Data Structures and Algorithms — 66-230
Discrete Mathematics Overview

Introduction

During the first three weeks of the semester we will cover some techniques from discrete mathematics relevant to computer science in general and algorithm analysis in particular. This material is somewhat more extensive than offered in Chapter 1 of the Weiss text. It does not even begin to substitute for a course in discrete mathematics, however. Students, especially those interested in graduate school in computer science, should consider taking a separate discrete mathematics course during their undergraduate careers.

Sets

- A set is an **unordered** collection of distinct objects (called *elements*). These elements may be anything, including other sets.

- There are many special sets with special designations. $\boldsymbol{Z}$ is the set of integers. $\boldsymbol{N}$ is the set of natural numbers (integers beginning with 0). $\boldsymbol{R}$ or $\Re$ is the set of real numbers. The set with no elements, often called the "null set" or the "empty set," is denoted $\{\}$.

- The elements of a set may be specified in a number of ways. The simplest one to list the elements, e.g.,

  $$V = \{a, u, e, i, o\}.$$

  We use the notation $a \in V$ to say "a is an element of V". Another way to specify the elements of a set is to describe them logically using what's referred to as "set-builder notation". For example,

  $$S = \{x \mid x \in \boldsymbol{R} \text{ and } 0 \leq x \leq 1\}$$

  is the set of **all** real numbers in the interval from 0 to 1 inclusive.

- A set S is a *subset* of another set T if and only if each element of S is also an element of T. This is denoted by $S \subseteq T$.

- The *cardinality* of a set S, denoted by $|S|$, is the number of distinct elements of S.

- New sets may be formed from existing sets, S and T, by a variety of operations, such as union ($S \cup T$), intersection ($S \cap T$), set difference ($S - T$) and set complement $\overline{S}$. Set complement requires the notion of a "universal set" U from which set elements can be drawn.

  Most of these set operations should be familiar. The only one that might be new is set difference:

  $$S - T = \{x \mid x \in S \text{ and } x \notin T\}.$$

- Two sets, A and B, are said to be *disjoint* if $A \cap B = \{\}$.

In-class problems

Work on these problems briefly. Most should be straightforward. We will discuss the solutions in class.

1. Given $S = \{a, b\}$ and $T = \{0, a, \{1, 2\}\}$.
   (a) List the elements of T.
   (b) List all subsets of S.
   (c) What is $|S|$? What is $|T|$?
   (d) What are $S \cap T$ and $S - T$?

2. Let S be the set of items on your shopping list and let T be the set of items you actually bought during a trip to the store. Describe each of the following in English.
   (a) $S \cap T$
   (b) $T - S$
   (c) $S - T$

Functions

- A function, f, is a mapping from one set, A, called the *domain*, to another set, B, called the *co-domain*, that associates **each** element of A with a **single** element of B. We write this as

  $$f\colon A \to B$$

  If $a \in A$ and f maps a to b, we write $f(a) = b$.

- The *range* of f is the set containing all $b \in B$ such that $f(a) = b$ for some $a \in A$.

- Special functions of interest

  - Floor: $f(x) = \lfloor x \rfloor$. This is the largest integer less than or equal to x.
  - Ceiling: $f(x) = \lceil x \rceil$. This is the smallest integer greater than or equal to x.
  - Factorial: $f(n) = n!$. Note that the domain of this function is the natural numbers.

- Exponentiation (base b): $f(x) = b^x$. Rules for manipulation:

  $$\begin{aligned} b^x b^y &= b^{x+y} \\ \frac{b^x}{b^y} &= b^{x-y} \\ (b^x)^y &= b^{xy} \neq b^{(x^y)} \end{aligned}$$

- Logarithm (base $b > 0$): $f(x) = \log_b x$. This means, $y = \log_b x$ if $b^y = x$. If b is unspecificed, 2 is assumed. Rules for manipulation:

  $$\begin{aligned} \log_a b &= \frac{\log_c b}{\log_c a} \qquad \text{for } c > 0 \\ \log xy &= \log x + \log y \\ \log(x^y) &= y \log x \neq (\log x)^y \\ \log x &< x \qquad \text{for all } x > 0 \end{aligned}$$

In-Class Problems on Functions

1. What are

   (a) $5!$

   (b) $\lceil 2 \rceil$, $\lceil 2.1 \rceil$, $\lceil -2.1 \rceil$,

   (c) $\lfloor 2 \rfloor$, $\lfloor 2.1 \rfloor$, $\lfloor -2.1 \rfloor$?

