Tutorial 06
Generating functions, orderings, selection and ordered sets
CS 240E Winter 2023
University of Waterloo
Monday, February 27, 2023

1. **Generating functions.** Many problems in CS240E have answers represented by a sequence of numbers

$$a_0, a_1, a_2, \ldots$$

The generating function $A(x)$ of $\{a_n\}$:

$$A(x) \coloneqq \sum_{n=0}^{\infty} a_n x^n,$$

is *data structure* for working with $\{a_n\}$ implicitly, e.g. through algebraic or functional equations.

We use generating functions when we want:

- to find an exact formula for a_n,
- to find a recurrence formula for a_n,
- to find averages and other statistical properties of a_n, or
- to find asymptotic behaviour of or approximations to a_n.

The last point is most useful for the problems that we work with.

Suppose a certain sequence $a_0, a_1, \ldots$ satisfies

$$a_{n+1} = 2a_n + 1,$$

for $n \geq 0; a_0 = 0$. Since a_n doubles with every increment in n, we know $a_n \in \Theta(2^n)$. We will use generating functions to show this formally; in fact, we will determine an exact formula for a_n.

(a) Multiply both sides of the recurrence by x^n and sum over all values of n for which the recurrence is valid, to show that

$$\frac{A(x)}{x} = 2A(x) + \frac{1}{1-x}.$$

(b) Using a partial fraction expansion, show that

$$a_n = 2^n - 1.$$

(c) With a similar approach, show the Binet's formula for the n-th Fibonacci number:

$$f_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^n + \left(\frac{1-\sqrt{5}}{2} \right)^n \right].$$

More generally, a sequence $\{a_n\}$ that satisfies linear recurrences of the form

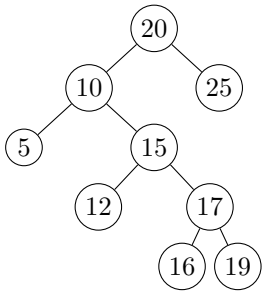
$$c_0 a_n + c_1 a_{n-1} + \cdots + c_r a_{n-r} = 0$$

for all n larger than some integer $N \geq r$, where each $c_0, c_1, \ldots, c_r$ is rational and $c_0 \neq 0$, is called a *C-finite sequence*. It is a remarkable fact established by [de Moivre, 1730] that C-finite sequences have rational generating functions and vice-versa.

2. **Splay trees.** Given the following splay tree S, calculate its potential using the potential function

$$\Phi(i) := \sum_{v \in S} \log n_v^{(i)},$$

where $n_v^{(i)}$ is the number of nodes in the subtree rooted at v after i operations, including v itself. Insert the key 18. Calculate the new potential. Verify that the difference between the potential difference is less than $4 \log n - 2R + 2$, where R is the number of rotations.



3. **Static ordering.** Let A be an unordered array with n distinct items $k_0, ..., k_{n-1}$. Give an asymptotically tight Θ-bound on the expected access cost if you put A in the optimal static order for the followng probability distributions:

   (a) $p_i = \frac{1}{n}$ for $0 \leq i \leq n-1$

   (b) $p_i = \frac{1}{2^{i+1}}$, for $0 \leq i \leq n-2$, $p_{n-1} = 1 - \sum_{i=0}^{n-2} p_i = \frac{1}{2^{n-1}}$

4. **Dynamic orderings.** Consider a linked list with the keys $k_1, k_2, \ldots, k_n$ in that order. Give a sequence of n searches such that the Move-To-Front heuristic uses $O(n)$ comparisons while the Transpose heuristic uses $\Omega(n^2)$ comparisons.

5. **Selection and ordered sets.**[1] The *set* ADT supports the operations:

   - *add*(x) : add x to the set,
   - *exists*(x) : determine whether x is present in the set, and
   - *delete*(x) : delete x from the set.

   We will implement an *ordered* set–whose elements are stored in some order–that will also support the operations:

[1] This problem is harder than what we should expect on assignments or exams. It is used to explore new data structures, develop critical thinking skills, and illustrate the ways of thinking of CS240E.

- *select*(k) : return the k-th element of the set,[2]
- *rank*(x) : determine the index that x would have if it were in the set.

The most often used realization that supports these operations efficiently is a balanced tree that stores the size of each node's subtree (see A3). We will discuss a new realization called *binary indexed tree*, first developing it in the context of sum/update queries on an array.

We are given an array A of n integers. Our goal is to answer queries of the form *sum*(a, b): return the sum

$$\sum_{i=a}^{b} A[i].$$

We will use the following array as our running example. For this problem, we will assume that all arrays are one-indexed to simplify implementation.

	1	2	3	4	5	6	7	8
A:	1	3	4	8	6	1	4	2

Figure 1.

(a) Consider the situation when the array is *static*, i.e. never updated between queries. Explain how to answer *sum* queries in constant time after $O(n)$ preprocessing.

A *binary indexed tree* (also commonly known as Fenwick tree) is a data structure that supports range sum queries *and updates* in logarithmic time; and it is easily built in $\Theta(n \log n)$ time.

Let $p(k)$ denote the largest power of two that divides k. A binary indexed tree is represented as an array, which we will denote by t, such that

$$t[k] = \text{sum}(k - p(k) + 1, k).$$

In other words, at index k we store the sum of the values of the original array A, whose length is $p(k)$ and that ends at index k.

(b) Give the binary indexed tree corresponding to the example array A in Fig. 1.

With a binary indexed tree, any value of $\text{sum}(1, k)$ can be computed in $O(\log n)$ time, because the range $[1, k]$ can always be divided into $O(\log n)$ ranges whose sums are stored in the tree. For example,

$$\text{sum}(1, 7) = \text{sum}(1, 4) + \text{sum}(5, 6) + \text{sum}(7, 7).$$

(c) Suppose we want to update an element of the array. Give an asymptotic bound on the number of values in the tree t that we must update.

[2]The *selection* problem receives as input a set of n items and an integer k with $0 \leq k \leq n - 1$, and it must return the item that would be at $A[k]$ if the items were put into an array A in sorted order.

For this question, we will assume that we can calculate any value of $p(k)$ in constant time. Note that this assumption is not realistic; in practice, we can use bit operations:

$$p(k) = k\& - k.$$

The following functions give implementations of the functions:

- $pref(k)$: return the prefix sum $\sum_{i=1}^{k} A[i]$.

```
pref(k):
    s = 0
    while(k >= 1):
        s += t[k]
        k -= p(k)
    return s
```

- $change(k, x)$: change the array value at position k by x (x could be positive or negative).

```
change(k, x):
    A[k] += x
    while(k <= n):
        tree[k] += x
        k += p(k)
```

(d) Give tight asymptotic bounds on the time complexity and auxiliary space of these algorithms.

We are now ready to return to our ordered set operations.

(e) Explain how to use a binary indexed tree to support all ordered set operations except for *selection* in time logarithmic in n, assuming that all numbers are integers in the range $[1, N]$, where N is independent of n.
Hint: suppose the index of an element x in A is x itself, and that $A[x]$ is the number of elements in A equal to x.

(f) Give an algorithm to support $select(k)$ in $O(\log n)$ time.
Hint: one approach is to use binary lifting.

The binary indexed tree data structure is known to perform very well in practice, and is very quick to implement (unlike AVL-tree for example). The major drawback of the binary index tree is its auxiliary space use. Our approach uses $\Theta(N)$ auxiliary space, which is very significant. Some possible solutions are *coordinate compression* and *hashing*, which are topics we will discuss in future tutorials and lectures.