MODUL PRAKTIKUM

IF4101 – Algorithma dan Struktur Data

Graph ADT

Minggu	:	14
Tanggal	:	29 April 2019
Tujuan	:	• Mampu menjelaskan definisi formal dari graf dan propertiesnya. • Mampu menjelaskan definisi formal dari varian graf. • Mampu menjelaskan teknik representasi graf. • Mampu menghitung kompleksitas memory dan operasi pada struktur data graf. • Mampu membuat struktur data graf dengan teknik matrix dan list kedekatan.
Setoran	:	**kertas jawaban (ke meja TA)** **program (via moodle)**
Waktu penyetoran	:	1 Mei 2019 Pukul 22:00

Petunjuk Praktikum

1. Anda dapat mengerjakan praktikum ini secara berkelompok dengan maksimum 3 mahasiswa per kelompok.

Referensi

1. T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, *Introduction to Algorithm, 2nd eds*., MIT Press 2001.

2. M.A. Weiss, *Data Structures and Algorithm Analysis in C, 2nd Eds*., Addison-Wesley, 1997.

Ulasan

1. Berikanlah definisi formal dari graf, graf berarah, dan graf tidak berarah.
2. Berikan 2 contoh nyata dari graf berarah dan tidak berarah.
3. Berikan definisi formal dari *adjacent*, *path*, dan *cycle*.
4. Berikan definis formal dari DAG (*Directed Acyclic Graph*) dan beri 2 contohnya.
5. Apakah graf tak berarah merupakan graf siklus? Jelaskan.
6. Jelaskan *complete graph*, *dense graph* dan *sparse graph*.
7. Jelaskan 2 cara representasi graf. Jelaskan pula kelebihan dan kekurangannya.

Pendalaman

1. Hitunglah kompleksitas memori untuk representasi graf dengan *adjacency list* (list kedekatan) dan *adjacency matrix* (matrix kedekatan). Buktikan jawaban anda.
2. Pada bagian pemrograman diberikan representasi sebuah graf dengan list kedekatan. Representasikanlah graf tersebut dengan matrix kedekatan.

3. Pada bagian pemrograman diberikan 2 program untuk membuat `Graph ADT`, yakni dengan matrix dan list kedekatan. Hitunglah kompleksitas waktu eksekusi untuk operasi (fungsi) `AddNode` dan `Remove Node` pada program dengan matrix dan list kedekatan.

Pemrograman

1. **Implementasi graf dengan *adjacency matrix* (matrix kedekatan)**
   Pada bagian ini anda akan membuat struktur data graf dengan matrix kedekatan. Representasi ini sangat baik untuk graf padat (*dense graph*). Pada no. 2 anda akan menggunakan list kedekatan, representasi yang baik untuk graf renggang (*sparse graph*).

   α) Antarmuka: `adjmatrix.h` diberikan sebagai berikut:

adjmatrix.h | adjmatrix.c | klien_adjmatrix.c

```
1  //author Tennov
2
3  typedef unsigned int ElementType;
4
5  #ifndef _Graph_H
6  #define _Graph_H
7
8  struct GraphADT;                        //structure of graph
9  typedef struct GraphADT *PtrToGraph;    //pointer to struct graph
10 typedef PtrToGraph Graph;               //graph represented as pointer
11
12 Graph ConstructGraph(Graph graph, unsigned int V);  //construct an empty graph
13 unsigned int GetNumberOfNodes(Graph g);
14 unsigned int GetNumberOfEdges(Graph g);
15 Graph AddEdge(Graph g, ElementType Start, ElementType End, int weight);
16 Graph RemoveEdge(Graph g, unsigned int i, unsigned int j);
17 unsigned int GetNumPredecessors(Graph g, ElementType node, ElementType V);
18 int *Predecessors(Graph g, ElementType node, ElementType V);//get predecessors of node ID
19 unsigned int GetNumSuccessors(Graph g, ElementType node, ElementType V);
20 int *Successors(Graph g, ElementType node, ElementType V);  //get predecessors of node ID
21 void PrintAdjMatrix(Graph G);
22
23 #endif
```

   Pernyataan baris ke-8 memberikan deklarasi generik untuk Graph ADT sehingga graf dapat direpresentasikan sesuai dengan kebutuhan. Baca komentar agar anda mengerti setiap deklarasi variabel, definisi tipe, dan deklarasi fungsi.

