REPORT

과목명 : 알고리즘
학 과 : 소프트웨어학부
학 번 : 2017203035
이 름 : 김수헌
교수명 : 김용혁 교수님

목차

Programing Language : C++

Compiler : MSVC, Microsoft Visual C++

1. 과제 알고리즘 분석

HW3 과제는 전구 끄기를 구현하는 과제였습니다. 저는 위 과제를 코딩하기 앞서 어떻게 구현할지 설계를 시작했습니다. 교수님이 요구하시는 시간 복잡도는 (2^n * n * n)이셨고, 저는 이 시간복잡도로 어떻게 구현을 할 수 있을지 생각해보았고 다음과 같은 결론을 내릴 수 있었습니다.

사전 분석

- **불을 끄는 순서는 결과에 구애받지 않는다.**
- **같은 버튼이 두 번 이상 눌릴 수 없다. 그렇게 되면 최솟값이 아니다.**
- **입력 받은 final state를 모두 꺼진 상태로 만들 수 있다면, 그것은 역으로 모두 꺼진 상태에서도 final state를 만들 수 있다는 것이다.**

나의 알고리즘 논리

1) 첫 줄에 대해서만 모든 경우의 수를 다 보는 **완전 탐색**을 진행한다.

2) 첫 줄의 경우의 수를 가지고 **나머지 행에 대해서 불을 끄는 작업**을 진행한다.

3) 두 번째 줄부터 불을 끄는데, 불을 끄는 작업은 **현재 행에서 한 칸 위쪽 행이 불이 켜져있으면 끈다.**

4) 맨 마지막 행까지 반복하고, **모두 불이 꺼진 상태라면 그것은 가능한 경우**이다.

5) 2^n 경우를 모두 확인하고 거기서 **최소로 불을 꺼서 만들 수 있는 경우**를 출력한다.

이렇게 되면 같은 버튼이 두 번 이상 눌리지 않으며, 모두 꺼진 상태에서 final state를 만들 수 있는 최소 경우의 수를 구할 수 있게 됩니다.

2. 코드 설명

우선 저는 다음과 같은 전역 변수를 선언해주었습니다.

```
int result = 0; //최솟값 설정하기 위한 변수
vector<pair<int, int>> pos; //경우의 수 최솟값일 때 해당 좌표를 저장할 벡터
```

주석에 있듯이 **result는 최솟값을 저장할 변수**이고, **vector<pair<int, int>> pos는 정답의 좌표**를 담아낼 공간입니다.
다음 코드는 changeBulb 메소드인데요. 이 메소드는 해당 row, col값에 해당하는 전구를 켜거나 꺼주는 역할을 합니다. 간단한 코드라 설명할 부분이 없습니다.

```
void changeBulb(int** temp, int row, int col, int n) //상하좌우 + 현재위치 반전
{
    temp[row][col] = !temp[row][col]; //현재 위치

    if (row + 1 < n) //위
        temp[row + 1][col] = !temp[row + 1][col];

    if (row - 1 >= 0) //아래
        temp[row - 1][col] = !temp[row - 1][col];

    if (col - 1 >= 0) //왼쪽
        temp[row][col - 1] = !temp[row][col - 1];

    if (col + 1 < n) //오른쪽
        temp[row][col + 1] = !temp[row][col + 1];
}
```

그리고 핵심적인 코드인 checkBulb 메소드입니다.

```
int checkBulb(int** temp, vector<pair<int, int>> &tempPos, int n) //첫 줄 경우의 수만 고려하고 나머지 전구들에 대해 가능한지 체크
{
    int count = 0;

    for (int row = 1; row < n; row++)
        for (int col = 0; col < n; col++)
            if (temp[row - 1][col]) //바로 윗 행에 켜진 곳이 발견된다면
            {
                if (result < count) //만약 result에 저장된 최솟값 보다 현재 count 값이 더 크다면, return MAX;
                    return n * n + 1;

                count++;
                tempPos.push_back(make_pair(row, col));
                changeBulb(temp, row, col, n); //현재 [row][col]을 꺼줌으로써 위를 꺼줌
            }

    for (int row = 0; row < n; row++)
        for (int col = 0; col < n; col++)
            if (temp[row][col] == 1) //검사 했는데 켜진 경우가 하나라도 있으면
                return n * n + 1; //MAX값 반환

    return count;
}
```

