Стек, очередь, дек

Кроме стандартных структур: список, множество, словарь, в программировании используется и ряд других структур. Особенность каждой из них состоит в способе хранения данных и алгоритме доступа к элементам структуры. При использовании каждой структуры есть определенные ограничения – то, что структура «не умеет делать быстро». Например, операция добавления элемента в конец списка выполняется быстро, а удаление первого элемента выполняется за длину списка. Структура очередь выполняет удаление элемента из начала быстро, а удаление произвольного времени выполняет долго.

Познакомимся с несколькими структурами и разберем их реализацию в python.

Стек (англ. *stack* – стопка) – это линейный список, в котором элементы добавляются и удаляются только с одного конца – вершины стека. Обозначается стек как LIFO - *Last In – First Out* «последним пришел – первым ушел»

Очередь (анг. *queue* – очередь) - это линейный список, в котором элементы добавляются в конец и удаляются из начала. Обозначается *FIFO - First In – First* «первым пришёл – первым ушел».

Дек (от англ. *deque - double ended queue,* очередь с двумя концами) – это линейный список, в котором можно добавлять и удалять элементы как с одного, так и с другого конца.

Очередь с приоритетом (англ. *priority queue* – приоритетная очередь) — абстрактный тип данных, поддерживающий две обязательные операции — добавить элемент и извлечь максимум (минимум). Предполагается, что для каждого элемента можно вычислить его приоритет — действительное число. Элемент с более высоким приоритетом находится перед элементом с более низким приоритетом. Если у элементов одинаковые приоритеты, они располагаются в зависимости от своей позиции в очереди. Очередь с приоритетом работает, как обычная очередь с удалением объекта, стоящего на первом месте, однако внутренний порядок элементов зависит от их приоритета.

Классический способ реализации очереди с приоритетом – использование структуры данных под названием **двоичная куча**. Она позволяет ставить в очередь и извлекать из неё элементы за O(logn), за это время происходит. Кучи - это бинарные деревья, для которых каждый родительский узел имеет значение меньшее или равное любого из его детей (куча на минимуме). Эта реализация использует массивы, для которых $heap[k] \leq heap[2k+1]$ и $heap[k] \leq heap[2k+2]$ для всех k, считая элементы, начиная с нуля. Для сравнения, несуществующие элементы считаются бесконечными. Интересное свойство кучи состоит в том, что её наименьший элемент всегда является корнем, heap[0].

Рассмотрим различные механизмы, реализующие эти структуры в python.

1. Использование структуры список для реализации стека

Поскольку **стек** – это линейная структура данных с переменным количеством элементов, для работы со стеком в программе на языке Python удобно использовать список. Вершина стека будет находиться в конце списка. Тогда для добавления элемента на вершину стека можно применить метод append: stack.append (x)
Чтобы удалить элемент из стека, используется метод pop: x = stack.pop()

Метод pop – это функция, которая выполняет две задачи:

1) удаляет последний элемент списка (если вызывается без параметров) за O(1);
2) возвращает удалённый элемент как результат функции, так что его можно сохранить в какой-либо переменной.

Использование специальных конструкций (см. ниже) не увеличит скорость обращения к элементам стека, поэтому чаще всего для реализации стека используют списки в Python и vector в C++.

2. Использование класса deque модуля collections

Модуль collections - предоставляет специализированные типы данных, на основе словарей, кортежей, множеств, списков. Класс **deque** модуля collections реализует двухконечную очередь, которая поддерживает добавление и удаление элементов с обоих концов за O(1) времени. Объекты deque представлены в виде двусвязных списков, что дает им высокую производительность для входящих и выходящих элементов, но при этом у него плохая производительность O(n) при работе с элементами в середине очереди. В связи с тем, что deque поддерживает вставку и удаление элементов и с начала и с конца, на deque можно реализовывать и очереди и стеки.

Быстрые методы класса deque

1. **append**(x) - добавляет x в конец.
2. **appendleft**(x) - добавляет x в начало.
3. x = **pop**() - удаляет и возвращает последний элемент очереди.
4. x = **popleft**() - удаляет и возвращает первый элемент очереди.
5. **clear**() - очищает очередь.

Операции за O(n)

1. **count**(x) - количество элементов, равных x
2. **remove**(value) - удаляет первое вхождение value.
3. **reverse**() - разворачивает очередь

3. Использование модуля heapq для реализации очереди с приоритетом

Этот модуль обеспечивает реализацию алгоритма очереди с кучами, также известный как алгоритм очереди с приоритетами. При подключении модуля heapq вы получаете доступ к функциям, которые работают над списком как над кучей.

Модуль поддерживает следующие функции:

1. heapq.**heapify**(arr) - превращает список *arr* в кучу, на месте, за линейное время.
2. heapq.**heappush**(*heap, item*) - добавляет значение объекта *item* в кучу *heap*, элемент «находит» свое место в списке, так, чтобы список сохранял свойства кучи.
3. heapq.**heappop**(*heap*) - удаляет и возвращает наименьший элемент кучи *heap*, перестраивает список так, . Если куча пуста, возвращает ошибку.
4. heapq.**heappushpop**(*heap, item*) - добавляет объект в кучу, затем удаляет и возвращает наименьший элемент из кучи. Комбинированное действие работает более эффективно, чем **heappush()** с последующим вызовом **heappop()**.

Стоит отметить, что при использовании функций модуля пользователь сам должен следить за корректностью операций. Если взять произвольный список и применить к нему одну из функций, например **heappop**(*heap*), но удалится первый элемент списка не зависимо от того является ли он

наименьшим. Т.к. корректность операций гарантируется только для списков, которые являются кучей.

