Graph Theory and Optimization
Weighted Graphs
Shortest Paths & Spanning Trees

Nicolas Nisse

Université Côte d'Azur, Inria, CNRS, I3S, France

October 2018

Outline

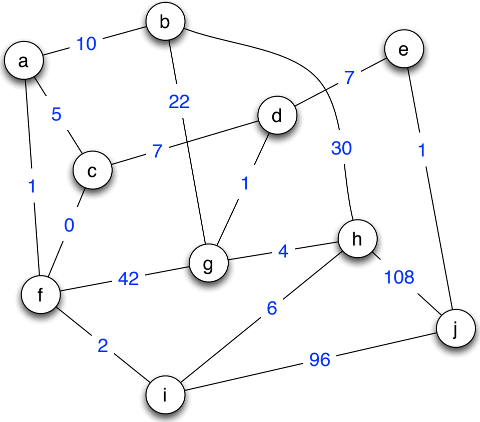
Weighted graphs (length/capacity/cost/distance)

Let $G = (V, E)$ be a graph, we can assign a weight to the edges

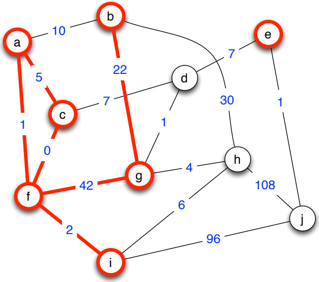
$$w : E \to \mathbb{R}^+$$

w may represent

- length
- capacity
- cost
- ...

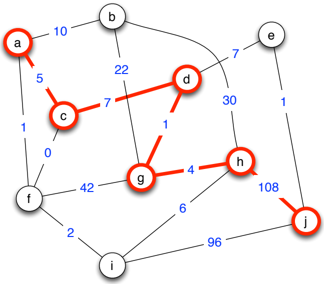


Weighted graphs (length/capacity/cost/distance)



- weigth of subgraph H: $w(H) = \sum_{e \in E(H)} w(e)$ **ex:** $w(H) = 72$

Weighted graphs (length/capacity/cost/distance)



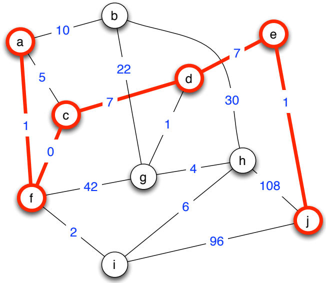
- weigth of subgraph H: $w(H) = \sum_{e \in E(H)} w(e)$
- length of path $P = (v_1, \cdots, v_\ell)$: $w(P) = \sum_{e \in E(P)^*} w(e) = \sum_{1 \leq i < \ell} w(\{v_i, v_{i+1}\})$

sum of weights of edges of P

ex: $w(P) = 125$

* a path $P = (v_1, \cdots, v_\ell)$ is seen as the subgraph $P = (\{v_1, \cdots, v_\ell\}, \{\{v_i, v_{i+1}\} \mid 1 \leq i < \ell\})$

Weighted graphs (length/capacity/cost/distance)



- weigth of subgraph H: $w(H) = \sum_{e \in E(H)} w(e)$
- length of path $P = (v_1, \cdots, v_\ell)$: $w(P) = \sum_{e \in E(P)} w(e) = \sum_{1 \leq i < \ell} w(\{v_i, v_{i+1}\})$

  sum of weights of edges of P
- distance $dist(x,y)$: minimum length of a path from $x \in V$ to $y \in V$.

  ex: $dist(a,j) = 16$

Outline

Weighted graphs: two important questions

- Computing distances and shortest paths
  - Breadth First Search (BFS) *(unweighted graph, i.e., weights= 1)*
  - Dijkstra's algorithm (1956)
  - Bellman-Ford algorithm (1958) *handle negative weights*

  Applications: GPS, routing in the Internet, basis of many algorithms...

You think it is easy?

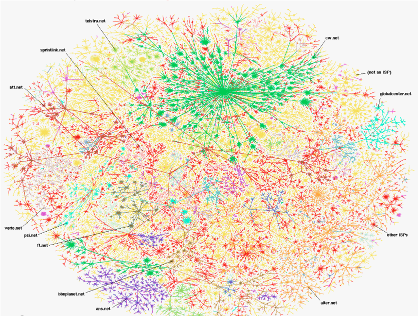
Weighted graphs: two important questions

- Computing distances and shortest paths
  - Breadth First Search (BFS) *(unweighted graph, i.e., weights* $= 1$*)*
  - Dijkstra's algorithm (1956)
  - Bellman-Ford algorithm (1958) *handle negative weights*

Applications: GPS, routing in the Internet, basis of many algorithms...

You think it is easy?

...really?.... Algorithms needed!!



[AS network in 2000, Burch, Cheswick]

Weighted graphs: two important questions

- Computing distances and shortest paths
  - Breadth First Search (BFS) *(unweighted graph, i.e., weights* $= 1$*)*
  - Dijkstra's algorithm (1956)
  - Bellman-Ford algorithm (1958) *handle negative weights*

Applications: GPS, routing in the Internet, basis of many algorithms...

