Связный список (LinkedList) -

Это структура данных, состоящая из узлов содержащих данные и ссылки (связки) на следующий или предыдущий узел списка. Принципиальным преимуществом перед массивом является структурная гибкость, те порядок элементов связного списка может не совпадать с порядковым расположением расположения данных в памяти компьютера, а порядок обхода списков всегда явно задается его внутренними связями

Списки бывают:

* Односвязный (линейный однонаправленный список) - это структура данных, состоящая из элементов одного типа связанных между собой последовательно посредством указателей. Каждый элемент списка имеет указатель на следующий элемент, последний элемент списка показывает Nan, элемент на которого нет указателя является первым элементом списка. Ссылка в каждом узле указывает на следующий узел в списке. В односвязном списке можно передвигаться только в сторону конца списка
* Двусвязный (двунаправленный связанный список) - ссылки в каждом узле указывают на предыдущий и на последующий узел в списке. Двусвязный список допускает только последовательный доступ к элементам, но при этом дает перемещаться в обе стороны
* Кольцевой - разновидность двусвязного списка, может быть односвязным или двусвязным. Последний элемент кольцевого списка содержит указатель на 1 элемент списка, а первый на последний

Преимущество связных списков:

* Эффективное добавление и удаление элементов
* Размер ограничен только обьемом памяти компьютера и разрядностью указателей
* Динамическое добавление и удаление элементов

Недостатки

* Сложность прямого доступа к элементу, определение физ адреса по его индексу в списке
* На поля указатели расходуется доп память
* Некоторые операции со спиками медленее чем с массивами
* Соседние элементы списка могут быть распределены в памяти не локально что снизит эффективность данных процессора

Граф -

Это совокупность точек соединеных в линии. Точками называют вершинами или узлами, а линии рёбрами или дугами

* **Степень входа** в вершину – количество входящих в неё рёбер.
* **Степень выхода** – количество исходящих рёбер.
* **Полный граф** – содержит рёбра между всеми пары вершин. **Взвешенные графы** – в рёбрах поставлено конкретное числовое значение, а это значение называется весом ребра. Когда у ребра оба ребра совпадают, такое ребро называется **петлёй**.

Виды графов:

* Неориентированный граф – подмножество пар В и Е, где В подмножество вершин, а Е подмножество ребер графов, которые соединяют эти вершины.
* Ориентированный граф. В нем всё тоже самое кроме того, что ребра тут называются дугами, и они четко соединяют одну точку с другой.

Способы представления:

* Матрица смежности для графа из n вершин хранится в виде двумерного массива размером n\*n, вершины графа в этом случае задаются номерами (индексами строк и столбцов матрицы), а ячейка графов (m[i][j]) отражает наличие дуги между соответствующими вершинами.
* Матрица инцидентности. Для графа из n вершин и m дуг, хранится в виде двумерного массива размером n\*m, ячейка матрица отражает инцидентности ребра j в вершине i, то есть тот факт что ребро выходит/входит в вершину i. Если ребро не связано с вершиной, тогда в соответствующей ячейки записывается 0, в противном случае 1 (если граф не взвешен)

Работая с приведенными матрицами, возможно найти график кратчайшего пути, однако у них есть недостатки:

* Избыточность. Зачастую в графах ребра существуют между небольшим количеством вершин, поэтому в матрице смежности будет огромное количество нулей. В матрице инцидентности в каждом столбце может быть лишь два ненулевых значения. На хранение нулей тратится память.
* Недостаточная расширяемость. В матрицу смежности можно добавлять новые дуги, но чтобы добавить вершину, нужно создавать новую матрицу большего размера и копировать в неё данные из старой. В матрице инцидентности такие проблемы возникнут как при добавлении дуг, так и при вершин

….. для каждой вершины при этом хранится список с номерами смежных вершин или инцидентных ребер. В качестве структуры данных при этом могут использоваться массивы или связнные списки.

Поиск в глубину

В процессе поиска идем в глубь графа настолько на сколько это возможно. Следуя алгоритму, последовательно обойдем все вершины графа которые доступны изначальному. Если ребро ведет в не пройденную до этого момента вершину то алгоритм запускается с нее. В случае если ребер которые ведут в не пройденную врешину больше нет, то идет обратно

Применение алгоритма поиска в глубину:

* Поиск любого пути в графе
* Проверка я вляется ли вершина дерева предком другой
* Поиск наименьшего общего предка
* Топологическая сортировка
* Поиск компонент связанности

Для реализации алгоритма удобно использовать стек или рекурсию

Поиск в ширину

Алгоритм позволяет найти кратчайший или содержащий нименьшее колво ребер путь из одной вершины графа для всех остальных вершин. В нем сначала посещаются все вершины смежные с текущей, а потом все потомки.

Применение алгоритма поиска в ширину:

* Поиск кратчайшего пути в не взвешенном графе
* Поиск компонент связности
* Нахождение решения какой либо задачи с наименьшем числом ходов
* Задача нахождения всех ребер, лежащих на каком либо кратчайшем пути заданных парой вершин
* Задача нахождения всех вершин, лежащих на каком либо кратчайшем пути заданных парой вершин

Для реализации алгоритма удобно использовать очередь