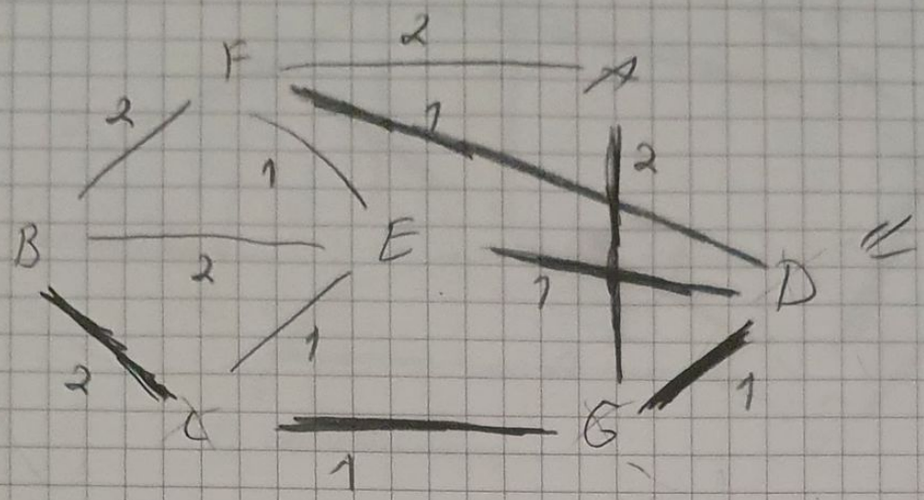
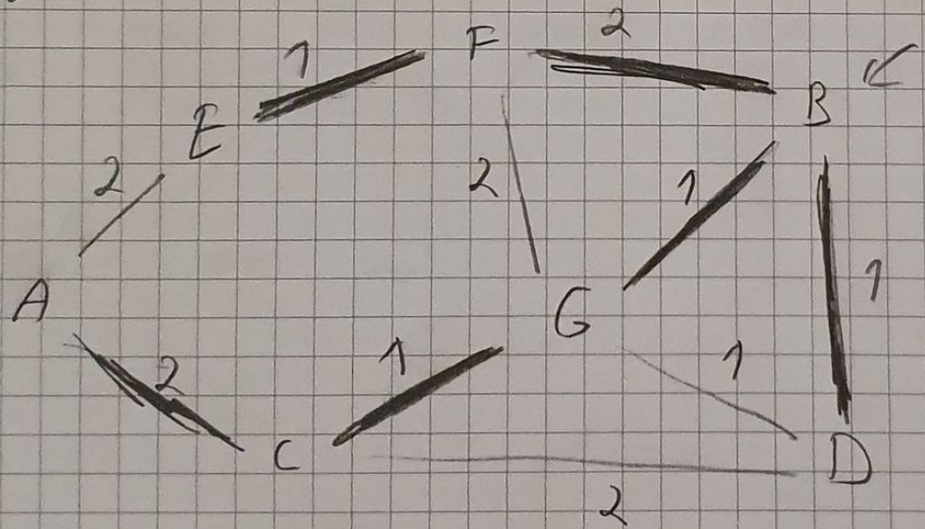


	G 1 d	C 1 g	F 1 d				
	F 1 d	F 1 d	E 1 d	E 1 d			
	E 1 d	E 1 d	B 2 c	B 2 c	B 2 c		
D*		A 2 g	A 2 g	A 2 g	A 2 g	A 2 g	A 2 g



	G 1 G	D 1 G					
	D 1 G	E 2 g	C 2 g	F 2 G	E 3 E		
	F 2 G	F 2 G	F 2 G	A 4 c	A 4 c	A 4 c	
B*							

BD B F B G F E G C C A

```cpp
#include <iostream>          //  cout
#include <string>
using namespace std;
struct Node{
    char ID;
    Node *left, *right;
    Node(const char id, Node * l , Node *r){
      ID= id;
      left = l;
      right = r;
   }
};

Node* gRoot = nullptr; ///< Rotpeker til hele treet
bool gKomplettTre = true,   ///< Er treet komplett eller ei.
    gNivaaOpp = false ; ///< Gått opp ett nivå en gang eller ei.
int gDybde = 0 , ///< Aktuell max.dybde å sjekke noder opp mot.
    gNivaa = -1; ///< Aktuelt nivå node/nullptr er på.

/**
 * 1. traverserer treet fra høyre barn til venstre. dvs høyre barn først
 *    dermed seg selv også venstre barn.(reverse inorder)
 *
 * 2. Finner aller først den høyrenoden lengst til høyre. dvs første node
 *    uten høyre barn helt til høyre. Når denne blir funnet setter den gDybde
 *    til å være gNivaa.(gNivaaOpp må være false for at sjekken skal foretas)
 *
 * 3. Dersom den finner en node på samme nivaa som gDybde og har enten kun
 *    venstre barn eller begge barna.
