

Лекции 5-6. Построение выпуклых оболочек

1. Выпуклая оболочка

Пусть на плоскости заданы k различных точек p_1, p_2, \dots, p_k . Множество точек

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \dots + \alpha_k p_k, \\ (\alpha_i \in \mathbb{R}, \alpha_i \geq 0, \alpha_1 + \alpha_2 + \dots + \alpha_k = 1)$$

называется *выпуклым множеством*, порожденным точками p_1, p_2, \dots, p_k , а точка p называется *выпуклой комбинацией* точек p_1, p_2, \dots, p_k .

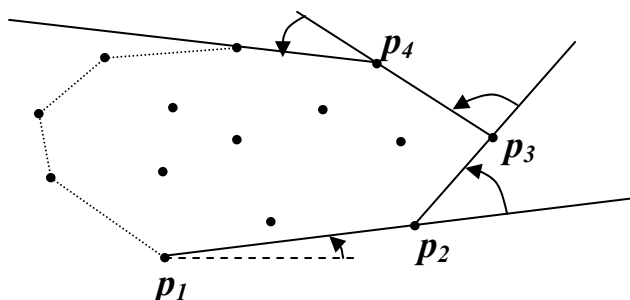
Выпуклой оболочкой множества точек L на плоскости (обозначается $\text{conv}(L)$) называется наименьшее выпуклое множество, содержащее L .

В дальнейшем мы рассматриваем лишь конечные множества L . Выпуклая оболочка конечного множества точек на плоскости является выпуклым многоугольником и наоборот, каждый выпуклый многоугольник является выпуклой оболочкой некоторого множества точек.

Задача построения выпуклой оболочки ставится следующим образом. Для множества L из n точек требуется построить выпуклую оболочку $\text{conv}(L)$ в виде полного описания границы. Это описание границы представляет собой упорядоченной подмножество точек из L , называемых *граничными точками*. Остальные точки из L , не являющиеся граничными, называются *внутренними*.

2. Метод Джарвиса (заворачивание подарка)

Предположим, что найдена наименьшая в лексикографическом порядке точка p_1 заданного множества L . Это значит, что p_1 имеет минимальную ординату (т.е. координату y) среди точек из L . Эта точка заведомо граничная, т.е. является вершиной выпуклой оболочки. Найдем теперь следующую за ней вершину p_2 выпуклой оболочки. Точка p_2 - это точка, имеющая наименьший положительный полярный угол относительно точки p_1 как начала координат. Далее следующая граничная точка p_3 выбирается таким образом, чтобы вектор $p_1 p_3$ имел наименьший положительный угол относительно вектора $p_1 p_2$. Аналогично ищутся остальные граничные точки: точка p_{k+1} выбирается так, чтобы вектор $p_k p_{k+1}$ имел наименьший положительный угол относительно вектора $p_{k-1} p_k$ для $k \geq 2$.



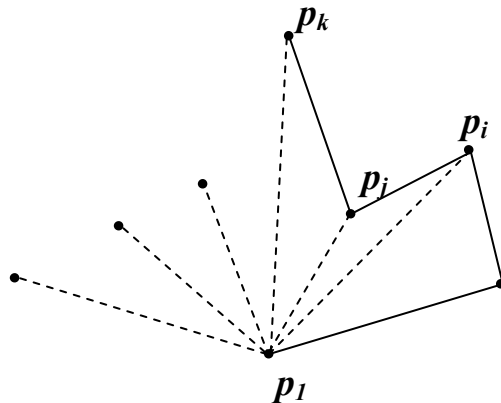
Алгоритм Джарвиса обходит кругом выпуклую оболочку, порождая в нужном порядке последовательность крайних точек по одной на каждом шаге. Так как все точки из L могут оказаться граничными, а алгоритм Джарвиса затрачивает на нахождение одной граничной точки линейное время, то общее время выполнения алгоритма в худшем случае составляет $O(n^2)$. Однако, если в действительности число вершин выпуклой оболочки равно h , то время выполнения алгоритма Джарвиса будет $O(nh)$, т.е. он очень эффективен, если h мало.

