Dynamic Programming 2

`http://bit.ly/VTProgDP2`

Review

Project Euler 67

3

7 4

2 **4** 6

8 5 **9** 3

Project Euler 67

- Maximize sum down a path
- Given a location, can pick to go left or right

Recurrence

$$D(i,j) = \begin{cases} 0 & \text{if } i = N \\ A_{i,j} + \mathbf{max} \begin{cases} D(i+1,j) \\ D(i+1,j+1) \end{cases} & \text{otherwise} \end{cases}$$

```java
static int M(int i, int j) {
    if (i >= pyramid.length)
        return 0;
    return pyramid[i][j] + Math.max(
            M(i + 1, j),
            M(i + 1, j + 1)
    );
}
```

Naive recursive code

```java
static Integer[][] table = new Integer[101][101];
static int M(int i, int j) {
    if (table[i][j] != null)
        return table[i][j];
    if (i >= pyramid.length)
        return 0;
    int answer = pyramid[i][j] +
                 Math.max(M(i + 1, j), M(i + 1, j + 1));
    table[i][j] = answer;
    return table[i][j];
}
```

```java
for (int row = pyramid.length - 1; row > 0; row--) {
    for (int col = 0; col < pyramid[row].length - 1; col++) {
        pyramid[row - 1][col] = Math.max(
            pyramid[row - 1][col] + pyramid[row][col],
            pyramid[row - 1][col] + pyramid[row][col + 1]
        );
    }
}
```

Bottom-Up style

Dynamic Programming 2

How to find a recurrence

- Hard part is usually not writing the code (after you've practiced enough)
- Hard part **is** finding what recurrence

How to find a recurrence

- What is the objective?
  - Minimize
  - Maximize (like PE 67, the pyramid)
  - Count (like the coins problem)
- How do we represent the state?
  - Usually with some integers denoting the “sub-problems”
  - For PE 67, we use i, j to be the row, column pair
  - Frequently use array indices, but can be many things

How to find a recurrence

- Finally, what are the options/choices?
  - How do we reduce this into sub problems?
  - For PE 67, this is going left/right down the pyramid
  - These problems must be smaller than your current one!
  - They may depend on some state, not necessarily the same for every state

Dynamic programming

- Given these, we can write a basic recursive function
- Parameters are simply the state
- Find the base case based on the state
- Make recursive calls based on the choices/options
- Combine based on the objective (max, min, add, multiply)
- Add memoization
- Done!

Edit Distance

Edit distance (Levenshtein distance)

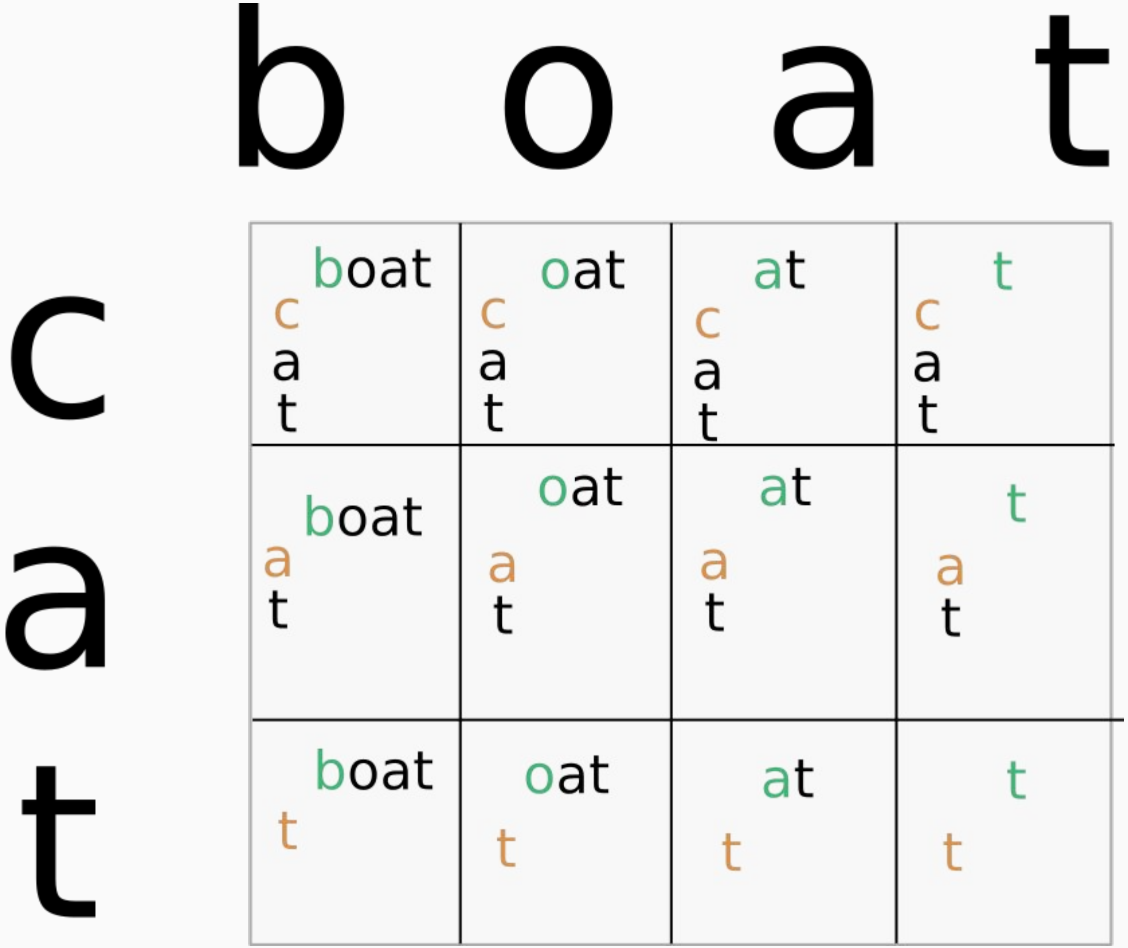
- Comparison of two strings based on how many “edits” to convert one to the other
- Edits are
    - Substitute - replace one character with another
    - Delete a single character
    - Add a single character

