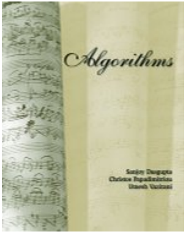
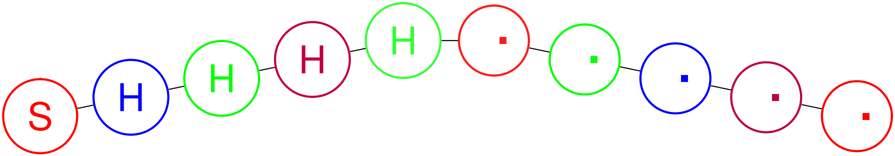
CS 170: Algorithms

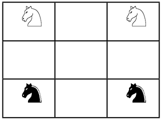


Prof David Wagner.
Slides edited from a version created by Prof. Satish Rao.
For UC-Berkeley CS170 Fall 2014 students use only.
Do not re-post or distribute

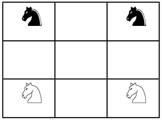
CS 170: Algorithms



Puzzle



Is it possible to reach this position?



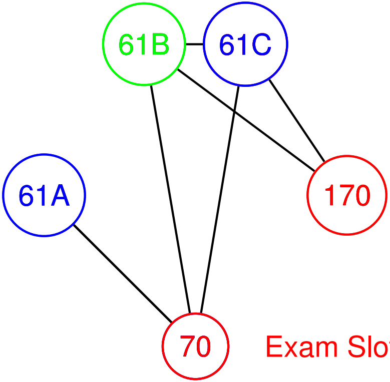
Design an algorithm to determine whether it is reachable.
Without looping forever.

Take 30 seconds to think about it *quietly* on your own.
Now work with someone next to you to solve this.

Today

1. Graphs
2. Reachability.
3. Depth First Search

Scheduling: coloring.

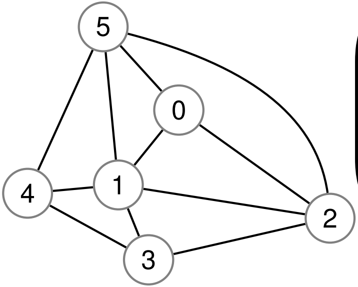


Exam Slot 1.

Exam Slot 2.

Exam Slot 3.

Graph Implementations.



Matrix Representation.

$$\begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 1 & 0 & 1 & 0 \end{pmatrix}$$

$$V = \{0,1,2,3,4,5\}$$

$$E = \{(0,1),(0,2),(0,5),(1,3)\ldots\}$$

Adjacency List

$0: \quad 1,2,5$
$1: \quad 0,2,3,4,5$
$2: \quad 0,1,3$
$3: \quad 1,2,4$
$4: \quad 1,3,5$
$5: \quad 0,1,2,4$

	Matrix	Adj. List
Edge (u,v)?	$O(1)$	$O(d)$
Neighbors of u	$O(\|V\|)$	$O(d)$
Space	$O(\|V\|^2)$	$O(\|V\|+\|E\|)$

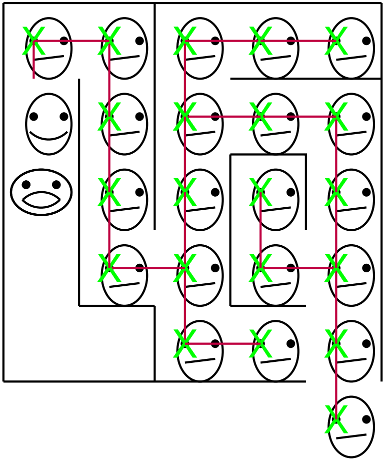
Exploring a maze.

Theseus: ...gotta kill the minatour ..in the maze
Ariadne: he's cute..fortunately ..she's smart.

Gives Theseus Ball of Thread and Chalk!

Explore a room:
Mark room with chalk.
For each exit.
  Look through exit. If marked, next exit.
  Otherwise go in room unwind thread.
    Explore that room.
Wind thread to go back to "previous" room.

Where is the minatour?

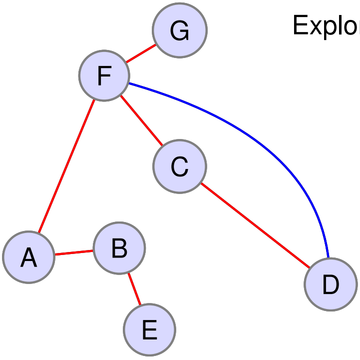


Searching

Find a minatour!

Find out which nodes are reachable from A.

Explore.



Explore(v):

1. Set visited[v] := true
2. for each edge (v,w) in E
3. if not visited[w]: Explore(w).

Chalk.

Stack is Thread.

Explore builds tree.

Tree and *back* edges.

Correctness.

Explore(v):

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.   if not visited[w]: Explore(w)

Property:

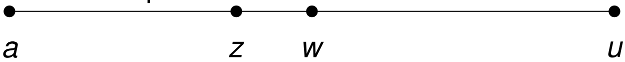
All and only nodes reachable from A are reached by explore.

Only: when u visited.

stack contains nodes in a path from a to u.

All: if a node u is reachable.

there is a path to it. Assume: u not found.

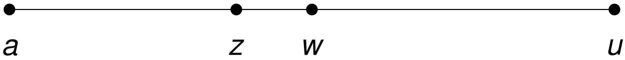


z is explored. w is not!

Explore (z) would explore(w)! Contradiction.

□

Proof was induction.



Property: Every node with a path of length k or less is reached.

Induction by Contradiction.
Find smallest k (path length) where property doesn't hold.
It does hold for $k-1$
So also for k
Must hold for every k.
Done!!! or $\square$.

Running Time.

Explore(v):

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.   if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

$T(n,m) \leq (d)T(n-1,m) + O(d)$ Exponential ?!?!?!

Don't use recurrence!

Running Time.

Explore(v):

1. Set visited[v] := **true**.
2. For each edge (v,w) in E
3.   if not visited[w]: Explore(w).

How to analyse?

Let $n = |V|$, and $m = |E|$.

"Charge work to something."
Put \$1 on each node, and \$2 on each edge, to pay for computation.

For node x:
  Explore once!
  Process each incident edge.

Each edge processed twice.

$O(n)$ - call explore on n nodes.
$O(m)$ - process each edge twice.
Total: $O(n+m)$.

Depth first search.

Process whole graph.

DFS(G)

1: For each node u,
2: visited[u] = **false**.
3: For each node u,
4: if not visited[u] **explore(**u**)**

Running time: $O(|V| + |E|)$.

Intuitively: tree for each “connected component”.
Several trees or Forest! Output connected components?

DFS and connected components.

Change explore a bit:

explore(v):

1. Set visited[v] := **true**.
2. previsit(v)
3. For each edge (v,w) in E
4.   if not visited[w]: explore(w).
5. postvisit(v)

Previsit(v):

1. Set cc[v] := ccnum.

DFS(G): 0. Set cc := 0.

1. for each v in V:
2.   if not visited[v]:
3.   explore(v)
4.   ccnum = ccnum+1

Each node will be labelled with connected component number.
Runtime: $O(|V|+|E|)$.

Connected Components.