COMP207 Tutorial ExerciseSolutions
Week 6 (9th–11th November)

The exercises below provide the opportunity to practice the concepts and methods discussed during this week's videos, i.e. the ones on query processing.

If you rather want to do something more practical to do with optimization AND B+ trees/indices and have access to a computer, you can instead try to do the tutorial on how to use index in MySQL `https://www.mysqltutorial.org/mysql-index/mysql-create-index/` (the optimization part comes from it showing you how to see how MySQL have optimized queries, i.e. using the EXPLAIN command in front of queries). You will need to use the sample database from `https://www.mysqltutorial.org/mysql-sample-database.aspx` to do that tutorial. It gives you a zip file containing just an sql file that creates the sample database. You need to run the sql file either using the command line or the Workbench, after connecting.

You can also try, now or when you next have access to the internet, to play around with the online visualization tool for B+ trees on `https://www.cs.usfca.edu/~galles/visualization/BPlusTree.html`. The references for the lectures on the Canvas course page provide some entry points.

Solutions to most of the questions will be be provided Friday like normal, except for the last one, which would be too long to give a solution for during a video.

Query Plans

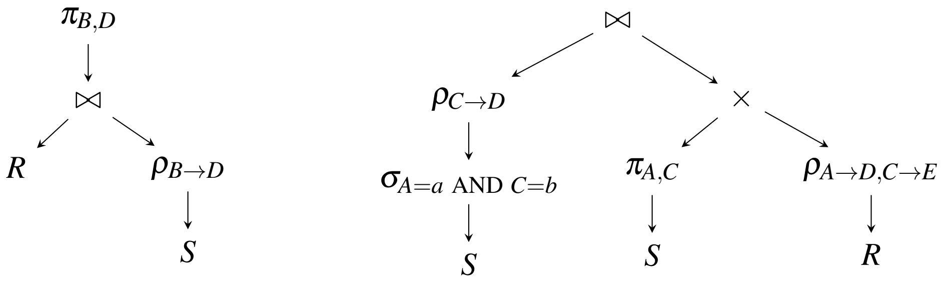
As you know from this tutorial and the videos, a DBMS takes SQL queries as input and translates these into *query plans* that are not yet optimised. Such plans can be represented as relational algebra expressions or as trees. This exercise is about the non-optimization part.

Exercise 1. Represent the following relational algebra expressions as query plans (in the form of a tree):

(a) $\pi_{B,D}(R \bowtie \rho_{B\to D}(S))$

(b) $\rho_{C\to D}(\sigma_{A=a \text{ AND } C=b}(S)) \bowtie (\pi_{A,C}(S) \times \rho_{A\to D,C\to E}(R))$

Solution: The query plans for (a) and (b) are (from left to right):



Let us now assume a database with the following two relations:

R

A	B	C
a	a	b
b	d	a

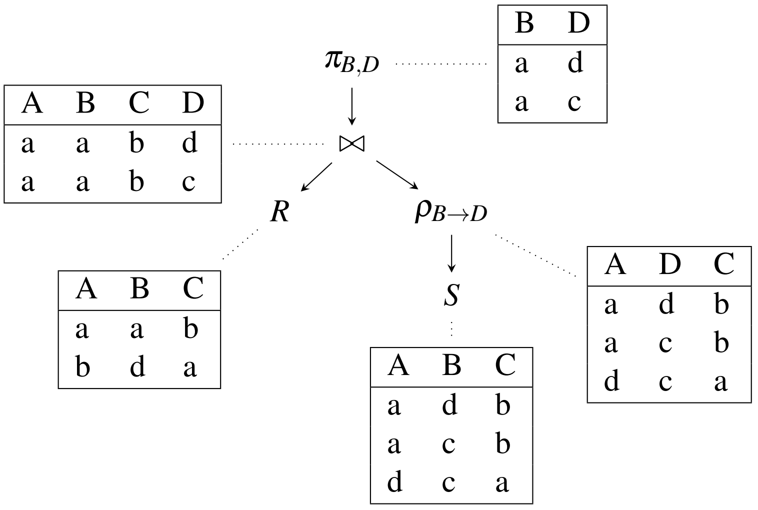
S

A	B	C
a	d	b
a	c	b
d	c	a

Exercise 2. Execute your query plans from Exercise 1 step by step on this database, as done in the examples and exercises in the video “Executing a query plan”.

