Ben-Gurion University of the Negev

Data Structures

202.1.1031

Assignment No. 4 - Solution

Author:

Publish date: 20.5.2023

Submission date: 10.06.2023

אוניברסיטת בן-גוריון בנגב

Ben-Gurion University of the Negev

Contents

Tasks and Questions

2 Skip-List

2.1 Warm-Up and Familiarization

2.1.1 Implementation of Abstract Functions

Answer 2.1: Implementation in code

2.1.2 Analysis of the Probabilistic Process

Answer 2.2: The tables are:

p = 0.33							
x	$\hat{\ell}_1$	$\hat{\ell}_2$	$\hat{\ell}_3$	$\hat{\ell}_4$	$\hat{\ell}_5$	**Expected Level** ($E[\ell]$)	**Average delta** ($\frac{1}{5} \cdot \sum_{i=1}^{5} (\hat{\ell}_i - E[\ell])$)
10	3.000	4.200	2.800	3.100	4.1000	3.030	0.514
100	2.870	2.860	3.150	3.250	3.420	3.030	0.212
1000	3.057	2.997	3.173	3.017	3.167	3.030	0.071
10000	2.991	3.011	3.019	3.045	3.015	3.030	0.020

p = 0.5							
x	$\hat{\ell}_1$	$\hat{\ell}_2$	$\hat{\ell}_3$	$\hat{\ell}_4$	$\hat{\ell}_5$	**Expected Level** ($E[\ell]$)	**Average delta** ($\frac{1}{5} \cdot \sum_{i=1}^{5} (\hat{\ell}_i - E[\ell])$)
10	2.400	1.700	2.300	2.200	2.000	2.000	0.240
100	2.000	2.080	1.840	2.040	2.060	2.000	0.068
1000	2.009	2.067	2.048	1.962	2.065	2.000	0.045
10000	1.987	2.016	2.011	2.012	1.993	2.000	0.012

p = 0.75							
x	$\hat{\ell}_1$	$\hat{\ell}_2$	$\hat{\ell}_3$	$\hat{\ell}_4$	$\hat{\ell}_5$	**Expected Level** ($E[\ell]$)	**Average delta** ($\frac{1}{5} \cdot \sum_{i=1}^{5} (\hat{\ell}_i - E[\ell])$)
10	1.600	1.300	1.800	1.700	1.300	1.333	0.233
100	1.290	1.270	1.290	1.530	1.280	1.333	0.080
1000	1.347	1.358	1.306	1.309	1.337	1.333	0.019
10000	1.333	1.332	1.324	1.331	1.330	1.333	0.003

p = 0.9							
x	$\hat{\ell}_1$	$\hat{\ell}_2$	$\hat{\ell}_3$	$\hat{\ell}_4$	$\hat{\ell}_5$	**Expected Level** ($E[\ell]$)	**Average delta** ($\frac{1}{5} \cdot \sum_{i=1}^{5} (\hat{\ell}_i - E[\ell])$)
10	1.100	1.200	1.500	1.000	1.000	1.111	0.142
100	1.150	1.100	1.130	1.220	1.140	1.111	0.041
1000	1.118	1.115	1.108	1.110	1.100	1.111	0.005
10000	1.116	1.109	1.109	1.113	1.110	1.111	0.003

Answer 2.3: The higher the probability for success the lower the average number of levels. (P is in inverse proportion to the average height)

Answer 2.4: The larger the sample size (x) we are using the lower the delta we get. This is because over a lot of operations the random elements even out.

2.1.3 Analysis of the operations

Answer 2.5: Implementation in code

Answer 2.6: The tables are:

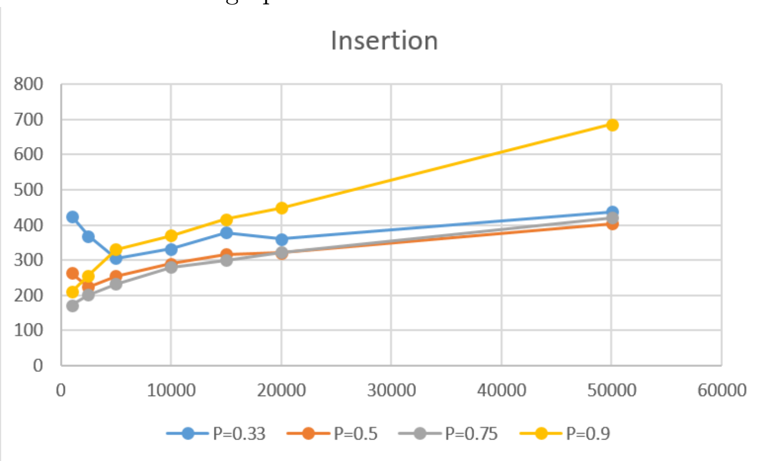
p = 0.33			
x	Average Insertion	Average Search	Average Deletion
1000	425.393	157.68	71.093
2500	368.808	178.936	36.388
5000	305.593	195.469	35.445
10000	331.812	214.872	40.699
15000	378.559	247.66	43.532
20000	359.962	269.007	50.1
50000	437.585	402.648	75.401