Алгоритм построения выпуклой оболочки методом Джарвиса

```
TYPE
  SetOfPoints = ARRAY [1..n] OF Point;
PROCEDURE ConvexHullJarvis(L: SetOfPoints; P:Head);
VAR v,w,p1,p2: Point;
    i: INTEGER;
BEGIN
  (* Найти в L точку p1 с минимальной координатой p1.y и
  поместить ее в список вершин P *)
  p1:=L[1];
  FOR i:=2 TO n DO
    IF (L[i].y<p1.y) THEN p1:=L[i];
  p1.into(P);
  (* Найти в L точку p2 такую, что все точки из L лежат
  не правее вектора p1,p2 и поместить p2 в список P *)
  p2:=L[1]; IF p2=p1 THEN p2:=L[2];
  FOR i:=1 TO n DO
    IF (L[i] лежит правее вектора (p1,p2)) THEN
      p2:=L[i];
  p2.into(P);
  (* Заворачивание подарка *)
  v:=p2;
  WHILE v<>p1 DO
    BEGIN w:=p1;
      FOR i:=1 TO n DO
        IF (L[i]<>v) AND
          (L[i] лежит правее вектора (v,w)) THEN
          w:=L[i];
      v:=w;
      v.into(P);
    END;
  END;
```

3. Метод Грэхема

Пусть найдена наименьшая в лексикографическом порядке точка p_1 заданного множества L , что требует времени $O(n)$. Упорядочим лексикографически все остальные точки в соответствии с полярными углами



векторов $p_i p_j$ и длинами этих векторов. Это значит, что на множестве точек L вводится отношение порядка «предшествует». Считается, что точка p_i предшествует точке p_j ($i, j \geq 2$), если точка p_j лежит левее либо впереди вектора $p_i p_j$. Классификация положений точки относительно вектора представлена в следующем разделе.

Представив упорядоченные точки в виде двусвязного списка, получим ситуацию, представленную на рисунке.

Если точка не является вершиной выпуклой оболочки (на рисунке это точка p_j , то она является внутренней точкой для некоторого треугольника $p_i p_j p_k$, где p_i и p_k - последовательные вершины выпуклой оболочки. Суть метода Грэхема состоит в однократном просмотре упорядоченной последовательности точек, в процессе которого удаляются внутренние точки. Оставшиеся точки являются вершинами выпуклой оболочки, представленными в требуемом порядке. При просмотре тройки последовательных точек многократно проверяются в порядке обхода против часовой стрелки с целью определить, образуют или нет они угол, больший или равный π . Если внутренний угол $p_i p_j p_k$ меньше π (т.е. точка p_k лежит правее вектора $p_i p_j$), то говорят, что точки $p_i p_j p_k$ образуют «правый поворот», иначе - «левый поворот». Таким образом, если при проверке выясняется, что тройка $p_i p_j p_k$ образует правый поворот, это значит, что точка p_j не может быть вершиной выпуклой оболочки, так как она является внутренней точкой треугольника $p_i p_j p_k$. Следовательно, в зависимости от результата проверки угла, образованного текущей тройкой точек, возможны два варианта продолжения просмотра:

1. Тройка $p_i p_j p_k$ образует правый поворот. Удалить среднюю точку p_j тройки из списка вершин и проверить вновь образовавшуюся текущую тройку $p_i p_k p_{k+1}$.
2. Тройка $p_i p_j p_k$ образует левый поворот. Продолжить просмотр, перейдя к проверке тройки $p_j p_k p_{k+1}$.

Просмотр завершится, когда, обойдя все вершины, мы вновь придем в вершину p_1 . Заметим, что эта вершина никогда не удаляется, так как она является граничной точкой множества L .

Оценим требуемое время для решения задачи. Проверка угла выполняется за фиксированное время. После каждой проверки происходит либо продвижение на одну точку (случай 2), либо удаление одной точки (случай 1). Так как множество L содержит n точек, то возможно не более n продвижений и не более n удалений. Следовательно, время обхода составляет $O(n)$. С учетом того, что сортировка массива точек заняла время $O(n \log n)$, получаем, что общее время работы алгоритма составляет именно $O(n \log n)$.

