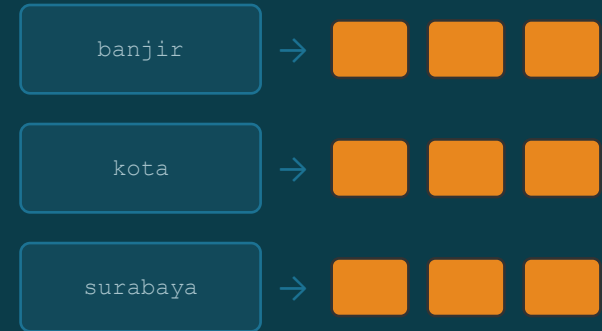


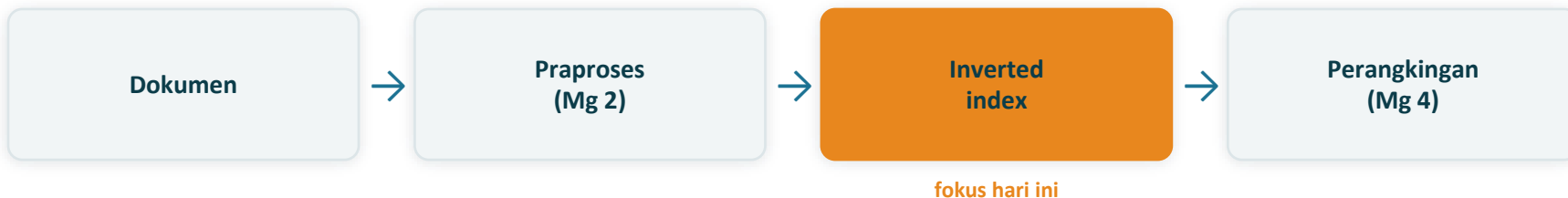
PERTEMUAN 3

Indexing dan Inverted File Structure



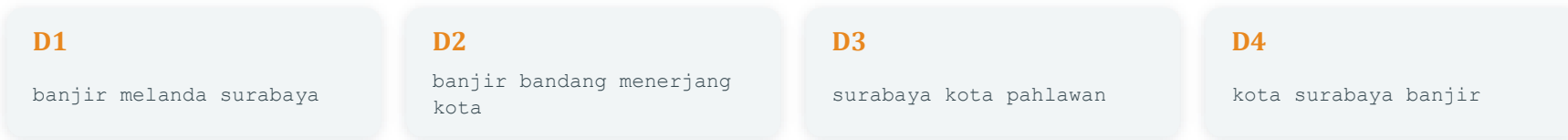
Dari Term Menjadi Struktur Pencarian

Menyusun proses pencarian dari daftar term yang dihasilkan dari pertemuan 2



Korpus contoh yang dipakai

Empat dokumen pendek, sudah melewati praproses berdasarkan pertemuan 2



Pertanyaan: bagaimana menemukan seluruh dokumen yang memuat “banjir” dan “surabaya” tanpa membaca satu pun dokumen dari awal?

Capaian Pembelajaran Pertemuan Ini

Sub-CPMK-3 — turunan dari CPMK-07.1 dan CPL-TF-07

Sub-CPMK-3

Mahasiswa mampu membangun inverted index serta menjalankan Boolean dan wildcard retrieval.

Ranah: C3 (menerapkan), P3 (presisi)

Indikator penilaian

- 1 Ketepatan menjelaskan struktur dictionary dan postings list
- 2 Kebenaran implementasi konstruksi inverted index
- 3 Ketepatan mengeksekusi query Boolean, frasa, dan wildcard

Bentuk penilaian: laporan praktikum dan latihan soal (formatif). Materi ini termasuk cakupan ETS pertemuan 8.

Mengapa Tidak Cukup Memindai Saja?

Merupakan pendekatan paling sederhana, namun gagal ketika koleksi membesar

```
for dok in koleksi:
    if "banjir" in dok.teks and "surabaya" in dok.teks:
        hasil.append(dok)
```

**Benar secara logika,
tetapi tidak dapat dipakai**

1

Biaya sebanding ukuran koleksi

Setiap query membaca ulang seluruh teks. Koleksi sepuluh kali lebih besar berarti query sepuluh kali lebih lambat.

