Great Ideas in Computing

University of Toronto CSC196
Fall 2020

Week 2: September 20-September 24 (2021)

Week 2 slides

Announcement: I will post Assignment A1 today (Monday, September 20). It is due Friday, October 8, 8AM.

The agenda for this week

- We will continue where we left off last week. Namely, we will first finish the dsicussion of the von Neumann model.
- We will then discuss floating point representation
- A brief discussion of Wikipedia
- The *dictionary* data type and differert ways to implement a dictionary.

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*.

- The von Neuman model assumes a random access memory in which each *fixed length word* is "addressable" and can be accessed at some unit (of time) cost.
  Is this how your lap top memory is organized?

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*.

- The von Neuman model assumes a random access memory in which each *fixed length word* is "addressable" and can be accessed at some unit (of time) cost.
  Is this how your lap top memory is organized?
  The"von Neumann bottleneck" addressed by caching and the memory hierarchy

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*.

- The von Neuman model assumes a random access memory in which each *fixed length word* is "addressable" and can be accessed at some unit (of time) cost.
  Is this how your lap top memory is organized?
  The"von Neumann bottleneck" addressed by caching and the memory hierarchy
- Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.
- Now here is a great idea (relating to digitalization): Instructions can also be represented by strings of symbols (indeed by strings of bits)! So instructions can also be stored in the memory, say for example one instruction per word!

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*.

- The von Neuman model assumes a random access memory in which each *fixed length word* is "addressable" and can be accessed at some unit (of time) cost.
  Is this how your lap top memory is organized?
  The"von Neumann bottleneck" addressed by caching and the memory hierarchy
- Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.
- Now here is a great idea (relating to digitalization): Instructions can also be represented by strings of symbols (indeed by strings of bits)! So instructions can also be stored in the memory, say for example one instruction per word!
  Why is this such a great idea?

Memory and the von Neumann architecture

The von Neumann architecture is a *random access stored program model*.

- The von Neuman model assumes a random access memory in which each *fixed length word* is "addressable" and can be accessed at some unit (of time) cost.
  Is this how your lap top memory is organized?
  The"von Neumann bottleneck" addressed by caching and the memory hierarchy
- Algorithms consist of individual instructions that say what "basic operations" to perform on data and also to indicate what instruction to do next.
- Now here is a great idea (relating to digitalization): Instructions can also be represented by strings of symbols (indeed by strings of bits)! So instructions can also be stored in the memory, say for example one instruction per word!
  Why is this such a great idea?
  Why is the von Neumann model such a great idea?
  Are we stuck in a "von Neumann tarpit?"

The benefits of a well agreed upon abstract model of computation

One of the main reasons to consider the von Neumann model a great idea is that by being a well agreed upon model, coordination amongst different people is minimized. That is,

- A computer architect doesn't need to know which programming languages will be run on their specific architecture. (The von Neumann model doesn't specify the instruction set, the memory management, how interupts are handled, etc.)
- A compiler writer for a programming language $\mathcal{L}$ doesn't have to know what algorithms will be implemented using the language $\mathcal{L}$.
- Without complete knowledge of the architecture, and the compiler, the algorithm designer can make a rough approximation for the memory and time requirements of their algorithm.

Progress in parallel computation had been relatively slow but there is now some common approaches (e.g., MapReduce for large scale parallel computation).

Dataflow architecture

Direct from Wikipedia:

Dataflow architecture is a computer architecture that directly contrasts the traditional von Neumann architecture or control flow architecture. Dataflow architectures do not have a program counter (in concept): the executability and execution of instructions is solely determined based on the availability of input arguments to the instructions,[1] so that the order of instruction execution is unpredictable, i.e. behavior is nondeterministic. Although no commercially successful general-purpose computer hardware has used a dataflow architecture, it has been successfully implemented in specialized hardware such as in digital signal processing, network routing, graphics processing, telemetry, and more recently in data warehousing.[citation needed] It is also very relevant in many software architectures today including database engine designs and parallel computing frameworks.[citation needed]

More on data flow architecture

From J. Paul Morrison's Flow-Based Programming text

The von Neumann machine is perfectly adapted to the kind of mathematical or algorithmic needs for which it was developed: tide tables, ballistics calculations, etc., but business applications are rather different in nature. . . .
Business programming works with data and concentrates on how this data is transformed, combined, and separated. . . . Broadly speaking, whereas the conventional approaches to programming (referred to as "control flow") start with process and view data as secondary, business applications are usually designed starting with data and viewing processes as secondary—processes are just the way data is created, manipulated, and destroyed. We often call this approach "data flow." (21)

Multicore and Parallel Computation

- As you probably already know, computers today are often multicore machines meaning that some "small" constant number of processes can be running simultaneously.
- When people refer to large scale parallelism hey have in mind that the number of processes running in parallel can depend (at least conceptually) on the computation.
- The von Neumann architecture is an abstract model for *sequential computation*.
- In contrast to the well accepted von Neumann model for sequential computation, the situation for *parallel computation* is more nuanced.
  - There is the issue of a constant number of parallel processes vs a number of processes that depends on the size of the data and/or the computation as it evolves.
  - Do the processes run syncronously (i.e. according to some golbal clock) or asynchronously?
  - Do the processes communicate mainly through a shared memory or via some communication bus?
  - How do we maintain consistency of the information being shared?

