锋从膜力出

一点微小的工作

朱永南

苏州大学 • 机电工程学院

2017 1.19

简单数论

1. 快速幂、矩阵快速幂
2. 费马小定理
3. 乘法逆元
4. 指数降幂公式
5. 素数筛、区间素数筛
6. 欧拉函数
7. 卢卡斯定理
8. 扩展欧几里德
9. 模线性方程组
10. 中国剩余定理(CRT)
11. 积性函数
12. 约数筛
13. 莫比乌斯反演
14. FFT/NTT

还有数学相关比较重要的：

博弈论、容斥原理、高斯消元、Burnside引理和Polya原理

动态规划(Dynamic Programming)

1. LIS,LCS
2. 背包
3. 区间DP
4. 树型DP
5. 数位DP
6. 概率期望DP
7. 状压DP
8. DP优化
   - 二进制优化
   - 单调队列、单调栈
   - 斜率优化
   - bitset优化
   - 平行四边形不等式

还有：

有关状态定义、博弈递归型DP、不(x)知(j)名(b)DP

计算几何

1. 常用三角函数
2. 点积、叉积
3. 直线、线段相交
4. 点与多边形关系
5. 圆相关
6. 极角排序
7. 二维凸包
   - Andrew扫描法
   - 稳定凸包
   - 凸包周长、面积
8. 旋转卡壳
   - 凸包直径
   - 最小矩形覆盖
   - 最大三角形面积
   - ……

树

1. 树直径
2. LCA
3. 任意两点路径最小边权
4. dfs序
5. 最小生成树
6. 曼哈顿距离MST
7. 有关树的构造
8. 树上莫队
9. 最小树形图
10. 生成树计数
11. 树分治
12. 普吕弗序列

技巧

1. STL
2. 二分、三分
3. 尺取(双指针)
4. ~~蛤习~~ 哈希
5. 倍增
6. 打表找规律
7. random_shuffle()
8. next_permutation()
9. 字典树
10. 折半搜索
11. cdq分治
12. .....

二分(Binary Search)

条件：单调性

例：在数组

$$A[7] = \{0, 666, 5, 15, 88, 233, 34\}$$

中找到第一个> 100的数字？

如果查找多次？复杂度？

```cpp
#include <algorithm>
using namespace std;
sort(A, A + 7); // {0,5,15,34,88,233,666}
```

```cpp
int low = 0, high = n - 1, mid; // n = 7
while (low < high) {
    mid = (low + high) >> 1; //(low+high) / 2
    if (A[mid] > goal) high = mid; // goal=100
    else low = mid + 1;
}
printf("%d\n", high);
```

二分(Binary Search)

条件：单调性

例：在数组

$$A[7] = \{0, 666, 5, 15, 88, 233, 34\}$$

中找到第一个>100的数字？

如果查找多次？复杂度？

```cpp
#include <algorithm>
using namespace std;
sort(A, A + 7); // {0,5,15,34,88,233,666}
```

```cpp
int low = 0, high = n - 1, mid; // n = 7
while (low < high) {
    mid = (low + high) >> 1; //(low+high) / 2
    if (A[mid] > goal) high = mid; // goal=100
    else low = mid + 1;
}
printf("%d\n", high);
```

二分(Binary Search)

条件：单调性

例：在数组

$$A[7] = \{0, 666, 5, 15, 88, 233, 34\}$$

中找到第一个>100的数字？

如果查找多次？复杂度？

```cpp
#include <algorithm>
using namespace std;
sort(A, A + 7); // {0,5,15,34,88,233,666}
```

```cpp
int low = 0, high = n - 1, mid; // n = 7
while (low < high) {
    mid = (low + high) >> 1; //(low+high) / 2
    if (A[mid] > goal) high = mid; // goal=100
    else low = mid + 1;
}
printf("%d\n", high);
```

二分(Binary Search)

条件：单调性

例：在数组

$$A[7] = \{0, 666, 5, 15, 88, 233, 34\}$$

中找到第一个>100的数字？

如果查找多次？复杂度？

```
#include <algorithm>
using namespace std;
sort(A, A + 7); // {0,5,15,34,88,233,666}
```

```
int low = 0, high = n - 1, mid; // n = 7
while (low < high) {
    mid = (low + high) >> 1; //(low+high) / 2
    if (A[mid] > goal) high = mid; // goal=100
    else low = mid + 1;
}
printf("%d\n", high);
```

二分(Binary Search)

lower_bound() : $\geq$的第一个位置
upper_bound() : $>$的第一个位置

$$A[7] = \{0, 5, 15, 34, 88, 233, 666\}$$

```
1 printf("%d\n", lower_bound(A, A + 7, 5) - A);
2 // 1
3 printf("%d\n", upper_bound(A, A + 7, 88) - A);
4 // 5
5 printf("%d\n", lower_bound(A, A + 7, 777) - A);
6 // 7
```

某个数出现在数组中的次数?

```
1 int count = upper_bound() - lower_bound();
```

二分(Binary Search)

lower_bound() : $\geq$的第一个位置
upper_bound() : $>$的第一个位置

$$A[7] = \{0, 5, 15, 34, 88, 233, 666\}$$

```
printf("%d\n", lower_bound(A, A + 7, 5) - A);
// 1
printf("%d\n", upper_bound(A, A + 7, 88) - A);
// 5
printf("%d\n", lower_bound(A, A + 7, 777) - A);
// 7
```

某个数出现在数组中的次数?

```
int count = upper_bound() - lower_bound();
```

浮点数二分

```
// eps = 1e-7
while (low < high) { ... } // ???
while (high - low > eps) { ... } // TLE
```

一般直接指定循环次数（100次）作为终止条件。1次循环可以把区间的范围缩小一半，100次的循环则可以达到$2^{-100} \approx 10^{-30}$的精度范围，基本上是没有问题的。

```
for (int i = 0; i < 100; ++i) {
    double mid = (low + high) / 2;
    if (check(mid)) high = mid; // low = mid;
    else low = mid; // high = mid;
}
```

浮点数二分

```
1 // eps = 1e-7
2 while (low < high) { ... } // ???
3 while (high - low > eps) { ... } // TLE
```

