그래프

심화 탐색

LCA Algorithm (최소 공통 조상)

```
vector<int> graph[MAX];

bool visited[MAX];
int Depth[MAX], parent[MAX][LOG];

void DFS(int curr, int depth) {
    visited[curr] = true;
    Depth[curr] = depth;

    for (int i = 0; i < graph[curr].size(); i++) {
        int next_node = graph[curr][i];
        if (visited[next_node]) continue;
        parent[next_node][0] = curr;
        DFS(next_node, depth + 1);
    }
}

void set_parent() {
    DFS(1, 0);
    for (int i = 1; i < LOG; i++) {
        for (int j = 1; j <= N; j++) {
            parent[j][i] = parent[parent[j][i-1]][i-1];
        }
    }
}

int Fast_LCA(int x, int y) {
    if (Depth[x] > Depth[y]) swap(x, y);

    for (int i = LOG - 1 ; i >= 0; i--) {
        if (Depth[y] - Depth[x] >= (1 << i)) {
            y = parent[y][i];
        }
    }

    if ( x == y ) return x;

    for (int i = LOG - 1; i >= 0; i--) {
        if (parent[x][i] != parent[y][i]) {
            x = parent[x][i];
            y = parent[y][i];
        }
```

```
    }
    return parent[x][0];
}
```

Network Flow (최대유량)

```
int networkFlow(int source, int destination) {
    memset(flow, 0, sizeof(flow));

    int totalFlow = 0;
    int amount = INF ;

    while(1) {
        vector<int> parent(MAX,-1);
        queue<int> q;
        parent[source] = source ;
        q.push(source);

        while(!q.empty() && parent[destination] == -1) {
            int here = q.front();
            q.pop();

            for (int there = 0 ; there < MAX ; there++) {
                if (capacity[here][there] - flow[here][there] > 0 && parent[there] ==
-1) {
                    q.push(there);
                    parent[there] = here;
                }
            }
        }

        if (parent[destination] == -1) break;

        amount = INF ;

        for(int p = destination; p != source ; p = parent[p]) {
            amount = min(amount, capacity[parent[p]][p] - flow[parent[p]][p]);
        }

        for(int p = destination; p != source ; p = parent[p]) {
            flow[parent[p]][p] += amount ;
            flow[p][parent[p]] -= amount ;
        }

        totalFlow += amount ;
    }
```

```
    return totalFlow ;
}
```

Critical Path (임계경로)

```
// graph[현재노드] = vector<pair<다음노드, 거리>>
vector < vector < pair <int, int> > > graph(n+1);

int get_critical_path_length(int start, int end, int *dis){
    queue <int> q;
    q.push(start);
    while(!q.empty()){
        int curr = q.front();
        q.pop();
        if(curr == end) continue;
        for(int i=0; i < graph[curr].size(); i++){
            int des = graph[curr][i].first;
            if(dis[des] < dis[curr] + graph[curr][i].second) {
                dis[des] = dis[curr] + graph[curr][i].second;
                q.push(des);
            }
        }
        return dis[end]
    }
```

Articulation Point

```

```

Bipartitie

```
int V, E;
vector <int> adj_list[100];
int colored[1000], visit[1000];
bool flag = true;

void dfs(int u, int color) {

    visit[u] = 1;
```

```
    colored[u] = color;

    for (int i = 0; i < adj_list[u].size(); i++) {
        int v = adj_list[u][i];

        // 방문 안한 경우 방문해서 색깔 칠해주기
        if (visit[v] == 0) {
            dfs(v, !color);
        }

        // 방문 했는데 인접한 정점인데 색깔이 같으면 false
        else {
            if (color == colored[v])  {
                flag = false;
                return;
            }
        }
    }
}

void bipartitleVerify() {
  for (int i = 0; i < V; i++) {
      if (!visit[i]) dfs(i, 0);
  }
}
```

