**Лекция 6**

**Динамическое программирование**

3: Как уже отмечалось, в некоторых случаях рекурсивное решение может оказаться очень трудоёмким, так как в процессе его выполнения одна и та же задача может решаться несколько раз. Простой анализ примера вычисления дистанции Левенштейна позволяет выявить такие повторные вычисления. Например, функция вычислялась в шагах 7, 8, 7, 16 алгоритма.

4, 5: Если в процессе выполнения рекурсивного алгоритма одна и та же задача решается несколько раз, то говорят, что алгоритм содержит ***перекрывающиеся подзадачи***. Например, задача вычисления функции, рассмотренная выше, является перекрывающейся подзадачей задачи вычисления функции

6: Метод решения задачи оптимизации, реализующей рекурсивный алгоритм с перекрывающимися подзадачами, в котором каждая такая подзадача решается один раз, а ее результат сохраняется для последующего применения, называется ***динамическим программированием***.

7: Пример реализации алгоритма вычисления члена последовательности Фибоначчи методом динамического программирования.

Обратите внимание, что новой реализацией функции **fib** для хранения промежуточных значений используется статический массив. Применение массива позволяете избежать повторных рекурсивных вычислений, которые были в реализации, представленной в пред. лекции.

Заметим, что задачи вычисления факториала и наибольшего общего делителя не имеет смысла решать методом динамического программирования, так как при их рекурсивном решении не образуются перекрывающиеся подзадачи и, следовательно, нет необходимости запоминать промежуточные вычисления.

Следует отметить, что рассматриваемый здесь метод является частью более общей теории динамического программирования, основы которой разработаны Р. Беллманом. Эта теория исследует процесс пошагового решения задач оптимизации, в котором на каждом шаге из множества допустимых решений выбирается одно, оптимизирующее заданную целевую функцию.

**Решение задачи о рюкзаке**

Применение динамического программирования при решении оптимизационных задач обычно предполагает создание специальных таблиц для хранения промежуточных результатов.

8: Изображены таблицы, которые используются для решения задачи о рюкзаке методом динамического программирования. Векторы, определяющие размеры()и стоимости () типов предметов, а также величина, характеризующая вместимость рюкзака (), заданы в верхней части рисунка.

Количество таблиц, которое необходимо построить для решения поставленной задачи, соответствует заданному в задаче  – количеству типов предметов (на рис. 2 ). Каждая таблица, кроме помеченной символом S, пронумерована и описывает все возможные случаи размещения в рюкзаке соответствующего типа предметов.

Например, таблица Т.0 описывает все возможные способы размещения в рюкзаке предметов, имеющих объем 85 и стоимость 150 за единицу веса, а таблица T.1 – все способы размещения предметов объемом 50 и стоимостью 25.

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

Построение таблиц осуществляется последовательно в порядке их нумерации. Каждая таблица строится на основе предшествующей. Таблица, помеченная символом S, имеет чисто техническое назначение и предназначена для построения таблицы T.0. Таблица S состоит из одной строки, в первом столбце этой строки указывается вместимость рюкзака, а два другие столбца заполняются нулями.

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

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

Определить стоимость ранее размещенных предметов можно по таблице T.0. Для этого необходимо найти строку, которая соответствует остатку 215. Это вторая строка, так как она описывает размещение в пустом рюкзаке (300 единиц незанятого объема) одного предмета объемом 85 и стоимостью 12750. Таким образом, стоимость предметов в рюкзаке, которая будет записана в десятой строке таблицы T.1, будет 

В последней таблице (в нашем случае в таблицеT.6) каждая строка соответствует одной из допустимых комбинаций предметов, помещенных в рюкзак. Для выбора комбинации, имеющей наибольшую стоимость, необходимо просмотреть всю таблицу и выбрать строку, в которой третий столбец имеет максимальное содержимое (на рис. эта строка выделена, содержимое третьего столбца – 39 650). Далее следует восстановить путь построения этой строки, продвигаясь от последней таблицы к первой, используя значение третьего столбца в качестве ключа.

Для того чтобы определить строку в таблице T.5, необходимо вычислить значение ее третьего столбца. Оно равно разнице содержимого третьего столбца выделенной в таблице T.6 строки (39 650) и стоимости предметов, общий объем которых является значением второго столбца (это значение равно 0). Таким образом, разница составляет Искомая в таблице T.5 будет иметь значение 39650 в третьем столбце (на рис. эта строка выделена).

Отыскав в таблице T.5 необходимую строку, следует подобным способом дойти до таблицы T.0 (на рис. все выбранные строки выделены). Разделив содержимое выделенных строк на объем соответствующего таблице типа предмета, получимих количество. Таким образом, решением данной задачи будет вектор  каждая компонента которого – количество предметов соответствующего типа.

