6.1 模运算

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21 | #include<bits/stdc++.h>  using namespace std;  typedef long long ll;  ll mul(ll a,ll b,ll m){ //乘法取模：a\*b mod m  a = a%m; //先取模，非常重要，能在一定程度上防止第10行a+a溢出  b = b%m;  ll res = 0;  while(b>0){ //递归思想  if(b&1) res=(res+a)% m; //用b&1判断b的奇偶  a=(a+a)% m; //需要保证a+a=2a不能溢出，否则答案错误  b>>=1;  }  return res;  }  int main(){  ll a = 0x7877665544332211;  ll b = 0x7988776655443322;  ll m = 0x998776655443322; //把m改成比a大的0x7977665544332211，mul()也会出错  cout << (a%m)\*(b%m)%m <<endl; //输出 145407782617487436，错误  cout << mul(a,b,m); //输出 411509877096934416，正确  } |

6.2 快速幂

|  |  |  |
| --- | --- | --- |
| 1  2  3  4  5  6  7  8  9 | int fastPow(int a, int n){ //计算an  int ans = 1; //用ans返回结果  while(n) { //把n看成二进制，逐个处理它的最后一位  if(n & 1) ans \*= a; //如果n的最后一位是1，表示这个地方需要乘  a \*= a; //递推：a2 --> a4 --> a8--> a16...  n >>= 1; //n右移一位，把刚处理过的n的最后一位去掉  }  return ans;  } | |
| 1  2  3  4  5  6  7  8  9  10  11 | | typedef long long ll; //变量改用较大的long long  ll fastPow(ll a, ll n, ll mod){  ll ans = 1;  a %= mod; //有一定作用，能在一定程度上防止下面的a\*a越界  while(n) {  if(n & 1) ans = (ans\*a) % mod; //取模  a = (a\*a) % mod; //取模  n >>= 1;  }  return ans;  } | |

6.3.2 矩阵快速幂

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22 | struct matrix{ int m[N][N]; }; //定义矩阵，常数N是矩阵的行数和列数  matrix operator \* (const matrix& a, const matrix& b){ //重载\*为矩阵乘法。注意const  matrix c;  memset(c.m, 0, sizeof(c.m)); //清零  for(int i=0; i<N; i++)  for(int j=0; j<N; j++)  for(int k = 0; k<N; k++)  //c.m[i][j] += a.m[i][k] \* b.m[k][j]; //不取模  c.m[i][j] = (c.m[i][j] + a.m[i][k] \* b.m[k][j]) % mod;//取模  return c;  }  matrix pow\_matrix(matrix a, int n){ //矩阵快速幂，代码和普通快速幂几乎一样  matrix ans;  memset(ans.m,0,sizeof(ans.m));  for(int i=0;i<N;i++) ans.m[i][i] = 1; //初始化为单位矩阵，类似普通快速幂的ans=1  while(n) {  if(n&1) ans = ans \* a; //不能简写为ans \*= a，这里的\*重载了  a = a \* a;  n>>=1;  }  return ans;  } |

6.3.4 矩阵乘法与路径问题

|  |  |  |
| --- | --- | --- |
| 例6-3. Cow Relays poj 3613 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | #include<cstdio>  #include<algorithm>  #include<cstring>  const int INF=0x3f;  const int N=120;  int Hash[1005],cnt=0; //用于离散化  struct matrix{int m[N][N]; }; //定义矩阵  matrix operator \*(const matrix& a, const matrix& b){ //定义广义矩阵乘法  matrix c;  memset(c.m,INF,sizeof c.m);  for(int i=1;i<=cnt;i++) //i、j、k可以颠倒，因为对c来说都一样  for(int j=1;j<=cnt;j++)  for(int k=1;k<=cnt;k++)  c.m[i][j] = std::min(c.m[i][j], a.m[i][k] + b.m[k][j]);  return c;  }  matrix pow\_matrix(matrix a, int n){ //矩阵快速幂，几乎就是标准的快速幂写法  matrix ans = a; //矩阵初值ans = M^1  n--; //上一行ans= M^1多了一次  while(n) { //矩阵乘法：M^n  if(n&1) ans = ans \* a;  a = a \* a;  n>>=1;  }  return ans;  }  int main(){  int n,t,s,e; scanf("%d%d%d%d",&n,&t,&s,&e);  matrix a; //用矩阵存图  memset(a.m,INF,sizeof a.m);  while(t--){  int u,v,w; scanf("%d%d%d",&w,&u,&v);  if(!Hash[u]) Hash[u] = ++cnt; //对点离散化. cnt就是新的点编号  if(!Hash[v]) Hash[v] = ++cnt;  a.m[Hash[u]][Hash[v]] = a.m[Hash[v]][Hash[u]] = w;  }  matrix ans = pow\_matrix(a,n);  printf("%d",ans.m[Hash[s]][Hash[e]]);  return 0;  } |

6.4.2 高斯-约当消元法

|  |  |  |
| --- | --- | --- |
| 例6-4. 高斯消元法 洛谷P3389 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | //改写自：https://www.luogu.com.cn/blog/tbr-blog/solution-p3389  #include<bits/stdc++.h>  using namespace std;  double a[105][105];  double eps = 1e-7;  int main(){  int n; scanf("%d",&n);  for(int i=1;i<=n;++i)  for(int j=1;j<=n+1;++j) scanf("%lf",&a[i][j]);  for(int i=1;i<=n;++i){ //枚举列  int max=i;  for(int j=i+1;j<=n;++j) //选出该列最大系数，真实目的是选一个非0系数  if(fabs(a[j][i])>fabs(a[max][i])) max=j;  for(int j=1;j<=n+1;++j) swap(a[i][j],a[max][j]); //移到前面  if(fabs(a[i][i]) < eps){ //对角线上的主元系数等于0,说明没有唯一解  puts("No Solution");  return 0;  }  for(int j=n+1;j>=1;j--) a[i][j]= a[i][j]/a[i][i]; //把这一行的主元系数变为1  for(int j=1;j<=n;++j){ //消去主元所在列的其他行的主元  if(j!=i) {  double temp=a[j][i]/a[i][i];  for(int k=1;k<=n+1;++k) a[j][k] -= a[i][k]\*temp;  }  }  }  for(int i=1;i<=n;++i) printf("%.2f\n",a[i][n+1]); //最后得到简化阶梯矩阵  return 0;  } | |

6.5.3 线性基的应用

|  |  |  |  |
| --- | --- | --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | | //不用高斯消元求线性基  #include<bits/stdc++.h>  using namespace std;  typedef long long ll;  const int M=63;  ll p[M]; //线性基  bool zero;  void Insert(ll x){  for(int i=M;i>=0;i--)  if(x>>i == 1) //x的最高位  if(p[i]==0){ p[i]=x; return; } //P[i]还没有，直接让P[i] = x  else x^=p[i]; // P[i]已经有了，逐个异或  zero = true; //A有异或和为0的组合  }  ll qmax(){  ll ans = 0;  for( int i=M;i>=0;i--) ans = max(ans,ans^p[i]);  return ans;  }  int main(){  ll x; int n; scanf("%d",&n);  for(int i=1;i<=n;i++) scanf("%lld",&x), Insert(x);  printf("%lld\n",qmax());  return 0;  } | |
| 例6-8. XOR Hdu 3949 | | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51 | | //用高斯消元求线性基  #include<bits/stdc++.h>  #define N 10100  using namespace std;  typedef long long ll;  int n;  bool zero; //消元后是否产生全0的行  ll a[N];  void Gauss(){ //高斯消元求线性基  int i,k=1; //k标记当前第几行  ll j = (ll)1<<62; //注意不是63  for(;j;j>>=1){  for(i=k;i<=n;i++)  if(a[i]&j) break; //找到第j位是1的a[]  if(i > n) continue; //没有第j位是1的a[]  swap(a[i],a[k]); //把这一行换到上面  for(i=1;i<=n;i++) //生成简化阶梯矩阵  if(i != k && a[i]&j) a[i]^=a[k];  k++;  }  k--;  if(k!=n) zero = true;  else zero = false;  n = k; //线性基中元素的个数  }  ll Query(ll k){ //第k小异或和  ll ans=0;  if(zero) k--;  if(!k) return 0;  for(int i=n;i;i--){  if(k&1) ans^=a[i];  k >>= 1;  }  if(k) return -1;  return ans;  }  int main(){  int cnt=0;  int T; cin>>T;  while(T--){  printf("Case #%d:\n",++cnt);  cin>>n;  for(int i=1;i<=n;i++) scanf("%lld",&a[i]);  Gauss();  int q; cin>>q;  while(q--){  ll k; scanf("%lld",&k);  printf("%lld\n", Query(k) );  }  }  } | | |

