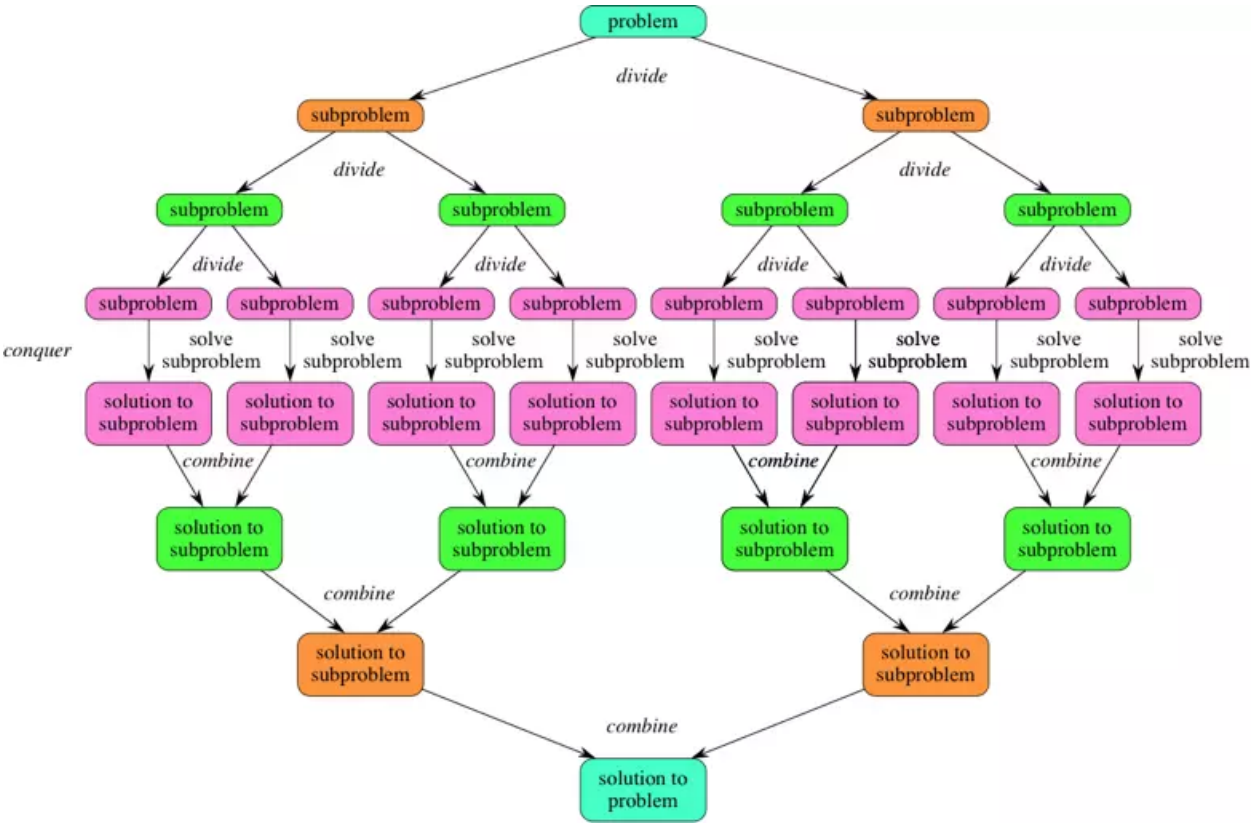
Homework 4: Divide and Conquer

Bùi Khánh Duy - 20001898

Phần 1: Lý thuyết phương pháp



Bước 1: Chia/Tách nhỏ

Bài toán ban đầu sẽ được chia thành các bài toán con cho đến khi không thể chia nhỏ được nữa. Các bài toán con sẽ trở thành 1 bước nhỏ trong việc giải quyết bài toán lớn.

Bước 2: Trị/Giải quyết bài toán con

Tìm phương án để giải quyết cho bài toán con một cách cụ thể.

