Soongsil University
Team: Smoothes

ID : team018
Password : uyrspvqe

Coach : HYEONSUK NA
Contestant : Yun Won Nam (weonwon123@naver.com),
Myeoung Won Kim (auddnjs2008@naver.com),
Tae Won Noh (kstar7021@naver.com)

Algorithm Checklist

* 항상 최악의 경우 (Worst Case)를 생각하면서 알고리즘을 생각하자!
 - 1초에 1억 번(10^9 정도의 연산을 수행할 수 있음을 염두에 두자!
* int형인가? long long형인가?
  - int형의 범위 : -2^{31} ~ $2^{\wedge 31}$ (10^9까지 커버 가능)
  - long long형의 범위 : 10^{10} 이상일 경우 사용
* memset()을 쓴다면 반드시! <string.h> 또는 <cstring>을 추가할 것!
* 뒤집어서 생각하기 – 특히 누적 부분에서! 처음과 끝? Bottom up을 쓸지 Top Down을 쓸지?
* Greedy를 쓰기 위해서는 “증명”을 해야 한다!
* DP – 점화식을 어떻게 세울 것인가? 누적하는 방법? Divide & Conquer Optimization
* Binary Search (+with extra weight) * Divide & Conquer
* Stack overflow 주의! 지역변수로 큰 배열(int로 대략 1만개 이상)을 선언하면 Run-Error
a = b: a만 움직이기, b만 움직이기, 두 개 동시에 움직이기
답의 상한이 Resonable하게 작은가?
문제가 안풀릴 때 “당연하다고 생각한 것”을 의심하고, 말도 안되는 것을 한 번은 생각해보기.
Random Algorithm
LIS, LCS, Based on DP
Q Sqrt(N), ..
HLD, BCC, SCC
* float, double형 쓰는 것을 자제하기 (쓴다면 0 비교시 주의)
 - float 또는 double type은 정확히 0을 나타낼 수 없음, 따라서 epsilon(매우 작은 수) 이용

ex) floating point 연산
```
int main(void) {
        double a, b;
        double eps = 0.0000000001;
        scanf(“%lf%lf”, &a, &b);
        if (a == b)
                // BAD
        If (-eps < a-b && a-b < eps)
                // GOOD
        return 0; }
```

C++ STL Details

Algorithm Example

1. upper_bound , lower_bound
 - key 값을 초과하는 가장 첫 번째 원소의 위치를 찾아냄
 - key 값이 없으면 key 값보다 큰 가장 작은 정수 값의 위치를 찾아냄
 - 예제 : lower_bound(arr, arr + n, key) - arr + 1
 - 예제 : upper_bound(arr, arr + 10, 1) - arr + 1 (원소의 index 반환)
2. distance
 - 두 iterator 간의 거리를 구하는 함수
 - 예제 : distance(first, last) (first : v.begin(), last : v.end())
3. unique
 - 연속된 값의 첫 번째 원소만 남기는 함수(중복 덮어씀)
 - auto new_ent = unique(arr.begin(), arr.end());
4. next_permutation
 - 현재 순열의 다음 순서의 순열을 찾아주는 함수
 - bool next_permutation(first_it, last_it);
5. nth_element
 - 주어진 원소들의 n번째 위치한 원소를 n위치에 가져다놓음
 - void nth_element(first_it, first_it + n, last_it);
6. partial_sort
 - 주어진 범위의 지정된 위치까지만 정렬을 하는 함수
 - void partial_sort(first_it, first_it + k, last_it);
7. sort, stable_sort
 - stable_sort는 같은 값이 나올 경우 순서가 바뀌지 않음
 - stable_sort(first_it, last_it);
8. misc
 1) minmax – min max 동시에 반환
  - auto p = minmax(a, b);
  - p.first = minValue / p.second = maxValue
 2) max_Element 원소들 중 가장 큰 값이 있는 주소 반환
  - max_element(first.iter, last.iter);
 3) reverse 지정된 범위의 원소들의 순서를 뒤집음
  - reverse(first.iter, last.iter);
* getline
  - cin.getline(char*, 받을 개수); // ‘\n’ 나오면 자동으로 종료

Divide & conquer

Binary Search – 이분 탐색

```
void BinarySearch() {
	long long left = 0;
	long long right = LLONG_MAX;
	while (left <= right) {
		long long mid = (left + right) / 2;
		long long result = 0;
		for (int i = 0; i < k; i++) {
			result += line[i] / mid;
		}
		if (result >= n) {
			left = mid + 1;
			if (mid > maxValue)
				maxValue = mid;
		}
		else {
			right = mid - 1;
		}
	}
}
```

Greedy Algorithms

An activity-selection problem – 회의실 배정 문제
입력값 : 시작시간, 끝시간
출력값 : 최대 사용할 수 있는 회의 수

```
#include <iostream>
#include <algorithm>
using namespace std;
pair <int, int> input[100000];
// 회의가 일찍 끝나는 순으로 정렬
// 만약 회의가 끝나는 시간이 같으면,
// 회의의 시작 시간이 빠른 순으로 정렬
bool comp(pair <int, int> a, pair <int, int> b) {
	if (a.second < b.second) return true;
	else if (a.second == b.second) {
		return a.first < b.first;
	}
	else return false;
}
int main(void)
{
	ios_base::sync_with_stdio(false);
	cin.tie(NULL); cout.tie(NULL);
	int N;
	cin >> N;

	for (int i = 0; i < N; i++) {
		cin >> input[i].first >> input[i].second;
	}

	// 회의가 일찍 끝나는 순으로 정렬
	// 만약 회의가 끝나는 시간이 같으면,
	// 회의의 시작 시간이 빠른 순으로 정렬
	sort(input, input + N, comp);

	// 그러면 맨 앞의 data만 꺼내서 회의시키고
	// 중간에 회의가 있는 사람은 반려시키면 됨
	int temp = -1;
	int answer = 0;
	for (int i = 0; i < N; i++) {
		if (input[i].first < temp) continue;
		temp = input[i].second;
		answer++;
	}

	cout << answer;
	return 0;
}
```

An activity-selection problem – 철로 문제
입력값 : 사람 수, 정수 쌍들 (집 또는 사무실의 위치 두 개), 철로의 길이
출력값 : 집과 사무실 모두가 철로 안에 포함되는 사람들의 최대 수

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <algorithm>
#include <queue>
#define max(a,b) (a > b ? a : b)
#define min(a,b) (a < b ? a : b)
using namespace std;
class Line {
public:
	int left, right;
```

```
};
bool compare(Line a, Line b) {
	if (a.right != b.right)
		return a.right < b.right;
	return a.left < b.left;
}
int main(void) {
	int i, a, b, d, ans = -1, n;
	scanf("%d", &n);
	Line line[100001];
	for (i = 0; i < n; i++) {
		scanf("%d%d", &a, &b);
		line[i].left = min(a, b);
		line[i].right = max(a, b);
	}
	scanf("%d", &d);
	sort(line, line + n, compare);
	priority_queue<int> pq;
	// priority_queue는 큰 값을 위로 보낸다. -를 보내면 반대로 할 수 있다.
	for (i = 0; i < n; i++) {
		pq.push(-line[i].left);
		while (!pq.empty() && -pq.top() < line[i].right - d)
			pq.pop();
		ans = max(ans, (int)pq.size());
	}
	printf("%d\n", ans);
}
```

Dynamic Programming

Bottom-Up 방법을 이용한 DP 알고리즘 – 누적

```
for (int i = 0; i < n; i++) {
	for (int j = a[i]; j <= k; j++) {
		b[j] = min(b[j], b[j - a[i]] + 1);
	}
}
```

Top-Down 방법을 이용한 DP 알고리즘 – Memoization, 재귀

```
for (int to = idx + 1; to <= N; to++)
	if (dist[idx][to])
		result = max(result, dist[idx][to] + maxTaste(to, visit + 1));
```

Memoization Example : ACM Craft 문제

- 건물 순서 규칙이 주어지고, 해당 규칙에 맞춰서 건물을 지을 때 특정한 건물을 가장 빨리 지을 때까지 걸리는 최소시간을 알아내는 문제

* 이 문제는 Top-Down 방식으로 풀되, TotalTime 함수를 계속 호출하면서 destination 을 만들기 전에 동시에 만들 건물 중, 제일 시간이 오래 걸리는 건물을 다시 destination 으로 설정하여 들어간다. 이 시간들을 계속 누적하여 더하면 최소시간을 알아낼 수 있다.

```
int N;  //최대 1000
int cache[1001];
int delay[1001];		// 건물을 짓는데 걸리는 시간
int order[1001][1001];	// 건물 짓는조건

int TotalTime(int destination) {
	int &result = cache[destination];
	if (result != -1)  return result;
	int time = 0;
	for (int i = 1; i <= N; i++) {
		if (order[i][destination])
			// destination을 만들기 전에 동시에 만들 건물 중
			// 제일 시간이 오래 걸리는 건물
			time = max(time, TotalTime(i));
	}
	return result = time + delay[destination];
}

int main(void) {
	int T;
	cin >> T;
	for (int i = 0; i < T; i++) {
		int K, D, X, Y;
		cin >> N >> K;
		for (int j = 1; j <= N; j++)
		    cin >> delay[j];
		memset(cache, -1, sizeof(cache));
		memset(order, 0, sizeof(order));
		for (int i = 0; i < K; i++) {
			cin >> X >> Y;
			order[X][Y] = 1;
		}
		cin >> D;
		cout << TotalTime(D) << "\n";
	}
	return 0;
}
```

Chained Matrix Multiplication

```
#include<iostream>
#include<cstring>
using namespace std;
int a[501][2];
int DP[501][501];
int solve(int i, int j) {
	if (i == j) return 0;
	if (i + 1 == j) {
		return a[i][0] * a[i][1] * a[j][1];
	}
	int &ret = DP[i][j];
	if (ret != -1) return DP[i][j];
	for (int k = i; k < j; k++) {
	int temp = solve(i, k) + solve(k+1, j) + a[i][0] * a[k][1] * a[j][1];
		if (ret == -1 || ret > temp) {
			ret = temp;
		}
	}
	return ret;
}
int main() {
	ios_base::sync_with_stdio(false);
	cin.tie(NULL);	cout.tie(NULL);
	int n;	cin >> n;
	memset(DP, -1, sizeof(DP));
	for (int i = 1; i <= n; i++) {
		cin >> a[i][0] >> a[i][1];
	}
	cout << solve(1, n);
}
```

Palindrome 문제

세준이는 어떤 문자열을 팰린드롬으로 분할하려고 한다.
예를 들어, ABACABA를 팰린드롬으로 분할하면,
{A, B, A, C, A, B, A}, {A, BACAB, A}, {ABA, C, ABA}, {ABACABA}등이 있다.
분할의 개수의 최솟값을 출력하는 프로그램을 작성하시오.

