目录

1. 博弈论
2. Wythoff(黄金分割，两堆石头，可以一起拿一样的)
3. Bash(石头谁先拿到最后一个)
4. nim游戏，N堆石子，每次可全拿
5. sg函数
6. 数论
7. 排列组合
   1. 一般组合

|  |
| --- |
| int n, m; // 从n个数中选出m个构成组合 |
| int rcd[MAX\_N]; // 记录每个位置填的数 |
| int num[MAX\_N]; // 存放输入的n个数 |
|  |
| void select\_combination(int l, int p) |
| { |
| int i; |
| if (l == m) // 若选出了m个数, 则打印 |
| { |
| for (i = 0; i < m; i++) |
| { |
| printf("%d", rcd[i]); |
| if (i < m - 1) |
| { |
| printf(" "); |
| } |
| } |
| printf("\n"); |
| return ; |
| } |
| for (i = p; i < n; i++) // 上个位置填的是num[p-1],本次从num[p]开始试探 |
| { rcd[l] = num[i]; // 在l位置放上该数 |
| select\_combination(l + 1, i + 1); } // 填下一个位置 |
| } |
| int read\_data() |
| { |
| int i; |
| if (scanf("%d%d", &n, &m) == EOF) return 0; |
| for (i = 0; i < n; i++) scanf("%d", &num[i]); |
| return 1; |
| } |

* 1. 不重复排列

|  |
| --- |
| using namespace std;  typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
| int n, m; // 共有n个数,其中互不相同的有m个 |
| int rcd[MAX\_N]; // 记录每个位置填的数 |
| int used[MAX\_N]; // 标记m个数可以使用的次数 |
| int num[MAX\_N]; // 存放输入中互不相同的m个数 |
|  |
| void unrepeat\_permutation(int l) |
| { |
| int i; |
| if (l == n) // 填完了n个数,则输出 |
| { |
| for (i = 0; i < n; i++) |
| { |
| printf("%d", rcd[i]); |
| if (i < n - 1) |
| { |
| printf(" "); |
| } |
| } |
| printf("\n"); |
| return ; |
| } |
| for (i = 0; i < m; i++) // 枚举m个本质不同的数 |
| { |
| if (used[i] > 0) // 若数num[i]还没被用完,则可使用次数减 |
| { |
| used[i]--; |
| rcd[l] = num[i]; // 在l位置放上该数 |
| unrepeat\_permutation(l+1); // 填下一个位置 |
| used[i]++; // 可使用次数恢复 |
| } |
| } |
| } |
|  |
| int read\_data() |
| { |
| int i, j, val; |
| if (scanf("%d", &n) == EOF) |
| { |
| return 0; |
| } |
| m = 0; |
| for (i = 0; i < n; i++) |
| { |
| scanf("%d", &val); |
| for (j = 0; j < m; j++) |
| { |
| if (num[j] == val) |
| { |
| used[j]++; break; |
| } |
| } |
| if (j == m) |
| { |
| num[m] = val; |
| used[m++] = 1; |
| } |
| } |
| return 1; |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |
|  |

* 1. 不重复组合

|  |
| --- |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
|  |
| int n, m; // 输入n个数,其中本质不同的有m个 |
| int rcd[MAX\_N]; // 记录每个位置填的数 |
| int used[MAX\_N]; // 标记m个数可以使用的次数 |
| int num[MAX\_N]; // 存放输入中本质不同的m个数 |
|  |
| void unrepeat\_combination(int l, int p) |
| { |
| int i; |
| for (i = 0; i < l; i++) // 每次都输出 |
| { |
| printf("%d", rcd[i]); |
| if (i < l - 1) |
| { |
| printf(" "); |
| } |
| } |
| printf("\n"); |
| for (i = p; i < m; i++) // 循环依旧从p开始,枚举剩下的本质不同的数 |
| { |
| if (used[i] > 0) // 若还可以用, 则可用次数减 |
| { |
| used[i]--; |
| rcd[l] = num[i]; // 在l位置放上该 |
| unrepeat\_combination(l+1, i); // 填下一个位置 |
| used[i]++; //可用次数恢复 |
| } |
| } |
| } |
|  |
| int read\_data() |
| { |
| int i, j, val; |
| if (scanf("%d", &n) == EOF) |
| { |
| return 0; |
| } |
| m = 0; |
| for (i = 0; i < n; i++) |
| { |
| scanf("%d", &val); |
| for (j = 0; j < m; j++) |
| { |
| if (num[j] == val) |
| { |
| used[j]++; |
| break; |
| } |
| } |
| if (j == m) |
| { |
| num[m] = val; |
| used[m++] = 1; |
| } |
| } |
| return 1; |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

* 1. 全排列枚举

|  |
| --- |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
|  |
| int n; // 共n个数 |
| int rcd[MAX\_N]; // 记录每个位置填的数 |
| int used[MAX\_N]; // 标记数是否用过 |
| int num[MAX\_N]; // 存放输入的n个数 |
|  |
| void full\_permutation(int l) |
| { |
| int i; |
| if (l == n) |
| { |
| for (i = 0; i < n; i++) |
| { |
| printf("%d", rcd[i]); |
| if (i < n-1) |
| { |
| printf(" "); |
| } |
| } |
| printf("\n"); |
| return ; |
| } |
| for (i = 0; i < n; i++) // 枚举所有的数(n个),循环从开始 |
| if (!used[i]) |
| { // 若num[i]没有使用过, 则标记为已使用 |
| used[i] = 1; |
| rcd[l] = num[i]; // 在l位置放上该数 |
| full\_permutation(l+1); // 填下一个位置 |
| used[i] = 0; // 清标记 |
| } |
| } |
|  |
| int read\_data() |
| { |
| int i; |
| if (scanf("%d", &n) == EOF) |
| { |
| return 0; |
| } |
| for (i = 0; i < n; i++) |
| { |
| scanf("%d", &num[i]); |
| } |
| for (i = 0; i < n; i++) |
| { |
| used[i] = 0; |
| } |
| return 1; |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

* 1. 全组合

|  |
| --- |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
|  |
| int n; // 共n个数 |
| int rcd[MAX\_N]; // 记录每个位置填的数 |
| int num[MAX\_N]; // 存放输入的n个数 |
|  |
| void full\_combination(int l, int p) |
| { |
| int i; |
| for (i = 0; i < l; i++) // 每次进入递归函数都输出 |
| { |
| printf("%d", rcd[i]); |
| if (i < l-1) |
| { |
| printf(" "); |
| } |
| } |
| printf("\n"); |
| for (i = p; i < n; i++) // 循环同样从p开始,但结束条件变为i>=n |
| { |
| rcd[l] = num[i]; // 在l位置放上该数 |
| full\_combination(l + 1, i + 1); // 填下一个位置 |
| } |
| } |
|  |
| int read\_data() |
| { |
| int i; |
| if (scanf("%d", &n) == EOF) |
| { |
| return 0; |
| } |
| for (i = 0; i < n; i++) |
| { |
| scanf("%d", &num[i]); |
| } |
| return 1; |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

* 1. 类循环排列

|  |
| --- |
|  |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
| int n, m; // 相当于n重循环,每重循环长度为m |
| int rcd[MAX\_N]; // 记录每个位置填的数 |
| void loop\_permutation(int l) |
| { |
| int i; |
| if (l == n) // 相当于进入了 n 重循环的最内层 |
| { |
| for (i = 0; i < n; i++) |
| { |
| cout << rcd[i]; |
| if (i < n-1) |
| { |
| cout << " "; |
| } |
| } |
| cout << "\n"; |
| return ; |
| } |
| for (i = 0; i < m; i++) // 每重循环长度为m |
| { |
| rcd[l] = i; // 在l位置放i |
| loop\_permutation(l + 1); // 填下一个位置 |
| } |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

* 1. 计数方法

|  |
| --- |
| C(n,k)+C(n,k+1) = c(n+1,k+1) |
| C(n,k+1)=C(n,k)\*(n-k)/(k+1) |

1. exgcd&&求逆元

|  |
| --- |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
| /\* |
| \* 扩展欧几里得法（求ax + by = gcd） |
| \*/ |
| // 返回d = gcd(a, b);和对应于等式ax + by = d中的x、y |
| long long extendGcd(long long a, long long b, long long &x, long long &y) |
| { |
| if (a == 0 && b == 0) |
| { |
| return -1; // 无最大公约数 |
| } |
| if (b == 0) |
| { |
| x = 1; |
| y = 0; |
| return a; |
| } |
| long long d = extendGcd(b, a % b, y, x); |
| y -= a / b \* x; |
| return d; |
| } |
|  |
| // 求逆元 ax = 1(mod n) |
| long long modReverse(long long a, long long n) |
| { |
| long long x, y; |
| long long d = extendGcd(a, n, x, y); |
| if (d == 1) |
| { |
| return (x % n + n) % n; |
| } |
| else |
| { |
| return -1; // 无逆元 |
| } |
| } |
| /\* |
| \* 简洁写法I |
| \* 只能求a < m的情况，且a与m互质 |
| \* 求ax = 1(mod m)的x值，即逆元(0 < a < m) |
| \*/ |
| long long inv(long long a, long long m) |
| { |
| if (a == 1) |
| { |
| return 1; |
| } |
| return inv(m % a, m) \* (m - m / a) % m; |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

