

## Contents

<b>1 Basic</b>	
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
<b>2 Data Structure</b>	
2.1 Disjoint Set . . . . .	2
2.2 BIT RARSQ . . . . .	2
2.3 zkw RMQ . . . . .	3
2.4 Segment Tree RARMQ . . . . .	3
2.5 Treap . . . . .	3
<b>3 Graph</b>	
3.1 Dijkstra . . . . .	4
3.2 SPFA(negative cycle) . . . . .	4
3.3 Floyd Warshall . . . . .	4
3.4 Topological Sort . . . . .	5
3.5 Tree Diameter . . . . .	5
3.6 Directed MST . . . . .	5
3.7 LCA . . . . .	6
3.8 Euler Circuit . . . . .	6
<b>4 Connectivity</b>	
4.1 Kosaraju SCC . . . . .	7
4.2 BCC . . . . .	8
4.3 Articulation Point . . . . .	8
4.4 Bridges . . . . .	9
<b>5 Flow &amp; Matching</b>	
5.1 Relation . . . . .	9
5.2 Bipartite Matching . . . . .	9
5.3 KM . . . . .	10
5.4 Dinic . . . . .	10
5.5 MCMF . . . . .	11
5.6 Stable Matching . . . . .	11
5.7 Max General Graph Matching . . . . .	12
<b>6 String</b>	
6.1 Manacher . . . . .	12
6.2 Trie . . . . .	12
6.3 Z-value . . . . .	13
<b>7 DP</b>	
7.1 LIS . . . . .	13
7.2 LCS . . . . .	13
7.3 Huge Knapsack . . . . .	14
7.4 Coin Change . . . . .	14
7.5 Edit Distance . . . . .	14
<b>8 Math</b>	
8.1 Number Theory . . . . .	14
8.2 Extended GCD . . . . .	14
8.3 Gaussian Elimination + det . . . . .	15
8.4 Prime Table . . . . .	15
8.5 Phi . . . . .	15
8.6 Chinese Remainder Thm . . . . .	15
8.7 Josephus . . . . .	16
8.8 Catalan . . . . .	16
8.9 Matrix Multiplication . . . . .	16
8.10 Fibonacci . . . . .	16
<b>9 Geometry</b>	
9.1 Point . . . . .	16
9.2 Line . . . . .	17
9.3 Area . . . . .	17
9.4 Convex Hull . . . . .	17

## 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

```

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

```

```

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 }

```

## 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() {

```

## 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
        vector<int> &o) {

```

```

30     T = new Node();
31     if (l == r) {
32         T->val = o[l];
33         return;
34     }
35     int mid = (l + r) / 2;
36     build(T->lc, l, mid, o);
37     build(T->rc, mid + 1, r, o);
38     T->pull();
39 }
40 void update(Node* &T, int l, int r, int ql, int qr,
    int v) {
41     if (ql <= l && r <= qr) {
42         T->val += v;
43         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, qr));
71     }
72 }
73 };

```

### 2.5 Treap

```

1 struct Treap {
2     int val, pri, sz;
3     Treap *lc, *rc;
4     Treap() {}
5     Treap(int _val) {
6         val = _val;
7         pri = rand();
8         sz = 1;
9         lc = rc = NULL;
10    }
11 };
12
13 int getSize(Treap *a) { return (a == NULL ? 0 :
    a->sz); }
14
15 void split(Treap *t, Treap *&a, Treap *&b, int k) {
16     if (t == NULL) {
17         a = b = NULL;
18         return;
19     }
20     if (getSize(t->lc) < k) {
21         a = t;
22         split(t->rc, a->rc, b, k - getSize(t->lc) - 1);
23     } else {
24         b = t;
25         split(t->lc, a, b->lc, k);
26     }
27 }

