UAS
ALGORITMA DAN STRUKTUR DATA

Disusun oleh Syahrul Akbar Ramdhani
(1123094000000270
DOSEN PENGAMPU: M. Irvan Sepriar Musti, M.si

Soal 1 - Truncatable & Rotatable Prime Challenge

1. Truncatable Alternatif

- Jika kita menghapus digit secara bergantian dari kiri, kanan, kiri, kanan, ... hingga habis (contoh: 739391 → 39391 → 3939 → 939 → 93 → 3),
- Setiap hasil penghapusan adalah bilangan prima.

2. Rotatable

- Semua rotasi siklis ke kiri dari nnn (contoh: 197 → 971 → 719) juga harus prima.

3. Mirror-Truncatable

- Buat cermin nnn dengan membalik urutannya (contoh: 739391 → 193937).
- Ulangi proses (1) untuk hasil cermin tersebut; setiap hasil pemotongan alternatifnya juga harus prima.

SOAL 1 - TRUNCATABLE & ROTATABLE PRIME CHALLENGE

Cek Bilangan Prima dan punya 0

```python
def is_prime(n: int) -> bool:
    if n <= 1: return False
    if n == 2: return True
    if n % 2 == 0: return False
    for i in range(3, int(n ** 0.5) + 1, 2):
        if n % i == 0:
            return False
    return True
```

```python
def has_zero(num: int) -> bool:
    while num > 0:
        if num % 10 == 0:
            return True
        num //= 10
    return False
```

```python
def alt_trunc_prime(n: int) -> bool:
    digits = []
    temp = n
    while temp > 0:
        digits.append(temp % 10)
        temp //= 10
    digits.reverse()

    left_stack = deque(digits)
    right_stack = deque(digits)

    turn_left = True
    while left_stack:
        # Ambil dari stack sesuai giliran
        current_digits = []
        temp_stack = deque(left_stack)
        while temp_stack:
            current_digits.append(temp_stack.popleft())
        # Konversi list ke angka
        num = 0
        for d in current_digits:
            num = num * 10 + d
        if not is_prime(num):
            return False
        # Pop dari kiri atau kanan
        if turn_left:
            left_stack.popleft()
        else:
            left_stack.pop()
        turn_left = not turn_left
    return True
```

Soal 1 - Truncatable & Rotatable Prime Challenge

- Truncatable Alternatif

– Jika kita menghapus digit secara bergantian dari kiri, kanan, kiri, kanan, … hingga habis (contoh: 739391 → 39391 → 3939 → 939 → 93 → 3),

– Setiap hasil penghapusan adalah bilangan prima.

```python
def all_rotations_prime(n: int) -> bool:
    digits = deque()
    temp = n
    while temp > 0:
        digits.appendleft(temp % 10)
        temp //= 10

    for _ in range(len(digits)):
        # Convert deque to number
        num = 0
        for d in digits:
            num = num * 10 + d
        if not is_prime(num):
            return False
        # Rotate left
        digits.append(digits.popleft())
    return True
```

SOAL 1 - TRUNCATABLE & ROTATABLE PRIME CHALLENGE

- Rotatable:

- Semua rotasi siklis ke kiri dari nnn (contoh: 197 → 971 → 719) juga harus prima

SOAL 1 - TRUNCATABLE & ROTATABLE PRIME CHALLENGE

```python
def reverse_number(n: int) -> int:
    rev = 0
    while n > 0:
        rev = rev * 10 + (n % 10)
        n //= 10
    return rev
```

- Mirror-Truncatable

– Buat cermin nnn dengan membalik urutannya (contoh: 739391 → 193937).

– Ulangi proses (1) untuk hasil cermin tersebut; setiap hasil pemotongan alternatifnya juga harus prima.

SOAL 1 - TRUNCATABLE & ROTATABLE PRIME CHALLENGE

```python
def classify_bpt(num: int) -> str:
    if not is_prime(num):
        return "not prime"
    if has_zero(num):
        return "has zero"
    if not alt_trunc_prime(num):
        return "fail alt-trunc"
    if not all_rotations_prime(num):
        return "fail rotation"
    mirror = reverse_number(num)
    if not alt_trunc_prime(mirror):
        return "fail mirror"
    return "valid"
```

print(classify_bpt(739)) # fail alt-trunc

digits = [7, 3, 9]
left_stack = deque([7, 3, 9])
turn_left = True

# Iterasi 1:
current_digits = [7, 3, 9] → angka: 739 → is_prime ✅
pop left → deque = [3, 9]

# Iterasi 2:
current_digits = [3, 9] → angka: 39 → is_prime ❌
return False

```python
def classify_bpt(num: int) -> str:
    if not is_prime(num):
        return "not prime"
    if has_zero(num):
        return "has zero"
    if not alt_trunc_prime(num):
        return "fail alt-trunc"
    if not all_rotations_prime(num):
        return "fail rotation"
    mirror = reverse_number(num)
    if not alt_trunc_prime(mirror):
        return "fail mirror"
    return "valid"
```

SOAL 1 - TRUNCATABLE & ROTATABLE PRIME CHALLENGE

print(classify_bpt(137)) # fail rotation

Rotasi:

1. 137 → prima ✅
2. 371 → bukan prima ❌

```python
def classify_bpt(num: int) -> str:
    if not is_prime(num):
        return "not prime"
    if has_zero(num):
        return "has zero"
    if not alt_trunc_prime(num):
        return "fail alt-trunc"
    if not all_rotations_prime(num):
        return "fail rotation"
    mirror = reverse_number(num)
    if not alt_trunc_prime(mirror):
        return "fail mirror"
    return "valid"
```

SOAL 1 - TRUNCATABLE & ROTATABLE PRIME CHALLENGE

print(classify_bpt(137)) # fail rotation

Rotasi:

1. 137 → prima ✅
2. 371 → bukan prima ❌

```python
def classify_bpt(num: int) -> str:
    if not is_prime(num):
        return "not prime"
    if has_zero(num):
        return "has zero"
    if not alt_trunc_prime(num):
        return "fail alt-trunc"
    if not all_rotations_prime(num):
        return "fail rotation"
    mirror = reverse_number(num)
    if not alt_trunc_prime(mirror):
        return "fail mirror"
    return "valid"
```

SOAL 1 - TRUNCATABLE & ROTATABLE PRIME CHALLENGE

print(classify_bpt(337))

classify_bpt(337)
↓
reverse_number(337) = 733
↓
alt_trunc_prime(733) = False (karena 33 bukan prima)
↓
⛔ Gagal di mirror
↓
"fail mirror"

Soal 2 - Perjalanan Tak Terlupakan

Buat kelas Node

```
class Node:
  def __init__(self, destinasi, jarak_destinasi_selanjutnya):
    self.destinasi = destinasi
    self.jarak_destinasi_selanjutnya = jarak_destinasi_selanjutnya
    self.next = None
```

Soal 2 - Perjalanan Tak Terlupakan

Buat kelas Node dan inisiasi kelas History_Perjalanan

```
class Node:
  def __init__(self, destinasi, jarak_destinasi_selanjutnya):
    self.destinasi = destinasi
    self.jarak_destinasi_selanjutnya = jarak_destinasi_selanjutnya
    self.next = None

class History_Perjalanan:
  def __init__(self):
    self.head = None
    self.total_tempat_dikunjungi = 0
```

Program ini mencatat perjalanan ke berbagai destinasi menggunakan struktur data linked list. Setiap node berisi nama tempat dan jarak ke tempat berikutnya.

Soal 2 - Perjalanan Tak Terlupakan

Definisikan tempat_baru

```
def tempat_baru(self, destinasi, jarak_destinasi_selanjutnya):
  baru = Node(destinasi, jarak_destinasi_selanjutnya)
  if self.head is None:
    self.head = baru
  else:
    cek = self.head
    while cek.next != None:
      cek = cek.next
    cek.next = baru
  self.total_tempat_dikunjungi += 1
```

tempat_baru(): Menambahkan simpul (node) baru ke akhir linked list yang mewakili destinasi baru yang dikunjungi. Jarak ke destinasi selanjutnya juga dicatat.

Soal 2 - Perjalanan Tak Terlupakan

Definisikan list_kunjungan

```
def list_kunjungan(self):
  print(f"Total Tempat Di Kunjungi : {self.total_tempat_dikunjungi}\n")
  cek = self.head
  i = 1
  while cek != None:
    print(f"tempat ke-{i}: {cek.destinasi}")
    cek = cek.next
    i += 1
```

list_kunjungan(): Menampilkan jumlah tempat yang telah dikunjungi dan mencetak daftar tempat beserta urutannya.

