目录

1. 博弈论
2. Wythoff(黄金分割，两堆石头，可以一起拿一样的)

|  |
| --- |
| /\* |
| 有2堆石子。A B两个人轮流拿，A先拿。每次可以从一堆中取任意个或从 |
| 2堆中取相同数量的石子，但不可不取。拿到最后1颗石子的人获胜。假设 |
| A B都非常聪明，拿石子的过程中不会出现失误。给出2堆石子的数量，问 |
| 最后谁能赢得比赛。 |
| \*/ |
| int main() |
| { |
| int t, a, b, m, k; |
| scanf("%d", &t); |
| while (t--) |
| { |
| scanf("%d%d", &a, &b); |
| if (a > b) |
| { |
| a ^= b; |
| b ^= a; |
| a ^= b; |
| } |
| m = b - a; |
| k = (int)(m \* (1 + sqrt(5)) / 2.0); |
| //m = ? \* a |
| //k = m / ? |
| //?:黄金分割数 |
| //如果a == k，则为后手赢，否则先手赢（奇异局） |
| printf("%s\n", a == k ? "B" : "A"); |
| } |
| return 0; |
| } |

1. Bash(石头谁先拿到最后一个)

|  |
| --- |
| /\* |
| 有一堆石子共有N个。A B两个人轮流拿，A先拿。每次最少拿1颗， |
| 最多拿K颗，拿到最后1颗石子的人获胜。假设A B都非常聪明， |
| 拿石子的过程中不会出现失误。给出N和K，问最后谁能赢得比赛。 |
| \*/ |
| /\* bashgame \*/ |
| int bash(int N, int K) |
| { |
| if (N % (K + 1) == 0) return 2; |
| return 1; |
| } |

1. nim游戏，N堆石子，每次可全拿

|  |
| --- |
| /\*有N堆石子。A B两个人轮流拿，A先拿。每次只能从一堆中取若干个， |
| 可将一堆全取走，但不可不取，拿到最后1颗石子的人获胜。假设A |
| B都非常聪明，拿石子的过程中不会出现失误。给出N及每堆石子的数量， |
| 问最后谁能赢得比赛。\*/ |
| /\*Nim 游戏\*/ |
| int main(int argc, const char \* argv[]) |
| { |
| int N, stone, tag = 0; |
| scanf("%d", &N); |
| while (N--){ |
| scanf("%d", &stone); |
| tag ^= stone; |
| } |
| //tag为0则为后手赢，否则为先手赢 |
| printf("%c\n", tag == 0 ? 'B' : 'A'); |
| return 0; |
| } |

1. sg函数

|  |
| --- |
| const int MAX\_DIG = 64; |
| //f[]：可以取走的石子个数 |
| //sg[]:0~n的SG函数值 |
| //hash[]:mex{} |
| int f[MAX\_DIG]={1,3},sg[MAX\_DIG],hash[MAX\_DIG]; |
| int k;//k是f[]的有效长度 |
| void getSG(int n) |
| { |
| memset(sg,0,sizeof(sg)); |
| for(int i=1; i<=n; i++) { |
| memset(hash,0,sizeof(hash)); |
| for(int j=0; f[j]<=i && j < k; j++) //k是f[]的有效长度 |
| hash[sg[i-f[j]]]=1; |
| for(int j=0; ; j++) { //求mes{}中未出现的最小的非负整数 |
| if(hash[j]==0) { |
| sg[i]=j; |
| break;} |
| } |
| } |
| } |
| int main(){ |
| int n; |
| read(n); |
| k=2; |
| getSG(n); |
| for(int i=0;i<=n;i++){ |
| cout<<sg[i]<<” ”; |
| } |
| return 0; |
| } |

1. Fefjasl
2. 数论
3. 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

|  |
| --- |
| 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; |
| } |

* 1. 不重复组合

|  |
| --- |
| 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; |
| } |

* 1. 全排列枚举

|  |
| --- |
| 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; |
| } |

* 1. 全组合

|  |
| --- |
| 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; |
| } |

* 1. 类循环排列

|  |
| --- |
| 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 |
| } |
| } |

* 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

|  |
| --- |
| /\* |
| \* 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; |
| } |

1. gcd && exgcd

|  |
| --- |
| 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; |
| } |

1. 判断连续素数

|  |
| --- |
| /\* |
| \* 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; |
| }}} |

1. 合数的分解

|  |
| --- |
| /\* |
| \* 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

|  |
| --- |
| /\* |
| \* 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);} |

1. 最大1矩阵(01组成的全是1的最大子矩阵)

|  |
| --- |
| 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; |
| } |

1. 模线性方程

|  |
| --- |
| /\* |
| \* 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 |
| \*/ |
| 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

|  |
| --- |
| 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 |
| \*/ |
| 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

|  |
| --- |
| 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; |
| } |

1. 波利亚计数(不同颜色的珠子组成项链)

|  |
| --- |
| /\* |
| \* c种颜色的珠子，组成长为s的项链，项链没有方向和起始位置 |
| \*/ |
| int gcd(int a, int b) return b ? gcd(b, a % b) : a; |
| int main(int argc, const char \* argv[]) |
| { |
| int c, s; |
| while (cin >> c >> s){ |
| int k; |
| long long p[64]; |
| p[0] = 1; // power of c |
| for (k = 0; k < s; k++) p[k + 1] = p[k] \* c; |
| // reflection part |
| long long count = s & 1 ? s \* p[s / 2 + 1] : (s / 2) \* (p[s / 2] + p[s / 2 + 1]); |
| // rotation part |
| for (k = 1 ; k <= s ; k++){ |
| count += p[gcd(k, s)]; |
| count /= 2 \* s; |
| } |
| cout << count << '\n'; |
| } |
| return 0; |
| } |

1. 简易素数筛

|  |
| --- |
| /\* |
| \* 素数筛选，判断小于MAXN的数是不是素数 |
| \* notprime是一张表，false表示是素数，true表示不是 |
| \*/ |
| const int MAXN = 1000010; |
| bool notprime[MAXN]; |
| void init() |
| { |
| memset(notprime, false, sizeof(notprime)); |
| notprime[0] = notprime[1] = true; |
| for (int i = 2; i < MAXN; i++){ |
| if (!notprime[i]){ |
| if (i > MAXN / i) continue; // 阻止后边i \* i溢出（或者i,j用long long) |
| // 直接从i \* i开始就可以，小于i倍的已经筛选过了 |
| for (int j = i \* i; j < MAXN; j += i) notprime[j] = true; |
| } |
| } |
| } |

1. 约瑟夫环

|  |
| --- |
| /\* |
| \* n个人(编号 1...n),先去掉第m个数,然后从m+1个开始报1, |
| \* 报到k的退出,剩下的人继续从1开始报数.求胜利者的编号. |
| \*/ |
| int main(int argc, const char \* argv[]){ |
| int n, k, m; |
| while (cin >> n >> k >> m, n || k || m){ |
| int i, d, s = 0; |
| for (i = 2; i <= n; i++) s = (s + k) % i; |
| k = k % n; |
| if (k == 0)k = n; |
| d = (s + 1) + (m - k); |
| if (d >= 1 && d <= n)cout << d << '\n'; |
| else if (d < 1)cout << n + d << '\n'; |
| else if (d > n)cout << d % n << '\n'; |
| } |
| return 0; |
| } |

1. 阶乘位数

|  |
| --- |
| #define PI 3.1415926 |
| int main(){ |
| int n, a; |
| while (~scanf("%d", &n)){ |
| a = (int)((0.5 \* log(2 \* PI \* n) + n \* log(n) - n) / log(10)); |
| printf("%d\n", a + 1); } |
| return 0; |
| } |

1. 阶乘最后一个非零位

|  |
| --- |
| /\* |
| \* 阶乘最后非零位 复杂度O(nlongn) |
| \* 返回改为，n以字符串方式传入 |
| \*/ |
| #define MAXN 10000 |
| const int mod[20] = {1, 1, 2, 6, 4, 2, 2, 4, 2, 8, 4, 4, 8, 4, 6, 8, 8, 6, 8, 2}; |
| int lastDigit(char \*buf){ |
| int len = (int)strlen(buf); |
| int a[MAXN], i, c, ret = 1; |
| if (len == 1) return mod[buf[0] - '0']; |
| for (i = 0; i < len; i++) a[i] = buf[len - 1 - i] - '0'; |
| for (; len; len -= !a[len - 1]){ |
| ret = ret \* mod[a[1] % 2 \* 10 + a[0]] % 5; |
| for (c = 0, i = len - 1; i >= 0; i--){ |
| c = c \* 10 + a[i]; |
| a[i] = c / 5; |
| c %= 5; |
| } |
| } |
| return ret + ret % 2 \* 5; |
| } |

