Camp Προετοιμασίας
Πανελλήνιος Διαγωνισμός Πληροφορικής
Αθήνα, 11-16 Απριλίου 2011

Αναδρομή

Δημήτρης Τερζόπουλος

Τι είναι η αναδρομή

Ένας πρόχειρος ορισμός για την αναδρομή είναι ο εξής:

Αναδρομή έχουμε όταν μία συνάρτηση καλεί τον εαυτό της.

Τι είναι η αναδρομή

Συνήθως ένας αναδρομικός αλγόριθμος αποτελεί μια εύκολη λύση σε πολλά προβλήματα καθώς είναι απλός στην σκέψη και εύκολος στην υλοποίηση του.

Όμως, τις περισσότερες φορές ένας αναδρομικός αλγόριθμος δεν αποτελεί την πιο αποδοτική λύση και είναι ανάγκη να συνδυαστεί με άλλες προγραμματιστικές τεχνικές που θα μάθουμε αργότερα.

Παραδείγματα

Μερικά παραδείγματα προβλημάτων που λύνονται με αναδρομή είναι:

- Εύρεση ν-οστού όρου ακολουθίας fibonacci
- Εύρεση παραγοντικού
- Εύρεση μέγιστου κοινού διαιρέτη
- Οι Πύργοι του Ανόι
- Το πρόβλημα των Ν-Βασιλισσών

Ακολουθία Fibonacci

Η ακολουθία fibonacci είναι μια «σειρά» αριθμών με την εξής ιδιότητα:

- Κάθε αριθμός είναι ίσος με το άθροισμα των δύο προηγούμενων.
- Εξαιρούνται οι δύο πρώτοι αριθμοί που είναι ίσοι με 1.

Δηλαδή η ακολουθία fibonacci είναι η εξής:

1, 1, 2, 3, 5, 8, 13, 21, 34, 21 + 34, …

Ακολουθία Fibonacci

Στόχος μας είναι να υπολογίσουμε τον ν-οστό αριθμό της ακολουθίας fibonacci.

Σκεπτόμαστε ως εξής:

- Ο ν-οστός όρος της ακολουθίας f(ν) είναι ίσος με τον ν-1 – οστό όρο + τον ν-2 – οστό όρο της ακολουθίας. Δηλαδή ισχύει:
  $f(\nu) = f(\nu-1) + f(\nu-2)$

Ακολουθία Fibonacci

Επίσης λαμβάνοντας υπόψη ότι αν ν=1 ή ν=2 τότε, f(1) = 1 ή f(2) = 1, η συνάρτηση που μας δίνει τον ν-οστό όρο της ακολουθίας θα είναι:

```
Fibonacci(N)
{
    if(N = 1 ή N = 2)
        επέστρεψε το 1.
    else
        επέστρεψε το Fibonacci(N-1) + Fibonacci(N-2)
}
```

Ακολουθία Fibonacci

Και σε C:

```
int fibonacci(int N)
{
  if((N == 1) || (N == 2)) return 1;

  return fibonacci(N-1) + fibonacci(N-2);
}
```

Παραγοντικό

- Το παραγοντικό ενός αριθμού Ν είναι το γινόμενο όλων των αριθμών από το 1 μέχρι το Ν. Το συμβολίζουμε με Ν! και διαβάζουμε Ν παραγοντικό.
- Π.χ.:
  5! = 1 * 2 * 3 * 4 * 5 = 120
  3! = 1 * 2 * 3 = 6
  1! = 1

Παραγοντικό

Προσοχή! Εξ’ ορισμού θεωρούμε ότι το 0 παραγοντικό (0!) είναι ίσο με 1.

$$0! = 1$$

Παραγοντικό

Υπολογίζουμε για παράδειγμα το:
6! = 1 * 2 * 3 * 4 * 5 * 6

Όμως ξέρουμε ότι:

5! = 1 * 2 * 3 * 4 * 5

Άρα μπορούμε να γράψουμε ότι:

6! = 5! * 6

Συνεπώς για τον τυχαίο αριθμό Ν μπορούμε να πούμε ότι:
N! = (N-1)! * N

Παραγοντικό

Άρα λαμβάνοντας υπόψη μας την ειδική περίπτωση ότι 0! = 1 μπορούμε να φτιάξουμε μια συνάρτηση που υπολογίζει το παραγοντικό:

```
Fractorial(N)
{
   if(N = 1) επέστρεψε το 1
   else επέτρεψε το Fractorial(N-1) * N
}
```

Παραγοντικό

Και σε C:

```
int fractorial(int N)
{
  if(N == 0) return 1;

  return N * fractorial(N - 1);
}
```

Μέγιστος Κοινός Διαιρέτης

- Ο μέγιστος κοινός διαιρέτης δύο αριθμών α, β είναι ο μεγαλύτερος αριθμός ο οποίος διαιρεί και τον α και τον β.

