Tugas 1

Perancangan dan Analisis Algoritma

Oleh :

Muhamad Fathur Rahman
(21343055)

Dosen Pengampu :
Randi Proska Sandra, S.Pd., M.Sc.

PRODI TEKNIK INFORMATIKA

JURUSAN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2023

THE MAXIMUM FLOW PROBLEM

A. Pengertian Maximum Flow Problem

Maximum Flow Problem adalah masalah di mana kita mencari aliran maksimum (flow) yang dapat mengalir dari satu titik ke titik lain di dalam jaringan (network) yang telah ditentukan.

Jaringan (network) dalam konteks Maximum Flow Problem terdiri dari sejumlah simpul (node) yang dihubungkan oleh sejumlah jalur (edge) dengan kapasitas tertentu. Sebuah aliran (flow) adalah distribusi kapasitas yang dikirimkan melalui jalur-jalur tersebut dari satu simpul ke simpul lainnya, dimana setiap jalur hanya dapat mengirimkan kapasitas yang tidak melebihi kapasitas maksimum yang telah ditentukan.

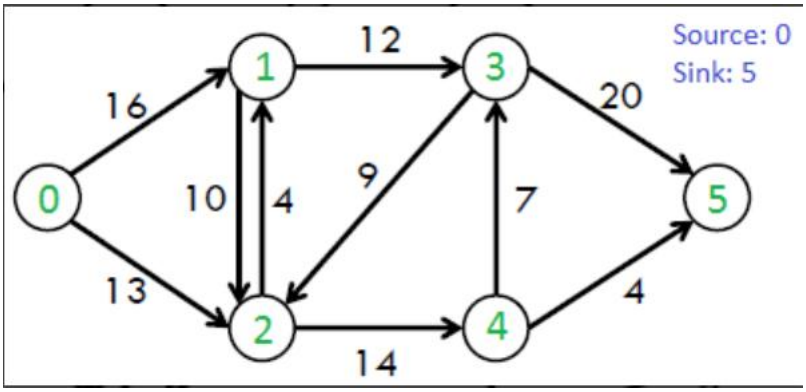
Tujuan dari Maximum Flow Problem adalah untuk mencari distribusi aliran (flow) yang memiliki jumlah total kapasitas yang maksimum dari satu simpul ke simpul lainnya. Masalah ini sering dijumpai dalam berbagai aplikasi nyata seperti perencanaan jaringan telekomunikasi, perancangan transportasi dan distribusi, serta optimasi rantai pasokan.

Untuk menyelesaikan masalah Maximum Flow Problem, terdapat beberapa teknik optimasi yang dapat digunakan, seperti algoritma Ford-Fulkerson, Edmonds-Karp, Dinic, atau Push-Relabel.

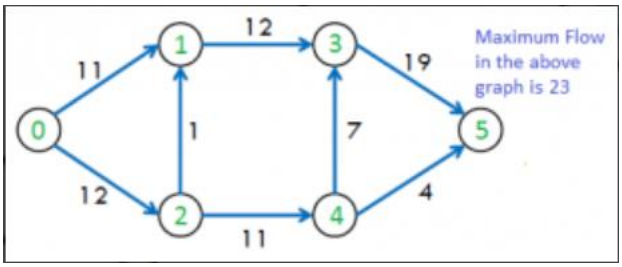
Secara formal, sebuah contoh dari masalah aliran maksimum ditentukan oleh beberapa unsur berikut ini:

• sebuah graf berarah G, dengan simpul-simpul V dan sisi-sisi berarah E;1
• sebuah simpul sumber s ∈ V ;
• sebuah simpul sink t ∈ V ;
• kapasitas nonnegatif dan integral ue untuk setiap sisi e ∈ E.

1. A Naive Greedy Algorithm



Setiap tepi diberi label dengan kapasitas, jumlah maksimum barang yang dapat dibawanya. Tujuannya adalah untuk mengetahui berapa banyak barang yang dapat didorong dari simpul s (source) ke simpul t (sink).



