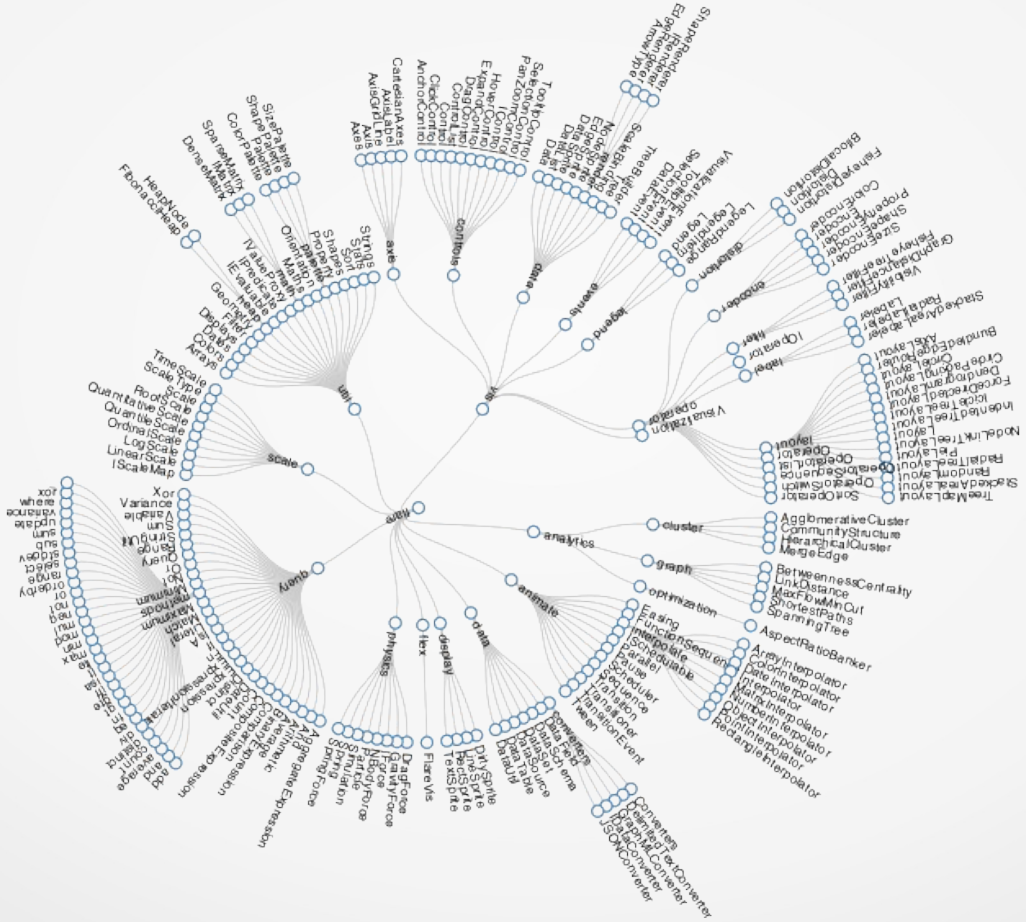
Hierarchical Data Structures

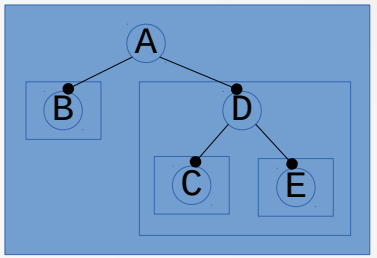
and Related Concepts



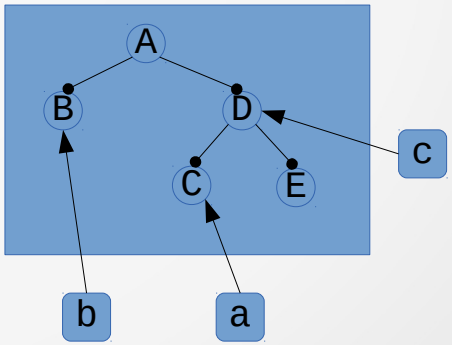
Design Choices

- Recursive vs. Cursors

Recursive



Cursors



Design Choices

- Recursive vs. Cursors
- Policies vs. Adaptors

Policy:

```
template <
  class T,
  class Allocator = std::allocator<T>>
class red_black_tree
  : tree<
      T, Allocator,
      red_black_balance<T,Alocator>>
{
  // ..
}
```

