Program Mesin Turing untuk Permasalahan Verifikasi String dengan Pola Identik Berurut dan Panjang Maksimum 10 Karakter

Rendi Adinata - 10123083
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : 10123083@mahasiswa.itb.ac.id, rendy.adinata88@gmail.com

Makalah ini menyajikan bukti bahwa permasalahan keputusan untuk menentukan string yang memiliki pola identik berurut merupakan permasalahan yang solvable. Untuk memudahkan penulisan, sebut suatu string yang terdiri dari dua substring yang identik secara berurutan adalah self-copy (swa-salin). Selanjutnya, dibuat program Mesin Turing untuk menyelesaikan permasalahan keputusuan tersebut untuk menunjukkan bahwa permasalahan tersebut adalah permasalahan yang solvable. Studi ini diharapkan dapat menginspirasi perancangan program Mesin Turing untuk permasalahan-permasalahan lainnya.

Kata Kunci—Mesin Turing; solvable; Persoalan Keputusan; Identik Berurut;

I. PENDAHULUAN

Dalam dunia komputasi modern, pemahaman terhadap batasan dan kapabilitas dari sistem komputasi merupakan fondasi fundamental dalam pengembangan perangkat lunak dan perangkat keras. Pemikiran tentang bagaimana suatu masalah dapat dipecahkan secara algoritmik telah melahirkan berbagai model komputasi, salah satu yang paling berpengaruh adalah Mesin Turing. Mesin Turing ditemukan oleh Alan Turing pada tahun 1936 dan menjadi model dasar untuk mendefinisikan apa yang dapat dihitung dan seberapa efisien.

Makalah ini berfokus pada eksplorasi program Mesin Turing sebagai model komputasi untuk menyelesaikan sebuah permasalahan keputusan. Permasalahan yang akan dikaji adalah untuk memverifikasi apakah sebuah string input merupakan *string self-copy* yang didefinisikan sebagai string yang terdiri dari dua substring identik yang diletakkan secara berurutan. Ini berarti paruh pertama string harus sama persis dengan paruh kedua. Makalah ini mendemonstrasikan bahwa permasalahan ini adalah permasalahan yang *solvable* (dapat diselesaikan secara algoritmik) dengan merancang program Mesin Turingnya. Hasil analisis lebih lanjut juga meunjukkan bahwa permasalahan ini termasuk ke dalam permasalahan kelas P , yakni permasalahan keputusan yang dapat diselesaikan dalam waktu polinomial.

II. LANDASAN TEORI

A. Mesin Turing

Mesin Turing adalah model matematis dari komputasi yang diajukan oleh Alan Turing pada tahun 1936. Model ini dirancang untuk merepresentasikan secara abstrak cara kerja sebuah komputer. Meskipun konstruksinya sederhana, Mesin Turing memiliki kapabilitas untuk mensimulasikan setiap algoritma komputasi yang dapat dilakukan oleh komputer modern mana pun. Oleh karena itu, Mesin Turing dikenal sebagai model komputasi universal, yang menjadi fondasi bagi studi teori komputabilitas dan kompleksitas.

1. Secara formal, Mesin Turing didefinisikan sebagai tuple $M = (Q, \Sigma, \Gamma, \delta, q0, q_{accept}, q_{reject})$, dengan keterangan dari setiap komponen adalah sebagai berikut:

2. Himpunan State (Q): Koleksi state atau keadaan yang mungkin dimiliki oleh Mesin Turing pada suatu waktu tertentu. Q mencakup state awal (q_0), state penerima (qaccept), dan state penolak (q_{reject}). State merepresentasikan memori internal mesin.

3. Alfabet Input (Σ): Himpunan simbol terbatas yang diizinkan sebagai masukan ke Mesin Turing. Simbol-simbol inilah yang membentuk string input yang akan diproses.

