Korea Aerospace University  
Team: Koala

Coach : Prof. Hyunyoung Kil  
Contestant : Chanhee Woo, Dabeen Jeong, Youngseo Jeon

# Table of Contents

Table of Contents 1

Algorithm tip 2

Data Structures 3

Binary Index Tree 3  
Segment Tree ........................................... 2

Trie 4

Suffix array 5

Graph Algorithms 6

Topology Sort(위상 정렬) 6

Disjoint Set 6

Kruskal MST 7

Floyd-Warshall 8

Dijkstra 9

Bellman-Ford 9

SubNode counting 11

Mathematical 11

큰 수 곱셈 11

Geometry 14

다각형 넓이 구하기 14

ccw 14

벡터 클래스, 다각형 클리핑 15

Dynamic Programming 23

행렬 거듭제곱 dp 23

Bitmasking dp(TSP) 23

Quantize dp(분할) 23

String 23

LCS(Longest Common Subsequence) 24

# Algorithm tip

1. 체계적인 접근을 위한 질문들

“알고리즘 문제 해결 전략” 책에서 발췌

* 비슷한 문제를 풀어본 적이 있던가?
* 단순한 방법에서 시작할 수 있을까? (brute force)
* 내가 문제를 푸는 과정을 수식화할 수 있을까? (예제를 직접 해결해보면서)
* 문제를 단순화할 수 없을까?
* 그림으로 그려볼 수 있을까?
* 수식으로 표현할 수 있을까?
* 문제를 분해할 수 있을까?
* 뒤에서부터 생각해서 문제를 풀 수 있을까?
* 순서를 강제할 수 있을까?
* 특정 형태의 답만을 고려할 수 있을까? (정규화)

2. dp 설계 팁

- n이 10,000 정도 된다 -> 2차원 10,000 \* 10,000 dp는 메모리 초과로 불가능하므로 1차원 dp이거나 그리디일 것임.

- 작은 문제가 중복이 되어서 중복된 문제의 정답이 모두 동일하다.(Optimal Substructure)

- 정답을 한 번 구했으면 어딘가에 정답을 메모한다.(Memoization)

3. int 형 범위

- 2^31 + 1 ~ 2^31 - 1

# Data Structures

## Binary Index Tree(구간 합 트리)

구간의 합/곱/최소/최대를 log(n)에 구해줄 수 있는 자료구조

build() : 초기 tree 생성

update(idx,val) : idx 값 val로 변경, 트리를 올라가며 구간의 값도 갱신(log(n))

query(l,r) : [l,r) 구간의 합을 구해줌.(log(n))

const int MAX = 2e6;

const int mod = 1e9 + 7;

ll tree[MAX];

void build() {

for (int i = n - 1; i > 0; i--) {

tree[i] = tree[i << 1] \* tree[(i << 1) | 1];

tree[i] %= mod;

}

}

void update(int idx, int val) {

idx += n - 1;

tree[idx] = val;

while (idx > 1) {

tree[idx / 2] = tree[idx] \* tree[idx ^ 1];

tree[idx / 2] %= mod;

idx >>= 1;

}

}

int query(int l, int r) { //[ㅣ,r)구간의 합 또는 곱을 구해줄 수 있음. 여기선 곱

ll ret = 1;

l += n - 1, r += n - 1;

while (l < r) {

if (l & 1) {

ret \*= tree[l];

ret %= mod;

l += 1;

}

if (r & 1) {

r -= 1;

ret \*= tree[r];

ret %= mod;

}

l >>= 1; r >>= 1;

}

return ret;

}

## Segment Tree(세그먼트 트리)

구간의 합/곱/최소/최대를 log(n)에 구해줄 수 있는 자료구조

build() : 초기 tree 생성

update(idx,val) : idx 값 val로 변경, 트리를 올라가며 구간의 값도 갱신(log(n))

query(l,r) : [l,r) 구간의 합을 구해줌.(log(n))

#include <bits/stdc++.h>

using namespace std;

typedef long long ll;

typedef pair<ll, ll> pll;

const int INF = INT\_MAX;

vector<ll>arr(100007);

vector<pll>MinMaxTree(300007); // min = first, max = second

int id[100007];

pll init(int node, int start, int end) {

if (start == end) { // 리프 노드

return MinMaxTree[node] = pll(start, start);