一般直接指定循环次数（100次）作为终止条件。1次循环可以把区间的范围缩小一半，100次的循环则可以达到$2^{-100} \approx 10^{-30}$的精度范围，基本上是没有问题的。

```
1 for (int i = 0; i < 100; ++i) {
2     double mid = (low + high) / 2;
3     if (check(mid)) high = mid; // low = mid;
4     else low = mid; // high = mid;
5 }
```

浮点数二分

```
1 // eps = 1e-7
2 while (low < high) { ... } // ???
3 while (high - low > eps) { ... } // TLE
```

一般直接指定循环次数（100次）作为终止条件。1次循环可以把区间的范围缩小一半，100次的循环则可以达到$2^{-100} \approx 10^{-30}$的精度范围，基本上是没有问题的。

```
1 for (int i = 0; i < 100; ++i) {
2     double mid = (low + high) / 2;
3     if (check(mid)) high = mid; // low = mid;
4     else low = mid; // high = mid;
5 }
```

一个栗子

给$n \leq 10000$个根木棍的长度，可以将木根任意剪切，但不可拼接，要求切出来$K \leq 10000$根等长的木棍，求这K 根木棍最长可以切得的长度是？
结果保留两位输出，长度单位是米，剪切精度是厘米。如果最终得到的长度小于1厘米，输出0.00。

输入：

```
4  11
8.02
7.43
4.57
5.39
1  2
2.33
1  10000
1.23
```

输出：

```
2.00
1.66
0.00
```

一个栗子

显然答案具有单调性：

如果可以切得b米，那么一定可以切得$a(a < b)$米。

所以可以二分。

下界：$low = 0$　　上界：$high = max(len[i])$

```
for (int i = 0; i < 100; ++i) {
    double mid = (low + high) / 2;
    if (check(mid)) low = mid;
    else high = mid;
}
printf("%.2lf\n", low); // ??? too naive
```

```
#include <math.h>
printf("%.2lf\n", (floor)(low * 100) / 100);
```

一个栗子

显然答案具有单调性：

如果可以切得b米，那么一定可以切得$a(a < b)$米。

所以可以二分。

下界：$low = 0$　　上界：$high = max(len[i])$

```
for (int i = 0; i < 100; ++i) {
    double mid = (low + high) / 2;
    if (check(mid)) low = mid;
    else high = mid;
}
printf("%.2lf\n", low); // ??? too naive
```

```
#include <math.h>
printf("%.2lf\n", (floor)(low * 100) / 100);
```

一个栗子

显然答案具有单调性：

如果可以切得b米，那么一定可以切得$a(a < b)$米。

所以可以二分。

下界：$low = 0$　　上界：$high = max(len[i])$

```
for (int i = 0; i < 100; ++i) {
    double mid = (low + high) / 2;
    if (check(mid)) low = mid;
    else high = mid;
}
printf("%.2lf\n", low); // ??? too naive
```

```
#include <math.h>
printf("%.2lf\n", (floor)(low * 100) / 100);
```

Problems

LightOJ 1138

求最小的自然数n使得n的阶乘结果以$Q \leq 10^{18}$个0结尾。如果不存在这样的数输出impossible。

对于$x \in N$，其阶乘末尾0的个数$num = \frac{x}{5} + \frac{x}{25} + \frac{x}{125} + \frac{x}{625} + \cdots$

对于任意的$x' > x$，其阶乘末尾0的个数$num' \geq num$

HDU 3264

给出$n \leq 20$个圆的圆心和半径，以这些圆的某个圆圆心为圆心画一个半径为R的圆，使得这个半径为R的圆覆盖这n个圆每个圆的面积至少一半，求出最少的R.

枚举圆心，二分最小半径，浮点数二分。

Problems

LightOJ 1138

求最小的自然数n使得n的阶乘结果以$Q \leq 10^{18}$个0结尾。如果不存在这样的数输出impossible。

对于$x \in N$，其阶乘末尾0的个数$num = \frac{x}{5} + \frac{x}{25} + \frac{x}{125} + \frac{x}{625} + \cdots$
对于任意的$x' > x$，其阶乘末尾0的个数$num' \geq num$

HDU 3264

给出$n \leq 20$个圆的圆心和半径，以这些圆的某个圆圆心为圆心画一个半径为R的圆，使得这个半径为R的圆覆盖这n个圆每个圆的面积至少一半，求出最少的R.

枚举圆心，二分最小半径，浮点数二分。

Problems

LightOJ 1138

求最小的自然数n使得n的阶乘结果以$Q \leq 10^{18}$个0结尾。如果不存在这样的数输出impossible。

对于$x \in N$，其阶乘末尾0的个数$num = \frac{x}{5} + \frac{x}{25} + \frac{x}{125} + \frac{x}{625} + \cdots$
对于任意的$x' > x$，其阶乘末尾0的个数$num' \geq num$

HDU 3264

给出$n \leq 20$个圆的圆心和半径，以这些圆的某个圆圆心为圆心画一个半径为R的圆，使得这个半径为R的圆覆盖这n个圆每个圆的面积至少一半，求出最少的R.

枚举圆心，二分最小半径，浮点数二分。

Problems

LightOJ 1138

求最小的自然数n使得n的阶乘结果以$Q \leq 10^{18}$个0结尾。如果不存在这样的数输出impossible。

对于$x \in N$，其阶乘末尾0的个数$num = \frac{x}{5} + \frac{x}{25} + \frac{x}{125} + \frac{x}{625} + \cdots$
对于任意的$x' > x$，其阶乘末尾0的个数$num' \geq num$

HDU 3264

给出$n \leq 20$个圆的圆心和半径，以这些圆的某个圆圆心为圆心画一个半径为R的圆，使得这个半径为R的圆覆盖这n个圆每个圆的面积至少一半，求出最少的R.