4. Alfabet Tape (Γ): Himpunan simbol yang dapat ditulis atau dibaca dari pita Mesin Turing. Γ selalu menyertakan semua simbol dari Σ ditambah simbol kosong (sering dilambangkan dengan _ atau B untuk blank) dan simbol-simbol lain yang digunakan Mesin Turing sebagai penanda internal.

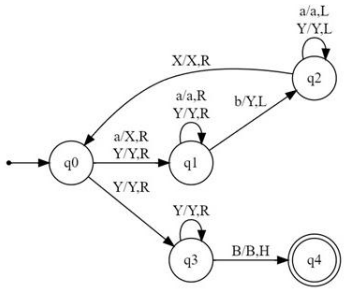
5. Pita (*Tape*): Ini adalah struktur data linear tak terbatas yang dibagi menjadi sel-sel. Setiap sel dapat menyimpan satu simbol dari Γ. Pita berfungsi sebagai memori kerja Mesin Turing, tempat input dibaca, hasil sementara ditulis, dan hasil akhir disimpan.

6. Kepala (*Head*): Objek yang dapat membaca simbol dari sel pita di bawahnya, menulis simbol ke sel tersebut, dan memindahkan dirinya satu sel ke kiri (L) atau ke kanan (R) pada pita.

7. Fungsi Transisi (δ): Ini adalah inti dari "program" Mesin Turing. Fungsi transisi adalah pemetaan yang mendefinisikan aksi Mesin Turing berdasarkan state saat ini ($q \in Q$) dan simbol yang dibaca dari pita ($\gamma \in \Gamma$). Output dari fungsi ini adalah state berikutnya ($q' \in Q$), simbol yang akan ditulis ke pita ($\gamma' \in \Gamma$), dan arah pergerakan kepala ($d \in L, R$). Formalnya,

$$\delta: Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}.$$

8. Ketika Mesin Turing memulai eksekusi, ia berada pada state awal (q_0), dengan kepala di atas simbol pertama dari string input. Mesin kemudian secara berulang menerapkan fungsi transisi berikut: membaca simbol, melakukan aksi (menulis, mengubah state, bergerak), dan terus beroperasi hingga mencapai state q_{accept} (menerima input) atau q_{reject} (menolak input). Jika Mesin Turing selalu berhenti (baik menerima atau menolak) untuk setiap input, maka masalah yang dipecahkannya disebut sebagai masalah yang *solvable* atau *decidable*.



Gambar 1 Proses Pergantian State pada Mesin Turing

Sumber: https://www.tutorialspoint.com/automata_theory/representation_of_turing_machine.htm

Sebuah Mesin Turing deterministik akan memulai pada *state* awal, membaca simbol pada posisi kepala, melakukan aksi sesuai fungsi transisi, dan berpindah *state*. Proses ini berlanjut hingga mesin mencapai *state* penerima atau penolak, atau tidak ada transisi yang didefinisikan untuk *state* dan simbol saat ini.

.

B. Masalah *Solvable* (*Decidable*) dan *Unsolvable* (*Undecidable*)

Konsep *solvable* dan *unsolvable* (juga dikenal sebagai *decidable* dan *undecidable*) adalah permasalahan yang fundamental dalam teori komputabilitas. Sebuah masalah keputusan dikatakan *solvable* atau *decidable* jika ada Mesin Turing yang selalu berhenti (selalu menghasilkan jawaban "ya" atau "tidak" dalam waktu hingga) untuk setiap input yang mungkin. Dengan demikian, ada algoritma yang dapat menyelesaikan masalah tersebut secara tuntas.

Sebaliknya, sebuah masalah dikatakan *unsolvable* atau *undecidable* jika tidak ada Mesin Turing yang dapat menyelesaikannya untuk semua input dalam waktu hingga. Contoh klasik dari masalah *unsolvable* adalah Halting Problem, yang menanyakan apakah sebuah program akan berhenti atau berjalan selamanya untuk input tertentu. Alan Turing membuktikan bahwa tidak ada algoritma (atau Mesin Turing) yang dapat secara universal memutuskan Halting Problem untuk semua pasangan program dan input.

