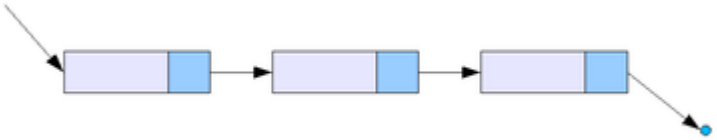
12. Динамические структуры данных: списки, стеки, очереди, деревья.

Если до начала работы с данными невозможно определить, сколько памяти потребуется для их хранения, память следует распределять во время выполнения программы по мере необходимости отдельными блоками. Блоки связываются друг с другом с помощью указателей. Такой способ организации данных называется *динамической структурой данных*, поскольку она размещается в динамической памяти и ее размер изменяется во время выполнения программы.

Из динамических структур в программах чаще всего используются *списки, стеки, очереди и деревья.* Они различаются способами связи отдельных элементов и допустимыми операциями. Динамическая структура, в отличие от массива или записи, может занимать несмежные участки оперативной памяти.

Свя́зный спи́сок — базовая динамическая структура данных в информатике, состоящая из узлов, каждый из которых содержит как собственно данные, так и одну или две ссылки на следующий и/или предыдущий узел списка



Основные виды списков: односвязный, двусвязный, кольцевой.

Расширенные реализации: Список с пропусками, Развёрнутый связный список, XOR-связный список

Достоинства:

- лёгкость добавления и удаления элементов
- размер ограничен только объёмом памяти компьютера и разрядностью указателей
- динамическое добавление и удаление элементов

Недостатки:

- сложность определения адреса элемента по его индексу в списке
- на поля-указатели (указатели на следующий и предыдущий элемент) расходуется дополнительная память (в массивах, например, указатели не нужны)
- работа со списком медленнее, чем с массивами, так как к любому элементу списка можно обратиться, только пройдя все предшествующие ему элементы
- элементы списка могут быть расположены в памяти разреженно, что окажет негативный эффект на кэширование процессора
- над связными списками гораздо труднее (хотя и в принципе возможно) производить параллельные векторные операции, такие как вычисление суммы

Стек — структура данных, представляющая собой список элементов, организованных по принципу *LIFO*

Чаще всего принцип работы стека сравнивают со стопкой тарелок: чтобы взять вторую сверху, нужно снять верхнюю.

Возможны три операции со стеком: добавление элемента (*push*), удаление элемента (*pop*) и чтение головного элемента (*peek*).

Зачастую стек реализуется в виде однонаправленного списка.

В некоторых языках (например, Lisp, Python) стеком можно назвать любой список, так как для них доступны операции pop и push.

Стеки широко применяются в системном программном обеспечении, компиляторах, в различных рекурсивных алгоритмах, стековых машинах (пример стековой машины - калькулятор Электроника Б3-21, вычисляет выражения заданные в обратной польской записи).

Например, локальные переменные в си-подобных языках выделяются на стеке (это быстрее чем выделение в куче + предотвращает утечки). Большинство соглашений о вызове подпрограмм используют стек для хранения адреса возврата и передачи аргументов. Можно упомянуть про атаку buffer overflow .

О́чередь — это динамическая структура данных, добавление элементов в которую выполняется в один конец, а выборка — из другого конца. Другие операции с очередью не определены. При выборке элемент исключается из очереди. Очередь реализует принцип обслуживания FIFO.

В программировании очереди применяются очень широко — например, при буферизованном вводе-выводе, диспетчеризации задач в операционной системе.

Выделяют также:

- очередь с приоритетом
- двусвязную очередь (элементы можно добавлять и удалять как в начало, так и в конец, то есть дисциплинами обслуживания являются одновременно FIFO и LIFO).

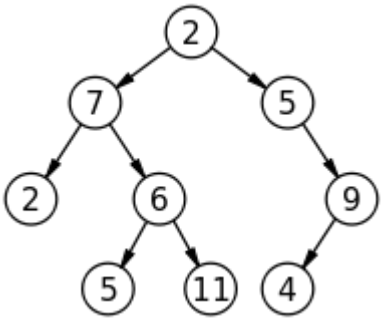
Дерево — одна из наиболее широко распространённых структур данных в информатике, эмулирующая древовидную структуру в виде набора связанных узлов. Является связанным графом, не содержащим циклы.

Существует много разновидностей деревьев:

- Двоичное
- Красно-черное
- АВЛ-дерево
- И др.

Рассмотрим двоичное дерево, как наиболее распространенное.

Двоичное *дерево* — это динамическая структура данных, состоящая из узлов, каждый из которых содержит кроме данных не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка. Начальный узел называется *корнем* дерева.

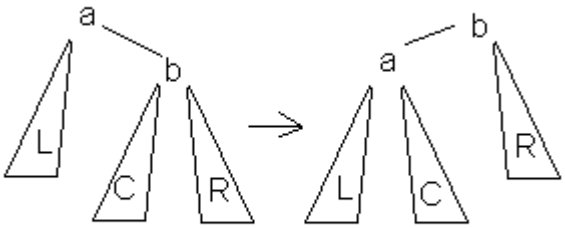


Если дерево организовано таким образом, что для каждого узла все ключи его левого поддерева меньше ключа этого узла, а все ключи его правого поддерева — больше, оно называется *деревом поиска*. Одинаковые ключи не допускаются. При такой организации для поиска элемента нужно в среднем O(log n) времени.

Балансировка.

Всегда желательно, чтобы все пути в дереве от корня до листьев имели примерно одинаковую длину, т.е. чтобы глубина и левого, и правого поддеревьев была примерно одинакова в любом узле. В противном случае теряется производительность.

Для балансировки дерева применяется операция "поворот дерева". Поворот налево:



- было Left(A) = L, Right(A) = B, Left(B) = C, Right(B) = R
- поворот меняет местами A и B, получая Left(A) = L, Right(A) = C, Left(B) = A, Right(B) = R
- также меняется в узле Parent(A) ссылка, ранее указывавшая на A, после поворота она указывает на B.

Поворот не нарушает упорядоченность дерева, и оказывает предсказуемое (+1 или -1) влияние на глубины всех затронутых поддеревьев.

Для принятия решения о том, какие именно повороты нужно совершать после добавления или удаления, используются такие алгоритмы, как "красно-чёрное дерево" и АВЛ.

Оба они требуют дополнительной информации в узлах - 1 бит у красно-черного или знаковое число у АВЛ.

Общее применение деревьев:

- управление иерархией данных;
- упрощение поиска информации;
- сортировка значений;
- синтактический разбор арифметических выражений, оптимизация программ;
- форма принятия многоэтапного решения (см. деловые шахматы).