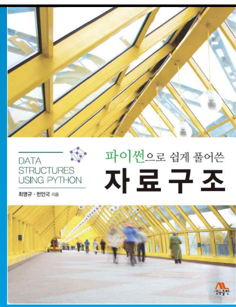

파이썬
자료구조



11 CHAPTER

가중치 그래프

11장. 학습 목표

- 가중치 그래프의 개념을 이해한다.
- 가중치 그래프를 표현하는 방법을 이해한다.
- 최소 비용 신장 트리 알고리즘을 이해한다.
- 최단 경로 알고리즘을 이해한다.
- 가중치 그래프를 이용한 문제해결 능력을 배양한다.

11.1 가중치 그래프란?

- 가중치 그래프란?

가중치 그래프란?

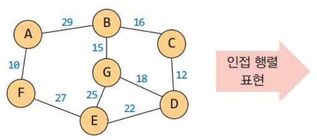
- 가중치 그래프(Weighted Graph)
  - 간선에 가중치가 할당된 그래프
  - G = (V, E, w)
    - w: 비용, 가중치(weight), 길이
- 경로 p의 길이

$$w(p)=\sum_{i=1}^{k} w(v_{i-1},\ v_i)$$

11.2 가중치 그래프의 표현

- 인접 행렬을 이용한 표현
- 인접 리스트를 이용한 표현

인접 행렬을 이용한 표현



	A	B	C	D	E	F	G
A	0	29	∞	∞	∞	10	∞
B	29	0	16	∞	∞	∞	15
C	∞	16	0	12	∞	∞	∞
D	∞	∞	12	0	22	∞	18
E	∞	∞	∞	22	0	27	25
F	10	∞	∞	∞	27	0	∞
G	∞	15	∞	18	25	∞	0

- 2차원 배열 → 파이썬: 리스트의 리스트

```
vertex =   ['A',    'B',   'C',   'D',   'E',   'F',   'G' ]
weight = [ [None,   29,    None,  None,  None,  10,    None],
           [29,     None,  16,    None,  None,  None,  15  ],
           [None,   16,    None,  12,    None,  None,  None],
           [None,   None,  12,    None,  22,    None,  18  ],
           [None,   None,  None,  22,    None,  27,    25  ],
           [10,     None,  None,  None,  27,    None,  None],
           [None,   15,    None,  18,    25,    None,  None]]
graph = (vertex, weight)              # 전체 그래프: 튜플 사용
```

인접 행렬: 간단한 연산

- 예) 인접 행렬에서의 가중치의 합 계산

```
def weightSum( vlist, W ):                  # 매개변수: 정점 리스트, 인접 행렬
    sum = 0                                 # 가중치의 합
    for i in range(len(vlist)) :            # 모든 정점에 대해(i: 0, ... N-1)
        for j in range(i+1, len(vlist)) :   # 하나의 행에 대해 (삼각영역)
            if W[i][j] != None :            # 만약 간선이 있으면
                sum += W[i][j]              # sum에 추가
    return sum                              # 전체 가중치 합을 반환
```

```
print('AM : weight sum = ', weightSum(vertex, weight))
```

```
C:\WINDOWS\system32\cmd.exe
AM : weight sum =  174
```

파이썬으로 쉽게 풀어쓴 자료구조

7

인접 행렬: 간단한 연산

- 예) 인접 행렬에서의 모든 간선 출력

```
def printAllEdges(vlist, W ):                        # 매개변수: 정점 리스트, 인접 행렬
    for i in range(len(vlist)) :
        for j in range(i+1, len(W[i])) :             # 모든 간선 W[i][j]에 대해
            if W[i][j] != None and W[i][j] != 0 :    # 간선이 있으면
                print("(%s,%s,%d)"%(vlist[i],vlist[j],W[i][j]), end=' ')
    print()
```

```
printAllEdges(vertex, weight)
```

```
C:\WINDOWS\system32\cmd.exe
(A,B,29) (A,F,10) (B,C,16) (B,G,15) (C,D,12) (D,E,22) (D,G,18) (E,F,27) (E,G,25)
```

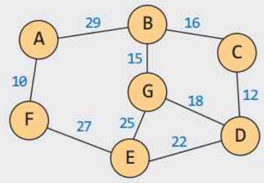
파이썬으로 쉽게 풀어쓴 자료구조

8

인접 리스트를 이용한 표현

- 예) 딕셔너리, 집합, 튜플, 리스트를 이용한 그래프 표현

```
graph ={'A' : set([('B',29),('F',10)        ]),
        'B' : set([('A',29),('C',16), ('G',15)]),
        'C' : set([('B',16),('D',12)        ]),
        'D' : set([('C',12),('E',22), ('G',18)]),
        'E' : set([('D',22),('F',27), ('G',25)]),
        'F' : set([('A',10),('E',27)        ]),
        'G' : set([('B',15),('D',18), ('E',25)]) }
```



- 예) 인접 리스트에서의 가중치의 합 계산

```
def weightSum(graph):              # 가중치의 총 합을 구하는 함수
    sum = 0
    for v in graph:                # 그래프의 모든 정점 v에 대해: 'A', 'B', ...
        for e in graph[v]:         # v의 모든 간선 e에 대해: ('B', 29), ...
            sum += e[1]            # sum에 추가
    return sum//2                  # 하나의 간선이 두 번 더해지므로 2로 나눔
```


인접 리스트

- 예) 인접 리스트에서의 모든 간선 출력

```
def printAllEdges(graph):          # 모든 간선을 출력하는 함수
    for v in graph:                # 그래프의 모든 정점 v에 대해: 'A', 'B', ...
        for e in graph[v]:         # v의 모든 간선 e에 대해: ('B', 29), ...
            print("(%s,%s,%d)"%(v,e[0],e[1]), end=' ')
```

- 예) 테스트 프로그램 (간선은 이중 출력)

```
print('AL : weight sum = ', weightSum(graphAL))
printAllEdges(graphAL)
```

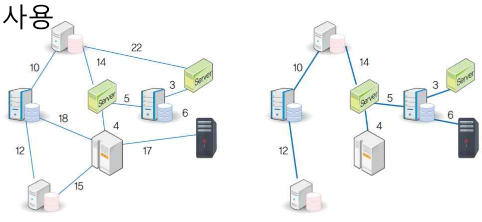
```
C:\WINDOWS\system32\cmd.exe
AL : weight sum =  174
(A,F,10) (A,B,29) (B,A,29) (B,G,15) (B,C,16) (C,B,16) (C,D,12) (D,C,12) (D,G,18)
(D,E,22) (E,F,27) (E,D,22) (E,G,25) (F,A,10) (F,E,27) (G,B,15) (G,E,25) (G,D,18)
```

11.3 최소비용 신장트리

- 최소비용 신장트리란?
- Kruskal의 MST 알고리즘
  - union-find 알고리즘
- Prim의 MST 알고리즘
- MST 알고리즘 시간 복잡도

최소비용 신장트리란?

