

Team Note of GeongongiIsMae

Junseo Park, Jimin Kim, Mingi Jeong

2025 ICPC Seoul Regional (BEXCO, Busan)

Contents

1 Python

1.1	정렬 비교 함수 구현	1
1.2	itertools	1

2 STL

2.1	set	2
2.2	map	2
2.3	priority_queue	2
2.4	bitset	2
2.5	permutation	2

3 정렬/이분 탐색 관련

3.1	정렬 비교 함수 구현	1
3.2	좌표 압축	1
3.3	이분 탐색 관련 STL	1

4 자료구조

4.1	PBDS	1
4.2	Erasable Priority Queue	1
4.3	펜윅 트리(BIT)	1
4.4	세그먼트 트리	1
4.5	세그먼트 트리 + 레이지 프로퍼게이션	1
4.6	파시스턴트 세그먼트 트리	1
4.7	화성 지도 세그	1
4.8	금광 세그(최대 부분합 세그)	1
4.9	머지 소트 트리	1
4.10	단조 큐(Monotone Queue)	1
4.11	Convex Hull Trick (Stack, LineContainer)	1

5 DP

5.1	$O(N \log N)$ LIS	1
5.2	Berlekamp-Massey, 카타마사법	1
5.3	SOS(Sum Over Subsets) DP	1
5.4	$O(N \times \max W)$ Subset Sum (Fast Knapsack)	1
5.5	DP Optimization, Knuth Optimization	1
5.6	Monotone Queue Optimization	1
5.7	Slope Trick	1
5.8	Aliens Trick	1

6 그래프

6.1	최단 거리 - Floyd Warshall	1
6.2	최단 거리 - Dijkstra	1
6.3	최단 거리 - Bellman Ford	1
6.4	유니온 파인드 + 최소 신장 트리(Kruskal)	1
6.5	Euler Tour	1
6.6	SCC - Kosaraju	1
6.7	BCC - Tarjan	1

6.8	최대 유량 - Dinic	8
6.9	MCMF	8
6.10	이분 매칭	9
6.11	이분 매칭 - Hopcroft Karp	9
6.12	플로우 관련 정리	9
6.13	$O(V^3)$ Hungarian Method	9
6.14	$O(V^3)$ General Matching	9
6.15	$O(V^3)$ Weighted General Matching	9
6.16	안정 결혼 문제	10
6.17	최소 공통 조상(LCA)	10
6.18	Heavy Light Decomposition	11
6.19	센트로이드 트리	11
7	수학	11
7.1	바닥함수, 천장함수	11
7.2	빠른 거듭제곱	11
7.3	Power Tower	11
7.4	Harmonic Lemma	11
7.5	FloorSum	11
7.6	Linear Sieve	11
7.7	확장 유클리드 알고리즘	12
7.8	중국인의 나머지 정리	12
7.9	Diophantine	12
7.10	Miller Rabin + Pollard Rho	12
7.11	원시근, 이산로그, 이산제곱근	12
7.12	$O(N^3 \log 1/\epsilon)$ Polynomial Equation	13
7.13	가우스 소거법 - RREF, 랭크, 행렬식, 역행렬	13
7.14	다항식 곱셈(FFT)	13
7.15	이항 계수를 소수로 나눈 나머지(Lucas' theorem)	13
7.16	이항 계수 관련 공식	14
7.17	Partition Number	14
7.18	De Bruijn Sequence	14
7.19	자그재그 순열의 개수 구하기	14
8	문자열	14
8.1	정규 표현식	14
8.2	문자열 해싱	14
8.3	문자열 매칭 - KMP	15
8.4	가장 긴 팰린드롬 부분 문자열 - Manacher	15
8.5	문자열 매칭 - Z	15
8.6	Aho-Corasick	15
8.7	접미사 배열	15
8.8	All LCS	15
9	계산 기하	15
9.1	2차원 계산 기하 템플릿 + CCW	15
9.2	360도 각도 정렬	16
9.3	다각형 넓이	16
9.4	선분 교차 판정	16
9.5	Intersect Series	16
9.6	Segment Distance, Segment Reflect	16
9.7	다각형 내부 판별	17
9.8	볼록 껍질 - Graham Scan	17
9.9	가장 면 두 점 - Rotating Calipers	17
9.10	볼록 다각형 내부 판별	17
9.11	Segment In Polygon	17
9.12	Polygon Cut, Center, Union	17
9.13	Polycon Raycast	17
9.14	$O(N \log N)$ Shamos-Hoey	18
9.15	$O(N \log N)$ 반평면 교집합	18
9.16	$O(N^2 \log N)$ Bulldozer Trick(Rotating Sweep Line Technique)	18
9.17	$O(N)$ 최소 외접원	18
10	Misc	19
10.1	삼분 탐색	19
10.2	Mo's Algorithm	19
10.3	C++ 랜덤, GCC 확장, 비트마스킹 트릭	19
10.4	Floating Point Add (Kahan)	19
10.5	Fast I/O, Fast Div, Fast Mod	19
10.6	Python Decimal	19
11	Notes	19
11.1	구간별 약수 최대 개수, 최대 소수	19
11.2	비트 연산	20
11.3	Triangles/Trigonometry	20
11.4	Calculus, Newton's Method	20
11.5	Zeta/Mobius Transform	20
11.6	Generating Function	20
11.7	Counting	20
11.8	Faulhaber's Formula ($\sum_{k=1}^n k^c$)	21
11.9	About Graph Degree Sequence	21
11.10	Burnside, Grundy, Pick, Hall, Simpson, Area of Quadrangle, Fermat Point, Euler, Pythagorean	21
11.11	About Graph Minimum Cut	21
11.12	Matrix with Graph(Kirchhoff, Tutte, LGV)	21
11.13	About Graph Matching(Graph with $ V \leq 500$)	21
11.14	Checklist	22
1	Python	
1.1	정렬 비교 함수 구현	
	Usage: 첫 번째가 두 번째보다 작으면(less-than) 음수 값(유지), 같으면 0, 크면(greater-than) 양수 값(교체)	
	# (예제) 큰 수 만들기	
	def f(a, b): return(int(a+b)<int(b+a))-(int(a+b)>int(b+a)) #int(b+a)가 크면 1,int(a+b)가 크면 -1, 같으면 0	
	import functools A = list(input().split()) print(int(''.join(sorted(A, key=functools.cmp_to_key(f)))))	
1.2	itertools	
	import itertools A = [1,2,3] itertools.permutations(A) # 순열 # [(1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 1, 2), (3, 2, 1)] itertools.permutations(A,2) # [(1, 2), (1, 3), (2, 1), (2, 3), (3, 1), (3, 2)] itertools.combinations(A, 2) # 조합 # [(1, 2), (1, 3), (2, 3)] itertools.combinations_with_replacement(A, 2) # 중복조합 # [(1, 1), (1, 2), (1, 3), (2, 2), (2, 3), (3, 3)] B=['a','b'] itertools.product(A,B) # 데카르트 곱(cartesian product) # [(1, 'a'), (1, 'b'), (2, 'a'), (2, 'b'), (3, 'a'), (3, 'b')]	

2 STL

2.1 set

```
#include <bits/stdc++.h>
using namespace std;

template<typename T>
struct Cmp {
    bool operator() (const T& a, const T& b) const {
        if (abs(a) != abs(b)) return abs(a) < abs(b);
        return a > b;
    }
};

int main() {
    set<int, greater<int>> S1 = { 1, 2, 3, 4 }; // descending order
    for (auto& i : S1) cout << i << ' '; cout << '\n'; // 4 3 2 1

    set<int, Cmp<int>> S2 = { 3, -1, -4, 1, 5, 1 }; // custom compare function
    for (auto& i : S2) cout << i << ' '; cout << '\n'; // 1, -1, 3, -4, 5
}

int main() {
    set<int> S = { 1, 2 }; // initializer_list
    auto [it1, flag1] = S.insert(2); // S = { 1, 2 }, *it1 == 2, flag1 == 0
    auto [it2, flag2] = S.insert(3); // S = { 1, 2, 3 }, *it2 == 3, flag2 == 1
    it1 = S.erase(it1); // S = { 1, 3 }, *it1 == 3
    auto cnt = S.erase(3); // S = { 1 }, cnt == 1

    cout << "S.size() : " << S.size() << '\n'; // 1
    cout << "S.empty() : " << S.empty() << '\n'; // 0
    S.clear();

    for (auto& i : { 3, 1, 4, 1, 5 }) S.insert(i); // S = { 1, 3, 4, 5 }
    for (auto& i : S) cout << i << ' '; cout << '\n'; // 1 3 4 5

    if (S.count(2)) cout << "found 2" << '\n'; // x
    if (S.count(3)) cout << "found 3" << '\n'; // o

    auto lo = S.lower_bound(3); // *lo == 3, *prev(lo) < 3 <= *lo
    auto hi = S.upper_bound(3); // *lo == 4, *prev(lo) <= 3 < *lo
}
```

2.2 map

```
int main() {
    fastio;
    map<string, int> M;
    M["abc"] = 3; // == M.insert({ "abc", 3 })
    cout << M["abc"] << '\n'; // 3
    M["abc"] = 4;
    cout << M["abc"] << '\n'; // 4

    auto it = M.erase("abc"); // M = { }, it == M.end()
    M["hi"] = 123; // M = { { "hi", 123 } }
    auto cnt = M.erase("hi"); // M = { }, cnt == 1
}
```

```
cout << "M.size() : " << M.size() << '\n'; // 0
cout << "M.empty() : " << M.empty() << '\n'; // 1
M.clear();

vector<string> s{ "ab", "cde", "fghi" };
vector<int> v{ 3, 1, 4 };
for (int i = 0; i < 3; i++) M[s[i]] = v[i];
for (auto& [a, b] : M) cout << '(' << a << ', ' << b << ')' << ' ';
cout << '\n'; // (ab,3) (cde,1) (fghi,4)

if (M.count("abc")) cout << "found abc" << '\n'; // x
if (M.count("cde")) cout << "found cde" << '\n'; // o
}
```

2.3 priority_queue

```
#include <queue>
using namespace std;
// 최대 힙 우선순위 큐
priority_queue<int, vector<int>> q1;
priority_queue<int, vector<int>, less<int>> q2;
// 최소 힙 우선순위 큐
priority_queue<int, vector<int>, greater<int>> q;
// 멤버 함수: push, pop, top, size, empty

// priority_queue의 정렬 기준을 지정하는 여러 가지 방법
struct Info {
    int val;
    bool operator< (const Info& i) const {
        return val > i.val; // val이 가장 작은게 최댓값
    }
};
priority_queue<Info> PQ;

struct Cmp {
    bool operator() (const int& a, const int& b) const {
        return a > b; // 가장 작은게 최댓값
    }
};
priority_queue<int, vector<int>, Cmp> PQ;

auto Cmp = [] (const int& a, const int& b) { return a > b; }; // 가장 작은게 최댓값
priority_queue<int, vector<int>, decltype(Cmp)> PQ(Cmp);

bool Cmp(const int& a, const int& b) {
    return a > b; // 가장 작은게 최댓값
}
priority_queue<int, vector<int>, decltype(&Cmp)> PQ(Cmp);
```

2.4 bitset

Usage: 낭비되는 7bit를 없애기 위해서 0, 1을 하나 당 1bit에 저장하는 자료형

```
int main() {
    bitset<10> B1; // B1 = "0000000000"
    bitset<10> B2(13); // B2 = "0000001101"
    bitset<10> B3("1011"); // B3 = "0000001011"
    cout << B1 << '\n' << B2 << '\n' << B3 << '\n';
}
```

```
bitset<5> B4("01011");
bitset<5> B5("00111");
cout << (B4 & B5) << '\n'; // 00011
cout << (B4 | B5) << '\n'; // 01111
cout << (B4 ^ B5) << '\n'; // 01100
cout << (B4 << 2) << '\n'; // 01100
cout << (B4 >> 2) << '\n'; // 00010
cout << ~B4 << '\n'; // 10100

bitset<100> B;
B.set(); cout << B.count() << '\n'; // 100
B.reset(); cout << B.count() << '\n'; // 0
cout << B.size() << '\n' << '\n'; // 100
```

```
B[2] = 1; // B = "...100"
cout << B.count() << '\n'; // 1
cout << B.any() << '\n'; // 1
cout << B.all() << '\n'; // 0
cout << B.none() << '\n' << '\n'; // 0

B[2].flip(); // B = "...000"
cout << B.count() << '\n'; // 0
cout << B.any() << '\n'; // 0
cout << B.all() << '\n'; // 0
cout << B.none() << '\n' << '\n'; // 1
```

```
B[1] = B[3] = B[4] = 1;
string s = B.to_string();
cout << s << '\n'; // ...11010
}
```

2.5 permutation

```
#include <bits/stdc++.h>
using namespace std;

int main() {
    vector<int> v{ 1, 2, 3, 4 };
    do {
        for (auto& i : v) cout << i << ' ';
        cout << '\n';
    } while (next_permutation(v.begin(), v.end()));

    string s = "dcba";
    do cout << s << '\n';
    while (prev_permutation(s.begin(), s.end()));
}
```

3 정렬/이분 탐색 관련

3.1 정렬 비교 함수 구현

```
struct Point{
    int x, y;
    // 방법 1: 연산자 오버로딩
    // x좌표 오름차순, x좌표 같으면 y좌표 오름차순
    // sort(시작 주소, 끝 주소)
    bool operator < (const Point &p) const {
        if(x != p.x) return x < p.x;
        else return y < p.y;
    }
}
```

```

};

// 방법 2: 비교 함수 구현
// sort(시작 주소, 끝 주소, Compare)
bool Compare(const Point &a, const Point &b){
    if(a.x != b.x) return a.x < b.x;
    else return a.y < b.y;
}

```

```

vector<Point> V;
V.push_back({1, 2});
V.push_back({1, 1});
V.push_back({2, 3});
sort(V.begin(), V.end()); // 방법 1
sort(V.begin(), V.end(), Compare); // 방법 2

```

3.2 좌표 압축

```

// 원소의 대소관계를 유지하면서 [0, N] 범위의 수로 압축함
// ex. {50, 31, 24, 10, 46, 10} -> {4, 2, 1, 0, 3, 0}
int N = 6, A[6] = {50, 31, 24, 10, 46, 10};
vector<int> C;
for(int i=0; i<N; i++) C.push_back(A[i]);
sort(C.begin(), C.end());
C.erase(unique(C.begin(), C.end()), C.end());
for(int i=0; i<N; i++){
    A[i] = lower_bound(C.begin(), C.end(), A[i]) - C.begin();
}
for(int i=0; i<N; i++) cout << A[i] << " "; // 4 2 1 0 3 0

```

3.3 이분 탐색 관련 STL

```

vector<int> v = {3, 5, 7, 7, 7, 10};

// lower_bound: x 이상인 가장 빠른 위치
// 배열은 정렬되어 있어야 함, O(log N)
for(int i=3; i<=8; i++){
    cout << lower_bound(v.begin(), v.end(), i) - v.begin() << " ";
} // 0 1 1 2 2 5
cout << "\n";

// upper_bound: x 초과인 가장 빠른 위치
// 배열은 정렬되어 있어야 함, O(log N)
for(int i=3; i<=8; i++){
    cout << upper_bound(v.begin(), v.end(), i) - v.begin() << " ";
} // 1 1 2 2 5 5
cout << "\n";

```

```

// binary_search: x가 있으면 true, 없으면 false
// 배열은 정렬되어 있어야 함, O(log N)
for(int i=3; i<=8; i++){
    cout << binary_search(v.begin(), v.end(), i) << " ";
} // 1 0 1 0 1 0
cout << "\n";

```

```

vector<int> a = {1, 3, 5, 2, 4, 6};

```

```

// nth_element(a.begin(), a.begin()+k, a.end())
// 정렬했을 때 a[k]에 와야 하는 수가 a[k]에 옴
// a[k] 미만의 수는 모두 a[0..k-1]으로 이동

```

```

// a[k] 초과의 수는 모두 a[k+1..]으로 이동
// 평균 시간 복잡도 O(N), 최악 시간 복잡도 O(N log N)
nth_element(a.begin(), a.begin()+3, a.end());
for(auto i : a) cout << i << " "; // a b c 4 d e
// a b c <= 4, d e >= 4

```

4 자료구조

4.1 PBDS

Usage: order_of_key(NUM) : ordered_set에서 NUM 보다 작은(미만의) 원소의 개수를 반환한다.
find_by_order(K) : ordered_set에서 (K+1)번째 원소가 있는 iterator 을 반환한다. (K가 0이면 1번째)

```

#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
#define ordered_set tree<int, null_type, less<int>,
rb_tree_tag,tree_order_statistics_node_update>

```

4.2 Erasable Priority Queue

```

template<class T=int, class O=less<T>>
struct pq_set {
    priority_queue<T, vector<T>, O> q, del;
    const T& top() const { return q.top(); }
    int size() const { return int(q.size())-del.size(); }
    bool empty() const { return !size(); }
    void insert(const T x) { q.push(x); flush(); }
    void pop() { q.pop(); flush(); }
    void erase(const T x) { del.push(x); flush(); }
    void flush() { while(del.size() && q.top()==del.top())
q.pop(), del.pop(); }
};

```

4.3 펜윅 트리(BIT)

Time Complexity: $O(\log N)$

```

SIZE = 1<<17
tree = [0]*SIZE

```

```

# 1-based
def update(i, diff):
    while i < SIZE:
        tree[i] += diff
        i += i&-i

```

```

def getSum(i):
    s = 0
    while i > 0:
        s += tree[i]
        i -= i&-i
    return s

```

4.4 세그먼트 트리

Time Complexity: $O(\log N)$

```

#include <bits/stdc++.h>
#define fastio cin.tie(0)->sync_with_stdio(0)
using namespace std;
using ll = long long;
// 13만-> 1 << 17 (131072), 26만-> 1 << 18 (262144)

```

```

// 52만-> 1 << 19 (524288), 100만-> 1 << 20 (1048576)
const int SIZE = 1<<20;
ll tree[SIZE*2];
ll arr[SIZE];

```

```

ll merge(ll a, ll b){
    return a+b;
}

```

// k번째 수를 반환

```

int getkth(int k, int node, int S, int E){
    if(S==E) return S;
    int mid = (S+E)/2;
    if(tree[node*2]>=k) return getkth(k, node*2, S, mid);
    else return getkth(k-tree[node*2], node*2+1, mid+1, E);
}

```

// x번째 수를 V로 지정

```

void update(int X, ll V, int node, int S, int E){
    if(S==E){
        tree[node] = V;
        return;
    }
    int mid = (S+E)/2;
    if(X<=mid) update(X, V, node*2, S, mid);
    else update(X, V, node*2+1, mid+1, E);
    tree[node] = merge(tree[node*2], tree[node*2+1]);
}

```

// [L, R] 구간 쿼리

```

ll query(int L, int R, int node, int S, int E){
    if(R<S || E<L) return 0; // 항등원
    if(L<=S && E<=R) return tree[node];
    int mid = (S+E)/2;
    return merge(query(L, R, node*2, S, mid), query(L, R,
node*2+1, mid+1, E));
}

```

// S-based indexing으로 초기화

```

ll init(int node, int S, int E){
    if(S==E) return tree[node] = arr[S];
    int mid = (S+E)/2;
    return tree[node] = merge(init(node*2, S, mid),
init(node*2+1, mid+1, E));
}

```

```

int main(){
    fastio;
    int n, m, k;
    cin >> n >> m >> k;
    for(int i=1; i<=n; i++) cin >> arr[i];
    init(1, 1, n);
    for(int i=0; i<m+k; i++){
        ll a, b, c;
        cin >> a >> b >> c;
        if(a==1) update(b, c, 1, 1, n);
        else cout << query(b, c, 1, 1, n) << '\n';
    }
}

```

4.5 세그먼트 트리 + 레이지 프로퍼게이션

Time Complexity: $O(\log N)$

```
// SZ: N보다 크거나 같은 2^k 꼴의 수
// 13만 -> 1 << 17 (131072), 26만 -> 1 << 18 (262144)
// 52만 -> 1 << 19 (524288), 100만 -> 1 << 20 (1048576)
constexpr int SZ = 1 << 20;
ll T[SZ<<1], L[SZ<<1];

void Push(int node, int s, int e){
    if(L[node] == 0) return;
    T[node] += (e - s + 1) * L[node];
    if(s != e) L[node*2] += L[node], L[node*2+1] += L[node];
    L[node] = 0;
}

// [l, r]번째 수에 v를 더함, 0 <= l <= r < SZ
void RangeAdd(int l, int r, ll v, int node=1, int s=0, int e=SZ-1){
    Push(node, s, e);
    if(r < s || e < l) return;
    if(l <= s && e <= r){ L[node] += v; Push(node, s, e); return; }
    int m = (s + e) / 2;
    RangeAdd(l, r, v, node*2, s, m);
    RangeAdd(l, r, v, node*2+1, m+1, e);
    T[node] = T[node*2] + T[node*2+1];
}

// [l, r]번째 수의 합을 구함
ll RangeSum(int l, int r, int node=1, int s=0, int e=SZ-1){
    Push(node, s, e);
    if(r < s || e < l) return 0;
    if(l <= s && e <= r) return T[node];
    int m = (s + e) / 2;
    return RangeSum(l, r, node*2, s, m) + RangeSum(l, r, node*2+1, m+1, e);
}
```

4.6 퍼시스턴트 세그먼트 트리

Usage: call init(root[0], s, e) before use

```
struct PSTNode{
    PSTNode *l, *r; int v;
    PSTNode(){ l = r = nullptr; v = 0; }
};

PSTNode *root[101010];
PSTNode(){} memset(root, 0, sizeof root); // constructor
void init(PSTNode *node, int s, int e){
    if(s == e) return;
    int m = s + e >> 1;
    node->l = new PSTNode; node->r = new PSTNode;
    init(node->l, s, m); init(node->r, m+1, e);
}

void update(PSTNode *prv, PSTNode *now, int s, int e, int x){
    if(s == e){ now->v = prv ? prv->v + 1 : 1; return; }
    int m = s + e >> 1;
    if(x <= m){
        now->l = new PSTNode; now->r = prv->r;
        update(prv->l, now->l, s, m, x);
    }
}
```

```
else{
    now->r = new PSTNode; now->l = prv->l;
    update(prv->r, now->r, m+1, e, x);
}
int t1 = now->l ? now->l->v : 0;
int t2 = now->r ? now->r->v : 0;
now->v = t1 + t2;
}

int kth(PSTNode *prv, PSTNode *now, int s, int e, int k){
    if(s == e) return s;
    int m = s + e >> 1, diff = now->l->v - prv->l->v;
    if(k <= diff) return kth(prv->l, now->l, s, m, k);
    else return kth(prv->r, now->r, m+1, e, k-diff);
}
```

