Chapter 22
Elementary Graph Algorithms
Part I

Chi-Yeh Chen

陳奇業

成功大學資訊工程學系

Graph representation

NCKU
National Cheng Kung University

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

Graph representation

Given graph $G = (V, E)$.

- May be either directed or undirected.
- Two common ways to represent for algorithms:
  1. Adjacency lists.
  2. Adjacency matrix.

When expressing the running time of an algorithm, it's often in terms of both $|V|$ and $|E|$.

In asymptotic notation – and *only* in asymptotic notation –

we'll drop the cardinality.

Example: $O(V + E)$.

藏行顯光 成就
共好 國立成功大學 九十週年 90th Anniversary of NCKU
Achieve Securely · Prosper Mutually

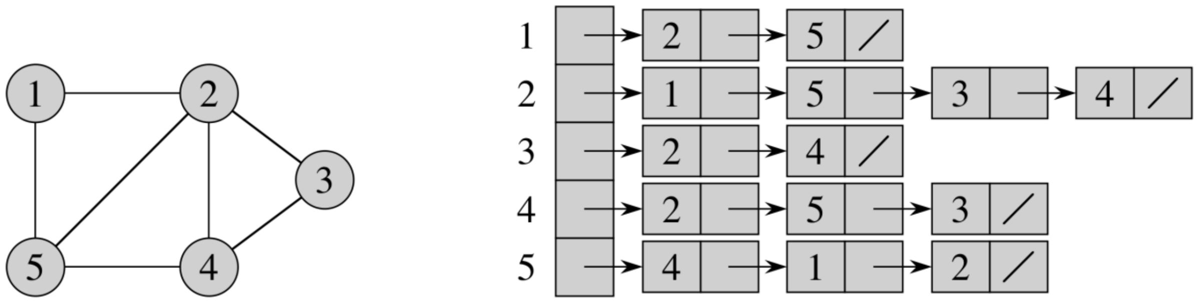
Adjacency lists

Array *Adj* of $|V|$ lists, one per vertex.

Vertex u's list has all vertices v such that $(u, v) \in E$.

(Works for both directed and undirected graphs.)

Example: For an undirected graph:



(a) (b)

	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

(c)

Adjacency lists

If edges have *weights*, can put the weights in the lists.

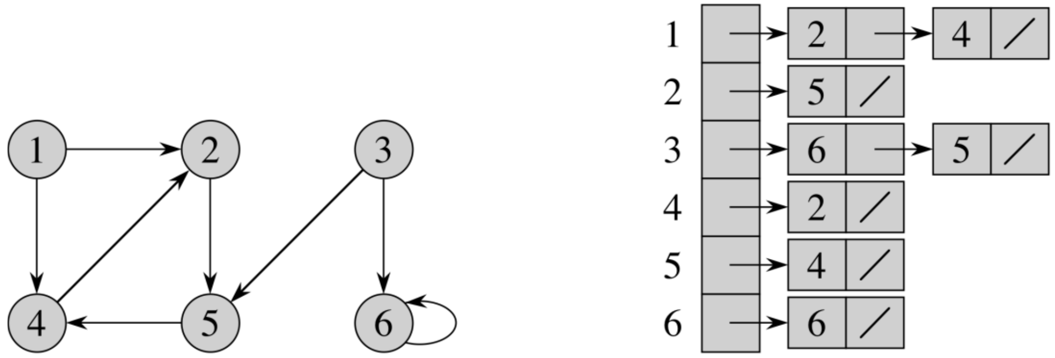
Weight: $w : E \to \boldsymbol{R}$

Space: $\Theta(V + E)$

Time: to list all vertices adjacent to u: $\Theta(\text{degree}(u))$

Time: to determine if $(u, v) \in E$: $\text{O}(\text{degree}(u))$

Example: For a directed graph:



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

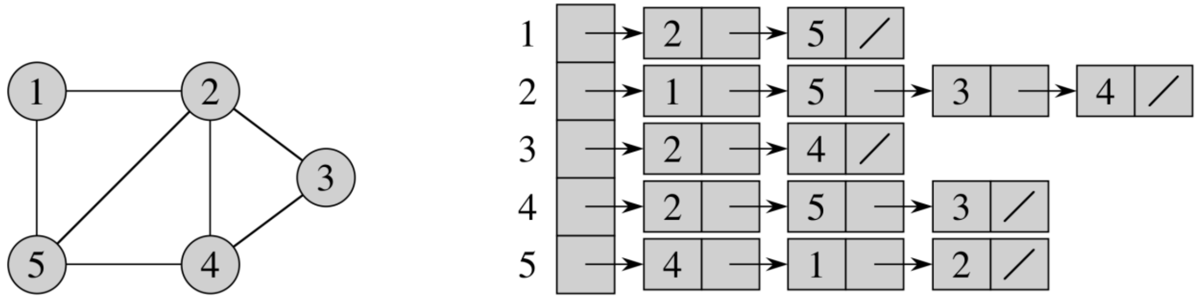
(a) (b) (c)

Same asymptotic space and time

Adjacency matrix

$|V| \times |V|$ matrix $A = (a_{ij})$

$$a_{ij} = \begin{cases} 1, \text{if} (i,j) \in E \\ 0, \text{otherwise} \end{cases}$$



