UFABC

Bitwise tricks

Operações com bits

Nós estamos acostumados com operações aritméticas: soma, subtração, multiplicação e divisão.

Mas os computadores entendem melhor operações booleanas, que computam diretamente os bits.

Veremos que são operadores úteis para os programadores.

UFABC

Operações em bits

Um truque é uma ideia que pode ser usada uma vez, uma técnica é um truque amadurecido que pode ser utilizado pelo menos duas vezes! (Donald Knuth)

Operadores de Bit

$x \,\&\, y \Leftrightarrow x_k \wedge y_k$ para todo bit k

$x \mid y \Leftrightarrow x_k \vee y_k$ para todo bit k

$x \oplus y \Leftrightarrow x_k \oplus y_k$ para todo bit k

$x << k = \lfloor 2^k x \rfloor$

$x >> k = \lfloor 2^{-k} x \rfloor$

UFABC

Empacotando

Considere uma data composta por:

ano

mes

dia

Quantos bits precisamos para representar cada um deles?

UFABC

Empacotando

Considere uma data composta por:

ano - quantos puder!
mes - 4 bits
dia   - 5 bits

Quantos bits precisamos para representar cada um deles?

Empacotando

Um char tem 8 bits, um int tem 32 ou 64 bits...e se compactassemos a data inteira em um int?

```
int data = (((ano << 4) + mes) << 5) + dia;
```

Empacotando

Podemos perceber alguns fatos interessantes:

Se data1 < data2, então a data1 precede a data2!

dia = data % 32
mes = (data >> 5) % 16
ano = data >> 9

Empacotando

E os operadores % podem se tornar:

$x \% 2^n = x \& (2^n - 1)$

dia = data & 31
mes = (data >> 5) & 15
ano = data >> 9

É par?

Uma operação comum feita com:

if (x % 2) faça algo...

Números binários

$1\ 0\ 0\ 1\ 0\ 1 = 2^0 + 2^2 + 2^5 = 1 + 4 + 32 = 37$

Soma de dois pares é par, soma de dois ímpares é ímpar.

Somente ímpar se somar par com ímpar!

2^n é sempre par, exceto quando n = 0...

UFABC

Números binários

Um número é ímpar apenas se o último bit é 1!

x & 1 == 0, se for par, e 1 se for ímpar:

if (!(x & 1)) faça algo

Exercício

Faça N grande operações de comparação utilizando % e &, verifique qual é mais rápido.

UFABC

Multiplicação e Divisão por 2

$1\ 0\ 0\ 1\ 0\ 0 = 2^2 + 2^5 = 4 + 32 = 36$

Se eu divido por 2, tenho 18, ou

$(2^2 + 2^5)/2 = 2^1 + 2^4 = 0\ 1\ 0\ 0\ 1\ 0$

Se multiplico por 2, tenho 72, ou

$(2^2 + 2^5)*2 = 2^3 + 2^6 = 1\ 0\ 0\ 1\ 0\ 0\ 0$

UFABC

Multiplicação e Divisão por 2

x << 1 /* multiplica por 2 */

x >> 1 /* divide por 2 */

x << n, x >> n /* multiplica/divide por 2^n */

Exercício

Faça N grande operações de multiplicação por 2 utilizando * e <<.

Atenção!!

Esse tipo de operação pode inverter o sinal de números negativos! Utilize apenas quando o tipo é **unsigned**.

Empacotando números primos

Imagine que queremos manter em memória uma tabela indicando se os números ímpares de 1 a 1024 são primos.

Uma array com tamanho 1024, utilizando o tipo de menor tamanho, custa 1024 bytes, ou 1 megabyte.

Por outro lado, podemos compactar esses números em 8 variáveis de 64 bits (long long int).

Criando os bits

```
long long int primos[8] = {0};
for (i=1; i<512;i++) {
  if (primo(2*i + 1)) seta(primos, i);
}
```

Criando os bits

```
void seta(long long int x[], int bit)
{
  int n = bit >> 6;
  x[n] |= (1LL << (bit & 63));
}
```

Criando os bits

```
unsigned int pega(unsigned long int x[], unsigned int bit)
{
  unsigned int n = bit >> 6;
  return (x[n] >> (bit & 63)) & 1;
}
```

log2()

$1\ 0\ 0\ 1\ 0\ 0 = 2^2 + 2^5 = 4 + 32 = 36$

log2(36) ~ 5

log2(72) ~ 6

log2(x) ~ posição do maior bit setado

log2()

```
log = 0;
while(x >>= 1)
  log++;
```

Trocando dois valores com XOR

x = 13, y = 11

x = 1101, y = 1011

x ^ y = 0110

(x ^ y) ^ y = 1101 == x

(x ^ y) ^ x = 1011 == y

```
x ^= y
y ^= x
x ^= y
```

UFABC

Trocando dois valores com XOR

Qual é mais rápido? Swap com XOR ou com uma variável intermediária?

Conjunto de partes

Dado S = {1,2,3,..n} encontrar:

{ {}, {1}, {2}, {3}, …, {n}, {1,2}, {1,3} … }

Conjunto de partes

Vamos pensar em um conjunto pequeno {1, 2, 3}:

{ { }, {1}, {2,} ,{3}, {1,2}, {1,3}, {2,3}, {1,2,3} }

Uso: dentre todas as combinações de n itens, qual é a melhor (dado algum critério)?