  π.χ. ο μέγιστος κοινός διαιρέτης του 3 και του 9 είναι το 3
  ενώ του 11 και του 13 το 1.

Μέγιστος Κοινός Διαιρέτης

- Μια ιδέα για να βρούμε τον μέγιστο κοινό διαιρέτη δύο αριθμών α, β είναι να δοκιμάσουμε όλους τους αριθμούς από το 1 μέχρι τον μικρότερο από τους α, β και να δούμε ποιος είναι ο μεγαλύτερος που διαιρεί και τους 2.

Μέγιστος Κοινός Διαιρέτης

Δηλαδή:

```
Gcd(a, b, number)
{
   if ο number διαιρεί και τον a και τον b
        επέστρεψε τον number
   else επέστρεψε τον gcd(a, b, number-1)
}
```

Μέγιστος Κοινός Διαιρέτης

Και σε C:

```
int gcd(int a, int b, int number)
{
  if((a % number == 0) && (b % number == 0)) return number;

  return gcd(a, b, number-1);
}
```

Μέγιστος Κοινός Διαιρέτης

Ο προηγούμενος αλγόριθμος είναι μεν εύκολος, όμως για μεγάλους αριθμούς είναι πολύ αργός.

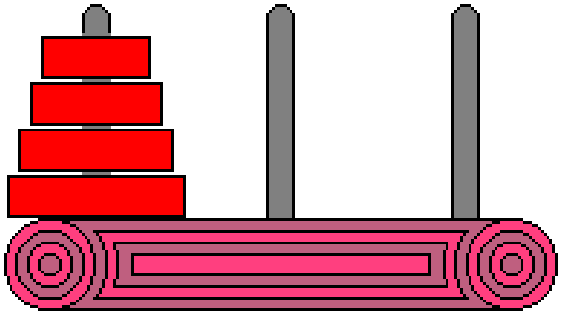
Ένας πιο γρήγορος αλγόριθμος είναι μια παραλλαγή του αλγορίθμου του Ευκλείδη.

```
int gcd(int a, int b)
{
  if(b == 0) return a;

  return gcd(b, a % b);
}
```

Πύργοι του Ανόι

•Έχουμε τρεις στήλες και ν δίσκους διαφορετικού μεγέθους τοποθετημένους στην πρώτη στήλη.
•Σκοπός μας είναι να μεταφέρουμε όλους τους δίσκους στην τελευταία στήλη, πραγματοποιώντας τις λιγότερες κινήσεις.



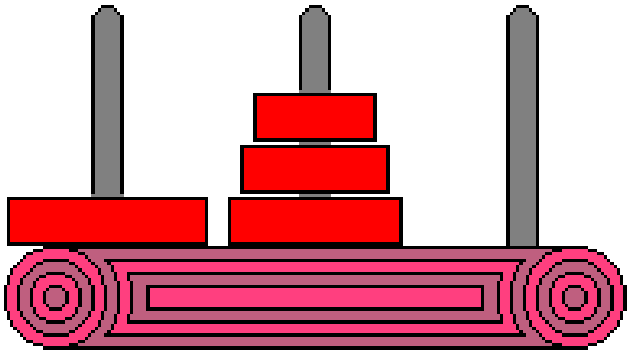
Κανόνες:

- Μπορούμε να μεταφέρουμε ένα μόνο δίσκο την φορά.
- Δεν μπορούμε να τοποθετήσουμε έναν μεγαλύτερο δίσκο πάνω σε έναν μικρότερο.

Πύργοι του Ανόι

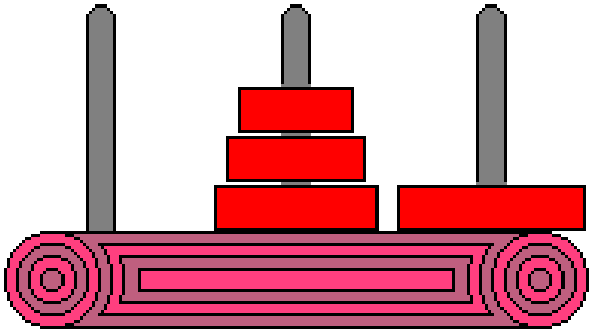
Για να λύσουμε αυτό το πρόβλημα σκεπτόμαστε ως εξής:

Αρχικά, μεταφέρουμε τους ν-1 δίσκους στην μεσαία στήλη (Έστω Β). (Δεν μας νοιάζει το πώς θα γίνει αυτό, κρατάμε μόνο την ιδέα για αρχή).



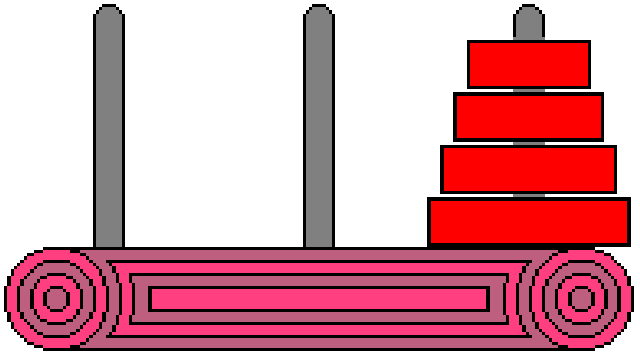
Πύργοι του Ανόι

Στη συνέχεια μεταφέρουμε τον ν-οστό δίσκο από την πρώτη στήλη (A) στην τρίτη (C).



Πύργοι του Ανόι

Τέλος, μεταφέρουμε τους ν-1 δίσκους από την μεσαία στήλη (B), στην τελευταία (C). (Και πάλι δεν μας νοιάζει το πώς θα γίνει αυτό, κρατάμε μόνο την ιδέα).



Πύργοι του Ανόι

- Έστω τώρα μια διαδικασία (με όνομα moveDiscs), η οποία μεταφέρει x δίσκους από μία στήλη σε μία άλλη.
- Η διαδικασία αυτή θα έχει ως παραμέτρους τον αριθμό των δίσκων που θα μεταφερθούν, την στήλη αφετηρίας, την στήλη προορισμού, και την στήλη που δεν θα χρησιμοποιηθεί.

moveDiscs(*αριθμός_δίσκων*, *στήλη_αφετηρίας*, *στήλη_προορισμού*, *στήλη_που_δεν_χρησιμοποιώ*)

Πύργοι του Ανόι

Όταν ο αριθμός των δίσκων που θέλουμε να μεταφέρουμε γίνει ίσος με 1, δηλαδή μεταφέρουμε ένα μόνο δίσκο, τότε αυξάνουμε τον αριθμό κινήσεων που έχουμε κάνει και η moveDiscs σταματά.

Σε κάθε άλλη περίπτωση ακολουθούμε την λογική που περιγράφτηκε πιο πάνω.

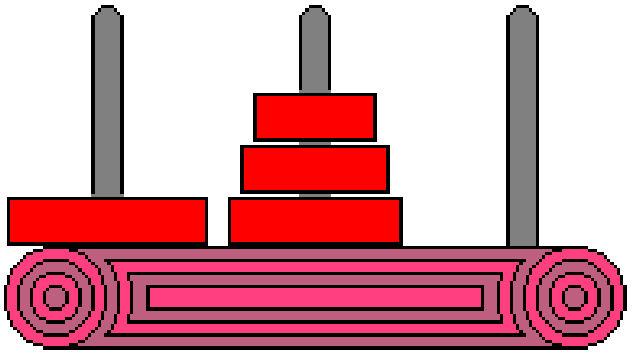
Δηλαδή:

Πύργοι του Ανόι

- Αρχικά, μεταφέρουμε τους ν-1 δίσκους στην μεσαία στήλη (Έστω Β).

  Δηλαδή εκτελούμε την

  moveDiscs(n-1, A, B, C);

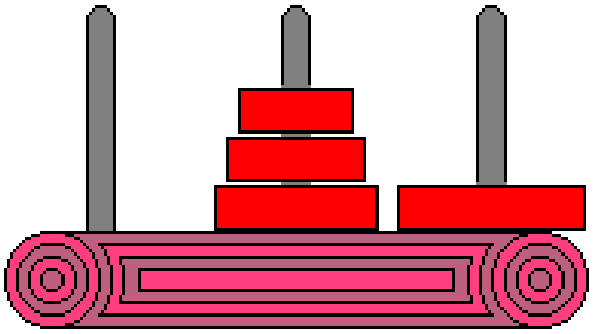


Πύργοι του Ανόι

- Στη συνέχεια μεταφέρουμε τον ν-οστό δίσκο από την πρώτη στήλη (A) στην τρίτη (C).

