Linked Lists : Cache Algorithms Basics

Making a good routine is one important thing in life. Can a CS student do something better ? Let's dive in !
For sometime, think you're Bill Gates (for sometime…) You've to meet many people on a daily basis.We have a series of meetings to conduct throughout the whole day. How should we make our time table ?
Sample: A -> B -> Z -> A -> X -> B -> A -> C -> B -> D
Plan : At a given time, if you've met a person X, don't meet him again too soon. You give someone else your time. i.e. in this example, after Z, we are scheduling A but we've met him recently. So, we should not be meeting him now. We'll meet X.
In short, try meeting people you've met for a long time and avoid meeting people you've met some time ago.
Do anything you like to do but use any linked list (singular/doubly/circular) for the list of meetings. For the rest, use anything i.e. arrays/lists/tuples etc. etc.
Steps :

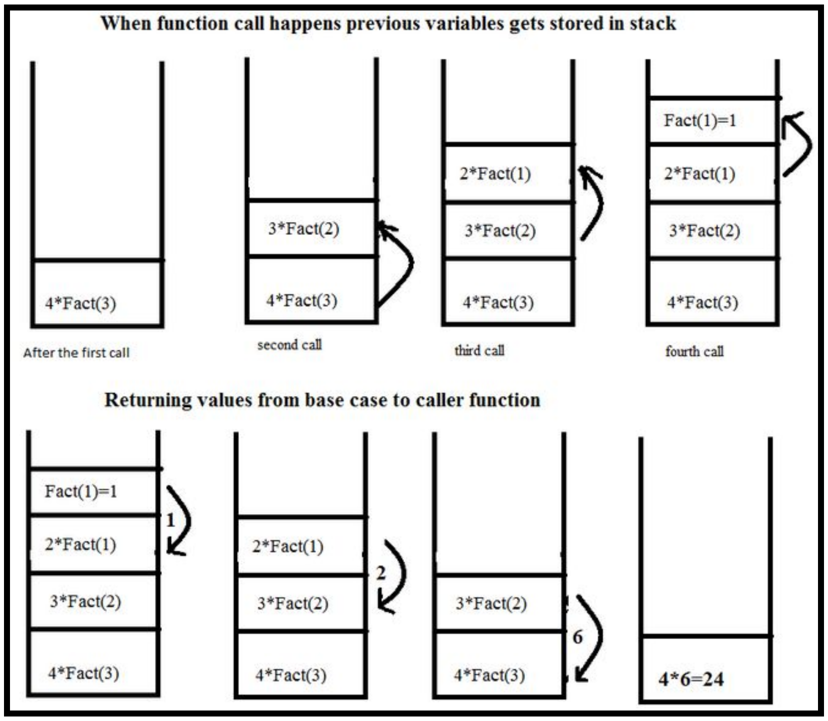
- At some node, you need to know what has happened in past
- Don't care about the future. You only know the next meeting
- Try to implement efficiently

Proof : Someone suggests Bill Gates (oh sorry You…) why don't you look at the next two meetings instead of one (in linked list language : instead of checking only node->next , check also node->next->next). That will yield a better time table for you. Now, we don't know if he's suggesting right or not.
(I don't know the answer either so it will be fun to discuss tomorrow.)

Learnings/Objectives

- Cache Algorithms
- Memory Replacement/Process Scheduling Algorithms
- Greedy Algorithms
- Modifying LRU/MRU to get good results
- What's better than LRU
- 3-4 Ideas of doing these things in O(n) & O(n^2)
- Using Hashing to improve efficiency
- Using different list type to half the time & improve the results

Recursion Iteration : What the Fuss…



This figure shows how recursion works in the memory in the case of factorials.
First, we've n recursive calls, then we've answers in the reverse direction. This can be done by the stacks (LIFO) structure.
We've to do this to show both iteration and recursion.
Though iteration is not shown in the figure, it is straightforward.
We'll maintain an activation record as well in both the cases. It will have main features like the following (but you can add more…)

- What's the memory address of this block ? (Block number of stack)
- What's the data inside it ? (Members of the class etc)
- We'll also display the types of data (using type etc commands)

In short, we've to simulate the Memory for Factorials Program both with Iteration (loops) and Recursion using stacks.

Now, observe what you've implemented. You're now free to think. Can we have another structure instead of Stacks ? Even if it is not better, don't worry ! Don't hesitate about how this will be implemented etc. etc. **Wrong solution, Inefficient solution to this question doesn't matter ! Just think ! Have a solid reason !**

Learnings/Objectives

- How Recursion & Induction Works Visually ? (in Math & in CS)
- How Induction & Recursion relate ?
- Why use Induction for Recursive Proofs ?
- Recursion implementation without Language Support
- Ideas for doing recursion (without stacks/queues)

Stacks & Libraries

Task 1

- Get Data from Sheet (i.e. Excel)
- Do some operations on it (can use Numpy etc or make your functions)
- Organize it in Table form
- Show the progress of work

Library	TQDM	OpenPyxl	PrettyTable
Usage	To show progress on bar	To work with sheets	Organize data in table form

I have attached links in the table (click on library names).

