g++ (from the GNU Compiler Collection) was used to compile this program. The source consists of three files: main.cpp, rb_tree.cpp, rb_tree.h.

main.cpp:
Entry point of the program - responsible for reading from the input file, initializing the red black tree, taking commands. Consists of only int main().

rb_tree.h
Header file defining the interface of the rb_tree class and the rb_node struct.

rb_tree.cpp
Implementation file for the rb_tree class.

struct rb_node:
Attributes:
rb_node* left;
- pointer to left child
rb_node* right;
- pointer to right child
rb_node* parent;
- pointer to parent node
bool black;
- whether the node is black (or red)
int count;
- count of events with given ID
int id;
- ID of node

Methods:
rb_node(int id, bool black)
- Constructor

class rb_tree:
Attributes:
rb_node* root;
- Pointer to root node
rb_node* nil;
- Pointer to nil node (sentinel node used to simplify operations)

Public Methods:
rb_tree();
- constructor for empty tree

```
 rb_tree(vector<pair<int, int> > &init);
  - constructor for tree initialized using list of id/count pairs
    calls init_rec to recursively construct the tree
 ~rb_tree();
  - destructor
    calls postorder_delete() to traverse the tree and delete nodes
 int increase(int id, int count);
  - increases event with id by count, or inserts it as a red node if it does not exist
    calls fixup_insert() to resolve rule violations and create_node() to create new nodes
 int reduce(int id, int count);
  - decreases event with id by count and removes it if count goes to 0 as a result
    calls fixup_delete() to resolve rule violations and transplant() to move subtrees
 int count (int id);
  - returns count of node with given id
 int in_range(int low, int high);
  - returns sum of counts of nodes within range
    calls in_range_rec() to traverse the tree recursively
 pair<int, int> next(int id);
  - returns the id and count of the successor of event with given id
 pair<int, int> previous(int id);
  - returns the id and count of the predecessor of event with given id
 void print();
  - prints the tree in-order (for debugging)
 void level_print();
  - prints the tree in level-order (for debugging)

Private Methods:
 void postorder_delete(rb_node* current);
  - Deletes nodes in post-order
 void in_order_print(rb_node* node);
  - Prints nodes in-order
 void fixup_insert(rb_node* node);
  - Fixes rule violations after an insertion
 void left_rotate(rb_node* node);
  - Performs left rotation
 void right_rotate(rb_node* node);
  - Performs right rotation
 void transplant(rb_node* original_node, rb_node* new_node);
  - Moves node new_node to position occupied by original_node
 void fixup_delete(rb_node* current);
  - Fixes rule violations after a deletion
 void in_range_rec(int low, int high, rb_node* current, int& result);
  - Recursively sums up counts of events in the range and adds them to result
```

void init_rec(vector<pair<int, int> > &init, int level, rb_node* parent, bool left, int low, int high);

- Recursively constructs a tree given a list of event/id pairs

rb_node* create_node(int id, bool black);

- Helper to create a node and initialize its pointers to the nil sentinel