Contents

1 計算幾何

1.1 基本儲存

```
typedef long double ld;
const ld eps = 1e-8;
int dcmp(ld x) {
  if(abs(x) < eps) return 0;
  else return x < 0 ? -1 : 1;
}
struct Pt {
  ld x, y;
  Pt(ld _x=0, ld _y=0):x(_x), y(_y) {}
  Pt operator+(const Pt &a) const {
    return Pt(x+a.x, y+a.y);  }
  Pt operator-(const Pt &a) const {
    return Pt(x-a.x, y-a.y);  }
  Pt operator*(const ld &a) const {
    return Pt(x*a, y*a);  }
  Pt operator/(const ld &a) const {
    return Pt(x/a, y/a);  }
  ld operator*(const Pt &a) const { //dot
    return x*a.x + y*a.y;  }
  ld operator^(const Pt &a) const { //cross
    return x*a.y - y*a.x;  }
  bool operator<(const Pt &a) const {
    return x < a.x || (x == a.x && y < a.y); }
  bool operator>(const Pt &a) const {
    return x > a.x || (x == a.x && y > a.y); }
  bool operator==(const Pt &a) const {
    return dcmp(x-a.x) == 0 && dcmp(y-a.y) == 0;  }
  friend ld cross(Pt a, Pt b, Pt c){
     return (c-a)^(c-b);}
};
ld norm2(const Pt &a) {
  return a*a; }
ld norm(const Pt &a) {
  return sqrt(norm2(a)); }
Pt perp(const Pt &a) {
  return Pt(-a.y, a.x); }
Pt rotate(const Pt &a, ld ang) {
  return Pt(a.x*cos(ang)-a.y*sin(ang), a.x*sin(ang)+a.y*cos(ang)); }
struct Line {
  Pt s, e, v; // start, end, end-start
  ld ang;
  Line(Pt _s=Pt(0, 0), Pt _e=Pt(0, 0)):s(_s), e(_e) { v = e-s; ang = atan2(v.y, v.x); }
  bool operator<(const Line &L) const {
    return ang < L.ang;
} };
struct Circle {
  Pt o; ld r;
  Circle(Pt _o=Pt(0, 0), ld _r=0):o(_o), r(_r) {}
};
```

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

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

1.5 點與線段距離

看點與線段兩端內積，若為負數則說明角度大於 90 度則距離為點到該端點的距離若內積皆 >=0，則距離為三角形面積/線段

1.6 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(Pt st, Pt ed, Pt p){ // 點是否在線上
    return collinearity(st, ed, p) && dot(p, st, ed) < 0;
}
```

1.7 線段相交、交點

```
bool intersect(Pt a, Pt b, Pt c, Pt 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);
}
```

1.8 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）

1.9 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;
}
```

1.10 凸包技巧

```
struct Convex {
    #define all(x) x.begin(), x.end()
    int n;
    vector<Pt> A, V, L, U;
    Convex(const vector<Pt> &_A) : A(_A), n(_A.size())
        { // n >= 3
        auto it = max_element(all(A));
        L.assign(A.begin(), it + 1);
        U.assign(it, A.end()), U.push_back(A[0]);
        for (int i = 0; i < n; i++) {
            V.push_back(A[(i + 1) % n] - A[i]);
        }
    }
    int PtSide(Pt p, Line L) {
        return dcmp((L.e - L.s)^(p - L.s));
    }
    int inside(Pt p, const vector<Pt> &h, auto f) {
        auto it = lower_bound(all(h), p, f);
        if (it == h.end()) return 0;
        if (it == h.begin()) return p == *it;
        return 1 - dcmp((p - *prev(it))^(*it  - *prev(
            it)));
    }
    // 1. whether a given point is inside the CH
    // ret 0: out, 1: on, 2: in
    int inside(Pt p) {
        return min(inside(p, L, less{}), inside(p, U,
            greater{}));
    }
    static bool cmp(Pt a, Pt b) { return dcmp(a ^ b) >
        0; }
    // 2. Find tangent points of a given vector
    // ret the idx of far/closer tangent point
    int tangent(Pt v, bool close = true) {
        assert(v != Pt{});
        auto l = V.begin(), r = V.begin() + L.size() -
            1;
        if (v < Pt{}) l = r, r = V.end();
        if (close) return (lower_bound(l, r, v, cmp) -
            V.begin()) % n;
        return (upper_bound(l, r, v, cmp) - V.begin())
            % n;
    }
    // 3. Find 2 tang pts on CH of a given outside
        point
    // return index of tangent points
    // return {-1, -1} if inside CH
    array<int, 2> tangent2(Pt p) {
        array<int, 2> t{-1, -1};
        if (inside(p) == 2) return t;
        if (auto it = lower_bound(all(L), p); it != L.
            end() and p == *it) {
            int s = it - L.begin();
            return {(s + 1) % n, (s - 1 + n) % n};
        }
        if (auto it = lower_bound(all(U), p, greater{})
            ; it != U.end() and p == *it) {
            int s = it - U.begin() + L.size() - 1;
            return {(s + 1) % n, (s - 1 + n) % n};
        }
        for (int i = 0; i != t[0]; i = tangent((A[t[0]
            = i] - p), 0));
        for (int i = 0; i != t[1]; i = tangent((p - A[t
            [1] = i]), 1));
        return t;
    }
    int find(int l, int r, Line L) {
        if (r < l) r += n;
        int s = PtSide(A[l % n], L);
        return *ranges::partition_point(views::iota(l,
            r),
            [&](int m) {
                return PtSide(A[m % n], L) == s;
            }) - 1;
    };
    // 4. Find intersection point of a given line
    // intersection is on edge (i, next(i))
    vector<int> intersect(Line L) {
        int l = tangent(L.s - L.e), r = tangent(L.e - L
            .s);
        if(PtSide(A[l], L) == 0)    return {l};
        if(PtSide(A[r], L) == 0)    return {r};
        if (PtSide(A[l], L) * PtSide(A[r], L) > 0)
            return {};
        return {find(l, r, L) % n, find(r, l, L) % n};
    }
};
```

