Programming Report #1

All-Pair-Shortest-Paths using Dijkstra's Algorithm

과목	: 컴퓨터알고리즘
담당교수	: 이상호
학과	: 컴퓨터공학과
학번	: 1829008
이름	: 김민영
이메일	: kmy8759@ewhain.net
제출일자	: 2020년 4월 18일

[Programming Report #1] : All-Pair-Shortest-Paths using Dijkstra's Algorithm

1) 문제설명

가중치 방향 그래프(weighted directed graph)를 파일로부터 입력받아서 가중치 인접행렬(adjacent Matrix)로 나타내어 출력하고, 이 인접 행렬을 이용하여 그래프의 각 정점을 출발 정점으로 하여 나머지 모든 정점까지의 최단경로를 다익스트라의 알고리즘을 이용하여 구하고 출력한다.

2) 입출력의 예

(1) 입력 자료 형식의 예 : 파일형식(graph1.txt~graph6.txt)

* graph1.txt

```
3 5              // 정점의 개수 n=3, 방향 에지의 개수 m=5
1 2 4            //<v1,v2> weight : 4
1 3 11           //<v1,v3> weight : 4
2 1 6            //<v2,v1> weight : 4
2 3 2            //<v2,v3> weight : 4
3 1 3            //<v3,v1> weight : 4
```

(2) 출력 자료 형식의 예

```
<graph1.txt>
Adjacency Matrix
0 4 11
6 0 2
3 999 0

Shortest paths from v1 = ( 0 4 6 )
Shortest paths from v2 = ( 5 0 2 )
Shortest paths from v3 = ( 3 7 0 )
```

3) 문제풀이방법(알고리즘)

하나의 vertex에서 다른 vertex 까지의 최단 경로를 찾는 알고리즘에는 여러가지 방법이 있지만 본 과제에서는 다익스트라 알고리즘을 이용하여 문제를 해결했다.

<Key idea> : 그리디 알고리즘

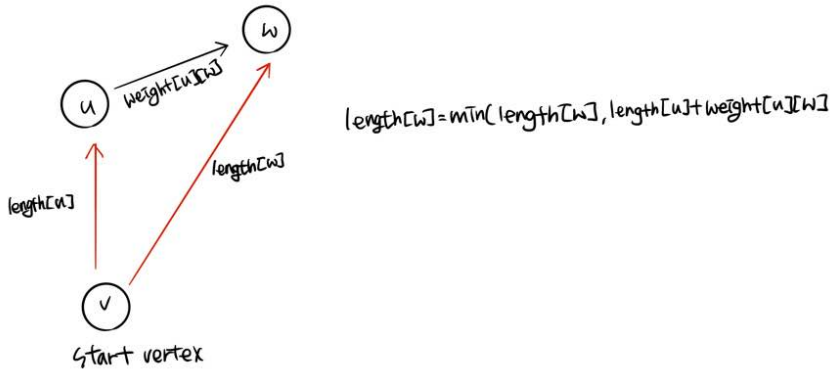
전체적인 문제풀이 순서

0. 우선 txt 입력파일을 읽어와 adjacent Matrix를 초기화 한다(그래프의 각 vertex에 대해 length가 정의되어 자기자신으로 가는 경우 0이고, 나머지는 INF(9999)값으로 초기화한다.)

1. 집합 F를 정의하고 공집합으로 초기화한다.

2. F에 속해있지 않은 것중에 length값이 최소가 되는 vertex를 선택하여 F에 추가한다.

3.기존의 length(i=1...k) 값보다 length(u)+weigth(u,w)의 값이 더 작다면 length의 값을 length(u)+weight(u,w)로 수정해준다. 즉 아래그림과 같다.



4. start와 v가 일치할 때까지 3번과 4번을 반복해준다. 최종적으로 얻어지는 length의 값이 s에서 임의의 정점 u까지의 최단 경로의 길이이다.

즉, 위의 방식을 살펴보면 결국 각 단계별로 전체-F의 원소중에서 length의 값이 최소가 되는 원소를 선택하는 "그리디 알고리즘"을 적용한다. 이 알고리즘이 항상 optimal 하다는 증명은 6)결과분석 및 토의에서 확인할 수 있다.