Task 2

Implement a Stack which does following operations (you can add more)

- Push
- Pop
- PopLF : pops the **least frequent** value
- PopMax : pops the maximum value
- Top : shows the top value

Task 3

Sort data (any data type) by using stacks in better than O(n^2)

Learnings/Objectives

- Using different libraries to do things
- TQDM : A library for checking progress made
- Openpyxl : A library to automate the spreadsheets
- PrettyTable : A library to express things in Database form
- How to process data using Python
- How to convert them for Databases or Objects/Structs/Classes
- Finding Least Frequent & Maximum in less than O(n)
- 2-3 Different Ideas for sorting data by Stacks

Queues : Time Scheduling Basics

Task 1

Implement a Circular Queue with basic operations i.e.

- Push
- Pop
- Top

Task 2

Suppose you've 10 kids to look after (I know it's difficult but suppose :P)
They all need attention. So, you decide that I'll give 2 minutes to each turn by turn.
But if you keep doing this, you'll do it forever.
So, to help you out, you're given the data about kids. That after some minutes this kid will sleep and so you should leave him. Now, you've to give two minutes to 9 kids because one is asleep. After some hours, everyone will sleep and you're done.

Example : kid1 needs 20 minutes, kid2 needs 10 minutes & kid3 needs 25 minutes
You give two minutes to kid1 and now 18 minutes of care are left for him
Now you move to kid2 and then to kid3. After one cycle
Kid1 needs 18 more minutes, Kid2 needs 8 minutes & Kid3 needs 23 minutes

You keep doing this and after some cycles Kid2 will be the first one to sleep and you'll be left with Kid1 & Kid3. Then, Kid1 will sleep and you give all your care to Kid3. Kid3 will sleep and you're done !

Use Queues (any type i.e. circular/array/list) to model this. Calculate the time when each kid sleeps. Also, calculate the time when you're free...

Learnings/Objectives

- Algorithms for Multiprocessing
- Round Robin in O(n)
- Round Robin in O(1) with Hashing
- Hash Tables as Foreign Keys

Trees

We will implement a Dictionary (not the python data type the real one). There are many ways to do this. For example, by binary trees, binary search trees, hashing and so on… But they all have their disadvantages because they're not very much suitable for this i.e. a binary search tree is good for numbers. We can easily check in O(1) if a number is less than or greater than. But for string it will not be the case. In hashing, i.e. we store the word "car" and then we store "card". We'll do this by checking collisions first and then, add "card" in front of "car" in a linked list or some other way. However, if we think of inserting each word in a different place in the hash table, that's not hashing anymore.

Now, we will do this problem with Trees. They don't have to be binary ones (they can have more than two pointers). It is not a limitation that the solution should have more than 2 pointers (it is just a suggestion). You can use anything related to trees for implementing a structure that helps us build a dictionary.

Implementation

In short, we have to implement a structure for the dictionary and we want to do it with Trees (may it be binary or may it not be binary depending on your idea but it should solve our problem).

Theory

Analyze your solution with the other data structures you've studied. (i.e. in terms of complexities of time & space etc)

Learnings/Objectives

- Autocomplete & Autocorrect Features
- Implementing faster than Hashing
- Find/Search in nearly Constant Time
- Why not Hashing etc ?
- Idea of Decision Trees, Trie, Trees in Probability/Discrete Math etc
- Managing nodes in a array/list/hash-table/tuple etc
- Saving huge space with back pointers

Heaps

We’ll use heap structure as Priority Queues. We’ve already implemented a case in Queues (baby-sit task). Now, instead of going one by one and giving a kid time, we’ll have a priority associated with each kid. We’ll give turns by priority. It’s like one kid is feeling very much irritated and so, you should look after him/her first and then, move forward. After giving time to the kid, you should update the priority by decreasing (otherwise only one kid will be taken care of…).

What if you’ve given some time (i.e. 2 minutes) to Kid-x and after updating the priority, the priority of Kid-x is still the maximum ? It is upto you to select from the following:

- You can give the Kid-x another turn based on priority
- You have chosen not to give consecutive turns to one kid

Implementation

In short, you are going to do the Queues task with Priority Queues now. You can make my assumption (like one mentioned above). The only limitation is to use Heaps for Priorities.

Theory

Is there a better way to think about Priority Queues ? Can we do the Priority stuff with any other data structure ? Compare this with Heaps.

