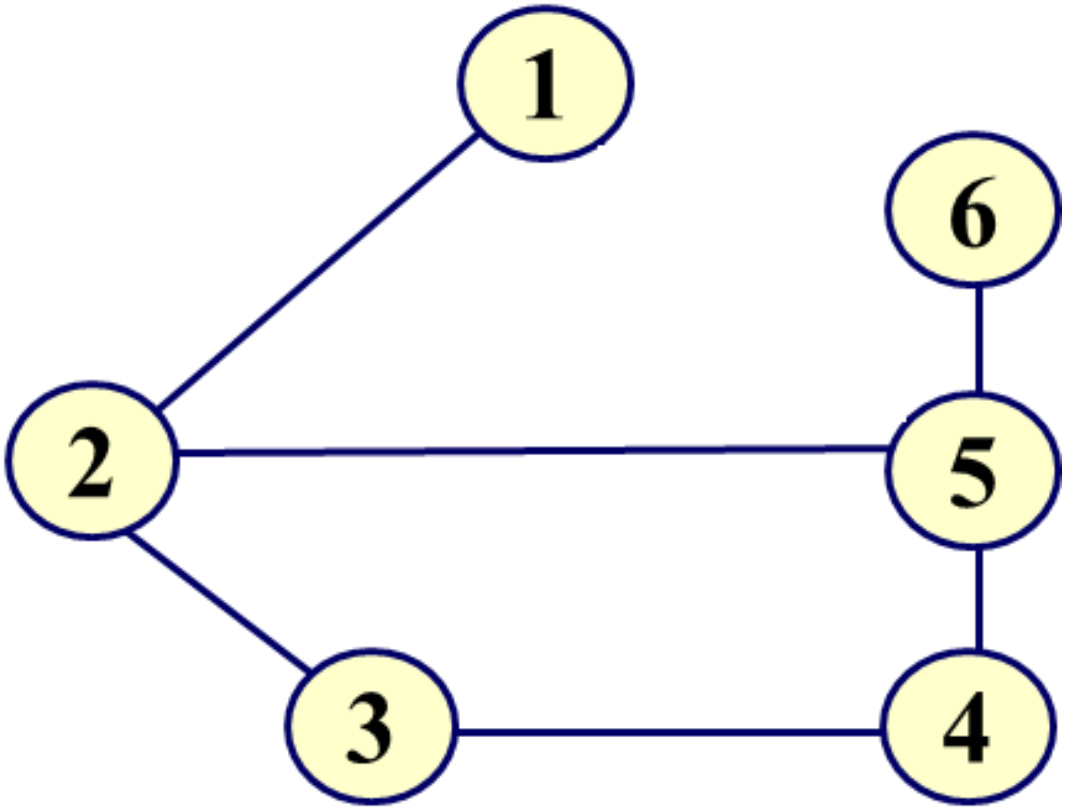
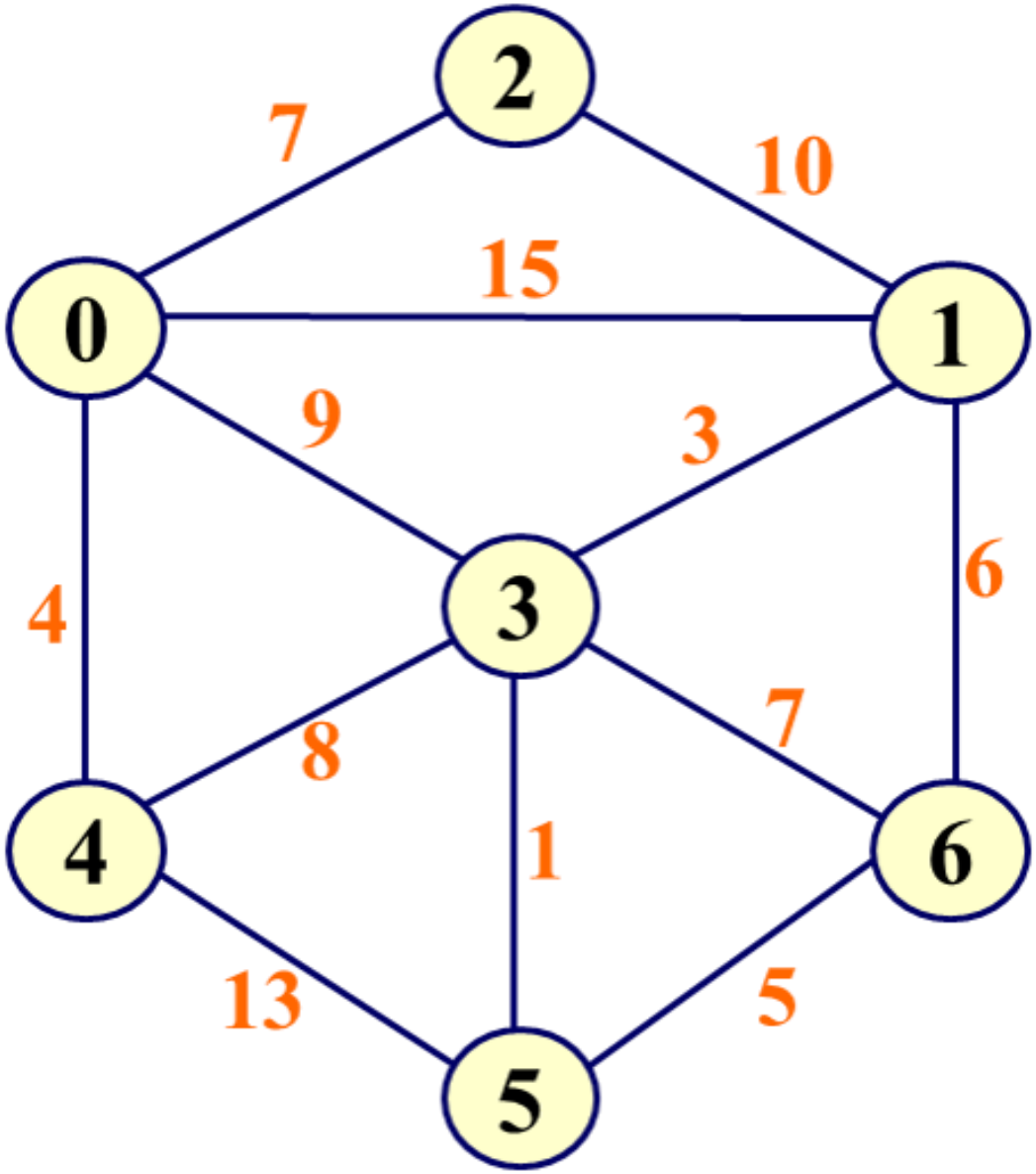
10주차 퀴즈 그림 모음

HaRim Jung, Ph.D.
Visiting Professor / Senior Researcher
SKKU Institute for Convergence / Convergence Research Institute
Sungkyunkwan University, Korea

