Aufgabe 1

main.cpp: cpp

```cpp
#include "binSearchTree.hpp"
#include <iostream>

int main() {
    binSearchTree t;

    t.insert(5);
    t.insert(6);
    t.insert(7);
    t.insert(4);
    t.insert(2);
    t.insert(1);
    t.insert(3);

    std::cout << "initial" << std::endl;
    t.inorderTreeLike();
    t.remove(5);
    std::cout << "remove 5" << std::endl;
    t.inorderTreeLike();
    t.remove(4);
    std::cout << "remove 4" << std::endl;
    t.inorderTreeLike();
    t.remove(2);
    std::cout << "remove 2" << std::endl;
    t.inorderTreeLike();
    t.remove(1);
    std::cout << "remove 1" << std::endl;
    t.inorderTreeLike();
}
```

binSeachTree.cpp:

```cpp
#include "binSearchTree.hpp"
#include <iostream>

//node implementation
binSearchTree::node::node(int val): val(val) {}

binSearchTree::node::~node() {}

void binSearchTree::node::insert(int val) {
    if (val == this->val)
        return;

    if (val < this->val) {
```

```
        if (left != nullptr)
            left->insert(val);
        else
            left = new node(val);
    } else {
        if (right != nullptr)
            right->insert(val);
        else
            right = new node(val);
    }
}

void binSearchTree::node::swapNextInorder() {
    node* tmp  = this->right;
    node* prev = this;
    while (tmp->left != nullptr) {
        prev = tmp;
        tmp = tmp->left;
    }

    this->val = tmp->val;
    if (prev == this)
        prev->right = tmp->right;
    else
        prev->left = tmp->right;
    delete tmp;
}

void binSearchTree::node::remove(int a, node &prev) {
    if (val == a) {
        if (left == nullptr && right == nullptr) {
            if (prev.left->val == a)
                prev.left = nullptr;
            else
                prev.right = nullptr;
            delete this;
        } else if (left != nullptr && right != nullptr) {
            this->swapNextInorder();
        } else {
            node *tmp;
            if (left == nullptr) {
                tmp = right;
            } else {
                tmp = left;
            }
```

```
            if (prev.left->val == a)
                prev.left = tmp;
            else
                prev.right = tmp;
            delete this;
        }
    } else if (val > a) {
        if (left != nullptr)
            left->remove(a, *this);
    } else if (val < a) {
        if (right != nullptr)
            right->remove(a, *this);
    }
}

void binSearchTree::node::inorderTreeLike(int depth) {
    if (right != nullptr)
        right->inorderTreeLike(depth+1);
    for (int i = 0; i<depth; i++)
        std::cout << "\t";
    std::cout << val << std::endl;
    if (left != nullptr)
        left->inorderTreeLike(depth+1);
}

void binSearchTree::node::inorder() {
    if (left != nullptr)
        left->inorder();
    std::cout << val << " ";
    if (right != nullptr)
        right->inorder();
}

//binSearchTree implementation
binSearchTree::binSearchTree() {}
binSearchTree::~binSearchTree() {
    delete root;
    root = nullptr;
}

void binSearchTree::insert(int val) {
    if (root == nullptr)
        root = new node(val);
    else
        root->insert(val);
}
```

```
void binSearchTree::inorderTreeLike() {
    if (root != nullptr)
        root->inorderTreeLike(0);
}

void binSearchTree::inorder() {
    if (root != nullptr) {
        root->inorder();
        std::cout << std::endl;
    }
}

void binSearchTree::remove(int a) {
    if (root == nullptr)
        return;
    if (root->val == a) {
        root->swapNextInorder();
    } else {
        if (a > root->val)
            root->right->remove(a, *root);
        else
            root->left->remove(a, *root);
    }

}
```

