CMPS 102: Winter 2017
Brief Section Notes

January 13, 2017

1 First Week Section

1.1 Basic Objects

Sets

Definition 1. A *set* S is a well-defined collection of distinct objects, considered as an object in its own right. The members of the collection are called *elements* of S.

Although clear from the above definition, it's worth to note here two things. First, by well-defined we mean that given a set S and object x, it should be unambiguous whether x is an element of S or not. We use the notation $x \in S$ to denote that x belongs to S and $x \notin S$ to denote that x doesn't belong to S. Second, sets can be as abstract as you want, as long as they remain well-defined. Thus a set can contain numbers, symbols, letters, people and even other sets.

We represent a set just by listing all its elements between curled braces. Thus the set $\{2, 3, 5, 7\}$ is precisely containing the elements $2, 3, 5$ and 7, so for example $2 \in \{2, 3, 5, 7\}$ but $9 \notin \{2, 3, 5, 7\}$. The elements of a set are not supposed to be written in a particular order or to be present only once in a representation. That is $\{3, 3, 3, 3, 5, 7, 2\}$ is exactly the same set with $\{2, 3, 5, 7\}$.

Sequences and Tuples

Definition 2. A *sequence* of objects is a list of these objects in some order.

We usually designate a sequence by writing the list within parentheses. For example the sequence $6, 2, 3$ is denoted as $(6, 2, 3)$. A big difference from sets is that the order and the repetition of the elements in a sequence matters. Hence $(6, 2, 3)$ is not the same with $(2, 3, 6)$ and both are different from $(6, 6, 2, 3)$.

For two sequences A and B, we say that A is a *subsequence* of B if A can be produced by B by deleting some of its objects. For example $(2, 3)$ is a subsequence of $(6, 6, 2, 3)$, while $(3, 6)$ isn't. As in the case of sets, sequences can have infinitely many elements. If a sequence is finite and contains k elements is called *k-tuple*. We can think of k-tuple as a string of length k, the only difference will be in the definition of the *substring*: We will say that the string s is a *substring* of a string l, if s contains some *contiguous* elements of l.

Functions

Definition 3. A *function* f from A to B, is a rule that maps each $a \in A$ to *exactly* one $b \in B$, writing $f(a) = b$. Also we denote f with $f : A \to B$. Set A and B are called the *domain* and *range* of f respectively.

For example, let $A = \{red, green, blue, black\}$ and $B = \{g, r, b\}$ then we define f to map each color to its first letter. That is:

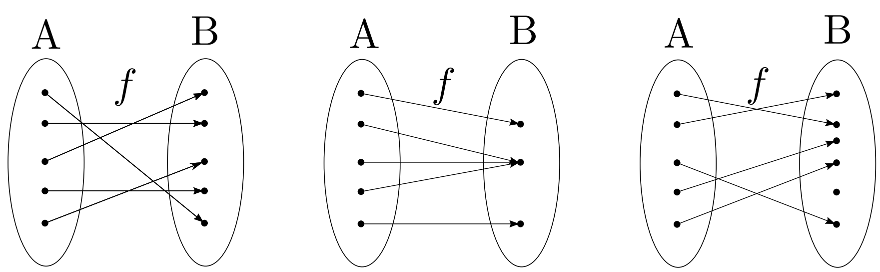
$$f(red) = r, f(green) = g, f(blue) = f(black) = b.$$

A function $f : A \to B$ is called *onto* (or *surjective*), if for every $b \in B$ there is exist at least one $a \in A$ such that $f(a) = b$.

A function $f : A \to B$ is called *one-to-one* (or *injective*), if for every $b \in B$ there is exist at most one $a \in A$ such that $f(a) = b$.

A function $f : A \to B$ is called *one-to-one and onto* (or *bijective*), if for every $b \in B$ there is exist exactly one $a \in A$ such that $f(a) = b$.

For example the function f we defined above is onto but not one-to-one, since f maps both *black* and *blue* to b. In the picture that follows, the first function is one-to-one and onto, the second is onto but not one-to-one and the third is one-to-one but not onto.