위 메소드는 받은 배열 정보를 통해서, 만약 바로 **한 칸 위에 행이 불이 켜져있다면 불을 꺼주는 동작**을 수행합니다. 그리고 그때의 좌표를 벡터에 저장합니다. 만약 중간에 현재 **Best 최솟값인 result에 있는 값보다 count하는 값이 커지게 되면 MAX값을 리턴하고 함수를 종료**합니다. 여기서 MAX 값이 n*n + 1인 이유는 뒤에 main 코드에서 설명드리도록 하겠습니다.
또한 **모든 배열이 꺼진 상태인지 확인을 하고 만약 중간에 켜진게 하나라도 발견된다면 MAX값을 반환**합니다. 이 같은 **조건을 모두 통과했을 시에 count 값 즉 최솟값을 반환**하게 됩니다.

```
void printOutput(int** temp, int n)
{
    for (int row = 0; row < n; row++)
    {
        for (int col = 0; col < n - 1; col++)
        {
            if (temp[row][col] == 1)
                cout << "O ";
            else if (temp[row][col] == 0)
                cout << "# ";
        }

        //각 행 마지막 좌표는 뒤에 공백 없이
        if (temp[row][n - 1] == 1)
            cout << "O";
        else if (temp[row][n - 1] == 0)
            cout << "#";

        cout << endl;
    }
}
```

다음은 printOutput 함수입니다. 배열 정보와 n을 넘기면 우리가 구하고자 하는 output을 출력하는 간단한 출력 함수입니다.

```
int main()
{
    int count = 0; //켜진 전구 최솟값 카운트하기 위함
    vector<pair<int, int>> tempPos; //좌표값 저장할 벡터

    int n;
    cin >> n;

    result = n * n + 1; //MAX값 설정

    //2차원 동적 배열 할당
    int** arr;
    arr = new int* [n];
    for (int i = 0; i < n; i++)
        arr[i] = new int[n];

    //2차원 동적 배열 할당
    int** temp;
    temp = new int* [n];
    for (int i = 0; i < n; i++)
        temp[i] = new int[n];

    //0으로 초기화
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            arr[i][j] = 0;
```

이제 main 함수입니다.

count는 final state로 만들기 위해 눌러야 하는 전구 갯수의 최솟값 카운트를 하기 위한 변수입니다.
temPos는 좌표를 저장할 임시 변수이고 count가 Best 최솟값을 저장하는 전역 변수인 result 보다 작다면 tempPos에 있는 **벡터값을 전역변수 pos에 저장**해줍니다.

n에 사용자가 입력을 하면, result 값은 초기에 n*n+1로 설정해줍니다. **왜 MAX 값이 n*n+1이냐면. 사전조사에서 한번 누른 전구는 다시 누를 수 없다고 조건을 제시했습니다. 이는 그렇다면 최솟값이 조건이 깨져버리기 때문입니다. 그래서 n*n+1 값은 전부 다 누르는 횟수 + 1이기 때문에 알고리즘 계산에선 나올 수가 없는 숫자라 MAX 값으로 설정해주었습니다.**

그 밑에 for문들은 입력받은 n에 대하여 2차원 배열을 할당해주고 0으로 초기화 해주는 것입니다.
참고로 arr 배열에 입력을 받을 것이고 temp배열은 계속해서 최솟값을 찾아나갈 때 사용하는 임시 배열입니다.

```
//O# 정보 입력
for(int i = 0; i < n ; i++)
{
    char c;
    for (int j = 0; j < n; j++)
    {
        cin >> c;
        if (c == 'O')
            arr[i][j] = 1; //켜짐 표시
    }
}
```

final state를 입력받아서 배열에 0과 1의 상태로 저장합니다. 0은 전구가 꺼진 상태이고 1인 켜진 상태입니다.

```
for (int i = 0; i < (int)pow(2, n); i++) //첫 줄의 모든 경우의 수 확인하기 위해 2^n 경우의 수
{
    //원본 배열값을 temp로 복사
    for (int row = 0; row < n; row++)
        for (int col = 0; col < n; col++)
            temp[row][col] = arr[row][col];

    count = 0;
    tempPos.clear();

    for (int j = 0; j < n; j++)
        if (i & (int)pow(2, j)) //첫 줄 10개를 누를지 말지 판단 - 비트마스킹 i비트에 j가 포함되어 있는지 비트가 1이면 존재, 0이면 없음
        {
            count++;
            tempPos.push_back(make_pair(0, j)); //벡터에 배열 저장
            changeBulb(temp, 0, j, n); //[0][j] 켜줌
        }

    count += checkBulb(temp, tempPos, n); //첫 줄을 제외한 나머지 행에서 카운트
```