枚举圆心，二分最小半径，浮点数二分。

树直径

定义：树上任意两点最短路径的最大值
边关系的存储：链式前向星或者vector

```cpp
#include <vector>

// MAX_N: node number
vector<pair<int, int> > edge[MAX_N];

// Add edge: (u, v) = w
edge[u].push_back(make_pair(v, w));
edge[v].push_back(make_pair(u, w));

// Traverse node u's edge
for (int i = 0; i < edge[u].size(); ++i) {
    int v = edge[i].first, w = edge[i].second;
    ......
}
```

树直径

定义：树上任意两点最短路径的最大值
边关系的存储：链式前向星或者vector

```cpp
#include <vector>

// MAX_N: node number
vector<pair<int, int> > edge[MAX_N];

// Add edge: (u, v) = w
edge[u].push_back(make_pair(v, w));
edge[v].push_back(make_pair(u, w));

// Traverse node u's edge
for (int i = 0; i < edge[u].size(); ++i) {
    int v = edge[i].first, w = edge[i].second;
    ......
}
```

链式前向星

```
1  int head[MAX_N], tot; // MAX_N: node number
2
3  struct Edge {
4      int to, wei, nxt;
5  } edge[MAX_M]; // MAX_M: edge number
6
7  void init() { // initialize
8      tot = 0;
9      memset(head, -1, sizeof (head));
10 }
11
12 void AddEdge(int u, int v, int w) {
13     edge[tot].to = v, edge[tot].nxt = head[u];
14     edge[tot].wei = w;
15     head[u] = tot++;
16 }
17
18 // Traverse node u's edge
19 for (int i = head[u]; i != -1; i = edge[i].nxt) {
20     int v = edge[i].v, w = edge[i].w;
21     ......
22 }
```

定理

以任意结点出发所能到达的最远点，一定是树直径的端点之一。

证明：假设直径是$\delta(s,e)$，任意结点为x，其最远能到达的结点为y。分两种情况：

- 如果x是直径$\delta(s,e)$上的结点，如果y不是s,e之一，即：$xy > xs, xy > xe$，此时满足：

$$ys = yx + xs > xe + xs = se$$

和

$$ye = yx + xe > xs + xe = se$$

这与直径是$\delta(s,e)$不符,所以y必然是s,e之一。
- 如果x不是直径$\delta(s,e)$上的结点，设xy路径上一结点z。如果z在直径$\delta(s,e)$上，根据上面的证明可知y必然是s,e之一。如果z不在直径$\delta(s,e)$上，即路径xy和直径$\delta(s,e)$完全不相交，这时连接sx,se，易得:

$$sy = sx + xy > sx + xe = se$$

这又与$\delta(s,e)$是直径相违，所以y也必然是s,e之一。

定理

以任意结点出发所能到达的最远点，一定是树直径的端点之一。

证明：假设直径是$\delta(s,e)$，任意结点为x，其最远能到达的结点为y。分两种情况：

- 如果x是直径$\delta(s,e)$上的结点，如果y不是s,e之一，即：$xy > xs, xy > xe$，此时满足：

  $$ys = yx + xs > xe + xs = se$$

  和

  $$ye = yx + xe > xs + xe = se$$

  这与直径是$\delta(s,e)$不符,所以y必然是s,e之一。
- 如果x不是直径$\delta(s,e)$上的结点，设xy路径上一结点z。如果z在直径$\delta(s,e)$上，根据上面的证明可知y必然是s,e之一。如果z不在直径$\delta(s,e)$上，即路径xy和直径$\delta(s,e)$完全不相交，这时连接sx, se，易得:

  $$sy = sx + xy > sx + xe = se$$

  这又与$\delta(s,e)$是直径相违，所以y也必然是s,e之一。

定理

以任意结点出发所能到达的最远点，一定是树直径的端点之一。

证明：假设直径是$\delta(s,e)$，任意结点为x，其最远能到达的结点为y。分两种情况：

- 如果x是直径$\delta(s,e)$上的结点，如果y不是s,e之一，即：$xy > xs, xy > xe$，此时满足：

  $$ys = yx + xs > xe + xs = se$$

  和

  $$ye = yx + xe > xs + xe = se$$

  这与直径是$\delta(s,e)$不符,所以y必然是s,e之一。
- 如果x不是直径$\delta(s,e)$上的结点，设xy路径上一结点z。如果z在直径$\delta(s,e)$上，根据上面的证明可知y必然是s,e之一。如果z不在直径$\delta(s,e)$上，即路径xy和直径$\delta(s,e)$完全不相交，这时连接sx,se，易得:

  $$sy = sx + xy > sx + xe = se$$

  这又与$\delta(s,e)$是直径相违，所以y也必然是s,e之一。

Simple Code

如何获得距离某一点的最远点？ dfs/bfs啊，少年！

```
void dfs(int u) { // int dis[MAX_N], vis[MAX_N];
    vis[u] = 1;
    for (int i = 0; i < edge[u].size(); ++i) {
        int v = edge[i].first, w = edge[i].second;
        if (vis[v]) continue;
        dis[v] = dis[u] + w;
        dfs(v);
    }
}
void GetDiameter() {
    dis[1] = 0; dfs(1); memset(vis, 0, sizeof (vis));
    int Max = 0, st = -1;
    for (int i = 1; i <= n; ++i) {
        if (dis[i] > Max) { Max = dis[i]; st = i; }
    }
    dis[st] = 0; memset(vis, 0, sizeof (vis));
    dfs(st, -1); Max = 0;
    for (int i = 1; i <= n; ++i) Max = max(Max, dis[i]);
    printf("%d\n", Max);
}
```

Simple Code

如何获得距离某一点的最远点？ dfs/bfs啊，少年！

```
1  void dfs(int u) { // int dis[MAX_N], vis[MAX_N];
2      vis[u] = 1;
3      for (int i = 0; i < edge[u].size(); ++i) {
4          int v = edge[i].first, w = edge[i].second;
5          if (vis[v]) continue;
6          dis[v] = dis[u] + w;
7          dfs(v);
8      }
9  }
10 void GetDiameter() {
11     dis[1] = 0; dfs(1); memset(vis, 0, sizeof (vis));
12     int Max = 0, st = -1;
13     for (int i = 1; i <= n; ++i) {
14         if (dis[i] > Max) { Max = dis[i]; st = i; }
15     }
16     dis[st] = 0; memset(vis, 0, sizeof (vis));
17     dfs(st, -1); Max = 0;
18     for (int i = 1; i <= n; ++i) Max = max(Max, dis[i]);
19     printf("%d\n", Max);
20 }
```

Are you OK?

时间复杂度？bfs的写法？获得直径上的边？
有没有别的方法求树直径？

求出每个结点到其他结点的最远距离：这个最远距离可能来自结点的子树也可能来自结点的父亲结点方向。[树型dp]

Are you OK?

