Somya Lalwani
2/20/19
349

DP Problem set

① unlimited coins - $d1 < d2 < \ldots < dn$ - make change for value v
= find set of coins whose total value is v
- set of coins = v → can be determined by dynamic programming
- sub-problems:
- consider the variable $s(u)$ to be sub-problem for $0 \leq u \leq v$
- consider $x_1, x_2, \ldots, x_n$ be set of denominations
- $s(u)$ is true, if it is possible to make change for value u using coin denominations $x_1, x_2, \ldots, x_n$

$$s(u) = \begin{cases} \text{True} & \text{if possible to make change for } u \\ \text{False} & \text{if not possible to make change for } u \end{cases}$$

- recursive formula to determine set of coins whose total val v:
$s(u)$ = True, if $s(u-x_i)$ True iff $s(u-x_i)$ is true for some i
- value $s(0)$ = True, to maintain consistency
- final answer = $s(v)$
- pseudocode:

```
makechange(x1, ... xn, v):
 s[0] = true
 for u=1 to v:
  s[u] = false
 for i = 1 to n:
 if u ≥ xi and s[u-xi]:
  s[u] = true
  return s[v]
```

- makeChange function takes $x_i, \ldots, x_n$ and v as parameters, where v is input value and $x_i, \ldots, x_n$ is given set of denominations
- function correctly determines whether possible to make change for v w/ given denomination of $x_1, \ldots, x_n$
- proof of correctness:
- proof of induction on value "v"
- if value = v = 0, then making changes for denominations $x_1, \ldots, x_n$ is possible

- if possible to make change for value "u", then it is possible to make change for any value belong to "u + xi"
- running time of alg: - v sub-problems
- each sub problem takes constant time of $\leq n$ to ✓ $s(v - x_i)$
= $O(nv)$

② Firestone is opening restaurants along Highway 1 - n possible locations, distances are in miles and in ↑ order $(m_1, m_2, \ldots m_n)$
- constraints: at each location, Firestone can only open 1 restaurant
- any 2 restaurants must be at least k miles apart, where k is a positive integer
- def: $P[i]$ is defined as the maximum expected profit at location i
- recursive def based on constraint:

$$P[i] = \max \begin{cases} \max_{j<i} \{ P[j] + \alpha(m_i, m_j) \cdot p_i \} \\ p_i \end{cases}$$

$$\alpha = \alpha(m_i, m_j) = \begin{cases} 0 & \text{if } m_i - m_j < k \\ 1 & \text{if } m_i - m_j \geq k \end{cases}$$

- max expected profit at location i comes from max of expected profits of location j and whether we can open a location at location i
- profit from opening restaurant = p_i
- $P[j]$ = max expected profit at location j * may/may not exist
* likely that at location i, p_i is higher than $P[j] + \alpha(m_i, m_j) \cdot p_i$
- pseudocode:
  expected profit (N, P)
- input: N locations; $P[1 \ldots n]$ where $P[i]$ is profit at location i
- output: max expected profit P_{max}
- array of max expected profit $Profit[1 \ldots n]$:
  $Profit[i]$ denotes max expected profit at location i

```
for i=1 to N:
   profit[i] = 0
for i = 2 to N:
  for j = 1 to i-1
     temp = Profit[j] + α(mi, mj) · P[i]
  if temp > Profit[i] :
     temp = Profit[i]
  if profit[i] < P[i]:
     profit[i] = P[i]
```

- complexity analysis: 2 for loops → $O(n^2)$

③ - you are going on a long trip - along the way n hotels at miles posts $a_1 < a_2 < \ldots < a_n$, where each a_i is measured from starting point

- must stop at final hotel (at distance a_n) which is your dest
- travel 300 miles a day
- algorithm that determines optimal seq of hotels at which to stop
- let OPT(i) be the minimum total penalty to get to hotel i
- recursive formula:
- to get to OPT(i), consider all possible locations j we can stay at night before reaching hotel i
- minimum penalty to reach i is the sum of:
  - minimum penalty of OPT(j) to reach j
  - and cost $(300-(a_j-a_i))^2$ of a one-day trip from j to i
- b/c interested in min penalty to reach i:

$$OPT(i) = \min_{0 \le j < i} \{ OPT(j) + (300-(a_j-a_i))^2 \}$$

- base case is OPT(0) = 0
- pseudocode:

```
//base case
OPT[0] = 0
// main loop
for i=1 ... n:
   OPT[i] = min(OPT[j]+(300-(aj-ai))^2 for j=0 ...i-1])
```

// Final result
return OPT[n]

- complexity:
- have n subproblems, each subproblem i takes o(i) time
- overall complexity:

$$\sum_{i=1}^{n} o(i) = o\left(\sum_{i=1}^{n}\right) i = o\frac{n(n-1)}{2} = o(n^2)$$

④ pebbling checkerboard
- given checkerboard → 4 rows and n columns
- constraint: placement of pebbles to be legal, no 2 of them can be on horizontally or vertically adjacent squares
- value of placement = sum of integers in the squares that are covered by pebbles of that placement
- possible patterns which can occur in any column:
① all empty - empty pattern
② 4 patterns which exactly have 1 pebble
③ 3 patterns which have 2 pebbles
- how patterns pair up in adjacent columns:
① every pattern is compatible w/ empty pattern
② patterns w/ 1 pebble are compatible w/ all patterns which do not have pebble in same row
③ patterns w/ 2 pebbles are compatible w/ complementary pattern pattern
- dynamic programming solution:
- maintain 8 arrays of n elements for each of 8 patterns
- max value from c, c[n] and pebble the nth column according to some j such that c = cj[n]
- subtract value of pebbled square from c and search for the best cj[n-1]
↳ this way, the dynamic programming solution grows r
- time complexity: o(n) running time b/c backtracking