INT102
Algorithmic Foundations and Problem Solving
Greedy Methods

Dr Pengfei Fan
Department of Intelligent Science

西交利物浦大學
Xi'an Jiaotong-Liverpool University

Learning outcomes

- ✓ Understand what greedy method is
- ✓ Able to apply Prim's algorithm to find minimum spanning tree
- ✓ Able to apply Kruskal's algorithm to find minimum spanning tree
- Able to apply Dijkstra's algorithm to find single-source shortest-paths

Dijkstra’s algorithm*

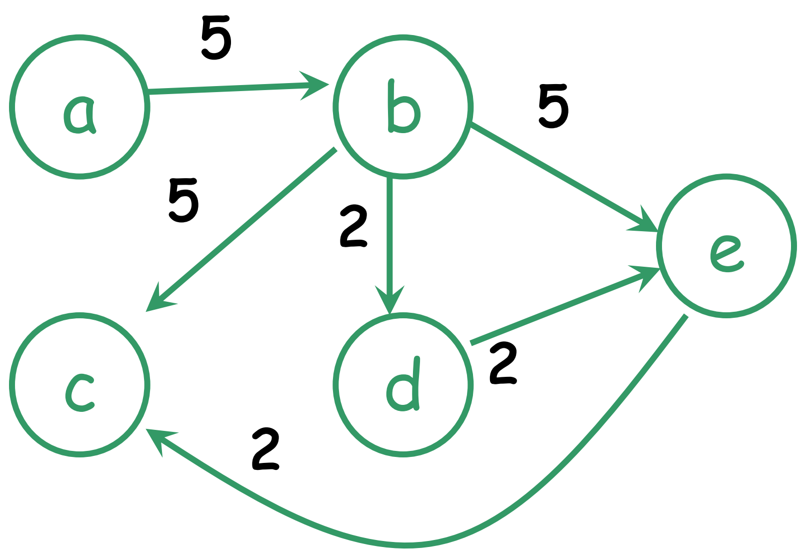
*Edsger W. Dijkstra (1930–2002), a noted Dutch pioneer of the science and industry of computing, discovered this algorithm in the mid-1950s. Dijkstra said about his algorithm: “This was the first graph problem I ever posed myself and solved. The amazing thing was that I didn’t publish it. It was not amazing at the time. At the time, algorithms were hardly considered a scientific topic.”

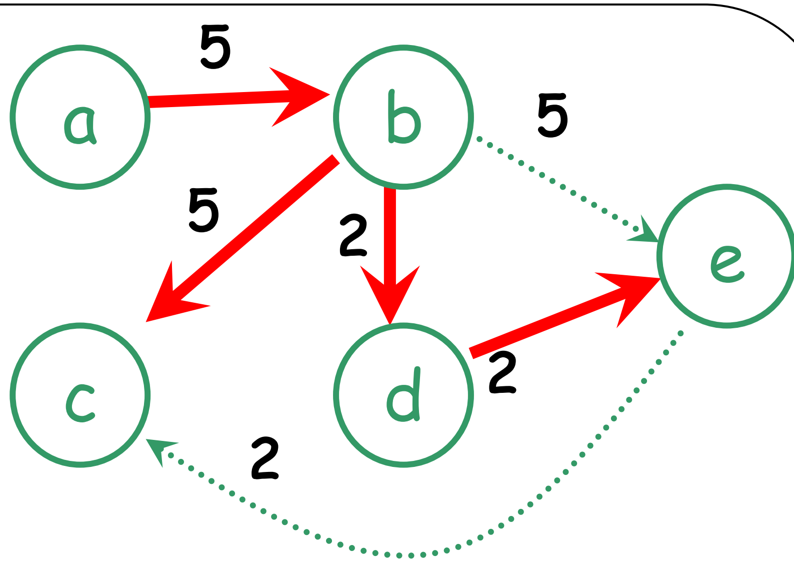
Single-source shortest-paths

- Consider a (un)directed connected graph *G*
  - The edges are labelled by weight
- Given a particular vertex called the ***source***
  - Find **shortest paths** from the source to all other vertices (shortest path means the total weight of the path is the smallest)

Example

Directed Graph G
(edge label is weight)
<u>a</u> is source node

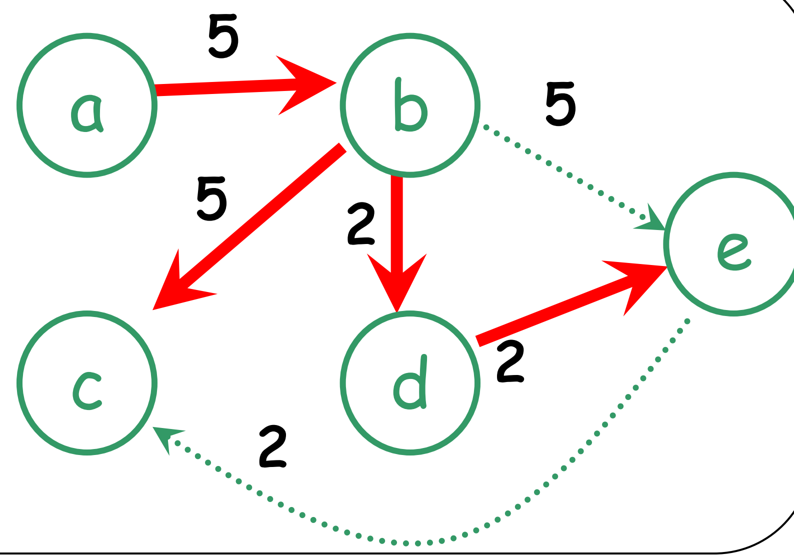


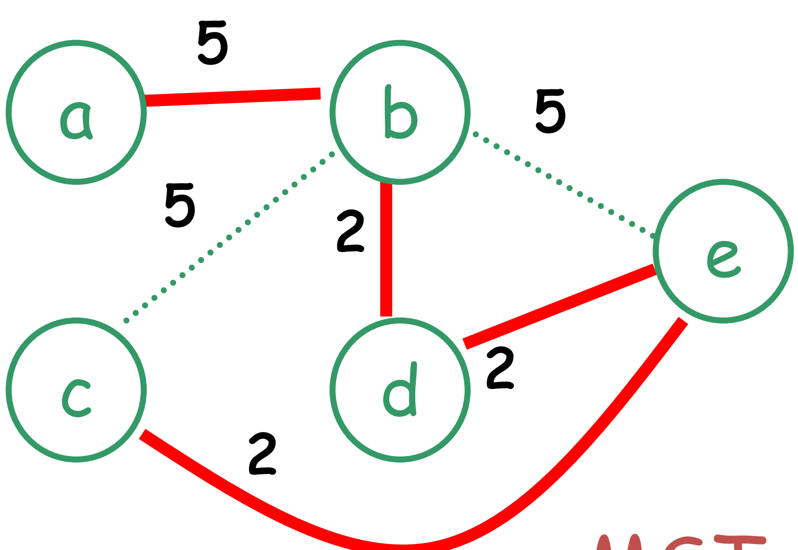


thick lines:
shortest path
dotted lines: not
in shortest path

MST and Single-source shortest paths

Shortest paths from a





MST

What is the difference between MST and shortest paths from a?

Optimal substructure of shortest-paths

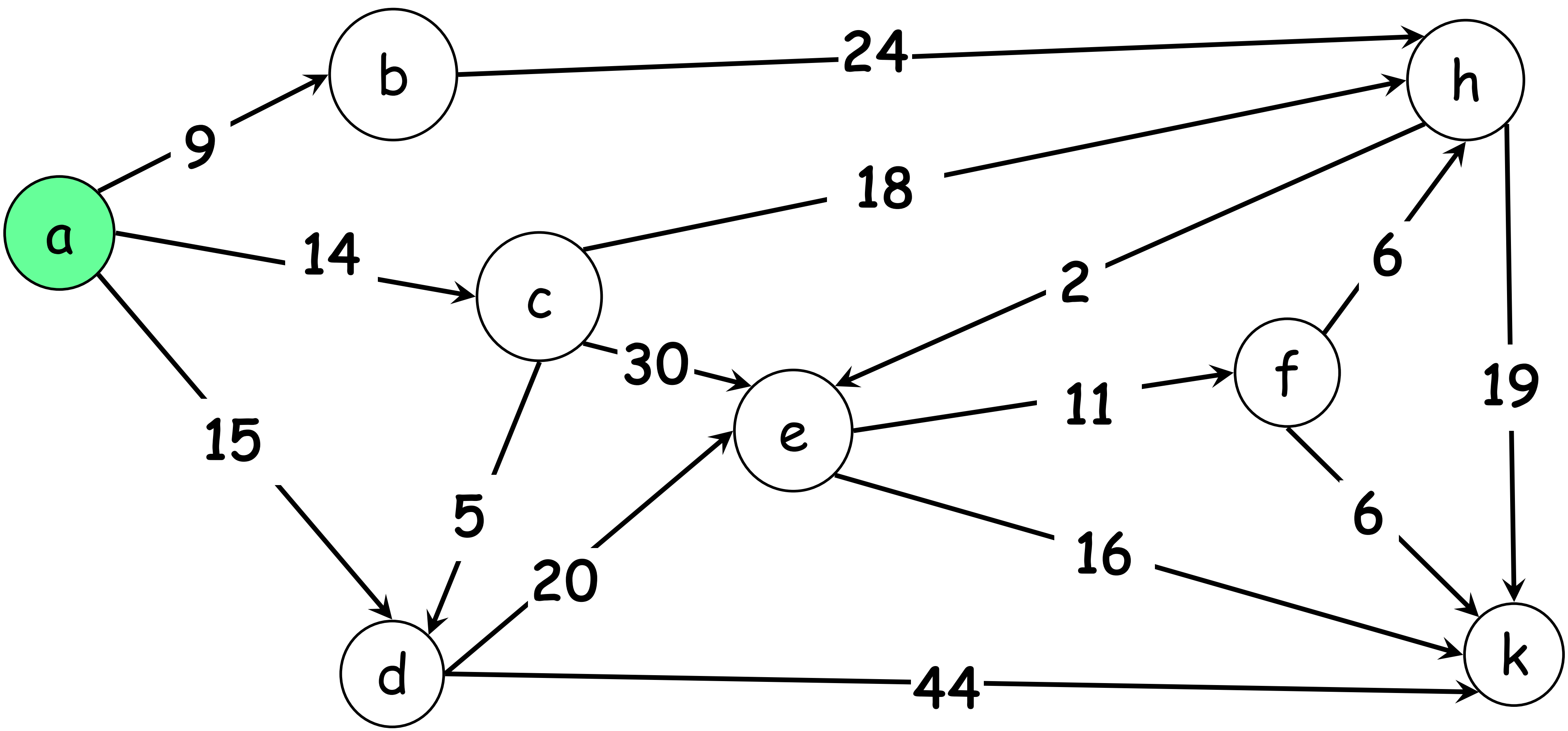
- Optimal substructure of a shortestpath
  - A shortest path between two vertices contains other shortest paths within it.
- Algorithms
  - there are many algorithms to solve this problem, one of them is Dijkstra's algorithm, which assumes the weights of edges are **non-negative**

