CS 186 Section 5: Tree Indexes & Relational Algebra

Vikram Sreekanti

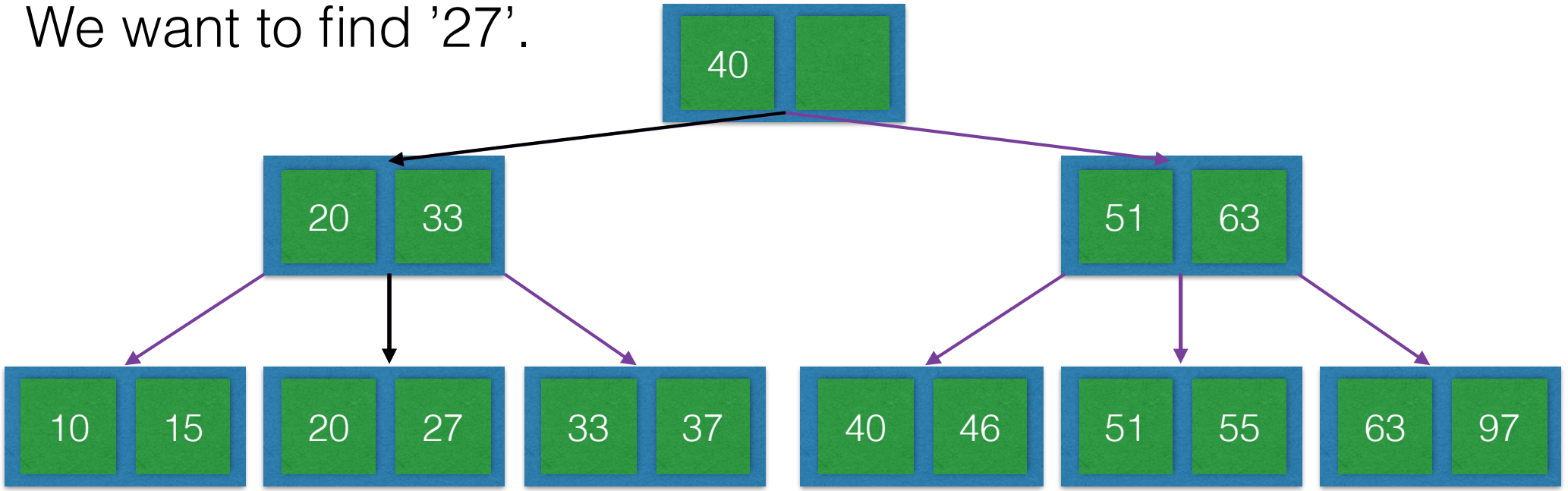
B+Trees

Why do we use tree-structures indexes?

- They support both equality and range queries.
- Different indexes on different search keys.

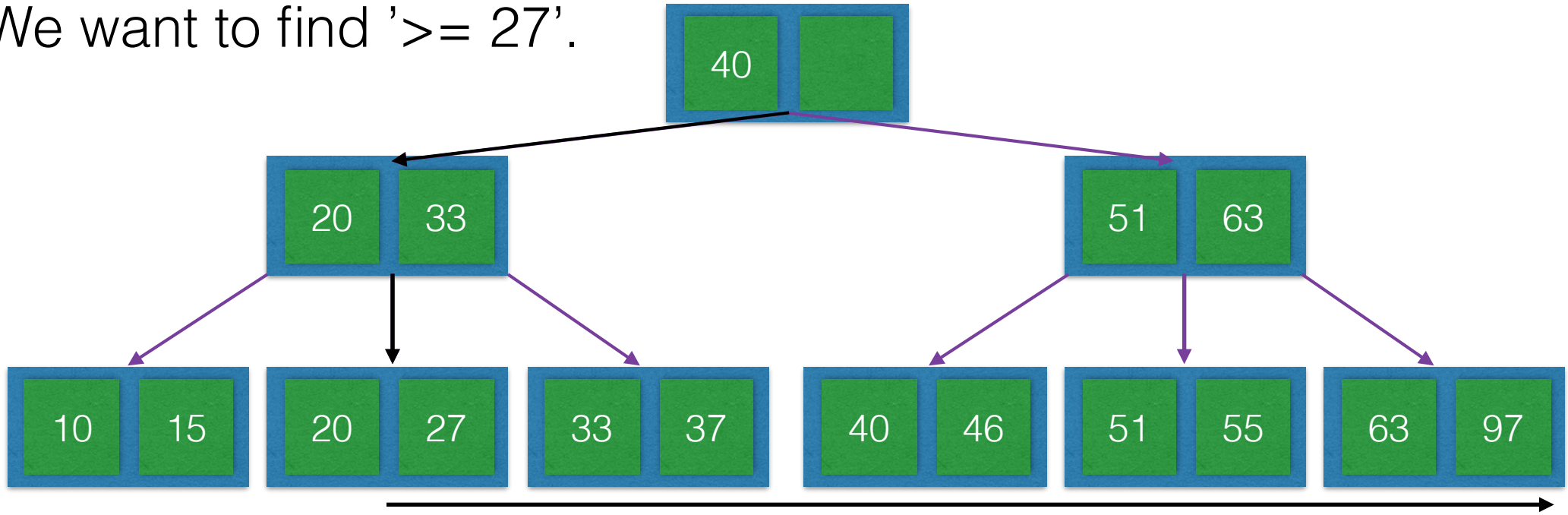
Tree Indexes: Equality Lookups

We want to find ’27’.



Tree Indexes: Range Lookups

We want to find '>= 27'.



Tree Indexes: Two Kinds

ISAM: Indexed Sequential Access Method

Old. No one uses these!

"Static" data structure.

What does this mean?

B+Tree: (note: B-Tree vs. B+Tree)

"Dynamic" data structure.

Adjusts gracefully under inserts and deletes.

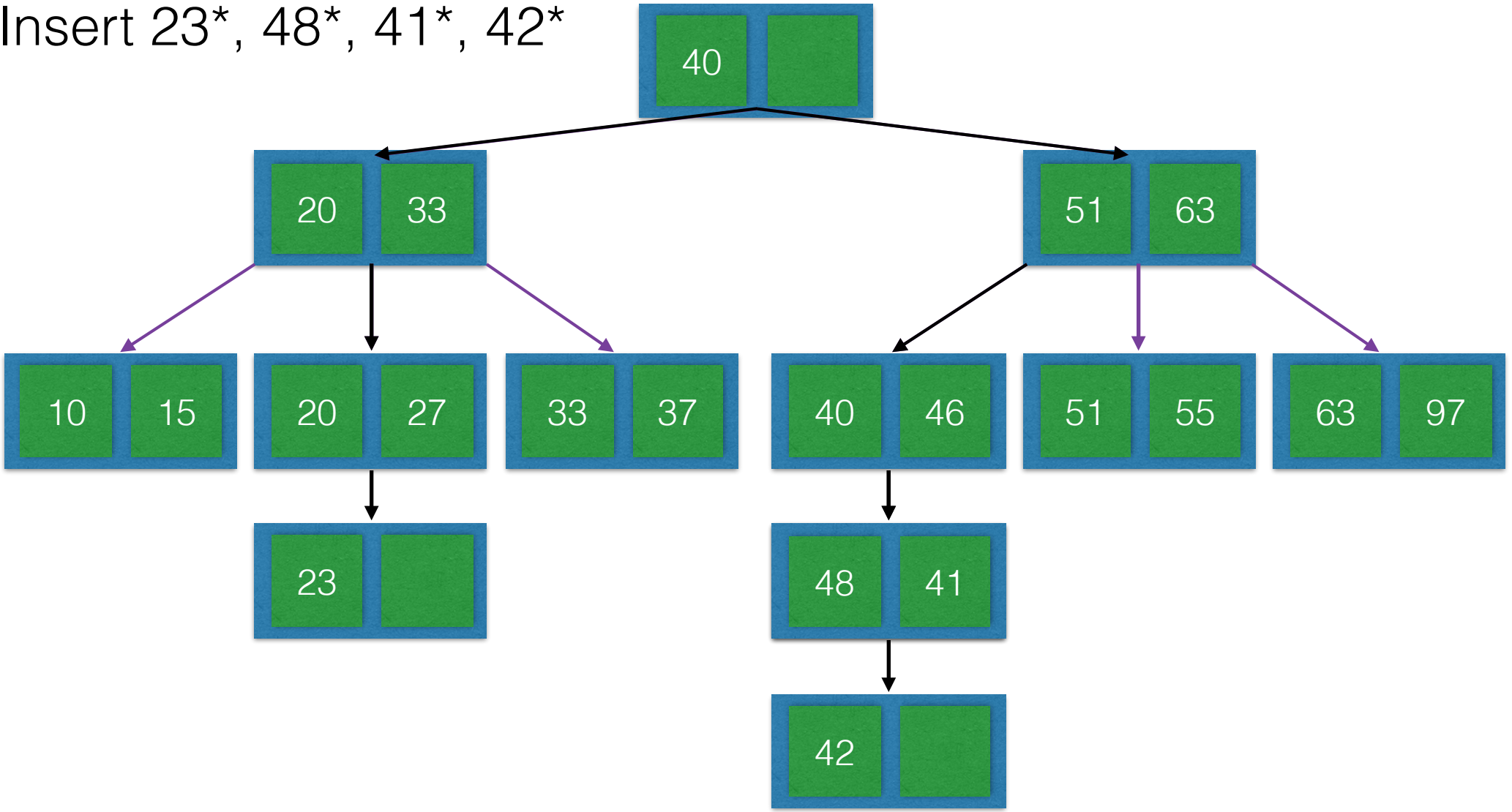
2-3-4 Trees from 61b, anyone?