- 간선들의 가중치 합이 최소인 신장 트리
  - 반드시 (n-1)개의 간선만 사용
  - 사이클이 포함되면 안됨



(a) 사이트 사이의 연결 비용 (b) 최소비용 신장트리의 예

- MST의 응용
  - 도로, 통신, 배관 건설: 모두 연결하면서 길이/비용을 최소화
  - 전기 회로: 단자를 모두 연결하면서 전선의 길이를 최소화
- Kruskal 알고리즘
- Prim 알고리즘

Kruskal의 MST 알고리즘

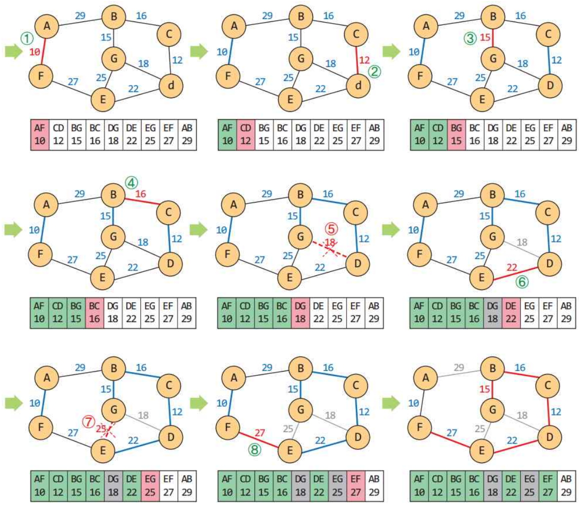
- 탐욕적인 방법(greedy method)
  - "그 순간에 최적"이라고 생각되는 것을 선택
  - 각 단계에서 최선의 답을 선택 → 최종적인 해답에 도달
  - 항상 최적의 해답을 주는지 검증 필요함
    - Kruskal MST 알고리즘은 최적의 해답임이 증명됨

Kruskal의 알고리즘

알고리즘 11.1 Kruskal의 최소 비용 신장 트리 알고리즘

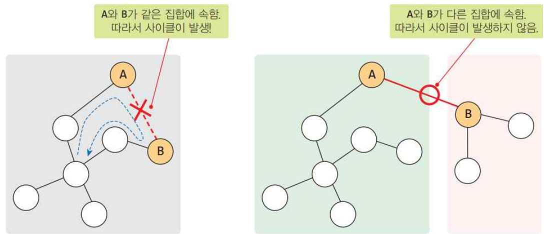
```
kruskal()
1. 그래프의 모든 간선을 가중치에 따라 오름차순으로 정렬한다.
2. 가장 가중치가 작은 간선 e를 뽑는다.
3. e를 신장트리에 넣었을 때 사이클이 생기면 넣지 않고 2번으로 이동한다.
4. 사이클이 생기지 않으면 최소 신장 트리에 삽입한다.
5. n-1개의 간선이 삽입될 때 까지 2번으로 이동한다.
```

Kruskal의 알고리즘 과정



union-find 알고리즘

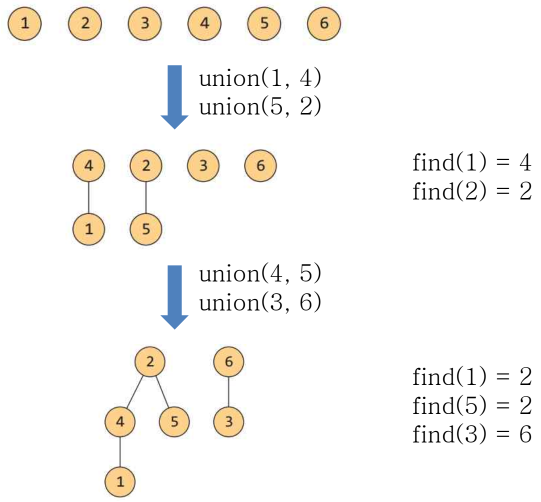
- 간선 추가시 사이클 검사 과정이 필요



➔ union-find 알고리즘 (트리를 이용해 구현)
- union: 두 집합의 합집합을 만드는 연산
- find: 원소가 속한 집합을 찾는 연산
- 예: union(4, 5}와 union(3, 6) 처리

{1, 4}, {5, 2}, {3}, {6} ➔ {1, 4, 5, 2}, {3, 6}

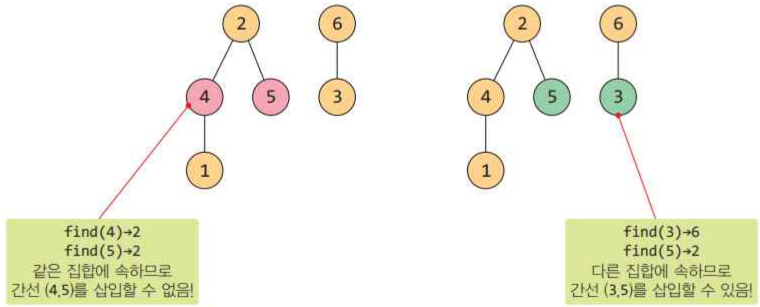
union과 find 연산 예



union-find 구현

```
parent = []                     # 각 노드의 부모노드 인덱스
set_size = 0                    # 전체 집합의 개수

def init_set(nSets) :               # 집합의 초기화 함수
    global set_size, parent         # 전역변수 사용(변경)을 위함
    set_size = nSets;               # 집합의 개수
    for i in range(nSets):          # 모든 집합에 대해
        parent.append(-1)           # 각각이 고유의 집합(부모가 -1)

def find(id) :                      # 정점 id가 속한 집합의 대표번호 탐색
    while (parent[id] >= 0) :       # 부모가 있는 동안(-1이 아닌 동안)
        id = parent[id]             # id를 부모 id로 갱신
    return id;                      # 최종 id 반환. 트리의 맨 위 노드의 id임

def union(s1, s2) :                 # 두 집합을 병합(s1을 s2에 병합시킴)
    global set_size                 # 전역변수 사용(변경)을 위함
    parent[s1] = s2                 # s1을 s2에 병합시킴
    set_size = set_size - 1         # 집합의 개수가 줄어 듦
```

find()를 이용한 간선의 사이클 검사



Kruskal의 MST 알고리즘

```
def MSTKruskal(vertex, adj):                    # 매개변수: 정점리스트, 인접행렬
    vsize = len(vertex)                         # 정점의 개수
    init_set(vsize)                             # 정점 집합 초기화
    eList = []                                  # 간선 리스트

    for i in range(vsize-1) :                   # 모든 간선을 리스트에 넣음
        for j in range(i+1, vsize) :
            if adj[i][j] != None :
                eList.append( (i,j,adj[i][j]) )  # 튜플로 저장

    # 간선 리스트를 가중치의 내림차순으로 정렬: 람다 함수 사용
    eList.sort(key= lambda e : e[2], reverse=True)

    edgeAccepted = 0
    while (edgeAccepted < vsize - 1) :          # 정점 수 - 1개의 간선
        e = eList.pop(-1)                       # 가장 작은 가중치를 가진 간선
        uset = find(e[0])                       # 두 정점이 속한 집합 번호
        vset = find(e[1])

        if uset != vset :                       # 두 정점이 다른 집합의 원소이면
            print("간선 추가 : (%s, %s, %d)" %   # 간선추가 출력
                (vertex[e[0]], vertex[e[1]], e[2]))
            union(uset, vset)                   # 두 집합을 합함
            edgeAccepted += 1                   # 간선이 하나 추가됨
```