각 함수설명

read_adjacentM 함수 : 파일을 읽어와 adjacent Matrix(W)를 만든다.

방법 -> 우선 W[i][j]를 INF 로 초기화를 하고, 본인에서 본인으로 가는 경우(대각선)를 0 으로 만들어준다. 이후 파일에서 읽어온 edge 들의 weight 들을 넣어준다.

print_adjacentM 함수 : adjacent Matrix 를 출력한다.

방법 -> W[i][j]의 모든 원소를 반복문을 이용해 출력한다. 이때, 출력화면의 가독성을 위해 tab 을 이용하거나, %5d 를 이용한다.

dijkstra 함수 : 다익스트라 알고리즘 구현

방법 -> 1~4 방식으로 이를 구현한다.

Main 함수 : read_adjacentM() 함수를 이용하여 파일을 읽어와 인접행렬을 만들고, print_adjacentM 함수를 이용하여 인접행렬을 출력한다. 또한 start vertex 를 1~n 으로 하여 dijkstra 함수를 이용하여 최단경로를 구하고 출력한다.

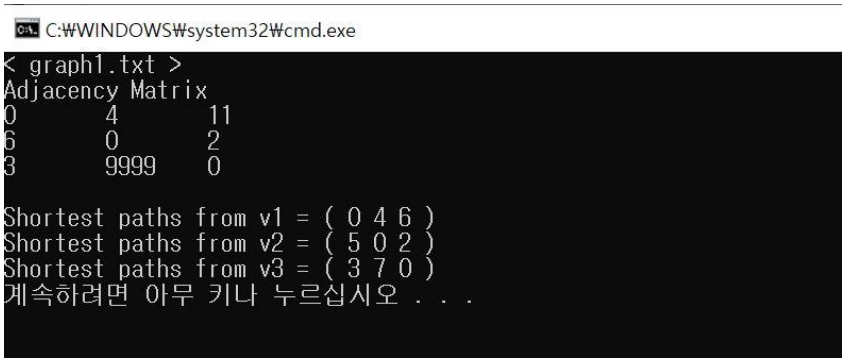
4) 소스코드(소스파일.cpp)

```
// 1829008 김민영
// computer algorithm
// Programing Report1 : 다익스트라 알고리즘

#include <stdio.h>
#define INF 9999; // Edge가 없는 경우의 가중치
int W[101][1001]; // adjacent Matrix
int F[101]; // edge set of shortest path tree rooted at Start
int length[101]; // 집합 F에서 각 vertex까지의 길이
int vnear = 0;
int n;

// 파일을 읽어와 adjacent Matrix W를 만든다.
void read_adjacentM() {
	FILE *fp;
	int m;
	fp = fopen("C:/Temp/PR1_data/graph6.txt", "r"); // txt 파일 읽어오기
	if (fp == NULL) { // 파일을 읽어오지 못하는 경우에 대한 예외처리
		printf("could not read file");
		return;
	}
	printf("< graph6.txt >\n");
	fscanf(fp, "%d", &n);
	fscanf(fp, "%d", &m);
	for (int i = 1; i <= n; i++) {
		for (int j = 1; j <= n; j++) {
			W[i][j] = INF;
		}
	}
	// 대각선 0으로
	for (int i = 1; i <= n; i++) {
		W[i][i] = 0;
	}
	// weight 초기화
	for (int i = 1; i <= m; i++) {
		int node1, node2, weight;
		fscanf(fp, "%d", &node1);
		fscanf(fp, "%d", &node2);
		fscanf(fp, "%d", &weight);
		W[node1][node2] = weight;
	}
}

// adjacent Matrix 출력
void print_adjacentM() {
	printf("Adjacency Matrix\n");
	for (int i = 1; i <= n; i++) {
		for (int j = 1; j <= n; j++) {
			printf("%5d", W[i][j]);
			//or printf("%d\t",W[i][j]);
		}
		printf("\n");
	}
}

// 다익스트라 알고리즘
void dijkstra(int n, int W[][1001], int F[], int start) {
	int min;
	// int touch[101];
	// length에 start vertex에서 해당 vertex 까지의 길이를 저장해놓는다
```