Solution:

(a) Execution of the query plan from Exercise 1 (a):



B	D
a	d
a	c

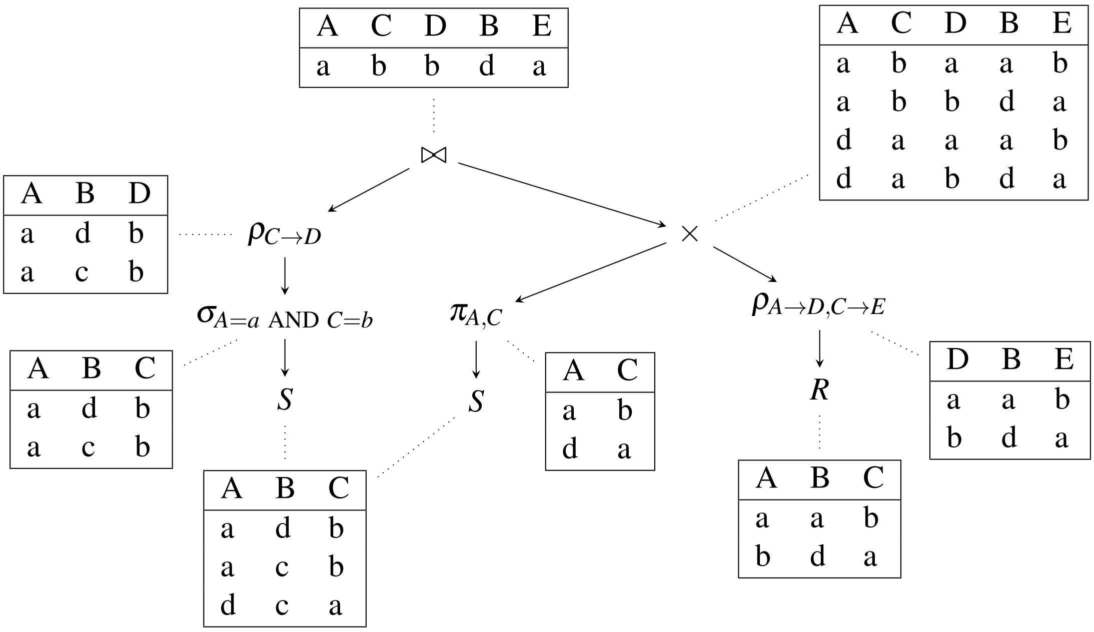
A	B	C	D
a	a	b	d
a	a	b	c

A	B	C
a	a	b
b	d	a

A	D	C
a	d	b
a	c	b
d	c	a

A	B	C
a	d	b
a	c	b
d	c	a

(b) Execution of the query plan from Exercise 1 (b):



