Lab -

DIP : Design and Analysis of Algorithm

AIM : Implement Geometric Algorithm which check the following

(1) Check whether a given polygon, represented by a set of points is CONVEX or CONCAVE

(2) Graham Scan Algorithm for convex Hull

(3) Convex Hull Using Divide & Conquer

Name : Ambalia Harshit

Department : Computer Engineering (M.Tech sem II)

Roll no : MT001

Date : Jan 2024

Program 01 : Check whether a given polygon, represented by a set of points is CONVEX or CONCAVE

- **Algorithm :**

  1) Initialize p as the leftmost point.

  2) Do the following while we don't come back to the first (or leftmost) point.

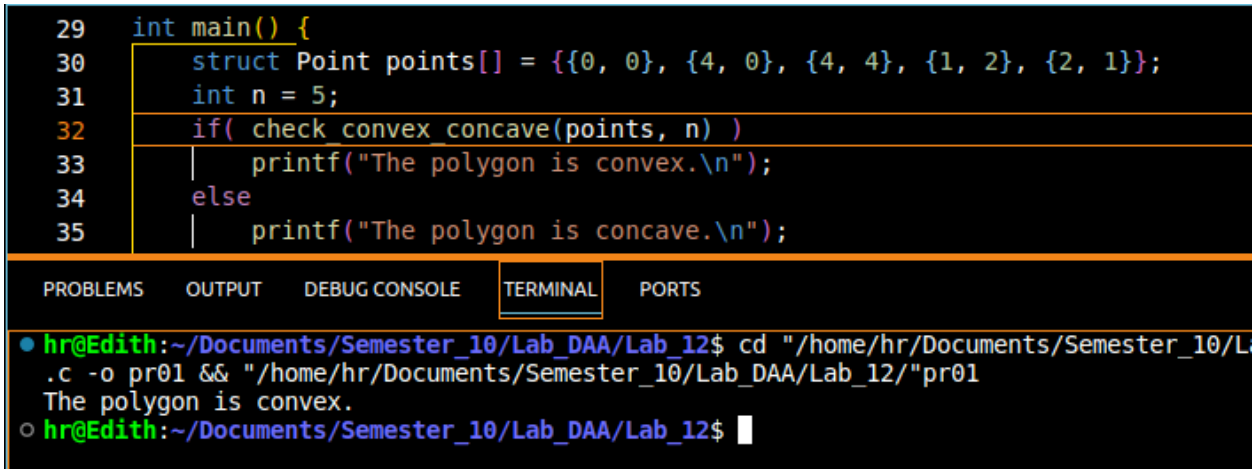
  - The next point q is the point, such that the triplet (p, q, r) is counter clockwise for any other point r. To find this, we simply initialize q as the next point, then we traverse through all points. For any point i, if i is more counter clockwise, i.e., orientation(p, i, q) is counter clockwise, then we update q as i. Our final value of q is going to be the most counter clockwise point.
  - next[p] = q (Store q as next of p in the output convex hull).
  - p=q (Set p as q for the Next Iteration)

- **Code :**

```
#include <stdio.h>
#include <stdbool.h>

struct Point {
   int x, y;
};

int orientation(struct Point p, struct Point q, struct Point r) {
   int val = (q.y-p.y)*(r.x-q.x) - (q.x-p.x)*(r.y-q.y);
   if( val==0 )
       return 0;
   return( val>0 ) ? 1 : 2;
}
```

```
bool check_convex_concave(struct Point points[], int n) {
   if( n<3 )
       return false;
   bool temp1=false, temp2=false;
   for( int i=0;i<n;i++ ) {
       int orient = orientation( points[i], points[(i+1)%n],
points[(i+2)%n] );
       if( orient>0 )
           temp1 = true;
       else if( orient<0 )
           temp2 = true;
   }
   return !(temp1 && temp2);
}

int main() {
   struct Point points[] = {{0, 0}, {4, 0}, {4, 4}, {1, 2}, {2, 1}};
   int n = 5;
   if( check_convex_concave(points, n) )
       printf("The polygon is convex.\n");
   else
       printf("The polygon is concave.\n");
   return 0;
}
```

- **Output Screen-shots / Tracing :**



Program 02 : Graham Scan Algorithm for convex Hull

- **Algorithm :**

The step by step working of a Graham Scan Algorithms on the point set is given below.

- Find the point (p0) with the smallest y-coordinate. In case of a tie, choose the point with the smallest x-coordinate. This step takes O(n) time.
- Sort the points based on the polar angle i.e. the angle made by the line with the x -axis. While implementing, we don't calculate the angle, instead, we calculate the relative orientation of two points to find out which point makes the larger angle. This can be explained with the help of a figure shown below.
- To find out whether the line P0P1 or the line P0P3 makes the larger angle with the x -axis, we calculate the cross-product of vector P1P0 and vector P1P3. If the cross-product is positive, that means vector P1P0 is clockwise from vector P1P3 with respect to the x-axis. This indicates that the angle made by the vector P1P3. is larger. We can use any sorting algorithm that has complexity O(nlogn).
- After sorting, we check for the collinear points. If we find any collinear points, we keep the furthest point from P0 and remove all other points. This step takes O(n) time.
- First two points in the sorted list are always in the convex hull. In the above figure, points P0 and P1 are the vertices of the convex hull. We maintain a stack data structure to keep track of the convex hull vertices. We push these two points and the next point in the list (points P0,P1 and P3 in the figure above) to the stack.
- Now we check if the next point in the list turns left or right from the two points on the top of the stack. If it turns left, we push this item on the stack. If it turns right, we remove the item on the top of the stack and repeat this process for remaining items. This step takes O(n) time.

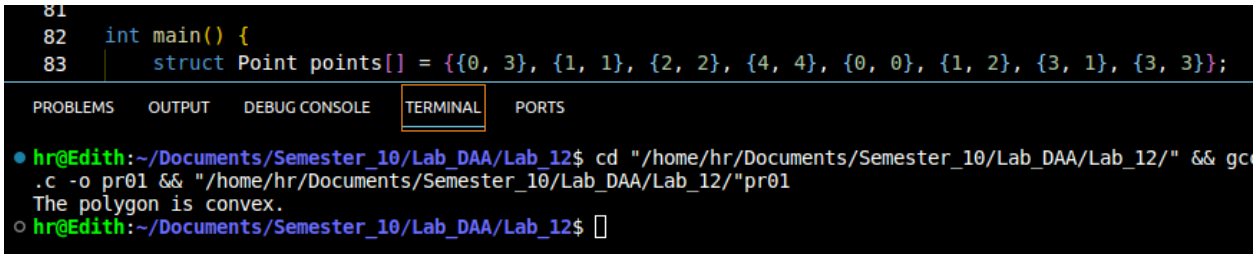
- **Code :**

```
#include <stdio.h>
#include <stdlib.h>

struct Point {
   int x, y;
};

struct Point p0;

struct Point next_to_top(struct Point *S, int *top) {
   struct Point p = S[*top];
   (*top)--;
   struct Point res = S[*top];
   S[++(*top)] = p;
   return res;
}

void swap(struct Point *p1, struct Point *p2) {
   struct Point temp = *p1;
   *p1 = *p2;
   *p2 = temp;
}

int distance_square(struct Point p1, struct Point p2) {
   return (p1.x - p2.x)*(p1.x - p2.x) + (p1.y - p2.y)*(p1.y - p2.y);
}

int orientation(struct Point p, struct Point q, struct Point r) {
   int val = (q.y - p.y) * (r.x - q.x) - (q.x - p.x) * (r.y - q.y);
   if (val == 0) return 0;
   return (val > 0) ? 1 : 2;
}

int compare(const void *vp1, const void *vp2) {
   struct Point *p1 = (struct Point *)vp1;
   struct Point *p2 = (struct Point *)vp2;
   int o = orientation(p0, *p1, *p2);
   if (o == 0)
```

```
        return (distance_square(p0, *p2) >= distance_square(p0, *p1)) ? -1
: 1;
    return (o == 2) ? -1 : 1;
}

void my_convex_hull(struct Point points[], int n) {
    int ymin = points[0].y, min = 0;
    for (int i = 1; i < n; i++) {
        int y = points[i].y;
        if ((y < ymin) || (ymin == y && points[i].x < points[min].x))
            ymin = points[i].y, min = i;
    }
    swap(&points[0], &points[min]);
    p0 = points[0];
    qsort(&points[1], n - 1, sizeof(struct Point), compare);

    int m = 1;
    for (int i = 1; i < n; i++) {
        while (i < n - 1 && orientation(p0, points[i], points[i + 1]) == 0)
            i++;
        points[m] = points[i];
        m++;
    }

    if (m < 3) return;

    struct Point S[n];
    int top = -1;
    S[++top] = points[0];
    S[++top] = points[1];
    S[++top] = points[2];

    for (int i = 3; i < m; i++) {
        while (orientation(next_to_top(S, &top), S[top], points[i]) != 2)
            top--;
        S[++top] = points[i];
    }

    while (top >= 0) {
        struct Point p = S[top--];
```

```
        printf("(%d, %d)\n", p.x, p.y);
    }
}

int main() {
    struct Point points[] = {{0, 3}, {1, 1}, {2, 2}, {4, 4}, {0, 0}, {1,
2}, {3, 1}, {3, 3}};
    int n = sizeof(points) / sizeof(points[0]);
    my_convex_hull(points, n);
    return 0;
}
```

