**Batch: C1 Roll No.: 16010122221**

**Experiment No. 8**

**Grade: AA / AB / BB / BC / CC / CD /DD**

**Signature of the Staff In-charge with date**

|  |
| --- |
| **Title: Implementation of sum of subset Algorithm** |

**Objective:** To learn the Backtracking strategy of problem solving for Sum of subset

**CO to be achieved:**

|  |  |
| --- | --- |
| CO 2 | Analyze and solve problems for divide and conquer strategy, greedy method, dynamic programming approach and backtracking and branch & bound policies. |

**Books/ Journals/ Websites referred:**

1. **Ellis horowitz, Sarataj Sahni, S.Rajsekaran,” Fundamentals of computer algorithm”, University Press**
2. **T.H.Cormen ,C.E.Leiserson,R.L.Rivest and C.Stein,” Introduction to algortihtms”,2nd Edition ,MIT press/McGraw Hill,2001**

**Pre Lab/ Prior Concepts:**

Data structures, Concepts of algorithm analysis

**Historical Profile:**

Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K. We are considering the set contains non-negative values. It is assumed that the input set is unique (no duplicates are presented).

One way to find subsets that sum to K is to consider all possible subsets. A [power set](http://en.wikipedia.org/wiki/Power_set) contains all those subsets generated from a given set. The size of such a power set is 2N.

***Input:***

A vector X={x1,x2… xn} for all n elements in the set where Xi=0 (element not added) or xi=1 (element added in the solution tuple).

***Output:***

Summation of the chosen numbers must be equal to given number M and one number can be used only once.

BACKTRACKING CONDITION

Diagram, letter, schematic

Description automatically generated

**New Concepts to be learned:**

Application of algorithmic design strategy to any problem, Backtracking method of problem solving Vs other methods of problem solving problem sum of subset and its applications.

**Algorithm:**

Algorithm sumOfSub(s, k, r)

{//It is assumed w[1]<=m and Sigma(i=1 to m)w[i]>=m

//generate the left child. Note: s+w(k)<=M since Bk-1 is true.

X{k]=1;

if (S+W[k]=m) then write(X[1:k]); //Subset found. there is no recursive call here

as W[j]>0,1<=j<=n.

else if (S+W[k]+W[k+1]<=m) then sumOfSub(S+W[k], k+1,r- W[k]); //moving to

next sub-problem.

Similarly, assume the array is presorted and we found one subset. We can generate

next node excluding the present node only when inclusion of next

node satisfies the constraints.

if ((S+ r- W[k]>=m)and (S+ W[k+1]<=m)) then//generate right {

//child and those satisfying 2 bounding functions

X{k]=0;

sumOfSub (S, k+1, r- W[k]);

