Analisis Cross-Reference dalam Teks Alkitab Versi WEB Menggunakan Algoritma String Matching dan Regular Expression

Samantha Laqueenna Ginting - 13523138
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: lana.ginting@gmail.com , 13523138@std.stei.itb.ac.id

***Abstrak*—Penelitian ini bertujuan untuk mengidentifikasi dan menganalisis keterkaitan antar ayat dalam teks Alkitab melalui pendekatan algoritma pencocokan *string* (*string matching*) dan ekspresi regular (*regular expression*). *Cross-reference* atau rujukan silang merupakan sebuah elemen penting dalam pemahaman Alkitab. Referensi silang terjadi di mana suatu ayat dapat merujuk atau mengulang konsep dari ayat atau pasal lain, baik secara eksplisit maupun implisit. Dalam penelitian ini, digunakan teknik-teknik untuk mengekstraksi pola referensial antar ayat dan pasal. Teks Alkitab yang dianalisa adalah terjemahan versi World English Bible (WEB) dalam format digital. Hasil dari pendekatan ini menunjukan efektivitas metode komputasional dalam menemukan rujukan silang yang tersebar di berbagai kitab dan pasal, serta memberikan kontribusi terhadap studi biblika dan eksplorasi semantik kitab suci secara sistematis.**

Kata Kunci*—Alkitab; cross-reference; string matching; regular expression

I. Pendahuluan

Alkitab merupakan salah satu teks yang paling banyak dibaca, diteliti, dan dianalisis sepanjang sejarah, dengan struktur yang kompleks dan kaya akan makna teologis, filosofis, dan historis. Salah satu aspek penting dalam studi Alkitab bagi umat Kristiani adalah keberadaan referensi silang, yaitu ayat-ayat atau pasal-pasal yang saling merujuk satu sama lain untuk memperkuat narasi, doktrin, atau tema tertentu. Rujukan silang sering kali bersifat eksplisit melalui penggunaan frasa atau istilah yang sama, namun juga dapat bersifat implisit dan tersembunyi di dalam struktur naratif yang lebih luas. Studi manual terhadap referensi silang telah dilakukan oleh banyak ahli teologi dan peneliti literatur selama berabad-abad. Namun pendekatan komputasional memberikan kecepatan, objektivitas, dan cakupan yang lebih luas untuk mengidentifikasi keterkaitan-keterkaitan ini.

Dalam konteks ilmu komputer, algoritma pencocokan *string* dan ekspresi regular menawarkan pendekatan yang sistematis untuk mengekstraksi pola dan frasa yang berulang dalam teks. Berbagai pilihan algoritma seperti Knuth-Morris-Pratt, Boyer-Moore, dan Aho-Corasick memungkinkan pencarian pola secara efisien dalam skala besar, sementara Regular Expression menyediakan cara yang fleksibel dan deklaratif untuk mendeteksi struktur linguistik tertentu.

Penelitian ini memanfaatkan kedua pendekatan tersebut untuk menganalisis teks Alkitab versi World English Bible (WEB) dalam format digital, dengan tujuan untuk mengungkapkan dan memetakan referensi silang antar ayat yang tersebar di seluruh kitab. Pendekatan ini diharapkan dapat memperkaya studi biblika serta mendemonstrasikan peran penting algoritma dalam eksplorasi literatur.

II. Dasar Teori

Pencocokan string (*string matching*) adalah proses menemukan semua kemunculan sebuah pola (*pattern*) **P** di dalam teks (*text)* **T**. Tujuan dari teknik ini adalah untuk menentukan semua posisi **i** di dalam teks **T** sehingga **T[i … i + m - 1] = = P[0 … m - 1]** dengan **m** = | **P** | dan **n** = | **T** |.

Teknik ini digunakan untuk menyelesaikan berbagai macam permasalahan fundamental dalam ilmu komputer dan teori algoritma, seperti bioinformatika (pencocokan urutan DNA), deteksi plagiarisme, pengolahan bahasa alami (NLP), dan sistem informasi.

A. Naïve String Matching

Seperti halnya algoritma Greedy, algoritma Naïve String Matching merupakan pendekatan paling *straight forward*. Teknik ini membandingkan pola **P** dengan semua kemungkinan *substring* **T[i … i + m - 1]** satu per satu.

Jika panjang pola **P** adalah **n** dan panjang teks **T** adalah **m**, untuk setiap posisi **i = 0** hingga **n – m**, periksa apakah **T[i + j] = = P[j]** untuk semua **j = 0** hingga **m – 1**. Jika ya, maka ditemukan kecocokan di posisi **i**. Dari langkah-langkah tersebut, dapat disimpulkan bahwa kompleksitas waktu dari algoritma ini adalah **O(n x m)**, karena setiap pergeseran dianalisa hingga karakter ke-**m**. Sementara kompleksitas ruang dari algoritma ini adalah **O(1)**.

Algoritma ini mudah untuk diimplementasikan dan sangat cocok untuk *string* yang pendek atau sistem dengan keterbatasan memori. Namun, algoritma ini menjadi sangat tidak efisien untuk pola yang panjang atau teks yang besar.

B. Knuth-Morris-Pratt (KMP) Algorithm

Algoritma ini ditemukan oleh Donald Ervin Knuth, Vaughan Pratt, dan James H. Morris. Algoritma ini merupakan penyempurnaan dari metode konvensional, dengan keunggulan utama dalam efisiensi pencocokan string tanpa melakukan perulangan yang tidak perlu.

Pada dasarnya, algoritma ini berjalan dari kiri ke kanan, namun saat terjadi ketidakcocokan (mismatch), algoritma ini tidak memulai pencocokan dari posisi awal, melainkan menggunakan informasi dari pencocokan sebelumnya untuk menentukan seberapa jauh pola bisa digeser.

Inti dari efisiensi algoritma KMP terletak pada struktur data yang disebut fungsi pinggiran, atau sering juga disebut sebagai tabel prefiks. Tabel ini menyimpan informasi tentang panjang prefiks yang juga merupakan sufiks dari *substring* pola hingga indeks ke-**q**. Secara formal, **π[q]** = panjang maksimum proper prefiks (prefiks yang bukan keseluruhan dari *string* itu sendiri) dari **P[0…q]** yang juga sufiks dari **P[0…q]**.

Misal diberikan pola P = “ABABAC”, maka tabel prefiks dari pola tersebut adalah,

i (indeks)	**0**	**1**	**2**	**3**	**4**	**5**
P[i]	A	B	A	B	A	C
π[i]	0	0	1	2	3	0

Dari tabel di atas,

- π[2] = 1, karena prefiks “A” = sufiks “A”
- π[4] = 3, karena prefiks “ABA” = sufiks “ABA”
- π[5] = 0, karena tidak ada proper prefiks yang juga sufiks di P[0…5]

Setelah melakukan *preprocessing* dengan membangun tabel prefiks, dilakukan proses pencocokan pada teks **T**. Jika terjadi ketidakcocokan saat membandingkan karakter **P[j]** ≠ **T[i]**, maka **j** dibuat ke **p[j - 1]**. Langkah ini memungkinkan untuk melewati sebagian besar perbandingan berulang.

Kompleksitas waktu dari algoritma ini adalah **O(n + m)** dengan **O(m)** adalah kompleksitas waktu membangun tabel prefiks, dan **O(n)** merupakan waktu pencocokannya. Sementara kompleksitas ruangnya adalah **O(m)** dari proses membangun tabel prefiks.

Dengan menggunakan algoritma ini, pencarian pola menjadi lebih efektif. Algoritma ini juga lebih stabil pada teks dan pola yang relatif panjang.

C. Boyer-Moore (BM) Algorithm

Algoritma Boyer-Moore diperkenalkan oleh Robert S. Boyer dan J. Strother Moore pada tahun 1977. Algoritma ini dipertimbangkan sebagai salah satu algoritma pencocokan *string* yang paling efisien dan paling sering digunakan. Sistem pencarian teks, *command grep* dan *diff* diaplikasikan menggunakan algoritma ini.

Pencocokan dilakukan dari kanan ke kiri dalam pola **P**. Ketika terjadi ketidakcocokan, ia menggunakan informasi dari pola itu sendiri untuk menentukan seberapa jauh pola dapat digeser. Untuk menentukan pergeseran, dua strategi heuristik digunakan.

1. *Bad Character Heuristic*
   Jika terjadi ketidakcocokan antara karakter **P[j]** dan **T[i + j]**, pola digeser sehingga,
   - Karakter teks **T[i + j]** selaras dengan kemunculan terakhir karakter tersebut dalam pola, atau
   - Jika karakter tidak muncul dalam pola, geser pola sepenuhnya melewati karakter tersebut.
2. *Good Suffix Heuristic*
   Jika terjadi ketidakcocokan setelah sebagain sufiks dari pola telah cocok, pola digeser sehingga,
   - Kemunculan sebelumnya dari sufiks tersebut di pola menjadi sejajar dengan teks, atau
   - Prefiks dari pola yang cocok dengan sufiks teks menjadi sejajar.