아까 알고리즘 논리 부분에서 언급한대로 2^n에 가능한 모든 경우의 수를 첫줄에서 확인합니다.
바로 앞에서 만나는 2중 for문은 원본 배열 arr 값을 temp 배열에 복사합니다.
그리고 count를 0으로, tempPos를 빈 상태로 만들어줍니다.

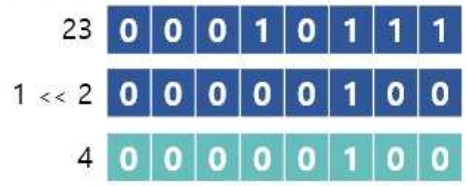
여기서 for문을 돌리는데 바로 나오는 if문 조건이 경우의 수를 확인하는 핵심 코드입니다.

저 부분은 비트마스킹이라는 기법이 들어간 코드입니다.
for문을 돌고 있는 2^i인 경우일 때 경우의 수를 의미하는데

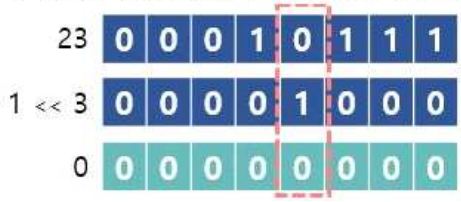
조회

이진수 10010에서 i번째 비트가 1인지 혹은 0인지 어떻게 확인할 수 있을까
확인할 숫자를 n이라고 하고 i번째 비트를 확인하고 싶다
`n & (1 << i)`가 0이라면 i번째 비트는 0이다
반대로 0보다 크다면 i번째 비트는 1이다

- `23 & (1 << 2)`
  23의 2번째 비트를 조회한다면 23과 `1 << 2`를 &연산한다
  그 결과는 0보다 크므로, 23에 2번째 비트는 1이라는 것이다



- `23 & (1 << 3)`
  23의 3번째 비트를 조회한다면 23과 `1 << 3`를 &연산한다
  그 결과는 0이고, 23의 3번째 비트는 0이다



앞서 요구되는 사전지식은, 2^n을 두가지로 표현할 수 있는데 **pow(2, n)와 1<<n**으로 표현 가능합니다.
제 코드에서는 전자의 방식으로 사용하였습니다.

이처럼 n이 23인 경우에 비트로 표현하면 [0 0 0 1 0 1 1 1]이 되는데,
1 << 2를 하면 두 번째 비트가 같은지 아닌지를 보고 싶다는 것이고, 1<< 3을 하면 3번째 비트가 같은지 아닌지 보고 싶다는 것입니다. **비트가 같으면 1 다르면 0을 조회**할 수 있습니다.

우리는 2^n가지 경우를 보고 있기 때문에 모든 경우에 대해서 비트마스킹-조회 기법을 사용하여
첫째 행에서 해당 전구가 꺼졌는데 켜졌는지 조회를 할 수 있게 됩니다.

그래서 만약 1이라면, 해당 배열의 좌표가 켜져있는 상태의 경우의 수라면, count 값을 증가시켜주고 해당 좌표를 tempPos에 넣습니다. 그리고 changeBulb를 통해 전구를 켜줍니다.

이후 앞에서 말씀드렸던 **checkBulb로 나머지 행에 대한 전구 ON/OFF 횟수를 count 변수에 더해줍니다.**

```
        if (count < result) //카운트한 값이 최솟값이라면
        {
            //전역변수 pos Vector에 tempPos 좌표값 복사
            pos.clear();
            copy(tempPos.begin(), tempPos.end(), back_inserter(pos)); //Deep Copy

            //result 최솟값으로 업데이트
            result = count;
        }
    }

    if (result == n * n + 1) //모든 경우에 대해 업데이트가 한번도 안된 경우
        cout << "no solution." << endl;
    else
    {
        //temp 배열 재사용
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                temp[i][j] = 0;

        //mark 해놓은 좌표들 1로 표시
        for(int i = 0 ; i < pos.size(); i++)
                temp[pos[i].first][pos[i].second] = 1;

        //결과 출력
        printOutput(temp, n);
    }
```

만약 카운트한 값이 현재 result 보다 작은 값이라면, 최소 경우의 수, **즉 Best Solution이므로 해당 좌표 값들을 전역 벡터 pos에 저장해주고 count값을 result에 저장**해줍니다.
여기까지가 2^n번 반복하는 for문이었습니다.

이후 result 값이 n * n +1 즉 MAX 값이라면, 입력 받은 final state에서 불을 모두 끌 수 있는 경우의 수가 하나도 없기 때문에 그런 것이므로 "no solution."을 출력합니다.

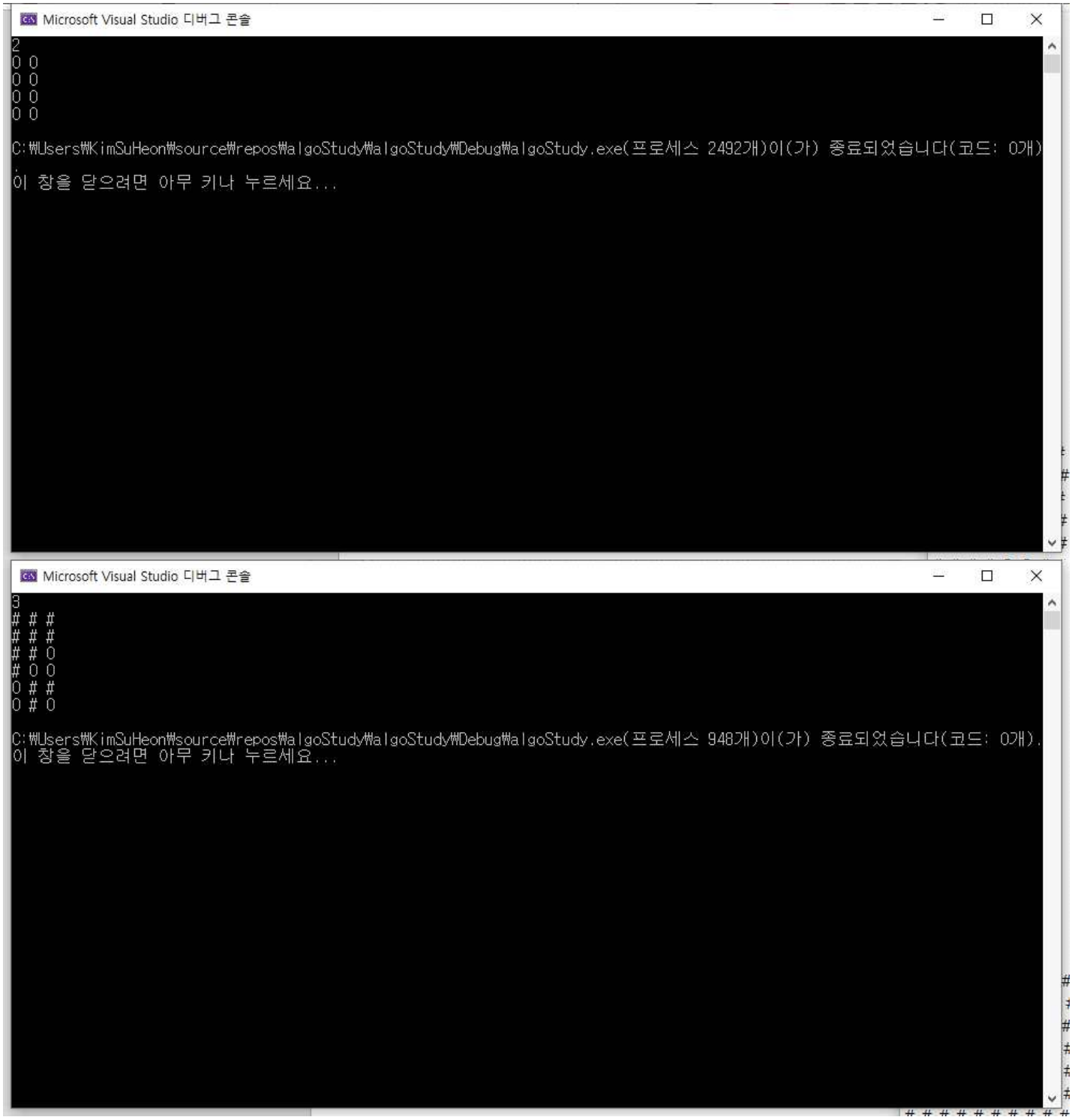
나머지 경우는 경우의 수가 존재하는 케이스 이므로 **temp 배열에 pos에 저장해둔 좌표를 1로 표시하고 printOutput 함수로 결과를 출력하여 output을 보여줍니다.**

