Sum of Subsets

...

Rebecca Lee, Kayte Chien, Jeremy Lesmana
Preferred Language: Java

What is the Sum of Subsets Problem?

- A decision problem where, given a set of non-negative integers and a value k, we must determine if there is a subset of the given set with a sum equal to k.
- Also known as the subset sum problem.

k = 9

The subset {4, 5} gives the sum of 9

Example by GeeksforGeeks

Solutions

- Many solutions to this problem already exist:
  - Recursion
  - Dynamic Programming
  - Memoization
  - Backtracking
- We will be covering two major methods:
  - Recursion
  - Dynamic Programming

Recursion: Description and Design Techniques

- The idea behind using the recursive approach is to generate all subsets recursively. This means in that in the worst case scenario our algorithm will generate all possible subsets.

Recursion: Pseudocode

```
static boolean subsetSum(int set[], int n, int sum)
   {
      if (sum == 0)
         return true;
      if (n == 0)
         return false;

      if (set[n - 1] > sum) // If last element is greater than sum, then we ignore it.
         return subsetSum(set, n - 1, sum);

      return subsetSum(set, n - 1, sum) || subsetSum(set, n - 1, sum - set[n - 1]);
   }
```

Recursion: Visualization

Here, we have a set for example
set[ ] = {10, 20, 30, 40, 50}
and we're trying to find the sum 80.

We are first going to start with the first iteration, which is going to be (the *n* number of elements, the sum).

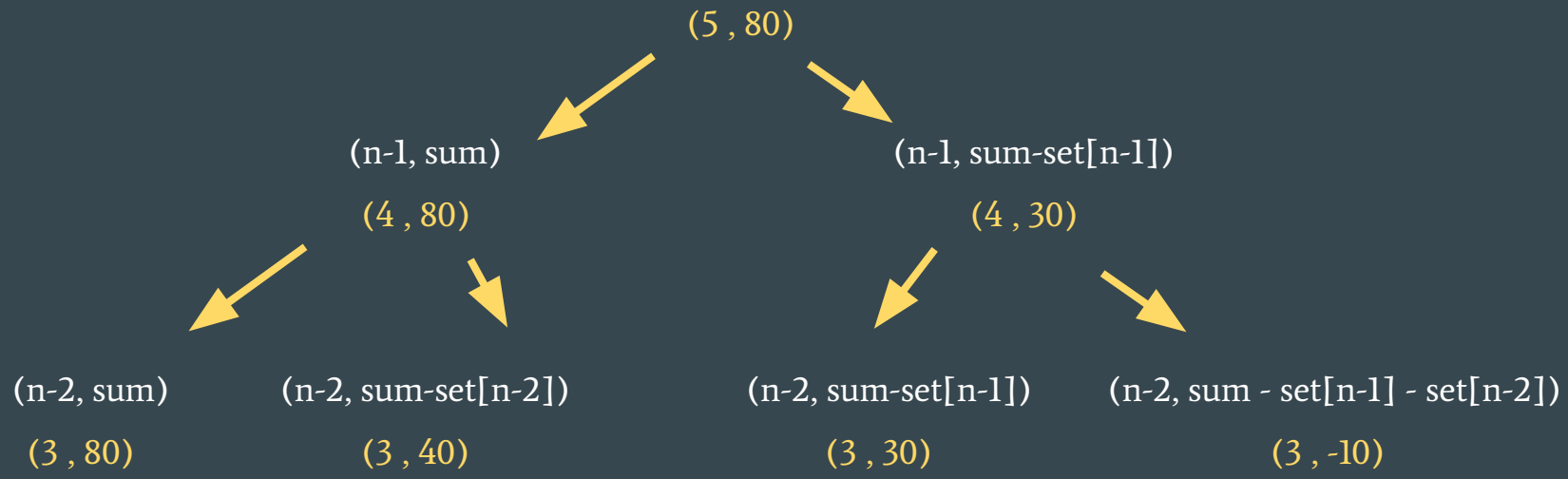
(5 , 80)

Recursion: Visualization

set[ ] = {10, 20, 30, 40, 50}
and we're trying to find the sum 80.

```
if (sum == 0)
   return true;
if (n == 0)
   return false;
```

Then we go from here and recursively iterate.