(a) (b)

$$\begin{array}{c|ccccc|} & 1 & 2 & 3 & 4 & 5 \\ \hline 1 & 0 & 1 & 0 & 0 & 1 \\ 2 & 1 & 0 & 1 & 1 & 1 \\ 3 & 0 & 1 & 0 & 1 & 0 \\ 4 & 0 & 1 & 1 & 0 & 1 \\ 5 & 1 & 1 & 0 & 1 & 0 \\ \hline \end{array}$$

(c)

Space: $\Theta(V^2)$
Time: to list all vertices adjacent to u: $\Theta(V)$
Time: to determine if $(u,v) \in E$: $\Theta(1)$
Can store weights instead of bits for weighted graph

Breadth-first search

NCKU
National Cheng Kung University
1931

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

Breadth-first search

Input: Graph $G = (V, E)$, either directed or undirected, and source vertex $s \in V$

Output: $v.d =$ (smallest # of edges) from s to v, for all $v \in V$

Also $v.\pi = u$ such that (u, v) is last edge on a shortest path $s \to v$
- u **is** v**' s** ***predecessor***
- **set of edges** $\{(v.\pi, v): \boldsymbol{v \neq s}\}$ **forms a tree**

Prim' s minimum-spanning-tree algorithm and Dijkstra' s single-source shortest-paths algorithm use ideas similar to those in breadth-first search.

Breadth-first search

Idea: Send a wave out from s.

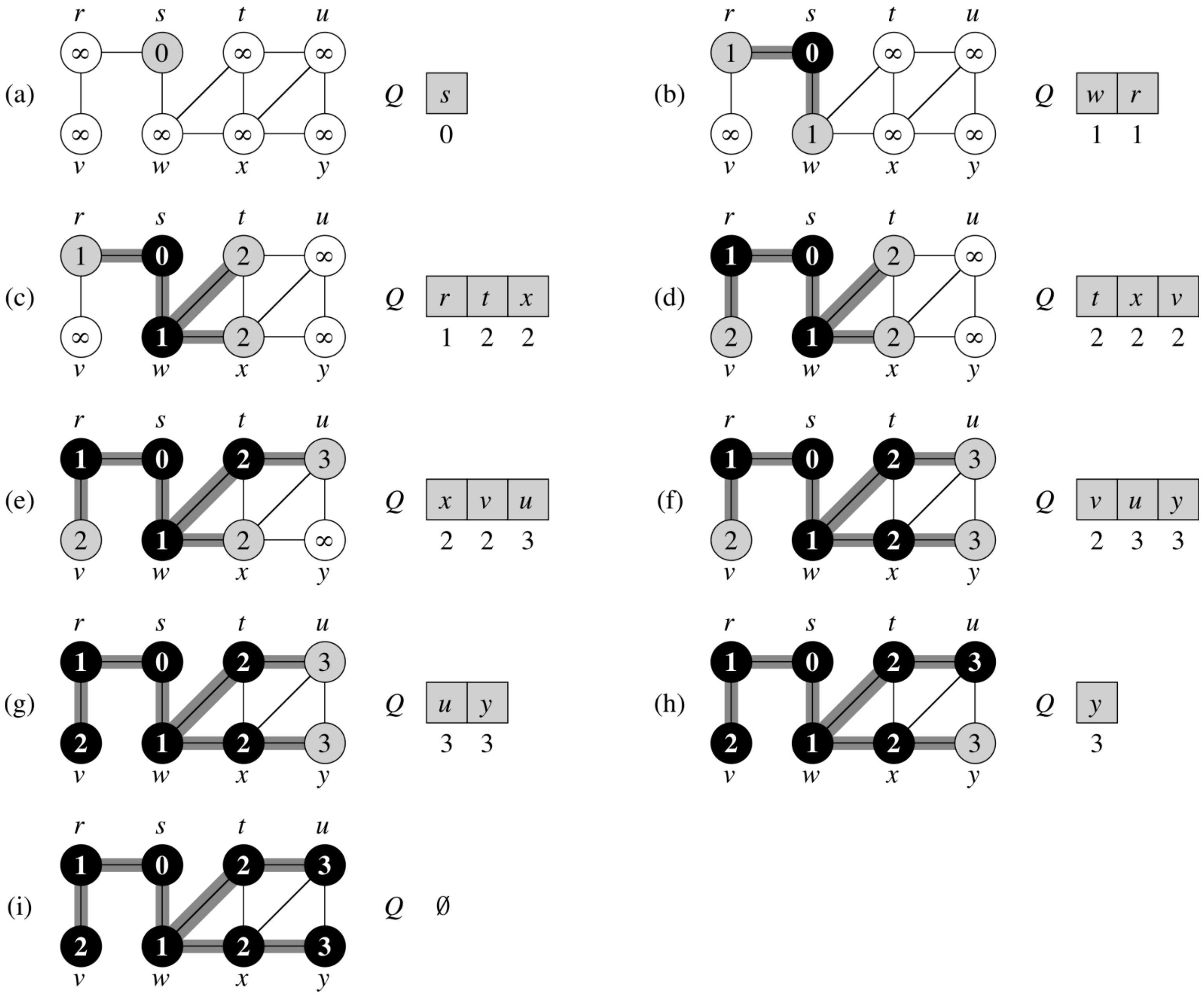
- First hits all vertices 1 edge from s.
- From there, hits all vertices 2 edges from s.
- Etc.

