

1 Dp

1.1 01_knapsack

```
1 for(int i = 0; i < 100005; i++) dp[i] = dp1[i] = 205;
2 dp[0] = dp1[0] = 0;
3 for(int i = 1; i <= n; i++) {
4     for(int j = 1e5+1; j >= a[i]; j--) { // 到著做回
5         dp[j] = min(dp[j-a[i]] + 1, dp[j]);
6     }
7 }
```

1.2 Josephus

```
1 int josephus(int n, int k) { // n people, kth is killed
2     if (n == 1) return 1;
3     else return (josephus(n - 1, k) + k - 1) % n + 1;
4     /* The position returned by josephus(n - 1, k)
5      is adjusted because the recursive call
6      josephus(n - 1, k) considers the
7      original position k % n + 1 as position 1 */
8 }
```

1.3 Bitmask

```
1 // n個城市, m個單向邊, 求從1出發走到n的所有 徑數
2 // 遞迴版本, 存反向圖
3 ll alln;
4 ll tbl[20][1<<20]; // 建表
5 ll dp(int i, ll vs) {
6     if(tbl[i][vs]) return tbl[i][vs];
7     if(vs == alln && i == 0) return 1;
8     if(vs == alln || i == 0) return 0;
9     ll r = 0;
10    For(j, n) {
11        if(!g[i][j]) continue;
12        if(vs & (1<<j)) continue;
13        r += dp(j, vs | (1<<j)) * g[i][j];
14        r %= mod;
15    }
16    return tbl[i][vs] = r % mod;
17 }
18
19 alln = (1<<n)-1;
20 ans = dp(n-1, 1<<(n-1))%mod; //從最後一點遞迴回去, bitmask n
    -1位為1, 其餘為0
21
22 // TLE版本, 迴圈版很難壓常, 存正向圖
23 N = (1<<n)-1; // 可表示n個bit的bitmask
24 dp[0][1] = 1;
25 for(int mask = 1; mask <= N; mask++) {
26     for(int i = 0; i < n; i++) {
27         if(!(1 & mask>>i)) continue;
28         int mask2 = mask - (1<<i);
29         for(int j = 0; j < n; j++) {
30             if(!(1 & mask2>>j) || g[j][i] == 0) continue;
```

```
31         dp[i][mask] += dp[j][mask2]*g[j][i]; // 非簡單
    圖, 可能有重複單向邊, g[i][j]存邊數
32         dp[i][mask] %= mod;
33     }
34 }
35 }
36 cout << dp[n-1][N] % mod << '\n';
```

1.4 InfinitKnapsack

```
1 // 找零問題
2 int main() { //O(n^2)
3     dp[0] = 1;
4     for(ll i = 1; i <= n; i++) {
5         for(ll j = a[i]; j < 30001; j++) { // 順著做過去
6             dp[j] += dp[j-a[i]];
7             if(dp[j-a[i]]) coin[j] = i; // 此 額當前拿ㄌ哪一
    個錢幣
8         }
9     }
10    ll ans = dp[sum]; // sum = 所求 額
11    while (sum) {
12        ans.push_back(coin[sum]);
13        sum -= a[coin[sum]]; // 遞迴找用過哪些錢幣
14    }
15 }
```

2 Data Structure

2.1 DSU

```
1 class DSU{
2 public:
3     DSU(int n) {
4         this->n = n;
5         reset();
6     }
7     int n;
8     vector<int> boss;
9     vector<int> rank;
10    vector<int> size;
11    void reset() {
12        this->boss.resize(n);
13        this->rank.resize(n,0);
14        this->size.resize(n,0);
15        for(int i = 0; i < n; i++) {
16            boss[i] = i;
17        }
18    }
19    int find(int x) {
20        if(boss[x] != x) {
21            boss[x] = find(boss[x]);
22        }
23        return boss[x];
24    }
25    int get_size(int x) {
26        return size[find(x)];
```

```
27 }
28 void merge(int x, int y) {
29     int a = find(x);
30     int b = find(y);
31     if(a!=b) {
32         if(rank[a]<rank[b]) {
33             boss[a] = b;
34             size[b] += size[a];
35         }else if (rank[a]<rank[b]) {
36             boss[b] = a;
37             size[a] += size[b];
38         }else{
39             boss[a] = b;
40             size[b] += size[a];
41             rank[b]++;
42         }
43     }
44 }
45 bool aresame(int a,int b){
46     return find(a)==find(b);
47 }
48 };
```

2.2 Monotonic Queue

```
1 class Monotonic_queue{
2 private:
3     deque<int> qu;
4 public:
5     void push(int n){
6         while(!qu.empty() && qu.back() < n) {
7             qu.pop_back();
8         }
9         qu.push_back(n);
10    }
11    int max() {
12        return qu.front();
13    }
14    int min() {
15        return qu.back();
16    }
17    int size() {
18        return qu.size();
19    }
20    void pop() {
21        qu.pop_front();
22    }
23 };
```

2.3 BIT

```
1 class BIT{
2 public:
3     vector<int> bit;
4     int N;
5     BIT(int n) {
6         this->N = n;
7         this->bit.resize(n);
8     }
9     void update(int x,int d) {
```

```

10 while(x<=N){
11     bit[x] +=d;
12     x +=x&(-x); // lowest bit in x;
13 }
14 }
15 int query(int x){
16     int res = 0;
17     while(x){
18         res+= bit[x];
19         x -= x&-x;
20     }
21     return res;
22 }
23 };

```

2.4 Treap

```

1 // 區間加值、反轉、rotate、刪除、插入元素、求區間
2 // srand(time(0))
3 class Treap {
4     private:
5     struct Node {
6         int pri = rand(), size = 1;
7         ll val, mn, inc = 0; bool rev = 0;
8         Node *lc = 0, *rc = 0;
9         Node(ll v) { val = mn = v; }
10    };
11    Node* root = 0;
12    void rev(Node* t) {
13        if (!t) return;
14        swap(t->lc, t->rc), t->rev ^= 1;
15    }
16    void update(Node* t, ll v) {
17        if (!t) return;
18        t->val += v, t->inc += v, t->mn += v;
19    }
20    void push(Node* t) {
21        if (t->rev) rev(t->lc), rev(t->rc), t->rev = 0;
22        update(t->lc, t->inc), update(t->rc, t->inc);
23        t->inc = 0;
24    }
25    void pull(Node* t) {
26        t->size = 1 + size(t->lc) + size(t->rc);
27        t->mn = t->val;
28        if (t->lc) t->mn = min(t->mn, t->lc->mn);
29        if (t->rc) t->mn = min(t->mn, t->rc->mn);
30    }
31    void discard(Node* t) { // 看要不要釋放記憶體
32        if (!t) return;
33        discard(t->lc), discard(t->rc);
34        delete t;
35    }
36    void split(Node* t, Node*& a, Node*& b, int k) {
37        if (!t) return a = b = 0, void();
38        push(t);
39        if (size(t->lc) < k) {
40            a = t;
41            split(t->rc, a->rc, b, k - size(t->lc) - 1);
42            pull(a);
43        } else {
44            b = t;
45            split(t->lc, a, b->lc, k);
46            pull(b);

```

```

47    }
48    }
49    Node* merge(Node* a, Node* b) {
50        if (!a || !b) return a ? a : b;
51        if (a->pri > b->pri) {
52            push(a);
53            a->rc = merge(a->rc, b);
54            pull(a);
55            return a;
56        } else {
57            push(b);
58            b->lc = merge(a, b->lc);
59            pull(b);
60            return b;
61        }
62    }
63    inline int size(Node* t) { return t ? t->size : 0; }
64    public:
65    int size() { return size(root); }
66    void add(int l, int r, ll val) {
67        Node *a, *b, *c, *d;
68        split(root, a, b, r);
69        split(a, c, d, l - 1);
70        update(d, val);
71        root = merge(merge(c, d), b);
72    }
73    // 反轉區間 [l, r]
74    void reverse(int l, int r) {
75        Node *a, *b, *c, *d;
76        split(root, a, b, r);
77        split(a, c, d, l - 1);
78        swap(d->lc, d->rc);
79        d->rev ^= 1;
80        root = merge(merge(c, d), b);
81    }
82    // 區間 [l, r] 向右 rotate k 次, k < 0 表向左 rotate
83    void rotate(int l, int r, int k) {
84        int len = r - l + 1;
85        Node *a, *b, *c, *d, *e, *f;
86        split(root, a, b, r);
87        split(a, c, d, l - 1);
88        k = (k + len) % len;
89        split(d, e, f, len - k);
90        root = merge(merge(c, merge(f, e)), b);
91    }
92    // 插入一個元素 val 使其 index = i <= size
93    void insert(int i, ll val) {
94        if (i == size() + 1) {
95            push_back(val); return;
96        }
97        assert(i <= size());
98        Node *a, *b;
99        split(root, a, b, i - 1);
100        root = merge(merge(a, new Node(val)), b);
101    }
102    void push_back(ll val) {
103        root = merge(root, new Node(val));
104    }
105    void remove(int l, int r) {
106        int len = r - l + 1;
107        Node *a, *b, *c, *d;
108        split(root, a, b, l - 1);
109        split(b, c, d, len);
110        discard(c); // 看你要不要釋放記憶體
111        root = merge(a, d);

```

```

112    }
113    ll minn(int l, int r) {
114        Node *a, *b, *c, *d;
115        split(root, a, b, r);
116        split(a, c, d, l - 1);
117        int ans = d->mn;
118        root = merge(merge(c, d), b);
119        return ans;
120    }
121 };

```

2.5 Segment Tree

```

1 class SegmentTree{
2     private:
3     const int n;
4     const vl arr;
5     // vl st;
6     vl summ;
7     vl minn;
8     vl maxx;
9     vl tag;
10    void pull(int l,int r,int v){
11        if(r-l==1)
12            return;
13        // st[v]=st[2*v+1]+st[2*v+2];
14        int mid=(l+r)/2;
15        push(l,mid,2*v+1);
16        push(mid,r,2*v+2);
17        summ[v]=summ[2*v+1]+summ[2*v+2];
18        // minn[v]=min(minn[2*v+1],minn[2*v+2]);
19        // maxx[v]=max(maxx[2*v+1],maxx[2*v+2]);
20    }
21    void push(int l,int r,int v){
22        summ[v]+=tag[v]*(r-l);
23        if(r-l==1)
24            return tag[v]=0,void();
25        tag[2*v+1]+=tag[v];
26        tag[2*v+2]+=tag[v];
27        tag[v]=0;
28    }
29    void build(int l,int r,int v=0){
30        if(r-l==1){
31            summ[v]=arr[l];
32            // summ[v]=minn[v]=maxx[v]=arr[l];
33            return;
34        }
35        int mid=(l+r)/2;
36        build(l,mid,2*v+1);
37        build(mid,r,2*v+2);
38        pull(l,r,v);
39    }
40    public:
41    SegmentTree(vl&_arr,int _n):arr(_arr),n(_n){
42        assert(arr.size()==n);
43        summ.assign(4*n,0);
44        // minn.assign(4*n,1e9);
45        // maxx.assign(4*n,-1e9);
46        tag.assign(4*n,0);
47        build(0,arr.size());
48    }
49    void modify(int x,int val,int l,int r,int v=0){
50

```

```

51 }
52 // query sum
53 loli query(int L,int R,int l,int r,int v=0){
54     // dbn(L,R,l,r,v)
55     push(l,r,v);
56     if(l==L && R==r){
57         return summ[v];
58         return minn[v];
59         return maxx[v];
60     }
61     int mid=(l+r)/2;
62     if(R<=mid)
63         return query(L,R,l,mid,2*v+1);
64     else if(mid<=L)
65         return query(L,R,mid,r,2*v+2);
66     else
67         return query(L,mid,l,mid,2*v+1)+query(mid,R,mid,r,2*v+2);
68 }
69 // plus `val` to every element in [L,R)
70 void update(int L,int R,loli val,int l,int r,int v=0){
71     // dbn(L,R,l,r,v)
72     push(l,r,v);
73     if(l==L && R==r){
74         tag[v]+=val;
75         push(l,r,v);
76         return;
77     }
78     int mid=(l+r)/2;
79     if(R<=mid)
80         update(L,R,val,l,mid,2*v+1);
81     else if(mid<=L)
82         update(L,R,val,mid,r,2*v+2);
83     else
84         update(L,mid,val,l,mid,2*v+1),update(mid,R,val,
85             mid,r,2*v+2);
86     pull(l,r,v);
87 }
88 };
89 void solve(){
90     int n,q;
91     cin>>n>>q;
92     vl arr(n);
93     for(auto&x:arr)
94         cin>>x;
95     SegmentTree st(arr,n);
96     while(q--){
97         int op=0;
98         // str op;
99         cin>>op;
100         if(op&1){
101             loli l,r,val;
102             cin>>l>>r>>val;
103             assert(r>=l);
104             st.update(l-1,r,val,0,n);
105             // loli k,u;
106             // cin>>k>>u;
107             // st.update(k-1,k,u-arr[k-1],0,n);
108             // arr[k-1]=u;
109         }else{
110             int x,y;
111             cin>>x>>y;
112             assert(y>=x);
113             cout<<st.query(x-1,y,0,n)<<endl;
114 }

```

```

115 }
116 }
117 }

```

2.6 Sparse Table

```

1 int a[N], sp[___lg(N) + 1][N];
2 void init(int n) { //0-based
3     for (int i = 0; i < n; ++i) {
4         sp[0][i] = a[i];
5     }
6     for (int i = 0; i < ___lg(n); ++i) {
7         for (int j = 0; j+(1<<i) < n; ++j) {
8             sp[i + 1][j] = max(sp[i][j], sp[i][j+(1<<i)]);
9         }
10    }
11 }
12 int query(int l, int r) { //[l, r)
13     int p = ___lg(r - l + 1);
14     return max(sp[p][l], sp[p][r - (1<<p) + 1]);
15 }

```

2.7 Monotonic Stack

```

1 vector<int> monotonic_stack(vector<int> nums){
2     int n = nums.size();
3     vector<int> res(n);
4     stack<int> st;
5     for(int i = n-1;i>=0;i--){
6         while(!st.empty() && st.top()<=nums[i]){
7             st.pop();
8         }
9         if(st.empty())res[i] = -1;
10        else res[i] = st.top();
11        st.push(nums[i]);
12    }
13    return res;
14 }

```

3 Flow

3.1 Maximum Simple Graph Matching

```

1 struct GenMatch { // 1-base
2     int V, pr[N];
3     bool el[N][N], inq[N], inp[N], inb[N];
4     int st, ed, nb, bk[N], djs[N], ans;
5     void init(int _V) {
6         V = _V;
7         for (int i = 0; i <= V; ++i) {
8             for (int j = 0; j <= V; ++j) el[i][j] = 0;
9             pr[i] = bk[i] = djs[i] = 0;
10            inq[i] = inp[i] = inb[i] = 0;
11        }
12    }

