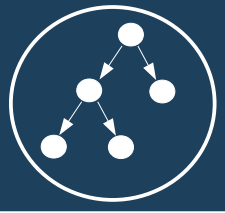


Data Structures and Algorithms

Генерация сочетаний с помощью алгоритма вращающейся двери

Сведение о алгоритме

Алгоритм был разработан W.H.Payne в 1979 году. Суть алгоритма что каждое сочетание получается путем замены одного (и только одного) элемента сочетания на элемент из еще не использованных. Называется так потому, как заменяемый элемент как бы выходит через вращающуюся дверь и в этот же момент через нее добавляется элемент который еще не использовался.

Сведение о алгоритме

Сложность по времени в наихудшем случае $O\left(\frac{n!}{k!(n-k)!}\cdot k\right)$

Затраты памяти $O(k+1)$

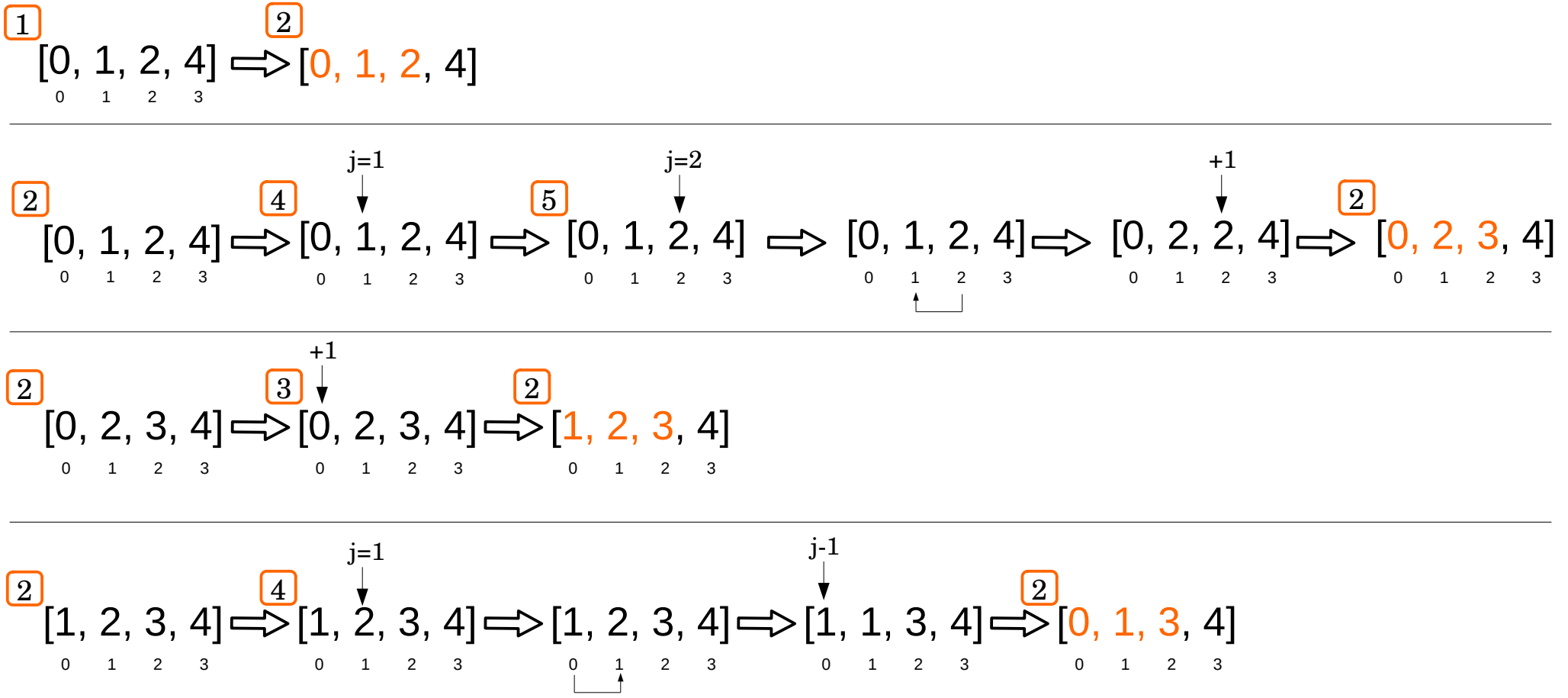
k — элементов выбранных из множества из n - элементов