Use FIFO queue Q to maintain wavefront.

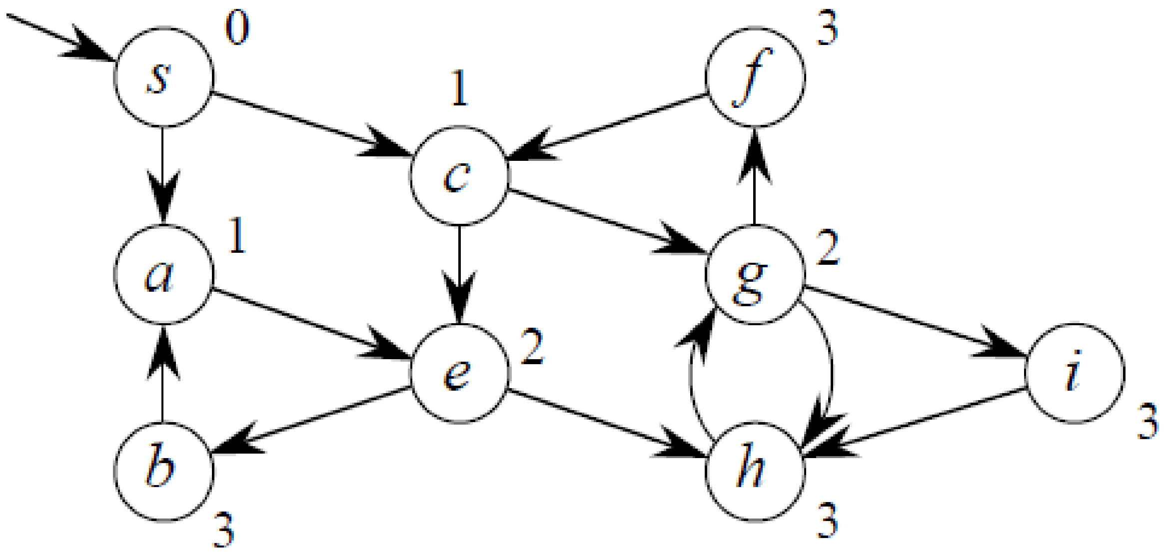
- $v \in Q$ if and only if wave has hit v but has not come out of v yet.

BFS(G, s)

```
for each vertex u ∈ G.V − {s} do
    u.color = WHITE
    u.d = ∞
    u.π = NIL
s.color = GRAY
s.d = 0
s.π = NIL
Q = ∅
ENQUEUE(Q, s)
while Q ≠ ∅ do
    u = DEQUEUE(Q)
    for each vertex v ∈ G.Adj[u] do
        if v.color == WHITE then
            v.color = GRAY
            v.d = u.d + 1
            v.π = u
            ENQUEUE(Q, v)
    u.color = BLACK
```


(a) Q s 0
(b) Q w r 1 1
(c) Q r t x 1 2 2
(d) Q t x v 2 2 2
(e) Q x v u 2 2 3
(f) Q v u y 2 3 3
(g) Q u y 3 3
(h) Q y 3
(i) Q Ø
r s t u
v w x y

Example: directed graph



Can show that Q consists of vertices with d values.

$$i \;\; i \;\; i \;\; \ldots. \; i \;\; i+1 \;\; i+1 \;\; \ldots \;\; i+1$$

- Only 1 or 2 values.
- If 2, differ by 1 and all smallest are first.

Since each vertex gets a finite *d* value at most once, values assigned to vertices are monotonically increasing over time.

BFS may not reach all vertices.

Time = $\mathrm{O}(V+E)$.

- $\mathrm{O}(V)$ because every vertex enqueued at most once.
- $\mathrm{O}(E)$ because every vertex dequeued at most once and we examine (u, v) only when u is dequeued. Therefore, every edge examined at most once if directed, at most twice if undirected.

Shortest paths

- Define the shortest-path distance $\delta(s, v)$ from s to v as the minimum number of edges in any path from vertex s to vertex v; if there is no path from s to v, then $\delta(s, v) = \infty$.

- We call a path of length $\delta(s, v)$ from s to v a shortest path from s to v.

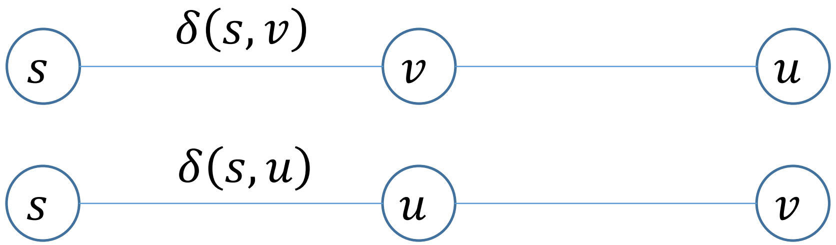
藏行顯光
成就共好
Achieve Securely
Prosper Mutually

Lemma 22.1

Let $G = (V, E)$ be a directed or undirected graph, and let $s \in V$ be an arbitrary vertex. Then, for any edge $(u, v) \in E$, $\delta(s, v) \le \delta(s, u) + 1$.