B+Tree Terminology

- Every B+Tree has a predefined order *d*.
  - Every node must contain between *d* and *2d* entries. (This will become important when we get to insertion and deletion).
- Height: Length of the path from root to leaf.
- Fanout: The maximum number of pointers out of the node (= *2d*).

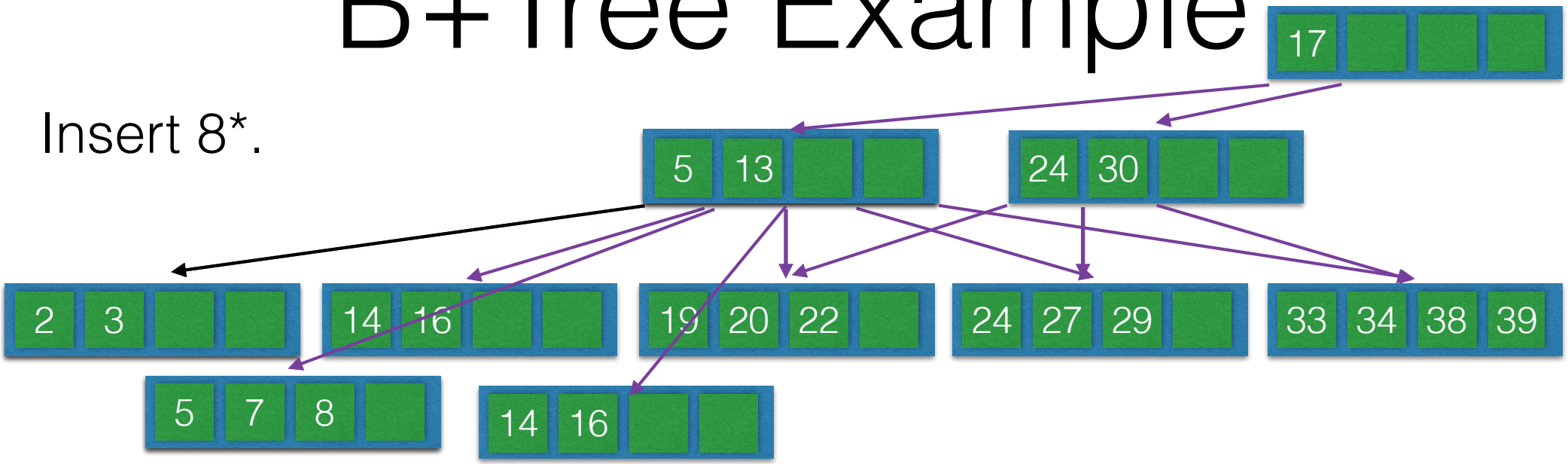
ISAM Example

Insert 23*, 48*, 41*, 42*



B+Tree Example

Insert 8*.



Now what?

Now we need a new root!

The leaf is full!

Split it.

Key: You must *push* up the middle key!

Split it!

B+Tree Insertion Algorithm

- Find correct leaf *L*.
- Put data entry into *L*.
  - If there is space in *L*, you're done!
  - Otherwise, split *L*.
    - Redistribute entries evenly.
    - Copy up the middle key.
    - Insert pointer to *L2* into parent of *L*.
  - Redistribution can happen recursively.
    - If splitting an index entry, then *push* up middle key.

B+Tree Deletion Algorithm

- Find correct leaf *L*.
- Delete entry from *L*.
  - If *L* is at least half full, then you're done.
  - Otherwise:
    - You can try to redistribute from neighbors.
      - Redistribution is done through rotation.
    - If you cannot redistribute, then you must merge.

Note: In practice, many B+Trees do not worry about this. They let pages empty slowly, and when they are empty, remove them altogether.

