CSE525 Lec 8: Dynamic Programming

Debajyoti Bera (M21)

Fibonacci Numbers

Q: Design (recursive, as usual) algorithm for computing n-th Fibonacci number.

```
def Fibonacci(int n): // returns n-th Fibonacci number
    ...
```

Q: Estimate running time complexity.
Derive both upper and lower bounds.

Q: Estimate space complexity.

Fibonacci Numbers: Memoization

Design better method using extra space

In the recursion tree of Fibonacci(n) ...

- How many times Fibonacci(n-1) is called?
- How many times Fibonacci(n-2) is called?
- ...
- How many times Fibonacci(1) is called?

R(n,a) = Number of times Fibonacci(a) is called in the recursion tree of Fibonacci(n)?
Q: Derive a recursive expression along with base case.
Q: Compute R(n,1) = ?

Memoization : Store intermediate values in a cache/lookup-table when they are computed.

Fibonacci Numbers: Dynamic Programming

1. Create the cache/lookup-table using the recurrence
2. Solve the problem using the values of the cache
3. Makes sense when all values of the cache are going to be used

Fib(0)=1	Fib(1)=1	Fib(2)	Fib(3)	Fib(3)	Fib(4)	Fib(5)	Fib(6)	Fib(7)

Q: Advantage over non-memoized recursive approach? Over memoized approach?

Fibonacci Numbers: Constant-space DP

Computing Fibonacci(n) requires only a table of size 2!

Fib(2) =	Fib(0)	Fib(1)
Fib(3) =	Fib(1)	Fib(2)
Fib(4) =	Fib(2)	Fib(3)
	…	

DP for solving a question Q

1. Specify a problem P (could be different from Q) :
   In plain English, the input(s) to the problem and its output(s)
   P(x) is "Given input x, compute Fib(x)"
2. Give a recurrence expression/formula or recursive algorithm for solving P :
   Along with base case, in terms of smaller instances of **P**
   P(0)=1,P(1)=1,P(x) = P(x-1) + P(x-2) for x > 1
3. Prove correctness of recurrence relation by explain an (optimal) substructure property.
   Fib(x) is defined as Fib(x-1) + Fib(x-2) which are computed by P(x-1),P(x-2).
4. Describe a memoization structure (need not always be arrays/tables)
   For computing P(x) we use an array T[1,2].
5. Give an algorithm/ordering for solving P for <u>all</u> values.
   Initialize T[1] = P(0) and T[2] = P(1).
   For i=2...n, compute tmp = T[2], T[2] = T[1] + T[2], T[1] = tmp
   At i=j T[2] will store P(j)
6. How to solve problem Q from values :
   n-th Fibonacci no. = Compute P(n) = value of T[2] after step-4.
7. What is space and time complexity for solving problem?
   Space complexity = 2, Time-complexity = O(n) (assuming O(1) int. addition)
8. ~~For certain problems, how to obtain the optimal structure?~~

Longest Increasing Subsequence of A[1 … n]

LIS2(j) = length of longest increasing subsequence of A[j … n] that starts with A[j] (defined for all j=1 … n)

If for all k=j+1 … n, A[j] > A[k] LIS2(j) = 1

Otherwise, LIS2(j) = $\max_k$ {1 + LIS2(k) } where max is taken over all k s.t. A[j] < A[k]

Q: Give an ordering for computing all values of LIS2(j) for all j=1…n : LIS(j+1) … LIS(n) are sufficient to compute LIS2(j). So LIS2 values can be computed in this order: n, n-1, … 3,2,1

Q: How to compute longest inc. subseq. of A[1 … n] using the LIS2(j) values?

1. Use of sentinel A[0] = -inf.
2. Process all LIS2(j) values.

DP for solving a question Q

1. Specify a problem P (could be different from Q) :
   Given j, compute LIS2(j) = length of the longest incr. subseq. in A[j … n] that starts with A[j]
2. Give a recurrence expression/formula or recursive algorithm for solving P
   LIS2(n)=1.
   For j < n, LIS2(j) = max{ 1 + LIS2(k) : k s.t. A[j] < A[k]}. If no such k is there, then LIS2(j)=1.
3. Justify recurrence
   Let S be the longest incr. subseq. in A[j...n] starting with A[j]. Clearly, S=A[j].T where T is the part after A[j]. T must start with A[k] for some k > j.
   Claim: A[j] < A[k]. This is since S must be an increasing subseq.
   Claim: T must be longest incr. subseq. in A[k...n] that starts with A[k]. If T was not the longest, instead there was a longer T' that is an incr. subseq. and starts with A[k], then consider S'=A[j].T' would be a subsequence by construction and also increasing since A[j] < A[k] (first element of T') and T' itself is increasing. Further, S' would have a longer length than S which contradicts the assumption that S is the longest incr. subseq. in A[j...n] starting with A[j]. <*End of claim*>
   Therefore, LIS2(j) = 1 (for A[j]) + max_k LIS2(k) where the max is taken over all i s.t. A[j] < A[i]. This justifies the recursive formula.
   If there is no such k, then A[j] is the only correct subsequence that is increasing and starts with A[j]. Hence, LIS2(j)=1 in that case.

3. Describe a memoization structure (need not always be arrays/tables)

   1-D array L[0 … n]. L[i] will store the value of LIS2(i).

4. Give an algorithm/ordering for solving P for <u>all</u> values.

   Initialize L[n] = LIS2(n) = 1.

   Define a new sequence A' = (-infinity).A.

   For j=(n-1)...0, compute L[j] = LIS2[j] using the recursive formula on the sequence A'.

5. How to solve problem Q from values :

   LIS of A = L[0]-1. This is because LIS of A' will always start with A'[0] and the rest of that sequence must be the LIS of A.

6. What is space and time complexity for solving problem?

   Space complexity = O(n).

   Time-complexity = $O(n^2)$ since computing L[j] requires taking the max of at most (n-j) <= n values and there are O(n) entries in L.

7. For certain problems, how to obtain the optimal structure?

To compute the longest sequence itself, along with values also store pointers in L (this can be implemented by storing indexes) that point to other indexes of L. We denote the pointer associated with L[j] as L[j].p. Let Sj be the longest incr. subseq. in A[j … n] that starts with A[j]. Clearly, Sj must start with A[j]. L[j].p stores the index k s.t. A[k] is the next element in S after A[j].

These pointers can be computed while calculating the values in L[].
L[j].p = NULL when there is no k in j+1 … n s.t. A[j] < A[k]
L[j].p = argmax_k\{ 1 + LIS2(k) : k s.t. A[j] < A[k]\} otherwise

Finally, to print the LIS of A:

```
i=0
While (L[i].pointer != NULL) {
    print A[L[i].pointer];
    i=L[i].pointer;
}
```

This prints all the elements of A as we trace the pointers starting from L[0] until we hit a NULL. Note that A'[0] itself is not printed which is the correct behaviour since A'[0] is not part of A.

LIS of 3,1,4,1,5,9,2,6

Index j	0	1	2	3	4	5	6	7	8
A[j]	-9999	3	1	4	1	5	9	2	6
LIS2[j]	5	4	4	3	3	2	1	2	1