1. C++과 STL

C++언어와 STL(Standard Template Library)에 대한 기초 점검 & Quick Review (vector, ~~list~~, string, stack, queue, deque, sort, stable_sort, pair, set, map, ~~unordered_set~~, ~~unordered_map~~, priority_queue

Array

```
#include <iostream>
#define MAXA 100
using namespace std;

const int MAXA_ = 100;

int A[MAXA];
int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);

  return 0;
}
```

vector

```
#include <iostream>
#include <vector>
using namespace std;

struct strt{
  int a, b;

  bool operator<(const strt& st)const{
    if(a == st.a)
      return b < st.b;
    return a < st.a;
  }
};

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);

  vector<pair<int, int>> vec(100, {0, 0});

  vec.push_back({1, 1});
```

```
  for(auto itr = vec.begin(); itr != vec.end(); itr++){
    (*itr).first;
    (*itr).second;
  }

  for(int i = 0; i < (int)vec.size(); ++i){
    vec[i].first;
    vec[i].second;
  }

  vec.front();
  vec.back();

  return 0;
}
```

string

```
#include <iostream>
#include <string>
using namespace std;

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);

  string str = "hello, world!";
  string str2 = "tmp";

  str += str2;

  str += '1';

  cout << str << endl;

  str = "Z";

  cout << str[0] - 'A' << endl;

  str.find("o");

  return 0;
}
```

stack

```cpp
#include <iostream>
#include <stack>
using namespace std;

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);

  stack<int> stk;

  stk.push(1);
  int t = stk.top();
  stk.pop();

  while(!stk.empty()){
    stk.pop();
  }

  return 0;
}
```

queue, priority_queue

```cpp
#include <iostream>
#include <queue>
using namespace std;

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);

  queue<int> q;

  q.push(1);
  int t = q.front();
  q.pop();

  q.empty();

  while(!q.empty()){
    int t = q.front();
    q.pop();
  }

  q.size(); // unsigned
```

```
  priority_queue<int> pq;

  pq.push(1);
  t = pq.top();
  pq.pop();


  pq.push(4);
  pq.push(1);
  pq.push(10);
  while(!pq.empty()){
    int t = pq.top();
    pq.pop();
    cout << t << endl;
  }

  priority_queue<int, vector<int>, greater<int>> pq2;

  pq2.push(4);
  pq2.push(1);
  pq2.push(10);
  while(!pq2.empty()){
    int t = pq2.top();
    pq2.pop();
    cout << t << endl;
  }

  return 0;
}
```

deque

```
#include <iostream>
#include <deque>
using namespace std;

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);

  deque<int> dq;

  dq.push_back(1);
  dq.push_front(1);

  dq.pop_front();
```

```
  dq.pop_back();

  return 0;
}
```

algorithm

```
#include <iostream>
#include <vector>
using namespace std;


bool cmp(int a, int b){
  return a < b;
}
int arr[100];
int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);

  vector<int> vec;


  min(1, 2);
  max(1, 2);

  sort(vec.begin(), vec.end());
  sort(vec.begin(), vec.end(), less<int>());
  sort(vec.begin(), vec.end(), cmp);

  sort(arr, arr + 10);

  lower_bound(vec.begin(), vec.end(), 10);
  upper_bound(vec.begin(), vec.end(), 10);
  lower_bound(arr, arr + 100, 10);

  return 0;
}
```

set

```
#include <iostream>
#include <set>
using namespace std;

int main(){
  ios::sync_with_stdio(false);
```

```
  cin.tie(0);

  set<int> s;

  s.insert(1);
  auto itr = s.find(1);
  if(itr != s.end()){
    int value = (*itr);
  }
  s.erase(1);
  if(s.find(1) != s.end())
    s.erase(s.find(1));

  itr = s.lower_bound(1);
  itr = s.upper_bound(1);

  for(auto itr = s.rbegin(); itr != s.rend(); ++itr){
    cout << (*itr) << endl;
  }

  multiset<int> ss;
  for(auto itr = ss.begin(); itr != ss.end(); ++itr){
    cout << (*itr) << endl;
  }
  if(ss.find(1) != ss.end())
    ss.erase(ss.find(1));


  return 0;
}
```

map

```
#include <iostream>
#include <map>
using namespace std;

int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);

  map<int, int> m;
  m.insert({1, 1});
  m[1] = 1;

  for(auto itr = m.begin(); itr != m.end(); ++itr){
    cout << (*itr).first << endl;
    cout << (*itr).second << endl;
```

```
  }

  if(m.find(2) != m.end()){
    int t = m[2];
  }

  if(m.find(10) != m.end())
    m.erase(m.find(10));

  multimap<int, int> mm;


  return 0;
}
```

