2020년도 2학기 컴퓨터공학설계및실험I

13주차 예비 보고서

20170175 김태안

1. 실습 목적

본 실험은 1주차 미로 만들기 프로젝트의 결과물인 ~.maz 파일을 읽어서 이를 윈도우 화면상에 표시함을 목적으로 한다. 실험목적인 GUI(Graphical User Interface) 구축과 동시에 3주차에서 행할 프로젝트를 위하여 학생들은 창의적이고 효율적인 자료구조를 설계하여야 한다. GUI를 용이하게 설계하기 위하여 OpenFrameWorks를 사용하여 구현하도록 한다.

1. 관련 이론

**DFS 알고리즘**

DFS(Depth First Search)와 BFS(Breadth First Search)는 그래프 자료구조를 탐색하는 알고리즘으로 구성하는 모든 노드를 방문하며 탐색한다. DFS는 출발 정점 v를 방문하며 시작한다. 다음으로 인접한 정점을 인접 리스트에 저장한 뒤, v에 인접하면서 아직 방문하지 않은 정점 w를 선택해 w에 인접하면서 아직 방문하지 않은 새 정점을 선택하며 DFS를 다시 시작한다. 탐색을 반복하면 결국 모든 정점을 탐색하게 된다. 이를 C 코드로 표현하면 다음과 같다.[1]

|  |
| --- |
| void dfs (int v) {  nodePointer w;  visited[v] = TRUE;  printf(“%d”, v);  for (w = graph[v]; w; w = w->link)  if (! visited[w->vertex])  dfs(w->vertex);  } |

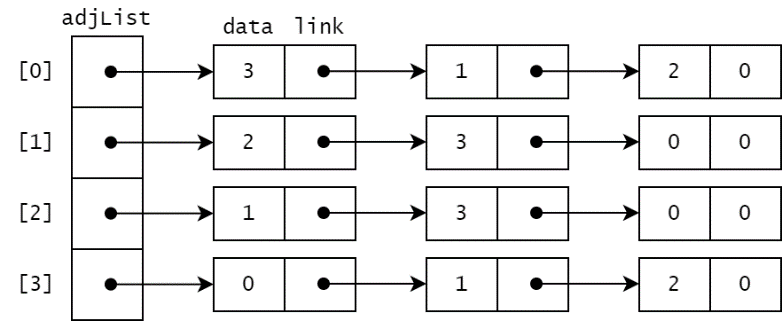
**BFS 알고리즘**

BFS는 DFS와 다르게 정점 v에서 시작하여 방문 표시를 한 뒤, v의 인접 정점을 모두 방문한다. 그 다음, v의 첫 인접 정점에 인접한 정점부터 모든 인접 정점의 인접 정점을 방문한다. 이를 반복해 모든 정점을 탐색한다. 이를 C 코드를 나타내면 다음과 같다.[1]

|  |
| --- |
| void bfs (int v) {  nodePointer w;  front = rear = NULL;  printf(“%d”, v);  visited[v] = TRUE;  addq(v);  while (front) {  v = deleteq();  for (w = graph[v]; w; w = w->link)  if (!visited[w->vertex]) {  printf(“%d”, v);  addq(w->vertex);  visited[w->vertex] = TRUE;  }  }  } |

1. 실습 방법

.maz 파일을 읽어 DFS나 BFS를 통해 미로의 해답을 구하기 위해서는 .maz의 미로 데이터를 저장해야 한다. 미로의 데이터는 완전 미로는 트리, 불완전 미로는 그래프의 모양으로 저장할 수 있다. 그 중, 많이 사용되는 그래프 표현법 중 하나인 인접 리스트(Adjacency List)를 사용한다. 인접 리스트는 각 정점과 연결된 정점을 원소로 하는 연결 리스트의 집합 배열 구조로 이루어져 있다.[1]



.maz 파일을 adjList로 저장하기 위해선 WIDTH\*HEIGHT의 크기를 갖는 배열이 필요하다. 각 칸에 0부터 WIDTH\*HEIGHT까지의 번호를 붙인 뒤, .maz 파일을 한 줄씩 읽으며 오른쪽 경계선이나 아래쪽 경계선이 없다면 인접한 두 칸의 정보를 각 칸에 삽입해 (예를 들어, 0과 1사이에 경계선이 없다면 0에 1을, 1에 0을 삽입한다.) 각 칸에서 이동할 수 있는 칸을 저장한다.

인접 리스트를 통해 DFS나 BFS를 구현하기 위해서는 방문한 노드를 확인해야한다. 이를 위해 방문한 노드는 visited를 true로 저장해 다시 방문하지 않도록 한다. 인접 리스트의 각 노드는 다음과 같이 구현한다.

|  |
| --- |
| typedef struct node {  int cellNum;  bool visited;  int parent;  node \*link;  }; |

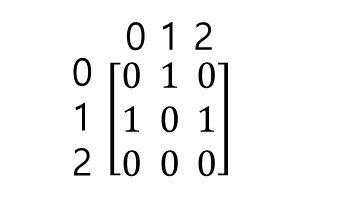
위 자료구조를 사용하면 DFS나 BFS를 쉽게 구현할 수 있다. DFS의 경우, adjList[0]부터 시작하여 인접한 정점에 대해 DFS를 재귀적으로 수행한다. 이때, 현재 위치를 스택에 삽입해 저장한다. WIDTH\*HEIGHT 칸을 만나면 DFS를 종료하고 스택을 반환한다. BFS의 경우, adjList[0]부터 시작하여 인접한 정점을 큐에 저장한다. 그리고, 큐에 저장된 노드의 인접 정점을 다시 저장한다. 이를 반복해 WIDTH\*HEIGHT칸을 찾는다. 그 뒤, parent를 통해 연결된 노드를 Backtracing 하여 경로를 찾는다.

두 탐색 방법 모두 미로의 경로를 찾을 수 있으나 DFS가 미로 찾기에 더 적합한 방법이라고 할 수 있다. 미로의 경우, 시작점과 도착점은 멀리 떨어져 있다. 따라서 BFS를 사용한 탐색을 한다면 항상 거의 모든 노드를 탐색해야 한다. 그러나, DFS의 경우 미로의 모양에 따라 필요한 노드만을 방문할 수도 있고, 모든 노드를 탐색할 수도 있다. 즉, 평균적으로 결리는 시간을 계산했을 때, DFS가 더 적은 탐색을 하게 된다.

1. 기타

**인접 행렬 (Adjacency Matrix)**

그래프를 표현하는 또 다른 방법 중 하나는 행렬을 사용하는 것이다. 행렬을 사용해 두 정점이 연결되어 있다면 1을, 아니라면 0으로 표시하여 그래프를 표현한다. 이때, 그래프를 표현하는 대에 필요한 공간은 (WIDTH\*HEIGHT)2이다. 이는 (WIDTH\*HEIGHT+Edges)인 인접 리스트보다 더 많은 공간이 필요하며, 탐색에서 더 큰 시간 복잡도를 보인다.



1. 참고 문헌

[1] Ellis Horowitz et al., “6. Graph.” *Fundamentals of Data Structures in C*, University Press, 2008.