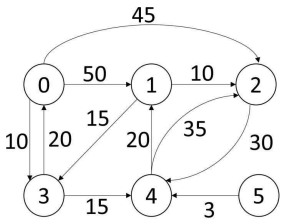
6.5 Shortest paths

A부터 B로 가는 path가 있는가? 있다면 A부터 B로 가는 shortest path는 무엇인가? 최단 경로 문제는 edge의 weight 또는 cost가 있는 그래프에 대하여 적용한다. 최단 경로의 출발점을 source, 도착점을 destination이라 한다.

6.5.1 Single source/all destinations: nonnegative edges costs

주어진 directed graph G = (V, E)에 대하여, length function인 length(i, j)이 있고 length(i,j) >= 0이 edge의 이동 소요 시간이라 하자. 그림 6.10처럼 출발점 v로 부터 G의 나머지 vertices로 가는 shortest path를 구하는 문제[1]이다.



(a) Graph

	Path	*Length*
1)	0, 3	10
2)	0, 3, 4	25
3)	0, 3, 4, 1	45
4)	0, 2	45

(b) 시작노드0으로 부터 최단 경로

그림 6.10 그래프 최단 경로.

S를 이미 찾은 shortest paths의 vertices 집합이라 하자. v를 source vertex(출발점)이라고 할 때 S = {v}이다. dist[w]를 v로 부터 시작하여 S에 있는 *vertices*만을 경유하여 w로 가는 shortest path의 length라 하자. w는 S에 포함되지 않은 vertex이다. 최단 경로를 구하는 greedy algorithm은 path length가 증가하는 순서로 shortest paths를 구한다. path length는 source로 부터 시작하여 경유 vertices를 통과하여 w까지 도달하는데 경유하는 edge들의 length의 합이다. greedy 알고리즘으로 path length 순서로 최단 경로를 구한다.

1) next shortest path가 v로 부터 u일 때 최단 경로는 S에 있는 vertices만을 경유해야 한다. v에서 u로 가는 최단 경로의 vertex는 S에 없는 vertex를 경유하지 않는다.
2) 다음에 생성되는 next path의 destination의 선택은 S에 있지 않는 vertices 중에서 minimum distance, dist[u]인 u를 선택한다.
3) u가 선택되어 S에 포함되면 S에 있지 않은 vertex w에 대하여 dist[w]를 다시 계산한다. dist[w]를 다시 계산할 때 S에 있는 vertex를 경유한 최단 경로의 dist[w]를 계산한다. dist[w]와 dist[u] + length(<u,w>)(v부터 시작하여 u를 경유하여 w로 가는 비용) 중에서 최솟값을 dist[w]로 계산한다.

```
 void Graph::ShortestPath(const int n, const int v) //[1]에서 인용
//dist[j], 0<= j < n 은 v로부터 j로 가는 최단 경로의 길이
// 방향성 G는 n 개의 vertex를 갖고 edge 길이는 length[i][j]로 표기
{
    for (int i = 0; i < n; i++) {
      s[i] = false;
      dist[i] = length[v][i];
  }
  s[v] = true;
  dist[v] = 0;

 for (int i = 0; i < n-2; i++) {//v로 부터 갈 수 있는 n-1 경로를 결정
     int u = choose(n); //다음 조건을 만족하는 u를 선택
 //dist[u] = minimum dist[w], where sw[w] = false;
s[u] = true;
for (int w = 0; w < n; w++)
    if (!s[w])
        if (dist[u] + length[u][w] < dist[w])
            dist[w] = dist[u] + length[u][w];
    }
}
```

ShortestPath 알고리즘은 Edsger Dijkstra(pronounced[dɛikstrɑ])[1]가 고안한 알고리즘으로 greedy한 이유는 각 loop에서 모든 경우를 check하여 매 선택 시마다 가장 최선의 선택을 하기 때문에 붙인 이름이다. S 집합은 array s[i]로 표현하며, vertex i가 S에 있으면 s[i] = true이다. graph는 length-adjacency matrix, length[i][j]로 표현한다. graph 문제를 adjacency list로 표현하여 알고리즘을 구현하는 대표적 응용이 minimal cost spanning tree이다. 반면에 graph를 matrix로 표현하여 알고리즘 문제를 푸는 전형적인 사례가 최단경로 문제이다.

```
class Graph {
private:
      int length[nmax][nmax];
      int dist[nmax];
      bool s[nmax];
public:
      void ShortestPath(const int, const int);
      int choose(const int);
};
```

