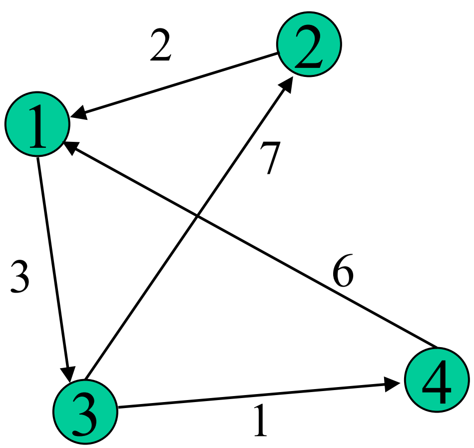
Recall: Floyd's Algorithm: All pairs shortest paths

In a weighted graph, find shortest paths between every pair of vertices

Same idea: construct solution through series of matrices $D^{(0)}$, $D^{(1)}$, … using an initial subset of the vertices as intermediaries.



	1	2	3	4
1	0	∞	3	∞
2	2	0	∞	∞
3	∞	7	0	1
4	6	∞	∞	0

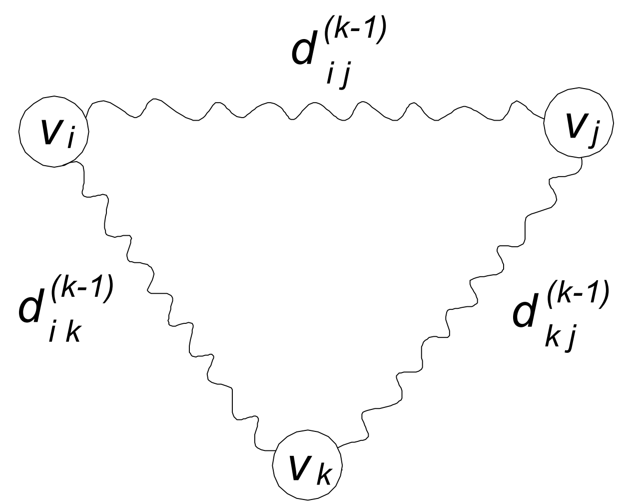
Weight matrix

	1	2	3	4
1	0	10	3	4
2	2	0	5	6
3	7	7	0	1
4	6	16	9	0

Distance matrix

Similar to Warshall's Algorithm

$d_{ij}^{(k)}$ **in** $D^{(k)}$ **is equal to the length of shortest path among all paths from the *i*th vertex to *j*th vertex with each intermediate vertex, if any, numbered not higher than *k***



$$d_{ij}^{(k)} = \min\{d_{ij}^{(k-1)}, d_{ik}^{(k-1)} + d_{kj}^{(k-1)}\} \textbf{ for } k \geq 1, d_{ij}^{(0)} = w_{ij}$$

Pseudocode of Floyd's Algorithm

The next matrix in sequence can be written over its predecessor

ALGORITHM $Floyd(W[1..n,1..n])$

$D \leftarrow W$

for $k \leftarrow 1$ **to** n **do**

  for $i \leftarrow 1$ **to** n **do**

    for $j \leftarrow 1$ **to** n **do**

      $D[i,j] \leftarrow \min\{D[i,j], D[i,k] + D[k,j]\}$

return D

Chapter 9: Greedy algorithms

Change-making problem

- Coin-system in US: 25(quarter), 10 (dime), 5(nickel), 1(penny)
- If you need to give a change of 48 cents using coins,
- 48 cents = 1 quarter + 2 dimes + 3 pennies
- This is a greedy algorithm: reduce the amount in the fastest way

The greedy approach constructs a solution through a sequence of steps until a complete solution is reached, On each step, the choice made must be

- *Feasible:* Satisfy the problem’s constraints
- *locally optimal:* the best choice
- *Irrevocable:* Once made, it cannot be changed later

Minimum Spanning Tree (MST)

Motivation: Planning the layout of cables or water pipes with the minimum length to cover all houses in a community

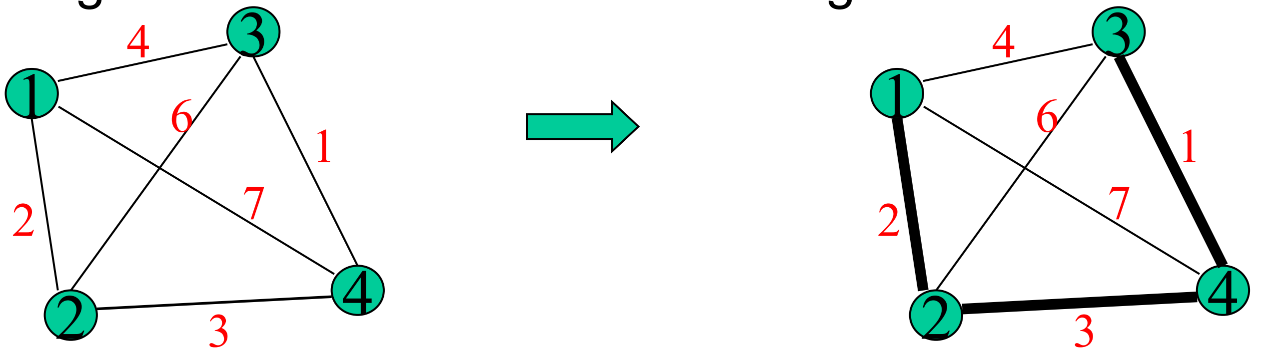
→ a tree structure (a connected acyclic graph)

Spanning tree of a connected graph *G*

- A connected acyclic subgraph of *G* that includes all of *G*'s vertices.
- At least one spanning tree exists for *G*.

Minimum Spanning Tree of a weighted, connected graph *G*:

- a spanning tree of *G* of minimum total weight.