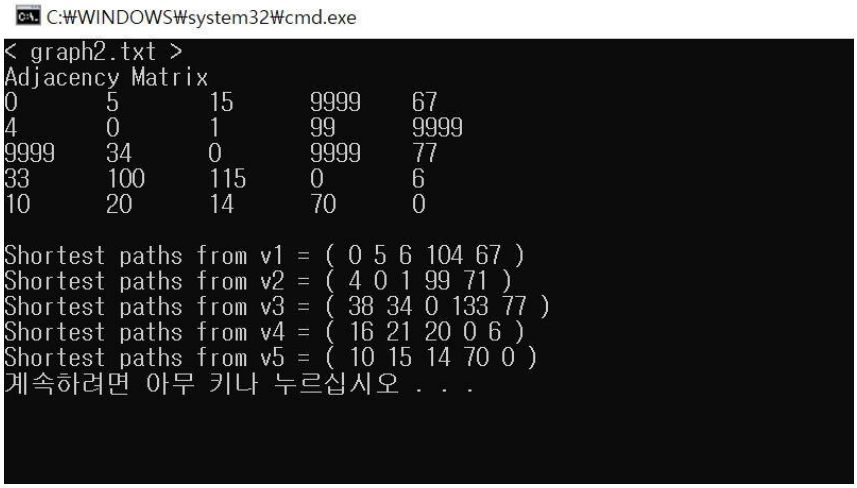
```
	for (int i = 1; i <= n; i++) {
		if (i != start) {
			// touch[i]= 1;
			length[i] = W[start][i];
		}
	}
	// start vertex 에서 start vertex 사이의 거리는 0 이다.
	F[start - 1] = 0;
	// vnear : 다음에 방문할 가까운 vertex
	for (int i = 1; i < n; i++) {
		min = INF;
		for (int i = 1; i <= n; i++) {
			if (i != start) {
				if (0 <= length[i] && length[i] < min) {
					min = length[i];
					vnear = i;
				}
			}
		}
		// touch , length 의 변화값 수정
		for (int i = 1; i <= n; i++) {
			if ((i != start) && (W[vnear][i] != 9999)) {
				if (length[vnear] + W[vnear][i] < length[i]) {
					length[i] = length[vnear] + W[vnear][i];
					//touch[i] = vnear;
				}
			}
		}
		// 방문한 vertex 값 min 을 F 에 저장하고 length 에는 방문했음에 대한 표시로 -
1 을 저장한다.
		F[vnear - 1] = min;
		length[vnear] = -1;
	}
	// 출력
	printf("Shortest paths from v%d = ", start);
	printf("( ");
	for (int i = 0; i < n; i++) {
		printf("%d ", F[i]);
	}
	printf(")");
	printf("\n");
}

int main(void) {
	read_adjacentM();
	print_adjacentM();
	printf("\n");
	// start vertex 지정
	for (int start = 1; start <= n; start++) {
		dijkstra(n, W, F, start);
	}
	return 0;
}
```

5) 수행결과(캡쳐화면)

