Backtracking & Graph Traversal

스터디 1주차

문종건

fragmentbombs@gmail.com

BOJ 1759 - 암호 만들기

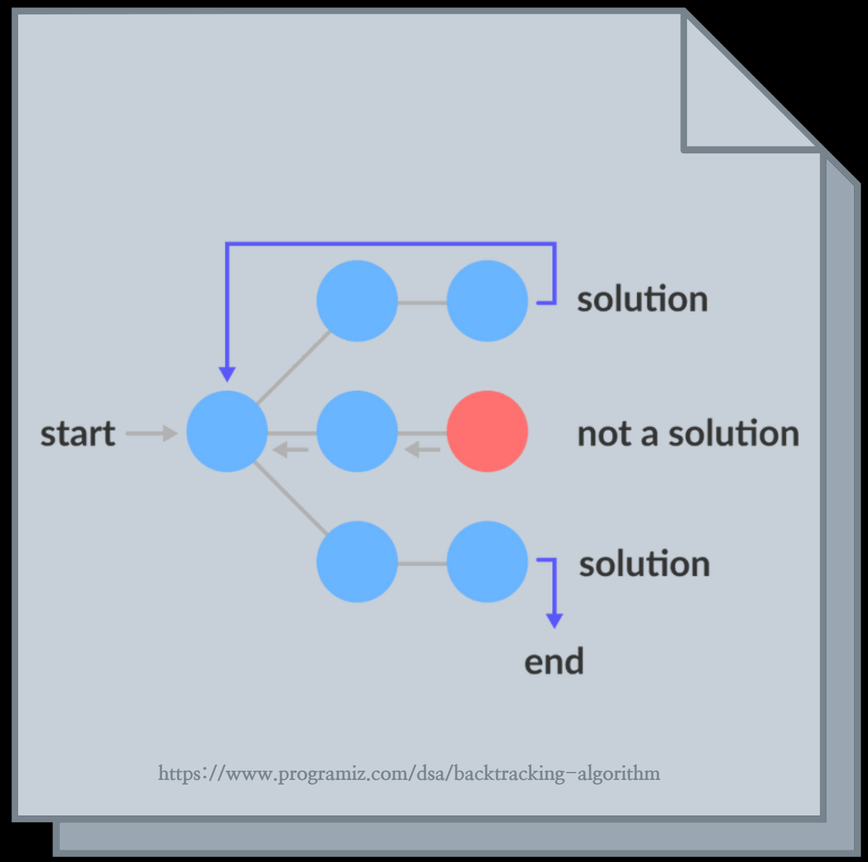
- 암호는 서로 다른 L개의 알파벳 소문자로, 최소 한개의 모음과 두개의 자음으로 구성된다.
- 알파벳이 암호에서 증가하는 순서대로 배열된다.
- C가지 문자가 주어졌을때, 가능한 암호를 모두 구해라.
- Time Limit : 2s, Memory Limit : 128MB
- 3 <= L <= C <= 15

풀이 탐색

- 주어진 L과 C의 범위가 매우 작지만, 시간 제한은 2초.
- 모든 암호를 구해야 한다 -> Brute force? O(15!)?
- 증가하는 순서로 알파벳이 배열된다 -> 주어진 문자 셋을 순회할 방법이 어느 정도 정해져 있다. -> 시간 복잡도가 크게 줄어든다.

Backtracking

- 조건 상으로 가능한 모든 경우의 수를 탐색한다.
- 주로 재귀함수를 이용한다.
- 시간복잡도가 기본적으로는 O(N!)이지만, 주어진 조건에 따라 크게 줄어들 수 있다.
- 주로 주어지는 N의 범위가 두자리수 이하이다.



