Математическое программирование

Преподаватель:

Бракович Андрей Игоревич

Оглавление

[Введение в мат. программирование. 4](#_Toc484499728)

[План лекции 4](#_Toc484499729)

[Введение 4](#_Toc484499730)

[Комбинаторные методы решения оптимизационных задач. 5](#_Toc484499731)

[Генерация множества всех подмножеств. 5](#_Toc484499732)

[Сочетания 6](#_Toc484499733)

[Задача про судно 6](#_Toc484499734)

[Генератор перестановок 6](#_Toc484499735)

[Коммивояжер с использованием генератора перестановок 6](#_Toc484499736)

[Генератор размещений 7](#_Toc484499737)

[Оптимальное размещение контейнеров на судне с помощью генератора размещений 7](#_Toc484499738)

[Общие принципы решения задач оптимизации методом ветвей и границ 7](#_Toc484499739)

[Задача коммивояжера 8](#_Toc484499740)

[Утверждение 3. 8](#_Toc484499741)

[Утверждение 4. 9](#_Toc484499742)

[Рекурсивные алгоритмы 11](#_Toc484499743)

[Решение задачи о рюкзаке 11](#_Toc484499744)

[Вычисление дистанции Левенштейна 12](#_Toc484499745)

[Решение задачи о расстановке скобок при перемножении матриц 12](#_Toc484499746)

[Решение задачи вычисления длины наибольшей общей подпоследовательности 13](#_Toc484499747)

[Динамическое программирование 13](#_Toc484499748)

[Решение задачи о рюкзаке 13](#_Toc484499749)

[С помощью алгоритма расстояний Левенштейна 14](#_Toc484499750)

[Математические основы сетевого планирования 14](#_Toc484499751)

[Основные понятия теории графов 15](#_Toc484499752)

[Кратчайшие и максимальные пути между вершинами графа. 15](#_Toc484499753)

[Оптимизационные алгоритмы на графах 19](#_Toc484499754)

[Алгоритм поиска в ширину 19](#_Toc484499755)

[Алгоритм поиска в глубину 21](#_Toc484499756)

[Алгоритм топологической сортировки. 26](#_Toc484499757)

[Транспортная задача 29](#_Toc484499758)

[Сетевое планирование (математические основы) 34](#_Toc484499759)

[Минимальные покрывающие деревья 34](#_Toc484499760)

[Алгоритм Крускала 39](#_Toc484499761)

[Алгоритм Прима 39](#_Toc484499762)

[Потоки в сетях 40](#_Toc484499763)

[Задача о максимальном потоке 40](#_Toc484499764)

[Алгоритм Форда-Фаркинсона. 40](#_Toc484499765)

[Линейное программирование 44](#_Toc484499766)

[Математическая модель 45](#_Toc484499767)

[Графическое решение 45](#_Toc484499768)

[Симплекс метод решения задач линейного программирования 46](#_Toc484499769)

[Преобразование задачи в стандартную форму. 46](#_Toc484499770)

[Задачи нелинейного программирования 51](#_Toc484499771)

[Задачи векторной оптимизации 53](#_Toc484499772)

[Постановка задачи векторной оптимизации 53](#_Toc484499773)

[Оптимизация по Парето 53](#_Toc484499774)

[Метод последовательных уступок 54](#_Toc484499775)

[Сетевые модели 55](#_Toc484499776)

Введение в мат. программирование.

План лекции

Введение в мат. программирование.

Построение математической модели программирования.

Этапы решения задача математического программирования.

Введение

Математическое программирование – область математики, разрабатывающая теорию и численные методы решения многомерных экстремальных задач, т.е. задач на экстремум функции многих переменных с ограничением на область определения.

Решение задач математического программирования осуществляется в 4 этапа:

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

Самая сложная задача – 1.

В общем виде модель задачи математического программирования выглядит следующим образом:

Z(x)->{max, min} x принадлежит X

x – искомая, в общем случае, векторная величина.

X – область определения искомой величины.

Z – функция цели (функция, определяющая значение критерия оптимальности).

В зависимости от природы множества X и вида функции Z, задачи математического программирования классифицируются как задачи:

дискретного программирования (комбинаторная оптимизация, X – конечно или счётно),

целочисленное программирование (X - подмножество множества целых чисел),

линейного программирования (Z – линейная функция, а X может быть определено с помощью линейных неравенств),

нелинейного программирование (Z – нелинейная функция и/или присутствует в описании Х хотя бы 1 нелинейная функция (нелинейное неравенство)).

3 этап - метод решения.

Метод решения задач математического программирования определяется в зависимости от исходных данных.

4 этап вычисление решения задач математического программирования.

Осуществляется при помощи компьютерной техники.

Смежные дисциплины.

Математическое моделирование –модели-> исследование операций <-методы оптимизации-математическое программирование <-модели-математическое моделирование.

Литература

1. Смелов В.В., Комбинаторные алгоритмы оптимизации, БГТУ 2011 – взять.
2. Смелов, Основы сетевого планирования, БГТУ 2011.
3. Смелов, алгоритмы на графах и реализация на с++. БГТУ 2011.
4. Костевич математическое программирования 2003 – рекомендовано.
5. Таха, Введение в исследование операций. Вильямс 2001.
6. Кузнецов, Сакович Высшая математика. Математическое программирование. Высшая школа. 1994.

Примеры зада математического программирования:

1. Задача о кратчайшем расстоянии между вершинами графа
2. Задача о рюкзаке
3. Задача коммивояжера
4. Задача о загрузке судна
5. Задача о нахождении максимального потока в сети
6. Задача линейного программирования
7. Транспортная задача
8. Задача нелинейного программирования
9. Векторная оптимизация
10. Сетевое планирование

Комбинаторные методы решения оптимизационных задач.

Цель: освоение навыков решения оптимизационных задач комбинаторными методами.

Генерация подмножества заданного множества.

Генерация сочетаний.

Генерация перестановок.

Генерация размещений.

Особенность алгоритмов:

1. Невозможность использовать для задач большой размерности.
2. Применяются, когда необходимо найти точное или достаточное решение.
3. Сложность алгоритмов является верхней оценкой сложности решения задач., а улучшение возможно только в статистическом виде

Комбинаторный анализ – раздел математики, посвященный решению задач выбора и расположения элементов некоторого, обычно конечного, множества в соответствии с заданными правилами.

Генерация множества всех подмножеств.

Пусть Х равное множеству {x1,x2,..xn} конечное множество мощности n.

Множество А – множество всех подмножеств, называется булеаном Х и обозначается 2^X. Количество элементов равно 2^|X|.

Алгоритм генерации множества всех подмножеств основывается на взаимно однозначном соответствии между элементами булеана 2^X множества х и всеми целыми числами числа -..2^|x|-1, записанными в двоичном коде. Сложность алгоритма O(2^|x|) и может применяться при небольших входных значениях.

Алгоритм построения элементов булеана:

1. Пронумеровать элементы заданного множества Х начиная с 0.
2. Сформировать битовую последовательность для данных числе, пронумеровать ее.
3. Последовательно выполнить шаги 4 и 5 2^|X| раз.
4. Выбрать ихз множества Х элементы с номерами i, для которых bi=1.
5. Интерпретируя битовую последовательность как целое положительное число, увеличить его на 1.

Принцип реализации генератора:

Все генераторы должны иметь одинаковы интерфейс.

Должны быть возможность применения нескольких генераторов одновременно.

Функции должны легко встраиваться в другие.

Решение упрощенной задачи о рюкзаке с помощью генератора множества всех подмножеств. Существует n различных предметов, характеризующихся объемом Vi и стоимостью NiCi. Необходимо выбрать несколько разных предметов таким способом, чтобы они поместились в заданный объем и при этом цена была максимальной.

Математическая задача при этом выглядит следующим образом:

Sum(x\*v\*c)->max, при этом sum(x\*v) <V

x is [0,1].

Xi – требуется найти.

Решением задачи при такой постановке будет вектор (x1,x2,..xn), каждый элемент вектора xi может принимать значение 0|1, если 0 – то предмет не выбран, иначе выбран.

Объем рюкзака ограничен. (В примере 100 единиц).

n=4 – количество предметов.

Вектор объемов предметов 25,30,60,20.

Вектор стоимостей 25, 10,20,30.

Сочетания

Булеан можно рассматривать как объединение всевозможных сочетаний, построенных из элементов множества X. Поэтому генерация множества C может быть сведена к генерации множества 2^N и выбора нужный строк из него.

C=N1/M1/(N-M)!.

Задача про судно

На палубе m мест для контейнеров, всего дано n контейнеров, даны их стоимости и веса. Дана максимальная грузоподъемность данного судна.

Найти решение с максимальной прибылью.

Ki неизвестные номера контейнеров, которые требуется найти.

Решением задачи будет вектор k1,k2..km, каждый элемент которого может принимать целое значение из отрезка от 1 до n – выбранные контейнеры. При этом все Ki должны быть разными.

Грузоподъемность ограничена.

Сложность решения O(N!).

Генератор перестановок

Наиболее известным методом построения множества всех перестановок конечного множества Х является алгоритм Джонсона – Троттера. Алгоритм подразумевает, что все элементы множества Х можно единственным образом перечислить в порядке возрастания. Каждый элемент из множества Х получает стрелку, которая мб направлена влево или вправо. В алгоритме используется понятие мобильного элемента. Элемент Хi называется мобильным, если соответствующая ему стрелка указывает на меньший соседний элемент. В первой перестановке все элементы кроме самого левого являются мобильными. Количество перестановок равно n!.

1. Построить первую перестановку.
2. Найти наибольший мобильный элемент в текущей перестановке. Если в последовательности нет мобильного элемента, то построены все перестановки элементов множества Х – алгоритм завершил свою работу.
3. Поменять местами наибольший мобильный элемент и элемент, на который указывает стрелка наибольшего мобильного элемента.
4. Найти все элементы большие, чем наибольший мобильный элемент (если он есть) и изменить их стрелки на противоположное направление.
5. Перейти к пункту 2.

Коммивояжер с использованием генератора перестановок

Найти минимальный путь обхода n городов.

Расстояние между каждой парой городов dij известно.

Ki- неизвестные номера городов, которые требуется найти. Решением задачи будет вектор K1, K2, …Kn. Каждый элемент этого вектора может принимать целое значение из отрезка 1..n-1 включая оба конца. Все значения Ki должны быть разными. Решение задачи сводится к генерации всех допустимых векторов K1, K2, ..Kn, вычислению целевой функции и выбору вектора, соответствующего минимальному значению функции.

Замечания:

1. При построении оптимального маршрута выбор стартового города (он же является и конечным) никак не влияет на конечный результат. Если задано n городов, то перебор следует осуществлять только для n-1 городов, поскольку стартовый город можно зафиксировать.

Пример матрицы расстояний:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0 | 45 | Бесконечность | 25 | 50 |
| 45 | 0 | 55 | 20 | 100 |
| 70 | 20 | 0 | 10 | 30 |
| 80 | 10 | 40 | 0 | 10 |
| 30 | 50 | 20 | 10 | 0 |

Бесконечность – из одного места в другое нельзя доехать.

Сложность O((n-1)!).

Генератор размещений

Множество размещений Ax,m из |X|=n по m можно получить перестановкой всех элементов сочетаний Cx,m. Другими словами, для получения множества размещений Ax,m, требуется сначала сгенерировать все сочетания по m элементов из множества Х, а затем все перестановки элементов для каждого сочетания.

1. Для множества {0,1,2,3} формируется множество всех сочетаний по три элемента. Таких сочетаний будет C3(4)=4.
2. Для элементов каждого сочетания Ci,i=1..4 генерируется все перестановки P. Каждому сочетанию будет соответствовать P3=6 перестановок. Таким образом всего будет построено A3(4) перестановок, которые в совокупности представляют собой все размещения множества {0,1,2,3}.
3. Применяя полученные на предыдущем шаге размещения в качестве индексов для элементов множества X={a,b,c,d}, формируется множество Ax,3 всех размещений по 3.

Количество размещений

|Am,B|=Am(n)=n!/(n-m)!

Оптимальное размещение контейнеров на судне с помощью генератора размещений

На палубе судна имеется n мест для размещения контейнеров, выбрать m контейнеров для погрузки на судно можно из n>=m имеющихся в наличии. Каждый контейнер i характеризуется весом vi и доходом ci, i=1..n от его перевозки. На каждое место i=1..m можно разместить контейнер с ограничениями по весу. Необходимо выбрать m контейнеров из n имеющихся таким образом, чтобы доход от перевозки был максимально возможным.

Математическая модель: получить максимальную прибыл, при этом не «убив» судно. Требуется найти номера выбранных контейнеров. Решением задачи будет вектор – k1,k2..kn, каждый элемент этого вектора может принимать целое значение из отрезка 1..n и при этом все значения ki должны быть разные.

Сложность O(An(m)).

Общие принципы решения задач оптимизации методом ветвей и границ

Метод ветвей и границ – общий алгоритмический метод решения задач комбинаторной оптимизации. Является вариацией полного перебора с отсевом подмножеств допустимых решений заведомо не содержащих оптимальных решений.

В основе метода лежат 2 процедуры – процедура ветвления (BR), позволяющая разбивать множество допустимых решений на непересекающиеся подмножества и процедура вычисления нижней или верхней границы.

Деление на 2 части, если в одной из частей нет оптимального решения, дальнейший проход по той ветви не производится.

Опишем процедуру ветвления BR, применяемую в этом алгоритме. 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. Пусть R – множество всех кольцевых маршрутов, проходящих через все вершины графа. Это множество R соответствует множеству всех допустимых решений задачи коммивояжера.

Процедура EV вычисления нижней границы основывается на двух утверждениях:

1. Изменение всех элементов строки матрицы расстояний на одно и то же число не влияет на выбор оптимального маршрута коммивояжера.
2. Изменение всех элементов столбца матрицы расстояний на одно и то же число не влияет на выбор оптимального маршрута коммивояжера.

Если последовательно для каждой вершины Vj графа G вычислить соответствующие a, b, то g=sum(a+b) будет составляющей веса кратчайшего кольцевого маршрута, но никогда его не превысит. Другими словами, g можно использовать как нижнюю границу веса кратчайшего кольцевого маршрута.

Задача коммивояжера

Задача коммивояжера решается исходя из количества городов и расстояний между каждой парой городов. Представление расстояний в виде матрицы. В общем виде матрица может быть несимметрична.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Город | 1 | 2 | 3 | 4 | 5 |
| 1 | Беск | 9 | 8 | 4 | 10 |
| 2 | 6 | Беск | 4 | 5 | 7 |
| 3 | 5 | 3 | Беск | 6 | 2 |
| 4 | 1 | 7 | 2 | Беск | 8 |
| 5 | 2 | 4 | 5 | 2 | Беск |

В соответствии с утверждением 1, если все элементы первой строки таблицы уменьшить на 4 (наименьшее значение в строке), то это не повлияет на порядок городов в кратчайшем кольцевом маршруте, а лишь сократит общую длину на 4 км. Эту операцию называют приведением таблицы по строке, а число 4 константой приведения.

Аналогично поступают с остальными строками исходной таблицы.

После приведения по строкам матрица:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Город | 1 | 2 | 3 | 4 | 5 |
| 1 | Беск | 5 | 4 | 0 | 6 |
| 2 | 2 | Беск | 0 | 1 | 3 |
| 3 | 3 | 1 | Беск | 4 | 0 |
| 4 | 0 | 6 | 1 | Беск | 7 |
| 5 | 0 | 2 | 3 | 0 | Беск |

Столбец Ai содержит константы приведения для каждой строки, а общая сумма равна 13. В соответствии с утверждением 2, если все элементы второго столбца таблицы уменьшить на 1 (наименьшее значение во втором столбце), то это не повлияет на порядок городов в кратчайшем кольцевом маршруте, а лишь сократит его длину на 1 км. (в остальных столбцах минимальные значения равны 0). Эта операция называется приведением таблицы по столбцу, а число 1-константой приведения.

Утверждение 3.

Приведение матрицы расстояний не влияет на выбор оптимального маршрута коммивояжера. В этом утверждении необходимо посчитать сумму констант приведения.

13 (по строкам) + 1 (по столбцам) – сумма констант приведения.

R- множество всех допустимых решений/возможный маршрутов.

Утверждение 4.

fi(R)=y( R) – нижняя граница длины маршрута.

Если бы после приведения таблицы расстояний все ее элементы были бы равны 0, то длина кратчайшего маршрута была бы равна y.

fi=13+1=14. fi можно принять в качестве нижней границы длины кратчайшего кольцевого маршрута.

Вершина R – корневая вершина дерева T, соответствующая множеству R всех допустимых решений данной задачи.

Полностью приведенная матрица имеет вид:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Город | 1 | 2 | 3 | 4 | 5 |
| 1 | Беск | 4 | 4 | 0 | 6 |
| 2 | 2 | Беск | 0 | 1 | 3 |
| 3 | 3 | 0 | Беск | 4 | 0 |
| 4 | 0 | 5 | 1 | Беск | 7 |
| 5 | 0 | 1 | 3 | 0 | Беск |

Прежде чем будет применена процедура ветвления BR, оценим как будет изменяться нижняя граница подмножеств, полученных после применения процедуры BR. В зависимости от выбора дуги, которая используется для разбиения. Наибольший интерес представляет та дуга, которая наиболее сильно может повлиять на нижнюю границу длины допустимых кольцевых маршрутов. Анализировать необходимо дуги, имеющие нулевой вес, так как только их удаление может повлиять на нижнюю границу. Удаление дуги моделируется установкой ее веса, равного бесконечности.

В каждый из 0 подставляется бесконечность и для каждого случая считается константа приведения. Необходимо выбрать тот маршрут, при котором константа приведения максимальная.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Город | 1 | 2 | 3 | 4 | 5 |  |
| 1 | Беск | 4 | 4 | Беск | 6 | 4 |
| 2 | 2 | Беск | 0 | 1 | 3 | 0 |
| 3 | 3 | 0 | Беск | 4 | 0 | 0 |
| 4 | 0 | 5 | 1 | Беск | 7 | 0 |
| 5 | 0 | 1 | 3 | 0 | Беск | 0 |
|  | 0 | 0 | 0 | 0 | 0 | S=4 |

Фрагмент графа, который мб построен на этом этапе решения задачи

R->R (1,4)

Вторая ветвь решения после процедуры BR соответствует множеству кольцевых маршрутов, которые включают дугу (1,4).

В этом случае необходимо перестроить таблицу, удалив первую строку (во всех допустимых маршрутах из города 1 есть дуга только в 4) и 4ый столбец. В городе 4 есть только единственная дуга из города 1.

Кроме того, допустимый кольцевой маршрут, содержащий дугу (1,4) не может содержать дугу (4,1).

На пересечении 4 строки и 1го столбца необходимо поставить символ бесконечности.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Город | 1 | 2 | 3 | 5 |  |
| 2 | 2 | Беск | 0 | 3 | 0 |
| 3 | 3 | 0 | Беск | 0 | 0 |
| 4 | Беск | 5 | 1 | 7 | 1 |
| 5 | 0 | 1 | 3 | Беск | 0 |
|  | 0 | 0 | 0 | 0 | S=1 |

Полностью приведенная матрица:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Город | 1 | 2 | 3 | 5 |  |
| 2 | 2 | Беск | 0 | 3 | 0 |
| 3 | 3 | 0 | Беск | 0 | 0 |
| 4 | Беск | 4 | 0 | 6 | 0 |
| 5 | 0 | 1 | 3 | Беск | 0 |
|  | 0 | 0 | 0 | 0 | S=0 |

Граф решения на данном этапе:

fi=13+1=14;

Если взять в маршрут (1,4), то fi=14+1=15;

Если не взять в маршрут (1,4), то fi=14+4=18.

