University of Washington March 4, 2020
Department of Computer Science and Engineering
CSE 417, Winter 2020

Homework 8, Problem 2

Problem 2 (10 points) Strict Subset Sum:
The strict subset sum problem is: Given a set of values $\{s_1, \ldots, s_n\}$, and an integer K, is there a subset of the items that sum to exactly K. Design an algorithm that solves the strict subset sum, and finds a set that sums to K with as large a number of items as possible. Your algorithm should have runtime polynomial in n and K.

Answer:

algorithm:

```
def strict_subset_sum(S, K):
    n = len(S)
    sorted(S)
    Opt = [[0 for i in range(K + 1)] for j in range (n + 1)]
    Opt[0][0] = 1

    for j in range(1, n + 1):
        for k in range(0, K + 1):
            Opt[j][k] = Opt[j-1][k]
            if k-S[j-1] >= 0:
                Opt[j][k] = Opt[j-1][k] or Opt[j-1][k-S[j-1]]

    ans = []
    k = K
    j = n
    while j >= 1:
        if Opt[j-1][k] != 1:
            ans.append(S[j-1])
            k -= S[j-1]
        j -= 1
    return ans
```

proof:

base case:

By defination, when there is no value, the sum of 0 can be attaibed with 1 possibility. Therefore the Opt[0][0] equal to zero.

induction hypothesis: First, sort the set of value. After sorting, smaller j_{th} value means smaller value. For each value s_j in set S, assume the value $s_0 \ldots s_{j-1}$ has been correctly determined :

Then when k equal to each value from 0 to K: Opt[j][k] = Opt[j-1][k] or Opt[j-1][k-S[j-1]]

The function apply to each inductive step since for each value k, there are only two possibilities: (1) value k can be attaibed with the first j_th value add a precaclulated sum, therefore it Opt[j-1][k-S[j-1]] is must true or value k can already be attained before j_{th} value, Opt[j][k] inherent the true value either from Opt[j-1][k] or Opt[j-1][k-S[j-1]].
(2) value k can not be attaibed with the first j_{th} value add a precaclulated sum, therefore it inherents the impossibility from the result of first j-1 th value to get k;
In either case, the value of Opt[k] rely on the correctly precaclulated value Opt[k] and Opt[k - s], regardless k can be attaibed by values from value $s_0 \ldots s_{j-1}$, since or operant can automatically inherent the true if it can be attained, if either one of Opt[j-1][k] and Opt[j-1][k-S[j-1]] is true, otherwise, it inherent the false value.
Since the algorithm calculate Opt$[0 \ldots k]$ from s_0 cummulative to s_n, and for each adding s_j, the algorithm attaines result built opon from correctly precomputed value, then algorithm is garantees to find the at first j value, wether it can get a sum of k. And the subproblems built up on top of prior value all the way to nth value.
Because the set of values was sorted, when reconstructing, the poins can now trace further back to the subproblem with smaller j_{th} also garantees it will get smaller value, instead of stop at larger value. And smaller value means the residue part (k-S[j-1]) has more room to more value, hence the subset size is larger.

complexity:
Since on each subproblem of subset $s_0 \ldots s_i$, the algorithm only traces the result from at most K result, so each subproblem will has K operations. And there are n subproblem, therefore the time complexity is $O(nK)$.