CES12 LAB TSP - 2023

ITA - IEC (profs. Luiz Mirisola e Carlos Alonso)

Objetivo: Implementar algoritmo aproximativo para o *Problema do Caixeiro Viajante, obedecendo estritamente convenções de ordem para alcançar resultados determinísticos exatos.*

Descrição

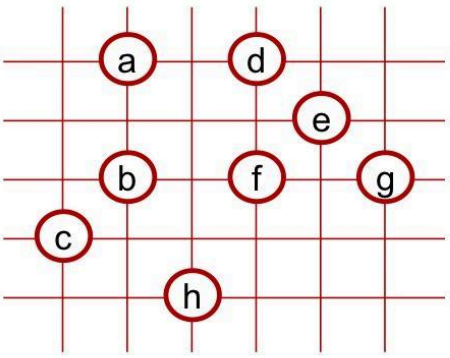
O *Problema do Caixeiro Viajante* (*Travelling Salesman Problem*) é um conhecido problema de combinatória, que pode ser formulado da seguinte maneira: dadas **n** cidades e as distâncias entre elas, qual é o menor ciclo possível que passe por todas? Este problema é muito conhecido na Teoria de Computação: é NP-Completo, ou seja, muito provavelmente[1] , não existe solução 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 devem ser euclidianas.

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

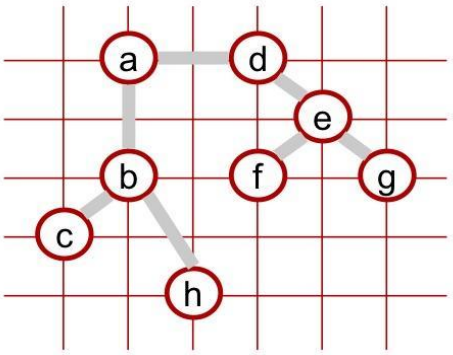



```
1 2 5
2 2 3
3 1 2
4 4 5
5 5 4
6 4 3
7 6 3
8 3 1
```

Considere que o valor **n** é sempre maior ou igual a 1

Repare que os vértices são 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 próximo (função `round`).

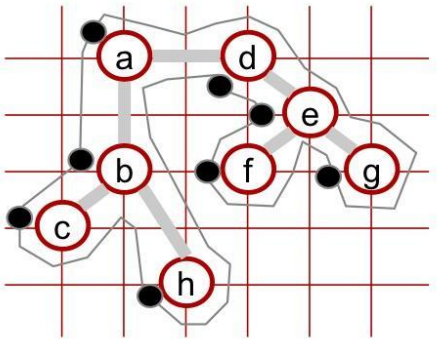
[1] Não existe exceto se P = NP



T

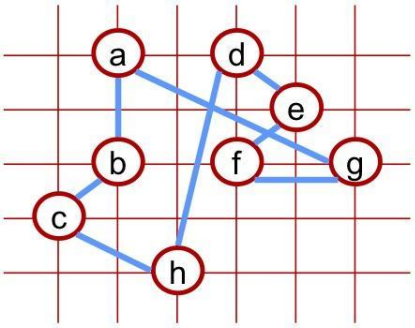
Em seguida, você deverá encontrar a *árvore geradora de custo mínimo* deste grafo, apresentada no Capítulo 9 do nosso curso. No exemplo anterior, veja a árvore correspondente, que chamaremos de T:

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):



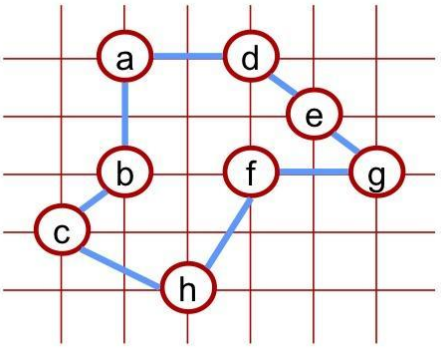
C'

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:



C

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:



C*

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, $c(C) \leq c(C') = 2.c(T)$, ou seja, $c(C) < c(C')$. 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
https://www.math.uwaterloo.ca/tsp/vlsi/index.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 <vertex 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, e desaloque memória dinamicamente alocada ao término da execução.
- Os testes Tsp checam se:
    - a resposta é uma permutação dos inteiros de 1 a n
    - custo do ciclo fornecido < 2 * custo do ciclo ótimo.
- Os testes EXACT checam se:
    - a resposta é uma permutação dos inteiros de 1 a n
    - o custo do ciclo fornecido é exatamente igual ao valor obtido ao se seguir exatamente as mesmas convenções de ordem na execução do algoritmo.