- Computing minimum spanning trees

Goal: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$
Compute a spanning tree T of G with $w(T)$ minimum

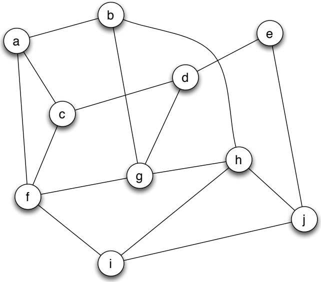
  - Borůvska (1926), Kruskal (1956), Prim (1957)

Applications:
Minimum (cheapest) substructure (subgraph) preserving connectivity.
ex: “first published by Borůvska as a method of constructing an efficient electricity network" (Wikipedia)

Outline

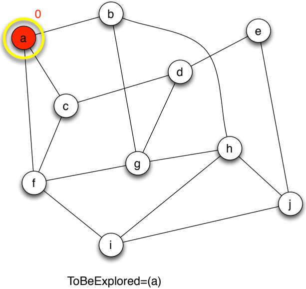
BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



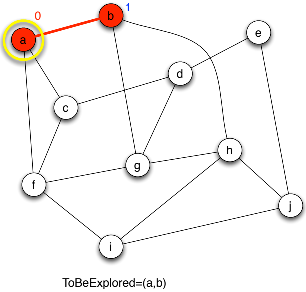
BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path P = $\#$ of edges of P = $|E(P)|$



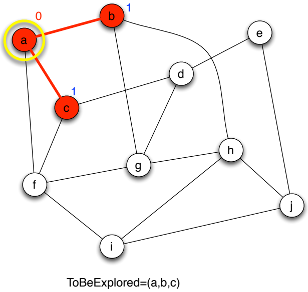
BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path P = $\#$ of edges of P = $|E(P)|$



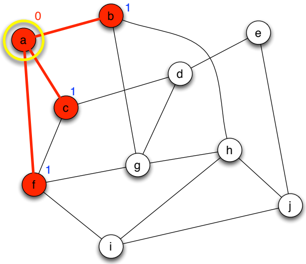
BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



BFS: Connectivity and distances in unweighted graphs

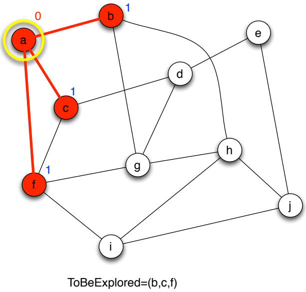
In unweighted graph, length of path P = $\#$ of edges of P = $|E(P)|$



ToBeExplored=(a,b,c,f)

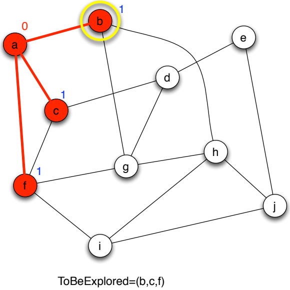
BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path P = $\#$ of edges of $P = |E(P)|$



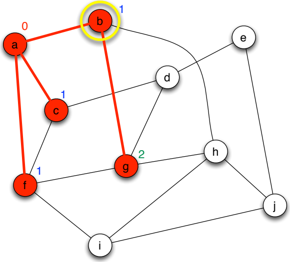
BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



BFS: Connectivity and distances in unweighted graphs

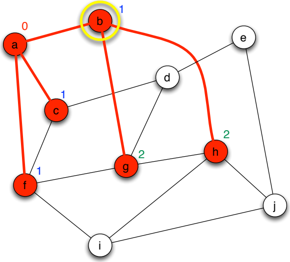
In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



ToBeExplored=(b,c,f,g)

BFS: Connectivity and distances in unweighted graphs

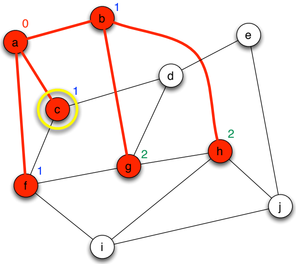
In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



ToBeExplored=(b,c,f,g,h)

BFS: Connectivity and distances in unweighted graphs

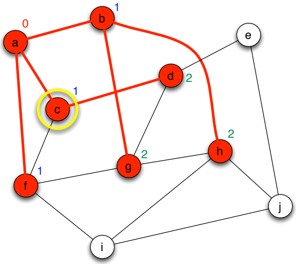
In unweighted graph, length of path P = $\#$ of edges of P = $|E(P)|$



ToBeExplored=(c,f,g,h)

BFS: Connectivity and distances in unweighted graphs

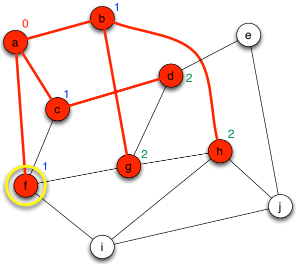
In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



ToBeExplored=(c,f,g,h,d)

BFS: Connectivity and distances in unweighted graphs

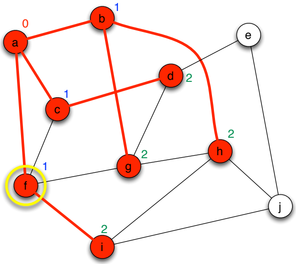
In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



