L17

4102

3.22.2016

abhi shelat
what is a graph cut?

what does it mean for a set A to respect a cut S?

what does the cut theorem say?
we want:

looking for a set of edges that  \( T \subseteq E \)

(a) connects all vertices
(b) has the least cost  \( \min \sum_{(u,v) \in T} w(u,v) \)
minimum spanning tree

looking for a set of edges that \( T \subseteq E \)
(a) connects all vertices
(b) has the least cost \( \min \sum_{(u, v) \in T} w(u, v) \)
looking for a set of edges that \( T \subseteq E \)
(a) connects all vertices
(b) has the least cost \( \min \sum_{(u,v) \in T} w(u,v) \)

how many edges does solution have? \( v - 1 \)

does solution have a cycle? No cycle
strategy

start with an empty set of edges $A$

repeat for $v-1$ times:

add lightest edge that does not create a cycle
Kruskal’s algorithm
T ← ∅
repeat V − 1 times:
add to T the lightest edge $e \in E$ that does not create a cycle

why does this work?
Cut is a partition of graph \( G = (V, E) \) into two sets of vertices \((S, V-S)\).
example of a cut
definition: crossing a cut

If \((S, V-S)\) is a cut, then edge \(e = (u, v)\) crosses the cut if 

\(u \in S\) and \(v \in V-S\).
definition: crossing a cut

an edge $e = (u, v)$ crosses a graph cut $(S, V - S)$ if

$$u \in S \quad v \in V - S$$
example of a crossing
definition: respect

A set $B$ of edges respects cut $(S, V-S)$ if

$\forall e \in B$, $e$ does not cross $(S, V-S)$. 

Cut theorem
Cut theorem

Suppose the set of edges $A$ is part of an m.s.t.
Let $(S, V - S)$ be any cut that respects $A$.
Let edge $e$ be the min-weight edge across $(S, V - S)$.

Then: $A \cup \{e\}$ is part of an m.s.t.
example of theorem
Theorem 2 Suppose the set of edges $A$ is part of a minimum spanning tree of $G = (V, E)$. Let $(S, V - S)$ be any cut that respects $A$ and let $e$ be the edge with the minimum weight that crosses $(S, V - S)$. Then the set $A \cup \{e\}$ is part of a minimum spanning tree.
proof of cut thm
Kruskal-pseudocode($G$)

1. $A \leftarrow \emptyset$
2. repeat $V - 1$ times:
3. add to $A$ the lightest edge $e \in E$ that does not create a cycle
1 Minimum Spanning Tree Algorithm

Kruskal-pseudocode \( G \)

1 \( A \leftarrow \emptyset \)

2 repeat \( V - 1 \) times:

3 add to \( A \) the lightest edge \( e \in E \) that does not create a cycle

Proof: by induction. In step 1, \( A \) is part of some MST. Suppose that after \( k \) steps, \( A \) is part of some MST (line 2). In line 3, we add an edge \( e = (u,v) \).
3 cases for edge e.
Case 1: $e=(u,v)$ and both $u,v$ are in $A$. 
3 cases for edge $e$.
Case 2: $e=(u,v)$ and only $u$ is in $A$. 

\[\begin{align*}
\text{(Diagram showing two sets with an edge between them.)}
\end{align*}\]
3 cases for edge e.
Case 3: $e=(u,v)$ and neither $u$ nor $v$ are in $A$. 
Kruskal-pseudocode

1. \( A \leftarrow \emptyset \)
2. repeat \( V - 1 \) times:
   3. add to \( A \) the lightest edge \( e \in E \) that does not create a cycle

Theorem 2
Suppose the set of edges \( A \) is part of a minimum spanning tree of \( G = (V,E) \). Let \( w(S, V-S) \) be any cut that respects \( A \) and let \( e \) be the edge with the minimum weight that crosses \( w(S, V-S) \). Then the set \( A \cup \{e\} \) is part of a minimum spanning tree.