1.11 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;
}
```

1.12 圓覆蓋面積

```
//init(int _c): t總共_c個圓
//Circle c[N]: 輸入圓心&半徑
//sovle()
//Area[i]: 至少i個圓覆蓋的面積
#define N 1021
#define D long double
struct CircleCover{//O(N^2logN)
  int C; Circle c[ N ]; //填入C(圓數量),c(圓陣列)
  bool g[ N ][ N ], overlap[ N ][ N ];
  // Area[i] : area covered by at least i circles
  D Area[ N ];
  void init( int _C ){ C = _C; }
  bool CCinter( Circle& a , Circle& b , Pt& p1 , Pt& p2
       ){
    Pt o1 = a.o , o2 = b.o;
    D r1 = a.r , r2 = b.r;
    if( norm( o1 - o2 ) > r1 + r2 ) return {};
    if( norm( o1 - o2 ) < max(r1, r2) - min(r1, r2) )
        return {};
    D d2 = ( o1 - o2 ) * ( o1 - o2 );
    D d = sqrt(d2);
    if( d > r1 + r2 ) return false;
    Pt u=(o1+o2)*0.5 + (o1-o2)*((r2*r2-r1*r1)/(2*d2));
    D A=sqrt((r1+r2+d)*(r1-r2+d)*(r1+r2-d)*(-r1+r2+d));
    Pt v=Pt( o1.y-o2.y , -o1.x + o2.x ) * A / (2*d2);
    p1 = u + v; p2 = u - v;
    return true;
  }
  struct Teve {
    Pt p; D ang; int add;
    Teve() {}
    Teve(Pt _a, D _b, int _c):p(_a), ang(_b), add(_c){}
    bool operator<(const Teve &a)const
    {return ang < a.ang;}
  }eve[ N * 2 ];
  // strict: x = 0, otherwise x = -1
  bool disjuct( Circle& a, Circle &b, int x )
  {return dcmp( norm( a.o - b.o ) - a.r - b.r ) > x;}
  bool contain( Circle& a, Circle &b, int x )
  {return dcmp( a.r - b.r - norm( a.o - b.o ) ) > x;}
  bool contain(int i, int j){
    /* c[j] is non-strictly in c[i]. */
    return (dcmp(c[i].r - c[j].r) > 0 ||
            (dcmp(c[i].r - c[j].r) == 0 && i < j) ) &&
                contain(c[i], c[j], -1);
  }
  void solve(){
    for( int i = 0 ; i <= C + 1 ; i ++ )
      Area[ i ] = 0;
    for( int i = 0 ; i < C ; i ++ )
      for( int j = 0 ; j < C ; j ++ )
        overlap[i][j] = contain(i, j);
    for( int i = 0 ; i < C ; i ++ )
      for( int j = 0 ; j < C ; j ++ )
        g[i][j] = !(overlap[i][j] || overlap[j][i] ||
                    disjuct(c[i], c[j], -1));
    for( int i = 0 ; i < C ; i ++ ){
      int E = 0, cnt = 1;
      for( int j = 0 ; j < C ; j ++ )
        if( j != i && overlap[j][i] )
          cnt ++;
      for( int j = 0 ; j < C ; j ++ )
        if( i != j && g[i][j] ){
          Pt aa, bb;
          CCinter(c[i], c[j], aa, bb);
          D A=atan2(aa.y - c[i].o.y, aa.x - c[i].o.x);
          D B=atan2(bb.y - c[i].o.y, bb.x - c[i].o.x);
          eve[E ++] = Teve(bb, B, 1);
          eve[E ++] = Teve(aa, A, -1);
          if(B > A) cnt ++;
        }
      if( E == 0 ) Area[ cnt ] += pi * c[i].r * c[i].r;
      else{
        sort( eve , eve + E );
        eve[E] = eve[0];
        for( int j = 0 ; j < E ; j ++ ){
          cnt += eve[j].add;
          Area[cnt] += (eve[j].p ^ eve[j + 1].p) * 0.5;
          D theta = eve[j + 1].ang - eve[j].ang;
          if (theta < 0) theta += 2.0 * pi;
          Area[cnt] +=
            (theta - sin(theta)) * c[i].r*c[i].r * 0.5;
}}}}};
```

1.13 多邊形聯集面積

```
//O(n^2logn)
inline double segP(Pt &p,Pt &p1,Pt &p2){
  if(dcmp(p1.x-p2.x)==0) return (p.y-p1.y)/(p2.y-p1.y);
  return (p.x-p1.x)/(p2.x-p1.x);
}
ld tri(Pt o, Pt a, Pt b){ return (a-o) ^ (b-o);}
double polyUnion(vector<vector<Pt>> py){ //py[0~n-1]
    must be filled
  int n = py.size();
  int i,j,ii,jj,ta,tb,r,d; double z,w,s,sum=0,tc,td,
      area;
  vector<pair<double,int>> c;
  for(i=0;i<n;i++){
    area=py[i][py[i].size()-1]^py[i][0];
    for(int j=0;j<py[i].size()-1;j++) area+=py[i][j]^py
        [i][j+1];
    if((area/=2)<0) reverse(py[i].begin(),py[i].end());
    py[i].push_back(py[i][0]);
  }
  for(i=0;i<n;i++){
    for(ii=0;ii+1<py[i].size();ii++){
      c.clear();
      c.emplace_back(0.0,0); c.emplace_back(1.0,0);
      for(j=0;j<n;j++){
        if(i==j) continue;
        for(jj=0;jj+1<py[j].size();jj++){
          ta=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj]))
              ;
          tb=dcmp(tri(py[i][ii],py[i][ii+1],py[j][jj
              +1]));
          if(ta==0 && tb==0){
            if((py[j][jj+1]-py[j][jj])*(py[i][ii+1]-py[
                i][ii])>0&&j<i){
              c.emplace_back(segP(py[j][jj],py[i][ii],
                  py[i][ii+1]),1);
              c.emplace_back(segP(py[j][jj+1],py[i][ii
                  ],py[i][ii+1]),-1);
            }
          }else if(ta>=0 && tb<0){
            tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
            td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
            c.emplace_back(tc/(tc-td),1);
          }else if(ta<0 && tb>=0){
            tc=tri(py[j][jj],py[j][jj+1],py[i][ii]);
            td=tri(py[j][jj],py[j][jj+1],py[i][ii+1]);
            c.emplace_back(tc/(tc-td),-1);
      } } }
      sort(c.begin(),c.end());
      z=min(max(c[0].first,0.0),1.0); d=c[0].second; s
          =0;
      for(j=1;j<c.size();j++){
        w=min(max(c[j].first,0.0),1.0);
        if(!d) s+=w-z;
        d+=c[j].second; z=w;
      }
      sum+=(py[i][ii]^py[i][ii+1])*s;
  } }
  return sum/2;
}
```

1.14 多邊形覆蓋面積

