**Теоретическая часть**

1. *Предмет изучения, структура, цели и задачи курса. Общее понятие задачи оптимизации. Классификация задач оптимизации. Обзор методов решения оптимизационных задач. Смежные дисциплины.*

Математическое программирование - область математики, разрабатывающая теорию и численные методы решения многомерных экстремальных задач с ограничениями, т. е. задач на экстремум функции многих переменных с ограничениями на область изменения этих переменных.

Решение задачи математического программирования осуществляется в 4 этапа:

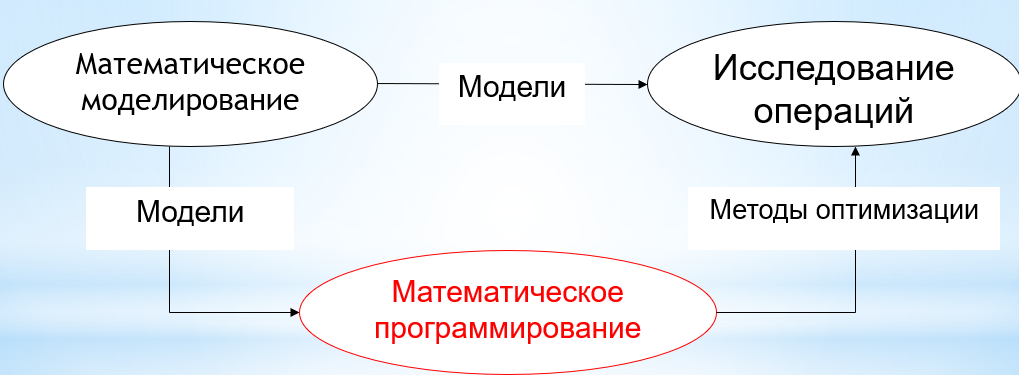
1. Построение математической модели
2. Классификация задачи
3. Выбор метода решения
4. Вычисление

Классификация задач оптимизации:

* Дискретное программирование
* Целочисленного программирования
* Линейное программирование
* Не линейное программирование

Метод решения задачи математического программирования зависит от исходных данных.

Вычисление решения задач мат. программирования. Осуществляется с помощью компьютера.



Общий вид задачи математического программирования:

*x* принадлежит *X*.

1. *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 функции при определённом наборе ограничений, налагаемых на аргументы.

Математическая модель любой задачи ЛП включает в себя:

* Переменные, которые следует определить
* Целевую функцию, подлежащую оптимизации
* Систему ограничений в форме линейных уравнений и неравенств

Общая форма записи задачи линейного программирования: необходимо найти экстремальное значение целевой функции:

C:\Documents and Settings\Максимка\Рабочий стол\image006.gif, далее идут ограничения…

1. *Геометрический метод решения задачи линейной оптимизации.*

Использование данного метода удобно для задачи с двумя переменными, при большем числе – необходимо применение алгебраического аппарата.

Графический метод состоит из двух этапов:

1. Построение пространства допустимых решений, удовлетворяющих всем ограничением 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 верстаемому значению этой функции пересекающей область допустимых решений.

Точка пересечения обладает допустим. значений и прямой, соответствующей максимально возможному значению целевой функции и будет точкой оптимума.

Оптимальное решение соответствует точке С, которая является местом пересечения прямых.

Координата точки С(*x1*; *x2*) находится, как решение системы уравнений, пересечением которых она образована.

1. *Симплекс-метод решения задачи линейной оптимизации. Алгоритмы нахождения опорного(базисного) и оптимального решений.*

Переход от геометрического способа решения к симплекс-методу лежит через алгебраическое описание крайних точек в пространстве решений. Для реализации этого подхода сначала требуется привести задачи ЛП к стандартной форме, преобразовав неравенство ограничений в равенство, путём введения дополнительных переменных.

Основное свойство симплекс-метода заключается в том, что решение задачи осуществляется интернационально. На каждой итерации алгоритм переходит к новой угловой точке, которая потенциально может улучшить значение целевой функции. И останавливается, когда дальнейшее улучшение невозможно.

Последовательность действий:

* Находим начальное допустимое базисное решение
* На основе условия оптимизации определяется вводимая переменная. Если вводимых переменных нет, то вычисления заканчиваются
* На основе условия допустимости выбирается исключаемая переменная
* Методом Рауса-Жордана вычисляем новое базисное решение и переходим к пункту 2

Определение базисных решений: Пусть ограничение задачи ЛП представляется в виде М равенств с N переменными, при этом М<N. Предположим, значение N-M переменных ровно нулю, а значение оставшихся М переменных найдём, как решение системы М уравнений. Если полученное решение получилось единственным, тогда эти М переменных называем базисными переменными, а оставшиеся N-M небазисными переменными. Значение базисных переменных называется базисным решением. Количество базисных решений не превосходит

1. *Транспортная задача. Математическая модель транспортной задачи. Алгоритм решения транспортной задачи. Методы построения исходного опорного решения.*

Транспортная задача – специальный класс задач линейного программирования. Они описывают перевозку какого-либо товара из пункта направления в пункт назначения.