A	C	D	B	E
a	b	b	d	a

A	C	D	B	E
a	b	a	a	b
a	b	b	d	a
d	a	a	a	b
d	a	b	d	a

A	B	D
a	d	b
a	c	b

A	B	C
a	d	b
a	c	b

A	B	C
a	d	b
a	c	b
d	c	a

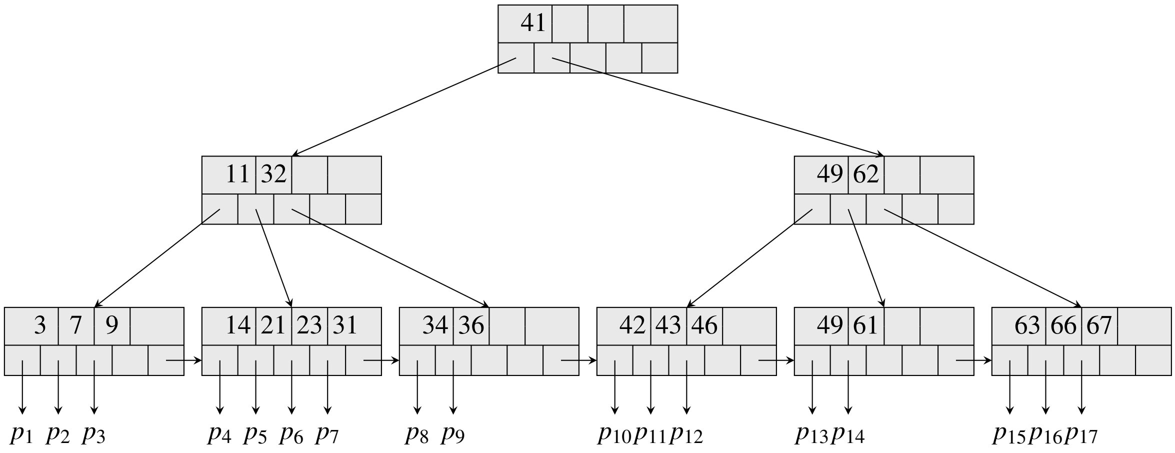
A	C
a	b
d	a

D	B	E
a	a	b
b	d	a

A	B	C
a	a	b
b	d	a

B+ Trees

Consider the following B+ tree:



Exercise 3. Use the method from the page “Idea, searching and inserting in B+-tree” to look up the pointers associated with the following values in the B+ tree. Try to follow the method step by step.

(a) 61

(b) 62

(c) All values greater than or equal to 44

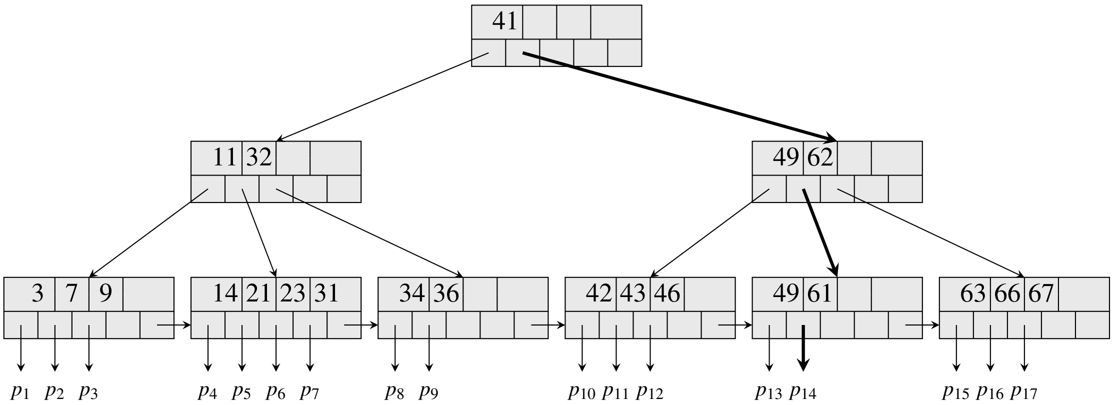
(d) All values in the range from 47 to 64 (i.e., all values x with $47 \leq x \leq 64$)

Solution:

(a) Procedure for looking up 61 in the B+ tree:

- Start at the root.
- Since $41 \leq 61$ and 41 is the last value at the root, proceed to the second child.
- Since $49 \leq 61 < 62$, proceed to the second child.
- Return the pointer associated with 61: p_{14}

