Homework 6, CS561, Fall 2014

Aaron Gonzales

November 11, 2014

1 Naming Omicronians

Omicronians reproduce asexually with each baby producing up to two other babies. A birth process results in a binary tree, where babies are assigned their names in teh folloowing way. The baby at the root of the tree is assigned the name "J". Then any baby that is a left child of a node with the name σ receives the name σL and any baby that is the right child of that node receives the name σR.
Your goal is to take a binary tree specifying a birth process and return the number of R's in the names of all babies.

(a) For a node v let $n(v)$ be the number of nodes in the subtree rooted at v and assume this value is stored at each node. For a node v let $f(v)$ be the number of R's in the names of nodes below v. For simplicity, if v is NULL, let $f(v) = 0$ and let $n(v) = 0$ Also, for a node v, let $l(v)$ (resp $r(v)$) be the left (resp. right) child of v if it exists or NULL otherwise. Give a recurrence relation for $f(v)$.

Answer:

$$f(v) = f(l(v)) + f(r(v)) + n(r(v)) \tag{1}$$

(b) Your friend Bender claims that once you have the recurrence relation, you can just write a recursive algorithm for this problem in order to compute $f(r)$ where r is the root node. There's no need to bother with a dynamic program that stores the computed f values at the nodes of the tree. Is Bender right? Explain in a couple of sentences.

Answer:

Yes, he is - there is no need to store all the previously calculated values in a table or other structure, you only need the current count.

(c) The Omicronians have evolved!

Now they can reproduce both sexually and assexually: i.e., each node can have 1 or 2 parents. So a birth process is now a rooted directed graph without cycles. Now each node can have multiple names, one for each path from the root down to that node. A particular path determins a name in the same way as before: start with J at the root and add a L whenever a left edge is taken and a R whenever a right edge is taken.
Professor Farnsworth claims that the same recurrence relation from part a can be used for this problem. Bender claims that this problem can also be done recursively, without the dynamic programming approach of storing f values at intermediate nodes. Which if either of your friends are correct? Explain briefly.

Answer:

Farnsworth is right. We need to store the f values at each node since the number of parents may be different.

2 Bins, Probability, and Expectation

There are two bins. Bin 1 initially has 3 white balls and 1 red ball. Bin 2 has 4 white balls. In every round, a ball is selected uniformly at random

from each bin and these two balls are swapped.
Let p_k be the probability that the red ball is in bin 1 at the beginning of the k-th round.

(a) Write a recurrence relation for p_k.

Answer:

Note that p_k is the probability that the red ball is in bin 1 at the beginning of the k-th round.

We can write this approaching from two angles, one if the red ball is in the right bin and another if the red ball is in the left bin at the beginning of the kth round. We know that if, at any round, the chance of the red ball moving from the right bin to the left if it is in the right bin is $\frac{1}{4}$ and the chance of it staying in the left bin if it is in the left bin is $\frac{3}{4}$.

A recurrence relation follows:

$$\begin{aligned} p(k) &= \frac{1}{4}\left(1-p(k-1)\right)+\frac{3}{4}\left(p\left(k-1\right)\right) \\ p(k) &= \frac{1}{4}-\frac{1}{4}p(k-1)+\frac{3}{4}p(k-1) \\ p(k) &= \frac{1}{2}p(k-1)+\frac{1}{4} \end{aligned}$$

(b) Use the guess and check and proof by induction method to solve this recurrence, Don't forget to label BC, IH, and IS and clearly say where you are using the IH

.

Answer:

First, we solve the recurrence.

$$\begin{aligned} p(k) &= 1/2p(k-1)+1/4 \\ p(k+1) &= 1/2p(k) \\ p(k)-1/2p(k) &= 0 \\ Lp-1/2p &= 0 \\ (L-1/2) &= 0\text{annihilates the sequences homogeneous part} \end{aligned}$$

$(L-1)$ annihilates the sequences non-homogeneous part, leaving us with $(L-1/2)\,(L-1)$.

