Directed Graphs

- introduction
- digraph search
- transitive closure
- topological sort
- strong components

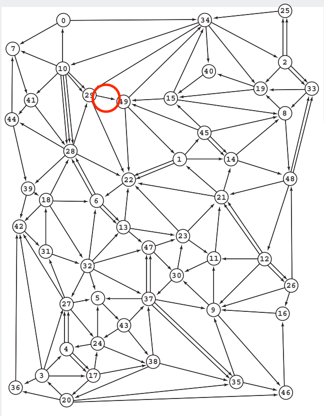
References: Algorithms in Java (Part 5), Chapter 19
Intro to Algs and Data Structures, Section 5.2

Directed Graphs

Digraph. Set of objects with oriented pairwise connections.

one-way streets in a map

hyperlinks connecting web pages

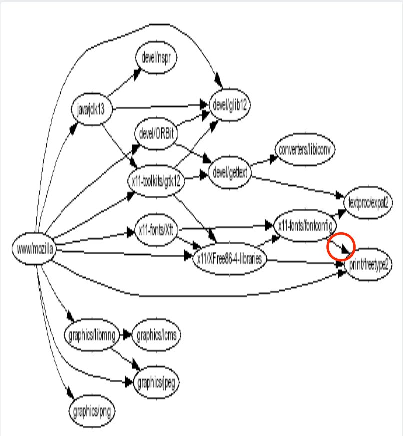


introduction
digraph search
transitive closure
topological sort
strong components

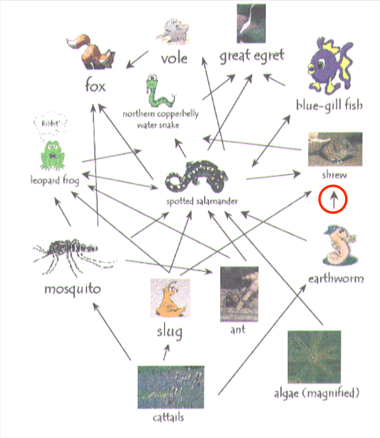
Directed graphs (digraphs)

Set of objects with oriented pairwise connections.

dependencies in software modules



prey-predator relationship among species



Digraph Applications

Digraph	Vertex	Edge
financial	stock, currency	transaction
transportation	street intersection, airport	highway, airway route
scheduling	task	precedence constraint
WordNet	synset	hypernym
Web	web page	hyperlink
game	board position	legal move
telephone	person	placed call
food web	species	predator-prey relation
infectious disease	person	infection
citation	journal article	citation
object graph	object	pointer
inheritance hierarchy	class	inherits from
control flow	code block	jump

Some Digraph Problems

Transitive closure. Is there a directed path from `v` to `w`?

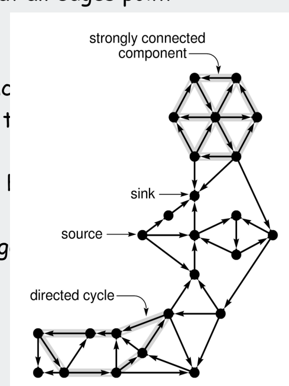
Strong connectivity. Are all vertices mutually reachable?

Topological sort. Can you draw the graph so that all edges point from left to right?

PERT/CPM. Given a set of tasks with precedenc
what is the earliest that we can complete each t

Shortest path. Given a weighted digraph, find l

PageRank. What is the importance of a web pag

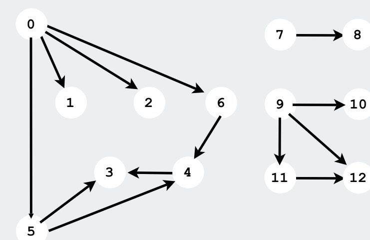


Digraph representation

Vertex names.

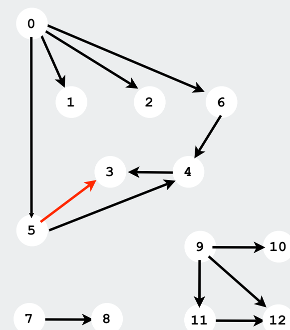
- This lecture: use integers between `0` and `V-1`.
- Real world: convert between names and integers with symbol table.

Orientation of edge is significant.



Adjacency Matrix Representation

Maintain a two-dimensional `V` × `V` boolean array.
For each edge `v→w` in graph: `adj[v][w] = true`.



