A. Невероятный Петрович

Для того, чтобы оценить минимальное и максимальное возможное количество сыворотки необходимо как-либо упорядочить данные о необходимом количестве ингредиентов и о количестве компонентов. Тогда максимальное количество сыворотки можно получить, если будут взяты компоненты с максимальным количеством вещества – из них нужно выбрать минимальное отношение k / e. Минимальное количество сыворотки получится если сопоставить ингредиент с максимальным количеством и компонент, которого меньше всего.

Решение на языке Python 3

```
e = sorted([int(x) for x in input().split()])
k = sorted([int(x) for x in input().split()])
print(k[0] // e[2], min([ k[i] // e[i] for i in range(-3, 0) ]))
```

Решение на языке GNU C++11

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  long long e[3] = {};
  long long k[6] = {};
  for(int i=0; i<3; ++i)
  {
    cin >> e[i];
    assert(e[i] > 0 && e[i] <= 1e10);
  }
  for(int i=0; i<6; ++i)
  {
    cin >> k[i];
    assert(k[i] > 0 && k[i] <= 1e10);
  }
  sort(e, e+3);
  sort(k, k+6);
  cout << k[0] / e[2] << " " << min(k[3] / e[0], min(k[4] / e[1], k[5] / e[2]));
  return 0;
}
```

B. Железный друг

Задача решается простым анализом дохода, который принес каждый из менеджеров. Нужно выбрать наибольший доход, а затем вывести номера всех менеджеров, которые этот доход принесли. Важно проверить, что доход был, т.е. что полученное значение максимума больше 0.

Решение на языке Python 3

```
w = [int(x) for x in input().split()]
c = [int(x) for x in input().split()]
r = [int(x) for x in input().split()]
d = [0] * 3
for i in range(3):
  d[i] = c[i] * r[i] - w[i]
mx = max(d)
if mx <= 0:
  print(0)
else:
  for i in range(3):
    if mx == d[i]:
print(i + 1, end=' ')
```

Решение на языке GNU C++11

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  long long data[3][3];
  long long res[3] = {};

  for(int i=0; i<3; ++i)
  {
    for(int j=0; j<3; ++j)
    {
      cin >> data[i][j];
    }
  }
  long long mx = 0;
  for(int i=0; i<3; ++i)
  {
    res[i] = data[1][i] * data[2][i] - data[0][i];
    mx = max(mx, res[i]);
  }
  if (!mx)
    cout << mx;
```

```
  else
    for(int i=0; i<3; ++i)
      if (res[i] == mx)
        cout << i + 1 << " ";
  return 0;
}
```

C. Круг

Для решения задачи нужно рассмотреть только один из четырех секторов (и потом умножить количество точек на 4). В рассматриваемом секторе нужно выбрать одну из координат для выполнения перебора, а для второй рассчитывать наибольшую целочисленную координату, которая может поместиться в круг с радиусом не превосходящим N / 2. Таким образом, в процессе перебора мы находим минимальный радиус и количество точек с целыми координатами.

Решение на языке Python 3

```
n = int(input())
y = step = 20
mxr = points = 0
r = n // 2
while y <= n // 2:
  x = int((r * r - y * y) ** 0.5) // 20 * 20
  points += x // 20 + 1
  calc_r = int((x * x + y * y) ** 0.5)
  while calc_r * calc_r < x * x + y * y:
    calc_r += 1
  mxr = max(mxr, calc_r)
  y += step
print(points * 4)
print(mxr)
```

Решение на языке GNU C++11

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

void solve(int n)
{
  int points = 0, mxr = 0;
  int y = 20;
  int r = n / 2;
```

```
  while (y <= r)
  {
    int x = (int)(sqrt(1LL * r * r - 1LL * y * y)) / 20 * 20;
    points += x / 20 + 1;
    int calc_r = (int) sqrt(1LL * x * x + 1LL * y * y) - 20;
    while (1LL * calc_r * calc_r < 1LL * x * x + 1LL * y * y) calc_r ++;
    mxr = max(mxr, calc_r);
    y += 20;
  }
  cout << 4 * points << endl << mxr;
}

int main() {
  int n;
  cin >> n;
  solve(n);
  return 0;
}
```

D. Венгерский кроссворд

Можно заметить, что так как по условию **гарантируется**, что все слова из списка можно вычеркнуть в филворде, то для получения ответа достаточно посчитать количество каждой буквы в кроссворде, а потом проходя по словам, которые подаются на вход, уменьшать соответствующий счетчик на 1. В конце нужно просто вывести буквы на экран.

Решение на языке Python 3

```
n, m = map(int, input().split())
cnt_alphabet = [0] * 26

for i in range(n):
  for c in input():
    cnt_alphabet[ord(c) - ord('A')] += 1

m = int(input())
for j in range(m):
  for c in input():
    cnt_alphabet[ord(c) - ord('A')] -= 1

for i in range(26):
print( chr(ord('A') + i) * cnt_alphabet[i], end='' )
```

Решение на языке GNU C++11

