Garbage Collection

Before garbage collection:

Free list: 10 \rightarrow 11 \rightarrow 14 \rightarrow 13 \rightarrow 12

Garbage not reachable

Free list: 10 \rightarrow 11 \rightarrow 14 \rightarrow 13 \rightarrow 12

Mark bit

Back bit

application program variable

Note Title: Garbage Collection

Free list

<table>
<thead>
<tr>
<th>0</th>
<th>10</th>
<th>-</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>9</td>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>11</td>
<td>-</td>
</tr>
<tr>
<td>11</td>
<td>12</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>13</td>
<td>-</td>
</tr>
<tr>
<td>13</td>
<td>14</td>
<td>-</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

left right

null
How can you efficiently find garbage so you can put it on the free list?

We can find what program can reach and then rest is garbage.

Simple idea: reference counts

Could we just keep track of # refs to a cell?
If ref count is 0 then cell is garbage and can be placed on free list.

What's the problem?

Some overhead to maintain bigger problem.
breadth-first search

depth-first search

Search can both find all vertices reachable from a given source.
Run garbage collection alg when there's nothing on free list.

Important to minimize space used by garbage collection alg.
Mark & Sweep Algorithm

Phase 1: Mark all reachable cells such that discovered = true means marked

Phase 2: Sweep (go through) all of memory & any cell unmarked is placed on front of free list
Should we use BFS or DFS to minimize space?

Queue (BFS) or stack (DFS) need enough space to hold a queue/stack as big as memory.
Use DFS where stack is held within the memory!
The diagram illustrates a stack operation with the following elements:

- **Stack**: c, b, a
- **2nd Stack**: d, c, b
- **Top Stack**: d, c, b
- **2nd on Stack**: d, c, b
- **Stack Label**: XR
- **Top Label**: XL

Arrows indicate the movement of elements between the stacks.
Restore memory to how it began

Stack 6, 5, 4, 3, 2, 1
Summary

Space for in-place DFS
is 2 bits per memory cell
(mark bit, back bit) + 3

no external stack

U/R

global vars (top, pred, temp)
Time complexity: $O(M + A)$

$M$ be the # memory cells
$A$ be the # of accessible (reachable) cells

$M - A$ cells of garbage

Mark Phase: $O(A)$ each accessible cell has at most 2 edges
Sweep Phase: $O(M)$
Cell ptr = null;
free(ptr);
free(ptr+1);
free(ptr+2);
free(ptr+3);

cell ptr = new cell();
ptr = null;
ptr = 10;
ptr = 10;
Can we avoid sweep phase?
Can we design an $O(n)$ garbage collection algorithm.

Move all accessible cells somewhere else & what remains is garbage.
Copying collection

Use in-place DFS
divide memory in half

From
Phase 1: Move reachable cells to other half
“from” in use
“to” other half
+ leave forwarding address

Phase 2: Update all reference in to half using forwarding address
Then switch to + from
For copying collection:

\[
\begin{align*}
\text{Time Complexity} & \quad O(A) \\
\text{Cost of garbage collection} & \quad = \quad O\left(\frac{A}{M^{1/2} - A}\right) \\
\text{# cells freed} & \quad = \quad O\left(\frac{A + M}{M - A}\right)
\end{align*}
\]