

# 1 Data Structure

## 1.1 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 };

```

## 1.2 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 };

```

## 1.3 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 };

```

## 1.4 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 };

```

## 1.5 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
41 public:
42    SegmentTree(vl&arr,int _n):arr(_arr),n(_n) {
43        assert(arr.size()==n);
44        summ.assign(4*n,0);
45        // minn.assign(4*n,1e9);
46        // maxx.assign(4*n,-1e9);
47        tag.assign(4*n,0);
48        build(0,arr.size());
49    }
50    void modify(int x,int val,int l,int r,int v=0) {
51
52    }
53    // query sum
54    loli query(int L,int R,int l,int r,int v=0) {
55        // dbn(L,R,l,r,v)
56        push(l,r,v);
57        if(l==L && R==r) {
58            return summ[v];
59            return minn[v];
60            return maxx[v];
61        }
62        int mid=(l+r)/2;
63        if(R<=mid)
64            return query(L,R,l,mid,2*v+1);
65        else if(mid<=L)
66            return query(L,R,mid,r,2*v+2);
67        else
68            return query(L,mid,l,mid,2*v+1)+query(mid,R,mid,r,2*v+2);
69    }
70    // plus `val` to every element in [L,R]
71    void update(int L,int R,loli val,int l,int r,int v=0) {
72        // dbn(L,R,l,r,v)
73        push(l,r,v);
74        if(l==L && R==r) {
75            tag[v]+=val;
76            push(l,r,v);
77            return;
78        }
79        int mid=(l+r)/2;
80        if(R<=mid)
81            update(L,R,val,l,mid,2*v+1);

```

```

82        else if(mid<=L)
83            update(L,R,val,mid,r,2*v+2);
84        else
85            update(L,mid,val,l,mid,2*v+1),update(mid,R,val,
86                mid,r,2*v+2);
87        pull(l,r,v);
88    }
89 };
90 void solve() {
91     int n,q;
92     cin>>n>>q;
93     vl arr(n);
94     for(auto&x:arr)
95         cin>>x;
96     SegmentTree st(arr,n);
97     while(q--){
98         int op=0;
99         // str op;
100        cin>>op;
101        if(op&1) {
102            loli l,r,val;
103            cin>>l>>r>>val;
104            assert(r>=l);
105            st.update(l-1,r,val,0,n);
106            // loli k,u;
107            // cin>>k>>u;
108            // st.update(k-1,k,u-arr[k-1],0,n);
109            // arr[k-1]=u;
110        }else{
111            int x,y;
112            cin>>x>>y;
113            assert(y>=x);
114            cout<<st.query(x-1,y,0,n)<<endl;
115        }
116    }
117 }

```

## 1.6 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 }

```

## 1.7 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 }

```

## 2 Flow

### 2.1 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    }

```

```

46    dis[from] = -1;
47    return 0;
48 }
49 int Maxflow(int s, int t){
50     this->s = s, this->t = t;
51     int flow = 0;
52     int df;
53     while(bfs()){
54         memset(cur, 0, sizeof(cur));
55         while(df = dfs(s, INF)){
56             flow += df;
57         }
58     }
59     return flow;
60 }
61 int main(){
62     int n = 4, m = 6;
63     MaxFlow maxflow;
64     for(int i = 0; i<m; i++){
65         int a, b, cap;
66         cin >> a >> b >> cap;
67         maxflow.add_edge(a, b, cap);
68     }
69     cout << maxflow.Maxflow(1, 3) << endl;

```

## 3 Formula

### 3.1 formula

#### 3.1.1 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

#### 3.1.2 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：

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

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

- $C(u, v) = W_{(u, v)}, (u, v) \in E$ ，雙向邊
- $C(S, v) = U, v \in V$
- $D_u = \sum_{(u, v) \in E} W_{(u, v)}$
- $C(v, T) = U + 2g - D_v - 2W_v, v \in V$
- 二分搜  $g$ ：  
 $l = 0, r = U, \text{eps} = 1/n^2$   
 if  $((U \times |V| - \text{flow}(S, T))/2 > 0)$   $l = \text{mid}$   
 else  $r = \text{mid}$
- $\text{ans} = \text{min\_cut}(S, T)$
- $|E| = 0$  要特殊判斷

#### 7. 弦圖：

- 點數大於 3 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

### 3.1.3 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： $O\left(V^{1/2}E\right)$

### 3.1.4 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 }

```

### 3.1.5 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  $M(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$

### 3.1.6 基本數論

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

### 3.1.7 排組公式

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

### 3.1.8 冪次, 冪次和

- $a^b \% P = a^{b \% \varphi(P) + \varphi(P)}, b \geq \varphi(P)$
- $1^3 + 2^3 + 3^3 + \dots + n^3 = \frac{n^4}{4} + \frac{n^3}{2} + \frac{n^2}{4}$
- $1^4 + 2^4 + 3^4 + \dots + n^4 = \frac{n^5}{5} + \frac{n^4}{2} + \frac{n^3}{3} - \frac{n}{30}$
- $1^5 + 2^5 + 3^5 + \dots + n^5 = \frac{n^6}{6} + \frac{n^5}{2} + \frac{5n^4}{12} - \frac{n^2}{12}$
- $0^k + 1^k + 2^k + \dots + n^k = P(k), P(k) = \frac{(n+1)^{k+1} - \sum_{i=0}^{k-1} C_i^{k+1} P(i)}{k+1}, P(0) = n+1$
- $\sum_{k=0}^{m-1} k^n = \frac{1}{n+1} \sum_{k=0}^n C_k^{n+1} B_k m^{n+1-k}$
- $\sum_{j=0}^m C_j^{m+1} B_j = 0, B_0 = 1$
- 除  $B_1 = -1/2$ , 剩下的奇數項都是 0
- $B_2 = 1/6, B_4 = -1/30, B_6 = 1/42, B_8 = -1/30, B_{10} = 5/66, B_{12} = -691/2730, B_{14} = 7/6, B_{16} = -3617/510, B_{18} = 43867/798, B_{20} = -174611/330,$

### 3.1.9 Burnside's lemma

- $|X/G| = \frac{1}{|G|} \sum_{g \in G} |X^g|$
- $X^g = t^{c(g)}$
- $G$  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。
- 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  $3 \times 3^4$ , 120(角) 有  $8 \times 3^2$ , 180(邊) 有  $6 \times 3^3$ , 全部  $\frac{1}{24} (3^6 + 6 \times 3^3 + 3 \times 3^4 + 8 \times 3^2 + 6 \times 3^3) = 57$

### 3.1.10 Count on a tree

- Rooted tree:  $s_{n+1} = \frac{1}{n} \sum_{i=1}^n (i \times a_i \times \sum_{j=1}^{\lfloor n/i \rfloor} a_{n+1-i \times j})$
- Unrooted tree:
  - Odd:  $a_n - \sum_{i=1}^{n/2} a_i a_{n-i}$
  - Even:  $Odd + \frac{1}{2} a_{n/2} (a_{n/2} + 1)$
- Spanning Tree
  - 完全圖  $n^{n-2}$
  - 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  $A$ ,  $ans = \det(A)$

## 4 Geometry

### 4.1 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 }

