KINGDOM OF SAUDI ARABIA
Ministry of Higher Education
Al-Imam Mohammad University
College of Computer & Information Sciences

المملكة العربية السعودية
وزارة التعليم العالي
جامعة الإمام محمد بن سعود الإسلامية
كلية علوم الحاسب والمعلومات

CS215 Project

Bridge to nowhere

Work by:

Anas Markhouss (438021838)

Amr AL Gamal (438021843)

2019/04/10

Bridge to Nowhere

Introduction

By definition, a bridge to nowhere is a bridge where one or both ends are incomplete or damaged, so the bridge itself does not lead to anywhere. Some of them are unfinished due to lack of funds, many were destroyed in wars or natural disasters such as floods or earthquakes, and some have simply collapsed due to poor construction or long-term use(1).

Problem:

Consider a 2-D map with a horizontal river passing through its center. There are n cities on the southern bank with x-coordinates a(1) … a(n) and n cities on the northern bank with x-coordinates b(1) … b(n). You want to connect as many north-south pairs of cities as possible with bridges such that no two bridges cross. When connecting cities, you can only connect city a(i) on the northern bank to city b(i) on the southern bank. Maximum number of bridges that can be built to connect north-south pairs with the aforementioned constraints(2).

```
8     1     4     3     5     2     6     7
<---- Cities on the other bank of river---->
-----------------------------------------
  <--------------- River--------------->
-----------------------------------------
1     2     3     4     5     6     7     8
<------- Cities on one bank of river------->
```

Dynamic Programming

In the Dynamic Programming,

1. We divide the large problem into multiple subproblems.

2. Solve the subproblem and store the result.

3. Using the subproblem result, we can build the solution for the large problem.

4. While solving the large problem, if the same subproblem occurs again, we can reuse the already stored result rather than recomputing it again. This is also called **memoization.**

Dynamic Programming Approaches

1. Bottom-Up approach

2. Top-Down approach

1. Bottom-Up approach

Start computing result for the subproblem. Using the subproblem result solve another subproblem and finally solve the whole problem.

Example

Let's find the nth member of a Fibonacci series.

Fibonacci(0) = 0

Fibonacci(1) = 1

Fibonacci(2) = 1 (Fibonacci(0) + Fibonacci(1))

Fibonacci(3) = 2 (Fibonacci(1) + Fibonacci(2))

We can solve the problem step by step.

1. Find Oth member

2. Find 1st member

3. Calculate the 2nd member using 0th and 1st member

4. Calculate the 3rd member using 1st and 2nd member

5. By doing this we can easily find the nth member.

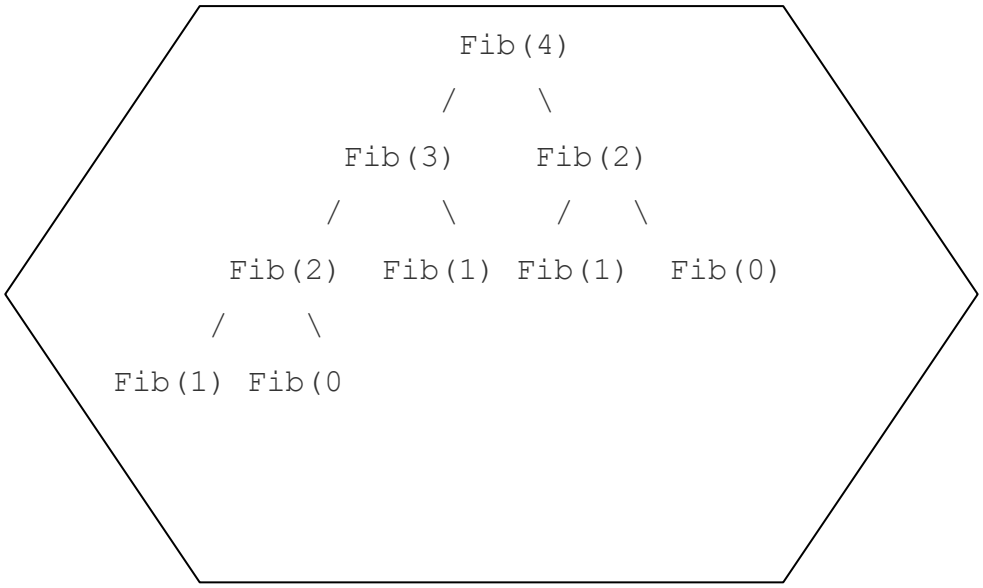
2.Top-Down approach

Top-Down breaks the large problem into multiple subproblems.

if the subproblem solved already just reuse the answer.

