1 Nomor 1 Pyhton

Jawaban singkat: akan dilakukan traversal BFS yang dimulai pada simpul start dan selama proses traversal akan dilakukan pencatatan mengenai jarak tiap simpul (yang *reachable* dari start position) relatif terhadap posisi start.

jawaban lengkap: sebelum melakukan BFS, kita akan mengubah edge list (variable routes) menjadi adjecancy list.

Algoritma BFS dapat dijelaskan sebagai berikut:

1. buat sebuah queue q yang nantinya akan memuat simpul yang akan dipreses.
2. buat sebuah array used[] yang nantinya akan menandakan simpul mana saja yang sudah dikunjungi
3. mula-mula, masukan simpul $start$ pada queue q dan set used[$start$]=$true$, dan untuk sembarang simpul v lainnya set used[v]=$false$
4. kemudian, lakukan looping sampai queue q kosong. Pada setiap iterasi, pop simpul terdepan pada q, lakukan kunjungan pada simpul tetangga yang belum dikunjungi (used[v]=$false$) dan memasukan simpul tersebut pada q (used[v] menjadi $true$)

Sebagai hasilnya, ketika q kosong, kita sudah mengunjungi setiap simpul yang *reachable* dari simpul *start*, dengan setiap simpul yang dikunjungi dilakukan dengan jarak terpendek. kita hanya perlu sebuah array dist[] untuk menyimpan jarak-jarak tersebut selama proses BFS berlangsung. Jawaban yang diminta tepat dist[end].

berikut ini adalah impelentasi dengan python:

```
    def solve(v_list,e_list,start,end):
    """ mencari jarak terpendek dari node start ke node end
        jika diberikan vertices_list (v_list) dan edge_list (
           ↪ e_list).

        jika tidak ada lintasan dari start ke end, fungsi akan
           ↪ mengembalikan -1 sebagai hasilnya
    """

    #membuat adjecancy_list
    adj=defaultdict(list)
    for u,v in e_list:
        adj[u]+=[v]

    visited=[] # list node yang sudah dikunjungi
    queue=[(0,start)] #queue (dengan element(jarak, node)) untuk
       ↪  proses bfs

    #proses bfs dilakukan
    while queue:
        dist, node= queue.pop(0) #pop first element
        if node==end: #jika sudah sampai posisi end return
           ↪ jaraknya
            return dist
        if node in visited: #jika sudah dikunjungi lanjutkan ke
           ↪ proses ke node lain
            continue

        visited.append(node) # tandai nodesudah dikunjungi
        for nxt_node in adj[node]: #kunjungi setiap node
           ↪ tetangga
           queue.append((dist+1,nxt_node))
    return -1
```