A Капитализация слова

Решение C++

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

int main() {
	string s;
	cin >> s;
	s[0] = toupper(s[0]);
	cout << s;
}
```

Решение Python

```python
s=input()
print(s[0].upper()+s[1:])
```

B Петя и строки

В данной задаче сначала нужно было привести обе строки к одинаковому регистру (например, сделать все буквы маленькими), а затем просто сравнить их лексикографически, пройдя слева направо по символам.

Решение C++

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

int main() {
	string s;
	cin >> s;
	s[0] = toupper(s[0]);
	cout << s;
}
```

Решение Python

```python
a = input().lower()
b = input().lower()
print(1 if a > b else -1 if a < b else 0)
```

C Перестановка строки

Нужно проверить, что обе строки состоят из одинакового набора символов.

Проще всего отсортировать символы в каждом строке и сравнить строки.

Решение C++

```
#include <iostream>
#include <string>

using namespace std;

int main()
{
	string str,str1;
	cin >> str>>str1;
	sort(str.begin(), str.end());
	sort(str1.begin(), str1.end());
	if (str == str1)
		cout << "YES";
	else cout << "NO";
}
```

Решение Python

```
print("YES" if sorted(input()) == sorted(input()) else "NO")
```

D Контрольная сумма

Можно явно решить задачу с помощью сортировки или структуры данных словарь (map)

Но есть хитрое решение за O(N). Воспользуемся побитовым оператором xor. Если мы применим xor к одинаковым числам, то они превратятся в ноль. А ноль с любым числом дает это число. И порядок действий не важен. Оператор xor чаще всего обозначается как ^

Решение C++

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

int main()
{
	int n, a;
	cin >> n;
	int b = 0;
	for (int i = 0; i < n; ++i) {
		cin >> a;
		b = b ^ a;
	}
	cout << b;
}
```

Решение Python

```
from functools import reduce
lst = list(map(int, input().split()))
print(reduce(lambda x, y: x ^ y, lst))
```

E Ложные новости

Воспользуемся методом двух указателей: пройдем циклом по строке одновременно сдвигая второй указатель в строке “heidi”. Если дошли до конца то ответ YES. O(N)

Решение C++

```
#include <iostream>
#include <string>
#include <algorithm>
using namespace std;
int main(){
	string s,p="heidi";
	int j=0;
	cin>>s;
	for(int i=0; i<s.size(); i++){
		if(s[i]==p[j]){
			j++;
			if(j==5) break;
		}
	}
	cout<<(j==5 ? "YES" : "NO");
}
```

Решение Python

```
s = str(input())
j = 0
ss = 'heidi#'
for i in s:
    if i == ss[j]:
        j += 1
print ('YES' if j == 5 else 'NO')
```

F Порядок 2

В данной задаче может показаться, что достаточно отсортировать числа как строки по возрастанию. Но это неверно, рассмотрим такой тест:

2

2 21

В данном случае 221 больше, чем 212.

Решение:

Пусть у нас есть массив 3 12 52 1

Склеим все в одну строку любым способом 312521.

Дальше будем много раз проходиться по массиву и пытаться обменять местами 2 числа. Если итоговая строка улучшается, то меняем. По сути, пузырьковая сортировка.

Еще короче: нужно проверять как выгоднее склеить a+b или b+a.

Можно добавить такое сравнение в сортировку (как компаратор)

Решение C++

```cpp
#include <iostream>
#include <vector>
#include <string>
#include <algorithm>
using namespace std;


bool cmp(string a, string b) {
    return a + b < b + a;
}

string mas[111];

int main()
{
    int n;
    cin >> n;

    for (int i = 0; i < n; i++)
        cin >> mas[i];

    sort(mas, mas + n, cmp);

    for (int i = 0; i < n; i++)
        cout << mas[i];

    return 0;
}
```

Решение Python

```python
import functools

def comp(x, y):
    if (x+y>y+x):
        return 1
    elif (x+y<y+x):
        return -1
    else:
        return 0
input()
arr = list(input().split())
print(*sorted(arr, key = functools.cmp_to_key(comp)), sep = '')
```

G Хеш строки

Задача на реализацию

```
#define _CRT_SECURE_NO_WARNINGS

#include <algorithm>
#include <iostream>
#include <sstream>
#include <string>

using namespace std;

uint32_t calcHash(string s, uint32_t p) {
    uint32_t res = 0;
    for (int i = 0; i < s.length(); i++)
        res = res * p + s[i];
    return res;
}

int main() {


    uint32_t p;
    cin >> p;
    string s;

    cin >> s;

    cout << calcHash(s, p);

    return 0;
}
```

H Hack hash

Проще всего было воспользоваться парадоксом дней рождения. Если мы сгенерируем миллион случайных строк, то среди них почти со 100% вероятностью будет две с одинаковыми хешами.

