set_parallel

This class uses a vector of sets to store your data. Just like the standard C++ set, this class will store unique elements following a specific order.

This class takes advantage of OpenMP to make the insertion, deletion and lookup of elements faster than the standard C++ set. This is done by creating a vector of sets, the size of the vector is determined by the number of threads your CPU has. This means that each thread oversees one set (tree) in the vector. A vector of right end points is used to determine in which tree each element should be inserted into. Think of the right end points as dividers, so if we have ***n*** threads then we have ***n-1*** right end points (dividers).

After each insertion and deletion, the trees are rebalanced. This is done by first figuring out what the new right end points should be so that the total number of elements are evenly split between the trees, then the function will go through each element in the tree and determine if it should be moved to another tree.

Constructors

The default constructor will only work if you are using strings. The default constructor uses the ascii table along with the number of threads your CPU has to set letters as your right end points.

The constructor that takes in two parameters should be used when you want the right end points to be determined from a given maximum and minimum value. If you are using a self-created data type with this constructor, the data type must also have the division (/) operator.

The constructor that takes a vector can be used to set the right end points when you construct the parallel set if you give it a vector with the same size as the number of threads your CPU has. If the vector given is larger then the constructor will create the right end points given this sample data and insert those elements from the given vector. Any data type, including strings, should work with this constructor.

This class also contains a copy constructor, copy assignment, move constructor, and move assignment.

Functions

The **insert_parallel(const std::vector<T> &)** function will insert a vector of elements in parallel, then rebalance. This is done by each thread going through every element in the vector and checking whether the element should go into its tree. If the element belongs in its tree then it inserts it, if it does not belong then it simply moves on to the next element in the vector.

The **insert_element(const T &)** function will insert a single element.

The **erase_parallel(const std::vector<T> &)** function will remove a vector of elements in parallel, then rebalance. This is done by each thread going through every element in the vector and checking whether the element that the user wants deleted would be in its tree. If it is then it does the deletion, if not it simply moves on to check the next element in the vector.

The **find_parallel(const std::vector<T> &, std::vector<int> &)** function must take two parameters, a vector of elements that you want to check if they are stored in the container and a vector of integers that will hold the results, the vectors must be the same size. The vector with the results will contain the number 1 in the same position as the element it looked for from your other vector if the element is in the container and a 0 if it is not in the container.

The **find_element(const T &)** function will return 1 if the element you passed to the function is in the container or 0 otherwise.

The **print_all_data( )** function will print all your stored elements to the screen.

The **size_p( )** function will return an integer representing the number of elements in the container.

The **empty_p( )** function will return 1 if empty and 0 otherwise.

The **clear_p( )** function will clear your entire container.

The **get_right_end_point(const int &)** function will return the i^{th} right end point.