Unobtrusive Garbage Collection

1 Unobtrusive Garbage Collection

**GC Requirements:**

- **batch programs:** We want short total GC time.
- **interactive programs:** We want unnoticable GCs.

**Unobtrusive GC:**

**Incremental Collection**

- Do a little GC-work every time an object is allocated, or a pointer is changed.

**Concurrent Collection**

- Run the collector and the program in different processes, or on different processors.

2 Incremental GC

- Use copying collection, but rather than stop when you run out of memory and then do all the GC work in one shot, do a little bit whenever a pointer variable is referenced or when a new object is allocated.
- We start out by forwarding (copying) the objects pointed to by global variables.
- Then, instead of continuing forwarding recursively, we resume the program.
- Every time a pointer is referenced we check to see whether it is pointing into from-space. If it is, we forward that object too.
3 Incremental GC...

- Even objects which are not explicitly referenced have to be checked, to see if they have become garbage. Therefore, every time we allocate a new object we forward $k$ pointers.
  A good value for $k$ has to be determined by experimentation.
- Eventually scan will catch up with next and we switch from-space and to-space and start an new cycle.
- Baker’s algorithm (on the next slide) is a variant of copying collection.

4 Incremental GC...

1. Copy and update objects pointed to by global pointers to to-space.
2. Resume program.
3. When an object in from-space is referenced, first copy it to to-space.
   
   $p := x \uparrow .next;
   \Downarrow \ (implemented \ as)
   $\text{IF } x \in \text{from-space } \text{THEN}
   \begin{align*}
   &\text{copy } x \text{ to to-space;}
   &\text{update } x, \text{ scan, and next;}
   &x := x \text{'s new address in to-space;}
   \end{align*}
   $\text{END;}
   
   $p := x \uparrow .next;$
4. Every time NEW is called, $k$ pointers are forwarded.

Cost of Garbage Collection

5 Cost of Garbage Collection

- The size of the heap is $H$, the amount of reachable memory is $R$, the amount of memory reclaimed is $H - R$.
- What is the cost of the different GC algorithms?

\[
\text{amortized GC cost} = \frac{\text{time spent in GC}}{\text{amount of garbage collected}} = \frac{\text{time spent in GC}}{H - R}
\]
6 Cost of GC — Mark-and-Sweep

- The mark phase touches all live nodes. Hence, it takes time $c_1 H$, for some constant $c_1$. $c_1 \approx 10$.
- The sweep phase touches the whole heap. Hence, it takes time $c_2 R$, for some constant $c_2$. $c_2 \approx 3$.

$$GC \ cost = \frac{c_1 R + c_2 H}{H - R} \approx \frac{10R + 3H}{H - R}$$

7 Cost of GC — Mark-and-Sweep...

- If $H \approx R$ we reclaim very little, and the cost of GC goes up. In this case the GC should grow the heap (increase $H$).

8 Cost of GC — Copying Collection

- The breadth first search phase touches all live nodes. Hence, it takes time $c_3 R$, for some constant $c_3$. $c_3 \approx 10$.
- The heap is divided into a from-space and a to-space, so each collection reclaims $\frac{H}{T} - R$ words.

$$GC \ cost = \frac{c_3 R}{\frac{H}{T} - R} \approx \frac{10R}{\frac{H}{T} - R}$$

9 Cost of GC — Copying Collection...

- If there are few live objects ($H \gg R$) the GC cost is low.
• If $H = 4R$, we get

$$GC\ cost = \frac{c_3R}{4R^2 - R} \approx 10.$$  

This is expensive: 4 times as much memory as reachable data, 10 instruction GC cost per object allocated.

10 Cost of GC — Generational Collection

- Assume the youngest generation ($G_0$) has 10% live data, i.e. $H = 10R$.
- Assume we’re using copying collection for $G_0$.

$$GC\ cost_{G_0} = \frac{c_3R}{H - R} = \frac{c_3R}{10R - R} \approx \frac{10R}{4R} = 2.5$$

11 Cost of GC — Generational Collection...

- If $R \approx 100$ kilobytes in $G_0$, then $H \approx 1$ megabyte.
- In other words, we’ve wasted about 900 kilobytes, to get 2.5 instruction/word GC cost (for $G_0$).

12 Exam Problem

1. Why is generational collection more appropriate for functional and logic languages (such as LISP and Prolog), than for object-oriented languages (such as Eiffel and Modula-3)?

2. The heap in the figure on the next slide holds 7 objects. All objects have one integer field and one or two pointer fields (black dots). The only roots are the three global variables X, Y, and Z. Free space is shaded. Show the state of To-Space after a copying garbage collection has been performed on From-Space. Note that several answers are possible, depending on the visit strategy (Depth-First or Breadth-First Search) you chose.
13 Exam Problem I.

14 Exam Problem.

1. Name five garbage collection algorithms!
2. Describe the Deutsch-Schorr-Waite algorithm! When is it used? Why is it used? How does it work?
3. What are the differences between stop-and-copy, incremental and concurrent garbage collection? When would we prefer one over the other?

15 Readings and References

- Read Scott, pp. 395–401.
- Aho, Hopcroft, Ullman. Data Structures and Algorithms, Chapter 12, Memory Management.

16 Readings and References...