4.7 화성 지도 세그

```
const int SIZE = 1<<16;
int tree[SIZE*2];
int len[SIZE*2];

void update(int L, int R, int V, int node, int S, int E){
    if(R < S || E < L) return;
    if(L <= S && E <= R) tree[node] += V;
    else{
        int mid = (S+E)/2;
        update(L, R, V, node*2, S, mid);
        update(L, R, V, node*2+1, mid+1, E);
    }
    if(tree[node]>0) len[node] = E-S+1;
    else if(S==E) len[node] = 0;
    else len[node] = len[node*2]+len[node*2+1];
}

int query(){return len[1];}
```

4.8 금광 세그(최대 부분합 세그)

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INF = 1e9;
struct Node {
    ll LMax; // 왼쪽에서 시작한 부분합 중 최댓값
    ll RMax; // 오른쪽에서 시작한 부분합 중 최댓값
    ll Max; // 그냥 최댓값
    ll Sum; // 그냥 합
    Node() {
        LMax = RMax = Max = Sum = -INF;
    }
    void setVal(ll x) {
        LMax = RMax = Max = Sum = x;
    }
};
int N;
ll arr[100001];
Node SegTree[400001];

Node merge(Node lNode, Node rNode) {
    Node ret;
    ret.LMax = max(lNode.LMax, lNode.Sum + rNode.LMax);
    ret.RMax = max(rNode.RMax, lNode.RMax + rNode.Sum);
    ret.Max = max(lNode.Max, rNode.Max, lNode.RMax + rNode.LMax);
    ret.Sum = lNode.Sum + rNode.Sum;
    return ret;
}
```

```
void init(int idx, int st, int ed) {
    if(st == ed) {
        SegTree[idx].setVal(arr[st]);
        return;
    }
    int mid = (st+ed)/2;
    init(2*idx, st, mid);
    init(2*idx+1, mid+1, ed);
    SegTree[idx] = merge(SegTree[2*idx], SegTree[2*idx+1]);
}

Node query(int idx, int nodeSt, int nodeEd, int reqSt, int reqEd) {
    if(reqEd < nodeSt || nodeEd < reqSt) return Node();
    else if (reqSt <= nodeSt && nodeEd <= reqEd) return SegTree[idx];
    else {
        int nodeMid = (nodeSt + nodeEd)/2;
        Node left = query(2*idx, nodeSt, nodeMid, reqSt, reqEd);
        Node right =
            query(2*idx+1, nodeMid+1, nodeEd, reqSt, reqEd);
        return merge(left, right);
    }
}

int main() {
    cin.tie(0)->sync_with_stdio(0);
    cin >> N;
    for (int i=0;i<N;i++) cin >> arr[i];
    init(1,0,N-1);
    int T; cin >> T;
    while (T--) {
        int a, b;
        cin >> a >> b;
        a--; b--;
        Node ret = query(1,0,N-1,a,b);
        cout << ret.Max << '\n';
    }
    return 0;
}
```

4.9 머지 소트 트리

Time Complexity: $O((\log N)^2)$

```
const int SIZE = 1<<17;
vector<int> tree[SIZE*2];
int arr[SIZE];

void init(int node, int S, int E){
    if(S==E){
        tree[node].push_back(arr[S]);
        return;
    }
    int mid = (S+E)/2;
    init(node*2, S, mid);
    init(node*2+1, mid+1, E);
}
```

```

init(node*2+1, mid+1, E);
vector<int> &c = tree[node], &l = tree[node*2], &r =
tree[node*2+1];
c.resize(l.size()+r.size());
for(int j=0, p=0, q=0; j<c.size(); j++){
    if(q==r.size() || (p<l.size() && l[p]<r[q])) c[j] =
l[p++];
    else c[j] = r[q++];
}
}

// [L, R] 구간에서 k보다 큰 원소의 개수 반환
int query(int L, int R, int k, int node, int S, int E){
    if(R<S || E<L) return 0;
    if(L==S && E==R) return
tree[node].end()-upper_bound(tree[node].begin(),
tree[node].end(), k);
    int mid = (S+E)/2;
    return query(L, R, k, node*2, S, mid)+query(L, R, k,
node*2+1, mid+1, E);
}

```

4.10 단조 큐(Monotone Queue)

Usage: 수열을 순회하면서 구간에 대해 최솟값을 찾는다.
Time Complexity: $O(N)$

```

from collections import deque
import sys
input = sys.stdin.readline
N, L = map(int, input().split())
a = [*map(int, input().split())]
chain = deque()
ans = [0]*N
for r in range(N):
    while chain and a[chain[-1]]>a[r]: chain.pop()
    chain.append(r)
    if r>=L and chain[0]==r-L: chain.popleft()
    ans[r] = a[chain[0]]
print(*ans)

```

4.11 Convex Hull Trick (Stack, LineContainer)

```

struct Line{ // call init() before use
    ll a, b, c; // y = ax + b, c = line index
    Line(ll a, ll b, ll c) : a(a), b(b), c(c) {}
    ll f(ll x){ return a * x + b; }
};

vector<Line> v; int pv;
void init(){ v.clear(); pv = 0; }
int chk(const Line &a, const Line &b, const Line &c) const {
    return (_int128_t)(a.b - b.b) * (b.a - c.a) <=
(_int128_t)(c.b - b.b) * (b.a - a.a);
}
void insert(Line l){
    if(v.size() > pv && v.back().a == l.a){ // fix if min query
        if(l.b < v.back().b) l = v.back(); v.pop_back();
    }
    while(v.size() >= pv+2 && chk(v[v.size()-2], v.back(), l))
v.pop_back();
    v.push_back(l);
}

```

```

p query(ll x){ // if min query, then v[pv].f(x) >= v[pv+1].f(x)
    while(pv+1 < v.size() && v[pv].f(x) <= v[pv+1].f(x)) pv++;
    return {v[pv].f(x), v[pv].c};
}
//// line container start (max query) /////
struct Line {
    mutable ll k, m, p;
    bool operator<(const Line& o) const { return k < o.k; }
    bool operator<(ll x) const { return p < x; }
}; // (for doubles, use inf = 1/.0, div(a,b) = a/b)
struct LineContainer : multiset<Line, less<> {
    static const ll inf = LLONG_MAX; // div: floor
    ll div(ll a, ll b) { return a / b - ((a ^ b) < 0 && a % b); }
    bool isect(iterator x, iterator y) {
        if (y == end()) return x->p = inf, 0;
        if (x->k == y->k) x->p = x->m > y->m ? inf : -inf;
        else x->p = div(y->m - x->m, x->k - y->k);
        return x->p >= y->p;
    }
    void add(ll k, ll m) {
        auto z = insert({k, m, 0}), y = z++, x = y;
        while (isect(y, z)) z = erase(z);
        if (x != begin() && isect(--x, y)) isect(x, y = erase(y));
        while ((y = x) != begin() && (--x)->p >= y->p) isect(x,
erase(y));
    }
    ll query(ll x) { assert(!empty());
        auto l = *lower_bound(x); return l.k * x + l.m; }
};


```

5 DP

5.1 $O(N \log N)$ LIS

```

# 길이만 구하기
from bisect import bisect_left
N = int(input())
A = [*map(int, input().split())]
L = []
for num in A:
    i = bisect_left(L, num)
    if i < len(L): L[i] = num
    else: L.append(num)
print(len(L))

```

```

# 수열 구하기
N = int(input())
A = [*map(int, input().split())]
lis = []
idx = 0
index = [0]*N
for i in range(N):
    k = bisect_left(lis, A[i])
    if k < idx:
        lis[k] = A[i]
        index[i] = k
    else:
        lis.append(A[i])
        index[i] = idx
        idx += 1
print(idx) # 길이
ans = [0]*idx

```

```

idx -= 1
for i in range(N-1, -1, -1):
    if idx == index[i]:
        ans[idx] = A[i]
        idx -= 1
print(*ans) # 수열


```

5.2 Berlekamp-Massey, 키타마사법

Time Complexity: Berlekamp-Massey: $O(NK + \log mod)$
키타마사법: $O(K^2 \log N)$

mod = 1_000_000_007

x에 3x(가장 짧은 접두식 길이) 이상의 항을 넣어줘야 함. 모듈러는 소수여야 함.

```

def berlekamp_massey(x):
    ls=[]; cur=[]
    lf=ld=0
    for i in range(len(x)):
        t=0
        for j in range(len(cur)): t=(t+x[i-j-1]*cur[j])%mod
        if(t-x[i])%mod==0: continue
        if len(cur)==0:
            cur.append(0)
            lf=i
            ld=(t-x[i])%mod
            continue
        k=-(x[i]-t)*pow(ld,mod-2,mod)%mod
        c=[k]
        for j in ls: c.append(-j*k%mod)
        if len(c)<len(cur):c+=[0]*(len(cur)-len(c))
        for j in range(len(cur)): c[j]=(c[j]+cur[j])%mod
        if i-lf+1>=len(cur): ls,lf,ld=cur,i,(t-x[i])%mod
        cur=c
    for i in cur: i=(i%mod+mod)%mod
    return cur

# n번째 항을 반환 (키타마사법)
# rec: 접두식, dp: 초기항
def get_nth(rec, dp, n):
    m=len(rec)
    s=[0]*m; t=[0]*m
    s[0]=1
    if m!=1: t[1]=1
    else: t[0]=rec[0]
    def mul(v, w):
        m=len(v)
        t=[0]*(2*m)
        for j in range(m):
            for k in range(m):
                t[j+k]+=v[j]*w[k]%mod
                if t[j+k]>=mod: t[j+k]-=mod
    for j in range(2*m-1,m-1,-1):
        for k in range(1,m+1):
            t[j-k]+=t[j]*rec[k-1]%mod
            if t[j-k]>=mod: t[j-k]-=mod
    return t[:m]
    while n:

```

```

if n&1:s=mul(s,t)
t=mul(t,t)
n>>=1
ret=0
for i in range(m): ret+=s[i]*dp[i]%mod
return ret%mod

```

수열 x의 n번째 항을 반환

```

def guess_nth_term(x, n):
    if n<len(x): return x[n]
    v=berlekamp_massey(x)
    if len(v)==0: return 0
    return get_nth(v, x, n)

```

5.3 SOS(Sum Over Subsets) DP

Time Complexity: $O(2^N \times N)$

```

// a[p][k]: p의 부분 마스크 중 k번째 비트의 원쪽 비트들이 p와
일치하는 것들의 집합의 원소들의 합
void sos(int n){
    for(int i = 0; i < 1 << n; i++){
        for(int j = 1; j <= n; j++){
            a[i][j] = a[i][j - 1];
            if(i & 1 << j - 1)a[i][j] += a[i - (1 << j - 1)][j - 1];
        }
    }
} // a[p][n]에 p의 부분집합의 합이 저장된다.

```

5.4 $O(N \times \max W)$ Subset Sum (Fast Knapsack)

```

// O(N*\maxW), maximize sumW <= t
int Knapsack(vector<int> w, int t){
    int a = 0, b = 0, x;
    while(b < w.size() && a + w[b] <= t) a += w[b++];
    if(b == w.size()) return a;
    int m = *max_element(w.begin(), w.end());
    vector<int> u, v(2*m, -1); v[a+m-t] = b;
    for(int i=b; (u==v,i<w.size()); i++){
        for(x=0; x<m; x++) v[x+w[i]] = max(v[x+w[i]], u[x]);
        for(x=2*m; --x>m; ) for(int j=max(0,u[x]); j<v[x]; j++)
            v[x-w[j]] = max(v[x-w[j]], j);
    } for(a=t; v[a+m-t]<0; a--);; return a;
}

```

5.5 DP Optimization, Knuth Optimization

```

// Quadrangle Inequality : C(a, c)+C(b, d) ≤ C(a, d)+C(b, c)
// Monotonicity : C(b, c) ≤ C(a, d)
// CHT, DnC Opt(Quadrangle), Knuth(Quadrangle and Monotonicity)
// Knuth: K[i][j-1] <= K[i][j] <= K[i+1][j]
// 1. Calculate D[i][i], K[i][i]
// 2. Calculate D[i][j], K[i][j] (i < j)
// Another: D[i][j] = min(D[i-1][k] + C[k+1][j]), C quadrangle
// i=1..k j=n..1 k=K[i-1,j]..K[i,j+1] update,
vnoi/icpc22_mn_c

```

// 2015 ICPC 예선 F번 파일 합치기 0(N^2)

```

int T, N;int A[503], S[503];
int D[503][503], K[503][503];
int main(){
    for (scanf("%d", &T);T--;){

```

```

        scanf("%d", &N);
        for (int i=1;i<=N;i++) scanf("%d", A+i), S[i] = S[i-1]
+ A[i];
        for (int i=1;i<=N;i++) D[i-1][i] = 0, K[i-1][i] = i;
        for (int d=2;d<=N;d++){
            // d = j-i
            for (int i=0;i+d<=N;i++){
                int j = i+d;
                D[i][j] = 2e9;
                for (int k=K[i][j-1];k<=K[i+1][j];k++){
                    int v = D[i][k] + D[k][j] + S[j] - S[i];
                    if (D[i][j] > v)
                        D[i][j] = v, K[i][j] = k;
                }
            }
            printf("%d\n", D[0][N]);
        }
    }
}

```

5.6 Monotone Queue Optimization

Time Complexity: $O(N \log N)$

```

template<class T, bool GET_MAX = false> // D[i] = func_{0 <= j
< i} D[j] + cost(j, i)
pair<vector<T>, vector<int>> monotone_queue_dp(int n, const
vector<T> &init, auto cost){
    assert((int)init.size() == n + 1); // cost function -> auto,
do not use std::function
    vector<T> dp = init; vector<int> prv(n+1);
    auto compare = [] (T a, T b){ return GET_MAX ? a < b : a > b;
};
    auto cross = [&] (int i, int j){
        int l = j, r = n + 1;
        while(l < r){
            int m = (l + r + 1) / 2;
            if(compare(dp[i] + cost(i, m), dp[j] + cost(j, m))) r = m
- 1; else l = m;
        } return l; };
        deque<int> q[0];
        for(int i=1; i<=n; i++){
            while(q.size() > 1 && compare(dp[q[0]] + cost(q[0], i),
dp[q[1]] + cost(q[1], i))) q.pop_front();
            dp[i] = dp[q[0]] + cost(q[0], i); prv[i] = q[0];
            while(q.size() > 1 && cross(q[q.size()-2], q.back()) >=
cross(q.back(), i)) q.pop_back();
            q.push_back(i);
        } /*for end*/ return {dp, prv}; }

```

5.7 Slope Trick

```

//NOTE: f(x)=min{f(x+i),i<a}+|x-k|+m -> pf(k)sf(k)ab(-a,m)
//NOTE: sf_inc에 담고하는게 들어있어서, 반드시 한 연산에 대해
pf_dec->sf_inc순서로 호출
struct LeftHull{
    void pf_dec(int x){ pq.emp1(x-bias); } //x이하의 기울기들 -1
    int sf_inc(int x){ //x이상의 기울기들 +1, pop된 원소 반환(Right
Hull관리에 사용됨)
        if(pq.empty() or argmin()<=x) return x; ans +=
argmin()-x; //이 경우 최솟값이 증가함
        pq.emp1(x-bias); /*x 이하 -1*/ int r=argmin(); pq.pop(); /*전체
+1*/

```

```

        return r;
    }
    void add_bias(int x,int y){ bias+=x; ans+=y; } int minval(){
return ans; } //x축 평행이동, 최소값
    int argmin(){return pq.empty()?-inf<int>():pq.top()+bias;}///
최소값 x좌표
    void operator+=(LeftHull& a){ ans+=a.ans; while(sz(a.pq))
pf_dec(a.argmin()), a.pq.pop(); }
    int size()const{return sz(pq);} PQMax<int> pq; int ans=0,
bias=0;
}
//NOTE: f(x)=min{f(x+i),a<i<b}+|x-k|+m -> pf(k)sf(k)ab(-a,b,m)
struct SlopeTrick{
    void pf_dec(int x){l.pf_dec(-r.sf_inc(-x));}
    void sf_inc(int x){r.pf_dec(-l.sf_inc(x));}
    void add_bias(int lx,int rx,int
y){l.add_bias(lx,0),r.add_bias(-rx,0),ans+=y; }
    int minval(){return ans+l.minval()+r.minval();}
    pint argmin(){return {l.argmin(),-r.argmin();}}
    void operator+=(SlopeTrick& a){
        while(sz(a.l.pq)) pf_dec(a.l.argmin()),a.l.pq.pop();
        l.ans+=a.l.ans;
        while(sz(a.r.pq)) sf_inc(-a.r.argmin()),a.r.pq.pop();
        r.ans+=a.r.ans; ans+=a.ans;
    } LeftHull l,r; int ans=0;
    int size()const{return l.size()+r.size(); }
};

//LeftHull 역추적 방법: 스텝i의 argmin값을 am(i)라고 하자. 스텝n
부터 스텝1까지 ans[i]=min(ans[i+1],am(i))하면 된다. 아래는 증명..은
아니고 간략한 이유
//am(i)<=ans[i+1]일때: ans[i]=am(i)
//x[i]>ans[i+1]일때: ans[i]=ans[i+1] 왜냐하면 f(i,a)는 a<x[i]에서
감소함수이므로 가능한 최대로 오른쪽으로 붙은 ans[i+1]이 최적.
//스텝i에서 add_bias(k,0)한다면 간격제한k가 있는것이므로
ans[i]=min(ans[i+1]-k,x[i])으로 수정.
//LR Hull 역추적은 케이스나눠서 위 방법을 확장하면 될듯

```

5.8 Aliens Trick

Time Complexity: $O(N^2 \log |W|)$

```

// pair<T, vector<int>> f(T c): return opt_val, prv
// cost function must be multiplied by 2
template<class T, bool GET_MAX = false>
pair<T,vector<int>> AliensTrick(int n,int k,auto f,T lo,T hi){
    T l = lo, r = hi; while(l < r) {
        T m = (l + r + (GET_MAX?1:0)) >> 1;
        vector<int> prv = f(m*2+(GET_MAX?-1:+1)).second;
        int cnt = 0; for(int i=n; i; i=prv[i]) cnt++;
        if(cnt <= k) (GET_MAX?l:r) = m;
        else (GET_MAX?r:l) = m + (GET_MAX?-1:+1);
    }
    T opt_value = f(1*2).first / 2 - k*l;
    vector<int> prv1 = f(1*2+(GET_MAX?-1:-1)).second, p1[n];
    vector<int> prv2 = f(1*2-(GET_MAX?-1:-1)).second, p2[n];
    for(int i=n; i; i=prv1[i]) p1.push_back(prv1[i]);
    for(int i=n; i; i=prv2[i]) p2.push_back(prv2[i]);
    reverse(p1.begin(),p1.end());reverse(p2.begin(),p2.end());
    assert(p2.size() <= k+1 && k+1 <=p1.size());
}

```

```

if(p1.size() == k+1) return {opt_value, p1};
if(p2.size() == k+1) return {opt_value, p2};
for(int i=1, j=1; i<p1.size(); i++){
    while(j < p2.size() && p2[j] < p1[i-1]) j++;
    if(p1[i] <= p2[j] && i - j == k+1 - (int)p2.size()){
        vector<int> res;
        res.insert(res.end(), p1.begin(), p1.begin()+i);
        res.insert(res.end(), p2.begin()+j, p2.end());
        return {opt_value, res};
    }
} /* if */ } /* for */ assert(false);
}

```

6 그래프

6.1 최단 거리 - Floyd Warshall

Time Complexity: $O(V^3)$

```

int N, G[111][111];
void Init(){ // 시작 전에 초기화 함
    memset(G, 0x3f, sizeof G);
    for(int i=1; i<=N; i++) G[i][i] = 0;
}
// s에서 e로 가는 가중치 w 간선 추가
void AddEdge(int s, int e, int w){
    G[s][e] = min(G[s][e], w);
}
void Run(){
    for(int k=1; k<=N; k++)
        for(int i=1; i<=N; i++)
            for(int j=1; j<=N; j++)
                G[i][j] = min(G[i][j], G[i][k] + G[k][j]);
}

```

6.2 최단 거리 - Dijkstra

Time Complexity: $O(E \log E)$

```

ll D[505050], P[505050];
vector<pair<ll, ll>> G[505050]; // {정점, 가중치}
// 주의: 가중치  $\geq 0$ , 음수 있으면 bellman ford 사용

// s -> t 최단 경로 출력
void Dijkstra(int s, int t){
    memset(D, 0x3f, sizeof D);
    priority_queue<pair<ll, ll>, vector<pair<ll, ll>>, greater<>> Q;
    Q.emplace(D[s]=0, s);
    while(!Q.empty()){
        auto [c, v] = Q.top(); Q.pop();
        if(c == D[v]) for(auto [i, w] : G[v]) if(D[i] > c + w)
            Q.emplace(D[i]=c+w, i), P[i] = v;
    }
    vector<int> path;
    for(int i=t; i!=s; i=P[i]) path.push_back(i);
    path.push_back(s);
    reverse(path.begin(), path.end());
    for(auto i : path) cout << i << " ";
}

```

6.3 최단 거리 - Bellman Ford

Time Complexity: $O(VE)$

```

int N, M; ll D[555]; // 주의: 웬만하면 long long으로 잡는 게 좋음
vector<tuple<int, int, ll>> E; // {from, to, weight}
void AddEdge(int s, int e, int w){
    E.emplace_back(s, e, w);
}
// s에서 도달 가능한 음수 사이클 있으면 false 반환
bool Run(int s){
    memset(D, 0x3f, sizeof D);
    ll INF = D[0];
    D[s] = 0;
    for(int iter=1; iter<=N; iter++){
        bool changed = false;
        for(auto [u, v, w] : E){
            if(D[u] == INF) continue;
            if(D[v] > D[u] + w) D[v] = D[u] + w, changed = true;
        }
        if(iter == N && changed) return false;
    }
    return true;
}

```

6.4 유니온 파인드 + 최소 신장 트리(Kruskal)

Time Complexity: UF: 연산마다 $O(\log N)$, MST: $O(E \log E)$

