Contents

1 Basic

1.1 Run

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

1.2 Binary Search

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

1.3 Ternary Search

```
const double EPS = 1e-6;
// target function
double f(double x) { return x * x; }
double ternarySearch() {
  double L = -1e5, R = 1e5;
  while (R - L > EPS) {
    double mr = (L + R) / 2.0;
    double ml = (L + mr) / 2.0;
    if (f(ml) < f(mr)) {
      R = mr;
    } else {
      L = ml;
    }
  }
  return L;
}
```

2 Data Structure

2.1 BIT RARSQ

```
// 1-base
#define lowbit(k) (k & -k)
int n;
vector<int> B1, B2;
void add(vector<int> &tr, int id, int val) {
  for (; id <= n; id += lowbit(id)) {
    tr[id] += val;
  }
}
void range_add(int l, int r, int val) {
  add(B1, l, val);
  add(B1, r + 1, -val);
  add(B2, l, val * (l - 1));
  add(B2, r + 1, -val * r);
}
int sum(vector<int> &tr, int id) {
  int ret = 0;
  for (; id >= 1; id -= lowbit(id)) {
    ret += tr[id];
  }
  return ret;
}
int prefix_sum(int id) {
  return sum(B1, id) * id - sum(B2, id);
}
int range_sum(int l, int r) {
  return prefix_sum(r) - prefix_sum(l - 1);
}
```

2.2 zkw RMQ

```
// 0-base
const int INF = 1e9;
const int MAXN = ;
int n;
int a[MAXN], tr[MAXN << 1];
// !!! remember to call this function
void build() {
  for (int i = 0; i < n; i++) {
    tr[i + n] = a[i];
  }
  for (int i = n - 1; i > 0; i--) {
    tr[i] = max(tr[i << 1], tr[i << 1 | 1]);
  }
}
void update(int id, int val) {
  for (tr[id += n] = val; id > 1; id >>= 1) {
    tr[id >> 1] = max(tr[id], tr[id ^ 1]);
  }
}
int query(int l, int r) { // [l, r)
  int ret = -INF;
  for (l += n, r += n; l < r; l >>= 1, r >>= 1) {
    if (l & 1) {
      ret = max(ret, tr[l++]);
    }
    if (r & 1) {
      ret = max(ret, tr[--r]);
    }
  }
  return ret;
}
```

2.3 Segment Tree RARMQ

```cpp
struct Node {
  int val, tag;
  Node *lc, *rc;
  Node() : lc(nullptr), rc(nullptr), tag(0) {}
  void pull() {
    if (!lc) {
      val = rc->val;
    } else if (!rc) {
      val = lc->val;
    } else {
      val = max(lc->val, rc->val);
    }
  }
  void push() {
    if (lc) {
      lc->tag += tag;
      lc->val += tag;
    }
    if (rc) {
      rc->tag += tag;
      rc->val += tag;
    }
    tag = 0;
  }
};
struct SegmentTree {
  Node *root;
  SegmentTree() : root(nullptr) {}
  void build(Node* &T, int l, int r, const vector<int> &o) {
    T = new Node();
    if (l == r) {
      T->val = o[l];
      return;
    }
    int mid = (l + r) / 2;
    build(T->lc, l, mid, o);
    build(T->rc, mid + 1, r, o);
    T->pull();
  }
  void update(Node* &T, int l, int r, int ql, int qr, int v) {
    if (ql <= l && r <= qr) {
      T->val += v;
      T->tag += v;
      return;
    }
    T->push();
    int mid = (l + r) / 2;
    if (qr <= mid) {
      update(T->lc, l, mid, ql, qr, v);
    } else if (mid < ql) {
      update(T->rc, mid + 1, r, ql, qr, v);
    } else {
      update(T->lc, l, mid, ql, mid, v);
      update(T->rc, mid + 1, r, mid + 1, qr, v);
    }
    T->pull();
  }
  int query(Node* &T, int l, int r, int ql, int qr) {
    if (ql <= l && r <= qr) {
      return T->val;
    }
    T->push();
    int mid = (l + r) / 2;
    if (qr <= mid) {
      return query(T->lc, l, mid, ql, qr);
    } else if (mid < ql) {
      return query(T->rc, mid + 1, r, ql, qr);
    } else {
      return max(query(T->lc, l, mid, ql, mid),
          query(T->rc, mid + 1, r, mid + 1, qr));
    }
  }
};
```

