삼성 기출문제 원탑, 구현 문제중에 백미 중의 백미라고 할 수 있는 완전탐색, 완탐과 백트래킹을 알아보도록 하겠습니다.


완전탐색은
brute force, exhaustive key search 이며 노가다 라고 보시면 됩니다.
모든 경우의 수를 탐색하는 알고리즘입니다.

그렇다면 경우의 수란 뭘까요?
순열 또는 조합을 말합니다. 이것 밖에 없습니다.
여기에다가 어떤 경우의 수가 있는 것이죠.
조합 + 로직이거나
조합을 구축하면서 로직이거나..

무든 2가지 밖에 없습니다.

자 그럼 어떨 때 완탐을 써야 할까요? 보통은 1억 미만일 때 완탐을 써도 됩니다. 문제마다 다르지만요. 1억이라..
너무나도 거대한 숫자지만, 우리의 컴퓨터는 해낼 수 있습니다.
믿고 넘기는 것을 잘하시면 됩니다.

“

반복문을 활용한 완전탐색

---

for or while 를 이용한 완전탐색입니다.
단순히 선형적으로 숫자 찾는 것도 완탐입니다. 그 수가 ~~야? 하는 것을 일일히 확인해보며 모든 경우의 수를 확인하니까요.

```
#include<bits/stdc++.h>
using namespace std;
int main() {
    vector<int> v = {1, 3, 2, 5, 6, 7, 8};
    for(int i = 0; i < v.size(); i++){
        if(v[i] == 5){
            cout << i << "\n";
            break;
        }
    }
    int i = 0;
    while(i < v.size()){
        if(v[i] == 5){
            cout << i << "\n";
            break;
        }
        i++;
    }
    return 0;
}
```

## 문제 : 2400!! 2400!! 2400!!

트위치BJ 랄로는 2400이란 숫자를 좋아한다. 파카는 랄로의 수를 만들고자 하는데 랄로의 수란 2400이 들어간 수를 말한다. 첫번째 랄로의 수는 2400이고 두번째 랄로의 수는 수는 12400, 세번째 랄로의 수는 22400이다.
N이 입력으로 주어졌을 때 N번째 랄로의 수를 구하라. N은 300이하로 주어진다.

```cpp
#include<bits/stdc++.h>
using namespace std;
int cnt, n;
int main() {
    cin >> n;
    int i = 2400;
    while(true){
        string a = to_string(i);
        if(a.find("2400") != string::npos){
            cnt++;
            if(n == cnt){
                cout << a << '\n';
                break;
            }
        }
        i++;
    }
    return 0;
}
```

> 재귀함수를 활용한 완전탐색

---

재귀함수를 활용한 완전탐색을 알아보죠. 근데 그 전에!! 반복문으로 되면 무조건 반복문으로 해야합니다.
함수호출을 여러번 하는 것은 코스트가 너무나도 큽니다.

반복문으로 안 될 거같다? 또는
너무 복잡하거나 어떠한 행위는 반복하는데 매개변수만 수정해서 넘기면 될 거 같다? 그러면 재귀함수로 하는게 좋습니다.
- 조합 or 순열 + (DFS, BFS 등 알고리즘)
- 경우의 수 마다 생각해야 하는 로직도 나옴.

