Algorithmics

Lecture 7

Dr. Oana Andrei

School of Computing Science
University of Glasgow

oana.andrei@glasgow.ac.uk

Section 3 – Graphs and graph algorithms

Graph basics

– definitions: directed, undirected, connected, bipartite, …

Graph representations

– adjacency matrix/lists and implementation

Graph search and traversal algorithms

– breadth/depth first search

Topological ordering

Weighted graphs

– shortest path (Dijkstra’s algorithm)
– minimum spanning tree (Prim–Jarnik and Dijkstra’s refinement)

Graphs – Recap

An (undirected) graph `G = (V,E)`

- V is finite set of vertices (the vertex set)
- E is set of edges, each edge is a subset of V of size 2 (the edge set)
- each edge is a set vertices of the form {u,v}

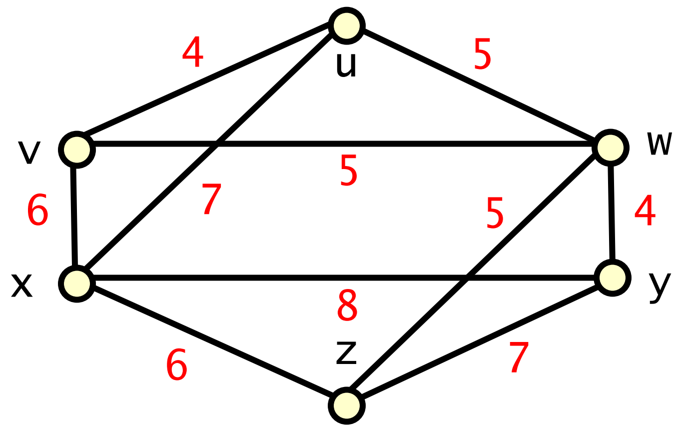
A directed graph (digraph) `G = (V,E)`

- V is the finite set of vertices and E is the finite set of edges
- here each edge is an ordered pair (x,y) of vertices

Weighted graphs – Recap

Each edge e has a positive integer weight given by `wt(e)>0`

- graph may be undirected or directed
- weight may represent length, cost, capacity, etc
- if an edge is not part of the graph its weight is infinity



Spanning trees

Spanning tree:

- subgraph (subset of edges) which is both a tree and 'spans' every vertex
- a spanning tree is obtained from a connected graph by deleting edges
- the weight of a spanning tree is the sum of the weights of its edges

Problem: for a weighted connected undirected graph, find a minimum weight spanning tree

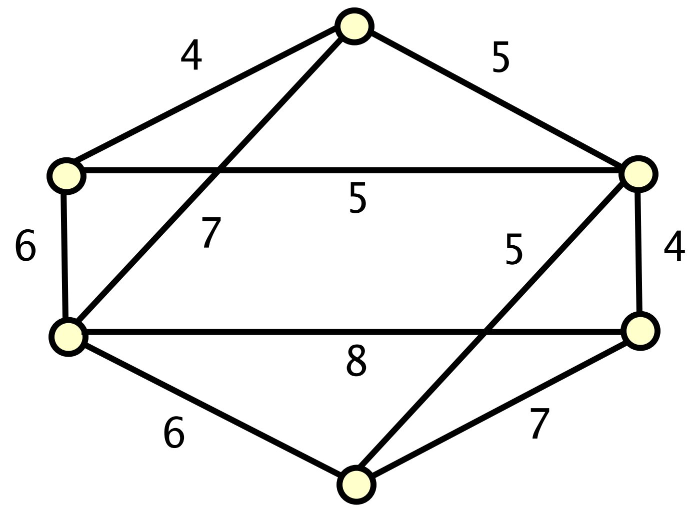
- this represents the 'cheapest' way of interconnecting the vertices

Applications include:

- design of networks for computer, telecommunications, transportation, gas, electricity, ...
- clustering, approximating the travelling salesman problem

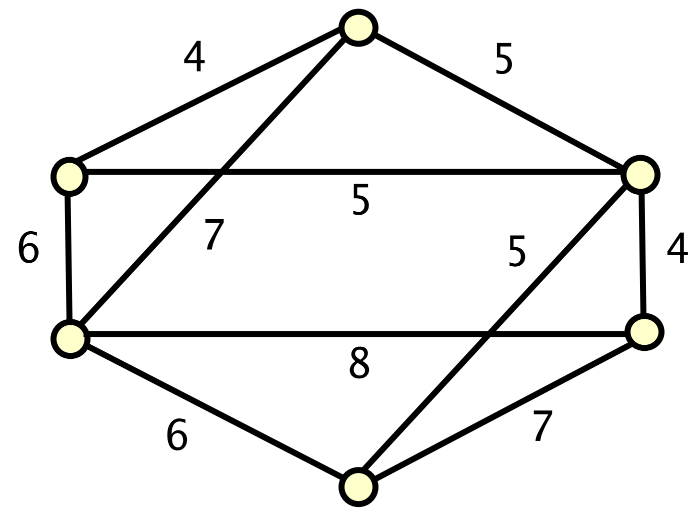
Weighted graphs – Example – Spanning tree

Weighted graph G



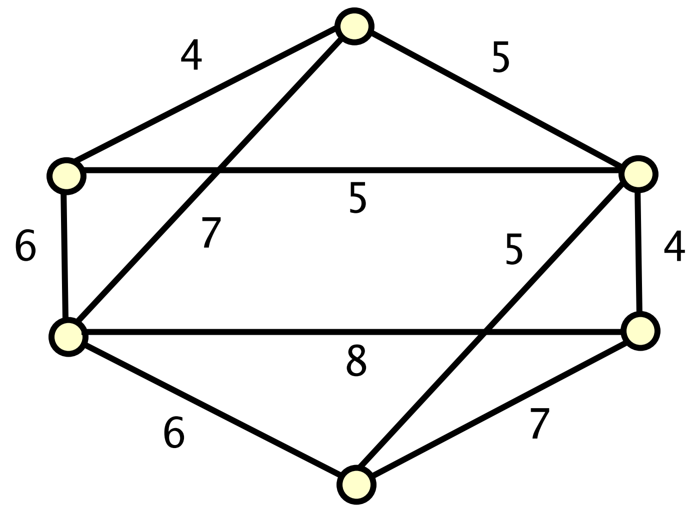
spanning tree:
subgraph which is
both a tree and
'spans' every vertex

delete edges while still
'spanning' vertices



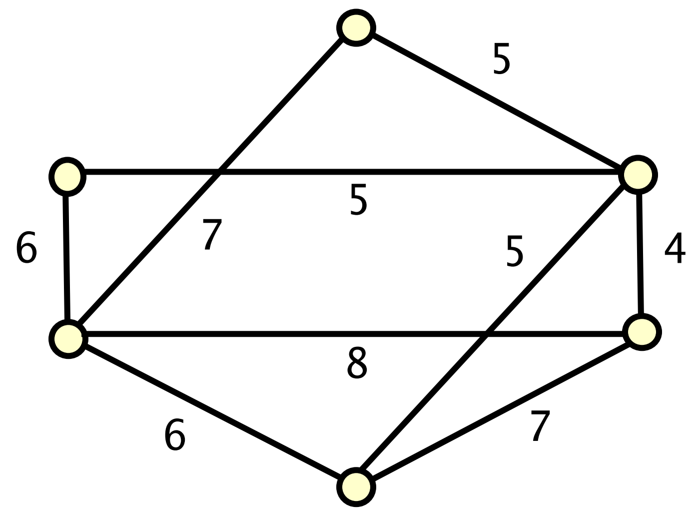
Weighted graphs – Example – Spanning tree

Weighted graph G



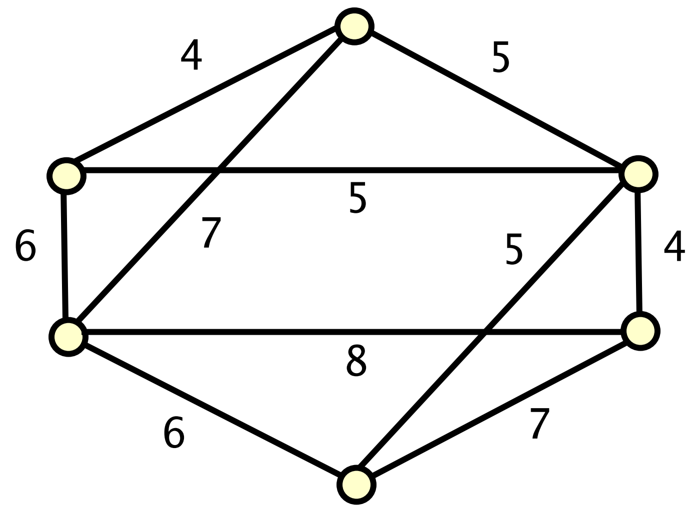
spanning tree:
subgraph which is
both a tree and
'spans' every vertex

delete edges while still
'spanning' vertices



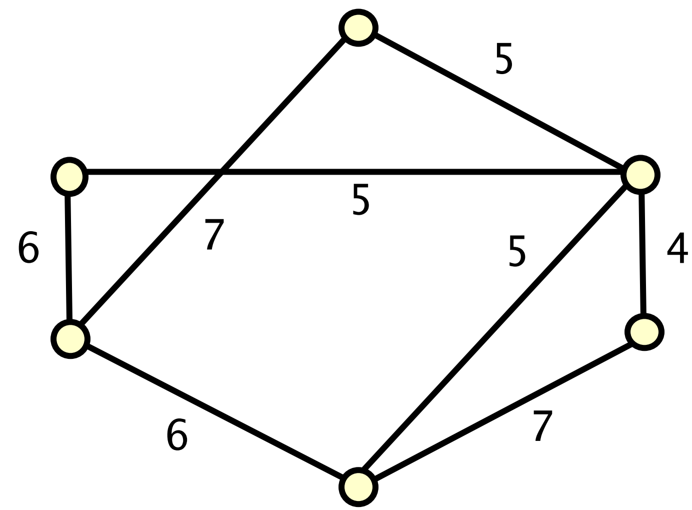
Weighted graphs – Example – Spanning tree

Weighted graph G



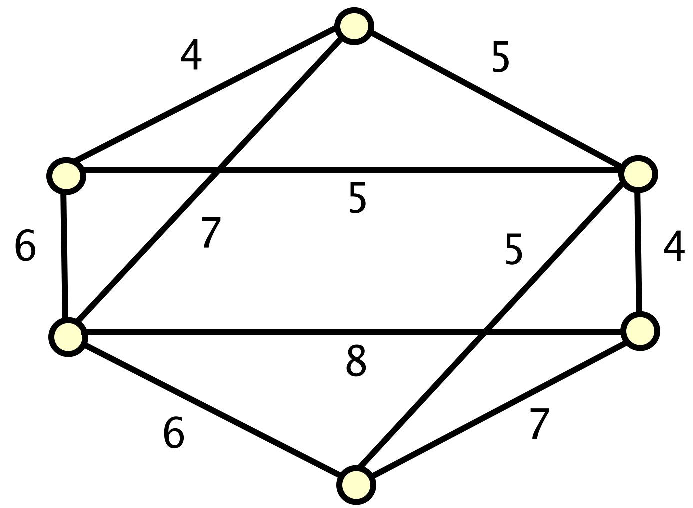
spanning tree: subgraph which is both a tree and 'spans' every vertex

delete edges while still 'spanning' vertices



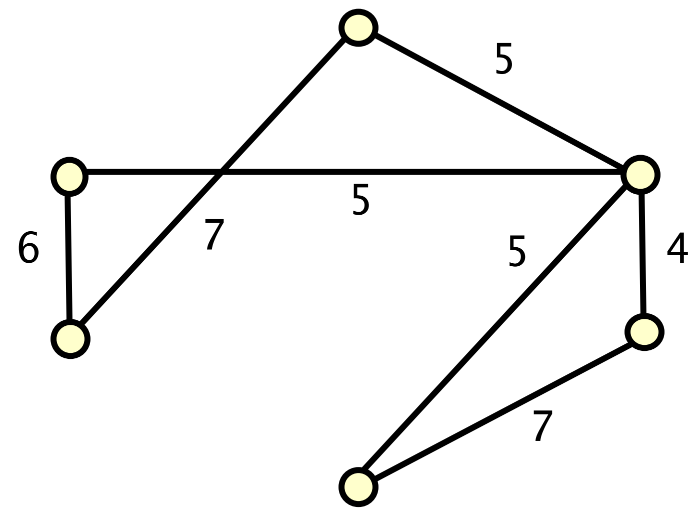
Weighted graphs – Example – Spanning tree

Weighted graph G



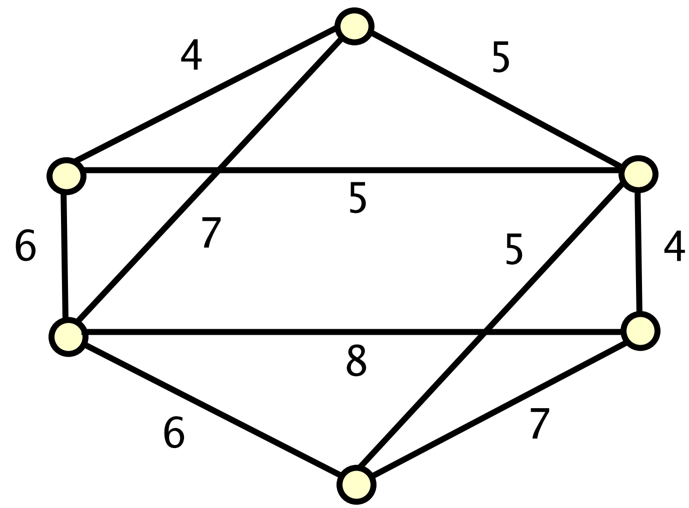
spanning tree:
subgraph which is
both a tree and
'spans' every vertex

delete edges while still
'spanning' vertices



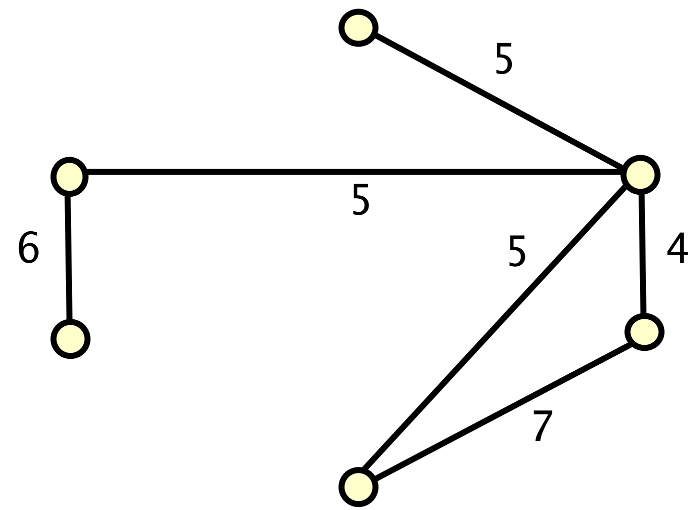
Weighted graphs – Example – Spanning tree

Weighted graph G



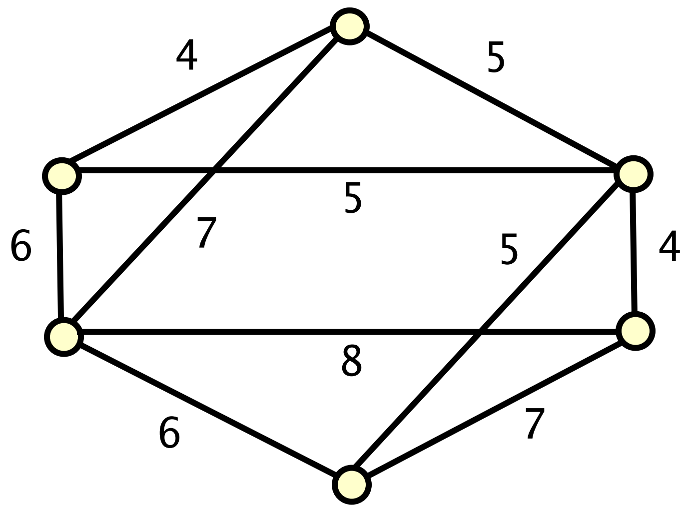
spanning tree:
subgraph which is
both a tree and
'spans' every vertex

delete edges while still
'spanning' vertices



Weighted graphs – Example – Spanning tree

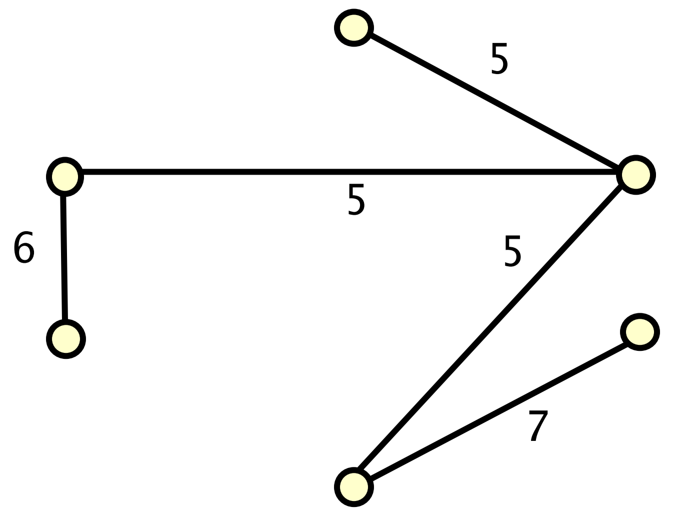
Weighted graph G



spanning tree: subgraph which is both a tree and 'spans' every vertex

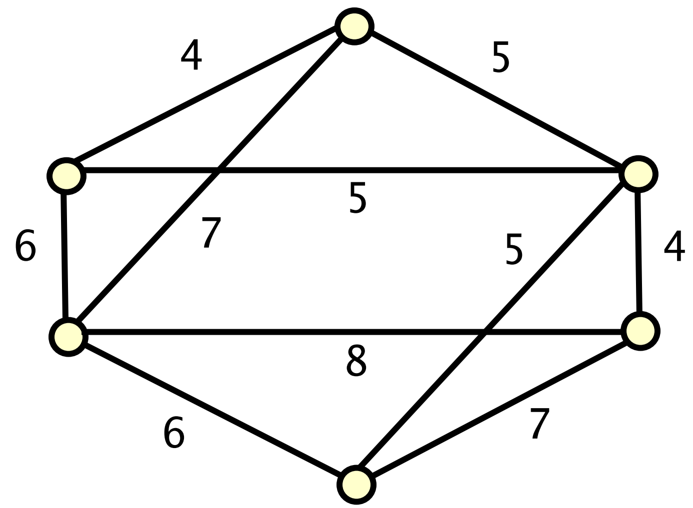
delete edges while still 'spanning' vertices

