Dynamic Programming

Content adapted from Kevin Wayne and Robert Sedgewick

Fibonacci numbers again

- 0, 1, 1, 2, 3, 5, 8, 13,

$$F_i = \begin{cases} 0, i = 0 \\ 1, i = 1 \\ F_{i-1} + F_{i-2}, i > 1 \end{cases}$$

- From ESC180: naïve recursive approach takes

$O\left(fib(n)\right) = O(1.61^n)$ time

Memoization

- Maintain a table of values that were already computed

```
def fib(n, mem = {}):
  if n in mem:
    return mem[n]
  mem[n] = fib(n-1, mem) + fib(n-2, meme)
  return mem[n]
```

Memoization: runtime analysis

```
def fib(n, mem = {}):
  if n in mem:
    return mem[n]
  mem[n] = fib(n-1, mem) + fib(n-2, mem)
  return mem[n]
```

- Only compute each entry in `mem` once
- `fib(n-1) + fib(n-2)` does not produce internal calls to `fib`
- Compute n entries in `mem`, each taking constant time
- O(n) time

Dynamic programming approach

- Solve subproblems, and store the solutions to those subproblems
- Use solutions to small subproblems to compute solutions to larger problems

```
def fib_iter(int n):
  fib_list = [0] * n
  fib_list[0:2] = [0, 1]
  for i in range(2, n+1):
    fib_list[i] = fib_list[i-1] + fib_list[i-2]
```

Dynamic programming: outline

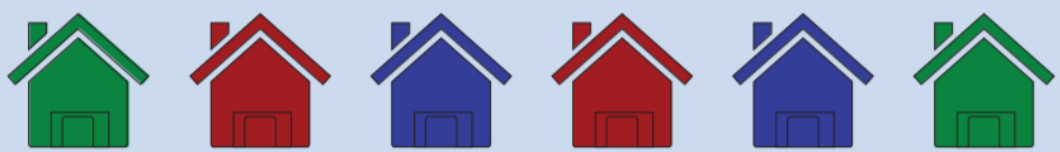
- Divide a complex problem into a number of simpler overlapping problems
  - n+1 problems, where the i-th problem is the i-th Fibonacci number
- Define a relationship between solutions to more complex problems and solutions to simpler problems
  - Can compute F_i using F_{i-1} and F_{i-2}

$$F_i = \begin{cases} 0, i = 0 \\ 1, i = 1 \\ F_{i-1} + F_{i-2}, i > 1 \end{cases}$$

- Store solutions to each subproblem, solving each subproblem once
  - Use `fib_list` to store solutions
- Use stored solutions to solve the original problem

Painting houses

- Goal: paint a row of *n* houses red, green, or blue s.t.
  - Total cost is minimized. `cost(i, col)` is the cost to paint the i-th hous in colour `col`
  - No two adjacent houses have the same colour

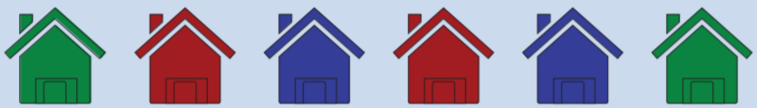


	1	2	3	4	5	6
cost(i, red)	7	$\boxed{6}$	7	$\boxed{8}$	9	20
cost(i, green)	$\boxed{3}$	8	9	22	12	$\boxed{8}$
cost(i, blue)	16	10	$\boxed{4}$	2	$\boxed{5}$	7

cost to paint house i the given color
(3 + 6 + 4 + 8 + 5 + 8 = 34)

Subproblems

- R(i): min cost to paint the first i houses, with the i-th house painted red
- G(i): min cost to paint the first i houses, with the i-th house painted green
- B(i): min cost to paint the first i houses, with the i-th house painted blue

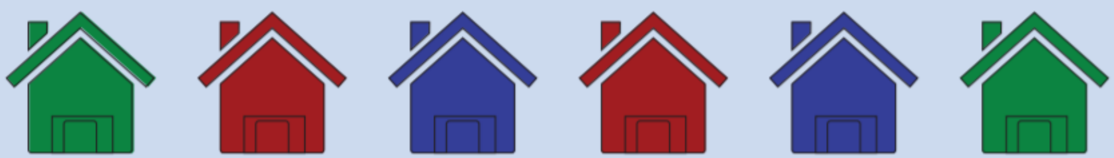


	1	2	3	4	5	6
$cost(i, red)$	7	$\boxed{6}$	7	$\boxed{8}$	9	20
$cost(i, green)$	$\boxed{3}$	8	9	22	12	$\boxed{8}$
$cost(i, blue)$	16	10	$\boxed{4}$	2	$\boxed{5}$	7

cost to paint house i the given color
(3 + 6 + 4 + 8 + 5 + 8 = 34)

Relationship between problems

$$R(i) = cost(i, red) + \min(G(i-1), B(i-1))$$
$$G(i) = cost(i, green) + \min(R(i-1), B(i-1))$$
$$B(i) = cost(i, blue) + \min\big(R(i-1), G(i-1)\big)$$
$$Cost(i) = \min(R(i), G(i), B(i))$$



	1	2	3	4	5	6
$cost(i, red)$	7	6	7	8	9	20
$cost(i, green)$	3	8	9	22	12	8
$cost(i, blue)$	16	10	4	2	5	7

cost to paint house i the given color
(3 + 6 + 4 + 8 + 5 + 8 = 34)

