

Advanced Algorithm Study

November 8, 2019

Contents

1 자료구조

1.1 세그먼트 트리	2
1.2 세그먼트 트리 + Lazy Propagation	2
1.3 펜윅 트리(BIT)	3
1.4 유니온 - 파인드	3
1.5 Ordered - Set	3
1.6 Persistent 세그먼트 트리	4

2 그래프

2.1 SPFA	4
2.2 네트워크 플로우 - Easy (에드몬드 카프)	5
2.3 네트워크 플로우 - Fast (디닉)	6
2.4 MCMF (Minimum Cost Maximum Flow)	6
2.5 이분 매칭 - Easy	7
2.6 이분 매칭 - Fast (호프크로프트 카프)	8
2.7 SCC (타잔 알고리즘)	8
2.8 2-SAT	9
2.9 LCA (최소 공통 조상)	10
2.10 BCC - 단절점 단절선	10
2.11 Heavy-light Decomposition	11

3 수학

3.1 빠른 거듭제곱 & 나눗셈의 역원 (페르마 소정리)	12
3.2 조합(nCr) 구하기	12
3.3 소수 찾기, 소인수분해 - Easy + 각종 체	12
3.4 소수 찾기 - Fast (밀러-라빈)	13
3.5 소인수분해 - Fast (풀라드-로)	14
3.6 FFT (고속 푸리에 변환)	14

2.7 중국인의 나머지 정리 + 확장 유클리드	15
4 문자열	16
4.1 KMP	16
4.2 Rabin - Karp 해싱	16
4.3 Suffix Array + LCP	16
4.4 모든 팰린드롬 찾기 - Manacher	17
4.5 Z 알고리즘 - 접미사의 접두사	18
4.6 트라이 - 문자열 전용 set	18
4.7 아호 코라식 - 다중 문자열 KMP	18
5 기하	19
5.1 기본 기하 라이브러리 + CCW	19
5.2 볼록 껍질(컨벡스 힐)	20
5.3 회전하는 캘리퍼스 - 볼록 껍질에서 가장 먼 점	20
6 기타	21
6.1 위상 정렬	21
6.2 Mo's Algorithm + Sqrt Decomposition	21
6.3 방향 그래프 사이클 판정	22
6.4 IO Trick	22
6.5 삼분 탐색	22
6.6 Convex Hull Trick	23
6.7 Berlekamp-Massey - 선형 점화식 DP 치트키	23
6.8 알쓸신잡	24
6.9 자주 쓰는데 헷갈리는 것들	25

1 자료구조

1.1 세그먼트 트리

* 최소값 세그먼트 트리

```
const int MAX, INF;
int arr[MAX], seg[MAX*4];
int init(int l, int r, int node){
    if(l == r) return seg[node] = arr[l];
    int mid = (l+r)/2;
    return seg[node] = min(init(l, mid, node*2),
                           init(mid+1, r, node*2+1));
}
int query(int l, int r, int node, int nodeL, int nodeR){
    if(r < nodeL || nodeR < l) return INF;
    if(l <= nodeL && nodeR <= r) return seg[node];
    int mid = (nodeL + nodeR)/2;
    return min(query(l, r, node*2, nodeL, mid), query(l, r, node*2+1,
                                                       mid+1, nodeR));
}
int update(int index, int value, int node, int nodeL, int nodeR){
    if(index<nodeL || nodeR<index) return seg[node];
    if(nodeL == nodeR) return seg[node] = min(seg[node], value);
    int mid = (nodeL + nodeR)/2;
    return seg[node] = min(update(index,value,node*2,nodeL,mid),
                           update(index,value,node*2+1,mid+1,nodeR));
}
```

1.2 세그먼트 트리 + Lazy Propagation

* 구간합 세그먼트 트리, INF = 항등원

```
const int MAX;
struct lazySeg{
    ll arr[MAX], seg[MAX*4], lazy[MAX*4];
    ll init(int node, int l, int r){
        if(l==r) return seg[node] = arr[l];
        int mid = (l+r)/2;
        return seg[node] = init(node*2,l,mid) + init(node*2+1,mid+1,r);
    }
};
```

```
//현 노드의 lazy 값을 해소
void update_lazy(int node, int nodeL, int nodeR){
    if(lazy[node] != 0){
        seg[node] += (nodeR - nodeL +1)*lazy[node];
        //leaf가 아니면 자식에 전파
        if(nodeL != nodeR){
            lazy[node*2] += lazy[node];
            lazy[node*2+1] += lazy[node];
        }
        lazy[node] = 0;
    }
}
ll update(int l, int r, ll value, int node, int nodeL, int nodeR){
    update_lazy(node, nodeL, nodeR);
    if(r<nodeL || nodeR<l) return seg[node];
    if(l <= nodeL && nodeR <= r){
        seg[node] += (nodeR - nodeL +1)*value;
        //leaf가 아니면 자식에 전파
        if(nodeL != nodeR){
            lazy[node*2] += value;
            lazy[node*2+1] += value;
        }
        return seg[node];
    }
    int mid = (nodeL+nodeR)/2;
    return seg[node] = update(l,r,value,node*2,nodeL,mid) +
        update(l,r,value,node*2+1,mid+1,nodeR);
}
ll query(int l, int r, int node, int nodeL, int nodeR){
    update_lazy(node, nodeL, nodeR);
    if(r<nodeL || nodeR<l) return 0;
    if(l <= nodeL && nodeR <= r) return seg[node];
    int mid = (nodeL + nodeR)/2;
    return query(l,r,node*2,nodeL,mid) +
        query(l,r,node*2+1,mid+1,nodeR);
};
```

1.3 펜윅 트리(BIT)

* $k\text{th}(k) = k\text{번째 원소}$ (1-indexed)

```
const int MAX;
struct BIT{
    int seg[MAX+1] = {};
    int query(int idx){
        int ret = 0;
        while(idx > 0){
            ret += seg[idx];
            idx -= idx & -idx;
        }
        return ret;
    }
    void update(int idx, int a){
        while(idx <= MAX){
            seg[idx] += a;
            idx += idx & -idx;
        }
    }
    int kth(int k){
        int ret = 0;
        for(int i=30; i>=0; --i){
            int pivot = ret + (1<<i);
            if(pivot <= MAX && seg[pivot] < k){
                k -= seg[pivot];
                ret = pivot;
            }
        }
        return ret+1;
    }
};
```

1.4 유니온 - 파인드

* 초기화 : `memset(parent, -1, sizeof(parent))`
 $\text{parent}[i] < 0$ 이면 해당 집합의 크기

```
const int MAX;
int parent[MAX];
```

```
int find(int a){
    if(parent[a]<0) return a;
    return parent[a] = find(parent[a]);
}
void merge(int a, int b){
    a = find(a), b = find(b);
    if(a==b) return;
    if(parent[a] < parent[b]) swap(a,b);
    parent[b] += parent[a];
    parent[a] = b;
}
//must memset parent

1.5 Ordered - Set
* GCC에서만 가능한 치트키

#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/trie_policy.hpp>

using namespace std;
using namespace __gnu_pbds;

typedef tree<int, null_type, less<int>, rb_tree_tag,
tree_order_statistics_node_update> ordered_set;
// multiset 사용시 pair<원소값, cnt++>

int main(){
    ordered_set X;

    //insert(a) : a를 삽입한다
    X.insert(1); // 0번째 원소 = 1
    X.insert(2); // 1번째 원소 = 2
    X.insert(4); // 2번째 원소 = 4
    X.insert(8); // 3번째 원소 = 8
    X.insert(16); // 4번째 원소 = 16

    //iterator
    for(auto it = X.begin(); it != X.end(); ++it)
        cout << *it << ' ';
```

```

cout << '\n'; // 1 2 4 8 16

//find_by_order(a) : a번째 원소의 iterator를 반환한다.
//단, a > X.size()면 X.end()를 반환한다.
cout<<*X.find_by_order(1)<<endl; // 2
cout<<*X.find_by_order(2)<<endl; // 4
cout<<*X.find_by_order(4)<<endl; // 16
cout<<(X.end()==X.find_by_order(6))<<endl; // true

//order_of_key(a) : a보다 작은 원소의 개수를 반환한다.
cout<<X.order_of_key(-5)<<endl; // 0
cout<<X.order_of_key(1)<<endl; // 0
cout<<X.order_of_key(3)<<endl; // 2
cout<<X.order_of_key(4)<<endl; // 2
cout<<X.order_of_key(400)<<endl; // 5
}