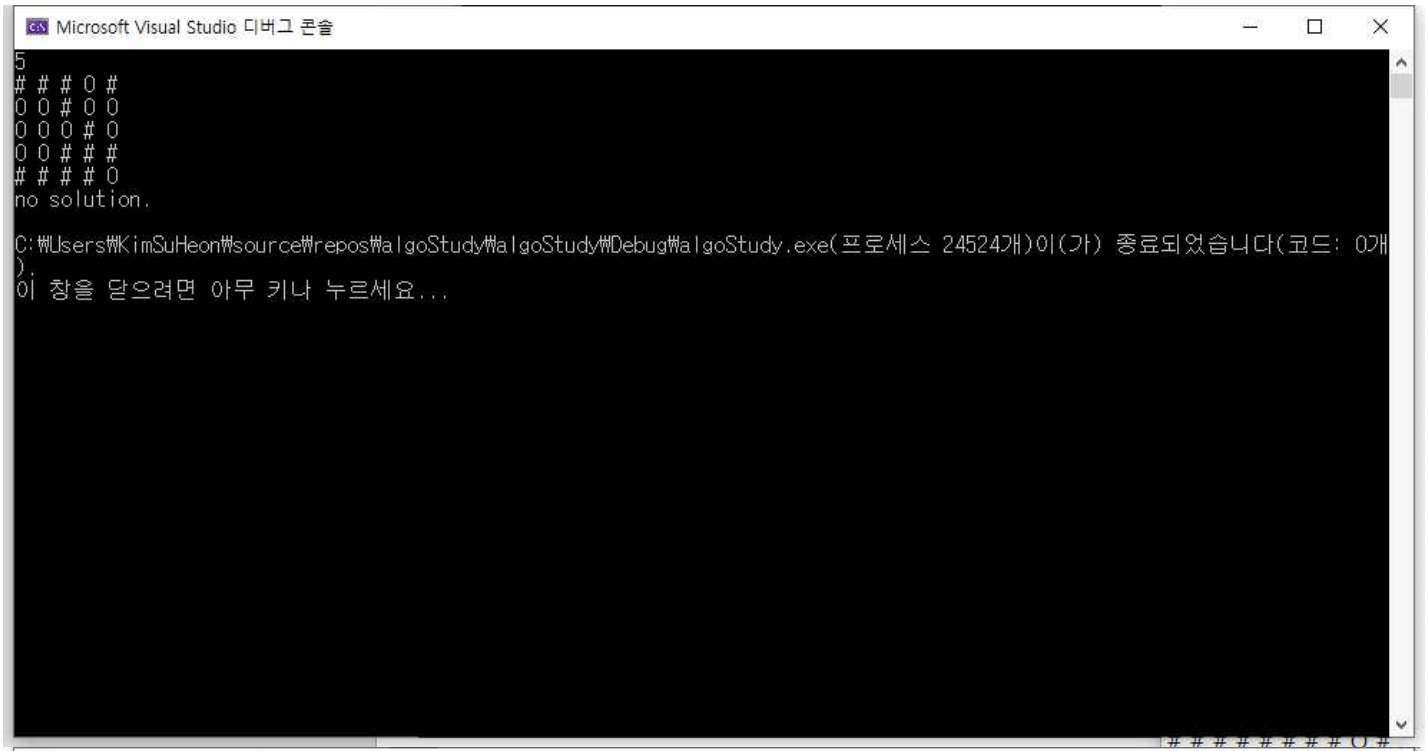
```
    //동적할당 메모리 해제
    for (int i = 0; i < n; i++)
    {
        delete[] arr[i];
        delete[] temp[i];
    }
    delete[] arr;
    delete[] temp;

    return 0;
}
```

