import threading as th

from copy import deepcopy

class Node:

def \_\_init\_\_(self, parent=None, state=[]):

self.parent = parent

self.generator\_lock = th.Lock()

self.generator = self.\_child\_gen()

self.state = state

def \_child\_gen(self):

for i in range(1, 4):

state = deepcopy(self.state) + [i]

yield Node(self, state)

def next\_child(self):

with self.generator\_lock:

return next(self.generator, None)

def is\_leaf(self):

return len(self.state) >= 10

def \_\_repr\_\_(self):

return '<Node state="{}">'.format(self.state)

class Worker:

def \_\_init\_\_(self, id, searcher):

self.searcher = searcher # type: Searcher

self.id = id

def \_\_call\_\_(self):

print("start worker: {}".format(self.id))

while not self.searcher.is\_end():

self.\_run()

print("end worker: {}".format(self.id))

def \_run(self):

node = self.searcher.get\_last\_node()

if node is None:

return

if node.is\_leaf():

self.searcher.remove\_node(node)

self.searcher.add\_result(node)

return

bounds = self.searcher.get\_bounds()

if not self.satisfy\_bounds(node, bounds):

self.searcher.remove\_node(node)

return

child = node.next\_child()

if child is None:

self.searcher.remove\_node(node)

else:

self.searcher.add\_node(child)

def satisfy\_bounds(self, node, bound):

return True

class Searcher:

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

self.root\_node = Node()

self.nodes = [self.root\_node] # TODO: priority queue

self.nodes\_lock = th.Lock()

self.\_is\_end = False

self.workers = [

Worker(i, self) for i in range(8)

]

self.results = set()

self.results\_lock = th.Lock()

self.bounds = [None, None]

self.bounds\_lock = th.Lock()

self.threads = []

def run(self):

self.threads = [

th.Thread(target=w, name="thread:{}".format(idx))

for idx, w in enumerate(self.workers)

]

for t in self.threads:

t.start()

for t in self.threads:

t.join()

def get\_last\_node(self):

with self.nodes\_lock:

if self.nodes:

return self.nodes[-1]

else:

self.\_is\_end = True

return None

def add\_node(self, node):

with self.nodes\_lock:

self.nodes.append(node)

def remove\_node(self, node):

with self.nodes\_lock:

if node in self.nodes:

self.nodes.remove(node)

def is\_end(self):

return self.\_is\_end

def check\_end(self):

with self.nodes\_lock:

self.\_is\_end = len(self.nodes) == 0

def add\_result(self, node):

with self.results\_lock:

self.results.add(node)

def get\_bounds(self):

with self.bounds\_lock:

return deepcopy(self.bounds)

def main():

s = Searcher()

s.run()

print(len(s.results))

assert len(s.results) == 3 \*\* 10

if \_\_name\_\_ == '\_\_main\_\_':

main()