B+Tree Question

You have decided to develop a new deals website which pushes nearby deals to user's mobile phones based on their age group. Assume that there are 2 million users in your database, that each user entry is 2kB in size, and that you are mainly performing range queries based on a user's age. Assume the page size is 16kB.

You have decided to create a clustered B+Tree on the age field. The tree has a fanout of 200 and a height of 3. Assume that you are on average returning 50,000 users per query. On average, how many I/O's are performed by such a query?

3 + (50000 * 2/16) = 6,253 IOs

B+Tree Question

You have decided to develop a new deals website which pushes nearby deals to user's mobile phones based on their age group. Assume that there are 2 million users in your database, that each user entry is 2kB in size, and that you are mainly performing range queries based on a user's age. Assume the page size is 16kB.

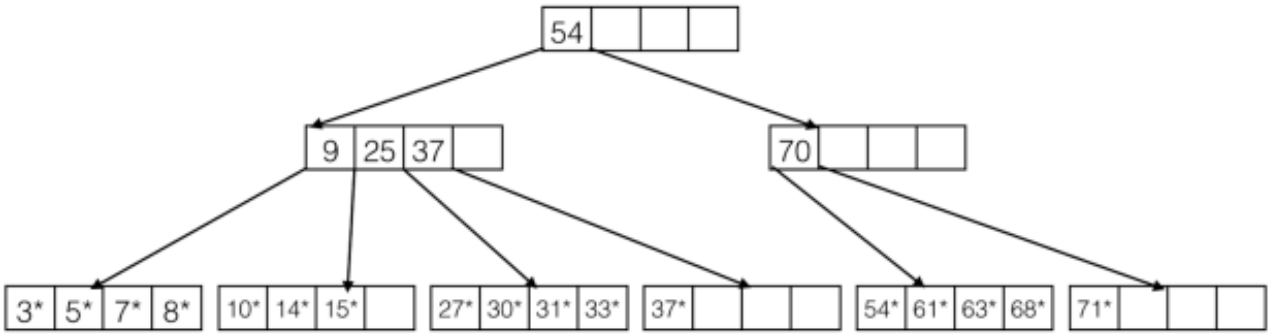
Assume your B+ tree is unclustered. In the worst case, how many I/O's do you need now? Assume that you are still returning 50,000 users per query on average, and that an index entry is 3 times smaller than a user entry.

- 3 I/Os to descend to leaf
- ceil(5000 * .67 / 16) = 2084 IOs to read leaf index pages
- 50,000 IOs to read unordered data pages