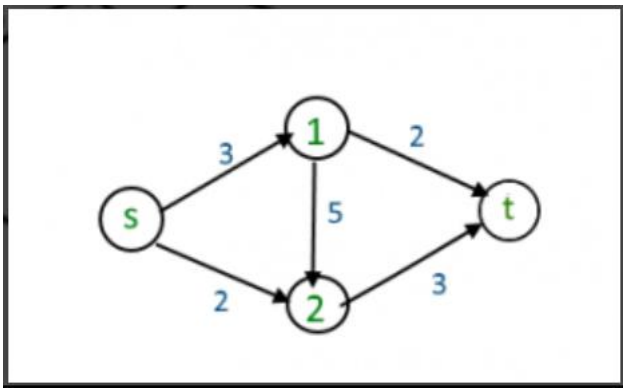
Maximum flow yang mungkin adalah : 23 Berikut adalah beberapa pendekatan untuk memecahkan masalah : 1. Pendekatan Algoritma Naif Greedy (Mungkin tidak menghasilkan hasil yang optimal atau benar) Pendekatan greedy untuk the maximum flow problem dimulai dengan semua aliran nol dan menghasilkan secara cepat mengalir dengan nilai yang semakin tinggi. Cara alami untuk melanjutkan dari satu ke yang berikutnya adalah dengan mengirimkan lebih banyak aliran pada beberapa jalur dari s ke t Bagaimana pendekatan Greedy bekerja untuk menemukan maximum flow :

A Naive Greedy Algorithm

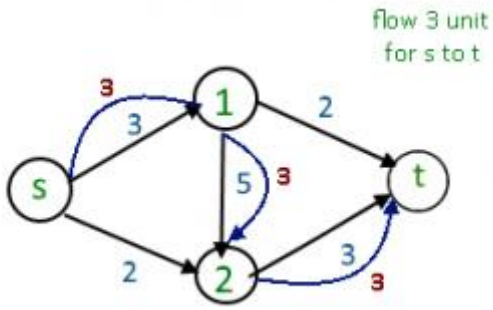
initialize $f_e = 0$ for all $e \in E$
repeat
search for an s-t path P such that $f_e < u_e$ for every $e \in P$
// takes $O(|E|)$ time using BFS or DFS
if no such path **then**
halt with current flow $\{f_e\}_{e \in E}$
else
let $\Delta = \min_{e \in P} \overbrace{(u_e - f_e)}^{\text{room on } e}$ (room on P)
for all edges e of P **do**
increase f_e by Δ

Perhatikan bahwa pencarian jalur hanya perlu menentukan apakah ada jalur s-t di sub-graf dari sisi-sisi e dengan fe < ue . Ini mudah dilakukan dalam waktu linear menggunakan sub-rutin pencarian graf favorit anda, seperti pencarian luas-pertama atau pencarian dalam pertama. Mungkin ada banyak jalur-jalur seperti itu; untuk saat ini, kita mengijinkan algoritma untuk memilih satu jalur secara acak. Algoritma kemudian mendorong aliran sebanyak mungkin pada jalur ini, sesuai dengan batasan kapasitas.

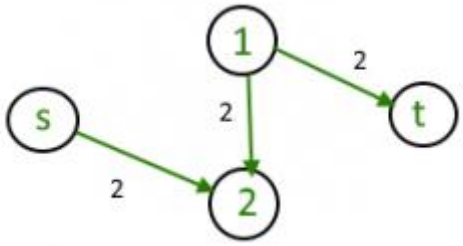
Perhatikan bahwa pencarian jalur hanya perlu menentukan apakah ada jalur s-t di subgraf tepi e dengan f(e) < C(e). Ini mudah dilakukan dalam waktu linier menggunakan BFS atau DFS.



Ada jalur dari sumber (s) ke sink(t) [s -> 1 -> 2 -> t] dengan aliran maksimum 3 unit (jalur ditampilkan dalam warna biru)



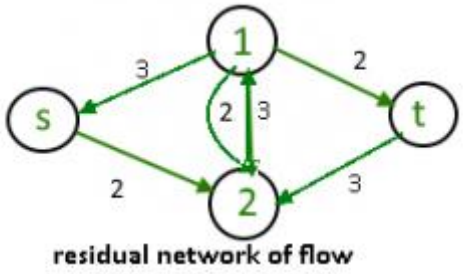
After removing all useless edge from graph it's look like



Untuk grafik di atas tidak ada jalur dari source ke sink sehingga maximum flow : 3 unit Tetapi maximum flow adalah 5 unit. untuk mengatasi masalah ini kami menggunakan Grafik sisa.