Soal 2 - Perjalanan Tak Terlupakan

Definisikan total_jarak dan _info

```
def total_jarak(self):
  cek = self.head
  total_jarak = 0
  while cek.next != None:
    total_jarak += cek.jarak_destinasi_selanjutnya
    cek = cek.next
  print(f"\nTotal Jarak : {total_jarak} km\n")

def info(self):
  self.list_kunjungan()
  self.total_jarak()
```

total_jarak(): mencatat semua jarak yang telah kita lalui,
_info(): melihat apa saja yang sudah kita kunjungi dan semua jarak yang telah kita lalui

Soal 2 - Perjalanan Tak Terlupakan

```
list_liburan = [
{'destinasi': 'Museum Kota', 'jarak_destinasi_selanjutnya': 2},
{'destinasi': 'Taman Kota', 'jarak_destinasi_selanjutnya': 1.5},
{'destinasi': 'Pantai Indah', 'jarak_destinasi_selanjutnya': 3},
{'destinasi': 'Kebun Binatang', 'jarak_destinasi_selanjutnya': 4.5},
{'destinasi': 'Pusat Perbelanjaan', 'jarak_destinasi_selanjutnya': 2.5},
{'destinasi': 'Menara Kota', 'jarak_destinasi_selanjutnya': 1}]

liburan = History_Perjalanan()

for i in list_liburan:
  liburan.tempat_baru(i['destinasi'], i['jarak_destinasi_selanjutnya'])

liburan.info()
```

Total Tempat Di Kunjungi : 6
tempat ke-1: Museum Kota
tempat ke-2: Taman Kota
tempat ke-3: Pantai Indah
tempat ke-4: Kebun Binatang
tempat ke-5: Pusat Perbelanjaan
tempat ke-6:
Menara Kota T
otal Jarak : 13.5 km

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Simulasi sistem boarding penumpang kereta cepat:

- 60 kursi per keberangkatan (tiap 15 menit)
- Ribuan penumpang dari 3 kelas: Premium, Business, Economy
- Aturan harus dipatuhi:
  - FIFO per kelas
  - Batas tunggu → voucher
  - Fast-track queue
  - Kursi minimal per kelas
  - Penyeimbang ketimpangan antrian
  - Penanganan kelompok penumpang

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

```python
class Passenger:
    def __init__(self, id, name, ticket_class, arrival_time, group_id):
        self.id = int(id)
        self.name = name
        self.ticket_class = ticket_class
        self.arrival_time = time_to_minutes(arrival_time)
        self.arrival_minute = time_to_minutes(arrival_time)
        self.group_id = group_id
        self.has_voucher = False

    def __repr__(self):
        return f"{self.id}-{self.ticket_class}-{self.arrival_time}"
```

Setiap penumpang punya:

- ID dan nama
- Kelas tiket: Premium / Business / Economy
- Waktu datang (dalam menit)
- Status has_voucher

📚 Gunakan deque() untuk antrian FIFO:

- premium_q, business_q, economy_q, voucher_q

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

```python
def time_to_minutes(t: str) -> int:
    h, m = map(int, t.split(":"))
    return h * 60 + m

def minutes_to_time(m: int) -> str:
    return f"{m // 60:02}:{m % 60:02}"
```

- Konversi 08:30 → 510 (dalam menit)
- Konversi kembali → 08:30

```python
def load_passenger_csv(filename: str):
    df = pd.read_csv(filename)
    passengers = []
    for _, row in df.iterrows():
        p = Passenger(
            id=row['id'],
            name=row['name'],
            ticket_class=row['ticket_class'],
            arrival_time=row['arrival_time'],
            group_id=row['group_id'] if row['group_id'] != '-' else None
        )
        passengers.append(p)
    return passengers
```

- Membaca file CSV berisi data penumpang
- Mengembalikan list Passenger

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Definisikan Proses keberangkatan

```
def process_boarding(departure_time: str, queues: dict, voucher_q: deque) -> list:
    capacity = 60
    min_limit = {"Economy": 9, "Business": 15, "Premium": 0}
    class_boarded = {"Premium": 0, "Business": 0, "Economy": 0}
    boarded = []
    seen_ids = set()
```

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

loop untuk fast track dan penuhi batas minimum kursi

```
# 1. Prioritaskan fast-track voucher queue
while voucher_q and len(boarded) < capacity:
    p = voucher_q.popleft()
    if p.id not in seen_ids:
        boarded.append(p)
        seen_ids.add(p.id)
        class_boarded[p.ticket_class] += 1

# 2. Penuhi batas minimal kursi
for cls in ["Economy", "Business"]:
    while class_boarded[cls] < min_limit[cls] and queues[cls] and len(boarded) < capacity:
        p = queues[cls].popleft()
        if p.id not in seen_ids:
            boarded.append(p)
            seen_ids.add(p.id)
            class_boarded[cls] += 1
```

1. Fast-track voucher dulu
2. Penuhi batas minimum kursi:
   - Economy ≥ 9
   - Business ≥ 15

```python
# 3. Penyeimbang ketidakseimbangan
qlens = {cls: len(queues[cls]) for cls in ["Premium", "Business", "Economy"]}
imbalance_order = ["Economy", "Business", "Premium"]
for cls in imbalance_order:
    other_total = sum(qlens[other] for other in qlens if other != cls)
    if qlens[cls] > 2 * other_total:
        total_allowable = 12  # max 20% pinjaman
        for donor in qlens:
            if donor == cls:
                continue
            donor_min = min_limit[donor]
            donor_current = class_boarded[donor]
            donor_max_avail = max(0, donor_current - donor_min)
            n_borrow = min(total_allowable, donor_max_avail, len(queues[cls]),
                           capacity - len(boarded))
            for _ in range(n_borrow):
                if queues[cls]:
                    p = queues[cls].popleft()
                    if p.id not in seen_ids:
                        boarded.append(p)
                        seen_ids.add(p.id)
                        class_boarded[cls] += 1
                        total_allowable -= 1
                if total_allowable == 0 or len(boarded) >= capacity:
                    break
        if len(boarded) >= capacity:
            break
```

SOAL 3 - MANAJEMEN ANTRIAN DI TERMINAL KERETA CEPAT

3. Penyeimbang antrian timpang:
   - Economy > 2×(P+B) → pinjam kursi
   - Max pinjam 12 kursi (20%)

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

loop untuk sisa kursi

```
# 4. Isi sisa kursi
for cls in ["Premium", "Business", "Economy"]:
    while queues[cls] and len(boarded) < capacity:
        p = queues[cls].popleft()
        if p.id not in seen_ids:
            boarded.append(p)
            seen_ids.add(p.id)
            class_boarded[cls] += 1

return boarded
```

4. Sisa kursi: Premium > Business > Economy

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Persiapan dan Inisialisasi Data untuk simulasi hari

```
def simulate_day(passenger_list):
    passengers = sorted(passenger_list, key=lambda x: x.arrival_time)

    group_map = defaultdict(list)
    for p in passengers:
        if p.group_id:
            group_map[p.group_id].append(p)
```

- Mengurutkan penumpang berdasarkan waktu kedatangan.
- Mengelompokkan penumpang berdasarkan group_id (jika ada).

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Inisialisasi Struktur Antrian dan Variabel Penting

```
premium_q = deque()
business_q = deque()
economy_q = deque()
voucher_q = deque()

all_boarded_ids = set()
over_waited = []

time_now = time_to_minutes("08:00")
end_time = time_to_minutes("24:00")
train_number = 1
passenger_idx = 0
wait_limit = {"Premium": 15, "Business": 30, "Economy": 45}
```

- Menyiapkan antrian berdasarkan kelas dan voucher.
- Menyiapkan waktu simulasi, nomor kereta, dan batas waktu tunggu untuk tiap kelas.

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Loop Simulasi Setiap 15 Menit dan Memasukkan Penumpang Baru ke Antrian

```
while time_now <= end_time:
    while passenger_idx < len(passengers) and passengers[passenger_idx].arrival_minute <= time_now:
        p = passengers[passenger_idx]
        if p.has_voucher:
            voucher_q.append(p)
        elif p.ticket_class == "Premium":
            premium_q.append(p)
        elif p.ticket_class == "Business":
            business_q.append(p)
        else:
            economy_q.append(p)
        passenger_idx += 1
```

- Memproses kedatangan penumpang dan keberangkatan kereta setiap 15 menit, dari jam 08:00 sampai 24:00.
- Menambahkan penumpang yang telah tiba ke antrian sesuai kelas atau ke voucher queue jika punya fast-track.

