Great Ideas in Computing

University of Toronto CSC196
Fall 2020

Week 2: September 27-October 1 (2021)

Week 3 slides

Announcements:

- Assignment A1 is now available for submission on Markus. Please let me know if there is an issue downloading your submission. It is due Friday, October 8, 8AM. There is a 5% penalty for each 24 hours the assignment is late with a maximum of 96 hours to submit a late assignment.
- Our first guest presenter will be Professor David Duvenaud who will talk about ML and neural neworks. He will conduct the presentation Wednesday on zoom (usual class zoom link).

The agenda for this week

- I will have some remarks regarding AI and ML in preparation for Professor Duvenaud's discussion on Wednesday.
- I will have a few brief comments on fixed point vs floating point representation.
- We will then continue where we left off last week. Namely, we will continue to discuss the *dictionary* data type and differert ways to implement a dictionary.

Some remarks about AI and ML

One hears about AI, machine learning (ML) and data science every day.

In our DCS research MSc and PHd applications, perhaps 60-70% of applicants indicate that ML is their first choice of fileds. In our applied research MScAC program, perhaps 60-70% indicate ML or data science is their first choice. One of the "concentrations" in the MScAC is data science which we do together with the Department of Statistics.

We can have a discussion about data bases vs data science vs ML vs AI. These are terms that don't have precise mathematical meanings but we can give some examples of what we might have in mind. (You might have your own more illustrative examples in mind.) Each of these is considered a different field and DCS and other departments offer several courses in each area.

- In data bases we decide how to store and retrieve information about records within some data base.
- In data science we are often trying to obtain statistical and statistical predictive information about large amounts of data.

Current ML

- ML is today the major part of AI (say in terms of impact and current research and development). The goal in ML is more ambitious in that it attempts (currently "learning" by using large sets of training data) to match or exceed performance at activities what we usually associate with human intelligence.
- As an example of a very succcessful activity consider recognizing human faces or identifying different parts of a scene (eg find all the birds in this photo). This example is part of what we call the area of computer vision. Playing Chess or Go is another example.
- As I said, the distinction between these areas is not entirely clear. Is computing the probability that an individual will default on a bank loan can be considered data science or ML. And, of course, when such predictive results are being made by an algorithm (with or wothout some human oversight), we clearly have some ethical issues.

ML vs AI

The field of artifical intelligence (AI) as envisioned by its pioneers considers what might be the ultimate raison d'etre for computer science, namely to be able to match or outperform human intellignece and moreover to understand how to learn even with little or no training examples.

That is, how do we as humans learn different things (e.g. natural language, recognizing different types of objects, forming concepts, reasoning)?

The history of AI has been called a rooler coaster, many promises and failures, followed by huge successes in what ML can now do.

But still now the fundamental debate continues: To what extent can machines learn and reason.

On the web site I am providing a link to an article on the history of AI and also Geoff Hinton's commencement address at IIT,Mumbai.

One specific question is the following: To what extent, can we learn new "useful" (and even possibly, theroetically optimal) algorithms for say a specific combinatorial optimization problem?

Fixed and Floating point representation: A quick review

- As we have already discussed. we have to approximate non integers by fractions where in particular . say every real number in the interval $[0, 1]$ can be approximated by a fraction $.b_{n-1} \ldots b_1 b_0$ where $b_i \in \{0, 1\}$. The more bits better the approximation.
- Some fractions can be represented exactly in such a binary representation (e.g. 1/2, 1/4, 3/4, etc.) while other numbers like 1/10 and is 1/3 can only be approximated. (Note: One can, of course, represent these numbers exactly as a ratio of two integers.)
- We may need very small or very large numbers but the number of bits in a computer word is fixed (for example, 32 bits) so this limits how big or how small numbers can be. This is not an artifact of the binary representation. The same limitations would apply to any base.
- In a *fixed point representation*, we represent numbers by agreeing to have some fixed number of fractional bits.

