[CSE3081(2반)] 알고리즘 설계와 분석

2020학년도 2학기

강의자료

(2020.12.08 화요일)

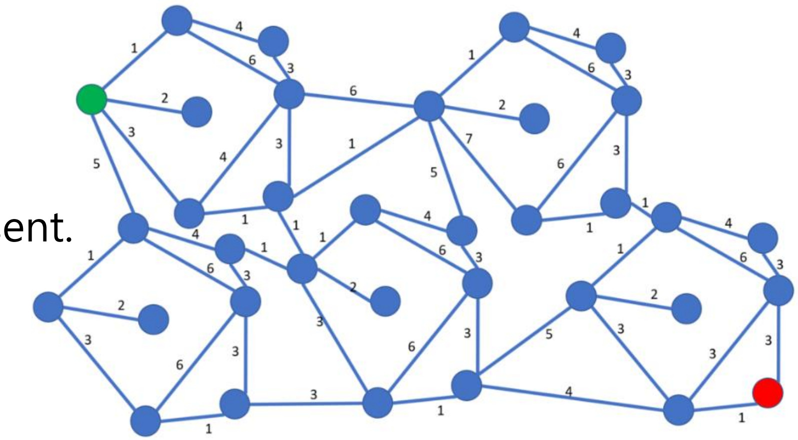
서강대학교 공과대학 컴퓨터공학과
임 인 성 교수

본 강의에서 제작하여 제공하는 PDF 파일, 동영상, 그리고 예제 코드 등의 강의 자료의 저작권은 특별히 명기되어 있지 않은 한 서강대학교에 있습니다.

본인의 학습 목적 외에 공개된 장소에 올리거나 타인에게 배포하는 등의 행위를 금합니다. 협조 부탁합니다.

[주제 6]
Graph Algorithms

Shortest-Paths Problems



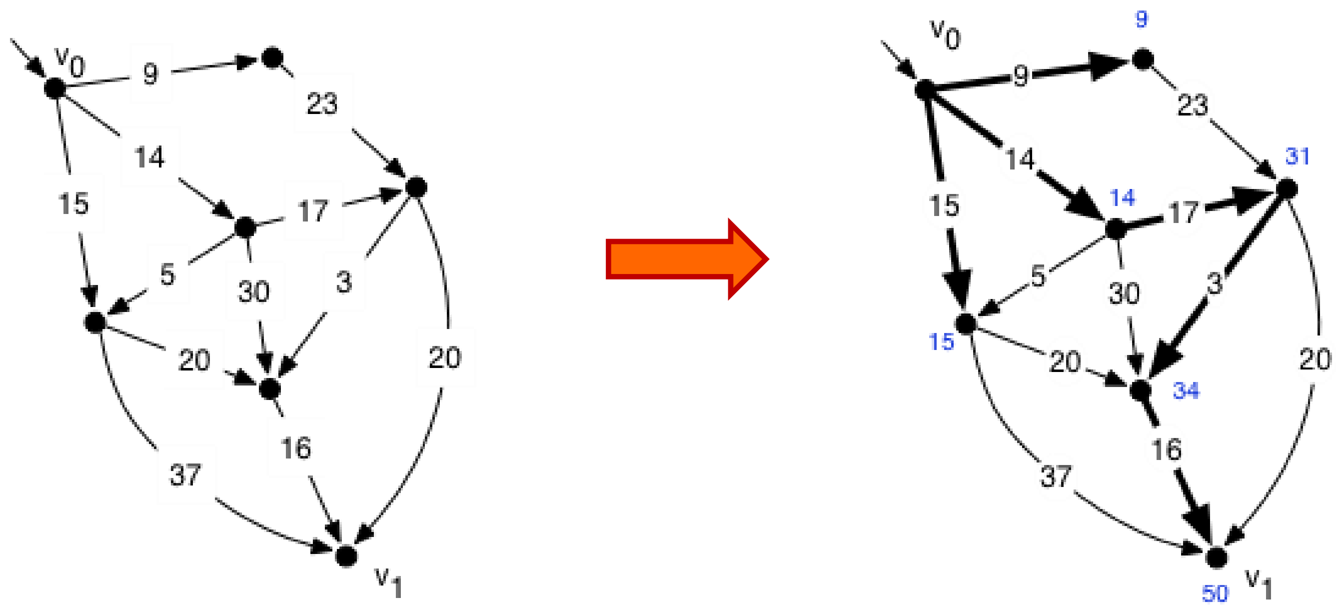
https://datascience.lc/2019/10/26/shortest-path-dijkstra-algorithm/

