Name:Vishal Chauhan

PRN:2020BTECS00090

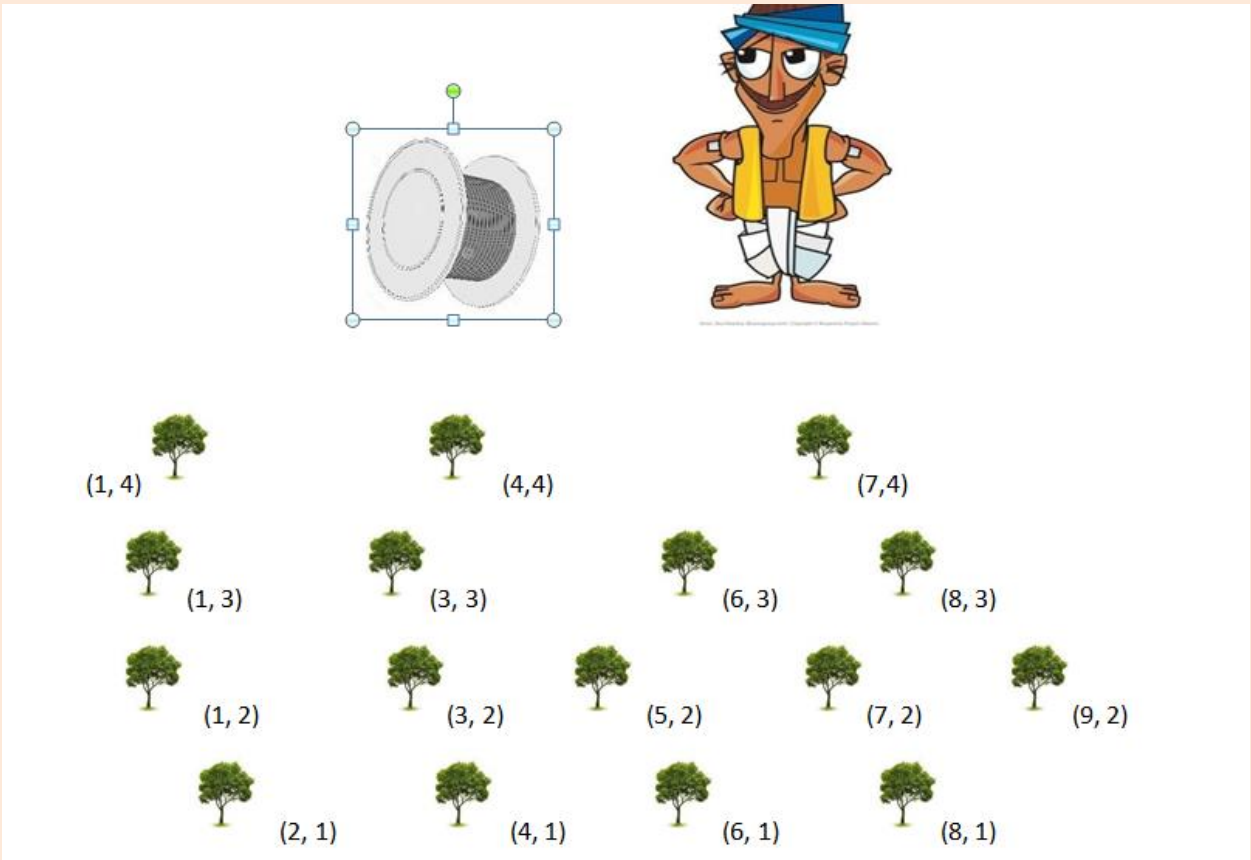
Sub:DAA

Assignment no:5

Topic:Divide and conquer strategy

Farmer has planted only mango trees in his farm as shown in figure (indicates position of each tree with *x, y* coordinates) It is not a fixed size farm. He wants to protect those mango trees from a thief, for that he decided to round up a tree with electric wire. But as the farm is not of fixed size, he decided to find all set of mango trees which will cover all the remaining mango trees and then he will wind all tree with electric wire.

Implement an algorithm to find set of mango trees which cover all remaining mango trees.Also find perimeter of rounded wire.



Solution:-

I→Algorithm

1. Find the point with minimum x-coordinate lets say, min_x and similarly the point with maximum x-coordinate, max_x.
2. Make a line joining these two points, say **L**. This line will divide the whole set into two parts. Take both the parts one by one and proceed further.
3. For a part, find the point P with maximum distance from the line L. P forms a triangle with the points min_x, max_x. It is clear that the points residing inside this triangle can never be the part of convex hull.
4. The above step divides the problem into two sub-problems (solved recursively). Now the line joining the points P and min_x and the line joining the points P and max_x are new lines and the points residing outside the triangle is the set of points. Repeat point no. 3 till there no point left with the line. Add the end points of this point to the convex hull.

