Weekly Discussion

Gang Chen

Department of Computer Science
Nanjing University

October 14, 2016

Problem Description

115. Distinct Subsequences

Given a string S and a string T, count the number of distinct subsequences of T in S.

Example:

S = "rabbbit", T = "rabbit"

Return 3.

Subsequence:

A subsequence of a string is a new string which is formed from the original string by deleting some (can be none) of the characters without disturbing the relative positions of the remaining characters. (ie, "ACE" is a subsequence of "ABCDE" while "AEC" is not).

Solution

How to understand the problem?

1. In fact, the situation is consist of some substates.
2. The essence of substates is always same.
3. Later substate can develop from one(some) former substate(s).

Solution

How to define the substates in the situation?

Situation:

Count the number of distinct subsequences of T in S.

Setting:

ans[i][j] means that S[0..j-1] contains T[0..i-1] as many times as distinct sub-sequences.

Result:

Obviously, the result should be ans [T.length()][S.length()].

Solution

How can we get ans [i][j] when we kown the former states ?

In general,we can divide the problem into two subproblems.

1. T[i] != S[j]:

   ans [i][j] = ans [i][j – 1]

2. T[i] == S[j]:

   ans [i][j] = ans [i][j - 1] + ans [i - 1][j - 1]

Solution

What else?

1. S.length() < T.length()?
2. S.length() >= T.length()
    2.1 T.length() == 0?
    2.2 T.length() > 0
        2.2.1 T[i] != S[i]?
        2.2.2 T[i] == S[i]?

Time Complexity: O(S.length()*T.length())

Solution

```
int numDistinct(string s, string t) {
    int slen = s.length(), tlen = t.length();
    vector<vector<int> > ans (tlen + 1, vector<int>(slen + 1));
    for (int j = 0; j <= slen; ++j) ans[0][j] = 1;
    for (int i = 1; i <= tlen; ++i) {
      for (int j = i; j <= slen; ++j) {
        ans [i][j] = ans [i][j - 1];
        if (t[i - 1] == s[j - 1]) ans[i][j] += ans [i - 1][j - 1];
      }
    }
    return ans [tlen][slen];
  }
```

Solution

Can we do better ?

```
int numDistinct(string s, string t) {
    int slen = s.size(), tlen = t.size();
    vector<vector<int> > ans (slen + 1, vector<int>(tlen + 1));
    for (int j = 0; j <= slen; ++j) ans[j][0] = 1;
    for (int j = 1; j <= slen; ++j){
      for (int i = 1; i <= tlen; ++i) {
        ans [j][i] = ans[j - 1][i];
        if (t[i - 1] == s[j - 1]) ans[j][i] += ans [j - 1][i - 1];}}
    return ans [slen][tlen];
  }
```

Solution

```
int numDistinct(string s, string t) {
    int slen = s.length(), tlen = t.length();
    if (slen < tlen) return 0;
    vector<int> ans (tlen+1);
    ans[0] = 1;
    for (int j = 1; j <= slen; ++j)
      for (int i = tlen; i >= 1; --i)
        if (t[i - 1] == s[j - 1])  ans[i] += ans[i - 1];
    return ans[tlen];
  }
```

What is more

Can we solve the problem by other solution?

1. Backtracking
2. ……

Problem Description

188. Best Time to Buy and Sell Stock IV

Say you have an array for which the ith element is the price of a given stock on day i. Design an algorithm to find the maximum profit. You may complete at most k transactions.

Note:

You may not engage in multiple transactions at the same time (ie, you must sell the stock before you buy again).

Solution

How to define the substates in the situation?

Situation:

Count the maximum profit when using at most k transactions up to day prices.size() (including day prices.size()).

Setting:

ans[i][j] means the maximum profit you get when using at most t transactions up to day i (including day i).

Result:

Obviously, the result should be ans[k][prices.size()].

Solution

How can we get ans[i][j] when we kown the former states ?

In general,we can divide the problem into two subproblems.

1. Transactions(at most i) including day j:

   ans [i][j] = ans[i][j - 1]

2. Transactions(at most i) excluding day j:

   ans [i][j] = max(ans[i - 1][jj] – prices[jj]) + prices[j] (jj in [0,j – 1])

Solution

What else?

1. k >= prices.size()/2 ?
2. k < prices.size()/2
   - 2.1 k == 0?
   - 2.2 k > 0

Time Complexity: O(k*prices.size())

Solution

k >= prices.size()/2

```
int quickR(vector<int> prices){
        int maxProfit = 0;
        for (int i = 1; i < n; ++i)
           if(prices[i] > prices[i - 1])
              maxProfit += prices[i] - prices[i - 1];
        return maxProfit;
}
```

Solution

k < prices.size()/2

```
vector<vector<int> > ans(k + 1, vector<int>(n));
      for (int i = 1; i <= k; ++i) {
         int tempMaxProfit = -prices[0];
         for (int j = 1; j < n ; ++j) {
            ans[i][j] = max(ans[i][j - 1], tempMaxProfit + prices[j]);
            tempMaxProfit = max(tempMaxProfit, ans[i - 1][j] -prices[j]);
         }
      }
 return ans[k][n - 1];
```

Solution

Can we do better ?