**Решение задачи о расстановке скобок при перемножении матриц**

9-14: На рис. представлена функция **OptimalMD**, которая реализует алгоритм динамического программирования для решения задачи о расстановке скобок.

**Решение задачи о наибольшей общей подпоследовательности**

15-17: На рис представлена программная реализация алгоритма динамического программирования для решения задачи о наибольшей общей подпоследовательности (LCS).

18: Для построения наибольшей общей подпоследовательности в функции **lcsd** используются два массива **С** и **B**.

Массив **C** предназначен для вычисления длины LCS. Элементы массива **C** – числа.

Порядок заполнения массива:

– первая строка и первый столбец заполняются нулями;

– каждый элемент последовательно заполняется по формуле для c[i, j]:

если символы для позиции i, j совпадают, то в неё записывается значение c[i−1, j −1]+1, иначе вычисляется максимум от соседей слева и сверху.

Элемент в правом нижнем углу показывает длину наибольшей общей подпоследовательности.

Массив **B** предназначен для построения массива LCS (самой подпоследовательности). Заполняется следующим образом: все ячейки кроме технических строк заполняются стрелками, направленными «вверх». Если символы для позиции i, j совпадают, то стрелка меняется на «лево-вверх». Если числовое значение от соседа слева больше, чем от соседа сверху, то стрелка «вверх» меняется на «лево».

Подпоследовательность получается обходом стрелок с правого нижнего угла путем добавления символов в подпоследовательность при наличии стрелки «влево-вверх» во время движения.

**Лекция 7**

**Математические основы сетевого планирования**

**Основные понятия теории графов**

3: Граф – это математическая модель, с помощью которой удобно представлять бинарное отношение. Хотя теория графов получила свое развитие задолго до появления теории множеств как самостоятельной дисциплины, большое число задач теории отношений формулируются и решаются в рамках именно этой теории.

Рассмотрим множество, которое будем назвать ***множеством вершин***, и набор упорядоченных пар, называемый ***множеством дуг***. Параназывается ***графом***. Запись обозначает граф с именем , состоящий из множества вершин и множества дуг .

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

4: На рис. изображен граф , где  и .

5: Две соединенные дугой вершины графа, называются ***смежными вершинами***. Например, и смежные вершины графа , т.к. есть дуга .

Считается, что дуга ***выходит из вершины*** и ***входит в вершину*** . При этом говорят, что вершины  и ***инцидентны*** дуге , а дуга  инцидентна вершинам  и . Например, вершины  и  инцидентны дуге , выходящей из  и входящей в.

Вершины  и дуги называют соответственно ***начальной*** и ***конечной*** вершинами этой дуги.

Дуги, которые выходят и входят в одну и ту же вершину, называются ***петлями***. Например, дуга является петлей.

Вершины, не имеющие смежных, называются ***изолированными вершинами***. Например, –изолированная вершина.

Очевидно, что множество дуг  можно интерпретировать как бинарное отношение, а  – множество, на котором это бинарное отношение строится.

Если множество дуг  не является симметричным отношением, то такой граф называется ***ориентированным графом***. Граф, изображенный на рис., является ориентированным графом.

6: Если множество дуг – симметричное отношение, то соответствующий граф называется ***неориентированным графом***. Симметричность отношения предполагает, что для каждой дуги  графа найдется дуга , имеющая противоположное направление.

На рис. изображен граф, представляющий симметричное отношение , которое может быть описано следующей матрицей  . Для того, чтобы разгрузить рисунок при изображении неориентированного графа, принято пару противоположных дуг изображать линией без стрелок, как это изображено на рис.

Для того, чтобы подчеркнуть, что порядок вершин в неориентированном графе не имеет значения, при обозначении пар вершин, соединенных двумя противоположными дугами, используется запись с круглыми скобками: , а сами такие пары называют ***ребрами***.

В дальнейшем нас будут интересовать только ориентированные графы, т.к. именно они используются для моделирования сетевых графиков. Поэтому в дальнейшем изложение основ теории графов будет посвящено ориентированным графам.

***Способы представления графов***

Существует три основных способа представления графов: матрица смежности, матрица инцидентности и списки смежных вершин.

***Матрица смежности***

***7: Матрица инцидентности*** имеет размерность *m*×*n*, где –количество вершин, а– количество дуг. Каждыйэлемент матрицы принимает одно из трех значений:  –вершина не инцидентна дуге ;  –дуга  выходит из вершины ; –дуга входит в вершину . Например, при заданной индексом нумерации ребер графа, изображенного на рис. ,его матрица инцидентности  будет следующей:

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

***8: Списки смежных вершин*** представляют собой набор из множеств, соответствующих  вершинам графа. Множество, соответ-ствующее вершине , либо пустое, либо содержит номера вершин, в которые входит дуга, выходящая из вершины *i*. Например, списки смежных вершин для графа на рис. можно представить в виде следующих пяти множеств: ; ; .Следует отметить, что этот способ является наиболее удобным при программировании большинства задач теории графов.