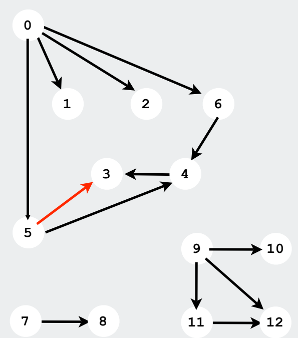
to

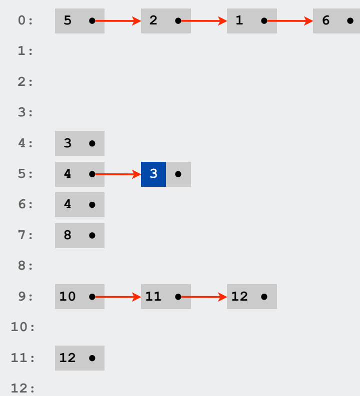
from	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	1	1	0	0	1	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	1	0	0	0	0	0	0	0	0	0
5	0	0	0	1	1	0	0	0	0	0	0	0	0
6	0	0	0	0	1	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	1	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0	1	1	1
10	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	1
12	0	0	0	0	0	0	0	0	0	0	0	0	0

Adjacency-list digraph representation

Maintain vertex-indexed array of lists.

Note: one representation of each directed edge





Adjacency-list digraph representation: Java implementation

Same as **Graph**, but only insert one copy of each edge.

```
public class Digraph
{
   private int V;
   private SET<Integer>[] adj;

   public Digraph(int V)
   {
      this.V = V;
      adj = (SET<Integer>[]) new SET[V];
      for (int v = 0; v < V; v++)
         adj[v] = new SET<Integer>();
   }

   public void addEdge(int v, int w)
   {
      adj[v].add(w);
   }

   public Iterable<Integer> adj(int v)
   {
      return adj[v];
   }
}
```

adjacency lists

create empty V-vertex graph

add edge from v to w (parallel edges allowed)

iterable SET for v's neighbors

Digraph Representations

Digraphs are abstract mathematical objects.

- ADT implementation requires specific representation.
- Efficiency depends on matching algorithms to representations.

Representation	Space	Edge from v to w?	Iterate over edges leaving v?
List of edges	E + V	E	E
Adjacency matrix	V^2	1	V
Adjacency list	E + V	outdegree(v)	outdegree(v)

In practice: Use adjacency-list representation

- Bottleneck is iterating over edges leaving v.
- Real world digraphs are sparse.

E is proportional to V

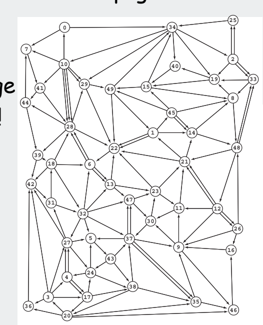
Typical digraph application: Google's PageRank algorithm

Goal. Determine which web pages on Internet are important.
Solution. Ignore keywords and content, focus on hyperlink structure.

Random surfer model.

- Start at random page.
- With probability 0.85, randomly select a hyperlink to visit next; with probability 0.15, randomly select any page.
- PageRank = proportion of time random surfer spends on each page.

Solution 1: Simulate random surfer for a long time.
Solution 2: Compute ranks directly until they converge
Solution 3: Compute eigenvalues of adjacency matrix!



None feasible without sparse digraph representation

Every square matrix is a weighted digraph introduction

digraph search

transitive closure

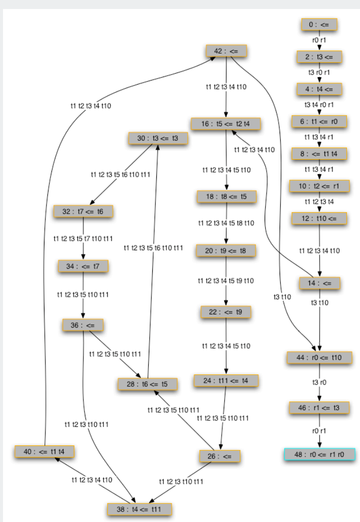
topological sort

strong components

pagerank

Digraph application: program control-flow analysis

Every program is a digraph (instructions connected to possible successors)



Dead code elimination.
Find (and remove) unreachable code

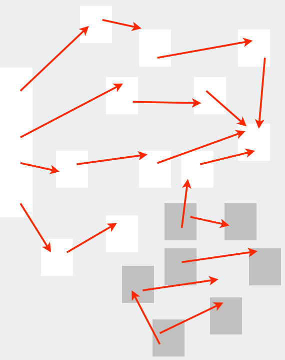
can arise from compiler optimization (or bad code)

Infinite loop detection.
Determine whether exit is unreachable

can't detect all possible infinite loops (halting problem)

Digraph application: mark-sweep garbage collector

Every data structure is a digraph (objects connected by references)



Roots. Objects known to be directly accessible by program (e.g., stack).

Reachable objects. Objects indirectly accessible by program (starting at a root and following a chain of pointers).

easy to identify pointers in type-safe language

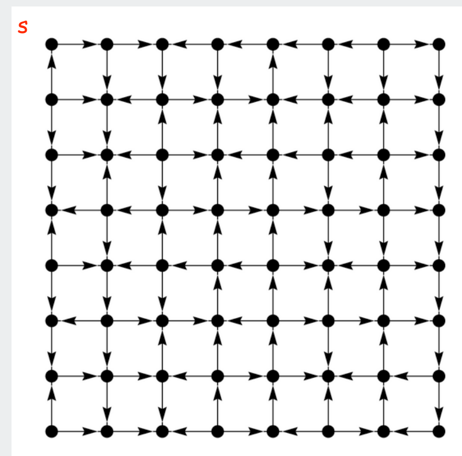
Mark-sweep algorithm. [McCarthy, 1960]

- Mark: run DFS from roots to mark reachable objects.
- Sweep: if object is unmarked, it is garbage, so add to free list.

