KELOMPOK 6

B -TREE

KELOMPOK 6

Anggota

- Danar Bagus Rasendriya (5027231055)
- Mey Rosalina (5027241004)
- Clarissa Aydin Rahmazea (5027241014)
- Ahmad Idza Anaafin (5027241017)
- Muhammad Fatihul Qolbi Ash Shiddiqi (5027241023)
- Zein Muhammad Hasan (5027241035)

KELOMPOK 6

Problem Statement

LET’S TAKE A LOOK

KELOMPOK 6

Problem Statement

Pengendalian kehadiran dosen dan penjadwalan kuliah pengganti masih menjadi tantangan di banyak perguruan tinggi. Ketidaktertiban dalam pengelolaan jadwal perkuliahan pengganti seringkali menyebabkan bentrok antar kelas dan ketidaksesuaian alokasi waktu serta ruangan. Hal ini berdampak pada efektivitas proses pembelajaran dan dapat menghambat pencapaian kehadiran minimal 80% yang diwajibkan oleh regulasi pendidikan tinggi. Sistem informasi akademik yang belum optimal dalam mengelola data jadwal, dosen, mahasiswa, dan ruangan secara efisien memperburuk permasalahan ini. Oleh karena itu, dibutuhkan sebuah aplikasi yang mampu mengelola penjadwalan kuliah pengganti secara cerdas dan efisien dengan mengimplementasikan struktur data seperti algoritma B-Tree untuk meningkatkan akurasi pencarian dan mengurangi kemungkinan bentrokan jadwal.

KELOMPOK 6

Pengertian, Struktur dan Algoritma B Tree

LET'S TAKE A LOOK

KELOMPOK 6

B-Tree (Balanced Tree)

Definisi:
B-Tree adalah struktur data pohon ber-orde m (m-ary) yang digunakan untuk menyimpan data dalam bentuk terurut dan memungkinkan pencarian, penyisipan, dan penghapusan data secara efisien dalam kompleksitas logaritmik.

Karakteristik B-Tree Orde m:

1. Setiap node memiliki maksimum m anak.
2. Setiap node internal (bukan daun) memiliki minimal ⌈m/2⌉ anak.
3. Setiap node (kecuali akar) memiliki minimal ⌈m/2⌉ − 1 kunci dan maksimal m − 1 kunci.
4. Semua daun berada di level yang sama (terseimbang).
5. Kunci dalam node terurut naik dan memisahkan nilai subpohon.

KELOMPOK 6

STRUKTUR B-TREE

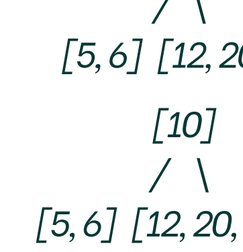
B-Tree memiliki orde atau derajat (T atau m). Ini menentukan banyaknya kunci dan anak maksimum dalam satu simpul (node). Misalnya, B-Tree orde 4 berarti satu node bisa punya maksimal 3 kunci dan 4 anak.

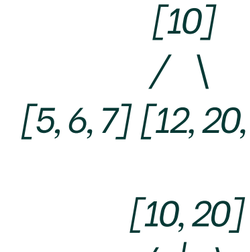
Setiap node dalam B-Tree memiliki:

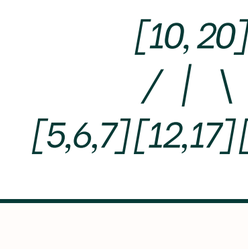
1. *n: Jumlah kunci saat ini disimpan dalam node.*
2. *keys[]: Array dari kunci yang selalu terurut naik.*
3. *children[]: Array pointer ke subtree/anak-anak. Jumlahnya bisa hingga n+1.*
4. *leaf: Boolean penanda apakah node ini daun atau internal.*

ILUSTRASIKAN B-TREE ORDE 3

(artinya satu node bisa memiliki maksimum 2 kunci dan 3 anak).







