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 Default

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

int main() {

}
```

1.3 Black Magic

```cpp
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/pb_ds/priority_queue.hpp>
using namespace std;
using namespace __gnu_pbds;
using set_t =
  tree<int, null_type, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;
using map_t =
  tree<int, int, less<int>, rb_tree_tag,
    tree_order_statistics_node_update>;
using heap_t =
  __gnu_pbds::priority_queue<int>;
using ht_t =
  gp_hash_table<int, int>;
int main() {
  //set----------------------------------
  set_t st;
  st.insert(5); st.insert(6);
  st.insert(3); st.insert(1);

  // the smallest is (0), biggest is (n-1), kth small
      is (k-1)
  int num = *st.find_by_order(0);
  cout << num << '\n'; // print 1

  num = *st.find_by_order(st.size() - 1);
  cout << num << '\n'; // print 6

  // find the index
  int index = st.order_of_key(6);
  cout << index << '\n'; // print 3

  // check if there exists x
  int x = 5;
  int check = st.erase(x);
  if (check == 0) printf("st not contain 5\n");
  else if (check == 1) printf("st contain 5\n");

  //tree policy like set
  st.insert(5); st.insert(5);
  cout << st.size() << '\n'; // print 4

  //map----------------------------------
  map_t mp;
  mp[1] = 2;
  cout << mp[1] << '\n';
  auto tmp = *mp.find_by_order(0); // pair
  cout << tmp.first << " " << tmp.second << '\n';

  //heap----------------------------------
  heap_t h1, h2;
  h1.push(1); h1.push(3);
  h2.push(2); h2.push(4);
  h1.join(h2);
```

```
  cout << h1.size() << h2.size() << h1.top() << '\n';
  // 404

  //hash-table----------------------------
  ht_t ht;
  ht[85] = 5;
  ht[89975] = 234;
  for (auto i : ht) {
    cout << i.first << " " << i.second << '\n';
  }
}
```

1.4 Python

```
### EOF
while True:
  try:
    pass
  except EOFError:
    break
###math
import math

math.ceil(x)#上高斯
math.floor(x)#下高斯
math.factorial(x)#接乘
math.fabs(x)#絕對值
math.fsum(arr)#跟sum一樣但更精確(小數點問題)
math.gcd(x, y)#bj4
math.exp(x)#e^x
math.log(x, base)
math.log2(x)#2為底
math.log10(x)#10為底
math.sqrt(x)
math.pow(x, y, mod)#精確些(float型態) MOD!!!
math.sin(x)# cos tan asin acos atan atan2(弧度) sinh
    cosh tanh acosh asinh atanh
math.hypot(x, y)#歐幾里德範數
math.degrees(x)#x從弧度轉角度
math.radians(x)#x從角度轉弧度
math.gamma(x)#x的gamma函數
math.pi#常數
math.e#常數
math.inf

### ascii
ord(x)#char to asc
chr(x)#asc to char

x.encode().hex()#string to hex
### reverse string
string = "abc"
string_reverse = string[::-1]
```

1.5 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);
```

1.6 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 Disjoint Set

```
// 0-base
const int MAXN = 1000;
int boss[MAXN];
void init(int n) {
  for (int i = 0; i < n; i++) {
    boss[i] = -1;
  }
}
int find(int x) {
  if (boss[x] < 0) {
    return x;
  }
  return boss[x] = find(boss[x]);
}
bool uni(int a, int b) {
  a = find(a);
  b = find(b);
  if (a == b) {
    return false;
  }
  if (boss[a] > boss[b]) {
    swap(a, b);
  }
  boss[a] += boss[b];
  boss[b] = a;
  return true;
}
```

2.2 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);
}
```

2.3 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;
}
```

