Classic String DP

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Objectives

Your Objectives:

- Compute the edit distance of two strings
- Compute the longest common subsequence of two strings.

Edit Distance

You are given two strings, and want to transform one to the other. You have three operations:

- Delete a character
- Insert a character
- Replace a character.

E.g., changing `DATA` to `BETA` needs 2 steps.
E.g., changing `ETA` to `BETA` needs 1 step.
E.g., changing `GRETA` to `BETA` needs 2 steps.

The Näive Algorithm

```c
// Thanks, Wikipedia!
int LD(const char *s, int len_s, const char *t, int len_t)
{
  int cost;

  /* base case: empty strings */
  if (len_s == 0) return len_t;
  if (len_t == 0) return len_s;

  /* test if last characters of the strings match */
  if (s[len_s-1] == t[len_t-1])
      cost = 0;
  else
      cost = 1;

```

Use DP!

```
// Thanks, Wikipedia!
int LD(const char *s, int len_s, const char *t, int len_t)
{
  int d[len_s+1][len_t+1];
  int cost;

  for(int i=0; i<=len_s; ++i)
     d[i][0] = i;

  for(int i=0; i<=len_t; ++i)
     d[0][i] = i;

  for(int i=1; i<=len_s; ++i)
    for(j=1; j<=len_t; ++j) {
      cost = s[i] == t[j] ? 0 : 1;
```

Longest Common Subsequence

```
int LCS(char *s, int len_s, char *t, int len_t) {

  if (len_s == 0 || len_t == 0)
      return 0;

  if (s[len_s-1] == t[len_t-1])
      return 1 + LCS(s,len_s-1,t,len_t-1)
  else
      return max(LCS(s,len_s,t,len_t-1),
                 LCS(s,len_s,t,len_t-1));
}
```

DP Solution

```
// Adapted from code on geeksforgeeks.com
int LCS(char *s, int len_s, char *t, int len_t) {

  int d[len_s][len_t];

  if (len_s == 0 || len_t == 0)
     return 0;

  for(int i=0; i<=len_s; i++)
    for(int j=0; j<=len_t; j++) {
      if (i==0 || j == 0) d[i][j] = 0;
      else if (s[i-1] == t[j-1])
         d[i][j] = d[i-1][j-1] + 1
      else
         d[i][j] = max(d[i-1][j],d[i][j-1]);
```