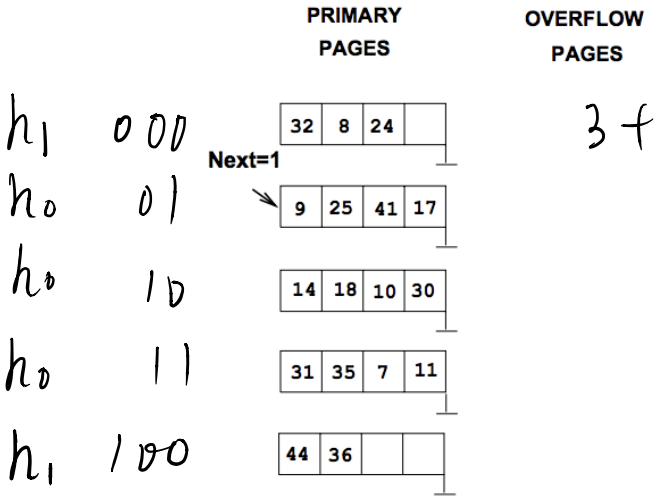
Ziqi Tan U88387734

Boston University
CAS CS 562: Advanced Database Applications
Homework #1
Fall 2020

Due Date: Feb 20, 2020 at 11:59PM, please submit a PDF file using Gradescope

Problem 1. (Linear Hashing)



split policy

Consider the Linear Hashing index shown above. Assume that we split whenever an overflow page is created. Also, assume that we use $N = 4$ and the family functions of $h_i(x) = h(x) \ mod \ 2^i * N$. We initially start with $h_0 = h(x) \ mod \ 4$ and $h_1(x) = h(x) \ mod \ 8$. The capacity of each bucket (page) is 4.

Answer the following questions about this index:

1. What can you say about the last entry that was inserted into the index?
2. What can you say about the last entry that was inserted into the index if you know that there have been no deletions from this index so far?
3. Suppose you know that there have been no deletions from this index so far. What can you say about the last entry whose insertion into the index caused a split?
4. Show the index after inserting an entry with hash value 4.
5. Show the index after inserting an entry with hash value 15 into the original index.
6. Find a list of entries whose insertion into the original index would lead to a bucket with two overflow pages. Use as few entries as possible to accomplish this. What is the maximum number of entries that can be inserted into this bucket before a split occurs that reduces the length of this overflow chain?

1. We cannot know which entry is the last insertion.

2. Although we have already known there have been no deletion, we still cannot know which entry is the last insertion. It could be one of {24, 17, 30, 11, 36}.

3. Answer: the last entry 24 or 36 whose insertion caused a split.

Why:

① The last split bucket is the 00 bucket.

② The sum of the number of entries of bucket 000 and 100 is 5 which is bigger than the bucket capacity of 4.

4. Next = 01

$h_0(4) = 4 \bmod 4 = 0 < Next$

$\therefore h_1(4) = 4 \bmod 8 = 4$ (100)

Index:

	32	8	24
	9	25	41	17
	14	18	10	30
	31	35	7	11
100	44	36	4	

5. Next = 01

$h_0(15) = 15 \bmod 4 = 3 > Next$

Index :	h_1	000	32	8	24	
Next →	h_0	01	9	25	41	17
	h_0	10	14	18	10	30
	h_0	11	31	35	7	11 → 15
	h_1	100	44	36		

overflow page

It causes split!

	h_1	000	32	8	24	
	h_1	001	9	25	41	17
Next →	h_0	10	14	18	10	30
	h_0	11	31	35	7	11 → 15
	h_1	100	44	36		
	h_1	101				

Sketch: $9 = 1001_b$

$25 = 11001_b$

$41 = 101001_b$

$17 = 10001_b$

6. Example: [57, 121, 73, 89, 105]

(a) Insert a list of entries that modulo 8 is 001.

(b) There may be two possible answers.

Answer 1:

The maximum number is infinite, because for example, you can insert a list of entries that modulo 100…00 (having K zeros) is 000…001 (having k-1 zeros). By doing that, the overflow page chain of bucket 000…001 (having k-1 zeros) will not get shorter.

(b)

Answer 2:

The maximum number is 16.