   β) Definisi dari `struct GraphADT` diberikan oleh kode di bawah ini.

adjmatrix.h | adjmatrix.c | klien_adjmatrix.c

```
1  //author Tennov
2
3  #include <stdio.h>
4  #include <stdlib.h>
5  #include <limits.h>
6  #include "adjmatrix.h"
7
8  struct GraphADT{
9      unsigned int V;      //cardinality of set of nodes
10     unsigned int E;      //cardinality of set of edges
11     ElementType **adjmatrix;    //adjacency matrix as double indirection pointer
12 };
```

Pada struktur tersebut di atas, matrix kedekatan diwakili oleh matrix dimensi 2 pada baris ke-11 yang akan dialokasikan secara dinamik.

χ) Fungsi `ConstructGraph` ditunjukkan oleh kode di bawah ini. Konstruksi dimulai dengan alokasi memori untuk pointer `graph`, penugasan nilai `V`, `E` dan alokasi memori untuk matrix kedekatan dimensi 2 dengan teknik pointer tak langsung ganda (*double indirection*). Jumlah baris dan kolom sama dengan jumlah simpul graf. Baris menunjukkan simpul sumber dan kolom sebagai simpul tujuan. Terakhir, semua node di-*set* tidak dapat diakses dengan menugaskan nilai untuk setiap *cell* pada matriks dimensi 2 sebesar bilangan maksimum dari `unsigned int` dikurang 1000. Untuk itu, anda perlu menyertakan **`<limits.h>`** pada fail implementasi.

```
14 Graph ConstructGraph(Graph graph, unsigned int V){
15     int i,j;
16     graph = malloc(sizeof(struct GraphADT));
17     graph->V = V;
18     //memory of adjacency matrix is allocated
19     graph->adjmatrix = (ElementType **)malloc(V * sizeof(ElementType *));
20     if(graph->adjmatrix == NULL){
21         printf("Memory is full");
22         exit(1);
23     }
24     for(i = 0; i < V; i++){
25         graph->adjmatrix[i] = (ElementType *) malloc(V * sizeof(ElementType));
26         if(graph->adjmatrix[i] == NULL){
27             printf("Memory is full");
28             exit(1);
29         }
30     }
31     //initialise all nodes unreachable
32     for(i = 0; i < V; i++)
33         for(j = 0; j < V; j++){
34             if(i == j)
35                 graph->adjmatrix[i][j]= 0;
36             else
37                 graph->adjmatrix[i][j] = UINT_MAX - 1000;
38         }
39
40     return graph;
41 }
```

δ) Fungsi untuk mengembalikan jumlah simpul dan busur merupakan fungsi sederhana dan ditunjukkan oleh kode berikut:

```
43 unsigned int GetNumberOfNodes(Graph g){
44     return g->V;
45 }
46
47 unsigned int GetNumberOfEdges(Graph g){
48     return g->E;
49
50 }
51
```

ε) Operasi untuk menambah busur dilakukan dengan menugaskan nilai *cell* dari matrix dimensi 2 dengan bobot dimana baris menunjukkan simpul sumber dan kolom sebagai simpul tujuan. Sebaliknya, operasi menghapus busur dilakukan dengan

menugaskan nilai *cell* sebesar bilangan maksimum dari `unsigned int` dikurang 1000 sebagai penanda bahwa tidak ada busur antara simpul yang direpresentasikan oleh baris ke simpul yang ditunjukkan oleh kolom. Fungsi menambah dan menghapus busur diberikan oleh kode berikut:

```
52 Graph AddEdge(Graph g, unsigned int i, unsigned int j, int weight){
53     if(g != NULL){
54         g->adjmatrix[i][j] = weight;
55         g->E++;
56     }
57     return g;
58 }
59
60 Graph RemoveEdge(Graph g, unsigned int i, unsigned int j){
61     if(g != NULL){
62         g->adjmatrix[i][j] = UINT_MAX - 1000; //max cost of a path = 1000
63         g->E--;
64     }
65     return g;
66 }
67
```

