****

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 №2.7

**Тема:** Алгоритмические стратегии. Разработка и программная реализация задач с применением метода сокращения числа переборов.

Дисциплина: **«**СТРУКТУРЫ И АЛГОРИТМЫ ОБРАБОТКИ ДАННЫХ**»**

Выполнил студент Кузнецов А. А.

группа ИКБО-01-21

**Москва 2022**

**Цель работы:** Получить знания и навыки работы с стратегиями алгоритмического программирования. Разработать алгоритм и написать программу для решения поставленной в рамках варианта задачи и протестировать её на примерах. Оценить эффективность по отношению к методу «в лоб».

**Индивидуальный вариант:**

|  |  |  |
| --- | --- | --- |
| **Вариант** | **Задача** | **Метод** |
| 2 | Дана последовательность целых чисел. Необходимо найти ее самую длинную строго возрастающую подпоследовательность. | Динамическое программирование |

**Модель решения**

Алгоритм находит не только длину максимальной возрастающей подпоследовательности, но и саму подпоследовательность.

Двоичный поиск поможет нам в заполнении массива подпоследовательности

С помощью алгоритма мы будем искать место для нового элемента в вспомогательном массиве, в котором мы храним для каждой длины подпоследовательности минимальный элемент, на котором она может заканчиваться.

Если элемент больше максимального элемента в массиве, добавляем элемент в конец.

Если такой элемент уже существует в массиве, ничего особо не меняется. Это тоже просто.

Нужно рассмотреть случай, когда следующий элемент меньше максимального в этом массиве. Мы не можем его поставить в конец, и он не обязательно вообще должен являться членом именно максимальной последовательности, или наоборот, та подпоследовательность, которую мы имеем сейчас и в которую не входит этот новый элемент, может быть не максимальной.

В основном происходит 2 случая:

1. Рассматриваемый элемент последовательности (x) меньше чем наибольший элемент в массиве (Nmax), но больше чем предпоследний.
2. Рассматриваемый элемент меньше какого-то элемента в середине массива.

**Случай 1**: мы просто можем откинуть Nmax в массиве и поставим на его место x. Так как понятно, что если бы последующие элементы были бы больше, чем Nmax, то они будут и больше, чем x — соответственно мы не потеряем ни одного элемента.

**Случай 2**: для того, чтобы этот случай был нам полезен, мы заведем еще один массив, в котором будем хранить размер подпоследовательности, в которой этот элемент является максимальным. Собственно этим размером и будет являться та позиция в первом вспомогательном массиве для этого элемента, которую мы найдем с помощью двоичного поиска. Когда мы найдем нужную позицию, мы проверим элемент справа от него и заменим на текущий, если текущий меньше.

Итак, необходимы:

1. Исходная последовательность
2. Создаем мутабельный массив, где будем хранить возрастающие элементы для подпоследовательности
3. Создаем мутабельный массив размеров подпоследовательности, в которой рассматриваемый элемент является максимальным.

**Сравнение метода динамического программирования с методом «в лоб»**

Сложность метода грубой силы: О(n^2)

Сложность метода динамического программирования: O(n\*log n)

