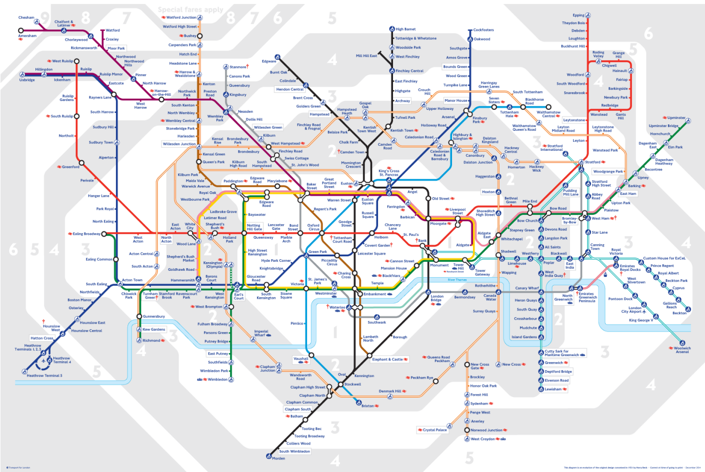
Directed graphs

- Introduction
- Representation
- Depth First Search / Breadth First Search
- Topological Sorting
- Strongly Connected Components
- Implicit Graphs

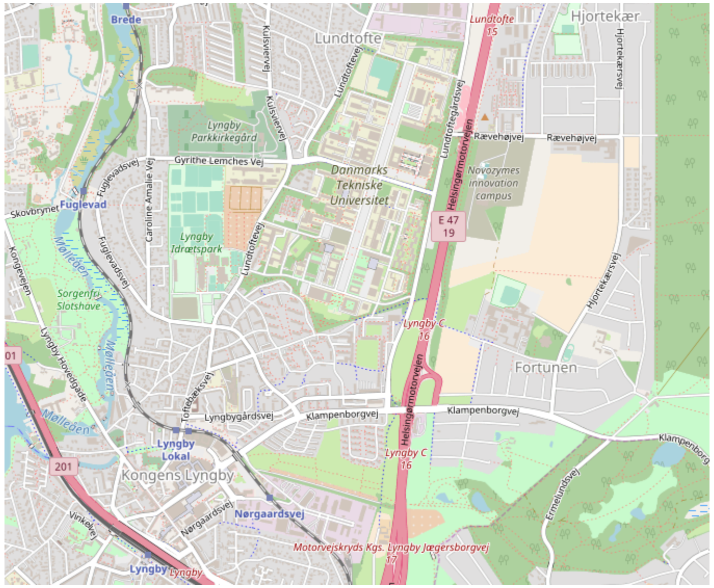
Directed graphs

- Introduction
- Representation
- Depth First Search / Breadth First Search
- Topological Sorting
- Strongly Connected Components
- Implicit Graphs

Un-directed graph example: Transport



Example: Streetmap – a graph?



Example: Streetmap – a graph?

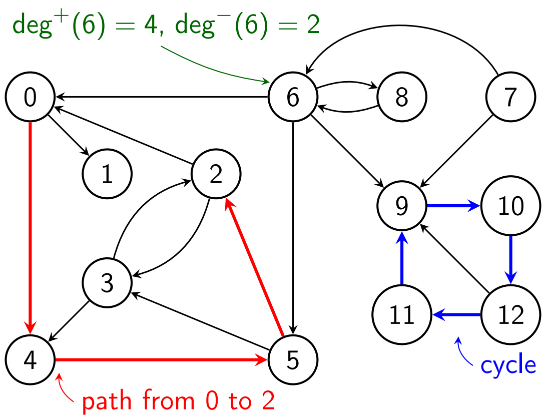
Example: Streetmap – a graph?

Some streets are *one-way*. This is modelled by directed graphs.

