Tugas Eksplorasi Mandiri
Perancangan dan Analisis Algoritma
Maximum Bipartite Matching

DOSEN PENGAMPU:

Randi Proska Sandra, S.Pd., M.Sc.

OLEH:

Nama : Agil Abrar

NIM : 22343018

Seksi : 202323430047

Kelompok G

PROGRAM STUDI INFORMATIKA

DEPARTEMEN ELEKTRONIKA

FAKULTAS TEKNIK

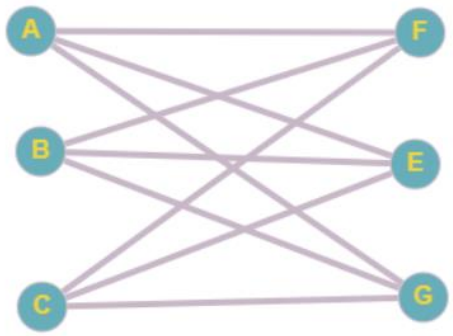
UNIVERSITAS NEGERI PADANG

2024

A. PENJELASAN ALGORITMA MAXIMUM BIPARTITE MATCHING

Algoritma Maximum Bipartite Matching adalah algoritma yang digunakan untuk menemukan pencocokan maksimum di dalam graf bipartit. Maksimum disini adalah matching dengan ukuran terbesar (jumlah sisi terbanyak) pada graf bipartit.Dengan kata lain, ini memaksimalkan jumlah "pasangan" yang mungkin terjadi.

Graf bipartit adalah jenis graf yang dibagi menjadi dua himpunan vertex yang terpisah, biasanya disebut sebagai bagian kiri dan kanan.Tidak ada sisi yang menghubungkan vertex dalam himpunan yang sama. Semua sisi harus menghubungkan vertex dari bagian kiri ke bagian kanan, atau sebaliknya.



Contoh Penerapan algoritma Maximum Bipartite Matching, bayangkan sebuah pesta dansa, di mana ada dua kelompok terpisah: tamu pria dan tamu wanita.Graf bipartit dapat memodelkan situasi ini.Ini adalah jenis graf yang dibagi menjadi dua himpunan vertex yang terpisah, biasanya disebut sebagai bagian kiri dan kanan.Tidak ada orang yang bisa berdansa dengan orang dari kelompok yang sama (tidak ada sisi yang menghubungkan vertex dalam himpunan yang sama).Semua undangan (sisi) harus menghubungkan tamu pria (vertex kiri) dengan tamu wanita (vertex kanan), atau sebaliknya.

Di pesta dansa, matching berarti kita memasangkan para tamu untuk berdansa. Pada graf bipartit, matching adalah kumpulan sisi pada graf yang tidak berbagi vertex yang sama.Setiap sisi merepresentasikan pasangan dansa antara seorang tamu pria dan seorang tamu wanita.Tujuannya adalah menemukan penugasan dansa yang paling optimal.Ini berarti kita ingin memaksimalkan jumlah pasangan yang bisa berdansa.Dengan kata lain, kita mencari matching dengan ukuran terbesar (jumlah sisi terbanyak) pada graf bipartit.

Algoritma umum untuk mencari maximum bipartite matching adalah Algoritma Hopcroft-Karp.Algoritma ini menggunakan teknik depth-first search (DFS) untuk mencari augmenting path.. Algoritma Hopcroft-Karp memiliki kompleksitas waktu yang baik, yaitu O(sqrt(V) * E).

Langkah-langkah Dasar Hopcroft-Karp:

Inisialisasi:

- Mulai dengan matching kosong.
- Tandai semua vertex di sisi kiri sebagai "bebas" dan semua vertex di sisi kanan sebagai "belum dikunjungi".

Iterasi Pencarian Augmenting Path:
Augmenting path adalah jalur bergantian pada graf yang dimulai dari vertex "Kiri" dan berakhir di vertex "Kanan". bergantian disi maksudnya jalur yang sudah matching dan yang bebas

- Pilih vertex bebas di sisi kiri.
- Gunakan DFS untuk mencari augmenting path terpendek yang dimulai dari vertex tersebut.
- Jika jalur ditemukan, perbesar matching dengan mengikuti jalur tersebut.

Pengulangan:

- Ulangi langkah 2 sampai tidak ada lagi augmenting path yang ditemukan.