```
// Area[i] : 至少i個多邊形覆蓋的面積 O(n^2logn)
vector<double> PolyCover(const vector<vector<Pt>> &P) {
    const int n = P.size();
    vector<double> Area(n + 1);
    vector<Line> Ls;
    for (int i = 0; i < n; i++)
        for (int j = 0; j < P[i].size(); j++)
            Ls.push_back({P[i][j], P[i][(j + 1) % P[i].
                size()]});
    auto cmp = [&](Line &l, Line &r) {
        Pt u = l.b - l.a, v = r.b - r.a;
        if (argcmp(u, v)) return true;
        if (argcmp(v, u)) return false;
        return PtSide(l.a, r) < 0;
    };
    sort(all(Ls), cmp);
    for (int l = 0, r = 0; l < Ls.size(); l = r) {
        while (r < Ls.size() and !cmp(Ls[l], Ls[r])) r
            ++;
        Line L = Ls[l];
        vector<pair<Pt, int>> event;
        for (auto [c, d] : Ls) {
            if (sgn((L.a - L.b) ^ (c - d)) != 0) {
                int s1 = PtSide(c, L) == 1;
                int s2 = PtSide(d, L) == 1;
                if (s1 ^ s2) event.emplace_back(
                    LineInter(L, {c, d}), s1 ? 1 : -1);
            } else if (PtSide(c, L) == 0 and sgn((L.a -
                 L.b) * (c - d)) > 0) {
                event.emplace_back(c, 2);
                event.emplace_back(d, -2);
            }
        }
        sort(all(event), [&](auto i, auto j) {
            return (L.a - i.ff) * (L.a - L.b) < (L.a -
                j.ff) * (L.a - L.b);
        });
        int cov = 0, tag = 0;
        Pt lst{0, 0};
        for (auto [p, s] : event) {
            if (cov >= tag) {
                Area[cov] += lst ^ p;
                Area[cov - tag] -= lst ^ p;
            }
            if (abs(s) == 1) cov += s;
            else tag += s / 2;
            lst = p;
        }
    }
    for (int i = n - 1; i >= 0; i--) Area[i] += Area[i
        + 1];
    for (int i = 1; i <= n; i++) Area[i] /= 2;
    return Area;
};
```

1.15 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);
```

1.16 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

1.17 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) { //搜半徑
  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){ //有了x再搜y
    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; //先搜x
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;
}
```

1.18 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 資料結構

2.1 離散化

```
vector<int> tmp(arr);  //將arr複製到tmp
sort(tmp.begin(), tmp.end());
tmp.erase(unique(tmp.begin(), tmp.end()), tmp.end());
for (int i = 0; i < n; i++)
  arr[i] = lower_bound(tmp.begin(), tmp.end(), arr[i])
      - tmp.begin();
```

2.2 線段樹

```
// 區間修改 查詢區間和
#define cl(x) (x<<1)
#define cr(x) (x<<1)+1
int seg[4*N], lazy[4*N], arr[N];
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);
    seg[id] = seg[cl(id)] + seg[cr(id)];
}
void propagation(int id, int l, int r){
    if(lazy[id]){
        seg[id] += (r - l + 1) * lazy[id];
        if(l != r){
            lazy[cl(id)] += lazy[id];
            lazy[cr(id)] += lazy[id];
        }
        lazy[id] = 0;
    }
}
int query(int id, int l, int r, int ql, int qr){
    propagation(id, l, r);
    if (ql > r || qr < l) return 0;
    if(ql <= l && qr >= r) return seg[id];
    int mid = (l + r) >> 1;
    return query(cl(id), l, mid, ql, qr) + query(cr(id)
        , mid+1, r, ql, qr);
}
void update(int id, int l, int r, int sl, int sr, int v
    ){
    propagation(id, l, r);
    if(sl > r || sr < l) return;
    if(sl <= l && r <= sr){
        lazy[id] += v;
        propagation(id, l, r);
        return;
    }
    int mid = (l + r) >> 1;
    update(cl(id), l, mid, sl, sr, v);
    update(cr(id), mid+1, r, sl, sr, v);
    seg[id] = seg[cl(id)] + seg[cr(id)];
}
```

2.3 莫隊算法

