ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

INSTITUTO POLITÉCNICO DA GUARDA

PROCURA SOFREGA

TRABALHO PRÁTICO N.º 4

Curso (s)	Engenharia Informática
Unidade (s) Curricular (es)	Inteligência Artificial
Ano Letivo	2017/2018
Docente	Celestino Gonçalves
Aluno	André Madeira || 1010066

Índice

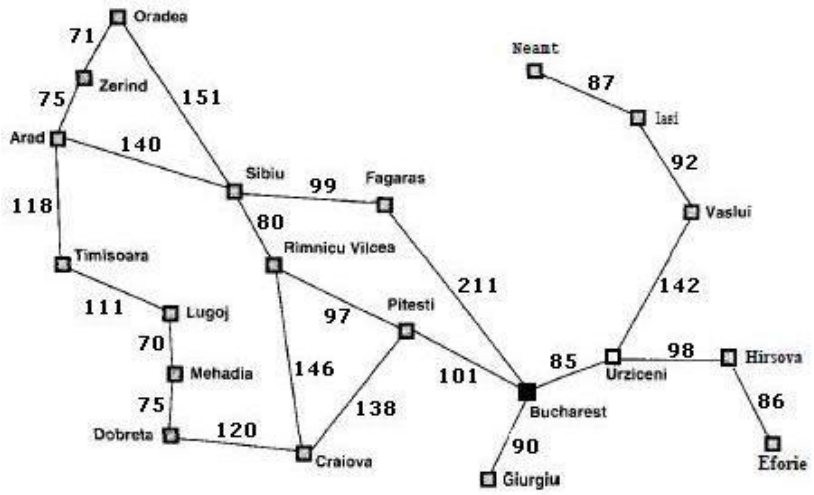
Índice Ilustrações

Introdução

“Na procura sôfrega o princípio consiste em escolher o nó na fronteira da árvore de procura que aparenta ser o mais promissor de acordo com o valor estimado por h(n).
Assim, o algoritmo limita-se a manter a fronteira da árvore de procura ordenada pelos valores de h(n)(valores da distancia em linha reta), sendo sempre escolhido o nó de valor mais baixo, ou seja, aquele que está hipoteticamente mais próximo da solução.”

Problema

Dada uma cidade de partida calcular o caminho até Bucharest;
Apresentar em cada interação a fronteira (nós não explorados);
Indicar, o caminho calculado e também o custo (em Km) do mesmo segundo os valores do mapa seguinte.



Distância em linha reta até Bucharest (Km)	
Arad	366
Bucharest	0
Craiova	160
Dobreta	242
Eforie	161
Fagaras	178
Giurgiu	77
Hirsova	151
Iasi	226
Lugoj	244
Mehadia	241
Neamt	234
Oradea	380
Pitesti	98
Rimnicu Vilcea	193
Sibiu	253
Timisoara	329
Urziceni	80
Vaslui	199
Zerind	374

Figura 1 Mapa Bucharest

Algoritmo

1. Colocar o nome da cidade de partida numa variável;
2. Verificar se a variável contem o nome da cidade de destino;
    2. Se for a cidade de destino saltar para o ponto 9 se não continua;
3. Adicionar as cidades que lhe estão ligadas a uma coleção (fronteira);
4. Ordenar a coleção por ordem crescente do valor H(n);
5. Mostrar no ecrã o nome das Cidades na coleção (fronteira);
6. Atribuir o nome da Cidade de menor H(n) à variável criada;
7. Remover essa Cidade da coleção (fronteira);
8. Repetir do ponto 2;
9. Mostrar no ecrã o caminho e custo finais resultantes.

Breve descrição do código

Foi criada a classe "Cidade" para poder ter objetos com um nome e duas distancias H(n) e G(n)) a ele associados. Sendo que G(n) é apenas utilizada para calculo do custo final e H(n) para a distancia em linha reta.
Cada cidade esta, ainda representada no código como uma lista de cidades. Para poder ser interpretada como um nó. Correspondendo o nome da lista ao nome da cidade em questão e o conteúdo da lista ás cidades com as quais esta tem ligação, como vemos no exemplo abaixo:

```
static  ArrayList<Cidade> Zerind=new ArrayList<>();
        Zerind.add(new Cidade("arad",366,75));
        Zerind.add(new Cidade("oradea",380,71));
```

Funções:

- **EscolherLista()** - dado um nome duma cidade devolve a respetiva lista (nó);
- **AdicionarLista() -** permite adicionar novos nós (cidades) à fronteira;
- **OrdenarLista() -** recebe uma lista de cidades e devolve uma lista com as mesmas cidades ordenadas por valores crescentes de H(n);
- **MostrarFronteira()** - recebe uma lista de cidades (a fronteira em cada iteração) e coloca no ecrã o nome e valor de H(n) correspondente;
- **MostrarCaminho()** - coloca no ecrã a sequencia de cidades que constituem o caminho calculado.

Exemplo Teste Aplicação

Vamos testar a aplicação querendo saber o caminho de Arad a Bucharest através da procura sôfrega:

```
run:
Lista Cidades:

Neamt
Lasi
Vaslui
Eforie
Hirsova
Urziceni
Giurgiu
Fagaras
Pitesti
Rimnicu
Mehadia
Lugoj
Timissoara
sibiu
oradea
Zerind
Arad

Introduzao nome da cidade de partida
 "1" para sair!
Arad
0- (arad)

1-(sibiu,253) (timisoara,329) (zerind,374)

2-(fagaras,178) (rimnicu,193) (timisoara,329) (arad,366) (zerind,374) (oradea,380)

3-(bucharest,0) (rimnicu,193) (sibiu,253) (timisoara,329) (arad,366) (zerind,374) (oradea,380)

4-(rimnicu,193) (sibiu,253) (timisoara,329) (arad,366) (zerind,374) (oradea,380)

Caminho: arad -> sibiu -> fagaras -> bucharest
Custo: 450
```

Figura 2 Teste Arad até Bucharest

Código

```
package sôfrega;

import java.util.ArrayList;
import java.util.Scanner;

/**
 *
 * @author André Madeira
 */
public class Sofrega {

    /**
     * @param args the command line arguments
     */

    static  ArrayList<Cidade> Arad=new ArrayList<>();
    static  ArrayList<Cidade> Zerind=new ArrayList<>();
    static  ArrayList<Cidade> Oradea=new ArrayList<>();
    static  ArrayList<Cidade> Sibiu=new ArrayList<>();
    static  ArrayList<Cidade> Timisoara=new ArrayList<>();
    static  ArrayList<Cidade> Lugoj=new ArrayList<>();
    static  ArrayList<Cidade> Mehadia=new ArrayList<>();
    static  ArrayList<Cidade> Dobreta=new ArrayList<>();
    static  ArrayList<Cidade> Craiova=new ArrayList<>();
    static  ArrayList<Cidade> Rimnicu=new ArrayList<>();
    static  ArrayList<Cidade> Pitesti=new ArrayList<>();
    static  ArrayList<Cidade> Fagaras=new ArrayList<>();
    static  ArrayList<Cidade> Giurgiu=new ArrayList<>();
    static  ArrayList<Cidade> Urziceni=new ArrayList<>();
    static  ArrayList<Cidade> Hirsova=new ArrayList<>();
    static  ArrayList<Cidade> Eforie=new ArrayList<>();
    static  ArrayList<Cidade> Vaslui=new ArrayList<>();
    static  ArrayList<Cidade> Lasi=new ArrayList<>();
    static  ArrayList<Cidade> Neamt=new ArrayList<>();
    static ArrayList<Cidade> Caminho = new ArrayList();


    public static ArrayList<Cidade> OrdenarLista(ArrayList<Cidade>
Lista){
```

```
        //recebe uma lista de cidades e devolve uma lista com as mesmas
cidades ordenadas por valores crescentes de H(n)(distancia em linha reta);

        ArrayList<Cidade> Arrayordenado= new ArrayList<>();
        Cidade Aux;
        Cidade Melhor;
        int Indice;
        while(Lista.size()>0){
            Melhor=Lista.get(0);
            Indice=0;
            for(int x=0;x<Lista.size();x++){
                Aux=Lista.get(x);
                if(Aux.Disth<Melhor.Disth){
                    Melhor=Aux;
                    Indice=x;
                }
            }
            Lista.remove(Indice);
            Arrayordenado.add(Melhor);

        }
        return Arrayordenado;

    }



    public static ArrayList<Cidade> EscolherLista(String cidade){
        ArrayList<Cidade> aux= new ArrayList<>();
        switch (cidade){
            case "arad":
                aux = Arad;
                break;
            case "zerind":
                aux= Zerind;
                break;
            case "oradea":
                aux=Oradea;
                break;
            case "sibiu":
                aux=Sibiu;
                break;
            case "timisoara":
```

```
    aux=Timisoara;
    break;
case "lugoj":
    aux=Lugoj;
    break;
case "mehadia":
    aux=Mehadia;
    break;
case "dobreta":
    aux=Dobreta;
    break;
case "craiova":
    aux=Craiova;
    break;
case "rimnicu":
    aux=Rimnicu;
    break;
case "pitesti":
    aux=Pitesti;
    break;
case "fagaras":
    aux=Fagaras;
    break;
case "giurgiu":
    aux=Giurgiu;
    break;
case "urziceni":
    aux=Urziceni;
    break;
case "hirsova":
    aux=Hirsova;
    break;
case "eforie":
    aux=Eforie;
    break;
case "vaslui":
    aux=Vaslui;
    break;
case "lasi":
    aux=Lasi;
    break;
case "neamt":
    aux=Neamt;
```

```
                break;
        }

        return aux;
    }

    public static void AdicionarLista(ArrayList<Cidade>
ListaAdicionar,ArrayList<Cidade> FronteiraNova){

        //permite adicionar novos nós (cidades) à fronteira;
        Cidade cidade;
        for(int x=0;x<ListaAdicionar.size();x++){
            cidade=ListaAdicionar.get(x);
            FronteiraNova.add(cidade);
        }
    }

    public static  void MostrarFronteira(ArrayList<Cidade> Lista){
       // recebe uma lista de cidades (a fronteira em cada iteração) e coloca no
ecrã o nome e valor do custo correspondente;

        for(int x=0;x<Lista.size();x++){

            System.out.print("("+Lista.get(x).Nome+","+Lista.get(x).Disth+")
");
        }
        System.out.print("\n");
    }
    public static  void MostrarCaminho(ArrayList<Cidade> Lista){

        //coloca no ecrã a sequencia de cidades que constituem o caminho
calculado.
        System.out.print("\nCaminho: ");
        for(int x=0;x<Lista.size();x++){
            System.out.print(Lista.get(x).Nome);
            if (x<Lista.size()-1)
                System.out.print(" -> ");
        }
    }

    public static void main(String[] args) {
        // cidade , disth(distancia em linha reta), distg(deistancia ente nos)
```

```
Arad.add(new Cidade("timisoara",329,118));
Arad.add(new Cidade("sibiu",253,140));
Arad.add(new Cidade("zerind",374,75));
Zerind.add(new Cidade("arad",366,75));
Zerind.add(new Cidade("oradea",380,71));
Oradea.add(new Cidade("zerind",374,71));
Oradea.add(new Cidade("sibiu",253,151));
