ВСП Миколаївський будівельний фаховий коледж
Київського національного університету будівництва і архітектури

Дослідницька робота:

Побудова опуклої оболонки, опукла оболонка, мінімально опукла оболонка

Виконав:

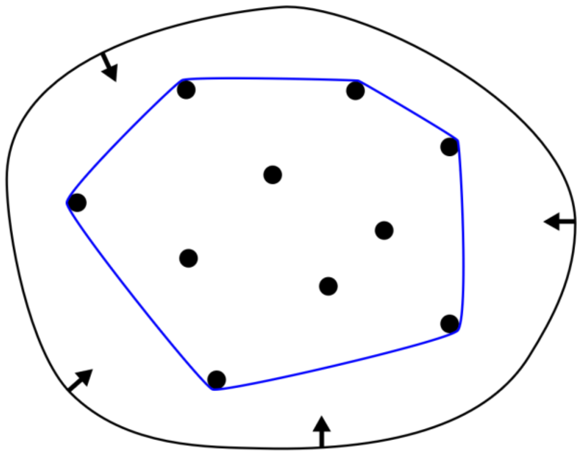
студент групи КН-307
Шепель Р. С.

Викладач: Маламан А. Ф.

ЗМІСТ

1. Опукла оболонка

Опукла оболонка (англ. Convex hull) множини точок X на евклідовій площині або у просторі — це мінімальна опукла множина, що містить X.
В обчислювальній геометрії, прийнято використовувати термін «опукла оболонка» для межі мінімальної опуклої множини, що містить дану не порожню скінченну множину точок на площині. Для скінченної множини точок, опукла оболонка являє собою ламану лінію.



Зображення опуклої оболонки

Опукла оболонка на C++

- Для знаходження точок які знаходяться в опуклій оболонці використав вектор.
- Побудував опуклу оболонку Алгоритмом Jarvis(Джарвіса)

```cpp
// Побудова опуклої оболонки Алгоритмом Jarvis
#include <iostream>
#include <vector>
using namespace std;

struct Point
{
    int x, y;
};

// Знаходження орієнтації впорядкованого триплету (p, q, r).
// Функція повертає наступні значення
// 0 --> p, q та r колінеарні
// 1 --> За голинковою стрілкою
// 2 --> Проти годинкової стрілки
int orientation(Point p, Point q, 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; // колінеарні
    return (val > 0) ? 1 : 2; // За годинковою стрілкою або проти годинкової стрілки
}

// Друкує опуклу оболонку набору з n точок
void convexHull(Point points[], int n)
{
    // Має бути не менше троьох балів
    if (n < 3) return;

    // Ініціалізація результату
    vector<Point> hull;

    // Знаходження крайної лівої точки
    int l = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[l].x)
            l = i;

    // Почніть з крайньої лівої точки, продовжуйте рух проти годинникової стрілки
    // поки знову не досягне початкової точки. Цей цикл працює O(h)
    // разів, де h – кількість точок у результаті або виведенні.
    int p = l, q;
```

```cpp
    int p = l, q;
    do
    {
        // Додати поточну точку до результату
        hull.push_back(points[p]);

        // Пошук точки 'q' такої, що orientation(p, q,
        // x) проти годинникової стрілки для всіх точок 'x'. Ідея
        // слід відстежувати останні відвідані найбільш протилежні годинники-
        // минула точка в q. Якщо будь-яка точка «i» більше проти годинника,
        // минула q, потім оновити q.
        q = (p + 1) % n;
        for (int i = 0; i < n; i++)
        {
            // Якщо і більше проти годинникової стрілки q, тоді
            // оновити q
            if (orientation(points[p], points[i], points[q]) == 2)
                q = i;
        }

        // Тепер q найбільше проти годинникової стрілки відносно p
        // Встановити p як q для наступної ітерації, щоб q додавався
        // result 'hull'
        p = q;

    } while (p != l); // Поки ми не дійшли до першого пункту

    // Виводимо результат
    for (int i = 0; i < hull.size(); i++)
        cout << "(" << hull[i].x << ", "
        << hull[i].y << ")\n";
}

// Перевірка вищезначених функцій
int main()
{
    Point points[] = { {0, 3}, {2, 2}, {1, 1}, {2, 1},
                      {3, 0}, {0, 0}, {3, 3} }; // 7 точок на Площині х та у.
    int n = sizeof(points) / sizeof(points[0]);
    // convexHull(points, n); перевірка виводу точок яких записав
    return 0;
}
```