cannot delete any more edges and we have a tree



Weighted graphs – Example – Spanning tree

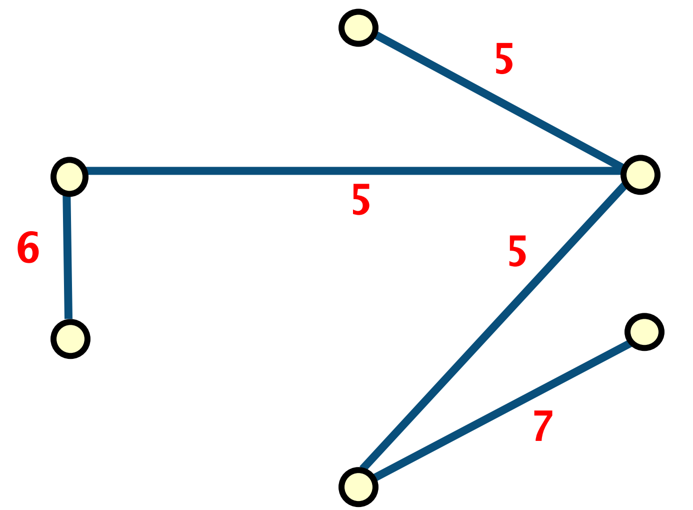
Weighted graph G



spanning tree:
subgraph which is
both a tree and
'spans' every vertex

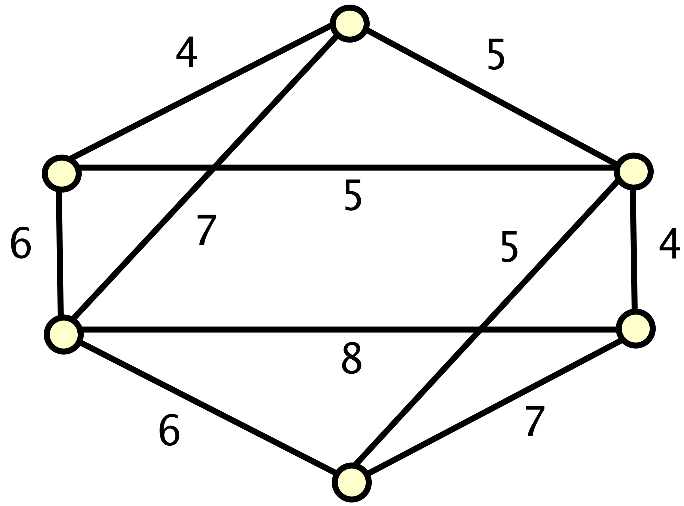
Spanning tree for G

– weight 28



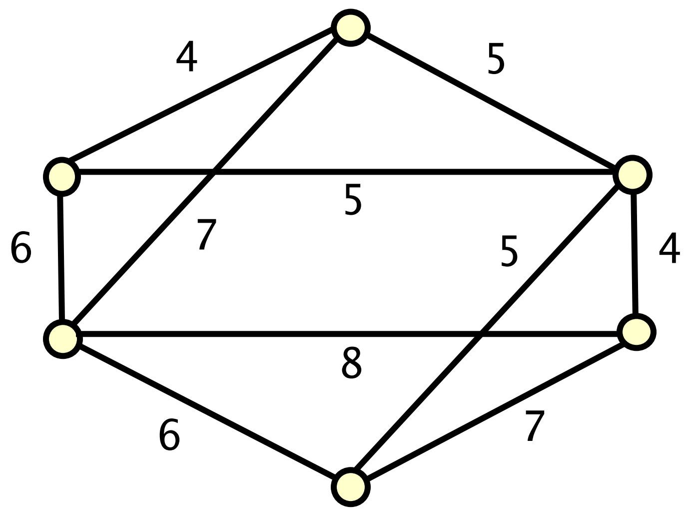
Weighted graphs – Example – Spanning tree

Weighted graph G



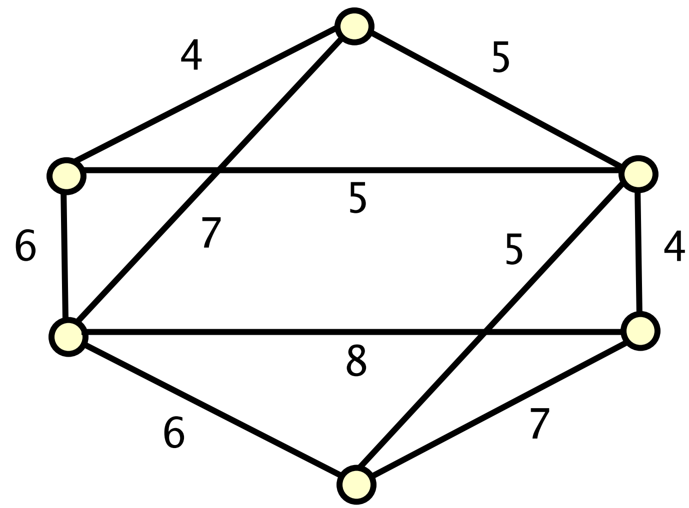
spanning tree:
subgraph which is
both a tree and
'spans' every vertex

delete edges while still
'spanning' vertices



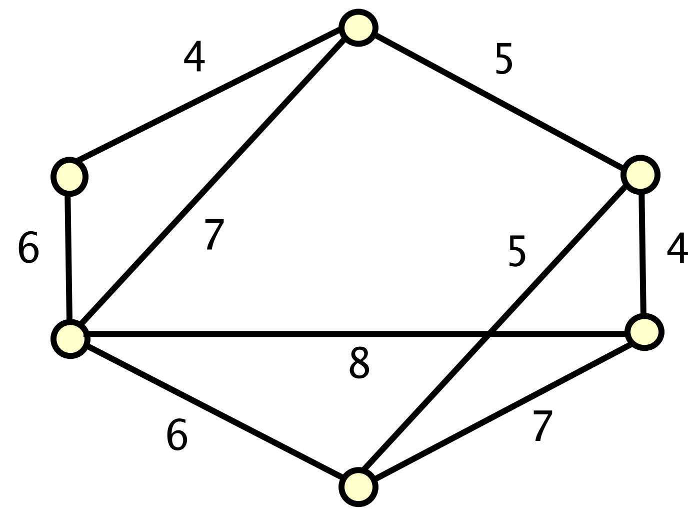
Weighted graphs – Example – Spanning tree

Weighted graph G



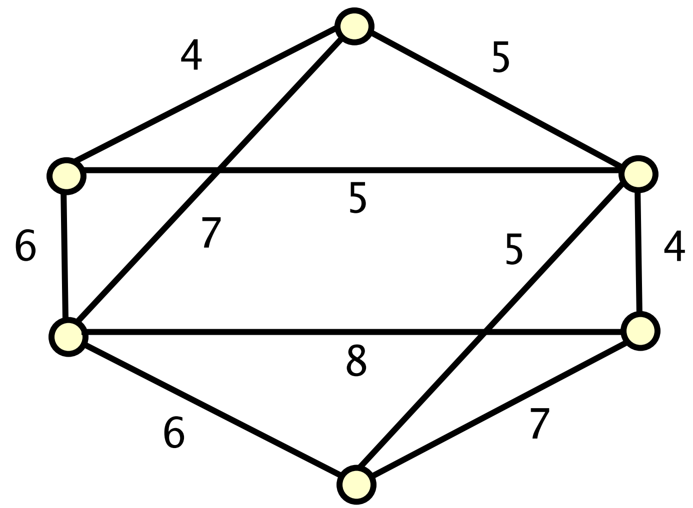
spanning tree:
subgraph which is
both a tree and
'spans' every vertex

delete edges while still
'spanning' vertices



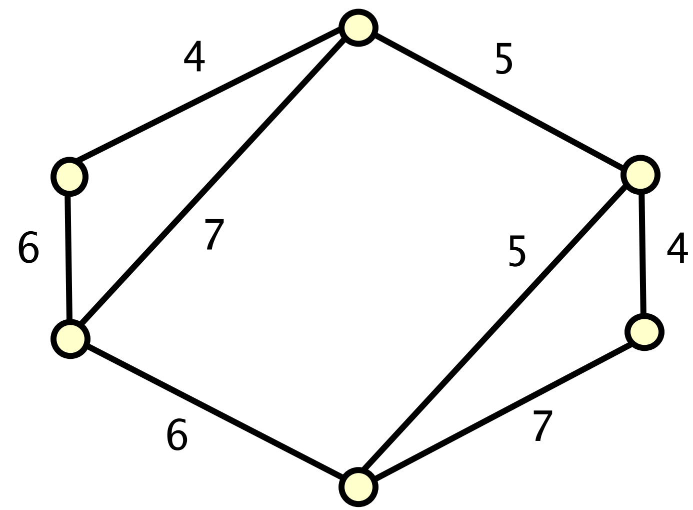
Weighted graphs – Example – Spanning tree

Weighted graph G



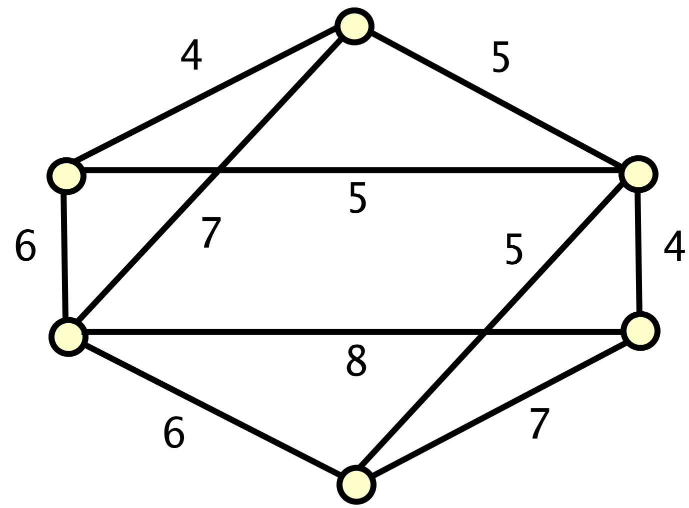
spanning tree:
subgraph which is
both a tree and
'spans' every vertex

delete edges while still
'spanning' vertices



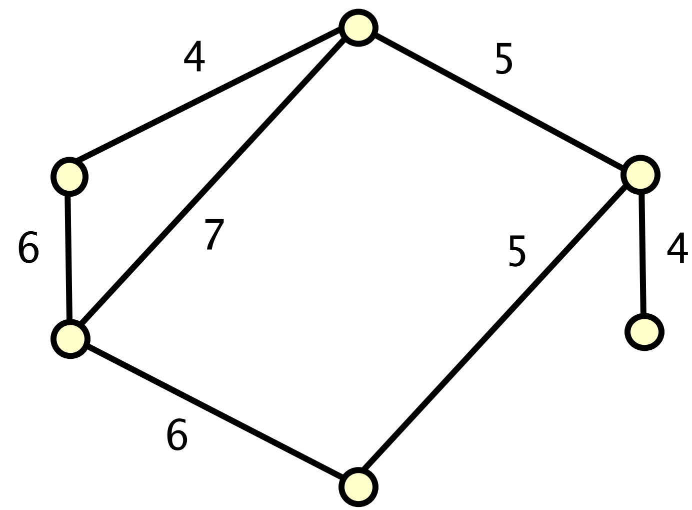
Weighted graphs – Example – Spanning tree

Weighted graph G



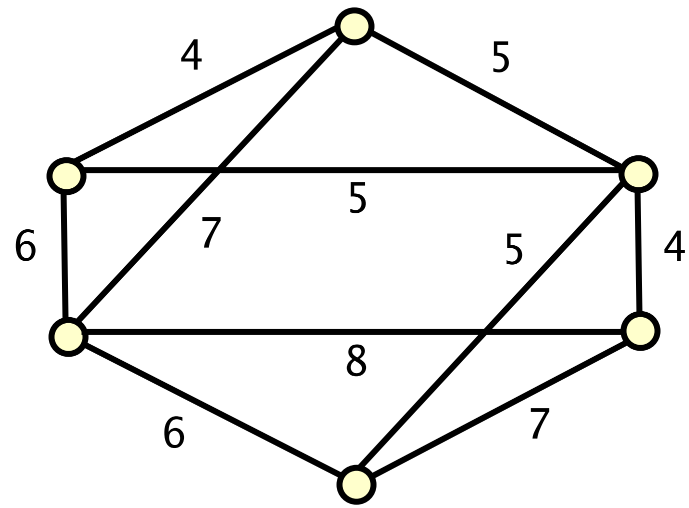
spanning tree:
subgraph which is
both a tree and
'spans' every vertex

delete edges while still
'spanning' vertices



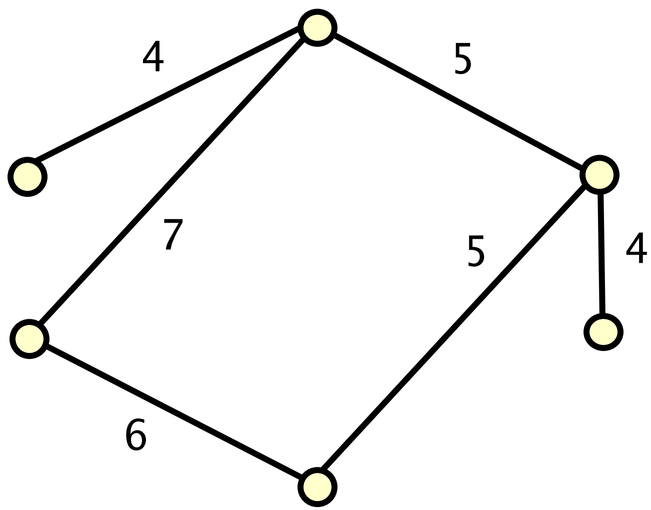
Weighted graphs – Example – Spanning tree

Weighted graph G



spanning tree:
subgraph which is
both a tree and
'spans' every vertex

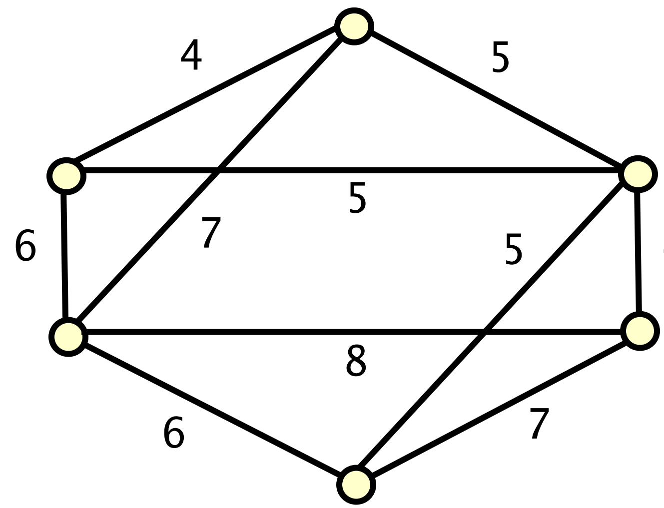
delete edges while still
'spanning' vertices



Weighted graphs – Example – Spanning tree

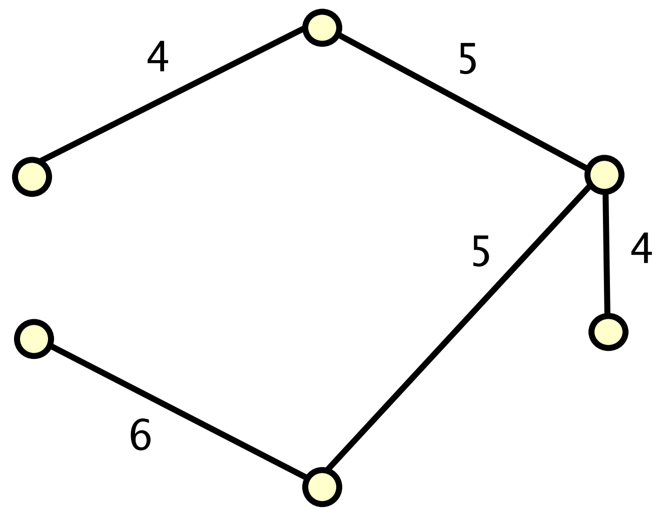
Weighted graph G

spanning tree: subgraph which is both a tree and 'spans' every vertex



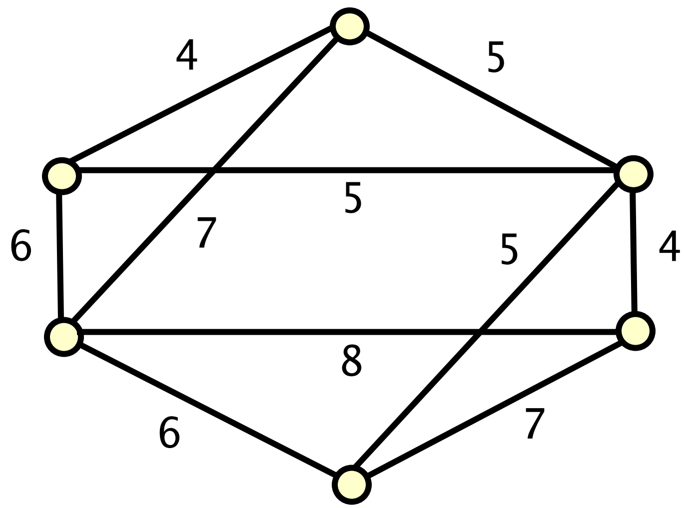
delete edges while still 'spanning' vertices

cannot delete any more edges and we have a tree



Weighted graphs – Example – Spanning tree

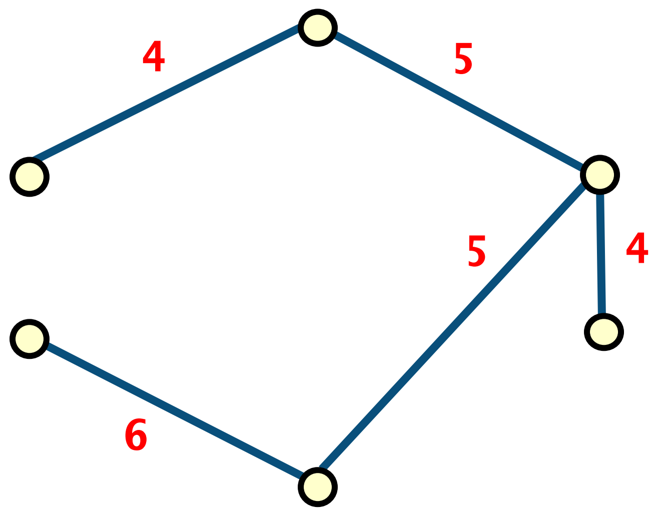
Weighted graph G



spanning tree:
subgraph which is
both a tree and
'spans' every vertex

Spanning tree for G

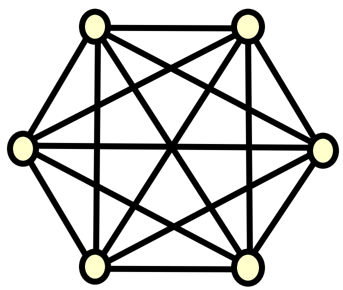
- weight 24



Minimum weight spanning tree problem

An example of a problem in combinatorial optimisation

- find ‘best’ way of doing something among a (large) number of candidates
- can always be solved, at least in theory, by exhaustive search (how?)
- however this may be infeasible in practice
- typically an exponential-time algorithm
  - e.g. K_n (clique of size n) has n^{n-2} spanning trees (Cayley’s formula)
  - recall: a graph is a clique if every pair vertices is joined by an edge



