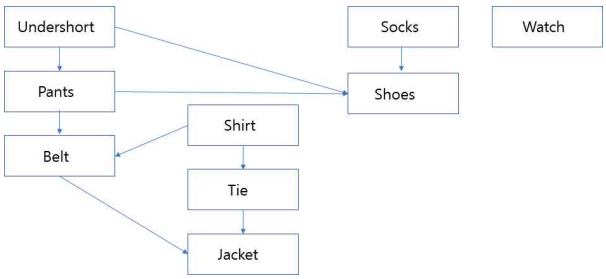
자료구조 과제 4 보고서

201711120 이상목

● **Windows 10, Dev-C++ 5.11, TDM-GCC 4.9.2 64-bit Release 환경에서 작성하였다.**

문제 1 : Topological Sort 구현

1. Topological Sort의 동작



- 왼쪽 그림과 같은 DAG에서 Undershorts->Pants->Belt->Shirt->Tie->Jacket->Socks->Shoes->Watch순이라고 하자.

- DFS를 기반으로 하고, DFS 시행 시 Vertex의 번호가 작은 것(그래프에 삽입된 순서가 빠른 것)부터 탐색한다.

1) 가장 먼저 UnderShorts가 탐색된다. 그 후, Undershorts->Pants->Belt->Jacket을 순서대로 방문한다. Jacket에서 더 이상 진행 불가하므로 Jacket의 탐색은 완료되고 Belt로 돌아간다. Belt에서도 방문할 곳이 없으므로 Belt의 탐색은 완료되고 Pants로 돌아간다. Pants에서 Shoes로 진행한 다음, Shoes에서 더 이상 진행할 수 없으므로 Shoes의 탐색은 완료되고 Pants로 돌아간다. Pants에서 더 이상 진행할 수 없으므로 Pants의 탐색은 완료되고 Undershorts로 돌아간다. UnderShorts에서 더 이상 진행할 수 없으므로 UnderShorts의 탐색은 완료된다. 이 과정까지의 Sort 결과는 다음과 같다.

Undershorts->Pants->Shoes->Belt->Jacket

2) 탐색을 하지 않은 Vertex 중 번호가 가장 작은 Shirt를 시작점으로 하여 DFS를 다시 시행한다. Tie로 진행한 후, Tie에서 더 진행 불가하므로 Tie의 탐색은 완료되고 Shirt로 돌아간다. Shirt에서도 더 이상 진행 불가하므로 Shirt의 탐색은 완료된다. 이 과정까지의 Sort 결과는 다음과 같다.

Shirt->Tie->Undershorts->Pants->Shoes->Belt->Jacket

3) 탐색을 하지 않은 Vertex 중 번호가 가장 작은 Socks를 시작점으로 하여 DFS를 다시 시행한다. Socks에서 DFS를 시작한다. Socks에서 더 이상 진행 불가하므로 Socks의 탐색은 완료된다. 이 과정까지의 Sort 결과는 다음과 같다.

Socks->Shirt->Tie->Undershorts->Pants->Shoes->Belt->Jacket

4) 탐색을 하지 않은 Vertex는 8번 Watch만 남았으므로, Watch를 기준으로 DFS를 시행한다. Watch에서 더 이상 진행 불가하므로 Watch의 탐색은 완료된다. 최종 Sort 결과는 다음과 같다.

Watch->Socks->Shirt->Tie->Undershorts->Pants->Shoes->Belt->Jacket

● **과제 수행 시 Topolocial Sort를 구현한 방법**

: 들어오는 edge가 0인 vertex들을 index가 작은 순서대로 stack에 넣은 다음, stack에서 vertex를 하나씩 뽑아서 vertex를 출력한다. stack에서 vertex를 뽑는 순서는 vertex 번호가 큰 것부터 뽑히게 되어, DFS 시 작은 번호의 vertex부터 방문하는 것과 같게 된다. 뽑은 vertex의 주변 vertex에 들어가는 edge의 수를 1개식 감소시키고, 어떤 vertex에 들어가는 edge의 수가 0이 되면 그 vertex들도 index가 작은 순서대로 stack에 넣는다. stack에 들어있는 것이 0이 될 때 까지 이 과정을 반복하면, Topogical sort 결과가 출력된다.

