CS 163 & CS 265: Graph Algorithms

Week 3: Paths

Lecture 8: Paths in DAGs

David Eppstein
University of California, Irvine

Fall Quarter, 2022

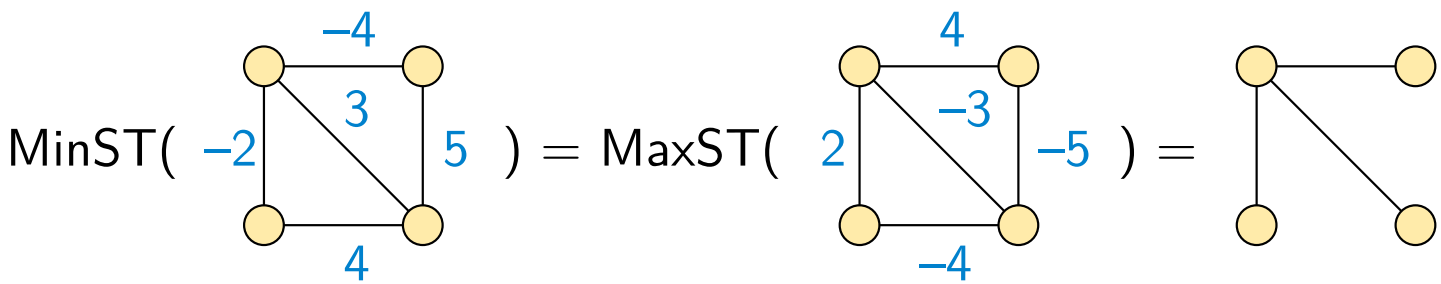

This work is licensed under a Creative Commons Attribution 4.0 International License

Longest vs shortest paths

When minimization and maximization are the same

In the minimum spanning tree problem,

- Minimum tree for weights $w(e)$
  $=$ maximum tree for weights $-w(e)$
- Algorithms don't care if numbers are positive or negative
- Can use same algorithms (with minor changes) for both minimum and maximum

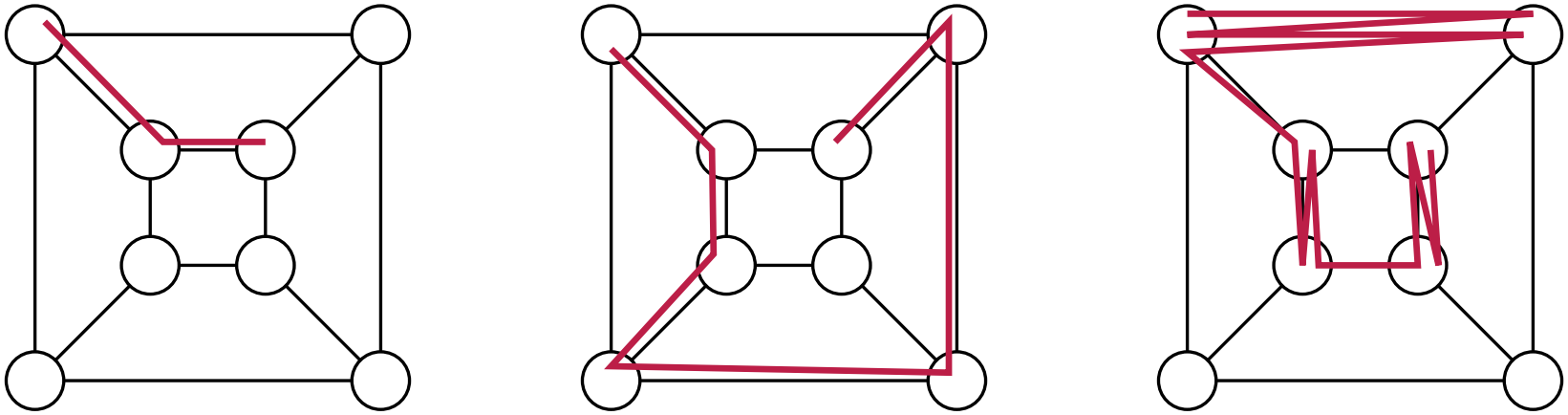


Same is true for shortest/ longest paths in directed acyclic graphs (today's topic; our application will use longest paths)

When minimization and maximization differ

For unweighted undirected graphs
(meaning that the length of a path is the number of edges)

- Shortest paths can be found by breadth first search
- Longest path is NP-hard (unlikely to have an efficient algorithm): includes Hamiltonian path through all vertices
- Longest walk doesn't generally exist (can go back and forth infinitely many times on same edge)

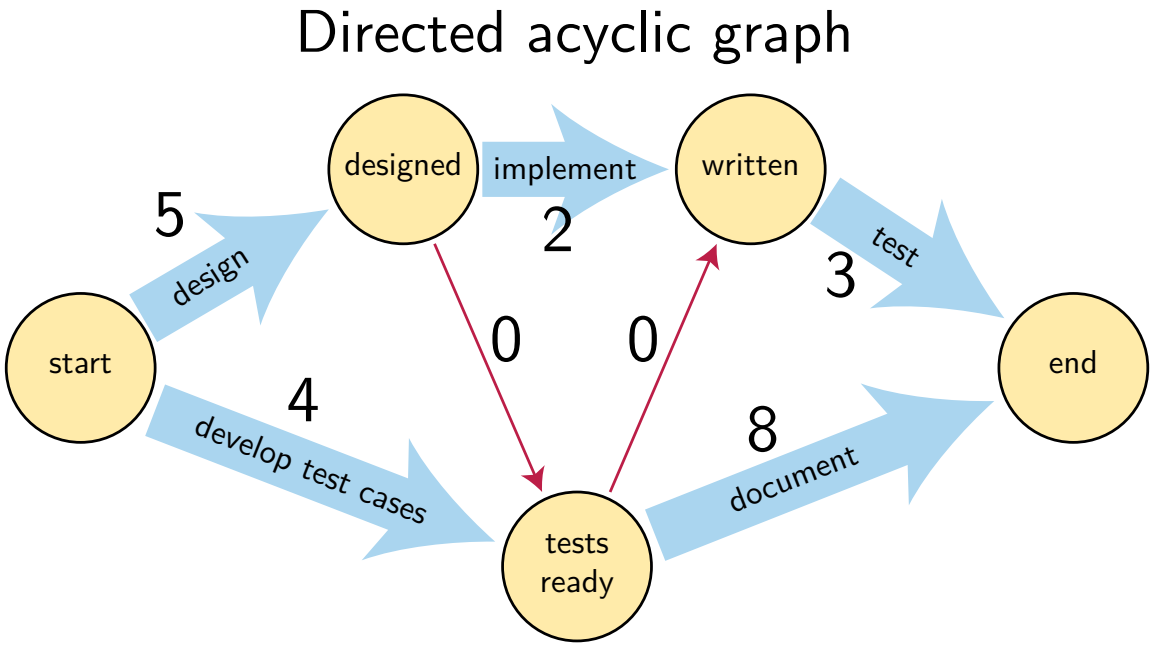


Critical path scheduling

Critical path scheduling ("PERT method")

- Input describes a project consisting of multiple tasks, each taking different amounts of time
- Some tasks depend on each other: certain tasks must be finished before others can start
- When two tasks do not depend on each other, they can both be active at the same time without slowing each other down (you have enough people working on the project to assign independent teams to each task)
- Goal: Find a schedule for all of the tasks that finishes the whole project in the minimum possible time

Activity-on-edge graph



Vertices = project milestones

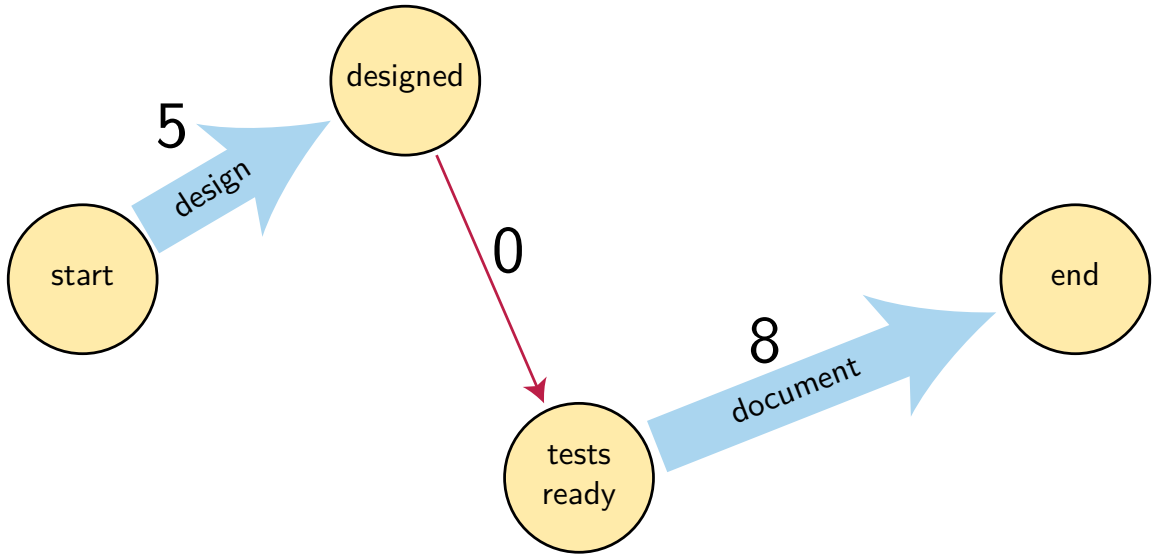
- Single start vertex has no incoming edges
- Single end vertex has no outgoing edges