ToBeExplored=(f,g,h,d)

BFS: Connectivity and distances in unweighted graphs

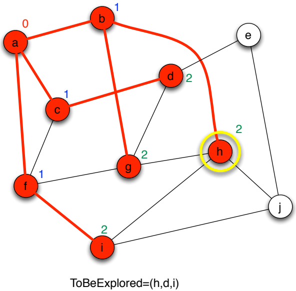
In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



ToBeExplored=(f,g,h,d,i)

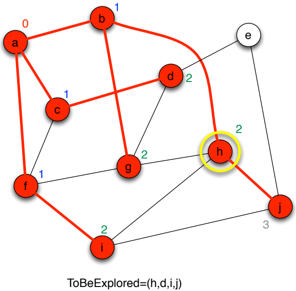
BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



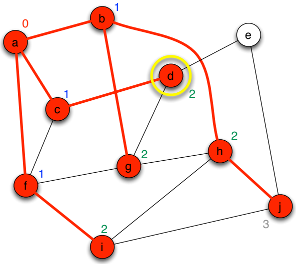
BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



BFS: Connectivity and distances in unweighted graphs

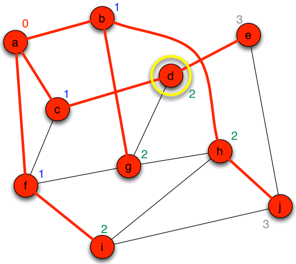
In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



ToBeExplored=(d,i,j)

BFS: Connectivity and distances in unweighted graphs

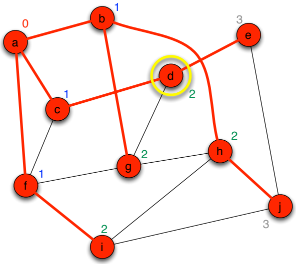
In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



ToBeExplored=(d,i,j,e)

BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path P = $\#$ of edges of P = $|E(P)|$



ToBeExplored=(d,i,j,e)

Breadth First Search

input: unweighted graph $G = (V, E)$ and $r \in V$

Initially: $d(r) = 0$, $\mathit{ToBeExplored} = (r)$
$\mathit{Done} = \emptyset$ and $T = (V(T), E(T)) = (\{r\}, \emptyset)$

While $\mathit{ToBeExplored} \neq \emptyset$ **do**

Let $v = \mathit{head}(\mathit{ToBeExplored})$

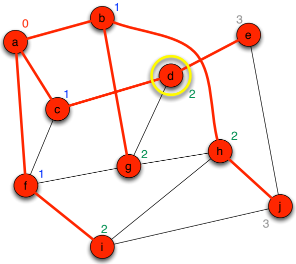
for $u \in N(v) \setminus (\mathit{ToBeExplored} \cup \mathit{Done})$ **do**

$d(u) \leftarrow d(v) + 1$
add u in $V(T)$ and $\{v, u\}$ in $E(T)$
add u at the end of $\mathit{ToBeExplored}$

remove v from $\mathit{ToBeExplored}$, add v to Done

BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



ToBeExplored=(d,i,j,e)

Breadth First Search

input: unweighted graph $G = (V, E)$ and $r \in V$

Initially: $d(r) = 0$, $ToBeExplored = (r)$
$Done = \emptyset$ and $T = (V(T), E(T)) = (\{r\}, \emptyset)$

While $ToBeExplored \neq \emptyset$ **do**

Let $v = head(ToBeExplored)$

for $u \in N(v) \setminus (ToBeExplored \cup Done)$ **do**

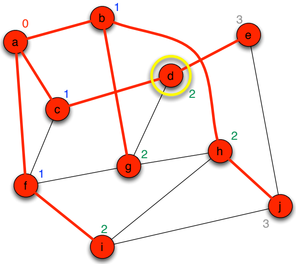
$d(u) \leftarrow d(v) + 1$
add u in $V(T)$ and $\{v, u\}$ in $E(T)$
add u at the end of $ToBeExplored$

remove v from $ToBeExplored$, add v to $Done$

Output: for any $v \in V$, $d(v) = dist(r, v)$.
T is a shortest path tree of G rooted in r: i.e., T spanning subtree of G s.t.
for any $v \in V$, the path from r to v in T is a shortest path from r to v in G.

BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path P = $\#$ of edges of P = $|E(P)|$



ToBeExplored=(d,i,j,e)

Breadth First Search

input: unweighted graph $G = (V, E)$ and $r \in V$

Initially: $d(r) = 0$, $ToBeExplored = (r)$
$Done = \emptyset$ and $T = (V(T), E(T)) = (\{r\}, \emptyset)$

While $ToBeExplored \neq \emptyset$ **do**

Let $v = head(ToBeExplored)$

for $u \in N(v) \setminus (ToBeExplored \cup Done)$ **do**

$d(u) \leftarrow d(v) + 1$
add u in $V(T)$ and $\{v, u\}$ in $E(T)$
add u at the end of *ToBeExplored*

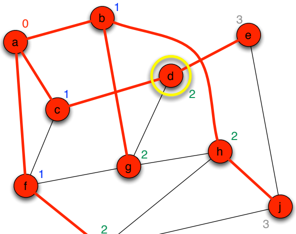
remove v from *ToBeExplored*, add v to *Done*

Time-Complexity: $\#$ operations $= O(|E|)$ *each edge is considered*

Exercise: Give an algorithm that decides if a graph is connected

BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path P = $\#$ of edges of P = $|E(P)|$



Breadth First Search

input: unweighted graph $G = (V, E)$ and $r \in V$

Initially: $d(r) = 0$, $ToBeExplored = (r)$
$Done = \emptyset$ and $T = (V(T), E(T)) = (\{r\}, \emptyset)$

While $ToBeExplored \neq \emptyset$ **do**

Let $v = head(ToBeExplored)$

for $u \in N(v) \setminus (ToBeExplored \cup Done)$ **do**

$d(u) \leftarrow d(v) + 1$
add u in $V(T)$ and $\{v, u\}$ in $E(T)$
add u at the end of $ToBeExplored$

remove v from $ToBeExplored$, add v to $Done$

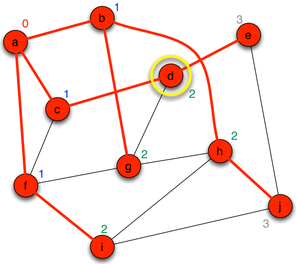
Time-Complexity: $\#$ operations $= O(|E|)$ *each edge is considered*

Rmk1: allows to decide whether G is connected

G connected iff $dist(r, v) < \infty$ defined for all $v \in V$

BFS: Connectivity and distances in unweighted graphs

In unweighted graph, length of path $P = \#$ of edges of $P = |E(P)|$



Breadth First Search

input: unweighted graph $G = (V, E)$ and $r \in V$

Initially: $d(r) = 0$, $ToBeExplored = (r)$
$Done = \emptyset$ and $T = (V(T), E(T)) = (\{r\}, \emptyset)$

While $ToBeExplored \neq \emptyset$ **do**

Let $v = head(ToBeExplored)$

for $u \in N(v) \setminus (ToBeExplored \cup Done)$ **do**

$d(u) \leftarrow d(v) + 1$
add u in $V(T)$ and $\{v, u\}$ in $E(T)$
add u at the end of *ToBeExplored*

remove v from *ToBeExplored*, add v to *Done*

Time-Complexity: $\#$ operations $= O(|E|)$ *each edge is considered*

Rmk2: gives only one shortest path tree, may be more...

depends on the ordering in which vertices are considered

BFS: Connectivity and distances in unweighted graphs

Diameter of a graph G: maximum distance between two vertices of G.

$$diam(G) = \max_{u,v \in V(G)} dist(u,v)$$

Exercise: Give an algorithm that computes the diameter of a graph.
What is the number of operations?

BFS: Connectivity and distances in unweighted graphs

Diameter of a graph G: maximum distance between two vertices of G.

$$diam(G) = \max_{u,v \in V(G)} dist(u,v)$$

Exercise: Give an algorithm that computes the diameter of a graph.
What is the number of operations?

Exercise: What does this algorithm computes??

input: unweighted tree $T = (V, E)$ and $r \in V$

1. Execute a BFS rooted in r
2. Let u be a node maximizing the distance from r
3. Execute a BFS rooted in u
4. Let w be a node maximizing the distance from u

return $dist(u, w)$

What is the number of operations?

Diameter of trees

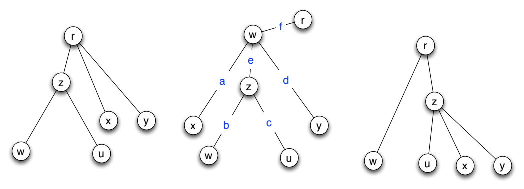
Theorem: Previous algorithm computes the diameter of T

Termination: two executions of BFS

Correctness: u is a leaf (otherwise, there would be a vertex further from r) Similarly, w is a leaf For contradiction, assume that $diam(T) = dist(x,y) > dist(u,w)$

(x and y must be leaves)

Several Cases:



As an example, consider the second one (from the left)

$f+e+c \geq \max\{f+a; f+e+b; f+d\}$ (u further from r)

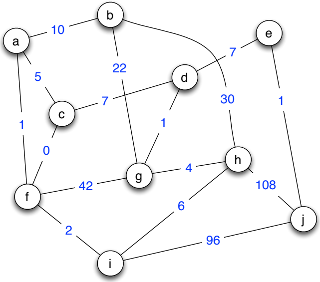
$b \geq \max\{e+a; e+f; e+d\}$ (w further from u)

So $dist(u,w) = b+c \geq a+d = dist(x,y)$, a contradiction

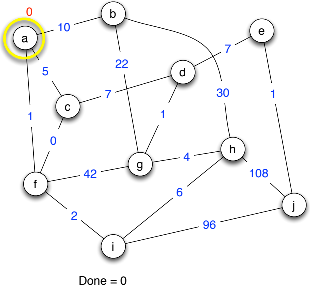
Outline

Dijkstra's algorithm (required positive weights)

BFS algorithm does not work in weighted graphs **Exercise:** Example?