Topological sort

```
```

SCC

```
```

Cycle - Checking

```
```

MST

Kruskal Algorithm

```
//edge 는 {가중치, {노드1, 노드2}} 의 형태로 저장

int find (int x) {
  if (Parent[x] == x) return x;
  return Parent[x] = find(Parent[x]);
}


void merge (int x, int y) {
  x = find(x);     // 먼저 x의 부모를 찾고
  y = find(y);     // y의 부모를 찾아준다.

  if (x != y) Parent[y] = x;
}


int kruskal(vector < pair <int, pair <int, int> > > edge){
    sort(edge.begin(),edge.end());
    int count = 0;
    int index= 0;
    int ans = 0;
    while (count != v-1) {
        int weight = edge[i].first;
        int node1 = find(edge[i].second.first);
        int node2 = find(edge[i].second.second);
        if (node1 == node2) continue;
        ans += curr.w;
        merge(node1, node2);
        index++;
        count++;
    }
    return ans;
}
```

Prim Algorithm

```
void Prim_Using_Heap() {
    priority_queue< pair <int, int> > PQ;

    for (int i = 0; i < Cost[1].size(); i++) {
        int Next = Cost[1][i].first;
```

```
        int Distance = Cost[1][i].second;
        PQ.push(make_pair(-Distance, Next));
    }

    Visit[1] = true;
    while (!PQ.empty()) {
        int Distance = -PQ.top().first;
        int Cur = PQ.top().second;
        PQ.pop();

        if (Visit[Cur] == false) {
            Visit[Cur] = true;
            Answer = Answer + Distance;
            for (int i = 0; i < Cost[Cur].size(); i++) {
                int nDistance = Cost[Cur][i].second;
                int Next = Cost[Cur][i].first;
                if (Visit[Next] == false) PQ.push(make_pair(-nDistance, Next));
            }
        }
    }
}
```

최단경로

Floyd-Warshall Algorithm (Shortest Path)

```
#define INF 99999999

int d[1000][1000];
int V, E;

void floyd() {
    // k = 중간정점
    for (int k = 1; k <= V; k++) {
        for (int i = 1; i <= V; i++) {
            for (int j = 1; j <= V; j++) {
                //if (i == k || j == k) continue;
                if (d[i][j] > d[i][k] + d[k][j])
                    d[i][j] = d[i][k] + d[k][j];
            }
        }
    }
}
```

Dijkstra Algorithm

```
#define PAIR pair<int, int>

int d[1000], p[1000];

vector <PAIR> adj_list[1000];


void dijsktra(int V, int E, int start) {

    for (int i = 1; i <= V; i++) {
        d[i] = 9999999;
        p[i] = -1;
    }

    d[start] = 0, p[start] = 0;

    set <int> S;
    priority_queue <PAIR, vector<PAIR>, greater<PAIR> > Q;

    for (int i = 1; i <= V; i++) {
        Q.push(make_pair(d[i], i));
    }

    while (!Q.empty()) {
        int u = Q.top().second;
        Q.pop();
        S.insert(u);

        for (int i = 0; i < adj_list[u].size(); i++) {
            int v = adj_list[u][i].first;
            int w = adj_list[u][i].second;

            if (d[v] > d[u] + w) {
                d[v] = d[u] + w;
                p[v] = u;
                Q.push(make_pair(d[v], v));
            }
        }
    }
}
```

Bellman-Ford Algorithm

```
#define MAX 99999999

vector < tuple <int, int, int> > Edge; // 0: src  1: dst  2: weight

int V, E, start;
int d[501]; // distance
int p[501]; // parents node


bool BellmanFord() {
    // init
    for (int v = 1; v <= V; v++) d[v] = MAX;

    // relaxation
    d[start] = 0;
    for (int i = 1; i < V; i++) {
        for (int j = 0; j < E; j++) {
            int u = get<0>(Edge[j]);
            int v = get<1>(Edge[j]);
            int w = get<2>(Edge[j]);

            if (d[u] != MAX && d[v] > d[u] + w) {
                d[v] = d[u] + w;
              p[v] = u;
            }
        }
    }

    // check negative weighted cycle
    for (int i = 0; i < E; i++) {
         int u = get<0>(Edge[i]);
         int v = get<1>(Edge[i]);
         int w = get<2>(Edge[i]);

         if (d[u] != INF && d[v] > d[u] + w)
            return false;
    }
    return true;

}
```