1. 随机大素数测试+大素数快速分解

|  |  |
| --- | --- |
|  | const int MAXN = 65; |
|  | long long x[MAXN]; |
|  | set<long long>prime; |
|  | long long qpow(long long a, long long b, long long p) { |
|  | long long ans = 1; |
|  | while(b) { |
|  | if(b&1LL) ans = ans\*a%p; |
|  | a = a\*a%p; |
|  | b >>= 1; |
|  | } |
|  | return ans; |
|  | } |
|  | bool Miller\_Rabin(long long n) { |
|  | if(n == 2) return true; |
|  | int s = 20, i, t = 0; |
|  | long long u = n-1; |
|  | while(!(u & 1)) { |
|  | t++; |
|  | u >>= 1; |
|  | } |
|  | while(s--) { |
|  | long long a = rand()%(n-2)+2; |
|  | x[0] = qpow(a, u, n); |
|  | for(i = 1; i <= t; i++) { |
|  | x[i] = x[i-1]\*x[i-1]%n ; |
|  | if(x[i] == 1 && x[i-1] != 1 && x[i-1] != n-1) return false; |
|  | } |
|  | if(x[t] != 1) return false; |
|  | } |
|  | return true; |
|  | } |
|  | long long gcd(long long a, long long b) return b ? gcd(b, a%b) : a; |
|  | long long Pollard\_Rho(long long n, int c) { |
|  | long long i = 1, k = 2, x = rand()%(n-1)+1, y = x; |
|  | while(true) { |
|  | i++; |
|  | x = (x\*x%n + c)%n; |
|  | long long p = gcd((y-x+n)%n, n); |
|  | if(p != 1 && p != n) return p; |
|  | if(y == x) return n; |
|  | if(i == k) { |
|  | y = x; |
|  | k <<= 1; |
|  | } |
|  | } |
|  | } |
|  | void find(long long n, int c) { |
|  | if(n == 1) return; |
|  | if(Miller\_Rabin(n)) { |
|  | prime.insert(-n); |
|  | return; |
|  | } |
|  | long long p = n, k = c; |
|  | while(p >= n) p = Pollard\_Rho(p, c--); |
|  | find(p, k); |
|  | find(n/p, k); |
|  | } |

1. 高斯消元求1类开关问题的解

|  |
| --- |
| // 高斯消元法求方程组的解 |
| // 适用于格子涂色一类问题 |
| const int MAXN = 300; |
| // 有equ个方程，var个变元。增广矩阵行数为equ，列数为var＋1，分别为0到var |
| int equ, var; |
| int a[MAXN][MAXN]; // 增广矩阵 |
| int x[MAXN]; // 解集 |
| int free\_x[MAXN]; // 用来存储自由变元（多解枚举自由变元可以使用） |
| int free\_num; // 自由变元的个数 |
| // 返回值为－1表示无解，为0是唯一解，否则返回自由变元个数 |
| int Gauss(){ |
| int max\_r, col, k; |
| free\_num = 0; |
| for (k = 0, col = 0; k < equ && col < var; k++, col++){ |
| max\_r = k; |
| for (int i = k + 1; i < equ; i++){ |
| if (abs(a[i][col]) > abs(a[max\_r][col])) max\_r = i; |
| } |
| if (a[max\_r][col] == 0){ |
| k--; |
| free\_x[free\_num++] = col; // 这是自由变元 |
| continue; |
| } |
| if (max\_r != k) for (int j = col; j < var + 1; j++) swap(a[k][j], a[max\_r][j]); |
| for (int i = k + 1; i < equ; i++){ |
| if (a[i][col] != 0) for (int j = col; j < var + 1; j++) a[i][j] ^= a[k][j]; |
| } |
| } |
| for (int i = k; i < equ; i++) if (a[i][col] != 0) return -1; // 无解 |
| if (k < var) return var - k; // 自由变元个数 |
| // 唯一解，回代 |
| for (int i = var - 1; i >= 0; i--) { |
| x[i] = a[i][var]; |
| for (int j = i + 1; j < var; j++) x[i] ^= (a[i][j] && x[j]); |
| } |
| return 0; |
| } |

1. FFT(多项式处理)

|  |
| --- |
| const double PI = acos(-1.0); |
| // 复数结构体 |
| struct Complex |
| { |
| double x, y; // 实部和虚部 x + yi |
| Complex(double \_x = 0.0, double \_y = 0.0){ |
| x = \_x; |
| y = \_y; |
| } |
| Complex operator - (const Complex &b) const{ |
| return Complex(x - b.x, y - b.y); |
| } |
| Complex operator + (const Complex &b) const{ |
| return Complex(x + b.x, y + b.y); |
| } |
| Complex operator \* (const Complex &b) const{ |
| return Complex(x \* b.x - y \* b.y, x \* b.y + y \* b.x); |
| } |
| }; |
| // 进行FFT和IFFT前的反转变换 |
| // 位置i和（i二进制反转后的位置）互换 |
| // len必须去2的幂 |
| void change(Complex y[], int len) |
| { |
| int i, j, k; |
| for (i = 1, j = len / 2; i < len - 1; i++){ |
| if (i < j) swap(y[i], y[j]); |
| // 交换护卫小标反转的元素，i < j保证交换一次 |
| // i做正常的+1，j左反转类型的+1，始终保持i和j是反转的 |
| k = len / 2; |
| while (j >= k){ |
| j -= k; |
| k /= 2;} |
| if (j < k) j += k; |
| } |
| return ; |
| } |
| // FFT |
| // len必须为2 ^ k形式 |
| // on == 1时是DFT，on == -1时是IDFT |
| void fft(Complex y[], int len, int on){ |
| change(y, len); |
| for (int h = 2; h <= len; h <<= 1){ |
| Complex wn(cos(-on \* 2 \* PI / h), sin(-on \* 2 \* PI / h)); |
| for (int j = 0; j < len; j += h){ |
| Complex w(1, 0); |
| for (int k = j; k < j + h / 2; k++){ |
| Complex u = y[k]; |
| Complex t = w \* y[k + h / 2]; |
| y[k] = u + t; |
| y[k + h / 2] = u - t; |
| w = w \* wn; |
| } |
| } |
| } |
| if (on == -1) for (int i = 0; i < len; i++) y[i].x /= len; |
| } |

1. FFT优化大数乘法1

|  |
| --- |
| // FFT |
| /\*HDU 1402 求高精度乘法 A \* B Problem Plus \*/ |
| const double PI = acos(-1.0); |
| // 复数结构体 |
| struct Complex{ |
| double x, y; // 实部和虚部 x + yi |
| Complex(double \_x = 0.0, double \_y = 0.0){ |
| x = \_x; |
| y = \_y; |
| } |
| Complex operator - (const Complex &b) const{ |
| return Complex(x - b.x, y - b.y); |
| } |
| Complex operator + (const Complex &b) const{ |
| return Complex(x + b.x, y + b.y); |
| } |
| Complex operator \* (const Complex &b) const{ |
| return Complex(x \* b.x - y \* b.y, x \* b.y + y \* b.x); |
| } |
| }; |
| // 进行FFT和IFFT前的反转变换 |
| // 位置i和（i二进制反转后的位置）互换 |
| // len必须去2的幂 |
| void change(Complex y[], int len){ |
| int i, j, k; |
| for (i = 1, j = len / 2; i < len - 1; i++){ |
| if (i < j) swap(y[i], y[j]); |
| // 交换护卫小标反转的元素，i < j保证交换一次 |
| // i做正常的+1，j左反转类型的+1，始终保持i和j是反转的 |
| k = len / 2; |
| while (j >= k){ |
| j -= k; |
| k /= 2; |
| } |
| if (j < k) j += k; |
| } |
| return ; |
| } |
| // FFT |
| // len必须为2 ^ k形式 |
| // on == 1时是DFT，on == -1时是IDFT |
| void fft(Complex y[], int len, int on){ |
| change(y, len); |
| for (int h = 2; h <= len; h <<= 1){ |
| Complex wn(cos(-on \* 2 \* PI / h), sin(-on \* 2 \* PI / h)); |
| for (int j = 0; j < len; j += h){ |
| Complex w(1, 0); |
| for (int k = j; k < j + h / 2; k++){ |
| Complex u = y[k]; |
| Complex t = w \* y[k + h / 2]; |
| y[k] = u + t; |
| y[k + h / 2] = u - t; |
| w = w \* wn; |
| } |
| } |
| } |
| if (on == -1){ |
| for (int i = 0; i < len; i++) y[i].x /= len; |
| } |
| } |
| const int MAXN = 200010; |
| Complex x1[MAXN], x2[MAXN]; |
| char str1[MAXN / 2], str2[MAXN]; |
| int sum[MAXN]; |
| int main(int argc, const char \* argv[]) |
| { |
| while (cin >> str1 >> str2) |
| { |
| int len1 = (int)strlen(str1); |
| int len2 = (int)strlen(str2); |
| int len = 1; |
| while (len < len1 \* 2 || len < len2 \* 2) len <<= 1; |
| for (int i = 0; i < len1; i++){ |
| x1[i] = Complex(str1[len1 - 1 - i] - '0', 0); |
| } |
| for (int i = len1; i < len; i++){ |
| x1[i] = Complex(0, 0); |
| } |
| for (int i = 0; i < len2; i++){ |
| x2[i] = Complex(str2[len2 - 1 - i] - '0', 0); |
| } |
| for (int i = len2; i < len; i++){ |
| x2[i] = Complex(0, 0); |
| } |
| // 求DFT |
| fft(x1, len, 1); |
| fft(x2, len, 1); |
| for (int i = 0; i < len; i++){ |
| x1[i] = x1[i] \* x2[i]; |
| } |
| fft(x1, len, -1); |
| for (int i = 0; i < len; i++){ |
| sum[i] = (int)(x1[i].x + 0.5); |
| } |
| for (int i = 0; i < len; i++){ |
| sum[i + 1] += sum[i] / 10; |
| sum[i] %= 10; |
| } |
| len = len1 + len2 - 1; |
| while (sum[len] <= 0 && len > 0) len--; |
| for (int i = len; i >= 0; i--) printf("%c", sum[i] + '0'); |
| putchar('\n'); |
| } |
| return 0; |
| } |

