1. Intro. This program was written (somewhat hastily) in order to experiment with sandpiles.

The first command line argument is the name of a file that specifies an undirected graph in Stanford GraphBase SAVE_GRAPH format; the graph may have repeated edges, but it must not contain loops. It should be connected. It shouldn't have more than 100 vertices. I don't check these assumptions.

An optional second argument is the number of the root vertex.

```
#include "gb_graph.h"
#include "gb_save.h"
  ⟨ Preprocessor definitions ⟩
  int vec[1000][1000];
  int x[1000], d[1000], t[1000];
  int n, r;
  ⟨ Subroutines 4 ⟩
  main(int argc, char *argv[])
  {
    register int j, k;
    Vertex *v;
    Arc *a;
    Graph *g;
    ⟨ Input the graph 2 ⟩;
    ⟨ Prepare the vec table 3 ⟩;
    ⟨ Reduce the vector d 5 ⟩;
  }
```

2. ⟨ Input the graph 2 ⟩ ≡

```
  if (argc < 2) {
    fprintf(stderr, "Usage:␣%s␣foo.gb␣[r]\n", argv[0]);
    exit(1);
  }
  g = restore_graph(argv[1]);
  if (¬g) {
    fprintf(stderr, "Sorry,␣can't␣create␣the␣graph␣from␣file␣%s!␣(error␣code␣%d)\n", argv[1],
        panic_code);
    exit(−1);
  }
  n = g→n;
  if (argc > 2) sscanf(argv[2], "%d", &r);
```

This code is used in section 1.

3. ⟨Prepare the *vec* table 3⟩ ≡

```
for (j = 0; j < n; j++) {
  v = g→vertices + j;
  for (a = v→arcs; a; a = a→next) {
    k = a→tip − g→vertices;
    d[j]++;
    vec[j][k]−−;
  }
  vec[j][j] = d[j];
}
if (r) {
  for (j = 0; j < n; j++) k = vec[0][j], vec[0][j] = vec[r][j], vec[r][j] = k;
  for (j = 0; j < n; j++) k = vec[j][0], vec[j][0] = vec[j][r], vec[j][r] = k;
  k = d[0], d[0] = d[r], d[r] = k;
}
```

This code is used in section 1.

4. The *reduce* subroutine topples a given vector x until it is stable.

⟨Subroutines 4⟩ ≡

```
void reduce( )
{
  register int j, k, h;
  while (1) {
    h = 0;
    for (j = 1; j < n; j++)
      if (x[j] ≥ d[j]) {
        h = 1;
        for (k = 1; k < n; k++) x[k] −= vec[j][k];
      }
    if (h ≡ 0) break;
  }
}
```

This code is used in section 1.

5. ⟨Reduce the vector d 5⟩ ≡

```
printf("The␣d␣vector␣is");
for (j = 1; j < n; j++) {
  x[j] = d[j];
  printf("␣%d", x[j]);
}
printf("\n␣and␣it␣reduces␣to");
reduce();
for (j = 1; j < n; j++) {
  printf("␣%d", x[j]);
  x[j] = d[j] − x[j];
}
printf("\nThe␣t␣vector␣is");
reduce();
for (j = 1; j < n; j++) {
  printf("␣%d", x[j]);
  x[j] = d[j] + d[j];
}
reduce();
printf("\nThe␣double-d␣vector␣reduces␣to");
for (j = 1; j < n; j++) {
  printf("␣%d", x[j]);
  x[j] = d[j] + d[j] − x[j];
}
reduce();
printf("\n␣and␣the␣zero␣vector␣is");
for (j = 1; j < n; j++) {
  printf("␣%d", x[j]);
}
printf("\n");
```

This code is used in section 1.

6. Index.

⟨ Input the graph 2 ⟩ Used in section 1.
⟨ Prepare the *vec* table 3 ⟩ Used in section 1.
⟨ Reduce the vector d 5 ⟩ Used in section 1.
⟨ Subroutines 4 ⟩ Used in section 1.

SAND