```

```

28
29 Treap *merge(Treap *a, Treap *b) {
30     if (!a || !b) {
31         return (a ? a : b);
32     }
33     if (a->pri > b->pri) {
34         a->rc = merge(a->rc, b);
35         return a;
36     } else {
37         b->lc = merge(a, b->lc);
38         return b;
39     }
40 }
41
42 void Insert(Treap *t, int x, int p) {
43     Treap *a, *b;
44     split(t, a, b, x);
45     t = merge(a, merge(new Treap(p), b));
46 }
47
48 void Delete(Treap *t, int x) {
49     Treap *a, *b, *c;
50     split(t, b, c, x);
51     split(b, a, b, x - 1);
52     t = merge(a, c);
53 }
54
55 /*
56 Usage
57 Treap *root = NULL; // declare
58 root = merge(root, new Treap(val)); // push back
59 Insert(root, x, y); // insert y after x-th element
60 Delete(root, x); // delete x-th element
61 */

```

## 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
24 void Dijkstra(int st, int ed = -1) {
25     priority_queue<Edge> pq;
26     pq.emplace(st, 0);
27     dis[st] = 0;
28     while (!pq.empty()) {
29         auto now = pq.top();
30         pq.pop();
31         if (now.to == ed) {
32             return;
33         }
34         if (now.cost > dis[now.to]) {
35             continue;
36         }

```

```

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
21 bool SPFA(int st) {
22     vector<int> cnt(n, 0);
23     vector<bool> inq(n, false);
24     queue<int> q;
25
26     q.push(st);
27     dis[st] = 0;
28     inq[st] = true;
29     while (!q.empty()) {
30         int now = q.front();
31         q.pop();
32         inq[now] = false;
33         for (auto &e : G[now]) {
34             if (dis[e.to] > dis[now] + e.cost) {
35                 dis[e.to] = dis[now] + e.cost;
36                 if (!inq[e.to]) {
37                     cnt[e.to]++;
38                     if (cnt[e.to] > n) {
39                         // negative cycle
40                         return false;
41                     }
42                     inq[e.to] = true;
43                     q.push(e.to);
44                 }
45             }
46         }
47     }
48     return true;
49 }

```

### 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 }
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     } 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
13 struct DMST {
14     int n;
15     int vis[MAXN], pre[MAXN], id[MAXN];
16     LL in[MAXN];
17     vector<Edge> edges;
18     void init(int _n) {
19         n = _n;
20         edges.clear();
21     }
22     void add_edge(int from, int to, LL cost) {
23         edges.eb(from, to, cost);
24     }
25     LL run(int root) {
26         LL ret = 0;
27         while (true) {
28             for (int i = 0; i < n; i++) {
29                 in[i] = INF;
30             }
31
32             // find in edge
33             for (auto &e : edges) {
34                 if (e.cost < in[e.to] && e.from != e.to) {
35                     pre[e.to] = e.from;
36                     in[e.to] = e.cost;
37                 }
38             }
39
40             // check in edge
41             for (int i = 0; i < n; i++) {
42                 if (i == root) {
43                     continue;
44                 }
45                 if (in[i] == INF) {
46                     return -1;
47                 }
48             }
49
50             int nodenum = 0;
51             memset(id, -1, sizeof(id));
52             memset(vis, -1, sizeof(vis));
53             in[root] = 0;
54
55             // find cycles
56             for (int i = 0; i < n; i++) {
57                 ret += in[i];
58                 int v = i;
59                 while (vis[v] != i && id[v] == -1 && v !=
60                     root) {
61                     vis[v] = i;
62                     v = pre[v];
63                 }
64             }
65         }
66     }
67 };

```

```

62     if (id[v] == -1 && v != root) {
63         for (int j = pre[v]; j != v; j = pre[j]) {
64             id[j] = nodenum;
65         }
66         id[v] = nodenum++;
67     }
68 }
69
70 // no cycle
71 if (nodenum == 0) {
72     break;
73 }
74
75 for (int i = 0; i < n; i++) {
76     if (id[i] == -1) {
77         id[i] = nodenum++;
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 = nodenum;
92 root = id[root];
93 }
94 return ret;
95 }
96 };

