

COMP108 Algorithmic Foundations

Greedy methods

Prudence Wong

Coin Change Problem

Suppose we have 3 types of coins



10p



20p



50p

Minimum number of coins to make
£0.8, £1.0, £1.4 ?

Greedy method

2

(Greedy)

Learning outcomes

- Understand what greedy method is
- Able to apply Kruskal's algorithm to find minimum spanning tree
- Able to apply Dijkstra's algorithm to find single-source shortest-paths
- Able to apply greedy algorithm to find solution for Knapsack problem

3

(Greedy)

Greedy methods

How to be greedy?

- At every step, make the best move you can make
- Keep going until you're done

Advantages

- Don't need to pay much effort at each step
- Usually finds a solution very **quickly**
- The solution found is usually **not bad**

Possible problem

- The solution found may **NOT** be the best one

4

(Greedy)

Greedy methods - examples

Minimum spanning tree

- Kruskal's algorithm

Single-source shortest-paths

- Dijkstra's algorithm

Both algorithms find one of the BEST solutions

Knapsack problem

- greedy algorithm does NOT find the BEST solution

5

(Greedy)

Kruskal's algorithm ...

Minimum Spanning tree (MST)

Given an undirected connected graph G

- The edges are labelled by weight

Spanning tree of G

- a tree containing all vertices in G

Minimum spanning tree of G

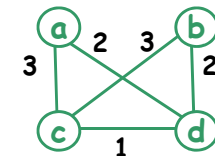
- a spanning tree of G with minimum weight

7

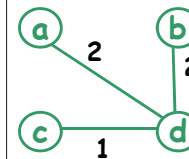
(Greedy)

Examples

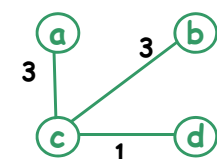
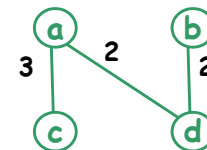
Graph G
(edge label is weight)



Spanning trees of G



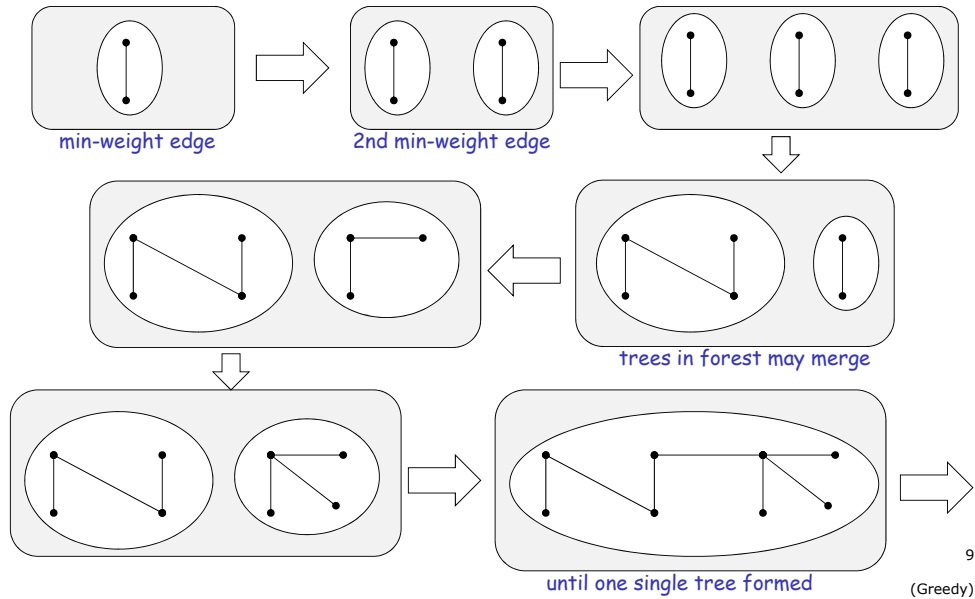
MST



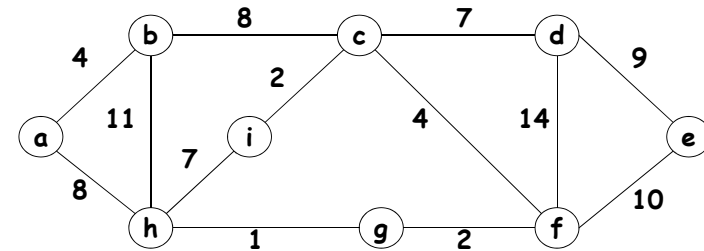
8

(Greedy)

Idea of Kruskal's algorithm - MST



Kruskal's algorithm - MST



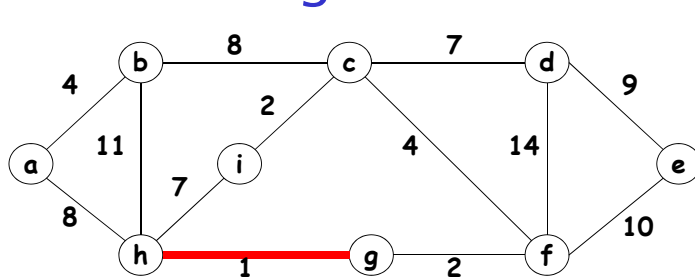
Arrange edges from smallest to largest weight