Adaptor:

```
template <
  class T,
  class Hiearchy
    = binary_tree<T>>
class red_black_tree
{
  // ..
  Hierarchy hierarchy;
}
```

Design Choices, Recursive, Pros

- Self-contained

begin	node_begin
clear	node_end
count	node_rbegin
empty	node_rend
end	parent
equal_range	post_order_begin
erase	post_order_end
find	post_order_node_begin
get	post_order_node_end
insert	pre_order_begin
is_root	pre_order_end
level_order_begin	pre_order_node_begin
level_order_end	pre_order_node_end
level_order_node_begin	rbegin
level_order_node_end	rend
lower_bound	set_clone
max_size	size
	swap
	upper_bound

Design Choices, Recursive, Pros

- Self-contained
- Uniform

begin	node_begin
clear	node_end
count	node_rbegin
empty	node_rend
end	parent
equal_range	post_order_begin
erase	post_order_end
find	post_order_node_begin
get	post_order_node_end
insert	pre_order_begin
is_root	pre_order_end
level_order_begin	pre_order_node_begin
level_order_end	pre_order_node_end
level_order_node_begin	rbegin
level_order_node_end	rend
lower_bound	set_clone
max_size	size
	swap
	upper_bound

Design Choices, Recursive, Cons

- Type Duplication

```
template<typename stored_type,
 typename node_compare_type =
  std::less<stored_type> >
class multitree
 : public associative_tree<
    stored_type,
    multitree<stored_type,
     node_compare_type>,
    std::multiset<
     multitree<stored_type,
      node_compare_type>*,
     multitree_deref_less<
      stored_type,
      node_compare_type> > >
{
  // associative_tree_type::
  // typedef pre_order_descendant_iterator<
  //  stored_type, tree_type, tree_type*,
  //  container_type, iterator, stored_type*,
  //  stored_type&>
  // pre_order_iterator;

  typedef typename
   associative_tree_type
    ::pre_order_iterator
  pre_order_iterator_type;

  pre_order_iterator_type
   pre_order_begin();
}
```

[haas]

Design Choices, Recursive, Cons

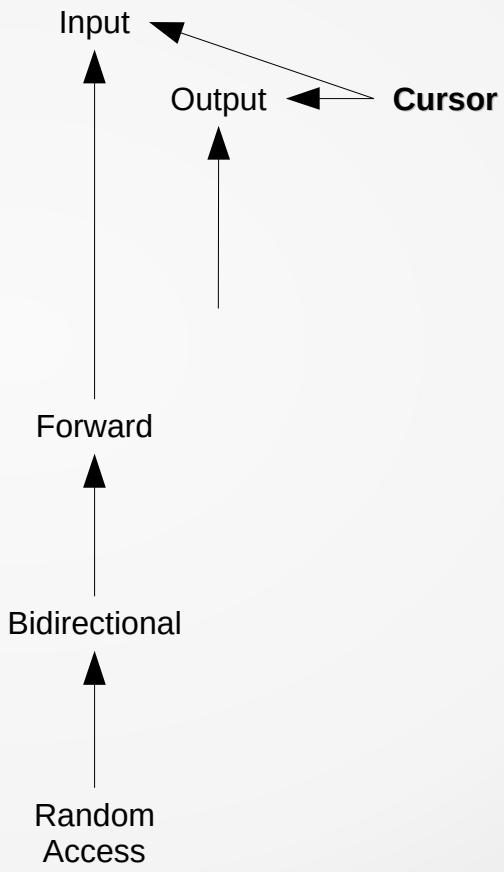
- Type Duplication
- Implementation Complexity

```
template<typename stored_type,
 typename node_compare_type =
  std::less<stored_type> >
class multitree
 : public associative_tree<
    stored_type,
    multitree<stored_type,
     node_compare_type>,
    std::multiset<
     multitree<stored_type,
      node_compare_type>*,
     multitree_deref_less<
      stored_type,
      node_compare_type> > >
{
  // associative_tree_type::
  // typedef pre_order_descendant_iterator<
  //  stored_type, tree_type, tree_type*,
  //  container_type, iterator, stored_type*,
  //  stored_type&>
  // pre_order_iterator;

  typedef typename
   associative_tree_type
    ::pre_order_iterator
  pre_order_iterator_type;

  pre_order_iterator_type
   pre_order_begin();
}
```