KELOMPOK 6

Algoritma

(INSERTION)

Langkah-langkah:

Fungsi utama: insert(k)

1. Jika root penuh:
   - Buat root baru s
   - Pecah (split) root lama
   - Sisipkan k ke anak yang sesuai
2. Jika root tidak penuh:
   - Sisipkan k ke bawah (subtree) sesuai lokasi yang benar

(DELETION)

Situasi:

- Jika kunci ada di daun → hapus langsung
- Jika kunci ada di node internal:
  - Jika anak kiri memiliki ≥ t kunci → ganti dengan pendahulu
  - Jika anak kanan ≥ t kunci → ganti dengan pengganti
  - Jika kedua anak punya t-1 kunci → gabungkan anak-anak, lalu hapus turun

(SEARCH)

Langkah:

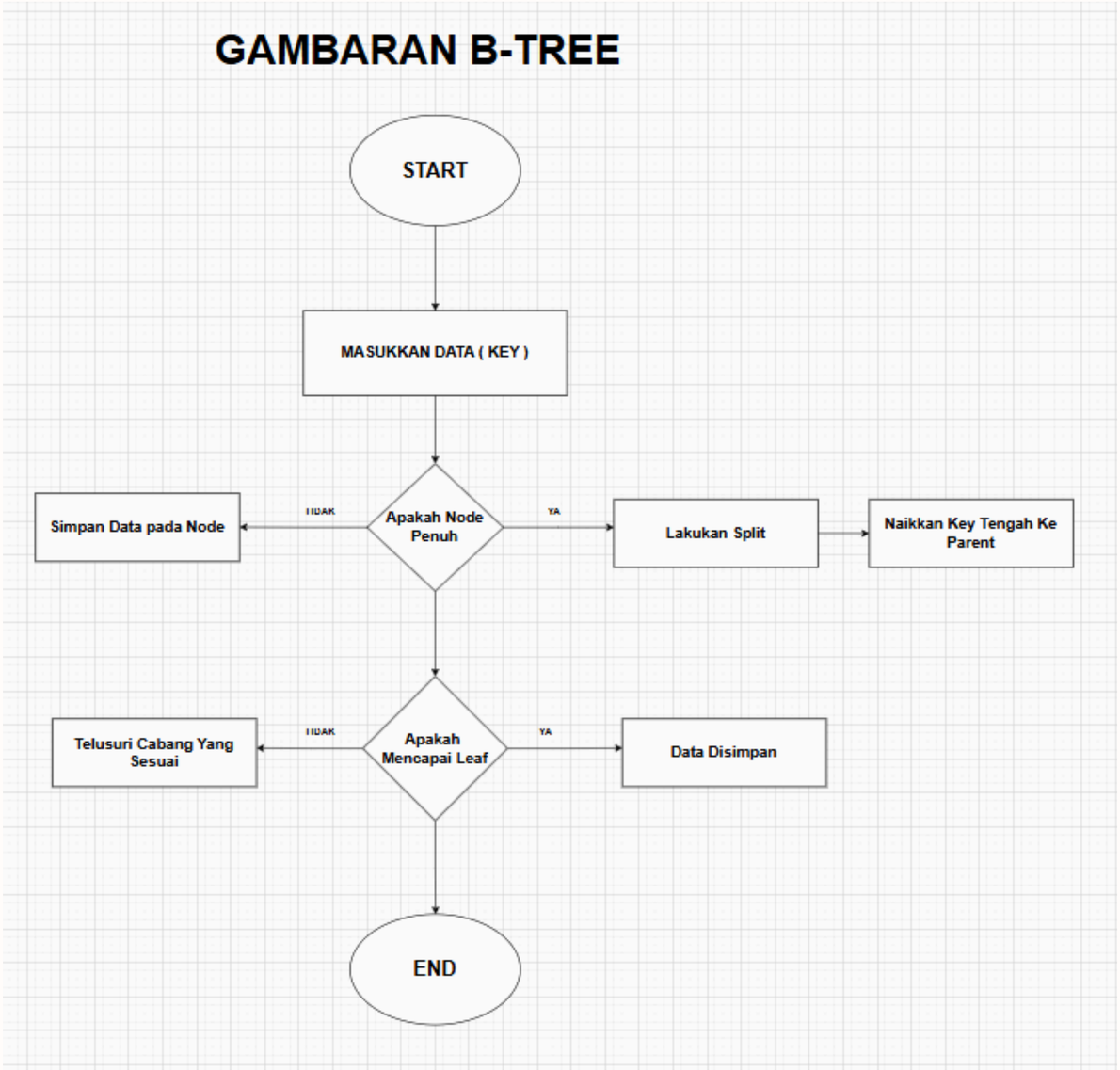
1. Mulai dari node saat ini
2. Temukan indeks i:
   - Jika keys[i] == k → ketemu
   - Jika leaf == true → tidak ada
   - Jika k < keys[i] → cari di children[i]
   - Jika k > semua keys → cari di anak terakhir children[n]

