Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет программной инженерии и компьютерной техники

Отчет по блоку №1 (timus)

По Алгоритмам и структурам данных

Выполнил:

Ступин Тимур Русланович

Группа № P3208

Поток № 1.3

Преподаватель:

Тараканов Денис Сергеевич

Санкт-Петербург 2025

Содержание

Задача 1 Куча камней

Решение

Очевидно что рассматривая камень с номером i мы должны выполнить одно из двух действий:

1. Положить камень i в первую кучу
2. Положить камень i во вторую кучу

Идея алгоритма заключается в переборе всех возможных разделений камней на две кучи и нахождении минимальной разницы.

Данный перебор реализуется рекурсивно. Сама функция принимает текущие значений сумм, номер рассматриваемого камня и ссылку на массив камней. Базой рекурсии является окончание массива камней, в данном случае просто возвращается модуль разности сумм.

Сложность алгоритма

1. **По времени:** Каждый камень может быть положен в одну из двух куч, что дает два различных состояния, а следовательно и два рекурсивных вызова. Всего получается 2^n вариантов, что даёт сложность O(2^n)
2. **По памяти:** С учётом того что каждый вызов хранится в стеке, а вызовов как ранее было посчитано 2^n получаем сложность O(2^n)

Итого:

- **По времени:** O(2^n)
- **По памяти:** O(2^n)

Код

```
#include <bits/stdc++.h>
using namespace std;

int f(int sum1, int sum2, int ptr, vector<int> &w) {
    if (ptr == w.size())
        return abs(sum2 - sum1);
    return min(f(sum1 + w[ptr], sum2, ptr + 1, w),
               f(sum1, sum2 + w[ptr], ptr + 1, w));
}

int main() {
    int n;
    cin >> n;
    vector<int> w(n);
    for (int i = 0; i < n; i++)
        cin >> w[i];
    cout << f(0, 0, 0, w) << endl;
    return 0;
}
```

Задача 2 Дуоны

Решение

Заметим, что возможны только три пары вершин:

1. На одно ребре
2. На диагонали стороны
3. На диагонали куба

При этом единственная неустранимая конфигурация это пары на диагонали определенной стороны. Действительно, в случае когда пары расположены на одном ребре всё очевидно, а в случае расположения на диагонали куба можно воспользоваться опцией создания дуонов. Вот поясняющий пример для данной ситуации:

Допустим мы хотим удалять пары дуонов в вершинах A и G. Тогда можно сначала выполнить прибавление BC+, после чего пару удалений AB- и GC-. Также есть второй вариант: EF+, AE-, FG-. Аналогичные комбинации существуют и для трех других диагоналей куба. При этом важно, что на другие вершины кроме рассматриваемой пары данное удаление не влияет.

Значит, достаточно можно разбить все вершины на две равные группы, каждая из которых будет состоять исключительно из пар второго типа. В таком случае, так как аннигиляция внутри группы невозможна, единственных вариантом будет взаимное уничтожение групп. Отсюда вытекает единственное условие невозможности разбиения: неравенство сумм вершин в двух группах. В противном случае аннигиляция возможна, и для этого достаточно последовательно последовательно занулять вершины например из первой группы, используя как пару вершины из второй.

В качестве пары групп я выбрал: {A, F, H, C} и {E, B, D, G}.

В начале программы происходит проверка равенства сумм в двух группах.

Далее просто происходит последовательный процесс обнуления вершин. Для удобства я использовал структуры, выделив два варианта пар:

Обычные (по ребрам), для которых для каждой вершины из первой группа я храню список из таких вершин второй группы.

Диагональные (куба), для них помимо пары вершин из первой и второй группы хранится пара вершин через которую будет происходить удаление по описанному выше алгоритму.

Сложность алгоритма

1. **По времени:** Размеры всех массивов фиксирован, а значит и время прохода по ним константно, что даёт сложность O(1)
2. **По памяти:** Размеры всех массивов фиксирован, что даёт сложность O(1)

Итого:

- **По времени:** O(1)
- **По памяти:** O(1)

Код

```
#include <bits/stdc++.h>
using namespace std;

struct basic_pairs {
  char g1; vector<char> g2s;
};

struct diagonal_pair {
  char g1; char g2;
  char g11; char g21;
};

void print_string_n_times(const string& s, int n) {
  while (n--) {
    cout << s << endl;
  }
}

int main() {
  vector<int> v(8);
  const vector group1 = {'A', 'C', 'F', 'H'};
  const vector group2 = {'B', 'D', 'E', 'G'};
  const vector<basic_pairs> basic = {
      {'A', {'B', 'D', 'E'}}, {'C', {'B', 'D', 'G'}},
      {'F', {'B', 'G', 'E'}}, {'H', {'G', 'D', 'E'}}
  };
  const vector<diagonal_pair> diagonal = {
      {'A', 'G', 'B', 'C'}, {'C', 'E', 'B', 'A'},
      {'F', 'D', 'B', 'A'}, {'H', 'B', 'D', 'A'}
  };
  for (int i = 0; i < 8; i++)
    cin >> v[i];

  int sum1 = 0;
  int sum2 = 0;
  for (const auto c : group1)
    sum1 += v[c - 'A'];
  for (const auto c : group2)
    sum2 += v[c - 'A'];
  if (sum1 != sum2) {
    cout << "IMPOSSIBLE" << endl;
```

```
    return 0;
  }

  for (const auto& [g1, g2s] : basic) {
    for (auto g2 : g2s) {
      const int min_val = min(v[g1 - 'A'], v[g2 - 'A']);
      if (min_val == 0)
        continue;
      v[g1 - 'A'] -= min_val;
      v[g2 - 'A'] -= min_val;
      print_string_n_times(string({g1, g2, '-'}), min_val);
    }
  }
  for (auto [g1, g2, g11, g21] : diagonal) {
    const int min_val = min(v[g1 - 'A'], v[g2 - 'A']);
    if (min_val == 0)
      continue;
    v[g1 - 'A'] -= min_val;
    v[g2 - 'A'] -= min_val;
    print_string_n_times(string({g11, g21, '+'}), min_val);
    print_string_n_times(string({g1, g11, '-'}), min_val);
    print_string_n_times(string({g2, g21, '-'}), min_val);
  }
  return 0;
}
```