Hashing

1 Hash Tables

Given a universe U of objects, hash tables maintain an evolving set $S \subset U$ by keeping n buckets, where $n \approx |S|$. To map objects to buckets, they use a hash function $H : U \rightarrow 0, 1, 2, \ldots, n-1$ and an array of length n, storing an object x in `A[h(x)]`.

1.1 Two-Sum Problem

Input: Array of integers `A[1\ldots n]`

Output: All pairs (a, b) such that $a + b = t$.

Reducing the problem to sorting yields an $O(n \log n)$ solution - sort the array, then binary search for $t - x$ for each x. Using a hash table yields an $O(n)$ solution.

2 Hashing and Pathological Data Sets

The **load** of a hash table is defined as

$$\alpha = \frac{n}{|S|} \tag{1}$$

$\alpha = O(1)$ is a necessary condition for constant-time hash table operations, and $\alpha << 1$ is necessary for open addressing when using a sequential hash function to deal with collisions. For good performance and efficiency with linked-list buckets, keep α close to 1.

A **pathological data set** is a data set that isolates one or very few buckets in a hash table to reduce it to linear efficiency, and exists for every hash function. Pathological data sets can paralyze real-world systems, and there are usually two ways to prevent this exploit:

1. Crytographic hashes are much more immune (practically completely) to pathological data sets (SHA-2, MD5, etc)

2. Hash randomization: pick a single hash function h in a family H of hash functions at runtime.