Назначение транспортной задачи – определение объёма перевозок из пунктов направления в пункт назначения с минимальной суммарной стоимостью перевозок.

Математическая модель: *Xij , i=~~1,m~~ , j=~~1,n~~  -* решение задачи.

Целевая функция: – целевая функция.

Алгоритм решения транспортной задачи:

1. Построение начального базисного решения:

* Метод северо-западного угла
* Метод наименьшей стоимости
* Метод Фогеля

1. Метод потенциалов

Метод наименьшей стоимости:

Транспортная задача с *m* пунктами направления и с *n* пунктами назначения

Имеют *n+m* ограничений в виде равенств по одному на каждый пункт направления и назначения. Т. к. транспортная задача должна быть сбалансирована, то одно из этих равенств избыточно. Поэтому задача имеет *n+m+1* независимых ограничений и начальное базисное решение состоит из *n+m+1* базисных переменных.

Сначала в таблице находим ячейку с наименьшей стоимостью, затем переменной в этой ячейке присваивается наибольшее значение, допустимое ограничениями по спросу и предложению. Если таких ячеек несколько, то выбор произволен. Далее вычёркиваем соответствующий столбец или строка и корректируется спрос и предложения. Затем просматриваются не вычеркнутые ячейки и выбирается новая ячейка с min стоимостью.

1. *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* Потенциалы *Ui , i=~~1,3~~ j=~~1,4~~*

>> Определяем потенциалы для всех базисных переменных…

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

>> Определяем небазисные переменные для свободных клеток. Вычисляем по формуле *Ui +* *Vj* **-** *Cij*

>> Вводимой в базис переменной будет та, которая имеет наибольшее положительное значение

>> Обозначим через Т количество груза перевезённого через 2,2. Сначала 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).

>> Чтобы удовлетворять ограничениям по спросу и предложению надо поочерёдно отнимать и прибавлять Т к значению базисных переменных расположенных в угловых ячейках цикла.

>> Направление обхода не имеет значения.

>> Определяем допустимое решение…

>> Определяем значение функции цели…

>> Пункт 1….

1. *Общие принципы решения задач оптимизации методом ветвей и границ.*

В основе метода лежит две процедуры:

* процедура ветвления (**BR)** позволяет разбивать множество допустимых решений на не пересекаемые подмножества
* процедура вычисления (EV) нижней или верхней границы

Метод ветвей и границ сводится к построению корневого дерева **T**, которое пошагово строится по следующему алгоритму:

1. Построить корневой узел дерева и пометить парой , где  – нижняя граница, полученная с помощью процедуры **EV**.
2. Применить процедуру **BR** к области  и сформировать две подобласти  и  К каждой из подобластей применить процедуру **EV** и получить оценки  и 
3. Построить два узла дочерних для коневого узла и пометить их соответственно  и 
4. Выбрать в текущем дереве **T** такой лист, в метке которого значение нижней границы не превышает значения границы в остальных листах дерева **T**, и применить к подобласти, соответствующей выбранному листу, процедуру **BR**. К каждой из сформированных новых подобластей применим процедуру **EV** и получить их нижние границы. Построить два новых узла дерева по тому же принципу, что и в пункте 3.
5. Выполнить пункт 4 до тех пор, пока не будет получен лист, в метке которого самое малое значение нижней границы среди всех листов дерева, а соответствующая подобласть состоит из одного элемента  и не подлежит дальнейшему разбиению.
6. Найденное значение  является оптимальным решением.

Таким образом, ***метод ветвей и границ*** – это общий алгоритмический метод решения задач комбинаторной оптимизации. По существу, метод является вариацией полного перебора с отсевом подмножеств допустимых решений, заведомо не содержащих оптимальных решений. В основе метода лежат две процедуры: ***процедура ветвления***, позволяющая разбивать множество допустимых решений на непересекающиеся подмножества, и ***процедура*** ***вычисления*** нижней или верхней ***границы***.

1. *Комбинаторные методы решения оптимизационных задач. Генерация подмножеств заданного множества.*

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*

1. Комбинаторные методы решения оптимизационных задач. Генерация сочетаний.

Булеан2x можно рассматривать как объединение всевозможных сочетаний, построенных из элементов множества.

Потгому генерация множества Cx,m может быть сведена к генерации булеана *2х* и выбору из него всех подмножеств с мощностью *т.*

Количество таких подмножеств:

1. Комбинаторные методы решения оптимизационных задач. Генерация перестановок.

