

Contents

1	Basic	1
1.1	Run	1
1.2	Default	1
1.3	Black Magic	1
1.4	Python	2
1.5	Binary Search	2
1.6	Ternary Search	2
2	Data Structure	2
2.1	Disjoint Set	2
2.2	BIT RARSQ	2
2.3	zkw RMQ	2
2.4	Segment Tree RARMQ	3
3	Graph	3
3.1	Dijkstra	3
3.2	SPFA(negative cycle)	4
3.3	Floyd Warshall	4
3.4	Topological Sort	4
3.5	Tree Diameter	4
3.6	Directed MST	5
3.7	Kosaraju SCC	5
3.8	BCC	6
3.9	LCA	6
4	Flow & Matching	6
4.1	Relation	6
4.2	Bipartite Matching	6
4.3	KM	7
4.4	Dinic	7
4.5	MCMF	7
5	String	8
5.1	Manacher	8
5.2	Trie	8
5.3	Z-value	9
6	DP	9
6.1	LIS	9
6.2	LCS	9
7	Math	9
7.1	Extended GCD	9
7.2	Gaussian Elimination + det	9
7.3	Prime Table	10
7.4	Phi	10
7.5	Chinese Remainder Thm	10
7.6	Josephus	10
7.7	Catalan	10
7.8	Matrix Multiplication	10
7.9	Fibonacci	11
8	Geometry	11
8.1	Point	11
8.2	Line	11
8.3	Area	11
8.4	Convex Hull	12

1 Basic

1.1 Run

```
1 #use -> sh run.sh {name}
2 g++ -O2 -std=c++14 -Wall -Wextra -Wshadow -o $1 $1.cpp
3 ./ $1 < t.in > t.out
```

1.2 Default

```
1 #include <bits/stdc++.h>
2 using namespace std;
3 using LL = long long;
4 #define IOS ios_base::sync_with_stdio(0); cin.tie(0);
5 #define pb push_back
6 #define eb emplace_back
7 const int INF = 1e9;
8 const int MOD = 1e9 + 7;
9 const double EPS = 1e-6;
10 const int MAXN = 0;
11
```

```
12 int main() {
13
14 }
```

1.3 Black Magic

```
2 1 #include <bits/stdc++.h>
2 2 #include <ext/pb_ds/assoc_container.hpp>
2 3 #include <ext/pb_ds/tree_policy.hpp>
2 4 #include <ext/pb_ds/priority_queue.hpp>
3 5 using namespace std;
3 6 using namespace __gnu_pbds;
3 7 using set_t =
4 8     tree<int, null_type, less<int>, rb_tree_tag,
4 9         tree_order_statistics_node_update>;
4 10 using map_t =
5 11     tree<int, int, less<int>, rb_tree_tag,
5 12         tree_order_statistics_node_update>;
6 13 using heap_t =
6 14     __gnu_pbds::priority_queue<int>;
6 15 using ht_t =
6 16     gp_hash_table<int, int>;
6 17 int main() {
7 18     //set-----
7 19     set_t st;
8 20     st.insert(5); st.insert(6);
8 21     st.insert(3); st.insert(1);
8 22
8 23     // the smallest is (0), biggest is (n-1), kth small
9     // is (k-1)
9 24     int num = *st.find_by_order(0);
9 25     cout << num << '\n'; // print 1
9 26
9 27     num = *st.find_by_order(st.size() - 1);
9 28     cout << num << '\n'; // print 6
9 29
9 30     // find the index
10 31     int index = st.order_of_key(6);
10 32     cout << index << '\n'; // print 3
10 33
10 34     // check if there exists x
11 35     int x = 5;
11 36     int check = st.erase(x);
11 37     if (check == 0) printf("st not contain 5\n");
11 38     else if (check == 1) printf("st contain 5\n");
11 39
12 40     //tree policy like set
41     st.insert(5); st.insert(5);
42     cout << st.size() << '\n'; // print 4
43
44     //map-----
45     map_t mp;
46     mp[1] = 2;
47     cout << mp[1] << '\n';
48     auto tmp = *mp.find_by_order(0); // pair
49     cout << tmp.first << " " << tmp.second << '\n';
50
51     //heap-----
52     heap_t h1, h2;
53     h1.push(1); h1.push(3);
54     h2.push(2); h2.push(4);
55     h1.join(h2);
56     cout << h1.size() << h2.size() << h1.top() << '\n';
57     // 404
58
59     //hash-table-----
60     ht_t ht;
61     ht[85] = 5;
62     ht[89975] = 234;
63     for (auto i : ht) {
64         cout << i.first << " " << i.second << '\n';
65     }
66 }
```

1.4 Python

```

1  ### EOF
2  while True:
3      try:
4          pass
5      except EOFError:
6          break
7  ###math
8  import math
9
10 math.ceil(x)#上高斯
11 math.floor(x)#下高斯
12 math.factorial(x)#接乘
13 math.fabs(x)#絕對值
14 math.fsum(arr)#跟sum一樣但更精確(小數點問題)
15 math.gcd(x, y)#bj4
16 math.exp(x)#e^x
17 math.log(x, base)
18 math.log2(x)#2為底
19 math.log10(x)#10為底
20 math.sqrt(x)
21 math.pow(x, y, mod)#精確些(float型態) MOD!!!
22 math.sin(x)# cos tan asin acos atan atan2(弧度) sinh
    cosh tanh acosh asinh atanh
23 math.hypot(x, y)#歐幾里德範數
24 math.degrees(x)#x從弧度轉角度
25 math.radians(x)#x從角度轉弧度
26 math.gamma(x)#x的gamma函數
27 math.pi#常數
28 math.e#常數
29 math.inf
30
31 ### ascii
32 ord(x)#char to asc
33 chr(x)#asc to char
34
35 x.encode().hex()#string to hex
36 ### reverse string
37 string = "abc"
38 string_reverse = string[::-1]

```

1.5 Binary Search

```

1 lower_bound(a, a + n, k);    //最左邊 ≥ k 的位置
2 upper_bound(a, a + n, k);    //最左邊 > k 的位置
3 upper_bound(a, a + n, k) - 1; //最右邊 ≤ k 的位置
4 lower_bound(a, a + n, k) - 1; //最右邊 < k 的位置
5 [lower_bound, upper_bound)   //等於 k 的範圍
6 equal_range(a, a + n, k);