При программировании реальных задач теории графов, как правило, применяются матрица смежности и списки смежных вершин. При этом часто этой информации бывает недостаточно, т.к. она отражает только структуру графа. Если с вершинами и/или дугами графа связаны какие-то дополнительные характеристики, необходимо предусмотреть возможность их хранения.

***9: Кратчайшие и максимальные пути между вершинами графа***

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

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

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

Поиск кратчайшего пути между двумя вершинами графа является одной из часто используемых в приложениях задач. Наиболее известными способами решения этой задачи являются ***алгоритмы Дейкстры***, ***Беллмана****–* ***Форда*** и ***Флойда – Уоршолла***. Здесь будет рассмотрен только алгоритм Дейкстры (Эдсгер Вибе Дейкстра, известный голландский ученый-математик, 1930–2002).

10: Алгоритм Дейкстры решает задачу о кратчайших путях во взвешенном графе  с дугами неотрицательного веса , , из вершины  во все достижимые вершины этого графа. Алгоритм последовательно преобразовывает исходный граф  в дерево кратчайших путей . ***Дерево кратчайших путей*** – это граф , обладающий следующими свойствами:

1. ;
2. –ориентированное дерево с корнем ;
3.  –множество достижимых вершин графа из;
4. для каждой вершиныпуть из вв деревеявляется кратчайшим путем в графе .

11: Алгоритм Дейкстры предполагает, что граф  задан с помощью списков смежных вершин, а также известна весовая функция , заданная на множестве*V×V*.Функция  ставит в соответствие каждой паре вершин вес дуги, соединяющей эти вершины, или, если такой дуги нет в графе . Кроме того, задана вершина , относительно которой определяются все кратчайшие пути.

Для пояснения работы алгоритма Дейкстры будем использовать следующие обозначения.

12: –множество вершин графа , смежных с вершиной .По сути, – это списки смежных вершин графа , с помощью которых задан граф .Иначе – это перечень всех вершин, которые являются конечными вершинами дуг с началом в.

– верхняя оценка кратчайшего пути из вершины  в вершину . В начале работы алгоритма,адля всех остальных оценка.

 –вершина, предшествующая вершине  вдереве кратчайших путей. В начале работы алгоритма для всех , где –специальный символ, обозначающий пустоту. После завершения работы алгоритма множество значений позволяет построить дерево кратчайших путей.

– множество вершин . Вначале множество пустое: . В процессе работы алгоритма пополняется вершинами из , для которых уже определен кратчайший путь из вершины .После окончания работы алгоритма .

13:  – множество вершин . Вначале . В процессе работы алгоритма элементы этого множества – вершины, не добавленные во множество : . После окончания работы алгоритма .

– процедура извлечения элемента из множества  с наименьшим текущим значением .При сравнении предполагается, что , где  –любое действительное число.

 –процедура релаксации, которая определена для произвольных двух вершин  графа . Релаксация состоит в следующем: значение уменьшается дов том случае, если значение  меньше текущего значения . При этом операция сложения предполагает, что, где  –любое действительное число или. Кроме того, если  уменьшилось, то значение  становится равным .

 –кратчайший путь из вершины  в вершину .

14: Выполнение алгоритма Дейкстры состоит из двух этапов. На первом этапе инициализируются , ,  и  значениями, описанными выше. Второй этап алгоритма опишем с помощью мнемокода, как показано на рис. Для удобства строки мнемокода пронумерованы.

1**while**

2**{**

3 = 

4****

5**for длявсех**

6**{**

7****

Основной цикл алгоритма (строки 1–9) на рис. выполняется до тех пор, пока все вершины графа не будут извлечены из множества  (строка 1). Извлечение вершин из множества  осуществляется с помощью процедуры(строка 3). Извлеченная вершина помещается сначала в переменную затем во множество (строка 4). Далее выполняется внутренний цикл (строки 5–8), в котором для всех вершин, имеющих входящие дуги с начальной вершиной , выполняется процедура релаксации.

Результатом выполнения алгоритма Дейкстры являются массивы и, состоящие из  элементов. Массив позволяет построить граф кратчайших путей, а каждый элемент массива  содержит вес кратчайшего пути между вершинами  и .

15-18: На рис. приведен пример решения задачи поиска кратчайшего пути в графе с помощью алгоритма Дейкстры. Изображен исходный граф и проинициализированные массивы , ,  и . В качестве меток для вершин графа используются числа от  до .