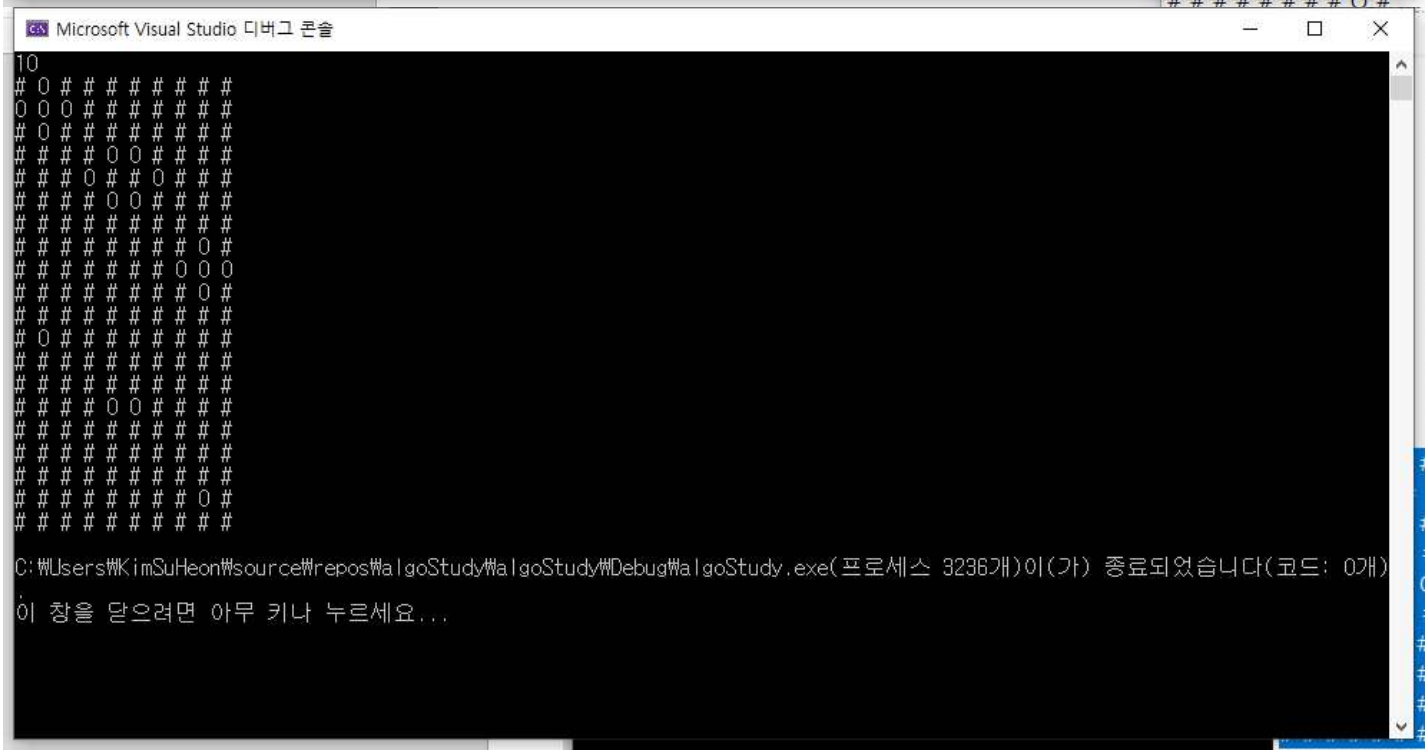
이후 사용하였던 동적 메모리 할당을 delete[] 해주고 프로그램을 종료합니다.

해당 알고리즘의 시간복잡도는 2^n for문 안에서, 2중 for문들만 존재하기 때문에 **(2^n * n * n)**입니다.

