ФЕДЕРАЛЬНОЕ АГЕНТСТВО ВОЗДУШНОГО ТРАНСПОРТА

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«МОСКОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ
УНИВЕРСИТЕТ ГРАЖДАНСКОЙ АВИАЦИИ» (МГТУ ГА)

Кафедра прикладной математики

А.А. Егорова, Н.Л. Рогожникова

АЛГОРИТМЫ ДИСКРЕТНОЙ МАТЕМАТИКИ

Учебно-методическое пособие
по выполнению лабораторных работ

для студентов
направления 01.03.04
очной формы обучения

Москва
ИД Академии Жуковского
2020

УДК 519.6
ББК 517.8
Е30

Рецензент:
Феоктистова О.Г. – д-р техн. наук, доцент

Егорова А.А.
Е30 Алгоритмы дискретной математики [Текст] : учебно-методическое пособие по выполнению лабораторных работ / А.А. Егорова, Н.Л. Рогожникова. – М.: ИД Академии Жуковского, 2020. – 52 с.

Данное учебно-методическое пособие издается в соответствии с рабочей программой учебной дисциплины «Алгоритмы дискретной математики» по учебному плану для студентов направления 01.03.04 очной формы обучения.

Рассмотрено и одобрено на заседаниях кафедры 19.03.2020 г. и методического совета 19.03.2020 г.

УДК 519.6
ББК 517.8

В авторской редакции

Подписано в печать 22.10.2020 г.
Формат 60x84/16 Печ. л. 3,25 Усл. печ. л. 3,02
Заказ № 645/0818-УМП05 Тираж 90 экз.

Московский государственный технический университет ГА
125993, Москва, Кронштадтский бульвар, д. 20

Издательский дом Академии имени Н. Е. Жуковского
125167, Москва, 8-го Марта 4-я ул., д. 6А
Тел.: (495) 973-45-68
E-mail: zakaz@itsbook.ru

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Настоящее учебное пособие предназначено для студентов направления подготовки 01.03.04, изучающих дисциплину «Алгоритмы дискретной математики». Дисциплина относится к обязательным дисциплинам базовой части образовательной программы направления подготовки 01.03.04, квалификация бакалавр.

Алгоритмы дискретной математики - дисциплина, изучающая вычислительные алгоритмы над комбинаторными объектами.

Лабораторные работы по дисциплине состоят из следующих этапов:

1) домашняя подготовка;
2) выполнение работы на компьютере в соответствии с заданием;
3) сдача выполненной работы преподавателю на персональном компьютере;
4) распечатка результатов работы на принтере;
5) оформление отчета;
6) защита лабораторной работы.

ТРЕБОВАНИЯ И ПОРЯДОК ВЫПОЛНЕНИЯ ЛАБОРАТОРНЫХ РАБОТ

В процессе домашней подготовки студент:

- изучает лекционный материал, материалы данного пособия и дополнительной литературы по теме лабораторной работы;
- знакомится с заданием на выполнение лабораторной работы;
- готовит отчет по выполнению лабораторной работы (пункты, отмеченные знаком *).

Выполнение лабораторной работы производится во время занятий в классе ИВЦ МГТУГА в присутствии преподавателя. В процессе выполнения лабораторной работы студент последовательно выполняет задание. По завершению работы - демонстрирует преподавателю результаты.

Сдача работы преподавателю на персональном компьютере заключается в демонстрации результатов работы и выполнении непосредственно при преподавателе индивидуального задания.

После приема преподавателем лабораторной работы на ПК студент:

1 сохраняет результаты лабораторной работы на внешнем носителе;

2 распечатывает результаты на принтере на подготовленных листах формата А4.

Отчет по каждой лабораторной работе должен содержать:

- название работы*;
- цель лабораторной работы*;
- задание на выполнение лабораторной работы*;
- краткие комментарии по выполнению лабораторной работы*;
- анализ вычислительной (и пространственной) сложности алгоритмов;
- распечатки файлов программы и результатов.

Защита лабораторной работы преподавателю проводится при наличии оформленного отчета (распечатки должны быть жестко закреплены, например, вклеены) по контрольным вопросам. После защиты лабораторной работы преподаватель делает соответствующую запись на отчете студента.

В соответствии с Положением МГТУ ГА о зачетах и курсовых экзаменах студент, не защитивший двух работ, не допускается к выполнению следующей лабораторной работы.

1. Лабораторная работа № 1

1.1. Разработка и анализ целочисленных алгоритмов на примере алгоритма Евклида

Одним из самых старых и концептуальных алгоритмов теории чисел является алгоритм Евклида, предназначенный для вычисления наибольшего общего делителя (НОД) двух целых чисел.

Теорема (рекурсивная теорема о НОД).

Для $\forall$ ($a \geq 0$ и $a \in$ Z) и $\forall$ ($b > 0$ и $b \in$ Z): $\gcd(a, b) = \gcd(a, a \bmod b)$.

На рекурсивной теореме о НОД основан рекурсивный алгоритм Евклида:

Euclid (a,b)

1 **if** $b = 0$

2 **then return** a

3 **else return** Euclid(b,a mod b)

В качестве входных данных в алгоритме выступают любые неотрицательные целые числа.

Пример работы процедуры Euclid для вычисления gcd (30,21):

Euclid(30,21) = Euclid(21,9) = Euclid(9,3) = Euclid(3,0) = 3.

Корректность процедуры Euclid следует из теоремы рекурсивной теоремы о НОД. Так как во второй строке алгоритма возвращается значение a, то $b = 0$, а из элементарного свойства НОД следует, что

$\gcd(a,b) = \gcd(a,0) = a$.

Время работы алгоритма Euclid при размере входных данных a и b можно оценить, используя теорему Ламе.

Теорема (Теорема Ламе). Если для произвольного числа $k \geq 1$ выполняются условия $a > b \geq 0$ и $b < F_{k+1}$, то в вызове процедуры Euclid (a,b) производится менее k рекурсивных вызовов.

Поскольку число F_k равно $\varphi^k/\sqrt{5}$, где – золотое сечение $(1+\sqrt{5})/2$, то количество рекурсивных вызовов в процедуре Euclid (a,b) равно $O(\log_2 b)$.

Таким образом, если с помощью алгоритма Евклида обрабатывается два β – битовых числа, то в нем производится $O(\beta)$ арифметических операций и $O(\beta^3)$ битовых операций (в предположении, что при умножении и делении β – битовых чисел выполняется $O(\beta^2)$ битовых операций).

Рассмотрим расширенный алгоритм Евклида.

Пусть выполняется $d = gcd(a,b) = ax + by$.

Построим алгоритм для вычисления НОД и нахождения целых коэффициентов x *и* y*:*

Extended-Euclid(a,b)

1 **if** b = 0

2 **then return** (a,1,0)

3 $(d',x',y') \leftarrow$ Extended-Euclid $(b, a \bmod b)$

4 $(d,x,y) \leftarrow (d',x',y' - \lfloor a/b \rfloor y')$

5 **return** (d,x,y)

В таблице представлена работа расширенного алгоритма Евклида:

Работа алгоритма Extended-Euclid(a,b) с входными числами 99 и 78					
a	b	$\lfloor a/b \rfloor$	d	x	y
99	78	1	3	-11	14
78	21	3	3	3	-11
21	15	1	3	-2	3
15	6	2	3	1	-2
6	3	2	3	0	1
3	0	-	3	1	0

В каждой строке таблицы показан один уровень рекурсии: входные величины a и b, вычисленная величина $\lfloor a/b \rfloor$, а также возвращаемые величины d, x и y. Возвращаемая тройка значений (d,x,y) становится тройкой, которая используется в ходе вычислений на следующем, более высоком уровне рекурсии.

Процедура Extended-Euclid(a,b) представляет собой разновидность процедуры Euclid(a,b). Первая строка в ней эквивалентна проверке равенства значения b нулю в первой строке процедуры Euclid. Если $b = 0$, то процедура Extended-Euclid во второй строке возвращает не только значение $d = a$, но и коэффициенты $x = 1$ и $y = 0$, так что $a = ax + by$. Если $b \neq 0$, то процедура Extended-Euclid сначала вычисляет набор величин (d',x',y'), таких что

$$d' = \gcd(b, a \bmod b) \text{ и}$$

$$d' = bx' + (a \bmod b)y'. \qquad (1.1)$$

Как и в процедуре Euclid, в этом случае имеем

$$d = \gcd(a, b) = d' = \gcd(b, a \bmod b). \qquad (1.2)$$

Чтобы получить значения x и y, для которых выполняется равенство $d = gcd(a,b) = ax + by$, сначала перепишем уравнение (1.1) с использованием равенств $d = d'$ и $\gcd(a, b) = \gcd(|a|, |b|)$:

$$d = bx' + (a - \lfloor a/b \rfloor b)\, y' = ay' + b(x' - \lfloor a/b \rfloor y'). \qquad (1.3)$$

Таким образом, при выборе величин $x = y'$ и $y' = x' - \lfloor a/b \rfloor y'$ удовлетворяется уравнение $d = ax + by$, что доказывает корректность алгоритма в процедуре Extended-Euclid.

Поскольку количество вызовов в процедуре Euclid равно количеству вызовов в процедуре Extended-Euclid, то время работы процедуры Euclid с точностью до постоянного множителя равно времени работы процедуры Extended-Euclid. Таким образом, $a > b > 0$ количество рекурсивных вызовов равно $O(\log_2 b)$.

```
1   Вход: a, b ∈ N
2   Переменные: x, y ∈ N
3   x := a
4   y := b
5   Цикл Пока x ≠ y
6           Если x > y, то
7                   x := x – y
8           Иначе
9                   x := y - x
10          Конец Если
11  Конец Цикла
12  d := x
```

Рисунок 1 - Простейший итеративный алгоритм Евклида для натуральных чисел.

```
1   Вход: a, b ∈ N
2   Переменные: x, y ∈ N
3   x := a; y := b; d := 1
4   Цикл Пока (x mod 2 = y mod 2 = 0)
5                   d = 2·d; x := x/2; y := y/2
6   Конец Цикла
7   Цикл Пока x ≠ y
8         Выбор
9               если x mod 2 = 0, то x := x/2
10              если y mod 2 = 0, то y := y/2
11              если x > y, то x := x - y
12              если x < y, то y := y - x
13        Конец Выбора
14  Конец Цикла
15  d = d·x
```

Риунок 2 - Бинарный итеративный алгоритм Евклида для натуральных чисел.

```
1   Вход: a, b ∈ N
2   Переменные: r, x, y ∈ N
3   x := a
4   y := b
5   Цикл Пока y ≠ 0
6         r = x mod y
7         x := y
8         y := r
9   Конец Цикла
10  d = x
```

Рисунок 3 - Итеративный алгоритм Евклида для натуральных чисел.

1.2. Задание на выполнение лабораторной работы

Цель работы: разработка итеративных и рекурсивных алгоритмов на множестве целых чисел и их анализ.

Варианты заданий на выполнение лабораторной работы № 1.

Вариант № 1, 12, 13, 22

1) Реализуйте алгоритмы нахождения наибольшего общего делителя двух целых чисел: используя рекурсивный алгоритм Евклида и простейший алгоритм вычисления НОД. Проведите сравнительный анализ двух алгоритмов.

2) Реализуйте рекурсивный алгоритм нахождения НОД и коэффициентов его разложения в линейную комбинацию.

