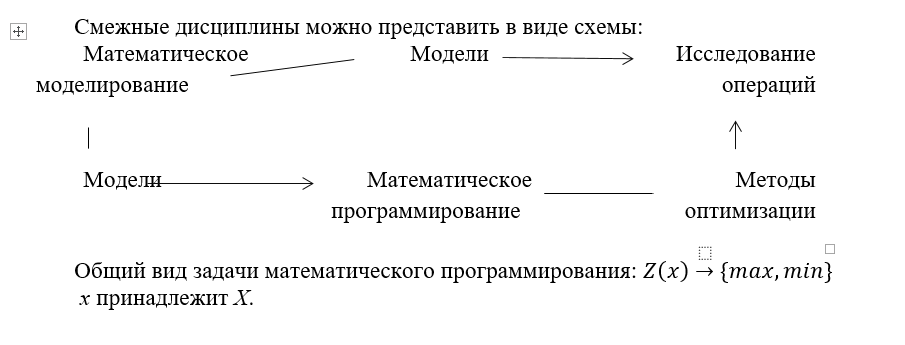
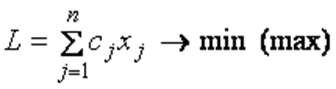
### *1.Предмет изучения, структура, цели и задачи курса. Общее понятие задачи оптимизации. Классификация задач оптимизации. Обзор методов решения оптимизационных задач.* *Смежные дисциплины.***Математическое программирование-** это раздел высшей математики, посвященный решению задач, связанных с нахождением экстремумов ф-ций нескольких переменных, при наличии ограничений на переменные. Математическое программирование можно рассматривать как совокупность самостоятельных разделов, занимающихся изучением и разработкой методов решения определенных классов задач. В зависимости от свойств целевой ф-ции и ф-ции ограничений все задачи математического программирования делятся на **2 основных класса:** 1.задачи линейного программирования,2.задачи нелинейного программирования; Если целевая функция и ф-ции ограничений – линейные ф-ции, то соответствующая задача поиска экстремума является задачей линейного программирования. Если хотя бы одна из указанных ф-ций нелинейна, то соответствующая задача поиска экстремума является задачей нелинейного программирования.

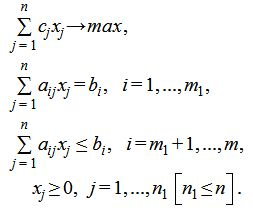


***Решение задачи математического программирования осуществляется в 4 этапа:***1. Построение математической модели,2. Классификация задачи,3. Выбор метода решения, 4. Вычисление   
Задачей оптимизации называется задача о нахождении экстремума (минимума или максимума) вещественной функции в некоторой области.  
**Классификация задач оптимизации:**1.Дискретное программирование,2.Целочисленного программирования, 3.Линейное программирование, 4.Нелинейное программирование,5.Векторная оптимизация  
Метод решения задачи математического программирования зависит от исходных данных. Вычисление решения задач мат. программирования. Осуществляется с помощью компьютера.

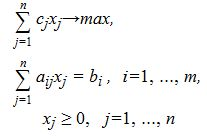


**2. Общая формулировка задачи линейной оптимизации. Формы записи задач линейной оптимизации.***ЛП* – метод математического моделирования, разработанный для оптимизации использования ограниченных ресурсов.Сущность линейного программирования состоит в нахождении наибольшего или наименьшего значения ф-ции при определённом наборе ограничений, налагаемых на аргументы. **Математическая модель любой задачи ЛП включает в себя:1.**Переменные, которые следует определить**;2.**Целевую ф-цию,подлежащую оптимизации; **3.**Систему ограничений в форме линейных уравнений и неравенств**;**Общая форма записи задачи линейного программирования: необходимо найти экстремальное значение целевой ф-ции: 

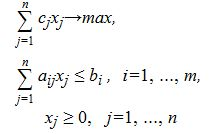
**Формы записи задач линейного программирования: общая, каноническая и стандартная:1.Общая форма ЗЛП**- это задача максимиза­ции или минимизации линейной функции с линейными ограничениями в виде как равенств, так и неравенств:



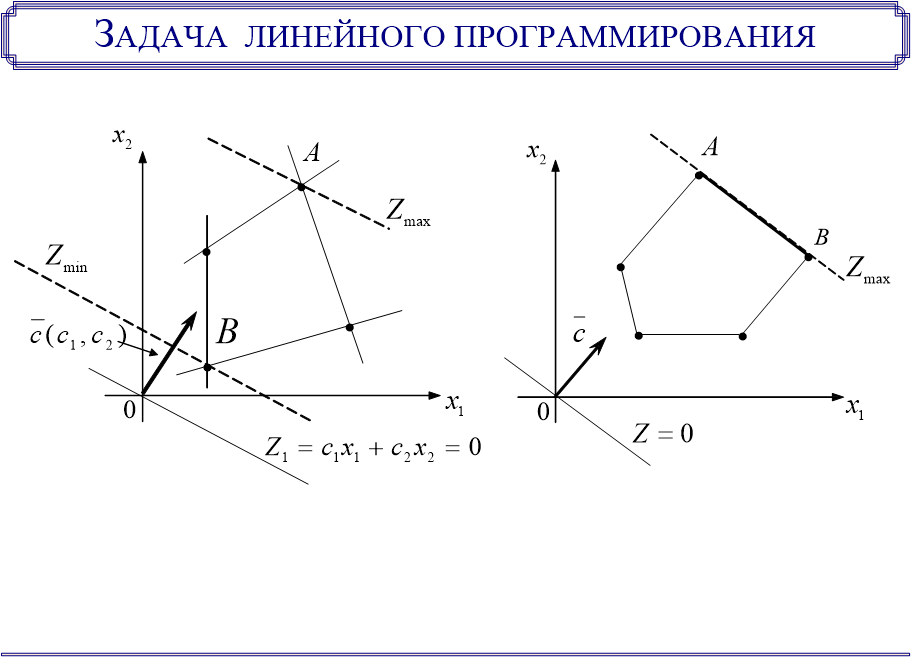
**2. Каноническая форма ЗЛП**- это задача максимиза­ции линейной функции с линейными ограничениями в виде ра­венств:

**

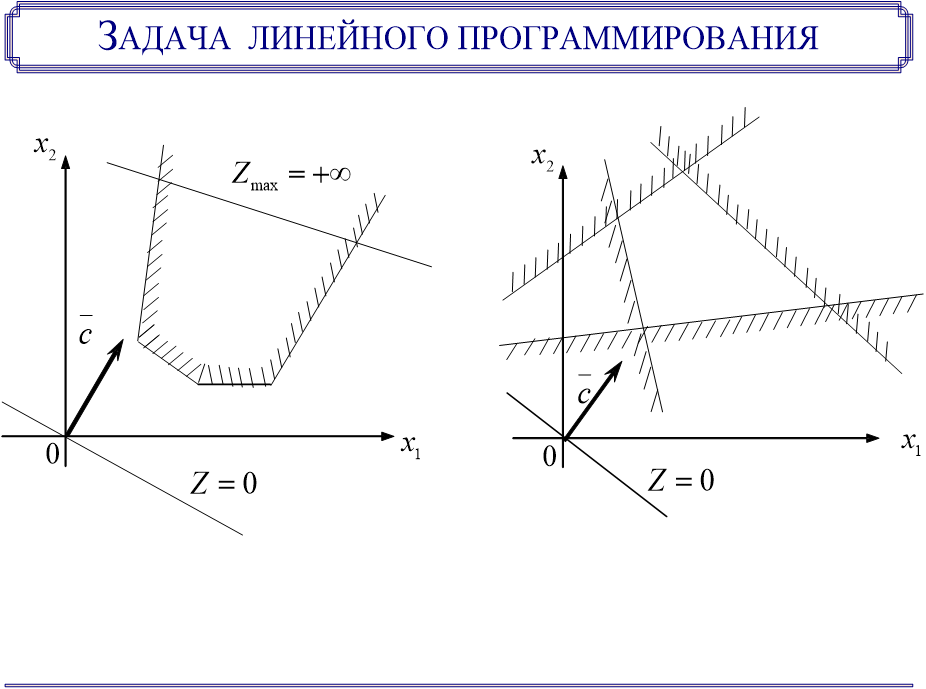
3. **Стандартная форма записи ЗЛП**– это задача максимизации (минимизации), если все ограничения – неравенства



### **3.Геометрический метод решения задачи линейной оптимизации. *Геометрический (графический) метод*** применим только для задач малой размерности (количество переменных в задаче равно 2 или 3), поскольку он основан на геометрическом построении множества допустимых решений в задаче ЛП. **Геометрический метод реализуется в два этапа: 1**)построение допустимого множества решений задачи ЛП, удовлетворяющих всем ограничениям модели; 2)нахождение оптимального решения среди всех точек пространства допустимых решений. Чтобы найти оптимальное решение необходимо определить направление возрастающей целевой функции *Z.* Можно приравнять *Z* к нескольким возрастающим значениям. Целевая функция может возрастать до тех пор, пока переменные, соответствующие возрастающему значению этой функции, пересекающей область допустимых решений. Оптимальное решение соответствует точке С, которая является местом пересечения прямых. Координата точки С(*x1*; *x2*) находится, как решение системы уравнений, пересечением которых она образована.



Если прямая функции параллельна отрезку [***АВ***], принадлежащему ОДР, то максимум функции ***Z*** достигается в точке ***А*** и в точке ***В***, а , следовательно, и в любой точке отрезка [***АВ***], т.к. эти точки могут быть выражены в виде линейной комбинации угловых точек ***А*** и ***В***.

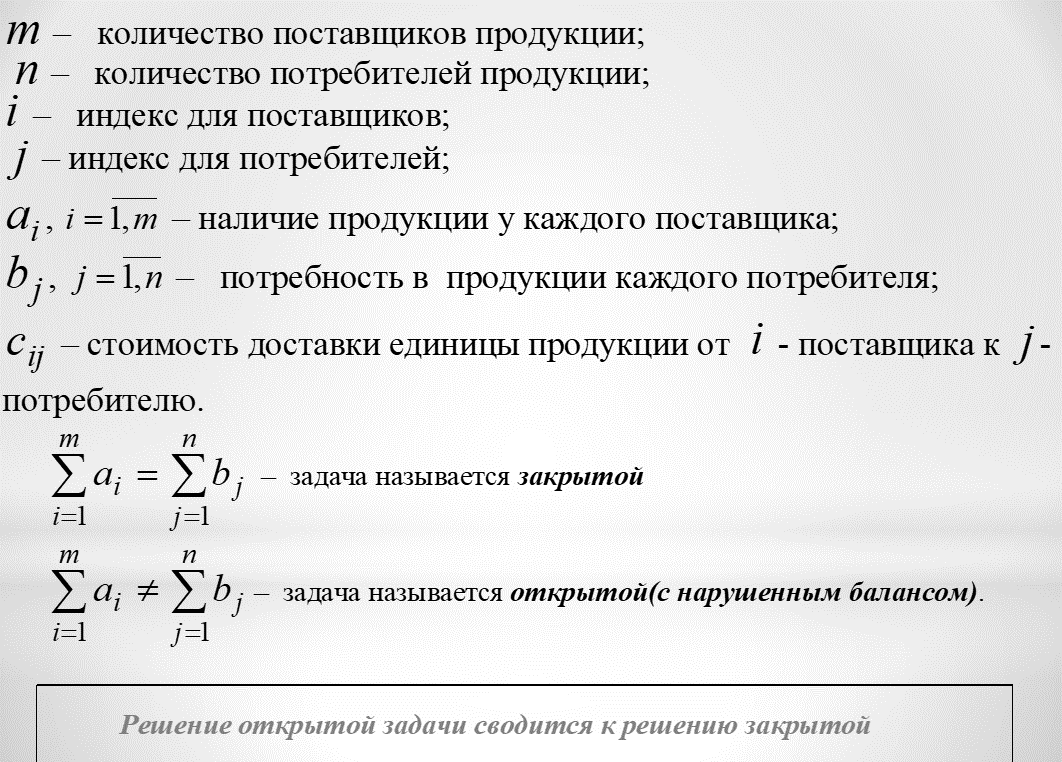
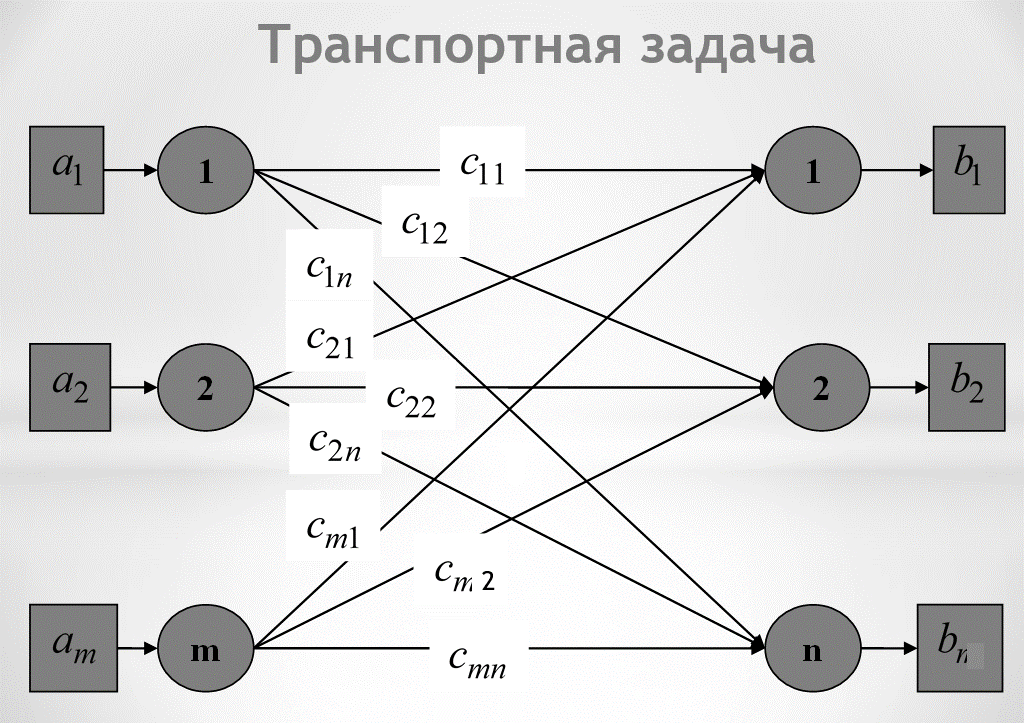


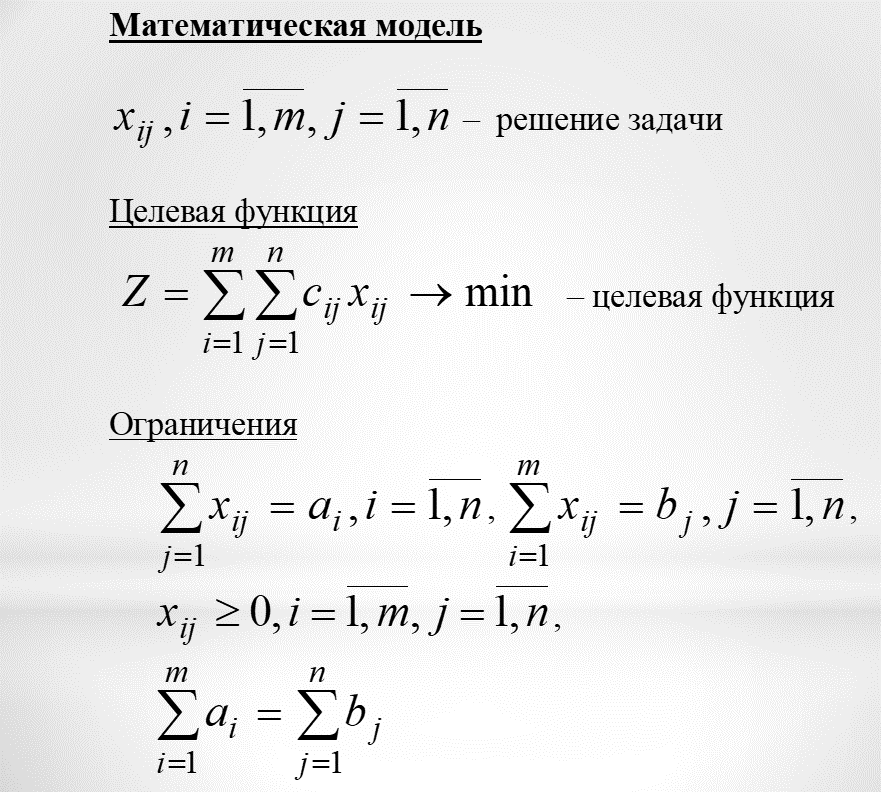
1.Cистема ограничений образует неограниченное сверху множество. Функция ***Z*** в данном случае стремится к бесконечности, так как прямую функции можно передвигать в направлении вектора градиента как угодно далеко.2.Случай несовместной системы ограничений.

### **4. Симплекс-метод решения задачи линейной оптимизации. Алгоритмы нахождения опорного(базисного) и оптимального решений.Симплекс-метод -** альтернатива графическому решению с тем преимуществом, что применим для задач любых размерностей. **Суть этого метода заключается в том, что: 1.**вначале получают допустимый вариант, удовлетворяющий всем ограничениям, но *не обязательно оптимальный* (так называемое начальное опорное решение); 2.*оптимальность* достигается последовательным улучшением исходного варианта за определенное число этапов (итераций); 3.*нахож­дение начального опорного решения и переход к следующему опорному* решению проводятся на основе применения метода Жордана-Гаусса для системы линейных уравнений *в канонической форме*, в которой должна быть предварительно записана исходная задача линейного программирования; 4.на­правление перехода от одного опорного решения к другому выбирается при этом на основе критерия оптимальности (целевой функции) исходной задачи. **Если говорить проще, то последовательность действий, выполняемых в симплекс-методе:1.**преобразование задачи в стандартную форму;**2.**находится начальное допустимое базисное решение;**3.**на основе условия оптимальности определяется вводимая переменная. Если вводимых переменных нет, вычисления заканчиваются;**4.**на основе условия допустимости выбирается исключаемая переменная;**5.**методом Гаусса-Жордана вычисляется новое базисное решение и осуществляется переход к пункту 3.

**Разберём алгоритм на следующей задаче линейного программирования:** 1)приводим к стандартной форме - каноническому виду, где все условия ограничения идут со знаком <= (кроме ограничений непосредственно на x1,x2,...) и задача исследуется на максимум. Для этого второе условие ограничения умножили на -1. 2)Чтобы все условия стали строгими, добавим дополнительные переменные (симплекс-метод с искусственным базисом. Если все условия строгие, то метод с естественным базисом, суть таже, но без доп переменных).3)Таким образом мы подготовили всё для симплекс-таблицы. Под х1, х2, у1, у2 пишем коэффициенты перед соответствующими переменными в условиях (т.е. будет 2 строки, ограничения что х и у неотрицательны - не в счёт), а bi - значение после равно.После отделяем линией эти две строки, и пишем коэффициенты и значение после равно для выражения F-3x1-4x2=0.4)Базис - это переменная, у которой в столбце только 1 ненулевой элемент. В нашем случае - это y1 и у2. 5)Если в последней строке есть отрицательные элементы, то план не оптимальный. Из отрицательных элементов определяется минимальный - так определяется разрешающий столбец. Далее необходимо определить разрешающую строку - заполнить последний столбец. По наименьшему положительному значению определяем разрешающую строку. На пересечении разрешающей строки и разрешающего столбца располагается разрешающий элемент. Далее необходимо разработать новый план. Для этого разрешающая строка переписывается без изменений, а остальные преобразуются так при помощи базисной строки умножением, делением суммой и разностью так, чтобы в базисном столбце ненулевым остался только базисный элемент.6)Повторяем шаг 4, и если в последней строке есть отрицательные элементы, то и шаг 5. Иначе мы получили оптимальное решение.