```

1.6 Persistent 세그먼트 트리

* 공간 복잡도가 $N \log N$ 인 2차원 세그먼트 트리

```

const int MAX = 100100;
struct PersistentSegmentTree{
    struct Node{
        int value = 0; // 반드시 항등원
        Node *left = 0, *right = 0;
    };
    vector<Node*> Tree;
    void init(){
        Tree.clear();
        Tree.push_back(new Node());
    }
    void make_tree(int idx, int value){
        Tree.push_back(make_tree(Tree.back(), idx, value, 0, MAX-1));
    }
    Node* make_tree(Node *prev, int idx, int value, int nodeL, int nodeR){
        Node *now = new Node();
        if(idx < nodeL || nodeR < idx) return prev;
        if (nodeL == nodeR) {
            now->value = prev->value + value;

```

```

            return now;
        }
        int mid = (nodeL + nodeR) / 2;
        if (!prev->left)
            prev->left = new Node(), prev->right = new Node();
        now->left = make_tree(prev->left, idx, value, nodeL, mid);
        now->right = make_tree(prev->right, idx, value, mid+1, nodeR);
        now->value = now->left->value + now->right->value;
        return now;
    }
    int query(int root_idx, int l, int r){
        return query(Tree[root_idx], l, r, 0, MAX-1);
    }
    int query(Node *here, int l, int r, int nodeL, int nodeR){
        if(here == 0) return 0;
        if(nodeR < l || r < nodeL) return 0;
        if(l <= nodeL && nodeR <= r) return here -> value;
        int mid = (nodeL + nodeR)/2;
        return query(here->left, l, r, nodeL, mid) + query(here->right,
        l, r, mid+1, nodeR);
    }
};

for(int i=0; i<size_of_x; ++i)
    PST.make_tree(p.yy, 1);

```

2 그래프

2.1 SPFA

* 음수 사이클을 해결 가능한 다익스트라
 $\text{vector<int> adj[MAX] = (가중치, 번호)}$
 시간복잡도: $\mathcal{O}(VE)$, Average = $\mathcal{O}(E)$

```

vector<int> SPFA(int n, int start, auto &adj){
    vector<int> dist(n, INF);
    queue<int> Q;
    bool inQ[MAX] = {};
    int visit[MAX] = {};
    dist[start] = 0;
    Q.push(start);

```

```

inQ[start] = true;
++visit[start];
while(!Q.empty()){
    int here = Q.front();
    Q.pop(); inQ[here] = false;
    for(auto &P : adj[here]){
        int there = P.first, newDist = dist[here] + P.second;
        if(newDist < dist[there]){
            dist[there] = newDist;
            if(!inQ[there]){
                Q.push(there);
                inQ[there] = true;
                if(++visit[there] >= n)
                    return vector<int>();
            }
        }
    }
}
return dist;
}

```

2.2 네트워크 플로우 - Easy (에드몬드 카프)

* INF = 최대 유량 값(f)
 Max-Flow Min-Cut : S와 E로 정점 그룹을 나눌 때, 최대 매칭이 곧 최소 컷의 용량의 합이다
 즉, 잔여 용량이 남은 간선을 타고 갈 수 있는 정점은 S에 속한다.
 시간복잡도: $\min(O(VE^2), O(Ef))$

```

const int MAX, INF;
struct Edge{
    int next, cap, flow=0;
    Edge* rev;
    Edge() {};
    Edge(int _next, int _cap) : next(_next), cap(_cap) {};
    int remain(){
        return cap - flow;
    }
    void push(int x){
        flow += x;
        rev->flow -= x;
    }
};

```

```

    }
};

vector<Edge*> adj[MAX];
inline void makeEdge(int u, int v, int cap){
    Edge *uv = new Edge(v, cap), *vu = new Edge(u, 0);
    uv->rev = vu, vu->rev = uv;
    adj[u].push_back(uv);
    adj[v].push_back(vu);
}

int maxFlow(int S, int E){
    int total = 0;
    while(true){
        int parent[MAX];
        Edge *path[MAX];
        memset(parent, -1, sizeof(parent));
        queue<int> Q;
        Q.push(S);
        while(!Q.empty()){
            int here = Q.front(); Q.pop();
            for(auto e : adj[here]){
                int there = e->next;
                if(e->remain() > 0 && parent[there] == -1){
                    Q.push(there);
                    parent[there] = here;
                    path[there] = e;
                    if(there == E) break;
                }
            }
        }
        if(parent[E] == -1) return total;
        int flow = INF;
        for(int i=E; i!=S; i=parent[i])
            flow = min(flow, path[i]->remain());
        for(int i=E; i!=S; i=parent[i])
            path[i]->push(flow);
        total += flow;
    }
}

```

2.3 네트워크 플로우 - Fast (디닉)

* INF = 최대 유량 값, DinicMaxFlow(S, E)
레벨은 ”소스까지의 최단거리의 간선의 개수”를 의미한다.

BFS로 ’레벨 그래프’를 만든다. 레벨 그래프에서 포함하는 간선은
1. 용량이 남아있는 간선 2. 레벨의 차이가 1인 정점을 연결하는 간선
레벨 그래프를 생성하고, 에드먼드 카프처럼 DFS를 돌려 유량을 흘려보내 준다.
시간복잡도: $\mathcal{O}(EV^2)$

```
const int MAX, INF;
int level[MAX], work[MAX];
struct Edge{
    int next, cap, flow=0;
    Edge* rev;
    Edge(){};
    Edge(int _next, int _cap) : next(_next), cap(_cap) {};
};
vector<Edge*> adj[MAX];
inline void makeEdge(int u, int v, int cap){
    Edge *uv = new Edge(v, cap), *vu = new Edge(u, 0);
    uv->rev = vu, vu->rev = uv;
    adj[u].push_back(uv);
    adj[v].push_back(vu);
}
//싱크까지 도달 가능한 레벨 그래프를 생성 할 수 있는가?
bool BFSdinic(int S, int E){
    memset(level, -1, sizeof(level));
    level[S] = 0;
    queue<int> Q;
    Q.push(S);
    while(!Q.empty()){
        int here = Q.front(); Q.pop();
        for(auto &i : adj[here]){
            int there = i->next;
            //레벨이 정해지지 않고, 남은 용량이 있는 간선인가?
            if(level[there] == -1 && i->cap >0){
                level[there] = level[here]+1;
                Q.push(there);
            }
        }
    }
}
```

```
return (level[E] != -1);
}
//here에서 E에 도달할 때 까지 flow 만큼 흘려보내본다.
//싱크까지 얼마나 유량을 흘릴 수 있었는지 리턴한다.
int DFSdinic(int here, int flow, int E){
    if(here == E) return flow;
    int size = adj[here].size();
    for(int &i = work[here]; i<size; ++i){
        int there = adj[here][i]->next, c = adj[here][i]->cap;
        if(level[here]+1 == level[there] && c >0){
            int nc = DFSdinic(there, min(flow, c), E);
            if(nc){
                adj[here][i]->cap -= nc;
                adj[here][i]->rev->cap += nc;
                return nc;
            }
        }
    }
    return 0;
}

int DinicMaxFlow(int S, int E){
    memset(level, 0, sizeof(level));
    int total = 0;
    while(BFSdinic(S,E)){
        memset(work, 0, sizeof(work));
        while(true){
            int flow = DFSdinic(S, INF, E);
            if(flow == 0) break;
            total += flow;
        }
    }
    return total;
}
```

2.4 MCMF (Minimum Cost Maximum Flow)

* INF = 가중치의 최장거리
(가중치의 최소값, 유량의 최대값)을 반환한다.
시간복잡도: $\mathcal{O}(VEf)$, Average = $\mathcal{O}(Ef)$

```

const int MAX, INF; //MAX = 최대 정점 개수, INF = 가중치의 최장거리
struct Edge{
    int next, cap, cost, flow=0;
    Edge* rev;
    Edge() {};
    Edge(int _next, int _cap, int _cost) : next(_next), cap(_cap),
    cost(_cost) {};
    int remain(){
        return cap - flow;
    }
    void push(int x){
        flow += x;
        rev->flow -= x;
    }
};
vector<Edge*> adj[MAX];
inline void makeEdge(int u, int v, int cap, int cost){
    Edge *uv = new Edge(v, cap, cost), *vu = new Edge(u, 0, -cost);
    uv->rev = vu, vu->rev = uv;
    adj[u].push_back(uv);
    adj[v].push_back(vu);
}
pair<int,int> MCMF(int S, int E){
    int minCost = 0, maxFlow = 0;
    while(true){
        int parent[MAX], dist[MAX];
        Edge *path[MAX];
        queue<int> Q;
        bool inQ[MAX] = {};
        memset(parent, -1, sizeof(parent));
        fill(dist, dist+MAX, INF);
        dist[S] = 0;
        Q.push(S);
        inQ[S] = true;
        while(!Q.empty()){
            int here = Q.front(); Q.pop();
            inQ[here] = false;
            for(auto e : adj[here]){
                int there = e->next;
                if(e->remain() > 0 && dist[there] > dist[here] + e->cost){

```

```

                    dist[there] = dist[here] + e->cost;
                    parent[there] = here;
                    path[there] = e;
                    if(!inQ[there]){
                        Q.push(there);
                        inQ[there] = true;
                    }
                }
            }
            if(parent[E] == -1) break;
            int flow = INF;
            for(int i=E; i!=S; i=parent[i])
                flow = min(flow, path[i]->remain());
            for(int i=E; i!=S; i=parent[i]){
                minCost += flow * path[i]->cost;
                path[i]->push(flow);
            }
            maxFlow += flow;
        }

        return {minCost, maxFlow};
    }
}

```

2.5 이분 매칭 - Easy

* adj는 A에서 B로 가는 간선들만 담는다.
배열 A[i]=j면 i와 j가 매칭됐음을 의미한다.