- a much more efficient algorithm may be possible

  and is true in the case of minimum weight spanning trees

Minimum weight spanning tree problem

An example of a problem in combinatorial optimisation

- find ‘best’ way of doing something among a (large) number of candidates
- can always be solved, at least in theory, by exhaustive search
- however this may be infeasible
- typically an exponential-time algorithm

The Prim-Jarnik minimum spanning tree algorithm

- an example of a greedy algorithm
- it makes a sequence of decisions based on local optimality
- and ends up with the globally optimal solution

For many problems, greedy algorithms do not yield optimal solution

- see examples later in the course

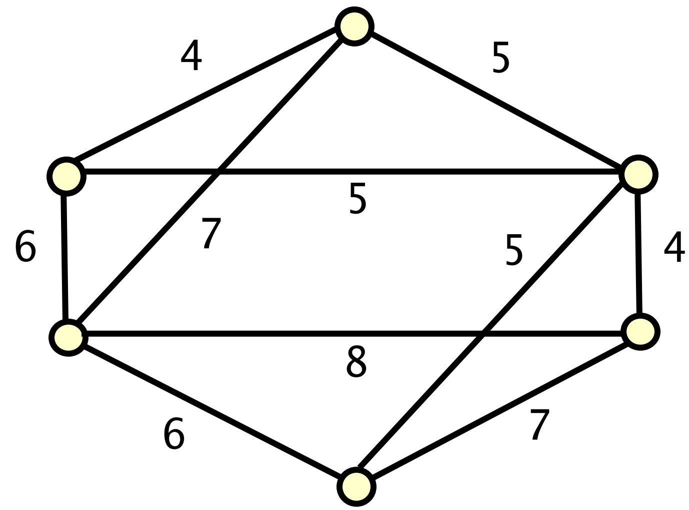
The Prim–Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edges

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of the graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

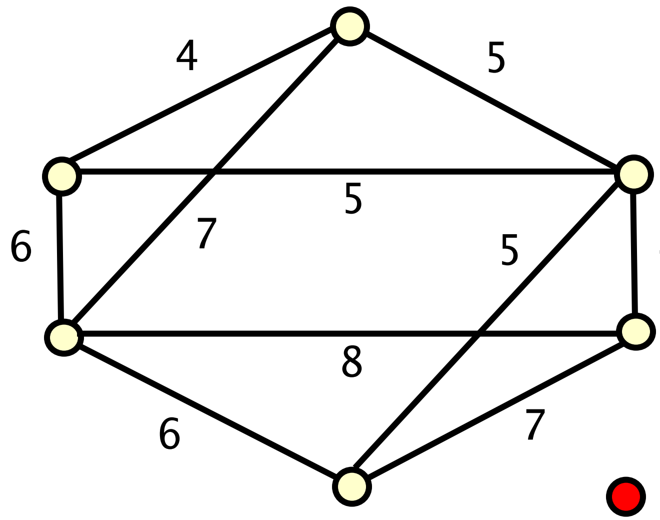
The Prim–Jarnik algorithm – Example

Weighted graph G



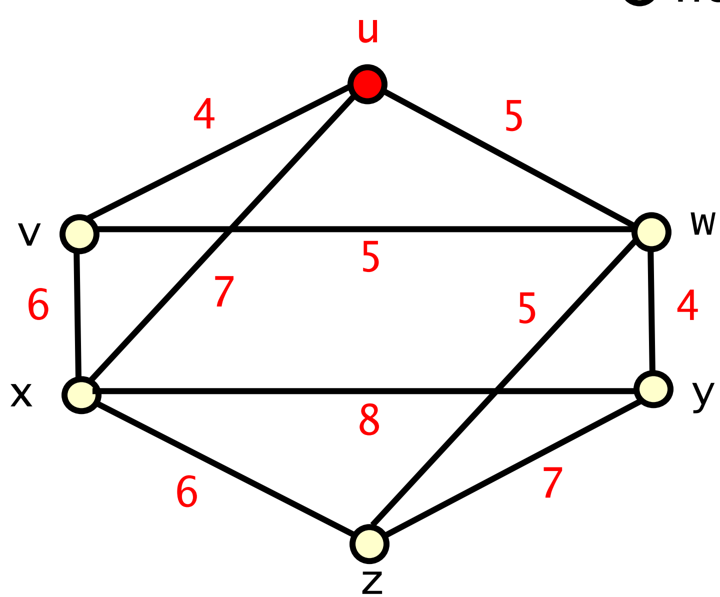
The Prim–Jarnik algorithm – Example

Weighted graph G



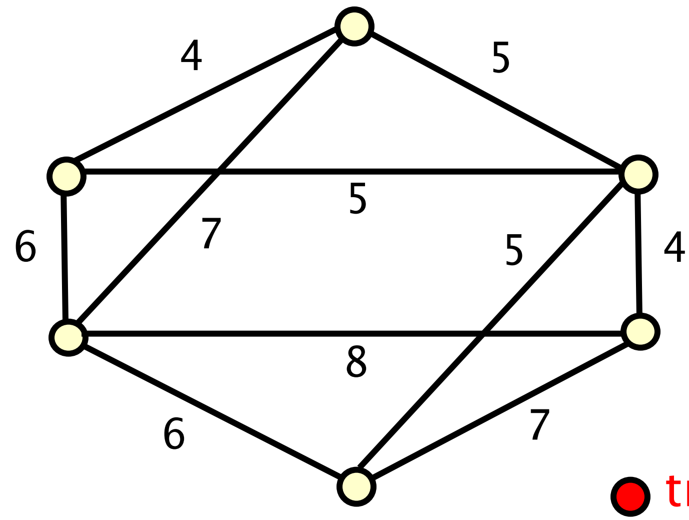
tree vertices: u

non-tree vertices: v,w,x,y,z



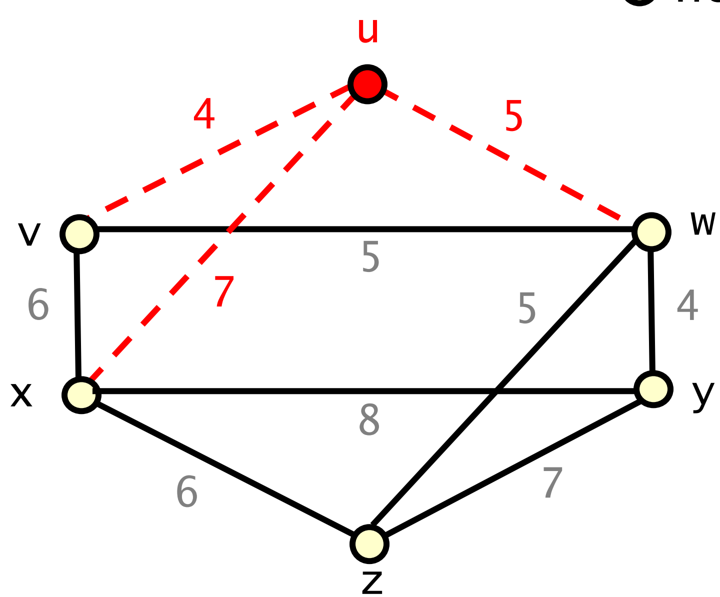
The Prim–Jarnik algorithm – Example

Weighted graph G



tree vertices: u

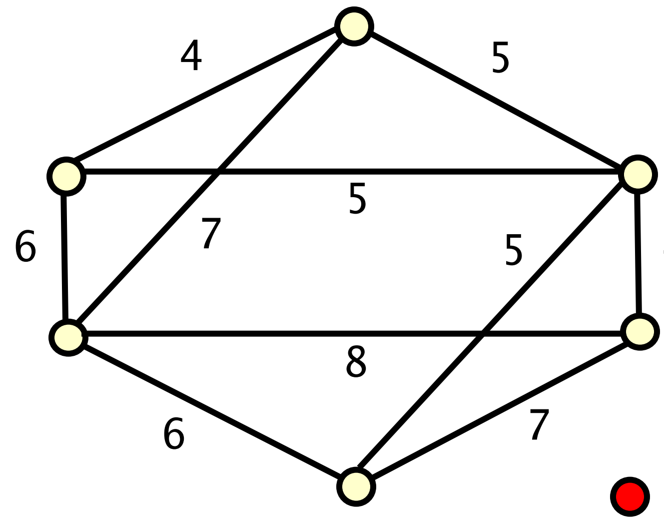
non-tree vertices: v,w,x,y,z



consider all tree vertex and non-tree vertex edges

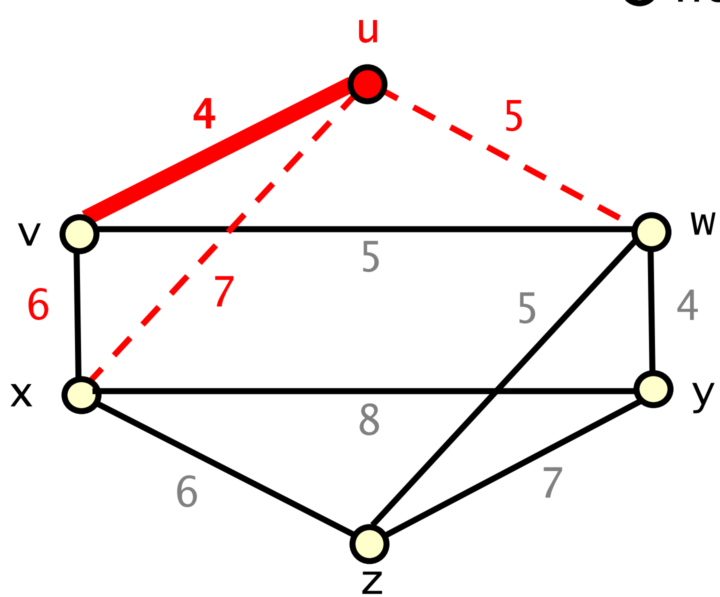
The Prim–Jarnik algorithm – Example

Weighted graph G



tree vertices: u

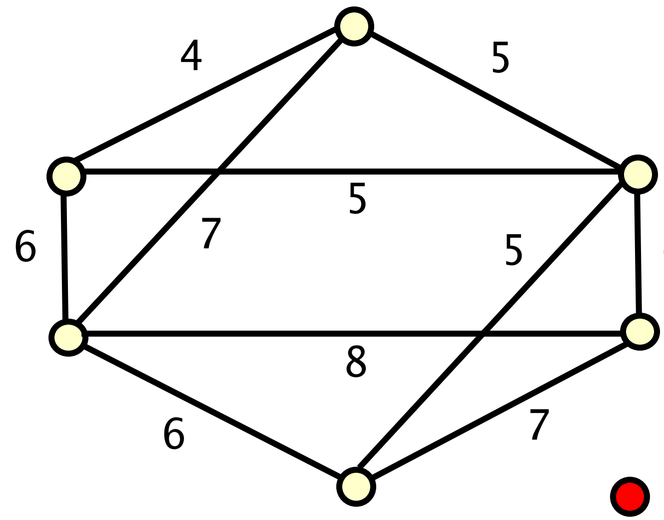
non-tree vertices: v,w,x,y,z



minimum such edge is {u,v}

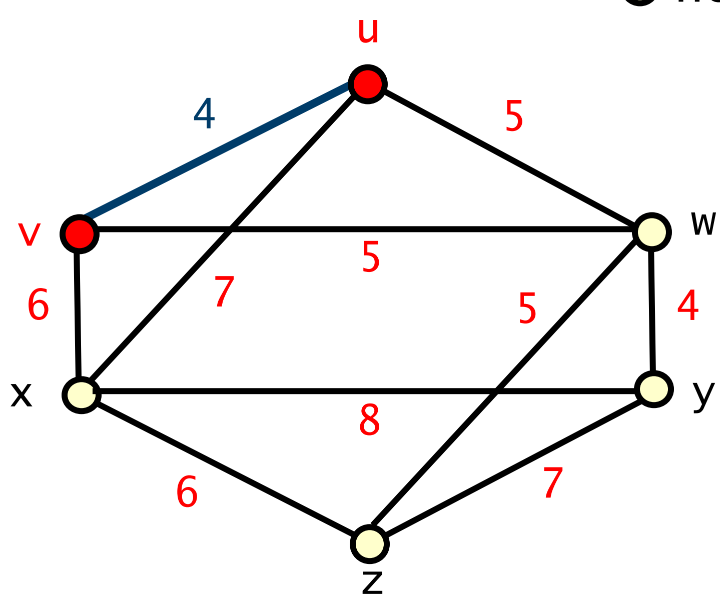
The Prim–Jarnik algorithm – Example

Weighted graph G



tree vertices: u,v

non-tree vertices: w,x,y,z

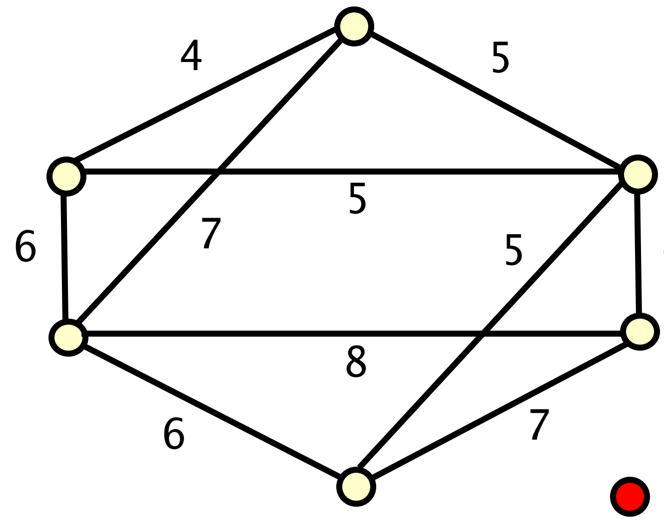


make v a tree vertex

& add edge to the tree

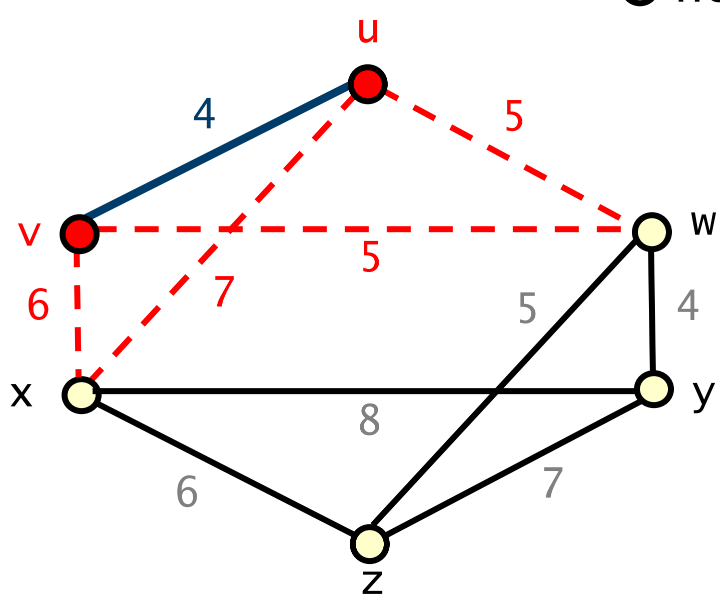
The Prim–Jarnik algorithm – Example

Weighted graph G



tree vertices: u,v

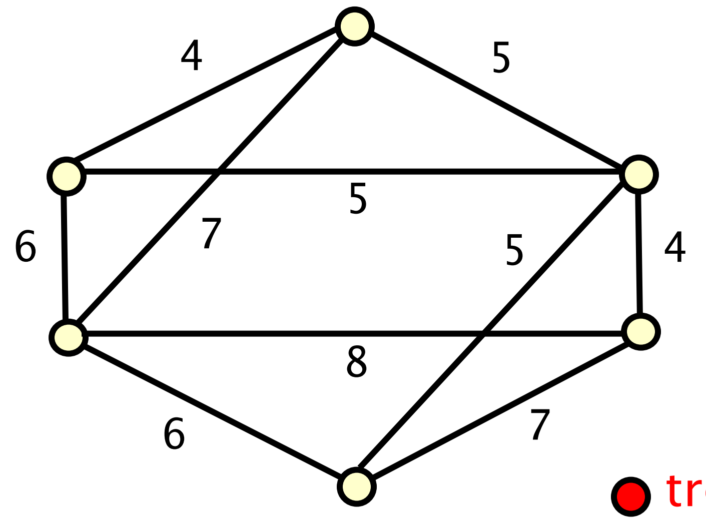
non-tree vertices: w,x,y,z



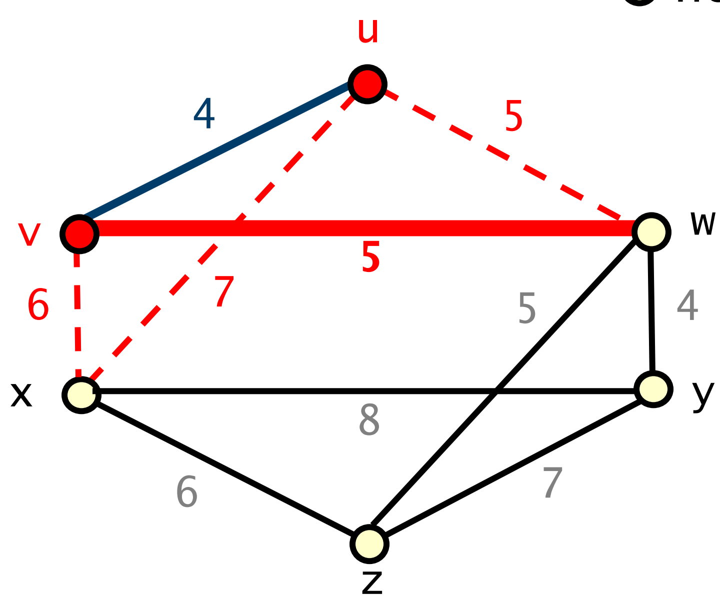
consider all
tree vertex
and non-tree
vertex edges

