

Graph Algorithms - I

CSIE 2136 Algorithm Design and Analysis, Fall 2018

<https://cool.ntu.edu.tw/courses/61>

Hsu-Chun Hsiao



Agenda

Graph basics

Real-world applications

Representations of graphs

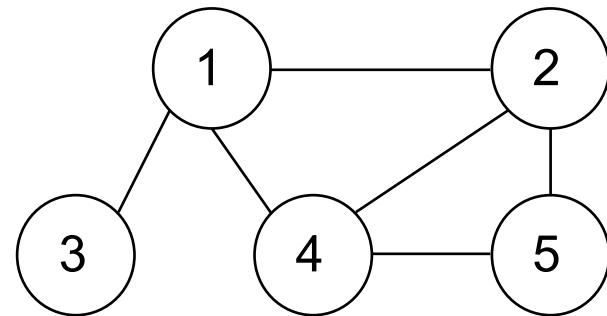
Graph traversal

- Breadth-first search (BFS)
- Depth-first search (DFS)

Graph basics

A graph G is a pair $G = (V, E)$

- V = set of vertices, or nodes
- E = set of edges, or links



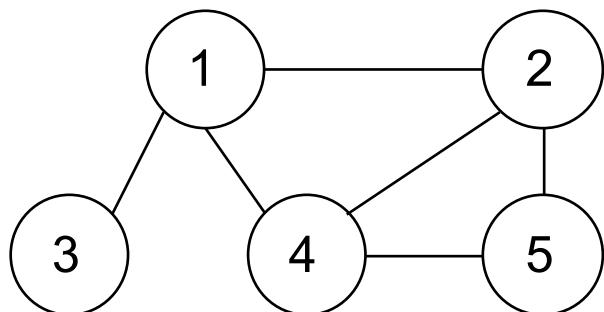
$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1,2), (1,3), (1,4), (2,5), (4,5)\}$$

Graph basics

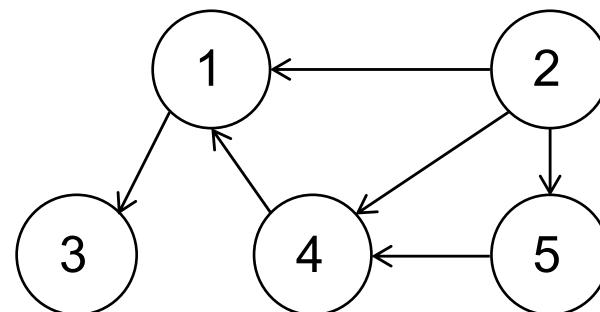
Type of graphs

- **Undirected:** edge $(u,v) = (v,u)$
- **Directed:** edge (u,v) goes from vertex u to vertex v
- **Weighted:** graph associates weights with edges



$$V = \{1, 2, 3, 4, 5\}$$

$$E = \{(1,2), (1,3), (1,4), (2,4), (2,5), (4,5)\}$$



$$V = \{1, 2, 3, 4, 5\}$$

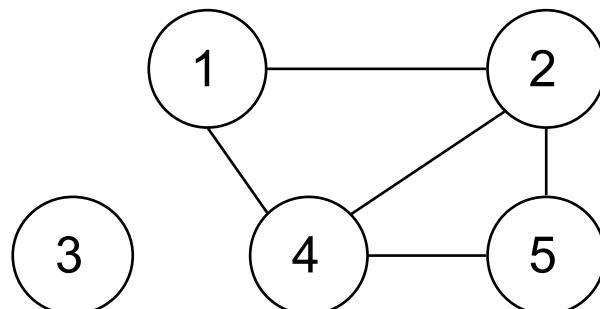
$$E = \{(1,3), (2,1), (2,4), (2,5), (4,1), (5,4)\}$$

Graph basics

A **path** in a graph is a sequence of edges which connect a sequence of vertices

Vertex v is **reachable** from u if there exists a path from u to v

A **cycle** is a path v_1, v_2, \dots, v_x in which $v_1 = v_x$, $x > 2$, and the first $x - 1$ nodes are all distinct



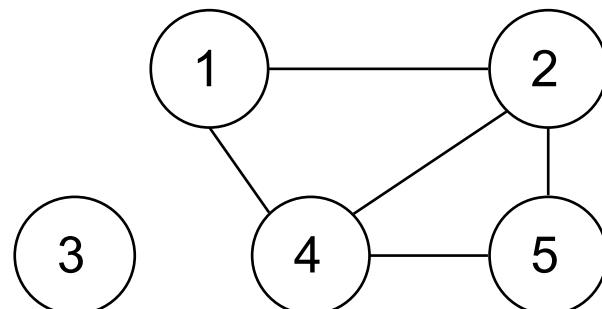
2,5,4,2 is a cycle.

Graph basics

A **connected** graph has a path from every vertex to every other

The **degree** of a vertex u is the number of edges incident to u

- In-degree of u = # of edges (x, u) in a directed graph
- Out-degree of u = # of edges (u, x) in a directed graph

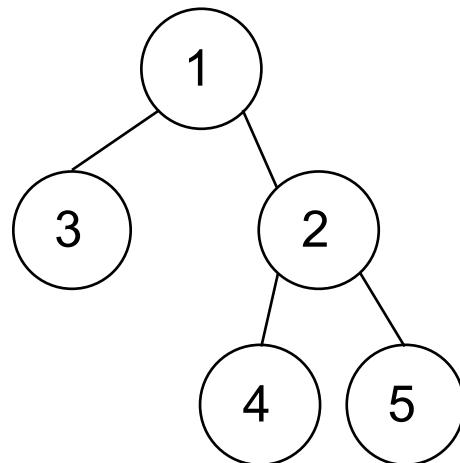


This graph is not connected.
Degree of vertex 2 is 3.

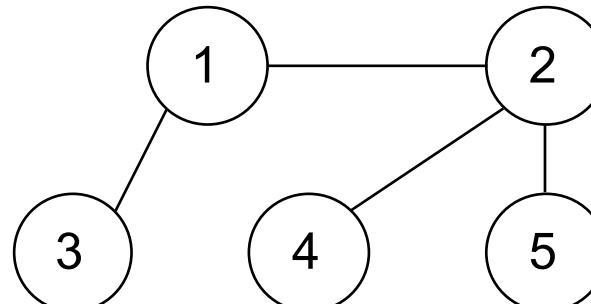
Graph basics

Tree is a connected, acyclic, undirected graph

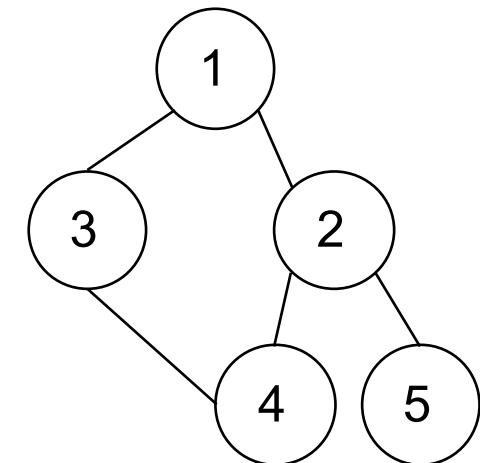
Forest is an acyclic, undirected but possibly disconnected graph



Tree?



Tree?



Tree?

Graph basics

Theorem. Let G be an undirected graph. The following statements are equivalent:

- G is a tree
- Any two vertices in G are connected by a unique simple path
- G is connected, but if any edge is removed from E , the resulting graph is disconnected.
- G is connected and $|E| = |V| - 1$
- G is acyclic, and $|E| = |V| - 1$
- G is acyclic, but if any edge is added to E , the resulting graph contains a cycle

See textbook Appendix B.5 for formal proofs.

Modeling real-world information

Graphs can model real-world structural info

- A vertex is an object with some properties
- An edge represents a relationship between two vertices

For example, in social network:

- vertex = user
- edge = friendship

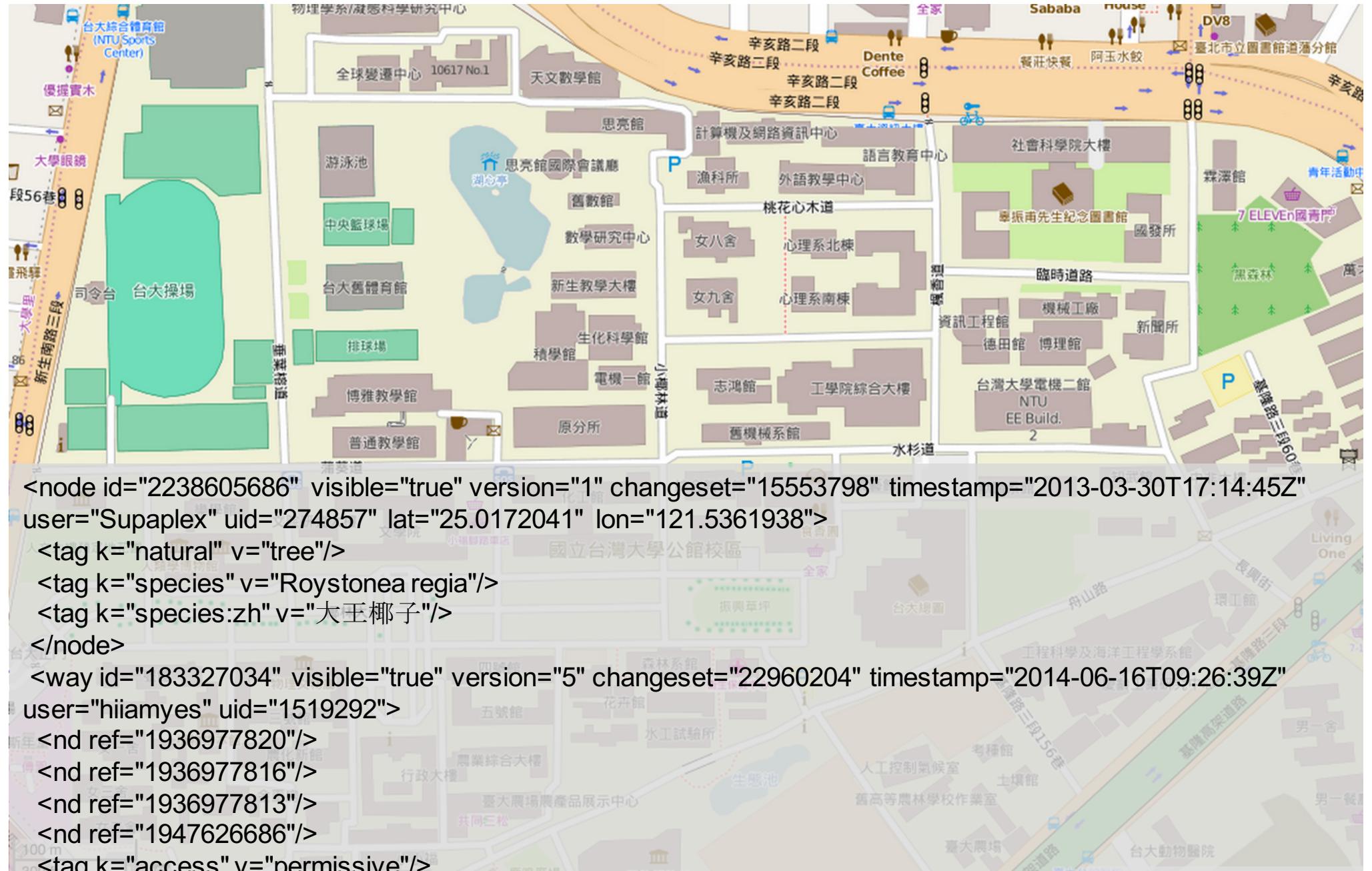
Modeling real-world information

Let's see some examples & answer following questions:

- What do the vertices represent?
- What do the edges represent?
- Undirected or directed?
- Is it always connected?
- Is it a tree?



