CS120: Intro. to Algorithms and their Limitations Hesterberg & Vadhan

Lecture 9: Dictionaries and Graphs

Harvard SEAS - Fall 2022 *Sept. 29, 2022*

1 Announcements

- Add/drop 10-03
- Midterm in class 10-04. Arrive by 9:45 am. Midterm grades will be given by percentage points, rather than by N/L/R grading.
- Midterm scope includes material on randomized algorithms (last time) and randomized data structures (today) — the main takeaways (e.g. Monte Carlo vs. Las Vegas, when to use each), not proofs.
- This week's sections will be review for midterm!
- Fill out ps3 feedback form: https://tinyurl.com/cs120ps3feedback

2 Randomized Data Structures

Recommended Reading:

- CLRS 11.0–11.4
- Roughgarden II 12.0–12.4

We can also allow data structures to be randomized, by allowing the algorithms Preprocess, EvalQ, and EvalU to be randomized algorithms, and again the data structures can either be Las Vegas (never make an error, but have random runtimes) or Monte Carlo (have fixed runtime, but can err with small probability).

A canonical data structure problem where randomization is useful is the *dictionary* problem. These are data structures for storing sets of key-value pairs (like we've been studying) but where we are *not* interested in the ordering of the keys (so min/max/next-smaller/selection aren't relevant).

Updates :	Insert or delete a key-value pair (K, V) with $K \in \mathbb{N}$ into the multiset
Queries :	Given a key K, return a matching key-value pair (K, V) from the multiset (if one exists)

Data-Structure Problem(Dynamic) Dictionaries

Of course the Dynamic Dictionary Problem is easier than the Predecessor Problem we have already studied, so we can use Balanced BSTs to perform all operations in time $O(\log n)$. So our goal here will be to do even better — get time $O(1)$.

Let's assume our keys come from a finite universe U. Last time we saw that we can get $O(1)$ time updates and queries as follows:

A deterministic data structure:

- Preprocess(U): Initialize an array A of size U.
- Insert(K, V): Place (K, V) into a linked list at slot $A[K]$.
- Delete(K): Remove the head of the linked list at slot $A[K]$.
- Search(K): Return the head of the linked list at $A[K]$.

A problem with this approach:

Attempted fix 1: Use an array A of size $m \ll U$, and put (K, V) at spot $A[K\%m]$.

Problem:

Attempted fix 2: Choose m as above, but choose some other function $h : [U] \to [m]$ as a replacement for %, and put a key K at $A[h(K)]$.
Problem: 5

Attempted fix 3: Choose a *random* function $h : [U] \to [m]$. Then the error probability is small even in the worst case.
Problem:

An actual fix: Choose a *random hash function* $h : [U] \to [m]$. For the purposes of CS 120, a *random hash function* is the same as a completely random function, except that:

Constructing and analyzing random hash functions is outside the scope of CS 120, but there is some optional reading on it in the detailed lecture notes in case you are curious.

A Monte Carlo data structure:

A Las Vegas data structure ("Hash Table"):

3 Storing and Search Synthesis

We have seen several approaches to storing and searching in large datasets (of key-value pairs):

1. Sort the dataset and store the sorted array

2. Store in a binary search tree (balanced and appropriately augmented)

3. Store in a hash table

4. Run Randomized QuickSelect

For each of these approaches, describe a feature or combination of features it has that none of the other approaches provide.

4 Graph Algorithms

Recommended Reading:

- Roughgarden II Sec 7.0.–7.3, 8.0–8.1.1
- CLRS Appendix B.4

Motivating Problem: Google Maps. Given a road network, a starting point, and a destination, how can/should I travel to get from the starting point to the destination?

Q: How to model a road network?

5 Shortest Walks

Motivated by a (simplified version) of the Google Maps problem, we wish to design an algorithm for the following computational problem:

Input	**:** A digraph $G = (V, E)$ and two vertices $s, t \in V$
Output	**:** A *shortest walk* from s to t in G, if any walk from s to t exists

Computational Problem ShortestWalk

Definition 5.1. Let $G = (V, E)$ be a directed graph, and $s, t \in V$.

- A *walk* w from s to t in G is
- The *length* of a walk w is
- The *distance* from s to t in G is
- A *shortest walk* from s to t in G is

Q: An algorithm immediate from the definition?
A:

Lemma 5.2. *If w is a shortest walk from s to t, then all of the vertices that occur on w are distinct. That is, every shortest walk is a* path *— a walk in which all vertices are distinct.*

Proof.

□

Q: With this lemma, what is the runtime of exhaustive search?
A:

Q: How can we get a faster algorithm?
A: