Aplikasi Graf dan Himpunan dalam Program Pemecah *Puzzle* Kakuro

Arsa Izdihar Islam – 13521101[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]13521101@std.stei.itb.ac.id

***Abstract*—Banyak *puzzle* dengan bentuk yang bervariasi, salah satunya adalah Kakuro. Kakuro yang juga dikenal dengan nama Cross Sum adalah sebuah teka-teki yang mirip dengan teka teki silang tetapi dalam representasi angka dan matematika. Kakuro dapat diselesaikan dengan banyak cara, salah satunya adalah dengan menggunakan graf berbobot.**

***Keywords*—Kakuro, *puzzle*, angka, graf.**

I. PENDAHULUAN

Teka-teki atau *puzzle* adalah permainan atau masalah yang dirancang untuk menguji kemampuan berpikir seseorang. Teka-teki biasanya terdiri dari pertanyaan atau situasi yang membutuhkan logika atau penalaran untuk menemukan jawabannya. Teka-teki dapat berupa permainan sederhana seperti teka-teki silang atau anagram, atau bahkan bisa menjadi soal-soal matematika atau logika yang lebih kompleks. Salah satu teka-teki yang melibatkan matematika dan logika adalah Kakuro.

Kakuro memiliki berbagai sebutan yaitu Cross Sum di AS dan *Kakro* di Jepang. Kakuro merupakan salah satu teka-teki yang populer di Jepang selain Sudoku. Awal munculnya Kakuro adalah dengan sebutan Cross Sum pada tahun 1966 ketika diterbitkan oleh Dell Magazines (juga menerbitkan sudoku satu dekade setelahnya). Setelah itu, Kakuro sering dimunculkan dalam publikasi matematika dan logika di AS.

Bentuk *puzzle* Kakuro adalah sebuah kisi dengan kotak "hitam" dan "putih". Bagian yang perlu diselesaikan dalam teka-teki ini adalah bagian putih. Umumnya, Kakuro berukuran 16x16 tetapi dapat bervariasi juga. Kisi pada Kakuro terbagi menjadi banyak entri yaitu kotak putih yang segaris. Setiap entri memiliki batasan tertentu, yaitu jumlah angka dalam entri harus sesuai dengan angka tertentu dan angka-angkanya merupakan angka 1 sampai 9 yang tidak berulang.

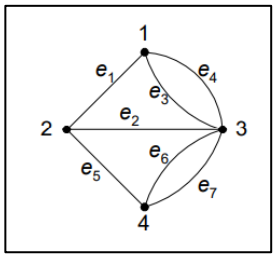
Aturan yang ada pada Kakuro dapat direpresentasikan dalam sebuah graf berbobot. Dalam makalah ini, penulis akan menggunakan representasi graf tersebut untuk membuat algoritma untuk memecahkan teka-teki ini.

II. DASAR TEORI

A. Graf

Graf adalah kumpulan simpul dan sisi untuk merepresentasikan berbagai objek dan hubungan antara objek-objek tersebut. Komponen pada graf terdiri dari simpul dan sisi, yang dapat direpresentasikan dengan:

G = (V, E), dengan
V = himpunan tidak kosong dari simpul-simpul
= { v_1, v_2. v_3, …, v_n }
E = himpunan tidak kosong dari sisi-sisi
= { e_1, e_2. e_3, …, e_n }

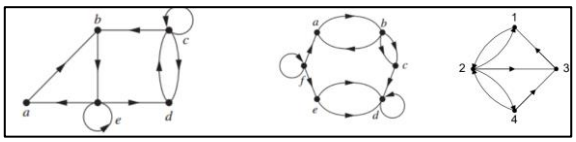


Gambar 1. Contoh graf
(Sumber: *https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf)*

Contoh, graf G pada gambar 1 adalah graf dengan G = (V, E), V = { 1, 2, 3, 4}, E = { (1, 2), (1,3), (1, 3), (2, 3), (2, 4), (3, 4), (3,4) }.