}

else {

pll left = init(node \* 2, start, (start + end) / 2), right = init(node \* 2 + 1, (start + end) / 2 + 1, end);

return MinMaxTree[node] = pll(min(left.first, right.first), max(left.second, right.second));

}

}

pll query(int node, int start, int end, int left, int right) {

if (end < left || right < start) return pll(INF, 0);

if (left <= start && end <= right) return MinMaxTree[node];

pll l = query(node \* 2, start, (start + end) / 2, left, right), r = query(node \* 2 + 1, (start + end) / 2 + 1, end, left, right);

return pll(min(l.first, r.first), max(l.second, r.second));

}

pll update(int node, int start, int end, int index, ll num) {

if (index < start || end < index) return MinMaxTree[node];

if (start == end) MinMaxTree[node] = pll(num, num);

else {

pll left = update(node \* 2, start, (start + end) / 2, index, num);

pll right = update(node \* 2 + 1, (start + end) / 2 + 1, end, index, num);

MinMaxTree[node].first = min(left.first, right.first);

MinMaxTree[node].second = max(left.second, right.second);

}

return MinMaxTree[node];

}

## Trie(트라이)

문자열을 저장하고 효율적으로 탐색하기 위한 트리 형태의 자료구조

Insert : 트라이에 문자열 추가

Find : 트라이에 해당 문자열이 있는지 확인

int t; //test case

int n; //call number

const int NUMBER = 10; // '0' ~ '9'

int toNumber(char ch) { return ch - '0'; }

struct TrieNode {

TrieNode\* children[NUMBER];

bool terminal;

//생성, 소멸자

TrieNode() : terminal(false) {

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

}

~TrieNode() {

for (int i = 0; i < NUMBER; i++)

if (children[i]) delete children[i];

}

//트라이에 번호 추가

void insert(const char\* key) {

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

else {

int next = toNumber(\*key);

if (children[next] == NULL)

children[next] = new TrieNode();

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

}

}

//트라이에 번호 일치 확인

TrieNode\* find(const char\* key) {

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

int next = toNumber(\*key);

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

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

}

};

int main()

{

ios\_base::sync\_with\_stdio(0);

cin.tie(0);

cin >> t;

while (t--) {

cin >> n;

vector<string>input; //전화번호 길이 순 정렬

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

char buf[11];

cin >> buf;

input.push\_back(buf);

}

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

TrieNode\* trie = new TrieNode();

bool check = false;

for (int i = 0; i < input.size(); i++) {

TrieNode\* node = trie->find(input[input.size() - 1 - i].c\_str());

if (node != NULL) check = true;

trie->insert(input[input.size() - 1 - i].c\_str());

}

if (check) cout << "NO\n";

else cout << "YES\n";

}

}

## Suffix Array & LCP(Longest Common Prefix)

Suffix array : 사전순으로 정렬된 모든 suffix의 시작 인덱스 배열

LCP: 인접한 접미사들이 겹치는 길이.

가장 긴 공통 문자열 = suffixarray[i~max(LCP[i])]

O(n\*log(n)^2)

const int MAX = 1e6;

string S;

int n, gap, suffix\_array[MAX], group[MAX];

int temp[MAX];

int LCP[MAX];

bool comp(int i, int j) {

//같은 그룹일 때

if (group[i] != group[j]) return group[i] < group[j];

//그룹이 다를 때

i += gap; j += gap;

return (i < n && j < n) ? (group[i] < group[j]) : (i > j);

}

void build\_suffix\_array() {

n = S.size();

//초기화

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

suffix\_array[i] = i;

group[i] = S[i]; //처음엔 첫 글자로 비교하기 위해

}

for (gap = 1;; gap \*= 2) {

// gap 글자만큼만 보고 정렬

sort(suffix\_array, suffix\_array + n, comp);

memset(temp, 0, sizeof(int)\*MAX);

//그룹 넘버링

for (int i = 0; i < n - 1; i++)

temp[i + 1] = temp[i] + comp(suffix\_array[i], suffix\_array[i + 1]);

for (int i = 0; i < n; i++) group[suffix\_array[i]] = temp[i]; //복사

if (temp[n - 1] == n - 1) break; //모두 그룹이 나눠졌으면 종료

}

}

void build\_LCP() {

for (int i = 0, k = 0; i < n; i++, k = max(k - 1, 0)) {

if (group[i] == n - 1) continue;

for (int j = suffix\_array[group[i] + 1]; S[i + k] == S[j + k]; k++); //j = 다음 그룹의 suffix array

LCP[group[i] + 1] = k;

}

}

int main() {

cin >> S;

build\_suffix\_array();

build\_LCP();

//print();

}

# Graph Algorithms

## Topology Sort(위상 정렬)

Directed acyclic graph에서 가능한 위상 정렬 순서를 구해줌.

순서는 res에 담기고 true를 반환

그러한 순서가 존재하지 않을 때 false를 반환.

Bfs와 들어가는 간선의 차수를 이용하여 구현.

int n; scanf("%d", &n);

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

int temp; scanf("%d", &temp);

hour[i] = temp; //걸리는 시간

int works; scanf("%d", &works); //작업 개수

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

int t; scanf("%d", &t);

a[t].push\_back(i);

ind[i] += 1; //차수 기록

}

}

queue<int>q;

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

if (ind[i] == 0) { //선행 관계 없는 작업들

q.push(i);

d[i] = hour[i]; //작업 완료 시간 고정

}

}

while (!q.empty()) {

int x = q.front();

q.pop();

for (int i = 0; i < a[x].size(); i++) {

int y = a[x][i];

ind[y] -= 1;

if (d[y] < d[x] + hour[y]) {

d[y] = d[x] + hour[y];

}

if (ind[y] == 0) {

q.push(y);

}

}

}

## Kruskal MST

최소스패닝트리의 total edge cost를 구해줌

Edge cost로 정렬하고 disjoint set으로 연결 여부를 확인하며 node들을 n-1개의 간선으로 연결하면 mst가 완성되고 그 비용을 구할 수 있음.

#include <bits/stdc++.h>

#define pii pair<int,int>

#define pipi pair<pair<int,int>,pair<int,int> >

#define fs first

#define sc second

#define sorta(a) sort(a.begin(),a.end());

typedef long long ll;

using namespace std;

//typedef vector<vector<int> > matrix;

#define pll pair<ll,ll>

#define ppl pair<ll,pair<ll,ll>>

const int INF = (int)1e9 + 10;

const int MAX = 100001;

int V, E;

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

// 여기에 disjoint set class 선언

//\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

int main() {

cin >> V >> E;

vector<ppl> edges;

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

int a, b, c;

cin >> a >> b >> c;

edges.push\_back({ c,{a,b} });

}

//sort by cost

sorta(edges);

//use disjoint set to check if node is connected

Set \*s = new Set();

ll ansCost = 0;

//\*\*\*iterate for all edges ('E' times) \*\*\*

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

int n1, n2, c;

n1 = edges[i].sc.fs; n2 = edges[i].sc.sc; c = edges[i].fs;

int p1, p2;

p1 = s->find(n1); p2 = s->find(n2);

if (p1 == p2) continue;

s->merge(n1, n2);

ansCost += c;

}

cout << (ll)cost << "\n";

}

## Floyd-Warshall

모든 노드에서 모든 노드로의 최단 거리를 구해줌.

O(n^3)

const int MAX = 101;

int graph[MAX][MAX];

int V, E;

void floyd() {

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

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

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

if (k == i || k == j || i == j) continue;

graph[i][j] = min(graph[i][j], graph[i][k] + graph[k][j]);

}

}

}

}

## Dijkstra

어떤 노드에서 다른 모든 노드로의 최단거리를 구해줌

Priority\_queue를 이용해서 그리디하게 최단거리 갱신.

memset(dist, 125, sizeof(dist));

dist[0][0] = arr[0][0];

priority\_queue<tuple<int, int, int>, vector<tuple<int, int, int> >, greater<tuple<int, int, int> > >pq;

pq.emplace(0, 0, arr[0][0]);

while (!pq.empty()) {

int y, x, cost;

tie(y, x, cost) = pq.top();

pq.pop();

if (dist[y][x] != cost) continue;

for (int k = 0; k < 4; k++) {

int ny = y + dy[k];

int nx = x + dx[k];

if (0 > ny || ny >= n || 0 > nx || nx >= n) continue;

if (dist[ny][nx] > cost + arr[ny][nx]) {

dist[ny][nx] = cost + arr[ny][nx];

pq.emplace(ny, nx, dist[ny][nx]);

}

}

}

## BellmanFord

음의 간선이 있을 때 최단 경로를 찾는 알고리즘

만약 음의 사이클이 있다면, 비어있는 벡터를 리턴하게 된다.

