Randomized Binary Tree - Search Structure

What is $\tilde{H}(N)$?

Just remember what we are doing - or maintain a history.

1. $\tilde{H}(N)$ is the history of computed partitions $H(N^0), \cdots, H(N^n)$.
2. Its not feasible to remember whole of $H(N^i)$, for all i's.
3. The difference between $H(N^i)$ and $H(N^{i+1})$ is small, so we just remember the difference between successive partitions.

Anil Maheswari
School of Computer Science
Carleton University

4. **history(i)** be the history of first i-additions.

5. $\tilde{H}(N)$ =history(n). History(0) consists of just one node, corresponding to $\Re$.

6. The leaves of history(i) will correspond to the intervals of $H(N^i)$, and its internal nodes will be labeled with the points in N^i.

7. Inductively, assume $history(i)$ has been defined. Let S_{i+1} be the randomly chosen point, belonging to interval I of $H(N^i)$. Interval I is killed and two new intervals I_1 and I_2 are created corresponding to the split of I.

8. history($i+1$) is obtained from history(i) by labeling the leaf of history(i), corresponding to interval I, with S_{i+1} and assigning two children, one each for I_1 and I_2.

Analysis of Randomized Binary Tree

Expected cost of computing $\tilde{H}(N)$ is $O(n \log n)$.
The analysis is similar to the quick-sort analysis, *so we skip*.

What is the expected search/query cost?

1. What does it mean?

2. Fix a query point $p \in \Re$.

3. *Backward Analysis*: Imagine tracing p backward in the history. Starting with $H(N) = H(N^n)$, imagine deleting points $S_n, \cdots, S_0$, one at a time, in the reverse order.

4. Define a 0-1 random variable V_j.
   $V_j = 1$ if and only if the interval in $H(N^j)$ containing p is adjacent to the deleted point S_j. (This interval is destroyed during the imaginary deletion o S_j)

5. Length of the search path in history(n)=V=$\sum_1^n V_j$.

6. Each point in N^j is equally likely to occur as S_j.

7. Probability$(V_j = 1)\leq 2/j$; since at most two points are adjacent to the interval in $H(N^j)$ containing p.

8. Expected Value of $V_j = 2/j$.

9. By Linearity of Expectation, Expected Value of $V = O(\sum 1/j) = O(\log n)$.

10. Thus for a fixed query point p, the expected length of the search path is $O(\log n)$.

Showing High-Probability Bounds: $\tilde{O}(\log n)$

Definition: $\tilde{O}$: We say $f(n)=\tilde{O}(\log n)$, if for some constant, $c > 0$, $f(n) < cg(n)$, with probability $1 - 1/p(n)$, where $p(n)$ is a polynomial whose degree depends upon c.

We want to show following two statements:

1. Length of the search path of a fixed query point is $\tilde{O}(\log n)$.
2. Search cost is $\tilde{O}(\log n)$regardless of the query point.

1 is shown using *Chernoff Bounds* for harmonic random variables.

High-Probability Bounds for Arbitrary Queries

Observation I: Search path of all query points lying in the same interval of $H(N)$ are identical. Therefore the number of distinct search paths are $n+1$.

Observation II: Each search path has $\tilde{O}(\log n)$length (*can be shown by taking a sample point in the interval and using the previous analysis*)

Observation III: For a fixed interval I, the error probability that the search path is longer than $c \log n$ is $1/p(n)$, and $p(n)$ is the polynomial whose degree can be made large by choosing a large value of c.

Observation IV: Since the number of intervals are $n+1$, the probability that search for any interval exceeds more than $c \log n$ is $O(n/p(n))$.

Therefore, maximum length of search path in $\tilde{H}(N)$ is $\tilde{O}(\log n)$

USEFUL OBSERVATION: A Generalization

Suppose the length of the search path in the given RSS is $\tilde{O}(\log n)$, for any fixed query point. Assume that the space under consideration can be partitioned into $O(n^a)$ regions, for a constant $a > 0$, so that
I) the partition depends only on the set N of objects in the search structure, and not on the state of the search structure.
II) the search path of any two points in the same region are identical.
Then the depth of the search structure is also $\tilde{O}(\log n)$.

Quick-sort runs in $\tilde{O}(n \log n)$

Main Idea: Amortize the cost of adding $S = S_{i+1}$.

This is achieved by charging $O(1)$ cost to each point in the conflict list of the (destroyed) interval $I \in H(N^i)$ containing S.

Total cost charged to a fixed point $Q \in N$, over the whole course of the algorithm is equal to the number of times the interval containing Q has been split (or destroyed).

This is same as the search path of Q in history(n). Therefore, the total cost charged to a fixed point in N is $\tilde{O}(\log n)$.

The total running time is $\tilde{O}(n \log n)$.

Ist Randomized Result

Theorem: Given a set of points N on the real line $\Re$, the partition $H(N)$ and an associated randomized search tree can be built in $\tilde{O}(n \log n)$time. The point location cost is $\tilde{O}(\log n)$.

Skip Lists

A Random-Sampling Technique to compute the Dynamic search structure

Caution: Change of notation. M = Set of point on $\Re$. Sample(M) denotes the search structure.

-Starting with M, compute a sequence of sets, $\emptyset = M_r \subseteq M_{r-1} \cdot \subseteq M_2 \subseteq M_1 = M$.

-M_{i+1} is obtained from M_i by flipping the coin independently for each point in M_i and retaining only those points for which there was success.

-These sets are called *gradation* of M.

-Sample(M) consists of r levels, $H(M_i)$ forms the ith level of Sample(M). With every point $S \in M_i$ stored at the ith level, we associate a *descent* *pointer* to the occurrence of the same point in the (i-1)th level. Also, attach a point at $-\infty$ at each level.

Point Location in Sample(M)

-Let S be the query point.

-Trivially locate S in $H(M_r)$, which consists of one interval, whole of R.

-Inductively, assume we have located S in $H(M_i)$. Let Δ_i be the partition (interval) containing S in $H(M_i)$.

-Using the descent pointer associated with the left endpoint of Δ_i, we can search through all children of Δ_i and we get the required interval containing S in $H(M_{i-1})$.

What is the cost of the search?
Its proportional to the number of children of the intervals (i.e. the intervals containing S) on the search path.

Anil Maheswari
School of Computer Science
Carleton University

Analysis

Many things to be analyzed

1. Expected number of levels in Sample(M) is $\tilde{O}(\log n)$.

2. Expected space requirement of Sample(M) is $\tilde{O}(m)$.

3. Sample(M) can be computed in $\tilde{O}(m \log m)$ time.

4. Search cost is $\tilde{O}(\log m)$.

Expected number of levels in Sample(M) is $\tilde{O}(\log n)$

1. Fix a positive integer k.

2. A fixed point $Q \in M$ is contained in more than k levels of Sample(M) with probability $1/2^k$.

3. Probability that at least one point in M is stored in more than k levels is bounded by $m/2^k$. (*probability of union of events is bounded by the sum of their probabilities.*)

4. If $k = c \log m$, $c > 1$, then the probability bound is $1/m^{c-1}$.

5. Therefore, the number of levels in Sample(M) is $\tilde{O}(\log m)$.

Anil Maheswari
School of Computer Science
Carleton University

Expected space requirement of Sample(M)

1. Fix a point $Q \in M$. Let $h(Q)$ denote the number of levels in Sample(M) containing Q. $h(Q)$=1 + the number of success obtained.

2. $h(Q) = i$ if and only if the first $i-1$ tosses for Q are success and the ith a failure.

3. $h(Q)$ *is said to be geometrically distributed with a parameter 1/2.* Probability($h(Q) = i$)=$1/2^i$.

4. Expected value of $h(Q) = \sum_i \frac{i}{2^i} = O(1)$.

5. Expected number of levels in Sample(M) containing a fixed point is $O(1)$.

6. Size of Sample(M) is proportional to $\sum_Q h(Q)$, where Q ranges for all points in M.

7. By linearity of expectation, it follows that the expected space requirement of sample(M) is $O(m)$.

8. Actually, it is $\tilde{O}(m)$, since all coin tosses are independent, and $h(Q)$'s are independent geometric distributions. This follows by using Chernoff's bounds on sum of independent geometric distributions.

Expected cost of constructing Sample(M)

1. Use quick-sort at each level of Sample(M). This needs $\tilde{O}(m_i \log m_i)$ time at the ith level.

2. Over all levels, this sums to $\tilde{O}(\sum_i m_i \log m)$.

3. We know that $\sum_i m_i$ is the size of Sample(M) and which is $\tilde{O}(m)$.

4. Sample(M) can be computed in $\tilde{O}(m \log m)$ time.

Anil Maheswari
School of Computer Science
Carleton University

Expected cost of search in Sample(M)

Earlier Observation: Cost of the search is proportional to the number of the children of intervals on the search path.

1. Fix a query point S.

2. For each Δ_i on the search path, let $l(\Delta_i)$ denote the number of its children. The cost of the search is bounded by $O(\sum_i 1 + l(\Delta_i))$.

3. Let $l_0(\Delta_i)$ and $l_1(\Delta_i)$ denote the number of points in $M_{i-1} \setminus M_i$ that are contained in Δ_i to the left and right of S, respectively.

4. $l(\Delta_i) = l_0(\Delta_i) + l_1(\Delta_i)$.

5. We estimate $\sum_i l_0(\Delta_i)$, other being similar.

6. Fix M_{i-1}. When is $l_0(\Delta_i) = k$?

7. Look at M_{i-1}. $l_0(\Delta_i) = k$ implies that the tosses for k of the points to the left of S in M_{i-1} were failures!

8. This happens with probability $1/2^k$. Therefore expected value of $l_0(\Delta_i)$ for a fixed M_{i-1} is $O(1)$.

9. The expected bound is independent of M_{i-1}, so this is unconditional as well.

10. Furthermore, Chernoff bounds will apply here, since tosses being independent.

11. We already know that the number of levels are $\tilde{O}(\log m)$, therefore the search cost is $\tilde{O}(\log m)$.

12. This h.p.bounds are for a fixed query point S. But notice that the number of distinct search paths in sample(M) is $O(m)$. Therefore the maximum search cost is also $\tilde{O}(\log m)$

Anil Maheswari
School of Computer Science
Carleton University

Updates - Dynamic Searching

How to add a new point to Sample(M)!

1. Let S be the new point.
2. Toss a coin, till we get a failure, let there be j successes.
3. We want to add S in j levels, 1 to $j+1$.
4. What's the cost of adding S?
5. Its same as the search cost! Why?

Anil Maheswari
School of Computer Science
Carleton University

6. What about deletion. Cost of removal per level is $O(1)$ and expected number of levels a point is in is $O(1)$. Hence it follows that deletion can be done in $O(1)$ time.

SUMMARY

1. **Randomized Incremental Construction**- backward analysis and conflict lists.

2. **Randomized Binary Trees**- Maintaining the History. Analyze for a single query and then for arbitrary queries.

3. **High Probability Bounds**- Use Chernoff bounds for random (almost independent) variables.

4. **Skip Lists**- Simple ideas and updates are same as the searching!