MINGGU 9
Heaps

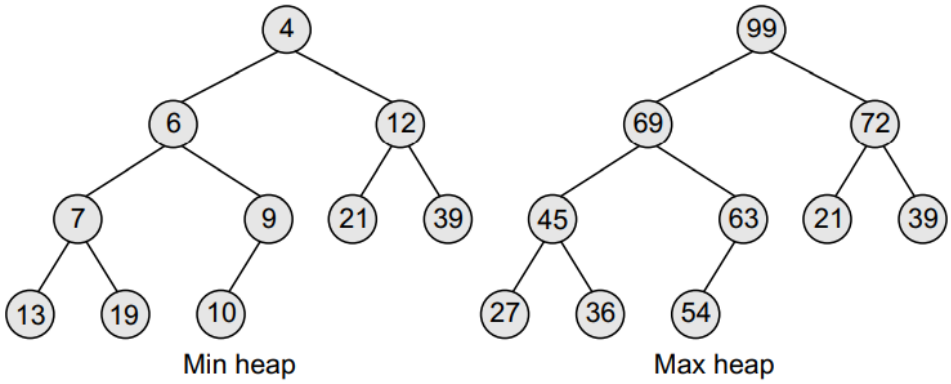
DESKRIPSI TEMA

A. Binary Heaps

Adalah sebuah Tree dengan properti sebagai berikut:

1. Merupakan Tree yang lengkap sehingga Binary Heap dapat diletakan dalam sebuah Array.
2. Sebuah Binary Heap hanya antara **Min Heap** atau **Max Heap**. Min Heap berarti value pada root harus merupakan nilai **terkecil** bagi anak-anaknya. Max Heap merupakan kebalikan dari Min Heap, yang berarti value pada root harus merupakan nilai **terbesar** bagi anak-anaknya.

Gambaran mengenai Min Heap dan Max Heap dapat dilihat pada gambar di bawah ini.



B. Binomial Heap

Adalah sebuah kumpulan dari Binomial Trees. Binomial Tree memiliki properti sebagai berikut:

1. Binomial Tree yang memiliki order 0 hanya memiliki `1` node
2. Binomial Tree yang memiliki `i` order memiliki child node sebanyak `i-1`, `i-2`, ..., `2`, `1`, dan `0`.
3. Sebuah Binomial Tree B_i memiliki node sebanyak 2^i.
4. Tinggi sebuah Binomial Tree B_i adalah `i`.

Binomial Heap H adalah kumpulan dari Binomial Tree yang memenuhi syarat-syarat berikut.

1. Setiap Binomial Tree di dalam H memenuhi ketentuan Minimum Heap.
2. Bisa terdapat satu atau nol Binomial Tree untuk setiap order termasuk order nol.

C. Fibonacci Heaps

Adalah sebuah heaps dengan kumpulan tree. Kurang lebih berbasis pada Binomial Heaps. Perbedaannya adalah Fibonacci Heaps lebih fleksibel sehingga meningkatkan batasan waktu.

CAPAIAN PEMBELAJARAN MINGGUAN (SUB-CAPAIAN PEMBELAJARAN)

1. Mahasiswa dapat mengimplementasikan Binary Heaps menggunakan Bahasa C
2. Mahasiswa dapat mengimplementasikan Fibinacci Heaps menggunakan Bahasa C

PENUNJANG PRAKTIKUM

1. Aplikasi CodeBlocks
2. Aplikasi Dev-C++ (alternatif)

LANGKAH-LANGKAH PRAKTIKUM

A. Binary Heaps

- Tutorial 1.1 – Insertion Binary Heaps

1. Buat sebuah file dengan nama W09_InsertionBinaryHeaps.c
2. Salin potongan code berikut. Perhatikan juga nomor barisnya.
3. Komentar yang terdapat code ditujukan untuk membantu. Bersifat opsional untuk disalin.
4. Perhatikan juga indentasi code.

