Contents

1 計算幾何

1.1 基本儲存

```
struct Pt{
    double x, y;
};
struct Line{
    Pt st, ed;
};
struct Circle{
    Pt o; // 圓心
    double r;
};
struct Poly{
    int n; // n邊形
    vector<Pt> pts;
};
```

1.2 距離

歐基里德距離

$$d(p_1, p_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

曼哈頓距離

$$d(p_1, p_2) = |x_1 - x_2| + |y_1 - y_2|$$

1.3 內積、外積

$$\vec{v_1} \cdot \vec{v_2} = x_1x_2 + y_1y_2$$
$$\vec{v_1} \times \vec{v_2} = x_1y_2 - x_2y_1$$

1.4 多邊形面積

$$\frac{1}{2}|\sum_{i=1}^{n} \overrightarrow{OP_i} \times \overrightarrow{OP_{i+1}}|$$

1.5 判斷點是否在線段上

```
bool collinearity(Pt p1, Pt p2, Pt p3){ // 三點共線
    return cross(p1, p2) * cross(p1, p3) == 0;
}

bool inLine(Pt a, Pt b, Pt p){ // 點是否在線上
    return collinearity(a, b, p) && dot(a - p, b - p)
        <= 0;
}
```

1.6 線段相交、交點

```
bool intersect(Pt a, Pt b, Pt c,  d){ // 線段相交
    return (cross(b - a, c - a) * cross(b - a, d - a) <
         0 && cross(d - c, a - c) * cross(d - c, b - c)
         < 0)
            || inLine(a, b, c) || inLine(a, b, d) ||
                inLine(c, d, a) || inLine(c, d, b);
}
Pt intersection(Pt a, Pt b, Pt c, Pt d){ // 線段交點
    assert(intersect(a, b, c, d)); // 沒有交點的狀況
    return a + cross(a - c, d - c) * (b - a) / cross(d
        - c, b - a);
}
```

1.7 點在多邊形內部

射線法：若點在多邊形內，則隨機選一個方向的射線出現會碰到奇數次邊而如果碰到多邊形的點，如果射線碰到多邊形的點則重選（需要特判點是否在多邊形的邊或頂點上）

1.8 凸包

```
  vector<Pt> convex_hull(vector<Pt> hull){
    sort(hull.begin(),hull.end());
    int top=0;
    vector<Pt> stk;
    for(int i=0;i<hull.size();i++){
        while(top>=2&&cross(stk[top-2],stk[top-1],hull[
            i])<=0)
            stk.pop_back(),top--;
        stk.push_back(hull[i]);
        top++;
    }
    for(int i=hull.size()-2,t=top+1;i>=0;i--){
        while(top>=t&&cross(stk[top-2],stk[top-1],hull[
            i])<=0)
            stk.pop_back(),top--;
        stk.push_back(hull[i]);
        top++;
    }
    stk.pop_back();
    return stk;
}
```

1.9 旋轉卡尺-最遠點對

```
double FarthestPair(vector<Pt> arr){ // 需要先凸包
    double ret=0;
    for(int i = 0, j = i+1; i<arr.size(); i++){
        while(distance(arr[i], arr[j]) <= distance(arr[
            i], arr[(j+1)%arr.size()]) ){
            j = (j+1) % arr.size();
        }
        ret = max(ret, distance(arr[i],arr[j]));
    }
    return ret;
}
```

1.10 極角排序

```
bool cmp(const Pt& lhs, const Pt rhs){
    if((lhs < Pt(0, 0)) ^ (rhs < Pt(0, 0)))
        return (lhs < Pt(0, 0)) < (rhs < Pt(0, 0));
    return (lhs ^ rhs) > 0;
} // 從 270 度開始逆時針排序
sort(P.begin(), P.end(), cmp);
```

1.11 皮克定理（多邊形內整數點數量）

$$A = i + \frac{b}{2} - 1$$

A: 多邊形面積 i: 內部整數點個數 b: 線上整數點個數

1.12 三分搜-最小包覆圓

平面上給 n 個點，求出半徑最小的圓要包住所有的點。求出圓心位置與與最小半徑。複雜度（$N\log^2 N$）

```
Pt arr[MXN];
double checky(double x, double y) {
  double cmax = 0;
  for(int i = 0; i < n; i++) {
    cmax = max(cmax,(arr[i].x - x) * (arr[i].x - x) +
                    (arr[i].y - y) * (arr[i].y - y));
  }// 過程中回傳距離^2 避免不必要的根號運算
  return cmax;
}
double checkx(double x){
    double yl = -1e9, yr = 1e9;
    while(yr - yl > EPS) {
        double ml = (yl+yl+yr) / 3, mr = (yl+yr+yr) /
            3;
        if (checky(x, ml) < checky(x, mr))    yr = mr;

        else                                  yl = ml;
    }
}
double xl = -1e9, xr = 1e9;
while(xr - xl > EPS) {
```

```
  double ml = (xl+xl+xr) / 3, mr = (xl+xr+xr) / 3;
  if (checkx(ml) < checkx(mr))    xr = mr;
  else                            xl = ml;
}
```

2 樹論

2.1 LCA

```
int timing;
int in[N],out[N];
void dfs(int u){
    in[u] = ++timing;//這時進入u
    for(int nxt : g[u])//跑過所有孩子
        dfs(nxt);
    out[u] = ++timing;//這時離開u
}
bool is_ancestor(int u,int v){ //用=因為自己是自己的祖
    先
    return in[u] <= in[v] && out[u] >= out[v]; //u是v的
        祖先
}
int getlca(int x, int y){
    if(is_ancestor(x, y))return x; // 如果 u 為 v 的祖
        先則 lca 為 u
    if(is_ancestor(y, x))return y; // 如果 v 為 u 的祖
        先則 lca 為 u
    for(int i=logN;i>=0;i--){    // 判斷 2^logN, 2^(
        logN-1),...2^1, 2^0 倍祖先
        if(!is_ancestor(anc[x][i], y)) // 如果 2^i 倍祖
            先不是 v 的祖先
            x = anc[x][i];            // 則往上移動
    }
    return anc[x][0]; // 回傳此點的父節點即為答案
}
int anc[N][logN]; //倍增法，從x往上走i步
signed main(){
    for(int i=1;i<=log2(N);i++){
        for(int now=1;now<=N;now++){
            anc[now][i]=anc[anc[now][i-1]][i-1];
        }
    }
}
```

2.2 換根 DP

```
void dfs(int u, int fa) {  // 預處裡dfs
  sz[u] = 1; // 以 u 為根的子樹數量
  dep[u] = dep[fa] + 1; // u 的深度
  for (int v : edge[u]) { //遍歷 u 的子節點
    if (v != fa) { //不等於父親
      dfs(v, u);
      sz[u] += sz[v];
    }
  }
}
void get_ans(int u, int fa) {  // 第二次dfs換根dp
  for (int v : edge[u]) { //遍歷子節點
    if (v != fa) {
      dp[v] = dp[u] - sz[v] * 2 + n; //轉移式
      get_ans(v, u);
    }
  }
}
```

3 資料結構

3.1 線段樹

```
#define cl(x) (x<<1)
#define cr(x) (x<<1)+1
int seg[N*4], arr[N], tag{N*4};
void build(int id,int l,int r){
    if(l==r){
        seg[id]=arr[l];
        return ;
    }
    int mid=(l+r)>>1;
    build(cl(id),l,mid);
    build(cr(id),mid+1,r);
    pull(id);
}
void push(int i, int l, int r) {
    if(tag[i]) {
        seg[i] += tag[i] * (r - l + 1); // 更新區間 [l,
            r]
        if(l != r) {                    // 把標記往下推
            tag[cl(i)] += tag[i];
            tag[cr(i)] += tag[i];
        }
        tag[i] = 0;                     // 更新完區間
            後，把標記歸 0
    }
}
void pull(int i, int l, int r) {
    int mid = (l+r)>>1;
    push(cl(i), l, mid);
    push(cr(i), mid + 1, r);
    seg[i] = seg[cl(i)] + seg[cr(i)];
}
int query(int i, int l, int r, int ql, int qr) {
    push(i, l, r);
    if(nl <= l && r <= nr)
        return seg[i];
    int mid = (l + r) / 2, ret = 0;
    if(ql <= mid)
        ret += query(cl(i), l, mid, ql, qr);
    if(qr >  mid)
        ret += query(cr(i), mid + 1, r, ql, qr);
    return ret;
}
void update(int i, int l, int r, int ql, int qr, int v)
    {
    push(i, l, r); // 懶標下推
    if(ql <= l && r <= qr) {
        tag[i] += v;
        return;
    }
    int mid = (l + r)>>1;
    if(ql <= mid)
        update(cl(i), l, mid, ql, qr, v);
    if(qr > mid)
        update(cr(i), mid + 1, r, ql, qr, v);
    pull(i, l, r); // 更新當前區間的總和
}
```

4 一些題目

4.1 最大子矩形-玉蟾宮

```
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) { //初始化
      l[j] = r[j] = j;
    }
    char c;
    for (int j = 1; j <= m; j++) { //對每一個直行做統
        計，若是上一個a[j]也是1則會變成2
      cin>>c;
      if (c == 'F') a[j]++;
      else if (c == 'R') a[j] = 0;
    }
    for (int j = 1; j <= m; j++)
      while (l[j] != 1 && a[l[j] - 1] >= a[j]) l[j] = l
          [l[j] - 1];
    for (int j = m; j >= 1; j--)
      while (r[j] != m && a[r[j] + 1] >= a[j]) r[j] = r
          [r[j] + 1];
    for (int j = 1; j <= m; j++) ans = max(ans, (r[j] -
        l[j] + 1) * a[j]);
  }
```