1.1)Создание мультисписка на питоне

from typing import List, Any

class Multilist:

def \_\_init\_\_(self):

self.sublists: List[List[Any]] = []

def add\_sublist(self, items: List[Any]):

self.sublists.append(items)

def flatten(self) -> List[Any]:

return [item for sublist in self.sublists for item in sublist]

def get\_sublist(self, index: int) -> List[Any]:

return self.sublists[index]

# Использование

ml = Multilist()

ml.add\_sublist([1, 2, 3])

ml.add\_sublist(['a', 'b'])

print(ml.flatten()) # [1, 2, 3, 'a', 'b']

1.2) создание очереди на питоне

from collections import deque

class Queue:

def \_\_init\_\_(self):

self.\_queue = deque()

def enqueue(self, item):

self.\_queue.append(item)

def dequeue(self):

if self.is\_empty():

raise IndexError("Queue is empty")

return self.\_queue.popleft()

def is\_empty(self):

return len(self.\_queue) == 0

def size(self):

return len(self.\_queue)

1.3) Создание дек на питоне

from collections import deque

class Deque:

def \_\_init\_\_(self):

self.\_deque = deque()

def add\_front(self, item):

self.\_deque.appendleft(item)

def add\_rear(self, item):

self.\_deque.append(item)

def remove\_front(self):

if self.is\_empty():

raise IndexError("Deque is empty")

return self.\_deque.popleft()

def remove\_rear(self):

if self.is\_empty():

raise IndexError("Deque is empty")

return self.\_deque.pop()

def is\_empty(self):

return len(self.\_deque) == 0

def size(self):

return len(self.\_deque)

1.4)

import heapq

from typing import Any, Tuple

class PriorityQueue:

def \_\_init\_\_(self):

self.\_heap = []

self.\_index = 0

def push(self, item: Any, priority: int):

heapq.heappush(self.\_heap, (priority, self.\_index, item))

self.\_index += 1

def pop(self) -> Any:

if self.is\_empty():

raise IndexError("Priority queue is empty")

return heapq.heappop(self.\_heap)[2]

def is\_empty(self) -> bool:

return len(self.\_heap) == 0

def size(self) -> int:

return len(self.\_heap)