|  |
| --- |
| https://www.mirea.ru/bitrix/templates/unlimtech/images/logo.png |
| МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ |
| Федеральное государственное бюджетное образовательное учреждение высшего образования  **"МИРЭА - Российский технологический университет"**  **РТУ МИРЭА** |

|  |  |
| --- | --- |
| Отчет по выполнению практического задания №6  По дисциплине «Структуры и алгоритмы обработки данных» | |
| **Тема:**  Алгоритмические стратегии. Перебор и методы его сокращения. | |
|  | |
| Выполнил студент | Сидоров С.Д. |
| группа | ИКБО-20-21 |

Отчёт

1. **Постановка задачи:**

Разработать алгоритм решения задачи с применением метода указанного в варианте и реализовать программу.

1. Оценить количество переборов при решении задачи стратегией «в лоб» - грубой силы
2. Привести анализ снижения числа переборов при применении метода.
3. **Задание варианта:**

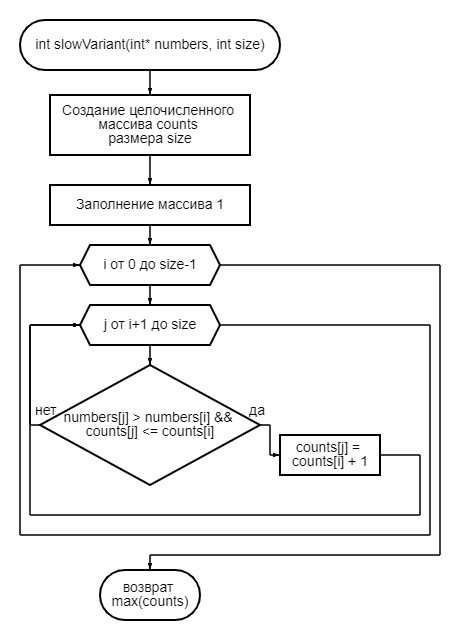
Вариант 2: Дана последовательность целых чисел. Необходимо найти её самую длинную строго возрастающую подпоследовательность.

Метод: Динамическое программирование.

1. **Подход к решению:**
2. Решение задачи полным перебором:
   1. Идея алгоритма:

Создание нового массива хранящего количество возможных подпоследовательностей для каждого элемента. Данный массив заполнен едениами так как каждый элемент уже является строго возврастающей последовательностью. После изначальный массив проходится с помощью двух вложенных циклов и для каждого элемента пройденного внутренним циклом проверяется два условия : внутренний элемент больше внешнего и подпоследовательность внешнего меньше или равна подпоследовательности внутреннего. Если условия выполнены, то в длину подпоследовательности внутреннего элемента записывается длина подпоследовательности внутреннего элемента плюс один.

* 1. Алгоритм:

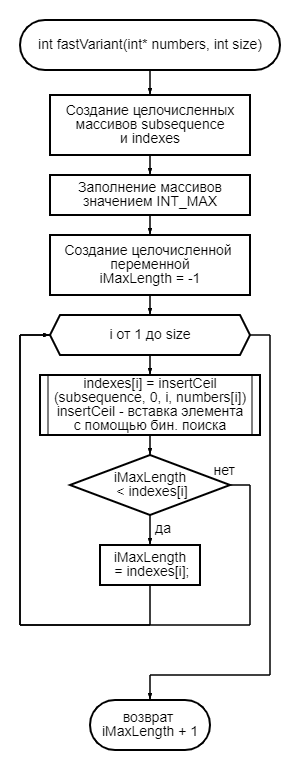


* 1. Реализация:

|  |
| --- |
| int slowVariant(int\* numbers, int size) {  int\* counts = new int[size];  for (int i = 0; i < size; i++) counts[i] = 1;  for (int i = 0; i < size - 1; i++) {  for (int j = i + 1; j < size; j++) {  if (numbers[j] > numbers[i] and counts[j] <= counts[i]) {  counts[j] = counts[i] + 1;  }  }  }  int result = 0;  for (int i = 0; i < size; i++)  if (counts[i] > result)  result = counts[i];  return result;  } |

* 1. Сложность алгоритма: O(n^2)

1. Решение задачи динамическим программированием:
   1. Идея алгоритма: создаются два массива: массив индексов и массив содержащий подпоследовательность. Оба массива изначально заполнены значением INT\_MAX. Нулевые элементы данных массивов заполнены нулем и нулевым элементом изначальной последовательности соответственно. Изначальный массив перебирается начианя с первого элемента. Для каждого элемента находится место в массиве подпоследовательностис помощью алгоритма бинарного поиска, так как подпоследовательность уже отсортирована. Индекс вставки записывается в массив индексов и если данный индекс больше предыдущей максимальной длины подпоследовательности то он заносится как максимальная длина подпоследовательности.
   2. Алгоритм:



