Survey: The Solutions to Data Structuring Problem using Randomized Algorithms

K. Arun[1] K. Kalaivani[2] R. Ieshwarya[3] S. Sridharani[4] K. S. Prakhash Raj[5]
[1,3,4,5]Student [2]Assistant professor
[1,2,3,4,5]Department of Computer Science & Engineering
[1,2,3,4,5]Arasu Engineering College, Kumbakonam, Tamil Nadu, India.

Abstract— The motivation of this paper is to provide the idea about what is randomized algorithm and how it solves the issues of data structuring problem using deterministic algorithm. Due to the increase in advancement of technologies, the randomization techniques are eventually grown with that. It is because of lack in deterministic techniques and the best advantages of randomization techniques in the data structuring problems. The hash functions using random value increase the security as well as improve the performance, and the inclusion of random values in online algorithm provides better solutions in the paging problems.

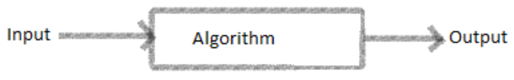
Key words: Randomized Algorithm, Data Structuring Problem, Model of Randomized Algorithms, and Universal Hashing

I. Introduction

Algorithm is a procedures or formula for solving a problem. Algorithm can be classified into two types,

1) Non-Deterministic Algorithm.
2) Deterministic Algorithm.

Non-deterministic algorithm is an algorithm that, even for the same input, can exhibit different behaviors on different runs, as opposed to a deterministic algorithm. There are several ways an algorithm may behave differently from run to run. Deterministic algorithm is an algorithm which, given a particular input, will always produce the same output, with the underlying machine always passing through the same sequence of states.

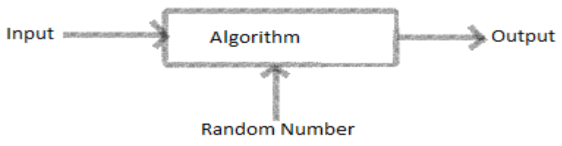


A. *Disadvantage of Deterministic Algorithm:*

It may be difficult to formulate an algorithm with good running time.

II. Randomized algorithm

In addition to the input, the algorithm uses a source of pseudo random numbers. During execution it takes random choices depending on those random numbers. The behaviors can vary if the algorithm is run multiple times on the same input. When compared to randomized algorithm the deterministic algorithm of running time or probability of getting the correct answers misuse fully difficult. In deterministic algorithm truly random number is impossible. Example: Randomized Quick Sort is an extension of Quick Sort in which pivot element is chosen randomly.



A. *Algorithm RandQS:*

Input: A set of numbers.
Output: The element of S sorted in increasing order.

1) Choose an element of uniformly at random from S: every element in S has equal probability of being chosen.
2) By comparing each element of S with Y, determine the set S_1 of element smaller than Y and the set S_2 of element larger than y.
3) Recursively sort S_1 and S_2. Output the sorted version of S_1 followed by Y and then the sorted version of S_2.

B. *Running Time*

The worst case time complexity of this algorithm is O (n^2).

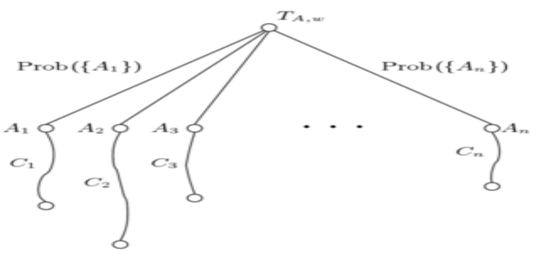
III. Models of randomized algorithm

The randomized algorithm is a special care of stochastic algorithm. The stochastic algorithm is partially controlled by a random process. The randomized algorithm is not allowed to behave poorly on any input instances. The design of randomized algorithm is subscribe to the very strong requirements. It's work efficiently and correctly with high probability on every input. Randomized algorithm has two models. They are,

1) First Model
2) Second Model

A. *First Model:*

The randomized algorithm as a probability distribution over a finite collection. A_1, A_2, A_3, A_4 …A_n of deterministic strategic algorithm.



A chosen the ith deterministic algorithm with probability A_i for I equals to at the computation is completely determine. {C_1, C_2... C_n} of A on W is a different length. One can measure the efficiency of the work A on W by the expected time complexity of A on W is defined by,

$$\text{Exp} - \text{Time}_A (W) = E|Z|$$

$= \sum_{i=1}^{n} \text{prob}(\{c_i\}) - Z(c_i)$

$= \sum_{i=1}^{n} \text{prob}(\{c_i\}) - \text{Time}(c_i)$

Expected time complexity of randomized algorithm A is the function.

$\text{Exp-Time}_A(n) = \max\{\text{Exp-Time}_A(W) \,/\, \text{the length of W is n}\}$

$\text{Time}_A(n) = \max\{\text{time}(c) \,/\, C \text{ is a run of A an input of length n}\}$

The Reliability of a randomized algorithm A^{-1} on input W.

$$X(i) = \begin{cases} 1 & \text{if } C_i \text{ computes the correct result on } W_i \\ 0 & \text{if } C_i \text{ computes the wrong result on } W_i \end{cases}$$

For {i=1, 2…n} then

$E(x) = \sum_{i=1}^{n} X((i).\text{prob}\{c_i\})$

$= \sum_{i=1}^{n} 1.\text{prob}\{Ci\} + \sum_{i=1}^{n} 0.\text{prob}\{c_i\}$

$E(x) = \text{Prob}(\text{Event}(X = 1))$

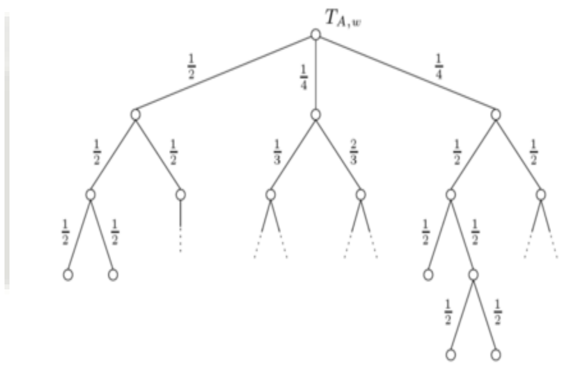
The probability that a computes the right result.

E(x) is a success probability of A on W.

1-E(x) is an error probability of A on W.

B. Second Model:

The randomized algorithm is represented as the non-deterministic algorithm with a probability distribution for every non-deterministic choice.



One can describe all computation of randomized algorithm A on input W by the computation tree $T_{A,\ W}$. Every path from a root to a leaf correspond a computation A on W.

IV. Classification of Randomized Algorithm

This classification is based on the kind and size of the error probability of randomized algorithms. The classification of randomized algorithms is not the same for all different computing tasks. The meaning of a "wrong output" may vary under different circumstances. For solving a decision problem or for computing a function, the error probability is really the probability of computing a wrong result.

1) LAS VEGAS Algorithm
2) One sided error Monte Carlo algorithm
3) Bounded Monte Carlo algorithm
4) Unbounded error Monte Carlo algorithm

A. LAS VEGAS Algorithms:

The Las Vegas algorithms are randomized algorithms that guarantee that every computed output is correct. This means that wrong results are forbidden in this model. It can find two different models. These models differ in whether or not the answer "?" with the meaning "I do not know" is allowed. Let us consider the case that the answer "?" is not allowed. In what follows,

A(x) always denotes the output of the algorithm A on a given input x.

1) Definition 1:

A randomized algorithm A is called a Las Vegas algorithm computing a function F if, for any input x.

Prob (A(x) = F(x)) = 1

2) Definition 2:

Let A be a randomized algorithm that allows the answer "?". We say that A is a Las Vegas algorithm for a function F if, for every input x,

1) Prob(A(x) = F(x)) ≥ 1/2, and
2) Prob (A(x) = "?") = 1 − Prob(A(x) = F(x)) ≤ 1/2.

B. One-Sided-Error Monte Carlo Algorithms:

This type of randomized algorithm is considered for decision making problems only.

Let A be a randomized algorithm and let (Σ, L) be a decision problem. We say that A is a one sided error Monte Carlo algorithm for L,

1) for every x ∈ L, Prob(A(x) = 1) ≥ ½ ,
2) For every x ∉ L, Prob (A(x) = 0) = 1.

C. Bounded-Error Monte Carlo Algorithm

Let F be a function. We say that a randomized algorithm A is a bounded-error Monte Carlo algorithm for F,

If there exist a real number ε, 0 ≤ ∈ ≤ ½. Such that, for every input x of F,

Prob (A(x) = F(x)) ≥ ½ +ε.

We call attention to the fact that the upper bound ½ - ε on the error probability.

D. Unbounded Error Monte Carlo Algorithm

Let F be a function. We say that a randomized algorithm A is a unbounded-error Monte Carlo algorithm computing F, if for every input x of F,

Prob (A(x) = F(x)) > 1/2.

A randomized algorithm cannot be used for computing a function F when the error probability is not less than 1/2. If the error probability is at least ½, then a wrong result may be produced as correct result.

E. DATA STRUCTURING PROBLEM

In the static dictionary problem we are giving a set of key 'S' and the data are must organized into a data structure that supports the efficient processing of finding queries. In the dynamic dictionary problem the set of key 'S' is not provided in advance. Instant it is constructed by a series of insert and delete operation that are inter mingled with the fine queries. These problems can be solved by using Random Treaps and Random Skiplist.

F. Random Treaps

It does not require any explicit balance information and the expected the number of rotations performed is small for each operation. It is extremely simply to implement. It is a full, endogenous binary tree whose nodes have key value associated with them in a binary search tree. Every node has both a search key and a priority where the in order sequence of search keys is sorted and each node priority is smaller than the priority of these children. Treaps is simultaneously a binary search tree for the priorities in our example leers are used for the search keys and number of the priority. Top half of each node shows its priority and bottom half shows its search keys. All the keys and priorities are distinct. The structure of the treap is completely the determined by the search keys and priority of its nodes.

G. *Random Skiplist:*

The skip list has many of the usual desirable properties of balanced BST, but their structure is very different. At a high level, a Skiplist is just a sorted linked list with some random shortcuts. To do a search in a normal single Linked list of length n, we obvisoly need to look at n, we need to look at n item in the worst case manner. To speed this process, we can make a second level list that contains roughly half the item from the original list. For each item in the original list, we duplicate it with probability of ½.

Then string together the entire duplicate in to a second sorted linked list has a pointer to its original. Just for the safe we also use sentinel nodes at the being and end of both list. To find the value x, first we scan for x in the shortcut list starting at the – sentinel, if we find x then we done, otherwise we reach some bigger value bigger than x and we know that x is not in the Skiplist. Let w be the largest item less than the item in the short cut list. In the second phase we scan for x in the original list from w again, if we reach the value bigger than the x. Then the expected node examine in the first phase is almost n/2. The expected number of examine the second phase is $1+\sum_{k\geq 0} 2^{k-2}$.

H. *Algorithm:*

```
SKIP LISTFIND (X, L)
    V←L
    while (v≠NULL & Key (v) ≠ (x)
    if (key (right (v)) > x
    V ←down (V)
    else
    V← right (v)
    return V
```

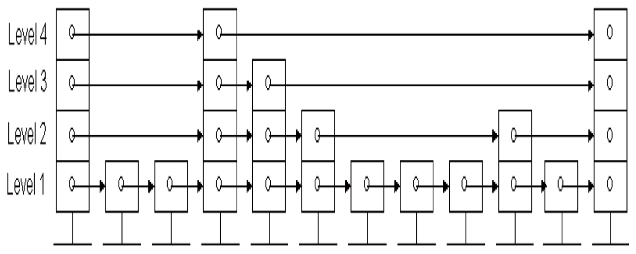


Fig. 2: The working of Skiplist

V. HASH TABLE

A hash table or hash map is data structure that associates keys with the locations. A hash table (T) is a data structure for the dictionary problem that consisting of ‘n’ cells indexed by n= {0, 1 ….n-1} and hash function ‘h’, which is a mapping from ‘M’ in to ‘n’. The hash function is a type of finger printing function for the keys In M and it specifies a location in the table for each element. A collision said to be occur between two distinct keys x and y. If h(x) = h(y) at they are collide at corresponding location of ‘T’.

A. *Hash function*

The mapping between a key and a bucket is called a hash function. A hash function H=M→N said to be a perfect for a set S⊆M, If h doesn’t cause any collision among the key of the set S, given a perfect hash function for a set S, we can use the hash table to process the sequence for finding the operation in O (1) time each and store each element K϶S at the location T [h (k)]. To search a key ‘q’ just check whether T [h (q)] =q or not. For n<m any function H must map some two elements of M to the same location and so it can’t be perfect for any set containing those two elements thus perfect has function are useless for dynamic dictionary problem.

Hashing is a ubiquitous information retrieval stagey for providing efficient access to information based on a key. A collision occurs when two or more keys map to the same location. No application can store all the keys in a same domain. These are the major issues in Hashing.

B. *Methods for creating a hash function*

- Division Method
- Multiplication method
- Universal hashing

C. *Universal Hashing*

Problem: For any hash function h, a set of key is exists that cause average access time of a hash table to sky rocket. (i.e.) An advisory can pick all keys from {k ϵ u; h (k) = i} for some slot of i.

Idea or solution : choose the hash function at random independently of the keys even if an advisory can see our code may can’t find a bad set of keys, since advisory doesn’t know how exactly which hash function will be chosen.

Definition: Let ‘U’ be a Universe of keys, H be a finite collection of hash function each maps ‘U’ to {0, 1, 2, ….. m-1}. We say H is universal if for all x, y ϵ u where x ≠ y. we have | {h ϵ H: h (a) = h(y)}| = |H|/m. (i.e.) the chance of a collision between x & y is 1/m. if we choose h randomly from H.
Theorem: Universality is good:

Let h be a hash function chosen (uniformly) at random from a universal set H of hash function. Suppose, h is used to hash n arbitrary keys into a m slot of a table T, then for a given key x, we have E=n/m. (#collisions with x).

D. *Proof*

Let C_x be a random variable denoting the total no of collisions in T with x, and let

$$C_{xy} = \begin{cases} 1 & h(x) = h(y) \\ 0 & \text{otherwise} \end{cases}$$

Note: E [C_{xy}] = 1/m and $C_x = \sum C_{xy}$ where y ϵ T-{x}.
Now take expectation on both sides,
E [C_x] = E [$\sum C_{xy}$ where y ϵ T-{x}]
Linearity of expectation gives that
= $\sum C_{xy}$ where y ϵ T-{x} E [C_x]
= $\sum C_{xy}$ where y ϵ T-{x} 1/m
=n-1/m algebra.

E. *Algorithm constructing a set of Universal Hash functions*

Step 1: Condition →Select m to be a prime

Step 2: Preprocessing → Decompose of key 'k' into r + 1 digits' $k = <k_0, k_{1....}k_r>$ Where $k_i \in \{0, 1... m-1\}$
Step 3: Randomness → pick $a = <a_0, a_1... a_r>$ at random each $i \in \{0, 1... m-1\}$
Step 4: Hash function $h(K) = (\sum a_i k_i) \bmod m$. for(i=0, 1...r). Ex: Hashing IP addressing and Hash family in hash method

VI. Online Algorithm

The computing problem considered until know are be classical task for which for any given input, one has compute an output. But in practice one also has to deal with the following tasks.

One obtains, only part of the input and forced to process that part.

After it complete the process, another part of the input that has to be given that has to be processed immediately.
The input may be arbitrarily long and this kind of task are called online algorithm.

A. Definition

The online algorithm are the algorithms which receives the input in part (or) requests services are answer, each request before going to the next. It does not have over all view of the entire request sequence. (i.e.) The algorithms which should be able to produce the output without looking at the entire input sequence is called online algorithm.

Eg: Memory paging, Data structure, Resource allocation

Using competitive analysis their performance is compared to office algorithm and it has been discovered that Randomized Online Algorithms appear to outperform to the deterministic ones. We define the competitive ratio of online algorithms in order to get a reasonable measure of the quantity.

B. Definition for online problem

Let U= ($\sum_I$, $\sum_O$, L, M, Cost, goal) be an optimization problem that can be viewed as an online problem. An algorithm 'A' is an online algorithm for U if for every input $X=x_1, x2....x_n \in L$, the following conditions are satisfied
For all i =1, 2... n. $\{x_1, x_2 ...x_i\}$ is a feasible input.

A(x) =M(x), (i.e.) A always computes feasible solution.

For all I =1, 2... n, $A(x_1, x_2 ...x_i)$ is a part of A(X), the decisions made for the prefix $x_1, x_2 ...x_i$ of x can't be changed any more.
For every input x ϵ l, the competitive ratio

$$\text{comp}_A(x) = \max\left\{\frac{opt(u(x))}{cost(a(x))}, \frac{cost(a(x))}{opt(u(x))}\right\}$$

Where, opt (u(x)) - Cost of an optimal solution for the instance x of problems U. Let ⌋ ≥1 we say that A is a ⌋ competitive algorithm for U, if $\text{comp}_A(x) \leq$ ⌋
Note: ⌋ → Delta.

VII. Online Paging Problem

We should be familiar with paging problem in the computer, the cache memory which can hold k pages there is much larger slow memory which can hold an arbitrary number of pages. When a computer accesses a memory item, if the page containing is in cache, then the access is entirely fast and said to be a cache hit. If the item is not in the cache memory than the page containing it must be moved in the cache and some other memory may moved out. This is called as cache miss. The following are the some of the important Deterministic paging algorithm,

1) LRU(Least Recently Used)
2) FIFO(First In First Out)
3) LFU(Least Frequently Used)

A. Advisory Models

The advisories are the opponents who are trying to break the algorithms. When the paging algorithm is deterministic there are two possible advisory models.

B. Oblivious advisory

In the Oblivious advisory model the advisory may produce a sequence of paging request that is independent of the behavior of the deterministic paging algorithms.

C. Adaptive advisory

In adaptive advisory the advisory may produce the page requests that dependent on the all the action of the paging algorithm up to the current time. This model two more sub categories as

1) Adaptive on-Line Advisory
2) Adaptive off-Line Advisory

D. Randomized Online Algorithm

Though the advisory knows the randomized online algorithm he doesn't know which random will be taken.

1) Definition

Let U= ($\sum_I$, $\sum_O$, L, M, Cost, goal) be an optimization problem that can be viewed as an online problem. An algorithm 'A' is an random online Algorithm for U if for every input $X=x_1, x2....x_n \in L$ & every i ϵ {1, 2..... n-1}.

The output of every run of A on $x_1, x_2 ...x_i$ is a Feasible solution for the instance $x_1, x_2 ...x_i$ of the probability (U).

For every input $(C(x_1, x_2 ...x_i), xi_{+1})$, where $C(x_1, x_2 ...x_i)$ is a feasible solution for the instance $x_1, x_2 ...x_i$ of U.

(i.e.) $C(x_1, x_2 ...x_i) \in M(x_1, x_2 ...x_i)$ all runs of A compute a feasible solution for the instance $x_1, x_2 ...x_{i+1}$ of U and all the feasible solution involves $C(x_1, x_2 ...x_i)$ as partial solution for every problem instance x, let $S_{A, x}$ be the set of all computations of A on x and let $\text{prob}_{A,x}$ be the corresponding probability distribution on S_{Ax}. Let Z_x be the Random variable in (S_{Ax}, $\text{prob}_{A,x}$) is defined by $Z_x(C) = \text{comp}_A(x)$, For each computation $C \in S_{Ax}$. We define the expected competitive ratio of A on x as
EXP- $\text{comp}_A(x) = E[Z_x]$

E. Randomized paging Algorithm against oblivious advisory

The maker algorithm was purposed by fiat et al, the algorithm process a request sequence in phases at the being of each phase all pages in the memory system are unmarked whenever a page is requested then it is marked on. On a fault page is chosen uniformly at random from among the unmarked pages in a fast memory and this pages evicted. At the phase end, all the pages in fast memory are marked then all may erased before the new phase is started.

F. Marker Algorithm

```
1) Do forever
        2) Set marker bits $m_i=0$, for all $i=1, 2…. k$
        3) Do until all $m_i=1$ for all $i=1, 2…. k$
            4) Accept a page request;
            5) If request page is in cache location 1;
                6) Then set $m^i=1$
            7) Other wise
                8) Choose an unmarked cache location i at random
            9) Evict cache location j and bring new page into j
            10) Set $m_j=1$
        11) End
```

G. Analysis of Marker Algorithm

Marker algorithm is $2H_k$ competitive against any oblivious advisory model.
Where,
$H_k=\sum_{i=1}^{k} {}^{1}/_{i}$ is the harmonic number roughly H_k is log k.

VIII. Conclusion

Randomized algorithm run faster than the best known deterministic algorithm. It over comes many of the drawbacks of deterministic algorithm. Many randomized algorithm are simpler to describe and implement than the deterministic algorithm of comparable performance. Produce optimum output with high probability. It gives better performance than the deterministic algorithm. The randomization involves many applications and randomization will became want filed of programming environment.

References

[1] Amihood Amir, Karger's Min-cut Algorithm, Bar-Ilan University, 2009.
[2] George Bebis, Randomizing Quick Sort, Lecture Notes of CS 477/677: Analysis of Algorithms, University of Nevada.
[3] Avrim Blum and Amit Gupta, Lecture Notes on Randomized Algorithms, CMU, 2011.
[4] Hsueh-I Lu, Yao's Theorem, Lecture Notes on Randomized Algorithms, National Taiwan University, 2010.
[5] Rada Mihalcea, Quick Sort, Lecture Notes of CSCE3110: Data Structures, University of North Texas, http://www.cs.unt.edu/~rada/CSCE3110.
[6] Rajeev Motwani and Prabhakar Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
[7] Prabhakar Raghavan, AMS talk on Randomized Algorithms, Stanford University.
[8] Lecture Notes in Computer Science, pages 90–106. Springer, 2001. R. Johnsonbaugh. Discrete Mathematics. Prentice Hall, 4th edition, 1997.
[9] R. Karp. An introduction to randomized algorithms. Discrete Applied Mathematics, 34:165–201, 1991.
[10] D. Karger. Global min-cuts in RNC and other ramifications of a simple min-cut algorithm. In ACM–SIAM Symposium on Discrete Algorithms, volume 4, pages 21–30, 1993.
[11] L. Khachian. A polynomial algorithm for linear programming. In Doklady Akad. Nauk USSR 224, volume 5, pages 1093–1096. 1979 (in Russian), Translation Soviet. Math. Doklady 20, 91–194.
[12] V. Klee and G. Minty. How good is the simplex algorithm? In Inequalities III, pages 159–175. Academic Press, 1972.
[13] E. Kushilevitz and N. Nisan. Communication Complexity. Cambridge University Press, 1997.