알고리즘 과제 2

Floyd's Algorithm

제출 마감일 : 2016.11.05

담당교수 : 최지웅 교수님
소속학부 : 컴퓨터학부
학번 : 20142577
이름 : 홍상원

1. Floyd 알고리즘 개요

(1) 정의 : 그래프에서 모든 vertex 사이의 최단 경로의 거리를 구하는 알고리즘
- 그래프에서 최단 경로의 거리 표현
: D(k)[i][j]: {v1, v2, v3, … , vk}의 vertex 들만을 통해서 vi 에서 vj 로 가는 최단 경로의 거리
(o<= k <= n, n 은 vertex 개수)

(2) Dynamic Programming 을 이용하여 해결
1)재귀적 관계식 정립
: D(k)[i][j] = minimum(D(k-1)[i][j], D(k-1)[i][k] + D(k-1)[k][j])
- D(k)[i][j] = D(k-1)[i][j]
: {v1, v2, v3, … , vk}의 vertex 들만을 통해서 vi 에서 vj 로 가는 최단 경로가 vk 를 거치지 않는 경우
- D(k)[i][j] = D(k-1)[i][k] + D(k-1)[k][j]
: {v1, v2, v3, … , vk}의 vertex 들만을 통해서 vi 에서 vj 로 가는 최단 경로가 vk 를 거치는 경우

2) 상향식 방식(Bottom-Up)으로 해결

2. Floyd 알고리즘 구현

- problem: 가중치가 포함된 그래프의 각 vertex 에서 다른 모든 vertex 까지의 최단거리를 계산하고, 각각의 경로를 구하라.
- Input: 가중치 포함 그래프 W 와 그 그래프에서의 vertex 의 수 n
- Output: 최단경로의 거리와 최단경로가 방문하는 vertex

[floyd.c]

```
#include <stdio.h>
#define N 5 //N: vertex 의 개수
#define INFINITY 10000 //무한대 설정

/* number, index 를 int type 으로 정의 */
typedef int number;
typedef int index;

/*
* < Floyd's Algorithm >
* n: vertex 의 개수
* W: 그래프를 인접행렬로 나타냄
* D[i][j]: 각 vi 에서 vj 까지 가는 최단 거리
* P[i][j]
```

* - vi 에서 vj 까지 가는 최단 경로의 중간에 놓여 있는 정점이 최소한 하나는 있는 경우 -> 그 놓여 있는 정점 중에서 가장 큰 index 대입
* - 최단경로의 중간에 놓여 있는 정점이 없는 경우 -> 0 대입
*/
```
void floyd(int n, const number (*W)[n + 1], number (*D)[n + 1], index (*P)[n + 1]) {
	index i, j, k;
	//P[i][j]를 0 으로 초기화
	for(i = 1; i <= n; i++)
		for(j = 1; j <= n; j++)
			P[i][j] = 0;

	/* vi 에서 vj 까지 다른 vertex 를 거치지 않고 직접 가는 경우는 W 와 같다 */
	for(i = 1; i <= n; i++)
		for(j = 1; j <= n; j++)
			D[i][j] = W[i][j];

	/* 최단경로의 거리 구하기 */
	for(k = 1; k <= n; k++) {
		for(i = 1; i <= n; i++) {
			for(j = 1; j <= n; j++) {
				//D[i][j] = minimum(D[i][j], D[i][k] + D[k][j]);
				if(D[i][k] + D[k][j] < D[i][j]) {
					P[i][j] = k;
					D[i][j] = D[i][k] + D[k][j];
				}
			}
		}
	}
}

/* 최단경로로 갈 때 지나가는 vertex 를 찾는 함수 */
void path(index q, index r, int (*P)[N + 1]) {
	if(P[q][r] != 0) {
		path(q, P[q][r], P);//vi 에서 v(k-1)까지 최단경로
		printf("v%d ", P[q][r]);//vk
		path(P[q][r], r, P);//v(k+1)에서 vj 까지 최단경로
	}
}

int main(void) {
	/*
	*  directed, weighted graph 의 인접행렬
```

```
	*  W[i][j]
	*  - vi 에서 vj 로 가는 edge 의 가중치 값
	*  - v1 에서 vj 로 가는 edge 가 없으면 무한대(INFINITY) 대입
	*  - i == j 이면 0 대입
	*/
	//vertex 를 1 부터 시작하기 위해 배열 index 가 0 인 부분은 사용하지 않음(사용하는 않는 공간은 -1 대입)
	const number W[N + 1][N + 1] = {

		{-1, -1, -1, -1, -1, -1},
		{-1, 0, 1, 1000, 1, 5},
		{-1, 9, 0, 3, 2, INFINITY},
		{-1, INFINITY, INFINITY, 0, 4, INFINITY},
		{-1, INFINITY, INFINITY, 2, 0, 3},
		{-1, 3, INFINITY, INFINITY, INFINITY, 0}
	};

	number D[N + 1][N + 1] = {-1};
	index P[N + 1][N + 1] = {-1};
	index i, j;

	floyd(N, W, D, P);

	//모든 최단경로 출력
	for(i = 1; i <= N; i++) {
		for(j = 1; j <= N; j++) {
			if(i == j)//자기 자신을 가리키므로 제외
				continue;
			// 최소 가중치 출력
			printf("v%d 에서 v%d 로 가는 최단 경로의 거리: %d₩n", i, j, D[i][j]);

			//최단경로로 갈 때 방문하는 모든 vertex 출력
			printf("최단경로: v%d ", i);
			if(P[i][j] != 0)
				path(i, j, P);
			printf("v%d", j);
			printf("₩n₩n");
		}
	}

	return 0;
}
```

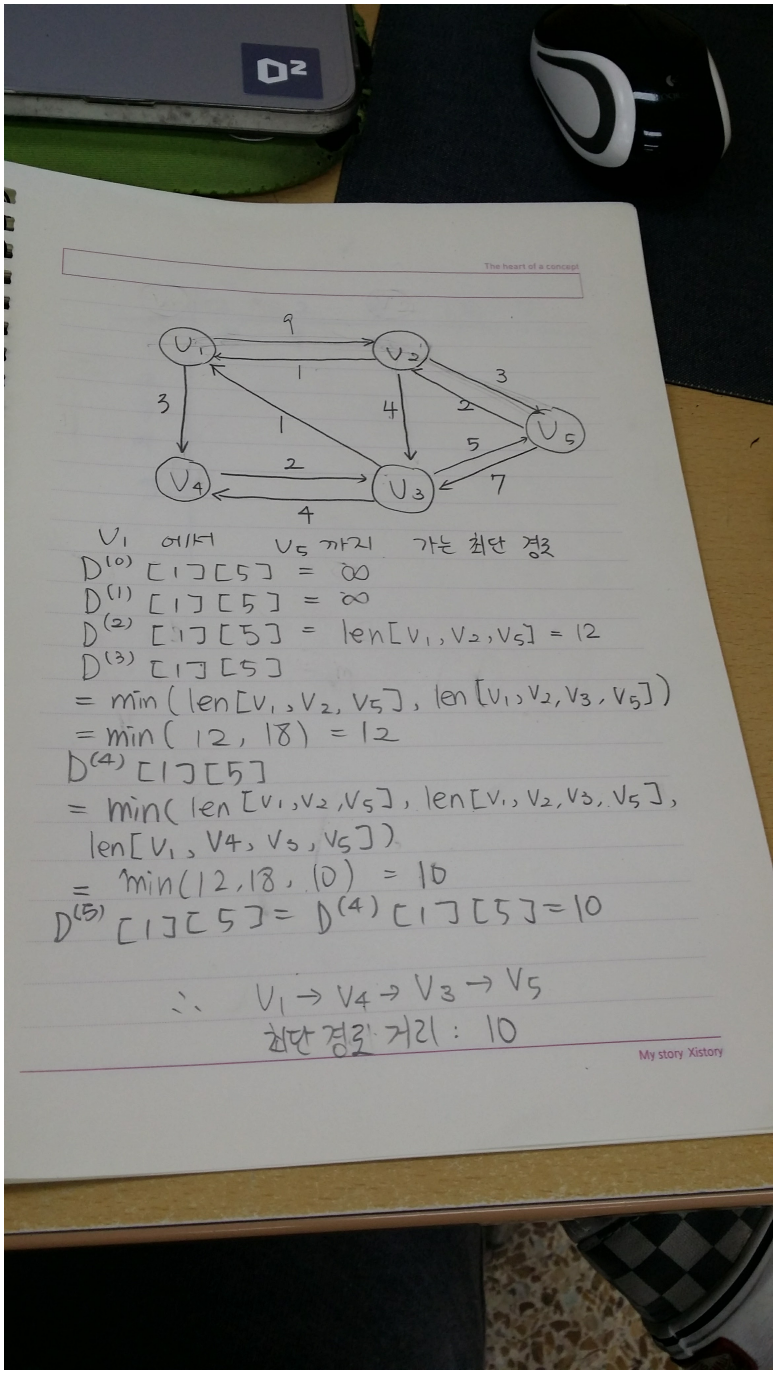
3. 실행 결과

```
hongsang-won-ui-MacBook-Pro:ch03 Frodo$ ./floyd
v1에서 v2로 가는 최단 경로의 거리 : 1
최단경로 : v1 v2

v1에서 v3로 가는 최단 경로의 거리 : 3
최단경로 : v1 v4 v3

v1에서 v4로 가는 최단 경로의 거리 : 1
최단경로 : v1 v4

v1에서 v5로 가는 최단 경로의 거리 : 4
최단경로 : v1 v4 v5

v2에서 v1로 가는 최단 경로의 거리 : 8
최단경로 : v2 v4 v5 v1

v2에서 v3로 가는 최단 경로의 거리 : 3
최단경로 : v2 v3

v2에서 v4로 가는 최단 경로의 거리 : 2
최단경로 : v2 v4

v2에서 v5로 가는 최단 경로의 거리 : 5
최단경로 : v2 v4 v5

v3에서 v1로 가는 최단 경로의 거리 : 10
최단경로 : v3 v4 v5 v1

v3에서 v2로 가는 최단 경로의 거리 : 11
최단경로 : v3 v4 v5 v1 v2

v3에서 v4로 가는 최단 경로의 거리 : 4
최단경로 : v3 v4

v3에서 v5로 가는 최단 경로의 거리 : 7
최단경로 : v3 v4 v5

v4에서 v1로 가는 최단 경로의 거리 : 6
최단경로 : v4 v5 v1

v4에서 v2로 가는 최단 경로의 거리 : 7
최단경로 : v4 v5 v1 v2

v4에서 v3로 가는 최단 경로의 거리 : 2
최단경로 : v4 v3

v4에서 v5로 가는 최단 경로의 거리 : 3
최단경로 : v4 v5

v5에서 v1로 가는 최단 경로의 거리 : 3
최단경로 : v5 v1

v5에서 v2로 가는 최단 경로의 거리 : 4
최단경로 : v5 v1 v2
```

```
v5에서 v3로 가는 최단 경로의 거리 : 6
최단경로 : v5 v1 v4 v3

v5에서 v4로 가는 최단 경로의 거리 : 4
최단경로 : v5 v1 v4
```

4. 다른 그래프 입력

1) 다른 그래프와 손 계산

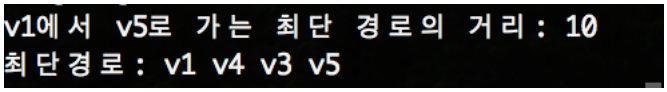


2) 새로운 그래프로 인접행렬 수정

[Floyd_handy.c]

```
int main(void) {
	/*
	*  directed, weighted graph 의 인접행렬
	*  W[i][j]
	*  - vi 에서 vj 로 가는 edge 의 가중치 값
	*  - v1 에서 vj 로 가는 edge 가 없으면 무한대(INFINITY) 대입
	*  - i == j 이면 0 대입
	*/
	//vertex 를 1 부터 시작하기 위해 배열 index 가 0 인 부분은 사용하지 않음(사용하는 않는 공간은 -1 대입)
	const number W[N + 1][N + 1] = {

		{-1, -1, -1, -1, -1, -1},
		{-1, 0, 9, INFINITY, 3, INFINITY},
		{-1, 1, 0, 4, INFINITY, 3},
		{-1, 1, INFINITY, 0, 4, 5},
		{-1, INFINITY, INFINITY, 2, 0, INFINITY},
		{-1, INFINITY, 2, 7, INFINITY, 0}
	};

	number D[N + 1][N + 1] = {-1};
	index P[N + 1][N + 1] = {-1};

	floyd(N, W, D, P);

	// 최소 가중치 출력
	printf("v%d 에서 v%d 로 가는 최단 경로의 거리: %d₩n", 1, 5, D[1][5]);

	//최단경로로 갈 때 방문하는 모든 vertex 출력
	printf("최단경로: v%d ", 1);
	if(P[1][5] != 0)
		path(1, 5, P);
	printf("v%d₩n", 5);

	return 0;
}
```

3) 컴퓨터 실행 결과



- 직접 계산한 결과와 컴퓨터 실행 결과가 같다.