```
int n,k = sqrt(n);//每塊大小為k
struct query{
    int l,r,id;
    bool operator<(const query &lhs, const query &rhs){
        if(lhs.l/k!=rhs.l/k)  return lhs.l/k<rhs.l/k;
        return ((lhs.l/k)&1?lhs.r<rhs.r:lhs.r>rhs.r);
}
};
int num = 0;
int cnt[1'000'005], ans[30'005];
vector<query> q;
void add(int index){ ... }
void sub(int index){ ... }
void solve(){
    sort(q.begin(),q.end());
    for(int i=0,l=0,r=-1;i<n;i++){
        while(l>q[i].l) add(--l);
        while(r<q[i].r) add(++r);
        while(l<q[i].l) sub(l++);
        while(r>q[i].r) sub(r--);
        ans[q[i].id] = num;
}    }
```

2.4 CDQ 分治

```
int CDQ (int l, int r) {
    if (l == r) return;
    int mid = (l + r)/2;
    CDQ(l, mid), CDQ(mid+1, r);
    vector<int> tmp;
    for (int i = l, j = mid+1; i <= mid or j <= r; ) {
        while (i < mid and (j == r or y[ord[i]] <= y[ord[j]])) {
            bit.add(z[ord[i]], 1);
            tmp.push_back(ord[i]);
            i++;
        }
        if (j <= r) {
            ans[ord[j]] += bit.que(z[ord[j]]);
            tmp.push_back(ord[j]);
            j++;
        }
    }

    for (int i = l; i <= mid; i++) bit.add(z[ord[i]], -1);
    copy(tmp.begin(), tmp.end(), ord.begin() + l);
};
```

2.5 可持久化線段樹

```
struct node{
    ll val;
    node *l, *r;
};
vector<node *> ver;     //用一個vector紀錄全部版本的根節點
void build(node *now_ver, l, r);
ll query(node *now_ver, l, r, ql, qr);
node *update_ver(node *pre_ver,int l,int r,int pos,int v); //回傳新建的節點
void add_ver(int x,int v){     //修改位置 x 的值為 v
    ver.push_back(update_ver(ver.back(), 0, n-1, x, v));
}
node *update_ver(node *pre_ver, node *x, int l, int r, int pos, int v){
    node *x = new node();     //當前位置建立新節點
    if(l == r){
        x->val = v;
        return x;
    }
    int mid = (l+r)>>1;
    if(pos <= mid){ //更新左邊
        x->l = update(pre_ver->l, x->l, l, mid, pos, v); //左邊節點連向新節點
        x->r = pre_ver->r;                    //右邊連到原本的右邊
    }
    else{ //更新右邊
        x->l = pre_ver->l;                     //左邊連到原本的左邊
        x->r = update(pre_ver->r, x->r, mid+1, r, pos, v);  //右邊節點連向新節點
    }
    x->val = x->l->val + x->r->val;
    return x;
}
```

3 動態規劃

3.1 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]);
```

3.2 無限背包

$O(NW)$

```
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]);
```

3.3 有限背包

$O(NW \log k)$

```
// 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
```

4 數學

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 ntod(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 dton(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;
}
```

4.5 O(1)mul

```
LL mul(LL x,LL y,LL mod){
  // LL ret=x*y-(LL)((long double)x/mod*y)*mod;
    //4捨5入，避免浮點數誤差
  LL ret=x*y-(LL)((long double)x*y/mod+0.5)*mod;
  return ret<0?ret+mod:ret;
}
```

4.6 Miller Rabin

```
// n < 4,759,123,141        3 :  2, 7, 61
// n < 2^64                       7 :
// 2, 325, 9375, 28178, 450775, 9780504, 1795265022
// 或前12個質數
#define LL __int128
LL magic[]={}
bool witness(LL a,LL n,LL u,int t){
  if(!a) return 0;
  LL x=mypow(a,u,n);
  for(int i=0;i<t;i++) {
    LL nx=mul(x,x,n);
    if(nx==1&&x!=1&&x!=n-1) return 1;
    x=nx;
  }
  return x!=1;
}
bool miller_rabin(LL n) {
  int s=(magic number size)
  // iterate s times of witness on n
  if(n<2) return 0;
  if(!(n&1)) return n == 2;
  ll u=n-1; int t=0;
  // n-1 = u*2^t
  while(!(u&1)) u>>=1, t++;
  while(s--){
    LL a=magic[s]%n;
    if(witness(a,n,u,t)) return 0;
  }
  return 1;
}
```

4.7 Pollard Rho

```
// does not work when n is prime  O(n^(1/4))
LL f(LL x, LL c, LL mod){ return add(mul(x,x,mod),c,mod
    ); }
LL pollard_rho(LL n) {
    LL c = 1, x = 0, y = 0, p = 2, q, t = 0;
    while (t++ % 128 or gcd(p, n) == 1) {
        if (x == y) c++, y = f(x = 2, c, n);
        if (q = mul(p, abs(x-y), n)) p = q;
        x = f(x, c, n); y = f(f(y, c, n), c, n);
    }
    return gcd(p, n);
}
```

4.8 FFT

```
// const int MAXN = 262144;
// (must be 2^k)
//steps: pre_fft->mul
typedef long double ld;
typedef complex<ld> cplx; //real() ,imag()
const ld PI = acosl(-1);
const cplx I(0, 1);
cplx omega[MAXN+1];
void pre_fft(){
  for(int i=0; i<=MAXN; i++)
    omega[i] = exp(i * 2 * PI / MAXN * I);
}
// n must be 2^k
void fft(int n, cplx a[], bool inv=false){
  int basic = MAXN / n;
  int theta = basic;
  for (int m = n; m >= 2; m >>= 1) {
    int mh = m >> 1;
    for (int i = 0; i < mh; i++) {
      cplx w = omega[inv ? MAXN-(i*theta%MAXN)
                         : i*theta%MAXN];
      for (int j = i; j < n; j += m) {
        int k = j + mh;
        cplx x = a[j] - a[k];
        a[j] += a[k];
        a[k] = w * x;
    } }
    theta = (theta * 2) % MAXN;
  }
  int i = 0;
  for (int j = 1; j < n - 1; j++) {
    for (int k = n >> 1; k > (i ^= k); k >>= 1);
    if (j < i) swap(a[i], a[j]);
  }
  if(inv) for (i = 0; i < n; i++) a[i] /= n;
}
cplx arr[MAXN+1];
inline void mul(int _n,ll a[],int _m,ll b[],ll ans[]){
  int n=1,sum=_n+_m-1;
  while(n<sum)
    n<<=1;
  for(int i=0;i<n;i++) {
    double x=(i<_n?a[i]:0),y=(i<_m?b[i]:0);
    arr[i]=complex<double>(x+y,x-y);
  }
  fft(n,arr);
  for(int i=0;i<n;i++)
    arr[i]=arr[i]*arr[i];
  fft(n,arr,true);
  for(int i=0;i<sum;i++)
    ans[i]=(long long int)(arr[i].real()/4+0.5);
}
```

4.9 約瑟夫問題