테스트 프로그램

```
vertex =  ['A',    'B',    'C',    'D',    'E',    'F',    'G' ]
weight = [ [None,  29,     None,   None,   None,   10,     None],
           [29,    None,   16,     None,   None,   None,   15 ],
           [None,  16,     None,   12,     None,   None,   None],
           [None,  None,   12,     None,   22,     None,   18 ],
           [None,  None,   None,   22,     None,   27,     25 ],
           [10,    None,   None,   None,   27,     None,   None],
           [None,  15,     None,   18,     25,     None,   None]]

print("MST By Kruskal's Algorithm")
MSTKruskal(vertex, weight)
```

```
C:\WINDOWS\system32\cmd.exe
MST By Kruskal's Algorithm
간선 추가 : (A, F, 10)
간선 추가 : (C, D, 12)
간선 추가 : (B, G, 15)
간선 추가 : (B, C, 16)
간선 추가 : (D, E, 22)
간선 추가 : (E, F, 27)
```

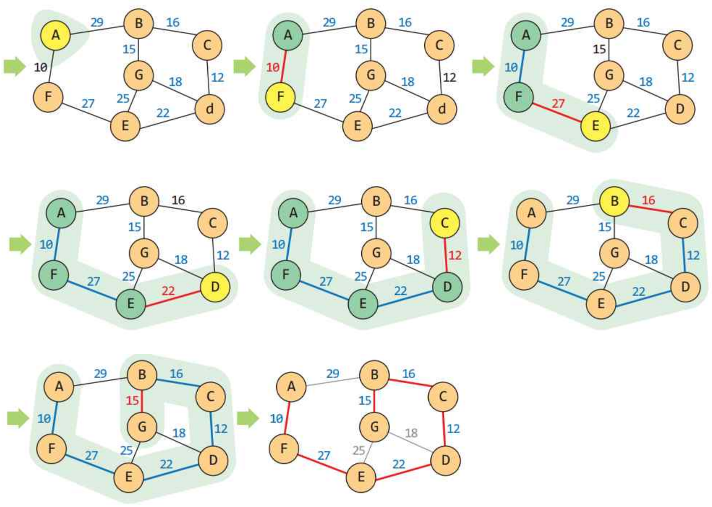

Prim의 MST 알고리즘

- 하나의 정점에서부터 시작하여 트리를 단계적으로 확장
  - 현재의 신장 트리 집합에 인접한 정점 중 최저 간선으로 연결된 정점 선택하여 신장 트리 집합에 추가
  - 이 과정을 n-1개의 간선을 가질 때까지 반복

알고리즘 11.2 Prim의 최소비용 신장트리 알고리즘

```
Prim()
1. 그래프에서 시작 정점을 선택하여 초기 트리를 만든다.
2. 현재 트리의 정점들과 인접한 정점들 중에서 간선의 가중치가 가장 작은 정점 v를 선택한다.
3. 이 정점 v와 이때의 간선을 트리에 추가한다.
4. 모든 정점이 삽입될 때 까지 2번으로 이동한다.
```

Prim의 MST



Prim의 알고리즘

```
def MSTPrim(vertex, adj) :
    vsize = len(vertex)
    dist = [INF] * vsize                # dist: [INF, INF, ... INF]
    selected = [False] * vsize          # selected: [False, False, ... False]
    dist[0] = 0                         # dist: [0, INF, ... INF]

    for i in range(vsize) :             # 정점의 수 만큼 반복
        u = getMinVertex(dist, selected)
        selected[u] = True              # u는 이제 선택됨
        print(vertex[u], end=' ')       # u를 출력
        for v in range(vsize) :         # 내부 루프
            if (adj[u][v] != None):     # (u,v) 간선이 있으면 dist[v] 갱신
                if selected[v]==False and adj[u][v]< dist[v] :
                    dist[v] = adj[u][v]
    print()
```

```
C:\WINDOWS\system32\cmd.exe
MST By Prim's Algorithm
A F E D C B G
```

MST 알고리즘 시간 복잡도

- Kruskal 알고리즘: $O(e \log e)$
  - 대부분 간선들을 정렬하는 시간에 좌우됨
  - 간선 e개를 정렬하는 시간
- Prim의 알고리즘: $O(n^2)$
  - 주 반복문이 n번, 내부 반복문이 n번 반복

- 희박한 그래프
  - $O(e \log e)$인 Kruskal이 유리
- 밀집한 그래프: 예) 완전그래프
  - $O(n^2)$ 인 Prim이 유리

11.4 최단 경로

- 최단 경로 알고리즘이란?
- 최단 경로를 위한 그래프 표현
- Dijkstra의 최단 경로 알고리즘
  - 최단 경로 증명
  - dist 갱신
- Floyd의 최단 경로 알고리즘
  - 알고리즘 아이디어
- 최단 경로 알고리즘 시간 복잡도

최단 경로 알고리즘이란?

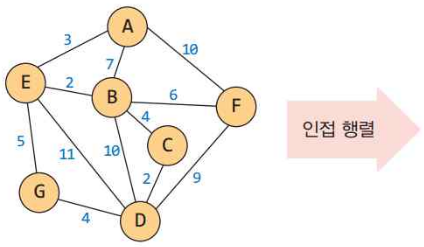
- 정점 u와 정점 v를 연결하는 경로 중에서 간선들의 가중치 합이 최소가 되는 경로
  – 간선의 가중치는 비용, 거리, 시간 등

- Dijkstra, Floyd 알고리즘

시험

최단 경로를 위한 그래프 표현

- 간선이 없으면 가중치를 무한대로 처리

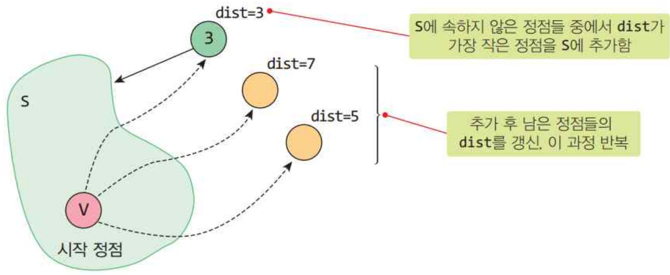


	A	B	C	D	E	F	G
A	0	7	∞	∞	3	10	∞
B	7	0	4	10	2	6	∞
C	∞	4	0	2	∞	∞	∞
D	∞	10	2	0	11	9	4
E	3	2	∞	11	0	∞	5
F	10	6	∞	9	∞	0	∞
G	∞	∞	∞	4	5	∞	0

