Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №2

по «Алгоритмам и структурам данных»

Базовые задачи / Timus

Выполнил:

Студент группы P3234

Баянов Р.Д.

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2024

Яндекс контест

Задача E. Коровы в стойла

Ход решения:

Для решения этой задачи я использовал перебор дистанций между коровами в стойлах с помощью бинарного поиска. Изначально увидим, что при любом расстоянии x, выгоднее всего разместить 1-ую корову в первое стойло. Мы имеем функцию check_distance, которая определяет подходит ли новое расстояние x, нашему условию (то есть все коровы находятся друг от друга на расстоянии x и каждой корове нашлось своё стойло). И в зависимости от ответа, мы бинарным поиском идём по значениям от 1 до длины отрезка между крайними стойлами. И таким образом мы находим наибольшее подходящее x для решения задачи.

Оценка сложности решения:

Время: O(N*logN)

Память: O(N)

Код решения:

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

int check_distance(int x, int count_cows, std::vector<int> stalls) {
	int cows = 1;
	int last_cow = stalls[0];
	int flag = 0;
	for (int i = 1; i < stalls.size(); i++) {
		if (stalls[i] - last_cow >= x) {
			last_cow = stalls[i];
			cows++;
		}
		if (cows == count_cows) {
			flag = 1;
			break;
		}
```

```
        }
        return flag;
}


int main()
{
        int count_stalls;
        int count_cows;
        std::cin >> count_stalls;
        std::cin >> count_cows;
        std::vector<int> stalls(count_stalls);
        for (int i = 0; i < count_stalls; i++) {
                std::cin >> stalls[i];
        }
        int l = 0;
        int r = stalls[stalls.size() - 1];
        while (r - l > 1) {
                int middle = (l + r) / 2;
                if (check_distance(middle, count_cows, stalls)) {
                        l = middle;
                }
                else {
                        r = middle;
                }
        }
        std::cout << l;
        return 0;
}
```

Задача F. Число

Ход решения:

Для решения этой задачи я написал компаратор, который будет сравнивать строки цифр между собой. Мы с помощью конкатенации проверяем какое число больше: при варианте, когда первый отрывок стоит в меньших разрядах числа, а второй в больших или же наоборот. Затем пользуясь

нашим компаратором, сортируем список, в котором находятся наши строки из цифр. В конце проходимся по отсортированному списку и выводим каждую строчку. Таким образом, получаем число, являющееся правильным ответом нашей задачи.

Оценка сложности решения:

Время: O(N)

Память: O(N)

Код решения:

```
#include <iostream>
#include <algorithm>
#include <fstream>
#include <string>
#include <list>

bool comparator(std::string str1, std::string str2) {
	return str1 + str2 > str2 + str1;
}

int main()
{
	std::list<std::string> numbers;
	std::string str;
	while (std::cin >> str) {
		numbers.push_back(str);
	}
	numbers.sort(comparator);
	for (std::string num : numbers) {
		std::cout << num;
	}
	return 0;
}
```

Задача G. Кошмар в замке

Ход решения:

Для решения этой задачи я выстраиваю мапу со значениями веса для каждой буквы. Выясняем, что для получения самой большой по весу строки, мы должны разместить две одинаковые буквы с самым большим весом как можно дальше друг от друга (если, конечно же, есть буквы, которые в строке представлены не в единичном экземпляре). Буквы с весом поменьше можно разместить чуть ближе, с ещё меньшим весом – ещё ближе. Все остальные буквы (единичные или же 3, 4, 5 и т. д. буквы тех, которые мы уже расставили по бокам, можно поставить в любое место по центру между парными буквами, так как веса они в строку добавлять не будут. Отсортировав буквы по весу, раскидываем их либо в центр, либо по бокам. Таким образом, обнаружим одну из подходящих под правильный ответ строку и выведем её.

Оценка сложности решения:

Время: O(N)

Память: O(N)

Код решения:

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

int main()
{
	std::cin.tie(0);
	std::ios_base::sync_with_stdio(0);
	std::string str;
	std::cin >> str;
	int count_of_letters = 26;
	int size_str = str.size();
	std::vector<int> weights(count_of_letters);
	std::unordered_map<char, int> letters_with_weights;
	std::vector<std::pair<char, int>> letters_in_string_sorted;
	std::string result1 = "";
	std::string result2 = "";
	int weight;
	for (int i = 0; i < count_of_letters; i++) {
		std::cin >> weight;
```

```
                weights[i] = weight;
        }
        for (int i = 0; i < count_of_letters; i++) {
                letters_with_weights['a' + i] = weights[i];
        }
        for (int i = 0; i < size_str; i++) {
                letters_in_string_sorted.push_back({str[i],
letters_with_weights[str[i]]});
        }
        std::sort(letters_in_string_sorted.begin(), letters_in_string_sorted.end(), []
(const std::pair<char, int>& a, const std::pair<char, int>& b) {
                if (a.second == b.second) {
                        return a.first > b.first;
                }
                return a.second > b.second;
                });
        for (int i = 0; i < size_str - 1; i++) {
                if (letters_in_string_sorted[i].first == letters_in_string_sorted[i +
1].first && ((!result1.empty() && result1[result1.size() - 1] !=
letters_in_string_sorted[i].first) || result1.empty())) {
                        result1 = result1 + letters_in_string_sorted[i].first;
                        i++;
                }
                else {
                        result2 = result2 + letters_in_string_sorted[i].first;
                }
        }
        if (size_str > result2.size() + 2 * result1.size()) {
                result2.push_back(letters_in_string_sorted[size_str - 1].first);
        }
        std::cout << result1 + result2;
        for (int i = result1.size() - 1; i > -1; i--) {
                std::cout << result1[i];
        }
        return 0;
}
```

