CS 211 : Tues 02/27 (lecture 16)

Prof. Hummel
(he/him)

- **Topics**: **efficiency, search trees, set**

February 2024

Sunday	Monday	Tuesday	Wednesday	Thursday	Friday	Saturday
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29		

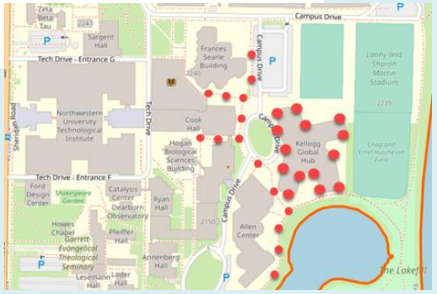
www.a-printable-calendar.com

Notes:

- *Lecture slides available on Canvas*
- ***HW 07** due tonight (Tuesday)*
- ***Project 07** due Friday night (can submit as late as Sunday with late days)*

Northwestern University

Project 06 --- efficiency



- **Imagine working with a real map**
  - *Millions of nodes / positions*
  - *Millions of footways / sidewalks / roads*
  - *Millions of buildings / structures*

```
hummel> ./a.out
** NU open street map **

Enter map filename>
ns.osm
# of nodes: 114044
# of buildings: 6916
# of footways: 1486336
Time: 23402ms

Enter building name (partial or complete), or * to list, or $ to end>
University Library
Northwestern University Library
Address: 1970 Campus Drive
Building ID: 175187764
Nodes: 268
Footways that intersect: 3072
Time: 9159ms
```

Correctness first, efficiency second…

Timing in C++

- **Add timing code to see what really works…**

```
#include <chrono>
```

```
auto start = chrono::high_resolution_clock::now();

  .
  .  // computation you want to time
  .

auto stop = chrono::high_resolution_clock::now();
auto diff = stop - start;
auto duration = chrono::duration_cast<chrono::milliseconds>(diff);

cout << "Time: " << duration.count() << "ms" << endl;
```

Approach #1

- **This takes nearly a minute… And yields duplicate footway IDs… what would you recommend as improvements?**

```
void Footways::intersectWithThisNode(long long nodeid, vector<long long>& footwayIDs)
{
  for (Footway F : this->MapFootways) {
    if (F.intersectWithThisNode(nodeid)) {
      footwayIDs.push_back(F.ID);
    }
  }
}
```

```
bool Footway::intersectWithThisNode(long long nodeid)
{
  for (long long id : this->NodeIDs)
    if (id == nodeid)
      return true;

  return false;
}
```

```
void Building::intersectWithFootways(Footways& footways)
{
  vector<long long> footwayIDs;

  auto start = chrono::high_resolution_clock::now();

  for (long long id : this->NodeIDs) { // for each node in this building
    footways.intersectWithThisNode(id, footwayIDs);
  }

  auto stop = chrono::high_resolution_clock::now();
  auto diff = stop - start;
  auto duration = chrono::duration_cast<chrono::milliseconds>(diff);

  cout << "Footways that intersect: " << footwayIDs.size() << endl;
  cout << "Time: " << duration.count() << "ms" << endl;
}
```

```
Enter building name (partial or complete), or * to list, or $ to end>
University Library
Northwestern University Library
Address: 1970 Campus Drive
Building ID: 175187764
Nodes: 268
Footways that intersect: 4096
Time: 56762ms
```

Improvements…

1. For loop should reference footway, not copy
2. Replace vector with set --- eliminates duplicates
3. Store footway nodes in a set for O(lgN) searching