Вариант № 2, 14, 23

1) Реализуйте алгоритмы нахождения наибольшего общего делителя двух целых чисел: используя итеративный алгоритм Евклида и рекурсивный алгоритм Евклида. Проведите сравнительный анализ двух алгоритмов

2) Реализуйте рекурсивный алгоритм нахождения НОД и коэффициентов его разложения в линейную комбинацию.

Вариант № 3, 6, 11, 19, 25

1) Реализуйте алгоритмы нахождения наибольшего общего делителя двух целых чисел: используя итеративный алгоритм Евклида и бинарный алгоритм вычисления НОД. Проведите сравнительный анализ двух алгоритмов.

2) Реализуйте рекурсивный алгоритм нахождения НОД и коэффициентов его разложения в линейную комбинацию.

Вариант № 4, 8, 20, 16

1) Реализуйте алгоритмы нахождения наибольшего общего делителя двух целых чисел: используя рекурсивный алгоритм Евклида и бинарный алгоритм вычисления НОД. Проведите сравнительный анализ двух алгоритмов.

2) Реализуйте рекурсивный алгоритм нахождения НОД и коэффициентов его разложения в линейную комбинацию.

Вариант № 5, 9, 17, 24

1) Реализуйте алгоритмы нахождения наибольшего общего делителя двух целых чисел: используя простейший алгоритм вычисления НОД и бинарный алгоритм вычисления НОД. Проведите сравнительный анализ двух алгоритмов.

2) Реализуйте рекурсивный алгоритм нахождения НОД и коэффициентов его разложения в линейную комбинацию.

Вариант № 7, 10, 15, 18, 21

1) Реализуйте алгоритмы нахождения наибольшего общего делителя двух целых чисел: используя простейший алгоритм вычисления НОД и бинарный алгоритм вычисления НОД. Проведите сравнительный анализ двух алгоритмов.

2) Реализуйте рекурсивный алгоритм нахождения НОД и коэффициентов его разложения в линейную комбинацию.

1.3. Контрольные вопросы

1. Докажите, что для всех целых чисел a, k и n выполняется соотношение $\gcd(a, b) = \gcd(a + kn, b)$.
2. Докажите, что из уравнений $a = p_1^{e_1} p_2^{e_2} p_3^{e_3} \dots p_r^{e_r}$ и $b = p_1^{f_1} p_2^{f_2} p_3^{f_3} \dots p_r^{f_r}$ следует $\gcd(a, b) = p_1^{\min(e_1,f_1)} p_2^{\min(e_2,f_2)} \dots p_r^{\min(e_r,f_r)}$
3. Дайте краткий сравнительный анализ итеративных алгоритмов Евклида по их вычислительной сложности.
4. Какие значения возвращает процедура Extended_Euclid (F_{k+1},F_k)? Докажите верность вашего ответа.
5 Что такое рекурсивный алгоритм?
6. Что такое бинарный алгоритм?
7. Сформулируйте теорему Ламе.

2. Лабораторная работа № 2

2.1. Алгоритмы решения задачи о рюкзаке(ранце)

Рассмотрим задачу о рюкзаке, а именно ее дискретную и непрерывную формулировки. Классическая задача о рюкзаке (о загрузке) известна очень давно.

Дискретная задача о рюкзаке:

У нас имеется ***n*** предметов. Предмет под номером i имеет стоимость v_i руб. и вес w_i кг, где w_i и v_i - целые числа. Нужно унести вещи, суммарная стоимость которых была бы как можно большей, однако грузоподъемность рюкзака ограничивается W килограммами, где W - целая величина. Какие предметы следует взять с собой?

Непрерывная задача о рюкзаке:

Формулировка та же, что и в дискретной задаче, только теперь тот или иной товар можно брать с собой частично, а не делать каждый раз бинарный выбор – брать или не брать (0-1).

В обеих задачах о рюкзаке проявляется свойство оптимальной подструктуры.

Рюкзак называется **сверхвозрастающим**, если в нем каждый элемент, начиная со второго, больше суммы двух предыдущих, т.е.

$$a_i > \sum_{j=i-2}^{i-1} a_j \quad \forall i \geq 2. \tag{2.1}$$

Существует несколько модификаций задачи:

- Каждый предмет можно брать только один раз;

- Каждый предмет можно брать сколько угодно раз;

- Каждый предмет можно брать определенное количество раз;

- На размер рюкзака имеется несколько ограничений;

- Некоторые вещи имеют больший приоритет, чем другие.

Алгоритмы**,** предназначенные для решения задач оптимизации, обычно представляют собой последовательность шагов, на каждом из которых предоставляется некоторое множество выборов. Определение наилучшего

выбора, руководствуясь принципами динамического программирования, во многих задачах оптимизации не является целесообразным: для этих задач подходят более простые и эффективные алгоритмы.

Алгоритмы решения задачи о рюкзаке можно разделить на два типа: **точные** и **приближенные**. Точные: применение *динамического программирования* (ДП), *полный перебор*, *метод ветвей и границ* (сокращение полного перебора). Приближенные алгоритмы: *жадный алгоритм*.

Полный перебор – перебор всех вариантов (всех состояний) – малоэффективный, но точный метод. **Метод ветвей и границ** – по сути сокращение полного перебора с отсечением заведомо плохих решений. **ДП** – алгоритм, основанный на принципе оптимальности Беллмана. **Жадный алгоритм** основан на нахождении относительно хорошего и дешевого решения.

Большинство используемых алгоритмов имеют полиномиальное время работы, если размер входных данных – $\boldsymbol{n}$, то время их работы в худшем случае оценивается как $O(n^k)$, где $\boldsymbol{k}$ - константа. Но встречаются задачи, которые нельзя разрешить за полиномиальное время. Это класс NP - полных задач. Задача называется NP - полной, если для нее не существует полиномиального алгоритма. Алгоритм называется полиномиальным, если его сложность $O(\boldsymbol{n})$ в худшем случае ограничена сверху некоторым многочленом (полиномом) от $\boldsymbol{n}$.

2.1.1 Жадный алгоритм

В случае применения жадного алгоритма сначала сортируем предметы по убыванию стоимости единицы каждого.

Рассмотрим в целочисленной задаче наиболее ценную загрузку, вес которой не превышает W кг. Если вынуть из рюкзака предмет под номером j, то остальные предметы должны быть наиболее ценными, вес которых не превышает $W - w_j$ и которые можно составить из n-1 исходных предметов, из множества которых исключен предмет под номером j. Для аналогичной непрерывной задачи можно провести подобные же рассуждения. Если удалить

из оптимально загруженного рюкзака товар с индексом j, который весит w кг, остальное содержимое рюкзака будет наиболее ценным, состоящим из n -1 исходных товаров, вес которых не превышает величину W - w плюс $w_j - w$ кг товара с индексом j.

Несмотря на сходство сформулированных выше задач, непрерывная задача о рюкзаке допускает решение на основе жадной стратегии, а дискретная - нет. Чтобы решить непрерывную задачу, вычислим сначала стоимость единицы веса v_i/w_i каждого товара. Придерживаясь жадной стратегии, мы сначала загружаем как можно больше товара с максимальной удельной стоимостью (за единицу веса). Если запас этого товара исчерпается, а грузоподъемность рюкзака – нет, загружаем как можно больше товара, удельная стоимость которого будет второй по величине. Так продолжается до тех порт, пока вес товара не достигает допустимого максимума. Таким образом, вместе с сортировкой товаров по их удельным стоимостям время работы алгоритма будет равно $O(n \lg n)$.

Чтобы убедиться, что подобная жадная стратегия не работает в целочисленной задаче о рюкзаке, рассмотрим пример, проиллюстрированный на рисунке 4.

Имеется три предмета и рюкзак, способный выдержать 50 кг. Первый предмет весит 10 кг и стоит 60 рублей. Второй предмет весит 20 кг и стоит 100 рублей. Третий предмет весит 30 кг и стоит 120 рублей. Таким образом, один килограмм первого предмета стоит 6 рублей, что превышает аналогичную величину для второго (5 рублей) и третьего (4 рубля) предметов. Поэтому жадная стратегия должна состоять в том, чтобы сначала взять первый предмет. Однако, как видно на рисунке б), оптимальное решение - взять второй и третий предмет, а первый - оставить. Оба возможных решения, включающих в себя первый предмет, не являются оптимальными.

Для непрерывной же задачи жадная стратегия, при которой сначала загружается первый предмет, позволяет получить оптимальное решение, как показано на рисунке в).

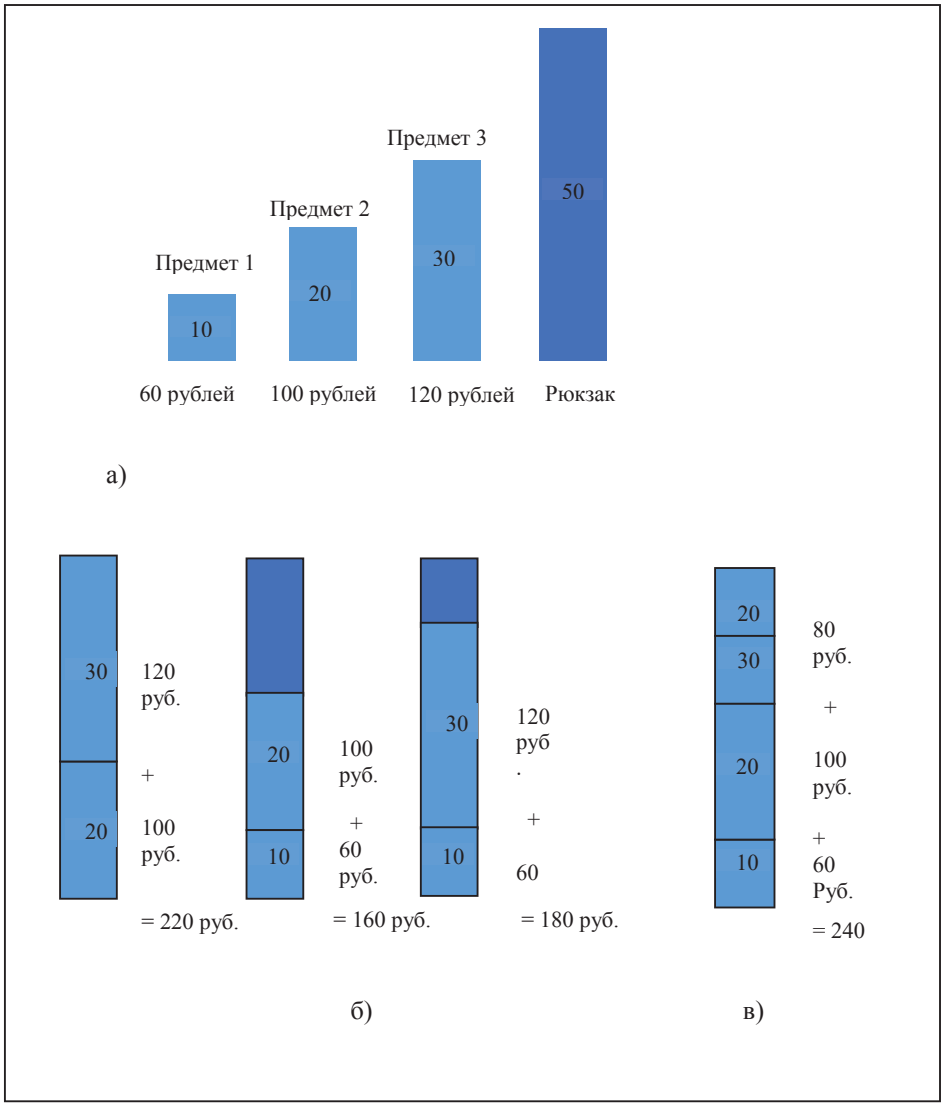

Рисунок 4 - Жадная стратегия не является точной для дискретной задачи о рюкзаке.

