FProjectLBandDP

0.3.0

Generated by Doxygen 1.8.17

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

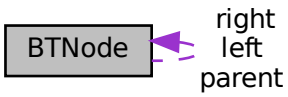
Here is a list of all files with brief descriptions:

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, currently breaks something by converting it to an int, don't use for comparisons.

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

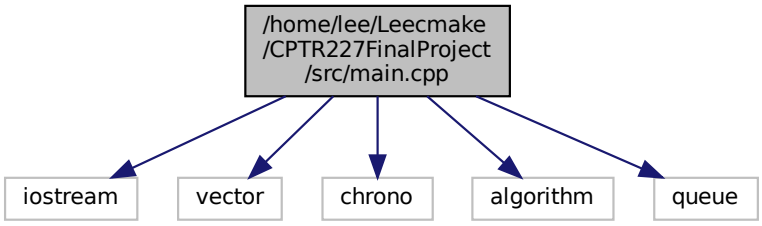
File Documentation

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 * addNode (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 RandomTree (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() [1/2]

```
BTNode* addNode (
            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->nodeData().ratio < w->nodeData().ratio){
115                 //cout << w->nodeData().ratio << " added" << endl;
116                 w = w->left;
117             } else if(n->nodeData().ratio > w->nodeData().ratio) {
118                 //cout << w->nodeData().ratio << " added" << endl;
119                 w = w->right;
120             } else { // data already in the tree
121                 return(NULL);
122             }
123         }
124         // now prev should contain the node that should be n's parent
125         // Add n to prev
126         if(n->nodeData().ratio < prev->nodeData().ratio) {
127             prev->left = n;
128         } else {
129             prev->right = n;
130         }
131     }
132     return(rootNode);
133 }
```

4.1.2.2 addNode() [2/2]

```
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 142 of file main.cpp.

```
142                                                 {
143     BTNode* newNode = new BTNode(dataval);
144     if(addNode(rootNode, newNode) == NULL) {
145         //cout << dataval.ratio << " already in tree" << endl;
146     } else {
147         //cout << dataval.ratio << " succesfully added" << endl;
148     }
149     return(rootNode);
150 }
```

4.1.2.3 comparator()

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

compares 2 products

Parameters

a	product a
b	product b

Definition at line 335 of file main.cpp.

```
335                                                 {
336     return a.ratio > b.ratio;
337 }
```

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 321 of file main.cpp.

```
321                                                 {
322     BTNode* root = new BTNode(tree[index]);
323     for (Products x : tree){
324         addNode(root, x);
325     }
326     printBT(root);
327 }
```

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 256 of file main.cpp.

```
256                                                                 {
257     BTNode* root = new BTNode(tree[index]);
258     int weight = 0;
259     int price = 0;
260     for (Products x : tree){
261             int newweight = x.weight + weight;
262             int newprice = x.price + price;
263             if(newweight>=500){
264                 continue;
265             }
266             else{
267                 weight = newweight;
268                 price = newprice;
269                 addNode(root, x);
270                 x.weight + weight;
271                 x.price + price;
272             }
273 
274 
275         };
276     cout « "Tree generated using a brute force algorithm after sorting the object's ratios" « endl;
277     cout « "Weight of the Knapsack: " « weight « " lbs" « endl;
278     cout « "Price of the Knapsack: " « price « "$" « endl;
279     printBT(root);
280 };
```

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 172 of file main.cpp.

```
172                                     {
173     vector<Products> output;
174     for (int i = 0; i < n; i++) {
175         output.push_back(Products(randomGen(1,1000), randomGen(5, 200)));
176     }
177     return output;
178 }
```

4.1.2.7 main()

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

Definition at line 339 of file main.cpp.

```
339                     {
340     srand(time(NULL));
341     vector<Products> products = genProducts(50);
342     auto max = std::max_element(products.begin(), products.end(), [](const Products& a, const Products&
        b){
343         return a.ratio < b.ratio;
344     });
345     int index = distance(max, products.end());
346     cout « max->ratio « endl;
347     sort(products.begin(), products.end(), &comparator);
348     for (int i = 1; i < products.size(); i++) {
349         cout « i « " : " « products[i].ratio « endl;
350     }
351
352     for (Products x : products){
353         cout « x.ratio « endl;
354     }
355
356     createTreeBruteForce(products, index);
357     RandomTree(products, index);
358 }
```

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 245 of file main.cpp.

```
246 {
247     printBT("", node, false);
248 }
```

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 221 of file main.cpp.

```
222 {
223     if( node != NULL )
224     {
225         cout « prefix;
226 
227         cout « (isLeft ? "L--" : "R--" );
228 
229         // print the value of the node
230         //cout « node->nodeName() « ':' « node->nodeData() « std::endl;
231         cout « node->nodeData().ratio « std::endl;
232 
233         // enter the next tree level - left and right branch
234         printBT( prefix + (isLeft ? "|   " : "    "), node->left, true);
235         printBT( prefix + (isLeft ? "|   " : "    "), node->right, false);
236     }
237 }
```

4.1.2.10 printTree()

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

prints a binary tree

Parameters

rootNode	The binary tree you want printed.

Definition at line 185 of file main.cpp.

```
185                                 {
186     queue<BTNode*> todo; // the queue of nodes left to visit
187     BTNode* cur; // current node
188     BTNode* prev; // The previous node
189 
190     todo.push(rootNode);
191 
192     while(!todo.empty()) {
193         cur = todo.front();
194         // Print current node
195         cout « cur->nodeName() « ':' « cur->nodeData().ratio « '\t';
196         // add cur->left to queue
197         if(cur->left != NULL) {
198             todo.push(cur->left);
199         }
200         // add cur->right to queue
201         if(cur->right != NULL) {
202             todo.push(cur->right);
203         }
204         // remove cur from queue
205         todo.pop();
206     }
207     cout « endl;
208 }
```

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 159 of file main.cpp.

```
159                                 {
160
161     double random = rand() % max + min;
162      //int random = rand() % max + min;
163      //cout « rand() % max « endl;
164      return random;
165 }
```

4.1.2.12 RandomTree()

```
void RandomTree (
            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 288 of file main.cpp.

```
288                                                 {
289     BTNode* root = new BTNode(tree[index]);
290     int weight = 0;
291     int price = 0;
292     int n = 0;
293     while(n < 10){
294         n++;
295         Products x = tree[randomGen(0,index)];
296         int newweight = x.weight + weight;
297         int newprice = x.price + price;
298         if(newweight>=500){
299                     continue;
300                 }
301                 else{
302                     weight = newweight;
303                     price = newprice;
304                     addNode(root, x);
305                     x.weight + weight;
306                     x.price + price;
307                 }
308     }
309     cout « "Tree generated using a random algorithm" « endl;
310     cout « "Weight of the Knapsack: " « weight « " lbs" « endl;
311     cout « "Price of the Knapsack: " « price « "$" « endl;
312     printBT(root);
313 };
```

Index