#include <iostream>

#include <vector>

#include <cstring>

#include <algorithm>

using namespace std;

vector<pair<int, int> >adj[510];

int tc, n, m, w;

const int INF = 987654321;

vector<int> bellmanFord(int src) {

vector<int> upper(n + 1, INF);

upper[src] = 0;

bool updated = false;

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

updated = false;

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

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

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

int cost = adj[here][i].second;

if (upper[there] > upper[here] + cost) {

upper[there] = upper[here] + cost;

updated = true;

}

}

}

//모든 간선에 대해서 완화 실패시 곧장 종료한다.

if (!updated) break;

}

//n번째 순회에 대해서도 완화가 성공했다면 음수 사이클이 존재한다.

if (updated) upper.clear();

return upper;

}

int main()

{

scanf("%d", &tc);

while (tc--) {

//input data

for (int i = 0; i < 510; i++) adj[i].clear();

scanf("%d %d %d", &n, &m, &w);

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

int s, e, t;

scanf("%d %d %d", &s, &e, &t);

adj[s].push\_back(make\_pair(e, t));

adj[e].push\_back(make\_pair(s, t));

}

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

int s, e, t;

scanf("%d %d %d", &s, &e, &t);

adj[s].push\_back(make\_pair(e, -t));

}

vector<int> ans = bellmanFord(1);

if (ans.size() == 0) printf("YES\n");

else printf("NO\n");

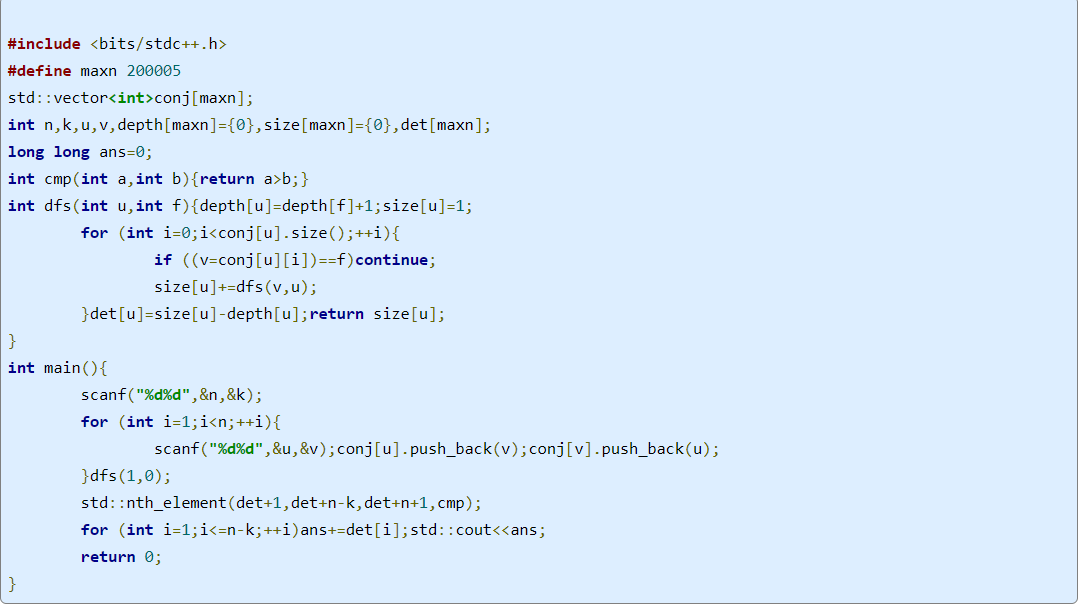
}

return 0;

}

## Subnode counting

트리에서 각 노드의 서브노드 개수 빠르게 구하기



# Mathematical

## 큰 수 곰셈

큰 수의 곱셈을 vector를 이용하여 오버플로우 없이 할 수 있음.

O(n^2)

//낮은 자리수부터 벡터에 담아야 계산 구현하기 편함

vector<int> stoVec(string s) {

vector<int> ret;

for (int i = s.size() - 1; i >= 0; i--)

ret.push\_back(s[i] - '0');

return ret;

}

//자리 올림 처리

void normalize(vector<int>& C) {

C.push\_back(0);

for (int i = 0; i < C.size() - 1; i++) {

if (C[i] < 0) {

int borrow = (abs(C[i] + 9)) / 10;

C[i + 1] -= borrow;

C[i] += 10 \* borrow;

}

else {

C[i + 1] += (C[i] / 10);

C[i] %= 10;

}

}

while (C.size() > 1 && C.back() == 0) C.pop\_back();

}

vector<int> multiply(vector<int>& A, vector<int>& B) {

vector<int> C(A.size() + B.size() + 1, 0);

for (int i = 0; i < A.size(); i++) {

for (int j = 0; j < B.size(); j++) {

C[i + j] += A[i] \* B[j];

}

}

normalize(C);

return C;

}

int main() {

string a, b;

cin >> a >> b;

vector<int> A = stoVec(a);

vector<int> B = stoVec(b);

vector<int> C = multiply(A, B); // 낮은 자리수부터 담겨있다.

for (int i = C.size() - 1; i >= 0; i--)

cout << C[i];

cout << "\n";

}

# Geometry

## 다각형 넓이 구하기

다각형을 외부의 한 점과 다각형의 인접한 두 꼭지점들을 한 방향으로 돌리면서 외적

한 것을 더하면 다각형의 면적을 구할 수 있음.

typedef long long ll;

#define pll pair<ll,ll>

vector<pll> a(n); //다각형을 이루는 꼭지점들의 좌표(점벡터). 정수 좌표에 대해서 구현.

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

cin >> a[i].fs >> a[i].sc;

}

ll tot = 0;

//외적을 다 합한 뒤에 절대값 취하고 2로 나누면 넓이.

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

tot += (a[i].fs\*a[(i + 1) % n].sc - a[i].sc\*a[(i + 1) % n].fs);

}

## Ccw

세 점 a,b,c를 가르키는 세 벡터가 있을 때,

a,b,c를 순서대로 잇는 두 선분은 b에서 어느방향으로 꺽을지

ccw(a,b) : 원점에서 벡터b가 벡터 a의 반시계 방향이면 양수, 시계방향이면 음수, 평행이면 0을 리턴

ccw(p,a,b): 점 p를 기준으로 벡터 b가 벡터 a의 반시계 방향이면 양수, 시계방향이면 음수, 평행이면 0을 리턴

double ccw(VECTOR a,VECTOR b){

return a.cross\_product(b);

}

double ccw(VECTOR p,VECTOR a,VECTOR b){

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

}

## 점과 선분 사이의 거리 by binary search

Case 1: 수선의 발이 선분 위에 떨어질 때 -> 거리 = 수선의 발 벡터 길이

Case 2: 수선의 발이 선분 밖에 있을 때 -> 거리 = 양 끝점 중 가까운 곳과의 거리

double pointToSegment2(VECTOR p, VECTOR a, VECTOR b) {

for (int i = 0; i < 100; i++) {

double pa = (p - a).norm(), pb = (p - b).norm();

if (pa > pb) a = (a + b)\*0.5;

else b = (a + b)\*0.5;

}

return (p - a).norm();

}

# Dynamic Programming

## 행렬 거듭제곱 dp

10^18의 dp 점화식을 적절히 더하고 빼서 행렬로 표현하면

분할정복을 이용해 행렬 곱셈을 log 수준으로 해낼 수 있다.

typedef long long ll;

typedef vector<vector<ll> > matrix;

const ll mod = 1000000007;

matrix operator \* (const matrix& A, const matrix& B) {

int n = A.size();

matrix C(n, vector<ll>(n));

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

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

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

C[i][j] += A[i][k] \* B[k][j];

}

C[i][j] %= mod;

}

}

return C;