 *    Vil den nye gDybden være gNivaa +1 og gNivaaOpp vil være true.
 *    (gNivaaOpp må være false for at sjekken skal foretas). Den har nå
 *    funnet maksdybden et komplett tre kan være.
 *
 * Krav for ett fullt tre
 * 4. Noder med gNivaa < gDybde må ha ett fult sett med barn.
 *
 * 5. Dersom gNivaaOpp er true dvs. et komplett tres maks dybde er funnet.
 *    Da må alle noder ha nivå lik eller under gDybde.
 *
 * 6. Dersom en node har ett høyre barn men ingen venstre barn er det ikke et
 *    komplett tre(uansett).
 */
void erKomplettTre(Node * node){
  if(node && gKomplettTre){
    gNivaa++;
    //sjekker om krav for ett komplett tre er brutt
    if((gNivaa < gDybde && !(node->left && node->right))//Er nivået til noden
                                                        //lavere enn gDybde
                                                        //og har den ikke
                                                       //fullt barn?(punkt 4)

        || (gNivaaOpp && gNivaa > gDybde)//Har gDybde gått opp et nivå
                                        //og er nivået til noden større enn
                                        //denne dybden? (punkt 5)
        || (node->right && !node->left)){//Har noden ett høyre barn uten ett
                                         //venstre barn? (punkt 6)
          gKomplettTre = false; //Hvis ja, da er det ikke et komplett tre.
          return; //De neste linjene vil ikke kjøres.
    }
```

```
59      erKomplettTre(node->right);//kaller seg selv rekursivt med høyre barn.
60                                 //Høyre barn lengst til høyre vil da bli funnet
61                                 //først. (punkt 1)
62      // leter etter maks dybde
63      if(!gNivaaOpp && !node->right){//Har gdybden ikke gått opp et nivå en
64                                //gang før og er det ingen høyre barn? (punkt 2)
65        gDybde = gNivaa;//Ja, da har høyrebarn lengst til høyre blitt funnet.
66      }
67      //Er gNivaaOpp false og noden sitt nivå det samme som dybden?
68      //Har noden minst ett nodebarn? (tilfellet der noden kun har
69      //høyre barn er ikke reelt siden det blir fanget opp i punkt 6) (punkt 3)
70      if(!gNivaaOpp && gNivaa == gDybde && (node->left || node->right)){
71        gDybde = gNivaa +1;//setter ny gDybde en mer enn node nivå
72        gNivaaOpp = true;//Denne inkrementering vil skje kun en gang.
73      }
74      erKomplettTre(node->left);
75      gNivaa--;
76    }
77  }
78
79
80  /**
81   * Istedenfor å sjekke barna til noden som metode erKomplettTre
82   * , denne metoden sjekker nivåene til nullptr funnet.
83   * 1. traverserer treet på en inorder måte dvs venstre, seg selv også høyre
84   *
85   * 2. Føste node som er nullptr som blir funnet blir satt til gDybde dersom
86   *   gDybde ikke allerede er satt.
87   *
88   * 3. Dersom det finnes en nullptr på et nivå over gDybde og gNivaaOpp er
89   * false. Da skal gDybde oppdateres til nåværende nivå og gNivaaOpp = true
90   *
91   * Tilfeller for ikke et komplett tre
92   * a1. Dersom det finnes en nullptr som ikke er lik gDybde
93   *    (gDybde blir endret til å dekke alle tilfeller )
94   *
95   */
96  void erKomplettTre2(Node * node){
97    gNivaa++;
98    if(node) { //dersom noden eksisterer?