The Prim–Jarnik algorithm – Example

Weighted graph G



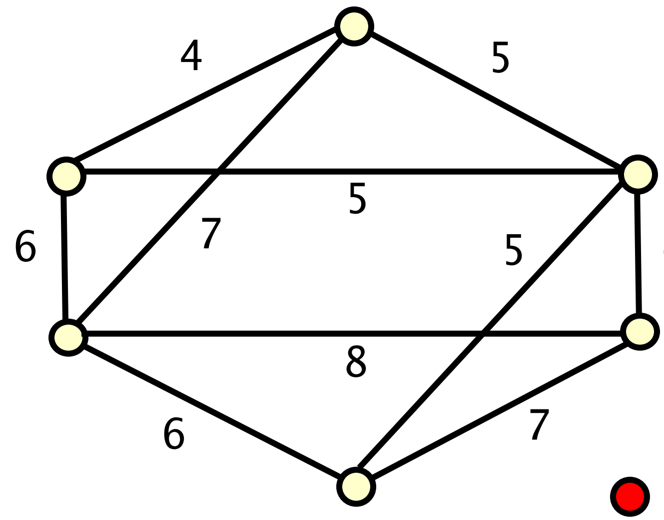
- tree vertices: u, v
- non-tree vertices: w, x, y, z



minimum such edge is {v,w}

The Prim–Jarnik algorithm – Example

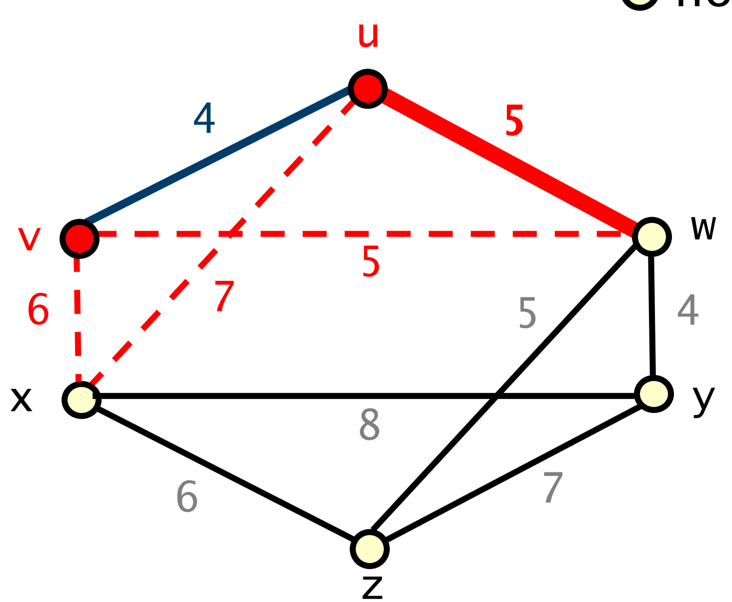
Weighted graph G



tree vertices: u,v

non-tree vertices: w,x,y,z

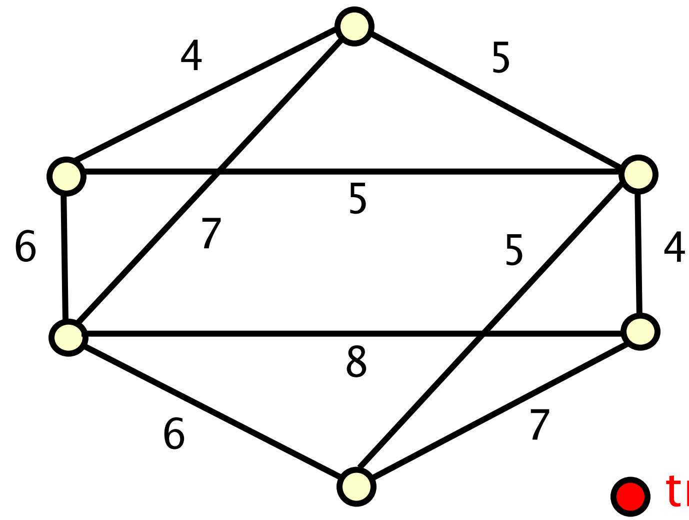
could also have chosen {u,w}



minimum such edge is {v,w}

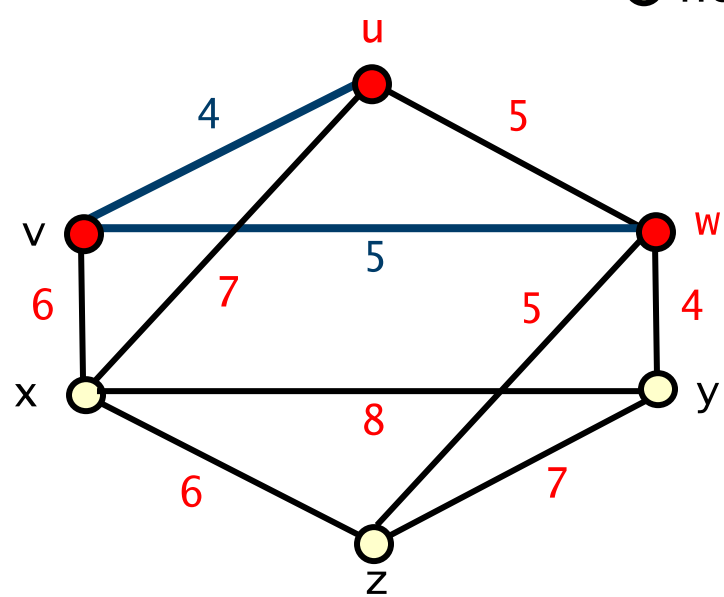
The Prim–Jarnik algorithm – Example

Weighted graph **G**



tree vertices: u, v, w

non-tree vertices: x, y, z

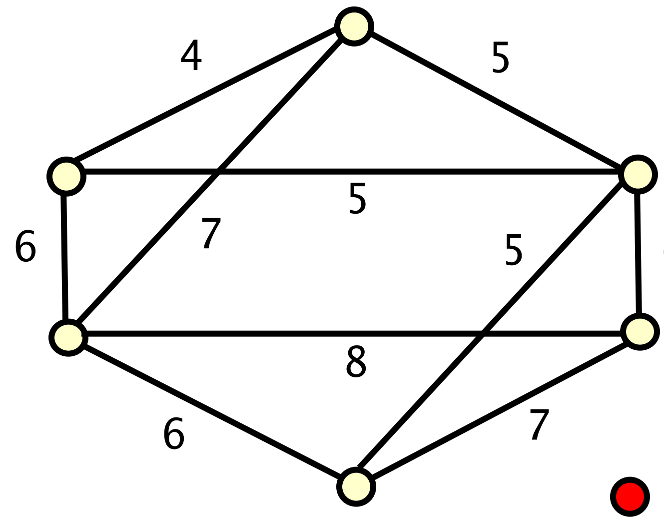


make w a tree vertex

& add edge to the tree

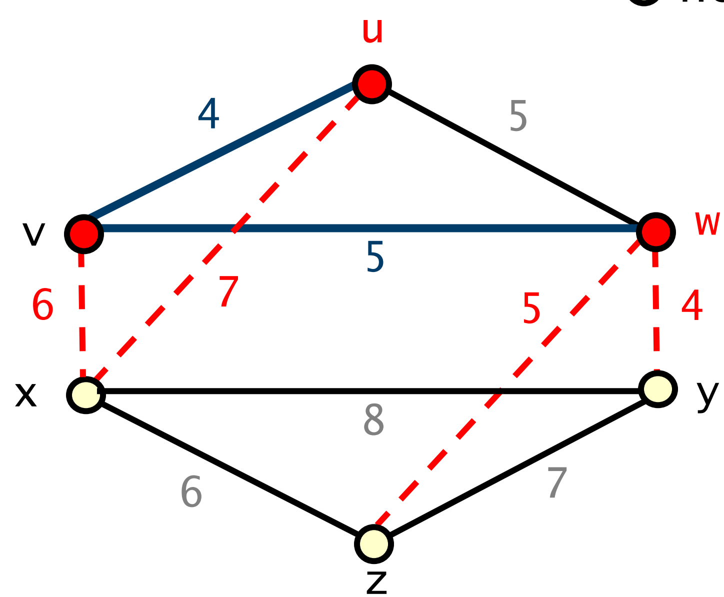
The Prim–Jarnik algorithm – Example

Weighted graph G



tree vertices: u,v,w

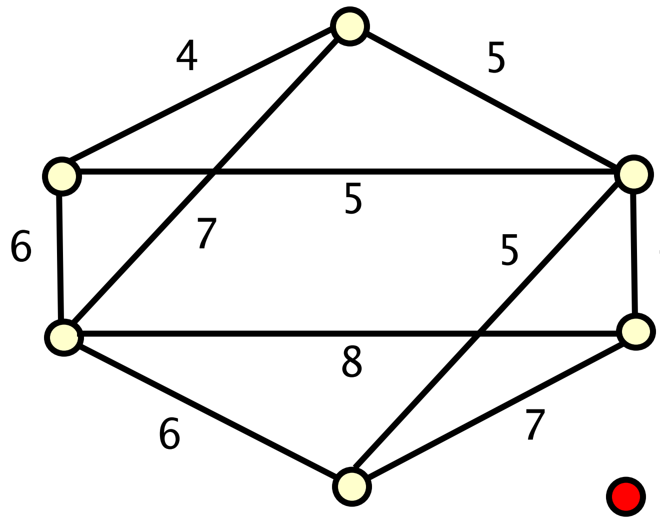
non-tree vertices: x,y,z



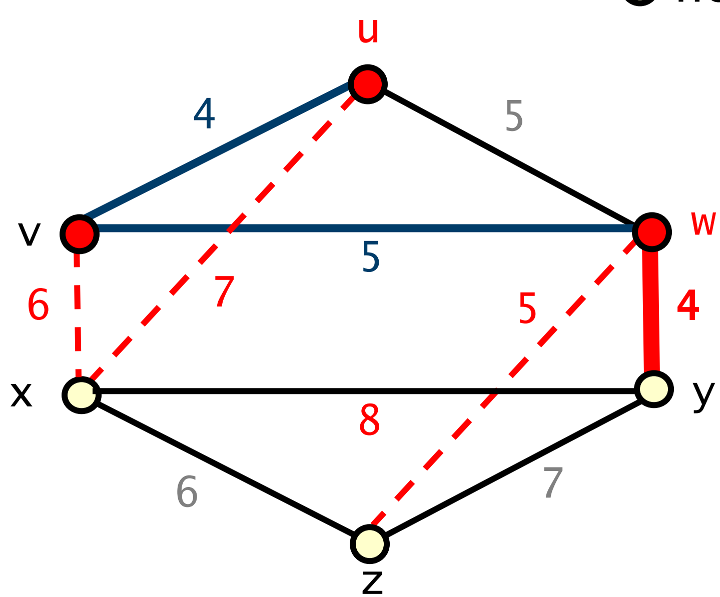
consider all tree vertex and non-tree vertex edges

The Prim–Jarnik algorithm – Example

Weighted graph **G**



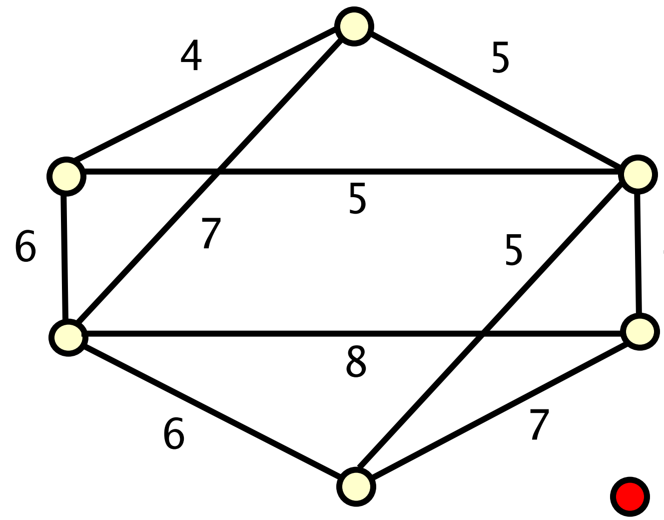
tree vertices: u,v,w
non-tree vertices: x,y,z



minimum such edge is {w,y}

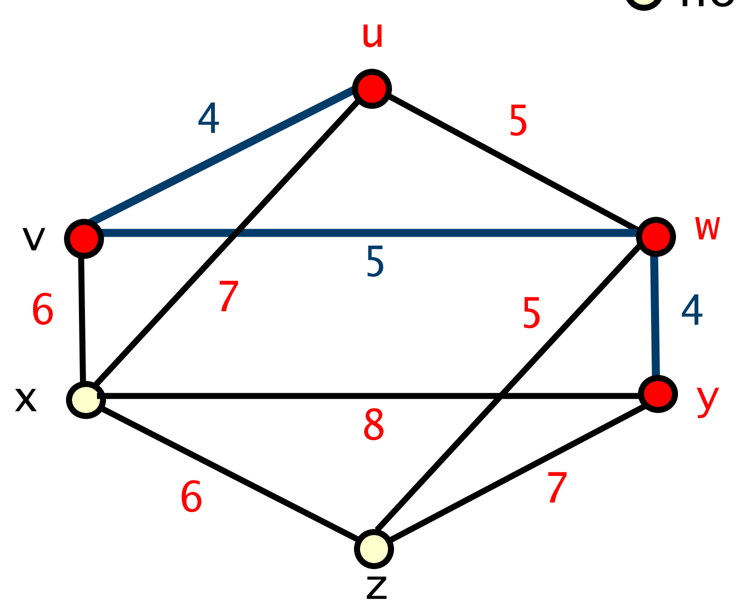
The Prim–Jarnik algorithm – Example

Weighted graph G



tree vertices: u, v, w, y

non-tree vertices: x, z

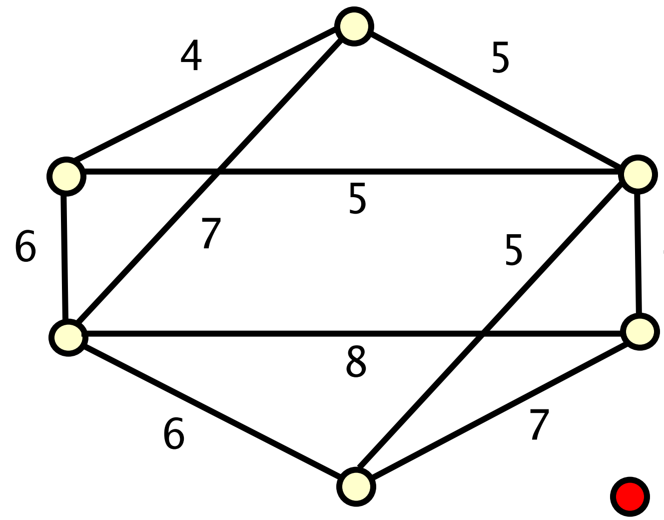


make y a tree vertex

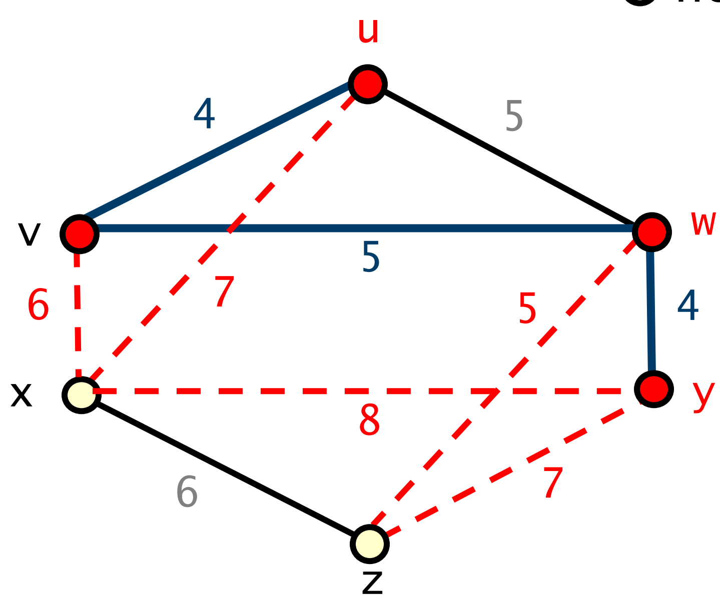
& add edge to the tree

The Prim–Jarnik algorithm – Example

Weighted graph G



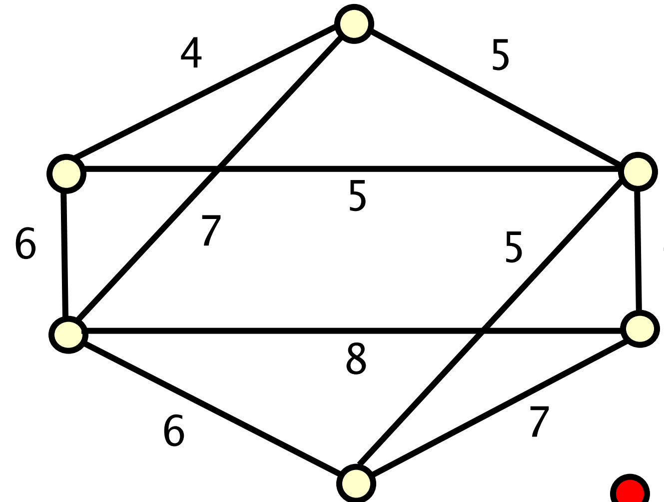
tree vertices: u,v,w,y
non-tree vertices: x,z



