# The Design and Analysis of Algorithms

## Lecture 5 Graphs II

Zhenbo Wang

Department of Mathematical Sciences, Tsinghua University

# Content

Connectivity in Directed Graphs

DAGs and Topological Ordering

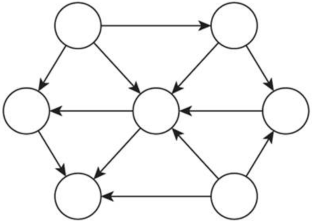
Coin Changing

# Directed Graphs

- *Directed graph*. $G = (V, E)$.

  Edge $(u, v)$ goes from node $u$ to node $v$.

- Web graph - hyperlink points from one web page to another.
- Directedness of graph is crucial.
- Modern web search engines exploit hyperlink structure to rank web pages by importance.



# Road Network

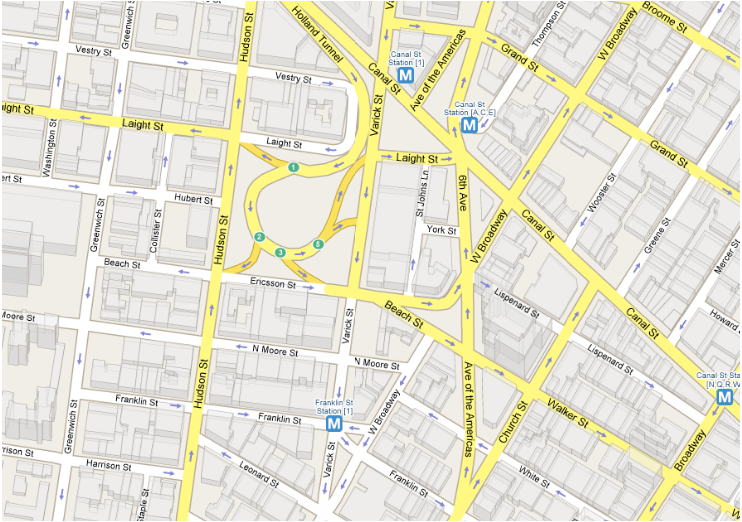



Figure 1: Vertex = intersection; edge = one-way street.

# Ecological Food Web

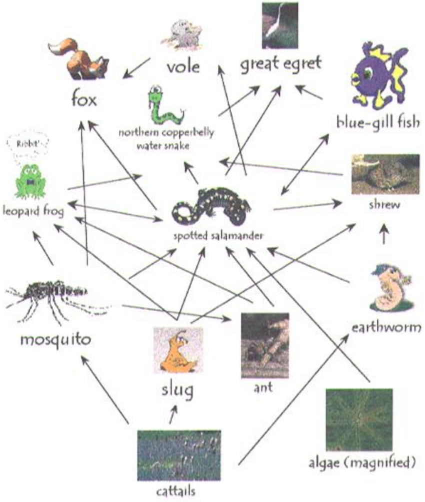



Figure 2: Vertex =species; edge = from prey to predator.

# Some Directed Graph Applications

| graph | node | edge |
|---|---|---|
| transportation | street intersection | one-way street |
| web | web page | hyperlink |
| food web | species | predator-prey relationship |
| scheduling | task | precedence constraint |
| financial | bank | transaction |
| cell phone | person | call |
| citation | journal article | citation |
| ... | ... | ... |

# Graph Search

- *Directed reachability*. Given a node $s$, find all nodes reachable from $s$.
- *Directed $s - t$ shortest path problem*. Given two node $s$ and $t$, what is the length of the shortest path from $s$ and $t$?
- *Graph search*. BFS extends naturally to directed graphs.
- *Web crawler*. Start from web page $s$. Find all web pages linked from $s$, either directly or indirectly.

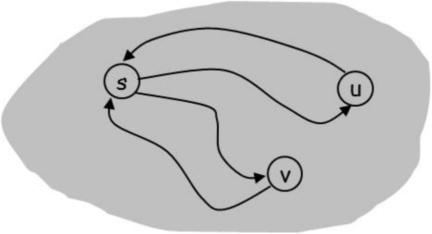
# Strong Connectivity

**Def.** Nodes $u$ and $v$ are mutually reachable if there is a both path from $u$ to $v$ and also a path from $v$ to $u$.

**Def.** A graph is strongly connected if every pair of nodes is mutually reachable.

**Lemma 1**

*Let $s$ be any node. $G$ is strongly connected iff every node is reachable from $s$, and $s$ is reachable from every node.*



# Strong Connectivity: Algorithm

### Theorem 2

*Can determine if G is strongly connected in $O(m+n)$ time.*

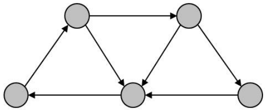
Pf. Pick any node $s$.

Run BFS from $s$ in $G$.

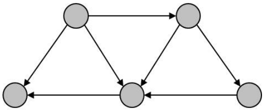
Run BFS from $s$ in $G^{rev}$ (reverse every edge in $G$).

Return true iff all nodes reached in both BFS executions.

Correctness follows immediately from previous lemma. □



strongly connected



not strongly connected

# Strong Components

Def. A strong component is a maximal subset of mutually reachable nodes.

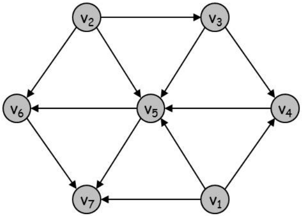
Theorem 3 (Tarjan 1972)

*Can find all strong components in $O(m + n)$ time.*

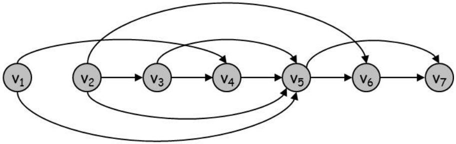
# Directed Acyclic Graphs

Def. A *DAG* is a directed graph that contains no directed cycles.

