11주차 결과보고서

전공: 컴퓨터공학과 학년: 2학년 학번: 20231632 이름: Jumagul Alua

- 알고리즘 설명:

미로를 탐색하는 프로그램에서 무작위로 벽을 제거하면서 미로를 생성하는 유니언-파인드(Union-Find) 알고리을 활요했다. 이 알고리즘은 Kruskal's Minimum Spanning Tree 알고리즘과 유사하다.

```c
typedef struct {
    int parent; //부모 노드
    int rank;//트리의 높이
} Set;

//유이언-파인드 초가화 함수
void initSet(Set *set, int size) {
    for (int i = 0; i < size; i++) {
        set[i].parent = i;//각 요소가 자기의 부모
        set[i].rank = 0;//initial rank is 0
    }
}
```

각 셀의 부모와 랭크를 저장하는 유니언-파인드 자료구조인 Set과 모든 셀을 자신을 부모로 가지는 독립된 집합으로 initSet 함수에서 초기화한다.

```c
//수평 벽을 그리는 함수
void drawHori(FILE *fp, int width, const int *walls) {
    for (int i = 0; i < width; i++) {
        fprintf(fp, "+");//a junction
        if (walls[i] == WALL) {
            fprintf(fp, "-");//wall
        } else {
            fprintf(fp, " ");//space
        }
    }
    fprintf(fp, "+\n");//ending junction
}
//수직 벽을 그리는 함수
void drawVert(FILE *fp, int width, const int *walls) {
    fprintf(fp, "|");//left wall
    for (int i = 0; i < width - 1; i++) {
        fprintf(fp, " ");//cell
        if (walls[i] == WALL) {
            fprintf(fp, "|");//wall
        } else {
            fprintf(fp, " ");//space
        }
    }
    fprintf(fp, " |\n");//right wall
}
```

수평 및 수직 벽을 모두 포함하는 리스트를 생성한다.

```c
//벽 리스트를 섞는 함수
void shuffle(Wall *array, int n) {
    for (int i = n - 1; i > 0; i--) {
        int j = rand() % (i + 1);//무작위로 인덱스 생성
        Wall temp = array[i];//두 벽을 교환
        array[i] = array[j];
        array[j] = temp;
    }
}
```

랜덤하게 벽 리스트를 랜덤하게 부여한다.

```c
//유니언-파인드의 find 함수
int find(Set *set, int x) {
    if (set[x].parent != x) {
        set[x].parent = find(set, set[x].parent);//path compression
    }
    return set[x].parent;
}
////유니언-파인드의  함수
void uSets(Set *set, int x, int y) {
    int rootX = find(set, x);
    int rootY = find(set, y);

    if (rootX == rootY) {
        return;
    }
    if (set[rootX].rank < set[rootY].rank) {
        set[rootX].parent = rootY;//rootY가 rootX의 부모가 되게끔
    } else if (set[rootX].rank > set[rootY].rank) {
        set[rootY].parent = rootX;//rootX가 rootY의 부모가 되게끔
    } else {
        set[rootY].parent = rootX;//if ranks are the same, make rootx the parent and increase its rank
        set[rootX].rank++;
    }
}
```

find 함수는 셀의 루트를 찾고 uSets 함수는 두 집합을 합친다. 그리고 두 셀이 다른 집합에 속하면 벽을 제거하여 미로를 생성한다.

```
// 첫 번째 줄 그리기
for (int i = 0; i < width; i++) {
    fprintf(fp, "+-");
}
fprintf(fp, "+\n");

// 나머지 미로 그리기
for (int y = 0; y < height - 1; y++) {
    drawVert(fp, width, &vWalls[y * (width - 1)]);//수직 벽
    drawHori(fp, width, &hWalls[y * width]);//수평 벽
}

// 마지막 줄 인접한 셀의 모든 벽 제거
for (int x = 0; x < width - 1; x++) {
    int cell1 = (height - 1) * width + x;
    int cell2 = (height - 1) * width + x + 1;
    if (find(sets, cell1) != find(sets, cell2)) {
        vWalls[(height - 1) * (width - 1) + x] = SPACE;//수직 벽 제거
        uSets(sets, cell1, cell2);//두 샐 합병
    }
}

drawVert(fp, width, &vWalls[(height - 2) * (width - 1)]);//마직막 수직 벽 그리기`

// 마지막 줄 그리기
for (int i = 0; i < width; i++) {
    fprintf(fp, "+-");
}
fprintf(fp, "+\n");
```

최종적으로 미로를 파일로 출력한다.

- 예상했던 방법과 실제 구현의 차이 예상 방법:

처음에는 Eller's 알고리즘을 사용할 계획이었다. 이 알고리즘은 각 행을 독립적으로 처리하면서 집합을 관리하고 임의로 벽을 제거하여 연결한다. 실제 구현할때 eller's algorithm 대신 유니언-파인드와 크루스칼 알고리즘을 사용하였다. 이는 전체 미로를 한꺼번에 처리하며, 무작위 벽을 제거하면서 모든 셀이 하나의 집합으로 합쳐질 때까지 반복한다. 차이점에 대해 말하자면, 예상 방법에서는 행별로 집합을 관리하지만, 실제 구현에서는 전체 미로를 한꺼번에 처리한다. 그리고 eller's 알고리즘에서는 임의의 벽을 선택하는 방식이지만, 실제 구현은 벽 리스트를 무작위로 셔플하여 순차적으로 처리한다. 이로 인해 실제 구현은 더 간단하고 일관되며, 모든 벽을 한꺼번에 처리할 수 있다는 장점이 있다. 반면, 예상 방법은 행별로 처리하여 좀 더 세밀한 제어가 가능하지만 구현이 복잡할 수 있다.