```

### 4.2 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() - 2], hull.back() - hull[hull.size() - 2]) >= 0)
25         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 }

```

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

```

### 4.4 MinCoveringCircle

```

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 Graph

### 5.1 Bipartite Matching

```

1 const int MAXN = 100;
2
3 struct Bipartite_matching{
4     int mx[MAXN], my[MAXN], vy[MAXN]; //matchX, matchY,
5     visitY
6     vector<int> edge[MAXN]; //adjcent list;
7     int x_cnt;
8     bool dfs(int x){
9         for(auto y: edge[x]){ //對 x 可以碰到的邊進 檢查
10            if(vy[y] == 1) continue; //避免遞迴 error
11
12            vy[y] = 1;
13            if(my[y] == -1 || dfs(my[y])){ //分析 3
14                mx[x] = y;
15                my[y] = x;
16                return true;
17            }
18        }
19        return false; //分析 4
20    }
21 }

```

```

20 int bipartite_matching() {
21     memset(mx, -1, sizeof(mx)); //分析 1,2
22     memset(my, -1, sizeof(my));
23     int ans = 0;
24     for(int i = 0; i < x_cnt; i++){ //對每一個 x 節點進
25         DFS(最大匹配)
26         memset(vy, 0, sizeof(vy));
27         if(dfs(i)) ans++;
28     }
29     return ans;
30 }
31 vector<vector<int>> get_match(){
32     vector<vector<int>> res;
33     for(int i = 0; i < x_cnt; i++){
34         if(mx[i] != -1){
35             res.push_back({i, mx[i]});
36         }
37     }
38     return res;
39 }
40 void add_edge(int i, int j){
41     edge[i].push_back(j);
42 }
43 void init(int x){
44     x_cnt = x;
45 }
46 };
47 int main(){
48     int n, m;
49     Bipartite_matching bm;
50     for(int i = 0; i < m; i++){
51         int a, b; cin >> a >> b;
52         bm.add_edge(a, b);
53     }
54     bm.init(n);
55     cout << bm.bipartite_matching() << endl;
56     auto match = bm.get_match();
57     for(auto t: match){
58         cout << t[0] << " " << t[1] << endl;
59     }
60 }
61 }

```

### 5.2 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 }

```

```

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 }

```

### 5.3 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 }

```

### 5.4 2 SAT

```

1 class TwoSAT{
2 public:
3     TwoSAT(int n) : n(n), graph(2 * n), visited(2 * n, false) {}
4     void addClause(int a, int b) { // 0-base;
5         a *= 2;
6         b *= 2;
7         // Add implications (~a => b) and (~b => a)
8         graph[a ^ 1].push_back(b);
9         graph[b ^ 1].push_back(a);
10    }
11    bool solve() { // Find SCCs and check for contradictions
12        for (int i = 0; i < 2 * n; ++i) {
13            if (!visited[i]) {
14                dfs1(i);
15            }
16        }
17        reverse(processOrder.begin(), processOrder.end());
18        // topological sort
19        for (int i = 0; i < 2 * n; ++i) {
20            visited[i] = false;
21        }
22        for (int node : processOrder) {
23            if (!visited[node]) {
24                scc.clear();
25                dfs2(node);
26                if (!checkSCCConsistency()) {
27                    return false;
28                }
29            }
30        }
31        return true;
32    }
33 private:
34    int n;
35    vector<vector<int>> graph;
36    vector<bool> visited;
37    vector<int> processOrder;
38    vector<int> scc;
39
40    void dfs1(int node) {
41        visited[node] = true;
42        for (int neighbor : graph[node]) {
43            if (!visited[neighbor]) {
44                dfs1(neighbor);
45            }
46        }
47        processOrder.push_back(node);
48    }
49
50    void dfs2(int node) {
51        visited[node] = true;
52        scc.push_back(node);
53        for (int neighbor : graph[node]) {
54            if (!visited[neighbor]) {
55                dfs2(neighbor);
56            }
57        }
58    }
59 }

```

```

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 }

```

### 5.5 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     for (int i = 0; i < n; i++) {
37         if (!visit[i])
38             dfs2(i, i);
39     }
40     for (int i = 0; i < n; i++) {
41         cout << i << " ";
42         if (i % 10 == 9) cout << endl;
43     }
44 }

```



```

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     } cout << endl;
48 }
49 }

```

## 5.6 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 int main() {
30     int startpoint;
31     dijkstra(startpoint);
32     //dis and parent
33 }

```

## 5.7 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 void floyd_warshall(int n) { // n is n nodes
6     for(int i = 0; i < n; i++) {
7         for(auto path: graph[i]) {
8             dis[i][path.second] = path.first;
9         }
10     }
11     for (int i=0; i<n; i++)
12         dis[i][i] = 0;
13     for (int k=0; k<n; k++) {
14         for (int i=0; i<n; i++) {
15             for (int j=0; j<n; j++) {
16                 if (dis[i][k] + dis[k][j] < dis[i][j] || dis[i][j]
17                     == -1) {
18                     dis[i][j] = dis[i][k] + dis[k][j];
19                     mid[i][j] = k; // 由i點走到j點經過 k點
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 int main() {
32     int n;
33     floyd_warshall(n);
34     for(int i = 0; i < 4; i++) {
35         for(int j = 0; j < 4; j++) {
36             cout << dis[i][j] << " ";
37         }
38     }
39     find_path(0, 2);
40 }