Laufzeitanalyse beim Löschen

```cpp
...
void binSearchTree::node::swapNextInorder() {          //O(log n)
    node* tmp  = this->right;                          //O(1)
    node* prev = this;                                 //O(1)
    while (tmp->left != nullptr) {                     //O(2 log n) = O(log n)
        prev = tmp;                                      //O(1)
        tmp = tmp->left;                                 //O(1)
    }

    this->val = tmp->val;                              //O(1)
    if (prev == this)                                  //O(1)
        prev->right = tmp->right;                      //O(1)
    else                                               //O(1)
        prev->left = tmp->right;                       //O(1)
    delete tmp;                                        //O(1)
}
...
```

```
void binSearchTree::node::remove(int a, node &prev) {     //O(2 log n) = O(log n)
    if (val == a) {                                       //O(1)
        if (left == nullptr && right == nullptr) {        //O(1)
            if (prev.left->val == a)                      //O(1)
                prev.left = nullptr;                      //O(1)
            else
                prev.right = nullptr;                     //O(1)
            delete this;                                  //O(1)
        } else if (left != nullptr && right != nullptr) { //O(1)
            this->swapNextInorder();                      //O(log n)
        } else {
            node *tmp;                                    //O(1)
            if (left == nullptr) {                        //O(1)
                tmp = right;                              //O(1)
            } else {
                tmp = left;                               //O(1)
            }

            if (prev.left->val == a)                      //O(1)
                prev.left = tmp;                          //O(1)
            else
                prev.right = tmp;                         //O(1)
            delete this;                                  //O(1)
        }
    } else if (val > a) {                                 //O(1)
        if (left != nullptr)                              //O(1)
            left->remove(a, *this);                       //WC: O(2 log n) = O(log
            n)
    } else if (val < a) {
        if (right != nullptr)
            right->remove(a, *this);                      //WC: O(log n)
    }
}
...
void binSearchTree::remove(int a) {                       // O(log(n))
    if (root == nullptr)
        return;
    if (root->val == a) {
        root->swapNextInorder();
    } else {
        if (a > root->val)
            root->right->remove(a, *root);
        else
            root->left->remove(a, *root);
    }
}
```

Aufgabe 2

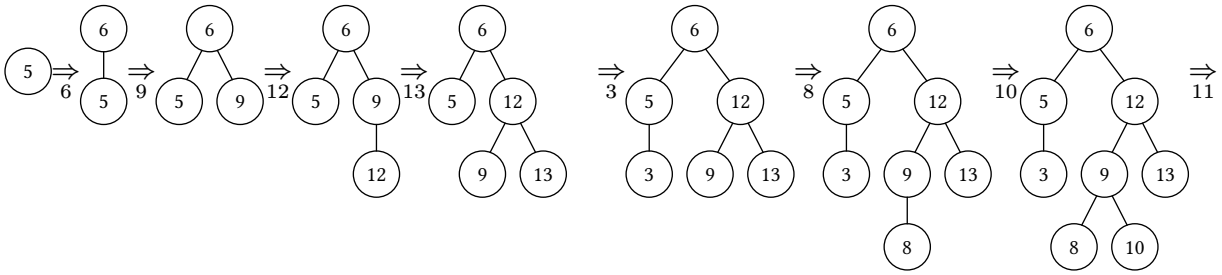
cpp

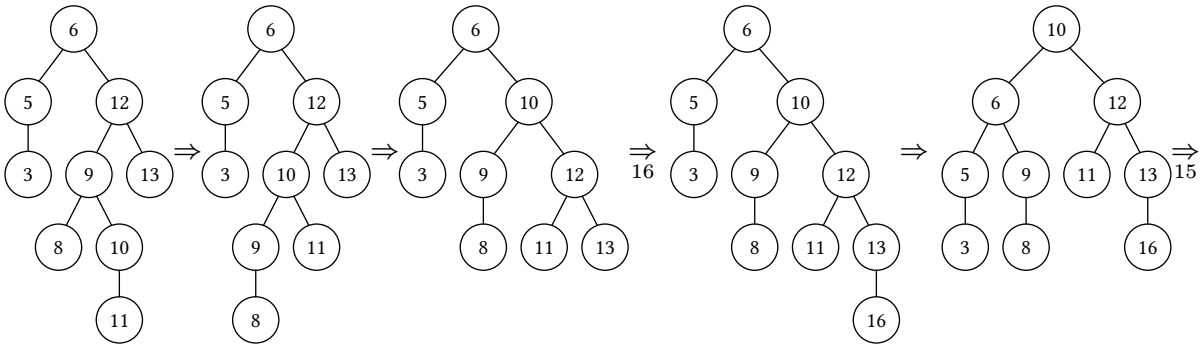
```cpp
void binSearchTree::node::descendInPreOrder(int *inorder, int *preorder, int inStart, int inEnd, int &index) {
    if (index > inEnd)
        return;
    char restAmount = inEnd - inStart + 1;
    if (restAmount == 1) {
        if (val > preorder[index])
            left = new node(index);
        else
            right = new node(index);
        index++;
        return;
    }

    int posCurrentElement = inStart;
    for (; inorder[posCurrentElement] != val && posCurrentElement <= inEnd; posCurrentElement++);

    if (preorder[index] < val)
        left  = new node(preorder[index++]);
        left->descendInPreOrder(inorder, preorder, inStart, posCurrentElement-1, index);

    if (preorder[index] > val)
        right = new node(preorder[index++]);
        right->descendInPreOrder(inorder, preorder, posCurrentElement+1, inEnd, index);
}

void binSearchTree::fromInAndPreOrder(int *inorder, int *preorder, unsigned maxIndex) {
    root = new node(preorder[0]);
    int index = 1;
    root->descendInPreOrder(inorder, preorder, 0, maxIndex, index);
}

binSearchTree createTree(int *inorder, int *preorder, unsigned int amount) {
    binSearchTree res;
    if (inorder == nullptr || preorder == nullptr || amount == 0)
        return res;

    res.fromInAndPreOrder(inorder, preorder, amount - 1);
    return res;
}
```