Fixed and floating point numbers continued

For example, in an 8 bit fixed point representation $b_7 b_6 \ldots b_0$, where b_7 is the sign bit, we can agree that the two lower order bits b_1 and b_0 are the fractional parts. Then the decimal number -18.5 would be represented exactly by 11001010 and -18.25 would be represented exactly by 11001001.

Note that in a pure integer or fixed point representation, the sizes of the smallest and largest numbers are severly restricted. For a 32 bit word with a sign bit, the largest number is $2^{31} - 1 \approx (10)^9$ (i.e. approximately one billion). And every bit of precision we use for the fractional part decreases the range of numbers representable by approximately a factor of 2.

Moreover, the precision for the fractional part is also severly restricted by the the number of bits in the fractional part.

The benefit of fixed point notation is that integers (or the integral part of a non-integer) is represented exactly within the range that the number of integral bits allows. Fixed point is the usual representation for integers within the allowable range.

Floating point numbers

The common solution to provide a large range as well as providing good precision is to use *floating point representation*.

The tradeoff is that we lose some precision for some numbers that have an exact fixed point representation. For example, the decimal fraction $\frac{1}{10}$ cannot be represented exactly.

All integers requiring at most 24 bits can be represented exactly.

Floating point arithmetic also has some delicate issues.

The "wonder" of the von Neuman model and programming languages is that the end user "usually" doesn't have to be aware of these represntation issus. However, algorithm designers for numerical computations do have to worry about how rounding errors can propogate. (Advice: Take a course in numerical analysis.)

End of Monday September 27 class

I clearly didn't get much past our discussion on ML and AI but that us fine. The discussion today shows that there is a good deal of class interest in ML and AI. I tbhink we are all looking forward to the Wednesday class.

Next Monday, I will finish up the distionary data type and its implementation in different data structures. Ortv we might do that in the tutorial.

I will soon stop using zoom for meetings and classes will be in person only. **Exception:** If you email me and tell me why you cannot attend in person on a given day, I will run a zoom session at the same time as the in person class. But you much tell me in advance.

Dictionaries: A quick review and some additional data structures

The basic operations in a dictionary are the following:

- *Search*: Look up if someone is in the organization and if so retrieve the information for this person.
- *Update content*: Change the information regarding an item
- *Insert*: Add a new person to the organization
- *Delete*: Remove a person from the organzation.

Sets of objects with these opeartions are refered to as a *Dictionary* data type. It is a static dictionary if we only want to look up and possibly modify records and a dynamic dictionary if we also want to add and delete. We can use different *data structures* to implement such a data type.

There can be many more operations that we want to perform on collections of data. More generally how one maintains and operates on data is known as the subfield of data bases. Analyzing data and extracting new (often statistical) information from collections of data is now called *data science* or *data analytics*. More ambitious learning of new information from data can be called *machine learning*.

Dictionaries lead to interesting concepts and ideas

- Many ways to implement a dictionary. What is important to note is that there are almost always TRADEOFFS in whatever we do in computing (and in life). How do you compare alternatives when there are multiple criteria for any given choice?. When can we say that choice 1 is better than choice 2 according to the given criteria.

Dictionaries lead to interesting concepts and ideas

- Many ways to implement a dictionary. What is important to note is that there are almost always TRADEOFFS in whatever we do in computing (and in life). How do you compare alternatives when there are multiple criteria for any given choice?. When can we say that choice 1 is better than choice 2 according to the given criteria.
- Here are some well known ways (called *data structures*) to implement a dictionary.
  1. An unordered list in an array
  2. An ordered list in an array
  3. A linked list
  4. A (balanced) search tree.
  5. A hash table.
- We will briefly talk about each of these possibilities. I do not want to get into details. Instead I just want to give a very high level idea of these different ways to implement a dynamic dictionary mentioning some tradeoffs and introducing some related concepts.

Brief discussion on these different methods

Let n be the current number of items in dictionary.

