Министерство образования и науки Российской Федерации

САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

Факультет программной инженерии и компьютерной техники
Направление подготовки 09.03.04 Программная инженерия

Дисциплина «Алгоритмы и структуры данных»

ОТЧЁТ

по лабораторной работе Динамическое программирование (Stepic)

Студенка Кузенкова Елизавета группы P3217

Преподаватель Муромцев Дмитрий Ильич

Санкт-Петербург

2019 г.

Содержание

Задача 1: наибольшая последовательнократная подпоследовательность

Дано целое число $1 \le n \le 10^3$ и массив $A[1...n]$ натуральных чисел, не превосходящих $2 \cdot 10^9$. Выведите максимальное $1 \le k \le n$, для которого найдётся подпоследовательность $1 \le i_1 < i_2 < ... < i_k \le n$ длины k, в которой каждый элемент делится на предыдущий (формально: для всех $1 \le j < k, A[i_j] | A[i_{j+1}]$).

Sample Input:

```
4
3 6 7 12
```

Sample Output:

```
3
```

Исходный код к задаче 1

```
#include <iostream>
#include <vector>
#include <algorithm>


int main()
{
	int num = 0;
	std::cin >> num;
	std::vector<int> arr;
	while (--num >= 0) {
		int val = 0;
		std::cin >> val;
		arr.push_back(val);
	}
	auto size = arr.size();
	std::vector<int> path_len(size);
	for (int i = 0; i < size; ++i) {
		path_len[i] = 1;
		for (int j = 0; j < i; ++j) {
			if ((arr[i] % arr[j] == 0) && (path_len[j] + 1 > path_len[i])) {
				path_len[i] = path_len[j] + 1;
			}
		}
	}
	auto res = std::max_element(path_len.begin(), path_len.end(), [](int e1, int e2)
{return e1 < e2;});
	std::cout << *res << std::endl;

	return 0;
}
```

Задача 2: наибольшая невозрастающая подпоследовательность

Дано целое число $1 \le n \le 10^5$ и массив $A[1...n]$, содержащий неотрицательные целые числа, не превосходящие 10^9. Найдите наибольшую невозрастающую подпоследовательность в A. В первой строке выведите её длину k, во второй — её индексы $1 \le i_1 < i_2 < ... < i_k \le n$ (таким образом, $A[i_1] \ge A[i_2] \ge ... \ge A[i_n]$

).

Sample Input:

```
5
5 3 4 4 2
```

Sample Output:

```
4
1 3 4 5
```

Исходный код к задаче 2

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <sstream>

int main()
{
	int num = 0;
	std::cin >> num;
	std::vector<int> arr;
	while (--num >= 0) {
		int val = 0;
		std::cin >> val;
		arr.push_back(val);
	}
	auto size = arr.size();
	std::vector<int> path_len(size + 1);
	std::vector<int> pos(size + 1);
	for (size_t i = 0; i < size + 1; ++i) pos[i] = -1;
	path_len[0] = std::numeric_limits<int>::max();
	for (size_t i = 1; i < size + 1; ++i) path_len[i] = std::numeric_limits<int>::min();

	for (size_t i = 0; i < size; ++i) {
		size_t j = static_cast<size_t>(std::upper_bound(path_len.begin(),
path_len.end(), arr[i],
			[](int e1, int e2) { return e1 > e2;}) - path_len.begin());
		if ((path_len[j] < arr[i]) && (path_len[j - 1] >= arr[i])) {
			path_len[j] = arr[i];
			pos[j] = i;
		}
		else if (j == 0) {
			path_len[1] = arr[0];
			pos[1] = 0;
		}
	}
	auto res = std::count_if(path_len.begin(), path_len.end(), [](int e1) {return e1 >
std::numeric_limits<int>::min() && e1 < std::numeric_limits<int>::max();});
	std::cout << res << std::endl;
	std::ostringstream oss;
	for (int i = 0; i < size + 1; ++i) {
		if (pos[i] >= 0) {
			oss << pos[i] + 1 << " ";
		}
	}
	std::cout << oss.str() << std::endl;

	return 0;
}
```

Задача 3: расстояние редактирования

Вычислите расстояние редактирования двух данных непустых строк длины не более 10^2, содержащих строчные буквы латинского алфавита.

Sample Input 1:

```
ab
ab
```

Sample Output 1:

```
0
```

Sample Input 2:

```
short
ports
```

Sample Output 2:

```
3
```

Исходный код к задаче 3

```
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>


int main()
{
	// Считываем первую строку
	std::string strA;
	std::cin >> strA;
	// Считываем вторую строку
	std::string strB;
	std::cin >> strB;
	// Определяем размер строк
	auto sizeA = strA.size();
	auto sizeB = strB.size();
	// Создаем массивы для двух строк
	std::vector<std::vector<int>> tab;
	// Инициализируем данные
	for (auto i = 0; i <= sizeB; ++i) tab.push_back(std::vector<int>(sizeA + 1));
	for (auto i = 0; i <= sizeA; ++i) tab[0][i] = i;
	for (auto i = 1; i <= sizeB; ++i) tab[i][0] = i;
	// Считаем таблицу
	for (auto j = 1; j <= sizeB; ++j) {
		for (auto i = 1; i <= sizeA; ++i) {
			auto c = (strA.at(i - 1) == strB.at(j - 1)) ? 0 : 1;
			auto min_val = std::min({ tab[j][i - 1] + 1, tab[j - 1][i] + 1, tab[j - 
1][i - 1] + c });
			tab[j][i] = min_val;
		}
	}
	std::cout << tab[sizeB][sizeA] << std::endl;
	return 0;
}
```

Задача 4: рюкзак

Первая строка входа содержит целые числа $1 \le W \le 10^4$ и $1 \le n \le 300$ — вместимость рюкзака и число золотых слитков. Следующая строка содержит n целых чисел $0 \le w_1, \ldots, w_n \le 10^5$, задающих веса слитков. Найдите максимальный вес золота, который можно унести в рюкзаке.

Sample Input:

```
10 3
```

```
1 4 8
```

Sample Output:

```
9
```

Исходный код к задаче 4

```
#include <iostream>
#include <vector>
#include <algorithm>


int main()
{
	//Считываем вместимость рюкзака
	size_t W = 0;
	std::cin >> W;
	// Считываем количество вещей
	size_t n = 0;
	std::cin >> n;
	// Считываем веса вещей
	std::vector<int> v;
	for (auto i = 0; i < n; ++i) {
		auto value = 0;
		std::cin >> value;
		v.push_back(value);
	}
	// Инициализируем таблицу значений
	std::vector<std::vector<int>> tab(W + 1, std::vector<int>(n + 1, 0));
	// Рассчитываем значения
	for (size_t j = 1; j <= n; ++j) {
		for (size_t w = 1; w <= W; ++w) {
			if (v[j - 1] > w) tab[w][j] = tab[w][j - 1];
			else tab[w][j] = std::max({ tab[w][j - 1], tab[w - v[j - 1]][j - 1] +
v[j - 1] });
		}
	}
	std::cout << tab[W][n] << std::endl;
	return 0;
}
```

Задача 5: лестница

Даны число $1 \le n \le 10^2$ ступенек лестницы и целые числа $-10^4 \le a_1, \ldots, a_n \le 10^4$, которыми помечены ступеньки. Найдите максимальную сумму, которую можно получить, идя по лестнице снизу вверх (от нулевой до n-й ступеньки), каждый раз поднимаясь на одну или две ступеньки.

Sample Input 1:

```
2
1 2
```

Sample Output 1:

```
3
```

Sample Input 2:

```
2
2 -1
```

Sample Output 2:

```
1
```

Sample Input 3:

```
3
-1 2 1
```

Sample Output 3:

```
3
```

Исходный код к задаче 5

```
#include <iostream>
#include <vector>
#include <algorithm>

int main()
{
	// Считываем число ступеней
	size_t count = 0;
	std::cin >> count;
	// Считываем значения ступеней
	std::vector<int> nominals;
	nominals.push_back(0);
	for (size_t i = 0; i < count; ++i) {
		int value = 0;
		std::cin >> value;
		nominals.push_back(value);
	}
	/*
	Будем искать максимальную сумму для каждой ступени. Обозначим через S[i] максимальную
сумму для ступеней от 0 до i.
	Выразим S[i] через ответы для меньших подзадач. Тогда S[i] = min{S[i-1] + S[i], S[i-2]
+ S[i]}. Первые два элемента
	равны 0 и nominals[1].
	*/
	// Инициализируем массив
	std::vector<int> S;
	S.push_back(0);           // 0-я ступень
	S.push_back(nominals[1]); // 1-я ступень
	for (size_t i = 2; i <= count; ++i) S.push_back(std::max({ S[i - 1] + nominals[i], S[i
- 2] + nominals[i] }));
	// Выводим значение в последней ступени
	std::cout << S[count] << std::endl;
	return 0;
}
```

Задача 6: калькулятор

У вас есть примитивный калькулятор, который умеет выполнять всего три операции с текущим числом x: заменить x на $2x$, $3x$ или $x+1$. По данному целому числу $1 \le n \le 10^5$ определите минимальное число операций k, необходимое, чтобы получить n из 1. Выведите k

и последовательность промежуточных чисел.

Sample Input 1:

```
1
```

Sample Output 1:

```
0
1
```

Sample Input 2:

```
5
```

Sample Output 2:

```
3
1 2 4 5
```

Sample Input 3:

```
96234
```

Sample Output 3:

```
14
1 3 9 10 11 22 66 198 594 1782 5346 16038 16039 32078 96234
```

Исходный код к задаче 6

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <string>
#include <sstream>

int main()
{
	// Считываем число N
	size_t N = 0;
	std::cin >> N;
	/*
	Будем искать минимальное количество операций для каждого числа. Обозначим S[i] - 
минимальное число операций
	для числа i. Выразим S[i] через ответы для меньших подзадач. S[i] = min{S[i/3] + 1, 
S[i/2] + 1, S[i-1] + 1}, причем
	деление выполняется в случае делимости i нацело.
	*/
	// Промежуточные результаты складываем в массив
	std::vector<size_t> S;
	// Массив для отслеживания промежуточных чисел
	std::vector<std::vector<size_t>> path(N + 1, std::vector<size_t>());
	S.push_back(0);
	S.push_back(0);		// Для i = 1 требуется 0 операций
	path[1].push_back(1); // Для i = 1 промежуточное значение и есть 1
	for (size_t i = 2; i <= N; ++i) {
		size_t x_div_3 = std::numeric_limits<int>::max();
		if ((i % 3) == 0) x_div_3 = S[i / 3] + 1;
		size_t x_div_2 = std::numeric_limits<int>::max();
		if ((i % 2) == 0) x_div_2 = S[i / 2] + 1;
		size_t min_val = std::min({ x_div_3, x_div_2, S[i - 1] + 1 });
		S.push_back(min_val);
		if (min_val == x_div_3) { path[i] = path[i / 3]; path[i].push_back(i); }
		if (min_val == x_div_2) { path[i] = path[i / 2]; path[i].push_back(i); }
		if (min_val == S[i - 1] + 1) { path[i] = path[i - 1]; path[i].push_back(i); }
	}
	std::cout << S[N] << std::endl;
	// Восстанавливаем решение
	for (auto elem : path[N]) std::cout << elem << " ";
	std::cout << std::endl;
	return 0;
}
```