Наиболее известным методом построения множества PX является алгоритм Джонсона-Троттера. Каждый элемент множества X помечается стрелкой, которая направленна вправо или влево. В первой перестановке все стрелки направленны влево.

В алгоритме используется понятиемобильного элемента. Эле­мент *xi* последовательности элементов множества *X* называется мо­бильным, если соответствующая ему стрелка указывает на меньший соседний элемент.

Построение множества всех перестановок с помощью алгоритма Джонсона- Троттера сводится к следующей процедуре:

* + 1. Построить первую перестановку - последовательность всех элементов множества X, перечисленных в порядке возрастании 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 X - 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, и изменить их стрелки на противоположное направление.
    5. Перейти к шагу 2.

Количество перестановок: *Pn=n!*

1. Комбинаторные методы решения оптимизационных задач. Генерация размещений.

Генерация размещений осуществляется в три этапа:

1. Для множества  формируется множество всех сочетаний по три элемента. Таких сочетаний будет 
2. Для элементов каждого сочетания ,  генерируется все перестановки  Каждому сочетанию будет соответствовать  перестановок. Таким образом, всего будет построено  перестановок, которые в совокупности представляют собой все размещения множества 
3. Применяя полученные на предыдущем шаге размещения в качестве индексов для элементов множества  формируется множество  всех размещений по 3.

Для того чтобы проследить этапы генерации множества  на схеме (рис. 5.1), ее следует рассматривать слева направо. В крайней левой позиции изображено множество , на основе которого формируются четыре сочетания.

Затем каждое сочетание рассматривается как отдельное множество, состоящее из трех элементов. Для каждого множества формируется по  перестановок. В итоге в третьем слева столбце на схеме отображены  перестановки множества 

На последнем этапе формируется множество  всех перестановок элементов множества  Элементы множества  на рис. 5.1 отображены в крайнем справа столбце.

Количество таких подмножеств:

1. Динамическое программирование. Вычислительная схема решения задачи динамического программирования (на примере решения задачи о рюкзаке).

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***.

Количество таблиц, которое необходимо построить для решения поставленной задачи, соответствует заданному в задаче  – количеству типов предметов.

Все таблицы имеют одинаковую структуру и содержат по три столбца: неиспользованный объем места в рюкзаке, объем, занятый соответствующим типом, стоимость вещей в рюкзаке.

Например, первая строка таблицы T.0 соответствует случаю, когда в рюкзак не помещено ни одного предмета этого типа. Третья строка описывает случай, когда в пустой рюкзак помещено два предмета, которые заняли объем  Третий столбец третей стоки содержит стоимость двух предметов, равную  =  единиц.

Десятая строка таблицы T.1 соответствует случаю, когда в рюкзаке остался незанятый объем, равный 215, и туда поместили два предмета, имеющих объем по 50 единиц. При этом общая стоимость вещей в рюкзаке вычисляется как сумма стоимости предметов, размещенных ранее (таблица T.0) в объеме  и предметов, помещенных в данный момент.

Решением данной задачи будет вектор  каждая компонента которого – количество предметов соответствующего типа.

1. 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, о расстановки скобок при перемножении матриц, о наибольшей общей подпоследовательности, вычисление дистанции Левенштейна.

1. 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 (иногда говорят более узком) их варианте.

происходит из области теории программирования.

Рекурсивная функция – это функция, которая вызывает саму себя.

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. На рис. 7.2 приведен пример функции, вычисляющей факториал числа с помощью цикла. Нерекурсивная функция вычисления факториала числа

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 операционной системой резервируется специальная секция памяти, называемая системным стеком. 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.

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». На рис 7.3 представлен пример рекурсивной функции поиска максимального элемента в массиве, которая использует эту схема.

1. Математические основы сетевого планирования. Основные понятия теории графов.

Определения и обозначения: Выражение a ∈ A обозначает, что a является элементом множества A, а выражение a ∉ A применяется для отрицания принадлежности объекта a множеству A.

Если множество A и множество B состоят из одних и тех же элементов, то говорят, что эти два множества равны. Говорят, что множество  содержится в множестве , если каждый элемент является элементом множества A. Для записи этого факта используются выражения  или . При этом, множество называют подмножеством множества . Мощностью множества A называется количество элементов этого множества. Множество, имеющее конечное количество элементов. называется конечным множеством, а множество, имеющее бесконечное количество элементов – бесконечным. Если элементы бесконечного множества можно перенумеровать натуральными числами, то такое множество называется счетным множеством, и для обозначения его мощности используется специальный символ  (читается: алеф ноль). ). Бесконечное множество, элементы которого нельзя пронумеровать, называется несчетным множеством, и говорят, что оно имеет мощность континиума.