Learnings/Objectives

- MultiTasking/Processing/Scheduling with Priorities
- Other Ways to Implement Priorities
- Why Heaps are better usually (not always)
- Updating Heaps
- Heap-Up vs Heap-Down ? Why ? When ?

Sorting & Strings

Today's work was on Sorting. To keep the work short (due to Eid), instead of doing something with sorting, today's task focuses on Strings/Files. This is not a difficult task but it will give the idea of some basics.

Task 1 : Basics of DNA Translation

We're given a sequence of DNA & a sequence of Proteins. Some part of DNA maps to some protein.

Example

ACTCATGACCTGA is a DNA Sequence.
ACT maps to protein X, CTG to protein Y and so on.

Task 2 : Basics of Encoding/Decoding

We're given a message and we'll encode & decode it using simple shifts (you can do better too)

Though these tasks are not difficult, this will give you a better sense of some basics. Also, we'll try to generalize our work… i.e. encoding/decoding should work for any message and we can choose any shifting scheme instead of a fixed one.

Task 3 : Quick-Sort *(Optional)*

Implement the Quick-Sort Algorithm. Make your own recurrence. Compare it with other sorting algorithms. Also, analyze its complexity. (You can read how randomization helps in Quick-Sort and improves the time complexity)

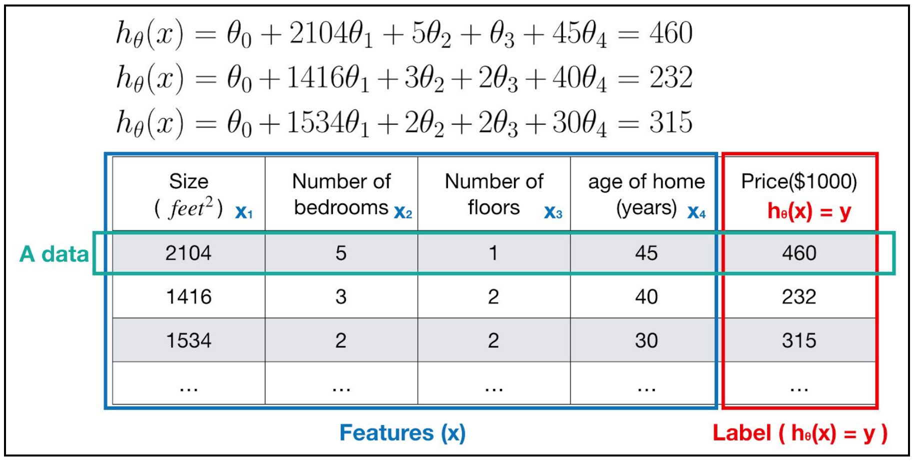
Learnings/Objectives

- File Reading Basics in Python
- Using Strings to find patterns
- How Modulus operation helps in cryptography
- Using Hashing instead of Shifting to improve time complexity
- An example of in-stable sort with recursion
- Basic Idea of Randomness in Algorithms

Divide & Conquer : Machine Learning Basics

Linear Regression is a very powerful technique in Machine Learning. We want to learn something from past experiences. Also, we want our learning to have **Minimum** errors. Of the very powerful algorithms, following are two of them listed:

- Gradient Descent : The idea is to keep taking derivatives for local minimums aiming finally for the global minimum. Don't worry about details.
- Storing Data in Matrices can make Linear Algebra useful. Now we are kind of taking the derivative of whole data instead of choosing a local minimum. We'll not implement the actual algorithm, but we'll do the multiplication or transpose part (or both) using Divide & Conquer.



$$h_\theta(x) = \theta_0 + 2104\theta_1 + 5\theta_2 + \theta_3 + 45\theta_4 = 460$$
$$h_\theta(x) = \theta_0 + 1416\theta_1 + 3\theta_2 + 2\theta_3 + 40\theta_4 = 232$$
$$h_\theta(x) = \theta_0 + 1534\theta_1 + 2\theta_2 + 2\theta_3 + 30\theta_4 = 315$$

Size ($feet^2$) x_1	Number of bedrooms x_2	Number of floors x_3	age of home (years) x_4	Price($1000) $h_\theta(x) = y$
2104	5	1	45	460
1416	3	2	40	232
1534	2	2	30	315
...	...	...	...	...

A data
Features (x)
Label ($h_\theta(x) = y$)

$$h_\theta(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

Parameters: $\theta = \{\theta_0, \theta_1, \theta_2, ...\theta_n\}$

Features: $x = \{x_0, x_1, x_2, ...x_n\}$

We will only do the Matrix Multiplication. So, don't worry about other things. You just need to multiply ***theta*** and ***values of x*** which in this case are 4 (x1,x2,x3,x4). Your answer should equal the corresponding ***y-values*** (of course…). For more simplicity, you can assume any theta values without caring about the y-values...

