국민대학교
전자정보통신대학
컴퓨터공학부

알고리즘
설계 프로젝트

프로젝트 명	Solving Sudoku and Crossword puzzle with Dancing Links
팀 명	*sixA*
문서 제목	결과보고서

Version	**1.4**
Date	2015-DEC-21

팀원	이 소령 (조장)
	신 은영
	이 지연
	이 창우
	이 한별
	임 솔빈
지도교수	최 준수 교수

문서 정보 / 수정 내역

Filename	최종보고서-SS&CP_with_DLX.docx
원안작성자	이소령, 이창우, 임솔빈
수정작업자	이소령, 이지연, 이창우, 이한별, 임솔빈

수정날짜	대표수정자	Revision	추가/수정 항목	내 용
2015-12-20	이창우	1.0	최초 작성	
2015-12-21	이소령	1.1	내용 추가	수행 내용, 최종보고서 본문
2015-12-21	이지연	1.2	내용 추가	최종보고서 본문
2015-12-21	임솔빈	1.3	내용 추가	수행 내용 및 중간 결과
2015-12-21	이한별	1.4	내용 추가	크로스워드 내용 추가

목 차

1 프로젝트 목표 4
2 수행 내용 및 중간결과 5
2.1 계획서 상의 연구내용 5
2.2 수행내용 6
3 최종보고서 본문 9
4 자기평가 19

1 프로젝트 목표

1.1 목표

Donald Knuth의 Dancing Links 알고리즘을 이용하여 Sudoku 퍼즐 문제를 해결하고 더 나아가 크로스워드 퍼즐 문제를 해결하는 알고리즘을 고안한다.

1.2 프로젝트 추진 배경 및 필요성

Donald Knuth의 DLX 알고리즘으로 구현되어 있는 프로젝트로는 N-Queens, Sudoku, Tetramino-3D 가 있다. 이 중에서 대표적 문제인 Sudoku를 해결하는 코드를 작성해보고 더 나아가 크로스워드 퍼즐 문제도 해결해보고자 한다.

DLX 알고리즘은 재귀, 비 결정적, 깊이 우선, 역 추적 알고리즘에 대한 모든 솔루션을 찾아 정확한 해결을 제시하는데, 알고리즘의 유사한 반복은 춤 동작이 연결되어 있는 것과 같은 방식으로 작동하기 때문에 이러한 이름이 붙여졌다고 한다.

Sudoku에 대한 풀이는 많이 찾아볼 수 있지만 크로스워드 퍼즐 문제 같은 경우는 DLX 알고리즘으로 해결해둔 경우를 찾아보기 힘들기 때문에 직접 만들어 볼 필요성이 존재한다. Sudoku와 크로스워드 퍼즐의 공통점은 문제를 해결함에 있어 한 번에 해결되지 않고, DLX 알고리즘의 특성인 재귀적인 방식, 문제를 푸는 매 순간마다 비 결정적이고 깊이가 우선되며 역 추적이 필요하다는 것이다. 이를 통해 문제를 해결할 수 있다고 생각한다.

2.3 기대효과 및 활용방안

DLX 알고리즘을 크로스워드 퍼즐 문제까지 해결하도록 확장하게 되면 비슷한 종류의 퍼즐 문제 등을 해결하는데 큰 도움이 될 것으로 사료된다.

2 수행 내용 및 중간결과

2.1 계획서 상의 연구내용

- **Sudoku**
  - 아무것도 입력되어 있지 않은 빈 칸을 0으로 둔다.
  - 0이 입력된 위치로 가서 이 위치의 행과 열, 같은 블록을 체크하여 이 위치에 들어갈 수 있는 숫자를 정한다. 예를 들어, 오른쪽 행렬의 원소 (3,2)에 들어 갈 수 있는 수는 1이다.

