Trabalho 2 de AA

Divisão e Conquista

Autores: 1320614 - Alexandre Werneck
1311162 - Hugo Roque

ÍNDICE

1 - Ambiente experimental

Sistema Operacional	Linux 64 bits
Memória	4GB
Linguagem de programação	Python
Versão da linguagem	2.7.6

2 – Algoritmos e estrutura de dados utilizados

Domínio (domain.py)

Point: Classe auxiliar que representa um ponto em um plano 2D. Os objetos instanciados a partir dela tem os atributos x e y, tornando o código mais expressivo.

```
1. point = Point(1.3, 5.0)
2. print "%.3f, %3f" % (point.x, point.y)
3. # => 1.300, 5.00
```

Exemplo de uso da classe Point

Pair: Classe que representa um par de pontos e encapsula a lógica que calcula a distância entre dois pontos. Essa classe implementa as funções _gt_ e _eq_, possibilitando comparar dois pontos baseado em suas distâncias.

```
1. pt1 = Point(1,1)
2. pt2 = Point(2,2)
3. pt3 = Point(3,3)
4.
5. pair1 = Pair(pt1, pt2)
6. pair2 = Pair(pt1, pt3)
7.
8. print(pair1.distance()) # => 1.41421356237
9. print(pair2.distance()) # => 2.82842712475
10. print(pair1 < pair2) # => True
```

Exemplo de comparação entre pares

Tais comparações possibilitam o uso das funções min() e max() para selecionar o par de pontos de menor ou maior distância.

```
pt1 = Point(1,1)
pt2 = Point(2,2)
pt3 = Point(3,3)
pt4 = Point(4,4)

pair1 = Pair(pt1, pt2)
pair2 = Pair(pt1, pt3)
pair3 = Pair(pt1, pt4)

print(min(pair1, pair2, pair3)) # => pair1
print(max(pair1, pair2, pair3)) # => pair3
```

Min e max de pares de pontos

Força bruta (brute_force.py)

Calcula as distâncias entre todos os pares de pontos e retorna o par de pontos que tem a menor distância. Também monitora o tempo de execução afim de não ultrapassar o limite de 3 minutos (180 segundos), caso ocorra ele retorna *None*.

Para isso ele utiliza a classe *Pair* (descrita acima) para comparar dois pares de ponto pelas suas distâncias e a biblioteca *time* que permite o acesso ao relógio do sistema operacional.

A complexidade assintótica de tempo é de $O(n^2)$, sendo n o número de pontos.

Divisão e conquista (divide_and_conquer.py)

O algoritmo utiliza a técnica de divisão e conquista que, neste caso, consiste em dois passos.

Primeiro dividimos o plano em dois sub-planos e aplicamos o algoritmo recursivamente em cada um deles de maneira a encontrar o par de menor distância que se encontra em um dos sub-planos.

O segundo passo consiste em verificar se o par de menor distância pertence aos dois sub-planos mutuamente, usando a distância que obtivemos no primeiro passo para reduzir a possibilidade de pontos.

A solução é consiste no par de menor distância dentre os dois pares obtidos nos passos 1 e 2.

Na implementação usamos as estruturas Array, Pair e Point.

A complexidade assintótica de tempo é de $O(n \log n)$, sendo n o número de pontos.

Ordenação (order_plan.py)

Ordena um conjunto de pontos em função dos eixos x ou y *(order_by_x / order_by_y)* usando o algoritmo *Merge sort (mergesort)*.

A complexidade assintótica de tempo para ordenar é de $O(n \log n)$, sendo n o número de pontos.

Instâncias (instances.py)

Tem como função primária a geração das instâncias de planos. Porém, ao invés de gerar todas as instâncias em um único momento, ele permite que módulos clientes iterem por cada grupo de 10 instâncias e gera essas instâncias no momento em que o grupo é requisitado.

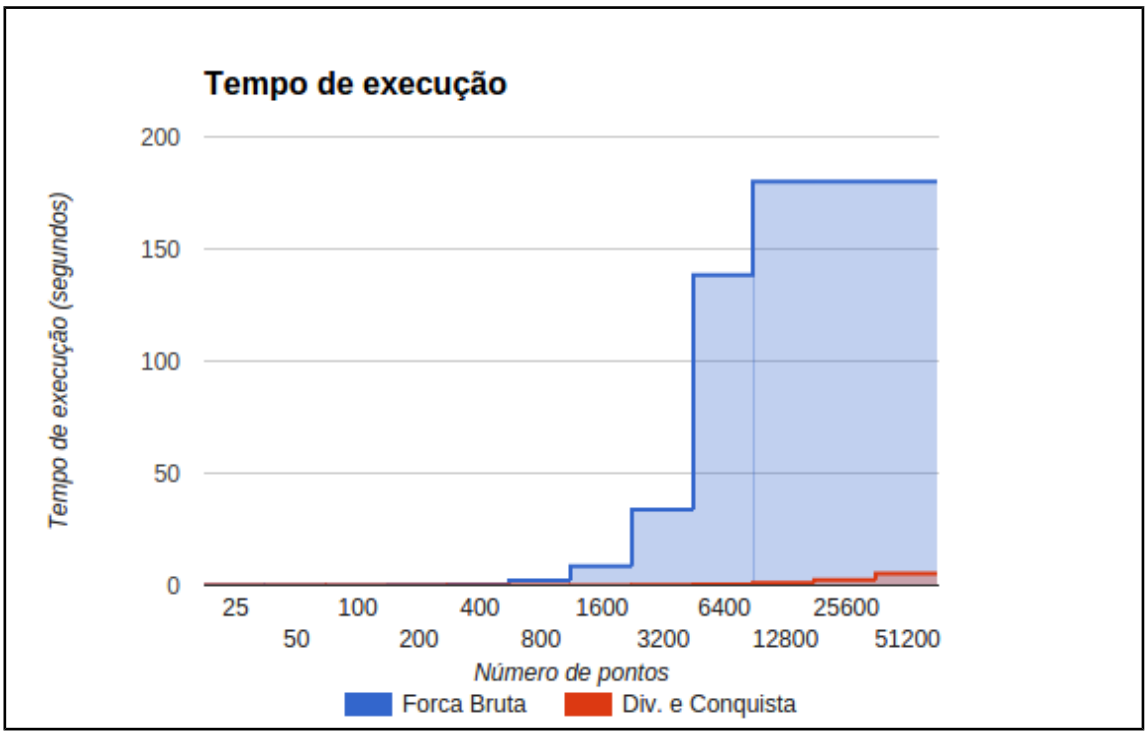
Além disso ele garante que após gerar as 120 instâncias não serão geradas mais instâncias, mesmo que o módulo seja chamado novamente ele retornará as 120 instâncias geradas anteriormente.