### **5. Транспортная задача. Математическая модель транспортной задачи. Алгоритм решения транспортной задачи. Методы построения исходного опорного решения.*Транспортная задача*** - математическая задача линейного программирования специального вида. Её можно рассматривать как задачу об оптимальном плане перевозок грузов из пунктов отправления в пункты потребления, с минимальными затратами на перевозки. Если суммарный объём предложений не равен общему спросу на товары, запрашиваемые пунктами потребления, то транспортная задача называется **несбалансированной**, или **открытой**.





Алгоритм решения состоит из двух этапов:1.Построение начального базисного решения: метод северо-западного угла, метод наименьшей стоимости(минимального элемента), метод Фогеля.2.Итеративный процесс поиска оптимального решения (метод потенциалов).  
**Метод «северо-западного угла»** состоит в последовательном переборе строк и столбцов транспортной таблицы, начиная с левого столбца и верхней строки, и выписывании максимально возможных отгрузок в соответствующие ячейки таблицы так, чтобы не были превышены заявленные в задаче возможности поставщика или потребности потребителя. На цены доставки в этом методе не обращают внимание, поскольку предполагается дальнейшая оптимизация отгрузок.  
**Метод «минимального элемента»**. Отличаясь простотой данный метод все же эффективнее чем, к примеру, метод Северо-западного угла. Кроме того, метод минимального элемента понятен и логичен. Его суть в том, что в транспортной таблице сначала заполняются ячейки с наименьшими тарифами, а потом уже ячейки с большими тарифами. То есть мы выбираем перевозки с минимальной стоимостью доставки груза. Это очевидный и логичный ход. Правда он не всегда приводит к оптимальному плану.  
**Метод «аппроксимации Фогеля»**. При методе аппроксимации Фогеля на каждой итерации по всем столбцам и по всем строкам находят разность между двумя записанными в них минимальными тарифами. Эти разности записывают в специально отведенных для этого строке и столбце в таблице условий задачи. Среди указанных разностей выбирают минимальную. В строке (или в столбце), которой данная разность соответствует, определяют минимальный тариф. Клетку, в которой он записан, заполняют на данной итерации.  
**Опорным решением транспортной задачи** называется любое допустимое решение, для которого векторы условий, соответствующие положительным координатам, линейно независимы. Для проверки линейной независимости векторов условий, соответствующих координатам допустимого решения, используют циклы.  
**Циклом** называется такая последовательность клеток таблицы транспортной задачи, в которой две и только соседние клетки расположены в одной строке или столбце, причем первая и последняя также находятся в одной строке или столбце. Система векторов условий транспортной задачи линейно независима тогда и только тогда, когда из соответствующих им клеток таблицы нельзя образовать ни одного цикла. Следовательно, допустимое решение транспортной задачи , i=1,2,...,m; j=1,2,...,n является опорным только в том случае, когда из занятых им клеток таблицы нельзя образовать ни одного цикла.

### **6. Метод потенциалов нахождения оптимального решения транспортной задачи.**В методе потенциалов каждой стоке i и каждому столбцу j транспортной таблице ставится в соответствие числа (потенциалы) Ui (поставщики) и Vj (потребители). Для каждой базисной переменной Xij потенциалы Ui и Vj удовлетворяют уравнению:***Ui +* *Vj* = *Cij*** Используется при решении транспортной задачи для оптимизации опорного плана. И сам алгоритм: **1.**Определяем потенциалы для всех базисных переменных.

**2.** Присваиваем одному из них произвольное значение (обычно *U1* = 0), и считаем исходя их полученных ранее уравнений (*Ui +* *Vj* = *Cij*)

**3.**Проверить план на оптимальность: Для каждой свободной клетки плана вычислим разности ΔCij = Cij — (Ui + Vj )  
План является оптимальным, если все разности ΔCij ≥ 0,  
Если план неоптимальный, то надо оптимизировать перераспределения постановок

**4.**Определяем небазисные переменные для свободных клеток. Вычисляем по формуле *Ui +* *Vj* **-** *Cij***5.**Вводимой в базис переменной будет та, которая имеет наибольшее положительное значение

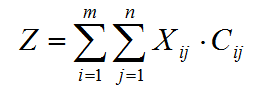
**6.**Строим замкнутый цикл, который начинается и заканчивается в искомой ячейке. Цикл состоит из последовательности вершин и горизонтальных отрезков, соединяющих ячейки с базисными переменными ввод. в ячейку(2,2). Так же в начальной ячейке ставим “+”, а в следующий чередуем “+”,”-”;

**7.**Ищем минимальное число в ячейках со знаком “-” и суммируем/отнимает от ячеек со знаками “+” и ”-” соответственно

**8.**Снова считаем значения потенциалов и разности ΔCij

**9.** Если план оптимальный идём дальше, **нет** - **4 пункт**

**10.**Считаем общие затраты на перевозку



### **7. Общие принципы решения задач оптимизации методом ветвей и границ. *Метод ветвей и границ*** – это общий алгоритмический метод решения задач комбинаторной оптимизации. Метод ветвей и границ был предложен для решения общей задачи целочисленного линейного программирования.Метод является вариацией полного перебора с отсеиванием подмножеств допустимых решений, заведомо не содержащих оптимальных решений. **В основе метода лежат две процедуры:1.**процедура ветвления, позволяющая разбивать множество допустимых решений на непересекающиеся подмножества,**2.**процедура вычисления нижней или верхней границы. *Далеко не для всех задач комбинаторной оптимизации найдены процедуры ветвления и оценки, позволяющие их решить методом ветвей и границ.* Метод ветвей и границ в основном используется для решения задачи коммивояжера — задачи нахождения самого выгодного маршрута, проходящего через все заданные точки по одному разу, с последующим возвратом в исходную точку. Есть много методов ее решения. **Суть метода ветвей и границ** состоит в последовательном переборе вариантов, рассмотрении лишь тех из них, которые по оказываются перспективными, и отбрасывании бесперспективных [неэффективных] вариантов. При использовании метода ветвей и границ область допустимых решений (ОДР) исходной задачи разбивается на непересекающиеся подмножества, и циклически решаются подзадачи. В основе метода ветвей и границ лежат две основные операции: как ни странно, ветвление (BR) и выделение границ (EV).**Ветвление (BR):** Пусть полный взвешенный ориентированный граф ***G*** = (***V, E***) с весовой функцией ***w*** моделирует города (множество вершин *V*) и расстояния между ними (взвешенные дуги *E*) в задаче коммивояжера. Решение этой задачи сводится к отысканию кольцевого маршрута ***r*** проходящего через все вершины графа и имеющего минимальную сумму весов дуг, составляющих кольцевой маршрут (кратчайший кольцевой маршрут). Пусть ***R*** – множество всех гамильтоновых циклов графа. Очевидно, что множество ***R*** соответствует множеству всех допустимых решений задачи коммивояжера. Решением же задачи является нахождение гамильтонова цикла графа (т.е. цикла по всем городам один раз с возвращением в исходный).То есть, в ходе решения задача будет ветвиться на разные варианты решения, которые мы будем отбрасывать или принимать. Как раз разбиение задачи на разные варианты называется ветвлением. **Вычисление границ (EV):** *Основывается на двух утверждениях:* ***Утверждение 1.*** Изменение всех элементов строки матрицы расстояний на одно и то же число не влияет на выбор оптимального маршрута коммивояжера. ***Утверждение 2.*** Изменение всех элементов столбца матрицы расстояний на одно и то же число не влияет на выбор оптимального маршрута коммивояжера.То есть в ходе решения будут находиться границы — минимальные возможные варианты решения задачи [минимальный вес гамильтонова цикла, т.е. минимальное расстояние маршрута].**Основные принципы:1.**изменение всех элементов строки или столбца матрицы расстояний на одно и то же число не влияет на итоговый выбор оптимального маршрута. **2.**редукция строк и столбцов не влияет на итоговое решение**3.** сумма констант приведения равна локальной нижней границе.**4.**корневая локальная нижняя граница выбирается только один раз.**5.**при выборе следующей ветви решения выбирается еще не ветвившееся решение с минимальной границей

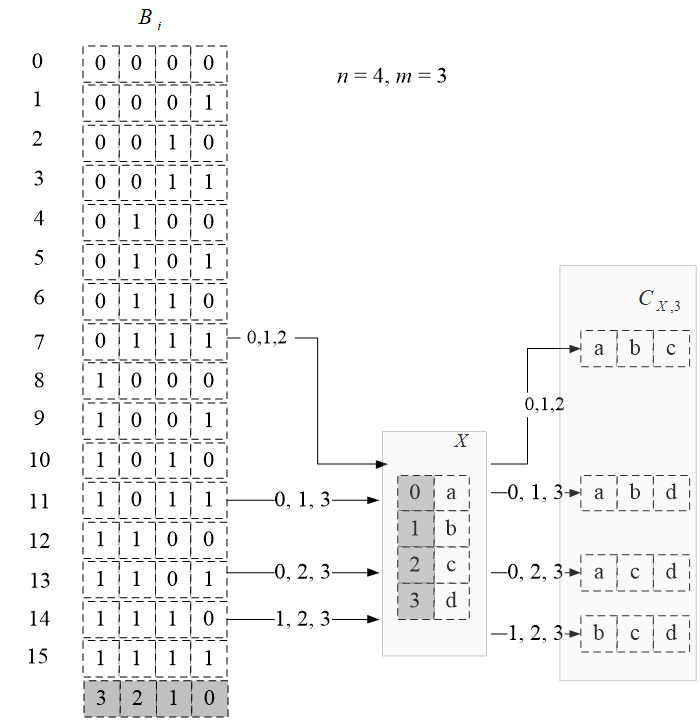
### **8. Комбинаторные методы решения оптимизационных задач. Генерация подмножеств заданного множества.**

Комбинаторный анализ (комбинаторика, комбинаторная математика) – раздел математики, посвященный решению задач выбора и расположения элементов некоторого, обычно конечного, множества в соответствии с заданными правилами.**Булеан** (*степень множества*, *показательное множество*, *множество частей*) — множество всех [подмножеств](https://ru.wikipedia.org/wiki/%D0%9F%D0%BE%D0%B4%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2%D0%BE) данного множества А включая нулевое и само множество А{0,1}. Алгоритм генерации множества всех подмножеств основывается на взаимно однозначном соответствии между элементами булеана 2^x множества х и всеми целыми числами множества {0,1….,2^|x|-1},записанными в двоичном виде. Построение элементов булеана множества сводится к следующему алгоритму:1. Пронумеровать элементы заданного множества начиная с нуля. 2. Сформировать битовую последовательность состоящую из двоичных нулей. Пронумеровать элементы этой последовательности справа налево, начиная с нуля.3. Последовательно выполнить шаги 4 и 5 алгоритма раз.4. Выбрать из множества элементы с номерами для которых Полученное подмножество будет являться элементом булеана В первом случае не будет выбран ни один элемент (пустое подмножество) множества так как исходная последовательность состоит только из нулей. 5. Интерпретируя битовую последовательность как целое положительное число, увеличить это число на единицу. Количество: *2n*

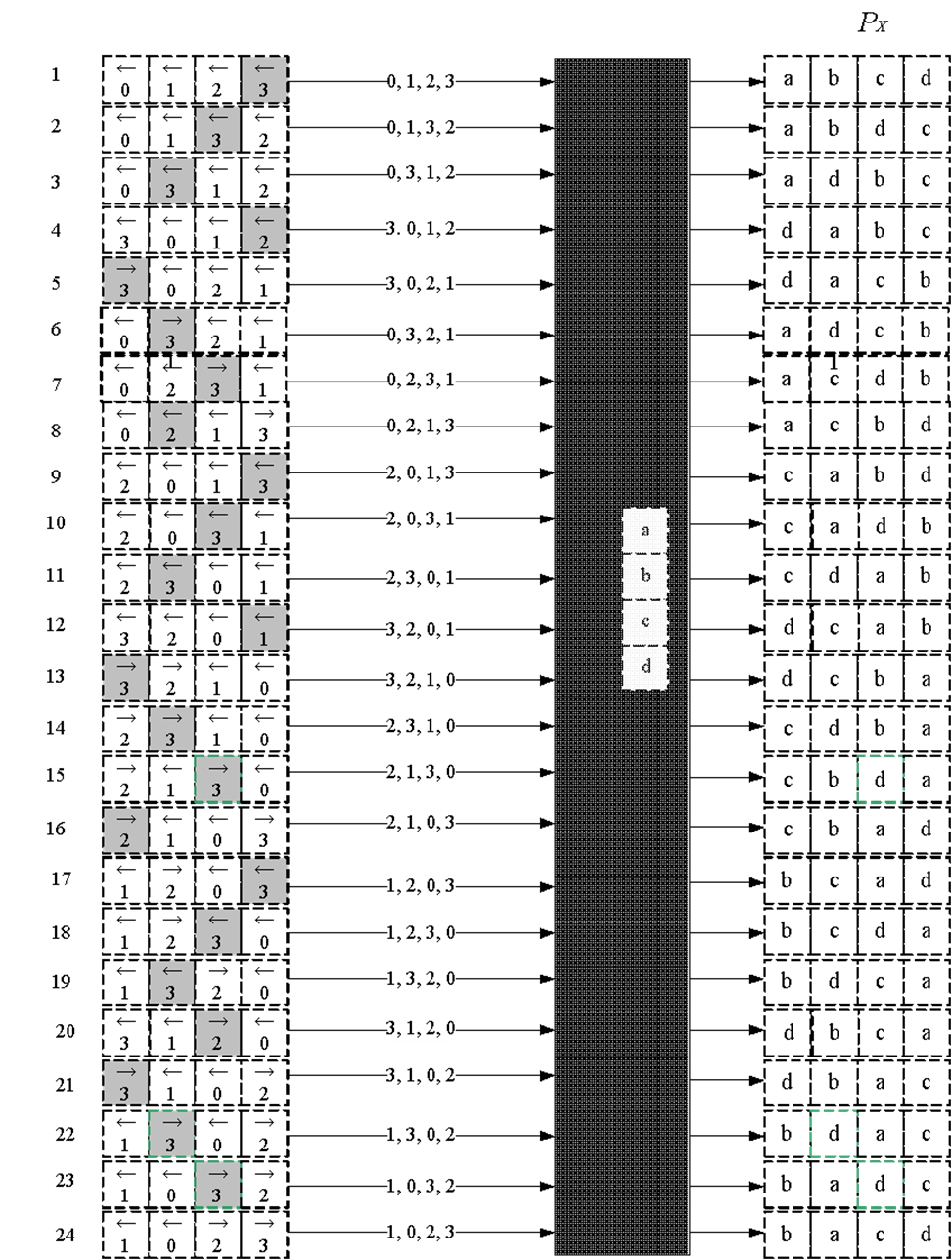
**9. Комбинаторные методы решения оптимизационных задач. Генерация сочетаний.**Комбинаторный анализ (комбинаторика, комбинаторная математика) – раздел математики, посвященный решению задач выбора и расположения элементов некоторого, обычно конечного, множества в соответствии с заданными правилами.

**Комбинаторные методы решения оптимизационных задач:Генерация подмножеств заданного множества**:**1.** разработка генератора подмножеств на С++;**2.**решение задачи о рюкзаке. **Генерация сочетаний:1.**разработка генератора сочетаний на С++;**2.**решение задачи об оптимальной загрузке.**Генерация перестановок: 1.**разработка генератора перестановок на С++;**2.**решение задачи о коммивояжере.**Генерация размещений: 1.**разработка генератора сочетаний на С++;**2.**решение задачи об оптимальной загрузке (с центровкой)

**Генерация сочетаний.**Схема построения множества сочетаний ***CX,3*** из элементов множества ***X***={***a, b, c, d***} Закрашенным прямоугольником на рисунке обозначены номера (индексы) элементов битовых последовательностей ***Bi*** и элементов множества X. Стрелки связывают битовые последовательности, содержащие три двоичные единицы и сгенерированные сочетания множества CX,3 Для каждой стрелки указаны индексы единичных позиций соответствующих битовых последовательностей. Эти индексы используются для выбора элементов из множества X для включения в соответствующее сочетание. Очевидно, что такой алгоритм генерации сочетаний имеет сложность как и алгоритм генерации множества всех подмножеств.

******

### **10. Комбинаторные методы решения оптимизационных задач. Генерация перестановок. Комбинаторные методы решения оптимизационных задач:***1. Генерация подмножеств заданного множества*( задачи о рюкзаке.);*2. Генерация сочетаний(*задачи об оптимальной загрузке).*3. Генерация перестановок:*(задачи о коммивояжере).*4. Генерация размещений*(задачи об оптимальной загрузке(с центровкой)) **Особенность алгоритмов:** 1. *Невозможно использовать для задач большой размерности.* 2. *Применяются тогда, когда требуется:*- точное решение или достаточное решение; - размерность задачи небольшая. 3. *Сложность этих алгоритмов:*- является верхней оценкой сложности решения задач;- не зависит от данных; - улучшение возможно только в статистическом смысле. Наиболее известным методом построения множества всех перестановок конечного множества Х является **алгоритм Джонсона** – **Троттера**. Алгоритм подразумевает, что все элементы множества Х можно единственным способом перечислить в порядке возрастания. Для конечного множества такой порядок всегда можно установить. В простейшем случае элементы конечного множеств Х можно перенумеровать и считать, что еслиВ дальнейшем будем предполагать, что – отрезок натурального ряда. Это позволяет сравнивать элементы естественным способом и никак не мешает обобщению. Каждый элемент исходного множества Х помечается специальным символом – стрелкой, которая может быть направлена влево или вправо. Первая перестановка в алгоритме Джонсона – Троттера выглядит следующим образом: В алгоритме используется понятие ***мобильного элемента***. Элемент последовательности элементов множества Х называется ***мобильным***, если соответствующая ему стрелка указывает на меньший соседний элемент. В первой перестановке все элементы кроме самого левого являются мобильными. ***Построение множества всех перестановок с помощью алгоритма Джонсона – Троттера сводится к следующей процедуре:1.***Построить первую перестановку. Первая перестановка – это последовательность всех элементов множества перечисленных в порядке возрастания. Стрелки всех элементов последовательности направлены влево.2.Найти наибольший мобильный элемент в текущей перестановке. Если в последовательности нет мобильного элемента, то построены все перестановки элементов множества Х– алгоритм закончил свою работу.3.Поменять местами наибольший мобильный элемент и элемент, на который указывает стрелка наибольшего мобильного элемента.4.Найти все элементы, большие, чем мобильный элемент (если они есть) и изменить их стрелки на противоположное направление.5. Перейти к пункту 2. Количество перестановок: *Pn=n!.* Схема алгоритма генерации множества всех перестановок множества приведена на рис. 1.



Второй слева столбец на – массив индексов, генерируемый с помощью алгоритма Джонсона – Троттера. Каждый массив индексов является одной из перестановок элементов множества . Количество массивов составляет: . Самый правый столбец (обозначен ) – множество всех перестановок элементов  Перестановки формируются как результат индексации массива

### **11.Комбинаторные методы решения оптимизационных задач. Генерация размещений. Комбинаторные методы решения оптимизационных задач:***1. Генерация подмножеств заданного множества*( задачи о рюкзаке.);*2. Генерация сочетаний(*задачи об оптимальной загрузке).*3. Генерация перестановок:*(задачи о коммивояжере).*4. Генерация размещений*(задачи об оптимальной загрузке(с центровкой))

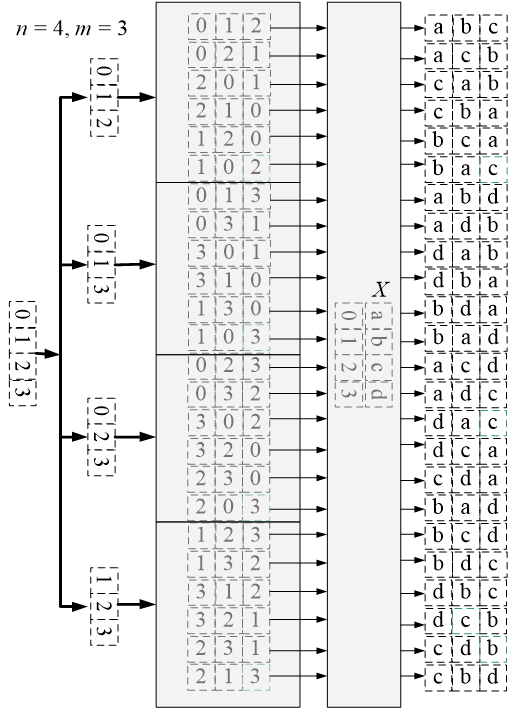
Для того чтобы проследить этапы генерации множества AX,3 на схеме, ее следует рассматривать слева направо. В крайней левой позиции изображено множество {0, 1, 2, 3}, на основе которого формируются четыре сочетания.