2.4 Segment Tree RARMQ

```
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));
    }
  }
};
```

2.5 Segment Tree RURMQ

```
// from aizu
const ll maxn = 1e6 + 5;
const ll INF = 2147483647;
ll tree[maxn << 2], a[maxn], laze[maxn << 2];
void push_down(int rt, int ln, int rn) {
  if (laze[rt] != -1) {
    laze[rt << 1] = laze[rt];
    laze[rt << 1 | 1] = laze[rt];
    tree[rt << 1] = laze[rt];
    tree[rt << 1 | 1] = laze[rt];
    laze[rt] = -1;
  }
}
void push_up(int rt) { tree[rt] = min(tree[rt << 1 |
    1], tree[rt << 1]); }
void build(ll l, ll r, ll rt) {
  if (l == r) {
    tree[rt] = a[l];
    return;
  }
  ll m = (l + r) >> 1;
  build(l, m, rt << 1);
  build(m + 1, r, rt << 1 | 1);
  push_up(rt);
}
//区间更新
void update(ll L, ll R, ll c, ll l, ll r, ll rt) {
```

```
  if (L <= l && r <= R) {
    laze[rt] = c;
    tree[rt] = c;
    return;
  }
  ll m = (l + r) >> 1;
  push_down(rt, m - l + 1, r - m);
  if (m >= L) update(L, R, c, l, m, rt << 1);
  if (m < R) update(L, R, c, m + 1, r, rt << 1 | 1);
  push_up(rt);
}

//区间更新的区间查询
ll query(ll L, ll R, ll l, ll r, ll rt) {
  if (L <= l && r <= R) {
    return tree[rt];
  }
  ll m = (l + r) >> 1, sum = INF;
  push_down(rt, m - l + 1, r - m);
  if (m >= L) sum = min(sum, query(L, R, l, m, rt <<
      1));
  if (m < R) sum = min(sum, query(L, R, m + 1, r, rt
      << 1 | 1));
  return sum;
}
void init() {
  for (int i = 0; i < maxn; i++) a[i] = INF;
  for (int i = 0; i < maxn * 4; i++) laze[i] = -1;
  build(0, n - 1, 1);
}
/*
update(x,y,z,0,n-1,1);
query(x,y,0,n-1,1)
*/
```

2.6 Treap

```
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 Dijkstra

```
// 0-base
const LL INF = 1e18;
const int MAXN = ;
struct Edge {
  int to;
  LL cost;
  Edge(int v, LL c) : to(v), cost(c) {}
  bool operator < (const Edge &other) const {
    return cost > other.cost;
  }
};

int n;
LL dis[MAXN];
vector<Edge> G[MAXN];

void init() {
  for (int i = 0; i < n; i++) {
    G[i].clear();
    dis[i] = INF;
  }
}
void Dijkstra(int st, int ed = -1) {
  priority_queue<Edge> pq;
  pq.emplace(st, 0);
  dis[st] = 0;
  while (!pq.empty()) {
    auto now = pq.top();
    pq.pop();
    if (now.to == ed) {
      return;
    }
    if (now.cost > dis[now.to]) {
      continue;
    }
    for (auto &e : G[now.to]) {
      if (dis[e.to] > now.cost + e.cost) {
        dis[e.to] = now.cost + e.cost;
        pq.emplace(e.to, dis[e.to]);
      }
    }
  }
}
```

3.2 SPFA(negative cycle)

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

```
const int MAXN = ;
struct Edge {
  int to;
  LL cost;
  Edge(int v, LL c) : to(v), cost(c) {}
};

int n;
LL dis[MAXN];
vector<Edge> G[MAXN];

void init() {
  for (int i = 0; i < n; i++) {
    G[i].clear();
    dis[i] = INF;
  }
}
bool SPFA(int st) {
  vector<int> cnt(n, 0);
  vector<bool> inq(n, false);
  queue<int> q;

  q.push(st);
  dis[st] = 0;
  inq[st] = true;
  while (!q.empty()) {
    int now = q.front();
    q.pop();
    inq[now] = false;
    for (auto &e : G[now]) {
      if (dis[e.to] > dis[now] + e.cost) {
        dis[e.to] = dis[now] + e.cost;
        if (!inq[e.to]) {
          cnt[e.to]++;
          if (cnt[e.to] > n) {
            // negative cycle
            return false;
          }
          inq[e.to] = true;
          q.push(e.to);
        }
      }
    }
  }
  return true;
}
```

3.3 Floyd Warshall