}}

**Implementation(Code):**

**#include <iostream>**

**#include <vector>**

**#include <algorithm>**

**using namespace std;**

**void sum\_of\_sub(vector<int>& X, int M, vector<int>& subset, int current\_sum, int k, int remaining\_sum) {**

**if (current\_sum == M) {**

**for (int num : subset) {**

**cout << num << " ";**

**}**

**cout << endl;**

**return;**

**}**

**if (current\_sum + X[k] <= M) {**

**subset.push\_back(X[k]);**

**sum\_of\_sub(X, M, subset, current\_sum + X[k], k + 1, remaining\_sum - X[k]);**

**subset.pop\_back(); // backtrack**

**}**

**if (current\_sum + remaining\_sum - X[k] >= M && current\_sum + X[k + 1] <= M) {**

**sum\_of\_sub(X, M, subset, current\_sum, k + 1, remaining\_sum - X[k]);**

**}**

**}**

**void sum\_of\_subset(vector<int>& X, int M) {**

**int n = X.size();**

**vector<int> subset;**

**int current\_sum = 0;**

**int remaining\_sum = 0;**

**for (int num : X) {**

**remaining\_sum += num;**

**}**

**sum\_of\_sub(X, M, subset, current\_sum, 0, remaining\_sum);**

**}**

**int main() {**

**vector<int> X;**

**int num;**

**cout << "Enter the list of integers (separated by spaces): ";**

**while (cin >> num) {**

**X.push\_back(num);**

**if (cin.get() == '\n') {**

**break;**

**}**

**}**

**int M;**

**cout << "Enter the target sum (M): ";**

**cin >> M;**

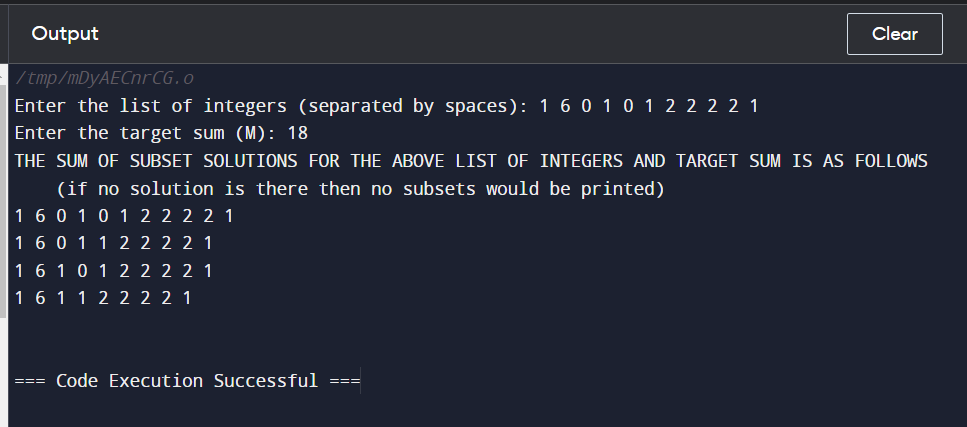
**cout << "THE SUM OF SUBSET SOLUTIONS FOR THE ABOVE LIST OF INTEGERS AND TARGET SUM IS AS FOLLOWS (if no solution is there then no subsets would be printed)" << endl;**

**sum\_of\_subset(X, M);**

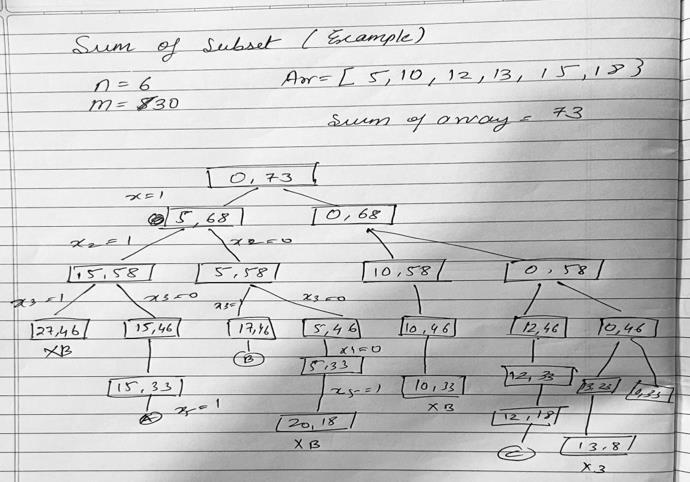
**return 0;**

**}**

**Output:**



**Example sum of subset Problem along with state space tree:**



**Analysis of Backtracking solution for sum of subset Problem:**

The time complexity of the backtracking solution is exponential, O(2^N), where N is the number of elements in the input set. In this case, there are 11 elements in the input set, so the time complexity is O(2^11), which is manageable.

**Conclusion:**

Through this experiment we implemented sum of subset problem using backtracking strategy. Subset sum problem is to find subset of elements that are selected from a given set whose sum adds up to a given number K. We are considering the set contains non-negative values. It is assumed that the input set is unique.