TCSS 343 - Week 6 - Monday

Jake McKenzie

February 24, 2019

Dynamic Programming

"We don't much care if you don't approve of the software we write."
...
Eric Hughes

"The best programs are the ones written when the programmer is supposed to be working on something else."
...
Melinda Varian

0. Consider the Job Selection problem as defined here:

$$JS(P[1\ldots n]) = \begin{cases} P[1] & \text{if } n = 1 \\ \max\{P[1], P[2]\} & \text{if } n = 2 \\ \max\{JS[P[1\ldots n-2]] + P[n], JS[P[1\ldots n-1]]\} & \text{if } n > 2 \end{cases}$$

You are a wealthy socialite who charges for public appearances. This month/year/life you have offers for appearances every day with different pay outs. The problem is they are all in different cities in North America so if you work one job you cannot work jobs in the day before or after due to travel. You want to select the jobs to work to make the most money this month/year/life.

Say I modified the problem to the following:

You are a wealthy socialite who charges for public appearances. This month/year/life you have offers for appearances every day with different pay outs. The problem is that you need some sort of break between jobs due to government regulations but the talent company requires you to work two days consecutively. That is to say: you cannot work jobs three days in a row due to regulation but you must work 2 days in a row to meet the obligations to your talent agency. You want to select the jobs to work to make the most money this month/year/life.

Express the modified problem formally with input and output conditions.

1. State a self-reduction for your problem. Use the reduction above for inspiration.

2. State a dynamic programming algorithm based off of your reduction that computes the maximum earnings.

3. Using the algorithm you constructed to compute the maximal earnings then recover your solution by keeping track of your maximal earnings at each stage and the solution array of 1s and 0s.

$$P = \{5, 9, 3, 6, 5, 8, 8, 4, 6, 6, 7, 7, 8, 4, 7\}$$

4. Construct an algorithm to recover your solution using the binary solution array you constructed on a prior page.

5. Consider the *stamp* problem. We have a postage system that has n stamps with values $v_1, v_2, \ldots, v_n$ and we want to pay a value y in such a way that the total number of stamps is minimized. More formally, we want to minimize the quantity:

$$\sum_{i=1}^{n} x_i$$

subject to the restraint

$$\sum_{i=1}^{n} x_i v_i = y$$

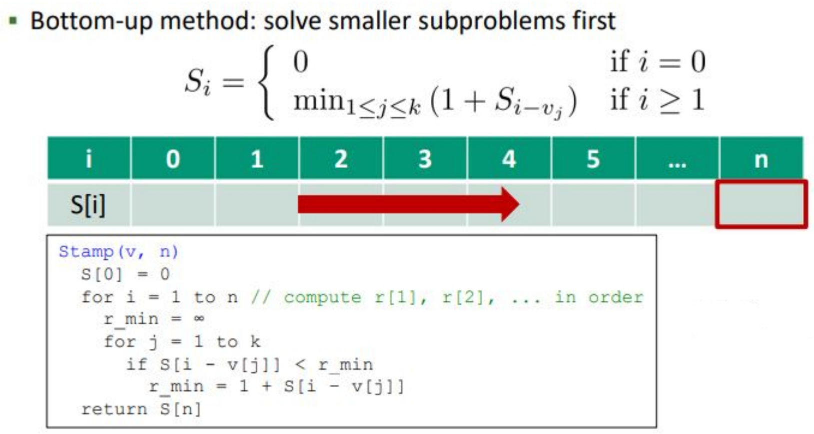
Here, $x_1, x_2, \ldots, x_n$ are positive integers, including zero.

Express the problem formally with input and output conditions.

6. What is the time complexity of the brute force solution to this problem below?

```
Stamp(v, n)
  r_min = ∞
  if n == 0 // base case
    return 0
  for i = 1 to k // recursive case
    r[i] = Stamp(v, n - v[i])
    if r[i] < r_min
      r_min = r[i]
  return r_min + 1
```

7. What is the time complexity of the optimal solution to the recursive definition of the problem below?



- Bottom-up method: solve smaller subproblems first

$$S_i = \begin{cases} 0 & \text{if } i = 0 \\ \min_{1 \le j \le k} (1 + S_{i-v_j}) & \text{if } i \ge 1 \end{cases}$$

i	0	1	2	3	4	5	...	n
S[i]								

```
Stamp(v, n)
  S[0] = 0
  for i = 1 to n // compute r[1], r[2], ... in order
    r_min = ∞
    for j = 1 to k
      if S[i - v[j]] < r_min
        r_min = 1 + S[i - v[j]]
  return S[n]
```

8. Construct the optimal solution by backtracking (dynamic programming algorithm using the recursive definition).