Inha University
Team : yoojin and dllrary

Coach : Prof. Jeong-seop Sim
Contestant : Min-kyum Kim, Hyeon-min Kim, Seung-hyun Jin

Contents

1. Graph

1. Dijkstra

```
typedef pair<ll, int> p;
const ll INF = 1e12;
auto dijkstra = [&](int start, vector<p> e[]) -> vector<ll>{
  vector<ll> v(N + 1);
  for (auto &i : v)i = INF;
  priority_queue<p, vector<p>, greater<p>> pq;
  v[start] = 0; pq.push({ 0, start });
  while (!pq.empty()) {
    ll cost, des, next, there;
    tie(cost, des) = pq.top(); pq.pop();
    for (auto k : e[des]) {
      tie(next, there) = k; next += cost;
      if (v[there] > next) {
        v[there] = next;
        pq.push({ next,there });
      }
    }
  }
  return v;
};
```

2. Floyd-Warshall

```
const int INF = 1000000000;
int N, M, dist[101][101];
int main() {
  cin >> N >> M;
  for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
      dist[i][j] = i == j ? 0 : INF;
    }
  }
  for (int i = 1; i <= M; i++) {
    int in1, in2, in3;
    cin >> in1 >> in2 >> in3;
    dist[in1][in2] = min(dist[in1][in2], in3);
  }
  for (int k = 1; k <= N; k++) {
    for (int i = 1; i <= N; i++) {
      for (int j = 1; j <= N; j++) {
        dist[i][j] = min(dist[i][j], dist[i][k] + dist[k][j]);
      }
    }
  }
  for (int i = 1; i <= N; i++) {
    for (int j = 1; j <= N; j++) {
      cout << dist[i][j] << ' ';
    }
    cout << '\n';
  }
}
```

3. Minimum Spanning Tree

```
typedef tuple<int, int, int> t;
int uf[10001];
int N, M, in1, in2, in3;
priority_queue<t, vector<t>, greater<t>> pq;
int find(int a) {
  if (uf[a] == a) return a;
  return uf[a] = find(uf[a]);
}
void merge(int a, int b) {
  a = find(a);
  b = find(b);
  if (a != b) uf[b] = a;
}
int main() {
  cin >> N >> M;
  for (int i = 0; i < M; i++) {
    cin >> in1 >> in2 >> in3;
    pq.push({ in3,in1,in2 }); //Distance, Start, End
  }
  for (int i = 1; i <= N; i++) uf[i] = i; //Union-Find Initialization
  ll cnt = 0, ans = 0;
  while (!pq.empty()) {
    if (cnt == N - 1) break; //N - 1 connections are enough for a spanning tree
    tie(cost, st, se) = pq.top();
```

```
    pq.pop();
    if (find(st) == find(se)) continue;
    merge(st, se);
    cnt++; ans += cost;
  }
  cout << ans;
}
```

4. Topological Sort

```
int main() {
  int N, M;
  int indegree[32001] = { 0 };
  vector<int> edge[32001];
  queue<int> q;
  vector<int> res;
  cin >> N >> M;
  for (int i = 1; i <= M; i++) {
    int in1, in2;
    cin >> in1 >> in2;
    indegree[in2]++;
    edge[in1].push_back(in2);
  }
  for (int i = 1; i <= N; i++) {
    if (indegree[i] == 0)q.push(i);
  }
  for (int i = 1; i <= N; i++) {
    if (q.empty()) {
      cout << "Cannot Sort";
      return 0;
    }
    int cur = q.front();
    q.pop();
    res.push_back(cur);
    for (int next : edge[cur]) {
      if (--indegree[next] == 0)q.push(next);
    }
  }
  for (int i = 0; i < res.size(); i++) {
    cout << res[i] << ' ';
  }
}
```

5. Strongly Connected Components

```
const int MAX = 10001;
int V, E;
vector<int> edge[MAX];
int sccCnt; //How many SCCs?
vector<vector<int>> SCC; //Stores Vertices of each SCCs
int dfscnt, dfsn[MAX], sccInd[MAX];
bool finished[MAX];
stack<int> s;
int makeSCC(int cur) { //return index cur's SCC number
  dfsn[cur] = ++dfscnt;
  s.push(cur);
  int res = dfsn[cur];
  for (int next : edge[cur]) {
    if (dfsn[next] == 0)res = min(res, makeSCC(next));
    else if (finished[next] == 0) res = min(res, dfsn[next]);
  }
  if (res == dfsn[cur]) {
    vector<int> curSCC;
    while (1) {
      int t = s.top();
      s.pop();
      curSCC.push_back(t);
      finished[t] = 1;
      sccInd[t] = sccCnt;
      if (t == cur) break;
    }
    //sort(curSCC.begin(), curSCC.end());
    SCC.push_back(curSCC);
    sccCnt++;
  }
  return res;
}
int main() {
  cin >> V >> E;
  for (int i = 0; i < E; i++) {
    int in1, in2;
    cin >> in1 >> in2;
```

```
    edge[in1].push_back(in2);
  }
  for (int i = 1; i <= V; i++) if (dfsn[i] == 0) makeSCC(i);
  //sort(SCC.begin(), SCC.end());
  //cout << sccCnt << '\n';
  for (auto& curSCC : SCC) {
    for (int cur : curSCC) {
      cout << cur << ' ';
    }
    cout << '\n';
  }
}
```

