

Boolean Compression

Stephanus – SIWare, Inc

Many times we need to declare array of boolean to indicate true/false. One concrete example is sieve of Eratosthenes. We use array of boolean to indicate whether a number is prime or not.

Example – Sieve of Eratosthenes – using array of boolean

```
bool prime[4];

prime[0] = false;
prime[1] = false;
prime[2] = true;
prime[3] = true;
```

However, bool requires 1 byte (= 8 bits) and actually 1 byte can represent 8 true/false value (0 = false, 1 = true). So, it's a waste of memory. Many people don't care about this since memory is cheap, but in competitive programming sometimes we need to declare a very big, let's say 100000000, array of boolean. Array of boolean won't hold in 32 MB memory limit. However, if you use Boolean Compression technique, we can get about 1/8 of the array of boolean size.

Memory analysis – 100 M boolean values

```
array of boolean    : 100 M x sizeof(bool) = 100 M x 1 bytes = 100 MB
boolean compression : 100 M / 8 x 1 bytes = 12.5 M x 1 bytes = 12.5 MB
```

Example – Sieve of Eratosthenes – using boolean compression

```
// Representation
// prime    : 0 1 2 3
// boolean  : 0 0 1 1 --convert to int value--> 3

char prime;
prime = 3;
```

Implementation

We use integer and utilize all 32 bits to represent 32 boolean values. The implementation is based on Programming Pearls.

C Source Code

```
#define BITSPERWORD 32
#define SHIFT      5
#define MASK       0x1F
#define N          100000000

int a[1 + N/BITSPERWORD];

void set(int i) { a[i>>SHIFT] |= (1<<(i & MASK)); }
```

```
void clr(int i) { a[i>>SHIFT] &= ~(1<<(i & MASK)); }
int test(int i) { return a[i>>SHIFT] & (1<<(i & MASK)); }
```

To use above code is very easy, to set true on i-th element write "set(i)" and to set false on i-th element write "clr(i)". To check whether the i-th element is true or false use test(i). Test(i) will return 0 if false otherwise it will return values other than 0.

Further Implementation

Another useful application for boolean compression is to represent static sets. For example, you want to have static set that can only contain values A,B,C,D. We can represent the set by 4 boolean values. 1 for existence 0 for non-existence.

Example – Static Set

S = { A, C }

S contain 1 A, 0 B, 1 C, 0 D, so the bits representation is 1010 which is equals to 10.

The advantage of this method is to unite and intersect we only need 1/32 n operation.

C++ Source Code

```
#define BITSPERWORD 32
#define SHIFT      5
#define MASK       0x1F

#include <cstring>

class Bits{
    int *bits;
    int size;

public:
    // constructor
    Bits(int n);
    ~Bits();

    // member function
    void set(int i);
    void clr(int i);
    int  on(int i);
    void unite(Bits &b);
    void isect(Bits &b);
    int getBit(int idx);

};

Bits::Bits(int n)
{
    size = 1 + n/BITSPERWORD;
    bits = new int[size];
    memset(bits,0,sizeof(bits)*size);
```

```

}

Bits::~Bits()
{
    delete bits;
}

int Bits::getBit(int idx)
{
    return bits[idx];
}

void Bits::set(int idx)
{
    bits[idx>>SHIFT] |= (1<<(idx & MASK));
}

void Bits::clr(int idx)
{
    bits[idx>>SHIFT] &= ~(1<<(idx & MASK));
}

int Bits::on(int idx)
{
    return bits[idx>>SHIFT] & (1<<(idx & MASK));
}

void Bits::unite(Bits &b)
{
    for (int i=0; i<size; i++)
        bits[i] |= b.getBit(i);
}

void Bits::isect(Bits &b)
{
    for (int i=0; i<size; i++)
        bits[i] &= b.getBit(i);
}

```