```
bool DP[2500][2500] = { false, };
int DP_min[2500];
//1. 입력을 받는다.
string input;
cin >> input;
int N = input.size();  // 문자열의 길이
//2.  DP 배열을 채운다.
for (int i = 0; i < N; i++) // 같을때 초기화
	DP[i][i] = true;
for (int i = 0; i < N-1; i++)  //두자릿수때 초기화
	if (input[i] == input[i + 1])
		DP[i][i + 1] = true;
for (int i = 2; i <= N - 1; i++) // 3자릿수이상일 때
	for (int j = 0; j < N - i; j++) {
		if ((input[j] == input[j + i]) && DP[j + 1][j + i - 1])
			DP[j][i+j] = true;
	}
for (int a = 2; a < N; a++)
	DP_min[a] = 99999;
//3. dp_min을 채우자
DP_min[0] = 1;
DP_min[1] = 1 + (input[0] != input[1]);

if (DP[0][N - 1]) { // 만일 아예 처음부터 팰린드롬 수일 경우
	cout << 1;	return 0;
}
for (int i = 2; i <= N - 1; i++) {
	if (DP[0][i]) { // 만일 아예 처음부터 팰린드롬 수일 경우
		DP_min[i] = 1;   continue;
	}
	for (int j = i; j >= 1; j--) {
		if (DP[j][i]) {
			if ( (DP_min[i] > DP_min[j - 1] + 1)) {
			DP_min[i] = DP_min[j - 1] + 1;
			}
		}
		else {
			if (DP_min[i] > DP_min[j - 1] + i - j + 1) {
			DP_min[i] = DP_min[j - 1] + i - j + 1;

			}	}	}	} cout << DP_min[N-1];
```

String Algorithms

KMP Algorithm

```
vector<int> kmp(string s, string p) {
	vector<int> ans;
	auto pi = getPi(p);
```

```
        int n = (int)s.size(), m = (int)p.size(), j =0;
        for (int i = 0 ; i < n ; i++) {
                while(j>0 && s[i] != p[j])
                j = pi[j-1];
                if(s[i] == p[j]) {
                        if(j==m-1) {
                                ans.push_back(i-m+1);
                                j = pi[j];
                        } else {
                                j++;
                        }
                }
        }
        return ans;
}
vector<int> getPi(string p) {
        int m = (int)p.size(), j=0;
        vector<int> pi(m, 0);
        for (int i = 1; i< m ; i++) {
                while(j > 0 && p[i] != p[j])
                 j = pi[j-1];
                if(p[i] == p[j])
                pi[i] = ++j;
        }
        return pi;
}
```

Longest Common Subsequence (LCS)
최장 공통 부분 수열 : 부분 수열 중에서 길이가 가장 긴 수열을 구하는 문제

```
string str1, str2;
int lcs[1001][1001];
int main() {
        string tmp1, tmp2;
        cin >> tmp1 >> tmp2;
        // LCS 알고리즘을 위해 앞에 '0'을 붙여준다.
        str1 = '0' + tmp1;
        str2 = '0' + tmp2;
        int len1 = str1.size();
        int len2 = str2.size();
        for (int i = 0; i < len1; i++) {
                for (int j = 0; j < len2; j++) {
                        if (i == 0 || j == 0) {
                                lcs[i][j] = 0;
                                continue;
                        }
                        // 현재 비교하는 값이 서로 같다면, lcs는 + 1
                        if (str1[i] == str2[j])
                                lcs[i][j] = lcs[i - 1][j - 1] + 1;
                        // 서로 다르다면 LCS의 값을 왼쪽 혹은 위에서 가져온다.
                        else {
                                if (lcs[i - 1][j] > lcs[i][j - 1])
                                        lcs[i][j] = lcs[i - 1][j];
                                else
                                        lcs[i][j] = lcs[i][j - 1];
                        }
                }
        }

/*
    // 검증 코드
    for (int i = 0; i < len1; i++)
    {
        for (int j = 0; j < len2; j++)
            cout << lcs[i][j] << " ";
        cout << endl;
    }
*/
         cout << lcs[len1-1][len2-1] << endl;
         return 0;
}
```

Longest Incrasing Subsequence (LIS)
어떤 수열에서 가장 긴 증가하는 부분 수열을 찾는 문제

```
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;

int main(void)
{
         ios_base::sync_with_stdio(false);
         cin.tie(NULL);   cout.tie(NULL);

         int N;  int num = 0;
         vector<int>dp;
         dp.push_back(-1);

         cin >> N;
         for (int i = 0; i < N; i++) {
                  cin >> num;
```

```
                if (num > dp.back()) {
                        if (dp.back() == -1)
                                dp.clear();
                        dp.push_back(num);
                }
                else {
                        auto pos = lower_bound(dp.begin(), dp.end(), num);
                        *pos = num;
                }
        }
        cout << dp.size() << "\n";
        return 0;
}
```

String Algorithms Example : 문자열 뽑기 문제

- **babbbaaabb -> baaaabb -> bbb -> 빈 문자열**
- **babbbaaabb -> babbbaaa -> baaaa -> b (빈 문자열로 바꿀 수 없음)**
- **aabbaabb -> aaaabb -> bb -> 빈 문자열**
- **abab -> abb -> a (빈 문자열로 바꿀 수 없음)**
- **입력된 문자열을 빈 문자열로 바꿀 수 있으면 1, 없으면 0을 출력하는 문제**

* 예외처리에 각별히 주의하여 코드를 작성해야 한다.

```
---------------------------------------------------------------------------
using namespace std;
#define inf 1000000000
#define eps 1e-9
int solve(string &str) {
        if (!str.size()) return 1;
        for (int i = 0; i < str.size(); i++) {
                int sz = 0;
                for (int j = i; j < str.size(); j++) {
                        if (str[i] == str[j]) sz++; else break;
                }
                if (sz > 1) {
                        string tmpGp = str;
                tmpGp.erase(tmpGp.begin() + i, tmpGp.begin() + i + sz);
                        if (solve(tmpGp)) return 1;
                }
                i += sz - 1;
        }
        return 0;
}
int main() {
        int tc;
        cin >> tc;
        while (tc--) {
                string tmp;
                cin >> tmp;
                cout << solve(tmp) << endl;
        }
        return 0;
}
```

Tree Details

Union-Find – 이 경우 트리는 배열로 만들게 된다.

```
// 배열 하나만으로 해도 가능하며 이 경우 root는 첫 번째
// 자신의 부모가 누구인지를 나타내는 up-tree 배열
int union_find[3000];

int find_set(int x) {
        if (union_find[x] == x) return x;
        find_set(union_find[x]);
}

void Union(int x, int y) {
        int a = find_set(x);
        int b = find_set(y);
        if (a < b) union_find[b] = a;
        else if (a > b) union_find[a] = b;
}
```

트리의 구성 – vector 사용

```
#include <iostream>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;

class Tree{
public:
    int N; // 정점의 개수
    vector<int> parent; // 부모 노드
    vector<vector<int>> children; // 자식 노드 리스트

    // 생성자
```

```
    Tree(): N(0){}
    Tree(int n): N(n){
        parent.resize(N, -1);
        children.resize(N);
    }
     // 각 정점마다 부모, 자식들을 보여주는 함수
    void print(){
        for(int i=0; i<N; i++){
            cout << "Node " << i << ": parent(";
            if(parent[i] != -1) cout << parent[i];
            else cout << "-";
            cout << "), children(";
            for(int j=0; j<children[i].size(); j++){
                cout << children[i][j];
                if(j < children[i].size()-1) cout << ", ";
            }
            cout << ")" << endl;
        }
    }
void setChildren(int p, int l, int r){
        if(l != -1) parent[l] = p;
        if(r != -1) parent[r] = p;
        lc[p] = l;
        rc[p] = r;
    }
     void preorder(int root){ // 전위 순회
        cout << root << ' ';
        if(lc[root] != -1) preorder(lc[root]);
        if(rc[root] != -1) preorder(rc[root]);
    }
    void inorder(int root){         // 중위 순회
        if(lc[root] != -1) inorder(lc[root]);
        cout << root << ' ';
        if(rc[root] != -1) inorder(rc[root]);
    }
    void postorder(int root){ // 후위 순회
        if(lc[root] != -1) postorder(lc[root]);
        if(rc[root] != -1) postorder(rc[root]);
        cout << root << ' ';
    }
    void levelorder(int root){
        queue<int> Q;
        Q.push(root);
        while(!Q.empty()){
            int curr = Q.front();
            Q.pop();
            cout << curr << ' ';
            if(lc[curr] != -1) Q.push(lc[curr]);
            if(rc[curr] != -1) Q.push(rc[curr]);
        }
    }
};
class Graph{
public:
    int N; // 정점의 개수
    vector<vector<int>> adj; // 인접 리스트
     // 생성자
    Graph(): N(0){}
    Graph(int n): N(n){ adj.resize(N); }
     // 간선 추가 함수
    void addEdge(int u, int v){
        adj[u].push_back(v);
        adj[v].push_back(u);
    }
     // 모든 리스트의 인접한 정점 번호 정렬
    void sortList(){
        for(int i=0; i<N; i++)
            sort(adj[i].begin(), adj[i].end());
    }
     // BFS로 트리 만들기(그래프는 완전 그래프라 가정)
    Tree makeTree(int root){
        Tree T(N);
        vector<bool> visited(N, false); // 방문 여부를 저장하는 배열
        queue<int> Q;
         visited[root] = true;
        Q.push(root);
        while(!Q.empty()){
            int curr = Q.front();
            Q.pop();
            for(int next: adj[curr]){
                if(!visited[next]){
                    visited[next] = true;
                    Q.push(next);
                    T.parent[next] = curr;
                    T.children[curr].push_back(next);
                }
            }
        }
        return T;
    }
};
```

