A Review on Succinct Dynamic Data Structure

Pradhumn Soni[1], Mani Butwall[2]

[1] *Assistant Professor, Computer Science and Engineering, Mandsaur Institute of Technology, Madhya Pradesh, India*

[2] *Assistant Professor, Computer Science and Engineering, Mandsaur Institute of Technology ,Madhya Pradesh, India*

---

1. Abstract:- Over the years, computer scientists have studied many different data structures for representing and manipulating data within computer main and secondary memory. Ways of representing and efficiently manipulating data are central to many computer programs. These investigations have shown, both in practice and in theory, that the choice of data structures often has a considerable effect on algorithmic performance. This review paper addresses this issue by analyzing space efficient geometric data structures. The study also represent the detail description of succinct data structure which can be useful for space and time efficient applications.

2.Keywords:- Rank,Select,bit vectors.

3.Introduction:- A succinct data structure is a data structure which uses an amount of space that is "close" to the information-theoretic lower bound, but (unlike other compressed representations) still allows for efficient query operations. The concept was originally introduced by Jacobson to encode bit vectors, (unlabeled) trees, and planar graphs.

Suppose that Z is the information-theoretical optimal number of bits needed to store some data. A representation of this data is called:

succinct if it takes $Z + o(Z)$ bits of space

For example, a data structure that uses $2Z$ bits of storage is compact, $Z + \sqrt{Z}$ bits is succinct.[8].

- **Rank and Select:-**

Succinct indexable dictionaries, also called *rank/select* dictionaries, form the basis of a number of succinct representation techniques, including binary trees, k-ary trees and multi sets, as well as suffix

trees and arrays.[3] The basic problem is to store a subset S of universe $U = [0 \dots n) = \{0, 1, \dots, n-1\}$, usually represented as a bit array $B[0 \dots n)$ where $B[i] = 1$ iff $i \in S$. An indexable dictionary supports the usual methods on dictionaries (queries, and insertions/deletions in the dynamic case) as well as the following operations:

$$\mathbf{rank}_q(x) = |\{k \in [0 \dots x] : B[k] = q\}|$$

$$\mathbf{select}_q(x) = \min\{k \in [0 \dots n) : \mathbf{rank}_q(k) = x\}$$

for $q \in \{0, 1\}$.

In other words, $\mathbf{rank}_q(x)$ returns the number of elements equal to q up to position x while $\mathbf{select}_q(x)$ returns the position of the x-th occurrence of q..[8].

There is a simple representation which uses $n + o(n)$ bits of storage space (the original bit array and an $o(n)$ auxiliary structure) and supports **rank** and **select** in constant time.

Given a bit vector B

rank_1(i) = # 1's up to position i in B

select_1(i) = position of the i-th 1 in B

(similarly rank_0 and select_0)

- Choosing b = $(\log m)^2$, and s = (1/2)log n makes the overall space to be O(m loglog m / log m) (= o(m)) bits.
- Supports rank in constant time.

Select can also be supported in constant time using an auxiliary structure of size O(m loglog m / log m) bits.

Lower Bounds for Rank and Select:-

- If the bit vector is read-only, any index (auxiliary structure) that supports rank or select in constant time (in fact in O(log m) bit probes) has size Ω(m loglog m / log m)
- Bit-vector (BV):

  i) space used be *m* + *o*(*m*) bits.
- Bit-vector *index* :

  i) bit-sequence stored in read-only memory

  ii)index of *o*(*m*) bits to assist operations

- Compressed bit-vector: with n 1's

  i)space used should be $B(m,n) + o(m)$ bits.

Applications :-

- Potential applications where succinct data structures are used :-
  i) memory is limited: small memory devices like PDAs, mobile phones etc.
  ii) massive amounts of data: DNA sequences, geographical/astronomical data, search engines etc.

Examples:-

- Trees, Graphs
- Bit vectors, Sets
- Dynamic arrays
- Text indexes
  iii) suffix trees/suffix arrays etc.
- Permutations, Functions
- XML documents, File systems (labeled, multi-labeled trees)
- DAGs and BDDs

4.Background:- Rank & select data structures are one of the fundamental building blocks for many modern succinct data structures. Asymptotically, these data structures use only the minimum amount of space indicated by information theory. With the continued growth of massive-scale information services, taking advantage of the space efficiency of succinct data structures is becoming increasingly attractive in practice [7]. The aim is to analyze the data structures that are asymptotically optimal with respect to operation times, but whose space usage is optimal to within lower –order additive terms. The succinct data structure given succinct partial sum data structure can perform sum., select and update in optimal O(lg n/lg lg n) time.[6]. Perhaps the single most fundamental class of data structuring problems broadly involve maintaining or manipulating a (possibly changing) set S of keys (which are linearly ordered). These include basic problems such as sorting, searching and priority queues. Most BSc courses in Computer Science teach classical solutions to these problems such as quick sort, radix sort, balanced search trees and binary heaps. However, there is now a compelling reason to re-assess the performance of these nearly 40-year-old solutions—the divergence between the computational models on which these solutions were developed, and real-life CPU architectures. This divergence can cause asymptotic analysis—the foundation of modern algorithm theory—to give

incorrect performance predictions, even for very large problem instances.[1]Many applications such as spatial databases, computer graphics and geographic information systems store and process geometric data sets that typically consist of point coordinates. In other applications such as relational databases and data mining applications, the given data are essentially sets of records whose fields are values of different properties, and thus can be modeled as geometric data in multidimensional space. Thus the study of geometric data structures which can potentially be used to preprocess these data sets so that various queries can be performed quickly is critical to the design of a large number of efficient software systems.[4].

Table 1 : Comparative study of various data structures

S.No.	Type	Key points
1.	Succinct Data Structures	• It is a representation of the underlying combinatorial object that uses an amount of space "close" to the information theoretic lower bound. • Efficient algorithm for navigation,search,insertion and deletion operations. • Introduced by Jacobson. • Encode bit vectors,trees • 2n+O(n) bits is used to represent the n node arbitrary binary tree.
2.	Implicit Data Structures	• It is a data structure that uses very little memory besides the actual data elements. • It is called implicit because most of the structure of elements is expressed implicitly by their order. • Space efficient • It can mean O(1) to O(log n) extra space. • Designed to improve main memory utilization • Examples are:- Heap and Beap
3.	Compressed Data	• It refers to a data structure whose operations are roughly as fast as those of a conventional data structure but whose size

	Structures	can be substantially smaller. • It is highly dependent upon the entropy of the data being represented. • Important examples include suffix array and the FM-index,both of which can represent an arbitrary text of characters T for pattern matching.
4.	Search Data Structures	• Aloow the efficient retrieval of specific items from a set of items,such as specific record from a database. • Allow faster retrieval. • Limited to queries of some specific kind.
5.	Persistent Data Structures	• Which always preserve the previous version of itself when it is modified. • Efficiently immutable, as their operations do not update the structure in place, but instead always committed to persistent storage such as disk. • Particularly common in logical and functional programming. • While persistence can be achieved by simple copying, this is inefficient in time and space, because most operations make only small changes to a data structure. • Examples are singly-linked list or cons-based list. • Many common reference based data structures, such as red-black trees and ques,can easily be adapted to create a persistent version.
6.	Concurrent Data Structures	• It is a particular way of storing and organizing data for access by multiple computing threads(or processes) on a computer. • Used in Multiprocessor computer architectures. • Usually reside in an abstract storage environment, though this memory may be physically implemented as either a “tightly coupled” or a distribute collection of storage modules.

5. Drawbacks of Standard Representation:-

a. Standard representations of trees support very few operations. To support other useful queries, they require a large amount of extra space.
b. In various applications, one would like to support operations like "subtree size" of a node, "least common ancestor" of two nodes, "height", "depth" of a node, "ancestor" of a node at a given level etc.
c. The space used by the tree structure could be the dominating factor in some applications.
    i) Eg. More than half of the space used by a standard suffix tree representation is used to store the tree structure.
    ii) "A pointer-based implementation of a suffix tree requires more than 20n bytes. A more sophisticated solution uses at least 12n bytes in the worst case, and about 8n bytes in the average. For example, a suffix tree built upon 700Mb of DNA sequences may take 40Gb of space."

6.Different Approach:-

a. If we group k nodes into a block, then pointers with the block can be stored using only lg k bits.
b. For example, if we can partition the tree into n/k blocks, each of size k, then we can store it using $(n/k) \lg n + (n/k) k \lg k = (n/k) \lg n + n \lg k$ bits.
c. A careful two-level `tree covering'
d. method achieves a space bound of $2n+o(n)$ bits.

7.Conclusion:- Though the savings in storage is at least a significant constant factor, the usefulness of the succinct representation is questionable; the key strategy is to use succinct data structures to either achieve improvement upon previous results in terms of running time, or to reduce space usage by non-constant factors.

Although one may argue that disk space is no longer a problem and we should not concern ourselves with improving space utilization, but succinct or implicit data structures are designed to improve main memory utilization. Hard disk or any other means of large capacity, I/O devices, are order of magnitudes slower than main memory. Hence, the higher percentage of a task can fit in buffers in main memory the less dependence is on I/O devices. Hence, if a larger chunk of an

succinct data structure fits in main memory the operations performed on it can be faster even if the asymptotic running time is not as good as its space-obvious counterpart. The review study shows different data structures with its applications and memory and time space efficiency, which can be useful for developing any architecture or application. It is also shown that succinct data structures perform better for the applications where time and space efficiency is required.

8. References:-

[1]" New Paradigms in Data Structure Design: Word-Level Parallelism and Self-Adjustment" EPSRC Grant GR/L 92150: Final Report Rajeev Raman Department of Mathematics and Computer Science University of Leicester

[2] Simon Gog1_ and Matthias Petri2 "Optimized Succinct Data Structures for Massive Data" Softw. Pract. Exper. 0000; 00:1–28 Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe.1Department of Computing and Information Systems, The University of Melbourne, VIC, 3010, Melbourne, Australia 2School of Computer Science and Information Technology, RMIT University, VIC, 3001, Melbourne, Australia

[3] Ankur Gupta1, Wing-Kai Hon2, Rahul Shah1, and Je_rey Scott Vitter1 "A Framework for Dynamizing Succinct Data Structures?" 1 Department of Computer Sciences, Purdue University, West Lafayette, IN 47907{2066, USA (fagupta, rahul, jsvg@cs.purdue.edu).
2 Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan (wkhon@cs.nthu.edu.tw).

[4] Meng He "Succinct and Implicit Data Structures for" Computational Geometry,Faculty of Computer Science, Dalhousie University, Halifax, NS, B3H 4R2, Canada,
mhe@cs.dal.ca

[5] James King "Succinct Data Structures for Tree Adjoining Grammars"
Department of Computer Science University of British Columbia 201-2366 Main Mall Vancouver, BC, V6T 1Z4, Canada king@cs.ubc.ca

[6] Rajeev Raman,Venkatesh Raman,S. Shrivastva Rao "Succinct Dynamic Data Structures" Department of Mathematics and Computer Science,University of Liecester,Liecester LE1 7RH UK, Institute of Mathematical Science,Chennai,India.

[7] Dong Zhou, David G. Andersen, Michael Kaminskyy "Space-Efficient, High-Performance Rank & Select Structures on Uncompressed Bit Sequences" The final version appears in Proceedings of the 12th International Symposium on Experimental Algorithms (SEA 2013), Rome, Italy, June 2013 Carnegie Mellon University, yIntel Labs.

[8] https://en.wikipedia.org/