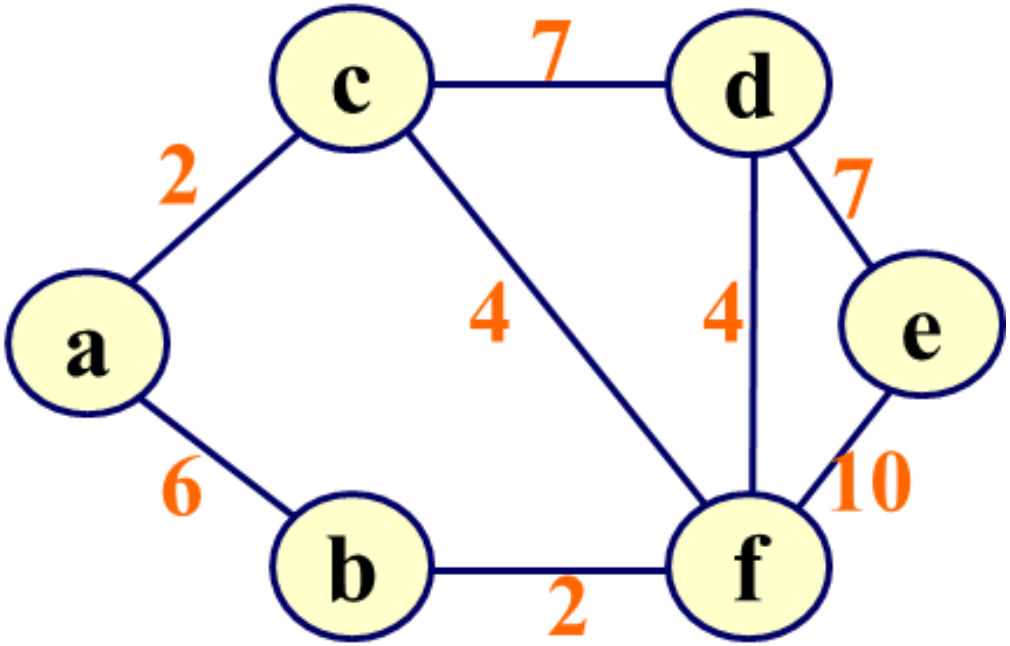
10주차 퀴즈 그림1



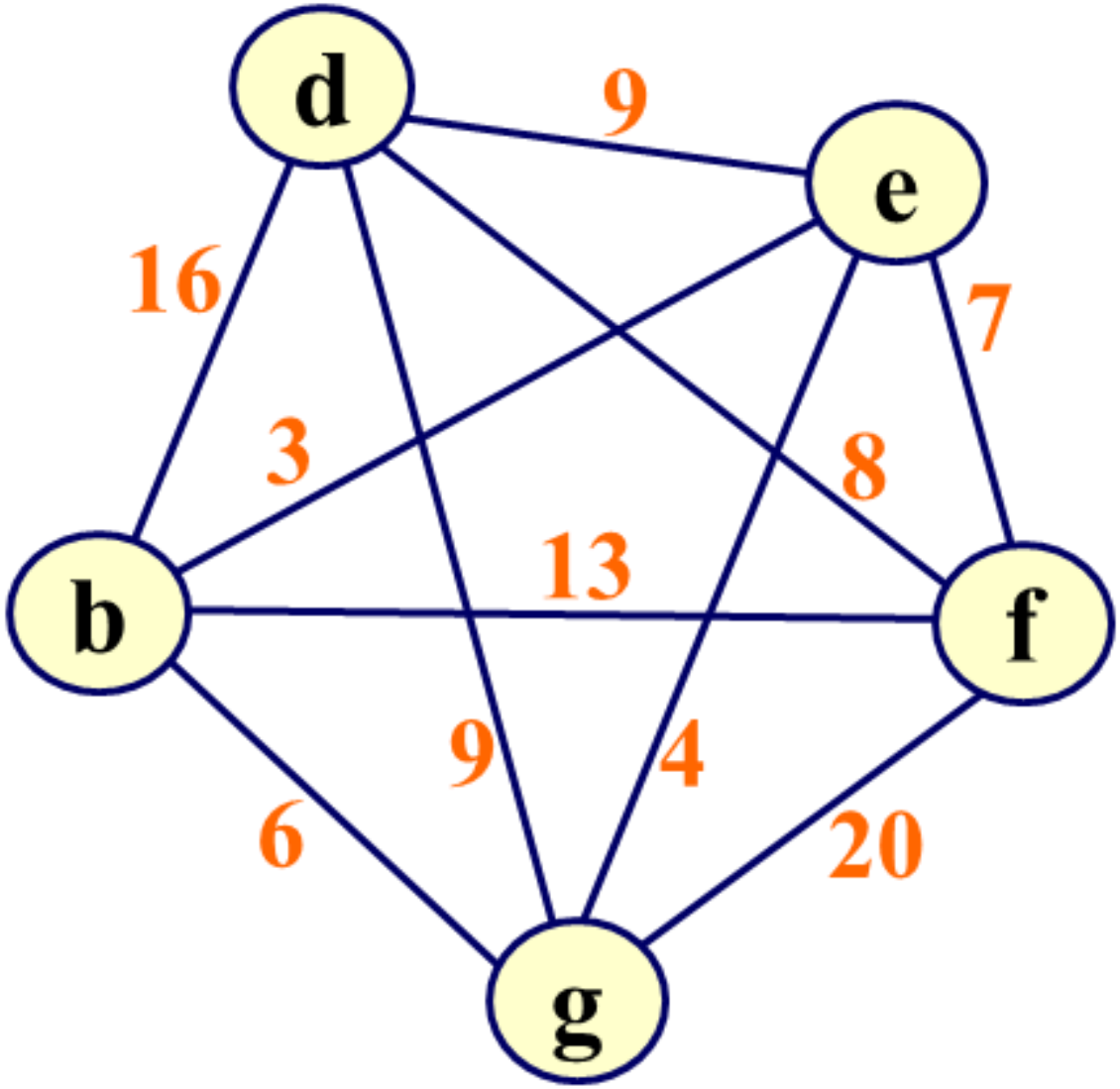
10주차 퀴즈 그림2



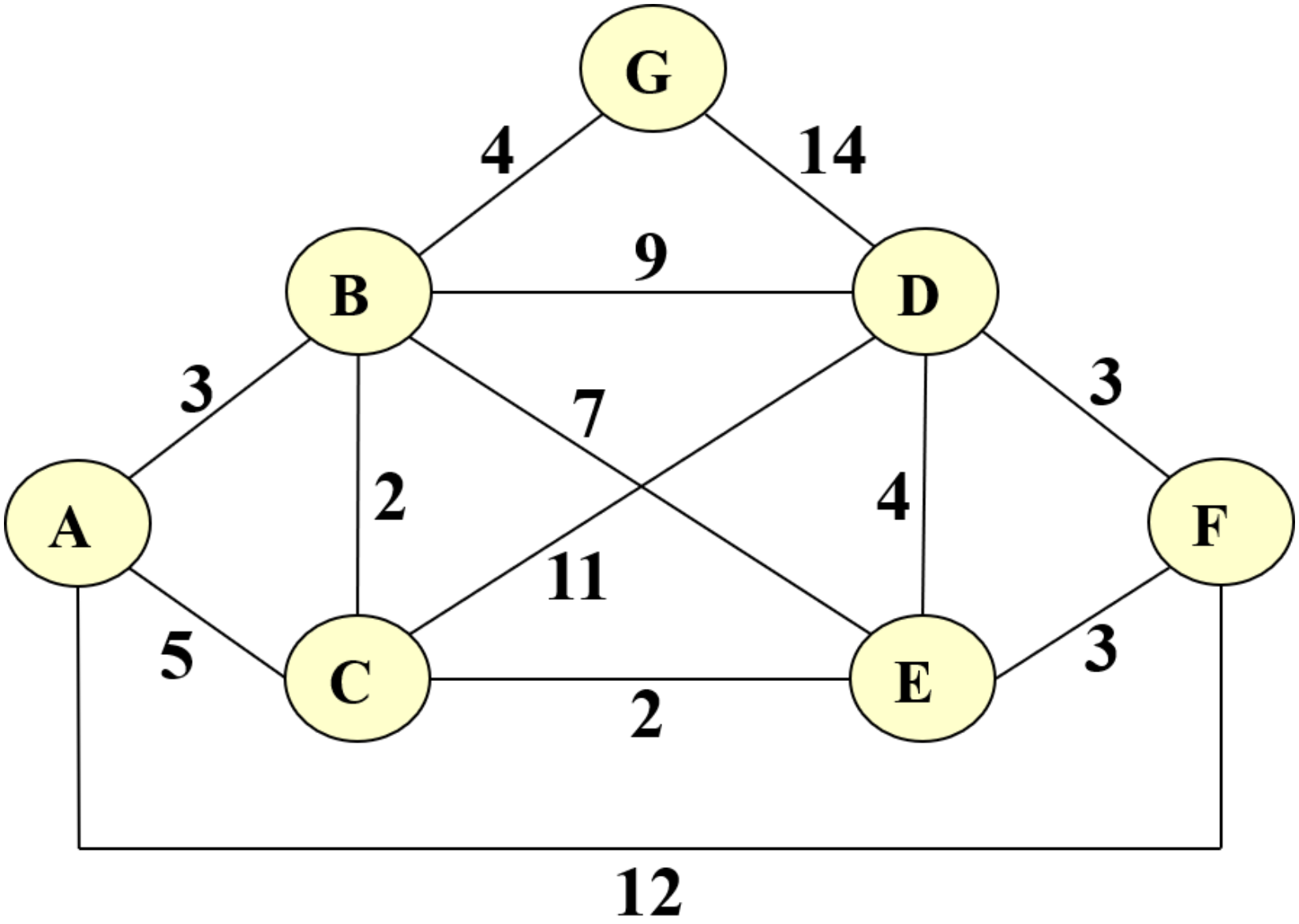
10주차 퀴즈 그림3



10주차 퀴즈 그림4



10주차 퀴즈 그림5



그래프- 3

-탐욕적 알고리즘과 최단 경로 문제-

HaRim Jung, Ph.D.

Visiting Professor / Senior Researcher

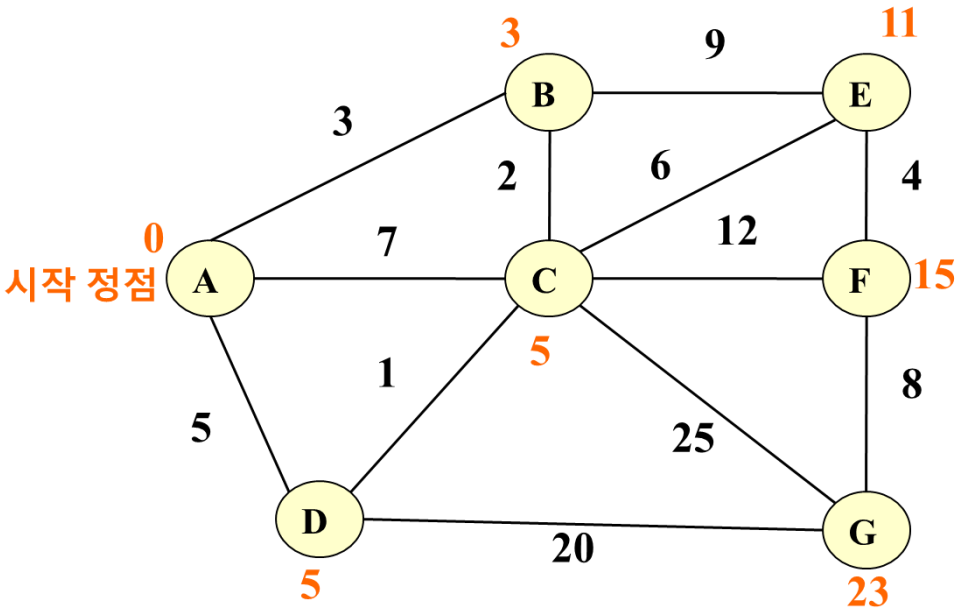
SKKU Institute for Convergence / Convergence Research Institute

Sungkyunkwan University, Korea

탐욕적 알고리즘 – 최단 경로 문제(1/11)

❑ 최단 경로 문제(shortest path problem)

- 가중치 그래프에서 시작 정점으로부터 도착 정점까지의 최단 경로(가중치의 합이 최소가 되는 경로)를 찾는 문제
- **다익스트라 알고리즘(Dijkstra's algorithm)**
  - 주어진 (**무방향**·방향) 가중치 그래프에서 **특정 시작 정점에서 다른 모든 정점까지의 최단 (단순) 경로**를 찾는 문제를 해결하기 위한 탐욕적 알고리즘(단, **가중치는 음수가 아니어야 함**)
  - 예: 아래의 무방향 가중치 그래프에서 시작 정점 A로부터 다른 모든 정점(B, C, D, E, F, G)까지의 최단 경로



[시작 정점 A로부터 B까지의 최단 경로] A → B
[시작 정점 A로부터 C까지의 최단 경로] A → B → C
[시작 정점 A로부터 D까지의 최단 경로] A → D
[시작 정점 A로부터 E까지의 최단 경로] A → B → C → E
[시작 정점 A로부터 F까지의 최단 경로] A → B → C → E → F
[시작 정점 A로부터 G까지의 최단 경로] A → B → C → E → F → G