(h,g)	1
(i,c)	2
(g,f)	2
(a,b)	4
(c,f)	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

10

(Greedy)

Kruskal's algorithm - MST



Choose the minimum weight edge

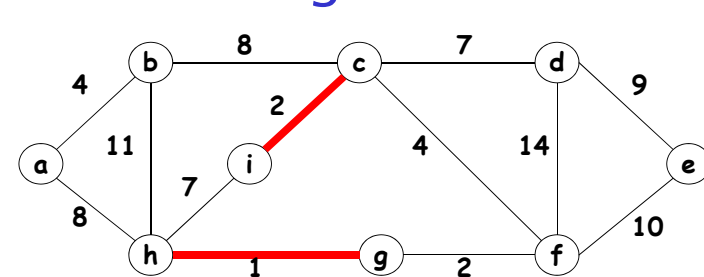
(h,g)	1
(i,c)	2
(g,f)	2
(a,b)	4
(c,f)	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

11

(Greedy)

Kruskal's algorithm - MST



Choose the next minimum weight edge

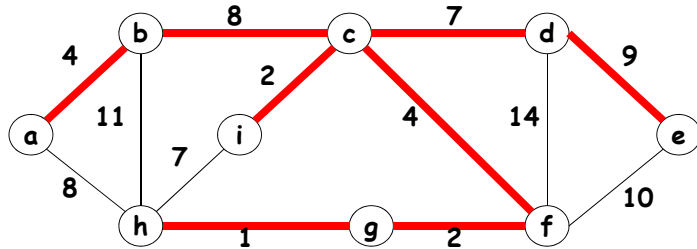
(h,g)	1
(i,c)	2
(g,f)	2
(a,b)	4
(c,f)	4
(c,d)	7
(h,i)	7
(b,c)	8
(a,h)	8
(d,e)	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen

12

(Greedy)

Kruskal's algorithm - MST



(f,e) cannot be included, otherwise, a cycle is formed

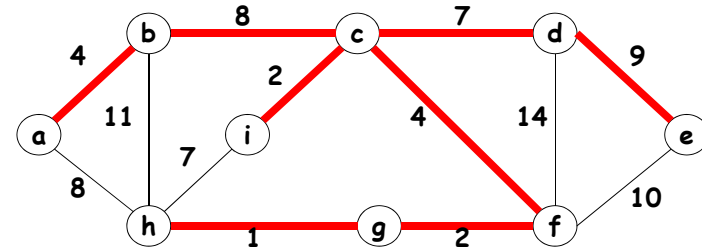


<i>(h,g)</i>	1
<i>(i,c)</i>	2
<i>(g,f)</i>	2
<i>(a,b)</i>	4
<i>(c,f)</i>	4
<i>(c,d)</i>	7
(h,i)	7
<i>(b,c)</i>	8
(a,h)	8
<i>(d,e)</i>	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen 21

(Greedy)

Kruskal's algorithm - MST



(b,h) cannot be included, otherwise, a cycle is formed

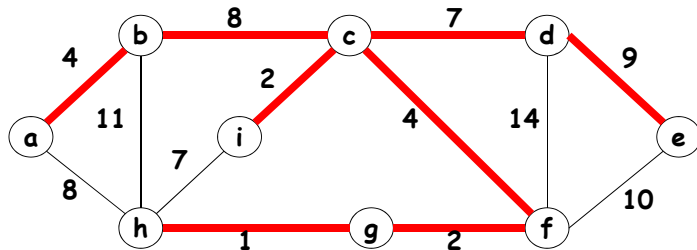


<i>(h,g)</i>	1
<i>(i,c)</i>	2
<i>(g,f)</i>	2
<i>(a,b)</i>	4
<i>(c,f)</i>	4
<i>(c,d)</i>	7
(h,i)	7
<i>(b,c)</i>	8
(a,h)	8
<i>(d,e)</i>	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen 22

(Greedy)

Kruskal's algorithm - MST



(d,f) cannot be included, otherwise, a cycle is formed

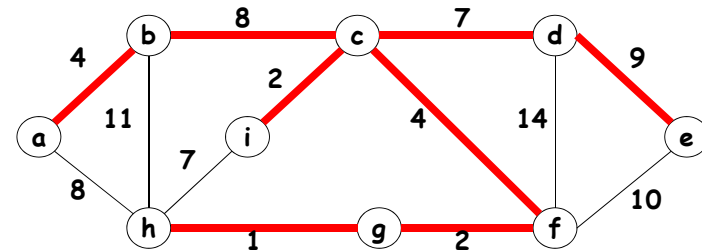


<i>(h,g)</i>	1
<i>(i,c)</i>	2
<i>(g,f)</i>	2
<i>(a,b)</i>	4
<i>(c,f)</i>	4
<i>(c,d)</i>	7
(h,i)	7
<i>(b,c)</i>	8
(a,h)	8
<i>(d,e)</i>	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen 23

(Greedy)

