Graph algorithms: BFS, DFS, and Topological Sort

Rossano Venturini[1]

1 **Computer Science Department, University of Pisa, Italy**
rossano.venturini@unipi.it

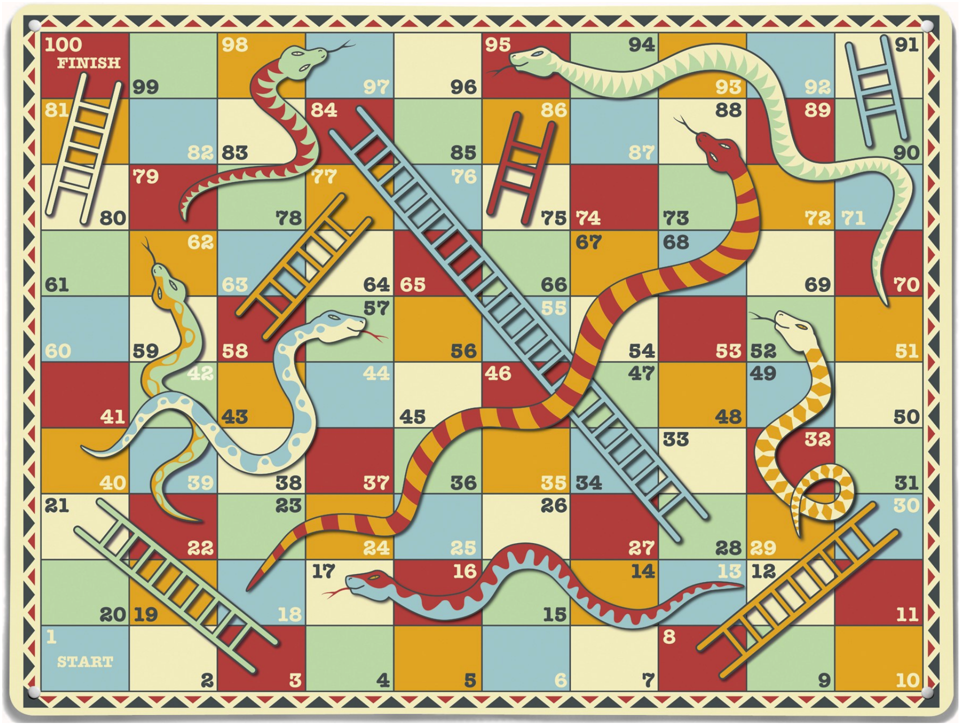
Notes for the course *"Competitive Programming and Contests"* at Department of Computer Science, University of Pisa.
Web page: `https://github.com/rossanoventurini/CompetitiveProgramming`

These notes sketch the content of the 9th lecture. The topics of this lecture are treated in any introductory book on Algorithms. You may refer to Chapter 22 of *Introduction to Algorithms, 3rd Edition*, Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein, The MIT Press, 2009 for a more detailed explanation.

1 Snakes and Ladders

Snakes and Ladders is a popular board game. Quoting Wikipedia: *Snakes and Ladders is played between two or more players on a game-board having numbered, gridded squares. A number of "ladders" and "snakes" are pictured on the board, each connecting two specific board squares. The object of the game is to navigate one's game piece, according to die rolls, from the start (bottom square) to the finish (top square), helped or hindered by ladders and snakes respectively.*

An example is shown in the following picture.

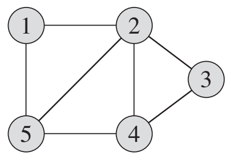


The question is: Given a board, what's the minimum number of tosses of 6-face die needed to win? What's the time complexity of an efficient algorithm to answer the previous question?

2 Graph representation

A graph $G = (V, E)$ is formed by n vertecis $V = \{1, 2, \ldots, n\}$ and m edges $E \subseteq V \times V$.

For example the graph G is $V = \{1, 2, 3, 4, 5\}$ e $E = \{(1, 2), (1, 5), (2, 3), (2, 4), (2, 5), (3, 4), (4, 5)\}$. The graph G is shown in the following figure.



A graph can be either *directed* or *undirected*.

There are two alternative ways to represent a graph: *adjacency list* or *adjacency matrix*.

The former requires $\Theta(m \log m)$ bits of space while the latter requires $\Theta(n^2)$ bits. Thus, if the graph is sparse, i.e., $m = o(n^2/\log n)$, the former is more space efficient.