2. What is the derivative of $\log_b x$? (Hint: recall that the derivative of the natural logarithm is $1/x$.)

Summations

- A summation is a short-hand notation to describe the addition of the terms of a sequence:

$$\sum_{i=m}^{n} f(i) = f(m) + f(m+1) + f(m+2) + \ldots + f(n)$$

  The variable i is called the index variable. Most often $m = 0$ or $m = 1$. Sometimes a summation is easiest to think about by thinking about the `for` loop you would write to calculate it.

- Special summation formulas:

$$\sum_{i=0}^{n} 2^i = 2^{n+1} - 1$$

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1} \qquad \text{for } a \neq 1$$

$$\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$$

$$\sum_{i=1}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{i=1}^{n} \frac{1}{i} \approx \log_e n$$

- Simple rules for manipulating summations. These rules may perhaps be best understood by writing the summation out long hand and then using standard rules of algebra.

$$\sum_{i=m}^{n}[f(i)+g(i)] = \left(\sum_{i=m}^{n} f(i)\right) + \left(\sum_{i=m}^{n} g(i)\right)$$
$$\sum_{i=m}^{n} a \cdot f(i) = a \cdot \sum_{i=m}^{n} f(i)$$
$$\sum_{i=m}^{n} a = a \cdot (n-m+1)$$
$$\sum_{i=m}^{n} f(i) = \sum_{i=0}^{n} f(i) - \sum_{i=0}^{m-1} f(i)$$
$$\sum_{i=m}^{n} f(i) = \sum_{j=0}^{n-m} f(j+m)$$

  In these formulas, a stands for an arbitrary but fixed constant (i.e. something that doesn't change when i changes).

Problems on summations

1. Write each of the following as a summation.

   (a) $5 + 10 + 15 + 20 + \ldots + 1000$

   (b) $3 + 8 + 13 + 18 + \ldots + 98$

   (c) $4 + 8 + 16 + 32 + \ldots + 1024$

2. Derive a formula for each of the following that does not involve a summation and does not involve i. Use the rules for manipulation and the special formulas.

   (a) $\displaystyle\sum_{i=1}^{n} 4i$.

   (b) $\displaystyle\sum_{i=3}^{n} (5i + 4)$.

   (c) $\displaystyle\sum_{j=0}^{k} (4^j + k)$.

   (d) $\displaystyle\sum_{i=0}^{N} (2 + \sum_{j=0}^{i} j)$

3. (Extra challenge!) Derive a formula for the following that does not involve a summation:

$$\sum_{i=1}^{N} \frac{i}{2^i}.$$

   Hint: use the technique given in class for proving that

$$\sum_{i=0}^{n} a^i = \frac{a^{n+1} - 1}{a - 1}.$$

Proofs

- A proof is a clear, logical demonstration of the truth of an assertion based on previously known facts (established propositions, lemmas and theorems).

- In writing a proof, the first step is to carefully determine what the assertion says, including the meanings of all terms used.

  For example, you can not prove that 2 is the only even prime number without understanding what it means for a number to be "even" and to be "prime".

  Once you understand what is to be proved, the next step is to write the actual proof. This is a bit of an art form, which we will not have time to investigate thoroughly. But, ...

- Proof techniques we will explore:

  - Direct proof. Starting from the assumptions — e.g. n is an even positive integer — construct a series of deductions leading to the claim — e.g. n is not prime.
  - Proof by example or counter-example. This is acceptable when proving that there is a member of a set satisfying an assertion, or when proving that not all members of a set satisify an assertion. It is not acceptable when proving that all members of an infinite set satisfy an assertion.
  - Proof by contradiction. This is done by assuming that the assertion to be proved is false and showing that the assumption leads to a statement that must be both true and false (a contradiction).
  - Proof by induction. This technique is closely related to data structure and algorithm analysis and is covered in detail below.

In-Class Exercises on Proofs

1. Is it true that $n^3 > 2^n$ for all integers $n > 1$? Prove your result.
2. Prove that 2 is the only even prime number.

Proof by Induction

- Mathematical induction is used to prove statements true for all integers greater than or equal a certain base value n_0. Here are three example statements we will prove in class using mathematical induction:
  - $\displaystyle\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ for $n \geq 0$ and $a \neq 1$.
  - $n! < n^n$, for $n \geq 2$.
  - A complete binary tree of height $h \geq 0$ contains exactly $2^{h+1} - 1$ nodes.

- An inductive proof always has three main components:

  Basis case: Prove the statement is true for $n = n_0$ (and perhaps $n = n_0 + 1$, etc.).