그림 6.11에서 dist[6] = ∞이고(값이 없으면 경로가 없으므로 무한대로 간주) 출발점 4 → 5 → 6로 이동하는 경로는 1150이다. 따라서 5가 선택되어 S에 포함되면 dist[6] = min(<4,6>, <4,5><5,6>) = 1150이 된다.

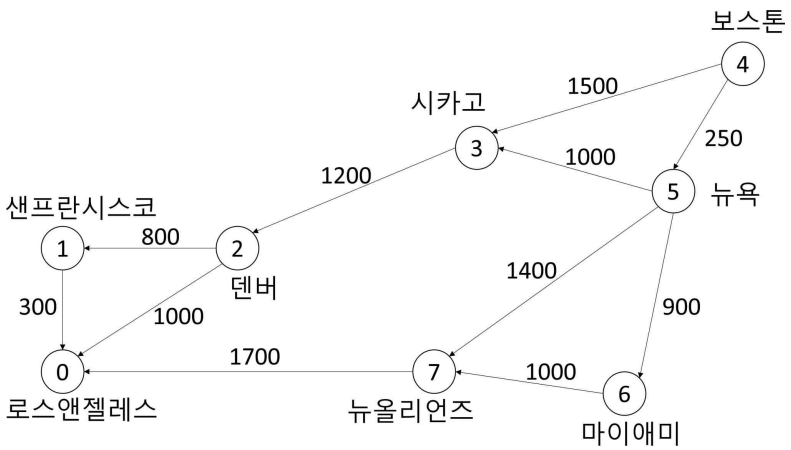
```
//소스 코드 6.4: Shortest path
import java.util.*;
public class Graph {
    private static final int INF = Integer.MAX_VALUE / 2; // Prevent overflow
    private final int[][] length;
    private final int[][] a;
    private final int[] dist;
    private final int[] newdist;
    private final boolean[] s;
    private final int n;
    public Graph(int n) {
        this.n = n;
        length = new int[n][n];
        a = new int[n][n];
        dist = new int[n];
        newdist = new int[n];
        s = new boolean[n];
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                length[i][j] = INF;
    }
    public void insertEdge(int start, int end, int weight) {
        if (start < 0 || end < 0 || start >= n || end >= n) {
            System.out.println("Invalid edge index.");
            return;
        }
        length[start][end] = weight;
    }
    public void displayConnectionMatrix() {
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                if (length[i][j] != INF)
                    System.out.print(i + " -> " + j + " (" + length[i][j] + "), ");
            }
            System.out.println();
        }
    }
    public void dijkstra(int v) {
        for (int i = 0; i < n; i++) {
            dist[i] = length[v][i];
            s[i] = false;
        }
        dist[v] = 0;
        s[v] = true;
```

```
        for (int i = 0; i < n - 1; i++) {
            int u = choose();
            s[u] = true;
            for (int w = 0; w < n; w++) {
                if (!s[w] && dist[u] + length[u][w] < dist[w]) {
                    dist[w] = dist[u] + length[u][w];
                }
            }
        }
        printDistances(v);
    }
    public void bellmanFord(int v) {
        for (int i = 0; i < n; i++)
            dist[i] = INF;
        dist[v] = 0;
        for (int k = 1; k < n; k++) {
            for (int u = 0; u < n; u++) {
                for (int i = 0; i < n; i++) {
                    if (length[i][u] < INF && dist[i] < INF && dist[u] > dist[i] +
length[i][u]) {
                        dist[u] = dist[i] + length[i][u];
                    }
                }
            }
        }
        printDistances(v);
    }
    public void AllLengths() {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                a[i][j] = length[i][j];
        for (int k = 0; k < n; k++) {
            for (int i = 0; i < n; i++) {
                for (int j = 0; j < n; j++) {
                    if (a[i][k] + a[k][j] < a[i][j]) {
                        a[i][j] = a[i][k] + a[k][j];
                    }
                }
            }
        }
        for (int i = 0; i < n; i++) {
            System.out.print("From " + i + ": ");
            for (int j = 0; j < n; j++) {
                System.out.print((a[i][j] >= INF ? "INF" : a[i][j]) + " ");
```

```
            }
            System.out.println();
        }
    }
    private int choose() {
        int min = INF, index = -1;
        for (int i = 0; i < n; i++) {
            if (!s[i] && dist[i] < min) {
                min = dist[i];
                index = i;
            }
        }
        return index;
    }
    private void printDistances(int start) {
        System.out.println("Distances from node " + start + ":");
        for (int i = 0; i < n; i++) {