Затем каждое сочетание рассматривается как отдельное множество, состоящее из трех элементов. Для каждого множества формируется по 3!=6 перестановок. В итоге в третьем слева столбце на схеме отображены 4∙6=24 перестановки множества {0, 1, 2, 3}.

На последнем этапе формируется множество AX,3 всех перестановок элементов множества X={a, b, c, d}. Элементы множества AX,3 отображены в крайнем справа столбце.



Схема построения:



### **12.Динамическое программирование. Вычислительная схема решения задачи динамического программирования. Динамическое программирование** - способ решения сложных задач путём разбиения их на более простые подзадачи. Типичная задача динамического программирования - задача о рюкзаке.

### 

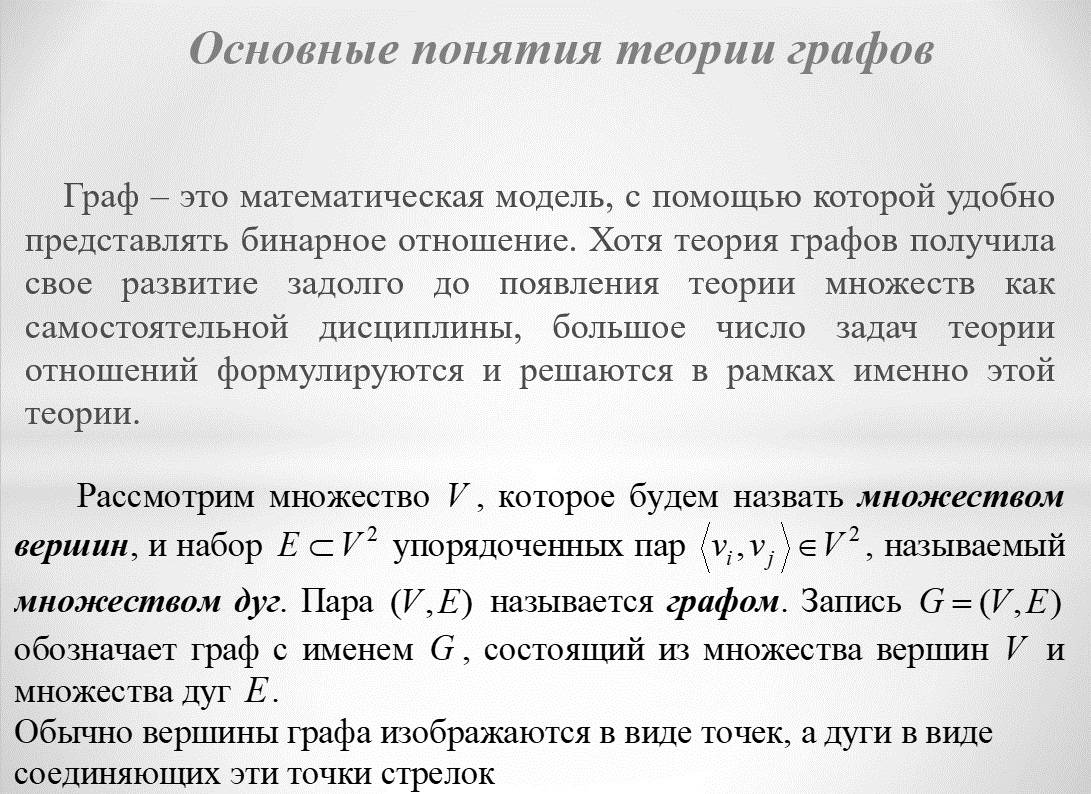
### 

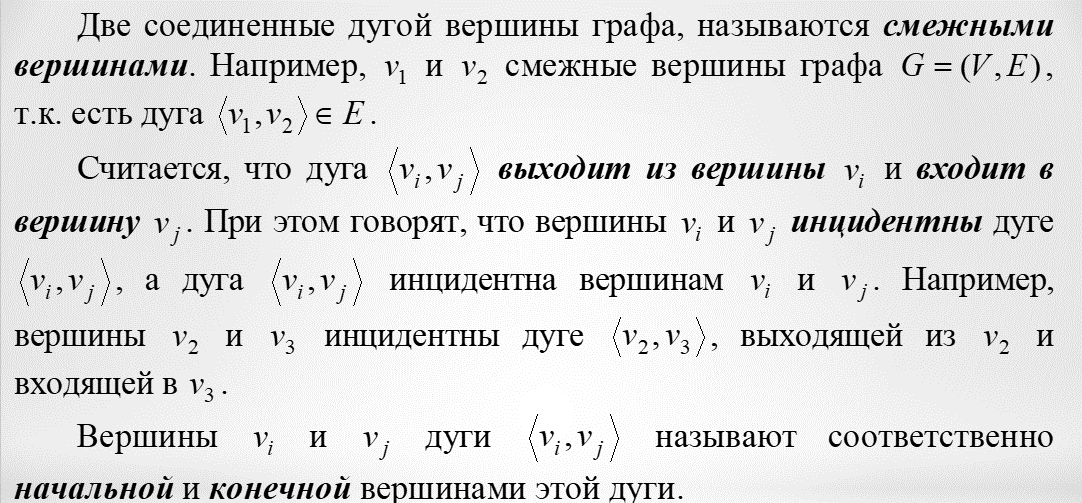
Пусть A(k,s) есть максимальная стоимость предметов, которые можно уложить в рюкзак вместимости s, если можно использовать только первые k предметов, то есть {n1,n2,…,nk}, назовем этот набор допустимых предметов для A(k,s).A(k,0)=0A(0,s)=0.Найдем A(k,s). Возможны 2 варианта: Если предмет k не попал в рюкзак. Тогда A(k,s) равно максимальной стоимости рюкзака с такой же вместимостью и набором допустимых предметов {n1,n2,…,nk−1}, то есть A(k,s)=A(k−1,s);Если k попал в рюкзак. Тогда A(k,s) равно максимальной стоимости рюкзака, где вес s уменьшаем на вес k-ого предмета и набор допустимых предметов {n1,n2,…,nk−1} плюс стоимость k, то есть A(k−1,s−wk)+pk; A(k,s)={A(k−1,s),A(k−1, s−wk)+pk, bk=0bk=1.То есть: A(k,s)=max(A(k−1,s),A(k−1,s−wk)+pk) Стоимость искомого набора равна A(N,W), так как нужно найти максимальную стоимость рюкзака, где все предметы допустимы и вместимость рюкзака W. Восстановим набор предметов, входящих в рюкзак. Будем определять, входит ли ni предмет в искомый набор. Начинаем с элемента A(i,w), где i=N, w=W. Для этого сравниваем A(i,w) со следующими значениями: Максимальная стоимость рюкзака с такой же вместимостью и набором допустимых предметов {n1,n2,…,ni−1}, то есть A(i−1,w). Максимальная стоимость рюкзака с вместимостью на wi меньше и набором допустимых предметов {n1,n2,…,ni−1} плюс стоимость pi, то есть A(i−1,w−wi)+pi. Заметим, что при построении A мы выбирали максимум из этих значений и записывали в A(i,w). Тогда будем сравнивать A(i,w) c A(i−1,w), если равны, тогда ni не входит в искомый набор, иначе входит. Метод динамического программирование всё равно не позволяет решать задачу за полиномиальное время, потому что его сложность зависит от максимального веса. Задача о ранце (или задача о рюкзаке) — одна из NP-полных задач комбинаторной оптимизации.]

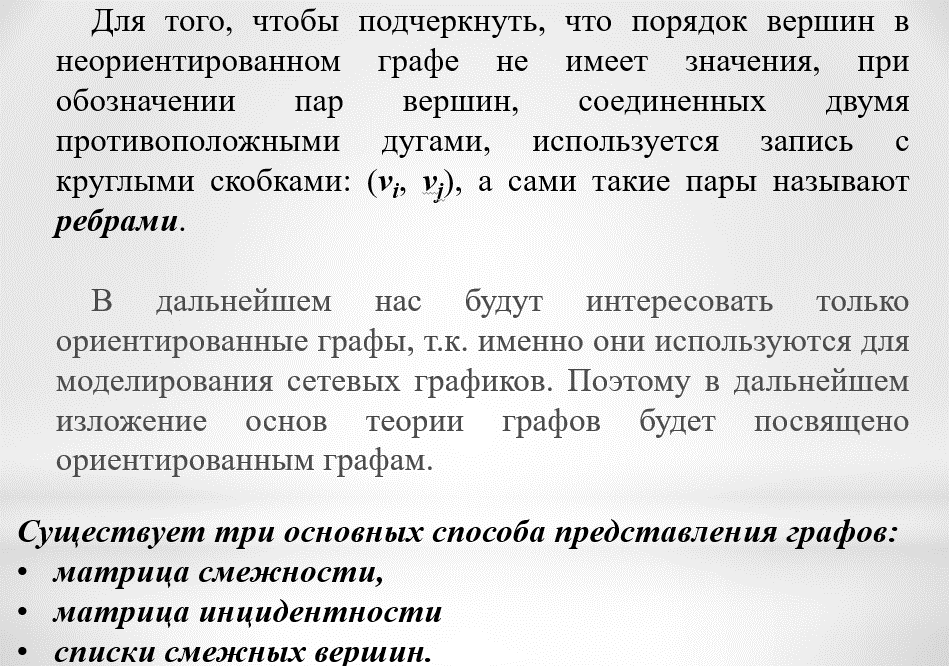
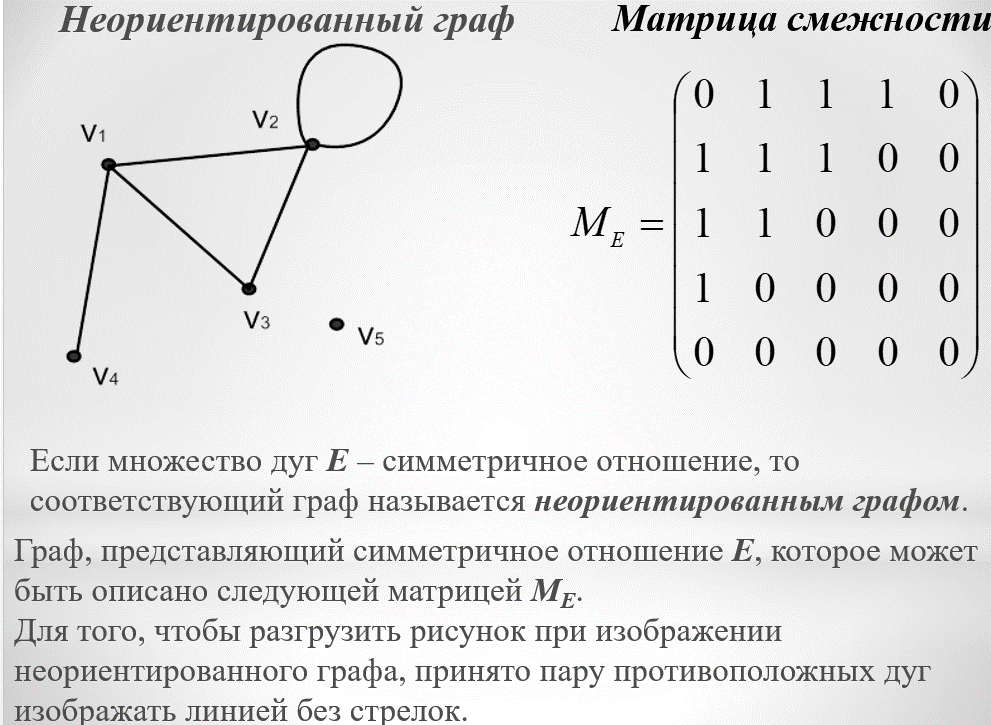
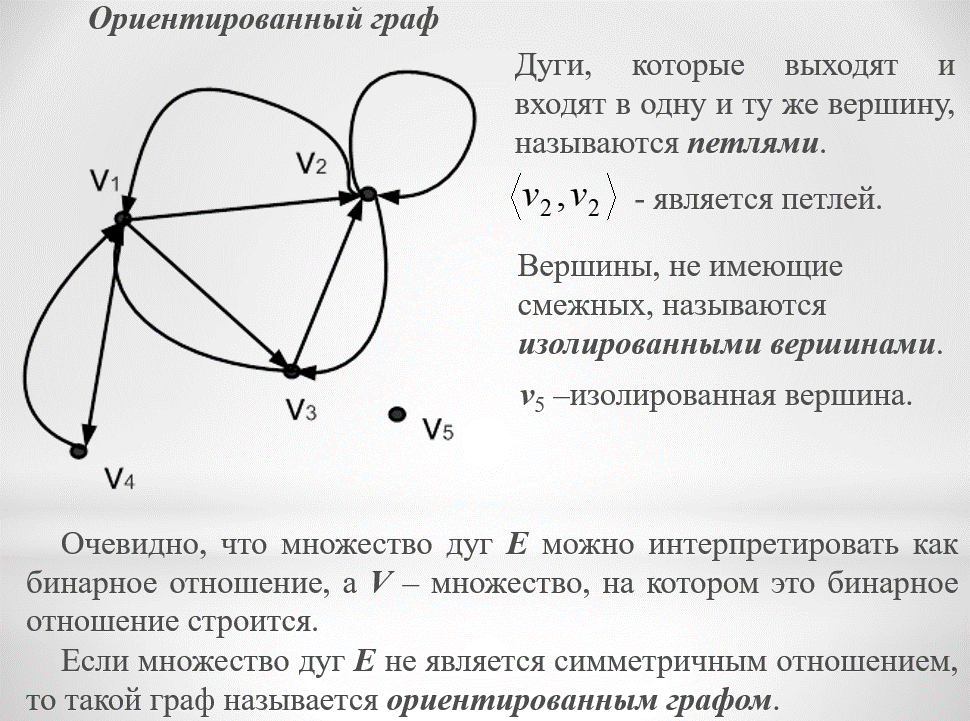
### **13. Основные приложения динамического программирования. Обзор задач, решаемых методами динамического программирования.** Если в процессе выполнения рекурсивного алгоритма одна и та же задача решается несколько раз, то говорят, что алгоритм содержит перекрывающиеся подзадачи. Метод решения задачи оптимизации, реализующей рекурсивный алгоритм с перекрывающимися подзадачами, в котором каждая такая подзадача решается один раз, а ее результат сохраняется для последующего применения, называется динамическим программированием. Задачи, решаемые методом динамического программирования: Решение **задачи о рюкзаке** (уложить как можно большее число ценных вещей в рюкзак при условии, что вместимость рюкзака ограничена), о **расстановки скобок при перемножении матриц**, о **наибольшей общей подпоследовательности** (X - подпоследовательность Y, если удаляя элементы из Y можно получить X, прим: X = A,D,C,F,G,E; Y = D,F,E, Y - подпоследовательность, т.к. при удалении A,C,G можно получить D,F,E), **вычисление дистанции Левенштейна** (минимальное количеству операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую). Применение динамического программирования при решении оптимизационных задач обычно предполагает создание специальных таблиц для хранения промежуточных результатов.

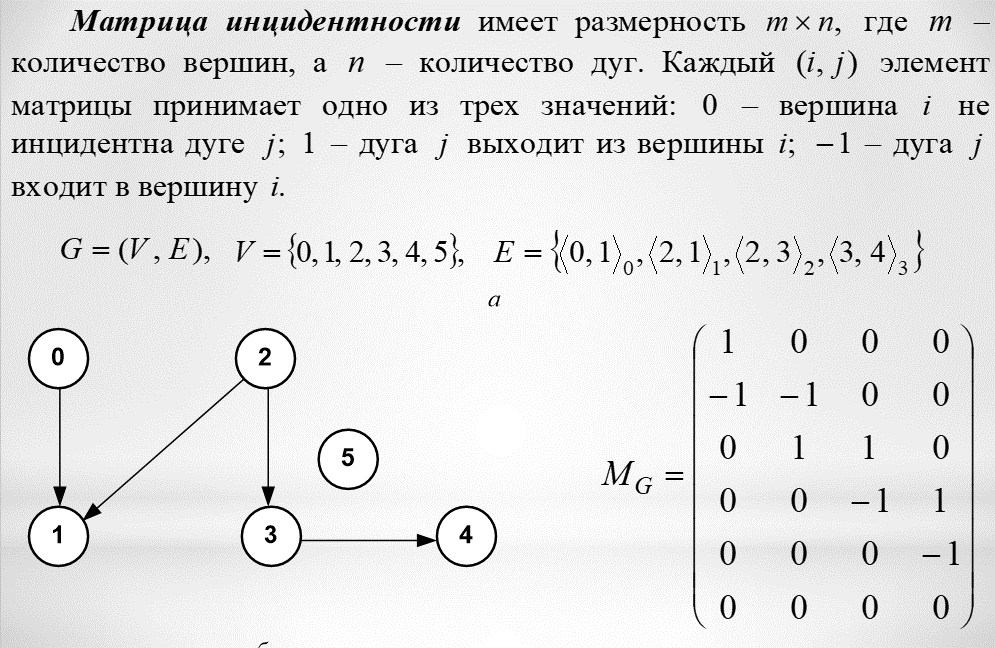
### **14. Рекурсивные алгоритмы.**Многие оптимизационные алгоритмы основаны на принципе разбиения основной задачи на подзадачи, каждая из которых повторяет основную, но входные их данные таковы, что область допустимых решений становится меньше.**Рекурсивный алгоритм** – это алгоритм, решающий задачу путем сведения ее к решению одной или нескольких таких же задач, но в сокращенном варианте. В решении используются рекурсивные функции: **Рекурсивная функция** – 1) Функция, которая вызывает саму себя. (на языке программирования). 2) Функция, заданная с помощью последовательных частичных функций таких, что каждая из них либо является простейшей, либо получена из предыдущих (на языке алгоритмизации). Рекурсивный алгоритм может быть записан в виде рекурсивной функции. Рекурсивную функцию всегда можно преобразовать в цикл, и, наоборот любой цикл можно представить в виде рекурсивной функции. Рекурсивная запись алгоритма, как правило, не дает выигрыша в скорости его работы. Скорее наоборот, так как вызов любой функции связан с сохранением и восстановлением контекста вызывающей функции, что является затратной по времени операцией. Кроме того, для хранения контекста ОС резервирует системный стек. Если цепочка вызовов функций является длинной (иногда говорят о большой глубине рекурсии), то это может привести к переполнению стека. Например, при вычислении факториала числа 25 глубина рекурсии достигает значения 24. Поэтому рекурсивные вычисления довольно медлительны. Классическими примерами рекурсивных функций являются функции для вычисления факториала, чисел Фибоначчи и наибольшего общего делителя с помощью алгоритма Евклида. Часто рекурсивные функции, применяемые для решения оптимизационных задач, используют более одного рекурсивного вызова, каждый из которых работает приблизительно с половиной входных данных. Такую схему решения называют «разделяй и властвуй». Рекурсивно решаются задача о рюкзаке и расстояние Левенштейна.

