FProjectLBandDP

0.3.0

Generated by Doxygen 1.8.17

1 Class Index 1

1.1 Class List . . . 1

2 File Index 3

2.1 File List . . . 3

3 Class Documentation 5

3.1 BTNode Class Reference . . . 5

3.1.1 Detailed Description . . . 5

3.1.2 Constructor & Destructor Documentation . . . 6

3.1.2.1 BTNode() . . . 6

3.1.3 Member Function Documentation . . . 6

3.1.3.1 nodeData() . . . 6

3.1.3.2 nodeName() . . . 6

3.1.3.3 nodeRatio() . . . 7

3.1.4 Member Data Documentation . . . 7

3.1.4.1 left . . . 7

3.1.4.2 parent . . . 7

3.1.4.3 right . . . 7

3.2 Products Class Reference . . . 7

3.2.1 Detailed Description . . . 8

3.2.2 Constructor & Destructor Documentation . . . 8

3.2.2.1 Products() [1/2] . . . 8

3.2.2.2 Products() [2/2] . . . 8

3.2.3 Member Data Documentation . . . 8

3.2.3.1 price . . . 9

3.2.3.2 ratio . . . 9

3.2.3.3 weight . . . 9

4 File Documentation 11

4.1 /home/lee/Leecmake/CPTR227FinalProject/src/main.cpp File Reference . . . 11

4.1.1 Detailed Description . . . 12

4.1.2 Function Documentation . . . 12

4.1.2.1 addNode() . . . 12

4.1.2.2 addNodeTree() . . . 13

4.1.2.3 comparator() . . . 13

4.1.2.4 createTree() . . . 14

4.1.2.5 createTreeBruteForce() . . . 14

4.1.2.6 genProducts() . . . 15

4.1.2.7 main() . . . 15

4.1.2.8 printBT() [1/2] . . . 16

4.1.2.9 printBT() [2/2] . . . 16

4.1.2.10 printTree() . . . 16

4.1.2.11 randomGen() 17

Index **19**

Chapter 1

Class Index

1.1 Class List

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

BTNode 5
Products 7

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

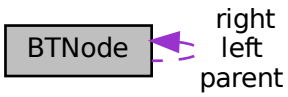
/home/lee/Leecmake/CPTR227FinalProject/src/main.cpp
This is the final project made with code from HW11 11

Chapter 3

Class Documentation

3.1 BTNode Class Reference

Collaboration diagram for BTNode:



Public Member Functions

- BTNode (Products dataVal)
- char nodeName ()
- Products nodeData ()
- int nodeRatio ()

Public Attributes

- BTNode * left
- BTNode * right
- BTNode * parent

3.1.1 Detailed Description

Definition at line 48 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BTNode()

```
BTNode::BTNode (
            Products dataVal )  [inline]
```

BTNode constructor

Parameters

dataVal	This is the product that is put into the binary tree.

Definition at line 59 of file main.cpp.

```
59                              {
60       //cout « "name = " « name « endl;
61       left = NULL;
62       right = NULL;
63       parent = NULL;
64       objName = name++;
65       data = dataVal;
66   }
```

3.1.3 Member Function Documentation

3.1.3.1 nodeData()

```
Products BTNode::nodeData ( )  [inline]
```

This reports the node's data

Definition at line 78 of file main.cpp.

```
78                          {
79         return(data);
80     }
```

3.1.3.2 nodeName()

```
char BTNode::nodeName ( )  [inline]
```

This reports the node's name

Definition at line 71 of file main.cpp.

```
71                      {
72         return(objName);
73     }
```

3.1.3.3 nodeRatio()

```
int BTNode::nodeRatio ( )  [inline]
```

This reports the node's ratio

Definition at line 85 of file main.cpp.

```
85                      {
86          return(data.ratio);
87      }
```

3.1.4 Member Data Documentation

3.1.4.1 left

```
BTNode* BTNode::left
```

Definition at line 50 of file main.cpp.

3.1.4.2 parent

```
BTNode* BTNode::parent
```

Definition at line 52 of file main.cpp.

3.1.4.3 right

```
BTNode* BTNode::right
```

Definition at line 51 of file main.cpp.

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

- /home/lee/Leecmake/CPTR227FinalProject/src/main.cpp

3.2 Products Class Reference

Public Member Functions

- Products ()
- Products (double p, double w)

Public Attributes

- double price
- double weight
- double ratio

3.2.1 Detailed Description

This is class has 2 different parameters used to make this object

Definition at line 22 of file main.cpp.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Products() [1/2]

```
Products::Products ( )  [inline]
```

Definition at line 31 of file main.cpp.

```
31                {
32
33     }
```

3.2.2.2 Products() [2/2]

```
Products::Products (
            double p,
            double w )  [inline]
```

This is the constructor for this class

Parameters

p	The price for the product.
w	The weight for the product.

Definition at line 41 of file main.cpp.

```
41                                  {
42         price = p;
43         weight = w;
44         ratio = w/p;
45     }
```

3.2.3 Member Data Documentation

3.2.3.1 price

```
double Products::price
```

Definition at line 27 of file main.cpp.

3.2.3.2 ratio

```
double Products::ratio
```

Definition at line 29 of file main.cpp.

3.2.3.3 weight

```
double Products::weight
```

Definition at line 28 of file main.cpp.

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

- /home/lee/Leecmake/CPTR227FinalProject/src/main.cpp

Chapter 4

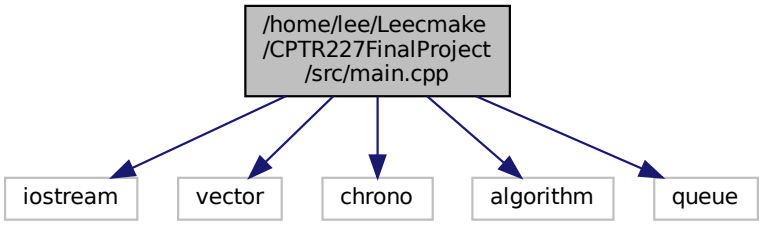
File Documentation

4.1 /home/lee/Leecmake/CPTR227FinalProject/src/main.cpp File Reference

This is the final project made with code from HW11.

```
#include <iostream>
#include <vector>
#include <chrono>
#include <algorithm>
#include <queue>
```

Include dependency graph for main.cpp:



Classes

- class Products
- class BTNode

Functions

- BTNode * addNodeTree (BTNode *rootNode, BTNode *n)
- BTNode * addNode (BTNode *rootNode, Products dataval)
- int randomGen (int min, int max)
- std::vector< Products > genProducts (int n)
- void printTree (BTNode *rootNode)
- void printBT (const string &prefix, BTNode *node, bool isLeft)
- void printBT (BTNode *node)
- void createTreeBruteForce (vector< Products > &tree, int index)
- void createTree (vector< Products > &tree, int index)
- bool comparator (const Products &a, const Products &b)
- int main (int, char **)

4.1.1 Detailed Description

This is the final project made with code from HW11.

This program is based on the knapsack problem and uses a binary tree to store the data.

Author

Daniel Pervis and Lee Beckermeyer

Date

4/21/2021

4.1.2 Function Documentation

4.1.2.1 addNode()

```
BTNode* addNode (
            BTNode * rootNode,
            Products dataval )
```

Adds a new node with the passed data value

Parameters

rootNode	pointer to root node
dataval	an integer for the new node's data

Returns

pointer to root node or NULL if not successful

Definition at line 140 of file main.cpp.

```
140                                                 {
141     BTNode* newNode = new BTNode(dataval);
142     if(addNodeTree(rootNode, newNode) == NULL) {
143         //cout « dataval « " already in tree" « endl;
144     } else {
145         //cout « dataval « " succesfully added" « endl;
146     }
147     return(rootNode);
148 }
```

4.1.2.2 addNodeTree()

```
BTNode* addNodeTree (
            BTNode * rootNode,
            BTNode * n )
```

This function adds a node to a binary search tree.