Bước 3: Kết hợp lời giải lại để suy ra lời giải

Khi đã giải quyết xong các bài toán nhỏ, lặp lại và kết hợp lại những lời giải đó để suy ra kết quả cần tìm (có thể ở dạng đệ quy).

Phần 2: Bài tập tư duy

Exercise 1.

Stock Pricing Problem: Consider the stock price of **CareerMonk** in n consecutive days. That means the input consists of an array with stock prices of the company. We know that the stock price will not be the same on all the days. In the input stock prices there may be dates where the
stock is high when we can sell the current holdings, and there may be days when we can buy the stock. Now our problem is to find the day on which we can buy the stock and the day on which we can sell the stock so that we can make maximum profit.

Exercise 2:

We are testing “unbreakable” laptops and our goal is to find out how unbreakable they really are. In particular, we work in an n-story building and want to find out the lowest floor from which we can drop the laptop without breaking it (call this “the ceiling”). Suppose we are given two laptops and want to find the highest ceiling possible. Give an algorithm that minimizes the number of tries we need to make f(n) (hopefully, f(n) is sub-linear, as a linear f(n) yields a trivial solution).

Giải: Đây là bài toán tìm kiếm nhị phân, với hàm có dạng $f(x) = h$ với x là số tầng và h là độ cao và đảm bảo rằng này tăng theo x.

Mã giả của bài toán như sau:

```
function binary_search(f, n):
    low = 0
    high = n - 1 // max
    ans = -1
    while low <= high:
```

```
        mid = (low + high) / 2
        if isNotBreak(f[mid]): // if not break so we reduce the high
            ans = mid
            high = mid - 1
        elif f[mid] < x: // otherwise, we increase the high of next point.
            low = mid + 1
    return ans

function isNotBreak(x):
    // .. check if laptop have broken or not yet.
```

Đoạn mã giả này đã khử đệ quy, nhưng nếu với code đệ quy thì sẽ tính được độ phức tạp thời gian như sau:

$$T(n) = T(n/2) + 1$$

Độ phức tạp thời gian của việc tìm kiếm sẽ là $O(\log(n)$

Phần 3: Bài tập lập trình

Bài 1: Tìm kiếm nhị phân trên mảng đã sắp xếp.

```
def binary_search(arr, x):
    return binary_search_recursive(arr, 0, len(arr) - 1, x)

def binary_search_recursive(arr, low, high, x):
    if low > high:
        return -1

    mid = (low + high) // 2
    if arr[mid] == x:
        return mid
    elif arr[mid] < x:
        return binary_search_recursive(arr, mid + 1, high, x)
    else:
        return binary_search_recursive(arr, low, mid - 1, x)
```

Bài 2: Tìm kiếm giá trị min và max trên mảng

```
def find_min_max(arr, low, high):
    # Base case: n = 1
    if low == high:
```

```
        return arr[low], arr[low]

    # Base case: n = 2
    elif low + 1 == high:
        if arr[low] < arr[high]:
            return arr[low], arr[high]
        else:
            return arr[high], arr[low]

    # Divide the array into two parts
    else:
        mid = (low + high) // 2
        min1, max1 = find_min_max(arr, low, mid)
        min2, max2 = find_min_max(arr, mid + 1, high)

        # Merge the results
        return min(min1, min2), max(max1, max2)
```

