Advanced Data Structure: Project1 Report

- Name: Peijia Xu
- UFID: 8085-8038
- Email: peijia.xu@ufl.edu

- Structure of codes
    - program structure
        - main: proj1.cpp
        - data sturcture implement: ds.h and ds.cpp
    - proj1.cpp
        - main function
            - Initialize the data structure
            - firstly run all of the commands
            - If there are still some building not finished finishe the building
        - Helper functions
            - string my_copy(string s,int start,int end);
                - copy string from start to end
            - void parse_line(string line,command & comm);
                - parse the input line to the struct command
            - void insert(command cur,min_heap& heap,br_tree &tree);
                - insert from command to min_heap and black-red tree
                - insert to the end of the heap and heapify when the current construction finished
                - raise error if insert to the tree fail
            - void print(command cur,br_tree tree);
                - Printbuilding num
    - ds.h
        - Defining structure and classes
        - struct br_node;
            - node structure for black-red tree
        - struct heap_node;
            - node of min-heap
        - struct command;
            - structure to save command after parsing
        - class br_tree;
            - class implement black-red tree and related functions
        - class min_heap;
            - structure implement min-heap and related function
    - ds.cpp
        - Implement class functions
        - helper funcs
            - bool cmp_heap(heap_node a,heap_node b);
                - return if a is defined as greater than b
                - which is with longer executed_time or bigger id when executed_time is same
        - br_tree functions

- Functions related to delete node from br_trees
    - void br_tree::deal_Xbn(br_node*py,br_node*v,br_node*y);
        - X = L,R
        - classify the deficiency case more specifically as the location(L or R) of y and color of v is certained(black)
    - void br_tree::deal_Xrn(br_node*py,br_node*v,br_node*y);
        - X = L,R
        - classify the deficiency case more specifically as the location(L or R) of y and color of v is certained(red)
    - void br_tree::del_Xbn_case(br_node*py,br_node*v,br_node*y);
        - X = X, L, R, n =0,1,case might extende with number
        - deal the deficiency specific case as function name after classification
        - Defined as same as the slide (same way to name the var)
    - void br_tree::del_Xbn_case(br_node*py,br_node*v,br_node*y);
        - X = X, L, R, n =0,1,case might extende with number
        - deal the deficiency specific case as function name after classification
        - Defined as same as the slide (same way to name the var)
    - void br_tree::del_Xbn_case(br_node*py,br_node*v,br_node*y);
        - X = X, L, R, n =0,1,case might extende with number to specify certain
        - deal the deficiency specific case as function name after classification
        - Defined as same as the slide (same way to name the var)
    - void br_tree::del_Xrn_case(br_node*py,br_node*v,br_node*y,br_node*w);
        - X = X, L, R, n =0,1,case might extende with number to specify certain
        - deal the deficiency specific case as function name after classification
        - Defined as same as the slide (same way to name the var)
    - void deal_deficient(br_node* node)
        - start to deal deficiency
        - will classify the situation and call deal_Xcn()
    - void del(br_node*node)
        - delete node from black-red tree
        - degree 2: find replace, swap and call del again
        - degree < 2: start to delete following slides
- Insert
    - bool br_tree::insert(br_node* node);
        - insert the node to the tree
    - void br_tree::check_bottom_up(br_node* node);
        - check if there are two consequent red node
    - void raise_error(command cur);
        - raise error
        - print time of wrong command
- Print
    - bool br_tree::print(int lb,int ub);
        - print at root

        - true for no output
      - bool br_tree::print_at(int lb,int ub,br_node*node);
        - print at particular node
        - will print recursively
      - void queue(br_node* node);
        - add to print queue
      - void write();
        - write the print queue nodes to output file

    - Other functions

      - void br_tree::change_parent(br_node*py,br_node*v);
        - change parent of v to parent of py
      - void br_tree::set_output(ofstream *f);
        - set output file pointer
      - br_node* br_tree::lookup(br_node* node,int id);
        - look through the tree to find node
        - return the suitable parent if not exist
      - void br_tree::swap_node(br_node*a,br_node*b);
        - swap value and heap pointer of node a and b
      - void br_tree::set_heap(min_heap *h);
        - set the corresponding heap
- min-heap funcitons
  - heap_node min_heap::get(int id);
    - get node with heap id - id
  - void min_heap::insert(heap_node node,int i);
    - recursively insert node bottom-up
    - start form the end of the prority queue
    - just save it at the end when current building is under construction(parameter building > 0)
  - int min_heap::pop(br_tree & tree,int time);
    - pop the finshed node, cal del function of tree to delete the corresponding node in the black-red tree
  - void min_heap::heapify(int id);
  - heapify top-down recursively
  - call start at the root(id 0)
  - end when id >= num(no children)
  - int min_heap::update_root();
    - increase building and executed_time of root node by 1
    - simulater building procedure
    - return 0 for finish, -1 for finish and need to pop, otherwise under construction
  - void min_heap::update_pointer(int id,br_node*p);
    - update corresponding pointer of heap node
  - void min_heap::set_output(ofstream *f);
    - set output file pointer
  - void min_heap::write(int id,int time);
    - write when building completed