Presentation of Mini-project

for

Generic Programming and library development

2006

Presentation points:

- Assignment
- Data structure
- Smart pointers
- Complexity
- Possible Improvements and known bugs

Assignment:

- Data structure for meldable priority queue
- STL queue interface
- Additional functions (CPH STL)
  - Iterator
  - `increase()`, `meld()`, `delete()`
- Complexity (Element comparisons / operations performed)
- Performance

Data structure:

- Chosen structure: Binomial heap
  - Draw binomial heap
- Thoroughly documented
- Max-heap
- Interesting properties (Cormen et al.)
  - 3 pointers: Parent, sibling and child
  - Degree of binomial trees
- Show binomial heap slide

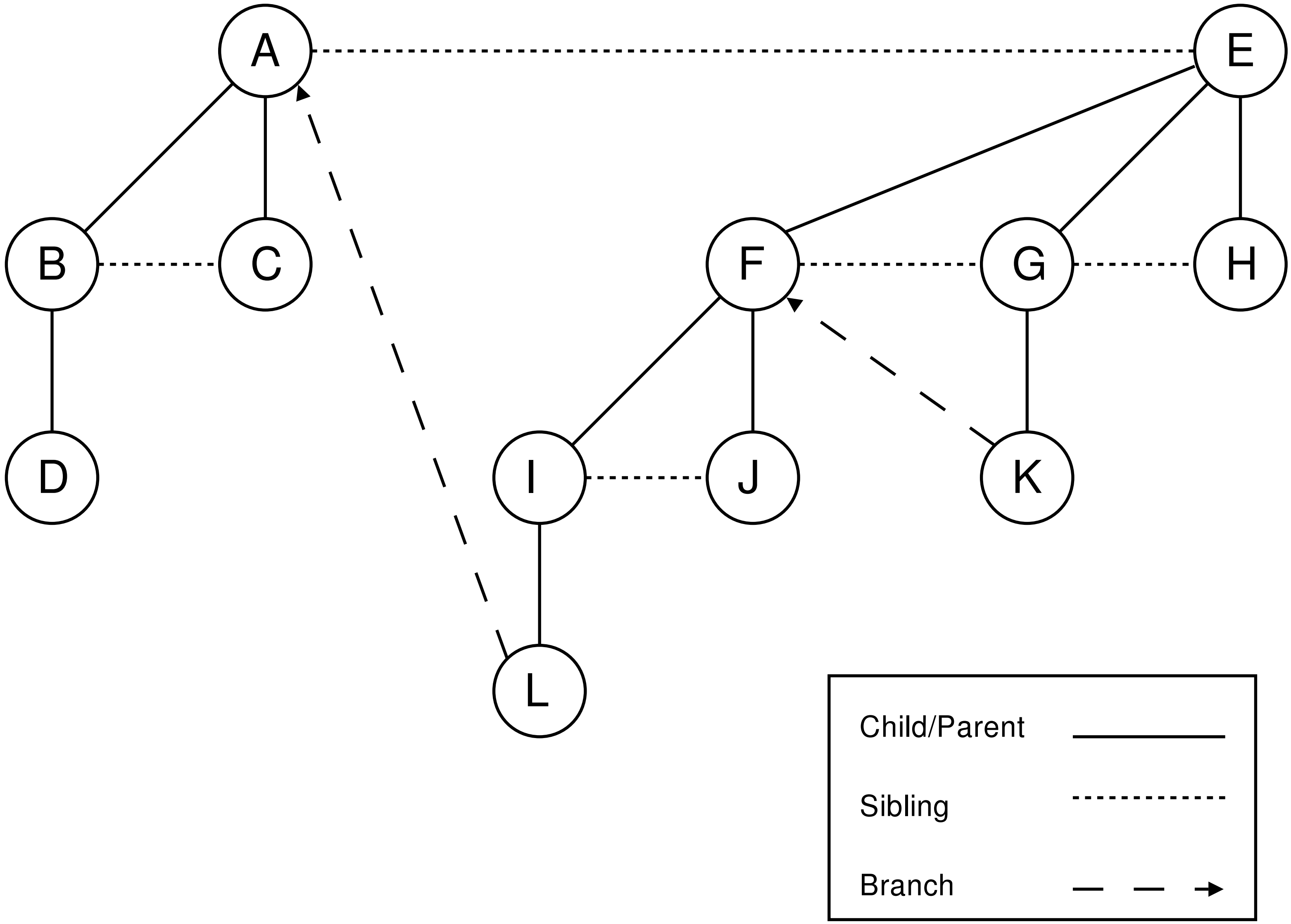

Figure 1: Binominal queue example.

