Time-Range LeaderBoard Query System

Generated by Doxygen 1.14.0

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 player_data Class Reference

Represents player data in the leaderboard.

`#include <LeaderBoard.hpp>`

Public Member Functions

- **player_data** ()
  Default constructor for player_data.
- player_data (int score, int player_id, int finish_time)
  Constructor for player_data given values.
- **player_data** (int score, int player_id, int finish_time)

Static Public Member Functions

- static player_data best_player (const player_data &a, const player_data &b)
  Compares two player_data parameters to determine the better player.
- static player_data **best_player** (const player_data &a, const player_data &b)

Public Attributes

- int **score**
  Player's score.
- int **player_id**
  Player's ID.
- int **finish_time**
  Player's finish time.

3.1.1 Detailed Description

Represents player data in the leaderboard.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 player_data()

```
player_data::player_data (
            int score,
            int player_id,
            int finish_time)
```

Constructor for player_data given values.

Parameters

score	Player's score
player_id	Player's ID
finish_time	Player's finish time

3.1.3 Member Function Documentation

3.1.3.1 best_player()

```
player_data player_data::best_player (
            const player_data & a,
            const player_data & b)  [static]
```

Compares two player_data parameters to determine the better player.

Parameters

a	The first player_data parameters.
b	The second player_data parameters.

Returns

The player_data parameters representing the better player.

The documentation for this class was generated from the following files:

- LeaderBoard.hpp
- project.cpp

3.2 segment_tree Class Reference

Public Member Functions

- segment_tree (int size)
  Constructor for segment_tree.
- void insert_into_tree (player_data p)
  Inserts a player_data into the segment tree.
- player_data query_the_tree_by_time (int start_time, int finish_time)
  Queries the segment tree for the best player within a time range.
- player_data query_the_tree_by_id (int begin_id, int end_id)
  Queries the segment tree by player ID range.
- bool update_player_data (player_data new_player_data)
  Updates the player data in the segment tree.
- **segment_tree** (int size)
- void **insert_into_tree** (player_data p)
- player_data **query_the_tree_by_time** (int start_time, int finish_time)
- player_data **query_the_tree_by_id** (int begin_id, int end_id)
- bool **update_player_data** (player_data new_player_data)

3.2.1 Constructor & Destructor Documentation

3.2.1.1 segment_tree()

```
segment_tree::segment_tree (
            int size)
```

Constructor for segment_tree.

Parameters

size	The size of the segment tree.

3.2.2 Member Function Documentation

3.2.2.1 insert_into_tree()

```
void segment_tree::insert_into_tree (
            player_data p)
```

Inserts a player_data into the segment tree.

Parameters

p	The player_data to insert.

3.2.2.2 query_the_tree_by_id()

```
player_data segment_tree::query_the_tree_by_id (
            int begin_id,
            int end_id)
```

Queries the segment tree by player ID range.

Parameters

begin↩ _id	The start ID of the player range.
end_id	The end ID of the player range.

Returns

The player_data representing the best player within the ID range.

3.2.2.3 query_the_tree_by_time()

```
player_data segment_tree::query_the_tree_by_time (
            int start_time,
            int finish_time)
```

Queries the segment tree for the best player within a time range.

Parameters

start_time	The start of the time range.
finish_time	The end of the time range.

Returns

The player_data representing the best player within the time range.

3.2.2.4 update_player_data()

```
bool segment_tree::update_player_data (
            player_data new_player_data)
```

Updates the player data in the segment tree.

Parameters

new_player_data	The new player_data to update.

Returns

True if the update was successful, false otherwise.

The documentation for this class was generated from the following files:

- LeaderBoard.hpp
- project.cpp

3.3 tree_node Class Reference

Represents a node in the segment tree.

`#include <LeaderBoard.hpp>`

Public Member Functions

- tree_node (int L, int R)

  Constructor for tree_node with bounds.
- tree_node (int L, int R, int score, int player_id, int finish_time)

  Constructor for tree_node with bounds and player data.
- **tree_node** (int L, int R)
- **tree_node** (int L, int R, int score, int player_id, int finish_time)

Public Attributes

- player_data **p**

  Player data.
- int **l**
- int **r**

  Left and right bounds.
- tree_node * **left_ptr**

  Pointer to left child.
- tree_node * **right_ptr**

  Pointer to right child.

3.3.1 Detailed Description

Represents a node in the segment tree.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 tree_node() [1/2]

```
tree_node::tree_node (
            int L,
            int R)
```

Constructor for tree_node with bounds.

Parameters

L	The left bound.
R	The right bound.

3.3.2.2 tree_node() [2/2]

```
tree_node::tree_node (
            int L,
            int R,
            int score,
            int player_id,
            int finish_time)
```

Constructor for tree_node with bounds and player data.