Edges = tasks or ordering constraints

- Tasks (thick blue arrows) labeled by amount of time
- Constraints (thin red arrows) take zero time

Critical path

Longest path from start to end
(length = sum of edge labels)

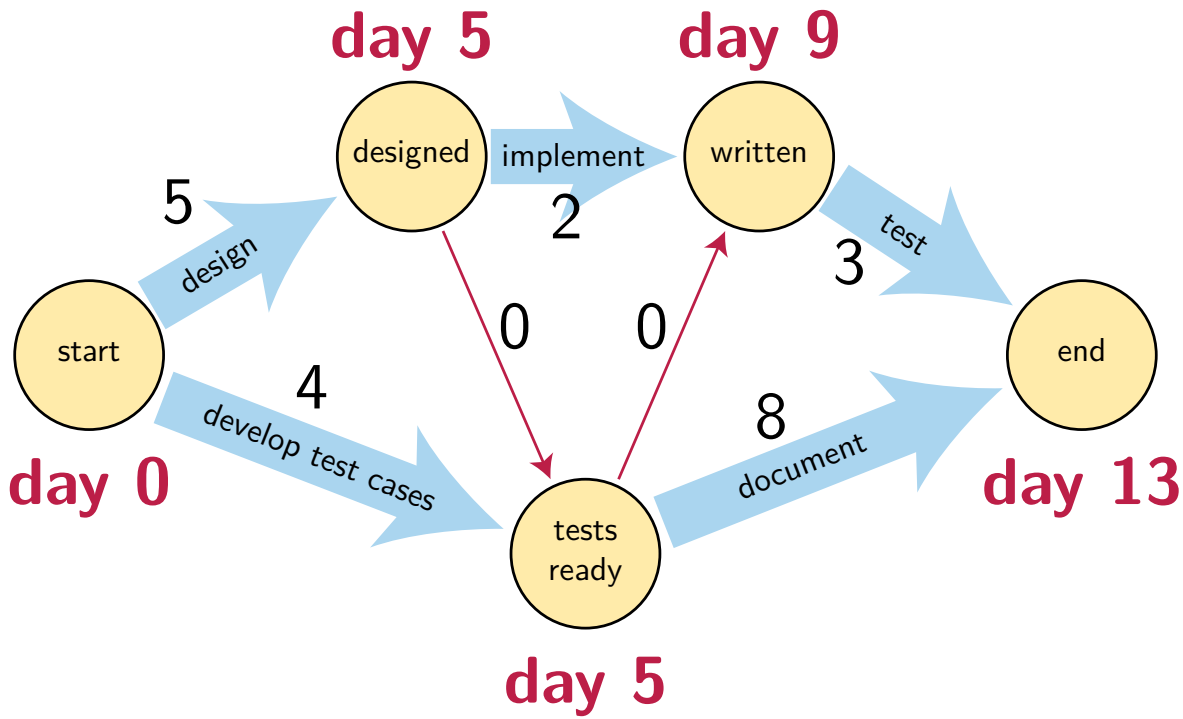


Determines how long whole project must take

What is a valid schedule?

