Universidade de São Paulo – Escola de Artes, Ciências e Humanidades

Bacharelado em Sistemas de Informação

Introdução à Ciência da Computação II

Professores Luciano Digiampietri e Fábio Nakano

Data de entrega: 20/10/2012

EXERCÍCIO PROGRAMA 2

Problema do Caminho Hamiltoniano

Neste trabalho você deverá implementar/completar uma classe que encontre a **solução ótima** para o problema do Caminho Hamiltoniano.

O problema do Caminho Hamiltoniano consiste em encontrar o menor caminho que passe por todas as cidades, dados: (a) um mapa com um conjunto de cidades e a distância entre cada cidade e suas vizinhas (b) neste problema específico, uma cidade de origem e uma cidade de destino..

O objetivo é identificar o menor caminho que parta da cidade de origem, chegue na cidade de destino passando exatamente uma vez por todas as cidades do mapa. A solução ótima para este problema pode ser encontrada utilizando a estratégia de *backtracking* e tem complexidade exponencial no número de cidades. Você deve implementar a solução empregando essa estratégia.

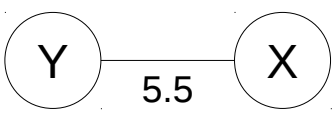
A título de informação (neste momento), sabe-se que soluções exponenciais tem desempenho pobre, no sentido de que solucionar problemas relativamente pequenos toma mais tempo que o aceitável e que aumentar a capacidade de processamento (por exemplo 10 ou 100 vezes) permite tratar problemas com uns poucos elementos a mais. É possível evitar uma implementação com complexidade exponencial utilizando a estratégia gulosa para a solução deste problema, infelizmente esta solução nem sempre será ótima. Por isso, neste EP será utilizada a estratégia de *backtracking* de forma a sempre se obter soluções ótimas.

Para este problema, sempre que houver mais de uma solução com distância mínima, você poderá escolher entre qualquer uma dessas soluções ótimas.

O método para identificar a solução ótima deverá retornar um arranjo de objetos do tipo Cidade, indicando qual será o caminho percorrido (a primeira cidade deverá ser a origem e a última a cidade destino).

A seguir são apresentados alguns exemplos (a numeração corresponde aos mapas que já estão disponíveis no código do EP2). Os caminhos serão apresentados como sequências dos nomes das cidades e entre o nome de duas cidades aparecerá a distância entre elas.

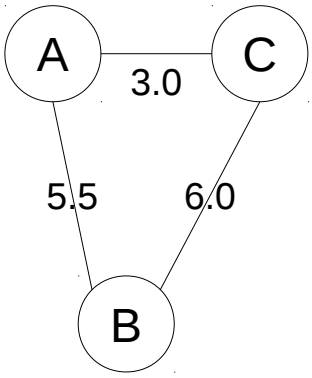
Exemplo 1:



O menor caminho hamiltoniano de X para Y será: **Y [5.5] X**. Distância do caminho: 5.5

O menor caminho hamiltoniano de X para Y será: **X [5.5] Y**. Distância do caminho: 5.5

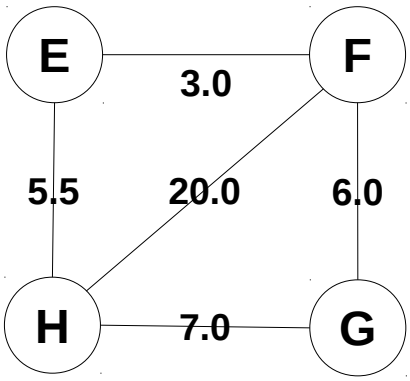
Exemplo 2:

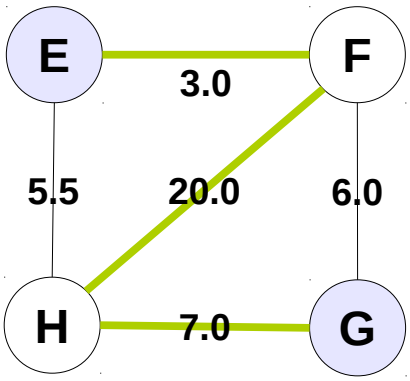


Menor caminho hamiltoniano de A para C será: **C [6.0] B [5.5] A**. Distância do caminho: 11.5

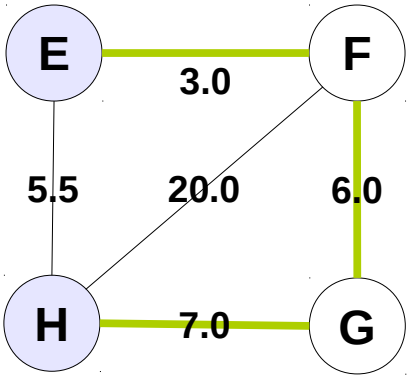
Menor caminho hamiltoniano de B para C será: **B [5.5] A [3.0] C**. Distância do caminho: 8.5

Exemplo 3:



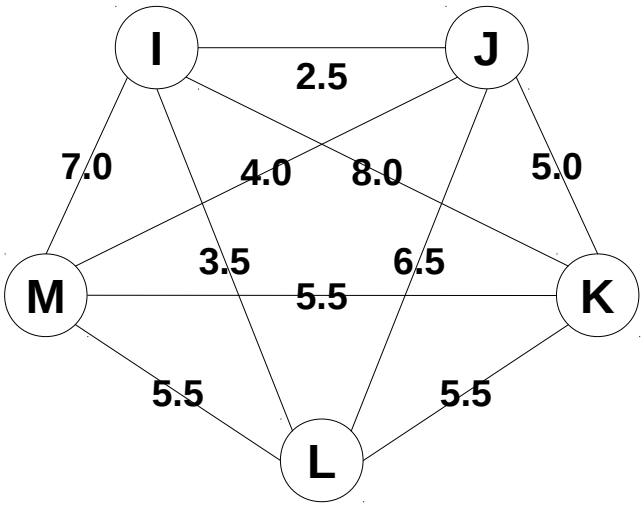


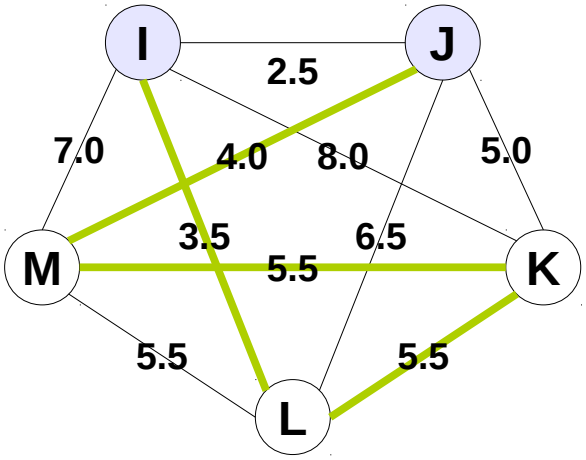
O menor caminho hamiltoniano de E para a G será:
E [3.0] F [20.0] H [7.0] G.
Distância do caminho: 30.0



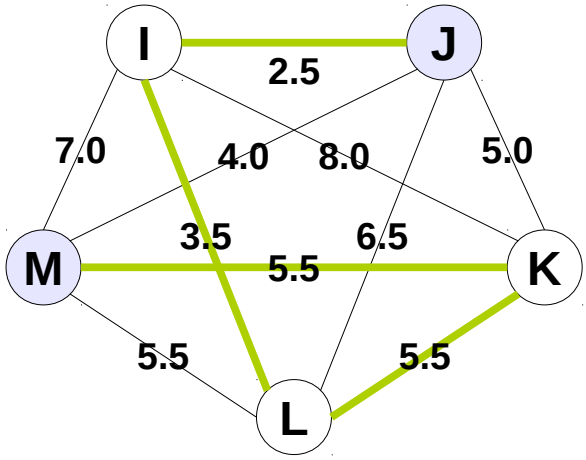
O menor caminho hamiltoniano de E para a H será:
E [3.0] F [6.0] G [7.0] H.
Distancia do caminho: 16.0

Exemplo 4:





Menor caminho hamiltoniano partindo da cidade I para a J: **I [3.5] L [5.5] K [3.0] M [4.0] J**. Distância do caminho: 16.0



Menor caminho hamiltoniano partindo da cidade J para a M: **J [2.5] I [3.5] L [5.5] K [3.0] M**. Distância do caminho: 14.5

O sistema será composto pelas seguintes classes (muitas delas já implementadas, disponibilizadas juntamente com o código do EP). A classe hachurada é aquela que vocês precisarão implementar/completar. A solução de cada aluno deve compilar e executar corretamente considerando a versão original das demais classes (isto é, as demais classes não devem ser modificadas).

Nome	Tipo	Resumo
Caminho	classe abstrata	Classe que contém o método para a solução ótima do problema do caminho hamiltoniano.
Cidade	classe	Classe que descreve uma cidade: nome, arranjo de distâncias (cidades vizinhas e suas distâncias) para a cidade atual.
Distancia	classe	Classe que contém dois atributos distância e cidade e serve para representar a distância de uma cidade vizinha.
ExecutaTestes	classe	Classe "executável" (que possui método *main*) que pode ser usada para testar as demais classes.
Mapa	classe	Classe que contém o arranjo das cidades de um mapa e alguns métodos auxiliares.

A seguir cada uma das classes é detalhada.

Classe Caminho

Classe abstrata que deve ser completada pelo aluno. Há um método que deve ser completo, além disso cada aluno poderá adicionar atributos e/ou métodos auxiliares. Cada aluno não deverá mudar a assinatura do método **`encontrarMelhorCaminho`**. O método encontrarMelhorCaminho deverá retornar um arranjo com todas as cidades do mapa e na ordem que o caminho deverá ser percorrido (começando pela cidade origem, terminando pela destino e sem repetir cidades).

```
/* Este metodo deve retornar o caminho mais curto que passe por
 * todas as cidades do mapa apenas uma vez,
 * comecando pela Cidade origem e terminando na Cidade destino.
 * O caminho deve ser retornado na forma de um arranjo de cidade.
 */
public static Cidade[] encontrarMelhorCaminho(Mapa mapa, Cidade origem, Cidade destino){
      //TODO COMPLETAR
}
```

Classe Cidade

Classe que representa uma cidade. Uma cidade é composta por 3 atributos, um construtor e dois métodos.

Atributos:

```
public String nome; // Nome da cidade
public boolean cidadeVisitada = false; // atributo auxiliar que pode ser utilizado para marcar se uma cidade foi visitada durante o backtracking
public Distancia[] vizinhas; // arranjo que contem as distancias das cidades vizinhas (cada objeto do tipo Distancia contem uma cidade e uma distancia)

/* Construtor da classe Cidade */
Cidade(String nomeDaCidade){
	nome = nomeDaCidade;
}

/* metodo para complementar o contrutor, adicionando o arranjo de cidades vizinhas */
void adicionarArranjoDeDistancias(Distancia[] vizinh){
	vizinhas = vizinh;
}

/* metodo para verificar se uma dada cidade eh vizinha da cidade atual */
boolean ehVizinha(Cidade cidade1){
	for (int i=0;i<vizinhas.length;i++){
		if (vizinhas[i].cidade == cidade1) return true;
	}
	return false;
}
```

Classe Distancia

Esta classe serve para unir dois atributos: distancia (do tipo double) e cidade (do tipo Cidade). Esta classe serve para indicar uma cidade vizinha (e sua distância), dada uma cidade qualquer.

A classe possui apenas dois atributos e um contrutor.

```
Cidade cidade; // atributo para indicar uma Cidade
double distancia; // atributo para indicar a distancia entre duas cidades
/* contrutor da classe Distancia */
Distancia(Cidade cid, Double dist){
	cidade = cid;
	distancia = dist;
}
```

Classe ExecutaTestes

Classe “executável” (possuidora do método main) e que chama três conjuntos de teste para o EP2. Possui um atributo estático para indicar qual conjunto de testes será executado: CONJUNTO_DE_TESTE.

O primeiro conjunto de dados é composto por 6 mapas diferentes (e um total de 20 testes). Para o segundo e terceiro conjuntos de dados, os mapas são gerados aleatoriamente e por isso os resultados irão variar.

