알고리즘 실습(8주차)

201000287 일어일문학과 유다훈

1. 과제 설명 및 해결 방법

1. Minimum Spanning Tree 구현

① Kuruskal's Algorithm

- 입력 받은 Adjacent matrix를 이용하여 MST를 구현한다.
- 가중치가 낮은 엣지부터 차례대로 트리를 만들어 나가면서 트리를 하나씩 합치는 구조

② Prim's Algorithm

- 입력 받은 Adjacent matrix를 이용하여 MST를 구현한다.
- 임의의 Vertex부터 가중치비용이 낮은 Edge를 따라 MST를 순서대로 구현하는 구조

2. 주요 부분 코드 설명(알고리즘 부분)

1. Kuruskal's Algorithm

```
void kurusukal(int **matrix, int n){
     struct subset *subsets = (struct subset*) malloc( n * sizeof(struct subset) );
    int count=0;

    /* Kuruskal 알고리즘을 구현하기 전 준비작업  */

    //Edge의 개수세기
    for(int i=0; i<n; i++) {
        for (int j=0; j<n ; j++) {
            if (matrix[i][j] != 0) {
                count++;
            }
        }
    }

    edge edge[count]; //Edge의 정보를 저장할 구조체 instance 선언

    int index=0;

    //Edge의 정보저장
    for(int i=0; i<n; i++) {
        for (int j=0; j<n ; j++) {
            if (matrix[i][j] != 0) {
                edge[index].weight = matrix[i][j];
                edge[index].vertex1 = i;
                edge[index].vertex2 = j;
                index++;
            }
        }

    }

    /* Kuruskal 알고리즘을 구현하기 전 준비작업 끝 */
```

- 입력 받은 adjacent matrix를 가지고 있는 2차원 배열과 데이터의 갯수를 인자로 받는 함수로 만든다.
- Edge의 개수가 얼마나 되는지 이중 for문을 사용하여 adjacent matrix를 검색하여 빈값으로 설정해놓은 값 0을 제외한 나머지 값을 검색하여 개수를 센다.
- Edge의 정보(선의 양 끝에 있는 vertex와 가중치)를 저장한다.

```
    //각각의 Vertex를 부분집합 만들어주기.
    for (int i=0; i < n; i++) {
        Make_set(i, subsets);
    }

    //가중치의 오름차순 정렬을 위한 삽입정렬
    for(int i=1; i<count; i++) {
        struct edge key = edge[i];
        int j = i-1;
        while(j>=0 && edge[j].weight > key.weight ) {
            edge[j+1] = edge[j];
            j--;
        }
        edge[j+1] = key;
    }

    struct edge result[n];
    int z=0;

    //Union해가면서 트리만들기
    for(int i=0; i<count ; i++) {
        int q = Find_set(edge[i].vertex1, subsets);
        int w = Find_set(edge[i].vertex2, subsets);
        if(q!= w){
            result[z++] = edge[i];
            Union(q, w, subsets);

        }
    }
    //결과출력
    printf("쿠르스칼알고리즘 \n");
    for(int i=0; i<z; i++) {
        printf("%d : %d => %d\n", result[i].vertex1, result[i].vertex2, result[i].weight);
    }


}
```

- 각각의 Vertex를 Subset으로 생성해준다.
- 가중치를 오름차순으로 정렬한다. 이때 사용한 정렬 방법은 vertex의 개수가 적으므로 본 과제에서는 삽입 정렬을 이용했다.
- 각각의 부분 집합인 vertex들이 이미 같은 집합으로 묶여있는지 검사를 한다. 이 때, 묶여있지 않다면 해당 vertex의 edge를 선택하므로 결과 배열에 입력해주고 해당 부분 집합을 하나의 집합으로 합쳐준다.
    - 이 작업을 엣지의 개수만큼 반복해주며, 부분 집합을 합치는 과정에서 해당 부분집합은 엣지의 가중치의 오름차순으로 정렬되어 있는 값이기 때문에 가중치가 낮은 값부터 순서대로 합집합 과정을 반복한다.

2. Prim's Algorithm

```
//Prim's Algorithms.
void prim(int **matrix, int n) {
    int visited[n];
    int heapArray[n];
    int a=0;


    for (int i =0; i < n; i++) {
        visited[i] = 0; //배열의 초기화. 0은 방문하지 않은 것, 0이외의 값은 가중치
        //처음에는 모두 방문하지 않았음.
    }
    for (int i = 0; i < n; i++) {

        enQueue(i); //모든 노드값을 큐에 저장
    }


    int i = deQueue(); //초기 시작값

    while(queue->value != -1 || queue->next != NULL) {
        //큐가 비어있지 않을 동안 혹은 큐의 다음값이 널이 아닌 동안
        for(int z = 0; z<n ; z++) {
            heapArray[z] = matrix[i][z]; //추출한 노드값의 가중치를 임시로 힙에 넘겨줄 배열에 저장
        }

        int min = heap_extract_min(heapArray, n); //가중치 중 제일 작은 녀석을 추출

        if (visited[i] == 0 ) { //방문을 안한 녀석이라면
            visited[i] = min; //해당 노드값의 최소 가중치값을 해당 노드값에 저장
        }
        for (int z =0; z<n; z++) {
            if(matrix[i][z] == min){ //만약 매트릭스의 z인덱스 값의 위치의 가중치와 최소값이 같다면
                a = z; //해당 값으로 이동해야하므로 z값을 따로 저장
                break; //반복문 탈출
            }
        }
        printf("%d : %d\n", i, a); //현재 노드값에서 최소값으로 진행할 다음 노드를 출력
        i = deQueue(); //다음 노드값 저장 후 while문 반복실행
    }


}
```

- 입력 받은 adjacent matrix가 저장되어있는 2차원 배열과 vertex의 개수를 넘겨받는 함수로 선언한다.
- 방문 했는지의 여부를 표시하기 위해 데이터 갯수 만큼의 크기를 가진 배열 visited를 선언하여 초기화를 해준다. 값이 0이라면 아직 방문하지 않은 것이고, 0이외의 값이라면 해당 vertex가 가진 Edge의 가중치값이다.
- 모든 Vertex값을 Queue에 넣어준다
- 초기 값을 넣어주기 위해 Queue에서 Vertex 하나를 꺼낸다.
- 이후 반목문을 이용하여 큐가 비어있지 않거나, 큐의 다음값이 NULL이 아닌 동안 반복작업을 한다.
    - 검사할 Vertex와 연결된 다른 Vertex들을 검색하여 가중치중 제일 작

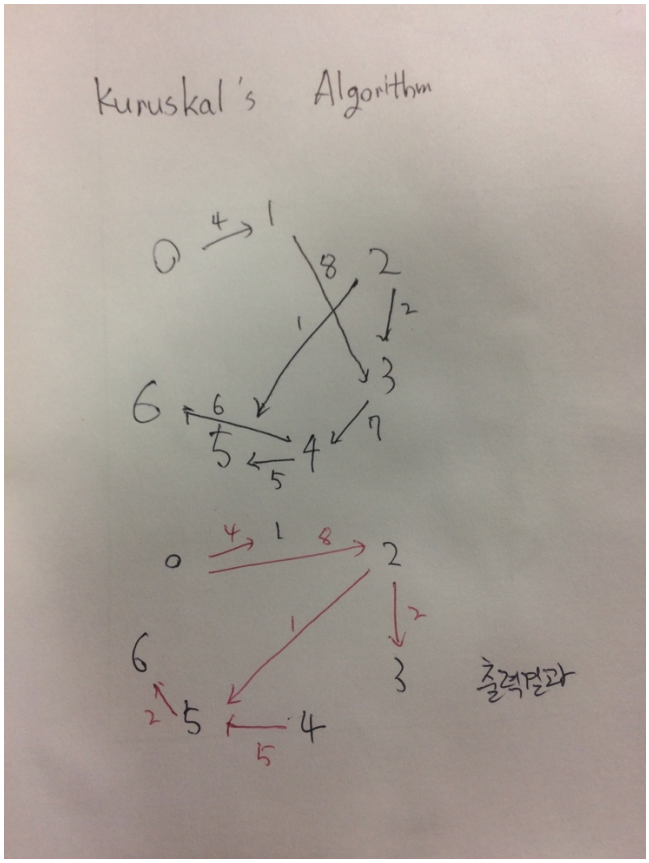
은 값을 뽑아내기 위해, 데이터를 1차원 배열로 옮긴다.

- 옮긴 데이터에서 extract_min을 이용하여 heap구조를 만든 후, 최소값을 추출해낸다.
- 가중치를 visited배열에 삽입한다.
- 해당 최소 가중치값을 가진 Edge로 연결된 vertex값을 찾아 저장한 후 출력한다.
- 다음 Vertex를 검사하기 위해 Queue에서 뽑아낸다.

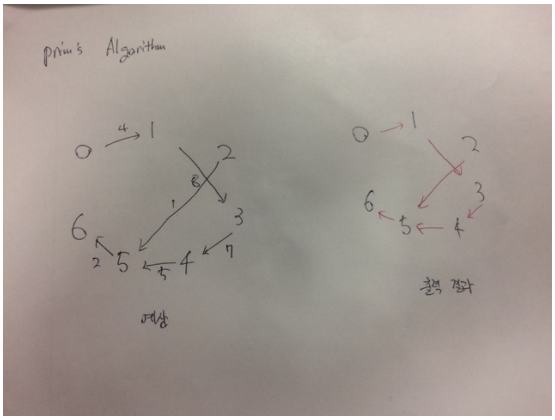
3. 결과

- Prim's Algoritm
  - 각 Vertex를 순서대로 방문하며 Vertex가 가진 Edge중 가장 작은 가중치를 가진 Edge를 선택하여 다음 Vertex로 진행한다.
- Kuruskal's Algorithm
  - 가중치가 가장 작은 Edge부터 순서대로 접근하며 같은 갑의 가중치라면 어느 것을 접근해도 상관없다.
  - 가중치가 제일 낮은 Edge부터 이어나가는 Kuruskal의 알고리즘이 Prim 알고리즘보다 더 낮은 가중치합을 가진 MST를 생성하였다.

- 예상과 결과
  - Kuruskal



    - 예상과는 다르게 출력되었으나 Cycle이 생성되지 않았다.
    - 또한 같은 가중치값을 가진 Edge는 어떤 Edge던 상관없게 생성되었다.
  - Prim



    - 예상대로 나왔다. Vertex를 순서대로 진행하며 방문한 Vertex에서

가장 작은 값의 가중치를 가진 Edge로 진행한다.