|  |  |  |  |
| --- | --- | --- | --- |
| ***1) Алгоритм и его свойства. Алгоритм –*** *точное, понятное предписание исполнителю совершить последовательность действий, направленных на решение поставленной задачи.* **Свойства алгоритма:** *1) Дискретность.* 2) *Конечность и понятность*  3) *Определенность* 4) *Массовость* 5) *Результативность*  ***Дискретност****ь*- это свойство алгоритма, когда алгоритм разбивается на конечное число элементарных действий. ***Конечность и понятность*** - свойство алгоритма, при котором каждое из этих элементарных действий являются законченными и понятными. ***Определенность*** - свойство, когда каждое действие должно пониматься в строго определённом смысле, чтобы не оставалась места произвольному толкованию. чтобы каждый, прочитавший указание, понимал его однозначно. ***Массовост****ь* - свойство, когда по данному алгоритму должна решаться не одна, а целый класс подобных задач. ***Результативност****ь* – свойство, при котором любой алгоритм в процессе выполнения должен приводить к определённому результату. Отрицательный результат также является результатом | **2) Сложность алгоритма.**  Определения:  1) Временнáя сложность алгоритма  2) Емкостная сложность алгоритма  3) Асимптотическая временная сложность  4) асимптотическая емкостная сложность    Мера сложности при данной размерности:  1) сложность в худшем случае,  2) средняя или усредненная сложность.  Алгоритм с наименьшей сложностью худшем случае не обязательно имеет лучшую скорость среднем. | **3) Математические основы анализа алгоритмов.**  Транзитивность:  f(n) = Θ(g(n)) и g(n) = Θ(h(n)) влечет f(n) = Θ(h(n)),  f(n) = *O*(g(n)) и g(n) = *O*(h(n)) влечет f(n) = *O*(h(n)),  f(n) = Ω(g(n)) и g(n) = Ω(h(n)) влечет f(n) = Ω(h(n)),  f(n) = o(g(n)) и g(n) = o(h(n)) влечет f(n) = o(h(n)),  f(n) = ω(g(n)) и g(n) = ω(h(n)) влечет f(n) = ω(h(n)).  Рефлексивность:  f(n) =Θ(f(n)), f(n) = *O*(f(n)), f(n) = Ω (f(n)).  Симметричность:  f(n) = Θ(g(n)) если и только если g(n) = Θ(f(n)).  Обращение:  f(n) = *О*(g(n)), если и только если g(n) = Ω(f(n)),  Т.е. для любых двух функций свойства f(n) = O(g(n)) и g(n) = Ω(f(n)) равносильны. | **4) Решение рекуррентных уравнений.**  **Способы решения:**  1) Метод итераций,  2) Подстановочный метод,  3) Метод рекурсивных деревьев  Метод Итераций заключается в том, что данное рекуррентное уравнение расписывается через множество других и затем происходит суммирование полученного выражения.  Подстановочный Метод заключается в том, что методом подбора находится такая функция *g(n),* при подстановке которой в рекуррентное уравнение вместо T(*n)* получается верное неравенство в котором ***g(n)=Левая часть* ≥ правой части*.***  Функция должна быть наименьшего возможного порядка.  Метод рекурсивных деревьев  1) На первой итерации формируется дерево следующего вида:  А) в корень дерева заносится свободный член исходного рекуррентного уравнения,  Б) сыновьями этого корня являются рекуррентные функции правой части исходного соотношения.  2) На последующих итерациях для каждого из сыновей строится аналогичная древовидная структура.  3) Процесс построения древовидной структуры заканчивается, когда все значения висячих вершин равны Т(1), |