```
// 0-base
// G[i][i] < 0 -> negative cycle
const LL INF = 1e18;
const int MAXN = ;

int n;
LL G[MAXN][MAXN];

void init() {
  for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
      G[i][j] = INF;
    }
    G[i][i] = 0;
  }
}
void floyd() {
  for (int k = 0; k < n; k++) {
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        if (G[i][k] != INF && G[k][j] != INF) {
          G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
        }
      }
    }
  }
}
```

3.4 Topological Sort

```
// 0-base
// if ret.size < n -> cycle
int n;
vector<vector<int>> G;

vector<int> topoSort() {
  vector<int> indeg(n), ret;
  for (auto &li : G) {
    for (int x : li) {
      ++indeg[x];
    }
  }
  // use priority queue for lexic. largest ans
  queue<int> q;
  for (int i = 0; i < n; i++) {
    if (!indeg[i]) {
      q.push(i);
    }
  }
  while (!q.empty()) {
    int u = q.front();
    q.pop();
    ret.pb(u);
    for (int v : G[u]) {
      if (--indeg[v] == 0) {
        q.push(v);
      }
    }
  }
  return ret;
}
```

3.5 Tree Diameter

```
// 0-base;
const int MAXN = ;

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

int n, d = 0;
int d1[MAXN], d2[MAXN];
vector<Edge> G[MAXN];
// dfs(0, -1);
void dfs(int u, int from) {
  d1[u] = d2[u] = 0;
  for (auto e : G[u]) {
    if (e.to == from) {
      continue;
    }
    dfs(e.to, u);
    int t = d1[e.to] + e.cost;
    if (t > d1[u]) {
      d2[u] = d1[u];
      d1[u] = t;
    } else if (t > d2[u]) {
      d2[u] = t;
    }
  }
  d = max(d, d1[u] + d2[u]);
}
```

3.6 Directed MST

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

```

```cpp
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;
  }
};
```

3.7 LCA

```cpp
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);
*/
```

3.8 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。

```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 Kosaraju SCC

```
// 0-base
int n;
vector<vector<int>> G, G2; // G2 = G rev
vector<bool> vis;
vector<int> s, color;
int sccCnt;
void dfs1(int u) {
  vis[u] = true;
  for (int v : G[u]) {
    if (!vis[v]) {
      dfs1(v);
    }
  }
  s.pb(u);
}
void dfs2(int u) {
  color[u] = sccCnt;
  for (int v : G2[u]) {
    if (!color[v]) {
      dfs2(v);
    }
  }
}
void Kosaraju() {
  sccCnt = 0;
  for (int i = 0; i < n; i++) {
    if (!vis[i]) {
      dfs1(i);
    }
  }
  for (int i = n - 1; i >= 0; i--) {
    if (!color[s[i]]) {
      ++sccCnt;
      dfs2(s[i]);
    }
  }
}
```

4.2 BCC

```
typedef pair<int, int> PII;
int low[MXV], depth[MXV];
bool is_cut_vertex[MXV], visit[MXV];
vector<int> G[MXV];
vector<PII> BCC[MXV];
int bcc_cnt = 0;
stack<PII> st;

vector<pair<int, int>> my_cut_edge;