[haas]

Design Choices, Cursors

- Cursor
  - Requirements of input & output iterator.



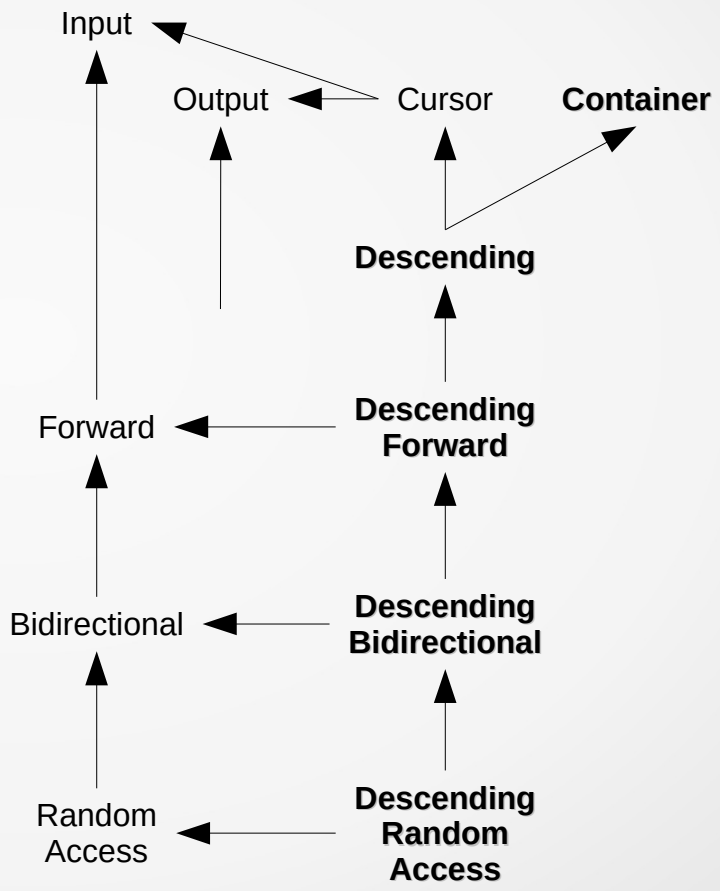
Design Choices, Cursors

- Descending Cursor
  - Children as container.

```
cursor begin();
cursor end();
const_cursor begin() const;
const_cursor end() const;
const_cursor cbegin() const;
const_cursor cend() const;

size_type size();
size_type max_size();
bool empty();
```

Design Choices, Cursors

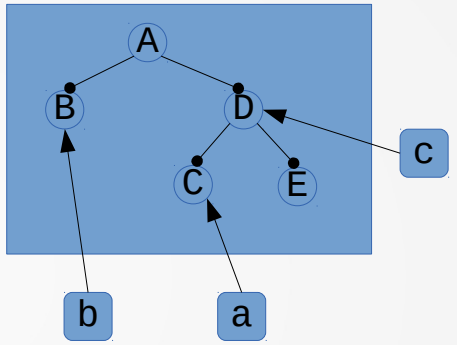
- Descending Cursor
  - Requirements of container.
  - Obtains cursors for the children of the cursor.



