EECS 4101

Prerequisites

This is a summary of basic concepts that you should be familiar with before taking this course. Most of the material here is from your discrete math course, the basic data structures course and your algorithms course.

1 Sets

The Basics

Intuitively, a set is a collection of objects. (The objects are called the *members* or *elements* of the set.) This is not a proper definition of "set" because "collection" is just a different word for "set", but the notion of a set is so basic that it is difficult to give a proper definition of what it means. The sets we deal with in this course will be fairly simple (as far as sets go), so this intuitive description of what a set is will be sufficient for our purposes.

If a set is finite, we can specify it by listing its elements explicitly. For example, the set of the smallest five positive integers can be written down as $\{1, 2, 3, 4, 5\}$.

Some important infinite sets are:

- $\mathbb{N}$: the set of all natural numbers, $\{0, 1, 2, \dots\}$
- $\mathbb{Z}$: the set of all integers, $\{\dots, -2, -1, 0, 1, 2, \dots\}$
- $\mathbb{Z}^+$: the set of all positive integers, $\{1, 2, 3, \dots\}$
- $\mathbb{Q}$: the set of all rational numbers
- $\mathbb{R}$: the set of all real numbers

Both finite and infinite sets can be specified using set-builder notation. Suppose $S(x)$ is a predicate with one variable, x. (Recall that, for each possible value we can assign to x, $S(x)$ becomes a statement that is either true or false.) Then, the notation $\{x : S(x)\}$ denotes the set of all elements x for which the statement $S(x)$ is true. In this notation, x is a dummy variable. This means that $\{y : S(y)\}$ is exactly the same set as $\{x : S(x)\}$.

Examples:

- $\{x : x \text{ is one of the five smallest positive integers}\} = \{1, 2, 3, 4, 5\}$
- $\{x : x \text{ is an integer and } x^2 = 4\} = \{-2, 2\}$
- $\{x : x \text{ is an integer and } x^2 > 4\}$ is an infinite set containing all of the integers, except $-2, -1, 0, 1$ and 2.

If x is an element of set A, we write $x \in A$.

When using set-builder notation, we often give the domain (*i.e.*, the set of possible values the variable can take) before the colon. For example,

- $\{x \in \mathbb{N} : x^2 = 4\} = \{2\}$ and
- $\{x \in \mathbb{Z} : x^2 = 4\} = \{-2, 2\}$.

Two sets A and B are considered equal if every element of A is in B and every element of B is in A. In particular, this means that the ordering of elements in the set is not important. For example, $\{1,2,3,4,5\} = \{4,3,2,5,1\}$. Also repetitions of an element in a set are not important. For example, $\{1,1,2,2,3,3,4,4,5,5\} = \{1,2,3,4,5\}$.

Notation

For any sets A and B, we shall use the following standard notation.

- $\varnothing$ or $\{\}$: "the empty set"; *i.e.*, the set that contains no elements.
- $x \in A$: "x is an element of A".
- $x \notin A$: "x is not an element of A".
- $A \subseteq B$: "A is a subset of B"; *i.e.*, every element of A is also in B.
- $A = B$: "A equals B"; *i.e.*, $A \subseteq B$ and $B \subseteq A$.
- $A \cap B$: "A intersect B"; *i.e.*, the set $\{x : x \in A \text{ and } x \in B\}$.
- $A \cup B$: "A union B"; *i.e.*, the set $\{x : x \in A \text{ or } x \in B\}$.
- $A \setminus B$ or $A - B$: "A minus B" (*set* difference); *i.e.*, the set $\{x : x \in A \text{ and } x \notin B\}$.
- $\overline{A}$: "the complement of A". This assumes there is some domain D from which the items in A are chosen; then $\overline{A} = D - A$.
- $|A|$: "cardinality of A" (intuitively, the number of elements in A; we shall talk about what this means for infinite sets later in the course.)
- $\mathscr{P}(A)$: "the power set of A"; *i.e.*, the set of all subsets of A; $\mathscr{P}(A) = \{B : B \subseteq A\}$.

Cartesian Products

An ordered pair is a sequence of two objects in a specified order. We write an ordered pair consisting of the objects a and b (in that order) as (a,b). Two ordered pairs (a,b) and (a',b') are equal if and only if $a = a'$ and $b = b'$.

If A and B are sets, the set of all ordered pairs whose first element is an element of x and whose second element is an element of B is called the Cartesian product of A and B, written $A \times B$. Thus, $A \times B = \{(a,b) : a \in A \text{ and } b \in B\}$.

