알고리즘 실습(9주차)

201000287 일어일문학과 유다훈

1. 과제 설명 및 해결 방법

1. Single Source Shotest Path구현

Single Source Path는 어떠한 한 노드로부터 다른 모든 노드로의 최단 거리를 의미한다. 이번 과제에서는 SSP의 두 가지 알고리즘을 구현한다.

① Bellman Ford Algorithm

- 입력 받은 Adjacent matrix를 이용하여 SSP를 구현한다.
- 가중치가 음수인 엣지가 포함되어 있다. 각 노드에 대하여 모든 엣지에 대해 Relax작업을 통하여 각 노드의 Distance값을 구해낸다.

② Dijkstra's Algorithm

- 입력 받은 Adjacent matrix를 이용하여 SSP를 구현한다.
- 가중치가 음수인 엣지는 없다고 가정을 하고 구현한다.
- 가중치가 작은 버텍스의 엣지들을 검사하여 이어나가는 방식을 한다.
- 이미 방문한 노드에 대해서는 다시 방문하지 않으며, 방문을 한 노드를 제외한 나머지 노드와 엣지들에 대해서만 Relax작업을 반복실행한다.

2. 주요 부분 코드 설명(알고리즘 부분)

1. Bellman Ford Algorithm

```
void bellman_ford(int **matrix, vertex *vertex, edge *edge, int n , int count) {
    initialize_Single_source(matrix, n, vertex); //버텍스 초기화

    printf("결과 출력\n");
    printf("u to v : distance\n");

    for(int i =0 ; i<n ; i++) { //이중 for문을 이용한 릴렉스 작업
        for(int j=0; j<count ; j++) {
            relax(vertex, edge, j);
        }
    }

    for(int i=0; i<count; i++) { //다른 경로가 있는지 재확인하는 방법
        int u = edge[i].vertex1;
        int v = edge[i].vertex2;
        if(vertex[v].distance > (vertex[u].distance + edge[i].weight) ) {
            printf("음수 가중치가 있습니다");
            break;
        }
    }

    for(int i=0; i<n ; i++) {

        printf("%d to %d : %d, %d\n", 0, i, vertex[i].distance, vertex[i].parent); //각 버텍스들의 키값과 부모 출력

    }

}
```

- 입력 받은 adjacent matrix와 vertex배열, edge배열, vertex의 개수와 edge의 개수를 전달받는다.
- 전달받은 vertex들의 distance와 parent값을 초기화한다.
- 이후 모든 vertex의 개수에 모든 edge를 Relax하는 작업을 한다.
- 다시 한 번 음수 가중치로 인해서 잘못된 Distance값이 있는지 확인한다.
- 최종 결과를 출력한다.

2. Dijkstra's Algorithm

```
void dijkstra(int **matrix, int n, vertex *v, edge *edge, int c){

    initialize_Single_source(matrix, n, v); //vertex의 초기화

    int a=0;//vertex number;
    int visitied[n]; //방문했는지 안했는지를 검사하기 위한 배열
    int minDistance[n]; //distance값들을 따로 저장하여 min값을 뽑아내기 위한 배열공간

    /*배열 초기화*/
    for(int i =0; i<n; i++) {
        visitied[i] = 0;
        minDistance[i] = v[i].distance; //초기값으로 각 vertex의 distance값을 다 넣어준다.
    }
    /*배열 초기화*/

    int count=0; //while문을 제어용.

    while(count < n) { //Vertex의 개수만큼 반복실행
        for(int i=0; i<n ;i++) {
            if(visitied[i] != 1) { //방문하지 않았을 때,
                minDistance[i] = v[i].distance; //해당 vertex의 distance를 저장한다.
            } else { //방문했다면
                minDistance[i] = 10000; //그냥 임의로 엄청 높은 값을 넣어준다. 최소값을 뽑는데 방해되지않도록.
            }
        }

        int u = heap_extract_min(minDistance, n); //최소값 추출

        for(int i=0; i<n; i++) {
            if( u == v[i].distance && visitied[i] != 1 ) { //최소값과 i번의 vertex의 distance값이
                같으면서 방문을 하지 않았다면,
                a=i; //해당 vertex의 인덱스를 저장하고
                visitied[i]=1; //방문처리
            }
        }
        for(int i=0; i<c; i++) { //엣지의 개수만큼 반복
            if( a == edge[i].vertex1){ //해당 버텍스를 vertex1(출발지점)으로 가진 edge를 찾는다면
                relax(v, edge, i); //릴렉스 작업 시작
            }
        }

        count++; //while문 반복을 위한 카운터 증가.
    }


    for(int i=0; i<n ; i++) {
        printf("%d to %d : %d, %d\n", 0, i, v[i].distance, v[i].parent); //각 버텍스들의 키값과
            부모 출력

    }


}
```

- 입력 받은 adjacent matrix와 vertex배열, edge배열, 버텍수의 개수 n, edge의 개수 c를 전달받는다.
- 전달받은 vertex 배열을 초기화시켜준다.
- 어떠한 vertex에 방문했는지 안했는지를 검사하는 배열 visited와 distance 값을 따로 저장하여 Extract min작업을 할 배열을 선언하고 초기화한다.
    - visited배열은 값이 0이라면 방문하지 않았고, 1이라면 방문한것이다.

- Vertex의 개수만큼 반복문을 실행한다.
  - 방문하지 않은 vertex라면 해당 vertex의 distance값을 따로 옮긴다.
  - 방문 했다면 최소값을 뽑아내는데 방해가 되지 않도록 임의의 높은 값을 넣는다.
  - vertex의 distance값들 중에서 최소값을 뽑아낸다.
  - 뽑아낸 최소값이 어떤 vertex의 distance인지 찾아낸다. 찾는다면 해당 vertex가 몇번 인지 저장하고, 방문처리를 한다.
  - 최소값을 가진 vertex를 출발지점으로 가지는 edge를 찾아 해당 edge들을 relax하는 작업을 한다.

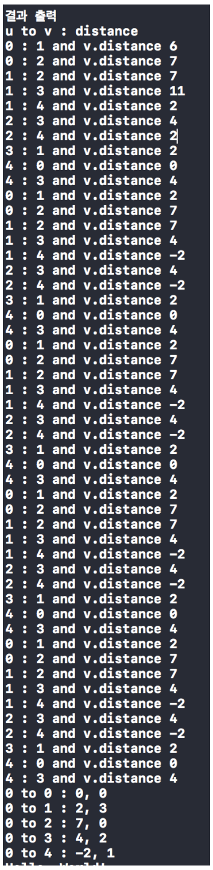
3. Relax and Initialize Single Source

```
void initialize_Single_source(int **matrix, int n, vertex *v) {


    for(int i=0; i<n; i++) {
        v[i].distance = 1000000;
        v[i].parent = NULL;
    }
    v[0].distance = 0;

}

void relax(vertex *vertex, edge * edge, int j) {
    int u = edge[j].vertex1;
    int v = edge[j].vertex2;
    if(vertex[v].distance > (vertex[u].distance + edge[j].weight) ) {
        vertex[v].distance = vertex[u].distance + edge[j].weight;
        vertex[v].parent = u;
    }
    printf("%d : %d and v.distance %d\n", u, v, vertex[v].distance );

}
```

- Initialize Single Source는 vertex들의 초기화를 하는 작업을 한다.
- 본 과제에서 출발 vertex는 0번이므로 0번의 distance를 0으로 지정해준다.
- Relax는 Vertex의 distance와 parent를 바꿔주는 작업을 한다.
- Edge가 도달하는 Vertex의 distance가 Edge가 출발하는 Vertex의 distance와 Edge의 가중치보다 크다면, 도달하는 Vertex의 distance를 해당 값으로 바꿔주고 도착하는 Vertex의 부모를 출발하는 Vertex로 바꾸어준다.
- 해당 작업을 반복하는 것을 통해 최적의 최단거리를 찾아낸다.

3. 결과

1. Bellman Ford Algorithm



```
결과 출력
u to v : distance
0 : 1 and v.distance 6
0 : 2 and v.distance 7
1 : 2 and v.distance 7
1 : 3 and v.distance 11
1 : 4 and v.distance 2
2 : 3 and v.distance 4
2 : 4 and v.distance 2
3 : 1 and v.distance 2
4 : 0 and v.distance 0
4 : 3 and v.distance 4
0 : 1 and v.distance 2
0 : 2 and v.distance 7
1 : 2 and v.distance 7
1 : 3 and v.distance 4
1 : 4 and v.distance -2
2 : 3 and v.distance 4
2 : 4 and v.distance -2
3 : 1 and v.distance 2
4 : 0 and v.distance 0
4 : 3 and v.distance 4
0 : 1 and v.distance 2
0 : 2 and v.distance 7
1 : 2 and v.distance 7
1 : 3 and v.distance 4
1 : 4 and v.distance -2
2 : 3 and v.distance 4
2 : 4 and v.distance -2
3 : 1 and v.distance 2
4 : 0 and v.distance 0
4 : 3 and v.distance 4
0 : 1 and v.distance 2
0 : 2 and v.distance 7
1 : 2 and v.distance 7
1 : 3 and v.distance 4
1 : 4 and v.distance -2
2 : 3 and v.distance 4
2 : 4 and v.distance -2
3 : 1 and v.distance 2
4 : 0 and v.distance 0
4 : 3 and v.distance 4
0 : 1 and v.distance 2
0 : 2 and v.distance 7
1 : 2 and v.distance 7
1 : 3 and v.distance 4
1 : 4 and v.distance -2
2 : 3 and v.distance 4
2 : 4 and v.distance -2
3 : 1 and v.distance 2
4 : 0 and v.distance 0
4 : 3 and v.distance 4
0 to 0 : 0, 0
0 to 1 : 2, 3
0 to 2 : 7, 0
0 to 3 : 4, 2
0 to 4 : -2, 1
```

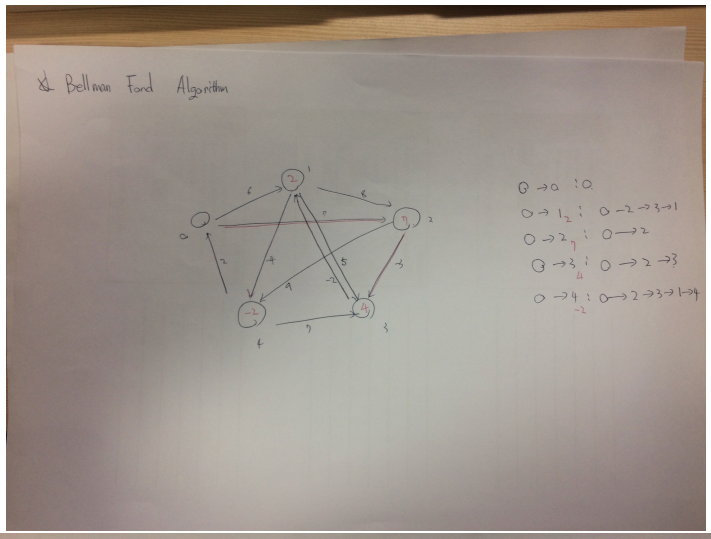
- 모든 Edge를 Vertex의 개수만큼 반복해가며 Relax 작업을 실시한다.
- 어느 반복회수부터는 똑같은 값이 계속해서 출력되는 것을 확인할 수 있다. 이것은 최단경로를 찾아냈다는 결과이다.

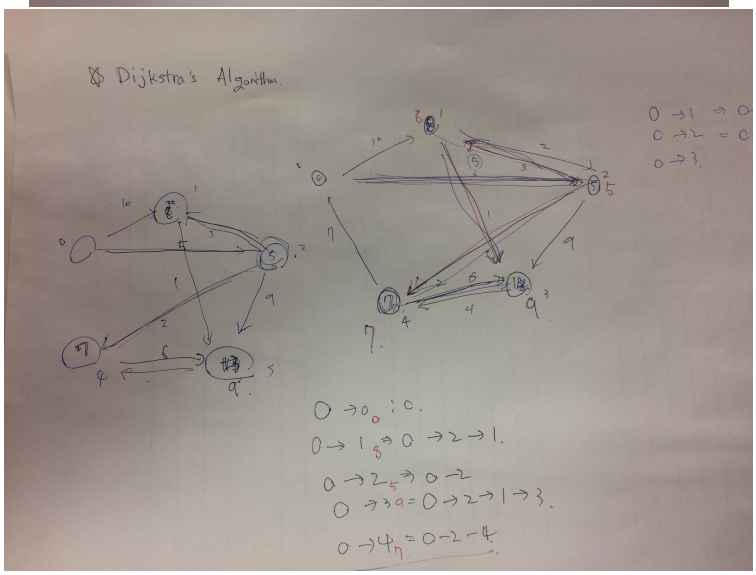
2. Dijkstra's Algorithm

```
0 : 1 and v.distance 10
0 : 2 and v.distance 5
2 : 1 and v.distance 8
2 : 3 and v.distance 14
2 : 4 and v.distance 7
4 : 0 and v.distance 0
4 : 3 and v.distance 13
1 : 2 and v.distance 5
1 : 3 and v.distance 9
3 : 4 and v.distance 7
0 to 0 : 0, 0
0 to 1 : 8, 2
0 to 2 : 5, 0
0 to 3 : 9, 1
0 to 4 : 7, 2
Hello, World!
Program ended with exit code: 0
```

- Distance의 값이 제일 작은 Vertex부터 방문하여 최단거리를 계산한다.
- 4번 Vertex에서 가장 작은 Distance를 갖는것은 0번 Vertex이지만, 이미 방문했으므로 그 다음 가지고 있던 가중치중 제일 작은 가중치를 가지고 있던 1번 Vertex부터 다시 계산한다.

3. 결론





- 이번 과제는 어떠한 노드로부터 다른 모든 노드로가는 최단경로를 구하는 과제이며, 출발노드는 0번노드이다. 따라 각 vertex들이 가진 parent들을 거꾸로 따라올라가면 최소 distance들을 가진 vertex만 나오며 이것이 최단거리이다.