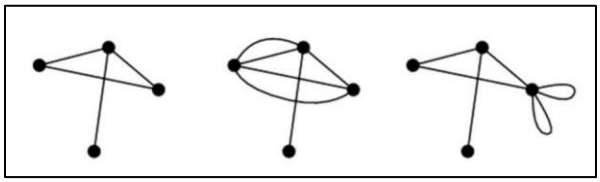
Berdasarkan orientasi arah pada sisinya, graf dapat dibedakan menjadi dua jenis yaitu:

1. Graf berarah (*directed graph atau digraph)*
   Graf berarah merupakan graf yang sisinya memiliki orientasi arah.



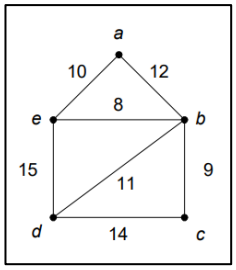
Gambar 2. Graf berarah
(Sumber: *https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf)*

2. Graf tak berarah (*indirected graph)*
   Graf tak berarah merupakan graf yang sisinya tidak memiliki orientasi arah.



Gambar 3. Graf tak berarah
(Sumber: *https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf)*

Salah satu terminology pada graf adalah graf berbobot yaitu graf yang setiap sisinya diberi suatu harga (bobot). Bobot ini dapat berbentuk apapun.



Gambar 4. Graf berbobot
(Sumber: *https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf)*

B. *Himpunan*

Himpunan atau *set* merupakan sekumpulan objek dengan nilai yang berbeda yang disebut dengan elemen, unsur atau anggota. Himpunan dapat dinyatakan dengan menggunakan tanda kurung kurawal "{}", diikuti dengan elemen-elemen yang ada di dalamnya. Contohnya, himpunan A = {1, 2, 3, 4, 5} adalah sebuah himpunan yang terdiri dari 5 elemen yaitu angka 1, 2, 3, 4, dan 5. Elemen dari himpunan juga mungkin berupa himpunan lainnya. Terdapat beberapa operasi pada himpunan yaitu sebagai berikut:

1. Irisan (*intersection)*
   Operasi irisan (notasi $A \cap B = \{ x \mid x \in A \text{ dan } x \in B \}$) menghasilkan himpunan yang elemennya merupakan elemen yang terdapat pada himpunan A dan himpunan B.
2. Gabungan (*union*)
   Operasi gabungan (notasi $A \cup B = \{ x \mid x \in A \text{ atau } x \in B \}$) menghasilkan himpunan yang elemennya merupakan elemen yang terdapat pada himpunan A atau himpunan B.
3. Komplemen (*complement)*
   Operasi komplemen (notasi $\overline{A} = \{ x \mid x \in U, x \notin A \}$) menghasilkan himpunan yang elemennya merupakan elemen pada semesta yang tidak ada pada himpunan A.
4. Selisih (*difference)*
   Operasi selisih (notasi $A - B = \{ x \mid x \in A \text{ dan } x \notin B \} = A \cap \overline{B}$) menghasilkan himpunan yang elemennya merupakan elemen himpunan A yang tidak terdapat pada himpunan B.
5. Beda setangkup (*symmetric difference)*
   Operasi beda setangkup (notasi $A \oplus B = (A \cup B) - (A \cap B) = (A - B) \cup (B - A)$) menghasilkan himpunan yang elemennya merupakan elemen himpunan A atau himpunan B tetapi bukan keduanya.
6. Perkalian kartesian (*cartesian product)*
   Operasi perkalian kartesian (notasi $A \times B = \{(a, b) \mid a \in A \text{ dan } b \in B \}$) menghasilkan himpunan yang elemennya terdiri dari semua pasangan-pasangan elemen yang elemen pertama pada pasangan tersebut adalah elemen himpunan A dan elemen keduanya adalah elemen himpunan B.

C. *Library* NetworkX di Python

NetworkX merupakan sebuah *library* pada bahasa pemrograman Python yang berguna untuk membuat, memanipulasi, dan mempelajari struktur, dinamika, dan fungsi dari *network* yang kompleks. *Library* ini dapat membantu merepresentasikan graf ke dalam program karena mempunyai data struktur dan metode untuk menyimpan graf. Setiap graf pada NetworkX memungkinkan kita untuk menyimpan objek Python apapun sebagai simpul dan atribut dari sisi.

Terdapat beberapa *class* dari graf bergantung pada struktur graf yang dibutuhkan, dengan rincian sebagai berikut:

Tabel 1. *Class* graf yang tersedia pada NetworkX
(Sumber: *https://networkx.org/documentation/stable/reference/classes/index.html*)

Class	Tipe	Sisi Gelang	Sisi Ganda
Graph	Tak berarah	Ya	Tidak
DiGraph	Berarah	Ya	Tidak
MultiGraph	Tak berarah	Ya	Ya
MultiDiGraph	Berarah	Ya	Ya

Setiap *class* tersebut memiliki beberapa *method* umum untuk menambahkan simpul dan menambahkan sisi pada graf. Misal, untuk membuat graf kosong tanpa simpul dan sisi, dapat dengan:

```
G = nx.Graph()
```