Принцип работы алгоритма

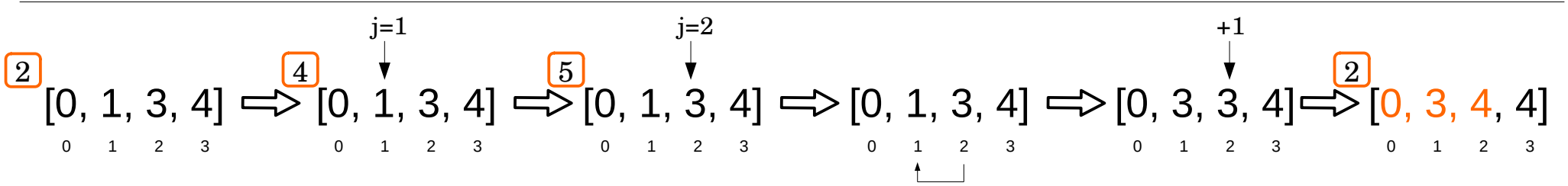
Генерируем все сочетания из n целых чисел [0,1,2...n-1] по k.

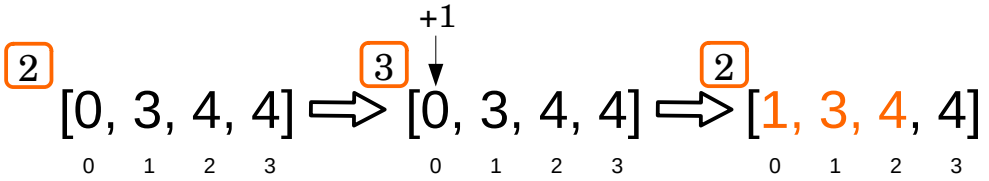
1) Создаем последовательность размером k+1 элемента (в дальнейшем c). Первые k — элементов устанавливаем равный индексу элемента. Элемент k+1 устанавливаем равным n-1. Вводим дополнительную переменную j. Перейти к **2**.
2) Возвращаем первые k элементов последовательность как очередное сочетание. Перейти к **3**.
3) Возможно два варианта:
   - k — нечетное. В случае если $c_0 + 1 < c_1$ установить $c_0 = c_0 + 1$ перейти к **2**. В противном случае установить j = 1 и перейти к **4**.
   - k — четное. В случае если $c_0 > 0$ установить $c_0 = c_0 - 1$ перейти к **2**. В противном случае установить j = 1 и перейти к **5**.
4) В случае $c_j > j$ установить $c_j = c_{j-1}$, $c_{j-1} = j-1$ и перейти к **2**. В противном случае установить j = j + 1 и перейти к **5**.
5) Если $c_j + 1 <= c_{j+1}$ установить $c_{j-1} = c_j$, $c_j = c_j + 1$ и перейти к **2**. В противном случае установить j = j + 1, если j<k перейти к **4**. В противном случае закончить алгоритм.

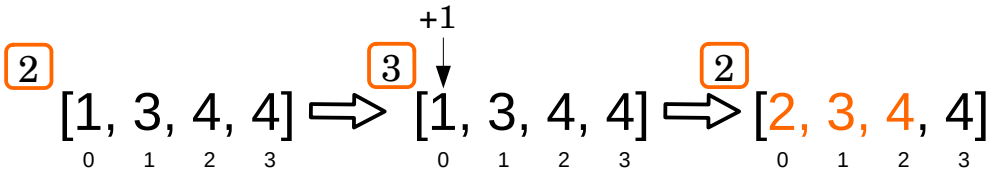
Графическая иллюстрация работы алгоритма (сочетание 3 из 5)

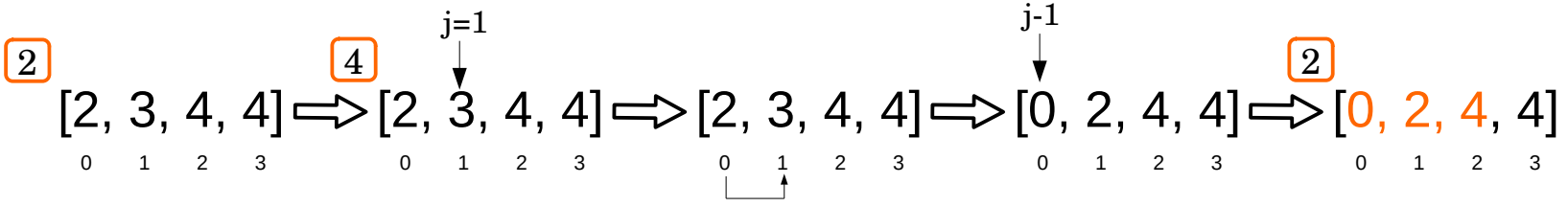


Графическая иллюстрация работы алгоритма (сочетание 3 из 5)

