Ответы на вопросы по графам

Что такое граф?

Граф — это просто набор точек (их называют вершинами) и линий, которые их соединяют (это рёбра). Эти рёбра могут быть направленными (то есть показывать, куда можно идти) или ненаправленными (просто соединяют вершины). Графы используют, например, в навигации (карты), в социальных сетях (связи между людьми) и во многих других задачах.

Какое максимальное количество рёбер в простом графе? Зависит ли оно от ориентированности?

Если у нас есть N вершин и граф неориентированный, то максимальное число рёбер — это количество возможных пар вершин, то есть:

$$\frac{N(N-1)}{2}.$$

Если граф ориентированный, то каждую пару вершин можно соединить в двух направлениях, поэтому максимум:

$$N(N-1).$$

Получается, что в ориентированном графе рёбер может быть в два раза больше.

Как проверить, ориентирован ли граф, если дана его матрица смежности?

Нужно посмотреть, симметрична ли матрица. Это значит, что если в ней для какой-то пары вершин (i, j) стоит 1, то в (j, i) тоже должна быть 1. Если где-то есть нарушение (например, $A[i][j] = 1$, но $A[j][i] = 0$), значит, граф ориентированный. Если матрица полностью симметрична, то граф неориентированный.

Как изменяется список рёбер и список смежности, если у графа есть веса?

В обычном графе мы просто храним список рёбер как пары (u, v). Но если у рёбер есть веса (например, расстояние между городами), то надо добавить ещё одно число — вес. Теперь ребро будет выглядеть как (u, v, w). В списке смежности тоже меняется структура: раньше мы просто писали список соседей для каждой вершины, а теперь к каждому соседу нужно добавлять вес. Например, раньше было $\{1 : [2, 3]\}$, а теперь $\{1 : [(2, 5), (3, 10)]\}$, где 5 и 10 — веса рёбер.

Что такое компонента связности? Сколько их может быть в графе?

Это просто группа вершин, которые соединены между собой, но не соединены с остальными. Если граф полностью соединён, то у него одна компонента связности. Если все вершины вообще никак не связаны друг с другом, то каждая вершина — отдельная компонента, и их будет N.

Можно ли искать циклы с помощью BFS? Или лучше использовать DFS?

BFS можно использовать для поиска циклов, но чаще используют DFS. BFS хорошо находит нечётные циклы в двудольных графах (когда нужно проверить, можно ли раскрасить граф в два цвета). DFS удобнее для поиска циклов в ориентированных графах: если при обходе мы возвращаемся в уже посещённую вершину, значит, нашли цикл. Кроме того, DFS обычно требует меньше памяти, чем BFS, который может запомнить слишком много вершин.

Почему у Дейкстры нельзя отрицательные рёбра? Можно ли заставить его работать с ними?

Дейкстра работает так: он всегда берёт ближайшую вершину и уверен, что короткий путь до неё уже найден. Но если есть отрицательные рёбра, то позже может появиться более короткий путь, а Дейкстра его уже не пересчитает. Чтобы работать с отрицательными рёбрами, лучше использовать алгоритм Форда-Беллмана. Можно ещё применить метод пересчёта весов (Johnson's Algorithm), но саму Дейкстру без изменений заставить работать нельзя.

Зачем Форд-Беллман запускают N-й раз? В каких случаях поиск кратчайшего пути может быть некорректным?

Мы запускаем Форд-Беллман N-й раз, чтобы проверить, есть ли отрицательный цикл. Если после $N-1$ итераций какой-то путь продолжает уменьшаться, значит, в графе есть бесконечно малый путь (можно ходить по циклу и уменьшать вес бесконечно). В таком случае задача поиска кратчайшего пути просто не имеет смысла, потому что путь можно уменьшать сколько угодно.

Когда Форд-Беллман может быть быстрее Дейкстры? Как его можно ускорить?

Обычно Дейкстра быстрее, но Форд-Беллман лучше работает в разреженных графах (если рёбер мало, около $O(N)$). Можно ускорить его с помощью

алгоритма SPFA (Shortest Path Faster Algorithm), который использует очередь и иногда работает за $O(M)$ вместо $O(NM)$.

Когда лучше использовать рекурсивный DFS, а когда итеративный? Можно ли использовать список как стек? А как лучше делать очередь в BFS?

Рекурсивный DFS проще писать, но если граф очень глубокий, стек вызовов может переполниться. Итеративный DFS работает надёжнее, но требует явного стека (обычно используют list). Использовать список (list) как стек в DFS можно, но deque быстрее, потому что pop() в list работает за $O(1)$, а pop(0) за $O(N)$. В BFS нельзя использовать список как очередь, потому что pop(0) медленный. Лучше применять collections.deque, так как popleft() там выполняется за $O(1)$.