ANGLIA RUSKIN UNIVERSITY  SID:2007201

Algorithm Analysis and data structures (mod007357)

Contents

[Task A: Interpolation Search 2](#_Toc101849121)

[Implementation of interpolation, binary and hybrid search in python 6](#_Toc101849122)

[Bibliography 9](#_Toc101849123)

[Task B: Filial-heir chains 11](#_Toc101849124)

# Task A: Interpolation Search

Binary search algorithm compares the search target with the middle element of the array, if they are not equal, the half in which the target cannot exist will be deleted. This is repeated until search target is found. (BBC, n.d.)



Figure 1 Time complexity of binary search in big-O notation for a successful search

Interpolation search algorithm is an extension of the binary search idea. Instead of comparing search target with the middle element of the array, a prediction will be made using the formula shown in figure 2. (Kaporis, 2006)

A picture containing diagram

Description automatically generated

Figure 2 interpolation formula



Figure 3 Time complexity of interpolation search in big-O notation for a successful case

Hybrid search algorithm is when bad predictions from the interpolation formula is intercepted, and binary search is implemented instead. A bad prediction is defined as: after a prediction more than ¾ of the array is left.

As you can see in figure 1 and 3, the best case of interpolation and binary search is constant. On average, for arithmetic progressions arrays interpolation performs better than binary search. However, in the worse case binary is logarithmic and interpolation is linear, therefore binary performs better in the worst case.

Table

Description automatically generated

Figure 4 shows average number of comparisons for interpolation, hybrid, and interpolation search

Figure 4 is manipulated to form figures 5-9.

Figures 5-9 use an average number of comparison (each array length was ran 10 times then a mean was calculated) and the arrays were randomly generated. This meant that the arrays used were a mixture of arithmetic and geometrics progressions. However, we concluded most of the arrays were GPs.

Chart, line chart

Description automatically generated

Figure shows graph (log scale) for average number of comparisons for interpolation search

Figure 5 shows how as array length increases, the average number of comparisons linearly increases.

Chart, line chart

Description automatically generated

Figure 6 shows how as array length increases, the average number of comparisons gradually increase.

Figure 6 shows graph (log scale) for average number of comparisons for binary search

Figure 7 shows how as array length increases, the average number of comparisons generally increase.

Chart, line chart

Description automatically generated

Figure 7 shows graph (log scale) for average number of comparisons for hybrid search

Chart, line chart

Description automatically generated

Figure 8 shows the average number of comparisons for binary search (shown in blue) interpolation search (shown in orange) and hybrid search (shown in grey) for different array lengths on a line graph using log scale.

Chart, bar chart

Description automatically generated

Figure 9 shows the average number of comparisons for binary search (shown in blue) interpolation search (shown in orange) and hybrid search (shown in grey) for different array lengths on a clustered column graph using log scale.

Figure 8 and 9 clearly indicates hybrid search significantly outperforms interpolation search as the work done in comparisons for interpolation search is much greater than the work done for hybrid search. In addition, hybrid search outperforms binary search, however the performance difference is not as significant in comparison to the difference in interpolation search and hybrid search. As you can see, as the array sizes increases, the difference in work done between interpolation search and hybrid search increases. This indicates that the performance of interpolation search compared to hybrid search gets worse as the array length increases. The best-case scenario in Big-O notation for hybrid search is O(1), this is the same performance as interpolation and binary search. The worst case is O(log n), which is a better performance than interpolation search and the same as binary search.



Figure Time complexity of hybrid search in big-O notation for a successful case

In conclusion, for randomly generated arrays of all lengths, hybrid search has the best performance when regarding number of comparisons as it has the least amount of work done for 4 out of 5 array lengths. Interpolation has the worst performance when regarding number of comparisons as it has the most amount of work done in 4 out of 5 array lengths. This means that if we intercept bad predictions and applied a binary search step instead, performance will be improved, and hybrid search approach outperforms regular binary search.