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Deteksi Over-Wait dan Pemberian Voucher

```
for cls, q in zip(["Premium", "Business", "Economy"], [premium_q, business_q, economy_q]):
    for p in list(q):
        if time_now - p.arrival_time > wait_limit[cls] and not p.has_voucher:
            p.has_voucher = True
            q.remove(p)
            voucher_q.append(p)
            over_waited.append(p)
```

📌 Setiap 15 menit:

- Penumpang baru yang datang → masuk antrian sesuai kelas
- Cek over-wait: Premium >15 m, Business >30 m, Economy >45 m

📌 Jika over-wait:

- Keluarkan dari antrian reguler
- Tambahkan ke voucher_q
- Tandai has_voucher = True

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Proses Boarding Kereta

```
queues = {
    "Premium": premium_q,
    "Business": business_q,
    "Economy": economy_q
}

boarded = process_boarding(minutes_to_time(time_now), queues, voucher_q)

for b in boarded:
    all_boarded_ids.add(b.id)
```

- Memanggil fungsi process_boarding untuk memilih 60 penumpang dari antrian dan voucher.
- Menandai siapa saja yang sudah naik ke kereta.

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Mencetak Log Keberangkatan

```
if boarded:
    log = f"--- Keberangkatan #{train_number} ({minutes_to_time(time_now)}) ---\n"
    for cls in ['Premium', 'Business', 'Economy']:
        ids = [str(p.id) for p in boarded if p.ticket_class == cls]
        log += f"{cls:<8}: {len(ids)} penumpang  (id: {', '.join(ids)})\n"
    vch = len([p for p in boarded if p.has_voucher])
    log += f"Fast-Track Voucher : {vch}\n"
    log += f"Kursi Kosong       : {60 - len(boarded)}\n"
    yield log
    train_number += 1

time_now += 15
```

- Jika ada keberangkatan, buat log rincian:
  - Jumlah dan ID penumpang per kelas
  - Jumlah pemegang voucher
  - Kursi kosong
- Lanjut ke interval waktu berikutnya (tiap 15 menit).

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Laporan Penumpang Over-Wait

```
if over_waited:
    log = "\n--- Daftar Penumpang Menerima Voucher (Over-wait) ---\n"
    for p in over_waited:
        log += f"ID {p.id} ({p.name}), Kelas: {p.ticket_class}, Tiba: {minutes_to_time(p.arrival_time)}\n"
    yield log
```

- Setelah semua simulasi selesai, tampilkan siapa saja yang menerima voucher karena menunggu terlalu lama.

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

Contoh Output

```
==============================
📅 Tanggal : 2025-07-21
🚄 Kereta  : KC-Shinobi (60 kursi / 15 m)
==============================
--- Keberangkatan #1 (08:00) ---
Premium : 0 penumpang  (id: )
Business: 1 penumpang  (id: 1)
Economy : 0 penumpang  (id: )
Fast-Track Voucher : 0
Kursi Kosong        : 59

--- Keberangkatan #2 (08:15) ---
Premium : 2 penumpang  (id: 2, 10)
Business: 1 penumpang  (id: 7)
Economy : 6 penumpang  (id: 3, 4, 5, 6, 8, 9)
Fast-Track Voucher : 0
Kursi Kosong        : 51

--- Keberangkatan #3 (08:30) ---
Premium : 3 penumpang  (id: 13, 15, 18)
Business: 2 penumpang  (id: 14, 17)
Economy : 6 penumpang  (id: 11, 12, 16, 19, 20, 21)
Fast-Track Voucher : 0
Kursi Kosong        : 49
```

- kereta pertama berangkat jam 8
- 1 penumpang naik yang memenuhi syarat boarding
- tidak ada fast track

Soal 3 - Manajemen Antrian di Terminal Kereta Cepat

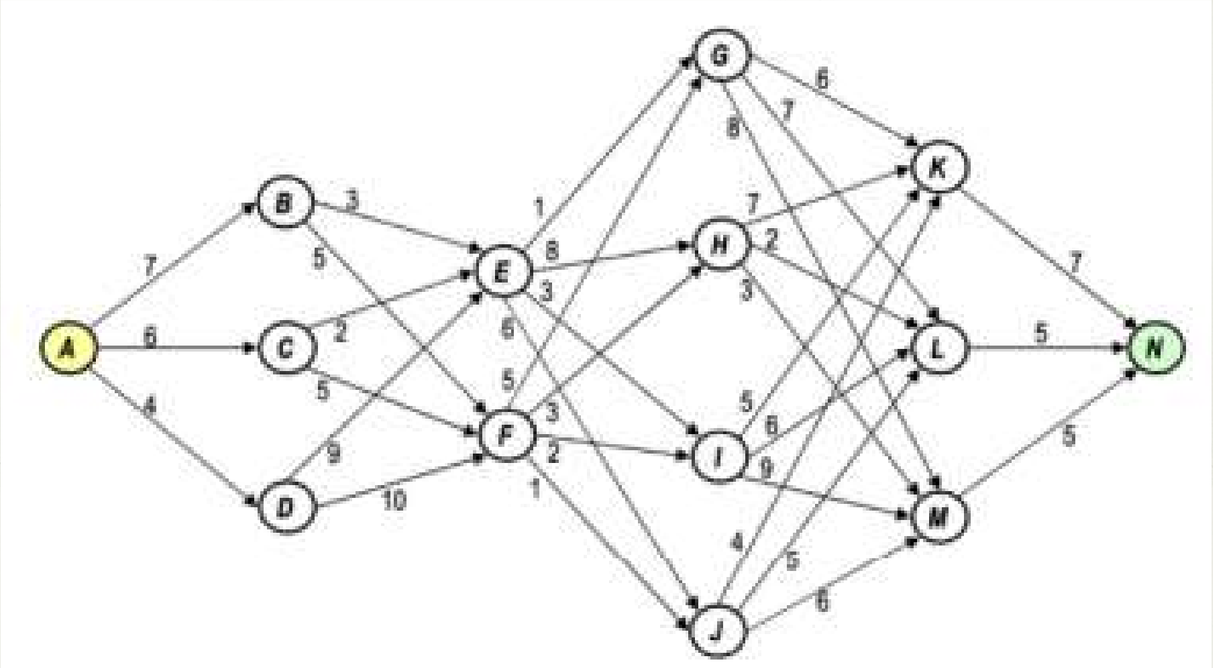
Case simulasi 1000 penumpang untuk evaluasi target performa ≥ 1 000 penumpang adalah syarat scalability

```
# ================================
# Main Entry
# ================================
if __name__ == "__main__":
    print("================================")
    print("📅 Tanggal : 2025-07-21")
    print("🚄 Kereta  : KC-Shinobi (60 kursi / 15 m)")
    print("================================")

    generate_passenger_data_sequential()

import time

start = time.time()

list_passengers = load_passenger_csv("generated_passengers_1000.csv")

for event in simulate_day(list_passengers):
    print(event)

end = time.time()

print(f"⏱️ Total Runtime: {end - start:.2f} detik\n")


print(f"Total penumpang     : {len(list_passengers)}")
```