- **Single-source shortest-paths problem**
  - **Dijkstra's algorithm**
    - Only nonnegative-weight edges are present.
  - Bellman-Ford algorithm
    - Negative-weight edges may be present, but there are no negative-weight cycles.
- **Single-destination shortest-paths problem**
- **Singe-pair shortest-path problem**
- **All-pairs shortest-paths problem**
  - **Floyd-Warshall algorithm**
    - Negative-weight edges may be present, but there are no negative-weight cycles.
  - Johnson's algorithm for sparse graphs
    - Negative-weight edges may be present, but there are no negative-weight cycles.

➢ **The optimal-substructure property of shortest paths**
  - Subpaths of shortest paths are shortest paths!

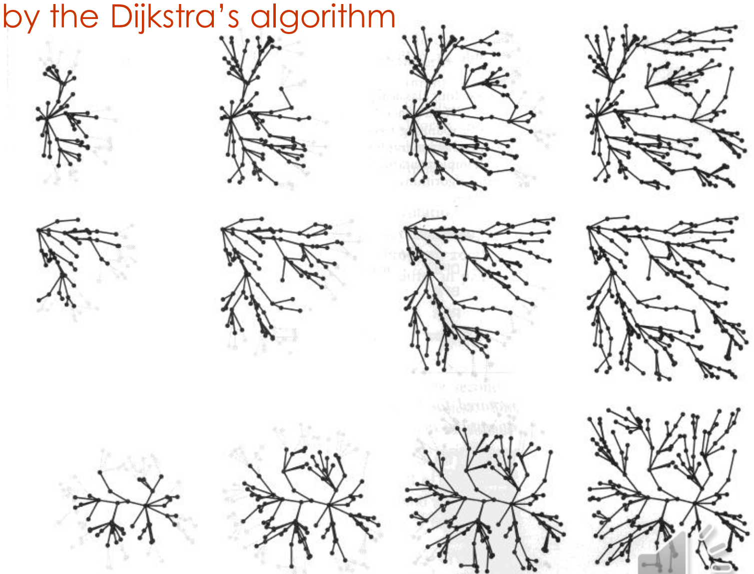
Single-Source Shortest Path

- **Problem**
  - Given **a weighted directed graph** G = (V, E) with a weighting function w(e) (w(e) ≥ 0 for the edges in G), and a source vertex v_0, find a shortest path from v_0 to each of the remaining vertices of G.