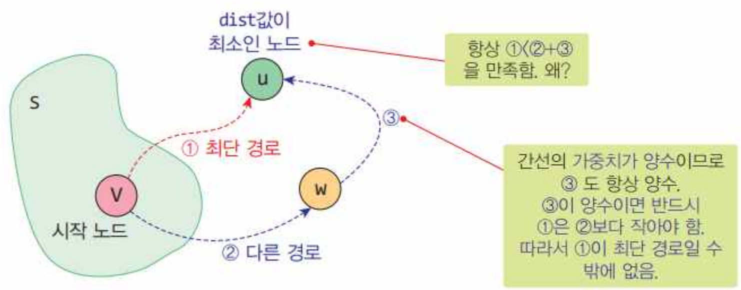
간선이 없으면 무한대(∞)

Dijkstra의 최단 경로 알고리즘

- 시작 정점 v에서 모든 다른 정점까지의 최단 경로 찾음
  - 시작 정점 v: 최단경로탐색의 시작 정점
  - 집합 S: 시작 정점 v로부터의 최단경로가 이미 발견된 정점들의 집합
  - dist배열: S에 있는 정점만을 거쳐서 다른 정점으로 가는 최단거리를 기록하는 배열.
- 매 단계에서 최소 distance인 정점을 S에 추가

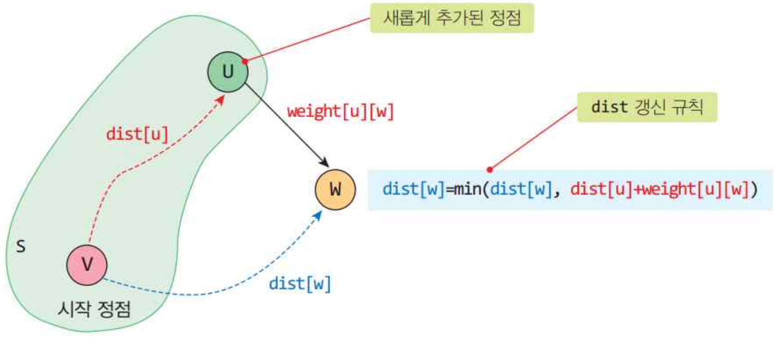


최단 경로 증명



dist 갱신

- 새로운 정점이 S에 추가되면 dist 갱신

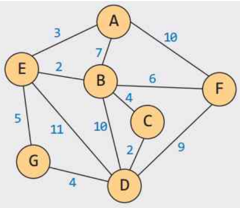


Dijkstra의 최단경로 알고리즘

```
def shortest_path_dijkstra(vtx, adj, start) :
    vsize = len(vtx)                              # 정점 수
    dist = list(adj[start])                       # dist 배열 생성 및 초기화
    path = [start] * vsize                        # path 배열 생성 및 초기화
    found= [False] * vsize                        # found 배열 생성 및 초기화
    found[start] = True                           # 시작정점: 이미 찾아짐
    dist[start] = 0                               # 시작정점의 거리 0

    for i in range(vsize) :
        print("Step%2d: "%(i+1), dist)            # 단계별 dist[] 출력용
        u = choose_vertex(dist, found)            # 최소 dist 정점 u 탐색
        found[u] = True                           # u는 이제 찾아짐

        for w in range(vsize) :                   # 모든 정점에 대해
            if not found[w] :                     # 아직 찾아지지 않았으면
                if dist[u] + adj[u][w] < dist[w] :   # 갱신 조건 검사
                    dist[w] = dist[u] + adj[u][w]    # dist 갱신
                    path[w] = u                      # 이전 정점 갱신

    return path                                   # 찾아진 최단 경로 반환
```

테스트 프로그램

```python
print("Shortest Path By Dijkstra Algorithm")
start = 0       # 시작 정점은 0번, 'A'로 선택
path = shortest_path_dijkstra(vertex, weight, start)

# 최종 경로를 출력하기 위한 코드
for end in range(len(vertex)) :
    if end != start :
        print("[최단경로: %s->%s] %s" %
               (vertex[start], vertex[end], vertex[end]), end='')
        while (path[end] != start) :
            print(" <- %s" % vertex[path[end]], end='')
            end = path[end]
        print(" <- %s" % vertex[path[end]])
```



C:\WINDOWS\system32\cmd.exe

```
Shortest Path By Dijkstra Algorithm
Step 1:  [0, 7, 999, 999, 3, 10, 999]
Step 2:  [0, 5, 999, 14, 3, 10, 8]
Step 3:  [0, 5, 9, 14, 3, 10, 8]
Step 4:  [0, 5, 9, 12, 3, 10, 8]
Step 5:  [0, 5, 9, 11, 3, 10, 8]
Step 6:  [0, 5, 9, 11, 3, 10, 8]
Step 7:  [0, 5, 9, 11, 3, 10, 8]
[최단경로: A->B] B <- E <- A
[최단경로: A->C] C <- B <- E <- A
[최단경로: A->D] D <- C <- B <- E <- A
[최단경로: A->E] E <- A
[최단경로: A->F] F <- A
[최단경로: A->G] G <- E <- A
```

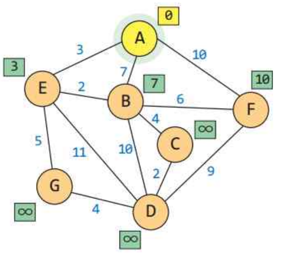
각 단계별 dist[] 배열의 값 변화

A부터 모든 정점까지의 최단 경로

파이썬으로 쉽게 풀어쓴 자료구조

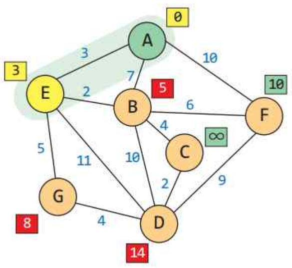
33

알고리즘 실행 과정: Step1-2



S = { A }

dist(A)=w(A,A)=0
dist(B)=w(A,B)=7
dist(C)=w(A,C)=∞
dist(D)=w(A,D)=∞
dist(E)=w(A,E)=3
dist(F)=w(A,F)=10
dist(G)=w(A,G)=∞



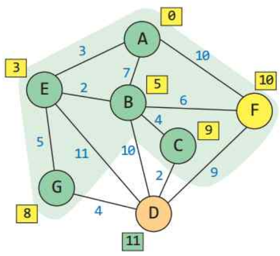
S = { A, E }

dist(A)=0
dist(E)=w(A,E)=3
dist(B)=min(dist(B),dist(E)+w(E,B))=min(7,3+2)=5
dist(C)=min(dist(C),dist(E)+w(E,C))=min(∞,3+∞)=∞
dist(D)=min(dist(D),dist(E)+w(E,D))=min(∞,3+11)=14
dist(F)=min(dist(F),dist(E)+w(E,F))=min(10,3+∞)=10
dist(G)=min(dist(G),dist(E)+w(E,G))=min(∞,3+5)=8

파이썬으로 쉽게 풀어쓴 자료구조

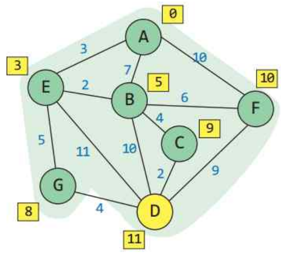
34

알고리즘 실행 과정: Step6-최종



S = { A, E, B, G, C, F }

```
dist(A)=0
dist(B)=5
dist(C)=9
dist(E)=3
dist(F)=10
dist(G)=8
dist(D)=min(dist(D),dist(F)+w(F,D))=min(11,10+9)=11
```



S = { A, E, B, G, C, F, D }

```
dist(A)=0
dist(B)=5
dist(C)=9
dist(D)=11
dist(E)=3
dist(F)=10
dist(G)=8
```


Floyd의 최단경로 알고리즘

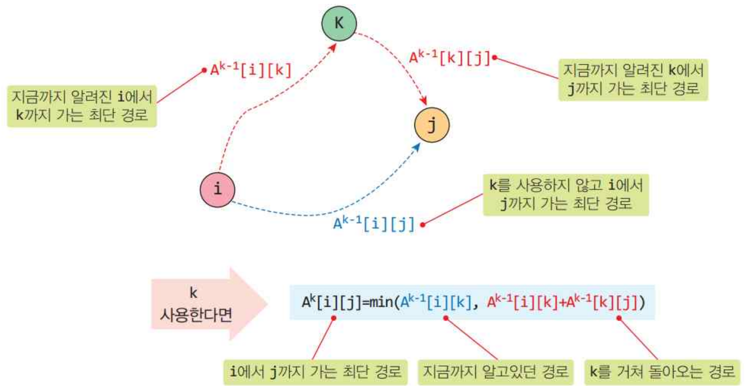
- 모든 정점 사이의 최단경로를 찾음
  - 2차원 배열 A를 이용하여 3중 반복을 하는 루프로 구성
  - 배열 A의 초기 값은 인접 행렬의 가중치

알고리즘 11.4 Floyd-Warshall의 최단 경로 알고리즘

```
shortest_path_floyd()
for k ← 0 to n - 1
   for i ← 0 to  n - 1
      for j ← 0 to n - 1
         A[i][j] = min(A[i][j], A[i][k] + A[k][j])
```

알고리즘 아이디어

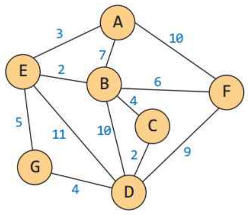
- $A^k[i][j]$
  - 0~k까지의 정점만을 이용한 정점 i에서 j까지의 최단 경로 길이
- $A^{-1} \rightarrow A^0 \rightarrow A^1 \rightarrow \ldots \rightarrow A^{n-1}$ 순으로 최단경로길이를 구함



Floyd의 최단 경로 알고리즘

```
def shortest_path_floyd(vertex, adj) :        # Floyd의 최단경로탐색 함수.
    vsize = len(vertex)                       # 정점의 개수
    A = list(adj)                             # 주의: 2차원 배열(리스트의 리스트)의 복사
    for i in range(vsize) :                   # 각각의 열에 대해
        A[i] = list(adj[i])                   # 열(리스트)을 복사

    for k in range(vsize) :                   # 정점 k를 추가할 때
        for i in range(vsize) :
            for j in range(vsize) :           # 모든 A[i][j] 갱신
                if (A[i][k] + A[k][j] < A[i][j]) :
                    A[i][j] = A[i][k] + A[k][j]
        printA(A)                             # 현재 A 행렬 출력
```

실행 결과




```
C:₩WINDOWS₩system32₩cmd.exe
Shortest Path By Floyd's Algorithm
==========================================
   0    7  INF  INF    3   10  INF
   7    0    4   10    2    6  INF
 INF    4    0    2  INF  INF  INF
 INF   10    2    0   11    9    4
   3    2  INF   11    0   13    5
  10    6  INF    9   13    0  INF
 INF  INF  INF    4    5  INF    0
==========================================
   0    7   11   17    3   10  INF
   7    0    4   10    2    6  INF
  11    4    0    2    6   10  INF
  17   10    2    0   11    9    4
   3    2    6   11    0    8    5
  10    6   10    9    8    0  INF
 INF  INF  INF    4    5  INF    0
==========================================
   0    7   11   13    3   10  INF
   7    0    4    6    2    6  INF
  11    4    0    2    6   10  INF
  13    6    2    0    8    9    4
   3    2    6    8    0    8    5
  10    6   10    9    8    0  INF
 INF  INF  INF    4    5  INF    0
==========================================
   0    7   11   13    3   10   17
   7    0    4    6    2    6   10
  11    4    0    2    6   10    6
  13    6    2    0    8    9    4
   3    2    6    8    0    8    5
  10    6   10    9    8    0   13
  17   10    6    4    5   13    0
```

```
==========================================
   0    5    9   11    3   10    8
   5    0    4    6    2    6    7
   9    4    0    2    6   10    6
  11    6    2    0    8    9    4
   3    2    6    8    0    8    5
  10    6   10    9    8    0   13
   8    7    6    4    5   13    0
==========================================
   0    5    9   11    3   10    8
   5    0    4    6    2    6    7
   9    4    0    2    6   10    6
  11    6    2    0    8    9
   3    2    6    8    0    8
  10    6   10    9    8    0
   8    7    6    4    5   13
==========================================
   0    5    9   11    3   10    8
   5    0    4    6    2    6    7
   9    4    0    2    6   10    6
  11    6    2    0    8    9    4
   3    2    6    8    0    8    5
  10    6   10    9    8    0   13
   8    7    6    4    5   13    0
계속하려면 아무 키나 누르십시오 . . .
```

Dijkstra 알고리즘(시작정점0)의 결과와 동일

최종 A행렬

최단 경로 알고리즘 시간 복잡도

- Dijkstra: $O(n^2)$
  - 주 반복문을 n번 반복
  - 내부 반복문을 2n번 반복
  - 모든 정점 쌍의 최단 경로를 구하려면 n번 반복 → $O(n^3)$
- Floyd-Warshall: $O(n^3)$
  - 모든 정점 쌍의 최단 경로 거리를 구함
  - 3중 반복문을 실행
  - Floyd의 알고리즘은 매우 간결한 반복 구문을 사용

11장 연습문제, 실습문제

파이썬으로 쉽게 풀어쓴
자료구조

41

감사합니다!