- **Output Screen-shots / Tracing :**



Program 03 : Convex Hull Using Divide & Conquer

- Algorithms :

```
Algorithm ConvexHull(P)
        // P is a set of input points
        Sort all the points in P and find two extreme points A and B
        S1 ← Set of points right to the line AB
        S2 ← Set of points right to the line BA
        Solution ← AB followed by BA
        Call FindHull(S1, A, B)
        Call FindHull(S2, B, A)


Algorithm FindHull(P, A, B)
if isEmpty(P) then
        return
else
        C ← Orthogonally farthest point from AB
        Solution ← Replace AB by AC followed by CB
        Partition P – { C } in X0, X1 and X2
        Discard X0 in side triangle
        Call FindHull(X1, A, C)
        Call FindHull(X2, C, B)
End
```

- Code :

```
// A divide and conquer program to find convex
// hull of a given set of points.
#include<bits/stdc++.h>
using namespace std;

// stores the centre of polygon (It is made
// global because it is used in compare function)
pair<int, int> mid;

// determines the quadrant of a point
// (used in compare())
int quad(pair<int, int> p) {
   if (p.first >= 0 && p.second >= 0)
       return 1;
   if (p.first <= 0 && p.second >= 0)
       return 2;
   if (p.first <= 0 && p.second <= 0)
       return 3;
   return 4;
}

// Checks whether the line is crossing the polygon
int orientation(pair<int, int> a, pair<int, int> b, pair<int, int> c) {
   int res = (b.second-a.second)*(c.first-b.first) -
(c.second-b.second)*(b.first-a.first);
   if (res == 0)
       return 0;
   if (res > 0)
       return 1;
   return -1;
}

// compare function for sorting
bool compare(pair<int, int> p1, pair<int, int> q1) {
   pair<int, int> p = make_pair(p1.first - mid.first, p1.second -
mid.second);
   pair<int, int> q = make_pair(q1.first - mid.first, q1.second -
mid.second);
```

```
    int one = quad(p);
    int two = quad(q);
    if (one != two)
        return (one < two);
    return (p.second*q.first < q.second*p.first);
}

// Finds upper tangent of two polygons 'a' and 'b'
// represented as two vectors.
vector<pair<int, int>> merger(vector<pair<int, int> > a, vector<pair<int,
int> > b) {
    // n1 -> number of points in polygon a
    // n2 -> number of points in polygon b
    int n1 = a.size(), n2 = b.size();
    int ia = 0, ib = 0;
    for (int i=1; i<n1; i++)
        if (a[i].first > a[ia].first)
            ia = i;
    // ib -> leftmost point of b
    for (int i=1; i<n2; i++)
        if (b[i].first < b[ib].first)
            ib=i;
    // finding the upper tangent
    int inda = ia, indb = ib;
    bool done = 0;
    while (!done) {
        done = 1;
        while (orientation(b[indb], a[inda], a[(inda+1)%n1]) >=0)
            inda = (inda + 1) % n1;
        while (orientation(a[inda], b[indb], b[(n2+indb-1)%n2]) <=0) {
            indb = (n2+indb-1)%n2;
            done = 0;
        }
    }
    int uppera = inda, upperb = indb;
    inda = ia, indb=ib;
    done = 0;
    int g = 0;
    while (!done) { //finding the lower tangent
        done = 1;
```

```
        while (orientation(a[inda], b[indb], b[(indb+1)%n2])>=0)
            indb=(indb+1)%n2;
        while (orientation(b[indb], a[inda], a[(n1+inda-1)%n1])<=0) {
            inda=(n1+inda-1)%n1;
            done=0;
        }
    }
    int lowera = inda, lowerb = indb;
    vector<pair<int, int>> ret;
    //ret contains the convex hull after merging the two convex hulls
    //with the points sorted in anti-clockwise order
    int ind = uppera;
    ret.push_back(a[uppera]);
    while (ind != lowera) {
        ind = (ind+1)%n1;
        ret.push_back(a[ind]);
    }
    ind = lowerb;
    ret.push_back(b[lowerb]);
    while (ind != upperb) {
        ind = (ind+1)%n2;
        ret.push_back(b[ind]);
    }
    return ret;
}

// Brute force algorithm to find convex hull for a set
// of less than 6 points
vector<pair<int, int>> bruteHull(vector<pair<int, int>> a) {
    // Take any pair of points from the set and check
    // whether it is the edge of the convex hull or not.
    // if all the remaining points are on the same side
    // of the line then the line is the edge of convex
    // hull otherwise not
    set<pair<int, int> >s;
    for (int i=0; i<a.size(); i++) {
        for (int j=i+1; j<a.size(); j++) {
            int x1 = a[i].first, x2 = a[j].first;
            int y1 = a[i].second, y2 = a[j].second;
            int a1 = y1-y2;
```

```
            int b1 = x2-x1;
            int c1 = x1*y2-y1*x2;
            int pos = 0, neg = 0;
            for (int k=0; k<a.size(); k++) {
                if (a1*a[k].first+b1*a[k].second+c1 <= 0)
                    neg++;
                if (a1*a[k].first+b1*a[k].second+c1 >= 0)
                    pos++;
            }
            if (pos == a.size() || neg == a.size()) {
                s.insert(a[i]);
                s.insert(a[j]);
            }
        }
    }
    vector<pair<int, int>>ret;
    for (auto e:s)
        ret.push_back(e);
    // Sorting the points in the anti-clockwise order
    mid = {0, 0};
    int n = ret.size();
    for (int i=0; i<n; i++) {
        mid.first += ret[i].first;
        mid.second += ret[i].second;
        ret[i].first *= n;
        ret[i].second *= n;
    }
    sort(ret.begin(), ret.end(), compare);
    for (int i=0; i<n; i++)
        ret[i] = make_pair(ret[i].first/n, ret[i].second/n);
    return ret;
}
// Returns the convex hull for the given set of points
vector<pair<int, int>> divide(vector<pair<int, int>> a) {
    // If the number of points is less than 6 then the
    // function uses the brute algorithm to find the
    // convex hull
    if (a.size() <= 5)
        return bruteHull(a);
    // left contains the left half points
```

```
    // right contains the right half points
    vector<pair<int, int>>left, right;
    for (int i=0; i<a.size()/2; i++)
        left.push_back(a[i]);
    for (int i=a.size()/2; i<a.size(); i++)
        right.push_back(a[i]);
    // convex hull for the left and right sets
    vector<pair<int, int>>left_hull = divide(left);
    vector<pair<int, int>>right_hull = divide(right);
    // merging the convex hulls
    return merger(left_hull, right_hull);
}

// Driver code
int main() {
    vector<pair<int, int> > a;
    a.push_back(make_pair(0, 0));
    a.push_back(make_pair(1, -4));
    a.push_back(make_pair(-1, -5));
    a.push_back(make_pair(-5, -3));
    a.push_back(make_pair(-3, -1));
    a.push_back(make_pair(-1, -3));
    a.push_back(make_pair(-2, -2));
    a.push_back(make_pair(-1, -1));
    a.push_back(make_pair(-2, -1));
    a.push_back(make_pair(-1, 1));
    int n = a.size();
    // sorting the set of points according
    // to the x-coordinate
    sort(a.begin(), a.end());
    vector<pair<int, int> >ans = divide(a);
    cout << "convex hull:\n";
    for (auto e:ans)
       cout << e.first << " " << e.second << endl;
    return 0;
}
```

- Output Screen-shots / Tracing :