CSE 462 | ALGORITHM ENGINEERING SESSIONAL

The Subset Sum Problem

Group 6

Group Leader - Sifat Ishmam Parisa(1505016)
Farhanaz Farheen (1505013)
Salman Shamil (1505021)
Kazi Sajeed Mehrab (1505025)
Syeda Nahida Akter (1505027)

Bangladesh University of Engineering and Technology

INTRODUCTION

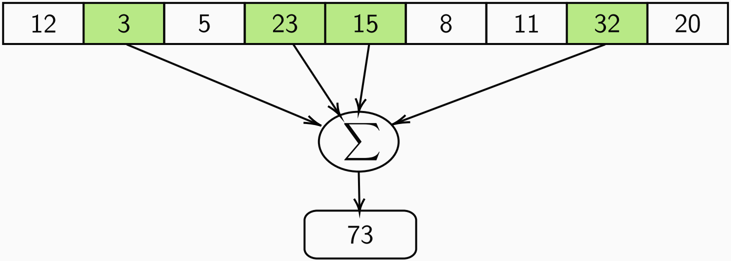
The Subset Sum Problem

Let's assume we're given a set of integers. Can we find a subset that sums up to a given target integer?

Input Set:

12	3	5	23	15	8	11	32	20

Selection of Subset:



The Subset Sum Problem

Formal definition of the problem:
Given a Multiset of integers, $S = \{x_1, x_2, x_3, \cdots, x_n\}$ and a target sum W, does there exist a subset $S' \subseteq S$ such that $\sum_{x \in S'} x = W$?

EXACT EXPONENTIAL ALGORITHMS

Exact Exponential Algorithms

Exact algorithms are algorithms that always solve an optimization problem to optimality.

Unless $\mathsf{P} = \mathsf{NP}$, an exact algorithm for an NP-hard optimization problem cannot run in worst-case polynomial time.

BRUTE-FORCE ALGORITHM

Brute-Force Algorithm

A brute-force algorithm is a method of problem-solving in which every possible scenario is examined and the best one is chosen.

For the subset sum problem, a brute-force algorithm would require:

- **Finding all possible subsets of S.** (Total $= 2^n$)
- **Finding the sum of the elements of every subset.** ($\forall S' \subseteq S$, find $\sum_{x \in S'} x$)
- **Checking if any of these sums equals target sum W.** (Check if $\exists S' \subseteq S$ for which $\sum_{x \in S'} x = W$?)

Brute-Force Algorithm

$$S = \begin{array}{|c|c|c|}\hline 1 & 3 & 6 \\ \hline\end{array} \qquad W = \boxed{10}$$

Brute-Force Algorithm

$S = \begin{array}{|c|c|c|}\hline 1 & 3 & 6 \\ \hline\end{array}$ $\qquad W = \boxed{10}$

$s_0 = \boxed{}$

Brute-Force Algorithm

$S = \boxed{1 \mid 3 \mid 6}$ $W = \boxed{10}$

$s_1 = \boxed{1}$

$s_0 = \boxed{}$ $s_2 = \boxed{3}$

$s_3 = \boxed{6}$

Brute-Force Algorithm

$S = \boxed{1 \mid 3 \mid 6}$ $\quad W = \boxed{10}$

$s_0 = \boxed{}$

$s_1 = \boxed{1}$ $\quad s_4 = \boxed{1 \mid 3}$

$s_2 = \boxed{3}$ $\quad s_5 = \boxed{1 \mid 6}$

$s_3 = \boxed{6}$ $\quad s_6 = \boxed{3 \mid 6}$

Brute-Force Algorithm

$S = \boxed{1 \mid 3 \mid 6}$ $\quad W = \boxed{10}$

$s_1 = \boxed{1}$ $\quad s_4 = \boxed{1 \mid 3}$

$s_0 = \boxed{}$ $\quad s_2 = \boxed{3}$ $\quad s_5 = \boxed{1 \mid 6}$ $\quad s_7 = \boxed{1 \mid 3 \mid 6}$