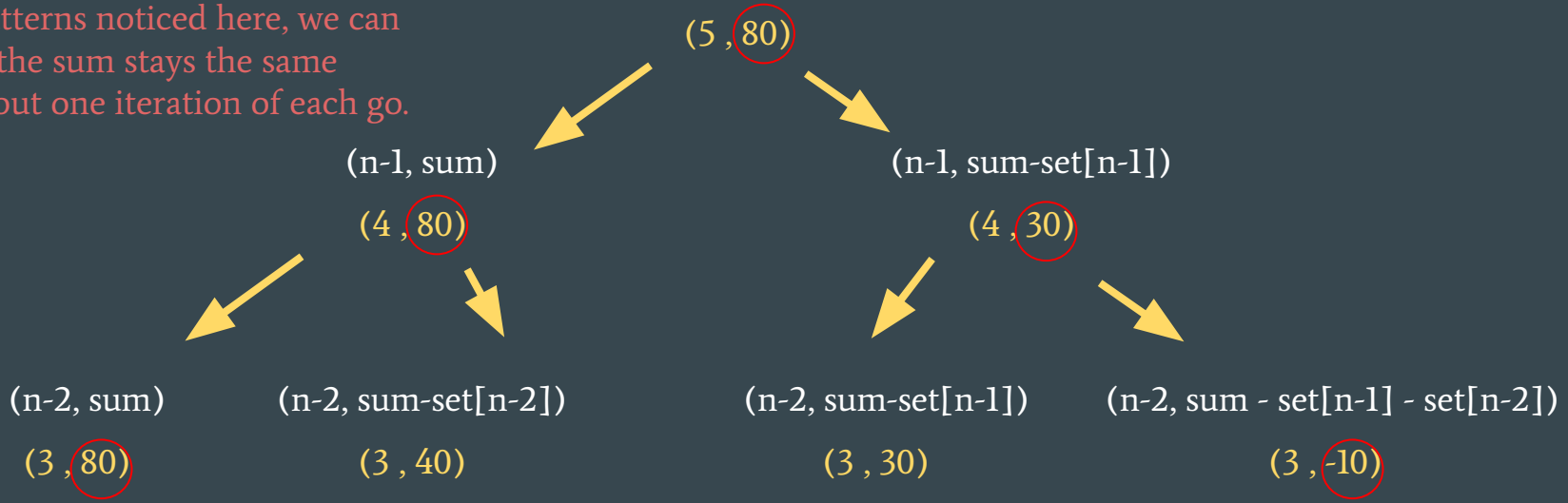
Recursion: Visualization

set[ ] = {10, 20, 30, 40, 50}
and we're trying to find the sum 80.

```
if (sum == 0)
  return true;
if (n == 0)
  return false;
```

Then we go from here and recursively iterate.

Some patterns noticed here, we can see that the sum stays the same throughout one iteration of each go.



and the other one, we notice that it always gets subtracted by the next element of the set when we iterate.