void dfs(int now, int cur_depth, int f) {
  visit[now] = true;
  depth[now] = low[now] = cur_depth;
  int cut_son = 0;
  for (auto i : G[now]) {
    if (i == f) continue;
    if (visit[i]) {  // ancestor
      if (depth[i] < depth[now]) { // #
        low[now] = min(low[now], depth[i]);
        st.push({now, i});
      }
    } else {  // offspring
      st.push({now, i});
      dfs(i, cur_depth + 1, now);
      cut_son += 1;
      low[now] = min(low[now], low[i]);
      if (low[i] >= depth[now]) {
        is_cut_vertex[now] = true;
        auto t = st.top();
        st.pop();
        while (t != make_pair(now, i)) {
          BCC[bcc_cnt].push_back(t);
          t = st.top();
          st.pop();
        }
        BCC[bcc_cnt].push_back(t);
        ++bcc_cnt;
      }
      // ###
      if (low[i] > depth[now])
        my_cut_edge.push_bach({now, i});
    }
  }
  if (cur_depth == 0)
    is_cut_vertex[now] = (cut_son != 1);
  return;
}

bool is_2_edge_connected(int n) {
  memset(visit, 0, sizeof(visit));
  dfs(1, 0, -1);
  return my_cut_edge.size() == 0;
}
```

4.3 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();
}
```

4.4 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;
  }
  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

```
// 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

```
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

```
#define eb emplace_back
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

```
// 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);
  }
};
```

5.6 Stable Matching

```
int t, n, b[N][N], bi[N], g[N][N], bg[N], gb[N];

void sol() {
  deque<int> dq;
  memset(gb, 0, sizeof(gb));
  memset(bi, 0, sizeof(bi));
  for (int i = 1; i <= n; i++) dq.push_back(i);
  while (!dq.empty()) {
    int x = dq.front();
    dq.pop_front();
    int y = b[x][++bi[x]];
    if (!gb[y]) {
      gb[y] = x;
      bg[x] = y;
    } else if (g[y][x] < g[y][gb[y]]) {
      dq.push_back(gb[y]);
      gb[y] = x;
      bg[x] = y;
    } else {
      dq.push_back(x);
    }
  }
  for (int i = 1; i <= n; i++) {
    cout << bg[i] << '\n';
  }
}

int main() {
  int x;
  cin >> t;
  for (int i = 0; i < t; i++) {
    cin >> n;
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        cin >> b[i][j];
      }
    }
    for (int i = 1; i <= n; i++) {
      for (int j = 1; j <= n; j++) {
        cin >> x;
        g[i][x] = j;
      }
    }
    if (i) cout << '\n';
    sol();
  }
}
```

5.7 Max General Graph Matching

```
#define maxn

int fa[maxn], pre[maxn], match[maxn], s[maxn],
    v[maxn];
vector<int> g[maxn];
queue<int> q;
void Init(int n) {
  for (int i = 0; i <= n; ++i) match[i] = pre[i] = n;
  for (int i = 0; i < n; ++i) g[i].clear();
}
void AddEdge(int u, int v) {
  g[u].push_back(v);
  g[v].push_back(u);
}
int Find(int u) { return u == fa[u] ? u : fa[u] =
    Find(fa[u]); }
int LCA(int x, int y, int n) {
  static int tk = 0;
  tk++;
  x = Find(x), y = Find(y);
  for (;; swap(x, y)) {
    if (x != n) {
      if (v[x] == tk) return x;
      v[x] = tk;
      x = Find(pre[match[x]]);
    }
  }
}

void Blossom(int x, int y, int l) {
  while (Find(x) != l) {
    pre[x] = y, y = match[x];
    if (s[y] == 1) q.push(y), s[y] = 0;
    if (fa[x] == x) fa[x] = l;
    if (fa[y] == y) fa[y] = l;
    x = pre[y];
  }
}

bool Bfs(int r, int n) {
  for (int i = 0; i <= n; ++i) fa[i] = i, s[i] = -1;
  while (!q.empty()) q.pop();
  q.push(r);
  s[r] = 0;
  while (!q.empty()) {
    int x = q.front();
    q.pop();
    for (int u : g[x]) {
      if (s[u] == -1) {
        pre[u] = x, s[u] = 1;
        if (match[u] == n) {
          for (int a = u, b = x, last; b != n; a =
              last, b = pre[a])
            last = match[b], match[b] = a, match[a] =
                b;
          return true;
        }
        q.push(match[u]);
        s[match[u]] = 0;
      } else if (!s[u] && Find(u) != Find(x)) {
        int l = LCA(u, x, n);
        Blossom(x, u, l);
        Blossom(u, x, l);
      }
    }
  }
  return false;
}