| **5) Основная теорема о решении рекуррентного уравнения.**  Теорема. Пусть *a, b, c, k - некоторые константы, причем a≥1, b>1.*  *Тогда решение рекуррентного уравнения*  *T(n) = aT(n/b) + cnk;* T(1) = *c*  имеет вид: | **6) Разработка эффективных алгоритмов.**  **1) Метод «разделяй и властвуй»**  **2) Динамическое программирование**  Метод разделяй и властвуй заключается в:  1) Задача разбивается на независимые подзадачи (части), которые не пересекаются.  2) Каждая подзадача решается отдельно.  3) Из отдельных решений подзадач строится решение исходной задачи.  Динамическое программирование:  1) Задача погружается в семейство задач той же природы. Другими словами разбивается на зависимые (могут пересекаться) задачи.  2) Каждая подзадача решается отдельно один раз. Оптимальные значения решений всех подзадач запоминаются (*обычно в таблице*).  3) Для исходной задачи строится возвратное соотношение, связывающее между собой оптимальные значения зависимых подзадач. | **7) Понятие “сортировка”. Сортировка с помощью включения.**  Сортировать - распределять, разбирать по сортам, качеству, размерам, по сходным признакам.  Два класса сортировок:  *1) внутренняя* сортировка,  *2) внешняя* сортировка.  *Частичным порядком* на множестве *S* называется такое бинарное отношение *R,* что для любых *а, b* и *с* из S  1) *aRa (R* рефлексивно),  2) *aRb* и *bRc =>* *aRc (R* транзитивно),  3) *aRb* и *bRa =>* *a=b (R* антисимметрично).  *Все алгоритмы сортировки можно разбить на три группы:*  1) сортировка с помощью включения,  2) сортировка выбором,  3) сортировка с помощью обменов.  Сортировка с помощью включения,  1) Пусть элементы *а*1*, а*2*, …* , *аi*-1*, ; 1 < i ≤ n* уже упорядочены на предыдущих этапах данным алгоритмом.  2) На очередном этапе необходимо взять элемент *аi,* и включить в нужное место уже упорядоченной последовательности *а*1*, а*2*, …* , *аi*-1. | **8) Сортировка выбором.**  Идея сортировки заключается в следующем:  1) Выбрать элемент с наименьшим ключом и поменять его с первым элементом. Теперь первый элемент стоит на своем месте.  2) Повторить действия с оставшимися *n* *- 1* элементами.  3) Процесс заканчивается, когда *n - 1* элементов будут помещены на свои места.  А) Сортировка выбором ассиметрична прямому включению. Мы просматриваем элементы оставшейся последовательности получаем один элемент упорядоченной последовательности.  Оценка трудоемкости алгоритма  **В худшем случае:**  T(n)=Cn +T(n-1), T(1)=0;  T(n)=Θ(n2) |
| **9) Сортировки с помощью обменов.**  1) Сортировки помощью обменов основываются на сравнении двух элементов.  2) Если порядок элементов не соответствует упорядоченности, то происходит их обмен.  3) Процесс повторяется до тех пор, пока элементы не будут упорядочены.  **А) Пузырьковая сортировка**  1) Просматриваем исходную последовательность справа налево и на каждом шаге меньший из двух соседних элементов перемещается в левую позицию.  2) Самый маленький элемент будет находиться в крайней левой позиции.  3) После чего повторяем описанный выше процесс, начиная со 2-ой позиции и т.д.  **Б) Шейкерная сортировка**  1) Если при некотором из проходов нет перестановок, то алгоритм можно завершить.  2) Чередование направлений для просмотра | **10) Сортировка слиянием.**  *Сортировка слиянием заключается в следующем:*  1) Делим последовательность элементов на две части;  2) Сортируем отдельно каждую из частей;  3) Производим слияние отсортированных частей последовательности:  a) при слиянии сравниваем наименьшие элементы и меньший из них отправляем в список вывода;  b) повторяем описанные действия до тех пор, пока не исчерпается одна из частей;  c) все оставшиеся элементы другой части пересылаем в список вывода. | **11) Быстрая сортировка.**  Суть алгоритма состоит в следующем:  1) Выбрать некоторый элемент *х* для сравнения  2) Используя обмены, выполнить процедуру разделения, суть которой заключается следующем: разбить массив на две части: левую с ключами ≤ *х* и правую с ключами ≥ *х.*  Данные действия могу быть выполнены, например, следующим алгоритмом:  a) просматриваем массив слева, пока не встретим элемент *а*[i] *> х*  b) просматриваем массив справа, пока не встретим элемент а[j] < ж  c) меняем местами эти два элемента  d) продолжаем просмотр до тех пор, пока не будут просмотрены все элементы массива (i *> j*).  