파이썬으로 쉽게 풀어쓴
자료구조

42

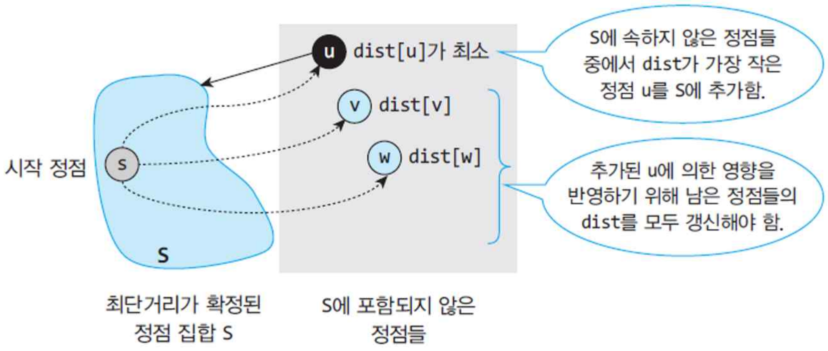
2. Dijkstra의 최단 경로 알고리즘

- 학습 목표
  - Dijkstra의 최단경로 알고리즘을 설명할 수 있다.
  - Dijkstra의 최단경로 알고리즘을 구현할 수 있다.
- 학습 내용
  - Dijkstra의 전략
  - Dijkstra 알고리즘

1. Dijkstra의 전략

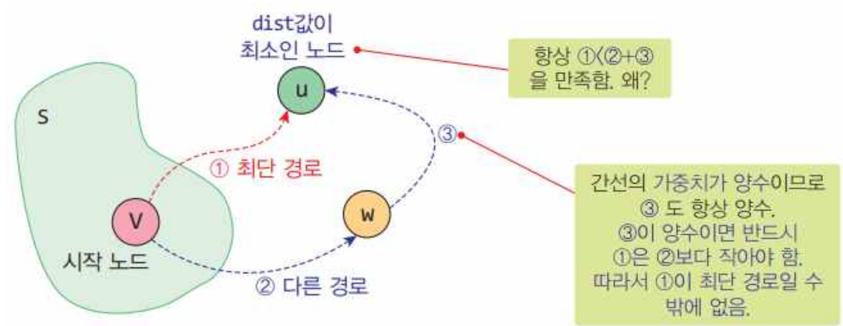
- 한 정점에서 다른 모든 정점까지의 최단 경로 거리 계산
- Dijkstra의 전략
  - 탐욕적 기법 사용
  - 최단거리가 확정된 정점들과 간선으로 직접 연결된 정점들 중에서 가장 가까운 정점 u를 선택
  - u까지의 거리 확정
  - u를 제외한 남은 정점들의 거리를 갱신

Dijkstra의 전략



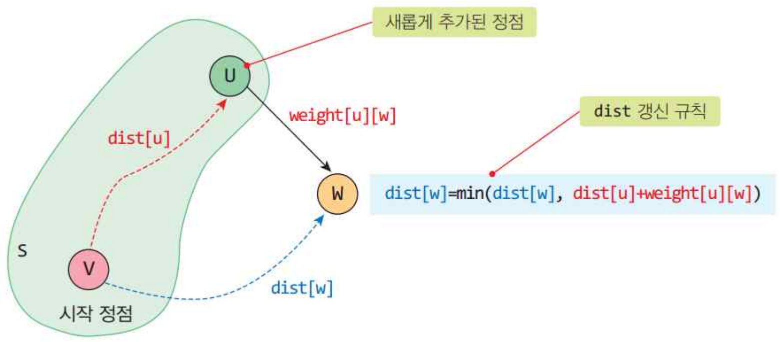
최단 경로 증명

- dist[u]가 최소라면 v→u의 거리를 확정할 수 있을까?
  - v→ … → w → u 가 최소일 수는 없을까?
  - 간선의 가중치가 양수라면 → 성립함



dist 갱신

- 새로운 정점이 S에 추가되면 dist 갱신



파이썬으로 쉽게 풀어쓴 자료구조

47

2. Dijkstra 알고리즘

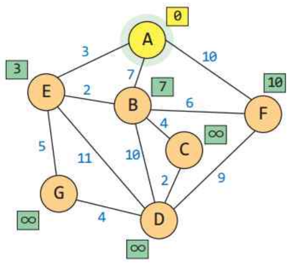
- 필요한 자료
  - dist[]: 시작 정점으로 부터의 거리
    - 최초: 시작 정점과의 간선의 가중치
  - found[]: 거리가 확정되었는지 여부
    - 최초: 시작 정점만 True, 나머지는 False
  - path[]: 이전 정점의 인덱스

- dist[]가 최소인 정점을 선택해 최단 거리를 확정하는 과정을 반복

파이썬으로 쉽게 풀어쓴 자료구조

48

알고리즘 실행 과정: Step1

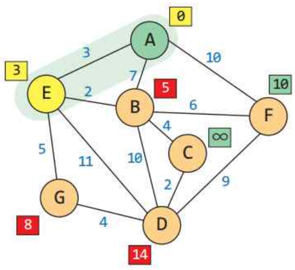


S = { A }

dist(A)=w(A,A)=0
dist(B)=w(A,B)=7
dist(C)=w(A,C)=∞
dist(D)=w(A,D)=∞
dist(E)=w(A,E)=3
dist(F)=w(A,F)=10
dist(G)=w(A,G)=∞

dist[]: 시작 정점으로 부터의 거리

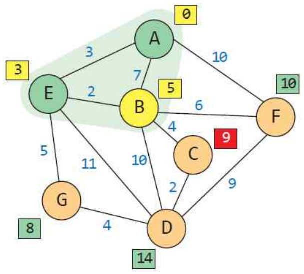
알고리즘 실행 과정: Step2



S = { A, E }

dist(A)=0
dist(E)=w(A,E)=3
dist(B)=min(dist(B),dist(E)+w(E,B))=min(7,3+2)=5
dist(C)=min(dist(C),dist(E)+w(E,C))=min(∞,3+∞)=∞
dist(D)=min(dist(D),dist(E)+w(E,D))=min(∞,3+11)=14
dist(F)=min(dist(F),dist(E)+w(E,F))=min(10,3+∞)=10
dist(G)=min(dist(G),dist(E)+w(E,G))=min(∞,3+5)=8

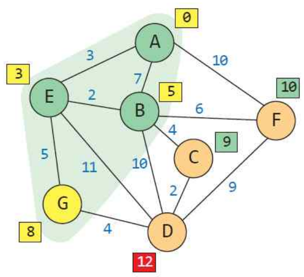
알고리즘 실행 과정: Step3



S = { A, E, B }

dist(A)=0
dist(B)=5
dist(E)=3
dist(C)=min(dist(C),dist(B)+w(B,C))=min(∞,5+4)=9
dist(D)=min(dist(D),dist(B)+w(B,D))=min(14,5+10)=14
dist(F)=min(dist(F),dist(B)+w(B,F))=min(10,5+6)=10
dist(G)=min(dist(G),dist(B)+w(B,G))=min(8,5+∞)=8

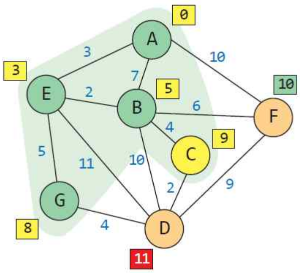
알고리즘 실행 과정: Step4



S = { A, E, B, G }

dist(A)=0
dist(B)=5
dist(E)=3
dist(G)=8
dist(C)=min(dist(C),dist(G)+w(G,C))=min(9,8+∞)=9
dist(D)=min(dist(D),dist(G)+w(G,D))=min(14,8+4)=12
dist(F)=min(dist(F),dist(G)+w(G,F))=min(10,8+∞)=10

