Analisis Perbandingan Waktu Eksekusi *de Brujin Sequence Constructions* dengan *Greedy Algorithm* dan *Euler Cycle*

Christian Albert Hasiholan - 13521078
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521078@std.stei.itb.ac.id

***Abstract*—De Bruijn sequence adalah sekuens yang terdiri atas beberapa angka atau karakter. Sekuens ini special karena setiap kombinasi dari n angka atau karakter tersebut pasti muncul sebanyak 1 kali pada sekuens itu. Oleh karena itu sekuens itu cukup banyak digunakan dalam banyak hal, khususnya dalam encode/decode position. Untuk membuat sekuens ini sendiri ada beberapa cara yang dapat dilakukan seperti brute force, greedy, depth first search, dan masih banyak lagi. Namun dengan semakin banyaknya kombinasi, maka sekuens yang perlu dibuat semakin besar dan tentunya memerlukan time and space yang lebih. Oleh karena itu pada makalah ini, akan dilakukan analisis efisiensi antara greedy algorithm dan DFS untuk mengenerate de Brujin sequence. Selain itu akan dilakukan juga pembuktian apakah de Bruijn sequence yang digenerate telah benar dengan pattern matching.**

Keywords—de Bruijn sequence; kombinasi; greedy algorithm; DFS; pattern matching;

I. PENDAHULUAN

Sekuens merupakan urutan atau rangkaian dari beberapa elemen yang urutannya diatur berdasarkan aturan tertentu. Elemen-elemen yang menyusun sebuah sekuens dapat berupa angka, huruf, karakter, atau objek-objek lainnya. Sekuens sering kali dinyatakan sebagai urutan elemen yang diindeks oleh bilangan bulat non-negatif. Misalnya sekuens (a1, a2, a3), dimana a1 menyatakan elemen pertama, a2 elemen kedua, dan seterusnya.

Selain itu sekuens dapat dinyatakan juga sebagai bentuk rekursif. Artinya elemen-elemen selanjutnya pada sekuens dihasilkan dari aturan atau rumus dengan elemen sebelumnya pada sekuens. Contoh dari hal ini dapat ditemukan pada deret Fibonacci, dimana suatu angka pada deret tersebut merupakan hasil penjumlahan dari 2 angka sebelumnya.

Sekuens banyak digunakan dalam berbagai bidang, misalnya pada pemrosesan teks, kriptografi, dan bioinformatika. Pada pemrosesan teks, sekuens digunakan untuk menggabungkan, momotong, pattern search dan operasi lainnya yang berhubungan dengan teks. Lalu pada kriptografi, sekuens dapat digunakan untuk melakukan enkripsi dan dekripsi. Kemudian dalam bioinformatika, sekuens seringkali digunakan untuk merepresentasikan urutan RNA dan DNA untuk dianalisis serta perakitan genom. Selain pada bidang-bidang tersebut, sekuens juga dapat digunakan pada permainan, seperti pada catur dan trik kartu.

Suatu permasalahan pernah diangkat pada tahun 1894 di France scientific journal, *L'Intermédiaire des mathématiciens.* Pada jurnal tersebut dipertanyakan apakah ada sekuens siklik berukuran 2^n yang terdiri atas 0 dan 1, yang mengandung semua 2^n sekuens biner dengan n digit. Permasalahan ini kemudian berhasil dibuktikan oleh seorang matematikawan asal Belanda yang bernama Nicolaas Govert de Bruijn.

Gambar 1.1. Nicolaas Govert de Bruijn
Sumber: *Wikipedia (https://en.wikipedia.org/wiki/Nicolaas_Govert_de_Bruijn)*

Permasalahan lain yang dihadapi ini berkaitan dengan kunci kombinasi misalnya pada sebuah pintu. Pada pintu tersebut hanya dapat dibuka oleh sebuah PIN yang terdiri dari n digit dan kita tanpa mengetahui PINnya ingin membuka pintu tersebut. Kunci ini hanya akan memeriksa n digit terakhir yang diinput pengguna tanpa perlu menunggu konfirmasi. Jika terdapat 4 digit dengan input hanya dapat berupa angka 0-9, maka diperlukan 40000 angka ditekan untuk mencoba 10000 kombinasi angka. Tentu hal ini memerlukan waktu yang cukup lama, apalagi bila memerlukan digit yang lebih banyak, jadi apakah ada cara yang lebih efisien?