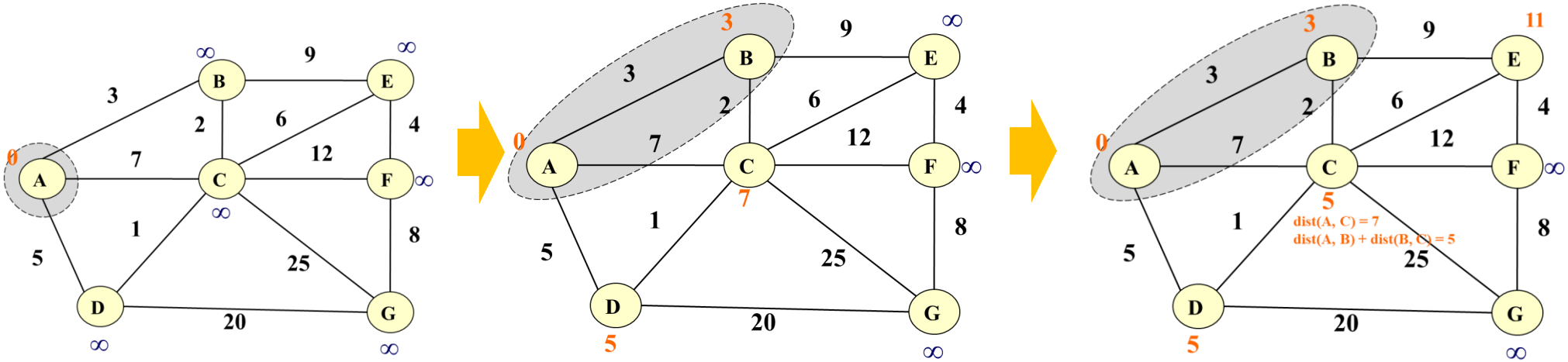
탐욕적 알고리즘 – 최단 경로 문제(2/11)

❑ 최단 경로 문제(shortest path problem) contd.

- 다익스트라 알고리즘 설계 과정

1. 정점들의 집합: V, 간선들의 집합: E, 시작 정점: s
2. 집합 S: 시작 정점 s로부터 최종 최단 경로가 이미 발견된 정점들의 집합으로 S = {s}로 초기화
3. 최단 거리 기록 리스트 D = [dist(s, s), dist(s, u_1), dist(s, u_2), …, dist(s, $u_{|V|}$)]: S에 존재하는 정점만을 거쳐서 다른 정점에 이르는 최단 거리를 기록하는 리스트로 D[s] = 0, s를 제외한 각 정점 u에 대해 (s, u)가 존재하면, i.e., (s, u) ∈ E, D[u]는 간선의 가중치로, (s, u)가 존재하지 않으면, i.e., (s, u) ∉ E D[u]는 무한대로 초기화

– **선정 과정 & 적정성 검사**: 현재 시작 정점 s로부터 최단 경로가 되는(선택 기준) 정점, i.e., s와의 거리가 최소인 정점 v를 V – S에 속한 정점 중에서 선정하여 S에 포함시키고, V – {v}에 속한 각 정점 u에 대한 dist(s, u) 갱신(edge relaxation: 간선 완화)



S = {A}, A를 이용해서 최단 경로가 되는 정점 B를 선정하여 S에 포함
Note: dist(A, B) = 3, dist(A, C) = 7, dist(A, D) = 5

S = {A, B}, B가 S에 포함됨에 따라 기존 dist(A, C) = 7은 dist(A, B) + dist(B, C) = 5이므로 5로 변경되고(따라서 A에서 C까지의 최단 경로가 A → C에서 A → B → C로 변경되고), 기존 dist(A, E) = ∞는 dist(A, B) + dist(B, E) = 15 이므로 15로 변경됨(A에서 E까지의 최단 경로는 존재하지 않았지만 최단 경로 A → B → E가 생성됨)

– **해답 점검**: S = V인지 점검하고(시작 정점으로부터 다른 모든 정점까지의 단순 경로를 찾았는지 점검하고)아니라면 선정과정 및 적정성 검사 반복

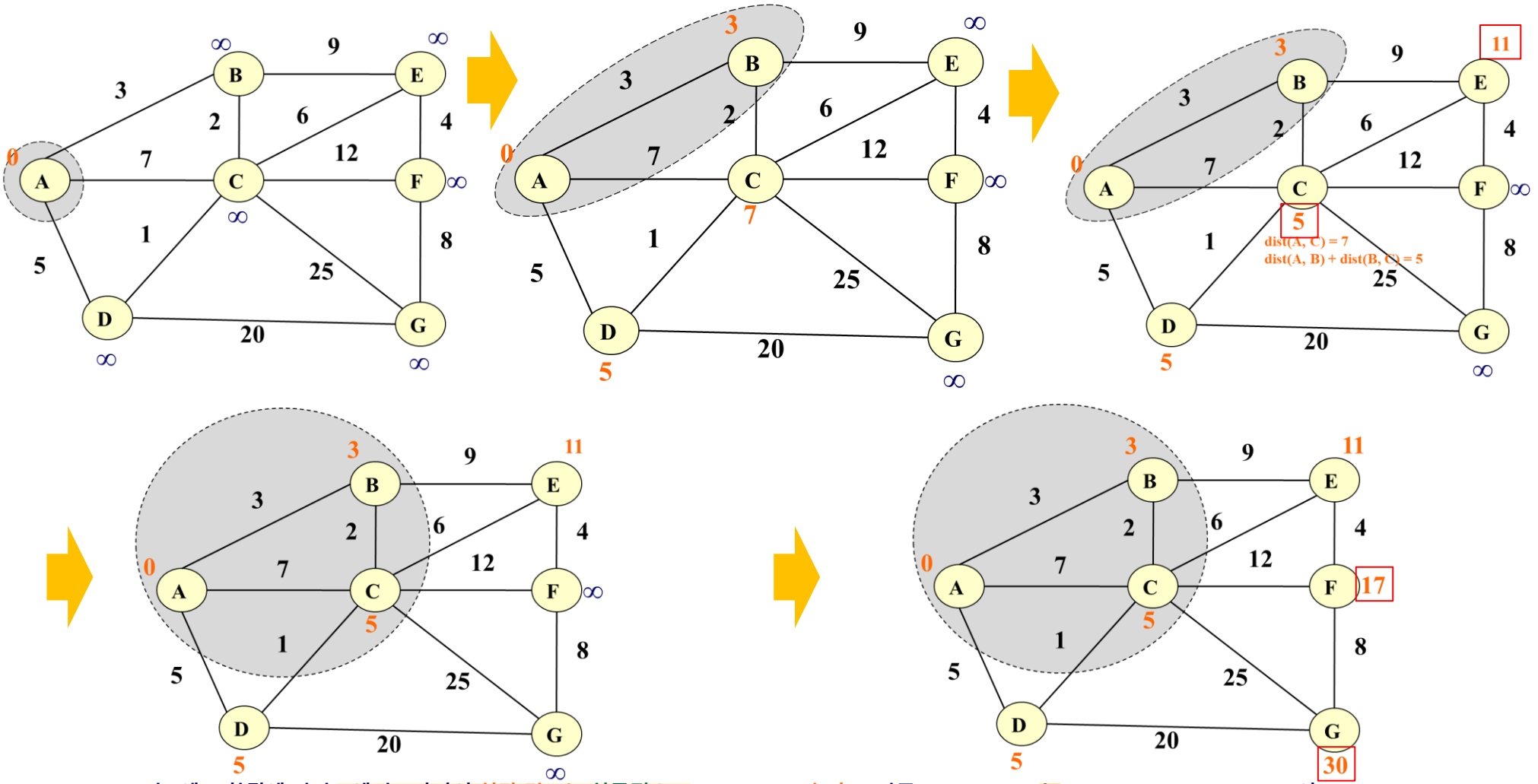
- 위의 예에서 S (= {A, B}) ≠ V (= {A, B, C, D, E, F, G})이므로 {A, B}에 속한 정점만을 거쳐서 최단 경로가 되는 정점 C 혹은 D를 찾음

