Stop-and-copy garbage collection


The idea is copy a graph from one region of the memory to another, unused region, in such a way that the resulting copy has been compressed into a contiguous region. The algorithm works as follows. Suppose we begin with this graph:

Notice that not all cells are accessible from the ROOT reference. Two pointers are maintained into the new region. NEXT refers to the next open location for copying. SCAN refers to next cell to be processed. Start by copying the root cell into the new region. After copying, place a mark in the old cell to indicate that it has been copied, and store in the old cell a forwarding
The algorithm continues until SCAN catches up with NEXT. Look at the word referenced by SCAN. If it contains a pointer, look at the cell it points to. If the cell contains a forwarding reference, update SCAN↑ with that reference. If not, copy the cell into the new region at location NEXT, incrementing NEXT to the next empty space. Then advance SCAN TO the next word.
The figure below shows the situation after a few more steps:

Figure ?? shows a version of the algorithm written in C by David Stern.
```c
#include <stdio.h>
#define NUMCELLS 50 /* number of cells N */

struct cell {
    cell() {}
    cell(int _p1, int _p2, int _d): p1(_p1), p2(_p2), d(_d), mark(0) {}
    int mark;
    int p1;
    int p2;
    int d;
};

cell from[NUMCELLS]; /* "from" is the input, and "to" is the output */
cell to[NUMCELLS];
int next;
int scan;

int forward(int n)
{
    if (from[n].mark) /* if this node has already been copied */
        return from[n].p1; /* the pointer was saved */
    else {
        to[next] = from[n]; /* copy the node over */
        from[n].p1 = next; /* save pointer to the "to" space */
        from[n].mark = 1;
        return next++;
    }
}
```
void gc()
{
    scan = 0;
    next = 0;
    forward(0); /* forward the root */
    while (scan < next) {
        to[scan].p1 = forward(to[scan].p1);
        to[scan].p2 = forward(to[scan].p2);
        scan++;
    }
}

main()
{
    cell* p = from;
    cell* q = to;
    p[0] = cell(12,17,1);
    p[12] = cell(12,12,2);
    p[17] = cell(12,0,3);
    gc();
    for (int i=0;i<next;i++) {
        printf("to[%d] = (%d,%d,%d)\n",i,q[i].p1,q[i].p2,q[i].d);
    }
}

Figure 1: A stop-and-copy collector written by David Stern (spring 1999).