Fundação CECIERJ - Vice Presidência de Educação Superior a Distância

Curso de Tecnologia em Sistemas de Computação
AD2 de Programação III
2° semestre de 2012

Nome:
Matrícula:
Pólo:

Exercício (ENTREGAR OS ARQUIVOS EM MÍDIA, PARA FINS DE TESTE, JUNTAMENTE COM A AD IMPRESSA):
Considere que sua empresa de software seja contratada pelos Correios para resolver o problema de entrega de correspondências. Este problema consiste na procura de um caminho que, começa numa cidade de origem (isto é, a cidade onde as correspondências estão armazenadas) já definida, dentre várias, visita cada cidade SOMENTE uma vez e regressa à cidade inicial.

Uma maneira de resolver este problema é, a partir da cidade de origem, descobrir qual é a cidade mais próxima (CMP), isto é, a cidade com o menor custo até a origem. Neste momento, sua cidade de origem passa a ser a CMP e repete-se o processo. A restrição existente neste algoritmo é proibir o retorno a cidades já visitadas, a fim de evitar ciclos.

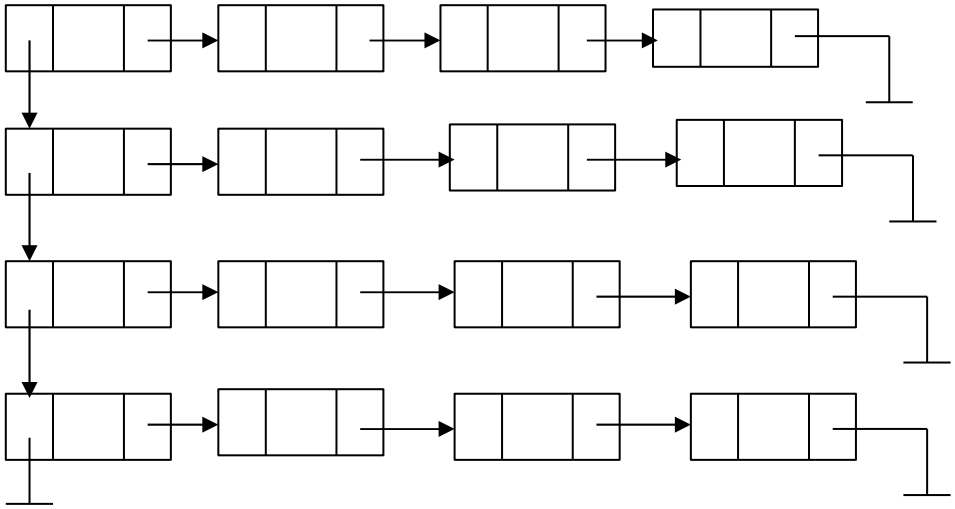
Seu programa deve receber, como parâmetro de entrada, um arquivo contendo distâncias entre cidades e a cidade de origem e deve retornar um arquivo com o caminho a ser seguido, juntamente com o custo do caminho. Um EXEMPLO de arquivo de entrada neste formato seria (OBSERVE QUE SEU PROGRAMA DEVE FUNCIONAR PARA QUALQUER ARQUIVO NO FORMATO ANTERIORMENTE DEFINIDO):

```
RJ/SP/700
RJ/VITORIA/500
VITORIA/SP/400
SP/BH/200
BH/VITORIA/2000
BH/RJ/1000
RJ
```

Observe que existe, para qualquer cidade do arquivo, distâncias associadas as demais. Logo após a leitura, o caminho deve ser calculado e gravado no arquivo **resp-<nome do arquivo de entrada>**. Para o exemplo acima, o arquivo de resposta seria composto do caminho **RJ → ES → SP → BH → RJ**, com custo **2100**.

Por questões de desempenho, seu programa deve ler o arquivo de entrada SOMENTE uma vez.

RESPOSTA:
Uma das maneiras de estruturar estes dados é usar listas encadeadas da seguinte forma:



A estrutura com duas setas está sendo chamada na solução de lista. A estrutura com uma única seta está sendo chamada de Vizinho nesta implementação. O Grafo é somente a referência para o primeiro retângulo da esquerda.

```
import java.io.*;
/*
A estrutura Vizinho é composta de:
  - nome da cidade vizinha
  - distância da cidade vizinha
  - referência para a próxima cidade vizinha (para fazer o encadeamento
de cidades vizinhas)

Esta estrutura é montada para ser usada na lista de vizinhos (próxima
estrutura criada para resolver o problema).

Esta classe é formada de construtor, getDist, getCid e método toString.
O último é usado para verificar se a estrutura está sendo criada de
maneira correta.
*/

class Vizinho{
  String cid_viz;
  float dist;
  Vizinho prox;

  Vizinho(String c, float d){
    cid_viz = c;
    dist = d;
    prox = null;
  }

  String getCid(){ return cid_viz;}

  float getDist(){ return dist; }
```

```
  public String toString(){
    String resp = cid_viz + "\t" + dist + "\n";
    return resp;
  }
}

/*
A estrutura Lista é composta de:
  - nome da cidade de origem
  - a lista de suas vizinhas
   –  a proxima cidade (para fazer o encadeamento de cidades)

Esta estrutura é montada para ser usada na lista que engloba todas as
cidades, chamada de grafo (estrutura principal criada para resolver o
problema).

Esta classe tem o método construtor, um método para testar se o vizinho
já está na lista, um método para inserir vizinho na primeira posição da
lista de vizinhos, e o método toString.
*/

class Lista{
  String cid;
  Vizinho prox_viz;
  Lista prox_cid;

  Lista(String c){
    cid = c;
    prox_viz = null;
    prox_cid = null;
  }

  Vizinho pertence(String cid){
    Vizinho resp = prox_viz;
    while((resp != null) && (!(cid.equals(resp.cid_viz))))
      resp = resp.prox;
    return resp;
  }

  void ins_Viz(String c, float d){
    Vizinho v = pertence(c);
    if(v != null){
      v.dist = d;
      return;
    }
    v = new Vizinho(c, d);
    v.prox = prox_viz;
    prox_viz = v;
  }

  public String toString(){
    String resp = cid + "\n";
    Vizinho p = prox_viz;
```

```
    while(p != null){
      resp += p.toString();
      p = p.prox;
    }
    return resp;
 }
}

/*
A estrutura Grafo é desenvolvida para resolver o problema. Ela é
composta da referência para o primeiro nó da lista. Tem os seguintes
métodos:
  - construtor;
  - para verificar se uma cidade existe na lista de cidades;
   - para inserir todas as cidades de origem da lista, incluindo as
cidades vizinhas; e
  - toString.
*/

class Grafo{
  Lista prim;

  Grafo(){ prim = null; }

  Lista pertence(String cid){
    Lista resp = prim;
    while((resp != null) && (!(cid.equals(resp.cid))))
      resp = resp.prox_cid;
    return resp;
  }

  void insere(String cid1, String cid2, float d){
    Lista p = pertence(cid1);
    if(p == null){
      p = new Lista(cid1);
      p.prox_cid = prim;
      prim = p;
    }
    p.ins_Viz(cid2, d);
    Lista q = pertence(cid2);
    if(q == null){
      q = new Lista(cid2);
      q.prox_cid = prim;
      prim = q;
    }
    q.ins_Viz(cid1, d);
  }

  public String toString(){
    String resp = "";
    Lista p = prim;
    while(p != null){
      resp += p.toString();
      p = p.prox_cid;
    }
    return resp; } }
```

```
public class AD2_2012_2{
  public static void main(String[] args) throws IOException{
    BufferedReader in = new BufferedReader(new FileReader(args[0]));
    try {
      Grafo g = new Grafo();
      String ini = "", s, vs[];

      //leitura do arquivo de entrada uma única vez
      //leitura da cidade de origem (ini)
      while((s = in.readLine()) != null){
      vs = s.split("/");
        if(vs.length == 3)
          g.insere(vs[0], vs[1], Float.parseFloat(vs[2]));
        else
          ini = vs[0];
      }

      in.close();

      //geração do nome do arquivo de saída
      String nome_arq_saida = "resp-" + args[0];

      //chamada da função que resolve o problema.
      //O caminho começa em ini passa por todas
      //as outras cidades e termina em ini
      guloso(g, ini, nome_arq_saida);
    }
    catch (Exception e){
      System.out.println("Excecao\n");
    }
  }

  //função que conta a quantidade de cidades do grafo
  static int conta (Grafo g){
    int resp = 0;
    Lista p = g.prim;
    while(p != null){
      resp++;
      p = p.prox_cid;
    }
    return resp;
  }

  //função que testa se a cidade a ser inserida no arquivo
  //já está no arquivo de resposta
  static boolean pertence(String nome, String[] cidades){
    int i = 0;
    while((i < cidades.length) && (cidades[i] != null))
      if(nome.equals(cidades[i])) return true;
      else i++;
    return false;
  }
```

```
  //função que indica a maior distância de uma cidade
  //a cidade indicada por l
  static float maiorDist(Lista l){
    float maior = l.prox_viz.getDist();
    Vizinho v = l.prox_viz;
    while(v != null){
      if(maior < v.getDist()) maior = v.getDist();
      v = v.prox;
    }
    return maior;
  }

  //função que gera o arquivo de saída e o valor da menor distância
    static void guloso(Grafo g, String ini, String nome) throws
IOException{
    try{
      BufferedWriter out;
      out = new BufferedWriter(new FileWriter(nome));

      //escrita do nome da cidade de origem ini
      out.write(ini + " -> ");

      int i = 1, n = conta(g);

      //vetor usado para guardar todas as cidades já visitadas
      //para evitar ciclos
      String[] cidades = new String[n];
      cidades[0] = ini;

      //variável para guardar o total
      float total = 0.0F;

      Lista l;
      do{
        //saindo-se de g, chega-se a lista de vizinhos de ini
        l = g.pertence(ini);
        Vizinho p = l.prox_viz;

        //descobre-se qual é a maior distância de ini a qualquer
        //outra cidade
        float menor_dist = maiorDist(l);
        String menor_cid = "";

        while(p != null){
          //se a distância entre a menor cidade encontrada
          //até o momento é maior que uma outra distância,
          //e a cidade ainda não está na resposta, seleciona-se
          //esta distância como a menor e guarda-se o nome da cidade
              if((menor_dist >= p.getDist()) && (!pertence(p.getCid(),
cidades))){
            menor_cid = p.getCid();
            menor_dist = p.getDist();
          }
          p = p.prox;
        }
```

```
        //acumula a distância na somatória total
        total += menor_dist;

        //guarda-se o nome da cidade no vetor de
        //cidades já viistadas e grava-se este
        //nome no arquivo de saída
        ini = menor_cid;
        cidades[i++] = ini;
        out.write(ini + " -> ");
      } while(i <= (n - 1));

      //por fim, guarda-se a cidade de início (ini) e
      //a distância da última cidade até ini. Atualizar
      //o valor das distâncias da última cidade até ini.
      out.write(cidades[0]);
      total += g.pertence(ini).pertence(cidades[0]).getDist();
      out.close();
      System.out.println(total);
    }
    catch (Exception e){
      System.out.println("Excecao\n");
    }
  }
}
```