1вариант.

Последовательно применяя допустимые действия, будут получаться следующие объёмы в вёдрах:

№ Действия		1	2	3	4	5	6	7	8	9	10
Действие		А	Б	А	Б	А	Б	00	Б	А	Б
Ведро **А**	0	5	0	5	0	5	1	1	0	5	0
Ведро **Б**	0	0	5	5	10	10	14	0	1	1	**6**

Где действие А обозначает наполнить ведро **А**.

Действие Б – пересыпать ведро **А** в ведро **Б**.

Действие 00 – освободить ведро **Б**.

Таким образом, на десятом действии в ведре **Б** получаем необходимый объём. Для чего потребовалось 4 наполнения ведра **А**.

2 вариант.

Решение аналогично.

№ Действия		1	2	3	4	5	6	7	8	9	10	11	12
Действие		А	Б	А	Б	А	Б	А	Б	00	Б	А	Б
Ведро **А**	0	4	0	4	0	4	0	4	3	3	0	4	0
Ведро **Б**	0	0	4	4	8	8	12	12	13	0	3	3	**7**

На двенадцатом действии в ведре **Б** получаем необходимый объём. Для чего потребовалось 5 наполнений ведра **А**.

Задача 2.

1 вариант.

Чтобы найти искомый квадрат, для начала найдём максимальную удалённость тайников по оси абсцисс lenX и по оси ординат lenY. Очевидно, что lenX=maxX-minX, где maxX - наибольшая абсцисса среди всех тайников, minX - наименьшая. И аналогично lenY=maxY-minY. Тем самым мы нашли стороны минимального прямоугольника, но не квадрата! Сторона искомого квадрата будет равна максимуму из lenX и lenY. Зная сторону квадрата, можно легко вычислить его площадь.

2 вариант.

Решение аналогично, за исключением того, что в конце не нужно выбирать максимум из lenX и lenY. Ответом будет lenX*lenY.

Пример программы(на С++).оба варианта

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

int main(){
  int n,x,y,maxX=-1000000,minX=1000000,maxY=-
1000000,minY=1000000;
  long long lenX,lenY;
     cin>>n;
  for(int i=0;i<n;++i){
   cin>>x>>y;
   if(x>maxX)
    maxX=x;
   if(x<minX)
    minX=x;
   if(y>maxY)
    maxY=y;
   if(y<minY)
    minY=y;
  }
  lenX=maxX-minX;
  lenY=maxY-minY;
  if(lenX>lenY)  //этот
   lenY=lenX; //фрагмент
  else    //нужно убрать
   lenX=lenY; //для второго варианта
  cout<<lenX*lenY;
  return 0;
}
```

Задача 3.

1 вариант.

Очевидно, что получится девять трёхзначных чисел, и независимо от того, как цифры были расставлены по кругу, каждая цифра по одному разу по участвует в записи единиц, по одному разу в записи десятков и по одному разу в записи сотен. Таким образом искомую сумму можно было найти так: 999+888+777+666+555+444+333+222+111=4995.

2 вариант.

Решение аналогично. Искомая сумма: 9999+8888+...+1111=49995.

Задача 4.

1 вариант.

Грань октаэдра - это треугольник, значит, с каждой грани можно попасть на любую из трёх соседних, переворачивая через рёбра. Для того, чтобы проверить является ли данная последовательность следом, нужно для каждой цифры в последовательности, кроме последней, проверить, что следующая цифра есть на одной из трёх соседних граней.

Для грани a_1 соседними являются грани a_2, a_4 и a_8.

Для грани a_2 соседними являются грани a_1, a_3 и a_7.

Для грани a_3 соседними являются грани a_2, a_4 и a_6.

Для грани a_4 соседними являются грани a_1, a_3 и a_5.

Для грани a_5 соседними являются грани a_4, a_6 и a_8.

Для грани a_6 соседними являются грани a_3, a_5 и a_7.

Для грани a_7 соседними являются грани a_2, a_6 и a_8.

Для грани a_8 соседними являются грани a_1, a_5 и a_7.

Пример программы(на C++).

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

int main(){
  int n,mass[1000],raz[8],num,
     matr[8][3]={{2,4,8},
          {1,3,7},
          {2,4,6},
          {1,3,5},
          {4,6,8},
          {3,5,7},
          {2,6,8},
          {1,5,7}};//этим массивом будем пользоваться для
проверки на "соседство"
  bool f;
  cin>>n;
  for(int i=0;i<n;++i)
    cin>>mass[i]; //считываем последовательность
  for(int i=0;i<8;++i) //считываем развёртку
    cin>>raz[i];
  for(int i=0;i<n;++i)
    for(int j=0;j<8;++j)
      if(mass[i]==raz[j])
```

```
      mass[i]=j+1;//помечаем на какой грани находится
считанный элемент
  for(int i=0;i<n-1;++i){
    f=false;
    for(int j=0;j<3;++j)
      if(mass[i+1]==matr[mass[i]-1][j]){//проверяем, является
ли следующий элемент соседом
        f=true;
        break;
      }
    if(!f)
      break;
  }
  if(f)
    cout<<1;
  else
    cout<<0;

  return 0;
}
```

