Lab04-Dynamic Programming

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

∗ If there is any problem, please contact TA Jiahao Fan.
∗ Name:Yu Wu Student ID:517021910915 Email: 1398035730@qq.com

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).

   $$OPT(a) = \begin{cases} 1, & a = i^2 \\ \min\limits_{j^2 \le a}\{OPT(a - j^2) + 1\}, & \text{otherwise} \end{cases}$$

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

Algorithm 1: perfectSquare

Input: An integer n
Output: m(the least number of perfect square numbers which sum to n)

```
1  for j ← 1 to n do
2      M[j] = empty;
3  Run Find-Solution(n);
4  Function M-Find-Solution(j):
5      if M[j] is empty then
6          if [j^{1/2}]^2 == j then
7              M[j] = 1;
8          else
9              M[j] = min_{i^2 ≤ a}{M − Find − Solution(j − i^2) + 1};
10     return M[j];
```

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).

   $$ANS(0, j) = \begin{cases} true, & j = 0 \\ ANS(0, j-1), & p[j-1] = \text{'*'} \\ false, & \text{otherwise} \end{cases}$$

$$ANS(i,0) = \begin{cases} true, & i = 0 \\ false, & \text{otherwise} \end{cases}$$

for $i, j > 0$,

$$ANS(i,j) = \begin{cases} ANS(i,j-1) \text{ or } ANS(i-1,j-1) \text{ or } ANS(i-1,j), & p[j-1] = \text{'*'} \\ ANS(i-1,j-1), & p[j-1] = \text{'?'} \\ ANS(i-1,j-1) \text{ and } (s[i-1] == p[j-1]), & \text{otherwise} \end{cases}$$

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

Algorithm 2: stringMatching

Input: 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 '*')

Output: **true** if s matches p, or **false** otherwise

```
1  m ← len(s);n ← len(p);
2  M[0] ← true;
3  for j ← 1 to n do
4      if p[j − 1] ==’*’ then
5          M[j] ← M[j − 1];
6      else
7          M[j] ← false;
8  for i ← 1 to m do
9      N[0] ← false;
10     for j ← 1 to n do
11         if p[j − 1] ==’*’ then
12             N[j] ←( M[j − 1] or M[j] or N[j − 1]);
13         else if p[j − 1] ==’?’ then
14             N[j] ← M[j − 1];
15         else
16             N[j] ← ( M[j − 1] and (s[i − 1] == p[j − 1]));
17     swap M[] and N[];
18 output N[n];
```

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

Solution. Time complexity : The algorithm computes every $ANS(i,j)$ $(0 \le i \le m, 0 \le j \le n)$once, so the time complexity is $O(mn)$.
Space complexity : For every $ANS(i,\cdot)$, it needs $ANS(i-1,\cdot)$ and itself, which result in the space complexity of $O(n)$. □

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.

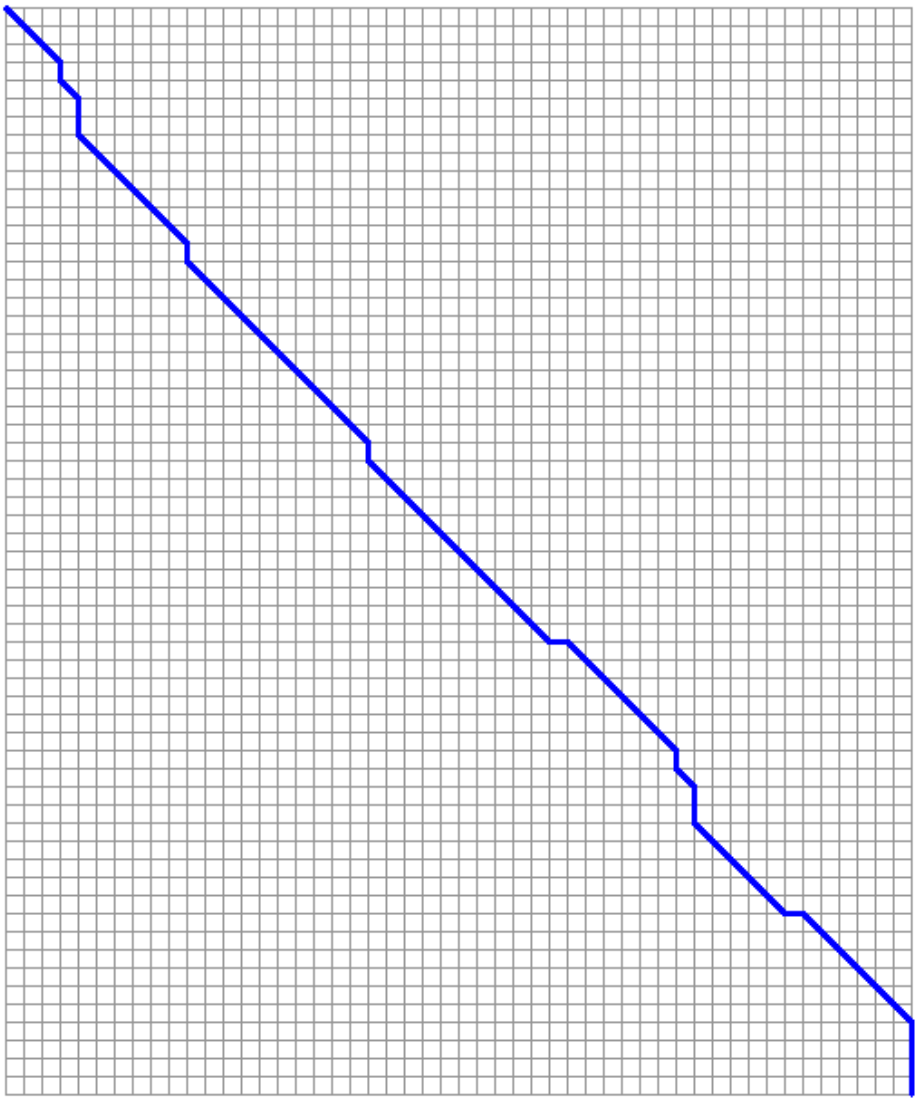
$$X[1..60] = PSQAKADIETSJPWUOMZLNLOMOZNLTLQ$$
$$CFQHZZRIQOQCOCFPRWOUXXCEMYSWUJ$$

$$Y[1..50] = SUYLVMUSDROFBXUDCOHAAEBKN$$
$$AAPNXEVWNLMYUQRPEOCQOCIMZ$$

Solution. Run *Code-SequenceAlignment.cpp* in (a) we get that the distance is 439. □

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

Solution. Using **Visisual Studio** and **Open CV**, I get the graph as follows. And the code is showed in *VisualizeThePath.cpp*.



□

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