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

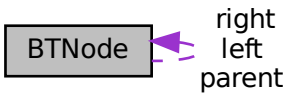
Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 BTNode Class Reference

Collaboration diagram for BTNode:



Public Member Functions

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

Public Attributes

- BTNode * left
- BTNode * right
- BTNode * parent
- int weight
- int price

3.1.1 Detailed Description

Binary Tree Node

This is from Open Data Structures in C++ by Pat Morin

Definition at line 23 of file main.cpp.

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BTNode()

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

BTNode constructor

Definition at line 35 of file main.cpp.

```
35                          {
36       //cout « "name = " « name « endl;
37       left = NULL;
38       right = NULL;
39       parent = NULL;
40       objName = name++;
41       data = dataVal;
42   }
```

3.1.3 Member Function Documentation

3.1.3.1 nodeData()

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

This reports the node's data

Definition at line 54 of file main.cpp.

```
54                      {
55         return(data);
56     }
```

3.1.3.2 nodeName()

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

This reports the node's name

Definition at line 47 of file main.cpp.

```
47                       {
48         return(objName);
49     }
```

3.1.4 Member Data Documentation

3.1.4.1 left

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

Definition at line 25 of file main.cpp.

3.1.4.2 parent

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

Definition at line 27 of file main.cpp.

3.1.4.3 price

```
int BTNode::price
```

Definition at line 29 of file main.cpp.

3.1.4.4 right

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

Definition at line 26 of file main.cpp.

3.1.4.5 weight

```
int BTNode::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

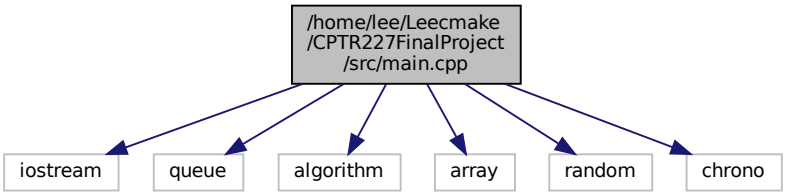
File Documentation

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

This is an implementation of the knapsack problem.

```
#include <iostream>
#include <queue>
#include <algorithm>
#include <array>
#include <random>
#include <chrono>
```

Include dependency graph for main.cpp:



Classes

- class BTNode

Functions

- BTNode * addNode (BTNode *rootNode, BTNode *n)
- BTNode * addNode (BTNode *rootNode, int dataval)
- BTNode * genExampleTree (BTNode *root)
- BTNode * genTree (vector< int > Supplier)

- void printTree (BTNode *rootNode)
- void printBT (const string &prefix, BTNode *node, bool isLeft)
- void printBT (BTNode *node)
- int height (BTNode *u)
- BTNode * randTreeGen (int n)
- int randTreeTest (int m, int n)
- int main (int, char **)

4.1.1 Detailed Description

This is an implementation of the knapsack problem.

This is an implementation of the knapsack problem, using a binary tree to store the data.

Author

Seth McNeill

Date

1/28/2021

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 74 of file main.cpp.

```
74                                          {
75     BTNode* prev = NULL;
76     BTNode* w = rootNode;
77     if(rootNode == NULL) { // starting an empty tree
78         rootNode = n;
79     } else {
```

```
80          // Find the node n belongs under, prev, n's new parent
81          while(w != NULL) {
82              prev = w;
83              if(n->nodeData() < w->nodeData()){
84                  w = w->left;
85              } else if(n->nodeData() > w->nodeData()) {
86                  w = w->right;
87              } else { // data already in the tree
88                  return(NULL);
89              }
90          }
91          // now prev should contain the node that should be n's parent
92          // Add n to prev
93          if(n->nodeData() < prev->nodeData()) {
94              prev->left = n;
95          } else {
96              prev->right = n;
97          }
98      }
99      return(rootNode);
100 }
```

4.1.2.2 addNode() [2/2]

```
BTNode* addNode (
              BTNode * rootNode,
              int 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 110 of file main.cpp.

```
110                                             {
111     BTNode* newNode = new BTNode(dataval);
112     if(addNode(rootNode, newNode) == NULL) {
113         //cout « dataval « " already in tree" « endl;
114     } else {
115         //cout « dataval « " succesfully added" « endl;
116     }
117     return(rootNode);
118 }
```

4.1.2.3 genExampleTree()

```
BTNode* genExampleTree (
              BTNode * root )
```

This generates a simple tree to play with

It is a bit of a hack.

Definition at line 125 of file main.cpp.

```
125                                     {
126     //int inData[] = {1,2,3,4,5,6,7};
127     int inData[] = {4,6,5,7,2,1,3};
128     int classData[] = {1,3,4,5,6,7,8,9,11,12,13,14};
129     for(int ii = 0; ii < 7; ii++) {
130         addNode(root, inData[ii]);
131     }
132     return root;
133 }
```

4.1.2.4 genTree()

```
BTNode* genTree (
              vector< int > Supplier )
```

This is a tree generated using a vector.