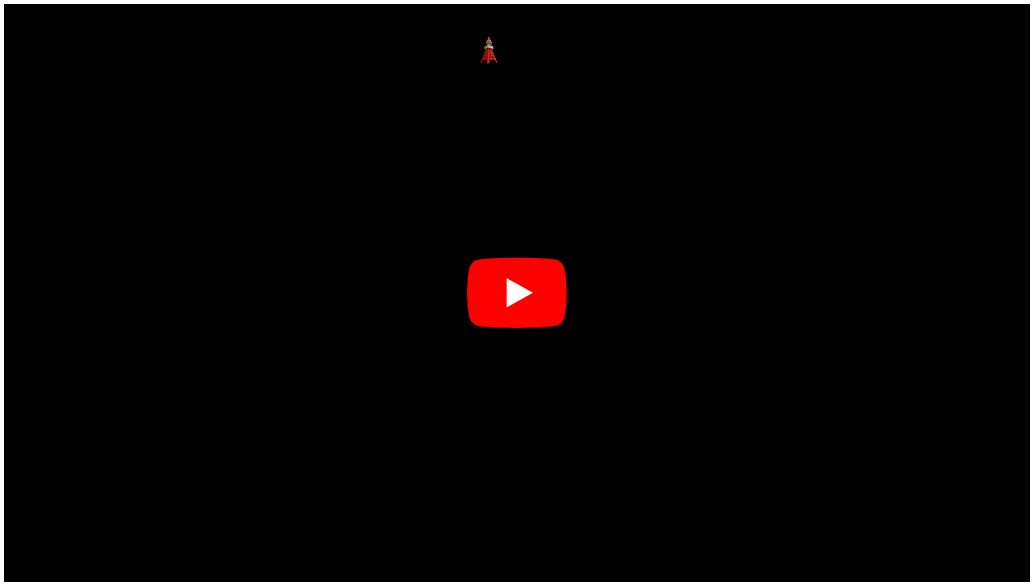
Ex) nC1, nC2, nC3 .. 등 이런 경우의 수를 다 생각해야 한다면?
재귀함수로 하는게 나음. (물론 비트마스킹도 있지만…)

## 문제 : 승철이의 문단속

승철이는 도쿄 위의 빨간 구름위에 올라가있다. 이 구름은 그대로 내버려두면 땅으로 떨어져 100만명의 사상자가 발생한다. 구름을 멈추는 방법은 구름의 특정 위치에 요석을 꽂으면 된다. 해당 위치에는 숫자가 표기가 되어있고 몇 개를 골라 숫자의 합이 “소수 " 가 될 때 구름은 멈춘다. 총 몇 개의 경우의 수가 있는지 말하라.
N개의 요석 후보의 숫자와 다음 줄에 해당 숫자들이 나온다. N <= 100

```
input:
10
24 35 38 40 49 59 60 67 83 98

output:
176
```

https://youtu.be/qmNBT3_wpko



```
#include<bits/stdc++.h>
using namespace std;
int n, temp;
vector<int> v;
bool check(int n) {
    if(n <= 1) return 0;
    if(n == 2) return 1;
    if(n % 2 == 0) return 0;
    for (int i = 3; i * i <= n; i++) {
        if (n % i == 0) return 0;
    }
    return 1;
}
int go(int idx, int sum){
    if(idx == n){
        //cout << "SUM " << sum << "\n";
        return check(sum);
    }
    return go(idx + 1, sum + v[idx]) + go(idx + 1, sum);
}
int main() {
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> temp;
        v.push_back(temp);
    }
    cout << go(0, 0) << "\n";
    return 0;
}
```

“

백트래킹

back tracking입니다. 완전탐색 & 가지치기이며 완탐에 인간미를 섞었다라고 보시면 됩니다.
최대한 불필요한 탐색을 피하는 것이죠.

재귀함수를 활용한 완전탐색
자, 예를 들어
N과 N개의 자연수가 주어진다. 여기서 몇개의 숫자를 골라 합을 mod 11을 했을 때 나오는 가장 큰수를 구하라.

```
Input:
10
24 35 38 40 49 59 60 67 83 98
output:
10
```

이를 완탐으로 하게 된다면?

```
#include<bits/stdc++.h>
using namespace std;
int n, temp, ret;
vector<int> v;
const int mod = 11;

void go(int idx, int sum){
    if(idx == n){
        ret = max(ret, sum % mod);
        return;
    }
    go(idx + 1, sum + v[idx]);
    go(idx + 1, sum);
}
int main() {
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> temp;
        v.push_back(temp);
    }
    go(0, 0);
    cout << ret << "\n";
    return 0;
}
```

이렇게 됩니다. 하지만 너무나도 불필요한 경우의 수를 탐색을 많이 하죠.

한번 디버깅을 해볼까요?

```cpp
#include<bits/stdc++.h>
using namespace std;
int n, temp, ret, cnt;
vector<int> v;
const int mod = 11;

void go(int idx, int sum){
    if(idx == n){
        ret = max(ret, sum % mod);
        cnt++;
        return;
    }
    go(idx + 1, sum + v[idx]);
    go(idx + 1, sum);
}
int main() {
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> temp;
        v.push_back(temp);
    }
    go(0, 0);
    cout << ret << "\n";
    cout << cnt << "\n";
    return 0;
}
```

cnt를 찍어봤더니 1024의 경우의 수를 모조리 탐색하는 것을 볼 수 있습니다.