时间复杂度？bfs的写法？获得直径上的边？
有没有别的方法求树直径？

求出每个结点到其他结点的最远距离：这个最远距离可能来自结点的子树也可能来自结点的父亲结点方向。[树型dp]

Problems

CF 755C

有$n \leq 10^4$个点，每个点属于一棵树，给出每个点到它所在树最远点的编号（多个最远点取最小），求有多少棵树？

每个点的最远点是树直径端点
用并查集合并树，求公共祖先个数

CF 746G

给出$n \leq 2 * 10^5$个点和每一层节点个数，根节点为1，且根节点到其他点的最远距离(路径上边的个数)为$t < n$，求构造一棵仅含$K < n$个叶子的树，输出每种度的点的数量。

先把根的最长链构造出来
接着尽量使树的结点个数尽量多
再调整结点个数使其为K个
// 好想难打

Problems

CF 755C

有$n \leq 10^4$个点，每个点属于一棵树，给出每个点到它所在树最远点的编号（多个最远点取最小），求有多少棵树？

每个点的最远点是树直径端点
用并查集合并树，求公共祖先个数

CF 746G

给出$n \leq 2 * 10^5$个点和每一层节点个数，根节点为1，且根节点到其他点的最远距离(路径上边的个数)为$t < n$，求构造一棵仅含$K < n$个叶子的树，输出每种度的点的数量。

先把根的最长链构造出来
接着尽量使树的结点个数尽量多
再调整结点个数使其为K个
// 好想难打

Problems

CF 755C

有$n \leq 10^4$个点，每个点属于一棵树，给出每个点到它所在树最远点的编号（多个最远点取最小），求有多少棵树？

每个点的最远点是树直径端点
用并查集合并树，求公共祖先个数

CF 746G

给出$n \leq 2 * 10^5$个点和每一层节点个数，根节点为1，且根节点到其他点的最远距离(路径上边的个数)为$t < n$，求构造一棵仅含$K < n$个叶子的树，输出每种度的点的数量。

先把根的最长链构造出来
接着尽量使树的结点个数尽量多
再调整结点个数使其为K个
// 好想难打

Problems

CF 755C

有$n \leq 10^4$个点，每个点属于一棵树，给出每个点到它所在树最远点的编号（多个最远点取最小），求有多少棵树？

每个点的最远点是树直径端点
用并查集合并树，求公共祖先个数

CF 746G

给出$n \leq 2 * 10^5$个点和每一层节点个数，根节点为1，且根节点到其他点的最远距离(路径上边的个数)为$t < n$，求构造一棵仅含$K < n$个叶子的树，输出每种度的点的数量。

先把根的最长链构造出来
接着尽量使树的结点个数尽量多
再调整结点个数使其为K个
// 好想难打

单调栈

单调栈主要用于解决某个元素它向左向右为最大值或最小值的最大范围是什么。
如果是最大值，那就要维护单调非递增栈（唯一最大就是单调递减栈），如果是最小值就要维护单调非递减栈（唯一最小就是单调递增栈）。

什么是单调的？如何维护单调性？和栈顶元素比较

$A[] = \{1, 3, 2, 5, 4, 3, 7\}$

维护单调递增性：

当前遍历元素	栈中下标相应元素				
{1}	1				
{1,3}	1	3			
{1,3,2}	1	2			
{1,3,2,5}	1	2	5		
{1,3,2,5,4}	1	2	4		
{1,3,2,5,4,3}	1	2	3		
{1,3,2,5,4,3,7}	1	2	3	7	

单调栈

单调栈主要用于解决某个元素它向左向右为最大值或最小值的最大范围是什么。
如果是最大值，那就要维护单调非递增栈（唯一最大就是单调递减栈），如果是最小值就要维护单调非递减栈（唯一最小就是单调递增栈）。

什么是单调的？如何维护单调性？和栈顶元素比较

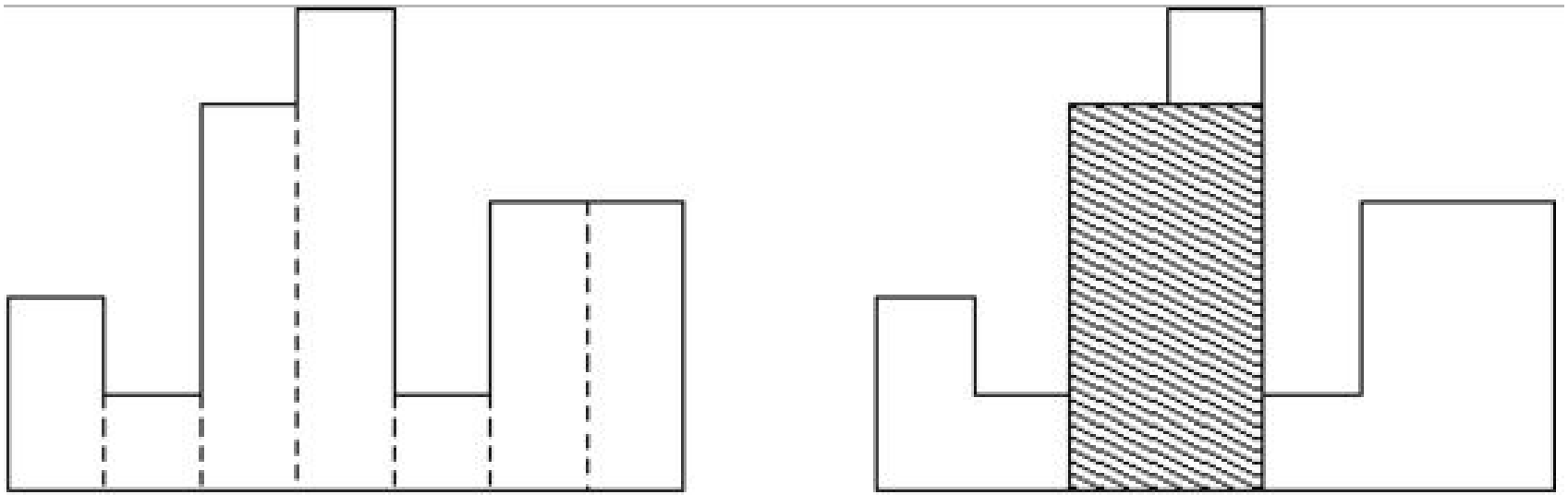
$$A[] = \{1, 3, 2, 5, 4, 3, 7\}$$

维护单调递增性：

当前遍历元素	栈中下标相应元素				
{1}	1				
{1,3}	1	3			
{1,3,2}	1	2			
{1,3,2,5}	1	2	5		
{1,3,2,5,4}	1	2	4		
{1,3,2,5,4,3}	1	2	3		
{1,3,2,5,4,3,7}	1	2	3	7	

两个栗子

给出一排n个紧密并列的矩形的高$h[i] \leq 10^9$，宽均为1，求从中可以划分出的最大的矩形面积？



```
Sample Input:
7 2 1 4 5 1 3 3
4 1000 1000 1000 1000
```

```
Sample Output:
8
4000
```

两个栗子

问题等价于：求每个长方形以它的高为最小值的最左最右范围
先看左边界：维护单调递增栈，栈中记录的是下标

$$H[] = \{-1, 2, 1, 4, 5, 1, 3, 3, 0\}$$

L[i]表示左边界，R[i]表示右边界，均是开区间

当前遍历元素	栈中元素					L[i]	R[i]
{-1}	0						
{-1,2}	0	1				L[1] = 0	
{-1,2,1}	0	2				L[2] = 0	R[1] = 2
{-1,2,1,4}	0	2	3			L[3] = 2	
{-1,2,1,4,5}	0	2	3	4		L[4] = 3	
{-1,2,1,4,5,1}	0	5				L[5] = 0	...
{-1,2,1,4,5,1,3}	0	5	6			L[6] = 5	
{-1,2,1,4,5,1,3,3}	0	5	7			L[7] = 5	R[6] = 7
{-1,2,1,4,5,1,3,3,0}	0	8					R[5] = R[7] = 8

两个栗子

问题等价于：求每个长方形以它的高为最小值的最左最右范围
先看左边界：维护单调递增栈，栈中记录的是下标

$$H[] = \{-1, 2, 1, 4, 5, 1, 3, 3, 0\}$$

L[i]表示左边界，R[i]表示右边界，均是开区间

当前遍历元素	栈中元素					L[i]	R[i]
{-1}	0						
{-1,2}	0	1				L[1] = 0	
{-1,2,1}	0	2				L[2] = 0	R[1] = 2
{-1,2,1,4}	0	2	3			L[3] = 2	
{-1,2,1,4,5}	0	2	3	4		L[4] = 3	
{-1,2,1,4,5,1}	0	5				L[5] = 0	...
{-1,2,1,4,5,1,3}	0	5	6			L[6] = 5	
{-1,2,1,4,5,1,3,3}	0	5	7			L[7] = 5	R[6] = 7
{-1,2,1,4,5,1,3,3,0}	0	8					R[5] = R[7] = 8

Code

```
height[0] = -1, height[n + 1] = 0;
int top = 0; sta[0] = 0;
for (int i = 1; i <= n + 1; ++i) {
    while(1) {
        int cur = sta[top];
        if (height[cur] < height[i]) break;
        R[cur] = i;
        top--;
    }
    L[i] = sta[top];
    sta[++top] = i;
}
```

Problems

POJ 3494

给一个$n * m(n, m \leq 2000)$的01矩阵，求最大全1子矩阵面积

每一行单独处理，把从这行每一列向上连续延伸全为1的最大长度看成是矩形的高

每个高求向左向右为最小值的范围，维护单调非递减栈

GYM 101102D

给出一个$R * C(R, C \leq 1000)$的方格矩阵，每个方格有一个数字$A[i][j] \leq 10^9$，求全等子矩阵个数

计算每个方格以它为右下角时的全等子矩阵个数

每一行单独处理，每一列求出向上完全相同的高度

转化为计算面积，维护单调递增栈

Problems

POJ 3494

给一个$n * m(n, m \leq 2000)$的01矩阵，求最大全1子矩阵面积

每一行单独处理，把从这行每一列向上连续延伸全为1的最大长度看成是矩形的高
每个高求向左向右为最小值的范围，维护单调非递减栈

GYM 101102D

给出一个$R * C(R, C \leq 1000)$的方格矩阵，每个方格有一个数字$A[i][j] \leq 10^9$，求全等子矩阵个数

计算每个方格以它为右下角时的全等子矩阵个数
每一行单独处理，每一列求出向上完全相同的高度
转化为计算面积，维护单调递增栈

Problems

POJ 3494

给一个$n * m(n, m \leq 2000)$的01矩阵，求最大全1子矩阵面积

每一行单独处理，把从这行每一列向上连续延伸全为1的最大长度看成是矩形的高
每个高求向左向右为最小值的范围，维护单调非递减栈

GYM 101102D

给出一个$R * C(R, C \leq 1000)$的方格矩阵，每个方格有一个数字$A[i][j] \leq 10^9$，求全等子矩阵个数

计算每个方格以它为右下角时的全等子矩阵个数
每一行单独处理，每一列求出向上完全相同的高度
转化为计算面积，维护单调递增栈

Problems

POJ 3494

给一个$n * m(n, m \leq 2000)$的01矩阵，求最大全1子矩阵面积

每一行单独处理，把从这行每一列向上连续延伸全为1的最大长度看成是矩形的高
每个高求向左向右为最小值的范围，维护单调非递减栈

GYM 101102D

给出一个$R * C(R, C \leq 1000)$的方格矩阵，每个方格有一个数字$A[i][j] \leq 10^9$，求全等子矩阵个数

计算每个方格以它为右下角时的全等子矩阵个数
每一行单独处理，每一列求出向上完全相同的高度
转化为计算面积，维护单调递增栈

Problems

POJ 3494

给一个$n * m(n, m \leq 2000)$的01矩阵，求最大全1子矩阵面积

每一行单独处理，把从这行每一列向上连续延伸全为1的最大长度看成是矩形的高
每个高求向左向右为最小值的范围，维护单调非递减栈

GYM 101102D

给出一个$R * C(R, C \leq 1000)$的方格矩阵，每个方格有一个数字$A[i][j] \leq 10^9$，求全等子矩阵个数

计算每个方格以它为右下角时的全等子矩阵个数
每一行单独处理，每一列求出向上完全相同的高度
转化为计算面积，维护单调递增栈

常用三角函数与公式

1. $\pi = \text{acos(-1.0)}$
2. $S = \frac{absin\theta}{2}$
3. $S = \sqrt{p(p-a)(p-b)(p-c)}$
   $(p = \frac{a+b+c}{2})$
4. 点积：$\vec{a} \bullet \vec{b} =$
   $x_1 \times x_2 + y_1 \times y_2 = |\vec{a}||\vec{b}|cos\theta$
5. 叉积：$\vec{a} * \vec{b} =$
   $x_1 \times y_2 - x_2 \times y_1 = |\vec{a}||\vec{b}|sin\theta$

