1 A. Биологическое исследование

1.1 Условие

Вы являетесь специалистом в области биологической информатики, и вашей задачей является классификация последовательностей клеток в живых организмах.

Все виды клеток упорядочены в некую "кодовую таблицу", в которой каждой клетке соответствует строчная или заглавная латинская буква. Гарантируется, что код для каждой клетки уникален.

Согласно вашим исследованиям, вы заметили, что затраты на анализ последовательностей, в которых соседние клетки расположены ближе друг к другу в выбранной модели кодовой таблицы, значительно ниже.

Расстояние между двумя соседними клетками можно рассчитать как манхэттенское расстояние в пространстве координат кодовой таблицы. Манхэттенское расстояние вычисляется как абсолютная разница между значениями координат по каждому измерению.

Вам дана некоторая последовательность клеток, для которой вам необходимо будет рассчитать суммарное расстояние между всеми соседними парами клеток.

При подсчете учтем, что курсор в начальный момент времени уже находится на первом коде клетки (что будет влиять на начальное значение счетчика).

1.2 Входные данные

В первой строке вводится слово s ($2 \leq len(s) \leq 10^5$).

В следующей строке вводятся через пробел два числа: m и k — высота и ширина клавиатуры.

В следующих m строках вводятся по k букв ($1 \leq m \times k \leq 2 * 26$).

1.3 Вывод

В единственной строке необходимо вывести одно число — ответ на задачу.

1.4 Пример входных данных

Sample Input:
fRnMf
7 3
bMG
xiZ
Xfn
LNR
dHl
Ceg
WIp

Sample Output:

8
0

Sample Input:
xk
1 25
eBylZmxPobMIkchTjCfrHLWYR

Sample Output:

6
0

1.5 Решение

Запомним для каждого символа его положение на клавиатуре. Эту информацию будем хранить в словаре. Пройдемся по строке и посчитаем расстояния между соседними в строке символами. Сложность решения - $O(m * k + len(s) * log(m * k))$.

```
#include <iostream>
#include <stdint.h>
#include <string>
#include <unordered_map>

#define iom                                                                    \
  std::cin.tie(0);                                                             \
  std::cout.tie(0);                                                            \
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"

using ll = long long;
using ull = unsigned long long;

void solve(std::istream &cin, std::ostream &cout) {
  std::string s;
  cin >> s;

  int m, k;
  cin >> m >> k;

  std::unordered_map<char, std::pair<int, int>> keyboard;
  for (int i = 0; i < m; ++i) {
    for (int j = 0; j < k; ++j) {
      char c;
      cin >> c;
      keyboard[c] = std::make_pair(i, j);
    }
  }

  int ans = 0;
  auto [cur_i, cur_j] = keyboard[s[0]];
  for (int i = 1; i < s.size(); ++i) {
    auto [next_i, next_j] = keyboard[s[i]];
    ans += abs(cur_i - next_i) + abs(cur_j - next_j);

    cur_i = next_i;
    cur_j = next_j;
  }

  cout << ans << endl;
}

int32_t main() {
  iom;

  int T = 1; // cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}
```

//

2 B. Революционный материал

2.1 Условие

Вы работаете в известной нобелевской лаборатории по изучению структуры атомов и их расположения в различных материалах. Вашей задачей является экспериментальное моделирование наноматериалов.

Около десяти лет назад ваша команда обнаружила необычный тип материала, структура внутренних атомов которого образовывала равнобедренный прямоугольный треугольник с катетами длиной n.

На пересечении строки и столбца в этой атомной решетке находится атом с определенной энергией.

Из-за ограничений оборудования вашей лаборатории максимум энергии в i-ом столбце и i-ой строке атомной решетки треугольника не должен превышать величины a_i. Строки решетки нумеруются сверху вниз, а столбцы – слева направо.

Вам необходимо посчитать максимально возможную сумму энергий атомов в решетке треугольника при заданных ограничениях на максимумы в столбцах и строках.

2.2 Входные данные

В первой строке вводится единственное число n ($2 \leq n \leq 10^5$) – длина катетов треугольной решетки.
Во второй строке через пробел вводятся разделенные пробелом числа a_i ($1 \leq a_i \leq 10^9$) – наибольшие разрешенные максимумы в i-ой строке и i-ом столбце.

2.3 Вывод

В единственной строке необходимо вывести одно число – максимальную возможную сумму.

2.4 Примечания

Пусть $n = 4$.
$a = [6, 7, 8, 4]$
Пример корректной расстановки:
4
3 5
6 7 8
1 2 3 4
Пример корректной расстановки с максимальной суммой:
6
6 7
6 7 8
4 4 4 4
В этом примере в первом столбце все элементы не больше шести, во втором — семи, в третьем — восьми и четвертом — четырех. Аналогично со строками.

2.5 Пример входных данных

Sample Input:
5
3 2 3 4 3

Sample Output:
41

Sample Input:
3
2 4 2

Sample Output:
14

2.6 Решение

Заметим, что на каждый элемент треугольника действуют 2 ограничения - по строке и по столбцу.

Поступим жадно, приравняем каждый элемент к минимуму из этих двух ограничений.

Рассмотрим все записанные в треугольник числа. Обнаружим, что элемент в строке i и в столбце j равняется $min(a[i], a[j])$, где a - исходный массив. А, значит, максимально возможная сумма равняется $\sum_{i=0}^{n-1} \sum_{j=0}^{n-1} min(a[i], a[j])$.

Рассмотрим числа в порядке возрастания, ведь от этого сумма попарных минимумов не изменится.

Для $i = 0$ $min(a[i], a[j]) = a[i]$, значит число $a[i]$ будет сложено n раз.

Для $i = 1$ $min(a[i], a[j]) = a[i]$ для всех j кроме $j = 0$, значит число $a[i]$ будет сложено $(n - 1)$ раз.

...

Для $i = n - 1$ $min(a[i], a[j]) = a[i]$ только для $j = n - 1$, значит число $a[i]$ будет сложено 1 раз.

Сложность решения - $O(n * log(n))$.

```
#include <algorithm>
#include <iostream>
#include <stdint.h>
#include <vector>

#define iom                                                                    \
  std::cin.tie(0);                                                             \
  std::cout.tie(0);                                                            \
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"

using ll = long long;
using ull = unsigned long long;

void solve(std::istream &cin, std::ostream &cout) {
  ull n;
  cin >> n;

  std::vector<ull> a(n);
  for (size_t i = 0; i < n; ++i) {
    cin >> a[i];
  }

  std::sort(a.begin(), a.end());

  ull ans = 0;
  for (size_t i = 0; i < n; ++i) {
    ans += a[i] * (n - i);
  }

  cout << ans << endl;
}

int32_t main() {
  iom;

  int T = 1; // cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}
```

//

3 C. Новая эпоха

3.1 Условие

В эпоху информационных технологий мы все регулярно сталкиваемся с большим количеством информации. В общем потоке данных важно уметь выделить именно ту, которая наиболее релевантна для нас.

Работники лаборатории решили автоматизировать процесс обработки данных, поступающих с космического оборудования, которое отправляет стрим данных, состоящий из сгруппированных последовательностей сигналов.

Для этих целей были разработаны специальные фильтры, которые состоят из последовательностей символов.

Старший разработчик лаборатории подхватил неизвестную космическую болезнь, поэтому вам предстоит разработать программу, которая распределит входные последовательности сигналов по определенным фильтрам.

Сигнал считается подходящим под фильтр, если совпадает с непустым префиксом фильтра. При совпадении с префиксами нескольких фильтров необходимо выбрать тот, что находился во входных данных раньше.

Гарантируется, что все фильтры различны.

3.2 Входные данные

В первой строке вводится число n ($1 \le n \le 10^4$) — количество фильтров.

В следующих n строках вводятся фильтры — f_i, состоящие из латинских строчных букв. $\sum_i len(f_i) \le 10^5$.

В $(n+1)$-ой строке вводится число m ($1 \le m \le 10^4$) — количество сигналов.

