Advanced Data Structures Jan-Apr 2012

Lecture 7 - February 9, 2012

Lecturer: Venkatesh Raman *Scribe: Ashutosh Rai*

1 Overview

In the last lecture we talked about a geometric view of Binary Search Trees and saw how the algorithmic problem of coming up with efficient Binary Search Trees corresponds to a combinatorial problem.

In this lecture, we will try to look at Hashing techniques which will help us in supporting dictionary operations in $O(1)$ time.

2 Hash Functions

We know that balanced binary search trees can support dictionary operations in $O(\log n)$ time, and we can't do much better if we use the comparability model. But if we have some extra information about the input, for example, if we know beforehand the range (the universe) from which the data is coming, then can we do better? It seems that we can. For example, if we know that the universe is $U = \{0, 1, 2, 3, ..., u-1\}$, then we can have an array A of size u and then we can support each of the dictionary operations in $O(1)$ time by mapping each element i to $A[i]$.

But the problem with this approach is that it takes space linear in the size of the universe, even is the number of elements stored at any stage is very small. For addressing this, we define hash functions.

2.1 Collisions

Let the universe U be $\{0, 1, 2, 3, ..., u-1\}$, and let T be $\{0, 1, 2, 3, ..., t-1\}$. We'll call T a *table*. Now, a hash function h is a function from U to T. Let $S \subseteq U$ be the set of elements present at any time in T. Let $|S| = n$.

Clearly, if $t < n$, for any hash function h, there exist $x, y \in S$ such that $h(x) = h(y)$. We call this a *collision*. In case of collisions, we maintain a linked list for each location in T. Now, it is easy to see that for every hash function h, there exists $S \subseteq U$ such that the number of collisions is $min(n, u/t)$, which will result in bad worst case cost. To solve this, we make use of randomness.

2.2 Randomized hash Functions

Here, we choose a function h uniformly at random from set of all functions from U to T. We use the function h to map the set S to T and perform the dictionary operations with it. In case of collisions, we use the concept of linked lists as mentioned earlier. We are interested in the expected size of the linked list, since that is what determines the cost. Now, Let $x \in S$. Clearly, the size of the linked list at $h(x)$ is $|h^{-1}(h(x))|$. So, the expected length of the linked list at $h(x)$

$$E[|h^{-1}(h(x))|] = 1 + \sum_{y \in S, y \neq x} [h(y) = h(x)]$$

Now, we know that choosing a function uniformly at random from U to T is equivalent to choosing uniformly at random an element of T for each element of U. Hence, for each y, such that $y \neq x$, the probability that $h(y) = i$ is $1/t$. In particular, the probability that $h(y) = h(x)$ is also $1/t$. So, we get

$$E[|h^{-1}(h(x))|] = 1 + \sum_{y \in S, y \neq x} 1/t$$

$$= 1 + n/t$$

Now, if we choose t to be $O(n)$, then we see that the expected size of the linked list for each location in T becomes $O(1)$, and hence the dictionary operations can be performed in $O(1)$ expected time.

2.3 Randomness as resource

Pure randomness is hard to obtain so one should try to decrease the number of random bits used. Now, for selecting a random function from U to T, the size of the set is t^u and hence the number of random bits needed is $(u \log t)$. But we observe that we didn't make use of the complete independence. Instead we looked at two elements from S and said that the probability that they are going to be mapped to the same location in T is small. So, we make use of this fact, and define 2-Universal Hash Families as follows-

Definition 1 (2-Universal hash Family). *A family $\mathcal{F}$ of functions from U to T is 2-Universal if, for a uniformly chosen h from $\mathcal{F}$*

$$\forall x, y \in U, x \neq y \Rightarrow Prob[h(x) = h(y)] \leq 1/t$$

Clearly, such family exists (the set of all functions being such a family), but we are interested in families which are small. One such family is-

$\mathcal{F} = \{h_{a,b} : a, b \in \mathbb{Z}_p,\ a \neq 0,\ p \geq |U|\}$ where $h_{a,b}(x) = ((ax + b) \mod p) \mod t$

It is easy to verify that the above family is 2-Universal. Now, we look at the size of the hash family. We see that a hash function is uniquely determined by a pair (a, b) such that $a, b < p$. Clearly, size of the hash family is p^2, so we need $2 \log p$ (which is $O(\log u)$) random bits to select a function from this hash family, which is a significant reduction from $O(u \log t)$.