B가 S에 포함되면, 시작 정점 A에서 정점 B까지의 최종 최단 경로는 A → B라는 의미(A에서 C까지의 최단 경로 A → C와 A에서 D까지의 최단 경로 A → D의 경우 현재까지는 최종적으로 결정되지 않음

탐욕적 알고리즘 – 최단 경로 문제(3/11)

❑ 최단 경로 문제(shortest path problem) contd.

- 다익스트라 알고리즘의 예

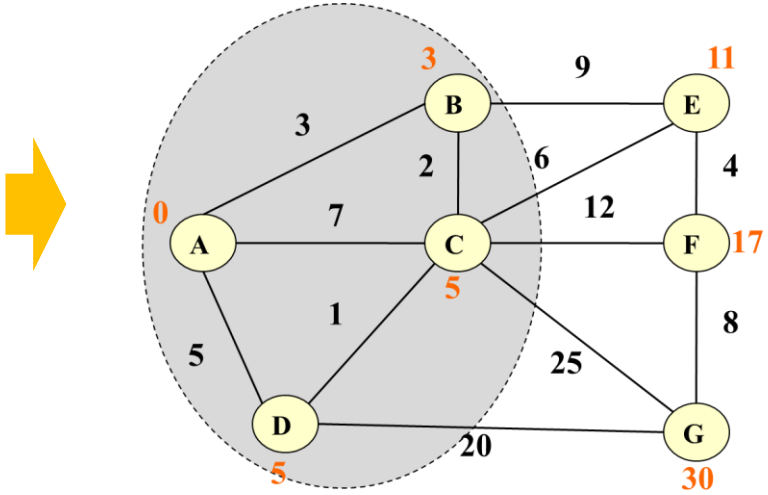


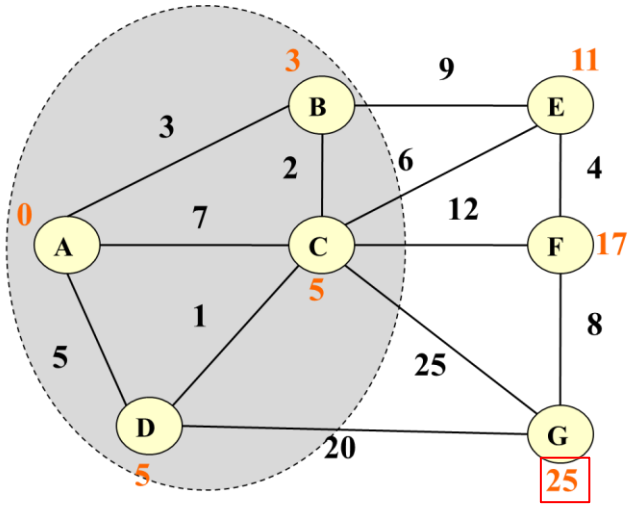
S = {A, B, C}, C가 S에 포함됨에 따라 A에서 C까지의 최단 경로는 최종적으로 A → B → C가 되고, 기존 dist(A, F) = ∞는 dist(A, C) + dist(C, F) = 17이므로 17로 변경되고(따라서 A에서 F까지의 최단 경로가 존재하지 않았지만 A → B → C → F로 변경되고), 기존 dist(A, G) = ∞는 dist(A, C) + dist(C, G) = 30 이므로 30으로 변경됨(A에서 G까지의 최단 경로는 존재하지 않았지만 최단 경로 A → B → C → G 가 생성됨)

탐욕적 알고리즘 – 최단 경로 문제(4/11)

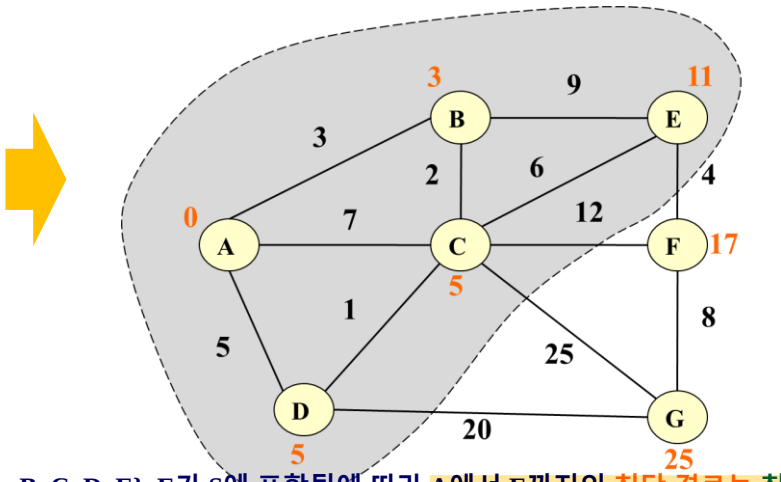
❑ 최단 경로 문제(shortest path problem) contd.

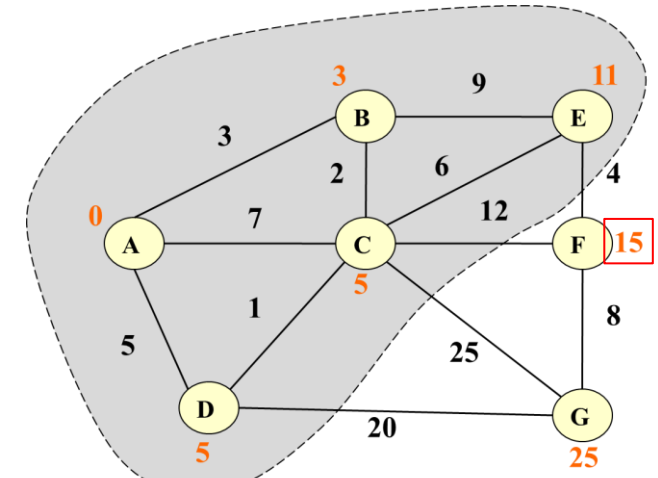
- 다익스트라 알고리즘의 예 contd.