End of Monday September 20 class

We ended the Monday class a little after slide 7 where we very briefly mentioned fixed point repreeentation. For continuity we will continue on what is now slide 9.

Important discussion regarding in-person nature of this course.
Starting next week, I intend to mainly be conducting this course as an *in-person* course as it is listed. I expect students to normally show up in person as it is awkward trying to run a discussion course in a hybrid manner. If you do not plan to regularly attend in person you should consider dropping the course this week when you can still enrol in other courses. I hope everyone will stay in the course but want to have an honest understanding.
My plan is to run a zoom session for the one student who is not able to attend in person but just do so on my lap top and not try to integrate it with the class as we have been doing.
The tutorial sessions will continue to be remote only.

Fixed and Floating point representation

- As mentioned we have to approximate real non rational numbers by fractions. It is easy to see that say every real number can be approximated to arbitrary precision. In particular, every number in the interval $[0, 1]$ can be approximated by a fraction $.b_0 b_1 b_2 \ldots b_n$ 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.

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

  NOTE: I am going to simplify the discussion and ignore the special meaning when all the bits are 0's or 1's (which in the IEEE standard are used for special numebrs such as ∞). Let's call this the simplified representation. It will not effect the question on the assignment.

Floating point numbers continued

We will stick to binary notation but again any base can be used.

A floating point number uses the following representation (where I am using # just for clarity) as the bits would all be consecutive :

$s\#e_{k-1}e_{k-2}\ldots e_0\#b_{\ell-1}\ldots b_1 b_0$

Here the bit s determines the sign (i.e. $+$ or $-$) of the number.

The e_i bits represent the unbiased exponent E with value $E \in [0, 2^k - 1]$. The biased exponent $E' = E - (2^{k-1} - 1)$.

The b_i bits represent the significand (i.e, the significant bits)

The number being represented is
$(-1)^s \cdot 2^{E'} \cdot (1.b_{\ell-1}\ldots b_1 b_0)_2 = (-1)^s \cdot 2^{E'} \cdot (1 + \sum_{i=1}^{\ell} b_{\ell-i}2^{-i})$.
NOTE: There is an implicit "1" preceding the implicit binary point.

For an 8 bit word, with say a $k = 4$ bit exponent field and 3 bit significand, the integer -15 would be represented as
$11010111 = -1 \cdot 2^3 \cdot (1 + \frac{7}{8})$ since $E' = E - (2^{k-1} - 1) = 10 - 7 = 3$.

Floating point numbers continued

- The IEEE standard for a 32 bit *single precision* number uses 8 bits for the exponent and 23 bits for the significand (and therefore 24 bits of precision counting the implicit "1". .
- There are also double (and multiple) precision numbers where a double precision number would occupy two 32 bit words.
- History: According to Wikipedia, Leonardo Torres y Quevedo used floating point numbers in his design of Babbage's Analytical Engine. See also the reference to Konrad Zuse who designed a computer in 1938 and later versions in 1941 who using floating point numbers.
- It is interesting to note that von Neumann argued for fixed point numbers (and not floating point) in the design for an Institute of Advanced Study machine.
- It is important to note that an algorithm designer (usually) doesn't need to know the specifics of the fixed point and floating point representations. but just needs to know the commands for specifying the type (i.e. integer using fixed point or "real" using floating point) of the number.

Wikipedia

Is Wikipedia a great idea?

Wikipedia

Is Wikipedia a great idea?

In a Netflix documentary that I saw, 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 a Netflix documentary that I saw, 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 reassonably 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?

Wikipedia

Is Wikipedia a great idea?

In a Netflix documentary that I saw, 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 reassonably 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?

As you can see, I tend to use and trust Wikipedia especially about mathematical and computational definitions and historical information.

Further discussion of Wikipedia

In class there was an active discussion as to how much to trust Wikipedia.

It was mentioned that in high school, students are sometimes told "not to trust Wikipedia". But does the teacher mean just to do not take this as the only source and use it as an introduction to the topic and follow the references?

It was also mentioned that Wikipedia has a hierarchy of board members and contributors to help improve articles and resolve disputes.
Wikipedia articles sometimes requests more contributions about the article.

Wikipedia does ask for contributions to pay for the administrative costs. But I do not think this biases what is posted on Wikipedia. (Or maybe I am naive).

It could be that in some number of years the quality and trustworthiness of Wikipedia will decline. We will see.

If ypu have had a "bad experience" with Wikiepedia (i.e. an article that was factually wrong or misleading), please send it to me.

Data Types

Let's give an informal definition of what we mean by a data type. Namely, a data type is a collection of items (data) and the alllowable operations,relations and queries 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 $+, -, *, \div$, exponentiation and perhaps logarithms. We also have the relations $<, =, >$.

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

We will next 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. The information ablout a person can be called a *record*. 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 (unique) identifier (e.g., their social insurance number).

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 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 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.

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. 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.