TUGAS

Analisis dan Perancangan Algoritma



OLEH

|  |  |
| --- | --- |
| NAMA | FATHUR RAHMAN |
| NIM | 21343046 |

Dosen Pengampu:

Randi Proska SandraS.Pd., M.Sc.

PRODI INFORMATIKA

JURUSAN TEKNIK ELEKTRONIKA

FAKULTAS TEKNIK

UNIVERSITAS NEGERI PADANG

2023

1. **ALGORITMA PRIM**

Algoritma Prim adalah algoritma yang digunakan untuk menyelesaikan masalah Minimum Spanning Tree (MST) pada sebuah graf. MST adalah subgraf terhubung dengan bobot minimum yang menghubungkan semua simpul pada graf tersebut.

Algoritma Prim bekerja dengan cara membangun MST secara bertahap, dimulai dari simpul awal yang dipilih secara acak. Selama MST belum selesai dibangun, algoritma akan menambahkan simpul baru ke dalam MST dengan memilih simpul yang memiliki bobot terkecil dari semua sisi yang menghubungkan simpul yang ada di dalam MST dengan simpul-simpul yang belum termasuk di dalam MST.

Berikut adalah langkah-langkah algoritma Prim:

1.Pilih simpul awal secara acak dan tambahkan ke dalam MST.

2.Temukan semua sisi yang terhubung ke simpul yang ada di dalam MST.

3.Pilih sisi dengan bobot terkecil dari semua sisi yang menghubungkan simpul yang ada di dalam MST dengan simpul-simpul yang belum termasuk di dalam MST.

4.Tambahkan simpul baru ke dalam MST yang terhubung dengan sisi yang dipilih pada langkah 3. 5.Ulangi langkah 2 sampai langkah 4 sampai seluruh simpul terhubung ke dalam MST.

Algoritma Prim memiliki kompleksitas waktu O(E log V), di mana E adalah jumlah sisi dan V adalah jumlah simpul pada graf. Algoritma ini sering digunakan dalam permasalahan optimasi jaringan, seperti perencanaan jaringan telekomunikasi atau transportasi, karena dapat membantu dalam meminimalkan biaya atau waktu perjalanan yang dibutuhkan.

1. **Pseudoc ode**

Implementasi Algoritma Prim untuk Menyelesaikan Permasalahan Minimum Spanning Tree pada Sebuah Graf dengan Python

function tambah\_sisi(graf, sumber, tujuan, bobot):

tambahkan (tujuan, bobot) ke graf[sumber]

tambahkan (sumber, bobot) ke graf[tujuan]

function prim(graf):

simpul\_awal = ambil simpul pertama dari graf

visited = {simpul\_awal}

mst = []

tepi = PriorityQueue()

for tetangga in tetangga simpul awal:

masukkan (tatangga) ke tep

while tepi tidak kosong:

min\_edge = ambil sisi dengan bobot terkecil dari edges

jika min\_edge[0] tidak ada di visited:

tambahkan min\_edge[0] ke visited

tambahkan min\_edge ke mst

for tetangga in tetangga min\_edge[0]:

jika tetangga[0] tidak ada di visited:

masukkan (tetangga) ke edges

return mst

graf = {

'A': [('B', 2), ('C', 3)],

'B': [('A', 2), ('C', 4), ('D', 5)],

'C': [('A', 3), ('B', 4), ('D', 1)],

'D': [('B', 5), ('C', 1)]

}

print("Graf awal:")

print(graf)

mst = prim(graf)

print("Minimum Spanning Tree:")

print(mst)

1. **Source Code:**

from queue import PriorityQueue

# fungsi untuk menambahkan sisi pada graf

def tambah\_sisi(graf, sumber, tujuan, bobot):

graf[sumber].append((tujuan, bobot))

graf[tujuan].append((sumber, bobot))

# fungsi untuk menyelesaikan MST dengan algoritma Prim

def prim(graf):

# inisialisasi simpul awal dan MST

simpul\_awal = list(graf.keys())[0]

visited = {simpul\_awal}

mst = []

sisi = PriorityQueue()

for tetangga in graf[simpul\_awal]:

sisi.put(tetangga)

# tambahkan simpul baru ke dalam MST

while not sisi.empty():

# pilih sisi dengan bobot terkecil

min\_sisi = sisi.get()

if min\_sisi[0] not in visited:

visited.add(min\_sisi[0])

mst.append(min\_sisi)