int Solve(int n) {
  int res = 0;
  for (int x = 0; x < n; ++x) {
    if (match[x] == n) res += Bfs(x, n);
  }
  return res;
}
```

6 String

6.1 Manacher

```cpp
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

```cpp
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;
  }
};
```

6.3 Z-value

```cpp
// 0-base
// 對於個長度為 n 的字串 s
// 定義函數 z[i] 表示 s 和 s[i, n - 1]
// (即以 s[i] 開頭的後綴) 的最長公共前綴 (LCP) 的長度
// z[0] = 0。
vector<int> z_function(string s) {
  int n = (int)s.length();
  vector<int> z(n);
  for (int i = 1, l = 0, r = 0; i < n; ++i) {
    if (i <= r && z[i - l] < r - i + 1) {
      z[i] = z[i - l];
    } else {
      z[i] = max(0, r - i + 1);
      while (i + z[i] < n && s[z[i]] == s[i + z[i]])
          ++z[i];
    }
    if (i + z[i] - 1 > r) l = i, r = i + z[i] - 1;
  }
  return z;
}
```

7 DP

7.1 LIS

```cpp
int LIS(vector<int> &a) {
  vector<int> s;
  for (int i = 0; i < a.size(); i++) {
    if (s.empty() || s.back() < a[i]) {
      s.push_back(a[i]);
    } else {
      *lower_bound(s.begin(), s.end(), a[i],
        [](int x, int y) {return x < y;}) = a[i];
    }
  }
  return s.size();
}
```

7.2 LCS

```cpp
int LCS(string s1, string s2) {
  int n1 = s1.size(), n2 = s2.size();
  vector<vector<int>> dp(n1 + 1, vector<int>(n2 + 1,
      0));
  for (int i = 1; i <= n1; i++) {
    for (int j = 1; j <= n2; j++) {
      if (s1[i - 1] == s2[j - 1]) {
        dp[i][j] = dp[i - 1][j - 1] + 1;
      } else {
        dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
      }
    }
  }
  return dp[n1][n2];
}
```

