

Universidade de São Paulo

Renan Rodrigues, 9278132

Inteligência Artificial

Métodos de Busca



São Carlos

Agosto/2017

Sumário

1- Introdução	2
2- Métodos de busca	3
2.1- Busca em largura (BFS)	3
2.2- Busca em profundidade (DFS)	3
2.3- Busca Best-First	4
2.4- Busca A*	4
3- Avaliação de tempo	4
4- Execução do código	5
5- Guia de compilação	5
6- Implementação	6

1- Introdução

O projeto teve como função praticar o uso de alguns métodos de busca vistos em sala. Elas são: Busca em Profundidade, Busca em Largura, Buscar Best-First e Busca A*. Com isso, o exercício induz a observação entre os diferentes métodos em relação à complexidade, implementação e, principalmente, o tempo de resposta de cada um.

Para este projeto, como os dados fornecidos na entrada são uma matriz, optei por fazer uma busca sobre ela marcando os elementos já visitados, suas coordenadas, distância do nó atual ao destino e distância do nó atual ao destino somado com a distância do atual ao seu pai. Segue a estrutura usada que me possibilitou isso (*Imagem 1*).

```

//Struct que representa cada posição da matriz de entrada
struct no_ {
    char charac;
    int visited; //0 = não visitado; 1 = visitado
    int coordX;
    int coordY;
    double distToFinal; // h(n) -> Distancia de no de no meta
    double distTotal; //g(n) -> distancia do nó ao nó meta somado com a distancia do nó pai dele
    struct no_* father;
};

struct queue_ {
    int coordX;
    int coordY;
};

```

Imagem 1: estrutura usada na implementação

2- Métodos de busca

2.1- Busca em largura (BFS)

Esse algoritmo consiste em percorrer um grafo passando, primeiramente, por todos os filhos do nó em que se está sendo executado.

Neste caso, foi utilizado um vetor “queue” tratado como uma lista para poder ser feito o caminho pelos nós filhos.

2.2- Busca em profundidade (DFS)

Este algoritmo consiste na busca de um nó percorrendo, primeiramente, todos os filhos mais a esquerda de um nó e depois os a direita.

Para a implementação desse caso, foi utilizada um vetor “stack” tratado como uma pilha para poder ser feita as manipulações necessárias para que o caminho fosse correto.

2.3- Busca Best-First

Esse algoritmo é o mais simples em relação aos algoritmos de busca informada. Ele consiste em usar uma heurística simples de saber a distância de cada nó até o nó destino. Com isso, ele percorre o grafo sempre no caminho do nó com menor peso.

No caso da implementação desenvolvida foi utilizado como heurística a Distância de Manhattan, que seria basicamente: $|x_1 - x_2| + |y_1 - y_2|$.

Não houve necessidade de algum vetor para alocação das distâncias, pois a própria estrutura base do projeto (mostrada na Introdução) já possui esse valor.

2.4- Busca A*

Esse algoritmo também é um tipo de busca informada, porém aplicado uma heurística a mais para refinar a busca.

Para a minha implementação foi utilizado uma heurística simples da Distância de Manhattan somado à distância de cada nó ao seu nó pai, podendo ser esse valor: um (1) ou raiz quadrada de 2. Como o problema não possui grandes pontos de dificuldade na busca, essa heurística foi utilizada para separar algum caso que possa ter empate no caso do Best-First.

3- Avaliação de tempo

Foram feitos alguns teste de comparação de tempo entre os algoritmos que levaram à conclusão que:

* Embora todos algoritmos sejam de certa maneira rápidos, os algoritmos de busca informada, principalmente o A* para esse projeto, levam uma vantagem.

* Como foram feitos teste em alguns exemplos mais limitados, nota-se que há muita variação no quesito de comparação entre os algoritmos, pois para alguns casos testes um algoritmo pode levar menos tempo que os outros.

* Em relação às buscas informadas a heurística usada para refinar suas buscas também interferem fortemente nesse tempo.