Otherwise, Solve the subproblem and store the result.

Top-Down uses memoization to avoid recomputing the same subproblem again. [3].



We used Bottom-Up approach since we solve each subproblem to solve the whole problem thus, Bottom-Up approach is suitable for this kind of problem.

HOW CAN WE SOLVE THIS PROBLEM?

To solve this kind of problem we use what is called (Longest increasing subsequence algorithm) or (LIS) in Dynamic Programing to find the largest set of bridges that we can built. Since every subproblem can be used to solve the whole problem.

The whole problem is every city in the north – south bank if we solve the bridge between the cities, we will have solve the whole problem.

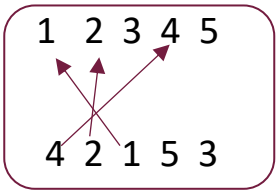
The subproblem is each pair of cities and building the bridge between them is the subproblem. Solving each subproblem will lead us to solve the whole problem.

- LIS & LDS
  There are couple of subsequence DP algorithms to use but, LIS and LDS are the most suitable for this problem because there is some kind of increasing / decreasing ordering in the subsequence. here is more explanation for the two subsequences:

We tried to solve the problem using LDS (Longest Decreasing Subsequence) but we found out that we can built more bridges using LIS than that we can using LDS thus, we used LIS for this problem.

For instance:

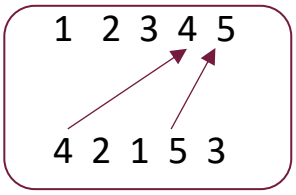
- LDS:



LDS length = 3

(4,2,1)

- LIS:

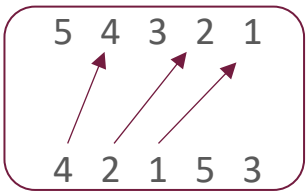


LIS length = 2

(4,5) or (2 ,5) or (2,3) or (1,3)

Note: that doesn't mean that LDS can't work as solution.it can work only if we reverse the order of the north bank.

For the same example as above:



LDS length = 3

(4,2,1)

reverse order

Longest increasing subsequence

One of the problem that can be solved using Dynamic Programing is The Longest Increasing Subsequence (LIS). (LIS) problem is to find the length of the longest subsequence of a given sequence such that all elements of the subsequence are sorted in increasing order.
For example, the length of LIS for {10, 22, 9, 33, 21, 50, 41, 60, 80} is 6 and LIS is {10, 22, 33, 50, 60, 80} [(4)].