6.6.1 二分法与0/1分数规划

|  |
| --- |
| 例6-9. Dropping tests poj 2976 |

代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | #include <stdio.h>  #include <algorithm>  using namespace std;  struct Pair{ int a, b; double y;} p[1005];  bool cmp(Pair a, Pair b){ return a.y > b.y; }  int n, k;  bool check(double x) {  for(int i=0; i<n; i++) p[i].y = p[i].a \* 1.0 - x \* p[i].b; //计算y=a-xb  sort(p, p + n, cmp); //按y值排序，非常重要  double f = 0;  for (int i=0; i<k; i++) f += p[i].y; //对前k个直线的y值求和  return f < 0; // f < 0：竖线在M的右侧  }  int main() {  while (scanf("%d%d", &n, &k) == 2 && n + k) {  k = n - k; //改为选出k对  for (int i = 0; i < n; i++) scanf("%d", &p[i].a);  for (int i = 0; i < n; i++) scanf("%d", &p[i].b);  double L = 0, R = 0;  for (int i = 0; i < n; i++) R += p[i].a; //R的初值  for (int i = 0; i < 50; i++) { //二分50次，本题足够了  double mid = L+(R-L)/2;  if (check(mid)) R = mid; //竖线在M的右侧，需要左移  else L = mid; //竖线在M的左侧，需要右移  }  printf("%d\n", (int)(100 \* (L + 0.005))); //四舍五入  }  return 0;  } |

6.6.2 应用场景

|  |  |  |
| --- | --- | --- |
| 例6-10. Talent Show G洛谷 P4377 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | //改写自: www.luogu.com.cn/blog/yjlakioi/post-18-open-g-t3-post  #include<bits/stdc++.h>  using namespace std;  const int INF=0x3f3f3f3f,N=255,WW=1005;  int n,W;  struct{int w, t; double y;}cow[N];  double dp[WW]; //dp[i],背包容量为i时最大的价值(y值之和)  bool check(double x){ // 0/1背包  int i,j;  for(i=1;i<=n;i++) cow[i].y=(double)cow[i].t-x\*cow[i].w;  for(i=1;i<=W;i++) dp[i]=-INF; //初始化为负无穷小  dp[0] = 0; //背包容量为0时价值为0  for(i=1;i<=n;i++)  for(j=W;j>=0;j--){ // 滚动数组  if(j+cow[i].w>=W) dp[W]=max(dp[W],dp[j]+cow[i].y); //大于W时按W算  else dp[j+cow[i].w]=max(dp[j+cow[i].w],dp[j]+cow[i].y);  }  return dp[W]<0; // dp[W] < 0，x大了；= dp[W] ≥ 0，x小了  }  int main(){  cin>>n>>W;  for(int i=1;i<=n;i++) cin>>cow[i].w>>cow[i].t;  double L=0,R=0;  for (int i = 1; i <= n; i++) R += cow[i].t; //R的初值  for(int i=0;i<50;i++){  double mid = L+(R-L)/2;  if(check(mid)) R = mid; //缩小  else L = mid; //放大  }  cout<<(int)(L\*1000)<<endl;  return 0;  } | |

6.8.2 扩展欧几里得算法与二元线性丢番图方程的解

|  |  |
| --- | --- |
| 例6-17. 青蛙的约会 洛谷 P1516 | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17 | #include<bits/stdc++.h>  using namespace std;  #define ll long long  ll extend\_gcd(ll a,ll b,ll &x,ll &y){  if(b == 0){ x=1; y=0; return a;}  ll d = extend\_gcd(b,a%b,y,x);  y -= a/b \* x;  return d;  }  int main(){  ll n,m,x,y,L; cin>>x>>y>>m>>n>>L;  ll a=n-m,c=x-y;  if(a<0){ a=-a; c=-c;} //处理负数  ll d = extend\_gcd(a,L,x,y);  if(c%d != 0) cout<<"Impossible"; //判断方程有无解。  else cout<<((x\*(c/d))%(L/d)+(L/d))%(L/d); //x的最小整数解  } | |