KELOMPOK 6

Diagram dan Visualisasi

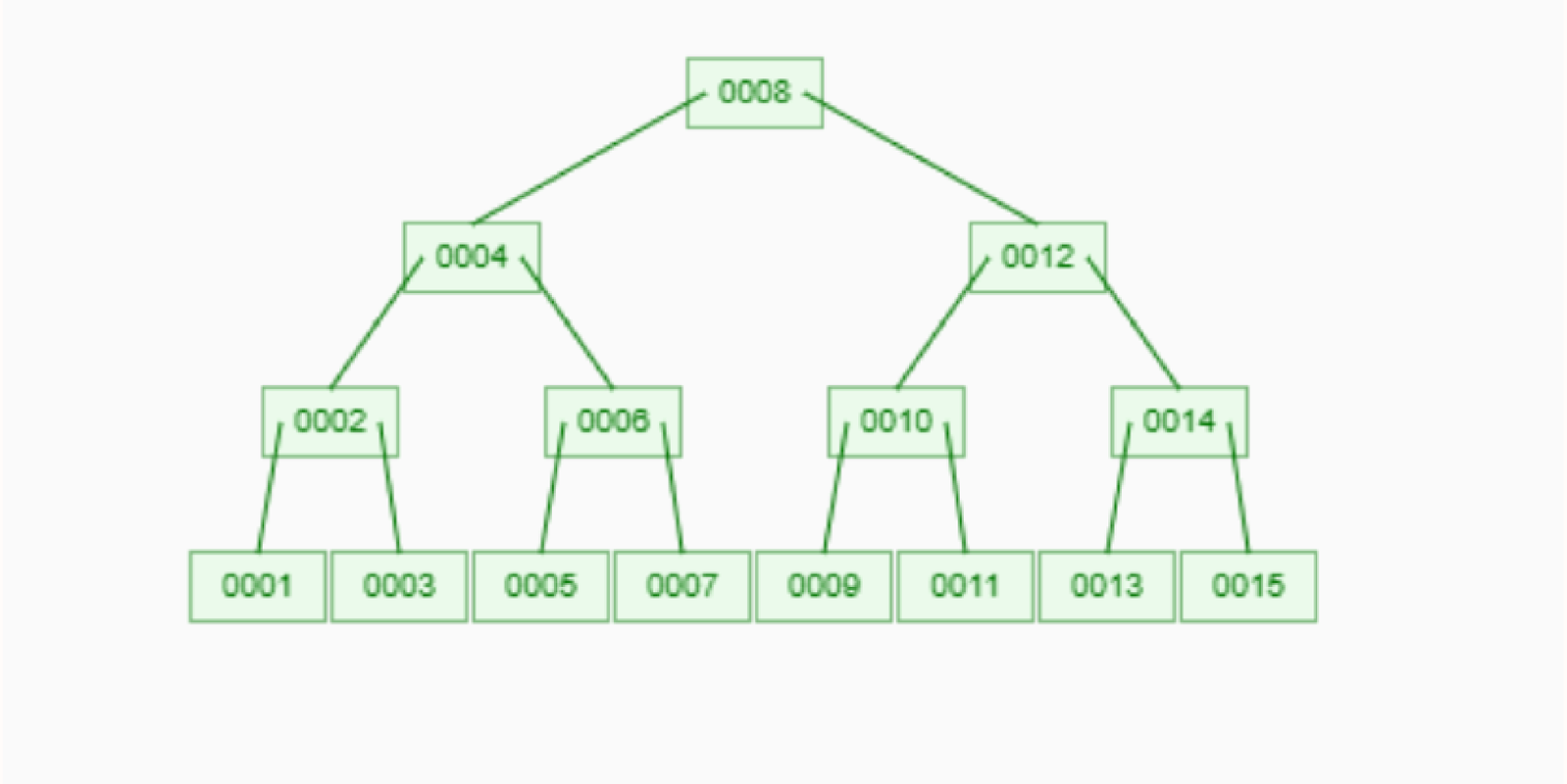
LET’S TAKE A LOOK

KELOMPOK 6

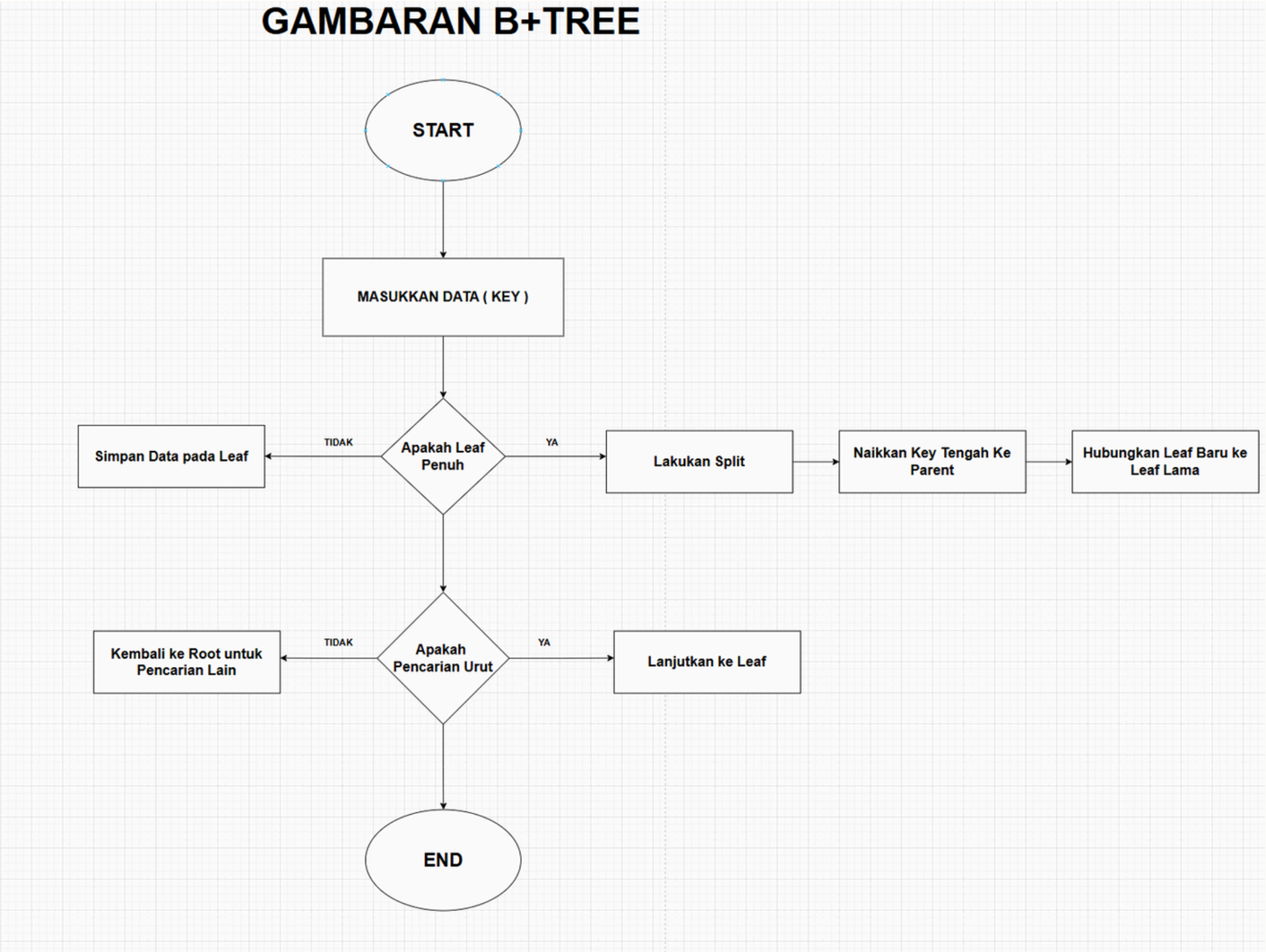

GAMBARAN B-TREE
START
MASUKKAN DATA (KEY)
Apakah Node Penuh
TIDAK
Simpan Data pada Node
