Data Structures with C++ : CS189

Lecture 11-1: Graphs Algorithms

Recap

- Graphs are a technology used to solve complex algorithms
- HasLoop/Topological Sort finds any loops in your graph
  - Used in operating systems to detect deadlocks
- DJ can find the shortest path from one node to another
  - A* adds in distance estimates to fix how DJ needs to process every node
  - Used in Google Maps and every game ever

HasLoop - Internal

- Every node gets a new int that starts with the normal of in pointers it has
- Find someone with a 0 in that tracker
- Loop through their outs and subtract one from all nodes on other side
- Mark this one -1 so it doesn't get picked again
- Keep finding 0's until you can't find any
- After that, if every node is -1 there is no loop
    - Every program ever should start with comments like these

HasLoop - External

- Make a map of VertexID to int
- Put all all verts in there with the number of ins as the int
- Loop through the map looking for a 0
- Loop through that vertex's outs and lower their numbers by 1
- Set the vertex's entry to -1 so it isn't picked again
- When you can't find any more 0's, if all nodes have a -1 there is no loop

HasLoop - Copy

- Make a copy of the graph
- Loop through verts looking for one with 0 pointers in its ins
- Call Remove
  - The updating of other nodes' scores is automatic as they literally lose in pointers
- If there are nodes remaining but none of them have 0 ins, there is a loop

HasLoop - Comparisons

- Internal is hard to use but easy to implement
- External is medium to use and medium to implement
- Copy is easy to use and hard to implement
- All are correct - at your job you'll consider speed, memory use, ease of implementation, safety, code standards, and which way makes enough sense for you to implement

Maximum Flow

- Instead of distance between nodes, lets use the weights as the capacity of a pipe
  - Remember the graph is a tool with no inherent meaning
- How much stuff (water, network traffic) can we push from a given node to another
- Easy case is three nodes in a row
  - Max is the smaller of the two pipes
- But what if there are multiple paths available?

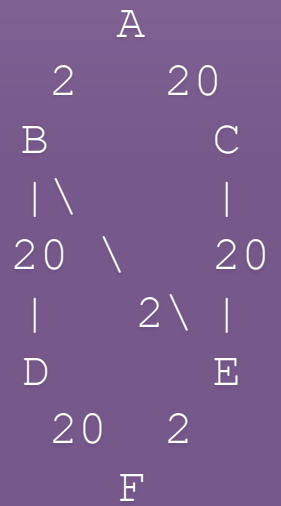
Greedy Algorithm for MaxFlow

- Use DJ to find a path through the graph
- Find the smallest link on that path
- Use one of the three techniques to mark that much flow as "in use"
  - Internal: Edges get a variable mCurWeight
  - External: Map of Edge to int
  - Copy: Lower actual weights or remove edges
- The sum of the ins to the sink is the answer
- One complication before we consider this algo finished though...

Flowing Backwards

- We are modeling flow, not a single lump
- Picture flow as a connection through the pipes, like a phone call
- It follows that anything that flows out could also flow back
  - It doesn't, but picture it
- So whenever the previous page reduced a weight by 2, it actually needs to mark the reverse direction as adding 2
  - Internal: int becomes mBackWeight
  - External: Map of Edge to struct with up/down
  - Copy: Literally add back links

Changing Flow Direction



- The "why" for that reversing is hard to describe except with a picture
  - Programmer art is the best kind of art
  - All edges are pointing "down"
- DJ says ABEF
  - Flow of 2, marked as taken
- DJ says we're done
  - But clearly the answer is 4, not 2
- But if we reverse the 2 from B to E instead of removing it, then ACEBDF finds the other 2
- Backlinks let us "change our mind"
- Answer is now sum of flow *in* to A

Final Version

- DJ path from source to sink
- Find smallest link on path
- Reverse (using one of the three techniques) that much flow
- Keep pathing until DJ fails
- Sum of weights that come in to the source is final answer
    - Only that much flow is moving, picture the rest as being stuck in dead end pipes

End

This is used to improve network performance, or city planning for sewers