Алгоритмы
(ч.1)

Комбинаторика, перебор, последовательности…

Евгений Беркунский
http://berkut.homelinux.com
eberkunsky@gmail.com

Сгенерировать все подмножества данного n-элементного множества {0,.., n-1}

Сгенерировать все подмножества данного n-элементного множества {0,.., n-1}

Заведем массив B[0..n] из (n+1) элемента. B[i]=0, если i-ый элемент в подмножество не входит, и B[i]=1 иначе.

Таким образом, пустому подмножеству будет соответствовать набор из n нулей,
а n-элементному подмножеству - набор из n единиц.

Тут явно заметна связь подмножества с двоичным представлением числа.

Сгенерировать все подмножества данного n-элементного множества {0,.., n-1}

Сначала B[i]=0 для всех i, что соответствует пустому подмножеству.

Будем рассматривать массив B как запись двоичного числа B[N]...B[1]B[0]

Сгенерировать все подмножества данного n-элементного множества {0,.., n-1}

Сначала B[i]=0 для всех i, что соответствует пустому подмножеству.

Будем рассматривать массив B как запись двоичного числа B[N]...B[1]B[0]

и моделировать операцию сложения этого числа с единицей.

Сгенерировать все подмножества данного n-элементного множества {0,.., n-1}

Сначала B[i]=0 для всех i, что соответствует пустому подмножеству.

Будем рассматривать массив B как запись двоичного числа B[N]...B[1]B[0]

и моделировать операцию сложения этого числа с единицей.

При сложении будем просматривать число справа налево заменяя единичные биты нулями до тех пор, пока не найдем нулевой бит, в который занесем 1. Генерация подмножеств заканчивается, как только B[N]=1

Напечатать все последовательности длины N из чисел 1,2..M

Первая = (1,1,...,1)

Последняя = (M,M,...,M)

Напечатать все последовательности длины N из чисел 1,2..M

Первая = (1,1,...,1)

Последняя = (M,M,...,M)

Всего таких последовательностей будет M^N

Напечатать все последовательности длины N из чисел 1,2..M

Первая = (1,1,...,1)

Последняя = (M,M,...,M)

Всего таких последовательностей будет M^N
Чтобы понять, как должна действовать функция *next()*, начнем с примеров.

Пусть N=4,M=3.

Тогда:

next(1,1,1,1) -> (1,1,1,2)

next(1,1,1,3) -> (1,1,2,1)

next(3,1,3,3) -> (3,2,1,1)

Напечатать все последовательности длины N из чисел 1,2..M

Теперь можно написать саму функцию next():

```
boolean next() {
  //найти i: x[i]<M,X[i+1]=M,...,X[N]=M;
  X[i]=X[i]+1;
  X[i+1]=...=X[N]=1;
}
```

Напечатать все последовательности длины N из чисел 1,2..M

Рекурсивное решение:

```
void generate(int k) {
  if (k==N-1) {
    for (int i=0; i<N; i++) System.out.print(X[i]);
    System.out.println();
  } else {
    for (int j=0; j<M; j++) {
      X[k+1]=j+1;
      generate(k+1);
    }
  }
}
```

Напечатать все перестановки чисел 1..N

Первая перестановка: 1, 2, 3, …, N-2, N-1, N

Последняя перестановка: N, N-1, N-2, …, 3, 2, 1

Напечатать все перестановки чисел 1..N

Первая перестановка: 1, 2, 3, …, N-2, N-1, N

Последняя перестановка: N, N-1, N-2, …, 3, 2, 1

Всего таких перестановок будет

$N!=N*(N-1)*...*2*1$

Напечатать все перестановки чисел 1..N

Первая перестановка: 1, 2, 3, …, N-2, N-1, N

Последняя перестановка: N, N-1, N-2, …, 3, 2, 1

Всего таких перестановок будет

N!=N*(N-1)*...*2*1

Для составления алгоритма зададимся вопросом:

В каком случае i-ый член перестановки можно увеличить, не меняя предыдущих?

Напечатать все перестановки чисел 1..N

Ответ:

Если он меньше какого-либо из следующих членов (членов с номерами больше i).

Напечатать все перестановки чисел 1..N

Опишем алгоритм:

```
boolean next(){
//найти i: X[i]<X[i+1]>X[i+2]>...>X[N];
//найти j: X[j]>X[i]>X[j+1]>...>X[N];
//обменять X[i] и X[j];
//X[i+1]>X[i+2]>...>X[N];
//перевернуть X[i+1],X[i+2],...,X[N];
}
```

Напечатать все перестановки чисел 1..N

Рекурсивное решение:

```
void generate(int k) {
    if (k==N-1) {
        for(int i=0; i<N; i++) System.out.print(X[i]);
        System.out.println();
    } else {
        for (int j=k+1; j<N; j++) {
            swap(k+1,j);
            generate(k+1);
            swap(k+1,j);
        }
    }
}
```

Задача о восьми ферзях

Задача о восьми ферзях — широко известная задача по расстановке фигур на шахматной доске.