YA
Lakukan Split
Naikkan Key Tengah Ke Parent
Apakah Mencapai Leaf
TIDAK
Telusuri Cabang Yang Sesuai
YA
Data Disimpan
END

KELOMPOK 6

MAX DEGREE : 3

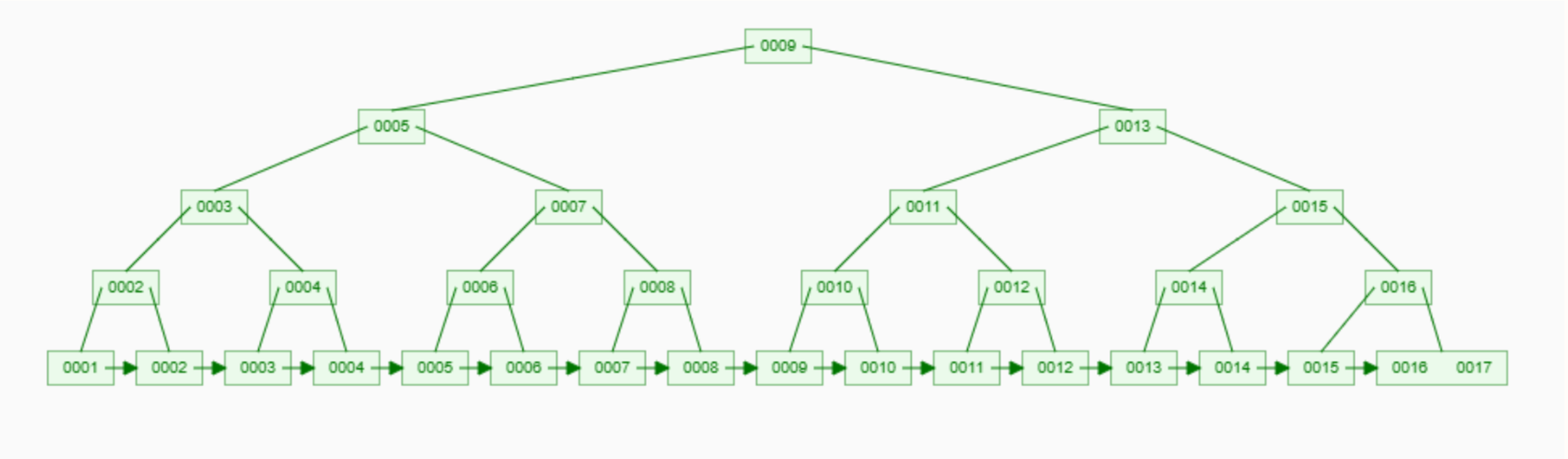


KELOMPOK 6


GAMBARAN B+TREE
START
MASUKKAN DATA (KEY)
Apakah Leaf Penuh
TIDAK
Simpan Data pada Leaf
YA
Lakukan Split
Naikkan Key Tengah Ke Parent
Hubungkan Leaf Baru ke Leaf Lama
Apakah Pencarian Urut
TIDAK
Kembali ke Root untuk Pencarian Lain
YA
Lanjutkan ke Leaf
END

