

**LAPORAN PRAKTIKUM STRUKTUR  
DATA DAN ALGORITMA**

**MODUL 9  
“GRAPH DAN TREE”**



**DISUSUN OLEH :  
BINTANG YUDHISTIRA  
2311102052**

**DOSEN  
WAHYU ANDI SAPUTRA, S.PD., M.PD.**

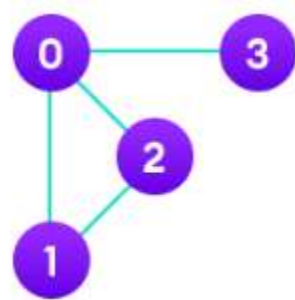
**PROGRAM STUDI S1 TEKNIK INFORMATIKA  
FAKULTAS INFORMATIKA  
INSTITUT TEKNOLOGI TELKOM PURWOKERTO  
2024**

## A. DASAR TEORI

### 1. Pengertian Graph

Graph adalah jenis struktur data umum yang susunan datanya tidak berdekatan satu sama lain (non-linier). Graph terdiri dari kumpulan simpul berhingga untuk menyimpan data dan antara dua buah simpul terdapat hubungan saling keterkaitan. Simpul pada graph disebut dengan verteks (V), sedangkan sisi yang menghubungkan antar verteks disebut edge (E). Pasangan (x,y) disebut sebagai edge, yang menyatakan bahwa simpul x terhubung ke simpul y.

Sebagai contoh, terdapat graph seperti berikut:



Graph di atas terdiri atas 4 buah verteks dan 4 pasang sisi atau edge. Dengan verteks disimbolkan sebagai V, edge dilambangkan E, dan graph disimbolkan G, ilustrasi di atas dapat ditulis dalam notasi berikut:

$$V = \{0, 1, 2, 3\}$$

$$E = \{(0,1), (0,2), (0,3), (1,2)\}$$

$$G = \{V, E\}$$

Graph banyak dimanfaatkan untuk menyelesaikan masalah dalam kehidupan nyata, dimana masalah tersebut perlu direpresentasikan atau diimajinasikan seperti sebuah jaringan. Contohnya adalah jejaring sosial (seperti Facebook, Instagram, LinkedIn, dkk).

Graph dapat dibedakan berdasarkan arah jelajahnya dan ada tidaknya label bobot pada relasinya. Berdasarkan arah jelajahnya graph dibagi menjadi Undirected graph dan Directed graph.

### **Undirected Graph**

Pada undirected graph, simpul-simpulnya terhubung dengan edge yang sifatnya dua arah. Misalnya kita punya simpul 1 dan 2 yang saling terhubung, kita bisa menjelajah dari simpul 1 ke simpul 2, begitu juga sebaliknya.

### **Directed Graph**

Kebalikan dari undirected graph, pada graph jenis ini simpul-simpulnya terhubung oleh edge yang hanya bisa melakukan jelajah satu arah pada simpul yang ditunjuk. Sebagai contoh jika ada simpul A yang terhubung ke simpul B, namun arah panahnya menuju simpul B, maka kita hanya bisa melakukan jelajah (traversing) dari simpul A ke simpul B, dan tidak berlaku sebaliknya.

Selain arah jelajahnya, graph dapat dibagi menjadi 2 berdasarkan ada tidaknya label bobot pada koneksinya, yaitu weighted graph dan unweighted graph.

## **Weighted Graph**

Weighted graph adalah jenis graph yang cabangnya diberi label bobot berupa bilangan numerik. Pemberian label bobot pada edge biasanya digunakan untuk memudahkan algoritma dalam menyelesaikan masalah. Contoh implementasinya misalkan kita ingin menyelesaikan masalah dalam mencari rute terpendek dari lokasi A ke lokasi D, namun kita juga dituntut untuk mempertimbangkan kepadatan lalu lintas, panjang jalan dll. Untuk masalah seperti ini, kita bisa mengasosiasikan sebuah edge  $e$  dengan bobot  $w(e)$  berupa bilangan ril.

## **Unweighted Graph**

Berbeda dengan jenis sebelumnya, unweighted graph tidak memiliki properti bobot pada koneksinya. Graph ini hanya mempertimbangkan apakah dua node saling terhubung atau tidak.

## **2. Pengertian Tree**

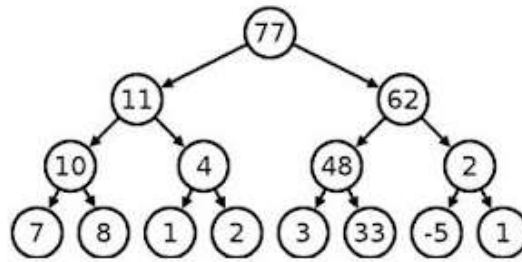
Tree adalah tipe struktur data yang sifatnya non-linier dan berbentuk hierarki.