Операции над множествами: Объединением множеств  и  называется множество , включающие все элементы двух множеств. Выражение  обозначает, что  является результатом операции объединения множеств  и . Пересечением множеств  и  называется множество , состоящее из элементов, одновременно принадлежащих множествам  и . Выражение  обозначает, что  является результатом операции пересечения множеств A и . Разностью множеств  и  называется множество , состоящее из всех элементов множества , но не принадлежащих множеству . Выражение  обозначает, что  является результатом операции разности множеств  и . Дополнением множества  называется множество , состоящее из всех элементов универсального множества , исключая элементы множества . Выражение  обозначает, что  является результатом операции дополнения множества .

Если , то дополнением множества B до множества  называется множество , состоящее из всех элементов множества , исключая элементы множества B. Выражение  обозначает, что  является результатом операции дополнения множества  до множества .

1. Математические основы сетевого планирования. Кратчайшие пути между вершинами графа.

Пусть  – ориентированный граф, на множестве дуг которого определена функция . Функция  ставит каждой дуге  графа  в соответствие действительное число , которое часто называют весом дуги. При этом саму функцию  обычно называют весовой функцией, а граф – взвешенным графом.

В общем случае граф  может содержать несколько путей , , …  из вершины  в вершину . Весом  пути  взвешенного графа называется сумма весов дуг, составляющих этот путь: .

Кратчайшим путем  из вершины  в вершину  называется путь с минимальной весом . В общем случае в одном графе может быть несколько минимальных путей.

Для поиска кратчайшего пути используется алгоритмы Дейк-стры, Беллмана

1. Математические основы сетевого планирования. Максимальные пути между вершинами графа.

 – 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  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  безконтурный и его вершины пронумерованы так, что у любой дуги  конечная вершина всегда имеет больший номер, чем начальная: .. Пусть все вершины графа  имеют номера от  до .

Рассматриваемый алгоритм основывается на рекуррентном выражении , где  – множество начальных вершин дуг, входящих в вершину . При этом предполагается, что для всех вершин  ранга  значение .

Вычисление значений  необходимо выполнить для каждой вершины графа в порядке возрастания номера. Каждое полученное значение  представляет собой максимальный вес пути в графе  с конечной вершиной .

Цикл вычисления  сопровождается построением массива элементов , которые формируются по тому же принципу, что и в алгоритме Дейкстры. Каждый элемент  соответствует вершине графа . Значение  – вершина (или ее номер), предшествующая вершине  в пути максимального веса с конечной вершиной .

1. Сетевые модели. Применение сетевых моделей. Сетевые графики.

Проект –деятельность, имеющая начало и конец во времени и направленую на достижение определеного результата. Как правило проект представлется виде ряда элементаргых работ, которые могут быть выполнены только после завершения одной или нескольких других операции.

Совокупность операций проекта их зависемостей называеться комплексом операций от 2 момента времени: начало и окончание операции. Эти моменты называются событиями. Различают 3 вида событий: исходное, промежуточное, завершающ.

Исходное-событие, которое не является конечным ни для одной операции комплекса.

Завершающее- событие, которое не является начальным ни для одной операции комплекса. Все остальные промежуточные. Моментом совершения события считается момент окончания всех операции.

Сетевой график представляет собой взвешенный ориентированный корневой граф без контуров (ациклический) и изолированных вершин, который построен по определенным правилам.

Пусть Z={z1,z2,…,zm}-множество операций, а I={i0,i1,...,in} – множество событи комплекса операций проекта р. Построим орграф Gp=(v,e) по следующим правилам:

Кол-во времени графа равно кол-ву событий комплекса операций (VIII)

Кол-во дуг графа = кол-ву операций комплекса операций |EZ|

Должны быть заданы 2 биективные функции разметки, сохраняющие инцидентность событий и операций:

f: –на множество вершин графа i

g: Lk->Zk, Zk на множестве дуг графа Lk

Граф должен иметь только одну вершину i=f-1(i), не имеющую входящих дуг, она должна соответсятвовать исходящему событию i3= f(vs) комплекса.

Граф должен иметь только одну вершину i=f-1(iz), не имеющую исходящих дуг она должна соответствовать завершающему событию Lt=f(t)

Граф не должен содержать онтуров.

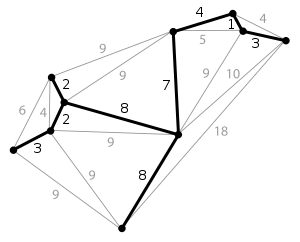
Любая вершина графа должна быть соеденины не более чем одной дугой.

Сетевой график комплекса операций- математическая модель, заданноя с помощью графа, построенного по описанным выше правилам.

1. Минимальные покрывающие деревья. Основные алгоритмы нахождения минимального остовного дерева.