```

int N, M, P[10101];
int Find(int v){ return v == P[v] ? v : P[v] = Find(P[v]); }
bool Merge(int u, int v){
    u = Find(u); v = Find(v);
    if(u == v) return false;
    P[u] = v; return true;
}
int main(){
    cin >> N >> M;
    vector<tuple<int, int, int>> E; // {weight, from, to}
    for(int i=1, u, v, w; i<=M; i++){
        cin >> u >> v >> w;
        E.emplace_back(w, u, v);
    }
    sort(E.begin(), E.end());
    for(int i=1; i<=N; i++) P[i] = i;
    long long res = 0;
    for(auto [w, u, v] : E) if(Merge(u, v)) res += w;
    cout << res;
}

def find(n):
    if uf[n] < 0: return n
    uf[n] = find(uf[n])
    return uf[n]

def union(a, b):
    a = find(a)
    b = find(b)
    if a == b: return
    if uf[a] > uf[b]: a, b = b, a # smaller to larger
    uf[a] += uf[b]
    uf[b] = a

uf = [-1]*n

```

6.5 Euler Tour

```

// Not Directed / Cycle
constexpr int SZ = 1010;
int N, G[SZ][SZ], Deg[SZ], Work[SZ];
void DFS(int v){
    for(int &i=Work[v]; i<=N; i++) while(G[v][i]) G[v][i]--;
    G[i][v]--, DFS(i);
    cout << v << " ";
}
// Directed / Path
void DFS(int v){
    for(int i=1; i<=pv; i++) while(G[v][i]) G[v][i]--;
    Path.push_back(v);
}
void Get(){
    for(int i=1; i<=pv; i++) if(In[i] < Out[i]){ DFS(i); return; }
    for(int i=1; i<=pv; i++) if(Out[i]){ DFS(i); return; }
} // WARNING: path.size() == M + 1 && not trail

```

6.6 SCC - Kosaraju

Time Complexity: $O(V + E)$

```

int N, M, C[10101]; // C[i] = i번 정점이 속한 SCC 번호
vector<int> G[10101], R[10101], V;
vector<vector<int>> S; // 각 SCC에 속한 정점 목록
void AddEdge(int s, int e){
    G[s].push_back(e);
    R[e].push_back(s);
}
void DFS1(int v){
    C[v] = -1;
    for(auto i : G[v]) if(!C[i]) DFS1(i);
    V.push_back(v);
}
void DFS2(int v, int c){
    C[v] = c; S.back().push_back(v);
    for(auto i : R[v]) if(C[i] == -1) DFS2(i, c);
}
int GetSCC(){ // SCC 개수 반환
    for(int i=1; i<=N; i++) if(!C[i]) DFS1(i);
    reverse(V.begin(), V.end());
    int cnt = 0;
    for(auto i : V) if(C[i] == -1) S.emplace_back(), DFS2(i, cnt++);
    return cnt;
} // 각 SCC는 위상 정렬 순서대로 번호 매겨져 있음

```

6.7 BCC - Tarjan

Time Complexity: $O(V + E)$

```

// 1-based, 다른 거 호출하기 전에 tarjan 먼저 호출해야 함
vector<int> G[MAX_V]; int In[MAX_V], Low[MAX_V], P[MAX_V];
void addEdge(int s, int e){ G[s].push_back(e);
G[e].push_back(s); }
void tarjan(int n){ // Pre-Process
    int pv = 0;
    function<void(int, int)> dfs = [&pv, &dfs](int v, int b){
        In[v] = Low[v] = ++pv; P[v] = b;
        for(auto i : G[v]) if(Low[i] == -1) tarjan(i);
        Low[v] = min(Low[v], Low[i]);
    };
    dfs(n, n);
}

```

```

for(auto i : G[v]){
    if(i == b) continue;
    if(!In[i]) dfs(i, v), Low[v] = min(Low[v], Low[i]); else
    Low[v] = min(Low[v], In[i]);
}
};

for(int i=1; i<=n; i++) if(!In[i]) dfs(i, -1);
}

vector<int> cutVertex(int n){
    vector<int> res; array<char,MAX_V> isCut; isCut.fill(0);
    function<void(int)> dfs = [&dfs,&isCut](int v){
        int ch = 0;
        for(auto i : G[v]){
            if(P[i] != v) continue; dfs(i); ch++;
            if(P[v] == -1 && ch > 1) isCut[v] = 1; else if(P[v] != -1 && Low[i] >= In[v]) isCut[v]=1;
        }
    };
    for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
    for(int i=1; i<=n; i++) if(isCut[i]) res.push_back(i);
    return move(res);
}

vector<PII> cutEdge(int n){
    vector<PII> res;
    function<void(int)> dfs = [&dfs,&res](int v){
        for(int t=0; t<G[v].size(); t++){
            int i = G[v][t]; if(t != 0 && G[v][t-1] == G[v][t]) continue;
            if(P[i] != v) continue; dfs(i);
            if((t+1 == G[v].size() || i != G[v][t+1]) && Low[i] > In[v]) res.emplace_back(min(v,i), max(v,i));
        }
    };
    for(int i=1; i<=n; i++) sort(G[i].begin(), G[i].end()); // multi edge -> sort
    for(int i=1; i<=n; i++) if(P[i] == -1) dfs(i);
    return move(res); // sort(all(res));
}

vector<int> BCC[MAX_V]; // BCC[v] = components which contains v
void vertexDisjointBCC(int n){ // allow multi edge, not allow self loop
    int cnt = 0; array<char,MAX_V> vis; vis.fill(0);
    function<void(int,int)> dfs = [&dfs,&vis,&cnt](int v, int c){
        vis[v] = 1; if(c > 0) BCC[v].push_back(c);
        for(auto i : G[v]){
            if(vis[i]) continue;
            if(In[v] <= Low[i]) BCC[v].push_back(++cnt), dfs(i, cnt);
            else dfs(i, c);
        }
    };
    for(int i=1; i<=n; i++) if(!vis[i]) dfs(i, 0);
    for(int i=1; i<=n; i++) if(BCC[i].empty())
    BCC[i].push_back(++cnt);
}

```

6.8 최대 유량 - Dinic

Time Complexity: $O(V^2E)$, 모든 간선의 용량이 1이면 $O(\min(V^{2/3}, E^{1/2})E)$

// Directed Graph이면 add_edge(s, e, c)
// Undirected Graph이면 add_edge(s, e, c, c)

```

template<typename flow_t, flow_t MAX_U=(1<<30)>
struct Dinic{ // 0-based
    struct edge_t{ int v, r; flow_t c, f; };
    int n;
    vector<vector<edge_t>> g;
    vector<int> lv, idx;
    Dinic(int n) : n(n) { clear(); }
    void clear(){
        g = vector<vector<edge_t>>(n);
        lv = vector<int>(n, 0);
        idx = vector<int>(n, 0);
    }
    void add_edge(int s, int e, flow_t c1, flow_t c2=flow_t(0)){
        g[s].push_back({e, (int)g[e].size(), c1, 0});
        g[e].push_back({s, (int)g[s].size()-1, c2, 0});
    }
    bool bfs(int s, int t, flow_t limit=1){
        fill(lv.begin(), lv.end(), 0);
        queue<int> que; que.push(s); lv[s] = 1;
        while(!que.empty()){
            int v = que.front(); que.pop();
            for(const auto &e : g[v]) if(!lv[e.v] && e.c - e.f >= limit) que.push(e.v), lv[e.v] = lv[v] + 1;
        }
        return lv[t] != 0;
    }
    flow_t dfs(int v, int t, flow_t fl=MAX_U){
        if(v == t || fl == flow_t(0)) return fl;
        for(int &i=idx[v]; i<g[v].size(); i++){
            auto &e = g[v][i];
            if(lv[e.v] != lv[v] + 1 || e.c - e.f == flow_t(0)) continue;
            flow_t now = dfs(e.v, t, min(fl, e.c - e.f));
            if(now == flow_t(0)) continue;
            e.f += now; g[e.v][e.r].f -= now;
            return now;
        }
        return 0;
    }
    flow_t maximum_flow(int s, int t){
        flow_t flow = 0, augment = 0;
        while(bfs(s, t)){
            fill(idx.begin(), idx.end(), 0);
            while(augment=dfs(s, t)) != flow_t(0) flow += augment;
        }
        return flow;
    }
    // {최소 컷 비용, s와 같은 집합, t와 같은 집합, 절단 간선}
    tuple<flow_t, vector<int>, vector<int>, vector<pair<int,int>>> minimum_cut(int s, int t){
        flow_t flow = maximum_flow(s, t);
        vector<int> a, b;
        vector<pair<int,int>> edges;
        bfs(s, t, 1);
        for(int i=0; i<n; i++) (lv[i] ? a : b).push_back(i);
        for(auto i : a) for(auto e : g[i]) if(e.c != flow_t(0) && !lv[e.v]) edges.emplace_back(i, e.v);
        return {flow, a, b, edges};
    }
};

```

6.9 MCMF

```

// 유량을 k 만큼 흘리는 경우: run(src, snk, k)
// 유량을 최대한 많이 흘리는 경우: run(src, snk)
template<typename flow_t=int, typename cost_t=long long, flow_t MAX_U=(1<<30), cost_t MAX_C=(1LL<<60)>
struct MinCostFlow{ // 0-based
    struct edge_t{ int v, r; flow_t c; cost_t d; };
    int n;
    vector<vector<edge_t>> g;
    vector<int> prv, idx, chk;
    vector<cost_t> dst;
    MinCostFlow(int n) : n(n) { clear(); }
    void clear(){
        g = vector<vector<edge_t>>(n);
        prv = idx = chk = vector<int>(n);
        dst = vector<cost_t>(n);
    }
    void add_edge(int s, int e, flow_t c, cost_t d){
        g[s].push_back({e, (int)g[e].size(), c, d});
        g[e].push_back({s, (int)g[s].size()-1, 0, -d});
    }
    bool find_path(int s, int t){
        fill(chk.begin(), chk.end(), 0);
        fill(dst.begin(), dst.end(), MAX_C);
        queue<int> que; que.push(s); dst[s] = 0; chk[s] = 1;
        while(!que.empty()){
            int v = que.front(); que.pop(); chk[v] = 0;
            for(int i=0; i<g[v].size(); i++){
                const auto &e = g[v][i];
                if(e.c > 0 && dst[e.v] > dst[v] + e.d){
                    dst[e.v] = dst[v] + e.d; prv[e.v] = v;
                    idx[e.v] = i;
                    if(!chk[e.v]) que.push(e.v), chk[e.v] = 1;
                }
            }
        }
        return dst[t] < MAX_C;
    }
    pair<flow_t, cost_t> augment(int s, int t, flow_t k=-1){
        if(!find_path(s, t)) return {0, 0};
        flow_t fl = MAX_U;
        for(int i=t; i!=s; i=prv[i]) fl = min(fl, g[prv[i]][idx[i]].c);
        if(k != -1) fl = min(fl, k);
        for(int i=t; i!=s; i=prv[i]){
            g[prv[i]][idx[i]].c -= fl;
            g[i][g[prv[i]][idx[i]].r].c += fl;
        }
        return {fl, fl * dst[t]};
    }
    pair<flow_t, cost_t> run(int s, int t, flow_t k=-1){
        flow_t flow = 0; cost_t cost = 0;
        while(true){
            auto [fl,cst] = augment(s, t, k);
            if(fl == 0) break;
            flow += fl; cost += cst;
            if(k != -1) k -= fl;
        }
    }
}
```

```

    }
    return {flow, cost};
}
};


```

6.10 이분 매칭

Time Complexity: $O(VE)$

```

def dfs(a):
    visited[a]=True
    for b in adj[a]:
        if B[b]==-1: B[b]=a; return True
    for b in adj[a]:
        if not(visited[B[b]]) and dfs(B[b]): B[b]=a; return True
    return False

match=0
B=[-1]*M
for i in range(N):
    visited=[0]*N
    if dfs(i): match+=1
print(match)

```

6.11 이분 매칭 - Hopcroft Karp

Time Complexity: $O(E\sqrt{V})$

```

// n: 왼쪽 정점 개수, m: 오른쪽 정점 개수, 0-based
struct HopcroftKarp{
    int n, m;
    vector<vector<int>> g;
    vector<int> dst, le, ri;
    vector<char> visit, track;
    HopcroftKarp(int n, int m) : n(n), m(m) { clear(); }
    void clear(){
        g = vector<vector<int>>(n); dst = vector<int>(n, 0);
        le = vector<int>(n, -1); ri = vector<int>(m, -1);
        visit = vector<char>(n, 0); track = vector<char>(n+m, 0);
    }
    void add_edge(int s, int e){ g[s].push_back(e); }
    bool bfs(){
        bool res = false;
        queue<int> que;
        fill(dst.begin(), dst.end(), 0);
        for(int i=0; i<n; i++) if(le[i] == -1) que.push(i), dst[i] = 1;
        while(!que.empty()){
            int v = que.front(); que.pop();
            for(auto i : g[v]){
                if(ri[i] == -1) res = true;
                else if(!dst[ri[i]]) dst[ri[i]] = dst[v] + 1,
                que.push(ri[i]);
            }
        }
        return res;
    }
    bool dfs(int v){
        if(visit[v]) return false;
        visit[v] = 1;
        for(auto i : g[v]){
            if(ri[i] == -1 || !visit[ri[i]] && dst[ri[i]] == dst[v] + 1 && dfs(ri[i])){

```

```

                le[v] = i; ri[i] = v; return true;
            }
        }
        return false;
    }
    int maximum_matching(){
        int res = 0;
        fill(le.begin(), le.end(), -1);
        fill(ri.begin(), ri.end(), -1);
        while(bfs()){
            fill(visit.begin(), visit.end(), 0);
            for(int i=0; i<n; i++) if(le[i] == -1) res += dfs(i);
        }
        return res;
    }
    vector<pair<int,int>> maximum_matching_edges(){
        int matching = maximum_matching();
        vector<pair<int,int>> edges; edges.reserve(matching);
        for(int i=0; i<n; i++) if(le[i] != -1)
            edges.emplace_back(i, le[i]);
        return edges;
    }
    void dfs_track(int v){
        if(track[v]) return; track[v] = 1;
        for(auto i : g[v]) track[n+i] = 1, dfs_track(ri[i]);
    }
    tuple<vector<int>, vector<int>, int> minimum_vertex_cover(){
        int matching = maximum_matching();
        fill(track.begin(), track.end(), 0);
        for(int i=0; i<n; i++) if(le[i] == -1) dfs_track(i);
        vector<int> lv, rv;
        for(int i=0; i<n; i++) if(!track[i]) lv.push_back(i);
        for(int i=0; i<m; i++) if(track[n+i]) rv.push_back(i);
        assert(lv.size() + rv.size() == matching);
        return {lv, rv, lv.size() + rv.size()};
    }
}
```

6.12 플로우 관련 정리

- 최대 유량 = 최소 컷(minimum cut)
- 버텍스 커버(vertex cover)의 여집합은 독립 집합(independent set)이다. 따라서 최소 버텍스 커버(minimum vertex cover)의 여집합은 최대 독립 집합(maximum independent set)이다.
- DAG에서 최소 경로 커버(minimum path cover)의 크기는 $N - (\text{maximum matching})$ 이다.
- 쾨너의 정리: 모든 bipartite graph에서 maximum matching의 크기와 minimum vertex cover의 크기는 동일하다.
- 이분 그래프의 maximum independent set의 크기 = $V(\text{전체 정점}) - \text{Max Matching}$
- 이분 그래프에서 최소 버텍스 커버는 최소 버텍스 커버와 같다.
- 딜워스의 정리: Poset(부분 순서 집합)에서 Maximum Anti Chain의 크기와 Minimum Path Cover의 크기는 같다.

6.13 $O(V^3)$ Hungarian Method

```

// C[j][w] = cost(j-th job, w-th worker), j <= w, 0(J^2W)
// ret[i] = minimum cost to assign 0..i jobs to distinct
workers
template<typename T>bool ckmin(T &a, const T &b){return b < a ?
a=b, 1 : 0;}
template<typename T>vector<T>Hungarian(const
vector<vector<T>>&C){

```

```

const int J = C.size(), W = C[0].size(); assert(J <= W);
vector<int> job(W+1, -1); //job[i] - i(worker) matched
vector<T> ys(J), yt(W+1, answers); //W-th worker is dummy
const T inf = numeric_limits<T>::max();
for(int j_cur=0; j_cur<J; j_cur++){
    int w_cur = W; job[w_cur] = j_cur;
    vector<T> min_to(W+1, inf); vector<int> prv(W+1, -1), in(W+1);
    while(job[w_cur] != -1){
        int w_cur=1; T delta=inf; int j = job[w_cur], w_next;
        for(int w=0; w<W; w++){ if(in[w] != 0) continue;
            if(ckmin(min_to[w], C[j][w]-ys[j]-yt[w])) prv[w]=w_cur;
            if(ckmin(delta, min_to[w])) w_next = w;
        }
        for(int w=0; w<=W; w++){
            if(in[w] == 0) min_to[w] -= delta;
            else ys[job[w]] += delta, yt[w] -= delta;
        } /*end for w*/ w_cur = w_next; } /* end while */
        for(int w; w<cur!=1; w<cur) job[w_cur]=job[w=prv[w_cur]];
        answers.push_back(-yt[W]);
    } return answers; }

```

6.14 $O(V^3)$ General Matching

```

int N, M, R, Match[555], Par[555], Chk[555], Prv[555],
Vis[555];
vector<int> G[555]; // n 500 20ms
int Find(int x){return x == Par[x] ? x : Par[x] =
Find(Par[x]);}
int LCA(int u, int v){ static int cnt = 0;
    for(cnt++; Vis[u]!=cnt; swap(u, v)) if(u) Vis[u] = cnt, u =
Find(Prv[Match[u]]); return u; }
void Blossom(int u, int v, int rt, queue<int> &q){
    for(; Find(u)!=rt; u=Prv[v]){
        Prv[u] = v; Par[u] = Par[v=Match[u]] = rt;
        if(Chk[v] & 1) q.push(v), Chk[v] = 2;
    } }
bool Augment(int u){ // iota Par 0, fill Chk 0
    queue<int> Q; Q.push(u); Chk[u] = 2;
    while(!Q.empty()){ u = Q.front(); Q.pop();
        for(auto v : G[u]){
            if(Chk[v] == 0){
                Prv[v]=u; Chk[v]=1; Q.push(Match[v]); Chk[Match[v]]=2;
                if(!Match[v]){ for(; u; v=u) u = Match[Prv[v]],
Match[Match[v]]=Prv[v] = v;; return true; }
            } else if(Chk[v] == 2){ int l = LCA(u, v); Blossom(u, v, l,
Q), Blossom(v, u, l, Q); }
        } /* for v */ } /* while */
    return 0; }
void Run(){ for(int i=1; i<=N; i++) if(!Match[i]) R +=
Augment(i); }

```

6.15 $O(V^3)$ Weighted General Matching

```

namespace weighted_blossom_tree{ // n 400 w 1e8 700ms, n 500 w
1e6 300ms
#define d(x) (lab[x.u]+lab[x.v]-e[x.u][x.v].w*2)

```

```

const int N=403*2; using ll = long long; using T = int; // sum of weight, single weight
const T inf=numeric_limits<T>::max()>>1;
struct Q{ int u, v; T w; } e[N][N]; vector<int> p[N];
int n, m=0, id, h, t, lk[N], sl[N], st[N], f[N], b[N][N],
s[N], ed[N], q[N]; T lab[N];
void upd(int u, int v){ if (!sl[v] || d(e[u][v]) < d(e[s1[v]][v])) sl[v] = u; }
void ss(int v){
    sl[v]=0; for(int u=1; u<=n; u++) if(e[u][v].w > 0 && st[u] != v && !s[st[u]]) upd(u, v);
}
void ins(int u){ if(u <= n) q[++t] = u; else for(int v : p[u]) ins(v); }
void mdf(int u, int w){ st[u]=w; if(u > n) for(int v : p[u]) mdf(v, w); }
int gr(int u,int v){
    if ((v=find(p[u].begin(), p[u].end(), v) - p[u].begin()) & 1){
        reverse(p[u].begin()+1, p[u].end()); return
(int)p[u].size() - v;
    }
    return v;
}
void stm(int u, int v){
    lk[u] = e[u][v].v;
    if(u <= n) return; Q w = e[u][v];
    int x = b[u][w.u], y = gr(u,x);
    for(int i=0; i<y; i++) stm(p[u][i], p[u][i^1]);
    stm(x,v);rotate(p[u].begin(), p[u].begin() + y, p[u].end());
}
void aug(int u, int v){
    int w = st[lk[u]]; stm(u, v); if (!w) return;
    stm(w, st[f[w]]); aug(st[f[w]], w); }
int lca(int u, int v){
    for(++id; u|v; swap(u, v)){
        if(!u) continue; if(ed[u] == id) return u;
        ed[u] = id; if(u == st[lk[u]]) u = st[f[u]]; // not ==
    }
    return 0;
}
void add(int u, int a, int v){
    int x = n+1; while(x <= m && st[x]) x++;
    if(x > m) m++;
    lab[x] = s[x] = st[x] = 0; lk[x] = lk[a];
    p[x].clear(); p[x].push_back(a);
    for(int i=u, j; i!=a; i=st[f[j]]) p[x].push_back(i),
p[x].push_back(j=st[lk[i]]), ins(j);
    reverse(p[x].begin() + 1, p[x].end());
    for(int i=v, j; i!=a; i=st[f[j]]) p[x].push_back(i),
p[x].push_back(j=st[lk[i]]), ins(j);
    mdf(x,x); for(int i=1; i<=m; i++) e[x][i].w=e[i][x].w=0;
    memset(b[x]+1, 0, n*sizeof b[0][0]);
    for (int u : p[x]){
        for(v=1; v<=m; v++) if(!e[x][v].w || d(e[u][v]) < d(e[x][v])) e[x][v] = e[u][v], e[v][x] = e[v][u];
        for(v=1; v<=n; v++) if(b[u][v]) b[x][v] = u;
    }
    ss(x); }
void ex(int u){ // s[u] == 1
    for(int x : p[u]) mdf(x, x);
    int a = b[u][e[u][f[u]].u], r = gr(u, a);
}

```