2.4 Treap

```cpp
struct Treap {
  int val, pri, sz;
  Treap *lc, *rc;
  Treap() {}
  Treap(int _val) {
    val = _val;
    pri = rand();
    sz = 1;
    lc = rc = NULL;
  }
};
int getSize(Treap *a) { return (a == NULL ? 0 : a->sz); }
void split(Treap *t, Treap *&a, Treap *&b, int k) {
  if (t == NULL) {
    a = b = NULL;
    return;
  }
  if (getSize(t->lc) < k) {
    a = t;
    split(t->rc, a->rc, b, k - getSize(t->lc) - 1);
  } else {
    b = t;
    split(t->lc, a, b->lc, k);
  }
}
Treap *merge(Treap *a, Treap *b) {
  if (!a || !b) {
    return (a ? a : b);
  }
  if (a->pri > b->pri) {
    a->rc = merge(a->rc, b);
    return a;
  } else {
    b->lc = merge(a, b->lc);
    return b;
  }
}
void Insert(Treap *&t, int x, int p) {
  Treap *a, *b;
  split(t, a, b, x);
  t = merge(a, merge(new Treap(p), b));
}
void Delete(Treap *&t, int x) {
  Treap *a, *b, *c;
  split(t, b, c, x);
  split(b, a, b, x - 1);
  t = merge(a, c);
}

/*
Usage
Treap *root = NULL; // declare
root = merge(root, new Treap(val)); // push back
Insert(root, x, y); // insert y after x-th element
Delete(root, x); // delete x-th element
*/
```

3 Graph

3.1 Directed MST

```cpp
// 0-base
const LL INF = 1e18;
const int MAXN = ;

struct Edge {
  int from;
  int to;
  LL cost;
  Edge(int u, int v, LL c) : from(u), to(v), cost(c) {}
};

struct DMST {
  int n;
  int vis[MAXN], pre[MAXN], id[MAXN];
  LL in[MAXN];
  vector<Edge> edges;
  void init(int _n) {
    n = _n;
    edges.clear();
  }
  void add_edge(int from, int to, LL cost) {
    edges.eb(from, to, cost);
  }
  LL run(int root) {
    LL ret = 0;
    while (true) {
      for (int i = 0; i < n; i++) {
        in[i] = INF;
      }

      // find in edge
      for (auto &e : edges) {
        if (e.cost < in[e.to] && e.from != e.to) {
          pre[e.to] = e.from;
          in[e.to] = e.cost;
        }
      }

      // check in edge
      for (int i = 0; i < n; i++) {
        if (i == root) {
          continue;
        }
        if (in[i] == INF) {
          return -1;
        }
      }

      int nodenum = 0;
      memset(id, -1, sizeof(id));
      memset(vis, -1, sizeof(vis));
      in[root] = 0;

      // find cycles
      for (int i = 0; i < n; i++) {
        ret += in[i];
        int v = i;
        while (vis[v] != i && id[v] == -1 && v != root) {
          vis[v] = i;
          v = pre[v];
        }
        if (id[v] == -1 && v != root) {
          for (int j = pre[v]; j != v; j = pre[j]) {
            id[j] = nodenum;
          }
          id[v] = nodenum++;
        }
      }

      // no cycle
      if (nodenum == 0) {
        break;
      }

      for (int i = 0; i < n; i++) {
        if (id[i] == -1) {
          id[i] = nodenum++;
        }
      }

      // grouping the vertices
      for (auto &e : edges) {
        int to = e.to;
        e.from = id[e.from];
        e.to = id[e.to];
        if (e.from != e.to) {
          e.cost -= in[to]; //!!!
        }
      }

      n = nodenum;
      root = id[root];
    }
    return ret;
  }
};
```

3.2 LCA

```
const int LOG = 20;
vector<int> tin(MAXN), tout(MAXN), depth(MAXN);
int par[MAXN][LOG];
int timer = 0;
vector<int> G[MAXN];
void dfs(int u, int f) {
  tin[u] = ++timer;
  par[u][0] = f;
  for (int v : G[u]) {
    if (v != f) {
      depth[v] = depth[u] + 1;
      dfs(v, u);
    }
  }
  tout[u] = ++timer;
}
void Doubling(int n) {
  for (int j = 1; j < LOG; ++j) {
    for (int i = 1; i <= n; ++i) {
      par[i][j] = par[par[i][j - 1]][j - 1];
    }
  }
}
bool anc(int u, int v) { return tin[u] <= tin[v] && tout[v] <= tout[u]; }
int LCA(int u, int v) {
  if (depth[u] > depth[v]) {
    swap(u, v);
  }
  if (anc(u, v)) {
    return u;
  }
  for (int j = LOG - 1; j >= 0; --j) {
    if (!anc(par[u][j], v)) u = par[u][j];
  }
  return par[u][0];
}
int dis(int u, int v) {
  int lca = LCA(u, v);
  return depth[u] + depth[v] - 2 * depth[lca];
}

/*
dfs(root, root);
Doubling(n);
*/
```

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。

```cpp
// Code from Eric
#define ll long long
#define PB push_back
#define EB emplace_back
#define PII pair<int, int>
#define MP make_pair
#define all(x) x.begin(), x.end()
#define maxn 50000+5
//structure
struct Eular {
  vector<PII> adj[maxn];
  vector<bool> edges;
  vector<PII> path;
  int chk[maxn];
  int n;
  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) adj[i].clear();
    edges.clear();
    path.clear();
    memset(chk, 0, sizeof(chk));
  }
  void dfs(int v) {
    for (auto i : adj[v]) {
      if (edges[i.first] == true) {
        edges[i.first] = false;
        dfs(i.second);
        path.EB(MP(i.second, v));
      }
    }
  }
  void add_Edge(int from, int to) {
    edges.PB(true);
    // for bi-directed graph
    adj[from].PB(MP(edges.size() - 1, to));
    adj[to].PB(MP(edges.size() - 1, from));
    chk[from]++;
    chk[to]++;
    // for directed graph
    // adj[from].PB(MP(edges.size()-1, to));
    // check[from]++;
  }
  bool eular_path() {
    int st = -1;
    for (int i = 1; i <= n; i++) {
      if (chk[i] % 2 == 1) {
        st = i;
        break;
      }
    }
    if (st == -1) {
      return false;
    }
    dfs(st);
    return true;
  }
  void print_path(void) {
    for (auto i : path) {
      printf("%d %d\n", i.first, i.second);
    }
  }
};
```

```cpp
// Code from allen(lexicographic order)
#include <bits/stdc++.h>
using namespace std;
const int ALP = 30;
const int MXN = 1005;
int n;
int din[ALP], dout[ALP];
int par[ALP];
vector<string> vs[MXN], ans;
bitset<MXN> vis, used[ALP];
void djsInit() {
  for (int i = 0; i != ALP; ++i) {
    par[i] = i;
  }
}
int Find(int x) { return (x == par[x] ? (x) : (par[x] = Find(par[x]))); }
void init() {
  djsInit();
  memset(din, 0, sizeof(din));
  memset(dout, 0, sizeof(dout));
  vis.reset();
  for (int i = 0; i != ALP; ++i) {
    vs[i].clear();
    used[i].reset();
  }
  return;
}
void dfs(int u) {
  for (int i = 0; i != (int)vs[u].size(); ++i) {
    if (used[u][i]) {
      continue;
    }
    used[u][i] = 1;
    string s = vs[u][i];
    int v = s[s.size() - 1] - 'a';
    dfs(v);
    ans.push_back(s);
  }
}
bool solve() {
  int cnt = 1;
  for (int i = 0; i != n; ++i) {
    string s;
    cin >> s;
    int from = s[0] - 'a', to = s.back() - 'a';
    ++din[to];
    ++dout[from];
    vs[from].push_back(s);
    vis[from] = vis[to] = true;
    if ((from = Find(from)) != (to = Find(to))) {
      par[from] = to;
      ++cnt;
    }
  }
  if ((int)vis.count() != cnt) {
    return false;
  }
  int root, st, pin = 0, pout = 0;
  for (int i = ALP - 1; i >= 0; --i) {
    sort(vs[i].begin(), vs[i].end());
    if (vs[i].size()) root = i;
    int d = dout[i] - din[i];
    if (d == 1) {
      ++pout;
      st = i;
    } else if (d == -1) {
      ++pin;
    } else if (d != 0) {
      return false;
    }
  }
  if (pin != pout || pin > 1) {
    return false;
  }
  ans.clear();
  dfs((pin ? st : root));
  return true;
}
int main() {
  int t;
  cin >> t;
  while (t--) {
    cin >> n;
    init();
    if (!solve()) {
      cout << "***\n";
      continue;
    }
    for (int i = ans.size() - 1; i >= 0; --i) {
      cout << ans[i] << ".\n"[i == 0];
    }
  }
}
```

4 Connectivity

4.1 Articulation Point

```
// from aizu
typedef long long int ll;
typedef unsigned long long int ull;
#define BIG_SIZE 2000000000
#define MOD 1000000007
#define EPS 0.000000001
using namespace std;
#define SIZE 100000
vector<int> G[SIZE];
int N;
bool visited[SIZE];
int visited_order[SIZE], parent[SIZE], lowest[SIZE], number;
void dfs(int cur, int pre_node) {
  visited_order[cur] = lowest[cur] = number;
  number++;
  visited[cur] = true;
  int next;
  for (int i = 0; i < G[cur].size(); i++) {
    next = G[cur][i];
    if (!visited[next]) {
      parent[next] = cur;
      dfs(next, cur);
      lowest[cur] = min(lowest[cur], lowest[next]);
    } else if (visited[next] == true && next != pre_node) {
      lowest[cur] = min(lowest[cur], visited_order[next]);
    }
  }
}
void art_points() {
  for (int i = 0; i < N; i++) visited[i] = false;
  number = 1;
  dfs(0, -1);
  int tmp_parent, root_num = 0;
  vector<int> V;
  for (int i = 1; i < N; i++) {
    tmp_parent = parent[i];
    if (tmp_parent == 0) {
      root_num++;
    } else if (visited_order[tmp_parent] <= lowest[i]) {
      V.push_back(tmp_parent);
    }
  }
  if (root_num >= 2) {
    V.push_back(0);
  }
  sort(V.begin(), V.end());
  V.erase(unique(V.begin(), V.end()), V.end());
  for (int i = 0; i < V.size(); i++) {
    printf("%d\n", V[i]);
  }
}
int main() {
  int E;
  scanf("%d %d", &N, &E);
  int from, to;
  for (int i = 0; i < E; i++) {
    scanf("%d %d", &from, &to);
    G[from].push_back(to);
    G[to].push_back(from);
  }
  art_points();
}
```

4.2 Bridges

```
// from aizu
typedef long long int ll;
typedef unsigned long long int ull;
#define BIG_NUM 2000000000
#define MOD 1000000007
#define EPS 0.000000001
using namespace std;
struct Edge {
  bool operator<(const struct Edge &arg) const {
    if (s != arg.s) {
      return s < arg.s;
    } else {
      return t < arg.t;
    }
  }
  int s, t;
};
struct Info {
  Info(int arg_to, int arg_edge_id) {
    to = arg_to;
    edge_id = arg_edge_id;
  }
  int to, edge_id;
};
int V, E, number;
int order[100000], lowlink[100000];
bool visited[100000];
Edge edge[100000];
vector<Info> G[100000];
void recursive(int cur) {
  order[cur] = number++;
  lowlink[cur] = order[cur];
  int next;
  for (int i = 0; i < G[cur].size(); i++) {
    next = G[cur][i].to;
    if (order[next] == -1) {
      visited[G[cur][i].edge_id] = true;
      recursive(next);
      lowlink[cur] = min(lowlink[cur], lowlink[next]);
    } else if (visited[G[cur][i].edge_id] == false) {
      lowlink[cur] = min(lowlink[cur], order[next]);
    }
  }
}
int main() {
  scanf("%d %d", &V, &E);
  for (int i = 0; i < E; i++) {
    scanf("%d %d", &edge[i].s, &edge[i].t);
    if (edge[i].s > edge[i].t) {
      swap(edge[i].s, edge[i].t);
    }
    G[edge[i].s].push_back(Info(edge[i].t, i));
    G[edge[i].t].push_back(Info(edge[i].s, i));
  }
  sort(edge, edge + E);
  number = 0;
  for (int i = 0; i < V; i++) {
    order[i] = -1;
    lowlink[i] = -1;
```

```cpp
  }
  for (int i = 0; i < E; i++) {
    visited[i] = false;
  }
  recursive(0);
  int from, to;
  for (int i = 0; i < E; i++) {
    from = edge[i].s;
    to = edge[i].t;
    if (order[edge[i].s] > order[edge[i].t]) {
      swap(from, to);
    }
    if (order[from] < lowlink[to]) {
      printf("%d %d\n", edge[i].s, edge[i].t);
    }
  }
  return 0;
}
```