```

## 5.8 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         if (!visited[neighbor]) {
18             children++;
19             parent_array[neighbor] = node;
20             dfs_articulation(neighbor, node);
21         }
22     }

```

```

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     } //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 }

```

## 5.9 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 }

```

## 5.10 Closest Pair

```

1 template<typename _IT=point<T>*>
2 T closest_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(closest_pair(L,mid),closest_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 }

```

## 5.11 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 }

```

```

33 bool isPlanar(const Graph& graph) {
34     // Subgraphs isomorphic to K and K ,
35     vector<int> k5 = {0, 1, 2, 3, 4}; // Vertices of K
36     vector<int> k33a = {0, 1, 2}; // Vertices of K
37         , (part A)
38     vector<int> k33b = {3, 4, 5}; // Vertices of K
39         , (part B)
40
41     if(containsSubgraph(graph, k5) || containsSubgraph(graph
42         , k33a) || containsSubgraph(graph, k33b)) {
43         return false; // The graph is non-planar
44     }
45     return true; // The graph is planar
46 }
47
48 int main() {
49     int vertices, edges;
50     Graph graph(vertices);
51     for(int i = 0; i < edges; ++i) {
52         int u, v; cin >> u >> v;
53         graph.addEdge(u, v);
54     }
55     if(isPlanar(graph)) {
56         cout << "The graph is planar." << endl;
57     } else {
58         cout << "The graph is non-planar." << endl;
59     }
60 }

```

## 5.12 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 }

```

```

33 if(a != b) {
34     if(id[a] > id[b]) swap(a, b);
35     //...
36 } else {
37     //...
38 }
39 }

```

## 5.13 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 Math

### 6.1 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 }

```

### 6.2 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 }

```

### 6.3 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 }

```

### 6.4 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 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 }

```

### 6.5 random

```

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

```

### 6.6 EulerTotientFunction

```

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 }

```

### 6.7 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) + 1, 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 }

```

### 6.8 MillerRabin

```

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 : pimes <= 13
4 // n < 2^64              7 :
5 // 2, 325, 9375, 28178, 450775, 9780504, 1795265022
6 bool Miller_Rabin(ll a, ll n) {
7     if ((a = a % n) == 0) return 1;
8     if ((n & 1) ^ 1) return n == 2;
9     ll tmp = (n - 1) / ((n - 1) & (1 - n));
10    ll t = lg((n - 1) & (1 - n)), x = 1;
11    for (; tmp; tmp >= 1, a = mul(a, a, n))
12        if (tmp & 1) x = mul(x, a, n);
13    if (x == 1 || x == n - 1) return 1;
14    while(--t)
15        if ((x = mul(x, x, n)) == n - 1) return 1;
16    return 0;
17 }

```

### 6.9 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        }
22    }
23 }

```

```

21     else
22         mu[p * i] = mu[p] * mu[i]; // 互质, 用积性
           函数性质
23     }
24 }
25 }

```

## 7 Misc

### 7.1 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,
   tree_order_statistics_node_update>;
7
8 int32_t main() {
9     ordered_set<int64_t> rbt;
10    // .insert(x); .erase(x)
11    // .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
   rbt2
16 }

```

### 7.2 Misc

```

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

```

## 7.3 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 String

### 8.1 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) %
   M;
24 }

```

### 8.2 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;

```

### 8.3 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 }

```

### 8.4 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 }

```

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

```

## 9 Tree

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

```

### 9.2 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 }

```

### 9.3 Radius

```

1 // Perform DFS to find the farthest node and its distance
2 // from the given node
3 pair<int, int> dfs(int node, int distance, vector<bool> &
4     visited, const vector<vector<int>> &adj_list) {
5     visited[node] = true;
6     int max_distance = distance;
7     int farthest_node = node;
8     for (int neighbor : adj_list[node]) {
9         if (!visited[neighbor]) {
10            auto result = dfs(neighbor, distance + 1, visited,
11                adj_list);
12            if (result.first > max_distance) {
13                max_distance = result.first;
14                farthest_node = result.second;
15            }
16        }
17    }
18    return make_pair(max_distance, farthest_node);
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     // Find the farthest node from the root (node 0)
25     auto farthest_result = dfs(0, 0, visited, adj_list);
26     // Reset visited array
27     fill(visited.begin(), visited.end(), false);
28     // Calculate the distance from the farthest node
29     int radius = dfs(farthest_result.second, 0, visited,
30         adj_list).first;
31 }
32 }
33 }

```

```

34     return radius;
35 }
36 int main() {
37     vector<vector<int>> adj_list;
38     int radius = tree_radius(adj_list);
39     cout << "Tree radius: " << radius << endl;
40     return 0;
41 }

```

## 10 Z\_Original\_Code/Data\_Structure

### 10.1 dsu-class

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 class DSU{
6     public:
7     DSU(int n ) {
8         this->n = n;
9         reset();
10    }
11    int n;
12    vector<int> boss;
13    vector<int> rank;
14    vector<int> size;
15
16    void reset() {
17        this->boss.resize(n);
18        this->rank.resize(n,0);
19        this->size.resize(n,0);
20        for(int i =0;i<n;i++){
21            boss[i] = i;
22        }
23    }
24    int find(int x) {
25        if(boss[x] != x) {
26            boss[x] = find(boss[x]);
27        }
28        return boss[x];
29    }
30    int get_size(int x) {
31        return size[find(x)];
32    }
33    void merge(int x, int y) {
34        int a = find(x);
35        int b = find(y);
36        // if(a!=b){
37        //     boss[a] = b;
38        //     size[b] += size[a];
39        // }
40        if(a!=b) {
41            if(rank[a]<rank[b]) {
42                boss[a] = b;
43                size[b] += size[a];
44            } else if (rank[a]<rank[b]) {
45                boss[b] = a;
46                size[a] += size[b];
47            } else {
48                boss[a] = b;

```