Выбирается маршрут, где константа приведения меньше.

Далее идет опять попытка подстановки бесконечностей и вычисление констант приведения.

Если не ехать в данную ветвь, то к предыдущей сумме добавляется текущая константа приведения.

Если выбрать (4,3), то

|  |  |  |  |
| --- | --- | --- | --- |
| Город | 1 | 2 | 5 |
| 2 | 2 | Беск | 3 |
| 3 | 3 | 0 | 0 |
| 5 | 0 | 1 | Беск |

Константа равна 2 (общая сумма fi=15+2=17)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| R | -> | R(1,4) | -> | R(1,4)(4,3) | -> | R(1,4)(4,3)(2,1) (18) |
| | |  |  |  | | |  |  |
| R(1,4) (18) |  |  |  | R(1,4)(4,3)!(2,1) (18) |  |  |

граф содержит три вершины со значением 18

В этом случае целесообразно продолжать наращивать ветви графа, начиная с вершин, наиболее удаленных от корня. В нашем случае выбирается вершина (1,4)(4,3)(2,1).

Анализ матрицы позволяет выявить 2 последних звена кольцевого маршрута. Это (3,5) и (5,2). Таблица не может быть приведена, поэтому окончательный вид графа

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| R | -> | R(1,4) | -> | R(1,4)(4,3) | -> | R(1,4)(4,3)(2,1) (18) | (3,5) |
| | |  | | |  | | |  |  | (5,2) |
| R!(1,4) (18) |  | R(1,4)!(4,3) (19) |  | R(1,4)(4,3)!(2,1) (18) |  |  |  |

Для получения окончательного решения необходимо расставить полученные дуги в правильном порядке.

(1,4) (4,3)(3,5)(5,2)(2,1)

Сложив расстояния, соответствующие дугам кольцевого маршрута, получим 18 км, что совпадает с нижней границей, приписанной последнему узлу графа.

Кроме того, необходимо обратить внимание на несколько листовым вершин дерева, которым тоже соответствует нижняя граница, равная 18ти. Продолжение вычислений для этих ветвей в лучшем случае приведет к получению кольцевого маршрута такой же длины, но никогда меньшей.

Рекурсивные алгоритмы

Рекурсивный алгоритм – алгоритм, решающий задачу путем сведения ее к решению одной или нескольких таких задач, но в сокращенном варианте.

Рекурсивная функция – функция, которая вызывает саму себя.

Пример: факториал.

Решение задачи о рюкзаке

Классическая формулировка задачи допускает выбор одинаковых предметов при размещении их в рюкзаке. Математическая модель классической задачи о рюкзаке выглядит следующим образом:

Sum(i=1..n) XiViCi->max, sum(i=1..n) XiVi<=V, т.е. Xi принадлежит {0,1,2..[V/Vi], I = 1,n.

где Xi – неизвестные (количество предметов каждого типа), которые требуется найти; Vi – объемы предметов, Ci – стоимость единицы объема каждого предмета; V – вместимость рюкзака.

Решением задачи при такой постановке будет вектор X=(x1,x2..xn). Каждый элемент Xi вектора может принимать целое значение из отрезка [0, [V/Vi]]. При этом, если Xi=0, то i-ый предмет не выбран, и если Xi>0, то –x предметов в рюкзак помещено Xi единиц.

Z=Z(1,n)=max(Z(1,n-1)+XnVnCn),

Xn принадлежит {0,1,..[V/Vi]},

Z(1,n-1)=max(Z(1,n-2)+X(n-1)V(n-1)\*C(n-1),

Xn-1принадлежит {0,1,..[(V-R(n,n))/V(n-1)]}

Z(1,1)=X1V1C1, X1=[(V-R(2,n)/V1]

Схема решения задачи о рюкзаке при n=3, V=120, v[]={30,50,20}, c[]={5,9,3}

n – общее число элементов

V – общий объем.

vi – объем, занимаемый книгой.

ci – цена единицы объема i-того элемента.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  |  | (0,120) |  |  |  |
| -> | -> |  |  | -> |  |
| x3=0 … | x3=1 |  |  | x3=2 | x3=3+.. |
|  | 60,100 |  |  | 120,80 |  |
| -> | -> | -> |  | -> | -> |
| x2=0 | x2=1 | x2=2 |  | x2=0 | x2=1 |
| 60, 100 | 510, 50 | 960, 0 |  | 120, 80 | 570, 30 |
| x1=3 | x1=1 | x1=0 |  | x1=2 | x1=1 |
| 510,10 | 660,20 | 960, 0 |  | 420,20 | 720,0 |

Для примера рассмотрим маршрут x[]={2,1,1}. Первый этап отмечен меткой (120, 80). Первое число – текущая стоимость рюкзака. Второе число – остаток неиспользованного объема рюкзака. В рюкзак поместили 2 предмета с номером 3, поэтому стоимость рюкзака равна 2\*20\*3=120 единиц. В рюкзаке остается 120 – 2\*20 = 80 единиц объема.

2 этап имеет метку 570, 30. Внутри 2 предмета 3 типа, 1 предмет 2 типа. Поэтому, 570 = 120+1\*50\*9.

3 этап, заполняем весь остаток книгами 1 типа. Стоимость рюкзака: 570+1\*30\*5 = 720.

Вычисление дистанции Левенштейна

Дистанция Левенштейна (расстояние Левенштейна, редакционное расстояние, дистанционное редактирование) определяется между двумя строками и равна минимальному количеству операций вставки одного символа, удаления одного символа и замены одного символа на другой, необходимых для превращения одной строки в другую.



len(x) – количество символов в заданной строке. lel(“Hello”) = 5.

cut(x) – заданная строка без последнего символа. cut(“Hello”) = “Hell”.

last(x) – последний символ указанной строки. last(“Hello”) = ‘o’.

Тупо рекурсия без ДП:-(.

Решение задачи о расстановке скобок при перемножении матриц

Вычислим количество операций умножения, которые необходимо выполнить при перемножении трех матриц A1, A2, A3 размерностью (10\*100), (100\*5), (5\*50).

При умножении (1\*2)\*3 будет 7500 операций, а в (1\*(2\*3) будет 75 000 операций.

к – индекс, который задается в соответствии с формулой.

mi,j­ = { 0, i=j;

pi-1pipi+1, j-i =1;

min (mi,k + mk+1 , j + pi-1pkpj), j-1>1 при (i<=k<j)

Расстановку скобок в последовательности матриц можно осуществить с помощью двухмерного массива s. Скобки расставляются по принципу «сначала внешние, затем внутренние».

Например,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 0 | 1 | 1 | 3 | 3 | 3 |
| 0 | 0 | 2 | 3 | 3 | 3 |
| 0 | 0 | 0 | 3 | 3 | 3 |
| 0 | 0 | 0 | 0 | 4 | 5 |
| 0 | 0 | 0 | 0 | 0 | 5 |
| 0 | 0 | 0 | 0 | 0 | 0 |

Найдем элемент (1,6) в матрице s, 1 строка = 6 столбец.

Это означает, что точка разрыва находится между первой и шестой матрицей после третьей матрицы, что позволяет расставить внешние скобки следующим образом:

(1\*2\*3) \* (4\*5\*6). – 1 равносильно A1.

Точку разрыва между первой и третьей матрицей определяет элемент (1,3). Между четвертой и шестой – стоит 5, значит, после 5ой матриц будет стоять скобка.

(1 \* (2 \* 3)) \* ((4 \* 5) \* 6).

Решение задачи вычисления длины наибольшей общей подпоследовательности

Пусть X = {xi}n - последовательность n символов.

Последовательность Z={zi}k называется подпоследовательностью последовательности Х, если она может быть получена из последовательности Х удалением ее некоторых элементов. Например, X={A,B,C,D,T,F,G,E}, Z={D,F,E} – верная подпоследовательность.

Подпоследовательность Z={zi}k называется общей подпоследовательностью, последовательностей X={xi}n и Y={yi}p , если Z – подпоследовательности и X, и Y.

Длина последовательности – это количество ее элементов.

Подпоследовательность Z, называется наиболее общей подпоследовательностью Х и У, если Z имеет наибольшую длину среди всех общих подпоследовательностей.

Обозначим Cn,p длину для LCS подпоследовательностей Х и У.



Динамическое программирование

// Перепись - собеседование

Метод решения задачи оптимизации, реализующей рекурсивный алгоритм, который разбивает задачу на подзадачи, и каждая такая подзадача считается только 1 раз, а остальные разы приведения к ней берутся готовые значения.

Решение задачи о рюкзаке

7 типов предметов (N)

Вектор 85, 50, 75, 65, 30, 40, 30 – объемы

Стоимость 150, 25, 30, 40, 40, 45.

Построение задач при решении задач ДП осуществляется постоянно в порядке их выполнимости на основе предыдущих значений.

Возьмем таблицу, 1 столбец – остаток объема, второй столбец – суммарно занимаемый объем, 3 столбец – суммарная цена.

Пример:

|  |  |  |
| --- | --- | --- |
| 300 | 0 | 0 |
| 300 | 85 | 12750 |
| 300 | 170 | 25500 |
| 300 | 255 | 38250 |

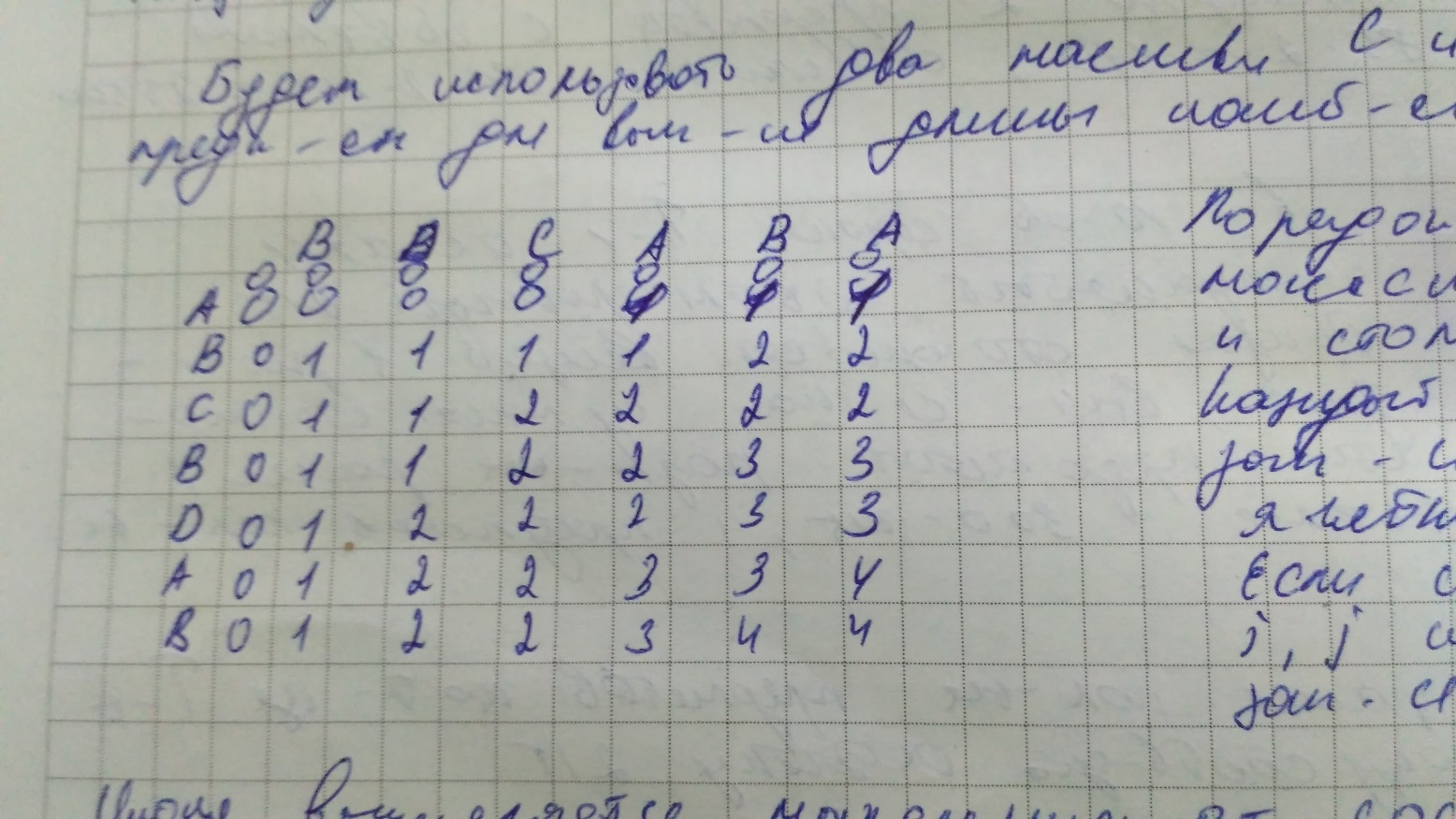
При возникновении ситуации с оставшимся объемом 300 необходимо лишь выбрать лучшее решение либо хранить только лучшие решения для каждой из ситуаций.

С помощью алгоритма расстояний Левенштейна

Решение задачи о наибольшей общей подпоследовательности при помощи ДП.

Будем использовать два массива С и В, массив С представлен для вычисления длины наибольшей общей подпоследовательсноти.

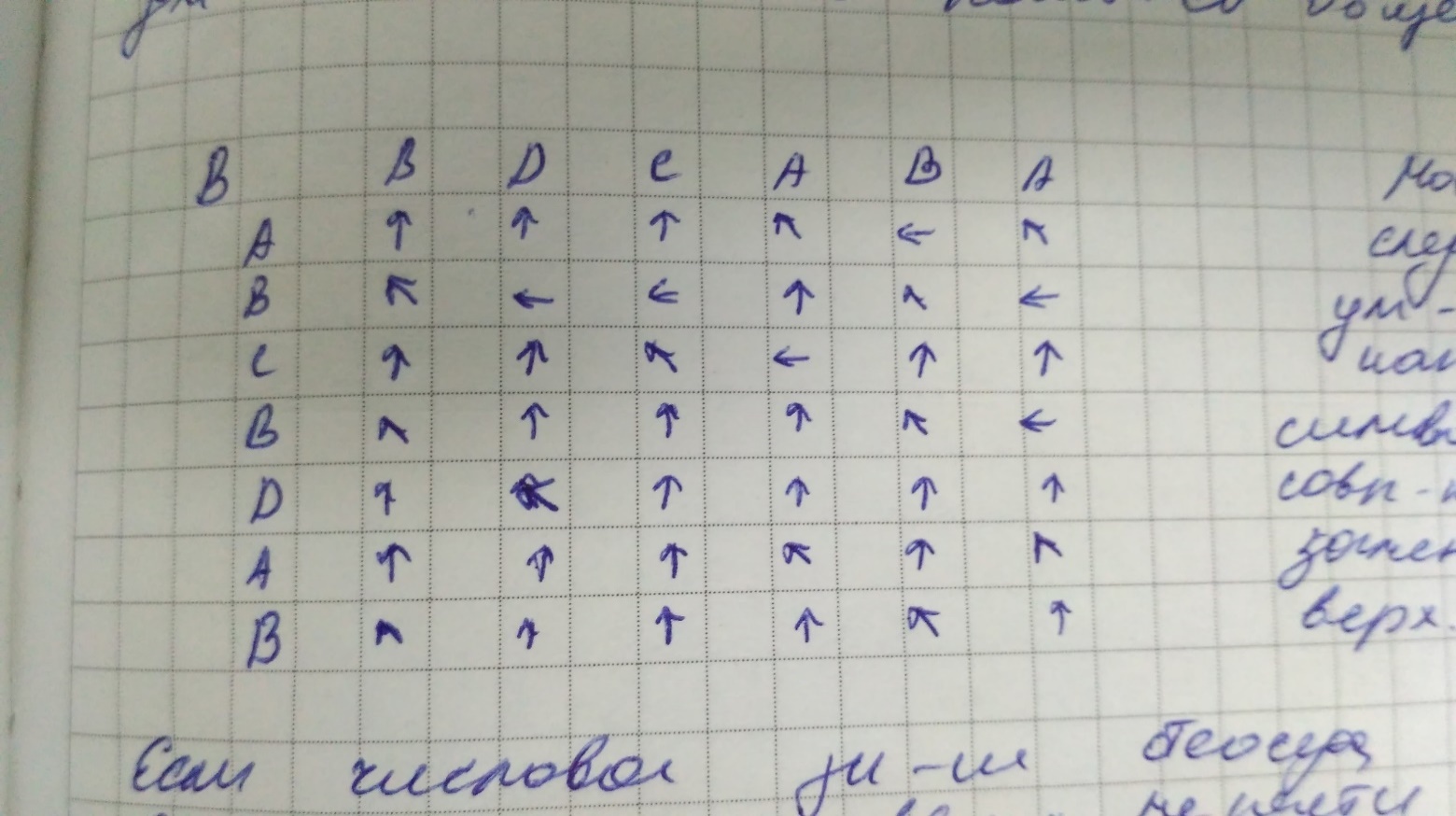
Порядок заполнения С – в первой строке и столбце нули.

 Тетрадь РЭВ.

Каждый элемент подпоследовательности получает Ci,j. Если символы на позициях I, j совпадают – в ячейку записывается Ci-1,j-1+1 иначе вычисляется максимум от соседа слева и сверху.

Длина наибольшей подпоследовательности будет находиться в правом нижнем углу массива С. Массив В предназначен для получения самой наибольшей общей подпоследовательности.

Массив В записывается следующим образом:

 Тетрадь РЭВ.

по умолчанию все стрелки направлены вверх, если символы для позиции совпадают, то стрелки направляются налево-вверх. Если число взято слева, то стрелка направляется налево, иначе вверх. Сама подпоследовательность получается обходом стрелок с правого нижнего угла влево вверх во время движения.

Математические основы сетевого планирования

Цель: освоение теоретических основ и практических навыков решения задач с применением основ сетевого планирования.

Основные понятия теории графов

Рассмотрим множество V, которое будем называть множеством вершин, и набор E является подмножеством V2 упорядоченных пар (v1, v2) принадлежащих V2, называемый множеством дуг. Пара (V, E) называется графом. Запись G= (V, E) обозначает граф с именем G, состоящий из множества вершин V и множества дуг E.

Обычно вершины графа изображаются в виде точек, а дуги в виде соединяющих это точки стрелок.

Смежные вершины – соединенные дугой вершины графа.

Вершины инцидентны дуге, если соединены ей.

Графы бывают ориентированные и неориентированные.

Способы представления графов:

1. Матрица смежности (около 30% случаев).

По строкам и столбцам вершины.

1. Матрица инцидентности имеет размер m (количество вершин) \* n (количество дуг). Каждый элемент принимает значение из (0,1,-1). 1 – если дуга j выходит из вершины i, -1 – если дуга j входит в вершину i, 0 – вершина не инцидентна.

G= (V, E), V={0,1,2,3,4,5}, E={(0,1),(2,1),(2,3),(3,5)}

Если петля, то ставится в таблице +-1.

1. Списки смежных вершин представляют собой набор из m множеств, соответствующих m вершинам графа. Множество, соответствующее вершине i, i=1,m либо пустое, либо содержит номера вершин, в которые входит дуга, выходящая из вершины i.

Пример:

S1= {2,3,4}, S2={2}, S3={2}, S4={1}, S5={пустое множество}.

Кратчайшие и максимальные пути между вершинами графа.

Пусть G=(V, E) – ориентированный граф, на множестве дуг которого определена функция v:E->R. Функция w ставит каждой дуге ei принадлежит E графа G в соответствие действительное число x принадлежит R, которое часто называют весом дуги. При этом саму фунцию w обычно называют весовой функцией, а граф – взвешенным графом.

В общем случае граф G=(V, E) может содержать несколько путей из вершину vi в vi. Весом w(pij) пути взвешенного графа называется сумма весов дуг, составляющих этот путь.

Кратчайшим путем pij из вершины vi в вершину vj называется путь с максимальным весом wij=min(w(pijk)). В общем случае в одном графе может быть несколько минимальных путей.

Алгоритм Дейкстры решает задачу о кратчайших путях во взвешенном графе G=(V, E) с дугами неотрицательного веса w(ei)>=0, ei принадлежит E, i=1,[E] из вершины s во все достижимые вершины этого графа. Алгоритм последовательно преобразовывает исходный граф G в дерево кратчайших путей G’. Дерево кратчайших путей это граф G’=(V’, E’), обладающий следующими свойствами:

1. G’ подмножество G;
2. G’ – ориентированное дерево с корнем s;
3. V’ – множество достижимых вершин графа G из s;
4. для каждой вершины vi принадлежит V’ путь из s в vi в дереве G’ является кратчайшим путем в графе G.

Алгоритм Дейкстры предполагает, что граф G=(V, E) задан с помощью списков смежных вершин, а аткже известна весовая функция W(vi,vj) =

1. w(vi,vj) принадлежит R, (vi,vj) принадлежит E

2. +бесконечность, (vi,vj) не принадлежит E

V\*V. Функция W ставит в соответствие каждой паре вершин (vi,vj) вес w(vi,vj) дуги, соединяющие эти вершины или + бесконечность, если такой дуги нет в графе G. Кроме этого, заданная вершина s относительно которой определяются все кратчайшие пути,

Реализация:

A[v] – множество вершин графа G, смежных с вершиной v принадлежит V.

d[v] – верхняя оценка кратчайшего пути из вершины s вершину v принадлежит V. В начале работы алгоритма d[s]=0, а для всех остальных v принадлежит V\{s} оценка d[v]=+бесконечность.

p[v] – вершина, предшествующая вершине v принадлежит V в дереве кратчайших путей. В начале работы алгоритма для всех v принадлежащих V p[v]=nil, где nil-специальный символ, обозначающий пустоту.

S – множество вершин s подмножество V. Вначале множество S пустое, S= пустое множество. В процессе работы алгоритма S пополняется вершинами из V, для которых уже определен кратчайший путь из вершины s. После окончания работы S=V.

Q – множество вершин Q подмножество V. Вначале Q=v. В прцоессе работы алгоритма элементы S:Q=V\S. После окончания работы алгоритма Q=пустое множество.

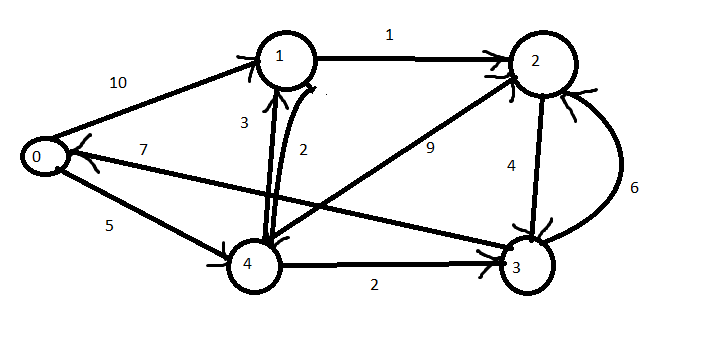
exstractQ- прцоедура извлечения элемента из множества Q с наименьшим значением d[v].

relax(u,v) – процедура релаксации, которая определена для произвольных двух вершин графа G. Релаксация состоит в следующем: d[v]-=(d[u]+w(u,v)) в том слечае, если d[u]+w(v,u)<d[u].

сигма(v) – кратчайший путь из s в v из V.

Выполнение алгоритма Дейкстры состоит из двух этапов. На первом инициализируются значения S,Q,d,p значениями. Второй этап алгоритма описан с помощью псевдокод.

|  |
| --- |
| while Q!=пустое множество  {  u=extractQ  S=S объединение {u}  for (v из A(u))  {  relax(u,v)  }  } |



d[v]

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | бесконечность |
| 2 | бесконечность |
| 3 | бесконечность |
| 4 | бесконечность |

p[v]

|  |  |
| --- | --- |
| 0 | nil |
| 1 | nil |
| 2 | nil |
| 3 | nil |
| 4 | nil |

Q=P0,1,2,3,4}.

Стоимость из 0 в соответствующие вершины.

d[v]

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 10 |
| 2 | бесконечность |
| 3 | бесконечность |
| 4 | 5 |

Массив предшествующих вершин

p[v]

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | nil |
| 2 | nil |
| 3 | nil |
| 4 | 0 |

Q={пусто, 1,2,3,4), S={0, пусто, пусто, пусто, пусто).

d[v]

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 5+3=8 |
| 2 | 5+9=14 |
| 3 | 5+2 |
| 4 | 5 |

p[v]

|  |  |
| --- | --- |
| 0 | nil |
| 1 | 4 |
| 2 | 4 |
| 3 | 4 |
| 4 | 0 |

Q={пусто, 1,2,3,пусто},S={0,4,free,free,free}.

Извлекаем ту вершину, которой еще не было, а также значение d[v] минимально.

Поэтому выбираем 3ку

d[v]

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 5+3=8 (не стало лучше) |
| 2 | 5+9=14 было, а стало 5+2+6=13 так как открылась вершина 3 |
| 3 | 5+2 так и осталось |
| 4 | 5 |

p[v]

|  |  |
| --- | --- |
| 0 | nil |
| 1 | 4 |
| 2 | 3 |
| 3 | 4 |
| 4 | 0 |

Q={пусто, 1,2,пусто,пусто},S={0,4,3,free,free}.

Выбираем единицу

d[v]

|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 5+3=8 (не стало лучше) |
| 2 | 9 |
| 3 | 7 |
| 4 | 5 |

p[v]

|  |  |
| --- | --- |
| 0 | nil |
| 1 | 4 |
| 2 | 1 |
| 3 | 4 |
| 4 | 0 |

Q={пусто, пусто,2,пусто,пусто},S={0,4,3,1,free}.

Выбираем двойку.

Выбираем единицу

d[v]

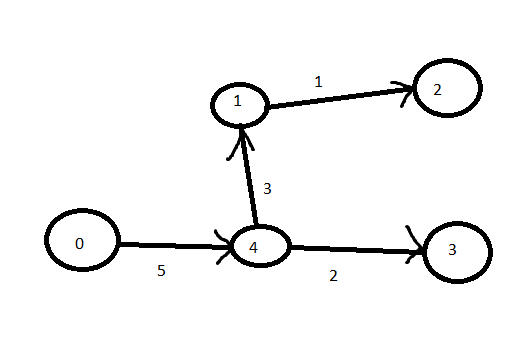
|  |  |
| --- | --- |
| 0 | 0 |
| 1 | 8 |
| 2 | 9 |
| 3 | 7 |
| 4 | 5 |

p[v]

|  |  |
| --- | --- |
| 0 | nil |
| 1 | 4 |
| 2 | 1 |
| 3 | 4 |
| 4 | 0 |

Q= {пусто, пусто, пусто, пусто, пусто},S={0,4,3,1,2}.

Итоговый граф:



//лекция 28.03.2017

Для поиска максимального массив d используется для верхней оценки.

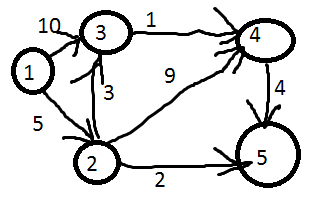
Максимальным путем называется путь с максимальным весом w=maxk,I,j(w(pyk)). В общем случае, в одном графе может быть несколько максимальных путей.

Рассматриваемый алгоритм основывается на рекурсии d[vj]=maxv i прин V-(v j)(d[vi]+w(vi,vj)), где V-(vj) – множество начальных вершин дуг, входящих в вершину vj. При этом предполагается, что для всех вершин uпринадлежит V0 ранга 0 значение d[u]=0.

Вычисление массива d необходимо выполнить для каждой вершины графа в порядке возрастания номера. Каждое полученное значение d представляет собой максимальный вес пути в графе G=(V,E) с конечной вершиной v.

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 v принадлежит V. Значение p[v] –вершина (или ее номер), предшествуя вершине v в пути максимального веса с конечной вершиной v.

Пример:



d[v]= {0, nil, nil, nil, nil}, p[v]= {nil, nil, nil, nil, nil}.

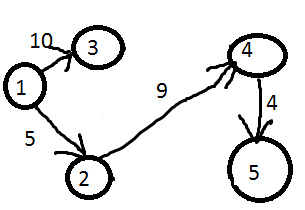
d[v]= {0, 5, nil, nil, nil}, p[v]= {nil, 1, nil, nil, nil}. Единственная дуга (1,2), которая позволит достичь вершины 2.

d[v]= {0, 5, 10, nil, nil}, p[v]= {nil, 1, 1, nil, nil}. Алгоритм рекурсивный. В вершину 3 входит 2 дуги. 1ый вариант 1-3 длиной 10, вой вариант 1-2-3 длиной 8. Выбираем первый.

d[v]= {0, 5, 10, 14, nil}, p[v]= {nil, 1, 1, 2, nil}. Варианты попадания в вершину 4: 1-3-4 (11),1-2-4(14),1-2-3-4(9), выбираем длину 14, путь 1-2-4.

d[v]= {0, 5, 10, 14, 18}, p[v]= {nil, 1, 1, 2, 4}. Варианты попадания: …4-5(максимум там найдей 14+4) либо …2-5(7).

Итоговое решение:



Оптимизационные алгоритмы на графах

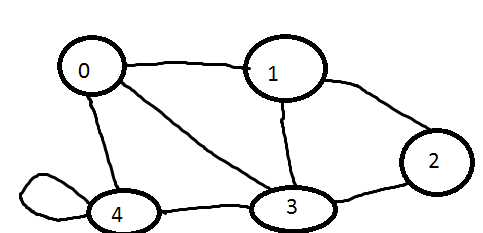
В данной лекции: алгоритм поиска в длину, в ширину, топологическая сортировка.

Алгоритм поиска в ширину

(BFS, breadth first search)

Алгоритм подразумевает, что задана исходная вершина, и основывается на простом правиле: при выборе очередной вершины, предпочтение отдается ближайшей. При этом считается, что все дуги графа имеют единичную длину. Сначала посещается стартовая вершина, затем все вершины, смежные ей (находящиеся на расстоянии 1), после чего вершины, находящиеся на расстоянии 2 от стартовой и т.д.

Для реализации используется Q-очередь вершин, C-массив окраска вершин, D-массив расстояний и P-массив предшествующих вершин.

Q={0}, C={G,W,W,W,W}, D={0,I,I,I,I}, P={N,N,N,N,N}.

Очередь Q используется для промежуточного хранения номеров вершин. На каждом шаге алгоритма в очередь помещаются номера вершин в порядке их обнаружения. Ка каждом шаге, кроме первого, из очереди извлекается очередной номер вершины, подлежащий отметке о посещении. На первом шаге алгоритма в очередь помещается номер стартовой вершины. На последнем шаге очередь пуста.

Массив С используется для хранения состояния вершин. Состояние вершин характеризуется при помощи цвета. Белый (W) – вершина не посещалась, серый (G) -вершина посещалась, черный (B) – фиксирован факт посещения вершины.

В массиве D для каждой вершины хранятся расстояния от стартовой вершины. На первом шаге для стартовой вершины в массиве D устанавливается значение 0, а для остальных вершин – значение бесконечность (I). На последнем шаге алгоритма для всех доступных вершин будут заполнены значения, равные их расстоянию от стартовой вершины.

Массив Р позволяет восстановить порядок обхода вершин и хранит для каждой вершины, кроме стартовой, предшествующую в обходе вершину. На первом шаге алгоритма всем элементам массива присваивается значение пустоты N. На последнем шаге алгоритма для всех доступных вершин будут заполнены значения, равные номеру предшествующей вершины в порядке обхода.

После посещения ближайших вершин 1, 3, 4:

Q= {1, 3, 4}.

C= {B, G, W, G, G}.

D= {0, 1, Inf, 1, 1}.

P= {N, 0, N, 0, 0}.

На следующем этапе добавляется 1-3:

Q= {3, 4, 2}.

C= {B, B, G, G, G}.

D= {0, 1, 2, 1, 1}.

P= {N, 0, 1, 0, 0}.

На следующем шаге извлекли 3 из очереди, покрасили ее, ходов нет:

Q= {4, 2}.

C= {B, B, G, B, G}.

D= {0, 1, 2, 1, 1}.

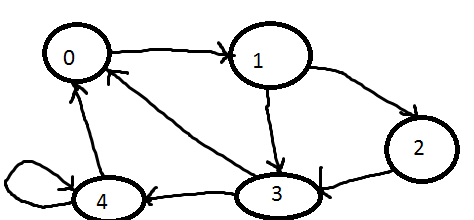
P= {N, 0, 1, 0, 0}.

Далее из очереди извлекается 4,2…Очередь пуста, C= {B, B, B, B, B}, D и P выше записаны.

В итоге получен граф обхода всех вершин на BFS дереве.

BFS дерево – это дерево, множество вершин которого является подмножеством вершин исходного графа, связанных дугами в порядке их посещения (в соответствии с массивом P), а корнем является стартовая вершина.

Теперь решение для ориентированного графа (ля чего алгоритм и используется, для неориентированного был просто пример):



Действия выполняются так же, как и в неориентированном графе, но движение допустимо только по направлению дуги.

После посещения 1 вершины

Q= {1}

C= {B, G, W, W, W}

D= {0, 1, I, I, I}

P= {N, 0, N, N, N}

Далее посещаются 2 и 3 вершины

Q= {2,3}

C= {B, B, G, G, W}

D= {0, 1, 2, 2, I}

P= {N, 0, 1, 1, N}

Next:

Q= {3}

C= {B, B, B, G, W}

D= {0, 1, 2, 2, I}

P= {N, 0, 1, 1, N}

Next:

Q= {4}

C= {B, B, B, B, G}

D= {0, 1, 2, 2, 3}

P= {N, 0, 1, 1, 3}

Next:

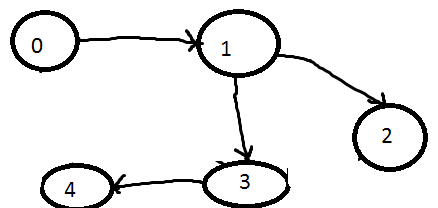
Q= {}

C= {B, B, B, B, B}

D= {0, 1, 2, 2, 3}

P= {N, 0, 1, 1, 3}

Результат:



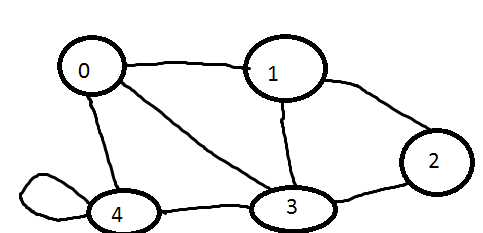
Алгоритм BFS сводится к следующей последовательности шагов:

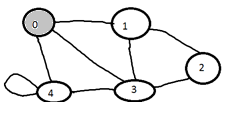
1. Массивы C, D, P, очередь W, окраска C[s]=G, D[s]=0. Стартовая вершина s.
2. Если Q пуста, то работа алгоритма завершена, иначе к 3 шагу.
3. Выбрать из очереди Q вершину к и окрасить ее в черный цвет C[k]=B.
4. Построить из множества J вершин белого цвета смежных вершине к. Если таких вершин нет, то перейти к шагу 2, иначе – к следующему шагу.
5. Каждую вершину j из множества J поместить в очередь Q. Обычно в очередь вершины помещаются в порядке возрастания номеров.
6. Каждую вершину j из множества J окрасить в серый цвет C[j]=G
7. Для каждой вершины j из множества J вычислить расстояния D[j]=D[k]+1
8. Для каждой вершины j из множества J указать предшествующую вершину P[j]=k.
9. К шагу 3.

Алгоритм поиска в глубину

Алгоритм поиска в глубину – DFS (depth-first search). Необходимо знать стартовую вершину. Алгоритм описывается следующим образом: для каждой не пройденной вершины начиная со стартовой, необходимо найти все смежные вершины и повторить для каждой.

Назначение и размерность массивов такие же, как и в BFS. В массиве D для каждой вершины записывается время обнаружения (шаг окраски в серый цвет). Массив F предназначен для хранения времени фиксации (окраска в черный цвет). Кроме того, используется переменная t – номер шага алгоритма. Массив С окраски вершин. Массив Р предыдущих вершин.





t = 1

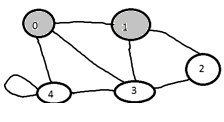
C = {G, W, W, W, W}

D = {1, Inf, Inf, Inf, Inf}

P = {N, N, N, N, N}

F = {0, 0, 0, 0, 0}

Следующий шаг (посещение вершины 1 из 0):



t = 2

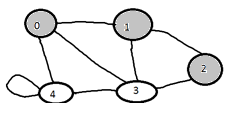
C = {G, G, W, W, W}

D = {1, 2, Inf, Inf, Inf}

P = {N, 0, N, N, N}

F = {0, 0, 0, 0, 0}.

Следующий шаг (посещение вершины 2 из 1):



t = 3

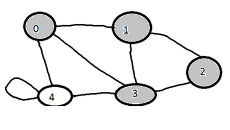
C = {G, G, G, W, W}

D = {1, 2, 3, Inf, Inf}

P = {N, 0, 1, N, N}

F = {0, 0, 0, 0, 0}.

Следующий шаг (посещение вершины 3 из 2):



t = 4

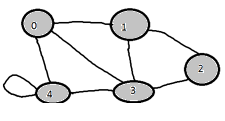
C = {G, G, G, G, W}

D = {1, 2, 3, 4, Inf}

P = {N, 0, 1, 2, N}

F = {0, 0, 0, 0, 0}.

Следующий шаг (посещение вершины 4 из 3):



t = 5

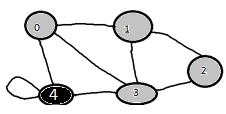
C = {G, G, G, G, G}

D = {1, 2, 3, 4, 5}

P = {N, 0, 1, 2, 3}

F = {0, 0, 0, 0, 0}.

