San Skulrattanakulchai
Gustavus Adolphus College
Oct 8, 2015

Set Theory Review & Recurrences

A *set* is a collection of distinct objects called *elements* or *members*. A set can be specified by enumerating its elements. For example, letting S_0 be the set of fundamental atomic particles, we can write $S_0 = \{\text{neutron, proton, electron}\}$.

The notation $x \in S$ (reads "x belongs to S", or "x is a member of S") says that x is an element in the set S; the notation $x \notin S$ says that x is not an element in the set S. For example, electron $\in S_0$, neutrino $\notin S_0$.

The order that an element appears in the enumeration does not matter. Also, an element may appear more than once inside the enclosing braces. So $\{3, 3, 1, 3\}$ is considered the same set as $\{1, 3\}$.

The *size* or *cardinality* of a set S is the number of elements in S and is denoted $|S|$. For example, $|S_0| = 3$ and $|\{3, 3, 1, 3\}| = 2$.

The *empty set*, denoted $\emptyset$, contains no element, i.e., $|\emptyset| = 0$.

Set T is a *subset* of S, denoted $T \subseteq S$, if every member of T is also a member of S. For example, $\{\text{proton, neutron}\} \subseteq S_0$. Two sets S and T are equal, denoted $S = T$, if $S \subseteq T$ and $T \subseteq S$. Set T is a *proper subset* of S, denoted $T \subset S$, if $T \subseteq S$ and $T \neq S$. For example, $\{\text{proton, neutron}\} \subset S_0$, and $\emptyset \subset S_0$.

Another way of specifying sets is by the *set former* notation.

For example, set S_0 can be denoted $\{x : x \text{ is an elementary atomic particle}\}$, and can be read "the set of all elements x such that x is an elementary atomic particle." In general, we can write any property of x after the colon, and for precision state from what set we are taking the general element x. For example, $\{x \in \mathbb{R} : 1 \leq x \leq 10\}$ is the set of all real numbers between 1 and 10 inclusive, i.e., the closed interval $[1, 10]$, and $\{n : n \text{ is an even integer}\}$ is the set of all even integers.

We can even write a formula in front of the colon. For example,
$\{2n : n \text{ is an integer}\}$ is the set of even integers as above;
$\{d \cdot e : d \text{ and } e \text{ are integers } > 1\}$ is the set of composite numbers;
if $A[1..10]$ is an array, $\{A[j] : 6 \leq j \leq 10\}$ is the set containing the last 5 elements of the array.

We can list more than 1 formula in a set former. For example,
$\{3n + 1, 3n + 2, 0 : n \text{ is an integer}\}$ is the set of integers not divisible by 3, plus 0.

We combine sets with the operations $\cup$, $\cap$, $\setminus$, and set complement $\overline{S}$. For example,
$\{x : x \text{ is an integer divisibleby } 2\} \cap \{y : y \text{ is an integer divisibleby } 3\}$ is
$\{x : x \text{ is an integer divisibleby } 6\}$.

Sets S and T are *disjoint* if they have empty intersection, i.e., $S \cap T = \emptyset$.

Set operations for dynamic programming
In dynamic programming, the important set operations are $\bigcup$, $\bigcap$, min, max, $\sum$, $\prod$, $\bigvee$, and $\bigwedge$.

Examples
$\min\{n : n \text{ is composite }\} = 4$
Let $A[1..10]$ be an array with $A[i] = 2i$. Then
$\min\{A[i] : 5 < i \leq 10\} = 12$.
$\max\{A[i] : 3 \leq i \leq 6\} \cup \{i : 5 < i \leq 15\} = 15$.

An index i achieving the minimum value is called a *minimizer*. Some authors use *arg min* for the minimizer. Similarly for *maximizer* (*arg max*).

Recurrences

A sequence is a function defined on the positive (or nonnegative) integers. For example, the sequence $\langle t_n \rangle_{n=0}^{\infty}$ defined by

$$t_n = 2^n \quad \text{for all integers } n \geq 0$$

is the sequence $\langle 1, 2, 4, 8, \ldots \rangle$.

A sequence may be defined by a formula like t_n above. It can also be defined as a *recurrence* like the following definition for the **Fibonacci Sequence** $\langle f_n \rangle$:

$$f_n = \begin{cases} n & \text{if } i = 0 \text{ or } 1 \\ f_{n-1} + f_{n-2} & \text{if } i > 1. \end{cases}$$

Given nonnegative integer n, we can compute f_n like so:

```
F(n) {
    if n < 2 then return n
    else return F(n − 1) + F(n − 2)
}
```

The problem is that this recursive procedure has an exponential running time; it computes F(k) for those $k < n$ repeatedly. We can avoid repeated computation by using a table $F[\cdot]$ and recording the values of every F(k) the first time we know it. This can be done by computing F(k) for all $k = 0, 1, 2, \ldots$ in that order. This iterative procedure takes $O(n)$ additions.

```
F(n) {
    F ← an empty array of length n
    F[0] ← 0; F[1] ← 1
    for i ← 2 to n do
        F[i] ← F[i − 1] + F[i − 2]
    return F[n]
}
```