```

### 3.7 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 */

```

### 3.8 Euler Circuit

七橋問題根據起點與終點是否相同，分成 Euler path (不同) 及 Euler circuit (相同)。

- 判斷法
  - 無向圖部分，將點分成奇點 (度數為奇數) 和偶點 (度數為偶數)。
    - Euler path: 奇點數為 0 或 2
    - Euler circuit: 沒有奇點
  - 有向圖部分，將點分成出點 (出度 - 入度 = 1) 和入點 (入度 - 出度 = 1) 還有平衡點 (出度 = 入度)。
    - Euler path: 出點和入點個數同時為 0 或 1。
    - Euler circuit: 只有平衡點。
- 求出一組解
- 用 DFS 遍歷整張圖，設  $s$  為離開的順序，無向圖的答案為  $s$ ，有向圖的答案為反向的  $s$ 。
- DFS 起點選定：
  - Euler path: 無向圖選擇任意一個奇點，有向圖選擇出點。
  - Euler circuit: 任意一點。

```

1  // Code from Eric
2  #define ll long long
3  #define PB push_back
4  #define EB emplace_back
5  #define PII pair<int, int>
6  #define MP make_pair
7  #define all(x) x.begin(), x.end()
8  #define maxn 50000+5
9
10 //structure
11 struct Euler {
12     vector<PII> adj[maxn];
13     vector<bool> edges;
14     vector<PII> path;
15     int chk[maxn];
16     int n;
17
18     void init(int _n) {
19         n = _n;
20         for (int i = 0; i <= n; i++) adj[i].clear();
21         edges.clear();
22         path.clear();
23         memset(chk, 0, sizeof(chk));
24     }
25
26     void dfs(int v) {
27         for (auto i : adj[v]) {
28             if (edges[i.first] == true) {
29                 edges[i.first] = false;
30                 dfs(i.second);
31                 path.EB(MP(i.second, v));
32             }
33         }
34     }
35
36     void add_Edge(int from, int to) {

```

```

37 edges.PB(true);
38
39 // for bi-directed graph
40 adj[from].PB(MP(edges.size() - 1, to));
41 adj[to].PB(MP(edges.size() - 1, from));
42 chk[from]++;
43 chk[to]++;
44
45 // for directed graph
46 // adj[from].PB(MP(edges.size()-1, to));
47 // check[from]++;
48 }
49
50 bool eular_path() {
51     int st = -1;
52     for (int i = 1; i <= n; i++) {
53         if (chk[i] % 2 == 1) {
54             st = i;
55             break;
56         }
57     }
58     if (st == -1) {
59         return false;
60     }
61     dfs(st);
62     return true;
63 }
64
65 void print_path(void) {
66     for (auto i : path) {
67         printf("%d %d\n", i.first, i.second);
68     }
69 }
70 };
71
72 // Code from allen(lexicographic order)
73 #include <bits/stdc++.h>
74 using namespace std;
75 const int ALP = 30;
76 const int MXN = 1005;
77 int n;
78 int din[ALP], dout[ALP];
79 int par[ALP];
80 vector<string> vs[MXN], ans;
81 bitset<MXN> vis, used[ALP];
82
83 void djsInit() {
84     for (int i = 0; i != ALP; ++i) {
85         par[i] = i;
86     }
87 }
88
89 int Find(int x) { return (x == par[x] ? (x) : (par[x] = Find(par[x]))); }
90
91 void init() {
92     djsInit();
93     memset(din, 0, sizeof(din));
94     memset(dout, 0, sizeof(dout));
95     vis.reset();
96     for (int i = 0; i != ALP; ++i) {
97         vs[i].clear();
98         used[i].reset();
99     }
100     return;
101 }
102
103 void dfs(int u) {
104     for (int i = 0; i != (int)vs[u].size(); ++i) {
105         if (used[u][i]) {
106             continue;
107         }
108         used[u][i] = 1;
109         string s = vs[u][i];
110         int v = s[s.size() - 1] - 'a';
111         dfs(v);
112         ans.push_back(s);
113     }
114 }

```