Aufgabe 3

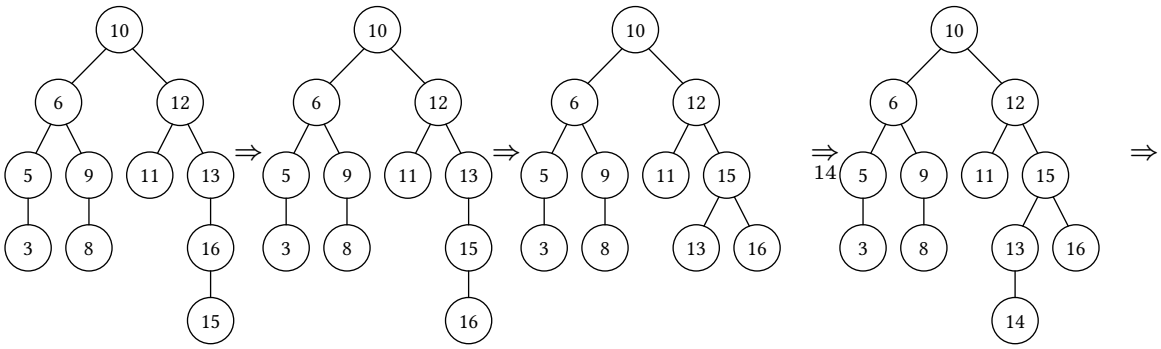
Füge zuerst [5,6,9,12,13,3,8,10,11,16,15,14,4,2,1] in dieser Reihenfolge in einen AVL-Baum ein dann entferne [13,8,5,4,3,6,15,14] in dieser Reihenfolge

- Einfügen:

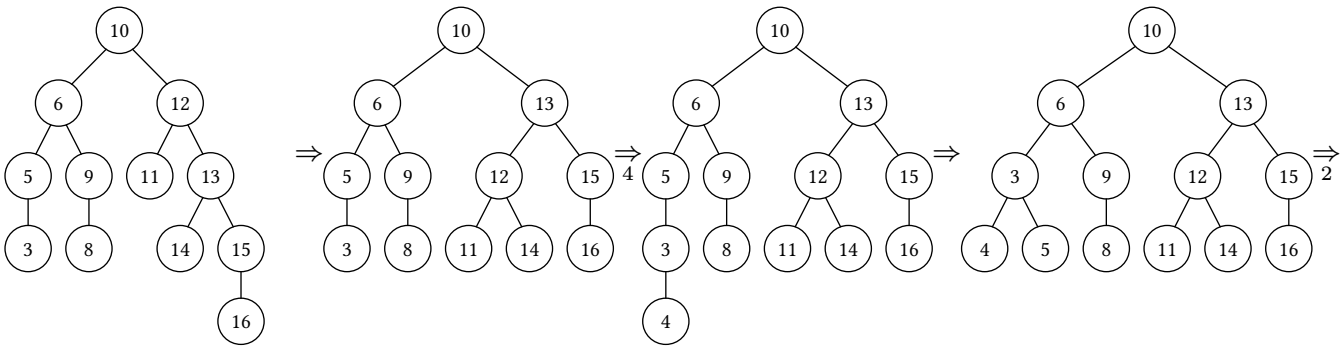




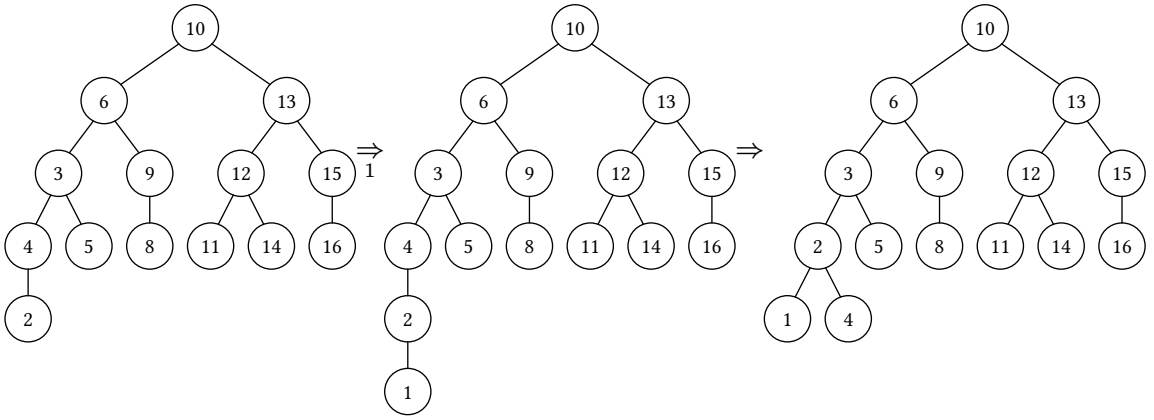
invalid invalid invalid



invalid invalid invalid

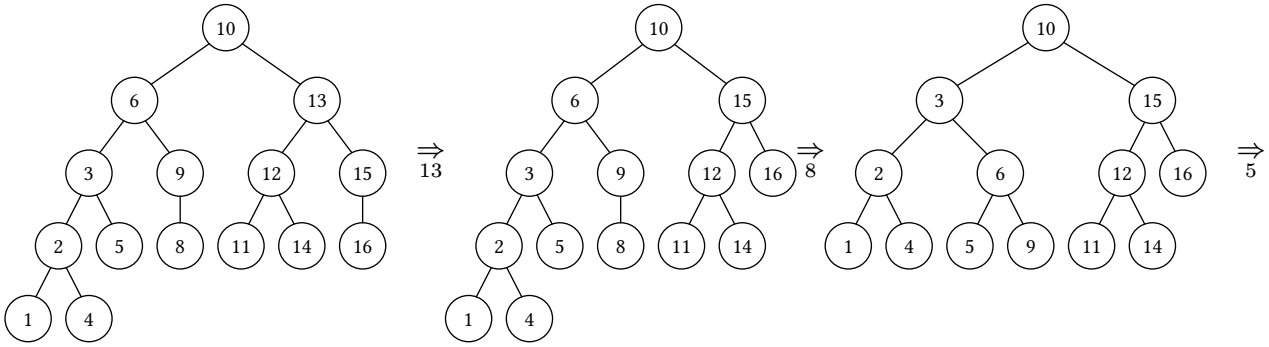


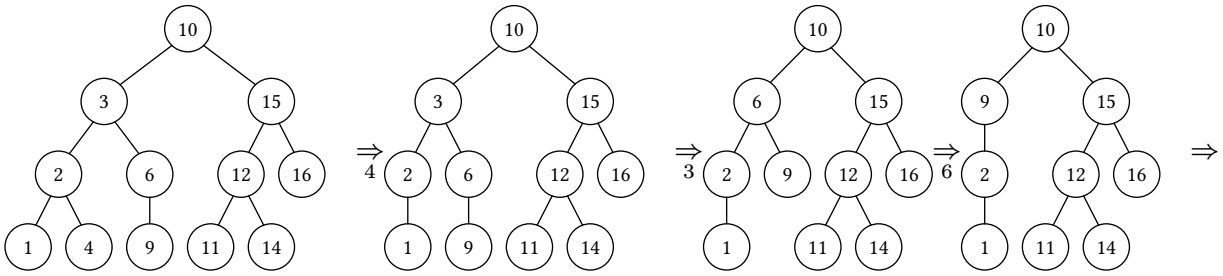
invalid invalid



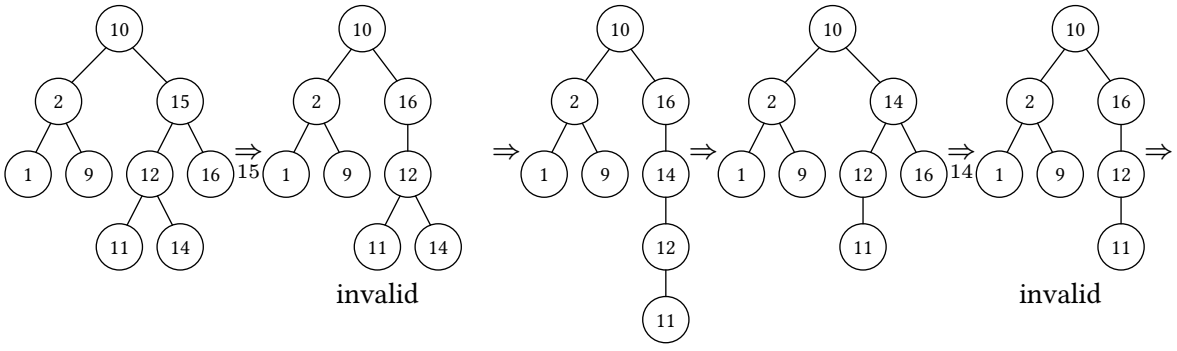
invalid

- Entfernen:





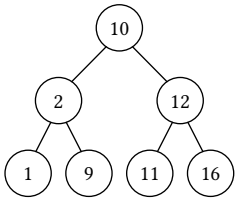
invalid



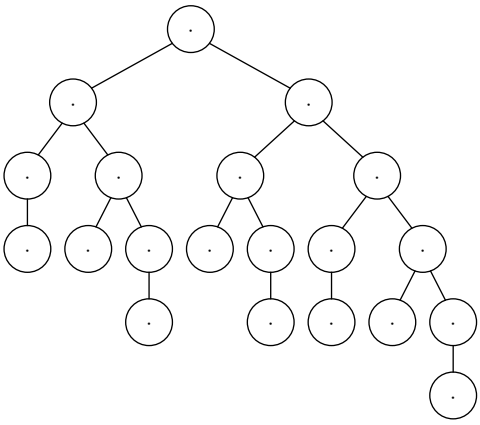
invalid

invalid

invalid



Aufgabe 4



Ein minimaler AVL-Baum der Höhe h setzt sich zusammen aus minimal AVL-Bäumen der Höhe h-1 und Höhe h-2 und dem Root Knoten. Also sei $A_{\min}(h)$ die MinimalAnzahl eines AVL-Baums der Höhe h, definiert durch:

$$A_{\min}(0) = 1, A_{\min}(1) = 2, A_{\min}(h) = A_{\min}(h-1) + A_{\min}(h-2) + 1$$

An jedem inneren Knoten kann man linken und rechten Teilbaum strukturell tauschen.

$$\#_{\text{Bäume}}(h) = 2^{\#_{\text{innere Knoten}}(h)}$$

Die Anzahl der Blätter ist nach Rekursionsformel $A_{\min}$:

$$\begin{aligned}
A_{\min}(0) &= \#_{\text{Blätter}}(0) = 1; A_{\min}(1) = \#_{\text{innere Knoten}}(1) + \#_{\text{Blätter}}(1) = 1 + 1 = 2; \\
A_{\min}(h) &= A_{\min}(h-1) + A_{\min}(h-1) + 1 \\
&= \#_{\text{innere Knoten}}(h-1) + \#_{\text{Blätter}}(h-1) + \#_{\text{innere Knoten}}(h-2) + \#_{\text{Blätter}}(h-2) + 1 \\
&= (\#_{\text{innere Knoten}}(h-1) + \#_{\text{innere Knoten}}(h-2) + 1) + (\#_{\text{Blätter}}(h-1) + \#_{\text{Blätter}}(h-2)) \\
&= \#_{\text{innere Knoten}}(h) + \#_{\text{Blätter}}(h)
\end{aligned}$$

Also gilt $\#_{\text{Blätter}}(h) = \#_{\text{Blätter}}(h-1) + \#_{\text{Blätter}}(h-2)$ mit $\#_{\text{Blätter}}(0) = 1$ und $\#_{\text{Blätter}}(1) = 1$ und $\#_{\text{innere Knoten}}(h) = \#_{\text{Knoten}}(h-1) + \#_{\text{innere Knoten}}(h-2) + 1$ mit $\#_{\text{innere Knoten}}(0) = 0$ und $\#_{\text{innere Knoten}}(1) = 1$

Für h = 5:

$$\begin{aligned}
\#_{\text{Bäume}}(5) &= 2^{\#_{\text{innere Knoten}}(5)} = 2^{\#_{\text{innere Knoten}}(4)+\#_{\text{innere Knoten}}(3)+1} \\
&= 2^{\#_{\text{innere Knoten}}(3)+\#_{\text{innere Knoten}}(2)+1+\#_{\text{innere Knoten}}(2)+\#_{\text{innere Knoten}}(1)+1+1} \\
&= 2^{\#_{\text{innere Knoten}}(2)+\#_{\text{innere Knoten}}(1)+1+\#_{\text{innere Knoten}}(1)+\#_{\text{innere Knoten}}(0)+1+\#_{\text{innere Knoten}}(1)+\#_{\text{innere Knoten}}(0)+1+1+3} \\
&= 2^{\#_{\text{innere Knoten}}(1)+\#_{\text{innere Knoten}}(0)+1+1+1+0+1+0+7} = 2^{12}
\end{aligned}$$

auch möglich:

$$n(h) := \#_{\text{min AVL-Bäume mit Höhe h}}$$

$$n(0) = 1$$

$$n(1) = 2$$

$$n(h) = 2 \cdot n(h-1) \cdot n(h-2)$$

Algorithmus für alle minimalBäume

```python
class node:
  def __init__(self, height):
    self.height = height
    self.left   = none
    self.right  = none
  def next_depth(self, height):
    if (height == 0):
      return
    if (height == 1):
      self.right = node(0)
      return
    self.right = node(height - 1)
    self.left  = node(height - 2)

    self.right.next_depth(height-1)
    self.left.next_depth(height-2)

  def swapPrinted(self, tree):
    print(tree)
    if (self.left == none && self.right == none): #case root node
      return

    self.if (self.left == none && self.right != none || self.left != none &&
    self.right == none): #case only one child node
      tmp = self.right
      self.right = self.left
      self.left = tmp
      print(tree)
      return

    tmp = self.right
    self.right = self.left
    self.left = tmp;

    self.left.swapPrinted();
    self.right.swapPrinted();

class tree:
  def __init__(self):
    root = none
  def createMinimal(h):
    if root == none:
      root = node(h);
    if (h > 0):
      root.next_depth(h)
```

```
  def getAllMinimal(self):
    if (self.root != none)
      self.root.swapPrinted(self)
```

Zusatz:
Sei $A_{\min}(h)$ die Anzahl von Knoten in einem minimalen AVL-Baum der Höhe h
Zu zeigen: für die Folge $A_{\min}(h)$:

$$A_{\min}(0) = 1, A_{\min}(1) = 2, A_{\min}(h) = A_{\min}(h-1) + A_{\min}(h-2) + 1$$

gilt

$$A_{\min}(h) = \text{Fib}(h+2) - 1$$

- Induktions Anfang $h = 0 \land h = 1$:

$$A_{\min}(0) = 1 = 1 + 1 - 1 = \text{Fib}(0) + \text{Fib}(1) - 1 = \text{Fib}(2) - 1$$
$$A_{\min}(1) = 2 = 1 + 2 - 1 = \text{Fib}(1) + \text{Fib}(2) - 1 = \text{Fib}(3) - 1$$

- Induktions Annahme: Es gibt ein h-1 und h-2 für das die Aussage gilt
- Induktions Behauptung: für die h-1 und h-2 gilt die Aussage auch für h
- Induktions Schritt: $h - 1 \land h - 2 \to h$

$$A_{\min}(h) = A_{\min}(h-1) + A_{\min}(h-2) + 1 \underset{\text{IA}}{=} F(h+1) - 1 + F(h) - 1 + 1$$
$$\Rightarrow A_{\min}(h) = \text{Fib } (h+2) - 1$$