2

Tidak mendukung perangkingan

Hasilnya hanya cocok atau tidak cocok. Tidak ada dasar untuk menyusun sepuluh hasil terbaik.

3

Operasi lanjutan sangat mahal

Query frasa, wildcard, dan NOT menuntut pemindaian berulang pada teks yang sama.

4

Tidak dapat didistribusikan

Struktur teks mentah sulit dipecah dan diparalelkan antar-mesin.

Jalan keluarnya: pindahkan seluruh pekerjaan berat ke tahap offline, lalu simpan hasilnya dalam struktur data khusus.

Term-Document Incidence Matrix

Ide pertama: catat kemunculan setiap term pada setiap dokumen sebagai bit

term	D1	D2	D3	D4
bandang	0	1	0	0
banjir	1	1	0	1
kota	0	1	1	1
landa	1	0	0	0
pahlawan	0	0	1	0
surabaya	1	0	1	1
terjang	0	1	0	0

Query sebagai operasi bit

`banjir AND surabaya AND NOT kota`

```
1101 AND 1011 AND NOT 0111
= 1001 AND 1000
= 1000 → hasilnya D1
```

Mengapa ide ini menarik

Query Boolean menjadi operasi AND, OR, dan NOT pada vektor bit. Operasi yang dieksekusi prosesor sangat cepat, tanpa harus menyentuh teks dokumen sama sekali.

Mengapa Matriks Tidak Dapat Dipakai

Gagasannya benar; ukurannya yang mustahil

Bayangkan satu juta dokumen dan lima ratus ribu term. Matriksnya akan memuat lima ratus miliar sel, padahal sebuah dokumen berita umumnya hanya memuat beberapa ratus term unik.

Hampir seluruhnya bernilai nol

Lebih dari 99,9 persen sel bernilai 0. Kita menyimpan ketiadaan, bukan keberadaan.

Tidak muat di memori

Ukurannya tumbuh sebagai perkalian jumlah term dengan jumlah dokumen.

Solusi

Simpan hanya sel yang bernilai 1, yang disebut inverted index.

Anatomi Inverted Index

Dua bagian: dictionary yang memuat term, dan postings list yang memuat dokumen

DICTIONARY



term	D1	D2	D3	D4
bandang	0	1	0	0
banjir	1	1	0	1
kota	0	1	1	1
landa	1	0	0	0
pahlawan	0	0	1	0
surabaya	1	0	1	1
terjang	0	1	0	0

Yang perlu dicatat

- Dictionary diurutkan menurut abjad supaya term dapat dicari secara biner atau lewat hash.
- Setiap term menyimpan document frequency (DF) yang dipakai untuk pembobotan di pertemuan 4.
- Postings list wajib terurut menurut docID → syarat yang membuat intersect berjalan cepat.

Dictionary biasanya cukup kecil untuk dimuat di memori; postings list disimpan di disk karena jauh lebih besar.

Empat Langkah Membuat Inverted Index

1

Hasilkan pasangan

Telusuri setiap dokumen, keluarkan pasangan (term, docID) untuk setiap token

2

Urutkan

Urutkan seluruh pasangan menurut term, lalu menurut docID di dalam term yang sama

3

Kelompokkan

Gabungkan pasangan dengan term identik menjadi satu entri dictionary

4

Susun postings

Buang docID kembar, hitung document frequency, tulis postings ke penyimpanan

(banjir,1) (landa,1) (surabaya,1) (banjir,2) ...

↓ urutkan

(bandang,2) (banjir,1) (banjir,2) (banjir,4) ...

Langkah pengurutan adalah yang paling mahal. Pada koleksi besar, seluruh pasangan tidak muat di memori → persoalan yang dipecahkan oleh BSBI (*Blocked Sort-Based Indexing*) dan SPIMI (*Single-Pass In-Memory Indexing*).

Mengeksekusi Query Boolean: Algoritma Intersect

Menggabungkan dua postings list terurut dalam satu kali penelusuran

```
i, j, hasil = 0, 0, []
while i < len(p1) and j < len(p2):
    if p1[i] == p2[j]: hasil.append(p1[i]); i += 1; j += 1
    elif p1[i] < p2[j]: i += 1
    else: j += 1
```

Biaya

Sebanding dengan jumlah panjang kedua postings, bukan dengan ukuran koleksi → sumber kecepatan mesin pencari.

Penelusuran query: banjir AND surabaya

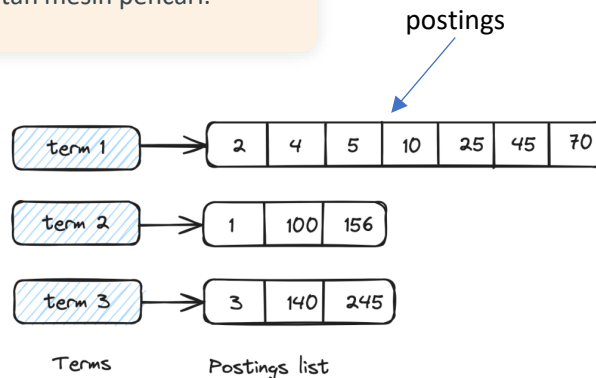
banjir D1 → D2 → D4

surabaya D1 → D3 → D4

hasil D1, D4

Langkah perbandingan

1 = 1 → simpan D1, majukan keduanya
2 < 3 → majukan penunjuk banjir
4 > 3 → majukan penunjuk surabaya
4 = 4 → simpan D4, selesai

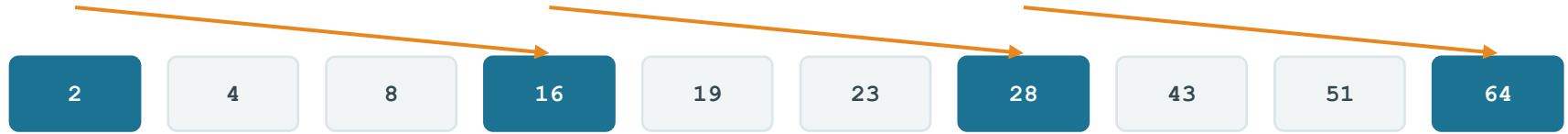


Untuk query dengan banyak term, kerjakan pasangan dengan document frequency terkecil lebih dahulu untuk menjaga agar daftar antara tetap pendek.

Mempercepat Intersect: Skip Pointer

Menambah jalan pintas agar bagian postings yang pasti tidak cocok dapat dilompati

skip pointer



postings list terurut

1

Cara kerjanya

Bila nilai pada ujung lompatan masih lebih kecil daripada target, seluruh elemen di antaranya dapat dilewati sekaligus.

2

Berapa banyak pointer

Aturan praktis yang lazim: sekitar akar dari panjang postings, disebar merata.

3

Kapan menguntungkan

Efektif pada index statis. Pada index yang sering berubah, biaya pemeliharaan pointer justru akan memberatkan.

Positional Index untuk Query Frasa

Boolean saja tidak cukup: “kota surabaya” berbeda dari “surabaya kota”

kota → D2:[4] D3:[2] D4:[1]
surabaya → D1:[3] D3:[1] D4:[2]

Setiap posting kini menyimpan daftar posisi kemunculan term di dalam dokumen.

Menguji frasa “kota surabaya”

D3

kota di posisi 2, surabaya di posisi 1

$2 + 1 \neq 1 \rightarrow$ bukan frasa

D4

kota di posisi 1, surabaya di posisi 2

$1 + 1 = 2 \rightarrow$ frasa cocok

Harga yang dibayar

Ukuran index membengkak beberapa kali lipat, karena setiap kemunculan token dicatat, bukan hanya setiap dokumen.