6. Maximum Flow

```
const int MAX = 800;
const ll INF = 2100000000;
int c[MAX][MAX], f[MAX][MAX], visited[MAX];
vector<int> edge[MAX];
/*
S에서 T로 가는 증가 경로 구하기(에드몬드 카프)
S : 시작점, T : 도착점
c[a][b] : a에서 b로 흐를 수 있는 최대 양 (Capacity)
f[a][b] : a에서 b로 흐른 실제 양 (Flow)
조건
용량 제한 : f[a][b] <= c[a][b]
유량의 대칭성 : f[a][b] == -f[b][a]
나오는 유량의 합 == 들어오는 유량의 합
*/
int maxFlow(int S, int T);
int bfs(int S, int T);
int maxFlow(int S, int T) {
  int result = 0;
  while (1) {
    int flow = bfs(S, T);
    if (!flow)break;
    result += flow;
  }
  return result;
}
int bfs(int S, int T) {
  memset(visited, -1, sizeof(visited));
  queue<int> q;
  q.push(S);
  while (!q.empty()) {
    int cur = q.front();
    q.pop();
    for (int next : edge[cur]) { //방문했는지, 용량이 남아 있는지 체크
      if (c[cur][next] - f[cur][next] <= 0)continue;
      if (visited[next] != -1)continue;
      q.push(next);
      visited[next] = cur; //cur->next 경로 기억
      if (next == T)break; //도착했을 경우 종료
    }
  }
  if (visited[T] == -1)return 0;
  int flow = INF;
  for (int i = T; i != S; i = visited[i]) { //최소 유량 탐색
    flow = min(flow, c[visited[i]][i] - f[visited[i]][i]);
  }
  for (int i = T; i != S; i = visited[i]) { //최소 유량 추가
    f[visited[i]][i] += flow;
    f[i][visited[i]] -= flow;
  }
  return flow;
}
int main() {
  int N; cin >> N;
  for (int i = 0; i < N; i++) {
    int in1, in2, in3;
    cin >> in1 >> in2 >> in3;
    c[in1][in2] += in3;
    c[in2][in1] += in3;
    edge[in1].push_back(in2);
    edge[in2].push_back(in1);
  }
  int S, T; cin >> S >> T;
  cout << maxFlow(S, T);
}
```

7. Bipartite Matching

```
const int MAX = 1001;
int N, M, A[MAX], B[MAX]; //A[i], B[i] : 각 정점이 매칭된 반대편 정점 인덱스
vector<int> edge[MAX];
bool visited[MAX];
//A그룹의 정점 a를 매칭시키는 데 성공하면 1 반환
//반대편이 매칭되지 않았으면 매칭시키고,
//이미 매칭되어 있다면 원래 매칭된 정점을 다른 정점과 매칭시킨다.
bool dfs(int a) {
  visited[a] = 1;
  for (int b : edge[a]) {
    if (B[b] == -1 || !visited[B[b]] && dfs(B[b])) {
      A[a] = b;
      B[b] = a;
      return 1;
    }
  }
  return 0;
}
int main() {
  cin >> N >> M;
  for (int i = 1; i <= N; i++) {
    int cnt;
    cin >> cnt;
    while (cnt--) {
      int in;
      cin >> in;
      edge[i].push_back(in);
    }
  }
  int match = 0;
  fill(A + 1, A + MAX - 1, -1);
  fill(B + 1, B + MAX - 1, -1);
  for (int i = 1; i <= N; i++) {
    if (A[i] == -1) {
      fill(visited + 1, visited + MAX - 1, 0);
      if (dfs(i))match++;
    }
  }
  cout << match;
}
```