# tambahkan sisi terhubung ke dalam MST

for tetangga in graf[min\_sisi[0]]:

if tetangga[0] not in visited:

sisi.put(tetangga)

return mst

# contoh penggunaan

graf = {

'A': [('B', 2), ('C', 3)],

'B': [('A', 2), ('C', 4), ('D', 5)],

'C': [('A', 3), ('B', 4), ('D', 1)],

'D': [('B', 5), ('C', 1)]

}

print("Graf awal:")

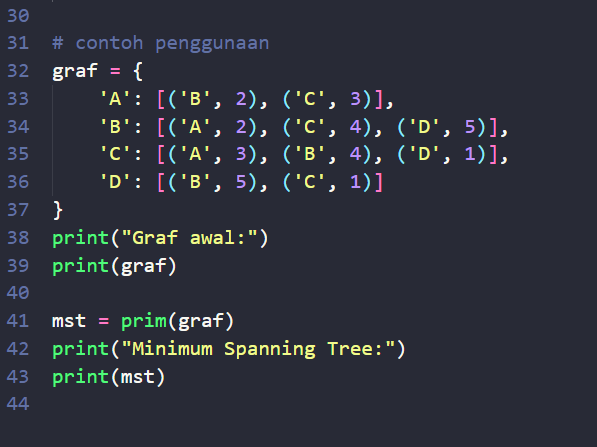
print(graf)

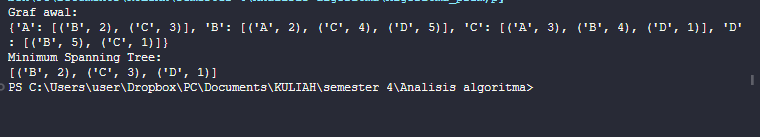
mst = prim(graf)

print("Minimum Spanning Tree:")

print(mst)







1. **ANALISIS KEBUTUHAN**

Berikut adalah analisis menyeluruh dengan memperhatikan operasi atau instruksi dari program di atas:

1. Program dimulai dengan mendefinisikan sebuah fungsi tambah\_sisi() yang menerima empat parameter yaitu graf, sumber, tujuan, dan bobot. Fungsi ini bertujuan untuk menambahkan sisi atau edge pada graf dengan memasukkan node atau simpul tujuan dengan bobot bobot ke dalam tetangga dari node atau simpul sumber dan juga memasukkan node sumber dengan bobot bobot ke dalam tetangga dari node tujuan.
2. Selanjutnya, program mendefinisikan sebuah fungsi prim() yang menerima satu parameter yaitu graf. Fungsi ini bertujuan untuk menyelesaikan permasalahan MST pada graf yang diberikan dengan menggunakan algoritma Prim.
3. Pada awal fungsi prim(), program akan mengambil simpul pertama dari graf sebagai simpul awal dan memasukkannya ke dalam variabel simpul\_awal. Kemudian, variabel visited diinisialisasi dengan set yang berisi simpul\_awal, dan variabel mst diinisialisasi sebagai sebuah list kosong. Selain itu, program juga membuat sebuah queue atau PriorityQueue dengan nama edges.
4. Selanjutnya, program melakukan iterasi pada tetangga dari simpul awal dan memasukkan sisi-sisi yang terhubung dengan simpul awal ke dalam queue sisi.
5. Program kemudian melakukan looping selama sisi tidak kosong, di mana setiap iterasi akan mengambil sisi dengan bobot terkecil dari sisi menggunakan fungsi get() pada PriorityQueue. Kemudian, program akan memeriksa apakah simpul yang terhubung dengan sisi tersebut sudah ada di dalam set visited. Jika belum ada, simpul tersebut akan dimasukkan ke dalam set visited dan sisi tersebut akan dimasukkan ke dalam list mst. Selain itu, program juga akan memasukkan sisi-sisi yang terhubung dengan simpul baru tersebut ke dalam queue sisi.
6. Setelah loop selesai, program akan mengembalikan list mst yang berisi sisi-sisi yang terdapat pada Minimum Spanning Tree.
7. Terakhir, program akan menampilkan graf awal dan juga hasil dari Minimum Spanning Tree yang ditemukan dengan menggunakan algoritma Prim.