Edit distance

- The edit distance, then, is the minimum number of edits to convert a string A into a string B
- For example,
  - “cat” and “cot” are edit distance 1 away, because we can do a single replace
  - “cat” and “boat” are edit distance 2 away, because we can add “b” and replace “c” with “o”

Edit distance

- Want an efficient algorithm for edit distance of two strings
- Any ideas?
- **Hint**: it’s dynamic programming



Illustration

	b	o	a	t
c	boat c a t	oat c a t	at c a t	t c a t
a	boat a t	oat a t	at a t	t a t
t	boat t	oat t	at t	t t 0

Illustration

	b	o	a	t
c	boat cat 2	oat cat 1	at cat 1	t cat 2
a	boat at 2	oat at 1	at at 0	t at 1
t	boat t 3	oat t 2	at t 1	t t 0

Illustration

Finding a recurrence

- What is the objective?
- Count? Maximize? Minimize?

Objective

- What are we minimizing?
- Some arbitrary "cost"
- Number of edits we have made

Finding a recurrence

- How do we represent our state?

Finding a recurrence

- We need at least two integers
- Why?

State

- Two integers i, j representing indices into both strings
- Can think of D(i, j) as working on the substrings A[i….], B[j…]

Choices

- Looking at a single state (a single value of i, j), what are the choices?

Choices

- Consider some cases
- What if A[i] == B[j]?
- What if it doesn’t?

Choices

- If A[i] == B[j], then we don’t have to do anything!
- Otherwise we can try some things:
  - Replace the current character (change A[i] <- B[j])
  - Delete the current character in A
  - Add a character at location i in A
- Each of these has “cost” 1, because they count as 1 edit

How do we represent the choices?

- Simple case, A[i] == B[j]:
  - Just the cost of “fixing” the rest of the string
  - D(i + 1, j + 1)
- What about the other cases?

Representing choices (as recursion)

- If we replace a character, we can assume we replace it correctly (change A[i] <- B[j] or the other way around)
- This has 1 cost, plus the cost of fixing the rest of the string
  - $1 + D(i + 1, j + 1)$

Representing choices (as recursion)

- What about deletion or addition?

Representing choices (as recursion)

- Deleting a character in A
  - D(i + 1, j) advances a character in A, but not in B
  - Cost of 1
- Adding a character to A
  - Assume we add the right character (not just a random one)
  - D(i, j + 1) advances a character in B, not A
  - Remember cost of 1

Basic recurrence

- How do we combine the choices?

Basic recurrence

$$D(i,j) = \begin{cases} D(i+1, j+1) & \text{if } A_i = B_j \\ 1 + \mathbf{min} \begin{cases} D(i+1, j+1) \\ D(i+1, j) \\ D(i, j+1) \end{cases} & \text{otherwise} \end{cases}$$

Basic recurrence

- Are we missing any cases?

Basic recurrence

- When does it end?

$$D(i,j) = \begin{cases} \mathbf{max}(N-i, M-j) & \text{if } i = N \text{ or } j = M \\ D(i+1, j+1) & \text{if } A_i = B_j \\ 1 + \mathbf{min} \begin{cases} D(i+1, j+1) \\ D(i+1, j) \\ D(i, j+1) \end{cases} & \text{otherwise} \end{cases}$$

With base case

“Basic” recurrence

- Got quite large quite quick
- Show me the code!

```java
static int recur(int i, int j) {
    if (i == a.length())
        return b.length() - j;
    if (j == b.length())
        return a.length() - i;

    if (a.charAt(i) == b.charAt(j)) {
        return dp(i + 1, j + 1);
    }
    return 1 + min(dp(i + 1, j),     // del
                   dp(i, j + 1),     // add
                   dp(i + 1, j + 1) // replace
              );
}
```

Dynamic programming

- What's the runtime without memoization?
  - Very slow is the answer
- How do we add memoization?

```java
static Integer[][] cache = new Integer[a.length()][b.length()];
static int dp(int i, int j) {
    if (i == a.length())
        return b.length() - j;
    if (j == b.length())
        return a.length() - i;

    if (cache[i][j] != null)
        return cache[i][j];

    int ans;
    if (a.charAt(i) == b.charAt(j)) {
        ans = dp(i + 1, j + 1);
    } else {
        ans = 1 + min(dp(i + 1, j),     // del
                      dp(i, j + 1),     // add
                      dp(i + 1, j + 1) // replace
                      );
    }
    cache[i][j] = ans;
    return ans;
}
```

Code for recurrence

Dynamic programming

- Runtime with memoization?
- O(N * M)

Calculating runtime for DP

- Rough rule of thumb: product of state space * runtime of body
- For this example, state is i, j bounded by N, M
- Body runs in O(1) ignoring subcalls
- Runtime: O(N)*O(M)*O(1) = O(N M)

Problems for today

- https://contest.spruett.me/problems
- Both string based DP

Other problems (if you're bored)

- https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=534&page=show_problem&problem=3956