Proof

If u is reachable from s, then so is v. In this case, the shortest path from s to v cannot be longer than the shortest path from s to u followed by the edge (u, v), and thus the inequality holds.



If u is not reachable from s, then $\delta(s, u) = \infty$, and the inequality holds.

Lemma 22.2

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then upon termination, for each vertex $v \in V$, the value $v.d$ computed by BFS satisfies $v.d \geq \delta(s, v)$.

Proof

We use induction on the number of ENQUEUE operations. Our inductive hypothesis is that $v.d \geq \delta(s, v)$ for all $v \in V$.

The basis of the induction is the situation immediately enqueuing s of BFS. The inductive hypothesis holds here, because $s.d = 0 = \delta(s, s)$ and $v.d = \infty \geq \delta(s, v)$ for all $v \in V - \{s\}$.

For the inductive step, consider a white vertex v that is discovered during the search from a vertex u. The inductive hypothesis implies that $u.d \geq \delta(s, u)$. From the assignment performed by $v.d = u.d + 1$ and from Lemma 22.1, we obtain

$$v.d = u.d + 1 \geq \delta(s, u) + 1 \geq \delta(s, v)$$

Vertex v is then enqueued, and it is never enqueued again because it is also grayed and the if-then clause is executed only for white vertices. Thus, the value of $v.d$ never changes again, and the inductive hypothesis is maintained.

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

Lemma 22.3

Suppose that during the execution of BFS on a graph $G = (V, E)$, the queue Q contains the vertices $\langle v_1, v_2, \dots, v_r \rangle$, where v_1 is the head of Q and v_r is the tail. Then, $v_r.d \leq v_1.d + 1$ and $v_i.d \leq v_{i+1}.d$ for $i = 1, 2, \dots, r-1$.

Proof

The proof is by induction on the number of queue operations. Initially, when the queue contains only s, the lemma certainly holds.

For the inductive step, we must prove that the lemma holds after both dequeuing and enqueuing a vertex. If the head v_1 of the queue is dequeued, v_2 becomes the new head. By the inductive hypothesis, $v_1.d \leq v_2.d$. But then we have $v_r.d \leq v_1.d + 1 \leq v_2.d + 1$, and the remaining inequalities are unaffected. Thus, the lemma follows with v_2 as the head.

In order to understand what happens upon enqueuing a vertex, we need to examine the code more closely. When we enqueue a vertex v of BFS, it becomes v_{r+1}. At that time, we have already removed vertex u, whose adjacency list is currently being scanned, from the queue Q, and by the inductive hypothesis, the new head v_1 has $v_1.d \geq u.d$. Thus, $v_{r+1}.d = v.d = u.d + 1 \leq v_1.d + 1$. From the inductive hypothesis, we also have $v_r.d \leq u.d + 1$, and so $v_r.d \leq u.d + 1 = v.d = v_{r+1}.d$, and the remaining inequalities are unaffected. Thus, the lemma follows when v is enqueued.

Corollary 22.4

Suppose that vertices v_i and v_j are enqueued during the execution of BFS, and that v_i is enqueued before v_j. Then $v_i.d \leq v_j.d$ at the time that v_j is enqueued.

Proof

Immediate from Lemma 22.3 and the property that each vertex receives a finite d value at most once during the course of BFS.

We can now prove that breadth-first search correctly finds shortest-path distances.

Theorem 22.5

Let $G = (V, E)$ be a directed or undirected graph, and suppose that BFS is run on G from a given source vertex $s \in V$. Then, during its execution, BFS discovers every vertex $v \in V$ that is reachable from the source s, and upon termination, $v.d = \delta(s, v)$ for all $v \in V$. Moreover, for any vertex $v \neq s$ that is reachable from s, one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$.

Proof

Assume, for the purpose of contradiction, that some vertex receives a d value not equal to its shortest-path distance. Let v be the vertex with minimum $\delta(s, v)$ that receives such an incorrect d value: clearly $v \neq s$.

By Lemma 22.2, $v.d \geq \delta(s, v)$, and thus we have that $v.d > \delta(s, v)$.

Vertex v must be reachable from s, for if it is not, then $\delta(s, v) = \infty \geq v.d$.

Let u be the vertex immediately preceding v on a shortest path from s to v, so that $\delta(s, v) = \delta(s, u) + 1$. Because $\delta(s, u) < \delta(s, v)$, and because of how we chose v, we have $u.d = \delta(s, u)$. Putting these properties together, we have

$$v.d > \delta(s, v) = \delta(s, u) + 1 = u.d + 1$$

藏行顯光
成就共好
Achieve Securely
Prosper Mutually

$$v.d > \delta(s, v) = \delta(s, u) + 1 = u.d + 1 \quad (22.1)$$

Now consider the time when BFS chooses to dequeue vertex u from Q. At this time, vertex v is either white, gray, or black.

If v is white, then $v.d = u.d + 1$ contradicting inequality (22.1).

If v is black, then it was already removed from the queue and, by Corollary 22.4, we have $v.d \leq u.d$ again contradicting inequality (22.1).

If v is gray, then it was painted gray upon dequeuing some vertex w, which was removed from Q earlier than u and for which $v.d = w.d + 1$. By corollary 22.4, however, $w.d \leq u.d$, and so we have $v.d = w.d + 1 \leq u.d + 1$, once again contradicting inequality (22.1).