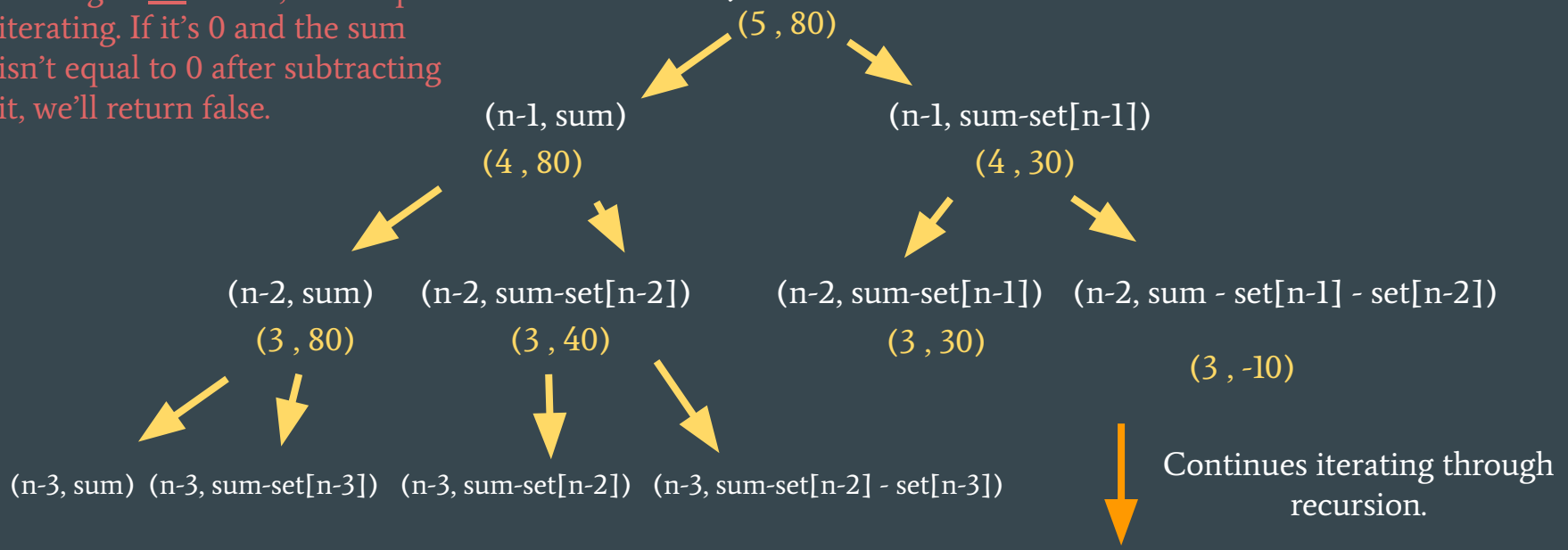
Recursion: Visualization

set[ ] = {10, 20, 30, 40, 50}
and we're trying to find the sum 80.

```
if (sum == 0)
   return true;
if (n == 0)
   return false;
```

As long as n-i isn't 0, it'll keep iterating. If it's 0 and the sum isn't equal to 0 after subtracting it, we'll return false.

Then we go from here and recursively iterate.



Continues iterating through recursion.

Recursion: Analyzing time complexity and runtime

- By using the recursion method, the worst time complexity is going to be O(2^n)
- Space complexity is going to be O(1)

Demonstration

Dynamic Programming: Description and Design Techniques

- Dynamic programming can be implemented with and without memoization. We will be using the approach **without** memoization, meaning no recursion will occur.
- The dynamic programming approach creates an n+1 by sum+1 boolean matrix, with rows representing the elements in the set and the columns representing their sums

Dynamic Programming: Pseudocode (setup)

```
int n = amount of elements in a set
int k = sum being found
int set[] = given set of elements
boolean matrix[][] = [n+1][k+1]

for(int a = 0; a <= k; a++) // i.e. each element in a row
        matrix[0][a] = false // set each element in the first row to false

for(int b = 0; b <= n; b++) // i.e. each column in the array
        matrix[b][0] = true // first element of each row (i.e. first column) set to true
```

Dynamic Programming: Pseudocode (main logic)

```
for(int i = 1; i <= n; i++)
        for(int j = 1; j <= k; j++)
                if(set[i-1] > j)
                        // if element is greater than current sum,
                        //  take value of above row (same column)
                        matrix[i][j] = matrix[i-1][j]
                else
                        // else take either above row (same column)
                        //  or above row, element columns left
                        // if element == sum, gets value from sum = 0
                        matrix[i][j] = matrix[i-1][j] || matrix[i - 1][j - set[i - 1]]
```

Dynamic Programming: Example

set = { 3, 4, 5, 2 }

n = 4

k = 6

```
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= k; j++)
        if(set[i-1] > j)
            matrix[i][j] = matrix[i-1][j]
        else
            matrix[i][j] = matrix[i-1][j] ||
                matrix[i - 1][j - set[i - 1]]
```