⏱️ Total Runtime: 0.06 detik

Total penumpang : 1000

Soal 4, Graf

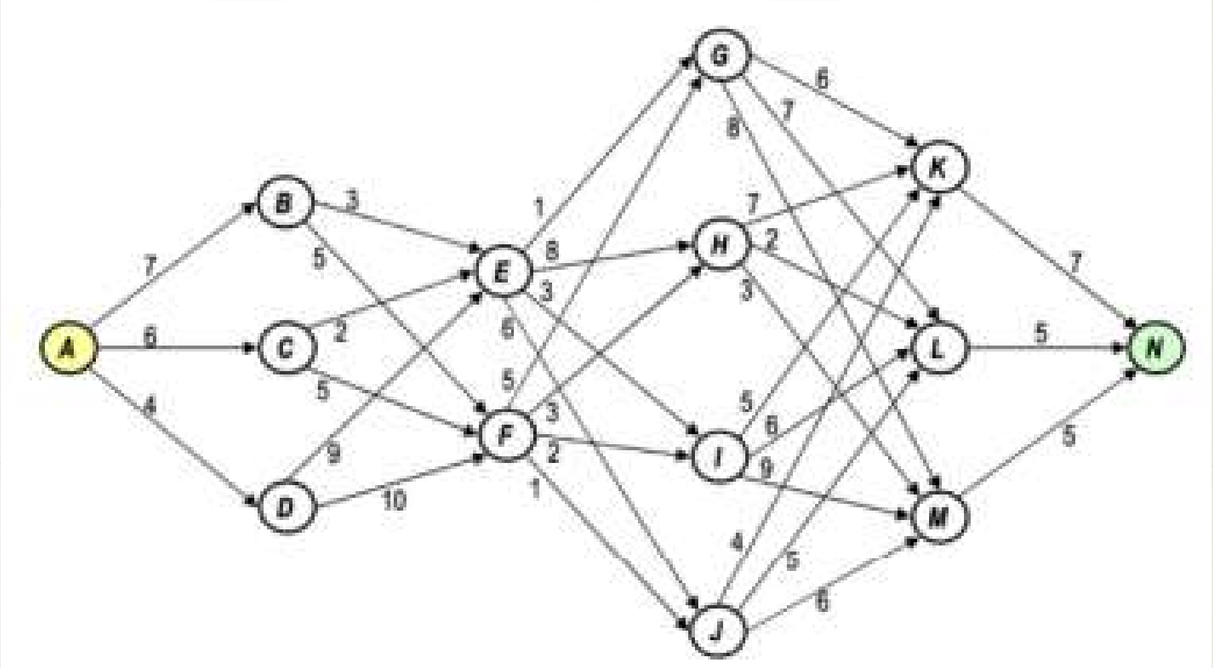


Inisiasi

- B (Visited): []
- R (Queue):[]
- U (Unvisited):[]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Pred	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Soal 4, Graf

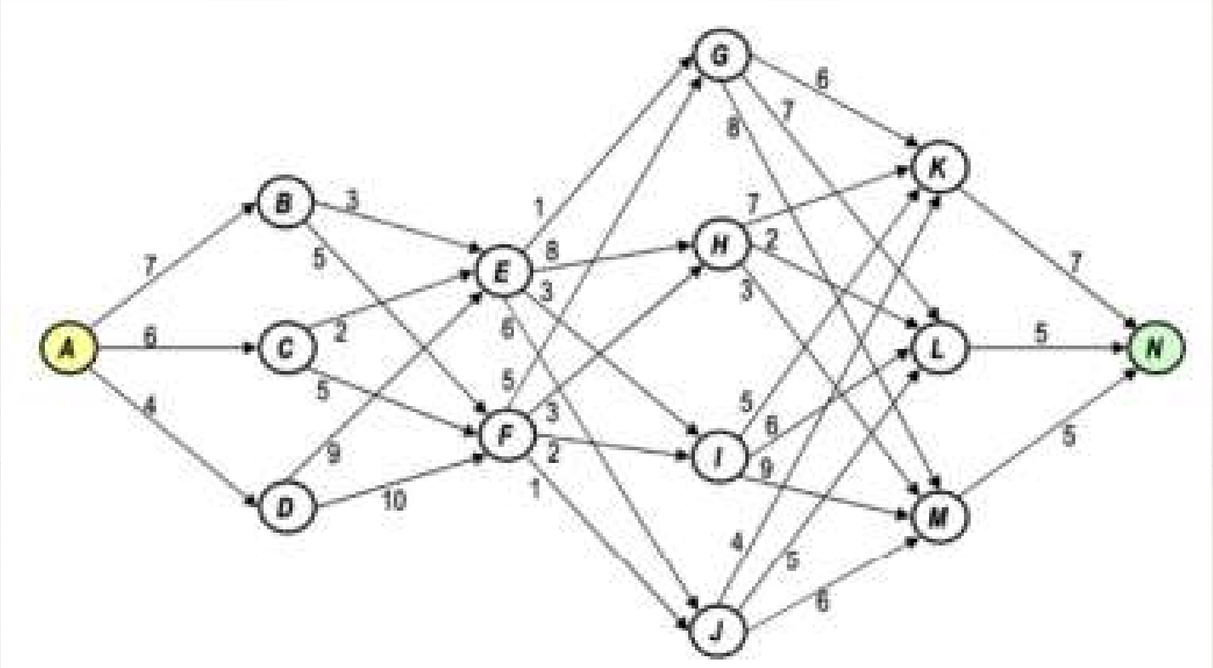


iterasi 0

- B (Visited): ['A']
- R (Queue):['D', 'B', 'C']
- U (Unvisited):['B', 'C', ‘D’, 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	∞	∞	∞	∞	∞	∞	∞	∞	∞	∞
Pred	-1	A	A	A	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Soal 4, Graf

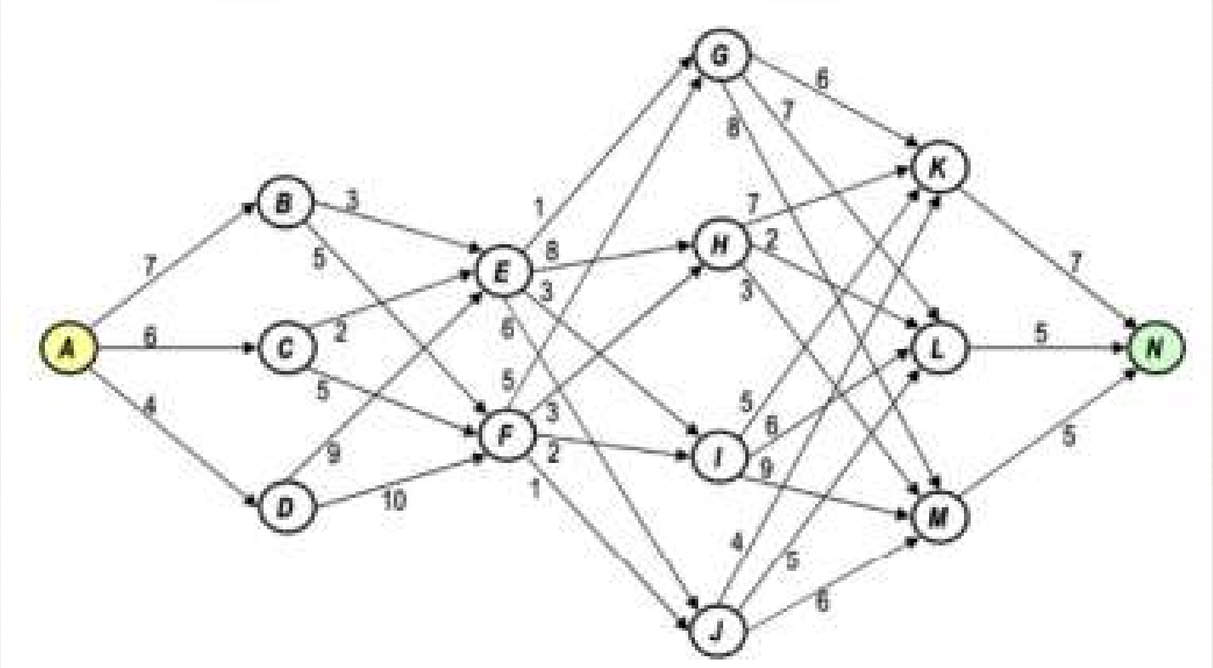


iterasi 1

- B (Visited): ['A', 'D']
- R (Queue):['C', 'B', 'E', 'F']
- U (Unvisited):['B', 'C', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	13	14	∞	∞	∞	∞	∞	∞	∞	∞
Pred	-1	A	A	A	D	D	-1	-1	-1	-1	-1	-1	-1	-1

