杂项

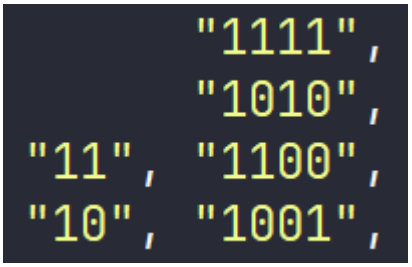
单测多测

```cpp
#include <bits/stdc++.h>

#define ranges std::ranges
#define views std::views

using u32 = unsigned;
using i64 = long long;
using u64 = unsigned long long;

using pii = std::pair<int, int>;
using a3 = std::array<int, 3>;
using a4 = std::array<int, 4>;

const int N = 1e6;
const int MAXN = 1e6 + 10;
const int inf = 1e9;
// const int mod = 1e9 + 7;
const int mod = 998244353;

std::mt19937_64 rng(std::chrono::steady_clock::now().time_since_epoch().count());

void solve() {

}

signed main() {
    std::ios::sync_with_stdio(false);
    std::cin.tie(0), std::cout.tie(0);
    int t = 1;//cin >> t;
    while (t--) {
        solve();
        std::cout << '\n';
    }
    return 0;
}
```

阿达马矩阵 (Hadamard matrix)

构造题用，其有一些性质：将 0 看作 -1；1 看作 $+1$，整个矩阵可以构成一个 2^k 维向量组，任意两个行、列向量的点积均为 0 See。例如，在 $k=2$ 时行向量 $\vec{2}$ 和行向量 $\vec{3}$ 的点积为 $1\cdot 1+(-1)\cdot 1+1\cdot(-1)+(-1)\cdot(-1)=0$。

构造方式：$\begin{bmatrix} T & T \\ T & !T \end{bmatrix}$

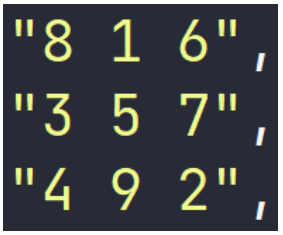



```cpp
int n;
cin >> n;
int N = pow(2, n);
vector ans(N, vector<int>(N));
ans[0][0] = 1;
for (int t = 0; t < n; t++) {
    int m = pow(2, t);
    for (int i = 0; i < m; i++) {
        for (int j = m; j < 2 * m; j++) {
            ans[i][j] = ans[i][j - m];
        }
    }
    for (int i = m; i < 2 * m; i++) {
        for (int j = 0; j < m; j++) {
            ans[i][j] = ans[i - m][j];
        }
    }
    for (int i = m; i < 2 * m; i++) {
        for (int j = m; j < 2 * m; j++) {
            ans[i][j] = 1 - ans[i - m][j - m];
        }
    }
}
```

幻方

构造题用，其有一些性质（保证 N 为奇数）：1 到 N^2 每个数字恰好使用一次，且每行、每列及两条对角线上的数字之和都相同，且为奇数 See 。

构造方式：将 1 写在第一行的中间，随后不断向右上角位置填下一个数字，直到填满。



```cpp
int n;
cin >> n;
int x = 1, y = (n + 1) / 2;
vector ans(n + 1, vector<int>(n + 1));
for (int i = 1; i <= n * n; i++) {
    ans[x][y] = i;
    if (!ans[(x - 2 + n) % n + 1][y % n + 1]){
        x = (x - 2 + n) % n + 1;
        y = y % n + 1;
    } else {
        x = x % n + 1;
    }
}
```

最长严格/非严格递增子序列 (LIS)

一维

注意子序列是不连续的。使用二分搜索，以 $\mathcal{O}(N \log N)$ 复杂度通过，另也有 $\mathcal{O}(N^2)$ 的 dp 解法。

> Dilworth: 对于任意有限偏序集，其最大反链中元素的数目必等于最小链划分中链的数目.
> 将一个序列剖成若干个单调不升子序列的最小个数等于该序列最长上升子序列的个数

```cpp
vector<int> val; // 堆数
for (int i = 1, x; i <= n; i++) {
    cin >> x;
    int it = upper_bound(val.begin(), val.end(), x) - val.begin(); // low/upp: 严格/非严格递增
    if (it >= val.size()) { // 新增一堆
        val.push_back(x);
    } else { // 更新对应位置元素
        val[it] = x;
    }
}
cout << val.size() << endl;
```