Алгоритм построения выпуклой оболочки методом Грэхема

```

PROCEDURE ConvexHullGraham(L: SetOfPoints; P: Polygon);
VAR u, v, w, p1: Point;
BEGIN
  1) Найти в L точку p1 с минимальной координатой p1.y и
  поместить ее в список вершин P
  2) Упорядочить точки из L в списке P лексикографически
  по полярному углу и расстоянию от p1
  3) Обход
    u:=p1; v:=u.sucring; w:=v.sucring;
    WHILE (v<>p1) DO
      BEGIN

```

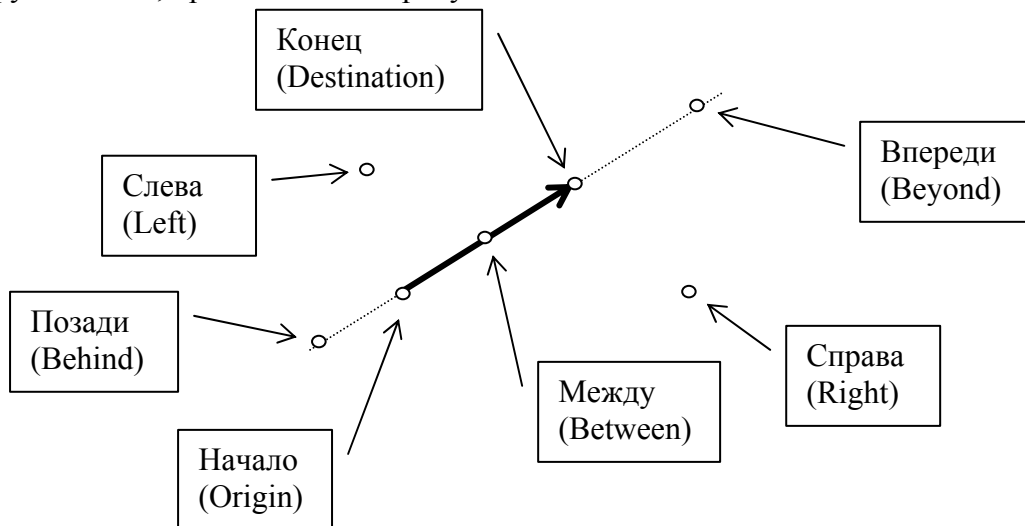
```

    IF (w лежит справа от вектора (u,v)) THEN v.out
    ELSE u:=u.suc;
    v:=u.sucring; w:=v.sucring;
  END;
END;

```

4. Относительное положение точки и вектора

Классификация взаиморасположения точки и вектора, задаваемого парой других точек, представлена на рисунке.



Структуры данных, описывающие объекты, используемые в алгоритмах вычислительной геометрии, объединим в модуле Geometry.

```

UNIT Geometry;
INTERFACE
USES Simset;
TYPE
  Position = (Left, Right, Beyond, Behind, Between,
              Origin, Destination);
  Point= CLASS(Link)
    x,y: DOUBLE;
    FUNCTION Classify(P1,P2: Point): Position;
  END;
IMPLEMENTATION
FUNCTION Point.Classify(P1,P2: Point): Position;
VAR ax,ay,bx,by,s,c,ma,mb: DOUBLE;
BEGIN
  ax:=P2.x-P1.x; ay:=P2.y-P1.y;
  bx:=x-P1.x;   by:=y-P1.y;
  s:=ax*by - ay*bx; (* векторное произведение *)
  IF s>0 THEN Result:=Left
  ELSE IF s<0 THEN Result:=Right
  ELSE
    BEGIN
      c:=ax*ay + bx*by; (* скалярное произведение *)
      IF c<0 THEN Result:=Behind
      ELSE IF c=0 THEN Result:=Origin
    END
  END
END;

```

```

ELSE
BEGIN
  ma:=ax*ax+ay*ay; (* квадрат длины вектора *)
  mb:=bx*bx+by*by; (* квадрат длины вектора *)
  IF ma<mb THEN Result:=Beyond
  ELSE IF ma=mb THEN Result:=Destination
  ELSE Result:=Between;
END;
END;
END;
END.