1. FFT优化高精度乘法2

|  |
| --- |
| template <class T> |
| void read(T &x){ |
| char c; |
| bool op = 0; |
| while(c = getchar(), c < '0' || c > '9') |
| if(c == '-') op = 1; |
| x = c - '0'; |
| while(c = getchar(), c >= '0' && c <= '9') |
| x = x \* 10 + c - '0'; |
| if(op) x = -x; |
| } |
| template <class T> |
| void write(T x){ |
| if(x < 0) putchar('-'), x = -x; |
| if(x >= 10) write(x / 10); |
| putchar('0' + x % 10); |
| } |
| const int N = 1000005; |
| const double PI = acos(-1); |
| typedef complex <double> cp; |
| char sa[N], sb[N]; |
| int n = 1, lena, lenb, res[N]; |
| cp a[N], b[N], omg[N], inv[N]; |
| void init(){ |
| for(int i = 0; i < n; i++){ |
| omg[i] = cp(cos(2 \* PI \* i / n), sin(2 \* PI \* i / n)); |
| inv[i] = conj(omg[i]); |
| } |
| } |
| void fft(cp \*a, cp \*omg){ |
| int lim = 0; |
| while((1 << lim) < n) lim++; |
| for(int i = 0; i < n; i++){ |
| int t = 0; |
| for(int j = 0; j < lim; j++) |
| if((i >> j) & 1) t |= (1 << (lim - j - 1)); |
| if(i < t) swap(a[i], a[t]); // i < t 的限制使得每对点只被交换一次（否则交换两次相当于没交换） |
| } |
| for(int l = 2; l <= n; l \*= 2){ |
| int m = l / 2; |
| for(cp \*p = a; p != a + n; p += l) |
| for(int i = 0; i < m; i++){ |
| cp t = omg[n / l \* i] \* p[i + m]; |
| p[i + m] = p[i] - t; |
| p[i] += t; |
| } |
| } |
| } |
| int main(){ |
| scanf("%s%s", sa, sb); |
| lena = strlen(sa), lenb = strlen(sb); |
| while(n < lena + lenb) n \*= 2; |
| for(int i = 0; i < lena; i++) |
| a[i].real(sa[lena - 1 - i] - '0'); |
| for(int i = 0; i < lenb; i++) |
| b[i].real(sb[lenb - 1 - i] - '0'); |
| init(); |
| fft(a, omg); |
| fft(b, omg); |
| for(int i = 0; i < n; i++) |
| a[i] \*= b[i]; |
| fft(a, inv); |
| for(int i = 0; i < n; i++){ |
| res[i] += floor(a[i].real() / n + 0.5); |
| res[i + 1] += res[i] / 10; |
| res[i] %= 10; |
| } |
| for(int i = res[lena + lenb - 1] ? lena + lenb - 1: lena + lenb - 2; i >= 0; i--) |
| putchar('0' + res[i]); |
| enter; |
| return 0; |
| } |

1. 五边形定理，整数划分

|  |
| --- |
| const int MAXN = 1e5 + 10; |
| const int MOD = 1e9 + 7; |
| int n, ans[MAXN]; |
| int main(){ |
| scanf("%d", &n);//读入n为求把n划分成至多n个数 // 任意划分 |
| ans[0] = 1; |
| for (int i = 1; i <= n; ++i){ |
| for (int j = 1; f(j) <= i; ++j){ |
| if (j & 1) ans[i] = (ans[i] + ans[i - f(j)]) % MOD; |
| else ans[i] = (ans[i] - ans[i - f(j)] + MOD) % MOD; |
| } |
| for (int j = 1; g(j) <= i; ++j){ |
| if (j & 1) ans[i] = (ans[i] + ans[i - g(j)]) % MOD; |
| else ans[i] = (ans[i] - ans[i - g(j)] + MOD) % MOD; |
| } |
| } |
| printf("%d\n", ans[n]); |
| return 0;} |

1. 整数划分,不能有k重复

|  |
| --- |
| // 问一个数n能被拆分成多少种情况 |
| // 且要求拆分元素重复次数不能≥k |
| const int MOD = 1e9 + 7; |
| const int MAXN = 1e5 + 10; |
| int ans[MAXN]; |
| // 此函数求ans[]效率比上一个代码段中求ans[]效率高很多 |
| void init(){ |
| memset(ans, 0, sizeof(ans)); |
| ans[0] = 1; |
| for (int i = 1; i < MAXN; ++i){ |
| ans[i] = 0; |
| for (int j = 1; ; j++){ |
| int tmp = (3 \* j - 1) \* j / 2; |
| if (tmp > i) break; |
| int tmp\_ = ans[i - tmp]; |
| if (tmp + j <= i) tmp\_ = (tmp\_ + ans[i - tmp - j]) % MOD; |
| if (j & 1) ans[i] = (ans[i] + tmp\_) % MOD; |
| else ans[i] = (ans[i] - tmp\_ + MOD) % MOD; |
| } |
| } |
| return ; |
| } |
| int solve(int n, int k){ |
| int res = ans[n]; |
| for (int i = 1; ; i++){ |
| int tmp = k \* i \* (3 \* i - 1) / 2; |
| if (tmp > n) break; |
| int tmp\_ = ans[n - tmp]; |
| if (tmp + i \* k <= n) tmp\_ = (tmp\_ + ans[n - tmp - i \* k]) % MOD; |
| if (i & 1) res = (res - tmp\_ + MOD) % MOD; |
| else res = (res + tmp\_) % MOD; |
| } |
| return res; |
| } |
| int main(int argc, const char \* argv[]) |
| { |
| init(); |
| int T, n, k; |
| cin >> T; |
| while (T--) |
| { |
| cin >> n >> k; |
| cout << solve(n, k) << '\n'; |
| } |
| return 0; |
| } |

1. 约数之和

|  |
| --- |
| /\* 求A^B的约数之和对MOD取模 |
| \* 需要素数筛选和合数分解的算法，需要先调用getPrime(); |
| \* 参考《合数相关》 |
| 1+p+p^2+p^3+...+p^n \*/ |
| const int MOD = 1000000; |
| 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; |
| } |
| long long pow\_m(long long a, long long n){ |
| long long ret = 1; |
| long long tmp = a % MOD; |
| while(n){ |
| if (n & 1) ret = (ret \* tmp) % MOD; |
| tmp = tmp \* tmp % MOD; |
| n >>= 1; } |
| return ret;} |
| // 计算1+p＋p^2+...+p^n |
| long long sum(long long p, long long n){ |
| if (p == 0) return 0; |
| if (n == 0) return 1; |
| if (n & 1) return ((1 + pow\_m(p, n / 2 + 1)) % MOD \* sum(p, n / 2) % MOD) % MOD; |
| else return ((1 + pow\_m(p, n / 2 + 1)) % MOD \* sum(p, n / 2 - 1) + pow\_m(p, n / 2) % MOD) % MOD; |
| } |
| // 返回A^B的约数之和%MOD |
| long long solve(long long A, long long B){ |
| getFactors(A); |
| long long ans = 1; |
| for (int i = 0; i < fatCnt; i++){ |
| ans \*= sum(factor[i][0], B \* factor[i][1]) % MOD; |
| ans %= MOD; |
| } |
| return ans;} |

1. 莫比乌斯单独求解

|  |
| --- |
| int MOD(int a, int b) {return a - a / b \* b;} |
| int miu(int n){ |
| int cnt, k = 0; |
| for (int i = 2; i \* i <= n; i++){ |
| if (MOD(n, i)) continue; |
| cnt = 0; |
| k++; |
| while (MOD(n, i) == 0){ |
| n /= i; |
| cnt++; |
| } |
| if (cnt >= 2) return 0; |
| } |
| if (n != 1) k++; |
| return MOD(k, 2) ? -1 : 1;} |
| int MOD(int a, int b) {return a - a / b \* b;} |
| int miu(int n){ |
| int cnt, k = 0; |
| for (int i = 2; i \* i <= n; i++){ |
| if (MOD(n, i)) continue; |
| cnt = 0; |
| k++; |
| while (MOD(n, i) == 0){ |
| n /= i; |
| cnt++; |
| } |
| if (cnt >= 2) return 0; |
| } |
| if (n != 1) k++; |
| return MOD(k, 2) ? -1 : 1;} |
| int MOD(int a, int b) {return a - a / b \* b;} |

