Garbage Collection

What is garbage collection?

- Automatic recycling of allocated heap memory that can not be used again
  - i.e., is garbage
  - i.e., is not reachable (from roots)
  - Convenient
  - Avoids memory leaks (usually)
  - Required for type safety

Java can leak memory

```
// Find the error
   class Stack {
     Object a[] = new Object[10];
     int top = -1;
     Object pop() {
          if (top == -1)
                 throw new NoSuchElementException();
          return a[top--];
          }
   void push(Object o) {
          if (top+1 == a.length)
                 throw new StackOverflowException();
          a[++top] = o;
          }
   }
```


Reachable != will be used again

```
// Error fixed in green
   class Stack {
     Object a[] = new Object[10];
     int top = -1;
     Object pop() {
          if (top == -1)
                 throw new NoSuchElementException();
          Object r = a[top];
          a[top--] = null;
          return r;
          }
   void push(Object o) {
          if (top+1 == a.length)
                 throw new StackOverflowException();
          a[++top] = o;
          }
   }
```

Finding pointers can be hard

- What are root pointers?
  - Pointers that can be accessed by program, even though no pointers point to them
    - For Java:
      - all local and stack variables of all threads
      - all loaded classes (class unloading?)
      - static variables of loaded classes
      - What is the type of a Java stack variable?
        - Need stack map, from PC to local/stack variable types
        - Can also take into account dead references

5

Finding pointers in objects can be hard

- Type unsafe languages are a nightmare
  - variant records (unions)
  - pointers into middle of objects
  - various bit-twiddling tricks
  - Even in typesafe languages
    - need run-time access to type information for objects

6

Reference counting

- Different approach to garbage collection
- For each object, keep track of number of references to it
- Cost to adjust counts (some optimization possible)
  - {Object tmp = a; a = b; b = tmp}
  - generates:
    a.refCount++; tmp = a;
    b.refCount++; a.refCount--; a = b;
    tmp.refCount++; b.refCount--; b = tmp;
    tmp.refCount--;
- Can't reclaim circular structures
- Use only in special circumstances
  – not generally recommended

7

Mark and Sweep [McCarthy 1960]

- From roots
  - Roots are pointers known to be accessible
    - e.g., registers, stack
- perform a depth-first search to mark live nodes
- Sweep through all memory
  - if node is unmarked, it is garbage, put on free list
  - if marked, is in use, unmark
    - to prepare it for next garbage collection

8

Problems with Mark and Sweep

- Many of these are problems with other garbage collection schemes
  - Stack required for DFS
    - could be big
  - Finding roots
  - Finding pointers
    - Easy for CONS cells
  - Stops the world

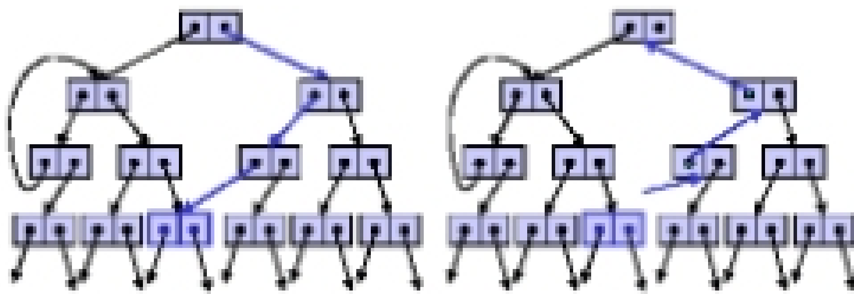
9

Pointer Reversal

- In depth first search, need stack as big as depth of data structure
  - for a long list, depth == length
- Often, garbage is being collected when we are short on space
- Can avoid storing DFS stack separately using pointer reversal
  - Efficient way to implement mark and sweep

10

Doing pointer reversal



11

Is pointer reversal worthwhile?

- Requires additional bits
  - log f bits, where f is number of pointer fields
- Requires additional visits and modifications of each node
- Generally not worth while
  - Instead, can do tail-call optimization for last field from object
  - when following last field out of an object, don't put object on stack for a return visit
  - helps a little

12