Algorithm Design and Analysis

Homework 5

Haochen Huang
5140309485

December 18, 2016

1

1.1 Question

Given a positive integer n and you can do operations as follow:

1. If n is even, replace n with n/2.

2. If n is odd, you can replace n with either n + 1 or n - 1.

What is the minimum number of replacements needed for n to become 1?

Example 1:

Input: 8

Output: 3

Explanation: 8 => 4 => 2 => 1

Example 2:

Input:7

Output:4

Explanation: 7 => 8 => 4 => 2 => 1 or 7 => 6 => 3 => 2 => 1

Input: int n;

Output: int minNum;

1.2 Answer

If the number n is even, we divide it by 2.

If the number n is odd, if n = 4k+1,

1) $4k + 1 \to 4k \to 2k \to k(\to k + 1)$

2) $4k + 1 \to 4k + 2 \to 2K + 1 \to 2k \to k$

3) $4k + 1 \to 4k + 2 \to 2K + 1 \to 2k + 2 \to k + 1$

So, 2) is one step more than 1). 3) is less than or equal to 1). So we need to make n to 4K.

So do when n = 4K+3. Let n be 4K+4.

Here is also one exception: when n = 3, the best solution is $3 \to 2 \to 1$, three steps.

Another exception is that when n = $INT\_MAX = 2^{32} - 1$, we can not let n be n+1 when n is an int, so we just return the steps = 32.

Time complexity O(log(n))

1.3 Code

C++ :

```cpp
#include <iostream>
using namespace std;

int minreplace(int n){
        int minNum = 0;
    if (n == 1)
        return minNum;
    if (n == 3)
        return minNum+2;
        if (n == INT_MAX)
            return 32;
        while (n!=1){
                if(n%2 == 0){
                        n = n/2;
                }
                else {
                        if (n%4 == 1)
                                n = n-1;
                        else
                                n = n+1;
                }
                ++minNum;
            if (n == 3)
                return minNum+2;
        }
        return minNum;
}

int main(){
        int n = 7;
        cout<<minreplace(n);
}
```

2

2.1 Question

Implement a basic calculator to evaluate a simple expression string.

The expression string may contain open (and closing parentheses), the plus + or minus sign -, non-negative integers and empty spaces.

You may assume that the given expression is always valid.

Some examples:

"1 + 1" = 2

" 2-1 + 2 " = 3

"(1+(4+5+2)-3)+(6+8)" = 23

Note: Do not use the eval built-in library function.

Input: string str;

Output: int result;

2.2 Answer

This is a classical and easy problem about expression. Here I use a char stack and a number stack to store the sign and the number.

First whenever there is a space, skip it.

If there is a number, push into the number stack. If there is a sign, push into the char stack. Here if it is a minus sign, just change the following number n to minus n, so that all the calculations are add and there order can be changed arbitrarily.

When there is a ')', we need to do the calculate until it meets the corresponding '('. The calculation includes pop two number and one sign and calculate it and push the result to the number stack.

At last when the sign stack is empty, return the top of the number stack.

Time complexity O(n)

