

Data Structures and Algorithms II

Lecture 14: Task Networks and Topological order

- Topological Order
- Task Networks
- Critical Path Algorithms

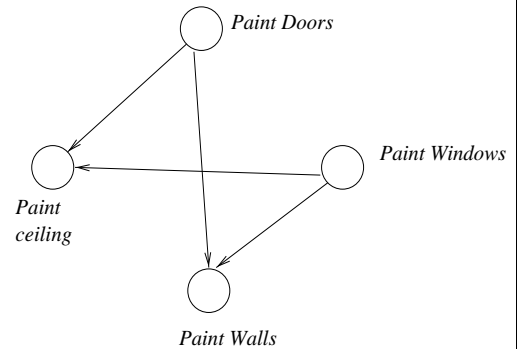
Corrections from last week:

- ◆ 113, slide2, 4th bullet: Should be “derived class t1..”.
- ◆ 112, slide11: Initialisation should be:

```
// initialise: shortest[startnode]=0;
// all other n - shortest[n]=inf;
```

1

Painting



(Paint gloss before emulsion; ceiling before walls)

What's a valid order for decorating?

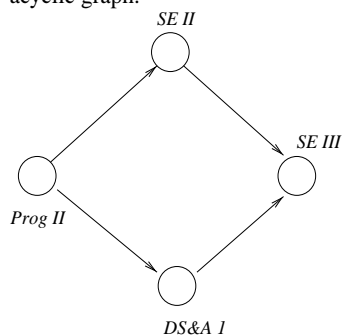
3

Topological Order

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

- Must pass programming II before doing SE II.
- Must paint ceiling before walls.

These relationships can be captured as a directed acyclic graph.



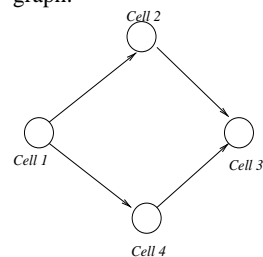
2

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* in calculating cells as a graph:



4

Algorithm for Finding Topological Order

For all these examples, we want to find a possible order for the “doing” the activities represented by the nodes, obeying the dependency relations.

Call this the “topological order”. Simple algorithm:

- 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.

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. Widely used in project management. e.g., Apollo moon project had 10,000 tasks!

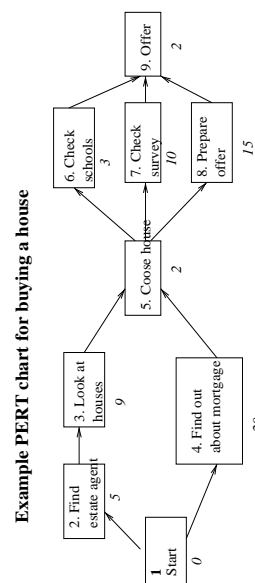
Simpler Version

- Calculate, for each node, the *in-degree* of that node (ie, how many edges end up there). Store these in array D.

- Repeat:
 - Remove (output) node such that $D[n]=0$.
 - Decrement $D[x]$ for all nodes x that are neighbours of n (edge from n to x).

.. Show how this algorithm works for doors/windows example.

PERT Charts Example



Critical Path

Important concept is critical path:

- If completion time on task in 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 will hold up project.

9

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.

11

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 analyse it.
 - Directed acyclic graph, where nodes are labelled with task duration.
 - Referred to as *task networks*.

10

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

12

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 $EST[3]=5$; $EFT[3]=14$.
- Node 5 has two predecessors, 3 and 4, so $EST[5]=\max(14, 20) = 20$; $EFT[5]=22$.

etc.

13

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*.

15

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:

$$LFT[n] = \text{minimum}(LST[w]) \text{ for all successors } w \text{ of } n.$$
$$LST[n] = LFT[n] - D[n]$$

14

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 get it all done, and which are the critical ones.

16