```

1.6 Ternary Search

```

1 const double EPS = 1e-6;
2 // target function
3 double f(double x) { return x * x; }
4 double ternarySearch() {
5     double L = -1e5, R = 1e5;
6     while (R - L > EPS) {
7         double mr = (L + R) / 2.0;
8         double ml = (L + mr) / 2.0;
9         if (f(ml) < f(mr)) {
10             R = mr;
11         } else {
12             L = ml;
13         }
14     }
15     return L;
16 }

```

2 Data Structure

2.1 Disjoint Set

```

1 // 0-base
2 const int MAXN = 1000;
3 int boss[MAXN];
4 void init(int n) {
5     for (int i = 0; i < n; i++) {
6         boss[i] = -1;
7     }
8 }
9 int find(int x) {
10     if (boss[x] < 0) {
11         return x;
12     }
13     return boss[x] = find(boss[x]);
14 }
15 bool uni(int a, int b) {
16     a = find(a);
17     b = find(b);
18     if (a == b) {
19         return false;
20     }
21     if (boss[a] > boss[b]) {
22         swap(a, b);
23     }
24     boss[a] += boss[b];
25     boss[b] = a;
26     return true;
27 }

```

2.2 BIT RARSQ

```

1 // 1-base
2 #define lowbit(k) (k & -k)
3
4 int n;
5 vector<int> B1, B2;
6
7 void add(vector<int> &tr, int id, int val) {
8     for (; id <= n; id += lowbit(id)) {
9         tr[id] += val;
10    }
11 }
12 void range_add(int l, int r, int val) {
13     add(B1, l, val);
14     add(B1, r + 1, -val);
15     add(B2, l, val * (1 - 1));
16     add(B2, r + 1, -val * r);
17 }
18 int sum(vector<int> &tr, int id) {
19     int ret = 0;
20     for (; id >= 1; id -= lowbit(id)) {
21         ret += tr[id];
22     }
23     return ret;
24 }
25 int prefix_sum(int id) {
26     return sum(B1, id) * id - sum(B2, id);
27 }
28 int range_sum(int l, int r) {
29     return prefix_sum(r) - prefix_sum(l - 1);
30 }

```

2.3 zkw RMQ

```

1 // 0-base
2 const int INF = 1e9;
3 const int MAXN = ;
4
5 int n;

```

```

6 int a[MAXN], tr[MAXN << 1];
7
8 // !!! remember to call this function
9 void build() {
10     for (int i = 0; i < n; i++) {
11         tr[i + n] = a[i];
12     }
13     for (int i = n - 1; i > 0; i--) {
14         tr[i] = max(tr[i << 1], tr[i << 1 | 1]);
15     }
16 }
17 void update(int id, int val) {
18     for (tr[id += n] = val; id > 1; id >>= 1) {
19         tr[id >> 1] = max(tr[id], tr[id ^ 1]);
20     }
21 }
22 int query(int l, int r) { // [l, r]
23     int ret = -INF;
24     for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
25         if (l & 1) {
26             ret = max(ret, tr[l++]);
27         }
28         if (r & 1) {
29             ret = max(ret, tr[--r]);
30         }
31     }
32     return ret;
33 }

```

2.4 Segment Tree RARMQ

```

1 struct Node {
2     int val, tag;
3     Node *lc, *rc;
4     Node() : lc(nullptr), rc(nullptr), tag(0) {}
5     void pull() {
6         if (!lc) {
7             val = rc->val;
8         } else if (!rc) {
9             val = lc->val;
10        } else {
11            val = max(lc->val, rc->val);
12        }
13    }
14    void push() {
15        if (lc) {
16            lc->tag += tag;
17            lc->val += tag;
18        }
19        if (rc) {
20            rc->tag += tag;
21            rc->val += tag;
22        }
23        tag = 0;
24    }
25 };
26 struct SegmentTree {
27     Node *root;
28     SegmentTree() : root(nullptr) {}
29     void build(Node* &T, int l, int r, const
30         vector<int> &o) {
31         T = new Node();
32         if (l == r) {
33             T->val = o[l];
34             return;
35         }
36         int mid = (l + r) / 2;
37         build(T->lc, l, mid, o);
38         build(T->rc, mid + 1, r, o);
39         T->pull();
40     }
41     void update(Node* &T, int l, int r, int ql, int qr,
42         int v) {
43         if (ql <= l && r <= qr) {
44             T->val += v;
45             T->tag += v;

```

```

44         return;
45     }
46     T->push();
47     int mid = (l + r) / 2;
48     if (qr <= mid) {
49         update(T->lc, l, mid, ql, qr, v);
50     } else if (mid < ql) {
51         update(T->rc, mid + 1, r, ql, qr, v);
52     } else {
53         update(T->lc, l, mid, ql, mid, v);
54         update(T->rc, mid + 1, r, mid + 1, qr, v);
55     }
56     T->pull();
57 }
58 int query(Node* &T, int l, int r, int ql, int qr) {
59     if (ql <= l && r <= qr) {
60         return T->val;
61     }
62     T->push();
63     int mid = (l + r) / 2;
64     if (qr <= mid) {
65         return query(T->lc, l, mid, ql, qr);
66     } else if (mid < ql) {
67         return query(T->rc, mid + 1, r, ql, qr);
68     } else {
69         return max(query(T->lc, l, mid, ql, mid),
70             query(T->rc, mid + 1, r, mid + 1,
71                 qr));
72     }
73 };