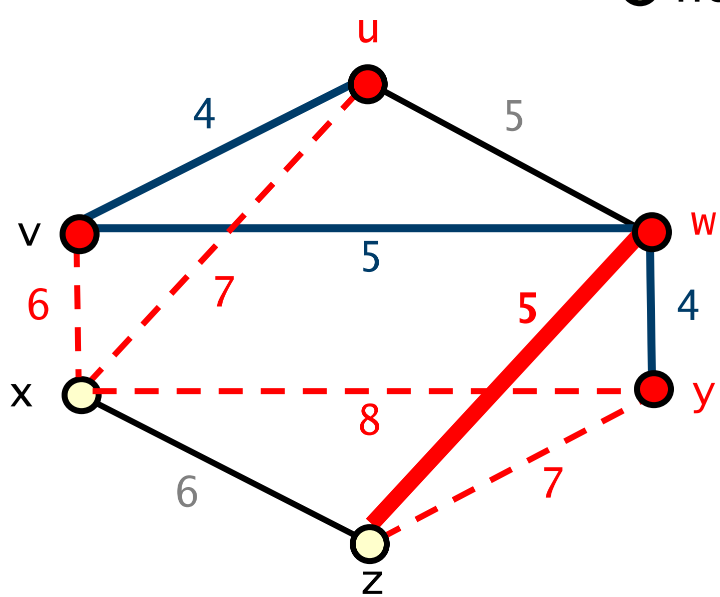
consider all tree vertex and non-tree vertex edges

The Prim–Jarnik algorithm – Example

Weighted graph G



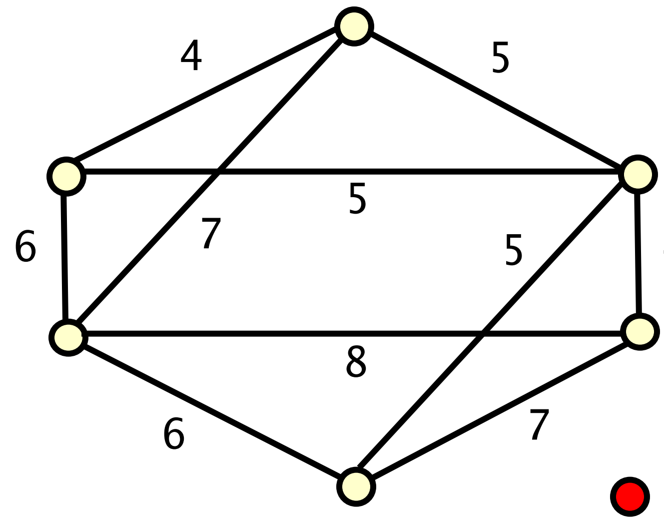
tree vertices: u,v,w,y
non-tree vertices: x,z



minimum such edge is {w,z}

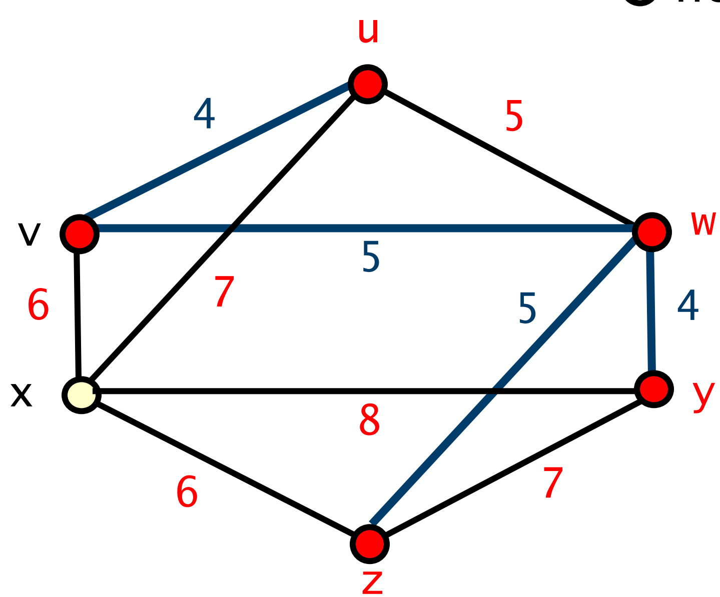
The Prim–Jarnik algorithm – Example

Weighted graph **G**



tree vertices: u,v,w,y,z

non–tree vertices: x

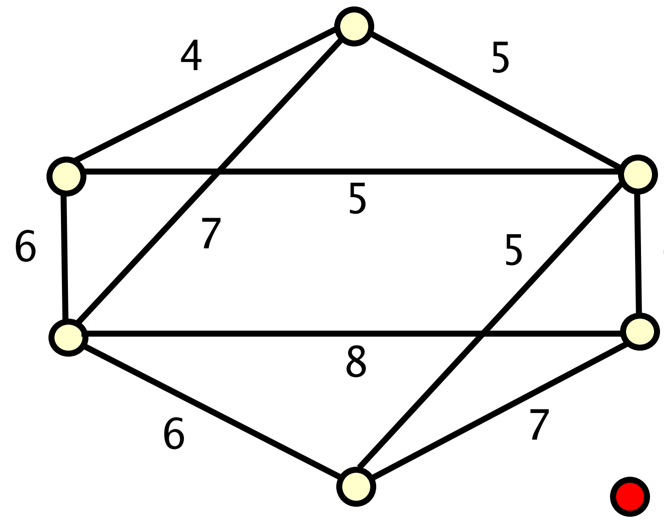


make z a tree vertex

& add edge to the tree

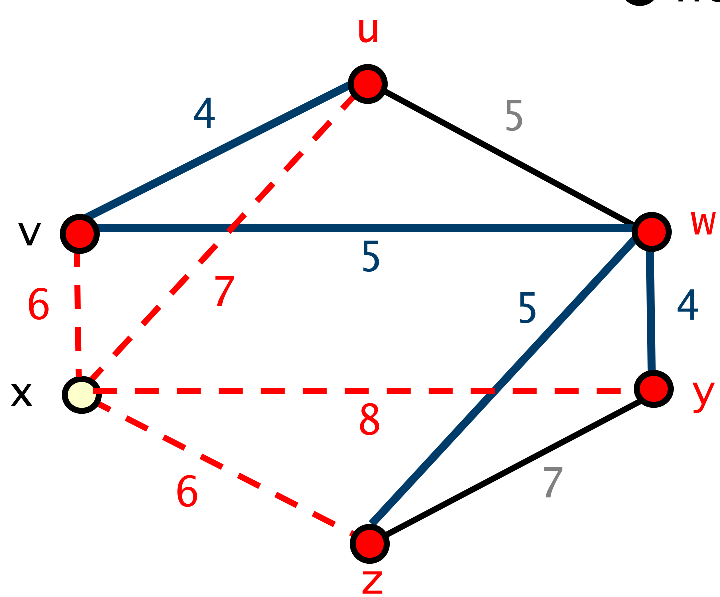
The Prim–Jarnik algorithm – Example

Weighted graph G



tree vertices: u,v,w,y,z

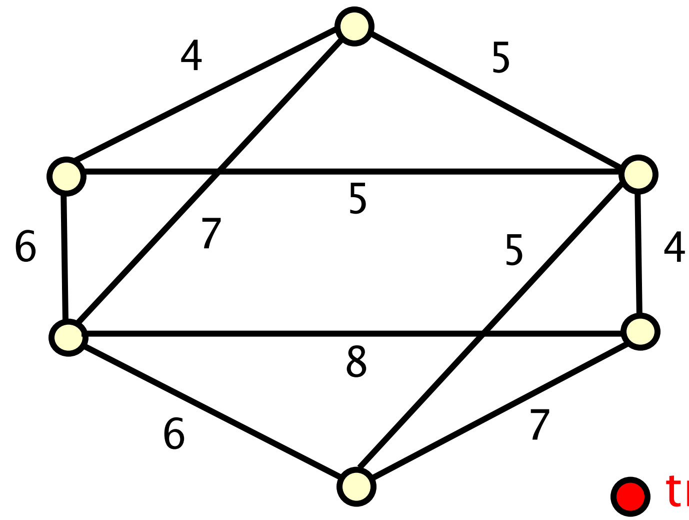
non-tree vertices: x



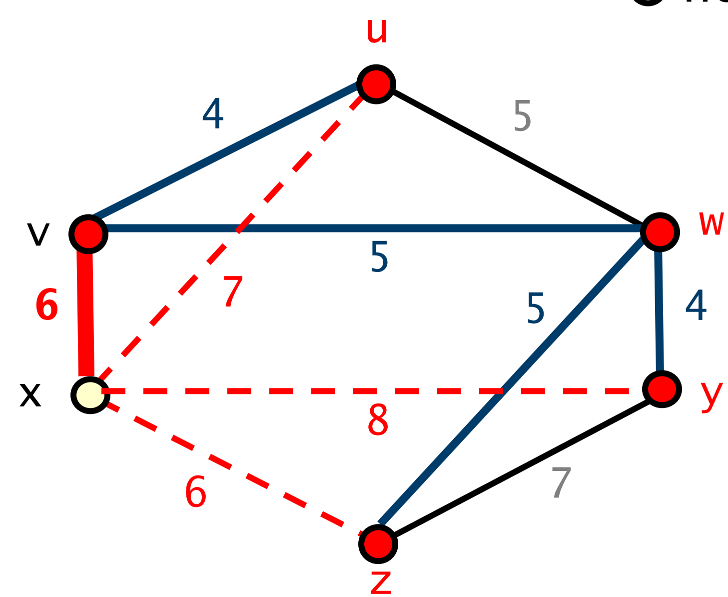
consider all tree vertex and non-tree vertex edges

The Prim–Jarnik algorithm – Example

Weighted graph G



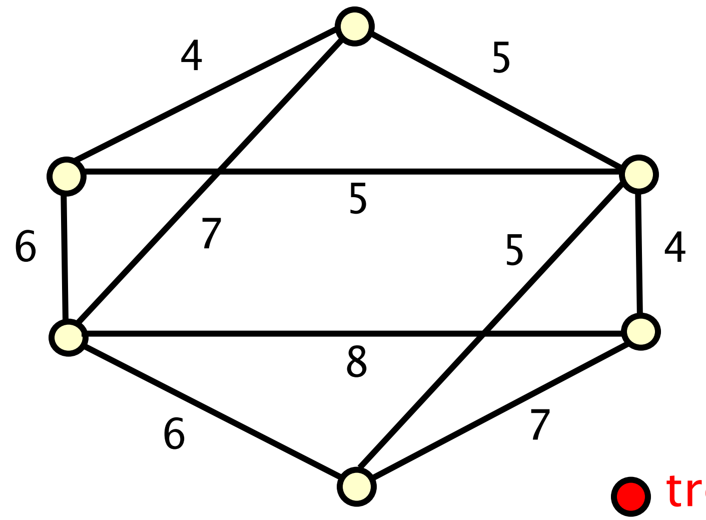
● tree vertices: u, v, w, y, z
○ non-tree vertices: x



minimum such edge is {x, v}

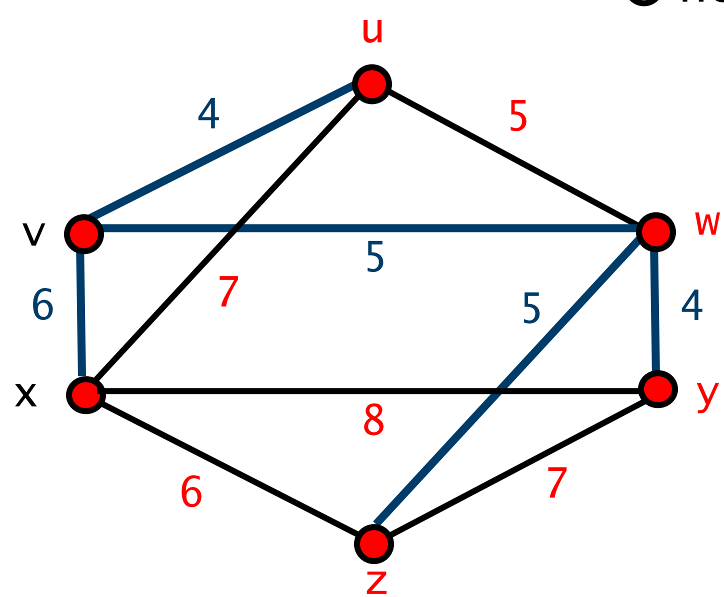
The Prim–Jarnik algorithm – Example

Weighted graph **G**



tree vertices: u, v, w, x, y, z

non-tree vertices:

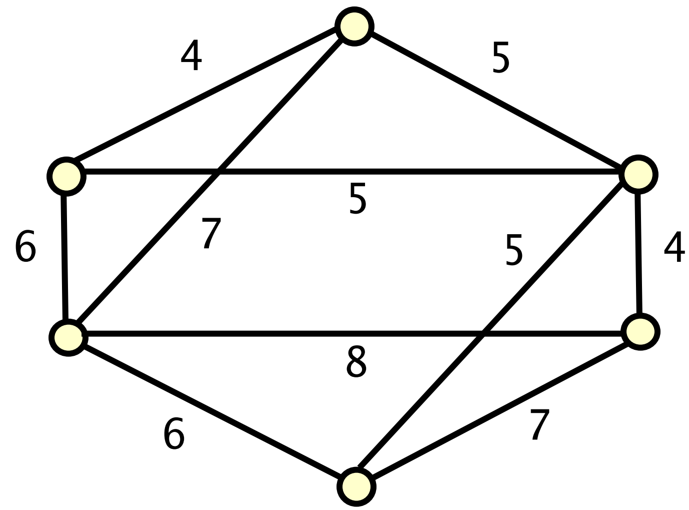


make x a tree vertex

& add edge to the tree

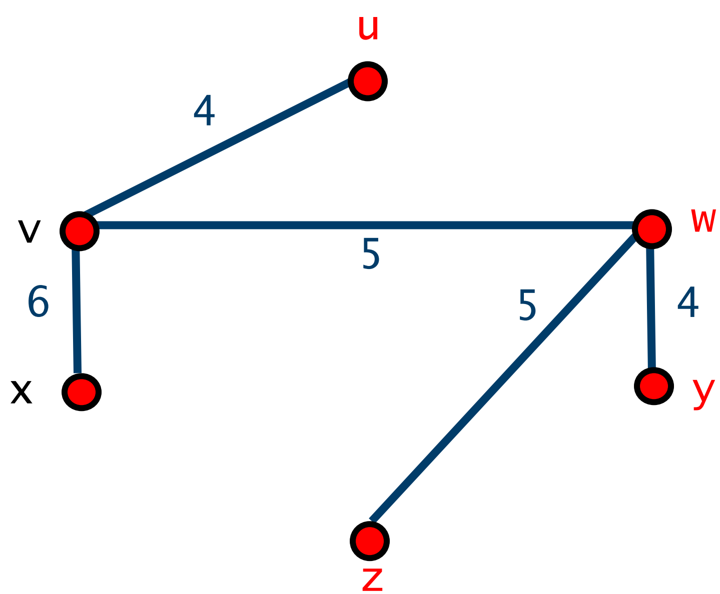
The Prim–Jarnik algorithm – Example

Weighted graph G



Minimum spanning tree for G

- weight 24



The Prim–Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edge

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

The Prim-Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edge

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

Analysis (**n** is the number of vertices)

The Prim–Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edge

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

Analysis (n is the number of vertices)

- initialisation O(n) (n operations to set vertices to be **tv** or **ntv**)

The Prim–Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edge

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

Analysis (n is the number of vertices)

- initialisation O(n) (n operations to set vertices to be **tv** or **ntv**)
- the outer loop is executed n-1 times
  - initially n-1 ntv vertices and each iteration turns one ntv to a tv

The Prim–Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edge

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

Analysis (n is the number of vertices)

- initialisation O(n) (n operations to set vertices to be **tv** or **ntv**)
- the outer loop is executed n-1 times
- the inner loop checks all edges from a tree-vertex to a non-tree-vertex
- there can be $O(n^2)$ of these

The Prim–Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edge

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

Analysis (**n** is the number of vertices)

- initialisation O(n) (n operations to set vertices to be **tv** or **ntv**)
- the outer loop is executed n-1 times
- the inner loop $O(n^2)$
- updating tree O(1) each iteration

The Prim-Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edge

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

Analysis (n is the number of vertices)

- initialisation $O(n)$ (n operations to set vertices to be **tv** or **ntv**)
- the outer loop is executed $n-1$ times
- the inner loop $O(n^2)$ and updating tree $O(1)$ each iteration
- so overall the algorithm is $O(n) + O(n^3) = O(n^3)$

Section 3 – Graphs and graph algorithms

Graph basics

- definitions: directed, undirected, connected, bipartite, …

Graph representations

- adjacency matrix/lists and implementation

Graph search and traversal algorithms

- breadth/depth first search

Topological ordering

Weighted graphs

- shortest path (Dijkstra's algorithm)
- minimum spanning tree (Prim-Jarnik and Dijkstra's refinement)

The Prim–Jarnik algorithm

Min spanning tree is constructed by choosing a sequence of edge

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
while (number of ntv > 0){
    find edge e = {p,q} of graph such that
        p is a tv;
        q is an ntv;
        wt(e) is minimised over such edges;
    adjoin edge e to the (spanning) tree;
    make q a tv;
}
```

Analysis (n is the number of vertices)

- initialisation $O(n)$ (n operations to set vertices to be **tv** or **ntv**)
- the outer loop is executed $n-1$ times
- **the inner loop $O(n^2)$ and updating tree $O(1)$ each iteration (!)**
- so overall the algorithm is $O(n) + O(n^3) = O(n^3)$

Dijkstra's refinement

Introduce an attribute bestTV for each non-tree vertex (ntv) q

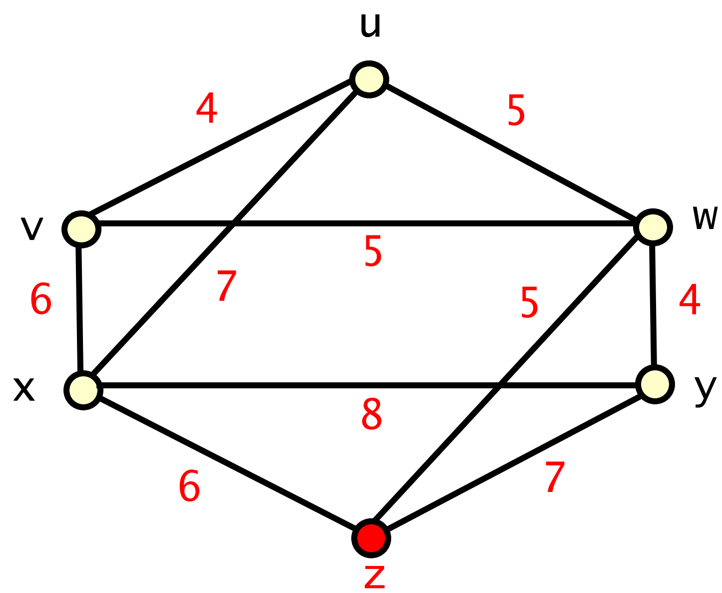
- bestTV is set to the tree vertex (tv) p for which wt({p,q}) is minimised

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
for (each ntv s) set s.bestTV = r; // r is the only tv

while (number of ntv > 0){
  find ntv q for which wt({q, q.bestTV}) is minimal;
  adjoin {q, q.bestTV} to the tree;
  make q a tv;

  for (each ntv s) update s.bestTV;
  // update bestTV as tree vertices have changed
}
```

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which
  wt({q, q.bestTV}) is minimal
  and make q a **tv**
- update s.bestTV for all **ntv** s

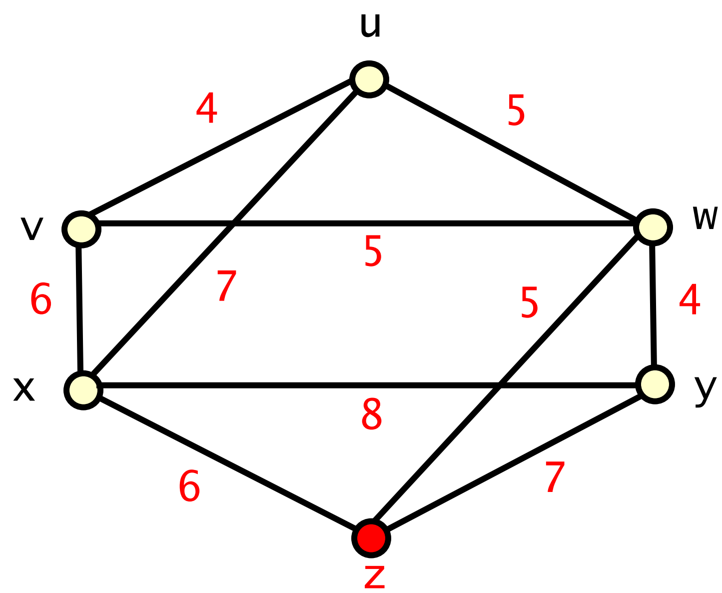


tree vertices: z
non-tree vertices: u,v,w,x,y

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	z	∞
v	z	∞
w	z	5
x	z	6
y	z	7
z	-	-

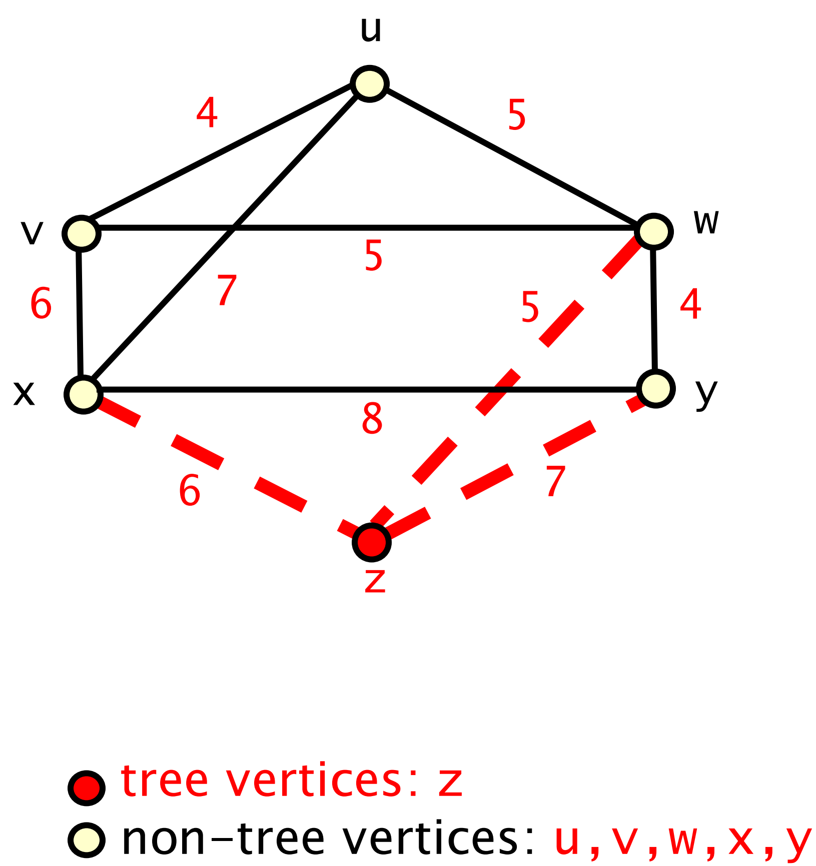
tree vertices: z
non-tree vertices: u,v,w,x,y

initialise **bestTV** to the only **tv** z

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s

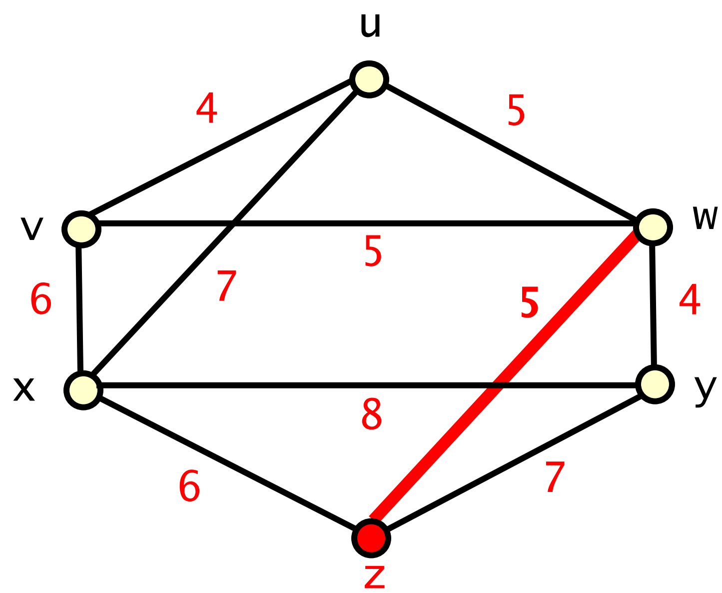


q	q.bestTV	wt({q.bestTV,q})
u	z	∞
v	z	∞
w	z	5
x	z	6
y	z	7
z	-	-

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



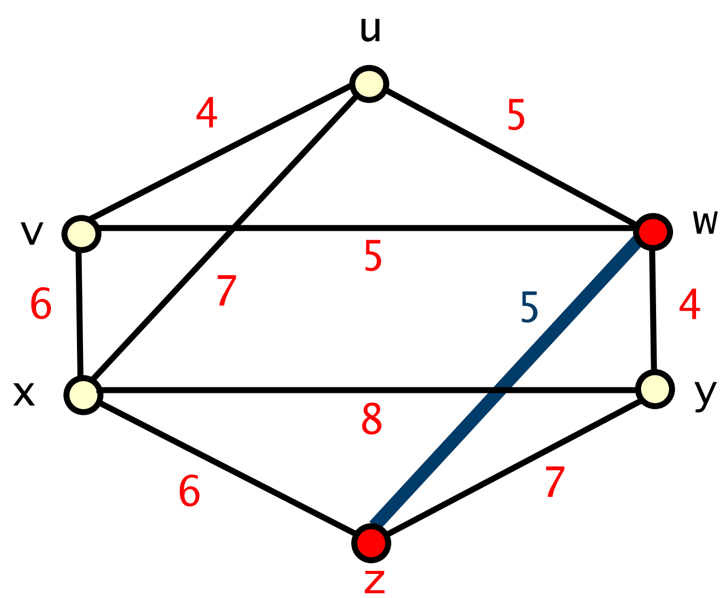
q	q.bestTV	wt({q.bestTV,q})
u	z	∞
v	z	∞
w	**z**	**5**
x	z	6
y	z	7
z	-	-

tree vertices: z
non-tree vertices: u,v,w,x,y

Dijkstra's refinement – Example

Weighted graph **G**

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



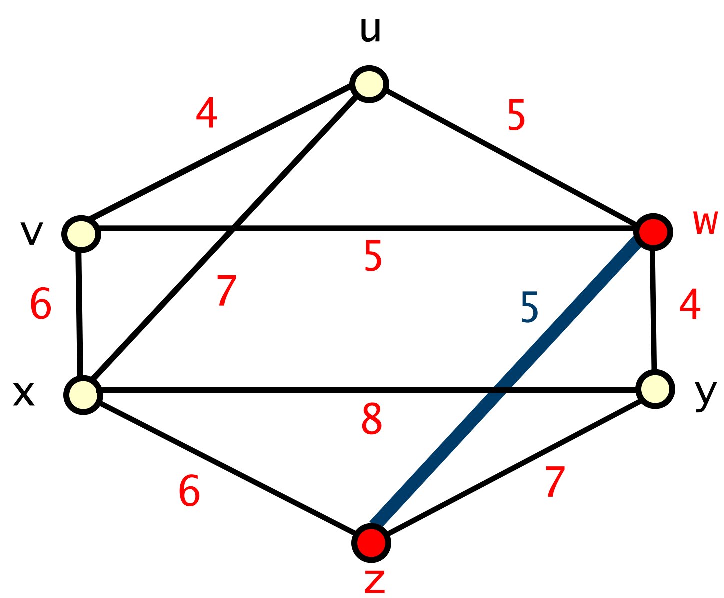
q	q.bestTV	wt({q.bestTV,q})
u	z	∞
v	z	∞
w	z	5
x	z	6
y	z	7
z	-	-

● tree vertices: w,z
○ non-tree vertices: u,v,x,y

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	z→w	∞→5
v	z→w	∞→5
w	-	-
x	z	6
y	z→w	7→4
z	-	-

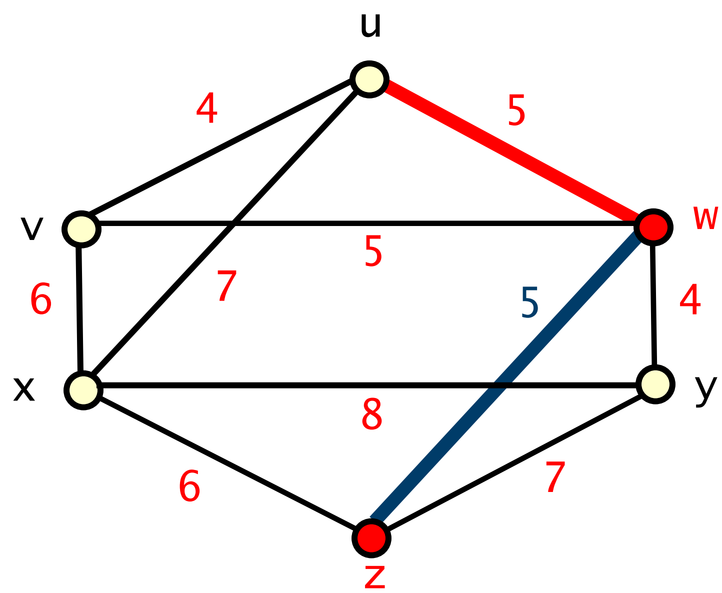
tree vertices: w,z
non-tree vertices: u,v,x,y

check bestTV against new tv vertex w

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	**z→w**	**∞→5**
v	z→w	∞→5
w	-	-
x	z	6
y	z→w	7→4
z	-	-

tree vertices: w,z

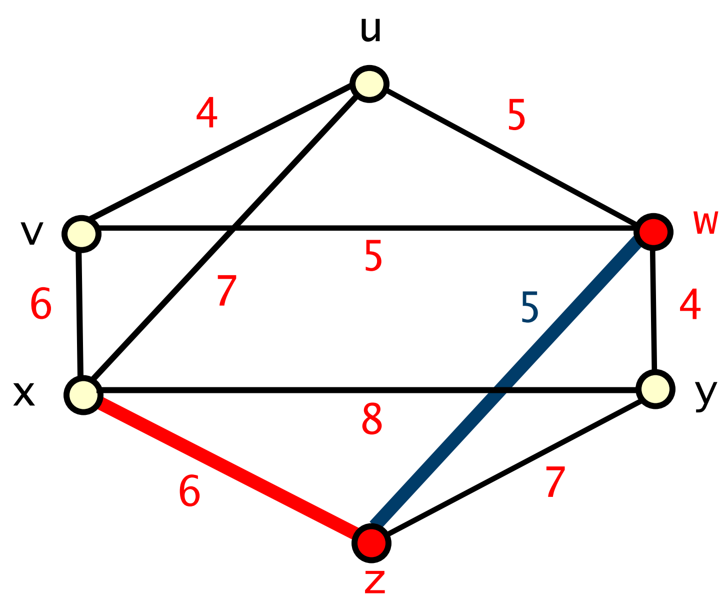
non-tree vertices: u,v,x,y

check bestTV against new tv vertex w

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	z→w	∞→5
v	z→w	∞→5
w	-	-
x	**z**	**6**
y	z→w	7→4
z	-	-

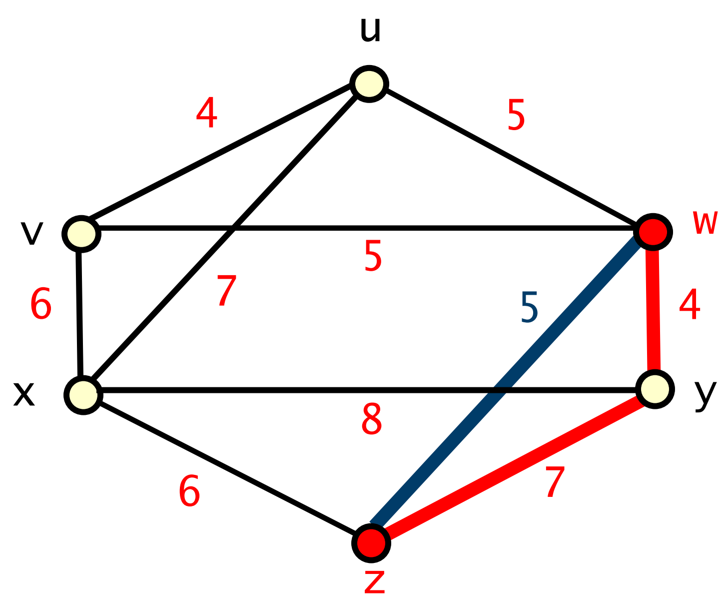
- tree vertices: w,z
- non-tree vertices: u,v,x,y

check bestTV against new tv vertex w

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	z→w	∞→5
v	z→w	∞→5
w	-	-
x	z	6
y	**z→w**	**7→4**
z	-	-