Solving for the constants gives us:

$$f(n) = c_1 * (1/2)^n + c_2$$

$$\begin{aligned} &\text{base cases :} \\ p(1) = 1 :& f(1) = c_1(1/2)^1 + c_2 = 1 \\ p(2) = 3/4 :& f(2) = c_1(1/2)^2 + c_2 = 3/4 \end{aligned}$$

$$\begin{aligned} c_1 &= (1/2)^2 + (1-1/2c_1) = 3/4 \\ c_1 &= \Big((1/2)^2-(1/2)\Big) \\ &= 3/4-1 \\ &= 1 \end{aligned}$$

$$c_2 = 1 - 1/2c_1$$
$$c_2 = 1 - 1/2 * 1$$
$$c_2 = 1/2$$

We can say that

$$p(k) = \left(\frac{1}{2}\right)^k + \frac{1}{2}$$

and use it as a guess to prove via induction.

Proof. let $f(n) = \frac{1}{2}f(n-1) + \frac{1}{4}$ represent our recurrence. suppose that :

$$f(n) = \left(\frac{1}{2}\right)^n + \frac{1}{2}$$

We have two base cases:

$$n = 1 : f(n) = (1/2)^1 + 1/2 = 1$$
$$n = 2 : f(2) = (1/2)^2 + 1/2 = 3/4$$

Inductive Hypothesis:

$$\forall j < n, f(j) = (1/2)^j + 1/2$$

Inductive step:

$$\begin{aligned} f(n) &= 1/2f(n-1) + 1/4 \\ &= 1/2\left((1/2)^{n-1} + 1/2\right) + 1/4 \\ &= 1/2\left(\frac{(1/2)^j}{1/2} + 1/2\right) + 1/4 \\ &= (1/2)^j + 1/4 + 1/4 \\ &= (1/2)^j + 1/2 \end{aligned}$$ □

(c) Assume you are paid one dollar for every round in which the red ball is in bin 1 and there are m rounds. What is the expected number of dollars you earn?

Answer:

Let X_k be an indicator random variable that denotes the event when the red ball is in bin 1. We define X_k as

$$X_k = \begin{cases} 1 \text{ if red ball is in bin 1 at beginning of round } k \\ 0 \text{ otherwise} \end{cases}$$

as such, we have:

$$
\begin{aligned}
E[X_k] &= Pr[X_k = 1] = \left(\frac{1}{2}\right)^k + 1/2 = \frac{1}{2^k} + 1/2 \\
E(x) &= E\left[\sum_{k=1}^{m} X_k\right] \\
E(x) &= \sum_{k=1}^{m} [X_k] \quad \textit{By linearity of expectation} \\
E(x) &= \sum_{k=1}^{m} \left(\left(\frac{1}{2^k}\right) + 1/2\right) \\
E(x) &= \sum_{k=1}^{m} \left(\left(\frac{1}{2^m}\right) + 1/2\right) \\
E(x) &= m\left(\left(\frac{1}{2^m}\right) + 1/2\right) \\
E(x) &= \frac{m}{2^m} + \frac{m}{2}
\end{aligned}
$$

3 Tree Induction

Prove via induction that any tree over n nodes has exactly $n - 1$ edges.

Answer:

Printed output 1. Any tree over n nodes has exactly $n - 1$ edges.

Proof. By induction on n.
Let E be the number of edges in a tree.
Base case: when $n = 1$, clearly $e = 0$ (there are no other nodes in the tree).
Inductive hypothesis:
The Theorem is true for all trees with fewer than n vertices.

Let T be a tree with n nodes and e be an edge from a node u to a node v. Only one path exists between u, v and that is e. As such, if we delete e, the tree T is disconnected. $T - e$ represents the tree T without the edge e which can be represented by two subcomponents of T, T_1, T_2. Each subcomponent is a tree, and let n_1, n_2 be the number of nodes in each subcomponent such that $n_1 + n_2 = n$.

Inductive step:
By the inductive hypothesis, the number of edges in T_1, T_2 are $n_1 - 1, n_2 - 1$ respectively and it follows that the tree T has

$$E = n_1 - 1 + n_2 - 1 + 1 = n_1 + n_2 - 1 = n - 1$$

□

4 Claim 1 from SSSP

- **if $dist(v) \neq \infty$, then $dist(v)$ is the total weight of the predessor chain ending at v:**

$$s \to \cdots \to pred(pred(v)) \to pred(v) \to v$$

- **This is easy to prove by induction on the number of edges in the path from s to v.**

Answer:

Proof. By induction on n where n is the number of edges in the path from s to v.

