20 Questions Design Document\README

Miles Izydorczak

December 2019

1 Data Structures and Algorithms

The primary data structure used by most of my program is a K-dimensional tree. Essentially each “entry” (Something that the computer is trying to guess) is represented as a binary sequence of 0s and 1s depending on their validity as an answer to each of the questions. A K-dimensional tree views these sequences as points in K-dimensional space, and tries to slice this space along a dimension that will most evenly divide the points in half.

This is represented internally as a tree where each node contains a “pool” of entries that are in the region of K-dimensional space that we have narrowed the possible entries down to. Non-leaf nodes also contain a question which can be selected in a number of different ways as I will outline later. Non-leaf nodes have pointers two other nodes: a ”yes” node that contains the entries for which the answer to the question is yes and a ”no” node that contains the entries for which the answer is no.

Versions of Project

My program can be compiled with “make 20Q” and different versions of my program can be ran as follows:

Version 0.0

- Run with ./20Q -v0.0 entriesFile questionsFile
- This version’s method of choosing questions to divide the data is very rudimentary. It first divides the questions with the first question given in the questions text file, the second division is done with the second question given in the file, and so on.
- While this method works, for eventually enough questions will be asked that the K-space has been segmented into regions that contain one point each, we end up wasting a lot of our guesses asking questions that divide the pool very little or not at all.

- For example a question such as ”Is it a freshman dorm” is pointless if we already know that we are guessing a Tufts building that is not a residential hall.
- In summary, while this version works, we can do better which is where the next algorithms come from.

Version 1.0

- Run with ./20Q -v1.0 entriesFile questionsFile
- This version now asks the best question, i.e. the question which divides the data points most closely to a 50-50 split.
- This is done by iterating through each question and each entry to determine which question has the answer for closest to half of the pool of entries.
- This method is a vast improvement from version 0.0, for now we are not wasting guesses, and are reaching the answer in the fastest way possible. However, this version has worse runtime than the prototype version.
- The process of picking the best question involves parsing through the entire set of entries, and we can do better than this.

Version 1.1

- Run with ./20Q -v1.1 entriesFile questionsFile
- This version sacrifices a little bit of question efficiency for a big improvement to the runtime. Now, when the pool has a greater size than ten, rather then pick the best question from among the entire pool, we make a random sample of ten entries and find the question that best divides those entries.
- An interesting side effect of this version is that it is more enjoyable to play because the human is not asked the same exact questions in the same order each time they play.

Version 1.5

- Run with ./20Q -v1.5 entriesFile questionsFile
- This version is an alternative way of improving the runtime of version 1.0. Here, we track a parallel vector of the total sum of entries whose answer is ”yes” for each of questions, and update these sums as points are removed from the pool.
- Here we can plainly check which total is closest to half of the size of the pool as we add each node without having to iterate through the entries

Version 2.0

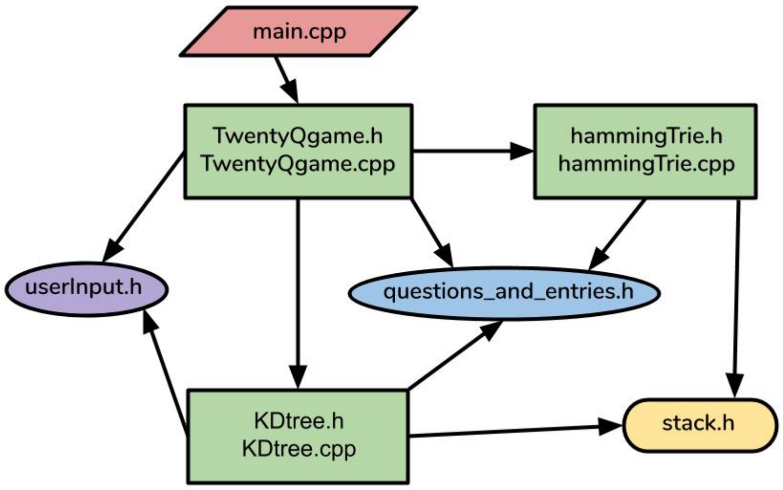
- Runs with ./20Q -v2.0 entriesFile questionsFile
- Version 2.0 marks the first major change in how the program operates since version 1.0. Here we seek to tackle another issue that my program may have: what happens if someone answers a question wrong? Maybe they have a good excuse for why they answered the way they did, that the creator of the data set did not account for.
- Now instead of merely yes and no, the player can enter yes, no, probably, probably not, and unknown.
- The tree structure is very handy here because we can store the addresses in memory of Nodes where the player answered one of the "unsure" answers (probably, probably not, and unknown) in a stack.
- Then when we reach a Node with a pool size of 1, ask our guess, and the player replies that our guess is incorrect, we can look at the top of our stack of "unsure" answers and return to that Node and try the other way.
- The Nodes which are marked "unknown" take precedent over the other two "unsure" answers because if a player doesn't know how to answer the question, it is more likely that this is where they made an error.
- Additionally I have taken measures to ensure that the same question will never be asked more than once in the game. When we change branches in the tree, we skip questions that have already been answered, while placing new "probablies" in the stack where needed.

