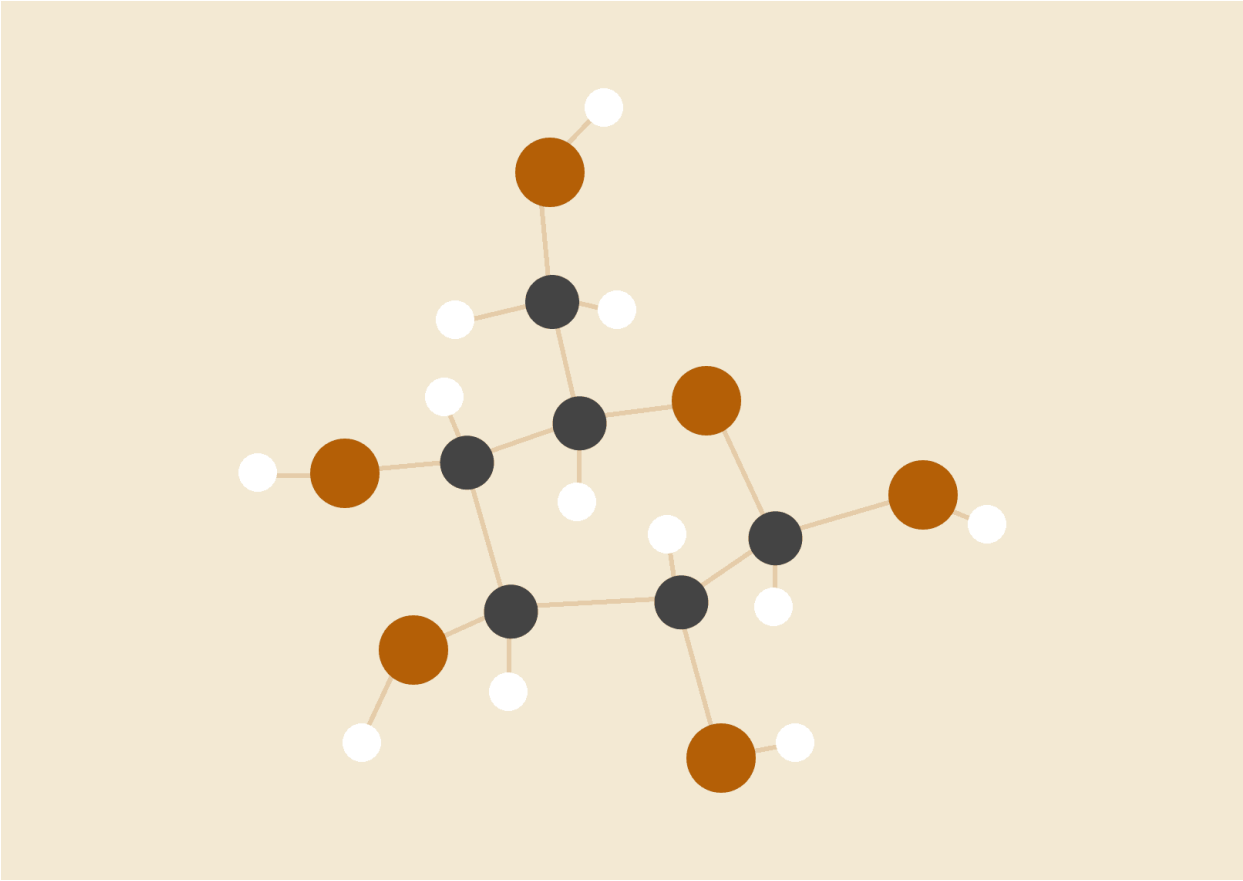
Relatório Par Mais Proximo



Vitor Yuji F. Matsushita (11921BCC021)

15/10/2023

Análise de Algoritmos

INTRODUÇÃO PAR MAIS PRÓXIMO (N^2)

Implementar a resolução para o Problema do Par Mais Próximo usando as seguintes abordagens.

Busca exaustiva com todos os pares, com complexidade O(n^2);

Divisão e conquista com a ordenação "simples" em cada chamada (informalmente vimos que esta solução tem complexidade O(n(lg n)^2)

Divisão e Conquista usando a técnica "pig back": "aproveitar" as excursões e trocar a ordenação - que gasta O(n lgn) por chamada - , por um merge menos complexo, resultando em um algoritmo final O(n lg n)

HIPÓTESE 1

Busca exaustiva com todos os pares, com complexidade O(n^2);

- Entrada de uma quantidade pontos, preenchidos com coordenadas aleatórias;
- Cálculo de distância de todos os pontos;
- Descoberta de qual a menor distância entre 2 pontos.
- Saída, a menor distância descoberta e os dois pontos com a menor distância.

PROCEDIMENTO Par mais Próximo (n^2)

1. Escolha de Quantidade de pontos com coordenadas X e Y
2. Calculo de distancia de Todos os pontos;
3. Saída de menor distância entre duas coordenada(Pontos) e o par de pontos

```
#Instruções

#Implementar a resolução para o Problema do Par Mais Proximo  usando  as
seguintes abordagens:



#Busca exaustiva com todos os pares, com complexidade O(n^2);
```

```
import math
from random import randint
import time

#Start do cronometro
start_time = time.perf_counter()
tam = int(input("Digite a quantidade de ponto no plano: "))
X = []
Y = []
k = int
par = [1]
for k in range (tam):
        X.append(randint(1,1000000))
        Y.append(randint(1,1000000))
print(X,Y)

def Dist(elemX1,elemY1,elemX2,elemY2):
        return abs(math.sqrt((elemX1 - elemX2) * (elemX1 - elemX2) +
(elemY1 - elemY2) * (elemY1 - elemY2)))



def ParProx(X,Y):
    d = 10000
    for i in range (2 , tam):
        for j in range (1 , i-1):
            if (Dist(X[i],Y[i],X[j],Y[j]))<d:
                d = Dist(X[i],Y[i],X[j],Y[j])
```

```
                par[0] = (X[i],Y[i]),(X[j],Y[j])
    return d


print("Menor distancia:", ParProx(X,Y))
print("O par mais proximo eh: ", par)
end_time = time.perf_counter()
execution_time = end_time - start_time
print(f"Programa executado em: {execution_time: .5f} segundos")
```

SAÍDA

```
Menor distancia: 170.09703113223347
O par mais proximo eh:  [((473951, 448085), (474014, 448243))]
Programa executado em:  36.02969 segundos
```

Quantidade de Pontos	Tamanho do Plano	Saída de Resultado
10000	1.000.000	Menor distância: 108.171160 Tempo: 34.40915 seg Par mais Prox: [((694473, 102911), (694368, 102885))]
100.000	1.000.000	Menor distância: 5.0 Tempo: 3682.38677 seg Par mais Prox: [((571599,35247),(571603,35244))]
1.000.000	1.000.000	Demorou muito. Tempo > 30 min

INTRODUÇÃO PAR MAIS PRÓXIMO DIVISÃO E CONQUISTA

Divisão e conquista com a ordenação "simples" em cada chamada (informalmente vimos que esta solução tem complexidade O(n(lg n)^2)

HIPÓTESE 1

- Entrada de uma quantidade pontos, preenchidos com coordenadas aleatórias;
- Divisão e conquista com a ordenação "simples" em cada chamada (informalmente vimos que esta solução tem complexidade O(n(lg n)^2)
- O algoritmo divide todos os pontos em dois conjuntos e chama recursivamente. Depois de dividir, ele encontra tempo O (nLogn) e finalmente encontra os pontos mais próximos na faixa no tempo O (n).
- Portanto, T (n) pode ser expresso da seguinte forma:
  T (n) = 2T (n / 2) + O (n) + O (n Log n) + O (n)
  T (n) = 2T (n / 2) + O (nLogn)
  T (n) = T (n x Logn x Logn) = O(n(lg n)^2)
- Saída, a menor distância descoberta.