В примере на рис. осуществляется поиск кратчайших путей из вершины  до всех остальных вершин графа. По мере решения задачи, метки вершин графа перемещаются из массива  в массив . В массивеформируется вес пути для каждой вершины, а в массиве  – список предшествующих вершин. Результат решения представляет собой дерево кратчайших путей. В этом дереве из вершины  до любой другой вершины графа существует единственный путь, который является кратчайшим.

19: При расчете временных характеристик сетевого графика, необходимо найти критический путь, определяющий минимальное время выполнения проекта. Отыскание критического, максимального, пути в графе сводится к поиску пути с самым большим весом, называемого максимальным путем в графе.

***Максимальным путем  называется путь с максимальным весом***. В общем случае, в одном графе может быть несколько максимальных путей.

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

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

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

20-25: На рис. представлен пример решения задачи поиска максимального пути в графе.

На рис. изображен заданный граф и проинициализированные массивы  и . Для построения максимального пути в графе необходимо найти максимальный элемент в(в нашем случае – это 18) и обратным порядком построить все предшествующие  вершины по массиву (в нашем случае – это вершины: , , , ).

**Лекция 8**

**ОПТИМИЗАЦИОННЫЕ АЛГОРИТМЫ НА ГРАФАХ**

1-2: Многие алгоритмы на графах могут быть решены путем систематического перебора (обхода) вершин и дуг графа. Такой обход можно выполнить многими способами. На практике широкое распространение получили два способа, получивших специальные названия: ***поиск в ширину*** и ***поиск в глубину***.

**Алгоритм поиска в ширину**

3: Для обозначения алгоритма поиска в ширину обычно используют аббревиатуру ***BFS*** (***Breadth-first search***). Алгоритм подразумевает, что задана исходная (***стартовой***) 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 (т. е. находящиеся на расстоянии 1), после чего вершины, находящиеся на расстоянии 2 от стартовой и т.д.

Очевидно, что алгоритм BFS посетит только те вершины, для которых существует последовательность дуг, связывающая с ними стартовую вершину. Обычно, по окончании работы алгоритма, осуществляется проверка на полноту обхода и, если имеются вершины, которые не посещались, то среди них выбирается любая, назначается стартовой и алгоритм выполняется снова.

4: Текущее состояние алгоритма хранится в следующих структурах памяти: **Q** – очередь вершин, **С**– массив окраски вершин, **D**– массив расстояний и **P** – массив предшествующих вершин. Все массивы имеют размерность, равную количеству вершин в графе.

Очередь **Q** (структура памяти, реализующая алгоритм «первый вошел − первый вышел») используется для промежуточного хранения номеров вершин. На каждом шаге алгоритма, в очередь помещаются номера вершин в порядке их обнаружения. На каждом шаге, кроме первого, из очереди извлекается очередной номер вершины, подлежащей отметке о посещении (фиксации). На первом шаге алгоритма в очередь помещается номер стартовой вершины. На последнем шаге очередь пуста.

5: Массив **C** используется для хранения состояния вершин. С каждым из трех возможных состояний обычно связывают цвет: белый (**W**) – вершина не посещалась, серый (**G**)– вершина посещалась, черный (**B**) – фиксирован факт посещения вершины. На первом шаге алгоритма стартовая вершина окрашивается в серый цвет, а остальные – в белый. На последнем шаге все вершины становятся черными.

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.

6: Массив **P** 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, интерпретируемое как «пустота» (**N**).На последнем шаге алгоритма для всех доступных вершин будут заполнены значения, равные номеру предшествующей вершины в порядке обхода.

Полученный в результате работы массив **P** позволяет построить такназываемое BFS-дерево.

7-10: ***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 – стартовая вершина.

11-14:

15: Алгоритм BFSсводится к следующей последовательности шагов.

1. Инициализировать массивы **С**, **D**, **P**.Стартовую вершину **s** поместить в очередь **Q**.и окрасить в серый цвет: **C[s] = G**. Для стартовой вершины установить расстояние, равное нулю: **D[s] = 0**.
2. Если очередь**Q** пуста, то работа алгоритма завершена, в противном случае перейти к следующему шагу.
3. Выбрать из очереди **Q**вершину **k** и окрасить ее в черный цвет: **С[k]=B**.
4. Построить множества **J** вершин белого цвета смежных вершине **k**. 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 (но не обязательно) в очередь вершины помещаются в порядке возрастания номеров.
6. Каждую вершину **j** из множества **J**окрасить в серый цвет: **С[j] = G**.
7. Для каждой вершины**j** из множества **J** вычислить расстояние: **D[j] = D[k] +1**.
8. Для каждой вершины**j** из множества **J** указать предшествующую вершину: **P[j] = k**.
9. Перейти к шагу 3.

