

LAPORAN PRAKTIKUM

STRUKTUR DATA

MODUL XIV

GRAPH



Disusun Oleh :

Nama : Besthian Guido Rafael Simbolon
NIM : 103112430258

Dosen

FAHRUDIN MUKTI WIBOWO

**PROGRAM STUDI STRUKTUR DATA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY PURWOKERTO
2025**

A. Dasar Teori

Graf (Graph) adalah struktur data non-linear yang digunakan untuk menggambarkan hubungan antar objek, yang terdiri dari **simpul (vertex)** dan **sisi (edge)** sebagai penghubung antar simpul. Dalam pemrograman, graf sering direpresentasikan menggunakan **Adjacency List**, yaitu setiap simpul memiliki daftar tetangga yang disimpan dalam bentuk linked list, karena cara ini lebih hemat memori dibandingkan Adjacency Matrix, terutama pada graf yang jumlah sisinya sedikit (sparse). Untuk menelusuri graf, terdapat dua algoritma utama yaitu **Depth First Search (DFS)** dan **Breadth First Search (BFS)**. DFS menelusuri graf secara mendalam dengan mengunjungi satu jalur sampai habis sebelum berpindah ke jalur lain, biasanya menggunakan rekursi atau stack. Sementara itu, BFS menelusuri graf secara bertahap per level dengan mengunjungi semua simpul terdekat terlebih dahulu menggunakan queue. Kedua algoritma ini penting untuk memahami cara menjelajahi dan mengelola data dalam struktur graf.

B. Guided

Guided 1

main.cpp

The screenshot shows a code editor interface with multiple files open in tabs. The current file is `main.cpp guided`. The code implements a graph creation and traversal process:

```
#include "graf.h"
#include "graf.cpp"
#include <iostream>
using namespace std;

int main() {
    Graph G;
    createGraph(G);

    // Tambah Node
    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');

    // Hubungkan Node (graph tidak berarah)
    ConnectNode(G, 'A', 'B');
    ConnectNode(G, 'A', 'C');
    ConnectNode(G, 'B', 'D');
    ConnectNode(G, 'C', 'E');

    cout << "==== Struktur Graph ====\n";
    PrintInfoGraph(G);

    cout << "\n==== DFS dari Node A ====\n";
    ResetVisited(G);
    PrintDFS(G, FindNode(G, 'A'));

    cout << "\n==== BFS dari Node B ====\n";
    ResetVisited(G);
    PrintBFS(G, FindNode(G, 'A'));

    cout << endl;
}
```

To the right of the code editor, a terminal window displays the output of the program:

```
Besthian Guido Rafael Simbolon
103112430258
IF - 12 - 06 |
```

The terminal also shows the current file path as `Modul 14` and the line number as `Ln 3, Col 14`.

graf.cpp

The screenshot shows a code editor interface with several tabs open. The tabs include `graf.cpp`, `graf.h`, `main.cpp`, `graph.cpp`, `graph.h`, and `main.cpp`. The `graf.cpp` tab is currently active, displaying the following C++ code:

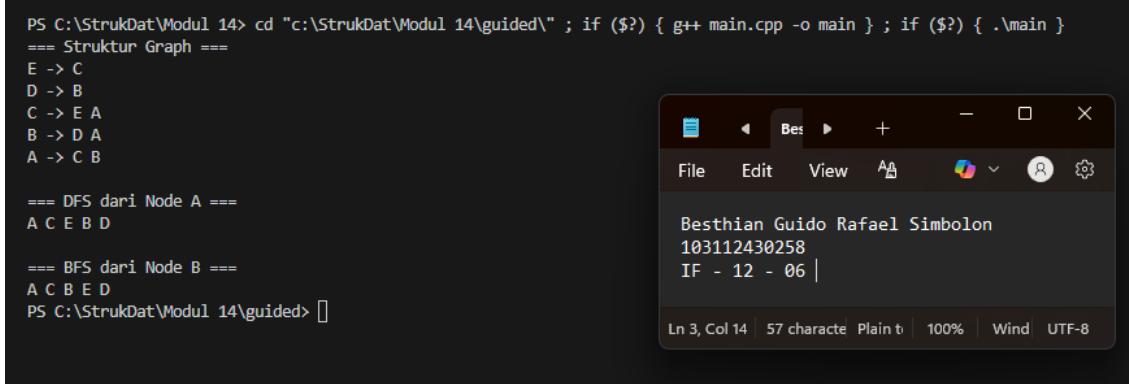
```
1 #include "graf.h"
2 #include <queue>
3 #include <stack>
4
5 void createGraph(Graph &G) {
6     G.first = NULL;
7 }
8
9 addrNode AllocateNode(infograf X) {
10    addrNode P = new ElmNode;
11    P->info = X;
12    P->visited = 0;
13    P->firstEdge = NULL;
14    P->next = NULL;
15    return P;
16 }
17
18 addrEdge AllocateEdge(addrNode N) {
19    addrEdge P = new ElmEdge;
20    P->node = N;
21    P->next = NULL;
22    return P;
23 }
24
25 void InsertNode(Graph &G, infograf X) {
26    addrNode P = AllocateNode(X);
27    P->next = G.first;
28    G.first = P;
29 }
30
31 addrNode FindNode(Graph G, infograf X) {
32    addrNode P = G.first;
33    while (P != NULL) {
34        if (P->info == X)
35            return P;
36    }
37 }
```

graf.h

The screenshot shows a C++ development environment with the following details:

- File Explorer:** Shows files in the current workspace, including `graf.cpp`, `graf.h`, `main.cpp`, `graph.cpp`, `graph.h`, and `main.cpp` (two instances).
- Editor:** The main editor window displays the `graf.h` file. The code defines a `Graph` class containing `ElmNode` and `ElmEdge` structures. Each node has fields for `info`, `visited`, `firstEdge`, and `next`. Each edge has fields for `node` and `next`. A `createGraph` function is also shown.
- Bottom Status Bar:** Shows the current line (Ln 11), character count (57), and other settings like font size (100%), window type (Wind), and encoding (UTF-8).

