Дедлайн: 02.10.2023 22:00

Инструкция

Задания в рамках домашней работы подразделяются на два блока:

1. Блок А «*Аналитические задачи*» - решения задач A1-A5 оформляются в письменном в виде в любом удобном формате (TeX, скан и др.).
2. Блок Р «*Задачи на разработку*» - решения задач P1-P2 загружаются в систему GitLab и проходят автоматизированное тестирование.

Блок А					**Блок Р**			**Итого**
A1	**A2***	**A3**	**A4**	**A5**	**P1**	**P2**	**P3**	
4	6	4	7	5	15	10	10	55 (61*)

Задачи, помеченные *, не являются обязательными для решения (относятся к *бонусным*). Подтверждение представленных решений бонусных заданий обязательно сопровождается *устной защитой.*

Плагиат влечет за собой обнуление результатов у всех вовлеченных лиц.

Удачи!

Блок A. Разработка, анализ корректности и временной сложности алгоритмов

Задание A1 (4 балла) Анализ корректности алгоритма SELECTION SORT

Дана следующая итерационная реализация алгоритма сортировки выбором заданного целочисленного массива *A*, количество элементов в котором обозначено *n*:

```
selectionSort(A)
1   for i = 0 to n – 1
2       minId = i
3       for j = i + 1 to n
4           if A[j] < A[minId]
5               minId = j
6       swap(A[minId], A[i])
```

1. (1 балл) Сформулировать условие P_1, которое подходит в качестве инварианта внутреннего цикла алгоритма по **j**. Представить краткое обоснование (например, с использованием частичной трассировки выполнения цикла).
2. (1 балл) Сформулировать условие P_2, которое подходит в качестве инварианта внешнего цикла алгоритма по **i**. Представить краткое обоснование.
3. (2 балла) Выполнить проверку выполнения найденных инвариантов P_1 и P_2 до входа в каждый из циклов (INIT), в каждой итерации циклов (CONT), при выходе из цикла (EXIT).

Задание A2* (6 баллов) Анализ корректности алгоритма FAST EXPONENT

Дана итерационная реализация алгоритма возведения некоторого положительного числа x в степень n, где операция `mod` соответствует вычислению остатка, а операция `div` – целочисленному делению:

```
fastExponent(x, n)
1   r = 1
2   p = x
3   e = n
4   while e > 0
5       if e mod 2 <> 0
6           r = r * p
7       p = p * p
8       e = e div 2
```

1. (2 балла) Какое *точное* количество операций умножения требуется выполнить, чтобы вычислить x^n с помощью алгоритма FAST EXPONENT? Всегда ли данный алгоритм лучше наивного способа вычисления, требующего в точности n умножений?
2. (4 балла) Сформулировать условие P, которое подходит в качестве инварианта цикла `while`. Представить достаточное обоснование выбора инварианта. Выполнить проверку выполнения найденного инварианта P до входа в цикл (INIT), в каждой итерации цикла (CONT), а также при выходе из цикла (EXIT).

Задание А3 (4 балла) Точная функция временной сложности $T(n)$ и порядок ее роста

Дан следующий фрагмент программного кода на языке C++, где n – это некоторая заранее заданная положительная целочисленная константа:

```
int x = 100;
int y = 0;
for (size_t r = 1; r <= n; r = 2 * r) {
    x = x + r;
    for (size_t c = 2; c < n; c++) {
        if (x > y / c)
            y = y + r / c;
        else
            y = y - 1;
```

1. (2 балла) Составьте *точное* выражение для функции временной сложности $T(n)$ с учетом того, что арифметическая операция, присваивание и сравнение считаются *одной элементарной операцией (каждая)*. В ответе представьте ход вычислений. При составлении выражения для $T(n)$ обратите особое внимание на условный оператор.
2. (2 балла) Найдите функцию $f(n)$, для которой справедливо $T(n) = \Theta\big(f(n)\big)$. Обоснуйте свой ответ в соответствии с определением Θ-нотации.