### **15. Математические основы сетевого планирования. Основные понятия теории графов.**

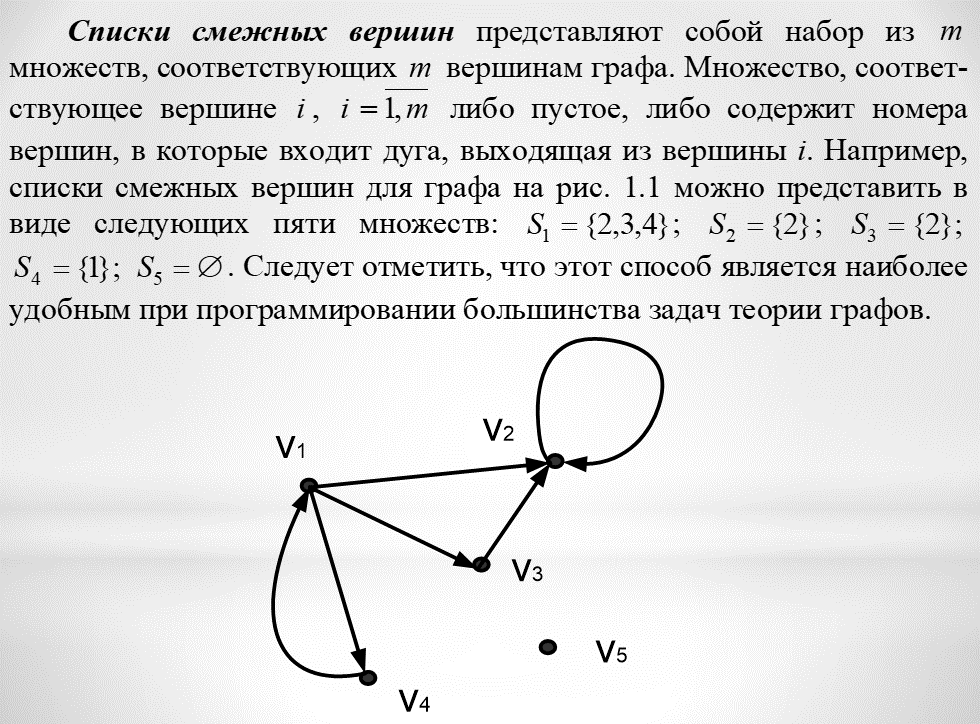




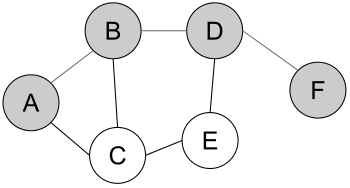




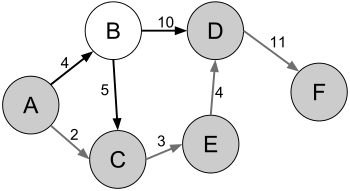
Следует обратить внимание, что петле в матрице ***M*** соответствует столбец с одной положительной единицей, что не всегда удобно при вычислениях. Например, при подсчете количества входящих в вершину дуг следует всегда учитывать, что может быть петля, которой нет соответствующей -1. Поэтому матрицы инцидентности применяются редко, особенно, если граф может иметь петли



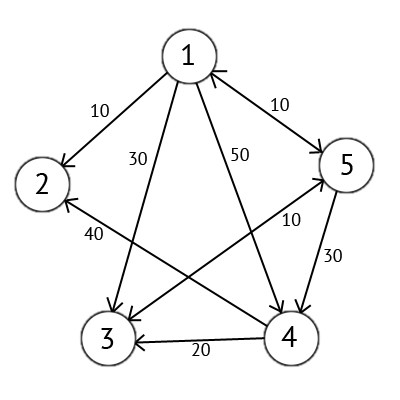
**16.Математические основы сетевого планирования. Кратчайшие пути между вершинами графа. Определение.** *Кратчайшим путем* между вершинами *a* и *b* в неориентированном графе называется путь между ними, содержащий наименьшее количество ребер. В зависимости от контекста, под *длиной пути* может иметься в виду как число ребер (*k*), число промежуточных вершин (*k*−1), так и суммарное число вершин (*k*+1), включая начало и конец. Кратчайший путь между парой вершин не всегда уникален.

От A до F три перехода

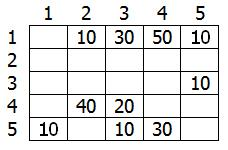
**Определение.** *Взвешенным графом* называется граф, в котором каждому ребру сопоставляется число, называемое *весом*, *длиной* или *стоимостью*.Во взвешенных графах длиной пути называется суммарная длина всех его ребер.



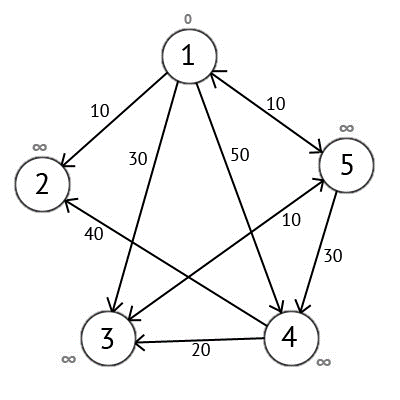
Несмотря на то, что в выделенном пути больше ребер, его суммарная стоимость меньше. Если не сказано обратного, длина ребра может быть любым числом — в том числе отрицательным. В графах с отрицательными ребрами может возникнуть контринтуитивная ситуация, когда есть *цикл* отрицательного веса, по которому можно ходить бесконечно и таким образом получить путь произвольного малого веса между двумя вершинами, в промежутке проходя много раз по этому циклу. Поэтому обычно рассматривают графы с неотрицательными длинами ребер. Для поиска кратчайшего пути используется алгоритмы Дейкстры, Беллмана-Форда. Алгоритм Дейкстры: Для примера возьмем такой ориентированный граф G:



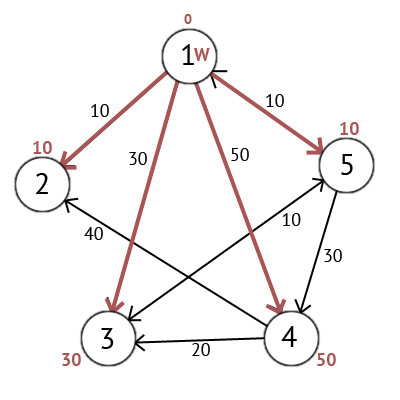
Этот граф мы можем представить в виде матрицы С:



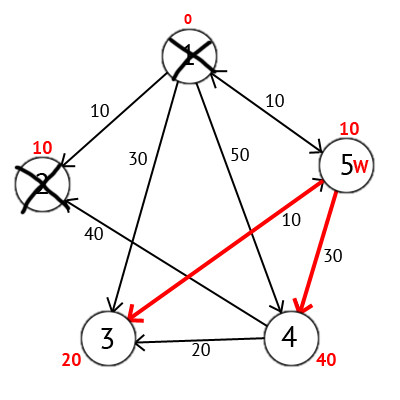
Возьмем в качестве источника вершину 1. Это значит что мы будем искать кратчайшие маршруты из вершины 1 в вершины 2, 3, 4 и 5. Данный алгоритм пошагово перебирает все вершины графа и назначает им метки, которые являются известным минимальным расстоянием от вершины источника до конкретной вершины. Рассмотрим этот алгоритм на примере. Присвоим 1-й вершине метку равную 0, потому как эта вершина — источник. Остальным вершинам присвоим метки равные бесконечности.



Далее выберем такую вершину W, которая имеет минимальную метку (сейчас это вершина 1) и рассмотрим все вершины в которые из вершины W есть путь, не содержащий вершин посредников. Каждой из рассмотренных вершин назначим метку равную сумме метки W и длинны пути из W в рассматриваемую вершину, но только в том случае, если полученная сумма будет меньше предыдущего значения метки. Если же сумма не будет меньше, то оставляем предыдущую метку без изменений.



После того как мы рассмотрели все вершины, в которые есть прямой путь из W, вершину W мы отмечаем как посещённую, и выбираем из ещё не посещенных такую, которая имеет минимальное значение метки, она и будет следующей вершиной W. В данном случае это вершина 2 или 5. Если есть несколько вершин с одинаковыми метками, то не имеет значения какую из них мы выберем как W. Мы выберем вершину 2. Но из нее нет ни одного исходящего пути, поэтому мы сразу отмечаем эту вершину как посещенную и переходим к следующей вершине с минимальной меткой. На этот раз только вершина 5 имеет минимальную метку. Рассмотрим все вершины в которые есть прямые пути из 5, но которые ещё не помечены как посещенные. Снова находим сумму метки вершины W и веса ребра из W в текущую вершину, и если эта сумма будет меньше предыдущей метки, то заменяем значение метки на полученную сумму.

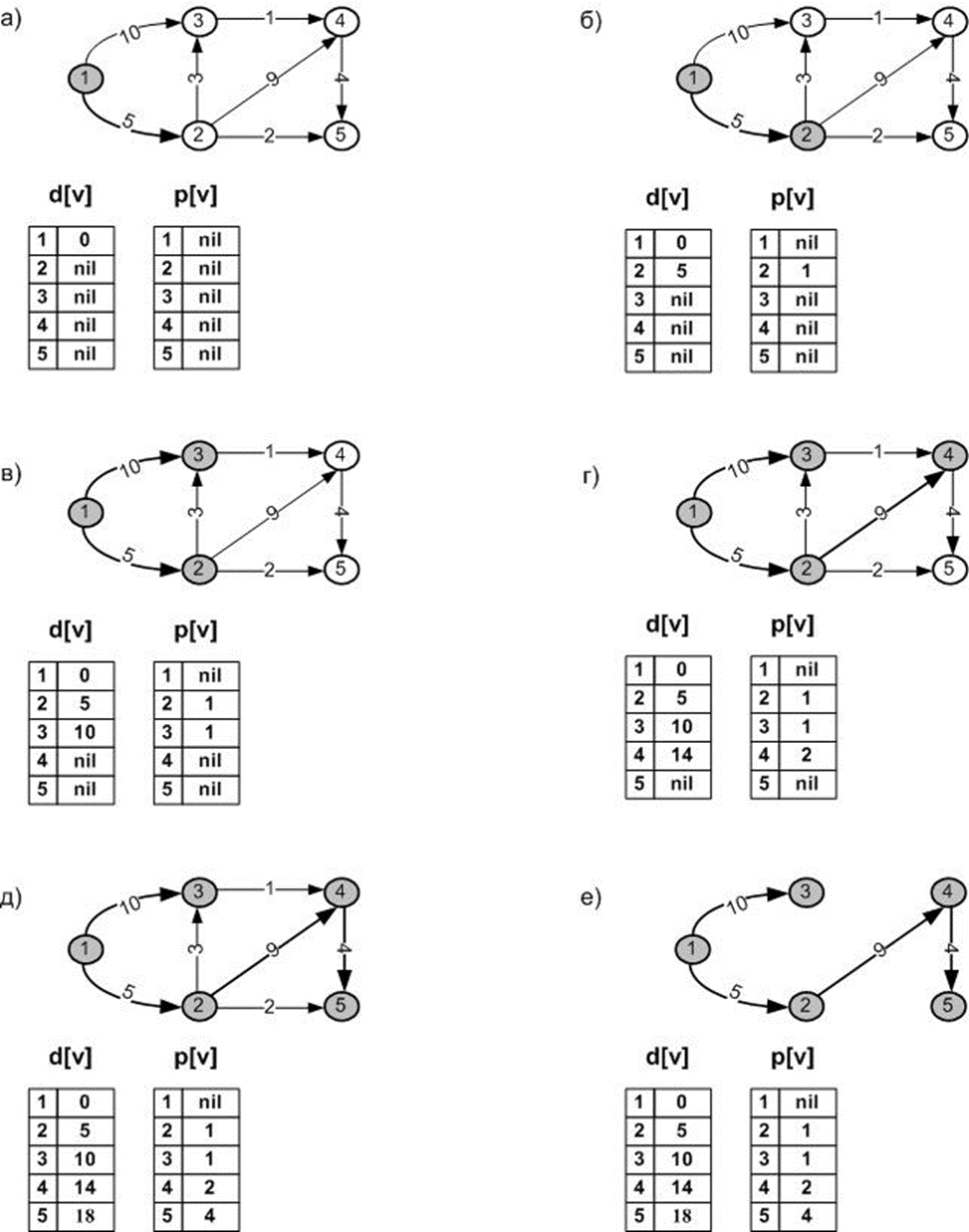


Исходя из картинки мы можем увидеть, что метки 3-ей и 4-ой вершин стали меньше, то есть был найден более короткий маршрут в эти вершины из вершины источника. Далее отмечаем 5-ю вершину как посещенную и выбираем следующую вершину, которая имеет минимальную метку. Повторяем все перечисленные выше действия до тех пор, пока есть непосещенные вершины.

Выполнив все действия получим такой результат:

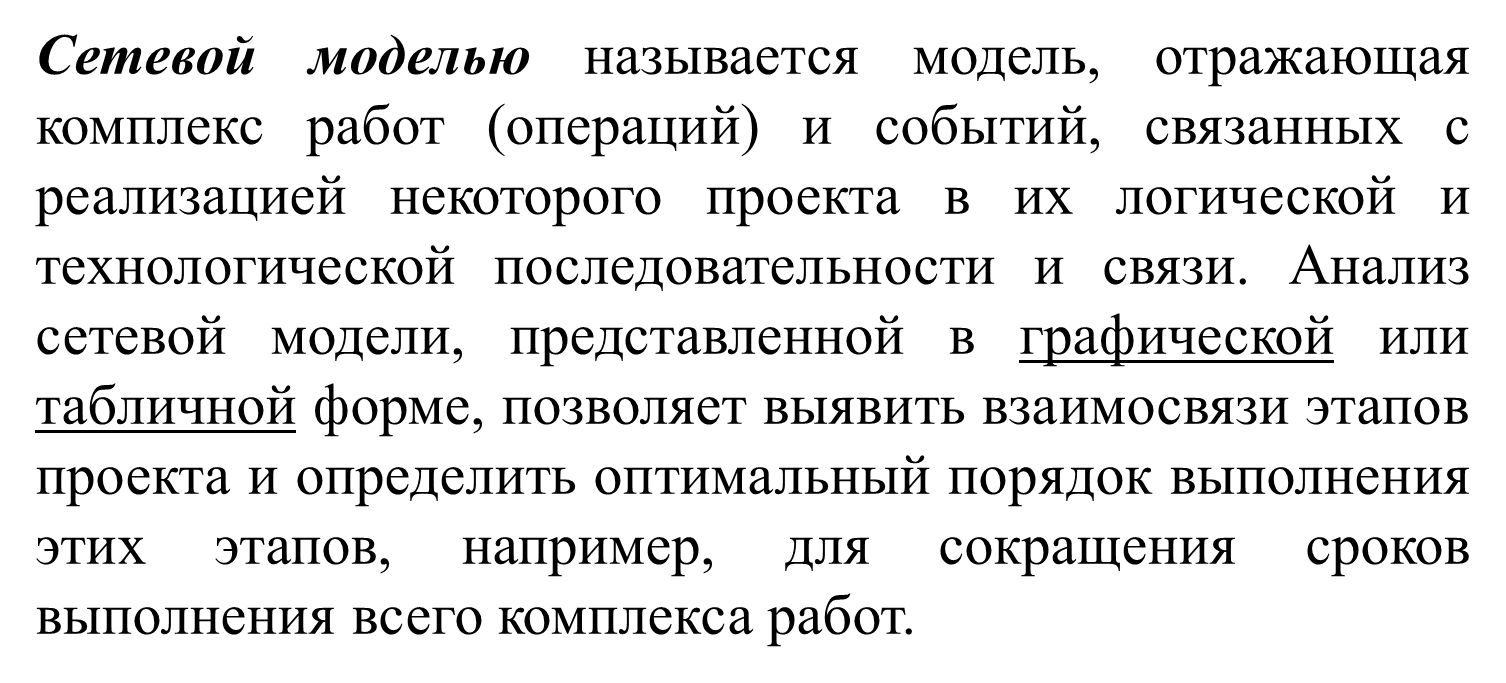
Также есть вектор Р, исходя из которого можно построить кратчайшие маршруты. По количеству элементов этот вектор равен количеству вершин в графе, Каждый элемент содержит последнюю промежуточную вершину на кратчайшем пути между вершиной-источником и конечной вершиной. В начале алгоритма все элементы вектора Р равны вершине источнику (в нашем случае Р = {1, 1, 1, 1, 1}). Далее на этапе пересчета значения метки для рассматриваемой вершины, в случае если метка рассматриваемой вершины меняется на меньшую, в массив Р мы записываем значение текущей вершины W. Например: у 3-ей вершины была метка со значением «30», при W=1. Далее при W=5, метка 3-ей вершины изменилась на «20», следовательно мы запишем значение в вектор Р — Р[3]=5. Также при W=5 изменилось значение метки у 4-й вершины (было «50», стало «40»), значит нужно присвоить 4-му элементу вектора Р значение W — P[4]=5. В результате получим вектор Р = {1, 1, 5, 5, 1}.Зная что в каждом элементе вектора Р записана последняя промежуточная вершина на пути между источником и конечной вершиной, мы можем получить и сам кратчайший маршрут.

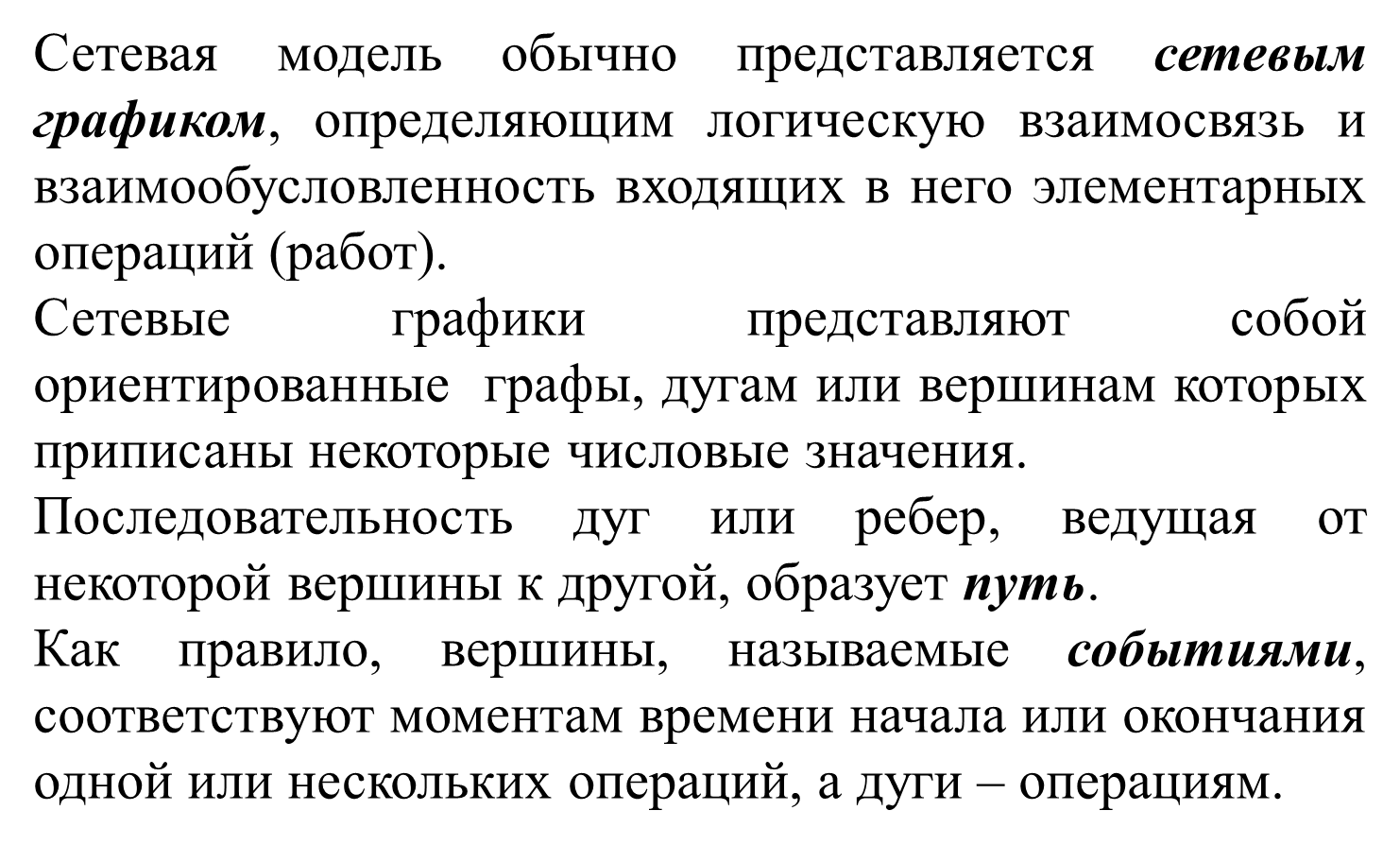
### **17. Математические основы сетевого планирования. Максимальные пути между вершинами графа.*Р– список предшествующих вершин.Максимальным путем называется путь с максимальным весом*** . В общем случае, в одном графе может быть несколько максимальных путей. Построение максимального пути во взвешенном ориентированном графе G=(V,E) возможно, если в нем нет контуров с положительным весом. Если в графе есть такой контур, то некоторые пути могут иметь сколь угодно большой вес, т.к. каждый обход контура увеличивает вес пути на величину веса этого контура. Будем предполагать, что граф G=(V,E) бесконтурный и его вершины пронумерованы так, что у любой дуги конечная вершина всегда имеет больший номер, чем начальная:i<j . Выше рассматривалась процедура ранжирования, основанная на алгоритме Демукрона, позволяющая привести любой безконтурный граф такому виду. Пусть все вершины графа G=(V,E) имеют номера от 1 до n=|V| . Рассматриваемый алгоритм основывается на рекуррентном выражении , где – множество начальных вершин дуг, входящих в вершину . При этом предполагается, что для всех вершин ранга 0 значение d[u]=0.



