Aayush Shah
19BCE245
13 October 2021

Design and Analysis of Algorithms

Practical 6

• **Code :**

```
/*
19BCE245 Aayush Shah
DAA practical 6
Binomial Heap
*/

#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>
#include <limits.h>

typedef struct _binom_tree_node {
     int key;       //data value in node
     struct _binom_tree_node *child, *parent, *sibling;
//connected nodes
     int k;         //degre of node
} binom_tree_node;

typedef struct _binom_heap_node {
     binom_tree_node *root;        //root of the heap
     struct _binom_heap_node *next;          //next node in
heap
} binom_heap_node;       //This will be a binomial tree

typedef struct _binom_heap {
     binom_heap_node *trees;       //trees in heap
     int size;      //number of trees in the heap
} binom_heap;
```

```
/* Merges two trees. swapped node pointer will be returned. */
binom_heap_node * merge_binom_trees(binom_heap_node *a,
binom_heap_node *b){
     if(a->root->key > b->root->key){
          binom_heap_node temp = *a;
          *a = *b;
          *b = temp;
     }

     b->root->parent = a->root;
     b->root->sibling = a->root->child;
     a->root->child = b->root;
     a->root->k++;

     return a;
}

/* fixes binomial heap after merge operation. */
void _fixup_binom_heap(binom_heap *c){
     if(c->size <= 1){
          return;
     }

     binom_heap_node *curr = c->trees,
     *next = c->trees->next,
     *next_next = c->trees->next->next;

     while(curr){
          if(next==NULL){
               curr = next;
          }
          else if(curr->root->k < next->root->k){
               curr = next;
               next = next_next;
               if(next_next){
                    next_next = next_next->next;
               }
          }
          else if(next_next && curr->root->k == next->root->k
&& curr->root->k == next_next->root->k){
               curr = next;
               next = next_next;
               next_next = next_next->next;
```