The Celebrity Twitter Ecosystem <http://www.nytimes.com/2009/03/29/fashion/29twitter.html>



```
<node id="2238605686" visible="true" version="1" changeset="15553798" timestamp="2013-03-30T17:14:45Z"
  user="Supaplex" uid="274857" lat="25.0172041" lon="121.5361938">
  <tag k="natural" v="tree"/>
  <tag k="species" v="Roystonea regia"/>
  <tag k="species:zh" v="大王椰子"/>
</node>
<way id="183327034" visible="true" version="5" changeset="22960204" timestamp="2014-06-16T09:26:39Z"
  user="hiiamyes" uid="1519292">
  <nd ref="1936977820"/>
  <nd ref="1936977816"/>
  <nd ref="1936977813"/>
  <nd ref="1947626686"/>
  <tag k="access" v="permissive"/>
  <tag k="highway" v="service"/>
  <tag k="maxspeed" v="20"/>
  <tag k="name" v="臨時道路"/>
</way>
```

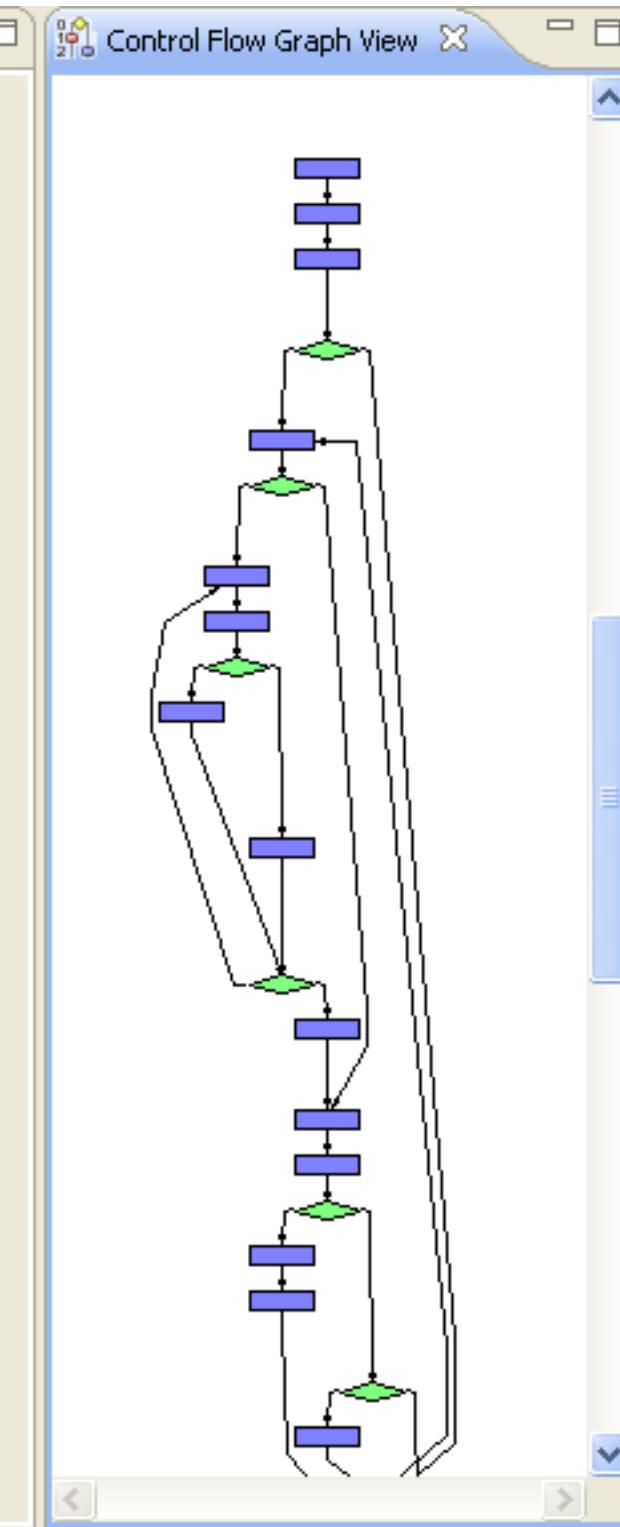


OpenStreetMap

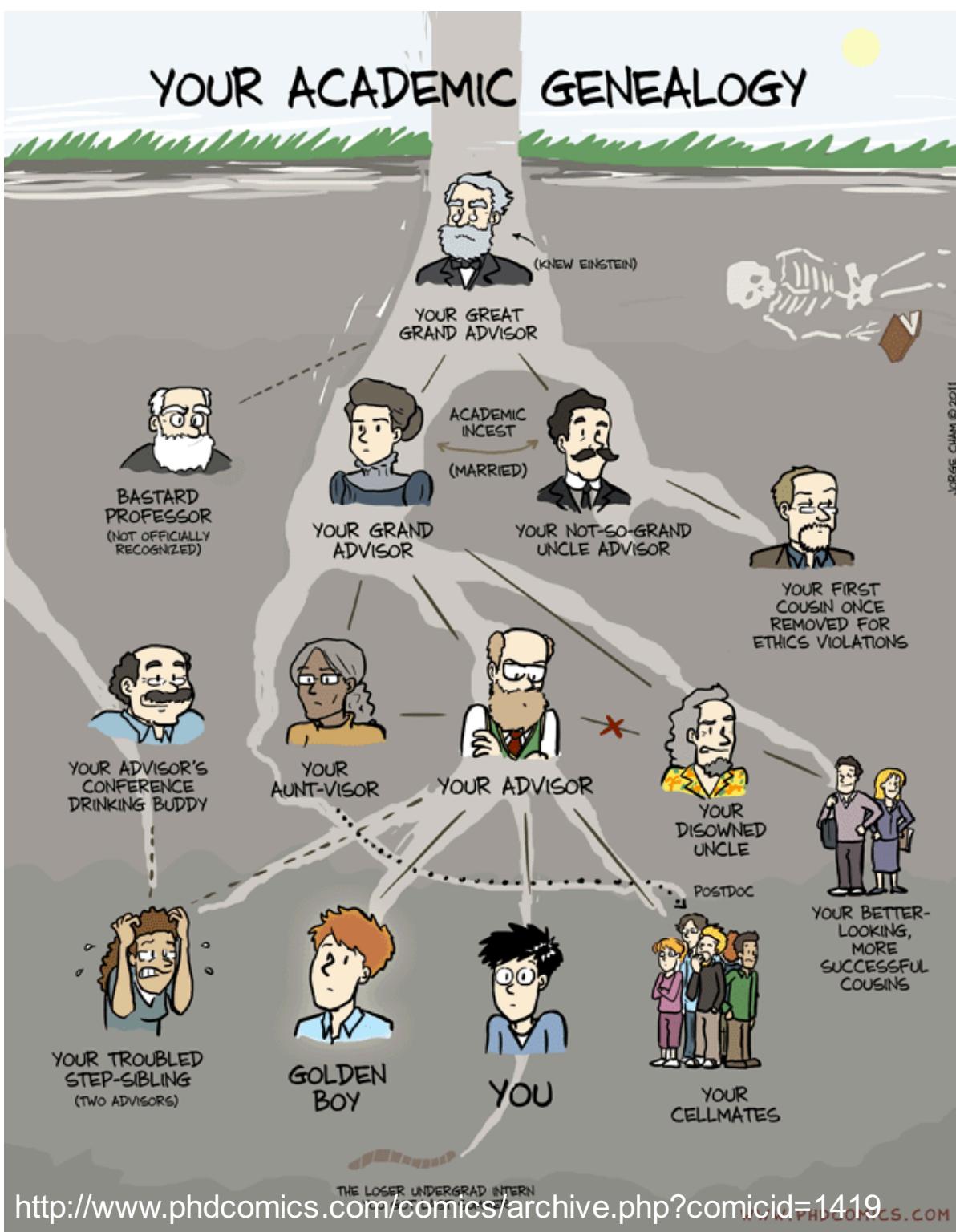
12

```
static int runTest2(int a) {
    int i = 0;
    int j = 0;
    int m_const = 1;

    if (10 > a) {
        do {
            a = i;
            if (2 > a) {
                do {
                    j++;
                    i++;
                    if (j > i) {
                        i++;
                    }
                    else {
                        j++;
                    }
                }
            }
            while (3 > j);
            a= j;
        }
        i++;
        j--;
        if(a > 10){
            i = i + m_const;
            i++;
        }else{
            if(j == a){
                j = j + 1;
            }
        }
    }
}
```



YOUR ACADEMIC GENEALOGY



Sybil Attack Detection

Sybil attack: a single entity forges multiple identities

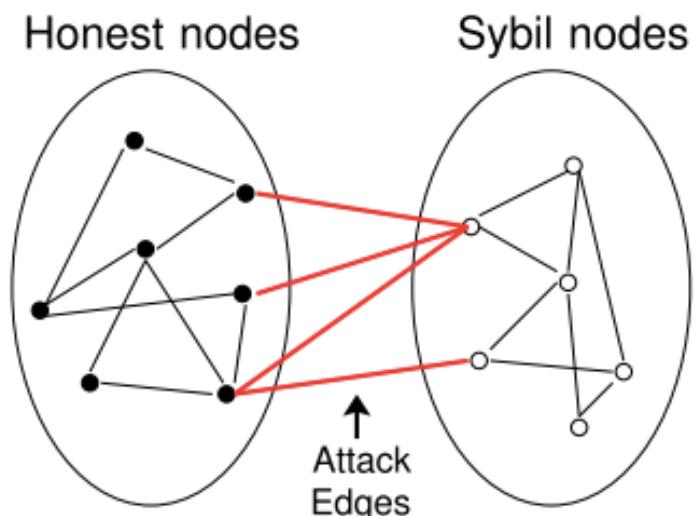


Figure 1. The social network.

Assumption: “if malicious users create too many sybil identities, the graph will have a small quotient cut— i.e., a small set of edges (the attack edges) whose removal disconnects a large number of nodes (all the sybil identities). On the other hand, “fast mixing” social networks do not tend to have such cuts.” [1]

[1] H. Yu, P. B. Gibbons, M. Kaminsky, and F. Xiao, “SybilLimit: A near-optimal social network defense against sybil attacks,” *IEEE/ACM Trans. Netw.*, vol. 18, no. 3, pp. 885–898, 2010.

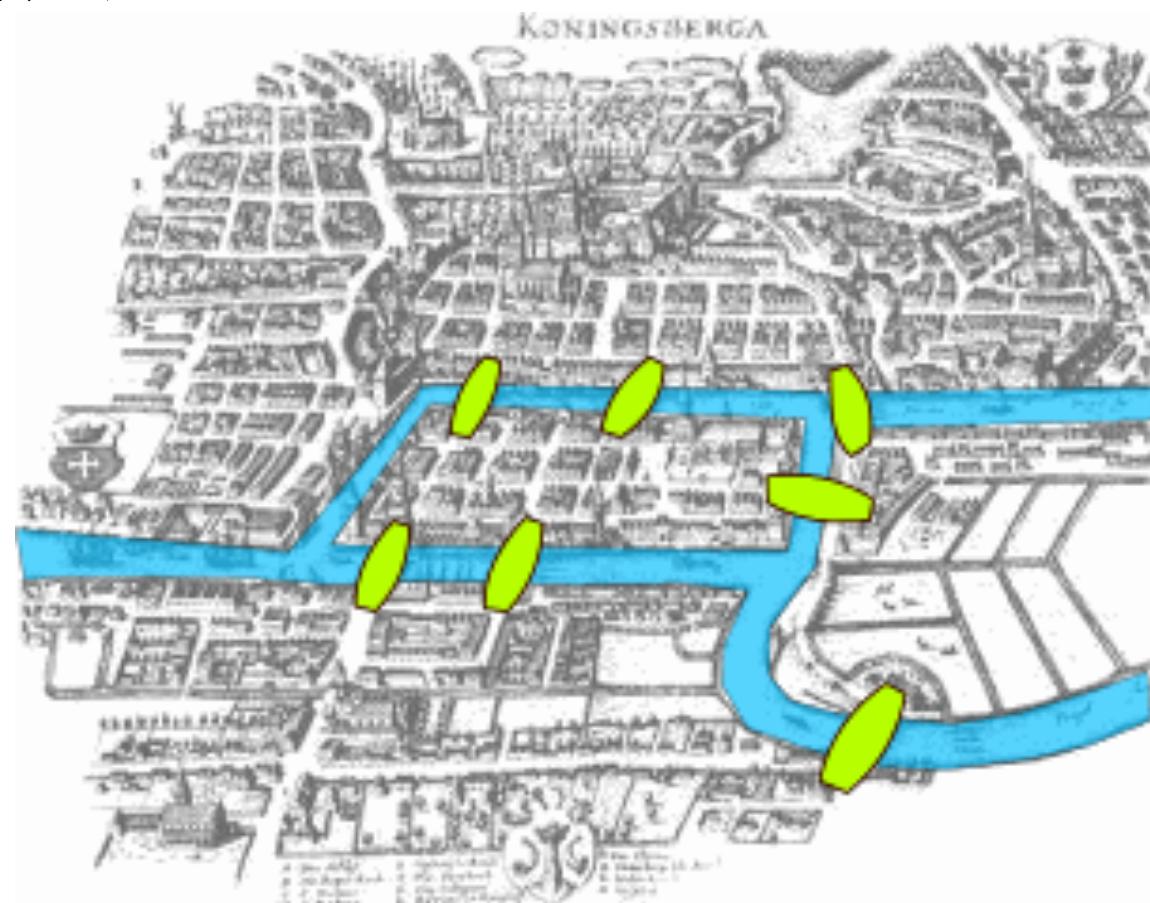
The origin of graph theory

七橋問題 (Seven Bridges of Königsberg)

在所有橋都只能走一遍的前提下，如何才能把這個地方所有的橋都走遍？

Graph modeling

- Vertex = ?
- Edge = ?