Программа для вичеслення опуклої оболонки

```
Консоль отладки Micro
(0, 3)
(0, 0)
(3, 0)
(3, 3)

C:\Users\Roman\source
Чтобы автоматически з
томатически закрыть к
Нажмите любую клавишу
```

Протокол роботи програми

2. Властивості опуклої оболонки

- X — опукла множина тоді і тільки тоді, коли $\operatorname{Conv} X = X$.
- Для довільної підмножини лінійного простору X існує єдина опукла оболонка $\operatorname{Conv} X$ — перетин усіх опуклих множин, що містять X.
  - При цьому

$$\operatorname{Conv} X = \bigcup_{n=1}^{\infty} \bigcup_{a_1,\ldots,a_n \in X} \bigcup_{\lambda_1+\cdots+\lambda_n=1} \lambda_1 a_1 + \cdots + \lambda_n a_n,\ \lambda_i \geq 0$$

  - Більш того, якщо вимірність простору дорівнює N тоді вірна наступна теорема Каратеодорі про опуклу оболонку:

$$\operatorname{Conv} X = \bigcup_{a_1,\ldots,a_{N+1} \in X} \bigcup_{\lambda_1+\cdots+\lambda_{N+1}=1} \lambda_1 a_1 + \cdots + \lambda_{N+1} a_{N+1},\ \lambda_i \geq 0$$

- Опуклою оболонкою скінченного набору точок на площині є опуклий плоский багатокутник (у вироджених випадках — відрізок або точка), причому його вершини є підмножиною похідного набору точок. Аналогічний факт вірний і для скінченного набору точок в багатовимірному просторі.
- Опукла оболонка X дорівнює перетину всіх півпросторів, що містять X.
- Для двох опуклих множин, які не перетинаються, завжди існує гіперплощина, що їх розділяє.

3. Алгоритм Джарвіса

- Алгоритм Джарвіса (або алгоритм загортання подарунка) — алгоритм знаходження опуклої оболонки. Часова складність — O(n * h), де n — кількість точок, h — кількість точок опуклої оболонки. Тобто, алгоритм найбільш ефективний у випадку малої кількості точок опуклої оболонки.

```cpp
//Point - some type that has x, y members and
//for which operator != is defined
template <typename Point>
inline bool determinantSignum(const Point& a,
                              const Point& b,
                              const Point& c)
{
    return (((b.x - a.x) * (c.y - b.y) -
             (c.x - b.x) * (b.y - a.y)) >= 0);
}

template <typename Point>
void jarvis(const std::vector<Point> & source,
            std::vector<Point>& result)
{
    if (source.empty())
    {
        return;
    }

    std::vector<Point> src = source;
    typedef typename std::vector<Point>::iterator PointIterator;
    PointIterator leftDownIt = src.begin();
    PointIterator it = leftDownIt + 1;
    Point currentPoint;
    Point leftDownPoint = *(leftDownIt);

    // Finding leftest lowest point

    for (PointIterator end = src.end(); it != end; ++it)
    {
        currentPoint = *it;
        if (currentPoint.x < leftDownPoint.x)
        {
            leftDownPoint = currentPoint;
            leftDownIt = it;
        }
        else if (currentPoint.x == leftDownPoint.x
                 && currentPoint.y < leftDownPoint.y)
        {
            leftDownPoint = currentPoint;
            leftDownIt = it;
        }
    }

    // Add selected point to answer
    src.erase(leftDownIt);
    result.push_back(leftDownPoint);

    currentPoint = leftDownPoint;

    Point nextPoint, tryPoint;
    PointIterator nextIt;

    do
    {
        nextPoint = leftDownPoint;
        nextIt = it;
        it = src.begin();
        for (PointIterator end = src.end(); it != end; ++it)
        {
            tryPoint = *it;
            if (determinantSignum(nextPoint,
                                  currentPoint,
                                  tryPoint))
```

```cpp
            if (determinantSignum(nextPoint,
                                  currentPoint,
                                  tryPoint))
            {
                nextPoint = tryPoint;
                nextIt = it;
            }
        }

        if (nextPoint != leftDownPoint)
        {
            src.erase(nextIt);
            result.push_back(nextPoint);
            currentPoint = nextPoint;
        }

    }
    while (nextPoint != leftDownPoint);
}
```

