Линейные структуры данных с конечными точками

Стек

Представьте, что у вас есть куча листов бумаги.

Мы кладём один лист в стопку. Теперь мы можем получить доступ только к верхнему листу.

Мы кладём ещё один лист в стопку. Предыдущий лист теперь скрыт и доступ к нему невозможен, мы можем использовать верхний лист. Когда мы закончим с верхним листом, мы можем убрать его со стопки, открыв доступ к лежащему под ним.

В этом заключается идея стека. Стек — это структура LIFO. Это расшифровывается как Last In First Out («последним вошёл, первым вышел»). При добавлении и удалении из стека последний добавленный элемент будет первым удаляемым.

Для стека нужно всего три операции: Push, Pop и Top.

Push добавляет объект в стек. Pop удаляет объект из стека. Top даёт самый последний объект в стеке. Эти контейнеры в большинстве языков являются частью стандартных библиотек. В C++ они называются *stack*. В Java и C# это *Stack*. (Да, единственная разница в названии с заглавной буквы.) Внутри стек часто реализуется как динамический массив. Как вы помните из этой структуры данных, самыми быстрыми операциями на динамических массивах являются добавление и удаление элементов из конца. Поскольку стек всегда добавляет и удаляет с конца, обычно push и pop объектов в стеке выполняется невероятно быстро.

Очередь

Представьте, что вы стоите в очереди за чем-то.

Первого человека в очереди обслуживают, после чего он уходит. Потом обслуживается и уходит второй в очереди. Другие люди подходят к очереди и встают в её конец. Вот в этом заключается идея структуры данных «очередь».

Очередь — это структура FIFO (First In First Out, «первым зашёл, первым вышел»).

При добавлении и удалении из очереди первый добавляемый элемент будет первым извлекаемым. Очереди нужно только несколько операций: Push_Back, Pop_Front, Front и Back. Push_Back добавляет элемент к концу очереди. Pop_Front удаляет элемент из начала очереди. Front и Back позволяют получить доступ к двум концам очереди.

Программистам часто нужно добавлять или удалять элементы из обоих концов очереди. Такая структура называется двухсторонней очередью (double ended queue, deque). В этом случае добавляется ещё пара операций: Push_Front и Pop_Back. Эти контейнеры тоже включены в большинство основных языков. В C++ это *queue* и *deque*. Java определяет интерфейсы для очереди и двухсторонней очереди, а затем реализует их через *LinkedList*. В C# есть класс *Queue*, но нет класса Deque.

Внутри очередь и двухсторонняя очередь могут быть устроены довольно сложно. Поскольку объекты могут поступать и извлекаться с любого конца, внутренний контейнер должен уметь наращивать и укорачивать очередь с начала и с конца. Во многих реализациях используются множественные страницы памяти. Когда любой из концов разрастается за пределы текущей страницы, добавляется дополнительная страница. Если страница больше не нужна, то она удаляется. В Java используется следующий способ: для связанного списка требуется немного дополнительной памяти, а не страниц памяти, но для этого языка такая реализация вполне работает.

Очередь с приоритетом

Это очень распространённая вариация очереди. Очередь с приоритетом очень похожа на обычную очередь.

Программа добавляет элементы с конца и извлекает элементы из начала. Разница в том, что можно задавать приоритеты определённым элементам очереди. Все самые важные элементы обрабатываются в порядке FIFO. Потом в порядке FIFO обрабатываются элементы с более низким приоритетом. И так повторяется, пока не будут обработаны в порядке FIFO элементы с самым низким приоритетом.

При добавлении нового элемента с более высоким приоритетом, чем остальная часть очереди, он сразу же перемещается в начало очереди. В C++ эта структура называется *priority_queue*. В Java это *PriorityQueue*. В стандартной библиотеке C# очереди с приоритетом нет. Очереди с приоритетом полезны не только для того, чтобы встать первым на очереди к принтеру организации. Их можно использовать для удобной реализации алгоритмов, например, процедуры поиска A*. Наболее вероятным результатам можно отдать более высокий приоритет, менее вероятным — более низкий. Можно создать собственную систему для сортировки и упорядочивания поиска A*, но намного проще использовать встроенную очередь с приоритетом.

Заключение

Стеки, очереди, двухсторонние очереди и очереди с приоритетом можно реализовать на основе других структур данных. Это не фундаментальные структуры данных, но их часто используют. Они очень эффективны, когда нужно работать только с конечными элементами данных, а серединные элементы не важны.