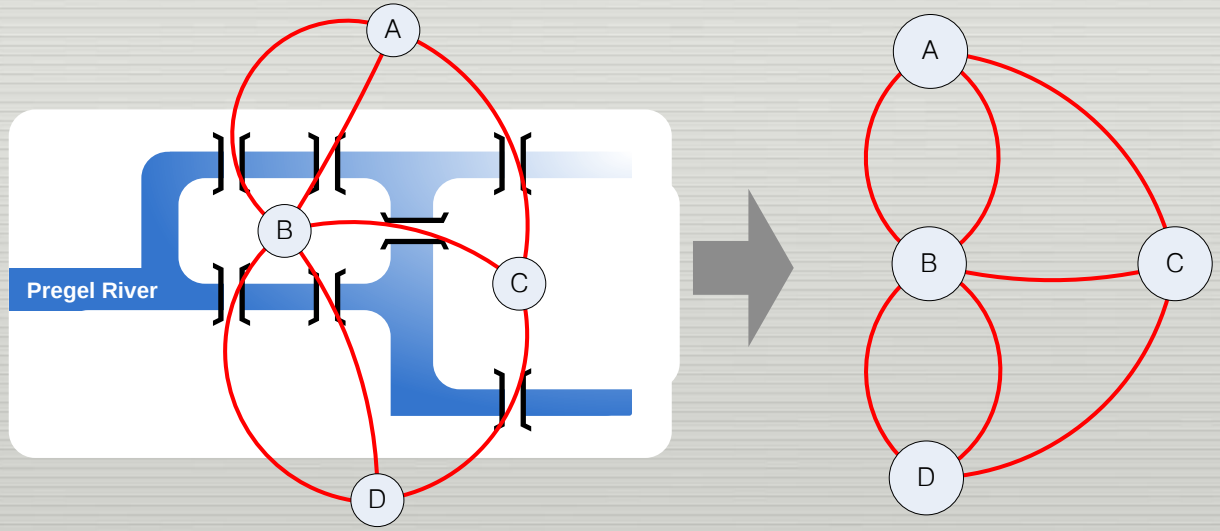
프로그래밍을 이용한 문제 해결
4일차 – 그래프, 탐욕 기법

강영민
동명대학교 게임공학과

창의적 소프트웨어 융합 전문 인력 양성 사업단
2016년 1월 – 소프트웨어 역량 강화 프로그램

그래프를 소개합니다

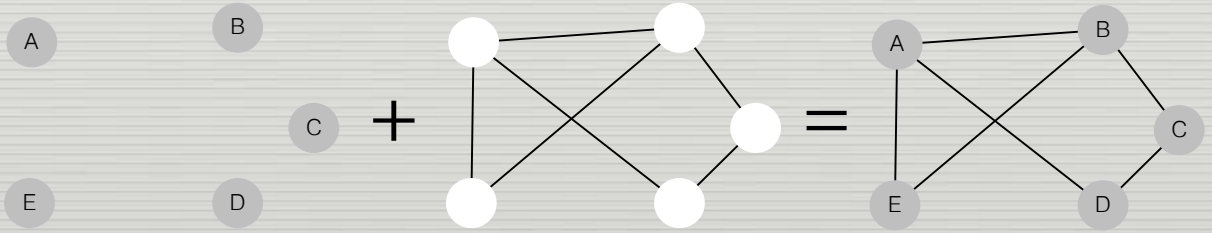
- 라인하르트 오일러가 "쾨니히스베르크의 7개의 다리 문제"를 풀기 위해 고안해 낸 수학적 도구.
  - 7개의 다리를 간선(Edge)로, 4개의 육지를 정점(Vertex)로 표현.



그래프를 소개합니다

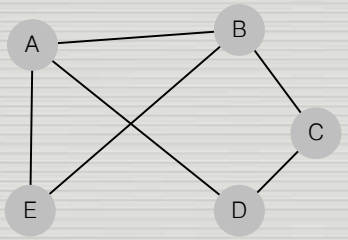
- 그래프는 "정점의 모음"과 이 정점을 잇는 "간선의 모음" 간의 결합

  "정점의 집합을 V, 간선의 집합을 E, 그리고 그래프를 G라고 했을 때,
  $G = (V, E)$ *이다."*



그래프를 소개합니다

- 인접(Adjacent)
  - 간선으로 연결되어 있는 두 정점을 일컫는 말. “이웃 관계”에 있다고 표현하기도 한다.

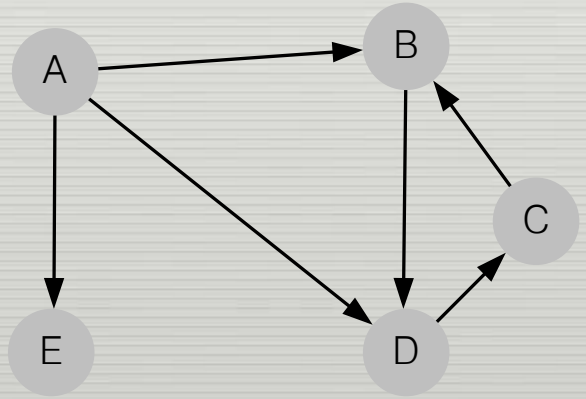


(A, B), (A, D), (A, E), (B, C), (B, E), (C, D)가 서로 이웃 관계

- 경로(Path)
  - 위 그래프에서 정점 A에서 정점 C까지의 경로는 A-B-C 와 A-D-C, A-E-B-C 등이 있다. A-B-C, A-D-C 경로의 길이는 2, A-E-B-C의 길이는 3이다.

그래프를 소개합니다

- 간선이 방향성을 가지면 그래프는 다음과 같은 방향성 그래프(Directed Graph)가 되고, 반대로 간선에 방향성이 없으면 무방향성 그래프(Undirected Graph)



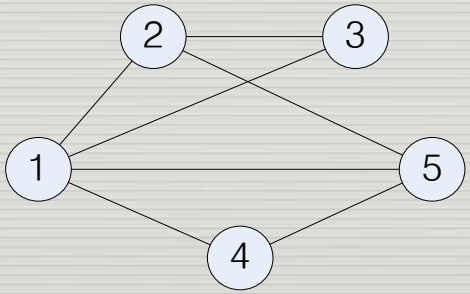
그래프를 어떻게 표현할 것인가?