Mungkin sekilas tidak dapat dilihat keterkaitan antara kedua masalah ini, namun jika dipahami maka kedua masalah ini sangat mengandalkan urutan pada sebuah sekuens. Urutan yang baik dapat membuat sekuens menjadi lebih efisien namun dengan tetap menjaga fungsi dan tujuan sekuens ini.

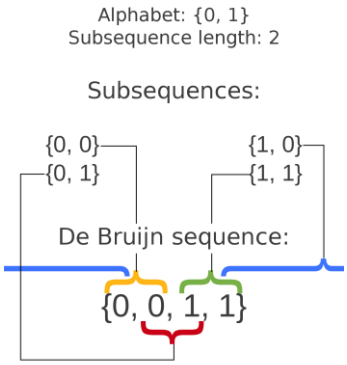
Salah satu cara yang bisa dilakukan adalah dengan membuat de Bruijn sequence. Terdapat beberapa cara untuk membuat sekuens ini, namun dengan semakin banyak jumlah elemen dan jumlah digit maka kombinasi yang diciptakan juga akan semakin banyak. Dengan demikian sekuens yang perlu dibuat juga akan semakin Panjang. Oleh karena itu, akan dilakukan perbandingan cara apa yang lebih cepat digunakan untuk membangkitkan de Bruijn sequence.

II. Dasar Teori

A. *De Bruijn sequence*

De Bruijn sequence adalah sekuens spesial yang berkaitan juga dengan graf dan kombinatorial. Sekuens ini mengandung seluruh kombinasi dari n digit elemen dan hanya muncul 1 kali saja. Sekuens ini dapat digolongkan sebagai sekuens rekursif karena elemen baru yang ditambahkan perlu memenuhi aturan, yaitu tidak boleh membentuk kombinasi yang sama dengan kombinasi yang telah ada pada sekuens tersebut.

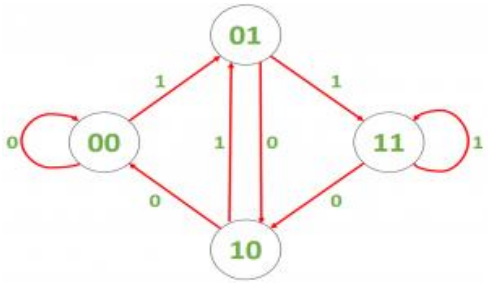
Untuk membuat sekuens ini terdapat 2 parameter yang harus dipenuhi yaitu k sebagai jumlah elemen, seperti angka atau karakter, dan n sebagai banyak digit atau panjang dari kombinasi elemen k. Contohnya dengan k = 2 dan n = 2 dengan elemen k {0, 1}, maka kombinasi 2 digit yang mungkin adalah 00, 01, 10, dan 11. Oleh karena itu sekuens yang mengandung semua kombinasi tersebut adalah 0011.



Gambar 2.1. de Bruijn sequence
Sumber: *Wikipedia (https://en.wikipedia.org/wiki/De_Bruijn_sequence)*

Perlu diingat juga bahwa sekuens ini siklik, yang artinya elemen di akhir selalu terhubung dengan elemen awal. Oleh karena itulah kombinasi 10 ada pada sekuens tersebut.

Contoh lainnya dengan k = 2 dan elemen k {0,1}, namun dengan n = 3. Kombinasi-kombinasi yang mungkin adalah 000,001,010,011,100,101,110,111. Maka sekuens de Bruijn yang mengandung seluruh kombinasi tersebut adalah 00010111.



Gambar 2.2. de Bruijn graph
Sumber: *Geeksforgeeks (https://www.geeksforgeeks.org/de-bruijn-sequence-set-1/)*

Sekuens de Bruijn juga dapat dibentuk dari graf dengan mencari sirkuit euler atau sirkuit Hamilton berdasarkan node pada graf seperti gambar di atas. Dapat diperhatikan bahwa sekuens yang diperoleh lebih efisien dan pendek daripada harus menggabungkan semua kombinasi yang ada menjadi 1 sekuens.