5 Flow & Matching

5.1 Relation

```
1. 一般圖
|最大匹配| + |最小邊覆蓋| = |V|
|最大獨立集| + |最小點覆蓋| = |V|
|最大圖| = |補圖的最大獨立集|
2. 二分圖
|最大匹配| = |最小點覆蓋|
|最大獨立集| = |最小邊覆蓋|
|最大獨立集| = |V| - |最大匹配|
|最大圖| = |補圖的最大獨立集|
```

5.2 Bipartite Matching

```cpp
// 0-base
const int MAXN = ;
int n;
vector<int> G[MAXN];
int vy[MAXN], my[MAXN];
bool match(int u) {
  for (int v : G[u]) {
    if (vy[v]) {
      continue;
    }
    vy[v] = true;
    if (my[v] == -1 || match(my[v])) {
      my[v] = u;
      return true;
    }
  }
  return false;
}
int sol() {
  int cnt = 0;
  memset(my, -1, sizeof(my));
  for (int i = 0; i < n; i++) {
    memset(vy, 0, sizeof(vy));
    if (match(i)) {
      cnt++;
    }
  }
  return cnt;
}
```

5.3 KM

```cpp
const int INF = 1e9;
const int MAXN = ;
struct KM { //1-base
  int n, G[MAXN][MAXN];
  int lx[MAXN], ly[MAXN], my[MAXN];
  bool vx[MAXN], vy[MAXN];
  void init(int _n) {
    n = _n;
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        G[i][j] = 0;
      }
    }
  }
  bool match(int i) {
    vx[i] = true;
    for (int j = 1; j <= n; j++) {
      if (lx[i] + ly[j] == G[i][j] && !vy[j]) {
        vy[j] = true;
        if (!my[j] || match(my[j])) {
          my[j] = i;
          return true;
        }
      }
    }
    return false;
  }
  void update() {
    int delta = INF;
    for (int i = 1; i <= n; i++) {
      if (vx[i]) {
        for (int j = 1; j <= n; j++) {
          if (!vy[j]) {
            delta = min(delta, lx[i] + ly[j] - G[i][j]);
          }
        }
      }
    }
    for (int i = 1; i <= n; i++) {
      if (vx[i]) {
        lx[i] -= delta;
      }
      if (vy[i]) {
        ly[i] += delta;
      }
    }
  }
  int run() {
    for (int i = 1; i <= n; i++) {
      lx[i] = ly[i] = my[i] = 0;
      for (int j = 1; j <= n; j++) {
        lx[i] = max(lx[i], G[i][j]);
      }
    }
    for (int i = 1; i <= n; i++) {
      while (true) {
        for (int i = 1; i <= n; i++) {
          vx[i] = vy[i] = 0;
        }
        if (match(i)) {
          break;
        } else {
          update();
        }
      }
    }
    int ans = 0;
    for (int i = 1; i <= n; i++) {
      ans += lx[i] + ly[i];
    }
    return ans;
  }
};
```

