

TUGAS JOB SHEET 7



DOSEN PENGAMPU:

RANDI PROSKA SANDRA, S.Pd, M.Sc

DISUSUN OLEH:

RIDHO HAMDANI PUTRA

23343052

PROGRAM STUDI INFORMATIKA
DEPARTEMEN TEKNIK ELEKTRONIKA
FAKULTAS TEKNIK
UNIVERSITAS NEGERI PADANG

2024

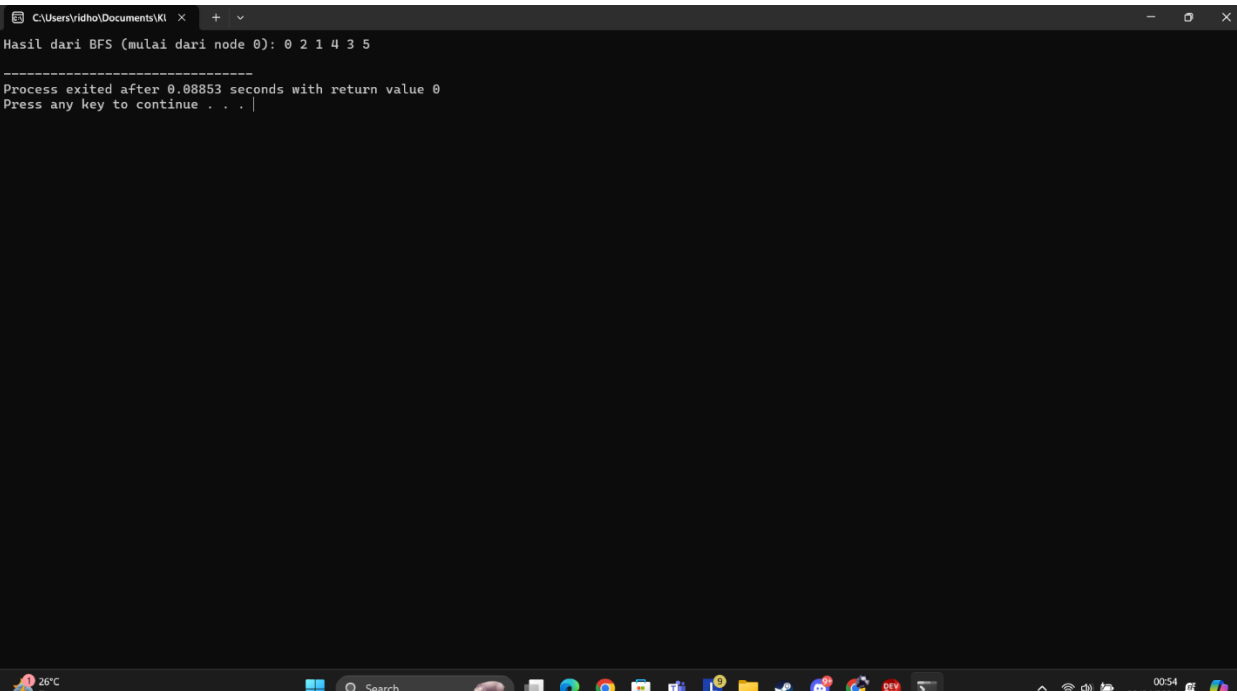
1. Source code

```
C:\Users\rkhe\Documents\KULIAH\Semester 2\Praktikum Struktur Data\Tugas Job Sheet\JobSheet 07 - Queue\Tugas\Tugas1_JS7.c - Embarcadero Dev-C++ 6.3
File Edit Search View Project Execute Tools AStyle Window Help
(globals)
Project < > Tugas1_JS7.c
1  /*
2     Nama : Ridho Hamdani Putra
3     Nim  : 23343052
4     Prodi : Informatika
5  */
6  #include <stdio.h>
7  #include <stdlib.h>
8
9  #define MAX_NODES 100
10
11 // Node dalam graph
12 typedef struct Node {
13     int data;
14     struct Node* next;
15 } Node;
16
17 // Representasi graph dengan list adjasensi
18 typedef struct Graph {
19     int numNodes;
20     Node* adjList[MAX_NODES];
21 } Graph;
22
23 // Fungsi untuk membuat node baru
24 Node* createNode(int data) {
25     Node* newNode = (Node*)malloc(sizeof(Node));
26     newNode->data = data;
27     newNode->next = NULL;
28     return newNode;
29 }
30
31 // Fungsi untuk membuat graph baru
32 Graph* createGraph(int numNodes) {
33     Graph* graph = (Graph*)malloc(sizeof(Graph));
34     graph->numNodes = numNodes;
35
36     // Inisialisasi list adjasensi
37     for (int i = 0; i < numNodes; i++) {
38         graph->adjList[i] = NULL;
39     }
40
41     return graph;
42 }
43
44 // Fungsi untuk menambahkan edge antara dua node
45 void addEdge(Graph* graph, int src, int dest) {
46     Node* newNode = createNode(dest);
47     newNode->next = graph->adjList[src];
48     graph->adjList[src] = newNode;
49
50     // Karena graph tak berarah, tambahkan edge dari dest ke src juga
51     newNode = createNode(src);
52     newNode->next = graph->adjList[dest];
53     graph->adjList[dest] = newNode;
54 }
55
56 // Fungsi untuk melakukan Breadth First Search
57 void BFS(Graph* graph, int startNode) {
58     // Array untuk menandai node yang telah dikunjungi
59     int visited[MAX_NODES] = {0};
60
61     // Queue untuk menyimpan node yang akan dikunjungi selanjutnya
62     int queue[MAX_NODES];
63     int front = 0, rear = 0;
64
65     // Mulai dari node awal
66     visited[startNode] = 1;
67     queue[rear++] = startNode;
68
69     // Selama queue tidak kosong, proses node secara BFS
70     while (front < rear) {
71         int currentNode = queue[front++];
72         printf("%d ", currentNode);
73
74         // Traverse melalui simpul-simpul bertetangga dari simpul saat ini
75         Node* temp = graph->adjList[currentNode];
76         while (temp != NULL) {
77             int adjNode = temp->data;
78             if (!visited[adjNode]) {
```