            System.out.print((dist[i] >= INF ? "INF" : dist[i]) + " ");
        }
        System.out.println();
    }
    public boolean hasNegativeEdge() {
        for (int i = 0; i < n; i++)
            for (int j = 0; j < n; j++)
                if (length[i][j] < 0) return true;
        return false;
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Graph g = null;
        System.out.println("1: custom, 2: default1, 3: default2, 4: allPairs");
        int opt = sc.nextInt();
        if (opt == 1) {
            System.out.print("Enter node count: ");
            g = new Graph(sc.nextInt());
        } else if (opt == 2) {
            g = new Graph(8);
            g.insertEdge(1, 0, 300);
            g.insertEdge(2, 1, 800);
            g.insertEdge(2, 0, 1000);
            g.insertEdge(3, 2, 1200);
            g.insertEdge(4, 3, 1500);
            g.insertEdge(4, 5, 250);
            g.insertEdge(5, 3, 1000);
```

```
            g.insertEdge(5, 7, 1400);
            g.insertEdge(5, 6, 900);
            g.insertEdge(6, 7, 1000);
            g.insertEdge(7, 0, 1700);
        } else if (opt == 3) {
            g = new Graph(7);
            g.insertEdge(0, 1, 6);
            g.insertEdge(0, 2, 5);
            g.insertEdge(0, 3, 5);
            g.insertEdge(1, 4, -1);
            g.insertEdge(2, 1, -2);
            g.insertEdge(2, 4, 1);
            g.insertEdge(3, 2, -2);
            g.insertEdge(3, 5, -1);
            g.insertEdge(4, 6, 3);
            g.insertEdge(5, 6, 3);
        } else if (opt == 4) {
            g = new Graph(3);
            g.insertEdge(0, 1, 4);
            g.insertEdge(0, 2, 11);
            g.insertEdge(1, 0, 6);
            g.insertEdge(1, 2, 2);
            g.insertEdge(2, 0, 3);
        }
        int select;
        do {
            System.out.println("\n1: AddEdge, 2: Display, 3: Dijkstra, 4:
Bellman-Ford, 5: Floyd-Warshall, 6: Quit");
            select = sc.nextInt();
            switch (select) {
                case 1:
                    System.out.print("Start: ");
                    int u = sc.nextInt();
                    System.out.print("End: ");
                    int v = sc.nextInt();
                    System.out.print("Weight: ");
                    int w = sc.nextInt();
                    g.insertEdge(u, v, w);
                    break;
                case 2:
                    g.displayConnectionMatrix();
                    break;
                case 3:
                    if (g.hasNegativeEdge()) {
```

```
                        System.out.println("Negative edge found. Cannot use
Dijkstra.");
                        break;
                    }
                    System.out.print("Start node: ");
                    g.dijkstra(sc.nextInt());
                    break;
                case 4:
                    System.out.print("Start node: ");
                    g.bellmanFord(sc.nextInt());
                    break;
                case 5:
                    g.AllLengths();
                    break;
                case 6:
                    break;
                default:
                    System.out.println("Invalid option");
            }
        } while (select != 6);
    }
}
```



(a) Diagraph

	0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0	0
1	300	0	0	0	0	0	0	0
2	1000	800	0	0	0	0	0	0
3	0	0	1200	0	0	0	0	0
4	0	0	0	1500	0	250	0	0
5	0	0	0	1000	0	0	900	1400
6	0	0	0	0	0	0	0	1000
7	1700	0	0	0	0	0	0	0

(b) Length – adjacency matrix

그림 6.11 미국 도시들의 최단경로 예제.

