Факультет программной инженерии и компьютерной техники

Алгоритмы и структуры данных

Введение в алгоритмы

Преподаватели: Косяков Михаил Сергеевич, Тараканов Денис Сергеевич

Выполнил: Кульбако Артемий Юрьевич

P3212

Санкт-Петербург
2020

1005. Куча камней

$$T(n) = O(2^n)$$

```
#include <iostream>
using namespace std;

long search(long array[], int size, int i = 0, long sum1 = 0, long sum2 = 0)
{
    static long min_ = 100000;
    if (i == size) min_ = min(min_, abs(sum1 - sum2));
    else {
        search(array, size, i + 1, sum1 + array[i], sum2);
        search(array, size, i + 1, sum1, sum2 + array[i]);
    }
    return min_;
}

int main() {
    int n;
    cin >> n;
    long rocks[n];
    for (int i = 0; i < n; i++) cin >> rocks[i];
    cout << search(rocks, n) << endl;
    return 0;
}
```

Нам нужно минимизировать разницу веса между двумя кучами камней. Первое что приходит в голову - разделить исходную кучу на 2 и посчитать их суммы. Очевидно, что метод не даст правильного результата, так как в одной половине могут лежать исключительно тяжёлые, а в другой исключительно лёгкие камни. И всё же, подобная куча может сыграть нам на руку. Вспомним как работают весы: для равенства на чашах, нам нужно класть на каждую чашу равные по массе гирьки. Для начала нам необходимо отсортировать кучу камней по возрастанию или убыванию, чтобы однозначно получить такое множество камней, где мы сможем удобно брать камни для получения равенства в кучах. Теперь попробуем класть в каждую кучу пары из самого тяжёлого и самого лёгкого камня в исходной куче. Ручная трассировка алгоритма на тестовом множестве показала, что вариант гиблый.

```
5 8 13 27 14 -> 27 14 13 8 5

27 + 5          14 + 13
27 + 5 + 8      14 + 13
40              27

40 - 27 = 25
```

Суммы между весом n-го камня и n + 1 камня в отсортированный по убыванию куче, может оказаться значительнее больше, чем у следующей пары. Получается, нам нельзя брать камни из противоположных концов множества, нужно брать их строго по порядку, чтобы в каждый из куч был максимально тяжёлый невзятый камень. А для

минимизации разрыва весов с каждый взятым камнем, будем действовать так: возьмём камень n1 для левой кучи. Возьмём камень n2 для правой кучи. Возьмём камень n3 и положим его в меньшую из куч. С каждым последующим камнем мы будем поступать похожим образом.

```
    unsigned long sum1;
    unsigned long sum2;
    for (int i = 0; i < n; i++) {
        if (sum1 <= sum2) sum1 += rocks[i];
        else sum2 += rocks[i];
    }
    cout << (sum1 - sum2) << endl;

      5 8 13 27 14 -> 27 14 13 8 5

      27              14
      27              14 + 13
      27 + 8          14 + 13
      27 + 8          14 + 13 + 5
      35              32
          35 - 32 = 3
```

К сожалению, существуют редкие выборки данных, на которых решение не является корректным (пр: 3 3 2 2 2). Мозговой шторм на модификацию текущего алгоритма не дал результата, поэтому было решено, что можно перебрать все значения. Будем на каждом шаге рекурсии класть камешек в разные кучи, а на вершине стека сравним модуль полученной разности куч (обязательно модуль! так-как в отличии от цикла мы не задаём явно, в какую кучу класть камень при их равенстве) с заведомо большим значением.

1296. Гиперпереход

$$\boldsymbol{T(n) = O(n)}$$

```
#include <iostream>
using namespace std;

int main() {
    int n, input, sequence = 0, answer = 0;
    cin >> n;
    for (int i = 0; i < n; i++) {
        cin >> input;
        sequence += input;
        if (sequence < 0) sequence = 0;
        if (sequence > answer) answer = sequence;
    }
    cout << answer << endl;
    return 0;
}
```

Наша цель - найти самую «тяжёлую» положительную последовательность из существующих. Для этого нам достаточно проверять на каждом шаге, стала ли последовательность < 0: если да, то мы её сбросим, так как гиперпереход уже не

возможен, после чего обновим переменную, накапливающую гравитационный потенциал текущим значением последовательности, если она превосходит накопленную.

2025. Стенка на стенку

$$T(n) = O(n)$$

```
#include <iostream>
using namespace std;

int main() {
    int t;
    long n, k, teamSize, extTeamSize, remnant;
    cin >> t;
    for (int i = 0; i < t; i++) {
        cin >> n >> k;
        teamSize = n / k;
        remnant = n % k;
        extTeamSize = teamSize + 1;
        cout << ((n - extTeamSize * remnant) * (n - teamSize) + (extTeamSize
* remnant) * (n - teamSize - 1)) / 2 << endl;
    }
    return 0;
}
```

Так как бойцы сражаются "стенка на стенку", необходимо найти, сколько людей в "стенке" у нас будет. n и k даны, значит количество людей в команде = n / k. В частном случае задача сводится к простой комбинаторике: каждая человек может сразится с каждом, за исключением его текущим напарников. Сложности же задачи добавляет тот факт, что n % k не всегда = 0, а значит надо раскидать оставшихся людей по командам так, чтобы максимизировать количество боёв. Так как остаток человек стремится, но никогда не привысит количество команд (в таком случае мы смогли бы сформировать ещё одну команду, что означало бы неккоректность условия), можно добавить по человеку из остатка в сформированные команды - такие группы можно называть "расширенными". Теперь можно считать бои. Для этого нужно перемножить количество людей в команде (n - extTeamSize * remnant) на количество их противников (n - teamSize) и, перемножить количество людей в расширенной команде (extTeamSize * remnant) на количество противников для них (n - teamSize - 1). Полученные результаты складываем, для получения всех возможных комбинаций боёв, и делим пополам, чтобы исключить повторяющиеся перестановки.