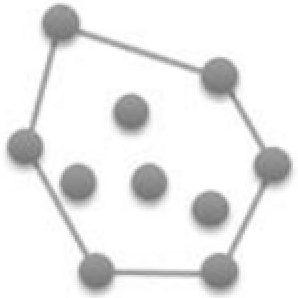
Приклад виконання алгоритму Джарвіса C++

Опис алгоритму Джарвіса

Нехай шукана опукла оболонка множини $P = \{p_1, p_2, \ldots p_n\}$ точок. Як початкову беремо найлівішу точку (точку з найменшою x-координатою), якщо їх буде декілька, то виберемо серед них найнижчу (точку з найменшою y-координатою). Нехай знайдена точка — точка p_1 (її можна знайти за час $O(n)$ звичайним проходом по всіх точках і порівнянням координат). Точка p_1 напевно є вершиною опуклої оболонки. Далі для кожної точки p_i шукаємо проти годинникової стрілки точки p_{i+1} шляхом знаходження за $O(n)$ серед точок, що залишились, (включно з p_1) точку з найменшим полярним кутом $p_{i-1}p_ip_{i+1}$. Вона і буде наступною вершиною опуклої оболонки. При цьому не обов'язково обчислювати кут — достатньо обчислити векторний добуток (узагальненням векторного добутку для двовимірного випадку є псевдоскалярний добуток) між векторами $p_ip'_{i+1}$ та $p_ip''_{i+1}$, де p'_{i+1} — знайдений на даний момент мінімум, p''_{i+1} — претендент (першим мінімумом може бути обрана довільна точка). Якщо векторний добуток від'ємний, то знайдено новий мінімум. Якщо рівний нулю, тобто p'_{i+1} та p''_{i+1} лежать на одній прямій, то мінімум — та, яка лежить далі від точки p_i . Алгоритм продовжує роботу доки $p_{i+1} \neq p_1$. Чому алгоритм зупиниться? Тому, що точка p_1 (найнижча серед найлівіших точок) у будь-якому випадку належить до точок опуклої оболонки.

4. Мінімально опукла оболонка C++

- Мінімальна опукла оболонка (МОО) — опукла оболонка, що лежить всередині всіх опуклих оболонок.
- Для знаходження мінімальних координатів МОО використав алгоритм Джарвіса.