근데 이 문제,
mod N를 하게 된다면 해당 숫자의 범위는 0 ~ n - 1의 범위를 가지는 것은 자명합니다.
따라서, 11 mod는 10이 최대의 숫자가 된다는 것은 자명합니다. 해당 부분을 넣어서 해당 경우의 수를 제거하는 것이 바로 백트래킹입니다.

```
#include<bits/stdc++.h>
using namespace std;
int n, temp, ret;
vector<int> v;
const int mod = 11;
int cnt = 0;
void go(int idx, int sum){
    if(ret == 10) return;
    if(idx == n){
        ret = max(ret, sum % mod);
        cnt++;
        return;
    }
    go(idx + 1, sum + v[idx]);
    go(idx + 1, sum);
}
int main() {
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> temp;
        v.push_back(temp);
    }
    go(0, 0);
    cout << ret << "\n";
    cout << cnt << "\n";
    return 0;
}
```

앞의 코드에서 다음 코드부분이 백트래킹입니다. 간단하죠?

```
if(ret == 10) return;
```

이렇게 필요없는 부분에 대한 경우의 수를 return; 등으로 빠르게 종료시켜 해당 부분에 대한 탐색을 이어나가지 않게 하는 것입니다.

그렇다면 얼마나 가지치기가 되었는지 볼까요?

```cpp
#include<bits/stdc++.h>
using namespace std;
int n, temp, ret, cnt;
vector<int> v;
const int mod = 11;

void go(int idx, int sum){
    if(ret == 10) return;
    if(idx == n){
        ret = max(ret, sum % mod);
        cnt++;
        return;
    }
    go(idx + 1, sum + v[idx]);
    go(idx + 1, sum);
}
int main() {
    cin >> n;
    for(int i = 0; i < n; i++){
        cin >> temp;
        v.push_back(temp);
    }
    go(0, 0);
    cout << ret << "\n";
    cout << cnt << "\n";
    return 0;
}
```

찍어보면 cnt는 10이 찍히게 됩니다.
이로써 필요없는 경우의 수는 모조리 가지치기를 한 것을 볼 수 있습니다.

