Shortest-Path Problems

Weighted Graphs

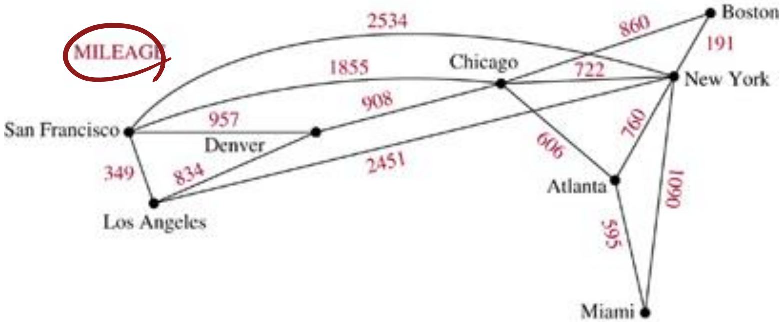
- Many problems can be modeled using **graphs with *weights* assigned to their edges**:
  - Airline flight times
  - Telephone communication costs
  - Computer networks response times

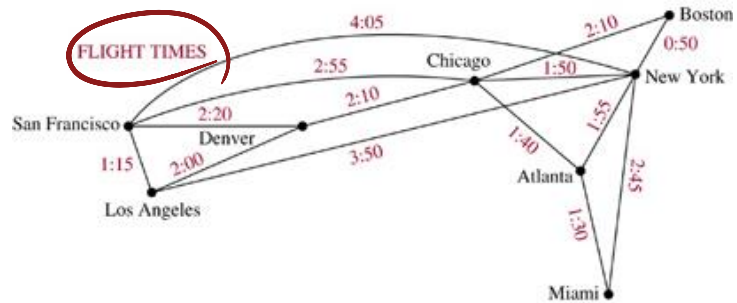
- In a weighted graph, each edge has an associated numerical value, called the weight of the edge.
- Edge weights may represent, distances, costs, etc.

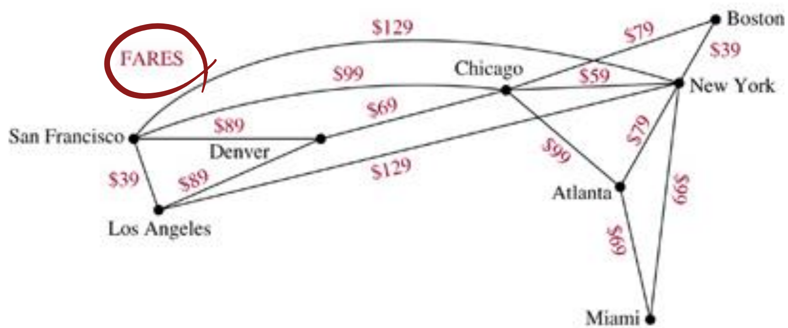
물리적인 값

Weighted Graphs – Example

- **Weighted Graphs Modeling and Airline System.**







Issue in Weighted Graphs

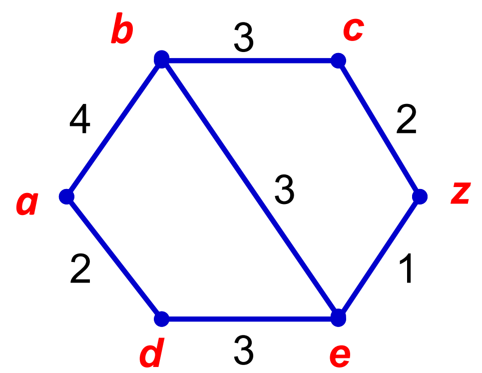
- How to find the fastest way to get to the destination ?
- How to find the cheapest flight route?
- How to connect to the sever with the minimum response time in computer network?
- These problems boil down to the ***shortest path problems***.