풀이 코드

- 모음 여부를 알 수 있도록 bool 배열 전처리
- 알파벳 순으로 이루어지므로 input 배열 정렬
- (현재까지) input배열 상에서 위치, 모음의 개수,
- 자음의 개수, 만들어진 문자열을 재귀적으로 넘겨줌
- 암호 문자열의 길이에 도달했을때, 정답여부 처리

```cpp
#include <bits/stdc++.h>

using namespace std;
int L,C;
vector<char> v;
vector<string> ans;
bool mo[102];

void sol(int idx, int vow, int con, string s){
    if(s.length()==L){
        if(vow>=1&&con>=2)ans.push_back(s);
        return;
    }
    if(idx>=C)return;
    for(int i = idx; i < v.size(); i++){
        string ts=s+v[i];
        sol(i+1,vow+mo[v[i]-'a'],con+!mo[v[i]-'a'],ts);
    }
}

int main()
{
    ios_base ::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> L>>C;
    v.resize(C);
    mo['a'-'a']=1;
    mo['e'-'a']=1;
    mo['i'-'a']=1;
    mo['o'-'a']=1;
    mo['u'-'a']=1;
    for(int i = 0; i < C; i++){
        cin >> v[i];
    }
    sort(v.begin(),v.end());
    sol(0,0,0,"");
    for(string s : ans){
        cout << s << '\n';
    }
}
```

BOJ 10026 - 적록색약

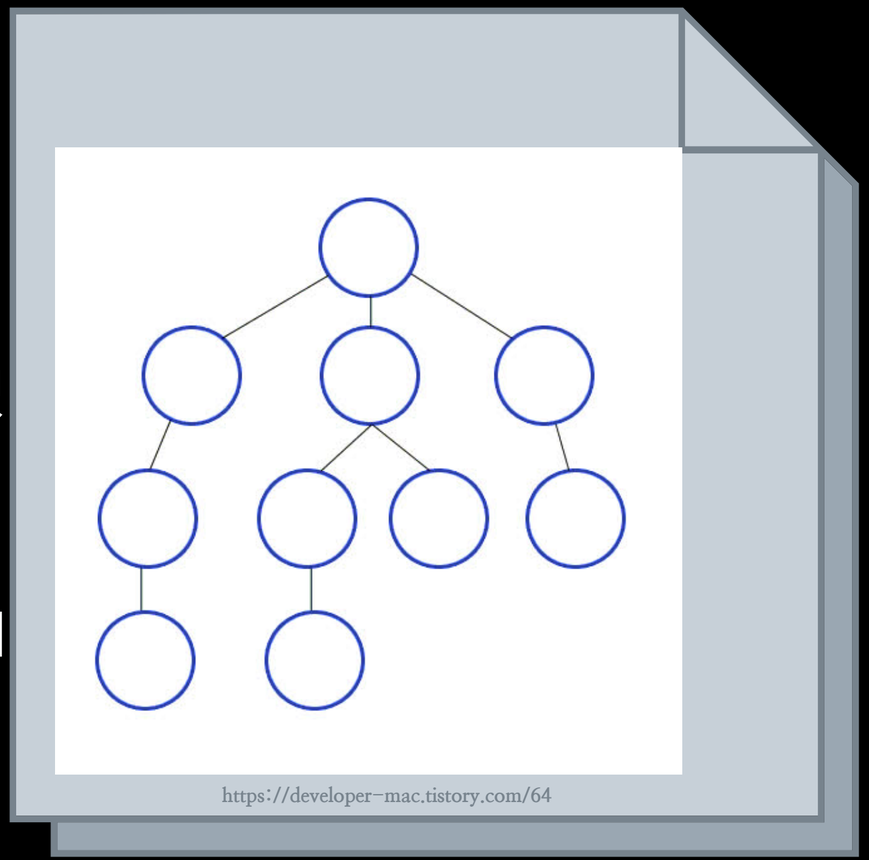
- 문자 R,G,B로 이루어진 2차원 배열이 주어진다.
- 상하좌우로 같은 문자가 인접하여 이루어지는 집합을 하나의 구역이라 한다.
- R과 G를 구별하지 못할때의 구역의 수와, 정상 상태에서의 구역의 수를 모두 구해라.
- Time Limit : 1s, Memory Limit : 128MB
- 1 <= N <= 100