2. Topological Sort의 구현 방법 및 코드

- Graph : Adjacency List로 구현
- Topological Sort : DFS의 결과 탐색이 완료된 순서대로 Stack에 삽입한 후, Stack의 위부터 entry를 Pop

2-1 Adjacency List Graph 구현 코드

class AdjListGraph : Adjacecy List Graph를 구현하기 위해서, Graph 클래스는 Linked List를 포함한다.

- string vertices[MAX_VTXS] : vertex 이름을 저장한다. MAX_VTXS는 1000으로 하여 1000개보다 많은 Vertex 입력을 방지한다.
- Node(adj[MAX_VTXS] : 각 vertex에 대한 Adjacecy List.
- insertVertex, insertEdge : vertex 삽입 시 string 배열에는 vertex의 이름을 추가. 비어있는 NULL 인 노드를 하나 추가하고, insertEdge시에 해당 노드에 인접 vertex의 정보를 넣은 다음 List에 추가한다.

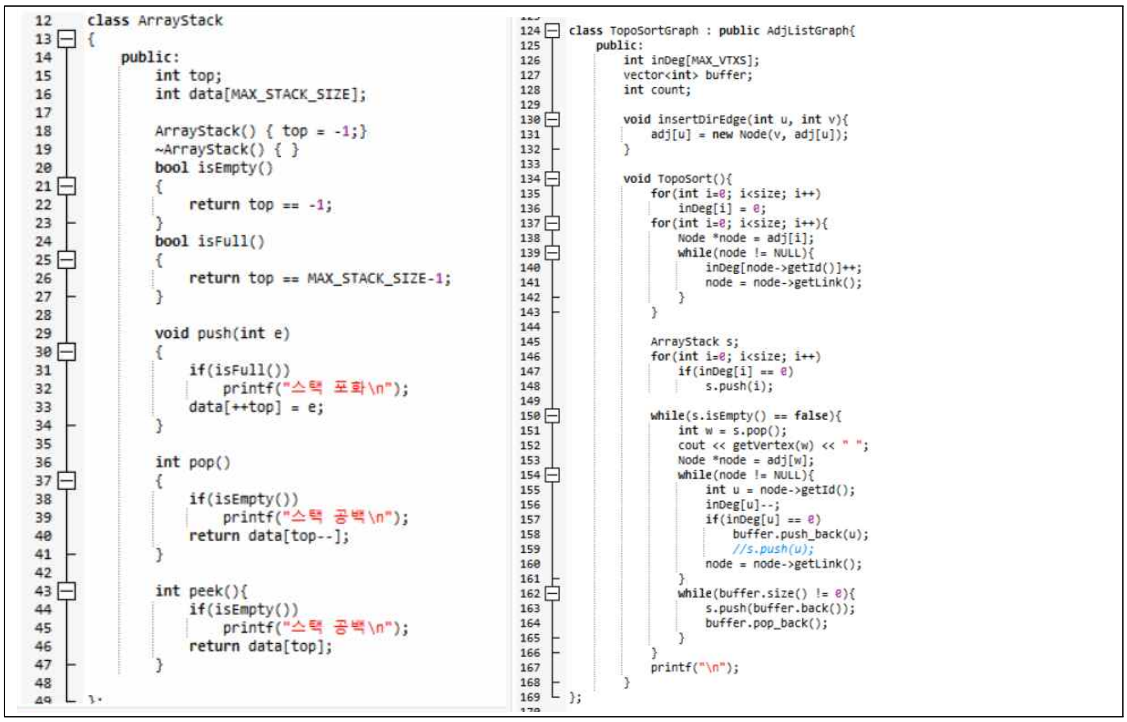
```
82  class AdjListGraph{
83      public:
84          int size;
85          string vertices[MAX_VTXS];
86          Node* adj[MAX_VTXS];
87
88          AdjListGraph() : size(0) {  }
89          ~AdjListGraph() { reset(); }
90          void reset(void){
91              for(int i=0; i<size; i++)
92                  if(adj[i] != NULL)
93                      delete adj[i];
94              size = 0;
95          }
96
97          bool isEmpty()  {return (size==0);}
98          bool isFull() { return (size>=MAX_VTXS);}
99          string getVertex(int i) { return vertices[i];   }
100
101         void insertVertex(string val){
102             if(!isFull()){
103                 vertices[size] = val;
104                 adj[size++] = NULL;
105             }
106             else
107                 printf("Error: 그래프 정점 개수 초과\n");
108
109         }
110
111         void insertEdge(int u, int v){
112             adj[u] = new Node(v, adj[u]);
113             adj[v] = new Node(u, adj[v]);
114         }
115
116 };
```

2-2 Topological Sort/Stack 구현 코드

class ArrayStack : Topological Sort의 순서를 결정하는 데 사용할 Stack

class TopoSortGraph : Topological Sort를 위한 Graph. AdjListGraph를 상속한다.