4. Использование модуля queue

Модуль **queue** реализует механизм очереди. Он особенно полезен в многопоточном программировании, когда необходимо безопасно передавать информацию между потоками. Класс **queue** этого модуля реализует всю необходимую семантику блокировки.

Модуль предоставляет реализации трех типов очередей, единственная разница которых это порядок получаемых значений.

class queue.**Queue**(*maxsize*)
Класс реализующий очередь FIFO. *maxsize* - параметр типа integer, который устанавливает предел для числа элементов, которые могут быть помещены в очередь. Вставка новых элементов блокируется, как только этот размер был достигнут, до тех пор пока элементы не будут удалены из очереди. Если значение параметра равно или меньше нуля, то очередь будет бесконечной.

class queue.**LifoQueue**(*maxsize*)
Класс реализующий очередь LIFO, или по-другому "стек. Параметр *maxsize* аналогичен параметру в классе queue.Queue.

class queue.**PriorityQueue**(*maxsize*)
Класс реализующий очередь с приоритетами. Параметр *maxsize* аналогичен параметру в классе queue.Queue. Первыми из очереди забираются элементы с меньшим приоритетом, полученным с помощью функции sorted().

Классы Queue, LifoQueue, PriorityQueue предоставляют следующие методы:

1. queue.**put**(*item, [block[, timeout]]*)

Помещает объект item в очередь. Если block=True и timeout не задан (None по умолчанию), то при необходимости произойдет блокировка до тех пор пока в очереди не будет доступного места.

2. queue.**get**(*[block, [timeout]]*)

Удалить и возвратить элемент из очереди. Если block=True и timeout не задан (None по умолчанию), произойдет блокировка до тех пор пока элемент не будет доступен.

3. queue.**empty**() - возвращает True если очередь пуста, False в противном случае.

4. x = queue.**qsize**() - возвращаем аппроксимированный размер очереди.

5. queue.**full**() - возвращает True если очередь заполнена, False в противном случае.

Примеры задач

Задача «Постфиксная запись»

В постфиксной записи (или обратной польской записи) операция записывается после двух операндов. Например, сумма двух чисел A и B записывается как A B +. Запись B C + D * обозначает привычное нам (B + C) * D, а запись A B C + D * + означает A + (B + C) * D. Достоинство постфиксной записи в том, что она не требует скобок и дополнительных соглашений о приоритете операторов для своего чтения.

Формат входных данных: в единственной строке записано выражение в постфиксной записи, содержащее однозначные числа и операции +, -, *.

Формат выходных данных: необходимо вывести значение записанного выражения.

Пример

Вход	***Выход***
8 9 + 1 7 - *	-102

Рекомендации по решению
Заведём числовой стек. Считаем все данные в список символов и пройдемся по нему. Если очередной элемент списка число, то добавляем его в стек. Если элемент символ, определяющий операцию, то вынимаем из стека два последних элемента и записываем в него результат соответствующего действия. В конце программы в стеке останется один элемент, это и есть ответ.

Задача «Игра в пьяницу»

В игре в пьяницу карточная колода раздается поровну двум игрокам. Далее они вскрывают по одной верхней карте, и тот, чья карта старше, забирает себе обе вскрытые карты, которые кладутся под низ его колоды. Тот, кто остается без карт – проигрывает.

Для простоты будем считать, что все карты различны по номиналу, а также, что самая младшая карта побеждает самую старшую карту ("шестерка берет туза").

Игрок, который забирает себе карты, сначала кладет под низ своей колоды карту первого игрока, затем карту второго игрока (то есть карта второго игрока оказывается внизу колоды).

Напишите программу, которая моделирует игру в пьяницу и определяет, кто выигрывает. В игре участвует 10 карт, имеющих значения от 0 до 9, большая карта побеждает меньшую, карта со значением 0 побеждает карту 9.

Формат входных данных: программа получает на вход две строки: первая строка содержит 5 карт первого игрока, вторая – 5 карт второго игрока. Карты перечислены сверху вниз, то есть каждая строка начинается с той карты, которая будет открыта первой.

Формат выходных данных: программа должна определить, кто выигрывает при данной раздаче, и вывести слово first или second, после чего вывести количество ходов, сделанных до выигрыша. Если на протяжении 10^6 ходов игра не заканчивается, программа должна вывести слово botva.

Пример

Вход	***Выход***
1 3 5 7 9 2 4 6 8 0	second 5

Рекомендации по решению

Для написания этой программы достаточно реализовать структуру "очередь" и дальше смоделировать все то, о чём написано в условии. А именно:

1) Задаём цикл на 10^6 итераций.
2) При каждом заходе в цикл берём по первому элементу из очередей, эмулирующих колоды первого и второго игроков.
3) Сравниваем их согласно описанной в условии методике.
4) Добавляем две взятые карты в конец колоды-очереди игрока выигравшего на данном сравнении.
5) Если при очередной итерации одна из очередей оказывается пуста, то выводим победителя и количество совершённых итераций.
6) Если все 10^6 итераций успешно выполнились - выводим "botva".

Используемые ресурсы

1. Марк Саммерфилд. Программирование на Python 3. Подробное руководство,
2. Марк Лутс. Изучаем Python
3. Девид Бизли, Python.Подробный справочник
4. Python v3.0.1 documentation. Документация по Python 3. Сайт: https://docs.python.org/3.0
5. Python: модуль Queue. Перевод документации. Примеры работы с очередями. Сайт: http://john16blog.blogspot.com/2012/05/python-queue.html