3 Graph traversal

3.1 Breadth First Search (BFS)

BFS is a simple algorithm for traversing a graph. Starting from a distinguished source vertex, BFS will visit vertices that are direct neighbors of the source vertex, neighbors of direct neighbors, and so on.

BFS starts with the insertion of the source vertex s into a queue, then processes the queue as follows: Take out the front most vertex u from the queue, enqueue all unvisited neighbors of u, and mark them as visited.

Below the pseudocode of the BFS.

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

The above pseudocode uses three colors (white, gray and black) for the vertecis. The color of a node u is as follows.

- color[u] = White - for the "undiscovered" state
- color[u] = Gray - for the "discovered but not finished" state
- color[u] = Black - for the "finished" state

Initially any node, it becomes gray once it is dicovered, and, finally, black once it is also visited.

While visiting the graph, the BFS builds a (rooted) tree. The source s is the root. The vertecis connected to the root in G are the children of s in the tree. In general, a vertex u is the parent of all the vertecis connected to u that were white (i.e., not already visited) when visiting u.

BFS identifies, for any (reachable) vertex u, a smallest path (in number of edges) from s to u (the path is represented by means of BFS tree) and $u.d$ is the length of that path.

We denote with $\delta(s, u)$ the lenght of a shortest path from s to u.

▶ **Theorem 1.** *For every vertex v, $v.d = \delta(s, v)$.*

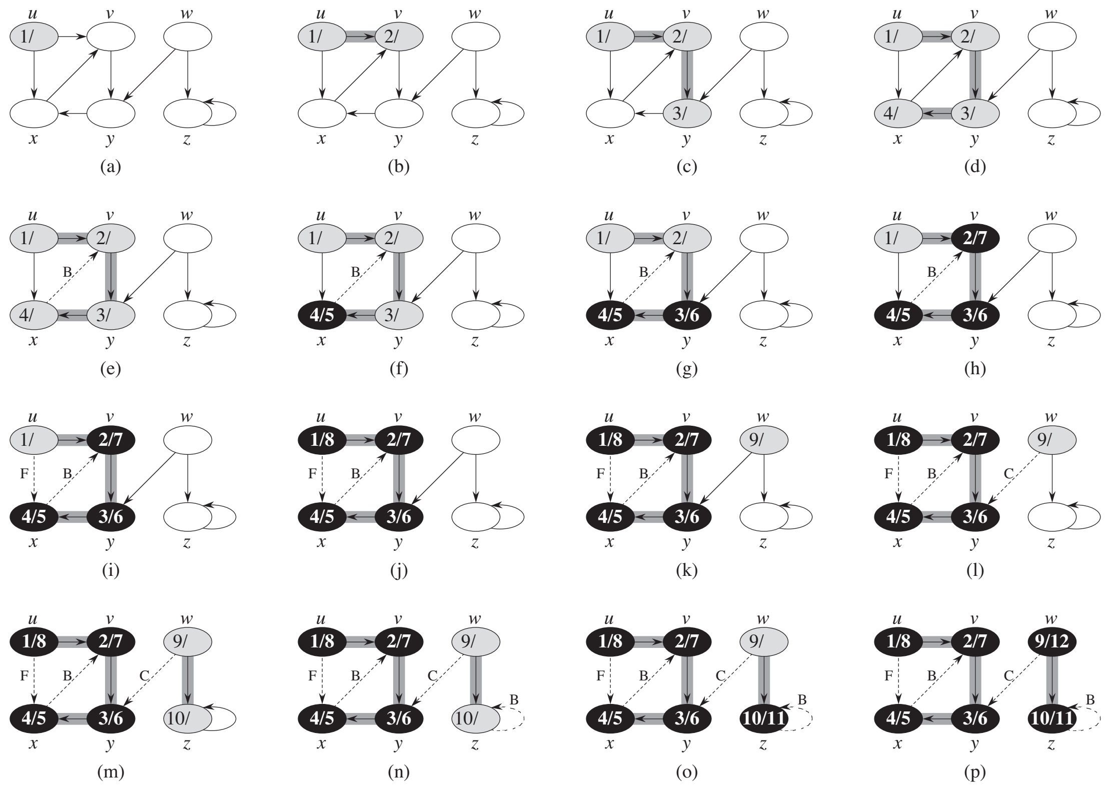
BFS runs in $\Theta(n + m)$ time, assuming that the graph is represented with adjacency lists, $\Theta(n^2)$ time otherwise.

