

알고리즘 HW03

실행환경 : Visual studio 2019

사용 언어 : C++



2018203065 김보현

1. Main 함수 코드 설명

```

int main() {
    //cin의 컴파일 속도 향상시키기 위해서 쓴
    ios::sync_with_stdio(false);
    cin.tie(0);
    cin >> N;//(1)N입력
    for (int i = 0; i < N; i++) { // (2)N*N 만큼의 결과스위치 입력, 초기스위치map 만들기
        for (int j = 0; j < N; j++) {
            cin >> inputmap[i][j];
            makemap[i][j] = '#';
        }
    }
    dfs(0, 0);
    if (ans == 0) {
        cout << "no solution\n";
        return 0;
    }
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            cout << reswitch_map[i][j];
        }
        cout << "\n";
    }
    return 0;
}

```

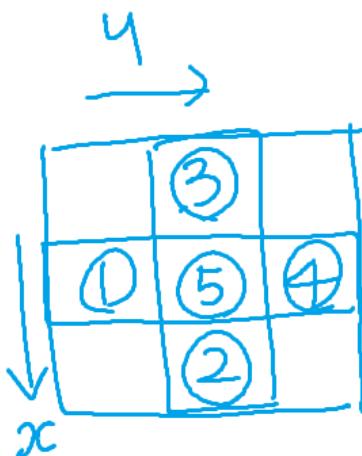
- (1) 입력 : N입력, N*N의 결과스위치입력, N*N의 초기스위치생성
- (2) dfs(0,0) 실행
- (3) 출력 : 초기스위치에서 놀려야하는 스위치표시한 map 출력
만약 존재하지 않는 경우, no solution을 출력

2. void switch_map(int x, int y, char arr[][MAXN]) 코드 설명

```

void switching(int x, int y, char arr[][MAXN]) {
    if (y > 0) {
        if (arr[x][y - 1] == '0') {
            arr[x][y - 1] = '#';
        }
        else {
            arr[x][y - 1] = '0';
        }
    }
    if (x < N - 1) {
        if (arr[x+1][y] == '0') {
            arr[x+1][y] = '#';
        }
        else {
            arr[x+1][y] = '0';
        }
    }
    if (x > 0) {
        if (arr[x-1][y] == '0') {
            arr[x-1][y] = '#';
        }
        else {
            arr[x-1][y] = '0';
        }
    }
    if (y < N - 1) {
        if (arr[x][y + 1] == '0') {
            arr[x][y + 1] = '#';
        }
        else {
            arr[x][y + 1] = '0';
        }
    }
    if (arr[x][y] == '0') {
        arr[x][y] = '#';
    }
    else {
        arr[x][y] = '0';
    }
}

```



먼저 이 함수는 행 x, 열 y에 대한 스위치를 눌렀을 때 발생하는 Switching을 구현한 함수이다. (x,y)를 누르면 자기 자신을 포함해 상하좌우 총 5개의 Switching이 발생하는데 각 위치별로 if문으로 0이면 #으로 #이면 0으로 바꾸도록 구현하였다. 또한 x,y값에 따라 상하좌우가 전부 포함안될 경우가 있다. x,y 값이 0보다 큰 경우에 x-1,y-1이 존재하고(상,좌) x,y값이 N-1보다 작은 경우에 x+1, y+1가 존재하기 때문에(하,우) 조건에 맞게 바꿔도록 구현하였다.

3. void dfs(int select, int column)

(1) 스위칭 하기

어떤 스위치를 2번 누르는건 원래의 상태로 돌아오기 때문에, 모든 스위치는 1번 혹은 0번 눌러야 한다. 행 단위로 스위치를 누르기로 설계하였다. (첫번째 행의 모든 열을 눌러야 다음 행으로 넘어가는 방식)

처음에, 첫번째 행의 스위칭의 모든 경우의 수를 본다 (2^N 가지의 경우의 수, $O(2^N)$)

두번째 행에 대한 스위칭은 첫번째 행이 결과값이 같다면 누르지 않고 같지 않다면 같은 행으로 누른다.

세번째 행에 대한 스위칭도 두번째 행이 결과값이 같다면 누르지 않고 같지 않다면 같은 행으로 누른다.

이렇게 두번째 - 마지막 행은 위의 행과 결과값과 비교를 통해 스위칭한다.

이때의 두번째 - 마지막 행의 시간복잡도는 N^2 이 된다. ($O(N^2)$)

스위칭 하기 전, 스위치하는 횟수와 스위치하는 (x,y)값을 저장한다.

```
void dfs(int select, int column) {
    int press=0;
    memcpy(made_map, makemap, sizeof(makemap));
    memcpy(switch_map, makemap, sizeof(makemap));
    int sum = 0;
    for (int i = 0; i < N; i++) { //첫번째 행 전체 둘린다
        if (switched[i]) { //해당 경우에서 선택한 열만 스위칭 한다.
            switch_map[0][i] = '0'; //스위칭한 열을 표시한다
            press++;
            switching(0, i, made_map); //지도중, 첫번째 행의 경우의 수
        }
    }
    for (int i = 1; i < N; i++) {
        for (int j = 0; j < N; j++) {
            if (made_map[i-1][j] != inputmap[i-1][j]) { //결과값이 같지 않을 경우,
                switch_map[i][j] = '0'; //아래의 행에서 스위칭을 한다
                press++;
                switching(i, j, made_map);
            }
        }
    }
}
```