**Алгоритм поиска в глубину**

16: Для обозначения алгоритма поиска в глубину используют аббревиатуру ***DFS*** (***Depth-first search***). Как и для поиска в ширину, задается стартовая вершина. Алгоритм описывается следующим образом: для каждой не пройденной вершины, начиная со стартовой, необходимо найти все смежные вершины и повторить поиск для каждой.

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

17-38:

39: В основе алгоритма DFS лежит рекурсивная процедура **Visit**, имеющая один входной параметр **k** – вершину графа.

Опишем пошагово процедуру **Visit**.

1. Принять параметр **k**– вершину графа.
2. Вершину **k**окрасить в серый цвет: **C[k] = G**.
3. Увеличить номер шага: **t = t +1**.
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 **k**. Если таких вершин нет, то перейти к шагу8.
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 **k**окрасить в черный цвет: **C[k] = B**.
9. Увеличить номер шага: **t = t +1**.
10. Отметить время фиксации вершины: **F[k] = t**.

40: Если задана стартовая вершина **s**, то алгоритмDFS теперь можно свести к следующим двум шагам.

1. Инициализировать массивы: **С** (все вершины белого цвета), **D** (все расстояния равны бесконечности), **P**(все элементы заполнены символом «пустота»).Установить номер шага: **t = 0**.
2. Выполнить процедуру **Visit** для вершины **s**.

**Топологическая сортировка вершин графа**

41: Часто с помощью графов описывается некоторая последовательность связанных действий. Например, если поставить в соответствие каждой странице сайта вершину графа, а дуге – ссылку (возможность перехода) с одной страницу на другую, то такой граф будет отображать модель (карту) сайта.

Другой пример: применение графов для описания проектов. Как правило, в этом случае вершинам графа соответствуют события проекта (завершение этапа), дугам – операции. Такой граф описывает комплекс связанных операций проекта.

Если графы включают много связанных вершин, то модели, основанные на них, становятся трудно воспринимаемыми. В некоторых случаях вершины графа можно упорядочить таким образом, что вид графа становится более структурированным.

***Топологическая сортировка*** − это процедура упорядочивания вершин ориентированного графа, не имеющего циклов (ациклического графа). В результате топологической сортировки для вершин графа определяется такой порядок, что если их расположить на рисунке в соответствии с этим порядком сверху вниз, то дуги будут направлены только от верхних вершин к нижним. Обычно после выполнения топологической сортировки вершины переименовываются (перенумеровываются) в соответствии с полученным порядком. После такого переименования граф обладает свойством: начальная вершина каждой дуги имеет номер (имя) меньший, чем номер конечной вершина этой дуги.

Наиболее известны два способа топологической сортировки графа: алгоритмы Демукрона и алгоритм применяющий поиск в глубину.

42-47:

**Лекция 9**

**МИНИМАЛЬНЫЕ ПОКРЫВАЮЩИЕ ДЕРЕВЬЯ**

3: Пусть  − связный неориентированный граф, а  − весовая функция, заданная на множестве ребер  и определяющая длину каждого ребра. Считаем, что 

При обозначении ребра используются круглые скобки (а не угловые как для обозначения дуг), подчеркивающие неважность порядка перечисления вершин в ребре.

Связный подграф  графа являющийся деревом и содержащий все его вершины, называется ***покрывающим деревом***, или ***остовным деревом***.

В общем случае один граф может иметь несколько покрывающих деревьев.

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

В общем случае граф может иметь несколько минимальных покрывающих деревьев.

Наиболее известными алгоритмами построения минимального остовного дерева являются ***алгоритмы Крускала*** (Джозеф Крускал (1928–2010) − американский математик) и ***Прима*** (Роберт Прим (род. 1921) − американский математик).

4: Построение минимального остовного дерева в этих алгоритмах осуществляется поэтапно. Начинается построение с пустого множества ребер  На каждом этапе множество  пополняется одним ребром, причем так, что множество ребер  всегда остается подмножеством ребер искомого минимального остовного дерева. Задача сводится к поиску необходимого ребра (обычно его называют ***безопасным ребром***) на каждом шаге алгоритма.

Пусть  − множество вершин графа  тогда ***разбиением*** этого множества будем называть семейство множеств  …,  обладающее следующими свойствами: 

5: Пусть  и – разбиение множества вершин связного неориентированного графа  Если провести линию, разделяющую множества  и  то эта линия пересечет ребра, концевые вершины которых лежат в разных подмножествах разбиения. Множество ребер, которые пересекла линия, разделяющая подмножества вершин  и называется ***разрезом графа*** Для обозначения разреза будем использовать запись . При этом говорят, что ***множество ребер******согласовано с разрезом*** если 

Путь  – множество всех ребер, принадлежащих разрезу  Тогда ребро  имеющее минимальную длину, называется ***легким ребром*** разреза 

6: **Теорема.** Пусть  − связный неориентированный граф и  – минимальное остовное дерево этого графа. Пусть также множество ребер  согласовано с некоторым разрезом , а – легкое ребро этого разреза. Тогда  является безопасным ребром для множества 

Воспользуемся теоремой для построения минимального остовного дерева для графа, но предварительно сформулируем еще одно утверждение.

