TUGAS EKSPLORASI MANDIRI
PERANCANGAN DAN ANALISIS ALGORITMA

Algoritma The Maximum-Flow Problem

Oleh :

ALVIN ANUGERAH PRATAMA

(22343019)

Dosen Pengampu :

RANDI PROSKA SANDRA, S.Pd., M.Sc.

(Seksi : 202323430047)

PRODI INFORMATIKA

DEPARTEMEN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2024

A. Algoritma The Maximum-Flow Problem

The Maximum-Flow Problem, adalah masalah optimasi klasik dalam teori graf yang melibatkan pencarian jumlah flow maksimum yang dapat dikirim melalui jaringan pipes, channels, atau jalur lain dengan batasan kapasitas. Permasalahan-permasalahan tersebut dapat digunakan untuk beberbagai pemodelan berbagai macam situasi di dunia nyata. Seperti sistem transportasi, jaringan komunikasi, dan alokasi sumber daya.

Salah satu bentuk umum dari algortima Maximum-Flow Problem adalah algoritma Ford-Fulkerson, yang mana didasarkan pada gagasan penambahan jalur. Algoritma ini dimulai dengan flow awal bernilai nol, dan secara iteratif menemukan jalur dari s ke t yang memiliki kapasitas tersedia, dan kemudian meningkatkan aliran di sepanjang jalur sebanyak mungkin.

Keuntungan :

1. The Max-Flow Problem adalah alat pemodelan yang fleksibel dan kuat
2. Algoritma Ford-Fulkerson dijamin dapat menemukan aliran maksimum dalam grafik, dan dapat diimplementasikan secara efisien untuk sebagian besar kasus praktis
3. The Max-Flow Problem memiliki banyak sifat teoritis yang menarik dan koneksi ke bidang matematika, seperti pemograman linier dan operasi kombinatorial

Kekurangan :

1. Dalam beberapa kasus dulit dilaksanakan secara efisien, terutama jika grafiknya sangat besar atau memiliki batasan kapasistas yang kompleks
2. Tidak selalu memberikan solusi unik / optimal secara global, bergantung pada contoh masalah spesifik dan algoritma yang digunakan

Algoritma Ford-Fulkerson merupakan algoritma yang banyak digunakan untuk menyelesaikan Max-Flow Problem dalam suatu jaringan aliran. Algoritma ini bekerja dengan cara mencari jalur augmenting secara iteratif, yaitu jalur dari sumber ke sink pada grafik sisa, yaitu grafik yang diperoleh dengan mengurangkan aliran arus dari kapasitas masing-masing sisi. Algoritme kemudian meningkatkan aliran di sepanjang jalur ini dengan jumlah maksimum yang mungkin, yaitu kapasitas minimum dari tepi di sepanjang jalur.

B. Pseudocode

```
class Graf:
    inisialisasi(graf):
        self.graf = graf
        self.BARIS = panjang(graf)

    BFS(s, t, parent):
        dikunjungi = ArrayBoolean[False]*(self.BARIS)
        antrian = Queue()
        antrian.enqueue(s)
        dikunjungi[s] = True
        selama antrian tidak kosong:
            u = antrian.dequeue()
            untuk       setiap       (ind,       val)       di
enumerate(self.graf[u]):
                jika tidak dikunjungi[ind] dan val > 0:
                    antrian.enqueue(ind)
                    dikunjungi[ind] = True
                    parent[ind] = u
                    jika ind == t:
                        kembalikan Benar
        kembalikan False

    FordFulkerson(sumber, tujuan):
        parent = ArrayInt[-1]*(self.BARIS)
        max_aliran = 0
        selama BFS(sumber, tujuan, parent) bernilai Benar:
            aliran_path = Infinity
            s = tujuan
            selama s tidak sama dengan sumber:
                aliran_path       =       minimum(aliran_path,
self.graf[parent[s]][s])
                s = parent[s]
            max_aliran += aliran_path
            v = tujuan
            selama v tidak sama dengan sumber:
                u = parent[v]
                self.graf[u][v] -= aliran_path
                self.graf[v][u] += aliran_path
                v = parent[v]
        kembalikan max_aliran

graf = [[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 = Graf(graf)

sumber = 0
tujuan = 5

cetak("Aliran    maksimum    yang    mungkin    adalah    "    +
FordFulkerson(sumber, tujuan))
```