Directed graph

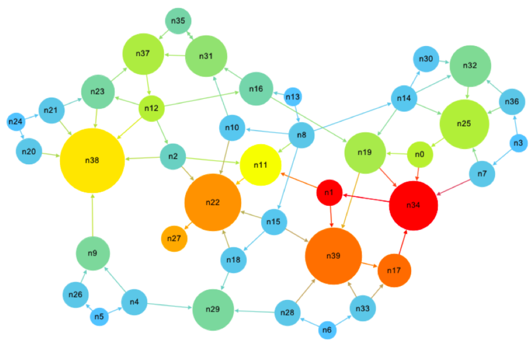
Definition (Directed graph)

Set of vertices, pairwise joined by *directed* edges.



Application: WWW

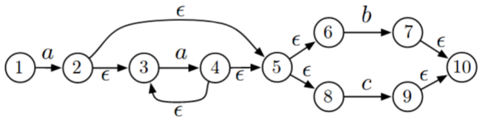
- Vertex: Web page. Edge: Hyperlink.
- Webcrawling. Page rank.



http://computationalculture.net/what_is_in_pagerank/

Application: Automata, regular expressions

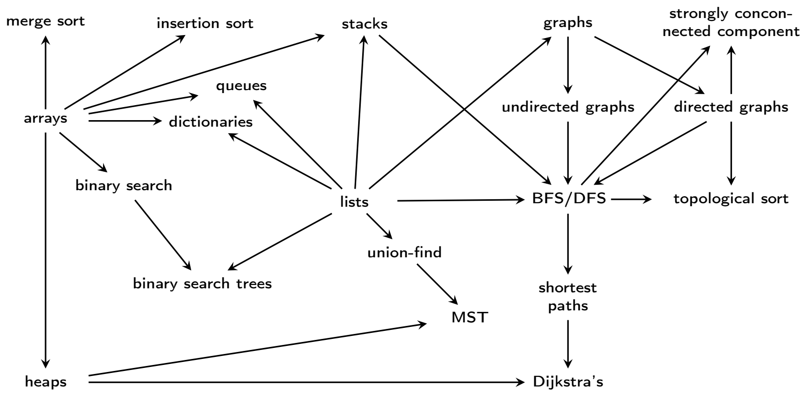
- Vertex: State. Edge: Transition.
- This automaton accepts "aaab" ⇔ there is a path from vertex 1 to vertex 10 that matches the string "aaab"
- Regular expressions can be represented by automata.



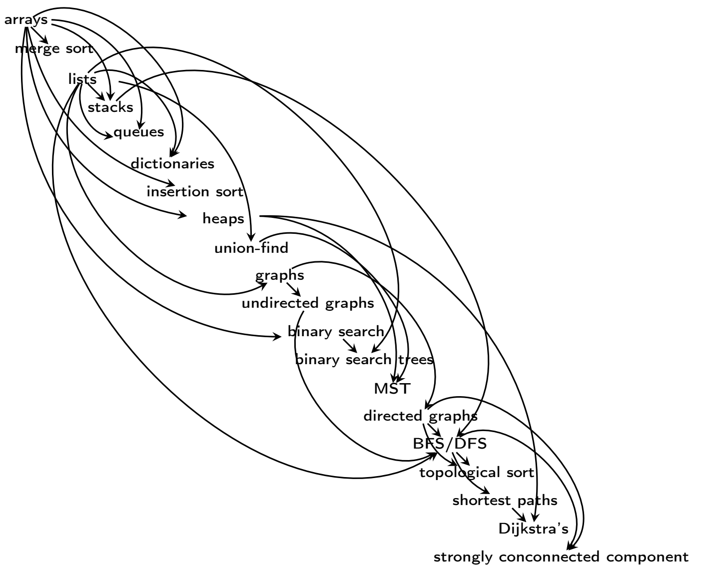
R = a·(a*)·(b|c)

Application: Dependencies

- Vertex: subject. Edge: dependency.
- Are there any cyclic dependencies? Can we avoid that the present subject depends on a future one?

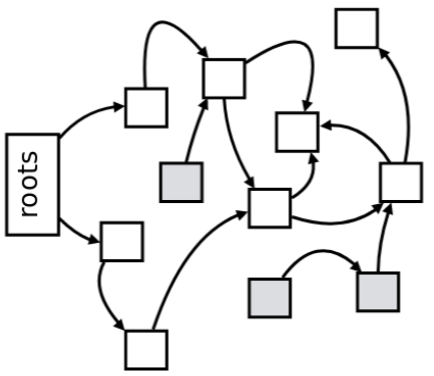


Application: Dependencies



Application: Garbage Collection

Vertex: Object. Edge: Reference.