二维+输出方案

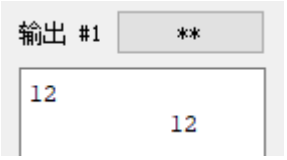
```cpp
vector<array<int, 3>> in(n + 1);
for (int i = 1; i <= n; i++) {
    cin >> in[i][0] >> in[i][1];
    in[i][2] = i;
}
sort(in.begin() + 1, in.end(), [&](auto x, auto y) {
    if (x[0] != y[0]) return x[0] < y[0];
    return x[1] > y[1];
});

vector<int> val{0}, idx{0}, pre(n + 1);
for (int i = 1; i <= n; i++) {
    auto [x, y, z] = in[i];
```

```
    int it = lower_bound(val.begin(), val.end(), y) - val.begin(); // low/upp: 严格/非严格递增
    if (it >= val.size()) { // 新增一堆
        pre[z] = idx.back();
        val.push_back(y);
        idx.push_back(z);
    } else { // 更新对应位置元素
        pre[z] = idx[it - 1];
        val[it] = y;
        idx[it] = z;
    }
}

vector<int> ans;
for (int i = idx.back(); i != 0; i = pre[i]) {
    ans.push_back(i);
}
reverse(ans.begin(), ans.end());
cout << ans.size() << "\n";
for (auto it : ans) {
    cout << it << " ";
}
```

cout 输出流控制

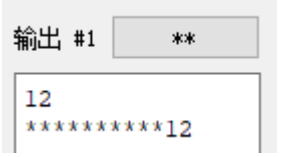
设置字段宽度：`setw(x)`，该函数可以使得补全 x 位输出，默认用空格补全。

```
bool Solve() {
    cout << 12 << endl;
    cout << setw(12) << 12 << endl;
    return 0;
}
```



设置填充字符：`setfill(x)`，该函数可以设定补全类型，注意这里的 x 只能为 `char` 类型。

```
bool Solve() {
    cout << 12 << endl;
    cout << setw(12) << setfill('*') << 12 << endl;
    return 0;
}
```



读取一行数字，个数未知

```
string s;
getline(cin, s);
stringstream ss;
ss << s;
while (ss >> s) {
    auto res = stoi(s);
    cout << res * 100 << endl;
}
```

约瑟夫问题

n 个人编号 $0,1,2\ldots,n-1$，每次数到 k 出局，求最后剩下的人的编号。

$\mathcal{O}(N)$。

```
int jos(int n,int k){
    int res=0;
    repeat(i,1,n+1)res=(res+k)%i;
    return res; // res+1，如果编号从1开始
}
```

$\mathcal{O}(K\log N)$，适用于 K 较小的情况。

```
int jos(int n,int k){
    if(n==1 || k==1)return n-1;
    if(k>n)return (jos(n-1,k)+k)%n; // 线性算法
    int res=jos(n-n/k,k)-n%k;
    if(res<0)res+=n; // mod n
    else res+=res/(k-1); // 还原位置
    return res; // res+1，如果编号从1开始
}
```

日期换算（基姆拉尔森公式）

已知年月日，求星期数。

```
int week(int y,int m,int d){
    if(m<=2)m+=12,y--;
    return (d+2*m+3*(m+1)/5+y+y/4-y/100+y/400)%7+1;
}
```

单调队列

查询区间 k 的最大最小值。

```
deque<int> D;
int n,k,x,a[MAX];
int main(){
```

```
    IOS();
    cin>>n>>k;
    for(int i=1;i<=n;i++) cin>>a[i];
    for(int i=1;i<=n;i++){
        while(!D.empty() && a[D.back()]<=a[i]) D.pop_back();
        D.emplace_back(i);
        if(!D.empty()) if(i-D.front()>=k) D.pop_front();
        if(i>=k)cout<<a[D.front()]<<endl;
    }
    return 0;
}
```

高精度快速幂

求解 $n^k \bmod p$，其中 $0 \leq n, k \leq 10^{1000000}$，$1 \leq p \leq 10^9$。容易发现 n 可以直接取模，瓶颈在于 k See。

