1. Intro. OK, you've heard about SIGGRAPH; what's this?

GRAPH-SIG is an experimental program to find potential equivalence classes in automorphism testing. Given a graph G and a vertex v_0, we compute "signatures" of all vertices such that, if there's an automorphism that fixes v_0 and takes v to v', then v and v' will have the same signature.

I plan to generalize the idea, but in this test case I just proceed as follows: First I compute level 0 signatures, which are just the distances from v_0. Then, given level k signatures σ_k, I compute signatures $\sigma_{k+1}(v) = \prod_{u \text{---} v}(x - \sigma_k(u))$, where x is a random integer and the multiplication is done mod 2^{64}. We keep going until reaching a round where no class is further refined.

My tentative name for these signatures is "lookahead invariants."

(Notes for the future: If there's an automorphism that takes v_0 into v_0', then the multiset of signatures computed with respect to v_0 will be the same as the multiset computed with respect to v_0', after each round. Also we can generalize to automorphisms that fix k vertices, by defining level 0 signatures as the ordered sequence of distances from v_0, …, v_{k-1}. Universal hashing schemes conveniently map such an ordered sequence into a single number.)

```
#define maxn 100   /* upper bound on vertices in the graph */
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "gb_graph.h"
#include "gb_save.h"
#include "gb_flip.h"
  long sg[maxn];   /* new signatures found in current class */
  Vertex *hd[maxn], *tl[maxn];   /* subdivisions of current class */
  main(int argc, char *argv[])
  {
    register int i, j, k, r, change;
    register Graph *g;
    register Vertex *u, *v;
    register Arc *a, *b;
    register long x, s;
    Vertex *v0, *prev, *head;
    ⟨Process the command line 2⟩;
    ⟨Make the initial signatures 3⟩;
    for (change = 1, r = 1; change; r++) {
      change = 0;
      ⟨Do round r 5⟩;
    }
  }
```

2. ⟨ Process the command line 2 ⟩ ≡

```
if (argc ≠ 3) {
  fprintf(stderr, "Usage:␣%s␣foo.gb␣v0\n", argv[0]);
  exit(−1);
}
g = restore_graph(argv[1]);
if (¬g) {
  fprintf(stderr, "I␣couldn't␣reconstruct␣graph␣%s!\n", argv[1]);
  exit(−2);
}
if (g→n > maxn) {
  fprintf(stderr, "Recompile␣me:␣g->n=%ld,␣maxn=%d!\n", g→n, maxn);
  exit(−3);
}
gb_init_rand(0);   /* the seed doesn't matter much */
for (v = g→vertices; v < g→vertices + g→n; v++)
  if (strcmp(v→name, argv[2]) ≡ 0) break;
if (v ≡ g→vertices + g→n) {
  fprintf(stderr, "I␣can't␣find␣a␣vertex␣named␣'%s'!\n", argv[2]);
  exit(−9);
}
v0 = v;
```

This code is used in section 1.

3. Vertices with the same signature are linked cyclically. As mentioned above, we start by simply computing distances from v_0.

```
#define sig w.I   /* signature of a vertex */
#define link u.V   /* link field in a circular list */
#define tag v.I   /* to what extent have we processed the vertex? */
```

⟨ Make the initial signatures 3 ⟩ ≡

```
printf("Initial␣round:\n");
for (v = g→vertices; v < g→vertices + g→n; v++) v→sig = −1, v→tag = 0;
v0→sig = 0, v0→link = v0, k = 1, v = v0;
while (v) {
  prev = head = Λ;
  while (1) {
    printf("␣%s␣dist␣%ld\n", v→name, v→sig);
    ⟨ Set signature of all v's unseen neighbors to k 4 ⟩;
    v→tag = k;
    v = v→link;
    if (v→tag) break;
  }
  if (prev ≡ Λ) break;   /* all vertices reachable from v0 have been seen */
  head→link = prev;   /* close the cycle */
  v = prev, k++;
}
```

This code is used in section 1.

4. ⟨ Set signature of all v's unseen neighbors to k 4 ⟩ ≡

```
for (a = v→arcs; a; a = a→next) {
  u = a→tip;
  if (u→sig < 0) {
    u→sig = k;
    if (prev ≡ Λ) head = u;
    else u→link = prev;
    prev = u;
  }
}
```

This code is used in section 3.

5. Now comes the fun part. As we pass from σ_{r-1} to σ_r, each equivalence class becomes one or more classes.

#define *oldsig* *z.I*

⟨ Do round r 5 ⟩ ≡

```
printf("Round␣%d:\n", r);
for (v = g→vertices; v < g→vertices + g→n; v++) v→oldsig = v→sig;
k++;     /* k is a unique stamp to identify this round */
x = (gb_next_rand() ≪ 1) + 1;     /* pseudorandom number used for new signatures */
for (v = g→vertices; v < g→vertices + g→n; v++)
  if (v→tag > 0) {
    if (v→tag ≡ k) continue;
    if (v→link ≡ v) {
      printf("␣%s␣is␣fixed\n", v→name);     /* class of size 1 */
      v→tag = −k;     /* we needn't pursue it further */
      continue;
    }
    for (j = 0; v→tag ≠ k; v = u) {
      u = v→link;
      ⟨ Compute s = σ_r(v) 6 ⟩;
      printf("␣%s␣%lx\n", v→name, s);
      v→sig = s;
      for (i = 0, sg[j] = s; sg[i] ≠ s; i++) ;
      if (i ≡ j) hd[j] = tl[j] = v, j++;     /* a new cyclic list begins */
      else v→link = tl[i], tl[i] = v;     /* continue building an existing list */
      v→tag = k;
    }
    for (i = 0; i < j; i++) hd[i]→link = tl[i];     /* complete the cycles */
    if (j > 1) change = 1;
  }
```

This code is used in section 1.

6. ⟨ Compute $s = \sigma_r(v)$ 6 ⟩ ≡

```
for (s = 1, a = v→arcs; a; a = a→next) s *= x − a→tip→oldsig;
```

This code is used in section 5.

7. Index.

$\langle$ Compute $s = \sigma_r(v)$ 6 $\rangle$ Used in section 5.
$\langle$ Do round r 5 $\rangle$ Used in section 1.
$\langle$ Make the initial signatures 3 $\rangle$ Used in section 1.
$\langle$ Process the command line 2 $\rangle$ Used in section 1.
$\langle$ Set signature of all v's unseen neighbors to k 4 $\rangle$ Used in section 3.

GRAPH-SIG-V0