Следующий шаг (Нет путей из 4ех – окрашиваем 4ре в черный цвет):



t = 6

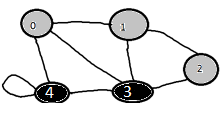
C = {G, G, G, G, B}

D = {1, 2, 3, 4, 5}

P = {N, 0, 1, 2, 3}

F = {0, 0, 0, 0, 6} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг (Нет путей из 3ех – окрашиваем 3рb в черный цвет):



t = 7

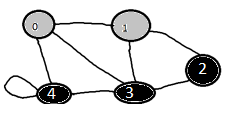
C = {G, G, G, B, B}

D = {1, 2, 3, 4, 5}

P = {N, 0, 1, 2, 3}

F = {0, 0, 0, 7, 6} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг (Нет путей из 2ех – окрашиваем 2 в черный цвет):



t = 8

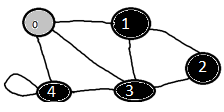
C = {G, G, B, B, B}

D = {1, 2, 3, 4, 5}

P = {N, 0, 1, 2, 3}

F = {0, 0, 8, 7, 6} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг (Нет путей из 1 – окрашиваем 1 в черный цвет):



t = 9

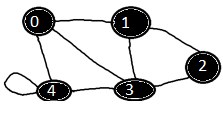
C = {G, B, B, B, B}

D = {1, 2, 3, 4, 5}

P = {N, 0, 1, 2, 3}

F = {0, 9, 8, 7, 6} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг (Нет путей из 1 – окрашиваем 1 в черный цвет):



t = 10

C = {B, B, B, B, B}

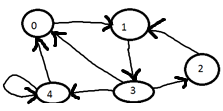
D = {1, 2, 3, 4, 5}

P = {N, 0, 1, 2, 3}

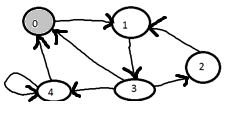
F = {10, 9, 8, 7, 6} – номер шага, когда вершина с данным индексом стала черной.

Итоговое решение строится по массиву Р – {N, 0, 1, 2, 3}.

Теперь обход в глубину для ориентированного графа.



Первый шаг:



t = 1

C = {G, W, W, W, W}

D = {1, Inf, Inf, Inf, Inf}

P = {N, N, N, N, N}

F = {0, 0, 0, 0, 0} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг: посещение 1 в серый цвет из 0

t = 2

C = {G, G, W, W, W}

D = {1, 2, Inf, Inf, Inf}

P = {N, 0, N, N, N}

F = {0, 0, 0, 0, 0} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг: посещение 3 в серый цвет из вершины 1

t = 3

C = {G, G, W, G, W}

D = {1, 2, Inf, Inf, Inf}

P = {N, 0, N, 1, N}

F = {0, 0, 0, 0, 0} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг: посещение 2 в серый цвет из вершины 3

t = 4

C = {G, G, G, G, W}

D = {1, 2, 4, 3, Inf}

P = {N, 0, 3, 1, N}

F = {0, 0, 0, 0, 0} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг: 2 окрашивается в черный цвет и возврат в 3ку

t = 5

C = {G, G, B, G, W}

D = {1, 2, 4, 3, Inf}

P = {N, 0, 3, 1, N}

F = {0, 0, 5, 0, 0} – номер шага, когда вершина с данным индексом стала черной.

Следующий шаг: посещение 4ки из тройки

t = 6

C = {G, G, B, G, G}

D = {1, 2, 4, 3, 6}

P = {N, 0, 3, 1, 3}

F = {0, 0, 5, 0, 0}

Массив Р уже получен, ответ уже можно получить, но нет, но если есть отдельно поддерево например 5->6 и необходимо окрасить все вершины в черный цвет.

Следующий шаг: посещение 4ки из тройки

t = 7

C = {G, G, B, G, G}

D = {1, 2, 4, 3, 6}

P = {N, 0, 3, 1, 3}

F = {0, 0, 5, 0, 0}

Следующий шаг: посещение уход из 4ки и окрас ее в черный цвет

t = 8

C = {G, G, B, G, B}

D = {1, 2, 4, 3, 6}

P = {N, 0, 3, 1, 3}

F = {0, 0, 5, 0, 7}

Следующий шаг: посещение уход из 4ки и окрас ее в черный цвет

t = 9

C = {G, G, B, B, B}

D = {1, 2, 4, 3, 6}

P = {N, 0, 3, 1, 3}

F = {0, 0, 5, 9, 7}

Следующий шаг: посещение уход из 4ки и окрас ее в черный цвет

t = 10

C = {G, B, B, B, B}

D = {1, 2, 4, 3, 6}

P = {N, 0, 3, 1, 3}

F = {0, 10, 5, 9, 7}

Следующий шаг: посещение уход из 4ки и окрас ее в черный цвет

t = 11

C = {B, B, B, B, B}

D = {1, 2, 4, 3, 6}

P = {N, 0, 3, 1, 3}

F = {11, 10, 5, 9, 7}

Итоговый обход выполняется по массиву P:

->2

0->1->3

->4

Пример программной реализации:

В основе лежит рекурсивная процедура Visit, имеющая один входной параметр – к – вершину графа.

Процедура Visit:

1. k – параметр графа.
2. Вершину к окрасить в серый цвет C[k] = G.
3. t++.
4. подсчитать расстояние до вершины D[k] = t. Расстояние до вершины в алгоритме DFS совпадает с номером шага, на котором эта вершина была обнаружена (окрашена в серый цвет).
5. Построить множества J вершин белого цвета, смежных вершине к. Если таких вершин нет, то перейти к шагу 8.
6. Для каждой вершины j из множества J указать предшествующую вершину P[j] = k.
7. Для каждой вершины j из множества J выполнить процедуру Visit.
8. Вершину к окрасить в черный цвет: C[k] = B.
9. t++.
10. Отметить фиксации вершин F[k] = t.

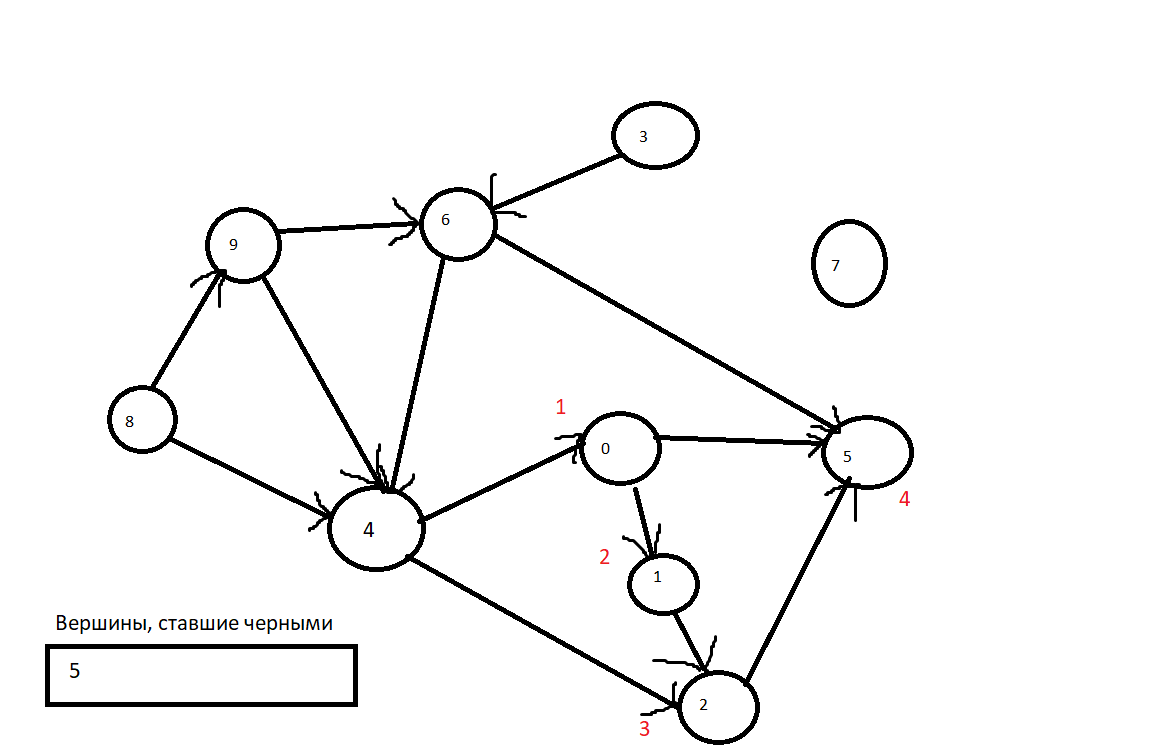
Если задана стартовая вершина s – алгоритм DFS теперь можно свести к следующим двум шагам.

1. Инициализировать массивы: С (все вершины белые), D (все расстояния Inf), P (все элементы пустота), t = 0 (1).
2. Выполнить процедуру Visit для вершины s.

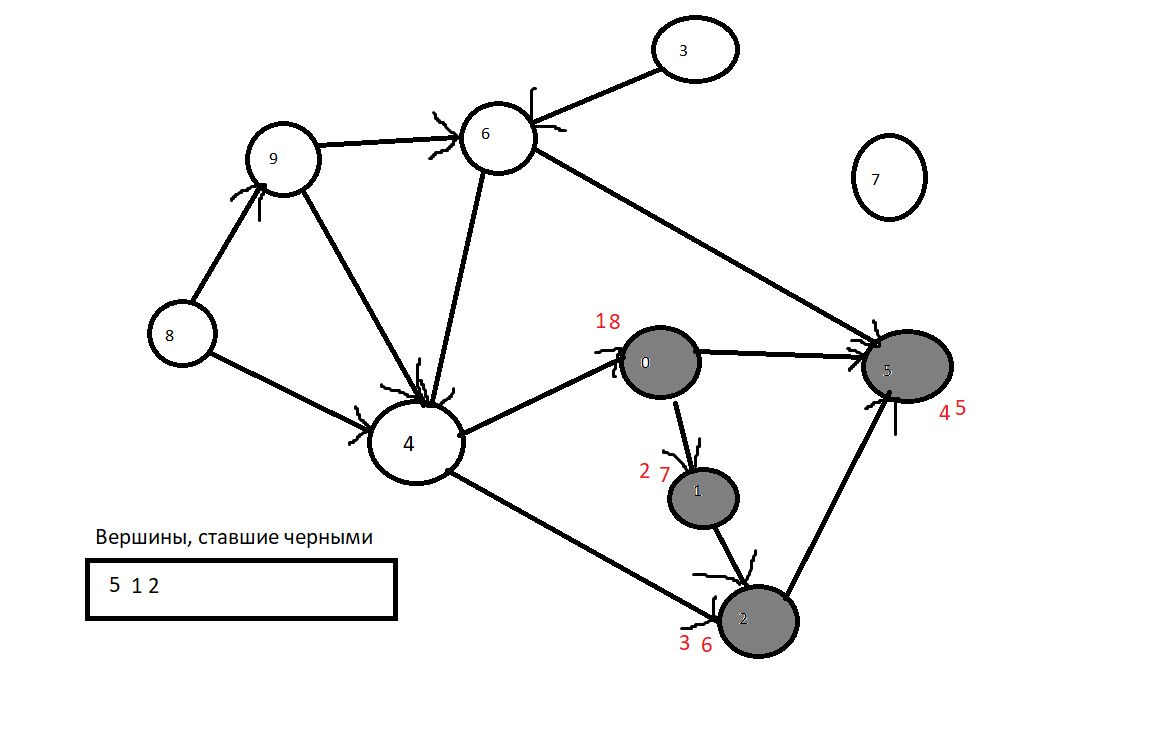
Алгоритм топологической сортировки.

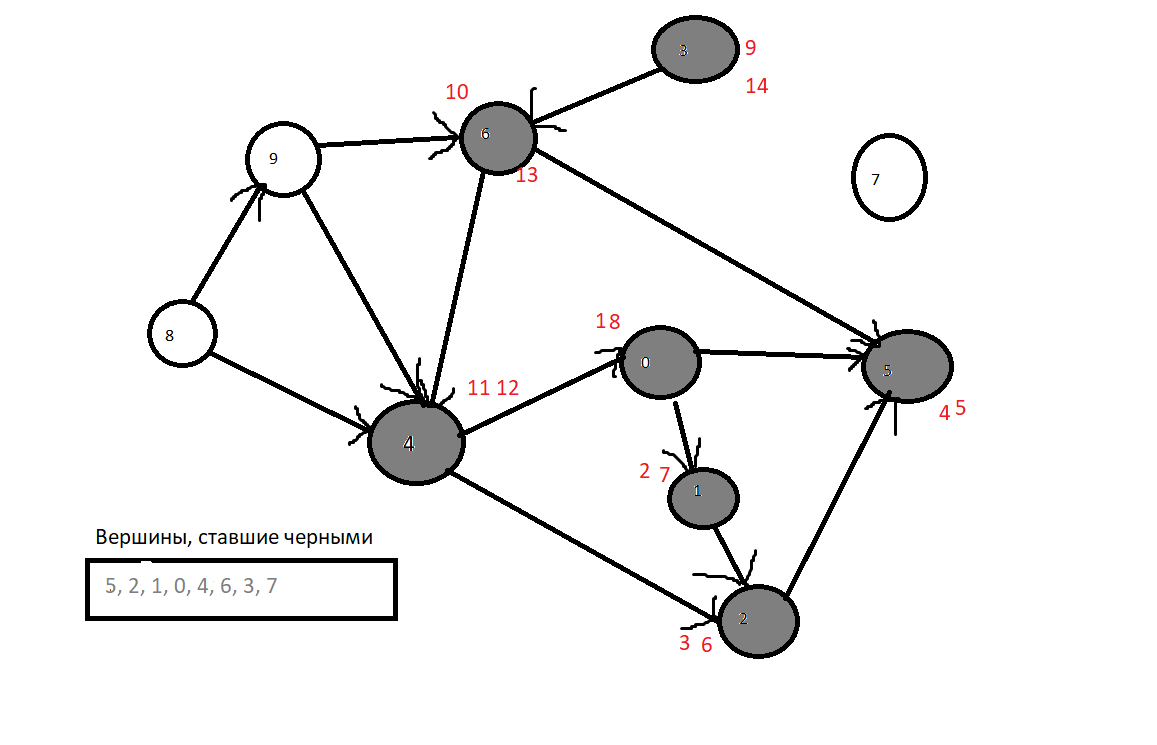
Топологическая сортировка – процедура упорядочивания вершин ориентированного графа, не имеющего циклов (ациклического графа). В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху вниз, то дуги будут направлены только от верхних к нижним. Обычно после выполнения топологической сортировки вершины переименовываются (перенумеровываются) в соответствии с полученным порядком. После такого переименовывания 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.



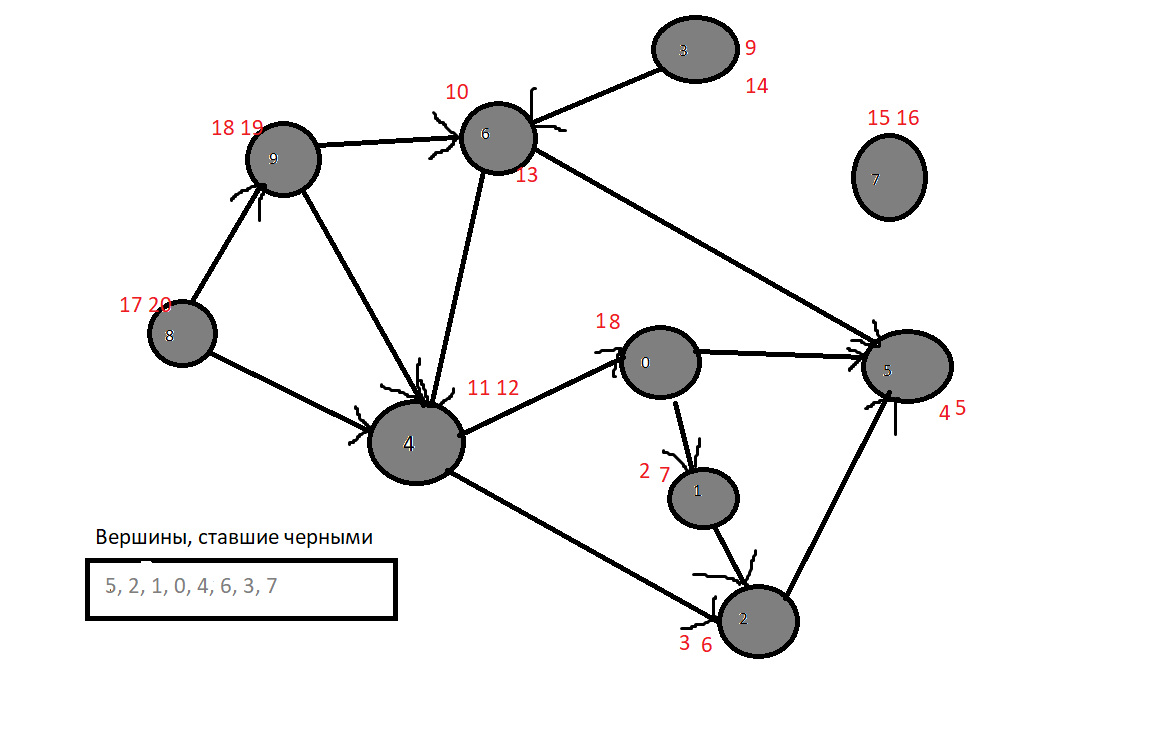
Красным – номера этапов обхода. Обход в глубину.

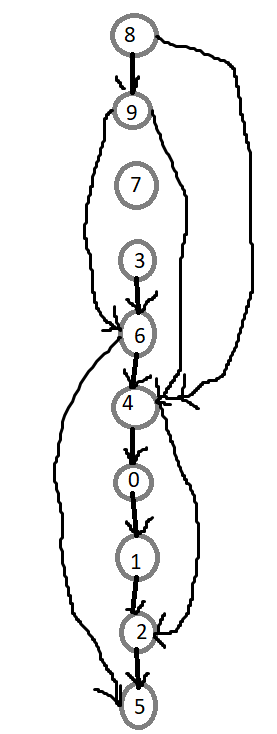




Далее одиночный обход 7ки, которая просто закрасится в черный за 2 этапа

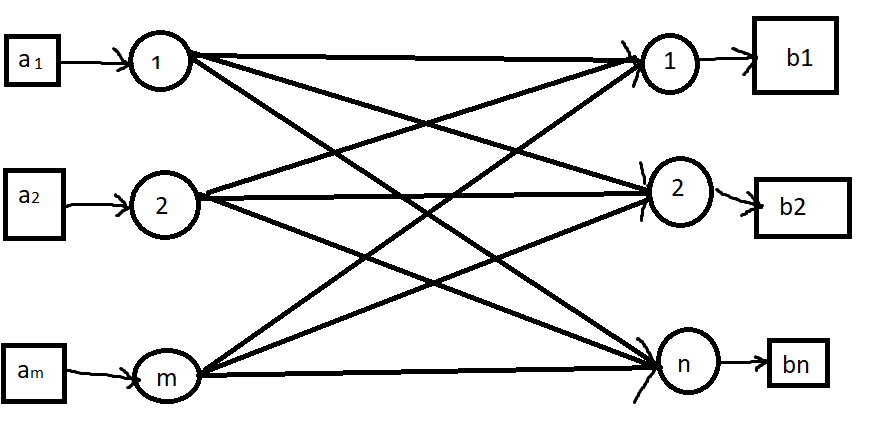
Используется 3 структуры: массив окраски в серый, массив окраски в черный, очередь/стек для порядка, шаг.