Soal 4, Graf

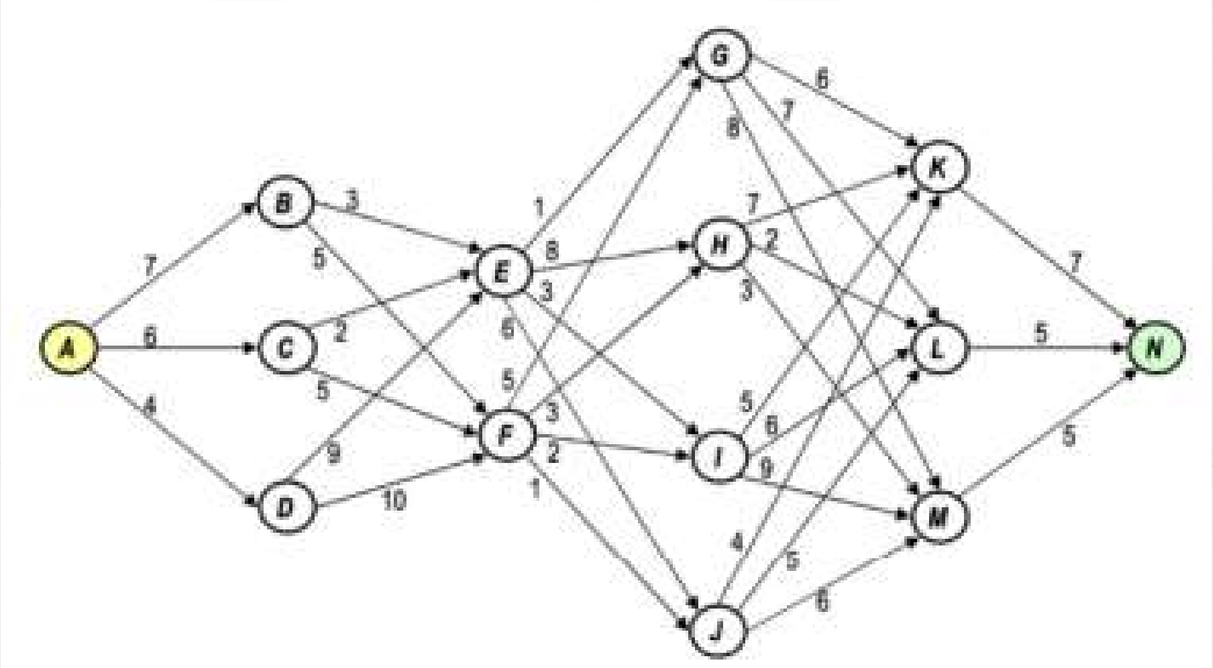


iterasi 2

- B (Visited): ['A', 'D', 'C']
- R (Queue):['B', 'E', 'E', 'F', 'F']
- U (Unvisited):['B', 'E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	∞	∞	∞	∞	∞	∞	∞	∞
Pred	-1	A	A	A	C	C	-1	-1	-1	-1	-1	-1	-1	-1

Soal 4, Graf

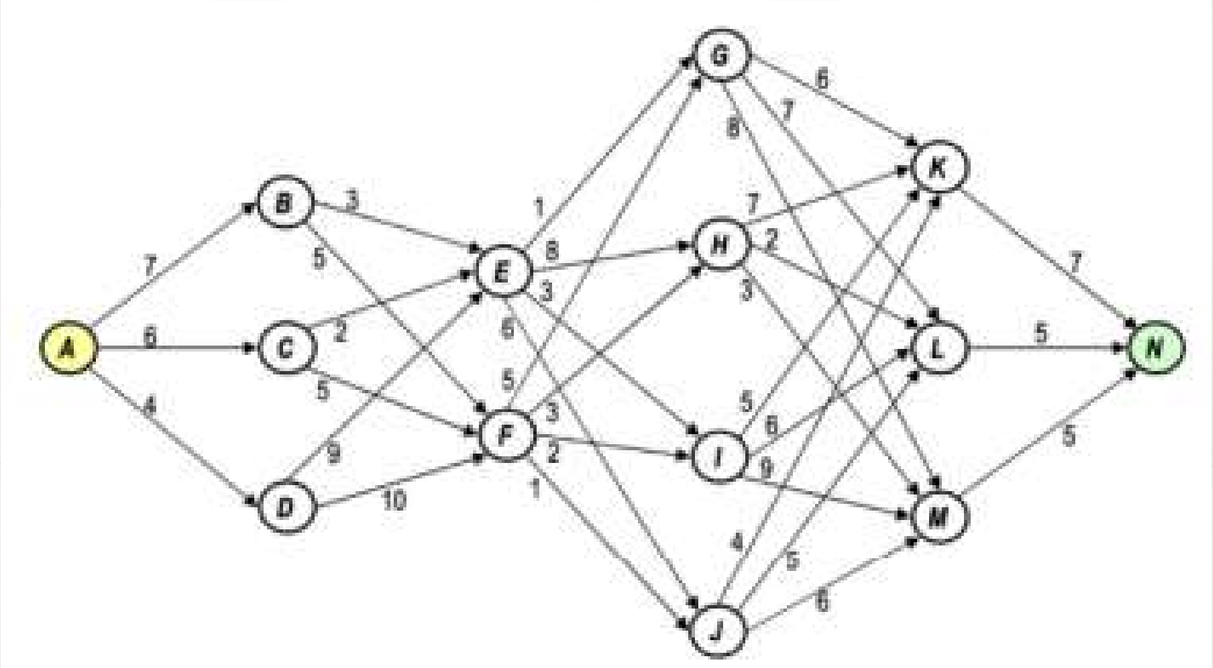


iterasi 3

- B (Visited): ['A', 'D', 'C', 'B']
- R (Queue):['E', 'F', 'E', 'F']
- U (Unvisited):['E', 'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	∞	∞	∞	∞	∞	∞	∞	∞
Pred	-1	A	A	A	C	C	-1	-1	-1	-1	-1	-1	-1	-1

Soal 4, Graf

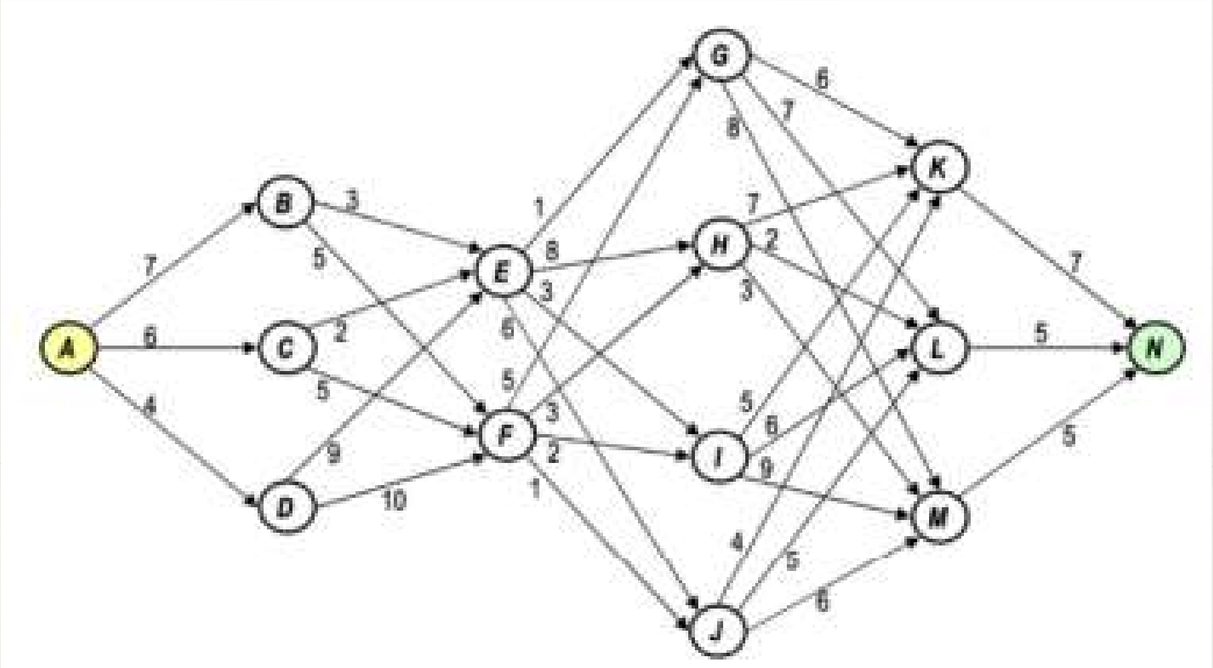


iterasi 4

- B (Visited): ['A', 'D', 'C', 'B', 'E']
- R (Queue):['G', 'F', 'I', 'F', 'H', 'J']
- U (Unvisited):['F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	16	11	14	∞	∞	∞	∞
Pred	-1	A	A	A	C	C	E	E	E	E	-1	-1	-1	-1