Number of subsets of a set

The *power set* of a set A is the set of all subsets of A, and we denote is with $\mathcal{P}(A)$. If A is the set $\{0, 1\}$, then $\mathcal{P}(A) = \{\emptyset, \{0\}, \{1\}, \{0, 1\}\}$. We have the following:

Preposition 1. *For every set S with $|S| = n$, it holds $|\mathcal{P}(S)| = 2^n$.*

Proof. Let $S = \{s_1, s_2, \cdots, s_n\}$. We are going to construct a function f from $\mathcal{P}(S)$ to the set B of all possible binary strings of length n. The fact that $|B| = 2^n$ is easy to be verified, since to fill one such string $b \in B$, we have decide between two choices for everyone of the n positions. Let $L \in \mathcal{P}(S)$, that is L is a subset of S, so we can write $L = \{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$, for some indices $i_1, i_2, \cdots, i_k \in \{1, 2, \cdots, n\}$. We define $f(L)$ to be the binary string having 1's on the positions $i_1, i_2, \cdots, i_k$ and 0's everywhere else. For example let $n = 5$ and $L = \{s_2, s_5\}$, then $f(L) = 01001$. We now show that our function is both one-to-one and onto:

- **f is one-to-one:** Let $L, L' \in \mathcal{P}(S)$ with $f(L) = f(L')$ we will show $L = L'$. As we it was defined the existence of a 1 on the i-th place of the string $f(L)$ implies $s_i \in L$. As $f(L) = f(L')$, every place with 1 in $f(L)$ is also a place of 1 in $f(L')$, and vice versa. That is $L \subseteq L'$ and $L' \subseteq L$ which gives $L = L'$.

- f **is onto:** Let $l \in B$. We will show that there is exist a set $L \subseteq S$ with $f(L) = l$. Let $i_1, i_2, \cdots, i_k$ denote all the positions where l has 1's, then for the set $L = \{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$ we will have $f(L) = l$.

$\square$

Permutations, Combinations, Binomial coefficient

Consider the following enumeration problem:

Given a set S, where $|S| = n$, find the the number of the elements of the set, call it P, of all possible k-tuples with elements from S, containing each element of S at most once. Each element of P is also called a *permutation of k elements of S*(or *k-permutations of S*). When $k = n$ we simply call them permutations of n elements. It is easy to obtain the following:

Preposition 2. *Given a set S, with $|S| = n$, the number of all possible permutations of k elements of S is $n!/(n-k)! = n(n-1)(n-2)\cdots(n-k+1)$*

For example, let $S = \{\spadesuit, \diamondsuit, \clubsuit, \heartsuit\}$, Then the set of all 2-permutations of S is:

$$\{(\spadesuit, \diamondsuit), (\diamondsuit, \spadesuit), (\spadesuit, \clubsuit), (\clubsuit, \spadesuit), (\spadesuit, \heartsuit), (\heartsuit, \spadesuit), (\diamondsuit, \clubsuit), (\clubsuit, \diamondsuit), (\diamondsuit, \heartsuit), (\heartsuit, \diamondsuit), (\clubsuit, \heartsuit), (\heartsuit, \clubsuit)\}$$

which contains $4 \times 3 = 12$ elements.

Let's now consider the above problem when we don;t care about order:

Given a set S, where $|S| = n$, find the number of elements of the set, call it C, of all subsets of S with k elements. Each element of C is also called a *combination of k elements of S*(or *k-combination of S*). The number of combinations of k elements out of n arise in many different situations in mathematics, therefore it has a symbol:

Definition 4. We define the *binomial coefficient* $\binom{n}{k}$ to be equal to the number of combinations of k elements of a set with n elements. [That is $\binom{n}{k} := |C|$ with the above notation.]

