WEEK4: DYNAMIC PROGRAMMING AND BACKTRACKING

Agenda

Survey Comments

Backtracking

- How should we approach backtracking problems? For example, in dynamic programming we have the following framework: 1) Identify params that affect problem 2) Identify subproblems 3) Define recursive formula 4) Implement What kind of steps could we go through to solve a backtracking problem?

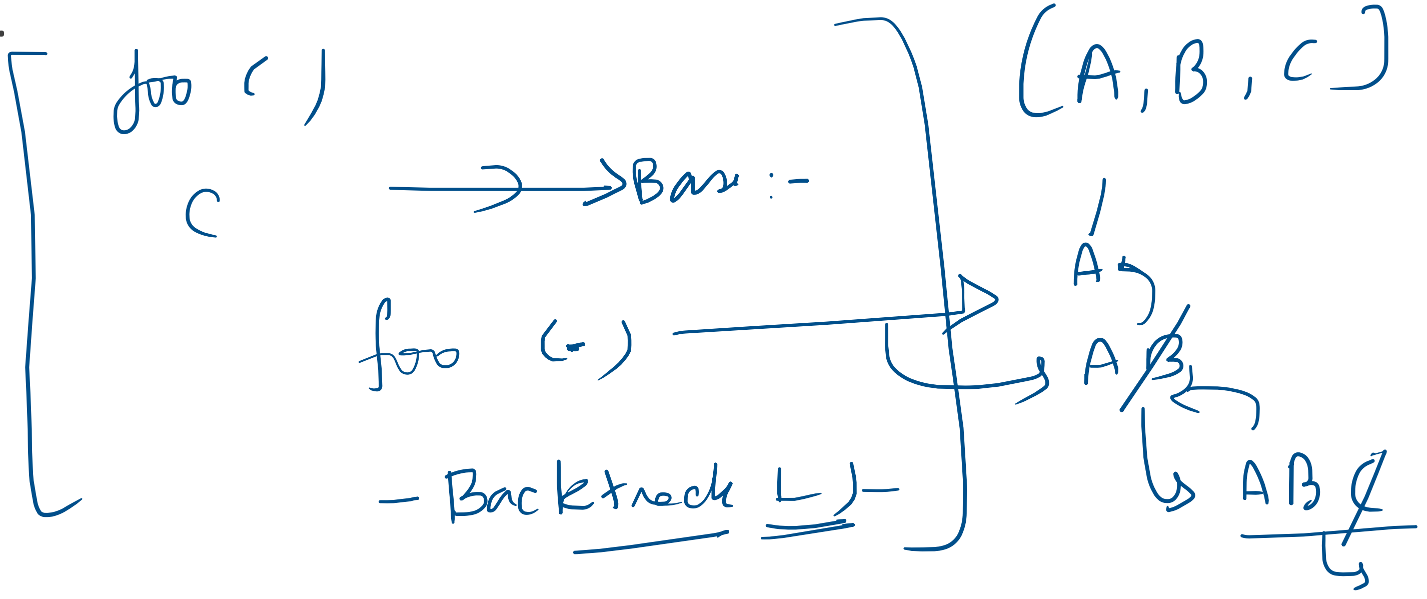
  - ALL Possible solutions - all possible combinations of those characters (all permutations); powerset

(_ _ _)

{1, 2, 3}

Backtracking

- Could you please walk through additional examples of backtracking code? Most resources I've found only show the decision making tree but fail to give examples of code or fail to explain how to arrive at the code.



Backtracking – time complexity

→ O, O, O, O, O, ...

↓ y/s ↓ y/s

[1, 2, 3, 4] | 2^n

↓ ↓

1,{}, 2,{}

$2 \times 2 \times 2 \times \dots = (2^n)$

→ {A, B, C} → (n!)