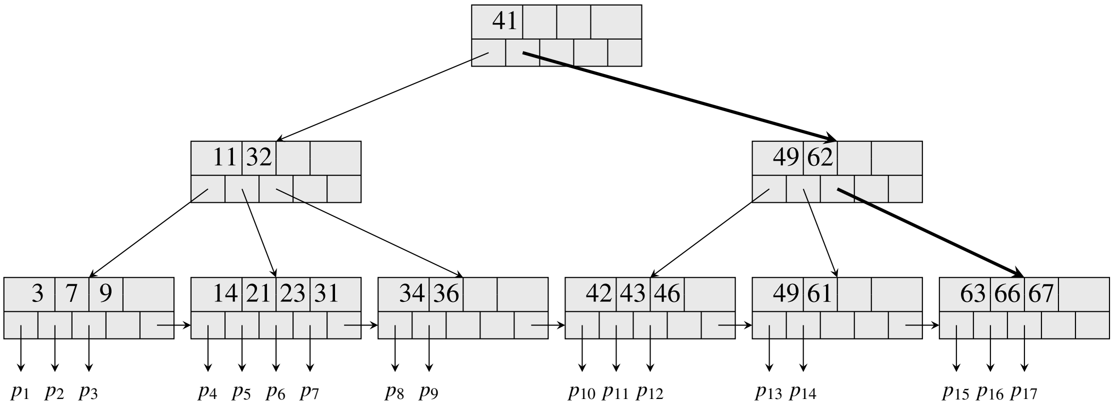
Here is a visual representation, with bold edges indicating the path taken through the tree to arrive at the pointer p_{14} associated with 61:



(b) Procedure for looking up 62 in the B+ tree:

- Start at the root.
- Since $41 \leq 62$ and 41 is the last value at the root, proceed to the second child of the root.
- Since $62 \leq 62$ and 62 is the last value at the current node, proceed to the third child of that node.
- The value 62 does not occur at the leaf, so return "not found".

Again, here is a visual representation, with bold edges indicating the path taken through the tree to arrive at the leaf for 62:

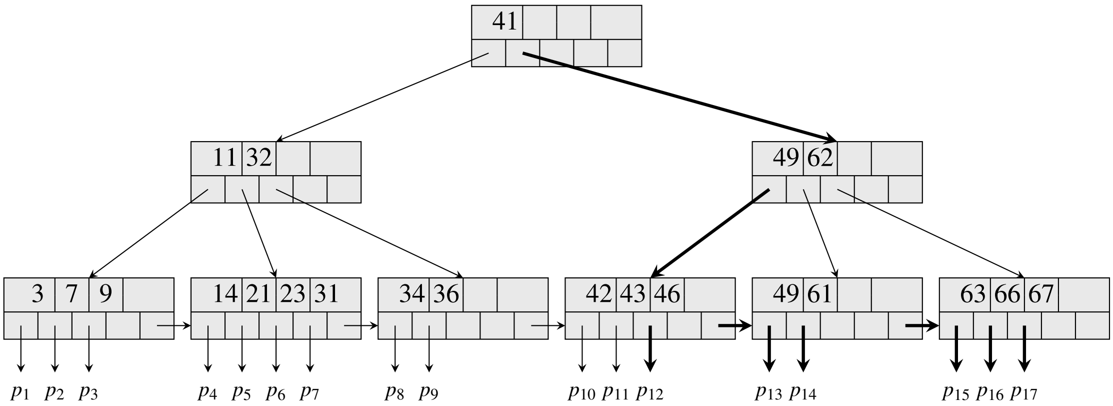


(c) Procedure for looking up all values greater than or equal to 44:

- Start at the root.
- Since $41 \leq 44$ and 41 is the last value at the root, proceed to the second child.
- Since $44 < 49$ and 49 is the first value at the current node, proceed to the first child.
- Return the pointers to all values of the leaf that are greater than or equal to 44. Thus, return p_{12}.

- Proceed to the leaf to the right and return the pointers to all values of that leaf. Thus, return p_{13} and p_{14}.
- Once again, proceed to the leaf to the right and return the pointers to all values of that leaf. Thus, return p_{15}, p_{16}, and p_{17}.