Исходная формулировка:

Расставить на стандартной 64-клеточной шахматной доске 8 ферзей так, чтобы ни один из них не находился под боем другого.

http://ru.wikipedia.org/wiki/Задача_о_восьми_ферзях

Задача о восьми ферзях

В более «математическом» виде задача может быть сформулирована несколькими способами, например, так:

Заполнить матрицу размером 8×8 нулями и единицами таким образом, чтобы сумма всех элементов матрицы была равна 8, при этом сумма элементов ни в одном столбце, строке или диагональном ряде матрицы не превышала единицы.

Задача о восьми ферзях

Конечная цель, поставленная перед решающим задачу, может формулироваться в нескольких вариантах:

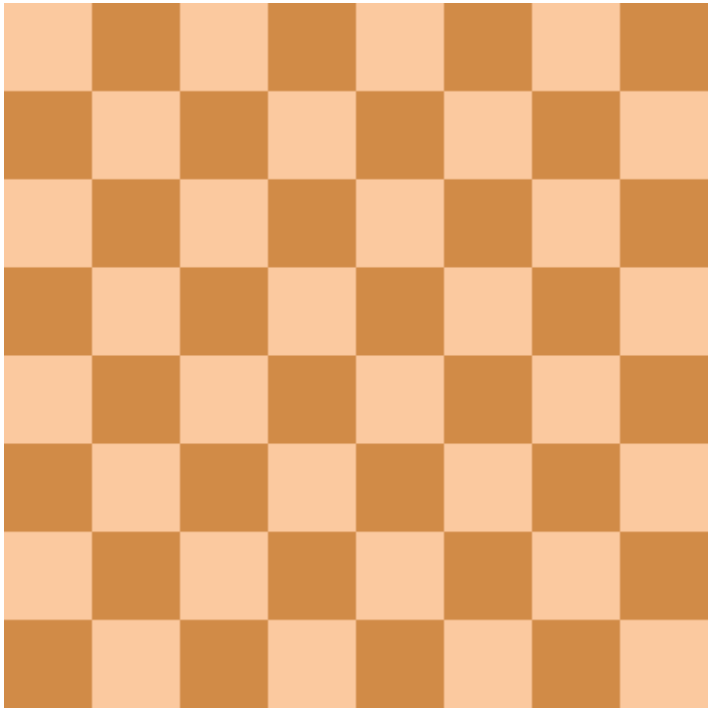
- Построить одно, любое решение задачи.
- Аналитически доказать, что решение существует.
- Определить количество решений.
- Построить все возможные решения.
- Одна из типовых задач по программированию алгоритмов перебора: создать компьютерную программу, находящую все возможные решения задачи.

Задача об N ферзях

Иногда постановка задачи требует нахождения способов расстановки N ферзей на доске N×N клеток

В этой, обобщенной постановке, задачу лучше всего решать методом перебора с отходом назад.

Задача о восьми ферзях (демо)



Задача о восьми ферзях

Функция «attempt»

```
void setupQueen(int i) {
    for (int j = 0; j < N; j++) {
        if (isFree(i - 1, j)) {
            put(i - 1, j);
            if (i == N) printSolution();
            else {
                setupQueen(i + 1);
            }
            delete(i - 1, j);
        }
    }
}
```

Задача о восьми ферзях

```
boolean isFree(int i, int j) {
    return hor[j] && diag1[i + j] &&
           diag2[i - j + N - 1];
}
void put(int i, int j) {
    x[i] = j + 1;
    hor[j] = false;
    diag1[i + j] = false;
    diag2[i - j + N - 1] = false;
}
void delete(int i, int j) {…}
```

Обход доски шахматным конем

Переборное решение:

- способ решения задачи, состоит, очевидно, в переборе с отходом назад. Ниже дана иллюстрация подхода для доски 8x8.
- Используем два одномерных массива row[64] и col[64] для хранения соответственно номеров строк и колонок, которые конь последовательно проходит по доске.
- Конь, находящийся в позиции (i, j), может следующим ходом оказаться в клетках с координатами
  (i-2, j+1), (i-1, j+2), (i+1, j+2), (i+2, ,j+1),
  (i+2, ,j-1), (i+1, j-2), (i-1, j-2), (i-2, j-1).

Если конь находится вблизи края доски, то некоторые ходы могут вызвать перемещение коня за ее пределы, что недопустимо.

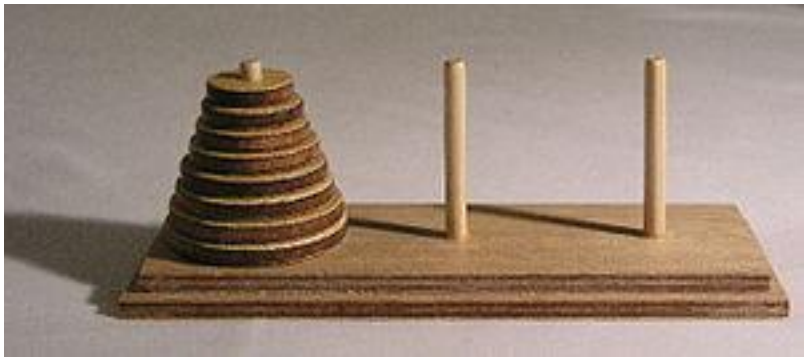
Ханойские башни

Есть три стержня A, B, и C.

На стержень A надето N дисков:

- наверху самый маленький,
- каждый следующий диск больше предыдущего,
- внизу самый большой.

На другие стержни дисков не надето.



Ханойские башни

- Необходимо перенести диски со стержня A на стержень C, пользуясь стержнем B, как вспомогательным, так, чтобы диски на стержне C располагались в том же порядке, в каком они располагаются на диске A перед перемещением.
- При перемещении никогда нельзя класть больший диск на меньший.

Ханойские башни

Рекурсивный метод

Для того, чтобы переложить всю пирамиду, надо:

- сначала переложить все, что выше самого большого диска, с первого на вспомогательный стержень;
- потом переложить самый большой диск с первого на третий стержень;
- а потом переложить оставшуюся пирамиду со второго на третий стержень, пользуясь первым стержнем, как вспомогательным.

Ханойские башни

```
void p(int n, char a, char b, char c) {
  if (n == 1) {
    System.out.println(a + " -> " + b);
  } else {
    p(n - 1, a, c, b);
    p(1, a, b, c);
    p(n - 1, c, b, a);
  }
}
```

Ханойские башни

Нерекурсивный метод:

- Если представить что стержни, на которые одеваются диски, расположены в углах равностороннего треугольника, то самый маленький диск каждым нечетным ходом движется по (или против, это зависит от первоначального количества дисков) часовой стрелке.
- Все четные ходы определяются однозначно. Какой диск меньше - тот и перекладывать (иначе противоречит условию).
  Т.к. трогать самый маленький диск нельзя и класть больший на меньший тоже нельзя.

Перечислить все разбиения N на целые положительные слагаемые

Пример: N=4.

разбиения: 1+1+1+1, 2+1+1, 2+2, 3+1, 4.

Первое разбиение = (1,1,...,1) - N единиц
Последнее разбиение = (N)

Перечислить все разбиения N на целые положительные слагаемые

Чтобы разбиения не повторялись, договоримся перечислять слагаемые в невозрастающем порядке.

Сказать, сколько их будет всего, не очень просто. Для составления функции next() зададимся таким вопросом:

В каком случае i-ый член разбиения можно увеличить, не меняя предыдущих?

Перечислить все разбиения N на целые положительные слагаемые

Ответ:

Во-первых, должно быть X[i-1]>X[i] или i=1.
Во-вторых, i должно быть не последним эл-том (увеличение i надо компенсировать уменьшением следующих).

Если такого i нет, то данное разбиение последнее.

Перечислить все разбиения N на целые положительные слагаемые

Увеличив i, все следующие элементы надо взять минимально возможными, т.е. равными единице:

```
void next() {
//найти i: (i<L) and ((X[i-1]>X[i]) or (i=1))
  X[i]:=X[i]+1;
  // L:= i + X[i+1]+...+X[L] - 1
  X[i+1]:=...:=X[L]:=1;
}
```

Через L обозначено количество слагаемых в текущем разбиении (понятно, что 1<=L<=N).

Перечислить все разбиения N на целые положительные слагаемые

```
int N,i,L;
int[]X;
int next(int[] X; int L) {
    int i=L-1;
    int s=X[L];
    //поиск i
    while (i>1 and X[i-1]<=X[i]) {
        s=s+X[i--];
    }
    X[i]++;
    L=i+s-1;
    for (int j=i+1; j<=L; j++) X[j]=1;
    return L;
}
```

Перечислить все разбиения N на целые положительные слагаемые

```
void run() {
  Scanner in = new Scanner(System.in);
  System.out.print(“N=“);
  int N = in.nextInt();
  int L=N;
  for (int i=1; i<=L; i++) X[i]:=1;
  for (int i=1; i<=L; i++)
    System.out.print(X[i]);
    System.out.println();
  do {
    L = next(X,L);
    for (int i=1; i<=L; i++) System.out.print(X[i]);
    System.out.println();
  } while (L>1);
}
```

Рекуррентные последовательности

Флаги.

Флаг состоит из n вертикальных полос белого, красного и синего цвета. Соседние полосы не могут иметь одинаковый цвет, а синяя полоса всегда должна находиться между красной и белой.

Сколькими способами можно покрасить флаг из n полос?

Вход. Файл INPUT.TXT содержит одно число – количество полос n ($1 \le n \le 45$) на флаге.

Выход. Запишите в файл OUTPUT.TXT одно число - количество способов, которыми можно покрасить флаг из n полос.

Вопросы и ответы

Q & A

Алгоритмы (ч.1)

Спасибо!

Евгений Беркунский
http://berkut.homelinux.com
eberkunsky@gmail.com