**Код программы**

|  |
| --- |
| #include <iostream> #include <vector> #define MAX\_INT 9999 using namespace std;  int linear\_itters\_counter = 0; int dynamic\_itters\_counter = 0;  int ceilIndex(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) {  dynamic\_itters\_counter++;   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; }  void makingResultSubsequence(vector<int> numbers, int subsequence[], int indexes[]) {   int subsequence\_index = -1;  for (int i = 0; i < numbers.size(); ++i) {  if (subsequence[i]!=MAX\_INT)  subsequence\_index++;  else  break;  }   bool flag = false;  vector<int> result\_subsequence;  for (int i = numbers.size() - 1; i >= 0; i--) {   if (flag) {  if ((indexes[i] == subsequence\_index) &&  (numbers.at(i) < result\_subsequence.back()) ){  result\_subsequence.push\_back(numbers.at(i));  subsequence\_index--;  }  } else { // первый раз  if (indexes[i] == subsequence\_index) {  result\_subsequence.push\_back(numbers.at(i));  subsequence\_index--;  flag = true;  }  }  if (subsequence\_index == -1)  break;  }   cout << "The found increasing sequence with DYNAMIC operations: ";  for (int i = result\_subsequence.size() - 1; i >= 0; i--) {  cout << result\_subsequence.at(i) << " ";  }  cout << endl;  }  int longestIncreasingSubsequenceLength(vector<int> numbers) {   if (numbers.size() <= 1) {  return 1;  }   int lis\_length = -1;   int subsequence[numbers.size()];  int indexes[numbers.size()];   for (int i = 0; i < numbers.size(); ++i) {  subsequence[i] = MAX\_INT;  }   subsequence[0] = numbers.at(0);  indexes[0] = 0;   for (int i = 1; i < numbers.size(); ++i) {  indexes[i] = ceilIndex(subsequence, 0, i, numbers.at(i));   if (lis\_length < indexes[i]) {  lis\_length = indexes[i];  }  }   makingResultSubsequence(numbers, subsequence, indexes);   return lis\_length + 1; }  int linear\_LISL(vector<int> numbers) {  int n = numbers.size();  vector<int> ind(n, 0);  vector<int> d(n, 0);  int last\_index = 0; // индекс последнего элемента подпоследовательности  d[0] = 1;  ind[0] = -1;  int len = 0; // общая длина максимальной подпоследовательности  for (int i = 1; i < n; ++i) {  int maxlen = 1;  ind[i] = -1;  for (int j = 0; j < i; ++j) {  linear\_itters\_counter++;   if ((d[j] >= maxlen) && (numbers[j] < numbers[i])) { // подзадача: ищем максимальную длину подпоследовательности для элемента  maxlen = d[j]; // обновляем максимальное значнеие подпоследовательности  ind[i] = j; // пишем предыдущее значние подпоследовательности для данного элемента  }  }  d[i] = maxlen + 1; // пишем макс. длину подпоследовательности для данного элемента  if (d[i] > len) {  len = d[i]; // обновляем общее макс. значение  last\_index = i;  }  }  vector<int> result;  int index = last\_index; // индекс элемента последовательности, который мы вставляем  while (index != -1) { // пока последовательность не кончилась  result.push\_back(numbers[index]);  index = ind[index];  }   cout << "The found increasing sequence with LINEAR operations: ";  for (int i = result.size() - 1; i >= 0; i--) { // выводим результат  cout << result[i] << " ";  }  cout << endl;  return result.size(); }  int main() {  vector<int> numbers = **{**5, 10, 6, 12, 3, 24, 7, 8**}**;  int n\_linear, n\_dynamic, num;   /\*cout << "Enter N:";  cin >> n\_dynamic\_itters;   for (int i = 0; i < n\_dynamic\_itters; ++i) {  cout << "Enter a " << i << " num:";  cin >> num;  numbers.push\_back(num);  }\*/   n\_linear = linear\_LISL(numbers);  n\_dynamic = longestIncreasingSubsequenceLength(numbers);  cout << endl;   cout << "The maximum length of the ascending sequence with LINEAR: " << n\_linear << endl;  cout << "The maximum length of the ascending sequence with DYNAMIC: " << n\_dynamic << endl;  cout << endl;   cout << "Number of iterations for a linear method: " << linear\_itters\_counter << endl;  cout << "Number of iterations for a dynamic method: " << dynamic\_itters\_counter << endl; } |

**Результаты тестирования**

Проверим программу на примере последовательности: 5, 10, 6, 12, 3, 24, 7, 8.

Для данной последовательности, правильными строго возрастающими подпоследовательностями будут являться:

1. 5, 6, 12, 24;
2. 5, 6, 7, 8.

Программе будет необходимо найти хотя бы одну из них. Правильным ответом максимальной длины строго возрастающей подпоследовательности будут равен 4. Результат работы программы представлен на рисунке 1.

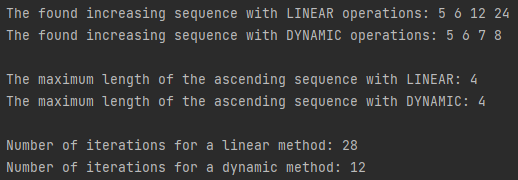


Рисунок 1. Результат тестирования программы

По результатам тестирования программа работает корректно.

**ВЫВОДЫ**

В результате выполнения данной работы мной был освоен один из методов алгоритмического программирования, а именно динамическое программирование. Была написана программа для решения классической задачи о ранце, она была протестирована на примере. Проведена оценка сложности алгоритма и выполнено сравнение эффективности по отношению к алгоритму грубой силы. Таким образом, мной был получен практический опыт использования алгоритмов динамического программирования