4. Order matters…

Approach #2

- **You can get < 1 second using sets and ordering things properly…**

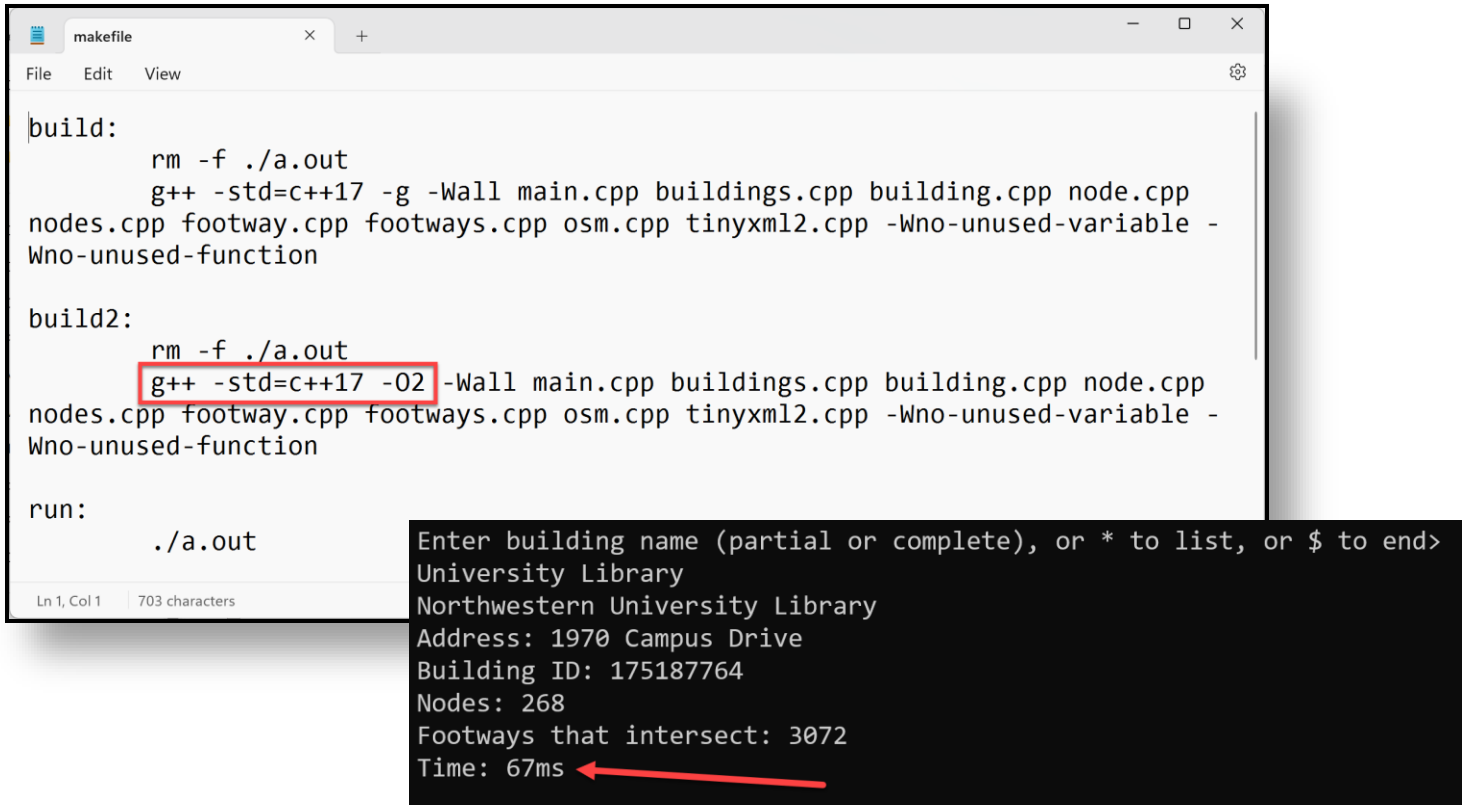
```
bool Footway::intersectWithBuilding(Building& B)
{
  for (long long id : this->NodeIDs)
    if (B.SetIDs.count(id) > 0)  // O(lgN) search:
      return true;

  return false;
}
```

```
void Footways::intersectWithBuilding(Building& B)
{
  vector<long long> footwayIDs;

  auto start = chrono::high_resolution_clock::now();

  for (Footway& F : this->MapFootways) {  // for each footway…
    if (F.intersectWithBuilding(B)) { // if footway intersects building, store ID:
      footwayIDs.push_back(F.ID);
    }
  }

  auto stop = chrono::high_resolution_clock::now();
  auto diff = stop - start;
  auto duration = chrono::duration_cast<chrono::milliseconds>(diff);

  cout << "Footways that intersect: " << footwayIDs.size() << endl;
  cout << "Time: " << duration.count() << "ms" << endl;
}
```

```
Enter building name (partial or complete), or * to list, or $ to end>
University Library
Northwestern University Library
Address: 1970 Campus Drive
Building ID: 175187764
Nodes: 268
Footways that intersect: 3072
Time: 743ms
```