5.4 Dinic

```cpp
#define eb emplace_back
```

```cpp
const LL INF = 1e18;
const int MAXN = ;
struct Edge {
  int to;
  LL cap;
  int rev;
  Edge(int v, LL c, int r) : to(v), cap(c), rev(r) {}
};
struct Dinic {
  int n;
  int level[MAXN], now[MAXN];
  vector<Edge> G[MAXN];
  void init(int _n) {
    n = _n;
    for (int i = 0; i <= n; i++) {
      G[i].clear();
    }
  }
  void add_edge(int u, int v, LL c) {
    G[u].eb(v, c, G[v].size());
    // directed graph
    G[v].eb(u, 0, G[u].size() - 1);
    // undirected graph
    // G[v].eb(u, c, G[u].size() - 1);
  }
  bool bfs(int st, int ed) {
    fill(level, level + n + 1, -1);
    queue<int> q;
    q.push(st);
    level[st] = 0;
    while (!q.empty()) {
      int u = q.front();
      q.pop();
      for (const auto &e : G[u]) {
        if (e.cap > 0 && level[e.to] == -1) {
          level[e.to] = level[u] + 1;
          q.push(e.to);
        }
      }
    }
    return level[ed] != -1;
  }
  LL dfs(int u, int ed, LL limit) {
    if (u == ed) {
      return limit;
    }
    LL ret = 0;
    for (int &i = now[u]; i < G[u].size(); i++) {
      auto &e = G[u][i];
      if (e.cap > 0 && level[e.to] == level[u] + 1) {
        LL f = dfs(e.to, ed, min(limit, e.cap));
        ret += f;
        limit -= f;
        e.cap -= f;
        G[e.to][e.rev].cap += f;
        if (!limit) {
          return ret;
        }
      }
    }
    if (!ret) {
      level[u] = -1;
    }
    return ret;
  }
  LL flow(int st, int ed) {
    LL ret = 0;
    while (bfs(st, ed)) {
      fill(now, now + n + 1, 0);
      ret += dfs(st, ed, INF);
    }
    return ret;
  }
};
```

5.5 MCMF

```cpp
// 0-base
const LL INF = 1e18;
const int MAXN = ;
struct Edge {
  int u, v;
  LL cost;
  LL cap;
  Edge(int _u, int _v, LL _c, LL _cap) : u(_u),
      v(_v), cost(_c), cap(_cap) {}
};
struct MCMF {     // inq times
  int n, pre[MAXN], cnt[MAXN];
  LL ans_flow, ans_cost, dis[MAXN];
  bool inq[MAXN];
  vector<int> G[MAXN];
  vector<Edge> edges;
  void init(int _n) {
    n = _n;
    edges.clear();
    for (int i = 0; i < n; i++) {
      G[i].clear();
    }
  }
  void add_edge(int u, int v, LL c, LL cap) {
    // directed
    G[u].pb(edges.size());
    edges.eb(u, v, c, cap);
    G[v].pb(edges.size());
    edges.eb(v, u, -c, 0);
  }
  bool SPFA(int st, int ed) {
    for (int i = 0; i < n; i++) {
      pre[i] = -1;
      dis[i] = INF;
      cnt[i] = 0;
      inq[i] = false;
    }
    queue<int> q;
    bool negcycle = false;

    dis[st] = 0;
    cnt[st] = 1;
    inq[st] = true;
    q.push(st);

    while (!q.empty() && !negcycle) {
      int u = q.front();
      q.pop();
      inq[u] = false;
      for (int i : G[u]) {
        int v = edges[i].v;
        LL cost = edges[i].cost;
        LL cap = edges[i].cap;

        if (dis[v] > dis[u] + cost && cap > 0) {
          dis[v] = dis[u] + cost;
          pre[v] = i;
          if (!inq[v]) {
            q.push(v);
            cnt[v]++;
            inq[v] = true;

            if (cnt[v] == n + 2) {
              negcycle = true;
              break;
            }
          }
        }
      }
    }

    return dis[ed] != INF;
  }
  LL sendFlow(int v, LL curFlow) {
    if (pre[v] == -1) {
      return curFlow;
    }
    int i = pre[v];
    int u = edges[i].u;
    LL cost = edges[i].cost;

    LL f = sendFlow(u, min(curFlow, edges[i].cap));

    ans_cost += f * cost;
    edges[i].cap -= f;
    edges[i ^ 1].cap += f;
    return f;
  }
  pair<LL, LL> run(int st, int ed) {
    ans_flow = ans_cost = 0;
    while (SPFA(st, ed)) {
      ans_flow += sendFlow(ed, INF);
    }
    return make_pair(ans_flow, ans_cost);
  }
};
```