The screenshot displays a C++ IDE with a project named "Tugas1_JS7.c". The code implements a Breadth-First Search (BFS) algorithm on a graph. The graph is defined by 6 nodes and 10 edges. The BFS starts at node 0 and traverses the graph level by level, printing the nodes in the order they are visited: 0, 1, 2, 3, 4, 5.

```
1  C:\Users\rifko\Documents\KULIAH\Semester 2\Praktikum Struktur Data\Tugas Job Sheet 07 - Queue\Tugas1_JS7.c - Embarcadero Dev C++ 6.3
2  File Edit Search View Project Execute Tools AStyle Window Help
3  | (globals) |
4  Project < > Tugas1_JS7.c
5
6  67     queue[rear++] = startNode;
7  68
8  69     // Selama queue tidak kosong, proses node secara BFS
9  70     while (front < rear) {
10 71         int currentNode = queue[front++];
11 72         printf("%d ", currentNode);
12 73
13 74         // Traverse melalui simpul-simpul bertetangga dari simpul saat ini
14 75         Node* temp = graph->adjlist[currentNode];
15 76         while (temp != NULL) {
16 77             int adjNode = temp->data;
17 78             if (!visited[adjNode]) {
18 79                 visited[adjNode] = 1;
19 80                 queue[rear++] = adjNode;
20 81             }
21 82             temp = temp->next;
22 83         }
23 84     }
24 85
25 86
26 87 int main() {
27 88     // Membuat graph
28 89     Graph* graph = createGraph(6);
29 90     addEdge(graph, 0, 1);
30 91     addEdge(graph, 0, 2);
31 92     addEdge(graph, 1, 3);
32 93     addEdge(graph, 1, 4);
33 94     addEdge(graph, 2, 4);
34 95     addEdge(graph, 3, 4);
35 96     addEdge(graph, 3, 5);
36 97     addEdge(graph, 4, 5);
37 98
38 99     printf("Hasil dari BFS (mulai dari node 0): ");
39 100    BFS(graph, 0);
40 101    printf("\n");
41 102
42 103    return 0;
43 104
44 105
45  Compiler Resources Compile Log Debug Find Results Console
46  Line: 67 Col: 31 Sel: 0 Lines: 105 Length: 2644 Insert Done parsing in 0.016 seconds
47  26°C T-storms 00:59 02/04/2024
```

2. Output



C:\Users\vridho\Documents\KI > +
 Hasil dari BFS (mulai dari node 0): 0 2 1 4 3 5

 Process exited after 0.08853 seconds with return value 0
 Press any key to continue . . . |

3. Penjelasan

1. Struktur Data Graph

Program ini menggunakan representasi graph dengan list adjasensi. Setiap simpul dalam graph direpresentasikan oleh struktur data Node. Graph secara keseluruhan direpresentasikan oleh struktur data Graph. Setiap simpul memiliki daftar simpul-simpul bertetanggaannya yang disimpan dalam array adjList pada struktur Graph.

2. Langkah-langkah algoritma BFS

Langkah 1: Inisialisasi

- Mulai dari simpul awal yang ditentukan (biasanya simpul 0).
- Tandai simpul awal tersebut sebagai "dikunjungi".
- Masukkan simpul awal ke dalam queue.

Langkah 2: Proses BFS

- Selama queue tidak kosong, lakukan langkah-langkah berikut:
- Ambil simpul pertama dari queue.
- Proses simpul tersebut (cetak atau lakukan operasi yang diinginkan).
- Tandai semua simpul bertetanggaannya dengan simpul tersebut sebagai "dikunjungi".
- Masukkan simpul-simpul bertetanggaannya yang belum dikunjungi ke dalam queue.

Langkah 3: Ulangi

- Ulangi langkah-langkah di atas hingga tidak ada simpul lagi yang tersisa dalam queue.

3. Penggunaan Queue dalam BFS

Queue digunakan untuk menyimpan simpul-simpul yang akan dikunjungi selanjutnya. Saat sebuah simpul diproses, semua simpul bertetanggaannya dimasukkan ke dalam queue. Simpul yang berada di depan queue diproses terlebih dahulu sesuai dengan prinsip FIFO (First-In-First-Out).

Penggunaan queue memastikan bahwa simpul-simpul dikunjungi sesuai dengan urutan yang sesuai dengan prinsip BFS, yaitu secara berurutan dari simpul awal ke simpul-simpul yang bertetangga, kemudian ke simpul-simpul yang lebih jauh.

Dengan menggunakan algoritma BFS, program ini dapat digunakan untuk melakukan traversal pada graph dan mencari lintasan terpendek antara dua simpul pada graph tak berarah.