```

43 }
44
45 bool solve() {
46     int cnt = 1;
47     for (int i = 0; i != n; ++i) {
48         string s;
49         cin >> s;
50         int from = s[0] - 'a', to = s.back() - 'a';
51         ++din[to];
52         ++dout[from];
53         vs[from].push_back(s);
54         vis[from] = vis[to] = true;
55         if ((from = Find(from)) != (to = Find(to))) {
56             par[from] = to;
57             ++cnt;
58         }
59     }
60     if ((int)vis.count() != cnt) {
61         return false;
62     }
63     int root, st, pin = 0, pout = 0;
64     for (int i = ALP - 1; i >= 0; --i) {
65         sort(vs[i].begin(), vs[i].end());
66         if (vs[i].size()) root = i;
67         int d = dout[i] - din[i];
68         if (d == 1) {
69             ++pout;
70             st = i;
71         } else if (d == -1) {
72             ++pin;
73         } else if (d != 0) {
74             return false;
75         }
76     }
77     if (pin != pout || pin > 1) {
78         return false;
79     }
80     ans.clear();
81     dfs((pin ? st : root));
82     return true;
83 }
84
85 int main() {
86     int t;
87     cin >> t;
88     while (t--) {
89         cin >> n;
90         init();
91         if (!solve()) {
92             cout << "***\n";
93             continue;
94         }
95         for (int i = ans.size() - 1; i >= 0; --i) {
96             cout << ans[i] << ".\n"[i == 0];
97         }
98     }
99 }

```

## 4 Connectivity

### 4.1 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     }

```

```

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 }

```

## 4.2 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 }

```

## 4.3 Articulation Point

```

1 // from aizu
2 typedef long long int ll;
3 typedef unsigned long long int ull;
4 #define BIG_SIZE 2000000000
5 #define MOD 1000000007
6 #define EPS 0.000000001
7 using namespace std;
8
9 #define SIZE 100000
10
11 vector<int> G[SIZE];
12 int N;
13 bool visited[SIZE];
14 int visited_order[SIZE], parent[SIZE], lowest[SIZE],
15     number;
16
17 void dfs(int cur, int pre_node) {
18     visited_order[cur] = lowest[cur] = number;
19     number++;
20
21     visited[cur] = true;
22
23     int next;
24     for (int i = 0; i < G[cur].size(); i++) {
25         next = G[cur][i];
26         if (!visited[next]) {
27             parent[next] = cur;
28             dfs(next, cur);
29             lowest[cur] = min(lowest[cur], lowest[next]);
30         } else if (visited[next] == true && next !=
31             pre_node) {
32             lowest[cur] = min(lowest[cur],
33                 visited_order[next]);
34         }
35     }
36 }
37
38 void art_points() {
39     for (int i = 0; i < N; i++) visited[i] = false;
40
41     number = 1;
42     dfs(0, -1);
43
44     int tmp_parent, root_num = 0;
45
46     vector<int> V;
47
48     for (int i = 1; i < N; i++) {
49         tmp_parent = parent[i];
50         if (tmp_parent == 0) {
51             root_num++;
52         } else if (visited_order[tmp_parent] <=
53             lowest[i]) {
54             V.push_back(tmp_parent);
55         }
56     }
57     if (root_num >= 2) {
58         V.push_back(0);
59     }
60     sort(V.begin(), V.end());
61     V.erase(unique(V.begin(), V.end()), V.end());
62
63     for (int i = 0; i < V.size(); i++) {
64         printf("%d\n", V[i]);
65     }
66 }

```



```

65 int main() {
66     int E;
67     scanf("%d %d", &N, &E);
68     int from, to;
69     for (int i = 0; i < E; i++) {
70         scanf("%d %d", &from, &to);
71         G[from].push_back(to);
72         G[to].push_back(from);
73     }
74     art_points();
75 }

```

## 4.4 Bridges

