Wenlan Tian

COP 3530

Section 1087, MAEB 211

10/16/2014

Homework 5

"On my honor, I have neither given nor received unauthorized aid in doing this assignment.”

Summary of learning experience:

The easiest part of the task is to understand the theory of a winner tree or loser tree.

The most difficult parts of the task: how to implement these trees, especially the “eliminate” part.

How well I think I achieved them: I’m unable to finish the program because i do not know how to implement it in c++, especially with the specific requirements of update\_player(), eliminate\_player(int player) and k-way merge because I don’t fully understand the requirements of them.

1. Does the program compile without errors?

I didn’t finish the program.

2. Does the program compile without warnings?

3. Does the program run without crashing?

4. Describe how you tested the program.

5. Describe the ways in which the program does not meet assignment’s specifications.

6. Describe all known and suspected bugs.

7. Does the program run correctly?

Wenlan Tian

COP 3530

Section 1087, MAEB 211

10/16/2014

Homework 5

Homework 5:

Implement a winner tree and loser tree.

#include <iostream>

template <class T>

class WinnerTree{

private:

T\* first;

int size;

T elimiate;

T[size];

public:

WinnerTree();

T winner(); //returns the winning player

void update\_player(int, int);

void eliminate\_player(int);

int size(); //returns the number of (non-eliminated) players

bool empty(); //returns false IFF there are any (non-eliminated) players

void pretty\_print( ostream& ); //prints a textual representation of the tree following the examples below.

};

template <class T>

class LoserTree{

private:

public:

LoserTree();

T winner(); //returns the winning player

void update\_winner( int new\_score );

void eliminate\_winner();

int size(); //returns the number of (non-eliminated) players

bool empty(); //returns false IFF there are any (non-eliminated) players

void pretty\_print( ostream& ); //prints a textual representation of the tree following the examples below.

};

//////////////////////////////////////WinnerTree//////////////////////////////

template <class T>

WinnerTree<T>::WinnerTree(){

}

//returns the winning player

template <class T>

T WinnerTree<T>::winner(){

int p1, p2;

return (T[p1] > T[p2] ? p1 : p2);

}

template <class T>

void WinnerTree<T>::update\_player( int player, int new\_score ){

T[player] = new\_score;

}

template <class T>

void WinnerTree<T>::eliminate\_player( int player ){

}

//returns the number of (non-eliminated) players

template <class T>

int WinnerTree<T>::size(){

}

//returns false IFF there are any (non-eliminated) players

template <class T>

bool WinnerTree<T>::empty(){

}

//prints a textual representation of the tree following the examples below.

template <class T>

void WinnerTree<T>::pretty\_print( ostream& ){

}

//////////////////////////////////////LoserTree//////////////////////////////

template <class T>

LoserTree<T>::LoserTree(){

}

//returns the winning player

template <class T>

T LoserTree<T>:: winner(){

int p1, p2;

return (T[p1] > T[p2] ? p1 : p2);

}

template <class T>

void LoserTree<T>::update\_player( int player, int new\_score ){

T[player] = new\_score;

}

template <class T>

void LoserTree<T>::eliminate\_player( int player ){

}

//returns the number of (non-eliminated) players

template <class T>

int LoserTree<T>::size(){

}

//returns false IFF there are any (non-eliminated) players

template <class T>

bool LoserTree<T>::empty(){

}

//prints a textual representation of the tree following the examples below.

template <class T>

void LoserTree<T>::pretty\_print( ostream& ){

}