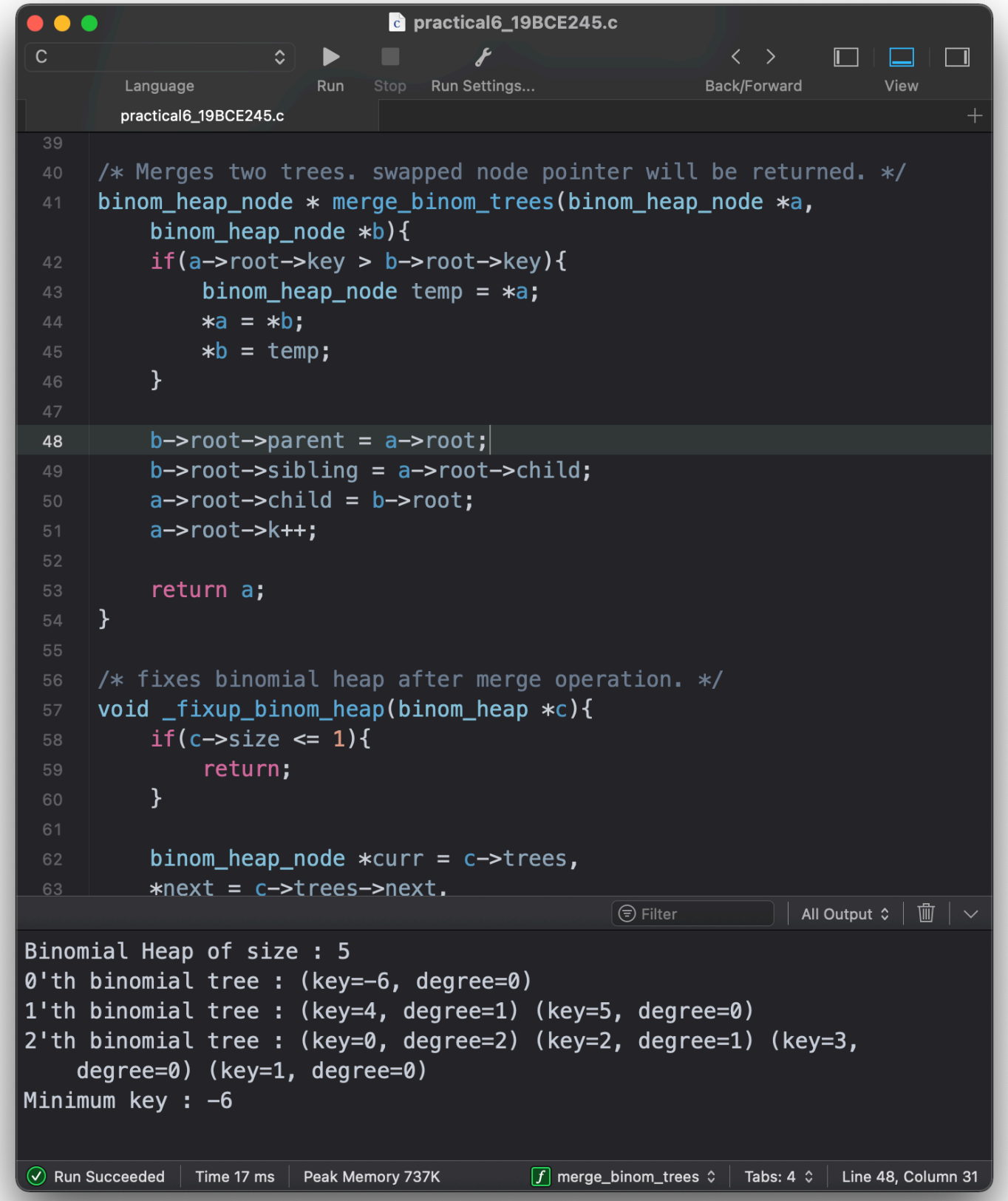
```
            }
            else if(curr->root->k == next->root->k){
                curr = merge_binom_trees(curr,next);
                curr->next = next_next;
                free(next);
                next = next_next;
                if(next_next){
                    next_next = next_next->next;
                    c->size--;
                }
            }
        }
}

/* prints binomial tree using preorder traversal. */
void _print_binom_tree(binom_tree_node *root){
        if(root) {
            printf("(key=%d, degree=%d) ", root->key, root->k);
            _print_binom_tree(root->child);
            _print_binom_tree(root->sibling);
        }
}

/* merges binomial heaps. */
void merge_binom_heaps(binom_heap *a, binom_heap *b,
binom_heap *c){
        binom_heap_node *curr_a = a->trees,
                                *curr_b= b->trees;
        c->size = a->size + b->size;

        if(curr_a->root->k < curr_b->root->k){
            c->trees = curr_a;
            curr_a= curr_a->next;
        }
        else{
            c->trees = curr_b;
            curr_b = curr_b->next;
        }

        binom_heap_node *curr_c = c->trees;

        while(curr_a && curr_b){
            if(curr_a->root->k < curr_b->root->k) {
```

```
                curr_c->next = curr_a;
                curr_a = curr_a->next;
            }
            else {
                curr_c->next = curr_b;
                curr_b = curr_b->next;
            }
            curr_c = curr_c->next;
        }

        if(curr_b){
            curr_c->next = curr_b;
        }
        else if(curr_a){
            curr_c->next = curr_a;
        }
}

/* performs union operation. */
void binom_heap_union(binom_heap *a, binom_heap *b, binom_heap
*c)
{
        if(a == NULL || b == NULL || c == NULL) {
            fprintf(stderr, "[in 'binom_heap_union'] Heaps a, b,
or c not initialized!\n");
            exit(1);
        }

        merge_binom_heaps(a, b, c);
        _fixup_binom_heap(c);
}

/* creates binomial heap. */
void binom_heap_create(binom_heap *heap, int key)
{
        if(heap == NULL) {
            fprintf(stderr, "[in 'binom_heap_create'] Heap not
initialized!\n");
            exit(1);
        }

        heap->trees = (binom_heap_node
*)malloc(sizeof(binom_heap_node));
```

```
     heap->trees->root = (binom_tree_node
*)malloc(sizeof(binom_tree_node));
     heap->trees->root->key = key, heap->trees->root->k = 0;
     heap->trees->root->child = heap->trees->root->parent =
heap->trees->root->sibling = NULL;
     heap->trees->next = NULL;

     heap->size = 1;
}

/* inserts element in binomial heap. */
void binom_heap_insert(binom_heap *heap, int key)
{
     binom_heap *new_heap = (binom_heap
*)malloc(sizeof(binom_heap));
     binom_heap_create(new_heap, key);

     binom_heap *merged_heap = (binom_heap
*)malloc(sizeof(binom_heap));
     binom_heap_union(heap, new_heap, merged_heap);
     *heap = *merged_heap;
     free(new_heap);
     free(merged_heap);
}

/* find minimum element in heap. */
int binom_heap_find_min(binom_heap *heap)
{
     binom_heap_node *temp = heap->trees;
     binom_heap_node *min_node = temp;

     while(temp) {
          if(min_node->root->key > temp->root->key){
               min_node = temp;
          }
          temp = temp->next;
     }

     return min_node->root->key;
}

/* prints binomial heap. */
```

```
void binom_heap_print(binom_heap *heap)
{
     int i = 0;
     printf("Binomial Heap of size : %d\n", heap->size);
     for(binom_heap_node *temp=heap->trees; temp; temp=temp-
>next, i++) {
          printf("%d'th binomial tree : ", i);
          _print_binom_tree(temp->root);
          printf("\n");
     }
}

int main() {
         binom_heap a;

    /* Create a binomial heap. */
    binom_heap_create(&a, 0);

    /* Insert values in the heap. */
    binom_heap_insert(&a, 1);
    binom_heap_insert(&a, 2);
    binom_heap_insert(&a, 3);
    binom_heap_insert(&a, 4);
    binom_heap_insert(&a, 5);
    binom_heap_insert(&a, -6);
    binom_heap_print(&a);

    /* Find the minimum key. */
    printf("Minimum key : %d\n", binom_heap_find_min(&a));

    return 0;
}
```

- **Output :**


practical6_19BCE245.c
C
Language
Run
Stop
Run Settings...
Back/Forward
View
practical6_19BCE245.c
39
40 /* Merges two trees. swapped node pointer will be returned. */
41 binom_heap_node * merge_binom_trees(binom_heap_node *a,
binom_heap_node *b){
42 if(a->root->key > b->root->key){
43 binom_heap_node temp = *a;
44 *a = *b;
45 *b = temp;
46 }
47
48 b->root->parent = a->root;
49 b->root->sibling = a->root->child;
50 a->root->child = b->root;
51 a->root->k++;
52
53 return a;
54 }
55
56 /* fixes binomial heap after merge operation. */
57 void _fixup_binom_heap(binom_heap *c){
58 if(c->size <= 1){
59 return;
60 }
61
62 binom_heap_node *curr = c->trees,
63 *next = c->trees->next,
Filter
All Output
Binomial Heap of size : 5
0'th binomial tree : (key=-6, degree=0)
1'th binomial tree : (key=4, degree=1) (key=5, degree=0)
2'th binomial tree : (key=0, degree=2) (key=2, degree=1) (key=3,
degree=0) (key=1, degree=0)
Minimum key : -6
Run Succeeded
Time 17 ms
Peak Memory 737K
merge_binom_trees
Tabs: 4
Line 48, Column 31