Soal 4, Graf

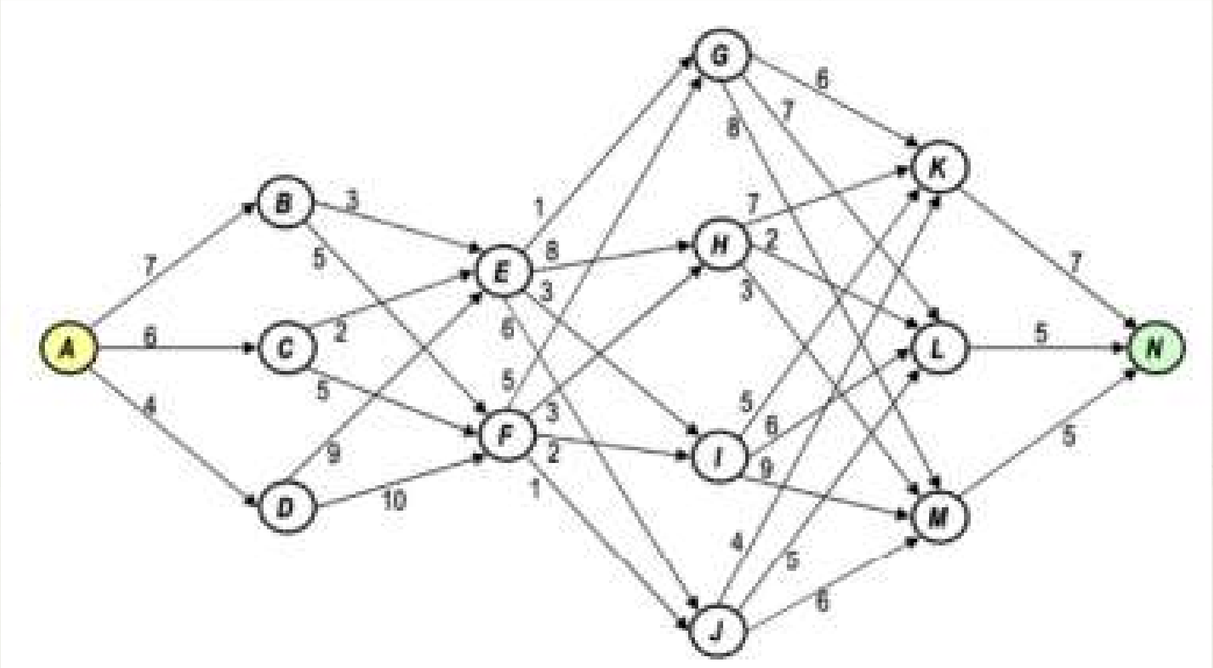


iterasi 5

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G']
- R (Queue):['F', 'F', 'I', 'J', 'H', 'K', 'L', 'M']
- U (Unvisited):['F', 'H', 'I', 'J', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	16	11	14	15	16	17	∞
Pred	-1	A	A	A	C	C	E	E	E	E	G	G	G	-1

Soal 4, Graf

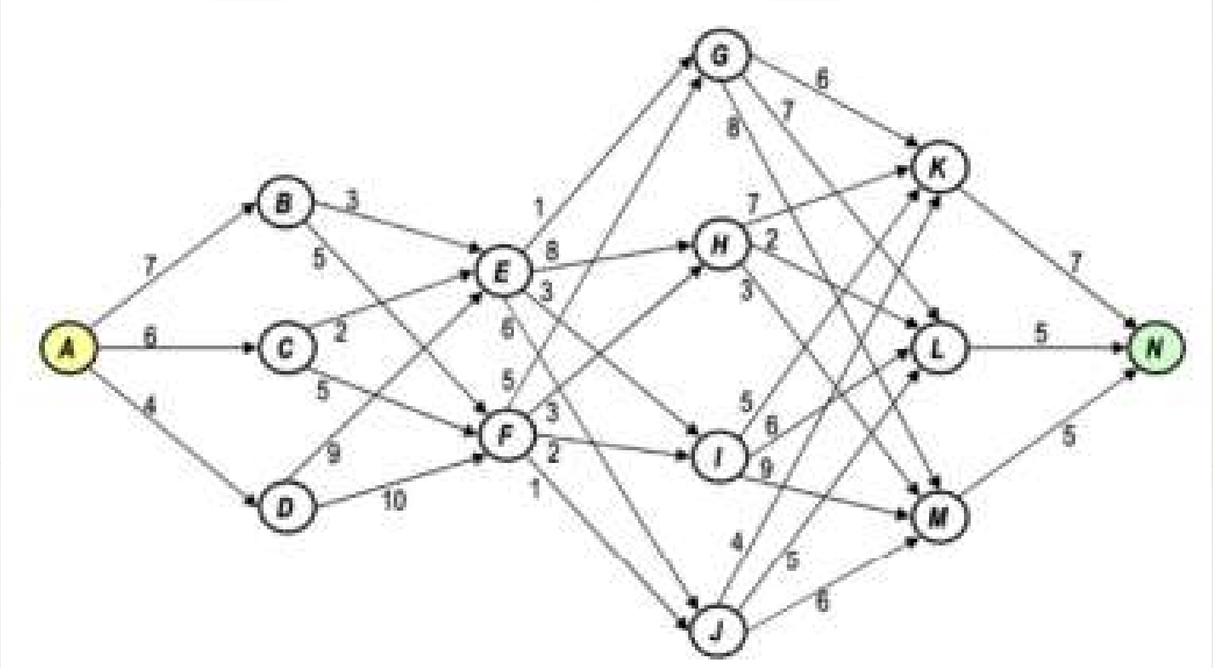


iterasi 6

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G', 'F']
- R (Queue):['I', 'J', 'H', 'M', 'K', 'L', 'J', 'H']
- U (Unvisited):['H', 'I', 'J', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	∞
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	-1

Soal 4, Graf

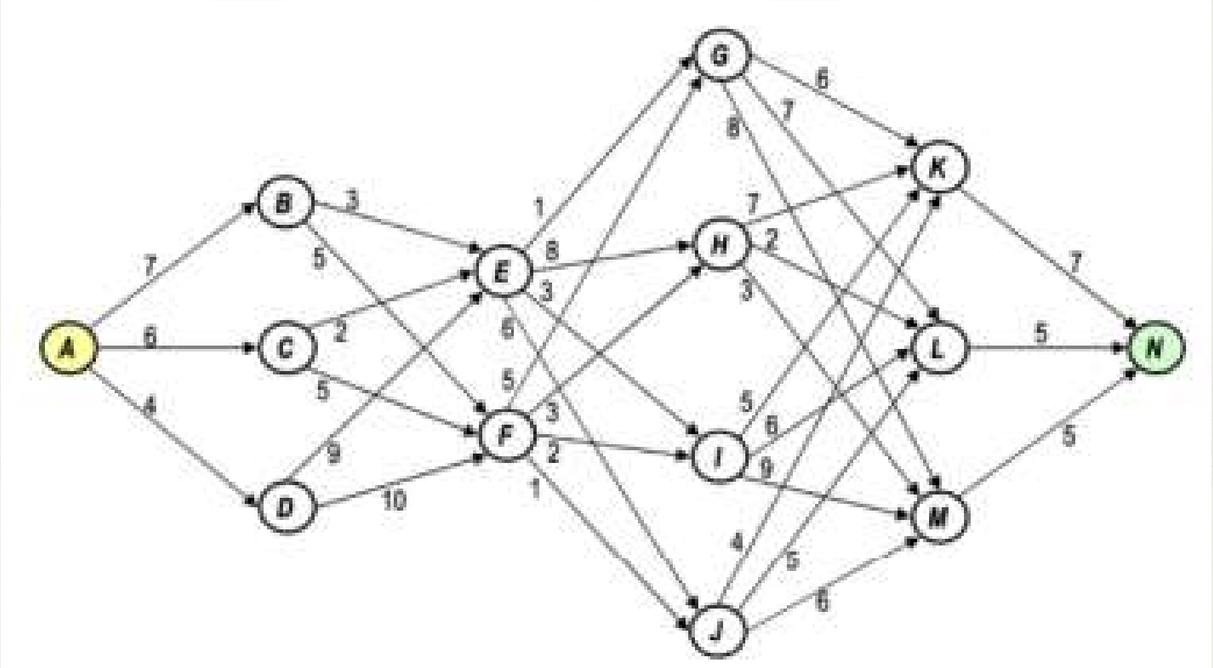


iterasi 7

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G', ‘F’, I’]
- R (Queue):['J', 'H', 'H', 'M', 'K', 'L', 'J']
- U (Unvisited):['H', 'J', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	∞
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	-1