Mengapa tree disebut sebagai struktur data non-linier? Alasannya karena data pada tree tidak disimpan secara berurutan. Sebaliknya, data diatur pada beberapa level yang disebut struktur hierarkis. Karena itu, tree dianggap sebagai struktur data non-linear.

Hierarki pada struktur tree dapat diibaratkan seperti sebuah pohon keluarga di mana terdapat hubungan antara orang tua dan anak. Titik yang lebih atas disebut simpul induk sedangkan simpul di bawahnya adalah simpul anak.

Tree atau pohon merupakan struktur data yang tidak linear yang digunakan untuk mempresentasikan data yang bersifat hirarki antara elemen-elemennya. Definisi tree yaitu kumpulan elemen yang salah satu elemennya disebut root (akar) dan elemen yang lain disebut simpul (node) yang terpecah menjadi sejumlah kumpulan yang tidak saling berhubungan satu sama lain yang disebut sub-tree atau cabang.

## Ilustrasi dari Tree



Struktur data tree dapat diklasifikasikan ke dalam 4 jenis, yaitu: General tree, Binary tree, Balanced tree, dan Binary search tree.

### 1. General tree

Struktur data tree yang tidak memiliki batasan jumlah node pada hierarki tree disebut General tree. Setiap simpul atau node bebas memiliki berapapun child node. Tree jenis adalah superset dari semua jenis tree.

### 2. Binary tree

Binary tree adalah jenis tree yang simpulnya hanya dapat memiliki paling banyak 2 simpul anak (child node). Kedua simpul tersebut biasa disebut simpul kiri (left node) dan simpul kanan (right node). Tree tipe ini lebih populer daripada jenis lainnya.

### 3. Balanced tree

Apabila tinggi dari subtree sebelah kiri dan subtree sebelah kanan sama atau kalaupun berbeda hanya berbeda 1, maka disebut sebagai balanced tree.

### 4. Binary search tree

Sesuai dengan namanya, Binary search tree digunakan untuk berbagai algoritma pencarian dan pengurutan. Contohnya seperti AVL tree dan Red-black tree. Struktur data tree jenis ini memiliki nilai pada simpul sebelah kiri lebih kecil daripada induknya. Sedangkan nilai simpul sebelah kanan lebih besar dari induknya.

## B. Guided

Guided 1.

Source Code:

```
#include <iostream>
#include <iomanip>
using namespace std;
string simpul[7] = {"Ciamis",
                   "Bandung",
                   "Bekasi",
                   "Tasikmalaya",
                   "Cianjur",
                   "Purwokerto",
                   "Yogyakarta"};

int busur[7][7] =
{
    {0, 7, 8, 0, 0, 0, 0},
    {0, 0, 5, 0, 0, 15, 0},
    {0, 6, 0, 0, 5, 0, 0},
    {0, 5, 0, 0, 2, 4, 0},
    {23, 0, 0, 10, 0, 0, 8},
    {0, 0, 0, 0, 7, 0, 3},
    {0, 0, 0, 0, 9, 4, 0}};

void tampilGraph()
{
    for (int baris = 0; baris < 7; baris++)
    {
        cout << " " << setiosflags(ios::left) << setw(15) <<
simpul[baris] << " : ";
        for (int kolom = 0; kolom < 7; kolom++)
        {
            if (busur[baris][kolom] != 0)
            {
                cout << " " << simpul[kolom] << "(" <<
busur[baris][kolom] << ")";
            }
        }
        cout << endl;
    }
}

int main()
{
    tampilGraph();
    return 0;
}
```



Screenshot Output:

```
PS C:\Users\Lenovo\Documents\Praktikum Struktur Data dan Algoritma Semester 2\Prak 2 Graph and Tree> .\Guided1.exe
Ciamis      : Bandung(7) Bekasi(8)
Bandung     : Bekasi(5) Purwokerto(15)
Bekasi      : Bandung(6) Cianjur(5)
Tasikmalaya : Bandung(5) Cianjur(2) Purwokerto(4)
Cianjur     : Ciamis(23) Tasikmalaya(10) Yogyakarta(8)
Purwokerto  : Cianjur(7) Yogyakarta(3)
Yogyakarta  : Cianjur(9) Purwokerto(4)
```

Deskripsi Program:

Program diatas adalah program yang dibuat untuk menghitung jarak antar kota. Program ini menggunakan dua array: simpul, yang menyimpan nama-nama simpul atau node (kota), dan busur, yang merupakan matriks 7x7 yang menyimpan bobot atau jarak antar simpul. Jika ada nilai selain nol dalam matriks busur, ini menunjukkan adanya hubungan atau busur (edge) antara simpul pada baris dengan simpul pada kolom tertentu dengan bobot tertentu. Fungsi tampilGraph digunakan untuk menampilkan graf ini ke layar. Fungsi ini bekerja dengan mengiterasi setiap baris dalam matriks busur, lalu mencetak nama simpul yang sesuai dengan baris tersebut, diikuti oleh semua simpul yang terhubung dengan simpul tersebut bersama dengan bobot busur yang menghubungkannya. Setiap koneksi dicetak dalam format simpul(kolom)(bobot). Fungsi main hanya memanggil fungsi tampilGraph dan kemudian mengakhiri program. Dengan demikian, program ini membantu memvisualisasikan struktur dan bobot dari graf yang sudah didefinisikan, di mana setiap simpul mewakili sebuah kota dan setiap busur mewakili jarak atau hubungan antar kota.

Guided 2.

Source Code :

```
#include <iostream>
using namespace std;
/// PROGRAM BINARY TREE
// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent;
};
Pohon *root, *baru;
// Inisialisasi
void init()
{
    root = NULL;
}
// Cek Node
int isEmpty()
{
    if (root == NULL)
        return 1; // true
    else
        return 0; // false
}
// Buat Node Baru
void buatNode(char data)
{
    if (isEmpty() == 1)
    {
        root = new Pohon();
        root->data = data;
        root->left = NULL;
        root->right = NULL;
        root->parent = NULL;
        cout << "\n Node " << data << " berhasil dibuat menjadi
root."
        << endl;
    }
    else
    {
        cout << "\n Pohon sudah dibuat" << endl;
    }
}
// Tambah Kiri
Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
}
```



```

        return NULL;
    }
    else
    {
        // cek apakah child kiri ada atau tidak
        if (node->left != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child
kiri!"
                << endl;
            return NULL;
        }
        else
        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->left = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke
child kiri "
                << baru->parent->data << endl;
            return baru;
        }
    }
}

// Tambah Kanan
Pohon *insertRight(char data, Pohon *node)
{
    if (root == NULL)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        // cek apakah child kanan ada atau tidak
        if (node->right != NULL)
        {
            // kalau ada
            cout << "\n Node " << node->data << " sudah ada child
kanan!"
                << endl;
            return NULL;
        }
        else

```

```

        {
            // kalau tidak ada
            baru = new Pohon();
            baru->data = data;
            baru->left = NULL;
            baru->right = NULL;
            baru->parent = node;
            node->right = baru;
            cout << "\n Node " << data << " berhasil ditambahkan ke
child kanan" << baru->parent->data << endl;
            return baru;
        }
    }
}

// Ubah Data Tree
void update(char data, Pohon *node)
{
    if (isEmpty() == 1)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ingin diganti tidak ada!!" <<
endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\n Node " << temp << " berhasil diubah menjadi
" << data << endl;
        }
    }
}

// Lihat Isi Data Tree
void retrieve(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data node : " << node->data << endl;
        }
    }
}

```

```

    }
}
}
// Cari Data Tree
void find(Pohon *node)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\n Node yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\n Data Node : " << node->data << endl;
            cout << " Root : " << root->data << endl;
            if (!node->parent)
                cout << " Parent : (tidak punya parent)" << endl;
            else
                cout << " Parent : " << node->parent->data << endl;
            if (node->parent != NULL && node->parent->left != node
&&
                node->parent->right == node)
                cout << " Sibling : " << node->parent->left->data <<
endl;
            else if (node->parent != NULL && node->parent->right !=
node &&
                node->parent->left == node)
                cout << " Sibling : " << node->parent->right->data
<< endl;
            else
                cout << " Sibling : (tidak punya sibling)" << endl;
            if (!node->left)
                cout << " Child Kiri : (tidak punya Child kiri)" <<
endl;
            else
                cout << " Child Kiri : " << node->left->data <<
endl;
            if (!node->right)
                cout << " Child Kanan : (tidak punya Child kanan)"
<< endl;
            else
                cout << " Child Kanan : " << node->right->data <<
endl;
        }
    }
}

```

```

// Penelurusan (Traversal)
// preOrder
void preOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// inOrder
void inOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Pohon *node = root)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data << ", ";
        }
    }
}

// Hapus Node Tree
void deleteTree(Pohon *node)
{