Simple Solution for the simple Problems

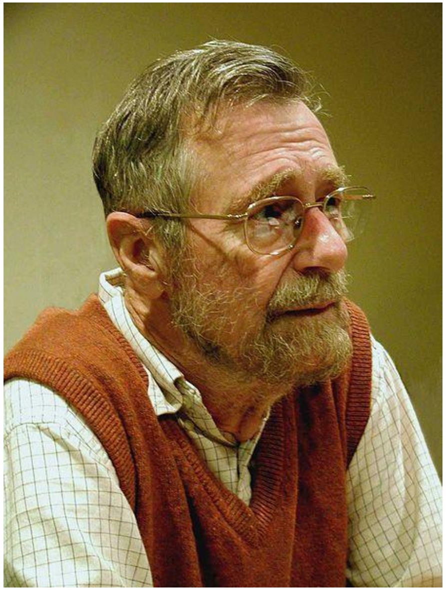
- What is the length of a shortest path between *a* to *z* in the given weighted graph?
  - How?

graph가 단순

1. List all the possible path with its length.
   - $a - b - c - z$ (length: 9)
   - $a - b - e - z$ (length: 8)
   - $a - d - e - z$ (length: 6)
   - $a - d - e - b - c - z$ (length: 13)
2. Chose a path with the shortest length
   - $a - d - e - z$ (length: 6)



Dijkstra's Algorithm

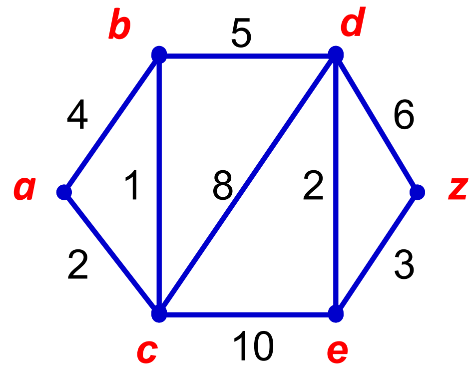


- Edsger Wybe Dijkstra (1930-2002)
  - a Dutch computer scientist
  - He received the 1972 Turing Award (Nobel Prize of computing) for fundamental contributions to developing programming languages.
  - He was the Schlumberger Centennial Chair of Computer Sciences at The University of Texas at Austin from 1984 until 2000.
- Dijkstra's Algorithm
  - Conceived by Edsger Dijkstra in 1956 and published in 1959.
  - A graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree.

인지도, 중요↑
좋은 알고리즘은 아님
처음 제안한 base model

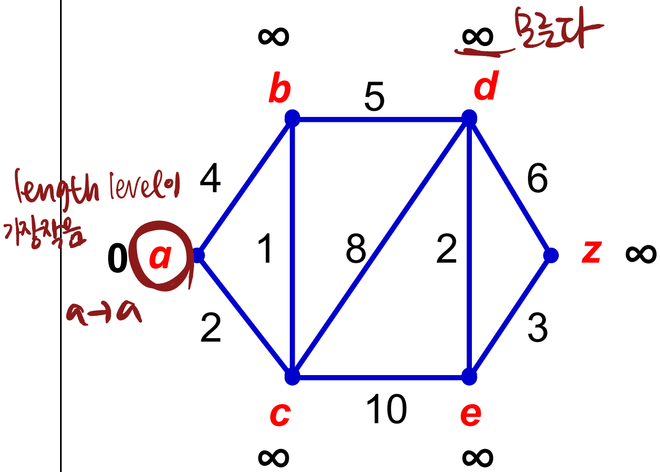
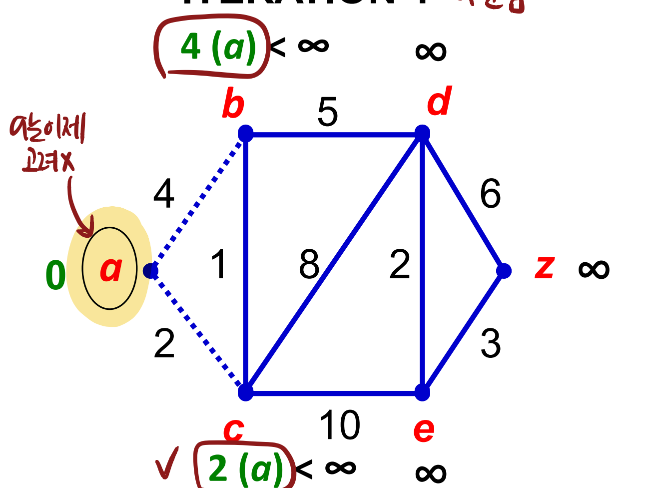
Dijkstra’s Algorithm – Example#1

- Use Dijkstra’s Algorithm to find the length of a shortest path between *a* to *z* in the given weighted graph.