KELOMPOK 6

MAX DEGREE : 3



KELOMPOK 6

Aplikasi dan Implementasi

LET’S TAKE A LOOK

KELOMPOK 6

IN VS CODE

KELOMPOK 6

Keunggulan dan Kekurangan

LET’S TAKE A LOOK

KELOMPOK 6

Kelebihan B-Tree

EFISIENSI PENCARIAN (SEARCH EFFICIENCY)

B-Tree dirancang untuk pencarian data yang cepat dengan kompleksitas waktu terburuk yang terjamin. B-Tree menjamin waktu pencarian O(log n)

OPERASI YANG KONSISTEN (BALANCED TREE)

B-tree selalu dalam keadaan seimbang (balanced), memastikan waktu pencarian, penyisipan, dan penghapusan tetap dalam kompleksitas logaritmik (O(log n)).

OPTIMAL UNTUK PENYIMPANAN DISK

B-Tree dirancang untuk sistem penyimpanan eksternal (seperti hard disk) yang membaca data per blok, bukan per byte (di B-Tree satu blok disk bisa memuat banyak data).

FAKTOR CABANG TINGGI

Dengan faktor cabang yang besar (biasanya >1000), B-Tree dapat menyimpan data dalam jumlah besar dengan tinggi pohon yang rendah, sehingga mengurangi waktu akses.

FLEKSIBEL TERHADAP UPDATE:

Operasi insert dan delete dapat dilakukan tanpa perlu melakukan rebalancing besar-besaran seperti pada AVL tree.

COCOK UNTUK DATA SKALA BESAR

Dapat menangani sejumlah besar data dengan efisien, terutama pada struktur disk-based karena desainnya yang mempertimbangkan blok memori eksternal.

KELOMPOK 6

Kekurangan B-Tree

KOMPLEKSITAS IMPLEMENTASI

Implementasi B-tree lebih kompleks dibandingkan struktur data seperti binary search tree atau AVL tree, terutama dalam hal split, merge, dan redistribusi node.

OVERHEAD MEMORI

Karena menyimpan pointer ke anak-anak dalam setiap node serta menyimpan banyak kunci dalam satu node, B-tree membutuhkan lebih banyak memori per node.

KURANG EFISIEN UNTUK DATA KECIL/IN-MEMORY:

Jika semua data dapat dimuat dalam memori utama (RAM), maka penggunaan B-tree mungkin kurang efisien dibandingkan struktur pohon lainnya yang lebih sederhana.

MEMERLUKAN NORMALISASI DATA

Seperti disebutkan dalam jurnal, implementasi B-Tree memerlukan normalisasi data terlebih dahulu untuk menghindari anomali (insertion, deletion, update anomaly), yang menambah tahapan persiapan.

KELOMPOK 6

Kelebihan B+Tree

EFISIENSI PENCARIAN YANG UNGGUL

B+Tree secara khusus dioptimalkan untuk operasi pencarian dengan kompleksitas waktu O(log n), bahkan dalam skenario terburuk.

STRUKTUR YANG SELALU SEIMBANG

Seluruh leaf node berada pada level yang sama, menjamin waktu akses yang konsisten untuk semua operasi.

OPTIMALISASI PENYIMPANAN DISK

Dirancang untuk bekerja dengan blok disk, dimana satu node biasanya sesuai dengan satu blok penyimpanan (4KB-16KB).