Data structure cont.:

- Iteration
  - Post-order iteration
  - Constant time iteration Impossible with 3 pointers
  - Introduction of “previous branch”, “previous sibling”, “last child” and “leftmost left”
- Smart pointers
  - Shared pointers for structure necessary pointers
  - Weak for the rest (prevention of cyclic dependency)
  - Pro and con discussion later
- Show node structure slide

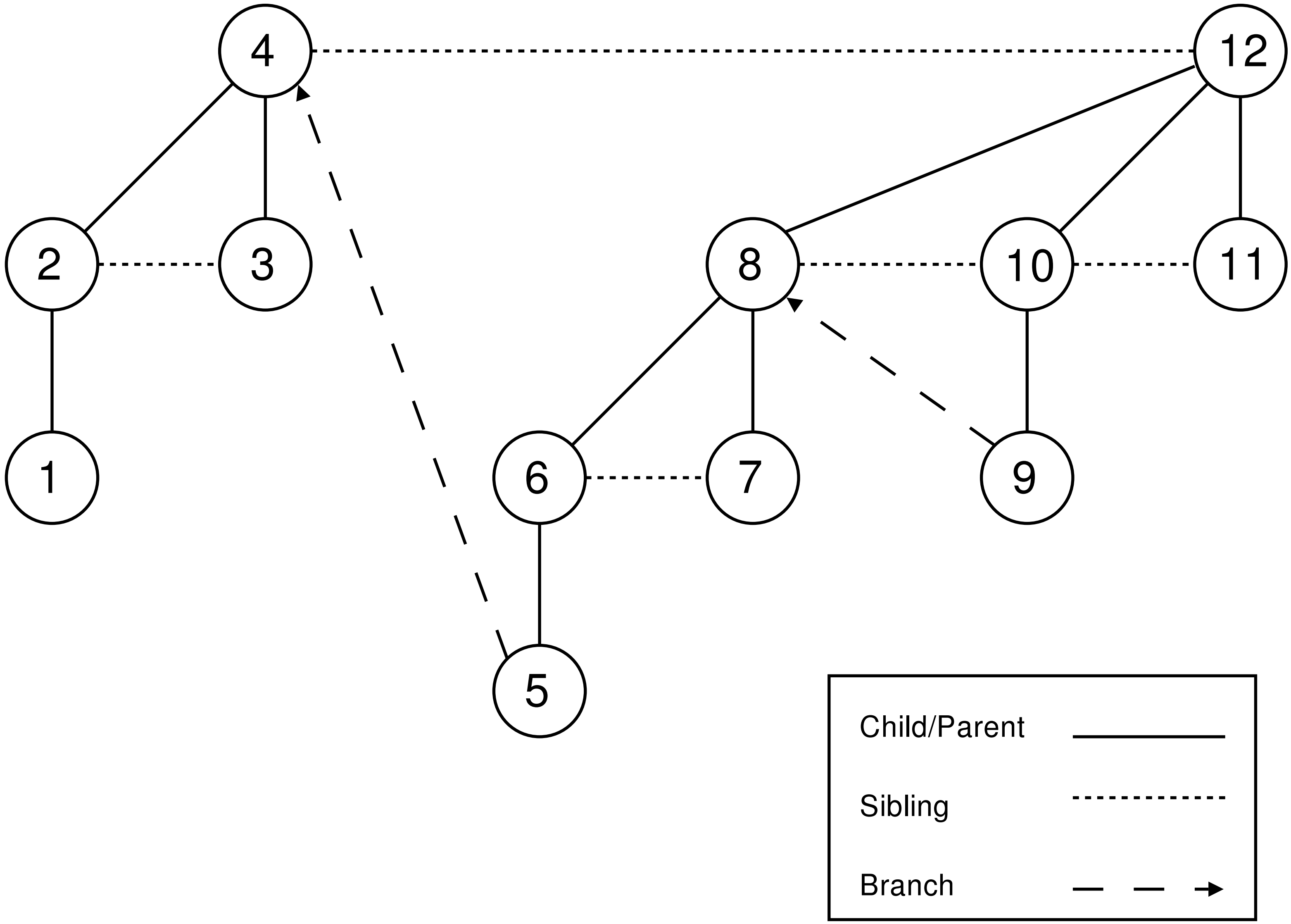

Figure 2: Binominal queue example numbered in post-order iteration.

Iteration rules:

- Forward iteration:
  - If there is a next sibling, visit the left-most leaf of that sibling.
  - Otherwise, visit the parent.
- Reverse iteration:
  - Visit the last child.
  - If there is no last child, visit the previous sibling.
  - If there is no previous sibling, visit the branch.