魔改十进制快速幂（暴力计算）

该算法复杂度 $\mathcal{O}(\texttt{len}(k))$。

```
int mypow10(int n, vector<int> k, int p) {
    int r = 1;
    for (int i = k.size() - 1; i >= 0; i--) {
        for (int j = 1; j <= k[i]; j++) {
            r = r * n % p;
        }
        int v = 1;
        for (int j = 0; j <= 9; j++) {
            v = v * n % p;
        }
        n = v;
    }
    return r;
}
signed main() {
    string n_, k_;
    int p;
    cin >> n_ >> k_ >> p;

    int n = 0; // 转化并计算 n % p
    for (auto it : n_) {
        n = n * 10 + it - '0';
        n %= p;
    }
    vector<int> k; // 转化 k
    for (auto it : k_) {
        k.push_back(it - '0');
    }
    cout << mypow10(n, k, p) << endl; // 暴力快速幂
}
```

扩展欧拉定理（欧拉降幂公式）

$$n^k \equiv \begin{cases} n^{k \bmod \varphi(p)} & \gcd(n,p)=1 \\ n^{k \bmod \varphi(p)+\varphi(p)} & \gcd(n,p)\neq 1 \wedge k \geq \varphi(p) \\ n^k & \gcd(n,p)\neq 1 \wedge k < \varphi(p) \end{cases}$$

最终我们可以将幂降到 $\varphi(p)$ 的级别，使得能够直接使用快速幂解题，复杂度瓶颈在求解欧拉函数 $\mathcal{O}(\sqrt{p})$。

```cpp
int phi(int n) { //求解 phi(n)
    int ans = n;
    for (int i = 2; i <= n / i; i++) {
        if (n % i == 0) {
            ans = ans / i * (i - 1);
            while (n % i == 0) {
                n /= i;
            }
        }
    }
    if (n > 1) { //特判 n 为质数的情况
        ans = ans / n * (n - 1);
    }
    return ans;
}
signed main() {
    string n_, k_;
    int p;
    cin >> n_ >> k_ >> p;

    int n = 0; // 转化并计算 n % p
    for (auto it : n_) {
        n = n * 10 + it - '0';
        n %= p;
    }
    int mul = phi(p), type = 0, k = 0; // 转化 k
    for (auto it : k_) {
        k = k * 10 + it - '0';
        type |= (k >= mul);
        k %= mul;
    }
    if (type) {
        k += mul;
    }
    cout << mypow(n, k, p) << endl;
}
```

快读

注意读入到文件结尾才结束，直接运行会无输出。

```cpp
char buf[1 << 21], *p1 = buf, *p2 = buf;
inline char getc() {
```

```
    return p1 == p2 && (p2 = (p1 = buf) + fread(buf, 1, 1 << 21, stdin), p1 == p2) ? 0
: *p1++;
}
template<typename T> void Cin(T &a) {
    T ans = 0;
    bool f = 0;
    char c = getc();
    for (; c < '0' || c > '9'; c = getc()) {
        if (c == '-') f = -1;
    }
    for (; c >= '0' && c <= '9'; c = getc()) {
        ans = ans * 10 + c - '0';
    }
    a = f ? -ans : ans;
}
template<typename T, typename... Args> void Cin(T &a, Args &...args) {
    Cin(a), Cin(args...);
}
template<typename T> void Cout(T x) { // 注意，这里输出不带换行
    if (x < 0) putchar('-'), x = -x;
    if (x > 9) Cout(x / 10);
    putchar(x % 10 + '0');
}
```

int128 输入输出流控制

int128 只在基于 $linux$ 系统的环境下可用，需要 C++20 。38位精度，除输入输出外与普通数据类型无差别。该封装支持负数读入，需要注意 `write` 函数结尾不输出多余空格与换行。

```
namespace my128 { // 读入优化封装，支持__int128
    using i64 = __int128_t;
    i64 abs(const i64 &x) {
        return x > 0 ? x : -x;
    }
    auto &operator>>(istream &it, i64 &j) {
        string val;
        it >> val;
        reverse(val.begin(), val.end());
        i64 ans = 0;
        bool f = 0;
        char c = val.back();
        val.pop_back();
        for (; c < '0' || c > '9'; c = val.back(), val.pop_back()) {
            if (c == '-') {
                f = 1;
            }
        }
        for (; c >= '0' && c <= '9'; c = val.back(), val.pop_back()) {
            ans = ans * 10 + c - '0';
        }
        j = f ? -ans : ans;
        return it;
```

```
    }
    auto &operator<<(ostream &os, const i64 &j) {
        string ans;
        function<void(i64)> write = [&](i64 x) {
            if (x < 0) ans += '-', x = -x;
            if (x > 9) write(x / 10);
            ans += x % 10 + '0';
        };
        write(j);
        return os << ans;
    }
} // namespace my128
```