Iteration	S	Vertex Selected	Distance							
			LA	SF	DEN	CHI	BOST	NY	MIA	NO
			[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]
Initial	--	---	+∞	+∞	+∞	1500	0	250	+∞	+∞
1	{4}	5	+∞	+∞	+∞	1250	0	250	1150	1650
2	{4,5}	6	+∞	+∞	+∞	1250	0	250	1150	1650
3	{4,5,6}	3	+∞	+∞	2450	1250	0	250	1150	1650
4	{4,5,6,3}	7	3350	+∞	2450	1250	0	250	1150	1650
5	{4,5,6,3,7}	2	3350	3250	2450	1250	0	250	1150	1650
6	{4,5,6,3,7,2} {4,5,6,3,7,2,1}	1	3350	3250	2450	1250	0	250	1150	1650

그림 6.12 최단 경로 알고리즘에 의한 미국 도시 최단 경로 처리 결과.

6.5.2 Single source/all destinations: general weights

어떤 edge가 negative length을 갖는 경우에 그림 6.12와 같이 동작하는 Edsger Dijkstra의 ShortestPath(greedy algorithm)은 correct results를 주지 못한다. 그 이유는 greedy- method이기 때문이다. 그림 6.13처럼 negative-length edge를 갖는 directed graph에서 출발점 0에 대한 1,2번 vertex의 dist[1] = 7, dist[2] = 5가 된다[1]. dist[u]는 source vertex v로 부터 u로 가는 length이다. 그리고 set S = {0,1,2}가 된다. 그런데 0 → 2로 가는 shortest path는 0,1,2이며 dist[2] = 2가 된다.

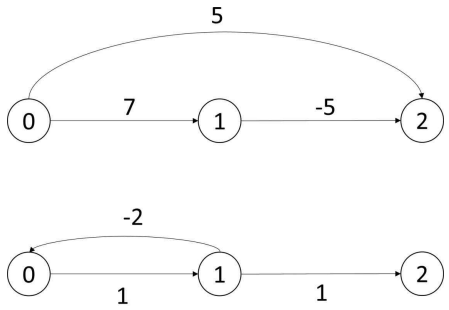


그림 6.13 Negative length를 갖는 edge가 있는 경우.

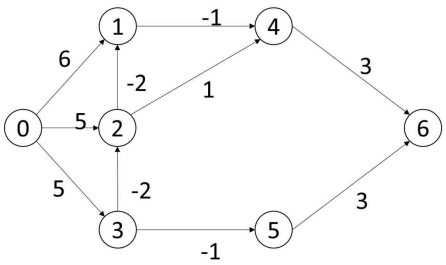
그래프 G에 negative length의 cycle 이 없다고 가정한다. 그 이유는 shortest paths는

유한개의 edges로 구성되어야 하기 때문이다. n-vertex를 갖는 graph에서 2개의 vertics 간의 shortest path는 많아야 n-1 edges를 갖는다.

$dist^l[u]$를 v → u로 가는 shortest path의 length라 하자. 이때 shortest path는 많아야 l개의 edges를 포함한다. $dist^1[u]$ = length[v][u]는 1개의 edge를 사용하는 경우이다. 많아야 k edges를 포함할 수 있는 v → u 경로의 shortest path가 k-1 edges만 포함하는 경우에는 $dist^k[u] = dist^{k-1}[u]$가 된다. 많아야 k edges를 포함할 수 있는 v → u 경로의 shortest path가 k edges를 포함하는 경우에는 "최대 k-1 edges를 포함하는 v→j 경로의 shortest path" + edge <j, u>이다. v → j 경로의 path는 k-1 edges를 포함하며 path length는 $dist^{k-1}[j]$이다. $dist^{k-1}[i]$ + length[i][u]를 최소화하는 i를 찾고 $dist^{k-1}[j]$와 비교하여 min을 선택한다.

