1. **수학**

* **에라토스테네스의 체:**

void sieveOfEratosthenes(int n) {

vector<bool> isPrime(n + 1, true);

isPrime[0] = isPrime[1] = false; // 0, 1은 소수 X

for (int i = 2; i \* i <= n; ++i) {

if (isPrime[i]) {

for (int j = i \* i; j <= n; j += i) {

isPrime[j] = false;

}

}

}

// 소수 출력

for (int i = 2; i <= n; ++i) {

if (isPrime[i]) {

std::cout << i << " ";

}

}

std::cout << std::endl;

}

* **오일러 피 함수:**

int euler\_phi(int n) { // [1, n]에서 n과 서로소인 개수 (cf. 1은 1과도 서로소 O)

double ret = n;

int sqrt\_n = (int)round(sqrt(n));

for(int div = 2; div <= sqrt\_n; div++) {

if(n % div == 0) {

ret \*= (1 - (double)1/(double)div); // 1 - 1/p

while(n % div == 0) {

n /= div;

}

}

}

if(n > 1) ret \*= (1 - (double)1/(double)n); // 빼먹지 않도록 조심! (sqrt\_n까지만 대칭적으로 고려했으므로)

return (int)round(ret);

}

* **확장 유클리드 알고리즘:**

pii extended\_gcd(int a, int b) {

if (b == 0) return {1, 0}; // 편의상 y값을 0으로 설정. 당연히 임의의 값 가능.

int x0, y0;

tie(x0, y0) = extended\_gcd(b, a % b);

int x, y;

x = y0;

y = x0 - (a / b) \* y0;

return {x, y};

}

* **소인수분해 (일반 버전):**

vector<int> factorization(int n) {

vector<int> ret;

int sqrt\_n = (int)round(sqrt(n));

for(int div = 2; div <= sqrt\_n; div++) { // div \* div <= n으로 해도 되는 듯

cout<<"div: "<<div<<"\n";

while(n % div == 0) {

n /= div;

cout<<"divide "<<div<<" -> "<<n<<"\n";

ret.push\_back(div);

}

}

if(n > 1) ret.push\_back(n); // 빼먹지 않도록 조심! (sqrt\_n까지만 대칭적으로 고려했음)

return ret;

}

* **폴라드 로 & 밀러-라빈 => 큰 수 소인수분해:**

#define MULTIPLY(a, b, c) (((a % c) \* (b % c)) % c)

#define f(a, b, c) ((a \* a + b) % c)

#define K 10 // false positive 확률 : 1 / 1024 이하

using namespace std;

typedef \_\_int128 ll;

////////////////////////////////////////////////////////////////////////

// \_\_int128 출력 함수 (by chatGPT)

void printInt128(ll n) { // O(lg N) -> 최대 18번

if (n == 0) {

cout << "0";

return;

}

string result;

bool isNegative = (n < 0);

if (isNegative) n = -n;

while (n > 0) {

result.push\_back((n % 10) + '0');

n /= 10;

}

if (isNegative) result.push\_back('-');

reverse(result.begin(), result.end());

cout << result;

}

// \_\_int128 입력 함수 (by chatGPT)

ll readInt128() { // O(lg N) -> 최대 18번

string s;

cin >> s;

ll result = 0;

bool isNegative = (s[0] == '-');

for (size\_t i = (isNegative ? 1 : 0); i < s.size(); ++i) {

result = result \* 10 + (s[i] - '0');

}

return isNegative ? -result : result;

}

////////////////////////////////////////////////////////////////////////

ll power(ll a, ll d, ll n) { // a ^ d mod n 구하기

if (d == 0) return 1;

if (d == 1) return a % n;

ll half = power(a, d / 2, n);

if (d % 2) return MULTIPLY(a, MULTIPLY(half, half, n), n);

else return MULTIPLY(half, half, n);

}

bool miller\_rabin(ll& a, ll& n) { // test 1 & 2

ll s = 0;

ll d = n - 1;

while (d % 2 == 0) {

d /= 2;

s++;