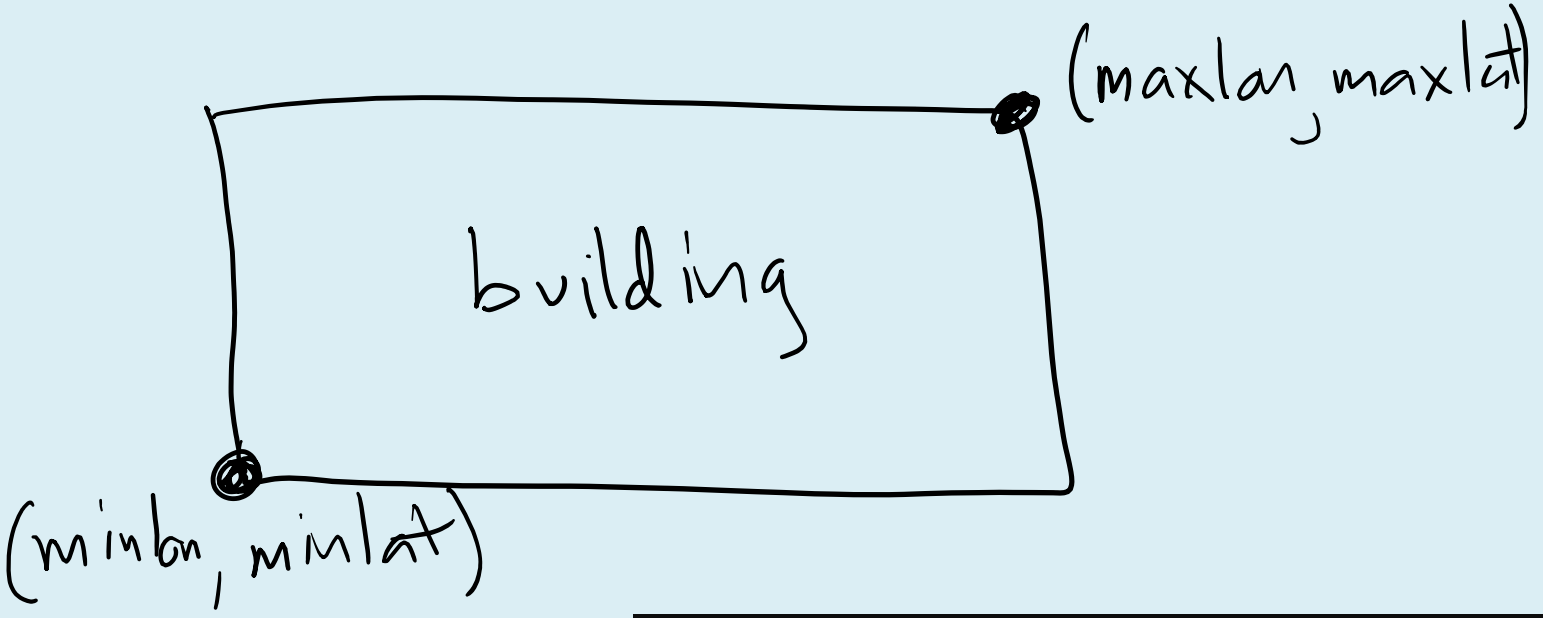
-O2

- **Code optimization yields another 10x speed increase**



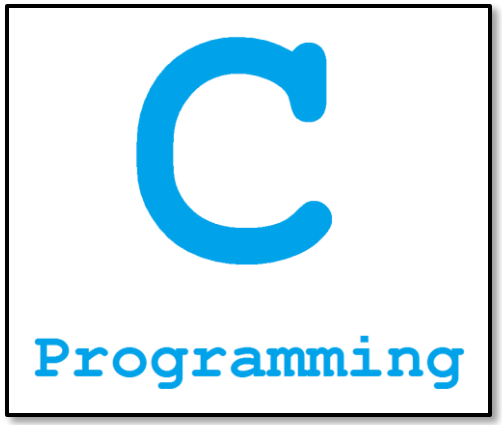
Approach #3

- **"Bounding Box" algorithm**
  - *For each building & footway, pre-compute "bounding box"*
  - *Only consider footways that overlap with building*



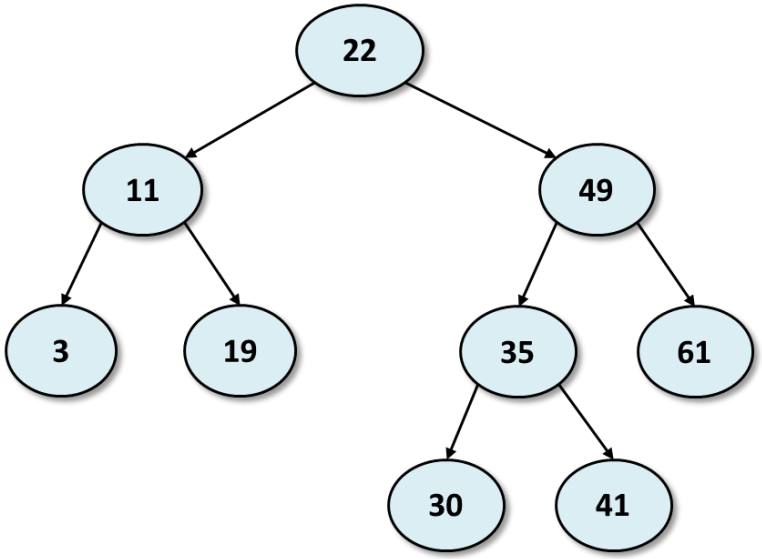

```
Enter building name (partial or complete), or * to list, or $ to end>
University Library
Northwestern University Library
Address: 1970 Campus Drive
Building ID: 175187764
Nodes: 268
Footways that intersect: 3072
Time: 293ms
```


C++


C
Programming

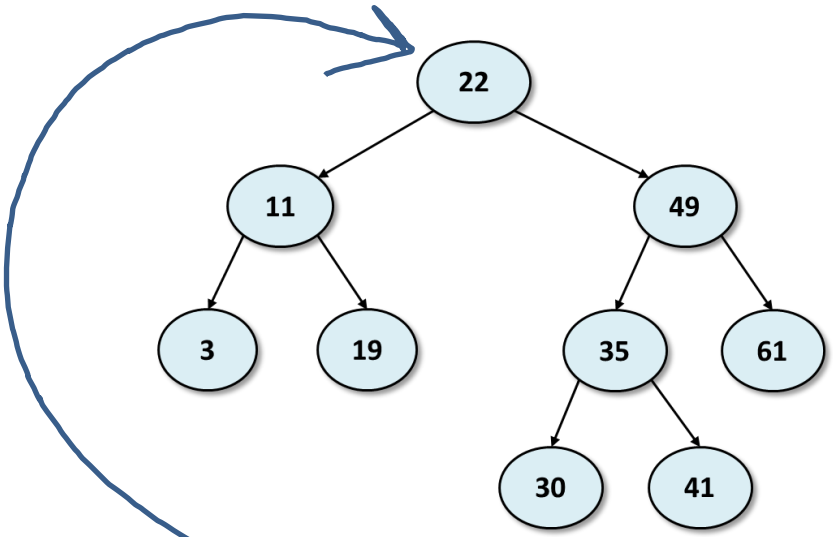
set

- **set is another C++ abstraction for a search tree**
  - *Models mathematical set (no duplicates)*
  - *Efficient operations => O(lgN)*