Задача H. Магазин

Ход решения:

Для решения этой задачи я сортирую вектор с ценами по убыванию. Тем самым, нетрудно догадаться, что при проходе вектора от большего к меньшему, каждый k-ый товар будет тем самым товаром, который нужно будет удалить, чтобы получить максимальную скидку. Мы будем обнулять цену в массиве на нужный нам товар. И в конце выведем сумму цен всех товаров. Вот и ответ.

Оценка сложности решения:

Время: O(N)

Память: O(N)

Код решения:

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

int main()
{
	int n;
	int k;
	std::cin >> n >> k;
	std::vector<int> prices(n);
	for (int i = 0; i < n; i++) {
		std::cin >> prices[i];
	}
	std::sort(prices.begin(), prices.end(), std::greater<>());
	for (int i = 1; i < n + 1; i++) {
		if (i % k == 0) {
			prices[i - 1] = 0;
		}
	}
	int sum = 0;
	for (int price : prices) {
		sum += price;
	}
	std::cout << sum;
	return 0;
}
```

Тимус

Задача 1444. Накормить элефпотама

Ход решения:

Для решения этой задачи я использовал немного знаний геометрии и сортировку. Идея такова: мы напишем компаратор, первый приоритет которого сортировка по углу, что образуется между осью OX и прямой проходящей через начало координат и координаты какой-либо тыквы. А второй приоритет по расстоянию от какой-либо тыквы до начала координат. Вспомним как находятся расстояния и как проверить что две точки на координатной плоскости лежат на одной прямой. Всё это поможет нам в компараторе. Если наша первая тыква не будет иметь значений 0 0, то мы путём вычитания из координат всех тыкв координаты первой тыквы, как бы сместим координатную плоскость к первой тыкве. И в конце для того, чтобы угадать с последовательностью, мы разобьём координаты всех тыкв на два вектора (те, что снизу и те, что сверху). Это поможет нам выбрать направление обхода такое, что элефпотам не будет пересекать свои же следы или своё начальное положение). Аккуратно выводим кол-во тыкв, первую тыкву и все тыквы из двух векторов, в том порядке в котором они были отсортированы нашим компаратором. Задача решена.

Оценка сложности решения:

Время: O(N)

Память: O(N)

Код решения:

```
#include <iostream>
#include <vector>
#include <algorithm>
#include <math.h>
#include <cmath>

struct pumpkin {
        int x, y;
```

```
        int i;
};

int comp_angle(struct pumpkin p1, struct pumpkin p2) {
        return p1.x * p2.y - p2.x * p1.y;
}

int distance(const struct pumpkin p) {
        return p.x * p.x + p.y * p.y;
}

bool comparator(const struct pumpkin p1, const struct pumpkin p2) {
        return comp_angle(p1, p2) > 0 || (comp_angle(p1, p2) == 0 && dis-
tance(p1) < distance(p2));
}
int main()
{
        int N;
        std::cin >> N;
        std::vector<struct pumpkin> pumpkins1;
        std::vector<struct pumpkin> pumpkins2;
        int x;
        int y;
        struct pumpkin first_pumpkin;
        struct pumpkin pk;
        int start_x;
        int start_y;
        std::cin >> start_x;
        std::cin >> start_y;
        first_pumpkin.i = 1;
        first_pumpkin.x = start_x;
        first_pumpkin.y = start_y;
        for (int i = 1; i < N; i++) {
                std::cin >> x;
                std::cin >> y;
                pk.i = i + 1;
                pk.x = x;
                pk.y = y;
                pk.x -= start_x;
                pk.y -= start_y;
                if (pk.y > 0 || (pk.y == 0 && pk.x > 0)) {
                        pumpkins1.push_back(pk);
                }
                else {
```