More generally, if k is a positive integer, a k-tuple is a sequence of k objects in a specified order. (Thus, a 2-tuple is just an ordered pair.) We use similar notation for k-tuples as for ordered pairs. For example, $(3,2,1,7)$ is an ordered 4-tuple. Two k-tuples $(a_1, a_2, \ldots, a_k)$ and $(a'_1, a'_2, \ldots, a'_k)$ are equal if and only if for every i in $\{1,2,\ldots,k\}$, $a_i = a'_i$. The Cartesian product of k sets is a set of k-tuples: $A_1 \times A_2 \times \cdots \times A_k = \{(a_1, a_2, \ldots, a_k) : a_i \in A_i \text{ for } 1 \leq i \leq k\}$.

2 Functions

Formally, a function $f : A \to B$ is a set $f \subseteq A \times B$, where, for each $x \in A$, there is exactly one element $y \in B$ such that $(x,y) \in f$. If $(x,y) \in f$, we use the notation $f(x)$ to denote y. This means that f associates with each value $x \in A$ a unique value $f(x)$. A is called the *domain* of the function f. Note that the domain can be a Cartesian product of other sets: in this case we use the notation $f(x,y)$ instead of $f((x,y))$.

Properties of Functions

A function $f : A \to B$ is called

- *surjective* (or *onto*) if, for every $z \in B$, there is some $x \in A$ such that $f(x) = z$.
- *injective* (or *one-to-one*) if $x \neq y$ implies $f(x) \neq f(y)$.
- *bijective* (or *a one-to-one correspondence*) if it is surjective and injective.

Let $f : A \to B$, where A and B are subsets of $\mathbb{R}$. The function f is called

- *(strictly) increasing* if $f(x) < f(y)$ whenever $x < y$.
- *(strictly) decreasing* if $f(x) > f(y)$ whenever $x < y$.
- *non-decreasing* if $f(x) \leq f(y)$ whenever $x < y$.
- *non-increasing* if $f(x) \geq f(y)$ whenever $x < y$.
- *monotone* if it is either increasing or decreasing.

Common Functions

Here are some common functions together with their definition and properties (unless noted otherwise, x and y stand for arbitrary real numbers and k, m, and n stand for arbitrary positive integers in what follows).

- $\min(x, y)$: "minimum of x and y" (the smallest of x or y). More generally, if A is a set of real numbers, $\min(A)$ denotes the smallest element in set A, if such a smallest element exists.
- $\max(x, y)$: "maximum of x and y" (the largest of x or y). More generally, if A is a set of real numbers, $\max(A)$ denotes the largest element in set A, if such a largest element exists.
- $\lfloor x \rfloor$: "floor of x" (the greatest integer less than or equal to x. For example, $\lfloor 5.67 \rfloor = 5$ and $\lfloor -2.01 \rfloor = -3$).

  Properties: $x - 1 < \lfloor x \rfloor \leq x$, $\lfloor -x \rfloor = -\lceil x \rceil$, $\lfloor x + k \rfloor = \lfloor x \rfloor + k$, $\left\lfloor \frac{k}{m} \right\rfloor \geq \frac{k}{m} - \frac{m-1}{m}$.
- $\lceil x \rceil$: "ceiling of x" (the smallest integer greater than or equal to x. For example, $\lceil 5.67 \rceil = 6$ and $\lceil -2.01 \rceil = -2$).

  Properties: $x \leq \lceil x \rceil < x + 1$, $\lceil -x \rceil = -\lfloor x \rfloor$, $\lceil x + k \rceil = \lceil x \rceil + k$, $\left\lceil \frac{k}{m} \right\rceil \leq \frac{k}{m} + \frac{m-1}{m}$.

  Additional property of $\lfloor \rfloor$ and $\lceil \rceil$: $\lfloor k/2 \rfloor + \lceil k/2 \rceil = k$; prove this by considering two cases, when k is even or odd.
- $|x|$: "absolute value of x" ($|x| = x$ if $x \geq 0$; $-x$ if $x < 0$, *e.g.*, $|5.67| = 5.67$, $|-2.01| = 2.01$)

  Do not confuse this with the notation for the cardinality of a set.
- $m \operatorname{div} n = \lfloor m/n \rfloor$, the integer portion of m divided by n. For example, $5 \operatorname{div} 6 = 0$ and $27 \operatorname{div} 4 = 6$.
- $m \bmod n = m - n \cdot \lfloor m/n \rfloor$: the remainder of m divided by n. For example, $5 \bmod 6 = 5$ and $27 \bmod 4 = 3$.