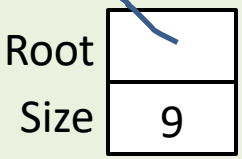


Set

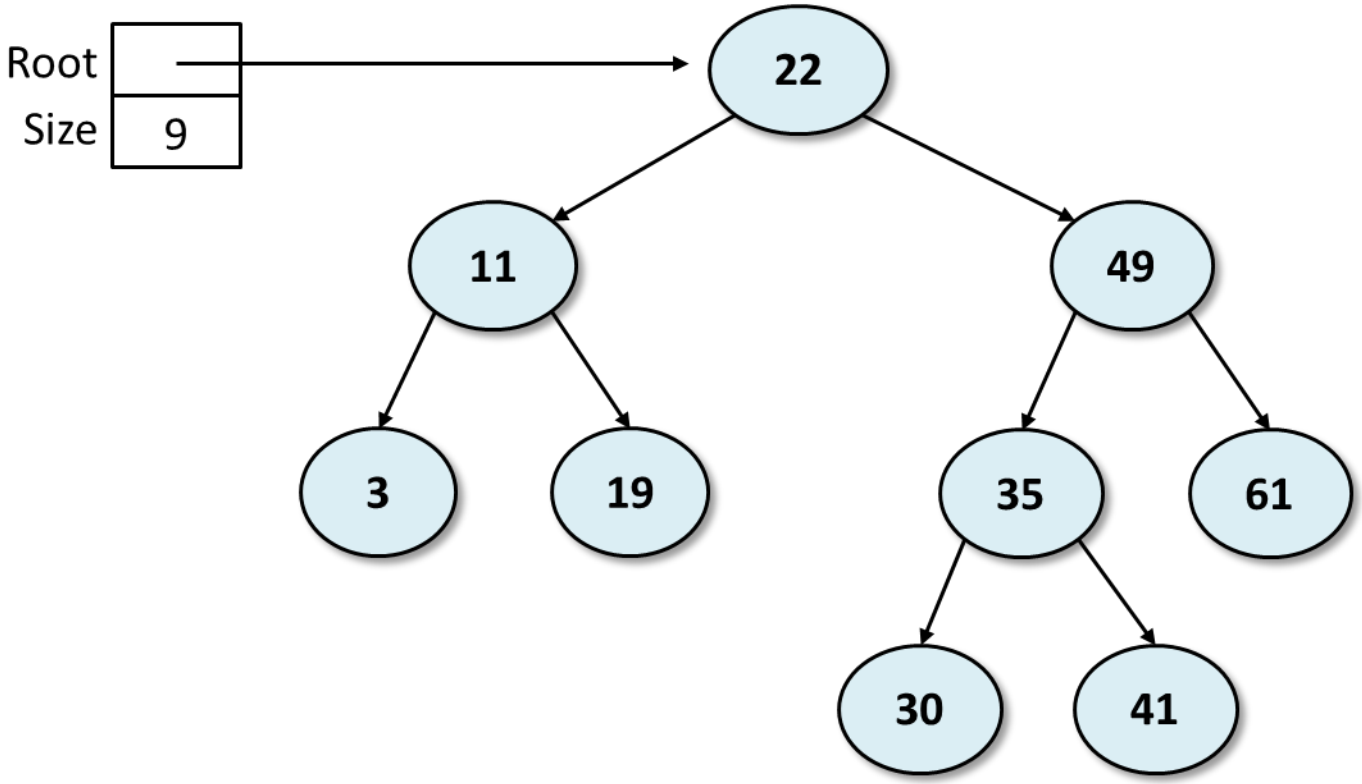
```
template<typename TKey>
class set
{
private:
  struct NODE
  {
    TKey  Key;
    NODE* Left;
    NODE* Right;
  };

  NODE* Root;  // pointer to root node
  int   Size;  // # of nodes in tree

public:
  // default constructor:
  set()
   : Root(nullptr),
     Size(0)
  { }
```




```
#include <set>

int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .
  .
```