Bài 3: Nhân ma trận bằng phương pháp Strassen

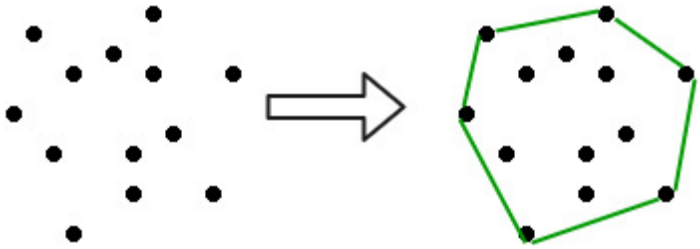
```
import numpy as np

def split(matrix):
  """
  Splits a given matrix into quarters.
  Input: nxn matrix
  Output: tuple containing 4 n/2 x n/2 matrices corresponding to a, b, c, d
  """
  row, col = matrix.shape
  row2, col2 = row//2, col//2
  return matrix[:row2, :col2], matrix[:row2, col2:], matrix[row2:, :col2], matrix[row2:, c
ol2:]

def strassen(x, y):
  """
  Computes matrix product by divide and conquer approach, recursively.
  Input: nxn matrices x and y
  Output: nxn matrix, product of x and y
  """

  # Base case when size of matrices is 1x1
  if len(x) == 1:
    return x * y

  # Splitting the matrices into quadrants. This will be done recursively
  # until the base case is reached.
  a, b, c, d = split(x)
  e, f, g, h = split(y)
```

```
  # Computing the 7 products, recursively (p1, p2...p7)
  p1 = strassen(a, f - h)
  p2 = strassen(a + b, h)
  p3 = strassen(c + d, e)
  p4 = strassen(d, g - e)
  p5 = strassen(a + d, e + h)
  p6 = strassen(b - d, g + h)
  p7 = strassen(a - c, e + f)

  # Computing the values of the 4 quadrants of the final matrix c
  c11 = p5 + p4 - p2 + p6
  c12 = p1 + p2
  c21 = p3 + p4
  c22 = p1 + p5 - p3 - p7

  # Combining the 4 quadrants into a single matrix by stacking horizontally and verticall
y.
  c = np.vstack((np.hstack((c11, c12)), np.hstack((c21, c22))))

  return c

a = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]
b = [[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]]

print(strassen(np.array(a), np.array(b)))
```

Phần 4: Đặt bài toán, thiết kế, phân tích và triển khai thuật toán

Thuật toán bao lồi

Phân tích: Bao lồi có hình dạng như sau



Hình 1

Với ví dụ ở hình 1, ta có 1 ví dụ về tập điểm như sau:

```
Input : points[] = {(0, 0), (0, 4), (-4, 0), (5, 0), (0, -6), (1, 0)};
Output : (-4, 0), (5, 0), (0, -6), (0, 4)
```