```
#include <bits/stdc++.h>
using namespace std;
int main()
{
  ios_base::sync_with_stdio(0);
  int n,m;
  cin >> n >> m;
  int cnt_alphabet[26] = {};

  for (int i = 0; i < n; ++i)
  {
    string s;
    cin >> s;
    for(auto c: s) cnt_alphabet[c - 'A'] ++;
  }
  cin >> m;
  for(int i=0; i < m; ++i)
  {
    string s;
    cin >> s;
    for(auto c: s) cnt_alphabet[c - 'A'] --;
  }

  for(int i=0; i<26; ++i)
  {
    for(int j = 0; j < cnt_alphabet[i]; ++j )
      cout << (char)('A' + i);
  }
}
```

Возможно решение с применением теории графов (например, поиска в глубину), однако, оно потребует применения рекурсивного перебора различных вариантов расстановки каждого слова. Кроме того, с указанными ограничениями решение при помощи поиска в глубину (или в ширину) не должно проходить по времени и не должно получить полный балл.

E. Максимальный deck

Эта задача является вариацией классической задачи на динамическое программирование. Для ее решение, необходимо сформулировать рекуррентное соотношение. Пусть нам известно значение функции F(i,j) –

максимальное количество очков, которые может набрать первый игрок, где i и j определяют отрезок [i, j], на котором мы ищем решение. Тогда рекуррентная формула будет следующей:

$$F(i,j) = \begin{cases} a[i], i == j \\ \max(a[i], a[j]), j - i == 1 \\ 0, i > j \\ \max(a[i] + \min\big(F(i+2,j), F(i+1,j-1)\big), \\ a[j] + min\big(F(i+1,j-1), F(i,j-2)\big)) \end{cases}$$

Первые три случая тривиальные, рассмотрим как вычисляется значения для случая, когда j – i > 1. В этом случае мы выбираем максимальное из двух возможных вариантов:

- берем a[i], и тогда нужно прибавить минимум из F(i+2,j), F(i+1,j-1), т.к. второй игрок своим ходом будет играть так, чтобы получить наибольшую возможную сумму
- берем a[j] и тогда нужно к нему прибавить минимум из F(i+1,j-1), F(i,j-2)

Решение на языке Python 3

```
suits = ['H', 'S', 'C', 'D']
values = ["2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"]

def get_value(suit, val):
  return suits.index(suit) * 13 + values.index(val);

n = int(input())
a = []
for i in range(n):
  suit, val = input().split()
  a.append(get_value(suit, val))

s = sum(a)
dp1 = []
for i in range(n):
  dp1.append( [0] * n )

for left in range(n-1,-1,-1):
  for right in range(left, n):
```

```
    if left == right:
      dp1[left][right] = a[left]
    elif right - left == 1:
      dp1[left][right] = max(a[left], a[right])
    else:
      dp1[left][right] = max(
        a[left] + min(dp1[left+2][right], dp1[left+1][right-1]),
        a[right] + min(dp1[left+1][right-1], dp1[left][right-2]),
      )

t = dp1[0][n-1]
if 2 * t > s:
  print('FYODOR')
elif 2 * t == s:
  print('DRAW')
else:
print('DJON')
```

Решение на языке GNU C++11

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

char suits[] = {'H', 'S', 'C', 'D'};
string values[] = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "J", "Q", "K", "A"};
int dp[101][101] = {};

int get_value(char suit, string val)
{
  int i = 0, j = 0;
  while (suit != suits[i]) ++i;
  while (val != values[j]) ++j;
  return i * 13 + j;
}

string get_card(int value)
{
  int i = value / 13;
  int j = value % 13;
  return string(1, suits[i]) + " " + values[j];
}

int f(int left, int right, const vector<int> & deck)
{
  if (!dp[left][right])
  {
    if (left > right) dp[left][right] = 0;
    else if (left == right) dp[left][right] = deck[left];
    else if (right - left == 1) dp[left][right] = max(deck[left], deck[right]);
    else
    {
      dp[left][right] = max(
```

```
        deck[left] + min(f(left+2, right, deck), f(left+1, right-1, deck)),
        deck[right] + min(f(left+1, right-1, deck), f(left, right-2, deck))
      );
    }
  }
  return dp[left][right];
}

void solve(const vector<int> & deck)
{
  int fedor = f(0, int(deck.size())-1, deck);
  int s = 0;
  for(int i=0; i<int(deck.size()); ++i) s += deck[i];
  if (2 * fedor > s)
    cout << "FYODOR";
  else if (2 * fedor == s)
    cout << "DRAW";
  else
    cout << "DJON";
}

int main() {
  vector<int> deck;
  int n;
  cin >> n;
  while(n > 0)
  {
    char suit;
    string val;
    cin >> suit >> val;
    deck.push_back(get_value(suit, val));
    n--;
  }
  solve(deck);
  return 0;
}
```