2.4 Use of small Hash Families in derandomization

The **Max-Cut** problem is known to be NP-complete, but approximation algorithms have been known with constant factor approximation. One such randomized algorithm just partitions the vertex set randomly into two parts (by assigning each of the vertices 0 or 1 u.a.r. and then putting all vertices with same assignment in one set). So, for this algorithm-
Pr[A particular edge (i, j) goes across] $= 1/2$

So, by linearity of expectations, expected number of edges going across the cut
E[Number of edges going across] $= m/2 \leq OPT/2$

So we achieve an expected constant factor approximation algorithm. We again observe that we just need pairwise independence, so a 2-Universal family from V to $\{0, 1\}$ does the job. Now, since the expected number of edges going across is $m/2$, there must exist a function in $\mathcal{F}$ which achieves it. So, for derandomizing, we apply each of the functions in $\mathcal{F}$ one by one and take the one with the maximum cut size. We know that such a family exists with size $O(|V|^2)$, hence the given randomized algorithm can be derandomized in time $O(|V|^2)$ and the constant factor approximation can be achieved deterministically.

3 FKS Algorithm

We've seen that the randomized hash functions can be used for getting expected constant time to perform dictionary operations. But in some cases, we like to have worst case constant time to perform such operations.

Let U be the universe, and let $S \subseteq U$, $|S| = n$ be the set from which the dictionary operations are going to get performed. FKS algorithm performs membership in $O(1)$ worst case time and uses $O(n)$ space. It does so by giving a function h (depending on S) from U to T where $|T| = O(n)$, such that h is injecive on S.

To prove that, we first prove the following lemma-

Lemma 2. *Let h is chosen uniformly at random from $\mathcal{F}$ which is a 2-Universal family. Then, expected number of collisions of h on S is at most $n(n-1)/2t$.*

Proof. Clearly, E[No. of collisions of h on$S$] $= E[|\{(x, y) : x, y \in \text{S}, x \neq y, h(x) = h(y)\}|]$

$$\leq \binom{n}{2}/t$$

$$= n(n-1)/2t$$

□

Corollary 3. *If $t = n^2$, there exists a function $h \in \mathcal{F}$ such that h is injective on S.*

Proof. In that case, clearly, the expected number of collisions is < 1 so there must exist a function which is injective. □

Corollary 4. *If $t = n$, there exists a function $h \in \mathcal{F}$ such that*

$$\sum_{0 \le i < n} b_i^2 \le 2n - 1$$

where $b_i = h^{-1}(i) \cap S$

Proof. From the lemma, the expected number of collisions is this case is at most $(n-1)/2$. Also, we observe that if there is a bucket (linked list) of size b_i, then $\binom{b_i}{2}$ collisions are happening there. Hence, the expected number of collisions,

$$E[\#collisions] = \sum_{0 \le i < n} \binom{b_i}{2}$$

Hence, we get,

$$\sum_{0 \le i < n} \binom{b_i}{2} \le (n-1)/2$$

$$\sum_{0 \le i < n} b_i^2 - b_i \le n - 1$$

$$\sum_{0 \le i < n} b_i^2 \le 2n - 1$$

Hence, the function which has $(n-1)/2$ (the expected number) or less collisions achieves the above. □

Now, let h' be the function obtained by applying Corollary 4 on S, and h_i's be functions in $\mathcal{F}$ which are injective from $B_i = (h')^{-1}(i) \cap S$ to $\{0, 2, ..., |B_i|^2 - 1\}$ given by Corollary 3. We define the desired function h as-

$$h(x) = \sum_{0 \le l < i} b_l^2 + h_i(x)$$

where $i = h'(x)$.

It is easy to see that h is injective since each of the h_i's is injective.