```
int main(){
    Graph G(9);
    G.addEdge(0, 1);    G.addEdge(0, 2);    G.addEdge(1, 3);
    G.addEdge(1, 5);    G.addEdge(3, 4);    G.addEdge(4, 5);
    G.addEdge(2, 6);    G.addEdge(2, 8);    G.addEdge(6, 7);    G.addEdge(6, 8);
    G.sortList();
    Tree T = G.makeTree(0);
    T.print();
}
int getHeight(int root){
        int result = 1;
        for(int c: children[root])
            result = max(result, getHeight(c) + 1);
        return result;
}
```

Diameter of the tree (트리의 지름)

```
vector < vector <pair<int, int>>> Graph;
int n, d[10010];
bool check[10010];

int dfs(int x) {
         check[x] = true;
         int ret = x;
         d[x] = 0;
         for (int i = 0; i < Graph[x].size(); i++) {
                  int next = Graph[x][i].first;
                  int cost = Graph[x][i].second;
                  if (check[next]) continue
                  int ret_next = dfs(next);
                  if (d[x] < d[next] + cost) {
                           d[x] = d[next] + cost;
                           ret = ret_next;
                  }
         }
         return ret;
}
int main() {
         cin >> n;
         Graph.resize(n);
         for (int i = 0; i < n - 1; i++) {
                  int u, v, w;
                  cin >> u >> v >> w;
                  u--; v--; // index를 0부터 시작하는 경우에 해당, 문제보고 판단하기
                  Graph[u].push_back( {v, w} );
                  Graph[v].push_back( {u, w} );
         }
         int s = dfs(0);
         memset(check, 0, sizeof(check));
         memset(d, 0, sizeof(d));
         int e = dfs(s);
         cout << d[s];
}
```

이진 트리 문제 (Parent = i, left = i * 2, right = i * 2 + 1)
tree : 인덱스 트리
path : i번째 노드부터 그 리프들까지의 최장경로
sum : i번째 노드부터 그 리프들까지 최장경로로 바꿨을 때의 거리의 합

```
#include <iostream>
#include <string>
#define SIZE (1<<21)
#define MAX(a,b) (a > b ? a : b)
using namespace std;

int main(void) {
         int i, a, k, size, left, right;
         cin >> k;
         size = 1 << (k + 1);
         int tree[SIZE] = { 0 }, path[SIZE] = { 0 }, sum[SIZE] = { 0 };
         for (i = 2; i < size; i++) {
                  cin >> a;
                  tree[i] = a;
         }

         for (i = 1 << k; --i > 0; ) {
                  left = i << 1; right = left + 1;
path[i] = MAX(path[left] + tree[left], path[right] + tree[right]);
sum[i] = sum[left] + sum[right] + (path[i] - path[left]) + (path[i] - path[right]);
         }
         cout << sum[1];
         return 0;
}
```

Graph Algorithms

DFS, BFS (1)

```
bool visit[MAX_N];
Vector<int>adj[MAX_N];
queue<int>Q;
int N, M, V;
void DFS(int x) {
         visit[x] = true;
         cout << x << " ";
         for (int y : adj[x]) {
                  if (!visit[y])
                           DFS(y);
         }
```

```
}

void BFS(int sx) {
	Q.push(sx);
	visit[sx] = true;
	while (!Q.empty()) {
		int x = Q.front();
		Q.pop();
		cout << x << " ";
		for (int y : adj[x]) {
			if (!visit[y]) {
				Q.push(y);
				visit[y] = true;
			}
		}
	}
}
```

DFS, BFS (2) – 미로 탈출 문제

```
int map[1000][1000];
int m, n, findPath = 0;
int dx[4] = {0,0,-1,1};    // 좌표를 나타내는 테크닉
int dy[4] = {1,-1,0,0};    // 좌표를 나타내는 테크닉

// 테두리 검사하기
int isInRanGe(int x, int y) {
	return (0 <= x && x < m) && (0 <= y && y < n);
}

void dfs(int a, int b) {
	map[a][b] = 1;
	if (a == m - 1) {
		findPath = 1;
		return;
	}
	for (int i = 0; i < 4; i++) {
		int ax = a + dx[i];
		int ay = b + dy[i];
		if (isInRanGe(ax, ay) && !map[ax][ay])
		    dfs(ax, ay);
	}
}

int main(void) {
	string tmp[1000];
	cin >> m >> n;
	memset(map, 0, sizeof(map));        // 초기화는 항상 중요!
	for (int i = 0; i < m; i++) {
		cin >> tmp[i];
		for (int j = 0; j < n; j++)
		map[i][j] = tmp[i][j] - '0';
	}
	for (int i = 0; i < n; i++) {
		if (!findPath && !map[0][i])
		dfs(0, i);
	}
	// 경로가 있다면 YES 그렇지 않다면 NO
	if (findPath == 1) cout << "YES";
	else cout << "NO";
	return 0;
}
```

DFS, BFS (3) – 스택과 큐를 사용하여 미로찾는 문제 풀기

```
void dfs_stack(int x, int y) {
	// 이용할 스택, (x,y) -> (행, 열)
	stack< pair<int,int> > s;

	// pair를 만들어서 stack에 넣는다.
	s.push(make_pair(x,y));

	// 처음 x,y를 방문 했기 때문에 초기화
	visited[x][y] = true;
	groups[group_id]++;

	while(!s.empty()) {
		// 스택의 top 원소 꺼내기
		x = s.top().first;
		y = s.top().second;
		s.pop();

		// 해당 위치의 주변을 확인
		for (int i = 0; i < 4; i++) {
			int nx = x + dx[i];
			int ny = y + dy[i];
			// 지도를 벗어나지 않고
			if(0 <= nx && nx < n && 0 <= ny && ny < n) {
			// 집이면서 방문하지 않았다면 -> 방문
			if(map[nx][ny] == 1 && visited[nx][ny] == false) {
				visited[nx][ny]=true;

				// 해당 단지의 집의 수를 증가시킴
				groups[group_id]++;

				// push하는 경우이므로 현재 위치도 넣어준다.
				s.push(make_pair(x,y));

				// 스택에 현재 nx,ny도 푸시
```

```
                                s.push(make_pair(nx,ny));
                        }
                }
        }
}}

void bfs(int x, int y) {
        // 이용할 큐, (x,y) -> (행, 열)
        queue< pair<int,int> > q;
        // pair를 만들어서 queue에 넣는다.
        q.push(make_pair(x,y));
        // 처음 x,y를 방문 했기 때문에 초기화
        visited[x][y] = true;
        groups[group_id]++;
        while(!q.empty()) {
                // 큐의 현재 원소를 꺼내기
                x = q.front().first;
                y = q.front().second;
                q.pop();
                // 해당 위치의 주변을 확인
                for (int i = 0; i < 4; i++) {
                        int nx = x + dx[i];
                        int ny = y + dy[i];
                        // 지도를 벗어나지 않고
                        if(0 <= nx && nx < n && 0 <= ny && ny < n) {
                        // 집이면서 방문하지 않았다면 -> 방문
                        if(map[nx][ny] == 1 && visited[nx][ny] == false) {
                                visited[nx][ny]=true;
                                // 해당 단지의 집의 수를 증가시킴
                                groups[group_id]++;
                                // 큐에 현재 nx,ny를 추가
                                q.push(make_pair(nx,ny));
                        }       }       }       }       }
```

DFS, BFS Example : Term Project 문제

예를 들어, 한 반에 7명의 학생이 있다고 하면, 학생들을 1번부터 7번으로 표현할 때, 선택의 결과가 다음과 같다고 하자.

1	2	3	4	5	6	7
3	1	3	7	3	4	6

위의 결과를 통해 (3)과 (4, 7, 6)이 팀을 이룰 수 있고, 1, 2, 5는 어느 팀에도 속하지 않는다. 주어진 선택의 결과를 보고 어느 프로젝트 팀에도 속하지 않는 학생들의 수를 계산하는 프로그램을 작성하는 문제이다.

```
* 사이클(cycle)을 찾는 문제이다. 중복 입력이 주어질 수 있으니 그 점에 주의
* 팀을 이룰 수 있는 경우는 2가지이다.
  1. 혼자 팀을 이루는 경우
  2. 여러 명이 팀을 이루는 경우
--------------------------------------------------------------------------------
const int MAX = 100000 + 1;
int N, cnt;
int want[MAX];
bool visited[MAX];
bool done[MAX]; // 방문이 끝났는지 여부를 확인
void DFS(int nodeNum) {
        visited[nodeNum] = true;
        int next = want[nodeNum];
        if (!visited[next]) DFS(next);
        //이미 방문했지만 방문이 끝난 노드가 아니라면 싸이클
        else if (!done[next]) {
                // 팀원을 모두 센다
                for (int i = next; i != nodeNum; i = want[i])
                cnt++;
                cnt++;   // 자기 자신을 센다.
        }
        done[nodeNum] = true;     // 더이상 해당 노드를 방문 할 일이 없다.
}

int main(void) {
        int T;
        cin >> T;
        while (T--) {
                memset(visited, false, sizeof(visited));
                memset(done, false, sizeof(done));
                cin >> N;
                        for (int j = 1; j <= N; j++)
                        cin >> want[j];
                        cnt = 0;
                        for (int j = 1; j <= N; j++) {
```

```
                                if (!visited[j]) DFS(j);
                        }
                cout << N - cnt << "\n";
        }
        return 0;
}
```

Djikstra Algorithm

```
// dist[] 배열 startnode 는 0, 나머지는 INF 로 초기화
typedef struct Node {
        end, val
};

vector<Node>Edge_arr[MAX_V];
int dist[MAX_v] = {0};

// 우선순위 큐를 사용하는 dijkstra() -> #include <queue> 필요!
void dijkstra() {
        priority_queue<pair<int,int>>pq;
        pq.push( {0,start_node_num} );

        while(!pq.empty()) {
                int now_node = pq.top().second;
                int cost = -1*pq.top().first;
                pq.pop();
                for (int k=0; k<EDGE_arr[now_node].size(); k++) {
                        int new_val =dist[now_node] +EDGE_arr[now_node][k].val;
                        int before val =dist[Edge_arr[now_node][k].end];

                        if(new_val<before_val) {
                                dist[EDGE_arr[now_node][k].end]=new_val;
                                pq.push({-1*new_val, EDGE_arr[now_node][k].end});
                        }
                }
        }
}
```

Djikstra Algorithm Example : 거의 최단경로 문제
- 최단경로에 포함되어 있는 간선은 제외하고 난 그래프에서의 최단경로 구하기

1. 처음 한 번 다익스트라를 돌려 모든 최단경로를 찾는다..
2. 최단경로에 속한 경로들을 모두 지운다.
3. 다시 한번 다익스트라를 돌려 거의 최단 경로를 찾는다.