1. gcd && exgcd

|  |
| --- |
| using namespace std; |
| int gcd(int x, int y) |
| { |
| if (!x || !y) |
| { |
| return x > y ? x : y; |
| } |
| for (int t; t = x % y, t; x = y, y = t) ; |
| return y; |
| } |
|  |
| /\* |
| \* Çóx£¬yÊ¹µÃgcd(a, b) = a \* x + b \* y; |
| \*/ |
| int extgcd(int a, int b, int &x, int &y) |
| { |
| if (b == 0) |
| { |
| x = 1; |
| y = 0; |
| return a; |
| } |
|  |
| int d = extgcd(b, a % b, x, y); |
| int t = x; |
| x = y; |
| y = t - a / b \* y; |
|  |
| return d; |
| } |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |
|  |

1. 判断连续素数

|  |
| --- |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
| /\* |
| \* 素数筛选，查找出小于等于MAXN的素数 |
| \* prime[0]存素数的个数 |
| \*/ |
|  |
| const int MAXN = 100000; |
| int prime[MAXN + 1]; |
|  |
| void getPrime() |
| { |
| memset(prime, 0, sizeof(prime)); |
| for (int i = 2; i <= MAXN; i++) |
| { |
| if (!prime[i]) |
| { |
| prime[++prime[0]] = i; |
| } |
| for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++) |
| { |
| prime[prime[j] \* i] = 1; |
| if (i % prime[j] == 0) |
| { |
| break; |
| } |
| } |
| } |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

1. 合数的分解

|  |
| --- |
|  |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
|  |
| /\* |
| \* 合数的分解需要先进行素数的筛选 |
| \* factor[i][0]存放分解的素数 |
| \* factor[i][1]存放对应素数出现的次数 |
| \* fatCnt存放合数分解出的素数个数(相同的素数只算一次) |
| \*/ |
|  |
| const int MAXN = 10000; |
|  |
| int prime[MAXN + 1]; |
|  |
| // 获取素数 |
| void getPrime() |
| { |
| memset(prime, 0, sizeof(prime)); |
| for (int i = 2; i <= MAXN; i++) |
| { |
| if (!prime[i]) |
| { |
| prime[++prime[0]] = i; |
| } |
| for (int j = 1; j <= prime[0] && prime[j] <= MAXN / i; j++) |
| { |
| prime[prime[j] \* i] = 1; |
| if (i % prime[j] == 0) |
| { |
| break; |
| } |
| } |
| } |
| return ; |
| } |
|  |
| long long factor[100][2]; |
| int fatCnt; |
|  |
| // 合数分解 |
| int getFactors(long long x) |
| { |
| fatCnt = 0; |
| long long tmp = x; |
| for (int i = 1; prime[i] <= tmp / prime[i]; i++) |
| { |
| factor[fatCnt][1] = 0; |
| if (tmp % prime[i] == 0) |
| { |
| factor[fatCnt][0] = prime[i]; |
| while (tmp % prime[i] == 0) |
| { |
| factor[fatCnt][1]++; |
| tmp /= prime[i]; |
| } |
| fatCnt++; |
| } |
| } |
| if (tmp != 1) |
| { |
| factor[fatCnt][0] = tmp; |
| factor[fatCnt++][1] = 1; |
| } |
| return fatCnt; |
| } |
|  |
| int main(){ |
| ll n ; |
| readll(n); |
| getPrime(); // 得到质数表 |
| fatCnt = getFactors(n); |
| rep(i , fatCnt){ |
| cout << factor[i][0] << " "; |
| } |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

1. 快速幂取模

|  |
| --- |
|  |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
|  |
| /\* |
| \* 欧拉函数法 |
| \* a 和 m 互质 |
| \*/ |
| // 快速幂取模 |
| long long powM(long long a, long long b, long long m) |
| { |
| long long tmp = 1; |
| if (b == 0) |
| { |
| return 1; |
| } |
| if (b == 1) |
| { |
| return a % m; |
| } |
|  |
| tmp = powM(a, b >> 1, m); |
| tmp = tmp \* tmp % m; |
|  |
| if (b & 1) |
| { |
| tmp = tmp \* a % m; |
| } |
|  |
| return tmp; |
| } |
|  |
| long long inv(long long a, long long m) |
| { |
| return powM(a, m - 2, m); |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