Untuk melakukan pencocokan dengan algoritma Boyer-Moore, dua tabel dari kedua heuristik dibangun terlebih dahulu. Kemudian, pola digeser dari kiri ke kanan atas teks, tetapi cocokkan karakter pola dari kanan ke kiri. Jika seluruh karakter cocok, maka *match* ditemukan. Namun jika terjadi ketidakcocokan, gunakan hasil maksimum dari kedua heuristik untuk menggeser pola.

Kompleksitas waktu dari *preprocessing* adalah **O(m + Σ)** dan rata-rata kompleksitas waktu dari proses pencocokannya adalah **O(n / m)**. Sementara kompleksitas ruang dari kedua tabel heuristik adalah **O(m + Σ)**. Dengan Σ adalah ukuran alfabet.

Keunggulan dari algoritma ini adalah kecepatan terutama pada teks berukuran besar karena tidak perlu membandingkan semua karakter dalam teks. Sehingga algoritma ini cocok untuk aplikasi real-time yang membutuhkan efisiensi tinggi. Namun, implementasinya lebih kompleks dari algoritma lain dan kurang efisien untuk pola pendek.

D. Aho-Corasick Algorithm

Algoritma Aho-Corasick ditemukan oleh Alfred V. Aho dan Margaret J. Corasick pada tahun 1975. Algoritma ini dirancang untuk menyelesaikan masalah pencocokan banyak pola (*multi-pattern matching*) dalam satu kali pemindaian teks.

Algoritma ini dibangun dalam tiga tahap utama,

1. *Trie Construction*
   Sebuah trie (pohon prefiks) dibangun dari semua pola. Setiap node merepresentasikan sebuah prefiks dari pola-pola. Tiap daun mewakili akhir dari sebuah pola.
2. *Failure Links*
   Setelah trie dibangun, setiap node diberikan *failure link*, mirip dengan **π** di KMP. Jika pada saat pencocokan terjadi ketidakcocokan, pola lain akan dicoba.
3. *Output Links*
   Setiap node menyimpan output list, yaitu sebuah daftar semua pola yang selesai di node tersebut, karena sebuah node bisa merupakan node akhir lebih dari satu pola.

Traversal teks dilakukan sekali dari kiri ke kanan. Pada setiap karakter **T[i]**, automaton berpindah ke *node* berikutnya jika terdapat transisi. Jika tidak, mengikuti *failure link* hingga menemukan transisi atau kembali ke *root*. Jika sampai pada node dengan *output list* tidak kosong, berarti ditemukan satu atau lebih pola.

Kompleksitas waktu pembangunan trie dan pembangunan failure masing-masing adalah **O(M)**. Sementara kompleksitas proses pencocokan adalah **O(n + z)**. Kompleksitas ruangnya adalah **O(M x Σ)**. Dengan Σ adalah ukuran alfabet.

Algoritma Aho-Corasick dapat mencocokkan ratusan hingga ribuan pola sekaligus dengan hanya satu lintasan teks. Algoritma ini diaplikasikan pada antivirus, *firewall*, dan *intrusion detection systems*.

E. Regular Expression (Regex)

Regular Expression atau ekspresi regular adalah suatu notasi atau pola yang digunakan untuk menyatakan himpunan *string* tertentu. Regex memungkinkan untuk mendefinisikan pola pencocokan secara fleksibel dan deklaratif.

Berikut merupakan simbol-simbol tertentu sebagai metakarakter,

Simbol	**Makna**
.	Mewakili sembarang satu karakter
*	Ulangi karakter sebelumnya sebanyak 0 kali atau lebih
+	Ulangi karakter sebelumnya sebanyak 1 kali atau lebih
?	Ulangki karakter sebelumnya sebanyak 0 atau 1 kali
^	Awal *string*
$	Akhir *string*
[…]	Salah satu karakter dalam himpunan
( )	Grup
\	Escape karakter literal

Sebagai contoh, jika diberikan regex "^a.*z$", regex ini diawali dengan a dan diakhiri dengan z. *String* yang cocok untuk regex tersebut adalah "abcz" dan "a123z".

Umumnya, komputasi regex membutuhkan kompleksitas waktu **O(m x n)**. Namun untuk regex dengan *backtracking*, kompleksitas dapat berubah menjadi eksponensial dalam kasus buruk.

F. Fuzzy Matching

Fuzzy matching atau pencocokan kabur adalah teknik pencocokan *string* yang memungkinkan pencarian kata atau frasa yang mirip atau hampir cocok, bukan hanya yang identik secara literal. Pendekatan ini sangat berguna ketika terjadi variasi dalam penulisan, kesalahan ketik, perbedaan ejaan, atau ketika membandingkan teks dengan struktur alami (*natural language*).

Pencocokan ini bekerja dengan menghitung skor kemiripan (*similarity score*) antara dua *string*. Salah satu metode paling umum adalah Rasio Kesamaan Levenshtein atau Levenshtein Distance.

Levenshtein Distance antara dua *string* **A** dan **B** adalah jumlah minimum operasi penyisipan (*insertion*), penghapusan (*deletion*), atau penggantian (*substitution*) yang diperlukan untuk mengubah **A** menjadi **B**.

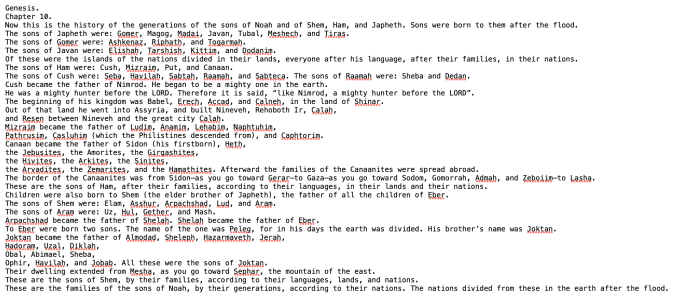
III. PEMBAHASAN

Penelitian ini mengimplementasikan sebuah sistem analisis teks berbasis Python. Tiga fokus utama dalam penelitian ini adalah,

- Mendeteksi kemunculan tema-tema populer namun penting seperti perjanjian, penebusan, dan nubuat, dengan menggunakan algoritma Boyer-Moore dan ekspresi regular (regex).

- Menganalisis pola rujukan silang, yaitu ayat-ayat yang secara eksplisit menyebutkan kitab lain, dengan teknik berbasis regex dan pemetaan fuzzy.
- Menganalisis frekuensi kata-kata atau nama-nama penting yang paling sering muncul dalam keseluruhan teks Alkitab.

Pemrosesan data mentah dimulai dengan membaca semua *file* .txt yang mewakili tiap-tiap pasal dari kitab-kitab dalam Alkitab. Baris pertama dalam *file* merupakan nama kitab dan baris kedua merupakan nomor pasalnya. Namun, pada data yang digunakan dalam penelitian, tidak terdapat penulisan nomor ayat secara spesifik. Sehingga, penelitian ini mengasumsikan tiap baris merupakan ayat-ayat yang berbeda. Berikut merupakan contoh data mentah yang digunakan penelitian.


Genesis.
Chapter 10.
Now this is the history of the generations of the sons of Noah and of Shem, Ham, and Japheth. Sons were born to them after the flood.
The sons of Japheth were: Gomer, Magog, Madai, Javan, Tubal, Meshech, and Tiras.
The sons of Gomer were: Ashkenaz, Riphath, and Togarmah.
The sons of Javan were: Elishah, Tarshish, Kittim, and Dodanim.
Of these were the islands of the nations divided in their lands, everyone after his language, after their families, in their nations.
The sons of Ham were: Cush, Mizraim, Put, and Canaan.
The sons of Cush were: Seba, Havilah, Sabtah, Raamah, and Sabteca. The sons of Raamah were: Sheba and Dedan.
Cush became the father of Nimrod. He began to be a mighty one in the earth.
He was a mighty hunter before the LORD. Therefore it is said, "like Nimrod, a mighty hunter before the LORD".
The beginning of his kingdom was Babel, Erech, Accad, and Calneh, in the land of Shinar.
Out of that land he went into Assyria, and built Nineveh, Rehoboth Ir, Calah,
and Resen between Nineveh and the great city Calah.
Mizraim became the father of Ludim, Anamim, Lehabim, Naphtuhim,
Pathrusim, Casluhim (which the Philistines descended from), and Caphtorim.
Canaan became the father of Sidon (his firstborn), Heth,
the Jebusites, the Amorites, the Girgashites,
the Hivites, the Arkites, the Sinites,
the Arvadites, the Zemarites, and the Hamathites. Afterward the families of the Canaanites were spread abroad.
The border of the Canaanites was from Sidon—as you go toward Gerar—to Gaza—as you go toward Sodom, Gomorrah, Admah, and Zeboiim—to Lasha.
These are the sons of Ham, after their families, according to their languages, in their lands and their nations.
Children were also born to Shem (the elder brother of Japheth), the father of all the children of Eber.
The sons of Shem were: Elam, Asshur, Arpachshad, Lud, and Aram.
The sons of Aram were: Uz, Hul, Gether, and Mash.
Arpachshad became the father of Shelah. Shelah became the father of Eber.
To Eber were born two sons. The name of the one was Peleg, for in his days the earth was divided. His brother's name was Joktan.
Joktan became the father of Almodad, Sheleph, Hazarmaveth, Jerah,
Hadoram, Uzal, Diklah,
Obal, Abimael, Sheba,
Ophir, Havilah, and Jobab. All these were the sons of Joktan.
Their dwelling extended from Mesha, as you go toward Sephar, the mountain of the east.
These are the sons of Shem, by their families, according to their languages, lands, and nations.
These are the families of the sons of Noah, by their generations, according to their nations. The nations divided from these in the earth after the flood.

Gambar 1. Contoh File .txt yang Digunakan

Seluruh data kemudian diproses melalui fungsi **load_all_ayat()** yang bertujuan untuk melakukan *parsing* data mentah ke dalam struktur tuple (kitab, pasal, nomor ayat, isi kalimat) yang disimpan ke dalam variable **ayat_all**.

Pertama, sistem akan mengeksekusi pencarian berbasis daftar kata kunci tematik. Penelitian ini mengambil tiga tema besar dengan kata kunci yang umum,

- Perjanjian: "*covenant*", "*promise*", "*oath*", "*agreement*", "*blessing*", "*testament*"
- Penebusan: "*sin*", "*transgression*", "*iniquity*", "*redeem*", "*redemption*", "*save*", "*salvation*", "*atonement*", "*sacrifice*", "*grace*", "*faith*", "*forgive*"
- Nubuat: "*prophecy*", "*prophet*", "*foretold*", "*fulfill*", "*Messiah*", "*Christ*", "*virgin*", "*born*", "*Bethlehem*", "*pierced*", "*crucified*", "*resurrected*"

Pencarian dilakukan dengan algoritma Boyer-Moore di fungsi **boyer_moore_fixed_search(text, pattern)**. *String* dicari dari kanan ke kiri menggunakan heuristik *bad character* untuk mengatur pergeseran pola.

Preprocessing dilakukan di fungsi **_build_bad_char_table(pattern)**. Fungsi tersebut mencatat posisi kemunculan terakhir dari setiap karakter dalam *pattern*.

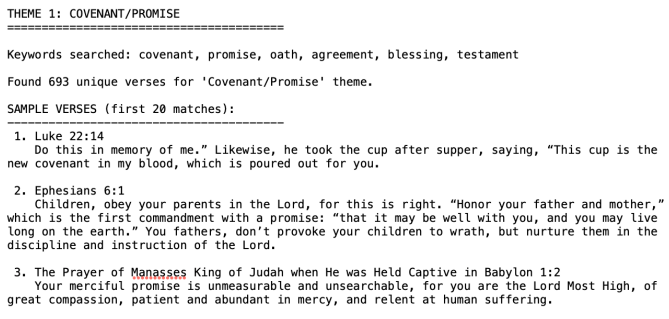
```
def _build_bad_char_table(pattern):
    table = {}
    for i in range(len(pattern)):
        table[pattern[i]] = i
    return table
```

Setelah selesai membuat tabel, pencocokan dimulai dari ujung kanan pola (**j = m – 1**) dan mundur ke kiri. Jika semua karakter cocok hingga **j < 0**, maka pencocokan berhasil. Namun jika terjadi ketidakcocokan, gunakan karakter **text[shift + j]** sebagai acuan untuk menghitung seberapa jauh pola dapat digeser.

```
shift = 0
    while shift <= n - m:
        j = m - 1
        while j >= 0 and pattern[j] == text[shift +
j]:
            j -= 1

        if j < 0:
            return True
        else:
            bad_char = text[shift + j]
            if bad_char in bad_char_table:
                shift += max(1, j -
bad_char_table[bad_char])
            else:
                shift += j + 1
```

Sistem akan mengembalikan ayat-ayat yang mengandung kata-kata kunci tersebut. Dalam gambar yang terlampir, sistem menunjukkan 20 ayat pertama dari 693 ayat unik yang mengandung tema perjanjian (*covenant*/*promise*).


THEME 1: COVENANT/PROMISE
==========

Keywords searched: covenant, promise, oath, agreement, blessing, testament

Found 693 unique verses for 'Covenant/Promise' theme.

SAMPLE VERSES (first 20 matches):
---

1. Luke 22:14
   Do this in memory of me." Likewise, he took the cup after supper, saying, "This cup is the new covenant in my blood, which is poured out for you.

2. Ephesians 6:1
   Children, obey your parents in the Lord, for this is right. "Honor your father and mother," which is the first commandment with a promise: "that it may be well with you, and you may live long on the earth." You fathers, don't provoke your children to wrath, but nurture them in the discipline and instruction of the Lord.

3. The Prayer of Manasses King of Judah when He was Held Captive in Babylon 1:2
   Your merciful promise is unmeasurable and unsearchable, for you are the Lord Most High, of great compassion, patient and abundant in mercy, and relent at human suffering.

Gambar 2. Contoh Hasil Algoritma Boyer-Moore untuk Pencarian Berbasis Kata Kunci Tematik

Pola-pola teks yang lebih kompleks juga dicari menggunakan pendekatan ekspresi regular (*regex pattern*). Berbeda dengan pencarian kata kunci biasa, pola regex memungkinkan pencocokan terhadap struktur frasa yang bervariasi namun memiliki makna serupa. Misalnya, frasa seperti "*make a covenant*", "*establish a covenant*", maupun "*everlasting covenant*" akan tetap dikenali meskipun susunan katanya berbeda. Pola ini dikompilasi menggunakan **re.compile()**.

```
elif algorithm_name == "regex_pattern":
        compiled_pattern = re.compile(keyword,
re.IGNORECASE)
```

Sistem akan mengembalikan jumlah ayat yang ditemukan dari pola regex tertentu, dan menunjukkan beberapa ayat pertamanya.

```
COMPLEX PATTERN MATCHES:
----------------------------------------
Pattern 1: (?:new|everlasting|eternal)\s+covenant
Matches: 32
  1. Luke 22:14 - Do this in memory of me." Likewise, he took the cup after supper, saying,
"This cup is the new coven...
  2. Jeremiah 32:32 - I will make an everlasting covenant with them, that I will not turn away
from following them, to do ...
  3. 1 Corinthians 11:33 - Do this in memory of me." In the same way he also took the cup after
supper, saying, "This cup is th...
  4. 2 Esdras 3:13 - You loved, and to him only you showed the end of the times secretly by
night, and made an everlastin...
  5. Ezekiel 16:57 - Nevertheless I will remember my covenant with you in the days of your
youth, and I will establish an...
  ... and 27 more.

Pattern 2: (?:promise|oath)\s+(?:to|of)\s+(?:Abraham|David|Israel)
Matches: 3
  1. Hebrews 6:8 - For when God made a promise to Abraham, since he could swear by no one
greater, he swore by himself,...
  2. The Second Book of Chronicles 1:8 - Now, LORD God, let your promise to David my father be
established; for you have made me king over a ...
  3. Romans 4:13 - For the promise to Abraham and to his offspring that he would be heir of the
world wasn't through th...
```

Gambar 3. Contoh Hasil Pencarian Regex dengan Pola Kompleks

Setelah tiga pencarian tematik dilakukan, sistem juga menjalankan deteksi referensi silang eksplisit antar ayat, yaitu ayat-ayat yang secara langsung menyebut nama kitab lain dalam teksnya. Deteksi ini dilakukan melalui fungsi **detect_explicit_biblical_references_optimized()**, yang menggabungkan teknik pencocokan regex dengan *fuzzy matching* menggunakan pustaka Python **difflib**.

```
matches = pattern.finditer(sentence)
        for match in matches:
            ref_name_raw = match.group(1).strip()

            normalized_ref_name =
ref_name_raw.lower()
            if normalized_ref_name.startswith("the
"):
                normalized_ref_name =
normalized_ref_name[4:]
```

```
            close_matches =
difflib.get_close_matches(
                normalized_ref_name,
                alias_keys,
                n=1,
                cutoff=similarity_threshold
            )

            if close_matches:
                best_match_key = close_matches[0]

                score =
difflib.SequenceMatcher(None, normalized_ref_name,
best_match_key).ratio()
                target_kitab =
alias_map[best_match_key]
                result.append(((kitab, chapter,
verse), target_kitab, ref_name_raw, score))
```