```

1 // from aizu
2 typedef long long int ll;
3 typedef unsigned long long int ull;
4 #define BIG_NUM 2000000000
5 #define MOD 1000000007
6 #define EPS 0.000000001
7 using namespace std;
8
9 struct Edge {
10     bool operator<(const struct Edge &arg) const {
11         if (s != arg.s) {
12             return s < arg.s;
13         } else {
14             return t < arg.t;
15         }
16     }
17     int s, t;
18 };
19 struct Info {
20     Info(int arg_to, int arg_edge_id) {
21         to = arg_to;
22         edge_id = arg_edge_id;
23     }
24     int to, edge_id;
25 };
26
27 int V, E, number;
28 int order[100000], lowlink[100000];
29 bool visited[100000];
30 Edge edge[100000];
31 vector<Info> G[100000];
32
33 void recursive(int cur) {
34     order[cur] = number++;
35     lowlink[cur] = order[cur];
36
37     int next;
38
39     for (int i = 0; i < G[cur].size(); i++) {
40         next = G[cur][i].to;
41
42         if (order[next] == -1) {
43             visited[G[cur][i].edge_id] = true;
44             recursive(next);
45             lowlink[cur] = min(lowlink[cur], lowlink[next]);
46         } else if (visited[G[cur][i].edge_id] == false) {
47             lowlink[cur] = min(lowlink[cur], order[next]);
48         }
49     }
50 }
51
52 int main() {
53     scanf("%d %d", &V, &E);
54     for (int i = 0; i < E; i++) {
55         scanf("%d %d", &edge[i].s, &edge[i].t);
56         if (edge[i].s > edge[i].t) {
57             swap(edge[i].s, edge[i].t);
58         }
59         G[edge[i].s].push_back(Info(edge[i].t, i));
60         G[edge[i].t].push_back(Info(edge[i].s, i));
61     }
62 }

```

```

63     sort(edge, edge + E);
64
65     number = 0;
66     for (int i = 0; i < V; i++) {
67         order[i] = -1;
68         lowlink[i] = -1;
69     }
70     for (int i = 0; i < E; i++) {
71         visited[i] = false;
72     }
73
74     recursive(0);
75
76     int from, to;
77     for (int i = 0; i < E; i++) {
78         from = edge[i].s;
79         to = edge[i].t;
80         if (order[edge[i].s] > order[edge[i].t]) {
81             swap(from, to);
82         }
83         if (order[from] < lowlink[to]) {
84             printf("%d %d\n", edge[i].s, edge[i].t);
85         }
86     }
87     return 0;
88 }
89 }

```

## 5 Flow & Matching

### 5.1 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|

```

### 5.2 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 }

```

```

28 }
29 return cnt;
30 }

```

### 5.3 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         }
67         int ans = 0;
68         for (int i = 1; i <= n; i++) {
69             ans += lx[i] + ly[i];

```

```

70 }
71 return ans;
72 }
73 };

```

### 5.4 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 };

```

## 5.5 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.eb(u, v, c, cap);
28         G[v].pb(edges.size());
29         edges.eb(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                         inq[v] = true;
59                     }
60                     if (cnt[v] == n + 2) {
61                         negcycle = true;
62                         break;
63                     }
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
81     LL f = sendFlow(u, min(curFlow, edges[i].cap));
82
83     ans_cost += f * cost;
84     edges[i].cap -= f;
85     edges[i ^ 1].cap += f;
86     return f;
87 }
88 pair<LL, LL> run(int st, int ed) {
89     ans_flow = ans_cost = 0;
90     while (SPFA(st, ed)) {
91         ans_flow += sendFlow(ed, INF);
92     }
93     return make_pair(ans_flow, ans_cost);
94 }
95 };

```

## 5.6 Stable Matching

```

1 int t, n, b[N][N], bi[N], g[N][N], bg[N], gb[N];
2
3 void sol() {
4     deque<int> dq;
5     memset(gb, 0, sizeof(gb));
6     memset(bi, 0, sizeof(bi));
7     for (int i = 1; i <= n; i++) dq.push_back(i);
8     while (!dq.empty()) {
9         int x = dq.front();
10        dq.pop_front();
11        int y = b[x][++bi[x]];
12        if (!gb[y]) {
13            gb[y] = x;
14            bg[x] = y;
15        } else if (g[y][x] < g[y][gb[y]]) {
16            dq.push_back(gb[y]);
17            gb[y] = x;
18            bg[x] = y;
19        } else {
20            dq.push_back(x);
21        }
22    }
23    for (int i = 1; i <= n; i++) {
24        cout << bg[i] << '\n';
25    }
26 }
27
28 int main() {
29     int x;
30     cin >> t;
31     for (int i = 0; i < t; i++) {
32         cin >> n;
33         for (int i = 1; i <= n; i++) {
34             for (int j = 1; j <= n; j++) {
35                 cin >> b[i][j];
36             }
37         }
38         for (int i = 1; i <= n; i++) {
39             for (int j = 1; j <= n; j++) {
40                 cin >> x;
41                 g[i][x] = j;
42             }
43         }
44         if (i) cout << '\n';