1. 莫比乌斯线性筛法

|  |
| --- |
| /\* 莫比乌斯反演公式 |
| 线性筛法求解积性函数（莫比乌斯函数） |
| 如果一个数包含平方因子，那么miu(n) = 0。 |
| 例如：miu(4), miu(12), miu(18) = 0。 |
| 如果一个数不包含平方因子，并且有k个不同的质因子， |
| 那么miu(n) = (-1)^k。例如：miu(2), miu(3), miu(30) |
| = -1,miu(1), miu(6), miu(10) = 1。\*/ |
| const int MAXN = 1000000; |
| bool check[MAXN + 10]; |
| int prime[MAXN + 10]; |
| int mu[MAXN + 10]; |
| void Moblus(){ |
| memset(check, false, sizeof(check)); |
| mu[1] = 1; |
| int tot = 0; |
| for (int i = 2; i <= MAXN; i++){ |
| if (!check[i]){ |
| prime[tot++] = i; |
| mu[i] = -1; |
| } |
| for (int j = 0; j < tot; j++){ |
| if (i \* prime[j] > MAXN) break; |
| check[i \* prime[j]] = true; |
| if (i % prime[j] == 0){ |
| mu[i \* prime[j]] = 0; |
| break; |
| } |
| else mu[i \* prime[j]] = -mu[i]; |
| } |
| } |
| } |
| int main(){ |
| Moblus(); // mu[]存储对应位置的莫比乌斯函数值 |
| cout << mu[5] << endl; } |

1. 莫比乌斯求解区间gcd(x,y)等于k的对数

|  |
| --- |
| 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; |
| } |

1. BGGS,a^x=b%n

|  |
| --- |
| /\* baby\_step giant \_step |
| a^x = b(mod n) n不要求是素数 |
| 求解上式0 ≤ x < n的解\*/ |
| #define MOD 76543 |
| int hs[MOD]; |
| int head[MOD]; |
| int \_next[MOD]; |
| int id[MOD]; |
| int top; |
| void insert(int x, int y){ |
| int k = x % MOD; |
| hs[top] = x; |
| id[top] = y; |
| \_next[top] = head[k]; |
| head[k] = top++; |
| return ; |
| } |
| int find(int x){ |
| int k = x % MOD; |
| for (int i = head[k]; i != -1; i = \_next[i]){ |
| if (hs[i] == x) return id[i]; |
| } |
| return -1; |
| } |
| long long BSGS(int a, int b, int n){ |
| memset(head, -1, sizeof(head)); |
| top = 1; |
| if (b == 1) return 0; |
| int m = (int)sqrt(n \* 1.0), j; |
| long long x = 1, p = 1; |
| for (int i = 0; i < m; i++, p = p \* a % n) insert(p \* b % n, i); |
| for (long long i = m; ; i++){ |
| if ((j = find(x = x \* p % n)) != -1) return i - j; |
| if (i > n) break; |
| } |
| return -1; |
| } |

1. 牛顿法多项式求根

|  |
| --- |
| /\* 牛顿法解多项式的根 |
| \* 输入:多项式系数c[],多项式度数n,求在[a,b]间的根 |
| \* 输出:根 要求保证[a,b]间有根\*/ |
| double fabs(double x){ return (x < 0) ? -x : x;} |
| double f(int m, double c[], double x){ |
| int i; |
| double p = c[m]; |
| for (i = m; i > 0; i--) p = p \* x + c[i - 1]; |
| return p; |
| } |
| int newton(double x0, double \*r, double c[], double cp[], int n, double a, double b, double eps){ |
| int MAX\_ITERATION = 1000; |
| int i = 1; |
| double x1, x2, fp, eps2 = eps / 10.0; |
| x1 = x0; |
| while (i < MAX\_ITERATION){ |
| x2 = f(n, c, x1); |
| fp = f(n - 1, cp, x1); |
| if ((fabs(fp) < 0.000000001) && (fabs(x2) > 1.0)) return 0; |
| x2 = x1 - x2 / fp; |
| if (fabs(x1 - x2) < eps2){ |
| if (x2 < a || x2 > b) return 0; |
| \*r = x2; |
| return 1; |
| } |
| x1 = x2; |
| i++; |
| } |
| return 0; |
| } |
| double Polynomial\_Root(double c[], int n, double a, double b, double eps){ |
| double \*cp; |
| int i; |
| double root; |
| cp = (double \*)calloc(n, sizeof(double)); |
| for (i = n - 1; i >= 0; i--) cp[i] = (i + 1) \* c[i + 1]; |
| if (a > b){ |
| root = a; |
| a = b; |
| b = root; |
| } |
| if ((!newton(a, &root, c, cp, n, a, b, eps)) && (!newton(b, &root, c, cp, n, a, b, eps))){ |
| newton((a + b) \* 0.5, &root, c, cp, n, a, b, eps); |
| } |
| free(cp); |
| if (fabs(root) < eps) return fabs(root); |
| else return root; |
| } |
| int main(){ |
| Dobule c[2]={2,-1}; |
| int n = 1, a = -5, b = 5; |
| double eps = 1e-8; |
| double root = Polynomial\_Root(c, n, a, b, eps); |
| cout << root << endl; } |
|  |
|  |

1. 自适应积分

|  |
| --- |
| const double eps = 1e-6; // 积分精度 |
| // 被积函数 |
| double **F**(double x){ |
| double ans; |
| // 被积函数 |
| // ans = x \* exp(x); // 椭圆为例 |
| return ans; |
| } |
| // 三点simpson法，这里要求F是一个全局函数 |
| double **simpson**(double a, double b){ |
| double c = a + (b - a) / 2; |
| return (F(a) + 4 \* F(c) + F(b)) \* (b - a) / 6; |
| } |
| // 自适应simpson公式（递归过程），已知整个区间[a, b]上的三点simpson指A |
| double **asr**(double a, double b, double eps, double A){ |
| double c = a + (b - a) / 2; |
| double L = simpson(a, c), R = simpson(c, b); |
| if (fabs(L + R - A) <= 15 \* eps){ |
| return L + R + (L + R - A) / 15.0; |
| } |
| return asr(a, c, eps / 2, L) + asr(c, b, eps / 2, R); |
| } |
| // 自适应simpson公式（主过程） |
| double **asr**(double a, double b, double eps){ return asr(a, b, eps, simpson(a, b)); } |
| int **main**(int argc, const char \* argv[]){ |
| // std::cout << asr(1, 2, eps) << '\n'; |
| return 0; } |

1. 容斥dfs

|  |
| --- |
| const int MAXN = 1111; |
| int n; |
| double ans; |
| double p[MAXN]; |
| void dfs(int x, int tot, double sum){ // dfs(1, 0, ?) |
| if (x == n + 1){ |
| if (sum == 0.0) return ; |
| if (tot & 1) ans += 1 / sum; // 可替换为任何公式 |
| else ans -= 1 / sum; |
| return ; |
| } |
| dfs(x + 1, tot, sum); |
| dfs(x + 1, tot + 1, sum + p[x]); |
| } |
| void init(){ |
| for (int i=1; i<=n; i++) cin>>p[i]; |
| } |
| int main(){ |
| while(cin >> n){ |
| ans = 0; |
| init(); |
| dfs(1,0,0.0); |
| printf("%.4f\n",ans); |
| } |
| return 0; |
| } |

1. 斐波那契单独求解

|  |
| --- |
| /\*求斐波那契数列第N项，模MOD \*/ |
| #define mod(a, m) ((a) % (m) + (m)) % (m) |
| const int MOD = 1e9 + 9; |
| struct MATRIX{ |
| long long a[2][2];}; |
| MATRIX a; |
| long long f[2]; |
| void ANS\_Cf(MATRIX a) |
| { |
| f[0] = mod(a.a[0][0] + a.a[1][0], MOD); |
| f[1] = mod(a.a[0][1] + a.a[1][1], MOD); |
| return ;} |
| MATRIX MATRIX\_Cf(MATRIX a, MATRIX b){ |
| MATRIX ans; |
| int k; |
| for (int i = 0; i < 2; i++){ |
| for (int j = 0; j < 2; j++){ |
| ans.a[i][j] = 0; |
| k = 0; |
| while (k < 2){ |
| ans.a[i][j] += a.a[k][i] \* b.a[j][k]; |
| ans.a[i][j] = mod(ans.a[i][j], MOD); |
| ++k; |
| } |
| } |
| } |
| return ans; |
| } |
| MATRIX MATRIX\_Pow(MATRIX a, long long n){ |
| MATRIX ans; |
| ans.a[0][0] = 1; |
| ans.a[1][1] = 1; |
| ans.a[0][1] = 0; |
| ans.a[1][0] = 0; |
| while (n){ |
| if (n & 1){ |
| ans = MATRIX\_Cf(ans, a); |
| } |
| n = n >> 1; |
| a = MATRIX\_Cf(a, a); |
| } |
| return ans; |
| } |
| int main(){ |
| long long n; |
| while (cin >> n){ |
| if (n == 1){ |
| cout << '1' << '\n'; |
| continue;} |
| a.a[0][0] = a.a[0][1] = a.a[1][0] = 1; |
| a.a[1][1] = 0; |
| a = MATRIX\_Pow(a, n - 2); |
| ANS\_Cf(a); |
| cout << f[0] << '\n'; |
| } |
| return 0; |
| } |

1. 求循环节长度

|  |
| --- |
| /\*求1/i的循环节长度的最大值，i<=n\*/ |
| const int MAXN = 1005; |
| int res[MAXN]; // 循环节长度 |
| int main(){ |
|  |
| int i, temp, j, n; |
| for (temp = 1; temp <= 1000; temp++){ |
| i = temp; |
| while (i % 2 == 0) i /= 2; |
| while (i % 5 == 0) i /= 5; |
| n = 1; |
| for (j = 1; j <= i; j++){ |
| n \*= 10; |
| n %= i; |
| if (n == 1){ |
| res[temp] = j; |
| break; } |
| } |
| } |
| int max\_re; |
| while (cin >> n){ |
| max\_re = 1; |
| for (i = 1; i <= n; i++){ |
| if (res[i] > res[max\_re]) max\_re = i; |
| } |
| cout << max\_re << endl; |
| } |
| return 0;} |

1. 矩阵
2. nn矩阵的x次幂快速幂

|  |
| --- |
| /\*矩阵快速幂 n\*n矩阵的x次幂\*/ |
| #define MAXN 111 |
| #define mod(x) ((x) % MOD) |
| #define MOD 1000000007 |
| #define LL long long |
| int n; |
| struct mat{ |
| int m[MAXN][MAXN]; |
| } unit; // 单元矩阵 |
| // 矩阵乘法 |
| mat operator \* (mat a, mat &b){ |
| mat ret; |
| memset(ret.m, 0, sizeof(ret.m)); |
| for (int k = 0; k < n; k++){ |
| for (int i = 0; i < n; i++){ |
| if (a.m[i][k]) |
| for (int j = 0; j < n; j++) ret.m[i][j] = mod(ret.m[i][j] + (LL)a.m[i][k] \* b.m[k][j]); |
| } |
| } |
| return ret; |
| } |
| void init\_unit(){ |
| for (int i = 0; i < MAXN; i++){ |
| unit.m[i][i] = 1; |
| } |
| return ;} |
| mat pow\_mat(mat a, LL n){ |
| mat ret = unit; |
| while (n){ |
| if (n & 1){ |
| // n--; |
| ret = ret \* a; |
| } |
| n >>= 1; |
| a = a \* a; |
| } |
| return ret; |
| } |
| int main() |
| { |
| LL x; |
| init\_unit(); |
| while (cin >> n >> x) |
| { |
| mat a; |
| for (int i = 0; i < n; i++) |
| { |
| for (int j = 0; j < n; j++) cin >> a.m[i][j]; |
| } |
| a = pow\_mat(a, x); // a矩阵的x次幂 |
| // 输出矩阵 |
| for (int i = 0; i < n; i++) |
| for (int j = 0; j < n; j++) |
| if (j + 1 == n) cout << a.m[i][j] << endl; |
| else cout << a.m[i][j] << " "; |
| } |
| return 0; |
| } |

1. 矩阵乘法

|  |
| --- |
| int n; |
| struct mat{ |
| int m[MAXN][MAXN];}; |
| mat operator \* (mat a, mat &b) |
| { |
| mat ret; |
| memset(ret.m, 0, sizeof(ret.m)); |
| for (int k = 0; k < n; k++){ |
| for (int i = 0; i < n; i++){ |
| if (a.m[i][k]) |
| for (int j = 0; j < n; j++) ret.m[i][j] = mod(ret.m[i][j] + (LL)a.m[i][k] \* b.m[k][j]); |
| } |
| } |
| return ret; } |

1. 矩阵乘法结构体实现加判等

|  |
| --- |
| /\*AB == C ??? \*/ |
| struct Matrix{ |
| Type mat[MAXN][MAXN]; |
| int n, m; |
| Matrix(){ |
| n = m = MAXN; |
| memset(mat, 0, sizeof(mat)); |
| } |
| Matrix(const Matrix &a){ |
| set\_size(a.n, a.m); |
| memcpy(mat, a.mat, sizeof(a.mat)); |
| } |
| Matrix & operator = (const Matrix &a){ |
| set\_size(a.n, a.m); |
| memcpy(mat, a.mat, sizeof(a.mat)); |
| return \*this; |
| } |
| void set\_size(int row, int column){ |
| n = row; |
| m = column; |
| } |
| friend Matrix operator \* (const Matrix &a, const Matrix &b){ |
| Matrix ret; |
| ret.set\_size(a.n, b.m); |
| for (int i = 0; i < a.n; ++i) |
| for (int k = 0; k < a.m; ++k) |
| if (a.mat[i][k]) |
| for (int j = 0; j < b.m; ++j) |
| if (b.mat[k][j]) |
| ret.mat[i][j] = ret.mat[i][j] + a.mat[i][k] \* b.mat[k][j]; |
| return ret; |
| } |
| friend bool operator == (const Matrix &a, const Matrix &b){ |
| if (a.n != b.n || a.m != b.m) return false; |
| for (int i = 0; i < a.n; ++i) |
| for (int j = 0; j < a.m; ++j) |
| if (a.mat[i][j] != b.mat[i][j]) return false; |
| return true; |
| } |
| }; |

1. Vdd
2. 求N以内因子最多的数

|  |
| --- |
| const int MAXP = 16; |
| const int prime[MAXP] = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47, 53}; |
| ll n, res, ans; |
| void dfs(ll cur, ll num, int key, ll pre) { // 当前值/当前约数数量/当前深度/上一个数 |
| if (key >= MAXP) return ; |
| else{ |
| if (num > ans){ |
| res = cur; |
| ans = num; |
| } |
| else if (num == ans){ // 如果约数数量相同，则取较小的数 |
| res = min(cur, res); |
| } |
| ll i; |
| for ( i = 1; i <= pre; i++){ |
| if (cur <= n / prime[key]) { // cur\*prime[key]<=n |
| cur \*= prime[key]; |
| dfs(cur, num \* (i + 1), key + 1, i); |
| } |
| else break; |
| } |
| }} |
| void solve(){ |
| res = 1; |
| ans = 1; |
| dfs(1, 1, 0, 15); |
| cout << res << ' ' << ans << endl;} |
| int main(int argc, const char \* argv[]){ |
| int T; |
| cin >> T; |
| while (T--){ |
| cin >> n; |
| solve(); |
| } |
| return 0; } |

1. Fsa
2. 树
   * 1. 二叉树
3. 中序遍历+后序遍历建二叉树

|  |
| --- |
| // UVa548 Tree |
| // Rujia Liu |
| // 题意：给一棵点带权（权各不相同，都是正整数）二叉树的中序和后序遍历，找一个叶子使得它到根的路径上的权和最小。如果有多解，该叶子本身的权应尽量小 |
| // 算法：递归建树，然后DFS。注意，直接递归求结果也可以，但是先建树的方法不仅直观，而且更好调试 |
| // 因为各个结点的权值各不相同且都是正整数，直接用权值作为结点编号 |
| const int maxv = 10000 + 10; |
| int in\_order[maxv], post\_order[maxv], lch[maxv], rch[maxv]; |
| int n; |
| bool read\_list(int\* a) { |
| string line; |
| if(!getline(cin, line)) return false; |
| stringstream ss(line); |
| n = 0; |
| int x; |
| while(ss >> x) a[n++] = x; |
| return n > 0; |
| } |
| // 把in\_order[L1..R1]和post\_order[L2..R2]建成一棵二叉树，返回树根 |
| int build(int L1, int R1, int L2, int R2) { |
| if(L1 > R1) return 0; // 空树 |
| int root = post\_order[R2]; |
| int p = L1; |
| while(in\_order[p] != root) p++; |
| int cnt = p-L1; // 左子树的结点个数 |
| lch[root] = build(L1, p-1, L2, L2+cnt-1); |
| rch[root] = build(p+1, R1, L2+cnt, R2-1); |
| return root;} |
| int best, best\_sum; // 目前为止的最优解和对应的权和 |
| void dfs(int u, int sum) { |
| sum += u; |
| if(!lch[u] && !rch[u]) // 叶子 |
| if(sum < best\_sum || (sum == best\_sum && u < best)) { best = u; best\_sum = sum; } |
| if(lch[u]) dfs(lch[u], sum); |
| if(rch[u]) dfs(rch[u], sum); |
| } |
| int main() { |
| while(read\_list(in\_order)) { |
| read\_list(post\_order); |
| build(0, n-1, 0, n-1); |
| best\_sum = 1000000000; |
| dfs(post\_order[n-1], 0); |
| cout << best << "\n"; |
| } |
| return 0; |
| } |