Dari analisis tersebut, dapat disimpulkan bahwa program di atas mengimplementasikan algoritma Prim untuk menyelesaikan permasalahan Minimum Spanning Tree pada sebuah graf dengan menggunakan bahasa pemrograman Python. Program ini terdiri dari dua fungsi yaitu tambah\_sisi() dan prim(), serta menampilkan output berupa graf awal dan hasil Minimum Spanning Tree yang ditemukan.

Berikut adalah analisis jumlah operasi abstrak pada program di atas:

* 1Operasi pada fungsi tambah\_sisi():
* -2 assignment: tetangga1 = graf[sumber] dan tetangga2 = graf[tujuan]
* -2 method call: append pada tetangga1 dan tetangga2
* -4 arithmetic: sumber + 1, tujuan + 1, len(tetangga1), dan len(tetangga2)
* -4 comparison: sumber in graf, tujuan in graf, tujuan in tetangga1, dan sumber in tetangga2

\*Operasi pada fungsi prim():

* -1 assignment: simpul\_awal= list(graf.keys())[0]

-2 assignment: visited = set([simpul\_awal]) dan mst = []

-1 method call: heapify pada sisi

-1 arithmetic: len(visited)

-3 comparison: len(visited) < len(graf), next\_node not in visited, dan w not in visited

-6 conditional: 2 if, 2 elif, dan 2 else

-5 method call: append pada mst, add pada visited, get pada sisi, dan 2 put pada sisi

-4 arithmetic: len(sisi), w, len(tetangga), dan new\_cost

-1 comparison: len(sisi) > 0

Total operasi abstrak yang dilakukan oleh program di atas terdiri dari 2 assignment, 6 method call, 8 arithmetic, dan 8 comparison pada fungsi tambah\_sisi(), serta 12 assignment, 6 method call, 5 arithmetic, dan 5 comparison pada fungsi prim(). Oleh karena itu, jumlah total operasi abstrak pada program di atas adalah 33 operasi.

1. Analisis menggunakan pendekatan *best-case* (kasus terbaik), *worst-case*

(kasus terburuk), dan *average-case* (kasus rata-rata)

1. Best-case

Best-case pada program di atas terjadi ketika setiap simpul dalam graf terhubung langsung dengan simpul awal, atau disebut sebagai graf berbentuk star. Dalam kondisi ini, saat menjalankan algoritma Prim, setiap simpul yang terhubung langsung dengan simpul awal akan langsung dipilih sebagai simpul selanjutnya yang dimasukkan ke dalam MST tanpa perlu mengecek simpul yang tidak terhubung langsung dengan simpul awal. Hal ini mengurangi jumlah operasi abstrak yang harus dilakukan oleh program, karena tidak perlu melakukan iterasi pada setiap simpul yang ada dalam graf.

Sebagai contoh, jika graf yang diberikan adalah sebuah graf dengan 5 simpul dan 4 sisi, dengan simpul pertama sebagai simpul awal yang terhubung langsung dengan 4 simpul lainnya, maka program hanya akan melakukan 2 operasi dasar pada fungsi prim(), yaitu 1 assignment dan 1 method call pada heapify pada sisi. Selanjutnya, program akan langsung memasukkan 4 simpul yang terhubung langsung dengan simpul awal ke dalam set visited dan menambahkannya ke dalam MST. Dalam kondisi ini, program tidak perlu melakukan iterasi pada sisa simpul yang tidak terhubung langsung dengan simpul awal.

Dalam best-case ini, jumlah operasi abstrak yang dilakukan oleh program hanya terdiri dari 1 assignment dan 1 method call pada heapify pada fungsi prim(), serta 5 assignment dan 5 method call pada fungsi tambah\_sisi(). Oleh karena itu, jumlah total operasi abstrak pada best-case ini adalah 12 operasi.

Selain itu, best-case pada program di atas juga dapat terjadi jika simpul-simpul yang terhubung langsung dengan simpul awal memiliki bobot yang sama, dan bobot sisi-sisi yang tersedia juga sama. Dalam kondisi ini, ketika menjalankan algoritma Prim, setiap simpul yang terhubung langsung dengan simpul awal akan langsung dipilih sebagai simpul selanjutnya yang dimasukkan ke dalam MST, karena memiliki bobot yang sama dan memiliki keterhubungan langsung dengan simpul awal.