Dijkstra's algorithm (required positive weights)



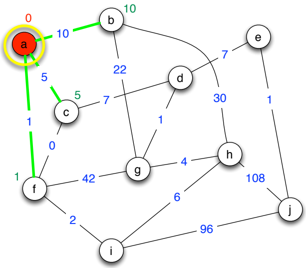
Dijkstra

input: graph $G = (V,E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r,v)$. Otherwise $dist(r,v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



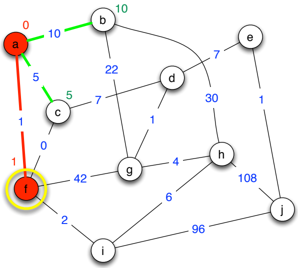
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



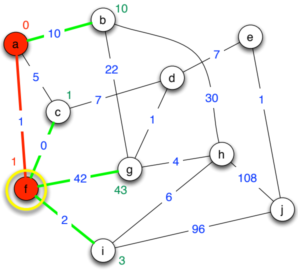
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



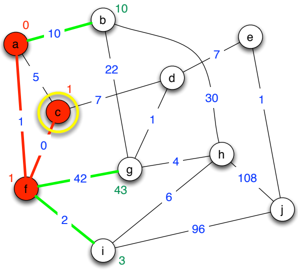
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



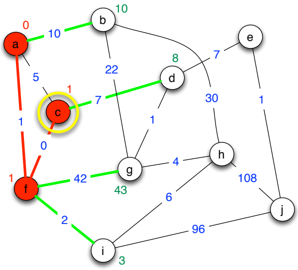
Dijkstra

input: graph $G = (V,E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}\ d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r,v)$. Otherwise $dist(r,v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



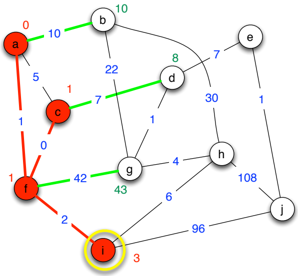
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



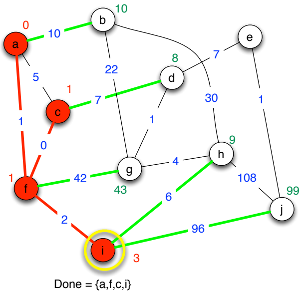
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



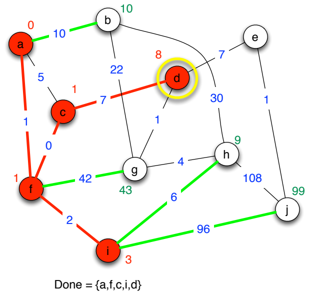
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



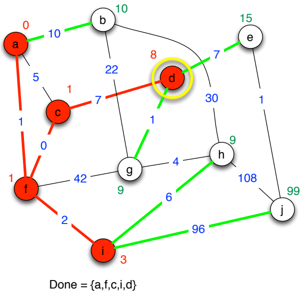
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}\ d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



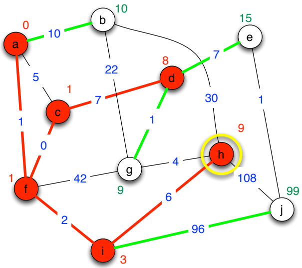
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}\ d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



Done = {a,f,c,i,d,h}

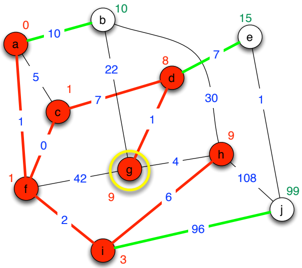
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}\ d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



Done = {a,f,c,i,d,h,g}

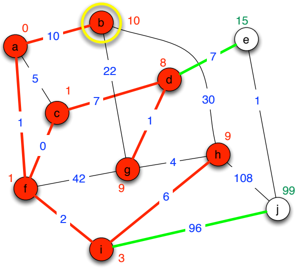
Dijkstra

input: graph $G=(V,E)$, weight w, and $r \in V$

Initially: $d(r)=0$, $(V_T,E_T)=(\emptyset,\emptyset)$, $Done=\emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v)=\infty$,

For all $v \in Done$, $d(v)=dist(r,v)$. Otherwise $dist(r,v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



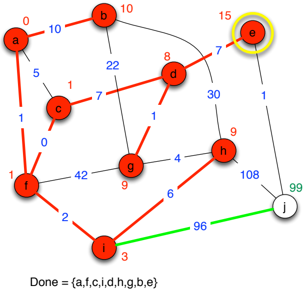
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



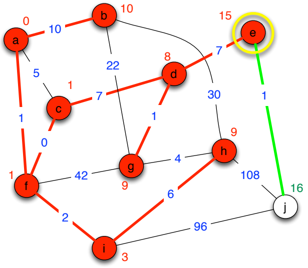
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}\ d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



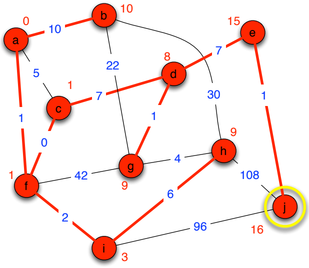
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



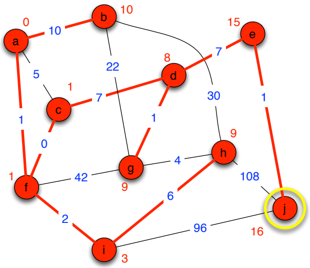
Dijkstra

input: graph $G = (V, E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$,

For all $v \in Done$, $d(v) = dist(r, v)$. Otherwise $dist(r, v) \leq d(v)$.

Dijkstra's algorithm (required positive weights)



Done = {a,f,c,i,d,h,g,b,e,j}=V

Dijkstra

input: graph $G = (V,E)$, weight w, and $r \in V$

Initially: $d(r) = 0$, $(V_T, E_T) = (\emptyset, \emptyset)$, $Done = \emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v) = \infty$, $parent(v) = \emptyset$

While $Done \neq V$ **do**

  Let $v \in V \setminus Done$ with $d(v)$ minimum *

  Add v in V_T and $\{v, parent(v)\}$ in E_T

  Add v in $Done$

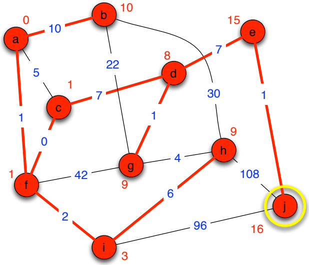
  for $u \in N(v) \setminus Done$ **do**

    if $d(u) > d(v) + w(\{u,v\})$ **then**

    $d(u) \leftarrow d(v) + w(\{u,v\})$

    $parent(u) \leftarrow v$

Dijkstra's algorithm (required positive weights)



Done = {a,f,c,i,d,h,g,b,e,j}=V

Dijkstra

input: graph $G=(V,E)$, weight w, and $r \in V$

Initially: $d(r)=0$, $(V_T,E_T)=(\emptyset,\emptyset)$, $Done=\emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v)=\infty$, $parent(v)=\emptyset$

While $Done \neq V$ **do**

    Let $v \in V \setminus Done$ with $d(v)$ minimum *

    Add v in V_T and $\{v, parent(v)\}$ in E_T

    Add v in $Done$

    for $u \in N(v) \setminus Done$ **do**

        if $d(u) > d(v) + w(\{u,v\})$ **then**

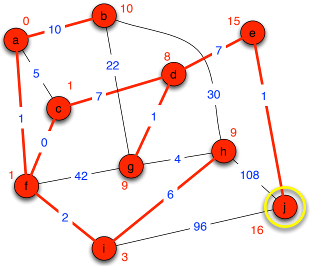
        $d(u) \leftarrow d(v) + w(\{u,v\})$

        $parent(u) \leftarrow v$

Output: $\forall v \in V$, $d(v) = dist(r,v)$, T is a shortest path tree of G rooted in r

Time-complexity: $O(|E| + |V| \log |V|)$ (requires sorting *)

Dijkstra's algorithm (required positive weights)



Done = {a,f,c,i,d,h,g,b,e,j}=V

Dijkstra

input: graph $G=(V,E)$, weight w, and $r \in V$

Initially: $d(r)=0$, $(V_T,E_T)=(\emptyset,\emptyset)$, $Done=\emptyset$, and $\forall v \in V \setminus \{r\}$ $d(v)=\infty$, $parent(v)=\emptyset$

While $Done \neq V$ **do**

Let $v \in V \setminus Done$ with $d(v)$ minimum *

Add v in V_T and $\{v, parent(v)\}$ in E_T

Add v in $Done$

for $u \in N(v) \setminus Done$ **do**

if $d(u) > d(v) + w(\{u,v\})$ **then**

$d(u) \leftarrow d(v) + w(\{u,v\})$

$parent(u) \leftarrow v$

Output: $\forall v \in V$, $d(v) = dist(r,v)$, T is a shortest path tree of G rooted in r

proof: since w positive $\Rightarrow$ a subpath of a shortest path is a a shortest path

Dijkstra's algorithm Proof of correctness

Termination: After i^{th} iteration of *while* loop, $|Done| = i$, then the algorithm terminates in $|V|$ iterations of *while* loop

Correctness: By induction on $1 \leq i < |V|$, after the i^{th} iteration of *while* loop, $|Done| = i$, and $\forall v \in Done$, $d(v) = dist(r, v)$. ok for $i = 0$

Assume the hypothesis holds ater the i^{th} iteration.
Let $v \in V \setminus Done$ be chosen at the $(i+1)^{th}$ iteration.

- By minimality of $d(v)$ (in $V \setminus Done$),

  if G connected, then $d(v) < \infty$ and $Done \cap N(v) \neq \emptyset$, and

  $d(v) = \min_{u \in Done \cap N(v)} d(u) + w(\{u, v\})$ by induction: $d(v) \geq dist(v, r)$
- For contradiction, assume that $d(v) < dist(v, r)$: there is a shortest path $P = (r, \cdots, x, v)$ of length $< d$.
  - $x \in Done$: otherwise it would contradict minimality of $d(v)$
  - $dist(v, r) = dist(x, r) + w(\{x, v\}) = d(x) + w(\{x, v\}) < d(v) = \min_{u \in Done \cap N(v)} d(u) + w(\{u, v\}) \leq d(x) + w(\{x, v\})$

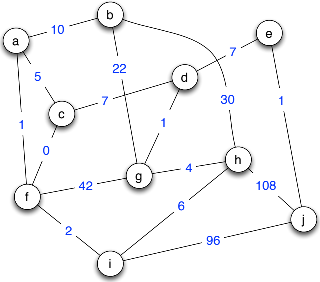
a contradiction

Outline

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \rightarrow \mathbb{R}$

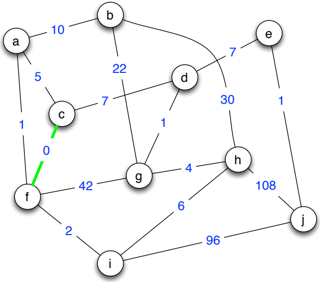
Compute a spanning tree T of G with $w(T)$ minimum



Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

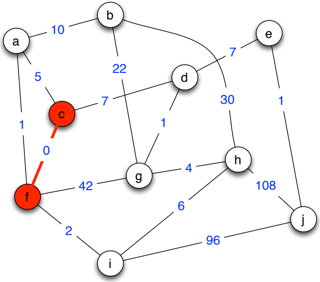
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

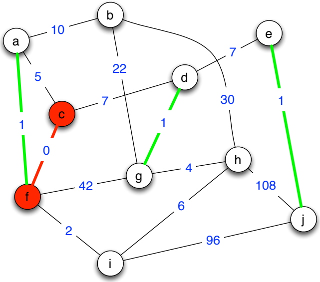
Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$
Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

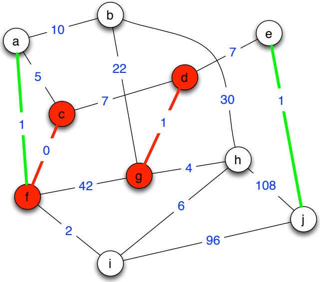
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

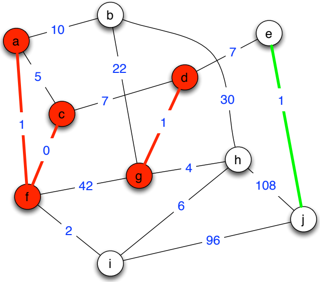
Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V,E)$ with weight $w : E \to \mathbb{R}$
Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V,E)$, weight w

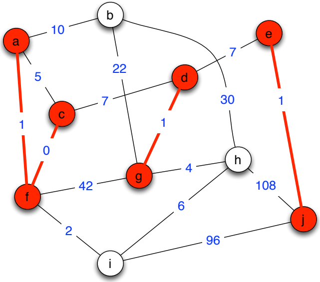
Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G=(V,E)$ with weight $w: E \to \mathbb{R}$
Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G=(V,E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T=(\emptyset,\emptyset)$

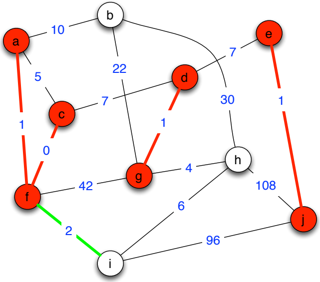
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

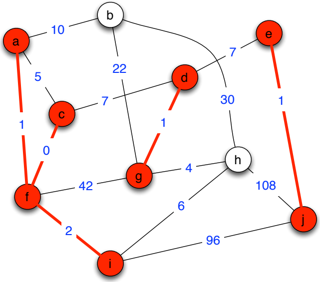
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \rightarrow \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

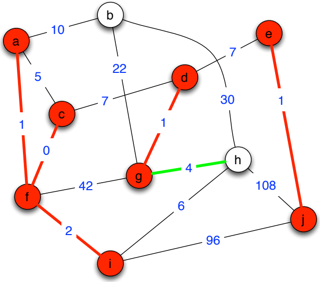
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \rightarrow \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

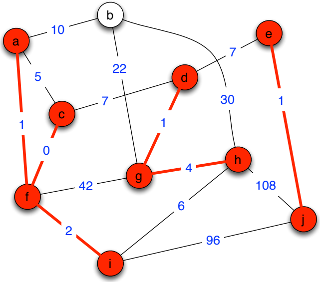
Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$
Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

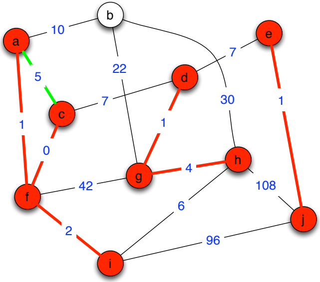
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