C. Source Code

```
1. from collections import defaultdict
2.
3. class Graf:
4.
5.     def __init__(self, graf):
6.         self.graf = graf
7.         self.BARIS = len(graf)
8.
9.     def BFS(self, s, t, parent):
10.              dikunjungi = [False]*(self.BARIS)
11.              antrian = []
12.              antrian.append(s)
13.              dikunjungi[s] = True
14.              while antrian:
15.                  u = antrian.pop(0)
16.                  for ind, val in enumerate(self.graf[u]):
17.                      if not dikunjungi[ind] and val > 0:
18.                          antrian.append(ind)
19.                          dikunjungi[ind] = True
20.                          parent[ind] = u
21.                          if ind == t:
22.                              return True
23.              return False
24.
25.          def FordFulkerson(self, sumber, tujuan):
26.              parent = [-1]*(self.BARIS)
27.              max_aliran = 0
28.              while self.BFS(sumber, tujuan, parent):
29.                  aliran_path = float("Inf")
30.                  s = tujuan
31.                  while(s != sumber):
32.                      aliran_path = min(aliran_path,
    self.graf[parent[s]][s])
33.                      s = parent[s]
34.                  max_aliran += aliran_path
35.                  v = tujuan
36.                  while(v != sumber):
37.                      u = parent[v]
38.                      self.graf[u][v] -= aliran_path
39.                      self.graf[v][u] += aliran_path
40.                      v = parent[v]
41.              return max_aliran
42.
43.      graf = [[0, 16, 13, 0, 0, 0],
44.              [0, 0, 10, 12, 0, 0],
45.              [0, 4, 0, 0, 14, 0],
46.              [0, 0, 9, 0, 0, 20],
47.              [0, 0, 0, 7, 0, 4],
48.              [0, 0, 0, 0, 0, 0]]
49.
50.      g = Graf(graf)
51.
52.      sumber = 0
53.      tujuan = 5
54.
55.      print("Aliran maksimum yang mungkin adalah %d" %
    g.FordFulkerson(sumber, tujuan))
56.
```

D. Analisis Kebutuhan Waktu

1. Analisis Menyeluruh

Inisialisasi:

a. Menciptakan matriks parent: O(V) operasi penugasan
b. Menginisialisasi variabel aliran maksimum: 1 operasi penugasan

BFS :

a. Menandai semua simpul sebagai tidak dikunjungi: O(V) operasi penugasan
b. Menambahkan simpul sumber ke antrian: 1 operasi penugasan
c. Loop while antrian tidak kosong:
   - Menandai semua simpul sebagai tidak dikunjungi: O(V) operasi penugasan
   - Menambahkan simpul sumber ke antrian: 1 operasi penugasan
   - Loop while antrian tidak kosong:
     - Menghapus simpul dari antrian: 1 operasi penghapusan
     - Melakukan loop untuk semua tetangga simpul: O(E) operasi
       - Memeriksa apakah simpul tetangga belum dikunjungi: O(1) operasi perbandingan
       - Jika belum dikunjungi:
         - Menandai simpul tetangga sebagai dikunjungi: 1 operasi penugasan
         - Menambahkan simpul tetangga ke antrian: 1 operasi penugasan
         - Memperbarui parent simpul tetangga: 1 operasi penugasan
     - Memeriksa apakah simpul tujuan tercapai: O(1) operasi perbandingan

Mencari Jalur Augmenting :

a. Menemukan aliran minimum di sepanjang jalur dari simpul tujuan ke simpul sumber: O(E) operasi
b. Memperbarui aliran di sepanjang jalur: O(E) operasi
c. Memperbarui parent simpul di sepanjang jalur: O(E) operasi

Total :

a. Inisialisasi: O(V)
b. BFS: O(V + E)
c. Mencari Jalur Augmenting: O(E)

Kompleksitas Waktu Keseluruhan :

O(V(V + E) + E^2)

2. Analisis Berdasarkan Operasi Abstrak

Algoritma Ford-Fulkerson melakukan operasi abstrak berikut:

a. Pencarian BFS: Dilakukan sekali per iterasi.
b. Pencarian Jalur Augmenting: Dilakukan sekali per iterasi.
c. Pembaruan aliran dan parent: Dilakukan per simpul di jalur augmenting.

Kompleksitas Waktu :
O(V(V + E) + E^2)

3. **Analisis Best-Case, Worst-Case, dan Average-Case**

a. Best-Case :
   - Jika terdapat jalur augmenting dengan kapasitas besar di setiap iterasi, maka algoritma akan berhenti dalam beberapa iterasi.
   - Kompleksitas waktu: O(V + E)

b. Worst-Case :
   - Kompleksitas waktu tergantung pada struktur jaringan dan nilai kapasitas.
   - Sulit untuk menentukan kompleksitas waktu rata-rata secara pasti.

c. Avarege-Case :
   - Kompleksitas waktu tergantung pada struktur jaringan dan nilai kapasitas.
   - Sulit untuk menentukan kompleksitas waktu rata-rata secara pasti.

E. Referensi

https://www.geeksforgeeks.org/max-flow-problem-introduction/?ref=lbp

https://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/?ref=lbp

https://www-geeksforgeeks-org.translate.goog/ford-fulkerson-algorithm-for-maximum-flow-problem/?_x_tr_sl=en&_x_tr_tl=id&_x_tr_hl=id&_x_tr_pto=tc

F. Link GitHub

https://github.com/Verruct51/The-Maximum-Flow-Problem.git