- 点积和叉积中的夹角θ指：$\vec{a}$逆时针旋转到$\vec{b}$的角度
- $\vec{a} * \vec{b}$ 与 $\vec{b} * \vec{a}$
- $\vec{a} * \vec{b} < 0$: $\vec{b}$在$\vec{a}$右侧
- 叉积也可以计算三角形面积

常用三角函数与公式

1. $\pi = \text{acos(-1.0)}$
2. $S=\frac{absin\theta}{2}$
3. $S = \sqrt{p(p-a)(p-b)(p-c)}$
   $(p = \frac{a+b+c}{2})$
4. 点积：$\vec{a} \bullet \vec{b} =$
   $x_1 \times x_2 + y_1 \times y_2 = |\vec{a}||\vec{b}|cos\theta$
5. 叉积：$\vec{a} * \vec{b} =$
   $x_1 \times y_2 - x_2 \times y_1 = |\vec{a}||\vec{b}|sin\theta$

- 点积和叉积中的夹角θ指：$\vec{a}$逆时针旋转到$\vec{b}$的角度
- $\vec{a} * \vec{b}$ 与$\vec{b} * \vec{a}$
- $\vec{a} * \vec{b} < 0$: $\vec{b}$在$\vec{a}$右侧
- 叉积也可以计算三角形面积

点类

```
int sgn(double x) { // const double eps = 1e-7;
    if (fabs(x) < eps) return 0;
    if (x < 0) return -1;
    return 1;
}

struct Point {
    double x, y; // int

    Point () {}
    Point(double _x, double _y): x(_x), y(_y) {}
    // Point a = Point(1, 4), b = Point(2, 3);
    bool operator < (const Point& rhs) const {
        return x == rhs.x ? y < rhs.y : x < rhs.x;
    } // a < b
    Point operator - (const Point& rhs) const {
        return Point(x - rhs.x, y - rhs.y);
    } // a - b = Point(-1, 1)
} P[MAX_N];

double CrossProduct(Point P1, Point P2) {
    return P1.x * P2.y - P1.y * P2.x;
} // double ret = CrossProduct(a, b); // ret = -5
// double DotProduct(..., ...) { ... }
```

极角排序

以最左下角点为源点，将其余点按照优先角度，其次距离从小到大排序

最左下角点：x最小，其次y最小

```
1  // find the bottom left corner
2  int id = 0;
3  for (int i = 1; i < n; ++i) if (P[i] < P[id]) id = i;
4  Point left_corner = P[id];
5
6  // get distance function
7  double Dis(Point a, Point b) {
8      double tx = a.x - b.x, ty = a.y - b.y;
9      return sqrt(tx * tx + ty * ty);
10 }
11
12 // compare function
13 bool cmp(Point a, Point b) {
14     double ret = Cross(a - left_corner, b - left_corner);
15     if (sgn(ret) != 0) return ret > 0;
16     else return Dis(a, left_corner) < Dis(b, left_corner);
17 }
18
19 sort(P, P + n, cmp);
```

凸包(Convex Hull)

平面凸包：
对一个简单多边形来说，如果给定其边界上或内部的任意两个点，连接这两个点的线段上的所有点都被包含在该多边形的边界上或内部的话，则该多边形为凸多边形。

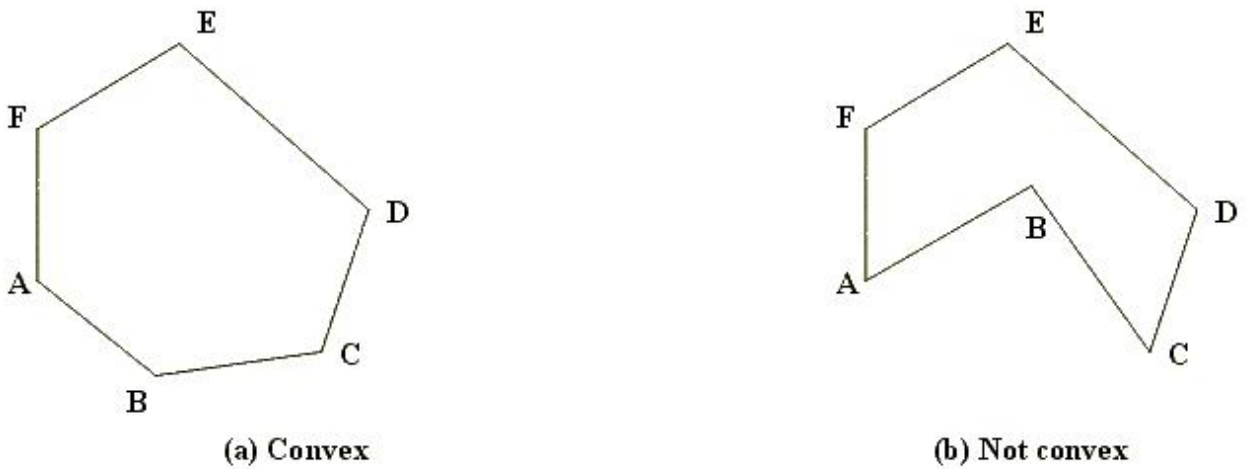


Figure 1: Closed polygons

三角形？矩形？五角星？

Andrew扫描法

由Graham扫描法改进得到，大致步骤：

1. 按优先x升序，其次y升序排序$\rightarrow O(n \log n)$
2. 从$0 \sim n-1$构造下凸包$\rightarrow O(n)$
3. 从$n-2 \sim 0$构造上凸包$\rightarrow O(n)$

总的时间复杂度是：$O(n \log n)$

Step One????

