**CES12 LAB TSP - 2019**

ITA - IEC

**Objetivo:** Implementar algoritmo aproximativo para o *Problema do Caixeiro Viajante.*

**Descrição**

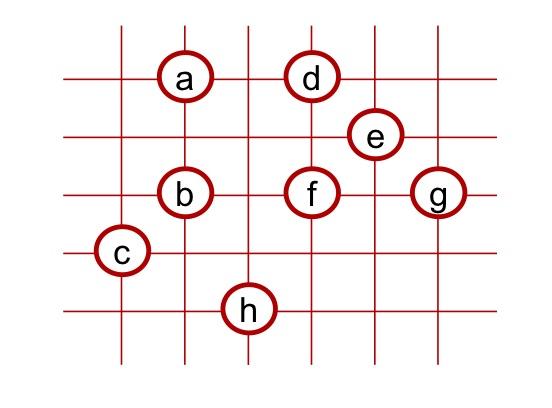
O *Problema do Caixeiro Viajante* (*Travelling Salesman Problem*) é um conhecido problema de combinatória, que pode ser formulado da seguinte maneira: dadas **n** cidades e as distâncias entre elas, qual é o menor ciclo possível que passe por todas? Este problema tem grande importância na Teoria de Computação: é NP-Difícil, ou seja, não se conhece nenhuma resolução de tempo polinomial em **n**.

No entanto, há um conhecido algoritmo aproximativo de tempo polinomial em **n** para este problema, que encontra uma solução cujo valor é menor que o dobro da solução ótima. Para que este algoritmo possa ser aplicado, algumas condições são necessárias:

a) O grafo referente às cidades deve ser completo, ou seja, sempre há um caminho entre qualquer par de cidades.

b) As distâncias entre os vértices são euclideanas.

Vamos descrever este algoritmo através de um exemplo. Considere o mapa abaixo, com **n** = 8, e o seu arquivo de entrada:

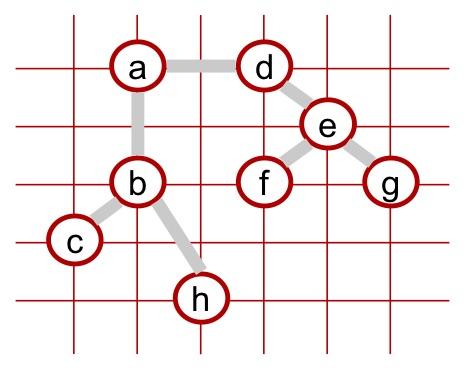






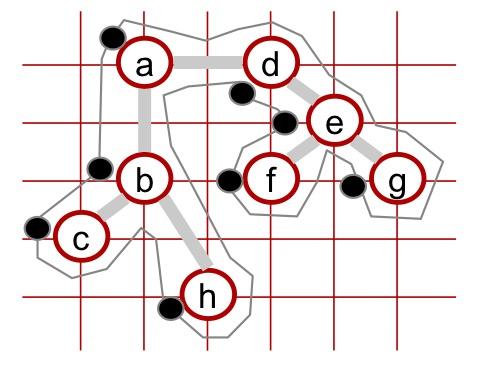
Repare que os vértices serão sempre representados por números de 1 a **n**. Lembre-se também de que, a partir das coordenadas de cada par de vértices, será preciso calcular a distância entre eles. Para evitar divergências no arredondamento, a distância deverá ser um valor inteiro, arredondado para o inteiro mais pŕoximo (função round).

Em seguida, você deverá encontrar a *árvore geradora de custo mínimo* deste grafo, apresentado no Capítulo 9 do nosso curso. No exemplo anterior, veja a árvore correspondente, que chamaremos de T:



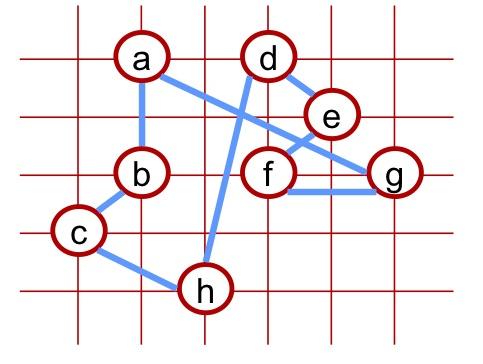


Uma possível solução para o *Problema do Caixeiro Viajante* pode ser obtida através de um ciclo C' ao redor de T, onde cada aresta dessa árvore é percorrida duas vezes. Veja na figura abaixo como seria C' calculado a partir da cidade a, (os pontos negros indicam a primeira vez que cada vértice é visitado):



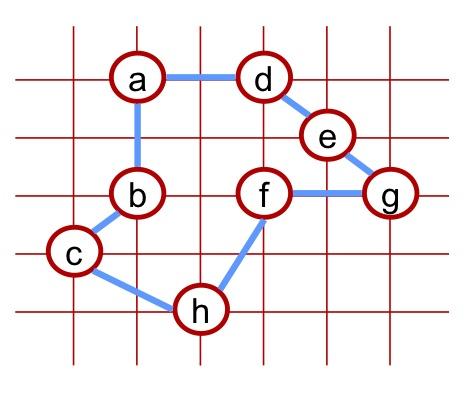


Repare que o ciclo C' pode ser encontrado a partir de um percurso pré-ordem em T. Por outro lado, esta solução pode ser melhorada evitando-se arestas que incidam em vértices já visitados, ou seja, incluem-se apenas arestas para o próximo vértice ainda não visitado. Veja abaixo como ficaria o novo ciclo calculado, chamado de C, que pode ser construído a partir de C' e dos pontos negros:





Este ciclo C é a nossa solução aproximada. Como foi dito, não há nenhuma garantia de que seja ótima. Para este exemplo, o ciclo de custo mínimo C\* está indicado na figura abaixo:





No entanto, é possível demonstrar uma importante propriedade da solução C. Lembrando:

* T: árvore de espalhamento de custo mínimo
* C': ciclo ao redor de T, com repetição de arestas
* C: ciclo baseado em C', sem repetição de arestas
* C\*: ciclo de custo mínimo

Seja c(G) o custo associado a um grafo G. Se removermos uma aresta qualquer do ciclo mínimo C\*, obteremos uma árvore de espalhamento. Portanto, c(T) < c(C\*).

Em C', cada aresta de T ocorre exatamente 2 vezes. Logo, c(C) ≤ c(C') = 2.c(T), ou seja, c(C) < 2.c(C\*). Em outras palavras, a solução C não é necessariamente ótima, mas seu custo é sempre menor que o dobro da solução ótima.

**Entrada**

Já foi fornecida a classe TspReader para ler arquivos TSP como os disponíveis em

<http://www.math.uwaterloo.ca/tsp/world/countries.html>

<http://www.math.uwaterloo.ca/tsp/world/summary.html>

A leitura é simplificada:

* apenas linhas cujo 1o char é inteiro são consideradas, portanto o cabeçalho descritivo não é lido.
* cada linha especifica um vértice
  + formato <int> <float> <float>
  + significado <vextex id> <x> <y>
  + exemplo de linha: 2 43.34513 34.123411
* além disso, ignoramos o campo vertex id e numeramos os vértices na ordem lida do arquivo. Isto não deve afetar o resultado, apenas evita erros decorrentes de numeração errada no arquivo.

São fornecidos testes com várias instâncias representando cidades em países e pontos de solda em circuitos integrados, de aproximadamente 20 a 1000 vértices por instância.

O aluno deverá implementar a classe TspSolver, que é fornecida com apenas um método, que recebe um TspReader já preenchido com valores; e deve preencher um vetor de inteiros com a resposta do Caixeiro Viajante.

a resposta do caixeiro viajante é uma permutação dos inteiros de 1 a n que representa a ordem em que os vértices são visitados.

por exemplo, na figura C, a ordem de visitação dos vértices é

abchfged

como os índices são inteiros de 1 a n, assumindo que a numeração segue a ordem alfabética, este percurso seria representado por:

12386754

que é uma permutação dos inteiros de 1 a 8.

A classe TspReader não cria ou representa um grafo, ela apenas fornece os dados na forma de triplas {<id>, <x>, <y>}. Seu programa deverá criar representação de um grafo não orientado, utilizando listas ou matrizes de adjacências, como visto em sala de aula.

*Importante*:

* Ao escolher a representação do grafo, considere que deve implementar primeiro o algoritmo para a árvore geradora mínima, e depois, ajustar o ciclo conforme o exemplo, de C’ para C.
* Nos grafos não orientados, lembre-se de que cada aresta deve estar presente na lista de adjacências de ambos os vértices incidentes.
* Cuidado para não "estourar" a memória ao longo da bateria de testes. Para isso, utilize uma forma adequada de alocação, cada vez que for necessário criar um novo grafo.
* Os testes apenas checam se:
  + a resposta é uma permutação dos inteiros de 1 a n
  + custo do ciclo fornecido < 2 \* custo do ciclo ótimo.

**Importante**

* Não é necessário verificar a consistência dos dados de entrada: você pode supor que cada arquivo de entrada seguirá perfeitamente a estrutura indicada acima.
* O processo de correção consistirá na submissão automática do seu programa a essa bateria de testes. Por isso, a formatação de entrada e de saída deve ser obedecida rigorosamente.
* Para que todos os alunos encontrem a mesma solução em cada teste, serão estabelecidas as seguintes regras:
  + O vértice 1 será sempre a raiz da árvore T.
  + T deverá ser encontrada através do algoritmo de *Prim*. Em cada passo deste algoritmo, o novo vértice a ser incluído será o mais próximo de T, dentre aqueles que ainda não pertencem a T. Em caso de empate, será escolhido o que for adjacente ao vértice de menor índice em T. Em caso de novo empate, será escolhido o vértice de menor índice.
  + Na sequência de visitas em T que gera o circuito C', sempre será dada prioridade ao vizinho mais próximo. Se houver vizinhos com a mesma distância, o desempate será através do menor índice.
* O aluno deve preencher a planilha resultados para garantir que todos estao seguindo as mesmas convenções - e serve como um teste mais preciso.
  + CUIDADO PARA NÃO MEXER INDEVIDAMENTE NA PLANILHA
* A princípio, nota máxima apenas se os resultados baterem com a turma

## FAQ:

### Precisa implementar heap em Prim ou pode fazer com outro tipo de implementação?

Precisa ser Prim ou o resultado não será o mesmo. E não precisa ser heap per se, mas se demorar muito mais do que o resto da turma pode perder nota, especialmente se a diferença for muito grande e atrapalhar a correção.

### E haverá nos grafos vértices distintos na mesma posição? (Ou seja, distância da aresta igual a 0)

Os pontos nos datasets são cidades em um país ou pontos em um circuito impresso. Podem haver pontos bem próximos, mas não deveria haver pontos na mesma posição. Tem? Não conferi.

### No lab 5 é necessário enviar algum relatório ou apenas o código implementado?

Se quiser explicar alguma coisa ou se a sua implementação for fora do comum pode incluir um pdf.

Mas a princípio não precisa

### Pq o dele é o certo e não o meu?

Se dois alunos terminaram, os resultados estão < 2\*otimo, mas não conseguem entender porque está diferente, podem trocar informações e/ou verificar o codigo do outro para ajudar a encontrar o que está fora da convenção. Depois de encontrar o problema, arrumar o código volta a ser individual.

O algoritmo é determinístico, se todos seguirem a mesma convenção deveria ser igual, a não ser que a especificação da convenção for insuficiente ou houver alguma ambiguidade no dataset.