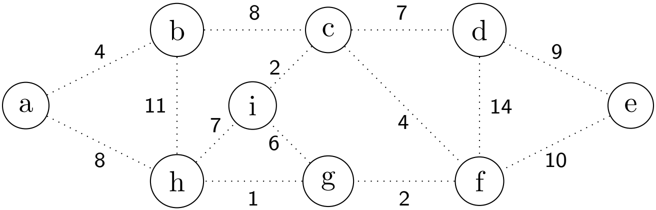
CSCB63 – Design and Analysis of Data Structures

Anya Tafliovich[1]

[1] based on notes by Anna Bretscher and Albert Lai

introduction



An (edge-)weighted graph

Applications?

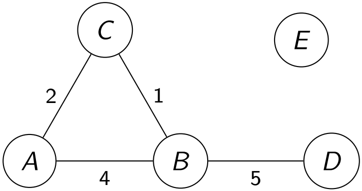
-

weighted graph

A weighted (edge-weighted) graph consists of:

- a set of vertices V
- a set of edges E
- weights: a map $w : E \to \mathbb{R}$ (usually ≥ 0)
    - if undirected graph: (u, v) and (v, u) have the same weight
    - if directed graph: (u, v) and (v, u) may have different weights

storing a weighted graph



Adjacency matrix:

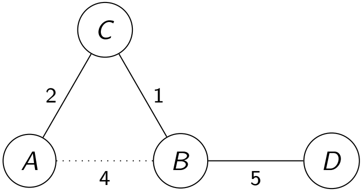
	A	B	C	D	E
A	0	4	2	∞	∞
B	4	0	1	5	∞
C	2	1	0	∞	∞
D	∞	5	∞	0	∞
E	∞	∞	∞	∞	0

Adjacency lists:

	adjacency list
A	$(B,4)$, $(C,2)$
B	$(A,4)$, $(C,1)$, $(D,5)$
C	$(A,2)$, $(B,1)$
D	$(B,5)$
E	

minimum spanning tree

- common task $\#1$ on weighted graphs
- find a spanning tree
  - a tree that covers all vertices
  - a tree T such that every vertex $v \in V$ is an endpoint of at least one edge in T
- minimise the sum of the weights of the edges used
  - $weight(T) = \sum_{(u,v) \in T} weight(u, v)$
  - want tree T with minimum $weight(T)$



Usually just for undirected, connected graphs.

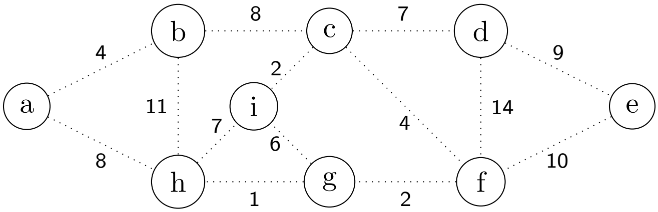
Kruskal's algorithm: idea

Kruskal's algorithm finds a MST by successive mergers.

1. At first, each vertex is its own small cluster/tree/set.
2. Find an edge of minimum weight, use it to merge two clusters/trees/sets into one.
   - Do not create cycles!
3. Do it again...
4. In general, find an edge of minimum weight that crosses two clusters; merge them into one.

Correctness idea: at each iteration find the cheapest way to merge two trees.

