Университет ИТМО

Факультет программной инженерии и компьютерной техники

Лабораторная работа №1

по «Алгоритмам и структурам данных»

Выполнил:

Студент группы P3212

Куприянов А.А

Преподаватели:

Косяков М.С.

Тараканов Д.С.

Санкт-Петербург

2020

Task 2025

```c
/*******************************************************
 * Task 2025                                           *
 *                                                     *
 * TIMUS                                               *
 *                                                     *
 *******************************************************/

#include <stdio.h>

int main() {
  int iter, n, k;
  scanf("%d", &iter);

  for (int i = 0; i < iter; i++) {
    scanf("%d%d", &n, &k);
    int diff = n % k;
    int each = n / k;
    int a = (n - each) * (k - diff) * each;
    int b = (n - (each + 1)) * diff * (each + 1);

    int res = (a + b) / 2;
    printf("%d\n", res);
  }
}
```

Основной идеей данной задачи является понимание в каком случае будет максимальное количество боев. Если понять это (а это максимальное количество команд и максимально возможное равномерное распределение), то задача просто сводится к тому, чтобы найти распределение по командам.

В данном случае, мы сначала распределяем всех участников строго равномерно (в каждой команде равное количество участников), а оставшихся условно распределяем по этим командам. Таким образом, у нас получается два типа команд: с количеством участников n и n+1. Остается просто посчитать количество боев.

Task 1005

```
/*****************************************************
 * Task 1005                                         *
 *                                                   *
 * SOLVE FOR TIMUS                                   *
 *                                                   *
 *****************************************************/

#include <stdio.h>
#include <stdlib.h>

int get_min_from(int x, int y) { return y ^ ((x ^ y) & -(x < y)); }

int r_func(int iter, int spot1, int spot2, int values[]) {

  if (iter == 0)
    return abs(spot1 - spot2);
  else {
    int to_first_spot =
        r_func(iter - 1, spot1 + values[iter - 1], spot2, values);

    int to_second_spot =
        r_func(iter - 1, spot1, spot2 + values[iter - 1], values);

    return get_min_from(to_first_spot, to_second_spot);
  }
}

int main(void) {
  int iter;

  if (scanf("%d", &iter) != 1) {
    printf("Invalid input");
  }

  int values[iter];

  for (int i = 0; i < iter; i++) {
    if (scanf("%d", &values[i]) != 1) {
      printf("Invalid input: you provided wrong amount of variables");
    }
  }

  printf("%d", r_func(iter, 0, 0, values));
}
```

Здесь основная идея заключалась в том, чтобы сделать обход дерева рекурсивным алгоритмом. Это прекрасная задача, на мой взгляд. Потому что первые мысли, которые приходили на голову - это построить дерево и найти минимальный или считать все данные потом сделать аналитические вычисления. Тем не менее, оказалось, что, грубо говоря, эти два метода можно объединить в один.

Сама функция очень простая - мы просто высчитываем все возможные варианты (образуя некое дерево). Важно при этом в конце каждой функции возвращать абсолютное значение.

С каждого узла дерева мы берем минимальное значение и так далее до корневого узла, откуда нам остается только вернуть минимальное значение.

Task 1296

```c
/*****************************************************
 * Task 1296                                         *
 *                                                   *
 * TIMUS                                             *
 *                                                   *
 *****************************************************/

#include <stdio.h>

int main(void) {
  int iter;

  if (scanf("%d", &iter) != 1) {
    printf("Invalid input");
  }

  int max = 0;
  int sum = 0;
  for (int i = 0; i < iter; i++) {
    int inp;
    if (scanf("%d", &inp) != 1) {
      printf("Invalid input: you provided wrong amount of variables");
    }

    sum = sum + inp;
    if (sum > max)
      max = sum;
    if (sum < 0)
      sum = 0;
  }

  printf("%d", max);
}
```

Перебор всех вариантов (как бы он не оптимизирован) не является оптимальным решением и тратит много времени. С другой стороны, мы можем находить максимальную сумму последовательности прямо во время ввода.

Будем сразу считывать максимальную сумму последовательности, а если она станет отрицательной, то в следующей последовательности она будет только уменьшать максимальную сумму. Поэтому мы сбрасываем этот "балласт" (равняем 0). Таким образом, находим максимальную сумму последовательности, и в случае, если все числа меньше 0 - выводим 0, так как не взяли ни одного числа.