```

```

        if (!root)
            cout << "\n Buat tree terlebih dahulu!" << endl;
        else
        {
            if (node != NULL)
            {
                if (node != root)
                {
                    node->parent->left = NULL;
                    node->parent->right = NULL;
                }
                deleteTree(node->left);
                deleteTree(node->right);
                if (node == root)
                {
                    delete root;
                    root = NULL;
                }
                else
                {
                    delete node;
                }
            }
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\n Node subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

// Hapus Tree
void clear()
{
    if (!root)
        cout << "\n Buat tree terlebih dahulu!!" << endl;
    else
    {
        deleteTree(root);
        cout << "\n Pohon berhasil dihapus." << endl;
    }
}
}

```

```

// Cek Size Tree
int size(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// Cek Height Level Tree
int height(Pohon *node = root)
{
    if (!root)
    {
        cout << "\n Buat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);
            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

```

```

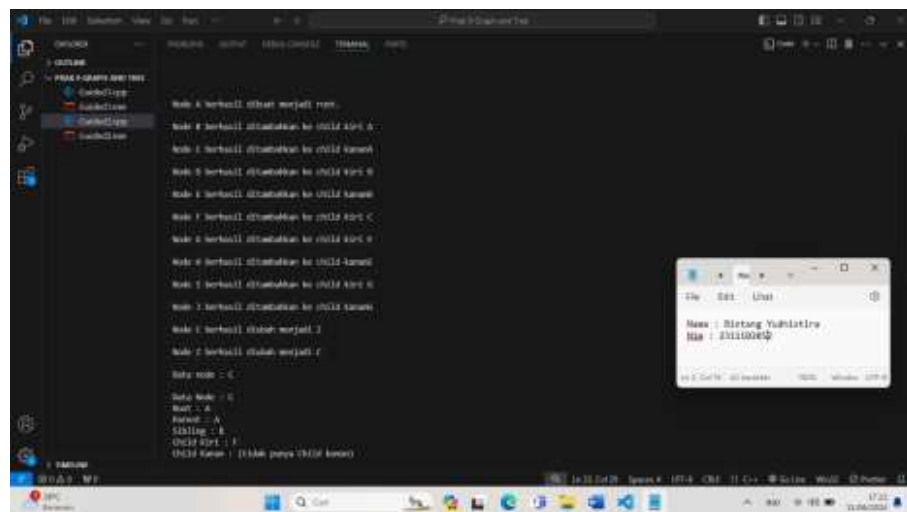
// Karakteristik Tree
void charateristic()
{
    cout << "\n Size Tree : " << size() << endl;
    cout << " Height Tree : " << height() << endl;
    cout << " Average Node of Tree : " << size() / height() << endl;
}
int main()
{
    buatNode('A');
    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
        *nodeI, *nodeJ;
    nodeB = insertLeft('B', root);
    nodeC = insertRight('C', root);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);
    update('Z', nodeC);
    update('C', nodeC);
    retrieve(nodeC);
    find(nodeC);
    cout << "\n PreOrder :" << endl;
    preOrder(root);
    cout << "\n"
        << endl;
    cout << " InOrder :" << endl;
    inOrder(root);
    cout << "\n"
        << endl;
    cout << " PostOrder :" << endl;
    postOrder(root);
    cout << "\n"
        << endl;
    charateristic();
    deleteSub(nodeE);
    cout << "\n PreOrder :" << endl;
    preOrder();
    cout << "\n"
        << endl;
    charateristic();
}

```

The screenshot shows a Windows 10 desktop with a Visual Studio Code editor open. The editor displays a C++ program that reads a character 'c' and prints its ASCII value. The program uses a switch statement to print the character's name if it is a letter. The terminal window shows the output of the program, which is '116' for the input 'c'.

```

File Edit Solution View Help
Project Explorer Search Explorer Run and Debug
c++
main.cpp
1 // Program 3: Character Type
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     char c;
8     cout << "Masukkan karakter: ";
9     cin >> c;
10
11     switch (c)
12     {
13         case 'a':
14             cout << "Karakter adalah huruf a";
15             break;
16         case 'b':
17             cout << "Karakter adalah huruf b";
18             break;
19         case 'c':
20             cout << "Karakter adalah huruf c";
21             break;
22         case 'd':
23             cout << "Karakter adalah huruf d";
24             break;
25         case 'e':
26             cout << "Karakter adalah huruf e";
27             break;
28         case 'f':
29             cout << "Karakter adalah huruf f";
30             break;
31         case 'g':
32             cout << "Karakter adalah huruf g";
33             break;
34         case 'h':
35             cout << "Karakter adalah huruf h";
36             break;
37         case 'i':
38             cout << "Karakter adalah huruf i";
39             break;
40         case 'j':
41             cout << "Karakter adalah huruf j";
42             break;
43         case 'k':
44             cout << "Karakter adalah huruf k";
45             break;
46         case 'l':
47             cout << "Karakter adalah huruf l";
48             break;
49         case 'm':
50             cout << "Karakter adalah huruf m";
51             break;
52         case 'n':
53             cout << "Karakter adalah huruf n";
54             break;
55         case 'o':
56             cout << "Karakter adalah huruf o";
57             break;
58         case 'p':
59             cout << "Karakter adalah huruf p";
60             break;
61         case 'q':
62             cout << "Karakter adalah huruf q";
63             break;
64         case 'r':
65             cout << "Karakter adalah huruf r";
66             break;
67         case 's':
68             cout << "Karakter adalah huruf s";
69             break;
70         case 't':
71             cout << "Karakter adalah huruf t";
72             break;
73         case 'u':
74             cout << "Karakter adalah huruf u";
75             break;
76         case 'v':
77             cout << "Karakter adalah huruf v";
78             break;
79         case 'w':
80             cout << "Karakter adalah huruf w";
81             break;
82         case 'x':
83             cout << "Karakter adalah huruf x";
84             break;
85         case 'y':
86             cout << "Karakter adalah huruf y";
87             break;
88         case 'z':
89             cout << "Karakter adalah huruf z";
90             break;
91         default:
92             cout << "Karakter bukan huruf";
93     }
94
95     return 0;
96 }
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
8
```