$s_3 = \boxed{6}$ $\quad s_6 = \boxed{3 \mid 6}$

Brute-Force Algorithm

$S = \boxed{1} \boxed{3} \boxed{6}$ $\quad W = \boxed{10}$

$s_0 = \boxed{}$ 0

$s_1 = \boxed{1}$ 1

$s_2 = \boxed{3}$ 3

$s_3 = \boxed{6}$ 6

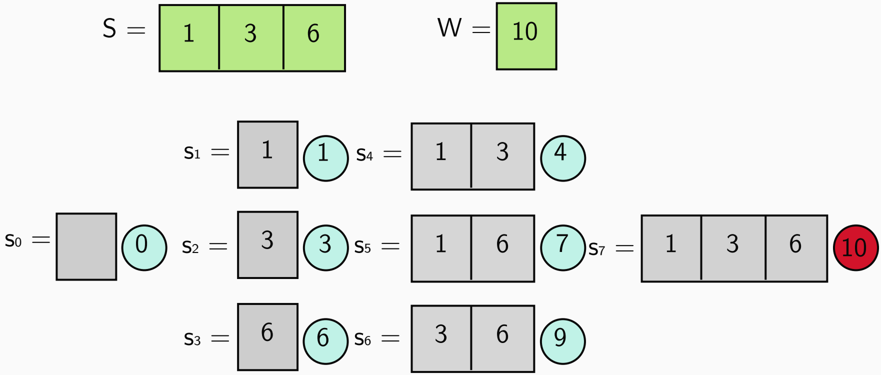
$s_4 = \boxed{1} \boxed{3}$ 4

$s_5 = \boxed{1} \boxed{6}$ 7

$s_6 = \boxed{3} \boxed{6}$ 9

$s_7 = \boxed{1} \boxed{3} \boxed{6}$ 10

Brute-Force Algorithm



Brute-Force Algorithm

Algorithm 1: BruteForceSubsetSum(S,i,r,W)

Input: Set S, Index of element (i), Remaining sum(r), Target (W)

Output: A decision on whether there is a subset whose sum is W.

```
if i == 0 then
    return (r == 0);
else
    if BruteForceSubsetSum(S,i-1,r,W) == true then
        return true ;
    else if (r − w_i ≥ 0) then
        return (BruteForceSubsetSum(S,i-1,r-w_i,W) == true) ;
    else
        return false ;
    end
end
```

Brute-Force Algorithm Time-Complexity

S	1	3	6

Target sum = 10

Subsets of ***S***

S_0	$\{ \}$		S_1	$\{1\}$	1
S_2	$\{3\}$	3	S_3	$\{6\}$	6
S_4	$\{1, 3\}$	4	S_5	$\{1, 6\}$	7
S_6	$\{3, 6\}$	9	S_7	$\{1, 3, 6\}$	10

Here, n = 3. By checking all $2^3 = 8$ subsets, the solution is S_7

- For, input set of size n, there are 2^n possible subsets.
- Each subset can be checked in $O(n)$ time.
- Overall, complexity of brute-force algorithm is exponential, $O(2^n n)$

BETTER EXPONENTIAL ALGORITHMS

Better Exponential Algorithms

- Backtracking Algorithm
- Branch and Bound Algorithm
- Dynamic Programming Algorithm

SUBSET SUM USING BACKTRACKING ALGORITHM

Backtracking

- Search the solution space tree in depth first manner.
- Upon reaching a candidate that cannot be a valid solution, backtrack.
- Search tree for subset problem is a binary tree of 2^n leaves mapping to 2^n subsets. (Leaves represent members of solution space.)
- With effective bounding functions, large instances can be solved.
  Examples of Bounding Functions:
  - When sum of a node equals target sum, terminate.
  - When sum of a node exceeds target sum, backtrack.

Backtracking Time-complexity

- There are 2^n leaf nodes.
- Forward and backward moves through the tree takes $O(1)$ time.
- So, Subset Sum Backtracking with Bounding Functions has complexity of $O(2^n)$
- Better than brute force solution that takes $O(2^n n)$.

BRANCH AND BOUND ALGORITHM

Branch and Bound Algorithm

- Let, $S = x_1, x_2, ...x_n$ such that $x_1 \geq x_i;\ for\ i = 2, 3...n$ and W is the target sum.
- There are 2^n possible subsets.
- There is a binary tree rooted at the empty subset with starting level -1 for convenience.
- Each node in level i is divided into a left and right branch, to either exclude x_i from the subset so far, or include x_i in the subset so far.

Branch and Bound Algorithm

- For each node, calculate two values: Sum So Far, SSF and Maximum Potential Sum, $MPS = SSF + \sum_{j=i+1}^{n-1} x_j$
- If $SSF > W$ then there is no node beneath current node that can add up to W. No point in creating those nodes.
- If $MPS < W$ then there is no node beneath that node that can add up to W. No point in creating those nodes.
- If $SSF = W$ or $MPS = W$, then this subset should be recorded in the list of solutions.
- At a node on level i, when $SSF < W < MPS$ the potential for a solution exists and the two branches to exclude or include x_{i+1} should be created and investigated.
- Complexity: $O(2^n)$ [1]

Dynamic Programming Algorithm

Dynamic Programming Algorithm attempts to solve a problem by combining solutions of the sub-problems. [2]

Steps in Dynamic Programming Algorithm for Subset Sum problem:

- For set S (where $|S| = n$) and target sum W, make a table T with $n + 1$ rows and $W + 1$ columns.
- Populate first row with 0s, and the top left cell with 1.
- For every other cell T[i,j], copy the value of previous row T[i-1,j].
- Check if current column index (j) exceeds or equals value of ith element (w_i).
- If so, populate cell with $max(T[i-1, j], T[i-1, j-w_i])$.