99     if(gKomplettTre){
100      erKomplettTre2(node->left); //(1)
101      erKomplettTre2(node->right);
102     }
103   } else { // nådd nullptr
104     if(gDybde == 0){ // er gDybde satt ? (2)
105       gDybde = gNivaa;
106     }else if(!gNivaaOpp && gNivaa == gDybde-1 ){//er ett nivå over gDybde (3)
107       gDybde = gNivaa;
108       gNivaaOpp = true;
109     }
110     if(gNivaa != gDybde){ //nivået er ikke lik gDybde
111       gKomplettTre =false;//treet er da ikke komplett (a1)
112     }
113   }
114   gNivaa--;
115 }
116
117 // teste funksjoner
118 /**
```

```
 * Printing a binary tree
 */
//https://stackoverflow.com/questions/36802354/print-binary-tree-in-a-pretty-way-using-c
void printBT(const std::string &prefix, const Node *node, bool isLeft)
{
    if (node != nullptr)
    {
        std::cout << prefix;
        std::cout << (isLeft ? "|--" : "L--");
        // print the value of the node
        std::cout << (int)node->ID << std::endl;
        // enter the next tree level - left and right branch
        printBT(prefix + (isLeft ? "|   " : "    "), node->right, true);
        printBT(prefix + (isLeft ? "|   " : "    "), node->left, false);
    }
}


void printBT(const Node *node)
{
    printBT("", node, false);
}

void nullStill(){
  gKomplettTre = true;
  gNivaaOpp = false;
  gDybde = 0;
  gNivaa = -1;
}

Node* byggCase1Tre(){
  Node *n1 = new Node(1,nullptr,nullptr);
  return n1;
}

Node* byggCase2Tre(){
  Node *n1 = new Node(1,nullptr,nullptr),
       *n2 = new Node(2,nullptr,n1 );

  return n2;
}

Node* byggCase3Tre(){
  Node *n1 = new Node(1,nullptr,nullptr),
       *n2 = new Node(2,n1,nullptr );

  return n2;
}

Node* byggCase4Tre(){
  Node *n1 = new Node(1,nullptr,nullptr),
       *n2 = new Node(2,nullptr,nullptr ),
       *root = new Node(3,n1,n2);
  return root ;
}

Node* byggCase5Tre(){
  Node *n0 = new Node(0,nullptr,nullptr),
        *n1 = new Node(1,n0,nullptr),
```

```
178       *n2 = new Node(2,nullptr,nullptr ),
179       *root = new Node(3,n1,n2);
180   return root ;
181 }
182
183 Node* byggCase6Tre(){
184   Node *n0 = new Node(0,nullptr,nullptr),
185        *n3 = new Node(3, nullptr,nullptr),
186        *n1 = new Node(1,n0,n3),
187       *n2 = new Node(2,nullptr,nullptr ),
188       *root = new Node(3,n1,n2);
189   return root ;
190 }
191
192 Node* byggCase7Tre(){
193
194   Node    *n5 = new  Node(5, nullptr,nullptr),
195     * n2 = new Node(2, n5, nullptr),
196   *n3 = new Node(3, nullptr,nullptr),
197        *n1 = new Node(1,n3,nullptr ),
198       *root = new Node('r',n1,n2);
199   return root ;
200 }
201
202 Node* byggCase8Tre(){
203   Node * n1 = new Node(1,nullptr,nullptr),
204        *n2 = new Node(2,nullptr,nullptr),
205        *n3 = new Node(3,nullptr,nullptr),
206        *n4 = new Node(4, n1,n2),
207        *n5 = new Node(5, n3,nullptr),
208        *n6 = new Node(6, n5,nullptr),
209        *n7 = new Node(7,nullptr,n6),
210        *n8 = new Node(8,n4,n7);
211   return n8;
212 }
213
214 Node* byggCase9Tre(){
215   Node *n1 = new Node(1,nullptr,nullptr),
216       *n5 = new Node(5, nullptr,nullptr),
217       *n2 = new Node(2, n1,nullptr),
218       *n3 = new Node(3, n2, nullptr),
219       *n4 = new Node(4,n3, n5);
220     return n4;
221 }
222
223 Node * byggCase10Tre(){
224   Node * n1 = new Node(1,nullptr,nullptr),
225        *n3 = new Node(3,nullptr,nullptr),
226        *n2 = new Node(2,n1,nullptr),
227        *n4 = new Node(4,n3,n2);
228        return n4;
229 }
230
231 Node* byggCase11Tre(){
232   Node * n1 = new Node(1,nullptr,nullptr),
233       * n2 = new Node(2,nullptr,nullptr),
234       * n3 = new Node(3,nullptr,nullptr) ,
235       * n4 = new Node(4,n1,nullptr),
236       * n5 = new Node(5,n4,n2),
237       * n6 = new Node(6,n3,nullptr),
```

```
238       * n7 = new Node(7,n5,n6);
239   return n7;
240 }
241
242 Node * byggCase12Tre(){
243   Node* n1 = new Node(1,nullptr,nullptr),
244       * n2 = new Node(2,nullptr,nullptr),
245       * n3 = new Node(3,nullptr,nullptr),
246       * n4 = new Node(4,nullptr,nullptr),
247       * n5 = new Node(5,nullptr,nullptr),
248       * n6 = new Node(6,n2,n3),
249       * n7 = new Node(7,n4,n5),
250       * n8 = new Node(8,n6,n7),
251       * n9 = new Node(9,n1,n8);
252     return n9;
253
254 }
255
256 Node * byggCase13Tre(){
257   Node* n6, * n8, * n11a, * n11b, * n12, * n13a, * n13b,
258         * n17, * n28, * n31, * n33, * n34, * n35a, * n35b, * n39, * n72;
259
260     n8 = new Node(8, nullptr, nullptr);
261     n11b = new Node(11, nullptr, nullptr);
262     n13b = new Node(13, nullptr, nullptr);
263     n31 = new Node(31, nullptr, nullptr);
264     n35b = new Node(35, nullptr, nullptr);
265     n34 = new Node(34,n35b, nullptr);
266     n12 = new Node(12, n11b, nullptr);
267     n39 = new Node(39,nullptr, nullptr);
268     n72 = new Node(72, nullptr, nullptr);
269     n6 = new Node(6, n72, nullptr);
270     n13a = new Node(13, n12, nullptr);
271     n28 = new Node(28, nullptr, nullptr);
272     n35a = new Node(35, n34, n39);
273     n11a = new Node(11, n6, n13a);
274     n33 = new Node(33, n28, n35a);
275     n17 = new Node(17, n11a, n33);
276     gRoot = n17;
277   return gRoot;
278 }
279
280 Node * byggCase14Tre(){
281   Node * n1 = new Node(1, nullptr,nullptr),
282        * n2= new Node(2,nullptr,nullptr),
283        * n3= new Node(3,nullptr,nullptr),
284        * n4= new Node(4,nullptr,nullptr),
285        * n5= new Node(5,nullptr,nullptr),
286        * n6= new Node(6,nullptr,nullptr),
287        * n7= new Node(7,nullptr,nullptr),
288        * n8= new Node(8,nullptr,nullptr),
289        * n9= new Node(9,n1,n2),
290        * n10= new Node(10,n3,n4),
291        * n11= new Node(11,n5,n6),
292        * n12= new Node(12,n7,n8),
293        * n13= new Node(13,n9,n10),
294        * n14= new Node(14,n11,n12),
295        * n15= new Node(15,n13,n14);
296     return n15;
297 }
```

```
298
299
300
301 // https://cplusplus.com/forum/beginner/4639/
302 typedef Node* (*AlleByggeFunksjoner)();
303
304 void testFunksjoner(int metodeNr){
305   AlleByggeFunksjoner byggeFunksjoner[] = {
306     byggCase1Tre,
307     byggCase2Tre,
308     byggCase3Tre,
309     byggCase4Tre,
310     byggCase5Tre,
311     byggCase6Tre,
312     byggCase7Tre,
313     byggCase8Tre,
314     byggCase9Tre,
315     byggCase10Tre,
316     byggCase11Tre,
317     byggCase12Tre,
318     byggCase13Tre,
319     byggCase14Tre
320   };
321
322   const int antallCase = sizeof(byggeFunksjoner)
323                          /sizeof(AlleByggeFunksjoner);
324   int sukksessArray[antallCase] = { 1,0,1,1,1, 1,0,0, 0,0,0 ,0,0,1};
325   int antallSukkses = 0;
326   if(metodeNr > 0 && metodeNr < 3 ){
327     cout << "Tester metode " << metodeNr << " \n\n";
328   }
329
330   for(int i = 0; i < antallCase; i ++){
331     gRoot = byggeFunksjoner[i]();
332     switch (metodeNr)
333     {
334     case 1:
335       erKomplettTre(gRoot);
336       cout << "kjorer erKomplettTre" << endl;
337       break;
338     case 2:
339       erKomplettTre2(gRoot);
340       cout << "kjorer erKomplettTre2" << endl;
341       break;
342     default:
343       cout << "Ikke et gyldig metode nummer " << endl;
344       return;
345     }
346     cout << "case " << (i+1) << " tre" << endl;
347     string komplettEllerIkke = sukksessArray[i] ? "Et " : "Ikke et " ;
348     cout << komplettEllerIkke<< "komplett tre " << endl;
349     printBT(gRoot);
350     if(gKomplettTre == sukksessArray[i]){
351       cout << "sukkses" <<endl;
352       antallSukkses++;
353     }  else {
354       cout << "feil" << endl;
355     }
356     cout << " - - - - - - - - - - \n" << endl;
357     nullStill();
```

```
358   }
359
360   cout << "Testet "<< antallCase <<" tilfeller. Metode " << metodeNr << endl;
361   if(antallCase == antallSukkses){
362     cout << "Alle testene var en sukkses. " << endl;
363   } else {
364     cout << antallSukkses << "/" << antallCase
365           << " tester var sukksesfulle." << endl;
366   }
367
368 }
369
370 int main(int argc, char const *argv[])
371 {
372   testFunksjoner(1) ;
373   testFunksjoner(2);
374   return 0;
375 }
376
```