}

ll base = power(a, d, n);

if (base == 1) return true;

for (ll i = 0; i < s; i++) {

if (base == n - 1) return true;

base = MULTIPLY(base, base, n);

}

return false;

}

bool is\_prime(ll& n) { // 3 이상의 홀수에 대하여 소수 판별

// for edge case에서의 소수 판정 (1, 2, 3, 짝수)

if(n == 1) return false;

if(n == 2 || n == 3) return true;

if(n % 2 == 0) return false; // 4이상의 짝수는 소수가 X

for (ll j = 0; j < K; j++) {

ll a = rand() % (n - 3) + 2;

if (!miller\_rabin(a, n)) return false;

}

return true;

}

ll pollardRho(ll N) { // 합성수에 대해서만 적용 가능 O

if (N % 2 == 0) return 2;

ll x = 2, y = 2, c = 1, d = 1; // 편의상 x, y를 2로, c를 1로 초기화

while (d == 1) { // (1, N) 구간의 d 찾기

x = f(x, c, N); // x <- f(x)

y = f(f(y, c, N), c, N); // y <- f(f(y))

ll diff = (x > y) ? x - y : y - x; // abs 사용하지 않기 위해

d = \_\_gcd(diff, N);

if (d == N) {

c++; // 다른 c로 재도전

x = 2;

y = 2;

d = 1;

}

}

return d;

}

void factorize(ll N, vector<ll>& factors) {

if (N <= 1) return; // 1은 무시

if (N % 2 == 0) { // 짝수라면,

factors.push\_back(2);

factorize(N / 2, factors);

return;

}

if (is\_prime(N)) { // 소수라면,

factors.push\_back(N);

return;

}

ll factor = pollardRho(N);

factorize(factor, factors);

factorize(N / factor, factors);

}

* **뤼카 정리:**

/\*

뤼카 정리:

소수 p진법으로 나타냈을 때,

N = N(l)N(l-1) ... N(0)

K = K(l)K(l-1) ... K(0)

=> (N, K) = (N(i), K(i))들의 곱

시간복잡도: O(p ^ 2 + lg N)

증명 idea:

- N을 p진법으로 나타냈었을 때 (1 + x) ^ N 식 변형

- 각각의 N(i)에 대한 x ^ K에 대한 계수를 생각

\*/

ll N, K, M;

vector<vector<ll>> C;

void init() {

for(int i = 0; i <= M; i++) C[i][0] = C[i][i] = 1; // 초기화 조심!

for(int i = 1; i <= M; i++) {

for(int j = 1; j <= i; j++) {

C[i][j] = ADD(C[i-1][j], C[i-1][j-1], M);

}

}

}

void print\_C() {

for(int i = 0; i <= M; i++) {

for(int j = 0; j <= i; j++) {

cout<<C[i][j]<<" ";

}

cout<<"\n";

}

}

int main() {

ios::sync\_with\_stdio(false);

cin.tie(NULL); cout.tie(NULL);

cin>>N>>K>>M;

C.resize(M + 1, vector<ll>(M + 1, 0));

init();

//print\_C();

ll ans = 1;

while(N || K) {

ans = MULTIPLY(ans, C[N%M][K%M], M);

N /= M; K /= M;

}

cout<<ans;

return 0;

}

1. **자료구조**

* **2D 벡터:**

const double PI = 2.0 \* acos(0.0); // pi 값

template <typename T>

struct vec2 {

T x, y;

vec2(T x = 0, T y = 0) : x(x), y(y) {}

// 벡터 동일 여부

bool operator == (const vec2<T>& r) const {

return x == r.x && y == r.y;

}

// 정렬 기준

bool operator < (const vec2<T>& r) const {

return x != r.x ? x < r.x : y < r.y;

}

// 두 벡터 합 연산

vec2<T> operator + (const vec2<T>& r) const {

return vec2<T>(x + r.x, y + r.y);

}

// 두 벡터 차 연산

vec2<T> operator - (const vec2<T>& r) const {

return vec2<T>(x - r.x, y - r.y);

}

// 스칼라 실수 k배

vec2<T> operator \* (T k) const {

return vec2<T>(x \* k, y \* k);

}

// 벡터 크기

double norm() const {

return hypot(x, y); // 피타고라스 정리로 벡터 크기 계산

}

// 정규화된 벡터 반환

vec2<double> normalize() const {

return vec2<double>(x / norm(), y / norm());

}

// 극좌표계에서의 각도

double polar() const {

return fmod(atan2(y, x) + 2 \* M\_PI, 2 \* M\_PI);

}

// 벡터 내적

T dot(const vec2<T>& r) const {

return x \* r.x + y \* r.y;

}

// 벡터 외적

T cross(const vec2<T>& r) const {

return x \* r.y - y \* r.x;

}

// 벡터 사영

vec2<double> project(const vec2<T>& r) const {

vec2<double> u = r.normalize();

return u \* u.dot(\*this);

}

};

// 벡터 버전 CCW (템플릿 적용)

template <typename T>

double ccw(const vec2<T>& a, const vec2<T>& b) {

return a.cross(b); // a X b 외적 (양수: 반시계, 음수: 시계)

}

// 세 점 버전 CCW (템플릿 적용)

template <typename T>

double ccw(const vec2<T>& a, const vec2<T>& b, const vec2<T>& c) { // a -> b, b -> c

return ccw(b - a, c - b);

}

template <typename T>

void get\_convex\_hull(vector<vec2<T>>& coord, vector<vec2<T>>& hull) { // 정수 좌표일 때 convex hull 구하기

int N = coord.size();

sort(coord.begin(), coord.end());

for(int i = 0; i < N; i++) {

while(hull.size() >= 2 && ccw(hull[hull.size() - 2], hull[hull.size() - 1], coord[i]) >= 0) {

hull.pop\_back();

}

hull.push\_back(coord[i]);

}

int upper\_hull = hull.size();

for(int i = N - 2; i >= 0; i--) {

while(hull.size() >= upper\_hull + 1 && ccw(hull[hull.size() - 2], hull[hull.size() - 1], coord[i]) >= 0) { // 위쪽 껍질은 건들지 않도록!

hull.pop\_back();

}

hull.push\_back(coord[i]);

}

hull.pop\_back(); // 처음 위치 중복 제거

}

* **세그먼트 트리:**

struct segment\_tree {

int n; // 배열의 길이

vector<ll> tree;

segment\_tree(vector<ll>& arr) {

n = arr.size();

tree.resize(n \* 4);

init(arr, 0, n - 1, 1); // [0, n) 초기화 (단, 루트 노드는 1)

}

// top-down 방식 초기화

// T(n) = 2 \* T(n/2) + 1 이므로 O(n)

ll init(vector<ll>& arr, int l, int r, int idx) { // 가정 : idx 노드 표현 범위 -> [l, r]

if(l == r) return tree[idx] = arr[l]; // 리프 노드라면,

int mid = l + (r - l)/2;

ll left\_result = init(arr, l, mid, idx \* 2); // 왼쪽 서브 트리

ll right\_result = init(arr, mid + 1, r, idx \* 2 + 1); // 오른쪽 서브 트리

return tree[idx] = left\_result + right\_result; // 병합

}

ll query(int l, int r, int sl, int sr, int idx) { // 가정 : idx 노드 표현 범위 -> [sl, sr]

if(l > sr || r < sl) return 0; // 공집합이라면 결과에 무의미한 0을 반환

if(l <= sl && r >= sr) return tree[idx]; // [sl, sr]이라면,

int smid = sl + (sr - sl)/2;

ll left\_query = query(l, r, sl, smid, idx \* 2); // 왼쪽 서브 트리

ll right\_query = query(l, r, smid + 1, sr, idx \* 2 + 1); // 오른쪽 서브 트리

return left\_query + right\_query;

}

ll query(int l, int r) { // overloading

return query(l, r, 0, n - 1, 1);

}

ll update(int target, ll change, int l, int r, int idx) { // 가정 : idx 노드 표현 범위 -> [l, r]

if(target < l || target > r) return tree[idx]; // 범위 밖이라면 변동이 없으므로 기존의 데이터 바로 return

if(target == l && target == r) { // 목표물이라면 업데이트하고 반환

return tree[idx] = change;

