

## Contents

1	Basic	1
1.1	Run	1
1.2	Ternary Search	1
2	Data Structure	1
2.1	BIT RARSQ	1
2.2	zkw RMQ	1
2.3	Segment Tree RARMQ	1
2.4	Treap	1
3	Graph	2
3.1	Directed MST	2
3.2	LCA	3
3.3	Euler Circuit	3
4	Connectivity	5
4.1	Articulation Point	5
4.2	Bridges	5
5	Flow & Matching	6
5.1	Relation	6
5.2	Bipartite Matching	6
5.3	KM	6
5.4	Dinic	7
5.5	MCMF	7
6	String	8
6.1	Manacher	8
6.2	Trie	8
7	Math	8
7.1	Number Theory	8
7.2	Extended GCD	9
7.3	Gaussian Elimination	9
8	Geometry	9
8.1	Point	9
8.2	Line	9
8.3	Area	10
8.4	Convex Hull	10

## 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 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 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 | }
```

### 2.2 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 | }
```

### 2.3 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 };

```

## 2.4 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 int getSize(Treap *a) { return (a == NULL ? 0 :
    a->sz); }
13 void split(Treap *t, Treap *&a, Treap *&b, int k) {
14     if (t == NULL) {
15         a = b = NULL;
16         return;
17     }
18     if (getSize(t->lc) < k) {
19         a = t;
20         split(t->rc, a->rc, b, k - getSize(t->lc) - 1);
21     } else {
22         b = t;
23         split(t->lc, a, b->lc, k);
24     }
25 }
26 Treap *merge(Treap *a, Treap *b) {
27     if (!a || !b) {
28         return (a ? a : b);
29     }
30     if (a->pri > b->pri) {
31         a->rc = merge(a->rc, b);
32         return a;
33     } else {
34         b->lc = merge(a, b->lc);
35         return b;
36     }
37 }
38 void Insert(Treap *&t, int x, int p) {
39     Treap *a, *b;
40     split(t, a, b, x);
41     t = merge(a, merge(new Treap(p), b));
42 }
43 void Delete(Treap *&t, int x) {
44     Treap *a, *b, *c;
45     split(t, b, c, x);
46     split(b, a, b, x - 1);
47     t = merge(a, c);
48 }
49
50 /*
51 Usage
52 Treap *root = NULL; // declare
53 root = merge(root, new Treap(val)); // push back
54 Insert(root, x, y); // insert y after x-th element
55 Delete(root, x); // delete x-th element
56 */

```

## 3 Graph

### 3.1 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    {}

```

```

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 !=
59                     root) {
60                     vis[v] = i;
61                     v = pre[v];
62                 }
63                 if (id[v] == -1 && v != root) {
64                     for (int j = pre[v]; j != v; j = pre[j]) {
65                         id[j] = nodelist;
66                     }
67                     id[v] = nodelist++;
68                 }
69             }
70
71             // no cycle
72             if (nodelist == 0) {
73                 break;
74             }
75
76             for (int i = 0; i < n; i++) {
77                 if (id[i] == -1) {
78                     id[i] = nodelist++;
79                 }
80             }
81
82             // grouping the vertices
83             for (auto &e : edges) {
84                 int to = e.to;
85                 e.from = id[e.from];
86                 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 };

```

### 3.2 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] &&
28     tout[v] <= tout[u]; }
29
30 int LCA(int u, int v) {
31     if (depth[u] > depth[v]) {
32         swap(u, v);
33     }
34     if (anc(u, v)) {
35         return u;
36     }
37     for (int j = LOG - 1; j >= 0; --j) {
38         if (!anc(par[u][j], v)) u = par[u][j];
39     }
40     return par[u][0];
41 }
42
43 int dis(int u, int v) {
44     int lca = LCA(u, v);
45     return depth[u] + depth[v] - 2 * depth[lca];
46 }
47
48 /*
49 dfs(root, root);
50 Doubling(n);
51 */

```

### 3.3 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 }
115
116 bool solve() {
117     int cnt = 1;
118     for (int i = 0; i != n; ++i) {
119         string s;
120         cin >> s;
121         int from = s[0] - 'a', to = s.back() - 'a';
122         ++din[to];
123         ++dout[from];
124         vs[from].push_back(s);
125         vis[from] = vis[to] = true;
126         if ((from = Find(from)) != (to = Find(to))) {
127             par[from] = to;
128             ++cnt;
129         }
130     }
131     if ((int)vis.count() != cnt) {
132         return false;
133     }
134     int root, st, pin = 0, pout = 0;
135     for (int i = ALP - 1; i >= 0; --i) {
136         sort(vs[i].begin(), vs[i].end());
137         if (vs[i].size()) root = i;
138     }

```

```

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 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],
    number;
15
16 void dfs(int cur, int pre_node) {
17     visited_order[cur] = lowest[cur] = number;
18     number++;
19
20     visited[cur] = true;
21
22     int next;
23
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 !=
            pre_node) {
31             lowest[cur] = min(lowest[cur],
                visited_order[next]);
32         }
33     }

```

```

34 }
35
36 void art_points() {
37     for (int i = 0; i < N; i++) visited[i] = false;
38
39     number = 1;
40     dfs(0, -1);
41
42     int tmp_parent, root_num = 0;
43
44     vector<int> V;
45
46     for (int i = 1; i < N; i++) {
47         tmp_parent = parent[i];
48         if (tmp_parent == 0) {
49             root_num++;
50         } else if (visited_order[tmp_parent] <=
            lowest[i]) {
51             V.push_back(tmp_parent);
52         }
53     }
54     if (root_num >= 2) {
55         V.push_back(0);
56     }
57     sort(V.begin(), V.end());
58     V.erase(unique(V.begin(), V.end()), V.end());
59
60     for (int i = 0; i < V.size(); i++) {
61         printf("%d\n", V[i]);
62     }
63 }
64
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 }

```

### 4.2 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
53 int main() {
54     scanf("%d %d", &V, &E);
55     for (int i = 0; i < E; i++) {
56         scanf("%d %d", &edge[i].s, &edge[i].t);
57         if (edge[i].s > edge[i].t) {
58             swap(edge[i].s, edge[i].t);
59         }
60         G[edge[i].s].push_back(Info(edge[i].t, i));
61         G[edge[i].t].push_back(Info(edge[i].s, i));
62     }
63
64     sort(edge, edge + E);
65
66     number = 0;
67     for (int i = 0; i < V; i++) {
68         order[i] = -1;
69         lowlink[i] = -1;
70     }
71     for (int i = 0; i < E; i++) {
72         visited[i] = false;
73     }
74
75     recursive(0);
76
77     int from, to;
78     for (int i = 0; i < E; i++) {
79         from = edge[i].s;
80         to = edge[i].t;
81         if (order[edge[i].s] > order[edge[i].t]) {
82             swap(from, to);
83         }
84         if (order[from] < lowlink[to]) {
85             printf("%d %d\n", edge[i].s, edge[i].t);
86         }
87     }
88     return 0;
89 }

```

## 5 Flow & Matching

### 5.1 Relation

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|
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     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 queue<int> q;
38 bool negcycle = false;
39
40 dis[st] = 0;
41 cnt[st] = 1;
42 inq[st] = true;
43 q.push(st);
44
45 while (!q.empty() && !negcycle) {
46     int u = q.front();
47     q.pop();
48     inq[u] = false;
49     for (int i : G[u]) {
50         int v = edges[i].v;
51         LL cost = edges[i].cost;
52         LL cap = edges[i].cap;
53
54         if (dis[v] > dis[u] + cost && cap > 0) {
55             dis[v] = dis[u] + cost;
56             pre[v] = i;
57             if (!inq[v]) {
58                 q.push(v);
59                 cnt[v]++;
60                 inq[v] = true;
61
62                 if (cnt[v] == n + 2) {
63                     negcycle = true;
64                     break;
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 };

```

## 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 };

```

## 7 Math

### 7.1 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:  
 $x \equiv a_i \pmod{m_i}$ .  
 $M = \prod m_i$ .  $M_i = M/m_i$ .  $t_i = M_i^{-1}$ .  
 $x = kM + \sum a_i t_i M_i$ ,  $k \in \mathbb{Z}$ .

## 7.2 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 }

```

## 7.3 Gaussian Elimination

```

1 const int MAXN = 300;
2 const double EPS = 1e-8;
3 int n;
4 double A[MAXN][MAXN];
5 void Gauss() {
6     for (int i = 0; i < n; i++) {
7         bool ok = 0;
8         for (int j = i; j < n; j++) {
9             if (fabs(A[j][i]) > EPS) {
10                swap(A[j], A[i]);
11                ok = 1;
12                break;
13            }
14        }
15        if (!ok) continue;
16        double fs = A[i][i];
17        for (int j = i + 1; j < n; j++) {
18            double r = A[j][i] / fs;
19            for (int k = i; k < n; k++) {
20                A[j][k] -= A[i][k] * r;
21            }
22        }
23    }
24 }

```

# 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

```

```

46 dvt dis_lp(Line l, Pt a) {
47     return area3x2(l.st, l.ed, a) / dis_pp(l.st, l.ed);
48 }

```

### 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 }
32 © 2022 GitHub, Inc.

```

### 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 }

```