Sibiu.add(new Cidade("arad",366,140));
Sibiu.add(new Cidade("oradea",380,151));
Sibiu.add(new Cidade("fagaras",178,99));
Sibiu.add(new Cidade("rimnicu",193,80));
Timisoara.add(new Cidade("arad",366,118));
Timisoara.add(new Cidade("lugoj",244,111));
Lugoj.add(new Cidade("timisoara",329,111));
Lugoj.add(new Cidade("mehadia",241,70));
Mehadia.add(new Cidade("lugoj",244,70));
Mehadia.add(new Cidade("dobreta",242,75));
Dobreta.add(new Cidade("mehadia",241,75));
Dobreta.add(new Cidade("craiova",160,120));
Craiova.add(new Cidade("dobreta",242,120));
Craiova.add(new Cidade("pitesti",98,138));
Craiova.add(new Cidade("rimnicu",193,146));
Rimnicu.add(new Cidade("sibiu",253,80));
Rimnicu.add(new Cidade("craiova",160,146));
Rimnicu.add(new Cidade("pitesti",98,97));
Pitesti.add(new Cidade("rimnicu",193,97));
Pitesti.add(new Cidade("craiova",160,138));
Pitesti.add(new Cidade("bucharest",0,101));
Fagaras.add(new Cidade("sibiu",253,99));
Fagaras.add(new Cidade("bucharest",0,211));
Giurgiu.add(new Cidade("bucharest",0,90));
Urziceni.add(new Cidade("bucharest",0,85));
Urziceni.add(new Cidade("hirsova",151,98));
Urziceni.add(new Cidade("vaslui",199,142));
Hirsova.add(new Cidade("eforie",161,86));
Hirsova.add(new Cidade("urziceni",80,98));
Eforie.add(new Cidade("hirsova",151,86));
Vaslui.add(new Cidade("urziceni",80,142));
Vaslui.add(new Cidade("lasi",226,92));
Lasi.add(new Cidade("neamt",234,87));
Lasi.add(new Cidade("vaslui",199,92));
Neamt.add(new Cidade("lasi",226,87));
```

```
        boolean terminar = false;
        while (terminar==false){
            System.out.print("Lista
Cidades:\n\nNeamt\nLasi\nVaslui\nEforie\nHirsova\nUrziceni\nGiurgiu\nF
agaras\nPitesti\nRimnicu\nMehadia\nLugoj\nTimissoara\nsibiu\noradea\nZ
erind\nArad\n"
                    + "");
            System.out.print("\nIntroduzao nome da cidade de partida\n \"1\"
para sair!\n");
            // vai buscar o valor que o utilizador introduz
            Scanner scanIn = new Scanner(System.in);
            String input= scanIn.nextLine();
            //se o valor for iguala 1 termina o programa
            if ("1".equals(input)){
            terminar= true;
               }
            else{

               String cidade =input.toLowerCase();
               //inicia a variavel cidade a 0
               Cidade inicio = new Cidade(cidade,0,0);
               //inicia a variavel custo a 0
               int custo=0;
               Caminho.add(inicio);
               boolean destino=false;
               //inicia a variavel iteração a 0
               int iteracao=0;

               ArrayList<Cidade> Fronteira= new ArrayList();

               System.out.print(iteracao+"- (");
               System.out.print(cidade);
               System.out.print(") \n\n");
               //enquanto não chegar ao destino = false
               while(destino==false){
                  iteracao++;
                  //se a cidade = budapeste termina o ciclo
                  if(cidade.equals("bucharest")){
                     destino=true;
                    System.out.print(iteracao+"-");
                     MostrarFronteira(Fronteira);
```

```
                    //coloca no ecrã a sequencia de cidades que constituem o
caminho calculado
                    MostrarCaminho(Caminho);
                    //mostra o custo do caminho
                    System.out.println("\nCusto: "+custo);

                }else{
                  //  permite adicionar novos nós (cidades) à fronteira;
                    AdicionarLista(EscolherLista(cidade), Fronteira);
                    Fronteira=OrdenarLista(Fronteira);
                 //recebe uma lista de cidades e devolve uma lista com as
mesmas cidades ordenadas por valores crescentes de H(n);

                    cidade=Fronteira.get(0).Nome;
                    //adiciona ao caminho a cidade com menor valor de
distancia a fronteira
                    Caminho.add(Fronteira.get(0));

                    System.out.print(iteracao+"-");
                //    MostrarFronteira() - recebe uma lista de cidades (a
fronteira em cada iteração) e coloca no ecrã o nome e valor de H(n)

                    MostrarFronteira(Fronteira);
                    System.out.print("\n");
                 //   adiciona ao custo o valor de dist g distancia do nó
                    custo+=Fronteira.get(0).Distg;
                    //remove o a primeira cidade na lista da fronteira ou seja a
com menor valor
                    Fronteira.remove(0);

                }


            }
        }
    }
  }

}
```

Conclusão

Podemos concluir que a procura sôfrega não garante a melhor solução, mas caso haja solução ela é encontrada.
Assim torna-se um algoritmo ideal para resolução de problemas em que a rapidez de resolução é mais valiosa que a otimização da solução.