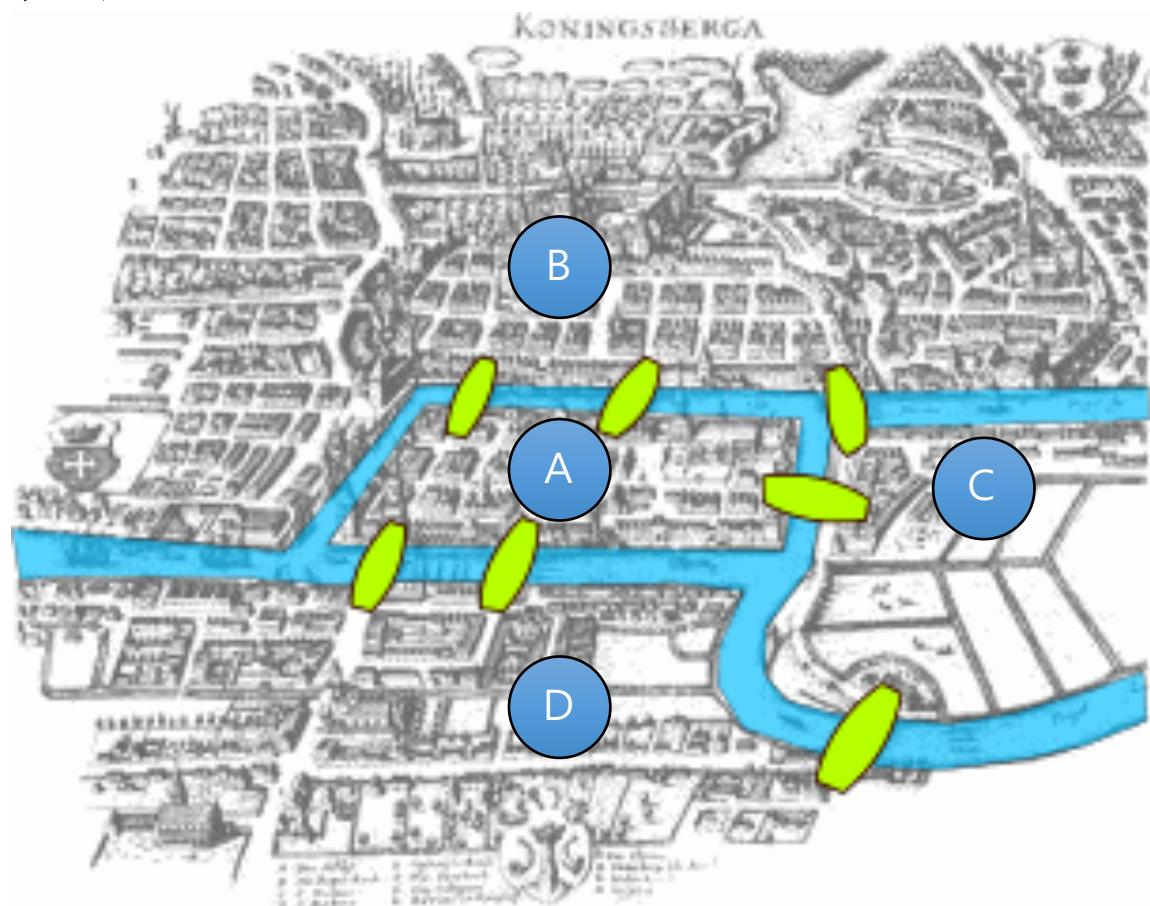
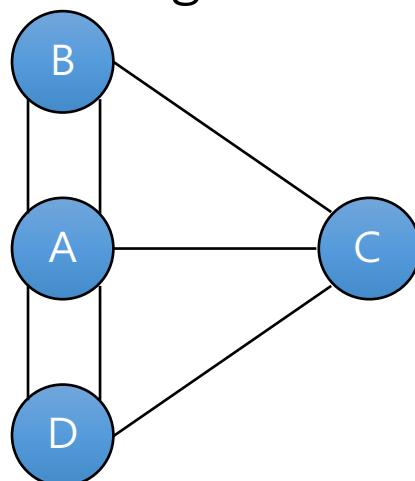


七橋問題 (Seven Bridges of Königsberg)

在所有橋都只能走一遍的前提下，如何才能把這個地方所有的橋都走遍？

Graph modeling

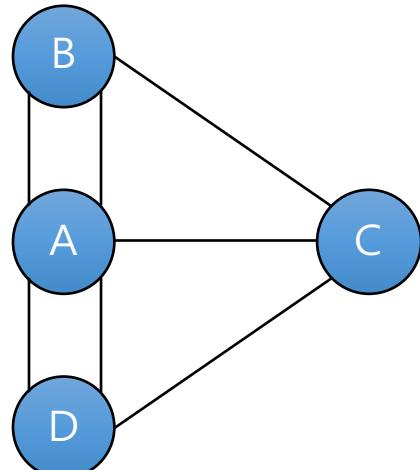
- Vertex = ?
- Edge = ?



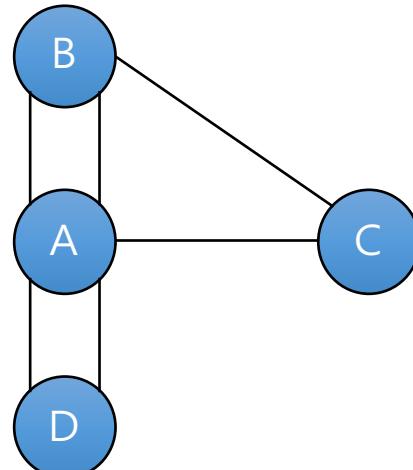
一筆畫問題 (Euler path)

Euler path: Can you traverse each edge in a connected graph exactly once without lifting the pen from the paper?

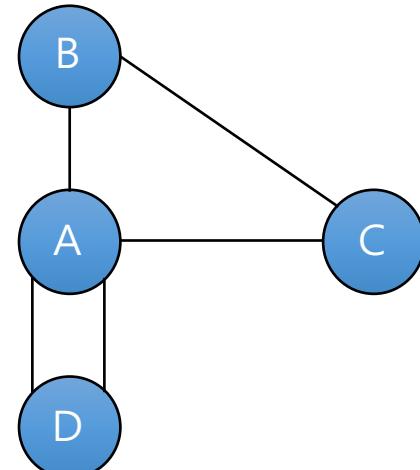
Euler tour: Can you finish where you started?



☹ Euler path
☹ Euler tour



☺ Euler path
☹ Euler tour



☺ Euler path
☺ Euler tour

一筆畫問題 (Euler path)

Euler path: Can you traverse each edge in a connected graph exactly once without lifting the pen from the paper?

Eulertour: Can you finish where you started?

Is it possible to determine whether a graph has an Euler path or an Eulertour, without necessarily having to find one explicitly?

Euler path and Euler tour

Solved by Leonhard Euler in 1736

Marked the beginning of graph theory



Euler path and Euler tour

G has an Euler path $\Leftrightarrow G$ has exactly 0 or 2 odd vertices

G has an Eulertour \Leftrightarrow all vertices must be even vertices

- Even vertices = vertices with even degrees
- Odd vertices = vertices with odd degrees

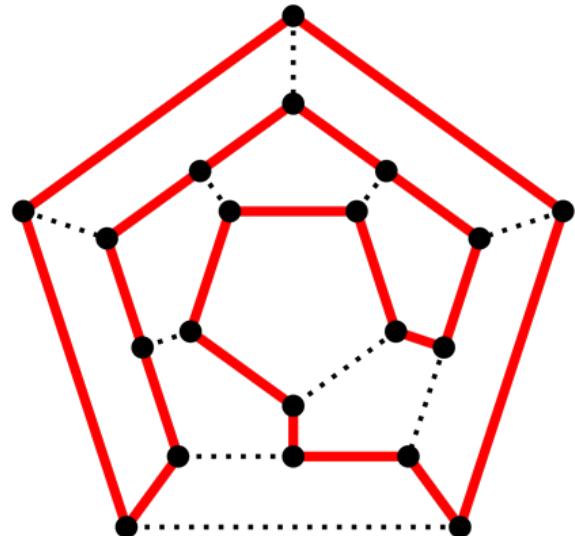
Any intuition?

漢彌爾頓路徑問題 (Hamiltonian path problem)

Hamiltonian path: Can you find a path that visits each vertex exactly once?

Hamiltonian cycle: Can you finish where you started?

Both problems are NP-complete

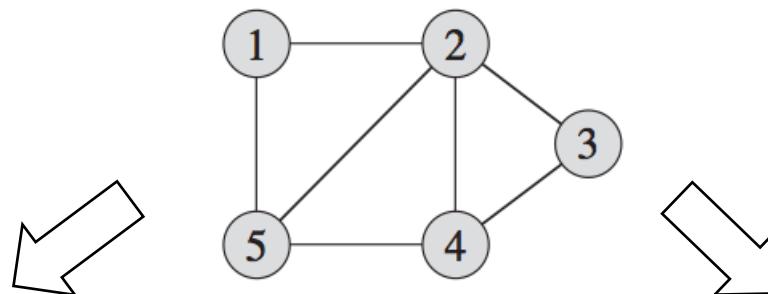


Representations of Graph

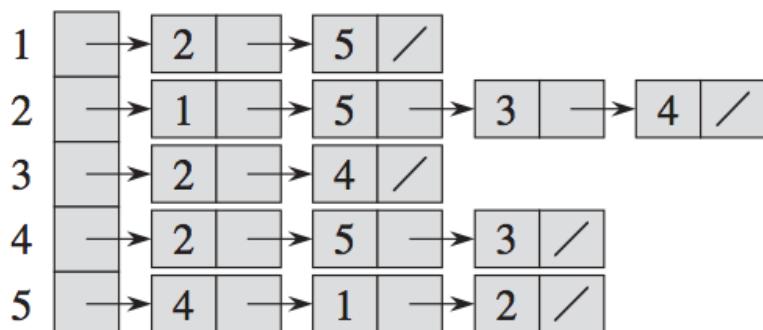
Representation of graphs

How to represent a graph in computer programs?