7.3 Huge Knapsack

```cpp
// from aizu
#include <bits/stdc++.h>
typedef long long int ll;
typedef unsigned long long int ull;
#define BIG_NUM 2000000000
#define HUGE_NUM 99999999999999999
#define MOD 1000000007
#define EPS 0.000000001
using namespace std;

#define SIZE 25

struct Info {
  Info() { value = 0, weight = 0; }
  Info(ll arg_value, ll arg_weight) {
```

```
    value = arg_value;
    weight = arg_weight;
  }
  bool operator<(const struct Info &arg) const {
      return weight < arg.weight; }

  ll value, weight;
};

ll N, W;
ll POW[SIZE];
ll table_B[1 << 21];
Info info[45];

int main() {
  POW[0] = 1;
  for (int i = 1; i < SIZE; i++) {
    POW[i] = POW[i - 1] * 2;
  }

  scanf("%lld %lld", &N, &W);

  for (int i = 0; i < N; i++) {
    scanf("%lld %lld", &info[i].value,
        &info[i].weight);
  }

  if (N == 1) {
    if (info[0].weight <= W) {
      printf("%lld\n", info[0].value);
    } else {
      printf("0\n");
    }

    return 0;
  }

  vector<int> A, B;
  for (int i = 0; i < N / 2; i++) {
    A.push_back(i);
  }
  for (int i = N / 2; i < N; i++) {
    B.push_back(i);
  }

  vector<Info> vec_A, vec_B;
  for (int state = 0; state < POW[A.size()]; state++)
      {
    ll sum_w = 0;
    ll sum_value = 0;
    for (int loop = 0; loop < A.size(); loop++) {
      if (state & POW[loop]) {
        sum_w += info[A[loop]].weight;
        sum_value += info[A[loop]].value;
      }
    }
    vec_A.push_back(Info(sum_value, sum_w));
  }
  sort(vec_A.begin(), vec_A.end());

  for (int state = 0; state < POW[B.size()]; state++)
      {
    ll sum_w = 0;
    ll sum_value = 0;
    for (int loop = 0; loop < B.size(); loop++) {
      if (state & POW[loop]) {
        sum_w += info[B[loop]].weight;
        sum_value += info[B[loop]].value;
      }
    }
    vec_B.push_back(Info(sum_value, sum_w));
  }
  sort(vec_B.begin(), vec_B.end());

  table_B[0] = vec_B[0].value;
  for (int i = 1; i < vec_B.size(); i++) {
    //ある重さ以下の最大価値を求める
    table_B[i] = max(table_B[i - 1], vec_B[i].value);
  }

  int tail = vec_B.size() - 1;
  ll ans = 0;
  for (int i = 0; i < vec_A.size(); i++) {
    while (tail >= 0 && vec_A[i].weight +
        vec_B[tail].weight > W) tail--;
    if (tail < 0) break;

    ans = max(ans, vec_A[i].value + table_B[tail]);
  }

  printf("%lld\n", ans);
  return 0;
}
```

7.4 Coin Change

```
// from aizu
int main() {
  int n, m, min, tmp;
  scanf("%d", &n);
  int minimum[n + 1];
  scanf("%d", &m);
  int coin[m];
  for (int i = 0; i < m; i++) scanf("%d", &coin[i]);

  minimum[0] = 0;
  minimum[1] = 1;
  for (int i = 2; i <= n; i++) {
    min = n + 1;
    for (int k = 0; k < m; k++) {
      if (coin[k] <= i) {
        tmp = 1 + minimum[i - coin[k]];
        min = (min <= tmp) ? min : tmp;
      }
    }
    minimum[i] = min;
  }

  printf("%d\n", minimum[n]);

  return 0;
}
```

7.5 Edit Distance

```
// from aizu
typedef long long int ll;
typedef unsigned long long int ull;
#define BIG_NUM 2000000000
#define MOD 1000000007
#define EPS 0.000000001

int main() {
  char A[1001], B[1001];
  int len_A, len_B;
  scanf("%s %s", A, B);

  for (len_A = 0; A[len_A] != '\0'; len_A++);
  for (len_B = 0; B[len_B] != '\0'; len_B++);

  int** dp = new int*[len_B + 1];

  for (int row = 0; row <= len_B; row++) {
    dp[row] = new int[len_A + 1];
  }

  for (int col = 0; col <= len_A; col++) {
    dp[0][col] = col;
  }

```

```
  for (int row = 1; row <= len_B; row++) {
    dp[row][0] = row;
  }

  int cost;

  for (int row = 1; row <= len_B; row++) {
    for (int col = 1; col <= len_A; col++) {
      if (A[col - 1] == B[row - 1]) {
        cost = 0;
      } else {
        cost = 1;
      }
      dp[row][col] = min(dp[row - 1][col - 1] + cost,
        min(dp[row - 1][col] + 1, dp[row][col - 1] +
            1));
    }
  }

  printf("%d\n", dp[len_B][len_A]);
}
```