6.9.3 逆

|  |  |  |
| --- | --- | --- |
| 例6-19. **同余方程（求逆）** 洛谷 P1082 | | |
| 1  2  3  4  5  6  7  8  9  10 | long long mod\_inverse(long long a, long long m){ //求逆  long long x,y;  extend\_gcd(a,m,x,y);  return (x%m + m) % m; //保证返回最小正整数  }  int main(){  long long a,m; cin >> a >>m;  cout << mod\_inverse(a,m);  return 0;  } |

|  |  |  |
| --- | --- | --- |
| 例6-23. Detachment hdu 5976 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | #include<bits/stdc++.h>  using namespace std;  #define ll long long  const int N = 1e5; //分解的数不会超过50000个，请自己分析  const int mod = 1e9 + 7;  ll sum[N], mul[N]; //前缀和、连续积  ll fast\_pow(ll x,ll y,int m){ //快速幂取模：x^y mod m  ll res = 1;  while(y) {  if(y&1) res\*=x, res%=m;  x = (x\*x) % m;  y>>=1;  }  return res;  }  long long mod\_inverse(long long a,long long mod){ //费马小定理求逆  return fast\_pow(a,mod - 2,mod);  }  void init(){ //预计算前缀和、连续积  sum[1] = 0; mul[1] = 1;  for(int i=2; i<=N; i++){  sum[i] = sum[i-1] + i; //计算前缀和  mul[i] = (i\*mul[i-1]) % mod; //计算连续积  }  }  int main(){  init();  int T; scanf("%d",&T);  while(T--){  int x; scanf("%d",&x);  if( x == 1) {puts("1"); continue;} //特殊情况  int k = upper\_bound(sum+1,sum+1+N,x)-sum-1; //分解成k个数  int m = x - sum[k]; //余数  ll ans;  if(k==m) ans = mul[k] \* mod\_inverse(2,mod) %mod \* (k+2) % mod; //第2种情况  else ans = mul[k+1] \* mod\_inverse(k-m+1,mod) %mod % mod; //第1种情况  printf("%lld\n",ans);  }  return 0;  } |

6.9.4 同余方程组

|  |  |  |
| --- | --- | --- |
| 例6-24. 扩展中国剩余定理 洛谷P4777 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | //改写自 https://www.luogu.com.cn/problem/solution/P4777  #include<bits/stdc++.h>  using namespace std;  typedef long long ll;  const int N = 100010;  int n;  ll ai[N], mi[N];  ll mul(ll a,ll b,ll m){ //乘法取模：a\*b % m  ll res=0;  while(b>0){  if(b&1) res=(res+a) % m;  a=(a+a) % m;  b>>=1;  }  return res;  }  ll extend\_gcd(ll a,ll b,ll &x,ll &y){ //扩展欧几里得  if(b == 0){ x=1; y=0; return a;}  ll d = extend\_gcd(b,a%b,y,x);  y -= a/b \* x;  return d;  }  ll excrt(){ //求解同余方程组，返回最小正整数解  ll x,y;  ll m1 = mi[1], a1 = ai[1]; //第1个等式  ll ans = 0;  for(int i=2;i<=n;i++){ //合并每2个等式  ll a2 = ai[i], m2 = mi[i]; // 第2个等式  //合并为：aX + bY = c  ll a = m1, b = m2, c = (a2 - a1%m2 + m2) % m2;  //下面求解 aX + bY = c  ll d = extend\_gcd(a,b,x,y); //用扩展欧几里得求x0  if(c%d != 0) return -1; //无解  x = mul(x,c/d,b/d); //aX + bY = c 的特解t，最小值  ans = a1 + x\* m1; //代回原第1个等式，求得特解x'  m1 = m2/d\*m1; //先除再乘，避免越界。合并后的新m1  ans = (ans%m1 + m1) % m1; //最小正整数解  a1 = ans; //合并后的新a1  }  return ans;  }  int main(){  scanf("%d", &n);  for(int i=1;i<=n;++i) scanf("%lld%lld",&mi[i],&ai[i]);  printf("%lld",excrt());  return 0;  } |