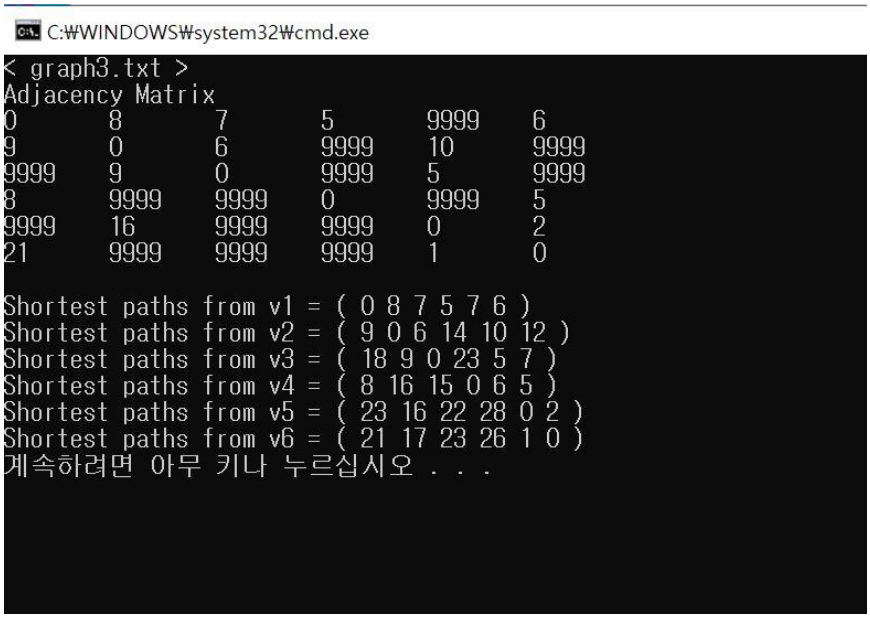
(1) graph1.txt



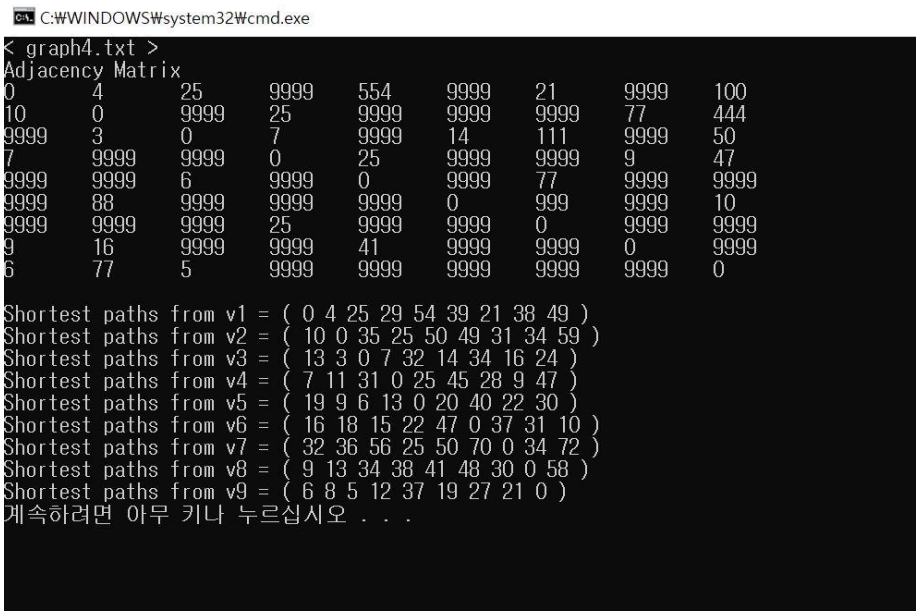
(2) graph2.txt



(3) graph3.txt



(4) graph4.txt



(5) graph5.txt

(6) graph6.txt

```
< graph6.txt >
Adjacency Matrix
   0   15   60  666 9999  211   17   65 9999 9999 9999 9999 9999 9999 9999   12   15 9999 9999   77
  59    0 9999   88 9999 9999 9999 9999 9999 9999   49 9999 9999 9999 9999 9999 9999 9999 9999 9999
   3  166    0   36   48  778   59 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
9999 9999    9    0 9999   56 9999  222   49 9999 9999 9999 9999 9999 9999  148    4    1 9999 9999
9999 9999 9999 9999    0 9999   58   91 9999   25 9999  100 9999 9999 9999 9999  200 9999  211 9999
9999 9999 9999   26   27    0   84 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
9999 9999 9999 9999 9999   89    0   48  647   36   69 9999 9999 9999 9999   78  896   29   44   39
9999 9999 9999 9999 9999   99  548    0 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999
9999 9999 9999 9999 9999 9999 9999 9999    0 9999 9999 9999 9999 9999 9999 9999 9999   73   95 9999
9999 9999 9999 9999 9999    1 9999 9999   72    0   90 9999 9999 9999 9999  135 9999 9999 9999 9999
9999 9999 9999 9999 9999 9999 9999 9999 9999 9999    0 9999 9999 9999 9999  456   84 9999   73 9999
9999 9999 9999   77   56   48 9999 9999 9999 9999 9999    0 9999 9999  200   51   53 9999 9999 9999
9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999    0 9999 9999   52   63   91 9999   92
9999 9999 9999 9999 9999 9999 9999 9999   49  999   30   57 9999    0 9999 9999 9999 9999 9999 9999
   4   66   75   34   44  665 9999 9999 9999 9999 9999 9999 9999 9999    0 9999 9999 9999 9999 9999
9999 9999    1 9999 9999   64   61   30 9999 9999 9999 9999 9999 9999 9999    0 9999 9999 9999 9999
   2 9999 9999    1   49   23 9999 9999 9999 9999 9999 9999 9999 9999 9999 9999    0 9999 9999 9999
9999 9999 9999 9999  999 9999 9999   21 9999 9999   59 9999 9999   64 9999 9999   61    0 9999 9999
9999 9999 9999 9999 9999  185  186  781  911 9999 9999 9999 9999 9999 9999 9999 9999 9999    0   69
  10 9999   15 9999 9999 9999   60 9999   80 9999  100  111  120 9999 9999 9999 9999 9999  180    0
```

