CPSC 335 Project 2: exhaustive search

Spring 2016

Prof. Doina Bein, CSU Fullerton

dbein@fullerton.edu

Introduction

In this project you will design, implement, and analyze two algorithms for the same problem. For this problem, you will design two separate algorithms, describe the algorithms using clear pseudocode, analyze them mathematically, implement your algorithms in C/C++/Java, measure their performance in running time, compare your experimental results with the efficiency class of your algorithms, and draw conclusions. The first algorithm has a tractable (polynomial) running time, while the second algorithm has an intractable (exponential or factorial) running time.

At the end of this document I have provided you with sample high resolution timing code in C++11, called "Template for a C/C++ program with high resolution timing". If you choose to use Java, you are responsible for figuring out how to do high resolution timing in that language.

The hypotheses

This experiment will test the following hypotheses:

1. *Exhaustive search algorithms are feasible to implement, and produce correct outputs.*
2. *Algorithms with exponential or factorial running times are extremely slow, probably too slow to be of practical use.*

The problem

The longest non-decreasing subsequence problem is to find a subsequence of a given sequence in which the subsequence's elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible. This subsequence is not necessarily contiguous, or unique.
The longest non-decreasing subsequence problem can be formulated as follows:
input: n, a positive integer, and the array A of n comparable elements
output: array R that contains the longest non-decreasing subsequence
The longest non-decreasing subsequence problem is solvable in $O(n \log n)$ time, where n denotes the length of the input sequence[1].

[1] Schensted, C. (1961), "Longest increasing and decreasing subsequences", Canadian Journal of Mathematics 13: 179–191, doi:10.4153/CJM-1961-015-3

A simple algorithm (algorithm End_to_Beginning)

There is a straightforward algorithm that solves the problem and has $O(n^2)$ running time, where denotes n the length of the input sequence. The algorithm uses an additional array H of length n, of non-negative integers. The value H[i] will indicate how many elements, greater or equal to than A[i], are further in the sequence A and have some special property.

Initially, the array H is set to 0 (all the elements are 0). The algorithm proceeds by attempting to increase the values of H starting with the previous to last element and going down to the first element. Then a longest subsequence can be identified by selecting elements of A in decreasing order of the H values, starting with the element in A that has the largest H value.

Algorithm End_to_Beginning
Step 1. Set all the values in the array H to 0.
Step 2. Starting with H[n-1] and going down to H[0] try to increase the value of H[i] as follows:
Step 3. Starting with index i+1 and going up to n-1 (the last index in the array A) repeat Steps 4 and 5:
Step 4. See if any element is bigger than A[i] and has its H value also bigger or equal to H[i] .
Step 5. If yes, then A[i] can be followed by that element in an non-decreasing subsequence, thus set H[i] to be 1 plus the H value of that element.
Step 6. Calculate the largest (maximum) value in array H. By adding 1 to that value we have the length of a longest non-decreasing subsequence.
Step 7. Identify a longest subsequence by identifying elements in array A that have decreasing H values, starting with the largest (maximum) value in array H.
Please see the Class Notes posted on TITANium for two executions of this algorithm on two separate instances.

An exhaustive algorithm Powerset

There is an exhaustive algorithm that solves the problem and has O(n 2^n) running time, where n denotes the length of the input sequence. The algorithm generates all possible subsequences of the array A and then tests each subsequence on whether it is in non-decreasing order. The longest such subsequence is a solution to the problem.
We note that a subsequence can be uniquely identified by the set of indices in the array A that are part of the subsequence. For example, given the array
A = [1, 0, 2, 1, 5, 3, 13, 8, 34, 21, 89, 55, 233, 143]
the subsequence R = [1, 0, 3] is uniquely identified by the set of indices {0,1,5} of the elements in the array A.
To generate all possible subsequences, one may consider generating the power set of {0,1,...n-1} and consider each subset of the power set as the set of indices in A of the subsequence.
There are several ways to generate the power set. One way is to implement an iterative algorithm that uses a stack to grow and shrink the set as needed. The benefit of this approach is that it prints the subsets in lexicographic order. Below find an implementation[2] in C++ that generates the power set of the set {1, 2, …, n} (where you can specify n in the program):

[2] http://www.programminglogic.com/powerset-algorithm-in-c/

```
void Powerset (int n)
// function to generate the power set of {1, .., n} and retrieve the best set
int *stack, k;
 stack = new int[n+1]; // allocate space for the set
 stack[0]=0; /* 0 is not considered as part of the set */
 k = 0;
 while(1) {
  if (stack[k] < n) {
        stack[k+1] = stack[k] + 1;
        k++;
  }
  else {
        stack[k-1]++;
        k--;
  }
  if (k==0) break;
 }
 delete [ ] stack; // deallocate space for the set
 return;
}
```

What to do

1. Write clear pseudocode for each algorithm.

2. Analyze your pseudocode mathematically and write its efficiency class using Big-Oh notation. (You need to compute the total number of steps of the algorithm.)

3. Type these notes (electronically or on paper) and submit it as a PDF report.

4. Implement your algorithm in C/C++/Java/Python. You may use the templates provided at the end of this file.

5. Compile and execute the program.

6. Create a file with the output of the program for an input value and submit it together with the program. Note, the output can be redirected to a file (for easy printing). For example, the following command line will create an output file in Linux-based operating system called `a1out.txt` by re-directing the output from the screen (display) to the file a1out.txt:

```
K:\cpscs335> a.out > a2out.txt
```

Sample outputs for End_to_Beginning Algorithm:

Example #1:
K:\202> ast2a
CPSC 335-x - Programming Assignment #2
Longest non-decreasing subsequence problem, end-to-beginning algorithm
Enter the number of elements in the sequence
5
Enter the elements in the sequence
0 8 4 12 2
Input sequence
0 8 4 12 2
The longest non-decreasing subsequence has length
3
The longest non-decreasing subsequence is
0 8 12
elapsed time: 0.000106 seconds

Example #2:
K:\202> ast2a
Longest non-decreasing subsequence problem, end-to-beginning algorithm
Enter the number of elements in the sequence
16
Enter the elements in the sequence
0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15
Input sequence
0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15
The longest non-decreasing subsequence has length
6
The longest non-decreasing subsequence is
0 4 6 9 13 15
elapsed time: 0.000119 seconds

Sample outputs for Powerset Algorithm:

Example #1:
K:\202> ast2b
CPSC 335-x - Programming Assignment #2
Longest non-decreasing subsequence problem, powerset algorithm
Enter the number of elements in the sequence
5
Enter the elements in the sequence
0 8 4 12 2
Input sequence
0 8 4 12 2
The longest non-decreasing subsequence has length
3
The longest non-decreasing subsequence is
0 8 12
elapsed time: 0.000107 seconds

Example #2:
K:\202> ast2b
Longest non-decreasing subsequence problem, powerset algorithm
Enter the number of elements in the sequence
16
Enter the elements in the sequence
0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15
Input sequence
0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15
The longest non-decreasing subsequence has length
6
The longest non-decreasing subsequence is
0 4 6 9 13 15
elapsed time: 0.005204 seconds

Template for a C/C++ program with high resolution timing:

```
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <chrono>

using namespace std;

int main( ) {
  // DECLARE VARIABLES
  // PRINT THE HEADER OF THE PROGRAM
  // READ THE INPUT

  // Start the chronograph to time the execution of the algorithm
  // Reading the input is NOT part of the algorithm so the timer starts after reading the input
  auto start = chrono::high_resolution_clock::now();

  // YOU NEED TO COMPLETE THIS PART OF CODE

  // End the chronograph to time the execution of the algorithm
  auto end = chrono::high_resolution_clock::now();

  // DISPLAY THE OUTPUT

  // Print the elapsed time in seconds and fractions of seconds
  // Displaying the output is NOT part of the algorithm so the timer ends before displaying the output
  int microseconds = chrono::duration_cast<chrono::microseconds>(end - start).count();
  double seconds = microseconds / 1E6;
  cout << "elapsed time: " << seconds << " seconds" << endl;

  return EXIT_SUCCESS;
}
```

Template for a C/C++ program doing end-to-beginning algorithm

```
// Assignment 2: Longest non-decreasing subsequence problem, end-to-beginning algorithm
// XX YY ( YOU NEED TO COMPLETE YOUR NAME )
// Given a sequence of elements the program finds a subsequence of it in which the subsequence's
// elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible.

// The program reads the number of elements in the sequence, then the elements and outputs the sorted
// sequence and the running time.
// INPUT: a positive integer n and a list of n elements
// OUTPUT: a longest non-decreasing subsequence of the initial sequence
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <chrono>
using namespace std;
void print_sequence(int, float*);
// function to print a sequence, given the number of elements and
// the actual sequence stored as an array
int main() {
  int n, i, j, max, index;
  float *A, *R;
  int *H;
  // display the header
  cout << endl << "CPSC 335-x - Programming Assignment #2" << endl;
  cout << "Longest non-decreasing subsequence problem, end-to-beginning algorithm" << endl;
  cout << "Enter the number of elements in the sequence" << endl;
  // read the number of elements
  cin >> n;
  // allocate space for the input sequence and array H
  A = new float[n];
  H = new int[n];
  // read the sequence
  cout << "Enter the elements in the sequence" << endl;
  for( i=0; i < n; i++)
          cin >> A[i];
  // print the sequence
  cout << "Input sequence" << endl;
  print_sequence(n,A);
  // Start the chronograph to time the execution of the algorithm
  auto start = chrono::high_resolution_clock::now();
  // loop to populate the array H with 0 values
  for(i=0; i< n; i++)
          H[i] = 0;
  // loop to calculate the values of array H
  for ( i = n-2;  i>= 0; i--) {
          for ( j = i+1; j < n ; j++)
```

```
        // WRITE THE CODE THAT IS AN IF CONDITION THAT DECIDES WHETHER
        // TO CHANGE OR NOT THE VALUE OF H[i]
 }
 // calculate in max the length of the longest subsequence by adding 1
 // to the maximum value in H
 max = H[0];
 for( i=1; i< n; i++)
        if (max < H[i])
        max = H[i];
 max ++;
 // allocate space for the subsequence R
 R = new float[max];
 // add elements to R by whose H's values are in decreasing order, starting
 // with max-1
 // store in index the H values sought
 index = max - 1;
 // store in j the index of the element appended to R
 j = 0;
 for(i=0; i< n; i++)
        if (H[i] == index) {
        // WRITE THE BLOCK OF STATEMENTS TO ADD A[i] TO THE R SEQUENCE BY
        // STORYING IT INTO R[j], DECREMENTING index AND INCREMENTING j
        }
 // End the chronograph to time the loop
 auto end = chrono::high_resolution_clock::now();
 // write the output
 cout << "The longest non-decreasing subsequence has length " << endl;
 cout << max << endl;
 cout << "The longest non-decreasing subsequence is" << endl;
 print_sequence(max, R);
 // print the elapsed time in seconds and fractions of seconds
 int microseconds = chrono::duration_cast<chrono::microseconds>(end - start).count();
 double seconds = microseconds / 1E6;
 cout << "elapsed time: " << seconds << " seconds" << endl;
 // de-allocate the dynamic memory space
 delete [] A;
 delete [] H;
 delete [] R;
 return EXIT_SUCCESS;
}
void print_sequence(int n, float *seq)
// function to print a sequence, given the number of elements and
// the actual sequence stored as an array
// n represents the number of elements in the sequence
// seq represents the actual sequence
// WRITE THE CODE TO PRINT THE ELEMENTS OF A SEQUENCE seq WITH n ELEMENTS
```