Design Choices, Cursors

- Descending Cursor
  - Requirements of container.
  - Obtains cursors for the children of the cursor.
  - Tree specific operations: parity().

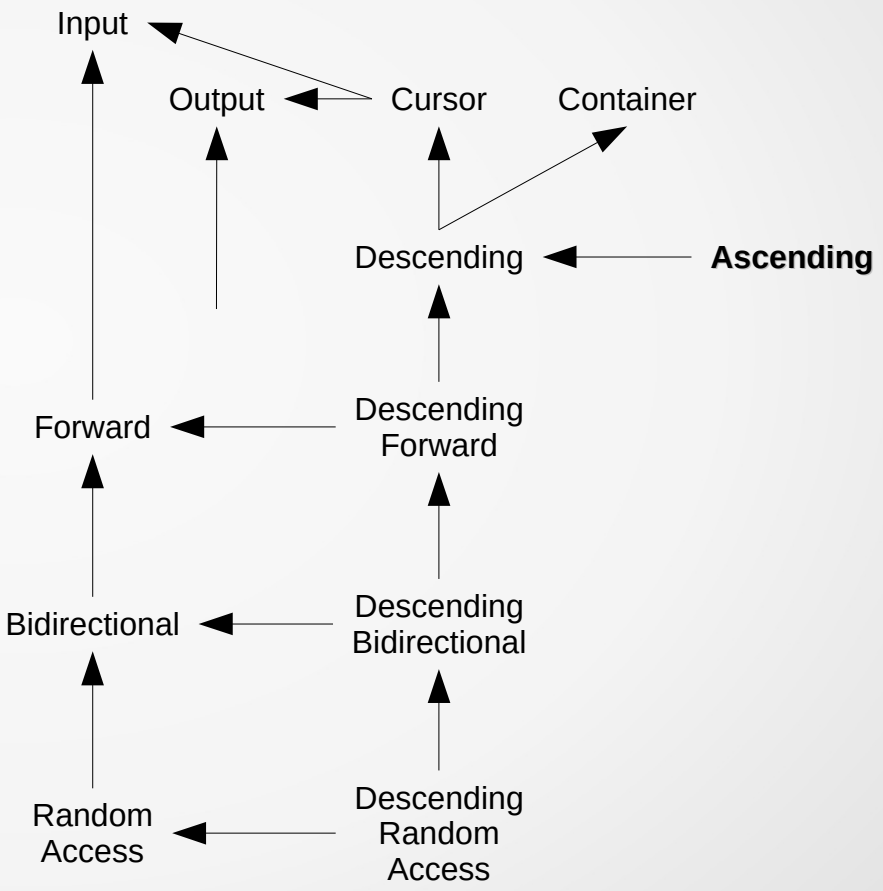
```
size_type parity();
```




```
a.parity() == 0
b.parity() == 0
c.parity() == 1
```