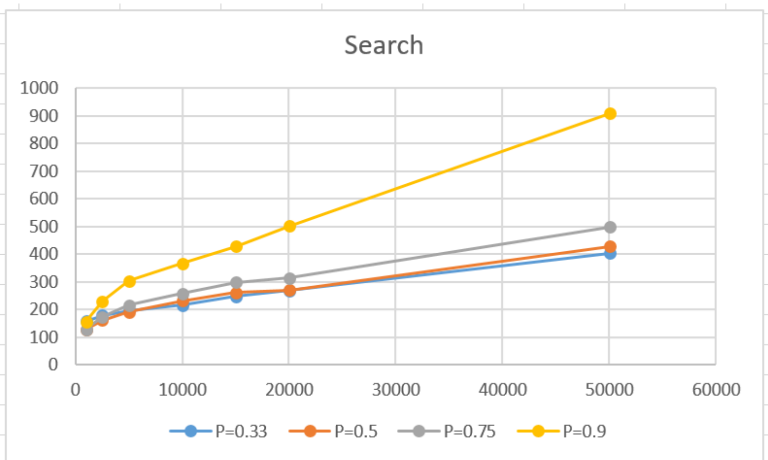
p = 0.5			
x	Average Insertion	Average Search	Average Deletion
1000	263.62	127.465	20.273
2500	224.68	160.755	20.38
5000	254.393	191.164	23.582
10000	290.292	231.926	36.012
15000	317.342	260.408	36.786
20000	321.419	269.433	37.494
50000	403.929	428.393	62.658

p = 0.75			
x	Average Insertion	Average Search	Average Deletion
1000	172.38	128.478	17.753
2500	201.575	172.668	15.352
5000	232.879	215.858	17.533
10000	280.273	258.018	24.16
15000	299.252	297.724	26.178
20000	322.698	313.567	27.05
50000	420.842	498.215	50.269

p = 0.9			
x	Average Insertion	Average Search	Average Deletion
1000	210.327	156.66	11.507
2500	255.7	229.199	13.279
5000	331.055	303.192	21.68
10000	369.724	365.948	21.682
15000	416.744	427.251	27.954
20000	449.097	501.986	27.703
50000	686.795	908.683	67.923

Answer 2.7: The graph are:





Answer 2.8: **Insertion** The average insertion time is the lowest when P is around the halfway point between 0 and 1. that because when P is too high the generate-height function and thus the linking on each level takes more time (because there are more levels). And also the find function used in the insert operation take longer due to the list being higher (that's because we also need to go down as well as to the right).
And if the P value is too low there won't be enough variety in different nodes' heights, making the find function more and more similar to linear search - thus making it slower.
Search The lower the P value, the lower the average search time, up to a certain threshold, due to the same reason as in Insertion.
Delete The lower the P, the higher the average deletion time.

Answer 2.9: Let SL be a skip-list with n elements.
The probability that an element x is in the list S_i is P^i, And the size of S_i has binomial distribution with parameters n and P^i. So $E[|S_i|] = n * P^i$
Therefore, the expected number of nodes is: $E[\Sigma_{i=0}^{\inf}|S_i|]$ and using the linearity of expectation we get $\Sigma_{i=0}^{\inf} E[|S_i|] = \Sigma_{i=0}^{\inf} n * P^i = n * \Sigma_{i=0}^{\inf} P^i$ and using the formula for sum of infinite geometric series with $|quotient| < 1$ we get $n * \frac{1}{1-P} = \frac{n}{1-P}$
Thus the expected number of nodes in a conceptual Skip-List containing elements, not including the Sentinels is $\frac{n}{1-P}$

Answer 2.10: For each i > 0 define the indicator random variable:

$$X_i = \begin{cases} 0, & \text{if } S_i \text{ is empty} \\ 1, & \text{if } S_i \text{ is not empty} \end{cases} \quad (1)$$