Soal 4, Graf

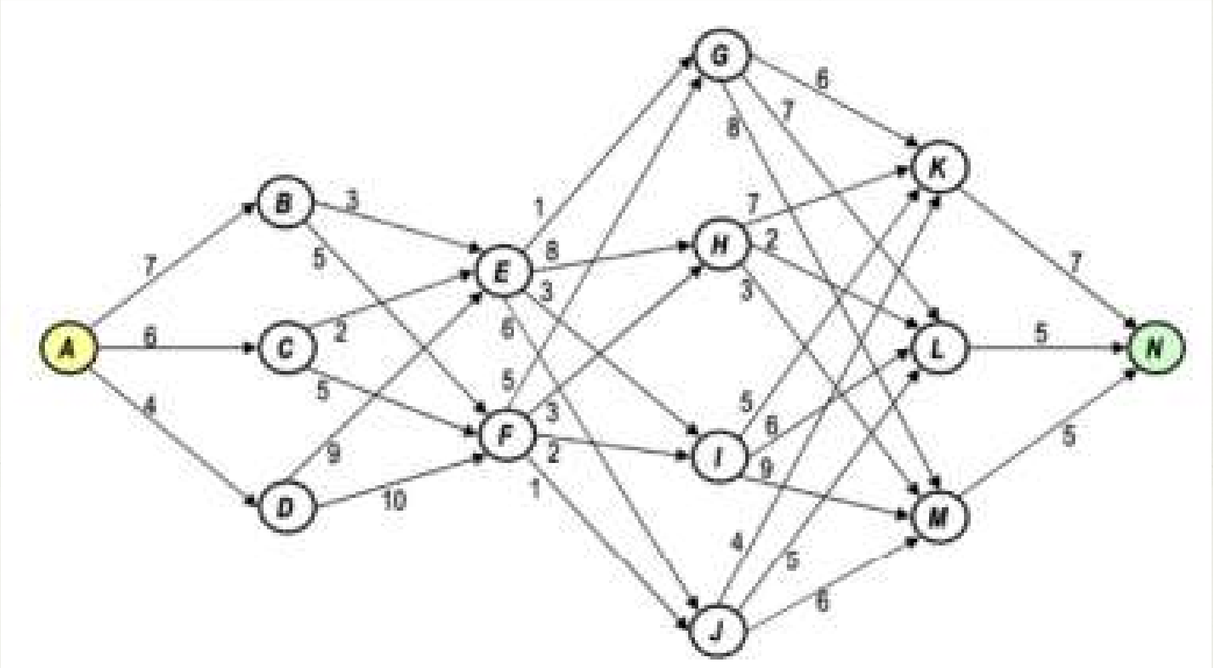


iterasi 8

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G', ‘F’, I’, ‘J’]
- R (Queue):['H', 'H', 'M', 'K', 'L']
- U (Unvisited):['H', 'K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	∞
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	-1

Soal 4, Graf

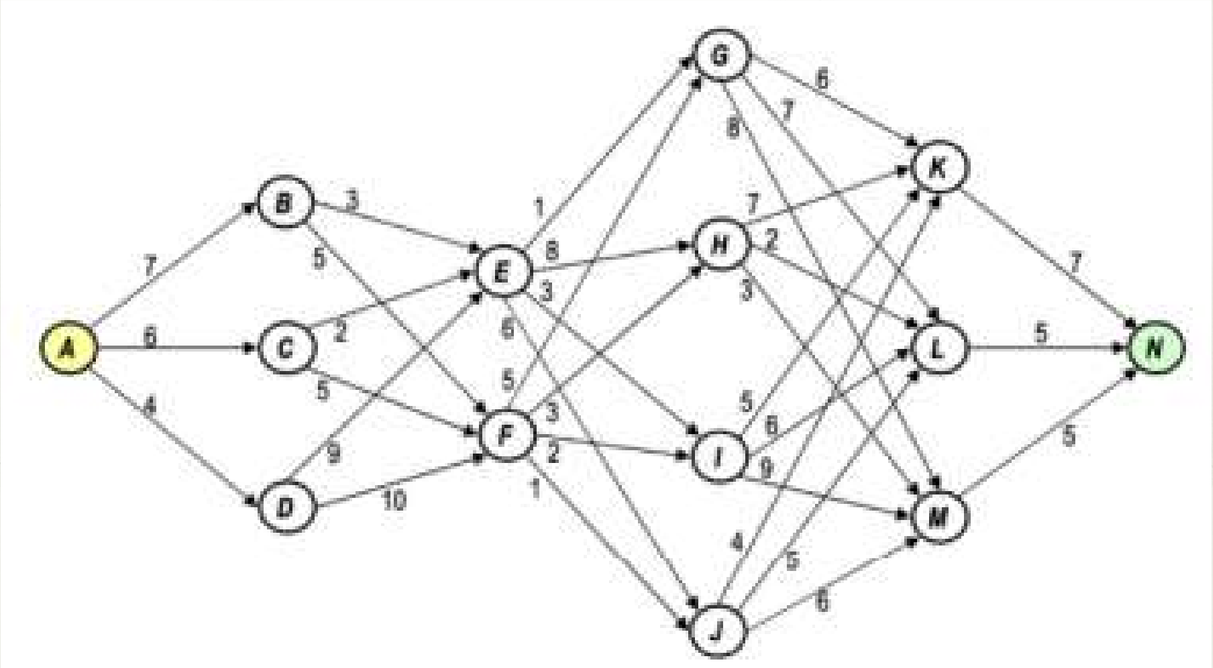


iterasi 9

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G', ‘F’, I’, ‘J’, ‘H’]
- R (Queue):['K', 'M', 'L']
- U (Unvisited):['K', 'L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	∞
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	-1

Soal 4, Graf

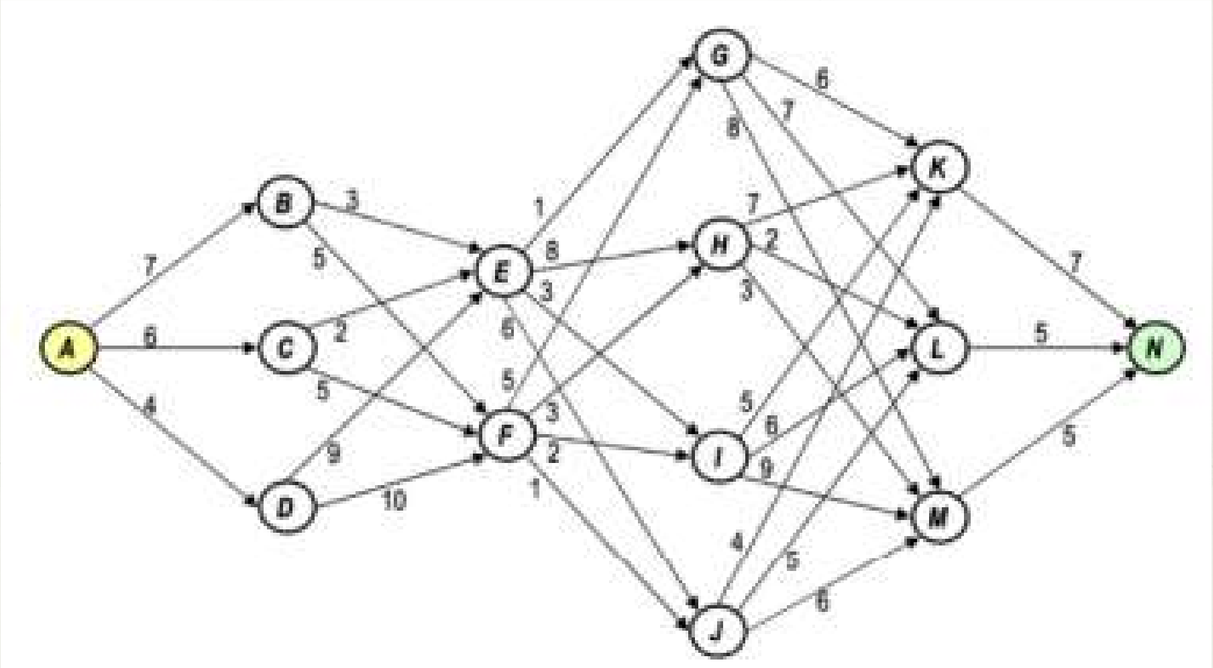


iterasi 10

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G', ‘F’, I’, ‘J’, ‘H’, K’]
- R (Queue):['L', 'M', 'N']
- U (Unvisited):['L', 'M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	22
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	K

