Hashing

Pawan Kumar Aurora

Department of Computer Science and Engineering
Indian Institute of Technology, Kanpur

Comprehensive Exam

Dictionary Problem

- Maintain a set S of size s efficiently under Insertion and Deletion of elements(*keys*) from a universe M of size m
- Facilitate efficient processing of FIND queries

Static Dictionary

- Set S given in advance
- Objective: Organize S such that FIND query operations are efficient

Dynamic Dictionary

- Set S not given in advance
- Constructed by INSERT and DELETE operations
- FIND queries intermingled with INSERT, DELETE
- Objective: Efficient Update and Search operations

Solution to the Dictionary Problem

- Use Balanced Search Trees
- Worst-case complexity $\Omega(\log s)$ for each update/search operation
- Matches lower bound, hence optimal
- Lower bound applicable **only** to comparison based methods

Hash Table

- An array T supporting random access
- Uses key as index to the table
- If $|T| = |M|$, each operation takes $O(1)$ time
- Challenge is to reduce the size of the table to $O(s)$ and still manage O(1) time

Hashing

- Uses a fingerprint function h to determine where a key should be located in the table
- $h : M \rightarrow N$ with $|M| > |N|$
- But collisions should be avoided
- Hence we desire a Perfect Hash Function

Perfect Hash Function

- **Definition:** A hash function $h : M \rightarrow N$ is said to be *perfect* for a set $S \subseteq M$ if h does not cause any collisions among the keys of the set S
- No single hash function is perfect for every $S \subseteq M$
- Hence not good for the Dynamic Dictionary problem
- However, can prove useful for the Static Dictionary problem

Hashing with $O(1)$ Search Time [1, 8.5]

- Set S is given in advance
- Interested in linear space, bounded search cost and a polynomially bounded preprocessing cost
- Hash Table size should be $O(s)$
- Use a hash function h that is perfect for S
- h cannot be perfect for every possible set S

Perfect Hash Family

- **Definition:** A family of hash functions $H = \{h : M \rightarrow N\}$ is said to be a *perfect hash family* if for each set $S \subseteq M$ of size $s < n$ there exists a hash function $h \in H$ that is perfect for S
- Family of all possible functions from M to T is a perfect hash family

Solution to the Static Dictionary Problem

- Solve the Static Dictionary problem by finding a $h \in H$ perfect for S
- Store each key $x \in S$ at the location $T[h(x)]$
- Examine $T[h(q)]$ for search query for a key q
- Preprocessing Cost: cost of identifying h for a specific choice of S
- Search Cost: time required to evaluate h

Constraints

- Description of h stored in T along with elements of S
- For $|H| = r$ the space required is $\Omega(\log r)$ bits
- Search time desired is $O(1)$
- Hence cannot afford more than $O(1)$ table locations for the hash function description
- Each table cell used to encode at most $\log m$ bits of information, where $m = |M|$
- Thus size of the hash family is constrained by $|H| = m^{O(1)}$
- Evaluation of h should be efficient on arbitrary keys

Size of a Perfect Hash Family

- **Claim:** Given $n = s$, any perfect hash family H must have size $2^{\Omega(s)}$
- **Proof:**
  1. Out of $\binom{m}{n}$ possible sets, $\prod_{i=1}^{n} a_i$ have a common perfect hash function $h \in H$. Here a_i for $1 \leq i \leq n$ is the number of elements that h maps to i
  2. Since $\sum a_i = m$, $\prod_{i=1}^{n} a_i \leq \left(\frac{m}{n}\right)^n$
  3. Thus the size of the perfect hash family is $\geq (n/m)^n \binom{m}{n}$
- For $|H| = m^{O(1)}$, by the above claim $m = 2^{\Omega(s)}$ or $s = O(\log m)$
- $m = 2^{32}$ gives $s = 32$ which is not practical

Double Hashing

- The first hash function partitions S into bins of maximum size $O(\log m)$
- Hash function selected from a family of size $|H| \leq m$
- Each bin stores the description of a hash function that is perfect for the keys hashing to that bin
- The keys get stored in the secondary hash tables
- Query time is clearly $O(1)$

b-perfect Hash Family

- **Definition:** Let $S \subset M$ and $h : M \to N$. For each table location $0 \leq i \leq n-1$, we define the bin

$$B_i(h, S) = \{x \in S \mid h(x) = i\}.$$

  The size of a bin is denoted by $b_i(h, S) = |B_i(h, S)|$.
- **Definition:** A hash function h is *b-perfect* for S if $b_i(h, S) \leq b$, for each i. A family of hash functions $H = \{h : M \to N\}$ is said to be a *b-perfect hash family* if for each $S \subseteq M$ of size s there exists a hash function $h \in H$ that is b-perfect for S

b-perfect Hash Family

- **Claim:** There exists an $O(\log n)$-perfect hash family with $|H| \leq m$, for any $m \geq n$
- **Proof:**
  1. Consider $n = s$ for simplicity
  2. Use result: n balls randomly assigned to n bins, with probability $\geq 1 - 1/n$, no bin has more than $(e \ln n) / \ln \ln n$ balls in it
  3. A truly random hash function h behaves similarly and is thus $O(\log n)$-perfect w.h.p.
  4. h is not $O(\log n)$-perfect with probability $\leq \frac{1}{n}$
  5. $\Pr\{$For some S there is no $h \in H$ that is $O(\log n)$-perfect$\}$ $\leq \binom{m}{n} \left(\frac{1}{n}\right)^{|H|} < \binom{m}{n} \left(\frac{1}{n}\right)^{m} < 1$
  6. With non-zero probability, for every S there is some $h \in H$ that is $O(\log n)$-perfect

Drawbacks

- Space complexity is $\Omega(s \log m)$
- Existence of hash families shown via probabilistic methods
- No efficient construction of the hash families is known

The idea of double hashing still looks promising

A Perfect Hash Family

- **Claim:** A perfect hash family $H = \{h : M \rightarrow R\}$ with $|H| \leq m$ exists for all $m \geq s$, provided that $|R| = \Omega\left(s^2\right)$
- **Proof:**
  1. Let H be a 2-universal hash family, also let $|R| = r$
  2. $O(\log m)$ bits suffice to store the hash function description
  3. Also for any h chosen u.r. from H, distinct $x, y \in M$, $Pr\{h(x) = h(y)\} \leq \frac{1}{r}$
  4. Expected number of colliding pairs C is given as follows:

$$E[C] = \sum_{x \neq y \in S} Pr\{h(x) = h(y)\} = \binom{s}{2} \cdot \frac{1}{r} \qquad (1)$$

  5. For $r \geq s^2$, (1) gives $E[C] \leq \frac{1}{2}$
  6. From Markov's inequality, $Pr\{C \geq 1\} \leq \frac{1}{2}$
  7. Thus $Pr\{h \text{ is perfect}\} \geq \frac{1}{2}$

Final Solution

- Primary table of size $n = s$
- Primary hash function h that ensures small bin sizes
- Secondary tables of size quadratic in the bin sizes to ensure perfect hashing
- Secondary hash functions chosen from the perfect hash family H
- Space required: $s + O\left(\sum_{i=0}^{s-1} b_i^2\right)$
- A Search operation clearly takes $O(1)$ time

Hash Functions

- Our goal now remains to find:
  1. A primary hash function which ensures that $\sum_{i=0}^{s-1} b_i^2$ is linear
  2. Perfect hash functions for the secondary tables, which use at most quadratic space

Hash Functions

- **Definition:** Consider any $V \subseteq M$ with $|V| = v$, and let $R = \{0, ..., r-1\}$ with $r \geq v$. For $1 \leq k \leq p-1$, define the function $h_k : M \to R$ as follows,

$$h_k(x) = (kx \, mod \, p) \, mod \, r.$$

Here $p = m+1$ is a prime. For each $i \in R$, the bins corresponding to the keys colliding at i are denoted as

$$B_i(k, r, V) = \{x \in V \mid h_k(x) = i\}.$$

and their sizes are denoted by $b_i(k, r, V) = |B_i(k, r, V)|$.

- Hash functions h_k are completely determined by k
- Description fits in a single table cell

Why h_k?

- **Claim:** For all $V \subseteq M$ of size v, and all $r \geq v$,

$$\sum_{k=1}^{p-1} \sum_{i=0}^{r-1} \binom{b_i(k, r, V)}{2} < \frac{(p-1)\, v^2}{r} = \frac{mv^2}{r}.$$

- **Proof:**
  1. L.H.S. = number of tuples $(k, \{x, y\})$ with $x, y \in V$ and $x \neq y$ such that $((kx \mathit{mod}\, p) \mathit{mod}\, r) = ((ky \mathit{mod}\, p) \mathit{mod}\, r)$
  2. For a given pair $x, y \in V$ with $x \neq y$,

     $$k(x-y) \mathit{mod}\, p \in \{\pm r, \pm 2r, \pm 3r, ..., \pm \lfloor (p-1)/r \rfloor r\}.$$

  3. For any fixed value of $x - y$ the following equation has a unique solution for k for any j

     $$k(x-y) \mathit{mod}\, p = jr$$

  4. Implies L.H.S. at most $\binom{v}{2}\frac{2(p-1)}{r} < \frac{(p-1)v^2}{r}$

Finally

- $\forall V \subseteq M$ of size v, and all $r \geq v, \exists k \in \{1, ..., m\}$ s.t.

$$\sum_{i=0}^{r-1} \binom{b_i(k, r, V)}{2} < \frac{v^2}{r}. \tag{2}$$

- **Claim:** For any $S \subseteq M$ with $|S| = s$ and $m \geq s$, there exists a hash table representation of S that uses space $O(s)$ and permits the processing of a FIND operation in $O(1)$ time.
- **Proof:**
  1. For $v = r = s$ from (2), $\exists k \in \{1, ..., m\}$ s.t.

$$\sum_{i=0}^{s-1} b_i(k, s, S)^2 < 3s.$$

  2. For $v = s_i, r = s_i^2$ from (2), $\exists k_i \in \{1, ..., m\}$ s.t.

$$\sum_{j=0}^{s_i^2-1} \binom{b_j(k_i, s_i^2, S_i)}{2} < 1.$$

  3. Space usage: $6s + 1$ table cells

Identification of Hash Functions

- Expensive to exhaustively try all values of $k \in \{1, ..., m\}$
- The following modification of (2) does the trick
  $\forall V \subseteq M$ of size v, and all $r \geq v$,

  $$\sum_{i=0}^{r-1} \binom{b_i(k, r, V)}{2} < 2\frac{v^2}{r}.$$

  for at least one-half of the choices of $k \in \{1, ..., m\}$.
- By random sampling from $\{1, ..., m\}$, a k satisfying the above can be found in $O(v)$ expected time

Further Work

- The $O(1)$ search time hashing scheme is based on the work of Fredman, Komlós, and Szemerédi [2]
- A version of the hash table for dynamic dictionaries has been provided by Dietzfelbinger, Karlin, Mehlhorn, Meyer auf der Heide, Rohnert, and Tarjan [3]
  1. Their data structure guarantees constant search time, and the update time is bounded by a constant only in the amortized and expected sense
  2. They also prove lower bounds showing that the worst-case amortized time for an update must be at least logarithmic

Cuckoo Hashing [4]

- Solves the Dynamic Dictionary problem
- Achieves worst case constant lookup time
- And amortized expected constant update time
- Space usage is roughly $2|S|$
- Does not use perfect hashing
- Very simple to implement

Cuckoo Hashing

- Dictionary uses two hash tables T_1 and T_2 each consisting of r words
- There are two hash functions $h_1, h_2 : U \to \{0, ..., r-1\}$
- Every key $x \in S$ is stored either in cell $h_1(x)$ of T_1, or in cell $h_2(x)$ of T_2, but never in both
- The lookup function is

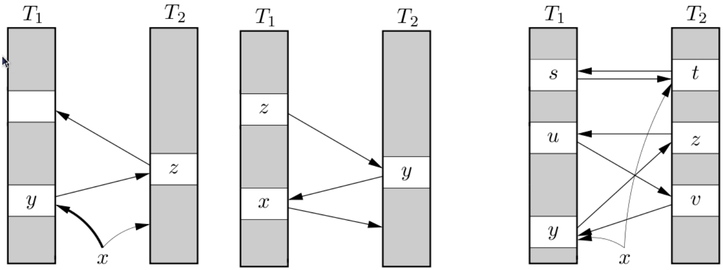
function lookup(x)

return $T_1[h_1(x)] = x \vee T_2[h_2(x)] = x$

end

Insertion

- "Cuckoo approach", kicking other keys away until every key has its own "nest"
- Figure: (a) Successful Insertion (b) Failed Insertion



(a) (b)

Insertion Procedure

procedure insert(x)
  if lookup(x) **then return** /* x already present */
  loop MaxLoop **times** /* MaxLoop is typically $O(\log n)$ */
    $x \leftrightarrow T_1[h_1(x)]$ /* swap the values */
    if $x = \perp$ **then return** /* $\perp$ indicates NULL value */
    $x \leftrightarrow T_2[h_2(x)]$
    if $x = \perp$ **then return**
  end loop
  rehash(); insert(x) /* Insertion has failed, select new h_1 and h_2 and attempt insertion again */
end