The height h of the skip list is then given by: $h = \Sigma_{i=0}^{\inf} X_i$
Note that $X_i \leq |S_i|$, thus $E[X_i] \leq E[|S_i|] = n * P^i$
Additionally, $Xi \leq 1$ so $E[Xi] \leq 1$

$$E[h] = E[\Sigma_{i=1}^{\inf} X_i] = \Sigma_{i=1}^{\inf} E[X_i]$$

Auxiliary calculation: $n * P^i = 1 \Rightarrow \mathrm{P}^i = \frac{1}{n} \Rightarrow \mathrm{i} = \log_P(\frac{1}{n})$

$$\Sigma_{i=1}^{\inf} E[X_i] = \Sigma_{i=1}^{\lfloor \log_P(\frac{1}{n}) \rfloor} E[X_i] + \Sigma_{\lfloor \log_P(\frac{1}{n}) \rfloor + 1}^{\inf} E[X_i]$$
$$\leq \Sigma_{i=1}^{\lfloor \log_P(\frac{1}{n}) \rfloor} 1 + \Sigma_{\lfloor \log_P(\frac{1}{n}) \rfloor + 1}^{\inf} n * P^i$$

Auxiliary calculation: $\Sigma_{\lfloor \log_P(\frac{1}{n}) \rfloor+1}^{\inf} n * P^i = n * P^{\lfloor \log_P(\frac{1}{n}) \rfloor+1} + n * P^{\lfloor \log_P(\frac{1}{n}) \rfloor+2} + ... =$
$n * P^{\lfloor \log_P(\frac{1}{n}) \rfloor+1} * [\Sigma_{i=0}^{\inf} P^i] = n * P^{\lfloor \log_P(\frac{1}{n}) \rfloor+1} * \frac{1}{1-P} \leq n * P^{\log_P(\frac{1}{n})} * \frac{1}{1-P} = n * \frac{1}{n} * \frac{1}{1-P} = \frac{1}{1-P}$

$\Sigma_{i=1}^{\lfloor \log_P(\frac{1}{n}) \rfloor} 1 + \Sigma_{\lfloor \log_P(\frac{1}{n}) \rfloor+1}^{\inf} n * P^i = log_P(\frac{1}{n}) + \frac{1}{1-P}$

So the expected height of a Skip-List as a function of p is most $log_P(\frac{1}{n}) + \frac{1}{1-P}$

Answer 2.11: As shown in lectures, $E[X] = \frac{1}{P}$.

Let R_i denote the number of steps the path has at S_i.

The number of steps of a reverse path is some S_i is a random variable which is similar to a geometric distribution random variable. except that we don't count the last coin flip and we stop when reaching a sentinel, so $R_i \leq X_i - 1$.

Using linearity of expectation we get: $E[R_i] \leq E[X_i - 1] = E[X_i] - 1 = \frac{1}{1-P} - 1 = \frac{P}{P-1}$.

Additionally, $R_i \leq |S_i|$ by the way it was defined, therefore: $E[R_i] \leq E[|S_i|] = n * P^i$.

Let R by the length of the search path for some search value.
$R = h + \Sigma_{i=0}^{\inf} R_i$.

Auxiliary calculation: $n * P^i = 1 \Rightarrow P^i = \frac{1}{n} \Rightarrow i = \log_P(\frac{1}{n})$

So we'll get $E[R] = E[h + \Sigma_{i=0}^{\inf} R_i] = E[h] + \Sigma_{i=0}^{\inf} E[R_i] = E[h] + \Sigma_{i=0}^{\lfloor \log_P(\frac{1}{n}) \rfloor} E[R_i] + \Sigma_{i=\lfloor \log_P(\frac{1}{n})+1 \rfloor}^{\inf} X_i]$

$\leq E[h] + \Sigma_{i=0}^{\lfloor \log_P(\frac{1}{n}) \rfloor} \frac{P}{1-P} + \Sigma_{i=\lfloor \log_P(\frac{1}{n})+1 \rfloor}^{\inf} n * P^i$

$\leq log_P(\frac{1}{n}) + \frac{1}{1-P} + (log_P(\frac{1}{n}) + 1) * (\frac{P}{1-P}) + \frac{1}{1-P}$

$= log_P(\frac{1}{n}) + \frac{1}{1-P} + \frac{(P * \log(\frac{1}{n})) + P}{1-P} + \frac{1}{1-P}$

$= \frac{log_P(\frac{1}{n}) + 2 + P}{1-P}$

Thus the expected time complexity of Insertion/Search as a function of p is: $\frac{log_P(\frac{1}{n}) + 2 + P}{1-P}$

Comparing this to the result we've got in task 2.6 it seems that the result are obeying this formula; as very high P value (close to 1) bump the time due to the fact that the numerator increase and the denominator decreases 1-P.