```

```

12 }
13 void add_edge(int u, int v) {
14     el[u][v] = el[v][u] = 1;
15 }
16 int lca(int u, int v) {
17     fill_n(inp, V + 1, 0);
18     while (1)
19         if (u = djs[u], inp[u] = true, u == st) break;
20         else u = bk[pr[u]];
21     while (1)
22         if (v = djs[v], inp[v]) return v;
23         else v = bk[pr[v]];
24     return v;
25 }
26 void upd(int u) {
27     for (int v; djs[u] != nb;) {
28         v = pr[u], inb[djs[u]] = inb[djs[v]] = true;
29         u = bk[v];
30         if (djs[u] != nb) bk[u] = v;
31     }
32 }
33 void blo(int u, int v, queue<int> &qe) {
34     nb = lca(u, v), fill_n(inb, V + 1, 0);
35     upd(u), upd(v);
36     if (djs[u] != nb) bk[u] = v;
37     if (djs[v] != nb) bk[v] = u;
38     for (int tu = 1; tu <= V; ++tu)
39         if (inb[djs[tu]])
40             if (djs[tu] = nb, !inq[tu])
41                 qe.push(tu), inq[tu] = 1;
42 }
43 void flow() {
44     fill_n(inq + 1, V, 0), fill_n(bk + 1, V, 0);
45     iota(djs + 1, djs + V + 1, 1);
46     queue<int> qe;
47     qe.push(st), inq[st] = 1, ed = 0;
48     while (!qe.empty()) {
49         int u = qe.front();
50         qe.pop();
51         for (int v = 1; v <= V; ++v)
52             if (el[u][v] && djs[u] != djs[v] &&
53                 pr[u] != v) {
54                 if ((v == st) ||
55                     (pr[v] > 0 && bk[pr[v]] > 0))
56                     blo(u, v, qe);
57                 else if (!bk[v]) {
58                     if (bk[v] = u, pr[v] > 0) {
59                         if (!inq[pr[v]]) qe.push(pr[v]);
60                     } else
61                         return ed = v, void();
62                 }
63             }
64     }
65 }
66 void aug() {
67     for (int u = ed, v, w; u > 0;)
68         v = bk[u], w = pr[v], pr[v] = u, pr[u] = v,
69         u = w;
70 }
71 int solve() {
72     fill_n(pr, V + 1, 0), ans = 0;
73     for (int u = 1; u <= V; ++u)
74         if (!pr[u])
75             if (st = u, flow(), ed > 0) aug(), ++ans;
76     return ans;
77 }

```

78 };

3.2 Dinic

```

1 #define maxn 2005
2 #define INF 0x3f3f3f3f
3 struct MaxFlow{
4     struct edge{
5         int to, cap, flow, rev;
6         edge( int v, int c, int f, int r) : to(v), cap(c),
7             flow(f), rev(r) {}
8     };
9     vector<edge> G[maxn];
10    int s, t, dis[maxn], cur[maxn], vis[maxn];
11    void add_edge(int from, int to, int cap) {
12        G[from].push_back(edge(to, cap, 0, G[to].size()));
13        G[to].push_back(edge(from, 0, 0, G[from].size()-1));
14    }
15    bool bfs() {
16        memset(dis, -1, sizeof(dis));
17        queue<int> qu;
18        qu.push(s);
19        dis[s] = 0;
20        while (!qu.empty()) {
21            int from = qu.front();
22            qu.pop();
23            for (auto &e: G[from]) {
24                if (dis[e.to]==-1 && e.cap != e.flow) {
25                    dis[e.to] = dis[from] + 1;
26                    qu.push(e.to);
27                }
28            }
29        }
30        return dis[t]!=-1;
31    }
32    int dfs(int from, int cap) {
33        if (from==t || cap==0) return cap;
34        for (int &i = cur[from]; i<G[from].size(); i++) {
35            edge &e = G[from][i];
36            if (dis[e.to]==dis[from]+1 && e.flow!=e.cap) {
37                int df = dfs(e.to, min(e.cap-e.flow, cap));
38                if (df) {
39                    e.flow+=df;
40                    G[e.to][e.rev].flow-=df;
41                    return df;
42                }
43            }
44        }
45        dis[from] = -1;
46        return 0;
47    }
48    int Maxflow(int s, int t) {
49        this->s = s, this->t = t;
50        int flow = 0;
51        int df;
52        while (bfs()) {
53            memset(cur, 0, sizeof(cur));
54            while (df = dfs(s, INF)) {
55                flow += df;
56            }
57        }
58        return flow;

```

```

59 };
60 int main() {
61     int n = 4, m = 6;
62     MaxFlow maxflow;
63     for (int i = 0; i < m; i++) {
64         int a, b, cap;
65         cin >> a >> b >> cap;
66         maxflow.add_edge(a, b, cap);
67     }
68     cout << maxflow.Maxflow(1, 3) << endl;
69 }

```

4 Formula

4.1 formula

4.1.1 Pick 公式

給定頂點坐標均是整點的簡單多邊形，面積 = 內部格點數 + 邊上格點數/2 - 1

4.1.2 圖論

- 對於平面圖， $F = E - V + C + 1$ ， C 是連通分數
- 對於平面圖， $E \leq 3V - 6$
- 對於連通圖 G ，最大獨立點集的大小設為 $I(G)$ ，最大匹配大小設為 $M(G)$ ，最小點覆蓋設為 $C_v(G)$ ，最小邊覆蓋設為 $C_e(G)$ 。對於任意連通圖：

$$(a) \quad I(G) + C_v(G) = |V|$$

$$(b) \quad M(G) + C_e(G) = |V|$$

- 對於連通二分圖：

$$(a) \quad I(G) = C_v(G)$$

$$(b) \quad M(G) = C_e(G)$$

- 最大權閉合圖：

$$(a) \quad C(u, v) = \infty, (u, v) \in E$$

$$(b) \quad C(S, v) = W_v, W_v > 0$$

$$(c) \quad C(v, T) = -W_v, W_v < 0$$

$$(d) \quad \text{ans} = \sum_{W_v > 0} W_v - \text{flow}(S, T)$$

- 最大密度子圖：

$$(a) \quad \text{求 } \max \left(\frac{W_e + W_v}{|V'|} \right), e \in E', v \in V'$$

$$(b) \quad U = \sum_{v \in V} 2W_v + \sum_{e \in E} W_e$$

$$(c) \quad C(u, v) = W_{(u, v)}, (u, v) \in E, \text{雙向邊}$$

$$(d) \quad C(S, v) = U, v \in V$$

$$(e) \quad D_u = \sum_{(u, v) \in E} W_{(u, v)}$$

$$(f) \quad C(v, T) = U + 2g - D_v - 2W_v, v \in V$$

$$(g) \quad \text{二分搜 } g:$$

$$l = 0, r = U, \text{eps} = 1/n^2$$

$$\text{if } ((U \times |V| - \text{flow}(S, T))/2 > 0) \quad l = \text{mid}$$

$$\text{else } r = \text{mid}$$

$$(h) \quad \text{ans} = \min_cut(S, T)$$

$$(i) \quad |E| = 0 \text{ 要特殊判斷}$$

- 弦圖：

$$(a) \quad \text{點數大於 } 3 \text{ 的環都要有一條弦}$$

- 完美消除序 從後往前依次給每個點染色，給每個點染上可以染的最小顏色
- 最大團大小 = 色數
- 最大獨立集：完美消除序 從前往後能選就選
- 最小團覆蓋：最大獨立集的點和他延伸的邊構成
- 區間圖是弦圖
- 區間圖的完美消除序：將區間按造又端點由小到大排序
- 區間圖染色：用線段樹做

4.1.3 dinic 特殊圖複雜度

- 單位流： $O\left(\min\left(V^{3/2}, E^{1/2}\right)E\right)$
- 二分圖： $O\left(V^{1/2}E\right)$

4.1.4 0-1 分數規劃

$x_i \in \{0, 1\}$ ， x_i 可能會有其他限制，求 $\max \left(\frac{\sum B_i x_i}{\sum C_i x_i} \right)$

- $D(i, g) = B_i - g \times C_i$
- $f(g) = \sum D(i, g) x_i$
- $f(g) = 0$ 時 g 為最佳解， $f(g) < 0$ 沒有意義
- 因為 $f(g)$ 單調可以二分搜 g
- 或用 Dinkelbach 通常比較快

```

1 binary_search() {
2     while (r-l>eps) {
3         g=(l+r)/2;
4         for (i:所有元素) D[i]=B[i]-g*C[i]; //D(i,g)
5         找出一組合法x[i]使f(g)最大;
6         if (f(g)>0) l=g;
7         else r=g;
8     }
9     Ans = r;
10 }
11 Dinkelbach() {
12     g=任意 態 (通常設為0);
13     do {
14         Ans=g;
15         for (i:所有元素) D[i]=B[i]-g*C[i]; //D(i,g)
16         找出一組合法x[i]使f(g)最大;
17         p=0, q=0;
18         for (i:所有元素)
19             if (x[i]) p+=B[i], q+=C[i];
20         g=p/q; // 新解，注意q=0的情況
21     } while (abs(Ans-g)>EPS);
22     return Ans;
23 }

```

4.1.5 學長公式

- $\sum_{d|n} \phi(n) = n$
- $g(n) = \sum_{d|n} f(d) \Rightarrow f(n) = \sum_{d|n} \mu(d) \times g(n/d)$
- Harmonic series $H_n = \ln(n) + \gamma + 1/(2n) - 1/(12n^2) + 1/(120n^4)$
- $\gamma = 0.57721566490153286060651209008240243104215$
- 格雷碼 $= n \oplus (n >> 1)$
- $SG(A+B) = SG(A) \oplus SG(B)$
- 旋轉矩陣 $M(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

4.1.6 基本數論

1. $\sum_{d|n} \mu(n) = [n == 1]$
2. $g(m) = \sum_{d|m} f(d) \Leftrightarrow f(m) = \sum_{d|m} \mu(d) \times g(m/d)$
3. $\sum_{i=1}^n \sum_{j=1}^m \text{互質數} = \sum \mu(d) \lfloor \frac{n}{d} \rfloor \lfloor \frac{m}{d} \rfloor$
4. $\sum_{i=1}^n \sum_{j=1}^n \text{lcm}(i, j) = n \sum_{d|n} d \times \phi(d)$

4.1.7 排組公式

1. k 卡特 $\frac{C_n^{kn}}{n(k-1)+1}, C_m^n = \frac{n!}{m!(n-m)!}$
2. $H(n, m) \cong x_1 + x_2 + \dots + x_n = k, num = C_k^{n+k-1}$
3. Stirling number of $2^{nd}, n$ 人分 k 組方法數目
 - (a) $S(0, 0) = S(n, n) = 1$
 - (b) $S(n, 0) = 0$
 - (c) $S(n, k) = kS(n-1, k) + S(n-1, k-1)$
4. Bell number, n 人分任意多組方法數目
 - (a) $B_0 = 1$
 - (b) $B_n = \sum_{i=0}^n S(n, i)$
 - (c) $B_{n+1} = \sum_{k=0}^n C_n^k B_k$
 - (d) $B_{p+n} \equiv B_n + B_{n+1} \pmod{p}$, p is prime
 - (e) $B_{p^m+n} \equiv mB_n + B_{n+1} \pmod{p}$, p is prime
 - (f) From $B_0: 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975$
5. Derangement, 錯排, 沒有人在自己位置上
 - (a) $D_n = n!(1 - \frac{1}{1!} + \frac{1}{2!} - \frac{1}{3!} + \dots + (-1)^n \frac{1}{n!})$
 - (b) $D_n = (n-1)(D_{n-1} + D_{n-2}), D_0 = 1, D_1 = 0$
 - (c) From $D_0: 1, 0, 1, 2, 9, 44, 265, 1854, 14833, 133496$

6. Binomial Equality

- (a) $\sum_k \binom{r}{m+k} \binom{s}{n-k} = \binom{r+s}{m+n}$
- (b) $\sum_k \binom{l}{m+k} \binom{s}{n+k} = \binom{l+s}{l-m+n}$
- (c) $\sum_k \binom{l}{m+k} \binom{s+k}{n-k} (-1)^k = (-1)^{l+m} \binom{s-m}{l-n-m}$
- (d) $\sum_{k \leq l} \binom{l-k}{m} \binom{s}{k-n} (-1)^k = (-1)^{l+m} \binom{s-m-1}{l-n-m}$
- (e) $\sum_{0 \leq k \leq l} \binom{l-k}{m} \binom{q+k}{n} = \binom{l+q+1}{m+n+1}$
- (f) $\binom{r}{k} = (-1)^k \binom{r-k}{k}$
- (g) $\binom{r}{m} \binom{r}{k} = \binom{r}{m-k} \binom{r-k}{k}$
- (h) $\sum_{k \leq n} \binom{r+k}{k} = \binom{r+n+1}{n}$
- (i) $\sum_{0 \leq k \leq n} \binom{r}{k} = \binom{n+1}{m+1}$
- (j) $\sum_{k \leq m} \binom{m+r}{k} x^k y^{m-k} = \sum_{k \leq m} \binom{-r}{k} (-x)^k (x+y)^{m-k}$

4.1.8 冪次, 冪次和

1. $a^b \% P = a^{b \% (p) + \varphi(p)}, b \geq \varphi(p)$
2. $1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
3. $1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
4. $1^5 + 2^5 + 3^5 + \dots + n^5 = \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}$
5. $0^k + 1^k + 2^k + \dots + n^k = P(k), P(k) = \frac{(n+1)^{k+1} - \sum_{i=0}^{k-1} C_i^{k+1} P(i)}{k+1}, P(0) = n+1$
6. $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1} \sum_{k=0}^n C_k^{n+1} B_k m^{n+1-k}$

7. $\sum_{j=0}^m C_j^{m+1} B_j = 0, B_0 = 1$
8. 除 $B_1 = -1/2$, 剩下的奇數項都是 0
9. $B_2 = 1/6, B_4 = -1/30, B_6 = 1/42, B_8 = -1/30, B_{10} = 5/66, B_{12} = -691/2730, B_{14} = 7/6, B_{16} = -3617/510, B_{18} = 43867/798, B_{20} = -174611/330,$