```
#include <iostream>
#include <cstdio>
#include <cmath>
#include <string>
#include <algorithm>
#include <vector>
#include <map>
#include <sstream>

using namespace std;
#pragma comment(linker, "/STACK:20000000")

uint32_t calcHash(string s, uint32_t p) {
    uint32_t res = 0;
    for (int i = 0; i < s.length(); i++)
        res = res * p + s[i];
```

```
    return res;
}

map <uint32_t, string> mp;

void solve()
{
    uint32_t p;
    cin >> p;
    while (true)
    {
        string s = "";

        int len = rand() % 30 + 1;
        for (int i = 0; i < len; i++)
        {
            char c = 'a' + rand() % 26;
            s += c;
        }

        uint32_t h = calcHash(s, p);

        if ((mp.find(h) != mp.end()) && mp[h] != s)
        {
            cout << s << endl;
            cout << mp[h] << endl;
            return;
        }
        mp[h] = s;
    }
}


int main()
{
    solve();

    return 0;
}
```

Либо можно воспользоваться генератором теста, описанным здесь (для модулей равных степеней двойки)

https://codeforces.com/blog/entry/4898?locale=ru

I Кольцо

Полиномиальный хеш.

`abcb`

a*p^3 + b*p^2 + c*p^1 + b = ((a*p + b)*p + c)*p + b

, где p достаточно большое простое число, например 3137

Считаем long long хеш с переполнением.

Как изменяется хеш при обмене двух элементов, например a и c

Вычитаем a*p^3 и c*p^1

Прибавляем c*p^3 и a*p^1

Как определять, эквивалентна ли текущая строка исходной.

Она должна быть подстрокой удвоенной исходной строки

`abcbabcb`

Посчитаем хеши для всех подстрок длины 4: `abcb bcba cbab babc abcb`

Это можно сделать за линейное время.

На каждом шаге запроса будем смотреть, есть ли у нас такой хеш.

Это можно сделать тремя способами

1. set или map. Это самый надежный. Но более медленный и требовательный к памяти. Сложность O(N*logN)
2. Использовать обычный массив. Отсортировать и искать бинарным поиском. Сложность O(N*logN). Работает быстрее сета и требует минимум памяти.
3. Использовать свою хеш-таблицу. Самый быстрый способ. Сложность O(N). Требователен к памяти. Размер массива лучше выбирать в 10 раз больше количества возможных хешей. У хеша берем например 20 младших бит. И смотрим в таблице есть ли на этом месте требуемый хеш. Если место занято, то переходим на следующую позицию. Так идем пока не дойдем до пустого места.

```
#include <stdio.h>
#include <sstream>
#include <iostream>
#include <string>
#include <algorithm>
#include <vector>
#include <list>
#include <iomanip>
#include <map>
#include <set>
#include <cmath>
#include <queue>
#include <cassert>
#include <string.h>
using namespace std;
#pragma comment(linker, "/STACK:20000000")

typedef vector<int> vi;
#define sz(a) int((a).size())
#define all(c) (c).begin(),(c).end()
```

```
string problem_name = "ring";

void init(){
	freopen((problem_name+".in").c_str(),"rt",stdin);
	freopen((problem_name+".out").c_str(),"wt",stdout);
}

const int maxlen=100000;
long long hval = 3137;
long long p[2*maxlen+115];
char st[2*maxlen+115];
long long hmas[2*maxlen+115];
int len;

long long calc(string &s)
{
	long long res=0;
	for (int i=0;i<sz(s);i++)
		res=res*hval+(long long)s[i];
	return res;
}
string s;

long long ht[(1<<20)+5];

void addhash(long long val)
{
	long long p = val&((1<<20)-1);
	while (ht[p])
	{
		p++;
		if (p>=(1<<20)) p = 0;
	}
	ht[p] = val;
}

int findhash(long long val)
{
	long long p = val&((1<<20)-1);
	while (ht[p])
	{
		if (ht[p]==val) return 1;
		p++;
		if (p>=(1<<20)) p = 0;
	}
	return 0;
}

long long gethash(int l, int r)  //  получаем значение на отрезке от l до r
{
	return hmas[r] - hmas[l-1]*p[r-l+1];
}


long long f(long long h,int a, int b) // обмен
{
	h-=s[a]*p[len-a-1];
	h-=s[b]*p[len-b-1];
	h+=s[a]*p[len-b-1];
	h+=s[b]*p[len-a-1];
	return h;
}
```

```
int main()
{
	init();