문자열

문자열 검색

KMP Algorithm

```
int fail[1000];
vector <int> result;

void getFailFunc(string P) {
  int M = 0;
  while (P[M]) M++;

  for (int i = 1, j = 0; i < M; i++) {
    while (j > 0 && P[i] != P[j])
            j = fail[j - 1];

    if (P[i] == P[j]) fail[i] = ++j;
    else fail[i] = 0;

  }
}

void KMP(string text, string pattern) {
    int t_len = text.length();
    int p_len = pattern.length();
    int begin = 0, m = 0;

    while(begin <= t_len - p_len) {
        if (m < t_len && text[begin+m] == pattern[m]){
            m++;.
            if(m == p_len) result.push_back(begin);
        }

        else {
            if(m == 0) begin++;
            else {
                begin += (m - fail[m-1]);
                m = fail[m-1];
            }
        }
    }
}
```

TRIE 구조

```
struct TRIE {
    bool Finish;
    TRIE *Node[26];
    TRIE() {
        Finish = false;
        for (int i = 0; i < 26; i++) Node[i] = NULL;
    }
}

void Insert(char *Str) {
    if (*Str == NULL) {
        Finish = true;
        return;
    }

    int Cur = *Str - 'A';
    if (Node[Cur] == NULL) Node[Cur] = new TRIE();
    Node[Cur]->Insert(Str + 1);
}

bool Find(char *Str) {
    if (*Str == NULL) {
        if (Finish == true) return true;
        return false;
    }
    int Cur = *Str - 'A';
    if (Node[Cur] == NULL) return false;
    return Node[Cur]->Find(Str + 1);
}
```

아호 코라식

```
```

펠린드롬

Manacher's Algorithm

```
#define MAXN 100001

int d[10000]; // 팰린드롬의 반경
string s; // 문자열
int n; // 문자열의 길이
int r, center; // 맨 끝의 위치, 중간의 위치

void Manacher() {
  r = center = -1;
  n = s.length();

  // even palindrome
  for (int i = n - 1; i >= 0; i--) {
    s[(i << 1)+1] = s[i];
    s[i << 1] = '#';
  }

  n <<= 1;
  s[n++] = '#';

  for (int i = 0; i < n; i++) {
    if (r >= i) d[i] = min(r - i, d[center*2 - i]); // 작은 쪽을 넣어준다.
    else d[i] = 0;

    while (i+d[i]+1 < n && i-d[i]-1 >= 0 && s[i+d[i]+1] == s[i-d[i]-1])
      p[i]++; // 같으면 증가

    if (i + d[i] > r) { // 끝지점을 넘어서면 그때마다 갱신
      r = i + d[i];
      center = i;
    }
  }
}
```

수학

정수론

에라토스테네스의 체

```
bool prime[10000000];

void sieve(int num) {
  for (int i = 2; i <= num; i++) {
    if (!prime[i]) {
      for (int j = i*2; j <= num; j+= i)
        prime[j] = true;
    }
  }
}
```

이항계수

```
long long memo[n+1][r+1];

void binomial_coefficient (int n, int k) {

  // nC0, nCn = 1로 초기화
  for (int i = 0; i < n; i++) {
    memo[i][0] = 1;
    memo[i][i] = 1;
  }

  // nCk = n-1Ck + n-1Ck-1
  for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= k; j++) {
      memo[i][j] = memo[i-1][j] + memo[i-1][j-1];
    }
  }
}
```