Thus we conclude that $v.d = \delta(s, v)$ for all $v \in V$. All vertices v reachable from s must be discovered, for otherwise they would have $\infty = v.d > \delta(s, v)$.

To conclude the proof of the theorem, observe that if $v.\pi = u$, then $v.d = u.d + 1$. Thus, we can obtain a shortest path from s to v by taking a shortest path form s to $v.\pi$ and then traversing the edge $(v.\pi, v)$

Depth-first search

NCKU
National Cheng Kung University
1931

藏行顯光
成就共好
Achieve Securely
Prosper Mutually
國立成功大學 九十週年
90th Anniversary of NCKU

Depth-first search

Input: $G = (V, E)$, directed or undirected. No source vertex given!

Output: 2 ***timestamps*** on each vertex:

- $v.d =$ **discovery time**
- $v.f =$ **finishing time**

These will be useful for other algorithms later on.

Can also compute $v.\pi$.

藏行顯光 成就共好 國立成功大學 九十週年 90th Anniversary of NCKU
Achieve Securely · Prosper Mutually

Will methodically explore *every* edge.

- Start over from different vertices as necessary.

As soon as we discover a vertex, explore from it.

- Unlike BFS, which puts a vertex on a queue so that we explore from it later.

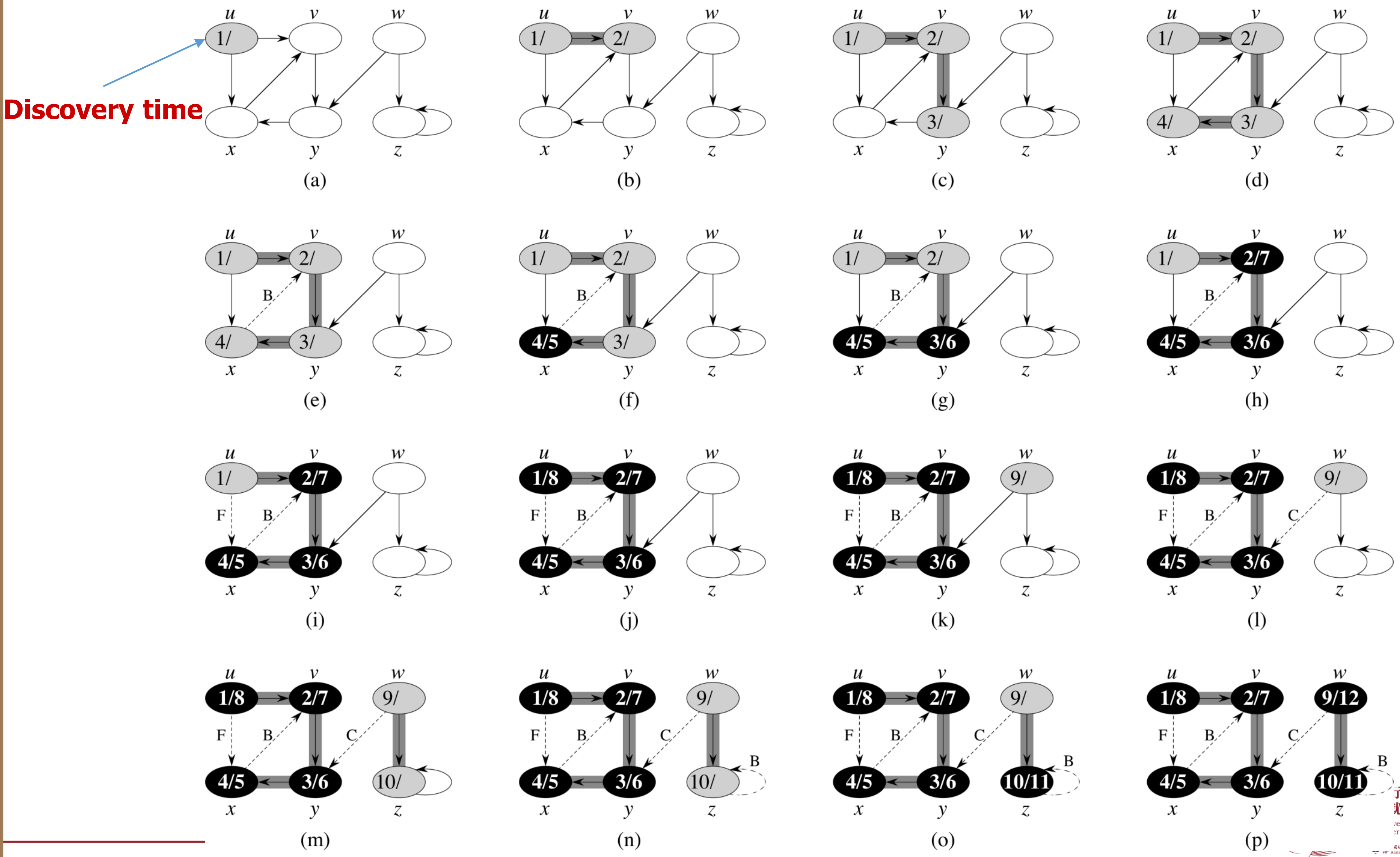
As DFS progresses, every vertex has a ***color***:

- WHITE = undiscovered
- GRAY = discovered, but not finished (not done exploring from it)
- BLACK = finished (have found everything reachable from it)

Discovery and finish times:

- Unique integers from 1 to $2|V|$.
- For all v, $v.d < v.f$.