Problem: Given a set $S = \{1, 3, 7, 9, 12\}$ and $W = 11$, is there a subset $S' \in S$, such that $\sum_{x \in S'} x = W$?

For $S' = \{1\}, sum = 1 < W$
$i = 1, j = 0, w_i = 1, j < w_i$
So, $T[i, j] = T[i-1, j]$
$T[1, 0] = T[0, 0] = 1$

	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	1	0	0	0	0	0	0	0	0	0	0	0
1	1											
3												
7												
9												
12												

Dynamic Programming

For $S' = \{1\}, sum = 1 < W$

$i = 1, j = 1, w_i = 1, j = w_i$

So, $T[i, j] = T[i-1, j] \mid\mid T[i-1, j-w_i]$

$T[1, 1] = T[0, 1] \mid\mid T[0, 0] = 1$

	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1										
3												
7												
9												
12												

Dynamic Programming

Complete table:

	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

As $T[5, 11] = 1$, there exists a subset $S' \in S$ where $\sum_{x \in S'} x = W = 11$.
Now to get the subset S' -

	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

Dynamic Programming

	0	1	2	3	4	5	6	7	8	9	10	11
∅	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

Dynamic Programming

	0	1	2	3	4	5	6	7	8	9	10	11
∅	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

So, $7 \in S'$, $S' = \{7\}$

	0	1	2	3	4	5	6	7	8	9	10	11
∅	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	⊗
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

$S' = \{7\}$

	0	1	2	3	4	5	6	7	8	9	10	11
$\emptyset$	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

Dynamic Programming

So, $3 \in S'$, $S' = \{7, 3\}$

	0	1	2	3	4	5	6	7	8	9	10	11
∅	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

$S' = \{7, 3\}$

	0	1	2	3	4	5	6	7	8	9	10	11
∅	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

So, $1 \in S'$, $S' = \{7, 3, 1\}$

	0	1	2	3	4	5	6	7	8	9	10	11
∅	1	$\otimes$	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	$\otimes$	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	$\otimes$
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

$S' = \{7, 3, 1\}$

	0	1	2	3	4	5	6	7	8	9	10	11
∅	1	0	0	0	0	0	0	0	0	0	0	0
1	1	1	0	0	0	0	0	0	0	0	0	0
3	1	1	0	1	1	0	0	0	0	0	0	0
7	1	1	0	1	1	0	0	1	1	0	1	1
9	1	1	0	1	1	0	0	1	1	1	1	1
12	1	1	0	1	1	0	0	1	1	1	1	1

Algorithm 2: DPSubsetSum(n,W)

Input: Number of elements (n), Target (W)

Output: A decision on whether there is a subset whose sum is W.

```
for j = 1 to W do T[0,j] = 0
T[0,0] = 1
for i = 1 to n do
    for j = 0 to W do
        T[i,j] = T[i-1,j]
        if j ≥ w_i then
            T[i,j] = max{T[i-1,j],T[i-1,j-w_i]}
    end
end
return T[n,W]
```

DPSubsetSum Algorithm Time Complexity

- Since the table formed has n+1 rows and W+1 columns, the algorithm runs in $O(nW)$.
- If W is represented in binary, its size is $log_2 W$.
- W is thus exponential in input size.
- But if input is given in unary, then $O(nW)$ is polynomial in size of input. So it is a pseudopolynomial algorithm.
- Better than Brute-Force approach that takes $O(2^n n)$

VARIATION OF SUBSET SUM PROBLEM

Polynomial Time Variation

SSP is solvable in polynomial time if the sequence $\{a_1, a_2, \cdots, a_n\}$ is restricted to be an arithmetic progression. [3]
The sequence can be specified concisely by the triple (a_1, n, j).

