

1 Nomor 1 Python

Penjelasan 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*.

Penjelasan 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 diproses.
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 implementasi 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].append(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

```

2 Nomor 2 Python

Penjelasan singkat(ide): kita cari sebuah fungsi (mapping) yang memetakan sembarang papan ke $\{0, 1, \dots, 63\}$ yang memiliki karakteristik berikut: kita bisa mengubah hasil petanya ke sembarang nilai pada kodomain ($\{0, 1, \dots, 63\}$) dengan hanya dengan mengubah satu posisi coin saja. Jika kita bisa mencari fungsi tersebut, kita bisa meng-encode lokasi kunci pada papan bagaimana-pun konfigurasinya

Penjelasan lengkap:

Misalkan A himpunan semua konfigurasi papan yang mungkin, tinjau fungsi $f : A \rightarrow \{0, \dots, 63\}$ dengan $f(x) = (b_5 b_4 b_3 b_2 b_1 b_0)_2$ dimana

1. $b_0 = 1$ jika jumlah koin bergambar pada kolom ke $[1, 3, 5, 7]$ adalah ganjil, $b_0 = 0$ jika lainnya.
2. $b_1 = 1$ jika jumlah koin bergambar pada kolom ke $[2, 3, 6, 7]$ adalah ganjil, $b_1 = 0$ jika lainnya.
3. $b_2 = 1$ jika jumlah koin bergambar pada kolom ke $[4, 5, 6, 7]$ adalah ganjil, $b_2 = 0$ jika lainnya.
4. $b_3 = 1$ jika jumlah koin bergambar pada baris ke $[1, 3, 5, 7]$ adalah ganjil, $b_3 = 0$ jika lainnya.
5. $b_4 = 1$ jika jumlah koin bergambar pada baris ke $[2, 3, 6, 7]$ adalah ganjil, $b_4 = 0$ jika lainnya.
6. $b_5 = 1$ jika jumlah koin bergambar pada baris ke $[4, 5, 6, 7]$ adalah ganjil, $b_5 = 0$ jika lainnya.

Table 1: pelabelan papan yang digunakan

r/c	0	1	2	3	4	5	6	7
0	0	1	2	3	4	5	6	7
1	8	9	10	11	12	13	14	15
2	16	17	18	19	20	21	22	23
3	24	25	26	27	28	29	30	31
4	32	33	34	35	36	37	38	39
5	40	41	42	43	44	45	46	47
6	48	49	50	51	52	53	54	55
7	56	57	58	59	60	61	62	63

mudah ditunjukkan bahwa fungsi f memetakan A ke himpunan $\{0, 1, \dots, 63\}$ dan fungsi f juga memenuhi karaktersitik yang diinginkan karena untuk sembarang $x \in A$ dan $y \in \{0, 1, \dots, 63\}$, jika x^* adalah papan x yang koin pada lokasi $f(x) \oplus y$ dibalikkan, maka $f(x^*) = y$. Dengan fungsi f tersebut, kita dapat pastikan akan selalu bisa meng-encode lokasi kuncinya, sehingga penebak akan selalu benar menebak dengan strategi ini.

Contoh: Misalkan untuk sembarang papan x dan kita punya $f(x) = 37$ (lihat tabel 1), namun lokasi kunci berada di 20 (lihat tabel 1). Dengan mengubah posisi koin pada $37 \oplus 20 = 49$ (lihat tabel 1), kita punya papan x^* sehingga $f(x^*) = 20$.

Berikut ini adalah implementasi fungsi f , infortmant, guesser-nya:

```
def bit_counter_strat(board):

    """program untuk melakukan strategi peng-encodingan papan,
    setiap sembarang susunan papan akan dimapping ke {0,1, ...,
    ↪ 63}
    """
    encode_for="" #hasil dari f(x) dalam binary
    for i in total_region: #column activity: mencari b0 , b1,
    ↪ b2
        sigma=0
        for j in i:
            for k in range (8):
                sigma+=board[k][j]
        if sigma%2==0:
            encode_for='0'+encode_for
        else:
            encode_for='1'+encode_for
```