```

for(int i=0; i<r; i+=2){
    int x = p[u][i], y = p[u][i+1];
    f[x] = e[y][x].u; s[x] = 1; s[y] = 0; sl[x] = 0; ss(y);;
    ins(y); }
s[a] = 1; f[a] = f[u];
for(int i=r+1; i<p[u].size(); i++) s[p[u][i]]=-1, ss(p[u][i]);
st[u] = 0; }
bool on(const Q &e){
    int u=st[e.u], v=st[e.v], a;
    if(s[v] == -1) f[v] = e.u, s[v] = 1, a = st[lk[v]], sl[v] =
sl[a] = s[a] = 0, ins(a);
    else if(!s[v]){
        a = lca(u, v); if(!a) return aug(u,v), aug(v,u), true;
    else add(u,a,v);
    }
    return false; }
bool bfs(){
    memset(s+1, -1, m*sizeof s[0]); memset(sl+1, 0, m*sizeof
sl[0]);
    h = 1; t = 0; for(int i=1; i<=m; i++) if(st[i] == i &&
!lk[i]) f[i] = s[i] = 0, ins(i);
    if(h > t) return 0;
    while (true){
        while (h <= t){
            int u = q[h++];
            if (s[st[u]] != 1) for (int v=1; v<=n; v++) if
(e[u][v].w > 0 && st[u] != st[v])
                if(d(e[u][v])) upd(u, st[v]); else if(on(e[u][v]))
return true;
        }
        T x = inf;
        for(int i=n+1; i<=m; i++) if(st[i] == i && s[i] == 1) x =
min(x, lab[i]>>1);
        for(int i=1; i<=m; i++) if(st[i] == i && sl[i] && s[i] !=
1) x = min(x, d(e[sl[i]][i])>>s[i]+1);
        for(int i=1; i<=n; i++) if(~s[st[i]]) if((lab[i] +=
(s[st[i]]*2-1)*x) <= 0) return false;
        for(int i=n+1; i<=m; i++) if(st[i] == i && ~s[st[i]])
lab[i] += (2-s[st[i]]*4)*x;
        h = 1; t = 0;
        for(int i=1; i<=m; i++) if(st[i] == i && sl[i] &&
st[sl[i]] != i && !d(e[sl[i]][i]) && on(e[sl[i]][i])) return
true;
        for(int i=n+1; i<=m; i++) if(st[i] == i && s[i] == 1 &&
!lab[i]) ex(i);
    }
    return 0; }
template<typename TT> pair<int,ll> run(int N, const
vector<tuple<int,int,TT>> &edges){ // 1-based
    memset(ed+1, 0, m*sizeof ed[0]); memset(lk+1, 0, m*sizeof
lk[0]);
    n = m = N; id = 0; iota(st+1, st+n+1, 1); T wm = 0; ll r =
0;
    for(int i=1; i<=n; i++) for(int j=1; j<=n; j++) e[i][j] =
{i,j,0};
    for(auto [u,v,w] : edges) wm = max(wm,
e[v][u].w=e[u][v].w=max(e[u][v].w,(T)w));
    for(int i=1; i<=n; i++) p[i].clear();
    for(int i=1; i<=n; i++) for (int j=1; j<=n; j++) b[i][j] =
i*(i==j); }

```

```

fill_n(lab+1, n, wm); int match = 0; while(bfs()) match++;
for(int i=1; i<=n; i++) r += e[i][lk[i]].w;
return {match, r/2};
}
#endif
} using weighted_blossom_tree::run, weighted_blossom_tree::lk;

```

6.16 안정 결혼 문제

Usage: 안정적인 매칭은 다음 조건을 만족한다.
각 집합의 원소는 상대 집합의 원소들에 대한 서로 다른 선호도를 가진다.
첫 번째 집합과 두 번째 집합의 원소가 일대일로 매칭되어 있다.
첫 번째 집합에 속한 원소 A, B가 두 번째 집합에 속한 원소 C, D와 각각 매칭되어 있을 때 A가 C보다 D를 선호하면서 D가 B보다 A를 선호하는 경우는 존재하지 않는다.

Time Complexity: $O(V^2)$

```

# a[i][j]: 첫 번째 그룹의 i번 원소가 두 번째 그룹의 a[i][j]번
원소를 j번째로 선호함
# b[i][j]: 두 번째 그룹의 i번 원소가 첫 번째 그룹의 j번 원소를
b[i][j]번째로 선호함
cnt = [-1]*N
A = [-1]*N
B = [-1]*N
for _ in range(N):
    for i in range(N):
        if A[i] >= 0: continue # 첫 번째 그룹의 i번 원소가 매칭된 경우
        while 1:
            cnt[i] += 1
            w = a[i][cnt[i]] # i번 원소가 다음으로 매칭을 시도할 두 번째
그룹의 원소
            if B[w] < 0: # w번 원소가 아직 매칭되지 않은 경우
                A[i] = w; B[w] = i; break
            if b[w][i] < b[w][B[w]]: # w번 원소가 현재 매칭된 원소보다 i
번 원소를 더 선호하는 경우
                A[i] = w; A[B[w]] = -1; B[w] = i; break
for i in A: print(i+1)

```

6.17 최소 공통 조상(LCA)

Time Complexity: 전처리 $O(N \log N)$, 쿼리 $O(\log N)$

```

int N, Q, D[101010], P[22][101010];
vector<int> G[101010];
void Connect(int u, int v){
    G[u].push_back(v); G[v].push_back(u);
}
void DFS(int v, int b=-1){
    for(auto i : G[v]) if(i != b) D[i] = D[v] + 1, P[0][i] = v,
DFS(i, v);
}
int LCA(int u, int v){
    if(D[u] < D[v]) swap(u, v);
    int diff = D[u] - D[v];
    for(int i=0; diff; i++, diff>>=1) if(diff & 1) u = P[i][u];
    if(u == v) return u;
    for(int i=21; i>=0; i--) if(P[i][u] != P[i][v]) u =
P[i][u], v = P[i][v];
    return P[0][u];
}

```

```

////
// 1. Connect로 간선 추가
// 2. DFS(1) 호출
// 3. 아래 코드 실행
for(int i=1; i<22; i++) for(int j=1; j<=N; j++) P[i][j] =
P[i-1][P[i-1][j]];
// 4. LCA(u, v)로 최소 공통 조상 구할 수 있음

6.18 Heavy Light Decomposition
Time Complexity: 전처리  $O(N)$ , 쿼리  $O(T(N) \log N)$ 

int N, Q, A[SZ], Top[SZ], Par[SZ], Dep[SZ], Sz[SZ], In[SZ];
vector<int> Inp[SZ], G[SZ];

void Connect(int u, int v){
    Inp[u].push_back(v); Inp[v].push_back(u);
}

void DFS0(int v, int b=-1){
    for(auto i : Inp[v]) if(i != b)
        Dep[i] = Dep[v] + 1, Par[i] = v, G[v].push_back(i),
        DFS0(i, v);
}

void DFS1(int v){
    Sz[v] = 1;
    for(auto &i : G[v]){
        DFS1(i); Sz[v] += Sz[i];
        if(Sz[i] > Sz[G[v][0]]) swap(i, G[v][0]);
    }
}

void DFS2(int v){
    static int pv = 0; In[v] = ++pv;
    for(auto i : G[v]) Top[i] = i == G[v][0] ? Top[v] : i,
    DFS2(i);
}

void VertexUpdate(int x, int v){
    Update(In[x], v);
}

long long PathQuery(int u, int v){
    long long res = 0;
    for(; Top[u] != Top[v]; u=Par[Top[u]]){
        if(Dep[Top[u]] < Dep[Top[v]]) swap(u, v);
        res += Query(In[Top[u]], In[u]);
    }

    if(In[u] > In[v]) swap(u, v);
    res += SegQuery(In[u], In[v]); // 경점 쿼리는 In[u], 간선
    쿼리는 In[u]+1
    return res;
}
/////
// 1. Connect로 간선 추가
// 2. DFS0(1); DFS1(1); DFS2(Top[1]=1); 호출
// 3. VertexUpdate, PathQuery로 연산 수행
// 3-1. Update, Query는 배열의 구간 쿼리를 지원하는 자료구조(ex.
    세그먼트 트리) 사용

```

6.19 센트로이드 트리

```

struct centroid {
    int N;
    vi tree_size, vis, par;
    vvi _edges, edges;
    centroid(int N) : N(N), tree_size(N), vis(N), _edges(N),
    edges(N), par(N) {}
    void add_edge(int u, int v) { _edges[u].pb(v); }
    int get_size(int cur, int p) {
        tree_size[cur] = 1;
        for (int to: _edges[cur]) if (to != p &&
        !vis[to])tree_size[cur] += get_size(to, cur);
        return tree_size[cur];
    }
    int get_centroid(int cur, int p, int cap) {
        for (int to: _edges[cur]) if (to != p && !vis[to] &&
        tree_size[to] * 2 > cap) return get_centroid(to, cur, cap);
        return cur;
    }
    int build_tree(int cur, int p = -1) {
        cur = get_centroid(cur, -1, get_size(cur, -1));
        par[cur] = p, vis[cur] = 1;
        for (int to: _edges[cur]) if (!vis[to]) to =
        build_tree(to, cur), edges[cur].pb(to), edges[to].pb(cur);
        return cur;
    }
};


```

7 수학**7.1 바닥함수, 천장함수**

$$\begin{aligned} \left\lfloor \frac{n}{m} \right\rfloor &= \left\lfloor \frac{n+m-1}{m} \right\rfloor = \left\lfloor \frac{n-1}{m} \right\rfloor + 1, n \text{은 정수}, m \text{은 양의 정수} \\ \left\lfloor -x \right\rfloor &= -\lceil x \rceil; \left\lceil -x \right\rceil = -\lfloor x \rfloor \\ \left\lfloor \frac{x+m}{n} \right\rfloor &= \left\lfloor \frac{\lfloor x \rfloor + m}{n} \right\rfloor; \left\lceil \frac{x+m}{n} \right\rceil = \left\lceil \frac{\lceil x \rceil + m}{n} \right\rceil \end{aligned}$$

7.2 빠른 거듭제곱

Usage: $a^b \pmod c$ 를 구하는 함수
Time Complexity: $O(\log b)$

```

def power(a, b, mod):
    r = 1
    while b:
        if b&1: r = r*a%mod
        a = a*a%mod
        b >>= 1
    return r

```

7.3 Power Tower

```

bool PowOverflow(ll a, ll b, ll c){
    __int128_t res = 1;
    bool flag = false;
    for(; b; b >= 1, a = a * a){
        if(a >= c) flag = true, a %= c;
        if(b & 1){
            res *= a; if(flag || res >= c) return true;
        }
    }
    return false;
}
ll Recursion(int idx, ll mod, const vector<ll> &vec){

```

```

if(mod == 1) return 1;
if(idx + 1 == vec.size()) return vec[idx];
ll nxt = Recursion(idx+1, phi[mod], vec);
if(PowOverflow(vec[idx], nxt, mod)) return Pow(vec[idx], nxt, mod) +
mod; else return Pow(vec[idx], nxt, mod);
}

ll PowerTower(const vector<ll> &vec, ll mod){ //
vec[0]^=(vec[1]^=(vec[2]^(...)))
if(vec.size() == 1) return vec[0] % mod;
else return Pow(vec[0], Recursion(1, phi[mod], vec), mod);
}

```

7.4 Harmonic Lemma

Usage: $\sum_{i=1}^n \lfloor \frac{n}{i} \rfloor$
Time Complexity: $O(\sqrt{n})$

```

ans = 0; i = 1
while i <= n:
    j = n//(n/i)
    ans += n//i*(j-i+1)
    i = j+1
print(ans)

```

7.5 FloorSum

```

// sum of floor((A*i+B)/M) over 0 <= i < N in O(log(N+M+A+B))
// Also, sum of i * floor((A*i+B)/M) and floor((A*i+B)/M)^2
template<class T, class U> // T must be able to hold arg^2
array<U, 3> weighted_floor_sum(T n, T m, T a, T b){
    array<U, 3> res{}; auto[qa,ra]=div(a,m);
    auto[qb,rb]=div(b,m);
    if(T n2 = (ra * n + rb) / m){
        auto prv=weighted_floor_sum<T,U>(n2, ra, m, m-rb-1);
        res[0] += U(n-1)*n2 - prv[0];
        res[1] += (U(n-1)*n*n2 - prv[0] - prv[2]) / 2;
        res[2] += U(n-1)*(n2-1)*n2 - 2*prv[1] + res[0];
    }
    res[2] += U(n-1)*n*(2*n-1)/6 * qa*qa + U(n)*qb*qb;
    res[2] += U(n-1)*n * qa*qb + 2*res[0]*qb + 2*res[1]*qa;
    res[0] += U(n-1)*n/2 * qa + U(n)*qb;
    res[1] += U(n-1)*n*(2*n-1)/6 * qa + U(n-1)*n/2 * qb;
    return res;
}
ll modsum(ull to, ll c, ll k, ll m){
    c = (c % m + m) % m; k = (k % m + m) % m;
    return to*c + k*sumsq(to) - m*divsum(to, c, k, m);
} // sum (ki+c)%m 0 <= i < to, O(log m) large constant

```

7.6 Linear Sieve

```

// sp : 최소 소인수, 소수라면 0
// tau : 약수 개수, sigma : 약수 합
// phi : n 이하 자연수 중 n과 서로소인 개수
// phi(n) = n*(p1-1)/p1*...*(pk-1)/pk
// mu : non square free이면 0, 그렇지 않다면 (-1)^(소인수 종류)
// e[i] : 소인수분해에서 i의 지수
vector<int> prime;
int sp[sz], e[sz], phi[sz], mu[sz], tau[sz], sigma[sz];
phi[1] = mu[1] = tau[1] = sigma[1] = 1;
for(int i=2; i<=n; i++){
    if(!sp[i]){

```

```

prime.push_back(i);
e[i] = 1; phi[i] = i-1; mu[i] = -1; tau[i] = 2; sigma[i] =
i+1;
}
for(auto j : prime){
    if(i*j >= sz) break;
    sp[i*j] = j;
    if(i % j == 0){
        e[i*j] = e[i]+1; phi[i*j] = phi[i]*j; mu[i*j] = 0;
        tau[i*j] = tau[i]/e[i*j]*(e[i*j]+1);
        sigma[i*j] = sigma[i]*(j-1)/(pw(j, e[i*j])-1)*(pw(j,
e[i*j]+1)-1)/(j-1); //overflow
        break;
    }
    e[i*j] = 1; phi[i*j] = phi[i] * phi[j]; mu[i*j] = mu[i] *
mu[j];
    tau[i*j] = tau[i] * tau[j]; sigma[i*j] = sigma[i] *
sigma[j];
}

```

7.7 확장 유clidean 알고리즘

Time Complexity: $O(\log \max(a, b))$

```

// 정수 a, b 주어지면 ax + by = gcd(a, b) = g
// 를 만족하는 정수 {g, x, y} 반환
tuple<ll, ll, ll> ext_gcd(ll a, ll b){
    if(b == 0) return {a, 1, 0};
    auto [g,x,y] = ext_gcd(b, a % b);
    return {g, y, x - a/b * y};
}

```

7.8 중국인의 나머지 정리

Time Complexity: $O(k \log m)$

```

// a = a1 (mod m1), a = a2 (mod m2)를 만족하는 {a, lcm(m1, m2)} 반환
// 만약 a가 존재하면 0 <= a < lcm(m1, m2) 에서 유일하게 존재
// a가 존재하지 않는 경우 {-1, -1} 반환
ll mod(ll a, ll b){ return (a % b) >= 0 ? a : a + b; }
pair<ll, ll> crt(ll a1, ll m1, ll a2, ll m2){
    ll g = gcd(m1, m2), m = m1 / g * m2;
    if((a2 - a1) % g) return {-1, -1};
    ll md = m2/g, s = mod((a2-a1)/g, m2/g);
    ll t = mod(get<1>(ext_gcd(m1/g%md, m2/g)), md);
    return {a1 + s * t % md * m1, m };
}
// a = a_i (mod m_i)를 만족하는 {a, lcm(m_1, ..., m_k)} 반환
// a가 존재하지 않는 경우 {-1, -1} 반환
pair<ll, ll> crt(const vector<ll> &a, const vector<ll> &m){
    ll ra = a[0], rm = m[0];
    for(int i=1; i<m.size(); i++){
        auto [aa, mm] = crt(ra, rm, a[i], m[i]);
        if(mm == -1) return {-1, -1}; else tie(ra, rm) = tie(aa, mm);
    }
    return {ra, rm};
}

```

7.9 Diophantine

```

// solutions to ax + by = c where x in [xlow, xhigh] and y in
[ylow, yhigh]
// cnt, leftsol, rightsol, gcd of a and b

```

```

template<class T> array<T, 6> solve_linear_diophantine(T a, T
b, T c, T xlow, T xhigh, T ylow, T yhigh){
    T g, x, y = euclid(a >= 0 ? a : -a, b >= 0 ? b : -b, x, y);
    array<T, 6> no_sol{0, 0, 0, 0, 0, g};
    if(c % g) return no_sol; x *= c / g, y *= c / g;
    if(a < 0) x = -x; if(b < 0) y = -y;
    a /= g, b /= g, c /= g;
    auto shift = [&](T &x, T &y, T a, T b, T cnt){ x += cnt * b, y -=
cnt * a; };
    int sign_a = a > 0 ? 1 : -1, sign_b = b > 0 ? 1 : -1;
    shift(x, y, a, b, (xlow - x) / b);
    if(x < xlow) shift(x, y, a, b, sign_b);
    if(x > xhigh) return no_sol;
    T lx1 = x; shift(x, y, a, b, (xhigh - x) / b);
    if(x > xhigh) shift(x, y, a, b, -sign_b);
    T rx1 = x; shift(x, y, a, b, -(ylow - y) / a);
    if(y < ylow) shift(x, y, a, b, -sign_a);
    if(y > yhigh) return no_sol;
    T lx2 = x; shift(x, y, a, b, -(yhigh - y) / a);
    if(y > yhigh) shift(x, y, a, b, sign_a);
    T rx2 = x; if(lx2 > rx2) swap(lx2, rx2);
    T lx = max(lx1, lx2), rx = min(rx1, rx2);
    if(lx > rx) return no_sol;
    return {(rx - lx) / (b >= 0 ? b : -b) + 1, lx, (c - lx * a) /
b, rx, (c - rx * a) / b, g};
}

```

7.10 Miller Rabin + Pollard Rho

```

# n<2^32: alist = [2, 7, 61]
# n<2^64
alist = [2, 325, 9375, 28178, 450775, 9780504, 1795265022]

```

```

def miller_rabin(n, a):
    d = (n-1)//2
    while d%2 == 0:
        if pow(a, d, n) == n-1: return 1
        d //= 2
    tmp = pow(a, d, n)
    return tmp == n-1 or tmp == 1

def is_prime(n):
    if n <= 1: return 0
    if n <= 10**10:
        for i in range(2, int(n**0.5)+1):
            if n%i == 0: return 0
            return 1
    for a in alist:
        if not miller_rabin(n, a): return 0
    return 1

from random import randint
from math import gcd

```

```

def PollardRho(n):
    y = x = randint(2, n)
    c = randint(1, n)
    d = 1
    while d==1:
        x = x*x%n+c
        if x >= n: x -= n

```

```

y = y*y%n+c
if y >= n: y -= n
y = y*y%n+c
if y >= n: y -= n
d = gcd(abs(x-y), n)
if is_prime(d): return d
return PollardRho(d)

res=[]
def getFactor(n):
    while n%2==0: n/=2; res.append(2)
    while n!=1 and not(is_prime(n)):
        d=PollardRho(n)
        while n%d==0: n/=d; res.append(d)
        if n!=1: res.append(n)

```

7.11 원시근, 이산로그, 이산제곱근

```

ll PrimitiveRoot(ll p){ // order p-1
vector<pair<ll, ll>> v = Factorize(p-1);
for(ll r=1; ; r++){
    bool flag = true; // Warning: 64bit Pow
    for(auto [d,e] : v) if(PowMod(r, (p-1)/d, p) == 1){ flag =
false; break; }
    if(flag) return r;
}
// Given A, B, P, solve A^x === B mod P, return smallest value
ll DiscreteLog(ll A, ll B, ll P){ // O(sqrt P) with hash set
__gnu_pbds::gp_hash_table<ll, __gnu_pbds::null_type> st;
ll t = ceil(sqrt(P)), k = 1; // use binary search?
for(int i=0; i<t; i++) st.insert(k), k = k * A % P;
ll inv = Pow(k, P-2, P);
for(int i=0, s=1; i<t; i++, s=s*inv%P){
    ll x = B * s % P;
    if(st.find(x) == st.end()) continue;
    for(int j=0, f=1; j<t; j++, f=f*A%P){
        if(f == x) return i * t + j;
    }
}
return -1;
}
// Given A, P, solve X^2 === A mod P, return arbitrary
ll DiscreteSqrt(ll A, ll P){//O(log^2P), O(logP) in random data
if(A == 0) return 0;
if(Pow(A, (P-1)/2, P) != 1) return -1;
if(P % 4 == 3) return Pow(A, (P+1)/4, P);
ll s = P - 1, n = 2, r = 0, m;
while(~s & 1) r++, s >>= 1;
while(Pow(n, (P-1)/2, P) != P-1) n++;
ll x = Pow(A, (s+1)/2, P), b = Pow(A, s, P), g = Pow(n, s,
P);
for(; ; r=m){
    ll t = b; for(m=0; m<r && t!=1; m++) t = t * t % P;
    if(!m) return x;
    ll gs = Pow(g, 1LL << (r-m-1), P);
    g = gs * gs % P; x = x * gs % P; b = b * g % P;
}
}

```

7.12 $O(N^3 \log 1/\epsilon)$ Polynomial Equation

```
/*
 * Author: Per Austrin
 * License: CCO
 * Source:
https://github.com/kth-competitive-programming/kactl/blob/main/content/numerical/PolyRoots.h
 */

vector<double> poly_root(vector<double> p, double xmin, double xmax){
    if(p.size() == 2){ return {-p[0] / p[1]}; }
    vector<double> ret, der(p.size()-1);
    for(int i=0; i<der.size(); i++) der[i] = p[i+1] * (i + 1);
    auto dr = poly_root(der, xmin, xmax);
    dr.push_back(xmin-1); dr.push_back(xmax+1);
    sort(dr.begin(), dr.end());
    for(int i=0; i+1<dr.size(); i++){
        double l = dr[i], h = dr[i+1]; bool sign = calc(p, l) > 0;
        if (sign ^ (calc(p, h) > 0)){}
            for(int it=0; it<60; it++){ // while(h-l > 1e-8)
                double m = (l + h) / 2, f = calc(p, m);
                if ((f <= 0) ^ sign) l = m; else h = m;
            }
            ret.push_back((l + h) / 2);
        }
    }
    return ret;
}
```

7.13 가우스 소거법 - RREF, 랭크, 행렬식, 역행렬

Time Complexity: $O(N^3)$

```
// T Add(T a, T b), Sub, Mul, Div 구현해야 함
// bool IsZero(T x) 구현해야 함
template<typename T> // return {rref, rank, det, inv}
tuple<vector<vector<T>>, int, T, vector<vector<T>>>
Gauss(vector<vector<T>> a, bool square=true){
    int n = a.size(), m = a[0].size(), rank = 0;
    vector<vector<T>> out(n, vector<T>(m, 0)); T det = T(1);
    for(int i=0; i<n; i++) if(square) out[i][i] = T(1);
    for(int i=0; i<m; i++){
        if(rank == n) break;
        if(IsZero(a[rank][i])){}
            T mx = T(0); int idx = -1; // fucking precision error
            for(int j=rank+1; j<n; j++) if(mx < abs(a[j][i])) mx =
abs(a[j][i]), idx = j;
            if(idx == -1 || IsZero(a[idx][i])){ det = 0; continue; }
            for(int k=0; k<m; k++)
                a[rank][k] = Add(a[rank][k], a[idx][k]);
            if(square) out[rank][k] = Add(out[rank][k],
out[idx][k]);
        }
    }
    det = Mul(det, a[rank][i]);
    T coeff = Div(T(1), a[rank][i]);
    for(int j=0; j<m; j++) a[rank][j] = Mul(a[rank][j], coeff);
    for(int j=0; j<m; j++) if(square) out[rank][j] =
Mul(out[rank][j], coeff);
    for(int j=0; j<n; j++) if(rank == j) continue;
```

