This thread is part of the proposed study plan for the second test. Discuss the dynamic programming practice problems here.

View questions here: http://beastie.cs.ua.edu/concepts/cs/al/dynamic.html

This thread covers all questions for dynamic programming/memoization.

Work together on the proposed answers to questions on this shared Google Doc (comment reasoning/arguments behind answers)

Subject: Re: Concept Review: Dynamic Programming
Posted by davidmccoy on Tue, 14 Mar 2017 17:24:10 GMT

For number 4,

Quote: Consider memoizing this function:

```javascript
function g(x,items,y) {
    if (x == 0) return 1;
    if (x < 0) return 0;
    if (y == items.size) return 0;
    return minimum(g(x-items[y],items,y),g(x,items,y+1));
}
```

Assuming all three original base cases are retained and the smallest memoization table possible, what would be the memoization table’s largest index/indices, where x refers to the original value of x?

Since there are no base cases stored in the table, I thought table[0][anything] is not needed, and additionally, table[anything][items.size] is not needed, as it is covered by a base case. I thought then that x-1 and items.size -1 should be the indices, but that is not a given answer.

Subject: Re: Concept Review: Dynamic Programming
Posted by dturner2 on Wed, 22 Mar 2017 21:50:07 GMT

I'm having trouble understanding these questions, does anyone understand them enough to explain the following from the question bank?

Quote:

Consider memoizing this function:

```javascript
function f(x) {
```

...
if (x == 0) return 0;
if (x == 1) return 1;
return f(x-2) + f(x-1);
}

In order to remove the two original base cases, how would the memoization table be initialized? Assume the memoization table is named memo.

Memoizing is the process of making an algorithm, which makes many repeated calls on similar parameters, more efficient (as an application of dynamic programming). This is achieved by storing a "memo" of the results of the function in a table. The table is a single- or multidimensional array, depending on how many parameters change in the process of recursive calls.

For this example, one parameter is taken in by the function ‘f’: x. This parameter changes during recursive calls to f as seen in the return line (f(x-2), f(x-1)). So the memoization table will have one dimension, and a value can be called from it like 'memo[i]'.

To remove base cases from the function, we can initialize them in the memo table. The memoized function we would write for f would contain something like "if (memo[x] exists) then return memo[x]" as its first line to provide the previously stored result of the function, which would save the time needed to calculate it.

In the case of the base cases, if we initialize in the memo table that are covered by the base cases, then we no longer need the base cases in the function. So look at the two places the base cases cover: if x is 0, or if x is 1, there is some value the function returns. So we can initialize memo[0] and memo[1] to have the values that are returned by the base cases (0 and 1, respectively).

Was this ever resolved? Because I was sort of under the same impression

http://beastie.cs.ua.edu/cs201/dynamic.html
When creating for loops for the make change example. coin size is listed as largest to smallest, but the for loop seems to be an endless loop. should it be c--?

Subject: Re: Concept Review: Dynamic Programming
Posted by MattM on Mon, 27 Mar 2017 18:28:20 GMT
View Forum Message <> Reply to Message

Does anyone have a link to some good extra examples for memoization besides the practice problems and dynamic programming page on the beastie website?

Subject: Re: Concept Review: Dynamic Programming
Posted by lusth on Tue, 28 Mar 2017 11:58:01 GMT
View Forum Message <> Reply to Message

jjlukas wrote on Sun, 26 March 2017 20:13 http://beastie.cs.ua.edu/cs201/dynamic.html

When creating for loops for the make change example. coin size is listed as largest to smallest, but the for loop seems to be an endless loop. should it be c--?

It should. Fixed now.