Two standard ways to represent a graph $G = (V, E)$:



Adjacency lists



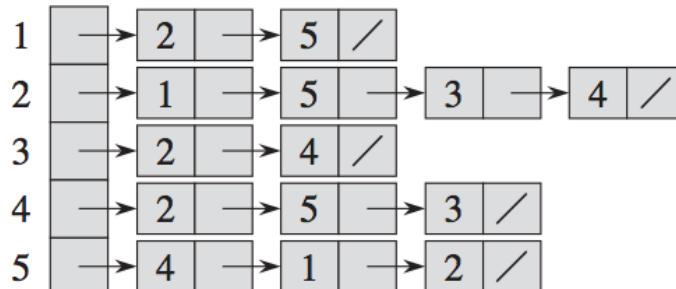
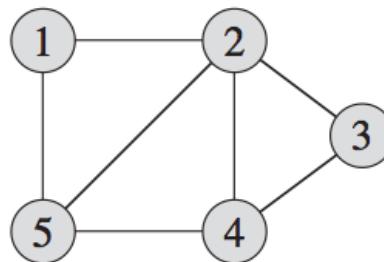
Adjacency matrix

	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

Graph representation: adjacency lists

Adjacency lists = vertex indexed array of lists

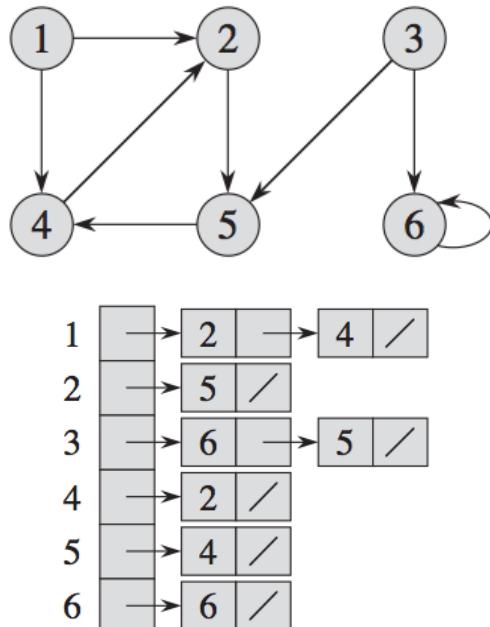
- An array Adj of $|V|$ lists
- One list per vertex
- For $u \in V$, $Adj[u]$ consists of all vertices adjacent to u
- If weighted, store weights in the adjacency lists as well



Graph representation: adjacency lists

Adjacency lists = vertex indexed array of lists

- An array Adj of $|V|$ lists
- One list per vertex
- For $u \in V$, $Adj[u]$ consists of all vertices adjacent to u
- If weighted, store weights in the adjacency lists as well



Space and time complexity

Express complexity in $|E|$ and $|V|$ ($|$ omitted for simplicity)

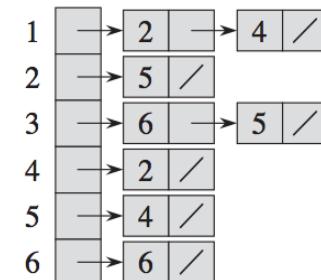
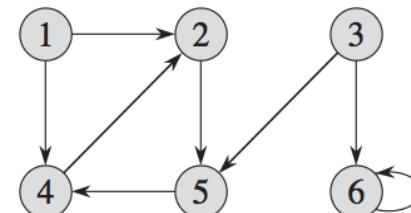
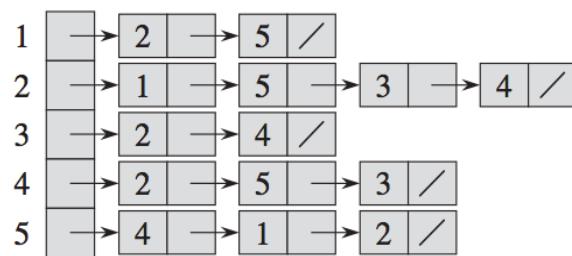
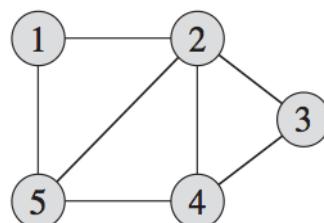
Space: sum of len of all adjacency lists + array len

Undirected: $2E+V = \Theta(E+V)$

- Each edge appears in *Adj* exactly twice

Directed: $E+V = \Theta(E+V)$

- Each edge appears in *Adj* exactly once



Space and time complexity

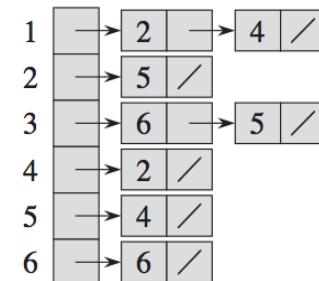
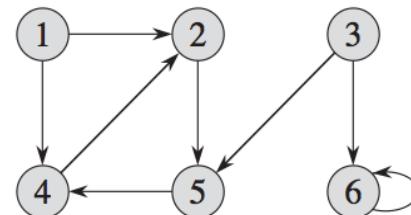
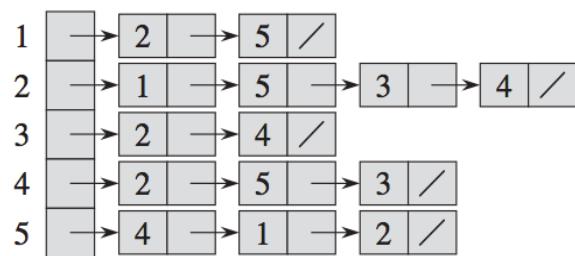
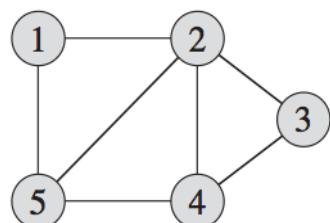
Time: checking if an edge (u,v) is in G takes $O(\text{degree}(u))$

- Linearly search for v in $\text{Adj}[u]$

Time: listing all neighbors of a vertex takes $\Theta(\text{degree}(u))$

Time: listing all edges takes $\Theta(E+V)$ time

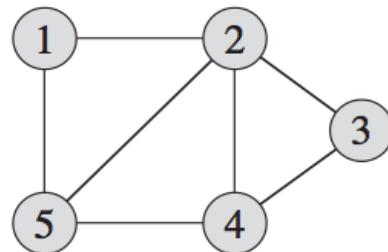
Finding the in-degree or out-degree of a vertex?



Graph representation: adjacency matrix

Adjacency matrix = $V \times V$ matrix A with $A_{uv} = 1$ iff (u, v) is an edge

- For undirected graphs, A is symmetric; i.e., $A = A^T$
- If weighted, store weights instead of bits in A

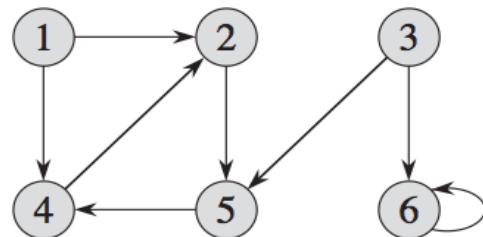


	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Graph representation: adjacency matrix

Adjacency matrix = $V \times V$ matrix A with $A_{uv} = 1$ iff (u, v) is an edge

- For undirected graphs, A is symmetric; i.e., $A = A^T$
- If weighted, store weights instead of bits in A



	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

Space and time complexity

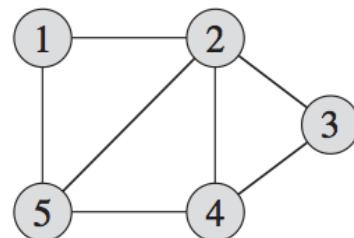
Space: $\Theta(V^2)$

Time: check if $(u, v) \in G$ takes $\Theta(1)$ time

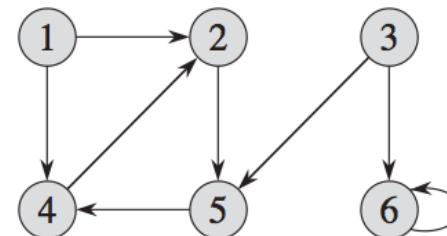
Time: list all neighbors of a vertex takes $\Theta(V)$ time

Time: Identify all edges takes $\Theta(V^2)$ time

Finding the in-degree or out-degree of a vertex?



	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



	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

Comparing the two representations

	Space	Time to check an edge	Time to list all neighbors of a vertex	Time to list all edges
Adjacency lists	$\Theta(E+V)$	$O(\text{degree}(u))$	$\Theta(\text{degree}(u))$	$\Theta(E+V)$
Adjacency matrix	$\Theta(V^2)$	$\Theta(1)$	$\Theta(V)$	$\Theta(V^2)$

Adjacency-list representation is suited to **sparse** graphs

- E is much less than V^2 , e.g., $E \approx V$

Adjacency-matrix representation is suited to **dense** graphs

- E is on the order of V^2

Besides graph density, you may also choose a data structure based on the performance of other operations

- E.g., which one is more efficient to answer “in-degree of a vertex”?

Breadth-first Search

Textbook Chapter 22.2

Graph traversal (or graph searching)

From a given source vertex s , systematically follow the edges of the graph to visit all reachable vertices

Useful to discover the structure of a graph

Standard graph-searching algorithms

- Breadth-first Search (BFS, 廣度優先搜尋)
- Depth-first Search (DFS, 深度優先搜尋)

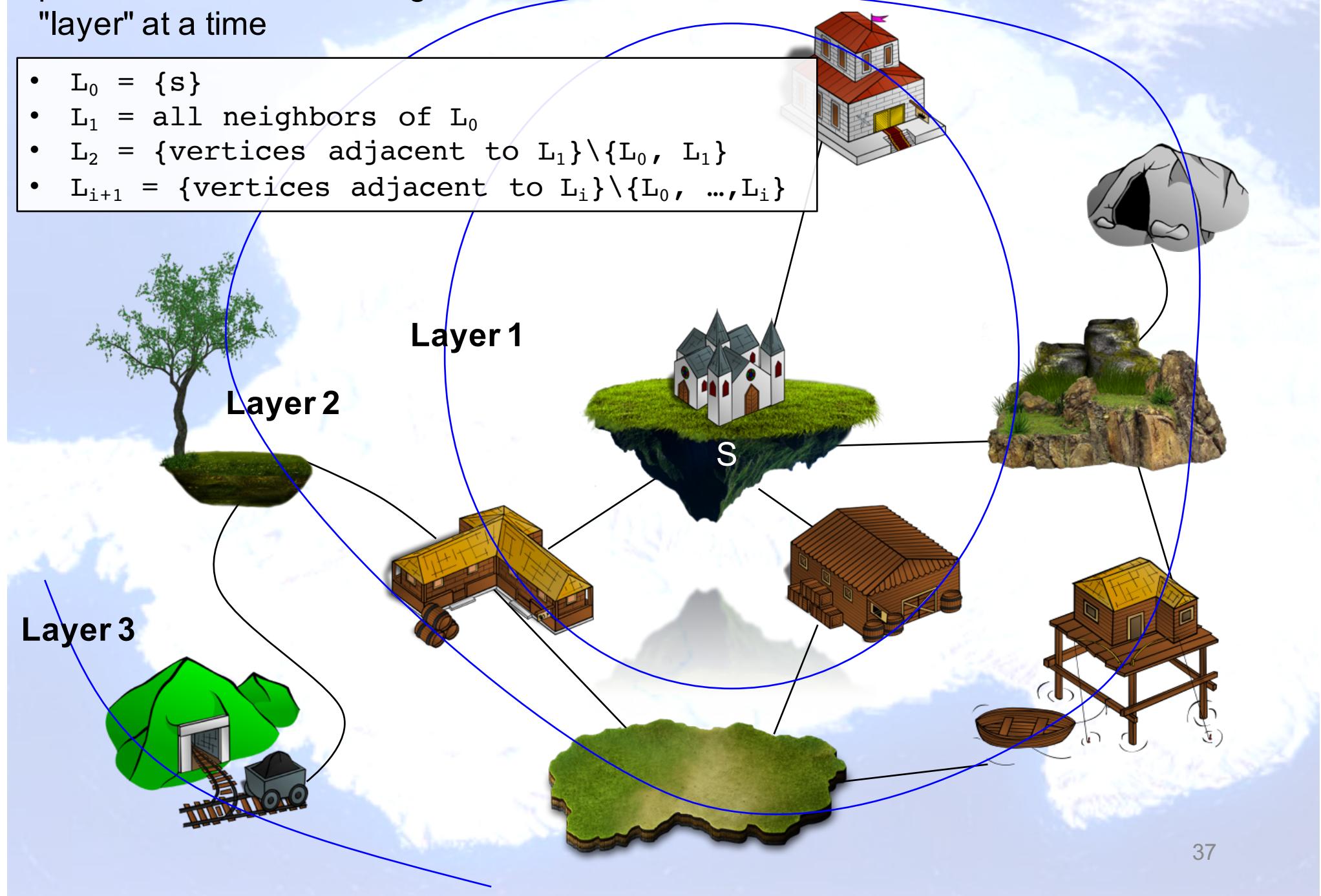
Ada聽到小道消息，地球發現了極神秘的無限寶石，然而各方人馬想要在行前趕緊規劃路線，到時才能比別人快點找到。