1. 二叉树建树

|  |
| --- |
| // UVa122 Trees on the level |
| const int maxn = 256 + 10; |
| struct Node{ |
| bool have\_value; |
| int v; |
| Node\* left, \*right; |
| Node():have\_value(false),left(NULL),right(NULL){} |
| }; |
| Node\* root; |
| Node\* newnode() { return new Node(); } |
| bool failed; |
| void addnode(int v, char\* s) { |
| int n = strlen(s); |
| Node\* u = root; |
| for(int i = 0; i < n; i++) |
| if(s[i] == 'L') { |
| if(u->left == NULL) u->left = newnode(); |
| u = u->left; |
| } else if(s[i] == 'R') { |
| if(u->right == NULL) u->right = newnode(); |
| u = u->right; |
| } |
| if(u->have\_value) failed = true; |
| u->v = v; |
| u->have\_value = true; |
| } |
| void remove\_tree(Node\* u) { |
| if(u == NULL) return; |
| remove\_tree(u->left); |
| remove\_tree(u->right); |
| delete u; } |
| char s[maxn]; |
| bool read\_input() { |
| failed = false; |
| remove\_tree(root); |
| root = newnode(); |
| for(;;) { |
| if(scanf("%s", s) != 1) return false; |
| if(!strcmp(s, "()")) break; |
| int v; |
| sscanf(&s[1], "%d", &v); |
| addnode(v, strchr(s, ',')+1); |
| } |
| return true; } |
| bool bfs(vector<int>& ans) { |
| queue<Node\*> q; |
| ans.clear(); |
| q.push(root); |
| while(!q.empty()) { |
| Node\* u = q.front(); q.pop(); |
| if(!u->have\_value) return false; |
| ans.push\_back(u->v); |
| if(u->left != NULL) q.push(u->left); |
| if(u->right != NULL) q.push(u->right); |
| } |
| return true; } |
| int main() { |
| vector<int> ans; |
| while(read\_input()) { |
| if(!bfs(ans)) failed = 1; |
| if(failed) printf("not complete\n"); |
| else { |
| for(int i = 0; i < ans.size(); i++) { |
| if(i != 0) printf(" "); |
| printf("%d", ans[i]); |
| } |
| printf("\n"); |
| } } return 0; } |

* + 1. saff

1. 其他
   1. C++大数

|  |
| --- |
| struct Bign |
| { |
| int len,s[MAXN]; |
| Bign(){ |
| memset(s,0,sizeof(s)); |
| len=1; |
| } |
| Bign(int num){\*this=num;} |
| Bign(const char \*num){\*this=num;} |
| void clean(){while(len>1&&!s[len-1])len--;} |
| Bign operator = (const int num){ |
| char s[MAXN]; |
| sprintf(s,"%d",num); |
| \*this=s; |
| return \*this; |
| } |
| Bign operator = (const char \*num) { |
| len=strlen(num); |
| for(int i=0;i<len;i++)s[i]=num[len-i-1]-'0'; |
| return \*this; |
| } |
| Bign operator + (const Bign& b) { |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;g||i<Max(len,b.len);i++){ |
| int x=g; |
| if(i<b.len)x+=b.s[i]; |
| if(i<len)x+=s[i]; |
| c.s[c.len++]=x%10; |
| g=x/10; |
| } |
| return c; |
| } |
| Bign operator - (const Bign& b){ |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;i<len;i++){ |
| int x=s[i]-g; |
| if(i<b.len)x-=b.s[i]; |
| if(x>=0)g=0; |
| else{g=1;x+=10;} |
| c.s[c.len++]=x; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator \* (const Bign& b){ |
| Bign c; |
| c.len=len+b.len; |
| for(int i=0;i<len;i++){ |
| for(int j=0;j<b.len;j++) c.s[i+j]+=s[i]\*b.s[j]; |
| } |
| for(int i=0;i<c.len;i++) { |
| c.s[i+1]+=c.s[i]/10; |
| c.s[i]%=10; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator \* (const int& b){ |
| Bign c; |
| c.len=0; |
| for(int i=0,g=0;g||i<len;i++){ |
| int x; |
| if(i<len)x=s[i]\*b+g; |
| else x=g; |
| c.s[c.len++]=x%10; |
| g=x/10; |
| } |
| return c; |
| } |
| Bign operator / (const Bign& b){ |
| Bign c,f=0; |
| for(int i=len-1;i>=0;i--){ |
| f=f\*10; |
| f.s[0]=s[i]; |
| while(f>=b){ |
| f=f-b; |
| c.s[i]++; |
| } |
| } |
| c.len=len; |
| c.clean(); |
| return c; |
| } |
| Bign operator / (const int& b){ |
| Bign c,d=\*this; |
| c.len=len; |
| for(int i=len-1,g=0;i>=0;i--){ |
| d.s[i]+=g\*10; |
| c.s[i]=d.s[i]/b; |
| g=d.s[i]%b; |
| } |
| c.clean(); |
| return c; |
| } |
| Bign operator % (const Bign& b){ |
| Bign c=\*this/b; |
| c=\*this-c\*b; |
| return c; |
| } |
| Bign operator += (const Bign& b) |
| {\*this=\*this+b;return \*this;} |
| Bign operator -= (const Bign& b) |
| {\*this=\*this-b;return \*this;} |
| Bign operator \*= (const Bign& b) |
| {\*this=\*this\*b;return \*this;} |
| Bign operator /= (const Bign& b) |
| {\*this=\*this/b;return \*this;} |
| Bign operator \*= (const int& b) |
| {\*this=\*this\*b;return \*this;} |
| Bign operator /= (const int& b) |
| {\*this=\*this/b;return \*this;} |
| Bign operator %= (const Bign& b) |
| {\*this=\*this%b;return \*this;} |
| bool operator < (const Bign& b){ |
| if(b.len!=len)return len<b.len; |
| for(int i=len-1;i>=0;i--) if(s[i]!=b.s[i])return s[i]<b.s[i]; |
| return 0; |
| } |
| bool operator > (const Bign& b){ |
| if(b.len!=len)return len>b.len; |
| for(int i=len-1;i>=0;i--) if(s[i]!=b.s[i])return s[i]>b.s[i]; |
| return 0; |
| } |
| bool operator == (const Bign& b) |
| {return !(\*this>b)&&!(\*this<b);} |
| bool operator <= (const Bign& b) |
| {return !(\*this>b);} |
| bool operator >= (const Bign& b) |
| {return !(\*this<b);} |
| bool operator != (const Bign& b) |
| {return !(\*this==b);} |
| string str() const { |
| string res; |
| for(int i=0;i<len;i++) |
| res=char(s[i]+'0')+res; |
| return res; |
| } |
| }; |
| //cin 读入 |
| istream& operator >> (istream&in,Bign &x){ |
| string s; |
| in>>s; |
| x=s.c\_str(); |
| return in; |
| } |
| ostream& operator << (ostream&out,Bign x){ |
| out<<x.str(); |
| return out; |
| } |
| int main(){ |
| Bign a,b; |
| cin>>a>>b; |
| cout<<a%b<<endl; |
| return 0; |
| } |

* 1. 基姆拉尔森公式,给年月日,计算星期几

W = (D + 2 \* M + 3 \* (M + 1) \ 5 + Y + Y \ 4 - Y \ 100 + Y \ 400) Mod 7;

* 1. fsfs

1. 图
   1. AStar\_K短路

|  |
| --- |
| const int maxn=100010; |
| int n,m,dis[maxn]; |
| int tot,head1[maxn],head2[maxn]; |
| bool flag[maxn]; |
| struct edge{ |
| int to; |
| int w; |
| int next; |
| }e[maxn\*2],e2[maxn\*2]; |
| struct node{ |
| int f; |
| int g; |
| int from; |
| bool operator < (node a)const{ |
| if(a.f==f) |
| return g>a.g; |
| return f>a.f; |
| } |
| }; |
| void add\_edge(int u,int v,int w){ |
| tot++; |
| e[tot].to=v; |
| e[tot].w=w; |
| e[tot].next=head1[u]; |
| head1[u]=tot; |
| e2[tot].to=u; |
| e2[tot].w=w; |
| e2[tot].next=head2[v]; |
| head2[v]=tot; |
| } |
| void prepare(){ |
| for(int i=1;i<=n;i++) |
| dis[i]=maxn;tot=0; |
| memset(head1,0,sizeof(head1)); |
| memset(head2,0,sizeof(head2)); |
| } |
| void spfa(int t){ |
| for(int i=1;i<=n;i++) |
| dis[i]=maxn; |
| dis[t]=0; |
| queue<int> q; |
| q.push(t); |
| flag[t]=1; |
| while(!q.empty()){ |
| int v=q.front(); |
| q.pop();flag[v]=0; |
| for(int i=head2[v];i;i=e2[i].next) |
| if(dis[e2[i].to]>dis[v]+e2[i].w){ |
| dis[e2[i].to]=dis[v]+e2[i].w; |
| if(!flag[e2[i].to]){ |
| q.push(e2[i].to); |
| flag[e2[i].to]=1; |
| } |
| } |
| } |
| } |
| int a\_star(int s,int t,int k){ |
| if(s==t) k++; |
| if(dis[s]==maxn) return -1; |
| priority\_queue<node> q; |
| int cnt=0; |
| node tmp,to; |
| tmp.from=s; |
| tmp.g=0; |
| tmp.f=tmp.g+dis[tmp.from]; |
| q.push(tmp); |
| while(!q.empty()){ |
| tmp=q.top(); |
| q.pop(); |
| if(tmp.from==t) cnt++; |
| if(cnt==k) return tmp.g; |
| for(int i=head1[tmp.from];i;i=e[i].next){ |
| to.from=e[i].to; |
| to.g=tmp.g+e[i].w; |
| to.f=to.g+dis[to.from]; |
| q.push(to); |
| } |
| } |
| return -1; |
| } |
| int main(){ // 该模板能处理带环图 |
| int x,y,z,s,t,k; |
| while(cin>>n>>m) {// 输入n个点 m条边 |
| prepare(); |
| cin>>s>>t>>k; // 输入起点 终点 第k短路 |
| for(int i=1;i<=m;i++) {// 输入边 |
| cin>>x>>y>>z; |
| add\_edge(x,y,z); |
| } |
| spfa(t); |
| int ans=a\_star(s,t,k); // ans 为第k短路的长度 |
| } |
| return 0; |
| } |