Dijkstra's algorithm

- **Input:** A directed connected weighted graph G and a source vertex s
- **Output:** For every vertex v in G, find the shortest path from s to v
- **Dijkstra's algorithm** runs in iterations:
  - in the i-th iteration, the vertex which is the i-th closest to s is found,
  - for every remaining vertices, the current shortest path to s found so far (this shortest path will be updated as the algorithm runs)

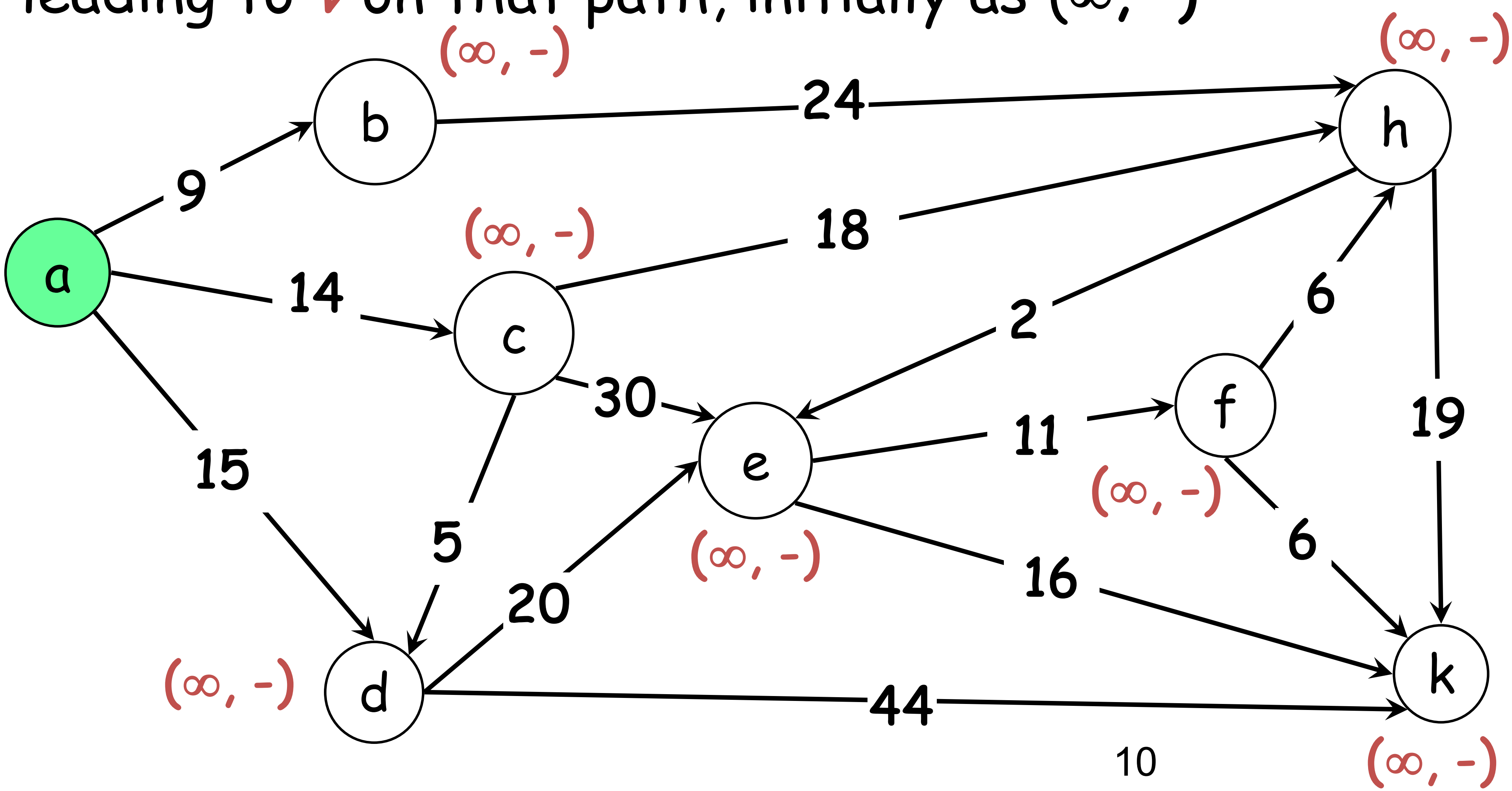
Dijkstra's algorithm

- Suppose vertex *a* is the source, we now show how Dijkstra's algorithm works



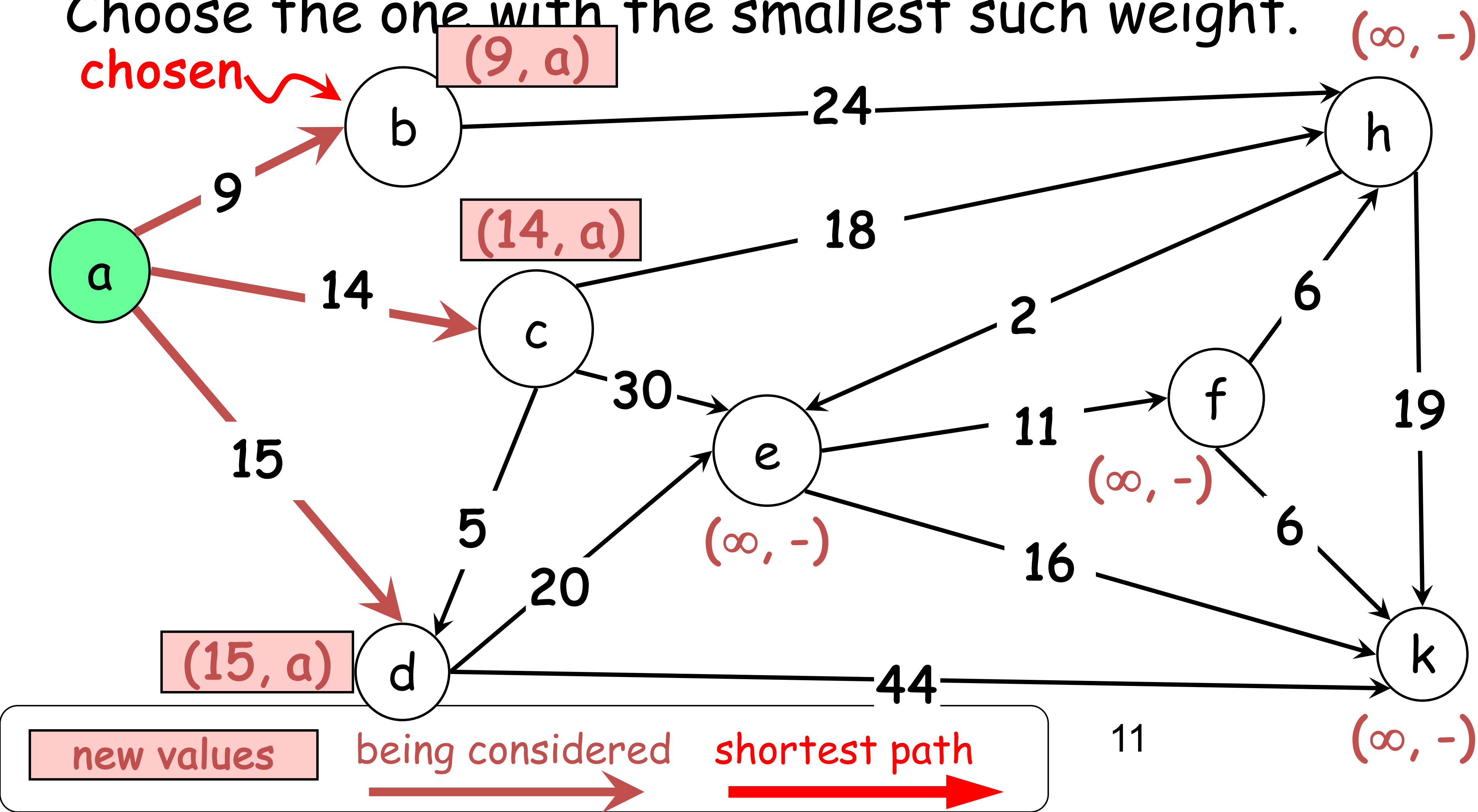
Dijkstra's algorithm

- Every vertex *v* keeps 2 labels: (1) the weight of the current shortest path from *a*; (2) the vertex leading to *v* on that path, initially as (∞, -)



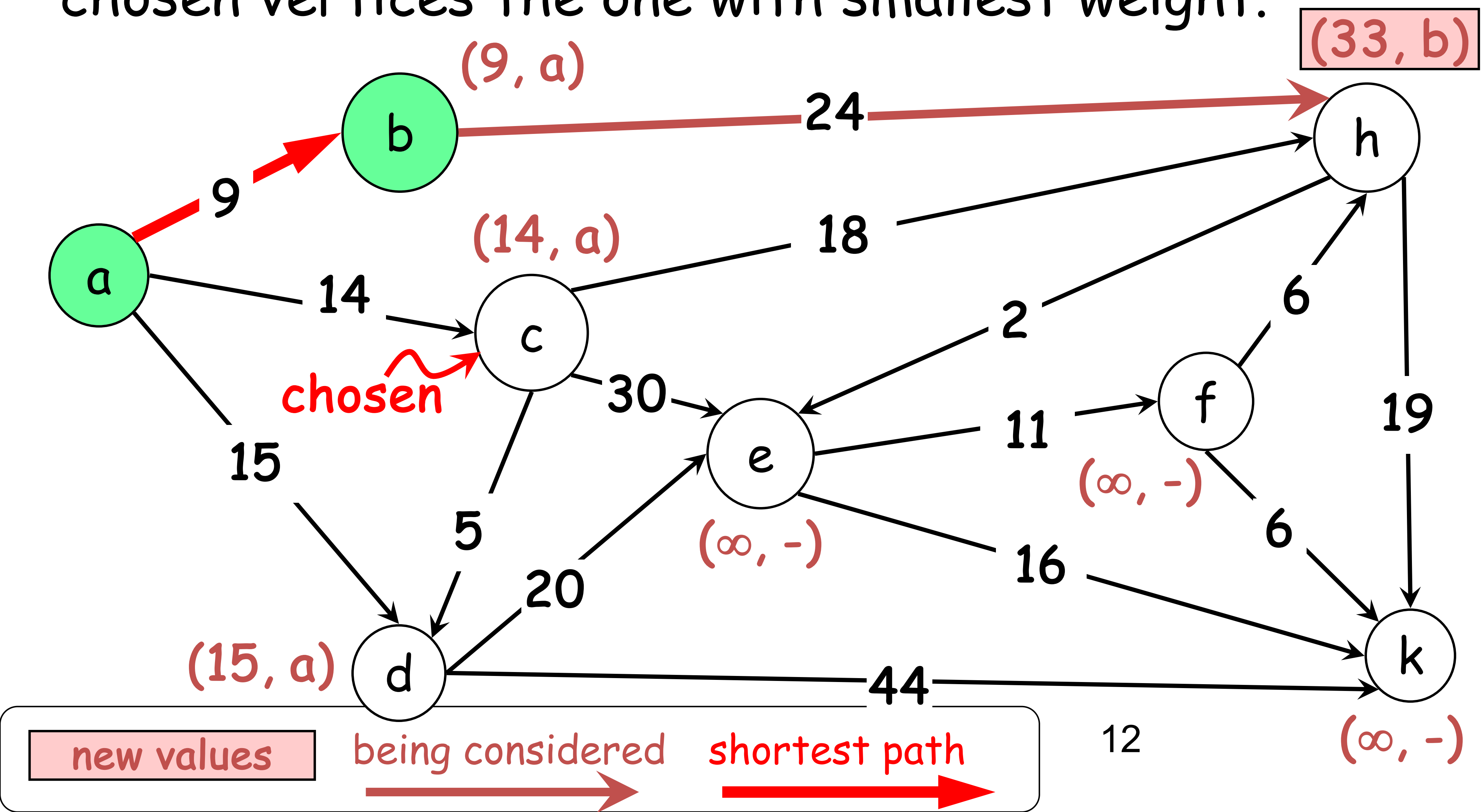
Dijkstra's algorithm

•For every neighbor *u* of *a*, update the weight to the weight of *(a,u)* and the leading vertex to *a*. Choose the one with the smallest such weight.



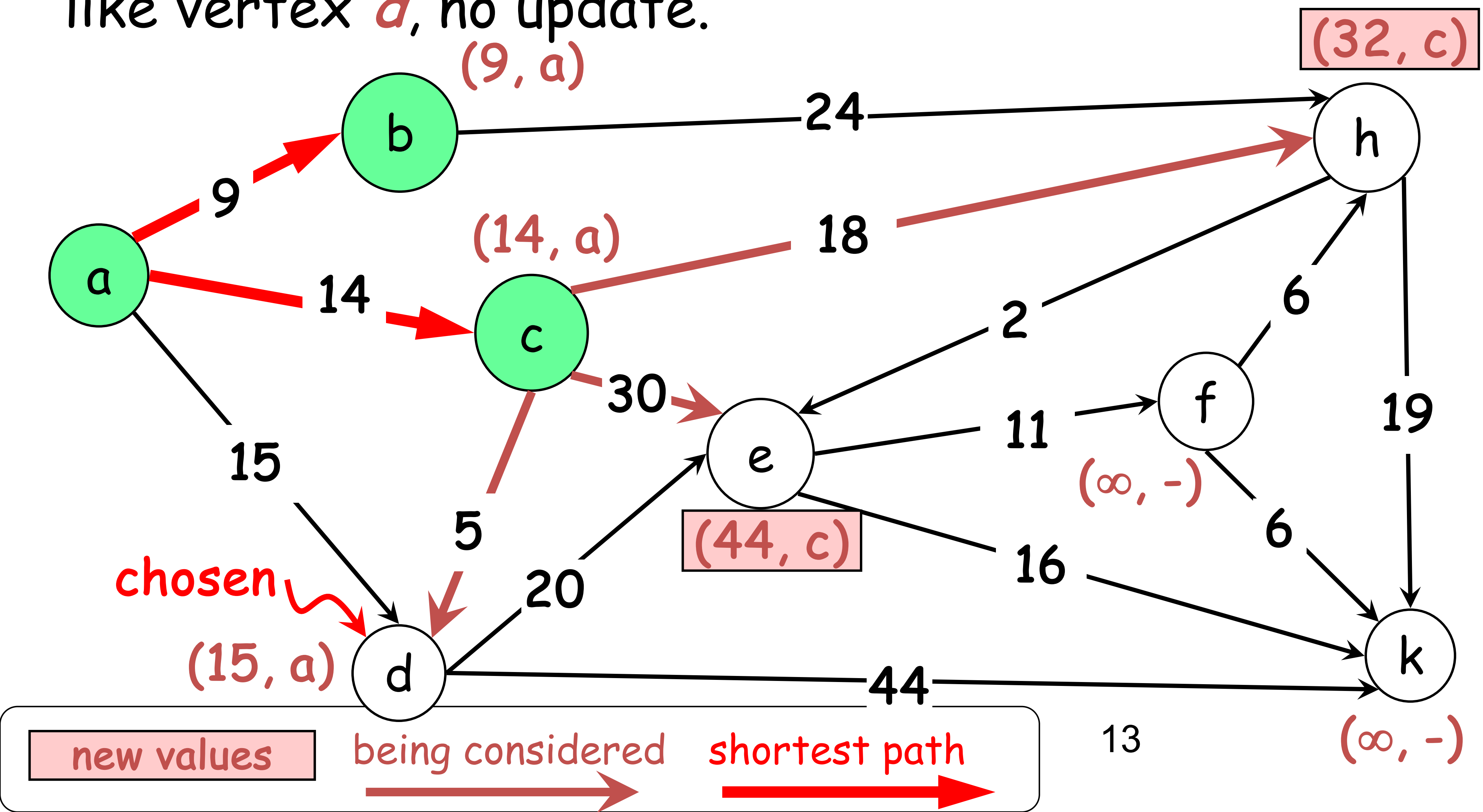
Dijkstra's algorithm

- For every un-chosen neighbor of vertex *b*, update the weight and leading vertex. Choose among all un-chosen vertices the one with smallest weight.