Gambar 5. Kode untuk membuat graf kosong

Kemudian, untuk menambahkan simpul dan sisi pada graf G tersebut, dapat dengan:

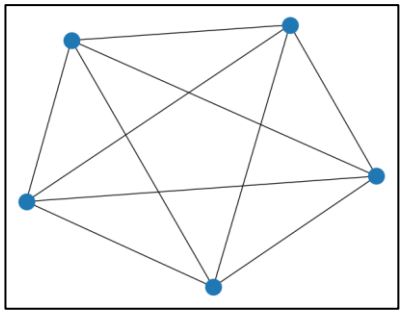
```
# Simpul
# Menambah satu simpul dengan identitas 1
G.add_node(1)

# Menambah banyak simpul sekaligus
G.add_nodes_from([2, 3])
G.add_nodes_from(range(100, 110))
H = nx.path_graph(10)
G.add_nodes_from(H)

# Sisi
# Menambah satu sisi dari simpul 1 dan 2
G.add_edge(1, 2)

# Menambah banyak sisi sekaligus
G.add_edges_from([(1, 2), (1, 3)])
```

Gambar 6. Kode untuk menambahkan simpul dan sisi pada graf

Selain untuk menyimpan struktur graf, *library* ini juga dapat melakukan visualisasi dengan menggunakan matplotlib sebagai berikut:

```
import networkx as nx
import matplotlib.pyplot as plt

G = nx.complete_graph(5)
nx.draw(G)
plt.show()
```

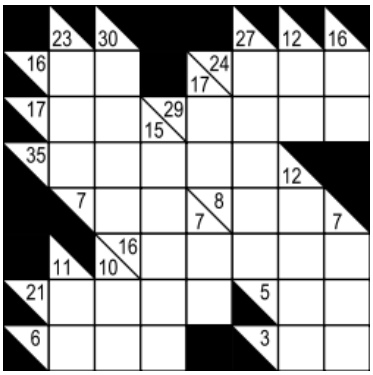
Gambar 7. Kode untuk visualisasi graf



Gambar 8. Output dari kode pada gambar 7

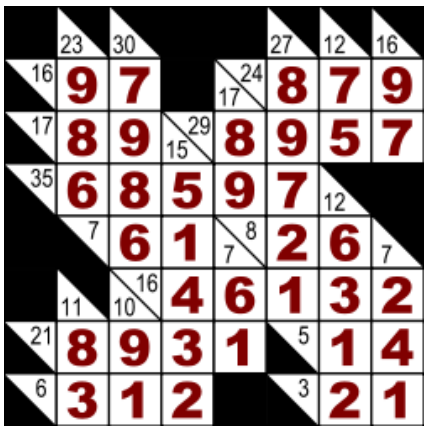
D. Kakuro

Kakuro merupakan sebuah *puzzle* yang mirip dengan teka-teki silang tetapi dalam representasi angka. Objektif dalam *puzzle* ini adalah dengan mengisi angka 1 sampai 9 pada semua kotak putih sehingga jumlah dari angka-angka pada tiap entri sama dengan batasan yang ada dengan tidak ada angka yang duplikat pada tiap entri.



Gambar 9. Contoh *puzzle* Kakuro
(Sumber: https://en.wikipedia.org/wiki/Kakuro)

Seperti halnya sudoku, memecahkan Kakuro melibatkan kombinasi dan permutasi. Contohnya, apabila terdapat entri dengan tiga kotak dan jumlahnya 6, maka kombinasi yang mungkin untuk tiga kotak tersebut hanyalah {1, 2, 3}. Hal ini merupakan kunci dalam memecahkan Kakuro.



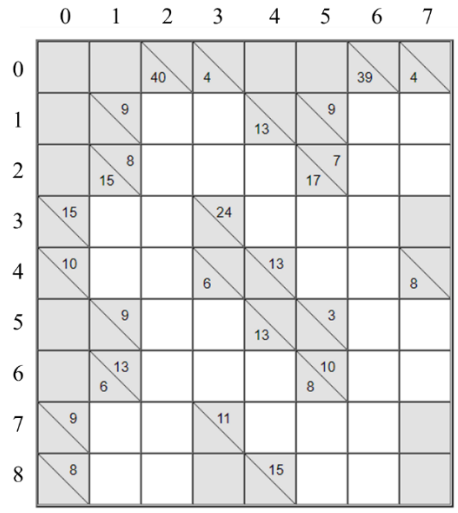
Gambar 10. Solusi dari *puzzle* pada gambar 6
(Sumber: https://en.wikipedia.org/wiki/Kakuro)