Memory cost: Uses 1 extra mark bit per object, plus DFS stack.

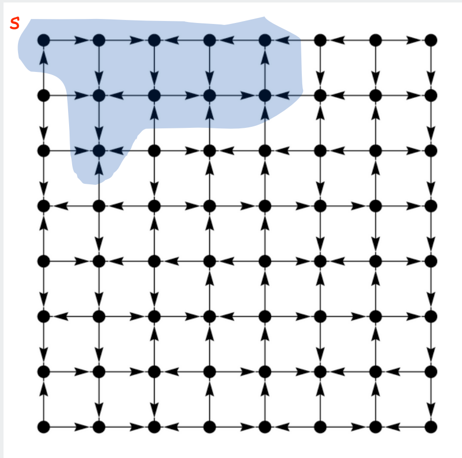
Reachability

Goal. Find all vertices reachable from s along a directed path.



Reachability

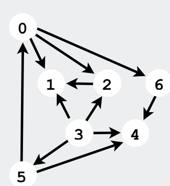
Goal. Find all vertices reachable from s along a directed path.



Digraph-processing challenge 1:

Problem: Mark all vertices reachable from a given vertex.

How difficult?
1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows



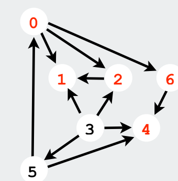
0-1
0-6
0-2
3-4
3-2
5-4
5-0
3-5
2-1
6-4
3-1

Digraph-processing challenge 1:

Problem: Mark all vertices reachable from a given vertex.

Use DFS

How difficult?
✓ 1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows



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

Depth-first search in digraphs

Same method as for undirected graphs

Every undirected graph is a digraph
- happens to have edges in both directions
- DFS is a digraph algorithm (never uses that fact)

DFS (to visit a vertex s)

Mark s as visited.
Visit all unmarked vertices w adjacent to v.

recursive

Depth-first search (single-source reachability)

Identical to undirected version (substitute `Digraph` for `Graph`).

```
public class DFSearcher
{
   private boolean[] marked;

   public DFSearcher(Digraph G, int s)
   {
      marked = new boolean[G.V()];
      dfs(G, s);
   }

   private void dfs(Digraph G, int v)
   {
      marked[v] = true;
      for (int w : G.adj(v))
         if (!marked[w]) dfs(G, w);
   }

   public boolean isReachable(int v)
   {
      return marked[v];
   }
}
```

true if connected to s

constructor marks vertices connected to s

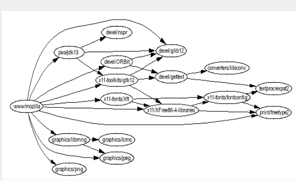
recursive DFS does the work

client can ask whether any vertex is connected to s

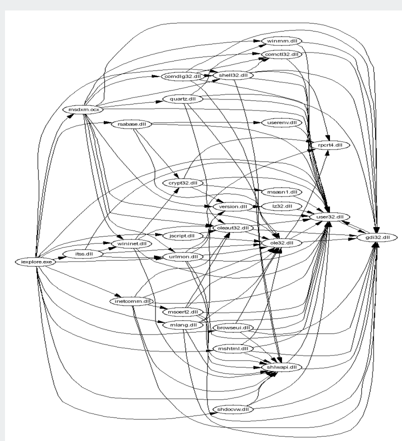
Digraph application: dependencies among software modules

Every software system is a digraph (modules dependent on others)

Mozilla



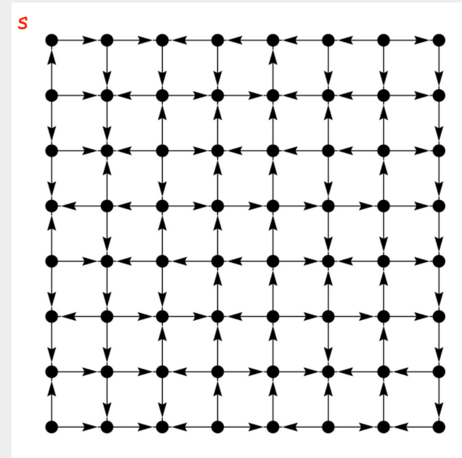
Internet explorer



Issue: Any cycles?