对拍板子

- 文件控制

```
// BAD.cpp, 存放待寻找错误的代码
freopen("A.txt","r",stdin);
freopen("BAD.out","w",stdout);

// 1.cpp, 存放暴力或正确的代码
freopen("A.txt","r",stdin);
freopen("1.out","w",stdout);

// Ask.cpp
freopen("A.txt", "w", stdout);
```

- C++版 bat

```
int main() {
    int T = 1E5;
    while(T--) {
        system("BAD.exe");
        system("1.exe");
        system("A.exe");
        if (system("fc BAD.out 1.out")) {
            puts("WA");
            return 0;
        }
    }
}
```

在$linux$中将$diff$换成fc

```
#include <stdio.h>
#include <stdlib.h>

int main() {
    // For Windows
    // 对拍时不开文件输入输出
```

```
    // 当然，这段程序也可以改写成批处理的形式
    while (true) {
        system("data > test.in");  // 数据生成器将生成数据写入输入文件
        system("solve < test.in > solve.out");  // 获取程序1输出
        system("std < test.in > std.out");  // 获取程序2输出
        if (system("diff solve.out std.out")) {
            // 该行语句比对输入输出
            // fc返回0时表示输出一致，否则表示有不同处
            system("pause");  // 方便查看不同处
            return 0;
            // 该输入数据已经存放在test.in文件中，可以直接利用进行调试
        }
    }
}
```

```
import os
tc = 0
os.system("g++ ./std.cpp -o ./std")
os.system("g++ ./solve.cpp -o ./solve")

while True:
    os.system("python ./data.py > ./data.in")
    os.system("./std < ./data.in > ./std.out")
    os.system("./solve < ./data.in > ./solve.out");
    if(os.system("diff ./std.out ./solve.out")):
        print("WA")
        exit(0)
    else:
        tc += 1
        print("AC #%d" %(tc))
```

随机数生成与样例构造

```

mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());
int r(int a, int b) {
    return rnd() % (b - a + 1) + a;
}

void graph(int n, int root = -1, int m = -1) {
    vector<pair<int, int>> t;
    for (int i = 1; i < n; i++) { // 先建立一棵以0为根节点的树
        t.emplace_back(i, r(0, i - 1));
    }

    vector<pair<int, int>> edge;
    set<pair<int, int>> uni;
    if (root == -1) root = r(0, n - 1); // 确定根节点
    for (auto [x, y] : t) { // 偏移建树
        x = (x + root) % n + 1;
        y = (y + root) % n + 1;
        edge.emplace_back(x, y);
```

```
        uni.emplace(x, y);
    }

    if (m != -1) { // 如果是图，则在树的基础上继续加边
        for (int i = n; i <= m; i++) {
            while (true) {
                int x = r(1, n), y = r(1, n);
                if (x == y) continue; // 拒绝自环
                if (uni.count({x, y})) continue; // 拒绝重边
                edge.emplace_back(x, y);
                uni.emplace(x, y);
            }
        }
    }

    random_shuffle(edge.begin(), edge.end()); // 打乱节点
    for (auto [x, y] : edge) {
        cout << x << " " << y << endl;
    }
}
```

手工哈希

```
struct myhash {
    static uint64_t hash(uint64_t x) {
        x += 0x9e3779b97f4a7c15;
        x = (x ^ (x >> 30)) * 0xbf58476d1ce4e5b9;
        x = (x ^ (x >> 27)) * 0x94d049bb133111eb;
        return x ^ (x >> 31);
    }
    size_t operator()(uint64_t x) const {
        static const uint64_t SEED =
chrono::steady_clock::now().time_since_epoch().count();
        return hash(x + SEED);
    }
    size_t operator()(pair<uint64_t, uint64_t> x) const {
        static const uint64_t SEED =
chrono::steady_clock::now().time_since_epoch().count();
        return hash(x.first + SEED) ^ (hash(x.second + SEED) >> 1);
    }
};
// unordered_map<int, int, myhash>
```

Python常用语法

读入与定义

- 读入多个变量并转换类型：`X, Y = map(int, input().split())`
- 读入列表：`X = eval(input())`
- 多维数组定义：`X = [[0 for j in range(0, 100)] for i in range(0, 200)]`