```
int josephus(int n, int m){ //n人每m次
    int ans = 0;
    for (int i=1; i<=n; ++i)
        ans = (ans + m) % i;
    return ans;
}
```

4.10 快速求歐拉函數

```
int prime[10010], phi[10010];
bool v[10010];
void quick_euler(){
  int cnt = 0;
  for(int i = 2; i <= N; ++i){
    if(!v[i]) prime[++cnt] = i, phi[i] = i - 1;
        // 若 i 是質數，所以 □(i) = i - 1
    for(int j = 1; i * prime[j] <= N && j <= cnt; ++j){
      v[i * prime[j]] = 1;
      if(i % prime[j] == 0){
        phi[i * prime[j]] = phi[i] * prime[j];
        break;
       }
       else phi[i * prime[j]] = phi[i] * (prime[j] - 1)
           ;
    }
  }
}
```

4.11 矩陣

```
struct Matrix{
    int n, m;
    int v[105][105];
    Matrix(int _n, int _m): n(_n), m(_m){}
    void init(){ memset(v, 0, sizeof(v));}
    Matrix operator*(const Matrix B) const{
        Matrix C(n, B.m);
        C.init();
        for(int i = 0; i < n; i++){
            for(int j = 0; j < B.m; j++){
                for(int k = 0; k < n; k++){
                    C.v[i][j] = C.v[i][j]+v[i][k]*B.v[k
                        ][j];
                }
            }
        }
        return C;
    }
    Matrix fastpow(Matrix &A, int y){
        Matrix C(A.n, A.m);
        C.init();
        for(int i = 0; i < C.n; i++) C.v[i][i] = 1;
        while(y){
            if(y & 1) C=C*A;
            A = A*A;
            y >>= 1;
        }
        return C;
    }
};
```

5 樹論

5.1 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];
        }
    }
}
```

5.2 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);
    }
  }
}
```

5.3 樹哈希

```
map<vector<int>, int> id;
int dfs(int x, int f){
  vector<int> sub;
  for (int v : edge[x]){
    if (v != f)
      sub.push_back(dfs(v, x));
  }
  sort(sub.begin(), sub.end());
  if (!id.count(sub))
    id[sub] = id.size();
  return id[sub];
}
```

6 圖論

6.1 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
                ]);
        }
    }
}
```

6.2 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];
            }
        }
    }
}
```

6.3 點雙連通分量

```
//step: init(n)->addEdge(u,v)->solve()
//return:二維vector
#define PB push_back
#define REP(i, n) for(int i = 0; i < n; i++)
struct BccVertex {
  int n,nScc,step,dfn[MXN],low[MXN];
  vector<int> E[MXN],sccv[MXN];
  int top,stk[MXN];
  void init(int _n) {
    n = _n; nScc = step = 0;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void addEdge(int u, int v)
  { E[u].PB(v); E[v].PB(u); }
  void DFS(int u, int f) {
    dfn[u] = low[u] = step++;
    stk[top++] = u;
    for (auto v:E[u]) {
      if (v == f) continue;
      if (dfn[v] == -1) {
        DFS(v,u);
        low[u] = min(low[u], low[v]);
        if (low[v] >= dfn[u]) {
          int z;
          sccv[nScc].clear();
          do {
            z = stk[--top];
            sccv[nScc].PB(z);
          } while (z != v);
          sccv[nScc++].PB(u);
        }
      }else
        low[u] = min(low[u],dfn[v]);
  } }
  vector<vector<int>> solve() {
    vector<vector<int>> res;
    for (int i=0; i<n; i++)
      dfn[i] = low[i] = -1;
    for (int i=0; i<n; i++)
      if (dfn[i] == -1) {
        top = 0;
        DFS(i,i);
      }
    REP(i,nScc) res.PB(sccv[i]);
    return res;
  }
}graph;
```

6.4 強連通分量

```
//step: init(n)->addEdge(u,v)->solve()
//有nScc個強連通分量 bln是點i所在的連通分量編號
#define PB push_back
#define FZ(x) memset(x, 0, sizeof(x)) //fill zero
struct Scc{
  int n, nScc, vst[MXN], bln[MXN];
  vector<int> E[MXN], rE[MXN], vec;
  void init(int _n){
    n = _n;
    for (int i=0; i<= n; i++)
      E[i].clear(), rE[i].clear();
  }
  void addEdge(int u, int v){
    E[u].PB(v); rE[v].PB(u);
  }
  void DFS(int u){
    vst[u]=1;
    for (auto v : E[u]) if (!vst[v]) DFS(v);
    vec.PB(u);
  }
  void rDFS(int u){
    vst[u] = 1; bln[u] = nScc;
    for (auto v : rE[u]) if (!vst[v]) rDFS(v);
  }
  void solve(){
    nScc = 0;
    vec.clear();
    fill(vst, vst+n+1, 0);
    for (int i=0; i<n; i++)
      if (!vst[i]) DFS(i);
    reverse(vec.begin(),vec.end());
    fill(vst, vst+n+1, 0);
    for (auto v : vec)
      if (!vst[v]){
        rDFS(v); nScc++;
      }
  }
};
```

7 字串

7.1 KMP

