Classic String Dynamic Programming

Edit Distance and Palindromes

Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Fall, 2022

Objectives

- Use DP to determine the edit distance between two strings.
- Use DP to find all palindromic substrings.
- Learn the word “aibohphobia”

The Problem

- Given two strings s and t, how many "edits" does it take to tranform one to another?
  - Edit = insert, delete, or change.
  - Usually each of these "costs" one unit.
- Usually called the Levinstein Distance
- Examples:
  - changing `DATA` to `BETA` needs 2 steps.
  - changing `ETA` to `BETA` needs 1 step.
  - changing `GRETA` to `BETA` needs 2 steps.

The Näive Algorithm

Base Cases

```
1  // Thanks, Wikipedia!
2  int LD(string s, int len_s, string t, int len_t)
3  {
4    int cost;
5
6    /* base case: empty strings */
7    if (len_s == 0) return len_t;
8    if (len_t == 0) return len_s;
9
10   /* test if last characters of the strings match */
11   if (s[len_s-1] == t[len_t-1])
12       cost = 0;
13   else
14       cost = 1;
```

The Näive Algorithm, ctd

Recursive Case

```
15    /* return minimum of delete char from s,
16                          delete char from t,
17                      and delete char from both */
18    return minimum(LD(s, len_s - 1, t, len_t    ) + 1,
19                   LD(s, len_s    , t, len_t - 1) + 1,
20                   LD(s, len_s - 1, t, len_t - 1) + cost);
21  }
```

Dynamic Programming using Memoization

Base Cases

```
1  int LD(const char *s, int len_s, const char *t, int len_t)
2  {
3    vvi d = vvi(len_s + 1, vi(len_t +1));
4    int cost;
5
6    for(int i=0; i<=len_s; ++i)
7       d[i][0] = i;
8
9    for(int i=0; i<=len_t; ++i)
10      d[0][i] = i;
```

Dynamic Programming using Memoization, ctd

Memoized Part

```
11    for(int i=1; i<=len_s; ++i)
12      for(j=1; j<=len_t; ++j) {
13          cost = s[i] == t[j] ? 0 : 1;
14
15          d[i,j] = minimum(d[i-1][j] + 1,
16                           d[i][j-1] + 1,
17                           d[i-1][j-1] + cost);
18          }
19     return d[len_s][len_t];
20  }
```

The Problem

Given a string s, find all the palindromic sub-strings.

- babba has two non-trivial palindormic substrings:
  - bb and abba

The algorithm

- Create a DP array $dp[|s|][|s|]$).
  - $dp[i][j]$ indicates if substring $s[i..j]$ is a palindrome.
  - Initialize diagonal to 1
- For each pair i, j, if $s[i] == s[j]$ then check if s[i+1,j-1]$ is also a palindrome.
- Must iterate over smaller gap sizes first.

Code

```
1  int numPalindromes(stirng s) {
2    int i,j,gap,count;
3    vvb dp(s.length(),vb(s.length()),false);
4
5    count = 0;
6    for(i=0; i<s.length(); ++i)
7      dp[i] = true; // one character palindroms
8
9    // base casee: two character palindromes
10   for(i=1; i<s.length(); ++i)
11     if (s[i-1] == s[i]) dp[i-1][i] = true;
```

Code, ctd

```
12      for(gap=2; gap<s.length()-1; ++gap)
13        for(j=gap, i=0; i<s.length(); ++i, ++j)
14          if (s[i] == s[j]))
15              if (dp[i+1][j-1]) {
16                  ++count;
17                  dp[i][j] = true;
18              }
19      return count;
20  }
```

Example

- Example for babba

Matrix

		b	a	b	b	a
	b					
	a					
	b					
	b					
	a					

Action

- Start with empty matrix

Example

- Example for `babba`

Matrix

	b	*a*	*b*	*b*	*a*
b	1				
a		1			
b			1		
b				1	
a					1

Action

- Start with empty matrix
- Initialize diagonal

Example

- Example for babba

Matrix

	b	*a*	*b*	*b*	*a*
b	1				
a		1			
b			1	1	
b				1	
a					1

Action

- Start with empty matrix
- Initialize diagonal
- Gap = 2, bb

Example

- Example for babba

Matrix

	b	*a*	*b*	*b*	*a*
b	1		1		
a		1			
b			1	1	
b				1	
a					1

Action

- Start with empty matrix
- Initialize diagonal
- Gap = 2, bb
- Gap = 3, bab

Example

- Example for babba

Matrix

	b	*a*	*b*	*b*	*a*
b	1		1		
a		1			1
b			1	1	
b				1	
a					1

Action

- Start with empty matrix
- Initialize diagonal
- Gap = 2, bb
- Gap = 3, bab
- Gap = 4, abba