  Inductive hypothesis: Assume the statement is true for all k, $n_0 \leq k < n$.

  Induction step: Prove that for all $n \geq n_0$, the inductive hypothesis implies that the statement is true for n.

- The principle of mathematical induction states that if the basis case and induction step are both proved, then the statement holds for all $n \geq n_0$.

- Notes:
  - At first proofs by induction appear to be circular, but in fact they are not. Instead, think of a set of dominoes standing on end: the basis case is the act of pushing over the first domino (or two); the induction step is the relationship between the positions of the dominoes showing that for every domino, if its precedecessors fall over, it will fall over as well.
  - Sometimes the inductive hypothesis is written $n_0 \leq k \leq n$ and the induction step proves that the inductive hypothesis shows the statement to be true for $n + 1$.
  - At other times, the inductive step is to show for all $n \geq n_0$, if the statement is true for n then it is true for $n + 1$. Students more comfortable with this method of inductive proof are welcome to use it here.
  - Very often the inductive hypothesis is not written out explicitly.

- For completeness, here are written out versions of the three example proofs by induction given in class.

- $\displaystyle\sum_{i=0}^{n} a^i = \frac{a^{n+1}-1}{a-1}$ for $n \geq 0$ and $a \neq 1$.

  Basis case: When $n = 0$, the summation reduces to $a^0 = 1$, and the right side of the equation is $(a^{0+1}-1)/(a-1) = 1$. Since these are both 1, the basis case is proved.

  Induction step: For any given $n \geq 1$, if the statement is true for $0 \leq k < n$, then

$$\begin{aligned}
\sum_{i=0}^{n} a^i &= \sum_{i=0}^{n-1} a^i \ + \ a^n \\
&= \frac{a^{(n-1)+1}-1}{a-1} \ + \ a^n && \text{by the induction hypothesis} \\
&= \frac{a^n-1}{a-1} \ + \ a^n\frac{a-1}{a-1} \\
&= \frac{a^n-1+a^{n+1}-a^n}{a-1} \\
&= \frac{a^{n+1}-1}{a-1}
\end{aligned}$$

  completing the induction step.

- $n! < n^n$, for $n \geq 2$.

  Basis case: For $n = 2$, the left side of the inequality evaluates to 2 and the right side evaluates 4. Since $2 < 4$, the basis case is established.

  Induction step: For $n > 2$, if $k! < k^k$ for $2 \leq k < n$ then in particular

$$(n-1)! < (n-1)^{n-1}.$$

  Multiplying both sides of this inequality by n shows that

$$n! < n \cdot (n-1)^{n-1}.$$

  But, since $n - 1 < n$ (obviously),

$$n! < n \cdot (n-1)^{n-1} < n \cdot n^{n-1}.$$

  So,

$$n! < n^n.$$

- A complete binary tree of height $h \geq 0$ contains exactly $2^{h+1} - 1$ nodes.

  Basis case: Any binary tree of height $h = 0$ is complete and has exactly one node. Since $2^{0+1} - 1 = 2 - 1 = 1$, the basis case is established.

  Induction step: For $h > 0$, assume any complete binary tree of height k, $0 \leq k \leq h-1$, contains $2^{k+1} - 1$ nodes, and consider any complete binary tree, T, of height h. Removing the root of T creates two complete binary trees of height $h-1$, one formed from the left subtree of T and the other formed from the right. By the inductive hypothesis each of these has $2^{(h-1)+1} - 1 = 2^h - 1$ nodes. Therefore, T had

  $$(2^h - 1) + (2^h - 1) + 1$$

  nodes to start with (the 1 at the end is for the root node). Adding these values shows that T had

  $$2^{h+1} - 1$$

  nodes, which completes the proof.

Class Exercises on Induction

1. Prove each of the following by mathematical induction.

   (a) $\sum_{i=1}^{n} 2i - 1 = n^2$ for $n \geq 1$.

   (b) $n < 2^n$ for $n \geq 0$.

   (c) Any set containing n elements has 2^n different possible subsets. (Hint: in the inductive step, consider a particular element a of S and count all the subsets that do contain a and all those that do not.)

2. What is wrong with the following inductive proof showing that all horses are the same color? (Or, more specifically, in any set H of n horses, each horse in H has the same color as every other horse in H.)

   Basis: $n = 1$. One horse is trivially the same color as itself.