Assignment of times to milestones

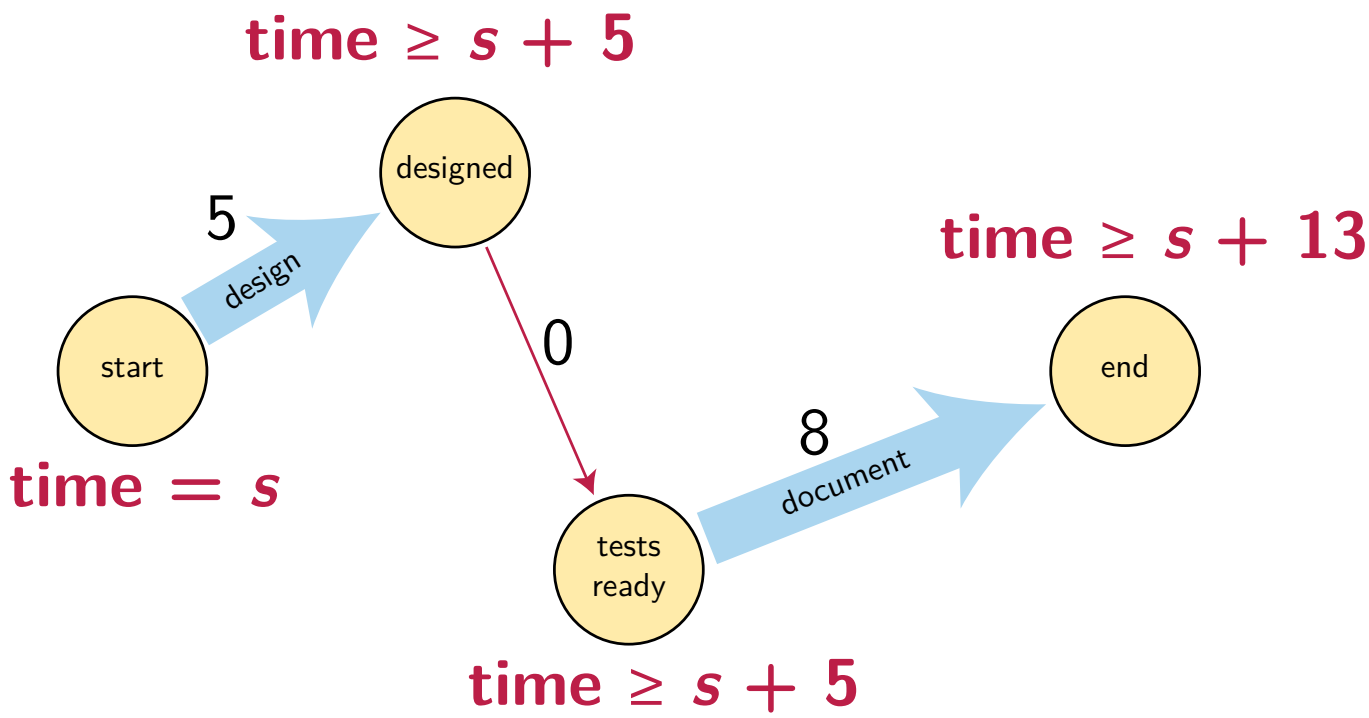
Tasks have enough time and ordering constraints are met:
For every edge, time at destination − time at source ≥ edge length



Total length = time at end vertex − time at start vertex

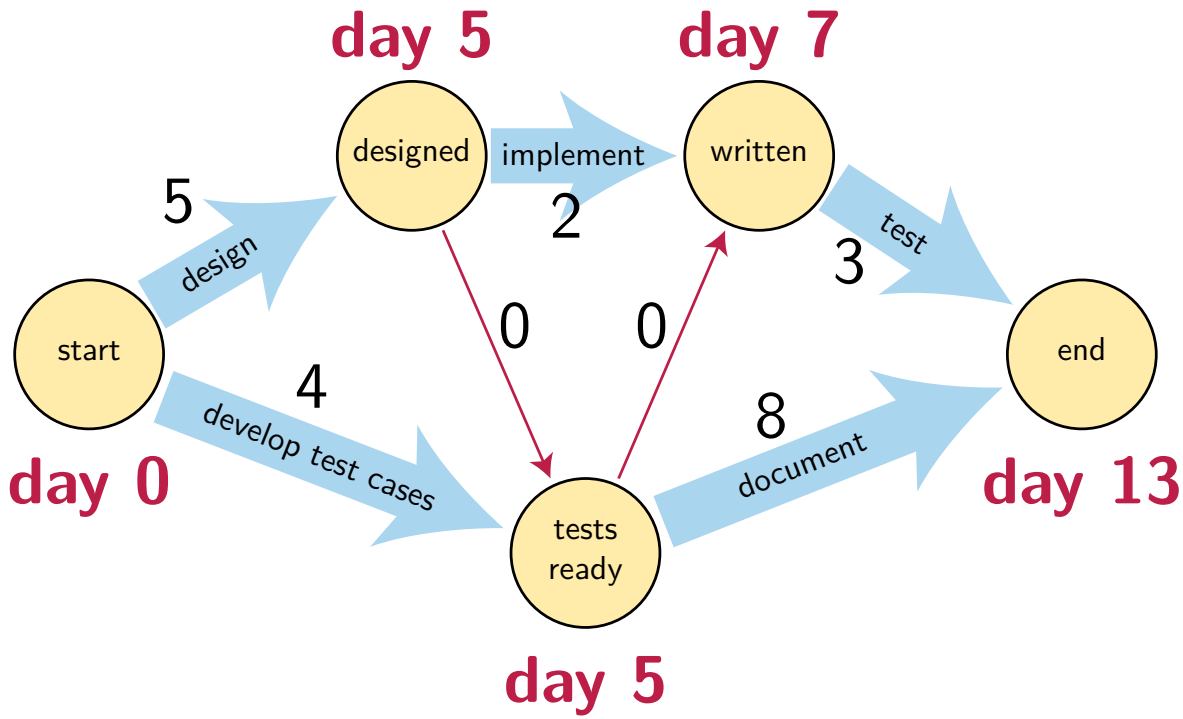
Valid schedule length $\geq$ critical path length

By induction on number of steps from start, each milestone must have a scheduled time $\geq$ length of the path from start



Optimal schedule length $\leq$ critical path length

Schedule each milestone at
time = length of longest path from start
(start milestone = 0, end milestone = critical path length)



Every edge has enough time, because otherwise it would be part of a longer path to its destination milestone

How to find the optimal schedule

L = empty dictionary

for each vertex v in topological order:

    if v is the starting vertex:

        $L[v] = 0$

    else:

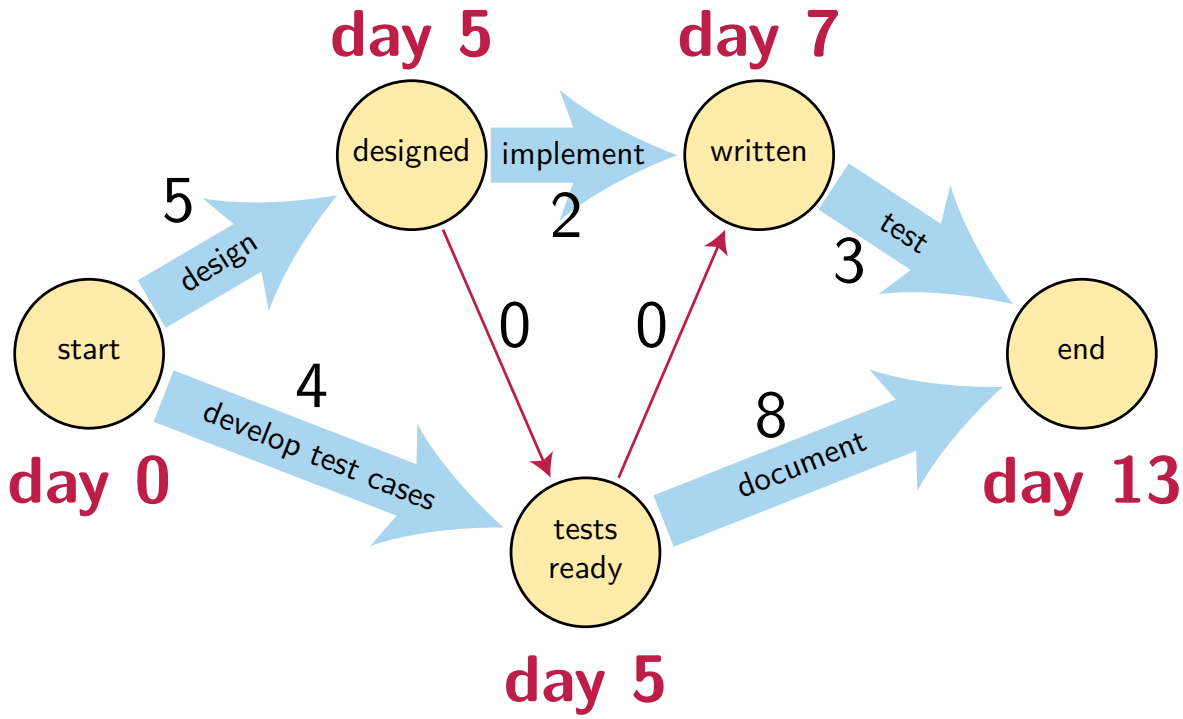
        $L[v] = \max_{\text{edge } u \to v} L[u] + \text{length}(u \to v)$

This computes longest path distances from start to all other vertices in linear time

Topological ordering implies that all predecessor vertices u already have the correct distance by the time we need them in the calculation for the distance for v