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

typedef struct MinHeap{
    int *hArr;
    int capacity;
    int heapSize;
}MinHeap;

MinHeap createHeap(int cap){ // Function untuk membuat heap baru
    MinHeap newHeap;
    newHeap.heapSize = 0;
    newHeap.capacity = cap;
    newHeap.hArr = malloc(sizeof(int) * cap);

    int i;
    for (i=0; i<cap; i++){
        newHeap.hArr[i] = 0;
    }

    return newHeap;
}

int parent(int i){ // Function untuk mencari index parent
    return (i-1)/2;
}

int left(int i){ // Function untuk mencari index anak kiri
    return (2*i + 1);
}

int right(int i){ // Function untuk mencari index anak kanan
    return (2*i + 2);
}

// Function untuk menukar 2 buah angka
void swap(int *x, int *y){
    int temp = *x;
    *x = *y;
    *y = temp;
}
```

```c
// Function untuk memasukkan angka baru ke heap
void insertKey(MinHeap *mHeap, int k){
  printf("Current Heap Size = %d\n\n", mHeap->heapSize);
  printf("Inserting %d to heap\n", k);
  if(mHeap->heapSize == mHeap->capacity){
    printf("\nOverflow: Could not insertKey\n");
    return;
  }

  // Pertama-tama, masukkan angka ke dalam index terakhir
  mHeap->heapSize++;
  int i = mHeap->heapSize - 1;
  mHeap->hArr[i] = k;

  // Memperbaiki properti dari min heap
  // Jika angka baru yang masuk tidak memenuhi kriteria min heap
  while (i != 0 && mHeap->hArr[parent(i)] > mHeap->hArr[i]){
    swap(&mHeap->hArr[i], &mHeap->hArr[parent(i)]);
    i = parent(i);
  }
}
```

```c
int getMin(MinHeap *mHeap){
  return mHeap->hArr[0];
}

void printHeap(MinHeap mHeap){
  printf("Current heap : ");
  for(int i=0; i<mHeap.heapSize; i++){
    printf("%d ", mHeap.hArr[i]);
  }
  printf("\n");
}
```

```c
int main(){
  MinHeap mHeap;

  mHeap = createHeap(11);

  printHeap(mHeap);
  insertKey(&mHeap, 3);
  printHeap(mHeap);
  insertKey(&mHeap, 2);
  printHeap(mHeap);
  deleteKey(&mHeap, 1);
  printHeap(mHeap);
  insertKey(&mHeap, 15);
  printHeap(mHeap);
  insertKey(&mHeap, 5);
  printHeap(mHeap);
  insertKey(&mHeap, 4);
  printHeap(mHeap);
  insertKey(&mHeap, 45);
  printHeap(mHeap);
  printf("Min Value in Heap: %d \n", getMin(&mHeap));

  extractMin(&mHeap);
  printHeap(mHeap);

  deleteKey(&mHeap, 1);
  printHeap(mHeap);

  return 0;
}
```

- Tutorial 1.2 – Deletion Binary Heaps
  1. Buat sebuah file dengan nama W09_DeletionBinaryHeaps.c
  2. Salin Tutorial 1.1 ke dalam file tersebut.
  3. Salinlah tambahan potongan code berikut ke dalam file tersebut.
  4. Komentar yang terdapat code ditujukan untuk membantu. Bersifat opsional untuk disalin.
  5. Perhatikan juga indentasi code.

```
106   // Mengurai nilai pada index i menjadi newValue dengan asumsi
107   // nilai newValue lebih kecil dari hArr[i]
108   void decreaseKey(MinHeap *mHeap, int i, int newValue)
109   {
110       mHeap->hArr[i] = newValue;
111       while (i != 0 && mHeap->hArr[parent(i)] > mHeap->hArr[i])
112       {
113           swap(&mHeap->hArr[i], &mHeap->hArr[parent(i)]);
114           i = parent(i);
115       }
116   }
117
118   // Pertama-tama dengan menguangi nilainya menjadi negatif tak terhingga
119   // Agar menjadi paling kecil
120   // Lalu memanggil extractMin() untuk menghapusnya
121   void deleteKey(MinHeap *mHeap, int i){
122     printf("Current Heap Size = %d\n\n", mHeap->heapSize);
123     printf("Deleting index %d from heap\n", i);
124     decreaseKey(mHeap, i, INT_MIN);
125     extractMin(mHeap);
126   }
```

