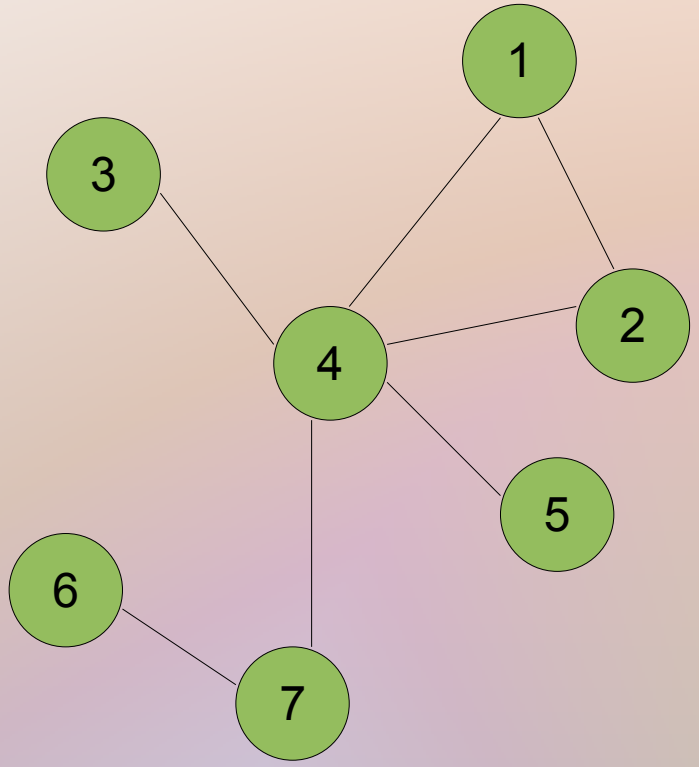
# C/C++ Bitsets

Γιάννης Χατζημίχος
( feedward@gmail.com )

Δεκέμβριος 2010

# Αναπαράσταση συνόλων




Θεωρούμε ως σύμπαν το σύνολο των κορυφών του διπλανού γραφήματος.

Εφόσον το σύμπαν αποτελείται από 7 στοιχεία, χρειαζόμαστε 7 bits για να αναπαραστήσουμε κάθε σύνολο.

Το σύνολο των κορυφών με degree 2 είναι το {1, 2, 7}. Αυτό μπορούμε να το αναπαραστήσουμε με ένα bitset που έχει αναμμένα το 1ο, 2ο και 7ο bit, δηλαδή το: **1000011**.

Αντίστοιχα το σύνολο των κορυφών με degree μεγαλύτερο του 2 είναι το {4}, το οποίο αναπαρίσταται με το bitset **0001000**.

# Μέγεθος τύπων μεταβλητών

| | bytes | bits |
|---|---|---|
| **char** | 1 | 8 |
| **short int** | 2 | 16 |
| **(long) int** | 4 | 32 |
| **long long int** | 8 | 64 |

# Bitwise τελεστές

- **ΚΑΙ / AND / τελεστης &**

  στο αποτέλεσμα θα είναι αναμμένα μόνο τα bits που ήταν αναμμένα και στους δύο τελεστέους.

  παράδειγμα:

  x = 101100

  y = 101011

  x&y = 101000

  *Χρησιμοποιείται για την εύρεση της τομής δύο συνόλων.*

# Bitwise τελεστές

- **Ή / OR / τελεστής |**

  στο αποτέλεσμα θα είναι αναμμένα μόνο τα bits που ήταν αναμμένα τουλάχιστον σε έναν από τους δύο τελεστέους.

  παράδειγμα:

  x = 101100

  y = 101011

  x|y = 101111

  *Χρησιμοποιείται για την εύρεση της ένωσης δύο συνόλων.*

# Bitwise τελεστές

- **Αποκλειστικό Ή / XOR / τελεστής ^**

  στο αποτέλεσμα θα είναι αναμμένα μόνο τα bits που ήταν αναμμένα μόνο στον έναν από τους δύο τελεστέους.

Παράδειγμα 1:

x = 101100

y = 101011

x^y = 000111

Παράδειγμα 2:

**0^0 = 0** **1^0 = 1**

**0^1 = 1** **1^1 = 0**

*Χρησιμοποιείται σε διαδικασίες αντιστροφής.*

# Bitwise τελεστές

- **ΟΧΙ / NOT / τελεστής ~**

  αντιστρέφει τα bits του αριθμού (τα μηδενικά γίνονται άσσοι και οι άσσοι γίνονται μηδενικά).

  παράδειγμα:

  x = 101100

  ~x = 010011

*Χρησιμοποιείται για την εύρεση του συμπληρωματικού συνόλου.*

# Bitwise τελεστές

- **Left shift / αριστερή ολίσθηση / τελεστής <<**

  μετακινεί όλα τα ψηφία κάποιες θέση αριστερά.