```

3 Graph

3.1 Dijkstra

```

1 // 0-base
2 const LL INF = 1e18;
3 const int MAXN = ;
4 struct Edge {
5     int to;
6     LL cost;
7     Edge(int v, LL c) : to(v), cost(c) {}
8     bool operator < (const Edge &other) const {
9         return cost > other.cost;
10    }
11 };
12
13 int n;
14 LL dis[MAXN];
15 vector<Edge> G[MAXN];
16
17 void init() {
18     for (int i = 0; i < n; i++) {
19         G[i].clear();
20         dis[i] = INF;
21     }
22 }
23 void Dijkstra(int st, int ed = -1) {
24     priority_queue<Edge> pq;
25     pq.emplace(st, 0);
26     dis[st] = 0;
27     while (!pq.empty()) {
28         auto now = pq.top();
29         pq.pop();
30         if (now.to == ed) {
31             return;
32         }
33         if (now.cost > dis[now.to]) {
34             continue;
35         }
36         for (auto &e : G[now.to]) {
37             if (dis[e.to] > now.cost + e.cost) {
38                 dis[e.to] = now.cost + e.cost;

```

```

39     pq.emplace(e.to, dis[e.to]);
40 }
41 }
42 }
43 }

```

3.2 SPFA(negative cycle)

```

1 // 0-base
2 const LL INF = 1e18;
3 const int MAXN = ;
4 struct Edge {
5     int to;
6     LL cost;
7     Edge(int v, LL c) : to(v), cost(c) {}
8 };
9
10 int n;
11 LL dis[MAXN];
12 vector<Edge> G[MAXN];
13
14 void init() {
15     for (int i = 0; i < n; i++) {
16         G[i].clear();
17         dis[i] = INF;
18     }
19 }
20 bool SPFA(int st) {
21     vector<int> cnt(n, 0);
22     vector<bool> inq(n, false);
23     queue<int> q;
24
25     q.push(st);
26     dis[st] = 0;
27     inq[st] = true;
28     while (!q.empty()) {
29         int now = q.front();
30         q.pop();
31         inq[now] = false;
32         for (auto &e : G[now]) {
33             if (dis[e.to] > dis[now] + e.cost) {
34                 dis[e.to] = dis[now] + e.cost;
35                 if (!inq[e.to]) {
36                     cnt[e.to]++;
37                     if (cnt[e.to] > n) {
38                         // negative cycle
39                         return false;
40                     }
41                     inq[e.to] = true;
42                     q.push(e.to);
43                 }
44             }
45         }
46     }
47     return true;
48 }

```

3.3 Floyd Warshall

```

1 // 0-base
2 // G[i][i] < 0 -> negative cycle
3 const LL INF = 1e18;
4 const int MAXN = ;
5
6 int n;
7 LL G[MAXN][MAXN];
8
9 void init() {
10     for (int i = 0; i < n; i++) {
11         for (int j = 0; j < n; j++) {
12             G[i][j] = INF;
13         }
14         G[i][i] = 0;
15     }
16 }

```

```

15 }
16 }
17 void floyd() {
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                 if (G[i][k] != INF && G[k][j] != INF) {
22                     G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
23                 }
24             }
25         }
26     }
27 }

```

3.4 Topological Sort

```

1 // 0-base
2 // if ret.size < n -> cycle
3 int n;
4 vector<vector<int>> G;
5
6 vector<int> topoSort() {
7     vector<int> indeg(n), ret;
8     for (auto &li : G) {
9         for (int x : li) {
10             ++indeg[x];
11         }
12     }
13     // use priority queue for lexic. largest ans
14     queue<int> q;
15     for (int i = 0; i < n; i++) {
16         if (!indeg[i]) {
17             q.push(i);
18         }
19     }
20     while (!q.empty()) {
21         int u = q.front();
22         q.pop();
23         ret.pb(u);
24         for (int v : G[u]) {
25             if (--indeg[v] == 0) {
26                 q.push(v);
27             }
28         }
29     }
30     return ret;
31 }

```

3.5 Tree Diameter

```

1 // 0-base;
2 const int MAXN = ;
3
4 struct Edge {
5     int to;
6     int cost;
7     Edge(int v, int c) : to(v), cost(c) {}
8 };
9
10 int n, d = 0;
11 int d1[MAXN], d2[MAXN];
12 vector<Edge> G[MAXN];
13 // dfs(0, -1);
14 void dfs(int u, int from) {
15     d1[u] = d2[u] = 0;
16     for (auto e : G[u]) {
17         if (e.to == from) {
18             continue;
19         }
20         dfs(e.to, u);
21         int t = d1[e.to] + e.cost;
22         if (t > d1[u]) {
23             d2[u] = d1[u];
24             d1[u] = t;
25         }
26     }
27 }

```

```

24     d1[u] = t;
25 } else if (t > d2[u]) {
26     d2[u] = t;
27 }
28 }
29 d = max(d, d1[u] + d2[u]);
30 }

```

3.6 Directed MST

```

1 // 0-base
2 const LL INF = 1e18;
3 const int MAXN = ;
4
5 struct Edge {
6     int from;
7     int to;
8     LL cost;
9     Edge(int u, int v, LL c) : from(u), to(v), cost(c) {}
10 };
11
12 struct DMST {
13     int n;
14     int vis[MAXN], pre[MAXN], id[MAXN];
15     LL in[MAXN];
16     vector<Edge> edges;
17     void init(int _n) {
18         n = _n;
19         edges.clear();
20     }
21     void add_edge(int from, int to, LL cost) {
22         edges.eb(from, to, cost);
23     }
24     LL run(int root) {
25         LL ret = 0;
26         while (true) {
27             for (int i = 0; i < n; i++) {
28                 in[i] = INF;
29             }
30
31             // find in edge
32             for (auto &e : edges) {
33                 if (e.cost < in[e.to] && e.from != e.to) {
34                     pre[e.to] = e.from;
35                     in[e.to] = e.cost;
36                 }
37             }
38
39             // check in edge
40             for (int i = 0; i < n; i++) {
41                 if (i == root) {
42                     continue;
43                 }
44                 if (in[i] == INF) {
45                     return -1;
46                 }
47             }
48
49             int nodelist = 0;
50             memset(id, -1, sizeof(id));
51             memset(vis, -1, sizeof(vis));
52             in[root] = 0;
53
54             // find cycles
55             for (int i = 0; i < n; i++) {
56                 ret += in[i];
57                 int v = i;
58                 while (vis[v] != i && id[v] == -1 && v != root) {
59                     vis[v] = i;
60                     v = pre[v];
61                 }
62                 if (id[v] == -1 && v != root) {
63                     for (int j = pre[v]; j != v; j = pre[j]) {
64                         id[j] = nodelist;

```

```

65         }
66         id[v] = nodelist++;
67     }
68 }
69
70 // no cycle
71 if (nodelist == 0) {
72     break;
73 }
74
75 for (int i = 0; i < n; i++) {
76     if (id[i] == -1) {
77         id[i] = nodelist++;
78     }
79 }
80
81 // grouping the vertices
82 for (auto &e : edges) {
83     int to = e.to;
84     e.from = id[e.from];
85     e.to = id[e.to];
86     if (e.from != e.to) {
87         e.cost -= in[to]; //!!!
88     }
89 }
90
91 n = nodelist;
92 root = id[root];
93 }
94 return ret;
95 }
96 };

```

3.7 Kosaraju SCC

```

1 // 0-base
2 int n;
3 vector<vector<int>> G, G2; // G2 = G rev
4 vector<bool> vis;
5 vector<int> s, color;
6 int sccCnt;
7 void dfs1(int u) {
8     vis[u] = true;
9     for (int v : G[u]) {
10         if (!vis[v]) {
11             dfs1(v);
12         }
13     }
14     s.pb(u);
15 }
16 void dfs2(int u) {
17     color[u] = sccCnt;
18     for (int v : G2[u]) {
19         if (!color[v]) {
20             dfs2(v);
21         }
22     }
23 }
24 void Kosaraju() {
25     sccCnt = 0;
26     for (int i = 0; i < n; i++) {
27         if (!vis[i]) {
28             dfs1(i);
29         }
30     }
31     for (int i = n - 1; i >= 0; i--) {
32         if (!color[s[i]]) {
33             ++sccCnt;
34             dfs2(s[i]);
35         }
36     }
37 }

```

3.8 BCC

```

1 typedef pair<int, int> PII;
2 int low[MXV], depth[MXV];
3 bool is_cut_vertex[MXV], visit[MXV];
4 vector<int> G[MXV];
5 vector<PII> BCC[MXV];
6 int bcc_cnt = 0;
7 stack<PII> st;
8
9 vector<pair<int, int>> my_cut_edge;
10
11 void dfs(int now, int cur_depth, int f) {
12     visit[now] = true;
13     depth[now] = low[now] = cur_depth;
14     int cut_son = 0;
15     for (auto i : G[now]) {
16         if (i == f) continue;
17         if (visit[i]) { // ancestor
18             if (depth[i] < depth[now]) { // #
19                 low[now] = min(low[now], depth[i]);
20                 st.push({now, i});
21             }
22         } else { // offspring
23             st.push({now, i});
24             dfs(i, cur_depth + 1, now);
25             cut_son += 1;
26             low[now] = min(low[now], low[i]);
27             if (low[i] >= depth[now]) {
28                 is_cut_vertex[now] = true;
29                 auto t = st.top();
30                 st.pop();
31                 while (t != make_pair(now, i)) {
32                     BCC[bcc_cnt].push_back(t);
33                     t = st.top();
34                     st.pop();
35                 }
36                 BCC[bcc_cnt].push_back(t);
37                 ++bcc_cnt;
38             }
39             // ###
40             if (low[i] > depth[now])
41                 my_cut_edge.push_back({now, i});
42         }
43     }
44     if (cur_depth == 0)
45         is_cut_vertex[now] = (cut_son != 1);
46     return;
47 }
48
49 bool is_2_edge_connected(int n) {
50     memset(visit, 0, sizeof(visit));
51     dfs(1, 0, -1);
52     return my_cut_edge.size() == 0;
53 }

```

3.9 LCA

```

1 const int LOG = 20;
2 vector<int> tin(MAXN), tout(MAXN), depth(MAXN);
3 int par[MAXN][LOG];
4 int timer = 0;
5 vector<int> G[MAXN];
6
7 void dfs(int u, int f) {
8     tin[u] = ++timer;
9     par[u][0] = f;
10    for (int v : G[u]) {
11        if (v != f) {
12            depth[v] = depth[u] + 1;
13            dfs(v, u);
14        }
15    }
16    tout[u] = ++timer;
17 }

```

```

18
19 void Doubling(int n) {
20     for (int j = 1; j < LOG; ++j) {
21         for (int i = 1; i <= n; ++i) {
22             par[i][j] = par[par[i][j-1]][j-1];
23         }
24     }
25 }
26
27 bool anc(int u, int v) { return tin[u] <= tin[v] &&
    tout[v] <= tout[u]; }
28
29 int LCA(int u, int v) {
30     if (depth[u] > depth[v]) {
31         swap(u, v);
32     }
33     if (anc(u, v)) {
34         return u;
35     }
36     for (int j = LOG - 1; j >= 0; --j) {
37         if (!anc(par[u][j], v)) u = par[u][j];
38     }
39     return par[u][0];
40 }
41
42 int dis(int u, int v) {
43     int lca = LCA(u, v);
44     return depth[u] + depth[v] - 2 * depth[lca];
45 }
46
47 /*
48 dfs(root, root);
49 Doubling(n);
50 */

```

4 Flow & Matching

4.1 Relation

```

1 1. 一般圖
2 |最大匹配| + |最小邊覆蓋| = |V|
3 |最大獨立集| + |最小點覆蓋| = |V|
4 |最大圖| = |補圖的最大獨立集|
5 2. 二分圖
6 |最大匹配| = |最小點覆蓋|
7 |最大獨立集| = |最小邊覆蓋|
8 |最大獨立集| = |V| - |最大匹配|
9 |最大圖| = |補圖的最大獨立集|

```

4.2 Bipartite Matching

```

1 // 0-base
2 const int MAXN = ;
3 int n;
4 vector<int> G[MAXN];
5 int vy[MAXN], my[MAXN];
6
7 bool match(int u) {
8     for (int v : G[u]) {
9         if (vy[v]) {
10             continue;
11         }
12         vy[v] = true;
13         if (my[v] == -1 || match(my[v])) {
14             my[v] = u;
15             return true;
16         }
17     }
18     return false;
19 }
20 int sol() {

```

```

21 int cnt = 0;
22 memset(my, -1, sizeof(my));
23 for (int i = 0; i < n; i++) {
24     memset(vy, 0, sizeof(vy));
25     if (match(i)) {
26         cnt++;
27     }
28 }
29 return cnt;
30 }

