F.G.H.
6/25/85

LispCourse #39: Solutions to Homework #38

Solution functions are attached.

Notes on Problem C

Problem C requires a ***breadth-first search*** of the family tree. In particular, starting at the node for the given person, you want to check all one-away relatives (e.g., parents and children). If none of these match, you want to check all two-away relatives (i.e., all one-away relatives of the one-away relatives). And so on until you find an N-away relative that matches or you are out of relatives.

> This contrasts with a ***depth-first search***, where you would search all of the person's mothers relatives first, then his or her father's relatives, then his or her childrens releative (where children are considered in some arbitrary order). For each of the mother's relatives, you would first search that persons' mother's relatives, then their father's and so on. Basically, you search one branch of the family tree until it ends, then search the next branch and so on.

In a *breadth-first* search, at each step you have the a set of N-away relatives, none of whom match the thing you are searching for. For each of these N-away relatives, you want to compute their 1-away relatives (i.e., the N+1-away relatives). While you are computing the N+1-away relatives for each of the N-away relatives, you need to maintain lots of information ž in particular, you need to keep a list of the N-away relatives that still have to be worked on AND you need to maintain a list of the N+1-away relatives already discovered (who will be "expanded" during the next round if no match is found among the N+1 relatives).

> The best way to maintain this information is in a global queue (see Homework #32 & LispCourse #33).
>
> > Just keep repeating the following until the queue becomes empty:
> >
> > > Take a person off of the head of the queue.
> > >
> > > If he matches what you're looking for, you're done ž exit.
> > >
> > > If there's no match, expand the person into a set of his 1-away relatives and add all these relatives to the end of the queue.

Note that with this scheme, you don’t need to keep explicit track of what level (i.e., K-away or L-away, or whatnot) you are working on. Because you always add the N+1-away to the end of the queue but take the next person to "expand" from the front of the queue, you are certain to process all the N-away relatives before you start on the N+1 away relatives. The moment you find a match, you just stop and you can be sure that you’ve found the closest relative that matches.

> However, if there are more than one matching relative at the Nth level, you will find only one and then stop. To find all closest relatives in case of a tie requires some small variations on this scheme.