Conjunto de partes

Como gerar?

Para i de 1 até n,
  print i // {1}, {2}, …

Para i de 1 até n,
  Para j de i+1 até n,
    print i,j // {1,2}, {1,3} …

Como generalizar?

UFABC

Conjunto de partes

Vamos pensar em um conjunto pequeno {1, 2, 3}:

{ { }, {1}, {2} ,{3}, {1,2}, {1,3}, {2,3}, {1,2,3} }

{} = 0b
{1} = 1b
{2} = 10b
{3} = 100b
{1,2} = 11b

UFABC

Conjunto de partes

Cada combinação é um número binário, os itens do subconjunto são os bits setados:

{} = 0b = 0

{1} = 1b = 1

{2} = 10b = 2

{3} = 100b = 4

{1,2} = 11b = 3

...

Conjunto de partes

Cada combinação é um número binário, os itens do subconjunto são os bits setados:

{} = 0b = 0

{1} = 1b = 1

{2} = 10b = 2

{3} = 100b = 4

{1,2} = 11b = 3

...

Conjunto de partes

Quantos subconjuntos existem? 2^n ou (1 << n):

```
void conjunto_partes(unsigned int n)
{
  for (int i=0; i<(1<<n); i++) {
    printf("{");
    print_bits(i);
    printf("}, ");
  }
  printf("\n");
}
```

UFABC

Conjunto de partes

```
void print_bits(unsigned int x)
{
  int j = 1;
  while (x) {
    if (x & 1) printf("%d ", j);
    x >>= 1;
    ++j;
  }
}
```

Gosper's hack

Vamos ver o que o seguinte código faz:

```
x = 3;
do {
  print_bits(x);
  x = gosper(x);
} while (x < (1<<5));
```

Gosper's hack

```
unsigned int gosper(unsigned int x)
{
  unsigned int u, v, y;

  u = x & -x;
  v = x + u;
  y = v + (((v^x)/u) >> 2);

  return y;
}
```

UFABC

Gosper's hack

Resultado:

{1, 2}, {1, 3}, {2, 3}, {1, 4}, {2, 4}, {3, 4}, {1, 5}, {2, 5}, {3, 5}, {4, 5}

Combinações 2 a 2 de n. Mas como???

Gosper's hack

x & -x

y = -x, tal que x + y = 0.

em números binários temos que $y = 2^n - x$, para n bits.

Da mesma forma -x = !x + 1, então x & (!x + 1)

Gosper's hack

```
   6 == 0 0 0 0 0 1 1 0
 & -6 == 1 1 1 1 1 0 1 0
----------------------
   2 == 0 0 0 0 0 0 1 0


  16 == 0 0 0 1 0 0 0 0
& -16 == 1 1 1 1 0 0 0 0
----------------------
  16 == 0 0 0 1 0 0 0 0
```

UFABC

Gosper's hack

u = x & -x = o bit menos significativo de x!

Fazendo $x = \{10\}^a 01^b 0^c$.

sendo a + b + c + 1 = 64 bits

temos que:

$u = 0^{(a+b)} 10^c$

UFABC

Gosper's hack

E v = x + u?

Bom, somar o bit menos significativo, vai provocar um "vai um" na soma, transformando esse bit em 0 e os seguintes até encontrar um bit 0 que se torna 1!

Gosper's hack

E v = x + u?

Ele deixa os k primeiros bits intactos e inverte todos os restantes:

$x = \{10\}^a 01^b 0^c$

$u = 0^{(a+b)} 10^c$

$v = \{10\}^a 10^{b+c}$

Gosper's hack

$y = v + (v \hat{} x)/u >> 2$

$(v \hat{} x)$ = zera os bits iguais, seta os bits diferentes

$x = \{10\}^a 01^b 0^c$

$v = \{10\}^a 10^{b+c}$

$v \hat{} x = 0^a 1^{b+1} 0^c$

Gosper's hack

y = v + (v^x)/u >> 2

(v^x)/u = u é potência de 2, então é equivalente ao shift para direita em c bits

$v^x = 0^a 1^{b+1} 0^c$

$u = 0^{(a+b)} 10^c$

$/ = 0^{a+c} 1^{b+1}$

Gosper's hack

y = v + (v^x)/u >> 2

(v^x)/u >> 2 = desloca mais duas posições

/ = $0^{a+c}1^{b+1}$

>>2 = $0^{a+c+2}1^{b-1}$

Gosper's hack

$y = v + (v \hat{} x)/u >> 2$

$v + (v \hat{} x)/u >> 2 =$ somando v

$>>2 = 0^{a+c+2}1^{b-1}$

$v = \{10\}^a 10^{b+c}$

$y = \{10\}^a 10^{c+1} 1^{b-1}$

UFABC

Gosper's hack

Comparando com x:

$y = \{10\}^a 10^{c+1} 1^{b-1}$

$x = \{10\}^a 01^b 0^c$

$y > x$

y e x tem a mesma quantidade de 1s

Dado um número x com **b** bits setados, a sequência gera os subconjuntos de b elementos contendo os números de 1 a 64 (ao usar long int).

UFABC