```
using namespace std;
const int MAX = 500;
const int INF = 987654321;
int N, M;
vector<pair<int, int>>graph[MAX];
vector<pair<int, int>>trace[MAX];
bool visited[MAX][MAX];
vector<int> dijkstra(int start, int vertex) {
        vector<int>distance(vertex, INF);
        distance[start] = 0;
        priority_queue<pair<int, int>>pq;

        // cost, vertex
        pq.push(make_pair(0, start));

        while (!pq.empty()) {
                int cost = -1 * pq.top().first;
                int curVertex = pq.top().second;
                pq.pop();

                if (distance[curVertex] < cost) continue;

        for (int i = 0; i < graph[curVertex].size(); i++) {
                int neighbor = graph[curVertex][i].first;
                int neighborDistance = cost + graph[curVertex][i].second;
                if (graph[curVertex][i].second == -1) continue;
                if (distance[neighbor] > neighborDistance) {
                        distance[neighbor] = neighborDistance;
                        pq.push(make_pair(-1*neighborDistance, neighbor));
                        trace[neighbor].clear();

        trace[neighbor].push_back(make_pair(curVertex, neighborDistance));
                }
                else if (distance[neighbor] == neighborDistance) {
        trace[neighbor].push_back(make_pair(curVertex, neighborDistance));
                }
        }
        return distance;
}

void BFS(int destination) {
        queue<int>q;
        q.push(destination);
        while (!q.empty()) {
                int curVertex = q.front();
                q.pop();
                for (int i = 0; i < trace[curVertex].size(); i++) {
                        int neighbor = trace[curVertex][i].first;
                        for (int i = 0; i < graph[neighbor].size(); i++)
```

```
                                if (graph[neighbor][i].first == curVertex)
                                        graph[neighbor][i].second = -1;
                        q.push(neighbor);
                }
        }
}

int main(void) {
        while (1) {
                memset(visited, false, sizeof(visited));
                memset(trace, 0, sizeof(trace));
                for (int i = 0; i < MAX; i++)
                        graph[i].clear();
                cin >> N >> M;
                if (N == 0 && M == 0)
                        break;
                int S, D;
                cin >> S >> D;
                for (int i = 0; i < M; i++) {
                        int source, destination, cost;
                        cin >> source >> destination >> cost;
                graph[source].push_back(make_pair(destination, cost));
                }
                dijkstra(S, N);
                BFS(D);
                vector<int> result = dijkstra(S, N);
                if (result[D] == INF) cout << -1 << "\n";
                else cout << result[D] << "\n";
                }
        return 0;
}
```

Djikstra Algorithm Example : 미확인 도착지 문제
- 최단경로 중 어떤 두 정점을 방문해야 하거나, 한 간선을 들러야 하는 경우

1. 다익스트라를 세 번 돌려서 해결한다.
   즉, start -> n1 -> n2 -> end일 경우
   (start -> n1), (n1 -> n2), (n2 -> end) 각각의 최단경로를 구하는 것이다..

2. 다익스트라를 한 번만 돌려서 해결한다.
   가중치가 모두 정수일 때, 모든 간선의 가중치들을 *2 해준 후 n1 -> n2 사이의 가중치를 -1 해줘서 홀수로 만든다.
   이후 다익스트라를 한 번 해 주면 후보지의 가중치가 홀수일 때 무조건 n1 -> n2를 지난 것으로 손쉽게 확인할 수 있다.
   -1을 해줄 경우 원래 간선 기준으로 n1 -> n2를 지나던 그렇지 않던 최단경로가 같을 경우, 무조건 n1 -> n2 경로를 지나게 할 수 있어, 중복경로를 걸러낼 수 있다.

```
#define INF 200000000
struct Vertex {
        int dist = INF, idx;
        vector<int>post;
        vector<pair<int, int>>next;
        bool operator <  (const Vertex &v)const {
                return dist > v.dist;
        }
};

int main(void) {
        int tc;
        cin >> tc;
        while (tc--) {
                int n, m, t, s, g, h;
                cin >> n >> m >> t >> s >> g >> h;
                Vertex vertex[2001];
                for (int i = 1; i <= n; i++) {
                        vertex[i].idx = i;
                }
                vector<int>target;
                // 후보 목적지 저장
                while (m--) {
                        int a, b, d;
                        cin >> a >> b >> d;
                        d *= 2;
                        if ((a == g && b == h) || (a == h && b == g))
                                d -= 1;
                        // 양방향 그래프
                        vertex[a].next.push_back(make_pair(d, b));
                        vertex[b].next.push_back(make_pair(d, a));

                }

        while (t--) {
                int ta;
                cin >> ta;
```

```
                target.push_back(ta);
        }

        priority_queue<Vertex> pq;
        vertex[s].dist = 0;
        pq.push(vertex[s]);

        while (!pq.empty()) {
                Vertex curVertex = pq.top();
                pq.pop();
                for (pair<int, int>pa : curVertex.next) {
                        int w = pa.first;
                        // 가중치
                        int nIdx = pa.second;
                        if (vertex[nIdx].dist > curVertex.dist + w) {
                                vertex[nIdx].dist = curVertex.dist + w;
                                pq.push(vertex[nIdx]);
                                }
                        }
                }
                sort(target.begin(), target.end());
                for (int tIdx : target) {
                        if (vertex[tIdx].dist % 2 != 0)
                                cout << tIdx << " ";
                }
                cout << "\n";
        }
        return 0;
}
```

Shortest Path Faster Algorithm

```
#define MAX 987654321
typedef pair<int,int>ii;
vector<ii>arr[501];
int v,e,i,a,b,w,t,f,s,flag;
int dist[501],chk[501], cnt[501];

int main() {
        // v : 도시개수(정점-edge 의 개수) e : 버스노선개수(간선-Vertex 의 개수)
        cin >> v >> e;

        // 입력받기
        for (i=0; i<e; i++) {
                cin >> a >> b >> w;
                // a : 출발 도시(시작점), b : 도착도시(도착점), w : 걸린시간 (가중치)
                arr[a].push_back(ii(b,w));
        }

        // 초기화하기
        for (i=1; i<=v; i++)
                dist[i]=MAX;
        dist[1]=0;          // 출발지점은 0으로 설정
        queue<int>q;
        q.push(1);          // queue에 출발지점 저장
        chk[1]=1;
        cnt[1]++;

        while(!q.empty()) {
                t= q.front();
                q.pop();
                chk[t]=0;
                for (auto i : arr[t]) {
                        f=i.first; s = I.second;
                        if(dist[f] > dist[t]+s) {
                                dist[f] =dist[t]+s;
                                if(cnt[f] ==v) {
                                        cout <<-1<<"\n";
                                        return 0;
                                }
                                if(!chk[f]) {
                                        q.push(f);
                                        chk[f]=1;
                                        cnt[f]++;
                                }
                        }
                }
        }

        for (i=2; i<=v; i++)
                (dist[i] == MAX) ? cout<<-1<<"\n" : cout<<dist[i] <<"\n";
}
```

Topological Sort

```
#include<vector>
#include<queue>
#define MAX1 0
int n, inDegree[MAX];
vector<int> a[MAX];

void topologySort() {
        int result[MAX];
        queue<int>q;

        // 진입 차수가 0인 노드를 큐에 삽입한다.
        for (int I=1; i<=N; I++) {
                if(inDegree[i] ==0) q.push(i);
        }
```

```
        // 정렬이 완전히 수행될려면 정확히 n개의 노드를 방문해야한다.
        for (int I=1; i<=N; I++) {
                // n개를 방문하기전 큐가 비어버리면  사이클이 발생한 것
                if(q.empty())
                        cout<< "싸이클"  return;

                int x =q.front();
                q.pop();
                result[i] =x;
                for (int I=0; i<a[x].size(); I++) {
                        int y = a[x][i];

                        // 새로 진입차수가 0이 된 정점을 큐에 삽입 한다.
                        if(--inDegree[y]==0) q.push(y);
                }
        }

        for (int I=1; I<=N; I++) {
                cout<<result[i]
        }
}
```

Strongly Connected Component

```
#include <cstdio>
#include <vector>
#include <stack>
#include <algorithm>
using namespace std;
const int MAX = 10000;

// SN: SCC 개수, sn[i]: i가 속한 SCC의 번호
int V, E, cnt, dfsn[MAX], SN, sn[MAX];
vector<int> adj[MAX];
bool finished[MAX];       // SCC 분리가 끝난 정점만 true
stack<int> S;
vector<vector<int>> SCC;

// 자신의 결과값을 리턴하는 DFS 함수
int DFS(int curr){
    dfsn[curr] = ++cnt; // dfsn 결정
    S.push(curr); // 스택에 자신을 push

    // 자신의 dfsn, 자식들의 결과나 dfsn 중 가장 작은 번호를 result에 저장
    int result = dfsn[curr];
    for(int next: adj[curr]){
        // 아직 방문하지 않은 이웃
        if(dfsn[next] == 0) result = min(result, DFS(next));
        // 방문은 했으나 아직 SCC로 추출되지는 않은 이웃
        else if(!finished[next]) result = min(result, dfsn[next]);
    }

    // 자신, 자신의 자손들이 도달 가능한 제일 높은 정점이 자신일 경우 SCC 추출
    if(result == dfsn[curr]){
        vector<int> currSCC;
        // 스택에서 자신이 나올 때까지 pop
        while(1){
            int t = S.top();
            S.pop();
            currSCC.push_back(t);
            finished[t] = true;
            sn[t] = SN;
            if(t == curr) break;
        }
        // 출력을 위해 원소 정렬
        sort(currSCC.begin(), currSCC.end());
        // SCC 추출
        SCC.push_back(currSCC);
        SN++;
    }
    return result;
}

int main(){
    // 그래프 정보 입력
    scanf("%d %d", &V, &E);
    for(int i=0; i<E; i++){
        int A, B;
        scanf("%d %d", &A, &B);
        adj[A-1].push_back(B-1);
    }
     // DFS를 하며 SCC 추출
    for(int i=0; i<V; i++)
        if(dfsn[i] == 0) DFS(i);
    // 출력을 위해 SCC들을 정렬
    sort(SCC.begin(), SCC.end());
     // SCC 개수
    printf("%d\n", SN);
    // 각 SCC 출력
    for(auto& currSCC: SCC){
        for(int curr: currSCC)
            printf("%d ", curr+1);
```

```
        puts("-1");
    }
}
```

Maximum Matching

```
namespace Matching{
//matching [1...n] <-> [1...m]
const int MX = 40040, MY = 40040;
vector <int> E[MX];
int xy[MX], yx[MY];
int n, m;

void addE(int x, int y) { E[x].pb(y); }
void setnm(int sn, int sm) { n = sn; m = sm; }

int tdis[MX], que[MX], *dis = tdis + 1;
int bfs() {
  int *fr = que, *re = que;
  for(int i=1;i<=n;i++) {
    if(xy[i] == -1) *fr++ = i, dis[i] = 0;
    else dis[i] = -1;
  }
  dis[-1] = -1;
  while(fr != re) {
    int t = *re++;
    if(t == -1) return 1;
    for(int e : E[t]) {
      if(dis[yx[e]] == -1) dis[yx[e]] = dis[t] + 1, *fr++ = yx[e];
    }
  }
  return 0;
}

int dfs(int x) {
  for(int e : E[x]) {
    if(yx[e] == -1 || (dis[yx[e]] == dis[x] + 1 && dfs(yx[e]))) {
      xy[x] = e;
      yx[e] = x;
      return 1;
    }
  }
  dis[x] = -1;
  return 0;
}

int Do() {
  memset(xy, -1, sizeof xy);
  memset(yx, -1, sizeof yx);

  int ans = 0;
  while(bfs()) {
    for(int i=1;i<=n;i++) if(xy[i] == -1 && dfs(i)) ++ans;
  }
  return ans;
}
}


void solve(){
  int n, m;
  scanf("%d%d", &n, &m);
  Matching::setnm(n, m);
  for(int i=1;i<=n;i++) {
    int x; scanf("%d", &x);
    while(x--) {
      int y; scanf("%d", &y);
      Matching::addE(i, y);
    }
  }
  printf("%d\n", Matching::Do());
}
```

Dinic's Algorithm

```
namespace MaxFlow{
const int MV = 20020;
const int ME = 40040;
const int INF = 1e9;
int dis[MV];
int st[MV], en[ME<<1], nxt[ME<<1], flow[ME<<1], ce;
int start[MV];
void init() {
  memset(st, 0, sizeof st);
  ce = 1;
}
void addE(int s, int e, int f = 1) {
  ++ce; nxt[ce] = st[s]; st[s] = ce; en[ce] = e; flow[ce] = f;
  ++ce; nxt[ce] = st[e]; st[e] = ce; en[ce] = s; flow[ce] = 0;
}
int que[MV];
int bfs(int N, int S, int E) {
  for(int i=1;i<=N;i++) dis[i] = -1;
  dis[S] = 0;
  int *fr = que, *re = que;
  *fr++ = S;
  while(fr != re) {
    int t = *re++;
    for(int i=st[t];i;i=nxt[i]) if(flow[i] > 0 && dis[en[i]] == -1) {
      dis[en[i]] = dis[t] + 1;
      *fr++ = en[i];
```

```
    }
  }
  return dis[E] != -1;
}

int dfs(int x, int E, int f) {
  if(x == E) return f;
  for(int &i=start[x],t;i;i=nxt[i]) if(flow[i] > 0 && dis[en[i]] == dis[x] + 1 && (t =
dfs(en[i], E, min(f, flow[i]))) > 0){
    flow[i] -= t;
    flow[i^1] += t;
    return t;
  }
  return 0;
}

int Get(int N, int S, int E) {
  int res = 0;
  while(bfs(N, S, E)) {
    for(int i=1;i<=N;i++) start[i] = st[i];
    while(1) {
      int t = dfs(S, E, INF);
      if(t == 0) break;
      res += t;
    }
  }
  return res;
}
}

int Do(int L) {
  MaxFlow::init();
  int S = n + m + 1, E = n + m + 2;
  for(int i=1;i<=n;i++) MaxFlow::addE(S, i, L);
  rep(i, m) {
    rep(j, 2) MaxFlow::addE(p[i][j], n + 1 + i), g[i][j] = MaxFlow::ce - 1;
  }
  for(int i=1;i<=m;i++) MaxFlow::addE(n + i, E);
  if(MaxFlow::Get(E, S, E) == m) {
    for(int i=1;i<=m;i++) ans[i] = MaxFlow::flow[g[i-1][0]] == 0;
    return 1;
  }
  return 0;
}

typedef pair<int,int> pii;
const int MX = 2505;
int C[MX][MX] = {}, G[MX][MX] = {};

void solve(vector<pii> &E, int N, int M){
  int X[MX] = {}, a, b;

  auto update = [&](int u){ for(X[u] = 1; C[u][X[u]]; X[u]++); };
  auto color = [&](int u, int v, int c){
    int p = G[u][v];
    G[u][v] = G[v][u] = c;
    C[u][c] = v; C[v][c] = u;
    C[u][p] = C[v][p] = 0;
    if( p ) X[u] = X[v] = p;
    else update(u), update(v);
    return p; };
  auto flip = [&](int u, int c1, int c2){
    int p = C[u][c1], q = C[u][c2];
    swap(C[u][c1], C[u][c2]);
    if( p ) G[u][p] = G[p][u] = c2;
    if( !C[u][c1] ) X[u] = c1;
    if( !C[u][c2] ) X[u] = c2;
    return p; };

  for(int i = 1; i <= N; i++) X[i] = 1;
  for(int t = 0; t < E.size(); t++){
    int u = E[t].first, v0 = E[t].second, v = v0, c0 = X[u], c = c0, d;
    vector<pii> L;
    int vst[MX] = {};
    while(!G[u][v0]){
      L.emplace_back(v, d = X[v]);
      if(!C[v][c]) for(a = (int)L.size()-1; a >= 0; a--) c = color(u, L[a].first, c);
      else if(!C[u][d])for(a=(int)L.size()-1;a>=0;a--)color(u,L[a].first,L[a].second);
      else if( vst[d] ) break;
      else vst[d] = 1, v = C[u][d];
    }
    if( !G[u][v0] ){
      for(;v; v = flip(v, c, d), swap(c, d));
      if(C[u][c0]){
        for(a = (int)L.size()-2; a >= 0 && L[a].second != c; a--);
        for(; a >= 0; a--) color(u, L[a].first, L[a].second);
      } else t--;
    }
  }
}
```

Minimum Cost Maximum Flow

```
#include <cstdio>
#include <vector>
#include <queue>
#include <algorithm>
using namespace std;
const int MAX_N = 100;
// 최대 N, M
const int MAX_V = 2*(MAX_N+1);
// 최대 정점 개수
const int S = MAX_V-2;
// 소스 정점 번호
const int E = MAX_V-1;
// 싱크 정점 번호
const int INF = 1000000000;
```

```
int main() {
	// 정점 번호: 0~MAX_N: 서점, MAX_N~MAX_N*2: 사람
	int N, M;
	int c[MAX_V][MAX_V] = {0};		// 각 간선의 용량
	int d[MAX_V][MAX_V] = {0};		// 각 간선의 비용
	int f[MAX_V][MAX_V] = {0} ;		// 각 간선에 흐르는 중인 유량
	vector<int> adj[MAX_V];	// 각 정점의 인접 리스트
	scanf("%d %d", &N, &M);

	// 각 사람 정점과 싱크 정점 사이 간선 추가 (비용 0)
	for (int i=MAX_N; i<MAX_N+N; i++) {
		scanf("%d", &c[i][E]);
		adj[E].push_back(i);
		adj[i].push_back(E);
	}

	// 소스 정점과 각 서점 정점 사이 간선 추가 (비용 0)
	for (int i=0; i<M; i++) {
		scanf("%d", &c[S][i]);
		adj[S].push_back(i);
		adj[i].push_back(S);
	}

	// 서점과 사람 사이 간선 추가 (비용 C_ij)
	for (int i=0; i<M; i++) {
		for (int j=MAX_N; j<MAX_N+N; j++) {
			scanf("%d", &d[i][j]);
			d[j][i] = -d[i][j];
			// 역방향 간선의 비용: 순방향의 -1배
			c[i][j] = INF;
			// 순방향 간선만 용량이 1 이상
			adj[i].push_back(j);
			adj[j].push_back(i);
		}
	}

	int result = 0;
	// 최소 비용
	// MCMF 시작
	while(1) {
		int prev[MAX_V], dist[MAX_V];
		bool inQ[MAX_V] = {0};
		// 해당 정점이 큐 안에 있는가?
		queue<int> Q;
		fill(prev, prev+MAX_V, -1);
		fill(dist, dist+MAX_V, INF);
		dist[S] = 0;
		inQ[S] = true;
		Q.push(S);
		while(!Q.empty()) {
			int curr = Q.front();
			Q.pop();
			inQ[curr] = false;
			// 큐에서 꺼냄
			for (int next: adj[curr]) {
			// 최단 경로를 찾는 중이지만, 여전히 여유 용량 있어야 함
			if(c[curr][next]-f[curr][next] > 0 && dist[next] >
dist[curr]+d[curr][next]) {
					dist[next] = dist[curr] +
d[curr][next];
					prev[next] = curr;
					// 큐에 들어있지 않다면 큐에 넣음
					if(!inQ[next]) {
						Q.push(next);
						inQ[next] = true;
					}
				}
			}
		}
		// 더 이상 경로가 없다면 루프 탈출
		if(prev[E] == -1) break;
		// 경로상에서 가장 residual이 작은 간선을 찾아 최대 흘릴 수 있는
flow 찾음
		int flow = INF;
		for (int i=E; i!=S; i=prev[i])
			flow = min(flow, c[prev[i]][i] - f[prev[i]][i]);
		// 경로상의 모든 간선에 flow만큼의 유량을 흘림
		for (int i=E; i!=S; i=prev[i]) {
			result += flow * d[prev[i]][i];
			// 총 비용이 각 간선 비용만큼 증가
			f[prev[i]][i] += flow;
			f[i][prev[i]] -= flow;
		}
	}
	// 정답 출력
	printf("%d\n", result);
}
```

Biperate Matching

```
#define MAX_N 1001
int n, m;
int visited[MAX_N];
int b[MAX_N];
vector<vector<int>> node;
int dfs(int here) {
        if (visited[here]) return 0;
        //방문 된 정점은 매칭 불가
        visited[here] = 1;
        for (int i = 0; i<node[here].size(); i++) {
                int there = node[here][i];
                if (!b[there] || dfs(b[there])) {
                        // 매칭이 되어있지 않은 정점을 만나거나
                        // 이미 매칭된 정점이 다른 정점과 매칭이 가능할 때
                        b[there] = here;
                        //매칭 시켜준 뒤 1을 리턴 해줌
                        return 1;
                }
        }
        return 0;
}
int bmatch() {
        int ret = 0;
        for (int i = 1; i <= n; i++) {
                //모든 정점에 대하여 매칭을 시도
                memset(visited, 0, sizeof(visited));
                if (dfs(i))ret++;
        }
        return ret;
}
```

Mathematical Stuff

최대공약수와 최소공배수 구하기

```
ll gcd(ll a,ll b){return b?gcd(b,a%b):a;}
int gcd(int a, int b){              // 최대공약수
        while(b!=0){
                int r = a%b;
                a= b;
                b= r;
        }
        return a;
}
int lcm(int a, int b){              // 최소공배수
    return a * ((long long)b / gcd(a,b));
}
```

Polynomial multimplcation – Fast Fourier Transform 이용

$$c_i = \sum_{j=0}^{i} a_j \times b_{i-j}$$

<- 여기에서 C를 계산하기 위한 테크닉임

```
#include <complex>
#include <vector>

using namespace std;
#define pb(x) push_back(x)

namespace FFT{
typedef complex<double> base;
typedef long long ll;

#define all(x) (x).begin(), (x).end()
#define sz(x) ((int)(x).size())

const double C_PI = acos(-1);

void fft(vector <base> &a, bool invert){
  int n = sz(a);
  for(int i=0,j=0;i<n;++i) {
    if(i>j) swap(a[i],a[j]);
    for(int k=n>>1;(j^=k)<k;k>>=1);
  }
  for (int len=2;len<=n;len<<=1){
    double ang = 2*C_PI/len*(invert?-1:1);
    base wlen(cos(ang), sin(ang));
    for (int i=0;i<n;i+=len){
      base w(1);
      for (int j=0;j<len/2;j++){
//      if((j & 511) == 511)w = base(cos(ang * j), sin(ang * j));
//오차가 클 경우 이 빈도를 늘린다. cos, sin 함수는 시간 부담이 있으니 주의
        base u = a[i+j], v = a[i+j+len/2]*w;
        a[i+j] = u+v;
        a[i+j+len/2] = u-v;
        w *= wlen;
      }
    }
  }
  if (invert){
    for (int i=0;i<n;i++) a[i] /= n;
  }
}

void multiply(const vector<int> &a,const vector<int> &b,vector<int> &res){
```

```
  int L = sz(a) + sz(b) + 1;
  vector <base> fa(all(a)), fb(all(b));
  int n = 1;
  while (n < max(sz(a),sz(b))) n <<= 1; n <<= 1;
  fa.resize(n); fb.resize(n);
  fft(fa,false); fft(fb,false);
  for (int i=0;i<n;i++) fa[i] *= fb[i];
  fft(fa,true);
  res.resize(L);
  for(int i=0;i<L;i++)res[i]=((int)(fa[i].real()+(fa[i].real()>0?0.5:-0.5))) & 1;
}

void multiply_with_modulo(const vector<int> &a,const vector<int> &b,vector<int> &res, const int
MOD){
  int n = 1;
  while (n < max(sz(a),sz(b))) n <<= 1; n <<= 1;
  vector <base> A(n), B(n);
  int L_BLOCK = 15;  //2^L_BLOCK ~= sqrt(MOD).
   for(int i=0;i<n;i++)
     A[i] = (i < sz(a) ? base(a[i] & ((1<<L_BLOCK)-1),a[i] >> L_BLOCK) : base(0));
   for(int i=0;i<n;i++)
     B[i] = (i < sz(b) ? base(b[i] & ((1<<L_BLOCK)-1),b[i] >> L_BLOCK) : base(0));
  fft(A, false); fft(B, false);
  vector <base> f1(n), f2(n), f3(n), f4(n);
  for(int i=0;i<n;i++) {
    int j=(n-i)&(n-1);
    f2[i]=(A[i]+conj(A[j]))*base(0.5,0);
    f1[i]=(A[i]-conj(A[j]))*base(0,-0.5);
    f4[i]=(B[i]+conj(B[j]))*base(0.5,0);
    f3[i]=(B[i]-conj(B[j]))*base(0,-0.5);
  }
  for(int i=0;i<n;i++) {
    A[i]=f1[i]*f3[i]+f1[i]*f4[i]*base(0,1);
    B[i]=f2[i]*f4[i]*base(0,1)+f2[i]*f3[i];
  }
  fft(A, true); fft(B, true);
  res.resize(n);
  for(int i=0;i<n;i++) {
    ll g1=(ll)(A[i].real()+0.5) % MOD;  //A[i].real > 0 이어야 함.
    ll g2=(ll)(A[i].imag()+0.5) % MOD;
    ll g3=(ll)(B[i].real()+0.5) % MOD;
    ll g4=(ll)(B[i].imag()+0.5) % MOD;
    res[i] = (g4 + ((g2+g3)<<L_BLOCK) + (g1<<(L_BLOCK<<1))) % MOD;
  }
}

void multiply_big(const vector<int> &a,const vector<int> &b, vector <ll> &res){
  // 단순히 오차가 심해 구하지 못하는 경우
  // 결과값은 long long 범위 안
  int n = 1;
  while (n < max(sz(a),sz(b))) n <<= 1; n <<= 1;
  vector <base> A(n), B(n);
  int L_BLOCK = 10;
   for(int i=0;i<n;i++)
     A[i] = (i < sz(a) ? base(a[i] & ((1<<L_BLOCK)-1),a[i] >> L_BLOCK) : base(0));
   for(int i=0;i<n;i++)
     B[i] = (i < sz(b) ? base(b[i] & ((1<<L_BLOCK)-1),b[i] >> L_BLOCK) : base(0));
  fft(A, false); fft(B, false);
  vector <base> f1(n), f2(n), f3(n), f4(n);
  for(int i=0;i<n;i++) {
    int j=(n-i)&(n-1);
    f2[i]=(A[i]+conj(A[j]))*base(0.5,0);
    f1[i]=(A[i]-conj(A[j]))*base(0,-0.5);
    f4[i]=(B[i]+conj(B[j]))*base(0.5,0);
    f3[i]=(B[i]-conj(B[j]))*base(0,-0.5);
  }
  for(int i=0;i<n;i++) {
    A[i]=f1[i]*f3[i]+f1[i]*f4[i]*base(0,1);
    B[i]=f2[i]*f4[i]*base(0,1)+f2[i]*f3[i];
  }
  fft(A, true); fft(B, true);
  res.resize(n);
  for(int i=0;i<n;i++) {
    ll g1=(ll)(A[i].real()+0.5);
    ll g2=(ll)(A[i].imag()+0.5);
    ll g3=(ll)(B[i].real()+0.5);
    ll g4=(ll)(B[i].imag()+0.5);
    res[i] = (g4 + ((g2+g3)<<(L_BLOCK)) + (g1<<(L_BLOCK<<1)));
  }
}
}
```

Polynomial multimplcation with special modulo

정수만을 사용하여 다항식 곱셈을 한 것.

```
#include <cstdio>
const int A = 7, B = 26, P = A << B | 1, R = 3;
const int SZ = 20, N = 1 << SZ;
int Pow(int x, int y) {
  int r = 1;
  while (y) {
    if (y & 1) r = (long long)r * x % P;
    x = (long long)x * x % P;
    y >>= 1;
  }
  return r;
}
void FFT(int *a, bool f) {
  int i, j, k, x, y, z;
  j = 0;
  for (i = 1; i < N; i++) {
    for (k = N >> 1; j >= k; k >>= 1) j -= k;
    j += k;
    if (i < j) {
      k = a[i];
      a[i] = a[j];
      a[j] = k;
    }
  }
  for (i = 1; i < N; i <<= 1) {
    x = Pow(f ? Pow(R, P - 2) : R, P / i >> 1);
```

```
    for (j = 0; j < N; j += i << 1) {
      y = 1;
      for (k = 0; k < i; k++) {
        z = (long long)a[i | j | k] * y % P;
        a[i | j | k] = a[j | k] - z;
        if (a[i | j | k] < 0) a[i | j | k] += P;
        a[j | k] += z;
        if (a[j | k] >= P) a[j | k] -= P;
        y = (long long)y * x % P;
      }
    }
  }
  if (f) {
    j = Pow(N, P - 2);
    for (i = 0; i < N; i++) a[i] = (long long)a[i] * j % P;
  }
}
int X[N];
int main() {
  int i, n;
  scanf("%d", &n);
  for (i = 0; i <= n; i++) scanf("%d", &X[i]);
  FFT(X, false);
  for (i = 0; i < N; i++) X[i] = (long long)X[i] * X[i] % P;
  FFT(X, true);
  for (i = 0; i <= n + n; i++) printf("%d ", X[i]);
}
```

Geometry

Point and Line

```
#define EPS 1e-9
// struct point_i { int x, y; };            // 가장 기본적인, 최소한의 형태
struct point_i {                            // 가능하면 point_i를 사용 (정수형)
    int x, y;
    point_i() { x = y = 0; }                    // 기본 생성자
    point_i(int _x, int _y) : x(_x), y(_y) {}      // 사용자 정의 생성자

    // 점들을 정렬해야 하는 경우
    bool operator < (point_i other) const {   // 연산자를 재정의한다.
        if (fabs(x-other.x) > EPS) // 이 연산자는 점들을 정렬할 때 유용하게
            return x < other.x;     // 먼저는 x좌표를 기준으로 비교하고
        return y < other.y;     // 다음으로는 y좌표를 기준으로 비교한다.
    };
};

struct point {              // 좌표를 실수 값으로 표현해야 하는 경우에만 사용
    double x, y;
    point() { x = y = 0; }                      // 기본 생성자
    point(double _x, double _y) : x(_x), y(_y) {}   // 사용자 정의 생성자

    // 점들을 정렬해야 하는 경우
    bool operator < (point other) const {      // 연산자를 재정의한다.
        if (fabs(x-other.x) > EPS) // 이 연산자는 점들을 정렬할 때 유용하게 사용
            return x < other.x;     // 먼저는 x좌표를 기준으로 비교하고
        return y < other.y;       // 다음으로는 y좌표를 기준으로 비교한다.
    };

    // 두 점이 서로 같은지를 검사해야 하는 경우
    // 두 실수가 서로 같은지를 비교할 때는 EPS(1e-9)를 사용한다.
    bool operator == (point other) const {
        return (fabs(x - other.x) < EPS && (fabs(y - other.y) < EPS));
    };

    // 유클리드 거리 구하기
    double dist(point p1, point p2) {
        //hypot(dx, dy)는 sqrt(dx * dx + dy * dy)를 반환한다.
        return hypot(p1.x - p2.x, p1.y - p2.y); // double형을 반환한다.
    };

    // p를 원점 (0, 0)을 중심으로 반시계 방향으로 theta도 회전한다.
    point rotate(point p, double theta) {
        double rad = DEG_TO_RAD(theta); // theta에 (PI / 180.0)을 곱한다.
        return point(p.x * cos(rad) - p.y * sin(rad), p.x * sin(rad) + p.y * cos(rad));
    };
};

// int main 함수 내부, vector <point> P를 미리 생성해두었다고 가정
sort(P.begin(), P.end());   // 비교 연산자를 위에서 정의하였다.

// int maiN 함수 내부, == 사용 예시
point P1(0,0), P2(0,0), P3(0,1);
printf("%d\n", P1 == P2);   // 참
printf("%d\n", P1 == P3);   // 거짓

// 1차원 도형 : 직선
// 직선의 방정식 ax + by + c = 0, 여기에서 직선이 수직선인 경우 b = 0, 수직선이 아닐 경우 b = 1
struct line { double a, b, c };

// 만약 직선을 지나는 점이 적어도 두 개 주어졌다면, 직선의 방정식을 세울 수 있다.
void pointsToLine(point p1, point p2, line &l) {
    if (fabs(p1.x - p2.x) < EPS) {                    // 수직선도 괜찮다.
        l.a = 1.0; l.b = 0.0; l.c = -p1.x;             // 기본 값
    }
    else {
        l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
        l.b = 1.0;                                  // 중요 : b의 값을 1.0으로 고정한다.
        l.c = -(double)(l.a * p1.x) - p1.y;
    }
}

// 두 직선이 평행(parallel)한지를 검사하려면, 계수 a와 b가 서로 같은지 검사하면 된다.
bool areParallel(line l1, line l2) {    // 계수 a와 b를 검사한다.
    return (fabs(l1.a - l2.a) < EPS) && (fabs(l1.b - l2.b) < EPS);
```

```
}

// 두 직선이 일치(same)하는지를 검사하려면, 두 직선이 평행하며 계수 c가 서로 같은지 검사해보면 된다.
// 이 경우 계수 a, b, c 세 개가 모두 서로 같은지 검사한다.
bool areSame(line l1, line l2) {       // 계수 c도 검사한다.
    return areParallel(l1, l2) && (fabs(l1.c - l2.c) < EPS);
}

// 두 직선이 서로 평행하지 않다면(일치하지도 않다면), 둘은 한 점에서 교차(intersect)한다. 그 교점
(intersection point) (x,y)를 구하려면, 미지수가 두 개이며 선형 방정식 두 개로 이루어진 연립 방정식을
풀어야 한다.
bool areIntersect(line l1, line l2, point &p) {
    if (areParallel(l1, l2)) return false;  // 교차하지 않는다.

    // 미지수가 두 개이며 선형 방정식 두 개로 이루어진 연립 방정식을 푼다.
    p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a * l2.b);

    // 예외 케이스, 0으로 나누지 않도록 하기 위해 수직선인지를 검사한다.
    if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
    else p.y = -(l2.a * p.x + l2.c);
    return true;
}
```

Geometry Algorithm Example : 교차점 문제
- 사각형과 선분의 교차점이 0개, 1개, 2개 중 어떤지 판정하는 문제

```
#include <iostream>
#include <set>
#include <algorithm>
#include <vector>
using namespace std;
typedef long long ll;
struct POS {
          int x, y;
};
vector<int> ans;
set<pair<int, int> > chk;
int ccw(int x1, int y1, int x2, int y2, int x3, int y3) {
          ll ret = (x1 * y2 + x2 * y3 + x3 * y1) - (y1 * x2 + y2 * x3 + y3 * x1);
          if (ret > 0) return 1;
          else if (ret < 0) return -1;
          else return 0;
}
bool isCross(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
          // ccw곱 홀수 + 사각형 내부 피하면 교차
          if (ccw(x1, y1, x2, y2, x3, y3) * ccw(x1, y1, x2, y2, x4, y4) <= 0 &&
                    ccw(x3, y3, x4, y4, x1, y1) * ccw(x3, y3, x4, y4, x2, y2) <= 0) {
                    if ((x1 < x3 && x1 < x4 && x2 < x3 && x2 < x4) ||
                              (x3 < x1 && x3 < x2 && x4 < x1 && x4 < x2)) return false;
                    if ((y1 < y3 && y1 < y4 && y2 < y3 && y2 < y4) ||
                              (y3 < y1 && y3 < y2 && y4 < y1 && y4 < y2)) return false;
                    return true;
          }
          return false;
}
bool isCross_rect_point(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
          int line = ccw(x1, y1, x2, y2, x3, y3) * ccw(x1, y1, x2, y2, x4, y4);
          int rect = ccw(x3, y3, x4, y4, x1, y1) * ccw(x3, y3, x4, y4, x2, y2);
          if (line == 0 && rect <= 0) return true;
          else return false;
}

bool isCross_rect_line(int x1, int y1, int x2, int y2, int x3, int y3, int x4, int y4) {
          int line = ccw(x1, y1, x2, y2, x3, y3) * ccw(x1, y1, x2, y2, x4, y4);
          int rect = ccw(x3, y3, x4, y4, x1, y1) * ccw(x3, y3, x4, y4, x2, y2);
          if (line < 0 && rect <= 0) return true;
          else return false;
}

int main(void)
{
          int T;
          cin >> T;

          int xmin, ymin, xmax, ymax;
          for (int i = 0; i < T; i++) {
                    cin >> xmin >> ymin >> xmax >> ymax;
                    POS r1 = { xmin, ymin };
                    POS r2 = { xmin, ymax };
                    POS r3 = { xmax, ymin };
                    POS r4 = { xmax, ymax };

                    POS l1, l2;
                    int x1, y1, x2, y2;
                    cin >> l1.x >> l1.y >> l2.x >> l2.y;
                    if (l1.x > l2.x) swap(l1, l2);

                    // 1. 교점이 없는 경우
                    bool isNone = true;
                    if (isCross(l1.x, l1.y, l2.x, l2.y, r1.x, r1.y, r2.x, r2.y) ||
                              isCross(l1.x, l1.y, l2.x, l2.y, r2.x, r2.y, r4.x, r4.y) ||
                              isCross(l1.x, l1.y, l2.x, l2.y, r4.x, r4.y, r3.x, r3.y) ||
                              isCross(l1.x, l1.y, l2.x, l2.y, r3.x, r3.y, r1.x, r1.y)) isNone
= false;
                    if (isNone) {
                              cout << 0 << '\n';
                              continue;
                    }

                    // 2. 교점이 무수히 많은 경우 -> x 좌표 기준으로만 정렬했으므로,
                    // y좌표는 뭐가 큰지를 모르니 두 경우를 다 고려한다.
                    if (l1.x == l2.x && l1.x == xmin) {
                              if ((l1.y < ymax) && (l2.y > ymin) || (l2.y < ymax) && (l1.y >
ymin)) {
                                        cout << 4 << "\n";
                                        continue;
                              }
                    }
                    else if (l1.x == l2.x && l1.x == xmax) {
                              if ((l1.y < ymax) && (l2.y > ymin) || (l2.y < ymax) && (l1.y >
```

```
ymin)) {
                                        cout << 4 << "\n";
                                        continue;
                                }
                        }
                        else if (l1.y == l2.y && l1.y == ymax) {
                                if ((l1.x < xmax) && (l2.x > xmin) || (l2.x < xmax) && (l1.x >
xmin)) {
                                        cout << 4 << "\n";
                                        continue;
                                }
                        }
                        else if (l1.y == l2.y && l1.y == ymin) {
                                if ((l1.x < xmax) && (l2.x > xmin) || (l2.x < xmax) && (l1.x >
xmin)) {
                                        cout << 4 << "\n";
                                        continue;
                                }
                        }
                        int cnt_rect_line = 0, cnt_rect_point = 0;
                        if (isCross_rect_line(l1.x, l1.y, l2.x, l2.y, r1.x, r1.y, r2.x, r2.y))
cnt_rect_line++;
                        if (isCross_rect_line(l1.x, l1.y, l2.x, l2.y, r2.x, r2.y, r4.x, r4.y))
cnt_rect_line++;
                        if (isCross_rect_line(l1.x, l1.y, l2.x, l2.y, r4.x, r4.y, r3.x, r3.y))
cnt_rect_line++;
                        if (isCross_rect_line(l1.x, l1.y, l2.x, l2.y, r3.x, r3.y, r1.x, r1.y))
cnt_rect_line++;

                        if (isCross_rect_point(l1.x, l1.y, l2.x, l2.y, r1.x, r1.y, r2.x, r2.y))
cnt_rect_point++;
                        if (isCross_rect_point(l1.x, l1.y, l2.x, l2.y, r2.x, r2.y, r4.x, r4.y))
cnt_rect_point++;
                        if (isCross_rect_point(l1.x, l1.y, l2.x, l2.y, r4.x, r4.y, r3.x, r3.y))
cnt_rect_point++;
                        if (isCross_rect_point(l1.x, l1.y, l2.x, l2.y, r3.x, r3.y, r1.x, r1.y))
cnt_rect_point++;

                        cout << cnt_rect_line + cnt_rect_point / 2 << "\n";
                }
                return 0;
}
```

Geometry Algorithm Example : 맹독 방벽 문제
- Convex Hull 구하기, 테두리 길이 구하기