S = {A, B, C, D}, D가 S에 포함됨에 따라 A에서 D까지의 최단 경로는 최종적으로 A → D가 되고, 기존 dist(A, G) = 30은 dist(A, D) + dist(D, G) = 25이므로 30으로 변경됨(따라서 A에서 G까지의 최단 경로는 A → B → C → F에서 A → D → G로 변경 됨)



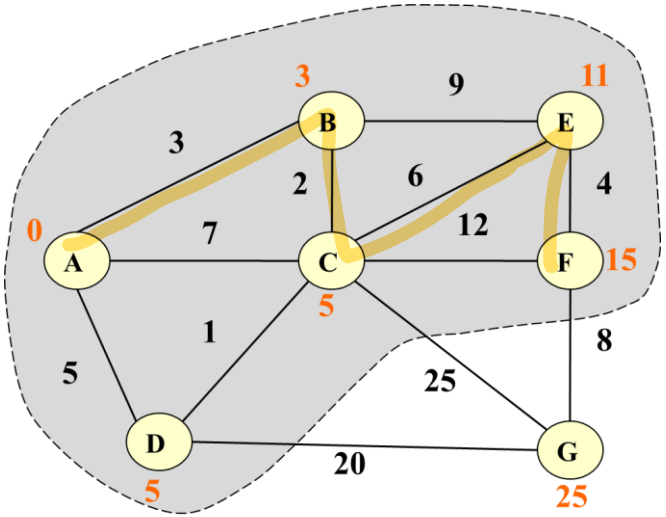


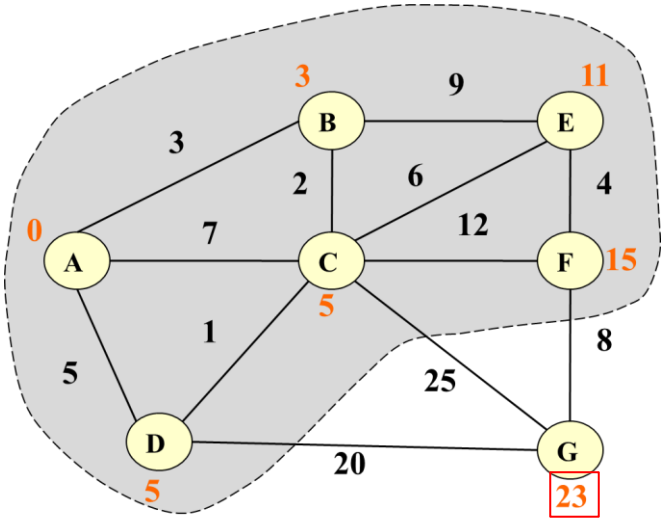
S = {A, B, C, D, E}, E가 S에 포함됨에 따라 A에서 E까지의 최단 경로는 최종적으로 A → B → C → E 가 되고, 기존 dist(A, F) = 17은 dist(A, E) + dist(E, F) = 15이므로 15로 변경됨(따라서 A에서 F까지의 최단 경로는 A → B → C → F에서 A → B → C → E → F로 변경 됨)

탐욕적 알고리즘 – 최단 경로 문제(5/11)

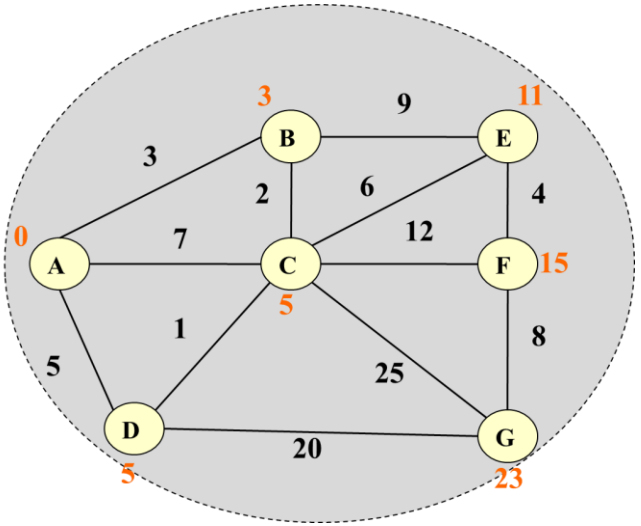
❑ 최단 경로 문제(shortest path problem) contd.

- 다익스트라 알고리즘의 예 contd.





S = {A, B, C, D, E, F}, F가 S에 포함됨에 따라 A에서 F까지의 최단 경로는 최종적으로 A → B → C → E → F가 되고, 기존 dist(A, G) = 23은 dist(A, F) + dist(F, G) = 23이므로 23으로 변경됨(따라서 A에서 G까지의 최단 경로는 A → D→ G에서 A → B → C → E → F → G로 변경 됨)



S = {A, B, C, D, E, F, G}, G가 S에 포함됨에 따라 A에서 G까지의 최단 경로는 최종적으로 A → B → C → E → F → G가 됨

해답점검: S = V이므로 종료

탐욕적 알고리즘 – 최단 경로 문제(6/11)

❑ 다익스트라 알고리즘의 구현

```
1  def choose_min_vertex(dist, found):
2      min = INF
3      minpos = -1
4      for i in range(len(dist)) :
5          if dist[i] < min and found[i] == False:
6              min = dist[i]
7              minpos = i
8      return minpos;
9
10 def dijkstra(vtx, adj, start):
11     vsize = len(vtx)
12     dist = list(adj[start])
13     path = [start] * vsize
14     found= [False] * vsize
15     found[start] = True
16     dist[start] = 0
17     for i in range(vsize):
18         u = choose_min_vertex(dist, found)
19         found[u] = True
20         for w in range(vsize):
21             if not found[w]:
22                 if  dist[u] + adj[u][w] < dist[w]:
23                     dist[w] = dist[u] + adj[u][w]
24                     path[w] = u
25     return path
```

```
27 INF = float('inf')
28 vertex =   ['A', 'B', 'C', 'D', 'E', 'F', 'G']
29 adj_matrix = [ [0, 3, 7, 5, INF, INF, INF],
30                [3, 0, 2, INF, 9, INF, INF],
31                [7, 2, 0, 1, 6, 12, 25],
32                [5, INF, 1, 0, INF, INF, 20],
33                [INF, 9, 6, INF, 0, 4, INF],
34                [INF, INF, 12, INF, 4, 0, 8],
35                [INF, INF, 25, 20, INF, 8, 0]]
36
37 print("Shortest Path by Dijkstra's Algorithm")
38 start = 0
39 path = dijkstra(vertex, adj_matrix, start)
40
41 for end in range(len(vertex)):
42     if end != start:
43         print("[Shortest Path: %s->%s] %s" %
44                 (vertex[start], vertex[end], vertex[end]), end='')
45         while (path[end] != start):
46             print(" <- %s" % vertex[path[end]], end='')
47             end = path[end]
48         print(" <- %s" % vertex[path[end]])
```