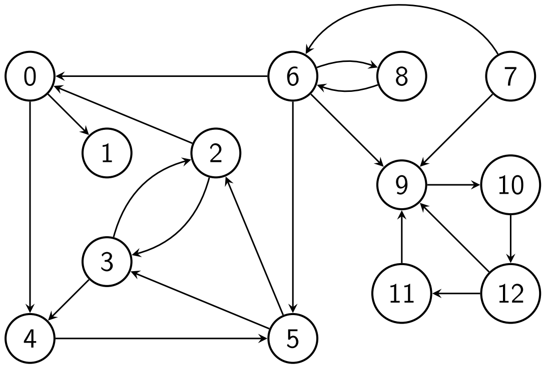
Applications

graph	vertices	edges
internet	webpage	hyperlink
transport	intersection	oneway street
scheduling	job	precedence relation
infectious disease	person	disease transmission
citation network	article	citation
object graph	objects	pointers
object hierarchy	class	inheritance

Directed Graphs

Lemma
$\sum_{v \in V} \deg^-(v) = \sum_{v \in V} \deg^+(v) = m$

Proof.
Every edge has exactly one head and one tail. □



Algorthmic problems on direted graphs

Reachability. Is there a path from s to t?

Shortest path. What is the shortest path from s to t?

Directed cycle. Does the graph contain a (directed) cycle?

Topological sort. Can we arrange the vertices such that all the edges go the same direction?

Strong connectivity. Is there a path from anywhere to anywhere else in the graph?

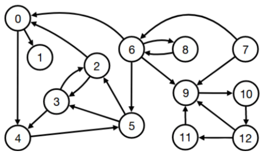
Transitive closure. Every path in a graph is represented by an edge in the transitive closure of that graph.

Directed graphs

- Introduction
- Representation
- Depth First Search / Breadth First Search
- Topological Sorting
- Strongly Connected Components
- Implicit Graphs

Representation

- G is a directed graph with n vertices and m edges
- Representation. We need the following operations:
  - `PointsTo(u,v)`: Does u point to v?
  - `Neighbours(v)`: Returns all the vertices that v points to.
    (Aka. all *out-neighbours* of v.)
  - `Insert(v,u)`: Add the edge (v, u) to G.
    (unless already present).



Adjacency matrix

Directed graph G with n vertices and m edges.
Adjacency matrix:

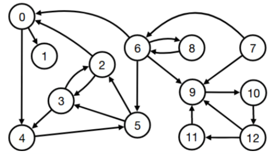
- $n \times n$ matrix A
- $A[i, j] = 1$ when $i \to j$, else 0.

Space $O(n^2)$.
Time

`PointsTo(u,v)` $O(1)$ time.

`Neighbours(v)` $O(n)$ time.

`Insert(v,u)` $O(1)$ time.



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

Adjacency list

Directed graph G with n vertices and m edges.
Adjacency list:

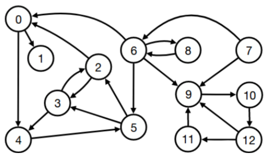
- Array $A[0 \ldots n-1]$
- $A[i]$ contains a list of all vertices that i points to.

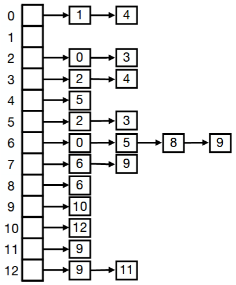
Space
$O(n + \sum_{v \in V} \deg^+(v)) = O(n + m)$.
Time

`PointsTo(u,v)` $O(\deg^+(u))$ time.

`Neighbours(v)` $O(\deg^+(v))$ time.

`Insert(v,u)` $O(\deg^+(v))$ time.