Def. A *topological order* of a directed graph $G = (V, E)$ is an ordering of its nodes as $v_1, v_2, \cdots, v_n$ so that for every edge $(v_i, v_j)$ we have $i < j$.




a DAG




a topological ordering

# Precedence Constraints

- *Precedence constraints*. Edge $(v_i, v_j)$ means task $v_i$ must occur before $v_j$.
- *Applications*.

  Course prerequisite graph: course $v_i$ must be taken before $v_j$.

  Compilation: module $v_i$ must be compiled before $v_j$.

  Computing jobs: output of job $v_i$ needed to determine input of job $v_j$.

# Topological Ordering

**Lemma 4**

*If G has a topological order, then G is a DAG.*

Pf. [by contradiction]

Suppose that $G$ has a topological order $v_1, v_2, \cdots, v_n$ and that $G$ also has a directed cycle $C$.

Let $v_i$ be the lowest-indexed node in $C$, and let $v_j$ be the node just before $v_i$.

Thus $(v_j, v_i)$ is an edge, and $j < i$.

By our choice of $i$, we have $i < j$, a contradiction. $\square$

# Topological Ordering

## Lemma 5

*If G is a DAG, then G has a node with no entering edges.*

## Lemma 6

*If G is a DAG, then G has a topological order.*

Pf. [by induction on $n$]

Base case: true if $n = 1$.

Given DAG on $n > 1$ nodes, find a node $v$ with no incoming edges.

$G - \{v\}$ is a DAG.

By inductive hypothesis, $G - \{v\}$ has a topological ordering.

Place $v$ first in topological ordering; then append nodes of $G - \{v\}$ in topological order.

This is valid since $v$ has no incoming edges. □

# Compute a Topological Ordering of $G$:

1: Find a node $v$ with no incoming edges and order it first.
2: Delete $v$ from $G$.
3: Recursively computer a topological ordering of $G - \{v\}$ and append this order after $v$.

**Theorem 7**

*The algorithm finds a topological order in $O(m+n)$ time.*

Pf. $count(w)$ = remaining number of incoming edges of node $w$;
$S$ = set of remaining nodes with no incoming edges.

Initialization: $O(m+n)$ via single scan through graph.

Update: to delete $v$,

remove $v$ from $S$;

decrement $count(w)$ for all edges from $v$ to $w$;

and add $w$ to $S$ if $count(w)$ hits 0.

this is $O(1)$ per edge. □

# Coin Changing

Goal. Given currency denominations: 1 (penny), 5 (nickel), 10 (dime), 25 (quarter), 100 (dollar), devise a method to pay amount to customer using fewest number of coins.
Ex. 34c.

- *Cashier's algorithm*. At each iteration, add coin of the largest value that does not take us past the amount to be paid.
Ex. \$2.89.

# Cashier's algorithm

$CASHIERS - ALGORITHM(x, c_1, \cdots, c_n)$

```
SORT n coin denominations so that c_1 < c_2 < ··· < c_n.
S ← ∅
while x > 0 do
    k ← largest coin denomination c_k such that c_k ≤ x.
    if no such k then
        return "no solution".
    else
        x ← x − c_k, S ← S ∪ {k}.
    end if
end while
return S.
```

Q. Is cashier's algorithm optimal?

# Properties of Optimal Solution

- *Property*. Number of pennies $\leq 4$.

Pf. Replace 5 pennies with 1 nickel.

- *Property*. Number of nickels $\leq 1$.
- *Property*. Number of quarters $\leq 3$.
- *Property*. Number of nickels $+$ number of dimes $\leq 2$.

Pf.

Recall: at most 1 nickel.

Replace 3 dimes and 0 nickels with 1 quarter and 1 nickel;

Replace 2 dimes and 1 nickel with 1 quarter. $\square$

# Analysis of Cashier's Algorithm

**Theorem 8**

*Cashier's algorithm is optimal for U.S. coins: 1, 5, 10, 25, 100.*

Pf. [by induction on $x$]

Let $k$ be the largest coin denomination $c_k$ such that $c_k \leq x$: greedy takes coin $k$.

Claim: Any optimal solution must also take coin $k$.

If not, it needs enough coins of type $c_1, \cdots, c_{k-1}$ to add up to $x$.

# Proof-Con't

| $k$ | $c_k$ | property | max value |
|---|---|---|---|
| 1 | 1 | $P \leq 4$ | - |
| 2 | 5 | $N \leq 1$ | 4 |
| 3 | 10 | $N + D \leq 2$ | $4 + 5 = 9$ |
| 4 | 25 | $Q \leq 3$ | $20 + 4 = 24$ |
| 5 | 100 | no limit | $75 + 24 = 99$ |

Table 1: property and max value of $c_1, \cdots, c_{k-1}$ in any optimum

No optimal solution can do this.

Coin-changing $x - c_k$ cents, by induction, is optimally solved by cashier's algorithm. $\square$

# Cashier's Algorithm for Other Denominations

Q. Is cashier's algorithm for any set of denominations?

A. No. Consider U.S. postage: 1, 10, 21, 34, 70, 100, 350, 1225, 1500.

Cashier's algorithm: $140c = 100 + 34 + 1 + 1 + 1 + 1 + 1 + 1$.

Optimal: $140c = 70 + 70$.


A. No. It may not even lead to a feasible solution if $c_1 > 1$: 7, 8, 9.

Cashier's algorithm: $15c = 9 + ???$

# Homework

- Read Chapter 3 of the textbook.
- Exercises 2 & 8 in Chapter 3.