> 

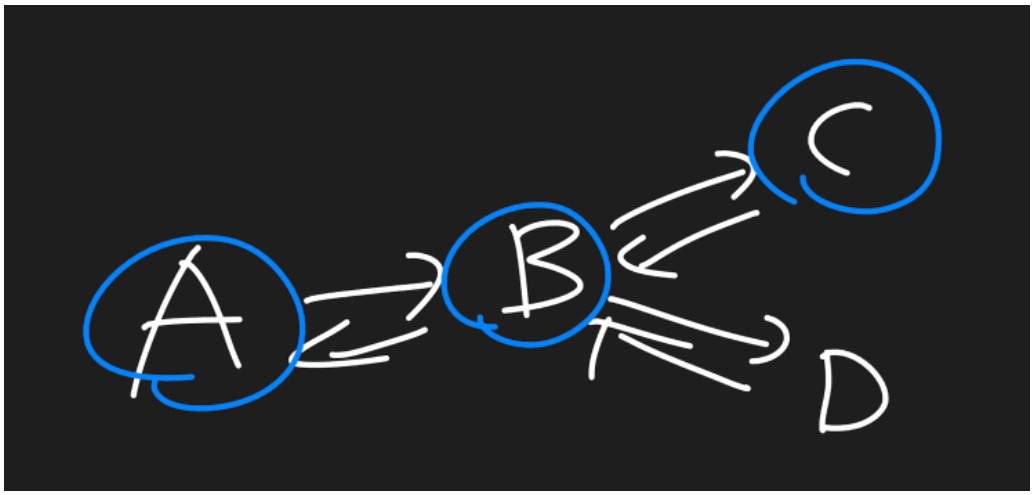
## 완전탐색 – 원복

---

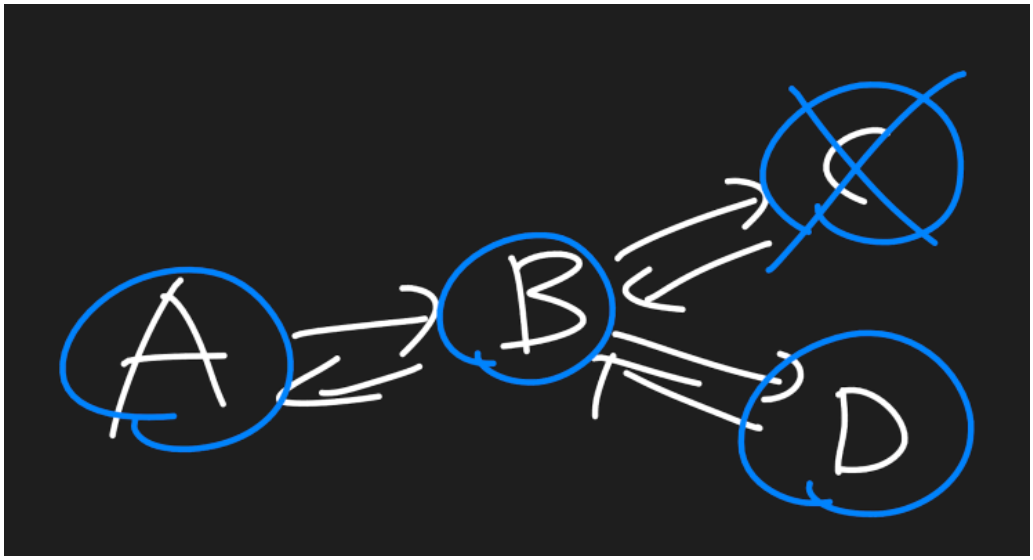
우리는 경우의 수를 모두 따지는 완탐을 하고 있습니다. 어떤 맵에서 어떤 것을 색칠하거나 뭘 세운다라고 했을 때 경우의 수들끼리 서로의 상태값이 영향을 미치지 않게 하려는 방법이 바로 원복입니다.

보통은 **방문배열인 visited 를 통해 "색칠하고" "다시 지운다(원복)"**를 통해 해하는 것이죠.

조금 더 간편하게 예를 들어볼게요.
A -> B -> C 와
A -> B -> D의 경우의 수를 출력하는 문제를 풀어 볼게요.




자 방문 배열을 통해 A, B, C를 색칠합니다.
색칠한다는 것은 visited[A] = 1 따위가 되겠죠?
자 그다음에 A -> B -> D라는 경우의 수를 뽑아낼 때 입니다.
C라는 상태가 색칠되어있지만, 해당 상태를 지워버립니다. A - > B - >D라는 경우의 수를 뽑아낼 때 필요없는 경우의 수고 C는 D로 갈 때 영향을 미치면 안되는 경우의 수기 때문입니다.




코드로 나타내면 어떻게 될까요?

```cpp
#include <bits/stdc++.h>
using namespace std;
int visited[4];
vector<int> adj[4];
vector<int> v;
void print(){
    for(int i : v) cout << char(i + 'A') << " ";
    cout << '\n';
}

void go(int idx){
    if(v.size() == 3){
        print(); return;
    }
    for(int there : adj[idx]){
        if(visited[there]) continue;
        visited[there] = 1;
        v.push_back(there);
        go(there);
        visited[there] = 0;
        v.pop_back();
    }
}
int main() {
    adj[0].push_back(1);
    adj[1].push_back(2);
    adj[1].push_back(3);
    adj[1].push_back(0);
    adj[2].push_back(1);
    adj[3].push_back(1);

    visited[0] = 1;
    v.push_back(0);
    go(0);
    return 0;
}
```

출력

```
A B C
A B D
```

앞의 코드에서 중요한 점은 바로 이부분입니다.

```cpp
        visited[there] = 1;
        v.push_back(there);
        go(there);
        visited[there] = 0;
        v.pop_back();
```

해당 정점을 방문한다. (visited[there] = 1)
그리고 해당 정점을 나중에 출력할 수 있도록 v에 push한다. (이부분은 문제에서 모든 경우의 수를 출력하라는 말이 없다면 안해도 됩니다. )

```
        visited[there] = 1;
        v.push_back(there);
```

자 이렇게 해서 해당 정점을 방문했다면!!

