자료구조 HW4

B711222 박조은
Hongik University
mrnglory@mail.hongik.ac.kr

October 25, 2019

I. List of Source Files

- hw4
  - maze.cpp
  - hw4.cpp
  - makefile
  - maze.in
  - maze.in2

II. hw4

i. maze.cpp

```
#include <iostream>
#include <stack>
using namespace std;

const int MAXSIZE = 100; // up to 100 by 100 maze allowed
bool maze[MAXSIZE + 2][MAXSIZE + 2];
bool mark[MAXSIZE + 1][MAXSIZE + 1] = {0};

enum directions {N, NE, E, SE, S, SW, W, NW};

struct offsets
{
        int a, b;
} move[8] = {-1,0, -1,1, 0,1, 1,1, 1,0, 1,-1, 0,-1, -1,-1};

struct Items
{
        Items(int xx = 0, int yy = 0, int dd = 0): x(xx), y(yy), dir(dd) {}
          int x, y, dir;
};

template <class T>
ostream& operator<< (ostream& os, stack<T>& s)
{
        // 스택의 내용을 역순으로 출력
        // 구현방법 = 내용을 하나씩 꺼내 다른 임시 스택에 넣어 저장한 후,
        // 최종적으로 그 임시 스택에서 하나씩 꺼내 출력하면 됨

        //os << "top = " << s.top() << endl;
        //for (int i = 0; i <= s.top(); i++)
        // os << i << ":" << s.stack[i] << endl;

        stack<T> s2;

        while (!s.empty())
        {
                s2.push(s.top());
                s.pop();
        }

        while (!s2.empty())
        {
                os << " -> " << s2.top();
                s2.pop();
        }

        return os;
}

ostream& operator<< (ostream& os, Items& item)
{
        // 5개의 Items가 출력 될 때마다 줄 바꾸기 위해

        static int count = 0;

        os << "(" << item.x << ", " << item.y << ")";
        count++;

        if ((count % 5) == 0)
                os << endl;

        return os;
}

void Path (const int m, const int p)
{
        // 구현은 책과 동일하다. 단, 최종적인 경로의 출력은 다음과 같이 한다.

        mark[1][1] = 1; // start at (1, 1)
        stack<Items> stack;
```

```
71          Items temp(1, 1, E);
72          stack.push(temp);
73
74          while (!stack.empty())
75          {
76                  temp = stack.top();
77                  stack.pop(); // unstack
78
79                  int i = temp.x;
80                  int j = temp.y;
81                  int d = temp.dir;
82
83                  while (d < 8) // move forward
84                  { // (i, j)에서 (g, h)로 이동한다고 하자.
85                          int g = i + move[d].a;
86                          int h = j + move[d].b;
87
88                          if ((g == m) && (h == p))
89                          {
90                                  int node = 0;
91
92                                  cout << stack;
93
94                                  temp.x = i;
95                                  temp.y = j;
96                                  cout << " -> " << temp;
97
98                                  temp.x = m;
99                                  temp.y = p;
100                                 cout << " -> " << temp
                                        ↪ << endl;
101
102                                 for (int i = 1; i < m + 1; i
                                        ↪ ++)
103                                         for (int j = 1; j <
                                                ↪ p + 1; j
                                                ↪ ++)
104                                                 if (mark[
                                                        ↪ i
                                                        ↪ ][
                                                        ↪ j
                                                        ↪ ]
                                                        ↪ )
105                                                         node
106                                 cout << "#nodes visited
                                        ↪ = " << node <<
                                        ↪ " out of " << m
                                        ↪ * p << endl;
107
108                                 return;
109                         }
110
111                         if ((!maze[g][h]) && (!mark[g][h])
                                ↪ ) // new position
112                         {
113                                 mark[g][h] = 1; //
                                        ↪ 방문한 적이
                                        ↪ 있다고 표시
114
115                                 temp.x = i;
116                                 temp.y = j;
117                                 temp.dir = d + 1; // 현
                                        ↪ 위치 및 실패 시
                                        ↪ 다음에 시도할
                                        ↪ 방향 저장
118
119                                 stack.push(temp); //
                                        ↪ stack it
120                                 i = g;
121                                 j = h;
122                                 d = N; // N방향부터 (
                                        ↪ 시계방향으로)
                                        ↪ 시도하자
123                         }
124                         else
125                                 d++; // try next
                                        ↪ direction
126                 } // end of while (d < 8)
127         } // end of while (!stack.empty())
128         cout << "No path in maze." << endl;
129 }
130
131 void getdata(istream& is, int& m, int& p)
132 { // 자료파일을 읽어들여 maze 에 저장한다.
133         is >> m >> p;
134
135         for (int i = 0; i < m + 2; i++)
136         { // 왼쪽 벽과 오른쪽 벽 작성
137                 maze[i][0] = 1;
138                 maze[i][p + 1] = 1;
139         }
140
141         for (int j = 1; j <= p; j++)
142         { // 윗쪽 벽과 아랫쪽 벽 작성
143                 maze[0][j] = 1;
144                 maze[m + 1][j] = 1;
145         }
146
147 ↪ ++;   for (int i = 1; i <= m; i++) // 자료 읽어들이기
148 ↪               for (int j = 1; j <= p; j++)
149                         is >> maze[i][j];
150 }
```

/*

* line 6: 경계선에 있는 경우, 즉 i == 1, i == m, j == 1, j == p 인 경우, 벽쪽으로는 진행하지 못하므로 가능한 방향은 8방향보다 작다. 따라서 경계조건을 매번 검사하지 않기 위해 미로의

주위를 1로 둘러싼다. 배열의 크기를 maze[m + 2][p + 2]로 선언한 이유가 이것이다.

* line 11 - 14: 미로 이동 시, 현재의 위치와 직전 이동 방향을 저장한 후, 한 방향을 선택한다. 북쪽부터 시작하여 시계방향으로 8가지의 방향을 배열 move에 정의해준다.
* line 50: Stack 과 Items 에 대해 연산자 다중화를 진행한다. friend 로 선언되어 Stack 의 전용 데이터 멤버 접근이 가능해진다.
* line 83: 갈 수 있는 모든 방향을 고려한다.
* line 102 - 105: 방문한 node 의 횟수를 카운팅하기 위해 중첩반복문으로 표현하였다. i 와 j 가 각각 1 부터 m 까지, 1 부터 p 까지만 반복되는 이유는 앞서 말했듯이 미로의 경계를 1로 둘러쌌기 때문에 해당 부분을 배제한 것이다.
* line 90: int 형 변수 node 를 0으로 초기화 해주었다.
* line 113: 이미 갔던 길을 다시 방문하지 않기 위함.
*

*/

ii. hw4.cpp

```
#include <iostream>
#include <fstream>
#include <cstdlib>
using namespace std;

void getdata(istream&, int&, int&);
void Path(int, int);

int main(int argc, char* argv[])
{
        int m, p; // m by p maze

        if (argc == 1)
                cerr << "Usage: " << argv[0] << "
                    ↪ maze_data_file\n" << endl;
        else
        {
                ifstream is(argv[1]);
                if (!is)
                {
                        cerr << argv[1] << " does not
                            ↪ exist\n";
                        exit(1);
                }

                cout << "For maze datafile (" << argv[1]
                    ↪ << ")\n";

                getdata(is, m, p);
                is.close();
                Path(m, p);
        }
}
```

```
/*
 * line 6: 미로의 정보를 읽어들인다. 해당 코드는 maze.cpp 의 line 131 - 150 에 구현되어있다.
 * line 7: 경로 설정을 위한 코드.

*/
```

iii. Results

iii.1 makefile

```
hw4: hw4.o maze.o
	g++ -o hw4 hw4.o maze.o
```

iii.2 maze.in

```
12 15
0 1 0 0 0 1 1 0 0 0 1 1 1 1 1
1 0 0 0 1 1 0 1 1 1 0 0 1 1 1
0 1 1 0 0 0 0 1 1 1 1 0 0 1 1
1 1 0 1 1 1 1 0 1 1 0 1 1 0 0
1 1 0 1 0 0 1 0 1 1 1 1 1 1 1
0 0 1 1 0 1 1 1 0 1 0 0 1 0 1
0 0 1 1 0 1 1 1 0 1 0 0 1 0 1
0 1 1 1 1 0 0 1 1 1 1 1 1 1 1
0 0 1 1 0 1 1 0 1 1 1 1 1 0 1
1 1 0 0 0 1 1 0 1 1 0 0 0 0 0
0 0 1 1 1 1 1 0 0 0 1 1 1 1 0
0 1 0 0 1 1 1 1 1 0 1 1 1 1 0
```

iii.3 maze.in2

```
9 9
0 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 0
1 0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 1
1 1 1 1 1 1 1 1 0
1 0 0 0 0 0 0 0 1
0 1 1 1 1 1 1 1 1
1 0 0 0 0 0 0 0 0
```

iii.4 compile

```
[B711222@localhost hw4d]$ hw4 maze.in
For maze datafile (maze.in)
  -> (1, 1) -> (2, 2) -> (1, 3) -> (1, 4) -> (1, 5)
  -> (2, 4) -> (3, 5) -> (3, 4) -> (4, 3) -> (5, 3)
  -> (6, 2) -> (7, 2) -> (8, 1) -> (9, 2) -> (10, 3)
  -> (10, 4) -> (9, 5) -> (8, 6) -> (8, 7) -> (9, 8)
  -> (10, 8) -> (11, 9) -> (11, 10) -> (10, 11) -> (10, 12)
  -> (10, 13) -> (9, 14) -> (10, 15) -> (11, 15) -> (12,
      ↪ 15)

#nodes visited = 48 out of 180
```

```
[B711222@localhost hw4d]$ hw4 maze.in2
For maze datafile (maze.in2)
  -> (1, 1) -> (1, 2) -> (1, 3) -> (1, 4) -> (1, 5)
  -> (1, 6) -> (1, 7) -> (1, 8) -> (2, 9) -> (3, 8)
  -> (3, 7) -> (3, 6) -> (3, 5) -> (3, 4) -> (3, 3)
  -> (3, 2) -> (4, 1) -> (5, 2) -> (5, 3) -> (5, 4)
  -> (5, 5) -> (5, 6) -> (5, 7) -> (5, 8) -> (6, 9)
  -> (7, 8) -> (7, 7) -> (7, 6) -> (7, 5) -> (7, 4)
  -> (7, 3) -> (7, 2) -> (8, 1) -> (9, 2) -> (9, 3)
  -> (9, 4) -> (9, 5) -> (9, 6) -> (9, 7) -> (9, 8)
  -> (9, 9)
#nodes visited = 40 out of 81
```

/* 어려웠던 점
 * 처음에는 미로의 입구와 출구는 잘 출력이 되는 반면에 이동 경로가 조금씩 다르게 나왔었다.
 * 아무리 생각해도 모르겠어서 그냥 포기하려고 했는데, maze.cpp 의 move[8] 배열에 진행방향 저장을 할 때, enum directions 에 저장되어있는 방향들과 맞춰서 동일하게 작성을 해야했는데, 원소들을 그냥 아무렇게나 적었기 때문에 이동경로가 조금씩 다르게 나왔었다는 걸 깨달았다.

*/