Fungsi ini mendeteksi pola frasa seperti "a*s it is written in Isaiah*" atau "*spoken by the prophet Daniel*", kemudian mengekstrak nama kitab yang disebut, dan mencocokannya dengan daftar **alias_map** yang telah diperluas. Jika nama tersebut cukup mirip (dengan skor *threshold* di atas ambang batas), maka ayat tersebut dicatat sebagai rujukan silang eksplisit.

```
alias_map = {
    "moses": "Exodus",
    "prophet isaiah": "Isaiah",
    "isaiah": "Isaiah",
    "jeremiah": "Jeremiah",
    "the law": "Leviticus",
    "law of moses": "Deuteronomy",
    "psalms": "Psalms",
    "book of psalms": "Psalms",
    "daniel": "Daniel",
    "matthew": "Matthew",
    "luke": "Luke",
    "john": "John",
    "paul": "Romans",
    "peter": "1 Peter",
    "revelation": "Revelation",
    "genesis": "Genesis",
    "exodus": "Exodus",
    "leviticus": "Leviticus",
    "numbers": "Numbers",
    "deuteronomy": "Deuteronomy",
...
```

Sistem akan mengembalikan referensi silang yang ditemukan berdasarkan pemetaan alias yang telah diberikan. Dalam gambar yang terlampir, sistem menunjukkan jumlah, skor similaritas, dan ayat yang memberikan referensi secara eksplisit.

```
EXPLICIT BIBLICAL CROSS-REFERENCES
==================================

Similarity threshold: 0.7

Detected 48 explicit cross-reference pairs.

TOP CROSS-REFERENCES (sorted by similarity score):
--------------------------------------------------
 1. Acts 27:10 -> 'Paul' (Romans)
    Similarity Score: 1.000
    Verse: When much time had passed and the voyage was now dangerous because the Fast had now
already gone by, Paul admonished them and said to them, "Sirs, I p...

 2. Luke 24:37 -> 'law of Moses' (Deuteronomy)
    Similarity Score: 1.000
    Verse: He said to them, "This is what I told you while I was still with you, that all things
which are written in the law of Moses, the prophets, and the psa...

 3. Acts 13:41 -> 'Paul' (Romans)
    Similarity Score: 1.000
    Verse: But when the Jews saw the multitudes, they were filled with jealousy, and
contradicted the things which were spoken by Paul, and blasphemed.

 4. Daniel (Greek) 9:10 -> 'law of Moses' (Deuteronomy)
    Similarity Score: 1.000
    Verse: As it is written in the law of Moses, all this evil has come on us.
```

Gambar 4. Contoh Hasil Pencarian Referensi Silang

Terakhir, sistem juga melakukan analisa frekuensi kata melalui fungsi **most_common_words_optimized()**. Sistem menghitung kemunculan kata-kata kapital (yang diasumsikan sebagai nama lokasi atau tokoh) dan menyaring stopwords umum seperti "*the*", "*so*", "*he*", dan sebagainya. Analisis ini berguna untuk mengidentifikasi tokoh dominan dalam narasi Alkitab, serta membantu menafsirkan fokus tematik dalam setiap kitab.

```
# Pre-compile regex for better performance
    word_pattern = re.compile(r'\b[A-Z][a-zA-Z]+\b')

    counter = Counter()

    for i, (_, _, _, sentence) in
enumerate(ayat_list):

        tokens = word_pattern.findall(sentence)
        filtered_tokens = [
            t.lower() for t in tokens
            if t.lower() not in stopwords and len(t)
> 2
        ]
        counter.update(filtered_tokens)

    print("Word frequency analysis complete.")
    return counter.most_common(top_n)
```

Sistem akan mengembalikan 50 nama lokasi atau tokoh yang paling sering muncul dalam keseluruhan teks Alkitab.

```
MOST COMMON NAMES/PLACES
========================================

TOP 50 MOST FREQUENT NAMES/PLACES:
----------------------------------------
Rank Name                 Frequency
----------------------------------------
1    Lord                 8472
2    God                  4903
3    Israel               2836
4    David                1167
5    Jerusalem            1031
6    Jesus                1000
7    Judah                896
8    Moses                893
9    Don                  872
10   Egypt                674
11   Christ               562
12   Jews                 445
13   King                 431
14   Jacob                429
15   Saul                 426
```

Gambar 5. Hasil Frekuensi Nama Lokasi atau Tokoh yang Paling Sering Muncul

IV. KESIMPULAN

Penelitian ini berhasil mengembangkan sebuah sistem berbasis Python untuk menganalisis referensi silang dalam teks Alkitab versi World English Bible (WEB) menggunakan kombinasi algoritma *string matching* seperti Boyer-Moore dan ekspresi regular. Sistem ini mampu melakukan pencarian tematik berdasarkan kata kunci, mendeteksi rujukan eksplisit antar ayat, serta menganalisis frekuensi kata atau nama yang sering muncul dalam Alkitab.

Dalam sistem yang telah dibangun, algoritma Boyer-Moore terbukti efisien dalam mencari kata atau frasa tertentu dalam jumlah ayat yang besar, terutama bila pola yang sicari bersifat tunggal dan eksplisit. Ekspresi regular memberikan fleksibilitas yang baik untuk mendeteksi pola-pola linguistic yang kompleks seperti frasa "*as it is written in [Book]*" atau "*spoken by the prophet [Name]*", yang umum ditemukan dalam bentuk referensi silang dalam teks Alkitab. Tidak hanya itu, kombinasi antara pendekatan berbasis kata kunci tematik dan analisis frekuensi kata mampu mengidentifikasi ayat-ayat yang relevan untuk tema seperti perjanjian, penebusan, dan nubuat, serta menemukan kitab-kitab yang paling banyak menjadi pusat narasi.

Dengan demikian, pendekatan algoritmik yang digunakan dalam penelitian ini menunjukkan potensi besar untuk diaplikasikan dalam konteks studi kitab suci berbasis teks.

V. Kesalahan Umum

Meskipun system yang dikembangkan telah berjalan dengan cukup baik, penulis menyimpulkan bahwa terdapat beberapa kesalahan umum dan keterbatasan yang penting untuk dicatat.

1. Kesalahan parsing struktur file

   Data berbentuk .txt yang digunakan dalam penelitian ini tidak mengikuti format standar, dimana nomor ayat tidak diberikan. Sehingga hasil nomor ayat yang diberikan oleh sistem kemungkinan besar tidak mengikuti format aktual terjemahan Alkitab pada umumnya.

2. Referensi kitab yang ambigu atau tidak eksplisit

   Banyak ayat dalam Alkitab yang merujuk pada peristiwa atau doktrin dari kitab lain tanpa menyebutkan nama kitabnya secara eksplisit. Hal ini mempersempit kemungkinan sistem untuk melakukan referensi silang yang tepat 100%.

3. *False positive* dalam *fuzzy matching*

   Pendekatan fuzzy menggunakan difflib.get_close_matches() terkadang menghasilkan kesalahan karena kemiripan kata yang tidak bermakna secara teologis. Misalnya “Job” bisa dinilai mirip dengan “John”.

VI. Ucapan Terima Kasih

Pertama-tama, penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas rahmat dan bimbingan-Nya selama proses penulisan makalah ini serta pembelajaran mata kuliah IF2211 Strategi Algoritma. Penulis juga ingin mengucapkan terima kasih kepada dosen IF2211 Strategi Algiritma, Bapak Monterico Adrian, yang telah membagikan ilmu dan wawasannya selama proses pembelajaran di kelas.

Penulis juga bersyukur atas segala kasih sayang dan dukungan dari anggota keluarga dan sahabatnya, Karina. Tidak hanya dukungan dan cinta mereka, namun kerja keras serta tekad mereka selalu menginspirasi penulis.

Terakhir, penulis juga ingin menyampaikan rasa terima kasih pada dirinya sendiri karena telah bertahan dan selalu berusaha, bahkan Ketika menghadapi tantangan atau kesulitan.

Lampiran Kode

https://github.com/sammmine/Makalah-Stima-13523138

Referensi

[1] Munir, Rinaldi. 2025. Pencocokan String dan Regular Expression (Regex). https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/stima24-25.htm. Diakses pada Juni 2025.

[2] Geeks For Geeks. 2024. Pattern Searching. https://www.geeksforgeeks.org/pattern-searching/. Diakses pada Juni 2025.

[3] Harrison, Chris. 2007. Bible Cross-References. https://www.chrisharrison.net/index.php/visualizations/BibleViz. Diakses pada Juni 2025.

[4] Ebible. 2025. World English Bible with Deuterocanon. https://ebible.org/find/show.php?id=eng-web. Diakses pada Juni 2025.

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bekasi, 23 Juni 2025

Samantha Laqueenna Ginting
13523138