1	2	3	0
0	4	0	2
2	3	4	1
4	1	2	0

  - 해당 위치에 들어 갈 수 있는 숫자가 여러 개라면 Recursive하게 수행하여
  - 행렬에 0이 없을 때까지 위의 조건에 부합하게 backtracking하여 행렬을 완성한다. 만약, 조건에 부합하는 숫자가 없는 경우 다시 back하여 다른 숫자를 넣어보고 Recursion을 수행한다.
  - 이러한 방식으로 Sudoku를 빈칸 없이 전부 완성하면 함수를 종료시킨다.
    위의 전체적인 알고리즘 이외에도, 추가적으로 프로그램을 처음 시작할 때, 난이도를 선택하는 메시지를 띄워 사용자에게 난이도를 선택하는 입력을 받는다. 그리고 Sudoku 판 아래에 난이도에 따른 남은 힌트 수를 출력하여 나타내고, 힌트를 사용할 때 마다 차감한다.
- **Crossword Puzzle**
  - 입력 값으로 사용될 단어들을 내림차순으로 정렬하여 저장한다.
    - 퍼즐 판의 길이에 맞는 임의의 한 단어를 퍼즐 판에 입력한다.
    - 그리고 다음 단어를 입력한다.
    - 퍼즐 판에 이미 있는 단어를 통해 이 단어와 교차해도 문제되지 않는 단어를 찾는다.
    - 만약 교차할 수 있는 단어를 찾는다면, 퍼즐 판에 모든 단어가 루프를 돈다. 그리고 새 단어가 조건에 맞는지 확인한다.
    - 만약 새 단어가 조건에 맞는다면(정답), 2번째 단계로 이동하고, 조건에 맞지 않다면 3번째 단계로 이동한다.
    - 퍼즐 판이 모두 채워지거나 퍼즐 판에 채워질 수 있는 단어가 없을 때까지 루프를 계속 돈다.
  - 위의 전체적인 알고리즘 이외에도, 추가적으로 프로그램을 처음 시작할 때, 난이도와 크로스워드 퍼즐의 유형을 선택하는 메시지를 띄워 사용자에게 입력을 받는다.

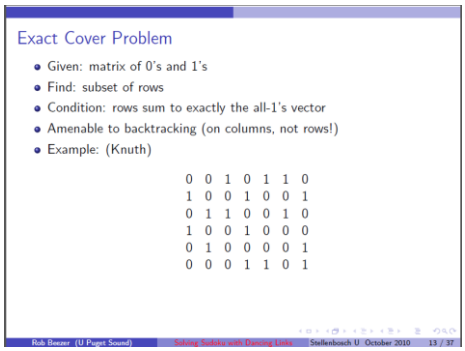
2.2 수행내용

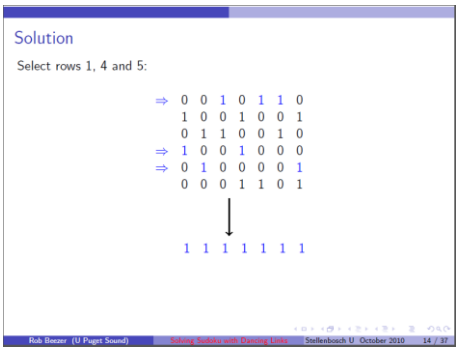
계획 인원	수행 내용	실제 참여 인원
이소령, 임솔빈	Dancing Links Algorithm 분석	신은영 외 5 인(전체)
임솔빈	Sudoku 소스 코드 분석	이소령, 이창우
신은영, 이한별	Sudoku 소스 코드 수정	이소령
이지연, 이창우	크로스워드 퍼즐 알고리즘 고안	신은영 외 5 인(전체)
이소령, 이한별	크로스워드 퍼즐 알고리즘 구현	이창우, 이한별

수행 내용 상세

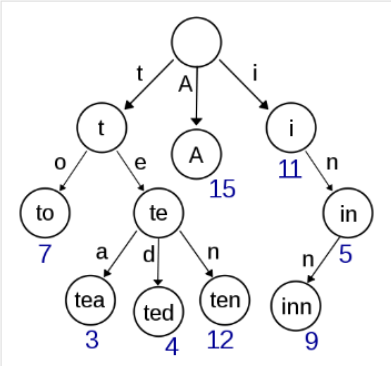
1. **Dancing Links Algorithm 분석**
   - **Algorithm X 조사 및 연구**
2. **Sudoku 소스코드 분석 및 수정**
   - **오픈 소스 검색 및 분석**
   - **채점서버 환경에 맞게 코드 수정**
3. **Crossword 소스코드 고안 및 구현**
   - **30만 개의 단어 저장과 검색을 위한 효율적 자료구조 탐색**
   - **크로스워드 퍼즐 맞추는 시간 단축을 위한 코드 작성**

- **Dancing Links Algorithm 분석**
  - **Algorithm X 조사 및 연구**
    - **팀 프로젝트 실을 대여하여 Dancing Links Algorithm 이해를 위한 부차적 대안으로 Algorithm X 를 팀원 모두 참여하여 연구 하였음.**

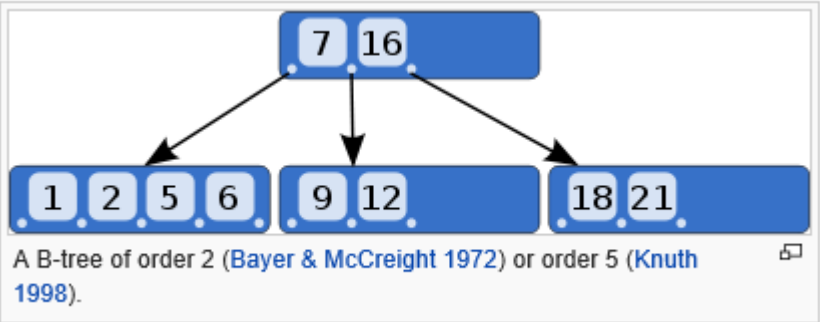




- **Sudoku 소스코드 분석 및 수정**
  - **인터넷으로 여러 오픈 소스 코드를 검색한 후 비교, 분석**
- **Crossword 소스코드 고안 및 구현**
  - **문제를 해결하기 위해 합당한 자료구조 고찰**
    - **Trie: 주로 스트링을 가지는 동적 셋 혹은 연관 배열을 저장하는데 사용되는 정렬된 트리 자료구조를 의미한다.**