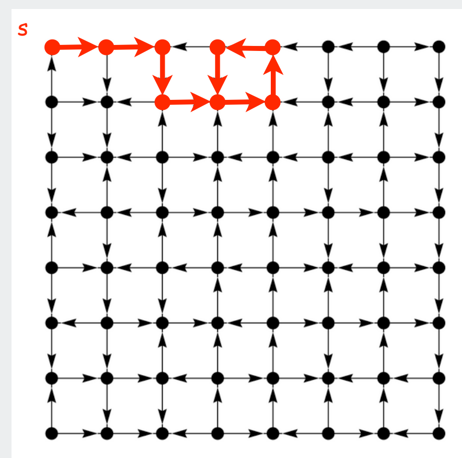
Cycle detection

Goal. Find any cycle in the graph



Cycle detection

Goal. Find any cycle in the graph

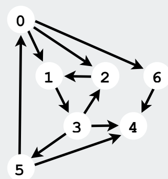


Can't find a cycle? The digraph is a DAG (directed acyclic graph)

Digraph-processing challenge 2:

Problem: Does a digraph contain a cycle ?
Equivalent: Is a digraph a DAG?

How difficult?
1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows



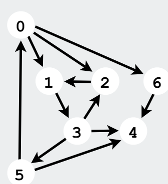

```
0-1
0-6
0-2
3-4
3-2
5-4
5-0
3-5
2-1
6-4
1-3
```


Digraph-processing challenge 2:

Problem: Does a digraph contain a cycle ?
Equivalent: Is a digraph a DAG?

How difficult? implementation
✓ 1) any CS126 student could do it
✓ 2) need to be a typical diligent CS226 student proof
3) hire an expert
4) intractable
5) no one knows

Use DFS
but prove that it works




```
0-1
0-6
0-2
3-4
3-2
5-4
5-0
3-5
2-1
6-4
1-3
```


Cycle detection in a digraph: Java implementation

```
public class CycleDetector
{
   private boolean[] marked;
   private boolean[] done;
   private boolean dagflag;

   public CycleDetector(Digraph G)
   {
      marked = new boolean[G.V()];
      done = new boolean[G.V()];
      count = G.V();
      for (int v = 0; v < G.V(); v++)
         if (!marked[v])
            dagflag = search(G, v);
   }

   private boolean search(Digraph G, int v)
   {
      marked[v] = true;
      for (int w : G.adj(v))
         if (!marked[w]) return search(G, w);
         else if (!done[w]) return false;
      done[v] = true;
      return true;
   }
}
```

standard DFS
with a few
modifications

add method dag()
to return dagflag on
client query

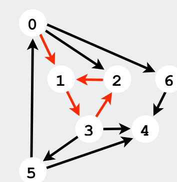
Cycle detection in a digraph: Example

Trace of marks set at beginning and end of search()

adj lists
```
0: 6 1 2
1: 3
2: 1
3: 4 2 5
4:
5: 0 4
6: 4
```

	marked[]	done[]
visit 0:	1 0 0 0 0 0 0	0 0 0 0 0 0 0
visit 6:	1 0 0 0 0 0 0	0 0 0 0 0 0 0
visit 4:	1 0 0 0 1 0 1	0 0 0 0 0 0 0
leave 4:	1 0 0 0 1 0 1	0 0 0 0 1 0 0
leave 6:	1 0 0 0 1 0 1	0 0 0 0 1 0 1
visit 1:	1 1 0 0 1 0 1	0 0 0 0 1 0 1
visit 3:	1 1 0 1 1 0 1	0 0 0 0 1 0 1
check 4:	ignore since marked and done	
visit 2:	1 1 1 1 1 0 1	0 0 0 0 1 0 1
check 1:		

cycle since marked but not done

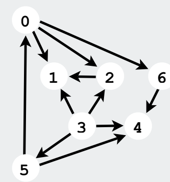



```
0-1
0-6
0-2
3-4
3-2
5-4
5-0
3-5
2-1
6-4
1-3
```

Cycle detection in a digraph: Correctness proof

`marked[v] = true && done[v] = false`

we know a directed path from source to `v`



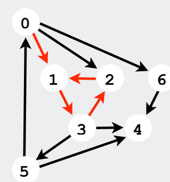
0-1
0-6
0-2
3-4
3-2
5-4
5-0
3-5
2-1
6-4
3-1

Case 1: no cycle

- `search()` will never touch a vertex that is marked and not done
- therefore will return `true`

Case 2: cycle

- `search()` must be called for some vertex on cycle
- that one will be found marked and not done
- return `false`



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

Depth First Search

DFS enables direct solution of simple digraph problems.

- ✓ Reachability.
- ✓ Cycle detection
- Topological sort
- Transitive closure.
- Is there a path from `s` to `t` ?

stay tuned

Basis for solving difficult digraph problems.

- Directed Euler path.
- Strong connected components.