6 String

6.1 Manacher

```
int p[2 * MAXN];
int Manacher(const string &s) {
  string st = "@#";
  for (char c : s) {
    st += c;
    st += '#';
  }
  st += '$';
  int id = 0, mx = 0, ans = 0;
  for (int i = 1; i < st.length() - 1; i++) {
    p[i] = (mx > i ? min(p[2 * id - i], mx - i) : 1);
    for (; st[i - p[i]] == st[i + p[i]]; p[i]++);
    if (mx < i + p[i]) {
      mx = i + p[i];
      id = i;
    }
    ans = max(ans, p[i] - 1);
  }
  return ans;
}
```

6.2 Trie

```
const int MAXL = ;
const int MAXC = ;
struct Trie {
  int nex[MAXL][MAXC];
  int len[MAXL];
  int sz;
  void init() {
    memset(nex, 0, sizeof(nex));
    memset(len, 0, sizeof(len));
    sz = 0;
  }
  void insert(const string &str) {
    int p = 0;
    for (char c : str) {
      int id = c - 'a';
      if (!nex[p][id]) {
        nex[p][id] = ++sz;
      }
      p = nex[p][id];
    }
    len[p] = str.length();
  }
  vector<int> find(const string &str, int i) {
    int p = 0;
    vector<int> ans;
    for (; i < str.length(); i++) {
      int id = str[i] - 'a';
      if (!nex[p][id]) {
        return ans;
      }
      p = nex[p][id];
      if (len[p]) {
        ans.pb(len[p]);
      }
    }
    return ans;
  }
};
```

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

```
// ax + by = c
int extgcd(int a, int b, int c, int &x, int &y) {
  if (b == 0) {
    x = c / a;
    y = 0;
    return a;
  }
  int d = extgcd(b, a % b, c, y, x);
  y -= (a / b) * x;
  return d;
}
```

7.3 Gaussian Elimination

```
const int MAXN = 300;
const double EPS = 1e-8;
int n;
double A[MAXN][MAXN];
void Gauss() {
  for (int i = 0; i < n; i++) {
    bool ok = 0;
    for (int j = i; j < n; j++) {
      if (fabs(A[j][i]) > EPS) {
        swap(A[j], A[i]);
        ok = 1;
        break;
      }
    }
    if (!ok) continue;
    double fs = A[i][i];
    for (int j = i + 1; j < n; j++) {
      double r = A[j][i] / fs;
      for (int k = i; k < n; k++) {
        A[j][k] -= A[i][k] * r;
      }
    }
  }
}
```

7.4 Phi

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

```
void phi_table(int n) {
  phi[1] = 1;
  for (int i = 2; i <= n; i++) {
    if (phi[i]) {
      continue;
    }
    for (int j = i; j < n; j += i) {
      if (!phi[j]) {
        phi[j] = j;
      }
      phi[j] = phi[j] / i * (i - 1);
    }
  }
}
```