arr[ ]	10	22	9	33	21	50	41	60	80
LIS	1	2		3		4		5	6

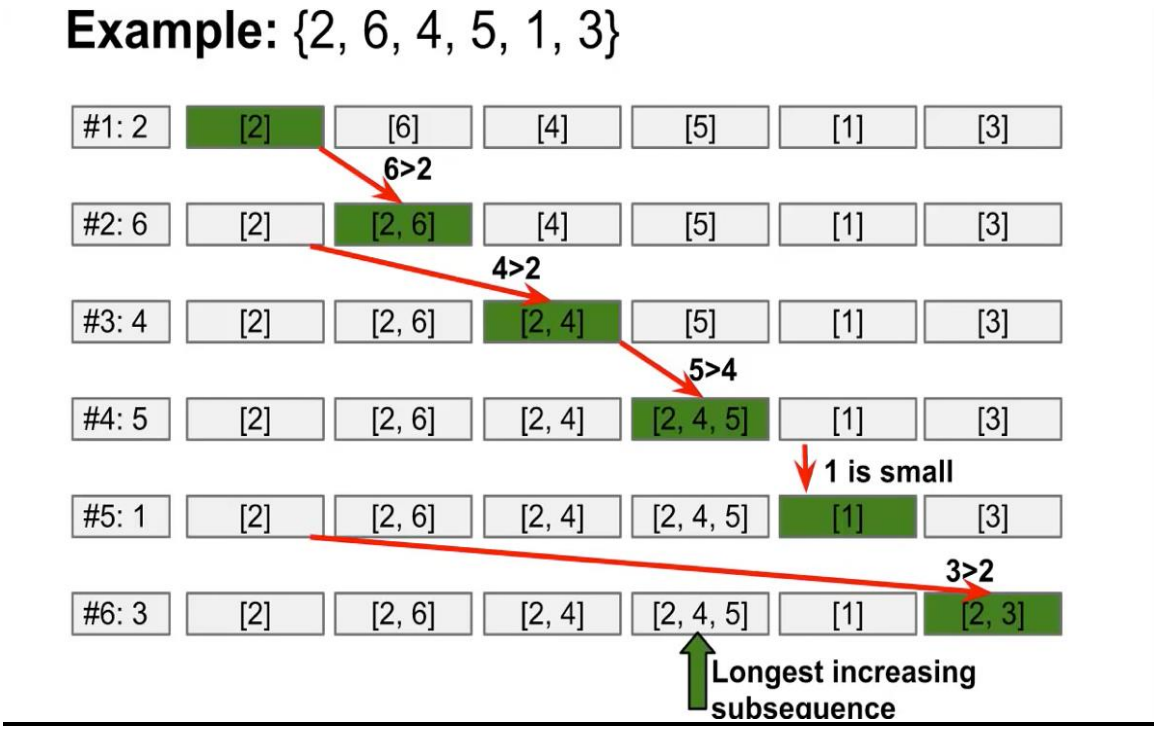
- There are two approaches for (LIS) Problem

  1 – Time complexity of (n^2).using DP

  2 - Time complexity of (n log(n)). Not DP

  The (n log(n)) takes more space and uses more resources. Thus, we used the (n^2) complexity since it applied the DP principles.

  For Example [5]:



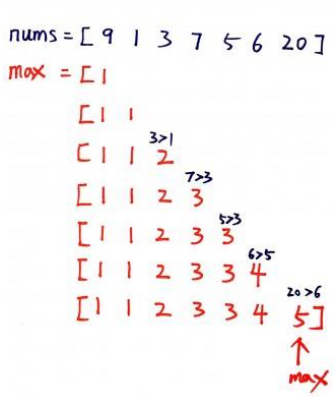
- **The (n log(n)) approaches (none DP):**

- Solution 1 – **Naïve**:

  Let max[i] represent the length of the longest increasing subsequence so far.

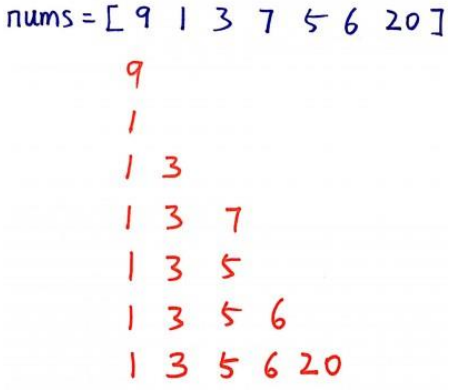
  If any element before i is smaller than nums[i], then max[i] = max(max[i], max[j]+1) [6] .

  Here is an example:



- Solution 2 - **Binary Search**:

We can put the increasing sequence in a list [6].



Algorithm & *Pseudo code*

- **Algorithm**:

1 – start by sorting the north -coordinate (in our case we assume that they are sorted) in acceding order.

2 – using a size array to keep track of LIS of south-coordinate (the DP part) and fill it with 1's initially.

3 – key step: if the current > previous ending and increasing then update the array and update the max length.