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 e passem nos testes EXACT, 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.

Nota:

considerando a proporção dos testes passando em cada categoria:

- 8.5 pontos para os testes Tsp
- 1.5 pontos para os testes Exact

- Penalidade de 2 pontos se não usar heap ou solução equivalente.
- Penalidade de 5 pontos se demorar mais de 10 segundos para passar nos testes.
- Nota ZERO se demorar mais de 15 segundos para passar nos testes.
- O recorde até agora é <100ms, então tem 2 ordens de grandeza de tolerância.

FAQ:

Precisa implementar heap em Prim ou pode ser outra implementação? E outro algoritmo (Kruskal)?

Precisa ser Prim ou o resultado não será o mesmo - passarão apenas os testes Tsp, não os EXACT. Não precisa usar heap *per se*, pode haver outra solução equivalente, mas é preciso estar na mesma ordem de grandeza de tempo de execução. E pode usar a classe heap da stdlib (ou seja, **não há motivos para preguiça**). Por sinal, é um aprendizado interessante usar uma classe da STL um pouco mais complexa do que vector e list.

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. Mas o arquivo de Luxemburgo, pelo menos, tem.

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

A princípio não precisa. Mas se quiser explicar alguma coisa ou se a sua implementação for fora do comum (Prim com heap) pode incluir um pdf.

Pq o dele é o certo e não o meu?

Se dois alunos terminaram, os resultados estão < 2*ótimo, passam pelos testes TSP, mas não conseguem entender porque as implementações são diferentes e pelo menos um deles não passa por todos os testes EXACT, podem trocar informações e/ou verificar o código um do outro para ajudar a encontrar o que está fora da convenção. Depois de encontrar o problema, corrigir o código volta a ser individual.
O algoritmo é determinístico, se todos seguirem a mesma convenção o resultado é igual. A não ser que a especificação da convenção for insuficiente ou houver alguma ambiguidade no dataset.

Pq precisa ser exato?

É um bom exercício de disciplina (tanto no sentido de 'curso' quanto no sentido de virtude) ser obrigado a seguir uma convenção. Precisa entender o algoritmo e ser estrito - não basta copiar do livro-texto - precisa pensar um pouco mais.
Além disso, deixará claro para alguns a diferença entre algoritmos aleatórios, que precisam, por exemplo, gerar números aleatórios para tomar decisões, e geram resultados diferentes a cada execução ; versus algoritmos determinísticos, mas com várias respostas possíveis, onde diferenças no resultado se devem a escolhas de detalhes na implementação.

VERSÃO 220609

CMakefile.txt incorpora novas receitas p/ novas versões.

VERSÃO 220201

CORRIGIDO Bug report:

O Gabriel Gobi encontrou erro na leitura dos arquivos em TspReader.h:

```
City(int x_, float y_, float id_) : x(x_),y(y_),id(id_) { };
```

deveria ser

```
City(float x_,float y_,int id_) : x(x_),y(y_),id(id_) { };
```

Isso causa erro de arredondamento na coordenada x pois o casting trunca ao invés de arredondar como o round(). Não afeta os datasets de placas eletrônicas pois as coordenadas são inteiras, mas causa diferença nos caminhos dos países.

Nota: fiz update do testTsp.cpp com novos valores (passados pelo Gobi) e testei com o Gobi e mais 2 alunos. Todos passaram nos testes exact. Ainda mantive os valores anteriores comentados no código. (o erro de arredondamento fez diferença em apenas 5 países)

VERSÃO 200717_1428

Os valores para os testes exact deveriam ser a mediana dos resultados da turma anterior. Mas... copiei os valores da linha 'max' da planilha por engano. Assim, 2 alunos encontraram valores sempre abaixo do esperado nos testes. A nova versão tem os valores corretos, e já foi testada com sucesso.

Mudança: arquivo test/testTsp.cpp
Valores em todos os testes EXACT, exceto o primeiro.