Meskipun Halting Problem menunjukkan adanya batasan terhadap apa yang dapat dihitung, tetapi sebagian besar masalah yang dihadapi dalam praktik komputasi sehari-hari adalah masalah yang *solvable*. Tujuan dari makalah ini adalah untuk secara eksplisit mendemonstrasikan bahwa masalah verifikasi string *self-copy* adalah salah satu dari masalah yang *solvable*.

C. Masalah Tractable dan Intractable

Dalam sub-bidang teori kompleksitas, masalah *solvable* lebih lanjut dikategorikan berdasarkan efisiensinya, yaitu berapa banyak sumber daya komputasi (waktu atau memori) yang dibutuhkan oleh algoritma terbaik untuk menyelesaikannya.

- Masalah Tractable

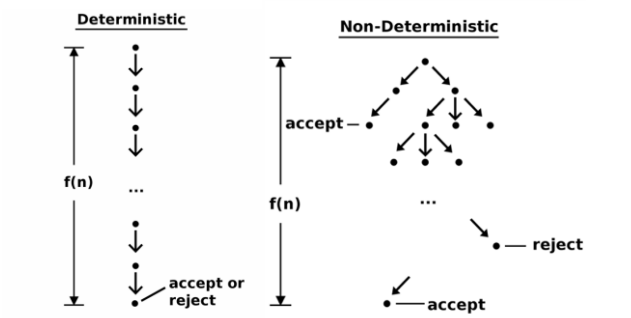
  Sebuah masalah dianggap *tractable* jika dapat diselesaikan oleh Mesin Turing deterministik dalam waktu polinomial. Artinya, waktu komputasi T(N) untuk input berukuran N berbanding lurus dengan N^k untuk suatu konstanta $k \geq 0$. Masalah-masalah di kelas P termasuk dalam kategori ini. Algoritma dengan kompleksitas O(N), $O(N^2)$, $O(N^3)$, dan seterusnya, dianggap *tractable*.

- Masalah Intractable

  Sebuah masalah dianggap *intractable* jika tidak ada algoritma polinomial yang diketahui untuk menyelesaikannya. Umumnya, masalah pada kategori ini merujuk pada masalah yang waktu komputasinya tumbuh secara eksponensial (misalnya, $O(2^N)$ atau O(N!)). Meskipun masalah *intractable* secara teoretis *solvable*, dalam praktiknya, permasalahan dalam kategori ini menjadi tidak layak untuk diselesaikan pada input yang besar karena waktu yang dibutuhkan akan menjadi astronomis. Contohnya adalah Traveling Salesperson Problem (TSP) dan Satisfiability Problem (SAT).

Permasalahan verifikasi *string self-copy* yang dibahas dalam makalah ini akan dibuktikan sebagai masalah *tractable*.

D. *Algoritma Deterministik dan Non-Deterministik*



Gambar 2 Ilustrasi Algoritma Deterministik dan Non Deterministik

Sumber:
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Teori-P-NP-dan-NPC-(Bagian%201).pdf

1. Algoritma Deterministik

   Sebuah algoritma (atau Mesin Turing) disebut deterministik jika pada setiap *state* dan untuk setiap input simbol yang dibaca, hanya ada satu dan hanya satu transisi yang mungkin. Ini berarti jalur komputasi (urutan *state* yang dilewati dan aksi yang dilakukan) sepenuhnya ditentukan oleh input awal. Jika algoritma deterministik dijalankan dua kali dengan input yang sama, akan selalu diperoleh urutan langkah dan hasil yang persis sama. Sebagian besar algoritma yang digunakan sehari-hari pada komputer adalah deterministik. Mesin Turing yang dijelaskan di bagian A, dengan fungsi transisi δ yang memetakan ke satu triple unik, adalah Mesin Turing deterministik.

2. Algoritma Non-Deterministik

   Sebuah algoritma (atau Mesin Turing) disebut non-deterministik jika pada suatu *state* dan untuk suatu input simbol, mungkin ada lebih dari satu transisi yang mungkin. Ini berarti mesin memiliki "pilihan" jalur komputasi yang berbeda untuk dilalui. Mesin Turing non-deterministik (NTM) dapat diilustrasikan sebagai mesin yang mencoba semua kemungkinan jalur komputasi secara paralel. Sebuah NTM dikatakan menerima sebuah input jika setidaknya ada satu jalur komputasi yang mengarah ke *state* penerima. Jika tidak ada jalur yang mengarah ke *state* penerima, input ditolak.

E. *Kelas Kompleksitas P dan NP*

Untuk mengklasifikasikan masalah keputusan berdasarkan efisiensinya, kelas kompleksitas dapat dibagi menjadi dua, yakni P dan NP.

1. Kelas P (*Polynomial Time*)

   Seperti yang telah dibahas, kelas P adalah kumpulan semua masalah keputusan yang dapat diselesaikan oleh Mesin Turing deterministik dalam waktu polinomial. Ini berarti jika sebuah masalah ada di kelas P, dapat ditemukan algoritma yang efisien untuk menyelesaikannya, bahkan untuk input yang besar. Masalah-masalah seperti pengurutan (*sorting*), pencarian (*searching*), dan operasi dasar pada graf (misalnya, mencari jalur terpendek dalam graf tanpa bobot) umumnya berada di kelas P.

2. Kelas NP (*Nondeterministic Polynomial Time*)

   Kelas NP adalah kumpulan semua masalah keputusan yang solusinya dapat diverifikasi oleh Mesin Turing non-deterministik dalam waktu polinomial. Ini berarti, jika diberikan sebuah kandidat jawaban yang diduga merupakan solusi dari masalah NP, dapat diverifikasi apakah bukti tersebut benar dalam waktu polinomial.

Permasalahan verifikasi string *self-copy* yang menjadi fokus makalah ini termasuk dalam kelas P, *solvable* dan *tractable*. Hal ini menunjukkan bahwa Mesin Turing yang dirancang, yang beroperasi secara deterministik, mampu menyelesaikan permasalahan ini.

III. IMPLEMENTASI DAN ANALISIS

A. Permasalahan Keputusan Verifikasi String *Self-Copy*

Permasalahan yang diselesaikan adalah untuk memverifikasi apakah string input S terdiri dari dua salinan identik dari sub-string yang sama yang diletakkan secara berurutan (*Self-Copy*). Secara formal, $S = XX'$, dengan X dan X′ adalah sub-string, dan X=X′. Namun, ini mengimplikasikan bahwa panjang string N harus genap, dan panjang sub-string X serta X′ adalah N/2. String input diasumsikan biner (mengandung hanya 0 dan 1) dengan panjang maksimum 10 karakter.

Contoh Input:
Diterima: 00, 001001, 10111011
Ditolak: 01, 00100, 10111101

B. Solusi Algoritmik Klasik

Permasalahan ini dapat diselesaikan dengan algoritma klasik dalam waktu O(N). Langkah-langkahnya meliputi pemeriksaan panjang string (jika ganjil, tolak), kemudian membagi string menjadi dua paruh, dan membandingkan setiap karakter pada paruh pertama dengan karakter yang sesuai pada paruh kedua. Jika semua karakter cocok, string diterima, jika tidak, string ditolak. Kompleksitas O(N) menempatkan masalah dalam kelas kompleksitas P.

C. Perancangan Mesin Turing

Mesin Turing yang dirancang didefinisikan sebagai tupel $M = (Q, \Sigma, \Gamma, \delta, q0, q_{accept}, q_{reject})$,, dengan komponen sebagai berikut:

- Q: Himpunan *state* (keadaan) Mesin Turing. Ini mencakup *state* untuk inisialisasi, penentuan panjang, perbandingan karakter, serta *state* penerima dan penolak.
- Σ={0,1}: Alfabet input, yang terdiri dari simbol biner.
- Γ={0,1,'L','R','X'}: Alfabet *tape*, meliputi simbol input (0, 1), simbol batas kiri (L), simbol batas kanan (R), dan simbol penanda (X). Simbol X digunakan untuk menandai karakter pada *tape* yang telah diproses atau dibandingkan.
- q0=1: *State* awal Mesin Turing. Setiap eksekusi selalu dimulai dari *state* ini.
- q_{accept}: Berbagai *state* penerima yang dicapai setelah verifikasi berhasil, seperti 14 atau 15 (untuk panjang 2), 16 (untuk panjang 4), 21 (untuk panjang 6), 28 (untuk panjang 8), dan 37 (untuk panjang 10).
- q_{reject}: Berbagai *state* penolak yang dicapai jika string tidak memenuhi kriteria, termasuk *state* 2 (untuk string kosong), 3, 5, 7, 9, 11 (untuk panjang ganjil), dan *state* tertentu lainnya jika ditemukan ketidakcocokan pola.

Selanjutnya, *Tape* Mesin Turing merepresentasikan string input. Pada awal eksekusi, string input ditempatkan di antara dua simbol batas khusus, yaitu L di sisi paling kiri dan R di sisi paling kanan. Struktur *tape* adalah sebagai berikut:

L | S_1 | S_2 | ... | S_N | R

Kepala Mesin Turing (pos) akan selalu berada pada salah satu sel *tape*, dan akan bergerak LEFT atau RIGHT sesuai instruksi fungsi transisi.

Mesin Turing yang diimplementasikan menggunakan pendekatan berbasis urutan *state* spesifik dengan setiap kemungkinan panjang string genap (2, 4, 6, 8, 10 karakter) memiliki serangkaian aturan transisi yang telah ditentukan secara eksplisit. Strategi ini memverifikasi apakah string input merupakan *string self-copy*.

Tahap Inisialisasi dan Penentuan Panjang String (State 1-12)

Pada awal eksekusi, kepala Mesin Turing berada pada simbol batas kiri L dengan *state* awal 1.

- Langkah Awal: Mesin Turing akan membaca simbol L.
  - Aturan: ["L", 1, "L", 2, "RIGHT"]
  - Aturan ini bermakna kepala Mesin Turing membaca L, menulis ulang L (tetap), berpindah ke *state* 2, dan bergerak satu posisi ke kanan, memasuki area string input.
  - Ilustrasi Tape Awal: [L] S_1 S_2 ... S_N R (Kepala di L, *State* 1)
  - Setelah Langkah Awal: L [S_1] S_2 ... S_N R (Kepala di S_1, *State* 2)

Dari *state* 2, Mesin Turing akan bergerak secara sekuensial ke kanan, melewati setiap simbol input (0, 1, atau X jika sudah ditandai). Untuk setiap langkah ke kanan, *state* Mesin Turing akan meningkat secara berurutan (dari i+2 ke i+3). Proses ini terus berlanjut hingga kepala mencapai simbol batas R. *State* Mesin Turing saat itu secara implisit akan mengindikasikan panjang string. Berikut adalah aturan umum pergerakan kanan:

["0", i+2, "0", i+3, "RIGHT"]

["1", i+2, "1", i+3, "RIGHT"]

["X", i+2, "X", i+3, "RIGHT"]

untuk state i dari 0 hingga 9.

Setelah panjang string diidentifikasi melalui nilai *state* yang disimpan pada tahap sebelumnya, Mesin Turing akan membuat keputusan penolakan jika string tidak memenuhi syarat awal:

- String Ganjil: Jika kepala Mesin Turing mencapai R dan *state* saat itu adalah *state* ganjil (3, 5, 7, 9, 11), yang berarti string memiliki panjang ganjil (1, 3, 5, 7, 9), string tersebut segera ditolak. Ini adalah implementasi dari aturan:

  ["R", 2*i+3, "R", 2*i+3, "REJECT"]
  untuk *state* i dari 0 hingga 4.

- String Kosong (""): Jika setelah membaca L (di *state* 2), kepala langsung bertemu R (menandakan panjang 0), string tersebut ditolak. Aturan yang berlaku adalah

  ["R", 2, "R", 2, "REJECT"].

Untuk setiap string dengan panjang genap (2, 4, 6, 8, 10), Mesin Turing akan melakukan serangkaian perbandingan untuk memverifikasi pola *self-copy*. Mekanisme ini dirancang untuk membandingkan karakter pada paruh pertama string ($S[i]$) dengan karakter yang sesuai pada paruh kedua ($S[i + N/2]$). Prosedur ini dilakukan dengan secara bolak-balik antara kedua paruh string.

- Tahap Inisiasi Perbandingan (Contoh: String Panjang 6)
    - Jika string memiliki panjang genap (misalnya, kepala mencapai R di *state* 8 untuk string panjang 6), Mesin Turing akan bergerak ke kiri dari simbol R untuk memulai proses perbandingan.
    - Aturan: ["R", 8, "R", 21, "LEFT"]. Kepala bergerak ke kiri dari R, masuk ke *state* 21.
    - Dari *state* 21, Mesin Turing akan memulai pencarian karakter terakhir dari paruh kedua string .
- Siklus Perbandingan
  Proses ini mengulang untuk setiap pasangan karakter yang perlu dibandingkan (N/2 kali).
    - Menandai Karakter dari Paruh Kedua.
        - Mesin Turing membaca karakter di posisi kepala saat ini (yaitu, karakter paling kanan dari paruh kedua yang belum ditandai)
        - Simbol ini kemudian diubah menjadi X pada *tape*.
        - Mesin Turing beralih ke *state* tertentu (22 untuk 0, 23 untuk 1) yang mengindikasikan simbol yang baru saja ditandai dan akan mencari pasangannya di paruh pertama. Kepala Mesin Turing kemudian bergerak ke kiri.
        - Contoh: (Panjang 6, Membaca S_6):
            - Jika S_6 adalah 0: ("0", 21, "X", 22, "LEFT").
            - Jika S_6 adalah 1: ("1", 21, "X", 23, "LEFT").
        - Ilustrasi (001001 setelah menandai S_6): L | 0 | 0 | 1 | 0 | 0 | [X] | R (Kepala di S_5, *State* 22 atau 23)
    - Navigasi ke Karakter yang Berpasangan di Paruh Pertama.
        - Dari *state* yang menyimpan informasi karakter dari paruh kedua (misal *state* 22 atau 23), kepala Mesin Turing akan terus bergerak ke kiri. Pergerakan ini melewati sisa simbol di paruh kedua (jika ada) dan seluruh simbol di paruh pertama yang belum ditandai (bukan X). Urutan *state* yang spesifik untuk setiap panjang string akan memandu navigasi ini hingga kepala mencapai karakter yang berpasangan di paruh pertama (contoh, S_3 untuk S_6).
        - Aturan seperti ("0", state_sebelumnya, "0", state_berikutnya, "LEFT") digunakan untuk melewati simbol tanpa mengubahnya, hanya berpindah *state* untuk menandai posisi relatif.
        - Ilustrasi: L | 0 | 0 | [S_3] | S_4 | S_5 | X | R (Kepala di S_3, *state* yang sesuai, siap membandingkan)
    - Membaca dan Membandingkan Karakter dari Paruh Pertama.
        - Setelah mencapai karakter di paruh pertama yang berpasangan, Mesin Turing membaca simbol tersebut.
        - Jika Cocok: Simbol di paruh pertama juga diubah menjadi X pada *tape*. Mesin Turing kemudian kembali ke *state* yang sesuai untuk memulai siklus perbandingan berikutnya, bergerak ke kanan untuk menemukan pasangan berikutnya yang belum ditandai di paruh kedua.
            - Contoh: (Panjang 6, Membandingkan S_3 dengan S_6): Jika Mesin Turing (misalnya dari *state* 27 yang menyimpan informasi bahwa S_6 adalah 1) bertemu 1 pada S_3: ("1", 27, "X", 23, "LEFT"). S_3 akan menjadi X, dan Mesin Turing kembali ke *state* 23 untuk mencari pasangan berikutnya.
        - Jika tidak cocok: Mesin Turing langsung berpindah ke *state* REJECT. Ini terjadi ketika karakter yang dibaca di paruh pertama tidak sesuai dengan karakter yang ditandai dari paruh kedua.

        - Contoh (Panjang 6): ("0", 27, "X", 23, "REJECT") (jika karakter S_3 adalah 0 namun seharusnya 1, tolak).

    - Kembali untuk Pasangan Berikutnya.

        - Setelah perbandingan dan penandaan X pada kedua pasangan karakter (jika cocok), kepala Mesin Turing akan bergerak ke kanan untuk menemukan karakter berikutnya yang belum ditandai di paruh kedua dan mengulang siklus perbandingan. Proses ini berlanjut hingga semua pasangan karakter telah dibandingkan.

- Penerimaan Akhir (ACCEPT State):

    - Jika semua pasangan karakter berhasil dibandingkan dan cocok tanpa ada penolakan, Mesin Turing akan terus menandai karakter hingga seluruh string (kecuali simbol L dan R) menjadi X.

    - Mesin Turing kemudian akan kembali ke simbol batas L dan akhirnya akan berpindah ke *state* ACCEPT.

    - Contoh: ("L", 21, "L", 21, "ACCEPT") untuk string panjang 6 yang sepenuhnya cocok. Demikian pula untuk panjang lainnya seperti ("L", 16, "L", 16, "ACCEPT") untuk panjang 4, dan seterusnya.

E. METODE PENGUJIAN

Pengujian dilakukan dengan memasukkan berbagai string biner (terdiri dari 0 dan 1) dengan panjang antara 0 hingga 10 karakter. String-string ini dipilih untuk merepresentasikan kasus yang diharapkan diterima (identik berurut) dan ditolak (tidak identik berurut). Hasil keluaran (*True* untuk diterima, *False* untuk ditolak) dicatat dan dibandingkan dengan hasil yang diharapkan.

F. HASIL PENGUJIAN

String Input	Panjang (N)	Keluaran Harapan	Keluaran Program
""	0	False	False
0	1	False	False
1	1	False	False
00	2	True	True
11	2	True	True
01	2	False	False
00100	5	False	False
001001	6	True	True
10111011	8	True	True
0110010011	10	False	False
1001110011	10	True	True

Tabel 1 Hasil Pengujian Program Mesin Turing

G. ANALISIS HASIL

Berdasarkan hasil pengujian pada Tabel I, Mesin Turing yang dirancang dan diimplementasikan dalam berkas turing_tape.py secara konsisten dan akurat memverifikasi pola string *self-copy*. Mesin Turing berhasil menerima semua input yang merupakan string *self-copy* (misalnya 00, 11, 0000, 001001, 11001100, 0000000000) dan menolak semua input yang bukan string *self-copy* (misalnya 01, 0010, 1001, 0110, 00100, 011110, 101010, 1001110101, serta semua string berpanjang ganjil).

Hal ini menunjukkan bahwa Mesin Turing yang diimplementasikan telah berhasil memecahkan permasalahan keputusan string yang *self-copy*. Waktu eksekusi yang diperoleh juga cukup singkat (kurang dari 0.000001 detik untuk semua kasus uji) menunjukkan bahwa program Mesin Turing yang telah dibuat cukup efisien. Penggunaan *state* dan transisi yang diatur secara spesifik untuk setiap panjang string memastikan ketepatan Mesin Turing dalam memverifikasi pola string *self-copy* pada batasan masalah yang diberikan.

Perancangan Mesin Turing dengan pendekatan berbasis urutan *state* spesifik ini, meskipun membutuhkan banyak aturan yang ditentukan secara eksplisit untuk setiap panjang, tetapi hasil yang diperoleh tetap memiliki nilai signifikan dalam konteks studi teoretis. Hasil ini membuktikan bahwa permasalahan verifikasi *self-copy* adalah *solvable* menggunakan model Mesin Turing. Pendekatan ini memberikan pemahaman yang lebih mendalam tentang bagaimana aturan transisi dapat disusun untuk memprogram Mesin Turing pada input dan mencapai tujuan komputasi tertentu.

IV. KESIMPULAN DAN SARAN

Berdasarkan hasil yang diperoleh, telah terbukti bahwa permasalahan keputusan apakah suatu string termasuk string *self-copy* untuk panjang maksimal 10 karakter merupakan permasalahan yang *solvable* karena terdapat Mesin Turing yang dapat menyelesaikannya dan *tractable* karena dapat diselesaikan dalam waktu yang wajar.

Penulis mengakui adanya keterbatasan intelektual dalam menyelesaikan permasalahan ini secara umum untuk string dengan panjang berapapun sehingga program Mesin Turing yang disajikan hanya berlaku untuk panjang maksimum 10 karakter. Namun demikian, penulis berharap hasil yang disajikan dapat bermanfaat untuk mendemonstrasikan proses perancangan Mesin Turing untuk menyelesaikan suatu permasalahan keputusan. Penulis berharap semoga kedepannya dapat ditemukan solusi untuk

V. PENUTUP

Dengan penuh rasa terima kasih, penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas berkat rahmat dan karunia-Nya, makalah “Program Mesin Turing untuk Permasalahan Verifikasi String dengan Pola Identik Berurut dan Panjang Maksimum 10 Karakter” dapat diselesaikan tanpa halangan yang berarti. Penulis juga berterima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. , Ibu Dr. Nur Ulfa Maulidevi, dan Bapak Dr. Ir. Rila Mandala selaku dosen pengampu matakuliah IF2211-Strategi Algoritma yang sudah memberikan materi dan ilmu untuk pembelajaran bagi penulis. Tidak lupa juga, penulis berterima kasih kepada seluruh pihak baik keluarga, teman, dan kakak tingkat yang sudah memberikan dukungan untuk menyelesaikan makalah ini. Penulis berharap makalah ini dapat membantu untuk pembelajaran dan penelitian terkait Mesin Turing dalam Teori Komputasi.

LAMPIRAN

Tautan *video:*

https://youtu.be/kOOSKA4wG5g

Tautan repository:
xinuzo/Turing_Machine_Repeated_String

REFERENSI

[1] R. Munir, "Algoritma Divide and Conquer bagian 1”, materi kuliah, Institut Teknologi Bandung, 2025. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Teori-P-NP-dan-NPC-(Bagian%201).pdf. Diakses pada 20 Juni 2025.

[2] R. Munir, "Teori P, NP, dan NPC (Bagian 2),” materi kuliah, Institut Teknologi Bandung, 2025. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2019-2020/Teori-P-NP-dan-NPC-(Bagian%202).pdf. Diakses pada 24 Juni 2025.

[3] Juha Kärkkäinen. "Data Structures and Algorithm spring 2025", modul, University of Helsinki, 2025. Tersedia: https://tira.mooc.fi/spring-2025/. Diakses pada 24 Juni 2025.

[4] GeeksforGeeks. "Types of Complexity Classes P, NP, CoNP, NP-Hard and NP-Complete", artikel, GeeksforGeeks, 2025. Tersedia: https://www.geeksforgeeks.org/types-of-complexity-classes-p-np-conp-np-hard-and-np-complete/. Diakses pada 24 Juni 2025.

[5] Brilliant. "Turing Machines", Brilliant, 2025. Tersedia: https://brilliant.org/wiki/turing-machines/. Diakses pada 24 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.

Bandung, 24 Juni 2025

Rendi Adinata