풀이 탐색

- N의 범위가 작다, 2차원 배열을 전부 탐색해야 한다.
- BFS/DFS(O(V+E))로 같은 문자로 이루어진 구역을 각각 완전탐색한다. -> 대략 O(N^2)~O(N^3) 예상
- 두가지 조건에 대한 답안을 별도로 구해야 한다 -> 상황에 따라 각 문자를 구별하는 로직이 별도로 필요

DFS/BFS

- 그래프에서 시작점으로부터 연결된 모든 노드를 탐색한다.
- 시간복잡도 O(V+E) (V는 노드의 개수, E는 Edge의 개수)
- 둘 다 완전탐색 알고리즘이지만, 실제로 모든 노드를 반드시 전부 탐색해야 하는 경우는 BFS가 낫다. (구현에 따라 차이 있음)
- 따라서 여기서는 BFS를 선택
- 현재 있는 노드와 연결된 탐색되지 않은 모든 노드를 Queue에
- 넣고, Queue에 있는 다음 노드를 탐색한다.



https://developer-mac.tistory.com/64

풀이 코드

- 적록색약인 경우를 판별하기 위해, BFS에서 문자 동치 여부를 판별할 때 마다 문자를 바꿔줄 함수를 호출
- 배열의 모든 원소마다, 적록색약 여부에 따라 탐색되지 않은 경우 그 자리에서 BFS를 호출

```cpp
#include <bits/stdc++.h>

using namespace std;
using pi = pair<int,int>;
int N;
bool visited[102][102][2];
char arr[102][102];
int dx[]={0,0,-1,1},dy[]={1,-1,0,0};
int ans[2]={0,0};
char code(char c,bool blind){
    if(blind&&c=='G')return 'R';
    else return c;
}
void bfs(int sx, int sy, bool blind){
    queue<pi> q;
    q.push({sx,sy});
    char par=code(arr[sx][sy],blind);
    visited[sx][sy][blind]=1;
    while(!q.empty()){
        auto [cx,cy]=q.front();
        q.pop();
        for(int i = 0; i < 4; i++){
            int nx=dx[i]+cx,ny=dy[i]+cy;
            if(nx<0||nx>=N||ny<0||ny>=N)continue;
            if(visited[nx][ny][blind]||code(arr[nx][ny],blind)!=par)continue;
            q.push({nx,ny});
            visited[nx][ny][blind]=1;
        }
    }
    ans[blind]++;
}

int main()
{
    ios_base ::sync_with_stdio(false);
    cin.tie(NULL);
    cout.tie(NULL);
    cin >> N;
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            cin>>arr[i][j];
        }
    }
    for(int i = 0; i < N; i++){
        for(int j = 0; j < N; j++){
            for(int k = 0; k < 2; k++){
                if(!visited[i][j][k])bfs(i,j,k);
            }
        }
    }
    cout << ans[0] << ' ' << ans[1];
}
```

BOJ 16933 - 벽 부수고 이동하기 3(G1)

- 0과 1로 이루어진 2차원 배열이 주어진다.
- 0은 이동할 수 있는 곳이고, 1은 벽이다. 상하좌우로 인접한 곳으로만 이동 가능하다.
- 처음 이동할 때는 낮이고, 한번 이동하거나 대기할때마다 낮밤이 바뀐다.
- 낮에만 벽을 부수고 이동할 수 있으며, 벽은 K개까지만 부술 수 있다.
- 목적지까지 이동에 걸리는 최단시간을 구해라. (이동 혹은 대기에 각각 1초 소요)
- Time Limit : 2s, Memory Limit : 512MB
- 1 <= N, M <= 1000, 1 <= K <= 10

풀이 탐색

- 왼쪽 위 끝으로부터 오른쪽 아래 끝까지 최단거리를 구하는 문제로 생각 가능
- 최단거리? -> Dijkstra?
- 벽을 K개까지 부술 수 있다 -> 각 정점에 머물렀을 때 현재까지 부순 벽의 개수에 따라 정점분할을 한다고 생각하면, 탐색해야 할 정점의 개수가 O(N*M*K)로 매우 많아질 수 있다.