In other words, $1 \leq v.d < v.f \leq 2|V|$.


Discovery time
u
v
w
x
y
z
1/
2/
3/
4/
4/5
3/6
2/7
1/8
9/
10/
10/11
9/12
B
F
C
(a)
(b)
(c)
(d)
(e)
(f)
(g)
(h)
(i)
(j)
(k)
(l)
(m)
(n)
(o)
(p)

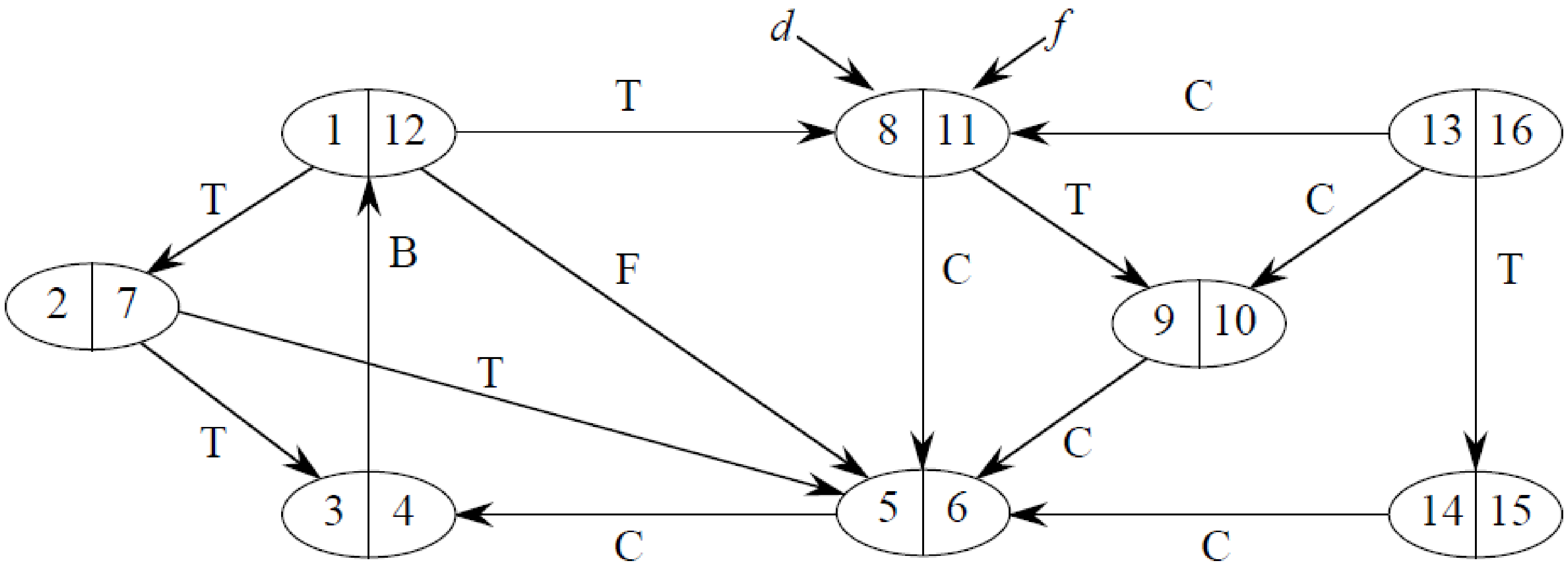
DFS(G)

```
for each vertex u ∈ G.V do
    u.color = WHITE
    u.π = NIL
time = 0
for each vertex u ∈ G.V do
    if u.color == WHITE then
        DFS-VISIT(G, u)
```

DFS-VISIT(G, u)

```
time = time + 1 // white vertex u has just been discovered
u.d = time
u.color = GRAY
for each vertex v ∈ G.Adj[u] do
    // explore edge (u, v)
    if v.color == WHITE then
        v.π = u
        DFS-VISIT(G, v)
u.color = BLACK // blacken u; it is finished
time = time + 1
u.f = time
```

Example:



Time= $\Theta(V + E)$.

- Similar to BFS analysis.
- Θ, not just O, since guaranteed to examine every vertex and edge.