```
66  void MinHeapify(MinHeap *mHeap, int i){
67    int l = left(i);
68    int r = right(i);
69    int smallest = i;
70    if (l < mHeap->heapSize && mHeap->hArr[l] < mHeap->hArr[i]) smallest = l;
71    if (r < mHeap->heapSize && mHeap->hArr[r] < mHeap->hArr[smallest]) smallest = r;
72    if (smallest != i) {
73      swap(&mHeap->hArr[i], &mHeap->hArr[smallest]);
74      MinHeapify(mHeap, smallest);
75    }
76  }
77
78  int extractMin(MinHeap *mHeap){
79    if(mHeap->heapSize <= 0) return INT_MAX;
80    if(mHeap->heapSize == 1) {
81      mHeap->heapSize--;
82      return mHeap->hArr[0];
83    }
84
85    // Simpan nilai minimumnya dan hapus dari heap
86    int root = mHeap->hArr[0];
87    mHeap->hArr[0] = mHeap->hArr[mHeap->heapSize-1];
88    mHeap->heapSize--;
89    MinHeapify(mHeap, 0);
90
91    return root;
92  }
```

B. Fibonacci Heaps

- Tutorial 2.1 – Insertion Fibonacci Heaps

1. Buat sebuah file dengan nama W09_InsertionFibonacciHeaps.c
2. Salinlah tambahan potongan code berikut ke dalam file tersebut.
3. Komentar yang terdapat code ditujukan untuk membantu. Bersifat opsional untuk disalin.
4. Perhatikan juga indentasi code.

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

typedef struct node {
    struct node* parent;
    struct node* child;
    struct node* left;
    struct node* right;
    int key;
}node;

// Function untuk memasukkan node ke dalam Heap
void insertion(node* *mini, int val){
    struct node* new_node = (node*)malloc(sizeof(node));
    new_node->key = val;
    new_node->parent = NULL;
    new_node->child = NULL;
    new_node->left = new_node;
    new_node->right = new_node;
    if ((*mini) != NULL) {
        ((*mini)->left)->right = new_node;
        new_node->right = *mini;
        new_node->left = (*mini)->left;
        (*mini)->left = new_node;
        if (new_node->key < (*mini)->key)
            (*mini) = new_node;
    }
    else {
        (*mini) = new_node;
    }
}
```

```
34  // Function untuk menampilkan Heap
35  void display(struct node* mini, int no_of_nodes){
36      node* ptr = mini;
37      if (ptr == NULL) printf("The Heap is Empty\n");
38
39      else {
40        printf("The root nodes of Heap are: \n");
41          do {
42        printf("%d", ptr->key);
43              ptr = ptr->right;
44              if (ptr != mini) {
45                printf("-->");
46              }
47          } while (ptr != mini && ptr->right != NULL);
48      printf("\nThe heap has %d nodes\n", no_of_nodes);
49      }
50  }
51
52  // Function untuk mencari node minimum dalam Heap
53  void find_min(struct node* mini){
54    printf("min of heap is: %d\n", mini->key);
55  }
56
57  int main(){
58    // Membuat variabel "mini" sebagai minimum pointer
59    struct node* mini = NULL;
60
61    int no_of_nodes = 0;
62      no_of_nodes = 7;
63      insertion(&mini, 4);
64      insertion(&mini, 3);
65      insertion(&mini, 7);
66      insertion(&mini, 5);
67      insertion(&mini, 2);
68      insertion(&mini, 1);
69      insertion(&mini, 10);
70
71      display(mini, no_of_nodes);
72
73      find_min(mini);
74
75      return 0;
76  }
```

- Tutorial 2.2 – Deletion Fibonacci Heaps
  5. Buat sebuah file dengan nama W09_DeletionFibonacciHeaps.c
  6. Salinlah tambahan potongan code berikut ke dalam file tersebut.
  7. Komentar yang terdapat code ditujukan untuk membantu. Bersifat opsional untuk disalin.
  8. Perhatikan juga indentasi code.

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

// Buat sebuah struct untuk merepresentasikan node dalam heap
typedef struct node {
    struct node* parent; // Parent pointer
    struct node* child; // Child pointer
    struct node* left; // Pointer ke node di kiri
    struct node* right; // Pointer ke node di kanan
    int key; // Value sebuah node
    int degree; // Degree sebuah node
    char mark; // Tanda hitam atau putih sebuah node
    char c; // Flag untuk membantu saat dalam function Find
}node;

// Membuat variabel "mini" sebagai minimum pointer
struct node* mini = NULL;

// Mendeklarasikan jumlah node dalam heap
int no_of_nodes = 0;

// Function untuk memasukkan node ke dalam heap
void insertion(int val){
    struct node* new_node = (struct node*)malloc(sizeof(struct node));
    new_node->key = val;
    new_node->degree = 0;
    new_node->mark = 'W';
    new_node->c = 'N';
    new_node->parent = NULL;
    new_node->child = NULL;
    new_node->left = new_node;
    new_node->right = new_node;
    if (mini != NULL) {
        (mini->left)->right = new_node;
        new_node->right = mini;
        new_node->left = mini->left;
        mini->left = new_node;
        if (new_node->key < mini->key) mini = new_node;
    }
    else {
        mini = new_node;
    }
    no_of_nodes++;
}
```

```
48  // Menghubungkan node heap dengan hubungan parent-child
49  void Fibonnaci_link(struct node* ptr2, struct node* ptr1){
50      (ptr2->left)->right = ptr2->right;
51      (ptr2->right)->left = ptr2->left;
52
53      if (ptr1->right == ptr1) mini = ptr1;
54
55      ptr2->left = ptr2;
56      ptr2->right = ptr2;
57      ptr2->parent = ptr1;
58
59      if (ptr1->child == NULL) ptr1->child = ptr2;
60
61      ptr2->right = ptr1->child;
62      ptr2->left = (ptr1->child)->left;
63      ((ptr1->child)->left)->right = ptr2;
64      (ptr1->child)->left = ptr2;
65
66      if (ptr2->key < (ptr1->child)->key) ptr1->child = ptr2;
67
68      ptr1->degree++;
69  }
```

```
71  // Menggabungkan heap
72  void Consolidate(){
73      int temp1;
74      float temp2 = (log(no_of_nodes)) / (log(2));
75      int temp3 = temp2;
76      struct node* arr[temp3];
77      for (int i = 0; i <= temp3; i++) arr[i] = NULL;
78
79      node* ptr1 = mini;
80      node* ptr2;
81      node* ptr3;
82      node* ptr4 = ptr1;
83      do {
84          ptr4 = ptr4->right;
85          temp1 = ptr1->degree;
86          while (arr[temp1] != NULL) {
87              ptr2 = arr[temp1];
88              if (ptr1->key > ptr2->key) {
89                  ptr3 = ptr1;
90                  ptr1 = ptr2;
91                  ptr2 = ptr3;
92              }
93              if (ptr2 == mini) mini = ptr1;
94              Fibonnaci_link(ptr2, ptr1);
95              if (ptr1->right == ptr1) mini = ptr1;
96              arr[temp1] = NULL;
97              temp1++;
98          }
99          arr[temp1] = ptr1;
100         ptr1 = ptr1->right;
101     } while (ptr1 != mini);
102
103     mini = NULL;
104     for (int j = 0; j <= temp3; j++) {
105         if (arr[j] != NULL) {
106             arr[j]->left = arr[j];
107             arr[j]->right = arr[j];
108             if (mini != NULL) {
109                 (mini->left)->right = arr[j];
110                 arr[j]->right = mini;
111                 arr[j]->left = mini->left;
112                 mini->left = arr[j];
113                 if (arr[j]->key < mini->key) mini = arr[j];
114             }
115             else {
116                 mini = arr[j];
117             }
118             if (mini == NULL) mini = arr[j];
119             else if (arr[j]->key < mini->key) mini = arr[j];
120         }
121     }
122 }
```

```
124  // Function untuk mengeluarkan nilai minimum dari heap
125  void Extract_min(){
126      if (mini == NULL) printf("The heap is empty\n");
127      else {
128          node* temp = mini;
129          node* pntr;
130          pntr = temp;
131          node* x = NULL;
132          if (temp->child != NULL) {
133              x = temp->child;
134              do {
135                  pntr = x->right;
136                  (mini->left)->right = x;
137                  x->right = mini;
138                  x->left = mini->left;
139                  mini->left = x;
140                  if (x->key < mini->key) mini = x;
141                  x->parent = NULL;
142                  x = pntr;
143              } while (pntr != temp->child);
144          }
145          (temp->left)->right = temp->right;
146          (temp->right)->left = temp->left;
147          mini = temp->right;
148          if (temp == temp->right && temp->child == NULL) mini = NULL;
149          else {
150              mini = temp->right;
151              Consolidate();
152          }
153          no_of_nodes--;
154      }
155  }
156
157  // Memutuskan sebuah node dalam heap untuk diletakkan dalam daftar root
158  void Cut(struct node* found, struct node* temp)
159  {
160      if (found == found->right) temp->child = NULL;
161
162      (found->left)->right = found->right;
163      (found->right)->left = found->left;
164      if (found == temp->child) temp->child = found->right;
165
166      temp->degree = temp->degree - 1;
167      found->right = found;
168      found->left = found;
169      (mini->left)->right = found;
170      found->right = mini;
171      found->left = mini->left;
172      mini->left = found;
173      found->parent = NULL;
174      found->mark = 'B';
175  }
```

```
177  // Function rekursif untuk Cut
178  void Cascase_cut(struct node* temp){
179      node* ptr5 = temp->parent;
180      if (ptr5 != NULL) {
181          if (temp->mark == 'W') {
182              temp->mark = 'B';
183          }
184          else {
185              Cut(temp, ptr5);
186              Cascase_cut(ptr5);
187          }
188      }
189  }
190
191  // Function untuk menurunkan value node dalam heap
192  void Decrease_key(struct node* found, int val){
193      if (mini == NULL) printf("The Heap is Empty\n");
194
195      if (found == NULL) printf("Node not found in the Heap\n");
196
197      found->key = val;
198
199      struct node* temp = found->parent;
200      if (temp != NULL && found->key < temp->key) {
201          Cut(found, temp);
202          Cascase_cut(temp);
203      }
204      if (found->key < mini->key)
205          mini = found;
206  }
207
208  // Function untuk mencari suatu node
209  void Find(struct node* mini, int old_val, int val){
210      struct node* found = NULL;
211      node* temp5 = mini;
212      temp5->c = 'Y';
213      node* found_ptr = NULL;
214      if (temp5->key == old_val) {
215          found_ptr = temp5;
216          temp5->c = 'N';
217          found = found_ptr;
218          Decrease_key(found, val);
219      }
220      if (found_ptr == NULL) {
221          if (temp5->child != NULL)
222              Find(temp5->child, old_val, val);
223          if ((temp5->right)->c != 'Y')
224              Find(temp5->right, old_val, val);
225      }
226      temp5->c = 'N';
227      found = found_ptr;
228  }
```

```
230  // Menghapuss node dari heap
231  void Deletion(int val){
232      if (mini == NULL) printf("The heap is empty\n");
233      else {
234          // Menurunkan value dalam node menajadi 0
235          Find(mini, val, 0);
236
237          // Memanggil function Extract_min untuk
238          // menghapus nilai minimum dalam node
239          Extract_min();
240          printf("Key Deleted\n");
241      }
242  }
243
244  // Function untuk menampilan isi heap
245  void display(){
246      node* ptr = mini;
247      if (ptr == NULL) printf("The heap is empty\n");
248
249      else {
250      printf("The root nodes of Heap are: \n");
251          do {
252        printf("%d", ptr->key);
253              ptr = ptr->right;
254              if (ptr != mini) {
255          printf("-->");
256              }
257          } while (ptr != mini && ptr->right != NULL);
258      printf("\nThe heap has %d nodes\n\n", no_of_nodes);
259      }
260  }
```

```c
int main(){
    // Membuat heap dan memasukan 3 node ke dalamnya
    printf("Creating an initial heap\n");
    insertion(5);
    insertion(2);
    insertion(8);

    // Menampilkan heap
    display();

    // Mengeluarkan nilai minimum dari heap
    printf("Extracting min\n");
    Extract_min();
    display();

    // Mengurangi node dengan value 8 menjadi 7
    printf("Decrease value of 8 to 7\n");
    Find(mini, 8, 7);
    display();

    // Menghapus node dengan value 7
    printf("Delete the node 7\n");
    Deletion(7);
    display();

    return 0;
}
```

REFERENSI

Thareja, Reema. 2014. Data Structures Using C Second Edition. Oxford University.
www.geeksforgeeks.com