October 19, 2014 at 05:34

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

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

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 writtin $U_j + U_k$ for exactly one pair (j, k) with $1 \le j < k \le n$. (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 $u = u_0u_1u_2\ldots$ and $v = v_0v_1v_2\ldots$ where $u_k = [k$ is an Ulam number] and $v_k = [k$ has more than one representation as a sum of distinct Ulam numbers]. To build this sequence we start with $u = 0110\ldots$ and $v = 000\ldots$; then we do the bitwise calculation $w_k \ldots w_{2k-1} \leftarrow w_k \ldots w_{2k-1} \circ u_0 \ldots u_{k-1}$ for $k = U_2, U_3, \ldots,$ 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 $2 < j < 2k$:

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

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 $n \le 10^6$ [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 10000000
#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 table[256];
  int count[gsize], example[gsize];
  main()
  {
    register unsigned int j, jj, k, kk, kq, kr, del, c, n, u, prevu, gap;
    ⟨Set up the table 5⟩;
    gap = 1, count[1] = 1, example[1] = 2;
    ubit[0] = #6, kr = n = prevu = 2, kq = 0, kk = 4;     /* U_1 = 1, U_2 = 2 */
    while (1) {
      ⟨Update w_k … w_{2k−1} from u_0 … u_{k−1} 2⟩;
      ⟨Advance k to U_{n+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;
```

```
          }
          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 $0 \le kr < 32$; also $kk = 1 \ll kr$. Bit k of u is $(ubit[kq] \gg kr)\ \&\ 1$, etc.

```
⟨Update w_k … w_{2k−1} from u_0 … 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]) ∘ 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]) ∘ del 3⟩;
  if (c ≠ 0) {
    jj++, del = c;
    ⟨Set (ubit[jj], vbit[jj]) to (ubit[jj], vbit[jj]) ∘ del 3⟩;
  }
update_done:
```

This code is used in section 1.

3. ⟨Set $(ubit[jj], vbit[jj])$ to $(ubit[jj], vbit[jj]) \circ del$ 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 */
if (kk & #ffff0000) kr = 16, u = kk ≫ 16; else kr = 0, u = kk;
if (u & #ff00) kr += 8, u ≫= 8;
if (u & #f0) kr += 4, u ≫= 4;
kr += table[u];
n++;
```

This code is used in section 1.

5. ⟨ Set up the *table* 5 ⟩ ≡

```
for (j = 2; j < 256; j ≪= 1) table[j] = 1 + table[j ≫ 1];
```

This code is used in section 1.

6. ⟨ Print gap stats 6 ⟩ ≡

```
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.

$\langle$ 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 *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 $w_k \ldots w_{2k-1}$ from $u_0 \ldots u_{k-1}$ 2 $\rangle$ Used in section 1.

KNULAM