- TopoSort() : 들어오는 edge가 0인 vertex들을 index가 작은 순서대로 stack에 넣은 다음, stack에서 vertex를 하나씩 뽑아서 vertex를 출력한다. 뽑은 vertex의 주변 vertex에 들어가는 edge의 수를 1개식 감소시키고, 어떤 vertex에 들어가는 edge의 수가 0이 되면 그 vertex도 stack에 넣는다. stack에 들어 있는 것이 0이 될 때 까지 이 과정을 반복하면, Topogical sort 결과가 출력된다.



```
class ArrayStack
{
    public:
        int top;
        int data[MAX_STACK_SIZE];

        ArrayStack() { top = -1;}
        ~ArrayStack() { }
        bool isEmpty()
        {
            return top == -1;
        }
        bool isFull()
        {
            return top == MAX_STACK_SIZE-1;
        }

        void push(int e)
        {
            if(isFull())
                printf("스택 포화\n");
            data[++top] = e;
        }

        int pop()
        {
            if(isEmpty())
                printf("스택 공백\n");
            return data[top--];
        }

        int peek(){
            if(isEmpty())
                printf("스택 공백\n");
            return data[top];
        }
};
```

```
class TopoSortGraph : public AdjListGraph{
    public:
        int inDeg[MAX_VTXS];
        vector<int> buffer;
        int count;

        void insertDirEdge(int u, int v){
            adj[u] = new Node(v, adj[u]);
        }

        void TopoSort(){
            for(int i=0; i<size; i++)
                inDeg[i] = 0;
            for(int i=0; i<size; i++){
                Node *node = adj[i];
                while(node != NULL){
                    inDeg[node->getId()]++;
                    node = node->getLink();
                }
            }

            ArrayStack s;
            for(int i=0; i<size; i++)
                if(inDeg[i] == 0)
                    s.push(i);

            while(s.isEmpty() == false){
                int w = s.pop();
                cout << getVertex(w) << " ";
                Node *node = adj[w];
                while(node != NULL){
                    int u = node->getId();
                    inDeg[u]--;
                    if(inDeg[u] == 0)
                        buffer.push_back(u);
                        //s.push(u);
                    node = node->getLink();
                }
                while(buffer.size() != 0){
                    s.push(buffer.back());
                    buffer.pop_back();
                }
            }
            printf("\n");
        }
};
```

2-3 main 함수

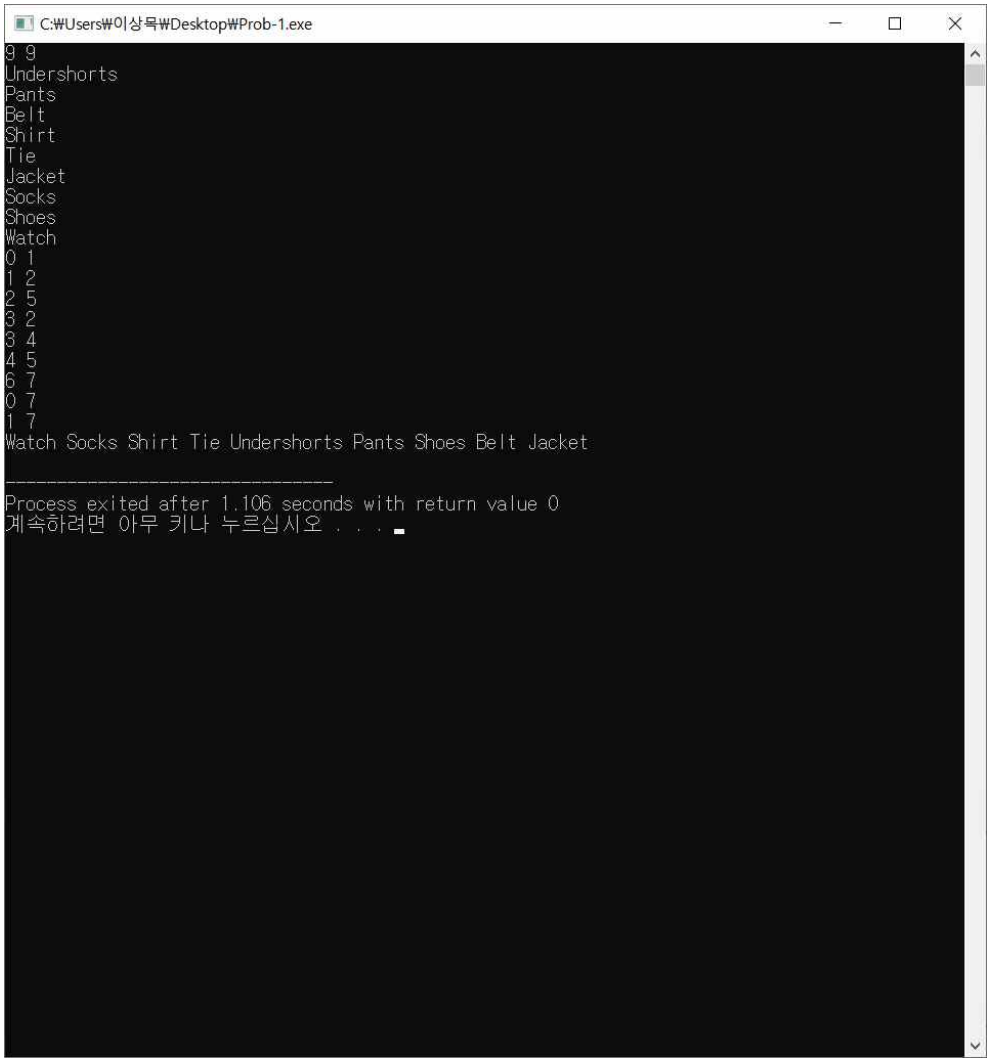
그래프가 저장할 vertex 수와 edge 수를 입력받는다.