Kruskal's algorithm - MST



MST is found when all edges are examined

<i>(h,g)</i>	1
<i>(i,c)</i>	2
<i>(g,f)</i>	2
<i>(a,b)</i>	4
<i>(c,f)</i>	4
<i>(c,d)</i>	7
(h,i)	7
<i>(b,c)</i>	8
(a,h)	8
<i>(d,e)</i>	9
(f,e)	10
(b,h)	11
(d,f)	14

italic: chosen 24

(Greedy)

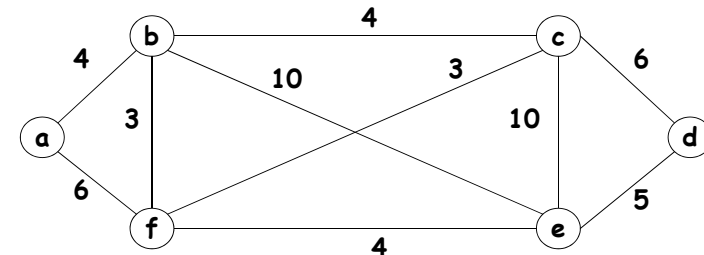
Kruskal's algorithm - MST

Kruskal's algorithm is **greedy** in the sense that it always attempt to select the **smallest** weight edge to be included in the MST

25

(Greedy)

Exercise – Find MST for this graph



order of (edges) selection:

26

(Greedy)

Pseudo code

// Given an undirected connected graph $G=(V,E)$

$T = \emptyset$ and $E' = E$

while $E' \neq \emptyset$ **do**

begin

pick an edge e in E' with minimum weight

if adding e to T does not form cycle **then**

add e to T , i.e., $T = T \cup \{e\}$

remove e from E' , i.e., $E' = E' \setminus \{e\}$

end

Time complexity?

Can be tested by
marking vertices

27

(Greedy)

Dijkstra's algorithm ...

Single-source shortest-paths

Consider a (un)directed connected graph G

- The edges are labelled by weight

Given a particular vertex called the **source**

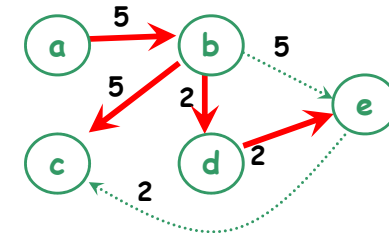
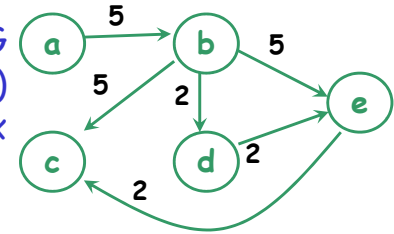
- Find **shortest paths** from the source to all other vertices (shortest path means the total weight of the path is the smallest)

29

(Greedy)

Example

Directed Graph G
(edge label is weight)
a is source vertex



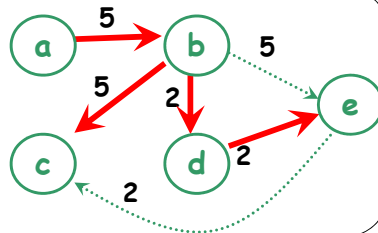
thick lines: shortest path
dotted lines: not in shortest path

30

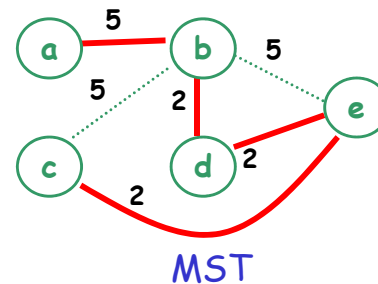
(Greedy)

Single-source shortest paths vs MST

Shortest paths from a



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



MST

31

(Greedy)

Algorithms for shortest paths

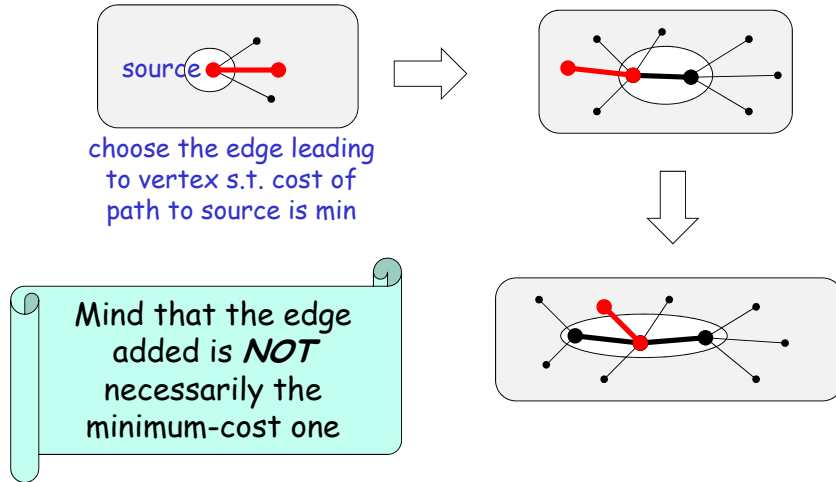
Algorithms

- there are many algorithms to solve this problem, one of them is **Dijkstra's** algorithm, which assumes the weights of edges are **non-negative**

