**Теория конечных графов**

**1. Построение минимального покрывающего дерева по алгоритму Краскала**

**Алгоритм Краскала** – эффективный алгоритм для построения минимального остовного дерева взвешенного неориентированного графа. **Минимальное остовное дерево** – это подграф, являющийся деревом и содержащий все вершины исходного графа, при этом сумма весов ребер минимальна.

**Алгоритм** основан на жадной стратегии и включает следующие шаги: 1. Сортировка всех ребер графа по возрастанию весов 2. Создание пустого множества ребер T (будущее дерево) 3. Последовательное рассмотрение ребер в порядке возрастания веса: - Если добавление ребра не создает цикл в T, добавить его в T - Иначе пропустить ребро 4. Алгоритм завершается, когда в T содержится n-1 ребро (где n – число вершин)

Для проверки образования цикла обычно используется структура данных “система непересекающихся множеств” (Union-Find), которая поддерживает операции: - MakeSet(x) – создание одноэлементного множества - Find(x) – определение, к какому множеству принадлежит элемент - Union(x,y) – объединение множеств, содержащих x и y

Временная сложность алгоритма: O(E log E), где E – число ребер.

**2. Построение максимального покрывающего дерева по алгоритму Краскала**

**Максимальное остовное дерево** – это остовное дерево с максимальной суммой весов ребер. Алгоритм Краскала можно модифицировать для построения максимального дерева, изменив только порядок сортировки ребер.

**Алгоритм построения** максимального остовного дерева: 1. Сортировка всех ребер графа по убыванию весов (отличие от минимального дерева) 2. Создание пустого множества ребер T (будущее дерево) 3. Последовательное рассмотрение ребер в порядке убывания веса: - Если добавление ребра не создает цикл в T, добавить его в T - Иначе пропустить ребро 4. Алгоритм завершается, когда в T содержится n-1 ребро

Остальные аспекты алгоритма (проверка циклов с помощью Union-Find, временная сложность) остаются такими же, как для минимального остовного дерева.

**3. Поиск маршрута и наименьшей длины по алгоритму Дейкстры**

**Алгоритм Дейкстры** находит кратчайшие пути от одной вершины до всех остальных во взвешенном графе с неотрицательными весами.

**Алгоритм** включает следующие шаги: 1. Инициализация: - Расстояние до начальной вершины = 0 - Расстояние до всех остальных вершин = ∞ - Создание множества непосещенных вершин, содержащего все вершины графа 2. Пока множество непосещенных вершин не пусто: - Выбор вершины u с минимальным расстоянием из непосещенных - Пометка u как посещенной (удаление из множества непосещенных) - Для каждого соседа v вершины u: - Если расстояние до v через u меньше текущего расстояния до v, - обновить расстояние до v: d[v] = d[u] + w(u,v) - запомнить, что v достигается через u

Для восстановления кратчайшего пути дополнительно хранится массив предшественников prev[v], где для каждой вершины указывается предыдущая вершина на кратчайшем пути.

Временная сложность алгоритма: - O(V²) – при использовании массива для хранения расстояний - O(E log V) – при использовании приоритетной очереди

Алгоритм неприменим для графов с отрицательными весами ребер.

**4. Особенности i-й строки и i-столбца для Алгоритма Уоршалла-Флойда**

**Алгоритм Флойда-Уоршалла** находит кратчайшие пути между всеми парами вершин взвешенного графа.

Особенности i-й строки и i-столбца в матрице расстояний: - i-я строка содержит расстояния от вершины i до всех остальных вершин - i-й столбец содержит расстояния от всех вершин до вершины i

В процессе работы алгоритма можно доказать, что: 1. После k-й итерации (для промежуточной вершины k): - Элемент d[i][j] представляет длину кратчайшего пути от i до j, использующего только вершины {1, 2, …, k} в качестве промежуточных - Значения в i-й строке и i-м столбце не изменяются на i-й итерации (когда k = i)

При k = i, для любых вершин u и v:

Путь u → i → v не может быть короче, чем уже найденный путь u → v через вершины {1, 2, …, i-1}, потому что i-я вершина уже оптимально соединена со всеми остальными через предыдущие итерации

Эти особенности можно использовать для оптимизации алгоритма, пропуская некоторые вычисления на i-й итерации.

**5. Особенности i-й строки и i-столбца для Алгоритма поиска транзитивного замыкания**

**Транзитивное замыкание графа G** – это граф G\*, такой что между вершинами u и v существует ребро тогда и только тогда, когда в G существует путь из u в v.

Алгоритм поиска транзитивного замыкания – модификация алгоритма Флойда-Уоршалла, где вместо расстояний используются булевы значения достижимости.