$dist^2[2]$=min{$dist^1[2]$, min{$dist^1[I]$+length[I][2]}, $dist^1[3]$+length[3][2]}로 계산되며 min{5, 5+-2}이므로 = 3이 된다. void Graph::BellmanFord2(const int v)[1]은 single source - all destinations[1]에 대한 최단 경로를 그림 6.14처럼 구한다.

$$dist^k[u] = \min_i\{dist^{k-1}[u], \min\{dist^{k-1}[i] + length[i][u]\}\}$$



k	$dist^k$ [7]						
	0	1	2	3	4	5	6
1	0	6	5	5	∞	∞	∞
2	0	3	3	5	5	4	∞
3	0	1	3	5	2	4	7
4	0	1	3	5	0	4	5
5	0	1	3	5	0	4	3
6	0	1	3	5	0	4	3

(a) A directed graph (b) $dist^k$

그림 6.14 BellmanFord 알고리즘 처리 결과.

```
void Graph::BellmanFord(const int n, const int v)  //[1]에서 인용
{
//negative edge 길이를 갖는 single source all destination 최단 경로
{
   for (int i = 0; i < n; i++)
      dist[i] = length[v][i];
   for (int k = 2; k <= n-1; k++)
      for(each u such that u != v and u has at least one incoming edge)
         for (each <i,u> in the graph)
            if (dist[u] > dist[i] + length[i][u])
               dist[u] = dist[i] + length[i][u];
}
```

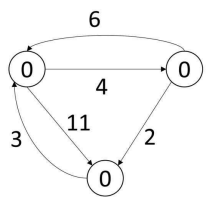
6.5.3 All-pairs shortest paths

vertices u와 v의 모든 쌍에 대한 shortest paths를 구하는 알고리즘이 all-pairs shortest path이다. G의 n vertices를 출발점으로 간주하여 single-source/all-destinations problems를 n번 반복 실행으로 구할 수 있다. 이 경우엔 time complexity는 $O(n^4)$이 되어 비효율적이다.

$A^k[i][j]$를 i → j로 가는 경로의 shortest path의 length라 하고 k보다 큰 vertex index를 중간 경유하지 않는다. $A^{-1}[i][j]$ = length[i][j], i→j paths는 어떠한 중간 경우 노드가 없다. all-pairs shortest path 알고리즘은 A^{-1}, A^0, A^1, ..., A^{n-1}을 연속적으로 계산하여 나가는 방식이다. A^{k-1}을 먼저 구했으면 $A^k[i][j]$가 다음 2가지 경우에 대하여 계산된다.

1) i→j로 가는 경로의 shortest path가 k보다 큰 vertex를 경유하지 않는 경우의 length는 vertex k를 경유하지 않는 length와 같다. 따라서 $A^k[i][j] = A^{k-1}[i][j]$이다.
2) i→j로 가는 경로의 shortest path가 vertex k를 경유하는 경우로서 i→k로 가는 경로의 length + k→j로 가는 경로의 length이다. $A^k[i][j] = A^{k-1}[i][k] + A^{k-1}[k][j]$이다.

$$A^k[i][j] = \min\{A^{k-1}[i][j], A^{k-1}[i][k] + A^{k-1}[k][j], k \geq 0$$



(a) 방향성 그래프

$A^{-1}[i][j] = length\,[i][j]$

A^{-1}	0	1	2
0	0	4	11
1	6	0	2
2	3	∞	0

(b) A^{-1}

A^0	0	1	2
0	0	4	11
1	6	0	2
2	3	7	0

(c) A^0

A^1	0	1	2
0	0	4	6
1	6	0	2
2	3	7	0

(d) A^1

A^2	0	1	2
0	0	4	6
1	5	0	2
2	3	7	0

(e) A^2

그림 6.15 All-pairs 최단 경로.