```

49        size[b] += size[a];
50        rank[b]++;
51    }
52    }
53    }
54    bool aresame(int a,int b){
55        return find(a)==find(b);
56    }
57 };
58 int main(){
59     DSU dsu(10);
60
61     dsu.merge(0, 1);
62     dsu.merge(2, 3);
63     dsu.merge(4, 5);
64     dsu.merge(6, 7);
65
66     cout << "Are 0 and 1 connected? " << (dsu.aresame(0, 1) ?
67         "Yes" : "No") << endl;
68     cout << "Are 2 and 3 connected? " << (dsu.aresame(2, 3) ?
69         "Yes" : "No") << endl;
70     cout << "Are 4 and 5 connected? " << (dsu.aresame(4, 5) ?
71         "Yes" : "No") << endl;
72     cout << "Are 6 and 7 connected? " << (dsu.aresame(6, 7) ?
73         "Yes" : "No") << endl;
74     cout << "Are 1 and 2 connected? " << (dsu.aresame(1, 2) ?
75         "Yes" : "No") << endl;
76
77     dsu.merge(1, 2);
78
79     cout << "Are 0 and 2 connected? " << (dsu.aresame(0, 2) ?
80         "Yes" : "No") << endl;
81     cout << "Are 1 and 3 connected? " << (dsu.aresame(1, 3) ?
82         "Yes" : "No") << endl;
83
84     return 0;
85 }

```

### 10.2 monotonic-queue

```

1 //ref:leetcode
2 #include<bits/stdc++.h>
3
4 using namespace std;
5
6 class Monotonic_queue{
7     private:
8     deque<int> qu;
9     public:
10    void push(int n) {
11        while(!qu.empty() && qu.back() < n) {
12            qu.pop_back();
13        }
14        qu.push_back(n);
15    }
16    int max() {
17        return qu.front();
18    }
19    int min() {
20        return qu.back();
21    }
22    int size() {
23        return qu.size();

```

```

24    }
25    void pop() {
26        qu.pop_front();
27    }
28 };
29
30 vector<int> maxSlidingWindow(vector<int> nums, int k) {
31     Monotonic_queue window;
32     vector<int> res;
33     for (int i = 0; i < nums.size(); i++) {
34         if (i < k - 1) {
35             window.push(nums[i]);
36         } else {
37             window.push(nums[i]);
38             res.push_back(window.max());
39             if (window.max() == nums[i-k+1]) {
40                 window.pop();
41             }
42         }
43     }
44     return res;
45 }
46
47 int main(){
48     vector<int> nums = {1,3,-1,-3,5,3,6,7};
49     int k = 3;
50     vector<int> res = maxSlidingWindow(nums,k);
51     for (auto r:res) cout << r << " ";
52 }

```

### 10.3 BIT

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 class BIT{
5     public:
6     vector<int> bit;
7     int N;
8     BIT(int n) {
9         this->N = n;
10        this->bit.resize(n);
11    }
12    void update(int x,int d) {
13        while(x<=N) {
14            bit[x] +=d;
15            x +=x&(-x); // lowest bit in x;
16        }
17    }
18    int query(int x) {
19        int res = 0;
20        while(x) {
21            res+= bit[x];
22            x -= x& -x;
23        }
24        return res;
25    }
26 };
27 // Driver program to test above functions
28 int main()
29 {

```

```

30 vector<int> freq = {0, 2, 1, 1, 3, 2, 3, 4, 5, 6, 7, 8,
31     9};
32 int n = freq.size();
33 BIT bit(n);
34 for(int i = 1; i < n; i++) {
35     bit.update(i, freq[i]);
36 }
37 for(int i = 1; i < n; i++) {
38     cout << bit.query(i) << " ";
39 } cout << endl;
40 for(int i = 1; i < n; i++) {
41     bit.update(i, -1);
42 }
43 for(int i = 1; i < n; i++) {
44     cout << bit.query(i) << " ";
45 } cout << endl;
46 }

```

## 10.4 segment-tree-simple-add

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4 struct node{
5     int left;
6     int right;
7     int value;
8 };
9 vector<node> segment_tree;
10 void build(int left, int right, int x, vector<int> & nums){
11     segment_tree[x].left = left;
12     segment_tree[x].right = right;
13     // cout << left << " " << right << " " << x << endl;
14     if(left == right){ // here is leaf
15         segment_tree[x].value = nums[left];
16         return;
17     }
18     int mid = (left+right)/2;
19     build(left, mid, x<<1, nums);
20     build(mid+1, right, x<<1|1, nums);
21     segment_tree[x].value = segment_tree[x<<1].value+
22         segment_tree[x<<1|1].value;
23 }
24 void modify(int position, int x, int value){
25     if(segment_tree[x].left == position && segment_tree[x].
26         right == position){ // here is leaf
27         segment_tree[x].value = value;
28         return;
29     }
30     int mid = (segment_tree[x].left+segment_tree[x].right)/2;
31     if(position <= mid){
32         modify(position, x<<1, value);
33     } else{
34         modify(position, x<<1|1, value);
35     }
36     segment_tree[x].value = segment_tree[x<<1].value+
37         segment_tree[x<<1|1].value;
38 }
39 int query(int i, int j, int x){
40     // cout << i << " " << j << " " << segment_tree[x].left << " " <<
41         segment_tree[x].right << endl;
42     int res = 0;

```