4.1.9 Burnside's lemma

1. $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$
2. $X^g = t^{c(g)}$
3. G 表示有幾種轉法, X^g 表示在那種轉法下, 有幾種是會保持對稱的, t 是顏色數, $c(g)$ 是循環節不動的面數。
4. 正立方體塗三顏色, 轉 0 有 3^6 個元素不變, 轉 90 有 6 種, 每種有 3^3 不變, 180 有 3×3^4 , 120 (角) 有 8×3^2 , 180 (邊) 有 6×3^3 , 全部 $\frac{1}{24} (3^6 + 6 \times 3^3 + 3 \times 3^4 + 8 \times 3^2 + 6 \times 3^3) = 57$

4.1.10 Count on a tree

1. Rooted tree: $s_{n+1} = \frac{1}{n} \sum_{i=1}^n (i \times a_i \times \sum_{j=1}^{\lfloor n/i \rfloor} a_{n+1-i \times j})$
2. Unrooted tree:
 - (a) Odd: $a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$
 - (b) Even: $Odd + \frac{1}{2} a_{n/2} (a_{n/2} + 1)$
3. Spanning Tree
 - (a) 完全圖 n^{n-2}
 - (b) 一般圖 (Kirchhoff's theorem) $M[i][i] = \text{degree}(V_i), M[i][j] = -1, \text{if have } E(i, j), 0 \text{ if no edge.}$
delete any one row and col in A , $ans = \det(A)$

5 Geometry

5.1 Sort by Angle

```

1 bool cmp(pii a, pii b) {
2   #define is_neg(k) (k.y < 0 || (k.y == 0 && k.x < 0));
3   int A = is_neg(a), B = is_neg(b);
4   if (A != B)
5     return A < B;
6   if (cross(a, b) == 0)
7     return (a.x*a.x + a.y*a.y) < (b.x*b.x + b.y*b.y);
8   return cross(a, b) > 0;
9 }

```

5.2 Geometry

```

1 const double PI=atan2(0.0,-1.0);
2 template<typename T>
3 struct point{
4   T x,y;
5   point() {}
6   point(const T&x, const T&y):x(x),y(y) {}
7   point operator+(const point &b) const {

```

```

      return point(x+b.x,y+b.y); }
9   point operator-(const point &b) const {
10    return point(x-b.x,y-b.y); }
11   point operator*(const T &b) const {
12    return point(x*b,y*b); }
13   point operator/(const T &b) const {
14    return point(x/b,y/b); }
15   bool operator==(const point &b) const {
16    return x==b.x&&y==b.y; }
17   T dot(const point &b) const {
18    return x*b.x+y*b.y; }
19   T cross(const point &b) const {
20    return x*b.y-y*b.x; }
21   point normal() const { //求法向
22    return point(-y,x); }
23   T abs2() const { //向 長度的平方
24    return dot(*this); }
25   T rad(const point &b) const { // 向 的弧度
26   return fabs(atan2(fabs(cross(b)),dot(b))); }
27   T getA() const { //對x軸的弧度
28     T A=atan2(y,x); //超過180度會變負的
29     if (A<=-PI/2) A+=PI*2;
30     return A;
31   }
32 };
33 template<typename T>
34 struct line{
35   line() {}
36   point<T> p1,p2;
37   T a,b,c; //ax+by+c=0
38   line(const point<T>&x, const point<T>&y):p1(x),p2(y) {}
39   void pton() { //轉成一般式
40     a=p1.y-p2.y;
41     b=p2.x-p1.x;
42     c=-a*p1.x-b*p1.y;
43   }
44   T ori(const point<T> &p) const { //點和有向直線的關係, >0左
45     邊, =0在線上, <0右邊
46     return (p2-p1).cross(p-p1);
47   }
48   T btw(const point<T> &p) const { //點投影 在線段上 <=0
49     return (p1-p).dot(p2-p);
50   }
51   bool point_on_segment(const point<T>&p) const { //點是否在线段
52     上
53     return ori(p)==0&&btw(p)<=0;
54   }
55   T dis2(const point<T> &p, bool is_segment=0) const { //點跟直線
56     /線段的距離平方
57     point<T> v=p2-p1,v1=p-p1;
58     if (is_segment) {
59       point<T> v2=p-p2;
60       if (v.dot(v1)<=0) return v1.abs2();
61       if (v.dot(v2)>=0) return v2.abs2();
62     }
63     T tmp=v.cross(v1);
64     return tmp*tmp/v.abs2();
65   }
66   T seg_dis2(const line<T> &l) const { // 線段距離平方
67     return min({dis2(l.p1,1),dis2(l.p2,1),l.dis2(p1,1),l.dis2(p2,1)});
68   }
69 }

```

```

66 point<T> projection(const point<T> &p) const { // 點對直線的投影
67     point<T> n = (p2 - p1).normal();
68     return p - n * (p - p1).dot(n) / n.abs2();
69 }
70 point<T> mirror(const point<T> &p) const {
71     // 點對直線的鏡射，要先呼叫 pton 轉成一般式
72     point<T> R;
73     T d = a * b * b;
74     R.x = (b * b * p.x - a * a * p.x - 2 * a * b * p.y - 2 * a * c) / d;
75     R.y = (a * a * p.y - b * b * p.y - 2 * a * b * p.x - 2 * b * c) / d;
76     return R;
77 }
78 bool equal(const line &l) const { // 直線相等
79     return ori(1.p1) == 0 && ori(1.p2) == 0;
80 }
81 bool parallel(const line &l) const {
82     return (p1 - p2).cross(1.p1 - 1.p2) == 0;
83 }
84 bool cross_seg(const line &l) const {
85     return (p2 - p1).cross(1.p1 - p1) * (p2 - p1).cross(1.p2 - p1) <= 0;
86     // 直線是否交線段
87 }
88 int line_intersect(const line &l) const { // 直線相交情況，-1 無
89     // 限多點、1 交於一點、0 不相交
90     return parallel(1) ? (ori(1.p1) == 0 ? -1 : 0) : 1;
91 }
92 int seg_intersect(const line &l) const {
93     T c1 = ori(1.p1), c2 = ori(1.p2);
94     T c3 = l.ori(p1), c4 = l.ori(p2);
95     if (c1 == 0 && c2 == 0) { // 共線
96         bool b1 = btw(1.p1) >= 0, b2 = btw(1.p2) >= 0;
97         T a3 = l.btw(p1), a4 = l.btw(p2);
98         if (b1 && b2 && a3 == 0 && a4 >= 0) return 2;
99         if (b1 && b2 && a3 >= 0 && a4 == 0) return 3;
100         if (b1 && b2 && a3 >= 0 && a4 >= 0) return 0;
101         return -1; // 無限交點
102     } else if (c1 * c2 <= 0 && c3 * c4 <= 0) return 1;
103     return 0; // 不相交
104 }
105 point<T> line_intersection(const line &l) const { // 直線交點
106     point<T> a = p2 - p1, b = 1.p2 - 1.p1, s = 1.p1 - p1;
107     // if (a.cross(b) == 0) return INF;
108     return p1 + a * (s.cross(b) / a.cross(b));
109 }
110 point<T> seg_intersection(const line &l) const { // 線段交點
111     int res = seg_intersect(1);
112     if (res <= 0) assert(0);
113     if (res == 2) return p1;
114     if (res == 3) return p2;
115     return line_intersection(1);
116 }
117 };
118 template<typename T>
119 struct polygon {
120     polygon() {}
121     vector<point<T>> > p; // 逆時針順序
122     T area() const { // 面積
123         T ans = 0;
124         for (int i = p.size() - 1, j = 0; j < (int)p.size(); i = j++)
125             ans += p[i].cross(p[j]);
126         return ans / 2;
127     }
128 };
129 point<T> center_of_mass() const { // 重心
130     T cx = 0, cy = 0, w = 0;
131     for (int i = p.size() - 1, j = 0; j < (int)p.size(); i = j++) {
132         T a = p[i].cross(p[j]);
133         cx += (p[i].x + p[j].x) * a;
134         cy += (p[i].y + p[j].y) * a;
135         w += a;
136     }
137     return point<T>(cx / 3 / w, cy / 3 / w);
138 }
139 char ahas(const point<T> &t) const { // 點是否在簡單多邊形內，
140     // 是的話回傳 1、在邊上回傳 -1、否則回傳 0
141     bool c = 0;
142     for (int i = 0, j = p.size() - 1; i < p.size(); j = i++)
143         if (line<T>(p[i], p[j]).point_on_segment(t)) return -1;
144     else if ((p[i].y > t.y) != (p[j].y > t.y) &&
145             t.x < (p[j].x - p[i].x) * (t.y - p[i].y) / (p[j].y - p[i].y) + p[i].x)
146         )
147         c = !c;
148     return c;
149 }
150 char point_in_convex(const point<T> &x) const {
151     int l = 1, r = (int)p.size() - 2;
152     while (l < r) { // 點是否在凸多邊形內，是的話回傳 1、在邊上回傳
153         // -1、否則回傳 0
154         int mid = (l + r) / 2;
155         T a1 = (p[mid] - p[0]).cross(x - p[0]);
156         T a2 = (p[mid + 1] - p[0]).cross(x - p[0]);
157         if (a1 >= 0 && a2 <= 0) {
158             T res = (p[mid + 1] - p[mid]).cross(x - p[mid]);
159             return res > 0 ? 1 : (res >= 0 ? -1 : 0);
160         } else if (a1 < 0) r = mid - 1;
161         else l = mid + 1;
162     }
163     return 0;
164 }
165 vector<T> getA() const { // 凸包邊對 x 軸的夾角
166     vector<T> res; // 一定是遞增的
167     for (size_t i = 0; i < p.size(); i++)
168         res.push_back((p[(i + 1) % p.size()] - p[i]).getA());
169     return res;
170 }
171 bool line_intersect(const vector<T> &A, const line<T> &l)
172     const { // O(logN)
173     int f1 = upper_bound(A.begin(), A.end(), (1.p1 - 1.p2).getA()) -
174         A.begin();
175     int f2 = upper_bound(A.begin(), A.end(), (1.p2 - 1.p1).getA()) -
176         A.begin();
177     return l.cross_seg(line<T>(p[f1], p[f2]));
178 }
179 polygon cut(const line<T> &l) const { // 凸包對直線切割，得到直
180     // 線 l 左側的凸包
181     polygon ans;
182     for (int n = p.size(), i = n - 1, j = 0; j < n; i = j++) {
183         if (1.ori(p[i]) >= 0) {
184             ans.p.push_back(p[i]);
185             if (1.ori(p[j]) < 0)
186                 ans.p.push_back(1.line_intersection(line<T>(p[i], p[
187                     j]])));
188             } else if (1.ori(p[j]) > 0)
189                 ans.p.push_back(1.line_intersection(line<T>(p[i], p[
190                     j]])));
191         }
192     }
193     return ans;
194 }
195 }
196 static bool graham_cmp(const point<T> &a, const point<T> &b)
197     { // 凸包排序函數
198     return (a.x < b.x) || (a.x == b.x && a.y < b.y);
199 }
200 void graham(vector<point<T>> &s) { // 凸包
201     sort(s.begin(), s.end(), graham_cmp);
202     p.resize(s.size() + 1);
203     int m = 0;
204     for (size_t i = 0; i < s.size(); i++) {
205         while (m >= 2 && (p[m - 1] - p[m - 2]).cross(s[i] - p[m - 2]) <= 0) --m;
206         p[m++] = s[i];
207     }
208     for (int i = s.size() - 2, t = m + 1; i >= 0; --i) {
209         while (m >= t && (p[m - 1] - p[m - 2]).cross(s[i] - p[m - 2]) <= 0) --m;
210         p[m++] = s[i];
211     }
212     if (s.size() > 1) --m;
213     p.resize(m);
214 }
215 T diam() { // 直徑
216     int n = p.size(), t = 1;
217     T ans = 0;
218     p.push_back(p[0]);
219     for (int i = 0; i < n; i++) {
220         point<T> now = p[i + 1] - p[i];
221         while (now.cross(p[t + 1] - p[i]) > now.cross(p[t] - p[i])) t = (t
222             + 1) % n;
223         ans = max(ans, (p[i] - p[t]).abs2());
224     }
225     return p.pop_back(), ans;
226 }
227 T min_cover_rectangle() { // 最小覆蓋矩形
228     int n = p.size(), t = 1, r = 1, l = 1;
229     if (n < 3) return 0; // 也可以做最小周長矩形
230     T ans = 1e99;
231     p.push_back(p[0]);
232     for (int i = 0; i < n; i++) {
233         point<T> now = p[i + 1] - p[i];
234         while (now.cross(p[t + 1] - p[i]) > now.cross(p[t] - p[i])) t = (t
235             + 1) % n;
236         while (now.dot(p[r + 1] - p[i]) > now.dot(p[r] - p[i])) r = (r + 1) % n;
237         if (!l) l = r;
238         while (now.dot(p[l + 1] - p[i]) <= now.dot(p[l] - p[i])) l = (l + 1) % n;
239         T d = now.abs2();
240         T tmp = now.cross(p[t] - p[i]) * (now.dot(p[r] - p[i]) - now.dot(p[l] - p[i])) / d;
241         ans = min(ans, tmp);
242     }
243     return p.pop_back(), ans;
244 }
245 T dis2(polygon &p1) { // 凸包最近距離平方
246     vector<point<T>> > &P = p, &Q = p1.p;
247     int n = P.size(), m = Q.size(), l = 0, r = 0;
248     for (int i = 0; i < n; i++) if (P[i].y < P[l].y) l = i;
249     for (int i = 0; i < m; i++) if (Q[i].y < Q[r].y) r = i;
250     P.push_back(P[0]), Q.push_back(Q[0]);
251     T ans = 1e99;
252     for (int i = 0; i < n; i++) {
253         while ((P[l] - P[l + 1]).cross(Q[r + 1] - Q[r]) < 0) r = (r + 1) % m;
254         ans = min(ans, line<T>(P[l], P[l + 1]).seg_dis2(line<T>(Q[r], Q[r + 1])));
255         l = (l + 1) % n;
256     }
257     return P.pop_back(), Q.pop_back(), ans;
258 }