Транспортная задача

Транспортные задачи – специальный класс задач линейного программирования. Эти задачи описывают перемещение/перевозку какого-либо товара из пункта направления (место производства) в пункт назначения (склад или магазин). В такой задаче необходимо определить объем перевозок из пунктов отправления в пункты назначения с минимальной суммарной стоимостью перевозок.



M – количество поставщиков, n – количество потребителей.

I – индекс для поставщиков.

J – индекс для потребителей.

Ai (i = 1..m) – наличие продукции у каждого поставщика.

Bj (j=1..n) – потребность в продукции каждого потребителя.

Cij – стоимость доставки единицы продукции от I-того поставщика j-тому потребителю.

Необходимо найти план доставки продукции от поставщиков потребителям с минимальными транспортными издержками.

SUM(Ai),i=1..m = SUM (Bj), j=1..n – задача называется закрытой.

SUM(Ai),i=1..m != SUM (Bj), j=1..n – задача называется открытой (с нарушенным балансом).

На ЛК разбор закрытой задачи, на ЛР открытая.

Решение открытой задачи сводится к решению закрытой. Для этого при a<b добавляем фиктивного поставщика с запасом b-a. Если a>b, то добавляем фиктивного потребителя с заказом a-b. В обоих случаях соответствующие фиктивным объектам тарифы перевозок Cij полагаем равными нулю. В результате суммарная стоимость перевозов не изменяется.

Математическая модель

Надо найти Xij – сколько товаров нужно везти из 1 пункта в другой.

Целевая функция

Z=SUMSUM(Cij\*Xij) ->min.

Ограничения

SUM(Xij),j=1..n=ai,i=1..n

SUM(Xij),j=1..m=bj,j=1..n

Xij>=0, i=1..m,j=1..n

SUM(ai),i=1..m=SUM(bj),1..n

Решение транспортной задачи

Этапы:

1. Построение начального базисного решения? Метод северо-западного угла, метод наименьшей стоимости (минимального элемента), метод Фогеля.
2. Итеративный процесс поиска оптимального решения (метод потенциалов).

Общая транспортная задача с M пунктами отправления и N пунктами назначения имеет M+N ограничений в виде равенств. По одному на каждый пункт направления и назначения. Так как транспортная задача должна быть сбалансированной, то одно из этих равенств избыточно. Таким образом, транспортная задача имеет M+N+1 независимых ограничений. А начально базисное решение состоит из M+N+1 базисных переменных.

1. Метод наименьшей стоимости.

Пусть m=3 поставщиков продукции, n=4 потребителя продукции.

Запасы А=(15,25,10).

Потребность (5,15,15,15).

Затраты на перевозку

C=

|  |  |  |  |
| --- | --- | --- | --- |
| 10 | 2 | 20 | 11 |
| 12 | 7 | 9 | 20 |
| 4 | 14 | 16 | 18 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | Запасы |
| 1 | 10 | 2 | 20 | 11 | 15 |
| 2 | 12 | 7 | 9 | 20 | 25 |
| 3 | 4 | 14 | 16 | 18 | 10 |
| Потребность | 5 | 15 | 15 | 15 |  |

Вначале в таблице находим ячейку с наименьшей стоимостью. Переменная Х в этой ячейке присваивается наибольшее значение, допускаемое ограничениями по спросу и предложению. Если таким ячеей с минимальной стоимостью несколько, то выбор является произвольным. Далее вычеркивается соответствующий столбец или строка и корректируется спрос и предложение. Затем просматриваются невычеркнутые ячейки и выбирается новая ячейка с минимальной стоимостью.

Выбираем пересечение 1-2.

Уменьшаем поставки на необходимость. Остаются нули.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | Запасы |
| 1 | 10 | 2/15 | 20 | 11 | 0 |
| 2 | 12 | 7 | 9 | 20 | 25 |
| 3 | 4 | 14 | 16 | 18 | 10 |
| Потребность | 5 | 0 | 15 | 15 |  |

Далее выбирается 3-1.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | Запасы |
| 1 | 10 | 2/15 | 20 | 11 | 0 |
| 2 | 12 | 7 | 9 | 20 | 25 |
| 3 | 4 | 14 | 16 | 18 | 10 |
| Потребность | 5 | 0 | 15 | 15 |  |

X3,1=min(5,10) – минимум из потребности и наличия.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | Запасы |
| 1 | 10 | 2/15 | 20 | 11 | 0 |
| 2 | 12 | 7 | 9 | 20 | 25 |
| 3 | 4/5 | 14 | 16 | 18 | 5 |
| Потребность | 0 | 0 | 15 | 15 |  |

Далее выбирается 2-3.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | Запасы |
| 1 | 10 | 2/15 | 20 | 11 | 0 |
| 2 | 12 | 7 | 9/15 | 20 | 10 |
| 3 | 4/5 | 14 | 16 | 18 | 5 |
| Потребность | 0 | 0 | 0 | 15 |  |

Далее остаток докидывается 5ке.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1 | 2 | 3 | 4 | Запасы |
| 1 | 10 | 2/15 | 20 | 11 | 0 |
| 2 | 12 | 7 | 9/15 | 20/10 | 0 |
| 3 | 4/5 | 14 | 16 | 18/5 | 0 |
| Потребность | 0 | 0 | 0 | 0 |  |

Так как переменных должно быть 6, то еще одну докидываем случайно.

X3,1=5, X1,2=15, X2,3=15, X3,4=5, X2,4=10, X1,4=0.

Первое допустимое решение на переменных указанных выше.

Значение функции цели

X=15\*2+0\*11+15\*9+10\*20+5\*4+5\*18=475.

Получено только базисное, стартовое решение, которое не гарантирует наилучшее.

1. Метод потенциалов

В методе потенциалов каждой строке i и каждому столбцу j транспортной таблицы ставятся в соответствии числа (потенциалы) ui(поставщики) и vj (потребители). Для каждой базисной переменной xij потенциалы ui и vj удовлетворяют уравнению.

ui+vj=cij

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | 1, v1 | 2, v2 | 3, v3 | 4, v4 | Запасы |
| 1, u1 | 10 | 2/15 | 20 | 11 | 0 |
| 2, u2 | 12 | 7 | 9/15 | 20/10 | 0 |
| 3, u3 | 4/5 | 14 | 16 | 18/5 | 0 |
| Потребность | 0 | 0 | 0 | 0 |  |

Определяем потенциалы для всех базисных переменных (6 переменных).

u1+v2=2.

u1+v4=11.

u2+v3=9.

u2+v4=20.

u3+v1=4.

u3+v4=18.

6 уравнение при 7 неизвестных. Обычно заменяют u1=0,

тогда v2=2,v4=11,u2=9,u3=7,v1=-3,v3=0.

Для улучшения предыдущего решения необходимо ввести в базис новую переменную.

Составим таблицу для свободных клеток

|  |  |
| --- | --- |
| X11 | u1+v1-c11=-13 |
| X13 | u1+v3-c13=-20 |
| X21 | u2+v1-c21=-6 |
| X22 | u2+v2-c22=4 |
| X32 | u3+v2-c32=-12 |
| X33 | u3+v3-c33=-9 |

Вводимой в базис переменной будет та, у которой наибольшее положительное значение.

При двух одинаковых значениях под выбор выбирается любая.

Если все отрицательные и нули, нужно попробовать ноль и убедиться, что улучшения нет.

Сейчас используется 7 переменных, а можно только 6.

Определив вводимую в базис переменную, теперь определим исключаемую из базиса переменную. Необходимо определить максимально возможное количество груза, перевозимого по маршруту 2-2:

При этом должно выполняться ограничение на спрос и предложение и ни по какому маршруту не должны производиться перевозки с отрицательным объемом грузов. Для этого необходимо построить замкнутый цикл, который начинается и заканчивается в искомой ячейке 2-2. Данный цикл состоит из последовательности горизонтальных и вертикальных отрезков (но не диагональных), соединяющих ячейки текущих базисных переменных и ячейку, соответствующую вводимым переменным.

X2,2- начало и конец цикла, используя только базисные переменные.

Для того, чтобы удовлетворять ограничению по спросу и предложению необходимо поочередно отнимать и прибавлять максимальное количество груза к значениям базисным переменных, расположенных в угловых ячейках цикла. Направление обхода цикла (по или против часовой) не имеет значения.

Плюс ставится на ячейке, которую вводят в базис.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Потребители  Поставщики | 1 | 2 | 3 | 4 | Запасы |
| 1 | 10 | 2|15 | 20 | 11|0 | 15 |
| 2 | 12 | 7 | 9|15 | 20|10 | 25 |
| 3 | 4|5 | 14 | 16 | 18|5 | 10 |
| Потребность | 5 | 15 | 15 | 15 | 50 |

1-2 (минус)

2-2(плюс)

2-4(минус)

1-4(плюс)

Итого 10 товаров перемещено по описанному выше циклу.

Новая сумма 435, разница с предыдущим решением 40.

Далее ищем аналогично улучшение решения.

Находим потенциалы.

u1+v2=2

u1+v4=11

u2+v2=7

u2+v3=9

u3+v1=4

u3+v4=18

Уравнений 6 неизвестных 7.

u1=0> по договоренности.

v2=2,v4=11

u2=5

v3=4

u3=7

v1=-3

Далее вычисляются потенциалы для небазисных ячеек.

X11=u1+v1-c11=-13

X13=u1+v3-c13=-16

X21=u2+v1-c21=-10

X24=u2+v4-c24=-4

X32=u3+v2-c32=-5

X33=u3+v3-c33=-5

Положительных значений нет, поэтому улучшений в решении нет.

Процесс поиска наилучшего решения прекращается в том случае, когда в предыдущей таблице нет положительных значений либо не с одним из положительных значений невозможно правильно построить цикл.

Сетевое планирование (математические основы)

Минимальные покрывающие деревья

Пусть G=(V,E) – связный неориентированный граф, а

W(i,j) = {w9i,j)принадлежит R, (I,j)принадлежит E}{+бесконечность, (i,j) не принадлежит E} – весовая функция, заданная на множестве ребер (I,j) и определяющая длину каждого ребра. Считаем, что W(I,j)=W(j,i).

Связный подграф G’=(V,E’) графа G=(V,E) являющийся деревом и содержащий все его вершины, называется покрывающим деревом или остовным деревом.

Покрывающее дерево T графа G, имеющее минимальную сумму длин его ребер, называется минимальным покрывающим деревом, или минимальным остовным деревом.

Построение минимального остовного дерева в известных алгоритмах осуществляется поэтапно. Начинается построение с пустого множества ребер A=пустое множество. На каждом этапе множество А пополняется одним ребром, причем так, что множество ребер А всегда остается подмножеством ребер искомого минимального остовного дерева. Задача сводится к поиску необходимого ребра (обычно называется безопасным ребром) на каждом шаге алгоритма.

Пусть V – множество вершин графа G=(V,E) тогда разбиением этого множества будем называть семейство множеств V1,V2,V3..Vk, k = 1..V, обладающее следующими свойствами:

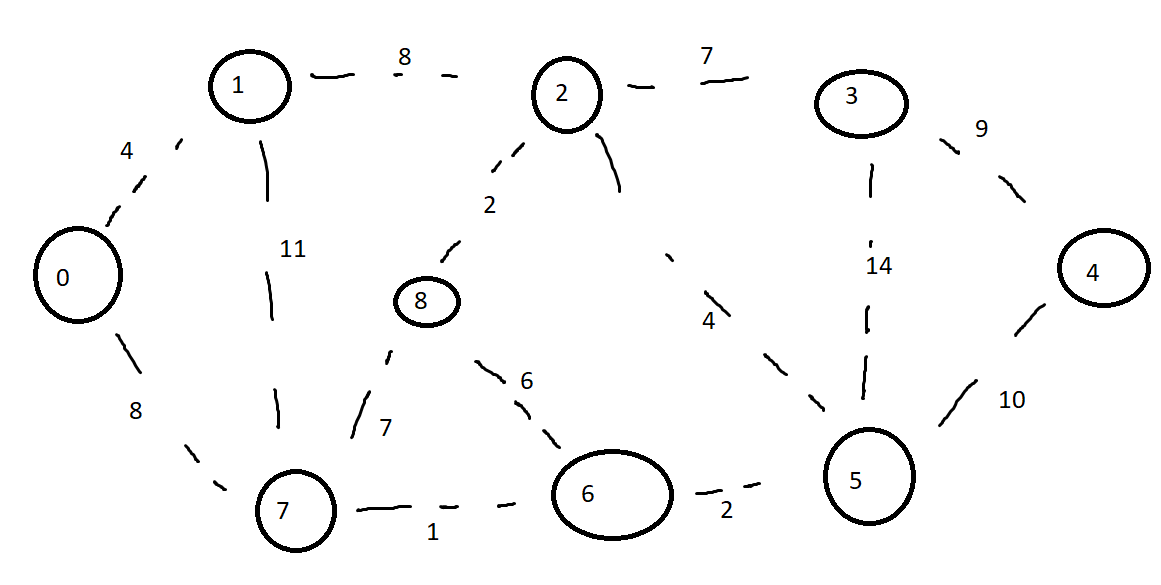
U(i=1..k)Vi=V,Vi пересечение Vj=пустое множество, i!=j, i,j,k=1..V.

Пусть S и V\S – разбиение множества вершин связного неориентированного графа G=(V,E). Если првоести линию, разделяющую множества S и V\S, то эта линия пересечет ребра, концевые вершины которых лежат в разных подмножествах разбиения. Множество ребер, которые пересекла линия, разделяющая множества вершин, называется разрезом графа. Множество ребер A из V согласовано с разрезом (S,V\S) если A пересекая (S,V\S)=пустому множеству.

Пусть {e1,e2,..em} – множество всех ребер, принадлежащих разрезу (S,V\S) тогда ребро Ej имеющее минимальную длину – легкое ребро разреза.

Теорема. Пусть G=(V,E) связный неориентированный граф и T=(V,R) – минимальное остовное дерево этого графа. Пусть множество ребер A из R согласовано с некоторым разрезом (S,V\S) а е-легкое ребро этого разреза. Тогда е является безопасным ребром для множества А.

Если е – ребро, имеющее минимальную длину в неориентированном связном графе, то оно будет ребром одного из минимальным остовных деревьев.

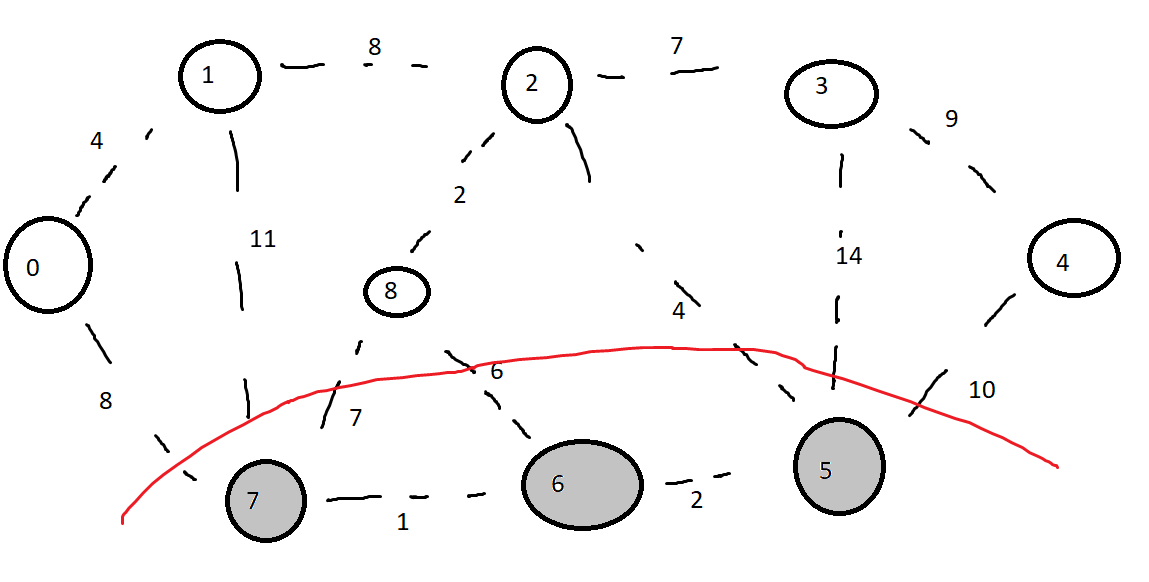


Изначально выбирается ребро с минимальной весовой функцией

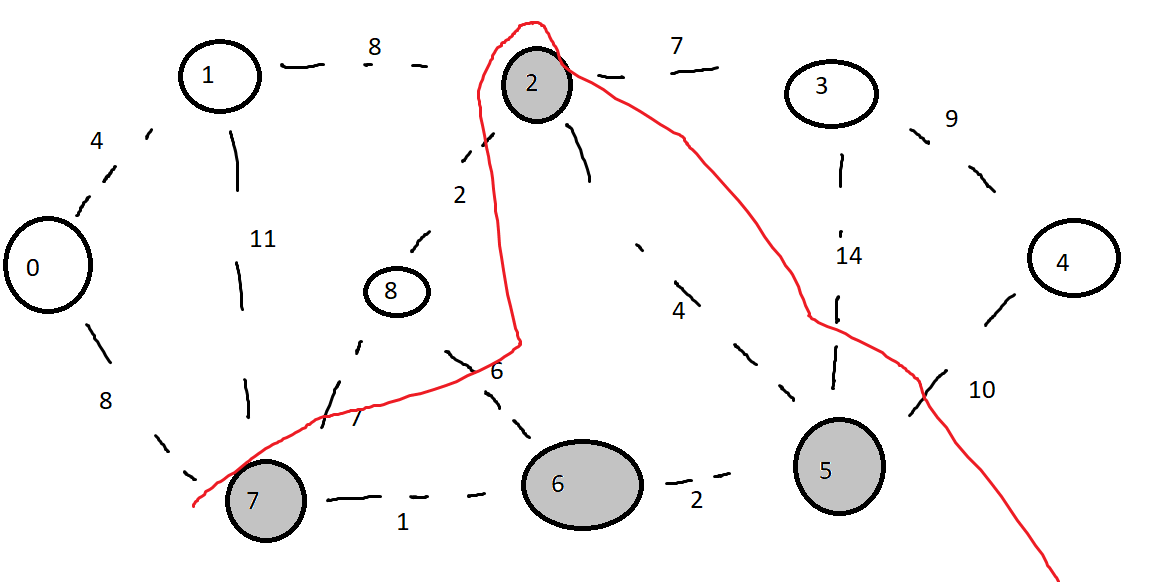


В разрезе участвуют ребра с весами 8, 11, 7, 6 ,2. Безопасным ребром будет ребро с весом 2.