Implementation:→

```cpp
// C++ implementation of the approach
#include <bits/stdc++.h>
#include <iostream>
#include <vector>
#include <map>
#include <queue>
#include <set>
#include <string>
#define llu long long int
using namespace std;


struct Point {

    llu x, y;

    bool operator<(Point p)
    {
        return x < p.x || (x == p.x && y < p.y);
    }
};

// Cross product of two vectors OA and OB
// returns positive for counter clockwise
// turn and negative for clockwise turn

llu cross_product(Point O, Point A, Point B)
{
    return (A.x - O.x) * (B.y - O.y)
        - (A.y - O.y) * (B.x - O.x);
```

```
}

// Returns a list of points on the convex hull
// in counter-clockwise order
vector<Point> convex_hull(vector<Point> A)
{
    int n = A.size(), k = 0;

    if (n <= 3)
        return A;

    vector<Point> ans(2 * n);

    // Sort points lexicographically
    sort(A.begin(), A.end());

    // Build lower hull
    for (int i = 0; i < n; ++i) {

        // If the point at K-1 position is not a part
        // of hull as vector from ans[k-2] to ans[k-1]
        // and ans[k-2] to A[i] has a clockwise turn
        while (k >= 2
            && cross_product(ans[k - 2],
                             ans[k - 1], A[i]) <= 0)
            k--;
        ans[k++] = A[i];
    }

    // Build upper hull
    for (size_t i = n - 1, t = k + 1; i > 0; --i) {

        // If the point at K-1 position is not a part
        // of hull as vector from ans[k-2] to ans[k-1]
        // and ans[k-2] to A[i] has a clockwise turn
        while (k >= t
            && cross_product(ans[k - 2],
                             ans[k - 1], A[i - 1]) <= 0)
            k--;
        ans[k++] = A[i - 1];
    }

    // Resize the array to desired size
    ans.resize(k - 1);
```

```
    return ans;
}

// Function to return the distance between two points
double dist(Point a, Point b)
{
    return sqrt((a.x - b.x) * (a.x - b.x)
                + (a.y - b.y) * (a.y - b.y));
}

// Function to return the perimeter of the convex hull
double perimeter(vector<Point> ans)
{
    double perimeter = 0.0;

    // Find the distance between adjacent points
    for (int i = 0; i < ans.size() - 1; i++) {
        perimeter += dist(ans[i], ans[i + 1]);
    }

    // Add the distance between first and last point
    perimeter += dist(ans[0], ans[ans.size() - 1]);

    return perimeter;
}

// Driver code
void print_all_boundary_point(vector<Point> ans)
{
    for(int i=0;i<ans.size();i++)
    {
        cout<<"[ "<<ans[i].x<<" , "<<ans[i].y<<" ]"<<"  ";
    }
}
int main()
{
    vector<Point>
points={{1,4},{1,3},{1,2},{2,1},{4,4},{3,3},{3,2},{4,1},{6,3},{5,2},{6,1},{7,4},{
7,2},{8,1},{8,3},{9,2}};

    // Find the convex hull
    vector<Point> ans = convex_hull(points);
    cout<<"Followinng are the bounded region :\n";
    print_all_boundary_point(ans);
```

```
    // Find the perimeter of convex polygon
    cout<<"\nthe perimeter of bounded region:" << perimeter(ans)<<endl;;

    return 0;
}
```

OUTPUT:→

```
PS C:\vis-clg+ass+all\DAA\ass5> cd "c:\vis-clg+ass+all\DAA\ass5\" ; if ($?) { g++
q1_ii.cpp -o q1_ii } ; if ($?) { .\q1_ii }

Followinng are the bounded region :
[ 1 , 2 ]  [ 2 , 1 ]  [ 8 , 1 ]  [ 9 , 2 ]  [ 7 , 4 ]  [ 1 , 4 ]
the perimeter of bounded region:19.6569
```

TIME AND SPACE COMPLEXITY:→

The time complexity of the above algorithm in the average case is of O(NlogN) while in worst case it gives the time complexity of O(N^2). The space complexity is O(K) as extra space is used to store the convex hull points where K is the number of points in the convex hull.

****