```

4.3 KM

```

1  const int INF = 1e9;
2  const int MAXN = ;
3  struct KM { //1-base
4      int n, G[MAXN][MAXN];
5      int lx[MAXN], ly[MAXN], my[MAXN];
6      bool vx[MAXN], vy[MAXN];
7      void init(int _n) {
8          n = _n;
9          for (int i = 1; i <= n; i++) {
10             for (int j = 1; j <= n; j++) {
11                 G[i][j] = 0;
12             }
13         }
14     }
15     bool match(int i) {
16         vx[i] = true;
17         for (int j = 1; j <= n; j++) {
18             if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
19                 vy[j] = true;
20                 if (!my[j] || match(my[j])) {
21                     my[j] = i;
22                     return true;
23                 }
24             }
25         }
26         return false;
27     }
28     void update() {
29         int delta = INF;
30         for (int i = 1; i <= n; i++) {
31             if (vx[i]) {
32                 for (int j = 1; j <= n; j++) {
33                     if (!vy[j]) {
34                         delta = min(delta, lx[i] + ly[j] - G[i][j]);
35                     }
36                 }
37             }
38         }
39         for (int i = 1; i <= n; i++) {
40             if (vx[i]) {
41                 lx[i] -= delta;
42             }
43             if (vy[i]) {
44                 ly[i] += delta;
45             }
46         }
47     }
48     int run() {
49         for (int i = 1; i <= n; i++) {
50             lx[i] = ly[i] = my[i] = 0;
51             for (int j = 1; j <= n; j++) {
52                 lx[i] = max(lx[i], G[i][j]);
53             }
54         }
55         for (int i = 1; i <= n; i++) {
56             while (true) {
57                 for (int i = 1; i <= n; i++) {
58                     vx[i] = vy[i] = 0;
59                 }
60                 if (match(i)) {
61                     break;
62                 } else {

```

```

63             update();
64         }
65     }
66     int ans = 0;
67     for (int i = 1; i <= n; i++) {
68         ans += lx[i] + ly[i];
69     }
70     return ans;
71 }
72 }
73 };

```

4.4 Dinic

```

1  #define eb emplace_back
2  const LL INF = 1e18;
3  const int MAXN = ;
4  struct Edge {
5      int to;
6      LL cap;
7      int rev;
8      Edge(int v, LL c, int r) : to(v), cap(c), rev(r) {}
9  };
10 struct Dinic {
11     int n;
12     int level[MAXN], now[MAXN];
13     vector<Edge> G[MAXN];
14     void init(int _n) {
15         n = _n;
16         for (int i = 0; i <= n; i++) {
17             G[i].clear();
18         }
19     }
20     void add_edge(int u, int v, LL c) {
21         G[u].eb(v, c, G[v].size());
22         // directed graph
23         G[v].eb(u, 0, G[u].size() - 1);
24         // undirected graph
25         // G[v].eb(u, c, G[u].size() - 1);
26     }
27     bool bfs(int st, int ed) {
28         fill(level, level + n + 1, -1);
29         queue<int> q;
30         q.push(st);
31         level[st] = 0;
32         while (!q.empty()) {
33             int u = q.front();
34             q.pop();
35             for (const auto &e : G[u]) {
36                 if (e.cap > 0 && level[e.to] == -1) {
37                     level[e.to] = level[u] + 1;
38                     q.push(e.to);
39                 }
40             }
41         }
42         return level[ed] != -1;
43     }
44     LL dfs(int u, int ed, LL limit) {
45         if (u == ed) {
46             return limit;
47         }
48         LL ret = 0;
49         for (int &i = now[u]; i < G[u].size(); i++) {
50             auto &e = G[u][i];
51             if (e.cap > 0 && level[e.to] == level[u] + 1) {
52                 LL f = dfs(e.to, ed, min(limit, e.cap));
53                 ret += f;
54                 limit -= f;
55                 e.cap -= f;
56                 G[e.to][e.rev].cap += f;
57                 if (!limit) {
58                     return ret;
59                 }
60             }
61         }
62         if (!ret) {

```

```

63     level[u] = -1;
64 }
65 return ret;
66 }
67 LL flow(int st, int ed) {
68     LL ret = 0;
69     while (bfs(st, ed)) {
70         fill(now, now + n + 1, 0);
71         ret += dfs(st, ed, INF);
72     }
73     return ret;
74 }
75 };

```

4.5 MCMF

```

1 // 0-base
2 const LL INF = 1e18;
3 const int MAXN = ;
4 struct Edge {
5     int u, v;
6     LL cost;
7     LL cap;
8     Edge(int _u, int _v, LL _c, LL _cap) : u(_u),
9         v(_v), cost(_c), cap(_cap) {}
10 };
11 struct MCMF { // inq times
12     int n, pre[MAXN], cnt[MAXN];
13     LL ans_flow, ans_cost, dis[MAXN];
14     bool inq[MAXN];
15     vector<int> G[MAXN];
16     vector<Edge> edges;
17     void init(int _n) {
18         n = _n;
19         edges.clear();
20         for (int i = 0; i < n; i++) {
21             G[i].clear();
22         }
23     }
24     void add_edge(int u, int v, LL c, LL cap) {
25         // directed
26         G[u].pb(edges.size());
27         edges.pb(u, v, c, cap);
28         G[v].pb(edges.size());
29         edges.pb(v, u, -c, 0);
30     }
31     bool SPFA(int st, int ed) {
32         for (int i = 0; i < n; i++) {
33             pre[i] = -1;
34             dis[i] = INF;
35             cnt[i] = 0;
36             inq[i] = false;
37         }
38         queue<int> q;
39         bool negcycle = false;
40         dis[st] = 0;
41         cnt[st] = 1;
42         inq[st] = true;
43         q.push(st);
44         while (!q.empty() && !negcycle) {
45             int u = q.front();
46             q.pop();
47             inq[u] = false;
48             for (int i : G[u]) {
49                 int v = edges[i].v;
50                 LL cost = edges[i].cost;
51                 LL cap = edges[i].cap;
52                 if (dis[v] > dis[u] + cost && cap > 0) {
53                     dis[v] = dis[u] + cost;
54                     pre[v] = i;
55                     if (!inq[v]) {
56                         q.push(v);
57                         cnt[v]++;
58                     }
59                 }
60             }
61         }
62         if (cnt[ed] == n + 2) {
63             negcycle = true;
64             break;
65         }
66     }
67     LL f = sendFlow(u, min(curFlow, edges[i].cap));
68     ans_cost += f * cost;
69     edges[i].cap -= f;
70     edges[i ^ 1].cap += f;
71     return f;
72 }
73 pair<LL, LL> run(int st, int ed) {
74     ans_flow = ans_cost = 0;
75     while (SPFA(st, ed)) {
76         ans_flow += sendFlow(ed, INF);
77     }
78     return make_pair(ans_flow, ans_cost);
79 }
80 };

```

```

60     inq[v] = true;
61     if (cnt[v] == n + 2) {
62         negcycle = true;
63         break;
64     }
65 }
66 }
67 }
68 }
69 }
70 }
71 return dis[ed] != INF;
72 }
73 LL sendFlow(int v, LL curFlow) {
74     if (pre[v] == -1) {
75         return curFlow;
76     }
77     int i = pre[v];
78     int u = edges[i].u;
79     LL cost = edges[i].cost;
80     LL f = sendFlow(u, min(curFlow, edges[i].cap));
81     ans_cost += f * cost;
82     edges[i].cap -= f;
83     edges[i ^ 1].cap += f;
84     return f;
85 }
86 pair<LL, LL> run(int st, int ed) {
87     ans_flow = ans_cost = 0;
88     while (SPFA(st, ed)) {
89         ans_flow += sendFlow(ed, INF);
90     }
91     return make_pair(ans_flow, ans_cost);
92 }
93 }
94 }
95 };

```

5 String

5.1 Manacher

```

1 int p[2 * MAXN];
2 int Manacher(const string &s) {
3     string st = "@#";
4     for (char c : s) {
5         st += c;
6         st += '#';
7     }
8     st += '$';
9     int id = 0, mx = 0, ans = 0;
10    for (int i = 1; i < st.length() - 1; i++) {
11        p[i] = (mx > i ? min(p[2 * id - i], mx - i) : 1);
12        for (; st[i - p[i]] == st[i + p[i]]; p[i]++);
13        if (mx < i + p[i]) {
14            mx = i + p[i];
15            id = i;
16        }
17        ans = max(ans, p[i] - 1);
18    }
19    return ans;
20 }

```

5.2 Trie

```

1 const int MAXL = ;
2 const int MAXC = ;
3 struct Trie {
4     int nex[MAXL][MAXC];
5     int len[MAXL];
6     int sz;
7     void init() {
8         memset(nex, 0, sizeof(nex));
9     }
10 };

```



```

9   memset(len, 0, sizeof(len));
10  sz = 0;
11  }
12  void insert(const string &str) {
13      int p = 0;
14      for (char c : str) {
15          int id = c - 'a';
16          if (!nex[p][id]) {
17              nex[p][id] = ++sz;
18          }
19          p = nex[p][id];
20      }
21      len[p] = str.length();
22  }
23  vector<int> find(const string &str, int i) {
24      int p = 0;
25      vector<int> ans;
26      for (; i < str.length(); i++) {
27          int id = str[i] - 'a';
28          if (!nex[p][id]) {
29              return ans;
30          }
31          p = nex[p][id];
32          if (len[p]) {
33              ans.pb(len[p]);
34          }
35      }
36      return ans;
37  }
38 };

```

5.3 Z-value

```

1  // 0-base
2  // 對於個長度為 n 的字串 s
3  // 定義函數 z[i] 表示 s 和 s[i, n - 1]
4  // (即以 s[i] 開頭的后綴) 的最長公共前綴 (LCP) 的長度
5  // z[0] = 0。
6  vector<int> z_function(string s) {
7      int n = (int)s.length();
8      vector<int> z(n);
9      for (int i = 1, l = 0, r = 0; i < n; ++i) {
10         if (i <= r && z[i - l] < r - i + 1) {
11             z[i] = z[i - l];
12         } else {
13             z[i] = max(0, r - i + 1);
14             while (i + z[i] < n && s[z[i]] == s[i + z[i]])
15                 ++z[i];
16             if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
17         }
18         return z;
19     }

```

6 DP

6.1 LIS

```

1  int LIS(vector<int> &a) {
2      vector<int> s;
3      for (int i = 0; i < a.size(); i++) {
4          if (s.empty() || s.back() < a[i]) {
5              s.push_back(a[i]);
6          } else {
7              *lower_bound(s.begin(), s.end(), a[i],
8                  [](int x, int y) {return x < y;}) = a[i];
9          }
10     }
11     return s.size();
12 }

```

6.2 LCS

```

1  int LCS(string s1, string s2) {
2      int n1 = s1.size(), n2 = s2.size();
3      vector<vector<int>> dp(n1 + 1, vector<int>(n2 + 1,
4          0));
5      for (int i = 1; i <= n1; i++) {
6          for (int j = 1; j <= n2; j++) {
7              if (s1[i - 1] == s2[j - 1]) {
8                  dp[i][j] = dp[i - 1][j - 1] + 1;
9              } else {
10                 dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
11             }
12         }
13     }
14     return dp[n1][n2];
15 }

```

7 Math

7.1 Extended GCD

```

1  // ax + by = c
2  int extgcd(int a, int b, int c, int &x, int &y) {
3      if (b == 0) {
4          x = c / a;
5          y = 0;
6          return a;
7      }
8      int d = extgcd(b, a % b, c, y, x);
9      y -= (a / b) * x;
10     return d;
11 }

```

7.2 Gaussian Elimination + det

```

1  const double EPS = 1e-6;
2  double Gauss(vector<vector<double>> &d) {
3      int n = d.size(), m = d[0].size();
4      double det = 1;
5      for (int i = 0; i < m; i++) {
6          int p = -1;
7          for (int j = i; j < n; j++) {
8              if (fabs(d[j][i]) < EPS) {
9                  continue;
10             }
11             if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) {
12                 p = j;
13             }
14         }
15         if (p == -1) {
16             continue;
17         }
18         if (p != i) {
19             det *= -1;
20         }
21         for (int j = 0; j < m; j++) {
22             swap(d[p][j], d[i][j]);
23         }
24         for (int j = 0; j < n; j++) {
25             if (i == j) {
26                 continue;
27             }
28             double z = d[j][i] / d[i][i];
29             for (int k = 0; k < m; k++) {
30                 d[j][k] -= z * d[i][k];
31             }
32         }
33     }
34     for (int i = 0; i < n; i++) {
35         det *= d[i][i];
36     }
37 }

```

```

37 | return det;
38 | }

```

7.3 Prime Table

```

1 | vector<int> p;
2 | bitset<MAXN> is_notp;
3 | void PrimeTable(int n) {
4 |     is_notp.reset();
5 |     is_notp[0] = is_notp[1] = 1;
6 |     for (int i = 2; i <= n; ++i) {
7 |         if (!is_notp[i]) {
8 |             p.push_back(i);
9 |         }
10 |         for (int j = 0; j < (int)p.size(); ++j) {
11 |             if (i * p[j] > n) {
12 |                 break;
13 |             }
14 |             is_notp[i * p[j]] = 1;
15 |             if (i % p[j] == 0) {
16 |                 break;
17 |             }
18 |         }
19 |     }
20 | }

```

7.4 Phi

- 歐拉函數計算對於一個整數 N ，小於等於 N 的正整數中，有幾個和 N 互質
- 如果 $\gcd(p, q) = 1$, $\Phi(p) \cdot \Phi(q) = \Phi(p \cdot q)$
- $\Phi(p^k) = p^{k-1} \times (p - 1)$

```

1 | void phi_table(int n) {
2 |     phi[1] = 1;
3 |     for (int i = 2; i <= n; ++i) {
4 |         if (phi[i]) {
5 |             continue;
6 |         }
7 |         for (int j = i; j < n; j += i) {
8 |             if (!phi[j]) {
9 |                 phi[j] = j;
10 |             }
11 |             phi[j] = phi[j] / i * (i - 1);
12 |         }
13 |     }
14 | }

```

7.5 Chinese Remainder Thm

```

1 | //参数可为负数的扩展欧几里德定理
2 | void exOJLD(int a, int b, int& x, int& y) {
3 |     //根据欧几里德定理
4 |     if (b == 0) { //任意数与0的最大公约数为其本身。
5 |         x = 1;
6 |         y = 0;
7 |     } else {
8 |         int x1, y1;
9 |         exOJLD(b, a % b, x1, y1);
10 |         if (a * b < 0) { //异号取反
11 |             x = -x1;
12 |             y = a / b * y1 - x1;
13 |         } else { //同号
14 |             x = x1;
15 |             y = x1 - a / b * y1;
16 |         }
17 |     }
18 | }
19 | //剩余定理
20 | int calSYDL(int a[], int m[], int k) {

```

```

21 | int N[k]; //这个可以删除
22 | int mm = 1; //最小公倍数
23 | int result = 0;
24 | for (int i = 0; i < k; i++) {
25 |     mm *= m[i];
26 | }
27 | for (int j = 0; j < k; j++) {
28 |     int L, J;
29 |     exOJLD(mm / m[j], -m[j], L, J);
30 |     N[j] = m[j] * J + 1; // 1
31 |     N[j] = mm / m[j] * L; // 2
32 |     //1和2这两个值应该是相等的。
33 |     result += N[j] * a[j];
34 | }
35 | return (result % mm + mm) % mm;
36 | //落在(0, mm)之间，这么写是为了防止result初始为负数，本例中不可能为
37 | //写成：return result%mm;即可。
38 | }
39 | int main() {
40 |     int a[3] = {2, 3, 6}; // a[i]=n%m[i]
41 |     int m[3] = {3, 5, 7};
42 |     cout << calSYDL(a, m, 3) << endl;
43 |     //輸出為滿足兩條陣列的最小n,第3參數為陣列長度
44 |     //所有滿足答案的數字集合為n+gcd(m0,m1,m2...)*k,
45 |     //k為正數
46 |     return 0;
47 | }

```

7.6 Josephus

```

1 | int josephus(int n, int k) { //
2 |     //有n個人圍成一圈，每k個一次
3 |     return n > 1 ? (josephus(n - 1, k) + k) % n : 0;
4 | } // 回傳最後一人的編號，0 index

```

7.7 Catalan

$$C_0 = 1 \quad \text{and} \quad C_{n+1} = \frac{2(2n+1)}{n+2} C_n$$

```

1 | long long f[N] = {1}, i, t, p;
2 | int main() {
3 |     for (int i = 1; i <= 100; i++) {
4 |         f[i] = f[i - 1] * (4 * i - 2) % mod;
5 |         for (t = i + 1, p = mod - 2; p; t = (t * t) %
6 |             mod, p >>= 1LL) {
7 |             if (p & 1) {
8 |                 f[i] *= t;
9 |                 f[i] %= mod;
10 |             }
11 |         }
12 |     }

```

7.8 Matrix Multiplication

```

1 | struct Matrix {
2 |     int row, col;
3 |     vector<vector<int>>> v;
4 |     Matrix() : row(0), col(0) {}
5 |     Matrix(int r, int c) : row(r), col(c) {
6 |         v = vector<vector<int>>>(r, vector<int>(c, 0));
7 |     }
8 | };
9 | Matrix operator * (Matrix &a, Matrix &b) {
10 |     assert(a.col == b.row);
11 |     Matrix ret(a.row, b.col);
12 |     for (int i = 0; i < a.row; i++) {
13 |         for (int j = 0; j < b.col; j++) {

```

```

14     for (int k = 0; k < a.col; k++) {
15         ret.v[i][j] += a.v[i][k] * b.v[k][j];
16     }
17 }
18 }
19 return ret;
20 }
21 Matrix mPow(Matrix a, int n) {
22     assert(a.row == a.col);
23     Matrix ret(a.row, a.col);
24     ret.v[0][0] = ret.v[1][1] = 1;
25     while (n > 0) {
26         if (n & 1) {
27             ret = ret * a;
28         }
29         a = a * a;
30         n >>= 1;
31     }
32     return ret;
33 }