Each item has a unique name or *identifier*.

After I describe each method (on the white board), lets discuss some pros and cons of each method.

Some pros and cons of an unordered list in array for a dictionary

Relatively easy to add or delete an item (assuming we don't exceed the size of the array)

Some pros and cons of an unordered list in array for a dictionary

Relatively easy to add or delete an item (assuming we don't exceed the size of the array)

Requires an "average" of $n/2$ comparisons to find a current item and n comparisons to determine if the requested item is not in the current array. This is a hint of an important issue: namely, what does *average* mean?

Some pros and cons of an unordered list in array for a dictionary

Relatively easy to add or delete an item (assuming we don't exceed the size of the array)

Requires an "average" of $n/2$ comparisons to find a current item and n comparisons to determine if the requested item is not in the current array. This is a hint of an important issue: namely, what does *average* mean?

We usually have to indicate the size of the array in advance and would then have to allocate a new array if the number of entries exceeds the array size.

We need some memory management system for dynamic dictionaries. But this is true for any data structure.

Some pros and cons of an ordered list in an array

Note: This is only applicable if the items or the identifiers can be ordered which is usually the case.

Can search for an item in at most $\approx \log_2 n$ comparisons. Doing an asymptotic analysis of the time (and memory) for an algorithm is one of the main aspects in the analysis of an algorithm. Of course, correctness of the algorithm is paramount.

Some pros and cons of an ordered list in an array

Note: This is only applicable if the items or the identifiers can be ordered which is usually the case.

Can search for an item in at most $\approx \log_2 n$ comparisons. Doing an asymptotic analysis of the time (and memory) for an algorithm is one of the main aspects in the analysis of an algorithm. Of course, correctness of the algorithm is paramount.

$\log_2 n = x : 2^x = n$. Note that x will not be an integer unless $n = 2^k$ for some k.

To be precise the worst case number of comparions is $\lfloor \log_2 n \rfloor + 1$ where the floor function is defined as $\lfloor x \rfloor =$ the largest integer $k \leq x$. You can verify that for $n = 2^k - 1$, the worst case number of comparison is k.

Ordered lists in an array continued

The differences between $\log n$ and n, can be dramatic (say if a search is within a *loop* of instructions). Even more dramatic is the difference between n and 2^n. We will be discussing further the importance of complexity issues.
It is more difficult to insert and delete records or modify the identifier of a record even for a fixed size array although updating the content of a record is easy once the item is accessed.

Can easily identify the i^{th} largest or smallest element.

And we usually have to specify the size of the array in advance.

Tables of some complexity bounding functions

Table 2

Polynomial-Time Algorithms Take Better Advantage of Computation Time

Time Complexity	n = 10	n = 20	n = 30	n = 40	n = 50	n = 60
n	0.00001 second	0.00002 second	0.00003 second	0.0000 second	0.00005 second	0.00006 second
n^2	0.0001 second	0.0004 second	0.0009 second	0.0016 second	0.0025 second	0.0036 second
n^3	0.001 second	0.008 second	0.027 second	0.064 second	0.125 second	0.216 second
n^5	0.1 second	3.2 seconds	24.3 seconds	1.7 minutes	5.2 minutes	13.0 minutes
2^n	0.001 second	1.0 second	17.9 minutes	12.7 days	35.7 years	366 centuries
3^n	0.059 second	58 minutes	6.5 years	3855 centuries	2×10^8 centuries	1.3×10^{13} centuries

Figure: Figure taken from Garey and Johnson "Computers and intractability : a guide to the theory of NP-completeness". Time in seconds based on an estimate of computers in the late 1970s. What if today computers are 100 times faster. Does this change the "message" in this figure.

A linked list

I may want to jump ahead to hash tables to motivate the exercises on Assignment A1.

Introduces the idea of a pointer

A linked list

I may want to jump ahead to hash tables to motivate the exercises on Assignment A1.