```

40 int left = segment_tree[x].left;
41 int right = segment_tree[x].right;
42 int mid = (left+right)/2;
43 if(segment_tree[x].left==i && segment_tree[x].right ==j){
44     return segment_tree[x].value;
45 }
46 if(i>mid) return query(i, j, x*2+1);
47 if(mid>=j) return query(i, j, x*2);
48 return query(i, mid, x*2)+ query(mid+1, j, x*2+1);
49 }
50 int main(){
51     vector<int> nums =
52         {1,10,5,148,78,2,56,231,5,64,65,32,1,8};
53     int n = nums.size();
54     segment_tree.resize(n*4);
55     build(0, n-1, 1, nums);
56     modify(5, 1, 100);
57     // cout << "+++++++\n";
58     for(int i = 0; i < n; i++){
59         for(int j = i; j < n; j++){
60             cout << query(i, j, 1) << " ";
61         } cout << endl;
62     }
63 }

```

## 10.5 monotonic-stack

```

1 /*
2 input: array A
3 output: array B
4 bi is the value aj such that j>i and aj>bi (j)
5 ex:
6 A = [2,1,2,4,3]
7 B = [4,3,4,-1,-1]
8 */
9 #include<bits/stdc++.h>
10
11 using namespace std;
12 vector<int> monotonic_stack(vector<int> nums){
13     int n = nums.size();
14     vector<int> res(n);
15     stack<int> st;
16     for(int i = n-1; i >= 0; i--){
17         while(!st.empty() && st.top() <= nums[i]){
18             st.pop();
19             // we want the value greater than nums[i], so we
20             // pop the value smaller and equal nums[i]
21         }
22         if(st.empty()) res[i] = -1;
23         else res[i] = st.top();
24         st.push(nums[i]);
25     }
26     return res;
27 }
28
29 int main(){
30     vector<int> res = monotonic_stack({2,1,2,4,3});
31     for(auto r:res){
32         cout << r << " ";
33     }
34 }

```

## 11 Z\_Original\_Code/Flow

### 11.1 dicnic

```

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

```

```

58     }
59     return flow;
60 }
61 };

```

## 12 Z\_Original\_Code/Graph

### 12.1 planar

```

1 #include <iostream>
2 #include <vector>
3 #include <unordered_set>
4
5 using namespace std;
6
7 class Graph {
8 public:
9     int V;
10    vector<vector<int>>> adj;
11    Graph(int vertices) : V(vertices), adj(vertices) {}
12    void addEdge(int u, int v) {
13        adj[u].push_back(v);
14        adj[v].push_back(u);
15    }
16 };
17
18 bool containsSubgraph(const Graph& graph, const vector<int>&
    subgraph) {
19    unordered_set<int> subgraphVertices(subgraph.begin(),
        subgraph.end());
20    for (int vertex : subgraphVertices) {
21        for (int neighbor : graph.adj[vertex]) {
22            if (subgraphVertices.count(neighbor) == 0) {
23                bool found = true;
24                for (int v : subgraph) {
25                    if (v != vertex && v != neighbor) {
26                        if (graph.adj[v].size() < 3) {
27                            found = false;
28                            break;
29                        }
30                    }
31                }
32                if (found)
33                    return true;
34            }
35        }
36    }
37    return false;
38 }
39
40 bool isPlanar(const Graph& graph) {
41     // Subgraphs isomorphic to K and K ,
42     vector<int> k5 = {0, 1, 2, 3, 4}; // Vertices of K
43     vector<int> k33a = {0, 1, 2}; // Vertices of K
44     // (part A)
45     vector<int> k33b = {3, 4, 5}; // Vertices of K
46     // (part B)
47
48     if (containsSubgraph(graph, k5) || containsSubgraph(graph,
        k33a) || containsSubgraph(graph, k33b)) {
49         return false; // The graph is non-planar
50     }
51 }

```

```

48     }
49     return true; // The graph is planar
50 }
51
52 int main() {
53     int vertices, edges;
54     cin >> vertices;
55     cin >> edges;
56
57     Graph graph(vertices);
58     for (int i = 0; i < edges; ++i) {
59         int u, v;
60         cin >> u >> v;
61         graph.addEdge(u, v);
62     }
63     if (isPlanar(graph)) {
64         cout << "The graph is planar." << endl;
65     } else {
66         cout << "The graph is non-planar." << endl;
67     }
68
69     return 0;
70 }

```

### 12.2 Dijkstra

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define maxn 200005
5 vector<int> dis(maxn, -1);
6 vector<int> parent(maxn, -1);
7 vector<bool> vis(maxn, false);
8 vector<vector<pair<int, int>>> graph;
9 void dijkstra(int source) {
10     dis[source] = 0;
11
12     priority_queue<pair<int, int>, vector<pair<int, int>>,
        greater<pair<int, int>>> pq;
13     pq.push({0, source});
14     while (!pq.empty()) {
15         int from = pq.top().second;
16         pq.pop();
17         // cout << vis[from] << endl;
18         if (vis[from]) continue;
19         vis[from] = true;
20         for (auto next : graph[from]) {
21             int to = next.second;
22             int weight = next.first;
23             // cout << from << ' ' << to << ' ' << weight;
24             if (dis[from] + weight < dis[to] || dis[to] == -1) {
25                 dis[to] = dis[from] + weight;
26                 parent[to] = from;
27                 pq.push({dis[from] + weight, to});
28             }
29         }
30     }
31 }
32
33 int main() {
34     graph = {
35         {{4, 1}, {5, 3}},
36         {{3, 3}},
37         {{}},
38         {{4, 0}, {2, 1}, {7, 2}}
39     };
40
41     dijkstra(0);
42
43     for (int i = 0; i < 4; ++i) {
44         for (int j = 0; j < 4; ++j) {
45             cout << dis[i][j] << " ";
46         }
47     }
48 }

```

```

37     {{}},
38     {{4, 0}, {2, 1}, {7, 2}}
39 };
40 dijkstra(0);
41 for (int i = 0; i < 4; ++i) {
42     cout << dis[i] << " ";
43 }
44 for (int i = 0; i < 4; ++i) {
45     cout << parent[i] << " ";
46 }
47 }

