Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ ИТМО»
(УНИВЕРСИТЕТ ИТМО)

Факультет «Систем управления и робототехники»

ОТЧЁТ
ПО НОМЕРАМ 2025, 1005, 1296

По дисциплине «Алгоритмы и структуры данных»

Студент:
Сухов Н.М. группа R3235

Проверил:
Тропченко А. А.

г. Санкт-Петербург
2025

2025

Цель

Максимизировать число столкновений между бойцами.

Задача

Бокс, каратэ, самбо... Классические боевые единоборства пресытили аудиторию. Поэтому известный спортивный канал запускает новый формат соревнований, основанный на традиционной русской забаве — боях стенка на стенку. В соревновании могут участвовать от двух до k команд, каждая из которых будет соперничать с остальными. Всего в соревновании примут участие n бойцов. Перед началом боя они должны разделиться на команды, каждый боец должен войти ровно в одну команду. За время боя два бойца сразятся, если они состоят в разных командах. Организаторы считают, что популярность соревнований будет тем выше, чем больше будет количество схваток между бойцами. Помогите распределить бойцов по командам так, чтобы максимизировать количество схваток между бойцами, и выведите это количество.

Описание программы

```cpp
#include <iostream>

int main(int argc, char const *argv[]) {
    int T;
    std::cin >> T;
    int n[T] {};
    int k[T] {};

    for (int i = 0; i < T; ++i)
        std::cin >> n[i] >> k[i];

    for (int i = 0, sum = 0; i < T; ++i, sum = 0) {
        int k_temp = k[i], n_temp = n[i], fighters_past = 0;
        int teams[k[i]] {};

        while (k_temp) {
            teams[k_temp - 1] = n_temp / k_temp;
            n_temp -= n_temp / k_temp--;
        }

        for (int j = 0; j < k[i] - 1; ++j) {
            fighters_past += teams[j];
            sum += teams[j] * (n[i] - fighters_past);
        }

        std::cout << sum << std::endl;
    }

    return 0;
}
```

1. Сначала алгоритм распределяет игроков по следующему правилу:

   Каждое оставшееся нераспределенное число участников `n_temp` целочисленно делится на оставшееся `k_temp` число команд. Таким образом, мы получаем максимально равномерное распределение бойцов по командам. Вот зачем это нужно:

   Для каждых 2-х команд число столкновений бойцов равно $x_1 \cdot x_2$, где x_i - число участников в i-той

команде. Сравним случай, когда $x_1 = x_2$ и когда $x_1 \neq x_2$

$$xx \vee (x-a)(x+a); \quad x^2 \vee x^2 - a^2; \quad 0 > -a^2$$

Здесь a - разница между числом бойцов в командах. Чем больше a, тем меньше боев мы увидим.

2. Во второй части алгоритма производится подсчет боев:

   Алгоритм проходит по каждой команде и умножает число её участников на число оставшихся бойцов в еще не пройденных командах. По сути, алгоритм представляет собой модифицированную версию формулы перестановок $p = n!$

Входные данные

```
3
17 5
8 4
540 49
```

Выходные данные

```
115
24
142824
```

Timus

ID	Дата	Автор	Задача	Язык	Результат проверки	№ теста	Время работы	Выделено памяти
10924086	02:42:03 2 апр 2025	Nikolai Sukhov	2025. Стенка на стенку	G++ 13.2 x64	Accepted		0.015	412 КБ

Выводы

Алгоритм оптимально справляется со своей задачей, и при том, занимает мало строк в описательном плане.

1005

Цель

Нахождение минимальной разности весов.

Задача

У вас есть несколько камней известного веса w1, ..., wn. Напишите программу, которая распределит камни в две кучи так, что разность весов этих двух куч будет минимальной.