3) Повторяем процедуру разделения к получившимся двум частям, затем частям частей так далее, пока каждая из частей не будет состоять из одного единственного элемента. | **12) Порядковые статистики. Алгоритмы нахождения k- ого наименьшего элемента.**  *Алгоритм для нахождения k- ого наименьшего элемента из n элементов*  *1) Разбиваем исходную последовательность А на n/5 подпоследовательностей по пять элементов в каждой. В каждой такой подпоследовательности находим медиану. Это потребует C1n операций.*  *2) Из найденных на первом шаге медиан строим последовательность M и рекурсивно находим ее медиану x.*  *3) Для полученного элемента x выполним процесс разделения, который потребует C2n операций. В результате вся рассматриваемая последовательность А будет разбита на части: А1, где элементы не больше x; А2 , где элемент не меньше x.*  *4) Решаем задачу нахождения k-oro наименьшего элемента оставшихся 3n/4 элементов, что потребует времени T(3n/4).* |
| **13) Сортировка вычёрпыванием.**  Алгоритм сортировки вычерпыванием (bucket sort) работает за линейное (среднее) время. Эта сортировка годится не для любых исходных данных: говоря о линейном среднем времени, мы предполагаем, что на вход подаётся последовательность независимых случайных чисел, равномерно распределённых на промежутке [0; 1).  Идея алгоритма:  1) промежуток [0; 1) делится на *п* равных частей, после чего для чисел из каждой части выделяется свой *ящик-черпак* (bucket), и *п* подлежащих сортировке чисел раскладываются по этим ящикам.  2) Поскольку числа равномерно распределены на отрезке [0;1), следует ожидать, что в каждом ящике их будет немного. | **14) Лексикографическая сортировка.**  1) Пусть *S* некоторое множество на котором задан ‹ – линейный порядок.  2) Лексикографическим порядком на множестве *S* называется такое продолжение отношения ‹ на кортежи (списки, слова) элементов из *S* при котором  (*s*1, *s*2, …, *sp*)‹(*t*1, *t*2, …, *tq*)  означает выполнение одного из условий:  a) существует целое *j*, что *sj* ‹ *tj* и для всех *i < j* справедливо *si = ti*.  *b) p ≤ q* и *si = ti* при 1 ≤ *i ≤ p.*  3) Очевидно, что любое целое число можно считать *k-*членным кортежем цифр от 0 до *n –* 1, где *n* - основание системы счисления, в которой рассматриваются цифры. | **15) Фундаментальные структуры данных. Массив. Линейный список. Базовые операции.**  Структура данных это:  1) – это класс однородных математических объектов, ориентированный на эффективное представление данных в некотором классе задач.  2) – это систематизированный способ организации данных и доступа к ним.  Список – упорядоченная последовательность данных, характеризующих однородные объекты, отличающиеся значениями своих признаков.  Линейный список – конечная последовательность элементов (множество), структурные свойства которой ограничиваются лишь линейным (одномерным) относительным порядком элементов. | **16) Фундаментальные структуры данных. Стек. Очередь. Дек. Базовые операции.**  Стек - линейный однонаправленный список, в котором все включения и исключения элементов делаются в одном конце списка. Реализуется принцип “последний вошел – первый вышел.  Очередь – линейный список, в котором все включения элементов производятся в одном конце списка, а все исключения делаются в другом его конце. Реализуется принцип “первый вошел первый вышел” (first-in, first-out – FIFO)*.*  Дек - линейный список, в котором все включения и исключения элементов делаются на обоих концах списка  *Deque* (double-ended queue) – очередь с двусторонним доступом. |