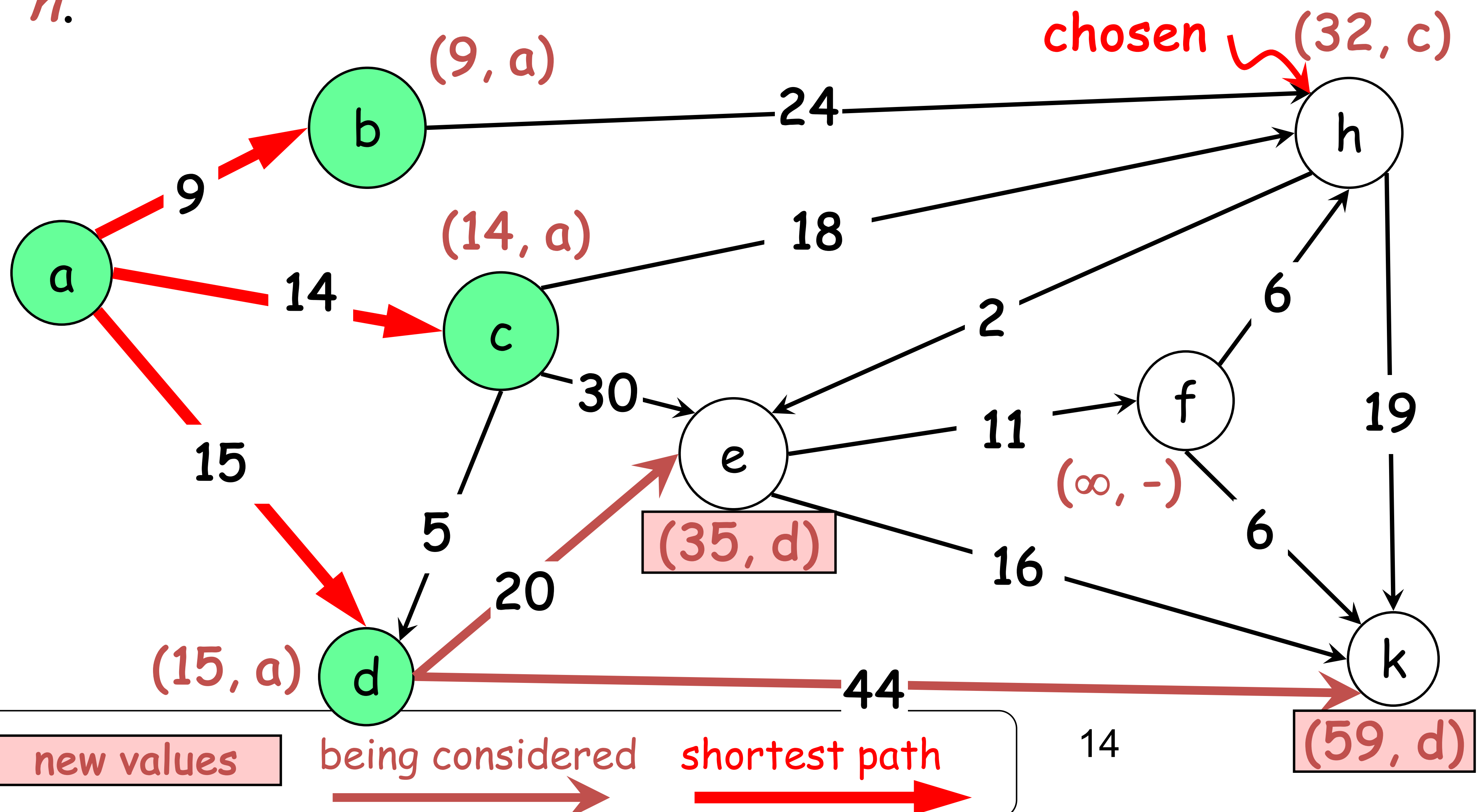
Dijkstra's algorithm

•If a new path with smallest weight is discovered, e.g., for vertex *h*, the weight is updated. Otherwise, like vertex *d*, no update.



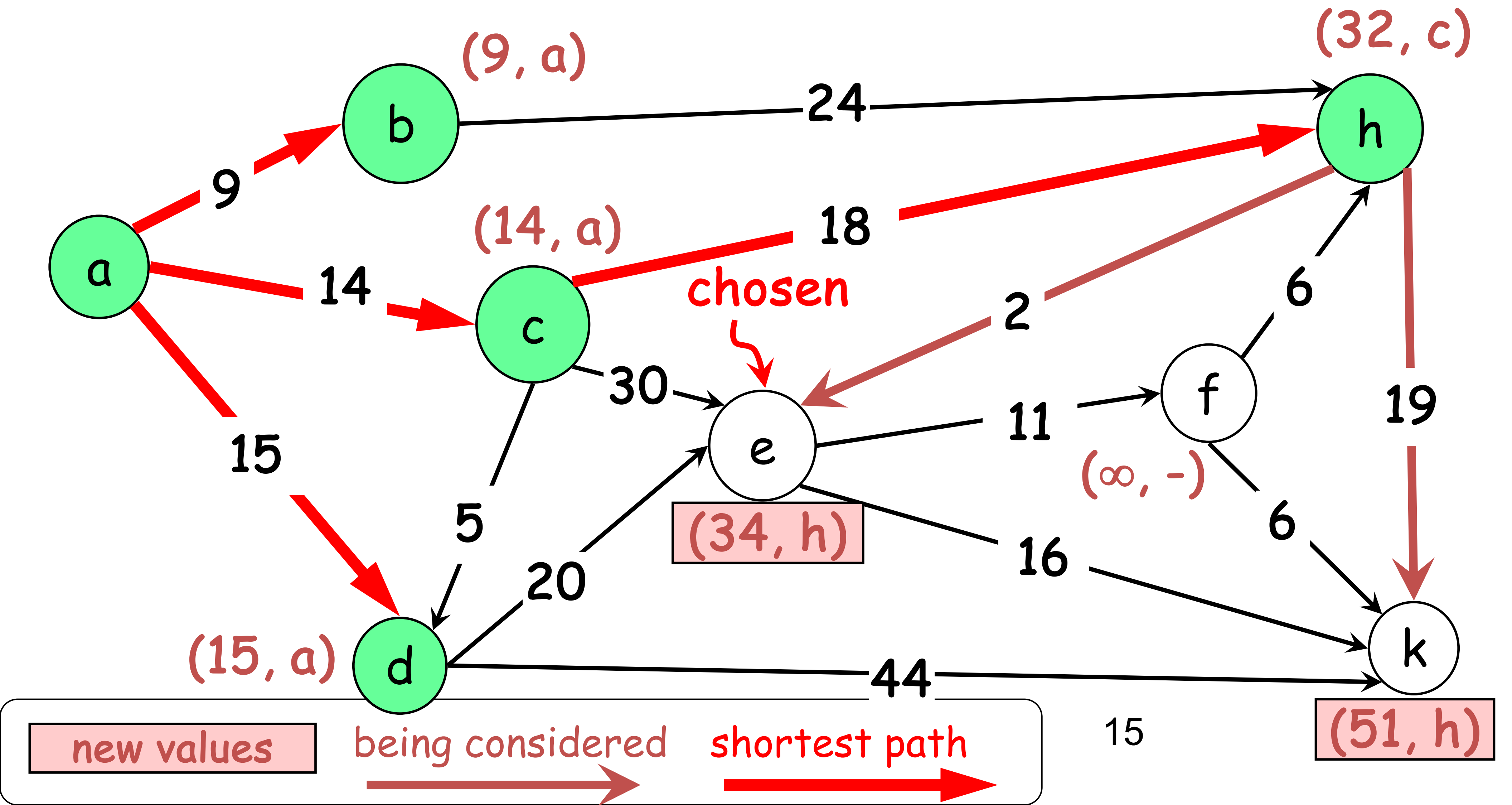
Dijkstra's algorithm

- Repeat the procedure. After *d* is chosen, the weight of *e* and *k* is updated. Next vertex chosen is *h*.



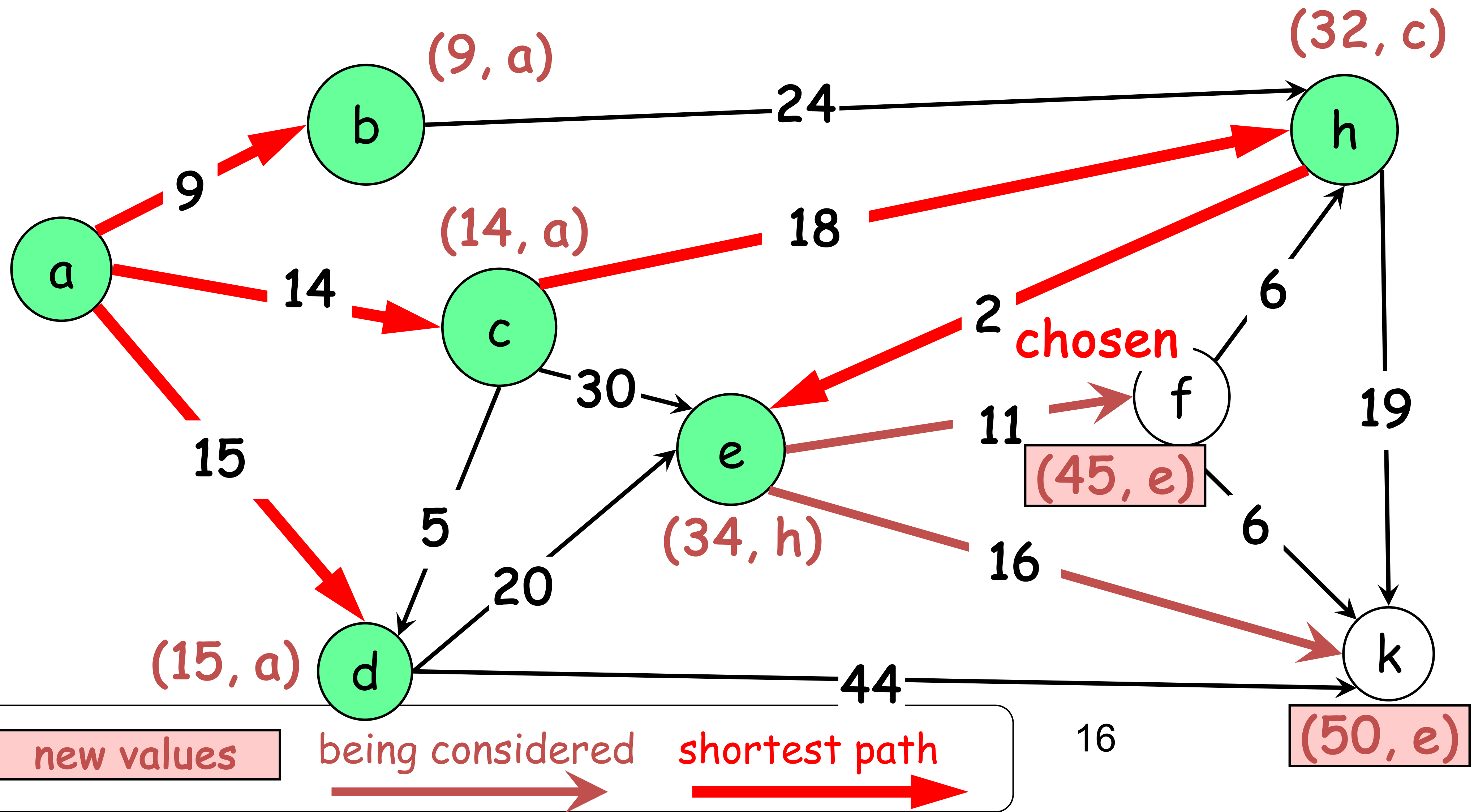
Dijkstra's algorithm

- After *h* is chosen, the weight of *e* and *k* is updated again. Next vertex chosen is *e*.



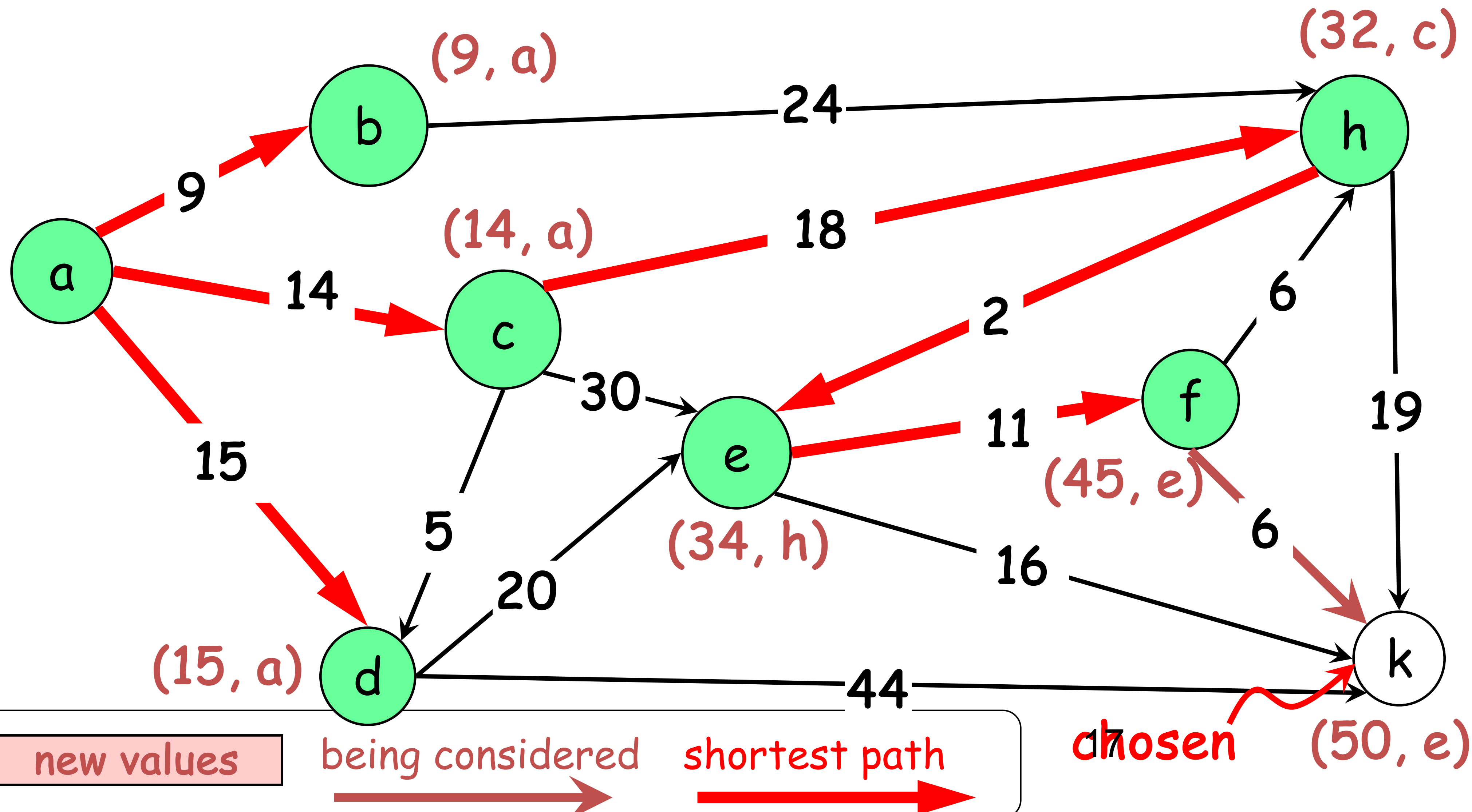
Dijkstra's algorithm

- After *e* is chosen, the weight of *f* and *k* is updated again. Next vertex chosen is *f*.



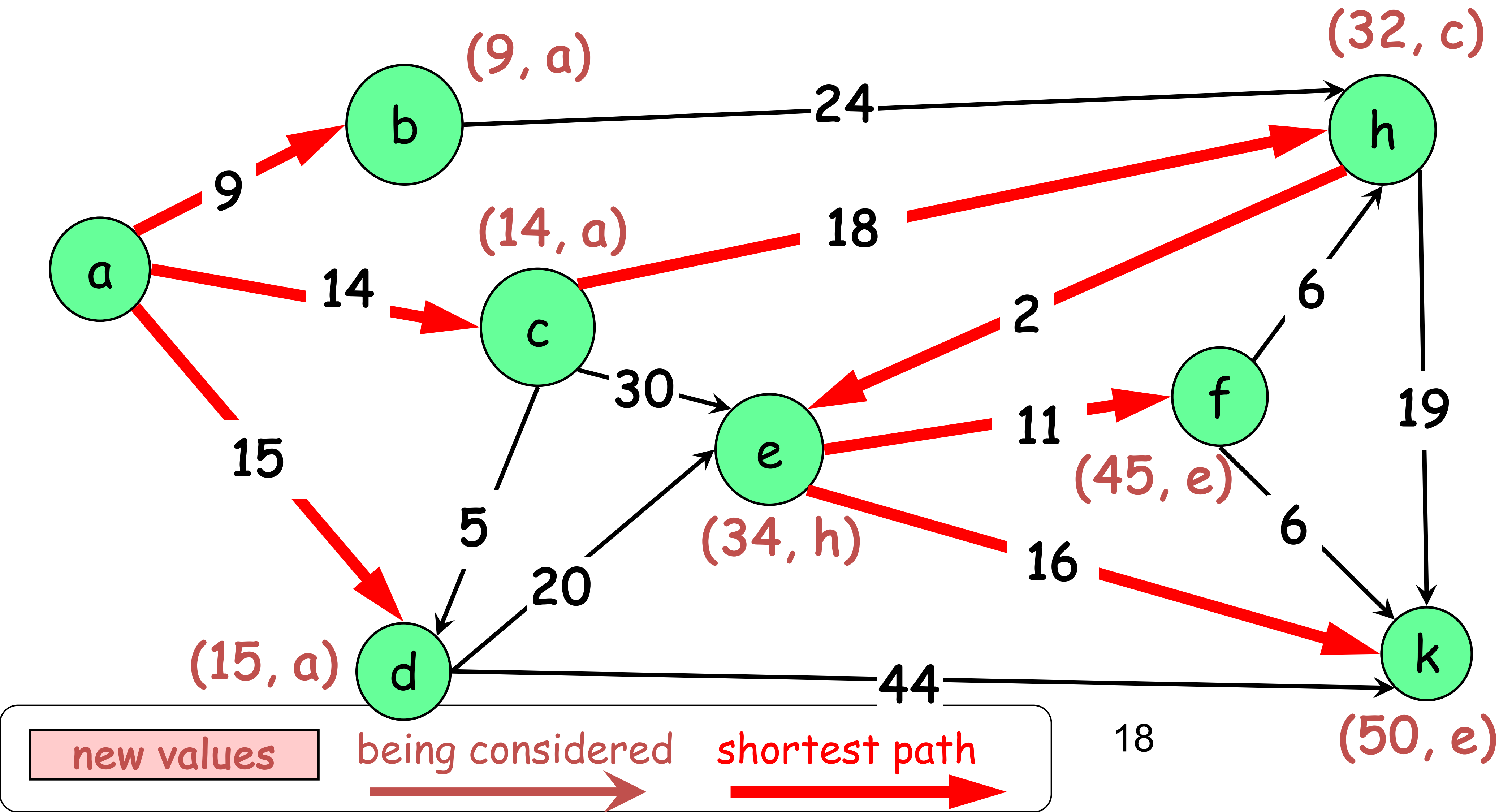
Dijkstra's algorithm

• After *f* is chosen, it is NOT necessary to update the weight of *k*. The final vertex chosen is *k*.



Dijkstra's algorithm

- At this point, all vertices are chosen, and the shortest path from *a* to every vertex is discovered.



Dijkstra's algorithm

- To describe the algorithm using pseudo code, we give some notations.
- Each vertex *v* is labelled with two labels:
  - a numeric label ***d(v)*** indicates the length of the shortest path from the source to *v* found so far
  - another label ***p(v)*** indicates next-to-last vertex on such path, i.e., the vertex immediately before *v* on that shortest path

Pseudo code

```
// Given a graph G=(V,E) and a source vertex s
for every vertex v in the graph do
    set d(v) = ∞ and p(v) = null
set d(s) = 0 and V_T = ∅
while V - V_T ≠ ∅ do        // there is still some vertex left
begin
    choose the vertex u in V - V_T with minimum d(u)
    set V_T = V_T ∪ { u }
    for every vertex v in V - V_T that is a neighbour of u do
        if d(u) + w(u,v) < d(v) then     // a shorter path is found
            set d(v) = d(u) + w(u,v) and p(v) = u
end
```

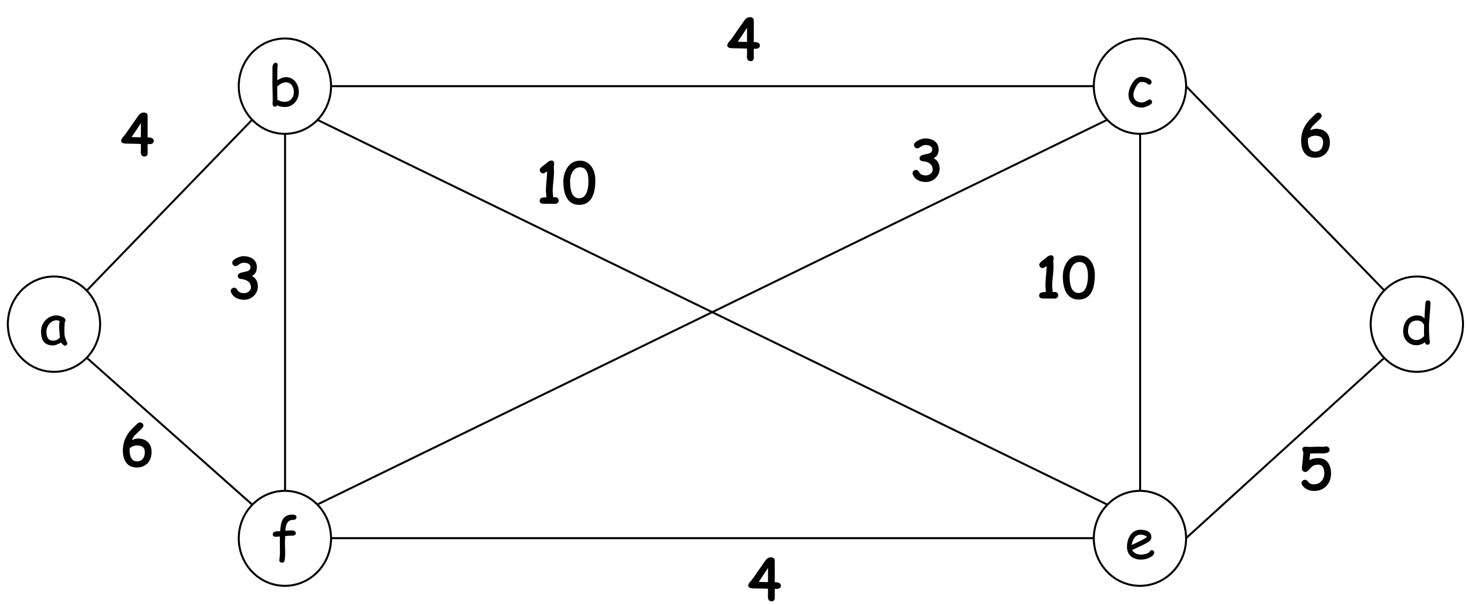
this should be ∅

https://www.youtube.com/watch?v=EFg3u_E6eHU&ab_channel=SpanningTree

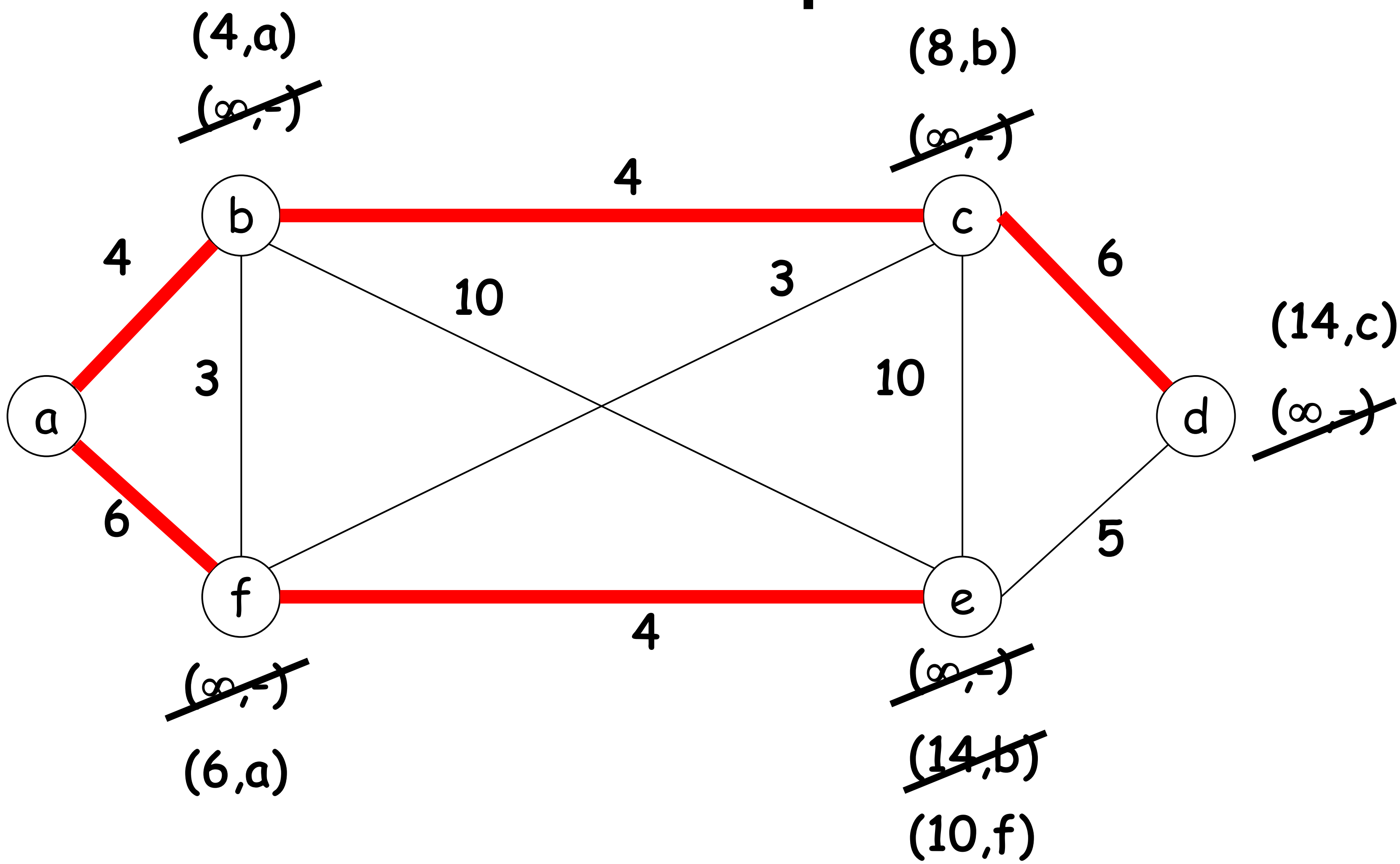
Example: Question 3 in Week6 Tutorial

Exercise

1. Find the shortest paths from vertex <u>*a*</u> to all other vertices



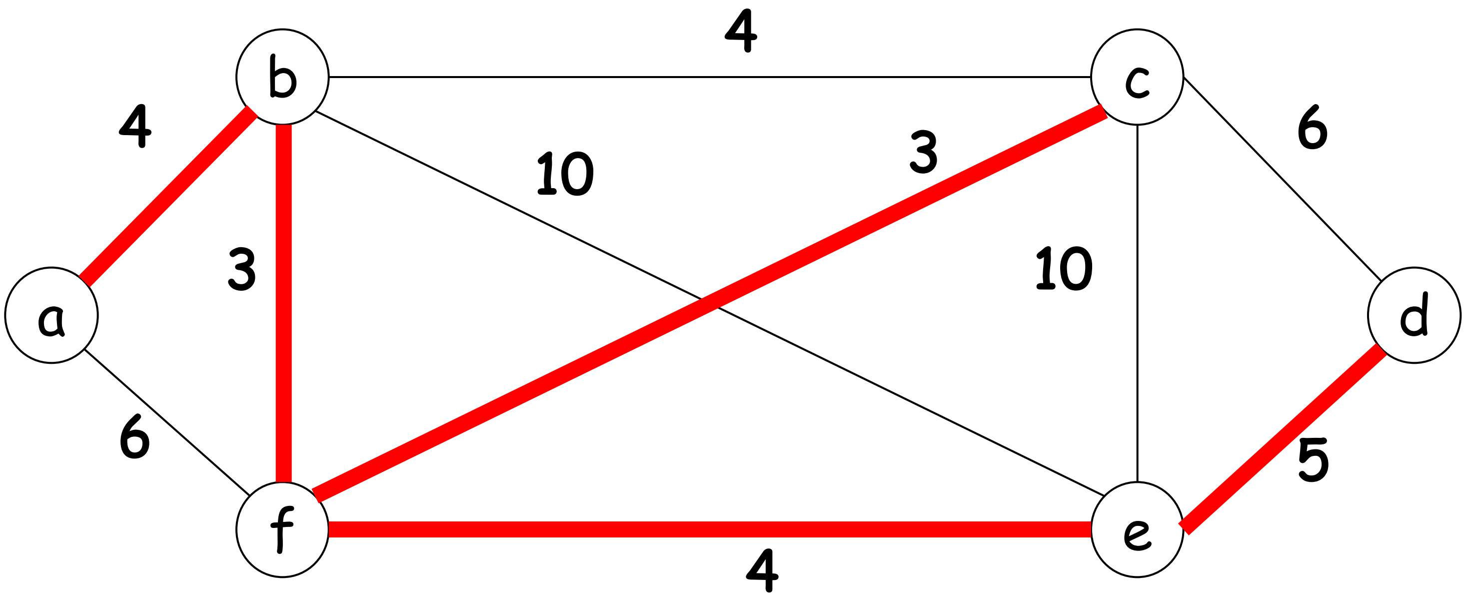
Exercise – Shortest paths from a



order of selection: (a,b), (a,f), (b,c), (f,e), (c,d)

Compare the solution with Exercise-MST

Exercise - MST



order of selection: (b,f), (c,f), (a,b), (f,e), (e,d)

Questions:

- Correctness: Does Dijkstra's algorithm always find the shortest paths?
- Complexity: what is the time complexity of Dijkstra's algorithm?

Correctness of Dijkstra's Algorithm

Observation (loop invariant):

Prior to each loop, for each $v \in S$, $d[v]$ is the length of the shortest path from s to v.

Complexity of Dijkstra's Algorithm (optional)

Depends on the data structure to represent the graph $G=(V, E)$ and the implementation of priority queue.

If G is represented by adjacency list and priority queue is implemented by min-heap, then the answer is

$$O(|E|\log|V|)$$

Learning outcomes

- ✓ Understand what greedy method is
- ✓ Able to apply Prim's algorithm to find minimum spanning tree
- ✓ Able to apply Kruskal's algorithm to find minimum spanning tree
- ✓ Able to apply Dijkstra's algorithm to find single-source shortest-paths

Does Greedy algorithm always return the best solution?

Knapsack Problem

- **Input:** Given n items with weights $w_1, w_2, \ldots, w_n$ and values $v_1, v_2, \ldots, v_n$, and a knapsack with capacity W.
- **Output:** Find the most valuable subset of items that can fit into the knapsack.
- **Application:** A transport plane is to deliver the most valuable set of items to a remote location without exceeding its capacity.

Example 1

w = 10
v = 60
item 1

w = 20
v = 100
item 2

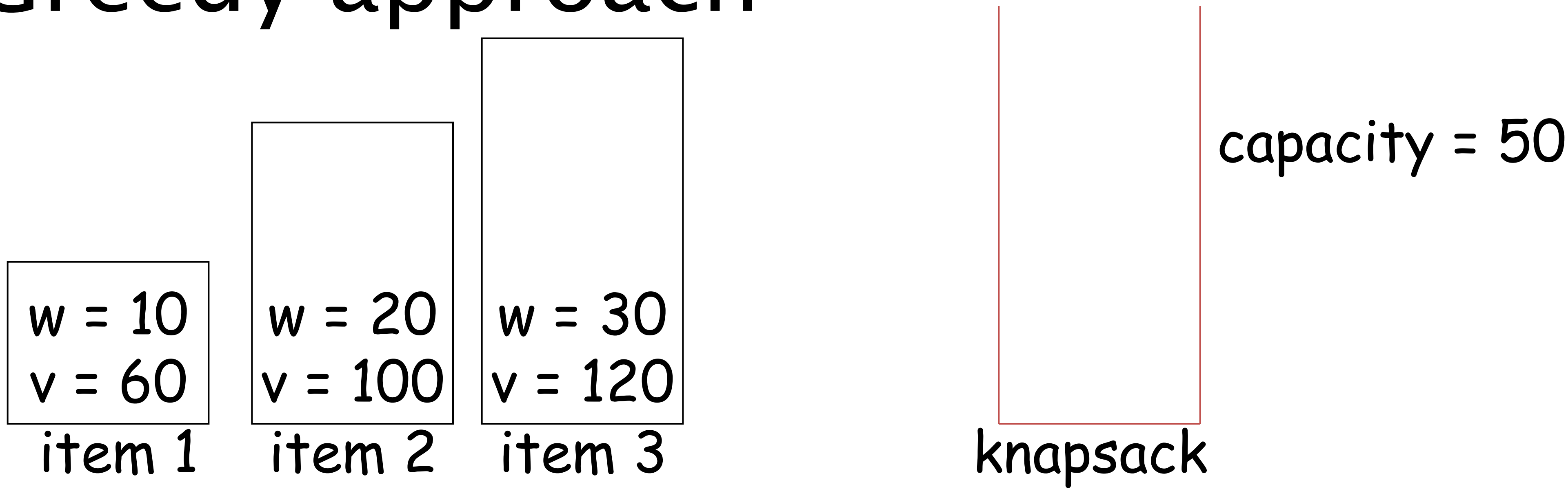
w = 30
v = 120
item 3

knapsack

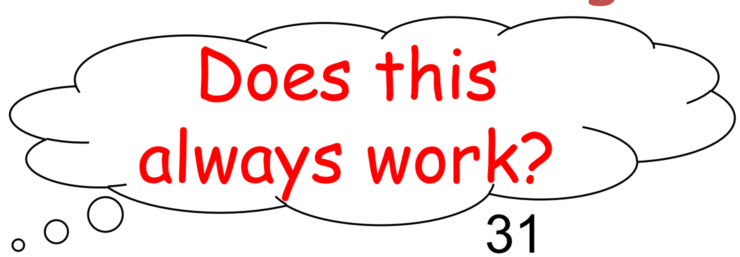
capacity = 50

subset	total weight	total value
ϕ	0	0
{1}	10	60
{2}	20	100
{3}	30	120
{1,2}	30	160
{1,3}	40	180
{2,3}	**50**	**220**
{1,2,3}	60	N/A

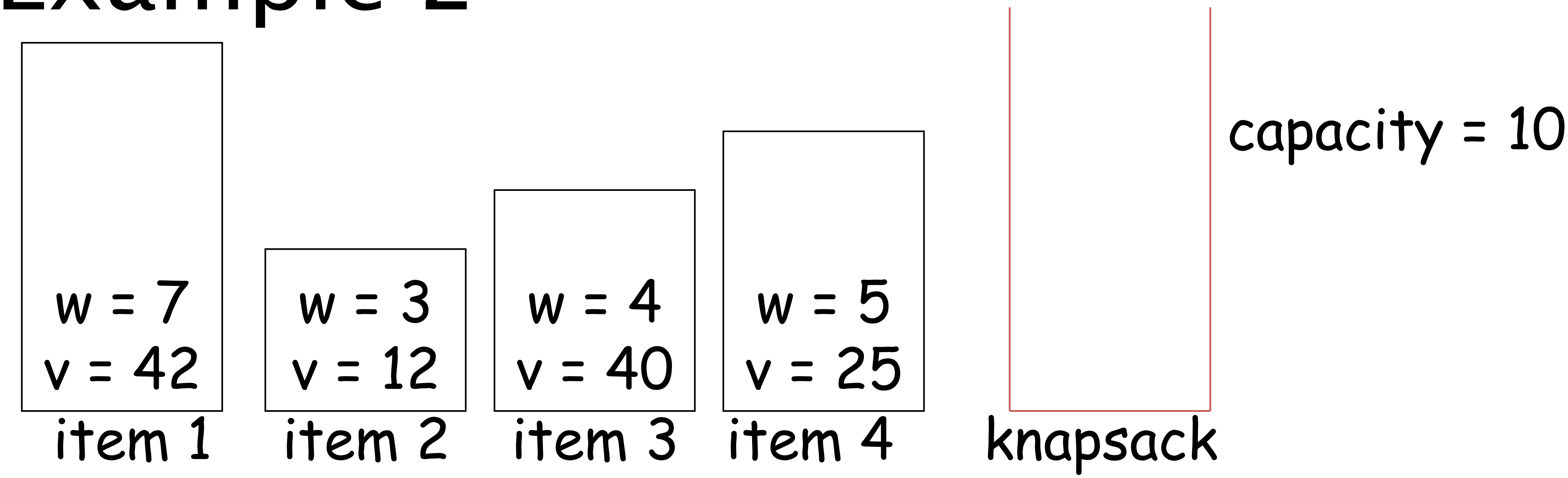
Greedy approach



- Greedy: pick the item with the next largest value if total weight <= capacity.
- Result:
  - item 3 is taken, total value = 120, total weight = 30
  - item 2 is taken, total value = 220, total weight = 50
  - item 1 cannot be taken



