Data Structures and Algorithms Topological Order and Task Networks

Goodrich & Tamassia Sections 13.4

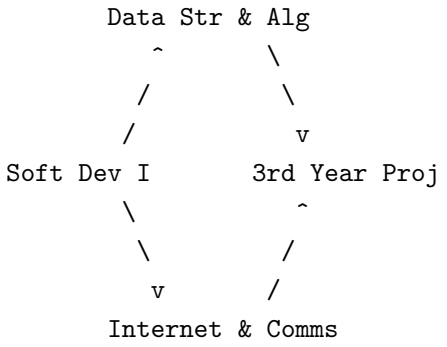
- Topological Order
- Task Networks
- Critical Path Algorithms

Topological Order

In many problems, one action depends on another being successfully completed, e.g.,

- Must pass Soft. Dev. II before doing Data Structs & Alg.
- Must paint ceiling before walls.

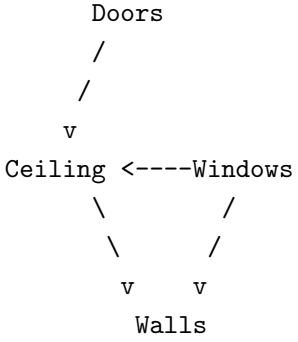
These relationships can be captured as a **directed acyclic graph** (DAG).



Painting

Rules: paint gloss before emulsion, ceiling before walls.

What's a valid order for painting doors, windows, ceiling & walls?

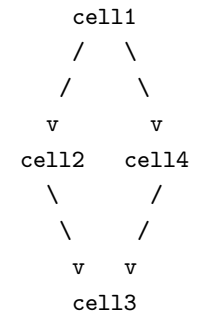


Spreadsheet Example

Cell	Contents	Value
1	100	100
2	cell1+10	110
3	cell2+cell4	310
4	cell1*2	200

In what order should the values in the cells be calculated?

Can represent *dependencies* between calculating cells as a graph:



Defining Topological Order

For all these examples, we want to find a possible order for the "doing" the activities represented by the vertices, obeying the dependency relations, and this is called a *topological order*.

Defn A topological ordering for the vertices of a digraph is a list of the vertices such that if there is a path from vertex A to vertex B, then A comes before B in the list.

There may be many topological orders, or none (e.g. for cyclic graphs)

Algorithm for Finding Topological Order

- For each node n, find *predecessors* of node n in the graph.

Node	Uncompleted predecessors
1	NONE
2	1
3	2 4
4	1

- Repeat:
  - Remove (output) node which doesn't depend on anything being done first.
  - Delete this node from lists associated with other nodes.

  Until all nodes output.

Simpler Version

- Calculate the *in-degree* of all nodes (ie, now many edges end up there) and store these in array D.
- Repeat:
  - Remove (output) node such that D[n]=0.
  - Decrement D[x] for all nodes x that are adjacent from n (edge from n to x).

  Exercise: Show how this algorithm works for painting example.

Task Networks and PERT Charts

Suppose we are managing a large project, and have to decide who does what, when, so it all gets done on time.

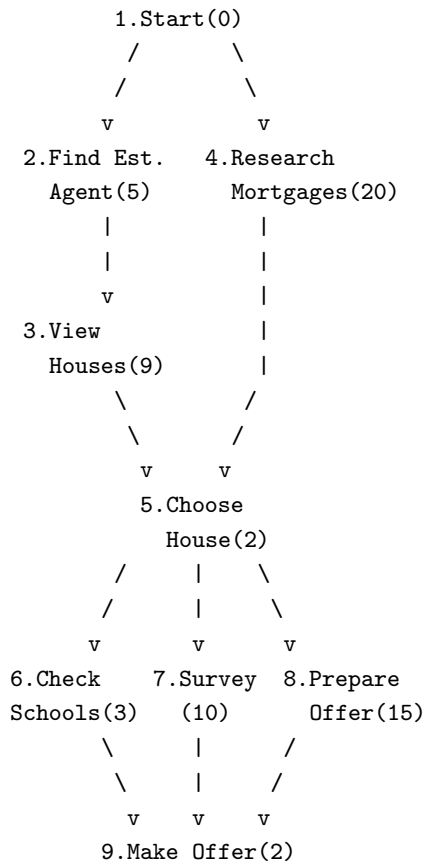
- Each project has a number of component activities called tasks.
- Each task has a duration (time to complete tasks).
- Each task is linked into network that specifies *predecessor* tasks (which must be accomplished before it can be started) and *successor* tasks which cannot be started until the task is complete.

This information can be illustrated in a PERT (Project Evaluation and Review Technique) chart.

PERT graphs and algorithms are also called the Critical Path Method (CPM)

They are widely used in project management, and supported by many tools e.g., MS Project will generate PERT charts.

Apollo moon project had 10,000 tasks!



Critical Path

Important concept is critical path:

- If the completion time of a task on the critical path slips, then completion time of whole project will slip! (What is the critical path in the example figure?)
- Tasks lying outside critical path have have some *slack time*. But If they exceed planned duration by more than their slack time, then they will hold up project.

Software Tools for Project Management

- There are lots of software tools developed to help people plan projects, using these basic ideas.
- May calculate critical path for you, and provide other views on project data (milestones, resource consumption etc).
- Underlying task network may be represented as a graph, and graph algorithms used to analyze it.
  - Directed acyclic graph, where nodes are labelled with task duration.
  - Referred to as *task networks*.

Computing the Critical Path

To compute the critical path we'll need to calculate, for each node:

- D[n] = duration of task for node n (given).
- EFT[n] = the earliest finish time for node n.
- LFT[n] = the latest finish time for node n (which won't hold up project).
- EST[n] = the earliest start time for node n.
- LST[n] = the last start time for node n (which won't hold up project).

Assume that "clock" starts at 0 (project starting time). We can only begin a task once all it's predecessors are completed.

Computing the Critical Path

- First thing to do is to calculate *topological order* for all the vertices in the task network.

  e.g.,: 1, 2, 4, 3, 5, 6, 7, 8, 9
- Also need the *predecessors* of each node.
- Then calculate earliest start and finishing times, doing calculations to nodes in topological order.

  If task n has no predecessors, then

  EFT[n]=D[n].

  Otherwise, if task n has predecessors:

  EST[n] = maximum(EFT[w]) for all predecessors w of n.
  EFT[n] = EST[n] + D[n]
- Maximum value of EFT[n] for all nodes n will give the earliest total project finish time (PFT).

Example

For house buying example:

- EFT[1]=EST[1]=0
- Node 2 has one predecessor, node 1, so EST[2]=0; EFT[2]=5.
- Node 4 has one predecessor, node 1, so EST[4]=0; EFT[4]=20.
- Node 3 has one predecessor, node 2, so EST[3]=5; EFT[4]=14.
- Node 5 has two predecessors, 3 and 4, so EST[3]=max(20, 14) = 20; EFT=22.

etc.

Computing the Critical Path

Once we have the best project finish time (PFT), we can find out the latest finish time of tasks that won't hold things up. We do this by looking at nodes in reverse topological order.

If task n has no successors then

LFT[n] =PFT
LST[n] = LFT[n] - D[n]

Otherwise, if task n has successors:

LFT[n] = minimum(LST[w]) for all successors w of n.
LST[n] = LFT[n] - D[n]

Computing Critical Path

After doing these calculations, we can find the *slack time* in each task: Slacktime[n] = LST[n]-EST[n].

If a task has a slack time of zero, that means it *must* be started on time if the whole project is to finish on time.

Tasks with a slack time of zero are on the *critical path*.

Summary

- Have looked at algorithms for processing graphs where nodes represent tasks or activities.
- Topological order - find an acceptable order for doing the activities, given that some must be done before others.
- Critical path - find how quickly you can complete a set of activities, and which are the critical activities.

Exercise: Weiss Exercise 9.1