III. Aplikasi Graf dan Himpunan dalam Program Pemecah Kakuro

A. Representasi Graf

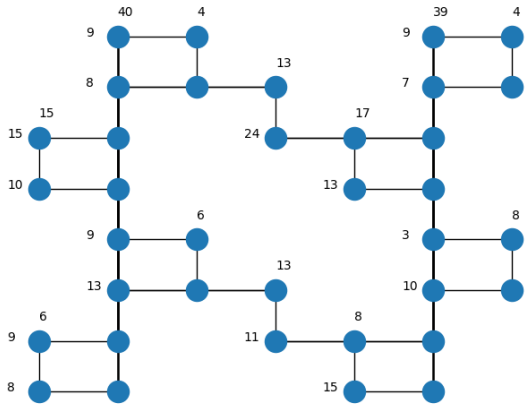
Graf yang digunakan adalah graf berbobot dan tak berarah. Setiap simpul menyatakan kotak putih yang harus diisi dengan identitas berupa baris dan kolom simpul itu berada pada *puzzle* (baris, kolom) dengan baris diawali dari atas dan kolom diawali dari kiri. Simpul-simpul ini juga akan memiliki nilai yaitu angka yang diisikan, apabila belum diisi, maka akan menyimpan nilai 0.

Sedangkan, setiap sisi merupakan batasan antara satu kotak dengan yang lain sehingga tiap kotak yang berada dalam satu baris atau satu kolom tanpa terhalangi kotak hitam (satu kelompok) akan saling bertetanggaan (memiliki sisi). Setiap sisi akan memiliki bobot berupa himpunan yang tiap anggotanya merupakan kombinasi angka-angka yang mungkin untuk dapat mengisi kelompok tersebut sehingga jika dalam satu kelompok terdapat tiga kotak yang belum terisi, bobot tiap sisi antar kotak dalam kelompok tersebut akan memiliki anggota berupa himpunan kombinasi tiga angka.



Gambar 11. Contoh *puzzle* Kakuro yang akan direpresentasikan
(Sumber: *https://www.kakuros.com/?s=9x8)*

Berdasarkan gambar tersebut, simpul (2, 2), (2, 3), dan (2, 4) jumlahnya harus 8 dengan tiga angka berbeda. Ketiga sisi tersebut akan memiliki sisi yang menghubungkan satu sama lain dengan bobot berupa kombinasi tiga angka berbeda yang apabila dijumlahkan bernilai 8 yaitu {{1, 2, 5}, {1, 3, 4}}. Simpul (1,3) dan (2,3) yang berjumlah 4 akan memiliki sisi dengan bobot {{1, 3}}. Simpul (1,2) dan (1, 3) yang berjumlah 9 akan memiliki sisi dengan bobot {{1, 8}, {2, 7}, {3, 6}, {4, 5}} dan seterusnya untuk semua kelompok yang ada pada *puzzle.* Apabila graph tersebut digambarkan tanpa bobot sisinya, akan menjadi seperti berikut:



Gambar 12. Plot representasi graf tanpa bobotnya

Dalam kode Python, dapat dibuat *class* Graph yang merupakan *inheritance* dari *class* Graph dari *library* NetworkX. *Class* ini memiliki *method* untuk menginisiasi konfigurasi Kakuro, yaitu kotak-kotak yang ada dan batasan-batasan untuk tiap entrinya. Selain itu, terdapat juga *method* untuk mencari kombinasi angka yang mungkin untuk memenuhi entri dengan banyak kotak dan jumlah tertentu.