FLEKSIBEL TERHADAP UPDATE:

Insert/Delete dilakukan melalui algoritma split dan merge yang terdefinisi dengan baik.
Contoh saat menambahkan jadwal baru, sistem akan:

a. Cari posisi leaf node yang sesuai
b. Lakukan insert
c. Split node jika penuh
d. Update parent nodes

MINIMASI BENTROK PENJADWALAN

Struktur yang teratur memungkinkan deteksi konflik jadwal yang cepat.

KELOMPOK 6

Kekurangan B+Tree

KOMPLEKSITAS IMPLEMENTASI

- Setiap leaf node menyimpan pointer ke sibling → memakan ruang ekstra (5–10% tergantung implementasi).

OVERHEAD MEMORI

- Setiap node menyimpan:
  - Array of keys
  - Array of pointers
  - Metadata (status leaf, jumlah key, dll)
- Untuk mencari 1 record, B+ Tree harus mencapai leaf node, bahkan jika kunci sudah ditemukan di node internal.

KOMPLEKSITAS IMPLEMENTASI

- Logika split/merge lebih rumit karena:
  a. Node internal dan leaf ditangani berbeda.
  b. Pointer sibling harus selalu di-update.

TIDAK OPTIMAL UNTUK SISTEM IN-MEMORY

- Kelebihan B+ Tree (optimasi akses disk) jadi tidak relevan jika data disimpan di RAM.

KELOMPOK 6

Aspek	B-Tree	B+Tree
Struktur Data	Data disimpan di semua node	Data hanya di leaf nodes
Range Query	Tidak efisien (O(k log n))	Sangat efisien (O(log n + k))
Pointer Sibling	Tidak ada	Leaf nodes terhubung via pointer
Tinggi Pohon	Lebih tinggi untuk orde sama	Lebih rendah
Overhead Penyimpanan	Lebih rendah (tidak ada pointer sibling)	Lebih tinggi (20-30% ekstra)
Operasi Insert/Delete	Lebih sering rebalancing	Lebih stabil
Kesesuaian Storage	Cocok untuk disk dan memory	Optimal untuk disk-based system
Kompleksitas Kode	Relatif sederhana	Lebih kompleks
Contoh Aplikasi	Sistem caching, filesystem	Database indexing, sistem akademik

KELOMPOK 6

Tree dasar vs Tree Modifikasi

LET’S TAKE A LOOK

KELOMPOK 6

Perbandingan B-Tree dan B+Tree

Aspek	B-Tree	B+Tree
Struktur Node	Menyimpan kunci dan data di setiap node, baik internal maupun leaf.	Menyimpan semua data hanya di leaf node; node internal hanya menyimpan kunci.
Traversal	Pencarian dapat berhenti di node internal jika kunci ditemukan.	Pencarian selalu mencapai leaf node karena data hanya ada di sana.
Efisiensi Pencarian	Kurang efisien untuk pencarian berurutan karena tidak ada link antar leaf node.	Lebih efisien untuk pencarian berurutan karena leaf node terhubung sebagai linked list.
Kinerja Pengambilan	Waktu pengambilan dokumen lebih lambat dibandingkan B+ Tree.	Waktu pengambilan dokumen lebih cepat, terutama untuk range query.
Aplikasi	Cocok untuk sistem dengan kebutuhan pencarian acak dan struktur data kecil.	Cocok untuk sistem dengan kebutuhan pencarian berurutan dan data besar, seperti basis data dan sistem file.