What is more

Can we solve the problem by other solutions?

Problem Description

321. Create Maximum Number

Given two arrays of length m and n with digits 0-9 representing two numbers.

Create the maximum number of length k <= m + n from digits of the two. The relative order of the digits from the same array must be preserved. Return an array of the k digits. You should try to optimize your time and space complexity.

Example:

nums1 = [3, 4, 6, 5]

nums2 = [9, 1, 2, 5, 8, 3]

k = 5

return [9, 8, 6, 5, 3]

Solution

How to understand the problem?

1. Create the maximum number of one array without disrupting the order.
2. Create the maximum number of two arrays formed from 1 using all of their digits.

Solution

How to create the maximum number of one array without disrupting the order?

For example, num = [5, 8, 2, 1, 0, 7, 6, 5] and temp = [] storing the answer.

Question: how is temp like when k = 3?

1. k = 8
2. k = 7
3. …
4. k = 3

Key: DP

Solution

How to create the maximum number of one array without disrupting the order?

Assuming the current point of num is i(i >= 0), the next point of temp is j(j>0)

1. num.length – (i + 1) == temp.length – j
2. num.length – (i + 1) > temp.length – j

Key: Greedy

Solution

```
public int[] maxNum(int[] nums, int k){
    int[] temp = new int[k];
    int n = nums.length;
    for (int i = 0, j = 0; i < n; ++i) {
      while (n - i > k - j && j > 0 && temp[j - 1] < nums[i]) j--;
      if (j < k) temp[j++] = nums[i];
    }
    return temp;
  }
```

Solution

How to create the maximum number of two arrays using all of their digits?

For example

num1 = [5, 8, 2, 4, 4, 7, 6, 5] and k = 9

Num2 = [5, 8, 3] or Num2 = [5, 8]?

num1 = [5, 5, 2, 4, 4, 7, 6, 5] and k = 9

Num2 = [5, 5, 3] or Num2 = [5, 5]?

num1 = [5, 3, 2, 4, 4, 7, 6, 5] and k = 9

Num2 = [5, 3, 3] or Num2 = [5, 3]?

Solution

How to create the maximum number of two arrays using all of their digits?

num1[0,1,...,i,...,num1.length-1]

num2[0,1,...,j,...,num2.length-1]

When nums1[i] == nums2[j], which one should we select?

lexicographical order

Solution

Actually, we should put the larger number as much as possible in the result.

Generally, consider the following situation:

num1[i-1] != num2[j-1]

num1[i] == num2[j]

num1[i + 1] == num2[j + 1]

...

num1[i+k-1] == num2[j+k-1] (k>1)

num1[i+k] != num2[j+k]

Solution

We only need to consider the values of num1[i] ,num1[i + 1] ,num1[i+k] and num2[j+k].

1. num1[i] < num1[i + 1]

1.1 num1[i+k] >= num2[j+k]: select num1[i]

1.2 num1[i+k] < num2[j+k]: select num2[j]

2. num1[i] >= num1[i + 1]: select either num1[i] or num2[j] is ok

Solution

In a word

1. num1[i+k] >= num2[j+k]: select num1[i]
2. num1[i+k] < num2[j+k]: select num2[j]

Solution

```
public boolean greater(int[] nums1, int n1, int[] nums2, int n2){
    //lexicographical order
    int len1 = nums1.length, len2 = nums2.length;
    while (n1 < len1 && n2 < len2 && nums1[n1] == nums2[n2]){
      n1++;n2++;
    }
    return n2 == len2 || (n1 < len1 && nums1[n1] > nums2[n2]);
}
```

Solution

```
public int[] merge(int[] nums1, int[] nums2){
    int len1 = nums1.length, len2 = nums2.length, k = len1 + len2;
    int[] temp = new int[k];
    for (int i = 0, j = 0, l = 0; l < k; ++l) {
      if (greater(nums1, i, nums2, j)) temp[l] = nums1[i++];
      else temp[l] = nums2[j++];
    }
    return temp;
}
```

Solution

```
public int[] maxNumber(int[] nums1, int[] nums2, int k) {
    int len1 = nums1.length,len2 = nums2.length;
    int[] ans = new int[k];
    for (int i = Math.max(0, k-len2); i <= Math.min(k, len1); ++i)
        if (k - i <= len2) {
            int[] temp = merge(maxNum(nums1,i),maxNum(nums2,k - i));
            if (greater(temp, 0, ans, 0)) ans = temp;
        }
    }
    return ans;
  }
```

The End

Thanks for your attention