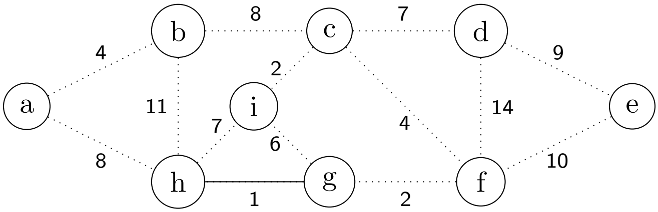
Kruskal's algorithm: example




```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {a}, {b}, {c}, {d}, {e}, {f}, {g}, {h}, {i}
MST: { }

Kruskal's algorithm: example

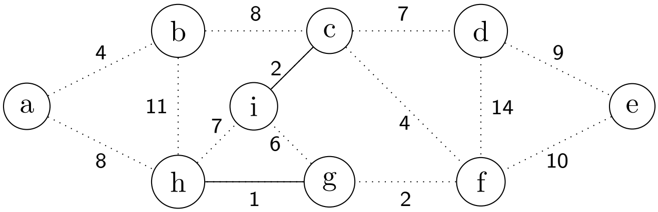



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {a}, {b}, {c}, {d}, {e}, {f}, {g,h}, {i}

MST: { (g,h), }

Kruskal's algorithm: example

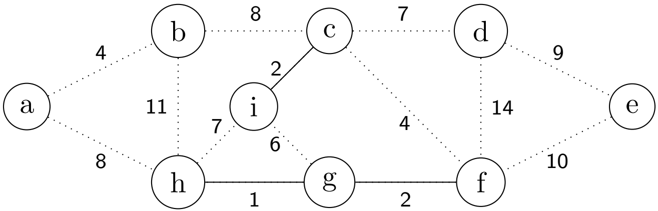



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {a}, {b}, {c,i}, {d}, {e}, {f}, {g,h}

MST: { (g,h), (c,i), }

Kruskal's algorithm: example

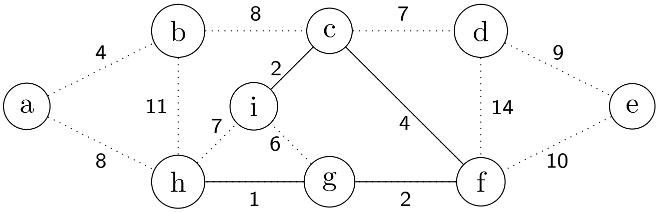



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: $\{a\}, \{b\}, \{c,i\}, \{d\}, \{e\}, \{f,g,h\}$

MST: $\{$ (g,h), (c,i), (f,g), $\}$

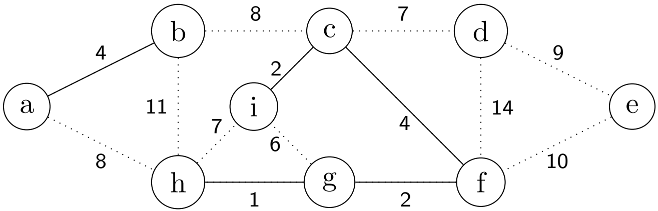
Kruskal's algorithm: example




```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {a}, {b}, {d}, {e}, {c,i,f,g,h}
MST: { (g,h), (c,i), (f,g), (c,f), }