要怎麼才能有系統地探索過所有的點呢？



Intuition. Explore outward from s in all possible directions, adding vertices one "layer" at a time

- $L_0 = \{s\}$
- $L_1 = \text{all neighbors of } L_0$
- $L_2 = \{\text{vertices adjacent to } L_1\} \setminus \{L_0, L_1\}$
- $L_{i+1} = \{\text{vertices adjacent to } L_i\} \setminus \{L_0, \dots, L_i\}$



Breadth-first Search (BFS)



Input: Directed or undirected graph $G = (V, E)$ and source vertex s

Output: a breadth-first tree with root s (T_{bfs}) that contains all reachable vertices

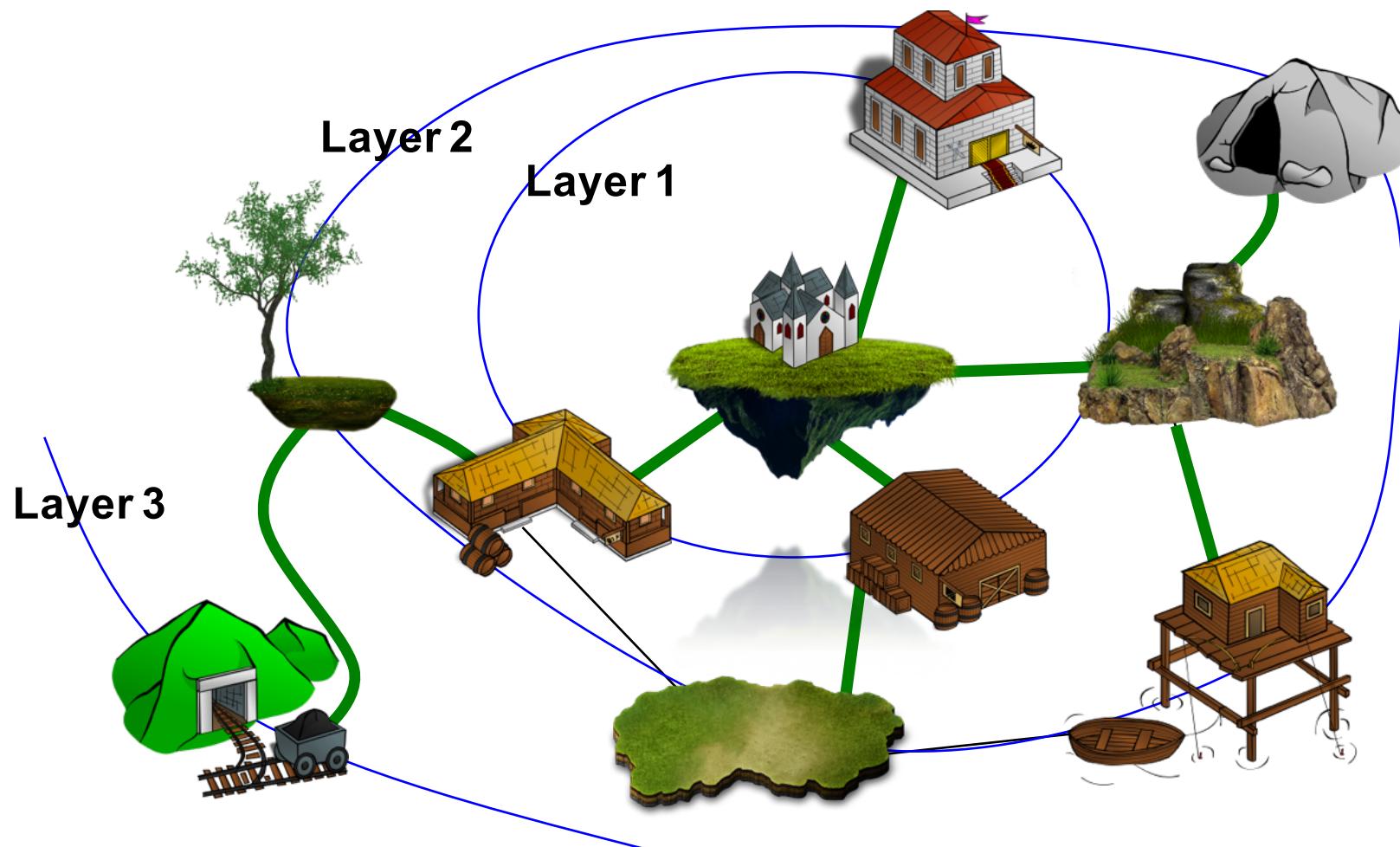
- $v.d = \text{distance}$ from s to v , for all $v \in V$
 - Distance is the length of a shortest path in G
 - $v.d = \infty$ if v is not reachable from s
 - $v.d$ is also the depth of v in T_{bfs}
- $v.\pi = u$ if (u, v) is the last edge on a shortest path to v
 - u is v 's predecessor in T_{bfs}

Breadth-first tree

Initially $T_{\text{ bfs}}$ contains only s

As v is discovered from u , v and (u,v) are added to $T_{\text{ bfs}}$

$T_{\text{ bfs}}$ is not explicitly stored; can be reconstructed from $v.\pi$



BFS algorithm

Implemented via a FIFO queue

Color the vertices to keep track of progress:

Gray: discovered (first time encountered)

Black: finished (all adjacent vertices discovered)

White: undiscovered

Q: a **queue** of discovered vertices

v.d: distance from s to v

v. π : predecessor of v

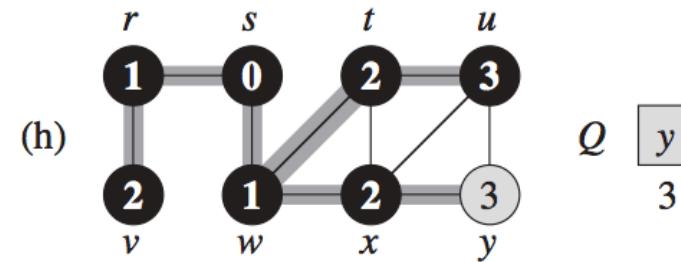
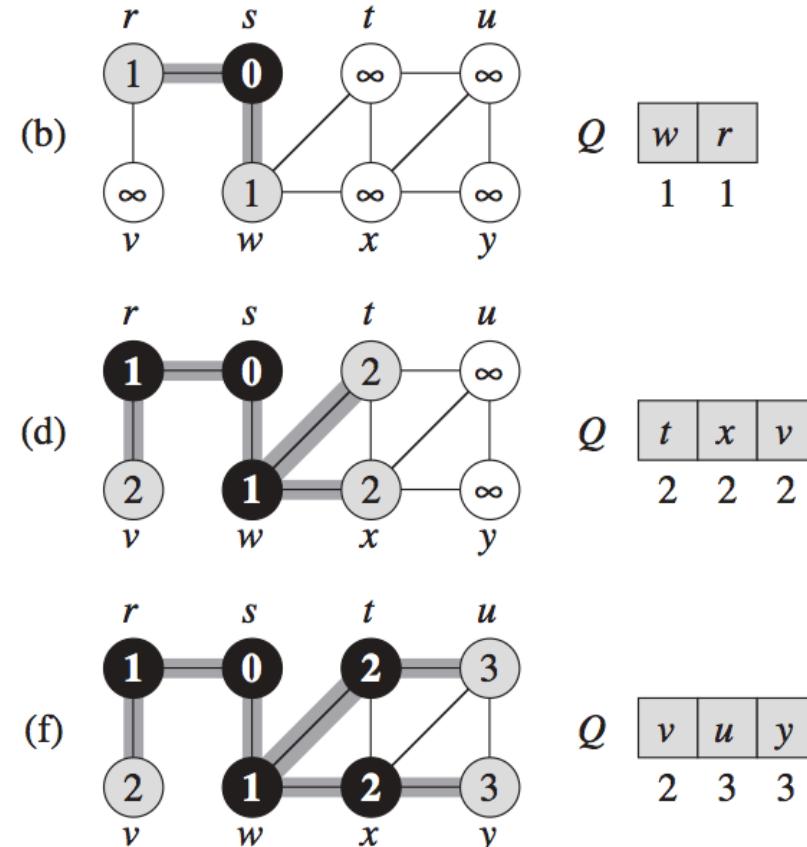
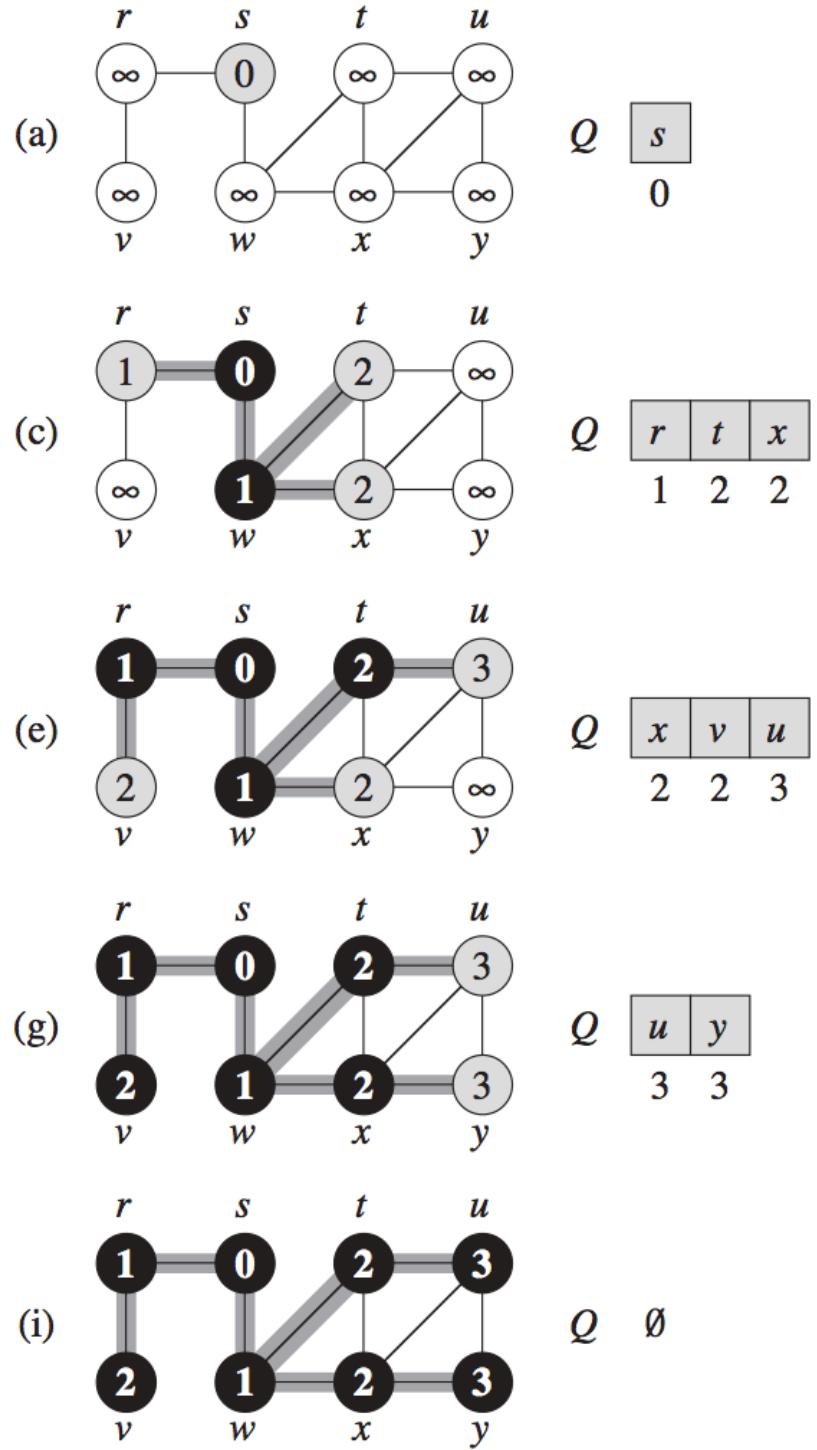
Running time using adjacency lists = ?

