МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ

ФЕДЕРАЦИИ федеральное государственное автономное образовательное учреждение высшего образования «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ АЭРОКОСМИЧЕСКОГО ПРИБОРОСТРОЕНИЯ» КАФЕДРА № 43

КАФЕДРА КОМПЬЮТЕРНЫХ ТЕХНОЛОГИЙ И ПРОГРАММНОЙ ИНЖЕНЕРИИ

КУРСОВОЙ ПРОЕКТ ЗАЩИЩЕН С ОЦЕНКОЙ РУКОВОДИТЕЛЬ

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Доц., к.т.н |  |  |  | В.А. Матьяш |
| я |  | подпись, дата |  | инициалы, фамилия |

|  |
| --- |
| ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К КУРСОВОМУ ПРОЕКТУ  Разработка программы |
| **«Регистрация постояльцев в гостинице»** |
| по курсу: Структуры и алгоритмы обработки данных |

РАБОТУ ВЫПОЛНИЛ

СТУДЕНТ ГР. № 4134к Столяров Н.С.

подпись, дата инициалы, фамилия

Санкт-Петербург 2022

1. **Содержание**
2. **Задание на курсовой проект**

* Номер студенческого билета: 2021/3026
* Предметная область: 2 Регистрация постояльцев в гостинице
* Метод хеширования: 2 Закрытое хеширование с квадратичным опробованием
* Метод сортировки: 5 Слиянием
* Вид списка: 1 Линейный двунаправленный
* Метод обхода дерева: 2 Прямой
* Алгоритм поиска слова в тексте: 0 Боуера и Мура (БМ)

1. **Введение**Информационная система для предметной области «Регистрация постояльцев в гостинице» должна осуществлять следующие операции:
   * регистрацию/удаление постояльца
   * Добавление/удаление информации о номере
   * Регистрация вселения/выселения постояльца
   * Поиск постояльца по паспорту и ФИО
   * Поиск номера по номеру и оборудованию
2. **Алгоритмы и структуры данных**Для реализации хранения данных было использовано две структуры: Хэш-таблица и АВЛ дерево. Их основные функции представлены ниже.
3. **Описание программы**

*Хеш-таблица*

class HashTable:

def \_\_init\_\_(self, template\_key="NNNN-NNNNNN", count\_sigments=2000):

self.template\_key = template\_key # формат ключа (N - число, C - символ)

self.count\_sigments = count\_sigments

self.hash\_dict = {}

def generate\_hash(self, key):

out = 1

for el in key:

out += ord(el) \*\* 2

return int(out % self.count\_sigments)

def add(self, roomer):

hash = self.generate\_hash(roomer.pasport)

count\_collision = 1

while True:

if hash in self.hash\_dict:

hash += count\_collision

count\_collision \*= 2

else:

self.hash\_dict[hash] = roomer

break

def get\_by\_hash(self, hash):

return self.hash\_dict[hash]

def remove(self, hash):

return self.hash\_dict.pop(hash)

def get\_by\_hash(self, hash):

return self.hash\_dict[hash]

def clear\_table(self):

self.hash\_dict = {}

def print\_table(self):

for el in self.hash\_dict:

print("%d\t" % el, end="")

self.hash\_dict[el].print()

# поиск по заданию

def find\_by\_pasport(self, pasport):

for hash in self.hash\_dict:

if pasport == self.hash\_dict[hash].pasport:

return hash, self.hash\_dict[hash]

return None, None

def find\_by\_name(self, name):

for hash in self.hash\_dict:

if name == self.hash\_dict[hash].name:

return hash, self.hash\_dict[hash]

return None, None

*Список*

class Element\_list:

def \_\_init\_\_(self, item, head=None):

self.item = item

self.next = None

self.previous = None

self.head = (self if (head == None) else head)

def add(self, item):

next = self.head

while (next.next != None):

next = next.next

next.next = Element\_list(item, self.head)

next.next.previous = next.next

return next.next

def get\_item(self):

return self.item

def get\_head(self):

return self.head

def get\_all(self):

result = []

head = self.head

while (head != None):

result.append(head.item)

head = head.next

return result

def get\_current\_item(self, id):

head = self.head

for \_ in range(id):

head = head.next

return head

def delete\_element(self, id):

if (id < 0 or id > (self.get\_length() - 1)):

return False

if id == 0:

self.head = self.head.next

return True

elem\_curr = self.head

elem\_prev = None

elem\_next = None

for i in range(id):

elem\_prev = elem\_curr

elem\_curr = elem\_curr.next

elem\_next = elem\_curr.next

elem\_prev.next = elem\_next

elem\_next.previous = elem\_prev

return True

def get\_length(self):

head = self.head

count = 0

while (head != None):

head = head.next

count += 1

return count

################################################################################

def search\_by\_pasport(self, pasport):

head = self.head

count = 0

while (head != None):

if self.item.pasport == pasport:

return count

head = head.next

count += 1

return None

def search\_by\_number(self, number):

head = self.head

result = []

while (head != None):

if head.item.number == number:

result.append(head.item)

head = head.next

return result

*АВЛ дерево*

class AVLTree:

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

self.root = None

self.client = None

self.left = None

self.right = None

self.height = 1

def clear(self):

self.root = None

self.client = None

self.left = None

self.right = None

def insert(self, node, client):

if node == None:

node = AVLTree()

node.client = None

if node.client == None:

node.client = client

return node

elif client.number < node.client.number:

node.left = self.insert(node.left, client)

else:

node.right = self.insert(node.right, client)

node.height = 1 + max(self.get\_height(node.left), self.get\_height(node.right))

###

# balance = self.get\_balance(node)

#

# if balance > 1 and client.number < node.left.client.number:

# return self.right\_rotate(node)

#

# if balance < -1 and client.number > node.right.client.number:

# return self.left\_rotate(node)

#

# if balance > 1 and client.number > node.left.client.number:

# node.left = self.left\_rotate(node.left)

# return self.right\_rotate(node)

#

# if balance < -1 and client.number < node.right.client.number:

# node.right = self.right\_rotate(node.right)

# return self.left\_rotate(node)

###

return node

def left\_rotate(self, node):

right = node.right

left = right.left

right.left = node

node.right = left

node.height = 1 + max(self.get\_height(node.left), self.get\_height(node.right))

right.height = 1 + max(self.get\_height(right.left), self.get\_height(right.right))

return right

def right\_rotate(self, node):

left = node.left

right = left.right

left.right = node

node.left = right

node.height = 1 + max(self.get\_height(node.left), self.get\_height(node.right))

left.height = 1 + max(self.get\_height(left.left), self.get\_height(left.right))

return left

def get\_height(self, node):

if node is None:

return 0

return node.height

def get\_balance(self, node):

if node is None:

return 0

return self.get\_height(node.left) - self.get\_height(node.right)

def search(self, node, number):

if node is None:

return None

if number == node.client.number:

return node.client

if number < node.client.number:

return self.search(node.left, number)

return self.search(node.right, number)

def pre\_order(self, node):

if (node):

print("-" \* 20)

if node.client != None:

node.client.print()

self.pre\_order(node.left)

self.pre\_order(node.right)

def pre\_order\_return(self, node):

out = []

if (node):

if node.client != None:

out.append(node.client)

out += self.pre\_order\_return(node.left)

out += self.pre\_order\_return(node.right)

return out

################################################################################

# поиск по заданию

def search\_by\_number(self, number):

current = self

while True:

if current == None:

return None

if current.client.number == number:

return current.client

elif number < current.client.number:

current = current.left

else:

current = current.right

def search\_by\_equipment(self, equipment, current=None):

if current == None:

current = self

result = []

if boyerMurSearch(equipment, current.client.equipment):

result.append(current.client)

if current.left != None:

result += self.search\_by\_equipment(equipment, current.left)

if current.right != None:

result += self.search\_by\_equipment(equipment, current.right)

return result

def delete\_by\_number(self, number):

current = self

go\_left = False

prev = None

while True:

if current == None:

return False

print(current.client.number)

if current.client.number == number:

if prev != None:

if go\_left:

prev.left = None

else:

prev.right = None

else:

prev = self

left = current.left

right = current.right

if left != None:

self.insert(prev, left.client)

if right != None:

self.insert(prev, right.client)

return True

else:

prev = current

if number < current.client.number:

current = current.left

go\_left = True

else:

current = current.right

go\_left = False

*В приложении были реализованы основные функции, которые представлены в введении.*

1. **Тестирование программы**
2. **Заключение**В процессе выполнения курсового проекта мною была разработана информационная система – «Регистрация постояльцев в гостинице». При написании данного кода я применил знания, полученные на дисциплине «Структуры и алгоритмы обработки данных», а также закрепила их.
3. **Список литературы**