| **17) Графы.**  *1) Граф G =* (V, E)состоит из непустого множества узлов (вершин) V имножества пар узлов E. Будем предполагать, что |V| = n и |E*|=*m*.*  2) Если множество E представлено в виде *упорядоченных пар* узлов (*v,w*), то граф называется *ориентированны),* а упорядоченная пара узлов (*v, w*)называется д*угой.*  *Путем* называется последовательность вершин вида *v*1, *v*2, … , *v*k-1, *v*k, где (*vi*, *vi*+1) ∈ E, *i* = 1, … , k-1.  Путь называется *простым,* если все узлы различны.  *Цикл –* это простой путь, который начинается и заканчивается в одном и том же узле.  Граф называется *связным,* если для любой пары вершин *существует простой путь*, их соединяющий; в противном случае, граф называется *несвязным.* | **18) Деревья. Бинарные деревья.**  **Деревом** называется связный граф без циклов.  **Коревое дерево** – это ориентированный граф, который удовлетворяет следующим условиям:  1) имеется в точности один узел, в который не входит ни одна дуга (корень);  2) в каждый узел, входит ровно одна дуга;  3) из корня имеется путь каждому узлу.  ***Глубиной узла*** *v* вкорневом дереве называется длина пути из корня в этот узел  ***Высотой узла*** *v* в корневом дереве называется длина самого длинного пути из *v* до одного из его потомков  ***Уровнем узла*** *v* называется разность высоты корневого дерева и глубины узла *v.*  ***Бинарное дерево*** *–* это упорядоченное корневое дерево у каждой вершины которого имеется не более двух сыновей. | **19) 3 способа обхода узлов бинарных деревьев.**  *1) Прямой обход* (сверху-вниз), заключается в том, что корень посещается раньше, чем поддеревья. Рекурсивно: корень, левое поддерево, правое поддерево.  *2) Обратный обход* (снизу-вверх), заключается в том, что корень посещается после поддеревьев. Рекурсивно: левое поддерево, правое поддерево, корень.  *3) Симметричный обход* (слева-направо). Рекурсивно: левое поддерево, корень, правое поддерево. | **20) Словарь. Поиск по дереву с включением.**  *Задача построения частотного словаря:* задана последовательность слов и нужно установить число появлений каждого слова.  a) Начиная с пустого дерева, каждое слово ищется в дереве. Если оно найдено, увеличивается его счетчик появлений, если нет – в дерево вставляется новое слово (с начальным значением счетчика, равным 1). |
| **21) Хэш-таблицы.**  Хэш–Таблица - это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции:  1) операцию добавления новой пары 2) операцию поиска и операцию 3) удаления пары по ключу  Важное свойство хеш-таблиц состоит в том, что, при некоторых разумных допущениях, все три операции (поиск, вставка, удаление элементов) в среднем выполняются за время O(1). Но при этом не гарантируется, что время выполнения отдельной операции мало́. | **22) Полное бинарное дерево. Базовые операции.**  Полным бинарным деревом будем называть такое упорядоченное корневое дерево, в котором  1) каждая вершина имеет не более двух сыновей;  2) заполнение дерева осуществляется от корня у листьям по уровням;  3) заполнение уровней производится слева направо.  **23) Бинарное поисковое дерево.**  Бинарное дерево называется деревом поиска, (бинарным поисковым деревом) если оно организовано так, что для каждого узла v справедливо утверждение, что все ключи в левом поддереве узла v меньше ключа узла v , а все ключи в правом поддереве больше. | **24) Б–деревья.**  *Б-деревом* называется сильно ветвящееся дерево, которое удовлетворяет следующим свойствам:  1) каждая страница содержит не более *2n* элементов ключей;  2) каждая страница, кроме корневой, содержит не менее *n* элементов;  3) каждая страница либо представляет собой лист (не имеет потомков), либо имеет *m* + 1 потомков, *m* число ключей на этой странице.  4) все страницы-листы находятся на одном уровне.  