Version 3.0

- Runs with ./20Q -v3.0 entriesFile questionsFile
- Version 3.0 uses an alternate method to get around the players potentially answering a question incorrectly.
- We use another data structure here- a trie used to find the similar entries to the one given. First we ask entries that have a Hamming distance of one- those whose binary sequence has one element different from the given. If we still haven't found the correct answer, we repeat this for entries that have a Hamming distance of two

2 Project Layout

Upon starting up, my main function will create an instance of the TwentyQgame class which is the driver class that the player interacts with. Depending on the version requested, a different KDtree will be built, and if version 3.0 is chosen, an instance of the hammingTrie class will also be created. userInput.h, questions_and_entries.h and Stack.h are header files used by these classes.



Description of Files

main.cpp

Reads in command line arguments, calling and running an instance of the TwentyQgame class.

TwentyQgame.cpp

Implementation file of the TwentyQgame class, this object reads in data from the given text files, creates an instance of the KDtree class made with the requested version. From there it handles all of the user input and output- asking questions and then telling the KDtree how to advance.

TwentyQgame.h

Implementation barrier of the TwentyQgame class.

hammingTrie.cpp

Implementation file of the hammingTrie class. This includes functionality to read in a guess, then find neighbors of that entry with a Hamming distance of 1 then 2, and storing these in a stack to be guessed one at a time.

hammingTrie.h

Interface of the hammingTrie class.

KDtree.cpp

Implementation file of the KDtree class. Contains various functions for building the tree in different ways, and then advancing through it with or without the inclusion of probablies.

KDtree.h

Interface of the KDtree class.

userInput.h

Declares an emum with 5 values: yes, no, prob, probnot, and unknown.

Stack.h

A standard stack implementation, but it is a template class so I can use it to store anytype that I need.

questions_and_entries.h

Declarations for two of the essential structs in the project: a Question and an Entry.

prototype_es.txt and prototype_qs.txt

A very simple data set where each of the three questions divides the data perfectly in half.

buildings_es.txt and buildings_qs.txt

One example of a data set that my program can guess from, this contains data points for each of 55 well-known buildings on Tufts' Medford-Somerville campus

presidents_es.txt and presidents_qs.txt

Another example of a data set that my program can guess from, this contains questions to divide the 44 entries of United States presidents

3 Changes Since Original

The progression of my project can be seen clearly as the versions increase in number. The main way my project has evolved, however, is that the initial stretch goals I had was not the direction that I took my project. At first I thought that I was going to try and make it so that if an answer was not in the database, then the player could add it in post win, but I ended up taking it more in the direction of accounting for player error, which proved to be a pretty interesting problem to tackle.

The main thing that I got out of this assignment was how to design a large project on my own. The biggest thing I learned was that as good as it was to spend hours designing, it proved to be a lot easier (and funner) once I started coding prototypes, even though I knew that a lot of those prototypes would not last until the final version. I thought it was wise to keep track of the progression through these prototypes by storing them as different versions.

4 Time Spent

I did not keep track of the time that I spent on the project, but I would estimate about 40-45 hours. A break down of this would be 5 preliminary hours of designing and writing the specifications, 30 hours of actual coding, 5 hours of writing data sets and testing with those, and 5 hours writing function contracts, polishing up the style, and writing this design document.

5 Acknowledgements

I had four meetings over the past two weeks with TAs Lawrence Chan and Kevin Destin where they advised me on how to proceed with my program. They gave me ideas as to how to proceed with my program, such as the "probablies" version, and the Hamming distance version. It was both helpful and necessary to speak some of my ideas out loud because many of them were not feasible in the three week period that I had to make this. I started out lost in my own aspirations for where I could take the program, and they gave me the advice to start small with a prototype version. As soon as I did this, I felt a lot more confident that I could make something good by the end of those weeks, and began to implement features one at a time. Thanks so much guys!!!

6 Suggestions for Future Independent Projects

This experience was awesome. As a person who enjoys making my own rules over following those of others, the freedom and responsibility that came with changing my spec as I went and saw what worked and didn't was very cool. I had a blast! That said, I could see why someone might not want to undertake this project because not following a guideline does mean that there is no guarantee that the project can be done. Maybe to encourage more interest, someone could make a pizza post with different thematic ideas for projects to inspire people.

I also talked to one of my friends who said that she had an idea for a project but that was a week after the proposal had been due. My recommendation then would be to post the independent project advertisement a week earlier so that people can have more time to think of an idea. I do understand that the course staff might not want people to be focused on a different project when, say, homework 5 was currently active.