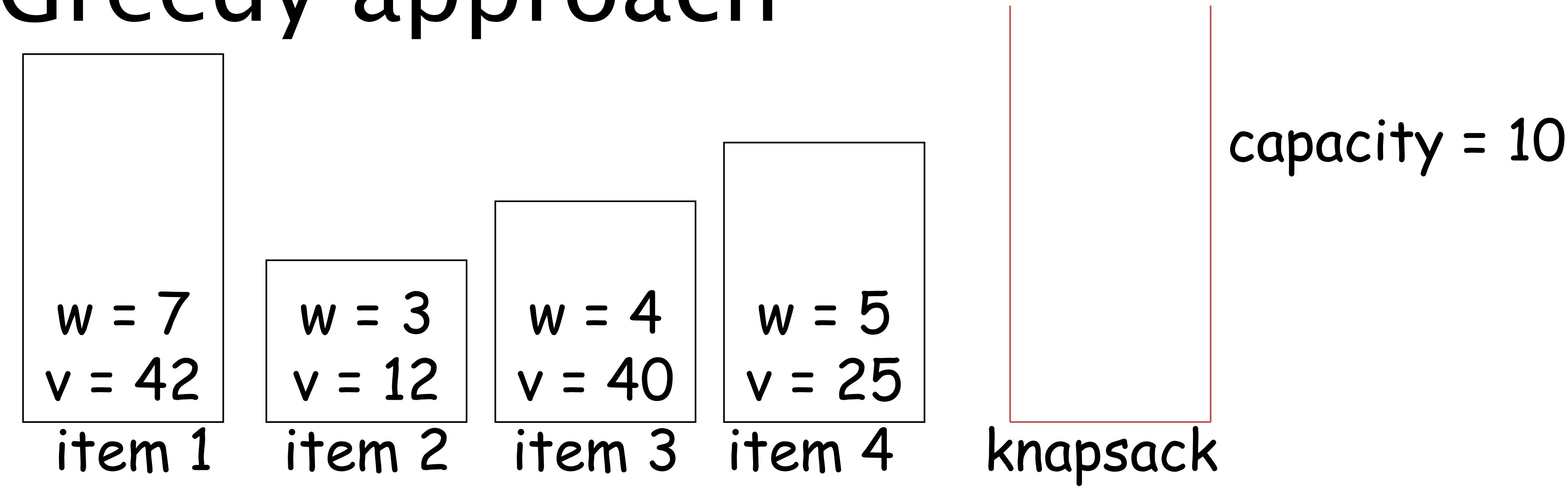
Example 2



subset	total weight	total value
ϕ	0	0
{1}	7	42
{2}	3	12
{3}	4	40
{4}	5	25
{1,2}	10	54
{1,3}	11	N/A
{1,4}	12	N/A

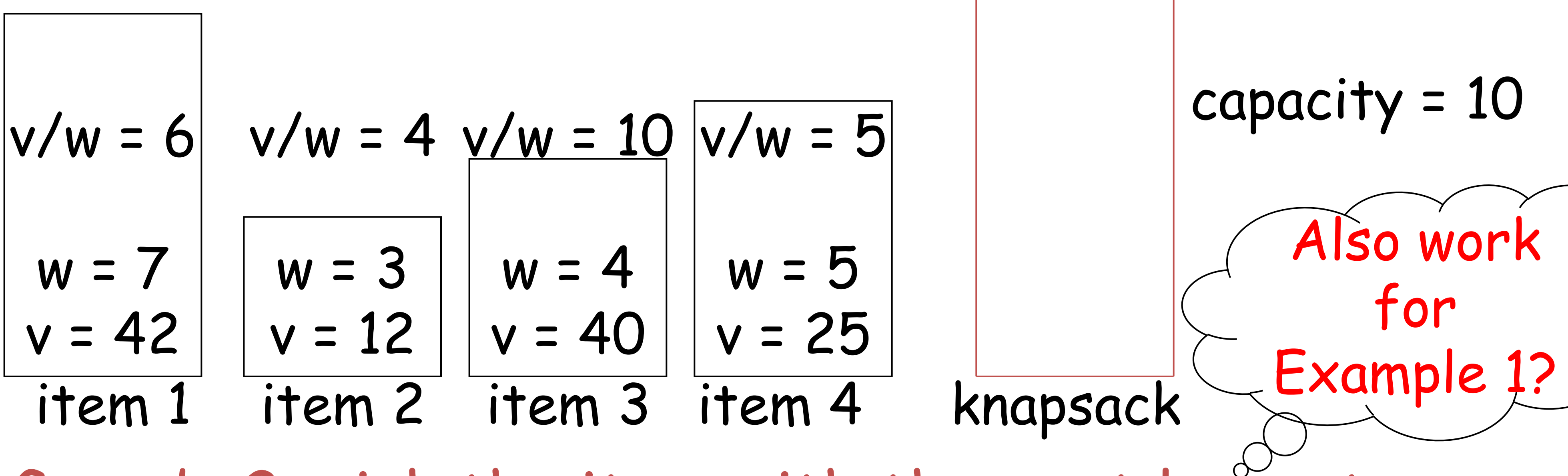
subset	total weight	total value
{2,3}	7	52
{2,4}	8	37
{3,4}	**9**	**65**
{1,2,3}	14	N/A
{1,2,4}	15	N/A
{1,3,4}	16	N/A
{2,3,4}	12	N/A
{1,2,3,4}	19	N/A

Greedy approach



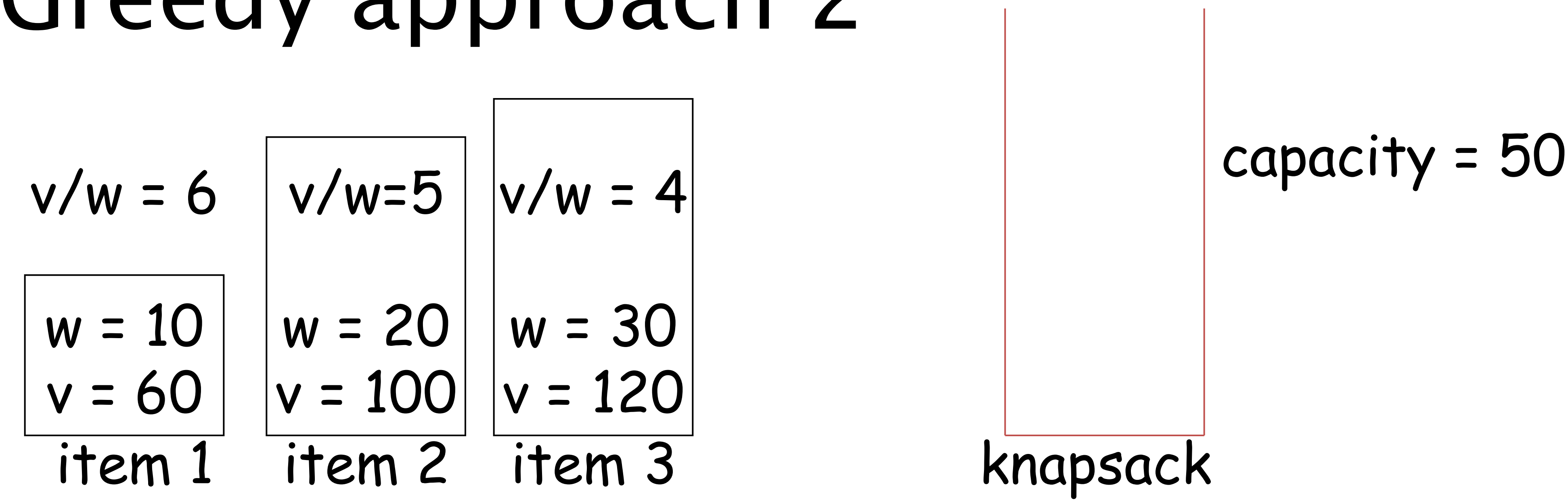
- Greedy: pick the item with the next largest value if total weight <= capacity.
- Result:
  - item 1 is taken, total value = 42, total weight = 7
  - item 3 cannot be taken
  - item 4 cannot be taken
  - item 2 is taken, total value = 54, total weight = 10

Greedy approach 2



- Greedy 2: pick the item with the next largest value/weight if total weight <= capacity.
- Result:
  - item 3 is taken, total value = 40, total weight = 4
  - item 1 cannot be taken
  - item 4 is taken, total value = 65, total weight = 9
  - item 2 cannot be taken

Greedy approach 2



- Greedy: pick the item with the next largest value/weight if total weight <= capacity.
- Result:
  - item 1 is taken, total value = 60, total weight = 10
  - item 2 is taken, total value = 160, total weight = 30
  - item 3 cannot be taken

Lesson Learned: Greedy algorithm does NOT always return the best solution