7.6 Maximum Subrectangle

```
int kadane2D(int array[N][M]) {
  // Modify the array's elements to now hold the sum
  // of all the numbers that are above that element
      in its column
  for (int i = 1; i < N; i++) {
    for (int j = 0; j < M; j++) {
      array[i][j] += array[i - 1][j];
    }
  }

  int ans = 0;  // Holds the maximum sum matrix found
      till now

  for (int bottom = 0; bottom < N; bottom++) {
    for (int top = bottom; top < N; top++) {
      // loop over all the N^2 sub problems
      int[] sums = new int[N];

      // store the sum of numbers between the two rows
      // in the sums array
      for (int i = 0; i < M; i++) {
        if (bottom > 0) {
          sums[i] = array[top][i] - array[bottom -
              1][i];
        } else {
          sums[i] = array[top][i];
        }
      }

      // O(n) time to run 1D kadane's on this sums
          array
      ans = Math.max(ans, kadane1d(sums));
    }
  }
  return ans;
}
```

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

```
// 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;
}
```

8.3 Gaussian Elimination + det

```
const double EPS = 1e-6;
double Gauss(vector<vector<double>> &d) {
  int n = d.size(), m = d[0].size();
  double det = 1;
  for (int i = 0; i < m; i++) {
    int p = -1;
    for (int j = i; j < n; j++) {
      if (fabs(d[j][i]) < EPS) {
        continue;
      }
      if (p == -1 || fabs(d[j][i]) > fabs(d[p][i])) {
        p = j;
      }
    }
    if (p == -1) {
      continue;
    }
    if (p != i) {
      det *= -1;
    }
    for (int j = 0; j < m; j++) {
      swap(d[p][j], d[i][j]);
    }
    for (int j = 0; j < n; j++) {
      if (i == j) {
        continue;
      }
      double z = d[j][i] / d[i][i];
      for (int k = 0; k < m; k++) {
        d[j][k] -= z * d[i][k];
      }
    }
  }
  for (int i = 0; i < n; i++) {
    det *= d[i][i];
  }
  return det;
}
// new
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;
      }
    }
  }
}
```

8.4 Prime Table

```
vector<int> p;
bitset<MAXN> is_notp;
void PrimeTable(int n) {
  is_notp.reset();
  is_notp[0] = is_notp[1] = 1;
  for (int i = 2; i <= n; ++i) {
    if (!is_notp[i]) {
      p.push_back(i);
    }
    for (int j = 0; j < (int)p.size(); ++j) {
      if (i * p[j] > n) {
        break;
      }
      is_notp[i * p[j]] = 1;
      if (i % p[j] == 0) {
        break;
      }
    }
  }
}
```

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 $gcd(p,q)=1, \Phi(p)\cdot\Phi(q)=\Phi(p\cdot q)$
- $\Phi(p^k)=p^{k-1}\times(p-1)$

```
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);
    }
  }
}
```

8.6 Chinese Remainder Thm

```
//参数可为负数的扩展欧几里德定理
void exOJLD(int a, int b, int& x, int& y) {
  //根据欧几里德定理
  if (b == 0) {  //任意数与0的最大公约数为其本身。
    x = 1;
    y = 0;
  } else {
    int x1, y1;
    exOJLD(b, a % b, x1, y1);
    if (a * b < 0) {  //异号取反
      x = -y1;
      y = a / b * y1 - x1;
    } else {  //同号
      x = y1;
      y = x1 - a / b * y1;
    }
  }
}
//剩余定理
int calSYDL(int a[], int m[], int k) {
  int N[k];    //这个可以删除
  int mm = 1;  //最小公倍数
  int result = 0;
  for (int i = 0; i < k; i++) {
    mm *= m[i];
  }
  for (int j = 0; j < k; j++) {
    int L, J;
    exOJLD(mm / m[j], -m[j], L, J);
    N[j] = m[j] * J + 1;   // 1
    N[j] = mm / m[j] * L;  // 2
        1和2这两个值应该是相等的。
    result += N[j] * a[j];
  }
  return (result % mm + mm) % mm;
  //落在(0, mm)之间，这么写是为了防止result初始为负数
  //本例中不可能为负可以直接
  //写成：return result%mm;即可。
}

int main() {
  int a[3] = {2, 3, 6};  // a[i]=n%m[i]
  int m[3] = {3, 5, 7};
  cout << calSYDL(a, m, 3) << endl;
  //輸出為滿足兩條陣列的最小n,第3參數為陣列長度
  //所有滿足答案的數字集合為n+gcd(m0,m1,m2...)*k,
      k為正數
  return 0;
}
```