Representation

Data structure	`PointsTo`	`Neighbours`	`Insert`	Space
Adjacency matrix	$O(1)$	$O(n)$	$O(1)$	$O(n^2)$
Adjacency list	$O(\deg^+(v))$	$O(\deg^+(v))$	$O(\deg^+(v))$	$O(n+m)$

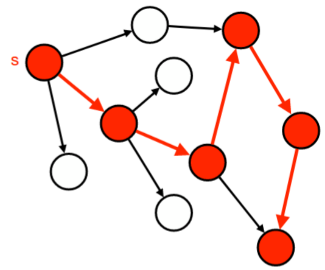
Directed graphs

- Introduction
- Representation
- Depth First Search and Breadth First Search
- Topological Sort
- Strongly Connected Components
- Implicit Graphs

Depth First Search / Breadth First Search

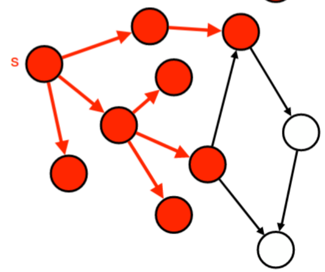
Depth First Search

- Let all vertices be unmarked. Visit s.
- When visiting v:
  - Mark v,
  - Recursively visit the *out-neighbours* of v.



Breadth First Search

- Let all vertices be unmarked.
- Mark s, add s to queue Q.
- While Q is not empty:
  - Dequeue v from Q,
  - For all u such that $v \to u$
    - Mark u, add u to Q.



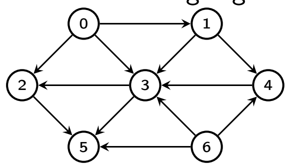
Time $O(n+m)$

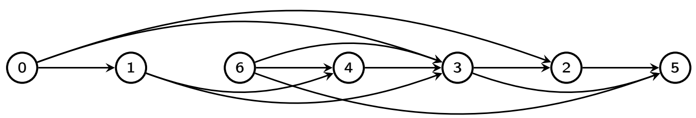
Directed graphs

- Introduction
- Representation
- Search
- Topological Sorting
- Strongly Connected Components
- Implicit Graphs

Topological Sorting and DAGs

DAG Directed Acyclic Graph. Does not contain a cycle.
Topological sorting. An ordering of the vertices on a horizontal line, such that all edges go left to right.





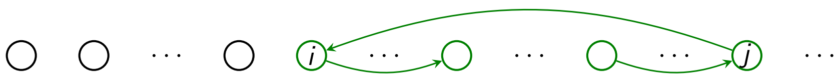
Algorithmic problems

- Determine whether the input graph G is a DAG.
- Return a topological sorting of the vertices (in the affirmative case).

Goal: Show G is a DAG $\Leftrightarrow$ G has topological sorting.
Give an algorithm for solving both.

Topological Sorting and DAGs

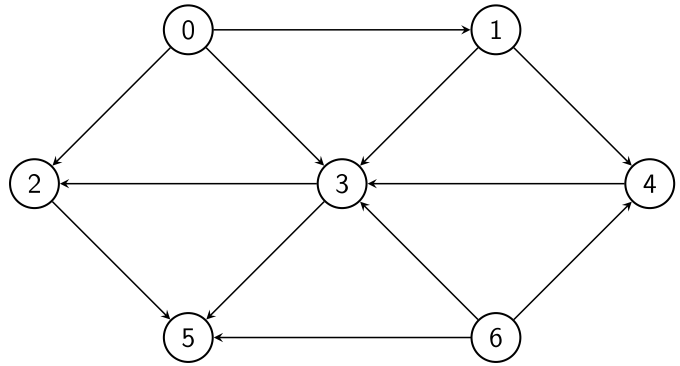
Lemma G has a topological sorting $\Rightarrow$ G is a DAG.
Proof. Assume G has a topological sorting.



If G is not a DAG, then it has a cycle, $K = v_{k_0}$.
Let j be the vertex of K *furthest* to the right.
There is some edge $j \to i$ in K, and thus in G.
But i is before j $\Rightarrow$ not a topological sorting.