上、下凸包的构造：

- 由向量(栈顶，栈中栈顶的下一个元素)和向量(栈顶，当前点P[i])的相对位置关系决定是否弹栈
- 当前点P[i]进栈

Andrew扫描法

由Graham扫描法改进得到，大致步骤：

1. 按优先x升序，其次y升序排序$\rightarrow O(n \log n)$
2. 从$0 \sim n-1$构造下凸包$\rightarrow O(n)$
3. 从$n-2 \sim 0$构造上凸包$\rightarrow O(n)$

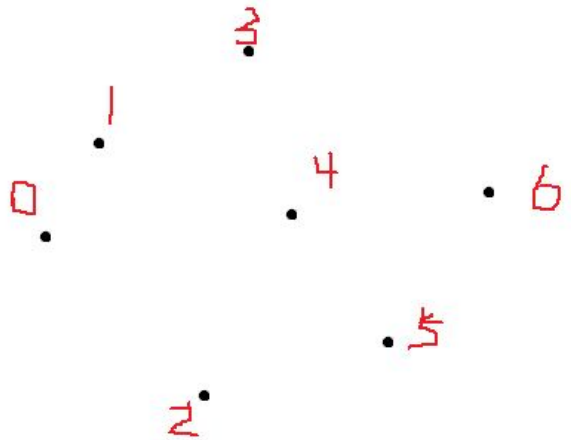
总的时间复杂度是：$O(n \log n)$

Step One????

上、下凸包的构造：

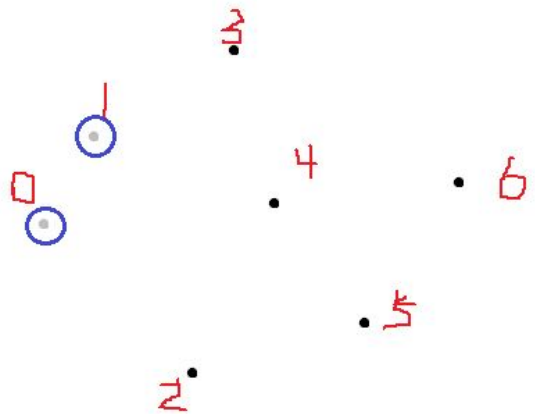
- 由向量(栈顶，栈中栈顶的下一个元素)和向量(栈顶，当前点P[i])的相对位置关系决定是否弹栈
- 当前点P[i]进栈