쾨닉의 정리 : 최소 버텍스 커버 = 최대 이분 매칭 (A 중 소스에서 도달 불가능 + B 중 소스에서 도달 가능)

시간복잡도: $\mathcal{O}(VE)$

```

const int MAXA, MAXB;
int A[MAXA], B[MAXB];
bool visit[MAXA];
vector<int> adj[MAXA];
bool DFS(int here){
    if(visit[here]) return false;
    visit[here] = true;
    for(int &there : adj[here])
        if(B[there] == -1 || DFS(B[there])){

```

```

        A[here] = there, B[there] = here;
        return true;
    }
    return false;
}
int biMatch(){
    memset(A, -1, sizeof(A));
    memset(B, -1, sizeof(B));
    int match = 0;
    for(int i=0; i<MAXA; ++i)
        if(A[i] == -1){
            memset(visit, 0, sizeof(visit));
            if(DFS(i))
                ++match;
        }
    return match;
}

```

2.6 이분 매칭 - Fast (호프크로프트 카프)

- * $n = A$ 의 크기, $MAX = \max(A\text{size}, B\text{size})$
 - 1. bfs-bmatching으로 레벨[=매칭 중이지 않은 정점과의 최단거리]을 매긴다.
 - 2. dfs-bmatching으로 level을 이용해 효율적으로 이분매칭을 할 수 있다
- 시간복잡도: $\mathcal{O}(E\sqrt{V})$

```

const int MAX; //MAX = max(Asize,Bsize)
int n, A[MAX], B[MAX], dist[MAX];
bool used[MAX];
vector<int> adj[MAX];
void bfs_bmatching(){
    queue<int> Q;
    for(int i=0; i<n; ++i){
        if(!used[i]){
            dist[i] = 0;
            Q.push(i);
        }
        else dist[i] = MAX*2+1;
    }
    while(!Q.empty()){
        int a = Q.front();
        Q.pop();

```

```

        for(int &b : adj[a]){
            if(B[b] != -1 && dist[B[b]] == MAX*2+1){
                dist[B[b]] = dist[a] + 1;
                Q.push(B[b]);
            }
        }
    }
    bool dfs_bmatching(int a){
        for(int &b : adj[a]){
            if(B[b] == -1 || (dist[B[b]] == dist[a]+1 &&
dfs_bmatching(B[b]))){
                used[a] = true;
                A[a] = b;
                B[b] = a;
                return true;
            }
        }
        return false;
    }
    int Hopcroft_Karp(){
        int match = 0, flow;
        memset(A, -1, sizeof(A));
        memset(B, -1, sizeof(B));
        while(1){
            bfs_bmatching();
            flow = 0;
            for(int i=0; i<n; ++i)
                if(!used[i] && dfs_bmatching(i)) flow++;
            if(!flow) break;
            match += flow;
        }
        return match;
    }
}

```

2.7 SCC (타잔 알고리즘)

- * SCC 정의 : 다른 그룹끼리 사이클이 없다.
- SCC(V)를 호출 하면 result에 SCC가 분류되어 저장된다.
- 이 때 그룹은 반드시 위상정렬의 역순으로 반환된다.

sccNumber[i] = i번 정점이 속하는 그룹

시간복잡도: $\mathcal{O}(V + E)$

```

const int MAX; //정점의 최대 개수
int discovered[MAX], sccNumber[MAX], cnt1, cnt2;
vector<int> adj[MAX];
stack<int> S;
vector<vector<int>> result;
int DFSscc(int here) {
    discovered[here] = cnt1++;
    int ret = discovered[here];
    S.push(here);
    for (int &there : adj[here]) {
        if (discovered[there] == -1)
            ret = min(ret, DFSscc(there));
        else if (sccNumber[there] == -1)
            ret = min(ret, discovered[there]);
    }
    if(ret == discovered[here]){
        result.push_back(vector<int>());
        auto &V = result.back();
        while(true){
            int tmp = S.top(); S.pop();
            sccNumber[tmp] = cnt2;
            V.push_back(tmp);
            if(tmp == here) break;
        }
        ++cnt2;
    }
    return ret;
}
void SCC(int V) {
    memset(discovered, -1, sizeof(discovered));
    memset(sccNumber, -1, sizeof(sccNumber));
    cnt1 = 0, cnt2 = 0;
    stack<int> S;
    for(int i=0; i<V; ++i)
        if(discovered[i] == -1)
            DFSscc(i);
}

```

2.8 2-SAT

* makeEdge = X —— Y인 엣지를 만든다. true = $2X+1$, false = $2X$
정점들의 참, 거짓 여부가 담겨 있는 벡터를 반환한다. 불가능하면, 빈 벡터를 반환한다

시간복잡도: $\mathcal{O}(V + E)$

```

const int MAX; //정점의 최대 개수
int discovered[MAX], sccNumber[MAX], cnt1, cnt2;
vector<int> adj[MAX];
stack<int> S;
vector<vector<int>> result;
int DFSscc(int here) {
    discovered[here] = cnt1++;
    int ret = discovered[here];
    S.push(here);
    for (int &there : adj[here]) {
        if (discovered[there] == -1)
            ret = min(ret, DFSscc(there));
        else if (sccNumber[there] == -1)
            ret = min(ret, discovered[there]);
    }
    if(ret == discovered[here]){
        result.push_back(vector<int>());
        auto &V = result.back();
        while(true){
            int tmp = S.top(); S.pop();
            sccNumber[tmp] = cnt2;
            V.push_back(tmp);
            if(tmp == here) break;
        }
        ++cnt2;
    }
    return ret;
}
void SCC(int V) {
    memset(discovered, -1, sizeof(discovered));
    memset(sccNumber, -1, sizeof(sccNumber));
    cnt1 = 0, cnt2 = 0;
    stack<int> S;
    for(int i=0; i<V; ++i)

```

```

if(discovered[i] == -1)
    DFSscc(i);
}

```

2.9 LCA (최소 공통 조상)

* NlogN만에 전처리를 하고, logN에 쿼리를 답한다

```

const int MAX, pMAX;
int depth[MAX], parent[MAX][pMAX];
void init(){
    //TODO : make_tree : fill depth[i] & parent[i][0]
    //parent[i][j] = i의 1<<j번째 조상은?
    for(int j=1; j<pMAX; ++j)
        for(int i=1; i<=n; ++i)
            parent[i][j] = parent[parent[i][j-1]][j-1];
}
int LCA(int u, int v){
    if(depth[u] > depth[v]) swap(u,v);
    //v를 u의 높이까지 옮린다
    int diff = depth[v] - depth[u];
    for(int i=0; i<pMAX; ++i){
        if(diff & (1<<i))
            v = parent[v][i];
    }
    if(u == v) return u;
    //부모가 다르면 옮긴다
    for(int i=pMAX-1; i>=0; --i)
        if(parent[u][i] != parent[v][i])
            u = parent[u][i], v = parent[v][i];
    return parent[u][0];
}

```

2.10 BCC - 단절점 단절선

시간복잡도: $\mathcal{O}(V + E)$

```

const int MAXN = 100;
vector<pair<int, int>> graph[MAXN]; // { next vertex id, edge id }
int up[MAXN], visit[MAXN], vtime;
vector<int> stk;

```

```

int is_cut[MAXN]; // v is cut vertex if is_cut[v] > 0
vector<int> bridge; // list of edge ids
vector<int> bcc_edges[MAXN]; // list of edge ids in a bcc
int bcc_cnt;
void dfs(int nod, int par_edge) {
    up[nod] = visit[nod] = ++vtime;
    int child = 0;
    for (const auto& e : graph[nod]) {
        int next = e.first, eid = e.second;
        if (eid == par_edge) continue;
        if (visit[next] == 0) {
            stk.push_back(eid);
            ++child;
            dfs(next, eid);
            if (up[next] == visit[next]) bridge.push_back(eid);
            if (up[next] >= visit[nod]) {
                ++bcc_cnt;
                do {
                    auto lasteid = stk.back();
                    stk.pop_back();
                    bcc_edges[bcc_cnt].push_back(lasteid);
                    if (lasteid == eid) break;
                } while (!stk.empty());
                is_cut[nod]++;
            }
            up[nod] = min(up[nod], up[next]);
        }
    }
    else if (visit[next] < visit[nod]) {
        stk.push_back(eid);
        up[nod] = min(up[nod], visit[next]);
    }
}
if (par_edge == -1 && is_cut[nod] == 1)
    is_cut[nod] = 0;
}
// find BCCs & cut vertexes & bridges in undirected graph O(V+E)
void get_bcc() {
    vtime = 0;
    memset(visit, 0, sizeof(visit));
    memset(is_cut, 0, sizeof(is_cut));
}