Breadth-first search in digraphs

Same method as for undirected graphs

Every undirected graph is a digraph

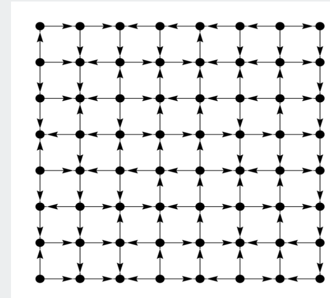
- happens to have edges in both directions
- BFS is a digraph algorithm (never uses that fact)

BFS (from source vertex `s`)

Put `s` onto a FIFO queue.

Repeat until the queue is empty:

- remove the least recently added vertex `v`
- add each of `v`'s unvisited neighbors to the queue and mark them as visited.



Finds the shortest directed path from s to t

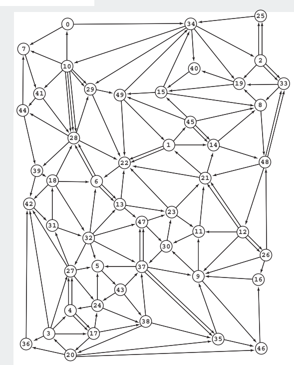
Digraph BFS application: Web Crawler

The internet is a digraph

Goal. Crawl Internet, starting from some root website.
Solution. BFS with implicit graph.

BFS.

- Start at some root website (say `http://www.princeton.edu`.).
- Maintain a `Queue` of websites to explore.
- Maintain a `SET` of discovered websites.
- Dequeue the next website and enqueue websites to which it links (provided you haven't done so before).



Q. Why not use DFS?

A. Internet is not fixed (some pages generate new ones when visited)

Web crawler: Java implementation

```
Queue<String> q = new Queue<String>();
SET<String> visited = new SET<String>();

String s = "http://www.princeton.edu";
q.enqueue(s);
visited.add(s);

while (!q.isEmpty())
{
   String v = q.dequeue();
   System.out.println(v);
   In in = new In(v);
   String input = in.readAll();

   String regexp = "http://(\\w+\\.)*(\\w+)";
   Pattern pattern = Pattern.compile(regexp);
   Matcher matcher = pattern.matcher(input);
   while (matcher.find())
   {
      String w = matcher.group();
      if (!visited.contains(w))
      {
         visited.add(w);
         q.enqueue(w);
      }
   }
}
```

queue of sites to crawl

set of visited sites

start crawling from s

read in raw html for next site in queue

http://xxx.yyy.zzz

use regular expression to find all URLs in site

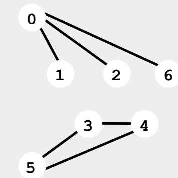
if unvisited, mark as visited and put on queue

Graph-processing challenge (revisited)

Problem: Is there a path from s to t ?

Assumptions: linear (V + E) preprocessing time

constant query time



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

How difficult?

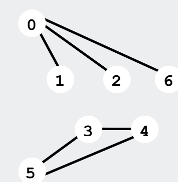
1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible

Graph-processing challenge (revisited)

Problem: Is there a path from s to t ?

Assumptions: linear (V + E) preprocessing time

constant query time



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

How difficult?

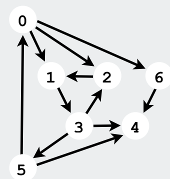
1) any CS126 student could do it
✓ 2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible

Use DFS
mark vertices with
connected component ID
(see lecture on undirected graphs)

Digraph-processing challenge 3

Problem: Is there a directed path from s to t ?
Assumptions: linear (V + E) preprocessing time
constant query time

How difficult?
1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible



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

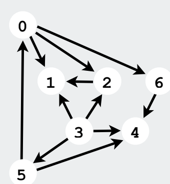
Digraph-processing challenge 3

Problem: Is there a directed path from s to t ?
Assumptions: linear (V + E) preprocessing time
constant query time

How difficult?
1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
✓ 6) impossible

V^2 possibilities

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



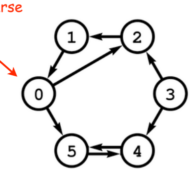
0-1
0-6
0-2
3-4
3-2
5-4
5-0
3-5
2-1
6-4
3-1

Transitive Closure

The transitive closure of G has an directed edge from **v** to **w**
if there is a directed path from **v** to **w** in G

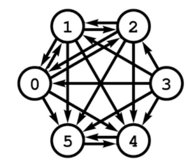
graph is usually sparse

G



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

Transitive closure of G



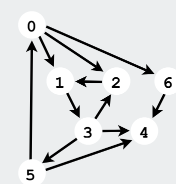
	0	1	2	3	4	5
0	1	1	1	0	1	1
1	1	1	1	0	1	1
2	1	1	1	0	1	1
3	1	1	1	1	1	1
4	0	0	0	0	1	1
5	0	0	0	0	1	1

TC is usually dense
so adjacency matrix
representation is OK

Digraph-processing challenge 3 (revised)

Problem: Is there a directed path from s to t ?
Assumptions: V^2 preprocessing time
constant query time

How difficult?
1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible



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

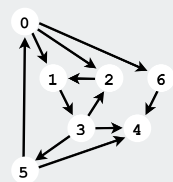
Digraph-processing challenge 3 (revised)

Problem: Is there a directed path from s to t ?

Assumptions: V^2 preprocessing time
constant query time

How difficult?

1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) ✓ no one knows
6) impossible



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

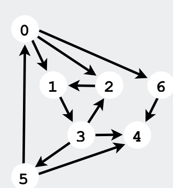
Digraph-processing challenge 3 (revised again)

Problem: Is there a directed path from s to t ?

Assumptions: VE preprocessing time
V^2 space
constant query time

How difficult?

1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible



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

Digraph-processing challenge 3 (revised again)

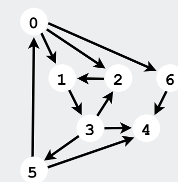
Problem: Is there a directed path from s to t ?

Assumptions: VE preprocessing time
V^2 space
constant query time

How difficult?

1) any CS126 student could do it
2) ✓ need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible

Use DFS once for each vertex to compute rows of transitive closure



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

Transitive Closure: Java Implementation

Use an array of `DFSearcher` objects, one for each row of transitive closure

```
public class DFSearcher
{
   private boolean[] marked;
   public DFSearcher(Digraph G, int s)
   {
      marked = new boolean[G.V()];
      dfs(G, s);
   }
   private void dfs(Digraph G, int v)
   {
      marked[v] = true;
      for (int w : G.adj(v))
         if (!marked[w]) dfs(G, w);
   }
   public boolean isReachable(int v)
   {
      return marked[v];
   }
}
```

```
public class TransitiveClosure
{
   private DFSearcher[] tc;

   public TransitiveClosure(Digraph G)
   {
      tc = new DFSearcher[G.V()];
      for (int v = 0; v < G.V(); v++)
         tc[v] = new DFSearcher(G, v);
   }

   public boolean reachable(int v, int w)
   {
      return tc[v].isReachable(w);
   }
}
```

is there a directed path from v to w ?

introduction
digraph search
transitive closure
topological sort
strong components

Digraph application: Scheduling

Scheduling. Given a set of tasks to be completed with precedence constraints, in what order should we schedule the tasks?

Graph model.

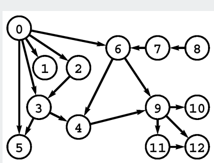
- Create a vertex v for each task.
- Create an edge v→w if task v must precede task w.
- Schedule tasks in topological order.

tasks

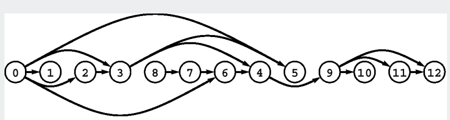
0. read programming assignment
1. download files
2. write code
3. attend precept

...
12. sleep

precedence constraints →

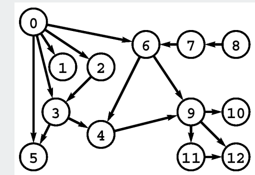


feasible schedule →

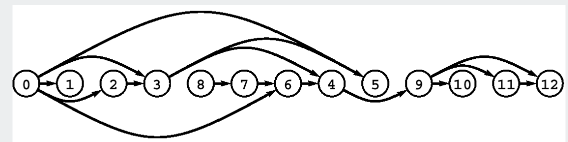


Topological Sort

DAG. Directed acyclic graph.



Topological sort. Redraw DAG so all edges point left to right.



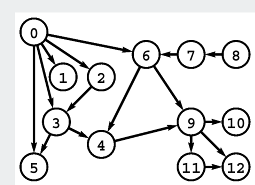
Observation. Not possible if graph has a directed cycle.

Digraph-processing challenge 4

Problem: Check that the digraph is a DAG.
If it is a DAG, do a topological sort.
Assumptions: linear (V + E) preprocessing time
provide client with vertex iterator for topological order

How difficult?

1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible



0-1
0-6
0-2
0-5
2-3
4-9
6-4
6-9
7-6
8-7
9-10
9-11
9-12
11-12

Digraph-processing challenge 4

Problem: Check that the digraph is a DAG.
If it is a DAG, do a topological sort.
Assumptions: linear (V + E) preprocessing time
provide client with vertex iterator for topological order

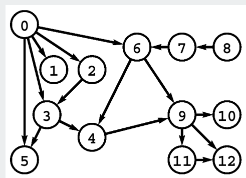
How difficult?

implementation

proof

✓ 1) any CS126 student could do it
✓ 2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible

Use DFS reverse postorder numbering



0-1
0-6
0-2
0-5
2-3
4-9
6-4
6-9
7-6
8-7
9-10
9-11
9-12
11-12

Topological sort in a DAG: Java implementation

```
public class TopologicalSorter
{
   private int count;
   private boolean[] marked;
   private int[] ts;

   public TopologicalSorter(Digraph G)
   {
      marked = new boolean[G.V()];
      ts = new int[G.V()];
      count = G.V();
      for (int v = 0; v < G.V(); v++)
         if (!marked[v]) tsort(G, v);
   }

   private void tsort(Digraph G, int v)
   {
      marked[v] = true;
      for (int w : G.adj(v))
         if (!marked[w]) tsort(G, w);
      ts[--count] = v;
   }
}
```

standard DFS
with 3
extra lines of code

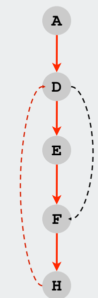
add iterator that returns
`ts[0], ts[1], ts[2]...`

Topological sort in a DAG: Correctness proof

To topologically sort a DAG.
- Run DFS to compute reverse postorder numbering in `ts[]`.
- Use client iterator to return `ts[0], ts[1], ts[2], ...`

Key observation. When DFS backtracks from a vertex `v`, all vertices reachable from `v` have already been explored.

no back edges in DAG



DFS
tree

Running time. linear

Topological sort applications.

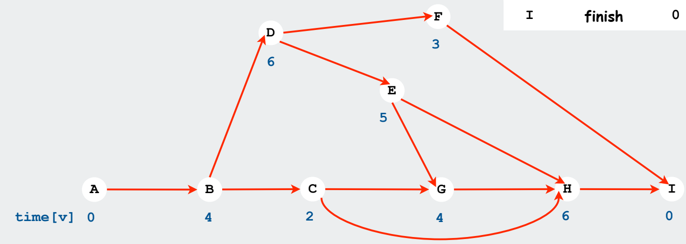
- Causalities.
- Compilation units.
- Class inheritance.
- Course prerequisites.
- Deadlocking detection.
- Temporal dependencies.
- Pipeline of computing jobs.
- Check for symbolic link loop.
- Evaluate formula in spreadsheet.
- Program Evaluation and Review Technique / Critical Path Method

Topological sort application (weighted DAG): PERT/CPM

Program Evaluation and Review Technique / Critical Path Method

- Task `v` takes `time[v]` units of time.
- Can work on jobs in parallel.
- Precedence constraints: must finish task `v` before beginning task `w`.
- What's earliest we can finish each task?

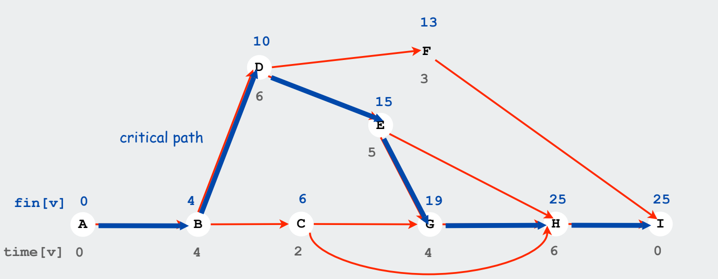
index	task	time	prereq
A	begin	0	-
B	framing	4	A
C	roofing	2	B
D	siding	6	B
E	windows	5	D
F	plumbing	3	D
G	electricity	4	C, E
H	paint	6	C, E
I	finish	0	F, H



Program Evaluation and Review Technique / Critical Path Method

PERT/CPM algorithm.

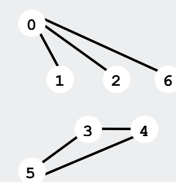
- Compute topological order of vertices.
- Initialize `fin[v] = 0` for all vertices `v`.
- Consider vertices `v` in topological order.
  - for each edge `v→w`, set `fin[w]= max(fin[w], fin[v] + time[w])`



Graph-processing challenge (revisited again)

Problem: Is there a path from s to t ?
Equivalent: Are s and t connected?

Assumptions: linear (V + E) preprocessing time
constant query time



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

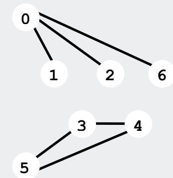
How difficult?

1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible

Graph-processing challenge (revisited again)

Problem: Is there a path from s to t ?
Equivalent: Are s and t connected?

Assumptions: linear (V + E) preprocessing time
constant query time



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

How difficult?
1) any CS126 student could do it
✓ 2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible

Use DFS
mark vertices with
connected component ID
(see lecture on undirected graphs)

Digraph-processing challenge 5

Problem: Is there a directed cycle containing s and t ?
Equivalent: Are there directed paths from s to t and from t to s?
Equivalent: Are s and t strongly connected?

Assumptions: linear (V + E) preprocessing time
constant query time

How difficult?
1) any CS126 student could do it
2) need to be a typical diligent CS226 student
3) hire an expert
4) intractable
5) no one knows
6) impossible

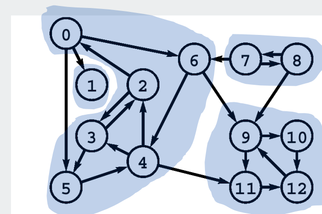
Strong components: terminology

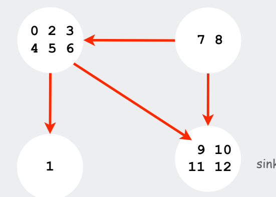
Def. Vertices v and w are strongly connected if there is a path from v to w and a path from w to v.

symmetric, transitive, reflexive.