Base Case:

$$s = v : dist(v) = 0$$

Inductive Hypothesis:

$$\forall j < n : dist(j) = dist(n) = \text{ total weight of pred ending at } v$$

Inductive Step: $(s \to x)$ where $x = pred(v)$. By the inductive hypothesis, $dist(x) = pred(v)$ from $s \to x$, therefore $dist(v) = dist(x) + w(x, v)$ □

5 Claim 2 from SSSP

- **If the algorithm halts, then $dist(v) \leq w(s \rightsquigarrow v)$ for *any* path $s \rightsquigarrow v$.**
- **This is easy to prove by induction on the number of edges in the path from s to v.**

Answer:

Proof. The definition of relaxation follows:

Definition 5.1. We call an edge (u, v) tense if $dist(u) + w(u, v) < dist(v)$ If (u, v) is tense, then the tentative shortest path from $(s \rightsquigarrow v)$ is incorrect since the path $(s \rightsquigarrow u)$ and then (u, v) is shorter. Our generic algorithm repeatedly finds a tense edge in the graph and relaxes it. If there are no tense edges, our algorithm is finished and we have our desired shortest path tree. [1]

Let k be the number of edges in $s \rightsquigarrow v$. By definition, our algorithm, upon finishing finds the shortest path tree. If we define $j < k$ as j being the number of edges in $s \rightsquigarrow u$ where u is a vertex in the path efore v. If our algorithm relaxes all edges in the path $s \rightsquigarrow u$, then we have a shortest path from $s \rightsquigarrow u$ and any tense edges from $u \rightsquigarrow s$ will be relaxed, and at termination, we will have a shortest path from $s \rightsquigarrow v$. This path will have $dist(v) \leq w(s \rightsquigarrow v)$ as the path $s \rightsquigarrow u$ had $dist(u) \leq w(s \rightsquigarrow u)$ and we add the one new relaxed edge from u, v to the path $s \rightsquigarrow u$. □

6 Problem 23-4: Alternative MST

In this problem, we give pseudocode for three different algorithms. Each one takes a connected graph and a weight function as input and returns a set of edges T . For each algorithm, either prove that T is a minimum spanning tree or prove that T is not a minimum spanning tree. Also describe the most efcient implementation of each algorithm, whether or not it computes a minimum spanning tree.

Before the problems, first a few definitions:

Definition 6.1. SpanningTree A tree is a connected undirected graph with no cycles. It is a spanning tree of a graph G if it spans G (that is, it includes every vertex of G) and is a subgraph of G (every edge in the tree belongs to G). A spanning tree of a connected graph G can also be defined as a maximal set of edges of G that contains no cycle, or as a minimal set of edges that connect all vertices. [2]

Definition 6.2. Minimal Minimum spanning tree. An edge-weighted graph is a graph where we associate weights or costs with each edge. A minimum spanning tree (MST) of an edge-weighted graph is a spanning tree whose weight (the sum of the weights of its edges) is no larger than the weight of any other spanning tree. [3]

For all below answers, we have to show that for a an algorithm to return a minimal spanning tree, it must meet both of the above criteria.

[1] from the SSSP lecture notes
[2] verbatim from wikipedia, not from CLRS
[3] from princeton algs book

(a)

```
Maybe–MST–A(G, w)
  sort the edges into nonincreasing order of edge weights w
  T = E
  for each edge e, taken in nonincreasing order by weight
    if T−{e} is a connected graph
      T = T − {e}
  return T
```

Answer:

analysis

MaybeMSTA removes edges in nonincreasing order as long as the graph stays connected which results in a tree T which is an MST. If we let S be an MST $S \in G$ If we remove an edge from g, $e \in S$ or $e \notin S$. If $e \in S$, removing e disconnects the tree S into two subtrees. If a larger edge existed that connected the subtrees with lower weight than edge e, then the algorithm would have removed it before e (nonincreasing order by weight). The graph is still connected, so anotehr edge with equal weight hasn't been discovered yet, safely allowing removal of e. The algorithm returns a tree that is both a spanning tree and is minimal.

implementation

represent T with an adjacency list and construct a priority queue from the egdes. Use union find with union-by-rank unions and path compression. We get a total running time of $O(Elog(E))$.