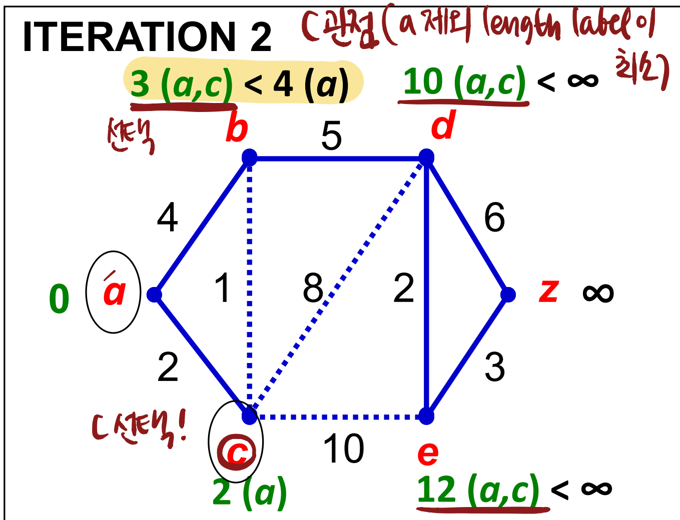


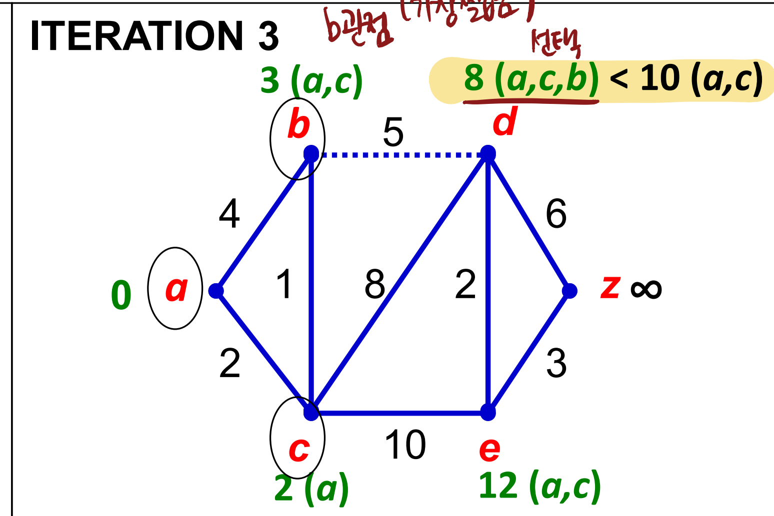
SOLUTION

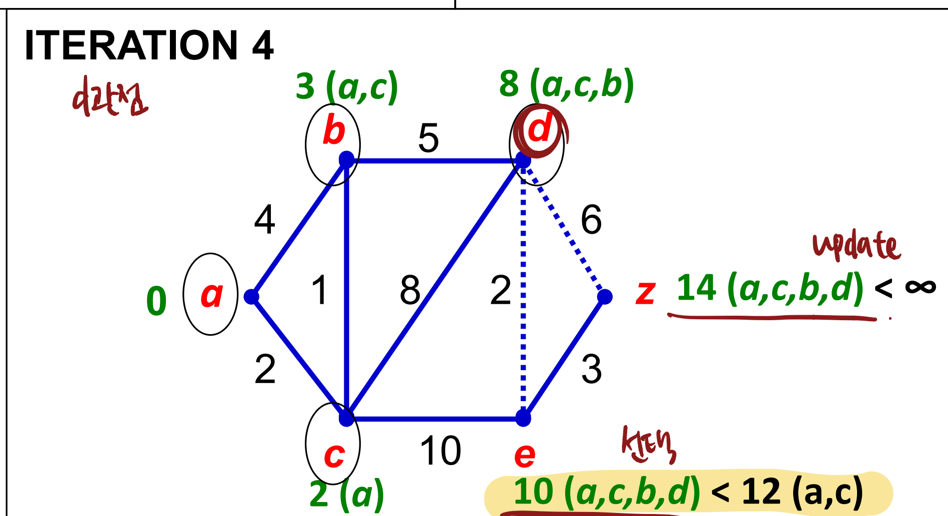
초기화

length label Initialization	ITERATION 1 a관점
	

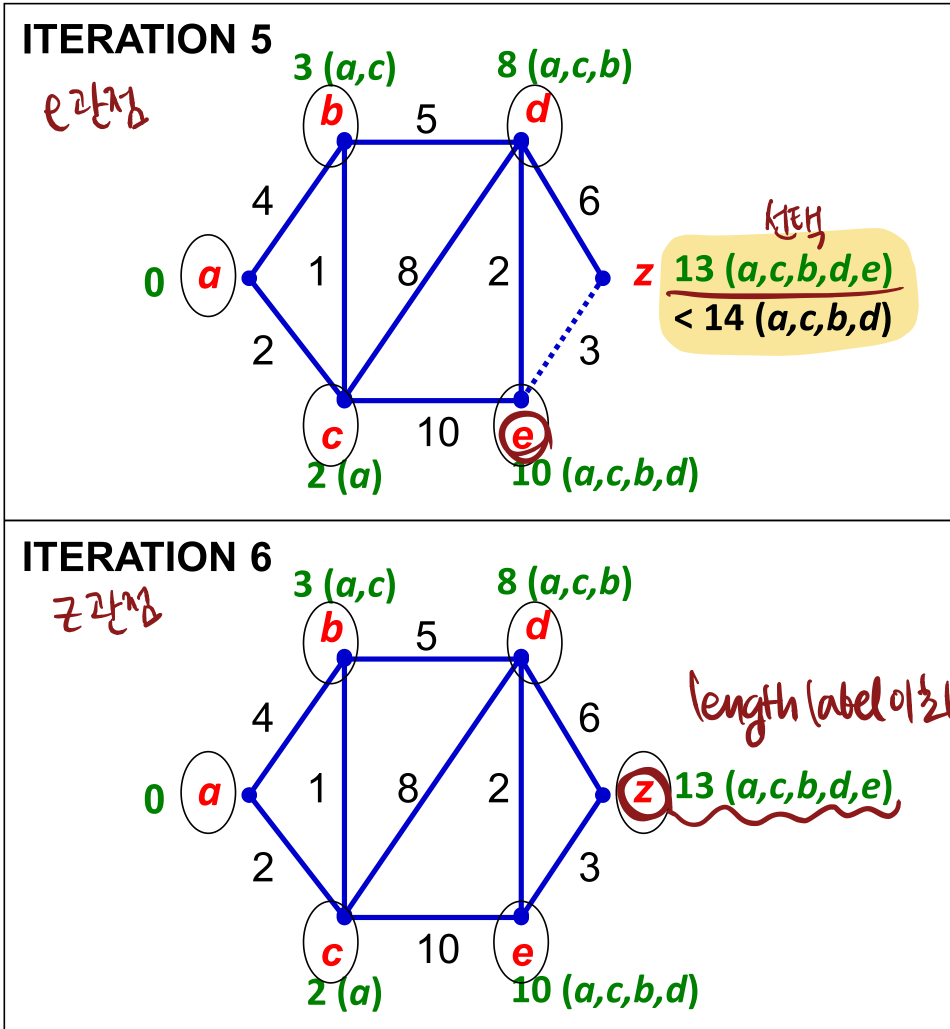
Dijkstra's Algorithm – Example#1







Dijkstra's Algorithm – Example#1



- The algorithm terminate when *z* is circled.
- A shortest path from *a* to *z* is *a*, *c*, *b*, *d*, *e*, *z* with length 13.

가장 좋은 예나 최악.

자주발생하는 오류!

Fallacies about Dijkstra’s Algorithm

- Previous ‘Example#1’ may cause some fallacies about Dijkstra's algorithm as follows.
  - **Fallacy#1**: The shortest path by Dijkstra’s algorithm is the sequence of visited nodes in order from the first iteration to the last iteration.
    - **In example#1:** Sequence of visited nodes (*a,c,b,d,e,z*) == shortest path

↳ 방문하는순서대로 shortest path sequence가 발견X, 즉 최적의 알고리즘X

  - **Fallacy#2**: Shorter sequence of nodes (previously recorded) is overwritten by longer sequence of nodes (currently calculated).
    - **In example#1: 3 (*a,c*) < 4 (*a*) , 8 (*a,c,b*) < 10 (*a,c*) , 10 (*a,c,b,d*) < 12 (a,c), 13 (*a,c,b,d,e*) < 14 (*a,c,b,d*)**

항상 새걸로 update된다? → No.