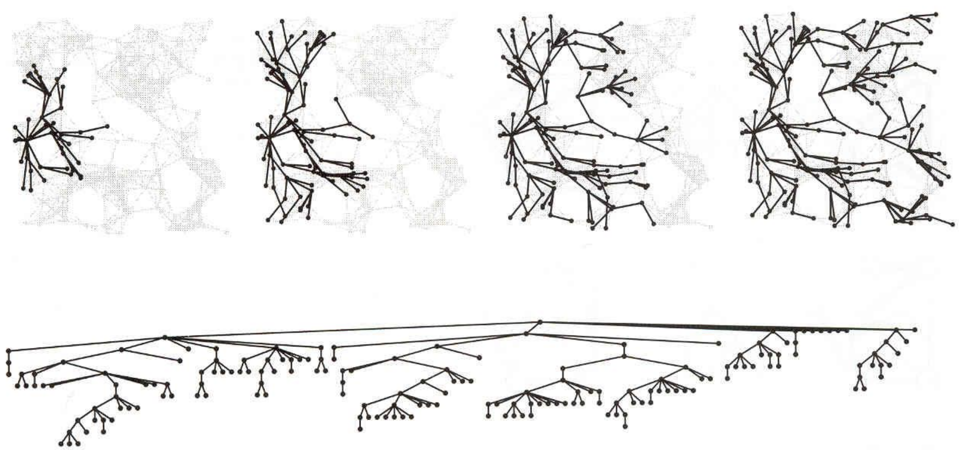


- **Note**
  - The case of **an undirected graph** can be handled by simply replacing each undirected edge e = (u, v) of length w(e) by two directed edges (u, v) and (v, u), each of the same length.
  - Only the case of a directed graph whose weights are positive (or nonnegative) is handled by the **Dijkstra's algorithm**. For a graph that allows a negative weight, the **Bellman-Ford algorithm** is one to be used.
  - Before learning the single-source shortest path algorithm that builds some tree, recall how the breadth first search tries to build a BFS tree.

A tree built by the Dijkstra's algorithm



A breadth first search tree



Dijkstra's Single-Source Shortest Path Algorithm

- A **greedy** approach

  Generate the shortest paths in non-decreasing order of lengths!

- S^1 = { v_0 }로 설정하고 시작.

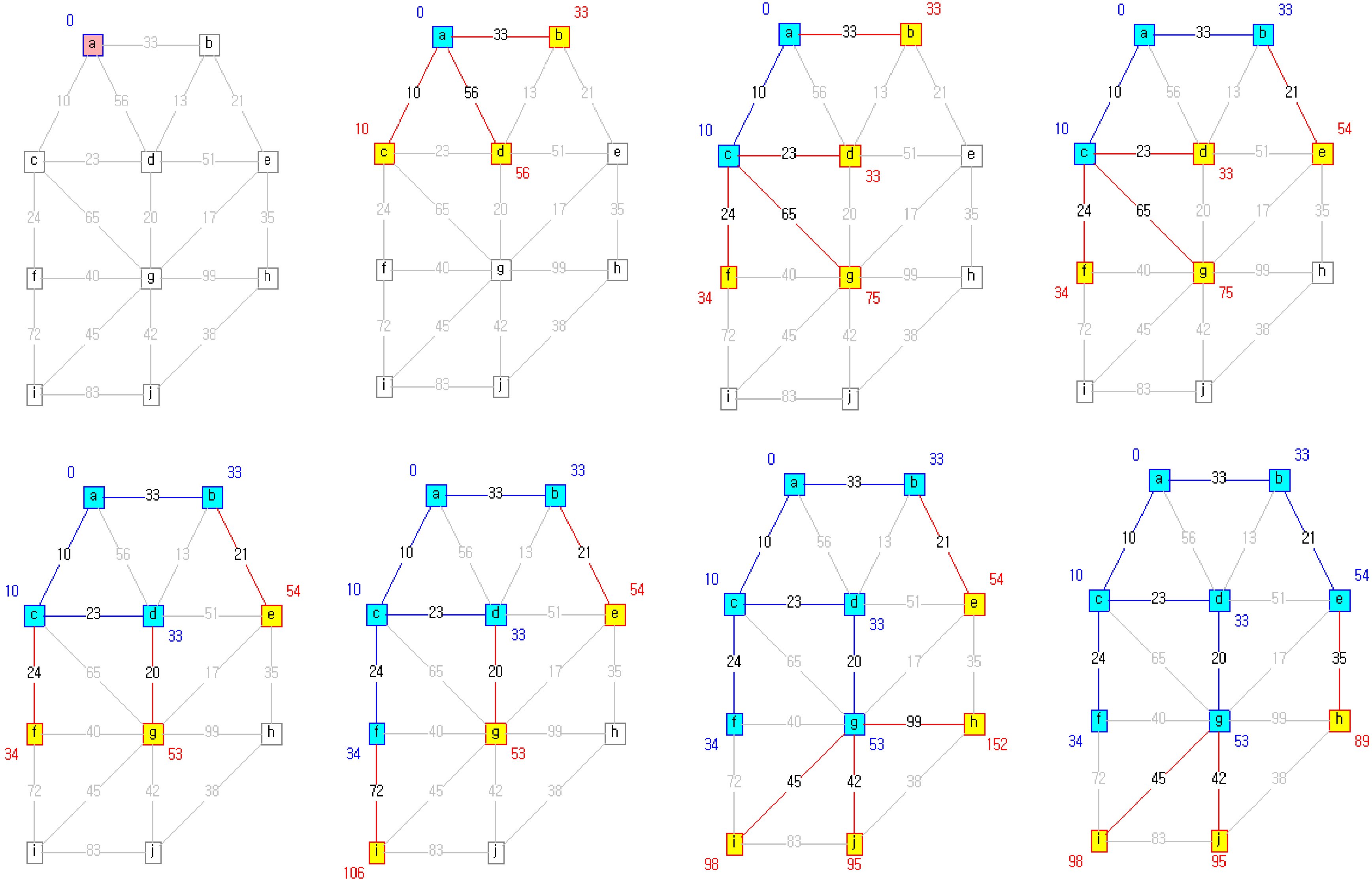
- (i = k일 때) S^k 의 꼭지점들만 사용할 경우에 대한 v_0에서 v까지의 shortest path가 구해져 있음. (v는 S^k의 꼭지점)