В следующих m строках вводятся сигналы — s_j, закодированные в латинских строчных буквах, $\sum_j s_j \le 10^5$.

3.3 Вывод

Выведите m чисел — номер i фильтра, с префиксом которого совпадает s_j сигнал или -1, если такого фильтра нет.
При нескольких возможных ответах необходимо выбрать ответ с наименьшим i.

3.4 Примечания

Группы
1. 15 баллов: $n \le 100$, $m \le 100$.
2. 25 баллов: $n = m, n, m \le 10000$. Гарантируется, что каждому слову соответствует только один фильтр.

3. 60 баллов: без дополнительных ограничений.

3.5 Пример входных данных

Sample Input:
3
abac
abad
abacaba
10
ab
abad
abaca
abacab
b
abacaba
abacazav
abacabas
a
aba
Sample Output:
1
2
3
3
-1
3
-1
-1
1
1

3.6 Решение

Для решения воспользуемся структурой данных "бор" ("префиксное дерево"), которое позволяет эффективно хранить и искать префиксы в наборе строк.

При считывании заполним бор и в каждом листовом узле запомним номер фильтра, к которому он относится. Построим граф с родительскими индексами для каждого узла для преподсчета ответа.

Перед всеми запросами пройдемся по бору от листов и заполним для каждого узла наименьший индекс фильтра, на префиксе которого он лежит.

Далее для каждого сигнала пройдемся по бору и выведем заранее найденный индекс фильтра, если сигнал не выходит за пределы дерева.

Оценим асимптотику алгоритма: Пусть F – суммарная длина фильтров, S – суммарная длина всех сигналов. На построение бора и преподсчета потребуется $2F$ действий: F для построения бора и F для прохождения по бору для преподсчета индексов. Для ответа на все запросы потребуется не более S действий для нахождения префиксов в боре.

Итоговая асимптотика $O(F + S)$.

```
#include <iostream>
#include <set>
#include <stdint.h>
#include <unordered_map>
#include <vector>

#define iom                                                    \
  std::cin.tie(0);                                             \
  std::cout.tie(0);                                            \
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"

using ll = long long;
using ull = unsigned long long;
using namespace std;
#define int long long

struct Node {
  int leaf = -1;
  vector<int> next;

  Node() : leaf(-1) { next.resize(26, -1); }
};

vector<Node> t;
vector<int> leafs;
unordered_map<int, set<int>> gr;

void build(int idx, const string &s) {
  int v = 0;

  for (size_t i = 0; i < s.length(); ++i) {
    char c = s[i] - 'a';

    if (t[v].next[c] == -1) {
      int next_v = t.size();
      t.emplace_back();
      t[v].next[c] = next_v;
    }

    auto next_v = t[v].next[c];

    gr[next_v].insert(v);

    v = next_v;
  }

  t[v].leaf = idx;
  leafs.push_back(v);
}

int find(const string &s) {
  int v = 0;
  for (int i = 0; i < s.size(); ++i) {
    int c = s[i] - 'a';

    int next = t[v].next[c];
    if (next == -1) {
      return -1;
    }

    v = next;
  }

  return t[v].leaf;
}

void setLeafValues() {
  vector<pair<int, int>> st;
  for (const auto &leaf : leafs) {
    st.emplace_back(leaf, t[leaf].leaf);
  }

  while (!st.empty()) {
    auto [v, idx] = st.back();
    st.pop_back();

    if (t[v].leaf != -1 && t[v].leaf < idx) {
```

```
      continue;
    }

    t[v].leaf = idx;
    if (gr.find(v) != gr.end()) {
      for (auto &p : gr[v]) {
        st.emplace_back(p, idx);
      }
    }
  }
}

void solve(std::istream &cin, std::ostream &cout) {
  int n;
  cin >> n;

  t.emplace_back();

  for (int i = 1; i <= n; ++i) {
    string s;
    cin >> s;

    build(i, s);
  }

  setLeafValues();

  int m;
  cin >> m;
  while (m--) {
    string s;
    cin >> s;

    int ans = find(s);
    cout << ans << endl;
  }
}

int32_t main() {
  iom;

  int T = 1; // cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}
```

r //

4 D. AI

4.1 Условие

Это интерактивная задача. Обратитесь к секции "Протокол взаимодействия"для лучшего понимания.