```
vector<int> KMP(string s, string t){
    s = t + '@' + s;
    int sz = s.size();
    vector<int> pi(sz);
    for(int i = 1; i < sz; i++){
        int len = pi[i-1];
        while(len != 0 && s[i] != s[len]) len = pi[len
            -1];
        if(s[i] == s[len]) pi[i] = len+1;
    }
    return pi;
}
```

7.2 Hash

```
const int mod = 1e9 + 7;
pair<int, int> Hash[N];
void get_hash(string s){
    int p1 = 13331, p2 = 75577;
    pair<int, int> val = {0, 0};
    for(int i = 0; i < s.size(); i++){
        val.first = (val.first * p1 + s[i]) % mod;
        val.second = (val.second * p2 + s[i]) % mod;
        Hash[i] = val;
    }
}
```

7.3 minRotation

```
//rotate(begin(s),begin(s)+minRotation(s),end(s))
int minRotation(string s) {
  int a = 0, N = s.size(); s += s;
  rep(b,0,N) rep(k,0,N) {
    if(a+k == b || s[a+k] < s[b+k])
      {b += max(0, k-1); break;}
    if(s[a+k] > s[b+k]) {a = b; break;}
  } return a;
}
```

7.4 Suffix Array

```
const int N = 300010;
struct SA{
#define REP(i,n) for ( int i=0; i<int(n); i++ )
#define REP1(i,a,b) for ( int i=(a); i<=int(b); i++ )
  bool _t[N*2];
  int _s[N*2], _sa[N*2], _c[N*2], x[N], _p[N], _q[N*2],
       hei[N], r[N];
  int operator [] (int i){ return _sa[i]; }
  void build(int *s, int n, int m){
    memcpy(_s, s, sizeof(int) * n);
    sais(_s, _sa, _p, _q, _t, _c, n, m);
    mkhei(n);
  }
  void mkhei(int n){
    REP(i,n) r[_sa[i]] = i;
    hei[0] = 0;
    REP(i,n) if(r[i]) {
      int ans = i>0 ? max(hei[r[i-1]] - 1, 0) : 0;
      while(_s[i+ans] == _s[_sa[r[i]-1]+ans]) ans++;
      hei[r[i]] = ans;
    }
  }
  void sais(int *s, int *sa, int *p, int *q, bool *t,
      int *c, int n, int z){
    bool uniq = t[n-1] = true, neq;
    int nn = 0, nmxz = -1, *nsa = sa + n, *ns = s + n,
        lst = -1;
#define MS0(x,n) memset((x),0,n*sizeof(*(x)))
#define MAGIC(XD) MS0(sa, n); \
    memcpy(x, c, sizeof(int) * z); \
    XD; \
    memcpy(x + 1, c, sizeof(int) * (z - 1)); \
    REP(i,n) if(sa[i] && !t[sa[i]-1]) sa[x[s[sa[i
        ]-1]]++] = sa[i]-1; \
    memcpy(x, c, sizeof(int) * z); \
    for(int i = n - 1; i >= 0; i--) if(sa[i] && t[sa[i
        ]-1]) sa[--x[s[sa[i]-1]]] = sa[i]-1;
    MS0(c, z);
    REP(i,n) uniq &= ++c[s[i]] < 2;
    REP(i,z-1) c[i+1] += c[i];
    if (uniq) { REP(i,n) sa[--c[s[i]]] = i; return; }
    for(int i = n - 2; i >= 0; i--) t[i] = (s[i]==s[i
        +1] ? t[i+1] : s[i]<s[i+1]);
    MAGIC(REP1(i,1,n-1) if(t[i] && !t[i-1]) sa[--x[s[i
        ]]]=p[q[i]=nn++]=i);
    REP(i, n) if (sa[i] && t[sa[i]] && !t[sa[i]-1]) {
      neq=lst<0||memcmp(s+sa[i],s+lst,(p[q[sa[i]]+1]-sa
          [i])*sizeof(int));
      ns[q[lst=sa[i]]]=nmxz+=neq;
    }
    sais(ns, nsa, p + nn, q + n, t + n, c + z, nn, nmxz
         + 1);
    MAGIC(for(int i = nn - 1; i >= 0; i--) sa[--x[s[p[
        nsa[i]]]]] = p[nsa[i]]);
  }
}sa;
int H[ N ], SA[ N ];
void suffix_array(int* ip, int len) {
  // should padding a zero in the back
  // ip is int array, len is array length
  // ip[0..n-1] != 0, and ip[len] = 0
  ip[len++] = 0;
  sa.build(ip, len, 128);
  for (int i=0; i<len; i++) {
    H[i] = sa.hei[i + 1];
    SA[i] = sa._sa[i + 1];
  }
  // resulting height, sa array \in [0,len)
}
```

7.5 馬拉車

```
void z_value_pal(char *s,int len,int *z){
  len=(len<<1)+1;
  for(int i=len-1;i>=0;i--)
    s[i]=i&1?s[i>>1]:'@';
  z[0]=1;
  for(int i=1,l=0,r=0;i<len;i++){
    z[i]=i<r?min(z[l+l-i],r-i):1;
    while(i-z[i]>=0&&i+z[i]<len&&s[i-z[i]]==s[i+z[i]])
        ++z[i];
    if(i+z[i]>r) l=i,r=i+z[i];
} }
```

7.6 Zvalue

```
int z[MAXN];
void Z_value(const string& s) { //z[i] = lcp(s[1...],s[
    i...])
  int i, j, left, right, len = s.size();
  left=right=0; z[0]=len;
  for(i=1;i<len;i++) {
    j=max(min(z[i-left],right-i),0);
    for(;i+j<len&&s[i+j]==s[j];j++);
    z[i]=j;
    if(i+z[i]>right) {
      right=i+z[i];
      left=i;
}   }   }
```

7.7 字典樹