Минимальное остовное дерево

**Минимальное остовное дерево** (или **минимальное покрывающее дерево**) в связанном взвешенном [неориентированном графе](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84) — это [остовное дерево](https://ru.wikipedia.org/wiki/%D0%9E%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) этого графа, имеющее минимальный возможный вес, где под весом дерева понимается сумма весов входящих в него рёбер.

[](https://commons.wikimedia.org/wiki/File:Minimum_spanning_tree.svg?uselang=ru)

Пример минимального остовного дерева в графе. Числа на ребрах обозначают вес ребер.

Задача о нахождении минимального остовного дерева часто встречается в подобной постановке: допустим, есть *n*городов, которые необходимо соединить дорогами, так, чтобы можно было добраться из любого города в любой другой (напрямую или через другие города). Разрешается строить дороги между заданными парами городов и известна стоимость строительства каждой такой дороги. Требуется решить, какие именно дороги нужно строить, чтобы минимизировать общую стоимость строительства.

Эта задача может быть сформулирована в терминах теории графов как задача о нахождении минимального остовного дерева в графе, вершины которого представляют города, рёбра — это пары городов, между которыми можно проложить прямую дорогу, а вес ребра равен стоимости строительства соответствующей дороги.

Существует несколько алгоритмов для нахождения минимального остовного дерева. Некоторые наиболее известные из них перечислены ниже:

* [**Алгоритм Прима**](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9F%D1%80%D0%B8%D0%BC%D0%B0)

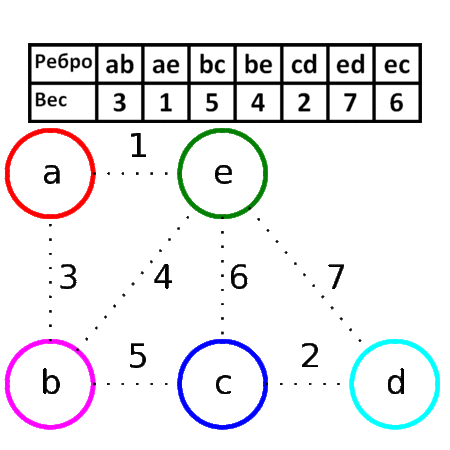
**Алгоритм Прима** — [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) построения [минимального остовного дерева](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BE%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) взвешенного связного неориентированного графа. Алгоритм впервые был открыт в 1930 году чешским математиком [Войцехом Ярником](https://ru.wikipedia.org/wiki/%D0%AF%D1%80%D0%BD%D0%B8%D0%BA,_%D0%92%D0%BE%D0%B9%D1%82%D0%B5%D1%85), позже переоткрыт [Робертом Примом](https://ru.wikipedia.org/w/index.php?title=%D0%9F%D1%80%D0%B8%D0%BC,_%D0%A0%D0%BE%D0%B1%D0%B5%D1%80%D1%82&action=edit&redlink=1) в 1957 году, и, независимо от них, [Э. Дейкстрой](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D0%B9%D0%BA%D1%81%D1%82%D1%80%D0%B0,_%D0%AD%D0%B4%D1%81%D0%B3%D0%B5%D1%80_%D0%92%D0%B8%D0%B1%D0%B5) в 1959 году.

На вход алгоритма подаётся связный неориентированный граф. Для каждого ребра задаётся его стоимость.

Сначала берётся произвольная вершина и находится ребро, инцидентное данной вершине и обладающее наименьшей стоимостью. Найденное ребро и соединяемые им две вершины образуют дерево. Затем, рассматриваются рёбра графа, один конец которых — уже принадлежащая дереву вершина, а другой — нет; из этих рёбер выбирается ребро наименьшей стоимости. Выбираемое на каждом шаге ребро присоединяется к дереву. Таким образом, при выполнении каждого шага алгоритма, высота формируемого дерева увеличивается на 1. Рост дерева происходит до тех пор, пока не будут исчерпаны все вершины исходного графа.

Результатом работы алгоритма является остовное дерево минимальной стоимости.

*  d[i]  — расстояние от i-й вершины до построенного дерева
*  p[i]  — предок i-й вершины, то есть такая вершина u, что (i,u) легчайшее из всех рёбер соединяющее i с вершиной из построенного дерева.
* w(i,j) — вес ребра (i,j)
* Q — приоритетная очередь вершин графа, где ключ — d[i]
* T — множество ребер минимального остовного дерева
* [**Алгоритм Краскала**](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%80%D0%B0%D1%81%D0%BA%D0%B0%D0%BB%D0%B0)**(или алгоритм Крускала)**

[](https://commons.wikimedia.org/wiki/File:MST_Kruskal.gif?uselang=ru)