Soal 4, Graf

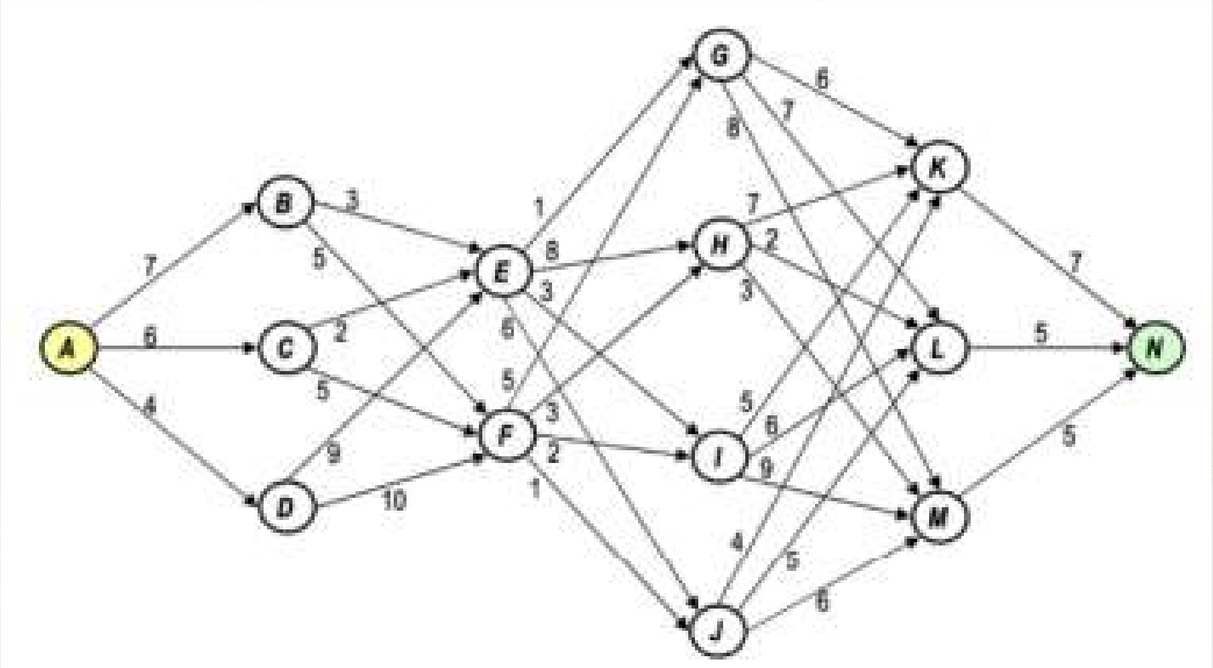


iterasi 11

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G', ‘F’, I’, ‘J’, ‘H’, K’, ‘L’]
- R (Queue):['M', 'N', ‘N’]
- U (Unvisited):['M', 'N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	21
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	L

Soal 4, Graf

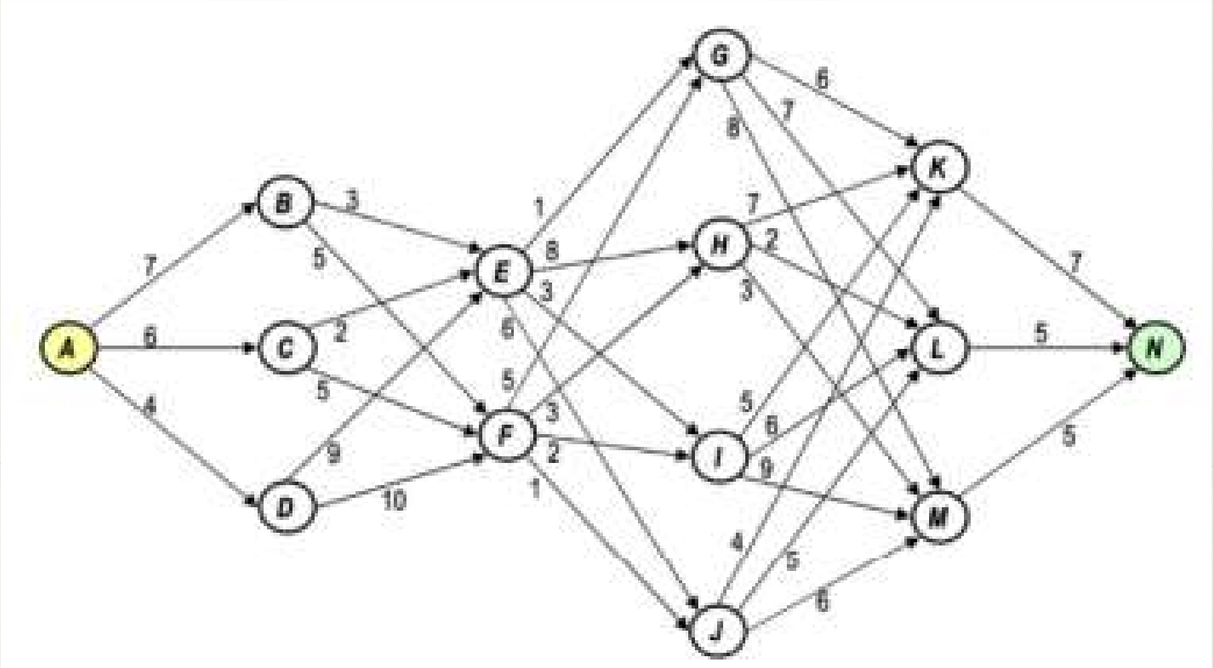


iterasi 12

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G', ‘F’, I’, ‘J’, ‘H’, K’, ‘L’, ‘M’]
- R (Queue):['N', ‘N’]
- U (Unvisited):['N']

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	21
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	L

Soal 4, Graf

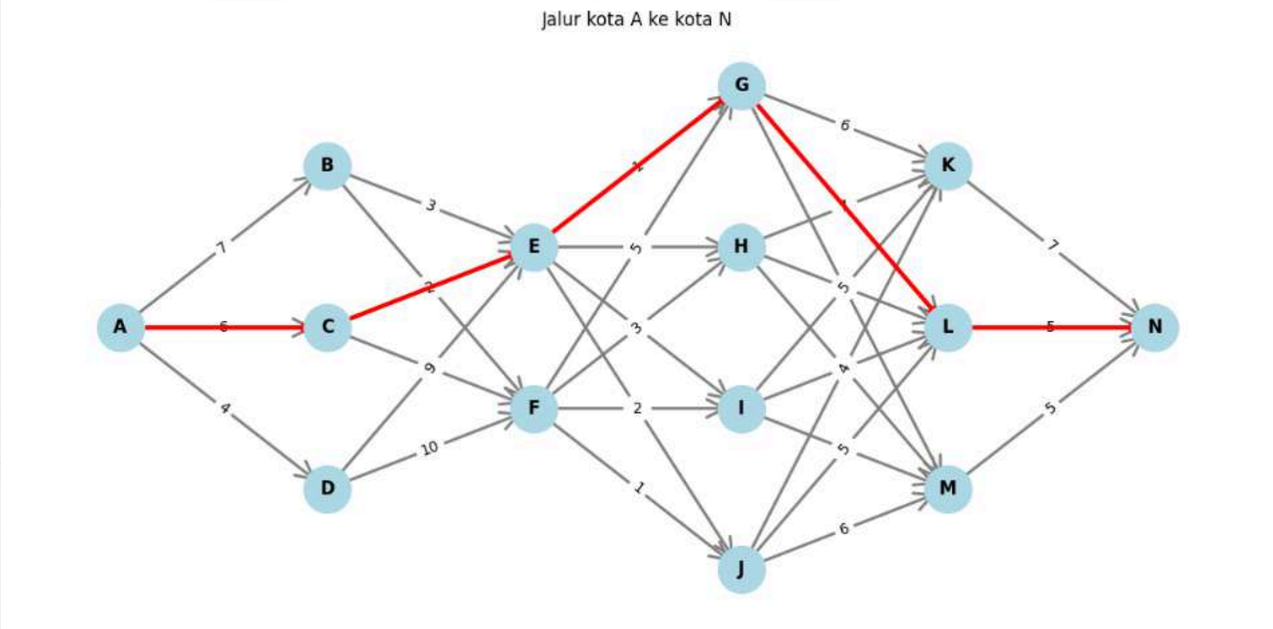


iterasi 13

- B (Visited): ['A', 'D', 'C', 'B', 'E', 'G', ‘F’, I’, ‘J’, ‘H’, K’, ‘L’, ‘M’, ‘N’]
- R (Queue):[]
- U (Unvisited):[]

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	21
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	L

Soal 4, Graf



Jarak terpendek

Kita lihat dari tabel
A← C← E← G← L← N
yang artinya jalur terpendek dari A ke N Adalah
A → C → E → G → L → N

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
Dist	0	7	6	4	8	11	9	14	11	12	15	16	17	21
Pred	-1	A	A	A	C	C	E	F	E	F	G	G	G	L

TERIMAKASIH

Semoga Bermanfaat