Prim’s MST algorithm

Start with tree consisting of one vertex

“Grow” tree one vertex/edge at a time to produce MST

- Construct a series of expanding subtrees T_1, T_2, …

Greedy step: at each stage construct T_{i+1} from T_i: add an edge with minimum weight connecting a vertex in tree (T_i) to one not yet in tree

For all vertices that are not yet in the tree, we have two groups

- Fringe nodes: has an edge to at least one node in current tree T_i
- unseen nodes: no edge to any node in T_i

A priority queue is used

- The node with highest priority will be select
- The priority queue will be updated every time when a new vertex is added

Algorithm stops when all vertices are included

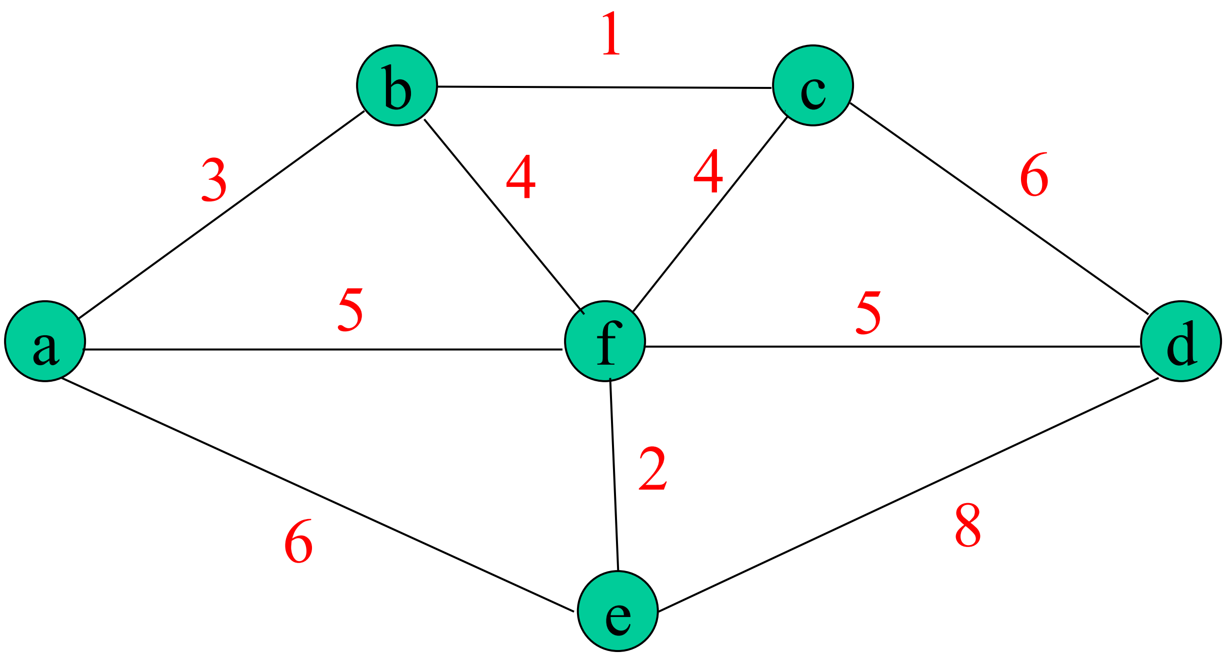
Prim's MST algorithm

ALGORITHM *Prim*(G)

//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \emptyset$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u)
    such that v is in V_T and u is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

An Example:

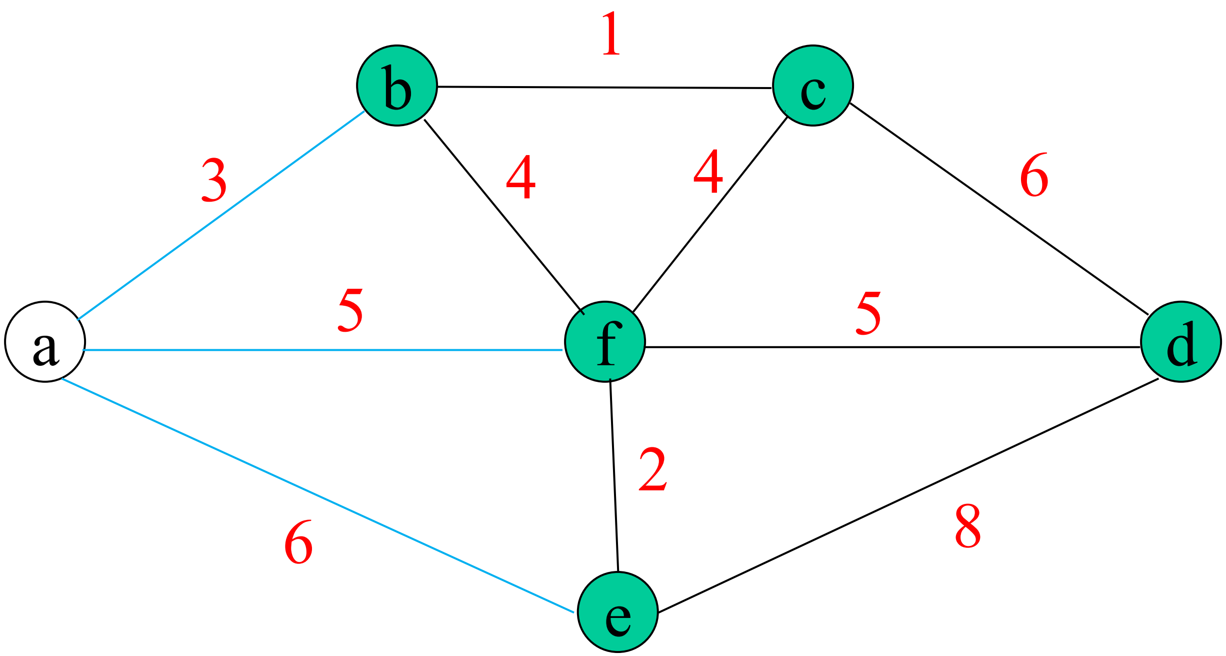
Finding the MST of the following graph using Prim's algorithm



Step 1:

Start from empty tree *T*, pick one vertex, a(-,-) and add it to *T*

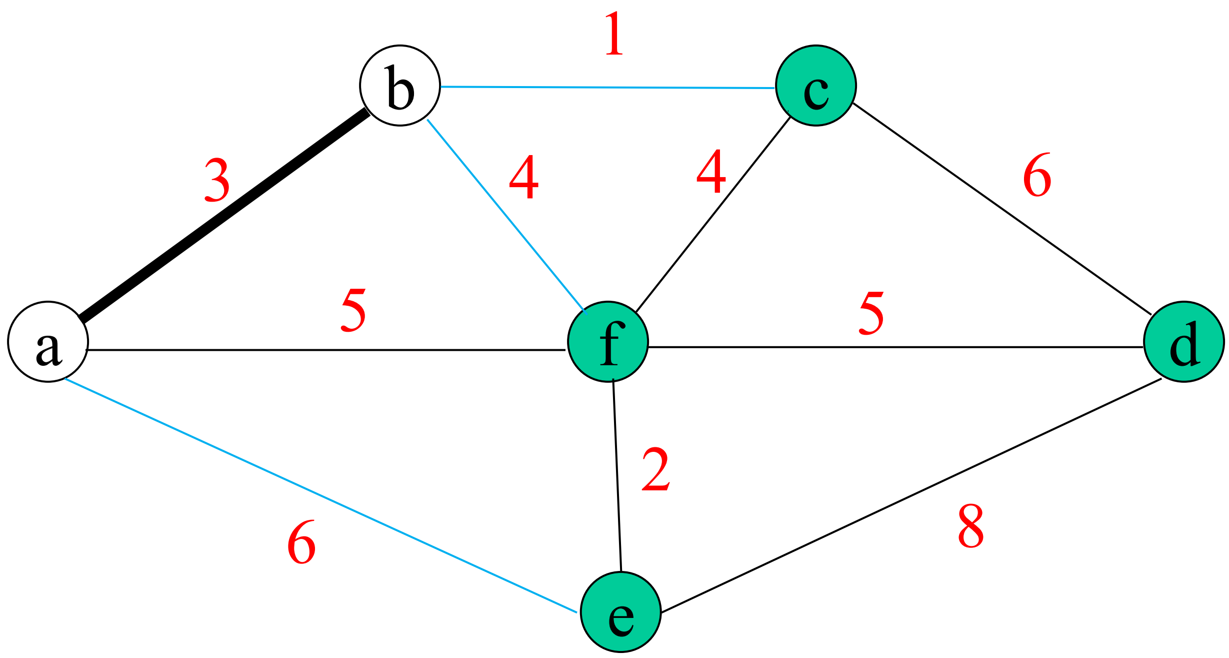
Priority queue: b(a,3), f(a,5), e(a,6), c(-,∞), d(-,∞)



Step 2:

Add the minimum-weight fringe edge b(a,3) into *T*

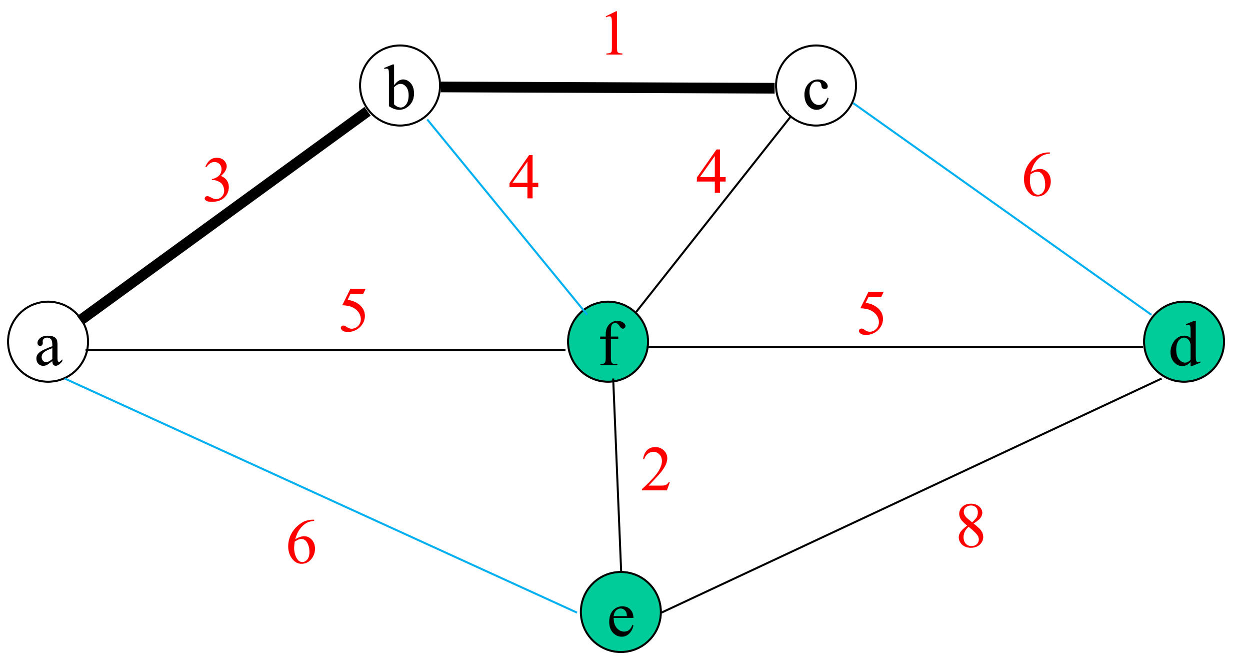
Priority queue: c(b,1), f(b,4), e(a,6), d(-,∞)



Step 3:

Add the minimum-weight fringe edge c(b,1) into *T*

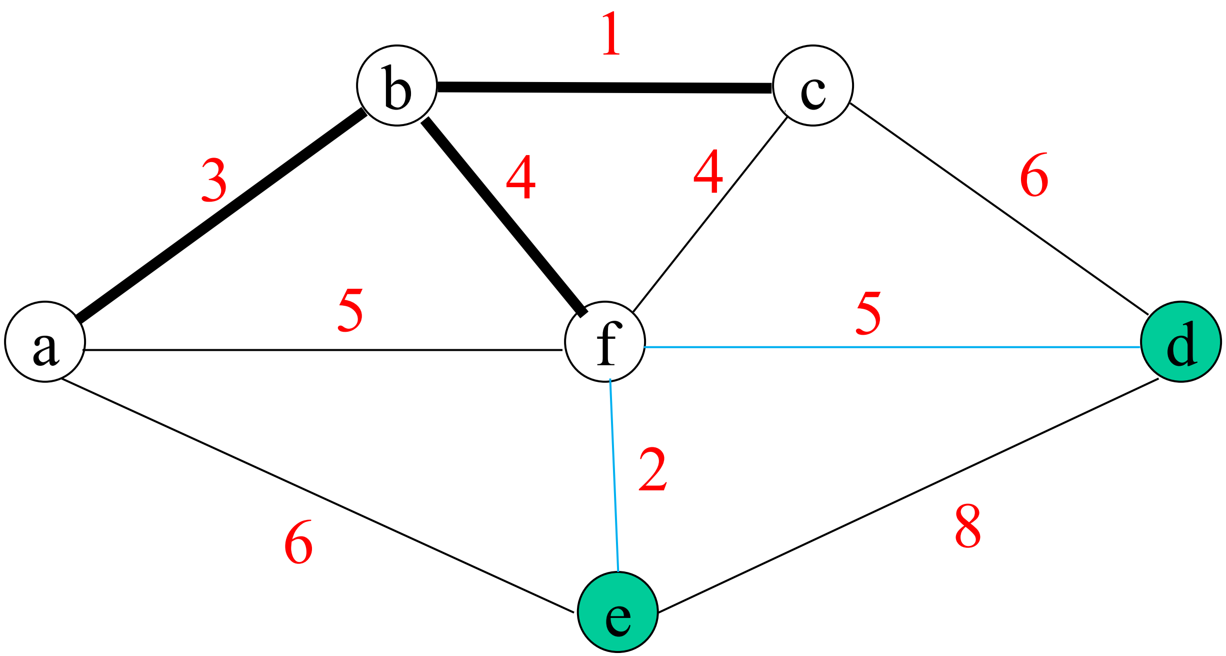
Priority queue: f(b,4), d(c,6), e(a,6)



Step 4:

Add the minimum-weight fringe edge f(b,4) into *T*

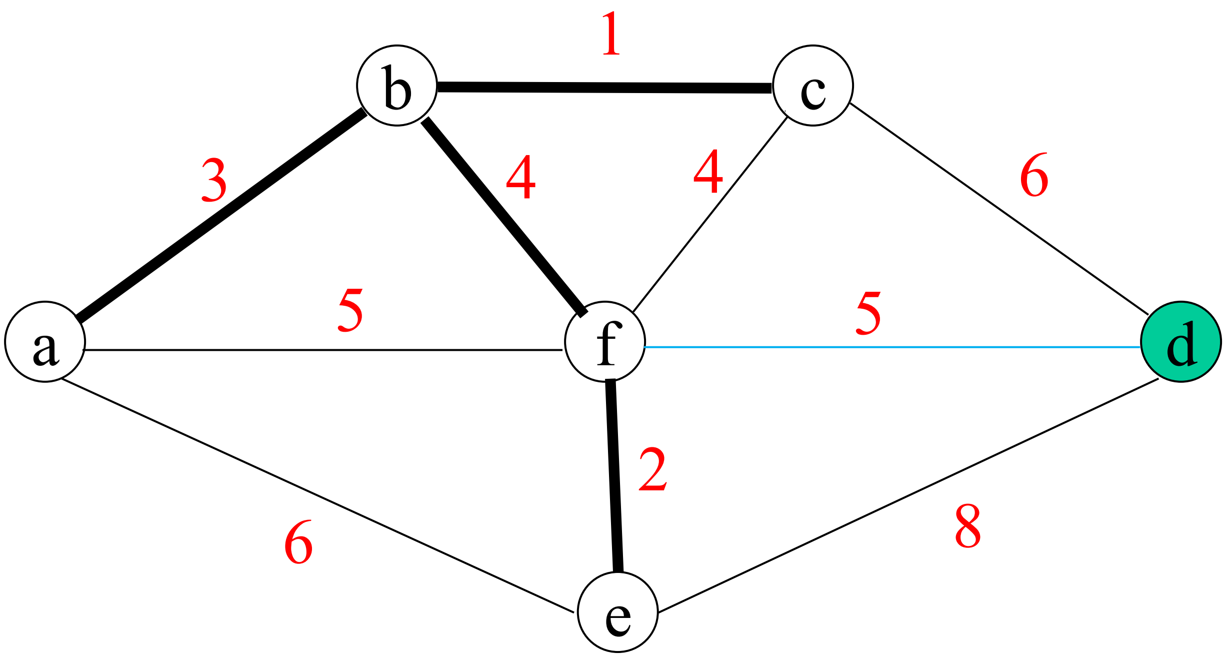
Priority queue: e(f,2), d(f,5)



Step 5:

Add the minimum-weight fringe edge e(f,2) into *T*

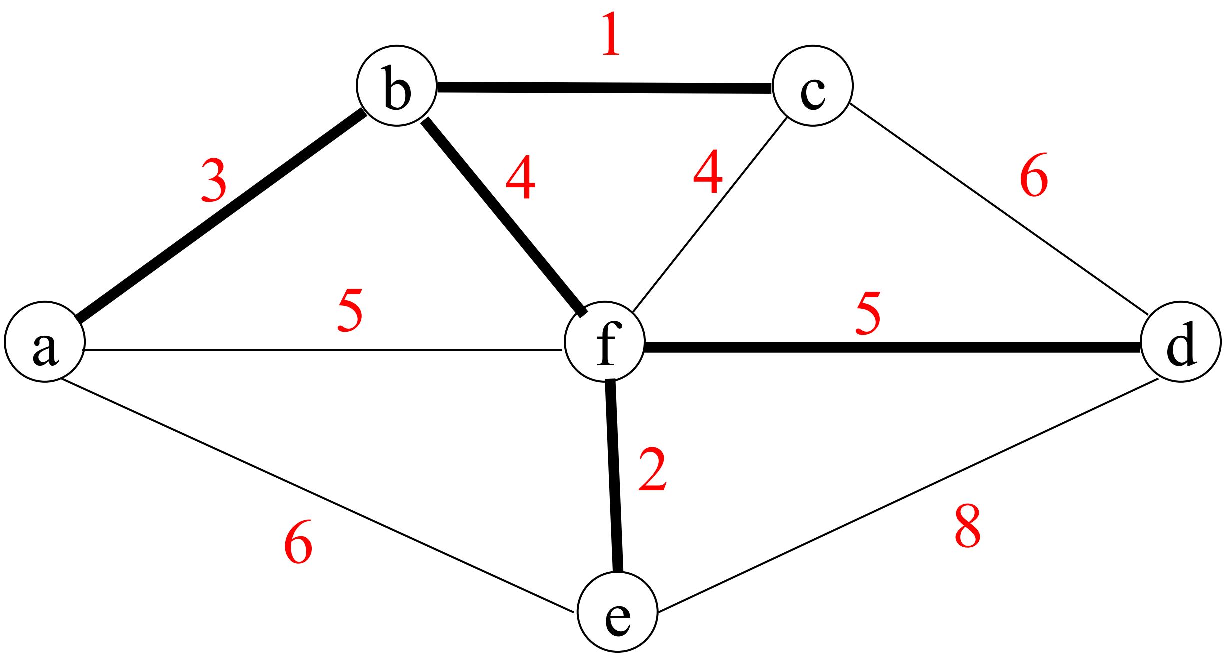
Priority queue: d(f,5)



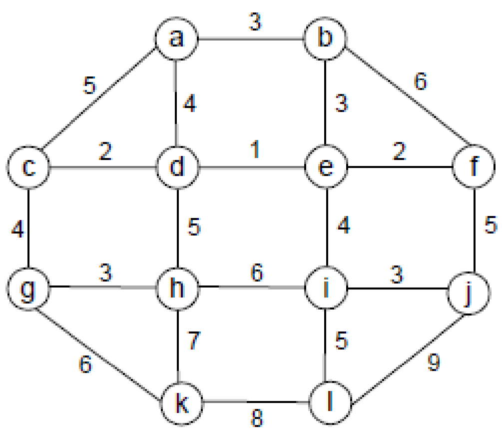
Step 6:

Add the minimum-weight fringe edge d(f,5) into *T*

No remaining vertices and the algorithm is done!

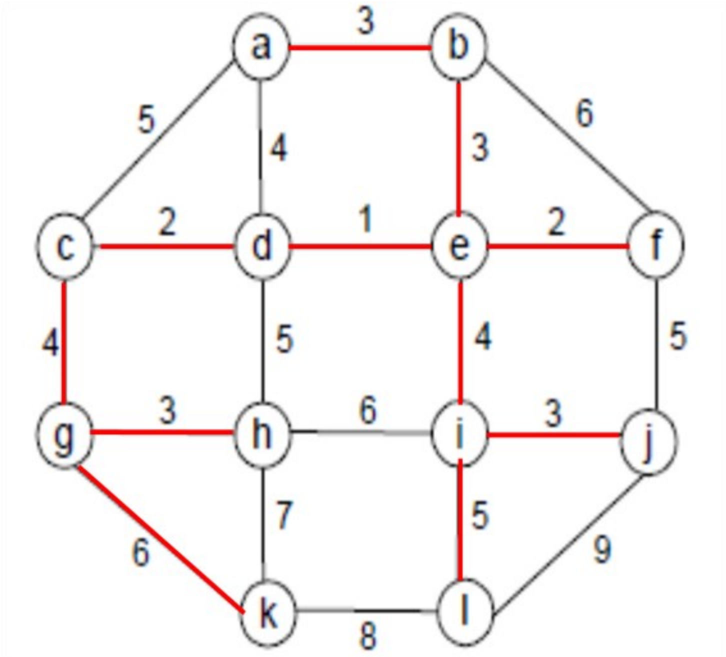


An Example



Tree vertices	Priority queue for the fringe vertices
a(-,-)	b(a,3), d(a,4), c(a,5), e(-,∞), f(-,∞), g(-,∞), h(-,∞), i(-,∞), j(-,∞), k(-,∞), l(-,∞)
b(a,3)	e(b,3), d(a,4), c(a,5), f(b,6), g(-,∞), h(-,∞), i(-,∞), j(-,∞), k(-,∞), l(-,∞)
e(b,3)	d(e,1) , f(e,2), i(e,4), c(a,5), g(-,∞), h(-,∞),j(-,∞), k(-,∞), l(-,∞)
d(e,1)	c(d,2), f(e,2), i(e,4), h(d,5), g(-,∞), j(-,∞), k(-,∞), l(-,∞)
c(d,2)	f(e,2), g(c,4), i(e,4), h(d,5), j(-,∞), k(-,∞), l(-,∞)
f(e,2)	g(c,4), i(e,4), h(d,5), j(f,5), k(-,∞), l(-,∞)
g(c,4)	h(g,3), i(e,4), j(f,5) , k(g,6), l(-,∞)
h(g,3)	i(e,4), j(f,5), k(g,6), l(-,∞)
i(e,4)	j(i,3), l(i,5), k(g,6)
j(i,3)	l(i,5), k(g,6)
l(i,5)	k(g,6)
k(g,6)	

