平面凸包

简单定义

最小包含给定所有点的最小凸多边形

求凸包算法

给定n个点的二维坐标, 求其凸包.

斜率逼近法 -O(n * m)

先找一个y最小的点p_1
从p_1开始, 从$k = 0$, 开始寻找, 即：找一个$k_{min} > 0$的点, 作为p_2, 以此类推
若有多个k相同的点, 则取最距离最远的, 原因显而易见
根据k的变化, $k > 0 \to k < 0 \to k > 0 \dots$
问题是当$k \to \infty$不好处理

jarvis -O(n * m)

考虑用一根棍子从外往内扫, 每次第一次扫到的点显然就是凸包的点.
转化成数学语言：
我们很难用数学语言描述"扫", 但是当我们枚举当前点与其他点的所有棍子, 我们可以知道对于其中任意一根棍子, 我们可以知道其他棍子是在他的左边还是右边
即：我们考虑从左往右扫出凸包, 那么这根棍子显然应该是从左往右扫, 也就是说, 如果我们找到一根棍子l'在当前棍子l的逆时针, 那么显然l'更好, 那么通过枚举
不难知道, 如果三点贡献, 也就是两根棍子重叠的时候我们显然应该选取更长的那一根

Graham-O(n*log(n))

事实上, 该算法其实就是$jarvis$算法的优化版
通过分析$jarvis$, 我们不难有一个优化的想法就是：
如果每次找下一个点的时候, 如果我们不需要穷举所有点来找最优, 而类似于记忆化搜索一样, 考虑过的点不需要再考虑
那么类似于记忆化搜索原理一样的有拓扑序, 我们考虑将我们枚举的点有序化, 这样来防止我们枚举过的点对后面的凸包还有贡献
由于我们总的思想是从左往右扫, 第一个碰到的点就是下一个点, 那么我们考虑的点显然也是应该从左到右, 更具体来说：
我们先选定一个始点(比如最左下角的), 然后我们按其他点与始点的极角排序(可以用叉乘或者$atan2$), 这样就完成有序化
接下来我们考虑维护一个凸包的点集, 考虑当前枚举的点对凸包的贡献
记我们当前考虑的点为$p1$, 上一个进入凸包的点为$p2$, 上上个进入凸包的点为$p3$
显然如果$p3 \to p1$在$p3 \to p2$的逆时针方向, 那么显然我们将$p2$从点集中舍弃, 将$p1$塞入点集
进一步的, 我们用考虑$p2$的方式来考虑$p3$, 直到不满足条件为止, 最后再塞入$p1$
事实上维护这个点集我们可以用单调栈来实现
总的来说做法：
我们选定一个初始点, 然后将其他点有序化
然后枚举每个点来维护一个有序的答案集

模板

stl单调栈

```
#include <bits/stdc++.h>
#define int long long
#define endl '\n'
#define LL __int128
using namespace std;
int qpow(int a, int b, int p) {int ret = 1; for(a %= p; b; b >>= 1, a = a * a % p) if(b & 1) ret = ret * a % p; return ret; }
int qpow(int a,int b) {int ret = 1; for(; b; b >>= 1, a *= a) if(b & 1) ret *= a; return ret; }
int gcd(int x,int y) {return y ? gcd(y, x % y) : x; }
pair<int,int> exgcd(int a,int b) { if(!b) return {1, 0}; pair<int,int> ret = exgcd(b, a % b); return {ret.second, ret.first - a / b * ret.second }; }
int lcm(int x,int y){ return x / gcd(x, y) * y; }

const int N = 5 + 1e5;

struct node{
    double x, y;
};
double mul(node a1, node a2, node b1, node b2) {
    return (a1.x - a2.x) * (b1.y - b2.y) - (a1.y - a2.y) * (b1.x - b2.x);
}
double d(node a, node b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
```

```
node a[N];
signed main() {
    int n; cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i].x >> a[i].y;
    }
    sort(a, a + n, [&](node u, node v){
        if (u.x != v.x) return u.x < v.x;
        else return u.y < v.y;
    });
    sort(a + 1, a + n, [&](node u, node v){
        int ck = mul(u, a[0], v, a[0]);
        if (ck < 0) {
            return 1;
        } else if (!ck && d(u, a[0]) < d(v, a[0])) {
            return 1;
        } else {
            return 0;
        }
    });
    stack<node> s;
    for (int i = 0; i < n; i++) {
        if (s.size() < 2) {
            s.push(a[i]);
        } else {
            node q1 = s.top(); s.pop();
            node q2 = s.top();
            while (s.size() > 1 && mul(q2, q1, q1, a[i]) >= 0) {
                q1 = q2;
                s.pop();
                q2 = s.top();
            }
            s.push(q1);
            s.push(a[i]);
        }
    }
    double ans = 0;
    for(node prev = a[0]; s.size(); s.pop()) {
        ans += d(prev, s.top());
        prev = s.top();
    }
    cout << fixed << setprecision(2) << ans << endl;
}
```

手写单调栈

```
#include <bits/stdc++.h>
#define int long long
#define endl '\n'
#define LL __int128
using namespace std;
int qpow(int a, int b, int p) {int ret = 1; for(a %= p; b; b >>= 1, a = a * a % p) if(b & 1) ret = ret * a % p; return ret; }
int qpow(int a,int b) {int ret = 1; for(; b; b >>= 1, a *= a) if(b & 1) ret *= a; return ret; }
int gcd(int x,int y) {return y ? gcd(y, x % y) : x; }
pair<int,int> exgcd(int a,int b) { if(!b) return {1, 0}; pair<int,int> ret = exgcd(b, a % b); return {ret.second, ret.first - a / b * ret.second }; }
int lcm(int x,int y){ return x / gcd(x, y) * y; }

const int N = 5 + 1e5;

struct node{
    double x, y;
};
double mul(node a1, node a2, node b1, node b2) {
    return (a1.x - a2.x) * (b1.y - b2.y) - (a1.y - a2.y) * (b1.x - b2.x);
}
double d(node a, node b) {
    return sqrt((a.x - b.x) * (a.x - b.x) + (a.y - b.y) * (a.y - b.y));
}
node a[N];
int cnt;
node q[N];
signed main() {
    int n; cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> a[i].x >> a[i].y;
    }
    sort(a, a + n, [&](node u, node v){
        if (u.x != v.x) return u.x < v.x;
        else return u.y < v.y;
    });
    sort(a + 1, a + n, [&](node u, node v){
        int ck = mul(u, a[0], v, a[0]);
        if (ck > 0) {
            return 1;
```

```
        } else if (!ck && d(u, a[0]) < d(v, a[0])) {
            return 1;
        } else {
            return 0;
        }
    });
    cnt = 0;
    for (int i = 0; i < n; i++) {
        if (cnt < 2) {
            q[++cnt] = a[i];
        } else {
            while (cnt > 1 && mul(q[cnt - 1], q[cnt], q[cnt], a[i]) <= 0) {
                cnt--;
            }
            q[++cnt] = a[i];
        }
    }
    q[++cnt] = a[0];
    double ans = 0;
    for (int i = 1; i < cnt; i++) {
        ans += d(q[i], q[i + 1]);
    }
    cout << fixed << setprecision(2) << ans << endl;
}
```

Andrew-O(n*log(n))

事实上,该算法和$Graham$的思想类似,都是将点有序化,然后再维护一个有序的答案集
将点按x从小到大排序
从第一个点开始遍历，如果下一个点在栈顶的两个元素所连成直线的左边，那么就入栈
否则如果在右边，说明凸包有更优的方案，上次的点出栈，并直到新点在栈顶两个点所在的直线的左边为止