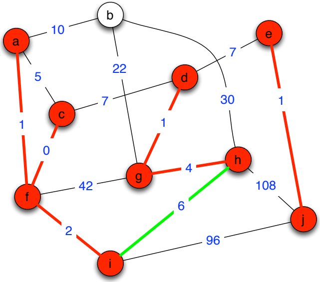
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

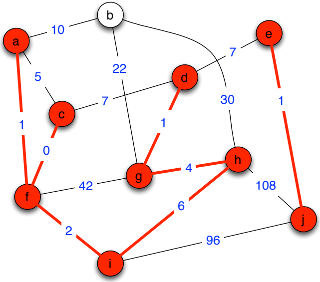
Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$
Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

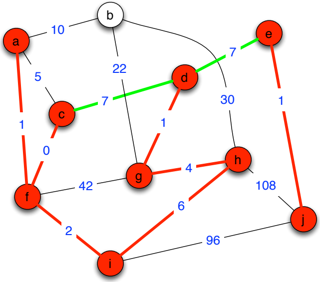
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

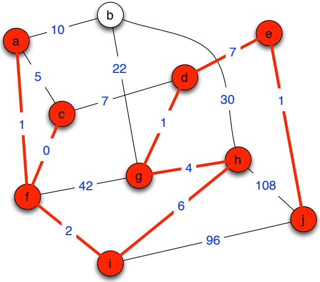
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

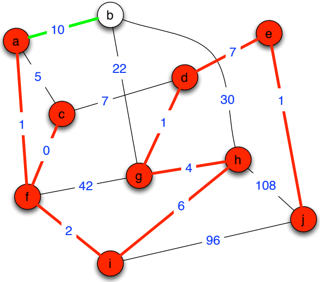
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G=(V,E)$ with weight $w:E\rightarrow\mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G=(V,E)$, weight w

Initially: Let $(e_1,\cdots,e_m)$ be an ordering of E in non decreasing ordering of w, and $T=(\emptyset,\emptyset)$

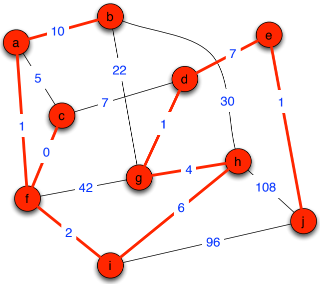
For $i\leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

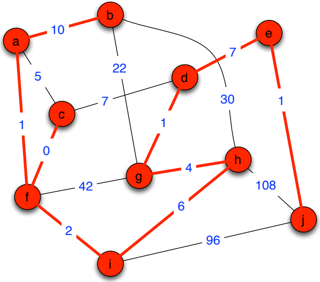
For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \rightarrow \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

For $i \leq m$ **do**

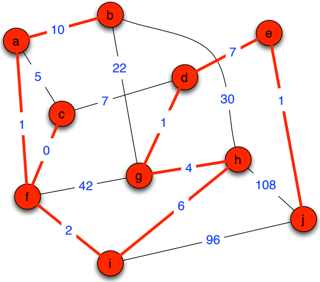
Add e_i in T if it does not create a cycle.

Time-Complexity: $\#$ operations $= O(|E| \log |E|)$ *sorting*

Kruskal's algorithm Minimum Spanning Tree

Reminder: given $G = (V, E)$ with weight $w : E \to \mathbb{R}$

Compute a spanning tree T of G with $w(T)$ minimum



Kruskal

input: connected graph $G = (V, E)$, weight w

Initially: Let $(e_1, \cdots, e_m)$ be an ordering of E in non decreasing ordering of w, and $T = (\emptyset, \emptyset)$

For $i \leq m$ **do**

Add e_i in T if it does not create a cycle.

Exercise: Prove that, T returned by the Alg. is a minimum spanning tree

Idea of proof: by contradiction

Kruskal's algorithm Proof of correctness

Terminaison: obvious

Correctness:(Sketch) Clearly, T is a spanning tree (it is acyclic by definition, and if it is not connected, some edges connecting the components should have been added)

Assume it is not minimum and let $(e_1, \cdots, e_{n-1})$ be its edges in non decreasing ordering of their weights.

Among the min. spanning tree of G, let T^* with edges $(f_1, \cdots, f_{n-1})$ such that the minimum index i with $e_i \neq f_i$ is maximized.

$T^* \cup e_i$ contains a cycle C and, there is $j > i$ such that $f_j \in E(C) \setminus E(T)$ and $w(f_j) \leq w(e_i)$ (otw, T^* is not minimum).

- if $w(f_j) < w(e_i)$ then the algorithm should have chosen f_j instead of e_i
- if $w(f_j) = w(e_i)$, T' obtained from $T^*$by replacing f_j by e_i is a minimum spanning tree, contradicting the maximality of i.

Summary: To be remembered

- weighted graph, distances
- Deciding connectivity
  Shortest path tree in undirected graph $O(|E|)$, *BFS*
- Computing Shortest path tree $O(|E| + |V| \log |V|)$, *Dijkstra*
- Computing Min. spanning tree $O(|E| \log |E|)$, *Kruskal*