6.10.2 大素数的判定

|  |  |  |
| --- | --- | --- |
| 例6-25. How many prime numbers hdu 2138 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49 | #include <bits/stdc++.h>  typedef long long LL;  LL fast\_pow(LL x,LL y,int m){ //快速幂取模：x^y mod m  LL res = 1;  x %= m;  while(y) {  if(y&1) res = (res\*x) % m;  x = (x\*x) % m;  y>>=1;  }  return res;  }  bool witness(LL a, LL n){ // Miller-Rabin素性测试。返回true表示n是合数  LL u = n-1; //注意，u的意义是：n-1的二进制去掉末尾0  int t = 0; // n-1的二进制，是奇数u的二进制，后面加t个零  while(u&1 == 0) u = u>>1, t++; // 整数n-1末尾0的个数，就是t  LL x1, x2;  x1 = fast\_pow(a,u,n); // 先计算 a^u mod n  for(int i=1; i<=t; i++) { // 做t次平方取模  x2 = fast\_pow(x1,2,n); // x1^2 mod n  if(x2 == 1 && x1 != 1 && x1 != n-1) return true; //用推论判断  x1 = x2;  }  if(x1 != 1) return true; //用费马测试判断是否为合数：*a*n-1≡1(mod *n*)不成立，是合数  return false;  }  int miller\_rabin(LL n,int s){ //对n做s次测试  if(n<2) return 0;  if(n==2) return 1; //2是素数  if(n % 2 == 0 ) return 0; //偶数  for(int i = 0;i < s && i < n;i++){ //做s次测试  LL a = rand() % (n - 1) + 1; //基值a是随机数  if(witness(a,n)) return 0; //n是合数，返回0  }  return 1; //n是素数，返回1  }  int main(){  int m;  while(scanf("%d",&m) != EOF){  int cnt = 0;  for(int i = 0; i < m; i++){  LL n; scanf("%lld",&n);  int s = 50; //做s次测试  cnt += miller\_rabin(n,s);  }  printf("%d\n",cnt);  }  return 0;  } |

6.10.3 素数筛

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15 | const int N = 1e7; //定义空间大小，1e7约10M  int prime[N+1]; //存放素数，它记录visit[i] = false的项  bool visit[N+1]; //true表示被筛掉，不是素数  int E\_sieve(int n) { //埃氏筛法，计算[2, n]内的素数  int k=0; //统计素数个数  for(int i=0; i<=n; i++) visit[i]= false; //初始化  for(int i=2; i<=n; i++) { //从第一个素数2开始。可优化（1）  if(!visit[i]) {  prime[k++] = i; //i是素数，存储到prime[]中  for(int j=2\*i; j<=n; j+=i) //i的倍数，都不是素数。可优化（2）  visit[j] = true; //标记为非素数，筛掉  }  }  return k; //返回素数个数  } |

。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | int E\_sieve(int n) {  for(int i = 0; i <= n; i++) visit[i]= false;  for(int i = 2; i\*i <= n; i++) //筛掉非素数。改为i<=sqrt(n)，计算更快  if(!visit[i])  for(int j=i\*i; j<=n; j+=i) visit[j] = true; //标记为非素数  //下面记录素数  int k=0; //统计素数个数  for(int i = 2; i <= n; i++)  if(!visit[i]) prime[k++] = i; //存储素数  return k;  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | int prime[N]; //保存质数，为节约空间，可以适当减小  bool vis[N]; //记录是否被筛  int euler\_sieve(int n){ //欧拉筛。返回质数的个数。  int cnt = 0; //记录质数个数  memset(vis,0,sizeof(vis));  memset(prime,0,sizeof(prime));  for(int i=2;i<=n;i++){ //检查每个数，筛去其中的合数  if(!vis[i]) prime[cnt++]=i; //如果没有筛过，是质数，记录。第一个质数是2  for(int j=0; j<cnt; j++){ //用已经得到的质数去筛后面的数  if(i\*prime[j] >n) break; //只筛小于等于n的数  vis[i\*prime[j]]=1; //**关键1**。用x的最小质因数筛去x  if(i%prime[j]==0) break; //**关键2**。如果不是这个数的最小质因子，结束  }  }  return cnt; //返回小于等于n的质数的个数  } |

6.10.4 质因数分解

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | int prime[N]; //记录质数  int vis[N]; //记录最小质因子  int euler\_sieve(int n){  int cnt=0;  memset(vis,0,sizeof(vis));  memset(prime,0,sizeof(prime));  for(int i=2;i<=n;i++){  if(!vis[i]){ vis[i]=i; prime[cnt++]=i;} //vis[]记录最小质因子  for(int j=0; j<cnt; j++){  if(i\*prime[j] >n) break;  vis[i\*prime[j]] = prime[j]; //vis[]记录最小质因子  if(i%prime[j]==0) break;  }  }  return cnt;  } |

