#### **Great Ideas in Computing**

#### University of Toronto CSC196 Fall 2023

Week 2: September 18-September 22 (2023)

# Week 2

This weeks agenda:

- Beyond the von Neuman sequential model: Multicore and parallel computation
- The dictionary data type and data structures for a dictionary. This may not be a "great idea" but it is a good idea that relates to a few recurring themes in computing; namely, abstraction, complexity of an algorithm, tradeoffs, graph theory concepts, randomization.
- I hope to post the start of Assignment A1 due October 6 at 9AM. Why 9 AM?

Next weeks schedule

- On Monday, September 25, there will be a tutorial
- I will be doing the class on Wednesday and Friday

# Multicore and Parallel Computation

- As you may 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 global 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?

### **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<sub>0</sub>b<sub>1</sub>b<sub>2</sub>...b<sub>n</sub> where b<sub>i</sub> ∈ {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  $b_7b_6...b_0$ , 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*.

# Simplifying the discussion of floating point numbers

**NOTE:** I am going to first 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  $\infty$ ).

Then I will simplify the representation of the exponent field.

We will continue to use binary (i.e., base 2) representation but again any base could be used.

### Floating point numbers continued

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

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

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  $(-1)^s \cdot 2^{E'} \cdot (1 \cdot b_{\ell-1} \dots b_1 b_0)_2 = (-1)^s \cdot 2^{E'} \cdot (1 + \sum_{i=1}^{\ell} b_{\ell-i} 2^{-i}).$ **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.

### The simplified representation for the assignment

Lets define a slightly simplified version as follows:

Consider again, a number x represented by  $s \# e_{k-1} e_{k-2} \dots e_0 \# b_{\ell-1} \dots b_1 b_0$ 

Now let  $e_{k-1}$  be the sign of the exponent so that  $x = (-1)^s \cdot 2^{E'} \cdot (1 + \sum_{i=1}^{\ell} b_{i-1} 2^{-i})$  where now  $E' = (-1)^{e^{k-1}} \cdot E$ . That is we are now letting the leading bit of the exponent be the sign of the exponent rather than using a biased exponent.

Let's again consider an 8 bit number with a k bit exponent field and a 3 bit significand.

Now the floating point word 11010111 is equal to  $-1 \cdot 2^{-2} \cdot (1 + \frac{7}{8}) = -\frac{1}{4} \cdot \frac{15}{8} = -\frac{15}{32}.$ 

And the decimal integar 15 is represented by 10011111.

From time to time, I will take a definition or an explanation from Wikipedia. Is Wikipedia a great idea?

From time to time, I will take a definition or an explanation from 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 reply page when they made a query using Wikipedia?

The documentary notes that when we are on social media we are often getting personalized news-feeds.

From time to time, I will take a definition or an explanation from 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 reply page when they made a query using Wikipedia?

The documentary 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?

From time to time, I will take a definition or an explanation from 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 reply page when they made a query using Wikipedia?

The documentary 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 will see, I tend to use and trust Wikipedia especially about mathematical and computational definitions and historical information.

# Further discussion of Wikipedia

In the past, students have told me that in high school, students are sometimes told "not to trust Wikipedia". But does that mean just to do not take this as the only source but rather 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. Or maybe it will even get better. We will see.

If you have had a "bad experience" with Wikipedia (i.e. an article that was factually wrong or misleading), please discuss it.

# Will Large Language Models (Chat-GPT) make Wikipedia obsolete?

As I have indicated, I am generally very imperessed with Wikipedia when I need to get some specific non-controversial information.

But why not get the same information from say Chat-GPT and thereby make Wikipedia obsolete?

# Will Large Language Models (Chat-GPT) make Wikipedia obsolete?

As I have indicated, I am generally very imperessed with Wikipedia when I need to get some specific non-controversial information.

But why not get the same information from say Chat-GPT and thereby make Wikipedia obsolete?