Proof.
By assumption, \( A \subseteq T \) for some minimum spanning tree \( T \) of \( G \).

Case 1
If \( A \cup \{e\} \subseteq T \), then the theorem is true already.

Case 2
Suppose \( A \cup \{e\} \nsubseteq T \). Let \( e = w(u, v) \). We shall construct a new tree \( T' \) that contains \( A \cup \{e\} \) by changing only a few edges of \( T \). First, draw a picture of the situation:

Now consider adding edge \( e \) to \( T \). This creates a cycle from \( u \) to \( v \) to \( u \). Why?

Let \( e = w(u, v) \) be the edge on this cycle that crosses \( w(S, V-S) \). Why must such an edge \( e \) exist?

Let \( T' = T \cup \{e\} \). Since \( T \) has \( V-1 \) edges and since \( T \) is connected, then \( T' \) is also a spanning tree. Now we shall argue that \( T' \) is also a minimum spanning tree. This follows because:

Since \( T \) is a minimum spanning tree, this means that the relation must be equality, and therefore \( T' \) is also a minimum spanning tree.
\begin{enumerate}
\item $A \leftarrow \emptyset$
\item repeat $V - 1$ times:
\item Pick a cut $(S, V - S)$ that respects $A$
\item Let $e$ be min-weight edge over cut $(S, V - S)$
\item $A \leftarrow A \cup \{e\}$
\end{enumerate}

$A$ be a tree. and $S = A$. 

\text{GENERAL-MST-STRATEGY}(G = (V, E))
Prim’s algorithm

**General-MST-Strategy** \((G = (V, E))\)

1. \(A \leftarrow \emptyset\)
2. repeat \(V - 1\) times:
   3. Pick a cut \((S, V - S)\) that respects \(A\)
   4. Let \(e\) be min-weight edge over cut \((S, V - S)\)
   5. \(A \leftarrow A \cup \{e\}\)

A is a subtree

edge \(e\) is lightest edge that grows the subtree
prim
prim
prim

Diagram of a graph with vertices labeled a, b, c, d, e, f, g, h, i.

- Edge weights:
  - a to b: 9
  - b to d: 8
  - d to g: 8
  - g to i: 2
  - i to c: 11
  - c to f: 6
  - f to h: 6
  - h to c: 1
  - c to e: 3
  - e to d: 7
  - a to d: 10
  - b to c: 12
  - e to i: 9
  - g to h: 9

- The graph is connected with multiple paths between vertices.
idea: Keep a data structure which identifies the "lightest edge that costs \((A=5, V=5)\)" — priority queue.
implementation
new data structure $(node, key)$. integer

- **makequeue**: insert a list of $(n_1, k_1), (n_2, k_2), \ldots, (n_k, k_k)$ into $Q$.

- **insert**

- **ExtractMin operation**: removes the node $(n_i, k_i)$ where $k_i$ is the smallest key in $Q$.

- **decrease key operation**: given a key $(n_i, k_i)$, decrease the value of $k_i$ to $k_i^*$. 
binary heap

1. full tree, key value <= to key of children

2. 
binary heap

full tree, key value <= to key of children
binary heap

full tree, key value <= to key of children
binary heap

full tree, key value <= to key of children
binary heap
full tree, key value \( \leq \) to key of children

how to extract min?

\[
\begin{array}{c}
8 \\
/ \quad \quad \\
11 \\
/ \\
23
\end{array}
\begin{array}{c}
10 \\
/ \\
13
\end{array}
\begin{array}{c}
11 \\
/ \\
44
\end{array}
\begin{array}{c}
5 \\
/ \\
6 \\
/ \\
9
\end{array}
\begin{array}{c}
13 \\
/
\end{array}
\]

\( \text{insert: time } \Theta(\log n) \) where \( n \) is the size of the heap.
binary heap
full tree, key value <= to key of children

how to extract min?
binary heap

As I want to decrease key 10 to 7!!
binary heap
full tree, key value <= to key of children

how to extract min?
how to decrease key?