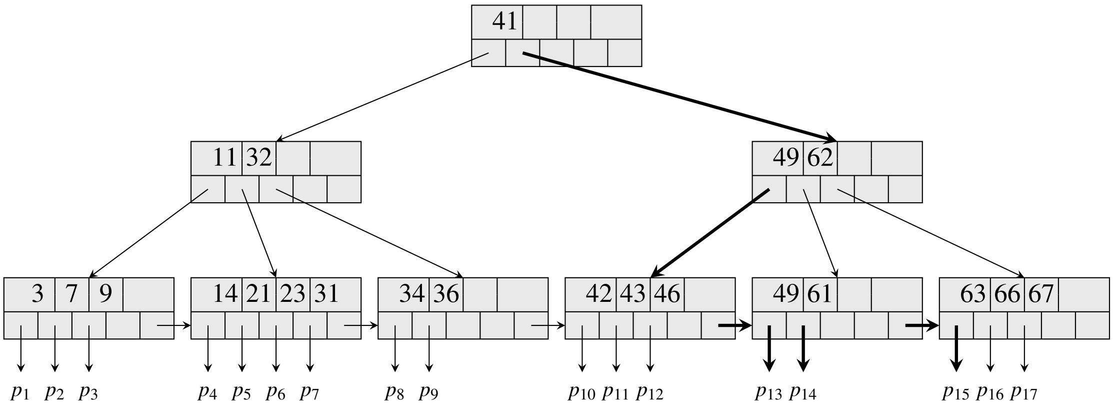
Here is a visual representation, with bold edges indicating all the edges followed to return the pointers associated with all the values that are greater than or equal to 44:



(d) Procedure for looking up all values in the range from 47 to 64:

- First find the leaf for 47. Start at the root.
- Since $41 < 47$ and 41 is the last value at the root, proceed to the second child.
- Since $47 \leq 49$ and 49 is the first value at the current node, proceed to the first child.
- Return the pointers to all values of the leaf in the range between 47 and 64. There are no such pointers, so proceed to the next leaf on the right.
- Return the pointers to all values of the current leaf that are less than or equal to 64. Thus, return p_{13} and p_{14}. Then, proceed to the next leaf on the right.
- Return the pointers to all values of the current leaf that are less than or equal to 64. Thus, return p_{15}.

Visual representation:



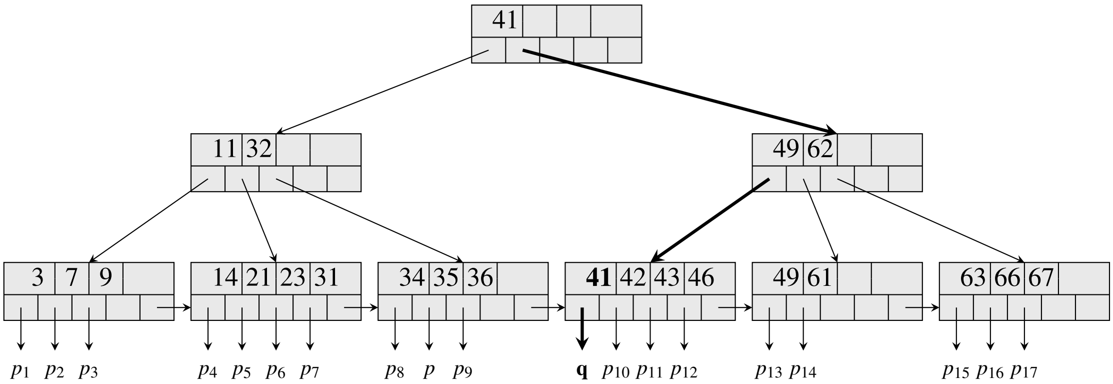
Exercise 4. Insert the following value/pointer pairs into the above B+ tree:

(a) Value: 41; pointer: *q*

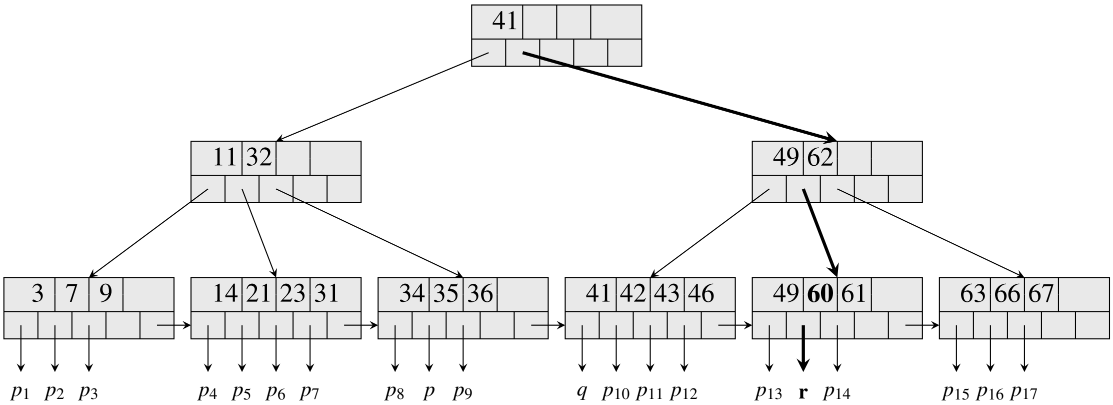
(b) Value: 60; pointer: *r*

Solution:

(a) We first navigate to the leaf associated with 41, which is the fourth leaf from left (the one containing 42, 43, and 46). At this leaf, we push all values and their pointers to the right and insert 41 and *q* into the first slot. This results in the following B+ tree (again, we mark the path to the leaf for 41 and to the pointer inserted):



(b) We navigate to the fifth leaf node from left (the one containing 49 and 61), and insert 60 and *r* between 49 and 61. The resulting B+ tree is:

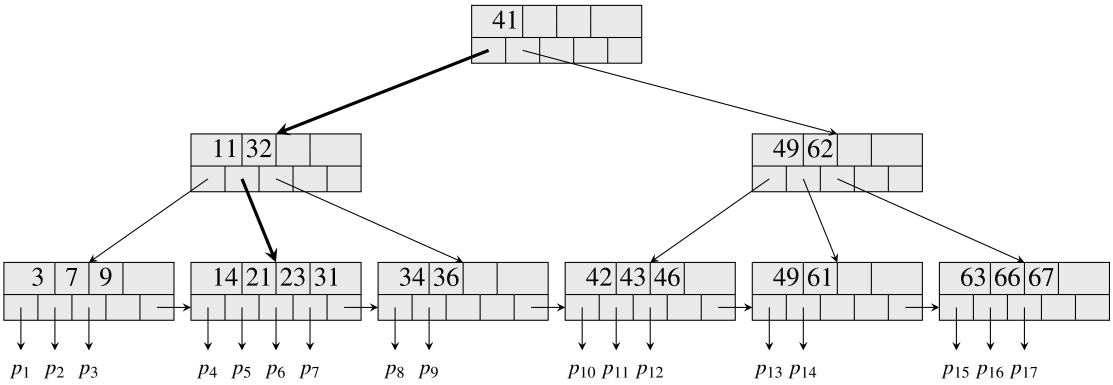


Exercise 5. Insert the following value/pointer pairs into the B+ tree above:

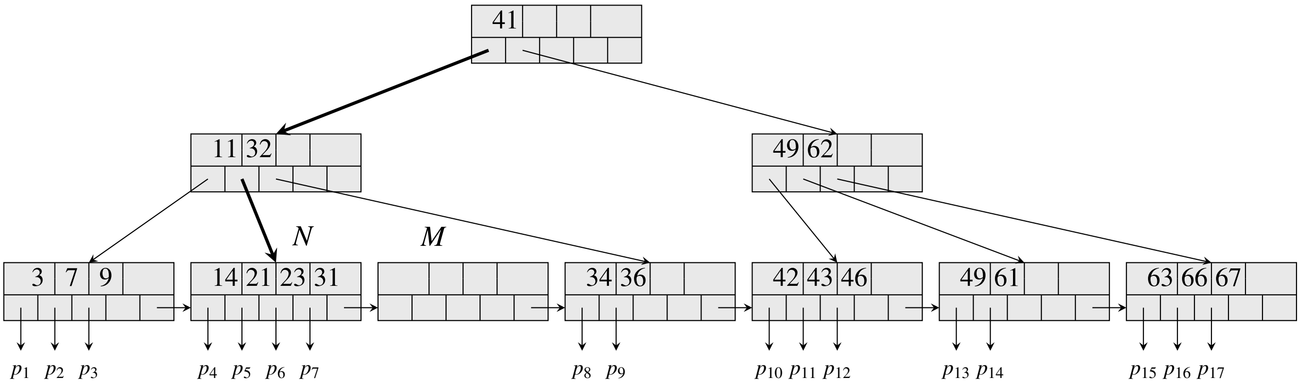
(a) Value 22; pointer *p*

(b) Value 17; pointer *p* (no solution provided)

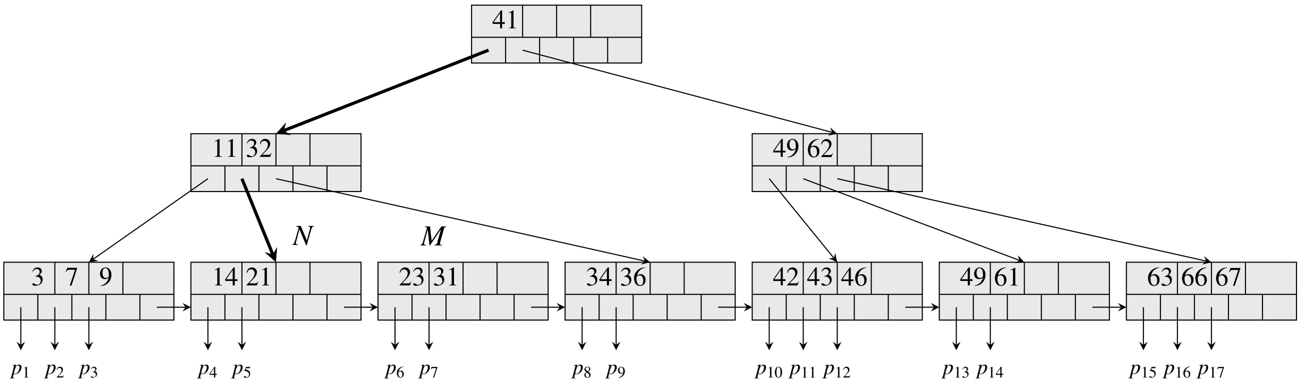
Solution for (a): Here is what happens when we insert 22 with pointer *p* into the B+ tree. We first navigate to the appropriate leaf:



Let us denote this leaf by N. Since N is full (i.e., it does not have any room to store 22), we have to split it. We first create a new leaf M and insert it between N and its right neighbour:

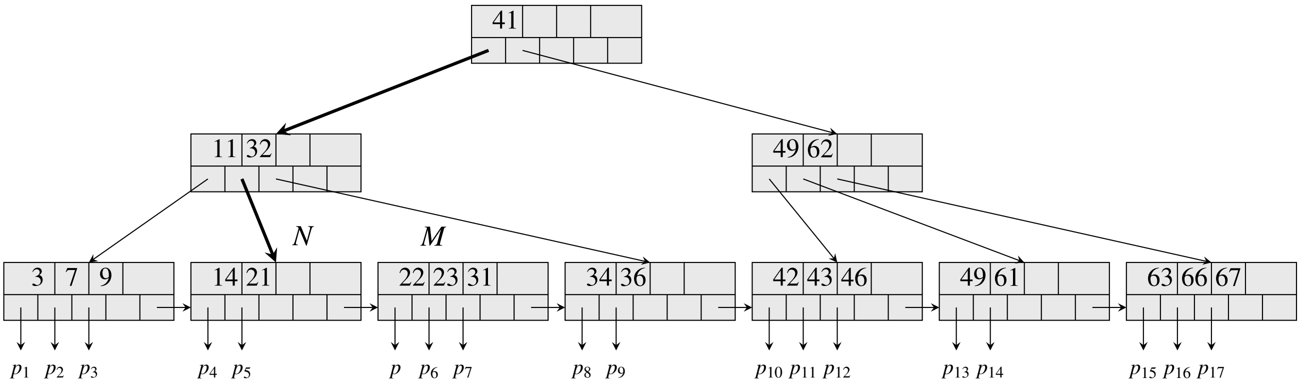


The next step is to move a sufficient number of values and pointers from N to M. In our case, we have $n = 4$, so in order to decide how many values and pointers we have to move we compare 22 with the value 21 at position $m = \lfloor \frac{n+1}{2} \rfloor = 2$. Since $22 > 21$, we keep 14, 21 and their pointers in N and only move 23, 31 and their pointers to M:

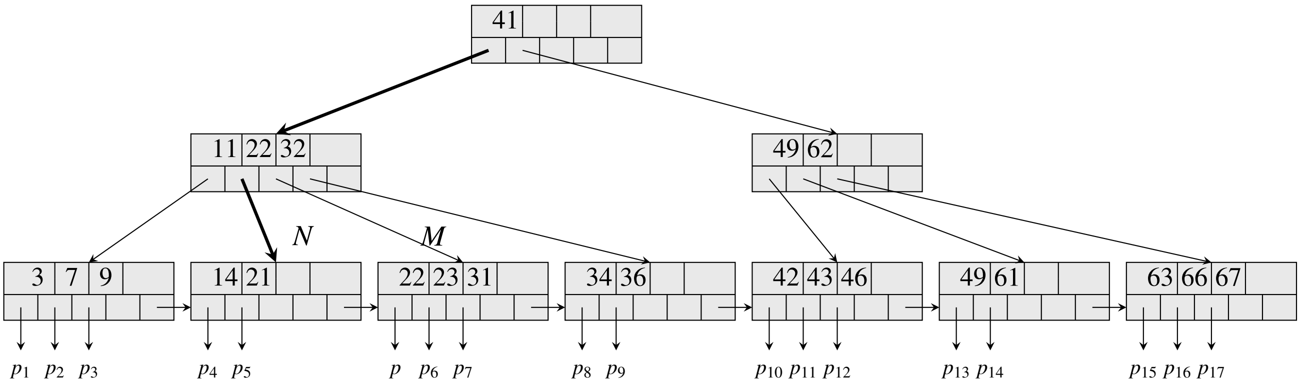


We now insert 22 and p into M:[1]

[1] This is arbitrary. We could have inserted 22 and p into N without destroying the property of B+ tree nodes, but here we follow the procedure given above.



It remains to add M as a child to the parent P of N. Note that M is the only leaf that can be reached from M, so the minimum value w stored in any leaf reachable from M is 22. In order to add M as a child to P we therefore insert the value 22 with a pointer to M into P. Since P is not full, we insert 22 and the pointer between 11 and 32. The final B+ tree is:



Implementing B+ Trees

Exercise 6 (no solutions provided)**.** Implement the B+ tree data structure in a programming language of your choice. Then try to experiment with your implementation by building a B+ tree for a large collection of values, by looking up values, by performing range searches, by inserting values, etc.