2.3 Code

C++ :

```cpp
#include <iostream>
#include <cstring>
#include <cmath>

using namespace std;

template <class elemType>
class stack{
        public :
                virtual bool isEmpty() const =0;
                virtual void push(const elemType &x)=0;
                virtual elemType pop()=0;
                virtual elemType top() const =0;
                virtual ~stack(){}
};
template <class elemType>
class seqStack: public stack <elemType>{
        private:
                elemType *elem;
                int top_p;
                int maxSize;

                void doubleSpace();
        public:
                seqStack(int init=30){
                        elem = new elemType[init];
                        maxSize = init;
                        top_p=-1;
                }
                ~seqStack(){
                        delete []elem;
                };
                bool isEmpty() const {return top_p==-1;}
                void push(const elemType &x){
                        if (top_p>=maxSize-2) doubleSpace();
                        top_p++;
                        elem[top_p]=x;
                }
                elemType pop(){
                        if (top_p!=-1){
                                top_p--;
                                return elem[top_p+1];
                        }
                }
                elemType top() const{
                        return elem[top_p];
                }
};
template <class elemType>
void seqStack<elemType>::doubleSpace(){
        elemType *tmp= elem;
        elem = new elemType[2*maxSize];
        for (int i=0;i<maxSize;++i){
                elem[i] = tmp[i];
        }
        maxSize*=2;
        delete [] tmp;
}

seqStack<char> seqF;
seqStack<int> seqI;

int cal(string str){
        for (int i=0; i<str.length(); ++i){
                if (str[i] == ' '){++i;}
                if (str[i] == '('){
                        seqF.push(str[i]);
                }
                if (str[i] == '+' ){
                        seqF.push(str[i]);
                }
                if (str[i] =='-'){
                        seqF.push('+');++i;
                        while (str[i] == ' ') ++i;
                        int tmp = str[i] - '0';
                        seqI.push(-tmp);
                }
                else if (str[i] >= '0' and str[i] <= '9'){
                        int t = str[i] - '0';
                        seqI.push(t);
                }
                if (str[i] == ')'){
                        while (1){
                                char fuhao = seqF.pop();
                                if (fuhao == '('){
                                        break;
                                }
                                int a = seqI.pop();
                                int b = seqI.pop();
                                seqI.push(a+b);
//                              cout<<seqI.top()<<endl;
                        }
                }
        }
        while (!seqF.isEmpty()){
                char fuhao = seqF.pop();
                int a = seqI.pop();
                int b = seqI.pop();
//              cout<<a<<" + "<<b<<" = "<<a+b;
                seqI.push(a+b);
        }

        return seqI.pop();
}
int main(){
         string str = "(1 + (4 + 5 + 2) - 3)+(6+8)";
         cout<<cal(str);
}
```

3

3.1 Question

In the computer world, use restricted resource you have to generate maximum benefit is what we always want to pursue.

For now, suppose you are a dominator of m 0s and n 1s respectively. On the other hand, there is an array with strings consisting of only 0s and 1s.

Now your task is to find the maximum number of strings that you can form with given m 0s and n 1s. Each 0 and 1 can be used at most once.

Note:

1. The given numbers of 0s and 1s will both not exceed 100.

2. The size of given string array won't exceed 600.

Example 1:

Input: Array = "10", "0001", "111001", "1", "0", m = 5, n = 3.

Output: 4.

Explanation: This are totally 4 strings can be formed by the using of 5 0s and 3 1s, which are 10,0001,1,0.

Example 2:

Input: Array = "10", "0", "1", m = 1, n = 1.

Output: 2.

Explanation: You could form "10", but then you'd have nothing left. Better form "0" and "1".

Input: string array[]; int N; int m; int n; // N: number of strings in array.

Output: int findMaxForm;

3.2 Answer

Use dynamic programming to solve this problem.

I use dp[i][j] to store the max number of strings that can be formed with i 0 and j 1.

For all the strings in the string array, we try to use them from left to right. And calculate the max number of all dps with this string considered.

Clearly, here $dp[i][j] = max(dp[i][j], dp[i - t0][j - t1] + 1)$;

For the following string in the string array, we again calculate the max number with one more string considered.

Time complexity O(m*n*len(string))

3.3 Code

C++ :

```cpp
#include <iostream>
#include <math.h>
using namespace std;
void num(string str, int &zero, int &one){
        for (int i=0;i<str.length();++i){
                if (str[i] == '0') ++zero;
                else ++one;
        }
}
int macForm(string *array, int N, int m, int n){
        int dp[m+1][n+1] = {0};
        for (int i=0 ; i<=m; ++i){
                for (int j=0; j<=n; ++j)
                        dp[i][j] = 0;
        }

        for (int k = 0; k<N; ++k){
                int t0 = 0, t1 = 0;
                num(array[k],t0,t1);
//              cout<<t0<<" "<<t1<<endl;
                for (int i=m; i>=t0; --i){
                        for (int j=n; j>=t1; --j){
                                dp[i][j] = max(dp[i][j],dp[i-t0][j-t1]+1);
//                              cout<<i<<" "<<j<<" "<<dp[i][j]<<endl;
                        }
                }
        }

        return dp[m][n];
}

int main(){
        string array[] = {"10", "0001", "111001", "1", "0"};
        int N = 5;
        int m = 5, n = 3;
        cout<<macForm(array, N , m, n);
}
```