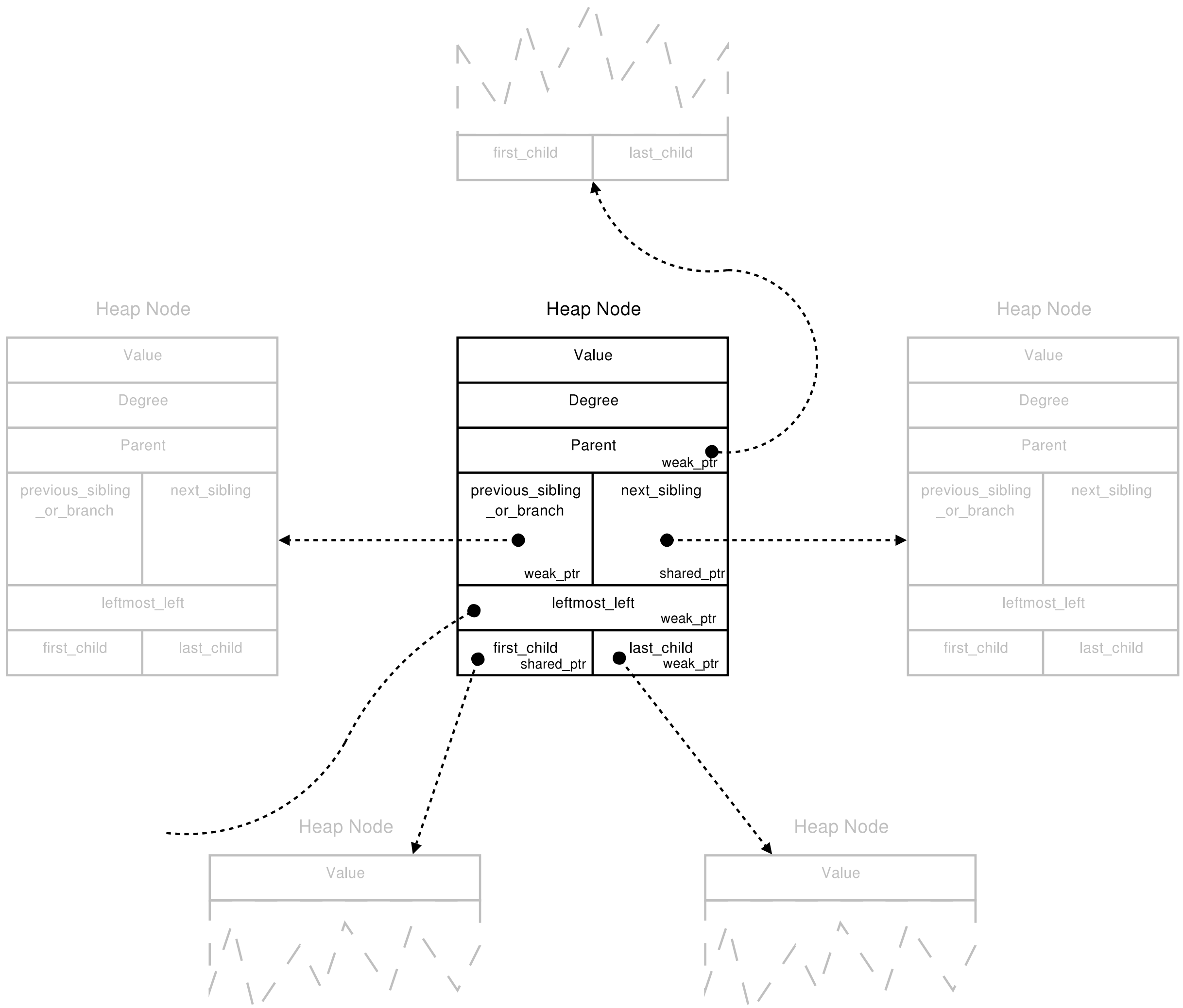

Figure 3: Visualization of heap node and some of its content.

Data structure cont.:

- Binomial heap implementation details
  - 2 sentinels: shared pointer to “head”, weak pointer “one past the end”.
  - Counter of nr. of elements
  - Pointer to maximum element
- Entire implementation is template based—STL standard
- Use for generic libraries

Smart pointers:

- Pro
  - Weak pointer invalidation
  - Automated Deletion
- Con
  - Space Usage
  - Performance overhead
  - Complexity
    - Shared and weak
    - Accessing
    - Constructor—`shared_from_this`
  - Debugging

Complexity:

- `find-max()`—0 comparisons. $O(1)$
- `insert()`—$\log(n)$ comparisons. $O(\log(n))$
  - New node instantly inserted at front of heap
  - Worst case: each binomial tree will be re-linked
- `delete()`—$2 * \log(n)$ comparisons. $O((\log(n))^2)$?
  - Shifting up requires no element comparisons.
  - Melding will cost $\log(n)$ comparisons.
  - Finding a new max will cost $\log(n)$ comparisons.
- General constructor—$n * \log(n)$ comparisons. $O(n * \log(n))$
  - Iteration through given heap and insertion of each element.
  - Improved version exists to $2n$.
- Iterator functions—0 comparisons. $O(1)$

Possible improvements and known bugs:

- Exceptions
- Normal pointers instead of “smart” pointers
- Improving general constructor complexity
  - Allowing two binomial trees of equal degree to exist
  - Balancing workload.
- Consolidate error
  - May invalidate top pointer. No longer points to root element.
  - One pointer check will solve this
  - line 275 ... `and (x != top_.lock())`
- More extensive unit test and a performance test