Program ini mendefinisikan struktur Pohon yang merepresentasikan node dalam pohon biner, dengan atribut data, pointer ke anak kiri (left), anak kanan (right), dan induk (parent). Fungsi init menginisialisasi pohon, sedangkan isEmpty memeriksa apakah pohon kosong. Fungsi buatNode membuat node baru sebagai root, dan fungsi insertLeft serta insertRight menambahkan node baru sebagai anak kiri atau kanan dari node tertentu. Fungsi update mengubah data node, sementara retrieve



dan find menampilkan data dan informasi lengkap tentang node. Program ini juga menyediakan penelusuran pohon dalam tiga cara: pre-order, in-order, dan post-order. Fungsi deleteTree, deleteSub, dan clear digunakan untuk menghapus node, subtree, atau seluruh pohon. Fungsi size dan height menghitung ukuran dan tinggi pohon. Fungsi characteristic menampilkan karakteristik pohon seperti ukuran, tinggi, dan rata-rata node. Pada fungsi main, berbagai operasi dilakukan untuk membangun dan memanipulasi pohon biner, termasuk penambahan node, perubahan data, penelusuran, dan penghapusan subtree.

### C. UNGUIDED

#### Unguided 1:

1. Buatlah program graph dengan menggunakan inputan user untuk menghitung jarak dari sebuah kota ke kota lainnya.

#### Source Code:

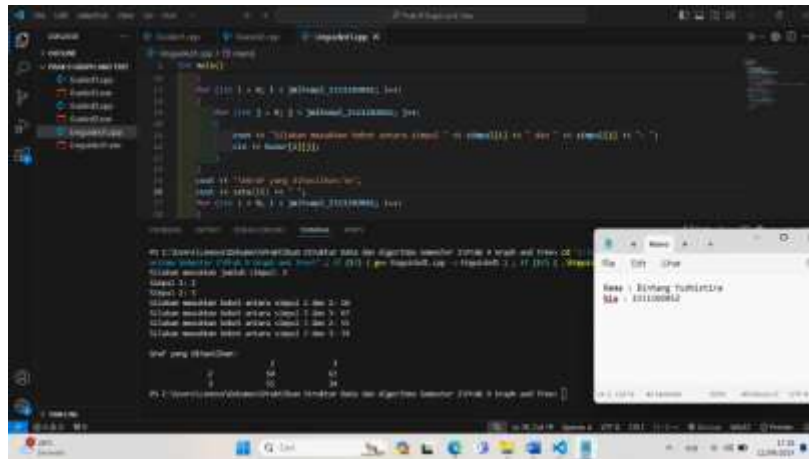
```
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;
```

```

int main()
{
    int jml_2311102052;
    cout << "Silakan masukkan jumlah simpul: ";
    cin >> jml_2311102052;
    string simpul[jml_2311102052];
    int busur[jml_2311102052][jml_2311102052];
    for (int i = 0; i < jml_2311102052; i++)
    {
        cout << "Simpul " << i + 1 << ": ";
        cin >> simpul[i];
    }
    for (int i = 0; i < jml_2311102052; i++)
    {
        for (int j = 0; j < jml_2311102052; j++)
        {
            cout << "Silakan masukkan bobot antara simpul " <<
simpul[i] << " dan " << simpul[j] << ": ";
            cin >> busur[i][j];
        }
    }
    cout << "\nGraf yang dihasilkan:\n";
    cout << setw(15) << " ";
    for (int i = 0; i < jml_2311102052; i++)
    {
        cout << setw(15) << simpul[i];
    }
    cout << endl;
    for (int i = 0; i < jml_2311102052; i++)
    {
        cout << setw(15) << simpul[i];
        for (int j = 0; j < jml_2311102052; j++)
        {
            cout << setw(15) << busur[i][j];
        }
        cout << endl;
    }
    return 0;
}