Ниже на рисунке 5 приведена процедура решения задачи о рюкзаке жадным методом:

```
procedure greedy(Take, sizeof(Take), False);
var i:integer;
begin
 i:= 0; {обнуляем список взятых предметов}
While NowWeight+W[i+1] < MaxWeight do
{пока текущий вес набора + следующий предмет,}
{который будет взят меньше предела вместимости}
begin
inc(i);
    {увеличиваем сумму цен на цену текущего предмета}
BestPrice := BestPrice + V[i];
    {увеличиваем сумму весов на вес тек. предмета}
NowWeight := NowWeight + W[i];
    {записываем что взяли этот предмет}
Take[i] := true;
end;
end;
```

Рисунок 5 - Процедура решения задачи о рюкзаке жадным методом.

2.1.2 Метод динамического программирования

В основе метода динамического программирования лежит принцип оптимальности Беллмана: каково бы ни было состояние системы перед очередным шагом, надо выбирать управление на этом шаге так, чтобы выигрыш на этом шаге плюс оптимальный выигрыш на всех последующих шагах был оптимальным. Проще говоря, оптимальное решение на *i* шаге находится исходя из найденных ранее оптимальных решений на предшествующих шагах. Из этого следует, что для того чтобы найти оптимальное решение на последнем шаге, надо сначала найти оптимальное решения для первого, затем для второго и так далее пока не пройдем все шаги до последнего.

Имеется набор из N предметов. Пусть $MaxW$ - объем рюкзака, V_i – стоимость i-го предмета, W_i – вес i-го предмета. Value [W, i] – максимальная сумма, которую надо найти. Суть метода динамического программирования – на каждом шаге по весу $1 < W_i < W$ находим максимальную загрузку Value [W_i, i], для веса W_i.

Допустим, мы уже нашли максимальную сумму Value [1..W, 1..i-1] для веса меньше либо равного W и с предметами, взятыми из 1..N-1. Рассмотрим предмет N, если его вес W_N меньше W проверим, стоит ли его брать. Если его взять, то вес станет W-W_i, тогда Value [W, i] = Value [W– W_i , i-1] + V_i (для Value [W– W_i , i-1]). То есть решение уже найдено, и остается только прибавить V_i. Если его не брать то вес останется тем же и Value [W , i] = Value [W– W_i , i- 1]. Из двух вариантов выбирается тот, который дает наибольший результат.

Динамическое программирование для задачи о рюкзаке дает точное решение, причем одновременно вычисляются решения для всех размеров рюкзака от 1 до MaxW. Но для хранения таблицы стоимости и запоминания того, брался каждый предмет или нет, требуется порядка O (N*MaxW) памяти, а временная сложность алгоритма так же будет равна O (N*MaxW).

Ниже на рисунке 6 приведен основной алгоритм решения задачи о рюкзаке методом динамического программирования:

```
{Загружаем рюкзак, если его вместимость равна Weight}
for Weight:=1 to MaxW do
begin
  for i:=1 to N do {берем предметы с 1 по N}
                   {если вес предмета больше Weight}
                   {или предыдущий набор лучше выбираемого}
    if (W[i]>Weight) or (Value[Weight, i-1] >=  Value[Weight-W[i], i-1]+V[i])
then   begin
```

```
                {Тогда берем предыдущий набор}
                 Value[Weight, i]:=Value[Weight, i-1];

                {делаем пометку, что вещь i не взята}
                 Take [Weight, i]:= false;

        end

{иначе добавляем к предыдущему набору текущий предмет}

      else

      begin

Value [Weight, i]:=Value [Weight - W[i], i-1]
+P[i];

{говорим что вещь i взята}
Take [Weight, i]:= true;
        end;
end;
```

Рисунок 6 - Процедура решения задачи о рюкзаке методом динамического программирования.

2.1.3 Метод простого перебора

Название метода говорит само за себя. Чтобы получить решение нужно перебрать все возможные варианты загрузки. Здесь мы будем рассматривать следующую постановку задачи. В рюкзак загружаются предметы N различных типов (количество предметов каждого типа не ограничено), каждый предмет типа i имеет вес W_i и стоимость V_i , i= 1,2..N. Требуется определить максимальную стоимость груза, вес которого не превышает W. Очевидна простая рекурсивная реализация данного подхода (см. рис 7.). Временная сложность данного алгоритма равна $O(N!)$. Алгоритм имеет сложность как функцию факториал и может работать лишь с небольшими значениями N. С ростом N, число вариантов очень быстро растет, и задача становится практически неразрешимой методом полного перебора. На рисунке 8 показано дерево перебора, дерево имеет 4 уровня. В каждом кружочке показан вес предмета, корень дерева – нулевой вес, то есть когда рюкзак пуст. Первый пред-

мет можно выбрать четырьмя способами, второй – тремя, третий – двумя, а дальше можем взять только один оставшийся предмет.

Пусть N - количество предметов, а MaxW - объем рюкзака, V_i – стоимость i-го предмета, W_i – вес i-го предмета.

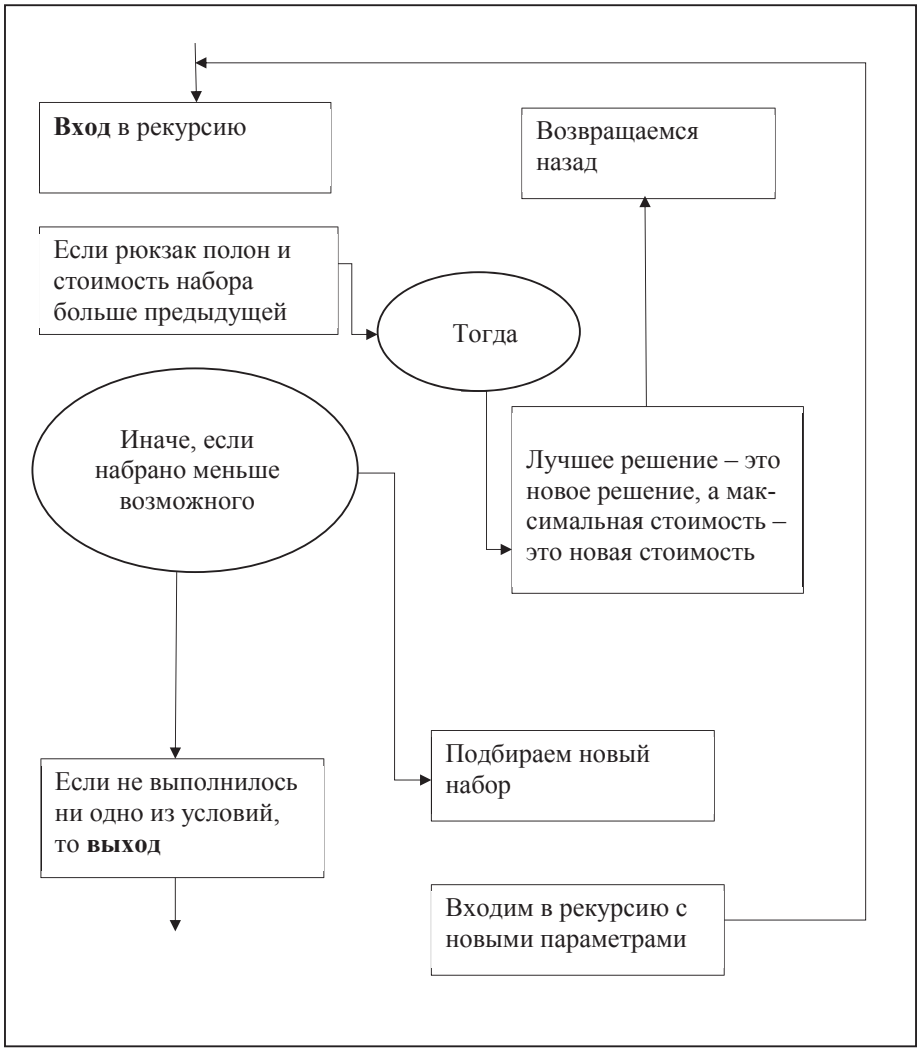

Рисунок 7 – Схема рекурсивного решения задачи о рюкзаке методом полного перебора.

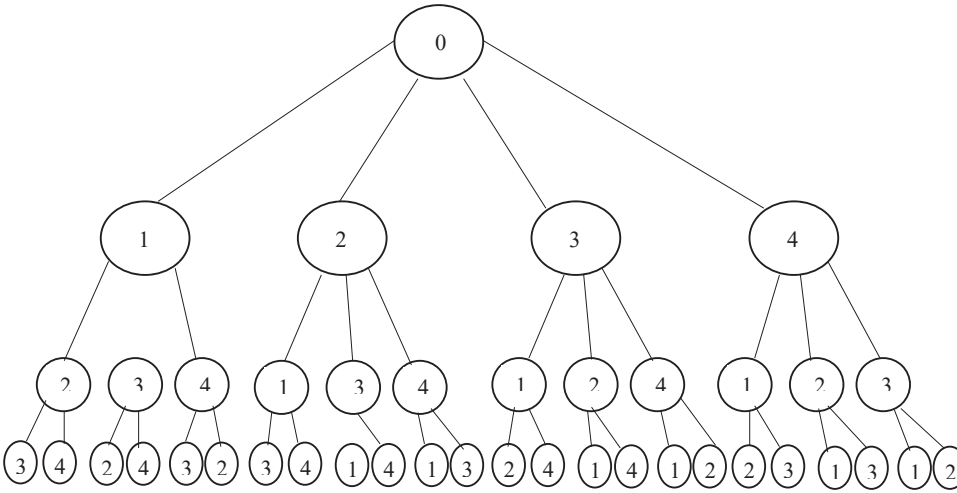

Рисунок 8 – Дерево перебора решения задачи о рюкзаке.

Ниже на рисунке 9 приведена процедура решения задачи о рюкзаке методом перебора:

```
{передаем Nab - номер набранной группы, OstW-вместимость}
{stoim - цена набранного (еще не набрали нисколько)}
Procedure Search (Nab, OstW, Stoim:integer);
var i:integer;
begin
   {здесь OstW-вес который следует набрать из оставшихся.}
    {Stoim - стоимость текущего решения}
    {Nab - набор предметов если наполнили рюкзак}
    {и набрали стоимость больше чем имеется, то считаем}
    {это новым решением}
```

```
if (Nab > N) and (Stoim > Max) then
    begin    {найдено решение}
    BestP:=NowP;
    Max:=Stoim;
          end
        {иначе если количество взятых <= объема,  забиваем рюкзак дальше}
      else if Nab<=N then
        {иначе если набрано меньше, чем помещается в рюкзак}
  for i:=0 to OstW div W[Nab] do
     begin {идем от 0 до ост. места}
          NowP[Nab]:=i; {берем предмет Nab пока есть место в ранце}
          Search(Nab+1,OstW-i*W[Nab],Stoim+i*P[Nab]);
   {после каждого взятия предмета увеличиваем стоимость набора и}
   {уменьшаем место в рюкзаке на вес предмета, так же увеличиваем}
   {количество раз взятия предмета}
  end;
end;
```

Рисунок 9 - Процедура решения задачи о рюкзаке методом перебора.