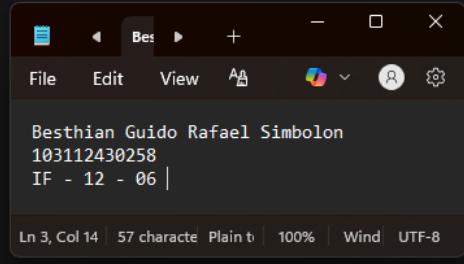
Screenshots Output



```
PS C:\StrukDat\Modul 14> cd "c:\StrukDat\Modul 14\guided\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\main }
 === Struktur Graph ===
 E -> C
 D -> B
 C -> E A
 B -> D A
 A -> C B

 === DFS dari Node A ===
 A C E B D

 === BFS dari Node B ===
 A C B E D
PS C:\StrukDat\Modul 14\guided>
```



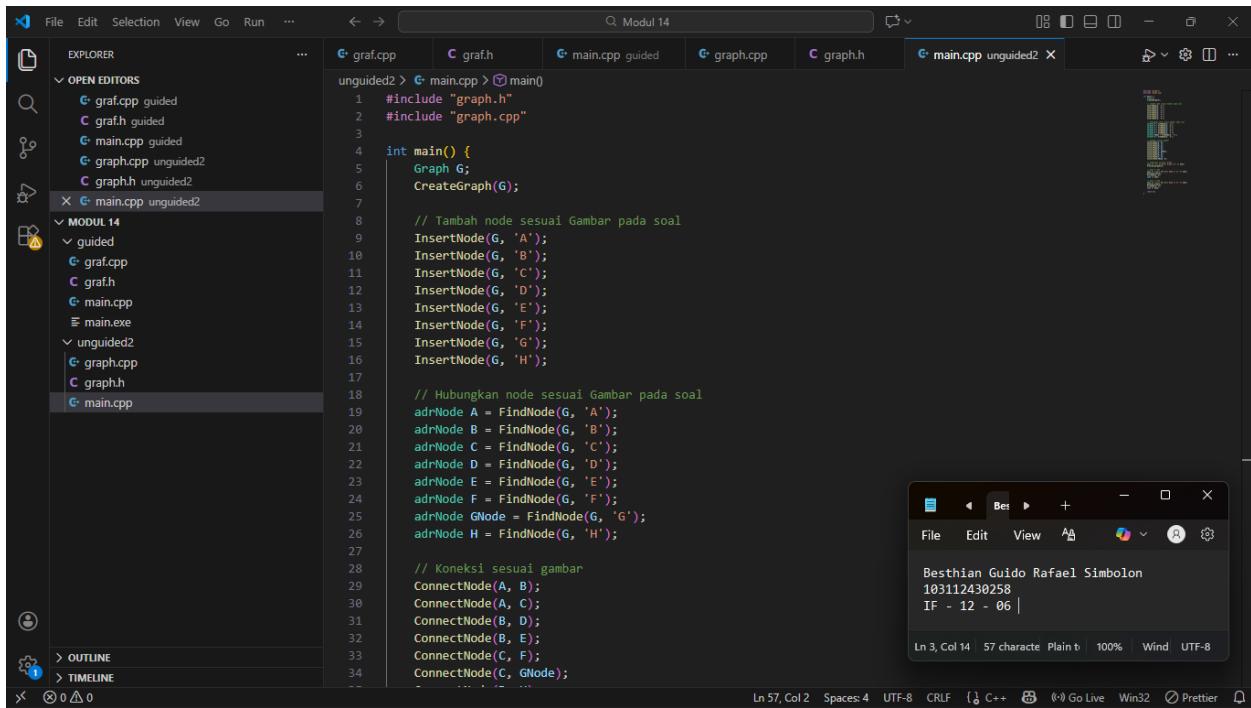
Deskripsi:

Program ini merupakan implementasi struktur data Graph tak berarah (Undirected Graph) menggunakan bahasa C++ dengan representasi adjacency list. Program mendefinisikan tipe data dinamis untuk node (simpul) dan edge (sisi) menggunakan pointer, sehingga setiap node memiliki daftar koneksi ke node-node lain yang terhubung dengannya. Secara fungsional, program memungkinkan pembuatan graph, penambahan node (seperti A, B, C, D, dan E), serta penghubungan antar node tersebut. Setelah struktur graph terbentuk, program menampilkan hubungan atau koneksi antar node secara visual ke layar. Selain itu, program ini juga mendemonstrasikan dua algoritma penelusuran utama, yaitu Depth First Search (DFS) yang melakukan penelusuran secara mendalam dengan menggunakan rekursi, dan Breadth First Search (BFS) yang melakukan penelusuran secara melebar dengan bantuan struktur data queue.