4 - repeating the key step (step 3) until we finish all the cities.

5 – after finding the LIS and the maximum Length of the LIS of south-coordinate The LIS length will be the maximum number of bridges we can build.

Pseudo code

Program: Determine the maximum number of bridges.

Input: number of cities.

Output: maximum number of bridges.

Algorithm: Longest increasing subsequence (cities[])

1. Initialize String array paths and integer array sizes to cities number.
2. Initialize max to 1
3. FOR i=0 to cities
4. Sizes[i] = 1
5. Paths[i]= cities[i]+space
6. ENDFOR
7. FOR i=0 to cities
8. FOR j=0 to i
9. IF cities[i]> cities[j] AND sizes[i]< sizes[j]+1
10. Sizes[i]= sizes[j]+1
11. Paths[i]=paths[j] + cities[i]+space
12. IF max< sizes[i]
13. max = sizes[i]

ENDIF

ENDIF

ENDFOR

ENDFOR

14. FOR i=0 to cities
15. IF sizes [i]= max
16. WRITE LIS: paths[i]

Skip a line

Get out of loop

ENDIF

ENDFOR

17. WRITE Max number of bridges = max

Algorithm Analysis

- Lines 1 – 6 : Initialization path & size array(the array which we used to compare the LIS) and max . **O(N)**

- Lines 7 – 13 : Initialize the cites and comparing the cities and the size list if there exist a cities[i]> cities[j] AND sizes[i]< sizes[j]+1, then we increase the sizes array by one and that's the dynamic part (saving the result so it can be used again without recalculating it again) also updating the max ,so we have O(N* N) = **$O(N^2)$**

- Lines 14 – 17 : see if the city at the index of size is the max (the city with the LIS) if yes this print the path (the pairs of cites) that the bridges is going to be build from/to . and print out the maximum number of bridges that we can build. **O(N)**

  Thus the complexity is going to be $O(N)+O(N^2)+O(N)$ = **$O(N^2)$**.

Output Screenshot

```
Enter The number of cities 1 - 100
10

Numbers Of City :10
North Cities :[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]


South Cities :[3, 2, 10, 4, 8, 5, 7, 1, 9, 6]

LIS: ( 3 4 5 7 9 )

Max number of bridges  =  5
```

Conclusion & Summary

◊ Conclusion:

- At the end we managed to solve this problem using Dynamic programing algorithm Longest increasing subsequence to find the optimal number of bridges. and as a result, we have:
    - The maximum number of bridges we can built (Optimal solution).
    - Pairs of cities the we are going to connect.
- As we mentioned before it can solved using LDS if we sort the north coordinate in decreasing order (n…1) and it can be solved using binary search also to get the maximum number of bridges, so it can be solved in other ways rather than LIS.
- The average number of bridges that we can built for 100 bridge is ,15-17 bridge and at max 19 bridge .

◊ Summary:

- Bridge to nowhere is a dynamic programing problem called building bridges problems which is a common problem in dynamic programing.
- Since we can break the problem into subproblem we can use DP algorithms such as (LIS,LDS,LCS,Matrix chain , knapsack,….)
- We approach LIS and LDS since they are the most suitable In our case and work our way around it .
- Working with that we solved the problem in optimal way.
- Thus solving this kind of problems that require (maximum, minimum, optimal …) are types of problems that we can solve following and modifying the same algorithm we used.

References

1. http://www.historyofbridges.com/facts-about-bridges/bridge-to-nowhere/
2. https://www.geeksforgeeks.org/dynamic-programming-building-bridges/
3. https://www.log2base2.com/algorithms/dynamic-programming/dynamic-programming.html
4. https://www.geeksforgeeks.org/longest-increasing-subsequence-dp-3/
5. https://www.youtube.com/watch?v=SZByPn0deMY
6. https://www.programcreek.com/2014/04/leetcode-longest-increasing-subsequence-java/
7. https://stackoverflow.com/questions/7288585/building-bridges-problem-how-to-apply-longest-increasing-subsequence
8. https://jgrapht.org/