В современном мире программируемые искусственные интеллекты стали неотъемлемой частью жизни общества.

Разработка систем обучения с подкреплением, обеспечивающих возможность обучения AI на больших массивах информации, является одной из главных задач специалистов в области искусственного интеллекта.

AI, под названием *MEMPHI-3000*, был спроектирован для работы с огромными базами данных. Для выполнения одного из основных типов задач - поиска информации - *MEMPHI-3000* использует структурированные блоки данных в виде матрицы. В матрице, размер которой составляет $n \times n$, в каждом столбце и строке числа идут по возрастанию. Гарантируется, что матрица генерируется перед всеми запросами от AI и остается между ними неизменной.

В процессе обучения от AI поступают интерактивные запросы, в рамках которых вводится определенное число X. Вашей задачей является обнаружение данного числа — определение индексов столбца и строки, в которых оно находится, или возвращение значения -1, если такое число в матрице отсутствует.

4.2 Входные данные

В первой строке вводится число q ($1 \le q \le 10^2$) — количество запросов.
Во второй строке вводится число n ($2 \le n \le 10^3$) — размер матрицы.

4.3 Вывод

На каждый запрос выведите: "! i j"— номер строки i и номер столбца j, в котором находится необходимое число X, если число присутствует в матрице и "! -1"— иначе.

4.4 Примечания

Группы

1. 30 баллов. Ограничение на количество запросов $n * n$
2. 70 баллов. $n > 10$, ограничение на количество запросов $2 * n * q$

Протокол взаимодействия

В начале каждого запроса ваша программа должна считать одно целое число X ($1 \le X \le 10^{18}$).

Далее ваша программа должна выводить в стандартный вывод запросы двух видов:

- если запрос имеет вид "? i j" ($1 \le i, j \le n$), где i и j – номера строки и столбца из матрицы, ответом от тестирующей системы будет число, находящееся на этой позиции в матрице;

- если программа сообщает позицию числа в матрице, запрос должен иметь вид: "! i j" ($1 \le i, j \le n$), если число присутствует в матрице и "! -1", если такого числа в матрице нет.

После вывода каждой строки не забудьте сбросить буфер вывода. Например:
- fflush(stdout) в C/C++;
- System.out.flush() в Java;
- sys.stdout.flush() в Python;
- flush(output) в Pascal;

пусть в тесте была сгенерирована матрица следующего вида:
1 2 3 4
5 7 8 10
6 12 20 70
30 50 90 101

—

Взаимодействие тестирующей системы и решения участника

сообщение системы: 2 // q – количество запросов
сообщение системы: 4 // n – размер матрицы

сообщение системы: 7 – число X
сообщение участника: ? 2 3
сообщение системы: 8
сообщение участника: ? 2 2
сообщение системы: 7
сообщение участника: ! 2 2

сообщение системы: 4 – число X
сообщение участника: ? 1 4
сообщение системы: 4
сообщение участника: ! 1 4

Лишние строки были добавлены для форматирования. При тестировании решения их не будет

4.5 Пример входных данных

Sample Output:
? 1 5
? 1 4
? 1 3
? 1 2
? 1 1
! 1 1

4.6 Решение

Для решения воспользуемся замечанием:
Пусть мы знаем значение ячейки $matrix_{i,j}$. По условию все строки и столбцы отсортированы по возрастанию. Отсюда известно, что $\forall k : 1 \leq k < j, matrix_{i,k} < matrix_{i,j}$ и $\forall h : i < h \leq n, matrix_{i,j} < matrixh, j$.

Если значение ячейки $matrix_{i,j}$ меньше загаданного X, ответ находится слева от ячейки: в прямоугольнике $matrix_{x,y} : (i \leq x \leq n, 1 \leq y < j)$. Если больше загаданного числа, то ответ находится снизу: в прямоугольнике $matrix_{x,y} : (i < x \leq n, 1 \leq y \leq j)$.