Визуализация Алгоритма Крускала

**Алгоритм Крускала** (или **алгоритм Краскала**) — эффективный [алгоритм](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC) построения[минимального остовного дерева](https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D0%B5_%D0%BE%D1%81%D1%82%D0%BE%D0%B2%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE) взвешенного связного [неориентированного графа](https://ru.wikipedia.org/wiki/%D0%9D%D0%B5%D0%BE%D1%80%D0%B8%D0%B5%D0%BD%D1%82%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%BD%D1%8B%D0%B9_%D0%B3%D1%80%D0%B0%D1%84). Также алгоритм используется для нахождения некоторых приближений для [задачи Штейнера](https://ru.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%A8%D1%82%D0%B5%D0%B9%D0%BD%D0%B5%D1%80%D0%B0)[[1]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%80%D1%83%D1%81%D0%BA%D0%B0%D0%BB%D0%B0#cite_note-.D0.94.D0.B8.D1.81.D0.BA.D1.80.D0.B5.D1.82.D0.BD.D0.B0.D1.8F_.D0.BC.D0.B0.D1.82.D0.B5.D0.BC.D0.B0.D1.82.D0.B8.D0.BA.D0.B0.E2.80.942006.E2.80.94.E2.80.94307-1). Алгоритм впервые описан [Джозефом Крускалом](https://ru.wikipedia.org/w/index.php?title=%D0%9A%D1%80%D1%83%D1%81%D0%BA%D0%B0%D0%BB,_%D0%94%D0%B6%D0%BE%D0%B7%D0%B5%D1%84&action=edit&redlink=1) в [1956 году](https://ru.wikipedia.org/wiki/1956_%D0%B3%D0%BE%D0%B4).

Вначале текущее множество рёбер устанавливается пустым. Затем, пока это возможно, проводится следующая операция: из всех рёбер, добавление которых к уже имеющемуся множеству не вызовет появление в нём цикла, выбирается ребро минимального веса и добавляется к уже имеющемуся множеству. Когда таких рёбер больше нет, алгоритм завершён. Подграф данного графа, содержащий все его вершины и найденное множество рёбер, является его остовным деревом минимального веса. Подробное описание алгоритма приведено в книге[[2]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%80%D1%83%D1%81%D0%BA%D0%B0%D0%BB%D0%B0#cite_note-.D0.94.D0.B8.D1.81.D0.BA.D1.80.D0.B5.D1.82.D0.BD.D0.B0.D1.8F_.D0.BC.D0.B0.D1.82.D0.B5.D0.BC.D0.B0.D1.82.D0.B8.D0.BA.D0.B0.E2.80.942006.E2.80.94.E2.80.94309-311-2).

До начала работы алгоритма необходимо отсортировать рёбра по весу, это требует *O(E × log(E))* времени. После чего компоненты связности удобно хранить в виде[системы непересекающихся множеств](https://ru.wikipedia.org/wiki/%D0%A1%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B0_%D0%BD%D0%B5%D0%BF%D0%B5%D1%80%D0%B5%D1%81%D0%B5%D0%BA%D0%B0%D1%8E%D1%89%D0%B8%D1%85%D1%81%D1%8F_%D0%BC%D0%BD%D0%BE%D0%B6%D0%B5%D1%81%D1%82%D0%B2). Все операции в таком случае займут *O(E × α(E, V))*, где α — функция, обратная к [функции Аккермана](https://ru.wikipedia.org/wiki/%D0%A4%D1%83%D0%BD%D0%BA%D1%86%D0%B8%D1%8F_%D0%90%D0%BA%D0%BA%D0%B5%D1%80%D0%BC%D0%B0%D0%BD%D0%B0). Поскольку для любых практических задач *α(E, V) < 5*, то можно принять её за константу, таким образом, общее время работы алгоритма Крускала можно принять за *O(E \* log(E))*.

Алгоритм Крускала действительно находит остовное дерево минимального веса, поскольку он является частным случаем [алгоритма Радо — Эдмондса](https://ru.wikipedia.org/wiki/%D0%96%D0%B0%D0%B4%D0%BD%D1%8B%D0%B9_%D0%B0%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%A0%D0%B0%D0%B4%D0%BE_%E2%80%94_%D0%AD%D0%B4%D0%BC%D0%BE%D0%BD%D0%B4%D1%81%D0%B0)[[3]](https://ru.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%9A%D1%80%D1%83%D1%81%D0%BA%D0%B0%D0%BB%D0%B0#cite_note-3) для[графического матроида](https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9_%D0%BC%D0%B0%D1%82%D1%80%D0%BE%D0%B8%D0%B4), где независимые множества — ациклические множества рёбер.

1. Оптимизационные алгоритмы на графах. Алгоритм поиска в ширину.