= 52,087 IOs

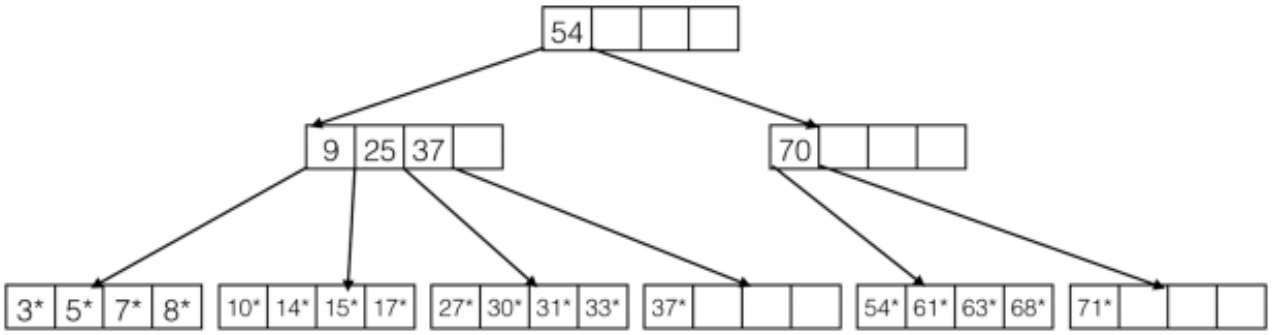
B+Tree Exercise

Insert 17, 18, 29



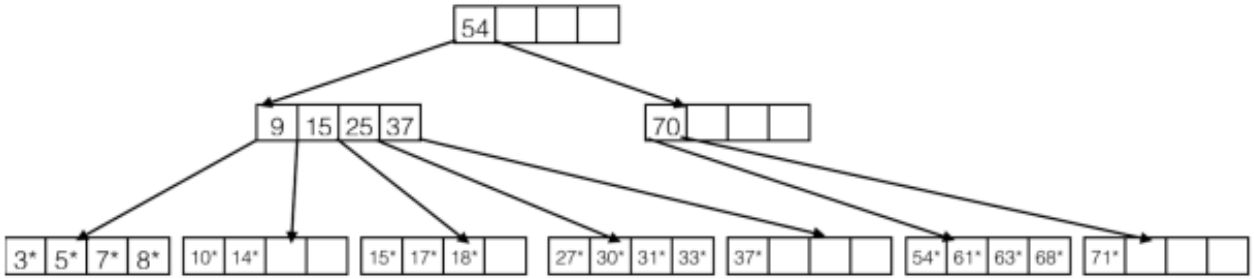
B+Tree Exercise

Insert 17



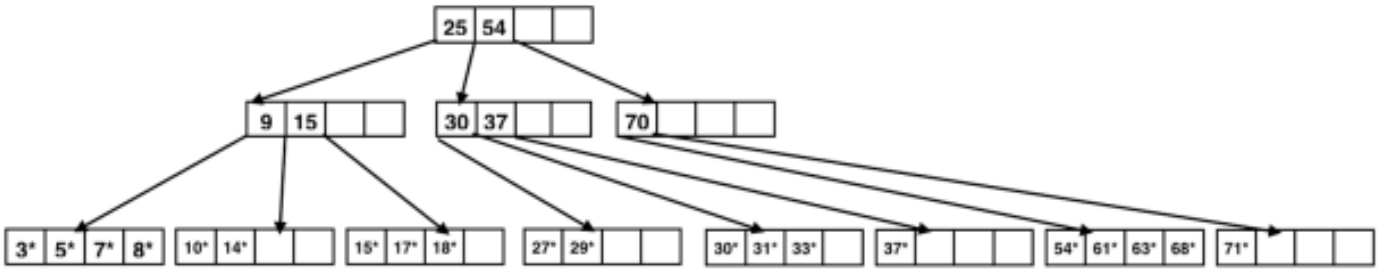
B+Tree Exercise

Insert 18



B+Tree Exercise

Insert 29



Relational Algebra

Basics of Relational Algebra

1. Relational operators are transformations on relations. That is, an operator takes in a relation and outputs a relation (which might have a different schema).
2. Pure relational algebra does not have duplicate — it is based on set theory.
   a. SQL allows duplicates for (most) operators — it is based on multiset theory.
   b. “Set operators” like UNION, INTERSECT remove duplicates by default in most SQL dialects.

Relational Operators

1. Selection (σ): Select a subset of rows.
2. Projection (π): Select a subset of columns.
3. Cross-Product (x): Combine two relations.
4. Set difference (-): Tuples in *r1* but not in *r2*.
5. Union (U): Tuples in *r1* or *r2*.

Relational Algebra Exercises

```
Songs (song_id, song_name, album_id, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id, album_name, artist_id, year_released, genre)
```

Find the name of the artists who have albums with a genre of either ‘pop’ or ‘rock’.

π Artists.artist_name (Artists $\bowtie$ (σ Albums.genre = ‘pop’ $\vee$ Albums.color = ‘rock’ Albums))

Relational Algebra Exercises

```
Songs (song_id, song_name, album_id, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id, album_name, artist_id, year_released, genre)
```

Find the name of the artists who have albums of genre ‘pop’ and ‘rock’.

$$\pi_{\text{Artists.artist\_name}}((\sigma_{\text{Albums.genre = 'pop'}}\ \text{Albums}) \bowtie \text{Artists}) \cap \pi_{\text{Artists.artist\_name}}((\sigma_{\text{Albums.genre = 'rock'}}\ \text{Albums}) \bowtie \text{Artists}$$

Relational Algebra Exercises

```
Songs (song_id, song_name, album_id, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id, album_name, artist_id, year_released, genre)
```

Find the id of the artists who have albums of genre ‘pop’ or have spent over 10 weeks in the top 40.

$$\pi_{\text{Artists.artist\_id}} (\text{Artists} \bowtie (\sigma_{\text{Albums.genre = 'pop'}} \text{Albums})) \cup \pi_{\text{Albums.artist\_id}} (\text{Albums} \bowtie (\sigma_{\text{Songs.weeks\_in\_top\_40 > 10}} \text{Songs}))$$

Relational Algebra Exercises

```
Songs (song_id, song_name, album_id, weeks_in_top_40)
Artists(artist_id, artist_name, first_year_active)
Albums (album_id, album_name, artist_id, year_released, genre)
```

Find the names of all artists who do not have any albums.

π Artists.artist_name (Artists $\bowtie$ ((π Artists.artist_id Artists) (π Albums.artist_id Albums)))