Если  – ребро, имеющее минимальную длину в неориентированном связном графе, то оно будет ребром одного из минимальных остовных деревьев. Утверждение является следствием теоремы: достаточно положить 

Алгоритмы Крускала и Прима относятся к классу алгоритмов, называемых ***жадными алгоритмами***. Такое название эти алгоритмы получили за стратегию, заключающуюся в принятии на каждом шаге локально оптимального решения (жадного решения), в предположении, что такая стратегия приведет к конечному оптимальному решению.

7: Пример построения минимального остовного дерева для связного неориентированного графа. Сначала все ребра графа изображены пунктирной линий, над которой указана длина этого ребра. На рисунке показано пошаговое построение разрезов, начиная с  и заканчивая  По мере того, как вершины и дуги включаются в состав минимального остовного дерева, они меняют свою окраску. На каждом шаге, кроме последнего, выбирается легкое ребро разреза, которое, согласно приведенной выше теореме, является безопасным для множества окрашенных ребер.

7-16:

**Алгоритм Крускала**

Алгоритм Крускала работает по тому же принципу. Отличие только в порядке объединения вершин в пошагово формирующемся минимальном покрывающем дереве.

17: Первоначально в алгоритме Крускала неориентированный связный граф  с заданной на его ребрах весовой функцией  разбивается на максимальное количество подграфов, каждый из которых является деревом. Очевидно, что каждое такое дерево будет представлять собой одну вершину графа, а общее количество таких подграфов будет 

Далее из множестваребер графа  поочередно в порядке возрастания длины выбираются ребра. При этом возможны два случая:

1) концевые вершины лежат в разных подграфах разбиения;

2) обе концевые вершины лежат в одном подграфе разбиения.

18: В первом случае из двух подграфов, которые можно соединить выбранным ребром, образуется один общий, включающий все вершины и ребра этих подграфов, а также новое связующее ребро. Очевидно, что такое объединение не может образовать циклы, а следовательно, объединенный подграф тоже 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 (т. е. остовное дерево).

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

19: Пример построения минимального остовного дерева с помощью алгоритма Крускала.

Построение минимального остовного дерева осуществляется по шагам. Номер шага указан для каждого нового построения на рисунке.

20: На первом шаге изображено разбиение исходного графа на максимальное число (равное 9 – количеству вершин в графе) подграфов.

21: На втором шаге отыскивается ребро исходного графа, имеющее минимальную длину (ребро (7,6) с длиной 1). Вершины, соединенные выбранным ребром, окрашиваются.

22-27: На всех последующих шагах из оставшихся ребер каждый раз выбирается ребро с минимальной длиной, но такое, чтобы концевые его вершины находились в изолированных друг от друга подграфах. После выбора ребра подграфы (как описывалось выше, они являются деревьями) становятся для алгоритма единой компонентой.

28: Следует обратить внимание, что минимальное остовное дерево, построенное в предыдущем примере, отличается от полученного результата. В обоих случаях построено минимальное остовное дерево. Если подсчитать сумму длин ребер деревьев, то в обоих случаях получается 37.

**Алгоритм Прима**

29: Построение минимального остовного дерева с помощью алгоритма Прима начинается с выбора произвольной вершины исходного неориентированного связного графа. Выбранная вершина окрашивается.

На последующих шагах просматриваются все окрашенные вершины и анализируются все ребра исходного графа, у которых одна концевая вершина окрашена, а другая нет. Среди всех 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 будет составлять искомое минимальное остовное дерево.

30: Пример применения алгоритма Прима для построения минимального остовного дерева.

31: Работа алгоритма Прима представлена пошагово (номер шага указывается на рисунке). На первом шаге выбирается стартовая вершина (на рисунке – вершина 8) и окрашивается.

32: На втором шаге среди всех ребер, инцидентных стартовой вершине, отыскивается ребро, имеющее наименьшую длину (на рисунке – ребро (8, 2)). Вторая (неокрашенная) вершина ребра окрашивается, а само ребро вместе с концевыми вершинами включается в будущее минимальное остовное дерево.

33: На шагах 3–10 алгоритма выбирается по одному ребру с минимальной длиной и одной неокрашенной концевой вершиной. Неокрашенные вершины окрашиваются, выбранные ребра пополняют строящееся минимальное остовное дерево. Цикл построения дерева продолжается до тех пор, пока не будут окрашены все вершины исходного графа.

34-39: На последнем, одиннадцатом шаге из выбранных ребер строится минимальное остовное дерево. Несложно подсчитать, что суммарная длина всех ребер сформированного остовного дерева равна 37, что совпадает с результатами, полученными с помощью других алгоритмов.

**Лекция 10**

**ПОТОКИ В СЕТЯХ**

**Задача о максимальном потоке**

3: ***Сеть*** – это ориентированный граф , каждому ребру  которого поставлено в соответствие число , называемое пропускной способностью, а также выделено две вершины - исток и  - сток, .

***Поток*** – это функция ,  обладающая тремя свойствами:

1. ;

2.  (кососимметричность);

3. , 

4: ***Величина потока *** это 

Примем , , 

 - величина ***входящего потока для вершины*** 

 - величина ***исходящего потока для вершины*** 

5:

6: ***Разрез  сети***  называется разбиение множества  на две части  и  такое, что , , , .

***Пропускная способность разреза*** ****** – это сумма пропускных способностей дуг соединяющих вершины в  и .

***Минимальный разрез сети*** – это разрез с минимальной пропускной способностью.

7-8:

9: **Теорема Форда-Фалкерсона: *В любой сети максимальная величина потока из истока  в сток  равна минимальной пропускной способности разреза отделяющего  от .***

10-16: Пример.

17: Пример множества стоков и истоков.

18: **Понятие остаточной сети**

Рассмотрим ориентированный граф. Будем рассматривать его как сеть труб, по которым некоторое вещество движется от истока (где оно производится с постоянной скоростью) к стоку (где оно потребляется — с той же скоростью). Вместо потоков вещества можно рассматривать движение тока по проводам, деталей по конвейеру, информации по линиям связи или товаров от производителя к потребителю.

Как и в задаче о кратчайших путях, на каждом ребре графа мы пишем число. Но если раньше это число означало длину пути, то теперь это скорее ширина дороги, или пропускная способность трубы — максимальная скорость потока в этой трубе.

Мы считаем, что в вершинах вещество не накапливается — сколько приходит, столько и уходит (если вершина не является истоком или стоком).

Задача о максимальном потоке для данной сети состоит в следующем: найти максимально возможную скорость производства (и потребления) вещества, при которой его еще можно доставить от истока к стоку при данных пропускных способностях труб.

Ключевую роль в методе Форда-Фалкерсона играют два понятия: остаточные сети и дополняющие пути.

Поиск максимального потока данным методом проводится по шагам. Вначале поток нулевой (и величина его равна нулю). На каждом шагу мы увеличиваем значение потока. Для этого мы находим "дополняющий путь", по которому можно пропустить ещё немного вещества, и используем его для увеличения потока. Этот шаг повторяется, пока есть дополняющие пути. Как следует из теории, полученный поток будет максимальным.

Остаточные сети

Пусть дана сеть и поток в ней. Тогда остаточная сеть состоит из тех ребер (называемых также остаточными), поток по которым можно увеличить. Заметим, что остаточное ребро не обязано быть ребром исходной сети. Такие "странные" ребра появляются когда имеется поток вещества в обратном направлении — т.к. этот поток можно уменьшить.

У остаточных сетей есть интересное свойство — если в остаточной сети существует поток f, то прибавив его к исходному потоку в сети, мы получим также поток, удовлетворяющий всем требованиям, но который больше исходного.

Дополняющие пути

Назовем дополняющим путем простой путь из истока в сток в остаточной сети. Из определения остаточной сети вытекает, что по всем ребрам дополняющего пути можно переслать ещё сколько-то вещества, не превысив их пропускную способность. Величину наибольшего потока, который можно переслать по дополняющему пути назовем остаточной пропускной способностью пути. Очевидно, она равна значению минимального остаточного ребра, входящего в данный путь.

19: АЛГОРИТМ ФОРДА-ФАЛКЕРСОНА

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

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

3. Пускаем через найденный путь (он называется увеличивающим путём или увеличивающей цепью) максимально возможный поток:

* На найденном пути в остаточной сети ищем ребро с минимальной пропускной способностью cmin.
* Для каждого ребра на найденном пути поток уменьшаем на cmin, а в противоположном ему - увеличиваем на cmin.
* Модифицируем остаточную сеть. Для всех рёбер на найденном пути, а также для противоположных им рёбер, вычисляем новую пропускную способность. Если она стала ненулевой, добавляем ребро к остаточной сети, а если обнулилась, стираем его.

4. Возвращаемся на шаг 2.