Задание А4 (7 баллов) Анализ разных алгоритмов для решения одной (?) задачи

Даны три алгоритма, в рамках которых выполняется обработка целочисленного массива A, содержащего n элементов, а **sort**(A) соответствует сортировке массива некоторым способом:

```
algorithm1(A)
  c = 0
  ind = -1
  for i = 0 to n
      c1 = 0
      for j = 0 to n
          if A[i] = A[j]
              c1 = c1 + 1
      if c1 > c
          c = c1
          ind = i
  if c > n / 2
      return A[ind]
```

```
algorithm2(A)
  c = 1
  ind = 0
  for i = 1 to n
      if A[ind] = A[i]
          c = c + 1
      else
          c = c - 1
      if c = 0
          ind = i
          c = 1

  return A[ind]
```

```
algorithm3(A)
  if n = 1
      return A[0]
  c = 1
  sort(A)
  for i = 1 to n
      if A[i - 1] = A[i]
          c = c + 1
      else
          if c > n / 2
              return A[i - 1]
          c = 1
```

1. (2 балла) Утверждается, что представленные алгоритмы должны решать одну и ту же задачу, т.е. выдавать один и тот же ответ на одинаковых входных данных.
    a. Согласны ли вы с этим утверждением? Результаты работы каких алгоритмов из представленных могут отличаться? Какую задачу решает каждый алгоритм?

   b. Приведите примеры входных данных, при которых результаты работы алгоритмов могут отличаться, а также совпадать. Поясните свой ответ с помощью трассировки (частичной) работы алгоритмов.
2. (3 балла) Вычислите асимптотическую верхнюю границу $O(f(n))$ временной сложности для каждого алгоритма. Обоснуйте свой ответ. Представлять полный расчет точного выражения функции временной сложности $T(n)$ не нужно.
3. (1 балл) Какие алгоритмы и каким образом необходимо доработать, чтобы в результате все представленные алгоритмы решали одну и ту же задачу? В ответе представьте инструкции, которые необходимо добавить или удалить.
4. (1 балл) Докажите, что представленные вами доработки *не ухудшают* ранее вычисленные в п. 2 верхние границы временной сложности. Представьте расчеты верхних границ сложности доработанных алгоритмов.

<u>Задание A5 (5 баллов)</u> Поиск значения в отсортированной матрице за линейное время

Дана квадратная матрица A размера $N \times N$, заполненная целыми числами, а также целое число key. Матрица A характеризуется тем, что:

- значения в строках отсортированы по возрастанию, а
- значения в столбцах отсортированы по убыванию.

Предполагается, что заданная матрица заполнена уникальными значениям. Например, таким условиям отвечает следующая квадратная матрица:

$$\begin{pmatrix} 11 & 12 & 18 \\ 8 & 9 & 10 \\ 5 & 6 & 7 \end{pmatrix}$$

1. (3 балла) Разработайте линейный алгоритм поиска значения key в заданной матрице. Представьте описание алгоритма любым удобным способом – псевдокод, блок-схема, код на C++ и пр.
2. (2 балла) Выполните анализ временной сложности разработанного алгоритма, составив точное выражение функции временной сложности $T(n)$. Докажите, что $T(n) = O(n)$ в соответствии с определением асимптотической верхней границы.

Блок P. Реализация и обработка линейных контейнеров

Задание P1 (15 баллов) Двусвязный список

Требуется реализовать структуру данных – **двусвязный список `DoublyLinkedList`**. Тип элементов, которые будут храниться в списке, – **`int`**.

Публичный интерфейс реализуемого класса включает следующие конструкторы:

1. Конструктор по умолчанию для создания пустого двусвязного списка.
2. Конструктор копирования для создания одного списка на основе другого.
3. Конструктор на массиве, который принимает **`std::vector`** и на его основе создает двусвязный список.

Публичный интерфейс реализуемого класса включает следующие поля:

1. Указатель на голову списка **`head`**.
2. Указатель на хвост списка **`tail`**.

