Рубан М.М.

Асистент кафедри алгебри фізико-математичного факультету СДПУ

Алгоритми побудови опуклої оболонки

Останнім часом геометричні розрахунки стають все більш важливими в таких галузях, як комп'ютерна графіка, робототехніка та автоматизована розробка, тому форма об'єкта – одна з основних його властивостей. Однією з більш поширеною задач обчислювальної геометрії є побудова опуклої оболонки. При великих масивах даних розв'язок задачі може займати дуже багато часу, тому потрібно знати та вміти використовувати різноманітні підходи до знаходження опуклої оболонки. Розглянемо деякі з них.

Опуклість багатокутника з вершинами $P_1,\ldots,P_n$, які перераховані в порядку його обходу, легко перевірити, якщо обчислити знаки косих добутків $\left[\overrightarrow{P_iP_{i+1}},\overrightarrow{P_{i+1}P_{i+2}}\right]$, $i=\overline{1,n}$ (де P_{n-1} є P_1, а P_{n-2} – P_2). Якщо ми знаємо напрямок обходу, то знак косих добутків для опуклого багатокутника визначений: при обході за годинниковою стрілкою всі косі добутки від'ємні, а проти годинникової стрілки – додатні.

Опуклою оболонкою деякої заданої множини точок називається перетин всіх опуклих множин, які містять задану множину. Для кінцевої множини точок опуклою оболонкою завжди буде опуклий багатокутник, всі вершини якого є точками вихідної множини.

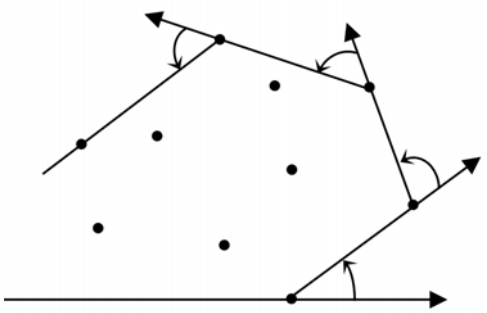

Рис. 1. Алгоритм побудови опуклої оболонки за методом Джарвиса

Завдання полягає в тому, щоб для заданої кінцевої множини точок знайти вершини опуклої оболонки цієї множини. Будемо перераховувати вершини в порядку обходу проти годинникової стрілки. Для ефективного рішення цієї задачі існує кілька різних алгоритмів [1, 2]. Приведемо найбільш просту реалізацію одного з них – алгоритму Джарвиса. Цей алгоритм іноді називають «загортанням подарунка» (рис. 1).

Перерахування точок шуканої границі опуклого багатокутника почнемо із правої нижньої точки P_1 яка завідомо належить границі опуклої оболонки. Позначимо її координати (x_1, y_1). Наступною при зазначеному порядку обходу буде точка $P_2(x_2, y_2)$. Легко бачити, що вона, володіє такою властивістю, що всі

інші точки лежать «ліворуч» від вектора $\overrightarrow{P_1P_2}$, тобто орієнтований кут між векторами $\overrightarrow{P_1P_2}$ та $\overrightarrow{P_1K}$ невід'ємний для будь-якої іншої точки K нашої множини. Для знаходження потрібної точки P_2 перевіряємо виконання умови $\left[\overrightarrow{P_1P_2}, \overrightarrow{P_1K}\right] > 0$ з усіма точками K. Якщо точок, які задовольняють цій умові, декілька, то вершиною шуканого багатокутника стане та з них, для якої довжина вектора $\overrightarrow{P_1P_2} = (x_2 - x_1, y_2 - y_1)$ максимальна.

Будемо робити так само й далі. Припустимо, що вже знайдена i-та вершина $P_i(x_i, y_i)$ опуклої оболонки. Для наступної точки $P_{i+1}(x_{i+1}, y_{i+1})$ косі добутки невід'ємні для всіх точок K. Якщо таких точок декілька, то вибираємо ту, для якої вектор $\overrightarrow{P_iP_{i+1}}$ має найбільшу довжину. Безпосередньо пошук такої точки можна здійснювати так. Спочатку ми можемо вважати наступною, $(i+1)$-ю, будь-яку точку. Потім обчислюємо значення $\left[\overrightarrow{P_iP_{i+1}}, \overrightarrow{P_{i+1}K}\right]$, розглядаючи в якості K всі інші точки. Якщо для однієї з них зазначене вираження менше нуля, вважаємо наступною її та продовжуємо перевірку інших точок (аналогічно алгоритму пошуку мінімального елемента в масиві). Якщо ж значення виразу дорівнює нулю, то порівнюємо квадрати довжин векторів. У результаті за $O(N)$ операцій чергова вершина опуклої оболонки буде знайдена. Продовжуючи цю процедуру, ми рано або пізно повернемося до точки P_1. Це буде означати, що опукла оболонка побудована.

