Dynamic Programming Integer Subset Sum Problem

CSCI 3100

Dynamic Programming Review

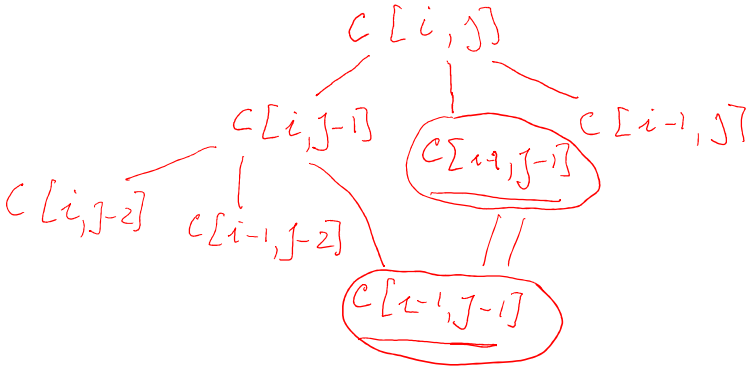
Key ideas

- Express solution in terms of sub-problems (recurrence)
- Sub-problems have overlapping sub-problems
- Order sub problems to avoid re-computation
- Number of sub-problems is polynomial (in terms of input size)

Longest Common Subsequence (LCS)

- Given strings X and Y find the longest subsequence of X that is also a subsequence of Y

LCS – overlapping sub-problems



LCS Recursive Solution Complexity

$2^{(n+m)}$

Subset Sum Problem

Let $w_1,\ldots,w_n$ = {6, 8, 9, 11, 13, 16, 18, 24}

Find a subset that has as large a sum as possible, without exceeding K

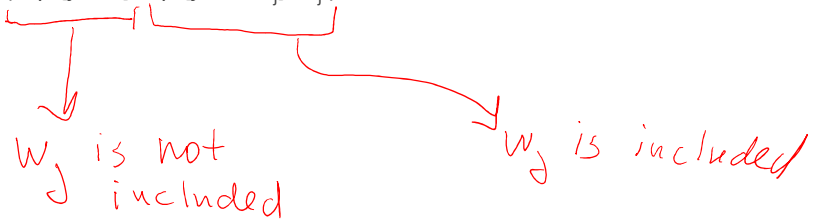
Assumptions:

- w_j are integers
- K is an integer

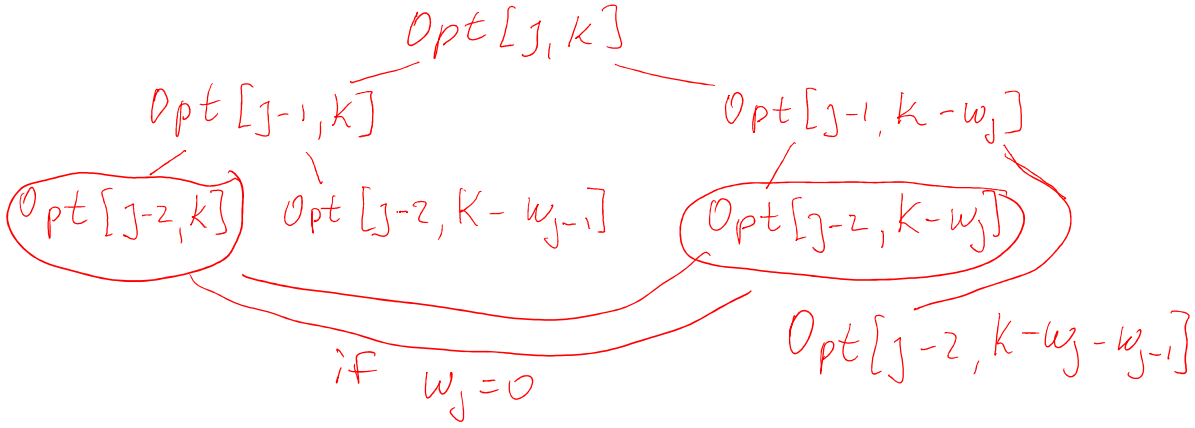
Subset Sum Recurrence

Opt[j, K] the largest subset sum of $\{w_1, \ldots, w_j\}$ that sums to at most K

Opt[j, K] = max(Opt[j-1, K], Opt[j-1, K-w_j]+w_j)



Subset Sum Overlapping Sub-problems



Order sub-problems to avoid re-computation

J = 1 to N

K = 1 to K

Opt[j, K] = max(Opt[j-1, K], Opt[j-1, K-w_j]+w_j)

Example: S={5, 8, 3, 2}, K = 13

S	k → / j ↓	1	2	3	4	5	6	7	8	9	10	11	12	13
5	1	0_F	0_F	0_F	0_F	5_T	5_T	5_T	5_T	5_T	5_T	5_T	5_T	5_T
8	2	0_F	0_F	0_F	0_F	5_F	5_F	5_F	8_T	8_T	8_T	8_T	8_T	13_T
3	3													
2	4													

Algorithm

```
for j = [1 to n]
  for k = [1 to K]
    if (j == 1) AND (w[j] ≤ k)
        S[j,k] = w[j]
        B[j,k] = true
    else
        case1 = S[j-1,k]
        case2 = 0
        if (w[j] ≤ k)
            case2 = S[j-1, k-w[j]] + w[j]
        if (case1 >= case2)
            S[j,k] = case1 ; B[j,k] = false
        else
            S[j,k] = case2 ; B[j,k] = true
```

S = n x K matrix initialized to 0

B = n x K matrix initialized to 0

Algorithm Complexity

K is small and fixed value

$\Theta(N * K)$

K is a function of N

$K = 2N \Rightarrow \Theta(N^2)$

$K = 2^N \Rightarrow \Theta(N * 2^N)$

Prove that the algorithm works for any N

Does the algorithm work for non-integer K

NO

Does the algorithm work for non-integer w_j

NO