자료구조 Assignment #5 보고서

김재학

(소프트웨어학과, 202220757)

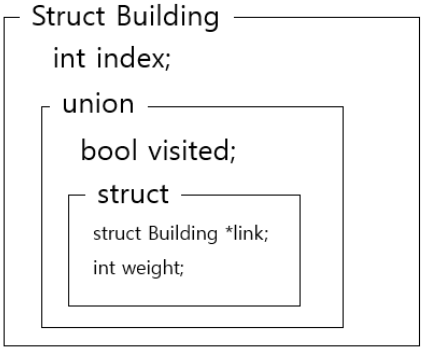
목차

1. 개발환경

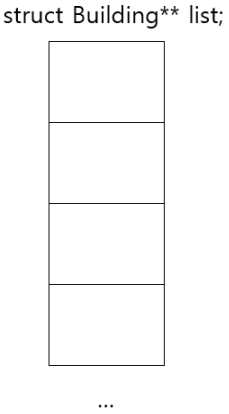
비주얼 스튜디오 2022, SDL체크 사용 안함

2. 프로그램 요약 및 실행 결과

이 프로그램의 작동 원리를 간략히 설명하자면, 먼저 건물의 수를 입력 받고, 건물을 그래프로 나타내기 위한 adjacency list를 입력 받은 수만큼 생성한다. 건물을 나타내는 노드(struct Building)의 구조는 [그림 1a]와 같고, adjacency list는 [그림 1b]처럼 동적 어레이로 생성된다.



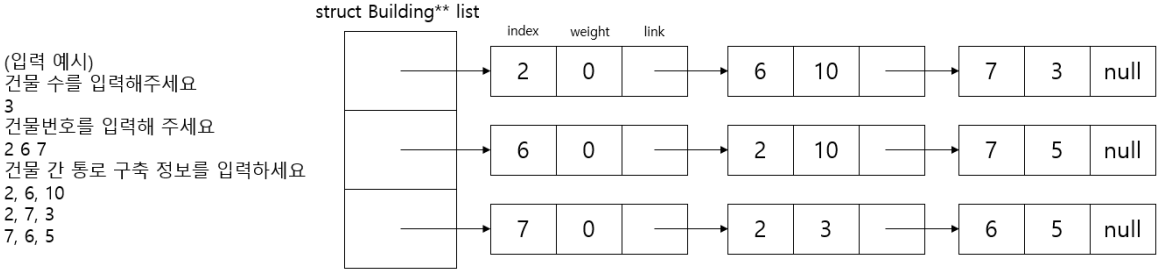
[그림 1a]



[그림 1b]

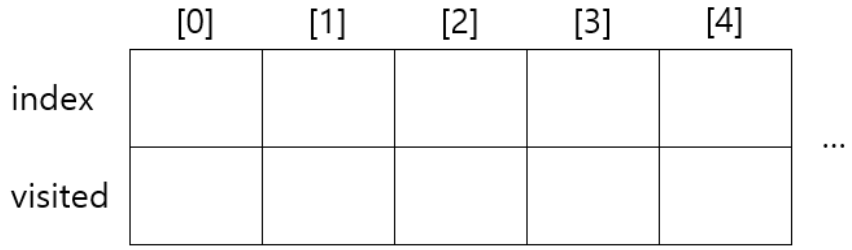
건물노드는 건물 번호를 나타내는 int index, 다음 노드를 가리키는 노드 포인터 link, 가중치를나타내는 int weight로 구성되어 있으며, 이 노드는 BFS와 DFS에서 사용되는 visited array에서도 사용되므로 bool visited를 포함하고 있다. *link와 weight를 포함하는 구조체와 visited는 동시에 사용될 일이 없기 때문에 메모리를 공유하도록 구성하였다. Adjacency list는 각 요소가 struct Building을 가리키는 포인터 어레이이다. 동적할당으로 생성되며, 각 요소가 가리키는 노드가 입력받은 건

물이다. 예를들어 list[0]->index는 첫번째로 입력받은 건물의 번호가 된다. 건물 간 통로 구축 정보를 입력 받게 되면 adjacency list의 방법으로 그래프를 표현하게 된다. 예시는 [그림 2]와 같다.



[그림 2]

건물 수를 입력 받을 때 DFS와 BFS에서 사용될 visited array 또한 함께 생성되는데 구조체의 크기로 건물 수만큼 동적배열을 생성하기 때문에 [그림 3]과 같은 구조를 가지고 있다.



[그림 3]

만들어진 건물 그래프를 통해서 DFS를 실행하였을 때, 1번의 실행에 모든 건물을 방문한다면, "성공적인 계획입니다" 를 출력하고 아니라면 "전체 건물들을 연결할 수 없습니다" 를 출력한다. 그 후 DFS, BFS, Kruskal's Algorithm의 실행 결과를 출력한다. 이 프로그램은 건물 수를 0으로 입력 받을 때까지 무한히 실행된다.