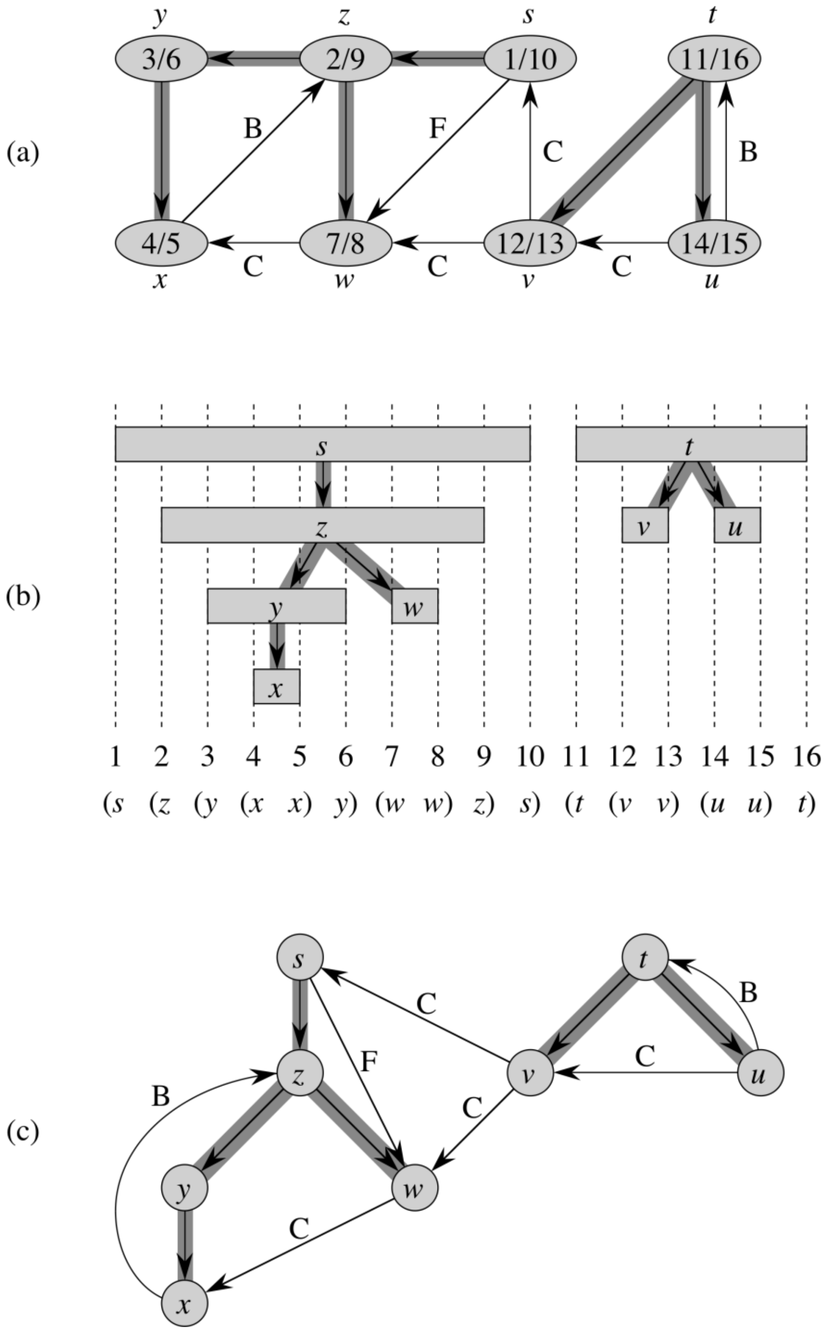
DFS forms a ***depth-first forest*** comprised of ≥ 1 ***depth-first trees***.

Each tree is made of edges (u, v) such that u is gray and v is white when (u, v) is explored.

Theorem 22.7 (Parenthesis theorem)

In any depth-first search of a (directed or undirected) graph $G = (V, E)$, for any two vertices u and v, exactly one of the following three conditions holds:

- The intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendant of the other in the depth-first forest
- The interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and u is a descendant of v in a depth-first tree.
- The interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and v is a descendant of u in a depth-first tree.


(a)
y z s t
3/6 2/9 1/10 11/16
B F C B
4/5 7/8 12/13 14/15
C C C
x w v u
(b)
s t
z v u
y w
x
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
(s (z (y (x x) y) (w w) z) s) (t (v v) (u u) t)
(c)
s t z v u y w x
C B F C C B C

Proof

We begin with the case in which $u.d < v.d$. We consider two subcases, according to whether $v.d < u.f$ or not.

The first subcase occurs when $v.d < u.f$, so v was discovered while u was still gray, which implies that v is a descendant of u. Moreover, since v was discovered more recently than u, all of its outgoing edges are explored, and v is finished, before the search returns to and finishes u. In this case, therefore, the interval $[v.d, v.f]$ is entirely contained within the interval $[u.d, u.f]$.

In the other subcase, $u.f < v.d$, and by inequality (22.2), $u.d < u.f < v.d < v.f$; thus the intervals $[u.d, u.f]$ and $[v.d, v.f]$ are disjoint. Because the intervals are disjoint, neither vertex was discovered while the other was gray, and so neither vertex is a descendant of the other.

Corollary 22.8 (Nesting of descendants' intervals)

Vertex v is a proper descendant of vertex u in the depth-first forest for a (directed or undirected) graph G if and only if $u.d < v.d < v.f < u.f$.

Proof

Immediate from Theorem 22.7.

Theorem 22.9 (White-path theorem)

In a depth-first forest of a (directed or undirected) graph $G = (V, E)$, vertex v is a descendant of vertex u if and only if at the time $u.d$ that the search discovers u, there is a path from u to v consisting entirely of white vertices.

Proof

⇒: If $v = u$, then the path from u to v contains just vertex u, which is still white when we set the value of $u.d$.

Now, suppose that v is a proper descendant of u in the depth-first forest. By corollary 22.8, $u.d < v.d$, and so v is white at time $u.d$. Since v can be any descendant of u, all vertices on the unique simple path from u to v in the depth-first forest are white at time $u.d$.

$\Leftarrow$: Suppose that there is a path of white vertices from u to v at time $u.d$, but v does not become a descendant of u in the depth-first tree.