8 Geometry

8.1 Point

```
// notice point type!!!
using dvt = int;
const double EPS = 1e-6;
const double PI = acos(-1);

struct Pt {
  dvt x;
  dvt y;
};
bool operator < (const Pt &a, const Pt &b) {
  return a.x == b.x ? a.y < b.y : a.x < b.x;
}
bool operator == (const Pt &a, const Pt &b) {
  return a.x == b.x && a.y == b.y;
}
Pt operator + (const Pt &a, const Pt &b) {
  return {a.x + b.x, a.y + b.y};
}
Pt operator - (const Pt &a, const Pt &b) {
  return {a.x - b.x, a.y - b.y};
}
// multiply constant
Pt operator * (const Pt &a, const dvt c) {
  return {a.x * c, a.y * c};
}
Pt operator / (const Pt &a, const dvt c) {
  return {a.x / c, a.y / c};
}
// |a| x |b| x cos(x)
dvt iproduct(const Pt &a, const Pt &b) {
  return a.x * b.x + a.y * b.y;
}
// |a| x |b| x sin(x)
dvt cross(const Pt &a, const Pt &b) {
  return a.x * b.y - a.y * b.x;
}
dvt dis_pp(const Pt &a, const Pt, &b) {
  dvt dx = a.x - b.x;
  dvt dy = a.y - b.y;
  return sqrt(dx * dx, dy * dy);
}
```

8.2 Line

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

```
struct Line {
  Pt st;
  Pt ed;
};
// return point side
// left, on line, right -> 1, 0, -1
int side(Line l, Pt a) {
  dvt cross_val = cross(a - l.st, l.ed - l.st);
  if (cross_val > EPS) {
    return 1;
  } else if (cross_val < -EPS) {
    return -1;
  } else {
    return 0;
  }
}
// AB infinity, CD segment
bool has_intersection(Line AB, Line CD) {
  int c = side(AB, CD.st);
  int d = side(AB, CD.ed);
  if (c == 0 || d == 0) {
    return true;
  } else {
    // different side
    return c == -d;
  }
}
// find intersection point, two line, not seg
pair<int, Pt> intersection(Line a, Line b) {
  Pt A = a.ed - a.st;
  Pt B = b.ed - b.st;
  Pt C = b.st - a.st;
  dvt mom = cross(A, B);
  dvt son = cross(C, B);
  if (std::abs(mom) <= EPS) {
    if (std::abs(son) <= EPS) {
      return {1, {}}; // same line
    } else {
      return {2, {}}; // parallel
    }
  } else {             // ok
    return {0, a.st + A * (son / mom)};
  }
}
// line to point distance
dvt dis_lp(Line l, Pt a) {
  return area3x2(l.st, l.ed, a) / dis_pp(l.st, l.ed);
}
```

8.3 Area

```
// triangle
dvt area3(Pt a, Pt b, Pt c) {
  return std::abs(cross(b - a, c - a) / 2);
}
dvt area3x2(Pt a, Pt b, Pt c) { // for integer
  return std::abs(cross(b - a, c - a));
}
```

```cpp
// simple convex area(can in)
dvt area(vector<Pt> &a) {
  dvt ret = 0;
  for (int i = 0, sz = a.size(); i < sz; i++) {
    ret += cross(a[i], a[(i + 1) % sz]);
  }
  return std::abs(ret) / 2;
}
// check point in/out a convex
int io_convex(vector<Pt> convex, Pt q) {
  // convex is Counterclockwise
  for (int i = 0, sz = convex.size(); i < sz; i++) {
    Pt cur = convex[i] - q;
    Pt nex = convex[(i + 1) % sz] - q;
    dvt cross_val = cross(cur, nex);
    if (std::abs(cross_val) <= EPS) {
      return 0;  // on edge
    }
    if (cross_val < 0) {
      return -1; // outside
    }
  }
  return 1;      // inside
}
```

8.4 Convex Hull

```cpp
vector<Pt> convex_hull(vector<Pt> &a) {
  sort(a.begin(), a.end());
  a.erase(unique(a.begin(), a.end()), a.end());
  int sz = a.size(), m = 0;
  vector<Pt> ret(sz + 5); // safe 1 up
  for (int i = 0; i < sz; i++) {
    while (m > 1 &&
      cross(ret[m - 1] - ret[m - 2], a[i] - ret[m - 2]) <= EPS) {
      m--;
    }
    ret[m++] = a[i];
  }
  int k = m;
  for (int i = sz - 2; i >= 0; i--) {
    while (m > k &&
      cross(ret[m - 1] - ret[m - 2], a[i] - ret[m - 2]) <= EPS) {
      m--;
    }
    ret[m++] = a[i];
  }
  if (sz > 1) {
    m--;
  }
  ret.resize(m);
  return ret;
}
```