  Δηλαδή εκτελούμε την

  moveDiscs(1, A, C, B);

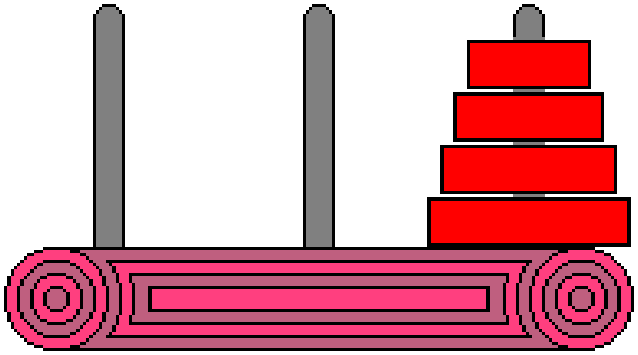


Πύργοι του Ανόι

- Τέλος, μεταφέρουμε τους ν-1 δίσκους από την μεσαία στήλη (B), στην τελευταία (C).

  Δηλαδή εκτελούμε την

  moveDiscs(n-1, B, C, A);



Πύργοι του Ανόι

Συνοψίζοντας όλα αυτά τα βήματα, η moveDiscs πρέπει να είναι η εξής:

```
moveDiscs(n, A, B, C)
{
    if(n = 1)
    {
        αύξησε τις συνολικές κινήσεις.
        σταμάτα.
    }
    else
    {
        moveDiscs(n-1, A, C, B);
        moveDiscs(1, A, B, C);
        moveDiscs(n-1, C, B, A);
    }
}
```

Πρέπει να προσέξουμε ότι στις προηγούμενες διαφάνειες τα A, B, C συμβόλιζαν τις στήλες (πρώτη, δεύτερη, τρίτη αντίστοιχα). Εδώ είναι οι παράμετροι της συνάρτησης για αυτό και διαφοροποιούνται.

Πύργοι του Ανόι

- Και σε C:

```c
#include <stdio.h>

int moves;

void moveDiscs(int num, int A, int B, int C)
{
  if(num == 1)
  {
    printf("%d -> %d\n", A, B);
    moves++;
    return;
  }

  moveDiscs(num - 1, A, C, B);
  moveDiscs(1, A, B, C);
  moveDiscs(num - 1, C, B, A);
}

int main()
{
  int nDiscs = 3;
  moveDiscs(nDiscs, 1, 3, 2);
  printf("Total Moves: %d\n", moves);
  return 0;
}
```

Στο κυρίως πρόγραμμα καλούμε την moveDiscs για τον συνολικό αριθμό των δίσκων, από την πρώτη στήλη, στην τρίτη, αφήνοντας κενή την δεύτερη.

Το Πρόβλημα των Ν-Βασιλισσών

- Έχουμε μια σκακιέρα με Ν γραμμές και Ν στήλες.
- Σκοπός μας είναι να τοποθετήσουμε Ν βασίλισσες στην σκακιέρα έτσι ώστε καμία να μην απειλείται από καμία άλλη.
- Προφανώς σε κάθε γραμμή και σε κάθε στήλη θα υπάρχει μόνο μια βασίλισσα.

Το Πρόβλημα των Ν-Βασιλισσών

Για να λύσουμε αυτό το πρόβλημα σκεπτόμαστε ως εξής:

- Αρχικά, ξεκινάμε από την πρώτη στήλη της σκακιέρας και δοκιμάζουμε να τοποθετήσουμε την βασίλισσα στην πρώτη γραμμή.

Το Πρόβλημα των Ν-Βασιλισσών

- Μετά, πηγαίνουμε στην δεύτερη στήλη και δοκιμάζουμε να τοποθετήσουμε την βασίλισσα στην πρώτη πάλι γραμμή. Προφανώς αυτό δεν γίνεται οπότε προσπαθούμε να τοποθετήσουμε την βασίλισσα στην δεύτερη γραμμή. Ούτε και αυτό γίνεται οπότε τοποθετούμε την βασίλισσα στην τρίτη γραμμή.

Το Πρόβλημα των Ν-Βασιλισσών

- Συνεχίζουμε την διαδικασία για τις επόμενες βασίλισσες στις επόμενες στήλες. Αν διαπιστώσουμε ότι δεν μπορούμε να τοποθετήσουμε πουθενά μια βασίλισσα, τότε αφαιρούμε την αμέσως προηγούμενη βασίλισσα που τοποθετήσαμε και την τοποθετούμε σε μια άλλη θέση. Μετά συνεχίζουμε κανονικά την διαδικασία.

Το Πρόβλημα των Ν-Βασιλισσών

- Όταν φτάσουμε στην Ν+1-οστή στήλη (πράγμα που σημαίνει ότι έχουμε ήδη τοποθετήσει την Ν-οστή βασίλισσα) σταματάμε. Το πρόβλημα έχει λυθεί.

Το Πρόβλημα των Ν-Βασιλισσών

- Δηλαδή αν η placeQueen είναι η διαδικασία που τοποθετεί μια βασίλισσα στην x στήλη της σκακιέρας τότε αυτή θα είναι:

```
placeQueen(στήλη)
{
    if(στήλη = N)
        τύπωσε την λύση και σταμάτα.
    else
        for γραμμή = 1 to γραμμή = N
            if(isLegal(γραμμή, στήλη))
            {
                τοποθέτησε την βασίλισσα
                placeQueen(στήλη + 1)
                αφαίρεσε την βασίλισσα της στήλης
            }
}
```

Το Πρόβλημα των Ν-Βασιλισσών

- Η συνάρτηση isLegal μας λέει αν μπορούμε να τοποθετήσουμε την βασίλισσα στην συγκεκριμένη θέση. Ελέγχει δηλαδή αν η συγκεκριμένη θέση απειλείται από κάποια άλλη βασίλισσα.

Το Πρόβλημα των Ν-Βασιλισσών

- Και σε C:

```
bool board[boardSize][boardSize];

void placeQueen(int col)
{
  if(col == boardSize) printSolution();

  for(int row = 0; row < boardSize; row++)
  {
    if(isLegal(row, col))
    {
      board[row][col] = true;
      placeQueen(col + 1);
      board[row][col] = false;
    }
  }
}
```

boardSize είναι το μέγεθος της σκακιέρας. Ο πίνακας board αντιπροσωπεύει την σκακιέρα. Όπου έχουμε false σημαίνει ότι δεν έχουμε τοποθετήσει βασίλισσα ενώ όπου έχουμε true, έχουμε τοποθετήσει βασίλισσα.

Το Πρόβλημα των N-Βασιλισσών

- Η συνάρτηση isLegal:

```
bool isLegal(int row, int col)
{
  int i, j;
  for(i = 0; i < boardSize; i++)
    if(board[row][i]) return false;
  for(i = 0; i < boardSize; i++)
    if(board[i][col]) return false;

  for(i = row, j = col; (i < boardSize) && (j < boardSize); i++, j++)
    if(board[i][j]) return false;
  for(i = row, j = col; (i >= 0) && (j >= 0); i--, j--)
    if(board[i][j]) return false;

  for(i = row, j = col; (i < boardSize) && (j >= 0); i++, j--)
    if(board[i][j]) return false;
  for(i = row, j = col; (i >= 0) && (j < boardSize); i--, j++)
    if(board[i][j]) return false;

  return true;
}
```

Ελέγχει αν υπάρχει βασίλισσα στην ίδια σειρά.

Ελέγχει αν υπάρχει βασίλισσα στην ίδια στήλη.

Ελέγχει αν υπάρχει βασίλισσα στην ίδια διαγώνιο. \

Ελέγχει αν υπάρχει βασίλισσα στην ίδια διαγώνιο. /

Το Πρόβλημα των Ν-Βασιλισσών

- Το κυρίως πρόγραμμα:

```
int main()
{
  for(int row = 0; row < boardSize; row++)
    for(int col = 0; col < boardSize; col++)
      board[row][col] = false;

  placeQueen(0);

  return 0;
}
```

Αρχικά κάνουμε όλο τον πίνακα board false γιατί δεν έχουμε τοποθετήσει καμία βασίλισσα.

Τοποθετούμε την πρώτη βασίλισσα στην πρώτη στήλη.