При розв'язку даної задачі у випадку цілочисельних координат ми повністю можемо уникнути застосування дійсної арифметики, а отже, позбутися від втрати точності обчислень. В противному випадку в рішення можуть бути включені «зайві» точки, близькі до границі опуклої оболонки, або не враховані деякі з «потрібних» точок. Складність даного алгоритму $O(mN)$, де m – кількість точок в опуклій оболонці, в найгіршому випадку рівне N.

Наведемо одну з можливих реалізацій рішення даної задачі алгоритмом Джарвиса. Множина вихідних точок знаходиться в масиві a, всі точки, які належать опуклій оболонці, будемо записувати в масив b.

```
type dot = record x, y: integer; end;
var a, b: array[1..100] of dot;
  min, m, i, j, k, n: integer;
function vec(a1,a2,b1,b2: dot): integer;
{косий добуток векторів a1a2 та b1b2}
begin
  vec := (a2.x - a1.x)*(b2.y - b1.y) - (b2.x - b1.x)*(a2.y - a1.y)
end;
function dist2(a1,a2: dot): integer;
{квадрат довжини вектора a1a2}
begin
  dist2 := sqr(a2.x - a1.x) + sqr(a2.y - a1.y)
end;
begin
  readln(n);{кількість точок}
```

```
 for i := 1 to n do
   read(a[i].x, a[i].y);
{пошук правої нижньої точки}
 m := 1;
 for i := 2 to n do
   if a[i].y < a[m].y then m := i else
   if (a[i].y = a[m].y) and
      (a[i].x > a[m].x) then m := i;
 b[1] := a[m];  a[m] := a[1];  a[1] := b[1];
 k := 1;
 min := 2;
 repeat   {пошук наступної вершини опуклої оболонки}
   for j := 2 to n do
    if (vec(b[k],a[min],b[k],a[j]) < 0) or
      ((vec(b[k],a[min],b[k],a[j]) = 0) and
      (dist2(b[k],a[min]) < dist2(b[k],a[j])))
    then min := j;
   k := k + 1;
   b[k] := a[min];
   min := 1;
 until (b[k].x=b[1].x)  and (b[k].y=b[1].y); {поки не отримаємо замкнену ломану}
 for j := 1 to k - 1 do {вивід результату}
   writeln(b[j].x,' ',b[j].y)
end.
```

Існує інший алгоритм рішення цієї задачі (алгоритм Грехема) з обчислювальною складністю $O(N\log N)$, він базується на попередньому сортуванні точок вихідної множини за значенням кута в полярній системі координат із центром в одній із точок опуклої оболонки. Тобто найбільш трудомісткою задачею виявляється саме сортування вихідних точок. Сортування точок можна робити за знаком косого добутку $\left[\overrightarrow{P_1P_i},\overrightarrow{P_1P_{i+1}}\right]$, де P_1 – будь-яка вершина опуклої оболонки (наприклад, права нижня точка). У відсортованому масиві точок всі зазначені добутки повинні бути невід'ємні. Точки з рівними кутами ($\left[\overrightarrow{P_1P_i},\overrightarrow{P_1P_{i+1}}\right]=0$) розташовуються в порядку збільшення довжин відповідних векторів $\overrightarrow{P_1P_i}$ (рис. 2).