```
T t = a[j][i]; // Warning: [j][k], [rank][k]
for(int k=0; k<m; k++) a[j][k] = Sub(a[j][k],
Mul(a[rank][k], t));
for(int k=0; k<m; k++) if(square) out[j][k] =
Sub(out[j][k], Mul(out[rank][k], t));
rank++;
}
return {a, rank, det, out};
}
```

7.14 다항식 곱셈(FFT)

Time Complexity: $O(N \log N)$

```
// 104,857,601 = 25 * 2^22 + 1, w = 3 | 998,244,353 = 119
* 2^23 + 1, w = 3
// 2,281,701,377 = 17 * 2^27 + 1, w = 3 | 2,483,027,969 = 37
* 2^26 + 1, w = 3
// 2,113,929,217 = 63 * 2^25 + 1, w = 5 | 1,092,616,193 = 521
* 2^21 + 1, w = 3
using real_t = double; using cpx = complex<real_t>;
void FFT(vector<cpx> &a, bool inv_fft=false){
    int N = a.size(); vector<cpx> root(N/2);
    for(int i=1, j=0; i<N; i++){
        int bit = N / 2;
        while(j >= bit) j -= bit, bit >>= 1;
        if(i < (j += bit)) swap(a[i], a[j]);
    }
    long double ang = 2 * acosl(-1) / N * (inv_fft ? -1 : 1);
    for(int i=0; i<N/2; i++) root[i] = cpx(cosl(ang*i),
sinl(ang*i));
    /*
    NTT : ang = pow(w, (mod-1)/n) % mod, inv_fft -> ang^{-1},
root[i] = root[i-1] * ang
    XOR Convolution : set roots[*] = 1, a[j+k] = u+v, a[j+k+i/2] =
u-v
    OR Convolution : set roots[*] = 1, a[j+k+i/2] += inv_fft ?
-u : u;
    AND Convolution : set roots[*] = 1, a[j+k] += inv_fft ? -v :
v;
    */
    for(int i=2; i<=N; i<<=1){
        int step = N / i;
        for(int j=0; j<N; j+=i) for(int k=0; k<i/2; k++){
            cpx u = a[j+k], v = a[j+k+i/2] * root[step * k];
            a[j+k] = u+v; a[j+k+i/2] = u-v;
        }
        if(inv_fft) for(int i=0; i<N; i++) a[i] /= N; // skip for
AND/OR convolution.
    }
    vector<ll> multiply(const vector<ll> &_a, const vector<ll>
&_b){
        vector<cpx> a(all(_a)), b(all(_b));
        int N = 2; while(N < a.size() + b.size()) N <<= 1;
        a.resize(N); b.resize(N); FFT(a); FFT(b);
        for(int i=0; i<N; i++) a[i] *= b[i];
        vector<ll> ret(N); FFT(a, 1); // NTT : just return a
        for(int i=0; i<N; i++) ret[i] = llround(a[i].real());
        while(ret.size() > 1 && ret.back() == 0) ret.pop_back();
    }
}
```

```
return ret;
}
// 더 높은 정밀도
vector<ll> multiply_mod(const vector<ll> &a, const vector<ll>
&b, const ull mod){
    int N = 2; while(N < a.size() + b.size()) N <<= 1;
    vector<cpx> v1(N), v2(N), r1(N), r2(N);
    for(int i=0; i<a.size(); i++) v1[i] = cpx(a[i] >> 15, a[i] &
32767);
    for(int i=0; i<b.size(); i++) v2[i] = cpx(b[i] >> 15, b[i] &
32767);
    FFT(v1); FFT(v2);
    for(int i=0; i<N; i++){
        int j = i ? N-i : i;
        cpx ans1 = (v1[i] + conj(v1[j])) * cpx(0.5, 0);
        cpx ans2 = (v1[i] - conj(v1[j])) * cpx(0, -0.5);
        cpx ans3 = (v2[i] + conj(v2[j])) * cpx(0.5, 0);
        cpx ans4 = (v2[i] - conj(v2[j])) * cpx(0, -0.5);
        r1[i] = (ans1 * ans3) + (ans1 * ans4) * cpx(0, 1);
        r2[i] = (ans2 * ans3) + (ans2 * ans4) * cpx(0, 1);
    }
    vector<ll> ret(N); FFT(r1, true); FFT(r2, true);
    for(int i=0; i<N; i++){
        ll av = llround(r1[i].real()) % mod;
        ll bv = ( llround(r1[i].imag()) + llround(r2[i].real()) ) %
mod;
        ll cv = llround(r2[i].imag()) % mod;
        ret[i] = (av << 30) + (bv << 15) + cv;
        ret[i] %= mod; ret[i] += mod; ret[i] %= mod;
    }
    while(ret.size() > 1 && ret.back() == 0) ret.pop_back();
    return ret;
}
```

7.15 이항 계수를 소수로 나눈 나머지(Lucas' theorem)

Time Complexity: 전처리 $O(P)$, 쿼리 $O(\log P)$

```
// Lucas C(13);
// C.calc(5, 3) = 5C3 = 10
// C.calc(10, 2) = 10C2 % 13 = 45 % 13 = 6
// 주의: P는 소수
// P가 크고(약 10억) n,r이 작으면(1000만 이하)
// 생성자에서 fac, inv를 1000만까지만 구해도 됨
struct Lucas{ // init : O(P), query : O(log P)
    const size_t P;
    vector<ll> fac, inv;
    ll Pow(ll a, ll b){
        ll res = 1;
        for(; b; b>>=1, a=a*a%P) if(b & 1) res = res * a % P;
        return res;
    }
    Lucas(size_t P) : P(P), fac(P), inv(P) {
        fac[0] = 1; for(int i=1; i<P; i++) fac[i] = fac[i-1] *
i % P;
        inv[P-1] = Pow(fac[P-1], P-2); for(int i=P-2; ~i; i--)
        inv[i] = inv[i+1] * (i+1) % P;
    }
}
```

```

    ll small(ll n, ll r) const { return r <= n ? fac[n] *
inv[r] % P * inv[n-r] % P : 0LL; }
    ll calc(ll n, ll r) const {
        if(n < r || n < 0 || r < 0) return 0;
        if(!n || !r || n == r) return 1; else return small(n%P,
r%P) * calc(n/P, r/P) % P;
    }
};

```

7.16 이항 계수 관련 공식

흡수 항등식: $\binom{r}{k} = \frac{r}{k} \binom{r-1}{k-1}$, 정수 $k \neq 0$
덧셈 공식: $\binom{r}{k} = \binom{r-1}{k} + \binom{r-1}{k-1}$
 $\binom{r}{k} = (-1)^k \binom{k-r-1}{k}$
 $\binom{r}{m} \binom{m}{k} = \binom{r}{k} \binom{r-k}{m-k}$
 $\sum_{k \leq n} \binom{r+k}{n} = \binom{r+n+1}{n}$, 정수 n
 $\sum_{0 \leq k \leq n} \binom{k}{m} = \binom{n+1}{m+1}$, 정수 $m, n \geq 0$
Vandermonde's Identity: $\binom{m+n}{r} = \sum_{k=0}^r \binom{m}{k} \binom{n}{r-k}$
 $m=n, r=n$ 을 대입하면 $\binom{2n}{n} = \sum_{k=0}^n \binom{n}{k} \binom{n}{n-k} = \sum_{k=0}^n \binom{n}{k}^2$
 $\sum_k \binom{r}{m+k} \binom{s}{n-k} = \binom{r+s}{m+n}$, 정수 m, n
 $\sum_k \binom{l}{m+k} \binom{s}{n+k} = \binom{l+s}{l-m+n}$, 정수 $l \geq 0$, 정수 m, n
 $\sum_k \binom{l}{m+k} \binom{s+k}{n} (-1)^k = (-1)^{l+m} \binom{s-m}{n-l}$, 정수 $l \geq 0$, 정수 m, n
 $\sum_{k \leq l} \binom{l-k}{m} \binom{s}{k-n} (-1)^k = (-1)^{l+m} \binom{s-m-1}{l-m-n}$, 정수 $l, m, n \geq 0$
 $\sum_{-q \leq k \leq l} \binom{l-k}{m} \binom{q+k}{n} = \binom{l+q+1}{m+n+1}$, 정수 $m, n \geq 0$, 정수 $l+q \geq 0$

7.17 Partition Number

Time Complexity: $O(N\sqrt{N})$

```

for(int j=1; j*(3*j-1)/2<=i; j++) P[i] +=
(j%2?1:-1)*P[i-j*(3*j-1)/2], P[i] %= MOD;
for(int j=1; j*(3*j+1)/2<=i; j++) P[i] +=
(j%2?1:-1)*P[i-j*(3*j+1)/2], P[i] %= MOD;
vector<ModInt> res(sz+1); res[0] = 1; int sq=sqrt(sz);
vector<vector<ModInt>> p(2, vector<ModInt>(sz+1)), d=p;
for(int k=1; k<sq; k++) { p[0][0] = k == 1; // calc p[k][n]
    for(int n=1; n<sz; n++) {
        p[k][n] = p[k-1][n-1] + (n-k>=0 ? p[k-1][n-k] : 0);
        res[n] += p[k][n]; }
    for(int a=sq; a>=0; a--) for(int b=sq; b<=sz; b++) {
        d[a&1][b] = d[a&1][b-sq] + p[sq-1][b-1] + (b-a-1)>=0 ?
d[a&1][b-a-1] : 0;
        if(a == 0) res[b] += d[a&1][b];
    }
}

```

7.18 De Bruijn Sequence

```

// Create cyclic string of length k^n that contains every
length n string as substring. alphabet = [0, k - 1]
int res[10000000], aux[10000000]; // >= k^n
int de_bruijn(int k, int n) { // Returns size (k^n)
    if(k == 1) { res[0] = 0; return 1; }
    for(int i = 0; i < k * n; i++) aux[i] = 0;
    int sz = 0;
    function<void(int, int)> db = [&](int t, int p) {
        if(t > n) {
            if(n % p == 0) for(int i=1; i<=p; i++) res[sz++] = aux[i];
        }
    };
    // regex_match: 주어진 문자열이 정규표현식 패턴을 만족하면 true를
    // 반환
}

```

```

    else {
        aux[t] = aux[t - p]; db(t + 1, p);
        for(int i=aux[t-p]+1; i<k; i++) aux[t]=i, db(t+1, t);
    }
}; db(1, 1); return sz;
}

```

7.19 지그재그 순열의 개수 구하기

Usage: 지그재그 순열: $a_1 < a_2 > a_3 < a_4 > \dots$ 또는 $a_1 > a_2 < a_3 > a_4 < \dots$

```

MAX = 101
E = [[0]*MAX for _ in range(MAX)] #Entringer Number
E[0][0] = 1
for n in range(1, MAX):
    for k in range(1, n+1): E[n][k] = E[n][k-1] + E[n-1][n-k]
# 2*E[N][N]: 1부터 N까지 자연수를 나열한 수열이 지그재그 순열이 되는
경우의 수

```

8 문자열

8.1 정규 표현식

```

#include <bits/stdc++.h>
#define fastio ios::sync_with_stdio(0), cin.tie(0), cout.tie(0)
using namespace std;

```

// sregex_iterator 사용법

```

int main() {
    fastio;
    regex r("((^\\w+)//([[:/]]+)(:\\d+)?(/.+)?$)");
    int n; cin >> n;
    for (int cnt = 1; cnt <= n; cnt++) {
        string s, ans[4]; cin >> s;
        auto it = *sregex_iterator(s.begin(), s.end(), r);
        for (int i = 0; i < 4; i++) {
            ans[i] = it[i + 1];
            if (ans[i].empty()) ans[i] = "<default>";
        }
        if (ans[2][0] == ':') ans[2] = ans[2].substr(1);
        if (ans[3][0] == '/') ans[3] = ans[3].substr(1);
        cout << "URL #" << cnt << '\n';
        cout << "Protocol = " << ans[0] << '\n';
        cout << "Host = " << ans[1] << '\n';
        cout << "Port = " << ans[2] << '\n';
        cout << "Path = " << ans[3] << '\n' << '\n';
    }
}

```

//regex_match 사용법

```

int main() {
    fastio;
    int N; cin >> N;
    regex r("^(A-F)?A+F+C+[A-F]?$");
    while (N--) {
        string s; cin >> s;
        regex_match(s, r) ? cout << "Infected!\n" : cout <<
"Good\n";
    }
}

```

// regex_match: 주어진 문자열이 정규표현식 패턴을 만족하면 true를 반환

// s = regex_replace(s, r, t): 바꿀 문자열 s, 적용할 정규표현식 r, 치환할 패턴 t

- ^x // ^은 문자열의 시작을 표현하며, x문자로 시작됨을 의미
- x\$ // \$은 문자열의 종료를 표현하며, x문자로 종료됨을 의미
- .x // .은 개행문자 \n을 제외한 다른 모든 문자를 의미
- x* // +은 1회 이상 반복을 의미, x문자가 1번 이상 반복됨을 의미 ({1,}과 동일)
- x* // *은 0회 이상 반복을 의미, x문자가 0번 이상 반복됨을 의미 ({0,}과 동일)
- x? // ?은 0 or 1개 문자 매칭 의미, x문자가 존재할 수도 있고 안할수도 있다는 의미 ({0,1}과 동일)
- x|y // |은 or를 표현, x 또는 y가 나온다는 의미
- (x) // ()은 그룹을 표현, 괄호로 묶인 패턴을 의미 ((abc){3}와 같이 사용해서 abcabcabc를 검출하는데 쓰임)
- x{n} // {}은 반복을 의미, x가 n번 반복됨을 의미
- x{n,m} // {}은 반복을 의미, x가 n번 이상 m번 이하로 반복됨을 의미
- [xy] // []은 x또는 y를 찾는다는 의미, [a-z0-9]이면 알파벳 소문자 또는 숫자를 찾는다는 의미
- [~xy] // [~]은 not을 의미, x 및 y 를 제외하고 찾는다는 의미
- [a-z] // [-]은 a ~ z 를 찾는다는 의미
- \d // \d은 digit으로 숫자를 의미
- \D // \D은 not digit으로 숫자를 제외하고 나머지 다른 문자를 의미
- \s // \s은 space로 공백문자를 의미
- \S // \S은 not space로 공백문자를 제외한 나머지 다른 문자를 의미
- \t // \t은 tab을 의미
- \w // \w은 알파벳 대문자,소문자와 숫자를 의미, [A-Za-z0-9]을 의미
- \W // \W은 not \w, 즉 \w를 제외한 특수문자를 의미
- (?:) // 캡쳐하지 않는 그룹 생성

8.2 문자열 해싱

Time Complexity: build: $O(N)$, get: $O(1)$

```

// 전처리 O(N), 부분 문자열의 해시값을 O(1)에 구함
// Hashing<917, 998244353> H;
// H.build("ABCDABCD");
// assert(H.get(1, 4) == H.get(5, 8));
// 주의: get 함수의 인자는 1-based 달희 구간
// 주의: M은 10억 근처의 소수, P는 M과 서로소

// 1e5+3, 1e5+13, 131'071, 524'287, 1'299'709, 1'301'021
// 1e9-63, 1e9+7, 1e9+9, 1e9+103
template<long long P, long long M> struct Hashing {
    vector<long long> h, p;
    void build(const string &s) {
        int n = s.size();
        h = p = vector<long long>(n+1); p[0] = 1;
        for(int i=1; i<=n; i++) h[i] = (h[i-1] * P + s[i-1]) %
M;
        for(int i=1; i<=n; i++) p[i] = p[i-1] * P % M;
    }
    long long get(int s, int e) const {
        long long res = (h[e] - h[s-1] * p[e-s+1]) % M;
        return res >= 0 ? res : res + M;
    }
};

```

8.3 문자열 매칭 - KMP

Time Complexity: GetFail: $O(|P|)$, $O(|S| + |P|)$

```
// s에서 p가 등장하는 위치 반환
// KMP("ABABCAB", "AB") = {0, 2, 5}
// KMP("AAAAA", "AA") = {0, 1, 2}
```

```
vector<int> GetFail(const string &p){
    int n = p.size();
    vector<int> fail(n);
    for(int i=1, j=0; i<n; i++){
        while(j && p[i] != p[j]) j = fail[j-1];
        if(p[i] == p[j]) fail[i] = ++j;
    }
    return fail;
}
```

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

8.4 가장 긴 팰린드롬 부분 문자열 - Manacher

Time Complexity: $O(N)$

```
// 각 문자를 중심으로 하는 최장 팰린드롬의 반경을 반환
// Manacher("abaaba") = {0,1,0,3,0,1,6,1,0,3,0,1,0}
// # a # b # a # a # b # a #
// 0 1 0 3 0 1 6 1 0 3 0 1 0
vector<int> Manacher(const string &inp){
    int n = inp.size() * 2 + 1;
    vector<int> ret(n);
    string s = "#";
    for(auto i : inp) s += i, s += "#";
    for(int i=0, p=-1, r=-1; i<n; i++){
        ret[i] = i <= r ? min(r-i, ret[2*p-i]) : 0;
        while(i-ret[i]-1 >= 0 && i+ret[i]+1 < n &&
s[i-ret[i]-1] == s[i+ret[i]+1]) ret[i]++;
        if(i+ret[i] > r) r = i+ret[i], p = i;
    }
    return ret;
}
```

8.5 문자열 매칭 - Z

Time Complexity: $O(N)$

```
// Z[i] = LongestCommonPrefix(S[0:N], S[i:N])
//      = S[0:N]과 S[i:N]이 앞에서부터 몇 글자 겹치는지
vector<int> Z(const string &s){
    int n = s.size();
    vector<int> z(n);
    z[0] = n;
    for(int i=1, l=0, r=0; i<n; i++) {
```

```
        if(i < r) z[i] = min(r-i-1, z[i-1]);
        while(i+z[i] < n && s[i+z[i]] == s[z[i]]) z[i]++;
        if(i+z[i] > r) r = i+z[i], l = i;
    }
    return z;
}
```

8.6 Aho-Corasick

```
struct Node{
    int g[26], fail, out;
    Node() { memset(g, 0, sizeof g); fail = out = 0; }
};
vector<Node> T(2); int aut[100101][26];
void Insert(int n, int i, const string &s){
    if(i == s.size()) { T[n].out++; return; }
    int c = s[i] - 'a';
    if(T[n].g[c] == 0) T[n].g[c] = T.size(), T.emplace_back();
    Insert(T[n].g[c], i+1, s);
}
int go(int n, int i){ // DO NOT USE `aut` DIRECTLY
    int &res = aut[n][i]; if(res) return res;
    if(n != 1 && T[n].g[i] == 0) res = go(T[n].fail, i);
    else if(T[n].g[i] != 0) res = T[n].g[i]; else res = 1;
    return res;
}
void Build(){
    queue<int> q; q.push(1); T[1].fail = 1;
    while(!q.empty()){
        int n = q.front(); q.pop();
        for(int i=0; i<26; i++){
            int next = T[n].g[i]; if(next == 0) continue;
            if(n == 1) T[next].fail=1; else
T[next].fail=go(T[n].fail,i);
            q.push(next); T[next].out += T[T[next].fail].out;
        } /* for i */ /* while q */ /* build */
    }
}
bool Find(const string &s){
    int n = 1, ok = 0;
    for(int i=0; i<s.size(); i++){
        n = go(n, s[i] - 'a'); if(T[n].out != 0) ok = 1;
    }
    return ok;
}
```

8.7 접미사 배열

Time Complexity: $O(N \log N)$

```
// LCP는 1-based
pair<vector<int>, vector<int>> SuffixArray(const string &s){ // O(N log N)
    int n = s.size(), m = max(n, 256);
    vector<int> sa(n), lcp(n), pos(n), tmp(n), cnt(m);
    auto counting_sort = [&](){
        fill(cnt.begin(), cnt.end(), 0);
        for(int i=0; i<n; i++) cnt[pos[i]]++;
        partial_sum(cnt.begin(), cnt.end(), cnt.begin());
        for(int i=n-1; i>=0; i--) sa[--cnt[pos[tmp[i]]]] = tmp[i];
    };
    for(int i=0; i<n; i++) sa[i] = i, pos[i] = s[i], tmp[i] = i;
    counting_sort();
    for(int k=1; ; k<=1){
        int p = 0;
```

```
        for(int i=n-k; i<n; i++) tmp[p++] = i;
        for(int i=0; i<n; i++) if(sa[i] >= k) tmp[p++] = sa[i] - k;
        counting_sort();
        tmp[sa[0]] = 0;
        for(int i=1; i<n; i++){
            tmp[sa[i]] = tmp[sa[i-1]];
            if(sa[i-1]+k < n && sa[i]+k < n && pos[sa[i-1]] == pos[sa[i]] && pos[sa[i-1]+k] == pos[sa[i]+k]) continue;
            tmp[sa[i]] += 1;
        }
        swap(pos, tmp); if(pos[sa.back()] + 1 == n) break;
    }
    for(int i=0, j=0; i<n; i++, j=max(j-1, 0)){
        if(pos[i] == 0) continue;
        while(sa[pos[i]-1]+j < n && sa[pos[i]]+j < n &&
s[sa[pos[i]-1]+j] == s[sa[pos[i]]+j]) j++;
        lcp[pos[i]] = j;
    }
    return {sa, lcp};
}
```

8.8 All LCS

```
// get LCS(A[0..i], B[l..r]) in O(N^2)
void AllLCS(const string &s, const string &t){
    vector<int> h(t.size()); iota(h.begin(), h.end(), 0);
    for(int i=0, v=-1; i<s.size(); i++, v=-1){
        for(int r=0; r<t.size(); r++){
            if(s[i] == t[r] || h[r] < v) swap(h[r], v);
            //LCS(s[0..i], t[l..r]) = r-l+1 - sum([h[x] >= 1] | x <=
r)
        } /* for r */ /* for i */ /* end */
    }
}
```

9 계산 기하

9.1 2차원 계산 기하 템플릿 + CCW

```
#include <bits/stdc++.h>
#define x first
#define y second
using namespace std;
using ll = long long;
using Point = pair<ll, ll>