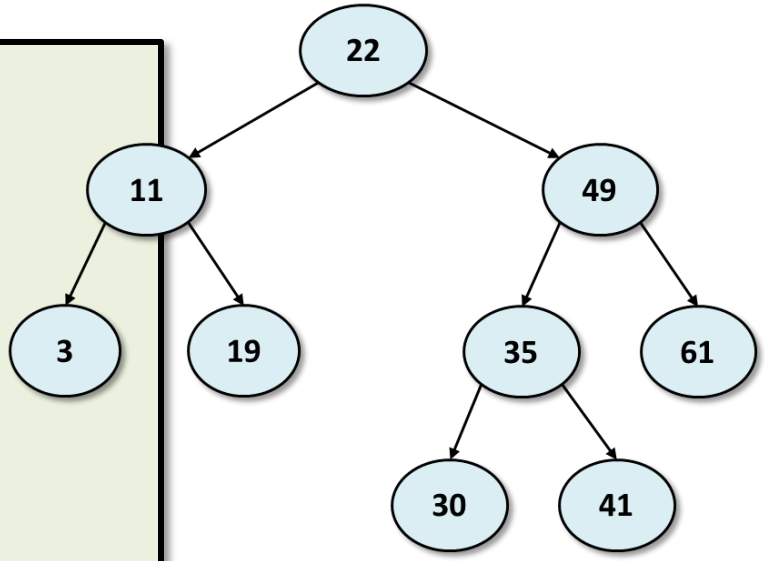
- **Search: You are searching for the key 40. Trace out the path searched, highlighting the edges traversed & circling the nodes visited.**



set::contains(key)

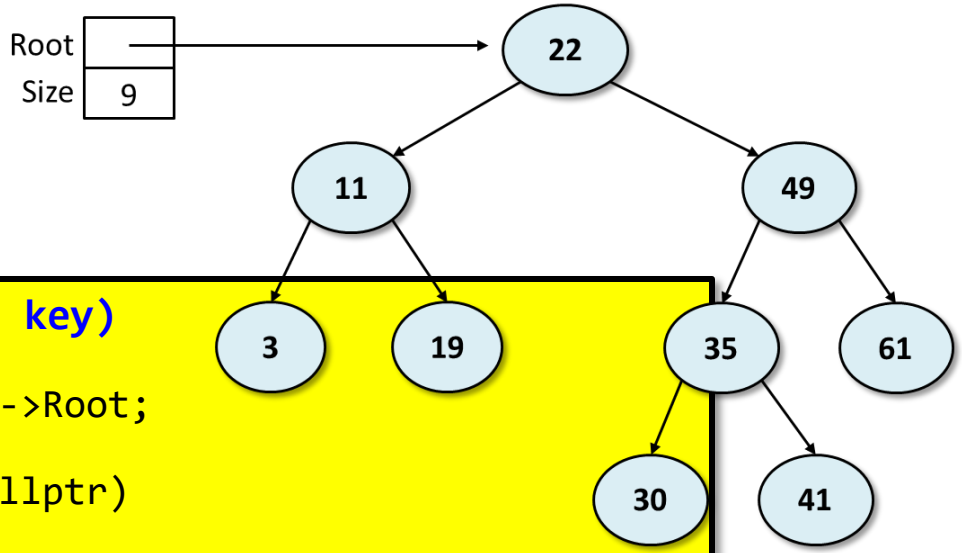
- **Searches set for given key**

```
int main()
{
  set<int>  S;

  S.insert(22);
  S.insert(11);
  S.insert(49);
  .
  .
  .

  int x;
  cout << "Enter an integer> ";
  cin >> x;

  if (S.contains(x))
    cout << x << " is a member" << endl;
```



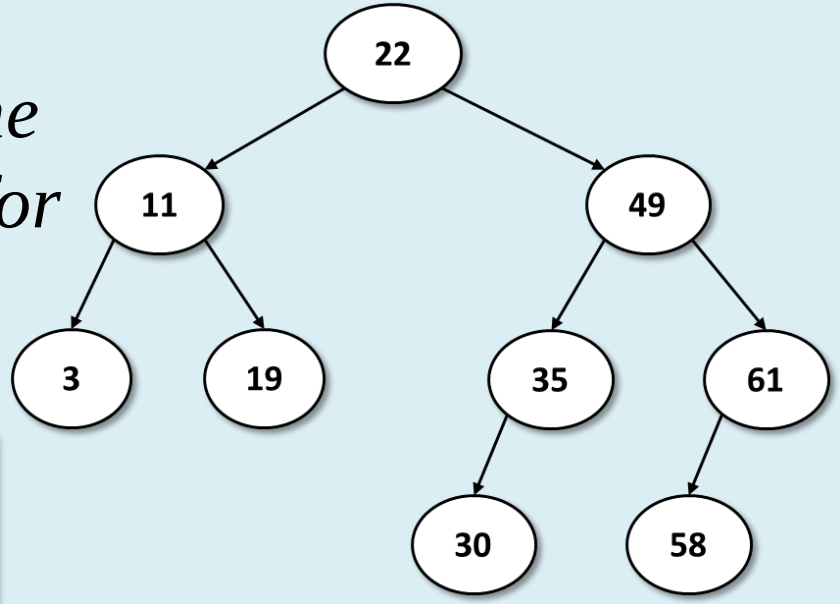
set::contains

```
template<typename TKey>
class set
{
private:
  struct NODE
  {
    TKey  Key;
    NODE* Left;
    NODE* Right;
  };

  NODE* Root;  // pointer to
  int   Size;  // # of nodes

public:
  // default constructor:
  set()
   : Root(nullptr),
     Size(0)
  { }
```




```
bool contains(TKey key)
{
   NODE* cur = this->Root;

   while (cur != nullptr)
   {
      if (key == cur->Key) { // found it!
        return true;
      }
      else if (key < cur->Key) { // search left:
        cur = cur->Left;
      }
      else { // search right:
        cur = cur->Right;
      }
   }

   // if get here, not found
   return false;
}
```

