# 11주차 결과보고서

전공: 신문방송학과 학년: 3학년 학번: 20191150 이름: 전현길

**1. 실험시간에 작성한 프로그램의 알고리즘과 자료구조를 요약하여 기술하시오. 완성한 알고리즘의 시간 및 공간 복잡도를 보이고 실험 전에 생각한 방법과 어떻게 다른지 아울러 기술하시오.**

실험 전에 생각한 방법 그대로 kruskal 알고리즘을 통해 완전 미로를 생성하는 코드를 작성했다. 시간, 공간 복잡도 역시 실습 전에 고려했던 O(E · log E), O(E)/O(WIDTH^2) 수준에서 변화가 없었다. 시간 복잡도가 최대인 작업은 무작위 가중치를 갖는 간선을 정렬하는 작업이며, 공간 복잡도가 최대인 변수는 2차원 char 배열 maze, 또는 1차원 간선 벡터 edge이다.

아래는 변수, 함수를 정리한 내용이다.

| 변수명 | 설명 |
|---|---|
| random_device rd<br>mt19937 gen(rd())<br>uniform_int_distribution<int> dis(1, 1000) | 간선별로 무작위 가중치를 부여하기 위한 변수 |
| int* parent | union-find를 수행하기 위해 각 정점의 부모 노드를 나타내는 **1차원 동적 할당 int 배열** |
| char** maze | 미로를 표현하기 위한 **2차원 동적 할당 char 배열** |
| vector<Edge> edge | 간선을 저장하는 **1차원 Edge 벡터,** Edge 자료형은 멤버 변수 start, end, weight에 두 정점과 가중치를 저장한다. |
| int WIDTH, HEIGHT, VERTEX | 미로의 가로세로 길이, 정점 개수를 저장하는 변수 |

| 함수명 | 설명 |
|---|---|
| int find(int x) | x번째 노드의 부모 노드를 반환한다. 두 노드가 동일한 집합에 속하는지 확인하기 위해 사용된다. |
| void unite(int u, int v) | parent 배열에서 부모 노드의 번호가 더 작은 쪽으로 노드를 병합한다. 두 노드 집합을 합치는 데 사용된다. |

```cpp
// 1 ~ 1000 범위의 난수 생성
random_device rd;
mt19937 gen(rd());
uniform_int_distribution<int> dis(1, 1000);
vector<Edge> edge; // 간선을 저장할 1차원 벡터

cout << "width: "; cin >> WIDTH;
cout << "height: "; cin >> HEIGHT;
VERTEX = WIDTH * HEIGHT;
parent = new int[VERTEX + 1];

// 미로 동적 할당
maze = new char *[2 * HEIGHT + 1];
for (int i = 0; i < 2 * HEIGHT + 1; i++) {
  maze[i] = new char[2 * WIDTH + 1];
}

// 미로의 모든 벽을 간선으로 추가(r: row, c: col)
for (int r = 1; r <= HEIGHT; r++) {
  for (int c = 1; c <= WIDTH; c++) {
    int u = (r - 1) * WIDTH + c; // 현재 노드

    // 마지막 열/행이 아닐 경우 오른쪽/아래쪽 노드와 연결
    if (c < WIDTH) edge.push_back(Edge(u, u + 1, dis(gen)));
    if (r < HEIGHT) edge.push_back(Edge(u, u + WIDTH, dis(gen)));
  }
}

// 각 vector의 부모 노드를 나타내는 parent 배열
for (int i = 1; i <= VERTEX; i++) {
  parent[i] = i;
}

sort(edge.begin(), edge.end());

for (int i = 0; i <= 2 * HEIGHT; i++) {
  for (int j = 0; j <= 2 * WIDTH; j++) {
    if (i % 2 == 0 && j % 2 == 0)
      maze[i][j] = '+';
    else if (i % 2 == 0) maze[i][j] = '-';
    else if (j % 2 == 0) maze[i][j] = '|';
    else
      maze[i][j] = ' ';
  }
}

for (auto e : edge) {
  int u = e.start, v = e.end;
  if (find(u) != find(v)) {
    unite(u, v);
    int x = (u - 1) / WIDTH + (v - 1) / WIDTH + 1;
    int y = (u - 1) % WIDTH + (v - 1) % WIDTH + 1;
    maze[x][y] = ' ';
  }
}

for (int i = 0; i <= 2 * HEIGHT; i++) {
  for (int j = 0; j <= 2 * WIDTH; j++) {
    cout << maze[i][j];
  }
  cout << '\n';
}

delete[] parent;
for (int i = 0; i < 2 * HEIGHT + 1; i++) {
  delete[] maze[i];
}
delete[] maze;
```

프로그램의 작동은 그리 복잡한 부분이 없고, 다음처럼 요약할 수 있다.

### 1. 입력 및 초기화

a) 미로의 가로, 세로 길이를 입력하고, 난수 생성 변수를 초기화하며 parent 배열, maze 2차원 배열에 메모리를 동적 할당한다.

b) parent 배열의 각 노드의 부모 노드를 자기 자신으로 초기화하고, 미로의 크기에 맞추어 간선을 edge 1차원 벡터에 추가한다. 이 때 **간선의 가중치를 무작위로 부여**한다.

c) 간선을 **가중치를 기준으로 정렬한다.** 가중치를 무작위로 부여하지 않고 정렬만 할 경우, 프로그램을 실행할 때마다 같은 입력값에 대해 항상 같은 미로를 출력하게 된다. 정렬을 하지 않을 경우, 매우 단조롭고 예측 가능한 미로를 생성하게 된다.

d) 미로의 기본 형태를 그린다.

### 2. kruskal 알고리즘에 따라 최소 신장 트리 생성

a) 무작위로 정렬된 간선에 대해 두 꼭짓점이 같은 집합에 속하는지 확인한다. find() 함수를 통해 확인할 수 있다.

b) 같은 집합에 속하면 사이클이 생성되는 간선이므로 버리고, 그렇지 않다면 union()을 통해 같은 집합에 합친다.

c) 간선의 두 정점이 같은 집합으로 합쳐졌다면, 두 정점 간의 벽을 부순다.

### 3. 출력 및 동적 할당 해제

a) 미로를 출력한다.

b) parent, maze 배열의 메모리를 해제한다.