그 방문한 정점의 상태값을 지워버립니다.

```
        visited[there] = 0;
        v.pop_back();
```

이를 통해 A -> B -> C와 A -> B -> D는 독립적인 상태로 남아있을 수 있는 것이죠.
색칠한 경우의 수의 상태값이 다음 경우의수에 영향을 미치지 않도록 A -> B -> C가 아닌 A -> B 상태로 그 이전 상태로 원상복구, 즉 원복을 해줘야 합니다.

## 문제 : 긍정왕 홍철이의 구걸 여행

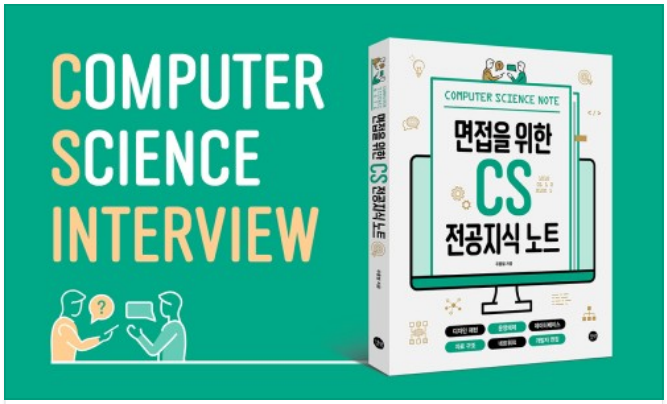
홍철이는 3 * 3 맵에서 {0, 0} 지점에서 길을 잃어버렸다. 긍정왕 홍철이는 길을 잃어버린 김에 구걸을 하면서 돈을 모으면서 여행을 가려고 한다. 목적지는 {2, 2}이며 방문한 정점은 다시 방문할 수 없고 해당 맵에 구걸로 얻을 수 있는 돈들이 있다. 홍철이는 4방향(상하좌우)로 움직일 수 있다.  {2, 2}까지 간다고 했을 때 이 돈들을 모으는 모든 경우의 수를 출력하여라.
맵 :
{10, 20, 21},
{70, 90, 12},
{80, 110, 120}

```cpp
#include <bits/stdc++.h>
using namespace std;
const int n = 3;
int a[3][3] = {
{10, 20, 21},
{70, 90, 12},
{80, 110, 120}
};
int visited[3][3];
const int dy[] = {-1, 0, 1, 0};
const int dx[] = {0, 1, 0, -1};
vector<int> v;
void print(){
    for(int i : v) cout << i << " ";
    cout << '\n';
}

void go(int y, int x){
    if(y == n - 1 && x == n - 1){
        print();
        return;
    }
    for(int i = 0; i < 4; i++){
        int ny = y + dy[i];
        int nx = x + dx[i];
        if(ny < 0 || nx < 0 || ny >= n || nx >= n) continue;
        if(visited[ny][nx]) continue;
        visited[ny][nx] = 1;
        v.push_back(a[ny][nx]);

        go(ny, nx);

        visited[ny][nx] = 0;
        v.pop_back();
    }
}
int main() {
    visited[0][0] = 1;
    v.push_back(a[0][0]);
    go(0, 0);
    return 0;
}
```

출력

```
10 20 21 12 120
10 20 21 12 90 110 120
10 20 21 12 90 70 80 110 120
10 20 90 12 120
10 20 90 110 120
10 20 90 70 80 110 120
10 70 90 20 21 12 120
10 70 90 12 120
10 70 90 110 120
10 70 80 110 90 20 21 12 120
10 70 80 110 90 12 120
10 70 80 110 120
```

https://inf.run/KPcs




CS지식의 정석 | CS면접 디자인패턴 네트워크 운영체제 데이터…

국내 1위 "면접을 위한 CS전공지식노트" 저자의 디자인패턴, 네트워크, …

inf.run

https://inf.run/YfJa


10주완성 C++ 코딩테스트 | 알고리즘 IT취업 - 인프런 | 강의

네이버, 카카오, 삼성의 코딩테스트를 합격시켰다! 10주 완성 C++ 코딩…

inf.run