

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

1. Intro. I'm trying to calculate a few million Ulam numbers. This sequence

$$(U_1, U_2, \dots) = (1, 2, 3, 4, 6, 8, 11, 13, 16, 18, 26, \dots)$$

is defined by setting $U_1 = 1$, $U_2 = 2$, and thereafter letting U_{n+1} be the smallest number greater than U_n that can be written $U_j + U_k$ for exactly one pair (j, k) with $1 \leq j < k \leq n$. (Such a number must exist; otherwise the pair $(j, k) = (n-1, n)$ would qualify and lead to a contradiction.)

This program uses a sieve method inspired by M. C. Wunderlich [*BIT* **11** (1971), 217–224]. The basic idea is to form infinite binary sequences $u = u_0 u_1 u_2 \dots$ and $v = v_0 v_1 v_2 \dots$ where $u_k = [k \text{ is an Ulam number}]$ and $v_k = [k \text{ has more than one representation as a sum of distinct Ulam numbers}]$. To build this sequence we start with $u = 0110\dots$ and $v = 000\dots$; then we do the bitwise calculation $w_k \dots w_{2k-1} \leftarrow w_k \dots w_{2k-1} \circ u_0 \dots u_{k-1}$ for $k = U_2, U_3, \dots$, where $w_k = (u_k, v_k)$ and

$$(u, v) \circ u' = ((u \oplus u') \wedge \bar{v}, (u \wedge u') \vee v).$$

The method works because, when $k = U_n$, the current settings of u and v satisfy the following invariant relations for $2 < j < 2k$:

$$\begin{aligned} u_j &= [j \text{ is a sum of distinct Ulam numbers } < k \text{ in exactly one way}]; \\ v_j &= [j \text{ is a sum of distinct Ulam numbers } < k \text{ in more than one way}]. \end{aligned}$$

In other words this program is basically an exercise in doing the requisite shifting and masking when the bits of u and v are packed as unsigned integers.

Besides computing U_n , I also report the value of U_n/n whenever n is a multiple of m . This ratio is reported to be about 13.5 when $n \leq 10^6$ [see Wolfram's *NKS*, page 908].

And I keep some rudimentary statistics about gaps, based on ideas of Jud McCranie.

```
#define gsize 1000
#define m 10000
#define nsize (1 << 14)
#define nmax (32 * nsize) /* we will find all Ulam numbers less than nmax */
#include <stdio.h>
unsigned int ubit[nsize + 1], vbit[nsize + 1];
char decode[64]; /* table for computing the ruler function */
int count[gsize], example[gsize];
main()
{
    register unsigned int j, jj, k, kk, kq, kr, del, c, n, u, prevu, gap;
    <Set up the decode table 5>;
    gap = 1;
    ubit[0] = #6, kr = n = prevu = 2, kq = 0, kk = 4; /* U1 = 1, U2 = 2 */
    while (1) {
        <Update wk ... w2k-1 from u0 ... uk-1 2>;
        <Advance k to Un+1 and advance n 4>;
        k = kr + (kq << 5);
        del = k - prevu;
        count[del]++, example[del] = k;
        if (del > gap) {
            if (del >= gsize) {
                fprintf(stderr, "Unexpectedly_large_gap_(%d)!_Recompile_me...\n", del);
                return -666;
            }
        }
    }
}
```

```

    }
    gap = del;
    printf("New gap %d: U %d = %d, U %d = %d\n", gap, n - 1, prevu, n, k);
    fflush(stdout);
}
prevu = k;
if ((n % m) == 0) {
    printf("U %d = %d is about %.5g * %d\n", n, k, ((double) k)/n, n);
    fflush(stdout);
}
}
done: ⟨Print gap stats 6⟩;
printf("There are %d Ulam numbers less than %d.\n", n, nmax);
}