- 그래프는 정점의 집합과 간선의 집합의 결합
  - → 그래프의 표현 문제는 "간선, 즉 정점과 정점의 인접 관계를 어떻게 나타내는가?"의 문제로 귀결

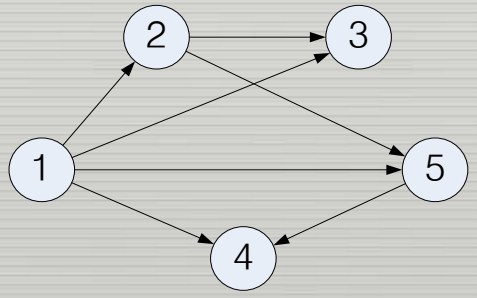
- 행렬을 이용하는 방식
  - 인접 행렬(Adjacency Matrix)
- 리스트를 이용하는 방식
  - 인접 리스트(Adjacency List)

그래프: 인접 행렬 표현

- 인접 행렬 : 정점끼리의 인접 관계를 나타내는 행렬
  - 그래프의 정점의 수를 N이라고 한다면, 크기의 행렬을 만들어 행렬의 각 원소를 한 정점과 또 다른 정점이 인접해 있는 경우(즉, 정점 사이에 간선이 존재하는 경우)에는 1로 표시하고, 인접해 있지 않은 경우에는 0으로 표시



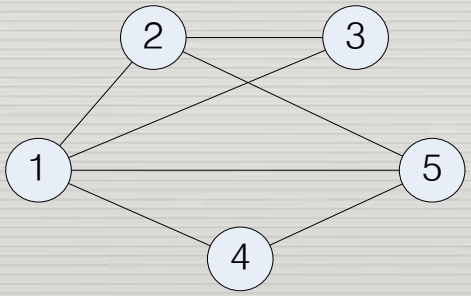
	1	2	3	4	5
1	0	1	1	1	1
2	1	0	1	0	1
3	1	1	0	0	0
4	1	0	0	0	1
5	1	1	0	1	0



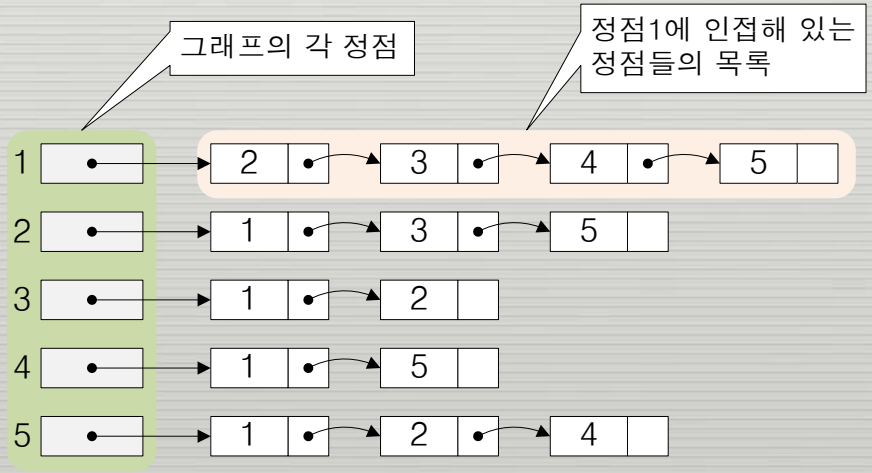
	1	2	3	4	5
1	0	0	0	0	0
2	1	0	0	0	0
3	1	1	0	0	0
4	1	0	0	0	1
5	1	1	0	0	0

그래프: 연결 리스트 표현

- 그래프 내의 각 정점의 인접 관계를 표현하는 리스트

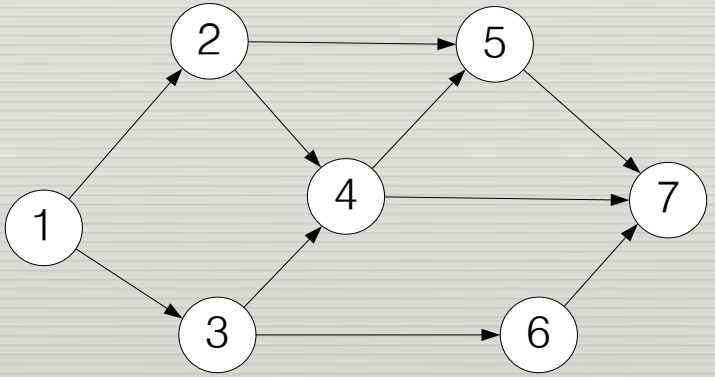


정점	인접 정점
1	2, 3, 4, 5
2	1, 3, 5
3	1, 2
4	1, 5
5	1, 2, 4



그래프 순회: 그래프를 따라 산책

- 어떻게 하면 아래의 그래프에서 모든 정점을 방문할 수 있을까?