```

```

bridge.clear();
for (int i = 0; i < n; ++i) bcc_edges[i].clear();
bcc_cnt = 0;
for (int i = 0; i < n; ++i) {
    if (visit[i] == 0)
        dfs(i, -1);
}
}

```

2.11 Heavy-light Decomposition

* 모든 경로가 항상 $\log N$ 개의 체인만 지난다. set, Segment Tree 등을 끼얹기.
1. adj 채우고 2. make_tree(root,root) 3.HLD(root,root,root)
가중치가 Edge에 붙는 구현. 노드에 붙으면 큐리를 DFS_cnt[x] + 1 대신 DFS_cnt[x].

```

const int MAX = 100100;
struct HeavyLight{
    struct SegmentTree{
        int seg[MAX*4] = {};
        int query(int l, int r, int node, int nodeL, int nodeR){
            if(l > r) return 0;
            //...
        };
        vector<int> adj[MAX]; //입력 필요
        int level[MAX]={}, sub_size[MAX]={}, parent[MAX]={};
        int DFS_cnt[MAX]={}, dn=0, chain[MAX]={}, tail[MAX]={};
        void make_tree(int here, int p){
            parent[here] = p;
            sub_size[here] = 1;

            for(int &i : adj[here]){
                if(i == parent[here]) continue;
                level[i] = level[here]+1;
                make_tree(i, here);
                sub_size[here] += sub_size[i];
            }
        }
        // DFS_cnt[i] = i번 정점의 DFS 넘버
        // chain[i] = DFS 넘버가 i인 정점이 속하는 체인 넘버
        // 체인의 머리의 노드 넘버는, 체인 넘버와 동일하다
        // tail[i] = 체인 넘버가 i인 체인의 꼬리의 DFS 넘버
    };
}

```

```

void HLD(int here, int p, int chain_cnt){
    DFS_cnt[here] = ++dn;
    chain[dn] = chain_cnt;
    tail[chain_cnt] = dn;

    int heavy_idx = -1;
    for(int &i : adj[here])
        if(i != p && (heavy_idx == -1 || sub_size[i] > sub_size[heavy_idx]))
            heavy_idx = i;

    if(heavy_idx != -1) HLD(heavy_idx, here, chain_cnt);

    for(int &i : adj[here])
        if(i != p && i != heavy_idx)
            HLD(i, here, i);
}

int LCA(int x, int y){
    //같은 체인에 속하는 경우 루트에 가까운 정점이 LCA
    if(chain[DFS_cnt[x]] == chain[DFS_cnt[y]])
        return level[x] < level[y] ? x : y;

    int x_head = chain[DFS_cnt[x]], y_head = chain[DFS_cnt[y]];
    if(level[x_head] > level[y_head])
        return LCA(parent[x_head], y);
    else return LCA(x, parent[y_head]);
}

SegmentTree seg;
void update(int node, int value){
    seg.update(DFS_cnt[node], value, 1, 0, MAX-1);
}
int query(int x, int y){
    //같은 체인에 속하는 경우
    if(chain[DFS_cnt[x]] == chain[DFS_cnt[y]]){
        if(level[x] > level[y]) swap(x,y);
        return seg.query(DFS_cnt[x]+1, DFS_cnt[y], 1, 0, MAX-1);
    }
    int x_head = chain[DFS_cnt[x]], y_head = chain[DFS_cnt[y]];
    if(level[x_head] > level[y_head])

```

```

        return max(seg.query(DFS_cnt[x_head], DFS_cnt[x], 1, 0, MAX-1)
            , query(parent[x_head], y));
    else
        return max(seg.query(DFS_cnt[y_head], DFS_cnt[y], 1, 0, MAX-1)
            , query(x, parent[y_head]));
}
};

```

3 수학

3.1 빠른 거듭제곱 & 나눗셈의 역원 (페르마 소정리)

* a와 MOD가 서로소여야 divmod를 사용 할 수 있다!
 $\text{fastPow}(a,b) : (a^b)\%MOD$ 를 반환한다
 $\text{divmod}(a,b) : (a/b)\%MOD$ 를 반환한다

```

long long fastPow(long long a, long long b){
    if(b==0) return 1;
    if(b&1) return (a * fastPow(a, b-1))%MOD;
    auto tmp = fastPow(a, b/2);
    return (tmp * tmp)%MOD;
}
// return (a/b)%MOD, MOD = prime
inline long long divmod(long long a, long long b){
    b = fastPow(b, MOD-2);
    return (a*b)%MOD;
}

```

3.2 조합(nCr) 구하기

* Method 1 : 1의 역원을 전처리로 구해놓는다 Method 2 : 루카의 정리
시간복잡도: Method 1 : init = $\mathcal{O}(N)$, query = $\mathcal{O}(1)$
Method 2 : init = $\mathcal{O}(MOD^2)$, query = $\mathcal{O}(\log_{MOD} N)$

```

// Method 1
const int MAX, MOD; //MAX = n의 최대값
long long inv[MAX+1], fac[MAX+1], facInv[MAX+1];
void combInit(int n){
    fac[1] = 1;
    for(int i=2; i<=n; ++i)

```

```

        fac[i] = (fac[i-1]*i) %MOD;
    inv[1] = 1;
    for(int i=2; i<=n; ++i)
        inv[i] = (MOD - MOD/i) * inv[MOD%i] %MOD;
    facInv[1] = 1;
    for(int i=2; i<=n; ++i)
        facInv[i] = (facInv[i-1]*inv[i]) %MOD;
    }

    inline long long comb(int n, int r){
        if(r == 0 || n == r) return 1;
        return (((fac[n]*facInv[r])%MOD)*facInv[n-r])%MOD;
    }

    // Method 2
    const int MOD;
    long long combMOD[2000][2000];
    void combInit(){
        for(int i=0; i<MOD; ++i){
            combMOD[i][0] = 1;
            for(int j=1; j<=i; ++j)
                combMOD[i][j] = (combMOD[i-1][j-1] + combMOD[i-1][j])%MOD;
        }
    }

    long long comb(long long n, long long k){
        long long ret = 1;
        while(n>0 || k>0){
            ret = (ret * combMOD[n%MOD][k%MOD])%MOD;
            n /= MOD, k /= MOD;
        }
        return ret;
    }
}
```

3.3 소수 찾기, 소인수분해 - Easy + 각종 체

* sqMAX까지의 소수만 표시하므로, MAX까지 구하려면 i반복문 수정
시간복잡도: $\mathcal{O}(\log N \log N)$

```

//소수 판별
const int MAX = 10000000;
bool isPrime[MAX+2];
vector<int> prime;

```

```

void find_prime(){
    memset(isPrime, true, sizeof(isPrime));
    isPrime[0] = isPrime[1] = false;
    int sqMAX = sqrt(MAX)+1;
    for(long long i=2; i<=sqMAX; ++i)
        if(isPrime[i]){
            prime.push_back(i);
            for(long long j=i*i; j<=MAX; j+=i)
                isPrime[j] = false;
        }
}

//소인수분해
const int MAX = 10000000;
int isPrime[MAX+2];
void find_prime(){
    memset(isPrime, -1, sizeof(isPrime));
    int sqMAX = sqrt(MAX)+1;
    for(long long i=2; i<=sqMAX; ++i)
        if(isPrime[i] == -1)
            for(long long j=i*i; j<=MAX; j+=i)
                if(isPrime[j] == -1)
                    isPrime[j] = i;
}

void print_factor(int n){
    while(isPrime[n] != -1){
        cout << isPrime[n] << ' ';
        n /= isPrime[n];
    }
    cout << n;
}

// $1^N$ 의 약수의 개수 구하기  $O(N \log N)$ 
void num_of_divisors(int n, int ret[]) {
    for (int i = 1; i <= n; ++i)
        for (int j = i; j <= n; j += i)
            ret[j] += 1;
}

//오일러 피 함수  $O(n * \log \log n)$ 
//  $0 < x < n \ \&\& \ gcd(n, x) = 1$  인 x의 개수