  παράδειγμα:

  x = 0011101

  x<<1 = 0111010

  x<<2 = 1110100

# Bitwise τελεστές

- **Right shift / δεξιά ολίσθηση / τελεστής >>**

  μετακινεί όλα τα ψηφία κάποιες θέση δεξιά.

  παράδειγμα:

  x = 0011101

  x>>1 = 0001110

  x>>2 = 0000111

# Πράξεις

- Κατασκευή συνόλου που έχει μόνο το n-οστο bit αναμμένο:

```
x = (1<<n);
```

- Κατασκευή συνόλου που έχει μόνο όλα τα πρώτα n-1 bits αναμμένα:

```
x = (1<<n) - 1;
```

**Προσοχή:** τα παραπάνω ισχύουν για σύνολα έως 32 bits, καθώς η σταθερά 1 είναι τύπου int. Για μεγαλύτερα σύνολα θα χρησιμοποιούσαμε σταθερά τύπου long long, δηλαδή θα γράφαμε (1LL << n).

| Δεκαδικό Σ. | Δυαδικό Σ. |
|---|---|
| 0 | 0 |
| **1** | **1** |
| **2** | **10** |
| 3 | 11 |
| **4** | **100** |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| **8** | **1000** |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| **16** | **10000** |

# Πράξεις

- Εύρεση του πλήθους των στοιχείων ενός συνόλου (population count):

```
y = __builtin_popcount(x);
```

Παράδειγμα:

```
x = 6;
y = __builtin_popcount(x);

printf("%d\n", y);
/* θα εμφανίσει: 2 */
```

**Προσοχή:** η __builtin_popcount παίρνει σαν όρισμα unsigned int. Για να μετρήσουμε τους άσσους ενός long long χρησιμοποιούμε την __builtin_popcountll .

| Δεκαδικό Σ. | Δυαδικό Σ. |
|---|---|
| 0 | 0 |
| **1** | **1** |
| **2** | **10** |
| 3 | 11 |
| **4** | **100** |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| **8** | **1000** |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| **16** | **10000** |

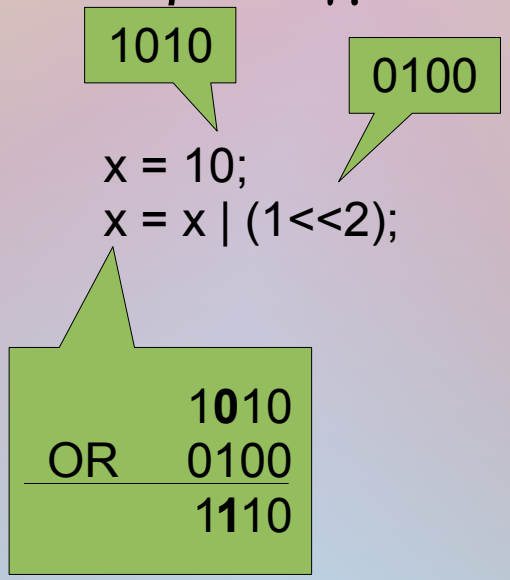
# Πράξεις

- Αλλαγή του n-οστού bit

**Σε 1:**

```
x |= (1<<n);
```

Παράδειγμα:




| Δεκαδικό Σ. | Δυαδικό Σ. |
|---|---|
| 0 | 0 |
| **1** | **1** |
| **2** | **10** |
| 3 | 11 |
| **4** | **100** |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| **8** | **1000** |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| **16** | **10000** |