* 1. Реализация:

|  |
| --- |
| int fastVariant(int\* numbers, int size) {  if (size <= 1) {  return 1;  }  int iMaxLength = -1;  int\* subsequence = new int[size];  int\* indexes = new int[size];  for (int i = 0; i < size; ++i) {  subsequence[i] = INT\_MAX;  indexes[i] = INT\_MAX;  }  subsequence[0] = numbers[0];  indexes[0] = 0;  for (int i = 1; i < size; ++i) {  indexes[i] = insertCeil(subsequence, 0, i, numbers[i]);  if (iMaxLength < indexes[i]) {  iMaxLength = indexes[i];  }  print(subsequence, size);  }  return iMaxLength + 1;  }  int insertCeil(int subsequence[], int startLeft, int startRight, int key) {  int mid = 0;  int left = startLeft;  int right = startRight;  int ceilIndex = 0;  bool ceilIndexFound = false;  for (mid = (left + right) / 2; left <= right && !ceilIndexFound; mid = (left + right) / 2) {  if (subsequence[mid] > key) {  right = mid - 1;  }  else if (subsequence[mid] == key) {  ceilIndex = mid;  ceilIndexFound = true;  }  else if (mid + 1 <= right && subsequence[mid + 1] >= key) {  subsequence[mid + 1] = key;  ceilIndex = mid + 1;  ceilIndexFound = true;  }  else {  left = mid + 1;  }  }  if (!ceilIndexFound) {  if (mid == left) {  subsequence[mid] = key;  ceilIndex = mid;  }  else {  subsequence[mid + 1] = key;  ceilIndex = mid + 1;  }  }  return ceilIndex;} |

* 1. Сложность алгоритма: O(n\*log(n))

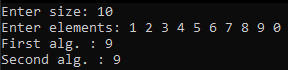
1. **Тестирование:**
   1. Тест 1 : 10 9 2 5 3 7 25 18



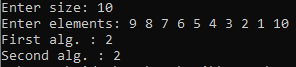
* 1. Тест 2 :



* 1. Тест 3 :



* 1. Тест 4:



1. **Сравнение алгоритмов:**

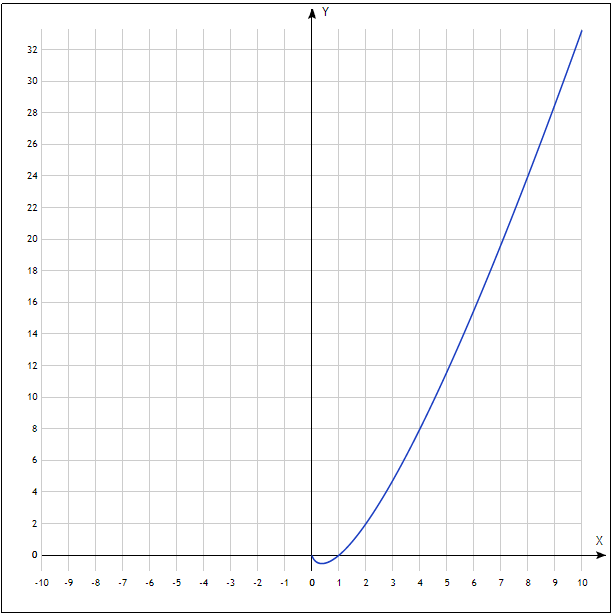
Алгоритм с применением динамического программирования ожидаемо оказался быстрее.

Для примера приведено примерное количество операций выполняемых каждым алгоритмом:

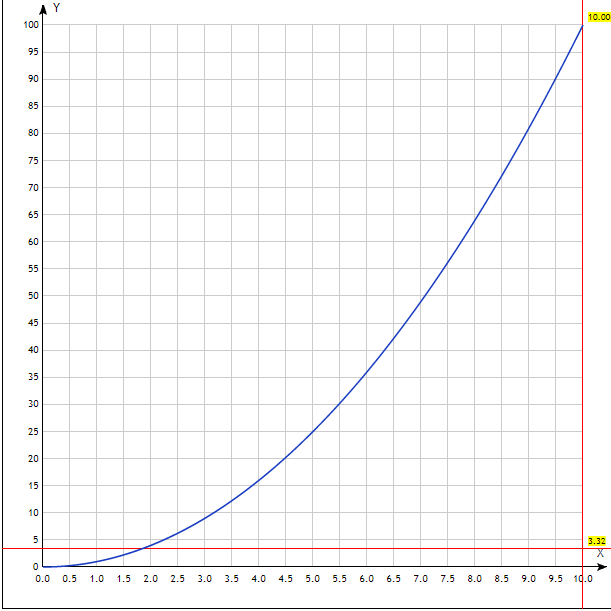
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Размер массива | 10 | 100 | 1000 | 10000 |
| Алг. 1 | 120 | 10 200 | 1 002 000 | 100 020 000 |
| Алг 2. | 40 | 800 | 11 000 | 140 000 |

Графики: Ось X - кол-во элементов в массиве, Ось Y -кол-во операций

Алгоритм.2



Алгоритм 1.



Выводы:

В ходе выполнения данной практической работы были получены навыки разработки и программной реализации задач с применением метода сокращеня числа переборов, также на практике было выявлено, как применение таких методов влияет на количество производимых операций.

**Список литературы:**

⦁ Лекции по структурам и алгоритмам обработки данных Рысин М.Л.

⦁ Методическое пособие по выполнению задания 1(битовые операции)