Dynamic Programming

250H

Example: $a_n = a_{n-1} + a_{\lfloor\sqrt{n}\rfloor}$

Recursion

```
example(n):
    if (n = 0)
        return 0
    else
        return example(n) + example(floor(sqrt(n)))
```

Example: $a_n = a_{n-1} + a_{\lfloor\sqrt{n}\rfloor}$

Dynamic Programing(Bottom Up):

```
example(n):
    a = array of length n
    a[0] = 0
    for i = 1 to n
        a[n] = a[n-1] + a[floor(sqrt(n))]
    return a[n]
```

Example: $a_n = a_{n-1} + a_{\lfloor\sqrt{n}\rfloor}$

Dynamic Programing with Memoization (Top Down):

```
example(n):
    a = array of length n
    if (n = 0)
        return 0
    else
        a[n] = a[n-1] + a[floor(sqrt(n))]
    return a[n]
```

Dynamic Programing

- Solves problems by combining the solutions to subproblems
  - When the subproblems overlap

Dynamic Programing

- Solves problems by combining the solutions to subproblems
  - When the subproblems overlap
- Solves each sub sub problem just once then saves its answer in a table

Dynamic Programing

- Solves problems by combining the solutions to subproblems
    - When the subproblems overlap
- Solves each sub sub problem just once then saves its answer in a table
- Typically Dynamic Programing is applied to optimization problems
    - Each solution has a value and we want to find a solution with the optimal value
    - This is *an* optimal solution to the problem
        - There may be several

Developing a Dynamic-Programing Algorithm

1. Characterize the structure of an optimal solution

Developing a Dynamic-Programing Algorithm

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution

Developing a Dynamic-Programing Algorithm

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution, typically in a bottom-up fashion

Developing a Dynamic-Programing Algorithm

1. Characterize the structure of an optimal solution
2. Recursively define the value of an optimal solution
3. Compute the value of an optimal solution, typically in a bottom-up fashion
4. Construct an optimal solution from computed information

If we only need the value of an optimal solution, and not the solution itself, then we can omit step 4.

Memoization

- The word really is memoization, not memorization
    - Comes from memo

Memoization

- The word really is memoization, not memorization
  - Comes from memo
- A memoized recursive algorithm maintains an entry in a table for the solution to each subproblem

Memoization

- The word really is memoization, not memorization
  - Comes from memo
- A memoized recursive algorithm maintains an entry in a table for the solution to each subproblem
- Each table entry initially contains a special value to indicate that the entry has yet to be filled in

Memoization

- The word really is memoization, not memorization
  - Comes from memo
- A memoized recursive algorithm maintains an entry in a table for the solution to each subproblem
- Each table entry initially contains a special value to indicate that the entry has yet to be filled in
- When the subproblem is first encountered as the recursive algorithm unfolds, its solution is computed and then stored in the table

Memoization

- The word really is memoization, not memorization
  - Comes from memo
- A memoized recursive algorithm maintains an entry in a table for the solution to each subproblem
- Each table entry initially contains a special value to indicate that the entry has yet to be filled in
- When the subproblem is first encountered as the recursive algorithm unfolds, its solution is computed and then stored in the table
- Each subsequent time that we encounter this subproblem, we simply look up the value stored in the table and return it