Projeto e Análise de Algoritmos
Trabalho 1

1 Descrição

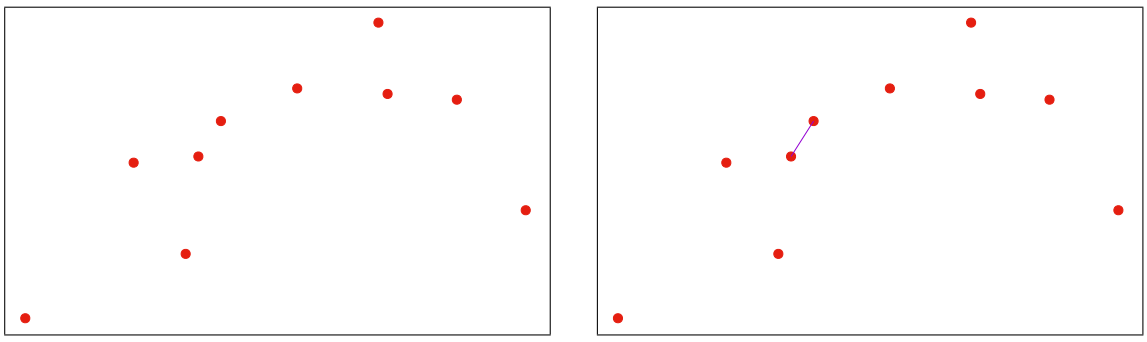
Este trabalho considera a implementação de um algoritmo de divisão e e conquista para o *Problema do Para mais Próximo*.

Cada trabalho submetido deverá ser desenvolvido por no máximo três estudantes. Os códigos devem conter cabeçalhos discriminando cada um dos estudantes que desenvolveu o trabalho submetido, com nome completo e registo acadêmico.

A avaliação do trabalho irá considerar os códigos, a execução e eficiência da implementação. Todo o conteúdo submetido pelos estudantes deve ser de autoria deles.

2 Problema do Par mais Próximo

Dentre n pontos $p_1, p_2, \ldots, p_n$ no plano, encontrar pontos p_i e p_j, $i \neq j$, tal que a distância entre p_i e p_j é a menor dentre todas as distâncias entre os pares de pontos dados. Se mais de um par de pontos possui a distância mínima, então um destes pares é escolhido arbitrariamente como resposta. Um ponto é dado por duas coordenadas $p_i = (x_i, y_i)$. A distância entre dois pontos é definida como a distância Euclidiana: $d(p_i, p_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. As Figuras 1a e 1b ilustram, respectivamente, um conjunto de pontos no plano e a distância entre o par de pontos mais próximo.



(a) (b)

Figura 1: Ilustração de alguns pontos no plano (a) e dos pontos mais próximos (b).

3 Algoritmos para o Problema

O problema pode se resolvido por um algoritmo força bruta que consiste em computar a distância entre todos os pares de pontos. Claramente este algoritmo possui complexidade de tempo $O(n^2)$.

O problema pode ser resolvido também por divisão e conquista, muito embora seja possível formular algoritmos de divisão e conquista também com complexidade $O(n^2)$ para o problema, tais soluções não serão consideradas neste trabalho. Há um algoritmo

conhecido de divisão e conquista que é assintoticamente mais rápido do que o algoritmo força bruta, tendo complexidade $O(n \log n)$. No desenvolvimento deste trabalho você deve pesquisar e considerar um algoritmo assintoticamente mais rápido do que $O(n^2)$.

4 Implementação e Execução

As implementações devem ser feitas em linguagem C ou C++ e compilar, respectivamente, usando os compiladores gcc e g++, em sistema operacional Linux.

Um arquivo Makefile deve ser disponibilizado para compilar e gerar o arquivo executável. A compilação deve ser feita usando o comando *make* em um terminal de comandos do Linux, conforme o exemplo abaixo. O símbolo $ representa o prompt de comando do terminal.

```
$ make
```

O programa gerado pelo comando *make* deve ter o nome "closest" e receber como argumento um arquivo TXT com os pontos de entrada para execução (o formato do arquivo será descrito na sequência). A execução deve ser iniciada em um terminal de comandos do Linux, como no exemplo abaixo.

```
$ ./closest input.txt
```

O programa "closest" deve computar o par mais próximo e, assim que terminar a execução dos métodos, o programa deve apresentar a saída (descrita na sequência), na saída padrão do terminal Linux, e finalizar a execução.

4.1 Arquivo de Entrada

O arquivo de entrada deve ser formatado da seguinte maneira:

- a primeira linha contém um inteiro n com o número de pontos no arquivo a serem lidos como entrada para o programa;
- as n linhas que seguem contêm as coordenadas dos pontos em valores reais. Cada linha representa um ponto e contém os valores de x e depois de y, separados em duas colunas por um espaço.

Um exemplo de arquivo de entrada com 10 pontos é mostrado abaixo.

```
10
3091.627826 601.006476
3248.708916 4082.621268
3845.638928 184.651283
7545.393290 1154.317298
7748.718331 9864.560519
2608.641732 838.303203
5728.058273 6604.539820
702.610314 8500.248091
8794.220145 6218.940344
405.380470 2500.490589
```

No Moodle da disciplina foi disponibilizado um código para gerar um arquivo de pontos aleatórios neste formado. O código tem nome "genpoints.c". Para compilar basta executar o Makefile que acompanha o arquivo do código usando o comando abaixo.

```
$ make
```

O programa recebe como argumento o número de pontos desejados, conforme o exemplo abaixo,

```
$ ./genpoints 1000000
```

e gera um arquivo chamado "input.txt" no formato da entrada.

4.2 Formatação da Saída

A saída do programa "closest" deve ser formada em uma única linha da seguinte maneira:
tempo *distancia* x_1 y_1 x_2 y_2
tal que

- *tempo* é o tempo de execução em segundos;
- *distancia* é a distância entre os pontos mais próximos encontrados;
- x_1, y_1, x_2 e y_2 são as coordenadas dos pontos mais próximos encontrados;

Os valores devem ser separados por um espaço e com uma quebra de linha após o último valor. Todos os valores devem ser representados em ponto flutuante com seis casas decimais. Ao executar o programa, um exemplo de saída esperada seria o seguinte.

```
$ ./closest input.txt
7.000148 87.843194 9295.508661 2453.464895 9208.706133 2439.983507
$
```

5 Entrega

As entregas deverão ser realizadas usando o link disponibilizado no Moodle e não serão aceitas entregas fora do prazo.

Os autores do trabalho devem reunir os códigos-fonte em um diretório nomeado com os números dos seus registros acadêmicos (RAs) separados por '_'. Por exemplo, se dois estudantes desenvolveram o trabalho e têm RAs 123456 e 789012, então o diretório deve ter o nome 123456_789012.

Dentro do diretório deverão estar os códigos-fonte e o arquivo Makefile. Por exemplo, o diretório poderia conter os arquivos listados abaixo (não coloque acentos ou espaços em nomes de arquivos).

```
123456_789012/hull.c
123456_789012/Makefile
```

O arquivo de entrega deve ser um ZIP do diretório. O arquivo ZIP deve inclusive conter o diretório e ser nomeado da mesma maneira que o diretório. No nosso exemplo: 123456_789012.zip.