Chapter 6: Dynamic programming

How efficient is efficient enough?

is n^2 enough or should I ask if n is possible?

Done via hw:

6.1

6.4

6.7

6.8

Others:

6.2

a1 a2 a3 ...an

Fill array S[i]:

1. S[i] = the minimum total penalty of the sequence ending at ai
2. Formula:

S[1] = (200-a1)2

S[i] = min { S[1] + (200- ai + a1)2 ; S[2] + (200- ai + a2)2 ; … ; S[i-1] + (200-ai + ai-1)2 }

1. Explanation:

* At any hotel ai we consider which hotel aj (0<j<i) is the optimal hotel immediately before ai.
* Because S[j] is the optimal penalty for stopping at aj, S[j] + (200- ai + aj)2 will be the optimal penalty for stopping at ai immediately after aj.

1. Answer: For each shell S[i], also store a pointer to the immediately before shell that gives the optimal value. We can find the answer by backtracking from the last shell S[n]
2. Complexity: O(n^2)

6.3

m1 m2 … mn

p1 p2 … pn

min distance between 2 restaurants = k

Filling an array P[i]:

* Definition of a shell

P[i] = max total profit of sequence of restaurants ending at mi

* Formula of the shell

P[1] = p1

P[i] = max { P[1...j] } + pi with j< i & mj <mi -k <mj+1

* Explanation
* Answer’s location

The max value in P

* Complexity

O(n^2)

6.5

Table C: 4 x n

2n pebbles

a) Patterns: in a column

0 pebble.

1 pebble: any row 1-4

2 pebbles: either 1,3 or 2,4

=> 7 types:

Type index: 0 1 2 3 4 5 6

Pebble index: # 1 2 3 4 1,3 2,4

b) Filling a table T of 7 x n

* Definition of a shell

T[i,j] = the maximized sum of the table from 1 to i, with the type of current shell is j

(i = 0,...6)

* Formula of the shell:

T[i,j] = C(i,j) + max {T[i-1, j’]} with j’ = 0,...6 and j’ compatible with j

Thus:

T[i,0] = C(i,0) + max { T[i-1,0];T[i-1,1];T[i-1,2];T[i-1,3];T[i-1,4];T[i-1,6]}

T[i,1] = C(i,1) + max { T[i-1,0];T[i-1,2];T[i-1,3];T[i-1,4];T[i-1,6]}

T[i,2] = C(i,2) + max { T[i-1,0];T[i-1,1];T[i-1,3];T[i-1,4];T[i-1,5]}

T[i,3] = C(i,3) + max { T[i-1,0];T[i-1,1];T[i-1,2];T[i-1,4];T[i-1,6]}

T[i,4] = C(i,4) + max { T[i-1,0];T[i-1,1];T[i-1,2];T[i-1,3];T[i-1,5]}

T[i,5] = C(i,5) + max { T[i-1,0];T[i-1,2];T[i-1,4];T[i-1,6]}

T[i,6] = C(i,6) + max { T[i-1,0];T[i-1,1];T[i-1,3];T[i-1,5]}

* Explanation
* Answer’s location

The max in the last column of T

* Complexity

O(n)

6.6 (oh well …)

6.9

...

6.10

* Definition of a shell

P[i] = possibility to get k heads from coins 1 to i

* Formula of the shell

P[i] = pi \* sum{j=1...i-1} ( P[i-1]/pj )

* Explanation
* Answer’s location
* Complexity

O(n^2)

6.11

string A = a1 a2 .. an

string B = b1 b2 ...bn

Table m x n

* Definition of a shell

S[i,j] = longest common subsequence

* Formula of the shell

S[i,j] = S[i-1,j-1] + 1 if A[i] = B[j]

or max { S[i,j-1] , S[i-1,j]} otherwise

* Explanation
* Answer’s location
* Complexity

O(mn)

-------------------------------------------------------------------------------------------------------------------------------

5.14, 20 , 23

5.