$\Theta(\log n)$ time
binary heap

full tree, key value \leq \text{ to } key of children

how to extract min?
how to decrease key?

\( O(\log n) \) time

node 3

parent * node
left, right * node

\( 3 \)
implementation

use a priority queue to keep track of light edges

insert: $O(n)$
makequeue: $O(n)$
extractmin: $O(\log n)$
decreasekey: $O(\log n)$
Prim’s algorithm

Initialize each node w/ kv = ∞
Pick some node v₀ and set kv₀ = 0
Q ← Make Queue [ all vertices ]

while Q is not empty
    u ← extractmin ( Q )
    for all neighbors u of v₀
        if u ∈ Q and kv ≥ w(u, v₀)
            Decrease Key ( Q, u, w(u, v₀) )
PRIM($G = (V, E)$)

1. $Q \leftarrow \emptyset$  ▶ $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. while $Q \neq \emptyset$
   6. do $u \leftarrow \text{EXTRACT-MIN}(Q)$
      7. for each $v \in \text{Adj}(u)$
         8. do if $v \in Q$ and $w(u, v) < k_v$
            9. then $\pi_v \leftarrow u$
               10. $\text{DECREASE-KEY}(Q, v, w(u, v))$  ▶ Sets $k_v \leftarrow w(u, v)$
PRIM($G = (V, E)$)

1. $Q \leftarrow \emptyset$  
   $\triangleright$ $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$

5. while $Q \neq \emptyset$
   6. $u \leftarrow \text{EXTRACT-MIN}(Q)$
   7. for each $v \in Adj(u)$
      8. do if $v \in Q$ and $w(u, v) < k_v$
         9. then $\pi_v \leftarrow u$

10. $\text{DECREASE-KEY}(Q, v, w(u, v))$  
    $\triangleright$ Sets $k_v \leftarrow w(u, v)$
**PRIM**($G = (V, E)$)

1. $Q \leftarrow \emptyset$ $\triangleright$ $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. while $Q \neq \emptyset$
6.     do $u \leftarrow \text{EXTRACT-MIN}(Q)$
7.      for each $v \in Adj(u)$
8.      do if $v \in Q$ and $w(u, v) < k_v$
9.      then $\pi_v \leftarrow u$
10.     $\text{DECREASE-KEY}(Q, v, w(u, v))$ $\triangleright$ Sets $k_v \leftarrow w(u, v)$
PRIM(G = (V, E))
1. \( Q \leftarrow \emptyset \quad \triangleright \quad Q \) is a Priority Queue
2. Initialize each \( v \in V \) with key \( k_v \leftarrow \infty, \pi_v \leftarrow \text{NIL} \)
3. Pick a starting node \( r \) and set \( k_r \leftarrow 0 \)
4. Insert all nodes into \( Q \) with key \( k_v \).
5. while \( Q \neq \emptyset \)
6. do \( u \leftarrow \text{EXTRACT-MIN}(Q) \)
7. for each \( v \in \text{Adj}(u) \)
8. do if \( v \in Q \) and \( w(u,v) < k_v \)
9. then \( \pi_v \leftarrow u \)
10. \( \text{DECREASE-KEY}(Q,v,w(u,v)) \quad \triangleright \quad \text{Sets } k_v \leftarrow w(u,v) \)
PRIM($G = (V,E)$)

1. $Q \leftarrow \emptyset$ \hspace{1em} $\triangleright$ $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. While $Q \neq \emptyset$
   
   6. $u \leftarrow \text{EXTRACT-MIN}(Q)$
   7. For each $v \in Adj(u)$
      
      8. Do if $v \in Q$ and $w(u,v) < k_v$
         
         9. Then $\pi_v \leftarrow u$
         
         10. $\text{DECREASE-KEY}(Q, v, w(u,v))$ \hspace{1em} $\triangleright$ Sets $k_v \leftarrow w(u,v)$
PRIM($G = (V, E)$)
1. $Q \leftarrow \emptyset$  \quad \triangleright \quad Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. \textbf{while} $Q \neq \emptyset$
6. \hspace{1em} do $u \leftarrow \text{EXTRACT-MIN}(Q)$
7. \hspace{2em} for each $v \in Adj(u)$
8. \hspace{3em} do if $v \in Q$ and $w(u, v) < k_v$
9. \hspace{4em} then $\pi_v \leftarrow u$
10. \hspace{3em} $\text{DECREASE-KEY}(Q, v, w(u, v))$  \quad \triangleright \quad Sets $k_v \leftarrow w(u, v)$
\text{prim} (G = (V, E))

1. \( Q \leftarrow \emptyset \) \quad \triangleright \quad Q \text{ is a Priority Queue}
2. Initialize each \( v \in V \) with key \( k_v \leftarrow \infty \), \( \pi_v \leftarrow \text{NIL} \)
3. Pick a starting node \( r \) and set \( k_r \leftarrow 0 \)
4. Insert all nodes into \( Q \) with key \( k_v \).
5. while \( Q \neq \emptyset \)
   \hspace{1em} do \( u \leftarrow \text{EXTRACT-MIN}(Q) \)
   \hspace{1em} for each \( v \in \text{Adj}(u) \)
   \hspace{2em} do if \( v \in Q \) and \( w(u, v) < k_v \)
   \hspace{3em} then \( \pi_v \leftarrow u \)
   \hspace{2em} \text{DECREASE-KEY}(Q, v, w(u, v)) \quad \triangleright \text{Sets} \ k_v \leftarrow w(u, v)

\# of calls to decrease key is \( \leq E \).
PRIM($G = (V, E)$)

1. $Q \leftarrow \emptyset$  \hspace{1cm} $\triangleright$ $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. while $Q \neq \emptyset$
   \hspace{1cm} do $u \leftarrow$ EXTRACT-MIN($Q$)
   \hspace{2cm} for each $v \in$ Adj($u$)
   \hspace{3cm} do if $v \in Q$ and $w(u, v) < k_v$
   \hspace{4cm} then $\pi_v \leftarrow u$
   \hspace{5cm} DECREASE-KEY($Q, v, w(u, v)$)  \hspace{1cm} $\triangleright$ Sets $k_v \leftarrow w(u, v)$

This work is called at most $E \times \log U$ times over entire execution.

$\Theta(E \log U + U \log U)$
PRIM($G = (V,E)$)

1. $Q \leftarrow \emptyset$ \hspace{1em} $\triangleright$ $Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow$ NIL
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. While $Q \neq \emptyset$
   
   - do $u \leftarrow$ EXTRACT-MIN($Q$)
   
   for each $v \in Adj(u)$
     
     - do if $v \in Q$ and $w(u,v) < k_v$
       
       - then $\pi_v \leftarrow u$
       
       - DECREASE-KEY($Q, v, w(u,v)$) \hspace{1em} $\triangleright$ Sets $k_v \leftarrow w(u,v)$

$O(V \log V + E \log V) = O(E \log V)$

$E \gg V$
use a priority queue to keep track of light edges

- **priority queue**
  - insert: $O(\log n)$
  - makequeue: $n$
  - extractmin: $O(\log n)$
  - decreasekey: $O(\log n)$

- **fibonacci heap**
  - log $n$ amortized
  - n amortized
  - log $n$ amortized
  - O(1) amortized
A $k$th order binomial tree, which I'll abbreviate $B_k$, is defined recursively. $B_0$ is a single node. For all $k > 0$, $B_k$ consists of two copies of $B_{k-1}$ that have been linked together, meaning that the root of one $B_{k-1}$ has become a new child of the other root.

Binomial trees have several useful properties, which are easy to prove by induction:

- The root of $B_k$ has degree $k$.
- The children of the root of $B_k$ are the roots of $B_0, B_1, ..., B_{k-1}$.
- $B_k$ has height $k$.
- $B_k$ has $2^k$ nodes.
- $B_k$ can be obtained from $B_{k-1}$ by adding a new child to every node.
- $B_k$ has $\binom{k}{d} \times n$ nodes at depth $d$, for all $0 \leq d < k$.
- $B_k$ has $2^k h_1$ nodes with height $h$, for all $0 < h < k$, and one node with the root with height $k$.