2.2. Задание на выполнение лабораторной работы

Цель работы:

разработка и анализ алгоритмов решения задачи о рюкзаке.

Варианты заданий на выполнение лабораторной работы № 2

Для задания граничных условий при реализации алгоритма используйте константы.

№ 1, 4, 7, 10, 13, 16, 19, 22: Разработайте программную реализацию Наивного алгоритма (полный перебор) решения задачи о рюкзаке и проведите его анализ.

№ 2, 5, 8, 11, 14, 17, 20, 23: Разработайте программную реализацию алгоритма решения задачи о сверхвозрастающем рюкзаке и проведите его анализ.

№ 3, 6, 9, 12,15, 18, 21: Разработайте программную реализацию решения задач о рюкзаке с помощью жадного метода.

2.3. Контрольные вопросы

1. Назовите основные алгоритмы решения задачи о рюкзаке.
2. Какова временная сложность жадного алгоритма при решении задачи о рюкзаке?
3. Какова временная сложность алгоритма решения задачи о рюкзаке методом динамического программирования (ДП)?
4. Какова временная сложность алгоритма решения задачи о рюкзаке методом полного перебора?
5. Какие два типа алгоритмов существует при решении задачи о рюкзаке? Чем они отличаются?
6. Верно ли, что жадный метод дает более точное решение для дискретного рюкзака, чем для непрерывного?
7. Верно ли, что метод решения с помощью ДП наиболее быстрый по сравнению с другими для решения задачи о рюкзаке?
8. Методом какого алгоритма является метод ветвей и границ при решении задачи о рюкзаке?
9. Какой алгоритм называют наивным?

3. Лабораторная работа № 3

3.1.1 Алгоритмы генерации основных комбинаторных объектов

Рассмотрим задачу обхода всех возможных объектов некоторой совокупности.

Эту задачу можно рассматривать как перечисление всех возможностей, то есть подсчет общего количества вариантов, а можно рассматривать как задачу составления списка всех возможностей.

Понятно, что неразумно, например, составлять список из 10! = 3 628 800 перестановок множества {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}. Но нам может понадобиться, чтобы каждая из них некоторое время пребывала в некоторой структуре данных с тем, чтобы программа могла по очереди поработать со всеми возможными перестановками. Поэтому нашу задачу мы будем формулировать как генерацию всех требующихся нам комбинаторных объектов и о поочередном «посещении» всех объектов: генерации *n*-кортежей, генерации перестановок и генерации сочетаний.

3.1.2 Алгоритмы генерации n - кортежей.

Рассмотрим обход всех 2^n строк, состоящих из n двоичных цифр. У нас есть n кортежей $(a_1, a_2, \ldots, a_n)$, где каждый элемент a_j является либо нулем, либо единицей. Эта задача эквивалентна просмотру всех подмножеств $\{x_1, x_2, \ldots, x_n\}$, поскольку можно сказать, что x_j находится в подмножестве тогда и только тогда, когда $a_j = 1$.

Простейшее решение поставленной задачи заключается в рассмотрении двоичного числа $(00\ldots0)_2 = 0$ и многократного добавления к нему 1, пока не будет достигнуто число $(11\ldots1)_2 = 2^n - 1$. Однако, эта простая задача имеет интересные особенности, которые рассмотрим в дальнейшем.

Двоичная запись может быть расширена и на другие типы n-кортежей. Например, для перечисления всех возможных вариантов кортежа из всех десятичных цифр, можно выполнить перечисление чисел от $(00\ldots0)_{10} = 0$ до $(99\ldots9)_{10} = 10^n - 1$ в десятичной системе счисления.

Для более общего случая обхода всех вариантов, при которых

$$0 \le a_j < m_j, \text{ для } 1 \le j \le n, \tag{3.1}$$

где верхняя граница k_j разных компонентов вектора $(a_1, a_2, \ldots, a_n)$ могут отличаться, задача сводится к многократному прибавлению 1 к числу

$$\begin{bmatrix} a_1, a_2 & \ldots, a_n \\ m_1, m_2 & \ldots, m_n \end{bmatrix} \tag{3.2}$$

в смешанной позиционной системе счисления.

Алгоритм А1 *(Генерация перечислений в смешанной системе счисления)*:

1. [Инициализация.] Установить $a_j \leftarrow 0$ для $0 \le j < n$, и установить $m_0 \leftarrow 2$.

2. [Посещение.] Посетить n-кортеж $(a_1, a_2, \ldots, a_n)$. Программа, которая должна посетить все n кортежей, должна приступить к работе.

3. [Подготовка к прибавлению единицы.] Установить $j \leftarrow n$.

4. [Перенос при необходимости.] Если $a_j = m_j - 1$, установить $a_j \leftarrow 0$, $j \leftarrow j-1$, и повторить этот шаг.

5. [Увеличение, если не выполнено.] Если $j = 0$, завершить работу алгоритма. В противном случае установить $a_j \leftarrow a_j + 1$, и вернуться к шагу 2.

Этот алгоритм посещает все n кортежей, удовлетворяющие условию (3.1), путем прибавления 1 к числу в смешанной позиционной системе счисления (3.2) до тех пор, пока не будет достигнуто переполнение. Введены вспомогательные переменные a_o и m_0. Алгоритм прост и прямолинеен, но при достаточно малых константах n использование вложенных циклов позволяет создать еще более простой алгоритм.

Например, для $n = 4$ можно было бы записать следующие инструкции, эквивалентные предыдущему алгоритму:

Для $a_1 = 0, 1, \ldots, m_1 - 1$ (в указанном порядке) выполнить следующее:

Для $a_2 = 0, 1, \ldots, m_2 - 1$ (в указанном порядке) выполнить следующее:

Для $a_3 = 0, 1, \ldots, m_3 - 1$ (в указанном порядке) выполнить следующее:

Для $a_4 = 0, 1, \ldots, m_4 - 1$ (в указанном порядке) выполнить следующее:

Посетить (a_1, a_2, a_3, a_4).

Алгоритм **A1** проходит по всем вариантам *n*-кортежа в лексикографическом порядке, как в словаре. Однако, бывают задачи, когда посещение *n*-кортежей предпочтительнее выполнить в некотором другом порядке.

Наиболее известным альтернативным упорядочиванием является *бинарный код Грея*, который перечисляет все 2^n строк из *n* бит таким образом, что каждый раз простым и регулярным способом изменяется только один бит. Например, для $n = 4$ код Грея имеет вид: 0000, 0001, 0011, 0010, 0111, 0101, 0100, 1100, 1101, 1111, 1110, 1010, 1011, 1001, 1000. Такие коды особенно важны в приложениях, у которых аналоговая информация преобразуется в цифровую и наоборот.

Существует множество эквивалентных способов определить код Грея. Пусть Γ_n - бинарная последовательность Грея, состоящую из *n*–битных строк. Тогда Γ_n можно определить рекурсивно с помощью следующих двух правил:

$$\Gamma_0 = \epsilon;$$

$$\Gamma_{n+1} = 0\Gamma_n, 1\Gamma_n^R. \qquad (3.3)$$

Здесь ϵ (эпсилон) обозначает пустую строку, $0\Gamma_n$ обозначает последовательность Γ_n с префиксом 0 в каждой строке, а $1\Gamma_n^R$ обозначает последовательность Γ_n в обратном порядке с префиксом 1 в каждой строке. Поскольку последняя строка в Γ_n эквивалентна первой строке в Γ_n^R, из (3.3) ясно, что на каждом шаге Γ_{n+1} изменяется ровно на один бит, если Γ_n обладает тем же свойством.

Другой способ определения последовательности $\Gamma_n = g(0), g(1), \ldots g(2^n - 1)$ заключается в указании явной формулы для отдельных элементов $g(k)$. Поскольку Γ_{n+1} начинается с $0\Gamma_n$, то бесконечная последовательность

$$\Gamma_\infty = g(0), g(1), g(2), \ldots = (0)_2,(1)_2,(11)_2,(10)_2,(110)_2, \ldots \qquad (3.4)$$

является перестановкой всех неотрицательных чисел, если рассматривать каждую строку из нулей и единиц как двоичное число с необязательными ведущими нулями. Тогда Γ_n состоит из первых 2^n элементов последовательности (3.4), преобразованных в *n*-битовые строки за счет вставки при необходимости ведущих нулей.

Когда $k = 2^n + r$, где $0 \le r < 2^n$, то из соотношения (3.3), получаем

$g(k) = 2^n + g(2^n - 1 - r)$.

Следовательно, по индукции по n можно доказать, что целое число k с двоичным представлением $(\ldots b_2 b_1 b_0)_2$ имеет эквивалент в бинарном коде Грея с представлением $(\ldots a_2 a_1 a_0)_2$, где

$$a_j = b_j \oplus b_{j+1} \quad \text{для } j \ge 0. \tag{3.5}$$

Например, $g((111001000011)_2) = (100101100010)_2$

И, наоборот, если дано $g(k) = (\ldots a_2 a_1 a_0)_2$, можно найти $k = (\ldots b_2 b_1 b_0)_2$, обращая систему уравнений (3.5) и получая

$$b_j = a_j \oplus a_{j+1} \oplus a_{j+2} \oplus \ldots \quad \text{для } j \ge 0. \tag{3.6}$$

Эта бесконечная сумма на самом деле конечна, так как $a_{j+t} = 0$ для всех больших t.

Из (3.5) получаем два следствия:

$$g(k) = k \oplus \lfloor k/2 \rfloor. \tag{3.7}$$

и

$$g(k \oplus k') = g(k) \oplus g(k') \tag{3.8}$$

где k, k ' - произвольные неотрицательные целые числа.

Функцию $g(k)$ (3.5) и обратную ей функцию $g^{[-1]}(k)$ (3.6) можно рассматривать как функции, преобразующие бинарные строки в другие бинарные строки; также их можно рассматривать как функции, преобразующие целые числа в целые числа с их промежуточным представлением в виде бинарных строк с игнорируемыми ведущими нулями.

В XIX веке французский судья Луи Грос продемонстрировал явную связь классической головоломки «Китайские кольца» с двоичными числами. На рисунке 3.1 показан пример такой головоломки с семью кольцами. Задача заключается в том, чтобы снять кольца со стержня, но кольца переплетены так, что возможны только два основных действия:

а) Крайнее справа кольцо можно снять или одеть в любой момент.

б) Любое другое кольцо можно снять или одеть тогда и только тогда, когда кольцо слева от него находится на стержне, а все кольца правее сняты.

Текущее состояние головоломки можно представить в двоичной системе счисления, где 1 – означает, что кольцо находится на стержне, а 0 – что кольцо снято.

Если кольца находятся в состоянии $a_{n-1} \ldots a_0$, и если определено двоичное число $k = (b_{n-1} \ldots b_0)_2$, то ровно k шагов необходимо и достаточно для того, чтобы решить головоломку.

Рисунок 10 – Головоломка «Китайские кольца»

На рисунке 10 слева кольца в состоянии 11…1 (все на стержне), а справа в состоянии 11…100 (два левых кольца сняты).

Если кольца находятся в произвольном состоянии, отличном от 00…0 или 10…0, то возможны ровно два действия: первое - типа (а), а второе типа (б). Только одно из этих действий дает продвижение к заданной цели, а любое другое представляет собой шаг назад. Действие типа (а) изменяет k на $k \oplus 1$; его следует выполнять при нечетном k, поскольку тем самым будет выполнено уменьшение k. Действие типа (б) в позиции заканчивается на $(10^{j-1})_2$ для $0 \le j < n$, изменяет k на $k \oplus (1^{j+1})_2 = k \oplus (2^{j+1} - 1)$. В этой формуле «$1^{j+1}$» означает j+1 повторений «1», а «2^{j+1}» означает степень двойки. Если k четно, то нужно приравнять $k \oplus (2^{j+1} - 1)$ к $k - 1$, что означает, что k должно быть кратно 2^j, но не 2^{j+1}. То есть $j = \rho(k)$, где ρ - так называемая *линеечная* функция, для которой выполняются два соотношения: $\rho(k + 1) = 0$ и $\rho(2k) = \rho(k) + 1$. Следовательно, для корректного решения головоломки кольца должны следовать красивому шаблону. Если пронумеровать кольца 0, 1, …,

n-1(начиная от свободного кольца), то последовательность снятия и одевания колец представляет собой последовательность из чисел, которые заканчиваются …, (ρ4(ρ ,)3(ρ ,)2ρ ,)(1).

Идя в обратном направлении, нужно начав с 00…0, последовательно одевать или снимать кольца до тех пор, пока не будет достигнуто конечное состояние 10…0, что приводит нас к алгоритму счета в бинарном коде Грея.

Ниже приведен алгоритм **A2** генерации бинарного кода Грея, посещающий все бинарные кортежи $(a_{n-1}, \ldots, a_1, a_0)$, начиная с (0,…, 0, 0) и изменяя только по одному биту за раз, а также сохраняя бит a_∞ такой, что

$$a_\infty = a_{n-1} \oplus \ldots \oplus a_1 \oplus a_0. \qquad (3.9)$$

Этот бит последовательно дополняет биты $\rho(1)$, $\rho(2)$, …, $\rho(2^n - 1)$, после чего алгоритм останавливается.

Алгоритм A2 *(Генерация бинарного кода Грея)*:

1. [Инициализация.] Установить $a_j \leftarrow 0$ для $0 \le j < n$, и установить $a_\infty \leftarrow 0$.
2. [Посещение.] Посетить n-кортеж $(a_{n-1}, \ldots, a_1, a_0)$.
3. [Изменение четности.] Установить $a_\infty \leftarrow 1 - a_\infty$.
4. [Выбор j.] Если $a_\infty = 1$, установить $j \leftarrow 0$. В противном случае $1 \le j$ – минимальное, при котором $a_{j-1} = 1$. (После k-ого выполнения этого шага $j = \rho(k)$).
5. [Дополнительные координаты j.] Завершить работу алгоритма, если $j = n$; в противном случае установить $a_j \leftarrow 1 - a_j$ и вернуться к шагу 2.

3.1.2 Алгоритмы решения задачи генерации перестановок.

Для решения задачи генерации перестановок разработано много различных способов. Рассмотрим некоторые из них.

Рассмотрим алгоритм **A3**, который, как и алгоритм **A1**, основан на общей процедуре получения объекта, лексикографически следующего за данным объектом $a_1 \ldots a_n$, и состоящей из трех шагов:

1. Найти наибольшее j, такое, что a_j может быть увеличено.
2. Увеличить a_j на наименьшее допустимое значение.

3. Найти лексикографически наименьший способ расширения новой части $a_1 \ldots a_j$ до полного объекта.

Алгоритм **А3** лексикографической генерации перестановок для заданной последовательности из n элементов $a_1\ a_2 \ldots a_n$, изначально отсортированной таким образом, что $a_1 \le\ a_2\ \le \ldots \le a_n$, генерирует все перестановки $\{a_1, a_2, \ldots, a_n\}$, посещая их в лексикографическом порядке.

Например, лексикографически упорядоченными перестановками для множества {1, 2, 3, 4} являются 1234, 1243, 1324, 1342, 1423, 1432, 2134, 2143, 2314, 2341, 2413, 2431,3124, 3142, 3214, 3241, 3412, 3421, 4123, 4132, 4213, 4231, 4312, 4321.

Алгоритм А3 *(Лексикографическая генерация перестановок)*:

1. [Посещение.] Посетить перестановку $a_1\ a_2 \ldots a_n$.

2. [Поиск j.] Установить $j \leftarrow n - 1$. Если $a_j \ge\ a_{j+1}$ то, уменьшать j на 1, пока не будет выполнено условие $a_j < a_{j+1}$. Завершить работу алгоритма, если $j = 0$ (В этот момент представляет собой наименьший индекс, такой, что все перестановки, начиная с $a_1 \ldots a_j$, уже были посещены. Так что лексикографически следующая перестановка увеличивает значение a_j.)

3. [Увеличение a_j.] Установить $l \leftarrow n$. Если $a_j \ge\ a_l$ то уменьшать l на 1, пока не будет выполнено условие $< a_l$. Затем выполнить обмен $a_j \leftrightarrow a_l$. (поскольку $a_{j+1} \ge \ldots \ge\ a_n$, элемент a_l является наименьшим элементом, который больше a_j и который может законным образом следовать в перестановке за $a_1 \ldots\ a_{j-1}$.
Перед обменом мы имели $a_{j+1} \ge \ldots \ge\ a_{l-1} \ge\ a_l > a_j \ge\ a_{l+1} \ldots \ge\ a_n$;
после мы получаем $a_{j+1} \ge \ldots \ge\ a_{l-1} \ge a_j > a_l \ge\ a_{l+1} \ldots \ge\ a_n$.)

4. [Обращение $a_{j+1} \ldots a_n$.] Установить $k \leftarrow j$+1и $l \leftarrow n$. Затем, пока $k < l$, выполнить обмен $a_k \leftrightarrow a_l$ и установки $k \leftarrow k + 1$, $l \leftarrow l$–1 . Вернуться к шагу 1.

Рассмотрим еще один алгоритм генерации перестановок. Ранее мы с вами убедились, что коды Грея позволяют эффективно генерировать n – кортежи. Такие же рассуждения применимы и к генерации перестановок.

Алгоритм **A4** основан на том факте, что простейшим изменением в перестановке является обмен местами только двух соседних элементов. Однако возникает вопрос о возможности обойти все перестановки мультимножества таким методом. К сожалению, в случае с мультимножеством такой обход не всегда возможен. Например, шесть перестановок мультимножества {1, 1, 2, 2} связаны одна с другой смежными обменами следующим образом:

$$1122 - 1212 < \begin{matrix} 2112 \\ 1221 \end{matrix} > 2121 - 2211.$$

У этого графа нет гамильтонова пути.

Однако существует простой алгоритм, который позволяет сгенерировать все $n!$ перестановок с помощью $n! - 1$ смежных обменов. Еще один смежный обмен возвращает нас в начальное состояние, так что мы получаем гамильтонов цикл, аналогичный бинарному коду Грея.

Идея алгоритма заключается в том, чтобы взять такую последовательность для $\{1,\ldots, n-1\}$ и вставить число в каждую из перестановок всеми возможными способами. Например, если n =3, то последовательность (12, 21) при вставке 3 во все три возможные позиции приводит к следующим столбцам массива:

123	213
132	231
312	321

Теперь мы получаем искомую последовательность, читая первый столбец сверху вниз, второй – снизу вверх: (123, 132, 312, 321, 231, 213).

Итак, алгоритм, **A4** для заданной последовательности $a_1 \ldots a_n$ из n различных элементов, генерирует все их перестановки путем многократных обменов смежных элементов. Алгоритм использует вспомогательный массив, который представляет собой инверсии $c_1 \ldots c_n$, перестановки и проходит по всем последовательностям целых чисел таких, что

$$0 \le c_j < j \text{ для } 1 \le j < n. \qquad (3.10)$$

Массив $o_1 \ldots o_n$, указывает направление изменений элементов c_j.

Алгоритм A4 *(Генерация перестановок методом простых изменений)*:

1. [Инициализация.] Установить $c_j \leftarrow 0$ и $o_j \leftarrow 0$ для $1 \le j < n$.

2. [Посещение.] Посетить перестановку $a_1 \ldots a_n$.

3. [Подготовка к изменению.] Установить $j \leftarrow n$ и $s \leftarrow 0$. (Следующие шаги определяют координату j, для которой будет изменяться c_j, сохраняя условие 3.10; переменная s представляет собой количество индексов $k > j$ таких, что $c_k = k - 1$.)

4. [Проверяем готовность к изменению] Установить $q \leftarrow c_j - o_j$. Если $q < 0$, то перейти к шагу 7; если $q = j$, то перейти к шагу 6.

5. [Изменение.] Обменять $a_{j-c_j+s} \leftrightarrow a_{j-q+s}$ и установки $c_j \leftarrow q$. Вернуться к шагу 2.

6. [Увеличение s.] Завершить работу алгоритма, если $j = 1$; в противном случае установить $s \leftarrow s + 1$.

7. [Изменение направления.] Установить $o_j \leftarrow -o_j$, $j \leftarrow j - 1$ и вернуться к шагу 4.

Это процедура простых изменений работает для всех $n \ge 1$, и появилась в Англии XVII века, когда звонари придумали обычай перезвона множества колоколов во всех возможных перестановках.

3.1.3 Алгоритмы решения задачи генерации сочетаний.

Как мы знаем, сочетанием из n элементов по t представляет собой способ выбора подмножества размером t из данного множества размером n.

Существует ровно $\binom{n}{t} = \frac{n(n-1)\ldots(n-t+1)}{t(t-1)\ldots(1)}$ способов сделать это.

Мы знаем так же, что выбор t из n объектов эквивалентен выбору $n - t$ оставшихся элементов, то есть можно написать, что

$$n = t + s. \tag{3.11}$$

и говоря о t-сочетании n элементов как о (s,t) – сочетании.

Имеется два основных способа представления (s,t) – сочетаний: можно перечислить выбранные элементы $c_t \ldots c_2 c_1$, а можно работать с бинарными строками для которых

$$a_{n-1} + \ldots + a_1 + a_0 = t. \tag{3.12}$$

Строковое представление содержит s нулей и t единиц, соответствующих невыбранным и выбранным элементам. Представление в виде списка $c_t \ldots c_2 c_1$ удобно, когда элементы являются членами множества $\{0, 1, \ldots, n-1\}$, и мы перечисляем их в убывающем порядке:

$$n > c_t > \ldots > c_2 > c_1 \geq 0. \tag{3.13}$$

Бинарная запись отлично объединяет эти два представления, поскольку список элементов соответствует сумме

$$2^{c_t} + \ldots + 2^{c_2} + 2^{c_1} = \sum_{k=0}^{n-1} a_k 2^k = (a_{n-1} \ldots a_1 a_0)_2. \tag{3.14}$$

Позиции нулей $b_s \ldots b_2 b_1$ в строке $a_{n-1} \ldots a_1 a_0$ можно перечислить следующим образом

$$n > b_s > \ldots > b_2 > b_1 \geq 0. \tag{3.15}$$

Сочетания важные не только сами по себе, но и как объект, являющийся эквивалентным многим другим конфигурациям. Например, каждое (s,t) – сочетание соответствует сочетанию $s + 1$ объекта по t с повторениями, называемому также *мультисочетанием* $s + 1$ элемента, а именно последовательности целых чисел $d_t \ldots d_2 d_1$, обладающей тем свойством, что

$$s > d_t > \ldots > d_2 > d_1 \geq 0. \tag{3.16}$$

Одна из причин заключается в том, что $d_t \ldots d_2 d_1$ удовлетворяет (3.16), тогда и только тогда, когда $c_t \ldots c_2 c_1$ удовлетворяет (3.13), где

$$c_t = d_t + t - 1, \ldots, c_2 = d_2 + 1, c_1 = d_1 . \tag{3.17}$$

Так же есть способ связать сочетания с повторениями и обычные сочетания (способ Соломона Коломба), а именно определить

$$e_j = \begin{cases} c_j, & \text{если } c_j \leq s; \\ e_{c_j - s}, & \text{если } c_j > s. \end{cases} \tag{3.18}$$

В этом виде числа $e_t \ldots e_2 e_1$ не обязательно находятся в убывающем порядке, но мультимножество $\{a_1, \ldots, a_t\}$ эквивалентно $\{c_1, \ldots, c_t\}$ тогда и только тогда, когда $\{e_1, \ldots, e_t\}$ является множеством.

(s,t) – сочетание также эквивалентно композиции (или разбиению) $n+1$ из $t+1$ частей, а именно - упорядоченной сумме

$$n + 1 = p_t + \ldots + p_1 + p_0, \text{ где } p_t, \ldots, p_1, p_0 \geq 1. \qquad (3.19)$$

В этом случае с соотношением (3.13) имеется связь

$$p_t = n - c_t,\ p_{t-1} = c_t - c_{t-1}, \ldots, p_1 = c_2 - c_1, p_0 = c_1 + 1.$$

Аналогично, если $q_j = p_j - 1$, то мы получаем композицию s из $t + 1$ неотрицательных частей $s = q_t + \ldots + q_1 + q_0$, где $q_t, \ldots, q_1, q_0 \geq 0$, которая связана с (3.16) соотношением $q_t = s - d_t,\ q_{t-1} = d_t - d_{t-1}, \ldots, q_1 = d_2 - d_1, q_0 = d_1$.

Кроме того (s,t) – сочетание эквивалентно пути длиной $s + t$ из угла в укол в сетке $s \times t$, потому что такой путь содержит s вертикальных шагов и t горизонтальных.

Рассмотрим, например, «случайную» битовую строку

$$a_{10} \ldots a_1 a_0 = 01010011001, \qquad (3.20)$$

в которой содержится $s = 6$ нулей и $t = 5$ единиц, так что $n = 11$. Дуальное сочетание $b_s \ldots b_2 b_1$ перечисляет позиции нулей, а именно

10 8 6 5 2 1,

поскольку крайняя слева позиция имеет номер $n - 1$, а крайняя справа – 0. Основное сочетание $c_t \ldots c_2 c_1$ перечисляет количество единиц, а именно

9 7 4 3 0.

Соответствующее мультисочетание $d_t \ldots d_2 d_1$ указывает количество нулей справа от каждой единицы:

5 4 3 3 0.

Сочетание $p_t \ldots p_1 p_0$ перечисляет расстояния между последовательными единицами, если представить наличие фиктивных единиц слева и справа от бинарной строки:

2 2 3 1 3 1.

И наконец неотрицательное сочетание $q_t \ldots q_1 q_0$ подсчитывает, сколько нулей находится между «ограждениями»-единицами:

1 1 2 0 2 0.

Итак, мы имеем $a_{n-1} \ldots a_1 a_0 = 0^{q_t} 10^{q_{t-1}} 1 \ldots 10^{q_1} 10^{q_0}$.

При небольших t с задачей генерации сочетаний легко могут справиться и циклы.

Для $t = 3$ достаточно выполнения ниже следующих инструкций:

Для $c_3 = 2, 3, \ldots, n-1$ (в указанном порядке) выполнить:

Для $c_2 = 1, 2, \ldots, c_3 - 1$ (в указанном порядке) выполнить:

Для $c_1 = 0, 1, \ldots, c_2 - 1$ (в указанном порядке) выполнить:

Посетить сочетание $c_3 c_2 c_1$.

С другой стороны, когда t - переменное или не столь малое значение, то можно *лексикографически* генерировать сочетания.

Лексикографический порядок обычно предлагает наиболее удобный способ генерации комбинаторных конфигураций. Рассмотрим такой алгоритм **A5** генерирующий сочетания в форме $a_{n-1}, \ldots, a_1, a_0$, опираясь на то, что (s,t) – сочетания в форме битовой строки представляют собой не что иное, как перестановки мультимножества $\{s \cdot 0, t \cdot 1\}$.

В нашем алгоритме генерируются все t-сочетания $c_t \ldots c_2 c_1$ из n чисел $\{0, 1, \ldots, n-1\}$ для данных $n \geq t \geq 0$. В качестве ограничителей используется дополнительные переменные c_{t+1} и c_{t+2}. Будем искать крайний справа элемент c_j, который может быть увеличен, после чего последующие элементы $c_{j-1}, \ldots c_1$ получают минимально возможные значения.

Алгоритм A5 *(Генерация сочетаний в лексикографическом порядке)*:

1. [Инициализация.] Установить $c_j \leftarrow j - 1$ для $1 \leq j \leq t$, и установить $c_{t+1} \leftarrow n$ и $c_{t+2} \leftarrow 0$.

2. [Посещение.] Посетить сочетание $c_t \ldots c_2 c_1$.

3. [Поиск j.] Установить $j \leftarrow 1$. Затем, пока $c_j + 1 = c_{j+1}$, установить $c_j \leftarrow j - 1$ и $j \leftarrow j + 1$. Это действие повторяем до тех пор, пока не выполнится условие $c_j + 1 \neq c_{j+1}$.

4. [Условие завершения.] Завершить выполнение алгоритма, если $j > t$.

5. [Увеличение c_j.] Установить $c_j \leftarrow c_j + 1$ и вернуться к шагу 2.

Проведем оценку времени работы этого алгоритма. На 3 шаге устанавливается $c_j \leftarrow j - 1$ сразу после посещения сочетания, для которого $c_1 + j = c_{j+1}$, а количество таких сочетаний равно количеству решений неравенств

$$n > c_t > \ldots > c_j > c_{j+1} \geq j;$$

однако эта формула представляет собой эквивалент $(t - j)$-сочетания из $n - j$ объектов $\{ n - 1, \ldots, j\}$, так что присваивание $c_j \leftarrow j - 1$выполняется ровно $\binom{n-j}{t-j}$ раз. Суммирование для $1 \leq j \leq t$ показывает, что цикл на 3 шаге выполняется $\binom{n-1}{t-1} + \binom{n-2}{t-2} + \ldots + \binom{n-t}{0} = \binom{n-1}{s} + \binom{n-2}{s} + \ldots + \binom{s}{s} = \binom{n}{s+1}$ раз или в среднем $\binom{n}{s+1} \Big/ \binom{n}{t} = \frac{n!}{(s+1)!(t-1)!} \Big/ \frac{n!}{s!t!} = \frac{t}{s+1}$ выполнений за одно посещение.

Это отношение меньше 1 при $t \leq s$, поэтому алгоритм достаточно эффективен для таких случаев.

Однако для случаев, когда t близко по своему значению к n, а s мало, величина $\frac{t}{s+1}$ может оказаться весьма большой.

Действительно, иногда алгоритм **A5** выполняет присваивание $c_j \leftarrow j - 1$ без необходимости, когда c_j и так равно $j - 1$. Также на шагах 4 и 5 индекс j не всегда нужно искать, так как его корректное значение можно предсказать на основе только что выполненных действий. Например, после того, как мы увеличили c_4 и сбросили $c_3c_2c_1$ в начальное значение 210, очередное сочетание неизбежно приведет к увеличению c_3. Исходя из этих наблюдений можно построить более эффективный алгоритм **A6** генерации сочетаний в лексикографическом порядке.

Алгоритм А6 *(Улучшенный алгоритм генерации сочетаний в лексикографическом порядке)*:

1. [Инициализация.] Установить $c_j \leftarrow j - 1$ для $1 \le j \le t$, и установить $c_{t+1} \leftarrow n$, $c_{t+2} \leftarrow 0$ и $j \leftarrow t$.

2. [Посещение.] (В этот момент наименьший индекс такой, что $c_{j+1} > j$.) Посетить сочетание $c_t \ldots c_2 c_1$. Затем, если $j > 0$, установить $x \leftarrow j$ и перейти к шагу 6.

3. [Проверка, является ли случай простым.] Если $c_1 + 1 < c_2$, установить $c_1 \leftarrow c_1 + 1$ и вернуться к шагу 2. В противном случае установить $j \leftarrow 2$.

4. [Поиск j.] Установить $c_{j-1} \leftarrow j - 2$ и $x \leftarrow c_j + 1$. Если $x = c_{j+1}$, установить $j \leftarrow j + 1$ и повторить шаг 4.

5. [Условие завершения.] Завершить выполнение алгоритма, если $j > t$.

6. [Увеличение c_j.] Установить $c_j \leftarrow c_j + 1$, $j \leftarrow j - 1$ и вернуться к шагу 2.

Этот алгоритм работает быстрее, чем **А5**. В алгоритме **А6** предполагается, что $0 < t < n$. На втором шаге $j = 0$ тогда, когда $c_1 > 0$, так что присваивания на шаге 4 никогда не будут избыточными. Мы добираемся до шага 3, когда $c_1 > 0$; до шага 5, когда $c_2 = c_1 + 1 > 1$; до шага 4 для тех $2 \le j \le t+1$, для которых $c_j = c_1 + j - 1 \ge j$. Таким образом, соответствующие величины равны: шаг 1 выполняется за 1, шаг 2 выполняется за $\binom{n}{t}$; шаг 3 за $\binom{n-1}{t}$; шаг 4 выполняется за $\binom{n-2}{t-1}+\binom{n-3}{t-2}+\ldots+\binom{n-t-1}{t-1} = \binom{n-1}{t-1}$; шаг 5 выполняется за $\binom{n-2}{t-2}$; шаг 6 выполняется за $\binom{n-1}{t-1} + \binom{n-2}{t-1} - 1$.

3.1.4 Задача генерации разбиений целого числа.

Слово «разбиение» в математике имеет множество значений. Разбиения множества представляют собой способы разделения его на непересекающиеся подмножества. Например, множество {0, 1, 2} имеет пять разбиений: {0, 1, 2}, {0}{1, 2}, {0, 1} {2}, {0, 2}{1}, {0} {1} {2}.

Разбиения целого числа представляют способы его записи в виде суммы натуральных чисел, независимых от их порядка. Например, число три имеет три разбиения: 3, 2+1, 1+1+1.

Рассмотрим задачу генерации разбиений целого числа и под разбиением далее будем понимать именно разбиение целого числа. Разбиение целого числа n можно представить как последовательность неотрицательных целых числе $a_1 \geq a_2 \ldots$, такую, что $n = a_1 + a_2 \ldots$.

Простейшим и быстрейшим способом генерации всех разбиений является посещение разбиений в обратной лексикографическом порядке, начиная с 'n' и заканчивая '11…1'. Например, разбиение 5 в указанном порядке представляет собой 5, 41, 32, 311, 221, 2111, 11111. Если разбиение состоит не из одних единиц, оно завершается некоторым числом (x+1), где $x \geq 1$, за которым следует нуль или несколько единиц. Следовательно, очередное наименьшее разбиение в лексикографическом порядке получается путем замены суффикса (x+1)1…1 на $x \ldots xr$ для некоторого соответствующего остатка $x \geq r$. Процесс достаточно эффективен, если отслеживать наибольший индекс q, такой что $a_q \neq 1$, как предложено Д.К.С. Мак-Кеем и заполнить массив единицами, как предложено А.Зогби [4].

Алгоритм А7 *(Разбиения n в обратном лексикографическом порядке)*:

1. [Инициализация.] Установить $a_m \leftarrow 1$ для $1 < m \leq n$, и установить $m \leftarrow 1$ и $a_0 \leftarrow 0$.

2. [Сохранение заключительной части.] Установить $a_m \leftarrow n$ и $q \leftarrow m - [n=1]$.

3. [Посещение.] Посетить разбиение $a_1\, a_2, \ldots, a_m$. Затем, если $a_q \neq 2$, перейти к шагу 5.

4. [Замена 2 на 1+1.] Установить $a_q \leftarrow 1$, $q \leftarrow q - 1$, $m \leftarrow m + 1$ и вернуться к шагу 3. У нас будет $a_q = 1$ для $q < k \leq n$.

5. [Уменьшение a_q.] Завершить работу, если $q = 0$. В противном случае установить $x \leftarrow a_q - 1$, $a_q \leftarrow x$, $n \leftarrow m - q + 1$ и $m \leftarrow q + 1$.

6. [Копирование x при необходимости.] Если $n \le x$, вернуться к шагу 2. В противном случае установить $a_m \leftarrow x$, $m \leftarrow m + 1$, $n \leftarrow n - x$ и повторить данный шаг.

Алгоритм **A7** генерирует все разбиения $a_1 \ge a_2 \ldots \ge a_m \ge 1$ с $n = a_1 + a_2 + \ldots + a_m$ и $1 < m \le n$ для $1 \le n$. Значение a_0 устанавливается равным нулю. Переход от одного разбиения к следующему выполняется особенно просто, если имеется двойка. В этом случае шаг 4 просто заменяет крайнюю справа двойку единицей и добавляет единицу справа. Для $n = 100$ двойку содержат около 79% разбиений.

3.2 Задание на выполнение лабораторной работы.

Цель работы: разработка и анализ простейших алгоритмов над комбинаторными объектами.

Варианты задания на выполнение работы №3.

Разработайте алгоритм, проведите его анализ и реализуйте на языке Pascal или Python:

Задание №1 (по вариантам).

1. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) четырех элементов из множества, состоящего из шести гласных букв русского алфавита.
2. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) трёх элементов из множества цифр восьмеричной системы счисления.
3. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) трёх элементов из множества цифр восьмеричной системы счисления.
4. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) трёх элементов из множества звонких согласных букв русского алфавита.

5. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) трёх элементов из множества гласных букв английского алфавита.
6. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) трёх элементов из множества гласных букв английского алфавита.
7. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трёх элементов из множества цифр десятичной системы счисления.
8. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трех элементов из множества 5 массивом действительных чисел размерности 3x3, сумма элементов которых больше 25.
9. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трех элементов из множества предметов, изучаемых в этом семестре, по которым планируется экзамен.
10. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трех элементов из множества 5 массивом действительных чисел размерности 3x3, сумма элементов которых больше 25.
11. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) четырех элементов из множества прилагательных, обозначающих цвета в радуге (красный, оранжевый и т.д.).
12. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трех элементов из множества цифр шестнадцатеричной системы счислений (1, 7, A, B, C, D), сумма цифр которых на позициях больше 20.
13.Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трёх элементов из множества цифр десятичной системы счисления.
14. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трех элементов из множества цифр восьмеричной системы, сумма цифр которых на позициях равна 15.
15. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трёх элементов из множества, содержащего все месяцы года, с

количеством дней не менее 31.

16. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) трёх элементов из множества шести произвольных цифр десятичной системы счисления, для которых сумма на всех трех позициях меньше семи.

17. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) трёх элементов из множества цифр шестнадцатеричной системы счисления.

18. Перечислите и рассчитайте количество всех размещений без повторений (и сочетаний) четырех элементов из множества первых шести месяцев (январь, февраль, …, июнь).

19. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) трёх элементов из множества цифр пятеричной системы счисления (0, 2, 3,4 ,1), для которых сумма на позициях меньше семи.

20. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) четырех элементов из множества согласных букв английского алфавита.

21. Перечислите и рассчитайте количество всех размещений с повторениями (и сочетаний) трех элементов из множества шести символов, которые содержат не менее двух одинаковых элементов.

Задание № 2 (Для всех вариантов).

Дано n ($n > 2$) произвольных цифр: a_1, a_2, a_3, …, a_n, где $a_i \in \{1, 2, ..., 9\}$, и произвольное целое число m. Написать программу, которая расставляла бы между каждой парой цифр: a_1, a_2, a_3, …, a_n, записанных именно в таком порядке, знаки «+», «-» так, чтобы значением получившегося выражения было число m.

3.3 Контрольные вопросы.

1. Перечислите основные комбинаторные объекты.
2. Перечислите известные вам методы порождения перестановок.

3. Назовите известные вам способы генерации кодов Грея Γ_n?
4. Какой эквивалентной задачей можно заменить задачу порождения всех подмножеств данного множества.
5. Что называется транспозицией двух элементов в перестановке?
6. Какие способы порождения всех двоичных наборов длины *n* вы знаете?
7. Сформулируйте задачу эквивалентную задаче генерации всех размещений с повторениями.
8. Поясните, рассмотрение разбиения как мультимножества. Приведите пример.
9. Перечислите известные вам алгоритмы генерации комбинаторных объектов в прямом лексикографическом порядке.
10. Сравните вычислительную сложность алгоритмов А5 и А6 генерации сочетаний в лексикографическом порядке.
11. Каким образом достигается эффективность алгоритма **А7** разбиения целого положительного числа *n* в обратном лексикографическом порядке?

4. Лабораторная работа № 4

4.1. Комбинаторные алгоритмы теории информации.

Теория информации - часть *кибернетики* - науки, изучающей общие законы получения, передачи и хранения информации. Основным предметом кибернетики являются так называемые *кибернетические системы*. **Кибернетические системы** – множество взаимосвязанных объектов – элементов системы, а также связей между ними, обеспечивающих в своей совокупности восприятие, запоминание, переработку и обмен информацией. Из определения следует, что как кибернетические системы могут рассматриваться и системы автоматического управления на предприятиях, и биологические системы, и различные социальные институты.

Информация - это совокупность сведений, подлежащих хранению, передаче, обработке и использованию в человеческой деятельности. Норберт Винер так охарактеризовал информацию: «Движение и действие больших масс или передача и преобразование больших количеств энергии направляется и контролируется при помощи небольших количеств энергии, осуществляющих управление - несущих информацию».

В любой системе информация представлена в виде **сообщений** — совокупности знаков, либо непрерывных сигналов, являющихся переносчиком информации. **Дискретные сообщения** формируются в результате последовательной выдачи источником сообщений отдельных элементов - **знаков.** При этом все множество возможных различных знаков называют **алфавитом** сообщения, а размер множества - **объемом алфавита**.

Ральф Хартли в 1928 определил **количество информации** в виде следующей функции:

$$I = \log_2 N = \log_2 mn = n \log_2 m.$$

Случайным событием называется событие, которое при осуществлении некоторых условий может произойти или не произойти. Пример случайного события - попадание или непопадание мяча в баскетбольную корзину. **Достоверным** называется событие, если в результате испытания оно обязательно происходит. **Невозможное** - такое событие, которое не может произойти в результате данного испытания. **Несовместными** называются такие случайные события, для которых одновременное появление никаких двух из них в рамках данного испытания невозможно. **Независимыми** являются такие события, появление одного которых не меняют вероятности появления других. **Зависимыми** называются такие события, вероятность которых меняется в зависимости от появления других событий, входящих в эту группу. **Полная группа** - множество событий, из которых в результате данного испытания обязательно появится произвольное, но при этом только одно событие. **Исходом** называются события, входящие в полную группу равновозможных несовместных случайных событий. **Исход** называется **благоприят-**

ствующим появлению события *A*, если появление этого исхода влечет за собой появление события *A*.

Вероятностью события А называют отношение числа *m* благоприятствующих этому событию исходов к общему числу *n* всех равновозможных несовместных элементарных исходов, образующих полную группу:

$$P(A) = \frac{m}{n}.$$

Вероятность в классическом определении обладает следующими элементарными свойствами:

- вероятность достоверного события равна 1;

- вероятность невозможного события равна 0;

- вероятность случайного события *A* удовлетворяет двойному неравенству $0 \le P(A) \le 1$.

Событие *A* называется **частным случаем события В**, если при наступлении А наступает и В ($A \subset B$). События *A* и *B* называются **равными**, если каждое из них является частным случаем другого (А = В). **Суммой событий А и В** называется событие $A + B$, наступающее тогда и только тогда, когда наступает хотя бы одно из событий А или В.

Теорема о сложении несовместных событий: вероятность появления одного из двух (одного из группы) несовместных событий равна сумме вероятностей этих событий:

$$\mathrm{P}(A + B) = \mathrm{P}(A) + \mathrm{P}(B),$$

$$P\left(\sum_{i=1}^{n} A_i\right) = \sum_{i=1}^{n} P(A_i). \qquad (4.1)$$

Если события образуют полную группу несовместных событий, то имеет место равенство:

$$P(A_1) + P(A_2) + \ldots + P(A_n) = 1. \qquad (4.2)$$

Теорема о сложении совместных событий: вероятность суммы совместных событий вычисляется по формуле:

$$P(A + B) = P(A) + P(B) - P(AB). \qquad (4.3)$$

Теорема об умножении вероятностей независимых событий: вероятность произведения независимых событий A и B равна:

$$P(AB) = P(A) \cdot P(B). \quad (4.4)$$

Если при вычислении вероятности события никаких других условий, кроме условий эксперимента, не налагается, то такая вероятность называется *безусловной*; если же налагаются и другие дополнительные условия, то вероятность события называют *условной*.

Условной вероятностью $P_A(B) = P(B|A)$ называют вероятность события B, вычисленную в предположении, что событие A уже наступило. Вероятность совместного появления двух зависимых событий равна произведению вероятности одного из них на условную вероятность второго, вычисленную при условии, что первое событие произошло:

$$P(AB) = P(B) \cdot P(A|B) = P(A) \cdot P(B|A) \Rightarrow P(B|A) = \frac{P(AB)}{P(B)}. \quad (4.5)$$

Если событие A может произойти только при выполнении одного из событий $B_1, B_2, \ldots, B_n$, образующих полную группу несовместных событий, то вероятность события A вычисляется по так называемой формуле полной вероятности :

$$P(A) = P(B_1) \cdot P(A|B_1) + P(B_2) \cdot P(A|B_2) + \ldots + P(B_n) \cdot P(A|B_n). \quad (4.6)$$

Рассмотрим полную группу несовместимых событий $B_1, B_2, \ldots, B_n$, вероятности появления которых $P(B_1), P(B_2), \ldots, P(B_n)$. Событие A может произойти только вместе с каким-либо из событий $B_1, B_2, \ldots, B_n$, которые будем называть *гипотезами*. По теореме умножения вероятностей:

$$P(AB_1) = P(B_1)\ P(A|B_1)\ = P(A) \cdot P(B_1|A)\ \Rightarrow\ P(B_1|A)\ = \frac{P(B_1) \cdot P(A|B_1))}{P(A)}. \quad (4.7)$$

Формулой Байеса называется аналогичное выражение, обобщенное для всех гипотез:

$$P(B_i|A)\ = \frac{P(B_i) \cdot P(A|B_i))}{P(A)}, \text{ где } i = 1, \ldots, n. \quad (4.8)$$

На сегодняшний день Байесовы методы и Байесов анализ широко применяется в теории информации, при построении интеллектуальных семантических фильтров (например, спам–фильтров), в алгоритмах анализа массивов данных, в предсказании состояний различных динамических процессов, а также во множестве алгоритмов цифрового приема и обработки сигналов.

Рассмотрим некоторые комбинаторные задачи теории информации.

Задача 1. Набирая номер телефона, абонент забыл последние 4 цифры и помня лишь, что эти цифры различны, набрал их наугад. Найти вероятность того, что номер телефона набран правильно.

Решение. Благоприятствующий исход здесь один - правильный выбор последних цифр (m = 1). Всех возможных исходов n здесь столько же, сколько существует комбинаций из 4-х цифр, порядок которых имеет значение. Таким образом, $n = A_{10}^4 = 3628800:24 = 151200$. Вероятность целевого события, то есть того, что номер будет набран правильно будет $P(A) = \frac{m}{n} = \frac{1}{151200}$.

Задача 2. Имеются марки достоинством в n_1, n_2, …, n_k рублей (все числа n_i различны, а запас марок неограничен). Сколькими способами, наклеивая марки в ряд, можно оплатить с их помощью сумму в N рублей, если два способа, отличающиеся порядком следования различных марок, считаются разными? При этом $f(N)=0$, если $N < 0$.

Решение будем искать в виде

$f(N) = f(N - n_1) + \ldots + f(N - n_k)$. Если $N < 0$, то $f(N)=0$ и $f(0)= 1$.

Для $N = 18$ рублей, и для $n_1= 4$, $n_2 = 6$ и $n_3 =10$ рублей

Получаем

$f(18) = f(14) + f(12) + f(8)$.

$f(14) = f(10) + f(8) + f(4)$

$f(12) = f(8) + f(6) + f(2)$

F(8) = F(4) + F(2) + F(-2)

Разбиения $f(14) = 3 + 1 + 1 = 5$

$f(12) = 1 + 1 + 0 = 2$ $f(8) = 1$

Итак, сумму 18 можно получить 8 способами.

Задача 3. Сообщение передается с помощью сигналов нескольких типов. Длительность передачи сигнала первого типа равна t_1 второго типа – t_2, …k - го типа - t_k единиц времени. Сколько различных сообщений можно передать с помощью этих сигналов за T единиц времени? При этом учитываются лишь "максимальные" сообщения, то есть сообщения, к которым нельзя присоединить ни одного сигнала, не выйдя за рамки отведенного для передачи времени. Решение. Обозначим число сообщений, которые можно передать за время T через $f(T)$. Рассуждая точно так же, как и в задаче о марках, получаем, что f(T) удовлетворяет соотношению

$$f(T) = f(T - t_1) + \ldots + f(T - t_k)\,. \qquad (4.9)$$

При этом снова $f(T) = 0$ если $T < 0$ и $f(0) = 1$.

Такой способ решения задач, при котором сразу не дается окончательной формулы ответа, а указывается лишь процесс, позволяющий сводить задачу ко всё меньшим и меньшим числовым данным, встречается в комбинаторике очень часто. Равенства вида (4.9) называют **рекуррентными формулами.**

Задача 4. В группе 20 студентов, среди которых 4 отличника, 6 хорошистов, 7 троечников, остальные двоечники. По списку наудачу отбираются 5 студентов. Какое количество информации содержится в сообщении о том, что среди отобранных студентов 3 отличника, 1 хорошист и 1 троечник?

Решение. Используем понятие т.н. *обобщенной гипергеометрической вероятности*, используемое, когда интересующие объекты не возвращаются в выборку. Общее количество исходов определяется, как $C_n^m = C_{20}^5$. В то же время, число благоприятных исходов по комбинаторному правилу произведения определяется, как $C_{n1}^{m1}\, C_{n2}^{m2}\, C_{n3}^{m3}\, C_{n3}^{m3} = C_4^3\, C_6^1\, C_7^1\, C_3^0$.

Итоговое значение вероятности совпадает с формулой гипергеометрического распределения и равно $P = \dfrac{C_4^3\, C_6^1\, C_7^1\, C_3^0}{C_{20}^5} = 0{,}011$.

Искомое количество информации: $I = -\log_2 P = -\log_2 0{,}011 = 6{,}528$ bit.

4.2 Задание на выполнение лабораторной работы.

Цель работы: разработка и анализ простейших комбинаторных алгоритмов теории информации.

Варианты задания на выполнение работы №4.

Разработайте алгоритм, проведите его анализ и реализуйте на языке Pascal или Python.

Задание №1 (по вариантам).

1. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) четырех элементов из множества, состоящего из шести гласных букв русского алфавита, содержащие только одинаковые буквы.
2. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества цифр восьмеричной системы счисления, содержащие только одинаковые цифры.
3. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества цифр восьмеричной системы счисления, содержащие на позициях различные цифры.
4. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества звонких согласных букв русского алфавита.
5. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества гласных букв английского алфавита, содержащие на позициях различные элементы.
6. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества гласных букв английского алфавита, содержащие на позициях различные элементы.

7. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества цифр десятичной системы счисления, содержащие только одинаковые цифры.

8. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трех элементов из множества 5 массивом действительных чисел размерности 3x3, сумма элементов которых больше 25.

9. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трех элементов из множества предметов, изучаемых в этом семестре, по которым планируется экзамен.

10. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трех элементов из множества 5 массивов действительных чисел размерности 3x3, сумма элементов которых больше 25.

11. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) четырех элементов из множества прилагательных, обозначающих цвета в радуге (красный, оранжевый и т.д.), содержащие только одинаковые цвета.

12. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трех элементов из множества цифр шестнадцатеричной системы счислений (1, 7, A, B, C, D), сумма цифр которых на позициях больше 20.

13. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества цифр десятичной системы счисления, содержащие только одинаковые цифры.

14. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трех элементов из множества цифр восьмеричной системы, сумма цифр которых на позициях равна 15.

15. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества, содержащего все месяцы года, у которых количество дней не менее 31.

16. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества шести произвольных цифр десятичной системы счисления, для которых сумма на всех трех позициях меньше семи.

17. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества цифр шестнадцатеричной системы счисления, содержащие только одинаковые цифры.

18. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) четырех элементов из множества первых шести месяцев (январь, февраль, …, июнь), которые содержат только одинаковые элементы.

19. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трёх элементов из множества цифр пятеричной системы счисления (0, 2, 3,4 ,1), для которых сумма на позициях меньше семи, содержащие только одинаковые цифры.

20. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) четырех элементов из множества согласных букв английского алфавита, содержащие только одинаковые буквы.

21. Какое количество информации содержится в сообщении, что у нас выписаны все размещения с повторениями (и сочетания) трех элементов из множества шести символов, которые содержат не менее двух одинаковых элементов.

Задание № 2 (по вариантам).

№ 1, 2, 3, 4, 5	Сколькими способами можно разменять n долларов, если имеются монеты по 50, 25, 10, 5 и 1 цент. Указание: запишите решение в виде рекуррентного соотношения и используйте рекурсивный алгоритм вычисления.
№ 6, 7, 8, 9, 10	Сообщение передается с помощью сигналов нескольких типов. Длительность передачи сигнала первого типа равна t1 второго типа – t_2, …k -го типа – t_k единиц времени. Сколько различных сообщений можно передать с помощью этих сигналов за T единиц времени? Для составления алгоритма используйте рекуррентное соотношение $f(T) = f(T - t_1) + \ldots + f(T - t_k)$. Составьте и решите задачу для $k = 3$.
№ 11, 12, 13, 14, 15	Сообщение передается с помощью сигналов нескольких типов. Длительность передачи сигнала первого типа равна t1 второго типа – t_2, …k -го типа – t_k единиц времени. Сколько различных сообщений можно передать с помощью этих сигналов за T единиц времени? Для составления алгоритма используйте рекуррентное соотношение $f(T) = f(T - t_1) + \ldots + f(T - t_k)$. Составьте и решите задачу для $k = 4$.
№ 16, 17, 18, 19, 20, 21	За пересылку бандероли надо уплатить 16 рублей, наклеивая на нее марки. На почте есть по одному виду марок достоинством в 4, 6 и 10 рублей (зато в неограниченном количестве). Сколькими способами можно оплатить пересылку бандероли, если два способа, отличающиеся номиналом или порядком наклеивания марок, считаются различными (марки наклеиваются в один ряд)?

4.3 Контрольные вопросы.

1. Дайте определение теории информации?
2. Какие комбинаторные объекты используются при моделировании задач теории информации?
3. Дайте определение количеству информации по Хартли?
4. Верно ли, что всякая композиция является разбиением?
5. Сформулируйте теорему о сложении вероятностей несовместных событий.
6. Что такое рекуррентное соотношение? Приведите пример.
7. Назовите теорему (теоремы), применяемые для вычисления количества информации, содержащемся в сообщении, при условии, что осуществляется выборка из какой-либо основной комбинаторной комбинации.
8. Назовите известные вам задачи теории информации, являющиеся задачами на разбиение.

5. СПИСОК РЕКОМЕНДУЕМОЙ ЛИТЕРАТУРЫ

№	Автор	Наименование, издательство, год издания
1.	Иванов Б.Н.	Дискретная математика. Алгоритмы и программы. Полный курс.-М.ФИЗМАТЛИТ. 2007. – 408 с.
2.	Кнут Д.Э.	Искусство программирования. Том 1, Основные алгоритмы. Издательство: Вильямс, 2012, с. 720.
3.	Кнут Д.Э.	Искусство программирования. Том 4, А. Комбинаторные алгоритмы. Часть 1. – Издательство: Вильямс, 2012, с. 960.
4.	Стивен С. Скиена	Алгоритмы. Руководство по разработке. – 2-е изд.: Пер. с англ.- СПб.: БХВ-Петербург, 2011.-720 с.
5.	Кормен Т., Лейзерстон Ч., Ривест Р., Штайн К.	Алгоритмы: построение и анализ. 2-е изд.: Пер. с англ.-М.: Издательский дом «Вильямс», 2012. – 1296 с.
6.	Захарова Л.Е.	Алгоритмы дискретной математики. – М.: Московский государственный институт электроники и математики, 2010.
7.	Новиков Ф.А.	Алгоритмы дискретной математики для программистов. – СПб.: Питер, 2001.
8.	Виленкин Н.Я., Виленкин А.Н., Виленкин П.А.	Комбинаторика. – М.: ФИМА, МЦНМО, 2006.
9.	Макконнелл Дж.	Анализ алгоритмов. Активный обучающий подход. - 3-е дополнительное изд.: Пер. с англ.- М.: ТЕХНОСФЕРА, 2013. – 415 с.