C. Unguided

Unguided 1

Main.cpp



The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a tree view of files under "MODUL 14". The "unguided2" folder is expanded, showing "main.cpp unguided2" which is currently selected.
- Editor:** The main editor area displays the content of "main.cpp unguided2". The code is as follows:

```
#include "graph.h"
#include "graph.cpp"

int main() {
    Graph G;
    CreateGraph(G);

    // Tambah node sesuai Gambar pada soal
    InsertNode(G, 'A');
    InsertNode(G, 'B');
    InsertNode(G, 'C');
    InsertNode(G, 'D');
    InsertNode(G, 'E');
    InsertNode(G, 'F');
    InsertNode(G, 'G');
    InsertNode(G, 'H');

    // Hubungkan node sesuai Gambar pada soal
    adrNode A = FindNode(G, 'A');
    adrNode B = FindNode(G, 'B');
    adrNode C = FindNode(G, 'C');
    adrNode D = FindNode(G, 'D');
    adrNode E = FindNode(G, 'E');
    adrNode F = FindNode(G, 'F');
    adrNode GNode = FindNode(G, 'G');
    adrNode H = FindNode(G, 'H');

    // Koneksi sesuai gambar
    ConnectNode(A, B);
    ConnectNode(A, C);
    ConnectNode(B, D);
    ConnectNode(B, E);
    ConnectNode(C, F);
    ConnectNode(C, GNode);
    ConnectNode(D, E);
    ConnectNode(D, F);
    ConnectNode(E, F);
    ConnectNode(F, G);
    ConnectNode(F, H);
    ConnectNode(G, H);
}
```

The status bar at the bottom indicates the file is saved (0 changes), has 57 characters, is in Plain text mode, and is 100% zoomed.

Graph.cpp

File Edit Selection View Go Run ...

Modul 14

EXPLORER

OPEN EDITORS

- graph.cpp guided
- graph.h guided
- main.cpp guided
- graph.cpp unguided2
- graph.h unguided2
- main.cpp unguided2