- S^k 상황에서 **가장 짧은** path에 대한 꼭지점 v를 S^k 로 옮긴 후 **적절한** 처리를 수행 → S^{k+1}

- (i = k+1일 때) S^{k+1} 의 꼭지점들만 사용할 경우에 대한 v_0에서 v까지의 shortest path가 구해져 있음. (v는 S^{k+1}의 꼭지점)

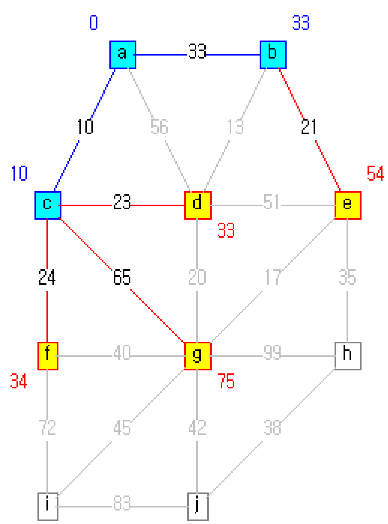
- S^n이 다 구해졌을 경우 ...

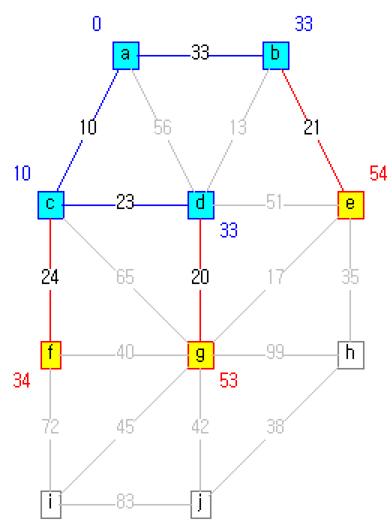
From Prof. Kenji Ikeda's Home Page



Dijkstra's Algorithm (from Introduction to Algorithms by T. Cormen)

- **A directed graph with nonnegative weight G(V, E) with w: E → [0,∞) and source s**
- **Two components for each vertex v**
  - **v.d**: an upper bound on the weight of a shortest path from s to v (a shortest path estimate)
  - **v.π**: the predecessor of v in the (current) shortest path