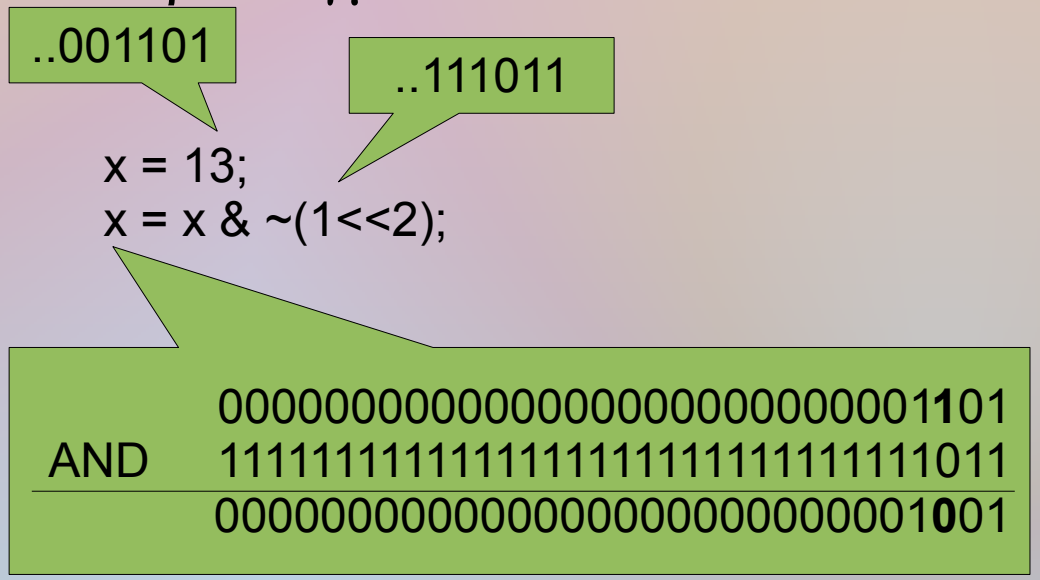
# Πράξεις

- Αλλαγή του n-οστού bit

**Σε 0:**

```
x &= ~(1<<n);
```

Παράδειγμα:




| Δεκαδικό Σ. | Δυαδικό Σ. |
|---|---|
| 0 | 0 |
| **1** | **1** |
| **2** | **10** |
| 3 | 11 |
| **4** | **100** |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| **8** | **1000** |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| **16** | **10000** |

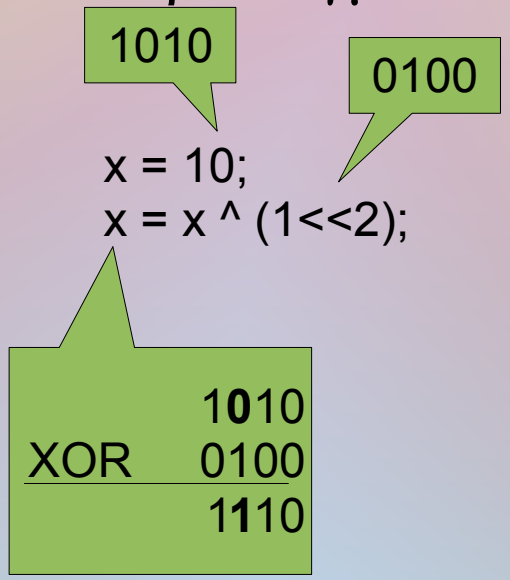
# Πράξεις

- Αλλαγή του n-οστού bit

**Αντιστροφή:**

```
x ^= (1<<n);
```

Παράδειγμα:




| Δεκαδικό Σ. | Δυαδικό Σ. |
|---|---|
| 0 | 0 |
| **1** | **1** |
| **2** | **10** |
| 3 | 11 |
| **4** | **100** |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| **8** | **1000** |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| **16** | **10000** |

# Πράξεις

- Έλεγχος (test) του n-οστού bit

```
( x & (1 << n) ) != 0
```

Παράδειγμα:

```
if ( x & (1 << n) ) {
    printf("το n-οστο bit είναι 1");
}
else {
    printf("το n-οστο bit είναι 0");
}
```

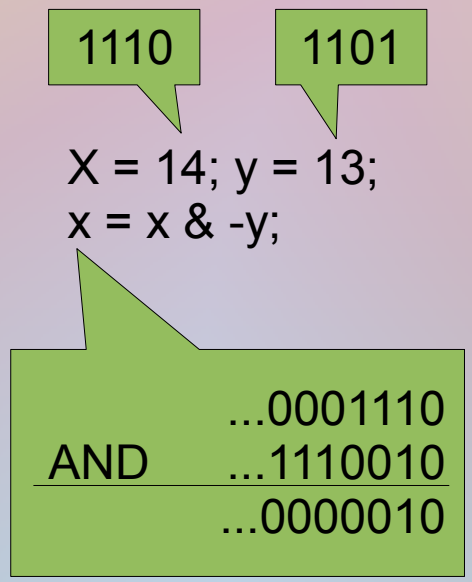
| Δεκαδικό Σ. | Δυαδικό Σ. |
|---|---|
| 0 | 0 |
| **1** | **1** |
| **2** | **10** |
| 3 | 11 |
| **4** | **100** |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| **8** | **1000** |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| **16** | **10000** |

# Πράξεις

- Αφαίρεση του σύνολου y

```
x &= ~y
```

Παράδειγμα:




| Δεκαδικό Σ. | Δυαδικό Σ. |
|---|---|
| 0 | 0 |
| **1** | **1** |
| **2** | **10** |
| 3 | 11 |
| **4** | **100** |
| 5 | 101 |
| 6 | 110 |
| 7 | 111 |
| **8** | **1000** |
| 9 | 1001 |
| 10 | 1010 |
| 11 | 1011 |
| 12 | 1100 |
| 13 | 1101 |
| 14 | 1110 |
| 15 | 1111 |
| **16** | **10000** |

