Data Structures and Algorithms

First Draft

Topics : Dynamic Programming

Algorithm 1 Find the maximum profit that you can earn by doing at most one transaction

Ensure: One Based Indexing

```
1: function BUY_SELL(Arr, len)
   ▷ maxRight[i] denotes the max element to the right of the i − th element, including it
2:     maxRight[len] ← a[len]
3:     for i = len − 1 : 1 do
4:         maxRight[i] ← max(maxRight[i + 1], a[i])
   ▷ profit[i] denotes the max profit that you can earn if you can only buy at the i − th day
5:     for i = len : 1 do
6:         profit[i] ← (maxRight[i] − a[i])
7:     return Maximum element of Profit array
```

Algorithm 2 Find the length of the subarray with the maximum sum

Ensure: One Based Indexing

```
1: function KADANE(Arr, len)
   ▷ start[i] denotes the length of the maximum sum subarray which starts at the i − th index
2:     start[len] ← a[len]
3:     for i = len − 1 : 1 do
4:         start[i] ← max(a[i] + start[i + 1], a[i])
5:     return Maximum element of Start array
```

Algorithm 3 Find the length of the longest increasing subsequence

Ensure: One Based Indexing

```
1: function LIS(Arr, len)
   ▷ lis[i] denotes the length of the LIS which starts at the i − th index
2:     for i = len : 1 do
3:         lis[i] ← 1
4:     for i = len : 1 do
5:         for j = i + 1 : len do
6:             if a[j] > a[i] then
7:                 lis[i] ← max(lis[i], 1 + lis[j])
8:     return Maximum element of LIS array
```

Algorithm 4 Find the minimum edit distance to convert str_1 to str_2

Ensure: Zero Based Indexing

```
1: function EDIT_DISTANCE(str1, str2)
   ▷ edit[i][j] denotes the edit distance of the first i characters of str1 and the first j characters
   of str2
2:     edit[i][0] ← i                          ∀i        ▷ Delete i characters
3:     edit[0][j] ← j                          ∀j        ▷ Insert j characters
4:     for i = 1 : str1.len do
5:         for j = 1 : str2.len do
6:             insert ← 1 + edit[i][j − 1]
7:             delete ← 1 + edit[i − 1][j]
8:             replace ← 1 + edit[i − 1][j − 1]
9:             match ← edit[i − 1][j − 1]
10:            if str1[i] == str2[j] then
11:                edit[i][j] ← match
12:            else
13:                edit[i][j] ← min(insert, delete, replace)
14:    return edit[str1.len][str2.len]
```

Algorithm 5 Find the length of the longest comomon subsequence of 2 strings

Ensure: Zero Based Indexing

```
1: function LIS(str_1, str_2)
   ▷ lcs[i][j] denotes the longest common subsequence of the first i characters of str_1 and the first
   j characters of str_2
2:     lcs[i][0] ← 0                                  ∀i        ▷ The first string is empty
3:     lcs[0][j] ← 0                                  ∀j        ▷ The second string is empty
4:     for i = 1 : str_1.len do
5:         for j = 1 : str_2.len do
6:             exclude_top ← lcs[i − 1][j]
7:             exclude_bot ← lcs[i][j − 1]
8:             exclude_both ← lcs[i − 1][j − 1]
9:             match ← 1 + lcs[i − 1][j − 1]
10:            if str_1[i] == str_2[j] then
11:                lcs[i][j] ← match
12:            else
13:                lcs[i][j] ← max(exclude_top, exclude_bot, exclude_both)
14:    return lcs[str_1.len][str_2.len]
```