      - **단점: 접두사가 부분적으로 겹치는 경우 의도하지 않은 데이터가 생길 수 있다.**
    - **B-tree: 이진 트리를 확장한 것으로 이진 트리는 하나의 노드가 가질 수 있는 자식 노드의 수가 최대 2개지만 B-tree는 2개 이상을 가질 수 있다. 모든 노드에 있는 값들은 정렬되어 있는 상태이며 order를 나타내는 숫자인 m을 가질 수 있다. 이 B-tree 를 B-tree of order m 이라고 한다.**



A B-tree of order 2 (Bayer & McCreight 1972) or order 5 (Knuth 1998).

- 단점: 삽입과 검색 알고리즘 자체가 어려워서 구현이 쉽지 않다.

◆ Array(2D): 순서대로 번호가 붙은 원소들이 연속적인 형태로 구성된 구조를 뜻하며, 이때 각 원소에 붙은 번호를 흔히 첨자(인덱스, index)라고 부른다. 원소들이 연속적으로 배치되어 있기에, 임의의 첨자로 각 원소를 접근하는 데에 드는 시간 복잡도는 O(1)이다. 따라서 임의 접근(random access)이 가능한 자료구조에 속한다.

	1열	2열
1행	arr[0][0]	arr[0][1]
2행	arr[1][0]	arr[1][1]
3행	arr[2][0]	arr[2][1]

[그림 7-5] 2차원 배열의 표현

- 단점: 한 원소에 접근하는데 걸리는 시간이 일반적으로 O(n^2)이다.

3 최종보고서 본문

3.1 Sudoku Source Code

3.1.1 isAvailable: 0~9 중에 puzzle에 들어갈 수 있는지 여부 판단

```
int isAvailable(int puzzle[][9], int row, int col, int num)
{
    int rowStart = (row / 3) * 3;
    int colStart = (col / 3) * 3;
    int i;

    for (i = 0; i<9; ++i)
    {
        if (puzzle[row][i] == num) return 0;
        if (puzzle[i][col] == num) return 0;
        if (puzzle[rowStart + (i % 3)][colStart + (i / 3)] == num) return 0;
    }
    return 1;
}
```

① num은 들어갈 숫자, 0~9까지 들어올 수 있음.

② if (puzzle[row][i] == num) -> 가로에 num이 있는지 확인

③ if (puzzle[i][col] == num) -> 세로에 num이 있는지 확인

④ if (puzzle[rowStart + (i % 3)][colStart + (i / 3)] == num)
-> 사각형 안에 num이 있는지 확인

⑤ return 1; -> 모두 num이 없으면 num이 들어갈 수 있으므로 1을 반환한다.

3.1.2 fillSudoku: 재귀적으로 현재 퍼즐의 위치에 숫자 기록 가능 확인

```
int fillSudoku(int puzzle[][9], int row, int col)
{
    int i;
    if (row<9 && col<9)
    {
        if (puzzle[row][col] != 0)
        {
            if ((col + 1)<9) return fillSudoku(puzzle, row, col + 1);
            else if ((row + 1)<9) return fillSudoku(puzzle, row + 1, 0);
            else return 1;
        }
        else
        {
            for (i = 0; i<9; ++i)
            {
                if (isAvailable(puzzle, row, col, i + 1))
                {
                    puzzle[row][col] = i + 1;
                    if ((col + 1)<9)
                    {
                        if (fillSudoku(puzzle, row, col + 1)) return 1;
                        else puzzle[row][col] = 0;
                    }
                    else if ((row + 1)<9)
                    {
                        if (fillSudoku(puzzle, row + 1, 0)) return 1;
                        else puzzle[row][col] = 0;
                    }
                    else return 1;
                }
            }
        }
        return 0;
    }
    else return 1;
}
```

⑥ if (row < 9 && col < 9) -> 9보다 작을 때만 수행된다.

⑦ if (puzzle[row][col] != 0) -> 해당하는 자리가 비어있지 않을 때

⑧ if ((col + 1) < 9) -> col과 row를 확인해서 스도쿠 규칙에 맞으면 1, 아니면 0을 반환하며 0이 반환되었다는 의미는 현재 자리에 들어가있는 값이 잘못되었음을 의미.

⑨ 해당하는 자리가 비어있으면, 0~9 숫자를 넣어보면서 가능 여부 판단

⑩ for (i = 0; i < 9; ++i) -> i는 칸 안에 넣어볼 숫자, 0~9까지의 범위를 가짐.