8. Lowest Common Ancestor

```
const int MAX = 100000;
const int MAXDEP = 18;//more than log(2, 100000)
int N, M;
int par[MAX][MAXDEP];//par[i][k] : i의 2^k번째 부모
int dep[MAX];
vector<int> edge[MAX];
void makeTree(int cur) {
  for (int next : edge[cur]) {
    if (dep[next] == -1) {
      par[next][0] = cur;
      dep[next] = dep[cur] + 1;
      makeTree(next);
    }
  }
}
int getLCA(int in1, int in2) {
  in1--; in2--;
  if (dep[in1] < dep[in2])swap(in1, in2);
  int dif = dep[in1] - dep[in2];
  for (int j = 0; dif; j++) {
    if (dif % 2)in1 = par[in1][j];
    dif /= 2;
  }
  if (in1 != in2) {
    for (int j = MAXDEP - 1; j >= 0; j--) {
      if (par[in1][j] != -1 && par[in1][j] != par[in2][j]) {
        in1 = par[in1][j];
        in2 = par[in2][j];
      }
    }
    in1 = par[in1][0];
  }
  return in1 + 1;
}
int main() {
  cin >> N;
  for (int i = 0; i < N - 1; i++) {
    int in1, in2;
```

```
    cin >> in1 >> in2;
    in1--; in2--;
    edge[in1].push_back(in2);
    edge[in2].push_back(in1);
  }
  memset(par, -1, sizeof(par));
  fill(dep, dep + N, -1);
  dep[0] = 0;
  makeTree(0);
  for (int j = 0; j < MAXDEP - 1; j++) {
    for (int i = 1; i < N; i++) {
      if (par[i][j] != -1) {
        par[i][j + 1] = par[par[i][j]][j];
      }
    }
  }
  cin >> M;
  for (int i = 0; i < M; i++) {
    int in1, in2;
    cin >> in1 >> in2;
    cout << getLCA(in1, in2) << '\n';
  }
}
```

2. Data Structure

1. Segment Tree

```
struct SegTree {
  ll sz, st, dep;
  vector<ll> pw2, v;
  SegTree(ll size) {
    ll k = 1;
    for (int i = 0; i < 30; i++)pw2.push_back(k), k <<= 1;
    sz = size; v.resize(sz * 4, 0); dep = 1;
    while(1) {
      if (sz <= pw2[dep - 1])break;
      dep++;
    }
    st = pw2[dep - 1] - 1;
  }
  void init(ll val) {
    for (int i = 1; i < 4 * sz; i++)v[i] = val;
  }
  ll val(ll ind) {
      return v[st + ind];
  }
  void update(ll ind, ll val) {
    v[st + ind] = val; ind = (ind + 1) / 2;
        for (int i = dep - 1; i >= 1; i--) {
      ll cur = pw2[i - 1] - 1 + ind;
      v[cur] = v[cur * 2] + v[cur * 2 + 1];
      ind = (ind + 1) / 2;
    }
  }
    ll query(ll start, ll end) {
    ll ret = 0;
    start += st; end += st;
    while (start <= end) {
      if (start % 2 == 1)ret += v[start];
      if (end % 2 == 0)ret += v[end];
      start = (start + 1) / 2;
      end = (end - 1) / 2;
    }
      return ret;
  }
};
```

2. Fenwick Tree with Lazy Propagation

```
struct lazy {
  struct fenwick {
    ll sz;
    ll *arr;
    fenwick(int size) {
      sz = size + 1;
      arr = new ll[sz];
      fill(arr, arr + sz, 0);
    }
    void update(int i, ll x) {
      while (i <= sz) {
```

```
      arr[i] += x;
      i += i & -i;
    }
  }
  ll sum(int i) const {
    ll x = 0;
    while (i) {
      x += arr[i];
      i -= i & -i;
    }
    return x;
  }
};
fenwick *suma, *sumb;
lazy(int size) {
  suma = new fenwick(size);
  sumb = new fenwick(size);
}
void add(int L, int R, ll val) {
  suma->update(L, val);
  suma->update(R + 1, -val);
  sumb->update(L, (1LL - L)*val);
  sumb->update(R + 1, 1LL * R*val);
}
ll query(int L, int R) {
  ll ans = 0;
  ans += suma->sum(R)*R + sumb->sum(R);
  ans -= suma->sum(L - 1)*(L - 1) + sumb->sum(L - 1);
  return ans;
}
};
```

3. String

1. KMP

```
string src, tar;
vector<int> fail, KMP;
void getFail(string t) {
  fail.resize(t.length());
  int j = 0;
  for (int i = 1; i < t.length(); ++i) {
    while (j > 0 && t[i] != t[j]) j = fail[j - 1];
    if (t[i] == t[j]) fail[i] = ++j;
  }
}
void getKMP(string s, string t) {
  int sLen = s.length(), tLen = t.length(), j = 0;
  for (int i = 0; i < sLen; ++i) {
    while (j > 0 && s[i] != t[j]) j = fail[j - 1];
    if (s[i] == t[j]) {
      if (j == tLen - 1) {
        KMP.push_back(i - tLen + 1 + 1);
        j = fail[j];
      }
      else j++;
    }
  }
}
int main() {
  getline(cin, src); getline(cin, tar);
  getFail(tar);
  getKMP(src, tar);
  cout << KMP.size() << "\n";
  for (int i : KMP) {
    cout << i << " ";
  }
}
```

