acm.mipt.ru

олимпиады по программированию на Физтехе

Выпуклая оболочка точек на плоскости

- теория?

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define EPS 1E-10

int debug = 0;

typedef struct point
{
  double x, y;
} point_t;

typedef struct line
{
  point_t a, b;
} line_t;

typedef struct ppoly
{
  int n;         // number of points in polyline
  point_t **ppts;      //array of pointers to points
} ppoly_t;

typedef struct poly
{
  int n;         // number of points in polyline
  point_t *pts;         //array of points
} poly_t;



/*
   Current axes origin for cmp function
*/
point_t *orig;

/*
   Calculates sign of VectorProduct [ orig->a, orig->b ].
*/
int
cmp (const void *aa, const void *bb)
{
   point_t *a = *(point_t **) aa;
     point_t *b = *(point_t **) bb;
     double s = (a->x - orig->x) * (b->y - orig->y) - (b->x - orig->x) * (a->y - orig->y);
     if (fabs (s) < EPS)
     {
       s = (a->x - orig->x) * (a->x - orig->x) + (a->y - orig->y) * (a->y - orig->y)
           - (b->x - orig->x) * (b->x - orig->x) + (b->y - orig->y) * (b->y - orig->y);
         if (fabs (s) < EPS)
         return 0;
         if (s < 0)
         return 1;
        return -1;
     };
     if (s < 0)
       return 1;
  return -1;
}


/*
   Calculates convex hull (polyline) of given array of points
*/
ppoly_t
convex_hull (point_t * pts, int n)
{
  int sp;         /* Stack pointer */
  int i, j, i_min = 0;
  point_t **stack = (point_t **) malloc ((n + 1) * sizeof (point_t *));
  ppoly_t result;

  for (i = 0; i < n; i++)
  {
      if (pts[i].y < pts[i_min].y)
      i_min = i;
      stack[i] = pts + i;
  }

  orig = stack[i_min];
  stack[i_min] = stack[0];
  stack[0] = orig;
  qsort (stack + 1, n - 1, sizeof (point_t *), cmp);
  sp = 2;
  i = 2;
  while (i < n)
  {
      orig = stack[sp - 1];
      if (cmp (&stack[sp], &stack[(i + 1) % n]) <= 0)
      stack[++sp] = stack[(++i) % n];
      else
```

Поиск
Поиск
Раздел «Алгоритмы»
Главная
Форум
Ссылки
El Judge
Инструменты:
Поиск
Изменения
Index
Статистика
Разделы
Информация
Алгоритмы
Язык Си
Язык Ruby
Язык Ассемблера
El Judge
Парадигмы
Образование
Сети
Objective C
Logon>>

```
      sp--;
  }
  result.ppts = stack;
  result.n = sp;
  return result;
}

void
free_poly (poly_t p)
{
  if (p.pts)
    free (p.pts);
}

void
free_ppoly (ppoly_t p)
{
  if (p.ppts)
    free (p.ppts);
}

/*
    Interesection of line ..--a0--a1--..  and  segment [b0, b1].
   Return TRUE if there is the intersection and put intersection point in variable *res;
    Otherwise return FALSE;
*/

int
line_segment_intersect (point_t a0, point_t a1, point_t b0, point_t b1,
         point_t * res)
{
  point_t da, db;
  double det = 0;
  double t_a = 0.0, t_b = 0.0;
  da.x = a1.x - a0.x;
  da.y = a1.y - a0.y;
  db.x = b1.x - b0.x;
  db.y = b1.y - b0.y;

  // line 1 = a0 + da*t_a
  // line 2 = b0 + db*t_b

  // Solve for t_a and t_b.

  // Check for divide by zero
  det = da.x * db.y - da.y * db.x;
  if (fabs (det) < EPS)
    return 0;

  t_a = db.x * (a0.y - b0.y) - db.y * (a0.x - b0.x);
  t_a /= det;

  t_b = da.x * (a0.y - b0.y) - da.y * (a0.x - b0.x);
  t_b /= det;

  if (t_b >= -EPS && t_b <= 1.0 + EPS)
    {
      // There is intersection
      res->x = db.x * t_b + b0.x;
      res->y = db.y * t_b + b0.y;
      return 1;
    }
  else
    {
      return 0;
    }
}

void
no_memory ()
{
  fprintf (stderr, "No memory\n");
  exit (2);
}

inline double
distance_rec (point_t a, point_t b)
{
  double t1 = fabs (a.x - b.x);
  double t2 = fabs (a.y - b.y);
  if (t1 > t2)
    return t1;
  return t2;
}

poly_t
line_poly_intersect (line_t line, ppoly_t ppoly)
{
   poly_t res;
     int i;
   int size = 2;
     int side;
     int prev_side;
     point_t *pb = &line.b;


     res.pts = (point_t *) malloc (size * sizeof (point_t));
     res.n = 0;
     if (res.pts == NULL)
    {
       no_memory ();
    }
     orig = &line.a;
```