8.7 Josephus

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

8.8 Catalan

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

```
long long f[N] = {1}, i, t, p;
int main() {
  for (int i = 1; i <= 100; i++) {
    f[i] = f[i - 1] * (4 * i - 2) % mod;
    for (t = i + 1, p = mod - 2; p; t = (t * t) %
        mod, p >>= 1LL) {
      if (p & 1) {
        f[i] *= t;
```

```
        f[i] %= mod;
      }
    }
  }
}
```

8.9 Matrix Multiplication

```
struct Matrix {
  int row, col;
  vector<vector<int>> v;
  Matrix() : row(0), col(0) {}
  Matrix(int r, int c) : row(r), col(c) {
    v = vector<vector<int>>(r, vector<int>(c, 0));
  }
};
Matrix operator * (Matrix &a, Matrix &b) {
  assert(a.col == b.row);
  Matrix ret(a.row, b.col);
  for (int i = 0; i < a.row; i++) {
    for (int j = 0; j < b.col; j++) {
      for (int k = 0; k < a.col; k++) {
        ret.v[i][j] += a.v[i][k] * b.v[k][j];
      }
    }
  }
  return ret;
}
Matrix mPow(Matrix a, int n) {
  assert(a.row == a.col);
  Matrix ret(a.row, a.col);
  ret.v[0][0] = ret.v[1][1] = 1;
  while (n > 0) {
    if (n & 1) {
      ret = ret * a;
    }
    a = a * a;
    n >>= 1;
  }
  return ret;
}
```

8.10 Fibonacci

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

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

$$O(logn)$$

```
LL fib(int n) {
  if (n <= 1) {
    return n;
  }
  Matrix a(2, 2), b(2, 1);
  a.v[0][0] = a.v[0][1] = a.v[1][0] = 1;
  b.v[0][0] = 1;
  auto t = mPow(a, n - 1);
  t = t * b;
  return t.v[0][0];
}
```

9 Geometry

9.1 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);
}
```

9.2 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);
}
```

9.3 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));
}
// 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
}
```

9.4 Convex Hull

```
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;
}
```

9.5 Closest Pair

```
// from aizu
struct Info {
  Info() { x = y = 0; }
  Info(double arg_x, double arg_y) {
    x = arg_x;
    y = arg_y;
  }
  bool operator<(const struct Info& arg) const {
    return x < arg.x;
  };
  double x, y;
};

int N;

bool compare_y(Info left, Info right) {
  return left.y < right.y;
}

Info* info;

double closest_pair(Info* array, int tmp_N) {
  if (tmp_N <= 1) return DBL_MAX;

  int mid = tmp_N / 2;
  double x = array[mid].x;
  double dist =
      min(closest_pair(array, mid),
          closest_pair(array + mid, tmp_N - mid));
  inplace_merge(array, array + mid, array + tmp_N,
      compare_y);

  vector<Info> V;
  for (int i = 0; i < tmp_N; i++) {
    if (fabs(array[i].x - x) >= dist) continue;

    for (int j = 0; j < V.size(); j++) {
      double dx = array[i].x - V[V.size() - j - 1].x;
      double dy = array[i].y - V[V.size() - j - 1].y;
      if (dy >= dist) break;
      dist = min(dist, sqrt(dx * dx + dy * dy));
    }
    V.push_back(array[i]);
  }
  return dist;
}

int main() {
  scanf("%d", &N);
  info = (Info*)malloc(sizeof(Info) * N);

  for (int i = 0; i < N; i++) {
    scanf("%lf %lf", &info[i].x, &info[i].y);
  }

  sort(info, info + N);

  printf("%.7lf\n", closest_pair(info, N));
}
```