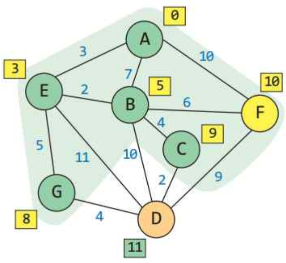
알고리즘 실행 과정: Step5




```
S = { A, E, B, G, C }

dist(A)=0
dist(B)=5
dist(C)=9
dist(E)=3
dist(G)=8
dist(D)=min(dist(D),dist(C)+w(C,D))=min(12,9+2)=11
dist(F)=min(dist(F),dist(C)+w(C,F))=min(10,9+∞)=10
```

파이썬으로 쉽게 풀어쓴
자료구조

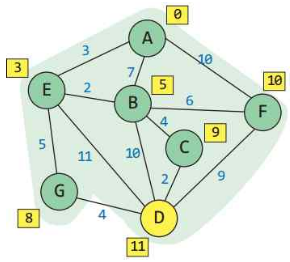
알고리즘 실행 과정: Step6




```
S = { A, E, B, G, C, F }

dist(A)=0
dist(B)=5
dist(C)=9
dist(E)=3
dist(F)=10
dist(G)=8
dist(D)=min(dist(D),dist(F)+w(F,D))=min(11,10+9)=11
```

파이썬으로 쉽게 풀어쓴
자료구조

알고리즘 실행 과정: Step7



S = { A, E, B, G, C, F, D }

dist(A)=0
dist(B)=5
dist(C)=9
dist(D)=11
dist(E)=3
dist(F)=10
dist(G)=8

Dijkstra 알고리즘

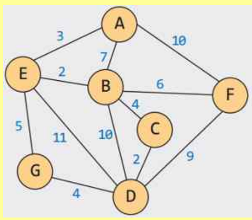
```
def shortest_path_dijkstra(vertex, adj, s) :
    vsize = len(vertex)
    dist = list(adj[s])                  # 시작정점과의 간선의 가중치(dist[s]=0)
    path = [s] * vsize                   # 일단 모두 시작 정점으로 초기화
    found= [False] * vsize               # 최단거리 확정 여부(False로 초기화)
    found[s] = True                      # 최초에 s만 True 나머지는 False

    for i in range(vsize) :              # 모든 정점을 포함해야 함
        print("Step%2d: "%(i+1), dist)  # 단계별 dist[] 출력용
        u = choose_vertex(dist, found)  # 최단 거리 정점 u
        found[u] = True                  # 이제 u까지의 거리는 확정됨

        for w in range(vsize) :          # dist[] 갱신
            if not found[w] :            # 아직 거리가 확정되지 않은 정점들에 대해
                if (dist[u] + adj[u][w] < dist[w]) :   # u를 거치면 가깝다면...
                    dist[w] = dist[u] + adj[u][w]      # dist[w] 갱신
                    path[w] = u                        # w의 이전 정점은 u

    return path                          # 경로 출력
```

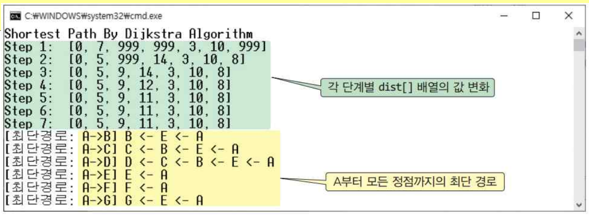
시간 복잡도

- Dijkstra: $O(n^2)$
  - 주 반복문을 n번 반복
  - 내부 반복문을 2n번 반복
  - 모든 정점 쌍의 최단 경로를 구하려면 n번 반복 → $O(n^3)$

테스트 프로그램

```
vertex =   [ 'A', 'B', 'C', 'D', 'E', 'F', 'G' ]
weight = [ [   0,   7, INF, INF,   3,  10, INF ],
           [   7,   0,   4,  10,   2,   6, INF ],
           [ INF,   4,   0,   2, INF, INF, INF ],
           [ INF,  10,   2,   0,  11,   9,   4 ],
           [   3,   2, INF,  11,   0,  13,   5 ],
           [  10,   6, INF,   9,  13,   0, INF ],
           [ INF, INF, INF,   4,   5, INF,   0 ]]

print("Shortest Path By Dijkstra Algorithm")
start = 0
path = shortest_path_dijkstra(vertex, weight, start)
...
```





실습 동영상: 24-2

- Dijkstra 알고리즘 테스트

3. Floyd의 최단 경로 알고리즘

- 학습 목표
  - Floyd 최단경로 알고리즘을 설명할 수 있다.
  - Floyd 최단경로 알고리즘을 구현할 수 있다.
- 학습 내용
  - Floyd의 전략
  - Floyd알고리즘

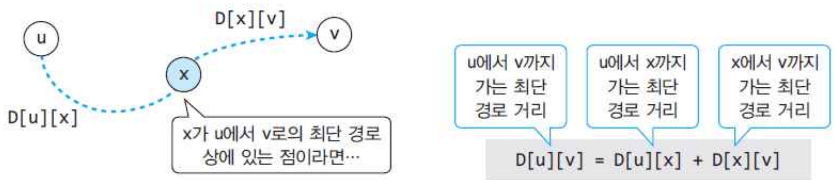
1. Floyd의 전략

- Floyd 알고리즘
  - 모든 정점 사이의 최단 경로 거리를 찾는 알고리즘
  - Floyd-Warshall 알고리즘 이라고도 함
  - 아이디어:
    - 모든 정점 사이의 최단 경로 거리를 구하려면 모든 정점을 거치는 상황을 고려해야 함.
    - 어떤 정점을 하나도 거치지 않는 (바로 가는) 경로에서 부터 시작
    - 정점을 하나씩 순차적으로 고려했을 때 경로를 갱신
    - 최종적으로 모든 정점을 고려한 경로 거리를 구함

  →동적 계획 (dynamic programming) 전략

Floyd의 전략

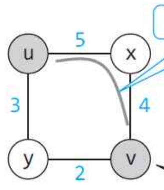
- 동적 계획 전략
  - 최적 부분 구조를 만족해야 함



[그림 7.20] 최단경로 문제는 최적 부분 구조 특성을 만족함

Floyd의 전략

- 최적 부분 구조를 만족하지 않는 예
  - 최장 경로 문제



u에서 v로의 최장 경로(u-x-v)의 길이 = 9

```
D[u][v] ≠ D[u][x] + D[x][v]
   9    ≠    9    +   10
```

u에서 x로의 최장 경로(u-y-v-x)의 길이 = 9
x에서 v로의 최장 경로(x-u-y-v)의 길이 = 10
부분 문제의 해답의 합 = 19

2. Floyd 알고리즘

- 인접 행렬 표현 그래프 $W \rightarrow D$
  - 최단거리 행렬 D
  - $D^k[i][j]$:
    - 1~k번째 정점까지 만을 이용한 정점 $i \rightarrow j$의 최단 경로 거리
  - 구하고자 하는 최적해: D^n
- 동적 계획 전략
  - D^0 □ D^1 □ D^2 □ ...□ D^n

