1. **Wibu Nolep.** Akhir - akhir ini marak terjadi penyerangan terhadap warga yang dilakukan oleh wibu nolep. Untuk mengatasi hal itu pihak keamanan melakukan penelusuran terhadap setiap orang untuk menentukkan apakah dia adalah wibu nolep atau bukan dengan cara menghitung jumlah tetangganya. Terdapat sebuah survei yang mengatakan bahwa apabila seseorang tidak memiliki tetangga lebih dari n tetangga maka dia adalah wibu nolep karena kecenderungan wibu nolep untuk menyendiri. Bantu pihak keamanan untuk menentukkan apakah seseorang adalah wibu nolep atau bukan.

Sample Input

5	2	5 adalah jumlah orang dan 2 adalah jumlah n
0	1	0 -> 1
0	4	0 -> 4
0	3	0 -> 3
1	2	1 -> 2
1	3	1 -> 3
2	3	2 -> 3
-1	-1	inisialisasi graph berhenti ketika vertex source dan destination bernilai -1

Sample Output

Vertex 4 adalah wibu nolep

1.1. Sumber Kode

```c
#include <stdio.h>
#include <stdlib.h>

// Node yang merepresentasikan edge di adjacency list
typedef struct Node {
    int dest;
    struct Node* next;
} Node;

// Structure graph
typedef struct Graph {
    int numVertices;
    Node** adjList;
} Graph;

// Membuat node
Node* createNode(int dest){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

// Membuat Graph
Graph* createGraph(int numVertices){
    Graph* newGraph = (Graph*)malloc(sizeof(Graph));
    newGraph->numVertices = numVertices;

    // Membuat array dari adjacency list
    newGraph->adjList = (Node**)malloc(numVertices * sizeof(Node*));

    // Isi nilai setiap adjacency list dengan NULL
    for(int i = 0; i < numVertices; i++){
        newGraph->adjList[i] = NULL;
    }

    return newGraph;
}

// Menambahkan edge pada undericted graph (menghubungkan vertex secara
undericted)
void addEdge(Graph* graph, int src, int dest){
    // Tambahkan edge dari vertex source ke vertex destination
    Node* newNode = createNode(dest);
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;

    // Tambahkan edge dari vertex destination ke vertex source
    newNode = createNode(src);
    newNode->next = graph->adjList[dest];
    graph->adjList[dest] = newNode;
}
```

```c
// Menampilkan graph
void printWibu(Graph* graph, int minNeighbor) {
    // Menghitung jumlah tetangga dari tiap orang(vertex)
    int len = graph->numVertices;
    int neighborCount[len], nolep = -1;
    for (int i = 0; i < len; ++i) {
        Node* temp = graph->adjList[i];
        neighborCount[i] = 0;
        while (temp) {
            neighborCount[i]++;
            temp = temp->next;
        }
        // Menetapkan orang dengan tetangga paling sedikit (dibawah minimum)
        if(neighborCount[i] < minNeighbor){
            nolep = i;
        }
    }
    // Menyatakan vertex dari si wibu nolep
    if(nolep > -1){
        printf("Vertex %d adalah wibu nolep.", nolep);
    }else{
        printf("Tidak ditemukan wibu nolep dalam komplek perumahan(graph).");
    }

}

// Membebaskan memori yang dipakai graph
void destroyGraph(Graph* graph){
    if(graph){
        if(graph->adjList){
            for(int i = 0; i < graph->numVertices; ++i){
                Node* temp = graph->adjList[i];
                while (temp){
                    Node* prev = temp;
                    temp = temp->next;
                    free(prev);
                }
            }
            free(graph->adjList);
        }
        free(graph);
    }
}

// Fungsi utama
void main(){
    int vertices, minNeighbor, src, dest;
    // Menerima inputan jumlah orang(vertex)
    printf("Jumlah Orang(Vertex): ");
    scanf("%d", &vertices);
    // Membuat graph yang menampung vertices sejumlah yang diinputkan di atas
    Graph* graph = createGraph(vertices);
    // Menerima inputan jumlah minimum tetangga
    printf("Jumlah Minimum Tetangga: ");
    scanf("%d", &minNeighbor);
    // Menambahkan edge(hubungan tetangga)
    do{
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }while(src > -1 && dest > -1);

    // Menampilkan orang(vertex) dengan tetangga paling sedikit(di bawah minimum)
    printWibu(graph, minNeighbor);

    // Membebaskan memory dari graph
    destroyGraph(graph);
}
```