```
                pumpkins2.push_back(pk);
            }
        }
        std::sort(pumpkins1.begin(), pumpkins1.end(), comparator);
        std::sort(pumpkins2.begin(), pumpkins2.end(), comparator);
        std::cout << N << std::endl << first_pumpkin.i << std::endl;
        if ((pumpkins1.empty() || pumpkins2.empty()) ||
comp_angle(pumpkins1.back(), pumpkins2.front()) > 0) {
            for (int i = 0; i < pumpkins1.size(); i++) {
                std::cout << pumpkins1[i].i << std::endl;
            }
            for (int i = 0; i < pumpkins2.size(); i++) {
                std::cout << pumpkins2[i].i << std::endl;
            }
        }
        else {
            for (int i = 0; i < pumpkins2.size(); i++) {
                std::cout << pumpkins2[i].i << std::endl;
            }
            for (int i = 0; i < pumpkins1.size(); i++) {
                std::cout << pumpkins1[i].i << std::endl;
            }
        }
        return 0;
}
```

Задача 1726. Кто ходит в гости…

Ход решения:

Для решения этой задачи мы представим карту города. Каждый дом члена комитета нужно связать с другим домом. Разобьём всю карту на маленькие дорожки от дома к дому. Мы отсортируем координаты всех домов по возрастанию, для того чтобы расстояния, которые мы берём не получались с отрицательными значениями. Увидим, что по каждой дороге проходит I * (n – i) раз, и если ещё учесть то, что по этой дороге ходят туда и обратно, то умножим на 2. Суммирую все расстояния между собой, мы получим сумму. Нетрудно догадаться, что в матрице домов количество расстояний между

элементами этой матрицы n * (n – 1). Именно поэтому мы полученную сумму разделим на это число. Так как переменная sum типа long long, она сама по себе округлится до нижнего целого значения. Отсюда и получаем средний путь члена комитета.

Оценка сложности решения:

Время: O(N)

Память: O(N)

Код решения:

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

int main()
{
	long long n;
	std::cin >> n;
	std::vector<int> X;
	std::vector<int> Y;
	int x;
	int y;
	for (int i = 0; i < n; i++) {
		std::cin >> x;
		std::cin >> y;
		X.push_back(x);
		Y.push_back(y);
	}
	std::sort(X.begin(), X.end());
	std::sort(Y.begin(), Y.end());
	long long s = 0;
	for (long long i = 1; i < n; i++) {
		s += ((X[i] - X[i - 1]) + (Y[i] - Y[i - 1])) * 2 * i * (n - i);
	}
	std::cout << s / (n * (n - 1)) << std::endl;
	return 0;
}
```

Дополнительное задание к 1 блоку

Задача 1005. Куча камней (код Грея)

Ход решения:

Для решения этой задачи я использовал код Грея. Мы будем с помощью классической генерации кодов Грея получать наборы булевых значений длины size (1 – камень в одну кучу, 0 - камень в другую кучу). При получении каждого кода Грея мы будем прогонять цикл по этому набору нулей и единичек и сохранять полученную разницу разбиения камней на две кучи. И конечно же, сравнивать это значение с минимальным. Сам по себе Код Грея создаёт коды, в которых каждый следующий отличается от другого минимальным изменением (то есть отличается на одну цифру). Алгоритм генерации наборов такой, что мы создаём набор значений, который изначально будет представлять из себя набор нулей. После этого в зависимости от последнего значения этого набора мы будем идти от начала и менять каждую новую цифру и обрабатывать эти значения на разбиение суммы. Когда мы дойдём до последнего элемента нашего образующего набора, мы выходим из генерации кодов и сама программа завершается, предварительно показав нам минимально возможную разницу между кучами. К сожалению, использование алгоритма Грея не ускорило работу моего предыдущего решения, но изучить этот алгоритм было крайне полезно и интересно.

Оценка сложности решения:

Время: O(N * 2^N)

Память: O(N)

Код решения:

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

int code_gray(int n, std::vector<int> stones) {
    std::vector<int> a(n + 1, 0);
```

```
    int j = 0;
    int minDiff = INT_MAX;
    std::vector<int> result(n);
    while (true) {
        std::cout << std::endl;
        int sum0 = 0;
        int sum1 = 0;
        std::copy(a.begin(), a.end() - 1, result.begin());
        a[n] = 1 - a[n];
        if (a[n] == 1) {
            j = 0;
        }
        else {
            for (int i = 0; i < n; i++) {
                if (a[i] == 1) {
                    j = i + 1;
                    break;
                }
            }
        }
        for (int i = 0; i < n; i++) {
            std::cout << result[i];
            if (result[i] == 0) {
                sum0 += stones[i];
            }
            else {
                sum1 += stones[i];
            }
        }
        minDiff = std::min(minDiff, abs(sum0 - sum1));
        if (j == n) {
            break;
        }
        a[j] = 1 - a[j];
    }
    return minDiff;
}



int main() {
    int size;
    std::cin >> size;
    std::vector<int> stones(size);
```

```
    for (int i = 0; i < size; i++) {
        std::cin >> stones[i];
    }
    int answer = code_gray(size, stones);
    std::cout << answer;
    return 0;
}
```