2. Residual Grap

Idenya adalah untuk memperluas algoritma serakah yang naif dengan mengizinkan operasi "undo". Misalnya, dari titik di mana algoritme ini macet di gambar di atas, kami ingin merutekan dua unit aliran lagi di sepanjang tepi (s, 2), lalu mundur di sepanjang tepi (1, 2), membatalkan 2 dari 3 unit kami merutekan iterasi sebelumnya, dan akhirnya di sepanjang tepi (1,t)



Bagian belakang : (f(e)) dan tepi depan : (C(e) – f(e)) Kita membutuhkan cara untuk secara formal menentukan operasi "undo" yang diperbolehkan. Ini memotivasi definisi sederhana namun penting berikut, dari jaringan residual. Idenya adalah, dengan memberikan graf G dan aliran f di dalamnya, kita membentuk jaringan aliran baru Gf yang memiliki himpunan simpul G yang sama dan memiliki dua sisi untuk setiap sisi dari G. Sebuah sisi e = (1,2) dari G yang membawa aliran f(e) dan memiliki kapasitas C(e) (untuk gambar di atas) memunculkan "tepi depan" dari Gf dengan kapasitas C(e)-f(e) (ruang tersisa) dan "mundur edge" (2,1) dari Gf dengan kapasitas f(e) (jumlah aliran yang dirutekan sebelumnya yang dapat dibatalkan). source(s)- sink(t) paths dengan f(e) < C(e) untuk semua edge, seperti yang dicari oleh algoritma naive greedy, sesuai dengan kasus khusus path s-t dari Gf yang hanya terdiri dari edge maju. Ide graf residual digunakan Algoritma Ford-Fulkerson dan Dinic Sumber : http://theory.stanford.edu/~tim/w16/l/l1.pdf Artikel ini disumbangkan oleh Nishant Singh . Jika Anda menyukai GeeksforGeeks dan ingin berkontribusi, Anda juga dapat menulis artikel menggunakan write.geeksforgeeks.org atau mengirimkan artikel Anda ke review-team@geeksforgeeks.org. Lihat artikel Anda muncul di halaman utama GeeksforGeeks dan bantu Geeks lainnya. Silakan tulis komentar jika Anda menemukan sesuatu yang salah, atau Anda ingin berbagi informasi lebih lanjut tentang topik yang

dibahas di atas.

3. The Ford-Fulkerson Algorithm

Ford-Fulkerson Algorithm

initialize $f_e = 0$ for all $e \in E$
repeat
  search for an s-t path P in the current residual graph G_f such that every edge of P has positive residual capacity
  // takes $O(|E|)$ time using BFS or DFS
  if no such path **then**
    halt with current flow $\{f_e\}_{e \in E}$
  else
    let $\Delta = \min_{e \in P}$ (e's residual capacity in G_f)
    // augment the flow f using the path P
    for all edges e of G whose corresponding forward edge is in P **do**
      increase f_e by Δ
    for all edges e of G whose corresponding reverse edge is in P **do**
      decrease f_e by Δ

Sebagai contoh, mulai dari jaringan residual pada Gambar 5, algoritma Ford-Fulkerson akan menambah aliran sebanyak unit di sepanjang jalur s → w → v → t. Penambahan ini menghasilkan aliran maksimum pada Gambar 1(b). Sekarang kita mengalihkan perhatian kita ke ketepatan algoritma Ford-Fulkerson. Kita akan membahas tentang mengoptimalkan waktu berjalan dalam kuliah selanjutnya

Kami mengklaim bahwa algoritma Ford-Fulkerson pada akhirnya akan berakhir dengan sebuah aliran yang layak. Hal ini mengikuti dari dua invarian, keduanya dibuktikan dengan induksi pada jumlah iterasi.

