Rohan Behera

CS014

10/14/20

Assignment #1

Exercise 1:

A. Using only core C++ (no special libraries, except STL vector or string if you want), write a C++ program that allows a user to input a string and

   (a) Checks if the expression is a valid polynomial. Parentheses or negation are not allowed. Spaces should be ignored. E.g., the following are valid

      i. *n^2+2*n+5*
      ii. *2*n + 4.54* n^5 +4 +5*n*

   and the following are invalid

      iii. *n^3n*
      iv. *n^4.2*
      v. *5n*
      vi. *n^3 -3*n*

   (b) If the polynomial is valid, outputs its big-O notation. E.g., for (ii) above it is *O(n^5)*.

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

string polyChecker(string s) {
   bool isInvalid = false;

   //checking if string is empty
   if (s.empty()) {
      cout << "Enter valid string" << endl;
      isInvalid = true;
  }

   //checking invalid characters (,), and -
    if (s.find('(') != string::npos || s.find(')') != string::npos || s.find('-') != string::npos) {
       cout << "Parentheses or negation is not allowed" << endl;
       isInvalid = true;
   }

   if (s.find('^') != string::npos) {
       //cout << "No exponent sign found" << endl;
       isInvalid = true;
   }

   int currentExp = 1;
  for (int i = 0; i < s.length(); i++) {
      //checking for exponent
      if (s.at(i) == '^') {
          if (s.at(i+1) > currentExp) {
              currentExp = s.at(i+1); //setting entered power as exponent
          }
           if (!isdigit(s.at(i+1))) {
                   isInvalid = true;
           }
          //checking if the exponent has any decimals
          if (i+2 < s.length()) {
              if (s.at(i+2) != '.') {
                  isInvalid = false;
              }
              else {
                  cout << "Error: A decimal was found in the exponent. Invalid polynomial" << endl;
                  isInvalid = true;
              }
          }
      }
  }

  //Output big-O notation of the equation
  if (isInvalid == false) {
      cout << "Polynomial " << s << " is valid" << endl;
      cout << "Big-O notation for: " << s << " is O(n^" << currentExp << ")" << endl;
  }
   if (isInvalid == true) {
       cout << "Invalid polynomial" << endl;
   }

}

int main(int argc, char** argv)
{
  string polynomialEq;
  cout << "Enter a valid polynomial: " << endl;
  getline(cin, polynomialEq);

  cout << polyChecker(polynomialEq) << endl;
```

```
  return 0;
}
```

B. If the length of the input expression is *m* chars, what is the big-O complexity of your program with respect to *m*?

There are 5 if statements outside of the main for loop that each have a time complexity of O(1). There is a for loop with 3 nested if statements with a time complexity of O(n). Therefore the big-O complexity of my program with respect to m is O(n).

C. What if we require that there is only one term for each degree? That is, (ii) above is invalid because it has two terms for degree *1* (*n^1*).
Modify your program accordingly.
What is the asymptotic complexity of the new program?
Throughout the exercise, make any assumptions necessary.

The asymptotic complexity of the new program is still O(n) because the only change I made to make my program accept only one term per degree is add an if statement.

```
#include <iostream>
#include <vector>
#include <string>
using namespace std;

string polyChecker(string s) {
   bool isInvalid = false;

   //checking if string is empty
   if (s.empty()) {
      cout << "Enter valid string" << endl;
      isInvalid = true;
   }

   //checking invalid characters (,), and -
    if (s.find('(') != string::npos || s.find(')') != string::npos || s.find('-') != string::npos) {
      cout << "Parentheses or negation is not allowed" << endl;
      isInvalid = true;
   }

   if (s.find('^') != string::npos) {
      //cout << "No exponent sign found" << endl;
      isInvalid = true;
   }

   int currentExp = 1;
   for (int i = 0; i < s.length(); i++) {
      //checking for exponent
      if (s.at(i) == '^') {
         if (s.at(i+1) > currentExp) {
            currentExp = s.at(i+1); //setting entered power as exponent
         }
          if (!isdigit(s.at(i+1))) {
                isInvalid = true;
          }
         //checking if the exponent has any decimals
         if (i+2 < s.length()) {
            if (s.at(i+2) != '.') {
               isInvalid = false;
            }
            else {
               cout << "Error: A decimal was found in the exponent. Invalid polynomial" << endl;
               isInvalid = true;
            }
         }
         //Condition changes to accept only one term per degree
         if (!isdigit(s.at(s.length()-1))) {
           // cout << "Multiple terms per degree is not allowed" << endl;
            isInvalid = true;
         }
      }
   }

   //Output big-O notation of the equation
   if (isInvalid == false) {
      cout << "Polynomial " << s << " is valid" << endl;
      cout << "Big-O notation for: " << s << " is O(n^" << currentExp << ")" << endl;
   }
    if (isInvalid == true) {
       cout << "Invalid polynomial" << endl;
    }

}

int main(int argc, char** argv)
{
   string polynomialEq;
   cout << "Enter a valid polynomial: " << endl;
   getline(cin, polynomialEq);

   cout << polyChecker(polynomialEq) << endl;
```

```
  return 0;
}
```

Exercise 2: Given an array A of n integers and an integer s, find a subset of the integers in *A* such that their product is *s*.

A. Write C++ function

```
#include <cstdlib>

#include <iostream>

using namespace std

/*

 *

 */

void productSubset(int array[], int length, int s) {

    cout << "Subset integers with a product of " << s << endl;

    for (int i = 0; i < length; i++) {

        for (int j = i+1; j < length; j++) {

            if (array[i]*array[j] == s) {

                cout << "Subset: " << array[i] << ", " << array[j] << endl;

            }

        }

    }

}

int main(int argc, char** argv) {

    int arr[] = {7, 10, 13, 16, 19, 22, 25, 28, 31};

    int numcountArray = 9;
    int productNum = 112;

    productSubset(arr, numcountArray, productNum);

    return 0;

}
```

B. Compute asymptotic complexity

There is a for loop inside of a for loop and there is an if statement to check if two predefined integers equal s. This gives O(n^2)+O(n) and n^2 is the highest power, the asymptotic complexity is O(n^2).