B. *Kombinasi*

Kombinasi adalah sebuah bentuk khusus dari permutasi. Perbedaan kombinasi dengan permutasi terdapat pada urutan kemunculan objeknya. Pada permutasi urutan kemunculan objek diperhitungkan, sedangkan pada kombinasi urutan kemunculan objek diabaikan, sehingga bila urutannya berbeda namun banyak kemunculan objek sama maka dianggap sebagai susunan yang sama.

Pada kombinasi, untuk menentukan jumlah pemilihan r elemen dari n buah elemen maka dapat digunakan rumus berikut

$$C(n,r) = \frac{n(n-1)(n-2)\dots\big(n-(r-1)\big)}{r!} = \frac{n!}{r!\,(n-r)!}$$

C. *Graf*

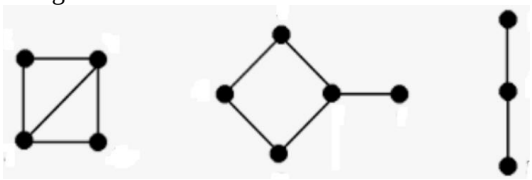
Graf adalah kumpulan simpul yang dihubungkan oleh sisi. Graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Secara formal, graf (G) didefinisikan sebagai $G = (V, E)$ yang dalam hal ini:

V = himpunan tidak kosong dari simpul-simpul (*vertices*)
$= \{v_1, v_2, \dots, v_n\}$
E = himpunan sisi (*edges*) yang menghubungkan simpul
$= \{e_1, e_2, \dots, e_n\}$

Berdasarkan ada dan tidaknya gelang atau sisi ganda pada suatu graf, maka graf dapat digolongkan menjadi 2 jenis, yaitu

1. Graf sederhana (*simple graph*)

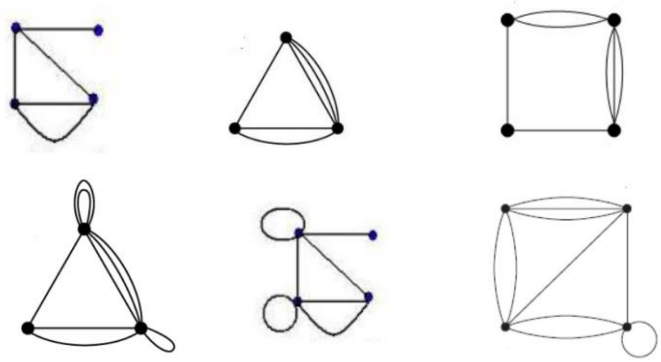
Graf sederhana adalah graf yang tidak memiliki gelang maupun sisi ganda.



Gambar 2.3. Graf Sederhana
Sumber: Slide Bahan Kuliah Graf (Bag 1)

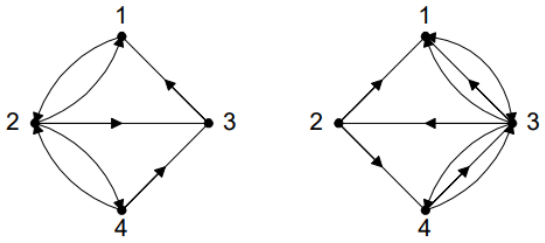
2. Graf tak-sederhana (*unsimple graph*)

Kebalikannya dari graf sederahana, graf tak sederhana memiliki sisi ganda atau sisi gelang. Graf tak sederhana juga dapat digolongkan kembali berdasarkan apakah graf tersebut memiliki sisi ganda atau sisi gelang. Graf yang mengandung sisi ganda disebut graf ganda (*multi-graph*), sedangkan graf yang mengandung sisi gelang disebut graf semu (*pseudo-graph*).