1.2. Hasil

```
PS D:\KULIAH\ITTS Semester 2\Tugas\Algoritma dan Struktur Data\Pertemuan 24 - 11 Juli 2023> .\tugas9_soal1
Jumlah Orang(Vertex): 5
Jumlah Minimum Tetangga: 2
0 1
0 4
0 3
1 2
1 3
2 3
-1 -1
Vertex 4 adalah wibu nolep.
PS D:\KULIAH\ITTS Semester 2\Tugas\Algoritma dan Struktur Data\Pertemuan 24 - 11 Juli 2023>
```

2. **Roronoa Zoro dan Kota Surabaya.** Di kota Surabaya, terdapat banyak sekali jalan raya sehingga konsekuensinya adalah terdapat banyak persimpangan di kota ini (mulai dari simpang dua, simpang tiga (pertigaan), dan seterusnya (perempatan, perlimaan, dst)). Roronoa Zoro yang banyak menghabiskan waktunya di laut sejak kecil, tiba-tiba datang ke Surabaya untuk berpiknik. Sebagai informasi, Roronoa Zoro memiliki kelemahan aneh yang berasal dari buah iblis (devil fruit). Kelemahan aneh tersebut membuat Zoro hanya bisa memahami maksimal simpang X ketika ia berumur X tahun. Sebagai contoh, ketika Zoro berusia 4 tahun hanya bisa memahami maksimal simpang 4 (perempatan) dan akan pingsan ketika menemui simpang 5 dan seterusnya. Apabila terdapat sebanyak N persimpangan di kota Surabaya, buatlah program untuk memberi tahu persimpangan mana saja yang harus dihindari oleh Roronoa Zoro agar ia tidak pingsan apabila saat ini Roronoa Zoro berusia M tahun.

   Note: bedakan simpang 4 dan persimpangan 4. Simpang 4 menandakan perempatan (artinya sebuah vertex memiliki 4 tetangga). Sedangkan, persimpangan 4 merupakan urutan persimpangan yang ada di kota Surabaya. Sebagai clue, dua persimpangan dihubungkan oleh 1 jalan raya yang merupakan representasi dari edge

   Sample Input

```
5       // banyak perempatan di Kota Surabaya (N)
2       // usia Roronoa Zoro saat ini (M)
0 1     // menghubungkan persimpangan 0 dengan persimpangan 1  (dipisah spasi)
0 4     // menghubungkan persimpangan 0 dengan persimpangan 4 (dipisah spasi)
1 2     // ... dan seterusnya
1 3     // ... dan seterusnya
1 4     // ... dan seterusnya
2 3     // ... dan seterusnya
3 4     // ... dan seterusnya
-1 -1   // inisialisasi graph berhenti ketika vertex source dan destination bernilai -1
```

   Sample Output

```
Persimpangan yang harus dihindari oleh Roronoa Zoro adalah:
Persimpangan 1
Persimpangan 3
Persimpangan 4
```