⑪ if (isAvailable(puzzle, row, col, i + 1)) -> i가 isAvailable 함수를 통해 판단됨.

⑫ puzzle[row][col] = i + 1; -> isAvailable이 1을 반환하면 i를 해당 자리에 삽입.

⑬ if ((col + 1) < 9) -> i를 넣고 나면 다시 col과 row를 확인하여 규칙에 위배되는지 판단한다. 0이 반환될 경우 넣은 i의 값이 잘못되어있다는 의미이다.

⑭ if (fillSudoku(puzzle, row, col + 1)) -> 재귀적인 방법을 통해 확인하고 0이 반환되면 해당 위치에 i를 넣을 수 없다는 뜻이므로 다시 0으로 해당 칸을 채운다.

⑮ 만약 col 과 row 모두를 만족하면 1 을 반환한다.

3.1.3 main: 파일 입출력을 통해 입력 값을 저장하여 fillSudoku를 호출

```
int main()
{
    FILE *fp = fopen("input.txt", "r");
    int puzzle[9][9] = { 0, };
    int i, j, T;
    for (fscanf(fp, "%d", &T); T--;)
    {
        for (i = 0; i < 9; i++)
        {
            fscanf(fp, "%d %d %d %d %d %d %d %d %d\n",
                &puzzle[i][0], &puzzle[i][1], &puzzle[i][2],
                &puzzle[i][3], &puzzle[i][4], &puzzle[i][5],
                &puzzle[i][6], &puzzle[i][7], &puzzle[i][8]);
        }

        if (fillSudoku(puzzle, 0, 0))
        {
            for (i = 1; i < 10; ++i)
            {
                for (j = 1; j < 10; ++j)
                {
                    printf("%d ", puzzle[i - 1][j - 1]);
                }
                printf("\n");
            }
        }
    }
    fclose(fp);
    return 0;
}
```

① fscanf 함수를 통해 파일에서 퍼즐의 값들을 읽어온다.

② fillSudoku 함수 호출 후 1이 반환되면 해당하는 값을 출력한다.

③ fillSudoku는 배열의 [0][0] 위치부터 기능을 수행한다.

3.1.4 입력 파일 내용과 출력 화면

➢ 입력 파일

input.txt sudoku.c

```
2
6 0 7 0 0 0 8 0 1
0 0 5 8 0 1 3 0 0
0 0 0 4 0 3 0 0 0
2 0 0 5 0 4 0 0 3
0 7 0 0 0 0 0 8 0
5 0 0 3 0 7 0 0 2
0 0 0 1 0 6 0 0 0
0 0 6 9 0 5 4 0 0
9 0 3 0 0 0 6 0 5
3 0 1 7 0 0 0 8 5
0 6 7 0 0 9 1 4 0
2 0 0 0 0 4 0 0 3
0 3 0 9 0 7 0 6 0
0 0 6 5 1 8 0 0 0
8 7 0 0 3 0 0 9 1
7 0 8 0 0 0 4 0 6
0 0 0 6 7 0 0 5 0
6 0 3 0 0 5 0 1 0
```

➢ 출력 화면

C:₩Windows₩system32₩cmd.exe

```
6 3 7 2 5 9 8 4 1
4 2 5 8 6 1 3 7 9
1 8 9 4 7 3 2 5 6
2 6 8 5 1 4 7 9 3
3 7 1 6 9 2 5 8 4
5 9 4 3 8 7 1 6 2
7 5 2 1 4 6 9 3 8
8 1 6 9 3 5 4 2 7
9 4 3 7 2 8 6 1 5
3 4 1 7 6 2 9 8 5
5 6 7 3 8 9 1 4 2
2 8 9 1 5 4 6 7 3
1 3 5 9 2 7 8 6 4
4 9 6 5 1 8 3 2 7
8 7 2 4 3 6 5 9 1
7 5 8 2 9 1 4 3 6
9 1 4 6 7 3 2 5 8
6 2 3 8 4 5 7 1 9
계속하려면 아무 키나 누르십시오 . . .
```

3.2 Crossword

3.2.1 Source code

```
#include <iostream>
#include <string>
#include <fstream>
#include <vector>
#include <ctime>
#include <algorithm>
using namespace std;

#define WIDTH   1
#define HEIGHT  2
#define INIT    3
#define UPDATE  4
#define END     999
#define KEEP    6
#define NON     -1
#define MAX_LENGTH  50

int row, col;
int wCount;     // 가로 갯수
int hCount;     // 세로 갯수
Map **map;      // FILED
ifstream file;  // FILE
vector<string> dic[MAX_LENGTH];
vector<char> history;
```

```
class Map{
public:
    char data;
    vector<int> mark;
    Map(char _d){
        data = _d;
    }
    Map() {}
    void setMark(int num){
        mark.push_back(num);
    }
};
```