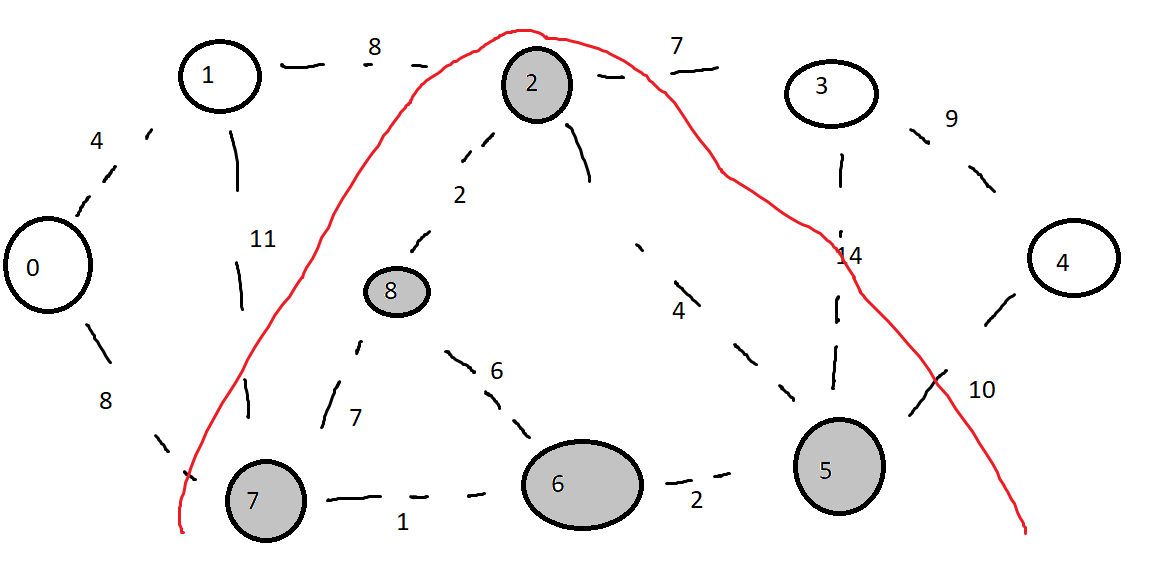
Включаем в граф вершину с номером 5.

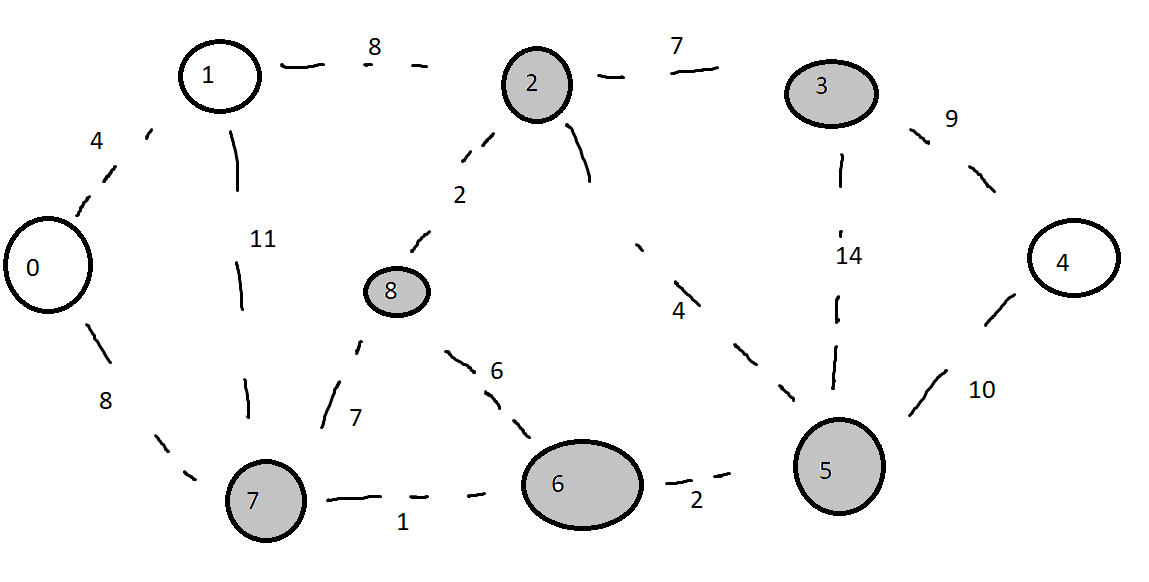


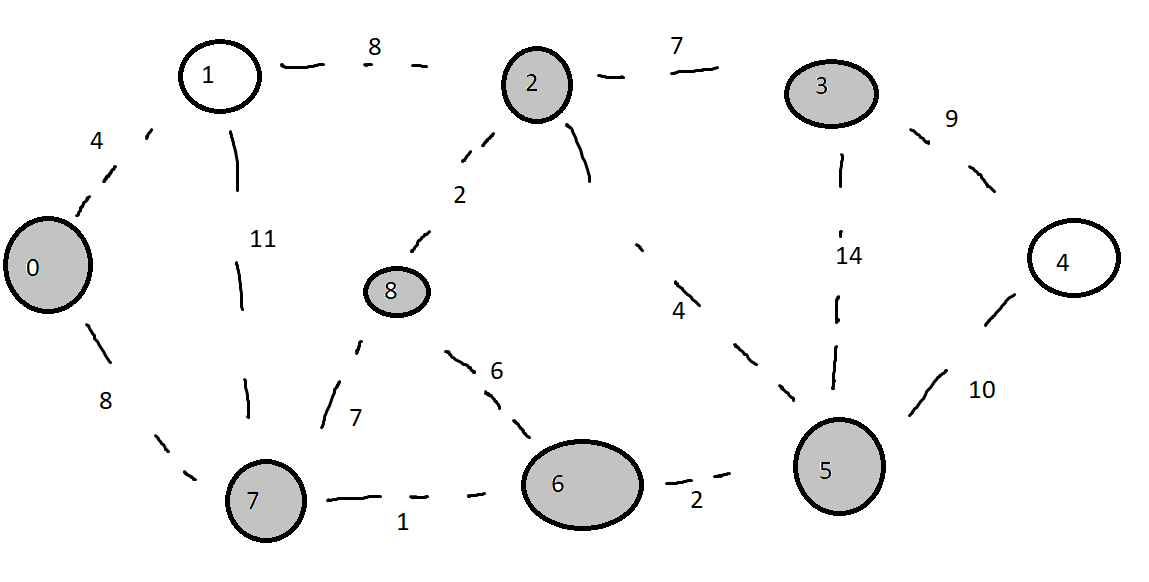
Далее добавляем вершину с весом ребра 4 – это вершина 2.

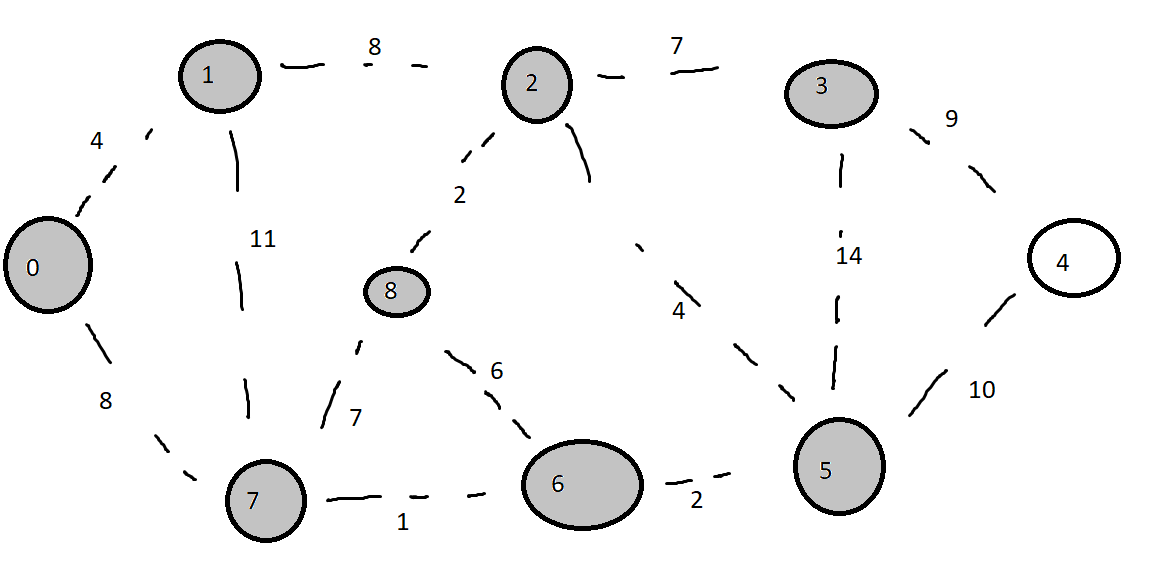


Далее вершина 8 (выбираем на свое усмотрение если 2+ варианта доступно.

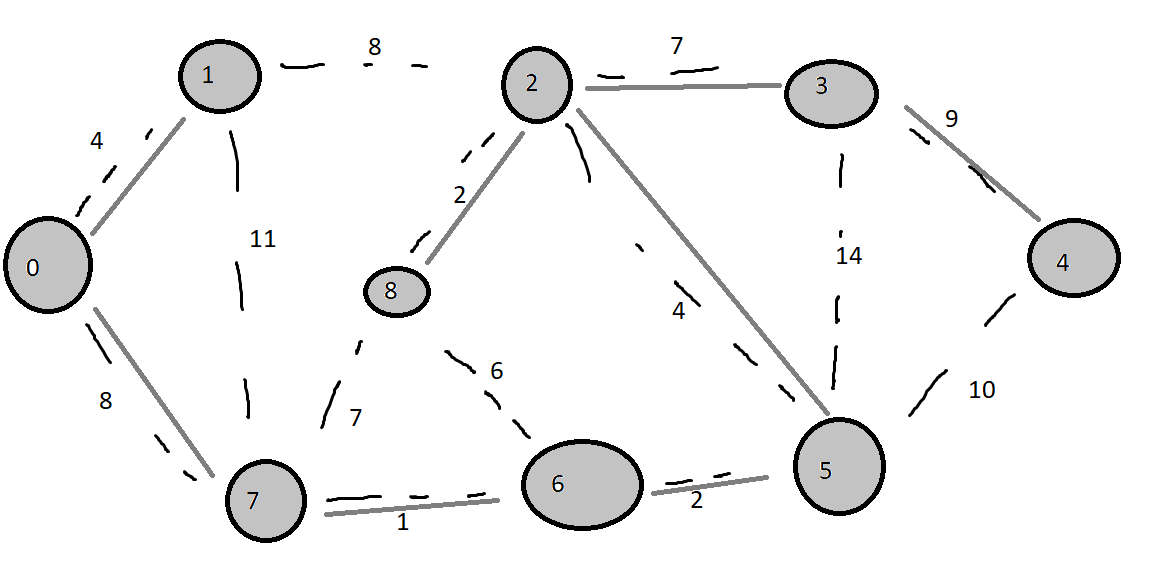




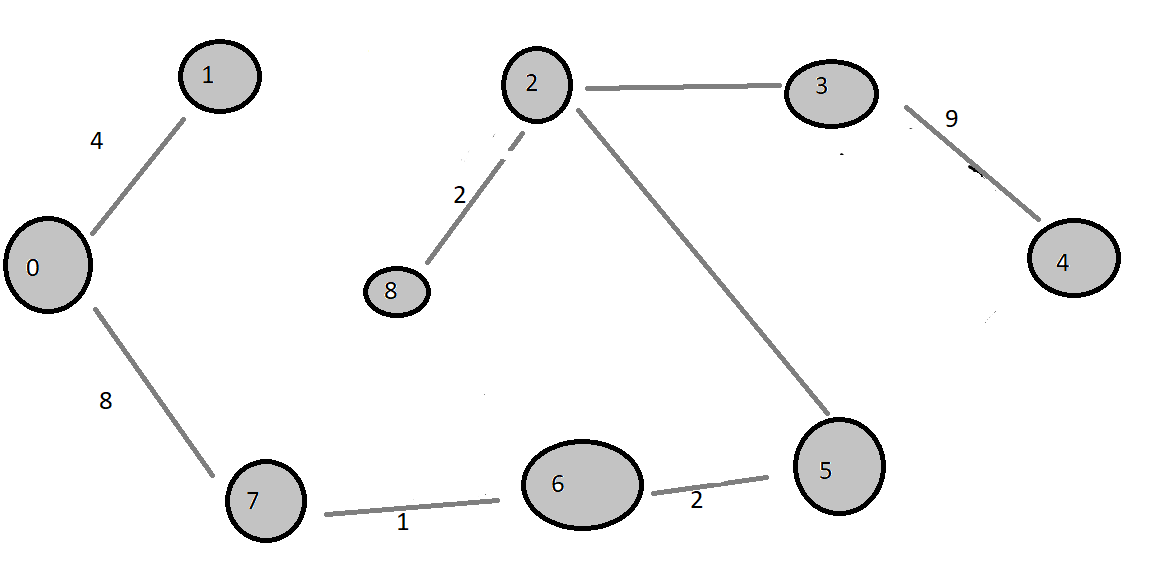




Итого:



Минимальное остовное дерево с ценой 37:



Алгоритм Крускала

Первоначально граф разбивается на максимальное количество подграфов, каждый из которых является деревом. Очевидно, что каждое такое дерево будет представлять собой одну вершину графа, а общее число подграфов будет |V|.

Далее из множества Е ребер графа G поочередно в порядке возрастания длины выбираются ребра. Возможны 2 случая:

1. Концевые вершины лежат в разных подграфах разбиения.
2. Обе концевые вершины лежат в одном подграфе разбиения.

В первом случае из двух подграфов, которые можно соединить выбранным ребром, образуется общий включающий все вершины и ребра этих подграфов, а также новое связующее звено. Очевидно, что такое объединение не может образовать циклы, а, следовательно, объединенный граф также является деревом.

Во втором случае не выполняется никаких новых построений.

Из условия связности исходного графа очевидно, что итогом работы такого алгоритма будет дерево, связывающее все вершины графа (остовное дерево).

Минимальность остовного дерева следует из того, что на каждом его шаге выбиралось безопасное ребро для совокупности ребер двух подграфов.

1. Разбить граф на максимальное количество подграфов. В данном случае 9 отдельных вершин.
2. Выбрать ребро минимальной длины. Если две вершины в разных подграфах, то ребро добавляют в общий граф. Если существуют 2 и более варианта выбора, то выбираем любой из них.

Итоговая сумма получилась 37, как и в прошлом алгоритме.

Алгоритм Прима

Построение минимального остовного дерева начинается с выбора произвольной вершины исходного неориентированного связного графа.

На последующих шагах рассматриваются все окрашенные вершины и анализируются все ребра исходного графа, у которых одна концевая вершина окрашена, а другая нет. Среди таких ребер выбирается ребро с наименьшей длиной. Неокрашенная вершина этого ребра окрашивается, а само оно добавляется в формируемое минимальное остовное дерево.

Алгоритм заканчивает свою работу, когда все вершины графа станут окрашенными. Сформированное множество выбранных ребер будут составлять искомое минимальное остовное дерево.

На лекции первый раз выбрали вершину 8. Далее минимальное прилегающее ребро 2-8. Следующих шагов добавили ребро 2-5.

Потоки в сетях

Задача о максимальном потоке

Сеть – ориентированный граф G=(V,E), каждому ребру которого поставлено в соответствии число c(vi,vj)>=0, называемое пропускной способностью, а также выделено две вершины v0-исток, vn-1-сток, n=|V|.

Исток – функция f(vi,vj),(vi,vj)из E обладающая тремя свойствами:

1. F(vi,vj)<=c(vi,vj).
2. F(vi,vj)=-F(vj,vi) (кососимметричность).
3. Sum (v из V) f(u,v)=0, u из V- {v0,vn-1}

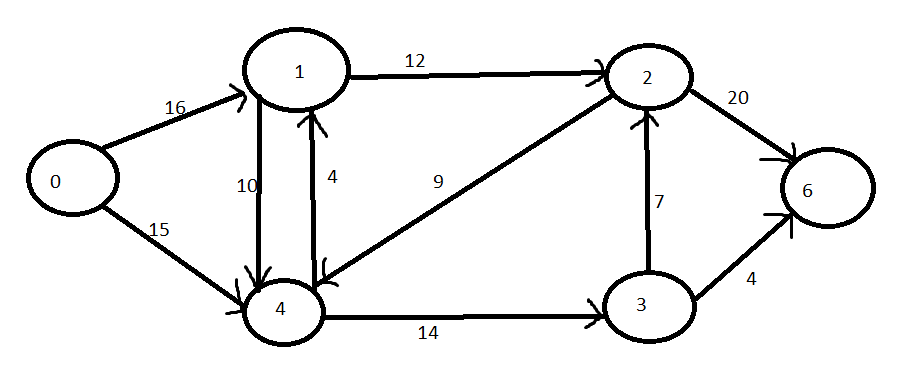
Разрез (S,T) сети G – разбиение множества V на две части S,T такие что их пересечение равно пустому множеству.

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

Минимальный разрез сети – разрез с минимальной пропускной способностью.

Поток через разрез – сумма соединений от S к T – сумма соединений от T к S.

Теорема Форда-Фалкерсона: в любой сети максимальная величина потоки из истока S в сток T равна минимальной пропускной способности разреза отделяющего S от T.



В разрез вначале включаются все возможные варианты, например, 0-1-2,0-1-3,0-3,0-2-3, 0-1-4, 1-4 и т.д. Выбирается минимальное пропускное значение.

Наименьшее значение 23 является ответом при разрезе 0-1-3-4.

На практике более 1 истока и стока.

Алгоритм Форда-Фаркинсона.

Основан на остаточных сетях и дополняющих путях.

Остаточные сети – пусть дана сеть и поток в ней. В этом случае остаточная сеть состоит из тех ребер, поток по которым можно увеличить.

Остаточное ребро не обязано быть ребром искомой сети. Такие ребра появляются, когда имеется поток вещества в обратном направлении. Этот поток можно уменьшить, так как он переправляет данные в обратную сторону.

У остаточных сетей есть следующее свойство: если в остаточной сети существует поток F, то прибавив его к исходному потоку в сети, мы также получим поток, удовлетворяющий всех требованиям, но который больше предыдущего.

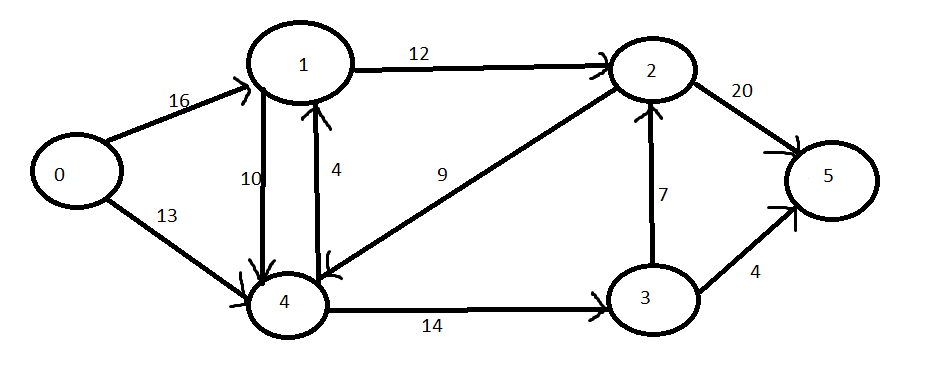
Дополняющие пути: назовем дополняющим путем простого пути из истока в сток в остаточной сети. Из определения остаточной сети следует, что по всем ребрам дополняющего пути можно переслать еще сколько-то вещества, не превысив их пропускную способность. Величину наибольшего потока, который можно переслать по дополняющему пути назовем остаточной пропускной способностью пути.