一

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13 | //代码改写自《算法竞赛进阶指南》河南电子音像出版社，李煜东，137页  int p[20]; //p[]记录因子，p[1]是最小因子。一个int数的质因子最多有十几个  int c[40]; //c[i]记录第i个因子的个数。一个因子的个数最多有三十几个  int factor(int n){  int m = 0;  for(int i = 2; i <= sqrt(n); i++)  if(n%i == 0){  p[++m] = i, c[m] = 0;  while(n%i == 0) n/=i, c[m]++; //把n中重复的因子去掉  }  if(n>1) p[++m] = n, c[m] = 1; //没有被除尽，是素数  return m; //共m个  } |

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41 | //poj 1811部分代码：输入一个整数n，2<=N<2^54，判断它是否为素数，如果不是，输出最小质因子。  typedef long long ll;  ll Gcd (ll a,ll b){return b? Gcd(b, a%b):a;}  ll mult\_mod (ll a,ll b,ll n){ //返回(a\*b) mod n  a %= n, b %= n;  ll ret=0;  while (b){  if (b&1){  ret += a;  if (ret >= n) ret -= n;  }  a<<=1;  if (a>=n) a -= n;  b>>=1;  }  return ret;  }  ll pollard\_rho (ll n){ //返回一个因子，不一定是质因子  ll i=1, k=2;  ll c = rand()%(n-1)+1;  ll x = rand()%n;  ll y = x;  while (true){  i++;  x = (mult\_mod(x,x,n)+c) % n; //(x\*x) mod n  ll d = Gcd(y>x?y-x:x-y, n); //**重要**：保证gcd的参数大于等于0  if (d!=1 && d!=n) return d;  if (y==x) return n; //已经出现过，直接返回  if (i==k) { y=x; k=k<<1;}  }  }  void findfac (ll n){ //找所有的素因子  if (miller\_rabin(n)) { //用miller\_rabin判断是否为素数  factor[tol++] = n; //存素因子  return;  }  ll p = n;  while (p>=n) p=pollard\_rho(p); //找到一个因子  findfac(p); //继续寻找更小的因子  findfac(n/p);  } |

6.13.2 求欧拉函数的通解公式

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | int euler(int n){  int ans = n;  for(int p = 2; p\*p <= n; ++ p){ //试除法：检查从2到sqrt(n)的每个数  if(n%p == 0){ //能整除，p是一个因子，而且是质因子，请思考  ans = ans/p\*(p-1); //求欧拉函数的通式  while(n%p == 0) //去掉这个因子的幂，并使得下一个p是质因子  n /= p; //减小了n  }  }  if(n != 1) ans = ans/n\*(n-1); //情况(1)：n是一个质数，没有执行上面的分解  return ans;  } |

6.13.3 线性筛（欧拉筛）求1～n内所有的欧拉函数

|  |  |  |
| --- | --- | --- |
| 例6-29. **仪仗队** 洛谷P2158 | | |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37 | #include<bits/stdc++.h>  using namespace std;  const int N = 50000;  int vis[N]; //记录是否被筛；或者用于记录最小质因子  int prime[N]; //记录质数  int phi[N]; //记录欧拉函数  int sum[N]; //计算欧拉函数的和  void get\_phi(){ //模板：求1～N范围内的欧拉函数  phi[1]=1;  int cnt=0;  for(int i=2;i<N;i++) {  if(!vis[i]) {  vis[i]=i; //vis[i]=1; //二选一：前者记录最小质因子，后者记录是否被筛  prime[cnt++]=i; //记录质数  phi[i]=i-1; //**情况(1)**：i是质数，它欧拉函数值=i-1  }  for(int j=0;j<cnt;j++) {  if(i\*prime[j] > N) break;  vis[i\*prime[j]] = prime[j]; //vis[i\*prime[j]]=1;  //二选一：前者记录最小质因子，后者记录是否被筛  if(i%prime[j]==0){ //prime[j]是最小质因子  phi[i\*prime[j]]=phi[i]\*prime[j]; //**情况(2)**：i是prime[j]的k次方  break;  }  phi[i\*prime[j]]=phi[i]\*phi[prime[j]];//**情况(3)**：i和prime[j]互素，递推出i\*prime[j]  }  }  }  int main(){  get\_phi(); //计算所有的欧拉函数  sum[1]=1;  for(int i=2;i<=N;i++) sum[i]=sum[i-1]+phi[i]; //打表计算欧拉函数的和  int n; scanf("%d",&n);  if(n==1) printf("0\n");  else printf("%d\n",2\*sum[n-1]+1);  return 0;  } |