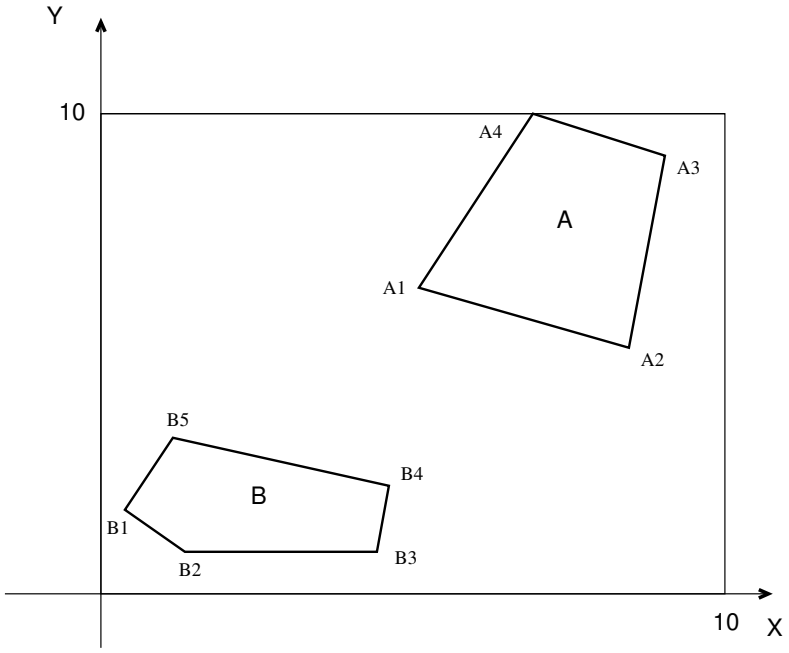
Why?

Although we can insert entries like 6(a) to make the overflow page chain as long as possible, the entries 9, 25, 41 will be assigned to a new bucket when the Next Pointer return to 001.

Problem 2. (Space Filling Curves)

a) Compute the Z-values and the Hilbert values for the points (10, 26) and (55, 63) for $K = 2$ and $K = 4$, where K is the number of bits per dimension.Assume that the data space is [0, 100) x [0, 100).

b) Compute the Z-values of regions **A** and **B** in the figure below, where A1=(5.5, 6.5), A2=(8.5, 5), A3=(9.1, 9.2), A4=(7, 10), B1=(0.4, 1.8), B2=(1.2, 0.6), B3=(4, 0.8), B4=(4.5, 2.3) and B5=(0.9, 3.2).Assume that $K = 4$ bits, the maximum number of bits per dimension ($2^4 = 16$ values per axis).



HINT: Methods to compute z- and Hilbert- values can be found in the following paper:
H. V. Jagadish: *Linear Clustering of Objects with Multiple Attributes.* **ACM SIGMOD Conference** 1990, pages 332-342.

Problem 3. (R-trees)

Let D be a 2-dimensional point dataset and $p = (x, y)$ a point in that set. The coordinates of all points are positive. Consider the function: $f(p) : D \rightarrow R$, where $f(p) = a_1x + a_2y$ and $a_1 + a_2 = 1$. The values for a_1 and a_2 are given by the user. The idea is that each user gives different importance (weight) to different attributes. We want to find the point (or points) that maximize(s) this function. This type of queries are called *preference* queries. Now, assume that an R-tree is used to store the dataset D.

(a) Design an efficient search procedure that uses the R-tree to find the point(s) that maximize the function f. Give the pseudo-code of the algorithm and explain how it works.

(b) What is the property that allows the design and guarantees the correctness of your algorithm? Explain.