Она равна значению минимального остаточного ребра, входящего в данный путь.

Шаги:

1. Обнуляем все потоки. Остаточная сеть изначально совпадет с исходной сетью. (берем сеть трубопроводов).
2. В остаточной сети находим любой путь из истока в сток. Если такого пути нет, останавливаемся и завершаем алгоритм.
3. Пускаем через найденный путь (он называется увеличивающим путем или увеличивающей цепью) максимально возможный поток:
   1. На найденном пути в остаточной сети ищем ребро с пропускной способностью Cmin.
   2. Для каждого ребра на найденном пути увеличиваем поток на Cmin, а в противоположном ему – уменьшаем на Cmin.
   3. Модифицируем остаточную сеть. Для всех ребер на найденном пути, а также для противоположных им ребер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.
4. Идем к пункту 2.

Выбираем любой путь из истока в сток.

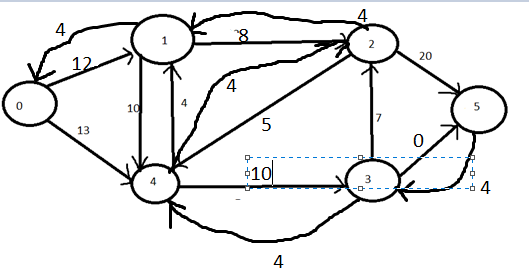


Выбираем путь 0-1-2-4-3-5. Максимальная пропускная способность – 4.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 4 | 0 | 0 | 0 | 0 |
| 1 | -4 | 0 | 4 | 0 | 0 | 0 |
| 2 | 0 | -4 | 0 | 0 | 4 | 0 |
| 3 | 0 | 0 | 0 | 0 | -4 | 4 |
| 4 | 0 | 0 | -4 | 4 | 0 | 0 |
| 5 | 0 | 0 | 0 | -4 | 0 | 0 |

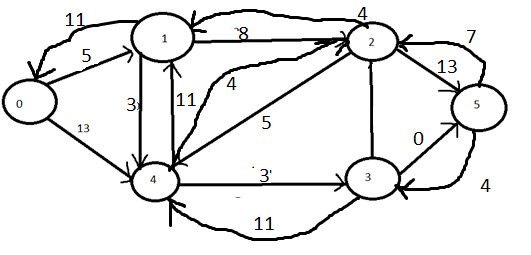
Путь 0-1. Было 16. Добавилась противоположная дуга с пропускной способностью 4. Итого пропускная способность стала равной 12. 1-2 было 12 стало 8. 2-4 было 9 стало 5. 4-3 было 14 стало 10.3-5 была дуга 4 литра. Она исчезла – стала равна 0 и теперь там дуга 4 в обратную сторону.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 11 | 0 | 0 | 0 | 0 |
| 1 | -11 | 0 | 4 | 0 | 7 | 0 |
| 2 | 0 | -4 | 0 | -7 | 4 | 7 |
| 3 | 0 | 0 | 7 | 0 | -11 | 4 |
| 4 | 0 | -7 | -4 | 11 | 0 | 0 |
| 5 | 0 | 0 | -7 | -4 | 0 | 0 |



Дуга имеющая наименьшую пропускную способность теперь 3-2, вес 7.

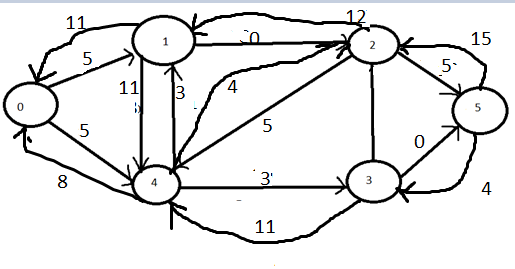
Для всех дуг которые начинают использоваться добавляется/отнимается 7 в зависимости от направления дуг.



Теперь идет 8ка по пути.

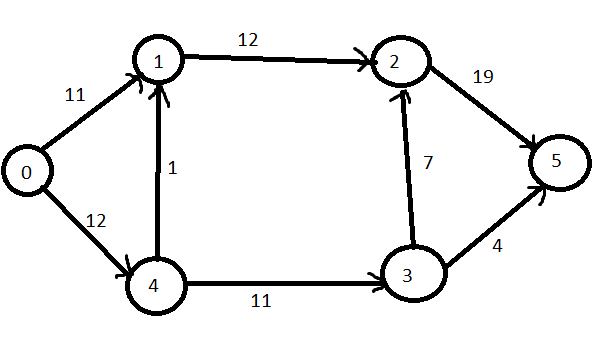
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 11 | 0 | 0 | 8 | 0 |
| 1 | -11 | 0 | 12 | 0 | -1 | 0 |
| 2 | 0 | -12 | 0 | -7 | 4 | 15 |
| 3 | 0 | 0 | 7 | 0 | -11 | 4 |
| 4 | -8 | 1 | -4 | 11 | 0 | 0 |
| 5 | 0 | 0 | -15 | -4 | 0 | 0 |

Далее



|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| 0 | 0 | 11 | 0 | 0 | 12 | 0 |
| 1 | -11 | 0 | 12 | 0 | -1 | 0 |
| 2 | 0 | -12 | 0 | -7 | 0 | 19 |
| 3 | 0 | 0 | 7 | 0 | -11 | 4 |
| 4 | -12 | 1 | -0 | 11 | 0 | 0 |
| 5 | 0 | 0 | -19 | -4 | 0 | 0 |

Алгоритм прекращает и предыдущая матрица – ответ задачи.



Проверить – сумма входных потоков равна сумме исходящих потоков.

Линейное программирование

Сущность линейного программирования состоит в нахождении наибольшего или наименьшего значения некоторой функции при определенном наборе ограничений, налагаемых на аргументы.

ЛП – метод математического моделирования, разработанный для оптимизации использования ограниченных ресурсов. При этом целевые функции и ограничения строго линейны.

Математическая модель любой задачи ЛП включает в себя:

1. Переменные, которые следует определить;
2. Целевую функцию, подлежащую оптимизации;
3. Систему ограничений в форме линейных уравнений и неравенств.

Компания производит типографскую краску двух цветов черную и синюю из сырья двух типов М1 и М2.

|  |  |  |  |
| --- | --- | --- | --- |
|  | Расход сырья | | максимально возможный ежедневный расход сырья |
|  | Черная | Синяя |
| Сырье М1 | 6 | 4 | 24 |
| Сырье М2 | 1 | 2 | 6 |
| Доход на единицу краски | 5 | 4 |  |

Отдел маркетинга ограничил ежедневное производство синей краски до 2 тонн из-за отсутствия спроса. Кроме этого поставил условие чтобы ежедневное производство синей краски не превышало более чем на тонну производство черной.

Компании требуется определить оптимальное соотношение между видами выпускаемой продукции для максимизации общего ежедневного дохода.

Математическая модель

Х1 = черная краска, х2 = синяя краска.

Целевая функция обычно Z.

Z=5\*x1+4\*x2->max

0<=X1<=4

0<=X2<=3

6X1+4X2<=24

X1+2X2<=6

-X1+X2<=1

X2<=7/3=2

Графическое решение

Этапы:

1. Построение пространства допустимых решений, удовлетворяющих всем ограничениям модели.
2. Нахождение оптимального решения среди всех точек пространства допустимых решений.

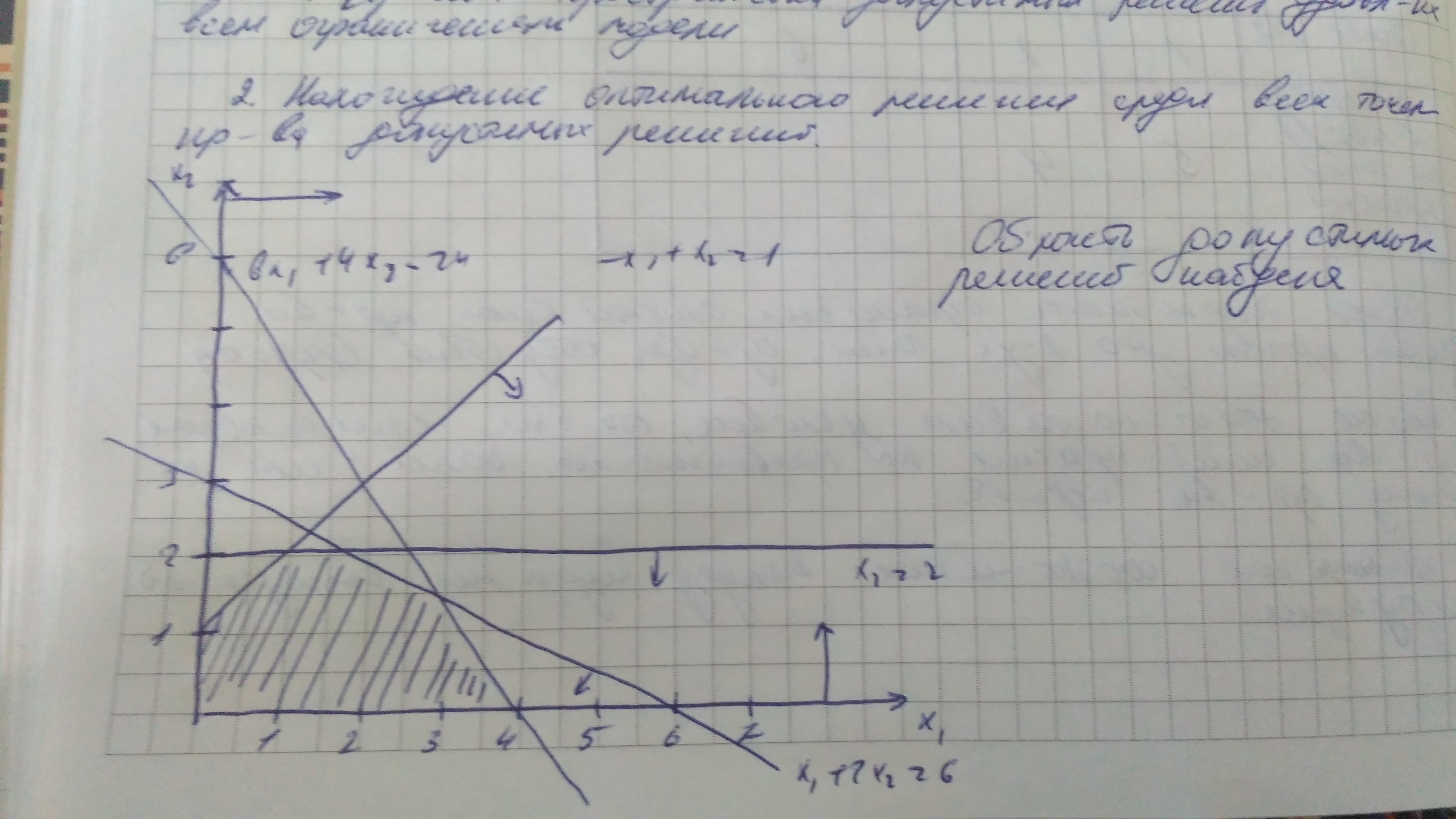
График и отсекать недопустимые решения.

Любая точка внутри или на границе области является одним из вариантов решения. Но поскольку точек бесконечное число необходима процедура отыскания оптимального решений.

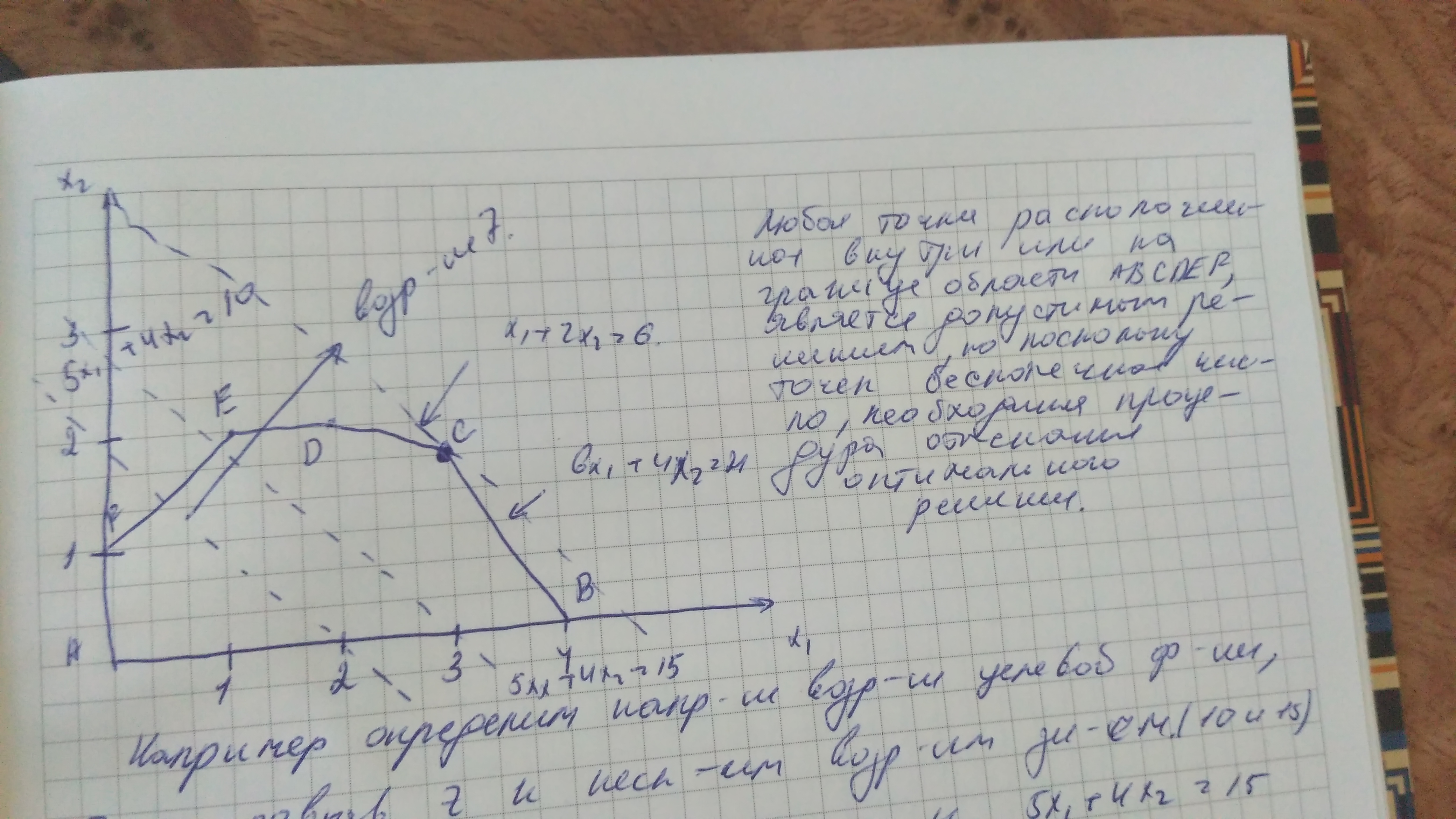
Например определим направление возрастания целевой функции Z, приравняв Z к нескольким возрастающим значениям (10 и 15).

5x1 + 4x2 = 10.

5x1+4x2=15.



Тетрадь РЭВ



Тетрадь РЭВ

Симплекс метод решения задач линейного программирования

Последовательность действий, выполняемых в симплекс методе:

1. Преобразование задачи в стандартную форму.
2. Находится начальное допустимое базовое решение.
3. На основе условия оптимальности определяется вводимая переменная. Если 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.

Преобразование задачи в стандартную форму.

1. Преобразовать свободные переменные в неотрицательные.
2. Преобразовать неравенство в равенство.
3. Преобразовать целевую функцию.

Пример Z=2x1+3x2+5x3->max

Где

X1+x2-x3>=-5,

-6x1+7x2-9x3<=4,

x1+x2+4x3=10,

x1>=0,

x2>=0,

x3 свободная (без ограничений)

x3=x3+-x3-,x3+>=0,x3->=0

Вычтем из левой части первого неравенства дополнительную переменную x4 и умножим все неравенство на -1 для того, чтобы правая часть равенства стала положительной.

X1+x2-x3-x4=-5

X4-x1-x2+x3=5

Добавим дополнительную переменную х5 к левой части второго неравенства.

Третье равенство остается без изменений.

Выполняет замену свободной переменной.

Z=2x1+3x2+5x3+-5x3🡪max

Ограничения в стандартной форму:

-x1-x2+x3+-x3-+x4=5

-6x1+7x2-9x3++9x3-+x5=4

x1+x2+4x3+-4x3‑=10

x1>=0, x2>=0, x3+>=0, x3->=0, x4>=0, x5>=0

Вернемся к задаче о краске. Приведем задачу к стандартной форме:

Z=5x1+4x2->max, где

6x1+4x2<=24,

X1+2x2<=6

X2-x1<=1

X2<=2

X1>=0, x2>=0

Ограничения преобразовываются в:

6x1+4x2+s1=24,

x1+2x2+s2=6,

-x1+x2+s3=1,

x2+s4=2,

x1>=0, x2>=0, s1>=0, s2>=0, s3>=0, s4>=0

Z-5x1-4x2-0s1-0s2-0s3-0s4=0

Для решения задачи ее следует представить в виде следующей таблице:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| базис | Z | x1 | x2 | s1 | s2 | s3 | s4 | Решение |
| Z | 1 | -5 | -5 | 0 | 0 | 0 | 0 | 0 |
| s1 | 0 | 6 | 4 | 1 | 0 | 0 | 0 | 24 |
| s2 | 0 | 1 | 2 | 0 | 1 | 0 | 0 | 6 |
| s3 | 0 | -1 | 1 | 0 | 0 | 1 | 0 | 1 |
| s4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 2 |

Какая переменная из небазисных дает наибольший прирост целевой функции Z? Именно ее следует ввести в состав базисных. У нас базисные переменные s1, s2, s3, s4, небазисные x1, x2.

Включаемой в базис переменной будет х1, так как целевая функция 5x1+4x2 и x1 влияет сильнее. Исключаемая переменная вычисляется по следующей таблице:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| базис | х1 | Решение | Точка пересечения | Комментарий |
| s1 | 6 | 24 | 24/6=4 | минимум |
| s2 | 1 | 6 | 6/1=6 |  |
| s3 | -1 | 1 | 1/(-1)=-1 | не подходит |
| s4 | 0 | 2 | 2/0=INF | не подходит |

Из таблицы выбирается наименьшее положительное значение точки пересечения. Исключаемой из базиса будет переменная s1.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| базис | Z | x1 | x2 | s1 | s2 | s3 | s4 | Решение |
| Z | 1 | -5 | -5 | 0 | 0 | 0 | 0 | 0 |
| s1 | 0 | 6 | 4 | 1 | 0 | 0 | 0 | 24 |
| s2 | 0 | 1 | 2 | 0 | 1 | 0 | 0 | 6 |
| s3 | 0 | -1 | 1 | 0 | 0 | 1 | 0 | 1 |
| s4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 2 |

Строка s1- ведущая строка, ведущий столбец – x1.