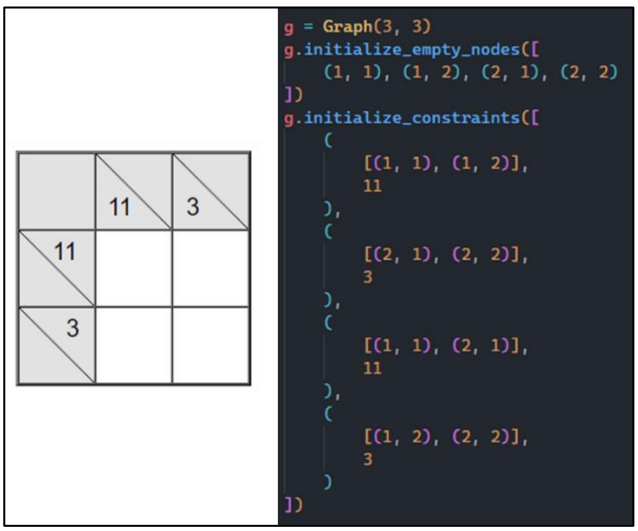
```python
def initialize_empty_nodes(self, positions: list[tuple[int, int]]):
    # Menambahkan kotak putih ke dalam graf, diinisiasi dengan value
    = 0
    for pos in positions:
        self.add_node(tuple(pos), value=0)

def initialize_constraints(self, contraints):
    # Menambahkan batasan tiap entri ke dalam graf, yaitu berupa
    sisi yang menghubungkan antar kotak pada suatu entri
    for constraint in contraints:
        len_nodes = len(constraint[0])
        sum_val = constraint[1]
        possible_values = Graph.get_possible_values(len_nodes,
        sum_val)
        _row, _col = constraint[0][0]
        for i, (row, col) in enumerate(constraint[0]):
            if row != _row and col != _col:
                raise Exception(
                    f"Invalid constraint {constraint}, must be in
                    same row or col")
            node = self.nodes.get((row, col))
            if node is None:
                raise Exception("Node is not empty")

            for j in range(i + 1, len(constraint[0])):
                row2, col2 = constraint[0][j]
                self.add_edge((row, col), (row2, col2),
                              weight=copy.deepcopy(possible_values))
```

Gambar 13. *Method* untuk menginisiasi Kakuro pada Python

```python
@staticmethod
def get_possible_values(length: int, sum_val: int, start=1):
    # Mengembalikan semua kemungkinan nilai yang dapat diisi
    pada entri dengan panjang length dan jumlah sum_val
    if start > 9:
        return []

    if length == 1:
        if sum_val < 1 or sum_val > 9 or sum_val < start:
            return []
        return [[sum_val]]

    # Menggunakan hasil kombinasi yang telah disimpan sebelumnya
    if (length, sum_val) in Graph.possible_values:
        pos_start, possible_values = Graph.possible_values[(
            length, sum_val)]
        if pos_start == start:
            return possible_values
        elif pos_start < start:
            filtered = filter(lambda x: min(x) >= start,
            possible_values)
            return list(filtered)

    possible_values = []
    rest_possible = Graph.get_possible_values(
        length - 1, sum_val - start, start + 1)
    for rest in rest_possible:
        possible_values.append([start] + rest)
    possible_values.extend(
        Graph.get_possible_values(length, sum_val, start + 1))

    # Menyimpan hasil kombinasi yang telah dihitung agar tidak
    perlu menghitung ulang kombinasi yang sama
    Graph.possible_values[(length, sum_val)] = (start,
    possible_values)
    return possible_values
```

Gambar 14. *Method* untuk mencari kombinasi angka untuk suatu entri

Kemudian, untuk menggunakan *method-method* tersebut dalam mentranslasikan *puzzle* Kakuro ke dalam program adalah dengan membuat objek dari *class* Graph. Kemudian menggunakan *method* "initialize_empty_nodes" dan "initialize_constraints". Berikut adalah contoh *puzzle* sederhana beserta cara translasinya:



Gambar 15. Kode untuk mentranslasi Kakuro ke Python

B. Mencari Solusi Kakuro

Setelah representasi graf berhasil dibuat, hal yang perlu dilakukan adalah mengisi tiap simpul dengan nilai yang sesuai.

Untuk itu, perlu ditemukan angka-angka yang mungkin untuk suatu simpul. Pada hubungan suatu simpul dengan satu entri, angka-angka yang mungkin bisa ditentukan dengan mengambil bobot dari simpul tersebut dengan salah satu simpul lainnya dalam entri tersebut. Kemudian, bobot yang himpunan kombinasi angka-angka diratakan, yaitu dengan menggabungkan atau *union* seluruh anggota himpunan bobot tersebut menjadi suatu himpunan. Misalnya entri yang memiliki tiga simpul dengan jumlah 8 pada sisi antar simpulnya akan memiliki bobot {{1, 2, 5}, {1, 3, 4}}. Maka angka yang mungkin untuk setiap simpul di entri tersebut sudah dibatasi hanya menjadi {1, 2, 3, 4, 5}.