페르마의 소정리

```
```

기하

Plane sweep

```
```

Convex Hall

```
```

CCW

```
```

검색

트리

Disjoint Set (Union Find)

```
int parent[100];

for (int i = 0; i< 100; i++)
  parent[i] = -1;

int find(int x){
  if (parent[x] < 0){
    return x;
  }

  else{
    int y = find(parent[x]);
    parent[x] = y;
    return y;
  }
}

void union(int x, int y){
```

```
  x = find(x);
  y = find(y);

  if (x == y)
    return;

  // parent[x], parent[y] 값은 음수이므로 값이 작은 경우가 더 높이가 큰 노드이다.
  if (parent[x] < parent[y]){
    parent[x] += parent[y];
    parent[y] = x;
  }
  else {
    parent[y] += parent[x];
    parent[x] = y;
  }
}
```

Segment Tree (with lazy propagation)

```
vector <int> arr;
int tree[1000];

// start: 시작 인덱스, end: 끝 인덱스
// index: 구간 합을 수정하고자 하는 노드
// left, right: 구간 합을 구하고자 하는 범위
// new_value: 수정할 값

int init(int start, int end, int node) {
  if(start == end) return tree[node] = arr[start];
  int mid = (start + end) / 2;

  return tree[node] = init(start, mid, node * 2) + init(mid + 1, end, node * 2 + 1); }

int sum(int start, int end, int node, int left, int right) {
  if(left > end || right < start) return 0;
  if(left <= start && end <= right) return tree[node];

  int mid = (start + end) / 2;
  return sum(start, mid, node * 2, left, right) + sum(mid + 1, end, node * 2 + 1, left,
right);

}

void update(int start, int end, int node, int index, int new_value) {
  if(index < start || index > end) return;

  tree[node] += new_value;
```

```
  if (start == end) return; int mid = (start + end) / 2;

  update(start, mid, node * 2, index, new_value);
  update(mid + 1, end, node * 2 + 1, index, new_value);

}
```

Panwick Tree (Binary Index Tree)

```
int Fenwick_Tree[1001];


void Update(int Idx, int Value) {
    while (Idx < Fenwick_Tree.size()) {
        Fenwick_Tree[Idx] = Fenwick_Tree[Idx] + Value;
        Idx = Idx + (Idx & -Idx);
    }
}


int Sum(int Idx) {
    long long Result = 0;
    while (Idx > 0) {
        Result = Result + Fenwick_Tree[Idx];
        Idx = Idx - (Idx & -Idx);
    }

    return Result;
}

void Make_PenwickTree() {
    for (int i = 1; i <= N; i++)
        Update(i, Fenwick_Tree[i]);
}
```

배열

Parametric search

Sliding Window

```
void sliding_window() {
  vector <int> arr;
  int N, k;

  int window_sum = 0;
  int max_sum = 0;
  int window_start = 0;

  for (int window_end = 0; window_end < arr.size(); window_end++) {
    window_sum += arr[window_end];

  if (window_end >= k-1) {
    max_sum = max(max_sum, window_sum);
    window_sum -= arr[window_start];
    window_start += 1;
    }

  }

  return max_sum;

}
```

Two pointer

```
#include <iostream>

using namespace std;

int arr[10001];

int twopointer() {
    int N, M;
    int answer = 0;

    int start = 0;
    int end = 0;
    int partial_sum = 0;

    while (end <= N) {
        if (partial_sum >= M) {
```

```
            partial_sum -= arr[start++];

        }
        else if (partial_sum < M)
            partial_sum += arr[end++];

        // 구간합 == M , 구간합 > M 일때 수정해서 적용
        if (partial_sum >= M)
            answer++;
    }

    return answer;
}
```

DP

Knapsack

```

```

LCS

```
int memo[1001][1001];

void LCS() {
    string s1, s2;

    // LCS Algorithm
    s1 = "0" + s1;
    s2 = "0" + s2;
    for (int i = 0; i < s1.length(); i++) {
        for (int j = 0; j < s2.length(); j++) {
            if (i == 0 || j == 0) {
                memo[i][j] = 0;
                continue;
            }

            if (s1[i] == s2[j])  memo[i][j] = memo[i-1][j-1] + 1;
            else  memo[i][j] = max(memo[i-1][j], memo[i][j-1]);

        }
    }
```

```
    // Find LCS string
    string LCS;
    int x = s1.length()-1;
    int y = s2.length()-1;
    int LCS_length = memo[x][y];

    while (LCS_length != 0) {
        if (memo[x-1][y] == LCS_length) x--;
        else if (memo[x][y-1] == LCS_length)  y--;
        else {
            LCS.push_back(s1[x] == s2[x] ? s1[x] : s2[y]);
            x--, y--;
            LCS_length--;
        }
    }
}
```