We are now going to find a simple formula for the binomial coefficients. To do that we will make use of the above result about permutations. The critical idea is that a single k-combination of S produces $k!$ distinct k-permutations of S. Specifically, let $S = \{s_1, s_2, \cdots, s_n\}$ and $c = \{s_{i_1}, s_{i_2}, \cdots, s_{i_k}\}$ a k-combination of S. Selecting a certain order for c and write the result in parenthesis, will give us a valid k-permutation of S. If we apply the above to every element of C we observe the following facts:

$$\textit{Every } c \in C \textit{ produces exactly } k! \textit{ permutations of } k \textit{ elements of } S \tag{1}$$

This is easy to see since, as we show above, there are precisely $k!$ to ways permute a set of k objects.

$$\textit{If } c \neq c' \textit{ then no permutation produced by } c \textit{ can be equal to a permutation produced by } c'. \tag{2}$$

The fact that c and c' are different means that there is exist an element s in c, which is not in c'. Thus every permutation produced by c contains s, while every permutation produced by c' don't.

For every permutation of k elements of S, there is exist a combination c that produces it. (3)

All needed is to just ignore the particular order of the permutation to get c.

Consider now again the set P of permutations of k elements of S we introduced before. We can partition P with the following rule: tow permutations p and p' will be in the same partition iff they have been produced by the same combination c according to the above procedure. By (1), (2), (3) it easy to see that the above is actually a partition where each part contains exactly $k!$ elements, and the number of parts is equal to $|C|$ (why?).

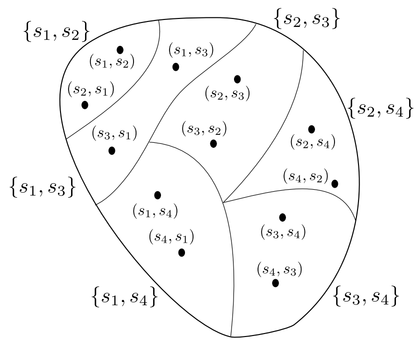


Figure 1: The set of all 2-permutations of $S = \{s_1, s_2, s_3, s_4\}$ partitioned with the above procedure.

Having this picture in mind let's count again the elements of P: we have $\binom{n}{k}$ parts, each containing $k!$ elements, thus:

$$|P| = \binom{n}{k} \times k! \Rightarrow \boxed{\binom{n}{k} = \frac{n(n-1)(n-2)\cdots(n-k+1)}{k!}}.$$

where in the last equality we used preposition 2.

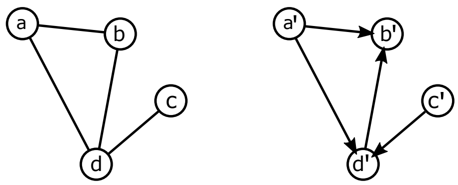
Graphs

We will mainly use the following two types of graphs:

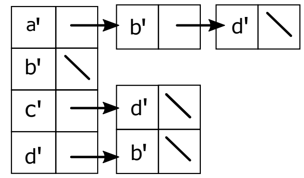
Definition 5 (Undirected Graph)**.** A *undirected graph* is a pair $G = (V, E)$ where V is a set of *vertices* or *nodes* and E is a set *edges* having as elements subsets of V with two elements.

Definition 6 (Directed Graph)**.** A *directed graph* is a pair $G = (V, E)$ where V is a set of *vertices* or *nodes* and E is a set *edges* having as elements pairs of V.

It is a common notation to use n, m for the number nodes and edges, respectively. We use two representations of graphs namely the *adjacency matrix* and the *adjacency list*. While the adjacency matrix needs constant time to verify the presence of a particular edge it takes $O(n^2)$ space. On the other hand, an adjacency list needs $O(n + m)$ space, but a query for the presence of a certain edge takes $O(n)$ time. In the next figure we give an example of an undirected and directed graph and their representation with adjacency matrix and adjacency list, respectively.



	a	b	c	d
a	0	1	0	1
b	1	0	0	1
c	0	0	0	1
d	1	1	1	0



We will call a graph *dense* if $|E| = m = O(n^2)$, and *sparse* if $|E| = m = O(n)$.

1.2 Asymptotic Notation

Big O, Ω, Θ

Definition 7. Let f, g be to functions from $\mathbb{N}$ to $\mathbb{R}^+$. We say that $f(n) \in \Theta(g(n))$ or $f(n) = \Theta(g(n))$ iff there exist constants $c_1, c_2, n_0 > 0$ such that for every $n \geq n_0$, $c_1 g(n) \leq f(n) \leq c_2 g(n)$

The equality $f(n) = \Theta(g(n))$ indicates that $g(n), f(n)$ have the same order. The set $\Theta(g(n))$ is precisely the set of all functions having the same order with $g(n)$. For example, $10^{-23}n^3 + n + 10 = \Theta(n^3)$, while $42n^2 + 5n^3 + 10n^3 \log n = \Theta(n^3 \log n)$

Definition 8. Let f, g be to functions from $\mathbb{N}$ to $\mathbb{R}^+$. We say that $f(n) \in O(g(n))$ or $f(n) = O(g(n))$ iff there exist constants $c, n_0 > 0$ such that for every $n \geq n_0$, $f(n) \leq cg(n)$

The equality $f(n) = O(g(n))$ indicates that $g(n)$ is an upper bound of the order of $f(n)$. The set $O(g(n))$ is precisely the set of all functions having order no higher than that of $g(n)$. For example, $10^{-23}n^3 + n + 10 = O(n^5)$, while $42n^2 + 5n^3 + 10n^3 \log n = O(n^3 \log n)$

Definition 9. Let f, g be to functions from $\mathbb{N}$ to $\mathbb{R}^+$. We say that $f(n) \in \Omega(g(n))$ or $f(n) = \Omega(g(n))$ iff there exist constants $c, n_0 > 0$ such that for every $n \geq n_0$, $f(n) \geq cg(n)$

The equality $f(n) = \Omega(g(n))$ indicates that $g(n)$ is a lower bound of the order of $f(n)$. The set $\Omega(g(n))$ is precisely the set of all functions having order no less than that of $g(n)$. For example, $10^{-23}n^3 + n + 10 \neq \Omega(n^4)$, while $42n^2 + 5n^3 + 10n^3 \log n = \Omega(n^3)$

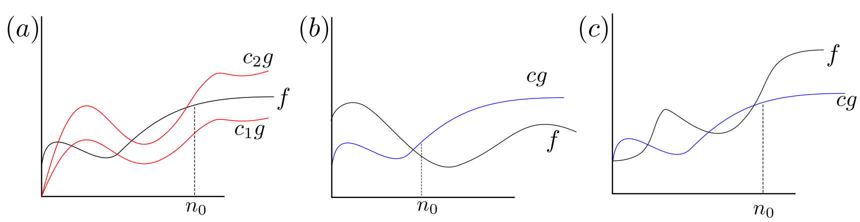


Figure 2: Graphs when (a) $f(n) = \Theta(g(n))$, (b) $f(n) = O(g(n))$ and (c) $f(n) = \Omega(g(n))$.

Sample algorithms, data structures

Binary search

The problem:

Given an array A of n elements with values $A[0]...A[n-1]$, sorted such that $A[0] \leq \cdots \leq A[n-1]$, and target value T, find the index of T in A.

The Algorithm:

1. Set L to 0 and R to $n-1$.
2. If $L > R$, the search terminates as unsuccessful.
3. Set $m = \lfloor (L+R)/2 \rfloor$.
4. If $A[m] < T$, set L to $m+1$ and go to step 2.
5. If $A[m] > T$, set R to $m-1$ and go to step 2.
6. Now $A[m] = T$, the search is done; return m.

Running time: How many times we need to divide by 2, a number n, to get 1? $\rightarrow \log_2(n)$. Thus time $O(\log(n))$

Heap and Dictionary

Two important data structures are the *heap* and the *dictionary*. We will not present here how they can be efficiently implemented, but just the basic operations and their times in an efficient implementation:

Heap	
max()	$\Theta(1)$
deleteMax()	$O(logn)$
Insert()	$O(logn)$
createHeap()	$O(n)$

Dictionary	
lookup()	$\Theta(1)$
insert()	$\Theta(1)$