An Example

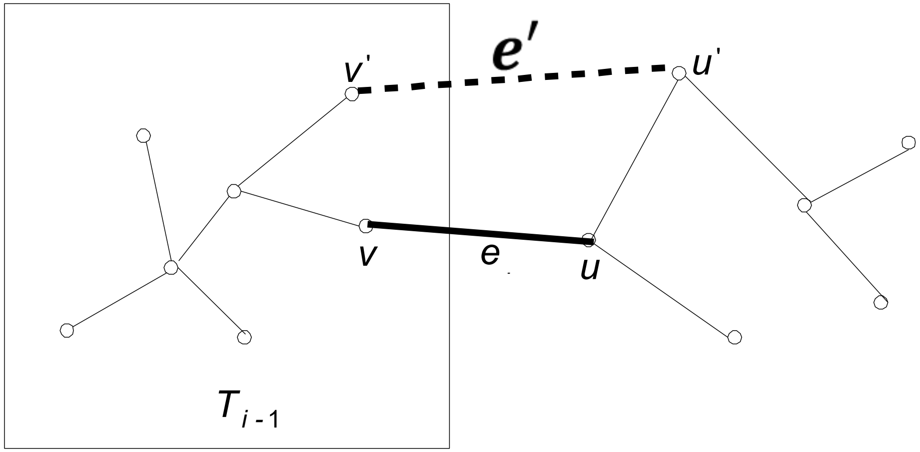


The MST consists of the edges ab, be, ed, dc, ef, cg, gh, ei, ij, il, and gk

Does Prim's Algorithm Really Produce MST?

Lemma: Let $\boldsymbol{T_{i-1}}$ be part of the minimum spanning tree T, which contains a subset of the vertices of $G(X)$. Let edge e be the smallest-weight edge connecting X (tree $\boldsymbol{T_{i-1}}$) to $G - X$ (remaining vertices). Then e (minimum-weight fringe edge) is part of the MST

Proof: Using contradiction, suppose $e = (u, v)$ is not part of MST. Then there is another edge $e' = (u', v')$ between X and $G - X$ and belongs to MST. Replace e' by e will result in a spanning tree with smaller total weight than MST. Contradiction!



Notes on Prim's algorithm

To locate the minimum-weight fringe edge, we can use the heap structure. But here we use the min-heap, where the root has a key smaller than both children

- Construct the min-heap -- $O(|V|)$
- Delete the min – $O(\log |V|)$, it can be performed $|V|$-1 times

- Verify minimum weight from any remaining vertex to the tree – this may be performed $|E|$ times. Each verification may result in a key priority change in the heap, which takes $O(\log |V|)$.

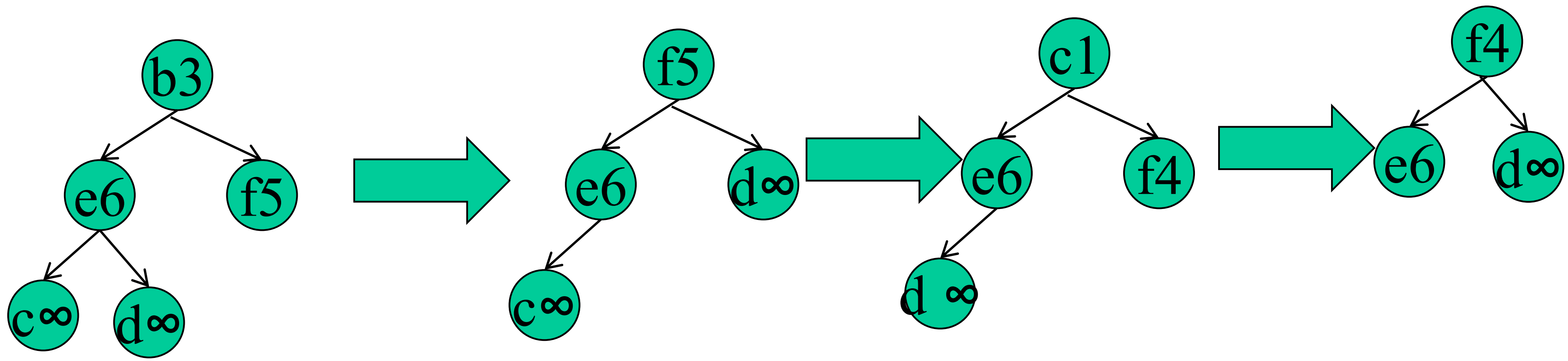
- Therefore, the total complexity is

$O[|V|+(|V|-1+|E|)*\log |V|]=O(|E| \log |V|)$

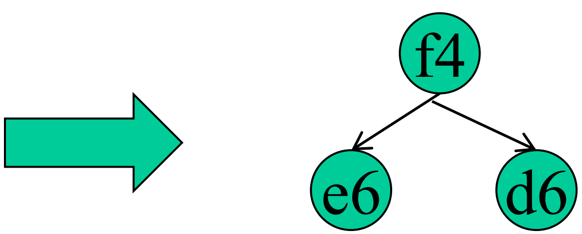
MinHeap and Prim's Algorithm

b(a,3), f(a,5), e(a,6), c(-,∞), d(-,∞)

c(b,1), f(b,4), e(a,6), d(-,∞)



f(b,4), d(c,6), e(a,6)



Dijkstra's Algorithm – Single-Source Shortest Paths

Single-source short paths problem – find the shortest path starting from a given vertex to any other vertex

Example: hub airports for airplane planning

Using greedy strategy to find the single-source shortest paths

- In Floyd's algorithm, we find the all-pair shortest paths, which may not be necessary in many applications
- Certainly, all-pair shortest paths contain the single-source shortest paths. But Floyd's algorithm has $O(|V|^3)$ complexity!

There are many algorithms that can solve this problem, here we introduce the **Dijkstra's algorithm**

Note: Dijkstra's algorithm only works **when all the edge-weight are nonnegative.**

Dijkstra's Algorithm on Undirected Graph

Similar to Prim's MST algorithm, with the following difference:

- Start with tree consisting of one vertex – **source**
- "grow" tree one vertex/edge, which has minimum length of path, at a time to produce spanning tree
  - Construct a series of expanding subtrees T_1, T_2, …
- Keep track of shortest path from source to each of the vertices in T_i
- at each stage construct T_{i+1} from T_i: add ~~minimum weight edge~~ connecting a vertex in tree (T_i) to one not yet in tree
  - choose from "fringe" nodes
  - (this is the "greedy" step!)

