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

$$\vec{v_1} \cdot \vec{v_2} = x_1 x_2 + y_1 y_2$$

$$\vec{v_1} \times \vec{v_2} = x_1 y_2 - x_2 y_1$$

1.4 多邊形面積

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

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(p2 - p1, p3 - p1) == 0;
}
bool inLine(Line li, Pt p){ // 點是否在線上
    return collinearity(li.st, li.ed, p) && dot(li.st -
        p, li.ed - 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;
}
```

1.13 旋轉矩陣、鏡射矩陣

逆時針轉 θ 角

$$\begin{bmatrix} cos\theta & -sin\theta \\ sin\theta & cos\theta \end{bmatrix}$$

對與 x 軸正向夾角為 θ 的直線 L 鏡射

$$\begin{bmatrix} cos2\theta & sin2\theta \\ sin2\theta & -cos2\theta \end{bmatrix}$$

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 數論

4.1 階乘與模逆元

```
long long fac[MXN], inv[MXN];
fac[0] = 1; // 0! = 1
for(long long i = 1; i <= N; i++)
    fac[i] = fac[i-1] * i % MOD;
inv[N] = FastPow(fac[N], MOD-2); // 快速冪
for(long long i = N-1; i >=0; i--)
    inv[i] = inv[i+1] * (i+1) % MOD;
```

4.2 擴展歐基里德

```
int exgcd(int a,int b,long long &x,long long &y) {
    if(b == 0){x=1,y=0;return a;}
    int now=exgcd(b,a%b,y,x);
    y-=a/b*x;
    return now;
}
long long inv(long long a,long long m){ //求模逆元
    long long x,y;
    long long d=exgcd(a,m,x,y);
    if(d==1) return (x+m)%m;
    else return -1; //-1為無解
}
```

4.3 中國剩餘定理

```
LL exgcd(LL a,LL b,LL &x,LL &y){
    if(!b){
        x = 1, y = 0;
        return a;
    }
    int now=exgcd(b, a % b, y, x);
    y -= a / b * x;
    return now;
}
LL CRT(LL k, LL* a, LL* r) {
    LL n = 1, ans = 0;
    for (LL i = 1; i <= k; i++) {
        n = n * r[i];
    }
    for (LL i = 1; i <= k; i++) {
        LL m = n / r[i], b, y;
        exgcd(m, r[i], b, y);
        ans = (ans + a[i] * m * b % n) % n;
    }
    return (ans % n + n) % n;
}
```

4.4 進制轉換

```
int ntoi(string str, int n){ // n進制轉10進制
    int ans = 0;
    for(int i = 0; i < str.size(); i++){
        if(str[i] >= '0' && str[i]<='9')
            ans = ans * n + str[i] - '0';
        else// 小寫減a 大寫減A
            ans = ans * n + str[i] - 'a' + 10;
    }
    return ans;
}
string iton(int num , int n){ // 10進制轉n進制
    string ans = "";
    do{
        int t = num % n;
        if(t >= 0 && t <= 9)
            ans += t + '0';
        else
            ans += t - 10 + 'a';
        num /= n;
    } while(num != 0);
    reverse(ans.begin(), ans.end());
    return ans;
}
```

5 圖論

5.1 最短路徑

dijkstra $O(V^2+E)$

```
vector<pair<int,int>>vec[N];
void dijkstra(int s,int t){//起點，終點
    int dis[N];
    for(int i=0;i<N;i++){//初始化
        dis[i]=INF;//值要設為比可能的最短路徑權重還要大
            的值
    }
    dis[s]=0;
    priority_queue<pii,vector<pii>,greater<pii>>pq;//以
        小到大排序
    pq.push({dis[s],s});
    while(pq.empty()==0){
        int u=pq.top().second;
        pq.pop();
        if(vis[u])continue;
        vis[u]=1;
        for(auto [v,w]:vec[u]){
            if(dis[u]+w<dis[v]){//鬆弛
                dis[v]=dis[u]+w;
                pq.push({dis[v],v});
            }
        }
    }
}
```

floyd-warshall $O(N^3)$

```
for(int k=1;k<=N;k++){//窮舉中繼點k
    for(int i=1;i<=N;i++){
        for(int j=1;j<=N;j++){//窮舉點對(i,j)
            dis[i][j]=min(dis[i][j],dis[i][k]+dis[k][j
                ]);
        }
    }
}
```

5.2 歐拉回路、漢米爾頓路徑

```
vector<int> path;
void dfs(int x){
    while(!edge[x].empty()){
        int u = edge[x].back();
        edge[x].pop_back();
        dfs(u);
    }
    path.push_back(x);
}
int main(){
    dfs(st);
    reverse(path.begin(),path.end());
    for(int i:path)    cout<<i<<' ';
    cout<<endl;
}
```