```

5. Метод редукции для оценки сложности задачи

Иногда удается установить сложность одной задачи по известной сложности другой задачи, используя метод *преобразования задач* или *редукцию*.

Предположим, что имеются две задачи **A** и **B**, которые связаны так, что задачу **A** можно решить следующим образом:

1. Исходные данные к задаче **A** преобразуются в соответствующие исходные данные для задачи **B**.
2. Решается задача **B**.
3. Результат решения задачи **B** преобразуется в правильное решение задачи **A**.

В этом случае говорят, что задача **A** *преобразуема* к задаче **B** или имеет место *редукция A к B*. Если шаги 1 и 3 такого преобразования выполняются за время $O(\tau(n))$, где n - размер входа задачи **A**, то **A** - $(\tau(n))$ -преобразуема в **B**.

Отношение редукции не симметричное отношение (т.е. если **A** преобразуема к **B**, то это не означает, вообще говоря, что **B** преобразуема к **A**), но в частном случае, когда **A** и **B** взаимно преобразуемы, они называются *эквивалентными*.

Следующие два очевидных утверждения демонстрируют возможности метода преобразования в предположении, что это преобразование сохраняет порядок размера задач.

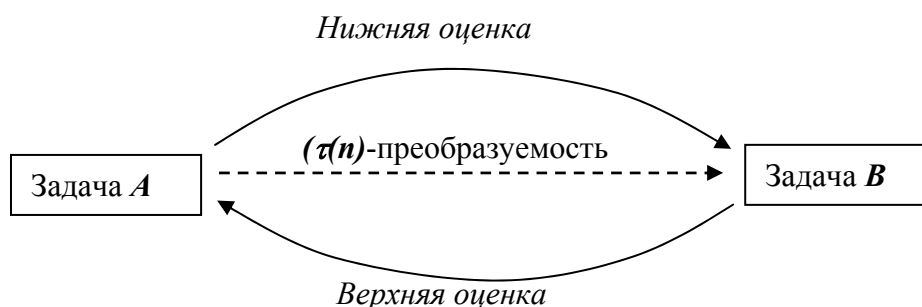
Теорема (Нижние оценки методом редукции).

Если известно, что задача **A** требует $T(n)$ времени и **A** $(\tau(n))$ -преобразуема в **B**, то задача **B** требует не менее $T(n) - O(\tau(n))$ времени.

Теорема (Верхние оценки методом редукции).

Если задачу **B** можно решить за время $T(n)$ и задача **A** $(\tau(n))$ -преобразуема в **B**, то **A** можно решить за время, не превышающее $T(n) + O(\tau(n))$.

Эти теоремы иллюстрируются следующей диаграммой, на которой показано, как верхняя и нижняя оценки переносятся от одной задачи к другой. Перенос оценок справедлив, когда $\tau(n) \in O(T(n))$, т.е. когда время преобразования не превосходит времени вычисления.



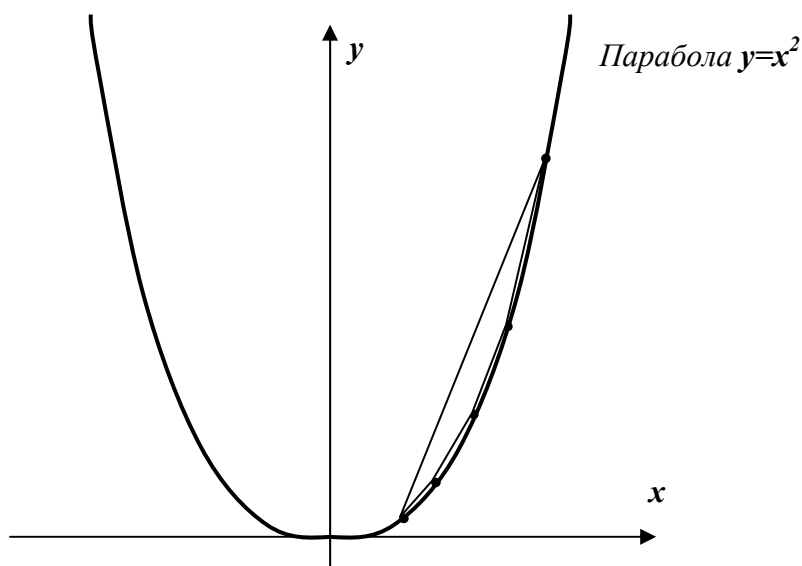
6. Нижняя оценка для алгоритмов выпуклой оболочки