```

2. As we compute, we'll implicitly have $k = 32kq + kr$, where $0 \leq kr < 32$; also $kk = 1 \ll kr$. Bit k of u is ($ubit[kq] \gg kr$) & 1, etc.

```

⟨Update  $w_k \dots w_{2k-1}$  from  $u_0 \dots u_{k-1}$  2⟩ ≡
for (j = c = 0, jj = j + kq; j < kq; j++, jj++) {
    if (jj ≥ nsize) goto update_done;
    del = (ubit[j] << kr) + c; /* c is a "carry" */
    c = (ubit[j] >> (31 - kr)) >> 1;
    ⟨Set (ubit[jj], vbit[jj]) to (ubit[jj], vbit[jj]) o del 3⟩;
}
if (jj ≥ nsize) goto update_done;
u = ubit[kq] & (kk - 1);
del = (u << kr) + c, c = (u >> (31 - kr)) >> 1;
⟨Set (ubit[jj], vbit[jj]) to (ubit[jj], vbit[jj]) o del 3⟩;
if (c ≠ 0) {
    jj++, del = c;
    ⟨Set (ubit[jj], vbit[jj]) to (ubit[jj], vbit[jj]) o del 3⟩;
}
update_done:

```

This code is used in section 1.

```

3. ⟨Set (ubit[jj], vbit[jj]) to (ubit[jj], vbit[jj]) o del 3⟩ ≡
u = (ubit[jj] ⊕ del) & ~vbit[jj];
vbit[jj] |= ubit[jj] & del;
ubit[jj] = u;

```

This code is used in section 2.

```

4. ⟨Advance k to  $U_{n+1}$  and advance n 4⟩ ≡
u = ubit[kq] & -(kk + kk); /* erase bits ≤ k */
while (¬u) {
    if (++kq ≥ nsize) goto done;
    u = ubit[kq];
}
kk = u & -u; /* now we must calculate kr = lg kk */
kr = decode[(mhmartin * kk) >> 27];
n++;

```

This code is used in section 1.

5. `#define mhmartin #07dcd629`

⟨ Set up the *decode* table 5 ⟩ \equiv

```
for (k = 0, j = 1; j; k++, j <= 1) decode[(mhmartin * j) >> 27] = k;
```

This code is used in section 1.

6. ⟨ Print gap stats 6 ⟩ \equiv

```
for (j = 1; j <= gap; j++)
```

```
if (count[j]) printf("gap_%d_occurred_%d_time%s, last_was_%d\n", j, count[j],
    count[j] == 1 ? "" : "s", example[j]);
```

This code is used in section 1.

7. Index.

c: [1](#).
count: [1](#), [6](#).
decode: [1](#), [4](#), [5](#).
del: [1](#), [2](#), [3](#).
done: [1](#), [4](#).
example: [1](#), [6](#).
fflush: [1](#).
fprintf: [1](#).
gap: [1](#), [6](#).
gsize: [1](#).
j: [1](#).
jj: [1](#), [2](#), [3](#).
k: [1](#).
kk: [1](#), [2](#), [4](#).
kq: [1](#), [2](#), [4](#).
kr: [1](#), [2](#), [4](#).
m: [1](#).
main: [1](#).
mhmartin: [4](#), [5](#).
n: [1](#).
nmax: [1](#).
nsize: [1](#), [2](#), [4](#).
prevu: [1](#).
printf: [1](#), [6](#).
stderr: [1](#).
stdout: [1](#).
u: [1](#).
ubit: [1](#), [2](#), [3](#), [4](#).
update_done: [2](#).
vbit: [1](#), [3](#).

- \langle Advance k to U_{n+1} and advance n 4 \rangle Used in section 1.
- \langle Print gap stats 6 \rangle Used in section 1.
- \langle Set up the *decode* table 5 \rangle Used in section 1.
- \langle Set $(ubit[jj], vbit[jj])$ to $(ubit[jj], vbit[jj]) \circ del$ 3 \rangle Used in section 2.
- \langle Update $w_k \dots w_{2k-1}$ from $u_0 \dots u_{k-1}$ 2 \rangle Used in section 1.

ULAM

	Section	Page
Intro	1	1
Index	7	4