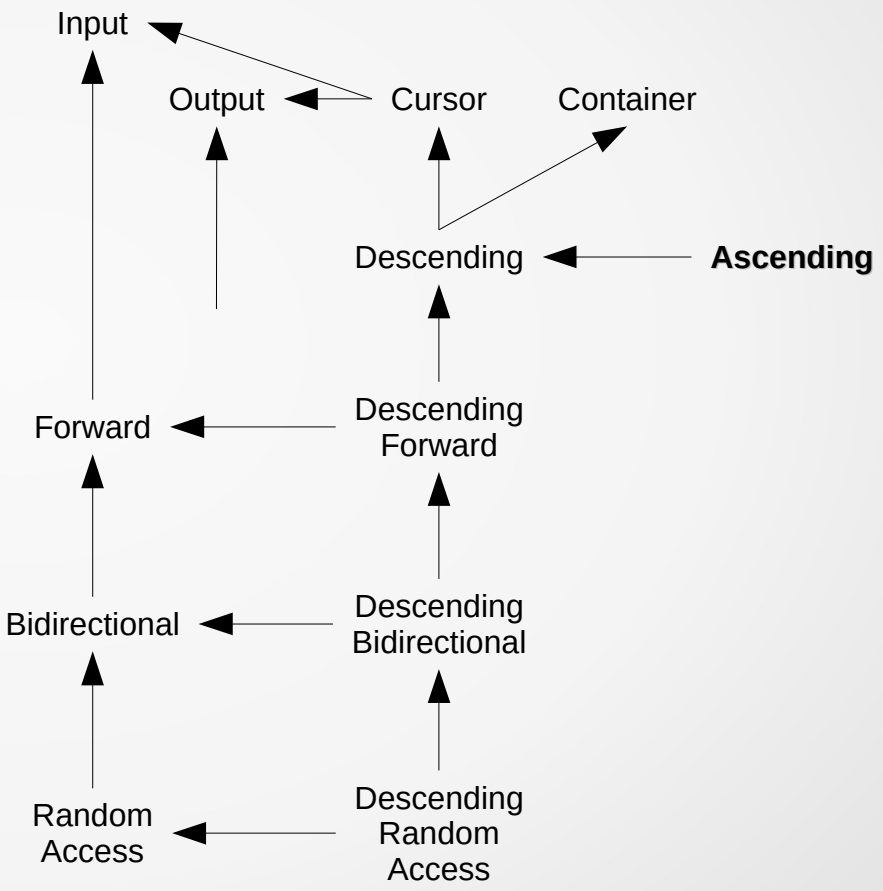
Design Choices, Cursors

- Ascending Cursor
  - Node parent



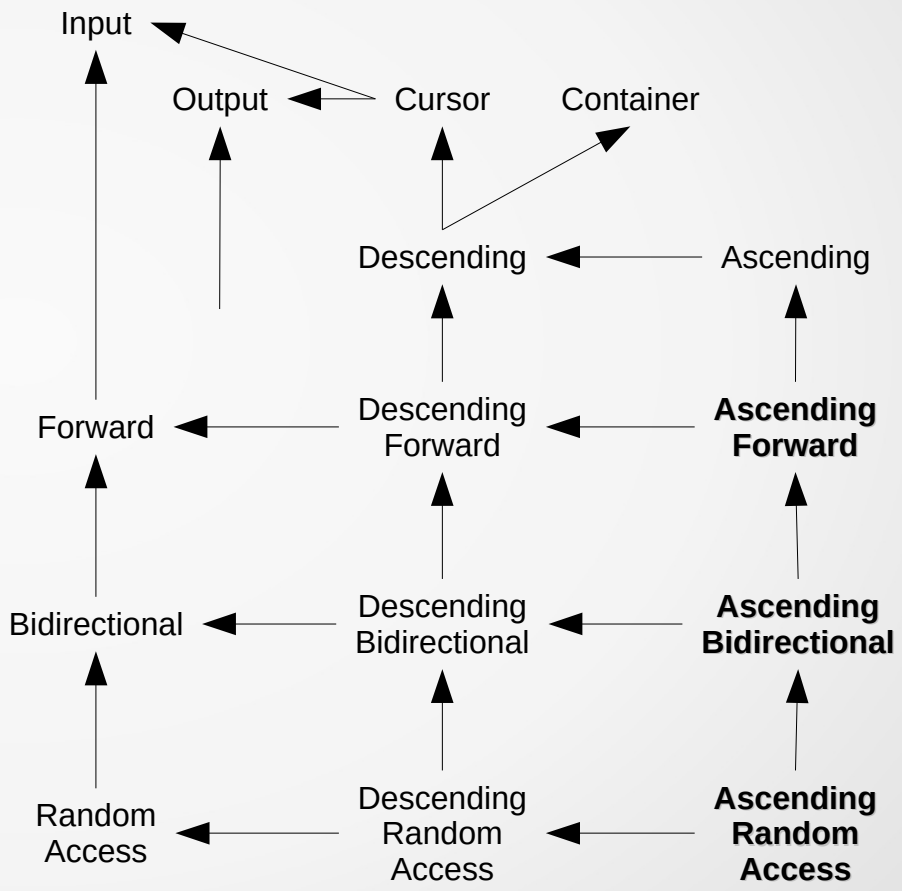
Design Choices, Cursors

- Ascending Cursor
  - Node parent
  - parent(), operator!()



Design Choices, Cursors

- Ascending Cursor
  - Node parent
  - parent(), operator!()
  - As descending cursors



Design Choices, Policies, Pros

- Simpler Customization

```
node_base:
  create_header()
  destroy_header()
  pre_rotate()
  pre_slice()
```

[austern]

Design Choices, Policies, Cons

- Type Duplication

```
Template <
  class Val, class Balance,
  class Key, class Extract,
  class Comp>
class search_tree
  : public key_ordered_type<
      Val, Key, Extract, Comp>
{
  public:

  typedef
    search_node<Val, Balance> Node;

  Node* begin();
}
```

[austern]

Design Choices, Policies, Cons

- Type Duplication
- Internal Knowledge

```
template <class Node>
inline void rotate(Node * q) {
  int c = q->parity;
  Node* p = q->parent;
  Node* B = q->child[!c];
  Node::pre_rotate(q);
  p->child[c] = B;
  B->parent = p;
  B->parity = c;
  q->parent = p->parent;
  q->parity = p->parity;
  q->parent->child[q->parity] = q;
  p->parent = q;
  p->parity = !c;
  q->child[!c] = p;
}

static void pre_rotate(Node* q) {
  Node* p = q->parent;
  if (q->parity == 0)
    p->left_size -= q->left_size;
  else
    q->left_size += p->left_size;
}
```

[austern]

Design Choices, Policies, Cons