   Induction Step: We have to prove for each $n \geq 2$ that if in all sets containing fewer than n horses all horses are the same color, then in any set of n horses all horses are the same color. Let H be any set containing n horses, and label the horses $h_1, h_2, \ldots, h_n$. Let $H_1 = H - \{h_n\}$ and let $H_2 = H - \{h_1\}$. By the inductive hypothesis, since $|H_1| = n - 1$, all horses in H_1 are the same color; call it c_1. Also by the inductive hypothesis, since $|H_2| = n - 1$, all horses in H_2 are the same color; call it c_2. Also, since $H_1 \cap H_2 = \{h_2, \ldots, h_{n-1}\}$, horses $h_2, \ldots, h_{n-1}$ are each both colors c_1 and c_2. Hence, $c_1 = c_2$. Therefore, all horses in H are the same color.

Recursion

Here is an outline of five steps I find useful in writing and debugging recursive functions:

1. Handle the base case(s) first, at the start of the function.
2. Define the problem solution in terms of smaller instances of the problem. This defines the necessary recursive calls.
3. Figure out what work needs to be done before making the recursive call(s).
4. Figure out what work needs to be done after the recursive call(s) complete(s) to finish the solution.
5. Assume the recursive calls work correctly, but make sure they are progressing toward the base case(s)!

Here are two example recursive functions: factorial and mergesort.

```
int Fact( int n )
{
  if ( n == 0 || n == 1 )
    return 1;
  else
    return n * Fact( n-1 );
}
```

```
template <class T>
void MergeSort( T * pts, int n )
{
  MergeSort( pts, 0, n-1 );
}

templage <class T>
void MergeSort( T * pts, int low, int high )
{
  if ( low == high ) return;

  int mid = (low + high) / 2;
  MergeSort( T, low, mid );
  MergeSort( T, mid+1, high );

  //  At this point the lower and upper halves
  //  of "pts" are sorted. All that remains is
  //  to merge them into a single sorted list.
  //  We will discuss the details of this later.
}
```

Class Exercises on Recursion

Now, here are some practice exercises to review recursion.

1. Assume the following structure for a node in a binary tree:

```
struct TreeNode {
  int value;
  TreeNode * left_child;
  TreeNode * right_child;
};
```

   Write a recursive function to count the number of nodes in the binary tree whose root is pointed to be T. Note the structural similarity between your solution and the inductive proof about complete binary trees.

2. The following is a recursive version of binary search. Will it work correctly? Why or why not?

```
template <class T>
int RBinSearch ( T * pts, int low, int high, T point, int& loc )
{
  if ( high == low ) {
    loc = low;
    return pts[loc] == point;
  }
  int mid = (low + high) / 2;
  if ( pts[mid] <= point )
    return RBinSearch( pts, mid, high, point, loc );
  else
    return RBinSearch( pts, low, mid-1, point, loc );
}
```

3. The Fibonacci numbers are defined recursively as

$$
\begin{aligned}
f_0 &= 0 \\
f_1 &= 1 \\
f_n &= f_{n-1} + f_{n-2}, \qquad n \geq 2
\end{aligned}
$$

   (The notation f_n means the same thing as $f(n)$, i.e. f is a function of n.)

   (a) Write a recursive function to calculate f_n.

   (b) How many recursive calls will your function make to calculate f_3, f_4, f_5, etc? Can you write a recursive expression similar to the recursive formula for the Fibonacci numbers to count the number of recursive calls?

Review Problems

Here are a few review problems which have appeared on homeworks or tests in previous semesters. Practice writing solutions carefully and then compare to solutions provided on-line. If you can solve these problems and the problems we worked on in class then you are ready for the chapter quiz!

1. Give an inductive proof showing that for all integers $n \geq 5$, $4n + 4 < n^2$.

2. Give an inductive proof showing that for all positive integers n,
$$\sum_{i=1}^{n} \frac{1}{(2i-1)(2i+1)} = \frac{n}{2n+1}.$$

3. Give an inductive proof showing that for all $n \geq 1$,
$$\sum_{i=1}^{n} i(i!) = (n+1)! - 1$$

4. Evaluate the following summations.
   (a) $\sum_{i=0}^{100} -1^i$
   (b) $\sum_{i=5}^{2n} i$
   (c) $\sum_{i=0}^{n} (a^i - 2i + n)$
   (d) $\sum_{i=0}^{n-1} \big(c + \sum_{j=0}^{i-1} (j+2)\big)$

5. Prove using mathematical induction that $f_n > (3/2)^n$, for $n \geq 5$. Here, f_n is the nth Fibonacci number. Use the definition of the Fibonacci numbers given in the text. Study the example inductive proof on page 6 carefully.

6. Write the code necessary to accomplish the merging step in `MergeSort`. This code should follow the comments in the main `MergeSort` function.