3. 테스트 케이스


Microsoft Visual Studio 디버그 콘솔
2
0 0
0 0
0 0
0 0
C:₩Users₩KimSuHeon₩source₩repos₩algoStudy₩algoStudy₩Debug₩algoStudy.exe(프로세스 2492개)이(가) 종료되었습니다(코드: 0개)
.
이 창을 닫으려면 아무 키나 누르세요...
Microsoft Visual Studio 디버그 콘솔
3
# # #
# # #
0
0 0
0 # #
0 # 0
C:₩Users₩KimSuHeon₩source₩repos₩algoStudy₩algoStudy₩Debug₩algoStudy.exe(프로세스 948개)이(가) 종료되었습니다(코드: 0개).
이 창을 닫으려면 아무 키나 누르세요...


Microsoft Visual Studio 디버그 콘솔
5
0
0 0 # 0 0
0 0 0 # 0
0 0 # # #
0
no solution.
C:₩Users₩KimSuHeon₩source₩repos₩algoStudy₩algoStudy₩Debug₩algoStudy.exe(프로세스 24524개)이(가) 종료되었습니다(코드: 0개
).
이 창을 닫으려면 아무 키나 누르세요...


Microsoft Visual Studio 디버그 콘솔
10
0 # # # # # # #
0 0 0 # # # # # # #
0 # # # # # # #
0 0 # # #
0 # # 0 # #
0 0 # # #
# # # # # # # # # #
0
0 0 0
0
# # # # # # # # # #
0 # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
0 0 # # #
# # # # # # # # # #
# # # # # # # # # #
# # # # # # # # # #
0
# # # # # # # # # #
C:₩Users₩KimSuHeon₩source₩repos₩algoStudy₩algoStudy₩Debug₩algoStudy.exe(프로세스 3236개)이(가) 종료되었습니다(코드: 0개)
.
이 창을 닫으려면 아무 키나 누르세요...


Microsoft Visual Studio 디버그 콘솔
20
C:₩Users₩KimSuHeon₩source₩repos₩AlgoStudy₩AlgoStudy₩Debug₩AlgoStudy.exe(프로세스 15528개)이(가) 종료되었습니다(코드: 0개
).
이 창을 닫으려면 아무 키나 누르세요...

4. 간단한 소감

저는 이번 과제를 하면서, 알고리즘 실력이 한층 더 성장했다고 느꼈습니다.
첫번째로는 STL을 많이 사용하다보니 자연스럽게 STL 사용하게 되며, 예전처럼 사용 방법을 하나하나 찾아보는 것이 아닌, 머리가 기억하여 즉각 적용이 가능한 경지에 이르게 된 것 같습니다. 과제에선 사용하지 않았지만, STACK, VECTOR, MAP, QUEUE, SET 등을 어떤 상황에 어떻게 써야할지 감을 많이 익혔다고 느꼈습니다.

두 번째는 시간 복잡도에 항상 고려해야한다는 생각을 갖게 되었습니다. 과거에 코딩할 때는 시간 복잡도를 하나도 고려하지 않고, 완성에 목적을 두고 코딩을 하였습니다. 지금 생각해보면 n^3의 경우도 있었고, 시간을 더 줄일 수 있었지만 직관적으로 코딩을 해나갔던 내 자신을 되돌아볼 수 있었습니다. 앞으로 코딩할 때는 조금 더 시간을 줄일 수 없을까? 라는 생각을 하며 구현에 임할 것 같습니다.

세 번째로는 비트마스킹이란 개념을 알게 되었습니다. 처음에 어떻게 해야할지 몰라서 검색을 하고 있었는데 비트마스킹이라는 개념을 알게 되어 바로 공부를 시작했습니다. 이해가 가지 않았지만, 이해될 때까지 반복하며 A4용지에 예시를 그려가며 비트가 어떻게 연산이 되고 이러한 수식이 나올 수 있는지 공부하다 보니까 점점 개념이 확립되었습니다. 나중에 심화된 알고리즘을 풀다보면 이 기법을 떠올려 적용하고 풀어나갈 수 있다는 자신감을 얻게 되었습니다. 이 또한 알고리즘 실력 향상에 도움이 된 것 같습니다.

이 같은 경험을 통해서 한층 더 성장했다고, 생각합니다. 벌써 알고리즘 마지막 과제라고 생각하니 기쁘도 하면서 시원섭섭한 두 가지 감정이 교차합니다. 마지막으로 유익한 과제를 내주신 교수님, 조교님께 감사드립니다.