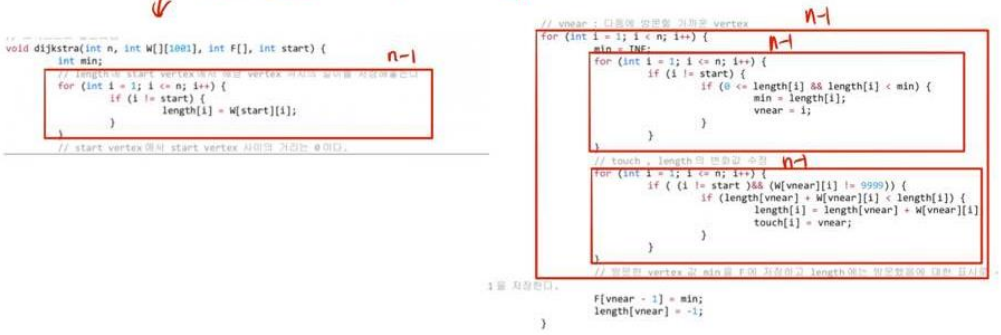
```
Shortest paths from v1 = ( 0 15 13 16 61 38 17 38 65 53 64 138 176 81 338 12 15 17 61 56 )
Shortest paths from v2 = ( 59 0 72 75 120 97 76 97 124 112 49 197 235 140 397 71 74 76 120 115 )
Shortest paths from v3 = ( 3 18 0 19 48 41 20 41 68 56 67 141 179 84 341 15 18 20 64 59 )
Shortest paths from v4 = ( 6 21 9 0 53 27 23 22 49 59 60 122 182 65 322 18 4 1 67 62 )
Shortest paths from v5 = ( 58 73 61 52 0 26 58 74 97 25 112 100 217 117 300 70 56 53 102 97 )
Shortest paths from v6 = ( 32 47 35 26 27 0 49 48 75 52 86 127 208 91 327 44 30 27 93 88 )
Shortest paths from v7 = ( 49 64 54 63 64 37 0 48 108 36 69 150 159 93 350 61 64 29 44 39 )
Shortest paths from v8 = ( 131 146 134 125 126 99 148 0 174 151 185 226 307 190 426 143 129 126 192 187 )
Shortest paths from v9 = ( 136 151 144 135 183 157 153 94 0 189 132 194 284 137 394 148 134 73 95 164 )
Shortest paths from v10 = ( 33 48 36 27 28 1 50 49 72 0 87 128 209 92 328 45 31 28 94 89 )
Shortest paths from v11 = ( 86 101 94 85 133 107 103 107 134 139 0 207 262 150 407 98 84 86 73 142 )
Shortest paths from v12 = ( 55 70 52 54 56 48 72 76 103 81 114 0 231 119 200 51 53 55 116 111 )
Shortest paths from v13 = ( 56 71 53 64 101 86 73 82 113 109 120 186 0 129 386 52 63 65 117 92 )
Shortest paths from v14 = ( 112 127 109 111 113 105 129 133 49 138 30 57 288 0 257 108 110 112 103 168 )
Shortest paths from v15 = ( 4 19 17 20 44 42 21 42 69 57 68 142 180 85 0 16 19 21 65 60 )
Shortest paths from v16 = ( 4 19 1 20 49 42 21 30 69 57 68 142 180 85 342 0 19 21 65 60 )
Shortest paths from v17 = ( 2 17 10 1 49 23 19 23 50 55 61 123 178 66 323 14 0 2 63 58 )
Shortest paths from v18 = ( 63 78 71 62 110 84 80 21 111 116 59 121 239 64 321 75 61 0 124 119 )
Shortest paths from v19 = ( 79 94 84 95 132 117 96 117 144 132 143 180 189 160 380 91 94 96 0 69 )
Shortest paths from v20 = ( 10 25 15 26 63 48 27 48 75 63 74 111 120 91 311 22 25 27 71 0 )
계속하려면 아무 키나 누르십시오 . . .
```

6) 결과분석 및 토의

1. Time complexity

$T(n) = (n-1) + 2(n-1)(n-1)$

첫번째반복문 두번째



```
void dijkstra(int n, int W[][1001], int F[], int start) {
    int min;
    // length의 start vertex에서 해당 vertex 까지의 최단거리를 저장해줍니다
    for (int i = 1; i <= n; i++) {
        if (i != start) {
            length[i] = W[start][i];
        }
    }
    // start vertex에서 start vertex 사이의 거리는 0이다.
```

n-1

```
// vnear : 다음에 방문할 거쳐갈 vertex
for (int i = 1; i < n; i++) {
    min = INF;
    for (int i = 1; i <= n; i++) {
        if (i != start) {
            if (0 <= length[i] && length[i] < min) {
                min = length[i];
                vnear = i;
            }
        }
    }
    // touch : length의 변화값 수정
    for (int i = 1; i <= n; i++) {
        if ( (i != start )&& (W[vnear][i] != 9999)) {
            if (length[vnear] + W[vnear][i] < length[i]) {
                length[i] = length[vnear] + W[vnear][i];
                touch[i] = vnear;
            }
        }
    }
    // 방문한 vertex 값 min을 F의 거쳐야하고 length에는 방문했음에 대한 표시를
    1을 저장한다.
    F[vnear - 1] = min;
    length[vnear] = -1;
}
```

n-1 n-1 n-1

$= 2(n-1)^2 + (n-1) \quad \in \Theta(n^2)$

$\therefore$ 이 알고리즘의 시간복잡도는 $\Theta(n^2)$이다.

2. 위의 다익스트라 알고리즘은 모든 edge가 음수가 아닐 때 항상 최단경로를 보장한다.

-> 증명

$e = (y, z)$, $y \in V'$, $z \in V - V'$라하자. * 다익스트라 알고리즘에 의해 선택된 edge

또한, $v, x_1 \cdots x_r, y$는 $v \to y$로의 최단경로라하자.

$P = v, x_1 \cdots x_r, y, z$라하면 $W(P) = d(v,y) + W(e)$

$P' = v, z_1, \cdots z_{k-1}, z_k \cdots, z$는 $v \to z$로의 다른 경로라하자.

그러면 우리는 $W(P) \le W(P')$임을 보이면 된다.

$z_{k-1} \in V'$이고 $z_k \in V - V'$이라하자.

(z_{k-1}, z_k) : candidate edge in 다익스트라 알고리즘

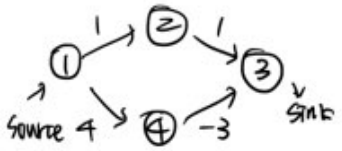
$W(P) = d(v,y) + W(e) \le d(v, z_{k-1}) + W(z_{k-1}, z_k)$

by choice of $e \in W(P')$

($\because$ $v, z_1, \cdots z_k$은 P'의 일부)

따라서 다익스트라 알고리즘은 Edge의 weight가 음수가 아닐 때 항상 최단경로를 제공한다.

* edge가 음수일때 correct X 테케



$v_1 \rightarrow v_4 \rightarrow v_3$: 이 optimal case이기 때문에
correct X

=> 우리의 출력예제에서는 weight 가 모두 음수가 아니기 때문에 최단경로가 예상한대로 잘 출력이 된 것을 확인할 수 있다.