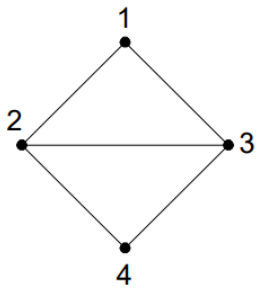
Gambar 2.4. Graf Tak Sederhana
Sumber: Slide Bahan Kuliah Graf (Bag 1)

Berdasarkan orientasi arahnya, graf juga dapat dibedakan menjadi dua jenis, yaitu graf tak berarah (*undirected graph*) dan graf berarah (*directed graph* atau *digraph*). Contoh dari graf berarah adalah sebagai berikut :



Gambar 2.5. Graf Berarah
Sumber: Slide Bahan Kuliah Graf (Bag 1)

Pada graf, terdapat beberapa terminologi, seperti lintasan dan sirkuit. Lintasan adalah barisan sisi yang menghubungkan suatu simpul ke simpul lainnya. Sedangkan sirkuit adalah lintasan yang berawal dan berakhir pada simpul yang sama. Pada graf di bawah ini lintasan 1, 2, 4, 3 adalah lintasan dengan barisan sisi (1,2), (2,4), (4,3), sedangkan 1, 2, 3, 1 adalah sebuah sirkuit.



Gambar 2.6. Contoh Graf
Sumber: Slide Bahan Kuliah Graf (Bag 1)

Lintasan Euler ialah lintasan yang melalui masing-masing sisi di dalam graf tepat satu kali. Sirkuit Euler ialah sirkuit yang melewati masing-masing sisi tepat satu kali. Graf yang mempunyai sirkuit Euler disebut graf Euler (Eulerian graph). Graf yang mempunyai lintasan Euler dinamakan juga graf semi-Euler (semi-Eulerian graph).

Lintasan Hamilton ialah lintasan yang melalui tiap simpul di dalam graf tepat satu kali. Sirkuit Hamilton ialah sirkuit yang melalui tiap simpul di dalam graf tepat satu kali, kecuali simpul asal (sekaligus simpul akhir) yang dilalui dua kali. Graf yang memiliki sirkuit Hamilton dinamakan graf Hamilton, sedangkan graf yang hanya memiliki lintasan Hamilton disebut graf semi-Hamilton.

D. Greedy Algorithm

Algoritma greedy adalah algoritma yang memecahkan persoalan secara langkah per langkah (step by step) sedemikian sehingga, pada setiap langkah:

1. mengambil pilihan yang terbaik yang dapat diperoleh pada saat itu tanpa memperhatikan konsekuensi ke depan (prinsip “take what you can get now!”)
2. dan “berharap” bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.

Elemen-elemen algoritma greedy:

1. Himpunan kandidat, C : berisi kandidat yang akan dipilih pada setiap Langkah (misal: simpul/sisi di dalam graf, job, task, koin, benda, karakter, dsb)

2. Himpunan solusi, S : berisi kandidat yang sudah dipilih

3. Fungsi solusi: menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi

4. Fungsi seleksi (selection function): memilih kandidat berdasarkan strategi greedy tertentu. Strategi greedy ini bersifat heuristik.

5. Fungsi kelayakan (feasible): memeriksa apakah kandidat yang dipilih dapat dimasukkan ke dalam himpunan solusi (layak atau tidak)

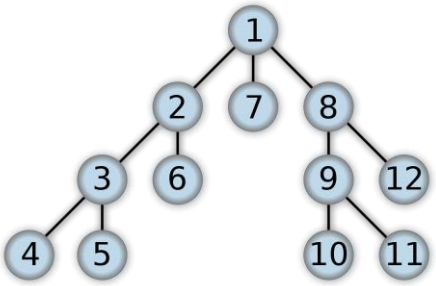
6. Fungsi obyektif : memaksimumkan atau meminimumkan

Dengan menggunakan elemen-elemen di atas, maka dapat dikatakan bahwa algoritma greedy melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu S menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.

E. DFS(Depth First Search)

Graf dapat dianggap sebagai representasi suatu persoalan, maka untuk memecahkan persoalan itu digunakanlah algoritma traversal graf. Algoritma traversal graf bertujuan untuk mengujungi simpul-simpul pada graf secara sistematik. Terdapat beberapa algoritma, 2 diantaranya adalah Breadth First Search (BFS) dan Depth First Search (DFS).