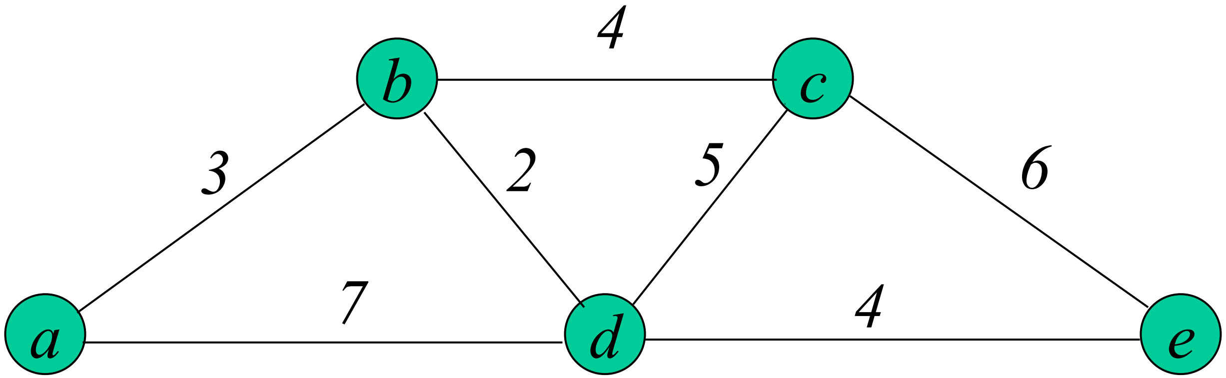
edge (v,w) with lowest d(s,v) + d(v,w)

source *destination*

- algorithm stops when all vertices are included

Example:

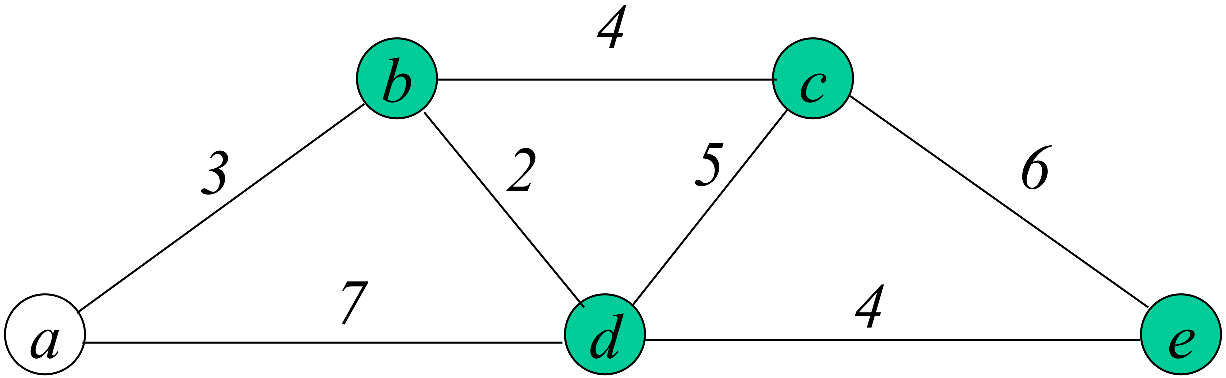
Find the shortest paths starting from vertex a



Step 1:

Tree vertices: a(-,0)

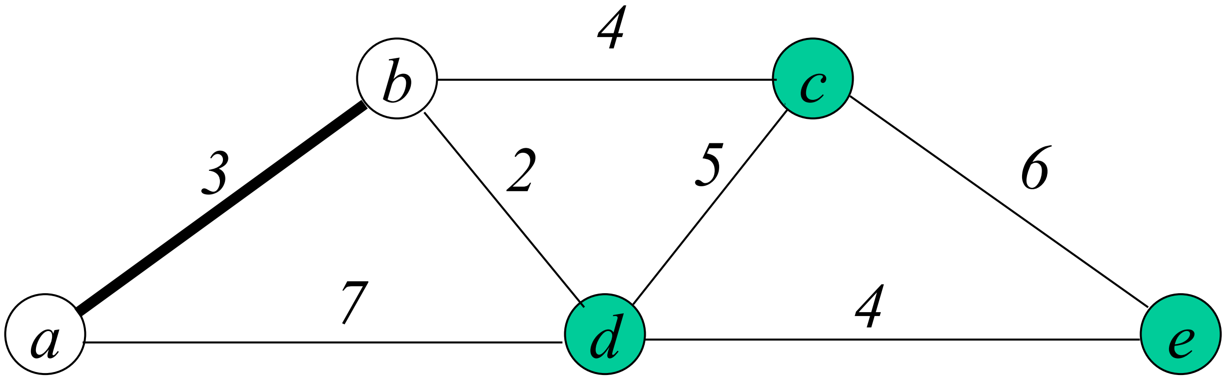
Priority queue: ***b(a,3)**, d(a,7), **c(-,∞)**, e (-,∞)*



Step 2:

***Tree vertices:* a(-,0), b(a,3),**

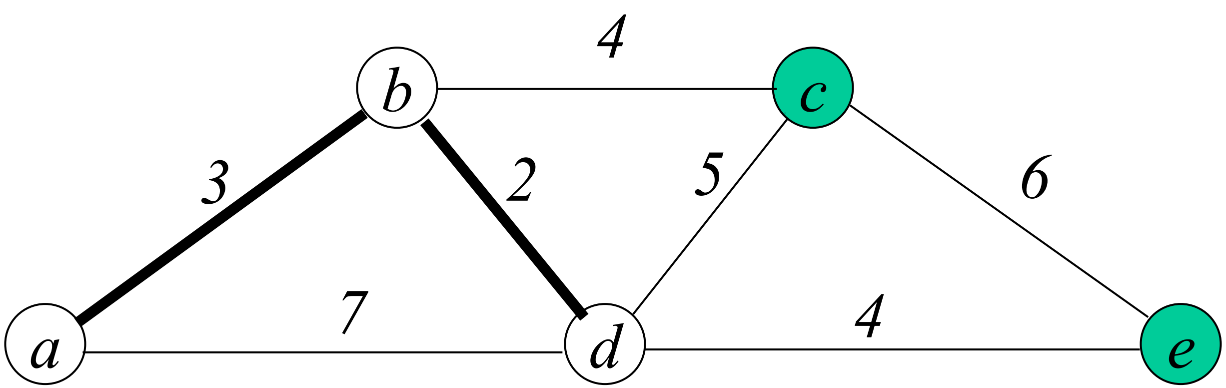
Priority queue: $d(a,7) \rightarrow d(b,3+2)$, $c(-,\infty) \rightarrow c(b,3+4)$, $e\ (-,\infty)$