그 다음 vertex들을 차례대로 입력하고, 그 사이 edge들을 차례대로 입력한 다음, TopoSort 함수를 실행하면 Topological Sort 결과가 출력된다.

```
int main(void)
{
    TopoSortGraph g;
    int vertexnum;
    int edgenum;
    string vertexname;
    int start;
    int end;

    cin >> vertexnum;
    cin >> edgenum;
    /*if(vertexnum > 1000 || edgenum > 2*vertexnum*(vertexnum-1)){
        printf("Wrong value! Enter vertexnum <= 1000 & edgenum <= 2n(n-1)\n");
        cin >> vertexnum;
        cin >> edgenum;
    }*/

    for(int i=0; i<vertexnum; i++){
        cin >> vertexname;
        g.insertVertex(vertexname);
    }


    for(int j=0; j<edgenum; j++){
        cin >> start;
        cin >> end;
        g.insertDirEdge(start,end);
    }
    g.TopoSort();
}
```

3. 실행 결과 캡쳐

실행 결과, 오류 발생 없이 정상적으로 작동하였다.

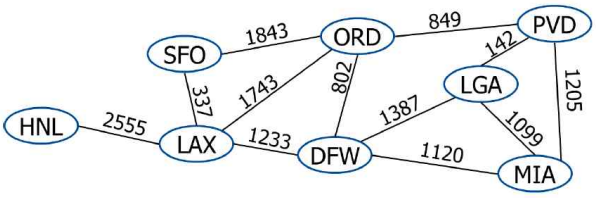


● Windows 10, Dev-C++ 5.11, TDM-GCC 4.9.2 64-bit Release 환경에서 작성하였다.

문제 2 : Weighed Graph 로 Shortest Path 찾기

1. Shortest Path를 찾는 알고리즘의 동작

아래 그림과 같은 n개의 vertex, m개의 edge를 가지는 Graph에서 A지점에서 B지점으로 가는 최단 경로를 찾자.



- Graph는 Adjacency Matrix로 구현
- vertex에 n개의 vertex 삽입 시, 입력 순서대로 vertex에 0~n-1까지의 번호를 부여하고, n*n 배열을 만든다.
- 2차원 배열에서 가로, 세로 index는 vertex의 번호를 의미하고, 2차원 배열 내의 값은 vertex 사이 edge의 weight를 의미한다.
- Graph 초기화 단계에서 vertex의 index가 같은 경우는 edge weight를 0으로 설정하고, 나머지 edge들은 weight를 9999(무한)으로 설정해 준다. (∵ 그래프 내의 모든 edge weight가 9999를 넘지 않기 때문에 무한대 값을 9999로 설정하였다.)
- Graph 초기화를 완료하고, vertex 사이 m개의 edge를 삽입할 때, 어떤 vertex 사이에 얼마의 weight를 가질지를 입력한다. edge weight를 설정해 준 결과는 아래와 같다.

```
   0 1843 9999 9999 9999  337 9999 9999
1843    0  849 9999 9999 1743  802 9999
9999  849    0  142 9999 9999 9999 1205
9999 9999  142    0 9999 9999 1387 1099
9999 9999 9999 9999    0 2555 9999 9999
 337 1743 9999 9999 2555    0 1233 9999
9999  802 9999 1387 9999 1233    0 1120
9999 9999 1205 1099 9999 9999 1120    0
```