Мінімальна опукла оболонка

- Код програми для знаходження МОО.

```cpp
#include <iostream>
#include <fstream>
#include <vector>
using namespace std;

int start, p[10];
struct point { int x, y; } a[10];
int rotate(point a, point b, point c)
{
    return (b.x - a.x) * (c.y - b.y) - (b.y - a.y) * (c.x - b.x);
}

int f(const int* k, const int* j)
{
    int r = rotate(a[start], a[*j], a[*k]);
    if (r != 0) return r; else
    {
        int dk = abs(a[start].x - a[*k].x) + abs(a[start].y - a[*k].y);
        int dj = abs(a[start].x - a[*j].x) + abs(a[start].y - a[*j].y);
        return dj - dk;
    }
}

int main()
{
    int j, n, n1;
    ifstream fi;
    ofstream fo;
    fi.open("graham.in.txt"); // 10 кількість точок в оболнці ->
    // 110 66  88 108  88 30  85 83  60 72  53 45  32 72  39 108  11 58  25 8 -> координати в оболонці
    fo.open("graham.out");
    fi >> n;
    n1 = n - 1;
    for (j = 0; j < n; j++)
    {
        fi >> a[j].x >> a[j].y;
        p[j] = j;
    }
    for (j = 0; j < n1; j++)
        if ((a[p[j]].x < a[p[n1]].x) ||
            ((a[p[j]].x == a[p[n1]].x) && (a[p[j]].y < a[p[n1]].y)))
            swap(p[n1], p[j]);
    start = p[n1];

    qsort(p, n1, sizeof(int), (int(*) (const void*, const void*)) f);

    vector <int> s = { p[n1],p[0] };
    bool line = true;
    for (j = 2; j < n; j++)
    {
        line = (rotate(a[0], a[1], a[j]) == 0);
        if (!line) break;
    }

    if (!line) for (j = 1; j < n1; j++)
    {
        while (rotate(a[s[s.size() - 2]], a[s[s.size() - 1]], a[p[j]]) < 0) s.pop_back();
        s.push_back(p[j]);
    }
    fo << s[0];
    for (j = 1; j < s.size(); j++) fo << " " << s[j];
    fo << endl;
    return 0;
}
```

Вхідні та вихідні дані

graham.in – Блокнот

Файл Правка Формат Вид Справка

```
10
110 66  88 108  88 30  85 83  60 72  53 45  32 72  39 108  11 58  25 8
```

Вхідний файл

graham – Блокнот

Файл Правка Формат Вид Справка

```
8 9 2 0 1 7
```

Результат програми

5. Опукла оболонка в повсякденному житті

- Розглянемо загальний випадок, де вхідними даними алгоритму є скінченна невпорядкована множина точок декартової площини.
- Якщо не всі точки лежать на одній прямій, їх опуклою оболонкою буде опуклий многокутник, вершини якого — це деякі точки зі вхідної множини. Найчастіше його подання є переліком вершин, відсортованих за годинниковою стрілкою або проти годинникової стрілки. В деяких випадках зручно представляти опуклий многокутник як перетин множини півплощин або півпросторів у просторовому випадку.

6. Висновки

- Опукла оболонка (англ. Convex hull) множини точок X на евклідовій площині або у просторі — це мінімальна опукла множина, що містить X. В обчислювальні геометрії, прийнято використовувати термін «опукла оболонка» для межі мінімальної опуклої множини, що містить дану не порожню скінченну множину точок на площині. Для скінченної множини точок, опукла оболонка являє собою ламану лінію.
- Мінімальна опукла оболонка (МОО) — опукла оболонка, що лежить всередині всіх опуклих оболонок.


Дякую за увагу!
Кчау
уютная
стена

Джерела

1. https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%94%D0%B6%D0%B5%D1%80%D0%B5%D0%BB%D0%B0

2. https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC%D0%B8_%D0%BE%D0%B1%D1%87%D0%B8%D1%81%D0%BB%D0%B5%D0%BD%D0%BD%D1%8F_%D0%BE%D0%BF%D1%83%D0%BA%D0%BB%D0%BE%D1%97_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D0%BD%D0%BA%D0%B8

3. https://uk.wikipedia.org/wiki/%D0%9E%D0%BF%D1%83%D0%BA%D0%BB%D0%B0_%D0%BE%D0%B1%D0%BE%D0%BB%D0%BE%D0%BD%D0%BA%D0%B0