Andrew扫描法



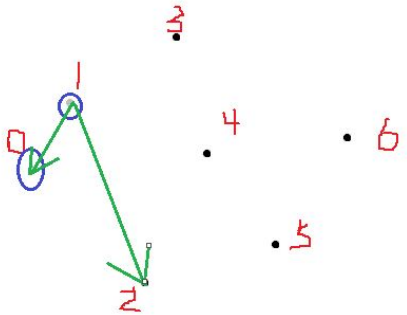

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



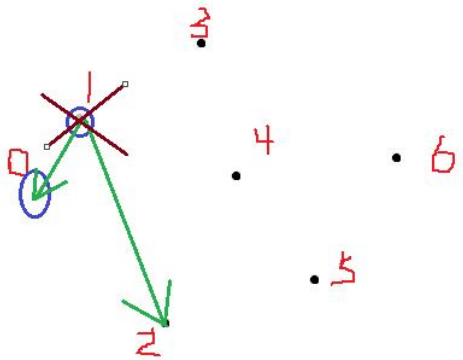

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



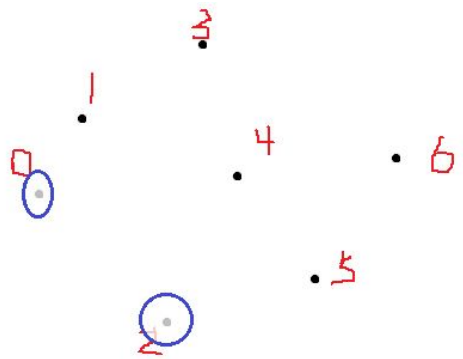

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



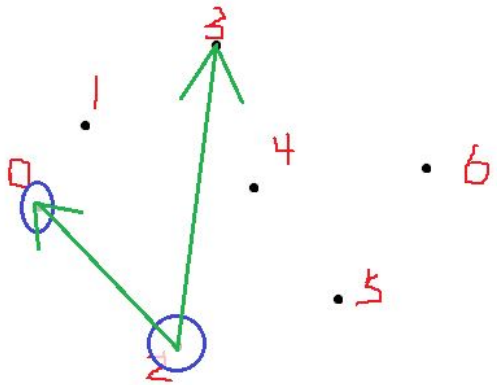

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



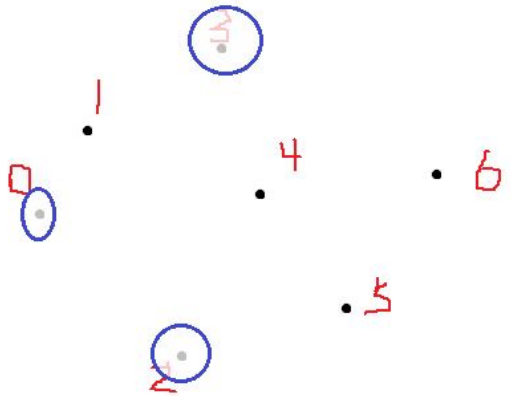

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



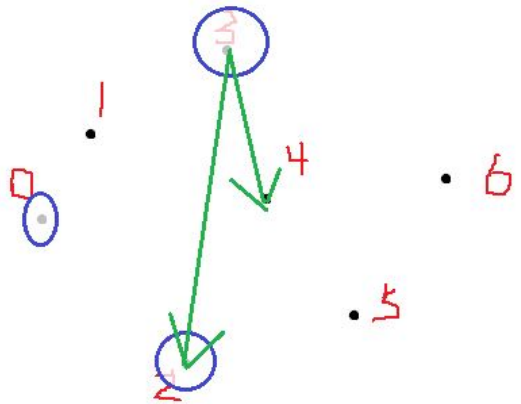

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



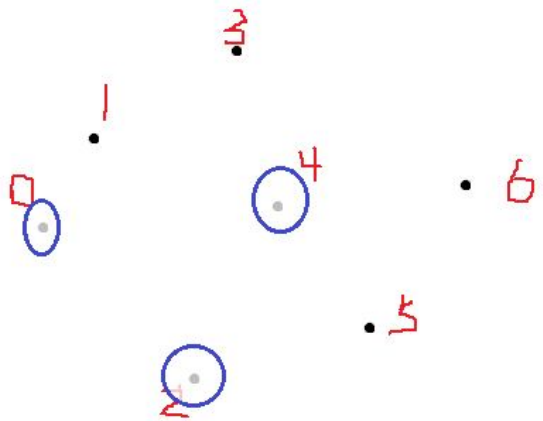

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



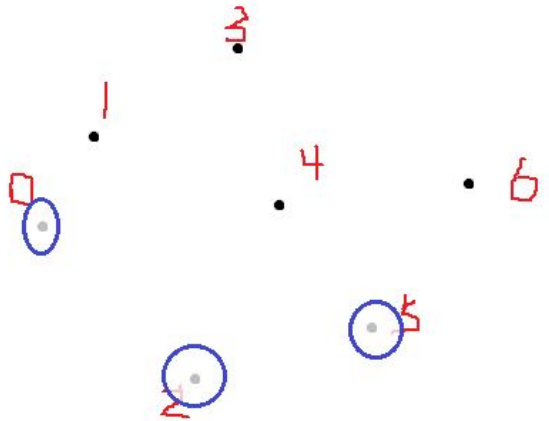

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



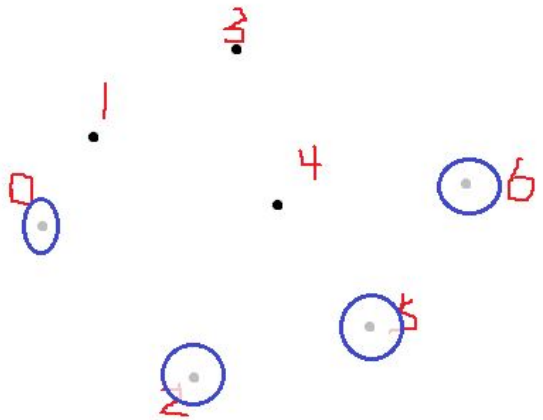

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



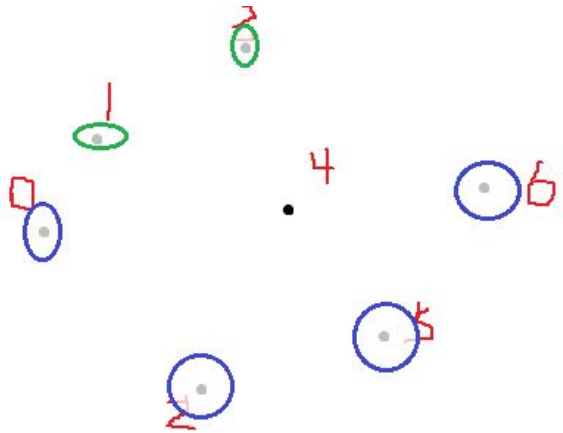

```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法



```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Andrew扫描法




```
int tot = 0; // convex hull vertex numbers
for (int i = 0; i < n; ++i) { // below convex hull
    while (tot >= 2 && // first two is always right
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
int m = tot;
for (int i = n - 2; i >= 0; --i) { // above convex hull
    while (tot > m && // protect below convex hull
        Cross(V[tot - 1] - V[tot], P[i] - V[tot]) >= 0) --tot;
    V[++tot] = P[i];
}
if (tot > 2) --tot; // P[0] is duplicated
```

Problems

CF 641C

给出一个圆心和$n \leq 10^5$边形的多边形的n个顶点，计算这个n边形绕圆心旋转所扫过的面积。已知圆心一定在多边形外面。

只要计算圆心到多边形顶点及边的最大距离和最小距离即可
最大距离一定在顶点位置，最小距离可能在顶点上也可能在边上
用余弦定理判断圆心到边的垂心是否在边上

LightOJ 1203

给出$n \leq 10^5$个点，找到一个点使从这个点看其他所有点所形成的最大夹角最小（视野内能看完其他所有点）。

符合条件的点一定在凸包顶点上
枚举凸包顶点，叉积点积求夹角

```
1 fabs(atan2(fabs((a - p).cross(b - p)), (a - p).dot(b - p)));
```

Problems

CF 641C

给出一个圆心和$n \leq 10^5$边形的多边形的n个顶点，计算这个n边形绕圆心旋转所扫过的面积。已知圆心一定在多边形外面。

只要计算圆心到多边形顶点及边的最大距离和最小距离即可
最大距离一定在顶点位置，最小距离可能在顶点上也可能在边上
用余弦定理判断圆心到边的垂心是否在边上

LightOJ 1203

给出$n \leq 10^5$个点，找到一个点使从这个点看其他所有点所形成的最大夹角最小（视野内能看完其他所有点）。

符合条件的点一定在凸包顶点上
枚举凸包顶点，叉积点积求夹角

```
1 fabs(atan2(fabs((a - p).cross(b - p)), (a - p).dot(b - p)));
```

Problems

CF 641C

给出一个圆心和$n \leq 10^5$边形的多边形的n个顶点，计算这个n边形绕圆心旋转所扫过的面积。已知圆心一定在多边形外面。

只要计算圆心到多边形顶点及边的最大距离和最小距离即可
最大距离一定在顶点位置，最小距离可能在顶点上也可能在边上
用余弦定理判断圆心到边的垂心是否在边上

LightOJ 1203

给出$n \leq 10^5$个点，找到一个点使从这个点看其他所有点所形成的最大夹角最小（视野内能看完其他所有点）。

符合条件的点一定在凸包顶点上
枚举凸包顶点，叉积点积求夹角

```
1 fabs(atan2(fabs((a - p).cross(b - p)), (a - p).dot(b - p)));
```

Problems

CF 641C

给出一个圆心和$n \leq 10^5$边形的多边形的n个顶点，计算这个n边形绕圆心旋转所扫过的面积。已知圆心一定在多边形外面。

只要计算圆心到多边形顶点及边的最大距离和最小距离即可
最大距离一定在顶点位置，最小距离可能在顶点上也可能在边上
用余弦定理判断圆心到边的垂心是否在边上

LightOJ 1203

给出$n \leq 10^5$个点，找到一个点使从这个点看其他所有点所形成的最大夹角最小（视野内能看完其他所有点）。

符合条件的点一定在凸包顶点上
枚举凸包顶点，叉积点积求夹角

```
1 fabs(atan2(fabs((a - p).cross(b - p)), (a - p).dot(b - p)));
```

Your ACM


寻找着，那个Wrong Answer的你。
ACM监督 最新作
君のACM。
Your ACM-ICPC
意例表律气力板D
题样打规运暴模C
猜过先看靠瞎套G
会能来般学何顿会
只只上一数几一只
拟心学P合算论论
模贪数D组计图数