2. 수학 1

모듈러 연산과 연산 법칙, GCD, LCM, 소수판별, 에라토스테네스의 체, 소인수 분해, 진법 변환, 팩토리얼(순열, 조합 등)

모듈러 연산과 연산 법칙

$(A+B)\%M = (A\%M + B\%M)\%M$

$(A-B)\%M = (A\%M - B\%M + M)\%M$

$(A\times B)\%M = (A\%M \times B\%M)\%M$

GCD, LCM

```
typedef long long int lli;

// GCD
lli gcd(lli a, lli b){
  return b ? gcd(b, a % b) : a;
}
// LCM
lli lcm(lli a, lli b){
  return a * b / gcd(a, b);
}
```

소수판별

```
// 소수판별
bool isPrime2(int a){
  for(int j = 2; j * j <= a; ++j){
    if(a % j == 0)
      return false;
  }
  return true;
}
```

에라토스테네스의 체

```
// 에라토스테네스의 체
vector<int> primes;
bool isPrime[MAXN + 1], visit[MAXN + 1];
void solve(){
  for(int v = 2; v <= MAXN; ++v){
    if(visit[v]) continue;
    isPrime[v] = true;
    primes.push_back(v);
    for(int j = v * 2; j <= MAXN; j += v){
      visit[j] = true;
    }
  }
}
```

소인수 분해

```
// 소인수 분해
void factorization(int x){
  vector<int> fact;
  for(int j = 2; j * j <= x && x > 1; ++j){
    while(x % j == 0){
      fact.push_back(j);
      x /= j;
    }
  }
  for(int j = 0; j < fact.size(); ++j)
    cout << fact[j] << ", ";
}
```

진법 변환

```
// digit진수인 문자열 num을 int로 변환
int stringToInt(string num, int digit){
  int ret = 0;
  for(int j = 0; j < num.size(); ++j){
    int n = num[j] - '0';
    if(num[j] >= 'a' && num[j] <= 'z')
      n = num[j] - 'a' + 10;
    ret = ret * digit + n;
  }
  return ret;
}
// int num을 digit진수인 문자열로 변환
string intToString(int num, int digit){
  string str = "";
  stack<int> stk;
  while(num){
    stk.push(num % digit);
    num /= digit;
  }
  while(!stk.empty()){
    char c = stk.top() + '0';
    if(stk.top() > 9)
      c = stk.top() - 10 + 'a';
    str += c;
    stk.pop();
  }
  return str;
}
```

팩토리얼

```
// 파스칼의 삼각형을 이용한 조합 계산 (전처리)
int C[101][101], M = 100000;
void solveComb(){
  C[0][0] = 1;
  C[1][0] = 1;
  C[1][1] = 1;

  for(int j = 1; j <= 100; ++j){
    C[j][0] = 1;
    for(int i = 1; i <= j; ++i)
      C[j][i] = (C[j - 1][i - 1] % M + C[j - 1][i] % M) % M;
  }
}
```

3. 그래프 1

그래프 정의, 그래프 표현 방법, DFS, BFS, 연결요소

그래프 정의

그래프 표현 방법

- 인접 리스트

```
vector<int> G[MAXN + 1]; // 인접 리스트
int main(){
  int V; cin >> V; // 정점 개수
  int E; cin >> E; // 간선 개수
  for(int j = 0; j < E; ++j){
    int v, u; cin >> v >> u;
    G[v].push_back(u);
    //양방향 간선
    G[u].push_back(v);
  }

  // 하나의 정점에 대해서는 빠름
  int v = 1;
  for(int j = 0; j < G[v].size(); ++j){
    int u = G[v][j];
  }

  // 임의의 두 정점에 대해서는 느림
  int v = 1, u = 1;
  for(int j = 0; j < G[v].size(); ++j){
```

```
    if(G[v][j] != u) continue;
    // 연결됨
  }
}
```

- 인접 행렬

```
int G[MAXN + 1][MAXN + 1]; // 인접 행렬
int main(){
  int V; cin >> V; // 정점 개수
  int E; cin >> E; // 간선 개수
  for(int j = 0; j < E; ++j){
    int v, u; cin >> v >> u;
    G[v][u] = 1;
    // 양방향 간선인 경우
    G[u][v] = 1;
  }

  // 하나의 정점에 대해서는 느림
  int v = 1;
  for(int j = 1; j <= MAXN; ++j){
    if(!G[v][j]) continue;
    int u = j;
  }

  // 임의의 두 정점에 대해서는 빠름
  int c = G[1][2];
  return 0;
}
```