- 라인 1-8: 시작 정점 s에서 최단 경로가 되는 정점인 v를 V – S에 속한 정점 중에서 선정하기 위한 choose_min_vertex 함수 정의(인자 dist는 시작 정점 s와 다른 모든 정점들 간의 거리를 저장하는 Python 리스트, 인자 found는 방문한 정점, i.e., S에 포함된 정점을 표시하기 위한 Python 리스트)
  - 라인 2-3: dist(s, v)와 v의 인덱스를 저장하기 위한 변수 min, minpos 선언 및 최기화
  - 라인 4-7: 그래프의 모든 정점에 대해서 방문하지 않은 (i.e., V – S에 속하는) 정점 중 s와의 거리가 최소인 정점 v를 선택 후 v의 인덱스를 반환

탐욕적 알고리즘 – 최단 경로 문제(7/11)

❑ 다익스트라 알고리즘의 구현

```
1  def choose_min_vertex(dist, found):
2      min = INF
3      minpos = -1
4      for i in range(len(dist)) :
5          if dist[i] < min and found[i] == False:
6              min = dist[i]
7              minpos = i
8      return minpos;
9
10 def dijkstra(vtx, adj, start):
11     vsize = len(vtx)
12     dist = list(adj[start])
13     path = [start] * vsize
14     found= [False] * vsize
15     found[start] = True
16     dist[start] = 0
17     for i in range(vsize):
18         u = choose_min_vertex(dist, found)
19         found[u] = True
20         for w in range(vsize):
21             if not found[w]:
22                 if  dist[u] + adj[u][w] < dist[w]:
23                     dist[w] = dist[u] + adj[u][w]
24                     path[w] = u
25     return path
```

```
27 INF = float('inf')
28 vertex =   ['A', 'B', 'C', 'D', 'E', 'F', 'G']
29 adj_matrix = [ [0, 3, 7, 5, INF, INF, INF],
30                [3, 0, 2, INF, 9, INF, INF],
31                [7, 2, 0, 1, 6, 12, 25],
32                [5, INF, 1, 0, INF, INF, 20],
33                [INF, 9, 6, INF, 0, 4, INF],
34                [INF, INF, 12, INF, 4, 0, 8],
35                [INF, INF, 25, 20, INF, 8, 0]]
36
37 print("Shortest Path by Dijkstra's Algorithm")
38 start = 0
39 path = dijkstra(vertex, adj_matrix, start)
40
41 for end in range(len(vertex)):
42     if end != start:
43         print("[Shortest Path: %s->%s] %s" %
44                 (vertex[start], vertex[end], vertex[end]), end='')
45         while (path[end] != start):
46             print(" <- %s" % vertex[path[end]], end='')
47             end = path[end]
48         print(" <- %s" % vertex[path[end]])
```

- 라인 11: 그래프의 정점 수를 저장하기 위한 변수 vsize 선언 및 정점 수 할당
- 라인 12: 시작 정점으로부터 다른 모든 정점 간의 최단 경로 거리를 저장하기 위한 Python 리스트 dist 선언 (사이즈는 정점 수와 동일)
- 라인 13: 각 정점의 바로 이전 정점의 인덱스를 저장하기 위한 Python 리스트로 바로 이전 정점을 따라 시작 정점까지 가는 경로가 최단 경로임
  - 예: {A, B, C, D, E, F, G}에서 path[2] = 1(C의 이전 정점은 B), path[1] = 0(B의 이전 정점은 A), 따라서 C의 경우 B → A

탐욕적 알고리즘 – 최단 경로 문제(8/11)

❑ 다익스트라 알고리즘의 구현

```
1  def choose_min_vertex(dist, found):
2      min = INF
3      minpos = -1
4      for i in range(len(dist)) :
5          if dist[i] < min and found[i] == False:
6              min = dist[i]
7              minpos = i
8      return minpos;
9
10 def dijkstra(vtx, adj, start):
11     vsize = len(vtx)
12     dist = list(adj[start])
13     path = [start] * vsize
14     found= [False] * vsize
15     found[start] = True
16     dist[start] = 0
17     for i in range(vsize):
18         u = choose_min_vertex(dist, found)
19         found[u] = True
20         for w in range(vsize):
21             if not found[w]:
22                 if  dist[u] + adj[u][w] < dist[w]:
23                     dist[w] = dist[u] + adj[u][w]
24                     path[w] = u
25     return path
```

```
27 INF = float('inf')
28 vertex =   ['A', 'B', 'C', 'D', 'E', 'F', 'G']
29 adj_matrix = [ [0, 3, 7, 5, INF, INF, INF],
30                [3, 0, 2, INF, 9, INF, INF],
31                [7, 2, 0, 1, 6, 12, 25],
32                [5, INF, 1, 0, INF, INF, 20],
33                [INF, 9, 6, INF, 0, 4, INF],
34                [INF, INF, 12, INF, 4, 0, 8],
35                [INF, INF, 25, 20, INF, 8, 0]]
36
37 print("Shortest Path by Dijkstra's Algorithm")
38 start = 0
39 path = dijkstra(vertex, adj_matrix, start)
40
41 for end in range(len(vertex)):
42     if end != start:
43         print("[Shortest Path: %s->%s] %s" %
44                 (vertex[start], vertex[end], vertex[end]), end='')
45         while (path[end] != start):
46             print(" <- %s" % vertex[path[end]], end='')
47             end = path[end]
48         print(" <- %s" % vertex[path[end]])
```

- 라인 14: 방문한 정점, i.e., S에 포함된 정점을 표시하기 위한 Python 리스트(특정 정점을 방문하였다면 True, 아니라면 False로 표시) found 선언 및 False로 초기화
- 라인 15-16: S = {s}부터 시작하므로(설계 과정 참조), found에서 시작 정점의 인덱스에 해당하는 값을 True로 설정하고 시작 정점과 시작 정점의 거리는 0이므로 dist에서 시작 정점의 인덱스에 해당하는 값을 0으로 설정
- 라인 17-19: S = V가 될 때까지(라인 17: 해답 점검을 위한 for-루프) 각 단계마다 choose_min_vertex 함수를 호출하여 정점 s와의 거리가 최소인 정점 v를 선정하고(라인 18), found에서 v의 인덱스에 해당하는 값을 True로 설정

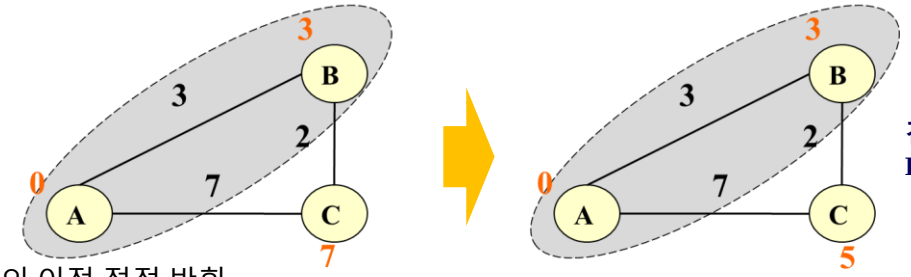
탐욕적 알고리즘 – 최단 경로 문제(9/11)