Величина *n* называется *порядком* Б-дерева.  В Б-дереве узел может иметь много сыновей, на практике до тысячи . Б-деревья позволяют реализовать многие операции с множествами размера *n* за время O(log *n*). | **25) Бинарные поисковые деревья. Базовые операции.**  1) поиск по ключу (поиск минимального, максимального);  2) добавление элемента с заданным ключом;  3) удаление элемента с заданным ключом.  4) Если бинарное поисковое дерево “вытянуто” в список, то высота дерева *h = n*, *n* – количество вершин дерева. Если бинарное поисковое дерево является сбалансированным, то *h=*log2*n .*  Трудоемкость операций – *O*(*h*) :  *O*(*n*)- в худшем,  *O*(log2*n*) – в худшем для сбалансированных деревьев.  Минимальный (максимальны элемент бинарного поискового дерева соответствует ключевому значению самой левой (правой) вершины дерева. |
| **26) АВЛ деревья. Базовые операции. Добавление элемента.**  АВЛ-дерево это бинарное поисковое дерево, у которого для каждой вершины *v* высота поддерева, корнем которого является левый сын вершины *v*, отличается не более чем на единицу от высоты поддерева, корнем которого является правый сын вершины *v.*  Пусть *k*2– вершина максимальной глубины для которой произошло нарушение инварианта в результате выполнения операции добавления нового элемента и для которой:  1) высота левого поддерева вершины *k*2больше высоты ее правого поддерева на 2,  2) у левого сына *k*1 вершины *k*2 высота его правого поддерева больше высоты его левого поддерева. | **27) АВЛ деревья. Базовые операции. Удаление элемента.**  АВЛ-дерево это бинарное поисковое дерево, у которого для каждой вершины *v* высота поддерева, корнем которого является левый сын вершины *v*, отличается не более чем на единицу от высоты поддерева, корнем которого является правый сын вершины *v.*  Пусть *k*2 – вершин для которой произошла разбалансировка.  1) высота ее левого поддерева больше высоты ее левого поддерева D на 2  2) у левого сына *k*1 вершины *k*2 высота правого поддерева больше высоты его левого поддерева. | **28) Красно-черные деревья. Базовые операции.**  Красно-черные деревья - это одно из самобалансирующихся двоичных деревьев поиска, гарантирующих логарифмический рост высоты дерева от числа узлов и быстро выполняющее основные операции дерева поиска: добавление, удаление и поиск узла. Сбалансированность достигается за счёт введения дополнительного атрибута узла дерева — «цвета». Этот атрибут может принимать одно из двух возможных значений — «чёрный» или «красный».  1) Удалить элемент. Трудоемкость О(lg *n*) (аналогично добавлению).  2) Добавить элемент. Трудоемкость О(lg *n*). | **29) Бинарные кучи.**  Бинарная куча - это полное бинарное дерево, для которого выполняется основное свойство структуры данных куча  Применение бинарных кyч:  *1) Для сортировки* элементов. Для этого достаточно вначале добавить их все в кучу, а затем, поочередно удалить. Трудоемкость O(n)+O(*n* log *n*)=O(*n* log *n*).  2) C помощью кучи быстро находить *k элемент* последовательности. Для этого нужно поместить все элементы в кучу, а затем достать из кучи *k* элементов. Если *k –* константа, то трудоемкость данного алгоритма равна O(*n*) *+* O(*k*log2 *n*) *=* O(*n*). Алгоритм так же остается линейным, если *k =* O (n / log2*п* )/  3) Кучи наиболее удобны *для поиска кратчайших путей в графах* с положительными длинам ребер. |