- 위 그림과 같이 Graph 생성이 완료되면, vertex 사이 최단 경로를 구한다. 최단 경로를 구하는 알고리즘은 다음과 같다.

```
Algorithm
    for k<-0 to n-1
        for i<-0 to n-1
            for j<-0 to n-1
                A[i][j] = min(A[i][j], A[i][k]+A[k][j])
```

- i번째 vertex에서 j번째 vertex로 가는 최단 경로를 찾는다고 하자. 그리고 그 중간 경로가 k번째 vertex라고 하자. 중간에 거치는 경로가 k번째 vertex라고 할 때, 기존에 저장되어 있던 i번째 vertex에서 j번째 vertex로 가는 경로가 i->k->j 경로보다 크다면, i부터 j로 가는 경로의 정보를 업데이트한다. 이 과정을 모든 중간 vertex k에 대해서, 모든 출발 vertex i와 모든 도착 vertex j에 대해서 반복해 주면 임의의 출발지에서 임의의 도착지 까지 가는 데 걸리는 weight를 저장할 수 있다. 업데이트 한 결과는 다음과 같다.

```
   0 1843 2692 2834 2892  337 1570 2690
1843    0  849  991 4298 1743  802 1922
2692  849    0  142 5147 2592 1529 1205
2834  991  142    0 5175 2620 1387 1099
2892 4298 5147 5175    0 2555 3788 4908
 337 1743 2592 2620 2555    0 1233 2353
1570  802 1529 1387 3788 1233    0 1120
2690 1922 1205 1099 4908 2353 1120    0
```

2. Weighted Graph/Shortest Path를 찾는 알고리즘 구현 방법 및 코드

- Graph : Adjacent Matrix
- Shortest Path : Floyd Warshall 알고리즘으로 Transitive Closure를 찾는 방법 응용

2-1 Graph 구현 코드

class AdjMatGraph : 2차원 배열을 이용한 Graph 클래스

- MAX_VTXS를 200으로 설정하여 vertex를 200개 까지만 받을 수 있게 함

class WGraph : Edge를 입력시 weight를 추가하는 함수가 추가됨. AdjMatGraph를 상속

- insertEdge() : weight 정보를 판단 후, setEdge 함수를 이용하여 weight 입력

class WGraphShortest : 원본 그래프를 복사한 후, 복사한 그래프에서 최단 경로를 찾는다. WGraph를 상속

- ShortestFind() : 복사한 그래프에 대해 앞서 설명한 알고리즘으로 최단 경로를 업데이트.

```
class AdjMatGraph{
    public:
        int size;
        string vertices[MAX_VTXS];
        int adj[MAX_VTXS][MAX_VTXS];  //그래프 원본 2차원 배

        AdjMatGraph() { reset();}
        string getVertex(int i) { return vertices[i];}
        int getEdge(int i, int j) { return adj[i][j];   }
        void setEdge(int i, int j, int val) { adj[i][j] = val;      }

        bool isEmpty() {return size == 0;       }
        bool isFull() {return size>=MAX_VTXS;       }

        void reset(){
            size = 0;
            for(int i=0; i<MAX_VTXS; i++){
                for(int j=0; j<MAX_VTXS; j++)
                    setEdge(i,j,0);
            }
        }
```

```
class WGraph : public AdjMatGraph{
    public:
        void insertEdge(int u, int v, int weight){
            if(weight > INFINITE)
                weight = INFINITE;
            setEdge(u,v,weight);
        }

        bool hasEdge(int i, int j){
            return (getEdge(i,j)<INFINITE);
        }
};
```

```
class WGraphShortest : public WGraph{
    public:
        int A[MAX_VTXS][MAX_VTXS];

        void ShortestPathFind(int vertexN){
            for(int i=0; i<vertexN; i++){
                for(int j=0; j<vertexN; j++){
                    A[i][j] = adj[i][j]; //원본 그래프를 새 배열에 복사
                }
            }

            for(int k=0; k<vertexN; k++){
                for(int i=0; i<vertexN; i++){
                    for(int j=0; j<vertexN; j++){
                        if(A[i][k]+A[k][j] < A[i][j])
                            A[i][j] = A[i][k]+A[k][j];
                    }
                }
                //printA();
            }
        }
```