Para o primeiro conjunto de dados, as saídas esperadas estão no arquivo saida1.txt. O tempo de execução é impresso apenas de maneira informativa, mas é importante tomar cuidado para não desenvolver uma solução muito ineficiente (isto é, que fique testando diversos caminhos impossíveis, por exemplo). **Durante os testes, todo teste que demorar para executar mais de 100 vezes o tempo da solução gabarito será considerado incorreto.**

Classe Mapa

Esta classe contem um mapa (isto é, um arranjo de Cidades) e alguns métodos auxiliares.

```
Cidade[] cidadesDoMapa; // arranjo com todas as cidades do mapa

/* construtor da classe Mapa */
Mapa (Cidade[] cids){
      cidadesDoMapa = cids;
}

/* metodo para retornar o numero de cidades do mapa atual */
public int numeroDeCidades(){
      return cidadesDoMapa.length;
}

/* dado um caminho, calcula a distancia total deste caminho */
static double calcularDistanciaDoCaminho(Cidade[] caminho){
 ...
}

/* retorna a distancia entre duas cidades, caso sejam vizinhas, ou -1 caso contrario
 * Dica: eh possivel (e recomendavel) implementar uma boa solucao sem utilizar este metodo
 */
static double distanciaDuasCidades(Cidade cidade1, Cidade cidade2){
      double distancia = -1;
      Distancia temp;
      for (int i=0;i<cidade1.vizinhas.length;i++){
            temp = cidade1.vizinhas[i];
            if (temp.cidade == cidade2) return temp.distancia;
      }
      System.err.println("As cidades " + cidade1 + " e " + cidade2 + " nao sao vizinhas.");
      return distancia;
}

/* metodo que imprime as cidades e distancias entre as cidades de
 * um dadao caminho
 */
static void imprimirCaminho(Cidade[] caminho){
 ...
}

/* metodo que gera mapas aleatorios, dados o numero de cidades e o numero de
 * ligacoes (estradas) entre as cidades
 */
static Mapa geradorDeMapa(int cidades, int ligacoes){
 ...
}
```

Regras de Elaboração e Submissão

- Este trabalho é individual, cada aluno deverá implementar e submeter via COL sua solução.
- A submissão será feita via um arquivo zip ou rar (o nome do arquivo deverá ser o número USP do aluno, por exemplo 1234567.zip). Este arquivo deverá conter APENAS o arquivo Caminho.java
- Além deste enunciado, você encontrará na página da disciplina um zip contendo todos os arquivos envolvidos neste trabalho (note que o arquivo Caminho.java a ser implementado também está disponível no site, você precisará apenas completá-lo).
- Qualquer tentativa de cola implicará em nota zero para todos os envolvidos.
- Guarde uma cópia do trabalho entregue e verifique, no Col, se o arquivo foi submetido corretamente.
- A data de entrega é 20 de outubro (sábado) até às 23:00h (com um hora de tolerância), não serão aceitos trabalhos entregues após esta data (não deixe para submeter na última hora).
- Se necessário, implemente na classe Caminho métodos auxiliares e crie atributos. Estes métodos e atributos deverão ser estáticos (*static*) (lembre-se de zerar/inicializar os atributos necessários em cada execução do método *encontrarMelhorCaminho*). Não crie nenhuma classe nova. Não utilize conceitos ou classes que não foram estudados em aula. Só complemente a implementação da classe solicitada (e, no máximo, implemente métodos auxiliares e/ou crie atributos na própria classe).
- **Caso o EP não compile a nota do trabalho será zero.** É importante que você teste seu trabalho executando a classe ExecutaTestes (note que ela não testa todas as funcionalidades do método implementado, por exemplo, ela não verifica se na sua solução não há cidades repetidas em um único caminho).
- Preencha o cabeçalho existente no início do arquivo Caminho.java (há espaço para se colocar turma, nome do professor, nome do aluno e número USP do aluno).
- Todas as classes pertencem ao pacote ep2_2012
- Todos os mapas terão no mínimo duas cidades e sempre haverá ao menos um caminho entre as cidades utilizadas como origem e destino nos testes.
- Todas as distâncias entre as cidades serão positivas.