6.14 整除分块（数论分块）

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | #include<bits/stdc++.h>  using namespace std;  int main(){  long long n,L,R,ans=0;  cin >> n;  for(L=1;L<=n;L=R+1){  R = n/(n/L); //计算R，让分块右移  ans += (R-L+1)\*(n/L); //分块求和  cout << L <<"-"<< R <<": "<< n/R << endl; //打印分块的情况  }  cout << ans; //打印和  } |

6.16 莫比乌斯函数和莫比乌斯反演

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16 | bool vis[N];  int prime[N];  int Mob[N];  void Mobius\_sieve(){  int cnt = 0;  vis[1] = 1;  Mob[1] = 1;  for(int i = 2; i <= N; i++){  if(!vis[i]) prime[cnt++] = i, Mob[i] = - 1;  for(int j = 0; j < cnt && 1LL \* prime[j] \* i <= N; j++){  vis[prime[j] \* i] = 1;  Mob[i \* prime[j]] = (i % prime[j] ? -Mob[i]: 0);  if(i % prime[j] == 0) break;  }  }  } |

6.17.4 杜教筛模板代码

下面给出洛谷P4213的代码

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71 | //代码改写自：https://blog.csdn.net/KIKO\_caoyue/article/details/100061406  #include <bits/stdc++.h>  using namespace std;  typedef long long ll;  const int N = 5e6+7; //超过n^(2/3)，够大了  int prime[N]; //记录质数  bool vis[N]; //记录是否被筛；  int mu[N]; //莫比乌斯函数值  ll phi[N]; //欧拉函数值  unordered\_map<int,int> summu; //莫比乌斯函数前缀和  unordered\_map<int,ll> sumphi; //欧拉函数前缀和  void init(){ //线性筛预计算一部分答案  int cnt = 0;  vis[0] = vis[1] = 1;  mu[1] = phi[1] = 1;  for(int i=2;i<N;i++){  if(!vis[i]){  prime[cnt++] = i;  mu[i] = -1;  phi[i] = i-1;  }  for(int j=0;j<cnt && i\*prime[j]<N;j++){  vis[i\*prime[j]] = 1;  if(i%prime[j]){  mu[i\*prime[j]] = -mu[i];  phi[i\*prime[j]] = phi[i]\*phi[prime[j]];  }  else{  mu[i\*prime[j]] = 0;  phi[i\*prime[j]] = phi[i]\*prime[j];  break;  }  }  }  for(int i=1;i<N;i++){ //最后，mu[]和phi[]改为记录1~N的前缀和。  mu[i] += mu[i-1];  phi[i] += phi[i-1];  }  }  int gsum(int x){ // g(i)的前缀和  return x;  }  ll getsmu(int x){  if(x < N) return mu[x]; //预计算  if(summu[x]) return summu[x]; //记忆化  ll ans = 1; //杜教筛公式中的 1  for(ll l=2,r;l<=x;l=r+1){ //用整除分块计算杜教筛公式  r = x/(x/l);  ans -= (gsum(r)-gsum(l-1))\*getsmu(x/l);  }  return summu[x] = ans/gsum(1);  }  ll getsphi(int x){  if(x < N) return phi[x];  if(sumphi[x]) return sumphi[x]; //记忆化，每个sumphi[x]只用算一次  ll ans = x\*((ll)x+1)/2; //杜教筛公式中的 n(n+1)/2  for(ll l=2,r;l<=x;l=r+1){ //用整除分块计算杜教筛公式，这里算 sqrt(x)次  r = x/(x/l);  ans -= (gsum(r)-gsum(l-1))\*getsphi(x/l);  }  return sumphi[x] = ans/gsum(1);  }  int main(){  init(); //用线性筛预计算一部分  int t; scanf("%d",&t);  while(t--){  int n; scanf("%d",&n);  printf("%lld %lld\n",getsphi(n),getsmu(n));  }  return 0;  } |