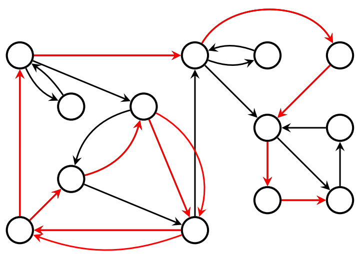
Topological Sorting and DAGs - Exercise

Come up with a strategy for finding a topological sorting of a given DAG.



Topological Sorting and DAGs

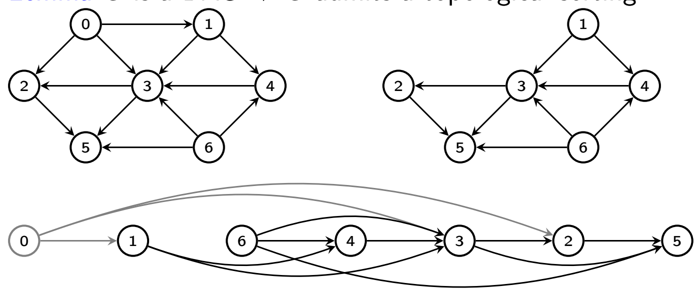
Lemma. G is a DAG $\Rightarrow$ G has a vertex v with $\deg^-(v) = 0$, that is, *in-degree* 0. No other vertex points to v.



Proof. Assume every vertex v has in-degree ≥ 1.
Walk backwards for $n + 1$ steps, starting at any vertex s.
There are only n vertices in G, so at least one vertex must have been visited twice; we have found a cycle. G is not a DAG.

Topological Sorting and DAGs

Lemma G is a DAG $\Rightarrow$ G admits a topological sorting



Proof by induction over the number of vertices in G.

- (Base Case) If the graph has only one vertex, it already sorted.
- (Induction Step)
  - Find a vertex v with $\deg^-(v) = 0$.
  - $G - v$ is still a DAG. $G - v$ has a topological sorting.
  - Place v furthest to the left, followed by the sorting of $G - v$. This is a valid topological sorting since no edges go into v!

Topological sorting – Implementation

Goal Efficient algorithm on the adjacency list representation.
Algorithm Based on the proof:

if $G = (\{v\}, \emptyset)$ **then**
 print v.
else
 find v with $\deg^-(v) = 0$
 print v
 TopSort$(G - v)$
end if

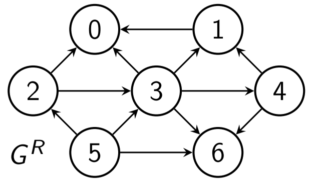
Correctness Follows from the proof.
Time Repeat until all but one vertex is removed: n times.

- Find a vertex of in-degree 0 How much time for this?
- Remove it from the graph. Every edge is removed exactly once $\Rightarrow$ Total time $O(m)$ on this step.

Topological sorting – Implementation 1 (not smart)

Solution 1 Construct the *reversed* graph G^R:

if $G = (\{v\}, \emptyset)$ **then**
 print v.
else
 find v with $\deg^-(v) = 0$
 print v
 TopSort$(G - v)$
end if



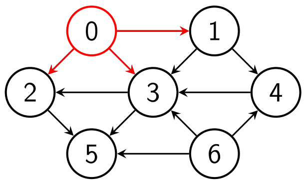
Linear search in G^R to find a vertex of out-degree 0.
Time Repeat until all but one vertex is removed: n times.

- Find a vertex of in-degree 0 $O(n)$ time
- Remove it from the graph Every edge is removed exactly once $\Rightarrow$ Total time $O(m)$ on this step.

Total $O(n^2 + m) = O(n^2)$.

Topological sorting – Implementation 2 (smart)

Solution 2 Maintain information about the indegrees of all vertices.
Keep a linked list of vertices of degree 0.



$(v, \deg^-(v))$ table

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

0-deg$^-$ list
$\boxed{0} \rightarrow \boxed{6}$

Initialising $O(n + m)$ time.

Repeat until all but one vertex is removed: n times.

- Find a vertex of in-degree 0 $O(1)$ time
- Remove it from the graph Every edge is removed exactly once $\Rightarrow$ Total time $O(m)$ on this step.

Total $O(n + m)$.

Topological Sorting and DAGs

Lemma
G is a DAG $\Leftrightarrow$ G has a topological sorting.

Theorem
There is an $O(n + m)$ time algorithm that determines whether G is a DAG, and, in the affirmative case outputs a topological sorting.

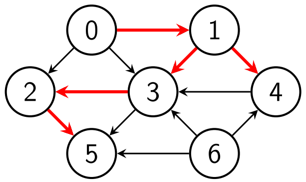
Topological Sorting via DFS

Idea:

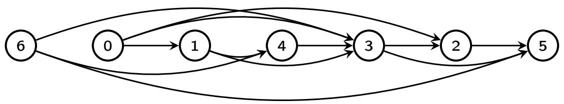
- Run DFS on G
- When returning from the recursive call on vertex v, push v to a stack.
- Print stack.

Time $O(m+n)$

Intuition Recursively finds vertices of out-degree 0.



6 0 1 4 3 2 5



Directed graphs

- Introduction
- Representation
- Depth First Search / Breadth First Search
- Topological Sorting
- Strongly Connected Components
- Implicit Graphs

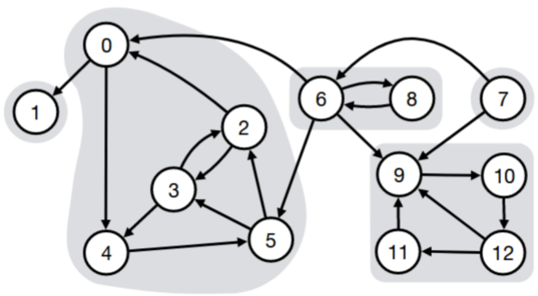
Strongly connected components

Definition (Strongly connected)

u and v are *strongly connected* if there is a path u to v, and a path v to u.

Definition (Strongly connected component)

Maximal subset of strongly connected vertices.



Strongly connected components via DFSes

Idea

- Run DFS on the reversed graph G^R.
  Note the finish times of all vertices.
- Run DFS on G, but when starting a new "round", always start on the unmarked vertex with the highest finish-time.
- Each round finds and marks a strongly connected component.

Correctness See Chapter 22.5 in CLRS

Time $O(n+m)$

Directed graphs

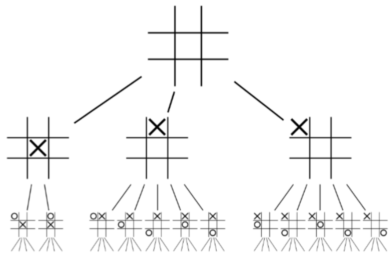
- Introduction
- Representation
- Depth First Search / Breadth First Search
- Topological Sorting
- Strongly Connected Components
- Implicit Graphs

Implicit Graph Representation

Implicit graph. Directed or undirected graph given by an *implicit representation*:

- initial vertex s
- algorithm for *generating* the neighbours of a vertex.

Applications Games, Artificial Intelligence, ...

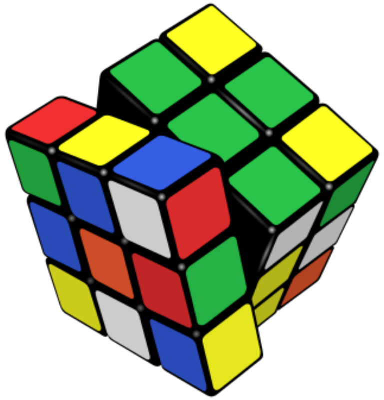


Implicit Graph Example: Rubik's Cube

Rubics Cube.

- $n + m = 43.252.003.274.489.856.000 \simeq 43$quintillion

What is the fewest moves to get the "tidy" cube, regardless how jumbled it is when you start?



Implicit Graph Example: Rubik's Cube

Year	lower bound	upper bound
1981	18	52
1990	18	42
1992	18	39
1992	18	37
1995	18	29
1995	20	29
2005	20	28
2006	20	27
2007	20	26
2008	20	25
2008	20	23
2008	20	22
2010	20	20

Directed graphs

- Introduction
- Representation
- Depth First Search / Breadth First Search
- Topological Sorting
- Strongly Connected Components
- Implicit Graphs