```

### 12.3 Floyd\_Warshall

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 #define maxn 2005
5 vector<vector<int>>> dis(maxn, vector<int>(maxn, 9999999));
6 vector<vector<int>>> mid(maxn, vector<int>(maxn, -1));
7 vector<vector<pair<int, int>>> graph;
8
9 void floyd_warshall(int n) { // n is n nodes
10     for (int i = 0; i < n; ++i) {
11         for (auto path : graph[i]) {
12             dis[i][path.second] = path.first;
13         }
14     }
15     for (int i = 0; i < n; ++i) {
16         dis[i][i] = 0;
17         for (int k = 0; k < n; ++k) {
18             for (int j = 0; j < n; ++j) {
19                 for (int l = 0; l < n; ++l) {
20                     if (dis[i][k] + dis[k][j] < dis[i][j] || dis[i][j]
                        == -1) {
21                         dis[i][j] = dis[i][k] + dis[k][j];
22                         mid[i][j] = k; // 由 i 點走到 j 點經過 k 點
23                     }
24                 }
25             }
26         }
27     }
28 }
29 void find_path(int s, int t) { // 印出最短 徑
30     if (mid[s][t] == -1) return; // 沒有中繼點就結束
31     find_path(s, mid[s][t]); // 前半段最短 徑
32     cout << mid[s][t]; // 中繼點
33     find_path(mid[s][t], t); // 後半段最短 徑
34 }
35
36 int main() {
37     graph = {
38         {{4, 1}, {5, 3}},
39         {{3, 3}},
40         {{}},
41         {{4, 0}, {2, 1}, {7, 2}}
42     };
43
44     floyd_warshall(4);
45     for (int i = 0; i < 4; ++i) {
46         for (int j = 0; j < 4; ++j) {
47             cout << dis[i][j] << " ";
48         }
49     }
50     find_path(0, 2);
51 }

```

48 }

## 12.4 2\_sat

```

1 #include <iostream>
2 #include <vector>
3 #include <stack>
4 #include <algorithm>
5
6 using namespace std;
7
8 class TwoSAT {
9 public:
10     TwoSAT(int n) : n(n), graph(2 * n), visited(2 * n, false) {}
11
12     void addClause(int a, int b) { // 0-base;
13         a *= 2;
14         b *= 2;
15         // Add implications (~a => b) and (~b => a)
16         graph[a ^ 1].push_back(b);
17         graph[b ^ 1].push_back(a);
18     }
19
20     bool solve() {
21         // Find SCCs and check for contradictions
22         for (int i = 0; i < 2 * n; ++i) {
23             if (!visited[i]) {
24                 dfs1(i);
25             }
26         }
27         reverse(processingOrder.begin(), processingOrder.end()); // topological sort
28         for (int i = 0; i < 2 * n; ++i) {
29             visited[i] = false;
30         }
31         for (int node : processingOrder) {
32             if (!visited[node]) {
33                 scc.clear();
34                 dfs2(node);
35                 if (!checkSCCConsistency()) {
36                     return false;
37                 }
38             }
39         }
40
41         return true;
42     }
43
44 private:
45     int n;
46     vector<vector<int>> graph;
47     vector<bool> visited;
48     vector<int> processingOrder;
49     vector<int> scc;
50
51     void dfs1(int node) {
52         visited[node] = true;
53         for (int neighbor : graph[node]) {
54             if (!visited[neighbor]) {
55                 dfs1(neighbor);
56             }
57         }
58     }

```

```

58         processingOrder.push_back(node);
59     }
60
61     void dfs2(int node) {
62         visited[node] = true;
63         scc.push_back(node);
64         for (int neighbor : graph[node]) {
65             if (!visited[neighbor]) {
66                 dfs2(neighbor);
67             }
68         }
69     }
70
71     bool checkSCCConsistency() {
72         for (int node : scc) {
73             if (find(scc.begin(), scc.end(), node ^ 1) != scc.end()) {
74                 return false; // Contradiction found in the same SCC
75             }
76         }
77         return true;
78     }
79 };
80
81 int main() {
82     int n, m;
83     cin >> n >> m; // Number of variables and clauses
84
85     TwoSAT twoSat(n);
86
87     for (int i = 0; i < m; ++i) {
88         int a, b;
89         cin >> a >> b;
90         twoSat.addClause(a, b);
91     }
92
93     if (twoSat.solve()) {
94         cout << "Satisfiable" << endl;
95     } else {
96         cout << "Unsatisfiable" << endl;
97     }
98
99     return 0;
100 }

```

## 12.5 bipartite\_matching

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int MAXN = 100;
4
5 struct Bipartite_matching {
6     int mx[MAXN], my[MAXN], vy[MAXN]; // matchX, matchY, visitY
7     vector<int> edge[MAXN]; // adjacent list;
8     int x_cnt;
9     bool dfs(int x) {
10         for (auto y : edge[x]) { // 對 x 可以碰到的邊進 檢查
11             if (vy[y] == 1) continue; // 避免遞迴 error
12         }
13     }

```

```

14         vy[y] = 1;
15         if (my[y] == -1 || dfs(my[y])) { // 分析 3
16             mx[x] = y;
17             my[y] = x;
18             return true;
19         }
20     }
21     return false; // 分析 4
22 }
23
24 int bipartite_matching() {
25     memset(mx, -1, sizeof(mx)); // 分析 1,2
26     memset(my, -1, sizeof(my));
27     int ans = 0;
28     for (int i = 0; i < x_cnt; ++i) { // 對每一個 x 節點進 DFS(最大匹配)
29         memset(vy, 0, sizeof(vy));
30         if (dfs(i)) ans++;
31     }
32     return ans;
33 }
34
35 vector<vector<int>> get_match() {
36     vector<vector<int>> res;
37     for (int i = 0; i < x_cnt; ++i) {
38         if (mx[i] != -1) {
39             res.push_back({i, mx[i]});
40         }
41     }
42     return res;
43 }
44
45 void add_edge(int i, int j) {
46     edge[i].push_back(j);
47 }
48
49 void init(int x) {
50     x_cnt = x;
51 }
52
53 };
54
55 int main() {
56     /*
57     0 3
58     0 4
59     1 3
60     1 5
61     2 3
62     2 4
63     2 5
64     */
65     Bipartite_matching bm;
66     for (int i = 0; i < 7; ++i) {
67         int a, b;
68         cin >> a >> b;
69         bm.add_edge(a, b);
70     }
71     bm.init(3);
72     cout << bm.bipartite_matching() << endl;
73     auto match = bm.get_match();
74     for (auto t : match) {
75         cout << t[0] << " " << t[1] << endl;
76     }
77 }