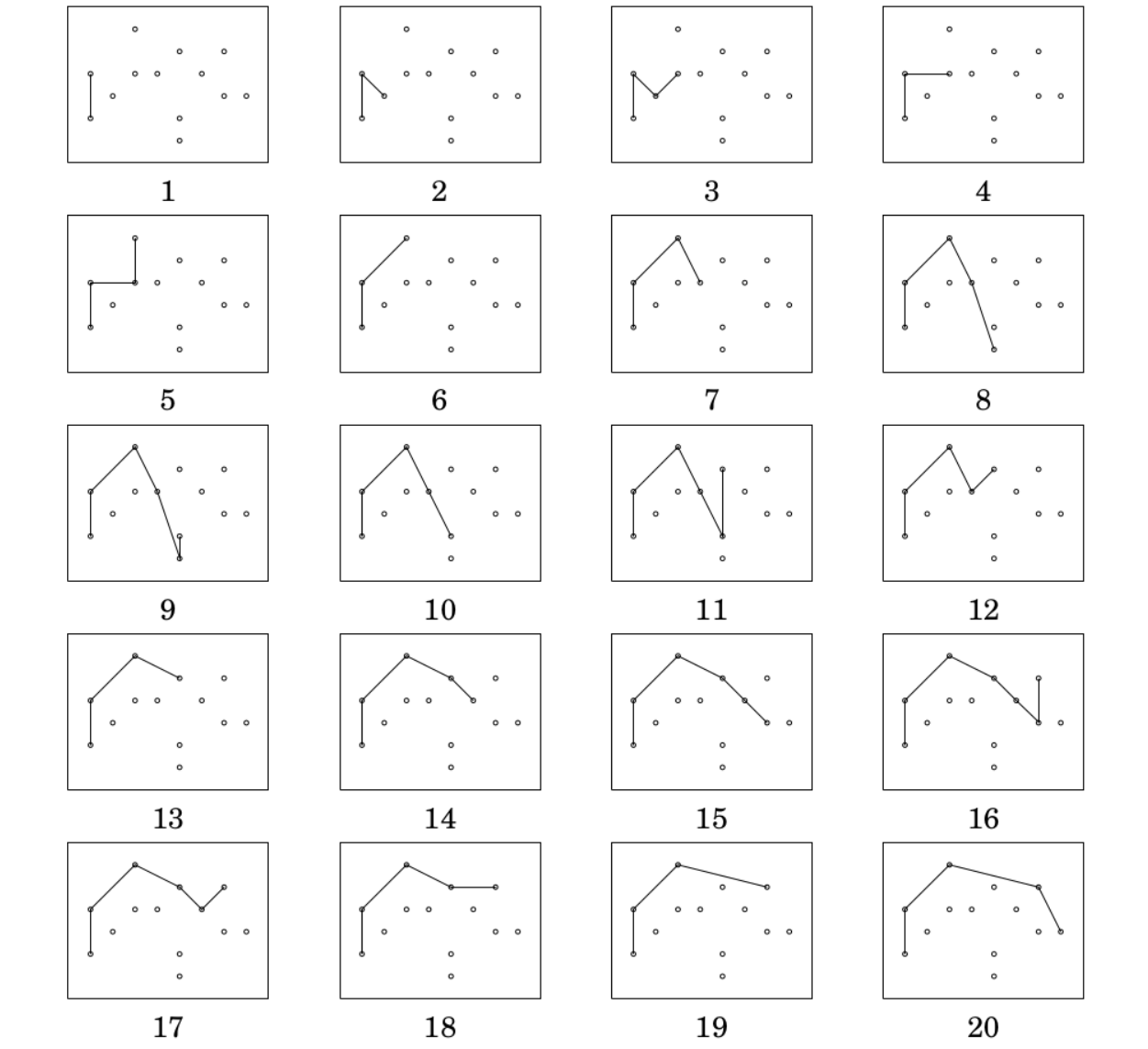
Ý tưởng

Sử dụng thuật toán chia để trị có ý tưởng đơn giản để chia bài toán con, nhưng việc code lại khá phức tạp. Ở đây sẽ sử dụng một thuật toán khác cho kết quả tương tự (cả về độ phức tạp) nhưng việc code lại nhẹ nhàng hơn.

Thuật toán Andrew: cung cấp một cách dễ dàng để xây dựng một bao lồi cho một tập các điểm trong thời gian $O(n \log n)$.

Đầu tiên thuật toán xác định các điểm trái nhất và phải nhất, và sau đó xây dựng một bao lồi trong 2 phần: đầu tiên lồi trên và sau đó là lồi dưới. Cả 2 phần là tương tự, vì vậy chúng ta tập trung vào xây dựng lồi trên.

Đầu tiên, chúng ta sắp xếp các điểm chủ yếu theo tọa độ x và thứ yếu theo tọa độ y. Sau đó, chúng ta duyệt qua các điểm và thêm mỗi điểm vào **bao (hull)**. Sau khi thêm một điểm vào bao, chúng ta phải đảm bảo rằng đoạn thẳng cuối cùng trong bao không được rẽ trái. Chừng nào nó còn rẽ trái, thì chúng ta sẽ xóa bỏ điểm cuối cùng thứ hai từ bao.



Nửa còn lại chúng ta làm tương tự, nhưng không điểm đã cho không được rẽ phải.

```
class Point(object):
    def __init__(self, x, y):
        self.x = x
        self.y = y

# A utility function to find next to top in a stack
def nextToTop(S):
    a = S.pop()
    b = S.pop()
    S.append(a)
    return b
```

```
# A utility function to swap two points
def swap(p1, p2):
    return p2, p1

# A utility function to return square of distance between
# two points
def distSq(p1, p2):
    return (p1.x - p2.x) * (p1.x - p2.x) + (p1.y - p2.y) * (p1.y - p2.y)

# Prints convex hull of a set of n points.
def convexHull(points, n):

    # There must be at least 3 points
    if (n < 3):
        return

    # Initialize Result
    hull = []

    # Find the leftmost point
    l = 0
    for i in range(1, n):
        if (points[i].x < points[l].x):
            l = i

    # Start from leftmost point, keep
    # moving counterclockwise until
    # reach the start point again
    # This loop runs O(h) times where h is
    # number of points in result or output.
    p = l
    q = 0
    while (True):

        # Add current point to result
        hull.append(points[p])

        # Search for a point 'q' such that
        # orientation(p, x, q) is counterclockwise
        # for all points 'x'. The idea is to keep
        # track of last visited most counterclock-
        # wise point in q. If any point 'i' is more
        # counterclock-wise than q, then update q.
        q = (p + 1) % n

        for i in range(0, n):

            # If i is more counterclockwise than
            # current q, then update q
            if (orientation(points[p], points[i], points[q]) == 2):
                q = i

        # Now q is the most counterclockwise with
        # respect to p. Set p as q for next iteration,
```

```
        # so that q is added to result 'hull'
        p = q

        # While we don't come to first point
        if (p == l):
            break

    # Print Result
    printHull(hull)

# To find orientation of ordered triplet (p, q, r).
# The function returns following values
# 0 --> p, q and r are colinear
# 1 --> Clockwise
# 2 --> Counterclockwise


def orientation(p, q, r):
    val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y)

    if (val == 0):
        return 0  # colinear
    elif (val > 0):
        return 1   # clock or wise
    else:
        return 2   # counterclock or wise

# Prints convex hull of a set of n points.
def printHull(hull):

    print("The points in Convex Hull are:")
    for i in range(len(hull)):
        print("(", hull[i].x, ", ", hull[i].y, ")")

# Driver Code
if __name__ == "__main__":

    points = [Point(0, 3), Point(2, 2), Point(1, 1), Point(2, 1), Point(3, 0), Point(0,
 0), Point(3, 3)]

    n = len(points)
    convexHull(points, n)
```

Output:

```
The points in Convex Hull are:
( 0 ,  3 )
( 0 ,  0 )
( 3 ,  0 )
( 3 ,  3 )
```

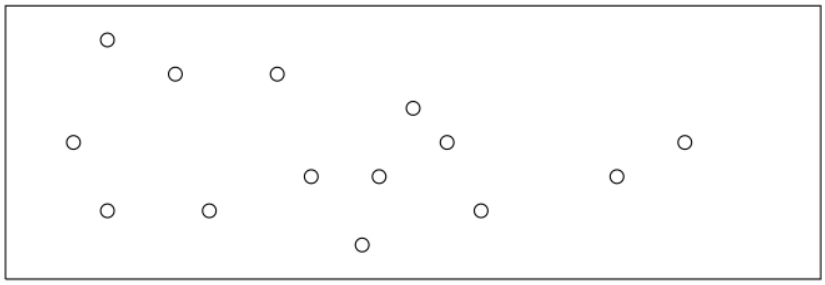
Độ phức tạp thời gian: $O(n \log n)$

trong đó: $O(n \log n)$ để sắp xếp tập các đỉnh, $O(n)$ để tính từng bao lồi ở nửa trên và nửa dưới.

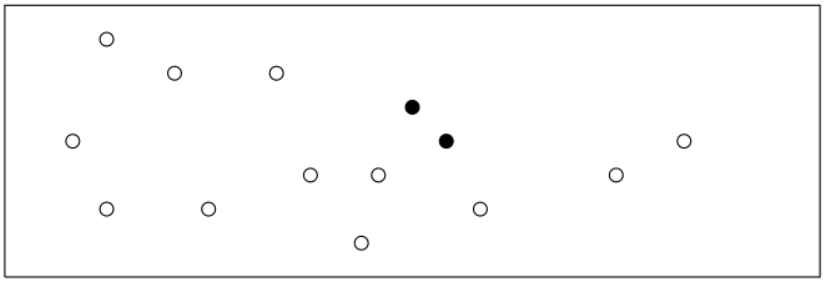
Bài toán tìm cặp điểm gần nhất

Mô tả bài toán

Cho một tập n điểm, tìm 2 điểm có khoảng cách Euclidean là nhỏ nhất. Ví dụ, các điểm là:



Ta cần tìm các điểm như sau

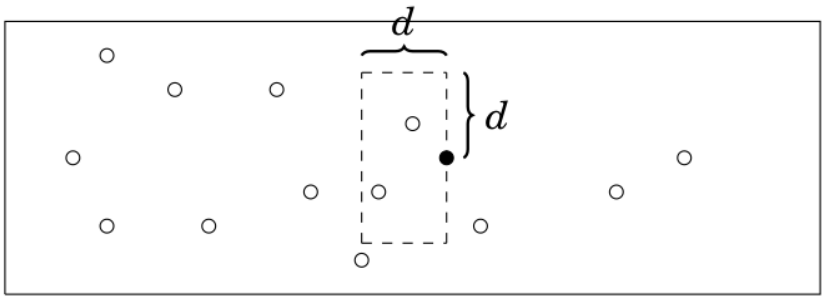


Đây là một bài toán có thể giải quyết trong thời gian $O(n \log n)$ bằng thuật toán quét đường (sweep line). Chúng ta duyệt qua các điểm từ trái sang phải và duy trì một giá trị d: khoảng cách nhỏ nhất giữa 2 điểm được tìm thấy tại một thời điểm. Tại mỗi điểm, chúng ta tìm một điểm gần nhất bên trái. Nếu khoảng cách nhỏ hơn d, thì nó là khoảng cách ngắn nhất mới và chúng ta cập nhật lại giá trị d.

Nếu điểm hiện tại là (x, y) và có một điểm nằm bên trái mà khoảng cách nhỏ hơn d, tọa độ x của điểm này phải nằm giữa $[x–d, x]$ và tọa độ y phải nằm giữa $[y–d, y + d]$.

Vì thế, chỉ cần xem xét các điểm có vị trí trong đoạn này, điều này giúp thuật toán hiệu quả hơn.

Ví dụ, trong hình sau, khu vực được đánh dấu bằng đường nét đứt chứa các điểm mà có thể nằm trong khoảng cách d từ điểm hoạt động:



Độ hiệu quả:

Độ hiệu quả của thuật toán được dựa trên yếu tố là vùng luôn chứa chỉ $O(1)$ điểm. Chúng ta có thể duyệt qua các điểm này trong thời gian $O(\log n)$ bởi duy trì một tập các điểm mà tọa độ x nằm giữa $[x - d, x]$, theo thứ tự tăng dần tương ứng với tọa độ y của chúng.

Độ phức tạp thời gian của thuật toán là $O(n \log n)$, vì chúng ta luôn duyệt qua n điểm và với mỗi điểm, tìm điểm gần nhất nằm bên trái, trong thời gian $O(\log n)$.

```
import sys
import math

# To find the closest pair of points
def closestPair(coordinates, n):

    # List of pairs to store points on plane
    points = [(coordinates[i][0], coordinates[i][1]) for i in range(n)]

    # Sort them according to their x-coordinates
    points.sort()

    # Minimum distance b/w points seen so far
    d = sys.maxsize

    # Keeping the points in increasing order
    st = set()
```

```
    st.add(points[0])

    for i in range(1, n):
        l = set([p for p in st if p[0] >= points[i][0]-d and p[1] >= points[i][1]-d])
        r = set([p for p in st if p[0] <= points[i][0]+d and p[1] <= points[i][1]+d])
        intersection = l & r
        if len(intersection) == 0:
            continue

        for val in intersection:
            dis = math.pow(points[i][0] - val[0], 2) + math.pow(points[i][1] - val[1], 2)

            # Updating the minimum distance dis
            if d > dis:
                d = dis

        st.add(points[i])

    return d

# Points on a plane P[i] = (x, y)
P = [(1, 2), (2, 3), (3, 4), (5, 6), (2, 1)]
n = len(P)

# Function call
print("The smallest distance is", closestPair(P, n))
```