Публичный интерфейс реализуемого класса включает следующие методы вставки:

1. Метод **`pushBack()`** для вставки нового элемента в конец списка (за **`tail`**).
2. Метод **`pushFront()`** для вставки нового элемента в начало списка (перед **`head`**).
3. Метод **`insert`**`(`**`Node*`**` prev, `**`int`**` data)` для вставки нового элемента в некоторую позицию списка после **`prev`**.
   - Если позиция **`prev`** является некорректной (**`nullptr`** или не находится в данном списке), то необходимо выдать исключение **`std::runtime_error`**`(“Wrong position for insertion!”)`.

Публичный интерфейс реализуемого класса включает следующие методы удаления:

1. Метод **`popFront()`** для удаления элемента из начала списка.
   - Если удаление невозможно, то необходимо выдать исключение **`std::runtime_error`**`(“Deletion error!”)`.
2. Метод **`popBack()`** для удаления элемента из конца списка.
   - Если удаление невозможно, то необходимо выдать исключение **`std::runtime_error`**`(“Deletion error!”)`.
3. Метод **`pop`**`(`**`Node*`**` pos)` для удаления элемента, находящегося на позиции **`pos`**.
   - Если позиция **`pos`** является некорректной (**`nullptr`** или не находится в данном списке), то необходимо выдать исключение **`std::runtime_error`**`(“Wrong position for deletion!”)`.

Публичный интерфейс реализуемого класса включает следующие общие методы обработки контейнера:

1. Метод **`erase()`** для полного удаления списка и его содержимого.

2. Метод **`reverse()`** для обращения порядка следования элементов в списке. Например, в результате разворота списка **`1<->2<->3<->4<->5`** должен получиться список **`5<->4<->3<->2<->1`**. *При выполнении разворота списка не должно создаваться промежуточных списков, т.е. обработка выполняется «по месту».*
3. Метод **`removeDuplicates()`** для удаления дублирующихся элементов в списке. Например, после удаления дубликатов из списка **`1<->2<->1<->1<->3`** получим список **`1<->2<->3`**. Временная сложность удаления дубликатов должна быть ограничена $O(n)$, где n – это текущая длина списка. *Встроенные методы сортировки использовать не разрешается.*
4. Метод **`replace(int el, int new)`** для замены всех вхождений элемента **`el`** на **`new`**. Если элемент **`el`** не входит в данный список, то *он остается неизменным.*

Обратите внимание, что публичный интерфейс класса **`DoublyLinkedList`** *дополнительно расширять не разрешается.*

Задание P2 (10 баллов) Вращение массива с максимальным отличием

Вращением массива называется операция *циклического сдвига* элементов одномерного массива вправо (по часовой стрелке).

Расстоянием Хэмминга d_H между парой массивов назовем количество позиций, в которых элементы массивов отличаются друг от друга. Например, расстояние Хэмминга $d_H(A, B)$ между двумя массивами $A = [1, 3, 1, 4, 5]$ и $B = [1, 4, 5, 1, 3]$ составляет 4, поскольку значения их элементов различаются во всех позициях, кроме первой.

Нетрудно заметить, что количество возможных вращений массива длины n составляет n.

Требуется реализовать функцию `CountArrayRotations` для подсчета количества таких *уникальных* вращений заданного одномерного массива, что расстояние Хэмминга между исходным массивом и результатом его вращения *максимально.*

Вход	Выход
1 1 8 0	**1** *для входного массива существует одно вращение 8 0 1 1, для которого расстояние Хэмминга максимально и составляет 4.*
1 2 3 4	**3** *для входного массива существуют три вращения 4 1 2 3, 3 4 1 2 и 2 3 4 1, для которых расстояние Хэмминга с исходным массивом максимально и составляет 4.*

Задание P3 (10 баллов) Распаковка строки

Строка *s* называется *запакованной*, если ее построение описывается следующим правилом:

s = <цифра от 0 до 9>[подстрока_*s* и строка *t* произвольное число раз],
где t - другая запакованная строка.

При этом, подстрока_*s* не содержит цифр и квадратных скобок.

Требуется разработать функцию `Decode`, которая "распаковывает" исходную строку *s* по следующему правилу:

<цифра *x* от 0 до 9>[строка *s'*] => повторение строки *s'* *x* раз.

Примеры правильной распаковки строки:

Вход	**Выход**
3[b]	**bbb**
1[a2[bbb]]	**abbbbbb**