```
    prev_side = cmp (&pb, &ppoly.ppts[ppoly.n - 1]);
    for (i = 0; i < ppoly.n; i++)
   {
      side = cmp (&pb, &ppoly.ppts[i]);
        //   fprintf (stderr, "%11g %11g : Side =%d prev = %d\n", ppoly.ppts[i]->x, ppoly.ppts[i]->y, side, prev_side);
      if (prev_side != side)
     {
         if (res.n >= size)
        {
           int size2 = (res.n * 3 + 1) / 2;
             if (res.pts = realloc (res.pts, size2 * sizeof (point_t)))
          {
              size = size2;
          }
             else
          {
              no_memory ();
          }
      }
       if (line_segment_intersect (line.a, line.b, *ppoly.ppts[i - 1], *ppoly.ppts[i], &res.pts[res.n]))
      {
          if (res.n == 0 || distance_rec (res.pts[res.n], res.pts[res.n - 1]) > EPS)
          res.n++;
      }
   }
    prev_side = side;
  }

  return res;
}

/*
   Find intersection points of convex_hull of given points array and a line
   Result is represented as poly_t.
   inters = convex_intersect(pts, n, line);
      inters.n — number of intersection points; could be 0, 1 or 2
      inters.pts — array of inters.n intersection points
      Call free_poly(inters) at the end;
*/
poly_t
convex_intersect (point_t * pts, int n, line_t line)
{
  ppoly_t hull = convex_hull (pts, n);
  poly_t inters = line_poly_intersect (line, hull);
  free_ppoly (hull);
  return inters;
}

int
main (int argc, char *argv[])
{
  int n, i;
  point_t *pts;
  ppoly_t hull;
  poly_t inters;
  line_t line;
  fprintf (stderr,
"Usage: ./convex\n   Read from STDIN points and finds convex hull.\n   Then reads coordinates of two points (point A and point B)


  if (scanf ("%d", &n) != 1 || n < 0 || n > 100000000)
  {
      fprintf (stderr, "Error:  First line should contain number of points\n Now n=%d\n", n);
      exit (1);
  }
  i = 0;
  pts = (point_t *) malloc (n * sizeof (point_t));
  if (pts == NULL)
    no_memory ();

  while (i < n && scanf ("%lf%lf", &pts[i].x, &pts[i].y) == 2) i++;

  if (i < n)
  {
      fprintf (stderr, "Warning: wrong number of point in the input.\nDeclared %d points, read only: %d points", n, i);
      n = i;
  }

  hull = convex_hull (pts, n);

  printf ("Found convex hull consisting of %d points\n", hull.n);
  for (i = 0; i < hull.n; i++)
     printf ("%11g %11g\n", hull.ppts[i]->x, hull.ppts[i]->y);

  printf ("Enter two points (4 numbers: A.x A.y B.x B.y  ) of the line\n");
  if( scanf ("%lf%lf%lf%lf", &line.a.x, &line.a.y, &line.b.x, &line.b.y) != 4 )
  {
      fprintf(stderr, "Cant read four real numbers. Stop\n");
      exit(3);
  }

  inters = line_poly_intersect (line, hull);
  free_ppoly (hull);


  printf ("Found %d intersections\n", inters.n);
  for (i = 0; i < inters.n; i++)
      printf ("%11g %11g\n", inters.pts[i].x, inters.pts[i].y);
  free_poly (inters);
  free (pts);
  return 0;
}
```

Старая версия :

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


#define EPS 1E-10
#define N 1000

int n;          //   число точек всего
double x[N][2]; // массив с координатами точек

/*
 текущий центр координат (для вычисления векторного произведения в функции cmp)
*/
double *xt;

double xtstatic[2];


/*
  Находит знак векторного произведения векторов, проведеденных из
  точки xt в точки a и b
*/
int
cmp (const void *a, const void *b)
{
  double *p1, *p2;
  double s;
  p1 = (double *) a;
  p2 = (double *) b;

  s = (p1[0] - xt[0]) * (p2[1] - xt[1]) - (p2[0] - xt[0]) * (p1[1] - xt[1]);
  if (fabs (s) < EPS)
    {
      s =
   (p1[0] - xt[0]) * (p1[0] - xt[0]) + (p1[1] - xt[1]) * (p1[1] - xt[1]);
      s -=
   (p2[0] - xt[0]) * (p2[0] - xt[0]) + (p2[1] - xt[1]) * (p2[1] - xt[1]);
      if (abs (s) < EPS)
   return 0;
      if (s < 0)
   return 1;
      return -1;
    };
  if (s < 0)
    return 1;
  return -1;
}


/*
  Для точек из глобального массива x находит выпуклую оболочку и кладёт
  индексы точек выпуклой оболочки в массив stack
*/
int
convexhull (int *stack)
{
  int i_ymin; //  индек точки с минимальной координатой y
  int sp;     //  индекс последнего элемента в стеке; в конце
              // — число точек в выпуклой оболочке

  for (i = 0; i < n; i++){
    if (x[i][1] < x[i_ymin][1]) i_ymin = i;
  }

  xt = xtstatic;
  xt[0] = x[i_ymin][0];
  xt[1] = x[i_ymin][1];
  x[i_ymin][0] = x[0][0];
  x[i_ymin][1] = x[0][1];
  x[0][0] = xt[0];
  x[0][1] = xt[1];

  qsort (&x[1], n, 2 * sizeof (double), cmp);

  stack[0] = 0;
  stack[1] = 1;
  stack[2] = 2;

  sp = 2;
  i = 2;
  while (i < n + 1)
   {
      xt = x[stack[sp - 1]];
      if (cmp (&x[stack[sp]][0], &x[(i + 1) % n][0]) < 0)
      {
        stack[++sp] = (++i) % n;
      } else {
        sp--;
      }
    }
   return sp;
}

int
main ()
{
  int stack[N];
// Scan points from std input
  scanf ("%d", &n);
  for (i = 0; i < n; i++) scanf ("%lf %lf", &x[i][0], &x[i][1]);
```

```
// Place id's of convexhull points into stack
  int length = convexhull (stack);
  return 0;
}
```

-- ArtemVoroztsov - 17 Mar 2004