DFS disebut sebagai pencarian mendalam karena simpul yang bertetangga dengan simpul saat ini akan ditelusuri dahulu semuanya sampai tidak ada simpul tetangga yang belum dikunjung lagi. Setelah itu akan dilakukan backtracking ke simpul terakhir yang dikunjungi yang masih memiliki simpul tetangga yang belum dikunjungi. Penelusuran akan berakhir apabila tidak ada lagi simpul yang belum dikunjungi yang dapat dicapai dari simpul yang telah dikunjungi.

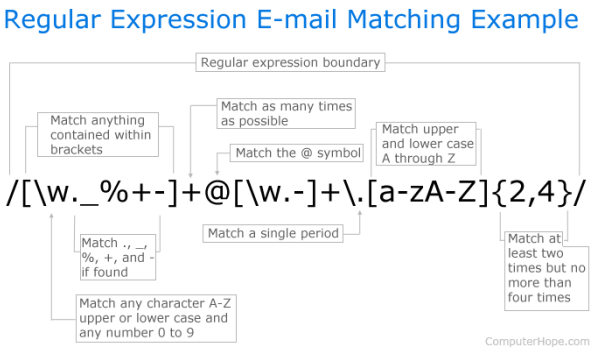


Gambar 2.7. Depth First Search
Sumber: *Wikipedia (https://en.wikipedia.org/wiki/Depth-first_search)*

F. Regex

Regex adalah pola yang dapat digunakan untuk melakukan matching pada suatu string. Tujuan dari matching ini adalah untuk mencari dan mencocokan string yang kita inginkan.

Regex juga merupakan salah satu metode untuk melakukan pattern matching. Pattern matching adalah pencarian atau pencocokan pattern pada sebuah teks yang jauh lebih besar dari pattern yang dicari.



Gambar 2.8. Regex
Sumber: *Computer Hope (https://www.computerhope.com/jargon/r/regex.htm)*

III. ALGORITMA

Di bawah ini merupakan algoritma dalam membuat de Bruijn sequence. Terdapat 2 metode yang akan digunakan, yang pertama dengan greedy algorithm, dan yang kedua dengan backtrack untuk mencari sirkuit Hamilton pada de Bruijn graph. Selain itu juga terdapat algoritma untuk melakukan pengecekan, apakah sekuens yang dihasilkan telah mengandung semua kombinasi atau tidak.

A. Greedy Algorithm

Greedy algorithm yang akan digunakan untuk membuat de Bruijn sequence ini adalah prefer-largest greedy construction. Langkah-langkahnya adalah:

1. Buat string dengan karakter pertama sebanyak n-1 digit.
2. Lakukan penambahan karakter dari karakter terakhir atau terbesar pada string, lakukan juga pengecekan apakah kombinasi telah ada pada string.
3. Ulangi langkah kedua hingga tidak ada karakter lagi yang bisa ditambahkan.
4. Hapus n-1 digit awal pada string hasil.

Berikut ini merupakan code yang telah dibuat untuk membuat sekaligus mengecek kemunculan kombinasi.

```
def countOccurance(text, pattern):
    regex = "(?=" + pattern + ")"
    occurance = [m.start() for m in re.finditer(regex,
text)]

    return len(occurance)

def sequenceWithGreedy(k, n):
    sequence = k[0] * (n-1)
    added = 0
```

```
    k = k[::-1]
    while added < math.pow(len(k), n):
        for c in k:
            pattern = sequence[-n+1:] + c
            if countOccurance(sequence, pattern) == 0:
                sequence += c
                added += 1
                break

    sequence = sequence[n-1:]
    return sequence
```

Dapat diketahui bahwa panjang de Bruijn sequence seharusnya adalah k^n, sehingga karakter yang ditambahkan pada string sekuens seharusnya juga tidak melebihi itu.

B. Graph Traversal with DFS

Seperti yang telah dijelaskan pada teori de Bruijn sequence bahwa sekuens ini dapat dicari dengan menelusuri graf de Bruijn. Pada graf ini akan ditelusuri sirkuit Eulernya dengan metode DFS secara rekursif. Langkah-langkahnya adalah:

1. Buat node awal, yaitu n-1 digit karakter pertama
2. Telusuri edge yang belum pernah dilewati dan catat edge tersebut setelah melewatinya.
3. Ulangi penelusuran sampai semua edge telah dikunjungi.
4. Simpan value dari edge yang dikunjungi, dari yang terakhir dikunjungi hingga pertama dikunjungi.
5. Gabungkan menjadi 1 string hasil.

Berikut merupakan code untuk menelusuri siklus Euler pada de Bruijn graph dengan DFS.

```
visited = set()
result = []

def dfs(node, k):
    for c in k:
        combination = node + c
        if combination not in visited:
            visited.add(combination)
            dfs(combination[1:], k)
            result.append(c)


def sequenceWithDFS(k, n):
    start = k[0] * (n-1)
    dfs(start, k)
    sequence = ''.join(result)

    return sequence
```

C. Checking with Regex Pattern Match

Untuk mengecek apakah sekuens yang dihasilkan mengandung semua kombinasi maka digunakanlah algoritma pattern matching dengan regex. Regex dipilih karena kemudahannya untuk mencari banyak kemunculan suatu pattern pada teks dibandingkan dengan metode lainnya. Langkah-langkah pengecekannya adalah:

1. Tempel n-1 digit pertama pada sekuens hasil ke akhir sekuens, mengingat sekuens de Bruijn adalah sekuens siklik.
2. Tentukan semua kombinasi yang mungkin dibuat.
3. Lakukan pencarian kombinasi pada sekuens dan hitung banyak kemunculannya.
4. Ulangi pencarian sampai semua kombinasi ada pada sekuens atau sampai ada kemunculan kombinasi yang tidak muncul sekali.

Berikut merupakan code yang dibuat untuk melakukan pengecekan.

```
def getAllCombination(k, n):
    comb = itertools.product(k, repeat=n)
    list_of_combination = [''.join(i) for i in comb]
    return list_of_combination


def checkSequence(sequence, k, n):
    cycle_sequence = sequence + sequence[:n-1]
    list_of_combination = getAllCombination(k, n)
    for combination in list_of_combination:
        if countOccurance(cycle_sequence, combination) !=
1:
            return False
    return True
```

D. Main Program

```
if __name__ == '__main__':
    k = str(input("Masukkan k(ex. 01): "))
    n = int(input("Masukkan n: "))
    print()

    t1 = time.time()
    g = sequenceWithGreedy(k, n)
    t2 = time.time()
    print("Sekuens dengan Greedy: ", g)
```

```
    print("Apakah sekuens tersebut valid? ",
checkSequence(g, k, n))
    print("Waktu eksekusi (s):", t2-t1)
    print()

    t1 = time.time()
    d = sequenceWithDFS(k, n)
    t2 = time.time()
    print("Sekuens dengan DFS: ", d)
    print("Apakah sekuens tersebut valid? ",
checkSequence(d, k, n))
    print("Waktu eksekusi (s):", t2-t1)
```

IV. TESTING

Berikut akan dilakukan 4 test, dimana tiap test memiliki k dan n yang berbeda.

A. *k = 01; n = 5*

```
PS C:\Users\LENOVO\Kuliah\Tingkat2\Stima\Makalah> python -
Masukkan k(ex. 01): 01
Masukkan n: 5

Sekuens dengan Greedy:  111110111001101011000101001000000
Apakah sekuens tersebut valid?  True
Waktu eksekusi (s): 0.0010042190551757812

Sekuens dengan DFS:  000011111011010111001010011000010
Apakah sekuens tersebut valid?  True
Waktu eksekusi (s): 0.001001119613647461
```

Gambar 4.1. Test 1
Sumber: Dokumentasi Penulis

Dari test ini diperoleh waktu eksekusi yang hampir mirip untuk kedua metode, namun DFS memenangkannya. Meskipun begitu kedua metode ini masih terbukti efisien dan reliable untuk jumlah kombinasi yang keci, dimana pada test ini hanya terdapat 32 kombinasi.

B. *k = 01234; n = 4*

```
Masukkan k(ex. 01): 01234
Masukkan n: 4

Sekuens dengan Greedy:  444434442444144404433444324431443044234422442144204413441244114410440344024
40144004343424341434043334332433143304323432243214320431343124311431043034302430143004242414240423
34232423142304223422242214220421342124211421042034202420142004141404133413241314130412341224121412
04113411241114110410341024101410040403340324031403040234022402140204013401240114010400340024001400
03333233313330332233213320331233113310330233013300323231323032223221322032123211321032023201320031
31303122312131203112311131103102310131003030223021302030123011301030023001300022221222022112210220
1220021212021112110210121002020112010200120001111011001010000
Apakah sekuens tersebut valid?  True
Waktu eksekusi (s): 0.03225111961364746

Sekuens dengan DFS:  00044443434433433334442344243423342424432343243323332423232442234224322332242
22322224441344124414341334123414241324122414144313431243143313331233142313231223141313144213421242
14321332123214221322122214121312121441134112411431133112311421132112211411131112111144403440244014
40434033402340134042403240224012404140314021401140404430343024301430433033302330133042303230223012
30413031302130113040303044203420242014204320332023201320422032202220122041203120212011204020302020
44103410241014104310331023101310421032102210121041103110211011104010301020101044003400240014004300
3300230013004200320022001200410031002100110040003000200010
Apakah sekuens tersebut valid?  True
Waktu eksekusi (s): 0.000997781753540039
```

Gambar 4.2. Test 2
Sumber: Dokumentasi Penulis

Pada test ini jumlah kombinasinya jauh meningkat menjadi 625 kombinasi, dapat dilihat bahwa keduanya memperoleh sekuens yang benar. Tetapi jika diperhatikan, waktu eksekusi dengan DFS lebih cepat dibandingkan dengan greedy.

C. *k = 0123; n = 5*

```
Masukkan k(ex. 01): 0123
Masukkan n: 5

Sekuens dengan Greedy:  33333233331333303332233321333203331233311333103330233301333003323233231332
30332223322133220332123321133210332023320133200331323313133130331223312133120331123311133110331023
31013310033032330313303033022330213302033012330113301033002330013300032322323213232032312323113231
03230232301323003223132230322223222132220322123221132210322023220132200321313213032122321213212032
11232111321103210232101321003203132030320223202132020320123201132010320023200132000313123131131310
31302313013130031230312223122131220312123121131210312023120131200311303112231121311203111231111311
10311023110131100310303102231021310203101231011310103100231001310003030230301303003022230221302203
02123021130210302023020130200301223012130120301123011130110301023010130100300223002130020300123001
13001030002300013000022222122220222112221022201222002212122120221112211022101221002202122020220112
20102200122000212112121021201212002112021111211102110121100210202101121010210012100020201202002011
1201102010120100200112001020001200001111101110011010110001010010000
Apakah sekuens tersebut valid?  True
Waktu eksekusi (s): 0.05341076850891113

Traceback (most recent call last):
  File "c:\Users\LENOVO\Kuliah\Tingkat2\Stima\Makalah\main.py", line 74, in <module>
    d = sequenceWithDFS(k, n)
  File "c:\Users\LENOVO\Kuliah\Tingkat2\Stima\Makalah\main.py", line 20, in sequenceWithDFS
    dfs(start, k)
  File "c:\Users\LENOVO\Kuliah\Tingkat2\Stima\Makalah\main.py", line 14, in dfs
    dfs(combination[1:], k)
  File "c:\Users\LENOVO\Kuliah\Tingkat2\Stima\Makalah\main.py", line 14, in dfs
    dfs(combination[1:], k)
  File "c:\Users\LENOVO\Kuliah\Tingkat2\Stima\Makalah\main.py", line 14, in dfs
    dfs(combination[1:], k)
  [Previous line repeated 994 more times]
  File "c:\Users\LENOVO\Kuliah\Tingkat2\Stima\Makalah\main.py", line 13, in dfs
    visited.add(combination)
RecursionError: maximum recursion depth exceeded while calling a Python object
```

Gambar 4.3. Test 3
Sumber: Dokumentasi Penulis

Dapat dilihat bahwa DFS mengalami error dalam membuat sekuensnya, padahal jumlah kombinasinya tidak berbeda cukup jauh dengan test sebelumnya. Jumlah kombinasi pada test ini adalah 1024, hal ini menunjukkan bahwa diantara kedua metode, greedy lebih reliable untuk membuat sekuens dari kombinasi yang lebih banyak dan waktu eksekusinya juga masih cepat.

V. KESIMPULAN

Dari 3 test tersebut dapat dilihat bahwa kedua metode cukup cepat dalam membuat sekuens dan sekuens yang dihasilkan pun telah mengandung semua kombinasi. Namun hal yang dapat menjadi masalah apabila jumlah kombinasi menjadi sangat banyak. Pada greedy sekuens masih dihasilkan dengan tepat, namun pada DFS terdapat RecursionError yang menyebabkan program terhenti. Tetapi, error ini dapat dihindarkan bila algoritma DFS tidak menggunakan metode rekursif oleh karena itu belum dapat diukur kemampuannya. Untuk saat ini, algoritma greedy menjadi metode yang efisien dan reliable untuk menyusun sekuens de Bruijn yang tepat.

PENUTUP

Puji syukur penulis panjatkan sebesar-besarnya pada Tuhan Yang Maha Esa, atas karuniaNya yang diberikan kepada penulis sehingga penulis dapat menyelesaikan makalah ini dengan tepat waktu. Penulis juga menyampaikan rasa terima kasih kepada seluruh dosen mata kuliah Strategi Algoritma khususnya Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc. selaku dosen untuk K02, yaitu kelas tempat saya berkuliah. Penulis juga mengucapkan terima kasih kepada kedua orang tua penulis yang selalu mendukung kegiatan perkuliahan penulis. Penulis juga ingin berterima kasih kepada teman-teman yang selalu

membantu dan memberikan ide, masukan, dan semangat dalam mengerjakan makalah ini.

REFERENCES

[1] Munir, Rinaldi. 2022. Kombinatorial (Bagian 1). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Kombinatorial-2020-Bagian1.pdf pada tanggal 22 Mei 2023.

[2] Munir, Rinaldi. 2022. Graf (Bagian 1). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf pada tanggal 22 Mei 2023.

[3] Munir, Rinaldi. 2022. Graf (Bagian 2). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian2.pdf pada tanggal 22 Mei 2023.

[4] Munir, Rinaldi. 2023. Algoritma Greedy (Bagian 1). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf pada tanggal 22 Mei 2023.

[5] Munir, Rinaldi. 2023. Breadth First Search (BFS) dan Depth First Search (DFS) (Bagian 1). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/BFS-DFS-2021-Bag1.pdf pada tanggal 22 Mei 2023.

[6] Munir, Rinaldi. 2023. Pencocokan String dengan Regular Expression (Regex). Diakses pada https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2018-2019/String-Matching-dengan-Regex-2019.pdf pada tanggal 22 Mei 2023.

[7] Opengenus. 2023. *De Bruijn Sequences.* Diakses pada https://iq.opengenus.org/de-brujin-sequences/pada tanggal 22 Mei 2023 pada tanggal 22 Mei 2023.

[8] Geeksforgeeks. 2022. *De Bruijn sequence Set 1*. Diakses pada https://www.geeksforgeeks.org/de-bruijn-sequence-set-1/ pada tanggal 22 Mei 2023.

[9] De Bruijn Sequence. 2022. *De Bruijn Sequence and Universal Cycle Constructions.* Diakses pada http://debruijnsequence.org/db/graph pada tanggal 22 Mei 2023.

[10] DataGenetics. 2013. *De Bruijn Sequences.* Diakses http://datagenetics.com/blog/october22013/index.html pada tanggal 22 Mei 2023.

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, 22 Mei 2023

Christian Albert Hasiholan
13521078