## Implementation of interpolation, binary and hybrid search in python

import random #allows program to generate random numbers

def interpolationSearch(array, element): #function for interpolation search

print("This is interpolation Search") #users benefit

interpForm = 0

start = 0

end = len(array) -1 #python index starts from 0

#these variables are used in interpolation search formula

NumOfComparison = 0

#setting number of comparison counter = 0

while (start <= end): #this runs until array is empty

interpForm = start + ((end - start) // (array[end] - array[start]) \*

(element - array[start])) #formula for interpolation search is used and saved to variable 'interpForm'

NumOfComparison = NumOfComparison +1 #counter to note down number of comparisons used

if element == array[interpForm]: # checks is element has been correctly

return NumOfComparison #number of comparison occured is returned if search target has been found

if element < array[interpForm]: #checks which side is no longer needeed

NumOfComparison = NumOfComparison +1

end = interpForm - 1 #removes side not needed

else:

NumOfComparison = NumOfComparison +1

start = interpForm + 1 #removes side not needed

print("Number of comparisons {}".format(NumOfComparison))

return -1 #'-1' is returned if search is unnsuccesful

def binarySearch(array, element): #function for binary search

print("This is Binary Search")

mid = 0

start = 0

end = len(array) -1

#these variables are used in binary search formula

NumOfComparison = 0

#setting number of comparison counter = 0

while (start <= end):

mid = (start + end) // 2 #formula to find the middle element

NumOfComparison = NumOfComparison +1 #counter to note down number of comparisons

if element == array[mid]:# checks whether array is in middle element

return NumOfComparison #Search ends when search target is found

if element < array[mid]:#checks which side is no longer needed and can be removed

NumOfComparison = NumOfComparison +1

end = mid - 1 #removes half which is not of interest

else: #checks which side is no longer needed and can be removed

NumOfComparison = NumOfComparison +1

start = mid + 1 #removes half which is not of interest

return -1 #'-1' is returned to indicate an unnsuccesful search

#####################################Hybrid Search#########################################

def interpolationBinarySearch(array, element): #function for hybrid search

print("This is hybrid Search which includes a combination of interpolation and binary search")

interpForm = 0

start = 0

end = len(array) -1

NumOfComparison = 0

mid = 0

#these variables are used in hybrid search formula

while (start <= end):

interpForm = start + ((end - start) // (array[end] - array[start]) \*

(element - array[start])) #formula for interpolation search is used and saved to variable 'interpForm'

NumOfComparison = NumOfComparison +1 #counter to note down number of comparisons used

if element == array[interpForm]: # checks whether prediction is correct

return NumOfComparison

if element < array[interpForm]:

if round((end-start+1)/4)<=end-interpForm+1: #checks whether the prediction is a 'bad' prediction.

#A bad prediction is defined as: more than 3/4 of the remaining array is left after a prediction

NumOfComparison = NumOfComparison +1

end = interpForm - 1

else: # if remaining array is more than 3/4

# A binary search is used when the predicted key leaves 3/4 of the array left

mid = start + ((end - start) // 2) #formula to find the middle element

print("Number of comparisons {}".format(NumOfComparison))

NumOfComparison = NumOfComparison +1 #counter to note down number of comparisons used NEW

if element == array[mid]: #checks whether middle element is search target

return NumOfComparison #search target is found in middle element

if element < array[mid]:

NumOfComparison = NumOfComparison +1

end = mid - 1

else:

NumOfComparison = NumOfComparison +1

start = mid + 1

else:

start + ((end - start) // 2)

if round((end-start+1)/4)<=interpForm - start + 1: #checks whether the prediction is a 'bad' prediction.

NumOfComparison = NumOfComparison +1

start = interpForm + 1

else:

mid = start + ((end - start) // 2)

NumOfComparison = NumOfComparison +1

if element == array[mid]:

return NumOfComparison

if element < array[mid]:

NumOfComparison = NumOfComparison +1

end = mid - 1

else:

NumOfComparison = NumOfComparison +1

start = mid + 1

return -1

def randomArrayGenerator(arraylen):#function to generate a random array with the array length as a parameter

array = []

for x in range (arraylen):

array.append(random.randint(1,arraylen)) #generate random numbers between 1 and 'arraylen' which is given as a parameter when function is called

array.sort() #sorts array as binary and interpolation search only works with sorted arrays

return array

array = randomArrayGenerator(100) #calling function with the parameter being the desired array length

element = random.choice(array) # The search target is randomised and is chosen from the randomly generated array

print(interpolationSearch(array, element)) #calls interpolation search function

print(binarySearch(array, element)) #calls binary search function

print(interpolationBinarySearch(array, element)) #calls hybrid search function

# Bibliography

BBC, n.d. *Digital design principles - CCEA.* [Online]   
Available at: https://www.bbc.co.uk/bitesize/guides/zts8v9q/revision/5  
[Accessed April 2022].