PROCEDIMENTO PAR MAIS PRÓXIMO DIVISÃO E CONQUISTA

1. Escolha de Quantidade de pontos com coordenadas X e Y
2. Calculo de distancia de Todos os pontos;
3. Saída de menor distância entre duas coordenada(Pontos) e o par de pontos

```
import math

import copy

from random import randint

import time

#Divisão e conquista com a ordenação "simples" em cada chamada
(informalmente vimos que esta solução tem complexidade O(n(lg n)^2)
```

```
# O algoritmo divide todos os pontos em dois conjuntos e chama
recursivamente. Depois de dividir, ele encontra tempo O (nLogn) e
finalmente encontra os pontos mais próximos na faixa no tempo O (n).


# Portanto, T (n) pode ser expresso da seguinte forma:


#T (n) = 2T (n / 2) + O (n) + O (nLogn) + O (n)


#T (n) = 2T (n / 2) + O (nLogn )


#T (n) = T (nx Logn x Logn) = O(n(lg n)^2)




#Start do cronometro

start_time = time.perf_counter()



class Point():

    def __init__(point, x, y):

        point.x = x

        point.y = y



def dist(p1, p2):

    return math.sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y
- p2.y))
```

```
def FB(P, n):
    min_val = float('inf')
    for i in range(n):
        for j in range(i + 1, n):
            if dist(P[i], P[j]) < min_val:
                min_val = dist(P[i], P[j])

    return min_val


def smaisProx(s, tam, d):
    min_val = d
    for i in range(tam):
        j = i + 1
        while j < tam and (s[j].y-s[i].y) < min_val:
            min_val = dist(s[i], s[j])
```

```
            j += 1



    return min_val



def maisProxEntre(P, Q, n):



    if n <= 3:

        return FB(P, n)



    mid = n // 2

    midPoint = P[mid]



    pl = P[:mid]

    pr = P[mid:]



    dl = maisProxEntre(pl, Q, mid)

    dr = maisProxEntre(pr, Q, n - mid)
```

```
    d = min(dl, dr)



    sP = []

    sQ = []

    r = pl + pr

    for i in range(n):

        if abs(r[i].x - midPoint.x) < d:

            sP.append(r[i])

        if abs(Q[i].x - midPoint.x) < d:

            sQ.append(Q[i])



    sP.sort(key = lambda point: point.y) #<-- REQUIRED

    min_a = min(d, smaisProx(sP, len(sP), d))

    min_b = min(d, smaisProx(sQ, len(sQ), d))



    return min(min_a,min_b)
```

```
def maisProx(P, n):
    P.sort(key = lambda point: point.x)
    Q = copy.deepcopy(P)
    Q.sort(key = lambda point: point.y)

    return maisProxEntre(P, Q, n)
P = []
tam = int(input("Digite a quantidade de ponto no plano: "))
for k in range (tam):
    P.append(Point(randint(1,1000000),randint(1,1000000)))
n = len(P)
print("Menor distancia eh:",maisProx(P, n))
end_time = time.perf_counter()
execution_time = end_time - start_time
print(f"Programa executado em: {execution_time: .5f} segundos")
```

SAÍDA

```
Digite a quantidade de ponto no plano: 10000
Menor distancia eh: 86.57944328765345
Programa executado em:  3.00405 segundos
```

Quantidade de Pontos	Tamanho do Plano	Saída de Resultado
10000	1.000.000	Menor distância: 55.08175 Tempo: 6.02056 seg
100.000	1.000.000	Menor distância: 13.34166 Tempo:10.19247 seg
1.000.000	1.000.000	Menor distância: 1.0 Tempo: 45.71993 seg

INTRODUÇÃO PAR MAIS PRÓXIMO DIVISÃO E CONQUISTA (n log n)

Divisão e Conquista usando a técnica "pig back": "aproveitar" as excursões e trocar a ordenação - que gasta O(n lgn) por chamada - , por um merge menos complexo, resultando em um algoritmo final O(n lg n)

HIPÓTESE 1

- Entrada de uma quantidade pontos, preenchidos com coordenadas aleatórias;
- Divisão e conquista com a ordenação "simples" em cada chamada (informalmente vimos que esta solução tem complexidade O(n lg n)
- O algoritmo divide todos os pontos em dois conjuntos e chama recursivamente. Depois de dividir, ele encontra tempo O (nLogn) e finalmente encontra os pontos mais próximos na faixa no tempo O (n).
- Saída, a menor distância descoberta.

PROCEDIMENTO PAR MAIS PRÓXIMO DIVISÃO E CONQUISTA(n log n)

4. Escolha de Quantidade de pontos com coordenadas X e Y
5. Calculo de distancia de Todos os pontos;
6. Saída de menor distância entre duas coordenadas(Pontos).

```
# Divisao e Conquista usando a tecnica "pig back":
# "aproveitar" as recursões e trocar a ordenação -
# que gasta O(n lgn) por chamada - , por um merge menos complexo,
# resultando em um algoritmo final O(n lg n)
import copy
import math
from random import randint
import time


def mergeSort(alist):
    if len(alist)>1:
        mid = len(alist)//2
        lefthalf = alist[:mid]
        righthalf = alist[mid:]
        mergeSort(lefthalf)
        mergeSort(righthalf)


        i=0
        j=0
        k=0
        while i < len(lefthalf) and j < len(righthalf):
            if lefthalf[i].x < righthalf[j].x and lefthalf[i].y <
righthalf[j].y:
```

```
                alist[k]=lefthalf[i]
                i=i+1
            else:
                alist[k]=righthalf[j]
                j=j+1
            k=k+1

        while i < len(lefthalf):
            alist[k]=lefthalf[i]
            i=i+1
            k=k+1

        while j < len(righthalf):
            alist[k]=righthalf[j]
            j=j+1
            k=k+1
    return alist

#Start do cronometro
start_time = time.perf_counter()

class Point():
    def __init__(point, x, y):
        point.x = x
        point.y = y

def dist(p1, p2):
    return math.sqrt((p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y
- p2.y))
```

```
def FB(P, n):
    min_val = float('inf')
    for i in range(n):
        for j in range(i + 1, n):
            if dist(P[i], P[j]) < min_val:
                min_val = dist(P[i], P[j])

    return min_val


def smaisProx(s, tam, d):
    min_val = d
    for i in range(tam):
        j = i+1
        while j < tam and ((s[j].y) - (s[i].y)) < min_val:
            min_val = dist(s[i], s[j])
            j += 1

    return min_val


def maisProxEntre(P, Q, n):
    if n <= 3:
        return FB(P, n)
    mid = n // 2
    midPoint = P[mid]
    pl = P[:mid]
    pr = P[mid:]
```

```
    dl = maisProxEntre(pl, Q, mid)
    dr = maisProxEntre(pr, Q, n - mid)
    d = min(dl, dr)
    sP = []
    sQ = []
    r = pl + pr

    for i in range(n):
        if abs((r[i].x) - (midPoint.x)) < d:
            sP.append(r[i])
        if abs((Q[i].x) - (midPoint.x)) < d:
            sQ.append(Q[i])

    sP.sort(key = lambda point: point.y) #ordenado por y
    min_a = min(d, smaisProx(sP, len(sP), d))
    min_b = min(d, smaisProx(sQ, len(sQ), d))

    return min(min_a,min_b)

def maisProx(P, n):
    mergeSort(P)
    Q = copy.deepcopy(P)

    return maisProxEntre(P, Q, n)
P = []
tam = int(input("Digite a quantidade de ponto no plano: "))
for k in range (tam):
    P.append(Point(randint(1,1000000),randint(1,1000000)))
```

```
n = len(P)

print("Menor distancia eh:",maisProx(P, n))

end_time = time.perf_counter()

execution_time = end_time - start_time

print(f"Programa executado em: {execution_time: .5f} segundos")
```

SAÍDA

Quantidade de Pontos	Tamanho do Plano	Saída de Resultado
10000	1.000.000	Menor distância: 103.60069 Tempo: 3.37871 seg
100.000	1.000.000	Menor distância: 238.170 Tempo:8.06593 seg
1.000.000	1.000.000	Menor distância: 64.3816744 Tempo: 49.22189 seg

COMPARAÇÃO (considerando 1.000.000 ponto e Plano 1.000.000x1.000.000):

Métodos:	Saída de Resultado
Força Bruta (n^2)	Demorou muito. Tempo > 30 min
Divisão e Conquista	Menor distância: 1.0 Tempo: 45.71993 seg
Divisão e Conquista(“pig back”)	Menor distância: 64.3816744 Tempo: 49.22189 seg