Dalam setiap simpul dalam graf Kakuro merupakan bagian dari dua entri. Maka dari itu, hasil akhir yang merupakan angka yang mungkin pada simpul tersebut adalah irisan atau *intersection* dari hasil pencarian kedua entri. Misal, simpul (1, 2) *puzzle* Kakuro pada gambar 15 adalah bagian dari entri dengan jumlah 11 dan 3 dengan masing-masing memiliki dua simpul. Bobot dari entri dengan jumlah 11 adalah {{2, 9}, {3, 8}, {4, 7}, {5, 6}} dan angka yang mungkin adalah {2, 3, 4, 5, 6, 7, 8, 9}. Sedangkan bobot dari entri dengan jumlah 3 adalah {{1, 2}} dan angka yang mungkin adalah {1, 2}. Irisan dari kedua himpunan tersebut adalah {2} sehingga hanya angka 2 yang mungkin untuk mengisi simpul tersebut.

Setelah mendapatkan angka-angka yang mungkin pada suatu simpul, yang perlu dilakukan adalah membuat fungsi untuk memasukkan nilai pada simpul dan memperbarui sisi-sisi pada simpul yang terdampak. Langkahnya adalah sebagai berikut. Misal simpul yang akan dimasukkan nilainya disebut simpul A. Pertama, melakukan *loop* untuk semua tetangga dari simpul A, misal tetangga simpul A adalah simpul B. Kemudian, di dalam *loop* tersebut dilakukan *loop* untuk semua tetangga simpul B, misal tetangga simpul B adalah simpul C. Simpul C akan diproses jika simpul tersebut terdapat dalam entri yang sama dengan simpul A dan simpul B. Proses yang dilakukan adalah *loop* semua anggota dari himpunan bobot simpul B dan simpul C. Jika angka yang ingin dimasukkan ke simpul A terdapat dalam anggota himpunan tersebut, maka angka tersebut dihapus dari himpunan. Sebaliknya, jika tidak ada, maka kemungkinan tersebut tidak terpenuhi lagi sehingga himpunan bisa dihapus dari himpunan bobot. Kemudian, setelah *loop* semua simpul C, jika pada simpul B terdapat lebih dari satu kemungkinan angka (kasus masih terdapat dua atau lebih simpul pada entri yang belum diisi) maka hapus sisi antara simpul A dan simpul B. Sebaliknya, sisi antara simpul A dan simpul B tidak dihapus supaya angka yang mungkin dimasukkan pada simpul B tetap tersimpan.

```python
def set_value(self, pos, value):
    # Mengisi nilai pada simpul tertentu dengan angka tertentu dan
    memperbarui sisi-sisi yang terdampak
    self.nodes[pos]["value"] = value
    to_remove = []
    processed = set()
    for _, neighbor, _ in self.edges(pos, True):
        if neighbor[0] == pos[0]:
            isSameRow = True
        else:
            isSameRow = False
        for _, neighbor2, data in self.edges(neighbor, True):
            # hanya memproses simpul yang berada pada entri yang sama
            dengan simpul A dan simpul B
            if (isSameRow and neighbor2[0] != pos[0]) or (not isSameRow
            and neighbor2[1] != pos[1]):
                continue
            # hanya memproses sekali
            if (neighbor2, neighbor) in processed:
                continue
            processed.add((neighbor, neighbor2))
            weight = data.get("weight", [])
            pos_to_remove = []
            for possible in weight:
                # jika nilai yang dimasukkan terdapat pada kemungkinan,
                maka hapus nilai tersebut pada kemungkinan, sebaliknya
                hapus kemungkinan tersebut
                if value in possible:
                    possible.remove(value)
                else:
                    pos_to_remove.append(possible)
            for p in pos_to_remove:
                weight.remove(p)

        to_remove.append(neighbor)
    for r in to_remove:
        edge = self.edges.get((r, pos))
        if edge is not None:
            weight = edge.get("weight", [])
            if len(weight) != 1 or len(weight[0]) != 1:
                self.remove_edge(r, pos)
```

Gambar 16. *Method* untuk mengisi nilai pada suatu simpul

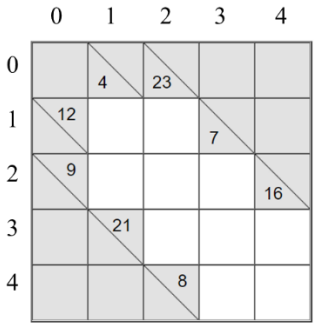
Kemudian, dilakukan pemanggilan fungsi rekursif untuk mencoba kemungkinan-kemungkinan yang ada hingga seluruh simpul/kotak berhasil diisi. Jika pada suatu kemungkinan didapatkan ujung yaitu tidak ada angka yang mungkin untuk suatu simpul, maka kemungkinan tersebut bukanlah solusi yang benar.

Langkah pada fungsi rekursifnya adalah sebagai berikut. Fungsi rekursif akan menerima input berupa *list* simpul yang belum ditentukan nilainya. Jika *list* tersebut kosong, artinya seluruh simpul telah berhasil dicari nilainya dan *puzzle* berhasil dipecahkan. Sebaliknya, mengambil elemen pertama pada *list* tersebut dan mencari angka-angka yang mungkin pada simpul yang diambil. Jika tidak ada angka yang mungkin, berarti solusi tidak ditemukan (kemungkinan yang diambil sebelumnya salah). Sebaliknya, memanggil *method* “set_value” untuk simpul dengan nilai-nilai yang ada kemudian memanggil kembali fungsi rekursif untuk menentukan nilai sisanya hingga ditemukan solusi yang benar.

```python
def solve(self):
    return self.__solve_rec(list(self.nodes(True)))

def __solve_rec(self, rest):
    if len(rest) == 0:
        return True

    first = rest[0]
    pos = first[0]
    possible_values = self.get_node_possible_values(pos)
    if len(possible_values) == 0:
        return False

    for value in possible_values:
        copy = self.copy()
        copy.set_value(pos, value)
        if copy.__solve_rec(rest[1:]):
            self.nodes = copy.nodes
            self.edges = copy.edges
            return True
    return False
```

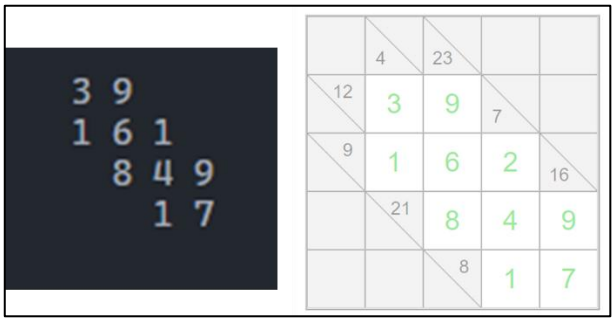
Gambar 17. *Method* untuk mencari solusi *puzzle*

C. Contoh Penyelesaian



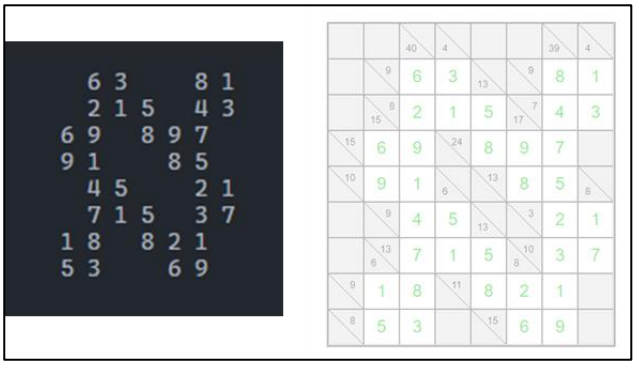
Gambar 18. Kakuro yang akan coba diselesaikan
(Sumber: https://www.kakuros.com/?s=5x5)

Pada gambar tersebut, fungsi rekursif akan dimulai dengan mengisi simpul (1, 1). Simpul tersebut memiliki nilai yang mungkin yaitu {1, 3} pada entri menurun dan {3, 4, 5, 7, 8, 9} pada entri mendatar sehingga nilai yang mungkin hanyalah angka 3. Maka simpul (1, 1) diberi angka 3. Setelah itu, bobot sisi antara simpul (1, 1) dan (1, 2) hanya tersisa {{9}} dan bobot sisi antara simpul (1, 1) dan (2, 1) hanya tersisa {{1}}. Kemudian, fungsi rekursif dipanggil kembali untuk memproses simpul (1, 2). Nilai yang mungkin pada entri mendatar adalah {9} dan menurun adalah {6, 8, 9} sehingga nilai yang mungkin hanyalah angka 9. Maka simpul (2, 2) diberi angka 9. Sisi pada entri mendatar terhapus semua dan bobot pada sisi menurun tersisa {{6, 8}}. Pada simpul (2, 1) nilai yang mungkin adalah {1} sehingga simpul (2, 1) diberi angka 1 yang mengakibatkan entri mendatar tersisa {{2, 6}, {3, 5}}. Kemudian, nilai yang mungkin pada simpul (2, 2) adalah {6} sehingga simpul tersebut diberi nilai 6. Simpul (2, 3) tersisa satu kemungkinan yaitu 2. Simpul (3, 2) tersisi satu kemungkinan yaitu 8. Kemudian, pada proses simpul (3, 3), nilai yang mungkin pada entri mendatar adalah {4, 6, 7, 9} dan pada entri menurun adalah {1, 4} sehingga simpul tersebut diberi angka 4. Simpul (3, 4) tersisa satu kemungkinan yaitu 9. Simpul (3, 4) tersisa satu kemungkinan yaitu 1. Simpul (4, 4) tersisa satu kemungkinan yaitu 9 sehingga seluruh simpul berhasil diberi nilai.

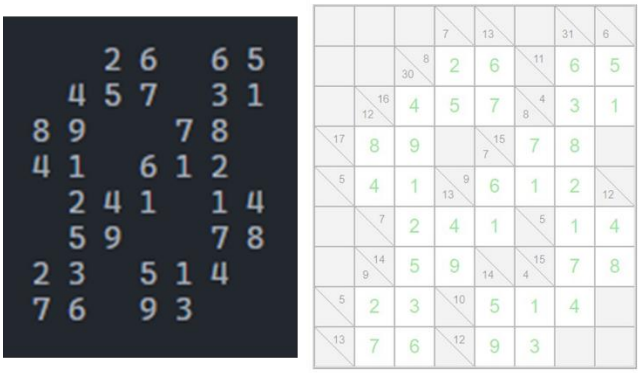


Gambar 19. Hasil pencarian solusi dari kode

Selain itu, telah dicoba dengan beberapa *puzzle* lainnya dengan hasil sebagai berikut:



Gambar 20. Hasil pencarian solusi dari kode pada percobaan kedua



Gambar 21. Hasil pencarian solusi dari kode pada percobaan ketiga

IV. KESIMPULAN

Kakuro merupakan salah satu *puzzle* yang mirip dengan teka-teki silang dan juga Sudoku. Dalam *puzzle* ini, pemain diharuskan untuk mengisi seluruh kotak putih yang kosong dengan batasan tertentu. Kotak yang harus diisi berdasarkan batasan-batasannya dapat direpresentasikan dengan menggunakan graf berbobot dan tak berarah. Bobot dari sisi pada graf tersebut bisa direpresentasikan dengan menggunakan himpunan berisi himpunan kombinasi-kombinasi yang mungkin untuk mengisi entri pada Kakuro. Representasi graf ini sangat membantu dalam membuat program untuk menyelesaikan teka-teki ini. Akan tetapi, mengubah bentuk *puzzle* Kakuro ke dalam inisiasi bentuk graf di program lumayan rumit seperti yang ditunjukkan pada gambar 15. Selain itu, program yang dibuat

dalam makalah ini masih cenderung *brute force* dalam mencari solusi yang tepat. Banyak hal yang dapat dioptimasi dalam menentukan simpul mana yang harus diisi terlebih dahulu.

V. Ucapan Terima Kasih

Penulis ingin mengucapkan terima kasih kepada beberapa pihak. Pertama, penulis mengucapkan puji syukur kepada Allah SWT karena atas nikmat dan rahmat-Nya, penulis dapat menyelesaikan makalah berjudul "Aplikasi Graf dan Himpunan dalam Program Pemecah Puzzle Kakuro" dengan baik. Selain itu, tidak lupa juga ucapan terima kasih kepada dosen mata kuliah Matematika Diskrit, Dr. Nur Ulfa Maulidevi, S. T, M. Sc., Dr. Ir. Rinaldi Munir, M. T., dan Dr. Fariska Zakhralativa Ruskanda, S.T. yang telah membimbing penulis selama berkuliah di mata kuliah ini. Penulis juga mengucapkan terima kasih kepada seluruh sumber yang dijadikan referensi pada makalah ini.

Referensi

[1] Munir, Rinaldi. 2020. "Graf (Bag. 1)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2020-2021/Graf-2020-Bagian1.pdf, diakses pada 10 Desember 2022, pukul 15.30 WIB.
[2] Munir, Rinaldi. 2022. "Teori Himpunan (Bagian 1 - versi update 2022)". https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2022-2023/Himpunan(2022)-1.pdf, diakses pada 10 Desember 2022, pukul 18.43 WIB.
[3] https://networkx.org/documentation/stable/, diakses pada 10 Desember 2022 pukul 16.50 WIB.
[4] https://www.kakurolive.com/about-kakuro.php, diakses pada 10 Desember 2022 pukul 15.05 WIB.
[5] https://www.kakuros.com, diakses pada 10 Desember 2022 pukul 15:40 WIB.

Tautan Video

https://youtu.be/v3nDNAevExs

Tautan Kode Program

https://github.com/arsaizdihar/kakuro-solver

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, 11 Desember 2022

Arsa Izdihar Islam 13521101