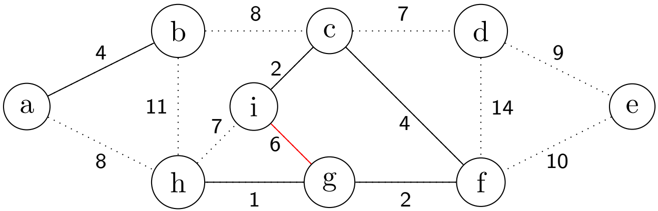
Kruskal's algorithm: example




```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {a,b}, {d}, {e}, {c,i,f,g,h}
MST: { (g,h), (c,i), (f,g), (c,f), (a,b), }

Kruskal's algorithm: example

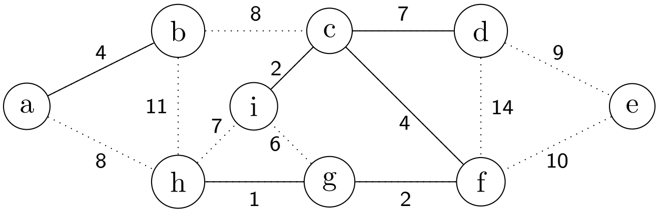



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {a,b}, {d}, {e}, {c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), }

Kruskal's algorithm: example

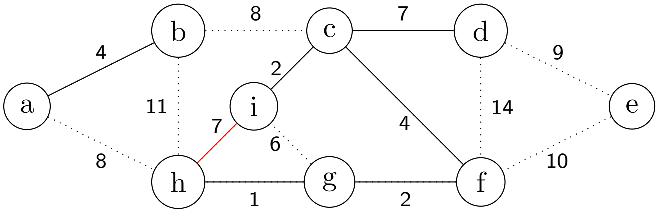



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {a,b}, {e}, {d,c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), }

Kruskal's algorithm: example

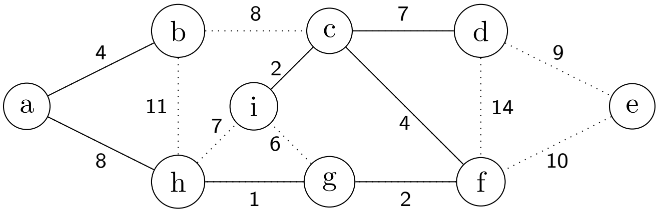



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {a,b}, {e}, {d,c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), }

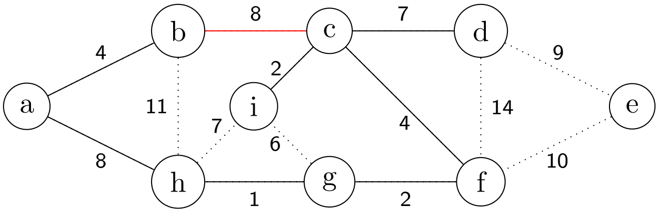
Kruskal's algorithm: example




```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {e}, {a,b,d,c,i,f,g,h}
MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), (a,h), }

Kruskal's algorithm: example

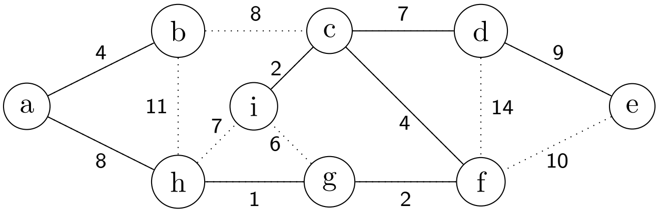



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {e}, {a,b,d,c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), (a,h), }

Kruskal's algorithm: example

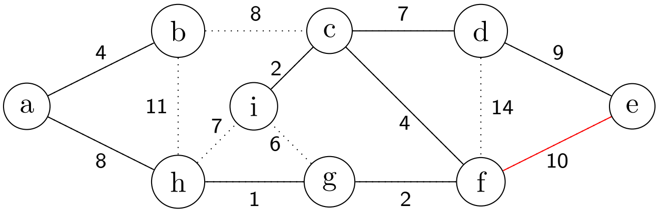



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {e,a,b,d,c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), (a,h), (d,e) }

Kruskal's algorithm: example

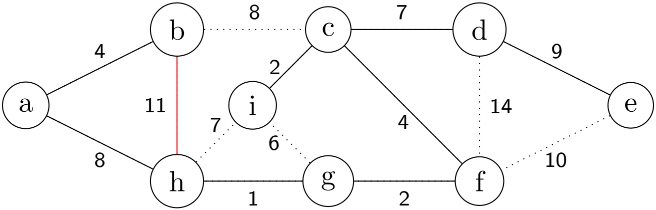



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {e,a,b,d,c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), (a,h), (d,e) }

Kruskal's algorithm: example

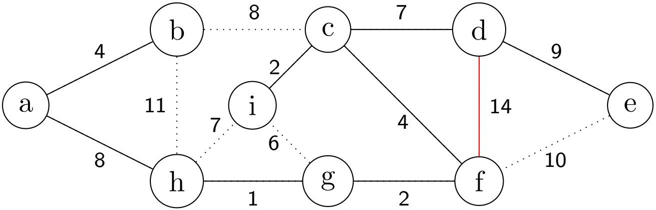



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {e,a,b,d,c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), (a,h), (d,e) }