거리값 length label

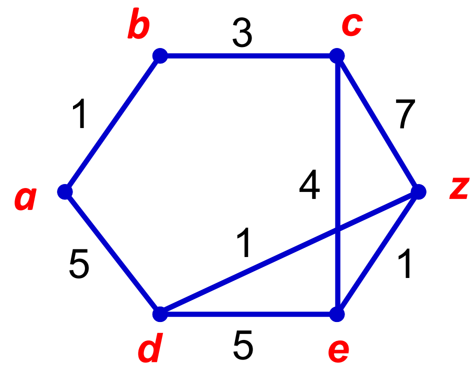
경로(배열구조) sequence

  - **Fallacy#3**: Dijkstra’s algorithm finishes when all of nodes are visited.
    - **In example#1:** The algorithm finishes when all of nodes (*a,b,c,d,e,z*) are visited.

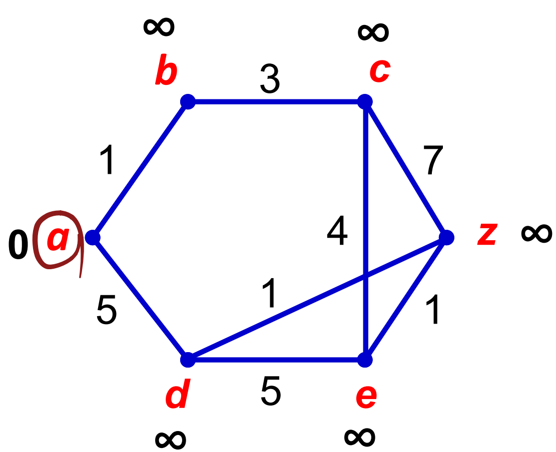
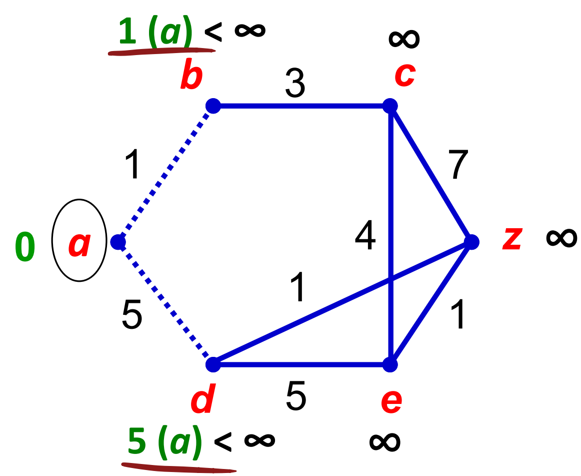
모든 node를 visit할 필요X

Dijkstra's Algorithm – Example#2

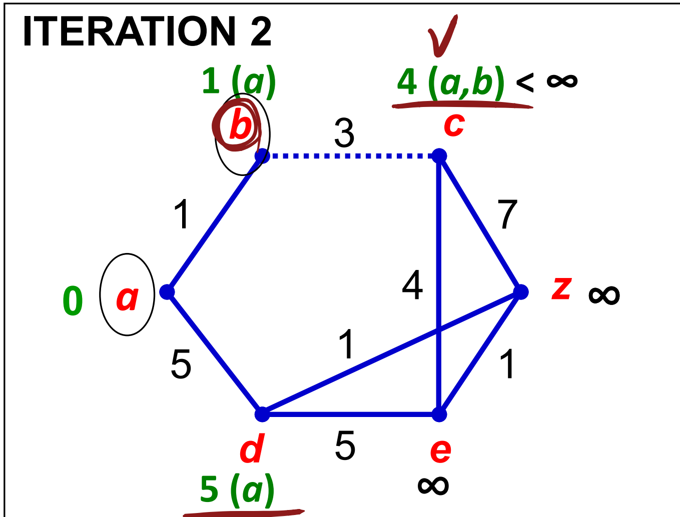
- Use Dijkstra's Algorithm to find the length of a shortest path between *a* to *z* in the given weighted graph.

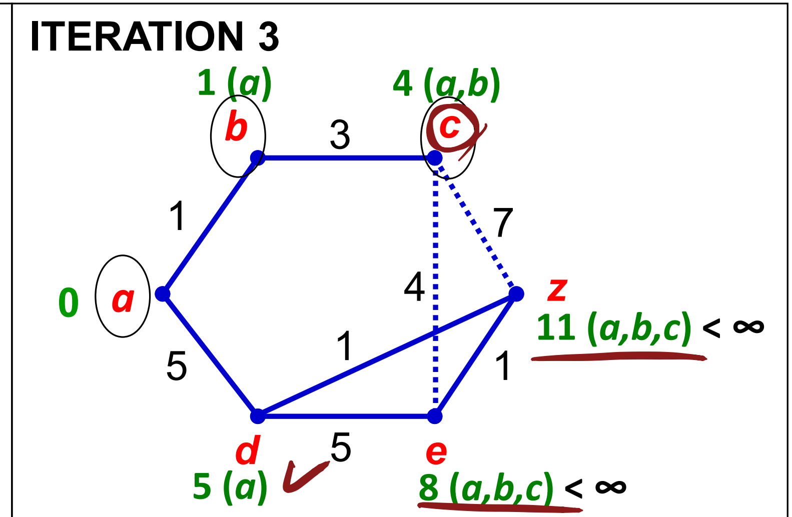


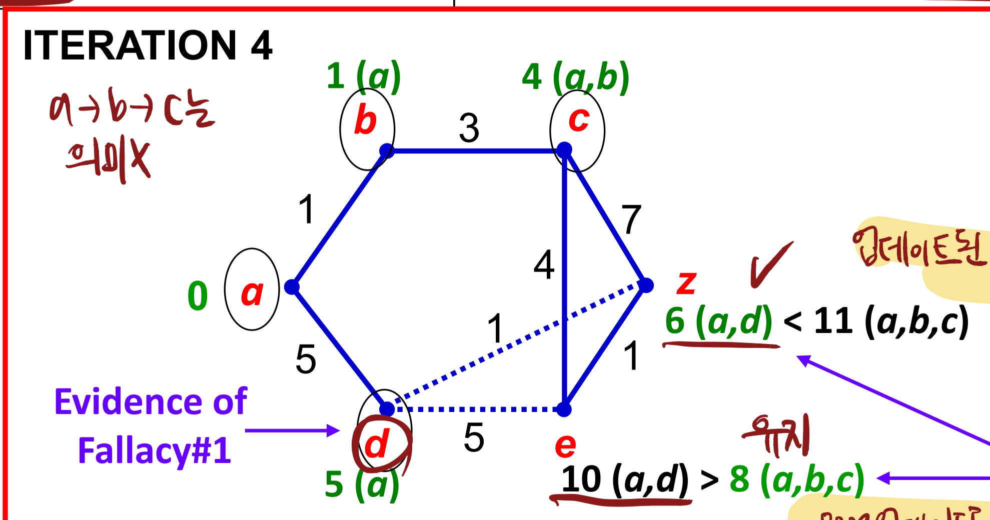
SOLUTION

length label Initialization	ITERATION 1
	

Dijkstra's Algorithm – Example#2







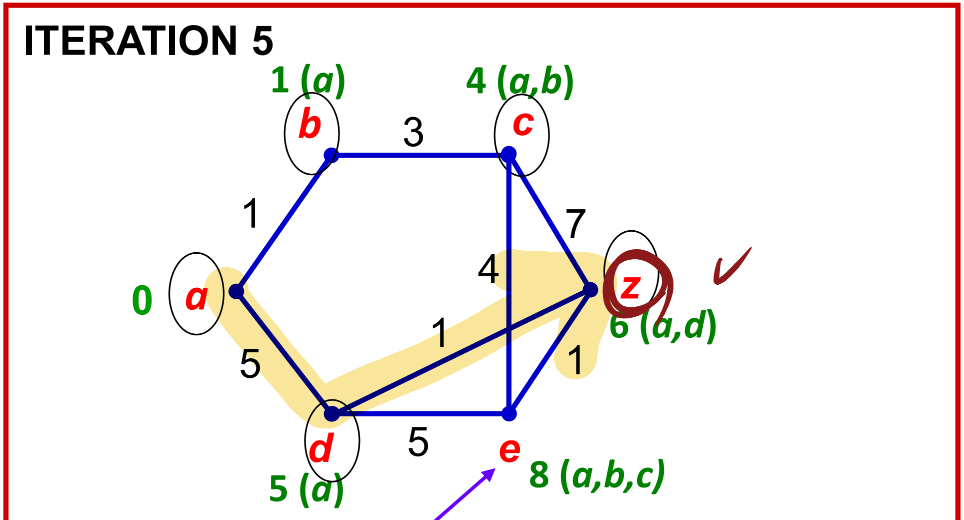
업데이트된 sequence가 무조건 긴 것X

방문한 node의 순서 ≠ shortest path

Evidence of Fallacy#2

무조건 업데이트되는 것X

Dijkstra's Algorithm – Example#2



Evidence of Fallacy#3

e는 안감. 않는다
visit X

local optima가
entire optima는 아니다.