Introduces the idea of a pointer

I have shown a singly linked list. Can have a doubly linked list.

A linked list

I may want to jump ahead to hash tables to motivate the exercises on Assignment A1.

Introduces the idea of a pointer

I have shown a singly linked list. Can have a doubly linked list.

Easy to add items if the list is unordered. If list is ordered then have to follow pointers to see where to insert a new item.

A linked list

I may want to jump ahead to hash tables to motivate the exercises on Assignment A1.

Introduces the idea of a pointer

I have shown a singly linked list. Can have a doubly linked list.

Easy to add items if the list is unordered. If list is ordered then have to follow pointers to see where to insert a new item.

May have to traverse the entire list to find an item or determine it is not there.

Blank page for drawing

A balanced binary search tree

A balanced binary tree with n "nodes" will have depth $\log_2 n$ and hence can search a balanced binary search tree in at most $\log_2 n$ "edge" traversals and comparisons.

I use the terminology of nodes and edges as a *tree* (in the sense of a search tree) is a special case of a *graph*. Graphs are also referred to as *networks* in many contexts (i.e. a social network, a transportation network, etc.).

The nodes (also called vertices) and egdes (also called arcs in some applications) can be undirected or directed. In the latter case, we call a graph with directed edges a *directed graph* and usually mean an undirected graph if we just say graph.

We will be discussing further some graph concepts as the term progresses.

A hash table

We have a hash function $h : I \rightarrow M$ where $I = \{ID_1, \ldots ID_N\}$ is the set of all possible integer identifiers and $M = \{A[0], \ldots, A[m-1]\}$ is a small set of memory locations

That is, we are going to hash each of the $N = |I|$ possible items to a small set of $m = |M|$ memory locations.

Here we can have $N >> n$ where n is the actual number of items we are storing.
What is a suitable hash function h?

A hash table

We have a hash function $h : I \rightarrow M$ where $I = \{ID_1, \ldots ID_N\}$ is the set of all possible integer identifiers and $M = \{A[0], \ldots, A[m-1]\}$ is a small set of memory locations

That is, we are going to hash each of the $N = |I|$ possible items to a small set of $m = |M|$ memory locations.

Here we can have $N >> n$ where n is the actual number of items we are storing.
What is a suitable hash function h?
One possiblility is $h(ID) = (a \cdot ID + b)(mod\ p)(mod\ m)$ where p is a large prime.

A hash table

Ignoring conflicts in the hash table, can search in constant time for a particular item

A hash table

Ignoring conflicts in the hash table, can search in constant time for a particular item

Need to deal with conflicts; i,.e multiple items hashing to the same place in the hash table. One possibility is to use a pointer to a linked list containing the IDs matched to the same place in the hash table.

A hash table

Ignoring conflicts in the hash table, can search in constant time for a particular item

Need to deal with conflicts; i,.e multiple items hashing to the same place in the hash table. One possibility is to use a pointer to a linked list containing the IDs matched to the same place in the hash table.
Nash tables introduce the use of probability, pseudo random numbers and pseudo random functions.

Note: When we draw random numbers in the execution of an algorithm, we are not drawing truly random numbers. The generation of pseudo random numbers and pseudo random functions is an interesting and substantial topic, one related to comlexity theory, our next topic.

The birthday paradox

The birthday paradox: In probability theory, the birthday problem or birthday paradox concerns the probability that, in a "small" set of n randomly chosen people, some pair of them will have the same birthday with high probability.

By the pigeonhole principle, the probability reaches 100% when the number of people reaches 366 (since there are only 365 possible birthdays, excluding February 29).

However, 99.9% probability is reached with just 70 people, and 50% probability with 23 people. These conclusions are based on the assumption that each day of the year (excluding February 29) is equally probable for a birthday.

Concluding remark

The choice of any particularv data structure or algorithm will usually depend on the app[ication. For example, in choosing a data structure what operations are being done more often than others is an essential consideration.