Matching Maksimum:

- Matching yang didapat pada akhir iterasi adalah Maximum Bipartite Matching.

Algoritma Maximum Bipartite Matching, seperti Hopcroft-Karp, menonjol dengan beberapa kelebihan yang membuatnya menjadi pilihan yang efisien dalam menyelesaikan masalah pencocokan maksimum dalam graf bipartit. Pertama, algoritma ini menunjukkan efisiensi waktu yang baik dengan kompleksitas membuatnya cocok untuk penanganan graf bipartit yang besar dan kompleks. Kelebihan lainnya adalah kemampuannya untuk memberikan solusi optimal, menghasilkan pencocokan maksimum dalam struktur data bipartit. Pendekatan menggunakan jalur meningkat memungkinkan algoritma untuk menyesuaikan diri dengan perubahan struktur graf atau pencocokan yang telah ada.

Namun, seiring dengan kelebihan tersebut, terdapat kekurangan yang perlu dipertimbangkan. Algoritma ini memiliki kompleksitas ruang yang tinggi, terutama untuk graf besar, dan ketergantungan pada algoritma BFS dapat menjadi faktor pembatas dalam beberapa situasi. Selain itu, algoritma ini hanya dirancang untuk graf bipartit, dan tidak memberikan hasil yang valid untuk struktur graf yang tidak memenuhi kriteria tersebut. Meskipun demikian, pemilihan Algoritma Maximum Bipartite Matching harus disesuaikan dengan karakteristik khusus dari masalah pencocokan yang dihadapi, serta mempertimbangkan trade-off antara kelebihan dan kekurangannya.

B. PSEUDO CODE

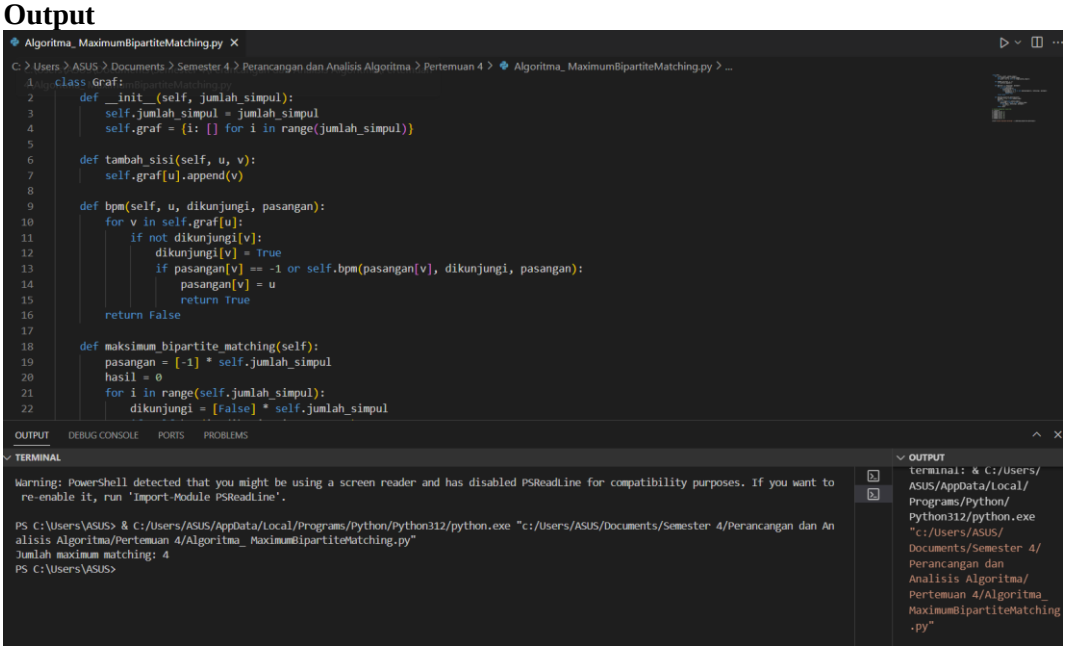
```
 Function maksimum_bipartite_matching(graf, jumlah_simpul):
  Inisialisasi pasangan dengan nilai -1 untuk setiap simpul
  hasil <- 0

  For setiap simpul i dari 0 hingga jumlah_simpul - 1:
    Dikunjungi <- array boolean dengan nilai False untuk setiap simpul
    Jika bpm(i, Dikunjungi, pasangan):
      hasil <- hasil + 1

  Return hasil

Function bpm(u, dikunjungi, pasangan):
  For setiap simpul v yang terhubung dengan u dalam graf:
    Jika v belum dikunjungi:
      Set dikunjungi[v] <- True
      Jika pasangan[v] == -1 atau bpm(pasangan[v], dikunjungi, pasangan):
        Set pasangan[v] <- u
        Return True
  Return False
```