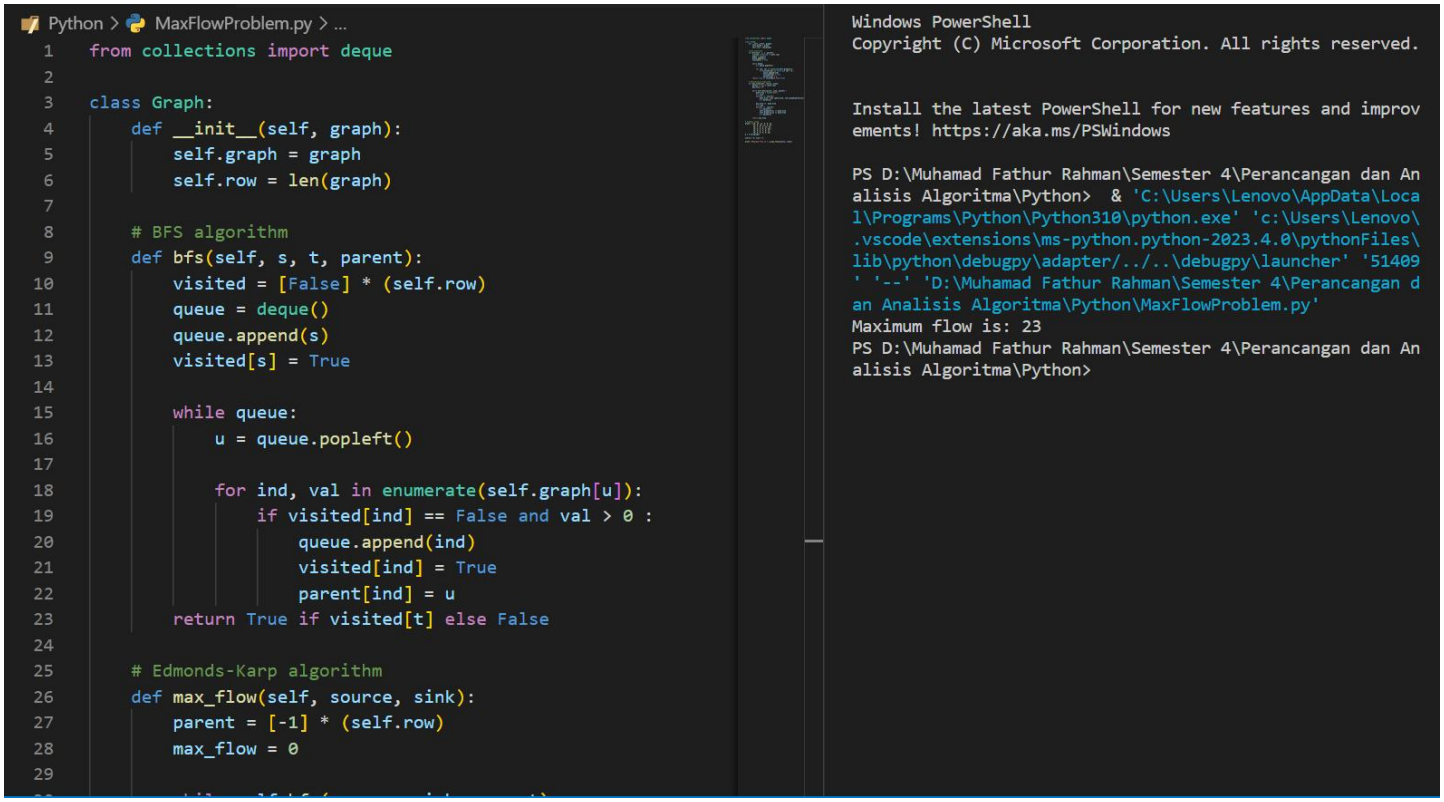
Pertama, algoritma ini mempertahankan invarian bahwa {f }ee∈E adalah sebuah aliran. Ini jelas benar pada awalnya. Parameter Δ didefinisikan sehingga tidak ada nilai aliran fe yang menjadi negatif atau melebihi kapasitas ue . Untuk batasan konservasi, perhatikan sebuah simpul v. Jika v tidak berada pada jalur penambahan P di Gf , maka aliran ke dalam dan ke luar v tetap sama. Jika v berada di P , dengan sisi-sisi (x, v) dan (v, w) milik P , maka ada empat kasus, tergantung apakah (x, v) dan (v, w) berkorespondensi dengan sisi-sisi maju atau mundur. Sebagai contoh, jika keduanya adalah sisi maju, maka penambahan aliran akan meningkatkan aliran ke dalam dan aliran keluar dari v meningkat sebesar Δ. Jika keduanya adalah sisi mundur, maka aliran masuk dan aliran keluar dari v berkurang sebesar Δ. Pada keempat kasus tersebut, aliran masuk dan aliran keluar berubah dengan jumlah yang sama, sehingga batasan konservasi terjaga.