///

///

```
class InsWord{
public:
    int x, y;   // 시작 좌표
    int len;    // 길이 getLen() 리턴값
    int type;   // 가로 세로 타입
    int dicPos = 0;         //  setWord 시 사전 위치 포인트
    int markPos = 0;        //  mark 위치
    int number;             // 자신의 넘버, 교차 지점 mark 용도
    Map **&ref_map = map;   // 맵 레퍼런스
    vector<string> wordList;    // 매칭 사전 원본 리스트
    vector<string> filterList;  // 매칭 사전 갱신 리스트
    vector<int> crossList;          // 자신과 교차된 넘버 목록 (하나인 경우)
    string preWord;             // setWord()하기 이전 백업된 스트링
    bool change = true;     // 바뀐 신호 처리

    InsWord(int _x, int _y, int _type, int num){
        x = _x;
        y = _y;
        type = _type;
        number = num;
        len = getLen();

    }
```

```
    int getMarkPos(){ return markPos; }
    void incMarkPos(){ markPos++; }
    void decMarkPos(){ markPos--; }
    void initMarkPos(){ markPos = 0; }
    int getMarkNum(){
        incMarkPos(); return crossList[markPos];
    }
    void getCross();    // 교차점 조사
    int checkWord();    // 현재 워드 _칸 조사
    // 단어 사전 리스트 갱신
    int setList(string findWord, int pos, vector<string> temp, int env);

    // 단어 사전 리스트 초기화
    int initList();

    int setWord(); // map에 단어 기입
    void rollBack(); // 기입한 단어 복구
    string getWord(); // 현재 객체 단어
    int getLen(); // 현재 객체 길이
    void changeSig(); // 바뀐 신호를 받으면 list와 사전 포인터 갱신

    friend std::ostream& operator<<(std::ostream& out, const InsWord *p){
        return out;
        // WORD 테스트 오퍼레이터
    }
};
```

////

///

```
int InsWord::checkWord(){
    // _칸 조사 있으면 계속 없으면 끝을 리턴
    string word = getWord();
    for (int i = 0; i < len; i++){
        if (word[i] == '_') return KEEP;
    }
    return END;
}
int InsWord::setList(string findWord, int pos, vector<string> temp,
    // 만약 _칸이 없으면 END 리턴
    if (checkWord() == END)
        return END;
    // 최초 작성
    if (env == INIT){
        wordList.clear();
        wordList = temp;
        filterList = wordList;
    }
    // 갱신
    if (env == UPDATE){
        filterList.clear();
        filterList = temp;
    }

    vector<string> newList; // 임시 리스트

    for (int i = pos; i < len; i++){
        if (findWord[i] != '_'){
            for (int j = 0; j < temp.size(); j++){
                if (findWord[i] == temp[j][i]){
                    newList.push_back(temp[j]);
                }
            }
            // 임시 리스트가 들어간 재귀
            return setList(findWord, i + 1, newList, env);
        }
    }
    // 자신의 영역 안에 변경점(change, getWord())을 기준으로 갱신함
    if (env == UPDATE)
        return filterList.size();
    else
        return wordList.size();
}
```

//

```
void InsWord::getCross(){
    int i = x, j = y;
    // 자신의 영역안의 맵 구조체를 탐색하면서 초기에 기록한
    // mark가 2 이상인 경우 교차 목록(crossList)에 기입함
    if (type == WIDTH){
        while (ref_map[i][j].data != '*'){
            if (map[i][j].mark.size() > 1)
            for (int k = 0; k < map[i][j].mark.size(); k++){
                crossList.push_back(map[i][j].mark[k]);
            }
            j++;
            if (j == row)
                break;
        }
    }
    if (type == HEIGHT){
        while (ref_map[i][j].data != '*'){
            if (map[i][j].mark.size() > 1)
            for (int k = 0; k < map[i][j].mark.size(); k++){
                crossList.push_back(map[i][j].mark[k]);
            }
            i++;
            if (i == col)
                break;
        }
    }
}
```

```
int InsWord::initList(){
    // 최초 맵 구성 갱신 용도
    wordList.clear();
    return setList(getWord(), 0, dic[len], INIT);
}
```

```
int InsWord::setWord(){
    // 사전 다 탐색했을 경우 END 리턴
    if (dicPos >= filterList.size())
        return END;
    // 만약 자신 구역 안에 변경 점이 있었을 경우 새로 갱신
    if (change == true){
        if (setList(getWord(), 0, wordList, UPDATE) == 0){
            return NON;
        }
        changeSig();
    }
    // 갱신된 리스트가 아예 없어서 막다른 길일 경우
    if (filterList.size() == 0){
        return NON;
    }
    // 이전 string 백업
    preWord = getWord();

    // map에 기입
    if (type == WIDTH)
    for (int i = y; i < len + y; i++){
        ref_map[x][i] = filterList[dicPos][i - y];
    }


    if (type == HEIGHT)
    for (int i = x; i < len + x; i++){
        ref_map[i][y] = filterList[dicPos][i - x];
    }

    // 사전 포인터 증가
    dicPos++;

    return KEEP;
}
```