Execução (3_execute.py)

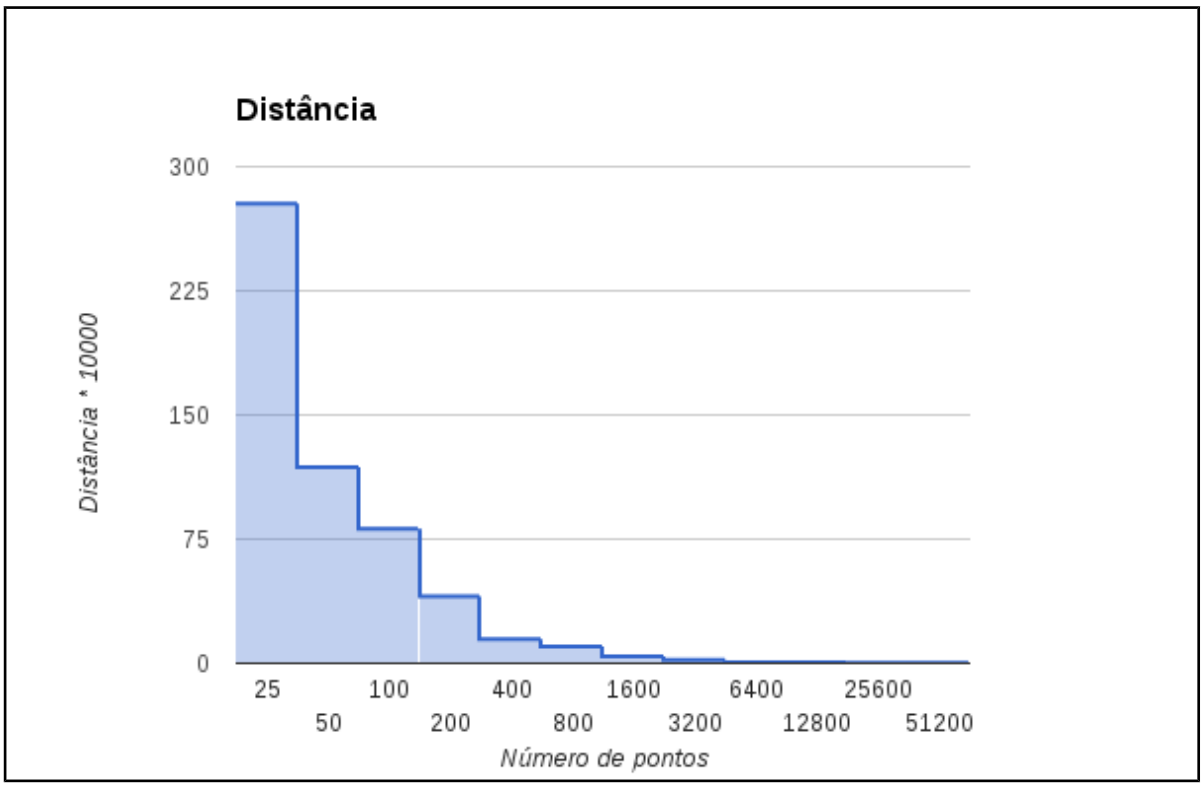
Módulo principal, que utiliza os outros módulos para gerar as 120 instâncias e computar o par de menor distância para cada instância usando o algoritmo de força bruta e de divisão e conquista, monitorando o tempo de execução e verificando se os resultados obtidos são os mesmos.

3 – Resultados e tempos de execução

Adicionamos em anexo ao trabalho um arquivo *log.txt* que contém as saídas dos algoritmos de força bruta e divisão e conquista para cada uma das 120 instâncias geradas. Os gráficos abaixo ilustram esses resultados. O primeiro apresenta um comparativo do tempo de execução para os dois métodos e o segundo auxilia na análise da relação entre quantidade de pontos e a menor distância encontrada.



Neste gráfico observamos claramente que a medida que o número de pontos aumenta o tempo de execução no algoritmo de força bruta (azul) aumenta em tempo quadrático e que por medida de controle a partir de 12800 pontos a execução fica incompleta por romper o limite de 3 minutos.



Analisando este gráfico percebemos que a menor distância encontrada diminui a medida que temos mais pontos no plano. Tal fenômeno pode ser explicado usando o princípio da casa dos pombos, pois temos um plano de tamanho limitado (1x1, especificado no enunciado), portanto aumentar a quantidade de pontos implica na aproximação de alguns pontos por falta de espaço.

URI para os dados dos gráficos: https://goo.gl/TcAUeQ