2.1. Sumber Kode

```c
#include <stdio.h>
#include <stdlib.h>

// Node yang merepresentasikan edge di adjacency list
typedef struct Node {
    int dest;
    struct Node* next;
} Node;

// Structure graph
typedef struct Graph {
    int numVertices;
    Node** adjList;
} Graph;

// Membuat node
Node* createNode(int dest){
    Node* newNode = (Node*)malloc(sizeof(Node));
    newNode->dest = dest;
    newNode->next = NULL;
    return newNode;
}

// Membuat Graph
Graph* createGraph(int numVertices){
    Graph* newGraph = (Graph*)malloc(sizeof(Graph));
    newGraph->numVertices = numVertices;

    // Membuat array dari adjacency list
    newGraph->adjList = (Node**)malloc(numVertices * sizeof(Node*));

    // Isi nilai setiap adjacency list dengan NULL
    for(int i = 0; i < numVertices; i++){
        newGraph->adjList[i] = NULL;
    }

    return newGraph;
}
```

```c
// Menambahkan edge pada undericted graph (menghubungkan vertex secara
undericted)
void addEdge(Graph* graph, int src, int dest){
    // Tambahkan edge dari vertex source ke vertex destination
    Node* newNode = createNode(dest);
    newNode->next = graph->adjList[src];
    graph->adjList[src] = newNode;

    // Tambahkan edge dari vertex destination ke vertex source
    newNode = createNode(src);
    newNode->next = graph->adjList[dest];
    graph->adjList[dest] = newNode;
}

// Menampilkan graph
void printHimbauan(Graph* graph, int maxSimpang) {
    printf("Persimpangan yang harus dihindari oleh Roronoa Zoro adalah:\n");
    // Menghitung jumlah simpangan/cabang/jalan dari setiap persimpangan(vertex)
    int len = graph->numVertices;
    int intersectionBranchCount[len];
    for (int i = 0; i < len; ++i) {
        Node* temp = graph->adjList[i];
        intersectionBranchCount[i] = 0;
        while (temp) {
            intersectionBranchCount[i]++;
            temp = temp->next;
        }
        // Memeriksa dan menampilkan persimpangan yang tidak boleh dilewati Zoro
        if(intersectionBranchCount[i] > maxSimpang){
            printf("Persimpangan %d\n", i);
        }
    }
}
```

```c
// Membebaskan memori yang dipakai graph
void destroyGraph(Graph* graph){
    if(graph){
        if(graph->adjList){
            for(int i = 0; i < graph->numVertices; ++i){
                Node* temp = graph->adjList[i];
                while (temp){
                    Node* prev = temp;
                    temp = temp->next;
                    free(prev);
                }
            }
            free(graph->adjList);
        }
        free(graph);
    }
}

// Fungsi utama
void main(){
    int vertices, umurZoro, src, dest;
    // Menerima inputan jumlah persimpangan(vertex)
    printf("Jumlah Persimpangan(Vertex): ");
    scanf("%d", &vertices);
    // Membuat graph yang menampung vertices sejumlah yang diinputkan di atas
    Graph* graph = createGraph(vertices);
    // Menerima inputan jumlah minimum tetangga
    printf("Umur Roronoa Zoro(Batasan): ");
    scanf("%d", &umurZoro);
    // Menambahkan edge(jalan/cabang/simpangan)
    do{
        scanf("%d %d", &src, &dest);
        addEdge(graph, src, dest);
    }while(src > -1 && dest > -1);
    // Menampilkan himbauan untuk Zoro sesuai umurnya
    printHimbauan(graph, umurZoro);
    // Membebaskan memory dari graph
    destroyGraph(graph);
}
```

2.2. Hasil

```
PS D:\KULIAH\ITTS Semester 2\Tugas\Algoritma dan Struktur Data\Pertemuan 24 - 11 Juli 2023> .\tugas9_soal2
Jumlah Persimpangan(Vertex): 5
Umur Roronoa Zoro(Batasan): 2
0 1
0 4
1 2
1 3
1 4
2 3
3 4
-1 -1
Persimpangan yang harus dihindari oleh Roronoa Zoro adalah:
Persimpangan 1
Persimpangan 3
Persimpangan 4
PS D:\KULIAH\ITTS Semester 2\Tugas\Algoritma dan Struktur Data\Pertemuan 24 - 11 Juli 2023>
```