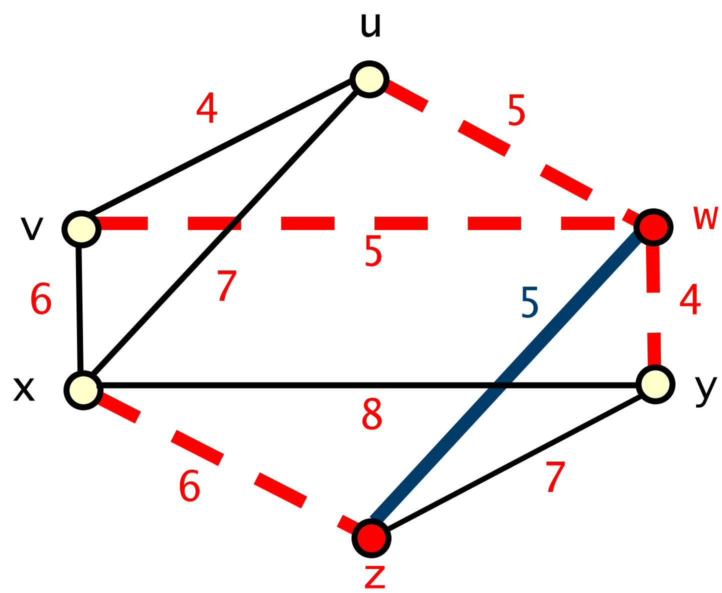
tree vertices: w,z
non-tree vertices: u,v,x,y

check bestTV against new tv vertex w

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



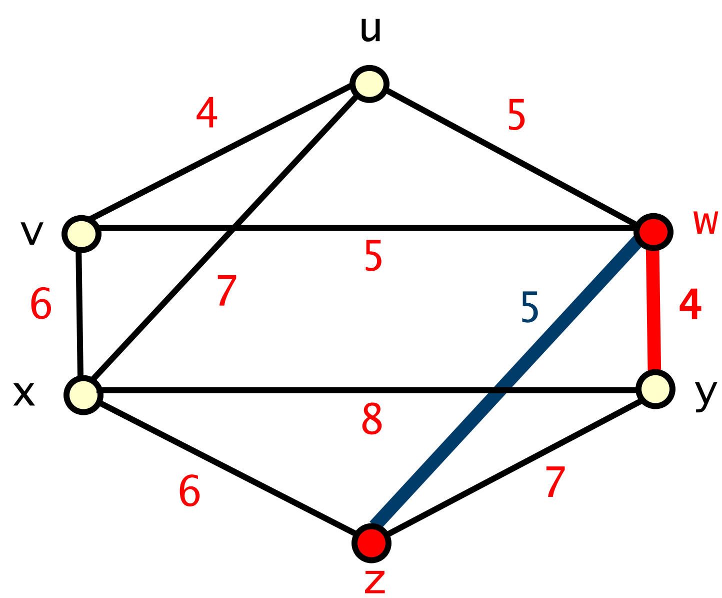
q	q.bestTV	wt({q.bestTV,q})
u	w	5
v	w	5
w	-	-
x	z	6
y	w	4
z	-	-

tree vertices: w,z
non-tree vertices: u,v,x,y

Dijkstra's refinement – Example

Weighted graph **G**

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



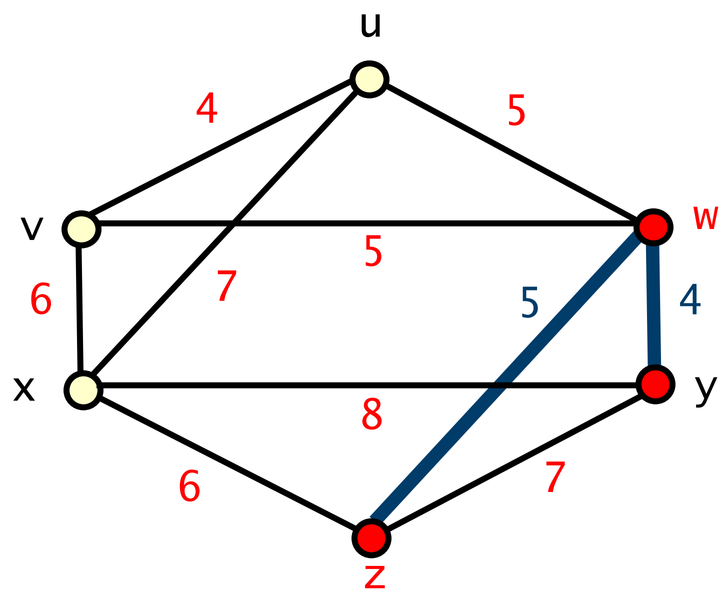
q	q.bestTV	wt({q.bestTV,q})
u	w	5
v	w	5
w	-	-
x	z	6
y	**w**	**4**
z	-	-

- tree vertices: w,z
- non-tree vertices: u,v,x,y

Dijkstra's refinement – Example

Weighted graph **G**

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



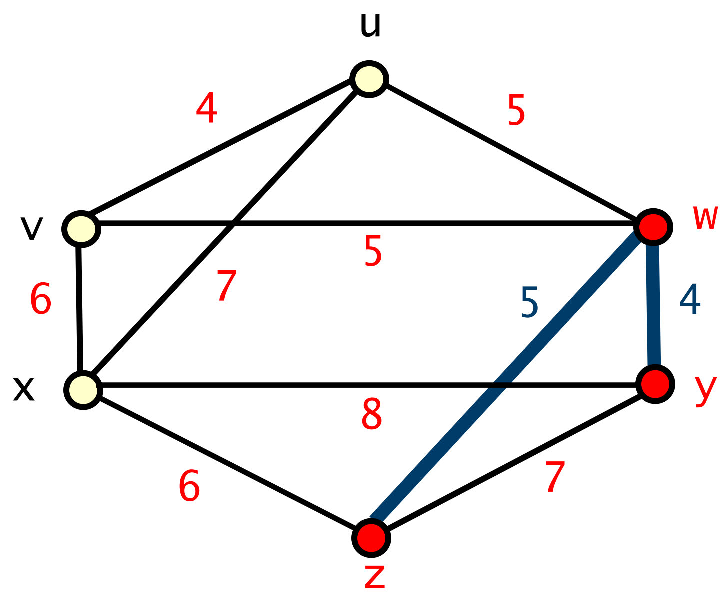
q	q.bestTV	wt({q.bestTV,q})
u	w	5
v	w	5
w	-	-
x	z	6
y	w	4
z	-	-

● tree vertices: w,y,z
○ non-tree vertices: u,v,x

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	w	5
v	w	5
w	-	-
x	z	6
y	-	-
z	-	-

tree vertices: w,y,z

non-tree vertices: u,v,x

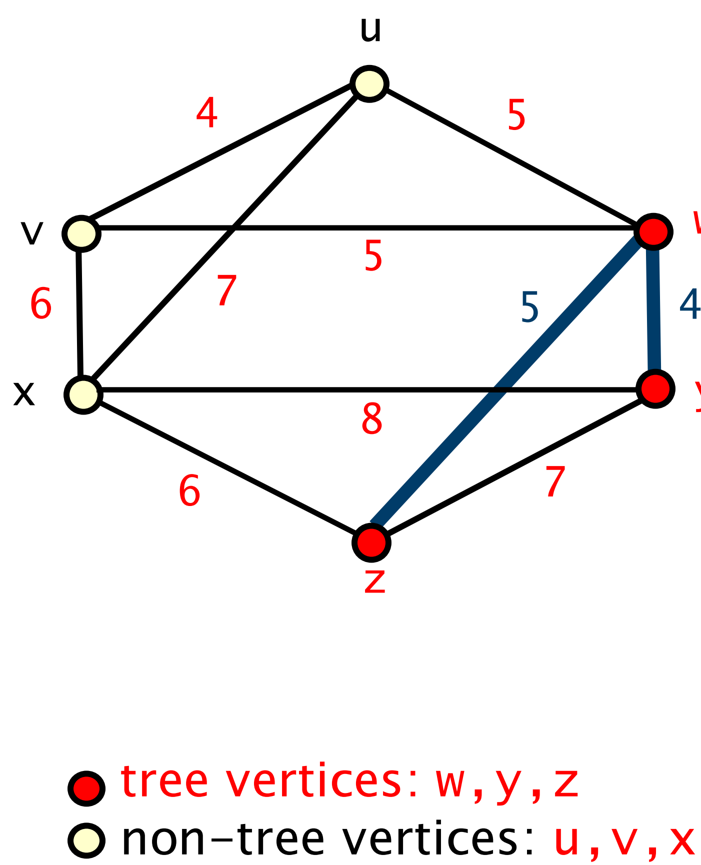
check bestTV against new tv vertex y

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s

q	q.bestTV	wt({q.bestTV,q})
u	**w**	**5**
v	**w**	**5**
w	-	-
x	z	6
y	-	-
z	-	-

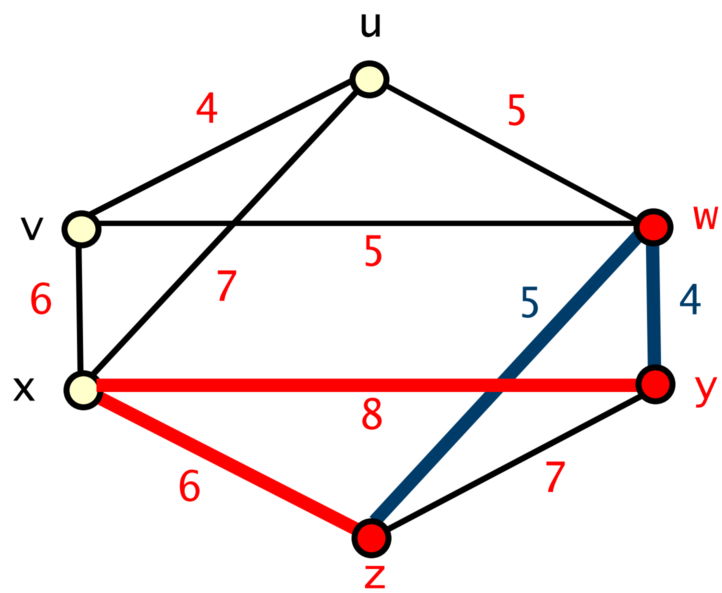


check bestTV against new tv vertex y

Dijkstra's refinement – Example

Weighted graph **G**

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update `s.bestTV` for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	w	5
v	w	5
w	-	-
x	**z**	**6**
y	-	-
z	-	-

tree vertices: w,y,z
non-tree vertices: u,v,x

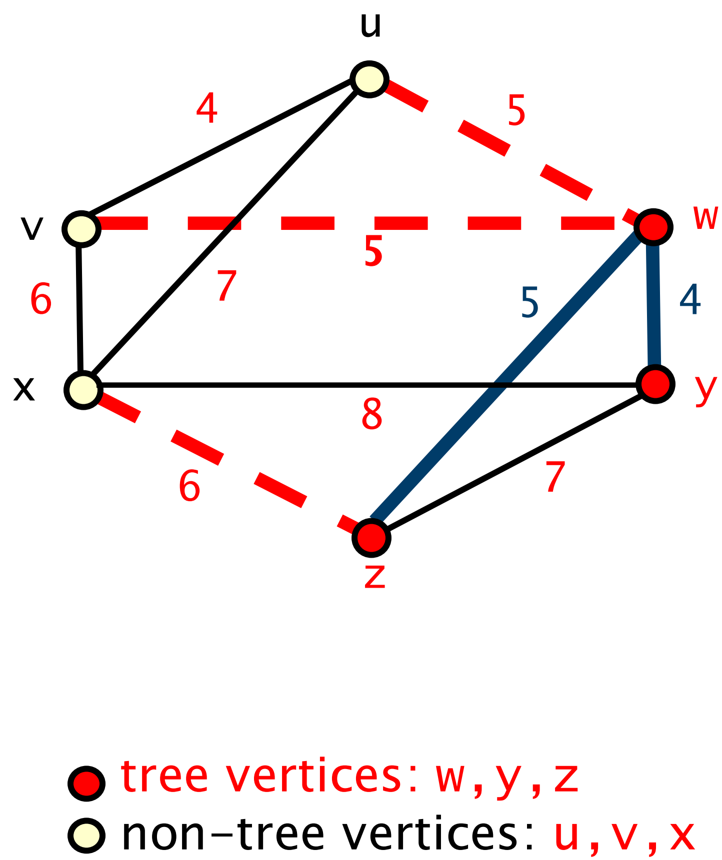
check bestTV against new tv vertex y

Dijkstra's refinement – Example

Weighted graph G

– choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**

– update s.bestTV for all **ntv** s

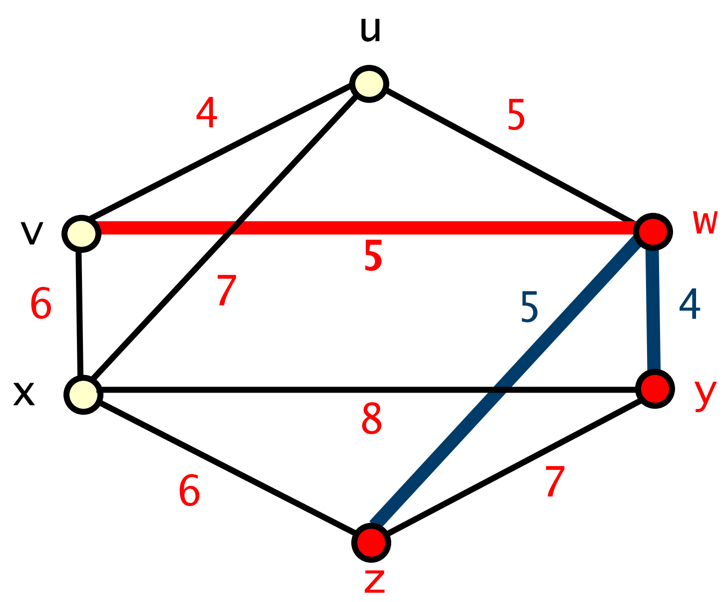


q	q.bestTV	wt({q.bestTV,q})
u	w	5
v	w	5
w	-	-
x	z	6
y	-	-
z	-	-

Dijkstra’s refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



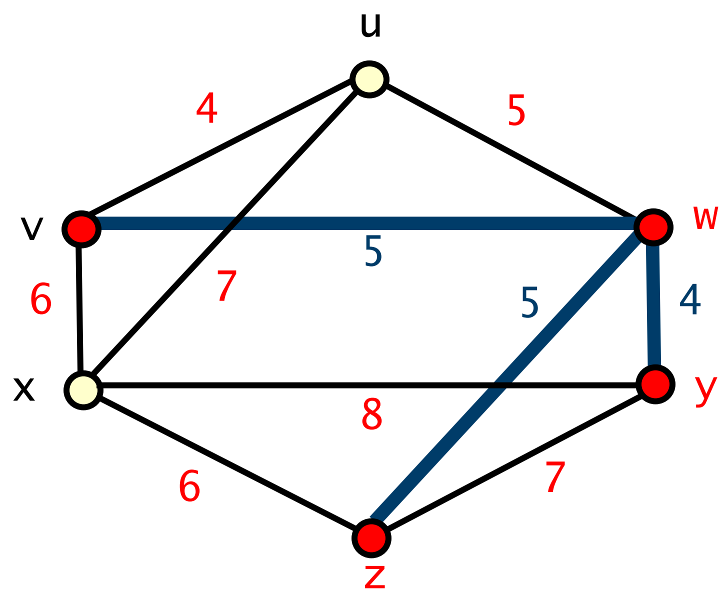
q	q.bestTV	wt({q.bestTV,q})
u	w	5
v	**w**	**5**
w	-	-
x	z	6
y	-	-
z	-	-

- tree vertices: w,y,z
- non-tree vertices: u,v,x

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which
  wt({q, q.bestTV}) is minimal
  and make q a **tv**
- update s.bestTV for all **ntv** s



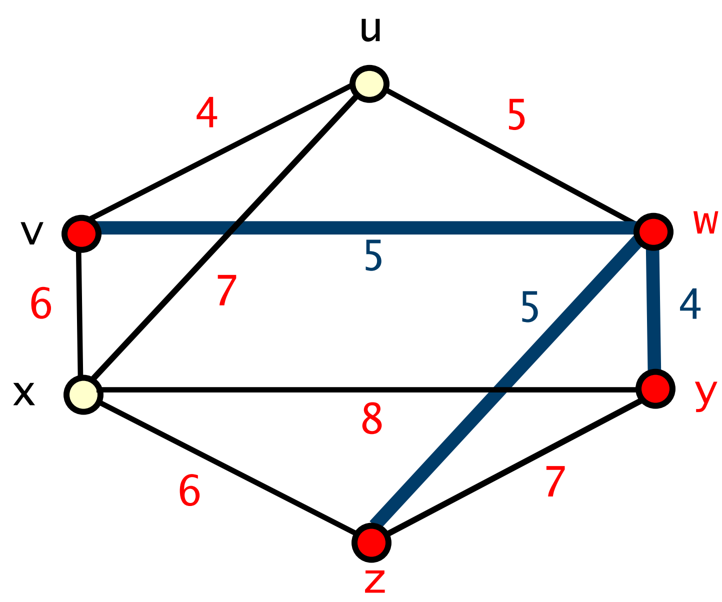
q	q.bestTV	wt({q.bestTV,q})
u	w	5
v	w	5
w	-	-
x	z	6
y	-	-
z	-	-

tree vertices: v,w,y,z
non-tree vertices: u,x

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update `s.bestTV` for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	w→v	5→4
v	-	-
w	-	-
x	z	6
y	-	-
z	-	-