Идея поиска в ширину состоит в том, чтобы посещать вершины в порядке их удаления от некоторой заране выбранной или указанной стартовой вершины а. Сначала посещаем вершину а, затем все вершины смежные с а, то есть находим от нее на расстояние 1, затем вершину находим на расстоянии 2 т.д. Рассмотрим алгоритм заданы стартовой вершины а: В начале все вершины помечены как новые. Первой посещается вершина а, но она становиться единственной открытой вершиной. В дальнейшем каждый шаг начинается с выбора некоторой открытой вершиной х, далее исследуется приобра инцидентные активное вершине х. Если такое ребро соединяет вершину х, с новой вершиной у, то вершина у посещается и превращается в открытую. Когда инцидентные активные вершины исследованные, она перестает быть активной и становиться закрытой. После этого выбирается новая активная вершина. Описанные действия повторяются. Процесс заканчивается, когда множество открытых вершин становиться пустым. Основная особенность поиска в ширину отличающая его от других способов обхода графа, состоит в том, что в качестве активной вершины выбирается та из открытых которая была посещена раньше других. Именно этим обеспечивается главное свойство поиска в ширину: тем ближе вершина к старту тем раньше она будет посещена. Для реализации такого правила, выбора активной вершины. Удобно использовать для хранения множества открытых вершин очередь. Тогда новая вершина становиться открытой она добавляется в конец очереди, а активная выбирается в её начале.

1. Оптимизационные алгоритмы на графах. Алгоритм поиска в глубину.

Поиск в глубину (DFS)- одна из наиболее важных из-за многочисленности приложений стратени обхода графа.

Идея этого метода- идти вперед не исследованную область, пока это возможно. Если же вокруг все исследовано, то отступить на шаг назад. И искать новые возможности для продвижение вперед.Метода поиска в глубину известен под разными названиями, например back tracking, поиск с возвращениями и др. Понятие новой, открытой, закрытой к активной вершин для поиска в глубину имеют такой же смысл, как и для поиска в ширину, однако всегда имеется не более чем одна активная вершина. Обход начинается с посещения заданной стартовой вершины а, которая становиться активной и единственной открытой вершиной. За тем выбирается инцидентная вершина а ребро (а,у) и посещается вершина у. Она становиться открытой и активной. Заметим, что при поиске в ширину вершина а оставалась активной до тех пор пока не были исследованы все инцидентные её ребра. В дальнейшем, как и при поиске в ширину, каждый очередной шаг начинается с выбора активной вершины из множества активных вершин. Если все ребра, инцидентные активной вершине х, уже исследованы, она превращается в закрытую. В противном случае выбирается одна из неисследованных ребер (х,у), это ребро исследуется. Если вершина у новая, то она посещается и превращается в открытую. Главное отличие от поиска в ширину состоит в том, что при поиске в глубину в качестве активной выбирается та из открытых вершин, которая была посещена последней. Для реализации такого правила выбора наиболее удобной структурой хранения множества открытых вершин является стек. Открываемые вершины складываются в стек в том порядке, в котором они открываются, а в качестве активной выбирается последняя вершина.

1. Оптимизационные алгоритмы на графах. Топологическая сортировка.

Топологическая сортировка - алгоритм упорядочивание вершин безюнтурного ориентированного графа согласно линейного порядка. Заданого дугами на множестве вершин.

Топологическая сортировка вершина орграфа G=(V,E) заключается в писваении его вершинам номеров 1,…..,n, т.о., что бы для любой дуги этого графа выполняется условие: (vi ; vj )ɛE;.Это возможно в том случае, если граф не имеет контуров, является ациклическим. Топологическая сортировка может рассматриваться как процесс оперделения линейного порядка на множестве вершин, в которые может быть вложен частичный порядок, заданный множеством дуг.

Суть алгоритма: Топологическая сортировка начинается с нахождения вершины из которой не выходит ни одной дуги. Такая вершина всегда существует, если в графе нет контуров. Её присваивается наибольший номер n и она удаляется вместе с входящими в нее дугами. Т.к. оставшийся граф также не содержит контуров, процесс повторяется и новой вершине, из которой не выходят дуги, присваивается наибольший номер n-1 и т.д.

1. Потоки в сетях. Задача о максимальном потоке. Теорема Форда-Фалкерсона.
2. Потоки в сетях. Задача о максимальном потоке. Алгоритм Форда-Фалкерсона.

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.

Обнуляем все потоки. Остаточная сеть изначально совпадает с исходной сетью.

В остаточной сети находим любой путь из источника в сток. Если такого пути нет – останавливаемся.

Пускаем через найденный путь (он называется увеличивающим путем) max возможный поток: -на найденном пути в остаточной сети ищем ребро с min пропускной способностью cmin

-Для каждого ребра на найдем пути увеличиваем поток на cmin, а в противоположном пути уменьшаем на cmin.

-Модифицируем остаточную сеть. Для всех ребер на найденном пути, а так же для противоположных им ребер, вычисляем левую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.

1. Задачи нелинейного программирования. Основные алгоритмы решения.