Manfaat tambahan

Posisi juga memungkinkan query kedekatan dan menjadi sinyal perangkikan: term yang berdekatan menandakan relevansi lebih tinggi.

Dua Cara Menangani Frasa

Biword index lebih sederhana, positional index lebih umum → keduanya sering dipakai bersama

Aspek	Biword Index	Positional Index
Isi index	Pasangan kata berurutan sebagai satu term	Term tunggal beserta daftar posisinya
Contoh entri	“kota surabaya” sebagai satu term	kota → D4:[1]
Frasa dua kata	Sekali pencarian, sangat cepat	Perlu pemeriksaan selisih posisi
Frasa panjang	Hanya perkiraan; dapat menghasilkan positif palsu	Tepat untuk panjang berapa pun
Ukuran index	Kamus membengkak sangat cepat	Postings membengkak, kamus tetap
Praktik lazim	Dipakai untuk frasa yang sangat sering dicari	Menjadi pilihan utama sistem modern

Strategi gabungan: simpan biword hanya untuk frasa populer dari log query, selebihnya tangani dengan positional index.

Konstruksi Index Skala Besar

Ketika seluruh pasangan (term, docID) tidak muat di memori

BSBI

Blocked Sort-Based Indexing

- Petakan setiap term menjadi termID lebih dahulu
- Proses koleksi per blok yang muat di memori
- Urutkan tiap blok, tulis index parsial ke disk
- Gabungkan seluruh blok dengan merge berjenjang

Kelemahan: pemetaan term ke termID sendiri memerlukan memori yang besar.

SPIMI

Single-Pass In-Memory Indexing

- Tidak memakai termID; bekerja langsung dengan term
- Bangun dictionary dan postings per blok secara langsung
- Postings tumbuh dinamis, tanpa tahap pengurutan global
- Bila memori penuh, tulis blok ke disk lalu mulai blok baru

Lebih cepat dan lebih hemat memori; menjadi dasar banyak mesin index modern.

Pada skala web, kedua pendekatan dijalankan secara terdistribusi → pola MapReduce: pemetaan menghasilkan pasangan, pereduksi menyusun postings per term.

Index yang Terus Berubah

Koleksi nyata bertambah, berubah, dan dihapus setiap saat

1

Index tambahan

Simpan dokumen baru pada index kecil di memori. Query dijalankan pada index utama dan index tambahan, hasilnya digabungkan.

2

Penggabungan berjenjang

Ketika index tambahan penuh, gabungkan ke index yang lebih besar — pola serupa penambahan bilangan biner.

3

Penghapusan

Dokumen tidak benar-benar dibuang. Sistem menandainya pada vektor bit penghapusan dan menyaringnya saat query.

4

Pembangunan ulang berkala

Cara paling sederhana bila koleksi tidak terlalu besar: bangun ulang seluruh index di luar jam sibuk.

Implikasi bagi final project: rancang sejak awal bagaimana index Anda akan diperbarui, jangan hanya bagaimana ia dibangun.

Wildcard Query: Permuterm dan k-gram

Menjawab query seperti $sura^*$, $*baya$, atau sur^*ya tanpa memindai seluruh kamus

Permuterm Index

Simpan seluruh rotasi term dengan penanda akhir \$

kota\$ ota\$k ta\$ko a\$kot \$kota

Query ko^* diputar menjadi $\$ko^*$ lalu cukup dicari sebagai awalan pada index yang terurut.

k-gram Index

Petakan setiap potongan k huruf ke daftar term yang memuatnya

$\$ko \rightarrow kota, kolam$ $ota \rightarrow kota, biota$

Query ko^*a dipecah menjadi $\$ko$ dan $a\$$. Hasil irisan kedua daftar lalu disaring ulang.

1

Kecepatan pencarian

Permuterm sangat cepat karena hanya perlu pencarian awalan

2

Ukuran penyimpanan

Permuterm membengkak berkali lipat; k-gram jauh lebih hemat

3

Penyaringan akhir

k-gram menghasilkan calon yang perlu diperiksa ulang terhadap pola aslinya

Koreksi Ejaan pada Query

Pengguna salah ketik; sistem tetap harus menemukan dokumen yang benar

Edit distance

Jumlah minimum penyisipan, penghapusan, dan penggantian huruf untuk mengubah satu kata menjadi kata lain.

surabaia	→	surabaya	jarak 1
bnjir	→	banjir	jarak 1
kota	→	kot	jarak 1

Dihitung dengan pemrograman dinamis; biayanya terlalu mahal bila diterapkan ke seluruh kamus.

Menyaring calon lebih dahulu

Pakai k-gram index untuk mengambil kata yang berbagi banyak potongan huruf dengan query, baru hitung edit distance pada calon tersebut.

Pemilihan koreksi juga mempertimbangkan frekuensi kata dalam korpus dan frekuensi query dalam log pencarian, bukan hanya jarak huruf.

Catatan untuk Bahasa Indonesia

Soundex dan algoritma fonetik sejenis dirancang untuk pelafalan Bahasa Inggris, sehingga hasilnya kurang tepat untuk kata Indonesia. Untuk korpus lokal, kombinasi edit distance dengan kamus kata baku dan daftar kesalahan ketik yang lazim biasanya memberi hasil lebih baik.

Kompresi Dictionary dan Postings

Index yang lebih kecil berarti lebih banyak bagian yang muat di memori dan query lebih cepat

Postings: simpan selisih, bukan nilai penuh

docID asli :	824	829	839	1001	1004
selisih :	824	5	10	162	3

Angka kecil dapat disimpan dengan bit jauh lebih sedikit.

Gap encoding

Postings

Simpan selisih antar-docID. Efektif karena postings selalu terurut menaik.

Variable byte

Postings

Pakai jumlah byte seperlunya; satu bit menjadi penanda lanjutan.

Front coding

Dictionary

Kata berurutan berbagi awalan yang sama, cukup simpan bagian yang berbeda.

Dictionary as a string

Dictionary

Simpan seluruh term sebagai satu untaian panjang beserta tabel penunjuk.

Ada trade-off antara berkurangnya jumlah ruang penyimpanan dengan bertambahnya sedikit waktu pemrosesan. Pada sistem IR, trade-off ini hampir selalu menguntungkan.

Praktikum: Membangun Inverted Index dari Nol

Lanjutkan modul praproses pertemuan 2; dikumpulkan melalui Ilmu2

1

Muat korpus

Pakai korpus berita yang sama dengan praktikum pertemuan 2

2

Bangun inverted index

Struktur: term $\rightarrow$ (df, daftar docID terurut). Dilarang memakai pustaka index siap pakai

3

Terapkan intersect

Tulis sendiri algoritma penggabungan dua postings list terurut

4

Dukung AND, OR, NOT

Terima query Boolean sederhana dan kembalikan daftar dokumen

5

Tambahkan posisi

Perluas menjadi positional index, lalu uji satu query frasa

6

Laporkan ukuran

Bandingkan ukuran index dengan dan tanpa informasi posisi

```
from collections import defaultdict
```

```
index = defaultdict(list)
```

Rangkuman & Persiapan Pertemuan Berikutnya

- Inverted index membalik arah pemetaan: dari dokumen ke term, menjadi term ke daftar dokumen.
- Postings list yang terurut adalah syarat mutlak yang merupakan kelebihan algoritma intersect dalam hal kecepatannya.
- Positional index memungkinkan query frasa dan kedekatan, dengan harga index yang jauh lebih besar.
- Permuterm dan k-gram menjawab wildcard; gap encoding dan front coding menekan ukuran index.

Persiapan Pertemuan 4

Baca Manning dkk. Bab 6. Bawa inverted index hasil praktikum. Bobot akan ditambahkan pada setiap posting agar hasil dapat diurutkan, bukan sekadar dicocokkan.

Pustaka pertemuan ini

- [1] Manning, Raghavan & Schütze (2008), Bab 1, 3, 4, 5
- [6] Büttcher, Clarke & Cormack (2010), Bab 4