1. 最大1矩阵(01组成的全是1的最大子矩阵)

|  |
| --- |
|  |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
| const int N = 1000; |
|  |
| bool a[N][N]; |
|  |
| int Run(const int &m, const int &n) // a[1...m][1...n] |
| { // O(m\*n) |
| int i, j, k, l, r, max=0; |
| int col[N]; |
| for (j = 1; j <= n; j++) |
| { |
| if (a[1][j] == 0 ) |
| { |
| col[j] = 0; |
| } |
| else |
| { |
| for (k = 2; k <= m && a[k][j] == 1; k++); |
| col[j] = k - 1; |
| } |
| } |
| for (i = 1; i <= m; i++) |
| { |
| if (i > 1) |
| { |
| for (j = 1; j <= n; j++) |
| { |
| if (a[i][j] == 0) |
| { |
| col[j] = 0; |
| } |
| else |
| { |
| if (a[i - 1][j] == 0) |
| { |
| for (k = i + 1; k <= m && a[k][j] == 1; k++); |
| col[j] = k-1; |
| } |
| } |
| } |
| } |
| for (j = 1; j <= n; j++) |
| { |
| if (col[j] >= i) |
| { |
| for (l = j - 1; l > 0 && col[l] >= col[j]; --l); |
| l++; |
| for (r = j + 1; r <= n && col[r] >= col[j]; ++r); |
| r--; |
| int res = (r - l + 1) \* (col[j] - i + 1); |
| if( res > max ) |
| { |
| max = res; |
| } |
| } |
| } |
| } |
| return max; |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

1. 模线性方程

|  |
| --- |
|  |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
|  |
| /\* |
| \* 模线性方程 a \* x = b (% n) |
| \*/ |
| void modeq(int a, int b, int n) |
| { |
| int e, i, d, x, y; |
| d = extgcd(b, a % b, x, y); |
| if (b % d > 0) |
| { |
| cout << "No answer!\n"; |
| } |
| else |
| { |
| e = (x \* (b / d)) % n; |
| for (i = 0; i < d; i++) |
| { |
| cout << i + 1 << "-th ans:" << (e + i \* (n / d)) % n << '\n'; |
| } |
| } |
| return ; |
| } |
|  |
| /\* |
| \* 模线性方程组 |
| \* a = B[1](% W[1]); a = B[2](% W[2]); ... a = B[k](% W[k]); |
| \* 其中W，B已知，W[i] > 0且W[i]与W[j]互质，求a（中国剩余定理） |
| \*/ |
|  |
| int china(int b[], int w[], int k) |
| { |
| int i, d, x, y, m, a = 0, n = 1; |
| for (i = 0; i < k; i++) |
| { |
| n \*= w[i]; // 注意不能overflow |
| } |
| for (i = 0; i < k; i++) |
| { |
| m = n / w[i]; |
| d = extgcd(w[i], m, x, y); |
| a = (a + y \* m \* b[i]) % n; |
| } |
| if (a > 0) |
| { |
| return a; |
| } |
| else |
| { |
| return (a + n);//最小正解 |
| } |
| } |
|  |
|  |
| /\* |
| w[i]和w[j]不互质 |
| \*/ |
|  |
| typedef long long ll; |
|  |
| const int MAXN = 11; |
|  |
| int n, m; |
| int a[MAXN], b[MAXN]; |
|  |
| int main(int argc, const char \* argv[]) |
| { |
| int T; |
| cin >> T; |
|  |
| while (T--) |
| { |
| cin >> n >> m; |
| for (int i = 0; i < m; i++) |
| { |
| cin >> a[i]; |
| } |
| for (int i = 0; i < m; i++) |
| { |
| cin >> b[i]; |
| } |
|  |
| ll ax = a[0], bx = b[0], x, y; |
| int flag = 0; |
| for (int i = 1; i < m; i++) |
| { |
| ll d = extgcd(ax, a[i], x, y); |
| if ((b[i] - bx) % d != 0) |
| { |
| flag = 1; // 无整数解 |
| break; |
| } |
|  |
| ll tmp = a[i] / d; |
| x = x \* (b[i] - bx) / d; // 约分 |
| x = (x % tmp + tmp) % tmp; |
| bx = bx + ax \* x; |
| ax = ax \* tmp; // ax = ax \* a[i] / d |
| } |
|  |
| if (flag == 1 || n < bx) |
| { |
| puts("0"); |
| } |
| else |
| { |
| ll ans = (n - bx) / ax + 1; |
| if (bx == 0) |
| { |
| ans--; |
| } |
| printf("%lld\n", ans); |
| } |
| } |
|  |
| return 0; |
| } |

