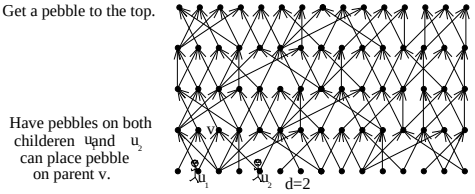
York University
CSE 5111 Pebbles Instructor: Jeff Edmonds

1. Iterative Pebbles (Loop Invariants): (Problems 1, 2, 3 should be done together.)

   For many computational problems there is a trade off between the time and the amount of memory needed. There are algorithms that use minimal time but lots of memory and algorithms that use minimal memory but lots of time. One way of modeling this is by a game of pebbling the nodes of a directed acyclic graph (DAG). Each node represents a value that needs to be computed. Each pebble corresponds to a memory register. Having a pebble on a node represents the fact that that value is stored in that memory register. Pebbles can be removed from a node and placed else where. This corresponds to replacing the value in the corresponding register with a newly computed value. However, there is a restriction. If there are edges from each of the nodes $In(v) = \{u_1, u_2, \ldots, u_d\}$ to v, then there needs to be a pebble on these nodes $u_1, u_2, \ldots, u_d$ before you can put a pebble on v. This is modeling the fact that you need the values associated with $u_1, u_2, \ldots, u_d$ in order to be able to compute the value associated with v. In contrast, a pebble can be placed at any time on a leaf of the DAG because these nodes have no incoming edges. These correspond to the easy to compute values. The input to the pebbling problem specifies a DAG and one of its nodes. The goal is to place a pebble on this node.

   As a working example, consider the DAG consisting of a rectangle of nodes w wide and h high. Here h much smaller than w. For each node v (not on the bottom row), there will be d nodes $In(v) = \{u_1, u_2, \ldots, u_d\}$ on the previous row with directed edges to v. Here, d is some small value.



   (a) Describe an extremely simple iterative algorithm that puts a pebble on the specified node as quickly as possible. The algorithm may use as many pebbles (memory) as needed, but does not redo any work. Give the key steps for developing an iterative algorithm as outlined in the class and handouts, i.e. start by defining the loop invariant.

   (b) How much time do you need for this iterative programming algorithm? Given that you are able to reuse pebbles, how many do you really need?

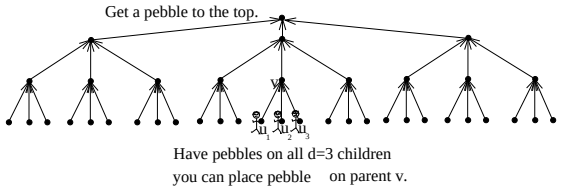
2. Recursive Pebbles: The problem is identical to that in question 1. Your task is to place a pebble on some node v which is r rows from the bottom.

   (a) Give **code** for a recursive algorithm that solves this problem. Your algorithm can redo work as necessary, but should use as few pebbles as possible. Your code should describe not only when a pebble should be placed on a node, but also when a pebble should be removed from node. Remember to get help from your friends.

(b) Let $Time(r)$ be the time used by your algorithm to go from there being no pebbles on the DAG to placing a pebble on one node r rows from the bottom. Give and solve a recurrence relation for this.

(c) Let $Pebbles(r)$ be the numbers of pebbles used by your algorithm. Argue why the recurrence relation is $Pebbles(r) = Pebbles(r-1) + (d-1)$. Solve this recurrence relation.

(d) Suppose the input for your recursive algorithm is not the matrix as described above but is a tree with n nodes. As before, each node (except leaves) has d children pointing at it. The height is h. What is the running time of this recursive algorithm?



3. Dynamic Programming Pebbles: The problem here is just slightly different than that in question 1 and 2. The input is the same rectangle graph except each edge has a positive weight. The *volume* of a path from a node v to the bottom is the *product* of the weights along the path. The task, given a node v, is to find the path from v to the bottom with largest volume.

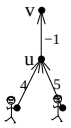
(a) Give the bird and friend algorithm for this. I know I taught you to ask the bird some question about the end of the solution, but to make things more consistent with the earlier questions ask about the beginning.

Give code for the dynamic programming algorithm.

(b) Suppose the weights on the edges could be negative. Does the same dynamic programming algorithm still work? Explain why. Hint: Consider the following input. In contrast, does the dynamic programming algorithm for shortest weight in a level graph work when the edges could be negative? Explain why.

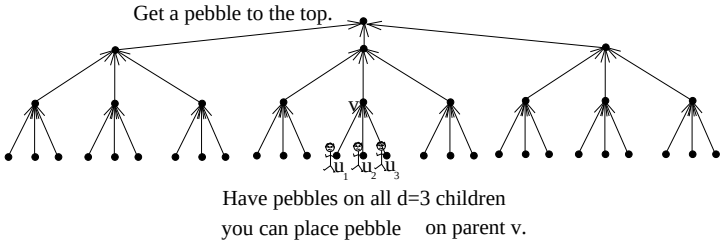
If you said yes, give a formal proof that it works. If you said no, give a simple way of fixing the dynamic programming algorithm to return the correct answer.

Example of max volume with negative weights.



4. Recall the pebbling game from an earlier question. You can freely put a pebble on a leaf. If there are edges from each of the nodes $In(v) = \{u_1, u_2, \ldots, u_d\}$ to v, then there needs to be a pebble on these nodes $u_1, u_2, \ldots, u_d$ before you can put a pebble on v.

   Consider a rooted tree such that each node (except leaves) has d children pointing at it. The height is h.



   (a) Give the Theta of the total number n of nodes in the tree. Show your math and give the intuition.

   (b) What is the connection between pebbles and memory?
   What does a node in the graph represent?
   What does an edge in the graph represent?

   (c) What is the minimum amount of time needed to get a pebble to the root even if you have lots of pebbles? Why?

(d) Briefly describe the recursive algorithm for placing a pebble on the root using as few pebbles as possible. Describe this algorithm using the friends paradigm.

(e) What is the number of pebbles for this recursive algorithm to get a pebble to the root? Give and solve the recurrence relation.

(f) What is the running time of this recursive algorithm? Compare it to that for the minimum time.

(g) Imagine you are tracing out this recursive algorithm. Stop the tracing at the point in time at which you need the most pebbles. Put an X on the nodes that have pebbles on them at that time.