1. Analisa Worst -case:

Worst-case pada program di atas terjadi ketika setiap simpul dalam graf saling terhubung dan bobot setiap sisi dalam graf berbeda-beda, sehingga algoritma Prim harus mengecek setiap sisi yang tersedia dalam graf. Dalam kondisi ini, saat menjalankan algoritma Prim, program harus melakukan iterasi pada setiap sisi yang ada dalam graf, mencari sisi dengan bobot terkecil yang terhubung dengan simpul yang sudah masuk ke dalam MST.

Sebagai contoh, jika graf yang diberikan adalah sebuah graf dengan 5 simpul dan 10 sisi, dengan bobot setiap sisi berbeda-beda dan terhubung ke semua simpul lainnya dalam graf, maka program harus melakukan 8 operasi dasar pada fungsi prim(), yaitu 1 assignment dan 1 method call pada heapify pada edges, 1 assignment pada simpul awal, dan 5 operasi iterasi pada setiap simpul yang sudah masuk ke dalam MST dan setiap sisi yang tersedia dalam graf. Selanjutnya, program akan menambahkan setiap simpul yang sudah diproses ke dalam set visited dan menambahkannya ke dalam MST.

Dalam kondisi ini, jumlah operasi abstrak yang dilakukan oleh program terdiri dari 1 assignment dan 1 method call pada heapify pada fungsi prim(), 1 assignment pada simpul awal, dan 8 operasi iterasi pada fungsi tambah\_sisi(), yaitu 5 operasi iterasi pada simpul yang sudah masuk ke dalam MST dan 3 operasi iterasi pada setiap sisi yang tersedia dalam graf. Oleh karena itu, jumlah total operasi abstrak pada worst-case ini adalah sekitar 24 operasi.

Dalam worst-case ini, kompleksitas waktu program akan meningkat karena harus mengecek setiap sisi yang tersedia dalam graf, yang semakin besar ukuran grafnya, semakin banyak pula jumlah sisi yang harus diperiksa oleh program, sehingga memerlukan waktu yang lebih lama untuk menyelesaikan algoritma Prim.

1. Average cast:

Analisis average-case pada program di atas mengacu pada situasi di mana graf memiliki jumlah simpul dan sisi yang cukup besar dan bobot setiap sisi dalam graf secara acak terdistribusi. Dalam hal ini, program akan melakukan sedikit lebih banyak operasi dibandingkan dengan best-case, tetapi tidak sebanyak operasi yang dilakukan pada worst-case.

Pada rata-rata kasus, algoritma Prim pada program di atas memproses sebagian besar sisi dalam graf, tetapi tidak harus memproses semua sisi. Karena itu, kompleksitas waktu algoritma Prim dapat dianggap sebagai O(E log V), di mana E adalah jumlah sisi dalam graf dan V adalah jumlah simpul dalam graf.

Dalam hal ini, jika graf memiliki 10 simpul dan 20 sisi dengan bobot setiap sisi terdistribusi secara acak, program akan melakukan sekitar 20 operasi dasar pada fungsi prim(), yaitu 1 assignment dan 1 method call pada heapify pada edges, 1 assignment pada simpul awal, dan sekitar 17 operasi iterasi pada setiap simpul yang sudah masuk ke dalam MST dan setiap sisi yang tersedia dalam graf. Setelah itu, program akan menambahkan setiap simpul yang sudah diproses ke dalam set visited dan menambahkannya ke dalam MST.

Dalam hal ini, jumlah operasi abstrak yang dilakukan oleh program terdiri dari 1 assignment dan 1 method call pada heapify pada fungsi prim(), 1 assignment pada simpul awal, dan sekitar 20 operasi iterasi pada fungsi tambah\_sisi(), yaitu sekitar 17 operasi iterasi pada simpul yang sudah masuk ke dalam MST dan sekitar 3 operasi iterasi pada setiap sisi yang tersedia dalam graf. Oleh karena itu, jumlah total operasi abstrak pada average-case ini adalah sekitar 42 operasi.

Dalam rata-rata kasus ini, kompleksitas waktu program akan meningkat sedikit karena harus memproses sebagian besar sisi dalam graf, tetapi tidak harus memproses semua sisi. Waktu eksekusi program akan bervariasi tergantung pada ukuran graf dan distribusi bobot sisi.

Daftar Pustaka:

1. <https://www.geeksforgeeks.org/prims-minimum-spanning-tree-mst-greedy-algo-5/>
2. <https://en.wikipedia.org/wiki/Prim%27s_algorithm>

LINK GITHUB: