Exercise: Space-efficient linear probing

Rasmus Pagh

July 13, 2014

Following an idea of Cleary, we will see how to save space in a linear probing hash table storing a size-n set $S \subseteq U$ that is "not too small" compared to U. Let $\varepsilon, \delta > 0$ be constants such that $(1+\delta)n$ and $\log_2(1/\varepsilon)$ are integer. In particular let $r = (1+\delta)n$ denote the hash table size, and suppose that $U = \{1, \ldots, r/\varepsilon\}$, such that S is roughly a ε-fraction of U. For simplicity we will assume that S is a random set, which can be achieved by performing an initial random permutation of U (or in some cases using simple hash functions, see application below).

The baseline solution is to store the elements of S using $\lceil \log_2 |U| \rceil$ bits, i.e., more than $n \log_2 |U|$ bits in total. To improve this for ε not too small the idea is to use a very simple hash function that extracts the $\log_2 r$ most significant bits of each key in S, more precisely $h(x) = \lfloor \varepsilon x \rfloor$.

a) Argue that knowledge of $h(x)$ and $q(x) = x \mod (1/\varepsilon)$ suffices to compute x, and that storing $q(x)$ requires only $\log_2(1/\varepsilon)$ bits.

b) Consider a "run" of keys $R \subseteq S$ stored in an interval I of size $|R|$. Argue that $2|I|$ bits suffice to encode the multiset $h(R)$ of hash values relative to I.

c) Suppose that you inserted elements of R, in *sorted order*. Argue that knowledge of I and the multiset of corresponding h-values, $\{\lfloor \varepsilon y \rfloor \mid y \in R\}$, suffices to locate the set of keys in R having a particular h-value.

d) Putting the above together, argue that $\log_2(1/\varepsilon) + 2$ bits per hash table entry suffices to encode S, giving a total space usage of $(1+\delta)n \log_2(1/\varepsilon) + O(n)$ bits.

Application. The *Bloom filter* is an important data structure for storing a set $S \subseteq U$ of n keys approximately, such that membership queries have an ε *false positive* probability. That is, a membership query for a key in S always returns 'yes', but a membership query for a key not in S is allowed to answer 'yes' with probability at most ε.[1] This allows a data structure of size around $1.44\, n \log_2(1/\varepsilon)$ bits — much less than what would be needed to store S exactly.

When $\varepsilon \leq 2^{-5}$, however, one can do better than a Bloom filter in terms of space and cache performance. To do this use linear probing with r and ε as above. It is not difficult to show, using a union bound, that:

Fact 1 *Suppose $H : U \to \{0, \ldots, r/\varepsilon - 1\}$ is 2-independent, and consider the set of hash values $H(S) = \{h(x) \mid x \in S\}$. Then if $x \notin S$ we have $\Pr[H(x) \in H(S)] \leq \varepsilon$.*

This tells us that to support approximate membership on S, it suffices to answer membership queries on $H(S)$. If we let $H(S)$ be 5-independent, we also get in the data structure above $h(H(x))$ is also 5-independent, so the analysis of query time is valid. The space becomes $(1+\delta)n(\log_2(1/\varepsilon) + 2)$ bits, which can be shown to be near-optimal:

Fact 2 *A space usage of $n \log_2(1/\varepsilon) - o(n)$ bits is required for any approximate membership data structure whenever U is much larger than S.*

[1] The probability is over random choices made when constructing the data structure