Floyd 알고리즘

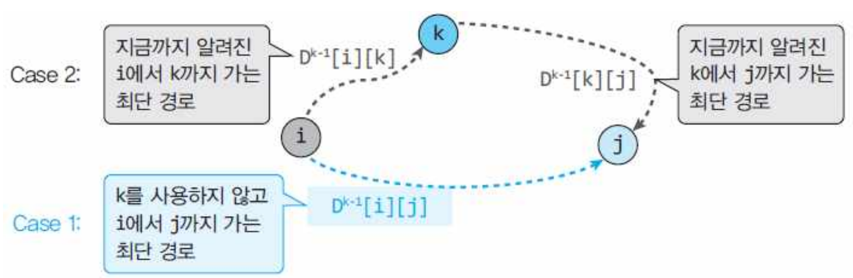
- 일반적인 경우
  - D^{k-1}□ 구해진 상태에서 k 번째 정점을 추가로 고려한 D^k를 구하려고 함.
  - 두 가지 상황:
    - Case1: k 를 거치지 않는 경로

$$D^k[i][j] \leftarrow D^{k-1}[i][j]$$

    - Case2: k 를 거치는 경로

$$D^k[i][j] \leftarrow D^{k-1}[i][k] + D^{k-1}[k][j]$$

Floyd 알고리즘



$$D^k[i][j] = \begin{cases} W[i][j] & k = 0 \\ \min\left(D^{k-1}[i][j],\ D^{k-1}[i][k] + D^{k-1}[k][j]\right) & \text{otherwise} \end{cases}$$

Floyd 알고리즘

- 기반 상황
  - A^0 : 아무런 정점을 거치지 않는 경로 거리

    → 인접 행렬 W
- 일반 상황
  - D^k : k□□□ 정점만을 이용한 부분 문제의 최적해
  - D^{k-1}□ 구해진 상태에서 k 번째 정점을 고려할때
    - 후보1: k 를 거치지 않는 경로

      $$D^k[i][j] \leftarrow D^{k-1}[i][j]$$
    - 후보2: k 를 거치는 경로

      $$D^k[i][j] \leftarrow D^{k-1}[i][k] + D^{k-1}[k][j]$$

Floyd 알고리즘

- 알고리즘(유사코드)
  - 2차원 배열 A를 이용하여 3중 반복을 하는 루프로 구성
  - 배열 A의 초기 값은 인접 행렬의 가중치

알고리즘 11.4 Floyd-Warshall의 최단 경로 알고리즘

```
shortest_path_floyd()
for k ← 0 to n - 1
   for i ← 0 to  n - 1
     for j ← 0 to n - 1
        A[i][j] = min(A[i][j], A[i][k] + A[k][j])
```

Floyd 알고리즘

```
def shortest_path_floyd(vertex, adj) :
    vsize = len(vertex)              # 정점의 개수

    A = list(adj)                    # 2차원 배열(리스트의 리스트)의 복사
    for i in range(vsize) :
        A[i] = list(adj[i])

    for k in range(vsize) :          # 모든 정점을 순서대로 고려함
        for i in range(vsize) :
            for j in range(vsize) :  # 정점 i에서 정점 j까지의 거리
                if (A[i][k] + A[k][j] < A[i][j]) :
                    A[i][j] = A[i][k] + A[k][j]
        printA(A)                    # 진행상황 출력용
```

파이썬으로 쉽게 풀어쓴 자료구조

69

시간 복잡도

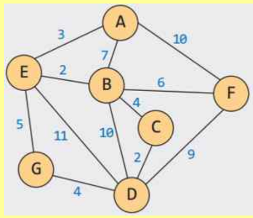
- Floyd-Warshall: $O(n^3)$
  - 모든 정점 쌍의 최단 경로 거리를 구함
  - 3중 반복문을 실행
  - Floyd의 알고리즘은 매우 간결한 반복 구문을 사용
- Dijkstra: $O(n^2)$
  - 모든 정점 쌍의 최단 경로를 구하려면 n번 반복 → $O(n^3)$

파이썬으로 쉽게 풀어쓴 자료구조

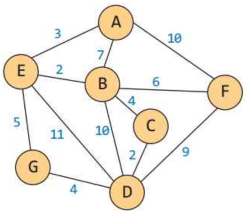
70

테스트 프로그램

```python
vertex =   [ 'A', 'B', 'C', 'D', 'E', 'F', 'G' ]
weight = [ [   0,   7, INF, INF,   3,  10, INF ],
           [   7,   0,   4,  10,   2,   6, INF ],
           [ INF,   4,   0,   2, INF, INF, INF ],
           [ INF,  10,   2,   0,  11,   9,   4 ],
           [   3,   2, INF,  11,   0,  13,   5 ],
           [  10,   6, INF,   9,  13,   0, INF ],
           [ INF, INF, INF,   4,   5, INF,   0 ]]


print("Shortest Path By Floyd's Algorithm")
shortest_path_floyd(vertex, weight)
```



실행 결과




```
C:\WINDOWS\system32\cmd.exe
Shortest Path By Floyd's Algorithm
=====================================
   0     7   INF   INF     3    10   INF
   7     0     4    10     2     6   INF
 INF     4     0     2   INF   INF   INF
 INF    10     2     0    11     9     4
   3     2   INF    11     0    13     5
  10     6   INF     9    13     0   INF
 INF   INF   INF     4     5   INF     0
=====================================
   0     7    11    17     3    10   INF
   7     0     4    10     2     6   INF
  11     4     0     2     6    10   INF
  17    10     2     0    11     9     4
   3     2     6    11     0     8     5
  10     6    10     9     8     0   INF
 INF   INF   INF     4     5   INF     0
=====================================
   0     7    11    13     3    10   INF
   7     0     4     6     2     6   INF
  11     4     0     2     6    10   INF
  13     6     2     0     8     9     4
   3     2     6     8     0     8     5
  10     6    10     9     8     0   INF
 INF   INF   INF     4     5   INF     0
=====================================
   0     7    11    13     3    10    17
   7     0     4     6     2     6    10
  11     4     0     2     6    10     6
  13     6     2     0     8     9     4
   3     2     6     8     0     8     5
  10     6    10     9     8     0    13
  17    10     6     4     5    13     0
```

```
=====================================
   0     5     9    11     3    10     8
   5     0     4     6     2     6     7
   9     4     0     2     6    10     6
  11     6     2     0     8     9     4
   3     2     6     8     0     8     5
  10     6    10     9     8     0    13
   8     7     6     4     5    13     0
=====================================
   0     5     9    11     3    10     8
   5     0     4     6     2     6     7
   9     4     0     2     6    10     6
  11     6     2     0     8     9  [illegible]
   3     2     6     8     0     8  [illegible]
  10     6    10     9     8     0  [illegible]
   8     7     6     4     5    13  [illegible]
=====================================
   0     5     9    11     3    10     8
   5     0     4     6     2     6     7
   9     4     0     2     6    10     6
  11     6     2     0     8     9     4
   3     2     6     8     0     8     5
  10     6    10     9     8     0    13
   8     7     6     4     5    13     0
계속하려면 아무 키나 누르십시오 . . .
```

Dijkstra 알고리즘(시작정점0)의 결과와 동일

최종 A행렬

실습 동영상: 24-3

- Floyd 알고리즘 테스트

파이썬으로 쉽게 풀어쓴
자료구조

73