Для того, чтобы проверить всю таблицу необходимо проверять с правого верхнего угла, то есть с ячейки $matrix_{1,n}$. После каждого запроса будем изменять один из индексов в зависимости от ответа системы.

Максимальное количество запросов, необходимых для нахождения числа X в таблице или проверки, что его там точно нет, равняется $[n+(n-1)]$, так как каждый поиск нужно начинать в ячейке $matrix_{1,n}$ и наибольшее количество действий будет, если мы дойдем до ячейки $matrix_{n,1}$.

Общая асимптотика работы программы будет $O(2 * n * q)$.

```
#include <iostream>
#include <map>
#include <stdint.h>
#include <unordered_map>

#define iom                                                    \
  std::cin.tie(0);                                             \
  std::cout.tie(0);                                            \
  std::ios_base::sync_with_stdio(false);
#define out std::cout.flush();
#define endl "\n"

using ll = long long;
using ull = unsigned long long;
using namespace std;

const ll NMAX = 1000000LL;

ll n, q;
std::unordered_map<ll, ll> cache;
std::unordered_map<ll, ll> known;

ll ask(int i, int j, std::ostream &cout) {
  ll hash = NMAX * i + j;
  if (cache.find(hash) != cache.end()) {
```

```
    return cache[hash];
  }

  cout << "? " << i << " " << j << endl;
  out;

  ll ans;
  cin >> ans;
  cache[hash] = ans;
  known[ans] = hash;

  return cache[hash];
}

void answer() {
  cout << "! -1\n";
  out;
}

void answer(int i, int j) {
  cout << "! " << i << " " << j << endl;
  out;
}

void solveInner(std::ostream &cout) {
  ll X;
  cin >> X;

  if (known.find(X) != known.end()) {
    ll hash = known[X];
    int i = hash / NMAX;
    int j = hash % NMAX;

    answer(i, j);
    return;
  }

  int cur_i = 1, cur_j = n;

  while (1 <= cur_i && cur_i <= n && 1 <= cur_j && cur_j <= n) {
    ll x = ask(cur_i, cur_j, cout);

    if (x == X) {
      answer(cur_i, cur_j);

      return;
    } else if (x < X) {
      ++cur_i;
    } else if (x > X) {
      --cur_j;
    }
  }

  answer();
  return;
}

void solve(std::istream &cin, std::ostream &cout) {
  cin >> q;
  cin >> n;

  while (q--) {
    solveInner(cout);
  }
}

int32_t main() {
  iom;

  int T = 1; // cin >> T;
  while (T--) {
    solve(std::cin, std::cout);
  }

  out;
  return 0;
}
```

//

5 E. Турнир

5.1 Условие

Женя, лаборант университета М, в обеденный перерыв решил провести мысленный эксперимент, связанный с турниром по игре "Камень-ножницы-бумага"(К-Н-Б). В турнире участвуют $N = 2^k$ игроков, где k - целое неотрицательное число. Турнир проходит по олимпийской системе: игроки разбиваются на пары (1-й играет со 2-м, 3-й с 4-м и т.д.), победители пар выходят в следующий раунд, а проигравшие выбывают из борьбы за первое место.

Для каждого игрока известны три числа: r_i - вероятность, с которой i-й игрок покажет камень, p_i - вероятность, с которой i-й игрок покажет бумагу, и s_i - вероятность, с которой i-й игрок покажет ножницы. Сумма этих трех вероятностей для каждого игрока равна 1.

Игроки, проигравшие в полуфиналах, играют между собой матч за третье место.

Ваша задача состоит в том, чтобы для каждого игрока вывести три числа: вероятность занять первое, второе и третье место в турнире соответственно.

5.2 Входные данные

В первой строке задано целое число T ($1 \leq T \leq 1000$) - количество тестовых случаев.

Далее следуют T тестовых случаев, каждый из которых имеет следующую структуру.

В первой строке тестового случая задано целое число k ($2 \leq k \leq 6$), определяющее количество игроков в турнире как $N = 2^k$.