MODUL 14

- guided
- graph.cpp
- graph.h
- main.cpp
- main.exe
- unguided2
- graph.cpp
- graph.h
- main.cpp

```

1 #include "graph.h"
2 #include <queue>
3
4 void CreateGraph(Graph &G) {
5     G.first = NULL;
6 }
7
8 adrNode AllocateNode(infoGraph X) {
9     adrNode P = new ElmNode;
10    P->info = X;
11    P->visited = 0;
12    P->firstEdge = NULL;
13    P->next = NULL;
14    return P;
15 }
16
17 adrEdge AllocateEdge(adrNode N) {
18     adrEdge P = new ElmEdge;
19     P->node = N;
20     P->next = NULL;
21     return P;
22 }
23
24 void InsertNode(Graph &G, infoGraph X) {
25     adrNode P = AllocateNode(X);
26     P->next = G.first;
27     G.first = P;
28 }
29
30 adrNode FindNode(Graph G, infoGraph X) {
31     adrNode P = G.first;
32     while (P != NULL) {
33         if (P->info == X)
34             return P;
35     }
36 }
37
38 void PrintBFS(Graph G, adrNode)
39 {
40     queue<adrNode> Q;
41     adrNode P;
42     Q.push(G.first);
43     while (!Q.empty()) {
44         P = Q.front();
45         cout << P->info;
46         Q.pop();
47         for (int i = 0; i < G.edges; i++) {
48             if (G.edges[i] == P->next)
49                 Q.push(G.edges[i]);
50         }
51     }
52 }
53
54 void ConnectNode(adrNode N1, adrNode N2);
55
56 void PrintGraph(Graph G)
57 {
58     adrNode P;
59     cout << "Graph: ";
60     for (int i = 0; i < G.edges; i++) {
61         P = G.edges[i];
62         cout << P->node->info;
63     }
64 }
65
66 void PrintInfo(adrNode P)
67 {
68     cout << "Info: ";
69     cout << P->info;
70 }
71
72 void PrintVisited(adrNode P)
73 {
74     cout << "Visited: ";
75     cout << P->visited;
76 }
77
78 void PrintFirstEdge(adrNode P)
79 {
80     cout << "First Edge: ";
81     cout << P->firstEdge->node->info;
82 }
83
84 void PrintNext(adrNode P)
85 {
86     cout << "Next: ";
87     cout << P->next->info;
88 }
89
90 void PrintElmNode(ElmNode P)
91 {
92     cout << "ElmNode: ";
93     cout << P.info;
94 }
95
96 void PrintElmEdge(ElmEdge P)
97 {
98     cout << "ElmEdge: ";
99     cout << P.node->info;
100 }
101
102 void PrintGraph(adrGraph G)
103 {
104     adrGraph P;
105     cout << "Graph: ";
106     for (int i = 0; i < G.edges; i++) {
107         P = G.edges[i];
108         cout << P->node->info;
109     }
110 }
111
112 void PrintInfo(adrGraph P)
113 {
114     cout << "Info: ";
115     cout << P->info;
116 }
117
118 void PrintVisited(adrGraph P)
119 {
120     cout << "Visited: ";
121     cout << P->visited;
122 }
123
124 void PrintFirstEdge(adrGraph P)
125 {
126     cout << "First Edge: ";
127     cout << P->firstEdge->node->info;
128 }
129
130 void PrintNext(adrGraph P)
131 {
132     cout << "Next: ";
133     cout << P->next->info;
134 }
135
136 void PrintElmNode(ElmNode P)
137 {
138     cout << "ElmNode: ";
139     cout << P.info;
140 }
141
142 void PrintElmEdge(ElmEdge P)
143 {
144     cout << "ElmEdge: ";
145     cout << P.node->info;
146 }
147
148 void PrintGraph(adrGraph G)
149 {
150     adrGraph P;
151     cout << "Graph: ";
152     for (int i = 0; i < G.edges; i++) {
153         P = G.edges[i];
154         cout << P->node->info;
155     }
156 }
157
158 void PrintInfo(adrGraph P)
159 {
160     cout << "Info: ";
161     cout << P->info;
162 }
163
164 void PrintVisited(adrGraph P)
165 {
166     cout << "Visited: ";
167     cout << P->visited;
168 }
169
170 void PrintFirstEdge(adrGraph P)
171 {
172     cout << "First Edge: ";
173     cout << P->firstEdge->node->info;
174 }
175
176 void PrintNext(adrGraph P)
177 {
178     cout << "Next: ";
179     cout << P->next->info;
180 }
181
182 void PrintElmNode(ElmNode P)
183 {
184     cout << "ElmNode: ";
185     cout << P.info;
186 }
187
188 void PrintElmEdge(ElmEdge P)
189 {
190     cout << "ElmEdge: ";
191     cout << P.node->info;
192 }
193
194 void PrintGraph(adrGraph G)
195 {
196     adrGraph P;
197     cout << "Graph: ";
198     for (int i = 0; i < G.edges; i++) {
199         P = G.edges[i];
200         cout << P->node->info;
201     }
202 }
203
204 void PrintInfo(adrGraph P)
205 {
206     cout << "Info: ";
207     cout << P->info;
208 }
209
210 void PrintVisited(adrGraph P)
211 {
212     cout << "Visited: ";
213     cout << P->visited;
214 }
215
216 void PrintFirstEdge(adrGraph P)
217 {
218     cout << "First Edge: ";
219     cout << P->firstEdge->node->info;
220 }
221
222 void PrintNext(adrGraph P)
223 {
224     cout << "Next: ";
225     cout << P->next->info;
226 }
227
228 void PrintElmNode(ElmNode P)
229 {
230     cout << "ElmNode: ";
231     cout << P.info;
232 }
233
234 void PrintElmEdge(ElmEdge P)
235 {
236     cout << "ElmEdge: ";
237     cout << P.node->info;
238 }
239
240 void PrintGraph(adrGraph G)
241 {
242     adrGraph P;
243     cout << "Graph: ";
244     for (int i = 0; i < G.edges; i++) {
245         P = G.edges[i];
246         cout << P->node->info;
247     }
248 }
249
250 void PrintInfo(adrGraph P)
251 {
252     cout << "Info: ";
253     cout << P->info;
254 }
255
256 void PrintVisited(adrGraph P)
257 {
258     cout << "Visited: ";
259     cout << P->visited;
260 }
261
262 void PrintFirstEdge(adrGraph P)
263 {
264     cout << "First Edge: ";
265     cout << P->firstEdge->node->info;
266 }
267
268 void PrintNext(adrGraph P)
269 {
270     cout << "Next: ";
271     cout << P->next->info;
272 }
273
274 void PrintElmNode(ElmNode P)
275 {
276     cout << "ElmNode: ";
277     cout << P.info;
278 }
279
280 void PrintElmEdge(ElmEdge P)
281 {
282     cout << "ElmEdge: ";
283     cout << P.node->info;