1) *Here's a* ***mystery*** *function in the set class. What value is returned for the tree shown?*




```
TKey mystery()
{
   NODE* cur = this->Root;

   if (cur == nullptr)
     throw runtime_error("mystery: empty");

   while (cur->Right != nullptr)
   {
      cur = cur->Right;
   }

   return cur->Key;
}
```

A) 49

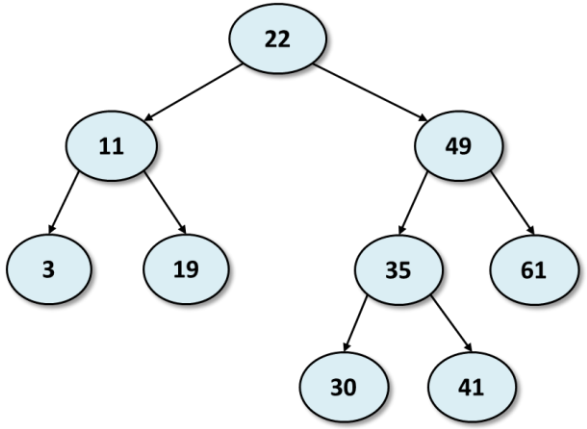
B) 61

C) 58

D) Throws runtime error

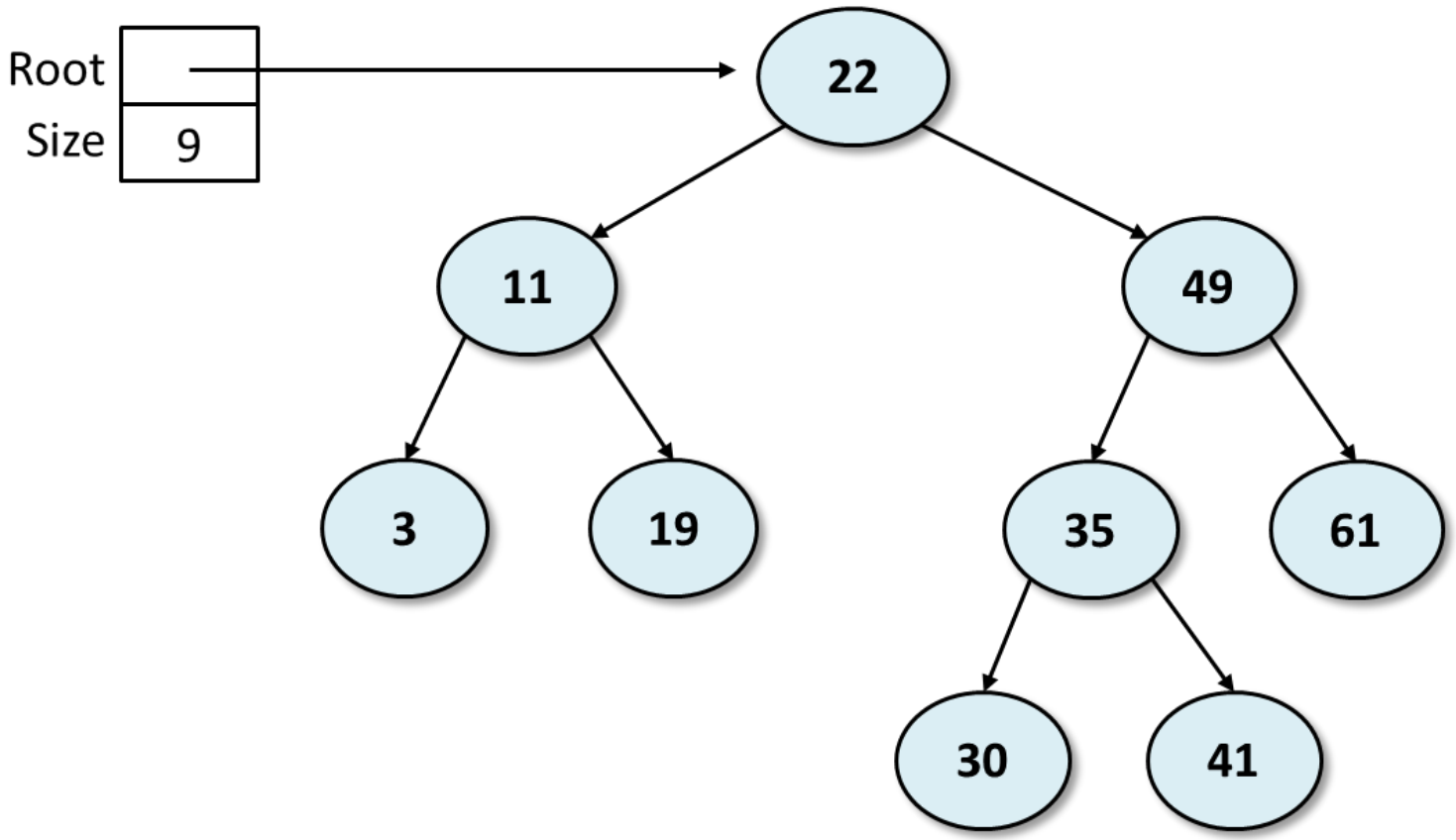
Inserting into a set

- **Elements are always inserted at the bottom**
  - *as a new "leaf"*



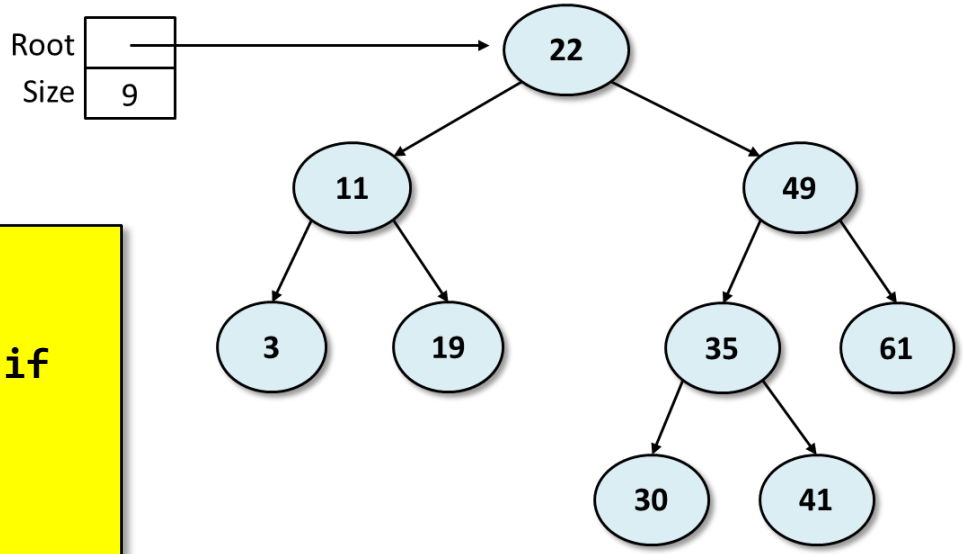
- **Algorithm:**