Problem 2

(a) $k=2$ for $(10, 26)$ Z

$100 \div 2 = 50$ $10 < 50, 26 < 50$ 0 0

$50 \div 2 = 25$ $10 < 25, 26 > 25$ 0 1

$\therefore$ Z value of $(10, 26)$ is 0001. 1

for $(55, 63)$ Z

$100 \div 2 = 50$ $55 > 50, 63 > 50$ 1 1

$50 + 50 \div 2 = 75$ $55 < 75, 63 < 75$ 0 0

$\therefore$ Z value of $(55, 63)$ is 1100. 12

$k=4$ for $(10, 26)$ Z

$100 \div 2 = 50$ $10 < 50$, $26 < 50$ 0 0

$50 \div 2 = 25$ $10 < 25$, $26 > 25$ 0 1

$25 \div 2 = 12.5$ $10 < 12.5$ 0 0

$25 + 25 \div 2 = 37.5$ $26 < 37.5$

$12.5 \div 2 = 6.25$ $10 > 6.25$ 1 0

$37.5 - 6.25 = 31.25$ $26 < 31.25$

$\therefore$ Z value of $(10, 26)$ is 0001 0010. 18

for (55, 63) Z

100 ÷ 2 = 50 55 > 50, 63 > 50 11

50 + 50 ÷ 2 = 75 55 < 75, 63 < 75 00

75 − 50 ÷ 2 ÷ 2 = 62.5 55 < 62.5, 63 > 62.5 01

62.5 + 50 ÷ 2 ÷ 2 ÷ 2 = 68.75 55 < 56.25 00

62.5 − 50 ÷ 2 ÷ 2 ÷ 2 = 56.25 63 < 68.75

∴ Z-value of (55, 63) is 1100 0100. 196

---

sketch:

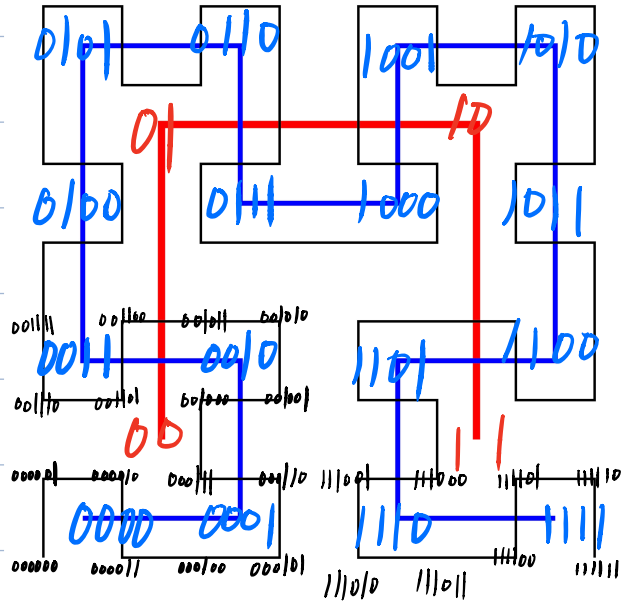
if 00: swap. Don't change.

Don't change. swap.

if 11: Don't change swap.

swap. Don't change.