(b)

```
Maybe–MST–B(G,w)
  T = null
  for each edge e, taken in arbitrary order
    if union(T, {e}) has no cycles
      T = union(T, {e})
    return T
```

Answer:

analysis:

No, the algorithm does not return an MST.

The graph in Figure 1 would have edges $\{a, b\}, \{b, c\}$ with weight 3. As cycles are taken in an arbitrary order it could try to add the edge $\{a, b\}$ to the previous picked sets $\{c, a\}, \{c, b\}$ which would add a cycle and result in a spanning tree, but that tree wouldn't be a minimal spanning tree.

implementation:

Use a union-find data structure to hold T in similar fashion to Kruskal's algorithm. We make V calls to MakeSet, $2E$ find-set operations, and $V - 1$ union operations, which gives us a $O(Vlog^{*}V + E)$ running time.

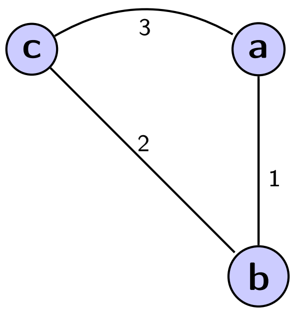

Figure 1: Figure blah

(c)

```
Maybe-MST-C(G,w)
  T = null
  for each edge e, taken in arbitrary order
  T = union(T, {e})
  if T has a cycle c
    let e_prime be a maximum weight edge on c
    T = T - {e_prime}
  return T
```

Answer:

analysis:

T will add every edge $e \in G$ and only edges e that make cycles will be deleted. If a graph has an edge that results in a cycle, removing that edge removing that edge not disconnect the graph. Because T has every edge added at some point and only delete edges that create cycles, T is a spanning tree.

Adding any edge within an MST creates a cycle, and T is built in a way such that any edge that is added that creates a cycle is discarded, T has the minimal number edges that comprise the set of edges that make a minimal spanning tree. The algorithm ensures that the maximal weighted edge within a cycle is removed, and following these steps, T is both a spanning tree and minimal.

implementation

Use an adjacency list for T and when adding an edge to T, use DFS to check it for a cycle. If there are no cycles for the edge, we keep it in T and continue. If there is a cycle, we have to detect it's maximum-weighted edge and remove it from T. DFS takes $O(V+E)$ which in this algorithm is bounded by $O(V)$ as the edges in T are never more than V since cycles are removed. For each edge we have to perform DFS and this results in $O(E*V)$ running time.

7 Problem 24-2 Nesting Boxes

a d-dimensional box with dimensions $(x_1, x_2, \ldots x_d)$ nests within another box with dimensions $(y_1, y_2, \ldots y_d)$ if there exists a permutation π on $\{1, 2 \ldots, d\}$ such that $x_{\pi(1)} < y_1, x_{\pi(2)} < y_2, \ldots x_{\pi(d)} < y_d$.

(a) Argue that the nesting relation is transitive.

Answer:

let $\mathcal{R} \subseteq SxS$ be a relation in S. In order to show that a relation is transitive, we must show that for

$$(x,y) \in \mathcal{R} \wedge (y,z) \in \mathcal{R} \implies (x,z) \in \mathcal{R}$$

$$\{(x,y),(y,z)\} \subseteq \mathcal{R} \implies (x,z) \in \mathcal{R}$$

In our problem, we must show that x nests in y and y nests in some box z. If we find a relation π for both of the relations above such that

$$x_{\pi_i} < y_i \text{and } y_{\pi_i} < z_j$$

if our number of dimensions is the set d We have $\{i,j\} \in d$, we can find for every i a unique j that satisfies $i = \pi_j$, resulting in

$$x_{\pi_i} < y_i = y_{\pi_i} < z_j$$

which reduces to

$$x_{\pi_i} < z_j$$

and the relation is transitive as we can find a unique z_j for every x_{π_i}.

(b) Describe an efficient method to determine weather or not one d-dimensional box nests inside another.

Answer:

If we sort the d dimension values within both $(x_1, x_2, \ldots x_d)$ and $(y_1, y_2, \ldots y_d)$ and then compare x_i, y_i for every value of $i = 1 \to d$ If $x_i < y_i$ for all i, this implies that x nests within y.

(c) Suppose that you are given a set of n d-dimensional boxes $\{B_1, B_2, \ldots B_n\}$. Give an efficient algorithm to find the longest sequence $\langle B_{i_1}, B_{i_2}, \ldots B_{i_k} \rangle$ of boxes such that B_{i_j} nests within $B_{i_{j+1}}$ for $j = 1, 2, \ldots, k-1$. Express the running time of your algorithm in terms of n and d.

Answer:

As before, sort the dimension values d within each box $(x_1, x_2, \ldots x_d)$ and $(y_1, y_2, \ldots y_d)$ Check if each possible pair of boxes (B_i, B_j) B_i nests inside of B_j or if the converse is true. Make a graph G with n nodes where each node represents the one B_i, so that each B_i, B_j pair where B_i nests in B_j gets a directed edge from $B_i \to B_j$. Add a root vertex s to the graph and give an edge from r to every vertex in the graph. Add a 'destination' vertex d to the graph that has an incoming edge from each vertex. Preform a topological sort of the graph with s as the root vertex. Run a Bellman-Ford SSSP search on $s \to d$. The most expensive part of the algorithm is building the graph, which involves $O(dn^2)$ operations, which is greater than the sorting time $O(ndlogd)$ or search time $O(V+E)$.

8 Saia Trucking

Saia Trucking is a very saftey concious (and algorithm loving) trucking company. They always try to find the safest route between any pair of cities. They are thus faced with the following problem.

There is a graph $G = (V, E)$ where the vertices represent cities and the edges represent roads. Each edge has a value associated with it that gives the probability of safe transport on that edge i.e. the prbability that there will be no accident when driving across that edge. The probability of safe transport along any path in the gtraph is the product Π of the probabilities of safe transport on each edge in that path.

The Goal is to find a path from s to t that maximizes the probability of safe transport. Describe an efficient algorithm to solve this problem.

Answer:

We have s as our start node and t as our destination. let $e(u,v)$ be the edge from vertex $u \rightarrow v$. We want to maximize the product of the probabilities on the path such that the set of vertices $p = \langle v_0, v_1, \dots, v_k \rangle$ where $v_0 = s, v_k = t$. We want

$$p = max \prod_{i=0}^{k} e(v_{i-1}, v_i)$$

If we log transform the edge weights, we can solve this problem easily. If we define $w(e) = -log\, e(u,v)$ as the 'negative' log transform of the edge weight, we can change the problem to a single-source shortest path problem as

$$p = min \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

This is easily solved with Dijkstra's algorithm with the additional step of log transforming the edges ($O(E)$). Using a Fibbonacci-heap-backed min-priority queue and an adjacency list to represent the graph, we can do this is total $O(|E| + |V| log|V|)$ time. If we wanted to extend the problem to show all pairs of shortest paths between cities, we can use Floyd-Warshall in $O(V^3)$ time or take advantage of the fact that there are no negative edges in our graph and run Dijkstra's algorithm from each vertex, giving a runtime of $O(|E| + |V^2| log|V|)$.

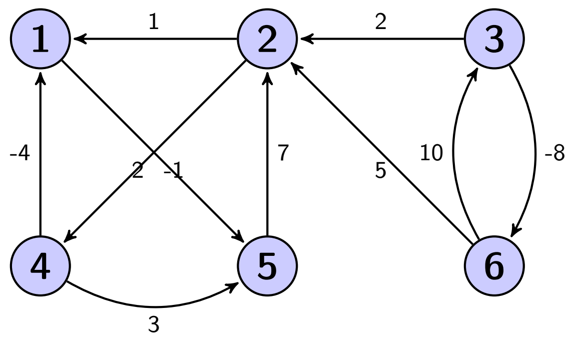

Figure 2: Figure 25.2 from CLRS

9 25.2-1 - Floyd—Warshall

Run the Floyd—Warshall algorithm on the weighted, directed graph of Figure 2. Show the matrix $D^{(k)}$ that results for each iteration of the outer loop.

Answer: