Covering allbases

Edwin Fuquen

edwin@somespider.com

@efuquen

UUIDs in URLs

www.themid.com/culture/what-you-need-to-know-this-week?u=de305d54-75b4-431b-adb2-eb6b9e546013

Deuglify

Remove dashes and
a more efficient encoding.

de305d54-75b4-431b-adb2-eb6b9e546013

$$\frac{128 \text{ bits}}{4 \text{ bits/char}} + 4 \text{ hyphens} = 36 \text{ chars}$$

to

3jBdVHW0QxutsutrnlRgEw==

$$ceil(\frac{128 \text{ bits}}{6 \text{bits/char}}) = 22 \text{ chars}$$

Can we do better?

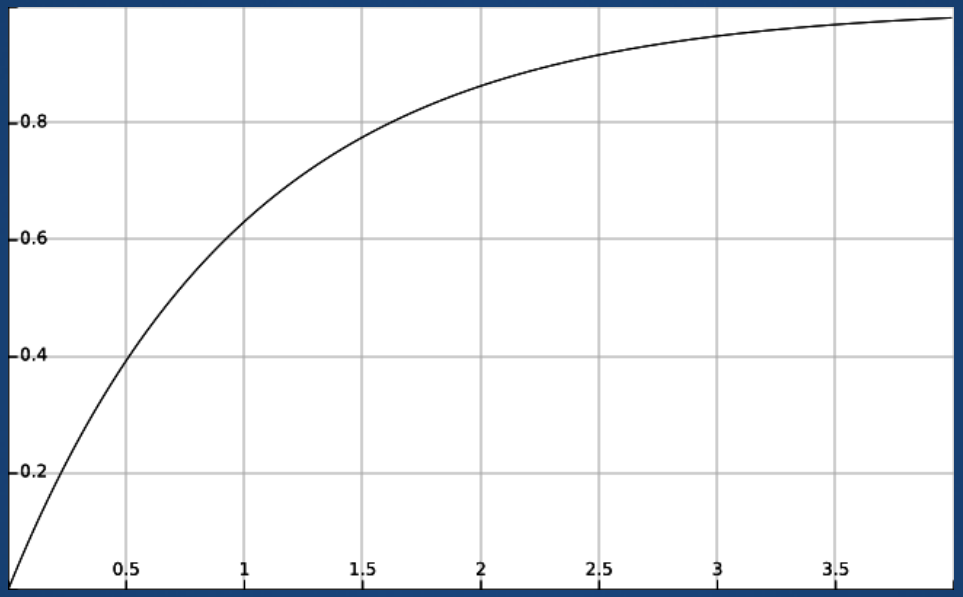
Do we really need 128 bits?

$$p(n) \approx 1 - e^{-\frac{n^2}{2m}}$$

- p(n) - probability of collision after *n* UUIDs generated
- m - permutations of the ID (i.e. for UUIDs 2^{122})
- notice the denominator will be fixed for an ID
- for a UUID it will equal 2 * 2^{122} which equals 2^{123}

n	**probability**
68,719,476,736 = 2^{36}	0.0000000000000004 (4×10^{-16})
2,199,023,255,552 = 2^{41}	0.000000000004 (4×10^{-13})
70,368,744,177,664 = 2^{46}	0.0000000004 (4×10^{-10})

$$y = 1 - e^{-x} \quad \text{where} \quad x = \frac{n^2}{2m}$$



The smaller the x, the lower the probability of collision.

$$y = 1 - e^{-x}$$ where $$x = \frac{n^2}{2m} = \frac{n^2}{2^{123}}$$

n	x
2^{36}	$\frac{(2^{36})^2}{2^{123}} = \frac{1}{2^{51}}$
2^{42}	$\frac{(2^{42})^2}{2^{123}} = \frac{1}{2^{41}}$
2^{62}	$\frac{(2^{62})^2}{2^{123}} = 2^1$

x	collision
$1 / 2^3$	11.75%
$1 / 2^5$	1.55%
$1 / 2^7$	0.79%
$1 / 2^{11}$	0.049%

$$1 - e^{-2} = 0.8647$$

$$\boxed{m = \frac{n^2}{x}}$$

- m - size of id needed
- n - number of ids generated
- x - variable dependent on probability of collision, as defined previously.

Assume we're fine with probability of collision of 0.79% after an **n** of 2^28 (over 268 million) ids.

$$m = \frac{(2^{28})^2}{\frac{1}{2^7}} = 2^{63} = 63\,bits$$

$$ceil(\frac{63\text{ bits}}{6\text{bits/char}}) = 11\text{ chars}$$

What do we want?

- Correctly randomly generated ids
- Encodable and decodable to many formats
- To an **arbitrarily** large size

Most libraries that might have worked
failed the last criteria, because ...

Number is 64 bits

Any transformation that goes above 64 bits will would give incorrect results.

```
num:     7.8938753285094 83e+37
bignum: 7893875328509483095840376098504975048 5
base62 num:     1030rJJojGqkMKCw4wGMyE
base62 bignum: 1030rJJojG2kAk201mn0ND
Num diff:     -9.44473296573929e+21
Bignum diff: 0
```

https://github.com/efuquen/base62-precision-test

Character Classes

Indices reperesent the decimal, or base 10, values of each character in the class.

```
//                 0              15
var HexChars    = "0123456789abcdef";

//                 0         10        20        30        40        50        61
var Base62Chars = "0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ";
```

All the bases can be represented this way and then the same base algorithms can be used to encode and decode.

Encode

```
//num is a bignum to decode
//chars is a string representing the character class
function encode(num, chars) {
        var len   = chars.length;
        var indices = [];
        do {
                indices.push(num.mod(len));
                num = num.div(len);
        } while (num.gt(0));

        var str = [];
        while (indices.length > 0) {
                str.push(chars[indices.pop()]);
        }
        return str.join("");
}
```

Let's turn 10 into binary

$10 \; mod \; 2 = 0$

```
indicies = [0];
num = 10 / 2 = 5;
```

$5 \; mod \; 2 \; = 1$

```
indices = [0, 1];
num = Math.floor(5/2) = 2
```

$2 \; mod \; 2 \; = 0$

```
indices = [0, 1, 0];
num = 2/2 = 1;
```

$1 \; mod \; 2 \; = 1$

```
indices = [0, 1, 0, 1]
```

```
str = [1, 0, 1, 0];
str = str.join("") = "1010";
```

Decode

```
//str:   string to decode
//chars: string representing character class
function decode(str, chars) {
  str = esrever.reverse(str);
  var num = bignum("0");
  var len = chars.length;
  for (var i = 0; i < str.length; i++) {
    var ch = str[i];
    var chIndex = chars.indexOf(ch);
    var placeNum = bignum.pow(len, i).mul(chIndex);
    num = num.add(placeNum);
  }
  return num;
}
```

Let's decode base62 D4x7

```
str = str.reverse() = "7x4D"
```

```
 0         10        20        30        40        50        61
"0123456789abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ"
```

i	calc	sum
0	$7 * 62^0$	7
1	$33 * 62^1$	2046
2	$4 * 62^2$	15376
3	$39 * 62^3$	9294792

num = 7 + 2046 + 15376 + 929472 = 946901

Random

```
function genRandom(numberOfBytes, chars, rand) {
  var buf = rand(numberOfBytes);
  return encode(bignum.fromBuffer(buf), chars);
}

function random(size, opts) {
  var len   = opts.chars.length;
  var combos = Math.pow(len, size);
  if (combos === Infinity) {
    throw "Too large of a size, can't estimate needed bytes"
  }
  var bits = Math.ceil(log(2, combos));
  var numberOfBytes = Math.ceil(bits / 8);
  return genRandom(numberOfBytes, opts.chars, opts.rand);
}
```

But, isn't a problem here? I'm using Number ...

Number.MAX_VALUE

- 1.7976931348623157e+308
- base62 can't generate more then 171 characters.
- Big Decimal? big.js library
- But, it doesn't have a log implementation
- Two possible solutions:
  - Find a good log estimate for large decimals
  - Use big.js to detect large combos and break it down to smaller numbers < Number.MAX_VALUE

Conclusions

- You can't shove everything into Number
- Be careful, encoding/decoding bases one of many issues.
- A little rigor can go a long way.
- **Warn** your users, they don't need any extra rope from you. Plenty of other dragons to deal with already.

Thanks

https://github.com/somespider/allbases