}

matrix I;

void make\_Identity(int n) {

I = matrix(n, vector<ll>(n));

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

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

if (i == j) I[i][j] = 1;

else I[i][j] = 0;

}

}

}

matrix pow(matrix& A, ll m) {

if (m == 0) return I;

if (m % 2 > 0) return pow(A, m - 1)\*A;

//else

matrix half = pow(A, m / 2);

return half \* half;

}

int main() {

ll n; cin >> n;

matrix C(2, vector<ll>(2));

//C[0][0] = 0, C[0][1] = 1, C[1][0] = 1, C[1][1] = 1;

make\_Identity(2);

matrix M\_i\_1 = pow(C, n - 1);

ll ans = M\_i\_1[1][1];

cout << ans << '\n';

}

## Bitmasking dp (TSP problem)

N<=16에 대한 TSP

방문여부를 정수 하나의 bit로 표현

args : here = 현재위치 , visited = 방문 여부

return : here에서 시작하는 남은 부분 경로의 최소 길이 반환

#include<bits/stdc++.h>

using namespace std;

typedef long long ll;

const int INF = 987654321;

const int MAX = 20;

int w[MAX][MAX];

int dp[MAX][1 << MAX];

int n;

int go(int here, int visited)

{

if (visited == (1 << (n + 1)) - 2) {

if (w[here][1] != 0) {

return w[here][1];

}

}

int& ret = dp[here][visited];

if (ret != -1) return ret;

ret = INF;

for (int next = 1; next <= n; next++)

{

if ((visited & (1 << next)) || w[here][next] == 0) continue;

int cand = w[here][next] + go(next, (visited + (1 << next)));

ret = min(ret, cand);

}

return ret;

}

int main()

{

cin >> n;

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

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

cin >> w[i][j];

}

}

memset(dp, -1, sizeof(dp));

int ans = go(1, (1 << 1));

cout << ans << "\n";

return 0;

}

## Quantize dp(분할)

냅색 dp

cin >> n >> m;

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

cin >> W[i] >> V[i];

}

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

cin >> bag[i];

}

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

for (int j = 0; j < N - 1; j++) {

dp[i][j] = max(dp[i][j], dp[i - 1][j]);

if (j - W[i] >= 0) dp[i][j] = max(dp[i][j], dp[i - 1][j - W[i]] + V[i]);

}

}

## lis nlogn 풀이

#include <cstdio>

#include <vector>

#include <algorithm>

using namespace std;

int n, arr[1000001], p[1000001], cnt;

vector<int> ans, print;

int main() {

scanf("%d", &n);

for (int i = 1; i <= n; i++) scanf("%d", &arr[i]);

ans.push\_back(arr[1]);

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

if (ans[cnt] < arr[i]) {

ans.push\_back(arr[i]), cnt++;

p[i] = cnt;

}

else {

int pos = lower\_bound(ans.begin(), ans.end(), arr[i]) - ans.begin();

ans[pos] = arr[i];

p[i] = pos;

}

}

printf("%d\n", cnt + 1);

for (int i = n; i >= 1 && cnt >= 0; i--) {

if (p[i] == cnt) {

print.push\_back(arr[i]);

cnt--;

}

}

for (int i = print.size() - 1; i >= 0; i--) {

printf("%d ", print[i]);

}

}

# String

## LCS(Longest Common Subsequence)

Longest Common Subsequence(공통 부분 수열) 길이구하기

const int MAX = 5050;

int dp[MAX][MAX]; //lcs 길이 저장용

string s, p; //두 문자열 s,p

//두 문자열으로 2차원 테이블 만들었을 때,

for (int i = 0; i < s.size(); i++) {

for (int j = 0; j < p.size(); j++) {

//운좋게 딱 맞았네. 그럼 두 인덱스를 모두++ 하고 길이는 1증가하면 되지.

if (s[i] == p[j]) dp[i + 1][j + 1] = dp[i][j] + 1;

else {//다르네. 그럼 테이블의 위에서 오거나 왼쪽에서 온 것 중에 최댓값.

dp[i + 1][j + 1] = max(dp[i + 1][j], dp[i][j + 1]);

}

lcs = max(lcs, dp[i + 1][j + 1]);

}

}