Parameters

rootNode	is the pointer to the tree's root node
n	is the node to add

Returns

pointer to rootNode if successful, NULL otherwise

Definition at line 105 of file main.cpp.

```
105                                                 {
106     BTNode* prev = NULL;
107     BTNode* w = rootNode;
108     if(rootNode == NULL) { // starting an empty tree
109         rootNode = n;
110     } else {
111         // Find the node n belongs under, prev, n's new parent
112         while(w != NULL) {
113             prev = w;
114             if(n->nodeRatio() < w->nodeRatio()){
115                 w = w->left;
116             } else if(n->nodeRatio() > w->nodeRatio()) {
117                 w = w->right;
118             } else { // data already in the tree
119                 return(NULL);
120             }
121         }
122         // now prev should contain the node that should be n's parent
123         // Add n to prev
124         if(n->nodeRatio() < prev->nodeRatio()) {
125             prev->left = n;
126         } else {
127             prev->right = n;
128         }
129     }
130     return(rootNode);
131 }
```

4.1.2.3 comparator()

```
bool comparator (
            const Products & a,
            const Products & b )
```

compares 2 products, currently not used.

Parameters

a	product a
b	product b

Definition at line 298 of file main.cpp.

```
298                                                     {
299     return a.ratio > b.ratio;
300 }
```

4.1.2.4 createTree()

```
void createTree (
            vector< Products > & tree,
            int index )
```

creates a binary tree

Parameters

tree	a vector of products you want to turn into a tree.
index	the size of the vector, needed with the current implementation.

Definition at line 281 of file main.cpp.

```
281                                                     {
282     BTNode* root = new BTNode(tree[index]);
283     int weight = 0;
284     int n = 0;
285     for (Products x : tree){
286         addNode(root, x);
287     };
288     cout « "Weight of Knapsack: " « weight « endl;
289     printBT(root);
290 }
```

4.1.2.5 createTreeBruteForce()

```
void createTreeBruteForce (
            vector< Products > & tree,
            int index )
```

creates a binary tree, also checks if the knapsack is full, if the knapsack isn't full it continues until the end of the vector.

Parameters

tree	a vector of products you want to turn into a tree.
index	the size of the vector, needed with the current implementation.

Definition at line 253 of file main.cpp.

```
253                                                                 {
254     BTNode* root = new BTNode(tree[index]);
255     int weight = 0;
256     int n = 0;
257     for (Products x : tree){
258             n++;
259             int newweight = x.weight + weight;
260             if(newweight>=500 or n == index){
261                 continue;
262             }
263             else{
264                 weight = newweight;
265                 addNode(root, x);
266                 x.weight + weight;
267             }
268
269
270         };
271     cout « "Weight of Knapsack: " « weight « endl;
272     printBT(root);
273 };
```

4.1.2.6 genProducts()

```
std::vector<Products> genProducts (
              int n )
```

generates the products.

Parameters

n	The amount of products you want generated.

Definition at line 169 of file main.cpp.

```
169                                         {
170     vector<Products> output;
171     for (int i = 0; i < n; i++) {
172         output.push_back(Products(randomGen(1,1000), randomGen(5, 200)));
173     }
174     return output;
175 }
```

4.1.2.7 main()

```
int main (
              int ,
              char **  )
```

Definition at line 302 of file main.cpp.

```
302                       {
303     srand(time(NULL));
304     vector<Products> products = genProducts(50);
305     auto max = std::max_element(products.begin(), products.end(), [](const Products& a, const Products&
       b){
306         return a.ratio < b.ratio;
307     });
308     int index = distance(max, products.end());
309     cout « max->ratio « endl;
310     sort(products.begin(), products.end(), &comparator);
311     for (int i = 1; i < products.size(); i++) {
312         cout « i « " : " « products[i].ratio « endl;
313     }
314     createTreeBruteForce(products, index);
315 }
```

4.1.2.8 printBT() [1/2]

```
void printBT (
            BTNode * node )
```

An overload to simplify calling printBT

Parameters

node	is the root node of the tree to be printed

Definition at line 242 of file main.cpp.

```
243 {
244     printBT("", node, false);
245 }
```

4.1.2.9 printBT() [2/2]

```
void printBT (
            const string & prefix,
            BTNode * node,
            bool isLeft )
```

Print a binary tree

This example is modified from: https://stackoverflow.com/a/51730733

Parameters

prefix	is a string of characters to start the line with
node	is the current node being printed
isLeft	bool true if the node is a left node

Definition at line 218 of file main.cpp.

```
219 {
220     if( node != NULL )
221     {
222         cout « prefix;
223 
224         cout « (isLeft ? "L--" : "R--" );
225 
226         // print the value of the node
227         //cout « node->nodeName() « ':' « node->nodeData() « std::endl;
228         cout « node->nodeData().ratio « std::endl;
229 
230         // enter the next tree level - left and right branch
231         printBT( prefix + (isLeft ? "|   " : "    "), node->left, true);
232         printBT( prefix + (isLeft ? "|   " : "    "), node->right, false);
233     }
234 }
```

4.1.2.10 printTree()

```
void printTree (
            BTNode * rootNode )
```

prints a binary tree

Parameters

rootNode	The binary tree you want printed.

Definition at line 182 of file main.cpp.

```
182                                 {
183     queue<BTNode*> todo; // the queue of nodes left to visit
184     BTNode* cur; // current node
185     BTNode* prev; // The previous node
186 
187     todo.push(rootNode);
188 
189     while(!todo.empty()) {
190         cur = todo.front();
191         // Print current node
192         cout « cur->nodeName() « ':' « cur->nodeData().ratio « '\t';
193         // add cur->left to queue
194         if(cur->left != NULL) {
195             todo.push(cur->left);
196         }
197         // add cur->right to queue
198         if(cur->right != NULL) {
199             todo.push(cur->right);
200         }
201         // remove cur from queue
202         todo.pop();
203     }
204     cout « endl;
205 }
```

4.1.2.11 randomGen()

```
int randomGen (
              int min,
              int max )
```

Randomly generates a "double"(float in C++) number

Parameters

min	The minimum number that can be generated.
max	The maximum number that can be generated.

Definition at line 157 of file main.cpp.

```
157                                 {
158 
159     double random = rand() % max + min;
160     //cout « rand() % max « endl;
161     return random;
162 }
```

Index

/home/lee/Leecmake/CPTR227FinalProject/src/main.cpp, 11

addNode
    main.cpp, 12
addNodeTree
    main.cpp, 13

BTNode, 5
    BTNode, 6
    left, 7
    nodeData, 6
    nodeName, 6
    nodeRatio, 6
    parent, 7
    right, 7

comparator
    main.cpp, 13
createTree
    main.cpp, 14
createTreeBruteForce
    main.cpp, 14

genProducts
    main.cpp, 15

left
    BTNode, 7

main
    main.cpp, 15
main.cpp
    addNode, 12
    addNodeTree, 13
    comparator, 13
    createTree, 14
    createTreeBruteForce, 14
    genProducts, 15
    main, 15
    printBT, 15, 16
    printTree, 16
    randomGen, 17

nodeData
    BTNode, 6
nodeName
    BTNode, 6
nodeRatio
    BTNode, 6

parent
    BTNode, 7
price
    Products, 8
printBT
    main.cpp, 15, 16
printTree
    main.cpp, 16
Products, 7
    price, 8
    Products, 8
    ratio, 9
    weight, 9

randomGen
    main.cpp, 17
ratio
    Products, 9
right
    BTNode, 7

weight
    Products, 9