, exploramos os seus vértices vizinhos inexplorados e assim por diante, até que ele encontre o alvo destino. Em seguida, é retornado o próximo passo a ser dado pelo Robô, adiciona-se a posição a ser visitada na soourceFifo e atualiza-se a lista de adjacência através do método updateAdjList().

- updateAdjList(): Esse método realiza a atualização da lista de adjacência para cada movimento de algum robô, para isso ele seta 0 nas posições em que não se pode caminhar com o robô e 1 nas posições permitidas.
- getDestination(): Aloca para cada um dos sources um destino mais próximo.
- findnextMove(): Após realizado a busca em largura, vai do destino até a fonte olhando os precessores de maneira a encontrar qual deve ser o próximo passo a ser tomado pelo robô(fonte).

Visto as funções principais do cálculo do caminho mínimo, a estratégia utilizada para resolução do problema 1 funciona da segue os seguintes passos:

1. Chama a função readAmbientFile() para realizar a leitura do arquivo e realizar a modelagem do grafo.
2. Chama a função getAdjList() para realizar o cálculo da lista de adjacência
3. Para cada um dos sources na SourceFifo chama o Método BFS()
4. Realiza o cálculo do caminho mínimo, verifica qual é o próximo passo do robô através da função findnextmove(). Atualiza a lista de adjacência através do método uptadeAdjList(), se o nextMove for o destino, não insere na SourceFifo, caso contrário realiza o push do nextMove.
5. Volte para o passo 3 caso fila não esteja vazia.

Um aspecto a se levar em consideração é que a solução proposta se esbarra no problema do ambiente 3, pois a solução adotada foi de assumir para cada robô um destino fixo, no entanto, quando o destino encontra-se a frente de algum outro destino (e este é o único caminho possível), causa-se deadlock (como é o caso do ambiente 3 dado como exemplo. Para alguns ambientes, algumas das soluções podem variar um pouco em relação ao número de iterações, no entanto, nunca ultrapassando a solução ótima.

3.4 Problema 2 - Caminhos mínimos com robôs imãs

Nesse problema foi adicionado uma terceira restrição. Caso robôs sejam vizinhos, eles estarão sujeitos a 3ª restrição, ou seja, apenas poderão mover-se na mesma direção

no grafo. Para implementação do problema 2, foi feito uma alteração no problema 1 de maneira que quando robôs estivessem em posições adjacentes horizontalmente ou verticalmente eles se tornariam apenas 1 robô, dessa forma caminhariam juntos em movimentos apenas horizontais e verticais. Para isso, a cada iteração verifica se existe um robô em sua vizinhança, caso exista é verificado se o robô moveu-se no mesmo sentido. Nota-se que o número de iterações a serem feitas será menor que o problema 1, pois diminui-se o número de robôs e passos a serem dados.

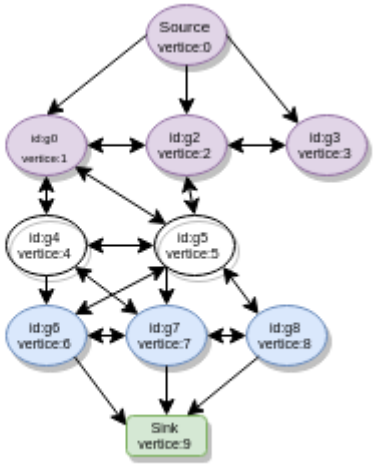
3.5 Problema 3 - Fluxo máximo sobre a ponte

Para realizar a modelagem do problema 3, foi proposto uma estratégia de algoritmo de fluxo máximo. Para isso, primeiramente modelamos o grafo de maneira com que cada caminho possível era um vértice e se um vértice estivesse na vizinhança haveria uma aresta para ele, como cada célula suporta apenas um robô por vez o peso do arco é 1. Essa modelagem pode ser vista a seguir:

Exemplo de ambiente de entrada:

```
+  +  g0  g2  g3  +
+  g4  g5  +  +  +
+  g6  g7  g8  +  +
```

Matriz de Adjacência	0	1	2	3	4	5	6	7	8	9
0	0	1	1	1	0	0	0	0	0	0
1	0	0	1	0	1	1	0	0	0	0
2	0	1	0	1	0	1	0	0	0	0
3	0	0	1	0	0	0	0	0	0	0
4	0	1	0	0	0	1	1	1	0	0
5	0	0	1	0	1	1	1	1	1	0
6	0	0	0	0	0	0	0	1	0	1
7	0	0	0	0	1	1	1	0	0	1
8	0	0	0	0	0	1	0	1	0	1
9	0	0	0	0	0	0	0	0	0	0



Para realizarmos o cálculo do fluxo máximo, foi implementado o algoritmo de ford-fulkerson. Após realizado a modelagem do grafo, como a imagem mostrada acima, utiliza-se o ford-fulkerson que realiza iterações enquanto encontrar melhorias no fluxo. Ele primeiramente realiza uma cópia do grafo que é conhecido como grafo residual, através desse grafo que é possível indicar se é possível adicionar fluxo, isto é, se existe um caminho entre o source e sink é porque é possível adicionar fluxo. Para verificar a existência desse caminho, realiza-se busca em largura, dessa forma é possível verificar se pode-se adicionar fluxo. Quando não é possível adicionar fluxo, dizemos que temos o fluxo máximo, é então retornado como resultado, no nosso problema, o número de carros que podem transitar ao mesmo tempo.

4 Análise de Complexidade

Nesse próximo capítulo é descrito a análise de complexidade, a tabela abaixo apresenta alguns símbolos utilizados na análise:

Tabela 1 – Definição dos símbolos comuns.

Símbolo	Descrição
$\boldsymbol{V}$	Número de vértices.
$\boldsymbol{E}$	Número de arestas
$\boldsymbol{P}$	Número de posições livres
$\boldsymbol{R}$	Número de robôs
$\boldsymbol{K}$	Tamanho do caminho

4.1 Análise do problema 1

Primeiramente, para realizar a leitura do arquivo temos a complexidade de O(V), pois temos que passar por todos os vértices e adicionar ao grafo. Em seguida, realizamos o cálculo da lista de adjacência. Para realização desse cálculo temos a complexidade de O(E), pois temos que verificar todas as arestas do grafo. O próximo passo é calcularmos o caminho mínimo entre a source(robô) e o destino, visto que utilizamos uma busca em largura, teremos que a complexidade da busca pelo caminho mínimo é O(P+E), pois não procuramos para todos os vértices, e sim apenas aqueles em que as posições são livres. Por fim, esse cálculo pelo caminho mínimo é realizado para cada robô em cada uma das posições de seu percurso, o que nos da uma complexidade de tempo final de $O(V+E)^{\mathrm{R*F}}$.

4.2 Análise do problema 2

A análise do problema 2 é similar a análise do problema 1, no entanto, o cálculo será reduzido pois quando dois robôs estão "Conectados", a quantidade de robôs é reduzida e o número de arestas possíveis de movimentação também, pois agora só pode se movimentar horizontalmente e verticalmente, logo para o problema 2 a complexidade é em torno de $O(V+E)^{\mathrm{R*F}}$.

4.3 Análise do problema 3

Primeiramente, para realizar a leitura do arquivo e criação do grafo temos a complexidade de O(V), Em seguida, executa-se o algoritmo Ford-Fulkerson para calcular o

fluxo máximo do grafo. O algoritmo realiza iterações enquanto houver melhorias no fluxo. Para encontrar como ele realiza o caminho dessa melhora, utiliza-se uma busca em largura no grafo residual, visto que foi utilizado matriz de adjacência para o cálculo da busca em largura, temos a complexidade da busca O(V) e resultando na complexidade total do fluxo máximo em O(V) o que é a complexidade total do algoritmo implementado para resolução do problema 3.

5 Conclusão

Esse trabalho teve como principal objetivo a implementação de algoritmos em grafos para a solução de 3 problemas. Com o desenvolvimento dessas modelagens, foi possível aprender ainda mais sobre o funcionamento de algoritmos de busca e fluxo máximo. Um ponto que vale ressaltar é a importância da utilização de boas técnicas de programação e otimização em códigos quando trabalhando com problemas Np-difíceis.

Para resolução dos dois primeiros problemas, utilizou-se busca em largura para para cada robô a cada movimento, o que não é uma estratégia muito eficiente. Para o problema 3, foi apresentado uma solução adaptando o problema para um problema de fluxo máximo, e em seguida utilização de algoritmos de cálculo de fluxo máximo. Portanto, é notável a necessidade de utilização de heurísticas que ajudem a melhorar a eficiência na resolução dos três problemas apresentados.

Algumas melhorias que poderiam ser consideradas nesse trabalho são:

- Utilizar ou implementar heurísticas que ajudem a realizar podas em caminhos evitando buscas em caminhos inúteis.
- Apresentar ou utilizar outros algoritmos mais eficientes de orientação em grafos
- Utilização de lista de adjacência no problema 3 reduziria a complexidade da busca em largura.

Este trabalho cumpriu como objetivo proposto, de colocar o aluno a frente de diferentes modelagens e algoritmos clássicos de grafos, não só implementando, mas também avaliando as possíveis melhorias.