```

```

45     sol();
46 }
47 }

```

## 5.7 Max General Graph Matching

```

1 #define maxn
2
3 int fa[maxn], pre[maxn], match[maxn], s[maxn],
  v[maxn];
4 vector<int> g[maxn];
5 queue<int> q;
6 void Init(int n) {
7     for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
8     for (int i = 0; i < n; ++i) g[i].clear();
9 }
10 void AddEdge(int u, int v) {
11     g[u].push_back(v);
12     g[v].push_back(u);
13 }
14 int Find(int u) { return u == fa[u] ? u : fa[u] =
  Find(fa[u]); }
15 int LCA(int x, int y, int n) {
16     static int tk = 0;
17     tk++;
18     x = Find(x), y = Find(y);
19     for (;;) swap(x, y) {
20         if (x != n) {
21             if (v[x] == tk) return x;
22             v[x] = tk;
23             x = Find(pre[match[x]]);
24         }
25     }
26 }
27
28 void Blossom(int x, int y, int l) {
29     while (Find(x) != l) {
30         pre[x] = y, y = match[x];
31         if (s[y] == 1) q.push(y), s[y] = 0;
32         if (fa[x] == x) fa[x] = 1;
33         if (fa[y] == y) fa[y] = 1;
34         x = pre[y];
35     }
36 }
37
38 bool Bfs(int r, int n) {
39     for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
40     while (!q.empty()) q.pop();
41     q.push(r);
42     s[r] = 0;
43     while (!q.empty()) {
44         int x = q.front();
45         q.pop();
46         for (int u : g[x]) {
47             if (s[u] == -1) {
48                 pre[u] = x, s[u] = 1;
49                 if (match[u] == n) {
50                     for (int a = u, b = x, last; b != n; a =
  last, b = pre[a])
51                         last = match[b], match[b] = a, match[a] =
  b;
52                     return true;
53                 }
54                 q.push(match[u]);
55                 s[match[u]] = 0;
56             } else if (!s[u] && Find(u) != Find(x)) {
57                 int l = LCA(u, x, n);
58                 Blossom(x, u, l);
59                 Blossom(u, x, l);
60             }
61         }
62     }
63     return false;
64 }
65
66 int Solve(int n) {

```

```

67     int res = 0;
68     for (int x = 0; x < n; ++x) {
69         if (match[x] == n) res += Bfs(x, n);
70     }
71     return res;
72 }

```

## 6 String

### 6.1 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 }

```

### 6.2 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         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 };

```

## 6.3 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     }
19     return z;
20 }

```

## 7 DP

### 7.1 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 }

```

### 7.2 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 }

```

### 7.3 Huge Knapsack

```

1 // from aizun
2 #include <bits/stdc++.h>
3 typedef long long int ll;
4 typedef unsigned long long int ull;
5 #define BIG_NUM 2000000000
6 #define HUGE_NUM 9999999999999999
7 #define MOD 1000000007
8 #define EPS 0.000000001
9 using namespace std;

```