Используя метод редукции, можно получить нижнюю оценку для алгоритмов построения выпуклой оболочки конечного множества n точек. Для этого воспользуемся тем фактом, что известна нижняя оценка $\Omega(n \log n)$ для задачи сортировки массива из n чисел.

Теорема. Задача сортировки преобразуема за линейное время к задаче построения выпуклой оболочки, и, следовательно, для нахождения выпуклой оболочки n точек на плоскости требуется время $\Omega(n \log n)$.

Доказательство. Продемонстрируем процедуру преобразования задачи сортировки к задаче построения выпуклой оболочки.

Пусть заданы n положительных действительных чисел x_1, x_2, \dots, x_n . Необходимо показать, как можно использовать алгоритм построения выпуклой оболочки для сортировки этих чисел, чтобы при этом дополнительные затраты времени линейно зависели от количества чисел. Поставим в соответствие числу x_i точку на плоскости с координатами (x_i, y_i) , где $y_i = x_i^2$, и присвоим точке номер i . Все эти точки лежат на параболе $y = x^2$. Выпуклая оболочка этого множества



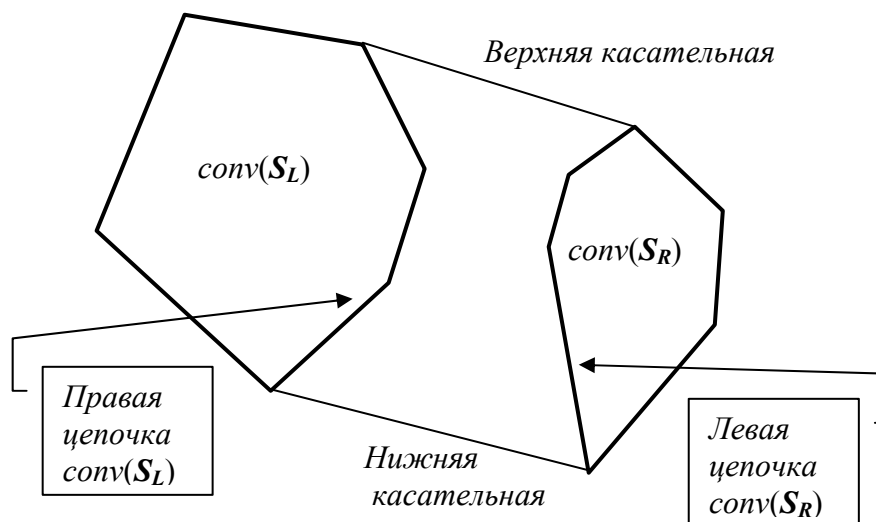
точек, представленная в стандартном виде, будет состоять из списка точек множества, упорядоченных по значению абсциссы. Один просмотр этого списка позволяет найти самую левую точку и еще один просмотр позволяет прочитать в нужном направлении значения x_i .

7. Слияние выпуклых оболочек

Рассмотрим алгоритм «разделяй и властвуй» для вычисления выпуклой оболочки конечного множества точек S . Идея заключается в разделении массива вертикальной прямой линией на два примерно равных по величине подмассива S_L и S_R , рекурсивном построении выпуклых оболочек $conv(S_L)$ и $conv(S_R)$ и формировании оболочки $conv(S)$ на основе объединения оболочек этих подмассивов. На нижнем уровне рекурсии подмассивы будут состоять из одной точки, которая сама является своей выпуклой оболочкой, поэтому построение их выпуклых оболочек становится тривиальной процедурой, выполняемой за фиксированное время. Аналогично тому, как это было сделано в задаче сортировки массива метода слияния, для получения эффективного алгоритма, реализующего оценку $\Theta(n \log n)$, нужно построить процедуру

слияния двух выпуклых оболочек за время, пропорциональное суммарному количеству элементов в S_L и S_R .