Template for a C/C++ program doing power set algorithm

```
// Assignment 2: Longest non-decreasing subsequence problem, power set algorithm
// XX YY ( YOU NEED TO COMPLETE YOUR NAME )
// Given a sequence of elements the program finds a subsequence of it in which the subsequence's
// elements are in sorted order, lowest to highest, and in which the subsequence is as long as possible.

// The program reads the number of elements in the sequence, then the elements and outputs the sorted
// sequence and the running time.
// INPUT: a positive integer n and a list of n elements
// OUTPUT: a longest non-decreasing subsequence of the initial sequence
#include <iostream>
#include <iomanip>
#include <cstdlib>
#include <chrono>
using namespace std;
void print_sequence(int, float*);
// function to print a sequence, given the number of elements the actual sequence stored as an array
void printPowerset(int, int*, int&, float*);
// function to generate the power set of {1, 2, …first argument} and retrieve the best set
void checkSet(int[], int, int*, int&, float*);
// function to check the currently generated set stack of size k against
// the current best set bestSet of size bestSize
int main() {
  int n, bestSize, i;
  float *A, *R;
  int *bestSet;
  // display the header
  cout << endl << "CPSC 335-x - Programming Assignment #2" << endl;
  cout << "Longest non-decreasing subsequence problem, powerset algorithm" << endl;
  cout << "Enter the number of elements in the sequence" << endl;
  // read the number of elements
  cin >> n;
  // allocate space for the input sequence and array R
  A = new float[n];
  R = new float[n];
  // read the sequence
  cout << "Enter the elements in the sequence" << endl;
  for( i=0; i < n; i++)
          cin >> A[i];
  // print the sequence
  cout << "Input sequence" << endl;
  print_sequence(n,A);
  // Start the chronograph to time the execution of the algorithm
  auto start = chrono::high_resolution_clock::now();
  // allocate space for the best set; initial its size is 0
```

```
 bestSet = new int[n+1];
 bestSize = 0;
 // calculate the best sequence
 printPowerset(n, bestSet, bestSize, A);
 // retrieve the indices for generating the subsequence
 for(i=0;i<bestSize;i++)
        // decrease each index by one since the indices of array A are in
        // the range 0..n-1 and not 1..n
        R[i]=A[bestSet[i+1]-1];
 // End the chronograph to time the loop
 auto end = chrono::high_resolution_clock::now();
 // display the output
 cout << "The longest non-decreasing subsequence has length " << endl;
 cout << bestSize << endl;
 cout << "The longest non-decreasing subsequence is" << endl;
 print_sequence(bestSize, R);
 // print the elapsed time in seconds and fractions of seconds
 int microseconds =  chrono::duration_cast<chrono::microseconds>(end - start).count();
 double seconds = microseconds / 1E6;
 cout << "elapsed time: " << seconds << " seconds" << endl;
 // de-allocate the dynamic memory space
 delete []A;
 delete []R;
 return EXIT_SUCCESS;
}

void print_sequence(int n, float *seq)
// function to print a sequence, given the number of elements the actual sequence stored as an array
// n represents the number of elements in the sequence
// seq represents the actual sequence
// WRITE THE CODE TO PRINT THE ELEMENTS OF A SEQUENCE seq WITH n ELEMENTS

void printPowerset (int n, int *bestSet, int &bestSize, float *A)
// function to generate the power set of {1, .., n} and retrieve the best set
// n represents the maximum value in the set
// bestSet represents the set
// bestSize is the size of the bestSet
 int *stack,k;
// allocate space for the set
 stack = new int[n+1];
 stack[0]=0; /* 0 is not considered as part of the set */
 k = 0;
 while(1){
        if (stack[k]<n){
        stack[k+1] = stack[k] + 1;
        k++;
        }
   else{
```

```
            stack[k-1]++;
            k--;
            }
            if (k==0)
            break;
            checkSet(stack, k, bestSet, bestSize, A);
  }
  // deallocate space for the set
  delete [ ] stack;
  return;
}

void checkSet(int *stack, int k, int *bestSet, int &bestSize, float *A)
// function to check the currently generated set stack of size k against the current
// best set bestSet of size bestSize
{
  int i;

  // check that the indices in stack generate a subsequence of non-decreasing order
  if (k < 2) {
            // the set contains a single index so the subsequence is in non-decreasing order
            if (k > bestSize) {
            // we found a better set
            // WRITE CODE TO STORE stack into bestSet and UPDATE bestSize TO k
            return;
            }
  }
  else {
            // the set contains more than a single index so check that the subsequence is in order
            for(i=0;i<k-1;i++) {
              // decrease each index by one since the indices of array A are in
              // the range 0..n-1 and not 1..n
            // WRITE CODE (AN IF STATEMENT) TO CHECK THAT THE ELEMENTS IN ARRAY
            // A AT INDICES stack[i+1]-1 AND stack[i+2]-1 ARE IN NON-DECREASING ORDER
            // IF THE TWO ELEMENTS ARE OUT OF ORDER THEN return
    }
  }
  // we have an non-decreasing so we compare it against the current best set
  if (k > bestSize) {
            // we found a better set
            // WRITE CODE TO STORE stack into bestSet and UPDATE bestSize TO k
            return;
  }
  else
            return;
}
```