//

```
void InsWord::rollBack(){
    // 백업 단어를 기입
    if (type == WIDTH)
    for (int i = y; i < len + y; i++){
        ref_map[x][i] = preWord[i - y];
    }


    if (type == HEIGHT)
    for (int i = x; i < len + x; i++){
        ref_map[i][y] = preWord[i - x];
    }
}
```

```
string InsWord::getWord(){
    string word;

    // 자신의 현재 영역 안의 단어를 생성하여 리턴
    if (type == WIDTH)
    for (int i = y; i < y + len; i++){
        word.append(1, ref_map[x][i].data);
    }

    if (type == HEIGHT)
    for (int i = x; i < x + len; i++){
        word.append(1, ref_map[i][y].data);
    }
    return word;
}
```

//

```
int InsWord::getLen(){
    // 영역 시작 지점부터 타입에 따른 방향으로 벽까지의 길이 리턴
    int i = x, j = y, _len = 0;
    if (type == WIDTH){
        while (ref_map[i][j].data != '*'){
            ref_map[i][j].setMark(number);
            _len++;
            j++;
            if (j == row)
                break;
        }
    }
    if (type == HEIGHT){
        while (ref_map[i][j].data != '*'){
            ref_map[i][j].setMark(number);
            _len++;
            i++;
            if (i == col)
                break;
        }
    }
    return _len;
}
```

```
void printMap(){
    for (int i = 0; i < row; i++){
        for (int j = 0; j < col; j++){
            cout << map[i][j].data;
        }
        cout << endl;
    }
}
```

```
int compute(vector<InsWord*> words){
    int pos = 0;

    while (1){
        if (words[pos]->setWord() == NON){
            cout << "ERORR" << endl;
            break;
        }
    //  //if (words[pos]->crossList.size() != 0){
    //      pos = words[pos]->getMarkNum();
    //  }
        system("cls");
        printMap();
        system("pause");
        pos++;
    }

    return END;
}
```

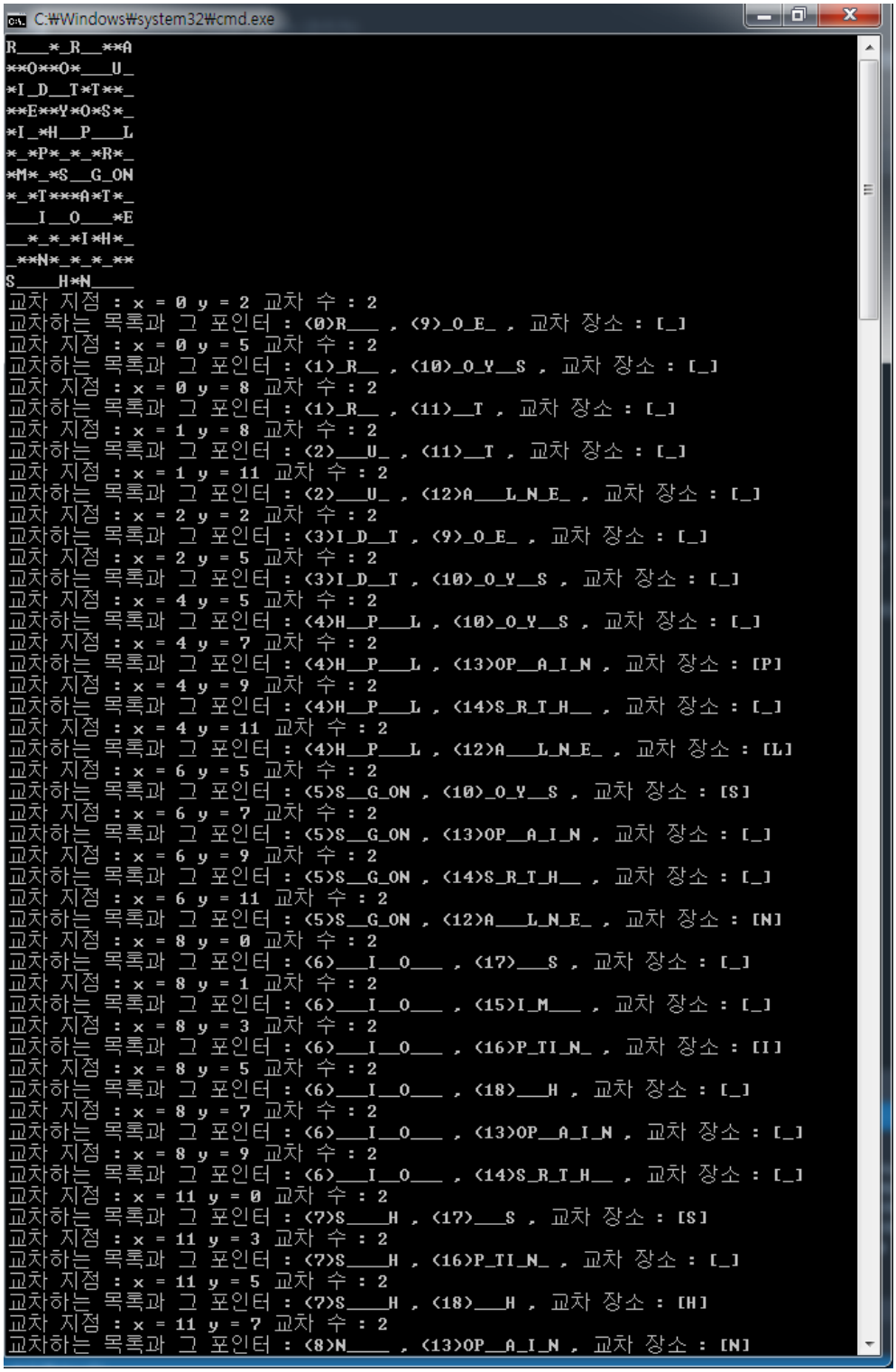
```
void InsWord::changeSig(){
    dicPos = 0;
    change = false;
}
```

```
void crossCheck(vector<InsWord*> words){
    for (int i = 0; i < row; i++){
        for (int j = 0; j < col; j++){
            if (map[i][j].mark.size() > 1){
                cout << "교차 지점 : x = " << i << " y = " << j << " " << "교차 수 : " << map[i][j].mark.size() << endl;
                cout << "교차하는 목록과 그 포인터 : ";
                for (int k = 0; k < map[i][j].mark.size(); k++){
                    cout << "(" << map[i][j].mark[k] << ")" << words[map[i][j].mark[k]]->getWord() << " , ";
                }
                cout << "교차 장소 : [" << map[i][j].data << "]";
                cout << endl;
            }

        }
    }

}
```

