Billy Saysavath

CPSC 362 – Section 3

Fall 2015

Assignment #2

# End-to-Beginning Algorithm:

# Greedy-pattern:

def end\_to\_beginning(n, A, H):

todo = <PIECES OF INPUTS>

result = <INITIAL RESULT>

for element in todo:

<PROCESS element and UPDATE result>

return result

“todo” nor “result” are needed. Set array “H”’s values to 0. There are two loops that we will use. The first loop will start at “i = n-2” and loop back to “i = 0” indices. In each iteration, the second loop will start to compare the next index “j = i + 1” until it reaches “j = n-1.” In this second loop, we are going to compare if “A[i] <= A[j] && H[i] <= H[j]”. If so, “H[i] = H[j] + 1.” Next, we are going to find max value in “H” and add 1 to “max.” Then, we initialize “R” with “max” indices. Finally, we loop from “i = 1” to “i = n.” If “H[i] == max – 1,” “R[j] = H[i].” Decrease “max – 1” and increase index for “R.”

* 1. Pseudocode:

def end\_to\_beginning(n, A, H):

H[] \* n = 0;

for i in range (n-2, 0):

for j in range (i, n-1):

if (A[i] <= A[j]) && (H[i] <= H[j]):

H[i] = H[j] + 1;

max = H[0];

for i in range(0, n):

if (max < H[i]):

max = H[i];

max++;

R[max];

index = max – 1;

for i in range (1, n):

if(H[i] == index):

R[j] = H[i];

index--;

j++;

return R;

* 1. Analyze:

Line 8 will take n times to initialize H. Next, the first loop will take n-2 times. The inner loop will iterate from i to n-1. Line 12 and 13 will be 2 + max(1, 0). Line 15 is 1. Line 16-18 is n[1+max(1,0)]. Line 19 is 1. Line 21 will take “m” times. Line 22 is 1. Line 24 is (n-1)[1+max(3,0)]. Line 30 is 1.

Since, m at its worst case is n. We have:

* 1. Output:

CPSC 335-3 - Programming Assignment #2

Longest increasing 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 increasing subsequence has length

3

The longest increasing subsequence is

0 8 12

elapsed time: 0.000061 seconds

Press ENTER to finish...

CPSC 335-3 - Programming Assignment #2

Longest increasing 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 increasing subsequence has length

6

The longest increasing subsequence is

0 4 6 9 13 15

elapsed time: 0.000004 seconds

Press ENTER to finish...

CPSC 335-3 - Programming Assignment #2

Longest increasing subsequence problem, end-to-beginning algorithm

Enter the number of elements in the sequence

20

Enter the elements in the sequence

27 7 9 19 13 6 26 3 15 9 14 8 16 17 30 18 20 12 2 5

Input sequence

27 7 9 19 13 6 26 3 15 9 14 8 16 17 30 18 20 12 2 5

The longest increasing subsequence has length

8

The longest increasing subsequence is

7 9 13 15 16 17 18 20

elapsed time: 0.000003 seconds

Press ENTER to finish...

1. Power Set Algorithm:

# Pseudocode:

// powerSet function template is already given

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;

}

// Below is the pseudocode of a checkSet function

void power\_set(int \*stack, int k, int \*bestSet, int \*bestSize, float \*A) {

if (k <= 2) {

// the set contains a single index so the subsequence is in increasing order

if (k > bestSize) {

// we found a better set

for (int i = 1; i <= k; i++) {

bestSet[i] = stack[i];

}

bestSize = 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

if (A[stack[i + 1] – 1] > A[stack[i + 2] – 1]) return;

}

}

// we have an increasing so we compare it against the current best set

if (k > bestSize) {

// we found a better set

for (int i = 1; i <= k; i++) {

bestSet[i] = stack[i];

}

bestSize = k;

return;

}

else

return;

}

* 1. Analyze:

As we learned in a class, the power set has a running time of (. Then, we need to add the running time of the “checkSet” function. We have the following:

Since, k at its worst case is n, so

* 1. Output:

CPSC 335-3 - Programming Assignment #2

Longest increasing 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 increasing subsequence has length

3

The longest increasing subsequence is

0 8 12

elapsed time: 0.000008 seconds

Press ENTER to finish...

CPSC 335-3 - Programming Assignment #2

Longest increasing subsequence problem, powerset algorithm

Enter the number of elements in the sequence

16

Enter the elements in the sequence

27 7 9 19 13 6 26 3 15 9 14 8 16 17 30 18 20 12 2 5

Input sequence

0 8 4 12 2 10 6 14 1 9 5 13 3 11 7 15

The longest increasing subsequence has length

6

The longest increasing subsequence is

0 4 6 9 13 15

elapsed time: 0.003393 seconds

Press ENTER to finish...

CPSC 335-3 - Programming Assignment #2

Longest increasing subsequence problem, powerset algorithm

Enter the number of elements in the sequence

20

Enter the elements in the sequence

27 7 9 19 13 6 26 3 15 9 14 8 16 17 30 18 20 12 2 5

Input sequence

27 7 9 19 13 6 26 3 15 9 14 8 16 17 30 18 20 12 2 5

The longest increasing subsequence has length

8

The longest increasing subsequence is

7 9 13 15 16 17 18 20

elapsed time: 0.015676 seconds

Press ENTER to finish...