Dijkstra인줄 알았는데 BFS

- Dijkstra's algorithm -> 한 정점으로부터 그래프의 다른 모든 정점까지의 최단 거리를 구하는 알고리즘. O(ElogV)
- 정해진 시작점으로부터 목표 지점까지의 최단 거리를 구하는 것이 목적이므로, Dijkstra 알고리즘을 활용하여 그래프 탐색을 하는 것이 불필요한 정점 방문 횟수를 줄이고 효과적일 것으로 보일 수 있음.
- 허나 이동이나 대기의 기회비용이 모두 1이므로, BFS를 하는 것과 결과는 차이가 없을 것임. (BFS의 방문 순서와 거의 차이가 없음)
- 오히려 다익스트라를 사용하면, 벽을 부순 상태 구별에 따른 정보가 과도하게 priority queue/max heap에 들어가기 때문에, 정렬에 필요한 시간이 기하급수적으로 늘어나 시간 초과가 발생한다.

풀이 코드

- visited[x][y][부순횟수][낮밤]
- 현재까지 진행하는데 걸린 시간의 홀짝 여부에 따라 낮밤 구별 가능
- 낮밤 여부, 벽 여부에 따라 인접 칸으로 이동 및 제자리 대기 여부 결정

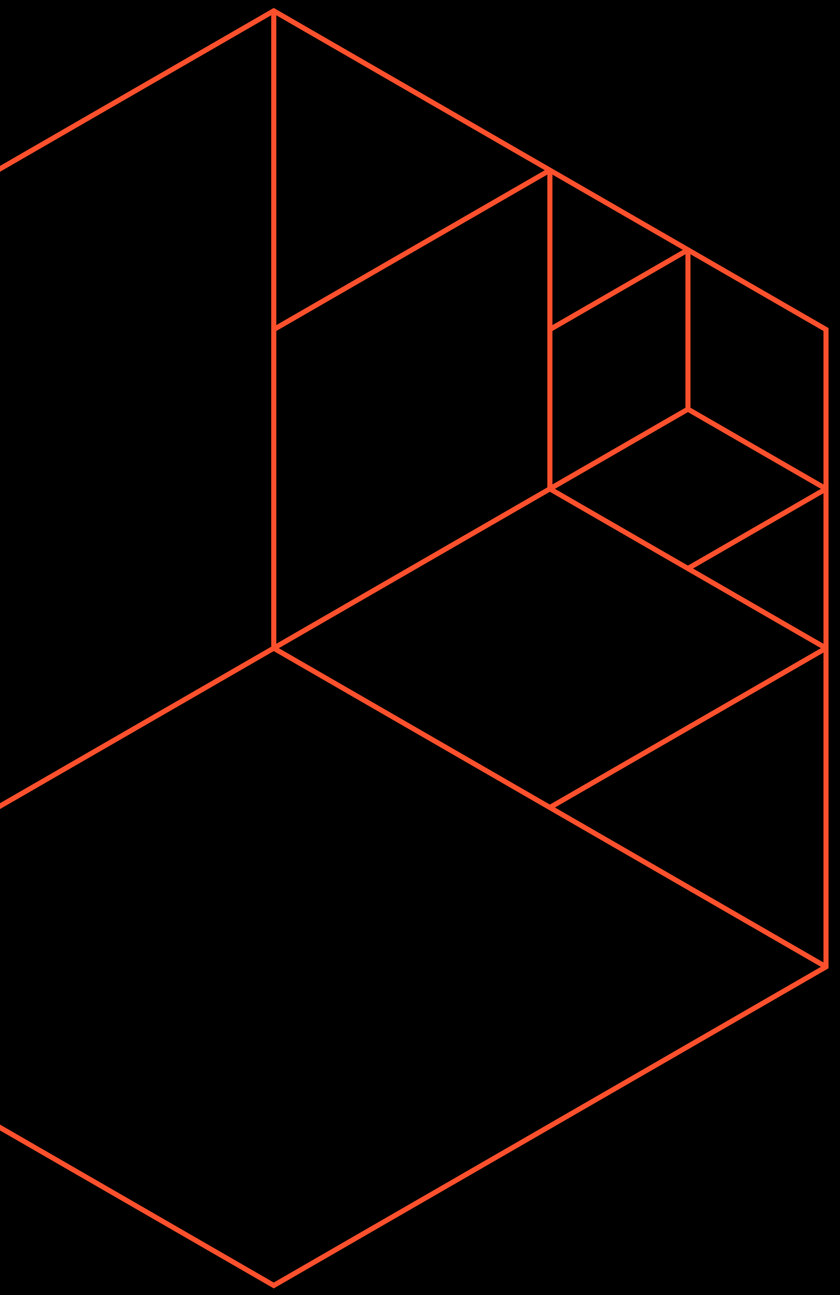
```cpp
int bfs(){
    queue<ppi> pq;
    pq.push({{1,0},{0,0}});
    int ret=-1;
    visited[0][0][0][1]=1;
    while(!pq.empty()){
        auto [p1,p2] = pq.front();
        auto [cx,cy] = p2; auto [cc,cw]=p1;
        bool cp = cc%2;
        //cout << cx << cy << cc << endl;
        pq.pop();
        if(cx==N-1&&cy==M-1){
            if(ret==-1)ret=cc;
            else ret=min(ret,cc);
        }
        if(visited[cx][cy][cw][cp]<cc)continue;

        for(int i = 0 ; i < 4; i++){
            int nx=dx[i]+cx,ny=dy[i]+cy;
            if(nx<0||nx>=N||ny<0||ny>=M)continue;
            if(arr[nx][ny]){
                if(!cp||cw>=K||visited[nx][ny][cw+1][!cp]<=cc+1)continue;