//

```
int main( ) {
    clock_t start = clock( );
    file.open("input.txt");

    int wordCount;
    file >> wordCount;

    for (int i = 0; i < wordCount; i++) {
        char *newWord = new char[MAX_LENGTH];
        file >> newWord;
        string *newString = new string(newWord);
        dic[newString->size( )].push_back(newWord);
        // 단어 길이번째 dic 칸에 넣는 분류
    }
    int cases;
    file >> cases;

    while (cases--){
        file >> row >> col;

        map = new Map*[row];
        for (int i = 0; i < row; i++){
            map[i] = new Map[col];
        }
        for (int i = 0; i < row; i++){
            for (int j = 0; j < col; j++)
                file >> map[i][j].data;
        }
        // 원본 출력
        printMap( );

        file >> wCount >> hCount;

        vector<InsWord*> words;
        for (int i = 0; i < wCount; i++){
            int x, y;
            file >> x >> y;
            InsWord *newWord = new InsWord(x - 1, y - 1, WIDTH, i);
            words.push_back(newWord);
        }
        // 가로 셋팅
        for (int i = 0; i < hCount; i++){
            int x, y;
            file >> x >> y;
            InsWord *newWord = new InsWord(x - 1, y - 1, HEIGHT, i + wCount);
            words.push_back(newWord);
        }
        // 세로 셋팅

        // 교차 지점 정보 테스트 출력
        //crossCheck(words);
        for (int i = 0; i < words.size( ); i++){
            words[i]->initList( ); // 단어 초기화
            words[i]->getCross( ); // 크로스 셋팅
        }

        compute(words);
        // 계산 부분
    }
    clock_t end = clock( ); // 시간 테스트

    cout << end - start << endl;

}
```

3.2.2 실행 화면



3.2.3 Source code 설명

먼저 사전을 불러와서 읽는다
읽을 때 그 길이를 구해서 길이에 해당하는 칸에 골라 넣는다 (2자, 3자, 4자 사전 따로).
x,y를 가진 구조체(InsWord)로 생성할 때 자신의 영역을 계산 getLen()
계산을 하면서 자신의 number를 map구조체 안에 vector mark로 남기고 간다.
나중에 이 vector 사이즈가 2 이상인 경우 그 곳은 교차 지점임을 알 수 있고
남은 number를 통해 해당 객체를 추적, 조작할 수 있다.
초기화가 끝나면 자신의 영역 안에 있는 vector mark들을 조사하여 교차점 목록을 정리한다 (crossCheck())

그 영역안에 현재 상태를 리턴 하는 것이 getWord()
처음 initList로 최초 사전을 등록한다.
자신 영역 len 길이에 맞는 dic을 찾아서 dic[len]을 참조하여
현재 상태 getWord()에 맞는 후보군을 wordList에 넣는다.