Change max to min in last line to get shortest path distances

Example of finding optimal schedule



Start: is starting vertex, $L[v] = 0$

Designed: edge from $u =$ start, $L[v] = L[u] + \text{length} = 5$

Tests ready: two in-edges, $L[u] + \text{length} = 0 + 4, 5 + 0$, max $= 5$

Written: two in-edges, $L[u] + \text{length} = 5 + 2, 5 + 0$, max $= 7$

End: two in-edges, $L[u] + \text{length} = 7 + 3, 5 + 8$, max $= 13$

How to find the critical path

(or one of the critical paths, if more than one equally long path)

pathlist = single-element list containing the end vertex

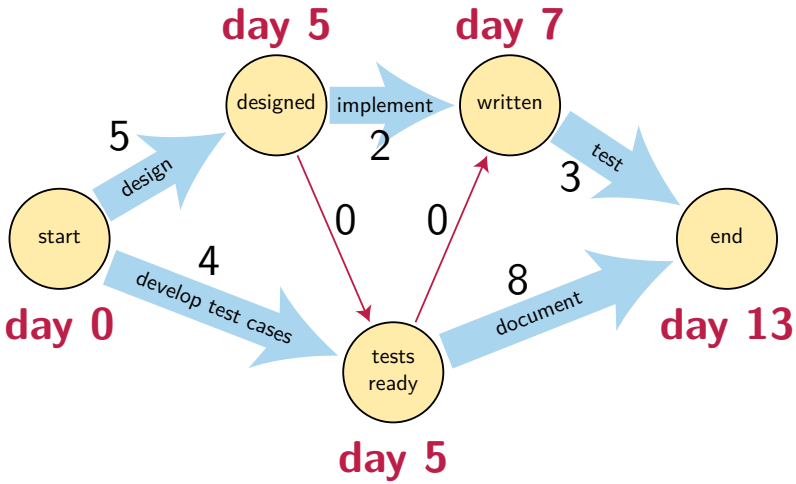
while the last vertex v in pathlist is not the start vertex:
    find an edge $u \to v$ such that $L[v] = L[u] + \text{length}(u \to v)$
    add u to end of pathlist

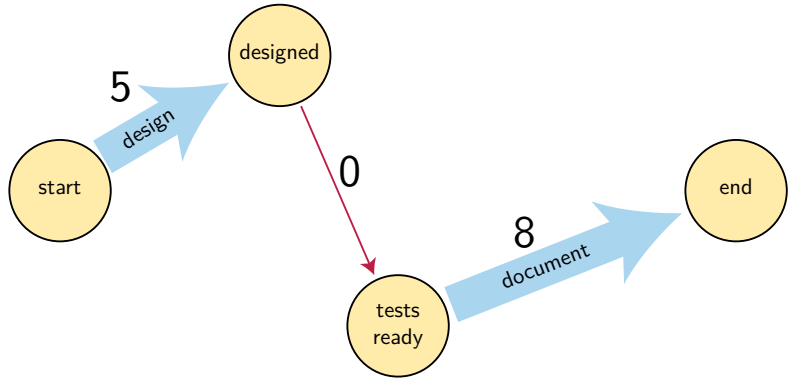
return reverse(pathlist)

This is just the standard backtracking method for turning a dynamic programming algorithm that computes the value of an optimal solution into an algorithm for finding the solution itself

Can test if there are multiple critical paths: true if we ever have more than one choice for u

Example of finding critical path





pathlist = [end]

edge to "end" with $L[u] + \text{length} = L[v]$ is from "tests ready"
pathlist = [end, tests ready]