3.2 Depth First Search (DFS)

Like BFS we color the vertecies.

```
DFS($G$)
1  for each vertex $u \in G.V$
2      $u.color$ = WHITE
3      $u.\pi$ = NIL
4  $time = 0$
5  for each vertex $u \in G.V$
6      if $u.color$ == WHITE
7          DFS-VISIT($G, u$)
```

```
DFS-VISIT($G, u$)
 1  $time = time + 1$              // white vertex $u$ has just been discovered
 2  $u.d = time$
 3  $u.color$ = GRAY
 4  for each $v \in G.Adj[u]$      // explore edge $(u, v)$
 5      if $v.color$ == WHITE
 6          $v.\pi = u$
 7          DFS-VISIT($G, v$)
 8  $u.color$ = BLACK              // blacken $u$; it is finished
 9  $time = time + 1$
10  $u.f = time$
```



Like BFS, depth-first search uses $\pi[v]$ to record the parent of vertex v. We have $\pi[v] = NIL$ if and only if vertex v is the root of a depth-first tree.

However, DFS forms a depth-first forest comprised of more than one depth-first trees.

DFS time-stamps each vertex when its color is changed.

1. When vertex v is changed from white to gray the time is recorded in $d[v]$.
2. When vertex v is changed from gray to black the time is recorded in $f[v]$.

The discovery and the finish times are unique integers, where for each vertex the finish time is always after the discovery time. That is, each time-stamp is an unique integer in the range of $[1, 2V]$.

For each vertex v, $d[v] < f[v]$.

▶ **Theorem 2** (Parenthesis Theorem). *For all u and v, exactly one of the following holds*

1. $d[u] < f[u] < d[v] < f[v]$ *or* $d[v] < f[v] < d[u] < f[u]$ *and neither of u and v is a descendant of the other.*
2. $d[u] < d[v] < f[v] < f[u]$ *and v is a descendant of u.*
3. $d[v] < d[u] < f[u] < f[v]$ *and u is a descendant of v.*

So, $d[u] < d[v] < f[u] < f[v]$ cannot happen.

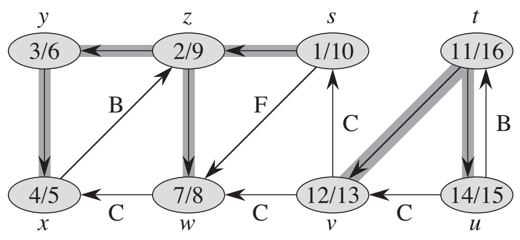
▶ **Theorem 3** (White-path Theorem). *Vertex v is a descendant of u if and only if at time $d[u]$, there is a path u to v consisting of only white vertices. (Except for u, which was just colored gray.)*

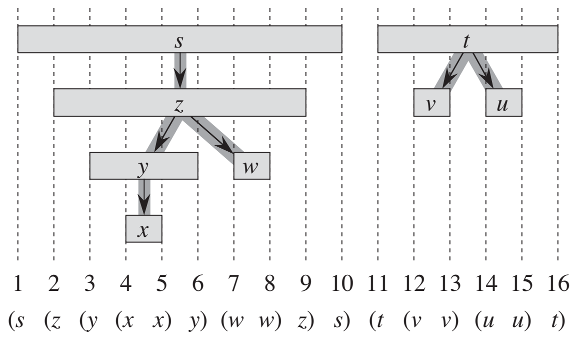
Consider a directed graph $G = (V, E)$. After a DFS of graph G we can put each edge into one of four classes:

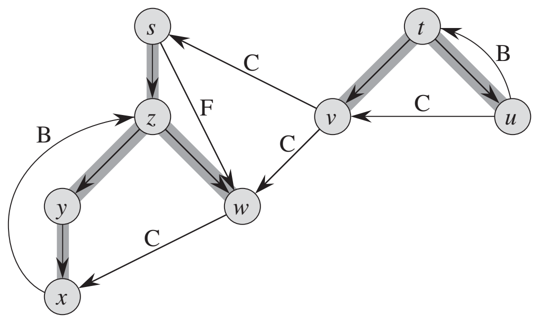
1. A **tree edge** is an edge in a DFS-tree.
2. A **back edge** connects a vertex to an ancestor in a DFS-tree.
3. A **forward edge** is a non-tree edge that connects a vertex to a descendent in a DFS-tree.
4. A **cross edge** is any other edge in graph G. It connects vertices in two different DFS-tree or two vertices in the same DFS-tree neither of which is the ancestor of the other.