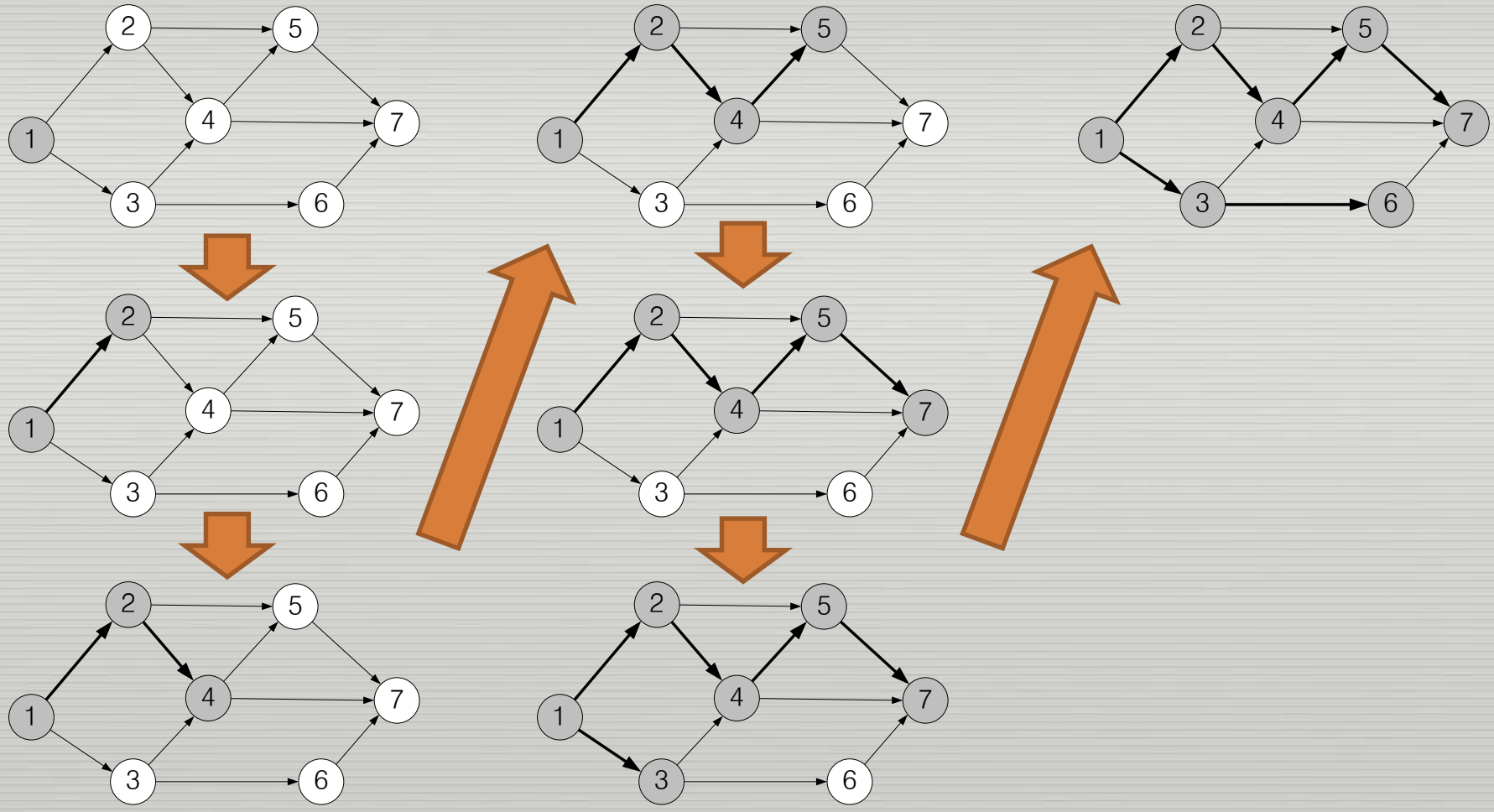
그래프 순회

- 깊이 우선 탐색(Depth First Search)
  - "더 나아갈 길이 보이지 않을 때까지 깊이 들어간다."

1) 시작 정점을 밟은 후 이 정점을 "방문했음"으로 표시합니다.
2) 그리고 이 정점과 이웃하고 있는 정점(즉, 인접 정점) 중에 아직 방문하지 않은 곳을 선택하여 이를 시작 정점으로 삼아 다시 깊이 우선 탐색을 시작합니다. 그러니까 단계 1)을 다시 하는 겁니다.
3) 정점에 더 이상 방문하지 않은 인접 정점이 없으면 이전 정점으로 돌아가서 단계 2)를 수행합니다.
4) 이전 정점으로 돌아가도 더 이상 방문할 이웃이 없다면 그래프의 모든 정점을 모두 방문했다는 뜻입니다. 이제 탐색을 종료합니다.

그래프 순회

- 깊이 우선 탐색의 예



그래프 순회

- 너비 우선 탐색(Breadth First Search)
  - "꼼꼼하게 좌우를 살피며 다니자."

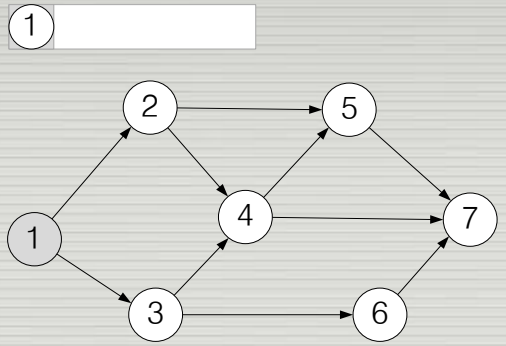
1) 시작 정점을 "방문했음"으로 표시하고 큐에 삽입(Enqueue)합니다.

2) 큐로부터 정점을 제거(Dequeue)합니다. 제거한 정점이 이웃하고있는 인접 정점 중 아직 방문하지 않은 곳들을 "방문했음"으로 표시하고 큐에 삽입합니다.

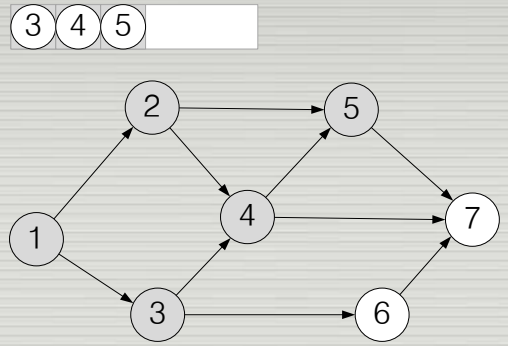
3) 큐가 비게 되면 탐색이 끝난 것입니다. 따라서 큐가 빌 때까지 2의 과정을 반복합니다.

