**UNIVERSIDADE FEDERAL DE PELOTAS**
**CENTRO DE DESENVOLVIMENTO TECNOLÓGICO**
**CURSOS DE CIÊNCIA E ENGENHARIA DE COMPUTAÇÃO**
**Disciplina de Programação Orientada a Objetos - 2015/1**
**Prof.: Felipe de Souza Marques (felipem@inf.ufpel.edu.br)**

# Especificação do Trabalho que Substitui a Segunda Prova:

## *Traveling Salesperson Problem*

(A versão original está em: http://www.cis.upenn.edu/~cis110/13sp/hw/hw08/tsp.shtml - A versão traduzida omite a parte que contextualiza o problema)

### 1. Preparando o Ambiente

- Baixe o arquivo, disponível em: http://ava.ufpel.edu.br/pre/mod/resource/view.php?id=35765, e o descompacte no local onde você pretende construir o seu projeto. Este arquivo contém Point.java, Node.java, programas clientes para cada uma das heurísticas, o template readme_tsp.txt e um conjunto de arquivos de entrada para testes.
- Estude a API da classe Point.java. Seu método principal lê dados pela entrada padrão em um formato de entrada específico (TSP) e desenha os pontos em uma área denominda *standard draw* (não é necessário entender como *StdDraw* está implementado – nem os detalhes de implementação da classe *Point*).
- Point.java representa um ponto em um plano, como descrito pela seguinte API:

```
public class Point (2D point data type) ----------------------------
    Point(double x, double y)  // cria o ponto (x, y)
    String toString()          // retorna um representação em uma String
    void draw()                // desenha um ponto usando standart draw
    void drawTo(Point that)    // desenha um segmento de linha entre dois
pontos
    double distanceTo(Point that)  // retorna a distância Euclidiana entre
dois pontos
```

Cada objeto Point pode retornar uma representação string dele mesmo, pode desenhar a si mesmo, desenhar uma linha partindo de si mesmo para outro ponto e calcular a distância Euclidiana entre dois pontos.

- Estude a classe Node.java que representa um dos nodos do TSP e contém um ponto e uma referência para o próximo nodo da rota do TSP.

```
public class Node {
    public Point p;
    public Node next;
}
```

## 2. Parte I: Definição da Classe *Tour* (5 pontos)

Sua tarefa é criar uma classe *Tour* que representa a sequencia de pontos visitados em uma travessia do TSP. Ela representa uma travessia por uma lista ligada circular de nodos. Sua classe DEVE implementar a seguinte API:

```
public class Tour
---------------------------------------------------------------------------
    Tour()
    Tour(Point a, Point b, Point c, Point d)
    void show()
    void draw()
    int size()
    double distance()
    void insertNearest(Point p)
    void insertSmallest(Point p)
```

- Cada objeto *Tour*deve ser capaz de imprimir seus pontos na saída padrão, de desenhar seus pontos, contar o seu número de pontos, calcular a distância total e inserir um novo ponto usando qualquer uma das duas heurísticas. Primeiro, o construtor cria uma rota vazia (sem pontos); em seguida, o construtor cria 4-pontos definindo uma rota para fins de debug.
- AVISO: Você perderá um número substancial de pontos se seus métodos tiverem comportamentos ou assinaturas do que foi definido acima. Você não pode adicionar métodos públicos na API; entretanto, você pode adicionar métodos privados se julgar necessário. O objetivo do trabalho é utilizar estruturas ligadas (encadeadas). Logo, você não pode utilizar as classes Java que implementam estruturas de dados (LinkedList por exemplo).
- Crie um arquivo Tour.java. Inclua um atributo na classe, chamado de *first*, do tipo Nodo que deve referenciar o primeiro nodo da lista circular.
- Para fins de debug, faça um construtor que recebe quatro pontos como argumento e constrói uma lista circular usando quatro objetos do tipo *Point*. Primeiro, crie os quatro pontos e faça o assinalamento de um para o outro, criando uma estrutura circular.
- Implemente o método show(). Ele deve passar por cada nodo (Node) da lista circular, iniciando pelo primeiro nodo e imprimindo cada ponto (Point) usando StdOut.println(). Este método precisa de poucas linhas de código, mas é importante pensar com cuidado, pois o debug do código de listas encadeadas pode ser difícil e frustrante. Inicie pela impressão do primeiro ponto. Em listas circulares, o último ponto é o próprio ponto inicial (depois de percorrer toda a lista). Sendo assim, cuidado com os loops infinitos! Se a rota tem 0 pontos, você não deve escrever nada na saída.
- Teste seu método escrevendo uma função main() que define quatro pontos, cria uma nova rota entre os quatro pontos e chama o método show(). Abaixo está a sugestão da função main.

```
// main method for testing
  public static void main(String[] args) {
    // define 4 points forming a square
    Point a = new Point(100.0, 100.0);
    Point b = new Point(500.0, 100.0);
    Point c = new Point(500.0, 500.0);
    Point d = new Point(100.0, 500.0);

    // Set up a Tour with those four points
    // The constructor should link a->b->c->d->a
```

```
    Tour squareTour = new Tour(a, b, c, d);

    // Output the Tour
    squareTour.show();
  }
```

Se seu método estiver funcionando adequadamente, você verá a seguinte saída:

```
(100.0, 100.0)
(500.0, 100.0)
(500.0, 500.0)
(100.0, 500.0)
```

Teste o método show() em rotas com 0, 1 e 2 pontos para verificar se ele funciona. Você pode criar instâncias modificando o construtor para debug que recebe quarto pontos, ligando apenas 0, 1 ou 2 dos quarto pontos da rota (Tour).