* 1. DAG深度优先队列标记

|  |
| --- |
| /\*DAG(有向无环图)的深度优先搜索标记 |
| \* INIT:edge[][]邻接矩阵；pre[], post[], tag全置0 |
| CALL:dfsTag(i, n); pre/post:开始/结束时间\*/ |
| const int V = 1010; |
| int edge[V][V]; |
| int pre[V]; |
| int post[V]; |
| int tag; |
| void dfsTag(int cur, int n){ |
| //vertex:0 ~ n - 1 |
| pre[cur] = ++tag; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (0 == pre[i]){ |
| std::cout << "Three Edge!" << '\n'; |
| dfsTag(i, n); |
| } |
| else{ |
| if (0 == post[i]) std::cout << "Back Edge!" << '\n'; |
| else if (pre[i] > pre[cur]) std::cout << "Down Edge!" << '\n'; |
| else std::cout << "Cross Edge!" << '\n'; |
| }} |
| } |
| post[cur] = ++tag; |
| return ; |
| } |

* 1. 无向图找桥

|  |
| --- |
| /\*无向图找桥 |
| \* INIT: edge[][]邻接矩阵；vis[],pre[],ans[],bridge置0； |
| CALL: dfs(0, -1, 1, n);\*/ |
| const int V = 1010; |
| int bridge; //桥 |
| int edge[V][V]; |
| int ans[V]; |
| int pre[V]; |
| int vis[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex: 0 ~ n - 1 |
| if (bridge) return ; |
| vis[cur] = 1; |
| pre[cur] = ans[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]){ |
| if (pre[i] < ans[cur]) ans[cur] = pre[i]; //back edge |
| } |
| if (0 == vis[i]) { //tree edge |
| dfs(i, cur, dep + 1, n); |
| if (bridge) return ; |
| if (ans[i] < ans[cur]) ans[cur] = ans[i]; |
| if (ans[i] > pre[cur]){bridge = 1; return ;} |
| } |
| } |
| } |
| vis[cur] = 2; |
| } |
| int main(){ |
| // 在这里输入n |
| /\* |
| \* 在这里输入图 |
| \*/ |
| // dfs(0,-1,1,n); 调用函数 |
| } |

* 1. 无向图连通度(割点)

|  |
| --- |
| const int V = 1010; |
| int edge[V][V]; |
| int anc[V]; |
| int pre[V]; |
| int vis[V]; |
| int deg[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex:0 ~ n - 1 |
| int cnt = 0; |
| vis[cur] = 1; |
| pre[cur] = anc[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]) if (pre[i] < anc[cur]) anc[cur] = pre[i]; //back edge |
| if (0 == vis[i]){ //tree edge |
| dfs(i, cur, dep + 1, n); |
| cnt++; //分支个数 |
| if (anc[i] < anc[cur]) anc[cur] = anc[i]; |
| if ((cur == 0 && cnt > 1) || (cnt != 0 && anc[i] >= pre[cur])) deg[cur]++; //link degree of a vertex |
| } |
| } |
| } |
| vis[cur] = 2; |
| } |
| int main(){ |
| /\* INIT: edge[][]邻接矩阵；vis[],pre[],anc[],deg[]置为0； |
| \* CALL: dfs(0, -1, 1, n); |
| \* k = deg[0], deg[i] + 1(i = 1...n - 1)为删除该节点后得到的连通图个数 |
| 注意: 0作为根比较特殊\*/ |
| } |
| const int V = 1010; |
| int edge[V][V]; |
| int anc[V]; |
| int pre[V]; |
| int vis[V]; |
| int deg[V]; |
| void dfs(int cur, int father, int dep, int n){ |
| //vertex:0 ~ n - 1 |
| int cnt = 0; |
| vis[cur] = 1; |
| pre[cur] = anc[cur] = dep; |
| for (int i = 0; i < n; i++){ |
| if (edge[cur][i]){ |
| if (i != father && 1 == vis[i]) if (pre[i] < anc[cur]) anc[cur] = pre[i]; //back edge |
| if (0 == vis[i]){ //tree edge |
| dfs(i, cur, dep + 1, n); |
| cnt++; //分支个数 |
| if (anc[i] < anc[cur]) anc[cur] = anc[i]; |
| if ((cur == 0 && cnt > 1) || (cnt != 0 && anc[i] >= pre[cur])) deg[cur]++; //link degree of a vertex |
| } |
| } |

* 1. 曼哈顿最小生成树

|  |
| --- |
| const int MAXN = 100010; |
| const int INF = 0x3f3f3f3f; |
| struct Point{ |
| int x; |
| int y; |
| int id; |
| }poi[MAXN]; |
| bool cmp(Point a, Point b){ |
| if (a.x != b.x) return a.x < b.x; |
| else return a.y < b.y; |
| } |
| //树状数组，找y - x大于当前的，但是y + x最小的 |
| struct BIT{ |
| int minVal; |
| int pos; |
| void init(){ |
| minVal = INF; |
| pos = -1; |
| } |
| }bit[MAXN]; |
| //所有有效边 |
| struct Edge{ |
| int u; |
| int v; |
| int d; |
| }edge[MAXN << 2]; |
| bool cmpEdge(Edge a, Edge b){ return a.d < b.d;} |
| int tot; |
| int n; |
| int F[MAXN]; |
| int find(int x){ |
| if (F[x] == -1) return x; |
| else return F[x] = find(F[x]); |
| } |
| void addEdge(int u, int v, int d){ |
| edge[tot].u = u; |
| edge[tot].v = v; |
| edge[tot++].d = d; |
| return ; |
| } |
| int lowbit(int x){ return x & (-x);} |
| //更新bit |
| void update(int i, int val, int pos){ |
| while (i > 0){ |
| if (val < bit[i].minVal){ |
| bit[i].minVal = val; |
| bit[i].pos = pos; |
| } |
| i -= lowbit(i); |
| } |
| return ; |
| } |
| //查询[i, m]的最小值位置 |
| int ask(int i, int m){ |
| int minVal = INF; |
| int pos = -1; |
| while (i <= m){ |
| if (bit[i].minVal < minVal){ |
| minVal = bit[i].minVal; |
| pos = bit[i].pos; |
| } |
| i += lowbit(i); |
| } |
| return pos; |
| } |
| int dist(Point a, Point b){ return abs(a.x - b.x) + abs(a.y - b.y);} |
| void ManhattanMinimumSpanningTree(int n, Point p[]){ |
| int a[MAXN], b[MAXN]; |
| tot = 0; |
| for (int dir = 0; dir < 4; dir++){ |
| //变换4种坐标 |
| if (dir == 1 || dir == 3){ |
| for (int i = 0; i < n; i++) std::swap(p[i].x, p[i].y); |
| } |
| else if (dir == 2){ |
| for (int i = 0; i < n; i++) p[i].x = -p[i].x; |
| } |
| std::sort(p, p + n, cmp); |
| for (int i = 0; i < n; i++) a[i] = b[i] = p[i].y - p[i].x; |
| std::sort(b, b + n); |
| int m = (int)(std::unique(b, b + n) - b); |
| for (int i = 1; i <= m; i++) bit[i].init(); |
| for (int i = n - 1; i >= 0; i--){ |
| int pos = (int)(std::lower\_bound(b, b + m, a[i]) - b + 1); |
| int ans = ask(pos, m); |
| if (ans != -1) addEdge(p[i].id, p[ans].id, dist(p[i], p[ans])); |
| update(pos, p[i].x + p[i].y, i); |
| } |
| } |
| return ; |
| } |
| int solve(int k){ |
| ManhattanMinimumSpanningTree(n, poi); |
| memset(F, -1, sizeof(F)); |
| std::sort(edge, edge + tot, cmpEdge); |
| for (int i = 0; i < tot; i++){ |
| int u = edge[i].u; |
| int v = edge[i].v; |
| int tOne = find(u); |
| int tTwo = find(v); |
| if (tOne != tTwo){ |
| F[tOne] = tTwo; |
| k--; |
| if (k == 0) return edge[i].d; |
| } |
| } |
| return -1; |
| } |
| int main(int argc, const char \* argv[]){ |
| //freopen("in.txt", "r", stdin); |
| //freopen("out.txt", "w", stdout); |
| int k; |
| while ((std::cin >> n >> k) && n){ |
| for (int i = 0; i < n; i++){ |
| std::cin >> poi[i].x >> poi[i].y; |
| poi[i].id = i; |
| } |
| std::cout << solve(n - k) << std::endl; |
| } |
| return 0; |
| } |