32

(Greedy)

Idea of Dijkstra's algorithm



33

(Greedy)

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:

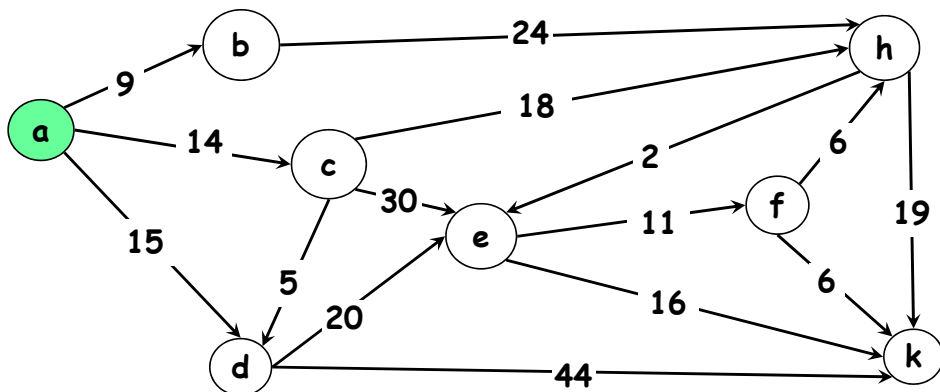
- in the i -th iteration, the vertex which is the i -th closest to s is found,
- for every remaining vertices, the current shortest path to s found so far (this shortest path will be updated as the algorithm runs)

34

(Greedy)

Dijkstra's algorithm

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

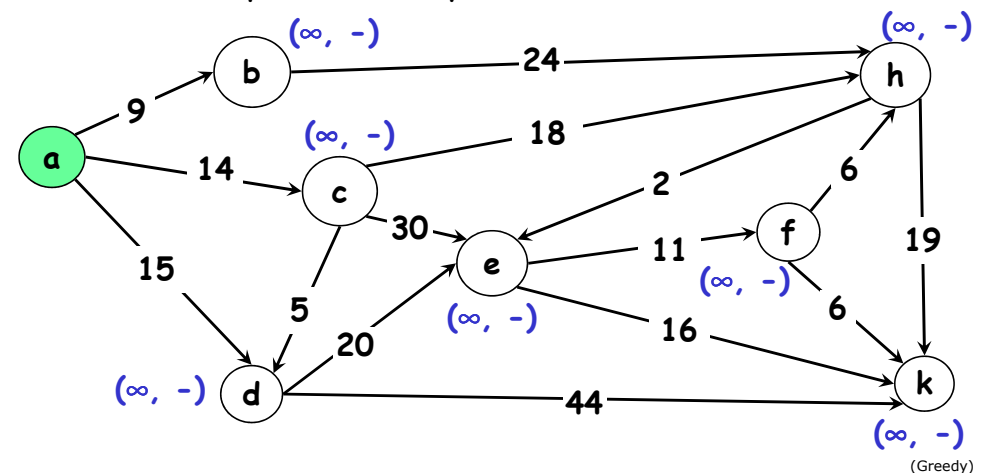


35

(Greedy)

Dijkstra's algorithm

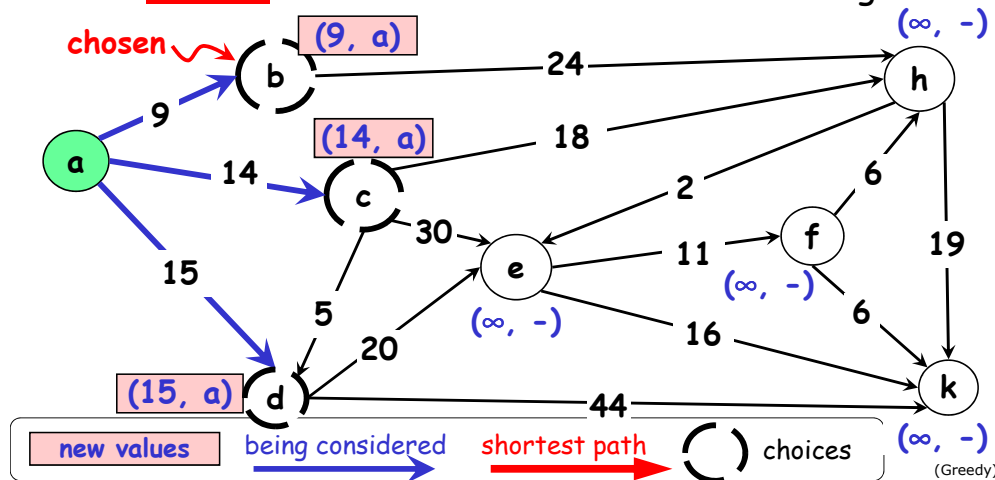
Every vertex v keeps 2 labels: (1) the weight of the current shortest path from a ; (2) the vertex leading to v on that path, initially as $(\infty, -)$



(Greedy)