}

int mid = l + (r - l)/2;

tree[idx] = update(target, change, l, mid, idx \* 2) + update(target, change, mid + 1, r, idx \* 2 + 1); // [l, mid] & (mid, r] 병합

return tree[idx];

}

ll update(int target, ll change) { // overloading

return update(target, change, 0, n - 1, 1);

}

};

* **Lazy propagation 세그먼트 트리:**

struct segment\_tree {

int n;

vector<ll> tree;

vector<ll> lazy;

segment\_tree(vector<ll>& arr) {

n = arr.size();

tree.resize(4 \* n);

lazy.resize(4 \* n, 0);

init(0, n - 1, 1, arr);

}

ll init(int sl, int sr, int idx, vector<ll>& arr) {

if(sl == sr) return tree[idx] = arr[sl];

int smid = sl + (sr - sl)/2;

return tree[idx] = init(sl, smid, idx \* 2, arr) + init(smid + 1, sr, idx \* 2 + 1, arr);

}

void propagate(int sl, int sr, int idx) { // lazy하게 쌓여있던 업데이트 & 전파

if(lazy[idx] != 0) {

tree[idx] += (ll)(sr - sl + 1) \* lazy[idx]; // 쌓여있던 lazy값 적용

if(sl != sr) { // 밑으로 lazy값 flush!

lazy[idx \* 2] += lazy[idx];

lazy[idx \* 2 + 1] += lazy[idx];

}

lazy[idx] = 0; // 초기화

}

}

ll query(int l, int r, int sl, int sr, int idx) {

propagate(sl, sr, idx); // 해당 위치의 노드를 최신 상태로 업데이트

if(r < sl || l > sr) return 0;

if(l <= sl && sr <= r) return tree[idx];

int smid = sl + (sr - sl)/2;

return query(l, r, sl, smid, idx \* 2) + query(l, r, smid + 1, sr, idx \* 2 + 1);

}

ll query(int l, int r) {

return query(l, r, 0, n - 1, 1);

}

ll update(int l, int r, ll value, int sl, int sr, int idx) {

propagate(sl, sr, idx); // 최신 상태로 업데이트

if(r < sl || l > sr) return 0;

if(l <= sl && sr <= r) {

tree[idx] += (ll)(sr - sl + 1) \* value; // 업데이트

// 밑의 자식에는 lazy하게 전파

if(sl != sr) { // <-- lazy 업데이트가 쌓여가는 곳

lazy[idx \* 2] += value;

lazy[idx \* 2 + 1] += value;

}

return tree[idx];

}

int smid = sl + (sr - sl)/2;

return tree[idx] = update(l, r, value, sl, smid, idx \* 2) + update(l, r, value, smid + 1, sr, idx \* 2 + 1);

}

ll update(int l, int r, int value) {

return update(l, r, value, 0, n - 1, 1);

}

};

* **K번째 수 찾기:**

vector<int> tree(MAX\_NUM \* 5, 0); // 초기에는 어떠한 원소도 저장 X

void update(int sl, int sr, int idx, int x) { // idx -> [sl, sr] 표현 <=> [sl, sr] 구간 내의 원소 개수

if(x < sl || x > sr) return; // 범위 밖이라면,

tree[idx]++; // 해당 범위내에 원소 개수 추가

if(sl == sr) return; // 리프 노드라면,

int smid = sl + (sr - sl)/2;

update(sl, smid, idx \* 2, x);

update(smid + 1, sr, idx \* 2 + 1, x);

}

void update(int x) {

update(1, MAX\_NUM, 1, x);

}

int query(int sl, int sr, int idx, int x) {

tree[idx]--; // 해당 범위 내에 있으므로 원소 개수 - 1

if(sl == sr) return sl; // 리프 노드이므로 x번째로 작은 원소 반환

int smid = sl + (sr - sl)/2;

return (x <= tree[idx \* 2]) ? query(sl, smid, idx \* 2, x) : query(smid + 1, sr, idx \* 2 + 1, x - tree[idx \* 2]);

}

int query(int x) {

return query(1, MAX\_NUM, 1, x);

}

* **트라이:**

const int ALPHABETS = 26;

int to\_number(char c) {return c - 'A';};

struct node { // 트라이 노드

node\* children[ALPHABETS];

bool terminal; // 리프 노드인가?

node() : terminal(false) {

memset(children, 0, sizeof(children));

}

~node() {

for(int i = 0; i < ALPHABETS; i++) {

if(children[i]) { // nullptr이 아니라면,

delete children[i];

}

}

}

void insert(const char\* key) { // 이 노드를 루트 노드로 하는 트라이에 key를 추가

if(\*key == 0) terminal = true;

else {

int next = to\_number(\*key);

// 해당 자식이 없다면 생성하기

if(children[next] == NULL) children[next] = new node();

children[next]->insert(key + 1); // 다음 문자로 시작

}

}

node\* find(const char\* key) { // 이 노드를 루트 노드로 하는 트라이에서 key에 대응되는 노드 찾기. 없다면 NULL 반환

if(\*key == 0) return this;

int next = to\_number(\*key);

if(children[next] == NULL) return NULL;

return children[next]->find(key + 1);

}

};

1. **알고리즘:**

* **다익스트라:**

int V, E;

vector<vector<pair<int, int>>> edges;

vector<int> d;

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<pair<int, int>>> Q;// STL의 'priority\_queue' 자료형은 디폴트가 max heap

/\* 필요한 자료구조 : 간선 정보 {[u] -> (v, w)}, 각 노드까지의 최단 경로 정보, 최단 경로에 대한 min heap {(d[i], i)} \*/

void init() {

edges.resize(V+1);

d.resize(V+1);

for(int i = 0; i < V + 1; i++) d[i] = INF;

}

void relaxation(int u, int v, int w) {

if(d[v] > d[u] + w) {

d[v] = d[u] + w;

Q.push(make\_pair(d[v], v));

}

}

void dijkstra(int s) {// time complexity: O(E \* T\_dk + V \* T\_min) = O((E + V) \* lg V)

d[s] = 0;

Q.push(make\_pair(d[s], s));

while(!Q.empty()) {

int curr = Q.top().second, curr\_dist = Q.top().first;

Q.pop();

if(curr\_dist > d[curr]) continue; // 구현과 이론과의 차이!! (우선순위 큐의 decrease-key 연산 시간 복잡도는 O(1)가 아니므로 중복 발생 O)

int adj\_size = edges[curr].size();

for(int i = 0; i < adj\_size; i++) {

int next = edges[curr][i].first;

relaxation(curr, next, edges[curr][i].second);// relaxation step & decrease key

}

}

}

* **다익스트라2:**

void dijkstra2(int src) {

vector<bool> visited(V + 1, false); // 1-based index

dist[src] = 0;

visited[src] = false;

parent[src] = -1; // 시작 노드는 부모가 없음

while (1) {

int closest = INF, here;

for (int i = 1; i <= V; i++) {

if (dist[i] < closest && !visited[i]) {

here = i;

closest = dist[i];

}

}

if (closest == INF) break;

visited[here] = true;

for (int i = 0; i < adj[here].size(); i++) {

int there = adj[here][i].first;

if (visited[there]) continue;

int next\_dist = dist[here] + adj[here][i].second;

if (next\_dist < dist[there]) {

dist[there] = next\_dist;

parent[there] = here; // 부모 노드 갱신

}

}

}

}

vector<int> get\_path(int target) {

vector<int> path;

for (int v = target; v != -1; v = parent[v]) {

path.push\_back(v);

}

reverse(path.begin(), path.end()); // 경로를 올바른 순서로 정렬

return path;

}

* **벨만-포드:**

vector<int> bellman\_ford(vector<vector<pii>>& edges, int V, int E, int start) {

vector<int> dist(V + 1, INF);

vector<int> parent(V + 1);

for(int i = 1; i <= V; i++) parent[i] = i; // parent 초기화

dist[start] = 0;

bool is\_updated = false;

for(int i = 0; i < V - 1; i++) { // |V| - 1번 수행

is\_updated = false;

for(int u = 1; u <= V; u++) {

for(int i = 0; i < edges[u].size(); i++) { // 모든 간선에 대하여 수행

int v = edges[u][i].first;

int w = edges[u][i].second;

if(dist[u] > dist[v] + w) { // relaxation 연산

dist[u] = dist[v] + w;

parent[u] = v;

is\_updated = true;

}

}

}

if(!is\_updated) break;

}

return parent;

}

* **플로이드-워셜:**

void get\_path(vector<int>& path, vector<vector<int>>& via, int n, int u, int v) { // u -> v

if(via[u][v] == -1) {

path.push\_back(u);

if(u != v) path.push\_back(v);

}

else {

int mid = via[u][v];

get\_path(path, via, n, u, mid); // u -> mid

path.pop\_back(); // mid 노드 중복 방지

get\_path(path, via, n, mid, v); // mid -> v

}

}

void floyd\_warshall(vector<vector<int>>& adj, int n) {

vector<vector<int>> via(n + 1, vector<int>(n + 1, -1)); // u -> v 경로에서 경유하는 노드들 중 가장 큰 번호를 가진 노드(즉, 가장 마지막에 업데이트된 노드)

for(int k = 1; k <= n; k++) {

for(int i = 1; i <= n; i++) {

for(int j = 1; j <=n; j++) {

if((i != j) && adj[i][k] != INF && adj[k][j] != INF && (adj[i][j] > adj[i][k] + adj[k][j])) {// 만약 i -> v에서 k 노드를 경유해야 한다면,

adj[i][j] = adj[i][k] + adj[k][j];

via[i][j] = k; // i -> j 경로에서 k 노드를 경유함. (경유하는 노드들 중 번호가 가장 큼.)

}

}

}

}

// 최단 거리 출력

for(int i = 1; i <= n; i++) {

for(int j = 1; j <= n; j++) {

if(adj[i][j] == INF) cout<<"0 ";

else cout<<adj[i][j]<<" ";

}

cout<<"\n";

}

// 최단 경로 출력

for(int i = 1; i <= n; i++) {

for(int j = 1; j <= n; j++) {

if(adj[i][j] == INF) cout<<"0\n";

else {

vector<int> path;

get\_path(path, via, n, i, j);

cout<<path.size()<<" ";

for(auto e : path) cout<<e<<" ";

cout<<"\n";

}

}

}

}

* **크루스칼:**

int V, E;

vector<int> parent;

vector<pair<int, pair<int, int>>> edges;

vector<pair<int, int>> mst\_edges; // MST를 구성하는 간선을 저장

long long int ans = 0;

void initialize() {

parent.resize(V + 1);

for(int i = 1; i <= V; i++) parent[i] = i;

}

int find\_root(int u) {

if(u == parent[u]) return u;

else return parent[u] = find\_root(parent[u]);

}

void kruskal\_algorithm() {

sort(edges.begin(), edges.end());

for(auto it : edges) {

int cost = it.first;

int u = it.second.first;

int v = it.second.second;

int root1 = find\_root(u);

int root2 = find\_root(v);

if(root1 != root2) {

if(root1 < root2) parent[root2] = root1;

else parent[root1] = root2;

ans += cost;

mst\_edges.push\_back({u, v}); // MST 간선을 저장

}

}

}

* **프림:**

int prim(int n, vector<vector<pair<int, int>>>& adj) {

vector<int> min\_weight(n, INF); // MST에 추가하는 데 드는 비용의 최솟값 저장

vector<bool> inMST(n, false); // MST 포함 여부

vector<int> parent(n, -1); // 각 노드의 부모 노드 저장 (역추적용)

priority\_queue<pair<int, int>, vector<pair<int, int>>, greater<>> pq; // (weight, node)

int sum = 0;