(2) 결과map과 같은 경우, 결과값 저장하기

```
bool flag = false;
for (int i = 0; i < N && !flag; i++) { //맨 아래가 결과값이 같은 경우는
    //모든 경우가 결과와 동일해진 것이기 때문에
    //맨 아래의 결과값만 확인한다
    if (made_map[N - 1][i] != inputmap[N - 1][i]) flag = true; //같지 않는다면 flag true으로 표시
}
if (!flag && press<result) {
    result = press;
    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            reswitch_map[i][j] = switch_map[i][j];
        }
    }
}
```

맨 아래 행의 상태가 결과값이 같은 경우는 모든 행의 결과가 결과값이 같아진 상태이다. 그때, 스위칭한 (x,y)값을 저장한 switch_map을 main에서 출력하기 위해 저장한다.

그리고 스위치의 횟수 중, 제일 작은 횟수의 경우를 출력하기 위해서 현재의 횟수 press값이 저장해둔 이전의 스위치보다 적으면 그때 저장한다.

(3) dfs 하기

`bool switched[i]` 변수를 이용해서 `true`인 값만 첫번째 행의 `i`열을 스위칭 해준다.
`column` 변수는 첫번째 행의 몇번째 열까지 스위칭 한지를 알려주는 변수이다.
 $N=2$ 이면 첫번째 행의 모든 원소를 스위칭 하지 않은 경우, 하나의 원소만 스위칭 하는 경우, 모든 원소를 스위칭 하는 경우 총 4번의 `dfs`를 진행한다.

```

        }
        if (column == N) return;
        for (int i = select; i < N; i++) {
            if (!switched[i]) {
                switched[i] = true;
                dfs(i, column + 1);
                switched[i] = false;
            }
        }
    }
}

```

4. 고찰

알고리즘 백트래킹 강의에서 **N-queen** 문제의 경우, 두 퀸이 같은 행과 열 같은 대각선 상에 있으면 안된다는 명확한 **non promising**의 조건이 있는 반면 이 문제는 그런 조건이 제시되어 있지 않아 **promising**의 기준을 생각해내는 것이 쉽지 않았다. 같이 첨부된 엑셀 파일로 여러 경우를 따져보다가 모든 불을 켜기위해선 위쪽 스위치의 상태를 확인해서 스위치는 누르면 된다는 아이디어를 이용해 이를 활용해보기로 하였다. 백트래킹 알고리즘은 재귀호출이 자주 발생하여 과정을 생각하고 디버깅을 하기가 매우 까다로웠다. 현재 이 알고리즘은 n 이 20일 때 시간이 10초 이상 걸린다. n 의 값에 따라 `dfs` 호출을 2^n 을 해주고 `dfs`안에서 이중 `for`문으로 n^2 시간복잡도로 시간이 오래 걸린다. **prune state map**을 이용할 시 **worst**시간 복잡도만 $O(2^n N^2)$ 이 되므로 결과map과 같을 시 종료하게 하였으나, 그렇게 되면 스위칭의 최소 경우를 구하지 못하는 상황이 생겨, 어쩔 수 없이 항상 $O(2^n N^2)$ 의 시간복잡도를 갖게 되었다.

최소한의 스위칭을 구하면서 **non promising**을 선형식을 이용한 알고리즘이거나 깊이가 깊지 않은 트리구조로 세운다면, 현재 알고리즘보다 빠른 경우가 존재하지 않을까 싶다.

5. 실행 예시

(1)pdf 실행 예시

Microsoft Visual Studio 디버그 콘솔

```

5
# # # #
# # # #
# # # #
# # # #
# # # #
# # # #
# # # #
# # # #
# # # #
# # # #

```

Select Microsoft Visual Studio 디버그

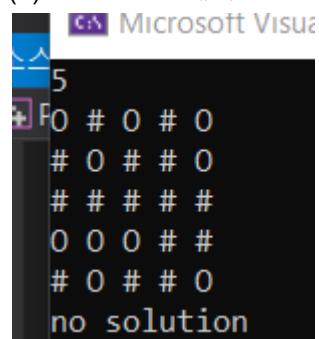
5		
# 0 # 0 #		4
0 # 0 # 0		0 0 0 0
# 0 # 0 #		0 0 0 0
0 # 0 # 0		0 0 0 0
# 0 # 0 #		0 0 0 0
0 0 0 # #		# 0 # #
# # # # #		# # # 0
# 0 # # 0		0 # # #
0 # 0 # 0		# # 0 #
# 0 # # 0		# 0 0 #

Microsoft Visual Studio 디버그 콘솔

5		
0 0 0 0 0		0 0 0 0 0
0 0 0 0 0		0 0 0 0 0
0 0 0 0 0		0 0 0 0 0
0 0 0 0 0		0 0 0 0 0

3,4번째 결과는 입출력간 한줄 띄움으로 가독성을 높였습니다.

(2) no solution 예시



```
Microsoft Visual Studio
소스 5
[+] F0 # 0 # 0
      # 0 # # 0
      # # # # #
      0 0 0 # #
      # 0 # # 0
no solution
```