```
dp[3][26]=dp[3][11010] //現在的點為3，走過1,3,4這三個點
if( edge[i][j] && ( (1<<j) & s ) == 0 ){
    //i->j有邊且點j尚未走過
    dp[j][s|(1<<j)]=dp[i][s];
}
//以下為程式碼
for(int s=0;s<(1<<n);s++){//枚舉點集合
    for(int i=0;i<n;i++){//枚舉現在的點
        if(s&(1<<i)==0)continue;
        for(int j=0;j<n;j++){//枚舉下一個點
            if(i==j)continue;
            if( edge[i][j] && ( (1<<j) & s ) == 0 ){
                dp[j][s|(1<<j)]=dp[i][s];
            }
        }
    }
}
```

6 動態規劃

6.1 0/1 背包

$O(NW)$

```
for (int i = 1; i <= cnt; i++) //幾個物品
  for (int j = weight; j >= w[i]; j--) //從物品耐重上限
      枚舉到此物品的重量，代表每個都最多選一次
    dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
```

6.2 無限背包

$O(NW)$

```cpp
for(int i = 1; i <= cnt; i++)
    for(int j = w[i]; j <= weight; j++)
        dp[j] = max(dp[j], dp[j - w[i]] + v[i]);
```

6.3 有限背包

$O(NW \log k)$

```cpp
// 有限背包二進制拆分
int index = 0;
for(int i = 1; i <= m; i++){
    int c = 1, p, h, k;
    cin >> p >> h >> k;
    while(k > c){
        k -= c;
        list[++index].w = c * p;
        list[index].v = c * h;
        c *= 2;
    }
    list[++index].w = p * k;
    list[index].v = h * k;
}
// 之後再去做0/1背包
```

7 字串

7.1 字典樹

```cpp
struct trie{
    struct node{
        node *nxt[26];
        int cnt, sz;
        node():cnt(0),sz(0){
            memset(nxt,0,sizeof(nxt));
        }
    };
    node *root;
    void init(){root = new node();}
    void insert(const string& s){
        node *now = root;
        for(auto i:s){
            now->sz++;
            if(now->nxt[i-'a'] == NULL){
                now->nxt[i-'a'] = new node();
            }
            now = now->nxt[i-'a'];
        }
        now->cnt++;
        now->sz++;
    }
};
```

7.2 KMP

```cpp
int failure[MXN]; //儲存以第i個為結尾的次長相同前綴後綴
vector<int> KMP(string& t, string& p){
    vector<int> ret;
    if (p.size() > t.size()) return;
    for (int i=1, j=failure[0]=-1; i<p.size(); ++i){
        while (j >= 0 && p[j+1] != p[i]) //當不相同無法匹配
            j = failure[j];
        if (p[j+1] == p[i]) j++;//如果可以加長長度，則為前一個答案+1
        failure[i] = j; // 紀錄答案
    }
    // i 為字串 t 當前 index, j 為字串 p 以匹配到的 index
    for (int i=0, j=-1; i<t.size(); ++i){
        while (j >= 0 && p[j+1] != t[i])
            j = failure[j]; // 當匹配失敗找到次長相同前綴後綴，移動字串W
        if (p[j+1] == t[i]) j++; // 當字元相等
        if (j == p.size()-1){ //  當字串完全匹配
            ret.push_bck( i - p.size() + 1 ); // 加進答案
            j = failure[j]; // 移動字串W
        }
    }
    return ret;
}
```

7.3 Hash

```cpp
// 如果發生碰撞可以做雙重雜湊
const ll P = 75577; // p = 13331, 14341, 75577
const ll MOD = 998244353; //MOD = 1e9+7, 998244353, 999997771
ll Hash[MXN];    //Hash[i] 為字串 [0,i] 的 hash值
void build(const string& s){
    int val = 0;
    for(int i=0; i<s.size(); i++){
        val = (val * P + s[i]) % MOD;
        Hash[i] = val;
    }
}
// h[l, r] = h[r] - h[l-1] * p^(r-l+1)
```

8 一些題目

8.1 最大子矩形-玉蟾宮

```cpp
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]);
  }
```

8.2 次小生成樹

先求出 MST 之後，窮舉每條不在 MST 上的邊放上去，而會形成環把環上除了新的邊中權重最大的邊移除，判斷是否為答案

9 其他

```
                  (__)     (__) ____/
              /| (oo) _   (oo)/----/_____      *
  _o\______/_|\_\/_/_|__\/|____|//////== *- *   * -
 /__________   \    AC |   AC | NO BUG /== -*
[_____/^^\_____\_____|_____/^^\_____]      *- * -
      \__/                   \__/        Chong~Chong~Chong~
```

```

/ ￣￣￣＼
|  ー  ー \
|   Q  Q   |    / ￣￣￣￣￣￣￣＼
\     ^    /  <    我好想要鍵盤    \
 ＼      \     \                 /
 ／       ＼    \_______________/
/  |        \ \
|  |         | |
```