I decied to ask ChapGPT3.5 what it "thought"

# The answer from ChapGPT

Chat-GPT and similar AI models are unlikely to make Wikipedia obsolete. While these AI models can generate text and provide information on a wide range of topics, there are several reasons why Wikipedia will likely continue to be a valuable resource:

Human Verification and Quality Control: Wikipedia relies on a large community of human editors who review and verify the content. These editors ensure that the information presented on Wikipedia is accurate, neutral, and reliable. Al models like Chat-GPT, while powerful, can generate incorrect or biased information, and they lack the ability to fact-check or verify the quality of their content.

Structured and Organized Information: Wikipedia organizes information in a structured manner, with articles, sections, citations, and references. It provides a clear and organized format for users to access information. Chat-GPT, on the other hand, generates text in a conversational format, which may not be as structured or easy to navigate. Community Engagement: Wikipedia encourages community engagement and collaboration. People from around the world contribute to Wikipedia articles, add references, update information, and discuss content on talk pages. This collective effort ensures that Wikipedia remains a dynamic and up-to-date resource.

Comprehensive Coverage: Wikipedia covers a vast array of topics, from historical events and scientific concepts to biographies of individuals and cultural references. While AI models like Chat-GPT have access to a broad range of information, they may not provide the same depth and breadth of coverage that Wikipedia offers.

Multilingual Support: Wikipedia is available in multiple languages and is a valuable resource for people worldwide. While AI models can generate text in various languages, the quality and accuracy of translation and language nuances can be challenging for AI to handle effectively.

Educational Value: Wikipedia is often used as an educational tool in schools and universities. It provides students with a platform for research and learning. Al models like Chat-GPT may be used as a complement to educational resources, but they are not a substitute for comprehensive educational materials.

Curation and Citations: Wikipedia emphasizes the importance of citing reliable sources. Information on Wikipedia articles is backed by citations to reputable publications and references. Al models do not have the capability to cite sources or provide the same level of rigor in referencing.

In summary, AI models like Chat-GPT can be valuable for generating text and providing information, but they are not a replacement for the unique qualities and human-driven aspects of Wikipedia. Wikipedia's collaborative and structured approach, along with its emphasis on accuracy and quality control, makes it a valuable resource that is likely to continue to coexist with AI-generated content.

**Warning**: Free Research Preview. ChatGPT may produce inaccurate information about people, places, or facts. ChatGPT August 3 Version  $_{15/32}$ 

# **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 about 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 are 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*.

#### End of Monday, September 18 class

We have just defined the dictionary data type. In the next class we will discuss different data structures for implementing a dictionary.

#### Announcements

- First two question of Assignment A1 are posted. More question to follow. Notice the 20% rule.
- I started fixing some of the broken links. But it requires finding reasonable substitutes for the broken links and this will take time. I hope by next week, all links will be operational.
- As announced there is a tutorial on Friday and depending on the class, we will decide if another tutorial on Monday, the 25th will take place. I will be doing the class on Wednesday and Friday next week.

### Dictionaries lead to interesting concepts and ideas

• Many ways to implement a dictionary. What is important is to again note 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 is to again note 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.
  - An unordered list in an array
  - 2 An ordered list in an array
  - A linked list
  - A (balanced) search tree.
  - 6 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

**Note:** You were essentially using a Dictionary when you found your name at the start of class.

How did you search the set of names

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

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

Polynomial-Time Algorithms Take Better Advantage of Computation Time

**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.

Introduces the idea of a pointer

Introduces the idea of a pointer

I wil show a singly linked list on the board. Can have a doubly linked list.

Introduces the idea of a pointer

I wil show a singly linked list on the board. 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.

Introduces the idea of a pointer

I wil show a singly linked list on the board. 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 binary search tree

Informally, 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. We will elaborate soon on search trees.

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.

The choice of any particular data structure will depend on the possible intended applications (assuming that is known).

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

We have a hash function  $h: I \to M$  where  $I = \{ID_1, \dots, ID_N\}$  is the set of all possible integer identifiers and  $M = \{A[0], \dots, A[m-1]\}$  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 possibility is  $h(ID) = (a \cdot ID + b)(mod \ p)(mod \ m)$  where p is a large prime.

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

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.

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.

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.

#### **Concluding remark**

The choice of any particular data structure or algorithm will usually depend on the application. For example, in choosing a data structure what operations are being done more often than others is a consideration.