```

10
11 #define SIZE 25
12
13 struct Info {
14     Info() { value = 0, weight = 0; }
15     Info(ll arg_value, ll arg_weight) {
16         value = arg_value;
17         weight = arg_weight;
18     }
19     bool operator<(const struct Info &arg) const {
20         return weight < arg.weight; }
21     ll value, weight;
22 };
23
24 ll N, W;
25 ll POW[SIZE];
26 ll table_B[1 << 21];
27 Info info[45];
28
29 int main() {
30     POW[0] = 1;
31     for (int i = 1; i < SIZE; i++) {
32         POW[i] = POW[i - 1] * 2;
33     }
34
35     scanf("%lld %lld", &N, &W);
36
37     for (int i = 0; i < N; i++) {
38         scanf("%lld %lld", &info[i].value,
39             &info[i].weight);
40     }
41
42     if (N == 1) {
43         if (info[0].weight <= W) {
44             printf("%lld\n", info[0].value);
45         } else {
46             printf("0\n");
47         }
48     }
49     return 0;
50 }
51
52 vector<int> A, B;
53 for (int i = 0; i < N / 2; i++) {
54     A.push_back(i);
55 }
56 for (int i = N / 2; i < N; i++) {
57     B.push_back(i);
58 }
59
60 vector<Info> vec_A, vec_B;
61 for (int state = 0; state < POW[A.size()]; state++) {
62     {
63         ll sum_w = 0;
64         ll sum_value = 0;
65         for (int loop = 0; loop < A.size(); loop++) {
66             if (state & POW[loop]) {
67                 sum_w += info[A[loop]].weight;
68                 sum_value += info[A[loop]].value;
69             }
70         }
71         vec_A.push_back(Info(sum_value, sum_w));
72     }
73 }
74 sort(vec_A.begin(), vec_A.end());
75
76 for (int state = 0; state < POW[B.size()]; state++) {
77     {
78         ll sum_w = 0;
79         ll sum_value = 0;
80         for (int loop = 0; loop < B.size(); loop++) {
81             if (state & POW[loop]) {
82                 sum_w += info[B[loop]].weight;
83                 sum_value += info[B[loop]].value;
84             }
85         }
86         vec_B.push_back(Info(sum_value, sum_w));
87     }
88 }

```

```

83 }
84 sort(vec_B.begin(), vec_B.end());
85
86 table_B[0] = vec_B[0].value;
87 for (int i = 1; i < vec_B.size(); i++) {
88     //ある重さ以下の最大価値を求める
89     table_B[i] = max(table_B[i - 1], vec_B[i].value);
90 }
91
92 int tail = vec_B.size() - 1;
93 ll ans = 0;
94 for (int i = 0; i < vec_A.size(); i++) {
95     while (tail >= 0 && vec_A[i].weight +
96            vec_B[tail].weight > W) tail--;
97     if (tail < 0) break;
98     ans = max(ans, vec_A[i].value + table_B[tail]);
99 }
100
101 printf("%lld\n", ans);
102 return 0;
103 }

```

## 7.4 Coin Change

```

1 // from aizu
2 int main() {
3     int n, m, min, tmp;
4     scanf("%d", &n);
5     int minimum[n + 1];
6     scanf("%d", &m);
7     int coin[m];
8     for (int i = 0; i < m; i++) scanf("%d", &coin[i]);
9
10    minimum[0] = 0;
11    minimum[1] = 1;
12    for (int i = 2; i <= n; i++) {
13        min = n + 1;
14        for (int k = 0; k < m; k++) {
15            if (coin[k] <= i) {
16                tmp = 1 + minimum[i - coin[k]];
17                min = (min <= tmp) ? min : tmp;
18            }
19        }
20        minimum[i] = min;
21    }
22
23    printf("%d\n", minimum[n]);
24
25    return 0;
26 }

```

## 7.5 Edit Distance

```

1 // from aizu
2 typedef long long int ll;
3 typedef unsigned long long int ull;
4 #define BIG_NUM 2000000000
5 #define MOD 1000000007
6 #define EPS 0.000000001
7
8 int main() {
9     char A[1001], B[1001];
10    int len_A, len_B;
11    scanf("%s %s", A, B);
12
13    for (len_A = 0; A[len_A] != '\0'; len_A++);
14    for (len_B = 0; B[len_B] != '\0'; len_B++);
15
16    int** dp = new int*[len_B + 1];
17
18    for (int row = 0; row <= len_B; row++) {
19        dp[row] = new int[len_A + 1];

```