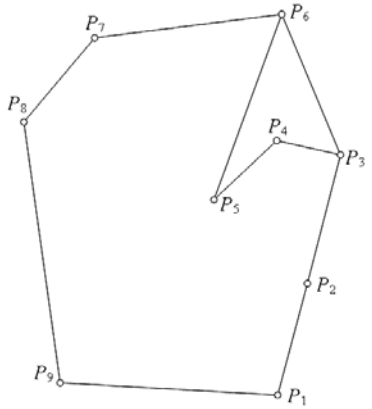

Рис. 2. Вершини опуклої оболонки множини точок

Далі в перегляді за Грехемом використовується стек, у якому зберігаються точки, які є кандидатами в опуклу оболонку. Спочатку в стек потрапляє перша з відсортованих точок. Потім – сусідня з нею вершина опуклої оболонки. Якщо на першому із променів точок декілька, то це точка цього променя P_i, яка найбільш віддалена від P_1. Нехай у вершині стека перебуває точка P_k. Розглянемо наступну в порядку збільшення полярного кута точку вихідної множини P_i. Поки ділянка ламаної $P_{k-1}P_kP_i$ не стане опуклою, зі стека видаляється чергова точка P_k. Далі P_i переходить до стеку. Під час закінчення перегляду всіх точок у стеці будуть перебувати всі вершини опуклої оболонки. Так як будь-яка точка додається в стек рівно один раз, та й видаляється вона з нього не більше одного разу, тому час перегляду становить $O(N)$. Наведемо реалізацію саме такого перегляду. Для наочності сортування проведемо «бульбашковим» алгоритмом. У якості стека використається масив b. Описи змінних та функцій збігаються з наведеними вище.

```
type dot = record x, y: integer; end;
var a, b: array[1..100] of dot;
  min, m, i, j, k, n: integer;
function vec(a1,a2,b1,b2: dot): integer;
{косий добуток векторів a1a2 та b1b2}
begin
  vec := (a2.x - a1.x)*(b2.y - b1.y) - (b2.x - b1.x)*(a2.y - a1.y)
end;
function dist2(a1,a2: dot): integer;
{квадрат довжини вектора a1a2}
begin
  dist2 := sqr(a2.x - a1.x) + sqr(a2.y - a1.y)
end;

begin
  readln(n);
  for i := 1 to n do
    read(a[i].x, a[i].y);
 {пошук правої нижньої точки}
  m := 1;
  for i := 2 to n do
    if a[i].y < a[m].y then m := i else
    if (a[i].y = a[m].y) and
       (a[i].x > a[m].x) then m := i;
 {запишемо її масив оболонки }
  b[1] := a[m];
  a[m] := a[1];
  a[1] := b[1];
 {інші точки сортуємо по значенню полярного кута}
  for i := n downto 3 do
    for j := 2 to i - 1 do
     if (vec(a[1],a[j],a[1],a[j + 1]) < 0) or
       ((vec(a[1],a[j],a[1],a[j + 1]) = 0 ) and
      (dist2(a[1],a[j]) > dist2(a[1],a[j + 1])))
        then
      begin
        b[n] := a[j];
```

```
        a[j] := a[j + 1];
        a[j + 1] := b[n]
      end;
 {пошук другої вершини оболонки}
 i := 2;
 while vec(a[1],a[i + 1],a[1],a[2]) = 0 do
   i := i + 1;
 b[2] := a[i];
 b[3] := a[i + 1];
 k := 3;
 for i := i + 2 to n do
  begin
   {перевірка опуклості}
    while vec(b[k-1],b[k],b[k],a[i]) <= 0 do
      k := k-1;{видаляємо точку зі стека}
    k := k + 1;
    b[k] := a[i] {добавляємо точку в стек}
  end;
 for j := 1 to k do {вивід результату}
   writeln(b[j].x,' ',b[j].y)
end.
```

Таким чином, було розглянуто алгоритми пошуку опуклої оболонки для множини точок на площині. На них спирається рішення великої кількості складних, зокрема олімпіадних, задач, тому їх розуміння та використання на практиці є дуже важливим.

Література

1. *Андреева Е.В.* Геометрические задачи на олимпиадах по информатике // Информатика № 14/2002.

2. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы. Построение и анализ. М.: МЦНМО, 2000.

3. *Курант Р., Роббинс Г.* Что такое математика. М.: ОГИЗ, государственное изд-во технико-теоретической литературы, 1947.

4. *Окулов С.М.* 100 задач по информатике. Киров: Изд-во ВГПУ, 2000.

5. *Станкевич А.С.* Решение задач I Всероссийской командной олимпиады по программированию. Информатика № 12/2001.

6. *Шикин Е.В., Боресков А.В., Зайцев А.А.* Начала компьютерной графики. М.: Диалог-МИФИ, 1993.