Lab04-Dynamic Programming

CS214-Algorithm and Complexity, Xiaofeng Gao, Spring 2019.

∗ If there is any problem, please contact TA Jiahao Fan.

∗ Name:Lynn Xiao Student ID:_________ Email: ____________

1. Given a positive integer n, find the least number of perfect square numbers (e.g., 1, 4, 9, . . .) which sum to n.

   (a) Assume that OPT(a) = the least number of perfect square numbers which sum to a. Please write a recurrence for OPT(a).

   Solution.

$$OPT(a) = \begin{cases} 0 & a = 0 \\ \min_{1 \le b \le \sqrt{a}}\{1 + OPT(a - b^2)\} & otherwise \end{cases}$$

□

   (b) Base on the recurrence, write down your algorithm in the form of *pseudo code*.

   Solution.

Algorithm 1: Tabulation

Input: An integer n

Output: The least number of perfect square numbers $M[n]$

```
$M[0] \leftarrow 1$;
for $i \leftarrow 1$ to $n$ do
    $M[i] \leftarrow 1$;
    for $j \leftarrow 2$ to $\sqrt{i}$ do
        $M[i] = min\{M[i], 1 + M[j - i^2]\}$
    return $M[n]$
```

□

2. Given an input string s (could be empty, and contains only lowercase letters a-z) and a pattern p (could be empty, and contains only lowercase letters a-z and characters like '?' or '*'), please design an algorithm using dynamic programming to determine whether s matches p based on the following rules:

   - '?' matches any single character.
   - '*' matches any sequence of characters (including the empty sequence).
   - The matching should cover the entire input string (not partial).

   Assume $m = len(s)$ and $n = len(p)$. Output **true** if s matches p, or **false** otherwise.

   (a) Assume that ANS(i, j) means whether the first i ($0 \le i \le m$) characters of s match the first j ($0 \le j \le n$) characters of p. Please write a recurrence for ANS(i, j).

Solution.

$$ANS(i,j) = \begin{cases} true & i = 0, j = 0 \\ false & i = 0, j = 0, p[j] \neq' *' \\ ANS(i, j-1) & i = 0, j = 0, p[j] =' *' \\ false & i \neq 0, j = 0 \\ false & s[i] \neq p[j] \quad \textbf{and} \quad p[j] \neq'?' \quad \textbf{and} \quad p[j] \neq' *' \\ ANS(i-1, j-1) & s[i] = p[j] \quad \textbf{or} \quad p[j] ='?' \\ ANS(i, j-1) \cup ANS(i-1, j) & p[j] =' *' \end{cases}$$

□

(b) Base on the recurrence, write down your algorithm in the form of *pseudo code.*

Solution.

```
Algorithm 2: Tabulation
  Input: Two string $s$ and $p$,the length of the string $m = len(s)$, $n = len(p)$
  Output: whether $s$ matchs $q$: ANS[m][n]

$ANS[0][0] \leftarrow true$;
for $i \leftarrow 1$ to $m$ do
    $ANS[i][0] \leftarrow false$
for $i \leftarrow 0$ to $m$ do
    for $j \leftarrow 1$ to $n$ do
        if $i$=0 then
            if $p[j] =' *'$ then
                $ANS[i][j] \leftarrow ANS[i][j-1]$
            else $ANS[i][j] \leftarrow false$
        else if $s[i] = p[j]$ or $p[j] ='?'$ then
            $ANS[i][j] \leftarrow ANS[i-1][j-1]$
        else if $p[j] =' *'$ then
            $ANS[i][j] \leftarrow \max\{ANS[i-1][j], ANS[i][j-1]\}$
            else $A[i][j] \leftarrow false$
return $ANS[m][n]$
```

□

(c) Analyze the time and space complexity of your algorithm.

Solution.
(i) Time complexity: We need $O(1)$ time to compute each $A[i][j]$, so the total time complexity is $O(mn)$.

(ii) Space complexity: We use a two-dimensional array to store the result, so the space complexity is $O(mn)$. □

3. Recall the *String Similarity* problem in class, in which we calculate the edit distance between two strings in a sequence alignment manner.

   (a) Implement the algorithm combining dynamic programming and divide-and-conquer strategy in C/C++ with time complexity $O(mn)$ and space complexity $O(m+n)$. (The template *Code-SequenceAlignment.cpp* is attached on the course webpage).

(b) Given $\alpha(x, y) = |ascii(x) - acsii(y)|$, where $ascii(c)$ is the ASCII code of character c, and $\delta = 13$. Find the edit distance between the following two strings.

$$\begin{aligned} X[1..60] = {} & PSQAKADIETSJPWUOMZLNLOMOZNLTLQ \\ & CFQHZZRIQOQCOCFPRWOUXXCEMYSWUJ \end{aligned}$$

$$\begin{aligned} Y[1..50] = {} & SUYLVMUSDROFBXUDCOHAAEBKN \\ & AAPNXEVWNLMYUQRPEOCQOCIMZ \end{aligned}$$

Solution.
By running the code we can get the distance is 439. □

(c) (Bonus) Visualize the shortest path found in (b) on the corresponding edit distance graph using any tools you like.

Remark: You need to include your .cpp, .pdf and .tex files in your uploaded .rar or .zip file.