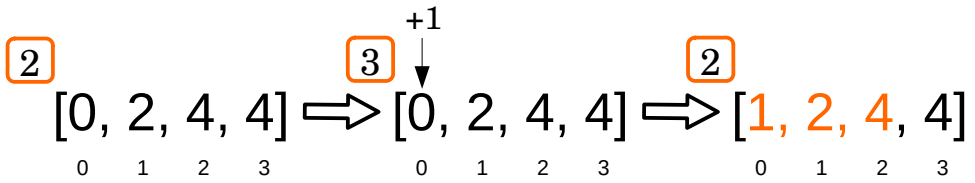


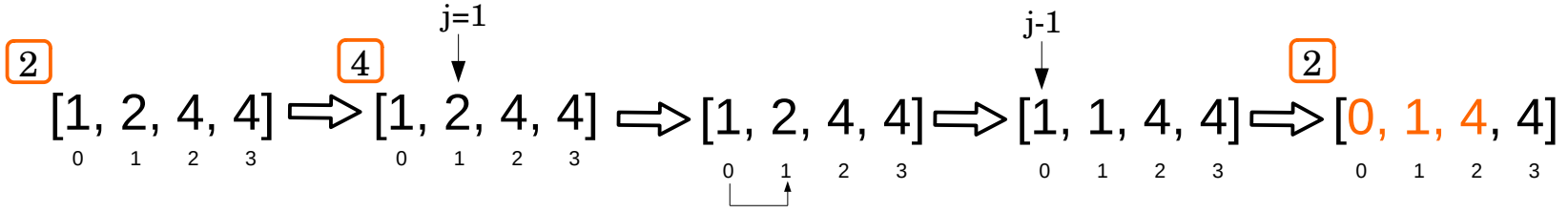


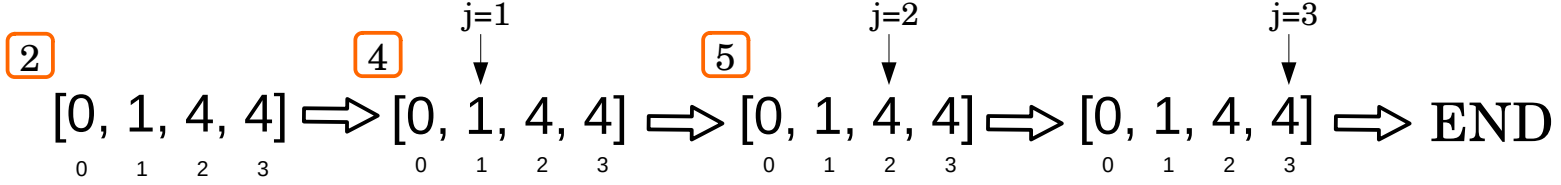




Графическая иллюстрация работы алгоритма (сочетание 3 из 5)







Python

Реализация алгоритма на Python

Python

Функция описывающая пункт 3 алгоритма

```
def check_first_element(c, k):
    j = 1
    if k % 2 != 0:
        if c[0]+1 < c[1]:
            c[0] += 1
            step = 2
        else:
            j = 1
            step = 4
    else:
        if c[0] > 0:
            c[0] -= 1
            step = 2
        else:
            j = 1
            step = 5
    return (step, j)
```

Python

Функции описывающие пункты 4-5 алгоритма

```
def decreas_element(c, j):
    if c[j] > j:
        c[j] = c[j-1]
        c[j-1] = j-1
        step = 2
    else:
        j += 1
        step = 5
    return (step, j)


def enlargement_element(c, j, k):
    if c[j]+1 <= c[j+1]:
        c[j-1] = c[j]
        c[j] += 1
        step = 2
    else:
        j += 1
        if j >= k:
            step = -1
        else:
            step = 4
    return (step, j)
```

Python

Функция для генерации сочетаний

```
def print_combination(k, n):
    c = [i for i in range(k)]
    c.append(n-1)
    step = 2
    j = 1
    while True:
        if step == 2:
            print(c[:k])
            step = 3
        elif step == 3:
            (step, j) = check_first_element(c, k)
        elif step == 4:
            (step, j) = decreas_element(c, j)
        elif step == 5:
            (step, j) = enlargement_element(c, j, k)
        else:
            break
```