→ O, O, O, O, O ... sum: X

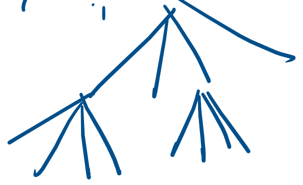
4

1, 2, 3, 2

{ {1, 3} {2, 2} }

$n^{(k)}$

EXP



Backtracking

week known new

meek

- More backtracking examples

https://www.techiedelight.com/backtracking-interview-questions/

https://www.geeksforgeeks.org/top-20-backtracking-algorithm-interview-questions/

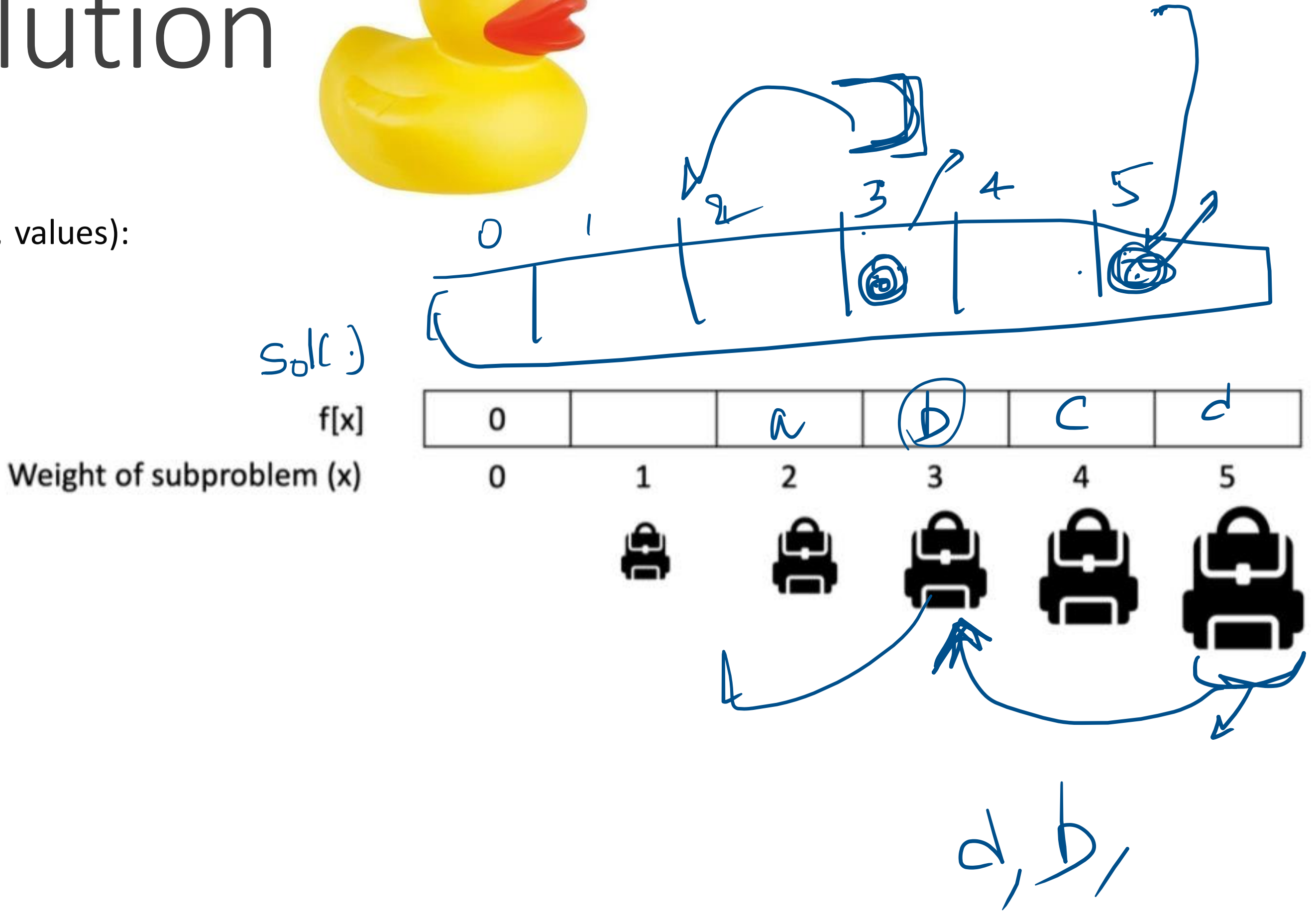
DP – Optimal Solution

```python
# Print weights of items that form the optimal solution
def unbound_knapsackOptimalSolution(W, n, weights, values):
   dp = [0]*(W+1)
   sol = [0]*(W+1)

   for x in range(1,W+1):
      for i in range(n):
         wi = weights[i]
         if wi <= x:
            if((dp[x-wi] + values[i] ) > dp[x]):
               dp[x]=dp[x-wi] + values[i]
               sol[x]= wi
   w = W
   solution = []
   while w>0:
      solution.append(sol[w])
      w = w-sol[W]

   return solution

print(unbound_knapsackOptimalSolution(10,5,[4,9,3,5,7], [10,25,13,20,8]))
```



- Another way -http://rosulek.github.io/vamonos/demos/rod_cutting.html

HW- question

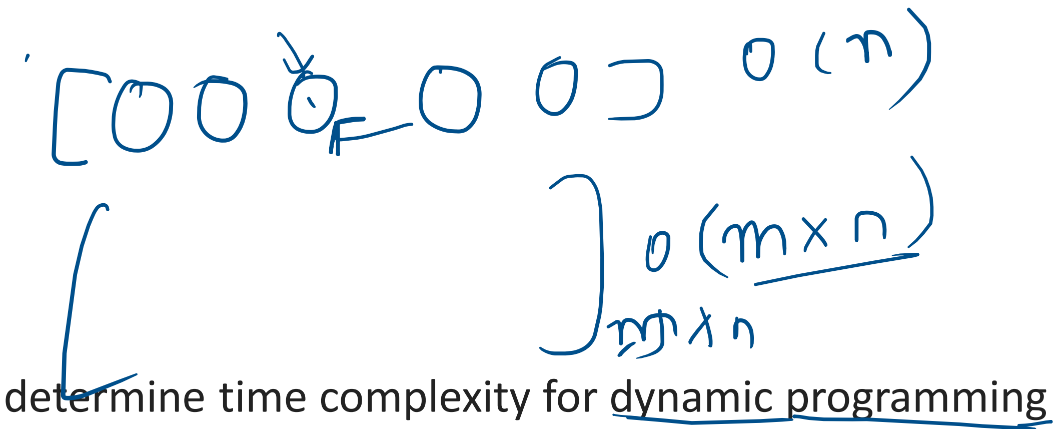
Given a list of numbers, return a subset of non-consecutive numbers in the form of a list that would have the maximum sum.

Example 1: Input: [7,2,5,8,6]
Output: [7,5,6] (This will have sum of 18)

Example 2: Input: [-1, -1, 0] 0 . [ ] , [0] .
Output: [0] or [] (Maximum possible sum for this array = 0 ; you may return [0] or [])

Example 3: Input: [-1, -1, -10, -34] [-1]
Output: [-1] or [0] or [] (The maximum possible sum is –ve or 0; you may return [-1] or [0] or [])

Other comments



- more examples step-by-step walk through of how to determine time complexity for dynamic programming and backtracking problems (like the steps that are followed in the mathematical analysis of algorithms module, once I get an equation I can usually apply one of the analysis methods with no problem, but I sometimes struggle to get the correct equation). thank you
- analyzing runtime and space complexity for dynamic programming algorithms
- Backtracking, Optimal Solutions
- 0-1 Knapsack and Backtracking