```

```

void euler_phi(int n, int ret[]) {
    for (int i = 1; i <= n; ++i) ret[i] = i;
    for (int i = 2; i <= n; ++i)
        if (ret[i] == i)
            for (int j = i; j <= n; j += i)
                ret[j] -= ret[j] / i;
}

3.4 소수 찾기 - Fast (밀러-라빈)

*
시간복잡도:  $\mathcal{O}(\log N \log N)$ 

typedef long long ll;
typedef unsigned long long ull;
// calculate a*b % m, x86-64 only
ll large_mod_mul(ll a, ll b, ll m) {
    return ll((__int128)a*(__int128)b%m);
}

//  $|m| < 2^{62}$ , x86 available  $O(\log b)$ 
ll large_mod_mul(ll a, ll b, ll m) {
    a %= m; b %= m; ll r = 0, v = a;
    while (b) {
        if (b&1) r = (r + v) % m;
        b >>= 1;
        v = (v << 1) % m;
    }

// calculate  $n^k \% m$ 
ll modpow(ll n, ll k, ll m) {
    ll ret = 1;
    n %= m;
    while (k) {
        if (k & 1) ret = large_mod_mul(ret, n, m);
        n = large_mod_mul(n, n, m);
        k /= 2;
    }
    return ret;
}
bool test_witness(ull a, ull n, ull s) {

```

```

if (a >= n) a %= n;
if (a <= 1) return true;
ull d = n >> s;
ull x = modpow(a, d, n);
if (x == 1 || x == n-1) return true;
while (s-- > 1) {
    x = large_mod_mul(x, x, n);
    if (x == 1) return false;
    if (x == n-1) return true;
}
return false;
}

// test whether n is prime based on miller-rabin test O(logn*logn)
bool is_prime(ull n) {
    if (n == 2) return true;
    if (n < 2 || n % 2 == 0) return false;
    ull d = n >> 1, s = 1;
    for(; (d&1) == 0; s++) d >>= 1;
#define T(a) test_witness(a##ull, n, s)
    if (n < 4759123141ull) return T(2) && T(7) && T(61);
    return T(2) && T(325) && T(9375) && T(28178)
    && T(450775) && T(9780504) && T(1795265022);
#undef T
}

```

3.5 소인수분해 - Fast (플라드-로)

* 밀러 라빈 필요. 소인수분해 결과를 무작위 순서로 반환
 시간복잡도: $\mathcal{O}(N^{0.25} \log N)$

```

ull pollard_rho(ull n) {
    random_device rd;
    mt19937 gen(rd());
    uniform_int_distribution<ull> dis(1, n - 1);
    ull x = dis(gen);
    ull y = x;
    ull c = dis(gen);
    ull g = 1;
    while (g == 1) {
        x = (large_mod_mul(x, x, n) + c) % n;
        y = (large_mod_mul(y, y, n) + c) % n;

```

```

        y = (large_mod_mul(y, y, n) + c) % n;
        g = gcd(abs(x - y), n);
    }
    return g;
}

// integer factorization O(n^0.25 * logn)
void factorize(ull n, vector<ull>& f1) {
    if (n == 1) {
        return;
    }
    if (n % 2 == 0) {
        f1.push_back(2);
        factorize(n / 2, f1);
    }
    else if (is_prime(n)) {
        f1.push_back(n);
    }
    else {
        ull f = pollard_rho(n);
        factorize(f, f1);
        factorize(n / f, f1);
    }
}

```

3.6 FFT (고속 푸리에 변환)

* 배열 B에 0을 채워넣고 뒤집은 후, B의 꼬리가 i에 있을 때 A와 B를 곱한 값
 시간복잡도: $\mathcal{O}(N \log N)$

```

void fft(vector<complex<double>> &a, bool invert){
    int n = a.size();
    for (int i=1,j=0;i<n;i++){
        int bit = n >> 1;
        for (;j>=bit;bit>>=1) j -= bit;
        j += bit;
        if (i < j) swap(a[i],a[j]);
    }
    for (int len=2;len<=n;len<<=1){
        double ang = 2*M_PI/len*(invert?-1:1);
        complex<double> wlen(cos(ang),sin(ang));
        for (int i=0;i<n;i+=len){

```

```

        complex<double> w(1);
        for (int j=0;j<len/2;j++){
            complex<double> u = a[i+j], v = a[i+j+len/2]*w;
            a[i+j] = u+v;
            a[i+j+len/2] = u-v;
            w *= wlen;
        }
    }
    if (invert){
        for (int i=0;i<n;i++) a[i] /= n;
    }
}
vector<int> multiply(const vector<int> &a,const vector<int> &b){
    vector<int> res;
    vector <complex<double>> fa(a.begin(), a.end()), fb(b.begin(),
    b.end());
    int n = 1;
    while (n < max((int)a.size(),(int)b.size())) n <= 1;
    fa.resize(n); fb.resize(n);
    fft(fa,false); fft(fb,false);
    for (int i=0;i<n;i++) fa[i] *= fb[i];
    fft(fa,true);
    res.resize(n);
    for (int i=0;i<n;i++) res[i] =
    int(fa[i].real()+(fa[i].real()>0?0.5:-0.5));
    return res;
}

```

3.7 중국인의 나머지 정리 + 확장 유클리드

* $n[]$ 은 모두 소수여야 한다. a MOD (n의 모든 곱)을 구해준다.
 $x \equiv a \pmod{m}$, $x \equiv b \pmod{n}$ 을 만족시키는 x를 구하는 방법.
m과 n을 소인수분해한 후 소수의 제곱꼴의 합동식들로 각각 풀면. 이 때 특정 소수에
하여 모순이 생기면 불가능한 경우, 모든 소수에 대해서 모순이 생기지 않으면 전체
식을 CRT로 합치면 된다. 이제 $x \equiv x_1 \pmod{p^{k_1}}$ 과 $x \equiv x_2 \pmod{p^{k_2}}$ 가 모순이 생길
조건은 $k_1 \equiv k_2$ 라고 했을 때, $x_1 \not\equiv x_2 \pmod{p^{k_1}}$ 인 경우이다. 모순이 생기지 않았을
때 답을 구하려면 CRT로 합칠 때 $x \equiv x_2 \pmod{p^{k_2}}$ 만을 남기고 합쳐주면 된다.

// $ac + bd = \text{gcd}(a,b)$ 가 되는 (c,d)를 찾는다
pair<long long, long long> extended_gcd(long long a, long long b) {

```

    if(b == 0) return make_pair(1, 0);
    pair<long long, long long> t = extended_gcd(b, a % b);
    return make_pair(t.second, t.first - t.second * (a / b));
}

//  $ax = \text{gcd}(a,m) \pmod{m}$  이 되는 x를 찾는다
long long modinverse(long long a, long long m) {
    return (extended_gcd(a, m).first % m + m) % m;
}

//  $a[i] \% n[i]$  가 동일한 최소 x를 찾아줌
long long chinese_remainder(long long *a, long long *n, int size) {
    if (size == 1) return *a;
    long long tmp = modinverse(n[0], n[1]);
    long long tmp2 = (tmp * (a[1] - a[0])) % n[1] + n[1] % n[1];
    long long ora = a[1];
    long long tgcd = gcd(n[0], n[1]);
    a[1] = a[0] + n[0] / tgcd * tmp2;
    n[1] *= n[0] / tgcd;
    a[1] = ora;
    return ret;
}

// mod( $\pi(M)$ )에서  $x = B[i] \% M[pos+i]$  인 x를 구해준다
ll CRT_nocoprime(vector<ll> &B, vector<ll> &M) {
    ll retB=B[0], retM=M[0];
    for(int i=1; i<M.size(); ++i){
        ll m1=M[i], b1=B[i], m2=retM, b2=retB;
        ll g=gcd(m1, m2);
        if((b1-b2)%g) return -1;
        retM=m1*m2/g;
        ll m=m1/g;
        ll v=((b1-b2)/g*m+m)%m;
        ll w=(extended_gcd(m2/g%m, m1/g).first%m+m)%m;
        retB=b2+v*w%m*m2;
    }
    return retB;
}

```

4 문자열

4.1 KMP

* $\text{fail}[i] = \text{str}[0:i]$ 의 접두사이며 접미사인 최대 부분 문자열의 길이 (문자열 전체 제외)
시간복잡도: $\mathcal{O}(N + M)$

```
//text 중에 pat이 등장하는 시작 인덱스가 담긴 벡터를 리턴
vector<int> KMP(string &text, string &pat){
    vector<int> ans, fail(pat.size(), 0);
    //pat과 pat끼리 KMP를 실행한다
    for(int i=1, match=0; i<pat.size(); ++i){
        while(match && pat[i] != pat[match])
            match = fail[match-1];
        if(pat[i] == pat[match])
            fail[i] = ++match;
    }
    for(int i=0, match=0; i<text.size(); ++i){
        while(match && text[i] != pat[match])// 매칭 중에 불일치 발생
            match = fail[match-1]; //맞았었던 부분의 실패함수 적용
        if(text[i] == pat[match]) //한 글자 매칭 성공
            if(++match == pat.size()){//완전 매칭 성공
                ans.push_back(i-match+1);
                match = fail[match-1];
            }
    }
    return ans;
}
```