Java

Реализация алгоритма на Java

Методы реализующие пункты 2 и 3 алгоритма

```
public static void printCurrentCombination(int[] c, int k) {
     System.out.println(Arrays.toString(Arrays.copyOf(c, k)));
}

public static int[] checkFirstElement(int[] c, int k) {
     int step;
     int j = 1;
     if (k % 2 != 0) {
          if (c[0] + 1 < c[1]) {
               c[0] += 1;
               step = 2;
          } else {
               step = 4;
          }
     } else {
          if (c[0] > 0) {
               c[0] -= 1;
               step = 2;
          } else {
               step = 5;
          }
     }
     return new int[] { step, j };
}
```

Методы реализующие пункты 4 и 5 алгоритма

```
public static int[] decreasElement(int[] c, int j) {
	int step;
	if (c[j] > j) {
		c[j] = c[j - 1];
		c[j - 1] = j - 1;
		step = 2;
	} else {
		j += 1;
		step = 5;
	}
	return new int[] { step, j };
}

public static int[] enlargementElement(int[] c, int j, int k) {
	int step;
	if (c[j] + 1 <= c[j + 1]) {
		c[j - 1] = c[j];
		c[j] += 1;
		step = 2;
	} else {
		j += 1;
		if (j < k) {
			step = 4;
		} else {
			step = -1;
		}
	}
	return new int[] { step, j };
}
```

Метод для генерации сочетаний

```
public static void printAllCombination(int k, int n) {
	int[] c = new int[k + 1];
	for (int i = 0; i < k; i++) {
		c[i] = i;
	}
	c[k] = n - 1;
	int step = 2;
	int j = 1;
	for (; step != -1;) {
		switch (step) {
			case 2:
				printCurrentCombination(c, k);
				step = 3;
				break;
			case 3:
				int[] r3 = checkFirstElement(c, k);
				step = r3[0];
				j = r3[1];
				break;
			case 4:
				int[] r4 = decreasElement(c, j);
				step = r4[0];
				j = r4[1];
				break;
			case 5:
				int[] r5 = enlargementElement(c, j, k);
				step = r5[0];
				j = r5[1];
				break;
		}
	}
}
```

Fortran

Реализация алгоритма на Fortran

Процедура реализующая пункт 3 алгоритма

```
subroutine check_first_element(c,k,step,j)
   integer,intent(inout)::c(0:)
   integer,intent(in)::k
   integer,intent(inout)::step,j
   if(mod(k,2)/=0) then
      if(c(0)+1<c(1)) then
         c(0) = c(0) + 1
         step = 2
      else
         j = 1
         step = 4
      end if
   else
      if(c(0)>0) then
         c(0) = c(0) - 1
         step = 2
      else
         j = 1
         step = 5
      end if
   end if
end subroutine check_first_element
```

Процедуры реализующие пункты 4-5 алгоритма

```
subroutine decreas_element(c,step,j)
   integer,intent(inout)::c(0:)
   integer,intent(inout)::step,j
   if(c(j) > j) then
      c(j) = c(j-1)
      c(j-1) = j-1
      step = 2
   else
      j = j + 1
      step = 5
   end if
end subroutine decreas_element

subroutine enlargement_element(c,k,step,j)
   integer,intent(inout)::c(0:)
   integer,intent(in)::k
   integer,intent(inout)::step,j
   if(c(j) + 1 <= c(j+1)) then
      c(j-1) = c(j)
      c(j) = c(j) + 1
      step = 2
   else
      j = j + 1
      if (j<k) then
         step = 4
      else
         step = -1
      end if
   end if
end subroutine enlargement_element
```

Процедура для генерации сочетаний

```
subroutine combination(k,n)
    integer,intent(in)::k,n
    integer::c(0:k)
    integer::j, step
    do j = 0,k-1
        c(j) = j
    end do
    c(k) = n-1
    step = 2
    do
        select case(step)
            case(2)
                write(*,*) c(0:k-1)
                step = 3
            case(3)
                call check_first_element(c,k,step,j)
            case(4)
                call decreas_element(c,step,j)
            case(5)
                call enlargement_element(c,k,step,j)
            case default
                exit
        end select
    end do
end subroutine combination
```

Список литературы

1) Д. Кнут. Искусство программирования. Том 4. Генерация всех сочетаний и разбиений, 3-е изд.