```

20 }
21
22 for (int col = 0; col <= len_A; col++) {
23     dp[0][col] = col;
24 }
25
26 for (int row = 1; row <= len_B; row++) {
27     dp[row][0] = row;
28 }
29
30 int cost;
31
32 for (int row = 1; row <= len_B; row++) {
33     for (int col = 1; col <= len_A; col++) {
34         if (A[col - 1] == B[row - 1]) {
35             cost = 0;
36         } else {
37             cost = 1;
38         }
39         dp[row][col] = min(dp[row - 1][col - 1] + cost,
40                           min(dp[row - 1][col] + 1, dp[row][col - 1] +
41                               1));
42     }
43 }
44
45 printf("%d\n", dp[len_B][len_A]);
46 }

```

## 8 Math

### 8.1 Number Theory

- Inversion:  
 $aa^{-1} \equiv 1 \pmod{m}$ .  $a^{-1}$  exists iff  $\gcd(a, m) = 1$ .
- Linear inversion:  
 $a^{-1} \equiv (m - \lfloor \frac{m}{a} \rfloor) \times (m \bmod a)^{-1} \pmod{m}$
- Fermat's little theorem:  
 $a^p \equiv a \pmod{p}$  if  $p$  is prime.
- Euler function:  
 $\phi(n) = n \prod_{p|n} \frac{p-1}{p}$
- Euler theorem:  
 $a^{\phi(n)} \equiv 1 \pmod{n}$  if  $\gcd(a, n) = 1$ .
- Extended Euclidean algorithm:  
 $ax + by = \gcd(a, b) = \gcd(b, a \bmod b) = \gcd(b, a - \lfloor \frac{a}{b} \rfloor b) = bx_1 + (a - \lfloor \frac{a}{b} \rfloor b)y_1 = ay_1 + b(x_1 - \lfloor \frac{a}{b} \rfloor y_1)$
- Divisor function:  
 $\sigma_x(n) = \sum_{d|n} d^x$ .  $n = \prod_{i=1}^r p_i^{a_i}$ .  
 $\sigma_x(n) = \prod_{i=1}^r \frac{p_i^{(a_i+1)x} - 1}{p_i^x - 1}$  if  $x \neq 0$ .  $\sigma_0(n) = \prod_{i=1}^r (a_i + 1)$ .
- Chinese remainder theorem:  
 $x \equiv a_i \pmod{m_i}$ .  
 $M = \prod m_i$ .  $M_i = M/m_i$ .  $t_i = M_i^{-1}$ .  
 $x = kM + \sum a_i t_i M_i$ ,  $k \in \mathbb{Z}$ .

### 8.2 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 }

```

### 8.3 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    return det;
38 }
39 // new
40 const int MAXN = 300;
41 const double EPS = 1e-8;
42 int n;
43 double A[MAXN][MAXN];
44 void Gauss() {
45     for (int i = 0; i < n; i++) {
46         bool ok = 0;
47         for (int j = i; j < n; j++) {
48             if (fabs(A[j][i]) > EPS) {
49                 swap(A[j], A[i]);
50                 ok = 1;
51                 break;
52             }
53         }
54         if (!ok) continue;
55         double fs = A[i][i];
56         for (int j = i + 1; j < n; j++) {
57             double r = A[j][i] / fs;
58             for (int k = i; k < n; k++) {
59                 A[j][k] -= A[i][k] * r;
60             }
61         }
62     }
63 }

```

### 8.4 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 }

```

### 8.5 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 }

```

### 8.6 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 = -y1;
12            y = a / b * y1 - x1;
13        } else { //同号
14            x = y1;
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 }

```

```

34 | return (result % mm + mm) % mm;
35 | //落在(0, mm)之間，這麼寫是為了防止result初始為負數
36 | //本例中不可能為負可以直接
37 | //寫成：return result%mm;即可。
38 | }
39 |
40 | int main() {
41 |     int a[3] = {2, 3, 6}; // a[i]=n%m[i]
42 |     int m[3] = {3, 5, 7};
43 |     cout << calSYDL(a, m, 3) << endl;
44 |     //輸出為滿足兩條陣列的最小n,第3參數為陣列長度
45 |     //所有滿足答案的數字集合為n+gcd(m0,m1,m2...)*k,
46 |     //k為正數
47 |     return 0;
48 | }

```

## 8.7 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

```

## 8.8 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 |     }

```

## 8.9 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 | }

```

## 8.10 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 | }

```

## 9 Geometry

### 9.1 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 | }

```



## 9.2 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 { // ok
42         return {0, a.st + A * (son / mom)};
43     }
44 }
45 // line to point distance
46 dvt dis_lp(Line l, Pt a) {
47     return area3x2(l.st, l.ed, a) / dis_pp(l.st, l.ed);
48 }

```

## 9.3 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 }

```

## 9.4 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 }

```