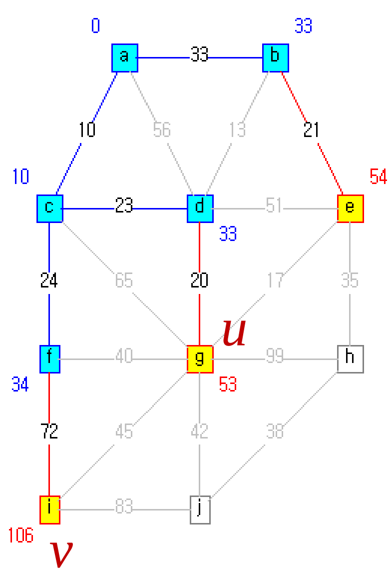


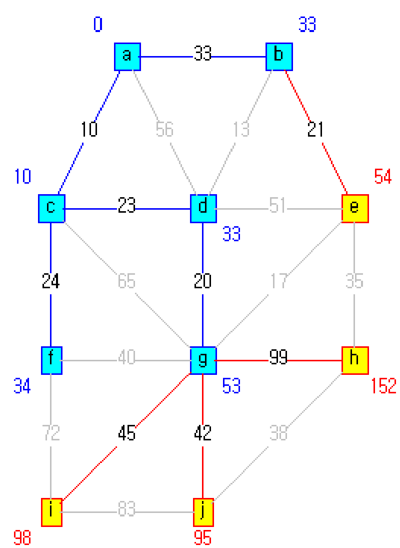


INITIALIZE-SINGLE-SOURCE(G, s)

for each vertex $v \in G.V$
  $v.d = \infty$
  $v.\pi = \text{NIL}$
$s.d = 0$

- **Relaxation**
  - The process of relaxing an edge (u, v) consists of testing whether we can **improve the shortest path to v found so far by going through u** and, if so, updating v.d and v.π.





RELAX(u, v, w)

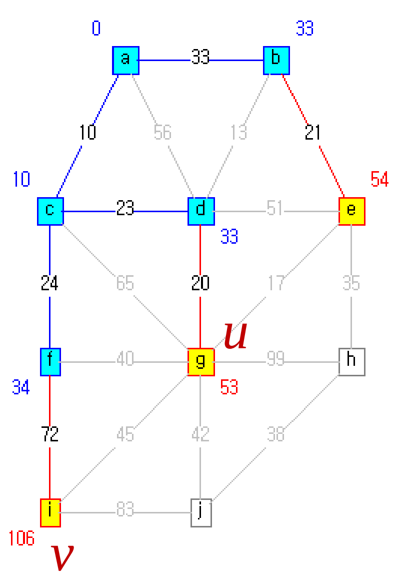
if $v.d > u.d + w(u, v)$

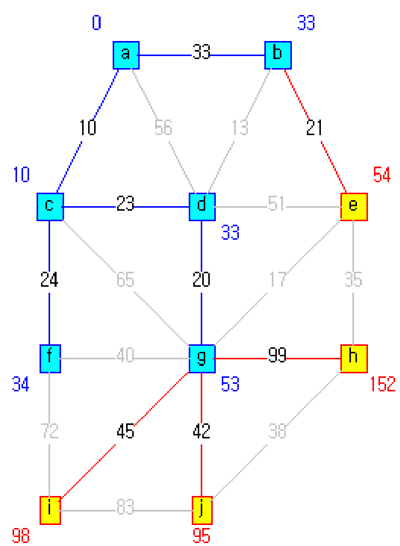
  $v.d = u.d + w(u, v)$

  $v.\pi = u$

- 아직 shortest path를 찾지 못한 vertex v에 대해
- 새롭게 선택된 vertex u에 adjacent한 vertex v에 대해

- **Dijkstra's algorithm**
  - Maintains a set **S** of vertices whose final shortest-path weight from the source **s** have already been determined.
  - 1. Select repeatedly the vertex **u** in **V-S** with the minimum shortest-path estimate, 2. adds **u** to **S**, and 3. relaxes all edges leaving **u**.





DIJKSTRA(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)
$S = \emptyset$
$Q = G.V$
while $Q \neq \emptyset$
    u = EXTRACT-MIN(Q)
    $S = S \cup \{u\}$
    for each vertex $v \in G.Adj[u]$
        RELAX(u, v, w)

➢ When the algorithm adds a vertex **u** to the set **S**, **u.d** is the final shortest-path weight from **s** to **u**.