Sum

Elements in Array

	0	1	2	3	4	5	6
0							
3							
4							
5							
2							

Dynamic Programming: Example

set = { 3, 4, 5, 2 }

n = 4

k = 6

```
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= k; j++)
        if(set[i-1] > j)
            matrix[i][j] = matrix[i-1][j]
        else
            matrix[i][j] = matrix[i-1][j] ||
                matrix[i - 1][j - set[i - 1]]
```

Sum

Elements in Array

	0	1	2	3	4	5	6
0	*T*	*F*	*F*	*F*	*F*	*F*	*F*
3	*T*						
4	*T*						
5	*T*						
2	*T*						

Dynamic Programming: Example

set = { 3, 4, 5, 2 }

n = 4

k = 6

```
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= k; j++)
        if(set[i-1] > j)
            matrix[i][j] = matrix[i-1][j]
        else
            matrix[i][j] = matrix[i-1][j] ||
                matrix[i - 1][j - set[i - 1]]
```

Sum

Elements in Array

	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
3	*T*						
4	T						
5	T						
2	T						

Dynamic Programming: Example

set = { 3, 4, 5, 2 }

n = 4

k = 6

```
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= k; j++)
        if(set[i-1] > j)
            matrix[i][j] = matrix[i-1][j]
        else
            matrix[i][j] = matrix[i-1][j] ||
                matrix[i - 1][j - set[i - 1]]
```

Same value as above
Sum same as element
From above row (e cols to the left)

Sum

Elements in Array

	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
3	T	F	F	T	F	F	F
4	T	F	F	T	T	F	F
5	T	F	F	T	T	T	F
2	T	F	T	T	T	T	T

Dynamic Programming: Example

set = { 3, 4, 5, 2 }

n = 4

k = 6

```
for(int i = 1; i <= n; i++)
    for(int j = 1; j <= k; j++)
        if(set[i-1] > j)
            matrix[i][j] = matrix[i-1][j]
        else
            matrix[i][j] = matrix[i-1][j] ||
                matrix[i - 1][j - set[i - 1]]
```

Sum

Elements in Array

	0	1	2	3	4	5	6
0	T	F	F	F	F	F	F
3	T	F	F	T	F	F	F
4	T	F	F	T	T	F	F
5	T	F	F	T	T	T	F
2	T	F	T	T	T	T	T

Demonstration

{4, 1, 10, 12, 5, 2}

n = 6

k = 9

Dynamic Programming: Analyzing time complexity and runtime

- O(n*sum)
- n = size of the array
- sum = target sum
- Like the recursion method, the time complexity depends largely on the size of the input array.
  - Performs better than the recursive method and the backtracking method.
  - Similar runtime with the memoization method.
- Space complexity: O(n*sum)

Pros/Cons

Recursive:

- Larger time complexity
  - $O(2^n)$
- Smaller space complexity
  - O(1)

Dynamic Programming:

- Smaller time complexity
  - O(n*sum)
- Larger space complexity
  - O(n*sum)

Real World Application

- Computer passwords
  - Computer passwords are used to verify a user's identity before allowing access.
  - Easy way is for the computer to store the password, and compare it with the input of the user, however, this is a big security issue if the internal file of the password storage is exposed
  - Instead, there is another method where the computer generates a large number (n) for example, and the computer keeps the total associated with the appropriate subset.
  - When the user types in the "subset" (aka password, which gets converted to subsets in the program), the computer will test if the total is correct. It's a more secure way because it's harder to impersonate unless they can reconstruct the subset knowing the n_i and the total.

Real World Application

- Message Verification
  - We can have a message verification system with a sender and a receiver.
  - In our scenario, the receiver wants to make sure that the message he received is a legitimate message and not from an impersonator.
  - The message client of the sender and the message client of the receiver will agree on a set of a_i numbers (eg. 500) and a set of totals of T_j (eg 200) numbers.
  - These numbers can be publicly known but only the sender would knows which subsets of the a_i corresponds to T_j.
  - The sender then verifies his legitimacy by sending in n number of subsets of a_i.

References

https://staff.fmi.uvt.ro/~stelian.mihalas/cry_sec/download/NUMERE/SECOND.PDF

https://www.geeksforgeeks.org/subset-sum-problem-osum-space/?ref=rp

https://www.geeksforgeeks.org/subset-sum-problem-dp-25/amp/