그래프 순회: 너비 우선 탐색

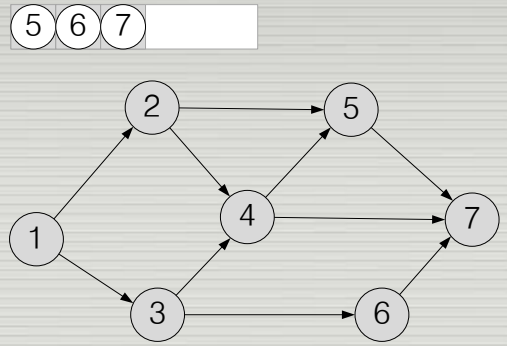
Queue



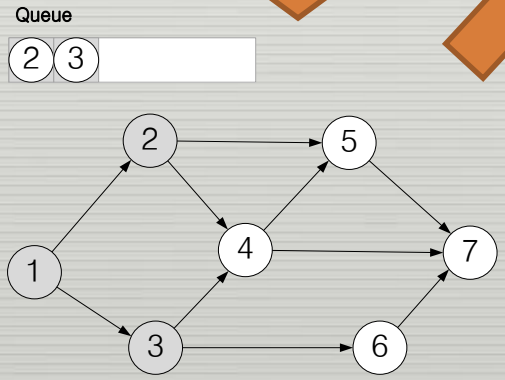
Queue



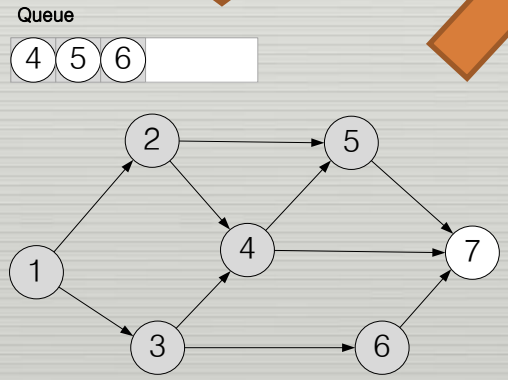
Queue



Queue

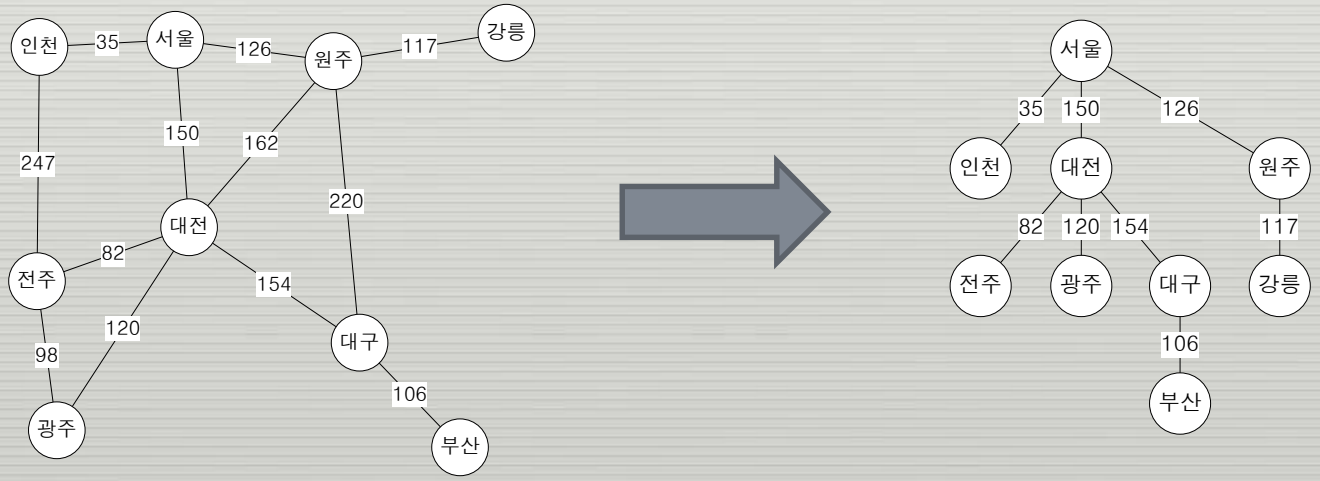


Queue



최소 신장 트리

- 최소 신장 트리
  - 간선에 "가중치(Weight)" 속성을 부여
  - 각 간선이 갖고 있는 가중치의 합이 최소가 되는 신장 트리



최소 신장 트리

- 프림 알고리즘(Prim's Algorithm)

1) 그래프와 최소 신장 트리를 준비합니다. 물론 이 때의 최소 신장 트리는 노드가 하나도 없는 상태입니다.

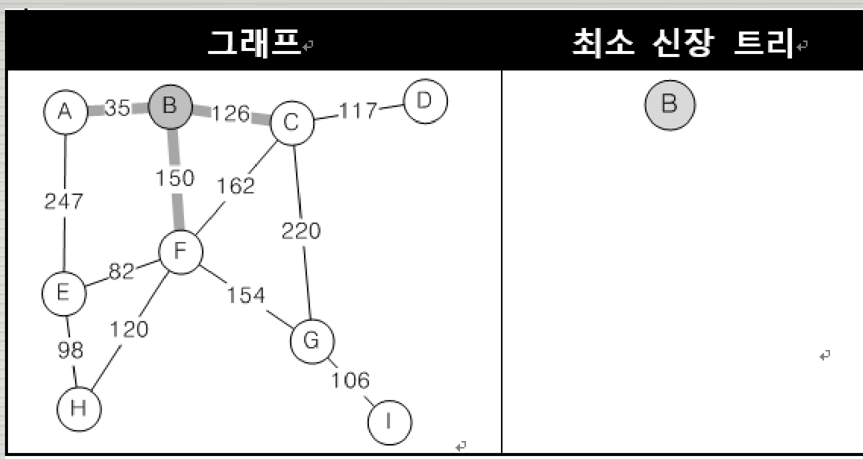
2) 그래프에서 임의의 정점을 시작 정점으로 선택하여 최소 신장 트리의 루트 노드로 삽입합니다.

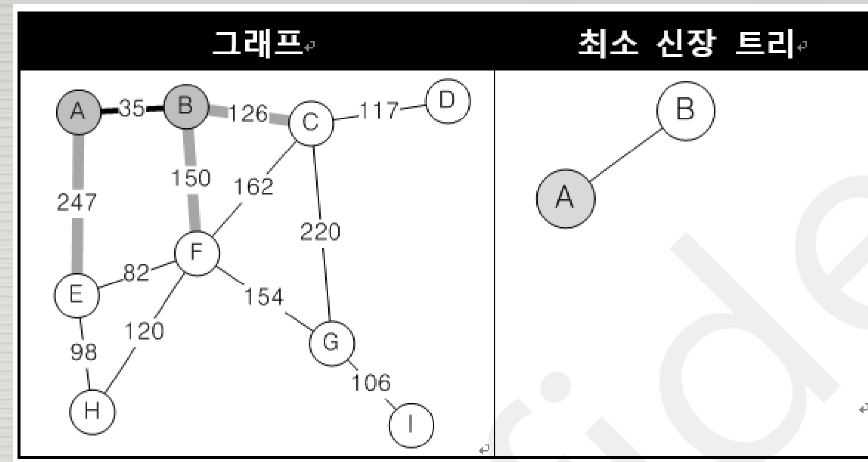
3) 최소 신장 트리에 삽입되어 있는 정점들과 이 정점들의 모든 인접 정점 사이에 있는 간선의 가중치를 조사합니다. 간선 중에 가장 가중치가 작은 것을 골라 이 간선에 연결되어 있는 인접 정점을 최소 신장 트리에 삽입합니다. 단, 새로 삽입되는 정점은 최소 신장 트리에 삽입되어 있는 기존의 노드들과 사이클을 형성해서는 안됩니다.

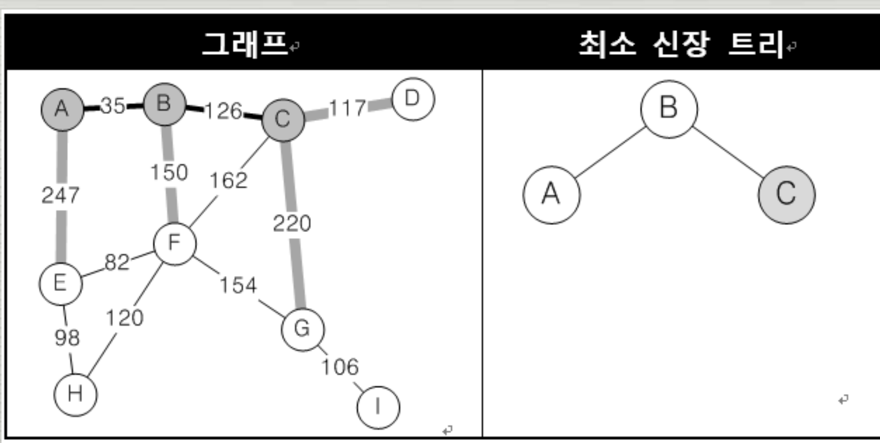
4) 3)의 과정을 반복하다가 최소 신장 트리가 그래프의 모든 정점을 연결하게 되면 알고리즘을 종료합니다.