2.2 Order Statistics

Answer 2.12: Sorry I didn't have enough time. this one is a bit unfair.

3 Hashing

3.1 Introduction

3.1.1 Hash Functions

Answer 3.1: After we perform true Modulus operation the result is always non-negative. and thus the sign bit of the result is 0.
>>> is unsigned-shift, so performing it (w - k) times result in shifting everything to the right (w - k) times and filling it with 0s.
And when looking at a binary number, shifting it to the right is equivalent to dividing it by 2 (integer division) (w - k) times, or $2^{(w-k)}$ - thus the two equation are equals.
Sources: https://stackoverflow.com/questions/2811319/difference-between-and
https://stackoverflow.com/questions/5385024/mod-in-java-produces-negative-numbers

Answer 3.2: We can look at the object's binary representation and collect it into chunks of 8 bits (fill the reminder of the last one with 0s if needed). After that look at each chunk as a single ASCII character and use the Carter-Wegman hashing for strings on the resulted string.

3.2 Hash Implementations

Answer 3.3: Implementation in code

Answer 3.4: Implementation in code

Answer 3.5: Implementation in code

3.3 Hash Tables

3.3.1 Introduction

Answer 3.6: Implementation in code

Answer 3.7: Implementation in code

Answer 3.8: The results are:

Linear Probing		
max α	Average Insertion	Average Search
1/2	38.662	40.893
3/4	33.784	57.281
7/8	34.242	95.738
15/16	40.789	287.951

Answer 3.9: The higher the load factor, the slower each operation takes, and it approach O(n) as the load factor approach 1. That's because in each traversal of the table we need to look for an empty cell for longer time (because the table is in higher capacity).

Answer 3.10: The results are:

Chaining		
max α	Average Insertion	Average Search
1/2	101.138	47.909
3/4	55.848	36.738
1	55.413	42.524
3/2	62.837	54.872
2	61.570	57.587

Answer 3.11: The higher the load factor the higher the search on averge, that's because in addition to going into the right cell we also need to perform linear search in the linked list to find the item (if exist). But the insertion stays relatively contant because we inserting element to the head of the list - making its length irrelevant.

Answer 3.12: The results are:

Long operations		
max α	Average Insertion	Average Search
1	91.441	40.1893

Sting operations		
max α	Average Insertion	Average Search
1	9922.089	9993.572

We can deduce that hashing strings is a lot more expensive that hashing numbers using the Dietzfelbinger et al hashing algorithm.

3.4 Theoretical Questions

Answer 3.13: The algorithm for finding the Successor is a brute force over all the elements of the hash table (one list at a time) until we find smallest number larger than val (aka the successor) .
Because we go over the entire data-structure one item at a time the complexity is O(n)

Require: val
$current - value \leftarrow \infty$
for map[0] to map[map.size() - 1] **do**
    $list \leftarrow map[i]$
    for element in list **do**
        if $(element < current - value)$ & $(element > val)$ **then**
            $current - value \leftarrow element$
        end if
    end for
end for
return current-value

Answer 3.14: The algorithm for finding the Minimum is a brute force over all the elements of the hash table (one list at a time) until we find smallest number in the DS.
Because we go over the entire data-structure one item at a time the complexity is O(n)

Require: val

$current - value \leftarrow \infty$
for map[0] to map[map.size() - 1] **do**
  $list \leftarrow map[i]$
  for element in list **do**
    if $(element < current - value)$ **then**
      $current - value \leftarrow element$
    end if
  end for
end for
return current-value

Answer 3.15: The algorithm for finding the Minimum is a brute force over all the elements of the hash table (one list at a time) and count how many numbers are less than or equal val.
Because we go over the entire data-structure one item at a time the complexity is O(n)

Require: val

$counter \leftarrow 0$
for map[0] to map[map.size() - 1] **do**
  $list \leftarrow map[i]$
  for element in list **do**
    if $(element <= val)$ **then**
      $counter \leftarrow counter + 1$
    end if
  end for
end for
return current-value

Answer 3.16: The algorithm for finding the ith number in the hash table is finding the maximun of the DS in O(n) and using the radix sort algorithm from class and going to the ith element Because we go over the entire data-structure to find the maximun in O(n) and then perform radix sort in O(n) the overall time complexity is O(n)

4 Designing a data structure according to given specifications

Answer 4.1: sorry don't have enough time :(
"The time is short and the work is a lot, the students are lazy and the Landlord knocks" - pirkey avot

Good Luck!