  Properties: $m = (m \operatorname{div} n) \cdot n + m \bmod n$, $0 \leq m \bmod n < n$.

3 Sums

Summation Notation

The notation $\sum_{i=a}^{b} t_i$, where a and b are integers with $b \geq a$ and the t_i's are real numbers, is used to denote the sum $t_a + t_{a+1} + t_{a+2} + \cdots + t_b$. The notation is defined formally by

$$\begin{aligned} \sum_{i=a}^{b} t_i &= t_a, \text{ if } b = a, \text{ and} \\ \sum_{i=a}^{b} t_i &= t_a + \sum_{i=a+1}^{b} t_i, \text{ if } b > a. \end{aligned}$$

For example, $\sum_{i=3}^{n} 2^i = 2^3 + 2^4 + \cdots + 2^n = 2^{n+1} - 8$.

Similar notation will be used for other associative operations. For example, $\prod_{i=a}^{b} t_i$ represents the product $t_a \cdot t_{a+1} \cdot t_{a+2} \cdot \cdots \cdot t_b$ and $\bigcup_{i=a}^{b} A_i$, where the A_i's are sets, represents the union $A_a \cup A_{a+1} \cup A_{a+2} \cup \cdots \cup A_b$.

Computing Sums

You should know some common sums:

- $\sum_{i=0}^{n} i = \frac{n(n+1)}{2}$
- $\sum_{i=0}^{n} i^2 = \frac{n(n+1)(2n+1)}{6}$
- $\sum_{i=0}^{n} r^i = \frac{r^{n+1}-1}{r-1}$ for $r \neq 1$

Some identities are useful in computing sums:

- $\sum_{i=a}^{b} (ct_i + s_i) = c \sum_{i=a}^{b} t_i + \sum_{i=a}^{b} s_i$
- $\sum_{i=a}^{b} t_i = \sum_{i=0}^{b} t_i - \sum_{i=0}^{a-1} t_i$ (where $b \geq a \geq 0$).

Two simple kinds of changes of variables are also useful in computing sums (a, b and c are integers, with $b \geq a$):

- $\sum_{i=a}^{b} t_i = \sum_{j=a+c}^{b+c} t_{j-c}$ (here, $j = c + i$).
- $\sum_{i=a}^{b} t_i = \sum_{k=c-b}^{c-a} t_{c-k}$ (here, $k = c - i$).

Bounding Sums

See Section A.2 of the textbook.

4 Exponents and Logarithms

For any $a, b, c \in \mathbb{R}^+$, $a = \log_b c$ if and only if $b^a = c$.

For any $a, b, c \in \mathbb{R}^+$ and any $n \in \mathbb{Z}^+$, the following properties always hold.

- $\sqrt[n]{b} = b^{1/n}$
- $b^a b^c = b^{a+c}$
- $(b^a)^c = b^{ac}$
- $b^a / b^c = b^{a-c}$
- $b^0 = 1$
- $a^b c^b = (ac)^b$
- $b^{\log_b a} = a = log_b b^a$
- $a^{\log_b c} = c^{\log_b a}$
- $\log_b(ac) = \log_b a + \log_b c$
- $\log_b(a^c) = c \cdot \log_b a$
- $\log_b(a/c) = \log_b a - \log_b c$
- $\log_b 1 = 0$
- $\log_b a = \log_c a / \log_c b$

5 Asymptotic Notation

See Chapter 3 of the textbook.

6 Binary Notation

A *binary number* is a sequence of bits $a_k \cdots a_1 a_0$ where each bit a_i is either 0 or 1. Every binary number represents a natural number in the following way:

$$a_k \cdots a_1 a_0 \text{ represents } \sum_{i=0}^{k} a_i 2^i = a_k 2^k + \cdots + a_1 2 + a_0.$$

For example, 1001 represents $1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 1 \cdot 2^0 = 8 + 1 = 9$, and 01110 represents $8 + 4 + 2 = 14$. Each positive integer has a unique binary representation whose leftmost bit is 1.

Properties:

- If $a = (a_k \cdots a_1 a_0)_2$, then $2a = (a_k \cdots a_1 a_0 0)_2$, *e.g.*, $9 = (1001)_2$ so $18 = (10010)_2$.
- If $a = (a_k \cdots a_1 a_0)_2$, then $\lfloor a/2 \rfloor = (a_k \cdots a_1)_2$, *e.g.*, $9 = (1001)_2$ so $4 = (100)_2$.
- Each positive integer has a unique binary representation whose leftmost bit is 1.
- The smallest number of bits required to represent natural number n in binary is called the *binary length* of n and is equal to $\lceil \log_2(n+1) \rceil$.

