Great Ideas in Computing

University of Toronto CSC196
Fall 2020

Week 2: September 16 - September 23 (2020)

Week 2: Agenda and Annoucements

- We will start with the last slide we looked at on Monday; namely, Is Wikipedia a Great Idea?
- We will then discuss the Dictionary Data Type and ways to implement a Dictionary
- Our TA for CSC 196 is Marta Skreta. Marta is a PHD student in Machine Learning and Computational Healthcare. Marta is only able to do the tutorials on Mondays so going forward we will do tutorials on Mondays and I will conduct the class on Wednesdays and Fridays. I do have some conflicts on Fridays so we will have to do some juggling. Marta's interests are a great match for this course.
- The first tutorial will be Monday, September 21. No class this Friday.

Wikipedia

Is Wikipedia a great idea?

Wikipedia

Is Wikipedia a great idea?

In the Netflix documentary that I mentioned, the following question/comment was made: What if everyone was given their own Wikipedia page when they made a query using Wikipedia?
The commentary notes that when we are on social media we are often getting personalized news-feeds.

Wikipedia

Is Wikipedia a great idea?

In the Netflix documentary that I mentioned, the following question/comment was made: What if everyone was given their own Wikipedia page when they made a query using Wikipedia?
The commentary notes that when we are on social media we are often getting personalized news-feeds.

Confession: I didn't think Wikipedia would work. More specifically, I didn't think that enough knowledgeable people would be willing to spend their time to help create reasonably authoratative articles without getting any credit.

What is your experience with Wikipedia? Do you always believe what you read is accurate? How does it compare with other sources of information?

Data Types

Last meeting we discussed floating point numbers and it was noted (by a student) that floating point numbers are an example of a *data type*.

This led me to give an informal definition of what we mean by a data type. Namely, I said it was a collection of items (data) and the alllowable operations (and relations) involving those items. So as an example we can have a data type called Float where the data is numbers represented in floating point representation, the operations are the standard arithmetic operations +,-,*, division, exponentiation and perhaps logarithms. We also have the relations $<, =, >$.

You can check how Wikipedia states what is a data type.

We will nest introduce the Dictionary data type.

Looking up a record

Suppose with every person in an organization we have information stored in some "devise". It could be an old printed telephone directory, a folder in a physical cabinet, or a file in a cell phone or computer.

Lets think about how it could be stored in a von Neuman archiecture type computer in analogy to a file cabinet. Namely, think about one folder placed after the other. And for simplicity, say we have the same amount of information on each person.

In a computer one way to do this is to think of each person taking up some p consecutive words in memory (i.e., an array), one word for the name of the person, and $p - 1$ words for the information. If there are n people in the organization then we would be taking up $n \cdot p$ words of memory if we stored this information in an array.

Instead of the name of the person we could have some other indentifier.

Dictionaries

What are the most basic operations we want to associate with such a collection of information?

Dictionaries

What are the most basic operations we want to associate with such a collection of information?

- *Search*: Look up if someone is in the organization and if so retrieve the information for this person.
- *Update content*: Change the information regarding am 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 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*. Note: terminology changes all the time.

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 *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.

End of Meeting on Wednesday, September 16

We ended on slide 7. We spent quite a bit of time discussing Wikipedia (but that is fine) so didn't get as far as I had expected. After some announcements, we will resume discussing data structures for Dictionaries next Wednesday and also in the Monday, September 21 tutorial.

Annoucements

- Welcome to three new students who were on the wait list.
- Reminder: When you are not speaking best to keep your audio off. If you have a camera then I like seeing who everyone is especially when you are speaking. Good to use the chat feature if you wish to speak and then I can recognize individuals in the order of the chats.
- I posted assignment A1 on the web page. I want to grade assignments on Markus and have requested to get that operational. I will soon post A1 on Markus and maybe also on Quercus.
- Another reminder: For general questions, good to ask in class. But between classes, I prefer that you ask general question on piazza as that will be the fastest way to get a response since students often provide a response (which I greatly welcome). For more personal questions, you can email me at 196instr@cs.toronto.edu or Marta at 196ta@cs.toronto.edu.

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 blank sheets), lets discuss some pros and cons of each method.

An unordered list in an array

Some pros and cons of an unordered list in array

Unordered lists and ordered lists were discussed in the tutorial but we casn quickly go over them again.

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

Unordered lists and ordered lists were discussed in the tutorial but we casn quickly go over them again.

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

Requires "average" $n/2$ comparison to find a current item and n comparison to determine if 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

Unordered lists and ordered lists were discussed in the tutorial but we casn quickly go over them again.

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

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

Need some memory management system if array is too small.

An ordered list in an array

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. As we know the set of complex numbers are not ordered (in a total order).

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

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. As we know the set of complex numbers are not ordered (in a total order).

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

$\log_b n = x : b^x = n$. 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.

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.

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. As we know the set of complex numbers are not ordered (in a total order).

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

$\log_b n = x : b^x = n$. 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.

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.

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. As we know the set of complex numbers are not ordered (in a total order).

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

$\log_b n = x : b^x = n$. 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.

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.

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. Note that the number of seconds was based on an estimate of current computers in the late 1970s. What if today

End of Wednesday, September 23 Class

We ended at slide 15 on Wednesday.

My plan is to finish the discussion of the data structures for the dynamic dictionary data type.

I will say that Friday, September 25 is the end of "week 2".

Next week we will move on to one or two new topics.

1. A discussion of Alan Turing's seminal work on computability and his precise definiton for a model of computation.
2. Search engines: *a killer application*

Announcements

We now have four guest lectures schedule:

- September 30 Henry Yuen quantum computing
- October 14 Eyal deLara virtualization
- October 28 Roger Grosse ML and deep learning
- November 18 Alexandar Nikolov differential privacy

A linked list

A linked list

Introduces the idea of a pointer

A linked list

Introduces the idea of a pointer

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

A linked list

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

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.

A balanced search tree

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.).

A hash table

We have a hash function $h : I \to M$ where $I = \{ID_1, \ldots ID_N\}$ is the set of all possible 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 $h : I \to M$ where $I = \{ID_1, \ldots ID_N\}$ is the set of all possible 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 $h(ID) = (a \cdot ID + b)(modp)(modm)$

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. When there is a conflict, one possibility is to use a pointer to a linked list containing the IDs that have been 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. When there is a conflict, one possibility is to use a pointer to a linked list containing the IDs that have been matched to the same place in the hash table.

Introduces the use of probability, pseudo random numbers and functions.

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. When there is a conflict, one possibility is to use a pointer to a linked list containing the IDs that have been matched to the same place in the hash table.

Introduces the use of probability, pseudo random numbers and functions.

The birthday paradox: In probability theory, the birthday problem or birthday paradox concerns the probability that, in a set of n randomly chosen people, some pair of them will have the same birthday. By the pigeonhole principle, the probability reaches 100% when the number of people reaches 367 (since there are only 366 possible birthdays, including 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.