// 시작 노드 설정

min\_weight[0] = 0;

pq.push({0, 0});

while (!pq.empty()) {

int u = pq.top().second;

pq.pop();

if (inMST[u]) continue;

inMST[u] = true;

sum += min\_weight[u];

// 인접 노드 처리

for (auto &[v, weight] : adj[u]) {

if (!inMST[v] && weight < min\_weight[v]) {

min\_weight[v] = weight;

pq.push({weight, v});

parent[v] = u; // 부모 노드 설정

}

}

}

// MST 구성 간선 출력

cout << "Edges in the MST:\n";

for (int i = 1; i < n; i++) { // 0번 노드 제외

if (parent[i] != -1) {

cout << parent[i] << " - " << i << " (Weight: " << min\_weight[i] << ")\n";

}

}

return sum;

}

* **프림2:**

vector<int> min\_weight(n, INF); // MST에 추가하는 데 드는 비용의 최솟값 저장

vector<bool> visited(n, false); // MST 포함 여부

vector<int> parent(n, -1); // 각 노드의 부모 노드 저장 (역추적용)

int sum = 0;

min\_weight[0] = 0; // 시작 노드 비용 0 설정

for(int iter = 0; iter < n; iter++) { // n개의 간선 선택

int next = -1;

// 최소 가중치 간선 선택

for(int i = 0; i < n; i++) {

if(!visited[i] && (next == -1 || min\_weight[i] < min\_weight[next])) next = i;

}

visited[next] = true;

sum += min\_weight[next];

// 인접 노드 갱신

for(int i = 0; i < n; i++) {

if(!visited[i] && dist[next][i] < min\_weight[i]) {

min\_weight[i] = dist[next][i];

parent[i] = next; // 부모 노드 설정

}

}

}

* **TSP:**

int N;

cin>>N;

vector<vector<int>> dist(N, vector<int>(N));

for(int i = 0; i < N; i++) {

for(int j = 0; j < N; j++) {

cin>>dist[i][j];

}

}

vector<vector<int>> dp(N, vector<int>(1 << N, INF)); // 각 자리수 당 한 개의 도시

// [i][mask] : 현재 i 도시에 있고 mask 상태로 도시들을 방문 완료하는데 드는 최소 비용

dp[0][1] = 0; // 출발점

for(int mask = 1; mask < (1 << N); mask++) { // 모든 상태에 대하여 탐색 (0번 도시는 이미 탐색함.)

for(int curr = 0; curr < N; curr++) { // 현재 위치

if(!(mask & (1 << curr))) continue; // 방문하지 않은 곳이라면,

for(int next = 0; next < N; next++) {

if(mask & (1 << next)) continue; // 이미 방문한 상태라면,

if(dist[curr][next] != 0) dp[next][mask | (1 << next)] = min(dp[next][mask | (1 << next)], dp[curr][mask] + dist[curr][next]);

}

}

}

// 마지막으로 돌아오는 비용

int ans = INF;

int mask = (1 << N) - 1;

for(int i = 1; i < N; i++) {

if(dist[i][0] != 0) ans = min(ans, dp[i][mask] + dist[i][0]);

}

cout<<ans;

* **위상정렬:**

int N, M;

vector<vector<int>> adj;

vector<bool> visited;

/\*

void topological\_sort() { // bfs 방식

queue<int> Q;

for(int i = 1; i <= N; i++) if(in\_degree[i] == 0) Q.push(i); // 위상 정렬 시작 전,

for(int iter = 0; iter < N; iter++) {

if(Q.empty()) {

cout<<"Cycle detected!\n";

return;

}

int curr = Q.front();

Q.pop();

cout<<curr<<" ";

for(auto next : adj[curr]) {

in\_degree[next]--;

if(in\_degree[next] == 0) Q.push(next);

}

}

}

\*/

void dfs(int curr) { // dfs 방식

visited[curr] = true;

for(auto next : adj[curr]) {

if(!visited[next]) dfs(next);

}

cout<<curr<<" ";

}

void dfs\_all() {

visited.resize(N + 1, false); // 초기화

for(int i = 1; i <= N; i++) {

if(!visited[i]) dfs(i);

}

}