To implement it, we take a table T and store h' at index 0, h_0 at index 1, h_1 at index 2 and so on till h_{n-1}, then we store the values of $\sum_{0 \le j < i} b_j^2$'s on the next n entries $(0 \le i < n)$. Lastly, we keep b_i^2 spaces for the elements in $(h')^{-1}(i)$ for each $0 \le i < n$. (Diagram required)

For testing membership of $x \in S$, we first evaluate $h'(x)$. Let $h'(x) = i$. Then we go to the entry corresponding to h_i and evaluate $h_i(x)$. After this, we just add up the places we need to skip (summation of b_i's, which is already stored) and then we add the value of $h_i(x)$ to get to the desired place. Asuuming that the hash function evaluation takes $O(1)$ time, the whole process can be done in $O(1)$ time (after we have done the preprocessing, ie, we have arrived at the right set of hash functions).

To analyze the space complexity, we see that for storing h_i's and summation of b_i's we need $2n$ cells. Then we need $\sum_{0 \leq i < n} b_i^2$ cells to store the data, which is at most $2n - 1$ by corollary 4. Also, we need a cell to store $h^{'}$. Hence, the total number of cells needed is at most $2n + 2n - 1 + 1$ or $4n$ which is $O(n)$. This concludes the description of the FKS algorithm, modulo the time required for preprocessing, which will be discussed in next lecture.

4 Space Reduction for FKS

We know that h' and all the h_i's come from a 2-Universal Hash Family, and as said earlier, such a family of size $O(u^n)$ exists, which is given by a prime $p = O(u)$. So, each of the functions take $2 \log u$ bits and storing $(n + 1)$ functions take $O(n \log u)$ bits. Also, we see that each of the $b_i^2 \leq n^2$, so storing n such b_i's take $O(n \log n)$ bits. Also, the numbers in the table take $(\log u)$ bits each. So, the total space requirement for the table is $O(n \log u + n \log n)$.

4.1 Universe Reduction

In general, the universe can be huge, so we don't want a large dependence on u. For that, we define the following-

Definition 5. *Let* $S = \{x_1, x_2, ..., x_n\}$

$$N = \prod_{i \neq j} (x_i - x_j)$$

Clearly, $N < u^{\binom{n}{2}}$ or $N < u^{n^2}$.

Claim 6. *There exists a prime* p *such that* $p \leq \log N$ *and* p *does not divide* N.

The claim can be proved using number theoretic thechniques. Let p be such a prime. We define $S' = \{x_1 \mod p, x_2 \mod p, ..., x_n \mod p\}$. We see that all the elements in S' are distinct, since p does not divide any of $x_i - x_j)$ where $x_i, x_j \in S$.

Now, we use the mapping $f(x) = (x \mod p)$ and get $U' = f(U)$. Clearly, $p = O(n^2 \log u)$ and $x < p, \forall x \in U'$. We take U' as the new universe and perform FKS. For storing f, we just store the prime, which takes $(\log n + \log \log u)$ bits. So, we get the space requirements as $O(n \log(n^2 \log u) + n \log n + \log n + \log \log u)$ which is $O(n \log n + n \log \log u)$ bits.

We try to reduce the universe even further, by applying another hash function, say h_0 which maps U' to $\{0, 1, 2, ..., n^2\}$ with no collisions on S (guaranteed by Corollary 3). Again, such a function will take $(2 \log |U'|)$ bits which is $O(\log n + \log \log u)$ bits. Then we perform the FKS algorithm taking the universe to be $U'' = h_0(S')$ such that $|U''| = n^2$. Now, we need to store the prime p (for function f) whic takes space $(\log n + \log \log u)$, the function h_0 which takes $O(\log n + \log \log u)$ and

h', h_i's which take space $O(n \log n)$. So, all the information about hash functions can be stored in space $O(n \log n + \log \log u)$, and the table having indices and the actual elements also takes $O(n \log n)$ space. So, the total space needed is $O(\log \log u + n \log n)$.

4.2 Lower Bounds and Further Work

Schmidt and Siegel [2] proved that any hash function supporting $O(1)$ worst case membership operation must take $(n + \log \log u)$ bits. In the same paper, they also give hash functions which take $O(n + \log \log u)$ space and hence are tight to within a constant factor of the lower bound.

References

[1] M. Fredman, J. Komlós, E. Szemerédi, *Storing a Sparse Table with $O(1)$ Worst Case Access Time*, Journal of the ACM, 31(3):538-544, 1984.