```



```

239 }
240 static char sign(const point<T>&t){
241     return (t.y==0?t.x:t.y)<0;
242 }
243 static bool angle_cmp(const line<T>& A,const line<T>& B){
244     point<T> a=A.p2-A.p1,b=B.p2-B.p1;
245     return sign(a)<sign(b)|| (sign(a)==sign(b)&&a.cross(b)>0);
246 }
247 int halfplane_intersection(vector<line<T> > &s){//半平面交
248     sort(s.begin(),s.end(),angle_cmp);//線段左側為該線段半
249     面
250     int L,R,n=s.size();
251     vector<point<T> > px(n);
252     vector<line<T> > q(n);
253     q[L=R=0]=s[0];
254     for(int i=1;i<n;++i){
255         while(L<R&&s[i].ori(px[R-1])<=0)--R;
256         while(L<R&&s[i].ori(px[L])<=0)++L;
257         q[++R]=s[i];
258         if(q[R].parallel(q[R-1])){
259             --R;
260             if(q[R].ori(s[i].p1)>0)q[R]=s[i];
261         }
262         if(L<R)px[R-1]=q[R-1].line_intersection(q[R]);
263     }
264     while(L<R&&q[L].ori(px[R-1])<=0)--R;
265     p.clear();
266     if(R-L==1) return 0;
267     px[R]=q[R].line_intersection(q[L]);
268     for(int i=L;i<=R;++i)p.push_back(px[i]);
269     return R-L+1;
270 };
271 template<typename T>
272 struct triangle{
273     point<T> a,b,c;
274     triangle(){}
275     triangle(const point<T> &a,const point<T> &b,const point<T>
276         &c):a(a),b(b),c(c){}
277     T area() const{
278         T t=(b-a).cross(c-a)/2;
279         return t>0?t:-t;
280     }
281     point<T> barycenter() const{//重心
282         return (a+b+c)/3;
283     }
284     point<T> circumcenter() const{//外心
285         static line<T> u,v;
286         u.p1=(a+b)/2;
287         u.p2=point<T>(u.p1.x-a.y+b.y,u.p1.y+a.x-b.x);
288         v.p1=(a+c)/2;
289         v.p2=point<T>(v.p1.x-a.y+c.y,v.p1.y+a.x-c.x);
290         return u.line_intersection(v);
291     }
292     point<T> incenter() const{//內心
293         T A=sqrt((b-c).abs2()),B=sqrt((a-c).abs2()),C=sqrt((a-b).
294             abs2());
295         return point<T>(A*a.x+B*b.x+C*c.x,A*a.y+B*b.y+C*c.y)/(A+B
296             +C);
297     }
298     point<T> perpencenter() const{//垂心
299         return barycenter()*3-circumcenter()*2;
300     }
301 };
302 template<typename T>
303 struct point3D{
304     T x,y,z;
305     point3D(){}
306     point3D(const T&x,const T&y,const T&z):x(x),y(y),z(z){}
307     point3D operator+(const point3D &b) const{
308         return point3D(x+b.x,y+b.y,z+b.z);
309     }
310     point3D operator-(const point3D &b) const{
311         return point3D(x-b.x,y-b.y,z-b.z);
312     }
313     point3D operator*(const T &b) const{
314         return point3D(x*b,y*b,z*b);
315     }
316     point3D operator/(const T &b) const{
317         return point3D(x/b,y/b,z/b);
318     }
319     bool operator==(const point3D &b) const{
320         return x==b.x&&y==b.y&&z==b.z;
321     }
322     T dot(const point3D &b) const{
323         return x*b.x+y*b.y+z*b.z;
324     }
325     point3D cross(const point3D &b) const{
326         return point3D(y*b.z-z*b.y,z*b.x-x*b.z,x*b.y-y*b.x);
327     }
328     T abs2() const{//向 長度的平方
329         return dot(*this);
330     }
331     T area2(const point3D &b) const{//和b、原點圍成面積的平方
332         return cross(b).abs2()/4;
333     }
334 };
335 template<typename T>
336 struct line3D{
337     point3D<T> p1,p2;
338     line3D(){}
339     line3D(const point3D<T> &p1,const point3D<T> &p2):p1(p1),p2
340         (p2){}
341     T dis2(const point3D<T> &p,bool is_segment=0) const{//點跟直
342         線/線段的距離平方
343         point3D<T> v=p2-p1,v1=p-p1;
344         if(is_segment){
345             point3D<T> v2=p-p2;
346             if(v.dot(v1)<=0) return v1.abs2();
347             if(v.dot(v2)>=0) return v2.abs2();
348         }
349         point3D<T> tmp=v.cross(v1);
350         return tmp.abs2()/v.abs2();
351     }
352     pair<point3D<T>,point3D<T> > closest_pair(const line3D<T> &
353         l) const{
354         point3D<T> v1=(p1-p2),v2=(l.p1-l.p2);
355         point3D<T> N=v1.cross(v2),ab(p1-l.p1);
356         //if(N.abs2()==0) return NULL;平 或重合
357         T tmp=N.dot(ab),ans=tmp*tmp/N.abs2(); //最近點對距離
358         point3D<T> d1=p2-p1,d2=l.p2-l.p1,D=d1.cross(d2),G=l.p1-p1
359             ;
360         T t1=(G.cross(d2)).dot(D)/D.abs2();
361         T t2=(G.cross(d1)).dot(D)/D.abs2();
362         return make_pair(p1+d1*t1,l.p1+d2*t2);
363     }
364     bool same_side(const point3D<T> &a,const point3D<T> &b)
365         const{
366         return (p2-p1).cross(a-p1).dot((p2-p1).cross(b-p1))>0;
367     }
368 };
369 template<typename T>
370 struct plane{
371     point3D<T> p0,n;//平面上的點和法向
372     plane(){}
373     plane(const point3D<T> &p0,const point3D<T> &n):p0(p0),n(n)
374     {}
375     T dis2(const point3D<T> &p) const{//點到平面距離的平方
376         T tmp=(p-p0).dot(n);
377         return tmp*tmp/n.abs2();
378     }
379     point3D<T> projection(const point3D<T> &p) const{
380         return p-n*(p-p0).dot(n)/n.abs2();
381     }
382     point3D<T> line_intersection(const line3D<T> &l) const{
383         T tmp=n.dot(l.p2-l.p1);//等於0表示平 或重合該平面
384         return l.p1+(l.p2-l.p1)*(n.dot(p0-l.p1)/tmp);
385     }
386     line3D<T> plane_intersection(const plane &p1) const{
387         point3D<T> e=n.cross(p1.n),v=n.cross(e);
388         T tmp=p1.n.dot(v);//等於0表示平 或重合該平面
389         point3D<T> q=p0+(v*(p1.n.dot(p1.p0-p0))/tmp);
390         return line3D<T>(q,q+e);
391     }
392 };
393 template<typename T>
394 struct triangle3D{
395     point3D<T> a,b,c;
396     triangle3D(){}
397     triangle3D(const point3D<T> &a,const point3D<T> &b,const
398         point3D<T> &c):a(a),b(b),c(c){}
399     bool point_in(const point3D<T> &p) const{//點在該平面上的投
400         影在三角形中
401         return line3D<T>(b,c).same_side(p,a)&&line3D<T>(a,c).
402             same_side(p,b)&&line3D<T>(a,b).same_side(p,c);
403     }
404 };
405 template<typename T>
406 struct tetrahedron{//四面體
407     point3D<T> a,b,c,d;
408     tetrahedron(){}
409     tetrahedron(const point3D<T> &a,const point3D<T> &b,const
410         point3D<T> &c,const point3D<T> &d):a(a),b(b),c(c),d(d)
411     {}
412     T volume6() const{//體積的六倍
413         return (d-a).dot((b-a).cross(c-a));
414     }
415     point3D<T> centroid() const{
416         return (a+b+c+d)/4;
417     }
418     bool point_in(const point3D<T> &p) const{
419         return triangle3D<T>(a,b,c).point_in(p)&&triangle3D<T>(c,
420             d,a).point_in(p);
421     }
422 };
423 template<typename T>
424 struct convexhull3D{
425     static const int MAXN=1005;
426     struct face{
427         int a,b,c;
428         face(int a,int b,int c):a(a),b(b),c(c){}
429     };
430     vector<point3D<T>> pt;
431     vector<face> ans;
432     int fid[MAXN][MAXN];
433     void build(){
434         int n=pt.size();
435         ans.clear();
436         memset(fid,0,sizeof(fid));
437         ans.emplace_back(0,1,2);//注意不能共線
438         ans.emplace_back(2,1,0);
439         int ftop = 0;

```

```

416 for(int i=3, ftop=1; i<n; ++i, ++ftop){
417     vector<face> next;
418     for(auto &f:ans){
419         T d=(pt[i]-pt[f.a]).dot((pt[f.b]-pt[f.a]).cross(pt[f.
420             c]-pt[f.a]));
421         if(d<=0) next.push_back(f);
422         int ff=0;
423         if(d>0) ff=ftop;
424         else if(d<0) ff=-ftop;
425         fid[f.a][f.b]=fid[f.b][f.c]=fid[f.c][f.a]=ff;
426     }
427     for(auto &f:ans){
428         if(fid[f.a][f.b]>0 && fid[f.a][f.b]!=fid[f.b][f.a])
429             next.emplace_back(f,a,f.b,i);
430         if(fid[f.b][f.c]>0 && fid[f.b][f.c]!=fid[f.c][f.b])
431             next.emplace_back(f,b,f.c,i);
432         if(fid[f.c][f.a]>0 && fid[f.c][f.a]!=fid[f.a][f.c])
433             next.emplace_back(f,c,f.a,i);
434     }
435     ans=next;
436 }
437 point3D<T> centroid()const{
438     point3D<T> res(0,0,0);
439     T vol=0;
440     for(auto &f:ans){
441         T tmp=pt[f.a].dot(pt[f.b].cross(pt[f.c]));
442         res=res+(pt[f.a]+pt[f.b]+pt[f.c])*tmp;
443         vol+=tmp;
444     }
445     return res/(vol*4);
446 }
447 };

```

5.3 Convex Hull

```

1 using pdd = pair<double, double>;
2 #define F first
3 #define S second
4 pdd operator-(pdd a, pdd b) {
5     return {a.F - b.F, a.S - b.S};
6 }
7 double cross(pdd a, pdd b) {
8     return a.F * b.S - a.S * b.F;
9 }
10 void solve() {
11     int n;
12     cin >> n;
13     vector<pdd> pnts;
14     for (int i = 0; i < n; ++i) {
15         double x, y;
16         cin >> x >> y;
17         pnts.push_back(x, y);
18     }
19     sort(iter(pnts));
20     vector<pdd> hull;
21     for (int i = 0; i < 2; ++i) {
22         int t = hull.size();
23         for (pdd j: pnts) {
24             while(hull.size() - t >= 2 && cross(j - hull[hull.size()
25                 () - 2], hull.back() - hull[hull.size() - 2]) >=
                0)
                hull.pop_back();

```

```

26         hull.push_back(j);
27     }
28     hull.pop_back();
29     reverse(iter(pnts));
30 }
31 double area = 0;
32 for (int i=0; i < hull.size(); ++i){
33     area += cross(hull[i], hull[(i + 1) % hull.size()]);
34 }
35 area /= 2.0;
36 }

```

5.4 Min Covering Circle

```

1 double dis(pdd a, pdd b) {
2     double dx = a.x - b.x, dy = a.y - b.y;
3     return sqrt(dx*dx + dy*dy);
4 }
5 double sq(double x) {
6     return x * x;
7 }
8 pdd excenter(pdd p1, pdd p2, pdd p3) {
9     double a1 = p1.x - p2.x, a2 = p1.x - p3.x;
10    double b1 = p1.y - p2.y, b2 = p1.y - p3.y;
11    double c1 = (sq(p1.x) - sq(p2.x) + sq(p1.y) - sq(p2.y)) /
12        2;
13    double c2 = (sq(p1.x) - sq(p3.x) + sq(p1.y) - sq(p3.y)) /
14        2;
15    double dd = a1*b2 - a2*b1;
16    return {(c1*b2 - c2*b1) / dd, (a1*c2 - a2*c1) / dd};
17 }
18 void solve(pdd a[], int n) {
19     shuffle(a, a + n, rng);
20     pdd center = a[0];
21     double r = 0;
22     for (int i = 1; i < n; ++i) {
23         if (dis(center, a[i]) <= r) continue;
24         center = a[i], r = 0;
25         for (int j = 0; j < i; ++j) {
26             if (dis(center, a[j]) <= r) continue;
27             center.x = (a[i].x + a[j].x) / 2;
28             center.y = (a[i].y + a[j].y) / 2;
29             r = dis(center, a[i]);
30             for (int k = 0; k < j; ++k) {
31                 if (dis(center, a[k]) <= r) continue;
32                 center = excenter(a[i], a[j], a[k]);
33                 r = dis(center, a[i]);
34             }
35         }
36     }
37     cout << fixed << setprecision(10) << r << '\n';
38     cout << center.x << ' ' << center.y << '\n';
39 }

```

5.5 Point in Polygon

```

1 const ll inf = 2000000000;
2 struct Point {
3     ll x, y;
4     Point(ll x = 0, ll y = 0):x(x), y(y) {}

```

```

5 Point operator+(const Point p) const {
6     return Point(x + p.x, y + p.y);
7 }
8 Point operator-(const Point p) const {
9     return Point(x - p.x, y - p.y);
10 }
11 ll operator*(const Point p) const { //dot
12     return x * p.x + y * p.y;
13 }
14 ll operator^(const Point p) const { //cross
15     return x * p.y - y * p.x;
16 }
17 bool onseg(Point a, Point b, Point o) {
18     return ((a - o) ^ (b - o)) == 0 && ((a - o) * (b - o)) <=
19         0;
20 }
21 int ori(Point a, Point b, Point o) {
22     ll w = (a - o) ^ (b - o);
23     return (w > 0 ? 1 : -1) : 0;
24 }
25 bool inters(Point a, Point b, Point c, Point d) {
26     if (onseg(a, b, c) || onseg(a, b, d)) return 1;
27     if (onseg(c, d, a) || onseg(c, d, b)) return 1;
28     if (ori(a, b, c) * ori(a, b, d) < 0 && ori(c, d, a) * ori(c,
29         d, b) < 0) return 1;
30     return 0;
31 }
32 Point poly[maxn];
33 void solve(int n, Point p) {
34     poly[n] = poly[0];
35     int cnt = 0;
36     for (int i = 0; i < n; ++i) {
37         if (onseg(poly[i], poly[i + 1], p)) {
38             cnt = -1;
39             break;
40         }
41         if (inters(poly[i], poly[i + 1], p, Point(inf, p.y))) {
42             ++cnt;
43         }
44         Point hi = (poly[i].y > poly[i + 1].y ? poly[i] : poly[i
45             + 1]);
46         if (hi.y == p.y && hi.x > p.x) {
47             --cnt;
48         }
49     }
50     if (cnt < 0)
51         cout << "BOUNDARY\n";
52     else if (cnt % 2)
53         cout << "INSIDE\n";
54     else
55         cout << "OUTSIDE\n";
56 }

```

6 Graph

6.1 Bipartite Matching

```

1 const int MAXN = 100;
2
3 struct Bipartite_matching{
4     int mx[MAXN], my[MAXN], vy[MAXN]; //matchX, matchY,
5     vector<int> edge[MAXN]; //adjacent list;
6     int x_cnt;

```



```

7 bool dfs(int x){
8     for(auto y: edge[x]){ //對 x 可以碰到的邊進 檢查
9         if(vy[y] == 1) continue; //避免遞迴 error
10
11         vy[y] = 1;
12         if(my[y] == -1 || dfs(my[y])){ //分析 3
13             mx[x] = y;
14             my[y] = x;
15             return true;
16         }
17     }
18     return false; //分析 4
19 }
20
21 int bipartite_matching(){
22     memset(mx, -1, sizeof(mx)); //分析 1,2
23     memset(my, -1, sizeof(my));
24     int ans = 0;
25     for(int i = 0; i < x_cnt; i++){ //對每一個 x 節點進
26         DFS(最大匹配)
27         memset(vy, 0, sizeof(vy));
28         if(dfs(i)) ans++;
29     }
30     return ans;
31 }
32 vector<vector<int>> get_match(){
33     vector<vector<int>> res;
34     for(int i = 0; i < x_cnt; i++){
35         if(mx[i] != -1){
36             res.push_back({i, mx[i]});
37         }
38     }
39     return res;
40 }
41 void add_edge(int i, int j){
42     edge[i].push_back(j);
43 }
44 void init(int x){
45     x_cnt = x;
46 }
47 };
48 int main(){
49     int n, m;
50     Bipartite_matching bm;
51     for(int i = 0; i < m; i++){
52         int a, b; cin >> a >> b;
53         bm.add_edge(a, b);
54     }
55     bm.init(n);
56     cout << bm.bipartite_matching() << endl;
57     auto match = bm.get_match();
58     for(auto t: match){
59         cout << t[0] << " " << t[1] << endl;
60     }
61 }

```

6.2 Tarjan SCC

```

1 const int n = 16;
2 vector<vector<int>> graph;
3 int visit[n], low[n], t = 0;

```

```

4 int st[n], top = 0;
5 bool instack[n];
6 int contract[n]; // 每個點收縮到的點
7 vector<vector<int>> block;
8 void dfs(int x, int parent){
9     // cout << x << endl;
10    visit[x] = low[x] = ++t;
11    st[top++] = x;
12    instack[x] = true;
13    for(auto to: graph[x]){
14        if(!visit[to])
15            dfs(to, x);
16
17        if(instack[to])
18            low[x] = min(low[x], low[to]);
19    }
20    if(visit[x] == low[x]){ //scc 裡最早拜訪的
21        int j;
22        block.push_back({});
23        do{
24            j = st[--top];
25            instack[j] = false;
26            block[block.size() - 1].push_back(j);
27            contract[j] = x;
28        }while(j != x);
29    }
30 }
31 int main(){
32     for(int i = 0; i < n; i++){
33         if (!visit[i])
34             dfs(i, i);
35     }
36     for(auto t: block){
37         for(auto x: t){
38             cout << x << " ";
39         } cout << endl;
40     }
41 }

```

6.3 Bridge

```

1 const int n = 9;
2 vector<vector<int>> graph;
3 vector<int> visit(n, 0);
4 vector<int> trace(n, 0);
5 vector<vector<int>> bridge;
6 int t = 0;
7 void dfs(int x, int parent){
8     visit[x] = ++t;
9     trace[x] = x; // 最高祖先預設為自己
10    for (auto to : graph[x]){
11        if (visit[to]) { // back edge
12            if (to != parent){
13                trace[x] = to;
14            }
15        }
16        else { // tree edge
17            dfs(to, x);
18            if (visit[trace[to]] < visit[trace[x]])
19                trace[x] = trace[to];
20        }
21    }

```

// 子樹回不到祖先暨自身。

```

22         if (visit[trace[to]] > visit[x])
23             bridge.push_back({x, to});
24     }
25 }
26 //call for()dfs(i, -1)
27 int main(){
28     for(int i = 0; i < 9; i++){
29         if(!visit[i])
30             dfs(i, -1);
31     }
32     for(auto x: bridge){
33         cout << x[0] << " " << x[1] << endl;
34     }
35 }

```

6.4 2 SAT

```

1 class TwoSAT{
2 public:
3     TwoSAT(int n) : n(n), graph(2 * n), visited(2 * n, false)
4     {}
5     void addClause(int a, int b) { // 0-base;
6         a *= 2;
7         b *= 2;
8         // Add implications (~a => b) and (~b => a)
9         graph[a ^ 1].push_back(b);
10        graph[b ^ 1].push_back(a);
11    }
12    bool solve() { // Find SCCs and check for contradictions
13        for (int i = 0; i < 2 * n; ++i) {
14            if (!visited[i]) {
15                dfs1(i);
16            }
17        }
18        reverse(processingOrder.begin(), processingOrder.end());
19        //topological sort
20        for (int i = 0; i < 2 * n; ++i) {
21            visited[i] = false;
22        }
23        for (int node : processingOrder) {
24            if (!visited[node]) {
25                scc.clear();
26                dfs2(node);
27                if (!checkSCCConsistency()) {
28                    return false;
29                }
30            }
31        }
32        return true;
33    }
34 private:
35     int n;
36     vector<vector<int>> graph;
37     vector<bool> visited;
38     vector<int> processingOrder;
39     vector<int> scc;
40
41     void dfs1(int node) {
42         visited[node] = true;
43         for (int neighbor : graph[node]) {
44             if (!visited[neighbor]) {

```

```

45         dfs1(neighbor);
46     }
47 }
48 processingOrder.push_back(node);
49 }
50
51 void dfs2(int node) {
52     visited[node] = true;
53     scc.push_back(node);
54     for (int neighbor : graph[node]) {
55         if (!visited[neighbor]) {
56             dfs2(neighbor);
57         }
58     }
59 }
60
61 bool checkSCCConsistency() {
62     for (int node : scc) {
63         if (find(scc.begin(), scc.end(), node ^ 1) != scc
64             .end()) {
65             return false; // Contradiction found in the
66                             same SCC
67         }
68     }
69     return true;
70 }
71
72 int main() {
73     int n, m; // Number of variables and clauses
74     TwoSAT twoSat(n);
75     for (int i = 0; i < m; ++i) {
76         int a, b;
77         twoSat.addClause(a, b);
78     }
79     if (twoSat.solve()) {
80         cout << "Satisfiable" << endl;
81     } else {
82         cout << "Unsatisfiable" << endl;
83     }
84 }

```

6.5 Kosaraju 2DFS

```

1 const int n = 16;
2 vector<vector<int>> graph;
3 vector<vector<int>> reverse_graph;
4 int visit[n];
5 int contract[n]; // 每個點收縮到的點
6 vector<vector<int>> block;
7 vector<int> finish; // fake topological sort
8 // need graph and reverse graph
9 void dfs1(int x) {
10     visit[x] = true;
11     for (auto to : graph[x]) {
12         if (!visit[to]) {
13             dfs1(to);
14         }
15     }
16     finish.push_back(x);
17 }
18 void dfs2(int x, int c) {
19     contract[x] = c;
20     block[c].push_back(x);

```

```

21     visit[x] = true;
22     for (auto to : reverse_graph[x]) {
23         if (!visit[to]) {
24             dfs2(to, c);
25         }
26     }
27 }
28 int main() {
29     graph = {};
30     reverse_graph = {};
31
32     for (int i = 0; i < n; ++i) {
33         if (!visit[i]) {
34             dfs1(i);
35         }
36         int c = 0;
37         memset(visit, 0, sizeof(visit));
38         for (int i = n - 1; i >= 0; i--) {
39             if (!visit[finish[i]]) {
40                 block.push_back({});
41                 dfs2(finish[i], c++);
42             }
43         }
44         for (auto t : block) {
45             for (auto x : t) {
46                 cout << x << " ";
47             }
48             cout << endl;
49         }
50     }
51 }

```

6.6 Minimum Steiner Tree

```

1 // Minimum Steiner Tree
2 // O(V^3 AT + V^2 2^AT)
3 struct SteinerTree { // 0-base
4     static const int T = 10, N = 105, INF = 1e9;
5     int n, dst[N][N], dp[1 << T][N], tdst[N];
6     int vcost[N]; // the cost of vertices
7     void init(int _n) {
8         n = _n;
9         for (int i = 0; i < n; ++i) {
10             for (int j = 0; j < n; ++j) dst[i][j] = INF;
11             dst[i][i] = vcost[i] = 0;
12         }
13     }
14     void add_edge(int ui, int vi, int wi) {
15         dst[ui][vi] = min(dst[ui][vi], wi);
16     }
17     void shortest_path() {
18         for (int k = 0; k < n; ++k)
19             for (int i = 0; i < n; ++i)
20                 for (int j = 0; j < n; ++j)
21                     dst[i][j] =
22                         min(dst[i][j], dst[i][k] + dst[k][j]);
23     }
24     int solve(const vector<int> &ter) {
25         shortest_path();
26         int t = SZ(ter);
27         for (int i = 0; i < (1 << t); ++i)
28             for (int j = 0; j < n; ++j) dp[i][j] = INF;
29         for (int i = 0; i < n; ++i) dp[0][i] = vcost[i];
30         for (int msk = 1; msk < (1 << t); ++msk) {
31             if (!(msk & (msk - 1))) {

```

```

32         int who = __lg(msk);
33         for (int i = 0; i < n; ++i)
34             dp[msk][i] =
35                 vcost[ter[who]] + dst[ter[who]][i];
36     }
37     for (int i = 0; i < n; ++i)
38         for (int submsk = (msk - 1) & msk; submsk;
39             submsk = (submsk - 1) & msk)
40             dp[msk][i] = min(dp[msk][i],
41                 dp[submsk][i] + dp[msk ^ submsk][i] -
42                 vcost[i]);
43     for (int i = 0; i < n; ++i) {
44         tdst[i] = INF;
45         for (int j = 0; j < n; ++j)
46             tdst[i] =
47                 min(tdst[i], dp[msk][j] + dst[j][i]);
48     }
49     for (int i = 0; i < n; ++i) dp[msk][i] = tdst[i];
50 }
51 int ans = INF;
52 for (int i = 0; i < n; ++i)
53     ans = min(ans, dp[(1 << t) - 1][i]);
54 return ans;
55 }
56 };

```

6.7 Dijkstra

```

1 #define maxn 200005
2 vector<int> dis(maxn, -1);
3 vector<int> parent(maxn, -1);
4 vector<bool> vis(maxn, false);
5 vector<vector<pair<int, int>>> graph;
6 void dijkstra(int source) {
7     dis[source] = 0;
8
9     priority_queue<pair<int, int>, vector<pair<int, int>>,
10         greater<pair<int, int>>> pq;
11     pq.push({0, source});
12     while (!pq.empty()) {
13         int from = pq.top().second;
14         pq.pop();
15         // cout << vis[from] << endl;
16         if (vis[from]) continue;
17         vis[from] = true;
18         for (auto next : graph[from]) {
19             int to = next.second;
20             int weight = next.first;
21             // cout << from << " " << to << " " << weight;
22             if (dis[from] + weight < dis[to] || dis[to] == -1) {
23                 dis[to] = dis[from] + weight;
24                 parent[to] = from;
25                 pq.push({dis[from] + weight, to});
26             }
27         }
28     }
29 }
30 int main() {
31     int startpoint;
32     dijkstra(startpoint);
33     // dis and parent
34 }

```

6.8 Maximum Clique Dyn

```

1 const int N = 150;
2 struct MaxClique { // Maximum Clique
3     bitset<N> a[N], cs[N];
4     int ans, sol[N], q, cur[N], d[N], n;
5     void init(int _n) {
6         n = _n;
7         for (int i = 0; i < n; i++) a[i].reset();
8     }
9     void addEdge(int u, int v) { a[u][v] = a[v][u] = 1; }
10    void csort(vector<int> &r, vector<int> &c) {
11        int mx = 1, km = max(ans - q + 1, 1), t = 0,
12            m = r.size();
13        cs[1].reset(), cs[2].reset();
14        for (int i = 0; i < m; i++) {
15            int p = r[i], k = 1;
16            while ((cs[k] & a[p]).count()) k++;
17            if (k > mx) mx++, cs[mx + 1].reset();
18            cs[k][p] = 1;
19            if (k < km) r[t++] = p;
20        }
21        c.resize(m);
22        if (t) c[t - 1] = 0;
23        for (int k = km; k <= mx; k++)
24            for (int p = cs[k]._Find_first(); p < N;
25                 p = cs[k]._Find_next(p))
26                r[t] = p, c[t] = k, t++;
27    }
28    void dfs(vector<int> &r, vector<int> &c, int l,
29            bitset<N> mask) {
30        while (!r.empty()) {
31            int p = r.back();
32            r.pop_back(), mask[p] = 0;
33            if (q + c.back() <= ans) return;
34            cur[q++] = p;
35            vector<int> nr, nc;
36            bitset<N> nmask = mask & a[p];
37            for (int i : r)
38                if (a[p][i]) nr.push_back(i);
39            if (!nr.empty()) {
40                if (l < 4) {
41                    for (int i : nr)
42                        d[i] = (a[i] & nmask).count();
43                    sort(nr.begin(), nr.end(),
44                        [&](int x, int y) { return d[x] > d[y]; });
45                }
46                csort(nr, nc), dfs(nr, nc, l + 1, nmask);
47            } else if (q > ans) ans = q, copy_n(cur, q, sol);
48            c.pop_back(), q--;
49        }
50    }
51    int solve(bitset<N> mask = bitset<N>(),
52            string(N, '1')) { // vertex mask
53        vector<int> r, c;
54        ans = q = 0;
55        for (int i = 0; i < n; i++)
56            if (mask[i]) r.push_back(i);
57        for (int i = 0; i < n; i++)
58            d[i] = (a[i] & mask).count();
59        sort(r.begin(), r.end(),
60            [&](int i, int j) { return d[i] > d[j]; });
61        csort(r, c), dfs(r, c, 1, mask);
62        return ans; // sol[0 ~ ans-1]
63    }

```

```

64 } graph;

```

6.9 Minimum Clique Cover

```

1 struct Clique_Cover { // 0-base, O(n2^Nn)
2     int co[1 << N], n, E[N];
3     int dp[1 << N];
4     void init(int _n) {
5         n = _n, fill_n(dp, 1 << n, 0);
6         fill_n(E, n, 0), fill_n(co, 1 << n, 0);
7     }
8     void add_edge(int u, int v) {
9         E[u] |= 1 << v, E[v] |= 1 << u;
10    }
11    int solve() {
12        for (int i = 0; i < n; ++i)
13            co[1 << i] = E[i] | (1 << i);
14        co[0] = (1 << n) - 1;
15        dp[0] = (n & 1) * 2 - 1;
16        for (int i = 1; i < (1 << n); ++i) {
17            int t = i & -i;
18            dp[i] = -dp[i ^ t];
19            co[i] = co[i ^ t] & co[t];
20        }
21        for (int i = 0; i < (1 << n); ++i)
22            co[i] = (co[i] & i) == i;
23        fwt(co, 1 << n);
24        for (int ans = 1; ans < n; ++ans) {
25            int sum = 0;
26            for (int i = 0; i < (1 << n); ++i)
27                sum += (dp[i] * co[i]);
28            if (sum) return ans;
29        }
30        return n;
31    }
32 };