❑ 다익스트라 알고리즘의 구현

```
1  def choose_min_vertex(dist, found):
2      min = INF
3      minpos = -1
4      for i in range(len(dist)) :
5          if dist[i] < min and found[i] == False:
6              min = dist[i]
7              minpos = i
8      return minpos;
9
10 def dijkstra(vtx, adj, start):
11     vsize = len(vtx)
12     dist = list(adj[start])
13     path = [start] * vsize
14     found= [False] * vsize
15     found[start] = True
16     dist[start] = 0
17     for i in range(vsize):
18         u = choose_min_vertex(dist, found)
19         found[u] = True
20         for w in range(vsize):
21             if not found[w]:
22                 if  dist[u] + adj[u][w] < dist[w]:
23                     dist[w] = dist[u] + adj[u][w]
24                     path[w] = u
25     return path
```

```
27 INF = float('inf')
28 vertex =   ['A', 'B', 'C', 'D', 'E', 'F', 'G']
29 adj_matrix = [ [0, 3, 7, 5, INF, INF, INF],
30                [3, 0, 2, INF, 9, INF, INF],
31                [7, 2, 0, 1, 6, 12, 25],
32                [5, INF, 1, 0, INF, INF, 20],
33                [INF, 9, 6, INF, 0, 4, INF],
34                [INF, INF, 12, INF, 4, 0, 8],
35                [INF, INF, 25, 20, INF, 8, 0]]
36
37 print("Shortest Path by Dijkstra's Algorithm")
38 start = 0
39 path = dijkstra(vertex, adj_matrix, start)
40
41 for end in range(len(vertex)):
42     if end != start:
43         print("[Shortest Path: %s->%s] %s" %
44                 (vertex[start], vertex[end], vertex[end]), end='')
45         while (path[end] != start):
46             print(" <- %s" % vertex[path[end]], end='')
47             end = path[end]
48         print(" <- %s" % vertex[path[end]])
```

- 라인 20-24: 간선 완화를 통해 아직 방문하지 않은 정점들의 이전 정점 갱신



간선 완화에 의해 아직 방문하지 않은 정점 C의 이전 정점 A를 B로 갱신

- 라인 25: 각 정점의 이전 정점 반환

탐욕적 알고리즘 – 최단 경로 문제(10/11)

❑ 다익스트라 알고리즘의 구현

```
1  def choose_min_vertex(dist, found):
2      min = INF
3      minpos = -1
4      for i in range(len(dist)) :
5          if dist[i] < min and found[i] == False:
6              min = dist[i]
7              minpos = i
8      return minpos;
9
10 def dijkstra(vtx, adj, start):
11     vsize = len(vtx)
12     dist = list(adj[start])
13     path = [start] * vsize
14     found= [False] * vsize
15     found[start] = True
16     dist[start] = 0
17     for i in range(vsize):
18         u = choose_min_vertex(dist, found)
19         found[u] = True
20         for w in range(vsize):
21             if not found[w]:
22                 if  dist[u] + adj[u][w] < dist[w]:
23                     dist[w] = dist[u] + adj[u][w]
24                     path[w] = u
25     return path
```

```
27 INF = float('inf')
28 vertex =   ['A', 'B', 'C', 'D', 'E', 'F', 'G']
29 adj_matrix = [ [0, 3, 7, 5, INF, INF, INF],
30                [3, 0, 2, INF, 9, INF, INF],
31                [7, 2, 0, 1, 6, 12, 25],
32                [5, INF, 1, 0, INF, INF, 20],
33                [INF, 9, 6, INF, 0, 4, INF],
34                [INF, INF, 12, INF, 4, 0, 8],
35                [INF, INF, 25, 20, INF, 8, 0]]
36
37 print("Shortest Path by Dijkstra's Algorithm")
38 start = 0
39 path = dijkstra(vertex, adj_matrix, start)
40
41 for end in range(len(vertex)):
42     if end != start:
43         print("[Shortest Path: %s->%s] %s" %
44                 (vertex[start], vertex[end], vertex[end]), end='')
45         while (path[end] != start):
46             print(" <- %s" % vertex[path[end]], end='')
47             end = path[end]
48         print(" <- %s" % vertex[path[end]])
```

- 라인 41-48: 각 정점마다 시작 정점으로부터의 최단 경로 출력(Note: path는 각 정점의 바로 이전 정점 인덱스를 저장하므로 바로 이전 정점을 따라 시작 정점까지 가는 경로가 최단 경로임)

```
Shortest Path by Dijkstra's Algorithm
[0, 0, 1, 0, 2, 4, 5]
[Shortest Path: A->B] B <- A
[Shortest Path: A->C] C <- B <- A
[Shortest Path: A->D] D <- A
[Shortest Path: A->E] E <- C <- B <- A
[Shortest Path: A->F] F <- E <- C <- B <- A
[Shortest Path: A->G] G <- F <- E <- C <- B <- A
```

수행시간 분석

입력 사이즈(그래프 정점의 수)를 n이라고 한다면,
- 라인 17의 주 반복문: n 번 반복
- choose_min_vertex 함수와 라인 20-24의 내부 반복문: 2n 번 반복

따라서 $O(n^2)$

참고: 탐욕적 알고리즘 – 최단 경로 문제(11/11)

❑ 다익스트라 알고리즘 정확성 검증

- **정리 1(Theorem 1): 다익스트라 알고리즘은 시작 정점 s로부터 그래프의 V에 속한 각 정점 v까지의 최단 경로를 구할 수 있다.**
  - **증명(Proof)**
    - 특정 단계(혹은 특점 시점)에서 집합 S에 v가 새로 추가될 차례라고 가정하자. 그러면 반드시 S에 이미 속해 있던 정점 u (i.e., $u \in S$)와 v (i.e., $v \in V-S$) 를 연결하는 간선 $e_1 = (u, v)$가 존재한다. 이 때 dist(s, u) + dist(u, v)가 s로부터 v까지의 최소 거리가 됨을 증명하면 된다. 모순을 통한 증명(proof by contradiction)을 위해 거리가 dist(s, u) + dist(u, v)보다 가까운 s로부터 v까지의 경로 P가 존재한다고 가정하자. u는 S에 포함되고 (i.e., $u \in S$) w는 S에 추가되지 않았으므로 (i.e., $v \notin S$), P에는 반드시 S에 포함된 정점 x와 S에 포함되지 안은 정점 y를 연결하는 간선 $e_2 = (x, y)$가 존재한다. s로부터 v까지의 최소 거리는 dist(s, u) + dist(u, v)보다 가까우므로 dist(s, x) + dist(x, y) < dist(s, u) + dist(u, v)를 만족하고, 그 결과 dist(s, y) < dist(s, v)를 만족한다. 하지만 다익스트라 알고리즘은 각 단계마다 choose_min_vertex() 함수를 호출하여 S에 추가되지 않은 정점 중 s와의 거리가 최소인 정점을 선택하므로 모순이다. 따라서 dist(s, u) + dist(u, v)보다 가까운 s로부터 v까지의 경로 P는 존재하지 않는다. □