Д/З по информатике
Стряпко Степан Андреевич
Б04-407

1) Граф – структура данных, состоящая из объектов (вершин графа) и связей между парой вершин (рёбра графа). Т.е. вершины графа содержат информацию об объектах, а рёбра о том как эти объекты могут взаимодействовать (и могут ли вообще: если ребро есть, то могут, если нету то нет)

2) В случае простого графа количество ребер зависит от ориентированности. В случае неориентированного графа M1=(N-1)! (т.к. первая вершина может образовать N-1 новых рёбер (со всеми кроме себя), а каждая последующая на 1 меньше, чем предыдущая) В случае ориентированного графа каждое из рёбер ориентированного представим как ориентированную пару с разными направлениями (такие ребра не являются кратными). Тогда общее число ребер M2=2*(N-1)!

3) Если матрица смежности M такая что MT=M, то граф неориентированный. Объяснение: в ориентированном графе, если из одной вершины (i) есть ребро в другую вершину (j), а в обратную сторону нет, то в матрице M mij=0, а mji=1. (числа обозначают только наличие ребра: 1 – ребро есть, 0 – нету). Тогда при транспонировании mij=1, а mji=0. Следовательно MMT.

4) В общем случае если граф будет взвешенным, то для каждой пары вершин надо будет хранить как существование ребра (0/1) так и его вес. В списке ребер помимо точек начала и конце будем хранить ещё и вес, а в списке смежности помимо существования связи, еще и её вес. В отдельном случае, если мы знаем что нет рёбер с весом 0, можно использовать только список весов, где вес 0 будет если ребра нет.

5) Компонента связности графа – это подграф данного графа, в котором между любой парой вершин есть ребро. Минимальное число компонент в графе 1, если каждая вершина связана с каждой (или есть изолированные точки, а все НЕ изолированные образуют компоненту связности). Максимальное число компонент. Если у вершины есть ребро по которому из неё можно попасть в неё же (петля), то её можно считать компонентой связности. Тогда если у всех точек графа есть петля то максимальное число компонент = N

6) 1. Да, с помощью BFS можно искать циклы. Если в графе нет циклов, то прийти в каждую вершину мы можем только 1 раз (именно прийти, возврат после того как весь граф пройден

не считается). Тогда если мы отследим попадается ли какая-то вершина 2 раза, то ответим на вопрос есть ли в графе циклы. 2. Нет, поиск через DFS выгоднее, т.к. использует меньше памяти. Это происходит из-за того, что если нам нужно найти сам цикл, то надо вернуть путь, а это легче сделать при обходе в глубину, т.к. выгоднее хранить один путь подлиннее, чем множество параллельных путей.

7) Проблема алгоритма при работе с отрицательными весами заключается в том что, если в графе есть цикл суммарный вес которого <0, т о алгоритм будет проходить его бесконечность раз, стремясь как можно сильнее понизить вес итогового пути. Для работы с отрицательными весами можно использовать ограничение на количество использований одинаковых циклов или использовать алгоритм Форда-Беллмана. Этот алгоритм ищет наименьшие пути от данной вершины до всех остальных. Проблема этого алгоритма в том что он медленнее алгоритма Дейкстры. $O(M*N)>O((M+N)*\log N)$

8) 1.За один запуск данный алгоритм находит один из возможных путей из a в b и запоминает его вес. Тогда N-ое выполнение алгоритма нужно для определения всех различных путей. Если после каждого выполнения алгоритма добавить счётчик наименьшего пути и сравнивать его с полученным, то после N-ой операции мы получим наименьшую длину.

9) Зная асимптотики процессов получаем, что Дейкстра будет медленнее чем Форд-Беллман если: $M*N<(M+N)*\log N$ или $M<(N*\log N)/(N-\log N)$. Т.е. количество рёбер значительно меньше чем количество вершин. Зная это условие мы понимаем что длина пути (по количеству рёбер, а не весу) будет $=<$ M, тогда можно ограничить количество операций как M, а не N, что оптимизирует процесс

10) Выбор реализации зависит от конкретной задачи. Основная проблема рекурсивного метода в ограничении на глубину рекурсии (т.е. ограничение на размер графа который можно обработать этим способом), а в динамике такой проблемы нет. В случаях когда можно использовать рекурсию, чаще код понятнее, чем при итеративном подходе. Эффективность использования списка тоже зависит от ситуации, т.к удаление или добавление в конец списка имеет быструю асимптотику O(1). Но в то же время, если придётся удалять/добавлять в начало то асимптотика станет медленной O(N). В свою очередь использование очереди для BFS однозначно эффективно, т.к. проходя граф “по слоям” нам необходим массив который может быстро добавить (удалять) элемент в начало O(1)