```

6.10 Floyd Warshall

```

1 #define maxn 2005
2 vector<vector<int>> dis(maxn, vector<int>(maxn, 9999999));
3 vector<vector<int>> mid(maxn, vector<int>(maxn, -1));
4 vector<vector<pair<int, int>>> graph;
5
6 void floyd_warshall(int n) { // n is n nodes
7     for (int i = 0; i < n; i++) {
8         for (auto path : graph[i]) {
9             dis[i][path.second] = path.first;
10        }
11    }
12    for (int i = 0; i < n; i++)
13        dis[i][i] = 0;
14    for (int k = 0; k < n; k++) {
15        for (int i = 0; i < n; i++) {
16            for (int j = 0; j < n; j++) {
17                if (dis[i][k] + dis[k][j] < dis[i][j] || dis[i][j]
18                    ] == -1) {
19                    dis[i][j] = dis[i][k] + dis[k][j];
20                    mid[i][j] = k; // 由 i 點走到 j 點經過 k 點
21                }
22            }
23        }
24    }
25 }

```

```

20     }
21     }
22     }
23 }
24
25 void find_path(int s, int t) { // 印出最短徑
26     if (mid[s][t] == -1) return; // 沒有中繼點就結束
27     find_path(s, mid[s][t]); // 前半段最短徑
28     cout << mid[s][t]; // 中繼點
29     find_path(mid[s][t], t); // 後半段最短徑
30 }
31
32 int main() {
33     int n;
34     floyd_warshall(n);
35     for (int i = 0; i < 4; i++) {
36         for (int j = 0; j < 4; j++)
37             cout << dis[i][j] << " ";
38     }
39     find_path(0, 2);
40 }

```

6.11 Articulation Vertex

```

1 const int n = 9;
2 int t = 0;
3 vector<int> disc(n, -1); // Discovery time
4 vector<int> low(n, -1); // Low time
5 vector<int> parent_array(n, -1); // Parent in DFS tree
6 vector<bool> visited(n, false);
7 vector<bool> is_articulation(n, false);
8 vector<vector<int>> graph;
9 void dfs_articulation(int node, int parent) {
10     visited[node] = true;
11     disc[node] = t;
12     low[node] = t;
13     t++;
14     int children = 0;
15
16     for (int neighbor : graph[node])
17     {
18         if (!visited[neighbor])
19         {
20             children++;
21             parent_array[neighbor] = node;
22             dfs_articulation(neighbor, node);
23             low[node] = min(low[node], low[neighbor]);
24
25             if (low[neighbor] >= disc[node] && parent != -1)
26             {
27                 is_articulation[node] = true;
28             }
29         }
30         else if (neighbor != parent)
31         {
32             low[node] = min(low[node], disc[neighbor]);
33         }
34     }
35
36     if (parent == -1 && children > 1)
37     {
38         is_articulation[node] = true;
39     }
40 }

```

```

39 }
40 } //call for() dfs(i, -1)
41 int main() {
42     for (int i = 0; i < n; ++i) {
43         if (!visited[i]) {
44             dfs_articulation(i, -1);
45         }
46     }
47     cout << "Articulation Points: ";
48     for (int i = 0; i < n; ++i) {
49         if (is_articulation[i]) {
50             cout << i << " ";
51         }
52     } cout << endl;
53 }

```

6.12 Number of Maximal Clique

```

1 struct BronKerbosch { // 1-base
2     int n, a[N], g[N][N];
3     int S, all[N][N], some[N][N], none[N][N];
4     void init(int _n) {
5         n = _n;
6         for (int i = 1; i <= n; ++i)
7             for (int j = 1; j <= n; ++j) g[i][j] = 0;
8     }
9     void add_edge(int u, int v) {
10         g[u][v] = g[v][u] = 1;
11     }
12     void dfs(int d, int an, int sn, int nn) {
13         if (S > 1000) return; // pruning
14         if (sn == 0 && nn == 0) ++S;
15         int u = some[d][0];
16         for (int i = 0; i < sn; ++i) {
17             int v = some[d][i];
18             if (g[u][v]) continue;
19             int tsu = 0, tnn = 0;
20             copy_n(all[d], an, all[d + 1]);
21             all[d + 1][an] = v;
22             for (int j = 0; j < sn; ++j)
23                 if (g[v][some[d][j]])
24                     some[d + 1][tsu++] = some[d][j];
25             for (int j = 0; j < nn; ++j)
26                 if (g[v][none[d][j]])
27                     none[d + 1][tnn++] = none[d][j];
28             dfs(d + 1, an + 1, tsu, tnn);
29             some[d][i] = 0, none[d][nn++] = v;
30         }
31     }
32     int solve() {
33         iota(some[0], some[0] + n, 1);
34         S = 0, dfs(0, 0, n, 0);
35         return S;
36     }
37 };

```

6.13 DominatorTree

```

1 struct dominator_tree { // 1-base
2     vector<int> G[N], rG[N];

```

```

3     int n, pa[N], dfn[N], id[N], Time;
4     int semi[N], idom[N], best[N];
5     vector<int> tree[N]; // dominator_tree
6     void init(int _n) {
7         n = _n;
8         for (int i = 1; i <= n; ++i)
9             G[i].clear(), rG[i].clear();
10    }
11    void add_edge(int u, int v) {
12        G[u].pb(v), rG[v].pb(u);
13    }
14    void dfs(int u) {
15        id[dfn[u] = ++Time] = u;
16        for (auto v : G[u])
17            if (!dfn[v]) dfs(v), pa[dfn[v]] = dfn[u];
18    }
19    int find(int y, int x) {
20        if (y <= x) return y;
21        int tmp = find(pa[y], x);
22        if (semi[best[y]] > semi[best[pa[y]]])
23            best[y] = best[pa[y]];
24        return pa[y] = tmp;
25    }
26    void tarjan(int root) {
27        Time = 0;
28        for (int i = 1; i <= n; ++i) {
29            dfn[i] = idom[i] = 0;
30            tree[i].clear();
31            best[i] = semi[i] = i;
32        }
33        dfs(root);
34        for (int i = Time; i > 1; --i) {
35            int u = id[i];
36            for (auto v : rG[u])
37                if (v = dfn[v]) {
38                    find(v, i);
39                    semi[i] = min(semi[i], semi[best[v]]);
40                }
41            tree[semi[i]].pb(i);
42            for (auto v : tree[pa[i]]) {
43                find(v, pa[i]);
44                idom[v] =
45                    semi[best[v]] == pa[i] ? pa[i] : best[v];
46            }
47            tree[pa[i]].clear();
48        }
49        for (int i = 2; i <= Time; ++i) {
50            if (idom[i] != semi[i]) idom[i] = idom[idom[i]];
51            tree[id[idom[i]]].pb(id[i]);
52        }
53    }
54 };

```

6.14 Topological Sort

```

1 vector<vector<int>> graph;
2 vector<int> visit(10, 0);
3 vector<int> order;
4 int n;
5 bool cycle; // 記錄DFS的過程中是否偵測到環
6 void DFS(int i) { //reverse(order) is topo
7     if (visit[i] == 1) {cycle = true; return;}
8     if (visit[i] == 2) return;

```

```

9     visit[i] = 1;
10    for(auto to :graph[i])
11        DFS(to);
12    visit[i] = 2;
13    order.push_back(i);
14 } //for() if(!vis[i])DFS(i)
15 int main() {
16     for (int i=0; i<n; ++i){
17         if (!visit[i])
18             DFS(i);
19     }
20     if (cycle)
21         cout << "圖上有環";
22     else
23         for (int i=n-1; i>=0; --i)
24             cout << order[i];
25 }

```

6.15 Closest Pair

```

1 template<typename _IT=point<T>*>
2 T cloest_pair(_IT L, _IT R) {
3     if (R-L <= 1) return INF;
4     _IT mid = L+(R-L)/2;
5     T x = mid->x;
6     T d = min(cloest_pair(L, mid), cloest_pair(mid, R));
7     inplace_merge(L, mid, R, ycmp);
8     static vector<point> b; b.clear();
9     for(auto u=L; u<R; ++u) {
10         if ((u->x-x)*(u->x-x)>=d) continue;
11         for(auto v=b.rbegin(); v!=b.rend(); ++v) {
12             T dx=u->x-v->x, dy=u->y-v->y;
13             if (dy*dy>=d) break;
14             d=min(d, dx*dx+dy*dy);
15         }
16         b.push_back(*u);
17     }
18     return d;
19 }
20 T closest_pair(vector<point<T>> &v) {
21     sort(v.begin(), v.end(), xcmp);
22     return closest_pair(v.begin(), v.end());
23 }

```

6.16 Minimum Mean Cycle

```

1 ll road[N][N]; // input here
2 struct MinimumMeanCycle {
3     ll dp[N + 5][N], n;
4     pll solve() {
5         ll a = -1, b = -1, L = n + 1;
6         for (int i = 2; i <= L; ++i)
7             for (int k = 0; k < n; ++k)
8                 for (int j = 0; j < n; ++j)
9                     dp[i][j] =
10                         min(dp[i - 1][k] + road[k][j], dp[i][j]);
11         for (int i = 0; i < n; ++i) {
12             if (dp[L][i] >= INF) continue;
13             ll ta = 0, tb = 1;

```

```

14     for (int j = 1; j < n; ++j)
15         if (dp[j][i] < INF &&
16             ta * (L - j) < (dp[L][i] - dp[j][i]) * tb)
17             ta = dp[L][i] - dp[j][i], tb = L - j;
18         if (ta == 0) continue;
19         if (a == -1 || a * tb > ta * b) a = ta, b = tb;
20     }
21     if (a != -1) {
22         ll g = __gcd(a, b);
23         return pll(a / g, b / g);
24     }
25     return pll(-1LL, -1LL);
26 }
27 void init(int _n) {
28     n = _n;
29     for (int i = 0; i < n; ++i)
30         for (int j = 0; j < n; ++j) dp[i + 2][j] = INF;
31 }
32 };

```

6.17 Planar

```

1 class Graph {
2 public:
3     int V;
4     vector<vector<int>>> adj;
5     Graph(int vertices : V(vertices), adj(vertices) {})
6     void addEdge(int u, int v) {
7         adj[u].push_back(v);
8         adj[v].push_back(u);
9     }
10 };
11
12 bool containsSubgraph(const Graph& graph, const vector<int>&
13     subgraph) {
14     unordered_set<int> subgraphVertices(subgraph.begin(),
15         subgraph.end());
16     for (int vertex : subgraphVertices) {
17         for (int neighbor : graph.adj[vertex]) {
18             if (subgraphVertices.count(neighbor) == 0) {
19                 bool found = true;
20                 for (int v : subgraph) {
21                     if (v != vertex && v != neighbor) {
22                         if (graph.adj[v].size() < 3) {
23                             found = false;
24                             break;
25                         }
26                     }
27                 }
28                 if (found)
29                     return true;
30             }
31         }
32     }
33     return false;
34 }
35
36 bool isPlanar(const Graph& graph) {
37     // Subgraphs isomorphic to K and K ,
38     vector<int> k5 = {0, 1, 2, 3, 4}; // Vertices of K
39     vector<int> k33a = {0, 1, 2}; // Vertices of K
40     , (part A)

```

```

38     vector<int> k33b = {3, 4, 5}; // Vertices of K 39 }
39     , (part B)
40     if (containsSubgraph(graph, k5) || containsSubgraph(graph
41         , k33a) || containsSubgraph(graph, k33b)) {
42         return false; // The graph is non-planar
43     }
44     return true; // The graph is planar
45 }
46 int main() {
47     int vertices, edges;
48     Graph graph(vertices);
49     for (int i = 0; i < edges; ++i) {
50         int u, v; cin >> u >> v;
51         graph.addEdge(u, v);
52     }
53     if (isPlanar(graph)) {
54         cout << "The graph is planar." << endl;
55     } else {
56         cout << "The graph is non-planar." << endl;
57     }

```

6.18 Heavy Light Decomposition

```

1 int dep[N], pa[N], sz[N], nxt[N];
2 int id[N], rt[N];
3 int dfs(int u, int lst, int d = 0) {
4     dep[u] = d;
5     pa[u] = lst;
6     sz[u] = 1;
7     nxt[u] = -1;
8     for (int v : g[u]) {
9         if (v == lst) continue;
10        sz[u] += dfs(v, u, d + 1);
11        if (nxt[u] == -1 || sz[v] > sz[nxt[u]]) {
12            nxt[u] = v;
13        }
14    }
15    return sz[u];
16 }
17 int tn = 0;
18 void mapId(int u, int lst, int root) {
19     id[u] = ++tn;
20     rt[u] = root;
21     if (~nxt[u]) mapId(nxt[u], u, root);
22     for (int v : g[u]) {
23         if (v == lst || v == nxt[u]) continue;
24         mapId(v, u, v);
25     }
26 }
27 void solve() {
28     while (rt[a] != rt[b]) {
29         if (dep[rt[a]] > dep[rt[b]]) swap(a, b);
30         //...
31         b = pa[rt[b]];
32     }
33     if (a != b) {
34         if (id[a] > id[b]) swap(a, b);
35         //...
36     } else {
37         //...
38     }

```

6.19 Centroid Decomposition

```

1 int sz[maxn]{};
2 bool ok[maxn]{};
3 int get_subtree_size(int u, int lst) {
4     sz[u] = 1;
5     for (int v : g[u]) {
6         if (v == lst || ok[v]) continue;
7         sz[u] += get_subtree_size(v, u);
8     }
9     return sz[u];
10 }
11 int get_centroid(int u, int lst, int tree_size) {
12     for (int v : g[u]) {
13         if (v == lst || ok[v]) continue;
14         if (2 * sz[v] >= tree_size) {
15             return get_centroid(v, u, tree_size);
16         }
17     }
18     return u;
19 }
20 void centroid_decomp(int u = 1) { //1-based
21     int centroid = get_centroid(u, u, get_subtree_size(u, u));
22     //...
23     ok[centroid] = 1;
24     for (int v : g[centroid]) if (!ok[v]) {
25         centroid_decomp(v);
26     }
27 }