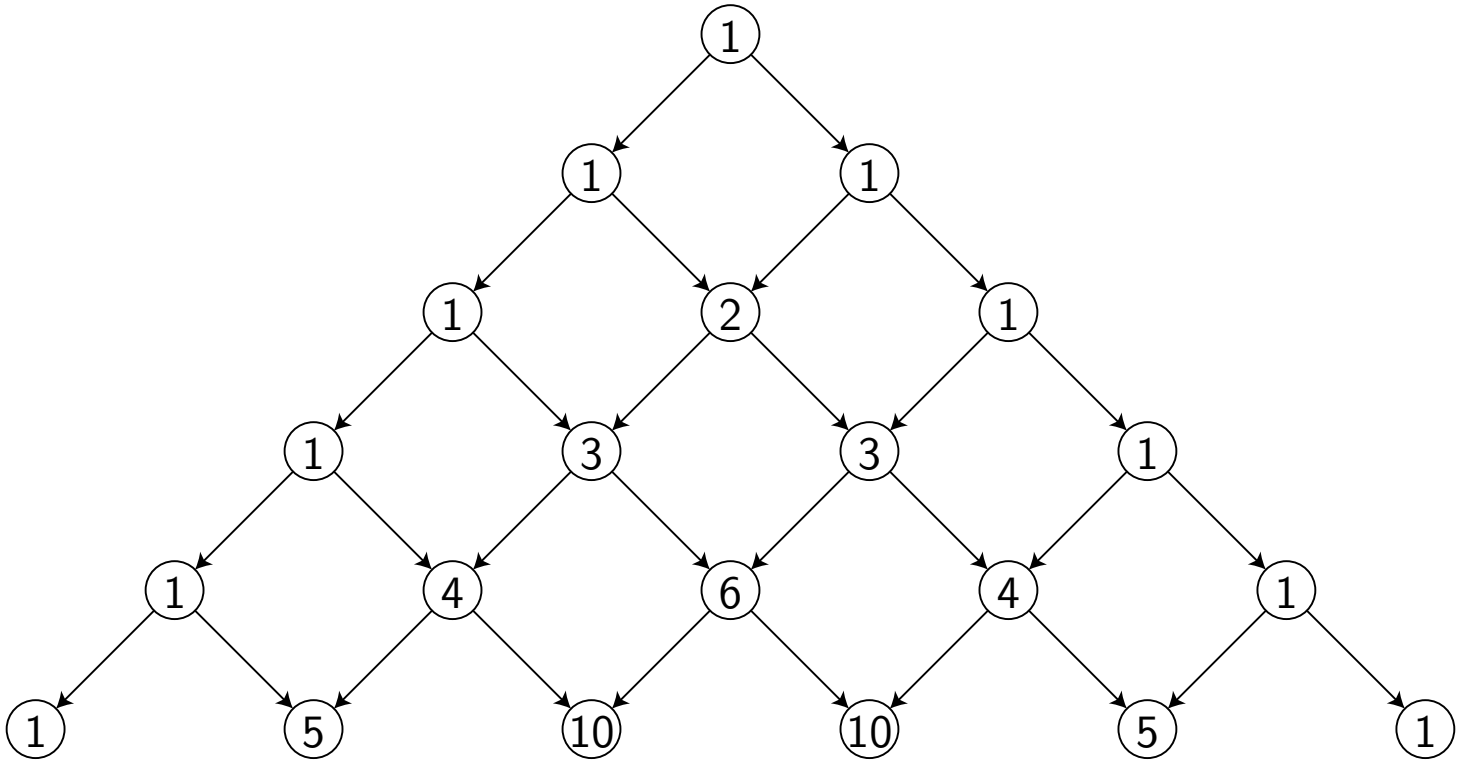
edge to "tests ready" with $L[u] + \text{length} = L[v]$ is from "designed"
pathlist = [end, tests ready, designed]

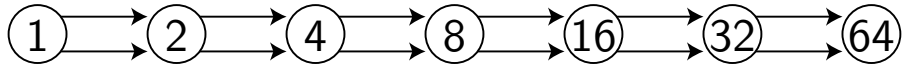
edge to "designed" with $L[u] + \text{length} = L[v]$ is from "start"
pathlist = [end, tests ready, designed, start]

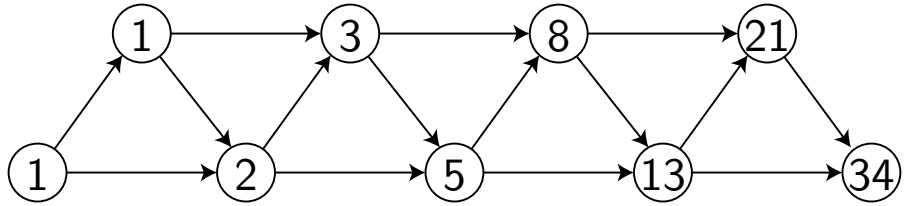
Reverse(list) is path: start $\longrightarrow$ designed $\longrightarrow$ tests ready $\longrightarrow$ end

Counting paths

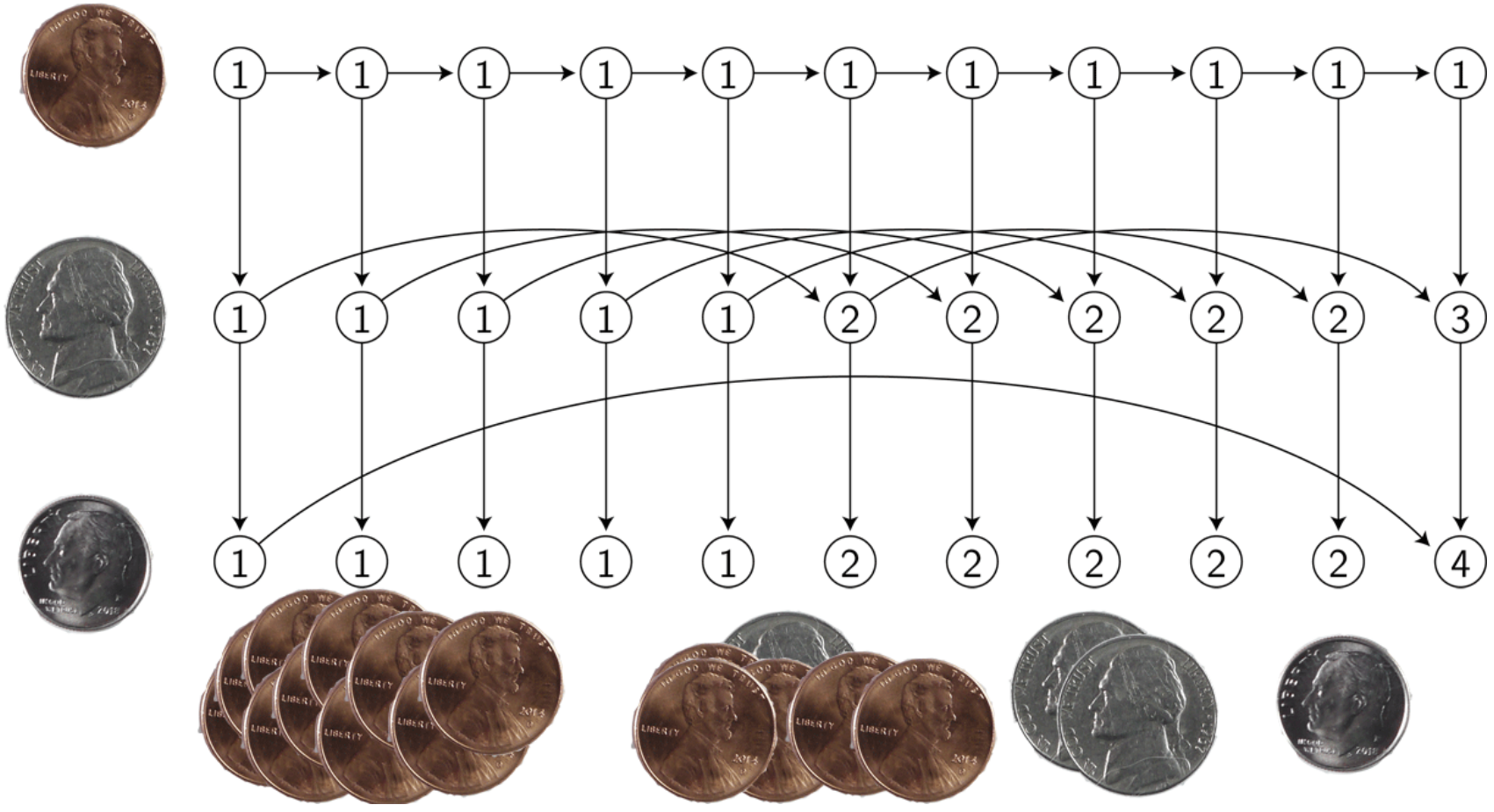
Pascal's triangle, powers of 2, and Fibonacci







How many ways can you make change for $0.10?



Based on CC-BY-SA image US coins front 01 by BrianStanding from Wikimedia commons

How to count paths

N = empty dictionary

for each vertex v in topological order:

    if v is the starting vertex:

        $N[v] = 1$

    else:

        $$N[v] = \sum_{\text{edge } u \to v} N[u]$$

This computes numbers of paths from start to all other vertices in linear time

Differences from longest-path algorithm are small:

- Base case $N[v] = 1$ instead of $N[v] = 0$
- Sum of numbers of paths instead of max of lengths of paths

Counting critical paths

An edge $u \to v$ belongs to a longest path from start to v exactly when $L[u] + \text{length}(u \to v) = L[v]$

Just use the path-counting algorithm on the graph formed by this subset of edges!

(Works equally well for counting shortest paths)

Speeding up a task will only help speed up the whole project when the task is on all critical paths. These tasks can be identified by counting paths: they are the ones for which

- $u \to v$ is part of a critical path, and
- #(longest paths start to u) $\times$ #(longest paths v to end) = #critical paths

Morals of the story

In directed acyclic graphs (but not other kinds of graphs)
shortest paths and longest paths are equally easy

Both can be found in linear time using topological ordering

Longest paths are important in project scheduling

Same algorithm can be adapted to many other DAG path problems