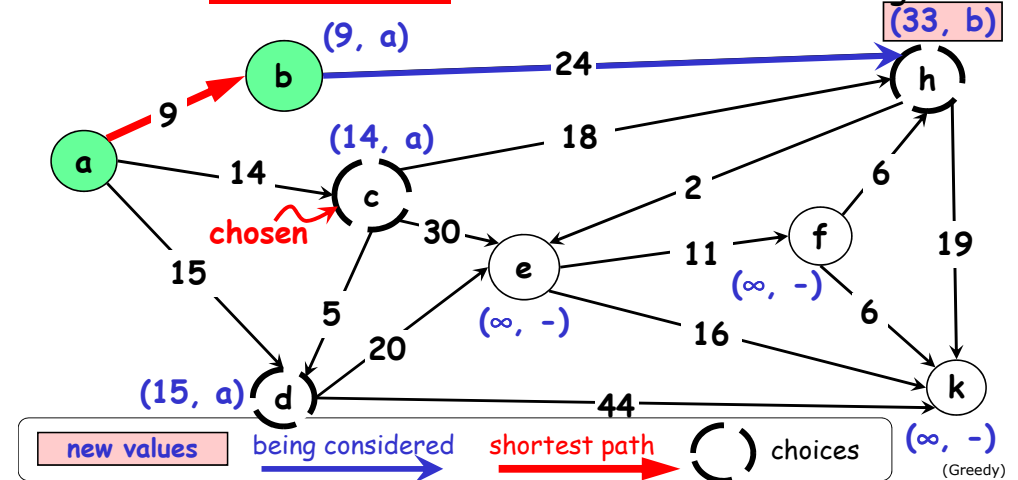
Dijkstra's algorithm

For every neighbor u of a , update the weight to the weight of (a, u) and the leading vertex to a . Choose from b, c, d the one with the smallest such weight.



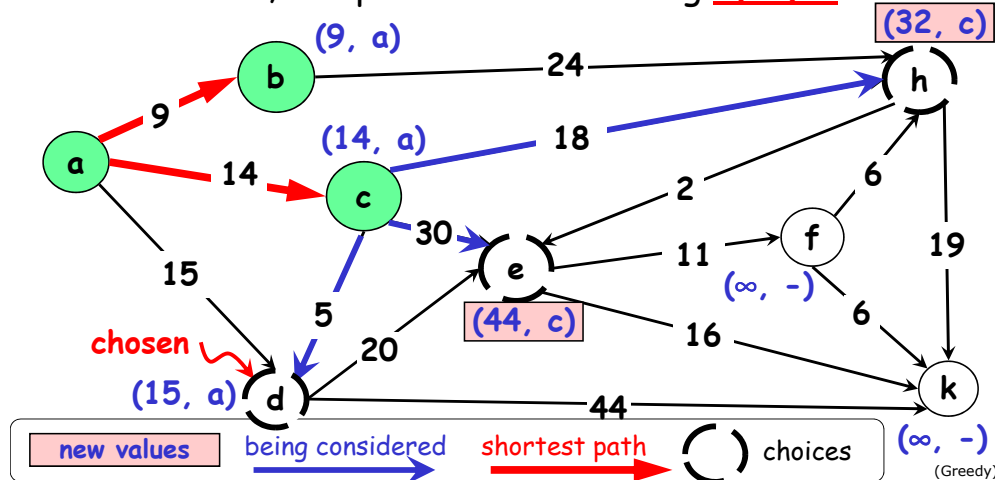
Dijkstra's algorithm

For every un-chosen neighbor of vertex b , update the weight and leading vertex. Choose from **ALL** un-chosen vertices (*i.e., c, d, h*) the one with smallest weight.



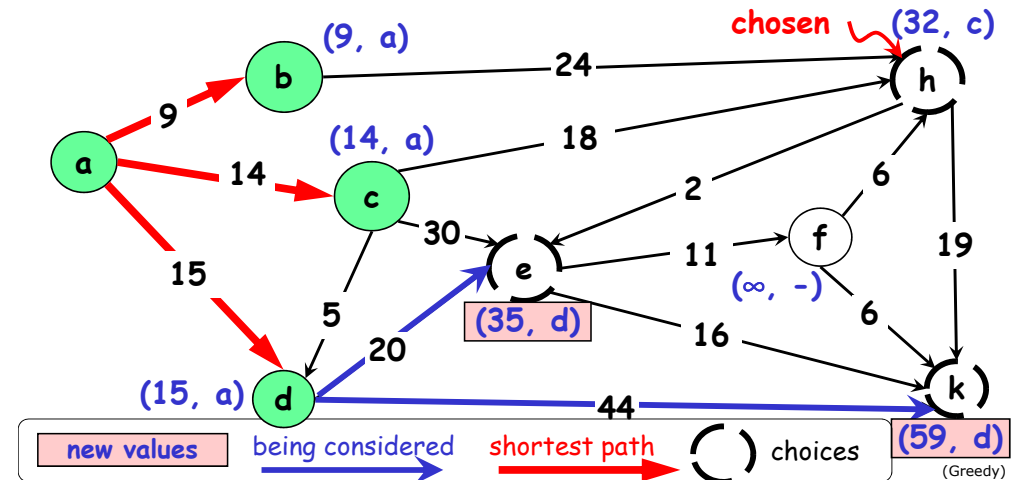
Dijkstra's algorithm

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



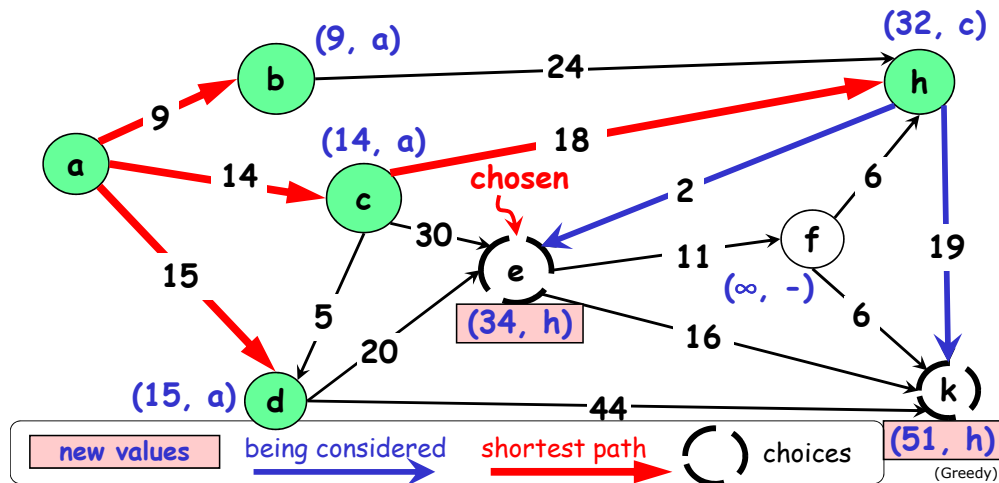
Dijkstra's algorithm

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



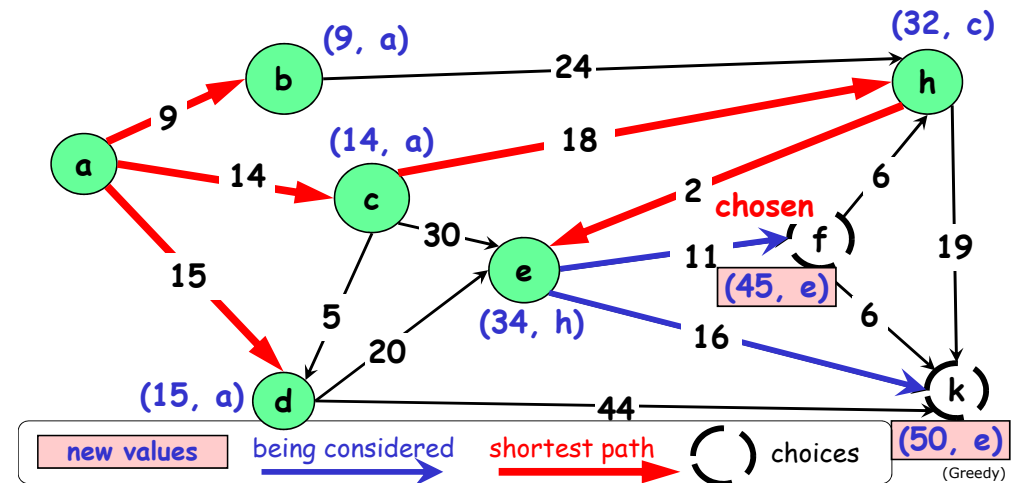
Dijkstra's algorithm

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



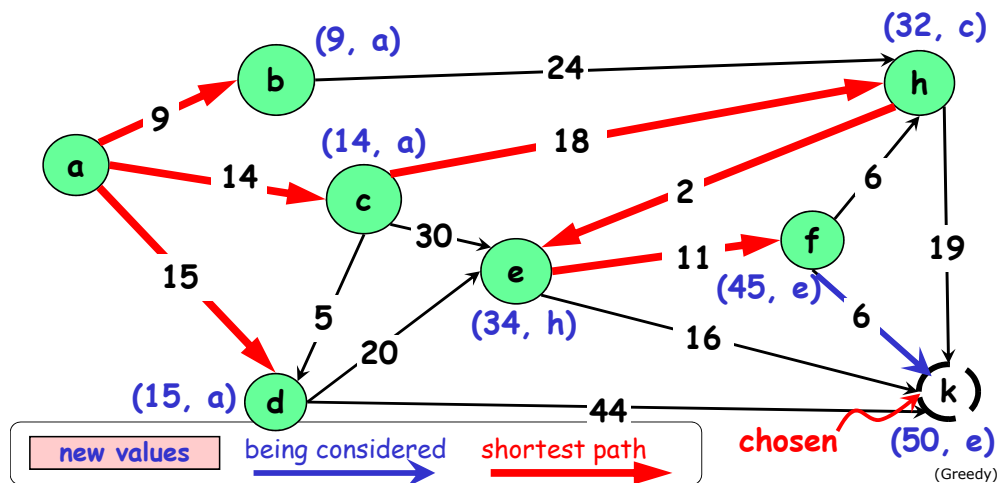
Dijkstra's algorithm

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



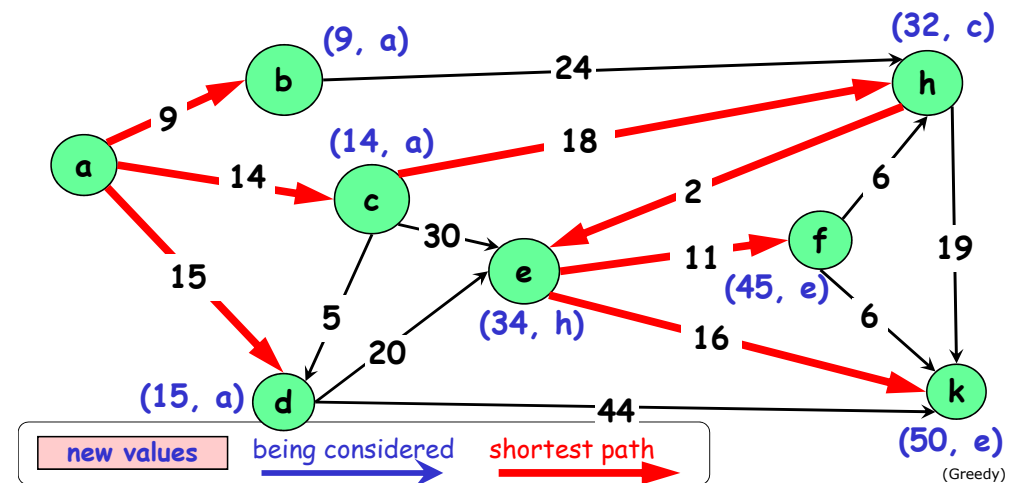
Dijkstra's algorithm

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

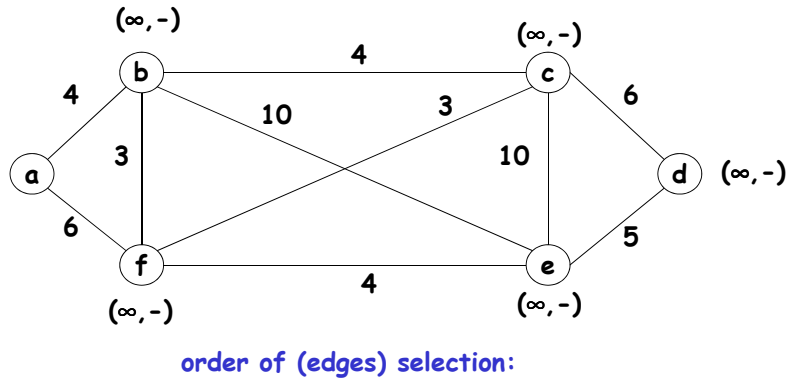


Dijkstra's algorithm

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



Exercise – Shortest paths from a



45

(Greedy)

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

46

(Greedy)

Pseudo code

```
// Given a graph  $G=(V,E)$  and a source vertex  $s$ 
for every vertex  $v$  in the graph do
    set  $d(v) = \infty$  and  $p(v) = \text{null}$ 
set  $d(s) = 0$  and  $V_T = \emptyset$ 
while  $V \setminus V_T \neq \emptyset$  do    // there is still some vertex left
begin
    choose the vertex  $u$  in  $V \setminus V_T$  with minimum  $d(u)$ 
    set  $V_T = V_T \cup \{u\}$ 
    for every vertex  $v$  in  $V \setminus V_T$  that is a neighbor 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
```

Time complexity?

47

(Greedy)

Does Greedy algorithm always return the best solution?

Knapsack Problem

Input: Given n items with weights w_1, w_2, \dots, w_n and values v_1, v_2, \dots, v_n , and a knapsack with capacity W .

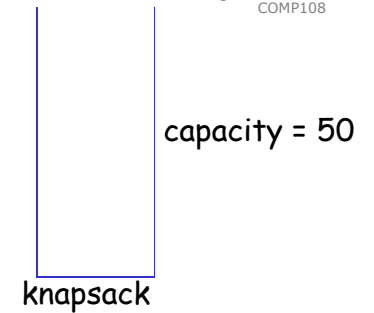
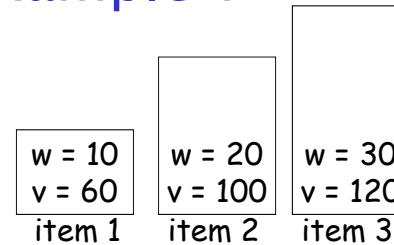
Output: Find the most valuable subset of items that can fit into the knapsack

Application: A transport plane is to deliver the most valuable set of items to a remote location without exceeding its capacity

49

(Greedy)