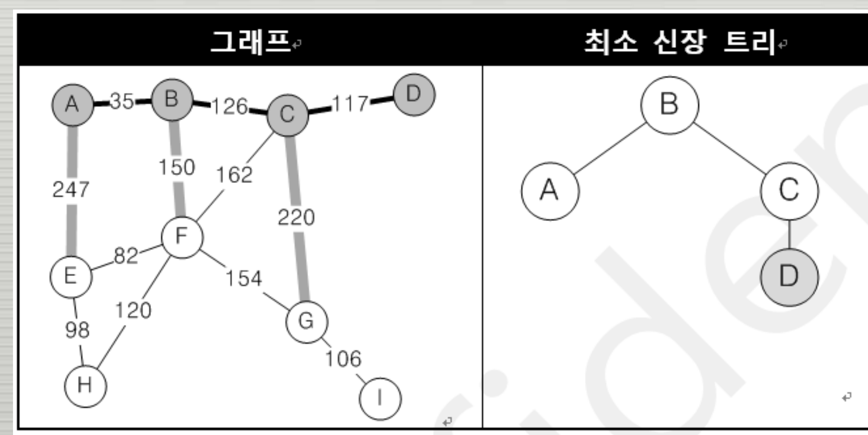
최소 신장 트리: 프림

- 프림 알고리즘의 예 (1~4)