Running time using adjacency matrix= ?

Can we use two colors only?

BFS(G, s)

```
1  for each vertex  $u \in G.V - \{s\}$ 
2       $u.color = \text{WHITE}$ 
3       $u.d = \infty$ 
4       $u.\pi = \text{NIL}$ 
5       $s.color = \text{GRAY}$ 
6       $s.d = 0$ 
7       $s.\pi = \text{NIL}$ 
8       $Q = \emptyset$ 
9      ENQUEUE( $Q, s$ )
10     while  $Q \neq \emptyset$ 
11          $u = \text{DEQUEUE}(Q)$ 
12         for each  $v \in G.Adj[u]$ 
13             if  $v.color == \text{WHITE}$ 
14                  $v.color = \text{GRAY}$ 
15                  $v.d = u.d + 1$ 
16                  $v.\pi = u$ 
17                 ENQUEUE( $Q, v$ )
18          $u.color = \text{BLACK}$ 
```



BFS properties

$\text{BFS}(G,s)$ computes $v.d$ and $v.\pi$ such that:

1. $v.d = \text{distance}$ (smallest number of edges) from s to vertex v
2. The π attributes corresponds to a **breadth-first tree** (BFS tree) with root s that contains all reachable vertices
3. The simple path in the breadth-first tree from s to v corresponds to a shortest path from s to v in G

G can be either directed or undirected

Theorem 22.5 proves these properties are indeed true

Running time analysis

BFS runs in $O(V + E)$ time given adjacency-list representation

- Initialization takes $O(V)$
- Each vertex is enqueued and dequeued at most once.
Hence, the adjacency list of each vertex is scanned at most once, and the sum of lengths of all adjacency lists is $\Theta(E)$
- => total running time of BFS is $O(V+E)$, linear in the size of the adjacency list representation of graph

Correctness of BFS

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

Theorem 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:

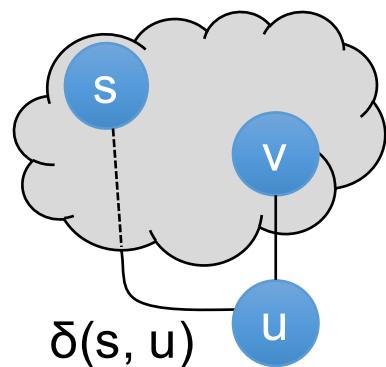
1. BFS discovers every vertex $v \in V$ that is reachable from the source s
2. Upon termination, $v.d = \delta(s, v)$ for all $v \in V$
3. For any vertex $v \neq s$ that is reachable from s , one of the shortest paths from s to v is a shortest path from s to $v.\pi$ followed by the edge $(v.\pi, v)$

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 $(u, v) \in E$, $\delta(s, v) \leq \delta(s, u) + 1$.

類似三角不等式

Proof of Lemma 22.1

Case 1: u is reachable from s



s - u - v is a path from s to v with length $\delta(s, u) + 1$. Hence, $\delta(s, v) \leq \delta(s, u) + 1$

Case 2: u is unreachable from s

$\delta(s, u) = \infty$
The inequality holds.

Lemma 22.2 Let $G = (V, E)$ be a directed or undirected graph, and suppose 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 $v.d \geq \delta(s, v)$.

證明BFS算出的d值一定會大於等於真正的距離

Proof of Lemma 22.2

Proof by induction on the number of ENQUEUE operations

Inductive hypothesis: $v.d \geq \delta(s, v)$ after n enqueue ops

- Holds when $n = 1$: s is in the queue and $v.d = \infty$ for all $v \in V \setminus \{s\}$
- After $n+1$ enqueue ops, consider a white vertex v that is discovered during the search from a vertex u

$$v.d = u.d + 1$$

$$\geq \delta(s, u) + 1 \text{ (by induction hypothesis)}$$

$$\geq \delta(s, v) \text{ (by Lemma 22.1)}$$

- Vertex v is never enqueued again so $v.d$ never changes again.

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 $i = 1, 2, \dots, r-1$.

換句話說，我們要證明：

- Q 中最後一個點的距離 $\leq Q$ 中第一個點的距離+1
- Q 中第*i*個點的距離 $\leq Q$ 中第*i*+1點的距離

Proof of Lemma 22.3

Proof by induction on the number of queue operations

When $Q = \langle s \rangle$, the lemma holds.

Consider two operations for the inductive step:

- Dequeue op: When $Q = \langle v_1, v_2, \dots, v_r \rangle$ and dequeue v_1
- Enqueue op: When $Q = \langle v_1, v_2, \dots, v_r \rangle$ and enqueue v_{r+1}

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 of Lemma 22.3 (cont'd)



By inductive hypothesis:

- (F1) Q 中最後一個點的距離 $\leq Q$ 中第一個點的距離 + 1
- (F2) Q 中第 i 個點的距離 $\leq Q$ 中第 $i+1$ 點的距離

After a dequeue op:



Prove that the following are still true after dequeue:

- (S1) Q' 中最後一個點的距離 $\leq Q'$ 中第一個點的距離 + 1?
- (S2) Q' 中第 i 個點的距離 $\leq Q'$ 中第 $i+1$ 點的距離?

- $v_1.d \leq v_2.d$ and $v_r.d \leq v_1.d + 1$ (by F1 and F2)
 - $\Rightarrow v_r.d \leq v_1.d + 1 \leq v_2.d + 1$
 - $\Rightarrow S1$ is true
- Also, $v_i.d \leq v_{i+1}$ for $i = 2, \dots, r-1$ (by F2)
 - $\Rightarrow S2$ is true

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 of Lemma 22.3 (cont'd)

Q	v_1	v_2	\dots	v_{r-1}	v_r	By inductive hypothesis: (F1) Q 中最後一個點的距離 $\leq Q$ 中第一個點的距離 + 1 (F2) Q 中第 <i>i</i> 個點的距離 $\leq Q$ 中第 <i>i</i> +1點的距離
After an enqueue op:	$v_1 \quad v_2 \quad \dots \quad \dots \quad v_r \quad v_{r+1}$					

Q'	v_1	v_2	\dots	\dots	v_r	v_{r+1}	Prove that the following are still true after dequeue: (S1) Q' 中最後一個點的距離 $\leq Q'$ 中第一個點的距離 + 1 (S2) Q' 中第 <i>i</i> 個點的距離 $\leq Q'$ 中第 <i>i</i> +1點的距離?
----	-------	-------	---------	---------	-------	-----------	---

- Let u be v_{r+1} 's predecessor, $v_{r+1}.d = u.d + 1$
- Since u has been removed from Q , the new head v_1 has $v_1.d \geq u.d$ (by F2)
- $\Rightarrow v_{r+1}.d = u.d + 1 \leq v_1.d + 1$
- $\Rightarrow S1$ is true
- $v_r.d \leq u.d + 1$ (by F1)
- $\Rightarrow v_r.d \leq u.d + 1 = v_{r+1}.d$
- Combining with $v_i.d \leq v_{i+1}$ for $i = 1, 2, \dots, r-1$ (by F2)
- $\Rightarrow v_i.d \leq v_{i+1}$ for $i = 1, 2, \dots, r$
- $\Rightarrow S2$ is true

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 enqueueued.

證明若 v_i 比 v_j 早加入queue， $v_i.d \leq v_j.d$

Proof of Corollary 22.4

Lemma 22.3證明了Q中第*i*個點的距離 \leq Q中第*i+1*點的距離

Also, each vertex receives a finite d value at most once during the course of BFS

=> 得証

Correctness of BFS

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:

1. BFS discovers every vertex $v \in V$ that is reachable from the source s
2. Upon termination, $v.d = \delta(s, v)$ for all $v \in V$
3. 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 $(v.\pi, v)$

Prove that $v.d = \delta(s, v)$ for all $v \in V$:

By contradiction, assume 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 $v.d > \delta(s, v)$

- Vertex v must be reachable from s ; otherwise $\delta(s, v) = \infty \geq v.d$.

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 $u.d = \delta(s, u)$

$$\Rightarrow v.d > \delta(s, v) = \delta(s, u) + 1 = u.d + 1 \text{ (Eq. 1)}$$

Prove that $v.d = \delta(s, v)$ for all $v \in V$ (cont'd):

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

When dequeuing vertex u from Q , vertex v is either white, gray, or black

- If v is white, then $v.d = u.d + 1$, contradicting (Eq. 1)
- If v is black, then it was already removed from the queue
 - By Corollary 22.4, we have $v.d \leq u.d$, contradicting (Eq. 1)
- If v is gray, then it was painted gray upon dequeuing some vertex w
 - Thus $v.d = w.d + 1$
 - Also, because w was removed from Q earlier than u , $w.d \leq u.d$ (By Corollary 22.4)
 - $\Rightarrow v.d = w.d + 1 \leq u.d + 1$, contradicting (Eq. 1)

Thus $v.d = \delta(s, v)$ for all v in V .

Prove that BFS discovers every vertex $v \in V$ that is
reachable from the source s :

All vertices v reachable from s must be discovered;
otherwise they would have $\infty = v.d > \delta(s, v)$.

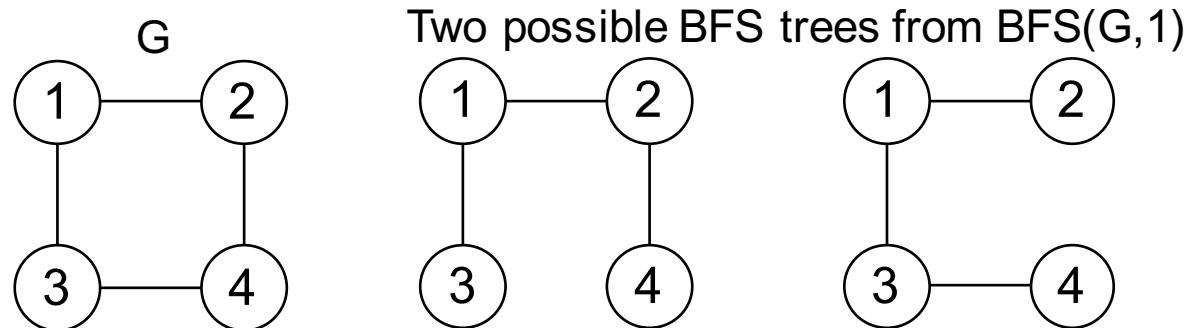
Prove that for any vertex $v \neq s$ that is reachable from s ,
one of the shortest paths from s to v is a shortest path
from s to $v.\pi$ followed by the edge $(v.\pi, v)$:

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 from
 s to $v.\pi$ and then traversing the edge $(v.\pi, v)$

BFS tree

Is the BFS tree unique?

No. The BFS algorithm does not specify the order in which the neighbors of a vertex are visited. The resulting BFS tree may depend on the visit order.



Will the algorithm output different v.d (the distance of v from s) under different visiting orders of neighbors?

No, because v.d is the length of a shortest path on G . While there can be multiple shortest paths between s and v , their lengths should be equal.

BFS forest

$\text{BFS}(G,s)$ forms a **BFS tree** with all reachable v from s

We can extend the algorithm to find a **BFS forest** that contains every vertex in G

```
//explore full graph and  
builds up a collection of  
BFS trees  
BFS(G)  
for u in G.V  
    u.color = WHITE  
    u.d = ∞  
    u. π = NIL  
for s in G.V  
    if(s.color == WHITE)  
        //build a BFS tree  
        BFS-Visit(G,s)
```

```
BFS-Visit(G,s)  
s.color = GRAY  
s.d = 0  
s. π = NIL  
Q = empty  
ENQUEUE(Q, s)  
while Q ≠ empty  
    u = DEQUEUE(Q)  
    for v in G.adj[u]  
        if v.color == WHITE  
            v.color = GRAY  
            v.d = u.d + 1  
            v. π = u  
            ENQUEUE(Q, v)  
    u.color = BLACK
```

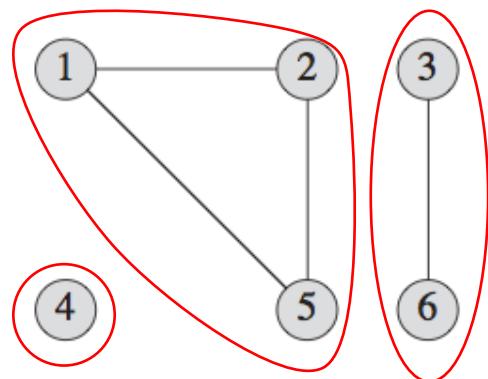
BFS applications

Find a shortest path between two vertices u and v

- Path length = number of edges

Find connected components

The **connected components** of an **undirected graph** are the equivalence classes of vertices under the “is reachable from” relation.



3 connected components: $\{1,2,5\}$, $\{3,6\}$, $\{4\}$
(regardless of the visiting order of neighbors)

A related concept: the **strongly connected components** of a **directed graph** are the equivalence classes of vertices under the “mutually reachable” relation. (DFS can be used to find strongly connected components)

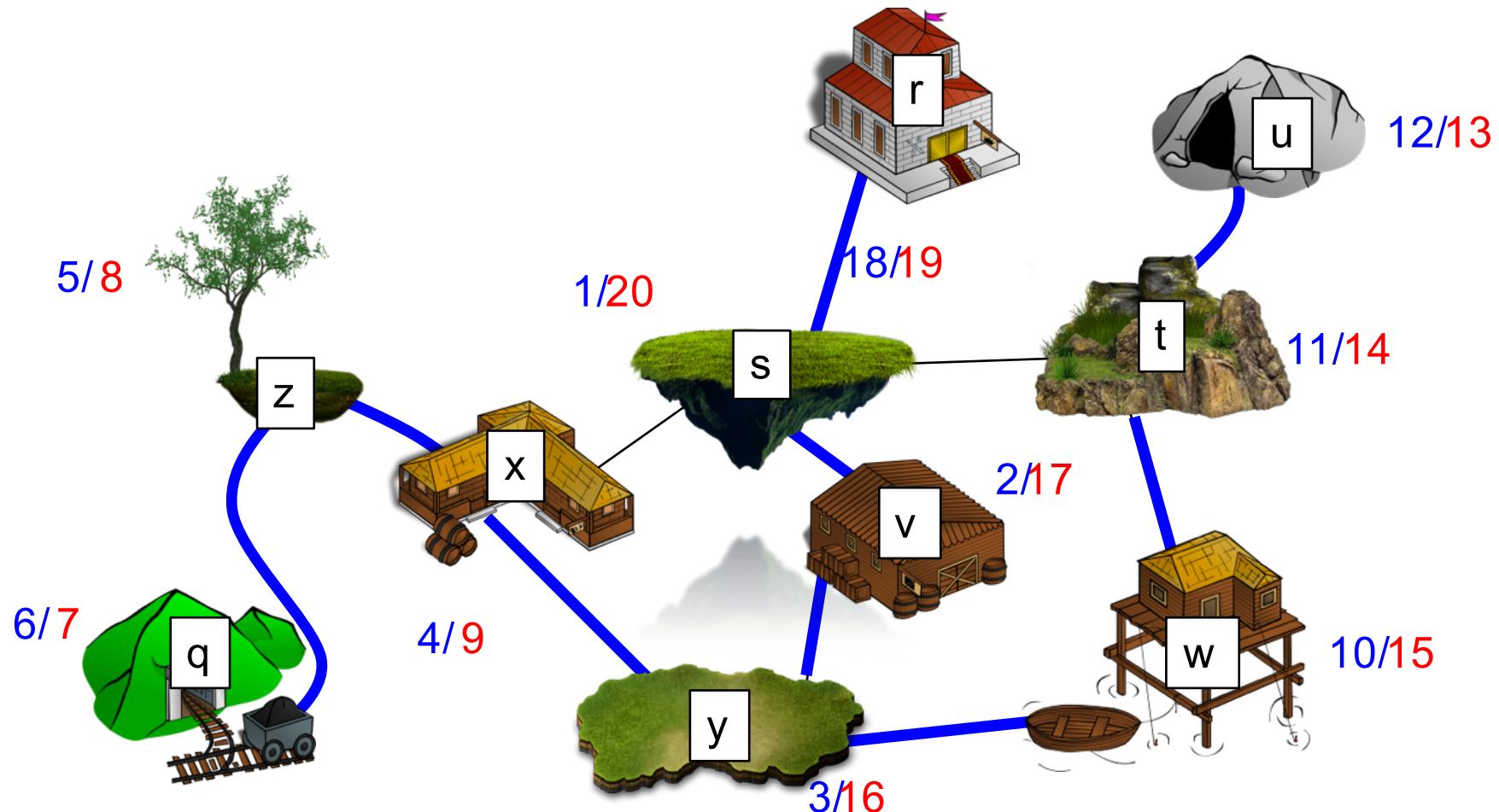
Depth-First Search

Chapter 22.3

Depth-first search (DFS)

Intuition. Search as deep as possible first, then backtrack until finding a new path to go down.

Timestamps: Discovery time / finishing time



Depth-first search (DFS)

Input: directed or undirected graph $G = (V, E)$ and source vertex s

Output: a **depth-first tree** with root s that contains all reachable vertices from s

For each v , keep two timestamps and the predecessor:

- $v.d$ = discovery time
- $v.f$ = finishing time
- $v.\pi$ = the predecessor of v in the DFS tree
- Like in BFS, we could also construct a **DFS forest** given a graph. In the following, we consider a DFS algorithm to explore the full graph and produce a DFS forest.

Is the simple path between u and v in a DFS tree = a shortest path between u and v in G ?

DFS algorithm

Implemented via recursion (stack)

Coloring vertices for progress tracking:

- **Gray**: discovered (first time encountered)
- **Black**: finished (all adjacent vertices discovered)
- **White**: undiscovered

v.d: discovery time

v.f: finishing time

v. π : predecessor

```
//Explore full graph and builds
up a collection of DFS trees
DFS(G)
for u in G.V
    u.color = WHITE
    u.  $\pi$  = NIL
time = 0 //global timestamp
for u in G.V
    if u.color == WHITE
        DFS-Visit(G,u)
```

```
DFS-Visit(G,u)
time = time + 1
u.d = time //discovery time
u.color = GRAY
for v in G.adj[u]
    if v.color == WHITE
        v.  $\pi$  = u
        DFS-Visit(G,v)
u.color = BLACK
time = time + 1
u.f = time //finishing time
```

Running time analysis

DFS runs in $\Theta(V + E)$ time given adjacency-list representation

- Initialization takes $\Theta(V)$
- For each vertex u , $\text{DFS-Visit}(G, u)$ is called exactly once
- Excluding the recursive part, $\text{DFS-Visit}(G, u)$ takes $\Theta(\text{degree}(u))$ time
- \Rightarrow total running time of BFS is $\Theta(V+E)$, linear in the size of the adjacency list representation of graph

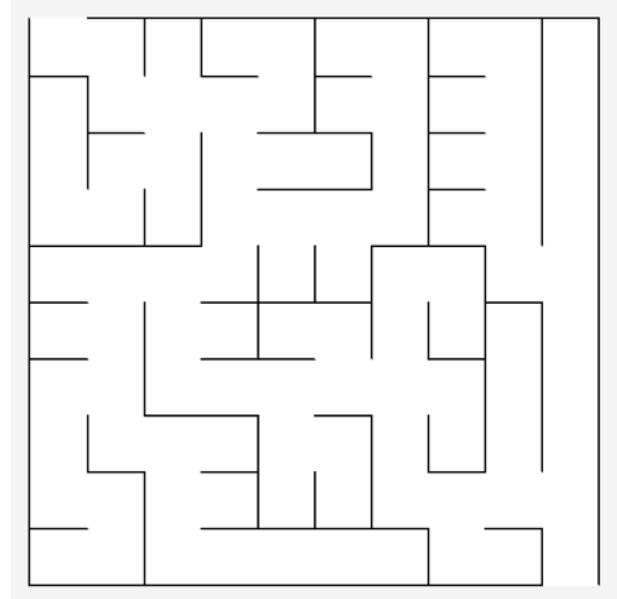
DFS applications

Finding connected components (like BFS)

Topological sorting (next topic)

Generating or solving mazes

Solving a maze using BFS? (Dis)advantage?



Properties of depth-first search

Parenthesis Theorem (括號定理)

- 對任一個 vertex u , 把發現 u 用 “ (u) ” 表示, 結束探索 u 用 “ $u)$ ” 表示, 那麼在所形成的 expression 中, 左括號和相應的右括號一定是匹配的。

White Path Theorem (白路徑定理)

- Vertex v 是 vertex u 的子孫節點 \Leftrightarrow 當 u 被發現時, u 和 v 之間存在一條路徑, 路徑上的點皆為白節點。

Classification of edges in G

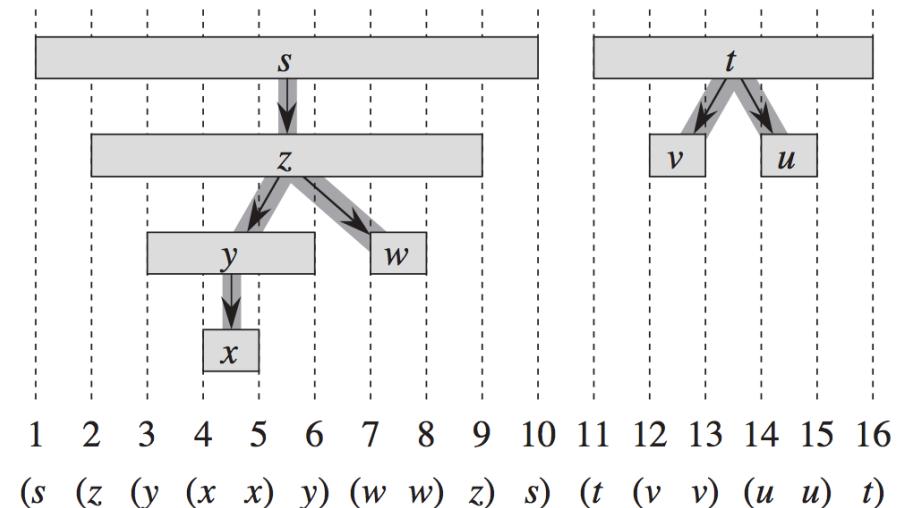
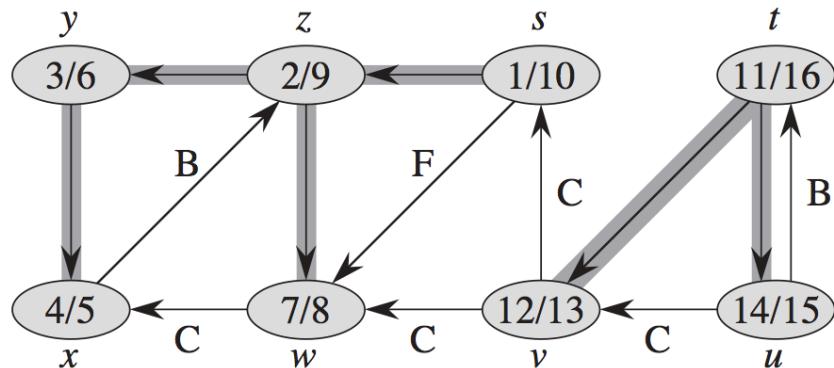
1. Tree edge : 樹林裡的邊。
2. Back edge : 連向祖先。
3. Forward edge : 連向子孫。
4. Cross edge : 枝葉之間的邊、樹之間的邊。

Parenthesis Theorem

Parenthesis structure: represent the discovery of vertex u with a left parenthesis “(u” and represent its finishing by a right parenthesis “u)”. In DFS, the parentheses are properly nested.

Properly nested: $(x (y y) x)$

Not properly nested: $(x (y x) y)$



Parenthesis Theorem

Theorem 22.7 Formally, for any u, v , exactly one of following holds:

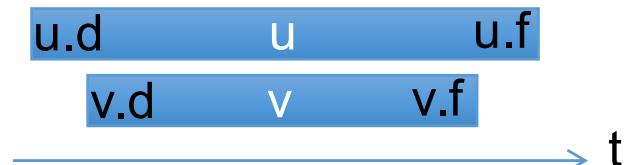
1. The intervals $[u.d, u.f]$ and $[v.d, v.f]$ are entirely disjoint, and neither u nor v is a descendant of the other in the DFS forest
2. The interval $[u.d, u.f]$ is contained entirely within the interval $[v.d, v.f]$, and u is a descendant of v in a DFS tree, or
3. The interval $[v.d, v.f]$ is contained entirely within the interval $[u.d, u.f]$, and u is a descendant of v in a DFS tree.

Disjoint interval example:

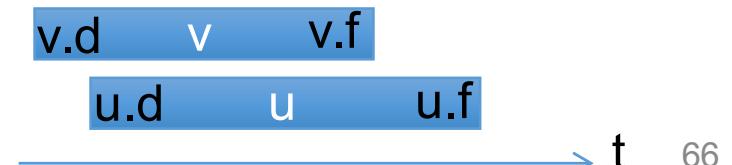
Proof: textbook p.608.



v 's interval is contained entirely in v 's



An impossible case:

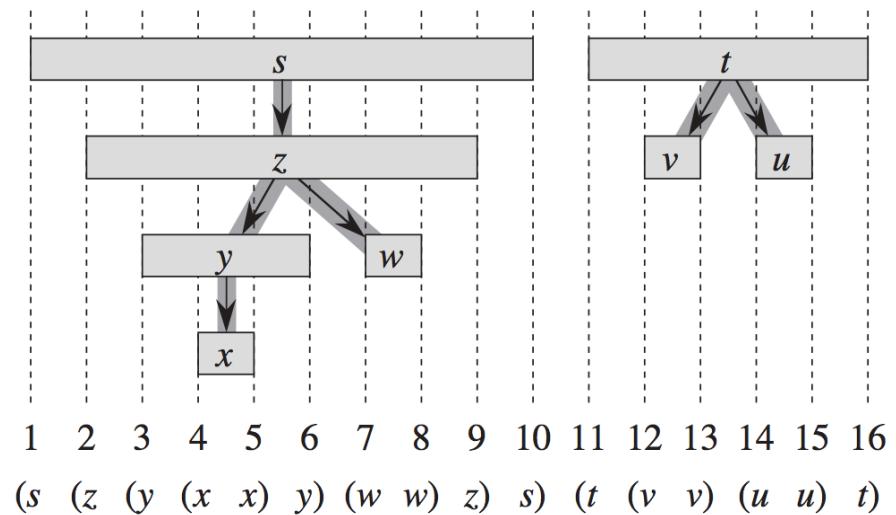
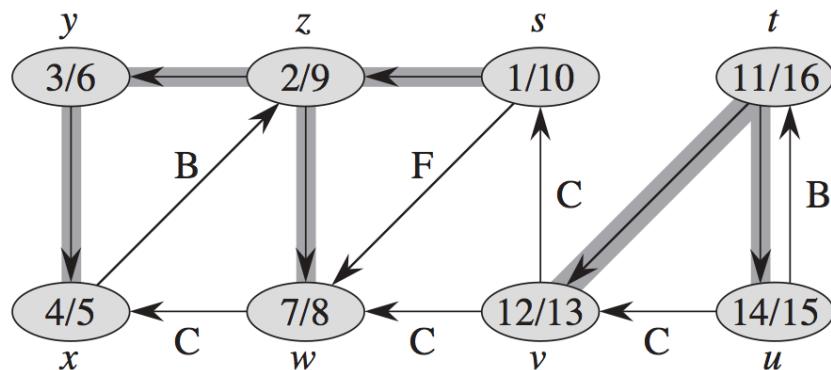


Nesting of descendants' intervals

Following the Parenthesis Theorem, we get

v is a proper descendant of u in the DFS forest

$$\Leftrightarrow u.d < v.d < v.f < u.f$$



White-path Theorem

Theorem 22.9 In a DFS forest of a directed or undirected graph $G = (V, E)$,

vertex v is a descendant of vertex u

\Leftrightarrow at the time $u.d$ that the search discovers u , there is a path from u to v consisting entirely of white vertices.

Proof

Left to right direction (\Rightarrow):

- By Parenthesis Theorem, $u.d < v.d$
- Hence, v is WHITE at time $u.d$
- In fact, since v can be any descendant of u , any vertex on the path from u to v are WHITE at time $u.d$

Right to left direction (\Leftarrow): See textbook p.608.

Classification of edges

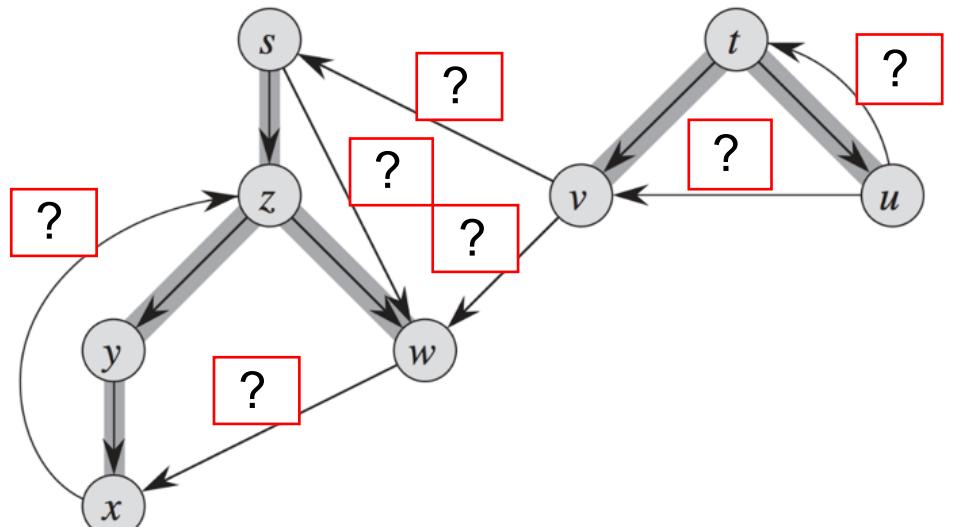
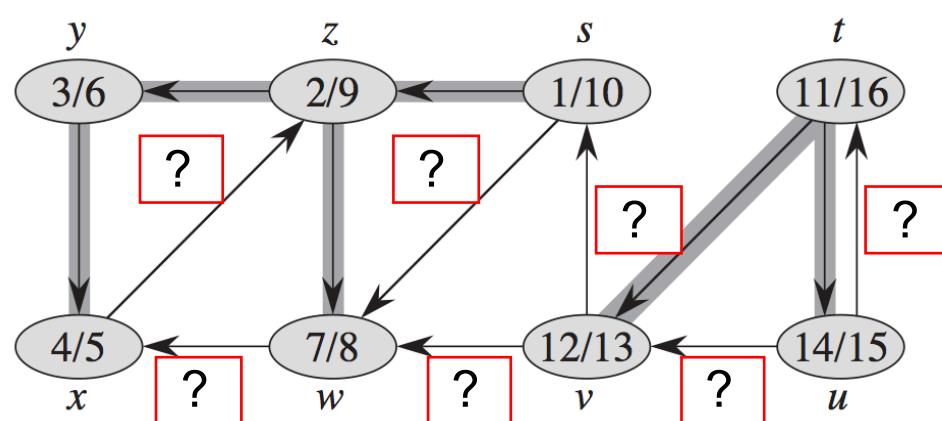
DFS introduces an important distinction among edges in the original graph:

1. Tree edge (GRAY to WHITE)

- Edges in the DFS forest. Found when encountering a new vertex v by exploring (u, v)

2. Back edge (GRAY to GRAY)

- (u, v) , from descendant u to ancestor v in a DFS tree



Classification of edges (cont'd)

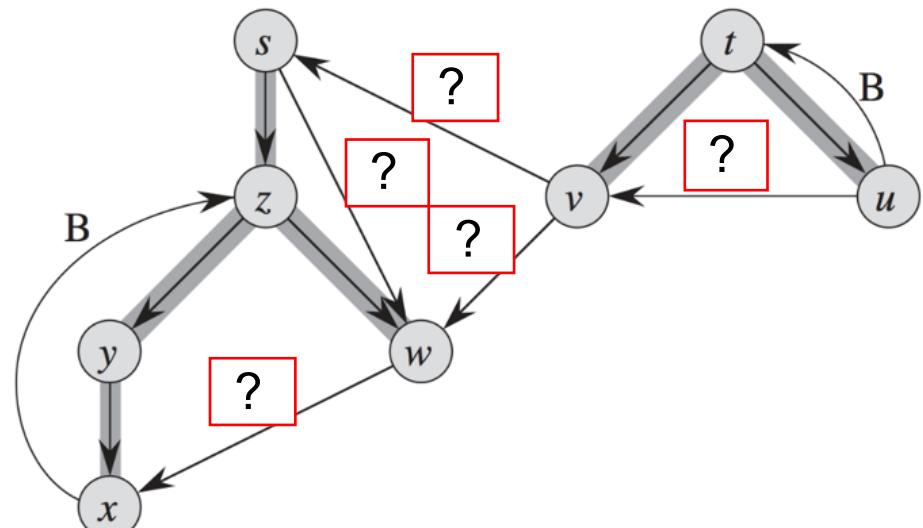
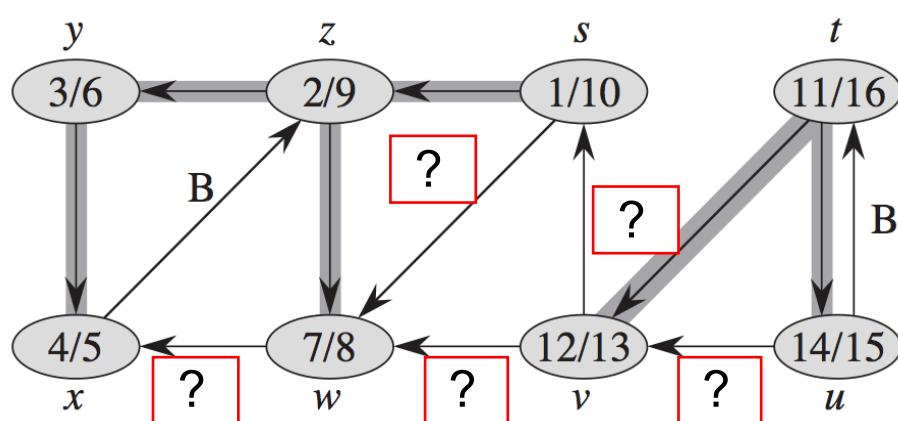
3. Forward edge (GRAY to BLACK)

- (u, v) , from ancestor u to descendant v . Not a tree edge.

4. Cross edge (GRAY to BLACK)

- Any other edge between trees or subtrees. Can go between vertices in same DFS tree or in different DFS trees

In an undirected graph, back edge = forward edge. To avoid ambiguity, classify edge as the first type in the list that applies.



Classification of edges

Theorem 22.10 In DFS of an undirected graph, we get only tree and back edges. No forward or cross edges.

Proof

No forward edge by definition

Consider any edge (u, v)

WLOG, suppose $u.d < v.d$. Thus, u is discovered (GRAY) while v is still WHITE

If (u, v) is first explored in the direction from u to v , then (u, v) is a tree edge

If (u, v) is first explored in the direction from v to u , then (u, v) is a back edge