2 вариант.

Решение аналогично, только соседей не три, а четыре.

Для грани a_1 соседними являются все грани, кроме a_4.

Для грани a_2 соседними являются все грани, кроме a_6.

Для грани a_3 соседними являются все грани, кроме a_5.

Для грани a_4 соседними являются все грани, кроме a_1.

Для грани a_5 соседними являются все грани, кроме a_3.

Для грани a_6 соседними являются все грани, кроме a_2.

Пример программы(на C++).

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

int main(){
  int n,mass[1000],raz[8],num,
  matr[8]={4,6,5,1,3,2};//этим массивом будем пользоваться
для проверки на "соседство"
  bool f;
  cin>>n;
  for(int i=0;i<n;++i)
    cin>>mass[i];  //считываем последовательность
  for(int i=0;i<6;++i) //считываем развёртку
```

```
    cin>>raz[i];
  for(int i=0;i<n;++i)
    for(int j=0;j<6;++j)
      if(mass[i]==raz[j])
        mass[i]=j+1;//помечаем на какой грани находится
считанный элемент
  for(int i=0;i<n-1;++i)
    if(mass[i+1]==mass[matr[mass[i]-1]-1]){//проверяем,
является ли следующий элемент соседом
      f=false;
      break;
    }
  if(f)
    cout<<1;
  else
    cout<<0;

  return 0;
}
```

Задача 5.

1 вариант.

В примере указано, как «переместить» число из одной ячейки в другую. Иначе можно описать этот алгоритм так: пока в первой ячейке не ноль, число в первой ячейке уменьшаем на 1 и число во второй ячейке увеличиваем. Таким образом, a раз уменьшается, число в первой ячейке, и a раз увеличивается число во второй ячейке.

Вернёмся к поставленной задаче. Уменьшая число во второй ячейке, будем увеличивать число в первой ячейке. Когда во второй ячейке окажется 0, получим значения ячеек $a + b$ 0 c 0. То же самое проделаем с третьей ячейкой и получим $a + b + c$ 0 0 0, что и требовалось.

Пример программы.

1. D 2
2. T 5
3. I 1
4. T 1
5. D 3
6. T 0
7. I 1

8. *T* 5

2 вариант.

Решение аналогично, только теперь уменьшая числа в первой, второй и третьей ячейке, увеличивать будем в четвёртой.

Пример программы.

1. *D* 1
2. *T* 5
3. *I* 4
4. *T* 1
5. *D* 2
6. *T* 9
7. *I* 4
8. *T* 5
9. *D* 2
10. *T* 0
11. *I* 4
12. *T* 9