Вычисление значений d[v] необходимо выполнить для каждой вершины графа в порядке возрастания номера. Каждое полученное значение d[v] представляет собой максимальный вес пути в графе G=(V ,E) с конечной вершиной .Цикл вычисления d[v] сопровождается построением массива элементов p[v], которые формируются по тому же принципу, что и в алгоритме Дейкстры. Каждый элемент p[v] соответствует вершине графа . Значение p[v] – вершина (или ее номер), предшествующая вершине в пути максимального веса с конечной вершиной v. На рис, а изображен заданный граф и проинициализированные массивы d[v] и p[v]. На рис, е представлен результат работы алгоритма. Для построения максимального пути в графе необходимо найти максимальный элемент в d[v] и обратным порядком построить все предшествующие вершины по массиву p[v] (в нашем случае – это вершины: 5, 4, 2, 1).

### ***18. Сетевые модели. Применение сетевых моделей. Сетевые графики.***

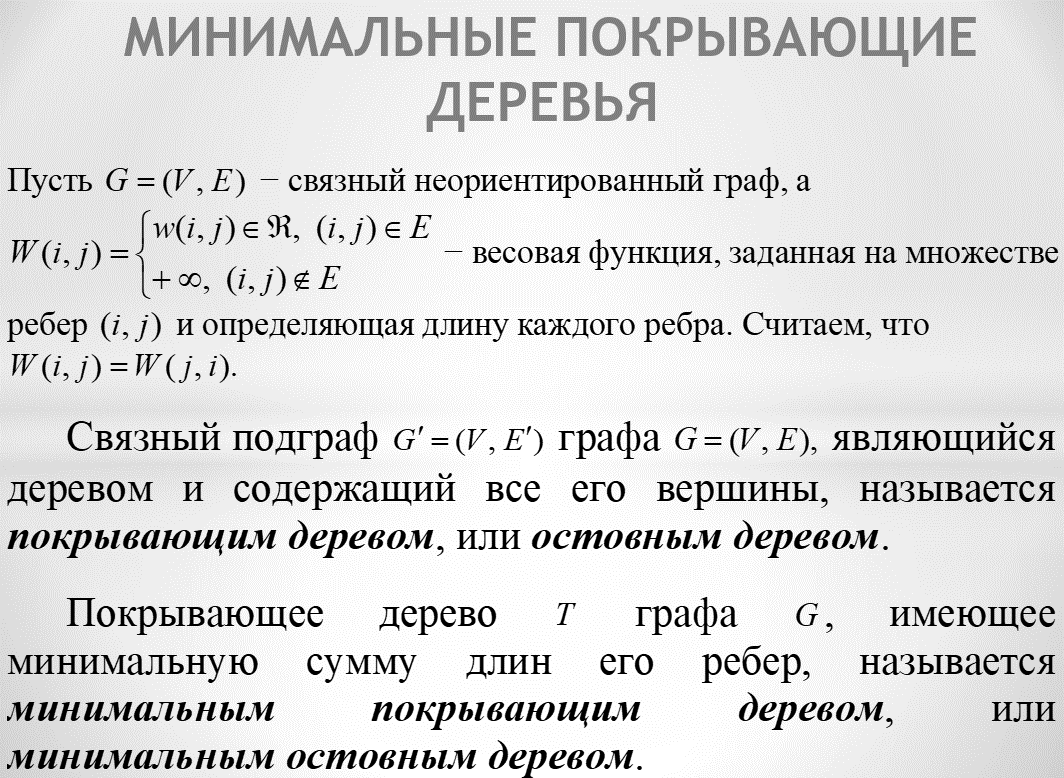




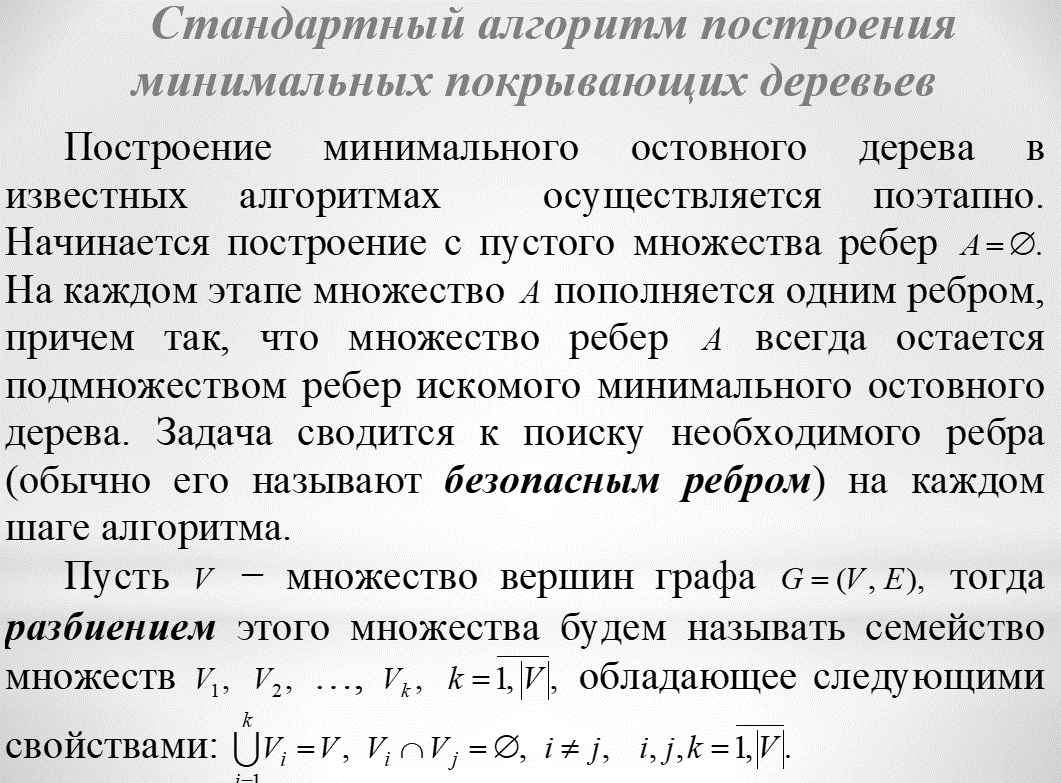
Про применение: В практике управления сложными системами широко применяются методы сетевого планирования и управления (СПУ). **Основные разновидности этих методов:** 1.*метод критического пути* (Critical Path Method - СРМ);2.*метод оценки и обзора программ* (Program Evaluation and Review Technique - PERT). **СПУ включает три основных этапа:**1.*Структурное планирование;*2.*Календарное планирование;*3.*Оперативное управление.*

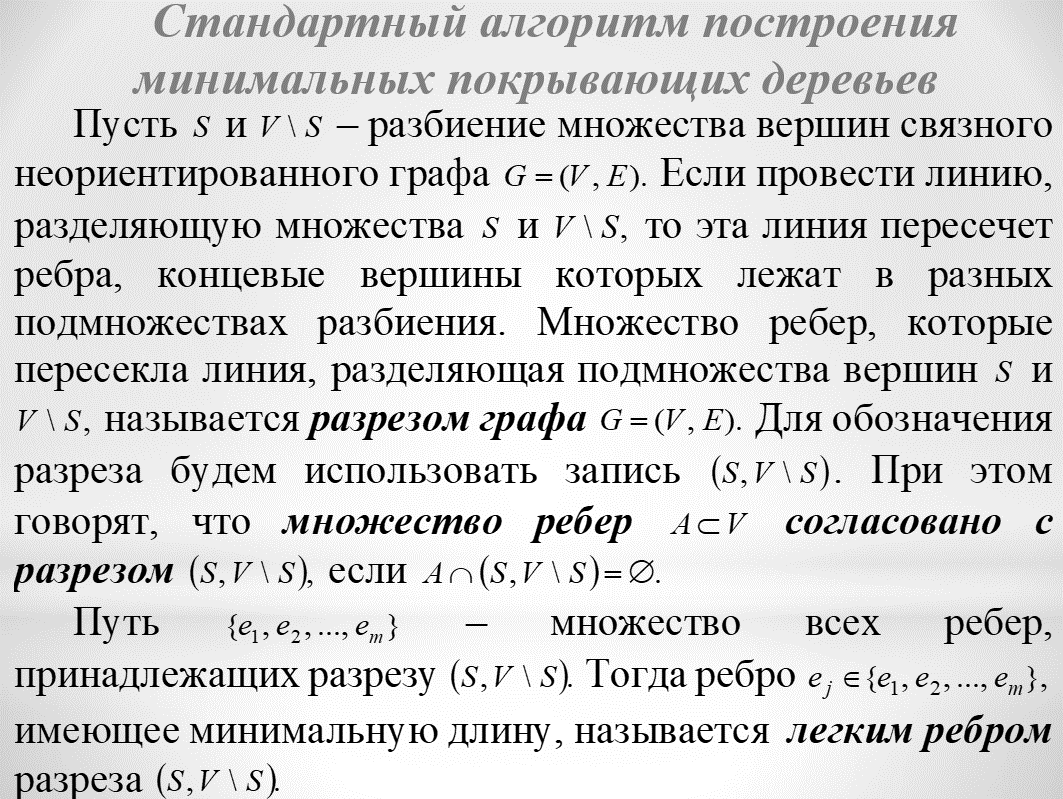
Структурное планирование начинается с разбиения проекта на четко определенные операции. Затем строится *сетевой график*, который представляет взаимосвязи работ проекта. Это позволяет детально анализировать все работы и вносить улучшения в структуру проекта еще до начала его реализации.Календарное планирование предусматривает построение календарного графика, определяющего моменты начала и окончания каждой работы и другие временные характеристики сетевого графика. Это позволяет, в частности, выявлять критические операции, которым необходимо уделять особое внимание, чтобы закончить проект в директивный срок. Во время календарного планирования определяются временные характеристики всех работ с целью *оптимизации сетевой модели*, которая улучшает эффективность использования какого-либо ресурса. В ходе оперативного управления используются *сетевой и календарный графики* для составления периодических отчетов о ходе выполнения проекта. При этом сетевая модель может подвергаться оперативной корректировке, вследствие чего будет разрабатываться новый календарный план остальной части проекта.

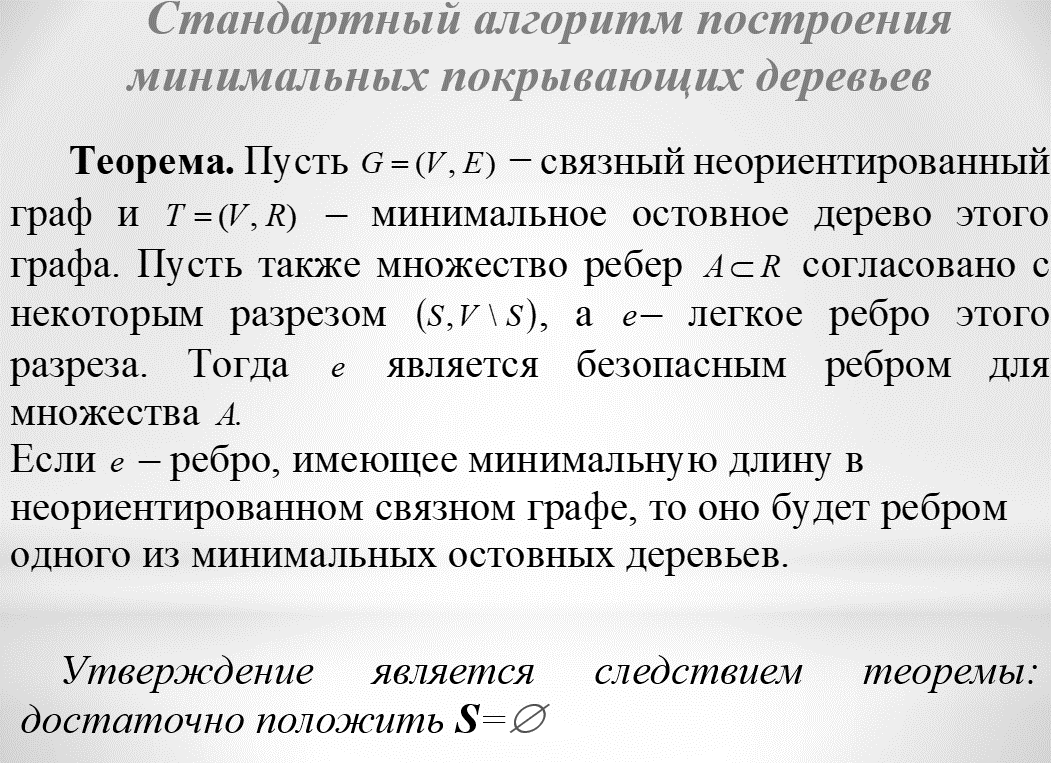
### **19. Минимальные покрывающие деревья. Основные алгоритмы нахождения минимального остовного дерева.**



Существует три алгоритма минимального остовного дерева. **1.Стандартный алгоритм построения минимального покрывающего дерева:** Если простыми словами, то вначале мы находим ребро с минимальной длиной. Затем “отрезаем” две вершины вместе с этим ребром, которое их соединяет. Ищем среди соседних вершин рёбра с минимальной длиной, включаем их в наше дерево и также отрезаем.

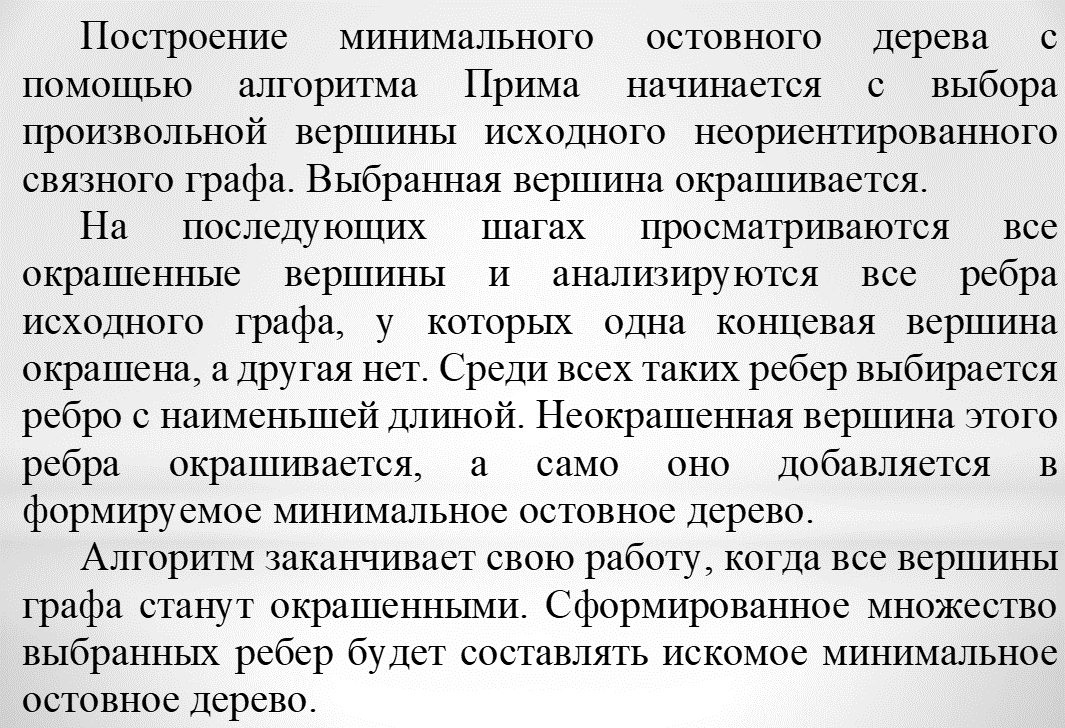






**2.Алгоритм Крускала.**Вначале находим ребро с наименьшей длиной, соединяем им две вершины. Затем снова находим минимальное ребро, но такое, чтобы концевые его вершины находились на изолированных друг от друга подграфах. После этого подграфы становятся единой компонентой для алгоритма графа.

**3.Алгоритм Прима**

****

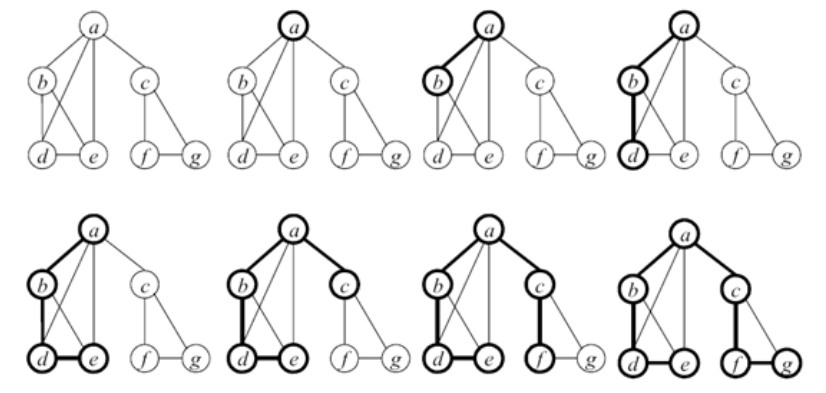
### **20. Оптимизационные алгоритмы на графах. Алгоритм поиска в ширину.**Многие алгоритмы на графах могут быть решены путем систематического перебора (обхода) вершин и дуг графа. Такой обход можно выполнить многими способами. На практике широкое распространение получили два способа, получивших специальные названия: ***поиск в ширину (BFS [Breadth-first search])*** и ***поиск в глубину (DFS [Depth-first search])***. Алгоритм поиска в ширину: 1.Считается, что у каждой дуги графа есть вес, равный единице.2.Выбирается нулевая стартовая вершина. 3.Подсоединяются ближайшие вершины: сначала те, которые находятся на расстоянии 1 от стартовой [смежные], потом на расстоянии 2 от нее и т.д.4.Логично, что будут посещены только те вершины, которые связаны со стартовой некоторой последовательностью дуг. 5.При окончании алгоритма проверяется полнота обхода и если есть неприсоединенные вершины, выбирается любая из них и алгоритм повторяется снова. Для алгоритма используется 4 основных массива: **Q** – очередь вершин: очередь, в которую записываются те вершины, которые сейчас серые

**С** – массив окраски вершин: W/G/B - текущий цвет вершины

**D** – массив расстояний: расстояние от стартовой вершины (для стартовой - 0, символ I - бесконечность, т.е. еще не присоединена) **P** – массив предшествующих вершин: сначала всем задается N - пустота, после для каждой вершины записывается предыдущая. В конце алгоритма строится BFS-дерево, множество вершин которого является подмножеством вершин исходного графа, связанных дугами в порядке их посещения (в соответствии с массивом **P**), а корнем – стартовая вершина.