2. Rabin-Karp

```
const int MOD = 1000000007;
ll mod(ll n) { if (n >= 0) return n % MOD; return ((-n / MOD + 1) * MOD + n) % MOD; }
int L;
string src;
int rabinKarp(int len) {
  ll H = 0, power = 1;
  unordered_map<int, vector<int>> hashTable;
  for (int i = 0; i <= src.length() - len; ++i) {
    if (i == 0) {
      for (int j = 0; j < len; ++j) {
```

```
      H = mod(H + 1LL * src[len - 1 - j] * power);
      if (j < len - 1) power = mod(power * 127);
    }
  }
  else {
    H = mod(127 * (H - 1LL * src[i - 1] * power) + src[i + len - 1]);
  }
  if (hashTable[H].size() == 0) hashTable[H].push_back(i);
  else {
    for (int pos : hashTable[H]) {
      for (int p = 0; p < len; ++p) {
        if (src[pos + p] != src[i + p]) break;
        if (p == len - 1) return true;
      }
    }
    hashTable[H].push_back(i);
  }
}
return false;
}
```

3. Trie

```
struct Trie;
typedef pair<char, Trie*> pct;
struct Trie {
  vector<pct> child;
  bool isRet;
  Trie() {
    isRet = false;
  }
  ~Trie() {
    for (pct c : child) delete c.second;
  }
  void insert(const char* key) {
    int k = *key;
    if (k == '\0') {
      isRet = true;
      return;
    }
    for (pct c : child) {
      if (c.first == k) {
        c.second->insert(key + 1);
        return;
      }
    }
    child.push_back(pct(*key, new Trie));
    child.back().second->insert(key + 1);
  }
  bool demoFunc(const char* key) {
    int k = *key;
    if (isRet && k == '\0') return true;
    for (pct c : child) {
      if (c.first == k) return c.second->demoFunc(key + 1);
    }
    return false;
  }
};
int main() {
  Trie* root = new Trie;
  int n; cin >> n;
  for (int i = 0; i < n; ++i) {
    string input; cin >> input;
    root->insert(input.c_str());
  }
  int m; cin >> m;
  for (int i = 0; i < m; ++i) {
    string input; cin >> input;
    cout << (root->demoFunc(input.c_str()) ? "YES" : "NO") << "\n";
  }
  delete root;
}
```

4. Aho-Corasick

```
struct Trie {
  unordered_map<char, Trie*> child;
  Trie* fail;
  bool isRet = false;
  void push(string& s) {
    Trie* cur = this;
    for (char const& c : s) {
```

```
      if (cur->child.find(c) == cur->child.end())
        cur->child[c] = new Trie;
      cur = cur->child[c];
      cur->fail = this;
    }
    cur->isRet = true;
  }
  void build() {
    queue<Trie*> Q;
    for (auto const& p : child)
      if (p.second) Q.push(p.second);
    while (!Q.empty()) {
      Trie* cur = Q.front(); Q.pop();
      for (auto const& c : cur->child) {
        Trie* p = cur->fail;
        while (p != this && p->child.find(c.first) == p->child.end()) p = p->fail;
        p = p->child[c.first];
        if (!p) p = this;
        c.second->fail = p;
        if (p->isRet) c.second->isRet = true;
        Q.push(c.second);
      }
    }
  }
  bool ask(string& s) {
    Trie* p = this;
    for (char& c : s) {
      while (p != this && p->child.find(c) == p->child.end()) p = p->fail;
      p = p->child[c];
      if (!p) p = this;
      if (p->isRet) return 1;
    }
    return 0;
  }
};
int main() {
  Trie* root = new Trie;
  int N, Q;
  string input;
  cin >> N; while (N--) {
    cin >> input; root->push(input);
  }
  root->build();
  cin >> Q; while (Q--) {
    cin >> input; cout << (root->ask(input) ? "YES" : "NO") << "\n";
  }
}
```

5. Suffix Array and Longest Common Prefix Array

```
const char baseChar = '`'; // @ for uppercase ` for lowercase
const int baseSize = 27; // 27 : 59
vector<int> suffixArray, LCPArray;
void getLCP(vector<int>& sa, vector<int>& lcpa, string& s) {
  int i, j, k, u = 0, m = baseSize, sLen = s.length();
  sa.resize(sLen, 0); lcpa.resize(sLen, 0);
  vector<int> cnt(max(sLen, m), 0), x(sLen, 0), y(sLen, 0);
  for (i = 0; i < sLen; ++i) cnt[x[i] = s[i] - baseChar]++;
  for (i = 0; i < m; ++i) cnt[i] += (i == 0 ? 0 : cnt[i - 1]);
  for (i = sLen - 1; i >= 0; --i) sa[--cnt[x[i]]] = i;
  for (int len = 1, p = 1; p < sLen; len <<= 1, m = p + 1) {
    for (i = sLen - len - 1, p = 0; ++i < sLen;) y[p++] = i;
    for (i = 0; i < sLen; ++i) if (sa[i] >= len) y[p++] = sa[i] - len;
    for (i = 0; i < m; ++i) cnt[i] = 0;
    for (i = 0; i < sLen; ++i) cnt[x[y[i]]]++;
    for (i = 0; i < m; ++i) cnt[i] += (i == 0 ? 0 : cnt[i - 1]);
    for (i = sLen - 1; i >= 0; --i) sa[--cnt[x[y[i]]]] = y[i];
    swap(x, y); p = 1; x[sa[0]] = 1;
    for (i = 0; i < sLen - 1; ++i) x[sa[i + 1]] = sa[i] + len < sLen&& sa[i + 1] + len < sLen &&
y[sa[i]] == y[sa[i + 1]] && y[sa[i] + len] == y[sa[i + 1] + len] ? p : ++p;
  }
  vector<int> rank(sLen, 0);
  for (i = 0; i < sLen; i++) rank[sa[i]] = i;
  for (i = 0; i < sLen; ++i) if (k = rank[i]) {
    j = sa[k - 1];
    while (i + u < sLen && j + u < sLen && s[i + u] == s[j + u]) u++;
    lcpa[k] = u;
    u = u ? u - 1 : 0;
  }
}
int main() {
```

```
  string s; cin >> s;
  getLCP(suffixArray, LCPArray, s);
  cout << "SA: "; for (int i = 0; i < s.length(); ++i) cout << setw(2) << suffixArray[i] << " "; cout
<< "\n";
  cout << "LA: "; for (int i = 0; i < s.length(); ++i) cout << setw(2) << LCPArray[i] << " "; cout
<< "\n";
}
```

6. Manacher

```
vector<int> manacher(string& src) {
  int srcLen = src.size(); src.resize(srcLen * 2 + 1, '#');
  for (int i = srcLen - 1; i >= 0; --i) src[i * 2 + 1] = src[i], src[i] = '#';
  int c = 0, r = 0, len = src.size();
  vector<int> ret(len, 0);
  for (int i = 0; i < len; ++i) {
    int sym = 2 * c - i;
    if (i < r) ret[i] = min(r - i, ret[sym]);
    while (i - ret[i] > 0 && i + ret[i] - 1 < len && src[i - ret[i] - 1] == src[i + ret[i] + 1])
ret[i]++;
    if (ret[i] + i > r) r = ret[i] + i, c = i;
  }
  return ret;
}
int main() {
  string src; cin >> src;
  vector<int> pal = manacher(src);
  for (int i : pal) cout << i << ' ';
}
```

7. Z

```
vector<int> Z;
void getZ(string& src) {
  int l = 0, r = 0, len = src.length();
  Z.resize(len);
  for (int i = 1; i < len; i++) {
    Z[i] = max(0, min(Z[i - l], r - i));
    while (src[i + Z[i]] && src[Z[i]] == src[i + Z[i]]) Z[i]++;
    if (i + Z[i] > r) l = i, r = i + Z[i];
  }
  Z[0] = len; // Z[0] = 0;
}
int main() {
  string src; cin >> src;
  getZ(src);
  for (auto i : Z) cout << i << " ";
}
```

4. Math

1. Greatest Common Divisor, Least Common Multiple

```
ll gcd(ll a, ll b) { for (; b; a %= b, swap(a, b)); return a; }
ll lcm(ll a, ll b) { return a * b / gcd(a, b); }
```

2. Sieve of Eratosthenes

```
const ll MAX = 100001;
bool isprime[MAX];
void sieve(){
  fill(isprime, isprime + MAX, 1);
  isprime[1] = 0;
  for (int i = 2; i*i <= MAX; i++) {
    if (isprime[i]) {
      for (int j = i*i; j < MAX; j += i) {
        isprime[j] = 0;
      }
    }
  }
}
```

3. Binomial Coefficient

```
const ll MOD = 1000000007;
const ll MAX_NUM = 4000001;
ll f[MAX_NUM];
ll mypow(ll base, ll exp, ll MOD) {
  ll ans = 1;
  while (exp > 0) {
    if (exp % 2 != 0) {
      ans *= base;
      ans %= MOD;
    }
    base *= base;
    base %= MOD;
    exp /= 2;
```

```
  }
  return ans;
}
void nCrInit() {
  f[0] = 1; f[1] = 1;
  for (int i = 2; i <= MAX_NUM - 1; i++) {
    f[i] = f[i - 1] * i;
    f[i] %= MOD;
  }
}
ll nCr(int n, int r) {
  return (f[n] * mypow((f[r] * f[n - r]) % MOD, MOD - 2, MOD)) % MOD;
}
int main() {
  ll n, r;
  nCrInit();
  cin >> n >> r;
  cout << nCr(n, r) << "\n";
}
```

