'Ads' Algorithm for Subset Sum Problem

Adarsh Kumar Verma
Student,Galgotias College Of Engineering and Technology
Greater Noida, G. B. Nagar
201306, India

ABSTRACT
The Subset Sum Problem is an important problem in Complexity Theory, Bin Packing and Cryptography. The Subset Sum Problem is NP Complete. In this paper we are introducing a new technique to find the solution of Subset Sum Problem. There are many algorithms based on greedy approach and lattice based reduction and many more approaches has been suggested earlier but suggested approach is based on the simple mathematics concept and binary search.

General Terms
Algorithm, NP Complete.

Keywords
Subset Sum, binary search.

1. INTRODUCTION
Subset Sum problems are special case of binary knapsack problems. One interesting special case of subset sum is the Partition Problem, in which half of the sum of all elements in the set. In the SSP we have to find a subset 's' of the given set S={ S_1, S_2, S_3, S_4,……S_n } where the elements of the set are n positive integers in such a manner such that subset s ∈ S and the elements are in the increasing order and sum of the elements of subset s is equal to some positive integer X.

The current upper bound for Subset Sum is apparently O($2^{n/2}$) when size of the input set (denoted *n*) is used as the complexity parameter . When the maximum value in the set (denoted *m*) is used as the complexity parameter, dynamic programming can be used to solve the problem in O(m^3) time. The SSP is known to be NP Complete [1] and hence difficult problem to solve generally. Cook, Karp and others, defined such class of problems as NP Hard problem [2]. Some of the NP Hard problems include Travelling Salesman Problem (TSP), Boolean Satisfiability Problem, Knapsack Problem, Hamiltonian Path Problem, Post Correspondence Problem (PCP), and Vertex Cover Problem (VCP). There are several ways to solve SSP in exponential and polynomial time. A naive algorithm with time complexity O($n2^n$) solves SSP by iterating all the possible subsets and each for its subset comparing its sum with target X. A better algorithm proposed in 1974 using the Horowitz and Sahni decomposition scheme which achieves time O($n2^{n/2}$). If the target T is relatively small then there exist dynamic programming algorithms that can run much faster. A classic Pseudo-Polynomial algorithm Bellman Recursion solves SSP in both time and space O(nc). And there are many other algorithms, for example Ibam and Kim [3] developed a fully polynomial approximation scheme for the SSP in 1975. It was improved upon by Lawler [4] and Lam by Martello and Toth [5]. Martello and Toth reported very good results for several approximation schemes in their survey and experimental analysis [6].

In this paper we'd study about a simple SSP solution based on arithmetic and binary search with O (n^2logn) time complexity and with linear space complexity.

2. BACKGROUND

2.1 Sorting
Sorting is any process of arranging items in some sequence, which can be done by sorting algorithms. A sorting algorithm is an algorithm that puts elements of a list in a certain order. In computer science, sorting is one of the most extensively researched subjects because of the need to speed up the operation on thousands or millions of records during a search operation. For subset sum problem we need to arrange the elements in non-decreasing order, then subset sum problem can be solved by 'Ads' algorithm.

Sorting of the list can be done by various algorithms of sorting and the complexity depends on the type of algorithm we are using. If there are less elements in the list then we can use simple sorting algorithm like Bubble sort with worst case complexity O (n^2) or we can use efficient quick sort with average case complexity O(n logn) or merge sort with complexity O (n log n) for more numbers.

2.2 Binary Search
Binary Search or half-interval search algorithm search the specified value within the sorted array. It is based on divide and conquer approach. Binary search will require far fewer comparisons than linear search, if the list to be searched contains more than a dozen elements. But it imposes the requirement that the list should be sorted. In each step, the algorithm compares the input value with the key value of the middle element. If these keys match, then matching element has been found. The position and the key is returned. Otherwise, if the sought key is less than the middle elements key, then algorithm repeat its action on the sub array to the left of the middle element, or if the input key is greater than the middle element's key then algorithm repeat its action to the right sub array. If the remaining array to be searched is reduced to zero, then key not found in the array and a "Key not found " indication is returned.

Binary search halves the number of items to check with each iteration, so locating an item (or determining its absence) takes logarithmic time. The worst case performance of binary search is O (log(n)) and best case is O(1), And the worst space complexity is O(1).

3. THE COMPLETE ALGORTITHM
With the Subset Sum problem, however, we do not find a mutual dependence between the number of objects in the set and the Target X. But using the basic concepts of mathematics we found a method to solve it. So, the procedure is, first we have to sort the given list to solve it by Ads algorithm or we

can say the proposed algorithm is only for sorted array or list. Suppose if list is not already sorted then we can sort it by efficient sorting algorithms. After sorting we will apply Ads algorithm for solving subset sum problem.

The algorithm uses simple mathematics, like, if sum of two numbers A, B is C then we can find B by subtracting it from C, such that B = C – A. Similarly if sum of three numbers A, B, C from the set { A, B, C, D } is X then we can also find the C by subtracting (A + B) from X such that C = X – (A + B). And we can search C by binary search from the sorted list. We will apply binary search in the list from the element 3 of the list, because we have already taken A, B. Searching will be in the remaining set { C, D }. Hence the set will be { A, B, C} whose sum is X.

3.1 Pseudo code for 'Ads' algorithm

Input : Set of n sorted positive integers e = { e_1, e_2, e_3,…e_n } where e_1, e_2, e_3, e_4,…….e_n are in non-decreasing order.

Target positive integer X.

ads (e, X)

```
1.  for ( i = 0 ; i < n-1 ; i++)
2.      sum = 0
3.      for ( j = i ; j < n - 1 ; j++ )
4.          sum = sum + e [ j ]
5.          c = X – sum
6.          if ( c > e [ j ] )
7.              flag = binarySearch ( c, j, n-1 )
8.              if  flag = 1
9.                  " subset found with target sum X"
10.                 for ( k = i, l=0 ; k < = j ; k++, l++ )
11.                         s [ l ] = e [ k ]
12.                 l++
13.                 s [ l ] = c
14.                 display( s )
15.         else
16.             break
17. if flag = - 1
18. " no subset found"
```

In Pseudo code binarysearch() searches the difference of the list elements from the target X, display() function displays the subset found with target sum X.

3.2 Pseudo code for binary search :

Input: array e , lower bound, upper bound

binarySearch (c, j, n-1)

```
1.  low = j,      high= n – 1
2.  while  ( low < = high )
3.          mid = ( low + high ) / 2
4.          if ( c > = e[ mid ] )
5.              if ( c = mid )
6.                  return ( 1 )
7.              else
8.                  low = mid + 1
9.
10.         else
11.             high = mid - 1
12. return( - 1 )
```

3.3 Analysis of 'Ads' Algorithm

It is very convenient to classify algorithms based on the relative amount of time or relative amount of space they require and specify the growth of time/space requirements as a function of the input size.

3.3.1 Time complexity

Running time of the algorithm as a function of the size of input. In computing time complexity, one good approach to count primitive operations. Some examples of primitive operations are assigning value to a variable, indexing into an array, calling a method, performing an arithmetic operation, returning from a method.

The time complexity of 'Ads' algorithm depends on the two for loops and the method binary search, first for loop runs 0 to (n – 1) in i, similarly second for loop runs j = i to (n – 1). Hence due to two for loops primitive operations will occur (n – 1) x (n – 1), that will be O (n^2). But due to binary search inside the for loops it will be O (n^2 log (n)).

3.3.2 Space Complexity:

Some forms of analysis of algorithms could be done on how much space an algorithm needs to complete its task. The space complexity analysis was critical in the early days of computing when storage space on the computers was limited. When considering space complexity, algorithms are divided into those that need extra space to do their work.

The space complexity of Ads algorithm is linear O (n) due to single array required for it's execution.

3.4 Computational Results

Problem : - Given a set e = { 2, 3, 6, 7, 10 } and Target sum X = 13. Find a subset whose sum is equal to X.

Solution :- Number of elements in set n = 5, we are considering the array from 0 to 4.

Pass 1 : i = 0

sum=0 ,

1. j = i = 0, e[0] = 2

   sum = sum + e[j] = 0 + 2 = 2

   c = X – sum = 13 – 2 = 11

   check if (11 > 2) , yes

   flag = binarySearch(c)

   binarySearch(11) from index 1 to 4

   flag = -1

2. j = 1, e[1]= 3

   sum = sum + e[j] = 2 + 3 = 5

   c = X – sum = 13 – 5 = 8

   check if (8 > 3), yes

   flag = binarySearch(c)

   binarySearch(8) from index 2 to 4

   flag = - 1

3. j = 2 , e[2] = 6

   sum = sum + e[j] = 5 + 6 = 11

   c = X – sum = 13 – 11 = 2

   check if (2 > 3), no

   break.

Pass 2 : i = 1

sum = 0

1. j = i = 1, e[1]= 3

   sum = sum + e[j] = 0 + 3 = 3

   c = X – sum = 13 – 3 = 10

   check if (10 > 3), yes

   flag = binarySearch(c)

   binarySearch(10) from index 2 to 4

   flag = 1

   hence, subset found S = { 3, 10 }

2. j = 2, e[2] = 6

   sum = sum + e[j] = 3 + 6 = 9

   c = X – sum = 13 – 9 = 4

   check if (4 > 6), no

   break

Pass 3 : i = 2

sum = 0

1. j = i = 2, e[2] = 6

   sum = sum + e[j] = 0 + 6 = 6

   c = X – sum = 13 – 6 = 7

   check if (7 > 6), yes

   flag = binarySearch(c)

   binarySearch(7) from index 3 to 4

   flag = 1

   hence, subset found S = { 6, 7 }

2. j = 3, e[3] = 7

   sum = sum + e[j] = 6 + 7 = 13

   c = X – sum = 13 – 13 = 0
   check if (0 > 6), no
   break ,

Pass 4 : i = 3

sum = 0

1. j = i = 3, e[3] = 7

   sum = sum + e[j] = 0 + 7 = 7

   c = X – sum = 13 – 7 = 6

   check if (6 > 7), no

   break.

Hence by suggested algorithm we get two subsets whose sum is X = 13 and subsets are { 3, 10 } and { 6, 7 }.

4. CONCLUSION AND FUTURE SCOPE

We have presented a simple mathematics based solution for Subset Sum problem with time complexity O ($n^2 \log n$) and space complexity O (n). The Subset Sum problem is NP-complete problem and that can execute in polynomial time. It can be argued that Subset Sum problem is easier than the other NP-complete problems, based on algorithms that solve the problem in sub-exponential time. We identify a generic construction of cryptosystems based on the Subset Sum Problem. The suggested approach can be used for the implementation of SSP-based cryptosystems and application of these cryptosystems in defining an efficient RFID (Radio-Frequency Identification) security and privacy solutions.

5. ACKNOWLEDGMENTS

I am very thankful to Monica Batham (Student, Computer Science and Engineering Department, Galgotias college of Engineering and Technology) for programming suggestions.

6. REFERENCES

[1] M. R. Garey& D. S. Johnson, Computers and Intractibility: A Guide to the theory of NP Completeness, W. H. Freeman and Company, New York (1979).

[2] Harsh Bhasin and NehaSingla, “Harnessing Cellular Automata and Genetic Algorithms to solve Travelling Salesman Problem”.

[3] O. H. Ibarra and C. E. Kim, Fast approximation algorithms for knapsack and sum of subset problem, journal of the ACM, 22, 463-468(1975).

[4] E. L. Lawler, Fast approximation algorithms for Knapsack problems, *Mathematics of operation research*, 4,339-356 (1979)

[5] S. Martello and P. Toth, Worst case analysis of greedy algorithms for the subset sum problem, *Mathematical Programming*, 28,198-205 (1984).

[6] S. Martello and P. Toth, Approximation schemes for the subset-sum problem: Survey and experimental analysis, *European Journal of Operational Research*, 22,56-69 (1985)