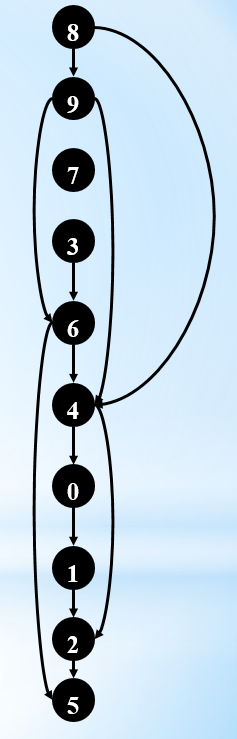
### **21. Оптимизационные алгоритмы на графах. Алгоритм поиска в глубину.** Алгоритм поиска в глубину: Задана стартовая вершина, и для каждой не пройденной вершины, начиная со стартовой, необходимо найти все смежные вершины и рекурсивно повторить поиск для каждой.

То бишь, мы посещаем стартовую вершину, потом все смежные ей, потом смотрим на все вершины, смежные стартовой, и для каждой из них опять проходимся по смежным вершинам, и так далее как бы заходим вглубь графа, пока не наткнемся на лист (без смежных вершин). Тогда при обнаружении листа возвращаемся обратно туда, откуда начинали, и рассматриваем следующую смежную ветку.



Для алгоритма используется 4 основных массива: **С** – массив окраски вершин: W/G/B - текущий цвет вершины (остался от BFS) **D** – шаг обнаружения вершины: номер шага, на котором она стала серой **P** – массив предшествующих вершин: сначала всем задается N - пустота, после для каждой вершины записывается предыдущая (такой же как в BFS) **F** – шаг фиксации вершины: номер окраски в черный цвет.Используется переменная ***t*** – шаг алгоритма. В конце также строим DFS-дерево по массиву **P.**

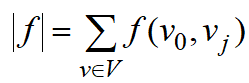
### **22. Оптимизационные алгоритмы на графах. Топологическая сортировка. Топологическая сортировка −** это процедура упорядочивания вершин ориентированного графа, не имеющего циклов (ациклического графа). В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху вниз, то дуги будут направлены только от верхних вершин к нижним. Обычно после выполнения топологической сортировки вершины нумеруются в соответствии с полученным порядком. После такого нумерования граф обладает свойством: начальная вершина каждой дуги имеет номер (имя) меньший, чем номер конечной вершины этой дуги. Наиболее известны два способа топологической сортировки графа: алгоритмы Демукрона и алгоритм, применяющий поиск в глубину. **Топологическая сортировка графа с помощью алгоритма поиска в глубину.**Запустить алгоритм DFS, при выходе из вершины добавляя вершину в конец списка с ответом. После окончания алгоритма список с ответом развернуть в противоположном порядке.Пример графа после топ. сортировки:



### **23. Потоки в сетях. Задача о максимальном потоке. Теорема Форда-Фалкерсона.** ***Сеть*** – это ориентированный граф G=(V,E) , каждому ребру которого поставлено в соответствие число , называемое пропускной способностью, а также выделено две вершины - исток и - сток,n=|V| . ***Поток*** – это функция обладающая тремя свойствами:1. ;2. (кососимметричность);3. ***Величина потока*** f это

***Разрез(S,T) сети*** G=(V,E) называется разбиение множества на две части S и T такое, что ***Пропускная способность разреза*** c(S,T) – это сумма пропускных способностей дуг соединяющих вершины в S и T. ***Минимальный разрез сети*** – это разрез с минимальной пропускной способностью. **Теорема Форда-Фалкерсона: *В любой сети максимальная величина потока из истока S в сток t равна минимальной пропускной способности разреза отделяющего S от t .***

### **24. Потоки в сетях. Задача о максимальном потоке. Алгоритм Форда-Фалкерсона.** ***Сеть*** – это ориентированный граф G=(V,E) , каждому ребру которого поставлено в соответствие число , называемое пропускной способностью, а также выделено две вершины - исток и - сток,n=|V| . ***Поток*** – это функция обладающая тремя свойствами:1. ;2. (кососимметричность);3.

***Величина потока***  f это 

***Разрез(S,T) сети*** G=(V,E) называется разбиение множества на две части S и T такое, что 

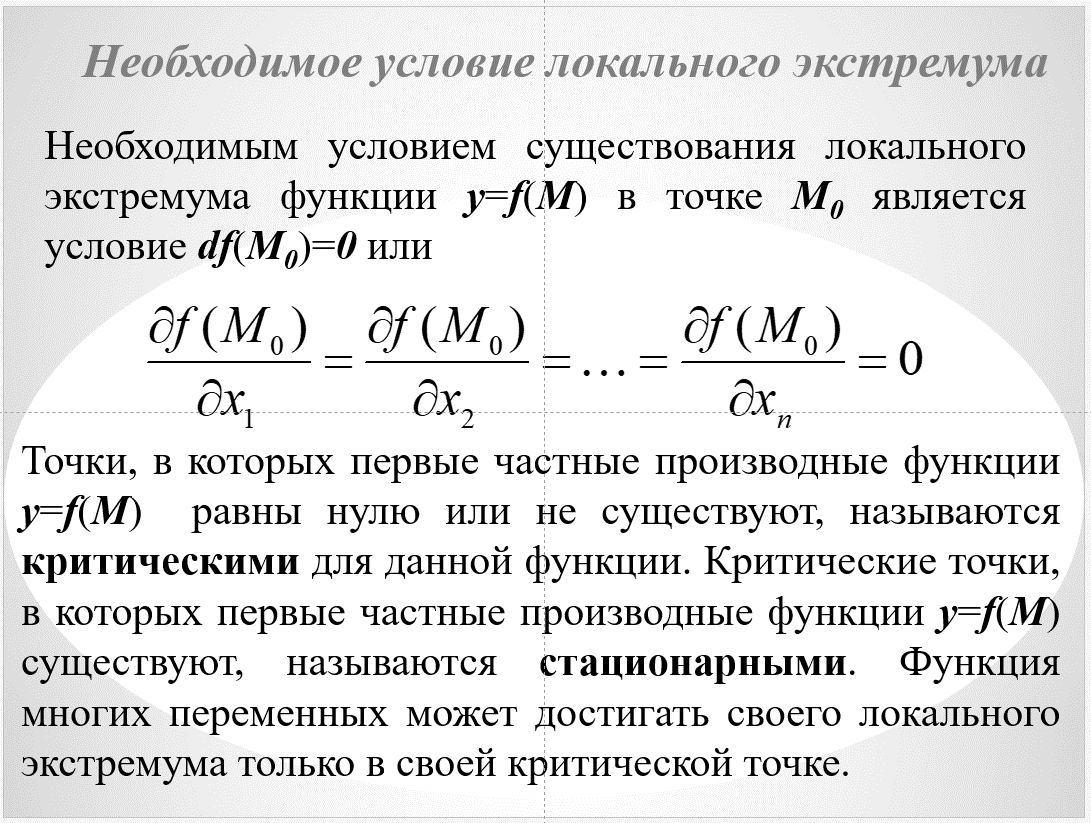
***Пропускная способность разреза*** c(S,T) – это сумма пропускных способностей дуг соединяющих вершины в S и T.

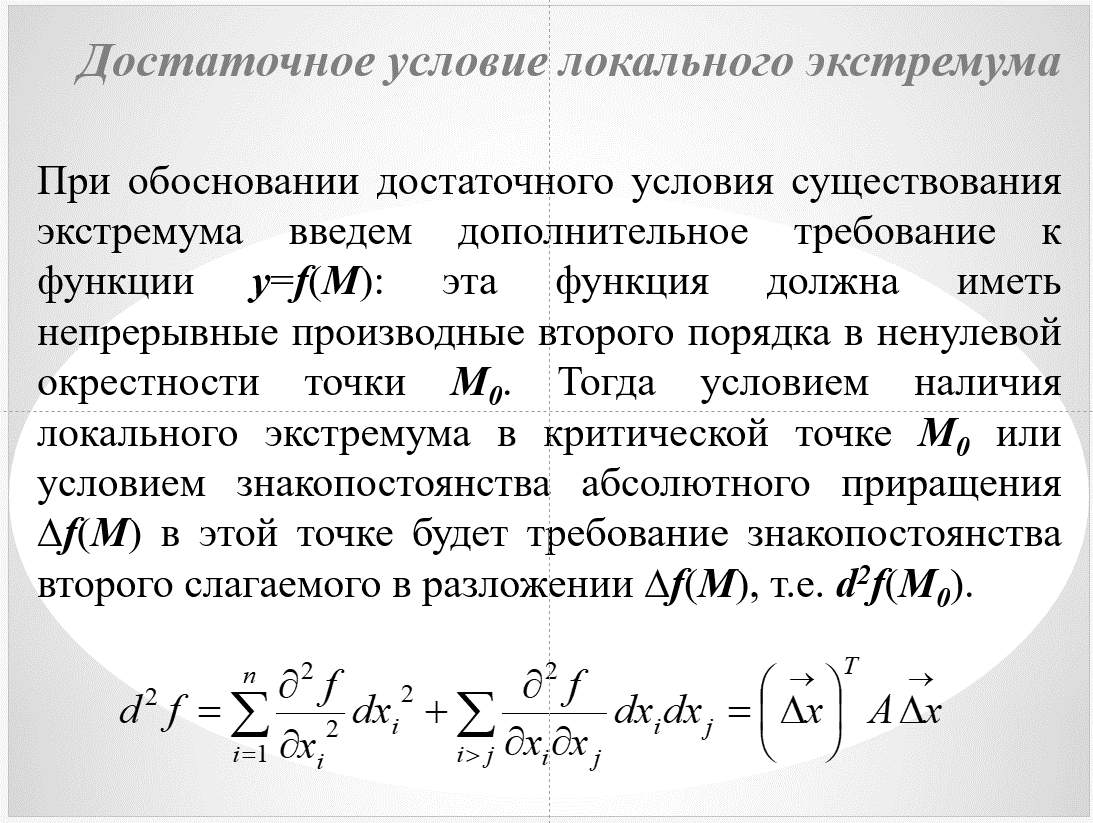
***Минимальный разрез сети*** – это разрез с минимальной пропускной способностью. **Теорема Форда-Фалкерсона: *В любой сети максимальная величина потока из истока S в сток t равна минимальной пропускной способности разреза отделяющего S от t .*** ***Алгоритм Форда-Фалкерсона:*** Остаточная сеть. Пусть дана сеть и поток в ней, тогда остаточная сеть состоит из тех ребер, поток по которым можно увеличить. При этом остаточное ребро не обязано быть ребром исходной сети. Такие (странные) ребра появляются, когда имеется поток в-ва в обратном направлении. У остальных сетей есть одно интересное св-во. Если в остаточной сети существует поток f, то прибавив его к исходящему потоку в сети мы получим также поток удовлетворяющий всем требованиям, но который больше исходящего. Назовем дополняющим путем простой путь из истока в сток в остаточной сети. Из определения остаточной сети вытекает, что по всем ребрам дополняющего пути можно переслать ещё сколько-то в-ва, не превысил их пропускную способность. Величину наибольшего потока, которого можно переслать по дополнительном пути, назовем остаточной пропускной способностью пути. Она равна значению min остаточного ребра, входящего в данный путь.

**Алгоритм форда-фалкерсона:** 1. Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной сетью.2. В остаточной сети находим любой путь из источника в сток. Если такого пути нет, останавливаемся.3. Пускаем через найденный путь (он называется увеличивающим путём или увеличивающей цепью) максимально возможный поток: **-**На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью cmin. **-**Для каждого ребра на найденном пути увеличиваем поток на cmin, а в противоположном ему - уменьшаем на cmin. **-**Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.4. Возвращаемся на шаг 2.

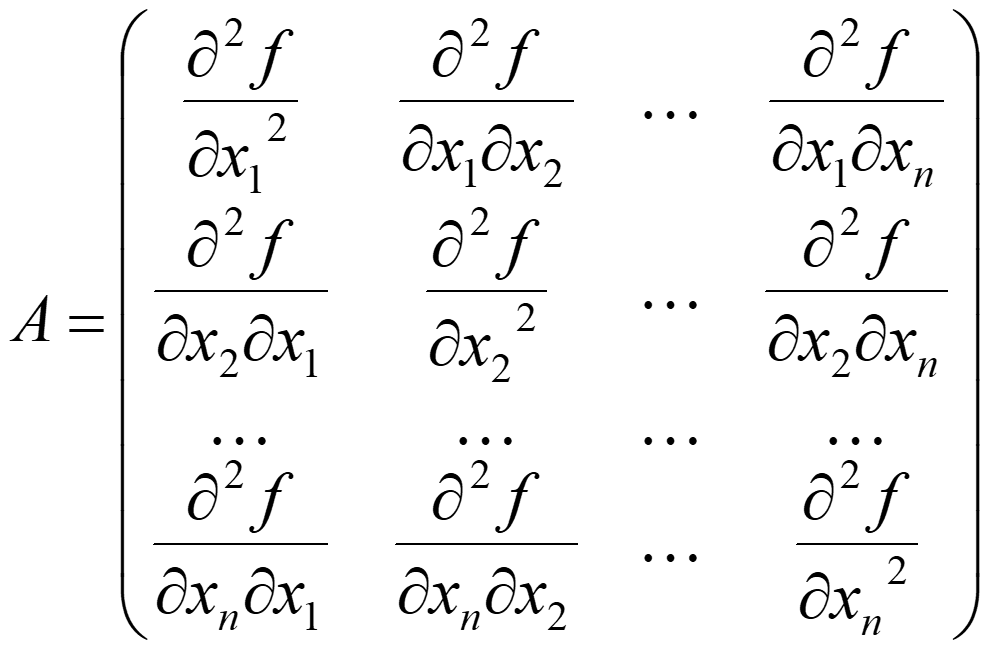
### **25. Задачи нелинейного программирования. Основные алгоритмы решения.Нелинейное программирование** – раздел математического программирования, изучающий методы решения экстремальных задач с нелинейной целевой функцией и (или) областью допустимых решений, определенной нелинейными ограничениями. К нелинейному программированию относят квадратичное, дробное, выпуклое, дискретное, целочисленное и геометрическое программирование.Среди большого числа вычислительных алгоритмов нелинейного программирования значительное место занимают:1.различные варианты градиентных методов (метод проекции градиента, метод условного градиента и т. п.); 2.методы штрафных функций;3.методы барьерных функций;4.метод модифицированных функций Лагранжа и др. В общем виде задачу нелинейного программирования можно сформулировать так: *f(M)→min (max)*, при условии *g(M)≤ 0*, где *M* – вектор искомых переменных; *f(M)* - целевая числовая функция; *g(M)* – вектор-функция системы ограничений.**При этом могут быть разные случаи: 1.**целевая функция – нелинейная, а ограничения – линейны; 2.целевая функция – линейная, а ограничения (хотя бы одно из них) – нелинейные;3.целевая функция и ограничения нелинейные. **Задачи условной оптимизации нелинейного программирования бывают двух типов: когда в ограничениях имеют место:** а) знаки равенства; б) знаки неравенства.

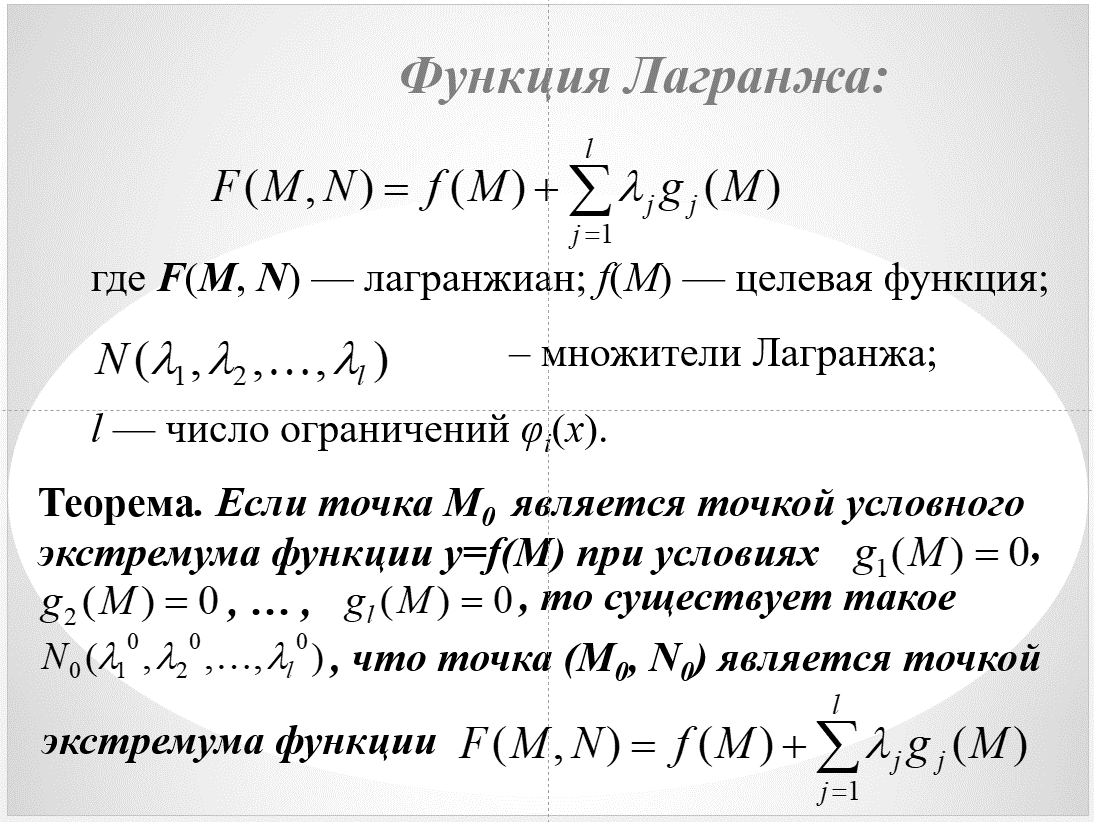
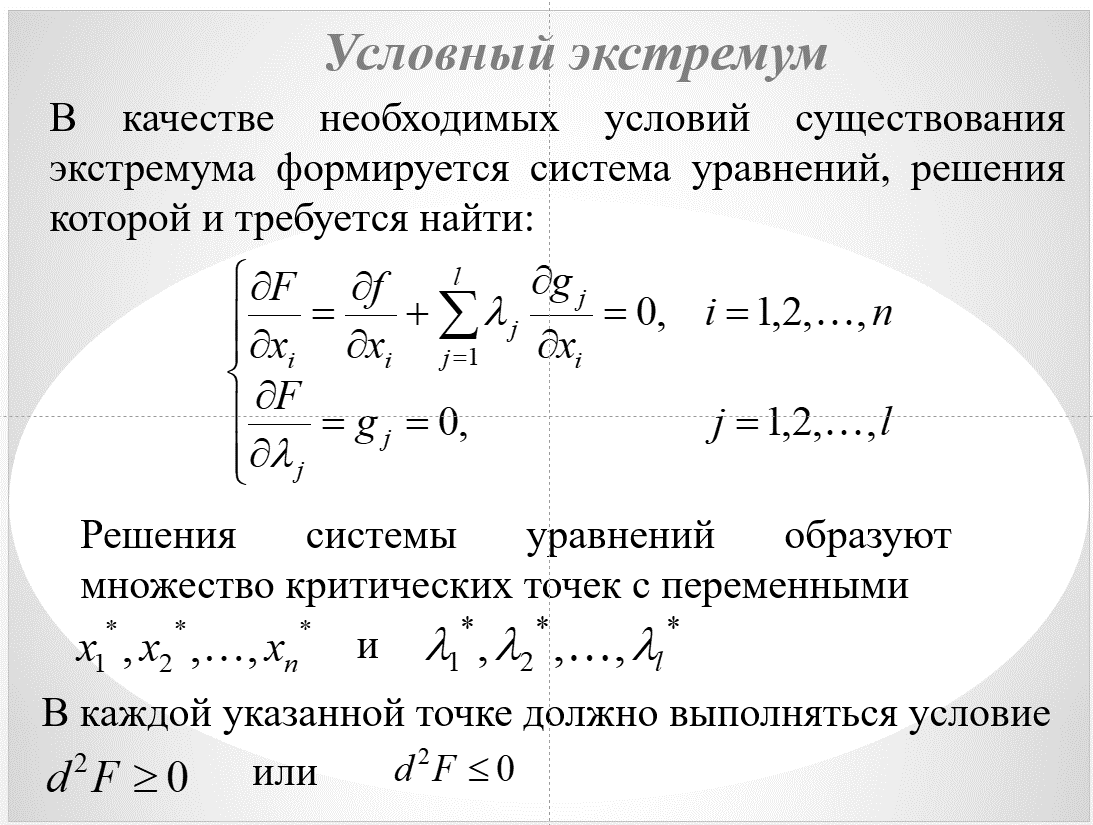
*Решение задачи нелинейного программирования* (поиск глобального минимума или максимума) состоит в отыскании таких значений переменных, подчиненных системе ограничений, при которых достигает минимума или максимума данная целевая функция. Наибольшее или наименьшее значение функции в данной области называется *абсолютным или глобальным экстремумом* функции (соответственно абсолютным максимумом или абсолютным минимумом) в этой области. **Теорема**. *Абсолютный (глобальный) экстремум функции в данной области достигается либо в критической точке функции, принадлежащей этой области, либо в граничной точке области*. Точки локального минимума или максимума называются точками *локального экстремума*. Для этих точек характерно знакопостоянство величины абсолютного приращения в пределах ненулевой окрестности.