В следующих N строках заданы по три числа r_i, p_i, s_i ($0.01 \leq r_i, p_i, s_i \leq 1$), ($r_i + p_i + s_i = 1$) - вероятности i-го игрока показать камень, бумагу и ножницы соответственно. Каждое из чисел является десятичной дробью и может содержать не более двух знаков после десятичной точки.

5.3 Вывод

Для каждого тестового случая выведите N строк, где N - количество игроков в данном тестовом случае. Каждая строка должна содержать три числа, разделенных пробелами:

1. вероятность i-го игрока занять первое место в турнире;
2. вероятность i-го игрока занять второе место в турнире;
3. вероятность i-го игрока занять третье место в турнире.

Все вероятности должны быть выведены в виде десятичных дробей с точностью до 7 знаков после десятичной точки.

5.4 Примечания

Группы тестов

1) ($k = 2$) (33 балла).
2) Без дополнительных ограничений (67 баллов).

5.5 Пример входных данных

Sample Input:
1
2
0.1 0.5 0.4
0.4 0.1 0.5
0.2 0.1 0.7
0.6 0.2 0.2

Sample Output:
0.1873030 0.2211477 0.2311912
0.2711268 0.3204225 0.2212441
0.1661069 0.1672264 0.3690683
0.3754633 0.2912034 0.1784964

5.6 Решение

Необходимо решить 2 подзадачи: определить результат игры двух игроков и определить вероятности каждого игрока занять первое, второе и третье место соответственно.

Пусть играют 2 игрока A и B. Обозначим за X вероятность победы первого. Тогда X = P(выиграть сразу) + P(выиграть не первым ходом). Заметим что P(выиграть не первым ходом) = P(ничьи) * P(выиграть первому), т.е. P(ничьи) * X. Получим уравнение X = P(выиграть сразу) + P(ничьи) * X, откуда находим X.