```

**Screenshot Output:**



### Deskripsi Program:

Pertama, program meminta pengguna untuk memasukkan jumlah simpul dalam graf. Setelah itu, pengguna diminta untuk memberikan nama setiap simpul yang disimpan dalam array simpul. Kemudian, program meminta pengguna untuk memasukkan bobot antara setiap pasangan simpul, yang disimpan dalam matriks busur. Setelah semua data dimasukkan, program menampilkan matriks bobot graf dalam format tabel yang rapi menggunakan fungsi setw dari pustaka iomanip untuk penyalarsan kolom. Outputnya mencakup baris dan kolom yang diberi label dengan nama simpul, diikuti oleh nilai bobot antar simpul, sehingga memberikan representasi visual yang jelas dari graf berbobot tersebut.

### Unguided 2:

2. Modifikasi guided tree diatas dengan program menu menggunakan input data tree dari user dan berikan fungsi tambahan untuk menampilkan node child dan descendant dari node yang diinput kan!

### Source Code:

```
#include <iostream>
using namespace std;

// Deklarasi Pohon
struct Pohon
{
    char data;
    Pohon *left, *right, *parent; // Pointer
};

// Pointer global
Pohon *root_2311102052;

// Inisialisasi
void init()
{
    root_2311102052 = NULL;
}

bool isEmpty()
{
    return root_2311102052 == NULL;
}

Pohon *newPohon(char data)
{
    Pohon *node = new Pohon();
    node->data = data;
    node->left = NULL;
    node->right = NULL;
    node->parent = NULL;
    return node;
}

void buatNode(char data)
{
    if (isEmpty())
    {
        root_2311102052 = newPohon(data);
        cout << "\nNode " << data << " berhasil dibuat menjadi
root." << endl;
    }
    else
    {
        cout << "\nPohon sudah dibuat" << endl;
    }
}
```

```

Pohon *insertLeft(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->left != NULL)
        {
            cout << "\nNode " << node->data << " sudah memiliki
child kiri!" << endl;
            return NULL;
        }
        else
        {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->left = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke
child kiri dari " << node->data << endl;
            return baru;
        }
    }
}

Pohon *insertRight(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return NULL;
    }
    else
    {
        if (node->right != NULL)
        {
            cout << "\nNode " << node->data << " sudah memiliki
child kanan!" << endl;
            return NULL;
        }
        else
        {
            Pohon *baru = newPohon(data);
            baru->parent = node;
            node->right = baru;
            cout << "\nNode " << data << " berhasil ditambahkan ke
child kanan dari " << node->data << endl;

```

```

        return baru;
    }
}

void update(char data, Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ingin diganti tidak ada!!" << endl;
        else
        {
            char temp = node->data;
            node->data = data;
            cout << "\nNode " << temp << " berhasil diubah menjadi "
<< data << endl;
        }
    }
}

void retrieve(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        else
        {
            cout << "\nData node : " << node->data << endl;
        }
    }
}

void find(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else

```

```

{
    if (!node)
        cout << "\nNode yang ditunjuk tidak ada!" << endl;
    else
    {
        cout << "\nData Node : " << node->data << endl;
        cout << "Root : " << root_2311102052->data << endl;

        if (!node->parent)
            cout << "Parent : (tidak memiliki parent)" << endl;
        else
            cout << "Parent : " << node->parent->data << endl;

        if (node->parent != NULL && node->parent->left != node
&& node->parent->right == node)
            cout << "Sibling : " << node->parent->left->data <<
endl;
        else if (node->parent != NULL && node->parent->right !=
node && node->parent->left == node)
            cout << "Sibling : " << node->parent->right->data <<
endl;
        else
            cout << "Sibling : (tidak memiliki sibling)" <<
endl;

        if (!node->left)
            cout << "Child Kiri : (tidak memiliki child kiri)"
<< endl;
        else
            cout << "Child Kiri : " << node->left->data << endl;

        if (!node->right)
            cout << "Child Kanan : (tidak memiliki child kanan)"
<< endl;
        else
            cout << "Child Kanan : " << node->right->data <<
endl;
    }
}

// Penelusuran (Traversal)
// preOrder
void preOrder(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {

```

```

        if (node != NULL)
        {
            cout << " " << node->data << ", ";
            preOrder(node->left);
            preOrder(node->right);
        }
    }
}

// inOrder
void inOrder(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            inOrder(node->left);
            cout << " " << node->data << ", ";
            inOrder(node->right);
        }
    }
}

// postOrder
void postOrder(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)
        {
            postOrder(node->left);
            postOrder(node->right);
            cout << " " << node->data << ", ";
        }
    }
}

// Hapus Node Tree
void deleteTree(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        if (node != NULL)

```



```

        {
            if (node != root_2311102052)
            {
                if (node->parent->left == node)
                    node->parent->left = NULL;
                else if (node->parent->right == node)
                    node->parent->right = NULL;
            }
            deleteTree(node->left);
            deleteTree(node->right);

            if (node == root_2311102052)
            {
                delete root_2311102052;
                root_2311102052 = NULL;
            }
            else
            {
                delete node;
            }
        }
    }
}

// Hapus SubTree
void deleteSub(Pohon *node)
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(node->left);
        deleteTree(node->right);
        cout << "\nNode subtree " << node->data << " berhasil
dihapus." << endl;
    }
}

// Hapus Tree
void clear()
{
    if (isEmpty())
        cout << "\nBuat tree terlebih dahulu!" << endl;
    else
    {
        deleteTree(root_2311102052);
        cout << "\nPohon berhasil dihapus." << endl;
    }
}
}

```

```

// Cek Size Tree
int size(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            return 1 + size(node->left) + size(node->right);
        }
    }
}

// Cek Height Level Tree
int height(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
        return 0;
    }
    else
    {
        if (!node)
        {
            return 0;
        }
        else
        {
            int heightKiri = height(node->left);
            int heightKanan = height(node->right);

            if (heightKiri >= heightKanan)
            {
                return heightKiri + 1;
            }
            else
            {
                return heightKanan + 1;
            }
        }
    }
}

```

```

    }
}

// Karakteristik Tree
void characteristic()
{
    int s = size(root_2311102052);
    int h = height(root_2311102052);
    cout << "\nSize Tree : " << s << endl;
    cout << "Height Tree : " << h << endl;
    if (h != 0)
        cout << "Average Node of Tree : " << s / h << endl;
    else
        cout << "Average Node of Tree : 0" << endl;
}

// Menampilkan Child dari Sebuah Node
void displayChild(Pohon *node)
{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        }
        else
        {
            cout << "\nChild dari node " << node->data << "
adalah:";
            if (node->left)
            {
                cout << " " << node->left->data;
            }
            if (node->right)
            {
                cout << " " << node->right->data;
            }
            cout << endl;
        }
    }
}

// Menampilkan Descendant dari Sebuah Node
void displayDescendant(Pohon *node)

```

```

{
    if (isEmpty())
    {
        cout << "\nBuat tree terlebih dahulu!" << endl;
    }
    else
    {
        if (!node)
        {
            cout << "\nNode yang ditunjuk tidak ada!" << endl;
        }
        else
        {
            cout << "\nDescendant dari node " << node->data << "
adalah:";
            // Gunakan rekursi untuk mencetak descendant
            if (node->left)
            {
                cout << " " << node->left->data;
                displayDescendant(node->left);
            }
            if (node->right)
            {
                cout << " " << node->right->data;
                displayDescendant(node->right);
            }
            cout << endl;
        }
    }
}

int main()
{
    init();
    buatNode('A');

    Pohon *nodeB, *nodeC, *nodeD, *nodeE, *nodeF, *nodeG, *nodeH,
*nodeI, *nodeJ;

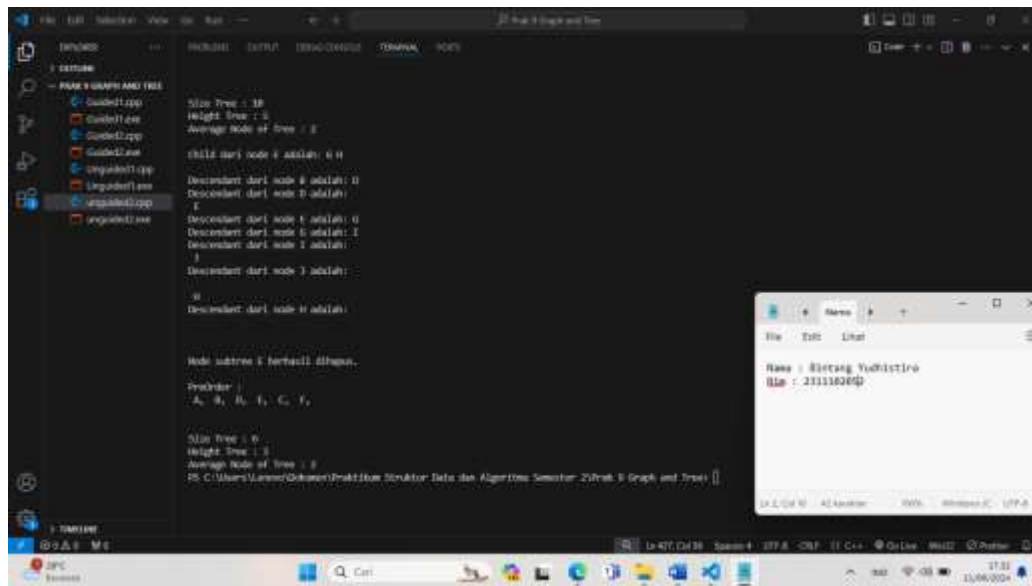
    nodeB = insertLeft('B', root_2311102052);
    nodeC = insertRight('C', root_2311102052);
    nodeD = insertLeft('D', nodeB);
    nodeE = insertRight('E', nodeB);
    nodeF = insertLeft('F', nodeC);
    nodeG = insertLeft('G', nodeE);
    nodeH = insertRight('H', nodeE);
    nodeI = insertLeft('I', nodeG);
    nodeJ = insertRight('J', nodeG);

```

```
update('Z', nodeC);
update('C', nodeC);
retrieve(nodeC);
find(nodeC);
cout << "\nPreOrder :" << endl;
preOrder(root_2311102052);
cout << "\n"
    << endl;
cout << "InOrder :" << endl;
inOrder(root_2311102052);
cout << "\n"
    << endl;
cout << "PostOrder :" << endl;
postOrder(root_2311102052);
cout << "\n"
    << endl;
characteristic();
displayChild(nodeE);
displayDescendant(nodeB);
deleteSub(nodeE);
cout << "\nPreOrder :" << endl;
preOrder(root_2311102052);
cout << "\n"
    << endl;
characteristic();
}
```

**Screenshot Output:**





### Deskripsi Program:

untuk manipulasi pohon biner, meliputi pembuatan, penambahan node kiri dan kanan, pengubahan data node, penghapusan pohon atau subtree, serta penelusuran dalam tiga cara: pre-order, in-order, dan post-order. Program dimulai dengan mendefinisikan struktur pohon dengan elemen data dan pointer ke anak kiri, kanan, serta parent. Sebuah fungsi inisialisasi mengatur root ke NULL dan fungsi isEmpty mengecek apakah pohon kosong. Untuk membuat node baru, newPohon mengalokasikan memori dan menginisialisasi node. Fungsi buatNode menetapkan node baru sebagai root jika pohon kosong, sementara insertLeft dan insertRight menambah node sebagai anak kiri atau kanan dari node tertentu jika belum ada anak. Fungsi update mengubah data node, retrieve menampilkan data node, dan find menampilkan detail node termasuk parent dan sibling. Fungsi deleteTree dan deleteSub menghapus pohon atau subtree. Fungsi size menghitung jumlah node, height menghitung tinggi pohon, dan characteristic menampilkan ukuran, tinggi, dan rata-rata node per level. Fungsi displayChild dan displayDescendant menampilkan anak dan keturunan dari node tertentu. Pada bagian main, berbagai fungsi tersebut digunakan untuk membentuk pohon biner, memperbarui, menampilkan, dan menghapus node, serta menampilkan karakteristik pohon setelah berbagai operasi.

#### **D. Kesimpulan**

Graph dan tree merupakan dua struktur data mendasar dalam C++ yang sangat penting untuk menyelesaikan masalah komputasi yang rumit. Graph sangat berguna untuk memodelkan hubungan dan koneksi antara berbagai entitas, sedangkan tree lebih sesuai untuk merepresentasikan data dengan struktur hierarkis. Pemahaman dan implementasi yang baik dari kedua struktur data ini memungkinkan pengembangan solusi yang efisien dan efektif untuk berbagai jenis masalah.

#### **E. Referensi**

[1] Asisten Pratikum “Modul 9 Graph dan Tree”, diakses dari :

Learning Management System, 2024.

[2] Trivusi. (2022, September 16). Struktur Data Graph: Pengertian, Jenis, dan Kegunaannya. Diakses pada 10 Juni 2024 dari :

[Struktur Data Graph: Pengertian, Jenis, dan Kegunaannya - Trivusi](#)

[3] Trivusi. (2022b, September 16). Struktur Data Tree: Pengertian, Jenis, dan Kegunaannya. Trivusi. Diakses pada 10 Juni 2024 dari :

[Struktur Data Tree: Pengertian, Jenis, dan Kegunaannya - Trivusi](#)