그림 6.15처럼 $A^{-1}[2][1]$은 2 → 1로 경로에 다른 vertex를 경유하지 않는다. $A^0[2][1]$은 2 → 1로 가는 경로에 vertex 0을 경유하는 것을 말한다. $A^0[2][1] = \min\{A^{-1}[2][1], A^{-1}[2][0] + A^{-1}[0][1]\} = \min\{\infty, 3 + 4\} = 7$이 된다. void Graph::AllLengths( )에 의한 all-pairs shortest path를 구한다[1].

greedy 알고리즘은 매번 선택할 때마다 best를 선택(차선책을 선택하는 양보 없이 항상 best만 선택하므로 greedy하다고 보는 것이다)하는 방식이다. greedy-method 로 풀 수 있는 activity selection problem(또는 interval scheduling maximization problem)은 주어진 시간대에서 특정 시간에 하나의 activity만 처리할 수 있는 공간에서 최대 많은 activity를 처리할 수 있는 해를 구하는 것이다. 각 activity는 시작 시간 s_i와 종료 시간 f_i가 있다. 두개의 activity a_i와 a_j가 시간적으로 겹치지 않으면 선택될 수 있다. fractional knapsack problem은 가치의 합이 최대가 되도록 배낭에 넣을 짐을 고르는 최적 해를 구한다. v_i를 가치, w_i를 무게이고 배낭의 최대 무게 제한이 있을 때 최적의 무게 대비 가치를

넣을 수 있는 최적 해를 구하는 문제이다. 무게가 초과될 경우에는 쪼개서 넣을 수 있다. 무게가 초과되어 쪼갤 수 있는 경우를 fractional knapsack problem이라 하고, 짐을 쪼갤 수 없는 경우를 0-1 knapsack problem이라 한다. 고수준의 알고리즘 코딩 시험 문제로 knapsack 알고리즘을 코딩하는 실습이 유익하다.

그림 6.16에 대하여 Edsger Dijkstra의 ShortestPath(greedy algorithm)을 사용하여 출발점을 0으로 하여 모든 다른 노드에 대한 최단 경로를 구한다. 그리고 다른 응용 예제로서 edge의 weight가 -2일 때 greedy 알고리즘이 동작하는지 그리고 void Graph::BellmanFord2(const int v)와 void Graph::AllLengths( )의 실행 결과를 확인한다.

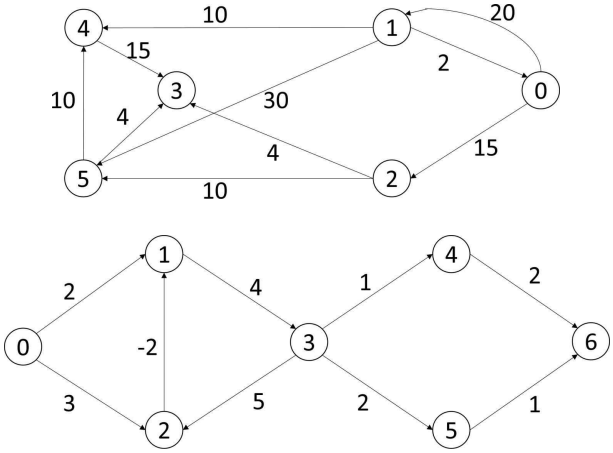


그림 6.16 최단 경로 알고리즘 적용 실습 예제.

```
void Graph::AllLength(const int n) //[1]에서 인용
//length[n][n]은 n개의 vertex를 갖는 그래프의 adjacency matrix
//a[i][j]는 i와 j 간의 최단경로의 길이
{
    for (int i = 0; i < n; i++)
        for (int j = 0; j < n; j++)
            a[i][j] = length[i][j];//adjacency matrix를 a로 복사
    for (int k = 0; k < n; k++) //가장 큰 vertex index k를 갖는 경로
        for (i = 0; i < n; i++) //vertex들의 모든 가능한 pairs에 대하여
            for (int j = 0; j < n; j++)
                if ((a[i][k] + a[k][j]) < a[i][j])
                    a[i][j] = a[i][k] + a[k][j];
}
```