2-2 Shortest Path를 찾는 알고리즘 구현 코드

```
for(int k=0; k<vertexN; k++){
    for(int i=0; i<vertexN; i++){
        for(int j=0; j<vertexN; j++){
            if(A[i][k]+A[k][j] < A[i][j])
                A[i][j] = A[i][k]+A[k][j];
        }
    }
    //printA();
}
```

3개의 For문으로 구성된다. 각 For 문의 반복 횟수는 입력한 vertex 수 만큼이다. i번째 vertex에서 j번째 vertex로 가는 최단 경로를 찾는다고 하자. 그리고 그 중간 경로가 k번째 vertex라고 하자. 중간에 거치는 경로가 k번째 vertex라고 할 때, 기존에 저장되어 있던 i번째 vertex에서 j번째 vertex로 가는 경로가 i->k->j 경로보다 크다면, i부터 j로 가는 경로의 정보를 업데이트한다. 이 과정을 모든 중간 vertex k에 대해서, 모든 출발 vertex i와 모든 도착 vertex j에 대해서 반복해 주면 임의의 출발지에서 임의의 도착지 까지 가는 데 걸리는 weight를 저장할 수 있다.

2-3 main 함수 (두 지점 사이의 Shortest Path를 찾아서 출력)

```
115     cin >> vertexnum;
116     cin >> edgenum;
117     /*if(vertexnum > 200 || edgenum > 10000){
118         printf("Wrong value! Enter vertexnum <=200 & edgenum <=10000\n");
119         cin >> vertexnum;
120         cin >> edgenum;
121     }*/
122     cin >> startpoint;
123     cin >> destination;
124
125     for(int i=0; i<vertexnum; i++){
126         for(int j=0; j<vertexnum; j++){
127             g.adj[i][j] = 9999;
128         }
129     }
130     for(int i=0; i<vertexnum; i++){
131         g.adj[i][i] = 0;
132     }
133
134     for(int i=0; i<vertexnum; i++){
135         cin >> vertexname;
136         g.insertVertex(vertexname);
137         names.push_back(vertexname);
138
139     }
140     for(int j=0; j<edgenum; j++){
141         cin >> s;
142         cin >> e;
143         cin >> weight;
144         for(int k=0; k<vertexnum; k++){
145             if (names[k] == s)
146                 sidx = k;
147             if (names[k] == e)
148                 eidx = k;
149         }
150         g.insertEdge(sidx,eidx,weight);
151         g.insertEdge(eidx,sidx,weight);
152     }
153
154     g.ShortestPathFind(vertexnum);
155
156     for(int p=0; p<vertexnum; p++){
157         if(names[p] == startpoint)
158             sp = p;
159         if(names[p] == destination)
160             dp = p;
161     }
```

- 그래프가 저장할 vertex 수와 edge 수를 입력받는다.
- 최단 경로를 구할 출발지와 도착지를 입력한다.
- 2차원 배열에서 index가 같은 부분은 weight를 0, 그 외는 weight를 9999로 초기화한다.
- 입력한 vertex들의 index를 따로 저장하기 위해서 vertex의 이름들을 순서대로 vector에 저장한다.
- edge의 weight를 입력할 때, (vertex이름, vertex이름, weight) 순으로 입력받기 때문에, vertex 이름을 index로 바꿔 주기 위해서 vector에서 출발, 도착지 vertex 이름을 검색한 다음, 출발지, 도착지의 index를 저장한다. index를 이용하여 Graph에 weight를 저장한다.
- 그래프에 대해서 최단 경로를 찾는다.
- 최단 경로를 저장한 2차원 배열에서 A[출발지 index][도착지 index]를 출력하면 두 지점 사이의 최단 경로가 출력된다.

3. 실행 결과 캡쳐

실행 결과, 최단 경로는 정상적으로 출력되나, 최단 경로에서 거치게 되는 중간 지점 Vertex의 이름은 구현 코드에서 직전 vertex에 대한 정보를 저장하지 않아서 출력되지 않았다. 코드 실행 시 오류는 발생하지 않았다.