| **30) d-кучи.**  **d-куча -** это полное *d* -дерево для которого выполняется основное свойство кучи  Полным d-деревом называется такое дерево, в котором каждая вершина имеет не более *d* сыновей , а заполнение вершин осуществляется в порядке от верхних уровней к нижним, причем, на одном уровне заполнение вершин дерева производится слева направо.  Свойства d-дерева:  1) Не более *d k* вершин имеют глубину *k.*  2) Не более (*d k*+1–1)/(*d*–1)вершин имеют глубину между 0 и *k*.  3) Глубина *d*-дерева, содержащего *n* вершин, не более log*d* *n* | **31) Биномиальные кучи.**  **Биномиальная куча** – это биномиальный лес для которого выполняется основное свойство кучи (*инвариант* 1).Дополнительным свойством (*инвариантом* 2)биномиальной кучи является требование, чтобы в куче не было двух биномиальных деревьев одинаковой высоты.  Биномиальный лес – это семейство биномиальных деревьев.  Основные операции с биномиальными кучами:  1) Поиск минимального элемента.  2) Слияние двух биномиальных куч  3) Добавление нового элемента в кучу  4) Создание биномиальной кучи  5) Удаление минимального элемента | **32) Кучи Фибоначчи.**  *Куча Фибоначчи - это семейство корневых деревьев, для которых выполняются следующие свойства :*  *1) Каждый узел в куче Фибоначчи удовлетворяет основному свойству кучи: приоритет отца не ниже приоритета каждого из его сыновей. . 2) Каждый некорневой узел может потерять не более одного сына при выполнении процедуры cut.*  *3) В семействе корневых деревьев нет двух деревьев с корнями одинакового ранга.*  *Название кучи Фибоначчи обусловлено тем, что для доказательства оценок трудоёмкости операций используются числа Фибоначчи.*  *Основные операции с кучей Фибоначчи могут быть реализованы в виде последовательности двух процедур: link(i,j), cut (i).* | **33) Основные алгоритмы на графах. Поиск в глубину.**  **Поиск в глубину** — один из методов обхода графа. Алгоритм поиска описывается следующим образом: для каждой непройденной вершины необходимо найти все непройденные смежные вершины и повторить поиск для них **Алгоритм поиска в глубину Выполним следующие действия: 1) Из множества всех *белых* вершин выберем любую вершину, обозначим её** u1 **2) Выполняем для неё процедуру DFS(**u1**). 3) Повторяем шаги 1-3 до тех пор, пока множество *белых* вершин не пусто.** |
| **34) Основные алгоритмы на графах. Поиск в ширину.**  **Поиск в ширину** — метод обхода и разметки вершин графа. Поиск в ширину выполняется в следующем порядке: началу обхода s приписывается метка 0, смежным с ней вершинам — метка 1. Затем поочередно рассматривается окружение всех вершин с метками 1, и каждой из входящих в эти окружения вершин приписываем метку 2 и т. д.Если исходный граф связный, то поиск в ширину пометит все его вершины. Дуги вида (i, i+1) порождают остовный бесконтурный граф, содержащий в качестве своей части остовное дерево, называемое поисковым деревом.Легко увидеть, что с помощью поиска в ширину можно также занумеровать вершины, нумеруя вначале вершины с меткой 1, затем с меткой 2 и т. д. | **35) Кратчайший путь в графе.**  *Алгоритм нахождения кратчайшего пути в графе:*  1) Добавляем в кучу стартовую вершину *s* с приоритетом *p*(*s*)← 0.  2) Пока не просмотрена конечная вершина *t* или куча не станет пустой, выполняем шаги:  а) из кучи удаляется вершина *v –* с минимальным приоритетом *p*(*v*);  б) если эта вершина ранее удалялась из кучи, то она игнорируется;  в) вершина *v* считается просмотренной;  г) для вершины *v* находятся ее не просмотренные соседи *w* (они уже могут быть в куче) и добавляются кучу приоритетом  *p*(*w*)← *p*(*v*)+ *c*(*v*, *w*). | **36) Максимальный поток в графе.**  Алгоритм построения максимального потока:  1) В качестве начального потока взять нулевой поток.  2) Используя расширенны поиск в ширину, расставить метки вершинам, начиная с вершины *s*,пока не будет помечена конечная вершина *t*. Если вершину *t* пометить не удается, то не существует увеличивающего пути, текущий поток является максимальным и алгоритм заканчивает работу.  