Example 1

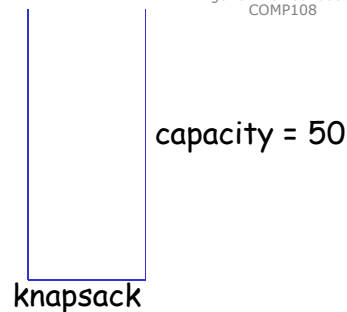
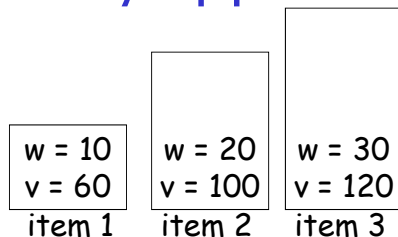


subset	total weight	total value
ϕ	0	0
{1}	10	60
{2}	20	100
{3}	30	120
{1,2}	30	160
{1,3}	40	180
{2,3}	50	220
{1,2,3}	60	N/A

50

(Greedy)

Greedy approach



Greedy: pick the item with the next largest value if total weight \leq 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

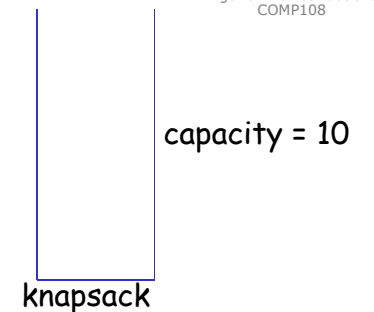
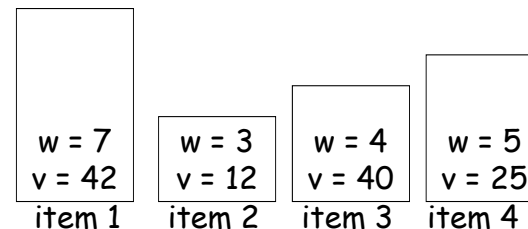
Time complexity?

Does this always work?

51

(Greedy)

Example 2

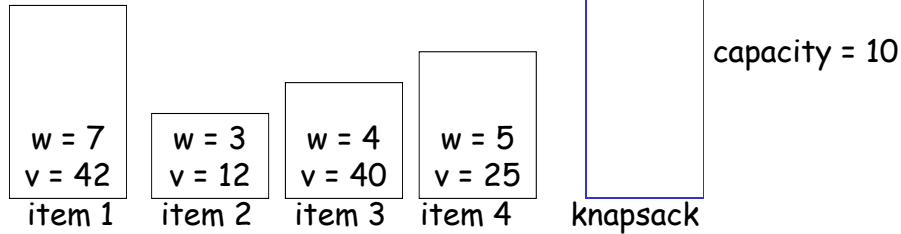


subset	total weight	total value	subset	total weight	total value
ϕ	0	0	{2,3}	7	52
{1}	7	42	{2,4}	8	37
{2}	3	12	{3,4}	9	65
{3}	4	40	{1,2,3}	14	N/A
{4}	5	25	{1,2,4}	15	N/A
{1,2}	10	54	{1,3,4}	16	N/A
{1,3}	11	N/A	{2,3,4}	12	N/A
{1,4}	12	N/A	{1,2,3,4}	19	N/A

52

(Greedy)

Greedy approach



Greedy: pick the item with the next largest value if total weight \leq capacity.

Result:

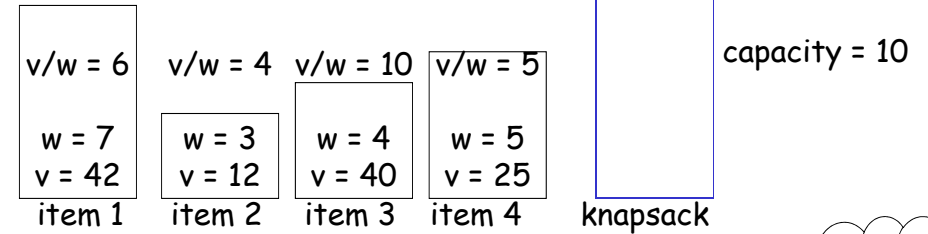
- item 1 is taken, total value = 42, total weight = 7
- item 3 cannot be taken
- item 4 cannot be taken
- item 2 is taken, total value = 54, total weight = 10

not the best!!

53

(Greedy)

Greedy approach 2



Greedy 2: pick the item with the next largest (value/weight) if total weight \leq capacity.

Result:

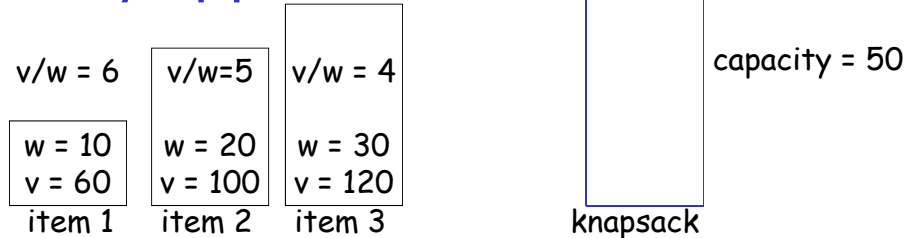
- item 3 is taken, total value = 40, total weight = 4
- item 1 cannot be taken
- item 4 is taken, total value = 65, total weight = 9
- item 2 cannot be taken

Work for Eg 1?

54

(Greedy)

Greedy approach 2



Greedy: pick the item with the next largest (value/weight) if total weight \leq 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

Not the best!!

55

(Greedy)

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