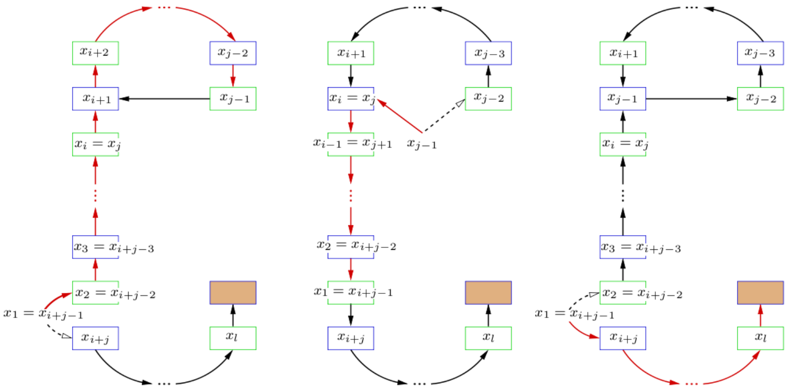
Hash Functions

- h_1 and h_2 are selected from a family that is $(1, n^\delta)$ - universal (for some constant $\delta > 0$) with probability $1 - O(1/n^2)$ when restricted to any set of r^2 keys [5]
- **Definition:** A family $\{h_i\}_{i \in I}, h_i : U \to R$, is (c, k)-universal if, for any k distinct $x_1, ..., x_k \in U$, any $y_1, ..., y_k \in R$, and u.r. $i \in I$, $Pr[h_i(x_1) = y_1, ..., h_i(x_k) = y_k] \leq c/|R|^k$.
- For the above reason the hashing algorithm ensures that no more than r^2 insertions are performed without changing the hash functions
- For n larger than some constant, $MaxLoop < n^\delta$ ensuring w.h.p. that the hash family is $(1, MaxLoop)$-universal
- Implies that h_1 and h_2 act like truly random functions on any set of keys processed during the insertion loop

Analysis: Behavior of the Insertion Procedure

1. No hash table cell is visited more than once. Runs through a sequence of nestless keys $x_1, x_2, \ldots$ with no repetitions
2. Refer to the following figure:

Sequence of pushes through T_1 and T_2:



Analysis: Behavior of the Insertion Procedure

- **Claim:** Suppose that the insertion procedure does not enter a closed loop. Then for any prefix $x_1, x_2, ..., x_p$ of the sequence of nestless keys, there must be a subsequence of at least $p/3$ consecutive keys without repetitions, starting with an occurrence of the key x_1, i.e., the key being inserted.
- **Proof:**
  1. Trivial for the case when the insertion procedure never returns to a previously visited cell
  2. If $p < i + j$, the first $j - 1 \geq \frac{i+j-1}{2} \geq p/2$ nestless keys form the desired sequence
  3. For $p \geq i + j$, one of the sequences $x_1, ..., x_{j-1}$ and $x_{i+j-1}, ..., x_p$ must have length at least $p/3$

Analysis: Probability Bounds

- The insertion loop runs for at least $t\,(\leq \mathit{MaxLoop})$ iterations, when one of the following events occurs:
  1. E_1: The insertion procedure has entered a *closed loop*, i.e., x_l moved to a previously visited cell, for $l \leq 2t$
  2. E_2: The insertion procedure without entering a *closed loop* has processed a sequence of $x_1, x_2, ..., x_{2t}$ nestless keys
  3. From the previous claim, E_2 is equivalent to the insertion procedure having processed a sequence of at least $(2t-1)/3$ consecutive distinct keys starting with x_1

Analysis: Probability Bounds

- $Pr\{E_1\}$
  1. Let $v \leq l$ denote the number of distinct nestless keys
  2. Number of ways in which the *closed loop* can be formed $< v^3 r^{v-1} n^{v-1}$
  3. Since $v \leq MaxLoop$, the hash functions are $(1, v)$-universal
  4. Implies each possibility occurs with probability $\leq r^{-2v}$
  5. Using $r/n > 1 + \epsilon$, we get $Pr\{E_1\}$ to be at most:

$$\sum_{v=3}^{l} v^3 r^{v-1} n^{v-1} r^{-2v} \leq \frac{1}{rn} \sum_{v=3}^{\infty} v^3 (n/r)^v = O\left(1/n^2\right).$$

Analysis: Probability Bounds

- $Pr\{E_2\}$
  1. Let $b_1, ..., b_v$ be the sequence of $v = \lceil (2t-1)/3 \rceil$ distinct nestless keys
  2. For either $(\beta_1, \beta_2) = (1, 2)$ or $(\beta_1, \beta_2) = (2, 1)$, we have

$$h_{\beta_1}(b_1) = h_{\beta_1}(b_2), h_{\beta_2}(b_2) = h_{\beta_2}(b_3), h_{\beta_1}(b_3) = h_{\beta_1}(b_4),$$

  3. Given b_1 there are at most n^{v-1} possible sequences of v distinct keys
  4. Since the hash functions are chosen from a $(1, MaxLoop)$-universal family, the probability that the $v-1$ equations above hold is bounded by $r^{-(v-1)}$
  5. Using $r/n > 1+\epsilon$, we can bound $Pr\{E_2\}$ by

$$2(n/r)^{v-1} \leq 2(1+\epsilon)^{-(2t-1)/3+1}.$$

Analysis: Number of Iterations

- Expected number of iterations in the insertion loop is bounded by:

$$1 + \sum_{t=2}^{MaxLoop} \left(2\,(1+\epsilon)^{-(2t-1)/3+1} + O\left(1/n^2\right)\right)$$

$$\leq 1 + O\left(\frac{MaxLoop}{n^2}\right) + 2\sum_{t=0}^{\infty}\left((1+\epsilon)^{-2/3}\right)^t$$

$$= O\left(1 + \frac{1}{1-(1+\epsilon)^{-2/3}}\right) = O\left(1 + 1/\epsilon\right).$$

Analysis: Cost of Rehashing

- A failed insertion causes a forced rehash and this happens when the insertion loop runs for $t = MaxLoop$ iterations due to:
  1. Entering a closed loop with probability $O\left(1/n^2\right)$
  2. Without entering a closed loop with probability at most $2\left(1+\epsilon\right)^{-(2MaxLoop-1)/3+1} = O\left(1/n^2\right)$ for $MaxLoop = \lceil 3 \log_{1+\epsilon} r \rceil$
- From above, the combined probability of forced rehash is $O\left(1/n^2\right)$
- Each rehash costs $O(n)$ on expectation, $O(1)$ expected time per insertion for a total of n insertions
- Thus the expected cost per insertion for a forced rehash is $O(1/n)$

Analysis: Cost of Rehashing

- Another rehash happens when r^2 insertions have been performed with no failed insertions
- The amortized expected cost per insertion of such rehashes is $O(1/n)$
- Summing up, we get constant amortized expected time for insertion

Further Work

- Due to the constraint $r > (1 + \epsilon)\, n$, the tables are a bit less than half full
- Fotakis et al. [6] analyzed a generalization of Cuckoo Hashing with d possible locations for each key, showing that in this case a space utilization of $1 - 2^{-\Omega(d)}$ can be achieved, with constant expected time for insertions
- Devroye and Morin [7] did a further analysis of Cuckoo Hashing using a graph-theoretic interpretation
- The analysis of Devroye and Martin was further extended by Kutzelnigg [8] who made several asymptotic results much more precise

References I

Motwani-Raghavan.
Randomized Algorithms, chapter 8, pages 221–228.
Cambridge University Press, 2004.

Fredman.
Storing a sparse table with $O(1)$ worst case access time.
Journal of the ACM, 31:538–544, July 1984.

M. Dietzfelbinger.
Dynamic perfect hashing: Upper and lower bounds.
In *FOCS*, pages 524–531, 1988.

Rasmus Pagh.
Cuckoo hashing.
Journal of Algorithms, 2004.

References II

Alan Siegel.
On universal classes of fast high performance hash functions, their time-space tradeoff, and their applications.
In *FOCS*, pages 20–25, 1989.

Dimitris Fotakis.
Space efficient hash tables with worst case constant access time.
In *STACS*, volume 2607, pages 271–282, 2003.

Luc Devroye.
Cuckoo hashing: Further analysis.
Information Processing Letters, 86(4):215–219, 2003.

Reinhard Kutzelnigg.
Bipartite random graphs and cuckoo hashing.
In *DMTCS*, pages 403–406, 2006.