Fibonacci Search

We know that for an ordered list (array), binary search is much efficient than the sequential/linear search. However binary search involves the / (division operator) which may slow down the execution of a program. So one may think of refining this algorithm, so that we can run it more faster. One such technique is the “Fibonacci Search”.

Here the basic idea is to split the list/array using the fibonacci sequence. This changes the binary search algorithm slightly. Instead of halving (making half) the index required to search the key value, we use a fibonacci number to partition the array. Fibonacci search is based on Golden Ratio i.e. 70:30 & not 50:50 like binary search.

Consider the following nos of fibonacci sequence

0 1 1 2 3 5 8 13 21 34 55 89 144 ……

fibo(n) = fibo(n-1) + fibo(n-2)

The term at position fibo(n-1) is 2/3rd of fibo(n) whereas fibo(n-2) is 1/3rd of fibo(n)

E.g. consider the terms 5, 8 & 13

8 is approx 2/3rd of 13. 13 \* 2/3 = 26/3 = 8

5 is approx 1/3rd of 13. 13 \* 1/3 = 13/3 = 4

Differences between fibonacci & binary search

1. fibonacci search divides the array into unequal parts, not so in binary search which divides the array into 2 halves approx of same size.
2. fibonacci search uses addition & subtraction to divide a range unlike binary search which uses division operator. The division operator may prove costly on some CPUs.
3. Fibonacci search examines relatively closer elements in subsequent steps. Hence when the array is too large, then fibonacci search proves to be more efficient than binary search which examines the elements at longer distances.

Steps to implement Fibonacci Search

1. Find an element/term in Fibonacci series such that it is just >= no of elements in array to search
2. Bring the index to search by 1 position to the left so that we can logically partition the array using golden ratio i.e. left (2/3rd) whereas right (1/3rd)
3. If the element at index position matches the key to search, the record is displayed and the search terminates.
4. If the element to search < element at index, ignore the 1/3rd part on right. To achieve this, move to the left in fibonacci series by 1 position. This will create a new partition again using golden ratio
5. If the element to search > element at index, ignore the 2/3rd part on left. As this part is smaller i.e. 1/3rd of total no of elements, add the 1/3rd part of current index to the current index to cover the portion from current index to the last element. Bring f1 and f2 by 2 positions behind as the portion to search is lesser. When we bring f1 & f2 to the left by 2 positions, f1 and f2 will contain smaller nos & that will help us to search in smaller intervals.

noe = 12

index index index index

positions 0 1 2 3 4 5 6 7 8 9 10 11

recs[12] 🡺 2 3 7 9 11 13 20 22 24 32 36 38

(rollnos)

f1 f1 f2 f2

f1 f2

fibo series 0 1 1 2 3 5 8 13

(only upto 13 is OK as noe are only 12)

f1 = 0; // curr\_term

f2 = 1; // next\_term

while (f2 < noe) 1<12, 1<12, 2<12, 3<12, 5<12, 8< 12 🡨 T, 13 < 12 (F)

{

f2 = f1 + f2; // 1, 2, 3, 5, 8, 13

f1 = f2 – f1; // 1, 1, 2, 3, 5, 8

}

f1 = 8;

f2 = 13;

// The 1st index should be @ position < noe

// Bring f1 and f2 back by 1 position

f1 = f2 – f1; // 5

f2 = f2 – f1; // 8

prev\_index = 0;

rno = 20;

while (f2 >= 0) // 8>=0 T, 2 >= 0 T 1 >= 0 T

{

index = prev\_index + f2; // 8, 7, 6

if (rno == recs[index].rollno) // 20 == 24, F 20 == 13, F 20 == 22 F 20 == 20 T

return index; // 6

else if (rno < recs[index].rollno || index >= noe) // search left 2/3rd part 20 < 24, T 20 < 22, T

{

f1 = f2 – f1; // 3, 1

f2 = f2 - f1; // 5, 1

}

else // rno > recs[index].rollno 1/3rd part to search on right 20 > 13, T

{

// bring f1 and f2 behind by 2 positions

f2 = f2 – f1; // 2

f1 = f1 – f2; // 1

prev\_index = index; // 5

}

}

return -1;

}

noe = 7

index

0 1 2 3 4 5 6

rollnos[ ] = 2 6 8 11 13 20 22

rno = 22

noe = 7

f1 = 0;

f2 = 1;

while (f2 < noe) // 1<7 T 1 < 7 T 2 < 7 T 3 < 7 T 5 < 7 T 8 < 7 F

{

f2 = f1 + f2; // 1 2 3 5 8

f1 = f2 – f1; // 1 1 2 3 5

}

f1 = f2 – f1; // 3

f2 = f2 – f1; // 5

prev\_index = index;

while (f2 >= 0) // 5>=0 T, 2 >= 0 T 1>=0 T

{

index = prev\_index + f2; // 0+5 = 5 5+2=7 5+1=6

if (rno == recs[index].rollno) // 22 == 22, T search terminates

return index;

else if (rno < recs[index].rollno || index > noe)

{

f1 = f2 – f1; // 1

f2 = f2 – f1; // 1

}

else // 22 == 20, F 22 > 20 search right 1/3rd part

{

f2 = f2 – f1; // 2

f1 = f1 – f2; // 1

prev\_index = index; // 5

}

}

return -1;