Описание программы

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

int main() {
    int n;
    std::cin >> n;
    std::vector<int> weights(n);
    int total_sum = 0;
```

```
    for (int i = 0; i < n; ++i) {
        std::cin >> weights[i];
        total_sum += weights[i];
    }

    int target = total_sum / 2;
    std::vector<std::vector<bool>> d_table(n + 1, std::vector<bool>(target + 1, false));
    d_table[0][0] = true;

    for (int i = 1; i <= n; ++i) {
        for (int j = 0; j <= target; ++j) {
            d_table[i][j] = d_table[i - 1][j];
            if (j >= weights[i - 1]) {
                d_table[i][j] = d_table[i][j] || d_table[i - 1][j - weights[i - 1]];
            }
        }
    }

    int best_sum = 0;
    for (int i = target; i >= 0; --i) {
        if (d_table[n][i]) {
            best_sum = i;
            break;
        }
    }

    std::cout << total_sum - 2 * best_sum;

    return 0;
}
```

Сначала мы находим половину от суммы всех весов - это значение, к которому должен стремиться вес камней в каждой куче, чтобы и разница была минимальной.

Дальше мы составляем таблицу размера `(n+1)×(target+1)`, в которой `dp[i][j]` будет `true`, если среди первых i грузов существует подмножество с суммой весов j.

Таким образом, мы получаем все возможные суммы весов в первой куче (не превышающие половину от суммы), и находим более близкую к половине сумы, после чего находим разность и получаем ответ.

Входные данные

```
7
32 45 2 56 11 19 4
```

Выходные данные

```
3
```

Timus

ID	Дата	Автор	Задача	Язык	Результат проверки	№ теста	Время работы	Выделено памяти
10925241	21:18:51 2 апр 2025	Nikolai Sukhov	1005. Куча камней	G++ 13.2 x64	Accepted		0.078	2 372 КБ

Выводы

Задача является частным случаем задач о рюкзаке, и динамическое программирование является одним из наиболее оптимальных подходов к её решению.

1296

Цель

Вывести максимальный гравитационный потенциал, который может накопить корабль в альфа-фазе прыжка.

Задача

Ограничение времени: 1.0 секунды Ограничение памяти: 64 МБ Гиперпереход, открытый ещё в начале XXI-го века, и сейчас остаётся основным способом перемещения на расстояния до сотен тысяч парсеков. Но совсем недавно физиками открыто новое явление. Оказывается, длительностью альфа-фазы перехода можно легко управлять. Корабль, находящийся в альфа-фазе перехода, накапливает гравитационный потенциал. Чем больше накопленный гравитационный потенциал корабля, тем меньше энергии потребуется ему на прыжок сквозь пространство. Ваша цель — написать программу, которая позволит кораблю за счёт выбора времени начала альфа-фазы и её длительности накопить максимальный гравитационный потенциал.

В самой грубой модели грави-интенсивность — это последовательность целых чисел pi. Будем считать, что если альфа-фаза началась в момент i и закончилась в момент j, то накопленный в течение альфа-фазы потенциал — это сумма всех чисел, стоящих в последовательности на местах от i до j.

Описание программы

```
#include <iostream>

int main(int argc, char const *argv[])
{
    int N, best_sum = 0, current_sum = 0;
    std::cin >> N;
    int p[N] {};

    for (int i= 0; i < N; ++i)
        std::cin >> p[i];

    for (int i = 0; i < N; ++i) {
        current_sum = std::max(p[i], current_sum + p[i]);
        best_sum = std::max(best_sum, current_sum);
    }

    std::cout << best_sum;

    return 0;
}
```

Задача решена при помощи алгоритма Кадане:

Этот алгоритм проходит весь массив, поддерживая текущую сумму подмассива `current_sum` и максимальную найденную сумму `best_sum`, сравнивая их на каждом шаге. В конце получаем максимально возможную сумма подмассива.

Входные данные

```
8
78
-967
345
789
-5
-8768
```

```
568
98
```

Выходные данные

```
1134
```

Timus

ID	Дата	Автор	Задача	Язык	Результат проверки	№ теста	Время работы	Выделено памяти
10925639	06:40:33 3 апр 2025	Nikolai Sukhov	1296. Гиперпереход	G++ 13.2 x64	Accepted		0.093	624 КБ

Выводы

Алгоритм Кадане - отличный способ решить задачу поиска наибольшего подмасива за линейное время.