```
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++;
    }
};
```

7.8 回文樹

```
// len[s]是對應的回文長度
// num[s]是有幾個回文後綴
// cnt[s]是這個回文子字串在整個字串中的出現次數
// fail[s]是他長度次長的回文後綴，aba的fail是a
const int MXN = 1000010;
struct PalT{
  int nxt[MXN][26],fail[MXN],len[MXN];
  int tot,lst,n,state[MXN],cnt[MXN],num[MXN];
  int diff[MXN],sfail[MXN],fac[MXN],dp[MXN];
  char s[MXN]={-1};
  int newNode(int l,int f){
    len[tot]=l,fail[tot]=f,cnt[tot]=num[tot]=0;
    memset(nxt[tot],0,sizeof(nxt[tot]));
    diff[tot]=(l>0?l-len[f]:0);
    sfail[tot]=(l>0&&diff[tot]==diff[f]?sfail[f]:f);
    return tot++;
  }
  int getfail(int x){
    while(s[n-len[x]-1]!=s[n]) x=fail[x];
    return x;
  }
  int getmin(int v){
    dp[v]=fac[n-len[sfail[v]]-diff[v]];
    if(diff[v]==diff[fail[v]])
        dp[v]=min(dp[v],dp[fail[v]]);
    return dp[v]+1;
  }
  int push(){
    int c=s[n]-'a',np=getfail(lst);
    if(!(lst=nxt[np][c])){
      lst=newNode(len[np]+2,nxt[getfail(fail[np])][c]);
      nxt[np][c]=lst; num[lst]=num[fail[lst]]+1;
    }
    fac[n]=n;
    for(int v=lst;len[v]>0;v=sfail[v])
        fac[n]=min(fac[n],getmin(v));
    return ++cnt[lst],lst;
  }
  void init(const char *_s){
    tot=lst=n=0;
    newNode(0,1),newNode(-1,1);
    for(;_s[n];) s[n+1]=_s[n],++n,state[n-1]=push();
    for(int i=tot-1;i>1;i--) cnt[fail[i]]+=cnt[i];
  }
}palt;
```

8 網路流

8.1 Dinic

```
#define PB push_back
#define SZ(x) (int)x.size()
struct Dinic{
  struct Edge{ int v,f,re; };
  int n,s,t,level[MXN];
  vector<Edge> E[MXN];
  void init(int _n, int _s, int _t){
    n = _n; s = _s; t = _t;
    for (int i=0; i<n; i++) E[i].clear();
  }
  void add_edge(int u, int v, int f){
    E[u].PB({v,f,SZ(E[v])});
    E[v].PB({u,0,SZ(E[u])-1});
  }
  bool BFS(){
    for (int i=0; i<n; i++) level[i] = -1;
    queue<int> que;
    que.push(s);
    level[s] = 0;
    while (!que.empty()){
      int u = que.front(); que.pop();
      for (auto it : E[u]){
        if (it.f > 0 && level[it.v] == -1){
          level[it.v] = level[u]+1;
          que.push(it.v);
    } } }
    return level[t] != -1;
  }
  int DFS(int u, int nf){
    if (u == t) return nf;
    int res = 0;
    for (auto &it : E[u]){
      if (it.f > 0 && level[it.v] == level[u]+1){
        int tf = DFS(it.v, min(nf,it.f));
        res += tf; nf -= tf; it.f -= tf;
        E[it.v][it.re].f += tf;
        if (nf == 0) return res;
    } }
    if (!res) level[u] = -1;
    return res;
  }
  int flow(int res=0){
    while ( BFS() )
      res += DFS(s,2147483647);
    return res;
} }flow;
```

8.2 最小花費最大流

```
#define PB push_back
#define SZ(x) (int)x.size()
struct zkwflow{
  static const int maxN=10000;
  struct Edge{ int v,f,re; ll w;};
  int n,s,t,ptr[maxN]; bool vis[maxN]; ll dis[maxN];
  vector<Edge> E[maxN];
  void init(int _n,int _s,int _t){
    n=_n,s=_s,t=_t;
    for(int i=0;i<n;i++) E[i].clear();
  }
  void addEdge(int u,int v,int f,ll w){
    E[u].push_back({v,f,(int)E[v].size(),w});
    E[v].push_back({u,0,(int)E[u].size()-1,-w});
  }
  bool SPFA(){
    fill_n(dis,n,LLONG_MAX); fill_n(vis,n,false);
    queue<int> q; q.push(s); dis[s]=0;
    while (!q.empty()){
      int u=q.front(); q.pop(); vis[u]=false;
      for(auto &it:E[u]){
        if(it.f>0&&dis[it.v]>dis[u]+it.w){
          dis[it.v]=dis[u]+it.w;
          if(!vis[it.v]){
            vis[it.v]=true; q.push(it.v);
    } } } }
    return dis[t]!=LLONG_MAX;
  }
  int DFS(int u,int nf){
    if(u==t) return nf;
    int res=0; vis[u]=true;
    for(int &i=ptr[u];i<(int)E[u].size();i++){
      auto &it=E[u][i];
      if(it.f>0&&dis[it.v]==dis[u]+it.w&&!vis[it.v]){
        int tf=DFS(it.v,min(nf,it.f));
        res+=tf,nf-=tf,it.f-=tf;
        E[it.v][it.re].f+=tf;
        if(nf==0){ vis[u]=false; break; }
      }
    }
    return res;
  }
  pair<int,ll> flow(){
    int flow=0; ll cost=0;
    while (SPFA()){
      fill_n(ptr,n,0);
      int f=DFS(s,INT_MAX);
      flow+=f; cost+=dis[t]*f;
    }
    return{ flow,cost };
  } // reset: do nothing
} flow;
```

8.3 最小割

