```
#include <iostream>
using namespace std;

class BST {
   int data;
   BST *left, *right;
public:
   BST() : data(0), left(NULL), right(NULL) {}
   BST(int value) : data(value), left(NULL), right(NULL) {}

   BST* insert(BST*, int);
   void inorder(BST*);
   BST* search(BST*, int);
   int minvalue(BST*);
   void mirror(BST*);
   int height(BST*);
};

BST* BST::insert(BST* root, int value) {
   if (!root) return new BST(value);
   if (value < root->data) root->left = insert(root->left, value);
   else if (value > root->data) root->right = insert(root->right, value);
   return root;
}

void BST::inorder(BST* root) {
   if (!root) return;
   inorder(root->left);
   cout << root->data << " ";
   inorder(root->right);
}
```

```
BST* BST::search(BST* root, int key) {
    if (!root || root->data == key) return root;
    if (key < root->data) return search(root->left, key);
    return search(root->right);
}

int BST::minvalue(BST* root) {
    if (!root) return -1;
    while (root->left) root = root->left;
    return root->data;
}

void BST::mirror(BST* root) {
    if (!root) return;
    swap(root->left, root->right);
    mirror(root->left);
    mirror(root->right);
}

int BST::height(BST* root) {
    if (!root) return -1;
    return max(height(root->left), height(root->right)) + 1;
}

int main() {
    BST b, *root = NULL;
    int ch, n, value, key;

    while (true) {
        cout << "\\n1) Insert new node"
```

```
     << "\\n2) Height of the tree"
     << "\\n3) Minimum Value"
     << "\\n4) Mirror the tree"
     << "\\n5) Search"
     << "\\n6) Display (Inorder)"
     << "\\n7) Exit\\n";
cout << "Enter your choice: ";
cin >> ch;

switch (ch) {
   case 1:
      cout << "Enter number of elements to insert: ";
      cin >> n;
      for (int i = 0; i < n; ++i) {
         cout << "NUMBER = ";
         cin >> value;
         root = b.insert(root, value);
      }
      cout << "Inorder Traversal: ";
      b.inorder(root);
      cout << endl;
      break;

   case 2:
      cout << "Height (Longest path in edges): " << b.height(root) << endl;
      break;

   case 3:
      cout << "Minimum Value = " << b.minvalue(root) << endl;
      break;
```

```
        case 4:
          cout << "Original Tree (Inorder): ";
          b.inorder(root);
          cout << endl;
          b.mirror(root);
          cout << "Mirrored Tree (Inorder): ";
          b.inorder(root);
          cout << endl;
          break;
         case 5:
          cout << "Enter key to search: ";
          cin >> key;
          if (b.search(root, key))
              cout << key << " found" << endl;
          else
              cout << key << " not found" << endl;
          break;
         case 6:
          cout << "Inorder Traversal: ";
          b.inorder(root);
          cout << endl;
          break;
         case 7:
          return 0;
         default:
          cout << "Invalid choice!" << endl;
      }
    }
return 0;
}
```

Output:

```
Input: 50, 30, 70, 20, 40, 60, 80

Inorder Traversal: 20 30 40 50 60 70 80
Height (Longest path in edges): 2
Minimum Value = 20
Original Tree (Inorder): 20 30 40 50 60 70 80
Mirrored Tree (Inorder): 80 70 60 50 40 30 20
Search 60: 60 found
```