Although we normally don't care in this class about the low-level details of data structures, we need to be specific about how Fibonacci heaps are actually implemented, so that we can be sure that certain operations can be performed quickly. Every node in a Fibonacci heap points to four other nodes: its parent, its 'next' sibling, its 'previous' sibling, and one of its children. The sibling pointers are used to join the roots together into a circular doubly-linked root list. In each binomial tree, the children of each node are also joined into a circular doubly-linked list using the sibling pointers.

With this representation, we can add or remove nodes from the root list, merge two root lists together, link one two binomial tree to another, or merge a node's list of children with the root list, in constant time, and we can visit every node in the root list in constant time per node. Having established that these primitive operations can be performed quickly, we never again need to think about the low-level representation details.
each node has 4 pointers

2 fields:
  degree
  marked

$D(n)$
faster implementation

PRIM\((G = (V, E))\)

1. \(Q \leftarrow \emptyset\)  \(\triangleright\) \(Q\) is a Priority Queue
2. Initialize each \(v \in V\) with key \(k_v \leftarrow \infty, \pi_v \leftarrow \text{NIL}\)
3. Pick a starting node \(r\) and set \(k_r \leftarrow 0\)
4. Insert all nodes into \(Q\) with key \(k_v\).
5. \textbf{while} \(Q \neq \emptyset\)
6. \hspace{1em} \textbf{do} \(u \leftarrow \text{EXTRACT-MIN}(Q)\)
7. \hspace{2em} \textbf{for} each \(v \in \text{Adj}(u)\)
8. \hspace{3em} \textbf{do if} \(v \in Q\) and \(w(u, v) < k_v\)
9. \hspace{4em} \textbf{then} \(\pi_v \leftarrow u\)
10. \hspace{2em} \text{DECREASE-KEY}(Q, v, w(u, v)) \  \triangleright\ \text{Sets} \(k_v \leftarrow w(u, v)\)

\[O(E + V \log V)\]
Research in mst

FREDMAN-TARJAN 84:
GABOW-GALIL-SPENCER-TARJAN 86:
CHAZELLE 97
CHAZELLE 00
PETTIE-RAMACHANDRAN 02:
KARGER-KLEIN-TARJAN 95:
(randomized)

Euclidean mst:

\[ E + V \log V \]
\[ E \log(\log^* V) \]
\[ E\alpha(V) \log \alpha(V) \]
\[ E\alpha(V) \]
\[ \Theta \text{ (optimal)} \]
\[ E \]
\[ V \log V \]
Ackerman function

\[ A(m, n) = \begin{cases} 
  n + 1 & m = 0 \\
  A(m - 1, 1) & m > 0, n = 0 \\
  A(m - 1, A(m, n - 1)) & m, n > 0
\end{cases} \]

\[ A(4, 2) = \]
inverse ackerman

\[ \alpha(n) = \]
application of mst
application of mst
application of mst
what is the length of the path from a to e?
shortest path property

definition:
\[ \delta(s, v) \]
shortest paths
algorithm
Dijkstra\texttt{(G = (V, E), s)}

1    \textbf{for all} \textit{v} \in V
2        \textbf{do} \textit{d}_{u} \leftarrow \infty
3        \pi_{u} \leftarrow \text{NIL}
4    \textit{d}_{s} \leftarrow 0
5 \textbf{Q} \leftarrow \text{MAKEQUEUE}(V) \quad \triangleright \text{use } \textit{d}_{u} \text{ as key}
6    \textbf{while } \textit{Q} \neq \emptyset
7        \textbf{do} \textit{u} \leftarrow \text{EXTRACTMIN}(\textit{Q})
8        \textbf{for each} \textit{v} \in \text{Adj}(\textit{u})
9            \textbf{do if } \textit{d}_{v} > \textit{d}_{u} + w(\textit{u}, \textit{v})
10               \textbf{then } \textit{d}_{v} \leftarrow \textit{d}_{u} + w(\textit{u}, \textit{v})
11               \pi_{v} \leftarrow \textit{u}
12        \textbf{DECREASEKEY}(\textit{Q}, \textit{v})