Kruskal's algorithm: example

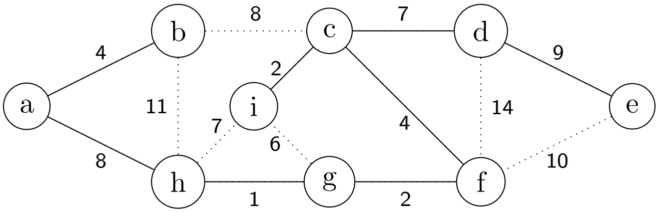



```
L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),
    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),
    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]
```

Clusters: {e,a,b,d,c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), (a,h), (d,e) }

Kruskal's algorithm: example



`L: [(g,h,1), (c,i,2), (f,g,2), (c,f,4), (a,b,4),`
`    (g,i,6), (c,d,7), (h,i,7), (a,h,8), (b,c,8),`
`    (d,e,9), (e,f,10), (b,h,11), (d,f,14)]`

Clusters: {e,a,b,d,c,i,f,g,h}

MST: { (g,h), (c,i), (f,g), (c,f), (a,b), (c,d), (a,h), (d,e) }

Kruskal's algorithm

```
T := new container for edges
L := edges sorted in non-decreasing order by weight
for each vertex v:
  v.cluster := make-cluster(v)
for each (u, v) in L:
  if u.cluster != v.cluster:
    T.add((u,v))
    merge u.cluster and v.cluster
return T
```

storing clusters

An easy way for now:

- each cluster is a linked list
- v.cluster is pointer to v's owning linked list
- $u.\text{cluster} \neq v.\text{cluster}$ is: pointer equality, $\Theta(1)$ time
- merging two clusters is merging two linked lists:
    - a lot of vertices may need their v.cluster's updated!

storing clusters

An easy way for now, continued...

Choose to always move the smaller list to the larger one:

- in the best case: smaller list has one node: 1 update
- in the worst case: smaller list has (almost) as many nodes as larger list
- in the worst case: the size of cluster roughly doubles as a result
- then how many such merges can we do?
- each v.cluster is updated at most: $\log |V|$ times

A much better way will appear later in this course.

Kruskal's algorithm: time

Let $n = |V|$ and $m = |E|$. Then:

- Collecting and sorting edges: $\Theta(m \log m)$.
- v.cluster updates: $\mathcal{O}(\log n)$ per vertex, so $\mathcal{O}(n \log n)$ total
- the rest is $\Theta(1)$ per vertex or edge

Total: $\mathcal{O}(n \log n + m \log m)$ time.

But lets look at n and m:

- maximum number of edges in a graph with n vertices: $n(n-1)/2$
- then

$$
\begin{aligned}
m &\leq n(n-1)/2 \leq n^2 \\
\therefore \log m &\leq \log(n^2) = 2 \log n \\
\therefore \log m &\in \mathcal{O}(\log n)
\end{aligned}
$$

Then total time is $\mathcal{O}((n+m) \log n)$.

Prim's algorithm: idea

Prim's algorithm finds a MST by a BFS with a twist:

- the queue is replaced with a minimum priority queue
- with an additional operation `decrease-priority(vertex, new-priority)`
  - **Exercise**: show that `decrease-priority` is $\mathcal{O}(\log n)$ where n is the size of the priority queue

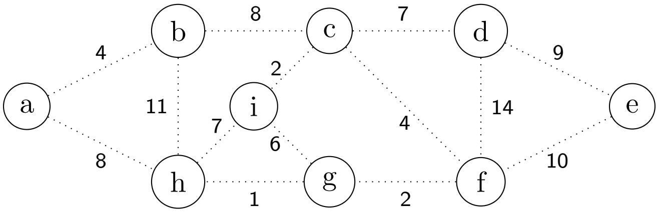
Keep unvisited vertices in the priority queue:

$$priority(v) = \text{minimum weight of any edge between } v \text{ and tree}$$
$$priority(v) = \infty \text{ if no such edge}$$

The algorithm grows a tree by one edge at a time.