7 Proofs

An important objective of EECS 4101 is to learn how to reason about data structures in a formal way. So there will be some opportunity to practice understanding and writing mathematical proofs. You should be familiar with basic proof techniques from your discrete math course.

If you do not feel too comfortable about this part of your background for the course, Daniel Solow's *How to Read and Do Proofs* (available from the library) is a very good introduction.

You should be familiar with the following proof techniques.

- Proof by cases: this just means that to prove a statement A is true under all possible circumstances, you may have to use different proofs for different circumstances. For example, to prove that for all real numbers x and y, $|x + y| \leq |x| + |y|$, you might want to consider several cases (for example, you could consider three cases: x and y are both positive, or both negative or one is positive and one is negative). When using this proof technique, you must be sure that you have covered all possible cases.
- Proof by contradiction: to prove A, derive a contradiction from $\neg A$.
- Direct proof of an implication: to prove $A \Rightarrow B$, assume A as a hypothesis and prove B.
- Proof of an implication by proving contrapositive: to prove $A \Rightarrow B$, prove $\neg B \Rightarrow \neg A$.
- Proof by mathematical induction. To prove a statement of the form $\forall n \in \mathbb{N}\ P(n)$, it is sufficient to prove
  1. $P(0)$, and
  2. $\forall n \geq 1, (P(n-1) \Rightarrow P(n))$.
- Proof by strong mathematical induction. To prove a statement of the form $\forall n \in \mathbb{N}\ P(n)$, it is sufficient to prove
  1. $P(0)$, and
  2. $\forall n \geq 1, ((\forall x < n\ P(x)) \Rightarrow P(n))$.
- Proof of existence by construction: to prove $\exists x\ P(x)$, explicitly construct an x and show that $P(x)$ is true for it.
- Proof of universally quantified statement by generalization: to prove $\forall x\ P(x)$, start with a generic element x of the domain and show that $P(x)$ holds. Note that your proof must apply to every x; it cannot use any properties of particular x's.

You should be able to combine proof techniques above, for example to prove "if and only if" statements or statements with nested quantifiers.

8 Basic Data Structures

You should be familiar with the following basic data structures, how they are represented in memory and the operations listed for each of them. If you want to review them, references to the course textbook are provided.

- Array (1-dimensional or multi-dimensional) [Chapter 10.1.1, 10.1.2]

  - read specified element
  - write specified element
  - iterate across all elements

- Singly- and doubly-linked list (including circular lists) [Chapter 10.2]
  - insert an element
  - remove an element
  - iterate across all elements

- Stack (implemented using an array or linked list) [Chapter 10.1.3]
  - push an element
  - pop an element

- Queue (implemented using an array or linked list) [Chapter 10.1.3]
  - enqueue an element
  - dequeue an element

- Heap implementation of a priority queue (represented using pointers or in an array) [Chapter 6]
  - heapify: build a heap containing a given set of n elements in $O(n)$ time
  - insert a new element
  - remove the element with the minimum/maximum priority (for min heap or max heap, respectively)
  - modify the priority of an element
  - delete an element

- Rooted tree [Chapter 10.3]
  - navigate through tree from parent to child or child to parent
  - perform in-order, pre-order or post-order traversal of tree
  - insert nodes
  - delete nodes

- Binary search tree to implement a set of elements drawn from an ordered universe [Chapter 12]
  - search for a given key
  - insert a new key
  - delete a key

  You should also have seen some technique for keeping the binary search tree *balanced* so that the height of a tree containing n elements is $\Theta(\log n)$, but we shall talk more about this during the course.

- Graph (represented using adjacency matrix or adjacency lists) [Chapter 20.1]

9 Basic Algorithms

You should know the following facts about algorithms.

- There are general-purpose sorting algorithms that sort n items in $O(n \log n)$ time (e.g., merge sort, heap sort or quick sort using a good pivot). [Sections 6.4, 7.1, 7.2]
- *Every* general-purpose (i.e., comparison-based) sorting algorithm takes $\Omega(n \log n)$ time [Section 8.1].
- Counting sort can sort n items drawn from the set $\{1, 2, \ldots, k\}$ in $O(n+k)$ time [Section 8.2].
- It is possible to select the kth smallest element of an unsorted collection of n items in $O(n)$ time [Section 9.2].

We shall look at data structures for the following algorithms, which should already be familiar to you:

- Kruskal's algorithm [Section 21.2]
- Prim-Jarník algorithm [Section 21.2]
- Dijkstra's algorithm [Section 22.3]