4.2 Rabin - Karp 해싱

```
vector<int> rabin_karp(string const& pat, string const& text) {
    const int B = 31;
    const int M = 1e9 + 9;
    int P = pat.size(), T = text.size();

    //P_pow[i] = (P^i)%M
    vector<ll> p_pow(max(P, T));
    p_pow[0] = 1;
    for (int i = 1; i < (int)p_pow.size(); i++)
```

```
p_pow[i] = (p_pow[i-1] * B) % M;

//H[i] = Hash(text[0...i-1])
vector<ll> H(T + 1, 0);
for (int i = 0; i < T; i++)
    H[i+1] = (H[i] + (text[i] - 'a' + 1) * p_pow[i]) % M;

// hv = Hash(pat)
ll hv = 0;
for (int i = 0; i < P; i++)
    hv = (hv + (pat[i] - 'a' + 1) * p_pow[i]) % M;

vector<int> matched;
for (int i = 0; i + P - 1 < T; i++) {
    ll cur_h = (H[i+P] + M - H[i]) % M;
    if (cur_h == hv * p_pow[i] % M)
        matched.push_back(i);
}
return matched;
}
```

4.3 Suffix Array + LCP

* suffix array - S에 대한 접미사 배열을 계산한 벡터를 반환
i번째 값은 사전순으로 i번째인 접미사의 시작 인덱스
LCP - 가장 긴 공통 접두사. 사전순으로 i번째 접미사와 i-1번째 접미사의 가장 긴 공통 접두사의 길이. i=0일때는 정의되지 않으므로 벡터의 크기가 1 작다
시간복잡도: $\mathcal{O}(N \log N)$

```
//NlogNlogN
vector<int> getSA(string &str){
    vector<int> ret(str.size()), rank(str.size()+1, -1),
    newRank(str.size()+1);
    for(int i=0; i<str.size(); ++i){
        ret[i] = i;
        rank[i] = str[i];
    }
    for(int pivot=1; pivot<str.size(); pivot<=1){
        sort(ret.begin(), ret.end(), [&](const int &a, const int &b){
            if(rank[a] != rank[b]) return rank[a]<rank[b];
            return rank[a+pivot] < rank[b+pivot];
        });
    }
}
```

```

});  

newRank[ret[0]] = 0;  

newRank[str.size()] = -1;  

for(int i=1; i<str.size(); ++i)  

    newRank[ret[i]] = newRank[ret[i-1]] + (rank[ret[i-1]] !=  

        rank[ret[i]] || rank[ret[i-1]+pivot] != rank[ret[i]+pivot]);  

rank = newRank;  

}  

return ret;  

}  
  

// calculates suffix array. O(n*logn)
typedef char T;
vector<int> suffix_array(const vector<T>& in) {
    int n = (int)in.size(), c = 0;
    vector<int> temp(n), pos2bckt(n), bckt(n), bpos(n), out(n);
    for (int i = 0; i < n; i++) out[i] = i;
    sort(out.begin(), out.end(), [&](int a, int b) { return in[a] < in[b]; });
    for (int i = 0; i < n; i++) {
        bckt[i] = c;
        if (i + 1 == n || in[out[i]] != in[out[i + 1]]) c++;
    }
    for (int h = 1; h < n && c < n; h <= 1) {
        for (int i = 0; i < n; i++) pos2bckt[out[i]] = bckt[i];
        for (int i = n - 1; i >= 0; i--) bpos[bckt[i]] = i;
        for (int i = 0; i < n; i++)
            if (out[i] >= n - h) temp[bpos[bckt[i]]++] = out[i];
        for (int i = 0; i < n; i++)
            if (out[i] >= h) temp[bpos[pos2bckt[out[i] - h]]++] = out[i] - h;
        c = 0;
        for (int i = 0; i + 1 < n; i++) {
            int a = (bckt[i] != bckt[i + 1]) || (temp[i] >= n - h)
                || (pos2bckt[temp[i + 1] + h] != pos2bckt[temp[i] + h]);
            bckt[i] = c;
            c += a;
        }
        bckt[n - 1] = c++;
        temp.swap(out);
    }
}

```

```

}  

return out;  

}  

// calculates lcp array. it needs suffix array & original sequence.  

O(n)
vector<int> lcp(const vector<T>& in, const vector<int>& sa) {
    int n = (int)in.size();
    if (n == 0) return vector<int>();
    vector<int> rank(n), height(n - 1);
    for (int i = 0; i < n; i++) rank[sa[i]] = i;
    for (int i = 0, h = 0; i < n; i++) {
        if (rank[i] == 0) continue;
        int j = sa[rank[i] - 1];
        while (i + h < n && j + h < n && in[i + h] == in[j + h]) h++;
        height[rank[i] - 1] = h;
        if (h > 0) h--;
    }
    return height;
}

```

4.4 모든 팰린드롬 찾기 - Manacher

* i번째 글자를 중심으로 하는 팰린드롬의 반지름을 구해준다

시간복잡도: $\mathcal{O}(N)$

```

vector<int> manacher(auto &s) {
    vector<int> p(s.size());
    int n = s.size(), r = -1, k = -1;
    for (int i=0; i<n; i++) {
        if (i<=r) p[i] = min(r-i, p[2*k-i]);
        while (i-p[i]-1>=0 && i+p[i]+1<n && s[i-p[i]-1] == s[i+p[i]+1])
            p[i]++;
        if (r > i+p[i]) r = i+p[i], k = i;
    }
    return p;
}

```

4.5 Z 알고리즘 - 접미사의 접두사

* i번째 접두사, 즉 $S[i:]$ 와 S의 최대 공통 접두사(문자열 전체 제외)
접두사이면서 접미사인 인 문제에 많이 쓰인다

시간복잡도: $\mathcal{O}(N)$

```
vector<int> Zalgorithm(string &S) {
    int n = S.size(), l=0, r=0;
    vector<int> Z(n);
    for (int i = 1; i < n; i++) {
        if (i > r) {
            l = r = i;
            while (r < n && S[r - 1] == S[r]) r++;
            Z[i] = r - l; r--;
        }
        else {
            int k = i - l;
            if (Z[k] < r - i + 1) Z[i] = Z[k];
            else {
                l = i;
                while (r < n && S[r - 1] == S[r]) r++;
                Z[i] = r - l; r--;
            }
        }
    }
    return Z;
}
```

4.6 트라이 - 문자열 전용 set

시간복잡도: $\mathcal{O}(|S|)$

```
const int MAXN = 100*1000 +1, MAXC = 10; // 트라이 노드 개수, 알파벳 개수
int piv, trie[MAXN][MAXC]; //트라이 번호 매기기, 트라이 노드
bool term[MAXN]; //N번째 트라이가 종말 노드인가?
void clear(){
    memset(trie, 0, sizeof(trie));
    memset(term, 0, sizeof(term));
    piv = 0;
}
```

```
void insert(string &i){
    int p = 0;
    for(char &j : i){
        if(!trie[p][j-'0']) trie[p][j-'0'] = ++piv;
        p = trie[p][j-'0'];
    }
    term[p] = 1;
}
bool query(string &s){
    int p = 0;
    for(char &i : s){
        if(!trie[p][i-'0']) return 0;
        p = trie[p][i-'0'];
        if(term[p]) return 1;
    }
    return 0;
}
```

4.7 아호 코라식 - 다중 문자열 KMP

시간복잡도: $\mathcal{O}(N + M)$

```
const int MAXN = 100*1000 +1, MAXC = 26; // 트라이 노드 개수, 알파벳 개수
int piv, trie[MAXN][MAXC], fail[MAXN]; //트라이 번호 매기기, 트라이, 실패 함수 링크
bool term[MAXN]; //N번째 트라이가 종말 노드인가?
void init(vector<string> &v){
    memset(trie, 0, sizeof(trie));
    memset(fail, 0, sizeof(fail));
    memset(term, 0, sizeof(term));
    piv = 0;
    for(string &i : v){
        int p = 0;
        for(char &j : i){
            if(!trie[p][j-'a']) trie[p][j-'a'] = ++piv;
            p = trie[p][j-'a'];
        }
        term[p] = 1;
    }
    queue<int> que;
```

```

for(int i=0; i<MAXC; i++){
    if(trie[0][i]) que.push(trie[0][i]);
}
while(!que.empty()){
    int x = que.front();
    que.pop();
    for(int i=0; i<MAXC; i++){
        if(trie[x][i]){
            int p = fail[x];
            while(p && !trie[p][i]) p = fail[p];
            p = trie[p][i];
            fail[trie[x][i]] = p;
            if(term[p]) term[trie[x][i]] = 1;
            que.push(trie[x][i]);
        }
    }
}
bool query(string &s){
    int p = 0;
    for(char &i : s){
        while(p && !trie[p][i-'a']) p = fail[p];
        p = trie[p][i-'a'];
        if(term[p]) return 1;
    }
    return 0;
}