Strong component. Maximal subset of strongly connected vertices.

Kernel DAG.
- vertex: set of vertices in same strong component
- edge: any edge from original graph connecting two vertices





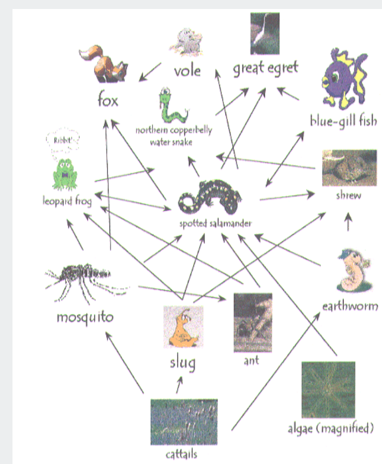
kernel DAG

	0	1	2	3	4	5	6	7	8	9	10	11	12
sc	2	1	2	2	2	2	2	3	3	0	0	0	0

Typical strong components application: Ecological food web

Strong component is subset of species with common energy flow
- source in kernel DAG: heading to extinction?
- sink in kernel DAG: heading for growth?

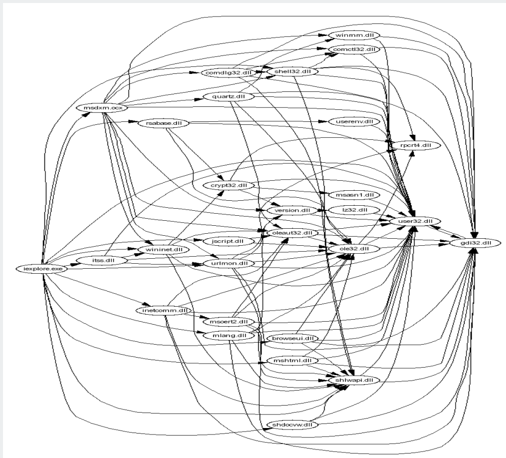
Digraph changes over time



Typical strong components application: Packaging software

Strong component is subset of mutually interacting modules

Approach: Package together modules in same strong component



Strong components algorithms: brief history

1960s: Core OR problem
- widely studied
- some practical algorithms
- complexity not understood

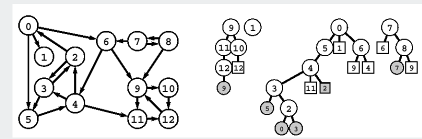
1972: Linear-time DFS algorithm (Tarjan)
- classic algorithm
- level of difficulty: CS226++
- demonstrated broad applicability and importance of DFS

1980s: Easy two-pass linear-time algorithm (Kosaraju)
- forgot notes for teaching algorithms class
- developed algorithm in order to teach it!
- later found in Russian scientific literature (1972)

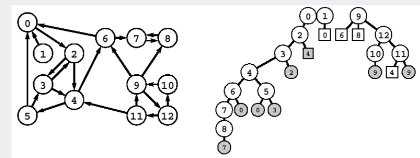
1990s: More easy linear-time algorithms (Gabow, Mehlhorn)
- Gabow: fixed old OR algorithm
- Mehlhorn: needed one-pass algorithm for LEDA

Kosaraju's Algorithm

Simple (but mysterious) algorithm for computing strong components
- Run DFS on G^R and compute postorder.
- Run DFS on G, considering vertices in reverse postorder.



	0	1	2	3	4	5	6	7	8	9	10	11	12
post	8	7	6	5	4	3	2	0	1	11	10	12	9



	0	1	2	3	4	5	6	7	8	9	10	11	12
sc	2	1	2	2	2	2	2	3	3	0	0	0	0

Theorem. Trees in second DFS are strong components. (!)

Digraph-processing challenge 5

Problem: Is there a directed cycle containing s and t ?
Equivalent: Are there directed paths from s to t and from t to s?
Equivalent: Are s and t strongly connected?

Assumptions: linear (V + E) preprocessing time
constant query time

Use DFS (twice)

How difficult?

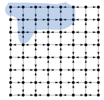
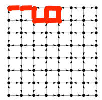
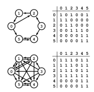
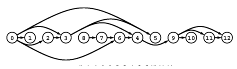
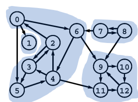
implementation

✓ 1) any CS126 student could do it
2) need to be a typical diligent CS226 student
✓ 3) hire an expert (well, maybe a CS341 student)

proof

4) intractable
5) no one knows
6) impossible

Digraph-processing summary: Algorithms of the day

Single-source reachability		DFS
cycle detection		DFS
transitive closure		DFS from each vertex
topological sort (DAG)		DFS
strong components		Kosaraju DFS (twice)