Map



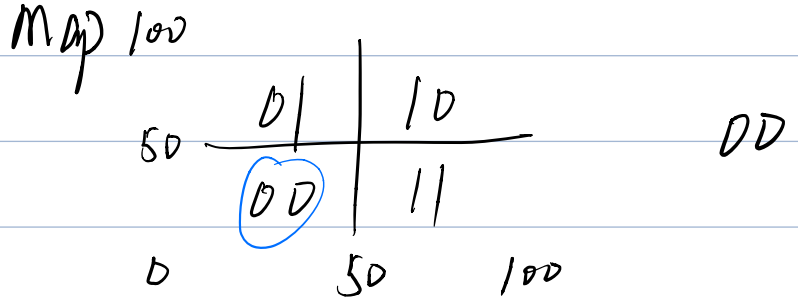
---

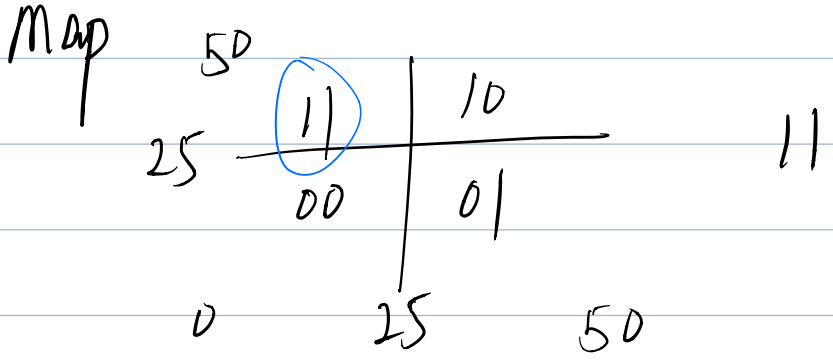
k = 2 (10, 26) Hilbert value 0011. 3

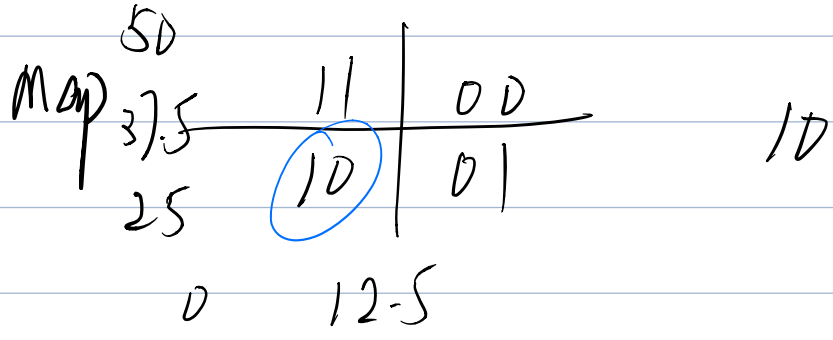
(55, 63) 1000. 8

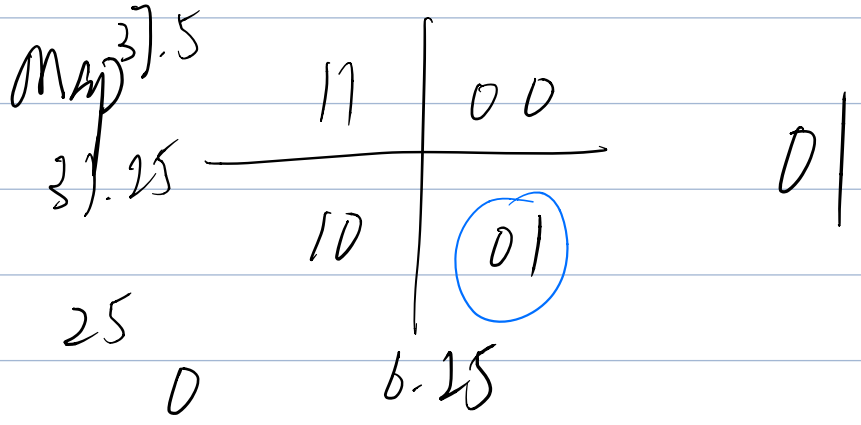
K=4 (10, 26) Hilbert 0011 1001 57

sketch:

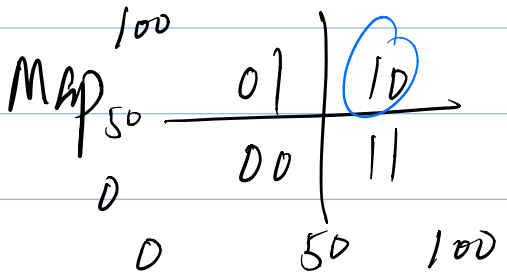




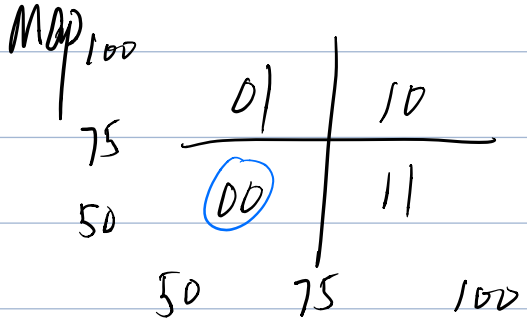




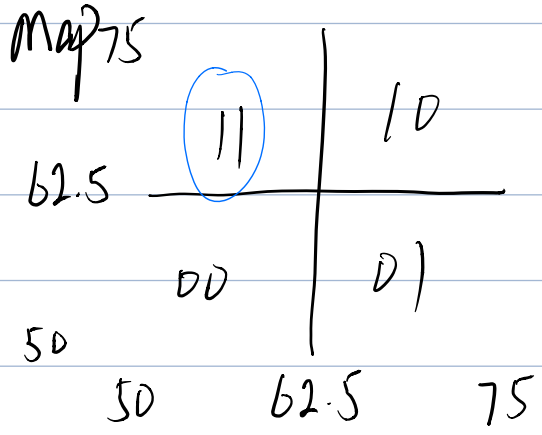
k=4 (55, 63) Hilbert value 10 00 11 10 142