Давайте для каждого игрока и этапа турнира посчитаем вероятность дойти до этой стадии турнира. Как пройти во второй раунд? Обыграть соперника. Вероятность выигрыша - вероятность участника дойти до второго раунда. Как ему пройти в 3 раунд? Играем за игрока A, победитель из игроков C, D играет с нами. Вероятность прохода A в 3 раунд A = P(A выиграет C) * P(вероятность A выйти во 2 раунд) * P(вероятность C выйти в 3 раунд) + P(A выиграет D) * P(вероятность A выйти во 2 раунд) * P(вероятность D выйти в 3 раунд). Для каждой стадии и каждого игрока храним вероятность оказаться в этой стадии турнира. И аккуратно разбираем случаи первого, второго и третьего места.

```
#include <iostream>
#include <vector>
#include <set>
#include <unordered_set>
#include <map>
#include <unordered_map>
#include <algorithm>
```

```cpp
#include <cmath>
#include <cassert>

using namespace std;

using ll = long long;

const int MOD = 997;


struct chel{
  double k, n, b;
};

////////////////////////////////////////////////////////////////////

void add(double& a, double b) {
  a += b;
}

double mult(double a, double b) {
  return a * b;
}

double sum(double a, double b) {
  add(a, b);
  return a;
}

double obr(double x) {
  return ((long double)1.0) / x;
}

////////////////////////////////////////////////////////////////////

pair<double, double> play(chel a, chel b) { // first is a win, second is a lose
  double win_first = mult(a.k, b.n); add(win_first, mult(a.n, b.b)); add(win_first, mult(a.b, b.k));
  double win_second = mult(b.k, a.n); add(win_second, mult(b.n, a.b)); add(win_second, mult(b.b, a.k));
  return {mult(win_first, obr(sum(win_first, win_second))), mult(win_second, obr(sum(win_first, win_second)))};
}

void solve() {
  int k;
  cin >> k;
  int n = (1 << k);
  vector<chel> chels(n);
  for (auto& chel : chels) {
      cin >> chel.k >> chel.b >> chel.n;
  }
  vector<vector<double>> tournir(k + 1, vector<double>(n)); // [x][y] -                                                    y
  for (int i = 0; i < n; ++i) {
    tournir[0][i] = 1;
  }
  for (int stage = 1; stage <= k; ++stage) {
    int block_size = (1 << stage);
    for (int beg_block = 0; beg_block < n; beg_block += block_size) {
      for (int who_win = beg_block; who_win < beg_block + block_size; ++who_win) {
        int start_range = beg_block;
        int fin_range = beg_block + block_size / 2;
        if ((who_win - beg_block) < block_size / 2) {
          start_range = beg_block + block_size / 2;
          fin_range = beg_block + block_size;
        }
        for (int who_play_with = start_range; who_play_with < fin_range; ++who_play_with) {
          //int p = mult(tournir[stage - 1][who_win], tournir[stage - 1][who_play_with]);
          //pair<int, int> game_res = play(chels[who_win], chels[who_play_with]);
          double p = mult(tournir[stage - 1][who_win], tournir[stage - 1][who_play_with]);
          pair<double, double> game_res = play(chels[who_win], chels[who_play_with]);
          add(tournir[stage][who_win], mult(game_res.first, p));
        }
      }
    }
  }
  vector<double> second_place(n);
  {
    int stage = k;
    int block_size = (1 << stage);
    for (int beg_block = 0; beg_block < n; beg_block += block_size) {
      for (int who_lose = beg_block; who_lose < beg_block + block_size; ++who_lose) {
        int start_range = beg_block;
        int fin_range = beg_block + block_size / 2;
        if ((who_lose - beg_block) < block_size / 2) {
          start_range = beg_block + block_size / 2;
          fin_range = beg_block + block_size;
        }
        for (int who_play_with = start_range; who_play_with < fin_range; ++who_play_with) {
          double p = mult(tournir[stage - 1][who_lose], tournir[stage - 1][who_play_with]);
          pair<double, double> game_res = play(chels[who_lose], chels[who_play_with]);
          add(second_place[who_lose], mult(game_res.second, p));
        }
      }
```

```
    }
  }
  vector<double> lose_pf(n);
  {
    int stage = k - 1;
    int block_size = (1 << stage);
    for (int beg_block = 0; beg_block < n; beg_block += block_size) {
      for (int who_lose = beg_block; who_lose < beg_block + block_size; ++who_lose) {
        int start_range = beg_block;
        int fin_range = beg_block + block_size / 2;
        if ((who_lose - beg_block) < block_size / 2) {
          start_range = beg_block + block_size / 2;
          fin_range = beg_block + block_size;
        }
        for (int who_play_with = start_range; who_play_with < fin_range; ++who_play_with) {
          double p = mult(tournir[stage - 1][who_lose], tournir[stage - 1][who_play_with]);
          pair<double, double> game_res = play(chels[who_lose], chels[who_play_with]);
          add(lose_pf[who_lose], mult(game_res.second, p));
        }
      }
    }
  }
  vector<double> third_place(n);
  {
    int stage = k;
    int block_size = (1 << stage);
    for (int beg_block = 0; beg_block < n; beg_block += block_size) {
      for (int who_win = beg_block; who_win < beg_block + block_size; ++who_win) {
        int start_range = beg_block;
        int fin_range = beg_block + block_size / 2;
        if ((who_win - beg_block) < block_size / 2) {
          start_range = beg_block + block_size / 2;
          fin_range = beg_block + block_size;
        }
        for (int who_play_with = start_range; who_play_with < fin_range; ++who_play_with) {
          double p = mult(lose_pf[who_win], lose_pf[who_play_with]);
          pair<double, double> game_res = play(chels[who_win], chels[who_play_with]);
          add(third_place[who_win], mult(game_res.first, p));
        }
      }
    }
  }
  for (int i = 0; i < n; ++i) {
    cout << tournir.back()[i] << " " << second_place[i] << " " << third_place[i] << endl;
  }
}

////////////////////////////////////////////////////////////////////

signed main() {
  //precalc();
  cin.tie(nullptr);
  cout.tie(nullptr);
  ios_base::sync_with_stdio(false);
  int t = 1; cin >> t;
  while (t--) {
      solve();
  }
  return 0;
}

  ////////////////////////////////////////////////////////////////////
```