284 }
285
286 void PrintGraph(adrGraph G)
287 {
288     adrGraph P;
289     cout << "Graph: ";
290     for (int i = 0; i < G.edges; i++) {
291         P = G.edges[i];
292         cout << P->node->info;
293     }
294 }
295
296 void PrintInfo(adrGraph P)
297 {
298     cout << "Info: ";
299     cout << P->info;
300 }
301
302 void PrintVisited(adrGraph P)
303 {
304     cout << "Visited: ";
305     cout << P->visited;
306 }
307
308 void PrintFirstEdge(adrGraph P)
309 {
310     cout << "First Edge: ";
311     cout << P->firstEdge->node->info;
312 }
313
314 void PrintNext(adrGraph P)
315 {
316     cout << "Next: ";
317     cout << P->next->info;
318 }
319
320 void PrintElmNode(ElmNode P)
321 {
322     cout << "ElmNode: ";
323     cout << P.info;
324 }
325
326 void PrintElmEdge(ElmEdge P)
327 {
328     cout << "ElmEdge: ";
329     cout << P.node->info;
330 }
331
332 void PrintGraph(adrGraph G)
333 {
334     adrGraph P;
335     cout << "Graph: ";
336     for (int i = 0; i < G.edges; i++) {
337         P = G.edges[i];
338         cout << P->node->info;
339     }
340 }
341
342 void PrintInfo(adrGraph P)
343 {
344     cout << "Info: ";
345     cout << P->info;
346 }
347
348 void PrintVisited(adrGraph P)
349 {
350     cout << "Visited: ";
351     cout << P->visited;
352 }
353
354 void PrintFirstEdge(adrGraph P)
355 {
356     cout << "First Edge: ";
357     cout << P->firstEdge->node->info;
358 }
359
360 void PrintNext(adrGraph P)
361 {
362     cout << "Next: ";
363     cout << P->next->info;
364 }
365
366 void PrintElmNode(ElmNode P)
367 {
368     cout << "ElmNode: ";
369     cout << P.info;
370 }
371
372 void PrintElmEdge(ElmEdge P)
373 {
374     cout << "ElmEdge: ";
375     cout << P.node->info;
376 }
377
378 void PrintGraph(adrGraph G)
379 {
380     adrGraph P;
381     cout << "Graph: ";
382     for (int i = 0; i < G.edges; i++) {
383         P = G.edges[i];
384         cout << P->node->info;
385     }
386 }
387
388 void PrintInfo(adrGraph P)
389 {
390     cout << "Info: ";
391     cout << P->info;
392 }
393
394 void PrintVisited(adrGraph P)
395 {
396     cout << "Visited: ";
397     cout << P->visited;
398 }
399
400 void PrintFirstEdge(adrGraph P)
401 {
402     cout << "First Edge: ";
403     cout << P->firstEdge->node->info;
404 }
405
406 void PrintNext(adrGraph P)
407 {
408     cout << "Next: ";
409     cout << P->next->info;
410 }
411
412 void PrintElmNode(ElmNode P)
413 {
414     cout << "ElmNode: ";
415     cout << P.info;
416 }
417
418 void PrintElmEdge(ElmEdge P)
419 {
420     cout << "ElmEdge: ";
421     cout << P.node->info;
422 }
423
424 void PrintGraph(adrGraph G)
425 {
426     adrGraph P;
427     cout << "Graph: ";
428     for (int i = 0; i < G.edges; i++) {
429         P = G.edges[i];
430         cout << P->node->info;
431     }
432 }
433
434 void PrintInfo(adrGraph P)
435 {
436     cout << "Info: ";
437     cout << P->info;
438 }
439
440 void PrintVisited(adrGraph P)
441 {
442     cout << "Visited: ";
443     cout << P->visited;
444 }
445
446 void PrintFirstEdge(adrGraph P)
447 {
448     cout << "First Edge: ";
449     cout << P->firstEdge->node->info;
450 }
451
452 void PrintNext(adrGraph P)
453 {
454     cout << "Next: ";
455     cout << P->next->info;
456 }
457
458 void PrintElmNode(ElmNode P)
459 {
460     cout << "ElmNode: ";
461     cout << P.info;
462 }
463
464 void PrintElmEdge(ElmEdge P)
465 {
466     cout << "ElmEdge: ";
467     cout << P.node->info;
468 }
469
470 void PrintGraph(adrGraph G)
471 {
472     adrGraph P;
473     cout << "Graph: ";
474     for (int i = 0; i < G.edges; i++) {
475         P = G.edges[i];
476         cout << P->node->info;
477     }
478 }
479
480 void PrintInfo(adrGraph P)
481 {
482     cout << "Info: ";
483     cout << P->info;
484 }
485
486 void PrintVisited(adrGraph P)
487 {
488     cout << "Visited: ";
489     cout << P->visited;
490 }
491
492 void PrintFirstEdge(adrGraph P)
493 {
494     cout << "First Edge: ";
495     cout << P->firstEdge->node->info;
496 }
497
498 void PrintNext(adrGraph P)
499 {
500     cout << "Next: ";
501     cout << P->next->info;
502 }
503
504 void PrintElmNode(ElmNode P)
505 {
506     cout << "ElmNode: ";
507     cout << P.info;
508 }
509
510 void PrintElmEdge(ElmEdge P)
511 {
512     cout << "ElmEdge: ";
513     cout << P.node->info;
514 }
515
516 void PrintGraph(adrGraph G)
517 {
518     adrGraph P;
519     cout << "Graph: ";
520     for (int i = 0; i < G.edges; i++) {
521         P = G.edges[i];
522         cout << P->node->info;
523     }
524 }
525
526 void PrintInfo(adrGraph P)
527 {
528     cout << "Info: ";
529     cout << P->info;
530 }
531
532 void PrintVisited(adrGraph P)
533 {
534     cout << "Visited: ";
535     cout << P->visited;
536 }
537
538 void PrintFirstEdge(adrGraph P)
539 {
540     cout << "First Edge: ";
541     cout << P->firstEdge->node->info;
542 }
543
544 void PrintNext(adrGraph P)
545 {
546     cout << "Next: ";
547     cout << P->next->info;
548 }
549
550 void PrintElmNode(ElmNode P)
551 {
552     cout << "ElmNode: ";
553     cout << P.info;
554 }
555
556 void PrintElmEdge(ElmEdge P)
557 {
558     cout << "ElmEdge: ";
559     cout << P.node->info;
560 }
561
562 void PrintGraph(adrGraph G)
563 {
564     adrGraph P;
565     cout << "Graph: ";
566     for (int i = 0; i < G.edges; i++) {
567         P = G.edges[i];
568         cout << P->node->info;
569     }
570 }
571
572 void PrintInfo(adrGraph P)
573 {
574     cout << "Info: ";
575     cout << P->info;
576 }
577
578 void PrintVisited(adrGraph P)