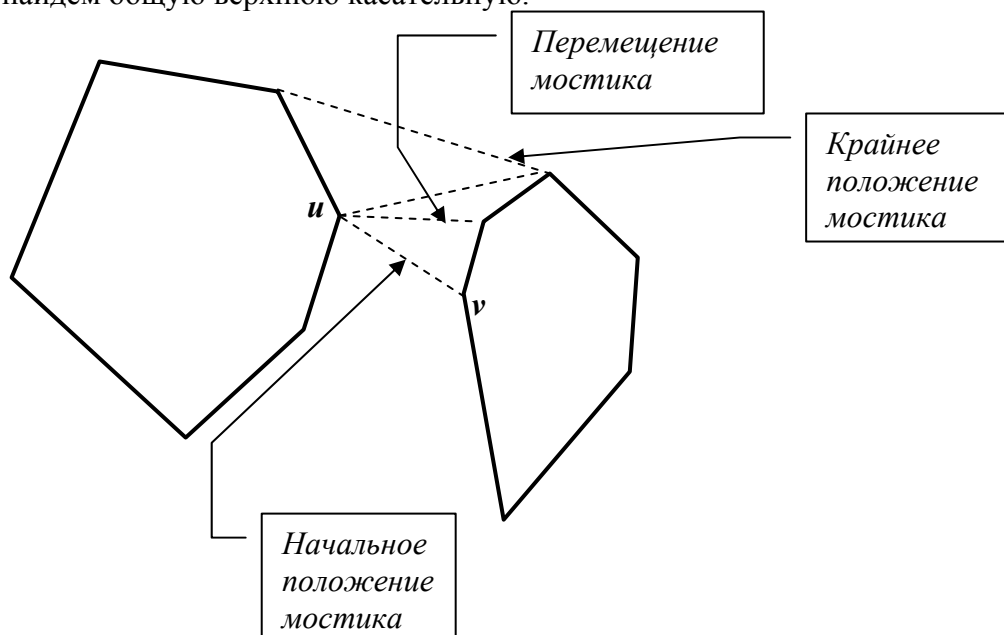
Поскольку множества S_L и S_R разделены вертикальной линией, их выпуклые оболочки $conv(S_L)$ и $conv(S_R)$ представляют собой непересекающиеся



выпуклые многоугольники. Следовательно, для построения общей выпуклой оболочки $conv(S)$ этих многоугольников достаточно найти их общие касательные - верхнюю и нижнюю, а затем отбросить правую цепочку в многоугольнике $conv(S_L)$ и левую цепочку в многоугольнике $conv(S_R)$.

Это слияние составляет основную часть данного алгоритма и сводится к поиску верхней и нижней касательных.

Пусть левый и правый многоугольники представлены списками вершин P_L и P_R . Поиск общей касательной начнем с мостика (u, v) , соединяющего самую правую вершину u в P_L с самой левой вершиной v в P_R . Далее будем последовательно перемещать эти вершины вдоль границ выпуклых многоугольников против часовой стрелки в P_L и по часовой стрелке в P_R до тех пор, пока существуют вершины, лежащие слева от вектора (u, v) . Таким образом мы найдем общую верхнюю касательную.



Аналогично, нижняя касательная ищется из того же самого начального положения соединяющего мостика (u,v) . Но перемещение его теперь осуществляется по часовой стрелке в P_L и против часовой стрелки в P_R . Поиск нижней касательной завершается, когда мостик окажется в положении, при котором не существует вершин, лежащих справа от вектора (u,v) .