▶ **Lemma 4.** *An edge (u, v) is*

1. *a back edge if and only if* $d[v] < d[u] < f[u] < f[v]$.
2. *a cross edge if and only if* $d[v] < f[v] < d[u] < f[v]$.
3. *a tree or forward edge if* $d[u] < d[v] < f[v] < f[u]$.



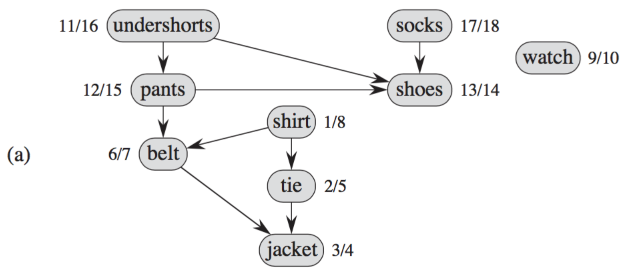


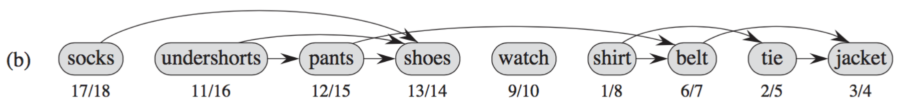


4 Topological Sort

A Directed Acyclic Graph (DAG) may seen as a way to express precedences among items. A topological sort of a Directed Acyclic Graph (DAG) $G = (V, E)$ is a linear ordering of all its vertices such that if G contains an edge (u, v), then u precedes v in the ordering.

Many applications use directed acyclic graphs to indicate precedences among events. For example, the DAG below shows temporal precedences in dressing.





There exists a simple algorithm to find the topological sort of a DAG.

- run DFS(G) to compute the finishing time $v.f$ of every vertex v
- as each vertex is finished, insert it onto a stack
- the final stack is the linear ordering

The algorithm runs in $\Theta(V + E)$ time.

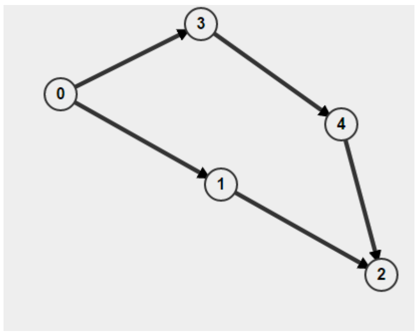
▶ **Lemma 5.** *A directed graph G is acyclic if and only if a DFS of G yields no back edge.*

Proof. $\Rightarrow$: Suppose a DFS produces a back edge (u, v). Then, vertex v is an ancestor of u in the DFS-forest. Thus, G has a path from v to u and (u, v) completes the cycle.

$\Leftarrow$: Suppose G is not a acyclic, i.e., it has a cycle c. Let v be the first vertex in c which is visited by DFS and (u, v) the preceding edge in c. At time $v.d$, the vertices in c form a path of white vertices from v to u. By the white-path theorem, vertex u becomes a descendant of v in the depth-first forest. Therefore, (u, v) is a back edge. ◀

Observe that BFS is not able to find cycles in directed graphs. The reason is that it does not suffice to check if a node has been already visited but we also need to remember form which vertex we are coming.

In the example below, the BFS would detect a non-existing cycle.



We prove now that the above algorithm correctly computes the topological sort of the input DAG G.

Suppose DFS is run to compute the finishing times of vertecis of G.

It suffices to show that for any pair of distinct vertecis u and v, if G contains an edge (u, v), then $v.f < u.f$.

Consider any edge (u, v) explored by DFS. When this edge is explored, v cannot be gray, since then v would be an ancestor of u and (u, v) would be a back edge.

Thus, v must be either white or black.

- If v is white, it becomes a descendant of u, and so $v.f < u.f$.
- If v is black, clearly $v.f$ has been already set and the relation holds.