Первый этап – вычисление элементов ведущей строки, для этого необходимо заменить x1 на s1 и все элементы ведущей строки теперь x1 делим на 6.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| базис | Z | x1 | x2 | s1 | s2 | s3 | s4 | Решение |
| Z | 1 | 0 | -2/3 | 5/6 | 0 | 0 | 0 | 20 |
| x1 | 0/6 | 6/6 | 4/6 | 1/6 | 0/6 | 0/6 | 0/6 | 24/6=4 |
| s2 | 0 | 0 | 4/3 | -1/6 | 1 | 0 | 0 | 2 |
| s3 | 0 | 0 | 5/3 | 1/6 | 0 | 1 | 0 | 5 |
| s4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 2 |

Второй этап – вычисление остальных элементов, включая Z

Zстрока вычисляется по формуле

Новая строка = текущая Z строка-(пересечение Z строки с ведущим столбцом) \* (новая ведущая строка)

(1,-5,-4,0,0,0,0,0)-(-5)\*(0,1,2/3,1/6,0,0,0,4)=(1,0,-2/3,5/6,0,0,0,20)

Вычисление строки s2:

Новая строка = текущая строка-(пересечение строки с ведущим столбцом)\*(новая ведущая строка)

(1,-5,-4,0,0,0,0,0)-(-5)\*(0,1,2/3,1/6,0,0,0,4)=(1,0,-2/3,5/6,0,0,0,20)

(0,1,2,0,1,0,0,6)-1\*(0,1,2/3,1/6,0,0,0,4)=(0,0,4/3,-1/6,1,0,0,2)

Новое базисное уравнение

X1=4

S2=2

S3=5

S4=2

Новое допустимое решение – 20(строка Z)

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| базис | Z | x1 | x2 | s1 | s2 | s3 | s4 | Решение |
| Z | 1 | 0 | -2/3 | 5/6 | 0 | 0 | 0 | 20 |
| x1 | 0 | 1 | 2/3 | 1/6 | 0 | 0 | 0 | 4 |
| s2 | 0 | 0 | 4/3 | -1/6 | 1 | 0 | 0 | 2 |
| s3 | 0 | 0 | 5/3 | 1/6 | 0 | 1 | 0 | 5 |
| s4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 2 |

Новое уравнение:

Z-2/3x2+5/6s1=20

Z=2/3x2-5/6s1+20

Наибольший рост обеспечит переменная x2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| базис | х2 | Решение | Точка пересечения | Комментарий |
| x1 | 2/3 | 4 | 4/(2/3)=6 |  |
| s2 | 4/3 | 2 | 2/(4/3)=3/2 | минимум |
| s3 | 5/3 | 5 | 5/(5/3)=3 |  |
| s4 | 1 | 2 | 2/1=2 |  |

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| базис | Z | x1 | x2 | s1 | s2 | s3 | s4 | Решение |
| Z | 1 | 0 | -2/3 | 5/6 | 0 | 0 | 0 | 20 |
| x1 | 0 | 1 | 2/3 | 1/6 | 0 | 0 | 0 | 4 |
| s2 | 0 | 0 | 4/3 | -1/6 | 1 | 0 | 0 | 2 |
| s3 | 0 | 0 | 5/3 | 1/6 | 0 | 1 | 0 | 5 |
| s4 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 2 |



|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| базис | Z | x1 | x2 | s1 | s2 | s3 | s4 | Решение |
| Z | 1 | 0 | 0 | 3/4 | 1/2 | 0 | 0 | 21 |
| x1 | 0 | 1 | 0 | 1/4 | -1/2 | 0 | 0 | 3 |
| x2 | 0 | 0 | 1 | -1/8 | ¾ | 0 | 0 | 3/2 |
| s3 | 0 | 0 | 0 | 3/8 | -5/4 | 0 | 1 | 5/2 |
| s4 | 0 | 0 | 0 | 1/8 | -3/4 | 0 | 1 | ½ |

Отрицательный элементов в строке Z нет поэтому функция достигла максимума, решением будет 21, а перебор 2 переменных для изменения базиса нет смысла, поскольку их уже пробовали.

X1=3

X2=3/2

Z=5x1+4x2=15+6=21.

Задачи нелинейного программирования

Определение: нелинейное программирование – раздел математического программирования, изучающий методы решения экстремальных задач с нелинейной целевой функцией и/или областью допустимых решений, определенный нелинейными ограничениями.

В общем виде задачу нелинейного программирования можно сформулировать так:

F(x)->min(max)

При условии g(x)<=0,

Где х – вектор искомых переменных, FFF(x) – целевая числовая функция; п(ч) – вектор-функция системы ограничений.

При этом бывают разные случаи:

1. Целевая функция – нелинейная, а ограничения линейны,
2. Целевая функция – линейная, а ограничения (хотябы одно из них) нелинейные,
3. Целевая функция и ограничения нелинейные.

Задачи условной оптимизации нелинейного программирования бывают двух типов: когда в ограничениях (2) имеют место:

1. Знаки равенства.
2. Знаки неравенства.

Среди большого числа вычислительных алгоритмов нелинейного программирования значительное место занимают:

1. Различные варианты градиентных методов (методов проекции градиента, метод условного градиента и др.)
2. Методы штрафных функций
3. Методы барьерных функций
4. Метод модифицированных функций Лагранжа и др.

Универсального метода, позволяющего находить наиболее эффективным способом решение любой нелинейной задачи не существует. Поэтому, учитывая специфику задачи каждый раз подбирают наиболее подходящий метод или алгоритм решения.

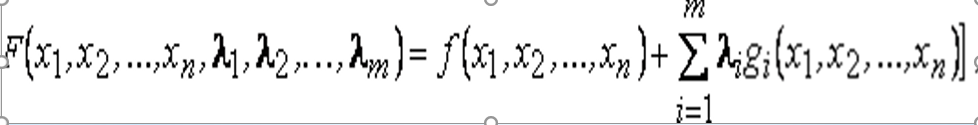
Рассмотрим частный случай общей задачи нелинейного программирования. Предполагаем, что система ограничений содержит только уравнения. Кроме того, отсутствуют условия неотрицательности переменной. Кроме этого, f(x) и g(x) функции непрерывные вместе со своими частными производными. Поскольку ограничения в задаче заданы уравнениями, то для ее решения воспользуемся классическим методом отыскания условного экстремума функции нескольких переменных – методом Лагранжа. Для этого введем набор переменных (множителей Лагранжа) и составим функцию Лагранжа.

Общий вид функции Лагранжа:

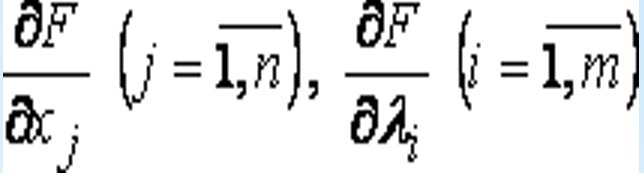
L(x, λ) = f(x) + Sum(i=1..m) λiφi(x)

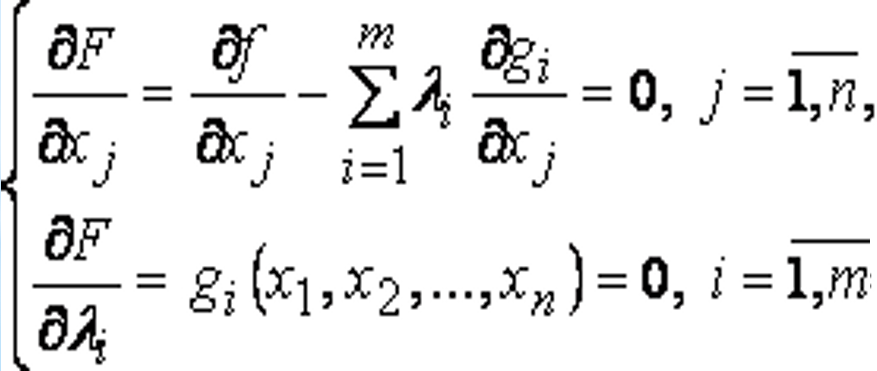
где *L*(*x*, λ) — лагранжиан; *f*(*x*) — целевая функция; λ*i* (*i* = 1, 2, ..., *m*) – множители Лагранжа; *m* — число ограничений *φi*(*x*).

Более подробная роспись функции Лагранжа:



Далее находят частные производные. Вначале все по х, далее все по λ. И рассматривают систему n+m уравнений





n+m неизвестных (x1,x2,..xn, λ1, λ2,.. λm). Уравнений будет столько, сколько неизвестных.

Решив данную систему уравнений получают все точки, в которых функция может иметь экстремальные значения. Метод множителей Лагранжа имеет ограниченное применение, поскольку предыдущая система уравнений, как правило, имеет несколько решений.

Пример:

Найти точку условного экстремума функции:

F(x) = x1x2+x2\*x3

При ограничениях:

x1+x2=2 и x2+x3 = 2

Применим метод модифицированных множителей Лагранжа.

Составим функцию Лагранжа (x1,x2,x3, λ1, λ2) = x1x2+x2x3+λ1(x1+x2-2)+ λ2(x2+x3-2) и продифференцируем ее по переменным (x1,x2,x3, λ1, λ2)

Далее приравняем их к нулю.

x2+λ1=0

x1+x3+λ1+λ2=0

x2+λ2=0

x1+x2-2=0

x2+x3-2=0

λ1 = λ2;

x1 + x3 + 2 λ=0

x2 + λ=0

x1 +x2 – 2 = 0

x2 +x3 – 2 = 0

x1-x3=0

x1=x3

2x1+2 λ=0

x2+ λ=0

x1=x2

x1=x2=x3

x=- λ

x1=x2=x3=1

λ=-1

x1\*=x2\*=x3\*=1

f(x\*) = 2

Задачи векторной оптимизации

Постановка задачи векторной оптимизации

Эффективность функционирования многих систем оценивается, как правило, несколькими критериями. Математической формой критерия эффективности в оптимизационных математических задачах является целевая функция.

Пусть имеется *к* критериев, которые можно записать в виде целевой функций fk(x) (k=1..k), где x=(x1x2..xn). Задача многокритериальной оптимизации:

F(x)={f1(x), f2(x),..fk(x)}->max (1)

Gi(x)<=bi, i=1..I (2)

x>=0 (3)

Оптимизация по Парето

Если точки максимума xk’, определенные при решении задач по каждому критерию fk(x) не совпадают, то решение задачи (1) -(3) может быть только компромиссным. В области допустимых значений задачи находится область компромиссов в другую, невозможно одновременное улучшение всех критериев. Принадлежащие области компромиссов планы называются эффективными или оптимальными по Парето.

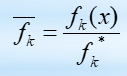
План x0 оптимален по Парето, если он допустим и не существует такого плана x1 для которого fk(x1) >= fk(x0) (k=1..K) и хотя бы для одного критерия выполняется строгое неравенство.

К задачам векторной оптимизации приходят в следующих случаях:

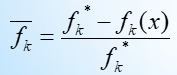
1. Качество моделируемого процесса нужно оценить с точки зрения нескольких показателей. Это могут быть прибыль, себестоимость, рентабельность и т.д.
2. Моделируемый процесс представляет собой составляющую нескольких процессов (частей), и каждая из этих частей имеет свой критерий качества.
3. Моделируемый 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.



1. 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.
2. 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:

1. Методы последовательных уступок.
2. Метод ведущего критерия.
3. Метод равных и наименьших относительных отклонений.
4. Метод минимакса.

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.

Метод последовательных уступок

Алгоритм:

1. Критерии (переменные х) нумеруются в порядке убывания важности.
2. Решается задача

f1(x)->max

gi(x)<=b, i=1..I (набор ограничений)

x>=0 (набор ограничений)

Определяется значений f1\*

1. Устанавливается уступка Δ1 по этому критерию. (Насколько можем отодвинуться от оптимального решения для того, чтобы второе значение функции было самым оптимальным)
2. Решается задача

f2(x)->max

f1(x)>=f1\*- Δ1

gi(x)<=b, i=1..I (набор ограничений)

x>=0 (набор ограничений)

Если в задаче более двух критериев, то пункты 3 и 4 повторяются до конечного значения.

Пример:

Есть два уравнения.

Уступка по первому критерию составляет 10%.

F(x)={f1=x1+3x2, f2=40x1+10x2)->max.

F2 должно быть оптимальным, и влияло до 10% на f1.

Ограничения:

2x1+x2<=90

x1+x2<=60

x2<=50

x1>=0

x2>=0

Решаем по критерию f1.

f1\*=160 (x2<=50, берем его по большему влиянию).

Доп условием будет x1+3x2 >= 160-16

f2=40x1+10x2->max

2x1+x2<=90

x1+x2<=60

x2<=50

x1+3x2>=144

x1>=0

x2>=0

Получим x\*= (18,42), f2\*(x)=1140), f1(x\*) =144.

Сетевые модели

Проект – деятельность, имеющая начало и конец во времени, направленная на достижение определенного результата. Проект представляется в виде ряда элементарных работ, которые называют операциями. Совокупность операций проекта и их зависимостей называется комплексом операций. Каждой операции комплекса соответствует два момента времени – начало и окончание операции. Эти моменты называются события. Исходным событием комплекса операций называется событие, которое не является конечным ни для одной операции комплекса. Завершающим событием называется событие, которое не является начальным ни для одной операции комплекса. Все остальные операции называются промежуточными. Моментом свершения события считается момент окончания всех операций, для которых это событие является завершающим.

Комплекса операций проекта разработки web-приложения WSP.

|  |  |  |  |
| --- | --- | --- | --- |
| Код операции | Наименование операции | Предшествующая операция | Продолжительность (дни) |
| 1. Анализ | | | |
| Z1 | Системный анализ |  | 15 |
| Z2 | Анализ требований | Z1 | 20 |
| 1. Проектирование | | | |
| Z3 | Проектировка БД | Z2,Z15,Z17 | 10 |
| Z4 | Проектирование классов | Z2,Z17 | 20 |
| Z5 | Проектирование интерфейсов пользователей | Z15, Z17 | 5 |
| 1. Кодирование | | | |
| Z6 | Кодирование интерфейсов пользователей | Z4, Z5, Z16, Z17 | 15 |
| Z7 | Кодирование процедур СУБД | Z3, Z4, Z15, Z17 | 15 |
| Z8 | Кодирование классов | Z4, Z4, Z15, Z17 | 30 |
| 1. Тестирование | | | |
| Z9 | Функциональное тестирование | Z6, Z7, Z8, Z18 | 30 |
| Z10 | Структурное тестирование | Z6, Z7, Z8, Z18 | 25 |
| 1. Внедрение | | | |
| Z11 | Разработка документации | Z6, Z7, Z8, Z9 | 10 |
| Z12 | Обучение пользователей | Z9, Z11 | 20 |
| Z13 | Испытание | Z9, Z10, Z11, Z12 | 60 |
| Z14 | Завершение работ | Z13 | 5 |
| 1. Дополнительные работы | | | |
| Z15 | Установка СУБД | Z1 | 3 |
| Z16 | Установка сервера | Z1 | 3 |
| Z17 | Установки инструментария | Z1 | 3 |
| Z18 | Подготовка полигона | Z1 | 4 |
|  |  |  |  |

Анализ таблицы позволяет выявить следующие закономерности:

1. У операции Z1 нет предшествующих операций. В соответствии с определением начальное событие Z1 является исходным событием комплекса операций.
2. У операции Z14 нет последующих операций, поэтому завершающее событие Z14 является завершающим событием комплекса операций.
3. Все операции, кроме Z5, Z12, Z13, Z14 предшествуют нескольким операциям.

Нумерация событий комплекса операций -

|  |  |  |  |
| --- | --- | --- | --- |
| Начальное событие | Код операции | Предшествующие операции | Конечное событие |
| 0 | Z1 |  | 1 |
| 2 | Z2 | Z1 | 3 |
| 4 | Z3 | Z2, Z15, X17 | 5 |
| 6 | Z4 | Z2, Z17 | 7 |
| 8 | Z5 | Z15, Z17 | 9 |
| 10 | Z6 | Z4, Z5, Z16, Z17 | 11 |
| 12 | Z7 | Z3, Z4, Z15, Z17 | 13 |
| 14 | Z8 | Z3, Z4, Z15, Z17 | 15 |
| 16 | Z9 | Z6, Z7, Z8, Z18 | 17 |
| 18 | Z10 | Z6, Z7, Z8, Z18 | 19 |
| 20 | Z11 | Z6, Z7, Z8, Z9 | 21 |
| 22 | Z12 | Z9, Z11 | 23 |
| 24 | Z13 | Z9, Z10, Z11, Z12 | 25 |
| 26 | Z14 | Z13 | 27 |
| 28 | Z15 | Z1 | 29 |
| 30 | Z16 | Z1 | 31 |
| 32 | Z17 | Z1 | 33 |
| 34 | Z18 | Z1 | 35 |

Сетевой график – взвешенный ориентированный корневой граф без контуров и изолированных вершин, построенный по определенным правилам.

Пусть Z – множество операций, I – множество событий комплекса операций P.

Построим ориентированный граф G по следующим правилам:

1. Количество вершин графа равно количестве событий комплекса операций. |V|=|I|.
2. Количество дуг графа равно количеству операций комплекса операций |E| = |Z|.
3. Должны быть заданы две биективные функции разметки, сохраняющие инцидентность событий и операций:
4. F: vi->i, i из I на множестве вершин графа vi из V
5. G: ek-> zk, zk из Z на множестве дуг графа ek из E.

4. Граф должен иметь только одну вершину не имеющую входящих дуг, она должна соответствовать исходящему событию комплекса.

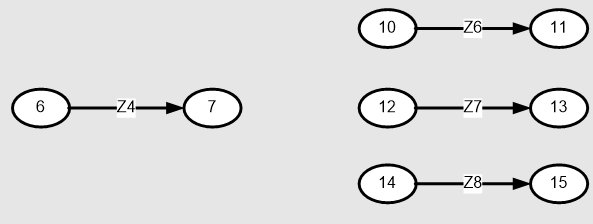
5. Граф должен иметь только 1 вершину, не имеющую исходящих дуг, она должна соответствовать завершающему событию.

6. Граф не должен содержать контуров.

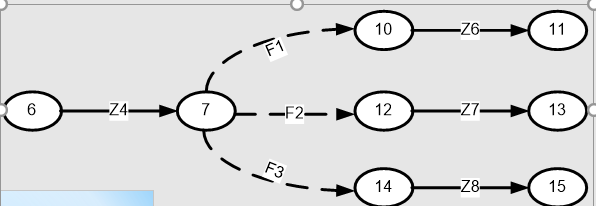
7. Любая пара вершин должна быть соединена не более, чем одной дугой.

35 вершин в графе поскольку 35 событий и 18 дуг

Попытка построить сетевой график приводит к некоторым сложностям, например, рассмотрим операцию Z4 имеющую начальное событие 6 и конечное 7. Она предшествует операциям Z6, Z7, Z8. Первоначальное построение приводит к следующей картинке:



Событие 7 должно предшествовать событиям 10, 12 и 14. Но в таблице нет операция, связывающих эти события. В таких случаях необходимо расширить комплекс операций, добавив фиктивные операции, которые позволяют отразить недостающие логические связи между событиями.



Фиктивные операции отражают технологическую или ресурсную зависимость в выполнении некоторых операций. F1 F2 F3 отражают тот факт, что кодирование интерфейсов пользователей (Z6), кодирование процедур СУБД (Z7) и кодирование классов (Z8) в данном проекте не могут быть начаты раньше, чем закончится проектирование классов. Пунктирные линии – фиктивные операции (можно не подписывать).