\textbf{Theorem 4} \quad \text{Given any weighted, directed graph } \texttt{G = (V, E)} \text{ with non-negative weights and source } \textit{s}, \texttt{dijkstra}(\texttt{G, s}) \text{ terminates with } \textit{d}_{u} = (\textit{s, v}) \text{ for all } \textit{v} \in V.
**Dijkstra**$(G = (V, E), s)$

1. for all $v \in V$
2.   do $d_u \leftarrow \infty$
3.   $\pi_u \leftarrow \text{NIL}$
4. $d_s \leftarrow 0$
5. $Q \leftarrow \text{MAKEQUEUE}(V) \triangleright$ use $d_u$ as key
6. while $Q \neq \emptyset$
7.   do $u \leftarrow \text{EXTRACTMIN}(Q)$
8.   for each $v \in Adj(u)$
9.     do if $d_v > d_u + w(u, v)$
10.    then $d_v \leftarrow d_u + w(u, v)$
11.   $\pi_v \leftarrow u$
12. $\text{DECREASEKEY}(Q, v)$

**Prim**$(G = (V, E))$

1. $Q \leftarrow \emptyset \triangleright Q$ is a Priority Queue
2. Initialize each $v \in V$ with key $k_v \leftarrow \infty$, $\pi_v \leftarrow \text{NIL}$
3. Pick a starting node $r$ and set $k_r \leftarrow 0$
4. Insert all nodes into $Q$ with key $k_v$.
5. while $Q \neq \emptyset$
6.   do $u \leftarrow \text{EXTRACT-MIN}(Q)$
7.   for each $v \in Adj(u)$
8.     do if $v \in Q$ and $w(u, v) < k_v$
9.        then $\pi_v \leftarrow u$
10.    $\text{DECREASE-KEY}(Q, w(u, v)) \triangleright$ Sets $k_v \leftarrow w(\cdot)$

**Theorem 4**
Given any weighted, directed graph $G = (V, E)$ with non-negative weights and source $s$, $\text{dijkstra}(G, s)$ terminates with $d_u = (s, v)$ for all $v \in V$. 

Theorem 4
Given any weighted, directed graph \( G = (V, E) \) with non-negative weights and source \( s \), \( \text{dijkstra}(G, s) \) terminates with \( d_u = (s, v) \) for all \( v \in V \).

The running time is $O(m \log n)$.
why does dijkstra work?

triangle inequality:

\[ \forall (u, v) \in E, \ \delta(s, v) \leq \delta(s, u) + w(u, v) \]

upper bound:

\[ d_v \geq \delta(s, v) \]
breadth first search

input: \( G = (V, E), s \)

output:
breadth first search

input: \( G = (V, E), s \)
output: \( \forall v \in V \quad d_v = \delta(s, v) \)

smallest # of edges from s to v
breadth-first search
breadth-first search
breadth-first search
breadth first search

input: \( G = (V, E), s \)
output: smallest # of edges from s to \( \forall u \in V \)
bfs(G, a)
bfs(G, a)
bfs(G, a)
bfs(G, a)
bfs(G, a)
bfs(G, a)
bfs(G, a)
bfs(G, a)
bfs(G, a)
breadth first search

BFS(V, E, s)
for each u ∈ V − {s}
    do d[u] ← ∞

 d[s] ← 0
Q ← ∅
ENQUEUE(Q, s)
while Q ̸= ∅
    do u ← DEQUEUE(Q)
        for each v ∈ Adj[u]
            do if d[v] = ∞
                then d[v] ← d[u] + 1
                ENQUEUE(Q, v)
bfs theorem