Алгоритм построения выпуклой оболочки методом слияния

```

PROCEDURE ConvexHullMerge(L: SetOfPoints; VAR P:Head);
(* Это процедура верхнего уровня - формирует выпуклый
   многоугольник с вершинами в списке P для множества
   точек из массива L *)
VAR v,w,p1,p2: Point;
    i: INTEGER;
BEGIN
(* Лексикографически упорядочиваем точки в массиве L
   по возрастанию координат x,y *)
MergeSort(L,0,n);
(* Строим выпуклую оболочку для упорядоченного
   массива *)
P:=ConvexPolygon(L,0,n);
END;

FUNCTION ConvexPolygon(L: SetOfPoints;
                      n1,n2: INTEGER): Head;
(* Для точек упорядоченного массива L точек с L[n1]
   по L[n2] строится выпуклая оболочка.
   Возвращается список вершин оболочки *)
VAR P,PL,PR: Head;
    M: INTEGER;
BEGIN
P:=Head.Create;
IF n1=n2 THEN (* Оболочка для 1 точки *)
L[n1].into(P)
ELSE
BEGIN
m:=(n1+n2) DIV 2; (* середина массива *)
PL:=ConvexPolygon(L,n1,m);
(* оболочка левого подмножества *)
PR:=ConvexPolygon(L,m+1,n2);
(* оболочка правого подмножества *)
MergePolygons(PL,PR,P); (* объединение оболочек *)
PL.Destroy;
PR.Destroy;
END;
Result:=P;
END;

FUNCTION Less(p1,p2: Point): BOOLEAN;
(* отношение лексикографического порядка для точек *)
BEGIN
Result:=(p1.x<p2.x) OR (p1.x=p2.x) AND (p1.y<p2.y);

```



```

END;

PROCEDURE MergePolygons(PL, PR, P);
(* Объединение выпуклых полигонов, разделимых по
вертикали *)
VAR u, v, u0, v0, u1, v1, u2, v2, t, s: Point;
    B: BOOLEAN;
BEGIN
(* ищем правую точку в левом подмножестве *)
    u0:=PL.first AS Point;
    t:=u0.suc AS Point;
    WHILE t<>NIL DO
    BEGIN (* поиск максимальной точки в PL *)
        IF Less(u0, t) THEN u0:=t;
        t:=t.suc AS Point;
    END;
(* ищем левую точку в правом подмножестве *)
    v0:=PR.first AS Point;
    t:=v0.suc AS Point;
    WHILE t<>NIL DO
    BEGIN (* поиск минимальной точки в PR *)
        IF Less(t, v0) THEN v0:=t;
        t:=t.suc AS Point;
    END;
    (* u0, v0 - начальное положение мостика *)
(* ищем верхнюю касательную u1, v1 *)
    u1:=u0; v1:=v0;
    B:=TRUE;
    WHILE B DO
    BEGIN B:=FALSE;
        t:=u1.sucring AS Point;
        IF t.Classify(u1, v1)=Left THEN
            (* сдвиг левого конца мостика *)
            BEGIN u1:=t; B:=TRUE END;
        t:=v1.predring AS Point;
        IF t.Classify(u1, v1)=Left THEN
            (* сдвиг правого конца мостика *)
            BEGIN v1:=t; B:=TRUE END;
        END;
    END;
(* ищем нижнюю касательную u2, v2 *)
    u2:=u0; v2:=v0;
    B:=TRUE;
    WHILE B DO
    BEGIN B:=FALSE;
        t:=u1.predring AS Point;
        IF t.Classify(u2, v2)=Right THEN
            (* сдвиг левого конца мостика *)
            BEGIN u2:=t; B:=TRUE END;
        t:=v1.sucring AS Point;
        IF t.Classify(u2, v2)=Left THEN

```

```

        (* сдвиг правого конца мостика *)
        BEGIN v2:=t; B:=TRUE END;
    END;
(* формируем оболочку P *)
    t:=u1;
    WHILE t<>NIL DO
    BEGIN
        IF t=u2 THEN s:=v2
        ELSE IF t=v1 THEN s:=NIL
        ELSE s:=t.sucring AS Point;
        t.into(P);
        t:=s;
    END;
    PL.clear;
    PR.Clear;
END;

```