格式化输出

- 保留小数输出：`print("{:.12f}".format(X))` 指保留 12 位小数
- 对齐与宽度：`print("{:<12f}".format(X))` 指左对齐，保留 12 个宽度

排序

- 倒序排序：使用 `reverse` 实现倒序 `X.sort(reverse=True)`
- 自定义排序：下方代码实现了先按第一关键字降序、再按第二关键字升序排序。

```
X.sort(key=lambda x: x[1])
X.sort(key=lambda x: x[0], reverse=True)
```

文件IO

- 打开要读取的文件：`r = open('X.txt', 'r', encoding='utf-8')`
- 打开要写入的文件：`w = open('Y.txt', 'w', encoding='utf-8')`
- 按行写入：`w.write(XX)`

增加输出流长度、递归深度

```
import sys
sys.set_int_max_str_digits(200000)
sys.setrecursionlimit(100000)
```

自定义结构体

自定义结构体并且自定义排序

```
class node:
    def __init__(self, A, B, C):
        self.A = A
        self.B = B
        self.C = C

w = []
for i in range(1, 5):
    a, b, c = input().split()
    w.append(node(a, b, c))
w.sort(key=lambda x: x.C, reverse=True)
for i in w:
    print(i.A, i.B, i.C)

```

数据结构

- 模拟于 C_{++}^{map} ，定义：`dic = dict()`
- 模拟栈与队列：使用常见的 `list` 即可完成，`list.insert(0, X)` 实现头部插入、`list.pop()` 实现尾部弹出、`list.pop(0)` 实现头部弹出

其他

- 获取ASCII码：`ord()` 函数
- 转换为ASCII字符：`chr()` 函数

OJ测试

对于一个未知属性的OJ，应当在正式赛前进行以下全部测试：

GNU C++ 版本测试

```cpp
for (int i : {1, 2}) {} // GNU C++11 支持范围表达式

auto cc = [&](int x) { x++; }; // GNU C++11 支持 auto 与 lambda 表达式
cc(2);

tuple<string, int, int> V; // GNU C++11 引入
array<int, 3> C; // GNU C++11 引入

auto dfs = [&](auto self, int x) -> void { // GNU C++14 支持 auto 自递归
    if (x > 10) return;
    self(self, x + 1);
};
dfs(dfs, 1);

vector in(1, vector<int>(1)); // GNU C++17 支持 vector 模板类型缺失

map<int, int> dic;
for (auto [u, v] : dic) {} // GNU C++17 支持 auto 解绑
dic.contains(12); // GNU C++20 支持 contains 函数

constexpr double Pi = numbers::pi; // C++20 支持
```

编译器位数测试

```cpp
using i64 = __int128; // 64 位 GNU C++11 支持
```

评测器环境测试

Windows 系统输出 -1 ；反之则为一个随机数。

```cpp
#define int long long
map<int, int> dic;
int x = dic.size() - 1;
cout << x << endl;
```

运算速度测试

	本地-20(64)	CodeForces-20(64)	AtCoder-20(64)	牛客-17(64)	学院OJ	CodeForces-17(32)	马蹄集
4E3量级-硬跑	2454	2886	874	4121	4807	2854	4986
4E3量级-手动加速	556	686	873	1716	1982	2246	2119

```cpp
// #pragma GCC optimize("Ofast", "unroll-loops")
#include <bits/stdc++.h>
using namespace std;

signed main() {
    int n = 4E3, cnt = 0;
    bitset<30> ans;
    for (int i = 1; i <= n; i++) {
        for (int j = 1; j <= n; j += 2) {
            for (int k = 1; k <= n; k += 4) {
                ans |= i | j | k;
                cnt++;
            }
        }
    }
    cout << cnt << "\n";
}
```

```cpp
// #pragma GCC optimize("Ofast", "unroll-loops")

#include <bits/stdc++.h>
using namespace std;
mt19937 rnd(chrono::steady_clock::now().time_since_epoch().count());

signed main() {
    size_t n = 340000000, seed = 0;
    for (int i = 1; i <= n; i++) {
        seed ^= rnd();
    }

    return 0;
}
```

编译器设置

```
g++ -O2 -std=c++20 -pipe
-Wall -Wextra -Wconversion /* 这部分是警告相关，可能用不到 */
-fstack-protector
-Wl,--stack=268435456
```

/END/