Особенности i-й строки и i-столбца при поиске транзитивного замыкания: - i-я строка показывает, какие вершины достижимы из вершины i - i-й столбец показывает, из каких вершин можно достичь вершину i

В процессе работы алгоритма на i-й итерации: - Если вершина j достижима из i (R[i][j] = true) и вершина k достижима из j (R[j][k] = true), - то k достижима из i (R[i][k] = true) - Аналогично, если i достижима из j (R[j][i] = true) и j достижима из k (R[k][j] = true), - то i достижима из k (R[k][i] = true)

Это свойство позволяет распространять транзитивность отношения достижимости через промежуточную вершину i.

**6. Поиск максимального потока в графе**

**Поток в сети** – это функция, которая каждому ребру (u,v) сопоставляет значение f(u,v), удовлетворяющее ограничениям: - 0 ≤ f(u,v) ≤ c(u,v) (пропускная способность) - ∑f(u,v) = ∑f(v,w) для всех вершин v, кроме источника s и стока t (сохранение потока)

**Максимальный поток** – поток с наибольшим значением |f| = ∑f(s,v).

**Алгоритм Форда-Фалкерсона** для поиска максимального потока: 1. Инициализация: f(u,v) = 0 для всех ребер (u,v) 2. Пока существует увеличивающий путь p от источника s к стоку t в остаточной сети: - Найти минимальную остаточную пропускную способность cf(p) на пути p - Для каждого ребра (u,v) на пути p: - Увеличить поток f(u,v) на cf(p) - Уменьшить поток f(v,u) на cf(p) (для обратных ребер) 3. Вернуть поток f

Остаточная сеть Gf содержит ребра с ненулевой остаточной пропускной способностью: - Для прямых ребер: cf(u,v) = c(u,v) - f(u,v) - Для обратных ребер: cf(v,u) = f(u,v)

Временная сложность: O(|E|·|f\_max|), где |f\_max| – значение максимального потока.

**Алгоритм Эдмондса-Карпа** (вариант алгоритма Форда-Фалкерсона) использует поиск в ширину для нахождения увеличивающего пути, что дает временную сложность O(V·E²).

**7. Поиск гамильтонова цикла в орграфе**

**Гамильтонов цикл** – это цикл в графе, проходящий через каждую вершину ровно один раз и возвращающийся в исходную вершину.

Поиск гамильтонова цикла – NP-полная задача, для которой не существует эффективного полиномиального алгоритма. Для ориентированных графов задача решается с помощью перебора с возвратом (backtracking).

Алгоритм с упрощением для поиска гамильтонова цикла в орграфе: 1. Выбор начальной вершины path[0] 2. Рекурсивная функция для построения пути: - Если путь содержит все вершины и существует ребро от последней вершины к начальной, цикл найден - Иначе перебор всех вершин v, не входящих в текущий путь: - Если существует ребро из последней вершины пути в v, добавить v к пути - Рекурсивно продолжить построение - Если решение не найдено, удалить v из пути (возврат)

Упрощения алгоритма могут включать: - Проверку степеней вершин: вершина может быть в гамильтоновом цикле, только если ее входная и выходная степени не меньше 1 - Использование эвристик для выбора порядка перебора вершин - Применение правила отсечения: если частичное решение невозможно расширить до полного, отсекать его

Временная сложность в худшем случае: O(n!), где n – число вершин.

**8. Поиск потока минимальной стоимости**

Задача о потоке минимальной стоимости: найти поток заданной величины с минимальной общей стоимостью, где каждому ребру (u,v) сопоставлена стоимость c(u,v) за единицу потока.

Алгоритм поиска потока минимальной стоимости: 1. Инициализация нулевого потока 2. Пока величина потока меньше требуемой: - Построение остаточной сети Gf - Поиск кратчайшего пути p от источника s к стоку t в Gf, используя стоимости ребер - Если путь не найден, задача не имеет решения - Определение максимально возможного увеличения потока δ по пути p - Увеличение потока на δ по пути p - Обновление остаточной сети

Для поиска кратчайшего пути можно использовать: - Алгоритм Беллмана-Форда, если в сети есть ребра с отрицательной стоимостью - Алгоритм Дейкстры, если все стоимости неотрицательны

Теорема: Если поток f имеет минимальную стоимость, то в соответствующей остаточной сети Gf нет циклов отрицательной стоимости.

Алгоритм отрицательных циклов (альтернативный подход): 1. Инициализация допустимого потока 2. Пока в остаточной сети есть цикл отрицательной стоимости: - Найти такой цикл - Увеличить поток вдоль цикла, что уменьшит общую стоимость

Временная сложность при использовании алгоритма Беллмана-Форда: O(V·E·|f\_max|).