Kedua, algoritma Ford-Fulkerson mempertahankan properti bahwa setiap jumlah aliran fe adalah bilangan bulat. (Ingat kita mengasumsikan bahwa setiap kapasitas tepi ue adalah sebuah bilangan bulat.) Secara induktif, semua kapasitas sisa adalah integral, sehingga parameter Δ adalah integral, sehingga arus tetap integral. Setiap iterasi dari algoritma Ford-Fulkerson meningkatkan nilai aliran arus sebesar nilai saat ini dari Δ. Invarian kedua mengimplikasikan bahwa Δ ≥ 1 dalam setiap iterasi dariAlgoritma Ford-Fulkerson. Karena hanya sejumlah aliran yang terbatas yang dapat keluar dari simpul sumber, algoritma Ford-Fulkerson pada akhirnya berhenti. Pada invarian pertama, algoritma ini berhenti dengan aliran yang layak.8 Tentu saja, semua ini juga berlaku untuk algoritma naif serakah pada Bagian 2.3. Bagaimana kita tahu apakah algoritma Ford-Fulkerson juga dapat berhenti dengan aliran yang tidak maksimum? Harapannya adalah karena algoritma Ford-Fulkerson

memiliki lebih banyak jalur yang memenuhi syarat untuk penambahan, algoritma ini akan berkembang lebih jauh sebelum berhenti.

B. Contoh Program Python

```
from collections import deque

class Graph:
    def __init__(self, graph):
        self.graph = graph
        self.row = len(graph)

    # BFS algorithm
    def bfs(self, s, t, parent):
        visited = [False] * (self.row)
        queue = deque()
        queue.append(s)
        visited[s] = True

        while queue:
            u = queue.popleft()

            for ind, val in enumerate(self.graph[u]):
                if visited[ind] == False and val > 0 :
                    queue.append(ind)
                    visited[ind] = True
                    parent[ind] = u
        return True if visited[t] else False

    # Edmonds-Karp algorithm
    def max_flow(self, source, sink):
        parent = [-1] * (self.row)
        max_flow = 0

        while self.bfs(source, sink, parent) :
            path_flow = float("Inf")
            s = sink
            while(s !=  source):
                path_flow = min (path_flow, self.graph[parent[s]][s])
                s = parent[s]

            max_flow +=  path_flow
            v = sink
            while(v !=  source):
                u = parent[v]
                self.graph[u][v] -= path_flow
                self.graph[v][u] += path_flow
                v = parent[v]

        return max_flow

# Example usage:
graph = [[0, 16, 13, 0, 0, 0],
         [0, 0, 10, 12, 0, 0],
         [0, 4, 0, 0, 14, 0],
         [0, 0, 9, 0, 0, 20],
         [0, 0, 0, 7, 0, 4],
         [0, 0, 0, 0, 0, 0]]
g = Graph(graph)

source = 0; sink = 5

print ("Maximum flow is:", g.max_flow(source, sink))
```



Program ini pertama-tama mendefinisikan kelas Graph yang menyimpan graf aliran dan metode untuk menjalankan algoritma BFS dan Edmonds-Karp. Kemudian program membuat objek Graph dengan graf aliran yang diberikan dalam bentuk matriks adjacency. Terakhir, program menjalankan algoritma Edmonds-Karp dengan memanggil metode max_flow pada objek Graph dan mencetak hasilnya. Dalam contoh ini, graf aliran terdiri dari 6 simpul dan 10 tepian dengan sumber pada simpul 0 dan tujuan pada simpul 5.