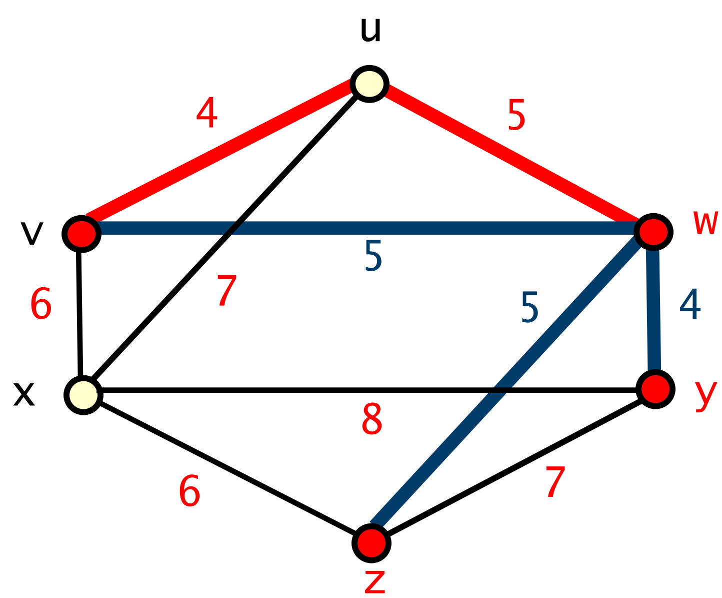
tree vertices: v,w,y,z
non-tree vertices: u,x

check bestTV against new tv vertex v

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	**w→v**	**5→4**
v	-	-
w	-	-
x	z	6
y	-	-
z	-	-

tree vertices: v,w,y,z

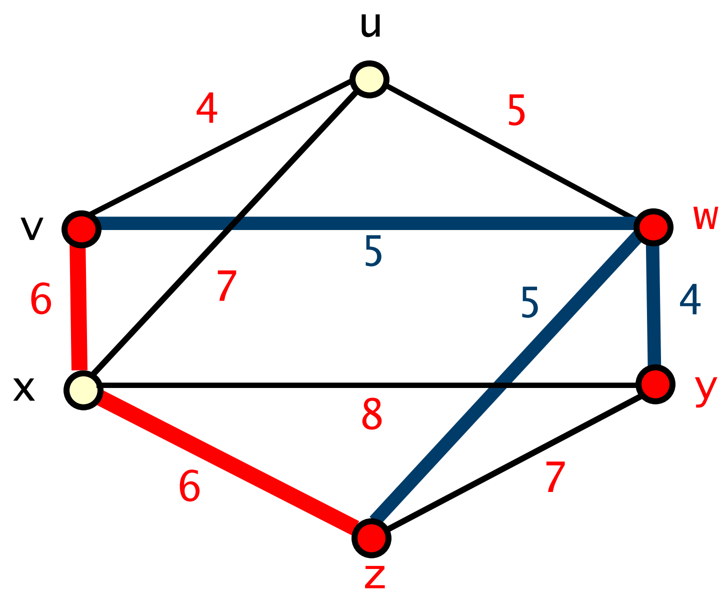
non-tree vertices: u,x

check bestTV against new tv vertex v

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	w→v	5→4
v	-	-
w	-	-
x	**z**	**6**
y	-	-
z	-	-

tree vertices: v,w,y,z

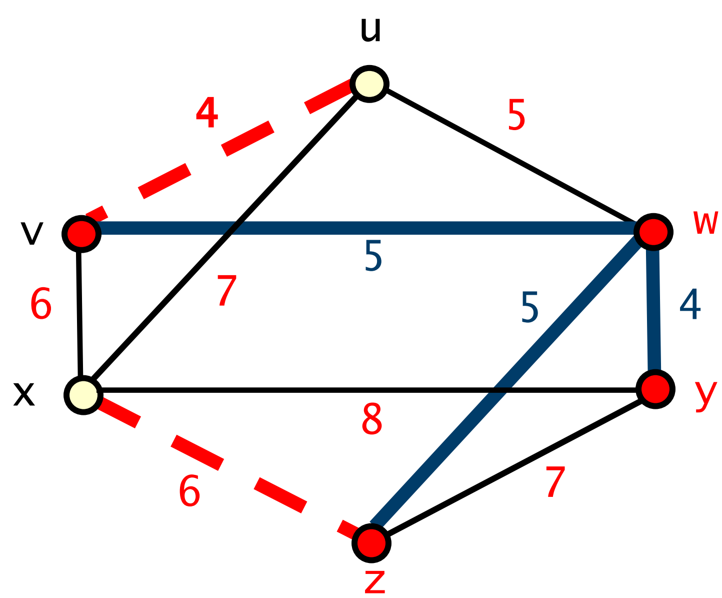
non-tree vertices: u,x

check bestTV against new tv vertex v

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



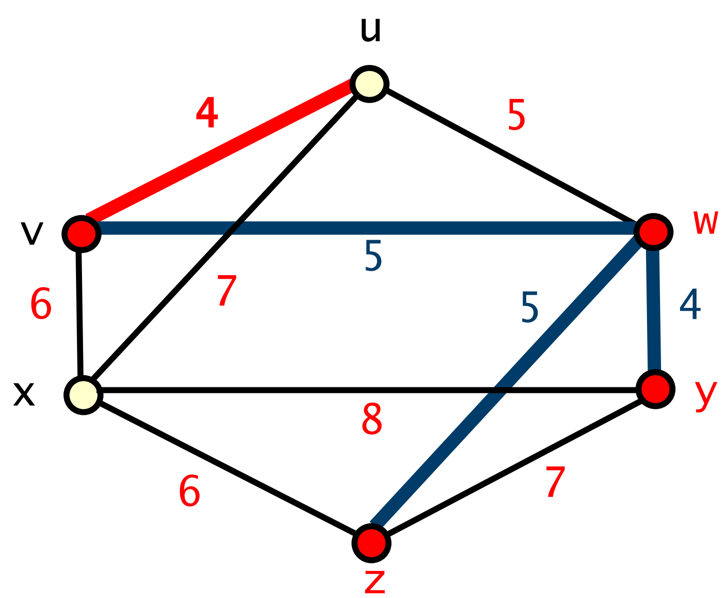
q	q.bestTV	wt({q.bestTV,q})
u	v	4
v	-	-
w	-	-
x	z	6
y	-	-
z	-	-

tree vertices: v,w,y,z
non-tree vertices: u,x

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



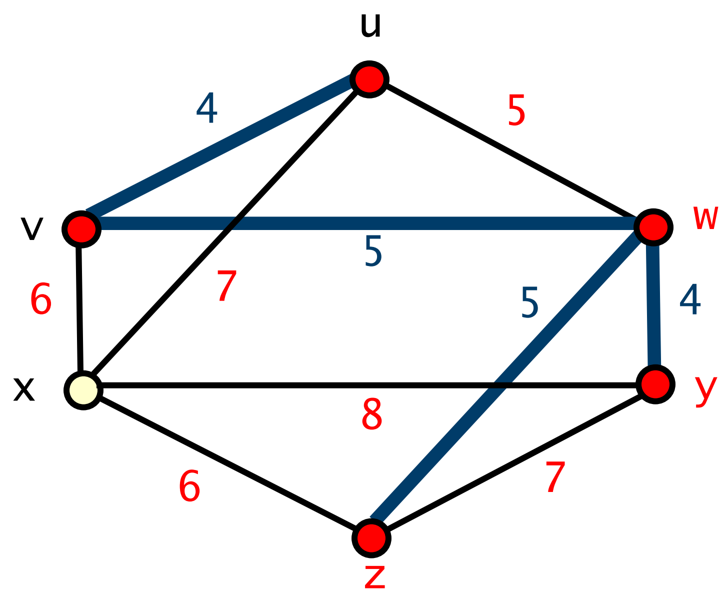
q	q.bestTV	wt({q.bestTV,q})
u	**v**	**4**
v	-	-
w	-	-
x	z	6
y	-	-
z	-	-

tree vertices: v,w,y,z
non-tree vertices: u,x

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



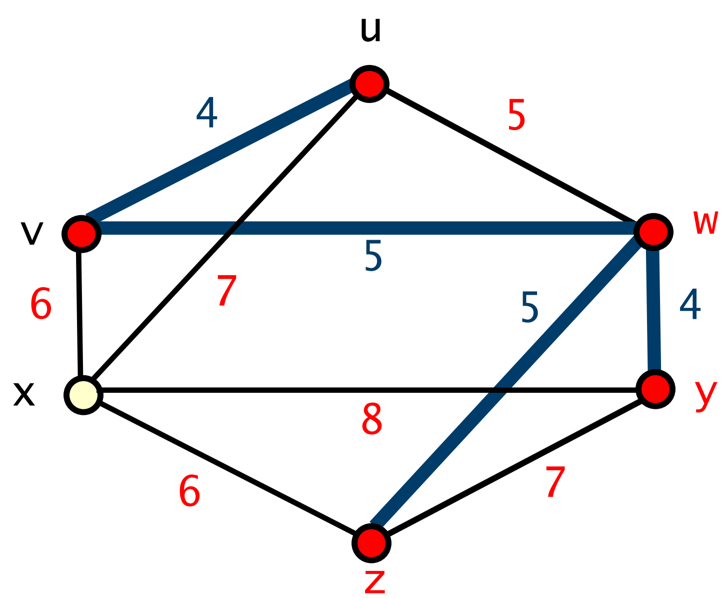
q	q.bestTV	wt({q.bestTV,q})
u	v	4
v	-	-
w	-	-
x	z	6
y	-	-
z	-	-

- tree vertices: u,v,w,y,z
- non-tree vertices: x

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	-	-
v	-	-
w	-	-
x	z	6
y	-	-
z	-	-

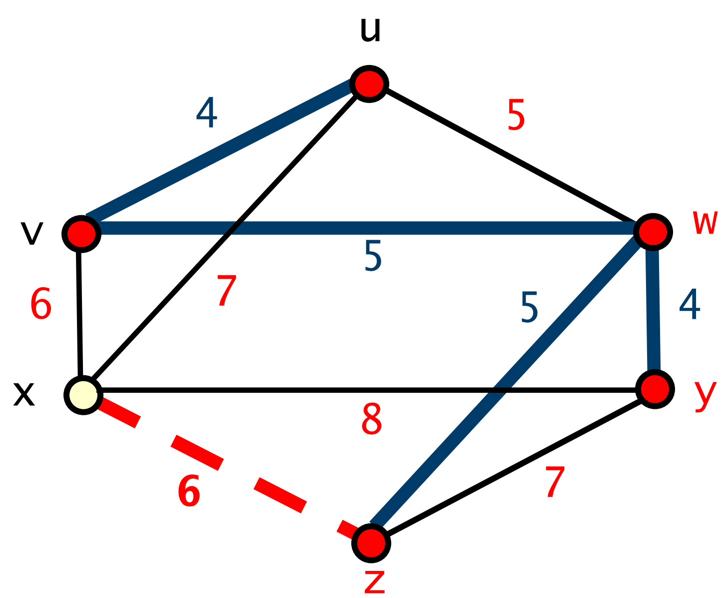
tree vertices: u,v,w,y,z
non-tree vertices: x

check bestTV against new tv vertex u

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



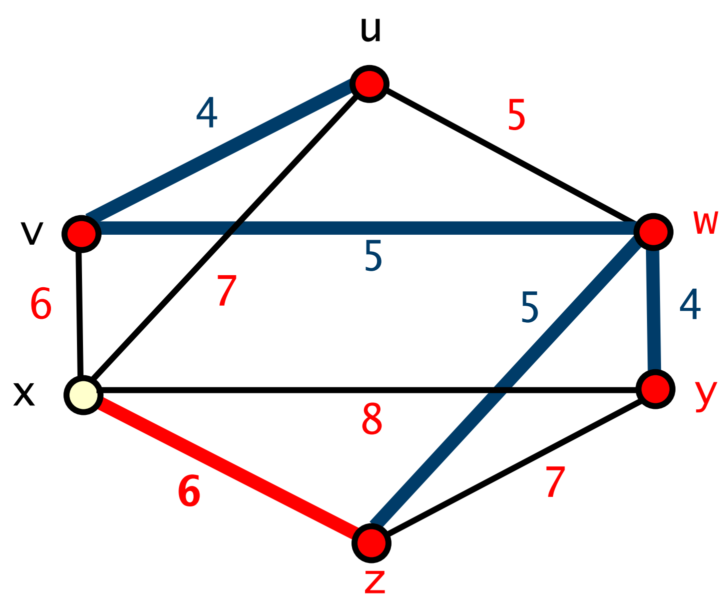
q	q.bestTV	wt({q.bestTV,q})
u	-	-
v	-	-
w	-	-
x	z	6
y	-	-
z	-	-

tree vertices: u,v,w,y,z
non-tree vertices: x

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which
  wt({q, q.bestTV}) is minimal
  and make q a **tv**
- update s.bestTV for all **ntv** s



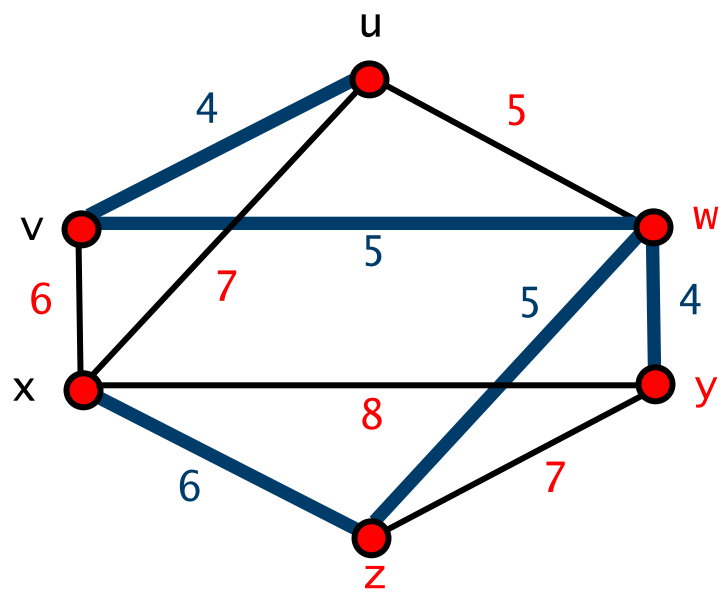
q	q.bestTV	wt({q.bestTV,q})
u	-	-
v	-	-
w	-	-
x	**z**	**6**
y	-	-
z	-	-

tree vertices: u,v,w,y,z
non-tree vertices: x

Dijkstra's refinement – Example

Weighted graph G

- choose **ntv** q for which **wt({q, q.bestTV})** is minimal and make q a **tv**
- update s.bestTV for all **ntv** s



q	q.bestTV	wt({q.bestTV,q})
u	-	-
v	-	-
w	-	-
x	-	-
y	-	-
z	-	-

tree vertices: u,v,w,x,y,z

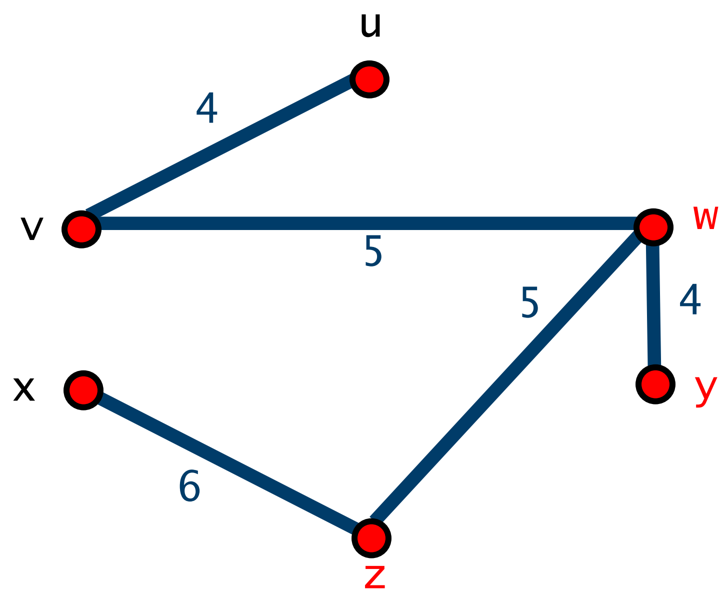
non-tree vertices:

Dijkstra's refinement – Example

Weighted graph G

Minimum spanning tree for G

– weight 24



q	q.bestTV	wt({q.bestTV,q})
u	–	–
v	–	–
w	–	–
x	–	–
y	–	–
z	–	–

Dijkstra’s refinement

Introduce an attribute bestTV for each non-tree vertex (ntv) q

– bestTV is set to the tree vertex (tv) p for which wt({p,q}) is minimised

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
for (each ntv s) set s.bestTV = r; // r is the only tv

while (number of ntv > 0){
  find ntv q for which wt({q, q.bestTV}) is minimal;
  adjoin {q, q.bestTV} to the tree;
  make q a tv;

  for (each ntv s) update s.bestTV;
  // update bestTV as tree vertices have changed
}
```

Dijkstra's refinement – Analysis

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
for (each ntv s) set s.bestTV = r; // r is the only tv

while (number of ntv > 0){
  find ntv q for which wt({q, q.bestTV}) is minimal;
  adjoin {q, q.bestTV} to the tree;
  make q a tv;

  for (each ntv s) update s.bestTV; // update as tvs have changed
}
```

– initialisation is O(n)

Dijkstra's refinement – Analysis

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
for (each ntv s) set s.bestTV = r; // r is the only tv

while (number of ntv > 0){
  find ntv q for which wt({q, q.bestTV}) is minimal;
  adjoin {q, q.bestTV} to the tree;
  make q a tv;

  for (each ntv s) update s.bestTV; // update as tvs have changed
}
```

- initialisation is O(n)
- while loop is executed n-1 times

Dijkstra's refinement – Analysis

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
for (each ntv s) set s.bestTV = r; // r is the only tv

while (number of ntv > 0){
  find ntv q for which wt({q, q.bestTV}) is minimal;
  adjoin {q, q.bestTV} to the tree;
  make q a tv;

  for (each ntv s) update s.bestTV; // update as tvs have changed
}
```

- initialisation is O(n)
- while loop is executed n-1 times
- first part takes O(n)
  - O(n) to find minimal **ntv** and O(1) to adjoin and update

Dijkstra's refinement – Analysis

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
for (each ntv s) set s.bestTV = r; // r is the only tv

while (number of ntv > 0){
  find ntv q for which wt({q, q.bestTV}) is minimal;
  adjoin {q, q.bestTV} to the tree;
  make q a tv;

  for (each ntv s) update s.bestTV; // update as tvs have changed
}
```

- initialisation is O(n)
- while loop is executed n-1 times
- first part takes O(n) and second part takes O(n)
  - for each **ntv** s only need to compare weights for **s.bestTV** and new **tv** vertex (i.e. q) to update the value of **s.bestTV**

Dijkstra's refinement – Analysis

```
set an arbitrary vertex r to be a tree-vertex (tv);
set all other vertices to be non-tree-vertices (ntv);
for (each ntv s) set s.bestTV = r; // r is the only tv

while (number of ntv > 0){
  find ntv q for which wt({q, q.bestTV}) is minimal;
  adjoin {q, q.bestTV} to the tree;
  make q a tv;

  for (each ntv s) update s.bestTV; // update as tvs have changed
}
```

- initialisation is $O(n)$
- while loop is executed $n-1$ times
  - first and second parts each take $O(n)$ so for both parts $O(n)+O(n) = O(n)$
- overall the algorithm is $O(n) + O(n^2) = O(n^2)$
  - while Prim-Jarnik is $O(n^3)$

Plan for next week

In class test on Monday

- 1h30 class test, additional time 15 min to deal with submission, plus 30 minutes for those entitled to additional time
- online, word document to fill in your answers and upload
- open book assessment
- 3 questions: 2 on algorithms tracing and one on describing an algorithm to solve a problem (based on some algorithms covered in weeks 1 & 2)

The assessed exercise will be released on Tuesday

- based on graph theory
  - ask for help re: the AE during any subsequent tutorial session
- two lab sessions 2h each to prepare for and help with the AE
  - Wednesday 19 March with a lab exercise to get you going
  - Wednesday 26 March dedicated for working on the AE and ask for help

Quiz week 2 and mid–way feedback

The quiz for week 2 opens on Wednesday 12 March at 12:00

- 5 questions
- time limit 1
- closes on Friday at 22:00

Mid–way feedback opens on Wednesday 12 March at 11:00

- remains open until Wednesday 19 March 13:00

Outline of course

Section 0: Quick recap on algorithm analysis

Section 1: Sorting algorithms

Section 2: Strings and text algorithms

Section 3: Graphs and graph algorithms

Section 4: An introduction to NP completeness

Section 5: A (very) brief introduction to computability