Kaporis, A. M. C. S. S. T. A. T. K. Z., 2006. *Dynamic interpolation search revisited.* [Online]   
[Accessed April 2022].

# 

# Task B: Filial-heir chains

**Theorem 1.**

If there are f2ary leaf nodes in a perfect 2-ary tree (k=2), the number of leaf nodes in the equivalent filial-heir chain, is given by:

Chart

Description automatically generated

Figure 11 Demonstration of theorem 1

**Theorem 2.**

If there are H heights in a filial-heir chain, the height in the equivalent k-ary tree if it is perfect, is given by:

(This theorem is generalized and will work for all values of K, as shown in the demonstrations)

A picture containing chart

Description automatically generated

Figure 12 Demonstration of theorem 2, where k=2

Chart

Description automatically generated with low confidence

Figure 13 Demonstration of theorem 2, where k=3

**Theorem 3.**

If there are f2ary leaf nodes in a perfect 2-ary tree (k=2), the number of nodes in the equivalent filial-heir chain, is given by:

Chart

Description automatically generated

Figure 14 Demonstration of theorem 3

**Theorem 4.**

If there are fkary nodes in a perfect k-ary tree, the number of edges in the equivalent filial-heir chain, is given by

(This theorem is generalized and will work for all values of K, as shown in the demonstrations)

Chart

Description automatically generated

Figure 15 Demonstration of theorem 4, where K=2

Chart

Description automatically generated

Figure 16 Demonstration of theorem 4, where K=3

**Theorem 5.**

If there are fkary leaf nodes in a perfect k-ary tree, the number of leaf nodes in the equivalent filial-heir chain, is given by:

(This theorem is generalized and will work for all values of K, as shown in the demonstrations)

Chart

Description automatically generated

Figure 17 Demonstration of theorem 5, where K=2

Chart

Description automatically generated

Figure Demonstration of theorem 5, where K=3

**Theorem 6.**

If there are fkary nodes in a perfect k-ary tree, the number of nodes in the equivalent filial-heir chain, is given by:

(This theorem is generalized and will work for all values of K, as shown in the demonstrations)

Chart, radar chart

Description automatically generated

Figure 19 Demonstration of theorem 6, where K=2

Chart

Description automatically generated

Figure 20 Demonstration of theorem 6, where K=3

**Theorem 7.**

If there are fkary edges in a perfect k-ary tree, the number of edges in the equivalent filial-heir chain, is given by:

(This theorem is generalized and will work for all values of K, as shown in the demonstrations)

A picture containing chart

Description automatically generated

Figure 21 Demonstration of theorem 7, where K=2

A picture containing chart

Description automatically generated

Figure 22 Demonstration of theorem 7, where K=3

**Theorem 8.**

If there are NP null pointers and N nodes in a perfect k-ary tree, the number of nodes in the equivalent filial-heir chain, is given by:

(This theorem is generalized and will work for all values of K, as shown in the demonstrations)

Chart

Description automatically generated with medium confidence

Figure 23 Demonstration of theorem 8, where K=2

Chart

Description automatically generated

Figure 24 Demonstration of theorem 8, where K=3

**Theorem 9.**

If there are NP null pointers and H height in a perfect 2-ary tree, the height in the equivalent filial-heir chain, is given by:

A picture containing scatter chart

Description automatically generated

Figure Demonstration of theorem 9

**Theorem 10.**

If there are NP null pointers and I interior nodes in a perfect 2-ary tree, the number of interior nodes in the equivalent filial-heir chain, is given by: (we assume the root node is an interior node) (This theorem is generalized and will work for all values of K, as shown in the demonstrations)

Chart, scatter chart

Description automatically generated

Figure Demonstration of theorem 10, where K = 2

Chart

Description automatically generated

Figure Demonstration of theorem 10, where K = 3