그 후 다른 곳에서 채워짐에 따라 교차점에 변화가 일어나면
해당 객체는 wordList가 전체 집합이므로 filterList를 갱신할 때 참조한다.

setWord()를 하면 filterList에서 dicPos 번째 위치 단어를 기입한다.
이 기입은 실제 map 레퍼런스를 가지고 있기 때문에 실시간으로 바뀌며
다른 객체에서 setWord()를 할 때 마다 바뀐 정보를 가지게 된다.
한칸 한칸 바뀔 때 해당 vector mark안에 number에 바뀌었다는 신호를 보내두게 된다.

이후 바뀌었다는 신호를 받았을 경우 (change = true) setWord()를 할 때
새로 filterList를 갱신하고 시작한다. 이 때 갱신된 filterList 사이즈가 0인 경우
매칭되는 단어가 하나도 없음을 의미하므로 막다른 길임을 알 수 있다.

이럴 경우 되돌아가서 다른 길을 찾도록 롤백 해야한다.

롤백하는 경우 연결된 mark리스트를 참조하여 뒤로 돌아가고
setWord()할 때 저장해둔 preString으로 되돌린다.

4 자기평가

우선 계획과는 달리 백트래킹 알고리즘이나 댄싱링크 알고리즘을 사용하고 싶었으나 사용하지 못했다. 그 이유는 다들 알고리즘에 대한 이해도 미흡과 시간 부족이다.

그래서 최대한 구현해보고자 한 것이 알고리즘 수업시간에 배운 recursion을 이용한 현재의 Sudoku 소스 코드이다. 사실 코드 전부를 우리 머리에서 짜내어 만든 것은 아니다. Google 검색을 통해 여러 소스 코드를 비교하고 분석해본 결과 현재의 짧고 간단한 코드를 이해하고 사용하는 것이 시간상 괜찮을 것이라는 판단 하에 그렇게 한 것이다.

시간 부족은 아무래도 핑계에 지나지 않을 수 있겠지만, 사실 과제가 처음 발표되었을 당시에는 과제 자체에 대한 이해가 부족했고 시간이 지나서 결과물을 만들고 제출해야 할 시기가 되었을 때는 여러 과목의 기말고사와 다른 기말 과제들이 즐비하고 있었기 때문에 상대적으로 투자해야 할 시간에 비해 투자 한 시간이 부족할 수밖에 없었다. 결국은 시간 관리를 못한 우리 탓임에는 다른 변명은 통하지 않을 것 같다.

그래도 그나마 발전적인 시간을 가져볼 수 있었던 것은, 처음 풀어보는 문제를 해결하기 위해 사용할 자료구조부터 고민해 볼 기회가 있었기 때문이다. Sudoku는 인터넷을 참고할 여러 소스 코드와 자료가 있었지만 상대적으로 크로스워드는 참조할 자료나 소스 코드가 거의 없었다. 전혀 없었다 해도 과언이 아니다. 그래서 밑바닥부터 우리가 고민해보아야 했고, 과제의 중점인 시간 단축에 있어 어떤 자료구조를 쓰느냐가 큰 영향을 미칠 것 같아서 정말 심사숙고 할 수 밖에 없었다.

처음으로 생각해 본 것은 Trie이다. 이 자료구조는 3-1학기 파일처리 시간에 배웠던 것으로 학기 말 진도 부분으로 인해 짧게 다루고 넘어갔던 내용이다. 그런데 빠르고 쉬운 검색을 위해 고민해보다가 이것을 구현해볼까 했으나, 크로스워드의 단어장에 aaa와 abc가 있으면 aa나 ab 등도 만들어 질 수 있는 등의 문제로 인해 포기하게 되었다.

그래서 다음으로 생각해 본 것은 역시 같은 시간에 배웠던 B-Tree이다. 삽입과 검색이 매우 빠른 편에 속하고 Trie 처음 의도하지 않은 데이터가 검색될 가능성도 없어 참 좋다고 생각이 들었으나 주어진 단어에 처음 스펠링이 안 주어졌을 때는 검색을 시도하기 어렵고 알고리즘 자체가 어려운 편이어서 그걸 구현하는 일은 결국 미뤄지게 되었다.

이렇게 자료구조를 고민하다 보니 우리가 배운 것으로는 크로스워드 시간 단축에 큰 영향을 미칠 자료구조가 없는 것 같았다. 시간이 조금 더 있었으면 오래 걸리더라도 크로스워드를 완벽하게 맞추는 프로그램을 만들 수 있지 않았을까 싶다. 무엇보다 혼자는 절대 힘들다. 팀원들의 머리를 빌려야 뭐라도 제대로 나오는 것 같아 다시 한번 팀플의 중요성을 실감할 수 있었다. 모두 수고는 많았으나 결과물의 사용 가능성은 솔직히 부족하지 않나 싶다.