Making change

- Given a set of coin denominations (e.g., [1, 5, 10, 25, 100, 200] for Canadian currency*), and an amount of money, find the way to represent the amount using the least number of coins
- For Canadian denominations, picking the largest coin you can use always works
- But consider [1, 4, 5, 10] and trying to make 8
    - 5 is the largest coin we can use, but 5+1+1+1 is worse than 4 + 4

*Back when the penny was in circulation

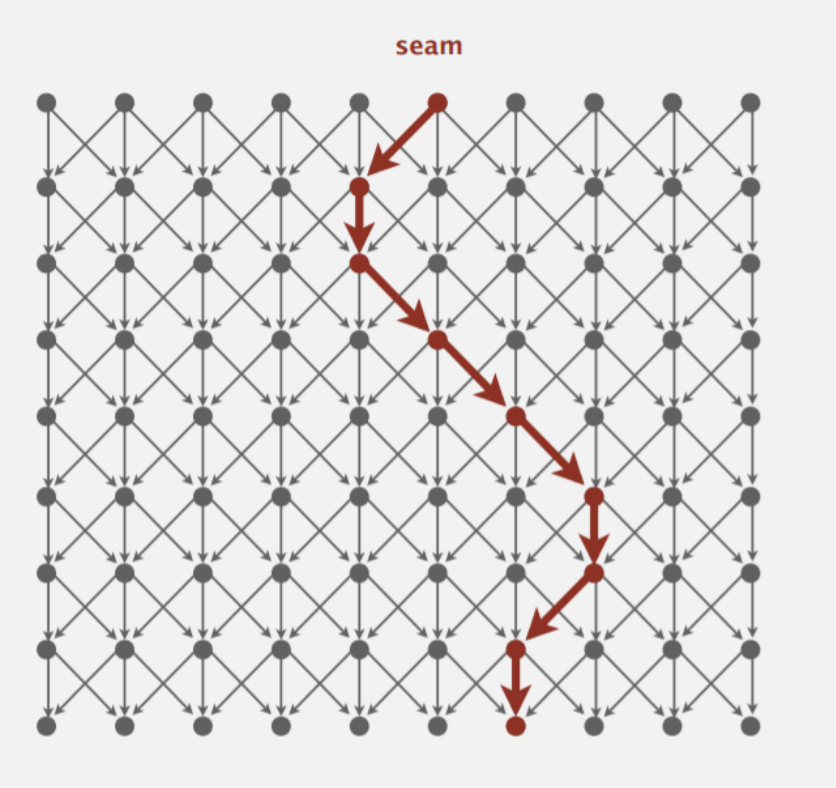
Making change: dynamic programming

- Problem: given n coin denominations $\{d_1, d_2, \ldots, d_n\}$ and a target value V, find the least number of coins needed to make change for V
- Subproblems: $OPT(v)$: the least number of coins needed to make change for v
- To make v using denomination d_i, use $OPT(v - d_i) + 1$ coins
  - Try every possible $d_i \leq v$
- Dynamic programming recurrence:

$$OPT(v) = \begin{cases} \infty, v < 0 \\ 0, v = 0 \\ \min\limits_{1 \leq i \leq n} \big(1 + OPT(v - d_i)\big), v > 0 \end{cases}$$

Seam carving

- Problem: find the min energy path (sum of energies at nodes) from top to bottom

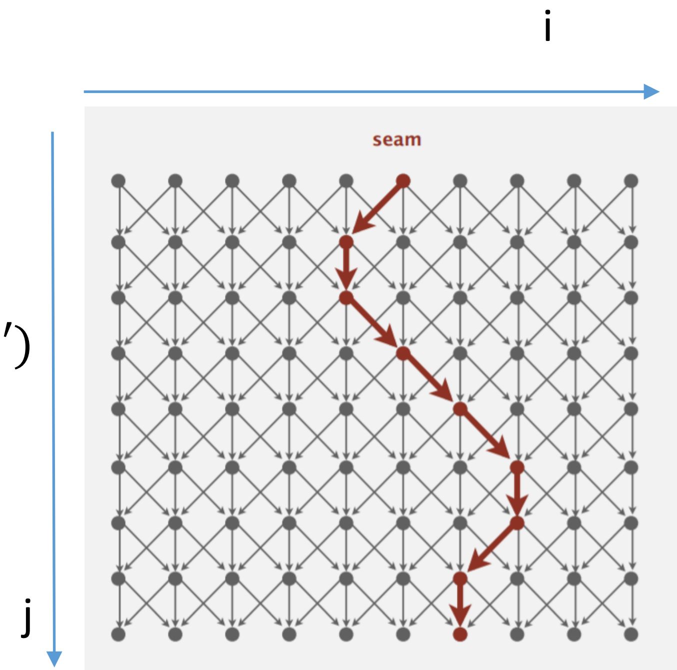


Seam carving

Subproblem: min. energy path from any top node to node (j, i)

$$OPT\big((j,i)\big) = E\big((j,i)\big) + \min_{i' \in \{i-1, i, i+1\}} OPT((j-1, i'))$$

(Need to take care of edge cases where the seam is near the boundary)



(j-1, i-1) (j-1, i) (j-1, i+1)
(j, i)