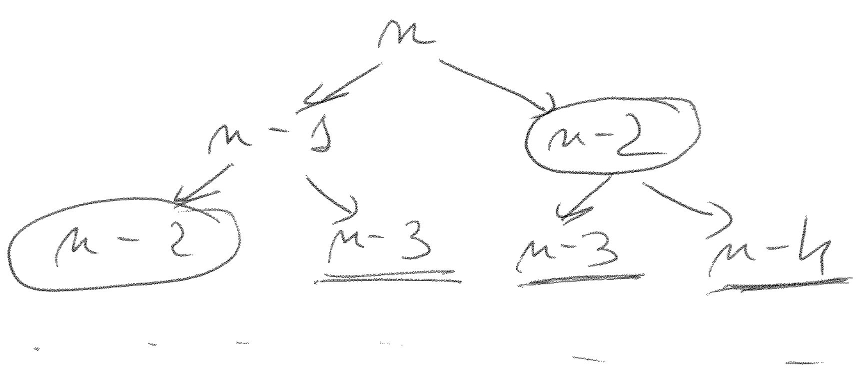
COMP250 - Lecture 30

Dynamic programming

Algorithm Fibonacci (n)

  if $n \leq 1$ return 1

  return Fibonacci ($n-1$) + Fibonacci ($n-2$)



$n-1$ levels complexity will be ballpark $O(2^n)$

(actually a bit better)

Common subproblems in different subtrees but results are re-computed every time.

Memoize results of previous computations

- ↳ increase memory cost a bit (poly increase)
- ↳ reduce computation time a lot (exp / combinatorical → poly)

Algorithm FibDynProg (n)

Array: int F[n+1]

F[0] = 1; || "Leaves" of prev tree → go up

F[1] = 1;

for (int i = 2 to n)

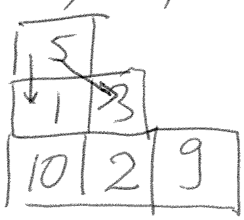
    F[i] = F[i-1] + F[i-2];

return F[n]; → O(n) memory

O(n) running time

Candy Maze problem (Midterm 2010)

n x n array filled in bottom half



sum of elements on path is as large as possible

Algorithm CandyMaze (a, i, j, n)

if (i == n-1) (last row)

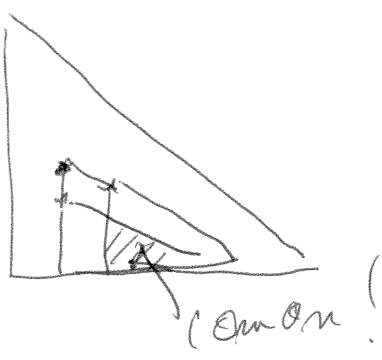
    return a[i][j]

else

    return a[i][j] + max (~~(a, i+1, j, n)~~

        CandyMaze(a, i+1, j, n),

        CandyMaze(a, i+1, j+1, n));



Algorithm CandyMazeDynProg (a, n) → Iterative

```
int result[n][n];
// bottom row same as in a
O(n) for (int j = 0; j < n; j++)
        result[n-1][j] = a[n-1][j];
for (int i = n-2; i ≥ 0; i--)   // going up
    for (int j = 0; j < i; j++)
        result[i][j] = a[i][j] + max(a[i+1, j],
                                    a[i+1][j+1]));
return result[0][0];
```

→ $O(n) \times O(n) = O(n^2)$

Extra n^2 memory (results array) but big reduction in time!

At row i after the loops all elems are value of optimal path below. → loop invariant

Making change: X amt of money

\$1, ~~\$~~ 25c, 10c, 5c, 1c

Convert X into a set of coins → as few coins as possible

Max of \$1 → n_1

$x = X - n_1$ → Max of 25c

...

"Greedy" algorithm: only X and cit. largest denomination matter!

Array C[K] of denominations

10c, 6c, 1c ~~\$~~ $X = 12c$

2 6c → optimal

Greedy solution: 10c + 2 * 1c → 3 coins
not optimal!

If greedy works → awesome! But this is not always true

$Opt(0) = 0$

$\underbrace{X - C_j}$ $j = 1 \dots K$

$$\underline{Opt}(x) = 1 + \min_j \left[\underline{Opt}(x - c_j) \right]$$

what is the best we could get with this set?

add 1 coin

(for loop)

greedy alg uses largest coin so far

Algorithm MakeChange (C, K, x)

int Opt[n+1] dep on X

Opt[0] = 0;

for (int i = 1 to n) // min Opt[x - C[j]]

m ← Large int   mini upto i

for (j = 0 to K-1)

if (C[j] ≤ i) then mini = min(mini, Opt[x-C[j]])

amt that we still have to make

Opt[i] = 1 + mini

O(n*K) (x is $+c ⇒ n = 100x)

Greedy alg would only loop over C ⇒ O(K)!

Longest Increasing Subsequence

1 3 2 5 4 8 1 9

Subset of array increasing $i < j$

$a[i] < a[j]$

as long as possible

For all lengths k:

For all possible subsequences

<u>Check</u> if order condition is satisfied

$\binom{n}{k}$

Loop over a large space, check each element

"Combinatorial" space

"Guess and check"