EC504 2019 Project Suggestions – Pick one or discuss another topic with Instructor:

Additional information input files for a few projects are on the course GitHub PROJECT_INFORMATION (Request for additional information on other projects can be made in class or on Slack

Next Tuesday Nov 12 you must have a team (2 to 3 students) and send in couple of slides describing the project plan. Be able to explain in class what you are trying to do. Of course it maybe changed, probable down sized, as you realize time constraints!

Below is a list of project suggestions. In all cases, you are to deliver a report documenting:

a) Algorithms and data structures used in the project
b) Experimental results documenting accuracy of response and sample outputs

You will also deliver prototype code along with instructions for compilation and execution so your work can be tested independently on new inputs. Instructions for this report are included at the end of this document.

1) Project #1 Nearest State/County Finder

You are given a huge number of reference points in the US, (We will use data extracted from the official US Board on Geographic Names dataset as input (you can download this dataset from course GitHub at PROJECT_INFORMATION)).

Your system will first load the reference points in an efficient nearest neighbor search amenable data structure. Then it will allow users to query your data structure by entering a decimal latitude and a decimal longitude. You will be asked to return the nearest K reference points, where K is a number from 1 to 10. Also, you will be asked to find the state and county of the point by computing a majority voting among the 5 nearest points.

For distance computation between two points use the "equirectangular approximation" (http://www.movable-type.co.uk/scripts/latlong.html), which can be defined as:

$x = (\lambda2-\lambda1) * \text{Cos}((\phi1+\phi2)/2);$

$y = (\phi2-\phi1);$

$\text{Distance} = \text{Sqrt}(x*x + y*y) * R;$

where ϕ is latitude, λ is longitude, R is earth's radius (mean radius = 6371km); In particular, you are to perform the following two tasks:

- loading the province, state, decimal latitude, decimal longitude data into your data structure

- accepting and responding to user queries efficiently and accurately

For example, assuming that the reference points are as follows:

STATE_ALPHA	COUNTY_NAME P	RIM_LAT_DEC	PRIM_LONG_DEC
AR	Benton	36.4805825	-94.4580681
AZ	Apache	36.4611122	-109.4784394
AZ	Maricopa	33.2486547	-112.7735045
AZ	Graham	32.4709038	-109.9361853 …

Given the query lat: 33.24, long: -112.75, the nearest reference point would be in AZ Maricopa with approximately 67.05 km distance.

You are free to use any data structure you like for storing the reference points. You must define the complexity of search in your system. Efficient data structure selection & usage will get higher points.

Minimum Requirements [60%]

• Command line user interface that lets loading of reference points, and querying of coordinates for state and county with a given K nearest neighbors.

Possible Extensions [15%] Implement alternative approaches and compare speed of response for different data structures.

Comparative Features [25%] Your algorithms and data structures will be compared against others in the class and ranked according to: Speed of search operation.

2) Project # 2. Twitter Keyword Search

Write a keyword search engine that accepts a text file containing a number of tweets as input (each line is one tweet), and provides a simple user interface for querying the list of tweets against keywords. The system replies to keyword queries with the top 10 tweets that are most relevant to user query keyword(s).

As an example, assuming that the input file contains the following three tweets:

tw1 = <that moment while reading a student paper when the font changes>

tw2 = <followed by that moment when you Google the passages before and after the font changes>

tw3 = <just happened to me in this moment>

Given the query “the moment font changes”, your search system must

i) find the list of tweets that contains at least one word in the given query, (in our example the list would be {tw1, tw2, tw3}),
ii) compute the similarity of the tweets in the list with the query keywords

according to the number of times they occur in the tweets (in our example the scores would be {tw1: 4, tw2: 5, tw3: 1}), and

iii) select the top 10 most similar tweets (in our example since we have less than 10 results, we will select all three of them)

iv) display the tweet texts to the querying user in the relevance order,

In our example that would be:

1) followed by that moment when you Google the passages before and after the font changes

2) that moment while reading a student paper when the font changes

3) just happened to me in this moment

You are free to use any data structure you like for storing the tweets, the intermediate data structures such as inverted indexes, etc… But you must define the complexity of each functionality in your search engine, e.g. if you are using an inverted index, building the inverted index, finding the inverted lists associated with the query keywords, taking the union of the inverted lists, computing a ranking, selecting the top 10 most relevant tweets, returning the tweet texts, etc… Efficient data structure selection and usage will get higher points.

Minimum Requirements [70%]

- Command line user interface that lets inserting raw tweet files, and querying.
- Ability to retrieve relevant tweets to the query keywords.
- Ability to display relevant tweets in correct importance order.

Comparative Features [30%]

Your algorithms and data structures will be compared against others in the class and ranked according to:

- Speed of key operations

3) Project #3: Image foreground/background segmentation using max-flow

A basic problem in computer vision is how to segment objects in images. In class we will discuss segmenting the foreground from the background in an image. This problem is generally treated as a clustering problem, where the pixels are clustered into two groups (foreground or background).