НП - раздел мат прога, изучающий методы решения экстремальных задач с не линейной целевой фун-ей и обл. допустимых решений, определенной нелинейными ограничениями. К НП относят квадратное, дробное, выпуклое, дискретное, целочисленное и геометрическое программирование. В общем случае задача НП: F(x)->min(max) при усл. g(x)≤0 где x искомых перем., F(x)-целевая числовая функция, g(x)-векторная функция системы ограничений.

1. Постановка задачи векторной оптимизации.

Эффективность функционирования экономической системы оценивается, как правило, несколькими критериями. Математической формой критерия эффективности в оптимизационных экономико-математических задачах является целевая функция.

Пусть имеется  критериев, которые можно записать в виде целевых функций , где . Поскольку , то для простоты в дальнейшем будем предполагать, что все целевые функции максимизируются. Задача многокритериальной оптимизации в этом случае запишется

 (1)

; (2)

. (3)

Если точки максимума , определенные при решении задач по каждому критерию  не совпадают, то решение задачи (1)-(3) может быть только компромиссным. В области допустимых значений задачи находится область компромиссов. При перемещении из одной точки области компромиссов в другую, невозможно одновременное улучшение всех критериев. Принадлежащие области компромиссов планы называются эффективными или оптимальными по Парето (по имени итальянского экономиста, впервые сформулировавшего проблему многокритериальной оптимизации и принцип оптимальности).

План  оптимален по Парето, если он допустим и не существует другого плана  для которого



и хотя бы для одного критерия выполняется строгое неравенство.

К задачам векторной оптимизации приходят в следующих случаях:

1. Качество моделируемого процесса нужно оценить с точки зрения нескольких показателей. Это могут быть прибыль, себестоимость, рентабельность и т.д.
2. Моделируемые процесс представляет собой составляющую нескольких процессов (частей), и каждая из этих частей имеет свой критерий качества.
3. Моделируемые процесс расчленяется на несколько шагов и на каждом шаге его качество определяется своей функцией. (Например, на отдельных временных промежутках)

При разработке методов решения многокритериальных задач приходится решать ряд специфических проблем.

1. *Проблема нормализации* возникает наиболее часто. Отдельные критерии как правило имеют различные единицы и масштабы измерения, что делает невозможным их непосредственное сравнение. К единому и безразмерному виду критерии приводятся посредством операции нормирования. Наиболее распространенными способами нормирования является замена абсолютных значений критериев их относительными величинами

,

или относительными значениями отклонений от оптимальных значений критериев

.

2. *Проблема учета приоритета критериев* встает, если критерии имеют различную значимость. В этом случае необходимо найти математическое определение приоритета и степень его влияния на решение задачи.

3. *Проблема определения области компромисса* возникает при решении многомерных нелинейных задач, поэтому для их решения необходимо применять методы, гарантирующие эффективное решение.

Методы решения задач многокритериальной оптимизации можно подразделить на четыре группы:

* + - * методы, основанные на свертывании критериев;
      * методы, использующие ограничения на критерии;
      * методы целевого программирования;
      * методы, основанные на отыскании компромиссного решения.

Вместо исходной многокритериальной задачи в соответствии с выбранным методом, формируется замещающая задача. В состав замещающей задачи входит один критерий, а к исходной системе ограничений добавляется одно или несколько дополнительных ограничений. Решение замещающей задачи называется субоптимальным.

1. Методы решения задач векторной оптимизации.

В методах, основанных на свертывании критериев, из локальных критериев формируется один. Наиболее распространенным является *метод линейной комбинации частных критериев*. Пусть задан вектор весовых коэффициентов критериев , характеризующих важность соответствующего критерия, ,  . Линейная скаляризованная функция представляет собой сумму частных критериев, умноженных на весовые коэффициенты. Задача математического программирования становится однокритериальной и имеет вид



;

.

Критерии в свертке могут быть нормированы. Решение, полученное в результате оптимизации скаляризованного критерия эффективно.

К недостаткам метода можно отнести то, что малым приращениям коэффициентов соответствуют большие приращения функции, т. е. решение задачи неустойчиво, а также необходимость определения весовых коэффициентов.

*3. Метод последовательных уступок.*

Рассмотрим один из методов, использующих ограничения на критерии – метод *последовательных уступок*. Алгоритм метода следующий:

1. Критерии нумеруются в порядке убывания важности.

2. Решается задача



;

.

Определяется значение .

3. Устанавливается уступка , по этому критерию.

4. Решается задача



;

.

Если в задаче более двух критериев, то пункты 3 и 4 повторяются для , ..., .

Пример. Решить задачу методом последовательных уступок, если уступка по первому критерию составляет 10% от его оптимального значения.



Решение.

Решим задачу по критерию . Получим . В соответствии с условием задачи величина уступки . Дополнительное ограничение будет иметь вид , то есть . Решая задачу



получим , , 