PERFORMANCE EVALUATION OF INVERTED FILES, B-TREE AND B+ TREE INDEXING ALGORITHM ON MALAY TEXT

KELOMPOK 6

Analisis
Kompleksitas Struktur Tree

LET’S TAKE A LOOK

B- TREE

Struktur:

- B-Tree adalah balanced tree: tinggi pohonnya logaritmik

- Setiap node bisa punya m - 1 kunci dan maksimal m anak

- Kunci disimpan di semua node (internal dan daun). Karena setiap node bisa menyimpan banyak kunci, tingginya rendah

Kompleksitas Waktu:

• Search: $O(\log_m n)$

Menelusuri dari root ke daun. Dalam setiap node, pencarian kunci O(log m) (jika pakai binary search) atau O(m) (jika linear). Total: $O(\log_m n \times m) \approx O(\log n)$

• Insert: $O(\log_m n)$ (dengan kemungkinan split)

Sama seperti search, lalu bisa terjadi split. Tetap O(log n)

• Delete: $O(\log_m n)$ (melibatkan merge atau redistribute)

Mirip dengan search, bisa merge atau borrow, tetap logaritmik

B+ TREE

Struktur:

- Variasi dari B-Tree di mana semua kunci hanya disimpan di daun.

- Node internal hanya menyimpan indeks (routing).

- Node daun saling terhubung membentuk linked list.

Kompleksitas Waktu:

- Search: $O(\log_m n)$

Sama seperti B-Tree, tapi traversal selalu sampai ke daun

- Insert: $O(\log_m n)$ (dengan split daun jika penuh)

Menyisipkan data ke daun. Bisa menyebabkan split, tetap logaritmik

- Delete: $O(\log_m n)$ (lebih mudah karena hanya di daun)

Hanya di daun, lebih simpel daripada B-Tree

- Range Query: $O(\log_m n + k)$ (lebih efisien dari B-Tree)

Cari titik awal (O(log n)), lalu baca k elemen berturut-turut (O(k)) via linked list di daun

PERBANDINGAN KOMPLEKSITAS

Operasi	B-Tree	B+ Tree	Keterangan
Search	$O(\log_m n)$	$O(\log_m n)$	Sama, tapi B+ harus sampai ke daun
Insert	$O(\log_m n)$	$O(\log_m n)$	Sama, tapi B+ lebih terstruktur
Delete	$O(\log_m n)$	$O(\log_m n)$	B+ lebih sederhana di daun
Range Query	$O(k \log_m n)$	$O(\log_m n + k)$	B+ jauh lebih efisien

Kelebihan B+ Tree: lebih efisien untuk pencarian berurutan (range query) karena daun terhubung dan semua data ada di daun.

KELOMPOK 6

Hasil Implementasi

LET’S TAKE A LOOK

KELOMPOK 6

```
┌──(idzoyy㉿windows)-[~/ITS/ITS-university/smt2/Struktur-Data-IT-24/tree]
└─$ ./test
Dosen: D001, Matkul: MK001, Kelas: KLS01
Dosen: D002, Matkul: MK002, Kelas: KLS02
Dosen: D003, Matkul: MK003, Kelas: KLS03
Found: Dosen: D002, Matkul: MK002, Kelas: KLS02
```

KELOMPOK 6

Kesimpulan

LET’S TAKE A LOOK

KELOMPOK 6

Pada implementasi algoritma B-Tree dan B+ Tree untuk pencarian kelas pengganti di Universitas Bunda Mulia, terdapat perbedaan signifikan dalam penyimpanan dan akses data.
B-Tree menyimpan data pada internal maupun leaf node, sehingga cocok untuk pencarian acak, seperti mencari kelas pengganti berdasarkan dosen tanpa urutan waktu. Namun, B-Tree kurang efisien jika data harus diakses secara urut karena tidak adanya hubungan langsung antar leaf node.
Sebaliknya, B+ Tree menyimpan data hanya pada leaf node yang saling terhubung dalam bentuk linked list. Struktur ini memungkinkan pencarian terurut menjadi lebih cepat, terutama saat menampilkan jadwal kelas berdasarkan tanggal. B+ Tree juga mendukung range query dengan lebih baik.
Secara keseluruhan, B-Tree lebih cocok untuk pencarian acak, sedangkan B+ Tree unggul dalam pencarian berurutan dan traversal data yang terstruktur. Pemilihan struktur bergantung pada kebutuhan akses data dalam pencarian kelas pengganti.

KELOMPOK 6

THANK YOU SO MUCH