```

5 기하

5.1 기본 기하 라이브러리 + CCW

```

const double eps = 1e-9; //오차 범위

inline int diff(double lhs, double rhs) { //오차 범위 내에서 대소관계
    if (lhs - eps < rhs && rhs < lhs + eps) return 0;
    return (lhs < rhs) ? -1 : 1;
}

inline bool is_between(double check, double a, double b) {

```

```

    if (a < b)
        return (a - eps < check && check < b + eps);
    else
        return (b - eps < check && check < a + eps);
}

struct Point {
    double x, y;
    bool operator==(const Point& rhs) const {
        return diff(x, rhs.x) == 0 && diff(y, rhs.y) == 0;
    }
    Point operator+(const Point& rhs) const {
        return Point{ x + rhs.x, y + rhs.y };
    }
    Point operator-(const Point& rhs) const {
        return Point{ x - rhs.x, y - rhs.y };
    }
    Point operator*(double t) const {
        return Point{ x * t, y * t };
    }
};

struct Line {
    Point pos, dir;
};

inline double inner(const Point& a, const Point& b) { //내적
    return a.x * b.x + a.y * b.y;
}
inline double outer(const Point& a, const Point& b) { //외적
    return a.x * b.y - a.y * b.x;
}

// 10면 ccw, -10면 cw, 0이면 일직선상
inline int ccw_line(const Line& line, const Point& point) {
    return diff(outer(line.dir, point - line.pos), 0);
}
inline int ccw(const Point& a, const Point& b, const Point& c) {
    return diff(outer(b - a, c - a), 0);
}

```

```

//점과 점사이 거리
inline double dist(const Point& a, const Point& b) {
    return sqrt(inner(a - b, a - b));
}

inline double dist(const Line& line, const Point& point, bool
segment = false) {
    double c1 = inner(point - line.pos, line.dir);
    if (segment && diff(c1, 0) <= 0) return dist(line.pos, point);
    double c2 = inner(line.dir, line.dir);
    if (segment && diff(c2, c1) <= 0) return dist(line.pos +
line.dir, point);
    return dist(line.pos + line.dir * (c1 / c2), point);
}

```

5.2 볼록 껍질(컨벡스 헬)

* 가장 왼쪽, 같을시 아래쪽 점을 시작으로 시계방향으로 탐색. 일직선의 점을 남기고 싶으면, 등호를 뺀다
시간복잡도: $\mathcal{O}(N \log N)$

```

vector<Point> convex_hull(vector<Point>& dat) {
    if (dat.size() <= 2) return dat;
    vector<Point> upper, lower;
    sort(dat.begin(), dat.end(), [](const Point& a, const Point& b) {
        return (a.x == b.x) ? a.y < b.y : a.x < b.x;
    });
    for (const auto& p : dat) {
        while (upper.size() >= 2 && ccw(*++upper.rbegin(),
*upper.rbegin(), p)
        >= 0) upper.pop_back();
        while (lower.size() >= 2 && ccw(*++lower.rbegin(),
*lower.rbegin(), p)
        <= 0) lower.pop_back();
        upper.emplace_back(p);
        lower.emplace_back(p);
    }
    upper.insert(upper.end(), ++lower.rbegin(), --lower.rend());
    return upper;
}

```

```

//IF int or long long
#define xx first
#define yy second
typedef long long ll;
typedef pair<ll, ll> point;

inline ll outer(const point a, const point b) { //외적
    return a.xx * b.yy - a.yy * b.xx;
}

inline ll ccw(const point& a, const point& b, const point& c) {
    auto tmp = outer({b.xx - a.xx, b.yy - a.yy}, {c.xx - a.xx, c.yy -
a.yy});
    return (tmp ? (tmp < 0 ? -1 : 1) : 0);
}

//IF double
struct Point + operator -
diff, outer , ccw

```

5.3 회전하는 캘리퍼스 - 볼록 껍질에서 가장 먼 점

* 볼록 껍질의 모든 점에 대해 가장 먼 점을 구해준다
시간복잡도: $\mathcal{O}(N)$

```

//+ 정수 컨벡스 헬
void farthest_pairs(vector<point>& pt) {
    sort(pt.begin(), pt.end(), [](const point& a, const point& b) {
        return (a.xx == b.xx) ? a.yy < b.yy : a.xx < b.xx;
    });
    vector<point> up, lo;
    for (const auto& p : pt) {
        while (up.size() >= 2 && ccw(*++up.rbegin(), *up.rbegin(), p) >=
0) up.
        pop_back();
        while (lo.size() >= 2 && ccw(*++lo.rbegin(), *lo.rbegin(), p) <=
0) lo.
        pop_back();
        up.emplace_back(p);
        lo.emplace_back(p);
    }
}

```

```

    }
    for (int i = 0, j = (int)lo.size() - 1; i + 1 < up.size() || j >
0; ) {
        get_pair(up[i], lo[j]); // DO WHAT YOU WANT
        if (i + 1 == up.size())
            --j;
        else if (j == 0)
            ++i;
        else if ((long long)(up[i + 1].yy - up[i].yy) * (lo[j].xx - lo[j -
1].xx)
            > (long long)(up[i + 1].xx - up[i].xx) * (lo[j].yy - lo[j -
1].yy))
            ++i;
        else
            --j;
    }
}

```

6 기타

6.1 위상 정렬

* indegree[] 배열은 미리 계산 되있어야 한다.

시간복잡도: $\mathcal{O}(N)$

```

const int MAX;
int n, indegree[MAX], result[MAX];
vector<int> adj[MAX];
bool topological_sort(){
    queue<int> Q;
    for(int i=0; i<n; ++i)
        if(!indegree[i]) Q.push(i);
    for(int i=0; i<n; ++i){
        if(Q.empty()) return false;
        int here = Q.front();
        Q.pop();
        result[i] = here;
        for(int &there : adj[here])
            if(!(--indegree[there]))
                Q.push(there);
    }
}

```

```

    }
    return true;
}

```

6.2 Mo's Algorithm + Sqrt Decomposition

시간복잡도: $\mathcal{O}(N + Q)\sqrt{N}$

```

typedef pair<int,int> pii;
int n, ans[MAX];
vector<pair<pii,int>> query;
inline void add(int k){//현 구간에 k를 하나 추가해준다.
    //do something : ex) if(++cnt[k] == 1) ++ret;
}
inline void pop(int k){//현 구간에서 k를 하나 제거해준다.
    //do something : ex) if(--cnt[k] == 0) --ret;
}
void MOquery(){
    int sqrtN = sqrt(n);
    sort(query.begin(), query.end(), [] (auto &a, auto &b){
        //구간 [L,R]들을 정렬하자. (sqrt decomposition)
        //1. L/sqrt(N) 기준으로 오름차순 정렬
        //2. 같다면 R 기준으로 오름차순 정렬
        int Ln = a.first.first/sqrtN, Rn = b.first.first/sqrtN;
        if(Ln != Rn)
            return Ln < Rn;
        return a.first.second < b.first.second;
    });

    // [0,0]부터 시작해 스위핑 해주자.
    int L = 0, R = 0;
    add(arr[0]);
    for(auto &P : query){
        int nL = P.first.first, nR = P.first.second;
        //1. L을 nL로 바꿔주기.
        if(L < nL){ //오른쪽으로 밀면서 [L, nL) 제거
            for(int i=L; i<nL; ++i)
                pop(arr[i]);
        }
        else{ //왼쪽으로 밀면서 추가 [nL, L) 추가
            for(int i=L-1; i>=nL; --i)

```

```

        add(arr[i]);
    }
    //2. R을 nR로 바꿔주기.
    if(R < nR){//오른쪽으로 밀면서 (R, nR] 추가
        for(int i=R+1; i<=nR; ++i)
            add(arr[i]);
    }
    else{//왼쪽으로 밀면서 (nR, R] 제거
        for(int i=R; i>nR; --i)
            pop(arr[i]);
    }
    //구간, 쿼리의 답 업데이트
    L = nL;
    R = nR;
    ans[P.second] = ret;
}
}

```

6.3 방향 그래프 사이클 판정

- * visit을 -1로 초기화
- 시간복잡도: $\mathcal{O}(V + E)$

```

int visit[MAX] = {};
bool haveCycle(int here){
    if(visit[here])
        return visit[here] == -1;
    visit[here] = -1;
    for(int &there : adj[here])
        if(haveCycle(there)) return true;
    visit[here] = 1;
    return false;
}