예제 입력을 통한 프로그램의 실행결과는 다음과 같다.

```
아주대학교 지하 통로 구축 서비스
건물 수를 입력해 주세요
10
건물 번호를 입력해 주세요
14 25 5 9 10 31 40 29 33 30 27 16
제시된 도시 수보다 도시 이름의 개수가 더 많습니다.
건물 수를 입력해 주세요
12
건물 번호를 입력해 주세요
14 25 5 9 10 31 40 29 33 30 27 16
건물 간 통로 구축 정보를 입력하세요
14, 25, 70
14, 16, 50
14,10, 19
10, 9, 21
5, 27, 31
9, 5, 29
14, 27, 80
27, 14, 70
중복된 정보입니다
14, 1, 200
건물이 잘못 입력되었습니다
30, 29, 40
29, 40, 11
40, 33, 23
30, 31, 35
33, 31, 27

전체 건물들을 연결할 수 없습니다
dfs : ( 14 25 16 10 9 5 27 ), ( 31 30 29 40 33 )
bfs : ( 14 25 16 10 27 9 5 ), ( 31 30 33 29 40 )
Kruskal : ((29, 40), (10, 14), (9, 10), (33, 40), (31, 33), (5, 9), (5, 27), (30, 31), (14, 16), (14, 25)), 316
----------------------------------------
```

```
아주대학교 지하 통로 구축 서비스
건물 수를 입력해 주세요
10
건물 번호를 입력해 주세요
14 25 5 10 40 29 33 30 27 16
건물 간 통로 구축 정보를 입력하세요
14, 16, 51
10, 14, 21
10, 5, 13
5, 27, 18
27, 40, 30
40, 33, 19
33, 30, 8
30, 16, 40
16, 25, 60
10, 25, 33
27, 25, 27
27, 29, 45
25, 29, 10
29, 40, 7
29, 30, 15
25, 30, 25

성공적인 계획입니다
dfs : ( 14 16 30 33 40 27 5 10 25 29 )
bfs : ( 14 16 10 30 25 5 33 29 27 40 )
Kruskal : ((29, 40), (30, 33), (25, 29), (5, 10), (29, 30), (5, 27), (10, 14), (25, 27), (16, 30)), 159
----------------------------------------
아주대학교 지하 통로 구축 서비스
건물 수를 입력해 주세요
0
        서비스를 종료합니다.
```

예제의 입력을 똑같이 입력했을 경우 예제와 같은 결과값이 출력되는 것을 볼 수 있다.

3. 중요 함수 및 기능설명

그래프 관련 함수:

`int inputBuildingCount()`	건물 수를 입력 받아 반환한다.
`Building** makeList(int num)`	adjacency list를 num만큼 동적할당하여 주소를 반환한다.
`Building* getBuilding(int index, int weight)`	건물 노드를 생성한다. 이때 getNode()함수를 이용해 가용공간에서 노드를 가져온다.
`int inputBuildingNum(int num, Building** list, Building* visited)`	건물 번호를 입력받아서 getBuilding()함수와 inputLA()함수를 호출하여 건물 정보를 adjacency list에 입력한다.
`void clearBuilding(Building** list, int num, Building* visited)`	모든 건물 노드를 retNode()함수를 이용해 반환하고, visited array와 adjacency list를 해제한다.
`void inputEdges(Building** list, int num)`	건물의 연결정보를 입력받아 그래프로 저장한다.
`int findBuilding(Building** list, int data, int num)`	전달받은 건물의 번호가 배열의 어떤 index에 저장되어있는지 반환한다.
`int printDFS(Building** list, Building* visited, int num)`	그래프의 DFS결과를 출력한다. 반환값은 connected component의 개수이다.
`int printBFS(Building** list, Building* visited, int num)`	그래프의 BFS결과를 출력한다. 반환값은 connected component의 개수이다.

`int noPrintDFS(Building** list, Building* visited, int num)`	출력없이 DFS를 실행하여 전체건물을 연결하는 경로가 있는지 확인하는 목적으로 사용된다. 반환값은 connected component의 개수이다.

함수 내에서 호출되어 사용되는 세부 기능을 가진 함수들은 표에서 생략하였다.

Queue 구현:

`#define Q_SIZE 100`
`int queue[Q_SIZE];`
`int front, back;`
`void push(int data);`
`int pop();`
	크기 100만큼의 circular queue를 구현하였다. 이 queue는 BFS함수를 실행할 때 사용된다.

Min Heap 구현:

`typedef struct Edge {`
`  int v1;`
`  int v2;`
`  int weight;`
`}Edge;`
	heap의 노드를 구성할 구조체 선언

`int edges;`	위에서 입력 받은 건물 연결정보의 개수
`int nh;`	heap에 들어있는 노드의 수
`Edge* makeHeap(int edges)`	edges개의 Edge array 동적할당
`void pushHeap(Edge* h, int v1, int v2, int weight);`	heap에 데이터 추가
`void popHeap(Edge* h, int* v1, int* v2, int* weight)`	heap에서 데이터 반환, 노드의 각 데이터를 인자로 전달받은 주소로 전달함
`void delHeap(Edge* h)`	heap 삭제 및 초기화
`int searchHeap(Edge* h, int v1, int v2, int weight)`	인자로 전달받은 데이터를 가지는 노드가 힙에 존재하는지 검색. 중복을 체크하기 위해 사용됨.

Kruskal's algorithm 구현:

`void btoh(Building** list, int num, Edge* heap);`	그래프의 edge정보를 heap에 저장
`void kruskal(Building** list, int num, Edge* heap);`	heap을 이용해 kruskal's algorithm 출력

4. 가산점 요소 구현

available space list 구현:

`Building* avail;`	가용공간을 나타내는 포인터 선언
`Building* getNode()`	가용공간에서 노드를 1개 반환. 가용공간에 노드가 없

	다면 새로운 노드를 할당하여 반환
`void retNode(Building* node)`	사용한 노드를 추후 재사용하기 위해 가용공간으로 반환

이 가용공간은 Building 노드를 사용할 때 사용되었다.