```



## 12.6 tarjan-SCC

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 const int n = 16;
4 vector<vector<int>> graph;
5 int visit[n], low[n], t = 0;
6 int st[n], top = 0;
7 bool instack[n];
8 int contract[n]; // 每個點收縮到的點
9 vector<vector<int>> block;
10 void dfs(int x, int parent) {
11     // cout << x << endl;
12     visit[x] = low[x] = ++t;
13     st[top++] = x;
14     instack[x] = true;
15     for(auto to: graph[x]) {
16         if(!visit[to])
17             dfs(to, x);
18
19         if(instack[to])
20             low[x] = min(low[x], low[to]);
21     }
22     if(visit[x] == low[x]) { // scc 裡最早拜訪的
23         int j;
24         block.push_back({});
25         do {
26             j = st[--top];
27             instack[j] = false;
28             block[block.size() - 1].push_back(j);
29             contract[j] = x;
30         } while(j != x);
31     }
32 }
33 int main() {
34     graph = {
35         {1},
36         {3, 4, 5},
37         {6},
38         {2},
39         {7},
40         {11, 15},
41         {2, 3},
42         {4, 6, 9},
43         {},
44         {},
45         {},
46         {15},
47         {14},
48         {13, 5},
49         {15},
50         {10, 12, 13}
51     };
52     for(int i = 0; i < n; i++) {
53         if (!visit[i])
54             dfs(i, i);
55     }
56     for(auto t: block) {
57         for(auto x: t) {
58             cout << x << " ";
59         } cout << endl;
60     }
61 }

```

## 12.7 topological\_sort

```

1 #include <bits/stdc++.h>
2 using namespace std;
3 vector<vector<int>> graph;
4 vector<int> visit(10, 0);
5 vector<int> order;
6 int n;
7 bool cycle; // 記錄DFS的過程中是否偵測到環
8 void DFS(int i)
9 {
10     if (visit[i] == 1) {cycle = true; return;}
11     if (visit[i] == 2) return;
12     visit[i] = 1;
13     for(auto to: graph[i])
14         DFS(to);
15     visit[i] = 2;
16     order.push_back(i);
17 }
18
19 int main() {
20     graph = {
21         {1, 2},
22         {3},
23         {3, 4},
24         {4},
25         {}
26     };
27     n = 5;
28     cycle = false;
29     for (int i = 0; i < n; ++i) {
30         if (!visit[i])
31             DFS(i);
32     }
33     if (cycle)
34         cout << "圖上有環";
35     else
36         for (int i = n - 1; i >= 0; --i)
37             cout << order[i];
38 }

```

## 13 Z\_Original\_Code/Math

### 13.1 extgcd

```

1 #include <bits/stdc++.h>
2 using namespace std;
3
4 int extgcd(int a, int b, int &x, int &y) // 擴展歐幾 得算法
5 {
6     if (b == 0)
7     {
8         x = 1;
9         y = 0;
10        return a; // 到達遞歸邊界開始向上一層返回
11    }
12 }

```

```

13 }
14 int r = extgcd(b, a % b, x, y);
15 int temp = y; // 把x y變成上一層的
16 y = x - (a / b) * y;
17 x = temp;
18 return r; // 得到a b的最大公因數
19 }
20
21 int main() {
22     int a = 55, b = 80;
23     int x, y;
24     int GCD = extgcd(a, b, x, y);
25     cout << "GCD: " << GCD << endl;
26     cout << x << " " << y << endl;
27     cout << a * x + b * y << endl;
28 }

```

## 14 Z\_Original\_Code/Tree

### 14.1 LCA

```

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

```

```

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

```

## 14.2 diameter

```

1 #include <bits/stdc++.h>
2
3 using namespace std;
4
5 vector<vector<int>> graph;
6 int diameter = 0;
7 int dfs(int start, int parent){
8     int h1 = 0,h2 = 0;
9     for(auto child: graph[start]){
10         if(child!= parent){
11             int h = dfs(child,start)+1;
12             if(h>h1){
13                 h2= h1;
14                 h1 = h;
15             }
16             else if(h>h2){
17                 h2 = h;
18             }
19         }
20     }
21     diameter = max(diameter,h1+h2);
22     return h1;
23 }
24
25 int main(){

```

```

26 graph = {
27     {1,3},
28     {0},
29     {3},
30     {0,2,4},
31     {3}
32 };
33 dfs(0,-1);
34 cout << diameter<<endl;
35 }

```

## 14.3 radius

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 // Perform DFS to find the farthest node and its distance
4 // from the given node
5 pair<int, int> dfs(int node, int distance, vector<bool>&
6 visited, const vector<vector<int>>& adj_list) {
7     visited[node] = true;
8     int max_distance = distance;
9     int farthest_node = node;
10
11     for (int neighbor : adj_list[node]) {
12         if (!visited[neighbor]) {
13             auto result = dfs(neighbor, distance + 1, visited
14 , adj_list);
15             if (result.first > max_distance) {
16                 max_distance = result.first;
17                 farthest_node = result.second;
18             }
19         }
20     }
21
22     return make_pair(max_distance, farthest_node);
23 }
24
25 // Calculate the radius of the tree using DFS
26 int tree_radius(const vector<vector<int>>& adj_list) {
27     int num_nodes = adj_list.size();
28     vector<bool> visited(num_nodes, false);
29
30     // Find the farthest node from the root (node 0)
31     auto farthest_result = dfs(0, 0, visited, adj_list);
32
33     // Reset visited array
34     fill(visited.begin(), visited.end(), false);
35
36     // Calculate the distance from the farthest node
37     int radius = dfs(farthest_result.second, 0, visited,
38 adj_list).first;
39
40     return radius;
41 }
42
43 int main() {
44     vector<vector<int>> adj_list = {
45         {1, 2},
46         {0, 3, 4},
47         {0, 5},
48         {1},
49         {1},
50         {2}