  - *Start by searching the tree…*
  - *If you find key, then return (don't insert twice)*
  - *Else insert new node where you fell out of the tree…*

- **Example: Insert 44**



Visualization: https://www.cs.usfca.edu/~galles/visualization/BST.html

set::insert

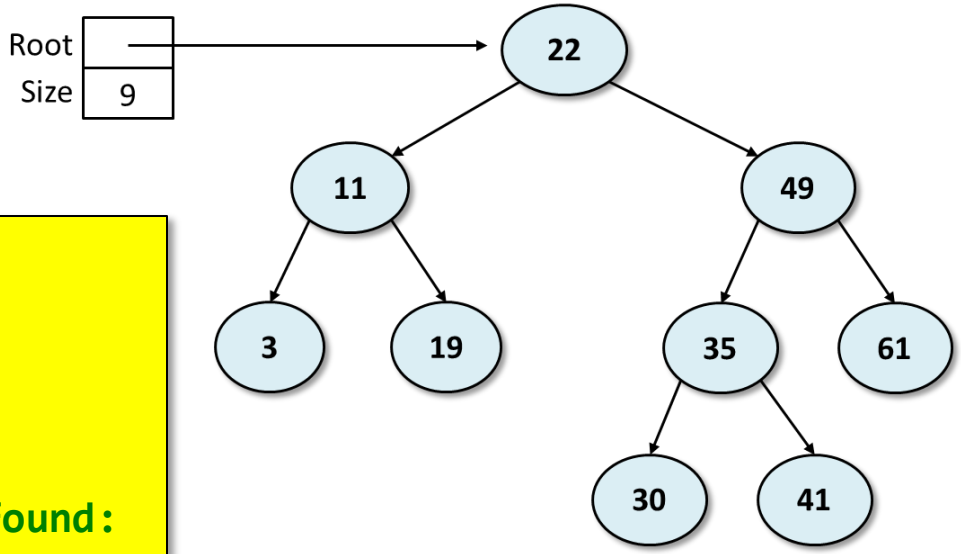



```
void insert(TKey key)
{
   // STEP 1: search for key, return if
   // found since no duplicates
   while (…)
   { … }

   // STEP 2: if not found, insert where
   // we fell out of the tree
   NODE *n = new NODE();
   link in…

   // STEP 3: update size and return
   this->Size++;
   return;
}
```