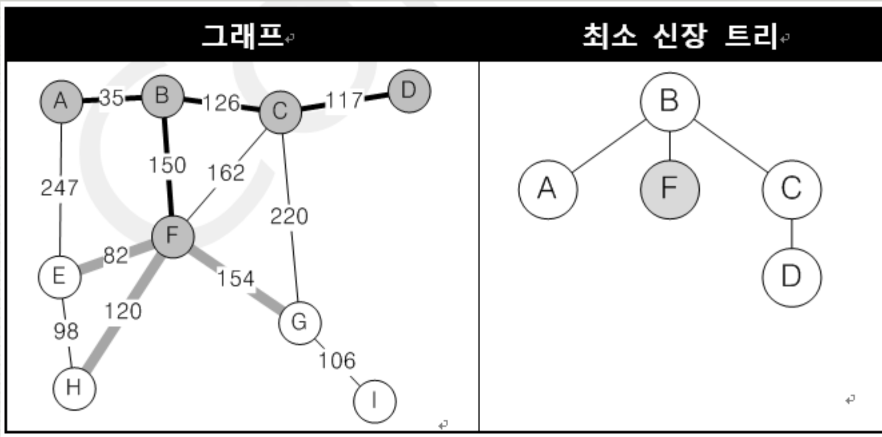


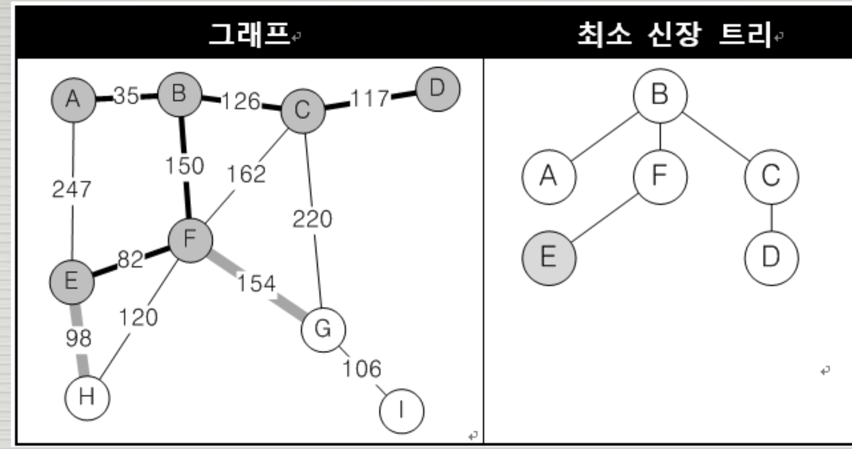


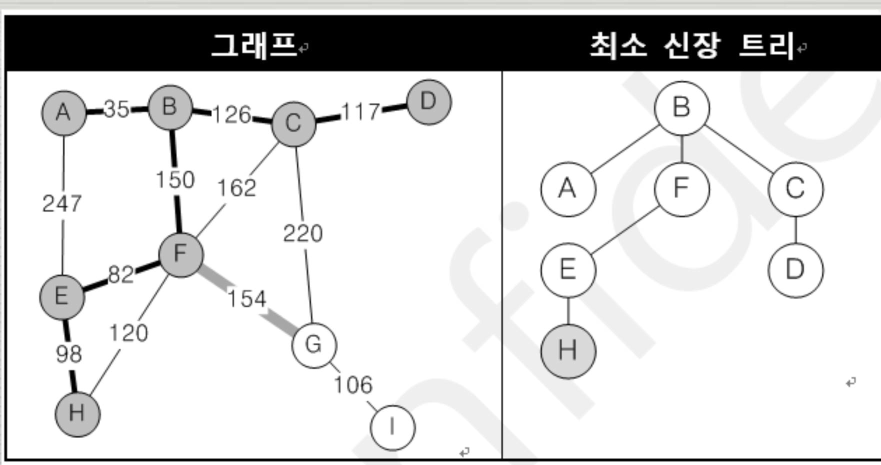


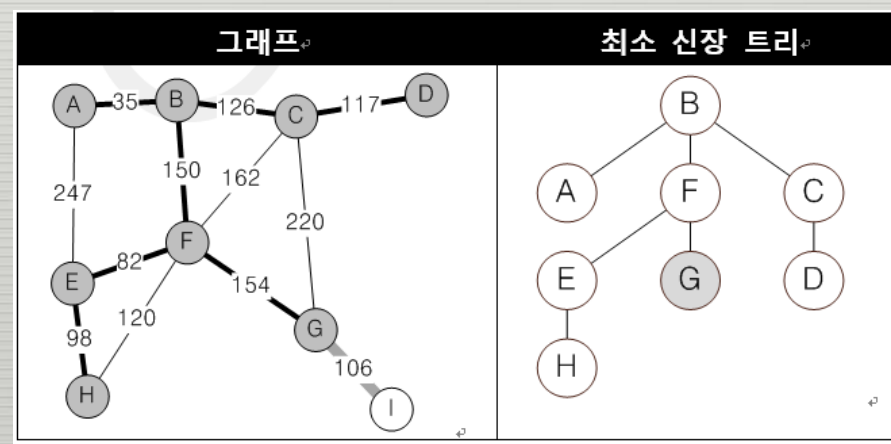


최소 신장 트리: 프림

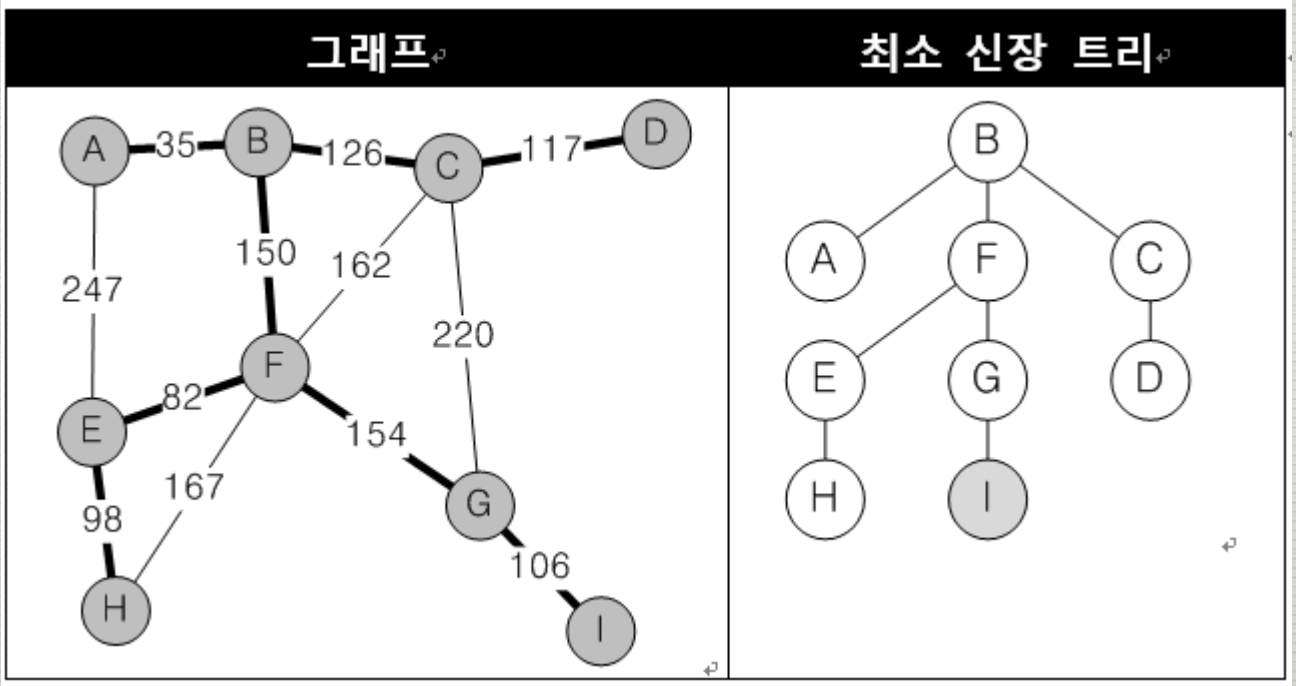

그래프
최소 신장 트리
A 35 B 126 C 117 D
247 150 162 220
E 82 F 154 G
98 120 106
H I
B
A F C
D


그래프
최소 신장 트리
A 35 B 126 C 117 D
247 150 162 220
E 82 F 154 G
98 120 106
H I
B
A F C
E D


그래프
최소 신장 트리
A 35 B 126 C 117 D
247 150 162 220
E 82 F 154 G
98 120 106
H I
B
A F C
E D
H

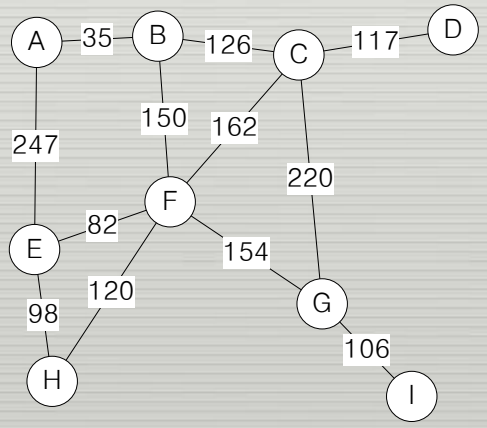

그래프
최소 신장 트리
A 35 B 126 C 117 D
247 150 162 220
E 82 F 154 G
98 120 106
H I
B
A F C
E G D
H

최소 신장 트리: 프림


그래프
최소 신장 트리
A
B
C
D
E
F
G
H
I
35
126
117
150
162
247
220
82
154
167
98
106

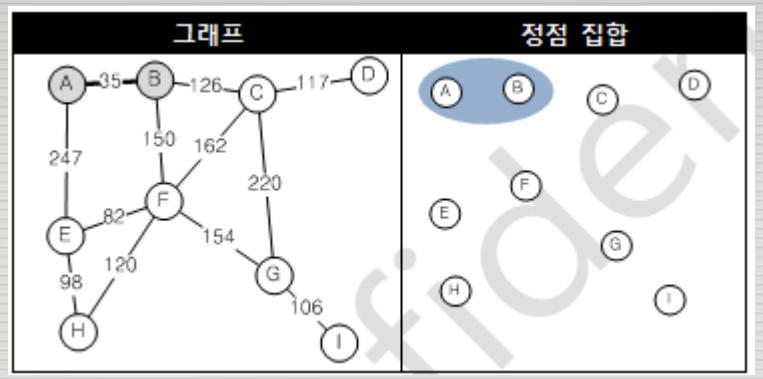
최소 신장 트리: 크루스칼

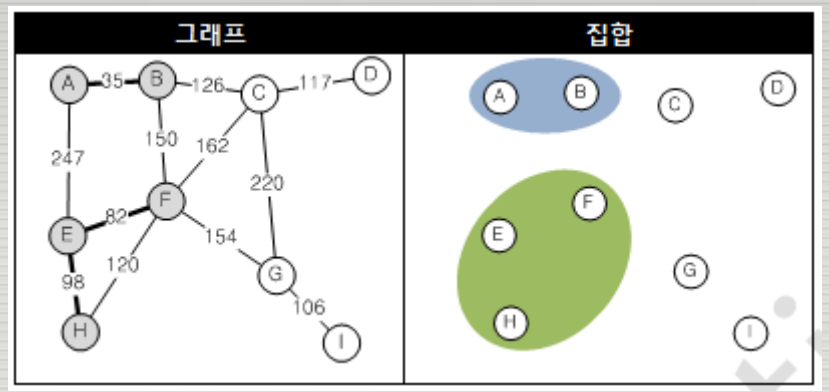
- **크루스칼 알고리즘**
  1) 그래프 내의 모든 간선을 가중치의 오름차순으로 목록을 만듭니다.
  2) 1)에서 만든 간선의 목록을 차례대로 순회하면서 간선을 최소 신장 트리에 추가합니다. 단, 이때 추가된 간선으로 인해 최소 신장 트리 내에 사이클이 형성되면 안됩니다
- **크루스칼 알고리즘의 예 : 간선의 오름차순 목록**