# Διαφορές από άλλες υλοποιήσεις

- Άλλες υλοποιήσεις:
  - **char[]**
  - **vector<bool>**
  - **bitset**
- Πλεονεκτήματα:
  - Λιγότερη μνήμη (χρησιμοποιούν 1 byte και οχι 1 bit για κάθε ψηφίο!)
  - Πολύ πιο γρήγορες πράξεις οι οποίες μεταφράζοναι απευθείας σε hardware instructions.
  - Πιο σύντομος και ευέλικτος κώδικας

# Διαφορές από άλλες υλοποιήσεις

- Άλλες υλοποιήσεις:
  - **char[]**
  - **vector<bool>**
  - **bitset**
- Μειονεκτήματα:
  - Λιγότερο αναγνώσιμος κώδικας (ποιος νοιάζεται;)
  - Πιο πολύπλοκος κώδικας όταν θέλουμε σύνολα πολλών στοιχείων (δηλαδή περισσότερων από 64)

# Συχνά bugs

- Οι τελεστές and, xor και or έχουν χαμηλότερη προτεραιότητα από τους τελεστές σύγκρισης! Αυτό σημαίνει ότι η παράσταση:

```
x & 1 == 0
```

μεταφράζεται σε:

```
x & (1 == 0)
```

# Συχνά bugs

- Αν πρόκειται να χρησιμοποιήσεις και τα 32/64 bits της μεταβλητής, τότε να προτιμάς τους unsigned τύπους (unsigned int και unsigned long long), καθώς οι signed έχουν απρόβλεπτη συμπεριφορά όταν χρησιμοποιείται το αριστερότερο bit.

# Πρόβλημα: knapsack

Έχουμε έναν σάκο ο οποίος έχει όγκο V και $2 \leq n \leq 20$ αντικείμενα, καθένα από τα οποία έχει όγκο K_i. Θέλουμε να χρησιμοποιήσουμε κάποια από αυτά τα αντικείμενα για να γεμίσουμε τον σάκο όσο το δυνατόν μπορούμε. Υπάρχει ο επιπλέον περιορισμός ότι δεν μπορούμε να χρησιμοποιήσουμε περισσότερα από U αντικείμενα. Στην πρώτη γραμμή δίνονται οι αριθμοί n V U. Στην επόμενη δίνονται n αριθμοί που αντιστοιχούν στον όγκο των αντικειμένων με τα οποία πρέπει να γεμίσουμε τον σάκο.

| Παράδειγμα Εισόδου | Παράδειγμα Εξόδου |
|---|---|
| 5 5.000 3<br>0.500 0.500 1.200<br>1.000 3.000 | 4.700 |

# Πρόβλημα: giorti

Μια τάξη που αποτελείται απο $2 \leq n \leq 100$ παιδιά διοργανώνει μια χριστουγεννιάτικη γιορτή. Στην γιορτή θα τραγουδήσουν κάποια από τα $2 \leq m \leq 20$ υποψήφια τραγούδια. Κάθε παιδί όμως ξέρει διαφορετικά τραγούδια. Κάθε φορά που η τάξη λέει ένα τραγούδι, συμμετέχουν όσα παιδιά το ξέρουν. Βρείτε τον ελάχιστο αριθμό τραγουδιών που πρέπει να πουν τα παιδιά στη γιορτή, έτσι ώστε να συμμετέχει το κάθε παιδί σε τουλάχιστον ένα τραγούδι.

- Στην πρώτη γραμμή εμφανίζονται οι αριθμοί n και m.
- Ακολοθούν n γραμμές. Ο m-οστος χαρακτήρας της n γραμμής είναι 'N' αν το n-οστο παιδί ξέρει το m-οστο τραγούδι και 'O' αν δεν το ξέρει.

| Παράδειγμα Εισόδου | Παράδειγμα Εξόδου |
|---|---|
| 5 5<br>NONNN<br>OOOON<br>NNNNN<br>NONOO<br>ONOOO | 3 |

Εξήγηση παραδείγματος: Αρκεί να πουν τα τραγούδια 1, 2 και 5.

# Πρόβλημα: corners

# Πρόβλημα: corners

## 1ος τρόπος


Κάθε bit είναι αναμένο αν υπάρχει πούλι στην αντίστοιχη θέση. Η παραπάνω κατάσταση κωδικοποιείται στον αριθμό:
11000011000000000000000000000000000

# Πρόβλημα: corners

## 2ος τρόπος


Κάθε πούλι παίρνει έναν αριθμό από το 0 έως το 35. Επομένως χρειαζόμαστε 6 bits για να ορίσουμε πλήρως την θέση του. Μιας και έχουμε 4 πούλια συνολικά, χρειαζόμαστε 4 * 6 = 24 bits.

011100 011101 100010 100011

1ο πούλι 2ο πούλι 3ο πούλι 4ο πούλι