Without loss of generality, assume that every vertex other than v along the path becomes a descendant of u. (Otherwise, let v be the closest vertex to u along the path that doesn't become a descendant of u.)

Let w be the predecessor of v in the path, so that w is a descendant of u (w and u may in fact be the same vertex).

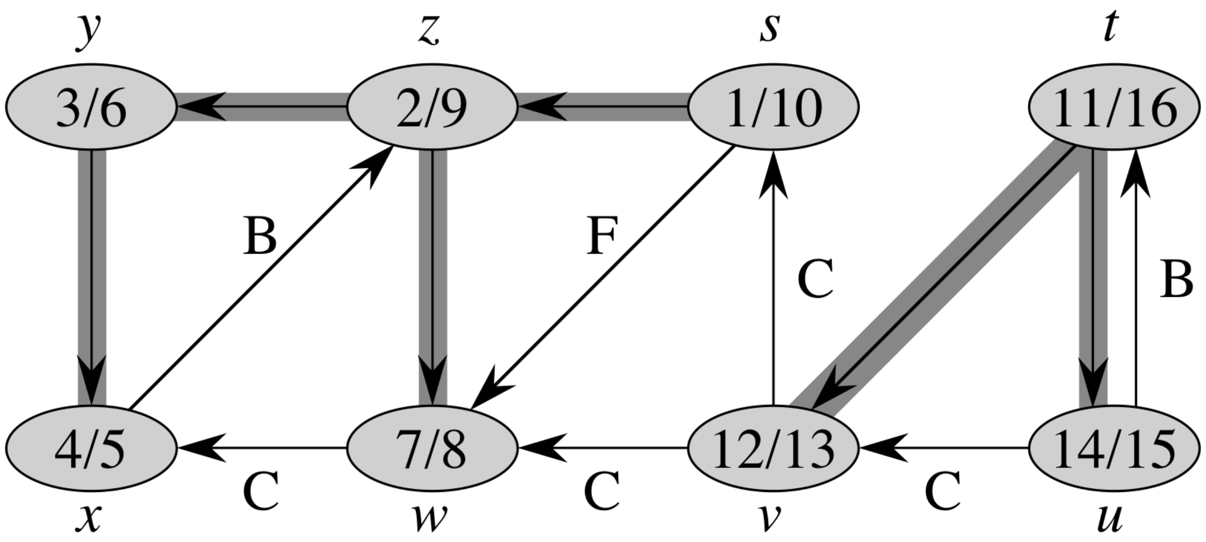
By corollary 22.8, $w.f \leq u.f$. Because v must be discovered after u is discovered, but before w is finished, we have $u.d < v.d < w.f \leq u.f$. Theorem 22.7 then implies that the interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$. By corollary 22.8, v must after all be a descendant of u.

Classification of edges

- ***Tree edge***: in the depth-first forest.

  Found by exploring (u, v).
- ***Back edge***: (u, v), where u is a descendant of v.
- ***Forward edge***: (u, v), where v is a descendant of u, but not a tree edge.
- ***Cross edge***: any other edge. Can go between vertices in the same depth-first tree or in different depth-first trees.

(a)



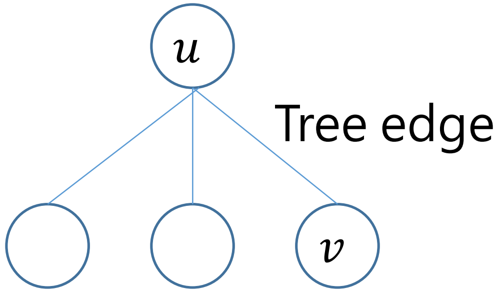
Theorem 22.10

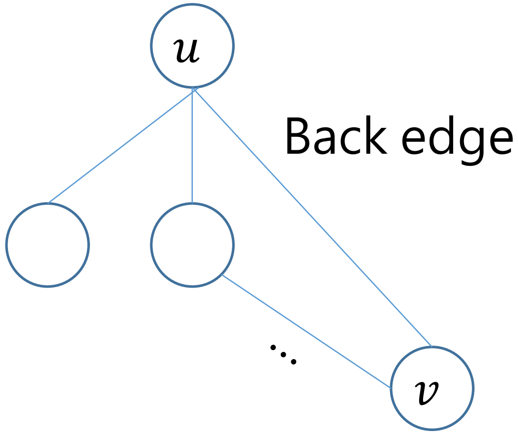
In a depth-first search of an undirected graph $G = (V, E)$, every edge of G is either a tree edge or a back edge.

Proof

Let (u, v) be an arbitrary edge of G, and suppose without loss of generality that $u.d < v.d$. Then the search must discover and finish v before it finishes u (while u is gray), since v is on u's adjacency list.

If the first time that the search explores edge (u, v), it is in the direction from u to v, then v is undiscovered (white) until that time, for otherwise the search would have explored this edge already in the direction from v to u. Thus, (u, v) becomes a tree edge.





If the search explores (u, v) first in the direction from v to u, then (u, v) is a back edge, since u is still gray at the time the edge is first explored.

藏行顯光
成就共好

Achieve Securely
Prosper Mutually

國立成功大學 九十週年
90th Anniversary of NCKU

國立成功大學
National Cheng Kung University