However, for this, we also need ***(a x n).(n x b)*** order which might not be given. For that one option is taking transpose.

In short, you need to multiply the ***theta-matrix*** with the ***features-matrix*** using Divide & Conquer (you can multiply in parts with recursion instead of using nested loops... Divide & Conquer means reducing the problem i.e. in Binary Search).

Learnings/Objectives

- Recursion & Divide-Conquer Strategy
- Changing Matrix Order w.r.t without Data/Result Loss
- Idea of Linear Algebra/Calculus in Computer Science
- Intuition of Strassen's Algorithm

(Optional)

Algorithms are pretty much everywhere. We've already implemented a complex tree structure for auto-correct/auto-complete features. If you've implemented it (using any tree approach), you would've realized the complications involved. Still, that works nearly in O(1) so we can go with that in Auto-Complete & Searching Dictionaries. However, Auto-Correct isn't very much efficient with this approach.

Example

A possible suggestion to the word "***henged***" is "**hanged**" (as Google suggested the same). Why not ***hang, hangared,hen… ?***

So, our task is to decide between the words and suggest one from the whole collection of words. One idea is to count the operations performed i.e.

- ***henged -> hanged*** is only one word changing **e->a**
- ***henged -> hang*** is 2 subtractions (ed) and 1 changing (e->a)
- ***henged -> hangared*** is 2 words addition (ar) at and 1 changing (e->a)

The best option here is "**hanged**" because we now know it is the most similar as it costs only 1 operation. Don't hesitate about whether this approach gives the best suggestion or not. It is just one algorithm of the many used. In-short, our approach is **if the cost is less, similarity is more**.

Task 1 : Theory

You can do it with any approach but not **Brute-Force**. Come up with a well-detailed **pseudocode**. If your solution is recursive, write the recursive steps in detail. Reason your approach and complexity.

Task 2 : Implementation (Optional)

Implementation for this is optional because the solutions can be difficult to think about and taking care of minor details might take upto a week.

Learnings/Objectives

- Algorithm for Plagiarism Detection/Speech Recognition/Spell-Check
- How to make Recurrences for problems
- Dynamic Programming & Memoization (What's the difference ?)
- Principle of Optimality & Structure of Problems/Subproblems
- Tabulation/Memoization & Relation with Arguments/Parameters

Graph Traversals : Seeders... Where ?

Task 1 : Help Torrent Find Seeders

Seeders are the users on the internet who've downloaded the torrent you're trying to download. Now, your request for torrent will not go to the server rather it will find seeders who've downloaded the same torrent. The more the seeders, the faster it is (as far I know). If there's no seeder (no-one downloaded/uploaded that file) then, your torrent will then request the server for downloading because no seeders are available. This way huge files are not being directly downloaded from the server but from the seeders. If the seeder is farer/deeper, the cost will increase. If the seeder is sitting next home, the cost will be less.

One way is to see this working as a Graph. Help your torrent request find the seeders that cost less. We're also giving the user an option to specify how many seeders he needs for his torrent download (by default, you can set any number).

In-short

- First search if your file is in the seeders graph
- If no, your output will be "Download from Server"
- If yes, also find how many
- Then, ask the user how many seeders do you want ?
- Then look-up for seeders
- The cost should be minimum (try finding nearer seeders)

Note : Try not to copy traversals code from the internet. This will help you think deeper about the traversals, their intuition & nature.

You can use any traversal (DFS/BFS) for this. **Either implement or write a detailed pseudo-code**. Reason your choice for the traversal and the way you implement them i.e. Should we choose the **linked list** representation or the **matrix** one ? Should it be **directed or undirected** ? etc.
Analyze the complexity & implementation/pseudo-code.

Learnings/Objectives

- Two approaches BFS/DFS or BFS/Arrays
- What to do where while solving a problem with a traversal?
- Thinking in terms of Graph (representation, type etc.)
- How BFS nature can save time complexity with slight modifications

Touring with Spanning Trees

Let’s get on a long journey. Suppose you’re going with someone special (someone you want to spend time with :P). You have 10 hours. You can imagine a graph of locations and their edges have minutes/hours as weights. You choose to think this the “Spanning Tree” way.

Task 1 : Theory

Before jumping into the solution, ask yourselves questions like

- What type of graph should it be ? Undirected ? Directed ? Other ?
- Should the graph have some features to make your work easy ?
- How can we make Spanning Tree work here ?
- Should we go with Minimum approach ? Maximum ? Any other ?
- Other such questions that can make the problem easily solvable

Task 2 : Pseudocode

Make any assumption. You’ve no restriction. You decide the problem. You decide the graph. You decide everything. Then, based on those, suggest a solution. This will be not easy to implement in a day or two. So, write a **detailed pseudocode**. Analyze the time complexity of the solution.

Learnings/Objectives

-