```

6.4 IO Trick

```

//FAST I/O
namespace fio {
    const int BSIZE = 524288;
    char buffer[BSIZE];
    int p = BSIZE;
}

```

```

inline char readChar() {
    if(p == BSIZE) {
        fread(buffer, 1, BSIZE, stdin);
        p = 0;
    }
    return buffer[p++];
}

int readInt() {
    char c = readChar();
    while ((c < '0' || c > '9') && c != '-') {
        c = readChar();
    }
    int ret = 0; bool neg = c == '-';
    if (neg) c = readChar();
    while (c >= '0' && c <= '9') {
        ret = ret * 10 + c - '0';
        c = readChar();
    }
    return neg ? -ret : ret;
}

```

```

//string stream
int num;
stringstream stream;
string string = "1 2 3 a 4";
stream.str(string);
while( stream >> num )
    cout << num << ' ';//1 2 3

```

6.5 삼분 탐색

- * 순증가-순감소 혹은 순감소 - 순증가 하는 볼록 함수

```

double ternary (double S, double E) {
    for(int i=0;i<100;i++) {
        double L = (2*S+E)/3, R = (S+2*E)/3;
        if(f(L) < f(R)) S = L;
        else E = R;
    }
    return value(S);
}

```

```

}

//대안 : 기울기로 이분 탐색
int ternary (int S, int E) {
    int ret = 0x7fffffff;
    while(S+3<E) {
        int L = (2*S+E)/3, R = (S+2*E)/3;
        if(f(L) < f(R)) S = L;
        else E = R;
    }
    for(int i=S;i<=E;i++) {
        ret = min(ret, f(i));
    }
    return ret;
}

```

6.6 Convex Hull Trick

```

//dp[i] = min{b[j]*a[i] + c[j]}+d[i]; j<i, b[j]는 증가/감소
const int MAX=100000;
struct ConvexHullTrick{
    int s=0, e=0, idx[MAX];
    pll deq[MAX]; //first * x + second
    double cross(int a, int b) {
        return 1.0 * (deq[a].yy - deq[b].yy) / (deq[b].xx - deq[a].xx);
    }
    void insert(pll v,int line_idx){//current line , i
        deq[e] = v;
        idx[e] = line_idx;
        while(s+1<e && cross(e - 2, e - 1) > cross(e - 1, e)){
            deq[e-1] = deq[e];
            idx[e-1] = idx[e];
            e--;
        }
        e++;
    }
    ll query(long long x){ // query가 증가수열일 경우
        //그래프 = ↗ 꼴 (>=), ↛ 꼴 (<=)
        while(s+1<e && deq[s+1].yy - deq[s].yy <= x * (deq[s].xx -
        deq[s+1].xx))

```

```

        s++;
        return deq[s].xx*x+deq[s].yy; //idx[s]
    }
    ll query2(long long x){ // 일반 query
        int l = 0, r = e - 1;
        while (l != r) {
            int m = (l + r) / 2;
            if (cross(m, m + 1) <= x) l = m + 1;
            else r = m;
        }
        return deq[l].xx*x+deq[l].yy; //idx[l]
    }
    void clear(){s = e = 0;}
} CHT;

```

6.7 Berlekamp-Massey - 선형 점화식 DP 치트키

* berlekampmassey : n개의 DP 결과값이 주어졌을 때, Berlekamp-Massey 알고리즘을 사용하여 해당 DP의 점화식을 찾아주는 루틴. 점화식의 크기가 k면, $2k$ 개 이상의 항이 있어야 답을 찾음이 보장된다.

getnth : 주어진 DP 점화식을 사용하여 n번째 항을 구하는 루틴.

시간복잡도: $\mathcal{O}(nk + n \log n)$, and get nth $\mathcal{O}(k^2 \log n)$

```

const int mod = 998244353;
ll ipow(ll x, ll p){
    ll ret = 1, piv = x;
    while(p){
        if(p & 1) ret = ret * piv % mod;
        piv = piv * piv % mod;
        p >>= 1;
    }
    return ret;
}
vector<int> berlekamp_massey(vector<int> x){
    vector<int> ls, cur;
    int lf, ld;
    for(int i=0; i<x.size(); i++){
        ll t = 0;
        for(int j=0; j<cur.size(); j++){
            t = (t + 1ll * x[i-j-1] * cur[j]) % mod;
        }

```

```

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

int get_nth(vector<int> rec, vector<int> dp, ll n){
    int m = rec.size();
    vector<int> s(m), t(m);
    s[0] = 1;
    if(m != 1) t[1] = 1;
    else t[0] = rec[0];
    auto mul = [&rec](vector<int> v, vector<int> w){
        int m = v.size();
        vector<int> t(2 * m);
        for(int j=0; j<m; j++){
            for(int k=0; k<m; k++){
                t[j+k] += 111 * v[j] * w[k] % mod;
                if(t[j+k] >= mod) t[j+k] -= mod;
            }
        }
        for(int j=2*m-1; j>=m; j--){
            for(int k=1; k<=m; k++){
                t[j+k] += 111 * t[j] * rec[k-1] % mod;
                if(t[j+k] >= mod) t[j+k] -= mod;
            }
        }
        t.resize(m);
        return t;
    };
    while(n){
        if(n & 1) s = mul(s, t);
        t = mul(t, t);
        n >>= 1;
    }
    ll ret = 0;
    for(int i=0; i<m; i++) ret += 111 * s[i] * dp[i] % mod;
    return ret % mod;
}

int guess_nth_term(vector<int> x, ll n){
    if(n < x.size()) return x[n];
    vector<int> v = berlekamp_massey(x);
    if(v.empty()) return 0;
    return get_nth(v, x, n);
}

```

6.8 알쓸신잡

- * *카탈란 수
1, 1, 2, 5, 14, 42, 132, 429, 1430, 4862, 16796, 58786, 208012, 742900, 2674440, 9694845, 35357670, 129644790,
 $C_0 = 1$
 $C_n + 1 = 2(2n + 1)/(n + 2) * C_n = \text{return binomial}(n * 2, n) / (n + 1);$
 - 길이가 $2n$ 인 올바른 팔호 수식의 수
 - $n + 1$ 개의 리프를 가진 풀 바이너리 트리의 수
 - $n + 2$ 각형을 n 개의 삼각형으로 나누는 방법의 수
- * Burnside's Lemma
 경우의 수를 세는데, 특정 transform operation(회전, 반사, ..) 해서 같은 경우들은 하나로 친다. 전체 경우의 수는?
 - 각 operation마다 이 operation을 했을 때 변하지 않는 경우의 수를 센다 (단, “아무것도 하지 않는다”라는 operation도 있어야 함!)
 - 전체 경우의 수를 더한 후, operation의 수로 나눈다. (답이 맞다면 항상 나누어

떨어져야 한다)

*알고리즘 게임

Nim Game의 해법 : 각 더미의 돌의 개수를 모두 XOR했을 때 0 이 아니면 첫번째, 0 이면 두번째 플레이어가 승리.

Grundy Number : 어떤 상황의 Grundy Number는, 가능한 다음 상황들의 Grundy Number를 모두 모은 다음, 그 집합에 포함 되지 않는 가장 작은 수가 현재 state의 Grundy Number가 된다. 만약 다음 state가 독립된 여러개의 state들로 나뉠 경우, 각각의 state의 Grundy Number의 XOR 합을 생각한다.

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

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

* Pick's Theorem

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

6.9 자주 쓰는데 헷갈리는 것들

512MB = int * 17,179,869,184

```
//strtok 파싱
string input = "how to,split!string";
char *token = strtok(input, " ,!");
while(token){
    cout << token << '\n';
    token = strtok(0, " ,!");
}

//cout 자리수 설정
cout.precision(X);
cout << fixed;

//비트마스크
__builtin_popcount(bits) // 켜져있는 비트의 개수
__builtin_ctz(bits) //최하위 비트의 인덱스
```

```
(bits & -bits) //최하위 비트의 실제 값
for(int i=bits; i; i=(i-1)&bits) //모든 부분집합 순회

bitset<64> B(42); // ...00101010
string S = B.to_string();
bitset<64> B2("101010");
cout << B2.to_ullong(); // 42

//신발끈 정리 - 변이 교차하지 않는 다각형의 넓이
for(int i=0; i<n; ++i)
    area += X[i]*Y[(i+1)%len] - X[i]*Y[(i+len-1)%len];
return abs(area/2);

// comparator overload
auto cmp = [] (seg a, seg b){return a.func() < b.func(); };
set<seg, decltype(cmp)> s(cmp);
map<seg, int, decltype(cmp)> mp(cmp);
priority_queue<seg, vector<seg>, decltype(cmp)> pq(cmp); // max heap

// hash func overload
struct point{
    int x, y;
    bool operator==(const point &p) const{ return x == p.x && y == p.y; }

};

struct hasher {size_t operator()(const point &p) const{ return p.x * 2 + p.y * 3; }
};

unordered_map<point, int, hasher> hsh;

//파일 입출력
#ifndef ONLINE_JUDGE freopen("a.in","r",stdin); #endif
```