```

```

47 };
48
49 int radius = tree_radius(adj_list);
50 cout << "Tree radius: " << radius << endl;
51
52 return 0;
53 }

```

## 14.4 bridge

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int n = 9;
4 vector<vector<int>> graph;
5 vector<int> visit(n,0);
6 vector<int> trace(n,0);
7 vector<vector<int>> bridge;
8 int t = 0;
9 void dfs(int x,int parent){
10     visit[x] = ++t;
11     trace[x] =x;// 最高祖先預設為自己
12     for(auto to:graph[x]){
13         if(visit[to]){ //back edge
14             if(to!=parent ) {
15                 trace[x] = to;
16             }
17         }else{ //treeedge
18             dfs(to,x);
19             if (visit[trace[to]] < visit[trace[x]])
20                 trace[x] = trace[to];
21
22             // 子樹回不到祖先暨自身。
23             if (visit[trace[to]] > visit[x])
24                 bridge.push_back({x,to});
25         }
26     }
27 }
28 int main(){
29     graph = {
30         {1,2},
31         {3},
32         {5,6},
33         {7},
34         {},
35         {},
36         {},
37         {8},
38         {4},
39     };
40     for(int i =0;i<9;i++){
41         if(!visit[i])
42             dfs(i,-1);
43     }
44     for(auto x: bridge){
45         cout << x[0]<<" "<< x[1]<<endl;
46     }
47 }

```

## 14.5 Articulation\_vertex

```

1 #include<bits/stdc++.h>
2 using namespace std;
3 const int n = 9;
4 int t = 0;
5 vector<int> disc(n, -1); // Discovery time
6 vector<int> low(n, -1); // Low time
7 vector<int> parent_array(n, -1); // Parent in DFS tree
8 vector<bool> visited(n, false);
9 vector<bool> is_articulation(n, false);
10 vector<vector<int>> graph;
11 void dfs_articulation(int node, int parent) {
12     visited[node] = true;
13     disc[node] = t;
14     low[node] = t;
15     t++;
16     int children = 0;
17
18     for (int neighbor : graph[node]) {
19         if (!visited[neighbor]) {
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             } else if (neighbor != parent) {
30                 low[node] = min(low[node], disc[neighbor]);
31             }
32         }
33
34         if (parent == -1 && children > 1) {
35             is_articulation[node] = true;
36         }
37     }
38
39     int main() {
40         graph = {
41             {1, 2},
42             {3},
43             {5, 6},
44             {7},
45             {},
46             {},
47             {8},
48             {4},
49         };
50         for (int i = 0; i < n; ++i) {
51             if (!visited[i]) {
52                 dfs_articulation(i, -1);
53             }
54         }
55         cout << "Articulation Points: ";
56         for (int i = 0; i < n; ++i) {
57             if (is_articulation[i]) {
58                 cout << i << " ";
59             }
60         }
61         cout << endl;
62     }

```

# NYCU\_Segmentree

## Codebook

### Contents

<b>1 Data Structure</b>	<b>1</b>	<b>5 Graph</b>	<b>5</b>	<b>8.3 Zvalue</b>	<b>10</b>
1.1 DSU	1	5.1 Bipartite Matching	5	8.4 KMP	10
1.2 Monotonic Queue	1	5.2 Tarjan SCC	5	8.5 Manacher	11
1.3 BIT	1	5.3 Bridge	6	<b>9 Tree</b>	<b>11</b>
1.4 Treap	1	5.4 2 SAT	6	9.1 LCA	11
1.5 Segment Tree	2	5.5 Kosaraju 2DFS	6	9.2 Diameter	11
1.6 Sparse Table	2	5.6 Dijkstra	7	9.3 Radius	11
1.7 Monotonic Stack	2	5.7 Floyd Warshall	7	<b>10 Z_Original_Code/Data_Structure</b>	<b>12</b>
<b>2 Flow</b>	<b>3</b>	5.8 Articulation Vertex	7	10.1dsu-class	12
2.1 Dinic	3	5.9 Topological Sort	7	10.2monotonic-queue	12
<b>3 Formula</b>	<b>3</b>	5.10Closest Pair	8	10.3BIT	12
3.1 formula	3	5.11Planar	8	10.4segment-tree-simple-add	13
3.1.1 Pick 公式	3	5.12Heavy Light Decomposition	8	10.5monotonic-stack	13
3.1.2 圖論	3	5.13Centroid Decomposition	8	<b>11 Z_Original_Code/Flow</b>	<b>13</b>
3.1.3 dinic 特殊圖複雜度	3	<b>6 Math</b>	<b>8</b>	11.1dicnic	13
3.1.4 0-1 分數規劃	3	6.1 fpow	8	<b>12 Z_Original_Code/Graph</b>	<b>14</b>
3.1.5 學長公式	4	6.2 modinv	8	12.1planar	14
3.1.6 基本數論	4	6.3 PollardRho	9	12.2Dijkstra	14
3.1.7 排組公式	4	6.4 extGCD	9	12.3Floyd_Warshall	14
3.1.8 冪次, 冪次和	4	6.5 random	9	12.42_sat	15
3.1.9 Burnside's lemma	4	6.6 EulerTotientFunction	9	12.5bipartite_matching	15
3.1.10Count on a tree	4	6.7 FFT	9	12.6tarjan-SCC	16
<b>4 Geometry</b>	<b>4</b>	6.8 MillerRabin	9	12.7topological_sort	16
4.1 Sort by Angle	4	6.9 mu	9	<b>13 Z_Original_Code/Math</b>	<b>16</b>
4.2 Convex Hull	4	<b>7 Misc</b>	<b>10</b>	13.1extgcd	16
		7.1 pbds	10	<b>14 Z_Original_Code/Tree</b>	<b>16</b>
		7.2 Misc	10	14.1LCA	16
		7.3 Mo'sAlgorithm	10	14.2diameter	17
		<b>8 String</b>	<b>10</b>	14.3radius	17
		8.1 Hashing	10	14.4bridge	17
		8.2 Trie	10	14.5Articulation_vertex	17