Admin: Cubes out in recitation tomorrow!

Reading: CLRS 22.3-22.4

Outlines
- DFS-review
- Acyclic directed graphs (DAG's)
- Topological sort
- Parallel processes/critical paths
- Other searches: time-memory tradeoffs
  Simulated annealing

DFS - review:

```
R = set()
for u in V:
    if u not in R:
        R.add(u)
        visit(u)

def visit(u):
    for v in Adj[u]:
        if v not in R:
            R.add(v)
            visit(v)
```

DFS classifies edges: Tree edges (u to child) 1, 2, 3, 6

Non-tree edges:
- Back (u to ancestor) 4
- Forward (u to descendant) 5
- Cross (other to another subtree) 8
- self-loop 7
Let \( G = (V, E) \) be a directed graph.

\( G \) is **acyclic** if it contains no cycles. \((\equiv \text{DAG}, \equiv \text{directed acyclic graph})\).

**Thm:** \( G \) is acyclic \( \iff \) DFS of \( G \) produces no back edges (or self-loops).

**pf:** \( \Rightarrow \) if there is a back edge: \( u \rightarrow \text{ancestor of } u \), a cycle is created.

if self-loop: cycle exists.

\( \Leftarrow \) Say a vertex "finishes" when its visit call terminates.

**Lemma:** in DFS, all edges \((u, v)\) that are not back edges (or self-loops) have property:

\( u \rightarrow v \)

- tree edge (\( \text{visit}(u) \) calls \( \text{visit}(v) \))
- forward edge (\( \text{visit}(v) \) already done)
- cross edge

\( \therefore \) there are no cycles, since following a path must yield a earlier and earlier finishing times...
Proposition: We can tell in $O(V+E)$ time whether a directed graph $G = (V,E)$ is acyclic.

Proof: Run DFS, see if any back edges are produced.

To follow book: use colors:
- all vertices initially white

```python
def visit(v):
    color[v] ← gray  // start visiting v
    for w in Adj[v]:
        if w not in R:
            R.add(w)
            visit(w)
    color[v] ← black  // finished!
```

if $color[v]$ = gray: back edge found! $G$ is cyclic!

gray vertices are "in progress", represent current "breadth-first" path.

Application: "Topological Sort"

Given: an acyclic graph $G = (V,E)$

Produce: A list of its vertices

\[ v_1, v_2, \ldots, v_n \]

such that all edges go left-to-right.
V = set of things to do (tasks)
edge (u, v) means "have to do before you can do v"
topological sort ⇒ feasible order for doing tasks

Example:

Output:

(Many other feasible orders exist)
Parallel processing

Suppose now we can do many tasks at once (as long as pre-req tasks already done).
How can we find out how long it takes to do all tasks?

(Previous example: 4 time steps
Critical path is A → B → C → F)

Idea:
1. Produce topological sort
2. Take elements 1 at a time, put each in earliest time slot possible

Time 1 2 3 4
A B C D E F

(Many other solutions exist.)
(Exercise: modify DFS to produce this more directly...)
Other kinds of graph searches:

1. time-memory tradeoffs for one-way fn inversion

   \[ x \rightarrow f(x) \]

given \( f(x) \), how to find \( x \)?

- Preprocess graph, store a useful table of info \( \text{(indep of } x, f(x)\text{)} \)
- Then, given \( f(x) \), find \( x \) quickly
- But - not enough memory to store all \( (x, f(x)) \) pairs...? (cleverness req’d)

2. neighborhood search for optimum \( \text{ (simulated annealing) } \)

   graph \( G \) of states, undirected
   "energy" \( E(v) \) at vertex \( v \)

   want to find very low-energy vertex \( \text{ (optimum or near-optimum) } \)

   do walk on graph:

   - pick neighbor of current vertex
   - if downhill (less energy): go there
   - else: go up with probability depending on \( \Delta E \)
     and on \( T \) (current temperature)

   decrease temperature every so often
**Pseudocode for simulated annealing:**

\[ T = T_0 \hspace{1cm} \text{initial temperature} \]
\[ s = s_0 \hspace{1cm} \text{start state (initial vertex)} \]

while True:

\[
\begin{align*}
    s' &= \text{random neighbor of } s \\
    \Delta E &= E(s') - E(s) \\
    \text{if } \Delta E < 0 : s &= s' \\
    \text{else with probability } \exp(-\Delta E / T) : s &= s'
\end{align*}
\]

every so often: \[ T = T \times 0.99 \hspace{1cm} \text{(cool down)} \]

As \( T \to 0 \), uphill moves become less & less frequent

\( T \) helps particle "get over hills"

Many variations on these ideas...