```

6.20 KM_O

```

1 // 二分圖最大權完美匹配
2 #define MAXN 100
3 #define INF INT_MAX
4 int g[MAXN][MAXN], lx[MAXN], ly[MAXN], slack_y[MAXN];
5 int px[MAXN], py[MAXN], match_y[MAXN], par[MAXN];
6 int n;
7 void adjust(int y) { //把增廣 上所有邊反轉
8     match_y[y] = py[y];
9     if (px[match_y[y]] != -2)
10         adjust(px[match_y[y]]);
11 }
12 bool dfs(int x) { //DFS找增廣
13     for (int y = 0; y < n; ++y) {
14         if (py[y] != -1) continue;
15         int t = lx[x] + ly[y] - g[x][y];
16         if (t == 0) {
17             py[y] = x;
18             if (match_y[y] == -1) {
19                 adjust(y);
20                 return 1;
21             }
22             if (px[match_y[y]] != -1) continue;
23             px[match_y[y]] = y;
24             if (dfs(match_y[y])) return 1;
25         } else if (slack_y[y] > t) {

```

```

26     slack_y[y]=t;
27     par[y]=x;
28 }
29 }
30 return 0;
31 }
32 inline int km() {
33     memset(ly, 0, sizeof(int)*n);
34     memset(match_y, -1, sizeof(int)*n);
35     for(int x=0; x<n; ++x) {
36         lx[x] = -INF;
37         for(int y=0; y<n; ++y) {
38             lx[x] = max(lx[x], g[x][y]);
39         }
40     }
41     for(int x=0; x<n; ++x) {
42         for(int y=0; y<n; ++y) slack_y[y] = INF;
43         memset(px, -1, sizeof(int)*n);
44         memset(py, -1, sizeof(int)*n);
45         px[x] = -2;
46         if(dfs(x)) continue;
47         bool flag = 1;
48         while(flag) {
49             int cut = INF;
50             for(int y=0; y<n; ++y)
51                 if(py[y] == -1 && cut > slack_y[y]) cut = slack_y[y];
52             for(int j=0; j<n; ++j) {
53                 if(px[j] != -1) lx[j] -= cut;
54                 if(py[j] != -1) ly[j] += cut;
55                 else slack_y[j] -= cut;
56             }
57             for(int y=0; y<n; ++y) {
58                 if(py[y] == -1 && slack_y[y] == 0) {
59                     py[y] = par[y];
60                     if(match_y[y] == -1) {
61                         adjust(y);
62                         flag = 0;
63                         break;
64                     }
65                     px[match_y[y]] = y;
66                     if(dfs(match_y[y])) {
67                         flag = 0;
68                         break;
69                     }
70                 }
71             }
72         }
73     }
74     int ans = 0;
75     for(int y=0; y<n; ++y) if(g[match_y[y]][y] != -INF) ans += g[match_y[y]][y];
76     return ans;
77 }

```

6.21 Minimum Arborescence

```

1 struct zhu_liu { // O(VE)
2     struct edge {
3         int u, v;
4         ll w;
5     };
6     vector<edge> E; // 0-base
7     int pe[N], id[N], vis[N];

```

```

8     ll in[N];
9     void init() { E.clear(); }
10    void add_edge(int u, int v, ll w) {
11        if (u != v) E.pb(edge{u, v, w});
12    }
13    ll build(int root, int n) {
14        ll ans = 0;
15        for(;;) {
16            fill_n(in, n, INF);
17            for(int i = 0; i < SZ(E); ++i)
18                if (E[i].u != E[i].v && E[i].w < in[E[i].v])
19                    pe[E[i].v] = i, in[E[i].v] = E[i].w;
20            for(int u = 0; u < n; ++u) // no solution
21                if (u != root && in[u] == INF) return -INF;
22            int cntnode = 0;
23            fill_n(id, n, -1), fill_n(vis, n, -1);
24            for(int u = 0; u < n; ++u) {
25                if (u != root) ans += in[u];
26                int v = u;
27                while (vis[v] != u && !~id[v] && v != root)
28                    vis[v] = u, v = E[pe[v]].u;
29                if (v != root && !~id[v]) {
30                    for(int x = E[pe[v]].u; x != v;
31                        x = E[pe[x]].u)
32                        id[x] = cntnode;
33                    id[v] = cntnode++;
34                }
35                if (!cntnode) break; // no cycle
36            }
37            for(int u = 0; u < n; ++u)
38                if (!~id[u]) id[u] = cntnode++;
39            for(int i = 0; i < SZ(E); ++i) {
40                int v = E[i].v;
41                E[i].u = id[E[i].u], E[i].v = id[E[i].v];
42                if (E[i].u != E[i].v) E[i].w -= in[v];
43            }
44            n = cntnode, root = id[root];
45        }
46        return ans;
47    }
48 };

```

6.22 Maximum Clique

```

1 struct Maximum_Clique {
2     typedef bitset<MAXN> bst;
3     bst N[MAXN], empty;
4     int p[MAXN], n, ans;
5     void BronKerbosch2(bst R, bst P, bst X) {
6         if (P == empty && X == empty)
7             return ans = max(ans, (int)R.count()), void();
8         bst tmp = P | X;
9         int u;
10        if ((R | P | X).count() <= ans) return;
11        for(int uu = 0; uu < n; ++uu) {
12            u = p[uu];
13            if (tmp[u] == 1) break;
14        }
15        // if (double(clock())/CLOCKS_PER_SEC > .999)
16        // return;
17        bst now2 = P & ~N[u];
18        for(int vv = 0; vv < n; ++vv) {
19            int v = p[vv];

```

```

20        if (now2[v] == 1) {
21            R[v] = 1;
22            BronKerbosch2(R, P & N[v], X & N[v]);
23            R[v] = 0, P[v] = 0, X[v] = 1;
24        }
25    }
26 }
27 void init(int _n) {
28     n = _n;
29     for(int i = 0; i < n; ++i) N[i].reset();
30 }
31 void add_edge(int u, int v) {
32     N[u][v] = N[v][u] = 1;
33 }
34 int solve() { // remember srand
35     bst R, P, X;
36     ans = 0, P.flip();
37     for(int i = 0; i < n; ++i) p[i] = i;
38     random_shuffle(p, p + n), BronKerbosch2(R, P, X);
39     return ans;
40 }
41 };

```

7 Math

7.1 Miller Robin

```

1 // n < 4,759,123,141      3 : 2, 7, 61
2 // n < 1,122,004,669,633  4 : 2, 13, 23, 1662803
3 // n < 3,474,749,660,383  6 : pirmses <= 13
4 // n < 2^64              7 : 2, 325, 9375, 28178, 450775,
5 //                        9780504, 1795265022
6 //From jacky860226
7 typedef long long LL;
8 inline LL mul(LL a, LL b, LL m) { //a*b%m
9     return (a%m)*(b%m)%m;
10 }
11 /*LL mul(LL a, LL b, LL m) { //a*b%m
12     a %= m, b %= m;
13     LL y = (LL)((double)a*b/m+0.5); //fast for m < 2^58
14     LL r = (a*b-y*m)%m;
15     return r<0 ? r+m : r;
16 }*/
17 template<typename T> T
18 pow(T a, T b, T mod) { //a^b%mod
19     T ans = 1;
20     while(b) {
21         if(b&1) ans = mul(ans, a, mod);
22         a = mul(a, a, mod);
23         b >>= 1;
24     } return ans;
25 }
26 template<typename T>
27 bool isprime(T n, int num) { //num = 3,7
28     int sprp[3] = {2, 7, 61}; //int範圍可解
29     //int llsprp[7] =
30         {2, 325, 9375, 28178, 450775, 9780504, 1795265022}; //至少
31         unsigned long long範圍
32     if(n==2) return true;

```



```

31 if(n<2 || n%2==0) return false;
32 //n-1 = u * 2^t
33 int t = 0; T u = n-1;
34 while(u%2==0) u >>= 1, t++;
35 for(int i=0; i<num; i++) {
36     T a = sprp[i]%n;
37     if(a==0 || a==1 || a==n-1) continue;
38     T x = pow(a,u,n);
39     if(x==1 || x==n-1) continue;
40     for(int j=1; j<t; j++) {
41         x = mul(x,x,n);
42         if(x==1) return false;
43         if(x==n-1) break;
44     }
45     if(x!=n-1) return false;
46 } return true;
47 }

```

7.2 fpow

```

1 ll fpow(ll b, ll p, ll mod) {
2     ll res = 1;
3     while (p) {
4         if (p & 1) res = res * b % mod;
5         b = b * b % mod, p >>= 1;
6     }
7     return res;
8 }

```

7.3 modinv

```

1 // 解 (ax == 1) mod p * p 必須是質數, a 是正整數。
2 ll modinv(ll a, ll p) {
3     if (p == 1) return 0;
4     ll pp = p, y = 0, x = 1;
5     while (a > 1) {
6         ll q = a / p, t = p;
7         p = a % p, a = t, t = y, y = x - q * y, x = t;
8     }
9     if (x < 0) x += pp;
10    return x;
11 }
12 // 解 (ax == b) mod p * p 必須是質數, a 和 b 是正整數。
13 ll modinv(ll a, ll b, ll p) {
14     ll ret = modinv(a, p);
15     return ret * b % p;
16 }

```

7.4 PollardRho

```

1 // does not work when n is prime
2 ll f(ll x, ll mod) { return add(mul(x,x,mod),1,mod); }
3 ll pollard_rho(ll n) {
4     if(!(n&1)) return 2;
5     while(1) {
6         ll y=2, x=rand()%(n-1)+1, res=1;

```

```

7         for(int sz=2; res==1; y=x, sz*=2)
8             for(int i=0; i<sz&&res<=1; ++i)
9                 x=f(x,n), res=__gcd(abs(x-y),n);
10        if(res!=0&&res!=n) return res;
11    }
12 }

```

7.5 Euler Totient Function

```

1 ll phi[maxn];
2 for (int i = 0; i < maxn; ++i) {
3     phi[i] = i;
4 }
5 for (int i = 2; i < maxn; ++i) if (phi[i] == i) {
6     phi[i] = i - 1; //prime
7     for (int j = 2; i*j < maxn; ++j) { //overflow
8         phi[i*j] = (phi[i*j] / i) * (i - 1);
9     }
10 }

```

7.6 extGCD

```

1 int extgcd(int a, int b, int &x, int &y) { //a*x + b*y = 1
2     if (b == 0) {
3         x = 1;
4         y = 0;
5         return a; //到達遞歸邊界開始向上一層返回
6     }
7     int r = extgcd(b, a % b, x, y);
8     int temp = y; //把x y變成上一層的
9     y = x - (a / b) * y;
10    x = temp;
11    return r; //得到a b的最大公因數
12 }
13 int main() {
14     int a = 55, b = 80;
15     int x, y; //a*x + b*y = 1;
16     int GCD = extgcd(a, b, x, y);
17 }

```

7.7 random

```

1 inline int ran() {
2     static int x = 20167122;
3     return x = (x * 0xdefaced + 1) & INT_MAX;
4 }

```

7.8 FFT

```

1 //OI Wiki
2 #include <complex>
3 using cd = complex<double>;
4 const double PI = acos(-1);

```

```

5 void change(vector<cd> &y) {
6     vector<int> rev(y.size());
7     for (int i = 0; i < y.size(); ++i) {
8         rev[i] = rev[i >> 1] >> 1;
9         if (i & 1) {
10             rev[i] |= y.size() >> 1;
11         }
12     }
13     for (int i = 0; i < y.size(); ++i) {
14         if (i < rev[i]) {
15             swap(y[i], y[rev[i]]);
16         }
17     }
18 }
19 void fft(vector<cd> &y, bool inv) {
20     change(y);
21     for (int h = 2; h <= y.size(); h <= 1) {
22         cd wn(cos(2 * PI / h), sin(2 * PI / h));
23         for (int j = 0; j < y.size(); j += h) {
24             cd w(1, 0);
25             for (int k = j; k < j + h / 2; ++k) {
26                 cd u = y[k];
27                 cd t = w * y[k + h / 2];
28                 y[k] = u + t;
29                 y[k + h / 2] = u - t;
30                 w = w * wn;
31             }
32         }
33     }
34     if (inv) {
35         reverse(begin(y), end(y));
36         for (int i = 0; i < y.size(); ++i) {
37             y[i] /= y.size();
38         }
39     }
40 }
41 void solve() {
42     int n;
43     int m = 1 << (lg(n) + 1); //power of 2
44     vector<cd> a(m), b(m);
45     //...
46     fft(a, 0);
47     fft(b, 0);
48     vector<cd> c(m);
49     for (int i = 0; i < m; ++i) {
50         c[i] = a[i] * b[i];
51     }
52     fft(c, 1);
53     for (auto &p: c) {
54         int ans = int(p.real() + 0.25);
55     }
56 }

```

7.9 mu

```

1 int mu[MAXN];
2 bool isnp[MAXN];
3 vector<int> primes;
4 void init(int n)
5 {
6     mu[1] = 1;
7     for (int i = 2; i <= n; ++i)
8     {

```

```

9     if (!isnp[i])
10         primes.push_back(i), mu[i] = -1; // 质数为-1
11     for (int p : primes)
12     {
13         if (p * i > n)
14             break;
15         isnp[p * i] = 1;
16         if (i % p == 0)
17         {
18             mu[p * i] = 0; // 有平方因数为0
19             break;
20         }
21         else
22             mu[p * i] = mu[p] * mu[i]; // 互质, 用积性
23     } // 函数性质
24 }
25 }

```

8 Misc

8.1 Mo's Algorithm

```

1 struct Query {
2     int L, R;
3     //...
4 };
5 vector<Query> query;
6 void solve() { //K = n / sqrt(q)
7     sort(iter(query), [&](Query &a, Query &b) {
8         if (a.L / K != b.L / K) return a.L < b.L;
9         return a.L / K % 2 ? a.R < b.R : a.R > b.R;
10    });
11    int L = 0, R = 0;
12    for (auto x: query) {
13        while (R < x.R) add(arr[++R]);
14        while (L > x.L) add(arr[--L]);
15        while (R > x.R) sub(arr[R--]);
16        while (L < x.L) sub(arr[L++]);
17        //...
18    }
19 }

```

8.2 pbds

```

1 #include <ext/pb_ds/assoc_container.hpp>
2 #include <ext/pb_ds/tree_policy.hpp>
3 using namespace __gnu_pbds;
4
5 template<typename T>
6 using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
7     tree_order_statistics_node_update>;
8
9 int32_t main() {
10     ordered_set<int64_t> rbt;
11     // .insert(x); .erase(x)
12     // .lower_bound(x); .upper_bound(x): iter