* 1. 最小生成树(prim)

|  |
| --- |
| #define inf 0x3f3f3f3f |
| typedef struct { |
| int point; |
| int value; |
| }node; |
| int N; |
| int sum; |
| vector<node>point[1000]; |
| int hashtable[1000] = {0}; |
| int num[1000]; |
| void init() { |
| int i; |
| for (i = 0; i < N; i++) num[i] = inf; |
| num[0] = 0; |
| sum = 0; |
| } |
| void prim() { |
| int n = N,i,k,min,min\_num; |
| while (n--) { |
| min=-1; |
| while (n--) { |
| min\_num = inf; |
| for (i = 0; i < N; i++) { |
| if (hashtable[i] == 0 && num[i] != inf) { |
| if (num[i] < min\_num) { |
| Min=i; |
| min\_num = num[i]; |
| } |
| } |
| } |
| sum += num[min]; |
| hashtable[min] = 1; |
| if (min == -1) return; |
| for (k = 0; k < point[min].size(); k++) { |
| int v = point[min][k].point; |
| int value = point[min][k].value; |
| if (num[v] > value && hashtable[v] == 0) num[v] = value; |
| } |
| } |
| } |
| int main() { |
| int M; |
| scanf("%d", &N); |
| scanf("%d", &M); |
| int x, y, i, check, value; |
| for (i = 0; i < M; i++) { |
| scanf("%d%d%d", &x, &y, &value); |
| node new\_node = { y,value }; |
| point[x].push\_back(new\_node); |
| new\_node.point = x; |
| point[y].push\_back(new\_node); |
| } |
| init(); |
| prim(); |
| printf("%d",sum); |
| scanf("%d", &check); |
| } |

* 1. 次小生成树

|  |
| --- |
| int g[M][M],path[M][M];//path求的是i到j最大的边权 |
| int dist[M],pre[M],vis[M]; |
| bool used[M][M];//是否在最小生成树中 |
| int n,m,mst; |
| void init(){ |
| for(int i=0;i<=n;i++) |
| for(int j=i+1;j<=n;j++) g[i][j]=g[j][i]=inf; |
| } |
| int prime() { |
| int mst=0; |
| memset(path,0,sizeof(path)); |
| memset(vis,0,sizeof(vis)); |
| memset(used,0,sizeof(used)); |
| vis[1]=1; |
| for(int i=1;i<=n;i++){ |
| dist[i]=g[1][i]; |
| pre[i]=1; |
| } |
| for(int i=1;i<n;i++) { |
| int u=-1; |
| for(int j=1;j<=n;j++){ |
| if(!vis[j]) if(u==-1||dist[j]<dist[u]) u=j; |
| } |
| used[u][pre[u]]=used[pre[u]][u]=true;//加入mst |
| mst+=g[pre[u]][u]; |
| vis[u]=1; |
| for(int j=1;j<=n;j++) |
| { |
| if(vis[j]&&j!=u)//从u到j这条路径上最大边的权值 |
| path[j][u]=path[u][j]=max(path[j][pre[u]],dist[u]); |
| if(!vis[j]) |
| if(dist[j]>g[u][j]){//更新相邻节点的距离 |
| dist[j]=g[u][j]; |
| pre[j]=u;//记录他的前驱 |
| } |
| } |
| } |
| return mst; |
| } |
| int second\_tree(){//求次小生成树 |
| int res=inf; |
| for(int i=1;i<=n;i++) |
| for(int j=1;j<=n;j++) |
| if(i!=j&&!used[i][j]) |
| res=min(res,mst-path[i][j]+g[i][j]);//删除树上权值最大的路径并且加上这条路径其它边 |
| return res; |
| } |
| int main() { |
| int t; |
| scanf("%d",&t); |
| while(t--) { |
| scanf("%d%d",&n,&m); |
| init(); |
| mst=prime();//最小生成树 |
| int second\_mst=second\_tree();//次小生成树 |
| } |
| } |

* 1. 欧拉路径

|  |
| --- |
| /\*SGU 101 \*/ |
| struct Edge{ |
| int to; |
| int next; |
| int index; |
| int dir; |
| bool flag; |
| } edge[220]; |
| int head[10]; //前驱 |
| int tot; |
| void init(){ |
| memset(head, -1, sizeof((head))); |
| tot = 0; |
| } |
| void addEdge(int u, int v, int index){ |
| edge[tot].to = v; |
| edge[tot].next = head[u]; |
| edge[tot].index = index; |
| edge[tot].dir = 0; |
| edge[tot].flag = false; |
| head[u] = tot++; |
| edge[tot].to = u; |
| edge[tot].next = head[v]; |
| edge[tot].index = index; |
| edge[tot].dir = 1; |
| edge[tot].flag = false; |
| head[v] = tot++; |
| return ; |
| } |
| int du[10]; |
| std::vector<int>ans; |
| void dfs(int u){ |
| for (int i = head[u]; i != -1; i = edge[i].next){ |
| if (!edge[i].flag){ |
| edge[i].flag = true; |
| edge[i ^ 1].flag = true; |
| dfs(edge[i].to); |
| ans.push\_back(i); //容器尾部插入i |
| } |
| } |
| return ; |
| } |
| int main(){ |
| //freopen("in.txt", "r", stdin); |
| //freopen("out.txt", "w", stdout); |
| int n; |
| while (std::cin >> n) { |
| init(); |
| int u, v; |
| memset(du, 0, sizeof(du)); |
| for (int i = 1; i <= n; i++){ |
| std::cin >> u >> v; |
| addEdge(u, v, i); |
| du[u]++; |
| du[v]++; |
| } |
| int s = -1; |
| int cnt = 0; |
| for (int i = 0; i <= 6; i++){ |
| if (du[i] & 1){ |
| cnt++; |
| s = i; |
| } |
| if (du[i] > 0 && s == -1) s = i; |
| } |
| if (cnt != 0 && cnt != 2){ |
| std::cout << "No solution" << '\n'; |
| continue; |
| } |
| ans.clear(); |
| dfs(s); |
| if (ans.size() != n){ |
| std::cout << "No solution" << '\n'; |
| continue; |
| } |
| for (int i = 0; i < ans.size(); i++){ |
| printf("%d ", edge[ans[i]].index); |
| if (edge[ans[i]].dir == 0) std::cout << "-" << '\n'; |
| else std::cout << "+" << '\n'; |
| } |
| } |
| return 0; |
| } |

* 1. 迪杰斯特拉优化模板

|  |
| --- |
| typedef struct { |
| int point;//能够到达的点 |
| int value;//第一尺度 |
| int cost; //第二尺度 |
| }node; |
| int N; |
| int num[1001];//第一尺度的最小值储存单位 |
| int cost[1001];//第二尺度的最小值储存单位 |
| int hashtable[2000] = { 0 };//哈希表，判断点是否访问过 |
| vector<node>point[2000];//邻接表 |
| void init(int start) {//初始化 |
| int i; |
| for (i = 0; i < N; i++) { |
| num[i] = Max; |
| cost[i] = Max; |
| } |
| num[start] = 0; |
| cost[start] = 0; |
| } |
| void djistra(int start) { |
| init(start); |
| int i; |
| int min = 0; |
| int min\_num; |
| int check; |
| while (1) { |
| min\_num = Max; |
| check = 0; |
| for (i = 0; i < N; i++) {//找出当前离起点最近的且未访问过的节点 |
| if (hashtable[i] == 0 && num[i] != Max) { |
| check = 1; |
| if (num[i] < min\_num) { |
| min = i; |
| min\_num = num[i]; |
| } |
| } |
| } |
| if (check == 0) |
| return;//如果没有就说明优化距离结束 |
| hashtable[min] = 1; |
| for (i = 0; i < point[min].size(); i++) { |
| if (hashtable[point[min][i].point] == 0) { |
| if (num[point[min][i].point] > point[min][i].value + num[min]) {//以第一尺度为标准，先计算出第一尺度的最小值下的第二尺度的值 |
| num[point[min][i].point] = point[min][i].value + num[min]; |
| cost[point[min][i].point] = point[min][i].cost + cost[min]; |
| } |
| else if (num[point[min][i].point] == point[min][i].value + num[min]) {//以计算出的第二尺度值为标准，计算出第二尺度的最小值 |
| if (cost[point[min][i].point] > point[min][i].cost + cost[min]) |
| cost[point[min][i].point] = point[min][i].cost + cost[min]; |
| } |
| } |
| } |
| } |
| } |
| int main() { //点标号0开头 |
| int M, start, end; |
| int x, y, value,cost\_value; |
| while (scanf("%d%d", &N, &M) && (N != 0 || M != 0) ) { |
| while (M--) { |
| scanf("%d%d%d%d", &x, &y, &value, &cost\_value); |
| node new\_node = { y,value,cost\_value }; |
| point[x].push\_back(new\_node);//无向边 |
| new\_node.point = x; |
| point[y].push\_back(new\_node); |
| } |
| scanf("%d%d", &start, &end); |
| djistra(start); |
| printf("%d %d\n", num[end], cost[end]); |
| } |
| } |

* 1. Dfja

1. 字符串
   1. a字符串增删改变b字符串的最小操作数
   2. KMP
   3. 扩展kmp
   4. 最短公共祖先
   5. Kdaj
2. Fsjla