```
bool vis[MXN];
void dfs(int x){
    vis[x] = 1;
    for(int i : flow.G[x]){
        if(i.f > 0 && !vis[i.v]){
            dfs(i.v);
        }
    }
}
dfs(source);
```

8.4 匈牙利演算法

```
bool dfs(int u){
    for(int i=1;i<=n;i++){
        if(Map[u][i]&&!vis[i]){ //有連通且未拜訪
            vis[i]=1; //紀錄是否走過
            if(S[i]==-1||dfs(S[i])){  //紀錄匹配
                S[i]=u;
                return true;  //反轉匹配邊以及未匹配邊
                    的狀態
            }
        }
    }
    return false;
}
// 記得每次使用需清空vis數組
// 其中Map為鄰接表 S為紀錄這個點與誰匹配
for(int i=1;i<=p;i++){
    memset(vis,0,sizeof(vis));
    if(dfs(i)) ans++;
}
```

8.5 二分圖最大權完美

```
struct KM{ // O(n^3)
  int n, mx[MXN], my[MXN], pa[MXN];
  ll g[MXN][MXN], lx[MXN], ly[MXN], sy[MXN];
  bool vx[MXN], vy[MXN];
  void init(int _n) { // 1-based, N個節點
    n = _n;
    for(int i=1; i<=n; i++) fill(g[i], g[i]+n+1, 0);
  }
  void addEdge(int x, int y, ll w) {g[x][y] = w;} //左
      邊的集合節點x連邊右邊集合節點y權重為w
  void augment(int y) {
    for(int x, z; y; y = z)
      x=pa[y], z=mx[x], my[y]=x, mx[x]=y;
  }
  void bfs(int st) {
    for(int i=1; i<=n; ++i) sy[i]=INF, vx[i]=vy[i]=0;
    queue<int> q; q.push(st);
    for(;;) {
      while(q.size()) {
        int x=q.front(); q.pop(); vx[x]=1;
        for(int y=1; y<=n; ++y) if(!vy[y]){
          ll t = lx[x]+ly[y]-g[x][y];
          if(t==0){
            pa[y]=x;
            if(!my[y]){augment(y);return;}
            vy[y]=1, q.push(my[y]);
          }else if(sy[y]>t) pa[y]=x,sy[y]=t;
      } }
      ll cut = INF;
      for(int y=1; y<=n; ++y)
        if(!vy[y]&&cut>sy[y]) cut=sy[y];
      for(int j=1; j<=n; ++j){
        if(vx[j]) lx[j] -= cut;
        if(vy[j]) ly[j] += cut;
        else sy[j] -= cut;
      }
      for(int y=1; y<=n; ++y) if(!vy[y]&&sy[y]==0){
        if(!my[y]){augment(y);return;}
        vy[y]=1, q.push(my[y]);
  } } }
  ll solve(){ // 回傳值為完美匹配下的最大總權重
    fill(mx, mx+n+1, 0); fill(my, my+n+1, 0);
    fill(ly, ly+n+1, 0); fill(lx, lx+n+1, -INF);
    for(int x=1; x<=n; ++x) for(int y=1; y<=n; ++y) //
        1-base
      lx[x] = max(lx[x], g[x][y]);
    for(int x=1; x<=n; ++x) bfs(x);
    ll ans = 0;
    for(int y=1; y<=n; ++y) ans += g[my[y]][y];
    return ans;
} }graph;
```

9 小技巧

9.1 快讀/快寫

```
inline int read(){
    int x=0,f=1;
    char ch=getchar();
    while(ch<'0'||ch>'9'){
        if(ch=='-') f=-1;
        ch=getchar();
    }
    while(ch>='0' && ch<='9') x=x*10+ch-'0',ch=getchar
        ();
    return x*f;
}
void write(int x){
    if(x<0) putchar('-'),x=-x;
    if(x>9) write(x/10);
    putchar(x%10+'0');
    return;
}
```

9.2 隨機數

```
#include<iostream>
#include<random>
using namespace std;
signed main() {
    mt19937 mt(hash<string>(":poop:"));
    for(int i=1;i<=5;i++) cout<<mt()<<" \n"[i==5];
    return 0;
}
```

9.3 Linus 指令

```
cd ..        回到上一層資料夾
cd ../..     回到上上層資料夾
cd test      到當前目錄的test資料夾
ls           顯示當前資料夾的檔案
cat a.cpp    印出當前檔案的內容
mkdir test   在當前目錄建立 test 的資料夾
rm  a.cpp    刪除 a.cpp 的檔案
g++ solve.cpp                編譯solve.cpp的檔案成 a.
    out 檔
g++ solve.cpp -o ac.out      編譯solve.cpp的檔案成 ac.
    out 檔
g++ solve.cpp -std=c++14     編譯solve.cpp的檔案成 a.
    out 檔 並且編譯版本為 c++14
./a.out                      執行 a.out 檔
-fsanitize=undefined
插入各種undefined behavior檢查，會在執行期輸出錯誤訊息
-Wall -Wextra
把warning都開起來，常能預防bug發生
-Wshadow
當有宣告了相同變數名稱的情形發生時予以警告
alias [name]='[value]'
alias g++ = `g++ -std=c++14 -fsanitize=undefined -Wall
    -Wextra -Wshadow`
factor 100 //產生質因數
```

9.4 Windows 對拍

```
g++ ac.cpp -o ac
g++ wa.cpp -o wa
$i = 0
while ($true) {
    Write-Output "$i"
    python gen.py > input
    Get-Content input | .\ac.exe > ac.out
    Get-Content input | .\wa.exe > wa.out
    $acOut = Get-Content .\ac.out
    $waOut = Get-Content .\wa.out
    if (diff $acOut $waOut) {
        diff $acOut $waOut
        break
    }
    $i++
}
```