- The algorithm terminate when *z* is circled.
- A shortest path from *a* to *z* is *a*, *d*, *z* with length 6.

Dijkstra's Algorithm

Pseudo code for dijkstra's algorithm

procedure *Dijkstra*(G: weighted connected simple graph with all weights positive)

{ G has vertices $a=v_1, \ldots, v_n=z$ and weights $w(v_i, v_j)$ }

for $i := 2$ to n { Length label initialization }

$\quad L(v_i) := \infty$ { $L(v)$ means the length label of v } 초기화

$L(a) := 0$ { $a = v_1$ } 시작점

$S := \varnothing$ { Set S is used for saving visited vertices } : 더이상 고려하지 않는 vertex를 넣음

create $P(v_n)_n$:= *all* v_0 { array $P(v_n)_n$ is 2-dimentional nxn array used for saving vertex-sequence of shortest path } ex. [a, b, e] 엄밀히 n-1

배열안의 값을 dummy로 초기화 { v_0 is dummy value for $P(v_n)_n$ initialization }

↳ 현재 기록된 값들 중 어디까지가 의미있는 값인지 알기 위한 플래그

while $z \notin S$

begin

이미 선택되지 X, 최소

$\quad u$:= a vertex not in S with $L(u)$ minimal

$\quad$ **if** $(u \neq z)$ **then**

$\quad$ **begin**

$\quad\quad$ **for** all adjacent vertices to u but not in S

$\quad\quad\quad$ **if** $L(u) + w(u, v) < L(v)$ **then**

$\quad\quad\quad$ **begin** 기존

$\quad\quad\quad\quad L(v) := L(u) + w(u, v)$ → update

$\quad\quad\quad\quad k := 1$

$\quad\quad\quad\quad$ **while** $P(u)_k \neq v_0$

위치증가 $\quad\quad\quad\quad\quad P(v)_{k++} := P(u)_{k++}$

$\quad\quad\quad\quad \boxed{P(v)_{k++} = u}$ 맨 마지막에 직전 vertex 넣기

$\quad\quad\quad\quad$ **while** $P(v)_k \neq v_0$ { To eliminate the garbage elements remained in $P(v)_n$ }

$\quad\quad\quad\quad\quad P(v)_{k++} := v_0$

$\quad\quad\quad$ **end**

P(v)가 원래 이전 length label로 갖고있던 path가 P(u)까지의 path보다 훨씬 길다? → 내가 들어간 다음자리에 v_0가 아니라 옛날의 쓰레기 값이 있다는 것

$\quad$ **end**

$\quad S := S \cup \{u\}$ { It corresponds to mark it as visited (use circle) } → 마지막 남은게 z인 경우 더 비교할 필요X → 알고리즘 종료

end

{ $P(z)_n$ = vertex-sequence of shortest path from a to z }, { $L(z)$ = length of a shortest path from a to z }

Analysis of Dijkstra's Algorithm

- How long does this take?
- Here, a step is considered as an addition.
- If the list has n vertices, **worst case scenario** is that it takes $n(n-1)/2$ steps" : $1+2+\ldots(n-1) = n(n-1)/2$ → key factor n^2

모든 vertice들이 연결돼있는 경우

```
    for all adjacent vertices to u but not in S
            if L(u) + w(u, v) < L(v) then {Maximum n-1 steps ~ minimum 1 step}
                begin
                    L(v) := L(u) + w(u, v)
                    k := 1
                    while P(u)_k ≠ v_0
                            P(v)_k++ := P(u)_k++
                    P(v)_k++ = u
                    while P(v)_k ≠ v_0 { To eliminate the garbage elements remained in P(v)_n }
                            P(v)_k++ := v_0
                end
  end
  S := S U {u}  {it corresponds to mark it as visited (use circle)}
end
{P(z)_n = vertex-sequence of shortest path from a to z}, {L(z) = length of a shortest path from a to z}
```

Analysis of Dijkstra’s Algorithm

- Time complexity of Dijkstra’s Algorithm
  - $n(n\text{-}1)/2$ which is $O(n^2)$.

↳ bubble sort