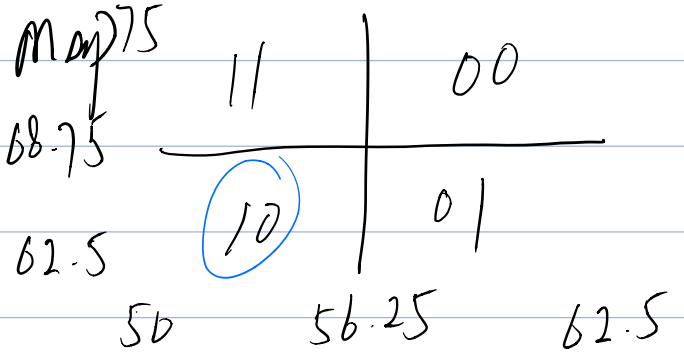
10



00



11



10

Problem 2 (b) $k = 4$

Space $[0, 10) \times [0, 10)$

z values

A_1 1100 0110 198

A_2 1110 0010 226

A_3 1111 1100 252

A_4 1101 1111 223

Region A:

195 ~ 196, 198 ~ 199, 201 ~ 207, 210 ~ 211, 216 ~ 231, 236 ~ 237, 240 ~ 249, 252

B_1 0000 0100 4

B_2 0000 0010 2

B_3 0010 1001 41

B_4 0010 1111 47

Region B:

2 ~ 9, 11 ~ 15, 18 ~ 19, 24, 26, 33, 35 ~ 39, 41, 44, 45, 47, 48, 50

Methodology:

Use geometry to find all little cells and find all z values of these cells.

Problem 3 (a) $\underset{x,y}{\text{maximize}}\ f(p) = a_1 x + a_2 y$

s.t. $x > 0,\ y > 0,\ a_1 > 0,\ a_2 > 0,\ a_1 + a_2 = 1$

Therefore, we only need to find the biggest x and y.

Notations:

Define M as the upper-right point of a MBR and M_x and M_y as the x coordinate and y coordinate respectively.

(b) Obviously, the upper bound of $f(p)$ in a MBR is $f(M) = a_1 M_x + a_2 M_y$.

Define a priority queue (PQ) where the elements are in descending order.

Algorithm:

1. Initialize PQ.
   (Push the $f(M)$ of the root MBR into PQ).
2. while (PQ is not empty)
   object ← PQ.pop();

```
if (object is a point)
    return object;

else
    // This object is a MBR
    Push the f(M) of all the children MBR/points
    of this object into PQ.
end if
end while
```

Problem 4. (General)

Answer the following questions about some of the methods that we discussed so far:

(a) Consider a *range counting query*, where the query asks for the number of points inside a range. Assume that you have a dataset of 2-d points and you want to create an index based on kd-trees to answer efficiently range counting queries. Explain how you will modify the original kd-tree data structure and give an algorithm that will use this modified kd-tree to answer efficiently range counting queries. What is the worst case performance of your data structure for range counting queries?

(b) Discuss the different splitting policies of the Grid File and what are the pros and cons of each method. Which method you would use in your implementation? Why? Explain.

(c) What is the **paging algorithm** of the LSD-tree and how we use it?

Problem 4 (a)

Data structure:

Each node records its number of nodes of the rectangle region it represents, which means the number of points surrounded by splitting lines.

We can still use the range queries algorithm for kd-tree.

Algorithm:

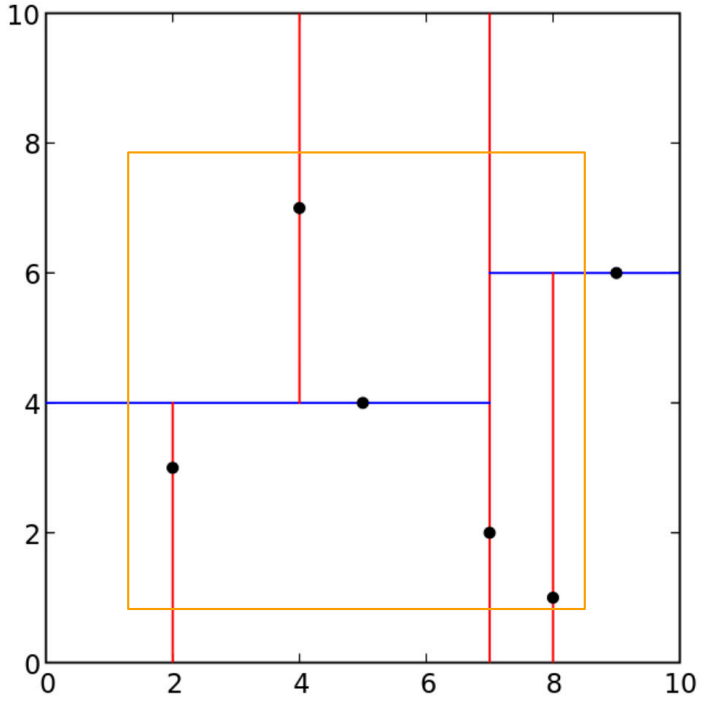
Initialize pointer V = root. $N = 0$ (the number of points to report).

```
Search (v, R):
    // R is the query region
    If v is a leaf
        N = N + 1 if v lies in R
    else
        if Reg(v) is contained in R
            N = N + Reg(v).getNumOfPoints()
        else
            if Reg(left(v)) intersects R,
                Search(left(v), R)
            if Reg(right(v)) intersects R
                Search(right(v), R)
```

Complexity: $O(\sqrt{n})$

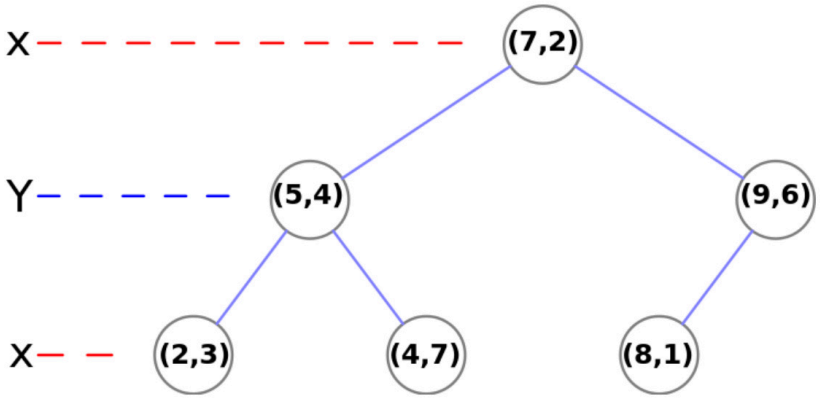
$$\begin{cases} Q(1) = O(1) \\ Q(N) = 2 + 2Q\left(\frac{N}{4}\right) \end{cases}$$

Worst case: the query region intersects all rectangles and contains no rectangle.



k-d tree decomposition for the point set

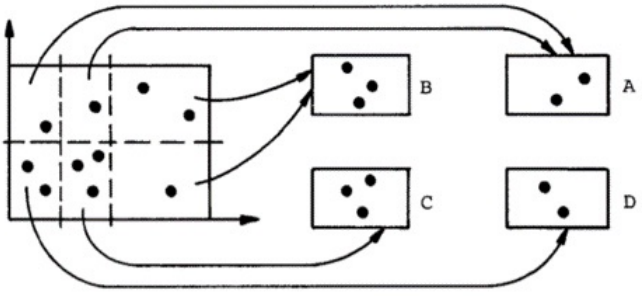
`(2,3), (5,4), (9,6), (4,7), (8,1), (7,2)`.



The resulting *k*-d tree.

Problem 4 (b)

The simplest splitting policy is to choose the dimension according to a fix schedule.



Pros: Simple and easy to implement.
Cons: The directory grows exponentially.

Other policy: split the corresponding dimensions more often than others.
This has the effect of increasing the precision of answers to partially specified queries in which the favored attributes is specified.

Problem 4 (c)

Paging algorithm is used to overcome the following drawback of multidimensional binary tree structures.

- A multidimensional binary tree may become unbalanced.

The paging algorithm is called when after an insertion of an additional node, the size of the internal prefix tree T_i reaches the maximal possible number of internal nodes.