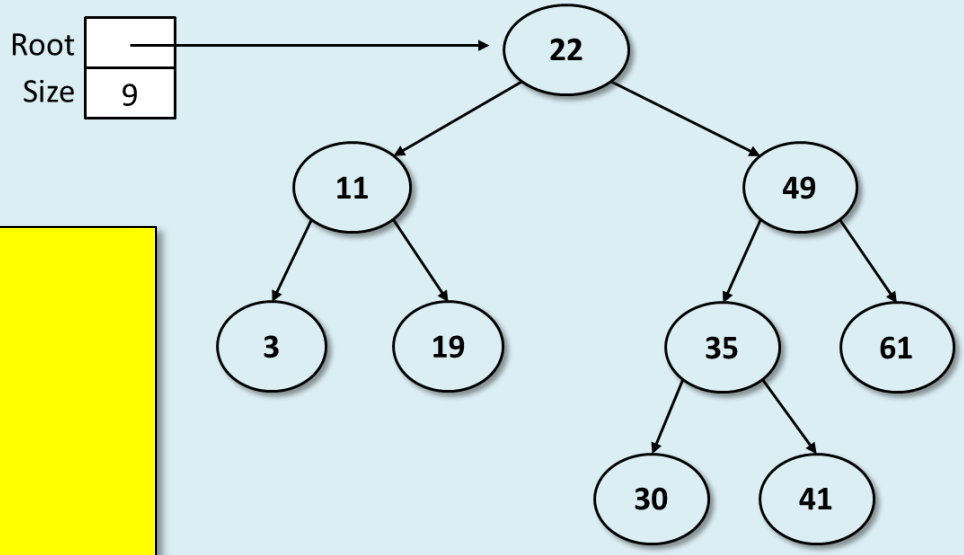
```
template<typename TKey>
class set
{
private:
  struct NODE
  {
    TKey  Key;
    NODE* Left;
    NODE* Right;
  };

  NODE* Root;  // pointer to root node
  int   Size;  // # of nodes in tree

public:
  // default constructor:
  set()
   : Root(nullptr),
     Size(0)
  { }
```

Step 1

```
void insert(TKey key)
{
   NODE* prev = nullptr;
   NODE* cur  = this->Root;

   //
   // 1. Search for key, return if found:
   //
   while (cur != nullptr)
   {
      if (key == cur->Key) {  // found
        return;
      }
      else if (key < cur->Key) { // left:
        prev = cur;
        cur  = cur->Left;
      }
      else { // right:
        prev = cur;
        cur  = cur->Right;
      }
   }
```

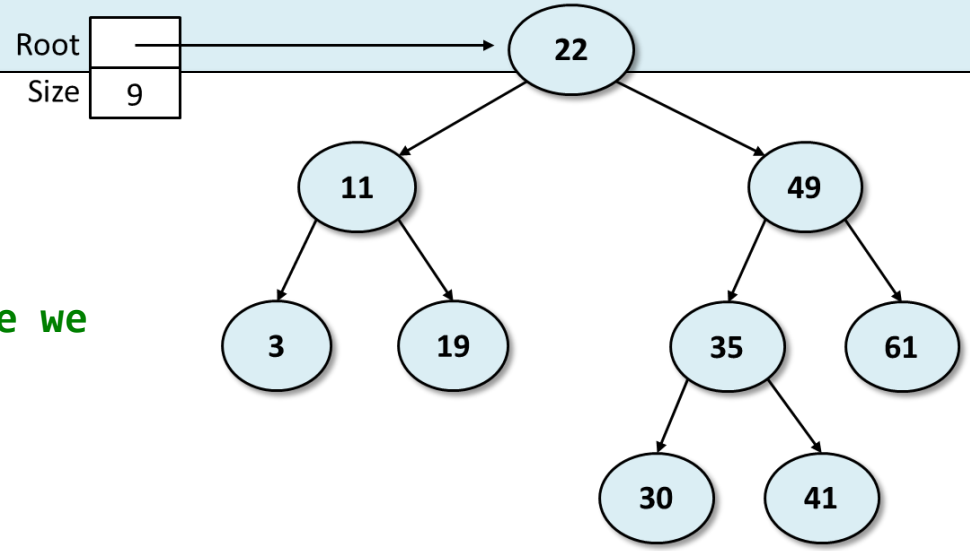


Step 2




```
void insert(TKey key)
{
   NODE* prev = nullptr;
   NODE* cur  = Root;

   .
   .  // search…
   .

   //
   // 2. If not found, insert where we
   //    fell out of the tree:
   //
   NODE *n  = new NODE();
   n->Key   = key;
   n->Left  = nullptr;
   n->Right = nullptr;

   << continued on next page>>
```

*2) The last bit of **step 2** is to link in the new node. Which is the correct code fragment that works in all cases?*




```
.
.
.

//
// 2. If not found, insert where we
//    fell out of the tree:
//
NODE *n  = new NODE();
n->Key   = key;
n->Left  = nullptr;
n->Right = nullptr;
```

(A)
```
if (key < prev->Key)
  prev->Left = n;
else if (key > prev->Key)
  prev->Right = n;
else
  this->Root = n;
```

(B)
```
if (prev == nullptr)
  this->Root = n;
else if (key < prev->Key)
  prev->Left = n;
else
  prev->Right = n;
```

(C)
```
if (cur == nullptr)
  this->Root = n;
else if (key < cur->Key)
  cur->Left = n;
else
  cur->Right = n;
```

What's due?

HW 07 *is due tonight*

Project 07 *is due Friday night*