Max flow algorithms can be used as way to solve the segmentation problem efficiently in the case of two segments. In order to actually implement the algorithm, one requires for every pixel a likelihood for the pixel to be assigned to the foreground or the background. Typically this is done by using an off-the-shelf clustering algorithm based the color (e.g., RGB values) of the pixels, such as k-means or Gaussian mixtures; existing code can be used for this step. This requires that selected regions of the image be labeled as background, and other regions as foreground. A good reference paper on this is:

Y. Boykov and M. P. Jolly. Interactive graph cuts for optimal boundary and region segmentation of objects in N-D images. In ICCV, volume 1, pages 105–112, July 2001.

A more recent paper implementing this is in
https://www.dip.ee.uct.ac.za/~nicolls/publish/mk09-prasa.pdf

As input, you will receive gray scale jpeg images of variable size. You must develop a way of interactively selecting a region of background pixels and another region of foreground pixels for training data that will be used to estimate a probability model based on intensity for both background and foreground pixels. You will also need to specify an appropriate model for the energy required to separate neighboring pixels into different classes. With these you can formulate the problem as a min cut problem, which can be solved using max-flow algorithms (which you must implement).

Minimum Requirements [70%]

- Command line user interface that lets user identify image
- Graphical interface to let the user specify parts of foreground, background regions for
- training
- Algorithms for computing most likely segmentation based on max flow
- Graphical display of segmentation output highlighting foreground and background regions.

Comparative Features [30%]

- Exploration of the effect of alternative models for foreground/background likelihoods
- Exploration of the effects of alternative separation energy models
- Speed of key operations

4) Project #4. Comparative analysis of max-flow algorithms

In this project, you will implement different versions of preflow-pusn algorithms for max flow (including excess scaling, changing push order, etc) as well as the Edmonds-Karp max flow algorithm. You will be provided with input files with a specific format for the max flow problem, of different sizes. Your goal will be to evaluate the relative performance of these algorithms as the problem size increases, and generate a report regarding this performance.

Your code will be tested on input sample networks by us to verify correctness, as well as computation speed.

Minimum Requirements [70%]

- Command line user interface that lets user identify image
- Graphical interface to let the user specify parts of foreground, background regions for
- training
- Algorithms for computing most likely segmentation based on max flow
- Graphical display of segmentation output highlighting foreground and background regions.

Comparative Features [30%]

- Exploration of the effect of alternative models for foreground/background likelihoods
- Exploration of the effects of alternative separation energy models
- Speed of key operations

5) Project #5. Comparative analysis of Data Structures

In this project, you will implement different versions of advanced data structures for search and retrieval, and compare their performance on benchmark data sets. These will include several data structures beyond what we discussed in class. Examples of comparable structures are:

a) Van Emde Boas trees, x-Fast tries, y-Fast tries for ordered dictionaries that support insert(x), lookup(x), (which returns whether x ∈ S.), delete(x), which removes x from S. max() / min(), successor(x), (which returns the smallest element of S greater than x), and predecessor(x), (which returns the largest element of S smaller than x.)
b) K-d trees and locality sensitive hashing for solving the nearest neighbor problem for multidimensional data.

Alternatively, you may suggest some data structures of interest, as well as the relevant comparisons. We will develop appropriate input files based on the problem you specify.

Minimum Requirements [70%]

- Command line user interface that lets user input queries as well as select appropriate structure for response.
- Command line interface to see response for queries.

Comparative Features [30%]

- Speed of key operations

Final Project Submissions

I. Documentation

- description of the problem
- relevant references and background materials
- high-level description of the implementation, with particular emphasis on
- the design decisions related to data structures and algorithms
- a list of the features from the project proposal that have been implemented
  for each feature implemented, provide a description of how it was implemented with an emphasis on data structures and algorithms used

II. Code : complete, working code for your implementation

- clear and terse instructions how an average student in class can compile and run your code from scratch

III. Supporting files

- all libraries needed for your project to run, and instructions for how to
- freely (and legally!) acquire them
- examples of how to use your project
- testing patterns: unit tests, ystem tests

IV. Work breakdown

- a statement detailing what each member of the group contributed to the project, signed by all members of the group.

Note that you are free to choose your implementation language and environment. However, you must identify what is needed for us to replicate your results. Simple C++ code in the style of the Homework exercises is recommended. Better to have a modest code that is correct.

Keep in mind that severe penalties will be assessed for any code that we cannot run from scratch with a reasonable amount of effort to run. The code must be compile and run on your CCS account. Little to no credit will be given to any code that is not working or which we can cause to substantially misbehave.

Each team should have code on a **GitHUB with a README** adequate to explain how to run the code.

All slides, documents and graphic output submitted at /project/alg504/yourloginname/PROJECT duplicated for all members of the team. The code must be able to be compiled by a Makefile and run to give some sample output.

:Dec 5 and Dec 10: Each Team will give a short slide presentation in the last two classes

Dec 11: The written documentation submitted in a well short written report (5 to 8 pages recommended) and submitted as a pdf file also at at /project/alg504/login_name/PROJECT