Definition at line 138 of file main.cpp.

```
138                                     {
139     BTNode* newNode = new BTNode(0);
140     for (int x : Supplier)
141             addNode(newNode, x);
142     cout « endl;
143     return newNode;
144
145 };
```

4.1.2.5 height()

```
int height (
              BTNode * u )
```

This calculates the height (max number of steps until leaf node)

Parameters

pointer	to a BTNode

Returns

integer count of height

Definition at line 221 of file main.cpp.

```
221                         {
222     if (u == NULL) {
223         return(-1);
224     }
225     return(1 + max(height(u->left), height(u->right)));
226 }
```

4.1.2.6 main()

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

Definition at line 294 of file main.cpp.

```
294                       {
295     randTreeTest(100,26);
296 }
```

4.1.2.7 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 210 of file main.cpp.

```
211 {
212     printBT("", node, false);
213 }
```

4.1.2.8 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 187 of file main.cpp.

```
188 {
189     if( node != NULL )
190     {
191         cout « prefix;
192
```

```
193         cout « (isLeft ? "L--" : "R--" );
194
195         // print the value of the node
196         //cout « node->nodeName() « ':' « node->nodeData() « std::endl;
197         cout « node->nodeData() « std::endl;
198
199         // enter the next tree level - left and right branch
200         printBT( prefix + (isLeft ? "|   " : "    "), node->left, true);
201         printBT( prefix + (isLeft ? "|   " : "    "), node->right, false);
202     }
203 }
```

4.1.2.9 printTree()

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

Prints out a representtation of a binary search tree

Parameters

rootNode	is a pointer to the root node

Definition at line 152 of file main.cpp.

```
152                                 {
153     queue<BTNode*> todo; // the queue of nodes left to visit
154     BTNode* cur; // current node
155     BTNode* prev; // The previous node
156
157     todo.push(rootNode);
158
159     while(!todo.empty()) {
160         cur = todo.front();
161         // Print current node
162         cout « cur->nodeName() « ':' « cur->nodeData() « '\t';
163         // add cur->left to queue
164         if(cur->left != NULL) {
165             todo.push(cur->left);
166         }
167         // add cur->right to queue
168         if(cur->right != NULL) {
169             todo.push(cur->right);
170         }
171         // remove cur from queue
172         todo.pop();
173     }
174     cout « endl;
175 }
```

4.1.2.10 randTreeGen()

```
BTNode* randTreeGen (
            int n )
```

randTreeGen takes a integer, n and makes a BT tree from 1 to n that is randomly shuffled.

Parameters

n	refers to the amount of data in those trees from 1 to n

Definition at line 232 of file main.cpp.

```
232                             {
233     int arr[n];
234     for(int k = 1; k < n+1; k++){
235         arr[k-1] = k;
236     };
237     int size = sizeof(arr) / sizeof(arr[0]); //gets size of array
238 
239     vector<int> Vect(arr, arr + n);
240 
241     unsigned seed = std::chrono::system_clock::now().time_since_epoch().count(); //gets a seed based on
       time to shuffle Vect by
242     shuffle (Vect.begin(), Vect.end(), std::default_random_engine(seed)); //shuffles the vector
243     BTNode* Final = genTree(Vect);
244     //printBT(Final);
245     return(Final);
246 }
```

4.1.2.11 randTreeTest()

```
int randTreeTest (
            int m,
            int n )
```

Takes two integers, generates random binary trees off of them, and returns the minimum height, maximum height, and average height.

Parameters

m	refers to the number of trees you want to make
n	refers to the amount of data in those trees from 1 to n

Definition at line 254 of file main.cpp.

```
254                                 {
255     float total = 0;
256     int min = 0;
257     int max = 0;
258     float a = m;
259     float average = 0.0;
260     BTNode* minTree = new BTNode(0);
261     for(int x = 0; x < m; x++) {
262        BTNode* testNode = randTreeGen(n);
263        int hi = height(testNode);
264        //cout << "height: " << hi << endl;
265        if(min == 0){
266            min = hi;
267            max = hi;
268            minTree = testNode;
269            //cout << "test min " << min << endl;
270        }
271        else if(hi > max)
272        {
273            max = hi;
274            //cout << " test max" << endl;
275        }
276        else if (hi < min)
277        {
278            min = hi;
279            minTree = testNode;
280            //cout << " test min" << endl;
281        }
282        total += hi;
283        cout << "tree " << x+1 << " has been successfully generated." << endl;
284     }
285     average = total/a;
286     cout << endl;
287     cout << "one of the smallest BT, heightwise: " << endl;
288     printBT(minTree);
289     //cout << "total: " << total << endl;
290     cout << " min: " << min << " max: " << max << " average: " << average << endl;
291     return(0);
292 }
```

Index