

(Downloaded from <https://cs.stanford.edu/~knuth/programs.html> and typeset on May 28, 2023)

**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>  $\equiv$

```
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.  $\langle$  Prepare the *vec* table 3  $\rangle \equiv$

```

for ( $j = 0; j < n; j++$ ) {
     $v = g\text{-vertices} + j$ ;
    for ( $a = v\text{-arcs}; a; a = a\text{-next}$ ) {
         $k = a\text{-tip} - g\text{-vertices}$ ;
         $d[j]++$ ;
         $vec[j][k]--$ ;
    }
     $vec[j][j] = d[j]$ ;
}
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.

$\langle$  Subroutines 4  $\rangle \equiv$

```

void reduce()
{
    register int  $j, k, h$ ;
    while (1) {
         $h = 0$ ;
        for ( $j = 1; j < n; j++$ )
            if ( $x[j] \geq d[j]$ ) {
                 $h = 1$ ;
                for ( $k = 1; k < n; k++$ )  $x[k] -= vec[j][k]$ ;
            }
        if ( $h \equiv 0$ ) break;
    }
}

```

This code is used in section 1.

5.  $\langle \text{Reduce the vector } d \text{ 5} \rangle \equiv$

```

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.**

*Arc*: [1](#).  
*arcs*: [3](#).  
*argc*: [1](#), [2](#).  
*argv*: [1](#), [2](#).  
*d*: [1](#).  
*exit*: [2](#).  
*fprintf*: [2](#).  
*Graph*: [1](#).  
*h*: [4](#).  
*j*: [1](#), [4](#).  
*k*: [1](#), [4](#).  
*main*: [1](#).  
*n*: [1](#).  
*next*: [3](#).  
*panic\_code*: [2](#).  
*printf*: [5](#).  
*r*: [1](#).  
*reduce*: [4](#), [5](#).  
*restore\_graph*: [2](#).  
*sscanf*: [2](#).  
*stderr*: [2](#).  
*t*: [1](#).  
*tip*: [3](#).  
*vec*: [1](#), [3](#), [4](#).  
*Vertex*: [1](#).  
*vertices*: [3](#).  
*x*: [1](#).

- ⟨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

	Section	Page
Intro .....	<a href="#">1</a>	1
Index .....	<a href="#">6</a>	4