**Лекция 11**

**Транспортная задача**

3: Транспортные задачи – специальный класс задач линейного программирования. Эти задачи описывают перемещение (перевозку) какого-либо товара из пункта отправления (например, места производства) в пункт назначения (склад или магазин). Назначение транспортной задачи – определение объемов перевозок из пунктов отправления в пункты назначения с минимальной суммарной стоимостью перевозок.

4: – количество поставщиков продукции;

 – количество потребителей продукции;

 – индекс для поставщиков;

 – индекс для потребителей;

,  – наличие продукции у каждого поставщика;

,  – потребность в продукции каждого потребителя;

 – стоимость доставки единицы продукции от  - поставщика к -потребителю.

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

 – задача называется ***закрытой***

– задача называется ***открытой(с нарушенным балансом)***.

***Решение открытой задачи сводится к решению закрытой***

С этой целью при *a* < *b* добавляем *фиктивного поставщика* с запасом b-a. Если же *a* > *b* , то добавляем *фиктивного потребителя* с заказом груза *a-b*. В обоих случаях соответствующие фиктивным объектам тарифы перевозок *cij* полагаем равными нулю. В результате суммарная стоимость перевозок не изменяется.

5: **Математическая модель**

 – решение задачи

Целевая функция

 – целевая функция

Ограничения

, , ,



6: **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 отправления и назначения. Т. к. транспортная задача д.б. сбалансированной, то одно из этих равенств избыточно. Т.о. транспортная задача имеет m+n-1 независимых ограничений, отсюда вытекает, что начальное базисное решение состоит из m+n-1 базисных переменных.

7:

1. **Метод наименьшей стоимости.**

Пусть  поставщиков продукции,

 потребителей продукции

Запасы ,

Потребность 

Затраты на перевозку продукции 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **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** | **50** |

8: Сначала в таблице находим ячейку с наименьшей стоимостью. Затем переменной в этой ячейке присваивается наибольшее значение, допускаемое ограничениями по спросу и предложению (если таких несколько, то выбор произволен). Далее вычеркивается соответствующий столбец или строка и корректируется спрос и предложение. Затем просматриваются не вычеркнутые ячейки, и выбирается новая ячейка с минимальной стоимостью и т.д.

1)Выбор ячейки с наименьшим значением 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потребители**  **Поставщики** | **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** | **50** |

2)

Выбор ячейки с наименьшим значением 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потребители**  **Поставщики** | **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** | **35** |

7: 3)

Выбор ячейки с наименьшим значением 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потребители**  **Поставщики** | **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** | **30** |

4)

Выбор ячейки с наименьшим значением 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потребители**  **Поставщики** | **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** | **15** |

5)

10: Выбор ячейки с наименьшим значением 

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Потребители**  **Поставщики** | **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** | **0** |
| **Потребность** | **0** | **0** | **0** | **10** | **10** |

6)

Должно быть базовых  переменных

Есть только 5, поэтому выбираем любую 

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

11:

Первое допустимое решение

, , , ,  , 

Значение функции цели



30 135 200 20 90

12: **II. Метод потенциалов**

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

*ui* + *vj* = *сij*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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** |

Потенциалы: , .

13: Определяем потенциалы для всех базисных переменных













Уравнений 6 неизвестных 7:

Присваиваем одному из них произвольное значение (обычно  ) ,

,





, , 

, ,,

14:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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** |

Для свободных клеток

|  |  |
| --- | --- |
| Небазисная переменная |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Вводимой в базис будет переменная, имеющая наибольшее положительное значение – х22.

Определив вводимую в базис переменную, следует найти исключаемую из базиса переменную. Обозначим через θ количество груза, перевозимого по маршруту (2,2). Максимально возможное значение θ определяем из следующих условий:

1. Должны выполняться ограничения на спрос и предложение.
2. Ни по какому маршруту не должны выполняться перевозки с отрицательным объемом грузов.

Сначала строим замкнутый цикл, который начинается и заканчивается в искомой ячейке. Цикл состоит из последовательности горизонтальных и вертикальных отрезков( но не диагональных), соединяющих ячейки, соответствующие текущим базисным переменным, и ячейку, соответствующую вводимой переменной. Для того, чтобы удовлетворять ограничениям по спросу и предложению, надо поочередно отнимать и прибавлять θ к значениям базисных переменных, расположенных в угловых ячейках цикла. Направление обхода цикла (по часовой стрелке или против не имеет значения).

15:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **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** |

Перемещаем 10 единиц товара по циклу.

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

16: Допустимое решение

, , , ,  , 

Значение функции цели



10 110 70 135 20 90



17:

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

Определяем потенциалы для всех базисных переменных ((2,4) уже не базисная)













Уравнений 6 неизвестных 7:

,

,







, , 

, ,,

18:

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

|  |  |
| --- | --- |
| Небазисная переменная |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Остановка, когда нет положительных чисел или нет цикла.