3) Восстановить увеличивающий путь из *s* в *t*,используя дерево поиска в ширину.  4) Увеличить поток из *s* в *t* на величину *met*[*t*]вдоль увеличивающего пути.  5) Убрать все метки приписанные вершинам, и перейти к шагу 2 алгоритма. | **37) Множества.**  *Основными операциями над множествами являются:*  1) Найти(*i*) – определяет имя множества, которому принадлежит элемент *i.*  2) Объединить (*X*, *Y, Z*)– процедура объединения непересекающихся множеств с именами *X* и *Y* в множество с именем *Z.*  Алгоритм поиска множества: Найти(*i*)  1) Фиксируем элемент *i* по рекуррентной формуле *i*, *P*[*i*], *P*[*P*[*i*]], … осуществляем движение до корня.  2) Результату присваиваем значение корня (имя множества, которому принадлежит элемент *i*).  3) Очевидно, что использование указанной стратегии поиска корня дерева требует O(*h*) операций в наихудшем случае, где *h* – высота корневого дерева. |
| **38) Минимальное остовное дерево, алгоритм Краскала.**  Остовным деревом *T* графа *G* называется неориентированное дерево, которое содержит все вершины графа. **алгоритм Краскала:**  1) Все ребра e∈ *E* графа *G* помещаем в кучу, где *приоритетом является стоимость ребра.*  2) Каждой вершине *i*∈*V* графа *G* соответствует одноэлементное множество. Пусть *k* – количество множеств на некотором шаге алгоритма  3) Полагаем минимальное остовное дерево *T* графа *G* пустым, T = {∅}.  4) Пока количество множеств *k* больше единицы, повторять следующую последовательность шагов:  a) Пусть *(i,j)* ∈ *E* некоторое ребро, полученное в результате выполнения операции удаления минимального элемента из кучи.  b) Если имя1 ≠ имя2 (т.е. вершины *i* и *j* принадлежат разным множествам, то выполнить следующие действия:  1) Объединить (имя1, имя2, имя1);  *2) k* ← *k* – 1; *T* ← *T* ∪ (*i*, *j*). | **39) Минимальное остовное дерево, алгоритм Прима**.  Остовным деревом *T* графа *G* называется неориентированное дерево, которое содержит все вершины графа.  Задача о минимальном остовном дереве заключается в построении остовного дерева *T* графа *G*, суммарная стоимость ребер которого минимальна.  **Сам алгоритм имеет очень простой вид. Искомый минимальный остов строится постепенно, добавлением в него рёбер по одному. Изначально полагается состоящим из единственной вершины. Затем выбирается ребро минимального веса, исходящее из этой вершины, и добавляется в минимальный остов. После этого остов содержит уже две вершины, и теперь ищется и добавляется ребро минимального веса, имеющее один конец в одной из двух выбранных вершин, а другой — наоборот, во всех остальных, кроме этих двух. Всякий раз ищется минимальное по весу ребро, один конец которого — уже взятая в остов вершина, а другой конец — ещё не взятая, и это ребро добавляется в остов. Этот процесс повторяется до тех пор, пока остов не станет содержать все вершины. В итоге будет построен остов, являющийся минимальным. Если граф был изначально не связен, то остов найден не будет.** | **40) Топологическая сортировка.**  *Топологическая сортировка -* это такая нумерация узлов ациклического графа, при которой, номер узла *vj* больше номера узла *vi*, если имеется путь из узла *vi* в узел *vj.*  *Зам. 1. Топологическое упорядочение вершин орграфа невозможно, если в графе присутствуют циклы.*  *Зам. 2. Топологическое упорядочение узлов ориентированного ациклического графа не всегда однозначно*  Алгоритм топологической сортировки  1) Поместить в очередь все узлы без входящих в них дуг.  2) Пока очередь не пуста выполнить следующие шаги:  шаг 1: Извлечь узел (*v*) из очереди.  шаг 2: Для каждого узла *w*, такого, что (*v,w*) *∈ A(v) (списку смежности)* выполнить:  1) удалить дугу *(v,w);*  2) Если у узла *w* не осталось входящих дуг, то поместить его в очередь. |