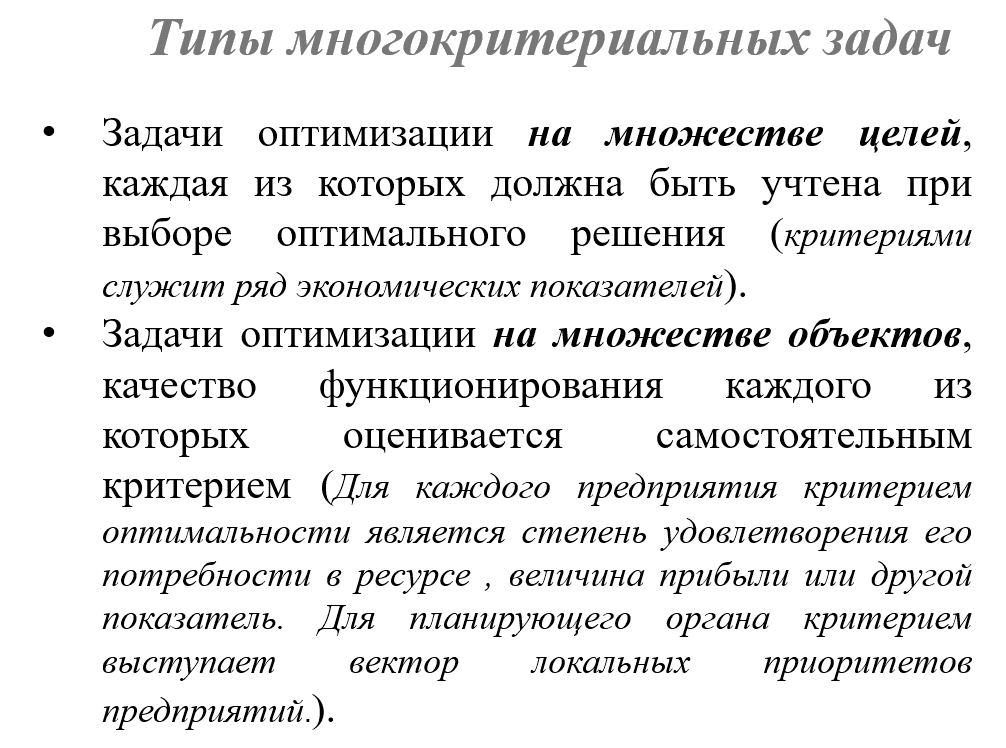
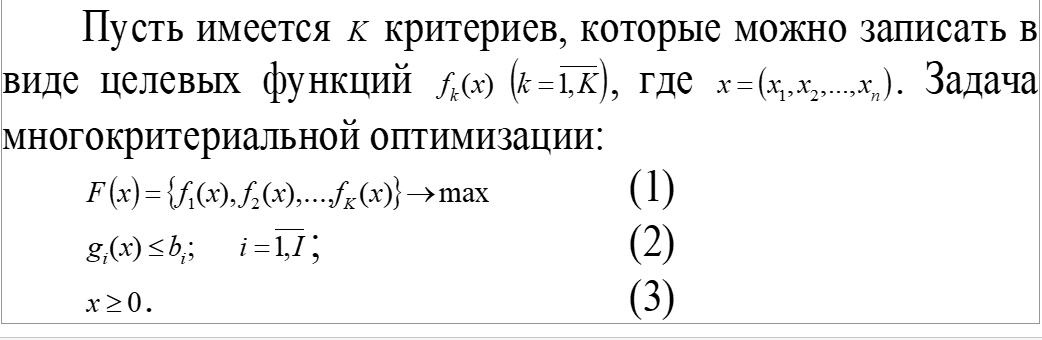


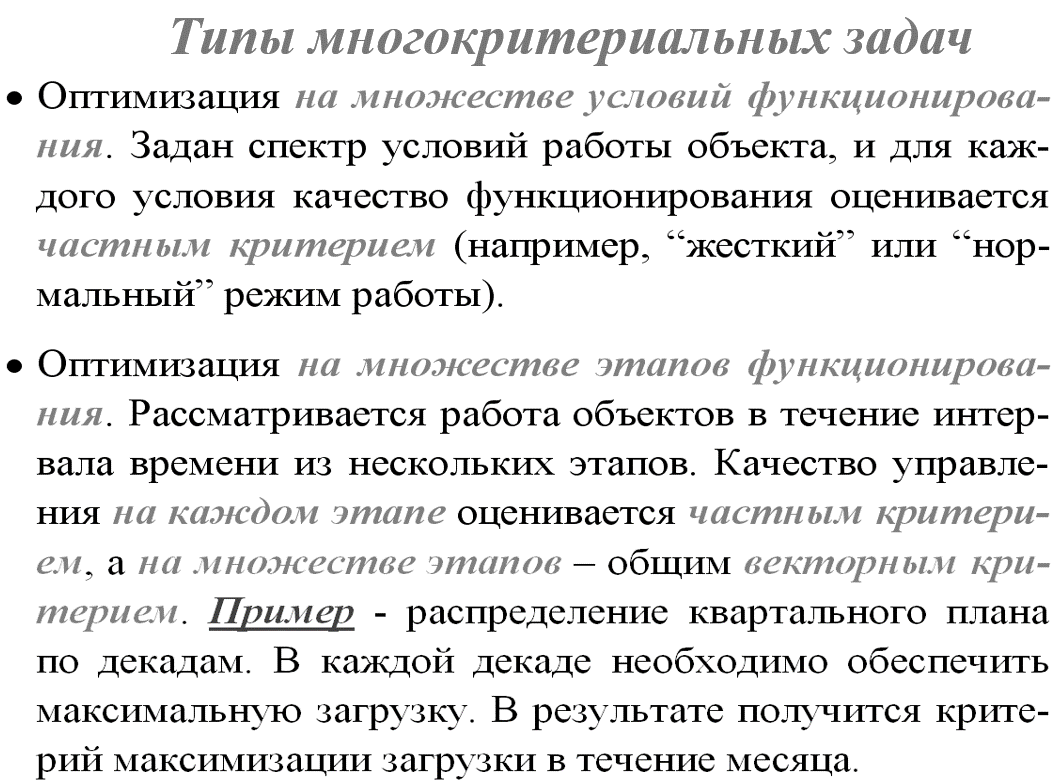
Основные алгоритмы решения:1)**матрица Гессе** (исп. в задачах оптимизации методом Ньютона)

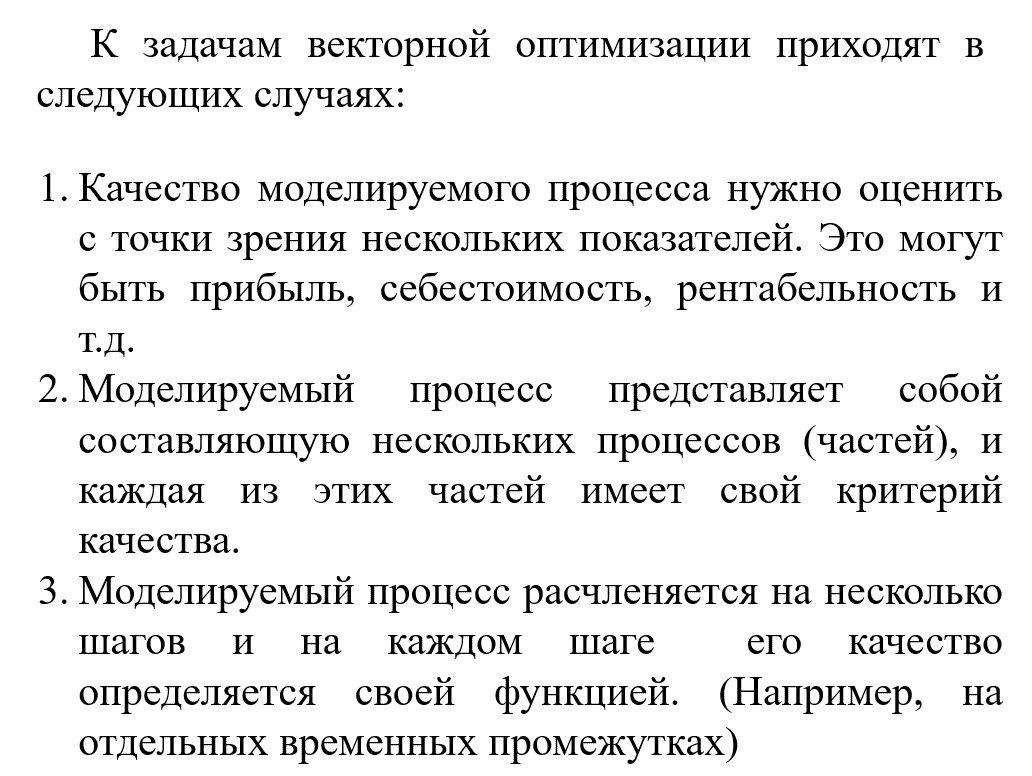


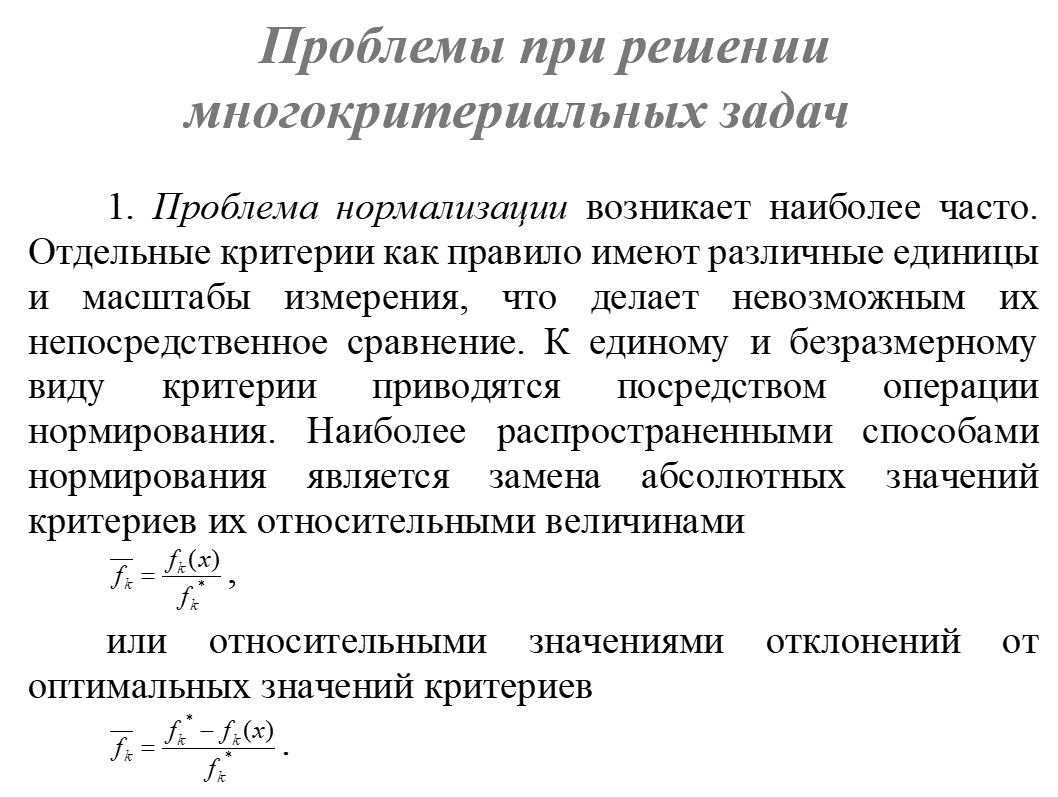
1. 
2. 

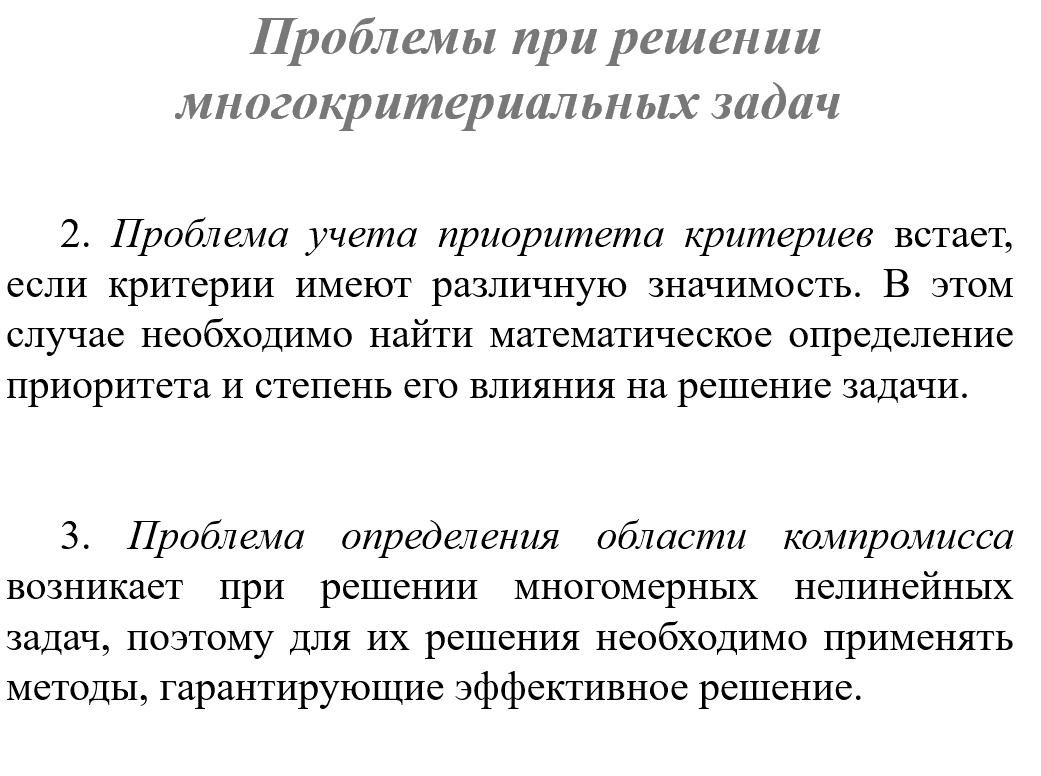
### **26. Постановка задачи векторной оптимизации.** Эффективность функционирования многих систем оценивается, как правило, несколькими критериями. Математической формой критерия эффективности в оптимизационных математических задачах является целевая функция.

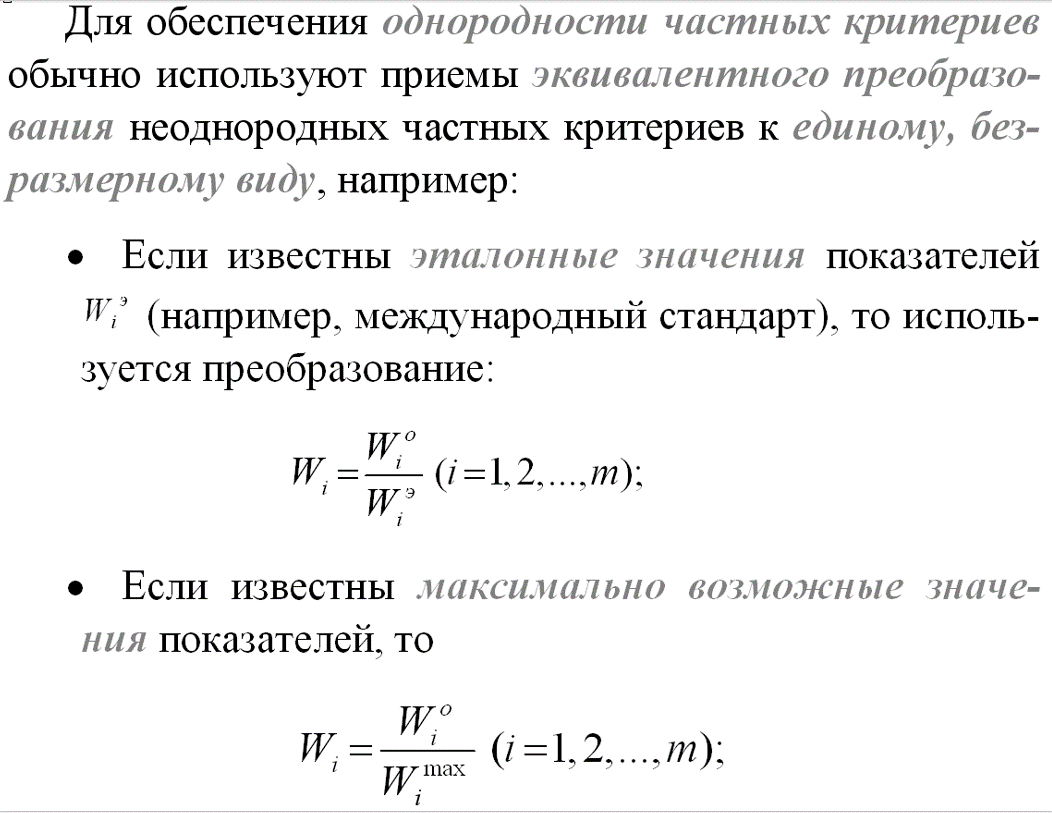


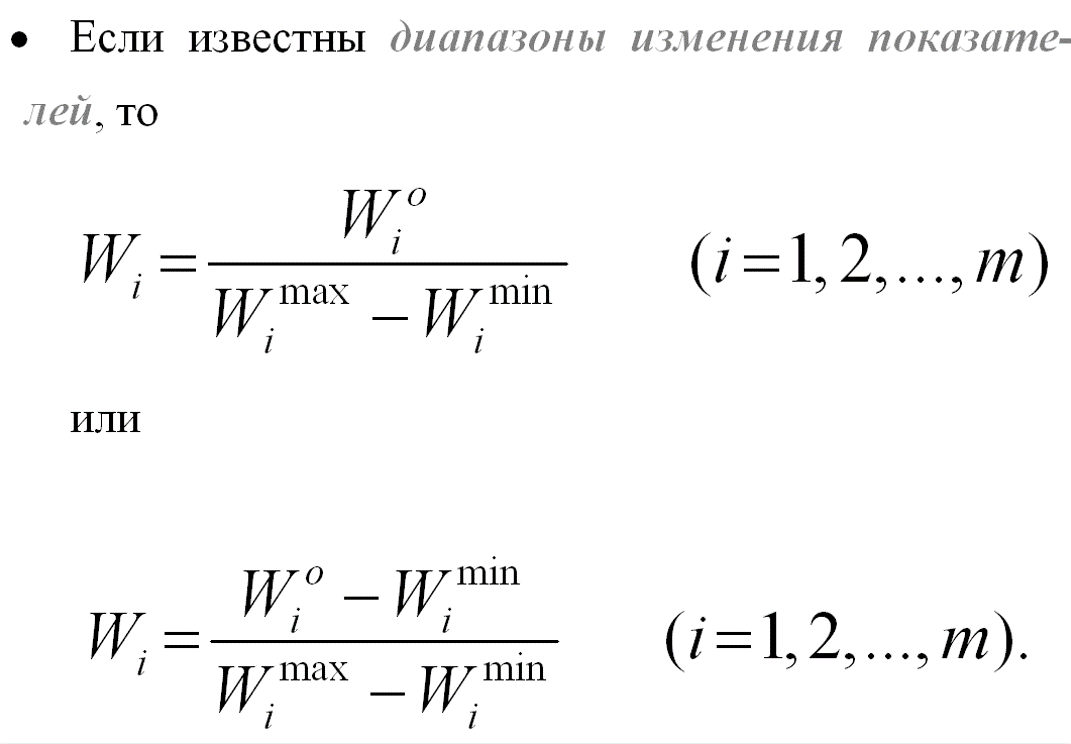






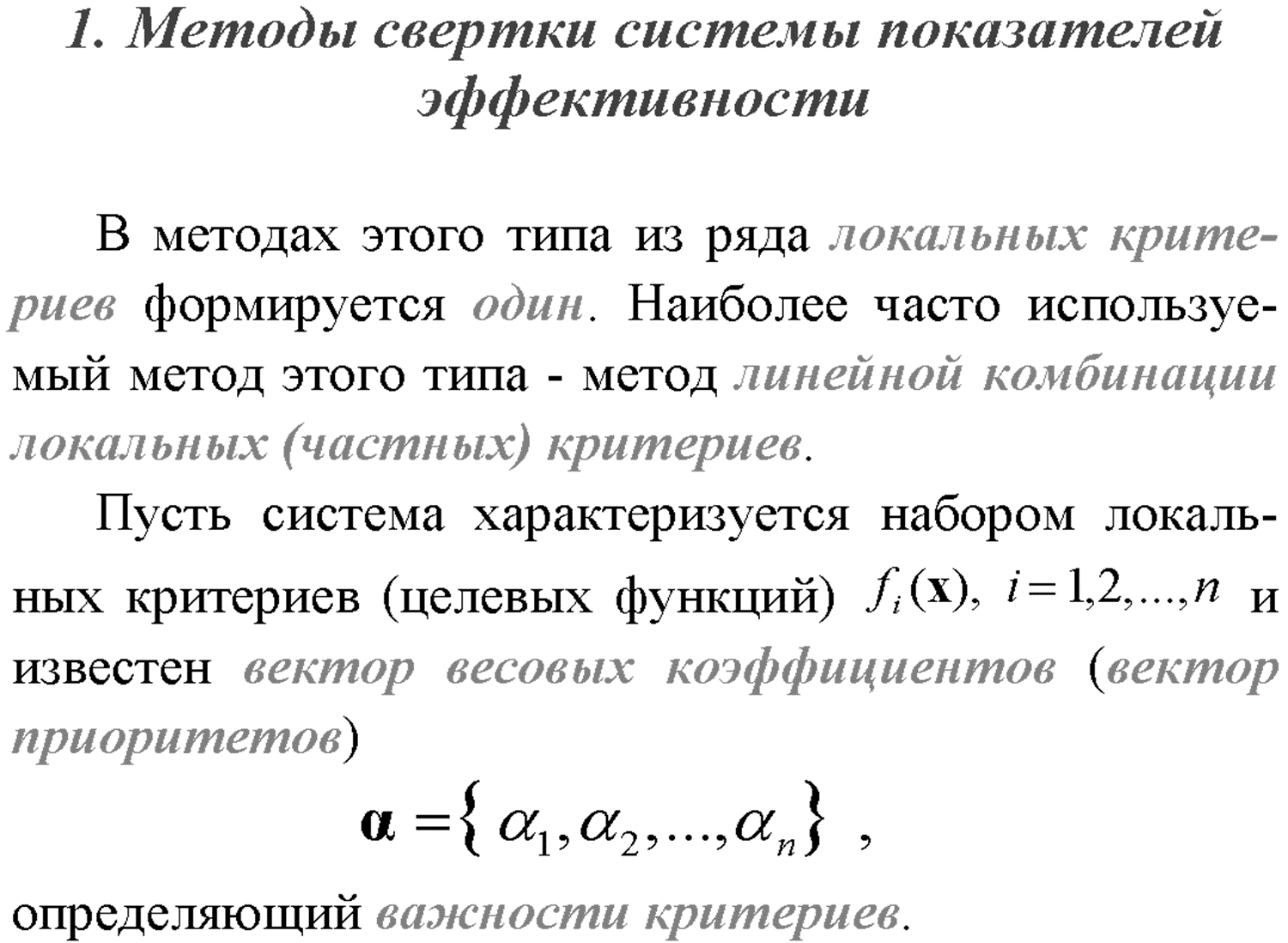


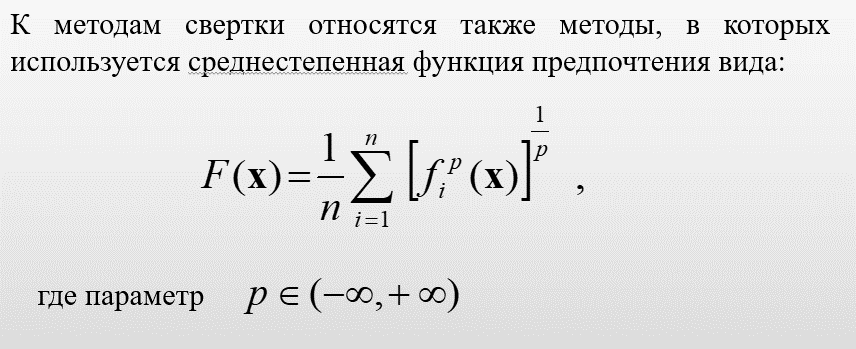




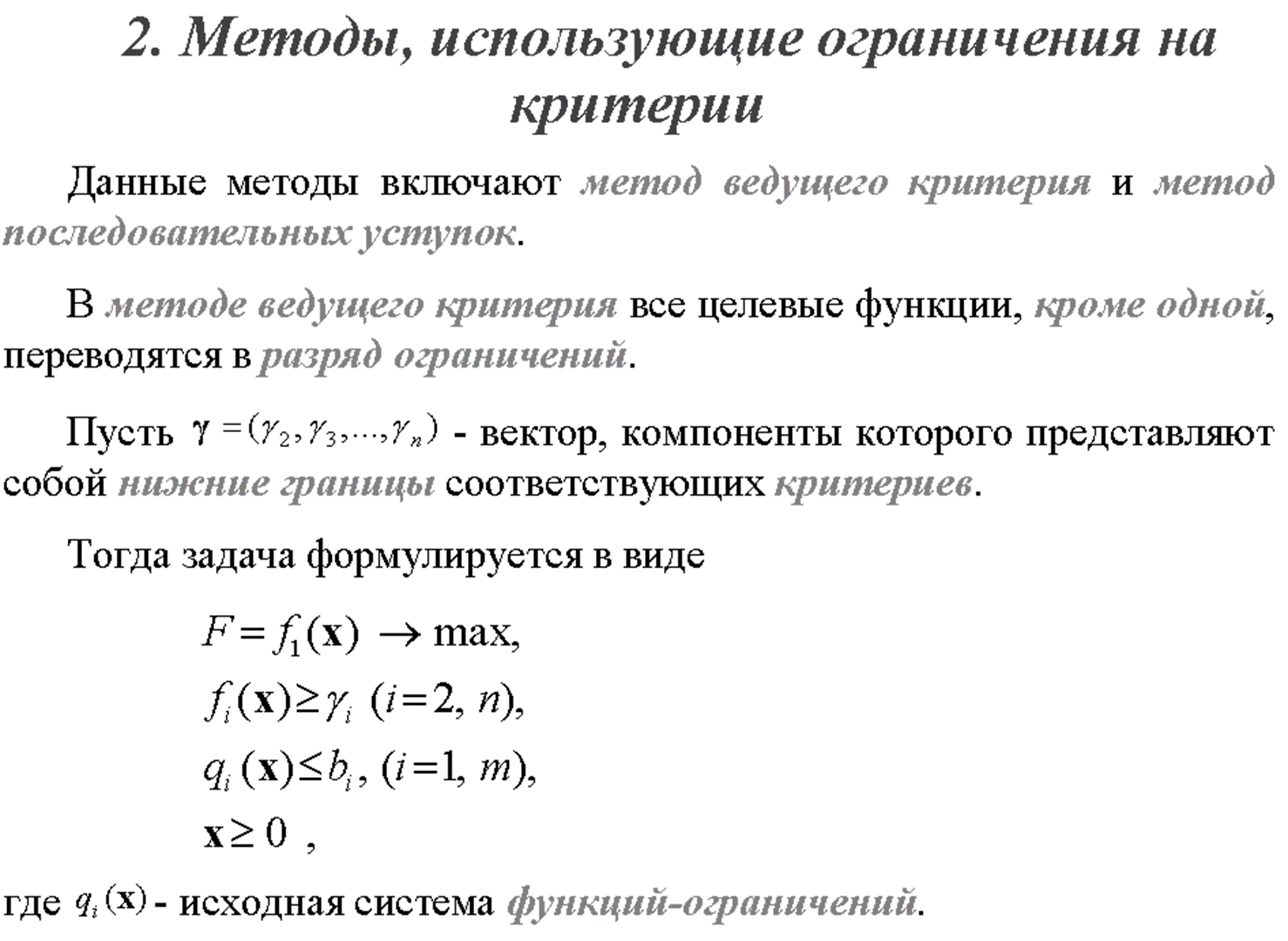
### 

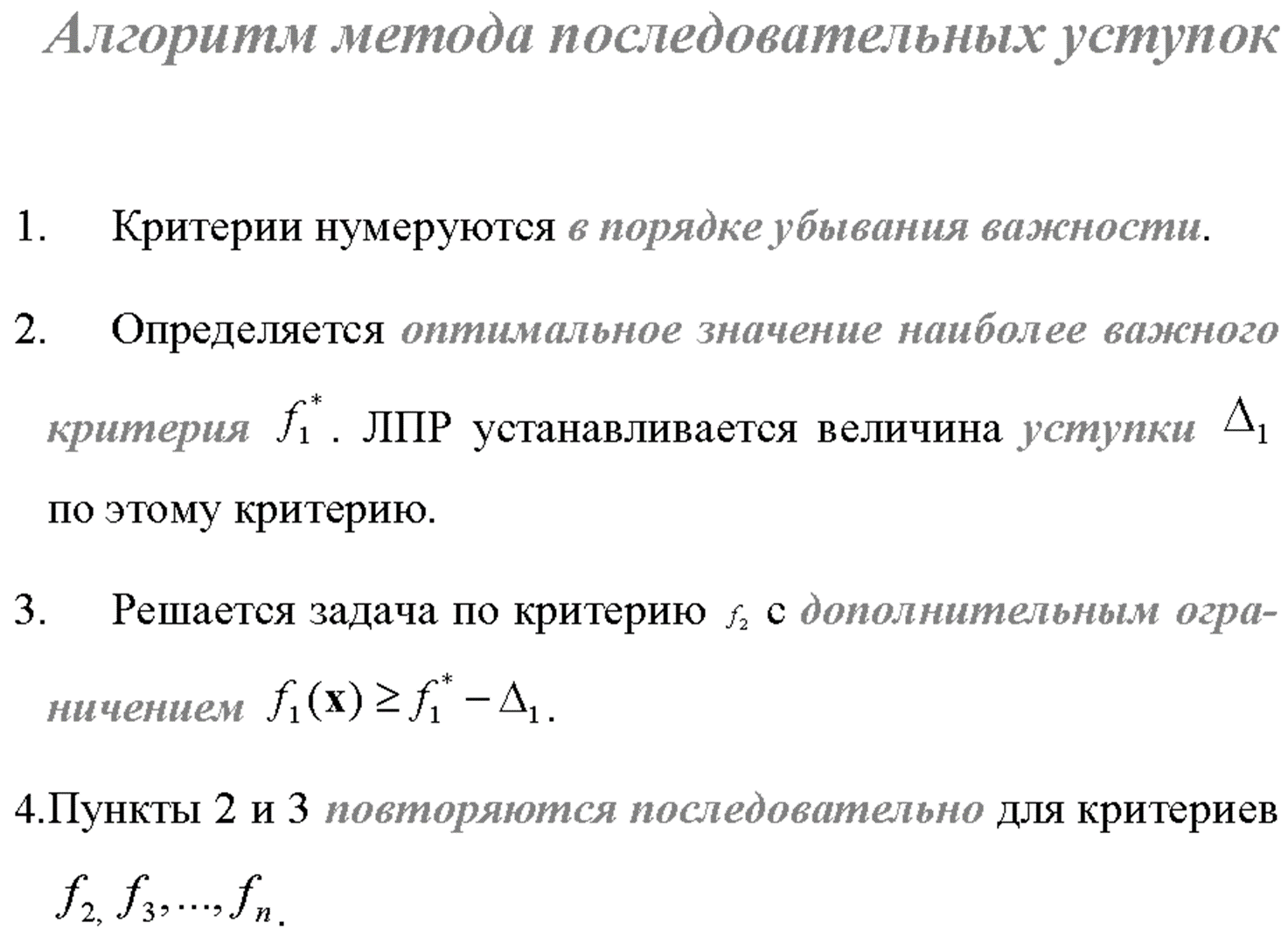
### **27. Методы решения задач векторной оптимизации.**

4 метода:**1.**Методы, основанные на свертывании системы показателей эффективности;**2.** Методы, использующие ограничения на критерии;**3.** Методы целевого программирования;**4.** Методы интерактивного программирования  
**1.** **Методы, основанные на свертывании системы показателей эффективности**

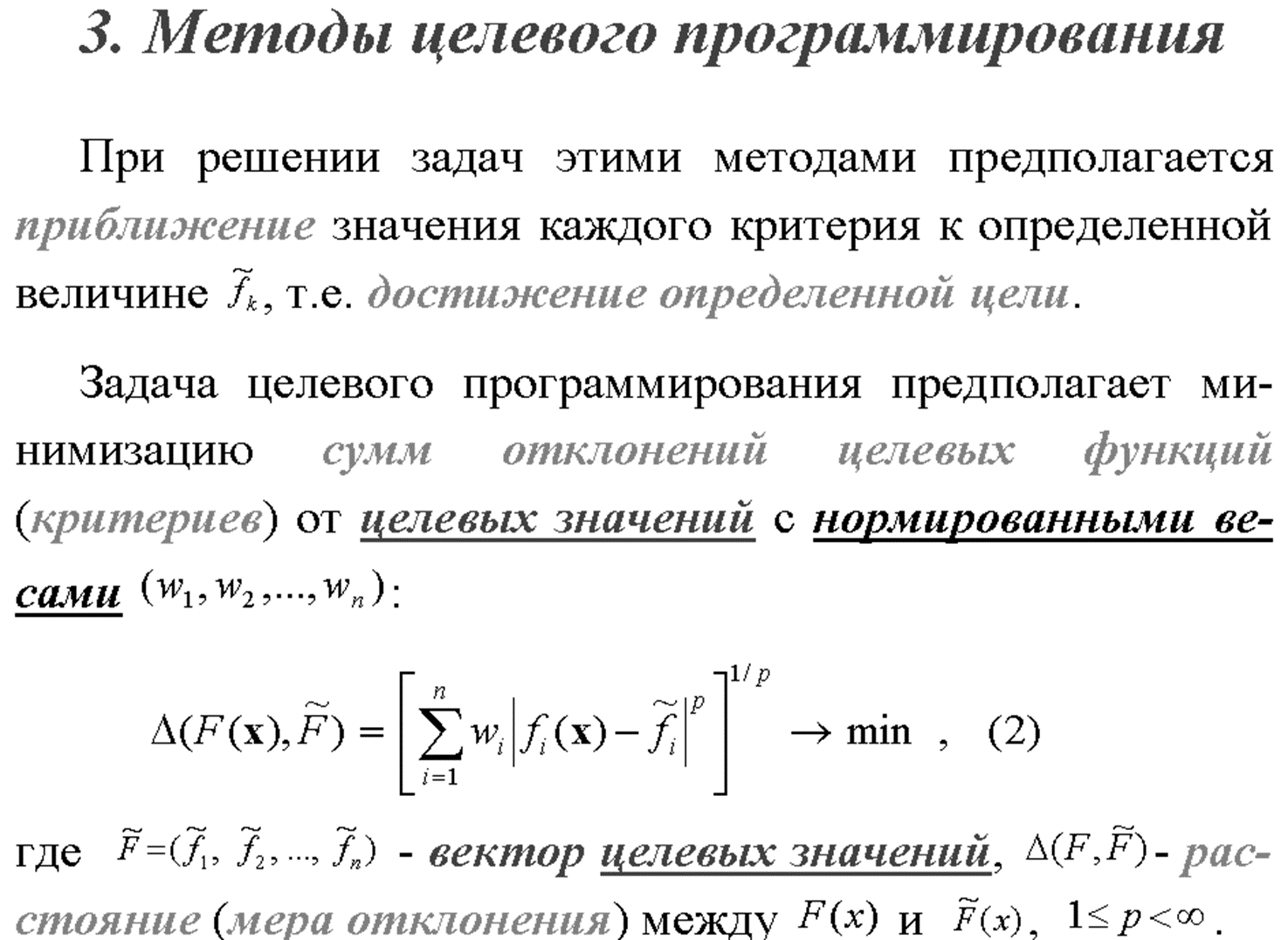


**2.Методы, использующие ограничения на критерии**

****

****

**3.Методы целевого программирования;**

****

1. Предмет изучения, структура, цели и задачи курса. Общее понятие задачи оптимизации. Классификация задач оптимизации. Обзор методов решения оптимизационных задач. Смежные дисциплины.
2. Общая формулировка задачи линейной оптимизации. Формы записи задач линейной оптимизации.
3. Геометрический метод решения задачи линейной оптимизации.
4. Симплекс-метод решения задачи линейной оптимизации. Алгоритмы нахождения опорного(базисного) и оптимального решений.
5. Транспортная задача. Математическая модель транспортной задачи. Алгоритм решения транспортной задачи. Методы построения исходного опорного решения.
6. Метод потенциалов нахождения оптимального решения транспортной задачи.
7. Общие принципы решения задач оптимизации методом ветвей и границ.
8. Комбинаторные методы решения оптимизационных задач. Генерация подмножеств заданного множества.
9. Комбинаторные методы решения оптимизационных задач. Генерация сочетаний.
10. Комбинаторные методы решения оптимизационных задач. Генерация перестановок.
11. Комбинаторные методы решения оптимизационных задач. Генерация размещений.
12. Динамическое программирование. Вычислительная схема решения задачи динамического программирования (на примере решения задачи о рюкзаке).
13. Основные приложения динамического программирования. Обзор задач, решаемых методами динамического программирования.
14. Рекурсивные алгоритмы.
15. Математические основы сетевого планирования. Основные понятия теории графов.
16. Математические основы сетевого планирования. Кратчайшие пути между вершинами графа.
17. Математические основы сетевого планирования. Максимальные пути между вершинами графа.
18. Сетевые модели. Применение сетевых моделей. Сетевые графики.
19. Минимальные покрывающие деревья. Основные алгоритмы нахождения минимального остовного дерева.
20. Оптимизационные алгоритмы на графах. Алгоритм поиска в ширину.
21. Оптимизационные алгоритмы на графах. Алгоритм поиска в глубину.
22. Оптимизационные алгоритмы на графах. Топологическая сортировка.
23. Потоки в сетях. Задача о максимальном потоке. Теорема Форда-Фалкерсона.
24. Потоки в сетях. Задача о максимальном потоке. Алгоритм Форда-Фалкерсона.
25. Задачи нелинейного программирования. Основные алгоритмы решения.
26. Постановка задачи векторной оптимизации.
27. Методы решения задач векторной оптимизации.