4- Execução do código

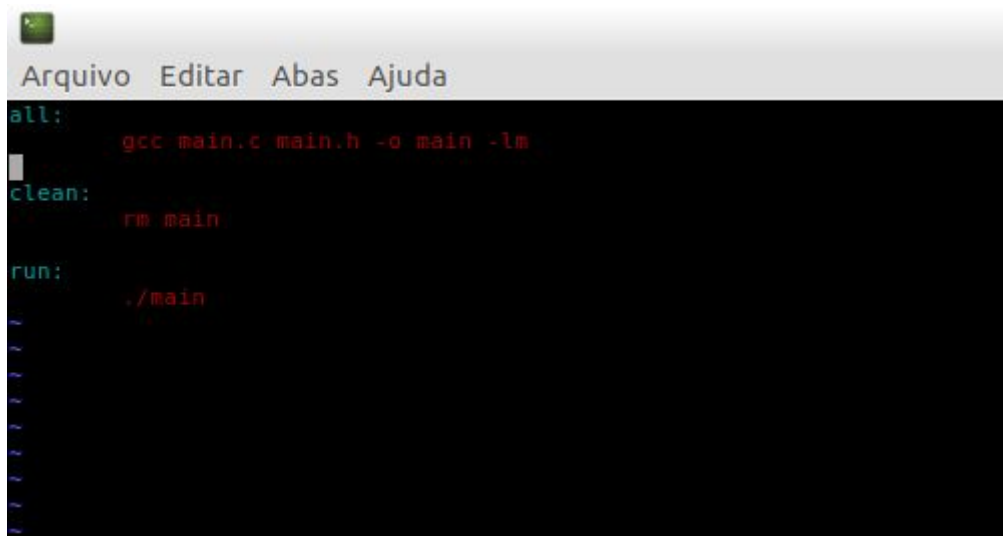
O código possui uma interface básica de menu para facilitar na escolha do algoritmo a ser executado (*Imagem 2*). O menu pode ser utilizado apenas uma vez por execução do código. Há junto com os demais arquivos um exemplo básico de matriz de entrada no arquivo “file.txt”. Ao executar o programa é necessário o nome de um arquivo com sua extensão.

```
renan@renan-VirtualBox:~/ia/proj1$ make all
gcc part1.c test.h -o part1 -lm
renan@renan-VirtualBox:~/ia/proj1$ make run
./part1
Insira o nome do arquivo txt (com a extensao) : file.txt
Arquivo aberto com sucesso.
Escolha uma forma de busca:
1 - BFS (busca em largura).
2 - DFS (busca em profundidade).
3 - Best First.
4 - A Estrela (A*).
```

Imagem 2: menu para interação com o usuário

5- Guia de compilação

Para realizar a compilação do código, é utilizado um simples Makefile (*Imagem 3*) disposto junto ao .zip do projeto. Esse arquivo possui os comandos para compilar (make all), executar (make run) o algoritmo e excluir o executável (make clean).



```
Arquivo  Editar  Abas  Ajuda
all:
    gcc main.c main.h -o main -lm
clean:
    rm main
run:
    ./main
```

Imagem 3: Makefile utilizado para compilar, executar e excluir

6- Implementação

A implementação segue um caminho simples de entender com várias funções para modelar melhor a leitura e entendimento do código. O projeto também conta um arquivo de cabeçalho onde estão todas as declarações de variáveis definidas, funções e bibliotecas.

A estrutura “queue_” vista na *Imagem 1* é utilizada como uma lista dos nós que fazem parte do caminho do nó origem ao nó meta. Ela é usada apenas na função de imprimir essa lista (*printCoords*).

A função que é o ponto chave da implementação é a “*checkPositions*”. De forma simplificada, ela é responsável por receber as coordenadas de um nó e a partir dele verificar quais são os seus nós vizinhos e, para cada nó encontrado, marcá-lo como visitado (para que outros nós não o visite novamente), salvar suas coordenadas e marcar o nó atual como pai deles. Desse jeito é possível percorrer os nós (do nó meta ao nó inicial de busca) que fazem parte da resposta e inseri-los na lista da estrutura *queue* para representar a resposta.