DFS

```
bool visit[MAXN + 1]; // 방문 여부
void dfs(int v){ // O(V + E)
  if(visit[v]) return;
  visit[v] = true;

  // do something

  for(int j = 0; j < G[v].size(); ++j){
    int u = G[v][j];
    dfs(u);
  }
}
```

BFS

```
bool visit[MAXN + 1]; // 방문 여부
void bfs(int start){ // O(V + E)
  queue<int> q;
  q.push(start);
  while(!q.empty()){
    int v = q.front(); q.pop();
    if(visit[v]) continue;
    visit[v] = true;
    // do something
    for(int j = 0; j < G[v].size(); ++j){
      int u = G[v][j];
      if(!visit[u])
        q.push(u);
    }
  }
}
```

연결요소

```
int main(){
  // 연결요소 개수 구하기
  int cnt = 0;
  for(int v = 1; v <= V; ++v){
    if(visit[v]) continue;
    ++cnt;
    dfs(v);
  }
}
```

4. 트리 1

트리 정의, 트리 표현 방법, 트리 순회 방법, 트리 지름, 트리 높이, 이진 트리

트리 정의

트리 표현 방법

- 부모와 자식이 구분되어 주어지는 경우

```
vector<int> child[100];
int par[100];
int main(){
  int N; cin >> N;
  for(int j = 0; j < N - 1; ++j){
    // 부모 자식
    int p, c; cin >> p >> c;
    par[c] = p;
    child[p].push_back(c);
  }
  return 0;
}
```

- 부모 자식이 구분되지 않는 경우

```
vector<int> G[100];

void buildTree(int pre, int v){
  for(int j = 0; j < G[v].size(); ++j){
    int u = G[v][j];
    if(pre == u) continue;

    child[v].push_back(u);
    par[u] = v;
    buildTree(v, u);
  }
}
int main(){
  int N; cin >> N;
  for(int j = 0; j < N - 1; ++j){
    int u, v; cin >> u >> v;
    // 부모 자식 구분 없음
    G[u].push_back(v);
    G[v].push_back(u);
  }
  int root = 0;
  par[root] = -1;
  buildTree(-1, root);
  return 0;
}
```

트리 순회 방법

재귀 함수를 사용하여 쉽게 구현할 수 있다.

- 전위 순회 (pre-order)
  - 노드를 먼저 방문한다.
- 중위 순회 (in-order)
  - 노드를 왼쪽 서브트리를 순회 하고 방문한다.
- 후위 순회 (post-order)
  - 노드를 왼쪽 서브트리와 오른쪽 서브트리를 순회하고 방문한다.

트리 지름

트리에서 임의의 두 정점을 선택하고 해당 정점 사이의 거리를 계산하면 n^2개의 쌍에 대한 거리가 나온다. 이 중 가장 긴 거리가 트리의 지름이다.

DFS를 다음과 같이 2번 돌려서 O(N)만에 찾을 수 있다.

1. 임의의 정점에서 부터 DFS를 돌려서 가장 멀리 떨어진 정점 하나를 구한다.
2. 단계 1에서 구한 정점에서 부터 DFS를 돌려서 가장 먼 정점까지의 거리를 구한다. 해당 거리가 트리의 지름이다.

트리 높이

특정 노드에서 부터 leaf노드 까지의 길이 중 가장 긴 길이

이진 트리

자식이 최대 2인 트리

완전 이진 트리인 경우 Array로 표현하는 것이 효율적임

최저 공통 조상

LCA (Lowest Common Ancestor)

임의의 두 노드에 대해 각각의 root까지의 경로에서 공통된 노드 중 높이가 가장 낮은 노드

서로소 집합 (Disjoint-set)

집합 A_i에서 $A_i \cap A_j = \varnothing$ 이면서 $\bigcup_i A_i = U$ 를 만족하는 집합

Disjoint-set 자료구조

다음의 연산을 지원함

- find(v)
  - 원소 v가 속한 집합을 반환한다.
- union(u, v)
  - 원소 u와 v가 속한 집합을 합한다

최적화

- find 연산 최적화
  - 경로 압축
- union 연산 최적화
  - 각 집합에 속한 원소 count

```
int djset[MAXN + 1], djset_cnt[MAXN + 1];
int djset_find(int v){
    if(djset[v] == v)
        return v;
    int r = djset_find(djset[v]);
    djset[v] = r;
    return r;
}
void djset_union(int v1, int v2){
    int r1 = djset_find(v1);
    int r2 = djset_find(v2);
    if(r1 == r2) return;
    if(djset_cnt[r1] > djset_cnt[r2]){
        djset[r2] = r1;
        djset_cnt[r1] += djset_cnt[r2];
    }else{
        djset[r1] = r2;
        djset_cnt[r2] += djset_cnt[r1];
    }
}
void init() {
    for(int j = 0; j < N; ++j){
        djset[j] = j;
        djset_cnt[j] = 1;
    }
}
```