C. SOURCE CODE

```
class Graf:
    def __init__(self, jumlah_simpul):
        self.jumlah_simpul = jumlah_simpul
        self.graf = {i: [] for i in range(jumlah_simpul)}

    def tambah_sisi(self, u, v):
        self.graf[u].append(v)

    def bpm(self, u, dikunjungi, pasangan):
        for v in self.graf[u]:
            if not dikunjungi[v]:
                dikunjungi[v] = True
                if pasangan[v] == -1 or self.bpm(pasangan[v],
dikunjungi, pasangan):
                    pasangan[v] = u
                    return True
        return False

    def maksimum_bipartite_matching(self):
        pasangan = [-1] * self.jumlah_simpul
        hasil = 0
        for i in range(self.jumlah_simpul):
            dikunjungi = [False] * self.jumlah_simpul
            if self.bpm(i, dikunjungi, pasangan):
                hasil += 1
        return hasil

# Contoh penggunaan algoritma
g = Graf(4)
g.tambah_sisi(0, 1)
```

```
g.tambah_sisi(0, 2)
g.tambah_sisi(1, 2)
g.tambah_sisi(2, 0)
g.tambah_sisi(2, 3)
g.tambah_sisi(3, 3)

print("Jumlah maximum matching:",
g.maksimum_bipartite_matching())
```

Output



D. ANALISIS KEBUTUHAN WAKTU

```
class Graf:
    def __init__(self, jumlah_simpul):
        self.jumlah_simpul = jumlah_simpul
        self.graf = {i: [] for i in range(jumlah_simpul)}
```

Analisis Menyeluruh:
Terdapat 2 operasi penugasan:

- self.jumlah_simpul = jumlah_simpul
- self.graf = {i: [] for i in range(jumlah_simpul)}

Operasi for dalam self.graf = {i: [] for i in range(jumlah_simpul)}

- Dilakukan sebanyak n kali
- Di dalam loop, terdapat 1 operasi penugasan []

Total operasi:

- 2 operasi penugasan
- n operasi penugasan di dalam loop for

Kompleksitas waktu: O(1 + n) = O(n)

Analisis Operasi Abstrak:

- Terdapat 1 operasi abstrak "inisialisasi" untuk self.jumlah_simpul
- Terdapat n operasi abstrak "inisialisasi" untuk setiap simpul dalam self.graf

Kompleksitas waktu: O(1 + n) = O(n)

Analisis Best-Case, Worst-Case, dan Average-Case:
Kompleksitas waktu algoritma inisialisasi graf tidak terpengaruh oleh struktur graf ataupun nilai input.

- Best-Case: O(n)
- Worst-Case: O(n)
- Average-Case: O(n)

```
def tambah_sisi(self, u, v):
    self.graf[u].append(v)
```

Analisis Menyeluruh:
Terdapat 2 operasi:
self.graf[u].append(v)

- Pencarian key u dalam dictionary self.graf
- Penambahan elemen v ke list yang terkait dengan key u

Analisis operasi append pada list bergantung pada implementasi internal list di Python, tetapi umumnya dianggap sebagai operasi konstan.

Kompleksitas waktu: O(1)

Analisis Operasi Abstrak:

- Terdapat 1 operasi abstrak "tambah sisi" yang menambahkan sisi (u, v) ke dalam graf.

Kompleksitas waktu: O(1)

Analisis Best-Case, Worst-Case, dan Average-Case:
Kompleksitas waktu algoritma tambah_sisi tidak terpengaruh oleh struktur graf ataupun nilai input u dan v.

- Best-Case: O(1)
- Worst-Case: O(1)
- Average-Case: O(1)

```
def bpm(self, u, dikunjungi, pasangan):
    for v in self.graf[u]:
        if not dikunjungi[v]:
            dikunjungi[v] = True
            if pasangan[v] == -1 or self.bpm(pasangan[v], dikunjungi, pasangan):
                pasangan[v] = u
                return True
    return False
```

Analisis Menyeluruh:
Loop for iterates through all neighbors of u.

