# 凸包

# Convex Hull

## 水平序 Graham 扫描法

**Idea**：首先将所有的点以 $x$ 为第一关键字、$y$ 为第二关键字排序，然后分上下凸包分别求解：求解下凸包时，维护一个栈存储当前凸包的点，顺序扫描每个点，每扫描到一个点便判断其与栈内前两个点的转向关系，若成左转关系，则该点入栈，否则弹出栈顶元素继续判断，直至判断到头或形成左转关系为止；同理，逆序扫描点求解上凸包。

**Complexity**：$O(n\lg n)$ （瓶颈在于排序的复杂度，若对于特殊情况采用基数排序可优化复杂度）

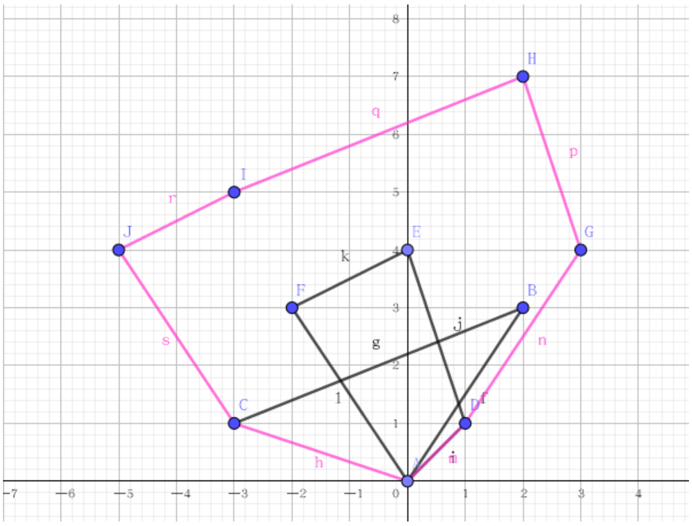
**Code**：

```
void ConvexHull(int n, Point p[], Point sta[], int &staid){
    // there're n points stored in p[], the points on convex hull will be saved in sta[]
    sort(p+1, p+n+1);
    n = unique(p+1, p+n+1) - (p+1);
    staid = 0;
    for(int i = 1; i <= n; i++){
//      while(staid > 1 && sgn((sta[staid]-sta[staid-1]) ^ (p[i]-sta[staid-1])) < 0) staid--;  // points on edge
        while(staid > 1 && sgn((sta[staid]-sta[staid-1]) ^ (p[i]-sta[staid-1])) <= 0) staid--; // no points on edge
        sta[++staid] = p[i];
    }
    int k = staid;
    for(int i = n-1; i >= 1; i--){
//      while(staid > k && sgn((sta[staid]-sta[staid-1]) ^ (p[i]-sta[staid-1])) < 0) staid--;  // points on edge
        while(staid > k && sgn((sta[staid]-sta[staid-1]) ^ (p[i]-sta[staid-1])) <= 0) staid--; // no points on edge
        sta[++staid] = p[i];
    }
    if(n > 1)   staid--;
}
```

## Minkowski 和

**Definition**：两个图形 $A,B$ 的 **Minkowski** 和定义为：$C=\{a+b\mid a\in A,b\in B\}$.

对于凸包，两凸包的 **Minkowski** 和的凸包是由原凸包的边构成的：



故将原边按极角排序后求一次凸包即可。

**Code**：

```
void Minkowski(int n1, Point p1[], int n2, Point p2[], Point tmp[], Point res[], int &resn){
    // tmp[] is an auxiliary array
    // p1[] is a convex hull consist of n1 points
    // p2[] is a convex hull consist of n2 points
    // res[] is the Minkowski tmp of these two convex hull consist of resn points
    p1[n1+1] = p1[1], p2[n2+1] = p2[1];
    vector<Vector> v1, v2;
    for(int i = 1; i <= n1; i++)    v1.emplace_back(p1[i+1] - p1[i]);
    for(int i = 1; i <= n2; i++)    v2.emplace_back(p2[i+1] - p2[i]);
    int pt1 = 0, pt2 = 0, tid = 1;
    tmp[1] = p1[1] + p2[1];
    while(pt1 < n1 && pt2 < n2){
        tid++;
        if(sgn(v1[pt1] ^ v2[pt2]) >= 0) tmp[tid] = tmp[tid-1] + v1[pt1++];
        else                            tmp[tid] = tmp[tid-1] + v2[pt2++];
    }
    while(pt1 < n1) tid++, tmp[tid] = tmp[tid-1] + v1[pt1++];
    while(pt2 < n2) tid++, tmp[tid] = tmp[tid-1] + v2[pt2++];
    ConvexHull(tid, tmp, res, resn);
}
```