ϕ) Kode di bawah ini menunjukkan cara untuk mengambil semua *predecessors* (pendahulu) dari suatu simpul. Langkah pertamanya adalah mengambil jumlah pendahulu yang diwakili oleh fungsi `GetNumPredecessors`. Fungsi ini dipanggil dari fungsi `Predecessors` agar nilai `counter` dapat dihitung. Setelah itu, fungsi `Predecessors` akan mencari semua pendahulu dari simpul, menambahkannya ke array `predecessors` yang dialokasikan secara dinamik dengan jumlah elemen sama dengan nilai `counter`, dan mengembalikan pointer ke program klien.

```
68 unsigned int GetNumPredecessors(Graph g, ElementType node, ElementType V){
69     unsigned int i, counter=0;
70
71     for(i = 0; i < V; i++){
72         if((g->adjmatrix[i][node] < UINT_MAX - 1000) && (i != node))
73             counter ++;
74     }
75     return counter;
76 }
77
78 int *Predecessors(Graph g, ElementType node, ElementType V){
79     int i=0, j=0, *predecessors;
80     unsigned int counter = GetNumPredecessors(g, node, V);
81     //allocate array of which elements = counter
82     predecessors = malloc(counter * sizeof(unsigned int));
83     for(i = 0; i < V; i++){
84         if((g->adjmatrix[i][node] < UINT_MAX - 1000) && (i != node)){
85             predecessors[j] = i;
86             j++;
87         }
88     }
89
90     return predecessors;
91 }
```

γ) Untuk menampilkan matrix kedekatan ke layar, diimplementasi fungsi `PrintAdjMatrix` seperti ditunjukkan oleh kode di bawah ini:

```
94 void PrintAdjMatrix(Graph g){
95     int i,j;
96     unsigned int V = GetNumberOfNodes(g);
97 
98     printf("\t");
99     for(i = 0;i < V; i++)
00         printf("   V[%d] \t", i);
01     printf("\n");
02     for(i = 0; i < V; i++){
03         printf("V[%d]\t", i);
04         for(j=0; j < V; j++){
05             printf("%.10u \t", g->adjmatrix[i][j]);
06         }
07         printf("\n");
08     }
09 }
```

η) Untuk mencoba GraphADT, buatlah program klien sebagai berikut:

```
1  //author Tennov
2  
3  #include <stdio.h>
4  #include "adjmatrix.h"
5  
6  int main(int argc, char * argv[]){
7      int *predecessors, i, counter;
8      Graph G = NULL;
9  
10     G = ConstructGraph(G, 5);              //construct graph with 7 nodes
11     int V = GetNumberOfNodes(G);           //get number of nodes
12     printf("Number of Nodes in Graph: %u\n\n", V);
13 
14     //graph with unequal weights
15     G = AddEdge(G,0,1,3);
16     G = AddEdge(G,0,2,5);
17     G = AddEdge(G,1,2,2);
18     G = AddEdge(G,1,3,6);
19     G = AddEdge(G,2,3,1);
20     G = AddEdge(G,3,4,4);
21     G = AddEdge(G,4,2,6);
22 
23     G = RemoveEdge(G,1,3);
24     PrintAdjMatrix(G);
25 
26     //get predecessors of node 2
27     counter = GetNumPredecessors(G, 2, V);
28     printf("\nnumber of predecessors of node 2 = %u\n", counter);
29     predecessors = Predecessors(G,2, V);
30     printf("predecessors of node 2\n");
31     for(i=0;i<counter;i++)
32         printf("%d predecessors of  node 2  = %d\n", i+1, predecessors[i]);
33 
34     return 0;
35 }
```

ι) Kompilasi dan jalankanlah program.

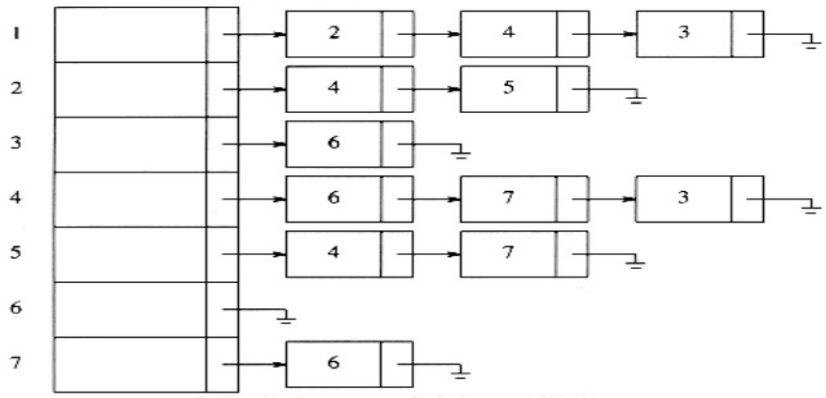
```
tennov@siftworkstation:~/c_alg_sd_S1/mg14_sesi3_graph/adjmatrix$ gcc -Wall klien_adjmatrix.c adjmatrix.c
tennov@siftworkstation:~/c_alg_sd_S1/mg14_sesi3_graph/adjmatrix$ ./a.out
Number of Nodes in Graph: 5

          V[0]           V[1]           V[2]           V[3]           V[4]
V[0]    0000000000     0000000003     0000000005     4294966295     4294966295
V[1]    4294966295     0000000000     0000000002     4294966295     4294966295
V[2]    4294966295     4294966295     0000000000     0000000001     4294966295
V[3]    4294966295     4294966295     4294966295     0000000000     0000000004
V[4]    4294966295     4294966295     0000000006     4294966295     0000000000

number of predecessors of node 2 = 3
predecessors of node 2
1 predecessors of  node 2  = 0
2 predecessors of  node 2  = 1
3 predecessors of  node 2  = 4
```

2. Implementasi graf dengan *adjacency list* (list kedekatan)

Graf yang digunakan untuk tugas ke-2 ini ditunjukkan oleh gambar di bawah ini. Graf tersebut diambil dari referensi [2] halaman 286.



α) Antarmuka: `graph.h` diberikan pada halaman berikut. Baris ke-5 s/d 24 merupakan pra pemrosesan makro yang berisi pendefinisian kondisional. Baris ke-8 mendeklarasikan struktur simpul secara generik sehingga pembuat struktur data secara bebas dapat mengimplementasikan anggota variabel sesuai dengan kebutuhan di fail implementasi. Dengan cara yang sama, baris ke-12 mendeklarasikan struktur graf. Pada baris ke-10, 11, dan 14 diberikan pendefinisian tipe data untuk **`Edge`**, **`Node`**, dan **`Graph`**. Pada baris ke-16 s/d 22 dideklarasikan beberapa operasi pada struktur data.

graph.h × graph.c × klien_graph.c ×

```
1 //author Tennov
2
3 typedef unsigned int ElementType;
4
5 #ifndef _Graph_H
6 #define _Graph_H
7
8 struct GraphNode;                          //structure of node
9 typedef struct GraphNode *PtrToGraphNode; //pointer to struct node
10 typedef PtrToGraphNode Edge;             //ptr to struct node used to represent edge
11 typedef PtrToGraphNode Node;             //ptr to struct node used to represent node
12 struct GraphADT;                         //structure of graph
13 typedef struct GraphADT *PtrToGraph;     //pointer to struct graph
14 typedef PtrToGraph Graph;                //graph represented as pointer
15
16 Graph ConstructGraph(unsigned int V);  //construct an empty graph
17 unsigned int GetNumberOfNodes(Graph g);
18 Graph AddNode(Graph g, ElementType X);
19 Node SearchNode(Graph g, ElementType X);
20 ElementType GetNodeID(Node node);
21 Graph AddEdge(Graph g, ElementType Start, ElementType End, int weight);
22 int *GetNeighbours(Graph g, ElementType ID); //get neighbours (successors) of node ID
23
24 #endif
```

β) Untuk implementasi, pertama sekali anda harus mendefinisikan struktur `struct GraphNode` yang merupakan representasi simpul pada graf. Edit kode berikut dan simpan sebagai `graph.c`.

```
7 struct GraphNode{
8   ElementType ID; //node identifier
9   Edge next;      //Edge as pointer ke next struct GraphNode
10  int weight;     //weight of edge
11 };
```

χ) Selanjutnya, anda akan mendefinisikan `struct GraphADT` yang merupakan representasi dari graf. Tambahkan kode berikut ke fail implementasi.

```
13 struct GraphADT{
14   unsigned int V;   //cardinality of set of nodes
15   Node *adjlist;    //adjacency list as dynamic array of pointers
16 };
```

Ada 2 anggota variable struktur, yakni **V** yang merupakan jumlah simpul dan pointer `adjlist` (list kedekatan) yang akan dideklarasikan sebagai arrray dinamik yang menampung pointer ke struktur `GraphNode`. Dengan demikian `adjlist` merupakan array pointer (***array of pointers***).

δ) Konstruksi graf diberikan oleh fungsi `ConstructGraph` di halaman selanjutnya. Pada baris ke-23 alokasi memory untuk menampung semua simpul pada list kedekatan dilakukan.

```
18 Graph ConstructGraph(unsigned int V){
19   Graph graph = malloc(sizeof(struct GraphADT));
20   graph->V = V;
21   /*memory of adjacency list is allocated
22     each block size is equal to the size of PtrToGraphNode*/
23   graph->adjlist = malloc(V * sizeof(PtrToGraphNode));
24
25   return graph;
26 }
27
28 unsigned int GetNumberOfNodes(Graph g){
29   return g->V;
30 }
```

ε) Operasi untuk menambah simpul diterapkan oleh fungsi `AddNode` (lihat antarmuka untuk deklarasi fungsi tersebut) di bawah ini.

```
32 Graph AddNode(Graph g, ElementType X){
33   static int count;
34   if(count < g->V){
35     g->adjlist[count] = malloc(sizeof(struct GraphNode));
36     g->adjlist[count]->ID= X;
37     g->adjlist[count]->next = NULL;
38     count ++;
39     //printf("Count = %d\n", count);
40   }else{
41     printf("New Node cannot be added");
42     exit(1); //force exit
43   }
44
45   return g;
46 }
47
```

Pada baris ke-34 dilakukan pengecekan batas (*bound checking*) untuk memastikan jumlah simpul yang hendak dimasukkan tidak melebihi kapasitas list kedekatan. Jika list kedekatan telah penuh, program keluar. Baris ke-35 merupakan alokasi memori untuk simpul yang diikuti oleh penugasan `ID` dan `next`.

ϕ) Fungsi untuk mencari simpul diberikan sebagai berikut. Fungsi tersebut mengembalikan pointer ke simpul (`Node`).

```
48 Node SearchNode(Graph g, ElementType X){
49   int i =0;
50   unsigned int limit = GetNumberOfNodes(g);
51   for(i = 0; i < limit ; ++i){
52     if(g->adjlist[i]->ID == X)
53       return g->adjlist[i];
54   }
55 }
56
57 ElementType GetNodeID(Node node){
58   return node->ID;
59 }
60
```

Pada baris ke-57 s/d 59 didefinisikan fungsi `GetNodeID` yang mengembalikan pengenal dari simpul.

γ) Operasi untuk menambahkan busur (*edge*) diberikan oleh potongan kode di bawah ini. *Edge* dianggap sebagai pointer dari suatu simpul ke simpul lainnya yang diuntai secara berurutan. Langkah pertama pada fungsi ini adalah mencari simpul `Start` (sumber) pada list kedekatan. Selanjutnya menciptakan simpul baru dan menugaskan `ID`, bobot (`weight` atau *cost*), dan `next`. Dan membuat pointer dari simpul terakhir ke simpul yang baru saja dibuat.

```
61 Graph AddEdge(Graph g, ElementType Start, ElementType End, int weight){
62   Node node = SearchNode(g, Start);
63   //printf("Node %d is found, id = %d\n", Start, node->ID);
64   if(node != NULL){
65     while(node->next != NULL)
66       node = node->next;
67     Edge new = malloc(sizeof(struct GraphNode));
68     new->ID = End;
69     new->weight = weight;
70     new->next = NULL;
71     node->next = new;
72   }
73   return g;
74 }
```