Point operator + (Point p1, Point p2){ return {p1.x + p2.x,
p1.y + p2.y}; }
Point operator - (Point p1, Point p2){ return {p1.x - p2.x,
p1.y - p2.y}; }
ll operator * (Point p1, Point p2){ return p1.x * p2.x + p1.y *
p2.y; } // 내적
ll operator / (Point p1, Point p2){ return p1.x * p2.y - p2.x *
p1.y; } // 외적
int Sign(ll v){ return (v > 0) - (v < 0); } // 양수면 +1, 음수면
-1, 0이면 0 반환
ll Dist(Point p1, Point p2){ return (p2 - p1) * (p2 - p1); } //
두 점 거리 계급
ll SignedArea(Point p1, Point p2, Point p3){ return (p2 - p1) /
(p3 - p1); }
int CCW(Point p1, Point p2, Point p3){ return
Sign(SignedArea(p1, p2, p3)); }
```

9.2 360도 각도 정렬

```
/* y축
   ↑
3 2 1
4 -1 0 → x축
5 6 7
원점 기준 각도 정렬
x축 위에 있는 점이 가장 먼저, 그리고 반시계 방향으로
*/
int QuadrantID(const Point p){
    static int arr[9] = { 5, 4, 3, 6, -1, 2, 7, 0, 1 };
    return arr[Sign(p.x)*3+Sign(p.y)+4];
}
```

```
sort(points.begin(), points.end(), [&](auto p1, auto p2){
    if(QuadrantID(p1) != QuadrantID(p2)) return QuadrantID(p1) <
QuadrantID(p2);
    else return p1 / p2 > 0; // 반시계
});
```

9.3 다각형 넓이

```
// 다각형의 넓이의 2배를 반환, 항상 정수, O(N)
ll PolygonArea(const vector<Point> &v){
    ll res = 0;
    for(int i=0; i<v.size(); i++) res += v[i] /
v[(i+1)%v.size()];
    return abs(res);
}
```

9.4 선분 교차 판정

```
// 선분 교차 - 선분 ab와 선분 cd가 만나면 true
bool Cross(Point s1, Point e1, Point s2, Point e2){
    int ab = CCW(s1, e1, s2) * CCW(s1, e1, e2);
    int cd = CCW(s2, e2, s1) * CCW(s2, e2, e1);
    if(ab == 0 && cd == 0){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        return !(e1 < s2 || e2 < s1);
    }
    return ab <= 0 && cd <= 0;
}
// 교차하지 않으면 0
// 교점이 무한히 많으면 -1
// 교점이 1개면 1 반환하고 res에 교점 저장
int Cross(Point s1, Point e1, Point s2, Point e2, pair<double,
double> &res){
    if(!Cross(s1, e1, s2, e2)) return 0;
    ll det = (e1 - s1) / (e2 - s2);
    if(!det){
        if(s1 > e1) swap(s1, e1);
        if(s2 > e2) swap(s2, e2);
        if(e1 == s2){ res = s2; return 1; }
        if(e2 == s1){ res = s1; return 1; }
        return -1;
    }
    res.x = s1.x + (e1.x - s1.x) * ((s2 - s1) / (e2 - s2) * 1.0
/ det);
    res.y = s1.y + (e1.y - s1.y) * ((s2 - s1) / (e2 - s2) * 1.0
/ det);
    return 1;
}
```

9.5 Intersect Series

```
P perp() const { return P(-y, x); }
#define arg(p, q) atan2(p.cross(q), p.dot(q))
bool circleIntersect(P a, P b, double r1, double r2, pair<P, P>
out){
    if (a == b) { assert(r1 != r2); return false; }
    P vec = b-a; double d2 = vec.dist2(), sum = r1+r2, dif =
r1-r2;
    double p = (d2 + r1*r1 - r2*r2)/(d2*2), h2 = r1*r1 - p*p*d2;
    if (sum*sum < d2 || dif*dif > d2) return false; // use EPS
plz...
    P mid = a + vec*p, per = vec.perp() * sqrt(fmax(0, h2) / d2);
    *out = {mid + per, mid - per}; return true;
}
vector<P> circleLine(P c, double r, P a, P b) {
    P ab = b - a, p = a + ab * (c-a) * ab / D2(ab);
    T s = (b - a) / (c - a), h2 = r*r - s*s / D2(ab);
    if (abs(h2) < EPS) return {p}; if (h2 < 0) return {};
    P h = ab / D1(ab) * sqrtl(h2); return {p - h, p + h};
} // use circleLine if you use double...
int CircleLineIntersect(P o, T r, P p1, P p2, bool segment){
    P s = p1, d = p2 - p1; // line : s + kd, int support
    T a = d * d, b = (s - o) * d * 2, c = D2(s, o) - r * r;
    T det = b * b - 4 * a * c; // solve ak^2 + bk + c = 0, a > 0
    if(!segment) return Sign(det) + 1;
    if(det <= 0) return det ? 0 : 0 <= -b && -b <= a + a;
    bool f11 = b <= 0 || b * b <= det;
    bool f21 = b <= 0 && b * b >= det;
    bool f12 = a+a+b >= 0 && det <= (a+a+b) * (a+a+b);
    bool f22 = a+a+b >= 0 || det >= (a+a+b) * (a+a+b);
    return (f11 && f12) + (f21 && f22);
} // do not use this if you want to use double...
double circlePoly(P c, double r, vector<P> ps){ // return area
auto tri = [&](P p, P q) { // ps must be ccw polygon
    auto r2 = r * r / 2; P d = q - p;
    auto a = d.dot(p)/d.dist2(), b = (p.dist2()-r*r)/d.dist2();
    auto det = a * a - b;
    if (det <= 0) return arg(p, q) * r2;
    auto s = max(0., -a-sqrt(det)), t = min(1., -a+sqrt(det));
    if (t < 0 || 1 <= s) return arg(p, q) * r2;
    P u = p + d * s, v = p + d * t;
    return arg(p,u) * r2 + u.cross(v)/2 + arg(v,q) * r2;
};
auto sum = 0.0;
rep(i,0,sz(ps)) sum += tri(ps[i] - c, ps[(i+1)%sz(ps)] - c);
return sum;
}
// extrVertex: point of hull, max projection onto line
#define cmp(i,j) sgn(dir.perp().cross(poly[(i)%n]-poly[(j)%n]))
#define extr(i) cmp(i + 1, i) >= 0 && cmp(i, i - 1 + n) < 0
int extrVertex(vector<P>& poly, P dir) {
    int n = sz(poly), lo = 0, hi = n;
    if (extr(0)) return 0;
    while (lo + 1 < hi) {
        int m = (lo + hi) / 2; if (extr(m)) return m;
        int ls = cmp(lo + 1, lo), ms = cmp(m + 1, m);
        (ls < ms || (ls == ms && ls == cmp(lo, m))) ? hi : lo = m;
    }
    return lo;
}
```

```
//(-1,-1): no collision
//(i,-1): touch corner
//(i,i): along side (i,i+1)
//(i,j): cross (i,i+1)and(j,j+1)
//(i,i+1): cross corner i
// O(log n), ccw no colinear point convex polygon
// P perp() const { return P(-y, x); }
#define cmpL(i) sgn(a.cross(poly[i], b))
array<int, 2> lineHull(P a, P b, vector<P>& poly) { // O(log N)
    int endA = extrVertex(poly, (a - b).perp());
    int endB = extrVertex(poly, (b - a).perp());
    if (cmpL(endA) < 0 || cmpL(endB) > 0) return {-1, -1};
    array<int, 2> res;
    rep(i,0,2) {
        int lo = endB, hi = endA, n = sz(poly);
        while ((lo + 1) % n != hi) {
            int m = ((lo + hi + (lo < hi ? 0 : n)) / 2) % n;
            (cmpL(m) == cmpL(endB) ? lo : hi) = m;
        }
        res[i] = (lo + !cmpL(hi)) % n;
        swap(endA, endB);
    }
    if (res[0] == res[1]) return {res[0], -1};
    if (!cmpL(res[0]) && !cmpL(res[1])) {
        switch ((res[0] - res[1] + sz(poly) + 1) % sz(poly)) {
            case 0: return {res[0], res[0]};
            case 2: return {res[1], res[1]};
        }
    }
    return res;
}
```

9.6 Segment Distance, Segment Reflect

```
double Proj(Point a, Point b, Point c){
    ll t1 = (b - a) * (c - a), t2 = (a - b) * (c - b);
    if(t1 * t2 >= 0 && CCW(a, b, c) != 0)
        return abs(CCW(a, b, c)) / sqrt(Dist(a, b));
    else return 1e18; // INF
}
double SegmentDist(Point a[2], Point b[2]){
    double res = 1e18; // NOTE: need to check intersect
    for(int i=0; i<4; i++)
        res=min(res,sqrt(Dist(a[i/2],b[i%2])));
    for(int i=0; i<2; i++) res = min(res, Proj(a[0], a[1],
b[i]));
    for(int i=0; i<2; i++) res = min(res, Proj(b[0], b[1],
a[i]));
    return res;
}
P Reflect(P p1, P p2, P p3){ // line p1-p2, point p3
    auto [x1,y1] = p1; auto [x2,y2] = p2; auto [x3,y3] = p3;
    auto a = y1-y2, b = x2-x1, c = x1 * (y2-y1) + y1 * (x1-x2);
    auto d = a * y3 - b * x3;
    T x = -(a*c+b*d) / (a*a+b*b), y = (a*d-b*c) / (a*a+b*b);
    return 2 * P(x,y) - p3;
}
```

9.7 다각형 내부 판별

```
// 다각형 내부 또는 경계 위에 p가 있으면 true, O(N)
bool PointInPolygon(const vector<Point> &v, Point p){
    int n = v.size(), cnt = 0;
    Point p2(p.x+1, 1'000'000'000 + 1); // 좌표 범위보다 큰 수
    for(int i=0; i<n; i++){
        int j = i + 1 < n ? i + 1 : 0;
        if(min(v[i],v[j]) <= p && p <= max(v[i],v[j]) &&
CCW(v[i], v[j], p) == 0) return true;
        if(SegmentIntersection(v[i], v[j], p, p2)) cnt++;
    }
    return cnt % 2 == 1;
}
```

9.8 볼록 껍질 - Graham Scan

```
// 모든 점을 포함하는 가장 작은 볼록 다각형, O(N log N)
vector<Point> ConvexHull(vector<Point> points){
    if(points.size() <= 1) return points;
    swap(points[0], *min_element(points.begin(),
points.end()));
    sort(points.begin()+1, points.end(), [&](auto a, auto b){
        int dir = CCW(points[0], a, b);
        if(dir != 0) return dir > 0;
        return Dist(points[0], a) < Dist(points[0], b);
    });
    vector<Point> hull;
    for(auto p : points){
        while(hull.size() >= 2 && CCW(hull[hull.size()-2],
hull.back(), p) <= 0) hull.pop_back();
        hull.push_back(p);
    }
    return hull;
}
```

9.9 가장 먼 두 점 - Rotating Calipers

```
// 가장 먼 두 점을 구하는 함수, O(N)
// 주의: hull은 반시계 방향으로 정렬된 볼록 다각형이어야 함
pair<Point, Point> Calipers(vector<Point> hull){
    int n = hull.size(); ll mx = 0; Point a, b;
    for(int i=0, j=0; i<n; i++){
        while(j + 1 < n && (hull[i+1] - hull[i]) / (hull[j+1] -
hull[j]) >= 0){
            ll now = Dist(hull[i], hull[j]);
            if(now > mx) mx = now, a = hull[i], b = hull[j];
            j++;
        }
        ll now = Dist(hull[i], hull[j]);
        if(now > mx) mx = now, a = hull[i], b = hull[j];
    }
    return {a, b};
}
```

9.10 볼록 다각형 내부 판별

```
// 다각형 내부 또는 경계 위에 p가 있으면 true, O(log N)
// 주의: v는 반시계 방향으로 정렬된 볼록 다각형이어야 함
bool PointInConvexPolygon(const vector<Point> &v, const Point
&pt){
    if(CCW(v[0], v[1], pt) < 0) return false; int l = 1, r =
v.size() - 1;
    while(l < r){

```

```
        int m = l + r + 1 >> 1;
        if(CCW(v[0], v[m], pt) >= 0) l = m; else r = m - 1;
    }
    if(l == v.size() - 1) return CCW(v[0], v.back(), pt) == 0
&& v[0] <= pt && pt <= v.back();
    return CCW(v[0], v[l], pt) >= 0 && CCW(v[l], v[l+1], pt) >=
0 && CCW(v[l+1], v[0], pt) >= 0;
}
```

9.11 Segment In Polygon

```
// WARNING: C.push_back(C[0]) before call function
bool segment_in_polygon_non_strict(vector<P> &C, P s, P e){
    if(!pip(C, s) || !pip(C, e)) return false;
    if(s == e) return true; P d = e - s;
    vector<pair<frac, int>> v; auto g=raypoints(C, s, d, v);
    for(auto [fr,ev] : v){ // in(06) out(27)
        if(fr.first < 0 || g < fr) continue;
        if(ev == 4) return false; // pass outside corner
        if(fr < g && (ev == 2 || ev == 7)) return false;
        if(0 < fr.first && (ev == 0 || ev == 6)) return false;
    } return true;
}
```

9.12 Polygon Cut, Center, Union

```
// Returns the polygon on the left of line l
// *: dot product, ^: cross product
// l = p + d*t, l.q() = l + d
// doubled_signed_area(p,q,r) = (q-p) ^ (r-p)
template<class T> vector<point<T>> polygon_cut(const
vector<point<T>> &a, const line<T> &l){
    vector<point<T>> res;
    for(auto i = 0; i < (int)a.size(); ++ i){
        auto cur = a[i], prev = i ? a[i - 1] : a.back();
        bool side = doubled_signed_area(l.p, l.q(), cur) > 0;
        if(side != (doubled_signed_area(l.p, l.q(), prev) > 0))
            res.push_back(l.p + (cur - l.p ^ prev - cur) / (l.d ^
prev - cur) * l.d);
        if(side) res.push_back(cur);
    }
    return res;
}

P polygonCenter(const vector<P>& v){ // center of mass
P res(0, 0); double A = 0;
for (int i = 0, j = sz(v) - 1; i < sz(v); j = i++) {
    res = res + (v[i] + v[j]) * v[j].cross(v[i]);
    A += v[j].cross(v[i]);
} return res / A / 3;

// 0(points^2), area of union of n polygon, ccw polygon
int sideOf(P s, P e, P p) { return sgn((e-s)/(p-s)); }
int sideOf(const P& s, const P& e, const P& p, double eps) {
    auto a = (e-s)/(p-s); auto l=D1(e-s) * eps;
    return (a > 1) - (a < -1);
}

double rat(P a, P b) { return sgn(b.x) ? a.x/b.x : a.y/b.y; }

double polyUnion(vector<vector<P>>& poly) { // (points)^2
    double ret = 0;
    rep(i,0,sz(poly)) rep(v,0,sz(poly[i])) {
        P A = poly[i][v], B = poly[i][(v + 1) % sz(poly[i])];
        vector<pair<double, int>> segs = {{0, 0}, {1, 0}};
        for(auto &[n, n] : R){
            if( n == 6 ) n ^= state, state ^= 1;
            else if( n == 4 ) n ^= state;
        }
    }
}
```

```
rep(j,0,sz(poly)) if (i != j) { // START
    rep(u,0,sz(poly[j])) {
        P C = poly[j][u], D = poly[j][(u + 1) % sz(poly[j])];
        int sc = sideOf(A, B, C), sd = sideOf(A, B, D);
        if (sc != sd) {
            double sa = C.cross(D, A), sb = C.cross(D, B);
            if (min(sc, sd) < 0)
                segs.emplace_back(sa / (sa - sb), sgn(sc - sd));
        }
    }
    else if (!sc && !sd && j < i && sgn((B-A).dot(D-C)) > 0){
        segs.emplace_back(rat(C - A, B - A, 1));
        segs.emplace_back(rat(D - A, B - A, -1));
    } /*else if*/ /*rep u*/ /*rep j*/ // END
    sort(all(segs));
    for (auto & s : segs) s.first = min(max(s.first, 0.0), 1.0);
    double sum = 0; int cnt = segs[0].second;
    rep(j,1,sz(segs)) {
        if (!cnt) sum += segs[j].first - segs[j - 1].first;
        cnt += segs[j].second;
    }
    ret += A.cross(B) * sum;
} return abs(ret) / 2;
```

9.13 Polycon Raycast

```
// ray A + kd and CCW polygon C, return events {k, event_id}
// 0: out->line / 1: in->line / 2: line->out / 3: line->in
// 4: pass corner outside / 5: pass corner inside / 6: out ->
in / 7: in -> out
// WARNING: C.push_back(C[0]) before use, ccw, no colinear
struct frac{
    ll first, second; frac(){}
    frac(ll a, ll b) : first(a), second(b) {
        if (b < 0) first = -a, second = -b; // operator cast
    }
    double v(){ return 1.*first/second; } // operator <, <=, ==
}; // assert(d != P(0,0));
frac raypoints(const vector<P> &C, P A, P d, vector<pair<frac,
int>> &R){ vector<pair<frac, int>> L;
    auto g = gcd(abs(d.x), abs(d.y)); d.x /= g, d.y /= g;
    for(int i = 0; i+1 < C.size(); i++) P v = C[i+1] - C[i];
    int a = sign(d/(C[i]-A)), b = sign(d/(C[i+1]-A));
    if(a == 0)L.emplace_back(frac(d*(C[i]-A)/size2(d), 1), b);
    if(b == 0)L.emplace_back(frac(d*(C[i+1]-A)/size2(d), 1), a);
    if(a*b == -1)L.emplace_back(frac((A-C[i])/v, v/d), 6);
} sort(L.begin(), L.end());
for(int i = 0; i < L.size(); i++){
    // assert(i+2 >= L.size() || !(L[i].first ==
L[i+2].first));
    if(i+1 < L.size() && L[i].first == L[i+1].first && L[i].second != 6){
        int a = L[i].second, b = L[i+1].second;
        R.emplace_back(L[i+1].first, a*b? a*b > 0?
4:6:(1-a-b)/2);
    } /*end if*/ else R.push_back(L[i]); } /*end for*/
int state = 0; // 0: out, 1: in, 2: line+ccw, 3: line+cw
for(auto &[n, n] : R){
    if( n == 6 ) n ^= state, state ^= 1;
    else if( n == 4 ) n ^= state;
}
```

```

        else if( n == 0 ) n = state, state ^= 2;
        else if( n == 1 ) n = state^(state>>1), state ^= 3;
    } return frac(g, 1);
}

bool visible(const vector<P> &C, P A, P B){
    if( A == B ) return true;//return outside?
    char I[4] = "356", O[4] = "157";
    vector<pair<frac, int>> R; vector<frac> E;
    frac s = frac(0, 1), e = raypoints(C, A, B-A, R);
    for(auto [f,n] : R){
        if(*find(0, 0+3, n+'0')) E.push_back(f);
        if(*find(I, I+3, n+'0')) E.push_back(f);
    }
    for(int j = 0; j < E.size(); j += 2) if( !(e <= E[j] ||
E[j+1] <= s) ) return false;
    return true;
}

9.14  $O(N \log N)$  Shamos-Hoey
Usage: n개의 선분 중 서로 교차하는 선분 쌍이 존재하는지 여부를  $O(n \log n)$ 에 판별

struct Line{
    static ll CUR_X; ll x1, y1, x2, y2, id;
    Line(Point p1, Point p2, int id) : id(id) {
        if(p1 > p2) swap(p1, p2);
        tie(x1,y1) = p1; tie(x2,y2) = p2;
    } Line() = default;
    int get_k() const { return y1 != y2 ? (x2-x1)/(y1-y2) : -1; }
    void convert_k(int k){ // x1,y1,x2,y2 = 0(COORD^2), use i128
        in ccw
        Line res;
        res.x1 = x1 + y1 * k; res.y1 = -x1 * k + y1;
        res.x2 = x2 + y2 * k; res.y2 = -x2 * k + y2;
        x1 = res.x1; y1 = res.y1; x2 = res.x2; y2 = res.y2;
        if(x1 > x2) swap(x1, x2), swap(y1, y2);
    }
    ld get_y(ll offset=0) const { // OVERFLOW
        ld t = ld(CUR_X-x1+offset) / (x2-x1);
        return t * (y2 - y1) + y1; }
    bool operator < (const Line &l) const {
        return get_y() < l.get_y(); }
    // strict
    /* bool operator < (const Line &l) const {
        auto le = get_y(), ri = l.get_y();
        if(abs(le-ri) > 1e-7) return le < ri;
        if(CUR_X==x1 || CUR_X==l.x1) return get_y()<l.get_y();
        else return get_y(-1) < l.get_y(-1);
    }*/
}; ll Line::CUR_X = 0;
struct Event{ // f=0 st, f=1 ed
    ll x, y, i, f; Event() = default;
    Event(Line l, ll i, ll f) : i(i), f(f) {
        if(f==0) tie(x,y) = tie(l.x1,l.y1);
        else tie(x,y) = tie(l.x2,l.y2);
    }
    bool operator < (const Event &e) const {
        return tie(x,f,y) < tie(e.x,e.f,e.y);
    // strict
    // return make_tuple(x,-f,y) < make_tuple(e.x,-e.f,e.y);
    }
}

```

```

};

tuple<bool,int,int> ShamosHoey(vector<array<Point,2>> v){
    int n = v.size(); vector<int> use(n+1);
    vector<Line> lines; vector<Event> E; multiset<Line> T;
    for(int i=0; i<n; i++){
        lines.emplace_back(v[i][0], v[i][1], i);
        if(int t=lines[i].get_k(); 0<=t && t<=n) use[t] = 1;
    }
    int k = find(use.begin(), use.end(), 0) - use.begin();
    for(int i=0; i<n; i++) { lines[i].convert_k(k);
        E.emplace_back(lines[i], i, 0); E.emplace_back(lines[i], i,
1);
    }
    sort(E.begin(), E.end());
    for(auto &e : E){ Line::CUR_X = e.x;
        if(e.f == 0){
            auto it = T.insert(lines[e.i]);
            if(next(it) != T.end() && Intersect(lines[e.i],
*next(it))) return {true, e.i, next(it)->id};
            if(it != T.begin() && Intersect(lines[e.i], *prev(it)))
                return {true, e.i, prev(it)->id};
        }
        else{
            auto it = T.lower_bound(lines[e.i]);
            if(it != T.begin() && next(it) != T.end() &&
Intersect(*prev(it), *next(it)))
                return {true, prev(it)->id, next(it)->id};
            T.erase(it);
        }
    }
    return {false, -1, -1};
}

b9.15  $O(N \log N)$  반평면 교집합
double CCW(p1, p2, p3); bool same(double a, double b); const
Point o = Point(0, 0);
struct Line{ // ax+by leq c
    double a, b, c; Line() : Line(0, 0, 0) {}
    Line(double a, double b, double c) : a(a), b(b), c(c) {}
    bool operator < (const Line &l) const {
        bool f1 = Point(a, b) > o, f2 = Point(l.a, l.b) > o;
        if(f1 != f2) return f1 > f2;
        double cw = CCW(o, Point(a, b), Point(l.a, l.b));
        return same(cw, 0) ? c * hypot(l.a, l.b) < l.c * hypot(a,
b) : cw > 0;
    }
    Point slope() const { return Point(a, b); }
}; Point LineIntersect(Line a, Line b){
    double det = a.a*b.b - b.a*a.b, x = (a.c*b.b - a.b*b.c) /
det, y = (a.a*b.c - a.c*b.a) / det; return Point(x, y);
}
bool CheckHPI(Line a, Line b, Line c){
    if(CCW(o, a.slope(), b.slope()) <= 0) return 0;
    Point v=LineIntersect(a,b); return v.x*c.a+v.y*c.b>=c.c;
}
vector<Point> HPI(vector<Line> v){
    sort(v.begin(), v.end()); deque<Line> dq; vector<Point> ret;
    for(auto &i : v){
        if(dq.size() && same(CCW(o, dq.back().slope(), i.slope()),
0)) continue;
    }
}
```

```

while(dq.size() >= 2 && CheckHPI(dq[dq.size()-2], dq.back(), i)) dq.pop_back();
while(dq.size() >= 2 && CheckHPI(i,dq[0],dq[1])) dq.pop_front();
dq.push_back(i);
}
while(dq.size() > 2 && CheckHPI(dq[dq.size()-2], dq.back(), dq[0])) dq.pop_back();
while(dq.size() > 2 && CheckHPI(dq.back(), dq[0], dq[1])) dq.pop_front();
for(int i=0; i<dq.size(); i++){
    Line now = dq[i], nxt = dq[(i+1)%dq.size()];
    if(CCW(o, now.slope(), nxt.slope()) <= eps) return
vector<Point>();
    ret.push_back(LineIntersect(now, nxt));
}
//for(auto &[x,y] : ret) x = -x, y = -y;
return ret;
} // MakeLine: left side of ray (x1,y1) -> (x2,y2)
Line MakeLine(T x1, T y1, T x2, T y2){
    T a = y2-y1, b = x1-x2, c = x1*a + y1*b; return {a,b,c};
}

b9.16  $O(N^2 \log N)$  Bulldozer Trick(Rotating Sweep Line Technique)
Usage: 서로 다른 정렬의 결과가  $O(N^2)$  가지임을 보이고, 가능한 모든 정렬 결과를  $O(N^2 \log N)$ 에 순회하는 테크닉

struct Line{
    ll i, j, dx, dy; // dx >= 0
    Line(int i, int j, const Point &pi, const Point &pj)
        : i(i), j(j), dx(pj.x-pi.x), dy(pj.y-pi.y) {}
    bool operator < (const Line &l) const {
        return make_tuple(dy*l.dx, i, j) < make_tuple(l.dy*dx, l.i,
l.j); }
    bool operator == (const Line &l) const {
        return dy * l.dx == l.dy * dx; }
}
void Solve(){ // V.reserve(N*(N-1)/2)
    sort(A+1, A+N+1); iota(P+1, P+N+1, 1); vector<Line> V;
    for(int i=1; i<=N; i++) for(int j=i+1; j<=N; j++)
        V.emplace_back(i, j, A[i], A[j]);
    sort(V.begin(), V.end());
    for(int i=0, j=0; i<V.size(); i=j){
        while(j < V.size() && V[i] == V[j]) j++;
        for(int k=i; k<j; k++){
            int u = V[k].i, v = V[k].j; // point id, index -> Pos[id]
            swap(Pos[u], Pos[v]); swap(A[Pos[u]], A[Pos[v]]);
            if(Pos[u] > Pos[v]) swap(u, v);
            // @TODO
        }
    }
}

b9.17  $O(N)$  최소 외접원
pt getCenter(pt a, pt b){ return pt((a.x+b.x)/2, (a.y+b.y)/2);
}
pt getCenter(pt a, pt b, pt c){
    pt aa = b - a, bb = c - a;
    auto c1 = aa*aa * 0.5, c2 = bb*bb * 0.5, d = aa / bb;
    auto x = a.x + (c1 * bb.y - c2 * aa.y) / d;
    auto y = a.y + (c2 * aa.x - c1 * bb.x) / d;
}
```

```

    return pt(x, y); }
Circle solve(vector<pt> v){
pt p = {0, 0};
double r = 0; int n = v.size();
for(int i=0; i<n; i++) if(dst(p, v[i]) > r + EPS){
    p = v[i]; r = 0;
    for(int j=0; j<i; j++) if(dst(p, v[j]) > r + EPS){
        p = getCenter(v[i], v[j]); r = dst(p, v[i]);
        for(int k=0; k<j; k++) if(dst(p, v[k]) > r + EPS){
            p = getCenter(v[i], v[j], v[k]); r = dst(v[k], p);
        }
    }
    return {p, r};
}

```

10 Misc

10.1 삼분 탐색

```

while(s + 3 <= e){
    T l = (s + s + e) / 3, r = (s + e + e) / 3;
    if(Check(l) > Check(r)) s = l; else e = r;
} // get minimum / when multiple answer, find minimum `s`
T mn = INF, idx = s;
for(T i=s; i<=e; i++) if(T now = Check(i); now < mn) mn = now,
idx = i;

```

10.2 Mo's Algorithm

Time Complexity: $O((N+Q)\sqrt{N})$

```

#include <bits/stdc++.h>
#define fastio cin.tie(0)->sync_with_stdio(0)
using namespace std;
using ll = long long;
int sqrtN;

struct Query{
    int idx, s, e;
    bool operator < (Query &x){
        if(s/sqrtN != x.s/sqrtN) return s/sqrtN < x.s/sqrtN;
        return e < x.e;
    }
};

int main(){
    fastio;
    int N, M; cin >> N >> M;
    sqrtN = sqrt(N);

    int A[N+1];
    for(int i=1; i<=N; i++) cin >> A[i];

    Query query[M];
    for(int i=0; i<M; i++){
        int s, e; cin >> s >> e;
        query[i] = Query({i, s, e});
    }
    sort(query, query+M);

    // 구간 [s, e]의 합을 구하는 예제
    ll sum = 0, ans[M];
    int s = query[0].s, e = query[0].e;
    for(int i=s; i<=e; i++) sum += A[i];
    ans[query[0].idx] = sum;
}

```

```

for(int i=1; i<M; i++){
    while(s < query[i].s) sum -= A[s++];
    while(s > query[i].s) sum += A[--s];
    while(e < query[i].e) sum += A[++e];
    while(e > query[i].e) sum -= A[e--];
    ans[query[i].idx] = sum;
}
for(int i=0; i<M; i++) cout << ans[i] << '\n';
}

```

10.3 C++ 랜덤, GCC 확장, 비트마스킹 트리

```

mt19937
rd((unsigned)chrono::steady_clock::now().time_since_epoch().count());
uniform_int_distribution<int> rnd_int(1, r); // rnd_int(rd)
uniform_real_distribution<double> rnd_real(0, 1); // rnd_real(rd)
/////
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
#include <ext/rope>
using namespace __gnu_pbds; //ordered_set :
find_by_order(order), order_of_key(key)
using namespace __gnu_cxx; //crope : append(str), substr(s, e),
at(idx)
template <typename T>
using ordered_set = tree<T, null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
////
int __builtin_clz(int x); // number of leading zero
int __builtin_ctz(int x); // number of trailing zero
int __builtin_popcount(int x); // number of 1-bits in x
lsb(n): (n & -n); // last bit (smallest)
floor(log2(n)): 31 - __builtin_clz(n | 1);
floor(log2(n)): 63 - __builtin_ctzll(n | 1);
long long next_perm(long long v){
    long long t = v | (v-1);
    return (t + 1) | (((~t & ~t) - 1) >> (__builtin_ctz(v) + 1));
}
int freq(int n, int i) { // # of digit i in [1, n]
    int j, r = 0;
    for (j = 1; j <= n; j *= 10) if (n / j / 10 >= !i) r += (n / 10 / j - !i) * j + (n / j % 10 > i ? j : n / j % 10 == i ? n % j + 1 : 0);
    return r;
}

```

10.4 Floating Point Add (Kahan)

```

template<typename T> T float_sum(vector<T> v){
    T sum=0, c=0, y, t; // sort abs(v[i]) increase?
    for(T i:v) y=i-c, t=sum+y, c=(t-sum)-y, sum=t;
    return sum; //worst O(eps*N), avg O(eps*sqrtN)
} //dnc: worst O(eps*logN), avg O(eps*sqrtlogN)

```

10.5 Fast I/O, Fast Div, Fast Mod

```

namespace io { // thanks to cgiosy
    const signed IS=1<<20; char I[IS+1], *J=I;
    inline void daer(){if(J>=I+IS-64){}
}

```

```

char*p=I;do*p++=*J++;
while(J!=I+IS);p[read(0,p,I+IS-p)]=0;J=I;};
template<int N=10,typename T=int>inline T getu(){
    daer();T x=0;int k=0;do x=x*10+*J-'0';
    while(*++J>='0'&&k<N);++J;return x;};
template<int N=10,typename T=int>inline T geti(){
    daer();bool e==*J=='-';J+=e;return (e?-1:1)*getu<N,T>();};
struct f{f(){I[read(0,I,IS)]=0;}}flu; };
struct FastMod{ // typedef __uint128_t L;
ull b, m; FastMod(ull b) : b(b), m(ull((L(1) << 64) / b)) {}}
ull reduce(ull a){ // can be proven that 0 <= r < 2*b
    ull q = (ull)((L(m) * a) >> 64), r = a - q * b;
    return r >= b ? r - b : r;
}
pair<uint32_t, uint32_t> Div(uint64_t a, uint32_t b){
    if(__builtin_constant_p(b)) return {a/b, a%b};
    uint32_t lo=a, hi=a>>32;
    __asm__("div %2 : "+a,a" (lo), "+d,d" (hi) : "r,m" (b));
    return {lo, hi}; // BOJ 27505, q r < 2^32
} // divide 10M times in ~400ms
ull mulmod(ull a, ull b, ull M){ // ~2x faster than int128
    ll ret = a * b - M * ull(1.L / M * a * b);
    return ret + M * (ret < 0) - M * (ret >= (11.M));
} // safe for 0 ≤ a,b < M < (1<<63) when long double is 80bit

```

10.6 Python Decimal

```

from fractions import Fraction
from decimal import Decimal, getcontext
getcontext().prec = 250 # set precision
N, two, itwo = 200, Decimal(2), Decimal(0.5)
# sin(x) = sum (-1)^n x^(2n+1) / (2n+1)!
# cos(x) = sum (-1)^n x^(2n) / (2n)!
def angle(cosT):
    #given cos(theta) in decimal return theta
    for i in range(N): cosT=((cosT+1)/two)**itwo
    sinT = (1-cosT*cosT)**itwo
    return sinT*(2**N)
pi = angle(Decimal(-1))

```

11 Notes

11.1 구간별 약수 최대 개수, 최대 소수

< 10^k	number	divisors	2 3 5 7 11 13 17 19 23 29 31 37
1	6	4	1 1
2	60	12	2 1 1
3	840	32	3 1 1 1
4	7560	64	3 3 1 1
5	83160	128	3 3 1 1 1
6	720720	240	4 2 1 1 1 1
7	8648640	448	6 3 1 1 1 1
8	73513440	768	5 3 1 1 1 1 1
9	735134400	1344	6 3 2 1 1 1 1
10	6983776800	2304	5 3 2 1 1 1 1 1 1
11	97772875200	4032	6 3 2 2 1 1 1 1 1
12	963761198400	6720	6 4 2 1 1 1 1 1 1
13	9316358251200	10752	6 3 2 1 1 1 1 1 1
14	97821761637600	17280	5 4 2 2 1 1 1 1 1 1
15	866421317361600	26880	6 4 2 1 1 1 1 1 1 1 1

16	8086598962041600	41472	8 3 2 2 1 1 1 1 1 1 1 1
17	74801040398884800	64512	6 3 2 2 1 1 1 1 1 1 1 1
18	897612484786617600	103680	8 4 2 2 1 1 1 1 1 1 1 1

< 10^k	prime	# of prime	< 10^k	prime
1	7	4	10	9999999967
2	97	25	11	99999999977
3	997	168	12	99999999989
4	9973	1229	13	999999999971
5	99991	9592	14	9999999999973
6	999983	78498	15	99999999999989
7	9999991	664579	16	999999999999937
8	99999989	5761455	17	999999999999997
9	99999937	50847534	18	9999999999999989

11.2 비트 연산

Some properties of bitwise operations:

- $a|b = a \oplus b + a \& b$
- $a \oplus (a \& b) = (a|b) \oplus b \Leftrightarrow b \oplus a \oplus (a \& b) = (a|b) \Leftrightarrow b \oplus (a \& b) = (a|b) \oplus a$
- $(a \& b) \oplus (a|b) = a \oplus b$

Addition:

- $a + b = a|b + a \& b$
- $a + b = a \oplus b + 2(a \& b)$

Subtraction:

- $a - b = (a \oplus (a \& b)) - ((a|b) \oplus a)$
- $a - b = ((a|b) \oplus b) - ((a|b) \oplus a)$
- $a - b = (a \oplus (a \& b)) - (b \oplus (a \& b))$
- $a - b = ((a|b) \oplus b) - (b \oplus (a \& b))$

application

- 모든 비트 켜기, 오른쪽 i개 비트 켜기 : 0, ($1 << i$) - 1
- 2의 거듭제곱으로 나머지, 곱셈, 나눗셈 연산 : $n \& ((1 << i) - 1)$, $n << i$, $n >> i$
- $-n = \sim(n - 1)$: n - 1은 LSB의 오른쪽 비트 flip, -n은 LSB의 왼쪽 비트 flip
- n의 LSB(가장 오른쪽의 1인 비트) 구하기 : $n \& -n$
- $n|2^i$ 의 거듭제곱인지 판별 : $(n \& n - 1) == 0$ 또는 $(n \& -n) == n$
- n의 가장 오른쪽 1 끄기 : $n \& (n - 1)$
- n의 가장 오른쪽 0 켜기 : $n | (n + 1)$
- n의 오른쪽 끝의 연속되는 1을 모두 끄기 : $n \& (n + 1)$
- n의 오른쪽 끝의 연속되는 0을 모두 켜기 : $n | (n - 1)$
- iterating subset in decreasing order : $i = (i - 1) \& mask$

11.3 Triangles/Trigonometry

- k차원 구 부피: $V_{2k} = \pi^k / k!$, $v_{2k+1} = 2^{k+1} \pi^k / (2k + 1)!!$
- $\sin(\alpha \pm \beta) = \sin \alpha \cos \beta \pm \cos \alpha \sin \beta$, $\sin 2\theta = 2 \sin \theta \cos \theta$
- $\cos(\alpha \pm \beta) = \cos \alpha \cos \beta \mp \sin \alpha \sin \beta$, $\cos 2\theta = 1 - 2 \sin^2 \theta$
- $\tan(\alpha \pm \beta) = \frac{\tan \alpha \pm \tan \beta}{1 \mp \tan \alpha \tan \beta}$, $\tan 2\theta = \frac{2 \tan \theta}{1 - \tan^2 \theta}$
- 반각 공식: $\sin^2 \theta/2 = \frac{1 - \cos \theta}{2}$, $\cos^2 \theta/2 = \frac{1 + \cos \theta}{2}$, $\tan^2 \theta/2 = \frac{1 - \cos \theta}{1 + \cos \theta}$
- 변 길이 $a, b, c; p = (a + b + c)/2$
- 넓이 $A = \sqrt{p(p - a)(p - b)(p - c)}$
- 외접원 반지름 $R = abc/4A$, 내접원 반지름 $r = A/p$
- 중선 길이 $m_a = 0.5 \sqrt{2b^2 + 2c^2 - a^2}$, 각 이등분선 $s_a = \sqrt{bc(1 - \frac{a}{b+c})^2}$
- 사인 법칙 $\frac{\sin A}{a} = 1/2R$, 코사인 법칙 $a^2 = b^2 + c^2 - 2bc \cos A$

- 탄젠트 법칙 $\frac{a+b}{a-b} = \frac{\tan(A+B)/2}{\tan(A-B)/2}$
- 중심 좌표 $(\frac{\alpha x_a + \beta x_b + \gamma x_c}{\alpha + \beta + \gamma}, \frac{\alpha y_a + \beta y_b + \gamma y_c}{\alpha + \beta + \gamma})$

이름	α	β	γ	
외심	$a^2 \mathcal{A}$	$b^2 \mathcal{B}$	$c^2 \mathcal{C}$	$\mathcal{A} = b^2 + c^2 - a^2$
내심	a	b	c	$\mathcal{B} = a^2 + c^2 - b^2$
무게중심	1	1	1	$\mathcal{C} = a^2 + b^2 - c^2$
수심	\mathcal{BC}	\mathcal{CA}	\mathcal{AB}	
방심(A)	$-a$	b	c	

11.4 Calculus, Newton's Method

- 음합수 미분: $f(x, y) = 0$ 의 양변을 x 에 대해 미분한 뒤 dy/dx 에 대해 정리
- (예시) $\frac{d}{dx}(x^3) + \frac{d}{dx}(y^3) - 3 \frac{d}{dx}(xy) = 3x^2 + 3y^2 \frac{dy}{dx} - 3(y + x \frac{dy}{dx}) = 0$
- 역함수 미분: $(f^{-1})'(x) = 1/f'(f^{-1}(x))$
- 뉴턴 래슨: $x_{n+1} = x_n - f(x_n)/f'(x_n)$
- 치환 적분: $x = g(t)$ 로 두면, $\int f(x)dx = \int f(g(t))g'(t)dt$
- (예시) $\int \frac{f'(x)}{f(x)} dx$ 에서 $t = f(x)$ 라고 두면 $f'(x) = dt/dx$ 따라서 $\int \frac{f'(x)}{f(x)} dx = \int 1/t dt = \ln|t| + C = \ln|f(x)| + C$
- 삼각 치환: $\sqrt{a^2 - x^2}$ 에서는 $x = a \sin t$, $\sqrt{a^2 + x^2}$ 에서는 $x = a \tan t$, 범위 조심
- 입체 도형 부피: $x = a, x = b$ 사이에서 단면의 넓이 함수 $A(x)$ 가 연속이면 부피는 $\int_a^b A(x)dx$
- (원주각법): 연속함수 $f(x)$ 가 $[a, b]$ 에서 $f(x) \geq 0$ 일 때, $f(x)$ 와 두 직선 $x = a, x = b$, 그리고 x축으로 둘러싸인 영역을 y축으로 회전시킨 부피는 $\int_a^b 2\pi x f(x) dx$
- 곡선 길이: $f(t)$ 가 $[a, b]$ 에서 연속이면, $x = a, x = b$ 사이의 곡선 길이는 $\int_a^b \sqrt{1 + [f'(x)]^2} dx$
- 회전 곡면 넓이: x축으로 회전시킨 부피는 $\int_a^b 2\pi f(x) \sqrt{1 + [f'(x)]^2} dx$
- $\oint_C (Ldx + Mdy) = \int \int_D (\frac{\partial M}{\partial x} - \frac{\partial L}{\partial y}) dx dy$
- where C is positively oriented, piecewise smooth, simple, closed; D is the region inside C ; L and M have continuous partial derivatives in D .

$f(x)$	$f'(x)$	$\int f(x)dx$
$\sin x$	$\cos x$	$-\cos x$
$\cos x$	$-\sin x$	$\sin x$
$\tan x$	$\sec^2 x = 1 + \tan^2 x$	$-\ln \cos x $
$\csc x$	$-\csc x \cot x$	$\ln \tan(x/2) $
$\sec x$	$\sec x \tan x$	$\ln \tan(x/2 + \pi/4) $
$\cot x$	$-\csc^2 x$	$\ln \sin x $
$\arcsin x$	$1/\sqrt{1-x^2}$	$x \arcsin x + \sqrt{1-x^2}$
$\arccos x$	$-1/\sqrt{1-x^2}$	$x \arccos x - \sqrt{1-x^2}$
$\arctan x$	$1/(1+x^2)$	$x \arctan x - \frac{\ln(x^2+1)}{2}$
$\csc^{-1} x$	$-1/x\sqrt{x^2-1}$	$x \csc^{-1} x + \cosh^{-1} x $
$\sec^{-1} x$	$1/x\sqrt{x^2-1}$	$x \sec^{-1} x - \cosh^{-1} x $
$\cot^{-1} x$	$-1/(1+x^2)$	$x \cot^{-1} x + \frac{\ln(x^2+1)}{2}$

11.5 Zeta/Mobius Transform

- Subset Zeta/Mobius Transform($n=sz=2^k, i^*=2$)
 - for $i=1..n-1$ for $j=0..n-1$ if(i and j) $v[j] \pm= v[i \text{ xor } j]$
- Superset Zeta/Mobius Transform($n=sz=2^k, i^*=2$)
 - for $i=1..n-1$ for $j=0..n-1$ if(i and j) $v[i \text{ xor } j] \pm= v[j]$
- Divisor Zeta/Mobius Transform($n=sz-1$)
 - for p:Prime for $i=1..n/p$ $v[i^*p] += v[i]$
 - for p:Prime for $i=n/p..1$ $v[i^*p] -= v[i]$

- Multiple Zeta/Mobius Transform($n=sz-1$)
 - for p:Prime for $i=n/p..1$ $v[i] += v[i^*p]$
 - for p:Prime for $i=1..n/p$ $v[i] -= v[i^*p]$
- AND Convolution: SupZeta(A), SupZeta(B), SupMobius(mul)
- OR Conv.: Subset, GCD Conv.: Multiple, LCM Conv.: Divisor
- AND/OR: $2^0 0.3s$, Subset: $2^{20} 2.5s$, GCD/LCM: 1e6 0.3s

11.6 Generating Function

- 등차수열: $(pn + q)x^n = p/(1-x) + q/(1-x)^2$
- 등비수열: $(rx)^n = (1-rx)^{-1}$
- 조합: $C(m, n)x^n = (1+x)^m$
- 중복조합: $C(m+n-1, n)x^n = (1-x)^{-m}$
- $f(n) = \sum_{k=0}^n k! \times S_2(n, k)$ 의 EGF: $1/(2-e^x)$
- Ordinary Generating Function $A(x) = \sum_{i \geq 0} a_i x^i$
 - $A(rx) \Rightarrow r^n a_n, xA(x)' \Rightarrow na_n$
 - $A(x) + B(x) \Rightarrow a_n + b_n, A(x)B(x) \Rightarrow \sum_{i=0}^n a_i b_{n-i}$
 - $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} a_{i_1} a_{i_2} \dots a_{i_k}$
 - $\frac{A(x)}{1-x} \Rightarrow \sum_{i=0}^n a_i$
- Exponential Generating Function $A(x) = \sum_{i \geq 0} \frac{a_i}{i!} x^i$
 - $A(x) + B(x) \Rightarrow a_n + b_n, A(x)B(x) \Rightarrow \sum_{i=0}^n \binom{n}{i} a_i b_{n-i}$
 - $A^{(k)}(x) \Rightarrow a_{n+k}, xA(x) \Rightarrow na_n$
 - $A(x)^k \Rightarrow \sum_{i_1+i_2+\dots+i_k=n} \binom{n}{i_1, i_2, \dots, i_k} a_{i_1} a_{i_2} \dots a_{i_k}$

11.7 Counting

조건 없음	단사 함수	전사 함수
k^n	$k!/(k-n)!$	$k! \times S_2(n, k)$
$C(n+k-1, n)$	$C(k, n)$	$C(n-1, n-k)$ 조심
$B(n, k)$	$[n \leq k]$	$S_2(n, k)$
$P_k(n+k)$	$[n \leq k]$	$P_k(n)$

- 단사 함수: 상자에 공 최대 1개, 전사 함수: 상자에 공 최소 1개
- 중복 조합: $C(n+k-1, n) = H(n, k)$
- 공 구별 X, 전사 함수에서 $n = k = 0$ 일 때 정답 1인 거 조심
- 교란 순열: 모든 i에 대해 $\pi(i) \neq i$ 성립하는 길이 n 순열 개수
 - 초항(0-based): 1, 0, 1, 2, 9, 44, 265, 1854
 - 일반항: $D(n) = \sum_{k=0}^n (-1)^k n!/k!$, $D(n) \approx n!/e$
 - 점화식: $D(0) = 1; D(1) = 0; D(n) = (n-1)(D(n-1) + D(n-2))$
 - 생성함수(EGF): $e^{-x}/(1-x)$
- 카탈란 수
 - 길이가 2n인 올바른 괄호 수식의 수
 - $n+1$ 개의 리프를 가진 풀 바이너리 트리의 수
 - $n+2$ 각형을 n개의 삼각형으로 나누는 방법의 수 초항(0-based): 1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786
 - 일반항: $Cat(n) = Comb(2n, n)/(n+1) = C(2n, n) - C(2n, n+1)$
 - 점화식: $Cat(0) = 1; Cat(n) = \sum_{k=0}^{n-1} Cat(k) \times Cat(n-k-1)$
 - 생성함수(OGF): $(1 - \sqrt{1-4x})/(2x)$
 - 여는 괄호 n개, 닫는 괄호 $k \leq n$ 개는 $C(n+k, k) \times (n-k+1)/(n+1)$
- 제 1종 스텔링 수: k개의 사이클로 구성된 길이 n 순열 개수
 - 점화식: $S_1(n, 0) = [n = 0]; S_1(n, k) = S_1(n-1, k-1) + S_1(n-1, k) \times (n-1)$, 생성함수(EGF): $(-\ln(1-x))^k/k!$
 - 성질: $\sum_{k=0}^n S_1(n, k) = n!$
- 제 2종 스텔링 수: n개의 원소를 k개의 공집합이 아닌 부분집합으로 분할
 - 일반항: $S_2(n, k) = 1/k! \times \sum_{i=1}^k (-1)^{k-i} \times C(k, i) \times i^n$, 단 $S(0, 0) = 1$
 - 점화식: $S_2(n, 0) = [n = 0]; S_2(n, k) = S_2(n-1, k-1) + S_2(n-1, k) \times k$
 - 생성함수(EGF): $(\exp(x)-1)^k/k!$
 - 성질: A, B를 $n-k$, $\lfloor (k-1)/2 \rfloor$ 의 켜진 비트 위치라고 하면, $S_2(n, k) \bmod 2 = [A \cap B = \emptyset]$
 - 성질: $S_2(2n, n)$ 은 n^o fibbinary number(연속한 1 없음) 일 때만 허수

• 벨 수 $B(n)$: n 개의 원소를 분할하는 방법의 수

초항(0-based): 1, 1, 2, 5, 15, 52, 203, 877, 4140, 21147, 115975
 일반항: $B(n) = \sum_{k=0}^n S_2(n, k)$, 몇 개의 상자를 벼릴지 다 돌아보기
 점화식: $B(0) = 1$; $B(n) = \sum_{k=0}^{n-1} C(n-1, k) \times B(k)$
 생성함수(EGF): $\exp(\exp(x) - 1)$

• 벨 수 $B(n, k)$: n 개를 집합 k 개로 분할하는 방법의 수(공집합 허용)

일반항: $B(n, k) = \sum_{i=0}^k S_2(n, i) = \sum_{i=0}^k \frac{i^n}{i!} \sum_{j=0}^{k-i} \frac{(-1)^j}{j!}$

• 분할 수 $P(n)$: n 을 자연수 몇 개의 합으로 나타내는 방법의 수, 순서 X

초항(0-based): 1, 1, 2, 3, 5, 7, 11, 15, 22, 30, 42

점화식: $P(0) = 1$, 팀노트 어딘가에 있는 코드로 $O(n\sqrt{n})$ 가능

• 분할 수 $P(n, k)$: n 을 k 개의 자연수의 합으로 나타내는 방법의 수, 순서 X

점화식: $P(0, 0) = 1$; $P(n, 0) = 0$; $P(n, k) = P(n-1, k-1) + P(n-k, k)$

성질: $\sum_{k=0}^n P(n, k) = P(n)$

• 피보나치 수 $F_0 = 0, F_1 = 1, F_n = F_{n-1} + F_{n-2}$

일반항: $\frac{F_n = (\phi^n - \bar{\phi}^n)}{\phi^n - \bar{\phi}^n}, \phi = \frac{1 + \sqrt{5}}{2}, \bar{\phi} = \frac{1 - \sqrt{5}}{2}$

생성함수(OGF): $F(x) = x/(1-x-x^2)$

성질: $F_1 + \dots + F_n = F_{n+2} - 1, F_1^2 + \dots + F_n^2 = F_n F_{n+1}$

성질: $\gcd(F_n, F_m) = F_{\gcd(n, m)}$, $F_n^2 - F_{n+1} F_{n-1} = (-1)^{n-1}$

• 벨 수 $B(n, k)$ 식 전개

$$\begin{aligned} B(n, k) &= \sum_{j=0}^k S(n, j) = \sum_{j=0}^k 1/j! \sum_{i=0}^j (-1)^{j-i} i Ci \times i^n \\ &= \sum_{j=0}^k \sum_{i=0}^j \frac{(-1)^{j-i}}{i!(j-i)!} i^n = \sum_{i=0}^k \sum_{j=i}^k \frac{(-1)^{j-i}}{i!(j-i)!} i^n \\ &= \sum_{i=0}^k \sum_{j=0}^{k-i} \frac{(-1)^j}{i!j!} i^n = \sum_{i=0}^k \frac{i^n}{i!} \sum_{j=0}^{k-i} \frac{(-1)^j}{j!} \end{aligned}$$

11.8 Faulhaber's Formula ($\sum_{k=1}^n k^c$)

• B_n : 베르누이 수

• 생성함수(egf): $\frac{x}{e^x - 1} = \frac{1}{(e^x - 1)/x} = \sum_{n=0}^{\infty} \frac{B_n}{n!} x^n$

• 일반항: $B_n = \sum_{k=0}^n \frac{k!(-1)^k}{k+1} S_2(n, k)$

• 점화식: $B_0 = 1$; $B_n = -\frac{1}{n+1} \sum_{r=0}^{n-1} \binom{n+1}{r} B_r$

• $\sum_{k=1}^n k^c = \sum_{k=0}^c \binom{-1}{k} \binom{c+1}{k} B_k n^{c+1-k}$

11.9 About Graph Degree Sequence

• 단순 무향 그래프(Erdos-Gallai): 차수열 $d_1 \geq \dots \geq d_n$ 의 합이 짝수 and 모든 $1 \leq k \leq n$ 에 대해 $\sum_{i=1}^k d_i \leq k(k-1) + \sum_{i=k+1}^n \min(d_i, k)$

• 단순 이분 그래프(Gale-Ryser): 차수열 $a_1 \geq \dots \geq a_n$, b_i 에서 $\text{sum}(a) = \text{sum}(b)$ and 모든 $1 \leq k \leq n$ 에 대해 $\sum_{i=1}^k a_i \leq \sum_{i=1}^n \min(b_i, k)$

• 단순 방향 그래프(Fulkerson-Chen-Anstee): $a_1 \geq \dots \geq a_n$ 를 만족하는 진입/진출 차수열 $(a_1, b_1), \dots, (a_n, b_n)$ 에서 $\text{sum}(a) = \text{sum}(b)$ and 모든 $1 \leq k \leq n$ 에 대해 $\sum_{i=1}^k a_i \leq \sum_{i=1}^k \min(b_i, k-1) + \sum_{i=k+1}^n \min(b_i, k)$

11.10 Burnside, Grundy, Pick, Hall, Simpson, Area of Quadrangle, Fermat Point, Euler, Pythagorean

• Burnside's Lemma

- 수식

$G = (X, A)$: 집합 X 와 액션 A 로 정의되는 군 G 에 대해, $|A||X/A| = \text{sum}(|\text{Fixed points of } a|, \text{for all } a \text{ in } A)$

X/A 는 Action으로 서로 변형 가능한 X 의 원소들을 동치로 묶었을 때 동치류(파티션) 집합

- 풀어쓰기

orbit: 그룹에 대해 두 원소 a, b 와 액션 f 에 대해 $f(a) = b$ 인거에 간선 연결한 컴포넌트(연결집합)

orbit 개수 = sum(각 액션 g 에 대해 $f(x)=x$ 인 x (고정점) 개수) / 액션 개수
 - 자유도 치트시트

회전 n 개: 회전 i 의 고정점 자유도 = $\gcd(n, i)$

임의의 뒤집기 $n =$ 홀수: n 개 원소 중심축(자유도 $(n+1)/2$)

임의의 뒤집기 $n =$ 짝수: $n/2$ 개 원소 중심축(자유도 $n/2+1$) + $n/2$ 개 원소 안지나는 축(자유도 $n/2$)

• 알고리즘 계업

- Nim Game의 해법(마지막에 가져가는 사람이 승): XOR = 0이면 후공승, 0 아니면 선공승

- Subtraction Game: 한 번에 k 개까지의 돌만 가져갈 수 있는 경우, 각 더미의 돌의 개수를 $k+1$ 로 나눈 나머지를 XOR 합하여 판단한다.

- Index-k Nim: 한 번에 최대 k 개의 더미를 골라 각각의 더미에서 아무렇게나 돌을 제거할 수 있을 때, 각 binary digit에 대하여 합을 $k+1$ 로 나눈 나머지를 계산한다. 만약 이 나머지가 모든 digit에 대하여 0이라면 두 번째, 하나라도 0이 아니라면 첫 번째 플레이어가 승리.

- Misere Nim: 모든 돌 무더기가 1이면 N이 홀수일 때 후공승, 그렇지 않은 경우 XOR 합 0이면 후공승

• Pick's Theorem

격자점으로 구성된 simple polygon이 주어짐. I는 polygon 내부의 격자점 수, B는 polygon 선분 위 격자점 수, A는 polygon의 넓이라고 할 때, 다음과 같은 식이 성립한다. $A = I + B/2 - 1$

• 훌의 결론 정리: 이분그래프(L-R)에서, 모든 L을 매칭하는 필요충분 조건 = L에서 임의의 부분집합 S를 골랐을 때, 반드시 $(S \text{의 크기}) \leq (S \text{와 연결되어있는 모든 } R \text{의 크기})$ 이다.

• Simpson 공식 (적분): Simpson 공식, $S_n(f) = \frac{h}{3}[f(x_0) + f(x_n) + 4\sum f(x_{i+1}) + 2\sum f(x_{2i})]$

- $M = \max |f''(x)|$ 라고 하면 오차 범위는 최대 $E_n \leq \frac{M(b-a)}{180} h^4$

• 브라마굽타: 원에 내접하는 사각형의 각 선분의 길이가 a, b, c, d 일 때

사각형의 넓이 $S = \sqrt{(s-a)(s-b)(s-c)(s-d)}$, $s = (a+b+c+d)/2$

• 브레치나이더: 임의의 사각형의 각 변의 길이를 a, b, c, d 라고 하고, 마주보는 두 각의 합을 2로 나눈 값을 θ 라 하면, $S = \sqrt{(s-a)(s-b)(s-c)(s-d) - abcd \times \cos^2 \theta}$

• 페르마 포인트: 삼각형의 세 꼭짓점으로부터 거리의 합이 최소가 되는 점 $2\pi/3$ 보다 큰 각이 있으면 그 점이 페르마 포인트, 그렇지 않으면 각변마다 정삼각형 그린 다음, 정삼각형의 끝점에서 반대쪽 삼각형의 꼭짓점으로 연결한 선분의 교점

$2\pi/3$ 보다 큰 각이 없으면 거리의 합은 $\sqrt{(a^2 + b^2 + c^2 + 4\sqrt{3}S)/2}$, S 는 넓이

• 오일러 정리: 서로소인 두 정수 a, n 에 대해 $a^{\phi(n)} \equiv 1 \pmod{n}$

모든 정수에 대해 $a^n \equiv a^{n-\phi(n)} \pmod{n}$

$m \geq \log_2 n$ 이면 $a^m \equiv a^{m \% \phi(n) + \phi(n)} \pmod{n}$

• $g^0 + g^1 + g^2 + \dots + g^{p-2} \equiv -1 \pmod{p}$ iff $g = 1$, otherwise 0.

• if $n \equiv 0 \pmod{2}$, then $1^n + 2^n + \dots + (n-1)^n \equiv 0 \pmod{n}$

• Eulerian numbers

Number of permutations $\pi \in S_n$ in which exactly k elements are greater than the previous element. k : s.t. $\pi(j) > \pi(j+1)$, $k+1$: s.t. $\pi(j) \geq j$, k : s.t. $\pi(j) > j$.

$E(n, k) = (n-k)E(n-1, k-1) + (k+1)E(n-1, k)$

$E(n, 0) = E(n, n-1) = 1$

$E(n, k) = \sum_{j=0}^k (-1)^j \binom{n+1}{j} (k+1-j)^n$

• Pythagorean triple: $a^2 + b^2 = c^2$ 이고 서로소인 (a, b, c) 생성

$(a, b, c) = (st, \frac{s^2-t^2}{2}, \frac{s^2+t^2}{2})$, $\gcd(s, t) = 1, s > t$

11.11 About Graph Minimum Cut

• N 개의 boolean 변수 v_1, \dots, v_n 을 정해서 비용을 최소화하는 문제 = true인 점은 T , false인 점은 F 와 연결되게 분할하는 민컷 문제

1. v_i 가 T일 때 비용 발생: i 에서 F로 가는 비용 간선

2. v_i 가 F일 때 비용 발생: i 에서 T로 가는 비용 간선

3. v_i 가 T이고 v_j 가 F일 때 비용 발생: i 에서 j 로 가는 비용 간선

4. $v_i \neq v_j$ 일 때 비용 발생: i 에서 j 로, j 에서 i 로 가는 비용 간선

5. v_i 가 T면 v_j 도 T여야 함: i 에서 j 로 가는 무한 간선

6. v_i 가 F면 v_j 도 F여야 함: j 에서 i 로 가는 무한 간선

• 5/6번 + v_i 와 v_j 가 달라야 한다는 조건이 있으면 MAX-2SAT

• Maximum Density Subgraph (NEERC'06H, BOJ 3611 팀의 난이도)

- density $\geq x$ 인 subgraph가 있는지 이분 탐색

- 정점 N 개, 간선 M 개, 차수 D_i 개

- 그래프의 간선마다 용량 1인 양방향 간선 추가

- 소스에서 정점으로 용량 M , 정점에서 싱크로 용량 $M - D_i + 2x$

- min cut에서 S와 불어 있는 애들이 1개 이상이면 x 이상이고, 그에 subgraph의 정점들

- while($r-l \geq 1.0/(n^*n)$) 으로 해야 함. 너무 많이 돌리면 실수 오차

11.12 Matrix with Graph(Kirchhoff, Tutte, LGV)

• Kirchhoff's Theorem: 그래프의 스패닝 트리 개수

- $m[i][j] := -(i-j \text{ 간선 개수})$ ($i \neq j$) / (유향) $-(i \rightarrow j \text{ 간선})$

- $m[i][i] :=$ 정점 i 의 degree / (유향) 정점 i 의 in degree

- res = (m 의 첫 번째 행과 첫 번째 열을 없앤 $(n-1)$ by $(n-1)$ matrix의 행렬식)

- (유향) m 의 루트 번째 행과 열을 삭제한 행렬의 행렬식

• Tutte Matrix: 그래프의 최대 매칭

- $m[i][j] :=$ 간선 (i, j) 가 없으면 0, 있으면 $i < j ? r : -r$, r 은 $[0, P)$

구간의 임의의 정수

- rank(m)/2가 높은 확률로 최대 매칭 ($\text{mod } P$)

• LGV Theorem: 간선에 가중치 있는 DAG에서 어떤 경로 P 의 간선 가중치

증치 곱을 $w(P)$, 모든 $a \rightarrow b$ 경로들의 $w(P)$ 의 합을 $e(a, b)$ 라고 하자. n

개의 시작점 a_i 와 도착점 b_j 가 주어졌을 때, 서로 정점이 겹치지 않는 n

개의 경로로 시작점과 도착점을 일대일 대응시키는 모든 경우에서 $w(P)$ 의 곱의 합은 $\det M(i, j) = e(a_i, b_j)$ 와 같음. 따라서 모든 가중치를 1로 두면 서로소 경로 경우의 수를 구함

11.13 About Graph Matching(Graph with $|V| \leq 500$)

• Game on a Graph: s 에 토큰이 있음. 플레이어는 각자의 텁마다 토큰을 인접한 정점으로 옮기고 못 옮기면 짐. s 를 포함하지 않는 최대 매칭이 존재함 \Leftrightarrow 후공이 이김

• Chinese Postman Problem: 모든 간선을 방문하는 최소 가중치 Walk를 구하는 문제. Floyd를 돌린 다음, 홀수 정점들을 모아서 최소 가중치 매칭(홀수 정점은 짝수 개 존재)

• Unweighted Edge Cover: 모든 정점을 덮는 가장 작은(minimum cardinality/weight) 간선 집합을 구하는 문제

$|V| - |M|$, 길이 3짜리 경로 없음, star graph 여러 개로 구성

• Weighted Edge Cover: $\text{sum}_{v \in V}(w(v)) - \text{sum}_{(u,v) \in M}(w(u) + w(v) - d(u, v))$, $w(x)$ 는 x 와 인접한 간선의 최소 가중치

• NEERC'18 B: 각 기계마다 2명의 노동자가 다뤄야 하는 문제. 기계마다 두 개의 정점을 만들고 간선으로 연결하면 정답은 $|M| - |\text{기계}|$ 임. 정답에 1/2씩 기여한다는 점을 생각해보면 좋음.

• Min Disjoint Cycle Cover: 정점이 중복되지 않으면서 모든 정점을 덮는 길이 3 이상의 사이클을 집합을 찾는 문제. 모든 정점은 2개의 서로 다른 간선, 일부 간선은 양쪽 끝점과 매칭되어야 하므로 플로우를 생각할 수 있지만 용량 2짜리 간선에 유량을 1만큼 흘릴 수 있으므로 플로우는 불가능.

각 정점과 간선을 2개씩 $((v, v'), (e_{i,u}, e_{i,v}))$ 로 복사하자. 모든 간선 $e = (u, v)$ 에 대해 e_u 와 e_v 를 잇는 가중치 w 짜리 간선을 만들고(like NEERC'18), $(u, e_{i,u}), (u', e_{i,u}), (v, e_{i,v}), (v', e_{i,v})$ 를 연결하는 가중치 0짜리 간선을 만들자. Perfect 매칭이 존재함 \Leftrightarrow Disjoint Cycle Cover 존재. 최대 가중치 매칭 찾은 뒤 모든 간선 가중치 합에서 매칭 빼면 됨.

• Two Matching: 각 정점이 최대 2개의 간선과 인접할 수 있는 최대 가중치 매칭 문제.

각 컴포넌트는 정점 하나/경로/사이클이 되어야 함. 모든 서로 다른 정점 쌍에 대해 가중치 0짜리 간선 만들고, 가중치 0짜리 (v, v') 간선 만들면 Disjoing Cycle Cover 문제가 됨. 정점 하나만 있는 컴포넌트는 self-loop, 경로 형태의 컴포넌트는 양쪽 끝점을 연결한다고 생각하면 편함.

11.14 Checklist

- (예비소집) bits/stdc++.h, int128, long double 80bit, avx2 확인
- (예비소집) 스택 메모리(지역 변수, 재귀, 람다 재귀), 제출 파일 크기 확인
- (예비소집) MLE(힙,스택), stderr 출력 RTE?, 줄 앞뒤 공백 채점 결과
- 비슷한 문제를 풀어본 적이 있는가?
- 단순한 방법에서 시작할 수 있을까? (Brute Force)
- 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결하면서)
- 문제를 단순화할 수 없을까? / 그럼으로 그려볼 수 있을까?
- 수식으로 표현할 수 있을까? / 문제를 분해할 수 있을까?
- 뒤에서부터 생각해서 풀 수 있을까? / 순서를 강제할 수 있을까?
- 특정 형태의 답만을 고려할 수 있을까? (정규화)
- 구간을 통째로 가져간다 : 플로우 + 적당한 자료구조
 $(i, i+1, k, 0), (s, e, 1, w), (N, T, k, 0)$
- a = b : a만 이동, b만 이동, 두 개 동시에 이동, 반대로 이동
- 말도 안 되는 것 / 당연하다고 생각한 것 다시 생각해 보기
- Directed MST / Dominator Tree
- 일정 비율 충족 or 2 3개로 모두 커버 : 랜덤
- 확률 : DP, 이분 탐색(NYPC 2019 Finals C)
- 최대/최소 : 이분 탐색, 그리디(Prefix 고정, Exchange Argument), DP(순서 고정)
- 냅색: 파라미터 순서 변경, min plus convolution, FFT
- signal(SIGSEGV, []{int){_Exit(0);}}; converts segfaults into WA.
SIGABRT(assertion fail), SIGFPE(0div)
- feenableexcept(29) kills problem on NaNs(1), 0div(4), inf(8), denormals(16)