**2022 Spring OOP Assignment Report**

과제 번호 : ASSN1\_prob3

학번 : 20210716

이름 : 최대현

Povis ID : daehyeonchoi

**명예서약 (Honor Code)**

나는 이 프로그래밍 과제를 다른 사람의 부적절한 도움 없이 완수하였습니다.

I completed this programming task without the improper help of others.

프로그램을 하다 보면 결정해야 할 세부 사항이 많은데, 이러한 세부 사항을 처리한 방법과 이유를 보고서에 쓰십시오.

독창적인 아이디어와 추가 기능은 보너스 점수를 받을 수 있으므로, 보고서에 명확히 기재하십시오.

문제가 여러 개인 경우, 각 문제별로 정리해서 작성합니다.

1. **프로그램 개요**

개발환경: Visual Studio 2019

Visual studio에서 프로젝트를 실행한 후 리소스 파일에 prob3\_20210716.cpp 파일을, 소스 파일에 input.txt를 드래그한 뒤 실행한다.

Input.txt: 조교님께서 제공해주시거나 스스로 만든 test case 파일이며, 행 수에는 제한이 없지만 (op, num1, num2 개행) 형태의 파일이어야 원활한 구동이 가능하다.

Output.txt: input.txt의 정보를 바탕으로 (key까지의 최단거리+ exit까지의 최단거리)를 계산하여 출력하는 대상 파일

1. **프로그램의 구조 및 알고리즘**

별도의 클래스는 구현하지 않았다.

**알고리즘 설명**

과제를 수행하기 위한 사전 조사를 하던 중 BFS라는 알고리즘에 대해 탐구했다. 이 문제를 푸는 데 있어 BFS가 최적의 방법이라고 생각해서 구현했다.

**BFS**

BFS는 넓이 우선 탐색의 약자이며, 시작 정점을 방문한 후 시작 정점에 인접한 모든 정점들을 우선 방문하는 방법이다. 더 이상 방문하지 않은 정점이 없을 때까지 방문하지 않은 모든 정점들에 대해서도 너비 우선 탐색을 적용한다. BFS는 출발노드에서부터 목표노드까지의 최단거리를 보장한다. BFS는 큐를 이용해서 구현할 수 있다. STL을 사용할 수 없으므로 queue와 관련 함수를 직접 구현하여 사용했다.

텍스트이(가) 표시된 사진

자동 생성된 설명

**사용자 정의 자료형 설명**

연결 리스트 중 하나인 큐를 직접 구현하고, 관련 기능(enqueue, dequeue, empty) 을 함수로구현했다. 단위 노드의 data는 int형으로 만들었다. (현재 위치의 행과 열 정보를 각각 넣어야 하기 때문)

Row와 col에 해당하는 큐 qrow와 qcol을 따로 정의하였고, 해당 좌표로부터 시작점까지의 거리값도 qcnt라는 큐를 만들어 갱신하였다.

**함수 설명**

텍스트이(가) 표시된 사진

자동 생성된 설명

1. **int** search(**int** des\_row, **int** des\_col, **int** map[][11], **int** visit[][11], **QUEUE** qrow, **QUEUE** qcol, **QUEUE** qcnt)

이번 과제에서 가장 비중이 큰 함수로, 목적지의 row와 col 값을 매개변수로 가져가고 BFS 알고리즘을 통해 출발지와 목적지까지의 최단거리를 return하는 함수이다.

이 함수를 구현한 뒤 2번의 반복실행에 대해 모두 이 함수를 사용했다. 첫 번째 시행은 출발지가 (0,0)이고 목적지가 (key\_row, key\_col)인 케이스, 두 번째 시행은 출발지가 (key\_row, key\_col)이고 목적지가 (exit\_row, exit\_col)인 케이스로 해석해서 함수 하나를 만든 뒤 간단히 해결할 수 있었다.

텍스트, 화면, 스크린샷, 은색이(가) 표시된 사진

자동 생성된 설명

텍스트이(가) 표시된 사진

자동 생성된 설명

while문을 사용하였고, 기본적으로 무한루프지만 while문 내에 중지조건을 달았다. 중지조건은 현재 위치가 목적지 위치와 같을 때 이다. While문을 1번 돌 때마다 상하좌우 인접한 1칸에 갈 수 있는지를 체크한다. 갈 수 있으면 그 위치의 행과 열을 각각 qrow와 qcol에 enqueue해준다. 그리고 qcnt에 1 증가한 값을 동시에 넣어준다. 그 뒤 while문을 돌 때마다 front의 값을 현재 row와 col로 갖고, dequeue해주는 과정을 반복한다. 마지막으로 함수를 리턴할 때 리턴값은 목적지까지의 거리이다. 이 과정을 key\_row, key\_col에 대해 1회, exit\_row, exit\_col에 대해 1회 시행해주면 원하는 거리 정보를 얻을 수 있다.

1. **int** enqueue(**QUEUE** \*q, **int** data)

FIFO 구조인 queue의 rear에 node를 삽입해주는 함수이다.

1. **int** dequeue(**QUEUE** \*q)

queue의 front에 위치한 node를 떼주는 함수이다.

1. **int** isEmpty(**QUEUE** \*q)

queue가 비어 있는지 확인하는 함수이다.

1. **int** check(**int** row, **int** col, **int** map[][11], **int** visit[][11])

현재 위치에서 상하좌우 1칸 이동한 위치에 대해 이동 가능한지를 판단하는 함수이다. 이 때 이동 가능한 조건은 index가 map의 범위 내여야 하고, 그 index의 data가 0이어야 하고, visit[row][col] 값이 0(이전에 방문한 적이 없어야 함)이어야 한다.

1. **void** clean(int **arr**[][11])

출발점 – key에 대해 1번, key – exit에 대해 1번, 총 2번의 반복 실행을 거치기 때문에 2번째 실행 직전에 visit array를 비워주기 위해 만든 함수이다.

1. **void** cleanq(**QUEUE** \*q)

isEmpty 함수 값이 0인 동안, 즉 큐가 비어있지 않은 동안 dequeue를 반복 실행해서 큐를 비워주는 함수이다.

**변수 설명**

**Main 함수에서 중요한 역할을 하는 변수들**

key\_row, key\_col: key가 위치한 행과 열의 index값

exit\_row, exit\_col: exit이 위치한 행과 열의 index값

dis\_key: 시작점부터 key까지의 최단 거리를 저장하는 변수

dis\_exit: key부터 exit까지의 최단 거리를 저장하는 변수

dis\_tot: 마지막으로 output.txt에 출력할 값을 저장하는 변수

1. **토론 및 개선**

텍스트이(가) 표시된 사진

자동 생성된 설명

상하좌우의 경우에 대해 반복적으로 check를 해 주기 때문에 이 부분을 더 간소화할 수 있을 것 같은데, 그러지 못해서 아쉽다. 관련한 아이디어에 대해 좀 더 고민해보고 싶다.

개인적으로 search 함수를 따로 만들지 않고 1개를 만들어 2번의 시행에 활용한 것이 효율적인 판단이었다고 생각한다.

BFS 알고리즘을 처음 접했을 때는 이해하기도 어려웠지만, 직접 코딩을 해보며 BFS를 이해하고 나아가 DFS 방식까지 이해해볼 수 있었다.

1. **참고 문헌**

BFS – Wikipedia (알고리즘을 참고하였음)

<https://ko.wikipedia.org/wiki/%EB%84%88%EB%B9%84_%EC%9A%B0%EC%84%A0_%ED%83%90%EC%83%89>