η) Selanjutnya, diberikan fungsi yang mengembalikan tetangga (*neighbours*) dari suatu simpul. Tetangga pada kasus ini adalah semua simpul yang dapat diakses dari simpul tertentu atau sekumpulan busur (*edges*) yang dimulai (bersumber) dari simpul tertentu ke sejumlah simpul lainnya. Untuk keperluan ini, digunakan fungsi yang mengembalikan pointer.

```
76 int *GetNeighbours(Graph g, ElementType ID){
77   int i=0;
78   //max neighbours is 3, change this as required
79   int *neighbours = malloc(3 * sizeof(int));
80
81   Node node = SearchNode(g, ID);
82   while(node->next!=NULL){
83     node = node->next;
84     neighbours[i] = node->ID;
85     i++;
86   }
87   return neighbours;
88 }
```

ι) Untuk menggunakan struktur data, buatlah program klien seperti yang ditunjukkan kode pada halaman selanjutnya.

graph.h × graph.c × *klien_graph.c ×

```c
//author Tennov

#include <stdio.h>
#include "graph.h"

int main(){
  Node node;
  int *neighbours, i;

  Graph G = ConstructGraph(7);  //construct graph with 7 nodes
  int V = GetNumberOfNodes(G);  //get number of nodes
  printf("Number of Nodes in Graph: %u\n", V);

  //add node 1 to 7 to graph G iteratively
  for(i = 1; i<=7; ++i)
    G = AddNode(G, i);

  //search node 1
  node = SearchNode(G, 1);
  printf("Node is found, id = %d\n", GetNodeID(node));

  //add edge, 0 weight means graph with equal weights
  G = AddEdge(G,1,2,0);
  G = AddEdge(G,1,4,0);
  G = AddEdge(G,1,3,0);
  G = AddEdge(G,2,4,0);
  G = AddEdge(G,2,5,0);
  G = AddEdge(G,3,6,0);
  G = AddEdge(G,4,6,0);
  G = AddEdge(G,4,7,0);
  G = AddEdge(G,4,3,0);
  G = AddEdge(G,5,4,0);
  G = AddEdge(G,5,7,0);
  G = AddEdge(G,7,6,0);

  //get neighbours of node 4
  neighbours = GetNeighbours(G,4);
  for(i=0;i<3;i++)
    printf("neighbours %d = %d\n", i+1, neighbours[i]);

  return 0;
}
```

φ) Kompilasi dan jalankan program sebagai berikut:

```
tennov@siftworkstation:~/c_alg_sd_S1/mg14_sesi3$ gcc -Wall -g klien_graph.c graph.c
tennov@siftworkstation:~/c_alg_sd_S1/mg14_sesi3$ ./a.out
Number of Nodes in Graph: 7
Node is found, id = 1
neighbours of node 4
neighbours of 1 = 6
neighbours of 2 = 7
neighbours of 3 = 3
```

3. **Tugas Pemrograman**:
    i. Dengan menggunakan Graph ADT dengan representasi matrix kedekatan (*adjacency matrix*), implementasikanlah fungsi `GetNumSuccessors` dan `Successors` yang dideklarasikan pada antarmuka.
    ii. Dengan menggunakan Graph ADT dengan representasi list kedekatan (*adjacency list*), buatlah fungsi untuk menghapus busur, menghapus simpul, dan menghapus graph. Tambahkan deklarasi di antarmuka, definisikan dengan detil di implementasi dan test pada klien.

Setoran

1. Kertas jawaban untuk soal ulasan dan pendalaman.
2. Kode program untuk tugas pemrograman.
3. Graph ADT dengan representasi matrix dan list kedekatan di laptop masing-masing.