```

for i in total_region: #row activity: mencari b3, b4,b5,
    sigma=0
    for j in i:
        for k in range (8):
            sigma+=board[j][k]
    if sigma %2==0:
        encode_for='0'+encode_for
    else:
        encode_for='1'+encode_for

return int(encode_for,2) #return dalam decimal

```

```

def informant(board, key):
    """melakukan 1 flip coin pada papan (dengan strategi)
    ↳ sehingga menghasilkan papan baru
    yang telah mengandung informasi mengenai lokasi kunci
    """
    num_encode=bit_counter_strat(board) #mencari f(x), x
    ↳ konfigurasi papan acak
    num_key= public_board[key[0]][key[1]] # mencari y dalam
    ↳ {0,..., 63}, ketika diberikan indeksny
    xor_num= num_encode ^ num_key # lokasi koin yang akan
    ↳ dibalikan
    row, column = get_index(public_board, xor_num) #indeks koin
    ↳ yang akan dibalikan
    board[row][column]= 1-board[row][column] #membalikkan koin
    ↳ tersebut
    return board #mereturn board (x*)

```

```

def guess(board):
    """ melakukan decode mengenai lokasi informasi kunci pada
    ↳ papan yang sudah dilakukan flip
    coin
    """
    num_encode=bit_counter_strat(board) #f(x*) [mendeocode
    ↳ lokasi kunci pada papan dari informant]
    print(get_index(public_board, num_encode)) #mereturn
    ↳ indeksny

```

3 Nomor 1 Mathematica

Misalkan A adalah kejadian minimal dua dari n orang memiliki tanggal lahir yang sama. Misalkan A' adalah kejadian komplemen dari A , semua orang tidak memiliki tanggal lahir yang sama. Dengan begitu, kita punya

$$P(A') = \frac{365}{365} \times \frac{364}{365} \times \cdots \times \frac{365 - n + 1}{365} = \frac{n! \binom{365}{n}}{365^n}$$

dan

$$P(A) = 1 - P(A')$$

4 Nomor 2 Mathematica

Kita akan melakukan traversal pada grid dengan pergerakan spiral, lalu mengisi cell tersebut dengan angka dari 1 sampai n^2 .

Tinjau bahwa, secara berurutan pergerakannya hanyalah $\{\downarrow, \rightarrow, \uparrow, \leftarrow \dots\}$. Kita hanya perlu kondisi kapan kita akan berganti gerakan (contoh: dari gerakan kebawah menjadi gerakan kekanan) Tinjau beberapa kasus general berikut:

Table 2: Kasus 1: mengganti gerakan ketika akan keluar dari grid-nya

1	-	-	-
2	-	-	-
3	-	-	-
X	-	-	-

1. Tinjau tabel 2. Pada kasus ini, kita sudah traverse kebawah hingga posisi X. untuk mengganti ke gerakan selanjutnya, kita gunakan conditional ketika gerakan awal (sebelum berganti) mengakibatkan posisi keluar dari grid. Dengan begitu, untuk contoh kasus di atas kita punya kondosi berikut:

```
IF posisi_baris+current_movement_baris>n
    current_movement_baris= next_movement_baris
    current_movement_kolom= next_movement_kolom
END IF
```

Untuk ketiga kasus out of grid lainnya similar.

Table 3: mengganti gerakan ketika posisi selanjutnya sudah diisi

1	-	-	-
2	-	-	-
3	10	9	8
4	5	6	7

2. Tinjau tabel 3. pada kasus ini kita berada di posisi 10 dan sedang bergerak ke kiri. Tinjau bahwa jika kita bergerak ke kanan, sel selanjutnya sudahlah diisi. oleh karena itu, kita bisa mengganti gerakannya dengan kondisikondisi sel selanjutnya sudah diisi. dalam pseudocode:

```

IF grid [ posisi_baris+current_movement+baris ] [ posisi_kolom+
    ↪ current_movement_kolom ] not empty
current_movement_baris= next_movement_baris
current_movement_kolom= next_movement_kolom
END IF

```

Selanjutnya, Nilai terbesar selalulah n^2 dan posisinya akan ditengah jika n ganjil. lalu sumasi dari elemen pada grid adalah $1 + 2 + \dots + n^2 = \frac{n^2(n^2+1)}{2}$