CPSC 320 2016W2: Quiz 1 Pre-Release Information

January 8, 2017

This week's quizzes will follow up on the problem domains described below. It's worth spending a few minutes reading and understanding each domain before your tutorial!

Note that if you collaborate on understanding this information, you should follow the academic conduct guidelines. Among other rules, take down the names of collaborators for acknowledgment, but take **no other notes** away from those collaborations!

NOTE: The names of problems and (on the quiz and assignment) subproblems are often intended to be fun and are completely irrelevant. Feel free to ignore them here, on the quiz, and on the assignment!

1 Save the Last Dance for Someone Other Than Me

You're arranging a formal dance evening. n people have signed up to lead and n people to follow. There will be a series of k dances (with $k \leq n$), each of which will match each leader with one follower. However, the same leader and follower will never be paired for more than one dance. (So, every person will dance with k different people across the k dances.) Your job is to create the series of k matchings for the k dances.

As with SMP, you have complete preference lists over the followers for each leader and complete preference lists over the leaders for each follower. (Where leaders and followers in this problem play similar roles to men and women—or vice versa—in SMP.)

2 THX 1138

A (conveniently-sized) set of $2n$ actors are applying for n roles in a play. For each role, an actor is needed as well as an *understudy*, someone who plays the role if the main actor has problems. Each actor has a preference list over the roles. Each one would rather have any of the roles than any of the understudy positions; however, if they have to take an understudy position, they have the same preference order for these as for the roles. The casting director is in charge of hiring the actors. The casting director has a list of preferences over the actors for each role (and has the same list of preferences for each understudy position as for the corresponding role). We call this problem THX (the THeatre eXtension to the Stable Marriage Problem).

We want to find a solution to THX that is *stable*.

3 a Bag of Make Me Words

A standard part of text analysis—for machine learning, machine translation, sentiment analysis, and so on—is transforming a document into a "bag of words" representation. A "bag of words" is a data structure that maps the unique words in the document to the number of times each one appears. For example, the phrase "make me a hot dog, a chili dog, and a bag of words" would become a 'bag' like: [a:3, and:1, bag:1, chili:1, dog:2, hot:1, make:1, me:1, of:1, words:1], with each word and its number of appearances. (We put the words in sorted order, but that's not necessary.) Note that the phrase has 13 words total but only 10 unique words (because "a" is repeated twice and "dog" once in the original phrase).

Specifically, you want to create an algorithm that—given a list of words W containing m words total but only n unique words, where $n \leq m$—produces a complete list of tuples (pairs) of words and their numbers of occurrences. The result should have n entries (exactly one for each unique word) and the total of the numbers of ocurrences across all entries should be m, but these entries can be in any order.

Most approaches we consider will use hash tables. For our purposes, such a hash table supports 6 operations. Here they are described for a standard hash table using chaining:

ConstructHashTable(e) creates and returns a hash table of size e. (The hash table will be empty, but its underlying array will have e entries. Takes time proportional to e.)

Size(T) returns the number of keys in hash table T. (Takes constant time.)

Contains$(T,\ k)$ returns true if hash table T contains an entry for the key k and false otherwise. (Takes expected constant time, worst-case linear time.)

Insert$(T,\ k,\ v)$ inserts key k with value v into hash table T. If k is already in T, overwrites its value with v. (Takes expected constant time, worst-case linear time.)

Find$(T,\ k)$ finds key k in hash table T and returns the value associated with it. If k is not in T, produces an error instead. (Takes expected constant time, worst-case linear time.)

Entries(T) returns a list of all the key/value pairs in hash table T. (If the table currently has n keys (i.e., Size$(T) = n$) and its underlying array has e entries, this takes time proportional to $n + e$.)