Expanded set from triple, $T = \{a, a + j, \cdots, a + (n - 1)j\}$

- If $S \subseteq T$ with $|S| = k$, let $SS_k = \sum_{s \in S} s$

Polynomial Time Variation

- If $S \subseteq T$ with $|S| = k$, let $SS_k = \sum_{s \in S} s$
- $SS_k = ka + mj, m \in Z^{\geq}$

Polynomial Time Variation

- If $S \subseteq T$ with $|S| = k$, let $SS_k = \sum_{s \in S} s$
- $SS_k = ka + mj, m \in Z^{\geq}$
- $c_k = MIN\{SS_k\}$ (Leftmost k elements)
  $d_k = MAX\{SS_k\}$ (Rightmost k elements)

Polynomial Time Variation

- For any k, SS_k can take any of the values from $c_k, c_k + j, c_k + 2j, \cdots, d_k - 2j, d_k - j, d_k$

Polynomial Time Variation

- For any k, SS_k can take any of the values from $c_k, c_k + j, c_k + 2j, \cdots, d_k - 2j, d_k - j, d_k$
- Increasing k causes c_k to increase

Polynomial Time Variation

- For any k, SS_k can take any of the values from $c_k, c_k + j, c_k + 2j, \cdots, d_k - 2j, d_k - j, d_k$
- Increasing k causes c_k to increase
- To achieve t from SS_k, $ka \equiv t \ (mod\ j)$ must hold

Polynomial Time Variation

- For any k, SS_k can take any of the values from $c_k, c_k + j, c_k + 2j, \cdots, d_k - 2j, d_k - j, d_k$
- Increasing k causes c_k to increase
- To achieve t from SS_k, $ka \equiv t\ (mod\ j)$ must hold
- Let k_1 be the lowest k with $d_k \geq t$

Polynomial Time Variation

- For any k, SS_k can take any of the values from $c_k, c_k + j, c_k + 2j, \cdots, d_k - 2j, d_k - j, d_k$
- Increasing k causes c_k to increase
- To achieve t from SS_k, $ka \equiv t \ (mod \ j)$ must hold
- Let k_1 be the lowest k with $d_k \geq t$
- k_1 can be found using binary search

Polynomial Time Variation

- For any k, SS_k can take any of the values from $c_k, c_k + j, c_k + 2j, \cdots, d_k - 2j, d_k - j, d_k$
- Increasing k causes c_k to increase
- To achieve t from SS_k, $ka \equiv t \ (mod\ j)$ must hold
- Let k_1 be the lowest k with $d_k \geq t$
- k_1 can be found using binary search
- Solution exists if and only if $c_{k_1} \leq t$

Polynomial Time Variation

Algorithm 3: PolynomialTimeSubsetSum(a,n,j,t)

Input: First element (a), increment value (j), in(r), Target (W)

Output: A decision on whether there is a subset whose sum is W.

Other polynomial time variation

- Low-Density Subset Sum Problem belongs to the class P.

Given a set $A = \{a_i : 1 < i < n\}$ of positive integers and positive integer M, find a subset of A that has sum equal to M.

The density of these a_i is defined by,

$$d = \frac{n}{log_2(max_i\ a_i)} \tag{1}$$

- [4] converts the problem to one of finding a particular short vector v in a lattice, and then uses a lattice based reduction algorithm to find v.
- Then for "almost all" problems of density $d < 0.645$, it is proved that lattice based reduction algorithm locates v in polynomial time.

Other polynomial time variation

- A sub-problem of the problem Subset Sum in which $s_1, \cdots, s_k$ are the members of increasing geometric progression belongs to the class P. [5]

CONCLUSION

Conclusion

- We have discussed exact exponential algorithms.
- We have presented Brute-force approach for the Subset sum problem.
- We have presented some better exact algorithms along with their time complexities.
- We have discussed the Dynamic Programming algorithm for Subset sum problem in detail.
- We have shown some polynomial time variations of Subset Sum problem.

REFERENCES

References I

- A. Shaheen and A. Sleit, "Comparing between different approaches to solve the 0/1 knapsack problem," *International Journal of Network Security*, vol. 16, pp. 1–10, 07 2016.
- R. Bellman *et al.*, "Notes on the theory of dynamic programming iv-maximization over discrete sets," *Naval Research Logistics Quarterly*, vol. 3, no. 1-2, pp. 67–70, 1956.
- J. R. Alfonsín, "On variations of the subset sum problem," *Discrete applied mathematics*, vol. 81, no. 1-3, pp. 1–7, 1998.
- J. C. Lagarias and A. M. Odlyzko, "Solving low-density subset sum problems," *J. ACM*, vol. 32, no. 1, p. 229–246, Jan. 1985. [Online]. Available: https://doi.org/10.1145/2455.2461

References II

. . , “Polynomial time algorithm for a sub-problem of subset sum with exponentially growing input,” in *International Journal” Information Theories and Applications”*. ITHEA–Publisher, 2018, pp. 32–37.

THANK YOU!