	//srand(5);
	p[0]=1;
	for (int i=1;i<100100;i++)
		p[i]=p[i-1]*hval;

		gets(st);
		len=strlen(st);
		s = st;
		s+=s;

		memset(ht,0,sizeof(ht));
		long long h=0;

		for (int i=0;i<len;i++)
			h=h*hval + s[i];

		addhash(h);

		long long cur = h;

		hmas[0]=s[0];
		for (int i=1;i<sz(s);i++)
			hmas[i]=hmas[i-1]*hval+s[i];

		for (int i=len;i<sz(s);i++)
			addhash(gethash(i-len+1,i));

		int n;
		scanf("%d",&n);

		for (int i=0;i<n;i++)
		{
			int a,b;
			scanf("%d%d",&a,&b);
			a--;
			b--;
			cur = f(cur,a,b);
			swap(s[a],s[b]);

			if (findhash(cur))
				printf("YES\n"); else
				printf("NO\n");
		}
	return 0;
}
```

J Тест

Можно решать через алгоритм **КМП** или **хешированием**.

Решение хешированием.
Пусть входные строки s_0, s_1, s_2. Постараемся найти кратчайшую строку, которая содержит s_0, s_1, s_2. Пройдём все порядки s_0, s_1, s_2 и поищем длиннейшее перекрытие в конце строки a и начале строки b. Польный перебор конечно требует слишком много времени. С другой стороны, хеширование может решить за $O(n)$ операции, где $n = min(len(a), len(b))$. Моя хеш-функция - полином $hash(x_0, x_1, ..., x_n) = x_0 + ax_1 + a^2x_2 + ... + a^nx_n$. Этот полином удобный в этой задаче потому что он имеет следующее свойство:
Если известно $hash(x_i, ..., x_j)$, тогда можно за $O(1)$ посчитать следующие значения:

- $hash(x_{i-1}, x_i, ..., x_j) = x_{i-1} + a \times hash(x_i, ..., x_j)$
- $hash(x_i, ..., x_j, x_{j+1}) = hash(x_i, ..., x_j) + a^{j+1-i} \times x_{j+1}$

Т.е., если известно значение хеш-функции какой-то подстроки, легко посчитать значение соседних подстрок. Для строк a, b, посчитаем значения хеш-функций подстрок в конце a и в начале b. Если они равные для подстрок размера n, тогда значит, что (может быть) есть дублирование n характеров в a и b.

Поэтому, пройдём все порядки s_0, s_1, s_2, и попробуем связать строки вместе. Осталась одна проблема --- если s_i подстрока s_j и $i \neq j$, тогда можно пропустить s_i. Используем хеширование быстро решить, s_i подстрока s_j или нет.

Решение C++

```
#include<iostream>
#include<cstdio>
#include<cstring>
using namespace std;
#define N 200020
#define NN 100010
char str[N];
char s[3][NN];
int p[N];

int kmp(char *s1,char *s2)
{
    int n=strlen(s1);
    int m=strlen(s2);
    int j=-1;
    p[0]=-1;
    for(int k=1;k<m;k++)
    {
        while(j>=0&&s2[k]!=s2[j+1]) j=p[j];
        if(s2[k]==s2[j+1]) j++;
        p[k]=j;
    }

    j=-1;
    for(int k=0;k<n;k++)
    {
        if(j==m-1) return 0;
        while(j>=0&&s1[k]!=s2[j+1]) j=p[j];
```

```
        if(s1[k]==s2[j+1]) j++;
    }
    return (m-1-j);
}
int compair(char *s1,char *s2,char *s3)
{
    int len=kmp(s1,s2);
    str[0]='\0';
    strcat(str,s1);
    strcat(str,s2+strlen(s2)-len);
    len=kmp(str,s3);
    return strlen(str)+len;
}
int main()
{
    while(scanf("%s%s%s",s[0],s[1],s[2])!=EOF)
    {
        int minn;
        int l;
        minn=compair(s[0],s[1],s[2]);

        l=compair(s[0],s[2],s[1]);
        if(l<minn)   minn=l;

        l=compair(s[1],s[2],s[0]);
        if(l<minn)   minn=l;

        l=compair(s[1],s[0],s[2]);
        if(l<minn)   minn=l;

        l=compair(s[2],s[0],s[1]);
        if(l<minn)   minn=l;

        l=compair(s[2],s[1],s[0]);
        if(l<minn)   minn=l;

        cout<<minn<<endl;
    }
}
```

Решение Python

```
def p(a,b):
    s,m,c,j=b+'#'+a,0,0,0;p=[0]*len(s)
    for i in range(1,len(s)):
        while j and s[i]!=s[j]: j=p[j-1]
        if s[i]==s[j]: j+=1
        p[i]=j
        if j==len(b): return a
    return a[:len(a)-p[-1]]+b
s=[input() for _ in '   ']
      print(min(len(p(s[x[0]],p(s[x[1]],s[x[2]]))) for x in
__import__('itertools').permutations([0,1,2])))
```