```
using namespace std;
#define x first
#define y second
typedef long long ll;
typedef pair <ll, ll> p;
const double pi = 3.1415926535;

int n;
ll l;
vector <p> v, hull;

int ccw(p a, p b, p c) {
        ll res = a.x * b.y + b.x * c.y + c.x * a.y;
        res -= b.x * a.y + c.x * b.y + a.x * c.y;
        if (res > 0) return 1;
        if (res) return -1;
        return 0;
}

double dist(p a, p b) {
        ll dx = a.x - b.x;
        ll dy = a.y - b.y;
        return sqrt(dx * dx + dy * dy);
}

// vector의 내적
ll dot(p a, p b) {
        return a.x * b.x + a.y * b.y;
}

int main(void)
{
        ios_base::sync_with_stdio(false);
        cin.tie(NULL); cout.tie(NULL);
        cin >> n >> l;
        v.resize(n); // 빨리 vector에 넣기 위함
        for (int i = 0; i < n; i++)
                cin >> v[i].x >> v[i].y;

        swap(v[0], *max_element(v.begin(), v.end()));
        sort(v.begin() + 1, v.end(), [&](p& a, p& b) {
                int cw = ccw(v[0], a, b);
                if (cw) return cw > 0;
                return dist(v[0], a) < dist(v[0], b);
                });

        for (auto i : v) {
                while (hull.size() >= 2 && ccw(hull[hull.size() - 2],
hull.back(), i) <= 0)
                        hull.pop_back();
                hull.push_back(i);
        }

        n = hull.size();
```

```
        double ans = 0;

        // 1. convex hull에 속해 있는 점들의 테두리 길이?
        for (int i = 0; i < n; i++) {
                p prv = hull[(i + n - 1) % n];
                p now = hull[i];
                p nxt = hull[(i + 1) % n];
                double dist1 = dist(now, nxt);
                ans += dist1;

                double dist2 = dist(prv, now);

                // 2. 곡선의 길이?
                // now->prv 벡터와 now->nxt 벡터의 내적
                ll inner = dot({ prv.x - now.x, prv.y - now.y }, { nxt.x -
now.x, nxt.y - now.y });
                double theta = acos((double)inner / dist1 / dist2);
                theta = pi - theta; // 180 - theta
                ans += l * theta; // 호의 길이를 구한다고 생각한다.
        }
        cout << round(ans); // 정수 단위로 반올림
}
```

Geometry Algorithm Example : 점 분리 문제
- Convex Hull 두 개를 구하고, 그 Convex Hull이 서로 겹치는지 확인하는 문제

```
#include <iostream>
#define ll long long
using namespace std;

struct node {
        ll x = 0, y = 0;
        ll p = 0, q = 0;
        node operator - (const node& p) {
                return { x - p.x, y - p.y, 0, 0 };
        }
        ll cross(const node& p) {
                return x * p.y - y * p.x;
        }
        bool operator < (const node& p) {
                if (x != p.x) return x < p.x;
                return y < p.y;
        }
};

node black[105];
node white[105];

bool comp(node p1, node p2) {
        if (p1.q * p2.p != p1.p * p2.q) return p1.q * p2.p < p1.p * p2.q;
        if (p1.y != p2.y) return p1.y < p2.y;
        return p1.x < p2.x;
}

void qsort(node* p, int left, int right) {
        if (left >= right) return;
        int l = left - 1;
        int r = right + 1;
        node mid = p[(l + r) / 2];
        while (1) {
                while (comp(p[++l], mid));
                while (comp(mid, p[--r]));
                if (l >= r) break;
                node tmp = p[l];
                p[l] = p[r];
                p[r] = tmp;
        }
        qsort(p, left, l - 1);
        qsort(p, r + 1, right);
}

struct STACK {
        int stack[105];
        int size = 0;

        void push(int n) {
                stack[size++] = n;
        }

        void pop() {
                size--;
        }

        int top() {
                return stack[size - 1];
        }
};

int ccw(node p1, node p2, node p3) {
        ll ans = (p2 - p1).cross(p3 - p2);
        if (ans > 0) ans = 1;
        else if (ans < 0) ans = -1;
```

```
        return ans;
}

bool isCross(node p1, node p2, node p3, node p4) {
        int ab = ccw(p1, p2, p3) * ccw(p1, p2, p4);
        int cd = ccw(p3, p4, p1) * ccw(p3, p4, p2);
        if (ccw(p1, p2, p3) == 0 && ccw(p1, p2, p4) == 0) {
                if (p2 < p1) swap(p1, p2);
                if (p4 < p3) swap(p3, p4);
                return !(p2 < p3 || p4 < p1);
        }
        return ab <= 0 && cd <= 0;
}

//점 a가 b안에 있는지 검사
bool isPointInside(STACK& s, node* a, node* b) {
        if (s.size <= 2) return false;

        int prevDir = ccw(b[s.stack[0]], b[s.stack[1]], a[0]);

        for (int i = 1; i < s.size; i++) {
                int nowDir = ccw(b[s.stack[i]], b[s.stack[(i + 1) %
s.size]], a[0]);
                if (prevDir != nowDir) return false;
        }
        return true;
}

void sortCCW(node* point, int num) {
        qsort(point, 0, num - 1);
        for (int i = 1; i < num; i++) {
                point[i].p = point[i].x - point[0].x;
                point[i].q = point[i].y - point[0].y;
        }
        qsort(point, 1, num - 1);
}

void convexHull(STACK& s, node* point, int num) {
        if (num > 0) s.push(0);
        if (num > 1) s.push(1);
        int next = 2;
        while (next < num) {
                while (s.size >= 2) {
                        int second = s.top();
                        s.pop();
                        int first = s.top();
                if (ccw(point[first], point[second], point[next]) > 0) {
                                s.push(second);
                                break;
                        }
                }
                s.push(next++);
        }
}

bool polygonIntersects(STACK& a, STACK& b) {
        if (isPointInside(b, black, white) || isPointInside(a, white, black))
return true;

        for (int i = 0; i < a.size; i++) {
                for (int j = 0; j < b.size; j++) {
                        if (isCross(black[a.stack[i]], black[a.stack[(i +
1) % a.size]], white[b.stack[j]], white[b.stack[(j + 1) % b.size]]))
                                return true;
                }
        }
        return false;
}

int main() {
        int t;  cin >> t;
        while (t--) {
                int n, m;
                cin >> n >> m;
                for (int i = 0; i < 105; ++i) {
                   black[i].p = black[i].q = white[i].p = white[i].q = 0;
                }
                for (int i = 0; i < n; i++) {
                        cin >> black[i].x >> black[i].y;
                }
                for (int i = 0; i < m; i++) {
                        cin >> white[i].x >> white[i].y;
                }
                if (n <= 1 && m <= 1) {
                        cout << "YES\n";
                        continue;
                }
                STACK bs;
                sortCCW(black, n);
                convexHull(bs, black, n);

                STACK ws;
```

```
                sortCCW(white, m);
                convexHull(ws, white, m);

                if (polygonIntersects(bs, ws)) cout << "NO\n";
                else cout << "YES\n";
        }
}
```

Geometry Algorithm Example : 연돌이와 고잠녀 문제
- 수선의 발을 구할 수 있는가?

```
* 두 선분 사이의 거리를 구할 수 있는가?
* 수선의 발을 구할 수 있는가? 를 물어보는 문제
* 벡터의 내적과 외적을 사용하여 문제를 푼다.
* 각도가 둔각이 있으면 수선의 발을 내리는 게 불가능하다.
* |a * b| = |a||b||sin(theta)| 이용
--------------------------------------------------------------------------------
using namespace std;
double distance(double x1, double y1, double x2, double y2) {
        return sqrt(pow(x1 - x2, 2) + pow(y1 - y2, 2));
}
double dot(double x1, double y1, double x2, double y2) {
        return x1 * x2 + y1 * y2;
}

double cross(double x1, double y1, double x2, double y2) {
        return x1 * y2 - x2 * y1;
}

// 선분 12에 점 3에서 수선의 발을 내릴 수 있으면 수선의 길이 아니면 -1
double perpendicular(double x1, double y1, double x2, double y2, double x3,
double y3) {
        double dot1 = dot(x2 - x1, y2 - y1, x3 - x1, y3 - y1);
        double dot2 = dot(x1 - x2, y1 - y2, x3 - x2, y3 - y2);
        // 점 3이 선분 12와 예각 2개를 이루면 수선을 내릴 수 있음
        if (dot1*dot2 >= 0)
                return abs(cross(x2 - x1, y2 - y1, x3 - x1, y3 - y1)) /
distance(x1, y1, x2, y2);
        return -1;
}
int main(void) {
        int N, M;
        double yx[4000], yy[4000], result = -1;
        cin >> N >> M;
        //신촌의 도로입력
        for (int i = 0; i < N * 2; i++) {
                cin >> yx[i] >> yy[i];
        }
        //안암의 도로입력
        for (int i = 0; i < M; i++) {
                double x1, y1, x2, y2, x3, y3, x4, y4;
                cin >> x3 >> y3 >> x4 >> y4;
                //입력받자 마자 신촌의 모든 도로들과 거리를 잰다.
                for (int j = 0; j < N; j++) {
                        //신촌의 한도로 12, 안암의 한도로 34
                        x1 = yx[j * 2];          x2 = yx[j * 2 + 1];
                        y1 = yy[j * 2];          y2 = yy[j * 2 + 1];
                        // 모든 점 쌍 사이의 거리가  후보가 될 수 있다.
                        double dist = distance(x1, y1, x3, y3);
                        double temp;
                        dist = min(dist, distance(x1, y1, x4, y4));
                        dist = min(dist, distance(x2, y2, x3, y3));
                        dist = min(dist, distance(x2, y2, x4, y4));
        // 어느 점에서 다른 선분에 수선을 내릴 수 있으면 수선의 길이도 후보
                        temp = perpendicular(x1, y1, x2, y2, x3, y3);
                        if (temp >= 0) dist = min(dist, temp);
                        temp = perpendicular(x1, y1, x2, y2, x4, y4);
                        if (temp >= 0) dist = min(dist, temp);
                        temp = perpendicular(x3, y3, x4, y4, x1, y1);
                        if (temp >= 0) dist = min(dist, temp);
                        temp = perpendicular(x3, y3, x4, y4, x2, y2);
                        if (temp >= 0) dist = min(dist, temp);
                        if (result < 0) result = dist;
                        else result = min(result, dist);
                }
        }

        cout << fixed;
        cout.precision(16);
        cout << result;  return 0;        }
```