4. Matrix Exponential

```
ll a[3][3], a2[3][3], ans[3][3], temp[3][3], N;
void mult(ll a[3][3], ll b[3][3]) {
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      for (int k = 0; k < 3; k++) {
        temp[i][j] += a[i][k] * b[k][j];
        temp[i][j] %= 1000000007;
      }
    }
  }
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      a[i][j] = temp[i][j];
      temp[i][j] = 0;
    }
  }
}
int main() {
  cin >> N; N--;
  /*
  행렬 설명
  a : 가운데 거듭제곱되는 행렬
    초기화는 점화식으로 유도된 행렬로 한다.
  a2 : 거듭제곱된 결과가 저장되는 행렬
    초기화는 영행렬로 한다.
  temp : mult함수에서 사용되는 임시 행렬
  ans : 답이 저장되어 있는 행렬.
    초기화를 할 때는 ans[0][i]의 꼴로 베이스 값을 저장해놓는다.
  mult(ans, a2)를 하면 ans[0][i]에 원하는 값이 들어간다.
  */
  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 3; j++) {
      a[i][j] = 1;
      if (i == j)a2[i][i] = 1;
    }
  }
  a[0][0] = a[1][1] = 0;
  ans[0][0] = ans[0][1] = ans[0][2] = 1;
  while (N > 0) {
    if (N % 2 == 1) {
      mult(a2, a);
    }
    mult(a, a);
    N /= 2;
  }
  mult(ans, a2);
  cout << (ans[0][0] + ans[0][1] + ans[0][2]) % 1000000007;
}
```

5. Fast Fourier Transform

```
using ld = double;
using base = complex<ld>;
const ld pi = acos(-1);
void fft(vector<base> &A, bool f) {
  int k = A.size(), i, j, l, t;
  base w, x, y; ld th;
  for (i = 1, j = 0; i < k; i++) {
    for (l = k >> 1; j >= l; l >>= 1) j -= l;
    j += l; if(i < j) swap(A[i], A[j]);
```

```
  }
  for (i = 1; i < k; i <<= 1, t--) {
    th = (f ? -pi : pi) / i;
    w = base(cos(th), sin(th));
    for (j = 0; j < k; j += i + i) {
      for (l = 0; l < i; l++) {
        if (l & 2047) x *= w;
        else x = l ? base(cos(th * l), sin(th * l)) : 1;
        y = x * A[l | i | j];
        A[l | i | j] = A[l | j] - y;
        A[l | j] += y;
      }
    }
  }
  if (f) for (i = 0; i < k; i++) {
    A[i] /= k;
  }
}
vector<ll> mult(vector<ll> &X, vector<ll> &Y) { //return vector's size is |X| + |Y| - 1
  ll s, i, j;
  for (s = 1; s < X.size() + Y.size(); s <<= 1);
  vector<base> P(s), Q(s);
  vector<ll> Z(X.size() + Y.size() - 1);
  for (i = 0; i < X.size(); i++)
    P[i] = base(X[i] >> 12, X[i] & 4095);
  for (i = 0; i < Y.size(); i++)
    Q[i] = base(Y[i] >> 12, Y[i] & 4095);
  fft(P, 0); fft(Q, 0);
  for (i = 0; i + i <= s; i++) {
    j = i ? s - i : 0;
    base v1 = P[i] + conj(P[j]), v2 = conj(P[i]) - P[j];
    tie(P[i], Q[i], P[j], Q[j]) = make_tuple(
      v1 * Q[i], conj(v2) * conj(Q[j]),
      conj(v1) * Q[j], -v2 * conj(Q[i]));
  }
  fft(P, 1); fft(Q, 1);
  for(i = 0; i < Z.size(); i++) {
    Z[i] = (((ll)round(P[i].real()) << 23) + ((ll)round(Q[i].real()) >> 1)
      + ((ll)round(P[i].imag() + Q[i].imag()) << 11));
  }
  return Z;
}
```

6. Berlekamp-Massey

```
ll ipow(ll x, ll p) {
  ll ret = 1, piv = x;
  while(p){
    if(p & 1) ret = ret * piv % MOD;
    piv = piv * piv % MOD;
    p >>= 1;
  }
  return ret;
}
vector<ll> berlekamp_massey(vector<ll> x) {
  vector<ll> ls, cur;
  ll lf, ld;
  for(int i = 0; i < x.size(); i++) {
    ll t = 0;
    for(int j = 0; j < cur.size(); j++) {
      t = (t + 1ll * x[i - j - 1] * cur[j]) % MOD;
    }
    if((t - x[i]) % MOD == 0) continue;
    if(cur.empty()) {
      cur.resize(i + 1);
      lf = i;
      ld = (t - x[i]) % MOD;
      continue;
    }
    ll k = -(x[i] - t) * ipow(ld, MOD - 2) % MOD;
    vector<ll> c(i - lf - 1);
    c.push_back(k);
    for(auto &j : ls) c.push_back(-j * k % MOD);
    if(c.size() < cur.size()) c.resize(cur.size());
    for(int j = 0; j < cur.size(); j++) {
      c[j] = (c[j] + cur[j]) % MOD;
    }
    if(i - lf + (ll)ls.size() >= (ll)cur.size()) {
      tie(ls, lf, ld) = make_tuple(cur, i, (t - x[i]) % MOD);
    }
    cur = c;
```

```
  }
  for(auto &i : cur) i = (i % MOD + MOD) % MOD;
  return cur;
}
ll get_nth(vector<ll> rec, vector<ll> dp, ll n) {
  ll m = rec.size();
  vector<ll> s(m), t(m);
  s[0] = 1;
  if(m != 1) t[1] = 1;
  else t[0] = rec[0];
  auto mul = [&rec](vector<ll> v, vector<ll> w) {
    int m = v.size();
    vector<ll> t(2 * m);
    for(int j = 0; j < m; j++) {
      for(int k = 0; k < m; k++) {
        t[j + k] += 1ll * v[j] * w[k] % MOD;
        if(t[j + k] >= MOD) t[j + k] -= MOD;
      }
    }
    for(int j = 2 * m - 1; j >= m; j--) {
      for(int k = 1; k <= m; k++) {
        t[j - k] += 1ll * t[j] * rec[k - 1] % MOD;
        if(t[j - k] >= MOD) t[j - k] -= MOD;
      }
    }
    t.resize(m);
    return t;
  };
  while(n) {
    if(n & 1) s = mul(s, t);
    t = mul(t, t);
    n >>= 1;
  }
  ll ret = 0;
  for(int i = 0; i < m; i++) ret += 1ll * s[i] * dp[i] % MOD;
  return ret % MOD;
}
ll guess_nth_term(vector<ll> x, ll n){
  if(n < x.size()) return x[n];
  vector<ll> v = berlekamp_massey(x);
  if(v.empty()) return 0;
  return get_nth(v, x, n);
}
```

7. Extended Euclidean Algorithm

```
tuple<ll, ll, ll> euclid(ll x, ll y) {
  if (x < y) swap(x, y);
  if (y == 0) return { x,1,0 };
  ll g, x1, y1; tie(g, x1, y1) = euclid(y, x%y);
  return { g, y1, x1 - (x / y) * y1 };
}
ll inv = (get<2>(euclid(base, MOD)) + MOD) % MOD;
```

5. Geometry

1. CCW

```
//determines direction of 3 points
ll ccw(const point& A, const point& B, const point& C) {
  ll t = 1LL * (B.x - A.x)*(C.y - A.y) - 1LL * (B.y - A.y)*(C.x - A.x);
  if (t > 0) return 1; //Counter-Clockwise
  if (t == 0) return 0; //Line
  return -1; //Clockwise
}
```

2. Convex Hull

```
const int MAX = 100000;
struct point {
  int x, y;//실제 위치
  int p, q;//기준점으로부터 상대 위치
  point() : point(0, 0, 1, 0) {}
  point(int x1, int y1) :point(x1, y1, 1, 0) {}
  point(int x1, int y1, int p1, int q1) :x(x1), y(y1), p(p1), q(q1) {}
  bool operator<(const point& O) {
    if (1LL * q*O.p != 1LL * p*O.q)return 1LL * q*O.p < 1LL * p*O.q;
    if (y != O.y)return y < O.y;
    return x < O.x;
  }
};
point p[MAX];
int main() {
  int N;
```

```
  cin >> N;
  for (int i = 0; i < N; i++) {
    int in1, in2;
    cin >> in1 >> in2;
    p[i] = point(in1, in2);
  }
  sort(p, p + N);
  for (int i = 1; i < N; i++) {
    p[i].p = p[i].x - p[0].x;
    p[i].q = p[i].y - p[0].y;
  }
  sort(p + 1, p + N);//반시계 방향 정렬
  stack<int> cvh;
  cvh.push(0); cvh.push(1);
  int next = 2;
  while (next < N) {
    while (cvh.size() >= 2) {
      int fst, scd;
      fst = cvh.top();
      cvh.pop();
      scd = cvh.top();
      if (ccw(p[scd], p[fst], p[next]) > 0) {
        cvh.push(fst);
        break;
      }
    }
    cvh.push(next++);
  }
  cout << cvh.size();
}
```

3. Line Intersection

```
struct point {
  int x, y;
  point(int x, int y) {
    this->x = x;
    this->y = y;
  }
  bool operator< (point a) {
    if (a.x != this->x) {
      return a.x < this->x;
    }
    else {
      return a.y < this->y;
    }
  }
  bool operator<= (point a) {
    if (a.x == this->x && a.y == this->y) return 1;
    if (a.x != this->x) {
      return a.x < this->x;
    }
    else {
      return a.y < this->y;
    }
  }
};
bool isIntersect(pair<point, point> line1, pair<point, point> line2) {
  point p1 = line1.first;
  point p2 = line1.second;
  point p3 = line2.first;
  point p4 = line2.second;
  int l1 = ccw(p1, p2, p3) * ccw(p1, p2, p4);
  int l2 = ccw(p3, p4, p1) * ccw(p3, p4, p2);
  if (l1 == 0 && l2 == 0) {
    if (p2 < p1) swap(p1, p2);
    if (p4 < p3) swap(p3, p4);
    return (p3 <= p2 && p1 <= p4);
  }
  return (l1 <= 0 && l2 <= 0);
}
```

6. Others

1. Decimal in python

```
import decimal; D = decimal.Decimal
# 정밀도 설정
decimal.getcontext().prec = 28000
# Decimal 객체에 인자 넘겨주기
g = D(1/(10**200))
```

2. Fraction in python

```
Import fractions; F = fractions.Fraction
F(f.numerator, f.denominator)
```

3. Bit Operations

```
k & (1 << N) // return N-th bit of k
k | (1 << N) // make N-th bit of k to 1
k & ~(1 << N) // make N-th bit of k to 0
k & ~k // return least index of bit which has value 1
```

4. Coordinate Compression

```
ll getind(ll in){
  return lower_bound(a.begin(), a.end(), in) - a.begin();
}
sort(a.begin(), a.end());
a.erase(unique(a.begin(), a.end()), a.end());
```

5. Knuth Optimization

Optimization of O(N^3) DP to O(N^2)
Condition 0 : dp[i][j] is defined under 1 <= i <= j <= n
Condition 1 : dp[i][j] = min(dp[i][k] + dp[k + 1][j]) + cost[i][j]
Condition 2 : under a <= b <= c <= d, cost[a][c] + cost[b][d] <= cost[a][d] + cost[b][c]
Condition 3 : under a <= b <= c <= d, cost[b][c] <= cost[a][d]
Original O(N^3) DP :

```
for(i = 1; i <= n; i++){
  for(j = i; j < n; j++){
      for(k = j + 1; k < i; k++){
        dp[i][j] = min(dp[i][j] , dp[i][k] + dp[k + 1][j]);
        }
         dp[i][j] += C[i][j];
  }
}
```

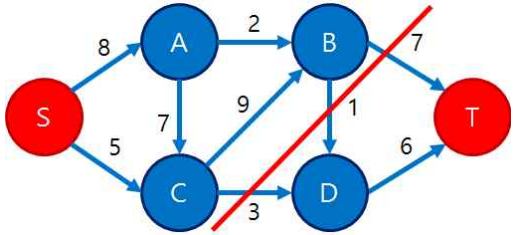
Optimized O(N^2) DP :

```
for(i=1; i<=n; i++){
  for(j=i; j<n; j++){
       dp[i][j] = MAX;
    for(k=A[i][j - 1]; k<=A[i + 1][j]; k++){ //상수 번 반복
      if(dp[i][j] > dp[i][k] + dp[k + 1][j]){
        dp[i][j] = dp[i][k] + dp[k + 1][j];
        A[i][j] = k;
      }
    }
    dp[i][j] += C[i][j];
  } //A[i][j] = minimum k to minimize dp[i][j]
} //A[i][j - 1] <= A[i][j] <= A[i + 1][j]
```

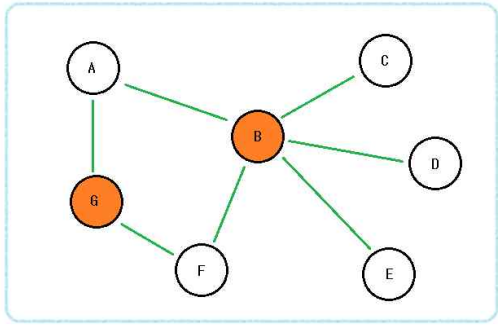
6. Some Prime Numbers

127, 131, 137, 139, 10007, 100003, 998244353, 1000000007, 1000000009

7. Tips



1. Minimum Cut of vertices is same as Maximum Flow of Vertices.



2. Minimum Vertex Cover is same as Maximum Matching.
3. Maximum Independent Set is Number of Vertices - Minimum Vertex Cover.
4. If node C is between node A and node B in a tree,
{LCA(A, C) == C && LCA(A, B) == LCA(C, B)} || {LCA(B, C) == C && LCA(A, B) == LCA(C, A)}
is true.
5. Derangement : dp[i] = (i - 1) * (dp[i - 1] + dp[i - 2]);
6. Erasing particular value in a vector

```
void erase(vector<int> &v, int val){
  for(auto it = v.begin(); it != v.end(); ){
    if(*it == val) it = v.erase(it);
    else it++;
  }
}
```

7. Partition of Number

Let dp[n][k] be the number of cases dividing n to k numbers.

a. dp[n][1] = dp[n][n] = 1

b. dp[n][k] = dp[n - k][1] + dp[n - k][2] + ... + dp[n - k][k]

c. dp[n][k] = dp[n - 1][k - 1] + dp[n - k][k]

8. Fermat's Little Theorem

If p is a prime number and a is not divisible by p, a^(p-1) % p == 1 is true.