Parameters

L	The left bound.
R	The right bound.
score	Player's score.
player_id	Player's ID.
finish_time	Player's finish time.

The documentation for this class was generated from the following files:

- LeaderBoard.hpp
- project.cpp

Chapter 4

File Documentation

4.1 LeaderBoard.hpp

```
00001 #pragma once
00005 class player_data{
00006 public:
00007     int score;
00008     int player_id;
00009     int finish_time;
00010
00014     player_data();
00015
00022     player_data(int score, int player_id, int finish_time);
00023
00030     static player_data best_player(const player_data &a, const player_data &b);
00031 };
00032
00036 class tree_node{
00037     public:
00038         player_data p;
00039         int l, r;
00040         tree_node* left_ptr;
00041         tree_node* right_ptr;
00042
00048         tree_node(int L, int R);
00049
00058         tree_node(int L, int R, int score, int player_id, int finish_time);
00059
00060 };
00061
00062 class segment_tree{
00063     private:
00064         tree_node *root_node;
00065         int max_num_of_players;
00066         int reached_index;
00067
00076         void insert_helper_function(tree_node* &node, int left, int right, int index, player_data
      data);
00077
00084
00085         player_data query_helper_function(tree_node* &node, int start_interval, int end_interval);
00086
00095         bool update(tree_node* &node, int left, int right, player_data new_data);
00096
00106         player_data query_by_index_helper(tree_node* &node, int left, int right, int start_id, int
      end_id);
00107
00108     public:
00109
00114         segment_tree(int size);
00115
00120         void insert_into_tree(player_data p);
00121
00128         player_data query_the_tree_by_time(int start_time, int finish_time);
00129
00136         player_data query_the_tree_by_id(int begin_id, int end_id);
00137
00143         bool update_player_data(player_data new_player_data);
00144 };
```

4.2 project.cpp

```
00001 #include <iostream>
00002
00003 using namespace std;
00004
00005 class player_data{
00006
00007 public :
00008
00009     int score ;
00010     int player_id;
00011     int finish_time;
00012
00013
00014     player_data (): score(-1), player_id(-1) , finish_time(-1)
00015     {}
00016
00017     player_data(int score, int player_id , int finish_time) :
00018     score(score) , player_id(player_id) , finish_time(finish_time)
00019     {}
00020
00021     static player_data best_player(const player_data &a, const player_data &b) {
00022         if(a.score == -1 && b.score == -1) {return player_data();} //no players
00023
00024         if(a.score == -1) {return b;}
00025
00026         if(b.score == -1) {return a;}
00027
00028         if(a.score > b.score) {return a;}
00029         else if (b.score > a.score) {return b;}
00030         else {
00031
00032             if(a.finish_time < b.finish_time) {return a;}
00033             else if (b.finish_time < a.finish_time) {return b;}
00034             else {
00035
00036                 if(a.player_id > b.player_id) {return a;}
00037                 else if (b.player_id > a.player_id) {return b;}
00038                 // no duplication allowed in player registration for the ID
00039                 else {cerr« "Error : Player is registered twice"«endl;
00040                     return a;
00041                 }
00042             }
00043
00044
00045         }
00046
00047     }
00048
00049
00050 };
00051
00052 class tree_node {
00053 public:
00054     player_data p;
00055     int l, r;
00056     tree_node* left_ptr;
00057     tree_node* right_ptr;
00058
00059
00060     tree_node(int L, int R)
00061         : l(L), r(R) ,left_ptr(0), right_ptr(0) {
00062             p = player_data();
00063         }
00064
00065     tree_node(int L, int R, int score, int player_id, int finish_time) :
00066          l(L), r(R), left_ptr(0), right_ptr(0) {
00067             p = player_data(score,player_id,finish_time);
00068         }
00069
00070 };
00071
00072 class segment_tree {
00073
00074 private:
00075
00076 tree_node * root_node ;
00077 int max_num_of_players;
00078 int reached_index ; // cannot be global , due multible instances of the segment tree will make
      indexing wrong
00079
00080 void insert_helper_function(tree_node* &node ,int left ,int right,int index,player_data data){
00081
00082 if(!node) {
00083     node = new tree_node(left,right);
```

```
00084 }
00085
00086 if(left == right){
00087
00088     node->p = data;
00089     return;
00090 }
00091
00092 int mid = (left + right) / 2 ;
00093
00094 if (index <= mid){
00095
00096     insert_helper_function(node->left_ptr,left,mid,index,data);
00097
00098 }
00099 else {
00100     insert_helper_function(node->right_ptr,mid+1,right,index,data);
00101 }
00102
00103 player_data p1 ;
00104 player_data p2 ;
00105
00106 if(node->left_ptr){
00107     p1 = node->left_ptr->p;
00108 }
00109 if(node->right_ptr){
00110     p2 = node->right_ptr->p;
00111 }
00112 node->p = player_data:: best_player(p1,p2);
00113 }
00114
00115
00116 player_data query_helper_function(tree_node* &node,int start_interval,int end_interval){
00117
00118     if(!node){
00119         return player_data();
00120     }
00121
00122     if(node->left_ptr == 0 && node->right_ptr == 0){
00123
00124         if(node->p.finish_time>=start_interval && node->p.finish_time<=end_interval){
00125
00126             return node->p;
00127         }
00128
00129         else {return player_data();}
00130
00131     }
00132
00133     player_data left =  query_helper_function(node->left_ptr,start_interval,end_interval);
00134
00135     player_data right = query_helper_function(node->right_ptr,start_interval,end_interval);
00136
00137     return player_data :: best_player(left,right);
00138 }
00139
00140
00141
00142
00143
00144
00145 bool update(tree_node* &node ,int left,int right ,player_data new_data) {
00146
00147     if(!node) {
00148         return false ;
00149     }
00150
00151     if(left == right && left == new_data.player_id){
00152
00153         node->p = new_data;
00154         return true ;
00155     }
00156
00157     bool check = false;
00158     int mid = (left + right) / 2 ;
00159
00160     if(new_data.player_id <= mid && node->left_ptr){
00161         check = update(node->left_ptr,left,mid,new_data);
00162     }
00163
00164     else if(new_data.player_id>mid && node->right_ptr){
00165         check = update(node->right_ptr,mid+1,right,new_data);
00166
00167     }
00168
00169     if (check){
00170         player_data left_player ;
```

```
00171         player_data right_player;
00172
00173         if(node->left_ptr){
00174             left_player = node->left_ptr->p;
00175         }
00176         else {left_player=player_data();}
00177
00178         if(node->right_ptr){
00179             right_player = node->right_ptr->p;
00180         }
00181         else {right_player=player_data();}
00182
00183         node->p = player_data :: best_player (left_player,right_player);
00184
00185
00186     }
00187
00188     return check;
00189
00190
00191
00192 }
00193
00194 player_data query_by_index_helper(tree_node* &node, int left, int right,int start_id, int end_id){
00195
00196 if(!node || end_id<left || start_id>right){return player_data();}
00197
00198 if(start_id <=left && right <= end_id){
00199     return node->p;
00200 }
00201
00202
00203 int mid = (left + right) / 2;
00204 player_data left_player = query_by_index_helper(node->left_ptr, left, mid, start_id, end_id);
00205 player_data right_player = query_by_index_helper(node->right_ptr, mid + 1, right, start_id, end_id);
00206
00207 return player_data::best_player(left_player, right_player);
00208
00209 }
00210
00211
00212 public:
00213
00214 segment_tree(int size) :
00215 root_node(0),max_num_of_players(size)
00216 {this->reached_index = -1;} // initialize reached_index to -1
00217
00218 void insert_into_tree(player_data p){
00219
00220     this->reached_index++;
00221     player_data p1(p.score,this->reached_index,p.finish_time);
00222     return insert_helper_function(root_node,0,max_num_of_players-1,this->reached_index,p1);
00223
00224 }
00225 // the tree is created by indices , not time ,thus complexity is close to O(n)
00226 player_data query_the_tree_by_time(int start_time, int finish_time){
00227     if(start_time < 0 || finish_time< 0 ){
00228         cerr«"Wrong Interval";
00229         return player_data();
00230     }
00231     return query_helper_function(root_node,start_time,finish_time);
00232
00233 }
00234 player_data query_the_tree_by_id(int begin_id, int end_id){
00235
00236     if(begin_id < 0 || end_id > this->reached_index){
00237
00238         cerr«"Invalid ID "«endl;
00239         return player_data();
00240     }
00241     else if (begin_id > end_id){
00242
00243         cerr«"Invalid Interval"«endl;
00244         return player_data();
00245     }
00246
00247     return query_by_index_helper(root_node, 0, max_num_of_players - 1, begin_id, end_id);
00248
00249 }
00250
00251 bool update_player_data(player_data new_player_data){
00252
00253     bool check = false;
00254
00255     if(new_player_data.player_id < 0 || new_player_data.player_id > this->reached_index){
00256         cerr«"Invalid Player ID"«endl;
00257         return false;
```

```
00258     }
00259
00260     else{
00261
00262         player_data p1(new_player_data.score,new_player_data.player_id,new_player_data.finish_time);
00263          check = update(root_node,0,max_num_of_players-1,p1);
00264         return check;
00265     }
00266     return check;
00267 }
00268
00269 };
00270
00271 //int segment_tree :: reached_index = -1;
00272
```

Index