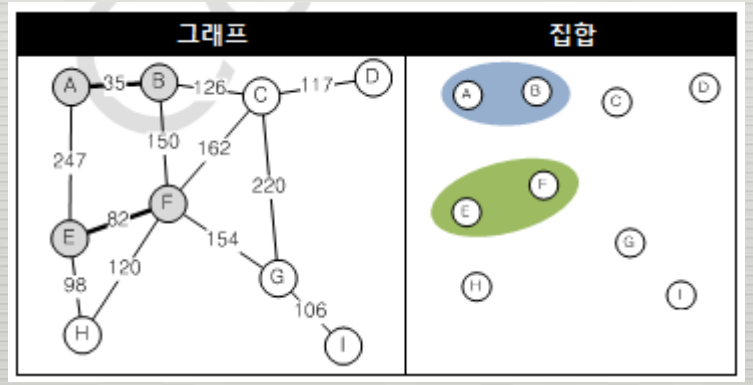


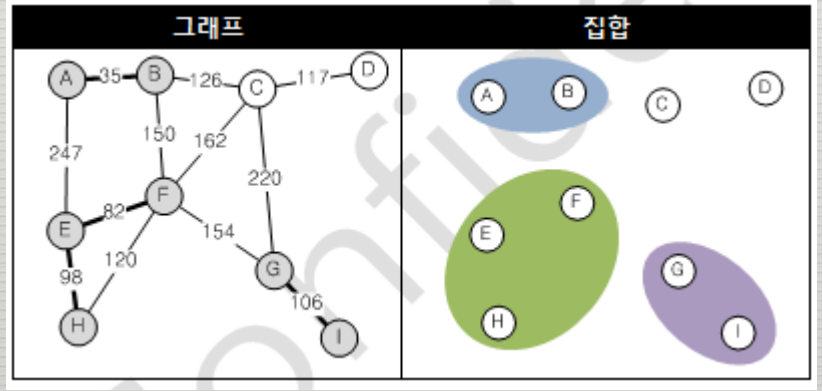
A - B : 35
E - F : 82
E - H : 98
G - I : 106
C - D : 117
F - H : 120
B - C : 126
B - F : 150
F - G : 154
C - F : 162
C - G : 220
A - E : 247

최소 신장 트리: 크루스칼

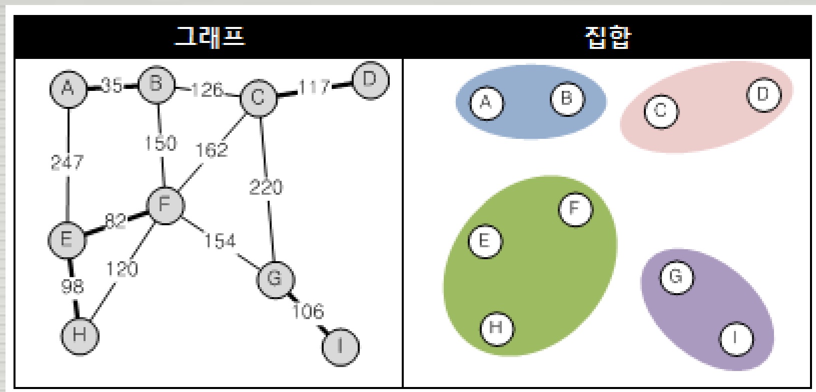

그래프
정점 집합
A
B
C
D
E
F
G
H
I
35
126
117
247
150
162
220
82
154
120
98
106

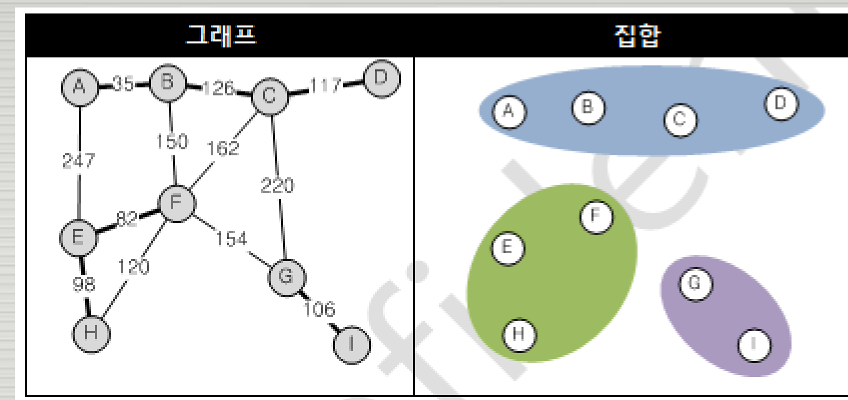

그래프
집합
A
B
C
D
E
F
G
H
I
35
126
117
247
150
162
220
82
154
120
98
106

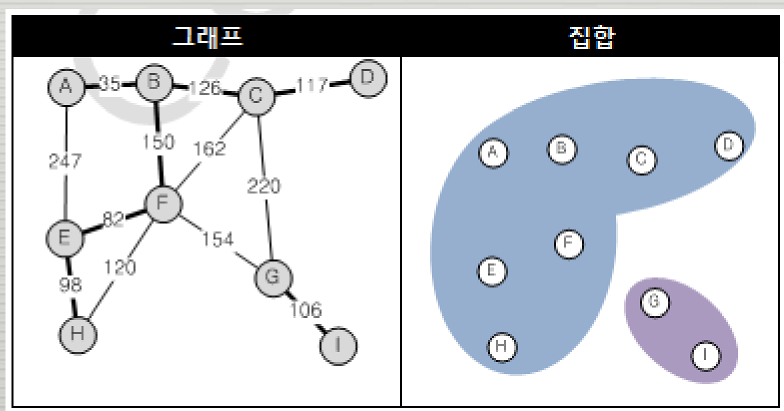

그래프
집합
A
B
C
D
E
F
G
H
I
35
126
117
247
150
162
220
82
154
120
98
106

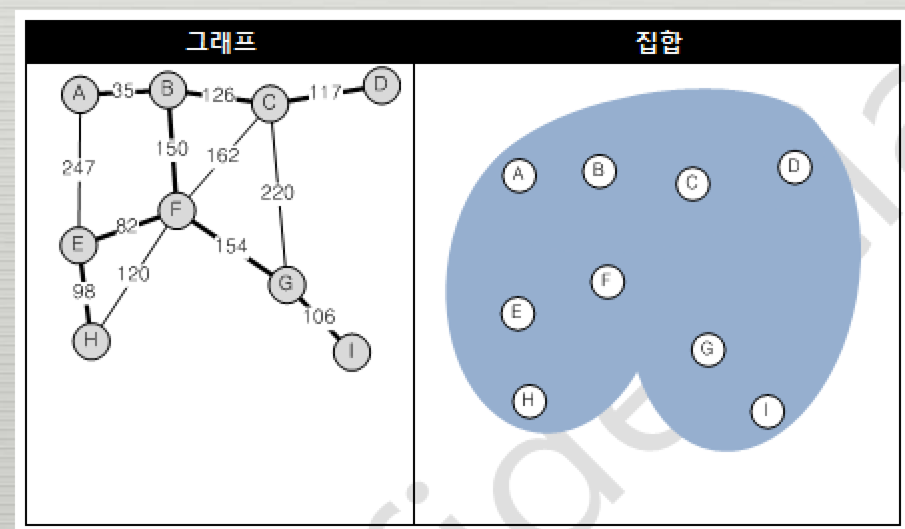

그래프
집합
A
B
C
D
E
F
G
H
I
35
126
117
247
150
162
220
82
154
120
98
106

최소 신장 트리: 크루스칼


그래프
집합
A
35
B
126
C
117
D
247
150
162
220
F
E
82
154
G
120
98
H
106
I
A
B
C
D
F
E
H
G
I


그래프
집합
A
35
B
126
C
117
D
247
150
162
220
F
E
82
154
G
120
98
H
106
I
A
B
C
D
F
E
H
G
I