```

```

12 // .find_by_order(k): find k-th small value(iter)
13 // .order_of_key(x): return x is k-th big
14 // .join(rbt2): merge with no mutiple same element
15 // .split(key, rbt2): rbt keeps value <= key, others to
16 // rbt2
17 }

```

8.3 Misc

```

1 mt19937 rng(chrono::steady_clock::now().time_since_epoch().
2     count());
3 int randint(int lb, int ub) {
4     return uniform_int_distribution<int>(lb, ub)(rng);
5 } //static unsigned x = 19; ++(x *= 0xdefaced);
6
7 #define SECs ((double)clock() / CLOCKS_PER_SEC)
8
9 struct KeyHasher {
10     size_t operator()(const Key& k) const {
11         return k.first + k.second * 100000;
12     } };
13 typedef unordered_map<Key, int, KeyHasher> map_t;
14
15 __lg
16 __gcd
17
18 __builtin_popcount // 二進位有幾個1
19 __builtin_clz // 左起第一個1之前0的個數
20 __builtin_parity // 1的個數的奇偶性

```

8.4 bit

```

1 int __builtin_ffs(unsigned int x)
2 int __builtin_ffsl(unsigned long)
3 int __builtin_ffsll(unsigned long long)
4 返回右起第一個1的位置
5 Returns one plus the index of the least significant 1-bit of
6 x, or if x is zero, returns zero.
7
8 int __builtin_clz(unsigned int x)
9 int __builtin_clzl(unsigned long)
10 int __builtin_clzll(unsigned long long)
11 返回左起第一個1之前0的個數
12 Returns the number of leading 0-bits in x, starting at the
13 most significant bit position. If x is 0, the result is
14 undefined.
15
16 int __builtin_ctz(unsigned int x)
17 int __builtin_ctzl(unsigned long)
18 int __builtin_ctzll(unsigned long long)
19 返回右起第一個1之後的0的個數
20 Returns the number of trailing 0-bits in x, starting at the
21 least significant bit position. If x is 0, the result is
22 undefined.
23
24 int __builtin_popcount(unsigned int x)
25 int __builtin_popcountl(unsigned long)
26 int __builtin_popcountll(unsigned long long)
27 返回1的個數

```

```

23 Returns the number of 1-bits in x.
24
25 int __builtin_parity(unsigned int x)
26 int __builtin_parityl(unsigned long)
27 int __builtin_parityll(unsigned long long)
28 返回1的個數的奇偶性(1的個數 mod 2的值)
29 Returns the parity of x, i.e. the number of 1-bits in x
30 modulo 2.

```

9 String

9.1 Hashing

```

1 const ll P = 401, M = 998244353;
2
3 ll hashes[10005], modp[10005];
4 ll hashp(string s, bool saveval) {
5     ll val = 0;
6     int index = 0;
7     for (char c: s) {
8         val = (val * P) % M + c % M;
9         if (saveval) hashes[index++] = val;
10    }
11    return val;
12 }
13 void init(int base, int exp) {
14     ll b = 1;
15     modp[0] = 1;
16     for (int i = 0; i < exp; i++) {
17         b = (b * base) % M;
18         modp[i + 1] = b;
19     }
20 }
21 ll subseq(int l, int r) { //[l, r]
22     if (l == 0) return hashes[r];
23     return (hashes[r] - hashes[l-1] * modp[r-l+1]) % M + M %
24     M;
25 }

```

9.2 Trie

```

1 struct node {
2     int ch[26]{};
3     int cnt = 0;
4 };
5 struct Trie {
6     vector<node> t;
7     void init() {
8         t.clear();
9         t.emplace_back(node());
10    }
11    void insert(string s) {
12        int ptr = 0;
13        for (char i: s) {
14            if (!t[ptr].ch[i - 'a']) {
15                t[ptr].ch[i - 'a'] = (int)t.size();
16                t.emplace_back(node());

```

```

17     }
18     ptr = t[ptr].ch[i - 'a'];
19 }
20 t[ptr].cnt++;
21 }
22 } trie;

```

9.3 Zvalue

```

1 vector<int> Zvalue(string &s) { //t + # + s
2     vector<int> Z(s.size());
3     int x = 0, y = 0;
4     for (int i=0; i<s.size(); ++i) {
5         Z[i] = max(0, min(y - i + 1, Z[i - x]));
6         while (i + Z[i] < s.size() && s[Z[i]] == s[i + Z[i]])
7             x = i, y = i + Z[i], ++Z[i];
8     }
9     return Z;
10 }

```

9.4 KMP

```

1 int F[maxn]{};
2 vector<int> match(string& s, string& t) {
3     int p = F[0] = -1;
4     for (int i = 1; i < t.size(); ++i) {
5         while (p != -1 && t[p + 1] != t[i]) p = F[p];
6         if (t[p + 1] == t[i]) ++p;
7         F[i] = p;
8     }
9     p = -1;
10    vector<int> v;
11    for (int i = 0; i < s.size(); ++i) {
12        while (p != -1 && t[p + 1] != s[i]) p = F[p];
13        if (t[p + 1] == s[i]) ++p;
14        if (p == t.size() - 1) v.push_back(i - p), p = F[p];
15    }
16    return v; //0-based
17 }

```

9.5 Manacher

```

1 int z[maxn * 2]{};
2 int manacher(string& s) {
3     string t = "#";
4     for (char c: s) t += c, t += '#';
5     int l = 0, r = 0, ans = 0; //l: mid, r: right
6     for (int i = 1; i < t.size(); ++i) {
7         z[i] = (r > i ? min(z[2 * l - i], r - i) : 1);
8         while (i - z[i] >= 0 && i + z[i] < t.size()) {
9             if (t[i - z[i]] == t[i + z[i]])
10                ++z[i];
11             else
12                break;
13         }
14         if (i + z[i] > r) r = i + z[i], l = i;

```

```

15     }
16     for (int i = 1; i < t.size(); ++i) ans = max(ans, z[i] - 1)
17     ;
18     string res;
19     for (int i = 1; i < t.size(); ++i) if (ans == z[i] - 1) {
20         for (int j = i - ans + 1; j < i + ans; ++j) if (t[j] != '
21             #') {
22                 res += t[j];
23             }
24         break;
25     }
26     return ans;
27 }

```

10 Tree

10.1 LCA

```

1 int n, logn, t=0;
2 vector<vector<int>> graph;
3 vector<vector<int>> ancestor;
4 vector<int> tin, tout;
5 void dfs(int x) {
6     tin[x] = t++;
7     for (auto y: graph[x]) {
8         if (y != ancestor[x][0]) {
9             ancestor[y][0] = x;
10            dfs(y);
11        }
12    }
13    tout[x] = t++;
14 }
15 bool is_ancestor(int x, int y) {
16     return tin[x] <= tin[y] && tout[x] >= tout[y];
17 }
18 void table() {
19     for (int i=1; i<logn; i++) // 上 輩祖先、上四輩祖先、上八輩
20         祖先、.....
21         for (int x=0; x<n; ++x)
22             ancestor[x][i] = ancestor[ancestor[x][i-1]][i-1];
23 }
24 int kth_ancestor(int x, int k) {
25     for (int i=0; i<logn; i++) // k拆解成二進位位數，找到第k祖
26         先。不斷上升逼近之。
27         if (k & (1<<i))
28             x = ancestor[x][i];
29     return x;
30 }
31 void rooted_tree(int root) { // build the tree with root at "
32     根"
33     ancestor[root][0] = root;
34     dfs(root);
35     table();
36 }
37 int LCA(int x, int y) {
38     if (is_ancestor(x, y)) return x;
39     if (is_ancestor(y, x)) return y;
40     for (int i=logn-1; i>=0; i--)

```

```

40     if (!is_ancestor(ancestor[x][i], y))
41         x = ancestor[x][i];
42     return ancestor[x][0];
43 }
44 int main() {
45     graph = {
46         {1,2},
47         {3},
48         {5,6},
49         {7},
50         {},
51         {},
52         {},
53         {8},
54         {4},
55     };
56     n = 9;
57     logn = ceil(log2(n));
58     ancestor.resize(n, vector<int>(logn));
59     tin.resize(n);
60     tout.resize(n);
61
62     rooted_tree(0);
63     while (true) {
64         int a, b;
65         cin >> a >> b;
66         cout << LCA(a, b) << endl;
67     }
68 }
69 int main() {
70     n = 9;
71     logn = ceil(log2(n));
72     ancestor.resize(n, vector<int>(logn));
73     tin.resize(n);
74     tout.resize(n);
75     rooted_tree(0);
76     while (true) {
77         int a, b;
78         cin >> a >> b;
79         cout << LCA(a, b) << endl;
80     }
81 }

```

10.2 Diameter

```

1 vector<vector<int>> graph;
2 int diameter = 0;
3 int dfs(int start, int parent) {
4     int h1 = 0, h2 = 0;
5     for (auto child: graph[start]) {
6         if (child != parent) {
7             int h = dfs(child, start) + 1;
8             if (h > h1) {
9                 h2 = h1;
10                h1 = h;
11            }
12            else if (h > h2) {
13                h2 = h;
14            }
15        }
16    }
17    diameter = max(diameter, h1 + h2);
18    return h1;

```

```

19 }
20 // call diameter
21 int main(){
22     dfs(0,-1);
23     cout << diameter<<endl;
24 }

```

10.3 Radius

```

1 // Perform DFS to find the farthest node and its distance
  from the given node
2 pair<int, int> dfs(int node, int distance, vector<bool> &
  visited, const vector<vector<int>> &adj_list){
3     visited[node] = true;
4     int max_distance = distance;
5     int farthest_node = node;
6
7     for (int neighbor : adj_list[node]){
8         if (!visited[neighbor]){
9             auto result = dfs(neighbor, distance + 1, visited
10                , adj_list);
11             if (result.first > max_distance){
12                 max_distance = result.first;
13                 farthest_node = result.second;
14             }
15         }
16     }
17     return make_pair(max_distance, farthest_node);
18 }
19
20 // Calculate the radius of the tree using DFS
21 int tree_radius(const vector<vector<int>> &adj_list){
22     int num_nodes = adj_list.size();
23     vector<bool> visited(num_nodes, false);
24
25     // Find the farthest node from the root (node 0)
26     auto farthest_result = dfs(0, 0, visited, adj_list);
27
28     // Reset visited array
29     fill(visited.begin(), visited.end(), false);
30
31     // Calculate the distance from the farthest node
32     int radius = dfs(farthest_result.second, 0, visited,
33        adj_list).first;
34
35     return radius;
36 }
37 int main() {
38     vector<vector<int>> adj_list;
39     int radius = tree_radius(adj_list);
40     cout << "Tree radius: " << radius << endl;
41     return 0;
42 }

```

10.4 Spanning Tree

```

1 const int V = 100, E = 1000;
2 struct Edge {int a, b, c;} e[E]; // edge list
3 bool operator<(Edge e1, Edge e2) {return e1.c < e2.c;}

```

```

4 int p[V];
5 void init() {for (int i=0; i<V; ++i) p[i] = i;}
6 int find(int x) {return x == p[x] ? x : (p[x] = find(p[x]));}
7 void merge(int x, int y) {p[find(x)] = find(y);}
8
9 void Kruskal() {
10     init();
11     sort(e, e+E);
12     int i, j;
13     for (i = 0, j = 0; i < V-1 && j < E; ++i){
14         while (find(e[j].a) == find(e[j].b)) j++;
15         merge(e[j].a, e[j].b);
16         cout << "起點: " << e[j].a << "終點: " << e[j].b << "權重: "
17             << e[j].c;
18         j++;
19     }
20     if (i == V-1) cout << "得到最小生成樹";
21     else cout << "得到最小生成森 ";
22 }

```

NYCU_Segmentree

Codebook

Contents

1 Dp	1				
1.1 01_knapsack	1				
1.2 Josephus	1				
1.3 Bitmask	1				
1.4 InfinitKnapsack	1				
2 Data Structure	1				
2.1 DSU	1				
2.2 Monotonic Queue	1				
2.3 BIT	1				
2.4 Treap	2				
2.5 Segment Tree	2				
2.6 Sparse Table	3				
2.7 Monotonic Stack	3				
3 Flow	3				
3.1 Maximum Simple Graph Matching	3				
3.2 Dinic	4				
4 Formula	4				
4.1 formula	4				
4.1.1 Pick 公式	4				
4.1.2 圖論	4				
4.1.3 dinic 特殊圖複雜度	4				
4.1.4 0-1 分數規劃	4				
		4.1.5 學長公式	4		
		4.1.6 基本數論	5		
		4.1.7 排組公式	5		
		4.1.8 冪次, 冪次和	5		
		4.1.9 Burnside's lemma	5		
		4.1.10 Count on a tree	5		
		5 Geometry	5		
		5.1 Sort by Angle	5		
		5.2 Geometry	5		
		5.3 Convex Hull	8		
		5.4 Min Covering Circle	8		
		5.5 Point in Polygon	8		
		6 Graph	8		
		6.1 Bipartite Matching	8		
		6.2 Tarjan SCC	9		
		6.3 Bridge	9		
		6.4 2 SAT	9		
		6.5 Kosaraju 2DFS	10		
		6.6 Minimum Steiner Tree	10		
		6.7 Dijkstra	10		
		6.8 Maximum Clique Dyn	11		
		6.9 Minimum Clique Cover	11		
		6.10 Floyd Warshall	11		
		6.11 Articulation Vertex	11		
		6.12 Number of Maximal Clique	12		
		6.13 Dominator Tree	12		
		6.14 Topological Sort	12		
		6.15 Closest Pair	12		
		6.16 Minimum Mean Cycle	12		
		6.17 Planar	13		
		6.18 Heavy Light Decomposition	13		
		6.19 Centroid Decomposition	13		
		6.20 KM_O	13		
		6.21 Minimum Arborescence	14		
		6.22 Maximum Clique	14		
		7 Math	14		
		7.1 Miller Robin	14		
		7.2 fpow	15		
		7.3 modinv	15		
		7.4 PollardRho	15		
		7.5 Euler Totient Function	15		
		7.6 extGCD	15		
		7.7 random	15		
		7.8 FFT	15		
		7.9 mu	15		
		8 Misc	16		
		8.1 Mo's Algorithm	16		
		8.2 pbds	16		
		8.3 Misc	16		
		8.4 bit	16		
		9 String	16		
		9.1 Hashing	16		
		9.2 Trie	16		
		9.3 Zvalue	17		
		9.4 KMP	17		
		9.5 Manacher	17		
		10 Tree	17		
		10.1 LCA	17		
		10.2 Diameter	17		
		10.3 Radius	18		
		10.4 Spanning Tree	18		