Algorithm Design and Analysis

Homework 3

Haochen Huang
5140309485

November 15, 2016

1

1.1 Question

You are given coins of different denominations and a total amount of money amount.

Write a function to compute the fewest number of coins that you need to make up that amount. If that amount of money cannot be made up by any combination of the coins, return -1.

Example 1:

coins = [1, 2, 5], amount = 11

return 3 (11 = 5 + 5 + 1)

Example 2:

coins = [2], amount = 3

return -1.

Input:

int coins[];

int n: length of coins[];

int amount;

Output:

int num;

1.2 Answer

This is a simple problem with dynamic programming. To calculate the number of coins of each amount, we can check the minimum number of coins of amount minus each coin and plus one. **Time complexity O(n*amount)**

1.3 Code

C++ :

```
int coinCount(int *coins,int n,int amount){
        int *money = new int [amount+1];
        money[0] = 0;
        for (int i=1; i<=amount; ++i){
                money[i] = amount+1;
                for (int j=0; j<n; ++j){
                        if (coins[j]<=i){
                                money[i] = min(money[i],money[i-coins[j]]+1);
                        }
                }
        }

        if (money[amount] > amount) return -1;
        else return money[amount];
}
```

2

2.1 Question

Given a string s, partition s such that every substring of the partition is a palindrome.

Return the minimum cuts needed for a palindrome partitioning of s.

For example, given s = "aab",

Return 1 since the palindrome partitioning ["aa","b"] could be produced using 1 cut.

Input:

string s;

Output:

int cuts;

2.2 Answer

Use dynamic programming as well. The idea is to calculate min cut of substring s[i-1,j+1] based on substring s[i, j]. We scan from right end to left of the string to search for any palindrome substring inside it so that we can get a candidate partition cut. We scan every possible right end palindrome substring to get the minimum cut. Eventually, the result to return is the min cut for s[0,len-1].

Time complexity O(n^2)

2.3 Code

C++ :

```
int strParti(string s){
        int ls = s.length();
        if (ls <= 1){
                return 0;
        }
        int p[100][100];  //true or false
        memset(p,0,sizeof(p));
        int *numP = new int [100];
        for (int i=0; i<ls; ++i){
                numP[i] = ls - i -1;        //max all partition
        }
        for (int i=ls-1; i>=0; --i){
                for (int j=i; j<ls; ++j){
                        if ((p[i+1][j-1] && s[i] == s[j])
                        or (i==j-1 && s[i] == s[j])
                        or (i==j)){
                                p[i][j] = 1;    //is Palindrome
                                numP[i] = min(numP[i], numP[j+1]+1);
                        }
                }
        }
        return numP[0];  //the whole string
```

```
23 }
```

3

3.1 Question

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

Create the maximum number of length $k \leq m + n$ 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 1:

nums1 = [3, 4, 6, 5]

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

k = 5

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

Example 2:

nums1 = [6, 7]

nums2 = [6, 0, 4]

k = 5

return [6, 7, 6, 0, 4]

Example 3:

nums1 = [3, 9]

nums2 = [8, 9]

k = 3

return [9, 8, 9]

Input:

int nums1[], int m;

int nums2[], int n;

int k;

Output:

int nums[];

3.2 Answer

We can use the greedy method to solve this.

Divide the whole k numbers into two parts. num1 can take charge of $i(0 \leq i \leq k)$ numbers and num2 can the charge of the rest k-i numbers

So now the question is converted to:

1)Get maximum number of k digits from an array.

When $nums[i] < nums[i+1]$, we can remove nums[i] in order to get the maximum number, as long as the rest of the array can guarantee to provide enough integers to form an integer of k digits.

2)Since we have two maximum array, with one size of i and another size of k-i, the problem is how can we combine the two into one maximum integer.

And we can do it similar to merge sort.

The difference is that if two digit are equal, we should choose according to the digits after them. So we should write a compare function that compare two array till the end, not just the current two digits. Then just merge two array together.

Analyze the complexities:

Time complexity O($n*m*k$) Space complexity O($n+m+k$)

3.3 Code

C++ :

```
vector<int> arraymax(vector<int> nums, int k) {
    while (nums.size() > k) {
        int  n = nums.size();
        for (i=0; i<n-1; ++i) {
            if (nums[i] < nums[i + 1]) {
                nums.erase(nums.begin() + i); //remove this digit with index i
                break;
            }
        }
        if (i == n - 1) nums.erase(nums.begin() + i);
    }
    return nums;
}

bool compare(vector<int> &nums1, int i, vector<int> &nums2, int j) {
    while (i < nums1.size() && j < nums2.size() && nums1[i] == nums2[j]) {
        ++i;
        ++j;
    }
    if (i < nums1.size() && nums1[i] > nums2[j])  //num2<num1
        or (j == nums2.size())                    //num2 empty
        {
                return true;
        }
    else return false;
}

vector<int> merge(vector<int> &nums1, vector<int> &nums2, int k) {
    vector<int> merg(k, 0);
    int i = 0, j = 0, t = 0;
    for (t = 0; t < k; ++t) {
        if (compare(nums1, i, nums2, j) == 1){  //nums1 > nums22
                merg[t] = nums1[i];++i;    //merge into the merg
```

```
                }
                else merg[t] = nums2[j];++j;
    }
    return merg;
}

vector<int> maxKtwoArray(vector<int>& nums1, vector<int>& nums2, int m, int n,
    int k) {
    vector<int> merg(k, 0);

    for (int i = max(0, k - n); i <= min(k, m); ++i) {
        vector<int> v1 = arraymax(nums1, i);
        vector<int> v2 = arraymax(nums2, k - i);
        vector<int> tmp = merge(v1, v2, k);
        if (compare(tmp, 0, merg, 0)) merg = tmp;
    }

    return merg;
}
```