Step 3:

Tree vertices: ***a(-,0), b(a,3),*** ***d(b,5)***

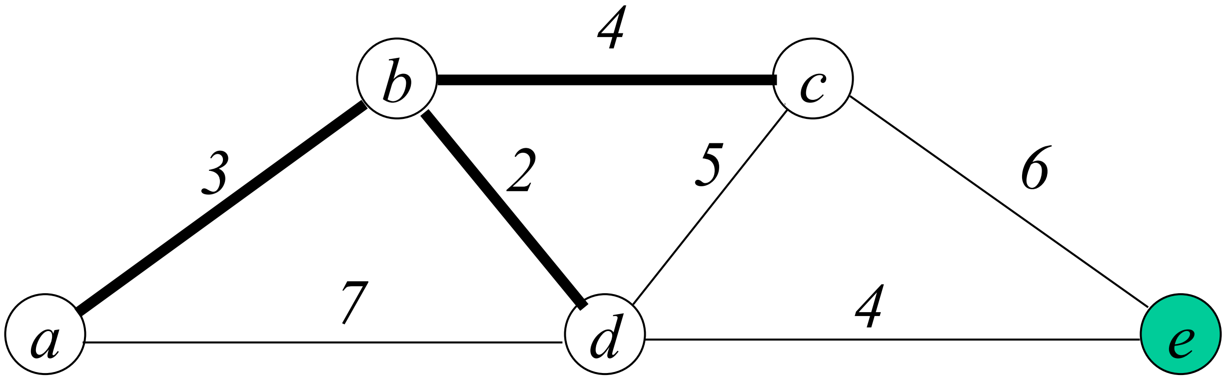
Priority queue: ***c(b,3+4)***, ***e*** ***(-,∞)*** →***e(d,5+4)***



Step 4:

Tree vertices: a(-,0), b(a,3), d(b,5), c(b,7)

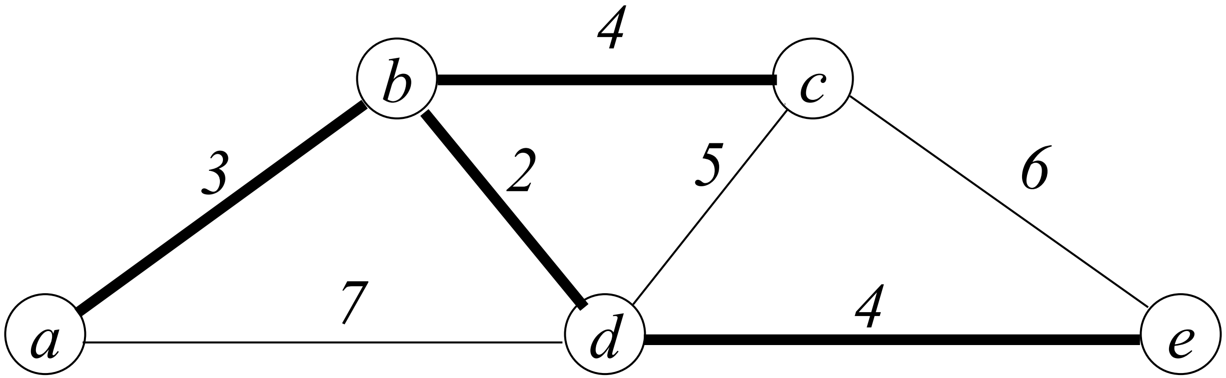
Priority queue: *e(d,9)*



Step 5:

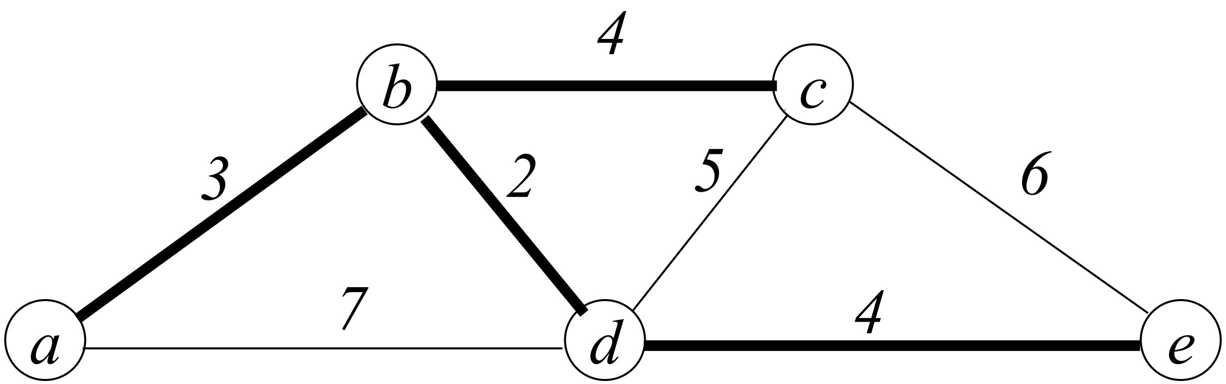
Tree vertices: a(-,0), b(a,3), d(b,5), c(b,7), e(d,9)

Remaining vertices: none → the algorithm is done!



Output the Single-Source Shortest Paths

Tree vertices:* *a(-,0), b(a,3), d(b,5), c(b,7), e(d,9)



from *a* to *b*:	**a-b**	**of length 3**
from *a* to *d*:	**a-b-d**	**of length 5**
from *a* to *c*:	**a-b-c**	**of length 7**
from *a* to *e*:	**a-b-d-e**	**of length 9**