579 {
580     cout << "Visited: ";
581     cout << P->visited;
582 }
583
584 void PrintFirstEdge(adrGraph P)
585 {
586     cout << "First Edge: ";
587     cout << P->firstEdge->node->info;
588 }
589
590 void PrintNext(adrGraph P)
591 {
592     cout << "Next: ";
593     cout << P->next->info;
594 }
595
596 void PrintElmNode(ElmNode P)
597 {
598     cout << "ElmNode: ";
599     cout << P.info;
600 }
601
602 void PrintElmEdge(ElmEdge P)
603 {
604     cout << "ElmEdge: ";
605     cout << P.node->info;
606 }
607
608 void PrintGraph(adrGraph G)
609 {
610     adrGraph P;
611     cout << "Graph: ";
612     for (int i = 0; i < G.edges; i++) {
613         P = G.edges[i];
614         cout << P->node->info;
615     }
616 }
617
618 void PrintInfo(adrGraph P)
619 {
620     cout << "Info: ";
621     cout << P->info;
622 }
623
624 void PrintVisited(adrGraph P)
625 {
626     cout << "Visited: ";
627     cout << P->visited;
628 }
629
630 void PrintFirstEdge(adrGraph P)
631 {
632     cout << "First Edge: ";
633     cout << P->firstEdge->node->info;
634 }
635
636 void PrintNext(adrGraph P)
637 {
638     cout << "Next: ";
639     cout << P->next->info;
640 }
641
642 void PrintElmNode(ElmNode P)
643 {
644     cout << "ElmNode: ";
645     cout << P.info;
646 }
647
648 void PrintElmEdge(ElmEdge P)
649 {
650     cout << "ElmEdge: ";
651     cout << P.node->info;
652 }
653
654 void PrintGraph(adrGraph G)
655 {
656     adrGraph P;
657     cout << "Graph: ";
658     for (int i = 0; i < G.edges; i++) {
659         P = G.edges[i];
660         cout << P->node->info;
661     }
662 }
663
664 void PrintInfo(adrGraph P)
665 {
666     cout << "Info: ";
667     cout << P->info;
668 }
669
670 void PrintVisited(adrGraph P)
671 {
672     cout << "Visited: ";
673     cout << P->visited;
674 }
675
676 void PrintFirstEdge(adrGraph P)
677 {
678     cout << "First Edge: ";
679     cout << P->firstEdge->node->info;
680 }
681
682 void PrintNext(adrGraph P)
683 {
684     cout << "Next: ";
685     cout << P->next->info;
686 }
687
688 void PrintElmNode(ElmNode P)
689 {
690     cout << "ElmNode: ";
691     cout << P.info;
692 }
693
694 void PrintElmEdge(ElmEdge P)
695 {
696     cout << "ElmEdge: ";
697     cout << P.node->info;
698 }
699
700 void PrintGraph(adrGraph G)
701 {
702     adrGraph P;
703     cout << "Graph: ";
704     for (int i = 0; i < G.edges; i++) {
705         P = G.edges[i];
706         cout << P->node->info;
707     }
708 }
709
710 void PrintInfo(adrGraph P)
711 {
712     cout << "Info: ";
713     cout << P->info;
714 }
715
716 void PrintVisited(adrGraph P)
717 {
718     cout << "Visited: ";
719     cout << P->visited;
720 }
721
722 void PrintFirstEdge(adrGraph P)
723 {
724     cout << "First Edge: ";
725     cout << P->firstEdge->node->info;
726 }
727
728 void PrintNext(adrGraph P)
729 {
730     cout << "Next: ";
731     cout << P->next->info;
732 }
733
734 void PrintElmNode(ElmNode P)
735 {
736     cout << "ElmNode: ";
737     cout << P.info;
738 }
739
740 void PrintElmEdge(ElmEdge P)
741 {
742     cout << "ElmEdge: ";
743     cout << P.node->info;
744 }
745
746 void PrintGraph(adrGraph G)
747 {
748     adrGraph P;
749     cout << "Graph: ";
750     for (int i = 0; i < G.edges; i++) {
751         P = G.edges[i];
752         cout << P->node->info;
753     }
754 }
755
756 void PrintInfo(adrGraph P)
757 {
758     cout << "Info: ";
759     cout << P->info;
760 }
761
762 void PrintVisited(adrGraph P)
763 {
764     cout << "Visited: ";
765     cout << P->visited;
766 }
767
768 void PrintFirstEdge(adrGraph P)
769 {
770     cout << "First Edge: ";
771     cout << P->firstEdge->node->info;
772 }
773
774 void PrintNext(adrGraph P)
775 {
776     cout << "Next: ";
777     cout << P->next->info;
778 }
779
780 void PrintElmNode(ElmNode P)
781 {
782     cout << "ElmNode: ";
783     cout << P.info;
784 }
785
786 void PrintElmEdge(ElmEdge P)
787 {
788     cout << "ElmEdge: ";
789     cout << P.node->info;
790 }
791
792 void PrintGraph(adrGraph G)
793 {
794     adrGraph P;
795     cout << "Graph: ";
796     for (int i = 0; i < G.edges; i++) {
797         P = G.edges[i];
798         cout << P->node->info;
799     }
800 }
801
802 void PrintInfo(adrGraph P)
803 {
804     cout << "Info: ";
805     cout << P->info;
806 }
807
808 void PrintVisited(adrGraph P)
809 {
810     cout << "Visited: ";
811     cout << P->visited;
812 }
813
814 void PrintFirstEdge(adrGraph P)
815 {
816     cout << "First Edge: ";
817     cout << P->firstEdge->node->info;
818 }
819
820 void PrintNext(adrGraph P)
821 {
822     cout << "Next: ";
823     cout << P->next->info;
824 }
825
826 void PrintElmNode(ElmNode P)
827 {
828     cout << "ElmNode: ";
829     cout << P.info;
830 }
831
832 void PrintElmEdge(ElmEdge P)
833 {
834     cout << "ElmEdge: ";
835     cout << P.node->info;
836 }
837
838 void PrintGraph(adrGraph G)
839 {
840     adrGraph P;
841     cout << "Graph: ";
842     for (int i = 0; i < G.edges; i++) {
843         P = G.edges[i];
844         cout << P->node->info;
845     }
846 }
847
848 void PrintInfo(adrGraph P)
849 {
850     cout << "Info: ";
851     cout << P->info;
852 }
853
854 void PrintVisited(adrGraph P)
855 {
856     cout << "Visited: ";
857     cout << P->visited;
858 }
859
860 void PrintFirstEdge(adrGraph P)
861 {
862     cout << "First Edge: ";
863     cout << P->firstEdge->node->info;
864 }
865
866 void PrintNext(adrGraph P)
867 {
868     cout << "Next: ";
869     cout << P->next->info;
870 }
871
872 void PrintElmNode(ElmNode P)