Depois de realizar os testes acima, garanta que o método construtor que recebe os 4 pontos continue funcionando como a especificação original.

- Implemente o método size(). Ele é muito similar ao show(). Se a rota tem 0 pontos, você deve retornar 0.
- Implemente o método distance(). Ele é muito similar ao show(), exceto que você precisa invocar o método distanceTo() da classe Point. Adicione o método distance() no main() e imprima a distância e o tamanho. No exemplo dos 4 pontos, a distância é 1600.0. Se a rota tem 0 pontos, distance() deve retornar 0.0.
- Implemente o método draw(). Ele também é muito similar ao show(), exceto que você precisará invocar o método drawTo() da classe Point. Se a rota tem 0 pontos, você não deve desenhar nada. Você precisará incluir os seguintes comandos

```
StdDraw.setXscale(0, 600);
StdDraw.setYscale(0, 600);
```

no main(), antes da chamada do método draw() da classe Tour. No exemplo dos quarto pontos, a figura desenhada deve ser um quadrado.

## 3. Parte II: Criando uma rota (5 pontos)

- Para este passo crucial, você deve escrever um método que insere um ponto p em uma rota. Como aquecimento, implemente um método simples chamado insertInOrder() para inserir um ponto em uma rota depois do ponto que foi adicionado por último. (Não há problema em deixar este método na classe Tour, mesmo que ele não faça parte da API especificada.) Para fazer este método, escreva um laço que encontra o último ponto e insere um novo ponto em um novo nodo (Node) após o último. Não h;a problema que você trate como caso especial o primeiro ponto que você inserir na lista, mas você não precisa criar nenhum outro caso especial. Para testar o método, utilize o programa OrderInsertion.java e um dos arquivos de teste pequenos. A ordem dos pontos de saída deve ser a mesma ordem dos pontos do arquivo de entrada.

- Implemente o método insertNearest(). Para determinar qual nodo que antecede o ponto p que será inserido, a distância Euclidiana entre cada ponto da rota e o ponto p deve ser calculado. Conforme você progride, armazene o nodo mais próximo e sua distância para p. Depois de encontrar o nodo mais próximo, crie um nodo contendo p e o insira depois do nodo mais próximo. Isso necessita mudança no campo *next* de ambos nodos: o novo e o mais próximo. Para verificar, o arquivo tsp10-nearest.ans contém o resultado da rota para uma problema de 10 cidades, cuja distância é 1566.1363. Note que a distância ótima seria 1552.9612. Logo, esta heurística nem sempre leva para a melhor (menor) rota. Olhe abaixo para mais instruções de como testar insertNearest().
- Depois de fazer a heurística da inserção mais próxima, você deve estar apto a escrever o método insertSmallest() por você mesmo, sem mais dicas. A única diferença é que você quer inserir o ponto p onde a inserção irá resultar no menor aumento do comprimento total. Para verificar o método, tsp10-smallest.ans contém o resultado da rota, com distância de 1655.7462. Neste caso, a heurística da menor inserção na verdade é pior que a heurística da inserção mais próxima (embora este seja um caso atípico).
- **Entrada e teste**: o formato de entrada inicia com dois inteiros w e h, seguidos de pares de coordenadas x e y. Todas as coordenadas x são números reais entre 0 e w; todas as coordenadas y são números reais entre 0 e h. Vários arquivos de teste estão disponíveis. Por exemplo, tsp1000.txt contém os seguintes dados:

```
% more tsp1000.txt
775 768
185.0411 457.8824
247.5023 299.4322
701.3532 369.7156
563.2718 442.3282
144.5569 576.4812
535.9311 478.4692
383.8523 458.4757
329.9402 740.9576
...
254.9820 302.2548
```

- Depois de implementar Tour.java, utilize o programa NearestInsertion.java para ler pontos da entrada padrão, rodar a heurística de inserção no vizinho mais próximo; imprimir o resultado da rota, sua distância e o número de pontos na saída padrão; e desenhar o resultado da rota na saída de desenho padrão. SmallestInsertion.java é análogo mas roda a heurística da menor inserção.

```
% java NearestInsertion < tsp1000.txt
(185.0411, 457.8824)
(198.3921, 464.6812)
(195.8296, 456.6559)
(216.8989, 455.126)
(213.3513, 468.0186)
(241.4387, 467.413)
(259.0682, 473.7961)
(221.5852, 442.8863)
...
(264.57, 410.328)
Tour distance = 27868.7106
Number of points = 1000
```

```
% java SmallestInsertion < tsp1000.txt
(185.0411, 457.8824)
(195.8296, 456.6559)
(193.0671, 450.2405)
(200.7237, 426.3461)
(200.5698, 422.6481)
(217.4682, 434.3839)
(223.1549, 439.8027)
(221.5852, 442.8863)
...
(186.8032, 449.9557)
Tour distance = 17265.6282
Number of points = 1000
```

- Se você quer ver a heurística em ação, redesenhe a rota depois de cada inserção. Veja as instruções em SmallestInsertion.java. Ela pode demorar um pouco para arquivos com um grande número de pontos. Então, você poderia modificar o método para redesenhar somente a cada 20 inserções ou mais.
- **AVISO**: remova ou comente qualquer print para debug antes de testar grandes arquivos ou o programa ficará rodando para "sempre". Grandes entradas irão produzir muitos dados de debug que você provavelmente não será capaz de interpretar.
- Se você quiser verificar seu trabalho, para o arquivo usa13509.txt, os resultados de distância são 77449.9794 e 45074.7769 para inserção no mais próximo e para a menor inserção, respectivamente. Para o arquivo circuit1290.txt, tem-se 25029.7905 e 14596.0971.