Correctness idea: every time we `extract-min`, we get the cheapest edge to add to the tree.

Prim's algorithm: example

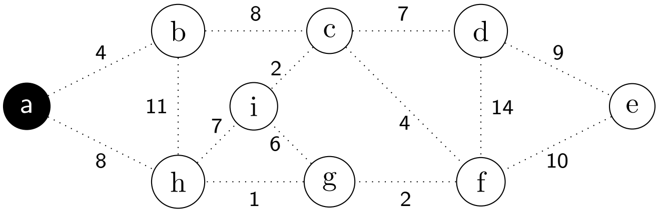


Priority queue contains vertices *not* in tree:

vertex	a	b	c	d	e	f	g	h	i
priority	0	∞	∞	∞	∞	∞	∞	∞	∞
pred									

MST: $\{\qquad\}$

Prim's algorithm: example

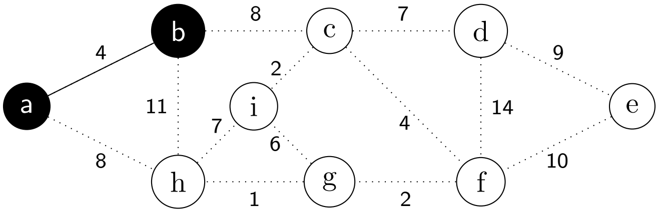


Priority queue contains vertices *not* in tree:

vertex	b	h	c	d	e	f	g	i
priority	4	8	∞	∞	∞	∞	∞	∞
pred	a	a						

MST: $\{ \qquad \}$

Prim's algorithm: example

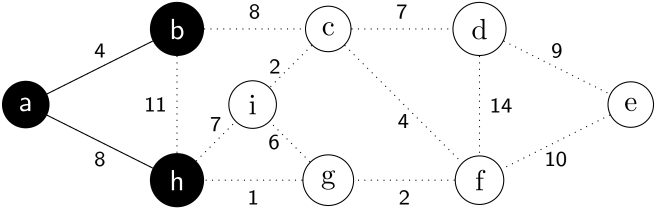


Priority queue contains vertices *not* in tree:

vertex	h	c	d	e	f	g	i
priority	8	8	∞	∞	∞	∞	∞
pred	a	b					

MST: $\{$ (a,b), $\}$

Prim's algorithm: example

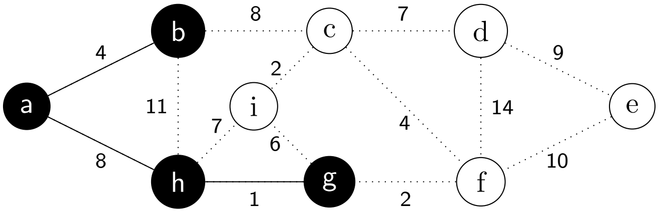


Priority queue contains vertices *not* in tree:

vertex	g	i	c	d	e	f
priority	1	7	8	∞	∞	∞
pred	h	h	b			

MST: { (a,b), (a,h), }

Prim's algorithm: example

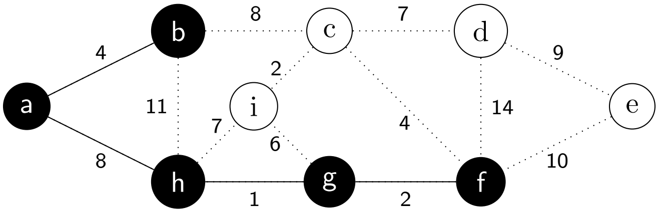


Priority queue contains vertices *not* in tree:

vertex	f	i	c	d	e
priority	2	6	8	∞	∞
pred	g	g	b		

MST: { (a,b), (a,h), (h,g), }

Prim's algorithm: example

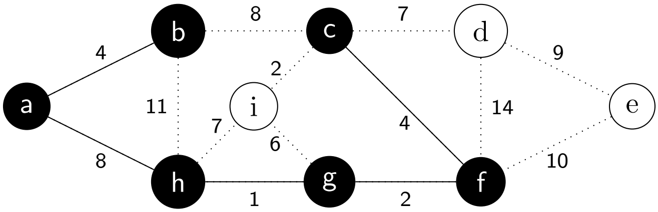


Priority queue contains vertices *not* in tree:

vertex	c	i	e	d
priority	4	6	10	14
pred	f	g	f	f

MST: { (a,b), (a,h), (h,g), (g,f), }

Prim's algorithm: example

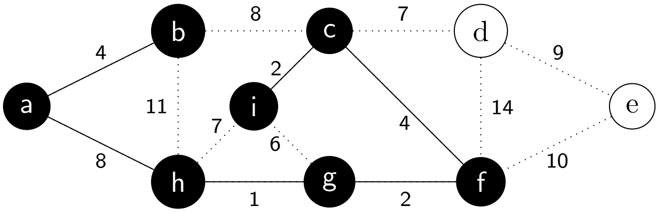


Priority queue contains vertices *not* in tree:

vertex	i	d	e
priority	2	7	10
pred	c	c	f

MST: { (a,b), (a,h), (h,g), (g,f), (c,f), }

Prim's algorithm: example

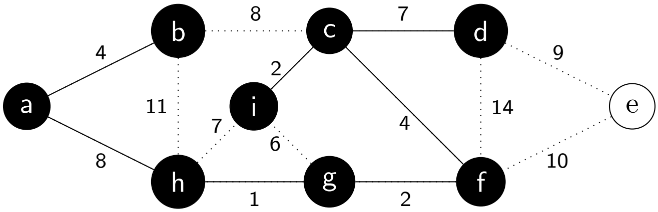


Priority queue contains vertices *not* in tree:

vertex	d	e
priority	7	10
pred	c	f

MST: { (a,b), (a,h), (h,g), (g,f), (c,f), (c,i), }

Prim's algorithm: example

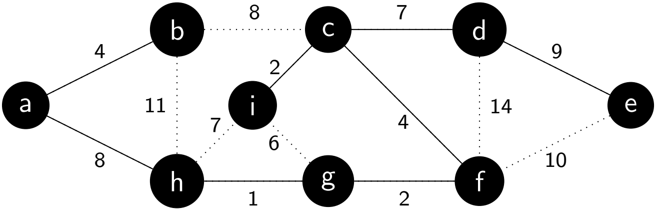


Priority queue contains vertices *not* in tree:

vertex	e
priority	9
pred	d

MST: { (a,b), (a,h), (h,g), (g,f), (c,f), (c,i), (c,d), }

Prim's algorithm: example



Priority queue contains vertices *not* in tree:

vertex	
priority	
pred	

MST: { (a,b), (a,h), (h,g), (g,f), (c,f), (c,i), (c,d), (d,e) }

Prim's algorithm

```
T := new container for edges
PQ := new min-heap()
start := pick a vertex
PQ.insert(0, start)
for each vertex v != start: PQ.insert(inf, v)
while not PQ.is-empty():
  u := PQ.extract-min()
  T.add((u.pred, u))
  for each v in u's adjacency list:
    if v in PQ and w(u, v) < priority(v):
      PQ.decrease-priority(v, w(u,v))
      v.pred := u
return T
```

Prim's algorithm: time

Let $n = |V|$ and $m = |E|$. Then:

- every vertex enters and leaves min-heap once
  - enters in the beginning only; continue until heap is empty
  - $\mathcal{O}(\log n)$ each, for a total of $\mathcal{O}(n \log n)$
- with every edge may call `decrease-priority`
  - $\mathcal{O}(\log n)$ each, for a total of $\mathcal{O}(m \log n)$
- the rest can be done in $\Theta(1)$ per vertex or per edge

Total time worst case: $\mathcal{O}((n + m) \log n)$

Kruskal's algorithm

```
T := new container for edges
L := edges sorted in non-decreasing order by weight
for each vertex v:
  v.cluster := make-cluster(v)
for each (u, v) in L:
  if u.cluster != v.cluster:
    T.add((u,v))
    merge u.cluster and v.cluster
return T
```

Kruskal's algorithm: correctness

Kruskal's algorithm maintains the loop invariants:

1. each cluster is a tree
2. $T \subseteq T_{min}$ for some MST T_{min}

Initially T is empty and clusters are single vertices, so trivially true.