LIS

```
int array[1000]; // 인덱스마다 각 입력값
int dp[1000]; // 인덱스마다 각 증가 수열의 길이
int max = 0;

for(int i = 0; i < N; i++) {
    dp[i] = 1;
    // i 를 기준으로 인덱스 0 에서부터 i-1까지 체크한다
    // 길이를 기준
    for(int j = 0; j < i; j++) {
        if (array[i] > array[j] && dp[j] + 1 > dp[i]) {
            // 증가 수열
            dp[i] = dp[j] + 1;
        }
    }

    if (max < dp[i]) max = dp[i];
}
```

MCM

```
int N;
int matrics[1001][2];
int d[2002];
int M[1001][1001];
```

```
int S[1001][1001];


void MCM() {
    // Make d[i] array
    d[0] = matrics[0][0];
    for (int i = 0; i < N; i++) {
        d[i+1] = matrics[i][1];
    }

    // Memoization
    // if (i,i) = 0
    for (int i = 1; i <= N; i++)
        M[i][i] = 0;

    // else
    for (int r = 2; r <= N; r++) { // r is chain length
        for (int i = 1; i <= N-r+1; i++) {
            int j = i+r-1;
            M[i][j] = INT_MAX;
            for (int k = i; k < j; k++) { // k is diverging point
                if (M[i][j] > M[i][k] + M[k+1][j] + d[i-1]*d[k]*d[j]) {
                    M[i][j] = M[i][k] + M[k+1][j] + d[i-1]*d[k]*d[j];
                    S[i][j] = k;
                }
            }
        }
    }
}
```

분할정복

power

```
```

행렬 곱셈

```
```

정렬

Quick Sort

```
#define MAX 100

int Arr[MAX];
bool Flag[10000];


int Partition(int Left, int Right) {
    int Pivot_Value = Left;
    int Store_Index = Pivot_Value;

    for (int i = Left + 1; i <= Right; i++) {
        if (Arr[Pivot_Value] > Arr[i]) {
            Store_Index++;
            swap(Arr[i], Arr[Store_Index]);
        }
    }
    swap(Arr[Pivot_Value], Arr[Store_Index]);
    Pivot_Value = Store_Index;

    return Pivot_Value;
}

void QuickSort(int Left, int Right) {
    if (Left < Right) {
        int Pivot = Partition(Left, Right);
        QuickSort(Left, Pivot - 1);
        QuickSort(Pivot + 1, Right);
    }
}
```

Merge Sort

```
#define MAX 100

int Arr[MAX];
int Temp_Arr[MAX];
bool Flag[10000];

void Merge(int Start, int Mid, int End) {
    for (int i = Start; i <= End; i++) {
```

```
        Temp_Arr[i] = Arr[i];
    }

    int Left_Part = Start;
    int Right_Part = Mid + 1;
    int Idx = Start;

    while (Left_Part <= Mid && Right_Part <= End) {
        if (Temp_Arr[Left_Part] <= Temp_Arr[Right_Part]) {
            Arr[Idx] = Temp_Arr[Left_Part];
            Left_Part++;
        }

        else {
            Arr[Idx] = Temp_Arr[Right_Part];
            Right_Part++;
        }
        Idx++;
    }

    if (Mid >= Left_Part) {
        for (int i = 0; i <= Mid - Left_Part; i++) {
            Arr[Idx + i] = Temp_Arr[Left_Part + i];
        }
    }
}

void Merge_Sort(int Start, int End) {
    if (Start < End) {
        int Mid = (Start + End) / 2;
        Merge_Sort(Start, Mid);
        Merge_Sort(Mid + 1, End);
        Merge(Start, Mid, End);
    }
}
```