5. Brute-Force

선형 탐색, 재귀 호출, 순열 조합, backtracking

선형 탐색

for문으로 가능한 모든 경우를 탐색하는 것

재귀 호출

DFS를 이용하여 모든 경우를 탐색하는 것

순열 조합

순열, 조합은 DFS로 구현할 수 있다.

C++에서는 모든 조합을 계산해 주는 `next_permutation`이이 `algorithm` 헤더에 존재 (단, 가지치기 불가능)

backtracking

탐색, 상태 검증, 상태 복원 과정을 DFS로 구현하여 전체탐색을 수행한다.

가지치기를 하여 상수 최적화를 할 수 있다.

backtracking

- 자연수 N과 M이 주어졌을 때, 1부터 N까지 자연수 중에서 중복 없이 M개를 고른 수열을 사전순으로 모두 출력하라.

```
#include <iostream>
#include <vector>
using namespace std;

int N, M;
vector<int> arr; // 선택한 숫자
bool sel[8+1]; // 중복체크
void dfs(int cnt){
  if(cnt == M){ // 종료조건
    for(int j = 0; j < arr.size(); ++j)
      cout << arr[j] << " ";
    cout << "\n";
    return;
  }
```

```
  for(int i = 1; i <= N; ++i){
    if(sel[i]) continue; // 중복체크

    // 탐색하기
    arr.push_back(i);
    sel[i] = true;
    dfs(cnt + 1);

    // 상태복원하기 (backtracking)
    arr.pop_back();
    sel[i] = false;
  }
}
int main(){
  ios::sync_with_stdio(false);
  cin.tie(0);
  cin >> N >> M;
  dfs(0);
  return 0;
}
```

6. DP 1

동적 계획법, 메모이제이션, 잘 알려진 문제들

동적 계획법

DP (Dynamic Programming)

주어진 문제를 작은 문제로 나누어, 작은 문제의 해답을 구해 큰 문제의 해답을 구하는 방식의 알고리즘

크게 세 가지 과정을 거침

- DP table을 자연어로 정의한다 (정의가 잘 안되면 DP테이블의 차원을 높여본다)
- DP 식을 세운다
- 초기항을 정의한다

크게 2가지 방식으로 해결할 수 있음

- top-down 방식
  - 재귀 함수를 이용하여 해결하는 방식
  - 비교적 쉽게 (직관적) 구현할 수 있다.
- bottom-up 방식
  - for문을 이용하여 해결하는 방식
  - DP식을 세웠을 때 구현할 수 있다. 빠르다.

피보나치 수열의 변형 문제

$F_{i+2} = F_{i+1} + F_i$

- 계단 문제
  - 각 계단에서 1칸 올라가거나, 2칸 올라갈 수 있을 때, N번째 계단까지 올라가는 경우의 수
  - DP table의 정의
    - dp[i] = i번째 계단까지 왔을 때, 경우의 수
  - DP 식 정의
    - dp[i] = dp[i - 1] + dp[i - 2]
  - 초기항 정의
    - dp[0] = 1
    - dp[1] = 1
- 타일 채우기 문제
  - 1x2타일과 2x1모양의 타일을 사용하여 2xN크기의 공간을 채울 때, 채우는 경우의 수

LCS

최장 공통 부분수열 (Longest Common Subsequence)

$$LCS(X_i, Y_j) = \begin{cases} \emptyset & \text{if } i = 0 \text{ or } j = 0 \\ LCS(X_{i-1}, Y_{j-1}) \hat{} x_i & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max\{LCS(X_i, Y_{j-1}), LCS(X_{i-1}, Y_j)\} & \text{if } i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

LIS

최장 증가 부분수열 (Longest Increasing Subsequence)

$$d[i] = \max_{\substack{j=0 \ldots i-1 \\ a[j]<a[i]}} (d[j] + 1)$$

배낭문제

냅색 문제 (Knapsack Problem)

- 배낭 문제의 DP 테이블 정의
  - dp[i][w] = i번째 물건까지 고려했을 때, 무게가 w인 경우의 최대 값어치
- 정답
  - max(dp[N - 1][w])