- Jumlah iterasi loop for sama dengan derajat simpul u (jumlah sisi yang terhubung dengan u).
- Di dalam loop for:

- Operasi in pada dictionary dikunjungi: O(1)
- Operasi if: O(1)
- Operasi rekursif bpm: O(T(v)), di mana T(v) adalah kompleksitas waktu bpm untuk simpul v.
- Operasi penugasan: O(1)

Kompleksitas waktu:

- O(d(u) * (T(v) + 1))
- d(u) adalah derajat simpul u
- T(v) adalah kompleksitas waktu bpm untuk simpul v

Analisis Operasi Abstrak:

- Algoritma bpm melakukan operasi abstrak "pencocokan" untuk setiap simpul yang belum dikunjungi.

Kompleksitas waktu: O(m), di mana m adalah jumlah sisi dalam graf.

Analisis Best-Case, Worst-Case, dan Average-Case:

- Best-Case: O(n), di mana n adalah jumlah simpul dalam graf.
- Worst-Case: O(n^3), di mana n adalah jumlah simpul dalam graf.
- Average-Case: O(m^2), di mana m adalah jumlah sisi dalam graf.

```
def maksimum_bipartite_matching(self):
    pasangan = [-1] * self.jumlah_simpul
    hasil = 0
    for i in range(self.jumlah_simpul):
        dikunjungi = [False] * self.jumlah_simpul
        if self.bpm(i, dikunjungi, pasangan):
            hasil += 1
    return hasil
```

Analisis Menyeluruh:
Jumlah iterasi loop for sama dengan jumlah simpul dalam graf, yaitu n.
Di dalam loop for:

- Operasi bpm: O(T(i)), di mana T(i) adalah kompleksitas waktu bpm untuk simpul i.
- Operasi penugasan: O(1)

Kompleksitas waktu:
O(n * T(i))

- n adalah jumlah simpul dalam graf
- T(i) adalah kompleksitas waktu bpm untuk simpul i

Analisis Operasi Abstrak:

- Algoritma maksimum_bipartite_matching melakukan operasi abstrak "pencocokan maksimum" untuk seluruh simpul dalam graf.

Kompleksitas waktu: O(m^2), di mana m adalah jumlah sisi dalam graf.

- Analisis Best-Case, Worst-Case, dan Average-Case:
- Best-Case: O(n), di mana n adalah jumlah simpul dalam graf.
- Terjadi ketika graf terhubung dan setiap simpul memiliki derajat 1.
- Worst-Case: O(n^3), di mana n adalah jumlah simpul dalam graf.
- Terjadi ketika graf terhubung penuh dan semua simpul memiliki derajat n-1.
- Average-Case: O(m^2), di mana m adalah jumlah sisi dalam graf.

Secara keseluruhan
1. __init__:

- Kompleksitas waktu: O(n)
- Kompleksitas ruang: O(n)

2. tambah_sisi:

- Kompleksitas waktu: O(1)
- Kompleksitas ruang: O(1)

3. bpm:

- Kompleksitas waktu: O(m^2) (kasus rata-rata)
- Kompleksitas ruang: O(n)

4. maksimum_bipartite_matching:

- Kompleksitas waktu: O(n * m^2) (kasus rata-rata)
- Kompleksitas ruang: O(n)

Total:

- Kompleksitas waktu: O(n + m^2) (kasus rata-rata)
- Kompleksitas ruang: O(n)

E. REFERENSI

A. Levitin, 2012. Introduction to the design & analysis of algorithms. Available : https://zlib.pub/book/introduction-to-the-design-analysis-of-algorithms-90imffvmv4a0

Wikipedia - Big O notation: https://en.wikipedia.org/wiki/Big_O_notation

Tom G. (2020, Oktober 27). Bipartite Graphs and Maximum Matchings [Video]. YouTube. https://youtu.be/pmMwGyoGlXk

OptWhiz. (2022, November 1). Can we assign everyone a job? (maximum matchings) | Bipartite Matchings [Video]. YouTube. https://youtu.be/ELcgI_C1mNM?list=LL

F. LAMPIRAN LINK GITHUB

https://github.com/BonaFidee/Assigment/tree/0f48c993a932f6b10f14a6c0bb8036b8a2dcbaf4/Agil%20Abrar_202323430047