1. 欧拉函数单独求解

|  |
| --- |
|  |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
|  |
| int IT\_MAX = 1 << 19; |
| int MOD = 1000000007; |
| const int INF = 0x3f3f3f3f; |
| const ll LL\_INF = 0x3f3f3f3f3f3f3f3f; |
| const db PI = acos(-1); |
| const db ERR = 1e-10; |
| const int MAX\_N = 100005; |
| bool cmp (int a , int b){ |
| return a>b; |
| } |
|  |
| unsigned euler(unsigned x); //小于等于x中的数与x互质的个数 |
| int main(){ |
| int n ; |
| read(n); |
| cout << euler(n) << endl; |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| return 0; |
| } |
| /\* |
| \* 单独求解的本质是公式的应用 |
| \*/ |
| unsigned euler(unsigned x) |
| { |
| unsigned i, res = x; // unsigned == unsigned int |
| for (i = 2; i < (int)sqrt(x \* 1.0) + 1; i++) |
| { |
| if (!(x % i)) |
| { |
| res = res / i \* (i - 1); |
| while (!(x % i)) |
| { |
| x /= i; // 保证i一定是素数 |
| } |
| } |
| } |
| if (x > 1) |
| { |
| res = res / x \* (x - 1); |
| } |
| return res; |
| } |

1. 欧拉线性筛

|  |
| --- |
| /\*  \* 同时得到欧拉函数和素数表 |
| \*/ |
| using namespace std; |
| const int MAXN = 10000000; |
|  |
| bool check[MAXN + 10]; |
| int phi[MAXN + 10]; |
| int prime[MAXN + 10]; //tot个素数 |
| int tot; // 素数个数 |
| void phi\_and\_prime\_table(int N); // N以内的素数 |
| int main(){ |
| int n; |
| scanf("%d",&n); |
| phi\_and\_prime\_table(n); |
| for(int i = 0 ; i < tot ; i ++){ |
| cout << prime[i] << " "; |
| } |
| cout << endl << tot << endl; |
| return 0; |
| } |
|  |
|  |
|  |
|  |
| void phi\_and\_prime\_table(int N) |
| { |
| memset(check, false, sizeof(check)); |
| phi[1] = 1; |
| tot = 0; |
| for (int i = 2; i <= N; i++) |
| { |
| if (!check[i]) |
| { |
| prime[tot++] = i; |
| phi[i] = i - 1; |
| } |
| for (int j = 0; j < tot; j++) |
| { |
| if (i \* prime[j] > N) |
| { |
| break; |
| } |
| check[i \* prime[j]] = true; |
| if (i % prime[j] == 0) |
| { |
| phi[i \* prime[j]] = phi[i] \* prime[j]; |
| break; |
| } |
| else |
| { |
| phi[i \* prime[j]] = phi[i] \* (prime[j] - 1); |
| } |
| } |
| } |
| return ; |
| } |

1. 求阶乘逆元

|  |
| --- |
| using namespace std; |
| typedef long long ll; |
| typedef unsigned long long ull; |
| typedef double db; |
| typedef pair<int , int> p\_ii; |
|  |
| const ll MOD = 1e9 + 7; // 必须为质数才管用 |
| const ll MAXN = 1e5 + 3; |
|  |
| ll fac[MAXN]; // 阶乘 |
| ll inv[MAXN]; // 阶乘的逆元 |
|  |
| ll QPow(ll x, ll n) |
| { |
| ll ret = 1; |
| ll tmp = x % MOD; |
|  |
| while (n) |
| { |
| if (n & 1) |
| { |
| ret = (ret \* tmp) % MOD; |
| } |
| tmp = tmp \* tmp % MOD; |
| n >>= 1; |
| } |
|  |
| return ret; |
| } |
|  |
| void init() |
| { |
| fac[0] = 1; |
| for (int i = 1; i < MAXN; i++) |
| { |
| fac[i] = fac[i - 1] \* i % MOD; |
| } |
| inv[MAXN - 1] = QPow(fac[MAXN - 1], MOD - 2); |
| for (int i = MAXN - 2; i >= 0; i--) |
| { |
| inv[i] = inv[i + 1] \* (i + 1) % MOD; |
| } |
| } |
|  |
| int main(){ |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
|  |
| } |

1. 波利亚计数(不同颜色的珠子组成项链)
2. 简易素数筛
3. 约瑟夫环
4. 阶乘位数
5. 阶乘最后一个非零位
6. 随机大素数测试+大素数快速分解
7. 高斯消元求1类开关问题的解
8. 树/二叉树
   * 1. 中序遍历+后序遍历建二叉树
     2. 二叉树建树