**Построение гамильтонова цикла**

**Гамильтоновым циклом** в графе называют цикл, проходящий через все вершины. Также можно определить понятие гамильтонова пути — незамкнутого пути, также проходящего через все вершины.

Построение гамильтонова цикла — сложная задача, в настоящее время неизвестно эффективного алгоритма его решения. Более того, скорее всего такого алгоритма (решающего  задачу за сложность, являющуюся многочленом от числа вершин в графе n) скорее всего просто не существует, но это одна из нерешенных на сей момент проблема теории сложности алгоритмов.

Можно придумать переборное решение, сложность которого будет порядка O(n!). Например, если перенумеровать вершины в графе, то номера вершин в порядке следования их в гамильтоновом цикле образуют некоторую перестановку чисел от 1 до n. Можно перебрать все n! возможных перестановок и для каждой из них проверить, что данная перестановка соответствует циклу на графе, то есть каждые два соседних элемента в перестановке, а также первый и последний элемент перестановки соединены ребром.

Для перебора перестановок можно использовать ранее разобранный алгоритм перебора с возвратом. Запишем этот алгоритм так, чтобы исключить заведомо неподходящие варианты, а именно, к уже построенной части пути будем добавлять только те вершины, которые соединены ребром с последней вершиной в пути (и не были посещены ранее), добавив новую вершину к пути рекурсивно запускаем алгоритм из новой вершины.

Во многом этот алгоритм напоминает *алгоритм поиска в глубину*, но главное его отличие заключается в том, что если из какой-то вершины не удается продолжить путь дальше (то есть были рассмотрены все ребра и все возможные продолжения привели в тупик), то алгоритм возвращается в предыдущую вершину, при этом покинутая вершина «перекрашивается», то есть с нее снимается отметка о том, что эта вершина была посещена ранее. При этом алгоритм может вернуться в эту вершину еще раз, уже по другому пути (и даже обязан это сделать, если в графе существует гамильтонов путь, так как гамильтонов путь проходит через все вершины).

Пусть n — число вершин в графе, вершины пронумерованы числами от 0 до n−1. Граф задан матрицей смежности A. В глобальной переменной Path будет храниться список вершин, входящих в путь.  Функция hamilton принимает в качестве параметра номер вершины, добавляемой к пути и возвращает значение true, если удалось построить гамильтонов путь и false, если не удалось. Причем если путь построить удалось, то построенный путь будет храниться в списке Path

**Пример на языке C++**

**vector <bool> Visited;**

**vector <int> Path;**

**void hamilton(int curr)**

**{**

**Path.append(curr);**

**if (Path.size() == n)**

**if (A[Path[0]][Path.back()] == 1)**

**return true;**

**else**

**{**

**Path.pop\_back();**

**return false;**

**}**

**Visited[curr] = true;**

**for (int nxt = 0; nxt < n; ++nxt)**

**if (A[curr][nxt] == 1 && !Visited[nxt])**

**if (hamilton(next))**

**return true;**

**Visited[curr] = false;**

**Path.pop\_back();**

**return false;**

**}**

**Пример на языке Python**

**Visited = [False] \* n**

**Path = []**

**def hamilton(curr):**

**Path.append(curr)**

**if len(Path) == n:**

**if A[Path[0]][Path[-1]] == 1:**

**return True**

**else:**

**Path.pop()**

**return False**

**Visited[curr] = True**

**for next in range(n):**

**if A[curr][next] == 1 and not Visited[next]:**

**if hamilton(next):**

**return True**

**Visited[curr] = False**

**Path.pop()**

**return False**

Функция Hamilton прежде всего добавляет вершину curr в конец списка Path. При этом если длина списка стала равна n, то есть все вершины включены в путь Path, проверяется, что первая и последняя вершина в пути соединены ребром (это не требуется при помощи гамильтонова пути), если это так — то алгоритм возвращает True (цикл найден), в противном случае из списка Path удаляется последний элемент и алгоритм возвращает False (цикл не найден).

Если же длина списка меньше n, то вершина curr отмечется, как посещенная и осуществляется перебор дальнейших продолжений. Последовательно перебираются все оставшиеся вершины next и если вершина next соединена ребром с curr и вершина next не была посещена, то алгоритм рекурсивно запускается из вершины next, пытаясь сделать продолжение пути в вершину next. При этом если рекурсивный вызов из вершины next вернет True, то есть удалось построить цикл, то алгоритм сразу же возвращает True, при этом из списка Path ничего не удаляется, поэтому Path будет хранить полный гамильтонов цикл. Если же ни одно из продолжений не получилось, то осуществляется «откат» вершины curr — она помечается, как непосещенная, удаляется из конца списка Path и управление передается назад, на последнюю вершину в списке Path.

Видно, что сложность алгоритма может быть не менее, чем n!, поэтому для больших графов такой алгоритм непригоден.

**Задача коммивояжера**

Близкой задачей к задаче нахождения гамильтонова цикла является задача коммивояжера. Коммивояжеру (странствующему торговцу) необходимо посетить n городов и вернуться домой. Коммивояжер не хочет посещать города более одного раза и при этом хочет проделать наиболее короткий путь. То есть в неориентированном взвешенном графе необходимо найти путь наименьшей стоимости.

Задача коммивояжера решается аналогично задаче о гамильтоновом пути, но при этом нужно перебрать все возможные пути. При замыкании пути нужно вычислить его вес (лучше это делать не в конце замыкания, а одновременно с добавлением следующей вершины увеличивать вес построенного фрагмента пути на вес рассмотренного ребра) и сравнить вес найденного пути с весом наилучшего известного пути.