873 {
874     cout << "ElmNode: ";
875     cout << P.info;
876 }
877
878 void PrintElmEdge(ElmEdge P)
879 {
880     cout << "ElmEdge: ";
881     cout << P.node->info;
882 }
883
884 void PrintGraph(adrGraph G)
885 {
886     adrGraph P;
887     cout << "Graph: ";
888     for (int i = 0; i < G.edges; i++) {
889         P = G.edges[i];
890         cout << P->node->info;
891     }
892 }
893
894 void PrintInfo(adrGraph P)
895 {
896     cout << "Info: ";
897     cout << P->info;
898 }
899
900 void PrintVisited(adrGraph P)
901 {
902     cout << "Visited: ";
903     cout << P->visited;
904 }
905
906 void PrintFirstEdge(adrGraph P)
907 {
908     cout << "First Edge: ";
909     cout << P->firstEdge->node->info;
910 }
911
912 void PrintNext(adrGraph P)
913 {
914     cout << "Next: ";
915     cout << P->next->info;
916 }
917
918 void PrintElmNode(ElmNode P)
919 {
920     cout << "ElmNode: ";
921     cout << P.info;
922 }
923
924 void PrintElmEdge(ElmEdge P)
925 {
926     cout << "ElmEdge: ";
927     cout << P.node->info;
928 }
929
930 void PrintGraph(adrGraph G)
931 {
932     adrGraph P;
933     cout << "Graph: ";
934     for (int i = 0; i < G.edges; i++) {
935         P = G.edges[i];
936         cout << P->node->info;
937     }
938 }
939
940 void PrintInfo(adrGraph P)
941 {
942     cout << "Info: ";
943     cout << P->info;
944 }
945
946 void PrintVisited(adrGraph P)
947 {
948     cout << "Visited: ";
949     cout << P->visited;
950 }
951
952 void PrintFirstEdge(adrGraph P)
953 {
954     cout << "First Edge: ";
955     cout << P->firstEdge->node->info;
956 }
957
958 void PrintNext(adrGraph P)
959 {
960     cout << "Next: ";
961     cout << P->next->info;
962 }
963
964 void PrintElmNode(ElmNode P)
965 {
966     cout << "ElmNode: ";
967     cout << P.info;
968 }
969
970 void PrintElmEdge(ElmEdge P)
971 {
972     cout << "ElmEdge: ";
973     cout << P.node->info;
974 }
975
976 void PrintGraph(adrGraph G)
977 {
978     adrGraph P;
979     cout << "Graph: ";
980     for (int i = 0; i < G.edges; i++) {
981         P = G.edges[i];
982         cout << P->node->info;
983     }
984 }
985
986 void PrintInfo(adrGraph P)
987 {
988     cout << "Info: ";
989     cout << P->info;
990 }
991
992 void PrintVisited(adrGraph P)
993 {
994     cout << "Visited: ";
995     cout << P->visited;
996 }
997
998 void PrintFirstEdge(adrGraph P)
999 {
1000     cout << "First Edge: ";
1001     cout << P->firstEdge->node->info;
1002 }
1003
1004 void PrintNext(adrGraph P)
1005 {
1006     cout << "Next: ";
1007     cout << P->next->info;
1008 }
1009
1010 void PrintElmNode(ElmNode P)
1011 {
1012     cout << "ElmNode: ";
1013     cout << P.info;
1014 }
1015
1016 void PrintElmEdge(ElmEdge P)
1017 {
1018     cout << "ElmEdge: ";
1019     cout << P.node->info;
1020 }
1021
1022 void PrintGraph(adrGraph G)
1023 {
1024     adrGraph P;
1025     cout << "Graph: ";
1026     for (int i = 0; i < G.edges; i++) {
1027         P = G.edges[i];
1028         cout << P->node->info;
1029     }
1030 }
1031
1032 void PrintInfo(adrGraph P)
1033 {
1034     cout << "Info: ";
1035     cout << P->info;
1036 }
1037
1038 void PrintVisited(adrGraph P)
1039 {
1040     cout << "Visited: ";
1041     cout << P->visited;
1042 }
1043
1044 void PrintFirstEdge(adrGraph P)
1045 {
1046     cout << "First Edge: ";
1047     cout << P->firstEdge->node->info;
1048 }
1049
1050 void PrintNext(adrGraph P)
1051 {
1052     cout << "Next: ";
1053     cout << P->next->info;
1054 }
1055
1056 void PrintElmNode(ElmNode P)
1057 {
1058     cout << "ElmNode: ";
1059     cout << P.info;
1060 }
1061
1062 void PrintElmEdge(ElmEdge P)
1063 {
1064     cout << "ElmEdge: ";
1065     cout << P.node->info;
1066 }
1067
1068 void PrintGraph(adrGraph G)
1069 {
1070     adrGraph P;
1071     cout << "Graph: ";
1072     for (int i = 0; i < G.edges; i++) {
1073         P = G.edges[i];
1074         cout << P->node->info;
1075     }
1076 }
1077
1078 void PrintInfo(adrGraph P)
1079 {
1080     cout << "Info: ";
1081     cout << P->info;
1082 }
1083
1084 void PrintVisited(adrGraph P)
1085 {
1086     cout << "Visited: ";
1087     cout << P->visited;
1088 }
1089
1090 void PrintFirstEdge(adrGraph P)
1091 {
1092     cout << "First Edge: ";
1093     cout << P->firstEdge->node->info;
1094 }
1095
1096 void PrintNext(adrGraph P)
1097 {
1098     cout << "Next: ";
1099     cout << P->next->info;
1100 }
1101
1102 void PrintElmNode(ElmNode P)
1103 {
1104     cout << "ElmNode: ";
1105     cout << P.info;
1106 }
1107
1108 void PrintElmEdge(ElmEdge P)
1109 {
1110     cout << "ElmEdge: ";
1111     cout << P.node->info;
1112 }
1113
1114 void PrintGraph(adrGraph G)
1115 {
1116     adrGraph P;
1117     cout << "Graph: ";
1118     for (int i = 0; i < G.edges; i++) {
1119         P = G.edges[i];
1120         cout << P->node->info;
1121     }
1122 }
1123
1124 void PrintInfo(adrGraph P)
1125 {
1126     cout << "Info: ";
1127     cout << P->info;
1128 }
1129
1130 void PrintVisited(adrGraph P)
1131 {
1132     cout << "Visited: ";
1133     cout << P->visited;
1134 }
1135
1136 void PrintFirstEdge(adrGraph P)
1137 {
1138     cout << "First Edge: ";
1139     cout << P->firstEdge->node->info;
1140 }
1141
1142 void PrintNext(adrGraph P)
1143 {
1144     cout << "Next: ";
1145     cout << P->next->info;
1146 }
1147
1148 void PrintElmNode(ElmNode P)
1149 {
1150     cout << "ElmNode: ";
1151     cout << P.info;
1152 }
1153
1154 void PrintElmEdge(ElmEdge P)
1155 {
1156     cout << "ElmEdge: ";
1157     cout << P.node->info;