- Type Duplication
- Internal Knowledge
- Limited Interface

```
Typedef
  binary_tree<
    std::string,
    rank_augment<
      binary_tree<std:string> >
    >
string_rank_tree;

// Where do we add rank operations?
// Free functions?

template <class R>
  std::string rank_is(
    R r,
    typename R::size_type n);
```

Design Choices, Adaptors

- Isolated Interface
  - Extended
  - Alternate

```
template <
  class T,
  class Hierarchy
    = binary_tree<T>
  >
class forest_tree;

template <
  class T,
  class Hierarchy
    = nary_tree< std::vector<T> >
  >
class multiway_tree;
```

Design Choices, Adaptors

- Isolated Interface
  - Extended
  - Alternate
- Augmenting
  - Rank

```
template <
  class T,
  Hierarchy = binary_tree<T>>
class rank_tree
{
  // ..

  cursor rank_is(size_type n);
  size_type rank_of(cursor c);

  // ..
}
```

Design Choices, Adaptors

- Isolated Interface
  - Extended
  - Alternate
- Augmenting
  - Rank
- Balancing
  - In-order invariance
  - Sequence

```
template < class T, class Hierarchy
  = binary_tree<T> >
class avl_tree;

template < class T, class Hierarchy
  = binary_tree<T> >
class red_black_tree;

template < class T, class Hierarchy
  = binary_tree<T> >
class splay_tree;

template < class T, class Hierarchy
  = binary_tree<T> >
class treap;

template < class T, class Hierarchy
  = multiway_tree<T> >
class b_tree;

template <class T, class Hierarchy
  = multiway_tree<T> >
class b_star_tree;
```