```

7.9 Fibonacci

$$f(n) = f(n-1) + f(n-2)$$

$$\begin{bmatrix} f(n) \\ f(n-1) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix}^{(n-1)} \begin{bmatrix} 1 \\ 0 \end{bmatrix}$$

$$O(\log n)$$

```

1 LL fib(int n) {
2     if (n <= 1) {
3         return n;
4     }
5     Matrix a(2, 2), b(2, 1);
6     a.v[0][0] = a.v[0][1] = a.v[1][0] = 1;
7     b.v[0][0] = 1;
8     auto t = mPow(a, n - 1);
9     t = t * b;
10    return t.v[0][0];
11 }

```

8 Geometry

8.1 Point

```

1 // notice point type!!!
2 using dvt = int;
3 const double EPS = 1e-6;
4 const double PI = acos(-1);
5
6 struct Pt {
7     dvt x;
8     dvt y;
9 };
10 bool operator < (const Pt &a, const Pt &b) {
11     return a.x == b.x ? a.y < b.y : a.x < b.x;
12 }
13 bool operator == (const Pt &a, const Pt &b) {
14     return a.x == b.x && a.y == b.y;
15 }
16 Pt operator + (const Pt &a, const Pt &b) {
17     return {a.x + b.x, a.y + b.y};
18 }
19 Pt operator - (const Pt &a, const Pt &b) {
20     return {a.x - b.x, a.y - b.y};
21 }
22 // multiply constant
23 Pt operator * (const Pt &a, const dvt c) {
24     return {a.x * c, a.y * c};
25 }
26 Pt operator / (const Pt &a, const dvt c) {
27     return {a.x / c, a.y / c};

```

```

28 }
29 // |a| x |b| x cos(x)
30 dvt iproduct(const Pt &a, const Pt &b) {
31     return a.x * b.x + a.y * b.y;
32 }
33 // |a| x |b| x sin(x)
34 dvt cross(const Pt &a, const Pt &b) {
35     return a.x * b.y - a.y * b.x;
36 }
37 dvt dis_pp(const Pt &a, const Pt &b) {
38     dvt dx = a.x - b.x;
39     dvt dy = a.y - b.y;
40     return sqrt(dx * dx + dy * dy);
41 }

```

8.2 Line

$$d(P, L) = \frac{|ax_0 + by_0 + c|}{\sqrt{a^2 + b^2}}$$

```

1 struct Line {
2     Pt st;
3     Pt ed;
4 };
5 // return point side
6 // left, on line, right -> 1, 0, -1
7 int side(Line l, Pt a) {
8     dvt cross_val = cross(a - l.st, l.ed - l.st);
9     if (cross_val > EPS) {
10         return 1;
11     } else if (cross_val < -EPS) {
12         return -1;
13     } else {
14         return 0;
15     }
16 }
17 // AB infinity, CD segment
18 bool has_intersection(Line AB, Line CD) {
19     int c = side(AB, CD.st);
20     int d = side(AB, CD.ed);
21     if (c == 0 || d == 0) {
22         return true;
23     } else {
24         // different side
25         return c == -d;
26     }
27 }
28 // find intersection point, two line, not seg
29 pair<int, Pt> intersection(Line a, Line b) {
30     Pt A = a.ed - a.st;
31     Pt B = b.ed - b.st;
32     Pt C = b.st - a.st;
33     dvt mom = cross(A, B);
34     dvt son = cross(C, B);
35     if (std::abs(mom) <= EPS) {
36         if (std::abs(son) <= EPS) {
37             return {1, {}}; // same line
38         } else {
39             return {2, {}}; // parallel
40         }
41     } else {
42         // ok
43         return {0, a.st + A * (son / mom)};
44     }
45 }
46 // line to point distance
47 dvt dis_lp(Line l, Pt a) {
48     return area3x2(l.st, l.ed, a) / dis_pp(l.st, l.ed);

```

8.3 Area

```

1 // triangle
2 dvt area3(Pt a, Pt b, Pt c) {
3     return std::abs(cross(b - a, c - a) / 2);

```

```

4 }
5 dvt area3x2(Pt a, Pt b, Pt c) { // for integer
6     return std::abs(cross(b - a, c - a));
7 }
8 // simple convex area(can in)
9 dvt area(vector<Pt> &a) {
10     dvt ret = 0;
11     for (int i = 0, sz = a.size(); i < sz; i++) {
12         ret += cross(a[i], a[(i + 1) % sz]);
13     }
14     return std::abs(ret) / 2;
15 }
16 // check point in/out a convex
17 int io_convex(vector<Pt> convex, Pt q) {
18     // convex is Counterclockwise
19     for (int i = 0, sz = convex.size(); i < sz; i++) {
20         Pt cur = convex[i] - q;
21         Pt nex = convex[(i + 1) % sz] - q;
22         dvt cross_val = cross(cur, nex);
23         if (std::abs(cross_val) <= EPS) {
24             return 0; // on edge
25         }
26         if (cross_val < 0) {
27             return -1; // outside
28         }
29     }
30     return 1; // inside
31 }

```

8.4 Convex Hull

```

1 vector<Pt> convex_hull(vector<Pt> &a) {
2     sort(a.begin(), a.end());
3     a.erase(unique(a.begin(), a.end()), a.end());
4     int sz = a.size(), m = 0;
5     vector<Pt> ret(sz + 5); // safe 1 up
6     for (int i = 0; i < sz; i++) {
7         while (m > 1 &&
8             cross(ret[m - 1] - ret[m - 2], a[i] - ret[m - 2]) <= EPS) {
9             m--;
10        }
11        ret[m++] = a[i];
12    }
13    int k = m;
14    for (int i = sz - 2; i >= 0; i--) {
15        while (m > k &&
16            cross(ret[m - 1] - ret[m - 2], a[i] - ret[m - 2]) <= EPS) {
17            m--;
18        }
19        ret[m++] = a[i];
20    }
21    if (sz > 1) {
22        m--;
23    }
24    ret.resize(m);
25    return ret;
26 }

```