Suppose (1) and (2) are true before line 4.

- on line 5, if $u.cluster \neq v.cluster$, then
- since u's cluster is a tree and v's cluster is a different tree,
- then the merged cluster (line 7) is a tree

Kruskal's algorithm: correctness

Suppose (1) and (2) are true before line 4.

- if $(u, v) \in T_{min}$, then choose $T'_{min} = T_{min}$ and done
- if $(u, v) \notin T_{min}$, then partition V into S and $V - S$ such that u's cluster $\subseteq S$, v's cluster $\subseteq V - S$, and no T edge between S and $V - S$
- in T_{min} there is a unique simple path connecting u and v
- in T_{min} there is some edge (u', v') connecting S and $V - S$
- without (u', v'), T_{min} disconnected; (u, v) would reconnect
- (u, v) is the minimum-weight edge in L connecting two clusters
- $\therefore weight(u, v) \leq weight(u', v')$
- then choose $T'_{min} = T_{min} - \{(u', v')\} + \{(u, v)\}$ is an MST

Prim's algorithm

```
T := new container for edges
PQ := new min-heap()
start := pick a vertex
PQ.insert(0, start)
for each vertex v != start: PQ.insert(inf, v)
while not PQ.is-empty():
  u := PQ.extract-min()
  T.add((u.pred, u))
  for each v in u's adjacency list:
    if v in PQ and w(u, v) < priority(v):
      PQ.decrease-priority(v, w(u,v))
      v.pred := u
return T
```

Prim's algorithm: correctness

Prim's algorithm maintains the loop invariants:

1. T contains vertices in $V - PQ$
2. for each v in PQ, $priority(v) =$ minimum weight of any edge between v and T
3. $T \subseteq T_{min}$ for some MST T_{min}

Initially T is empty, PQ contains all of V, and all priorities are ∞, so trivially true.

Suppose (1), (2), and (3) are true before line 5.

- line 6 extracts u from PQ, line 7 adds edge $(u.pred, u)$ to T, so (1)
- lines 8-11 update priorities of vertices adjacent to u, so (2)

Prim's algorithm: correctness

Suppose (1), (2), and (3) are true before line 5. Let $p = u.pred$.

- if $(p, u) \in T_{min}$, then choose $T'_{min} = T_{min}$ and done
- if $(p, u) \notin T_{min}$, then in T_{min} there is a unique simple path connecting p and u
- in T_{min} there is some edge (x, y) where x no longer in PQ and y in PQ on a path from p to u
- without (x, y), T_{min} disconnected; (p, u) would reconnect
- u was just extracted from PQ, so $weight(p, u) = priority(u) \leq priority(y) = weight(x, y)$
- then choose $T'_{min} = T_{min} - \{(x, y)\} + \{(p, u)\}$ is an MST

General Theorem

Suppose

- $T \subseteq T_{min}$
- can partition V into S and $V - S$ (cut), such that
  - no T edge between V and $V - S$
  - (u, v) is the cheapest edge (light edge) connecting V and $V - S$ (crosses the cut)

Then $T + \{(u, v)\} \subseteq T'_{min}$

- if $(u, v) \notin T_{min}$
- T_{min} has a unique simple path from u to v, via some edge (u', v') with $u' \in S$ and $v' \in V - S$
- T_{min} without (u', v') disconnected; (u, v) would would reconnect
- $weight(u, v) \leq weight(u', v')$
- Choose $T'_{min} = T_{min} - \{(u', v')\} + \{(u, v)\}$