                visited[nx][ny][cw+1][!cp]=cc+1;
                pq.push({{cc+1,cw+1},{nx,ny}});
            }
            else{
                if(visited[nx][ny][cw][!cp]<=cc+1)continue;
                visited[nx][ny][cw][!cp]=cc+1;
                pq.push({{cc+1,cw},{nx,ny}});
            }
        }
        if(visited[cx][cy][cw][!cp]>cc+1){
            visited[cx][cy][cw][!cp]=cc+1;
            pq.push({{cc+1,cw},{cx,cy}});
        }
    }
    return ret;
}
```

```cpp
#include <bits/stdc++.h>
using namespace std;
using ll = long long;
using pi = pair<int,int>;
using ppi = pair<pi,pi>;
int N,M,K;
int arr[1002][1002];
int visited[1002][1002][11][2];
int dx[]={0,0,-1,1},dy[]={1,-1,0,0};

```

```cpp
int main()
{
    ios_base::sync_with_stdio(0);
    cin.tie(0);
    cout.tie(0);
    cin >> N>>M>>K;
    for(int i = 0; i <= N; i++)
        for(int j = 0; j <= M; j++)
            for(int k = 0; k <= K; k++)
                for(int l = 0; l < 2; l++)
                    visited[i][j][k][l]=1e9;
    //memset(visited,0xc0,sizeof(visited));
    for(int i = 0; i < N; i++){
        string s;
        cin>>s;
        for(int j = 0; j < M; j++){
            arr[i][j]=s[j]-'0';
        }
    }
    cout << bfs();
}
```



감

- 사합
  - 니다