

```
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <sys/types.h>
#include <time.h>
```

LIBRARIES AND DEFINITIONS

```
#define NodesMax 20000
#define M 2
#define N 1000
#define MATRIX_A 0x9908b0dfUL /* constant vector a */
#define UPPER_MASK 0x80000000UL /* most significant w-r bits */
#define LOWER_MASK 0x7fffffffUL /* least significant r bits */

static unsigned long mt[N]; /* the array for the state vector */
static int mti=N+1; /* mti==N+1 means mt[N] not initialized*/
```

```
main(int argc, char *argv[]) {
    void init_genrand(unsigned long s); /* this is the seed of random
number */
    unsigned long genrand_int32(void);
    double genrand_real2(void), rnddest;
    int i, j, k, edges=0, nodes=N, m=M, m0, degree[NodesMax],
winner[NodesMax], windeg[NodesMax];
    int totdeg, totnds, found, DegDistr[NodesMax], maxdeg=2;
    FILE *fout;
    time_t rawtime;
    struct tm *info;

    time(&rawtime);
    info = gmtime(&rawtime);
    j = info->tm_min;
    k = info->tm_sec;
    for (i=1; i<j+k; i++) rnddest = genrand_real2();

    /* open file */
    fout = fopen ("BAedges.txt", "w");
    if (fout==NULL) { printf("no output file\n"); exit(-1); }

    if (argc>2) sscanf(argv[2], "%d", &m);
    if (argc>1) sscanf(argv[1], "%d", &nodes);
    if (nodes > NodesMax) {
        printf("Error: too large network, increase NodeMax the header of the
program\n");
        exit(0);
    }
    for(i=m; i<nodes; i++) DegDistr[i]=0;
```

DECLARATIONS AND INITIALIZATION

```

m0 = m+m+1;
for (i=0; i<m0; i++) {
    for (j=i+1; j<m0; j++) {
        edges++;
        fprintf(fout, "%d %d\n", i, j);
    }
    degree[i] = m+m;
}
totdeg = 2*m*m0;
totnds = m0;

```

INITIAL GRAPH

```

/* main loop for creating edges */
for(i=m0; i<nodes; i++) {
    edges += m;
    degree[i] = m;
    found = 0;
    for(j=0; j<m; j++) {
        rnddest = genrand_real2()*totdeg;
        for(k=0; k<totnds & found<j+1; k++)
            if ((rnddest <= degree[k]) & (degree[k] > 0)) {
                found++;
                windeg[j] = degree[k];
                winner[j] = k;
                fprintf(fout, "%d %d\n", i, k);
                totdeg = totdeg-degree[k];
                degree[k] = 0;
            }
        else rnddest = rnddest-degree[k];
    }
    for(j=0; j<m; j++) {
        degree[winner[j]] = windeg[j]+1;
        totdeg += windeg[j];
    }
    totdeg += (2*m);
    totnds++;
}

```

MAIN LOOP

```

fclose(fout);

/* final reports */
printf("produced %d edges, and %d\n node degree\n",edges,totdeg);
for(i=0; i<nodes; i++) {
    printf(" %5d %5d \n", i, degree[i]);
    if (degree[i]>maxdeg) maxdeg = degree[i];
    DegDistr[degree[i]]++;
}
printf("Graph Degree Distribution\n degree number of nodes\n");
for (i=0; i<=maxdeg; i++) if (DegDistr[i]>0)
    printf(" %5d      %5d\n", i, DegDistr[i]);
}

```

CLOSING

```

/*----- Additional funtions -----*/
void init_genrand(unsigned long s) { RANDOM NUMBER GENERATOR

    mt[0]= s & 0xffffffffUL;
    for (mti=1; mti<N; mti++) {
        mt[mti] = (1812433253UL * (mt[mti-1] ^ (mt[mti-1] >> 30)) + mti);
        mt[mti] &= 0xffffffffUL;
    }
}

unsigned long genrand_int32(void){
    unsigned long y;
    static unsigned long mag01[2]={0x0UL, MATRIX_A};
    if (mti >= N) {
        int kk;

        if (mti == N+1) init_genrand(5489UL);

        for (kk=0;kk<N-M;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+M] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        for (;kk<N-1;kk++) {
            y = (mt[kk]&UPPER_MASK)|(mt[kk+1]&LOWER_MASK);
            mt[kk] = mt[kk+(M-N)] ^ (y >> 1) ^ mag01[y & 0x1UL];
        }
        y = (mt[N-1]&UPPER_MASK)|(mt[0]&LOWER_MASK);
        mt[N-1] = mt[M-1] ^ (y >> 1) ^ mag01[y & 0x1UL];
        mti = 0;
    }

    y = mt[mti++];
    y ^= (y >> 11);
    y ^= (y << 7) & 0x9d2c5680UL;
    y ^= (y << 15) & 0xefc60000UL;
    y ^= (y >> 18);

    return y;
}

double genrand_real2(void) {
    return genrand_int32()*(1.0/4294967296.0);
}

```