```

Screenshot Output

The screenshot shows a terminal window and a code editor window side-by-side.

Terminal Output:

```
PROBLEMS OUTPUT TERMINAL PORTS DEBUG CONSOLE
PS C:\StrukDat\Modul 14\unguided2> cd "c:\StrukDat\Modul 14\unguided2\" ; if ($?) { g++ main.cpp -o main } ; if ($?) { .\ma
in }
== Struktur Graph ==
H -> G F E D
G -> H C
F -> H C
E -> H B
D -> H B
C -> G F A
B -> E D A
A -> C B

== DFS dari Node A ==
A C G H F E B D

== BFS dari Node A ==
A C B G F E D H
PS C:\StrukDat\Modul 14\unguided2> []
```

Code Editor Window:

```
Bestian Guido Rafael Simbolon
103112430258
IF - 12 - 06
```

Ln 3, Col 14 | 57 caractere Plain t | 100% | Wind | UTF-8

Deskripsi:

Kode ini merupakan implementasi lengkap struktur data **Graph tak berarah (Undirected Graph)** menggunakan bahasa C++ dengan representasi **adjacency list**. Pada program utama (`main.cpp`), dibangun sebuah topologi graph yang terdiri dari **8 node (A hingga H)**, di mana Node A berperan sebagai titik awal yang bercabang ke Node B dan C, kemudian cabang-cabang tersebut terus berkembang hingga akhirnya bertemu kembali pada satu node tujuan, yaitu Node H. Program ini tidak hanya menampilkan struktur koneksi antar node ke layar, tetapi juga bertujuan untuk memperlihatkan perbedaan cara kerja algoritma penelusuran graph. Hal tersebut ditunjukkan melalui penggunaan **Depth First Search (DFS)** yang menelusuri graph secara mendalam hingga satu cabang selesai sebelum berpindah ke cabang lain, serta **Breadth First Search (BFS)** yang menelusuri graph secara melebar berdasarkan level. Kedua algoritma tersebut dijalankan secara berurutan dengan titik awal penelusuran yang sama, yaitu Node A, sehingga perbedaan urutan kunjungan node dapat diamati dengan jelas.

D. Kesimpulan

Berdasarkan implementasi program yang telah dilakukan, dapat disimpulkan bahwa penggunaan representasi graf **adjacency list** dengan pointer memberikan fleksibilitas tinggi dalam membangun dan memodifikasi struktur graf secara dinamis, terutama pada topologi jaringan yang kompleks dengan banyak percabangan dan titik temu. Pengujian terhadap graf menunjukkan perbedaan yang jelas antara algoritma **Depth First Search (DFS)** dan **Breadth First Search (BFS)**, di mana DFS menelusuri graf dengan memprioritaskan kedalaman suatu jalur hingga selesai sebelum berpindah ke jalur lain, sedangkan BFS menelusuri graf dengan memprioritaskan keluasan dengan mengunjungi seluruh node pada level yang sama terlebih dahulu. Hasil ini menegaskan bahwa pemilihan algoritma traversal harus disesuaikan dengan kebutuhan aplikasi, di mana DFS lebih cocok

untuk pencarian solusi berbasis jalur, sedangkan BFS lebih efektif untuk menemukan node terdekat atau jalur terpendek pada graf tak berbobot.

E. Referensi

Rosa, A. S., & Shalahuddin, M. (2018). *Rekayasa perangkat lunak terstruktur dan berorientasi objek*. Bandung: Informatika.

Munir, R. (2018). *Matematika diskrit* (Edisi ke-6). Bandung: Informatika.

Kurniawan, D. (2016). *Struktur data menggunakan C++*. Jakarta: Elex Media Komputindo.

Kadir, A. (2012). *Algoritma dan struktur data*. Yogyakarta: Andi Offset.

Goponenko, A., & Carroll, S. (2019). A C++ implementation of a lock-free priority queue based on Multi-Dimensional Linked List. *Link: https://www.researchgate.net/publication/337020321_A_C_Implementation_of_a_Lock-Free_Priority_Queue_Based_on_Multi-Dimensional_Linked_List.*

net/publication/337020321_A_C_Implementation_of_a_Lock-Free_Priority_Queue_Based_on_Multi-Dimensional_Linked_List.

Malik, D. S. (2010). *Data structures using C++*. USA.