그래프
집합
A
35
B
126
C
117
D
247
150
162
220
F
E
82
154
G
120
98
H
106
I
A
B
C
D
F
E
H
G
I


그래프
집합
A
35
B
126
C
117
D
247
150
162
220
F
E
82
154
G
120
98
H
106
I
A
B
C
D
F
E
G
H
I

최단 경로 탐색

- 그래프 내의 한 정점에서 다른 정점으로 이동할 때 가중치 합이 최소값이 되도록 만드는 경로를 찾는 알고리즘

- 다익스트라 알고리즘
  - 최소 신장 트리 알고리즘인 프림 알고리즘과 유사.
  - 프림 알고리즘이 단순히 간선의 길이를 이용해 어떤 간선을 먼저 연결할지를 결정하는데 반해, 다익스트라 알고리즘은 "*경로의 길이*"를 감안해서 간선을 연결한다

최단 경로 탐색

다익스트라 알고리즘

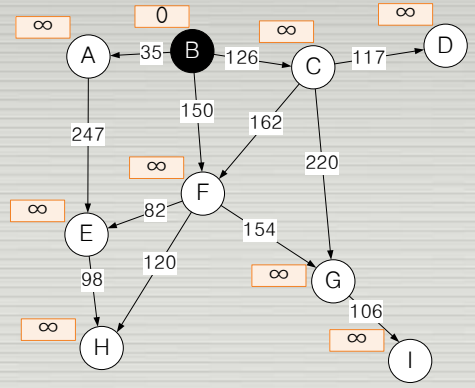
1) 각 정점 위에 시작점으로부터 자신에게 이르는 경로의 길이를 저장할 곳을 준비하고 모든 정점 위에 있는 경로의 길이를 ∞(무한대)로 초기화.

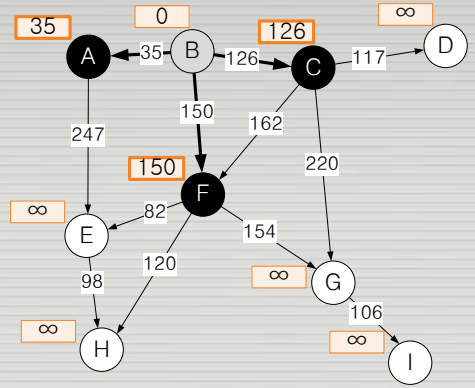
2) 시작 정점의 경로 길이를 0으로 초기화하고(시작 정점에서 시작정점까지의 거리는 0이기 때문) 최단 경로에 추가합니다.

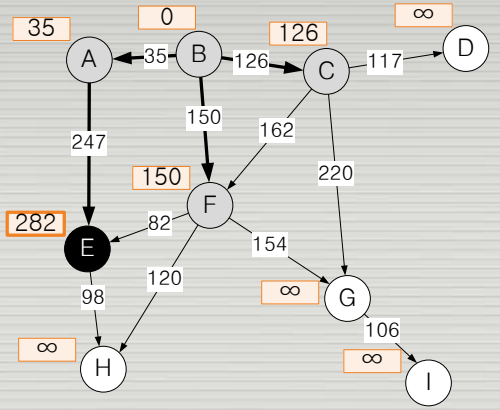
3) 최단 경로에 새로 추가된 정점의 인접 정점들에 대해 경로 길이를 갱신하고 이들을 최단 경로에 추가합니다. 만약 추가하려는 인접 정점이 이미 최단 경로 안에 존재한다면 갱신되기 이전의 경로 길이가 새로운 경로의 길이보다 더 큰 경우에 한해, 다른 선행 정점을 지나던 기존의 경로를 현재 정점을 경유하도록 수정합니다.

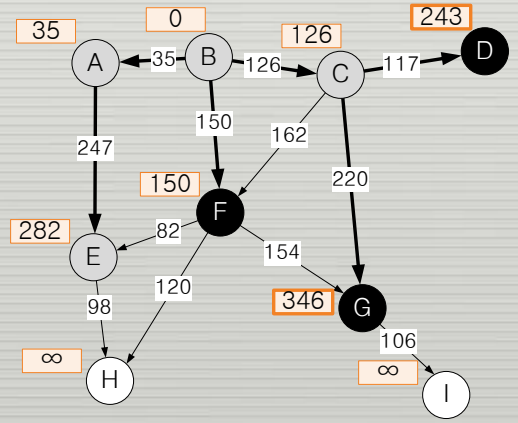
4) 그래프 내의 모든 정점이 최단 경로에 소속될 때까지 3) 과정을 반복합니다.

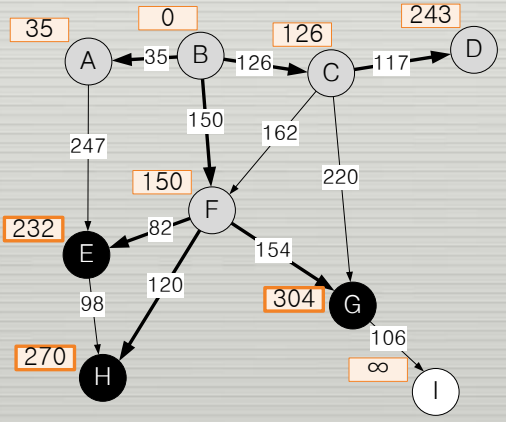
최단 경로 탐색

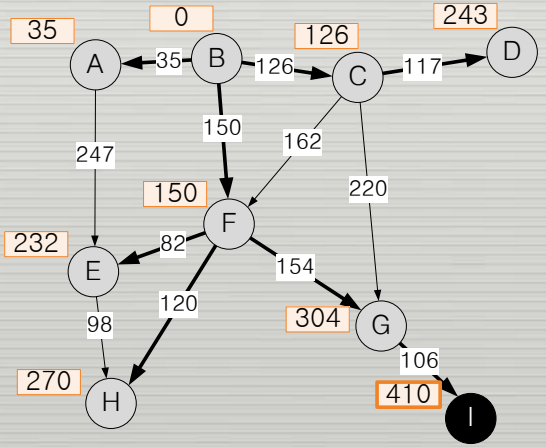

∞
0
∞
∞
A
35
B
126
C
117
D
150
247
162
∞
220
F
∞
82
E
154
120
98
∞
G
106
∞
H
∞
I


35
0
126
∞
A
35
B
126
C
117
D
150
247
162
150
220
F
∞
82
E
154
120
98
∞
G
106
∞
H
∞
I


35
0
126
∞
A
35
B
126
C
117
D
150
247
162
150
220
F
282
82
E
154
120
98
∞
G
106
∞
H
∞
I


35
0
126
243
A
35
B
126
C
117
D
150
247
162
150
220
F
282
82
E
154
120
98
346
G
106
∞
H
∞
I


35
0
126
243
A
35
B
126
C
117
D
150
247
162
150
220
F
232
82
E
154
120
98
304
G
106
270
H
∞
I


35
0
126
243
A
35
B
126
C
117
D
150
247
162
150
220
F
232
82
E
154
120
98
304
G
106
270
H
410
I