C. Pseudocode The Maximum Flow Problem

```
import deque from collections

class Graph:
    function __init__(self, graph):
        self.graph = graph
        self.row = len(graph)

    function bfs(self, s, t, parent):
        visited = [False] * (self.row)
        queue = deque()
        queue.append(s)
        visited[s] = True

        while queue is not empty:
            u = queue.popleft()

            for ind, val in enumerate(self.graph[u]):
                if visited[ind] is False and val > 0 :
                    queue.append(ind)
                    visited[ind] = True
                    parent[ind] = u
        return True if visited[t] else False

    function max_flow(self, source, sink):
        parent = [-1] * (self.row)
        max_flow = 0

        while self.bfs(source, sink, parent) is True:
            path_flow = float("Inf")
            s = sink
```

```
            while s is not source:
                path_flow = min (path_flow, self.graph[parent[s]][s])
                s = parent[s]

            max_flow += path_flow
            v = sink
            while v is not source:
                u = parent[v]
                self.graph[u][v] -= path_flow
                self.graph[v][u] += path_flow
                v = parent[v]

        return max_flow

# Example usage:
graph = [[0, 16, 13, 0, 0, 0],
         [0, 0, 10, 12, 0, 0],
         [0, 4, 0, 0, 14, 0],
         [0, 0, 9, 0, 0, 20],
         [0, 0, 0, 7, 0, 4],
         [0, 0, 0, 0, 0, 0]]
g = Graph(graph)

source = 0; sink = 5

print ("Maximum flow is:", g.max_flow(source, sink))
```

Pseudocode ini menggambarkan fungsi dan logika dari program dengan sintaksis yang lebih umum dan tidak tergantung pada bahasa pemrograman tertentu. Program ini mengimplementasikan algoritma Edmonds-Karp untuk mencari aliran maksimum pada suatu graf berarah dengan kapasitas pada setiap edge. Program ini menggunakan struktur data deque dari modul collections untuk implementasi BFS (Breadth-First Search).

Fungsi-fungsi yang terdapat dalam program ini adalah:

1. `__init__(self, graph)` : Constructor untuk class Graph. Inisialisasi atribut graph dan row dengan parameter graph (graf yang diberikan) dan panjang row (jumlah node/graf).
2. `bfs(self, s, t, parent)` : Implementasi BFS (Breadth-First Search) untuk mencari jalur dari source (s) ke sink (t) pada graf dengan menggunakan deque sebagai struktur data untuk queue. Fungsi ini akan mengembalikan nilai True jika ada jalur dari s ke t, dan False jika tidak ada.
3. `max_flow(self, source, sink)` : Implementasi algoritma Edmonds-Karp untuk mencari aliran maksimum pada graf dengan menggunakan BFS yang telah diimplementasikan di atas. Fungsi ini akan mengembalikan nilai dari aliran maksimum yang ditemukan.
4. `graph` : Atribut untuk menyimpan graf.
5. `row` : Atribut untuk menyimpan jumlah node/graf.
6. `visited` : List untuk menyimpan status visited (dikunjungi) dari setiap node/graf.
7. `queue` : Struktur data deque untuk menyimpan node/graf yang akan dikunjungi selanjutnya.
8. `parent` : List untuk menyimpan parent (node/graf sebelumnya) dari setiap node/graf.
9. `max_flow` : Variabel untuk menyimpan nilai aliran maksimum yang ditemukan.
10. `path_flow` : Variabel untuk menyimpan nilai kapasitas terkecil pada jalur yang ditemukan saat pencarian BFS.
11. `s` : Variabel untuk menyimpan nilai node/graf saat pencarian jalur dari sink ke source.
12. `v` : Variabel untuk menyimpan nilai node/graf saat memperbarui nilai kapasitas edge pada jalur yang ditemukan.
13. `u` : Variabel untuk menyimpan nilai node/graf saat memperbarui nilai kapasitas edge pada jalur yang ditemukan.

D. Referensi

1. https://www.geeksforgeeks.org/max-flow-problem-introduction/
2. http://theory.stanford.edu/~tim/w16/l/l1.pdf