目录

[数学 2](#_Toc30506)

[数论 2](#_Toc13155)

[埃氏筛 2](#_Toc19530)

[线性筛 2](#_Toc31446)

[逆元 3](#_Toc7583)

[质因数分解 3](#_Toc7019)

[组合数 3](#_Toc23362)

[欧拉函数 4](#_Toc27900)

[高中数学 5](#_Toc3387)

[图论 5](#_Toc4259)

[最小生成树 5](#_Toc27821)

[最短路径 9](#_Toc11708)

[网络流 14](#_Toc4923)

[数据结构 14](#_Toc12332)

[线段树 14](#_Toc23230)

[树状数组 16](#_Toc22988)

[字典树（Trie） 16](#_Toc6021)

[划分树 17](#_Toc18463)

[伸展树（SPlay） 17](#_Toc27060)

[主席树 17](#_Toc22715)

[字符串 17](#_Toc1664)

[相关函数 17](#_Toc27157)

[KMP 18](#_Toc8276)

[Manacher算法（需改） 19](#_Toc9102)

[AC自动机 20](#_Toc3369)

[后缀数组 22](#_Toc1773)

[后缀自动机 22](#_Toc1175)

[计算几何 22](#_Toc24236)

[二维几何 22](#_Toc23396)

[点的结构体 22](#_Toc30665)

[其他方法 23](#_Toc4256)

[杂七杂八 25](#_Toc19191)

[STL 25](#_Toc24719)

[bitset 25](#_Toc6234)

[string 25](#_Toc29680)

[vector 25](#_Toc21052)

[set 25](#_Toc13917)

[map 25](#_Toc22110)

[list 25](#_Toc29917)

[位运算 25](#_Toc13005)

# 数学

## 数论

素数有关

1. 算数基本定理：任何一个大于1的整数都可以分解成为有限个素数的乘积
2. 素数分布定理：对于正实数x，定于π（x）为不大于x的素数的个数，
3. 设函数P(x)为第n个素数的值，

### 埃氏筛

时间复杂度

|  |
| --- |
| bool v[MAXN]; //标记是否为素数  int prime[MAXN / 3], p = 0; //prime用来保存素数，p记录素数的个数  void is\_prime2()  {  memset(v,0,sizeof(v));  for (int i = 2; i <= n; i++)  {  if (!v[i]) p++;  for (int j = i \* i; j <= n; j += i)  v[j] = true;  }  } |

### 线性筛

|  |
| --- |
| bool v[MAXN];  int prime[MAXN / 3], p = 0;  void is\_prime3()  {  memset(v,0,sizeof(v));  for (int i=2;i<=n;i++)  {  if(!v[i])  {  v[i]=i;  prime[++p]=i;  }  for (int j=1;j<=p;j++)  {  if (prime[j]>v[i]||prime[j]>n/i) break;  v[i\*prime[j]]=prime[j];  }  }  } |

### 逆元

#### 费马小定理

|  |
| --- |
| ll inv(ll a,ll mod)  {  return pow\_m(a,mod-2);  } |

#### 求一组数的逆元

这个可以参考求组合数的那个板子

|  |
| --- |
| inv[1] = 1;  for(int i = 2; i < N; i ++){  inv[i] = (MOD - MOD / i) \* 1ll \* inv[MOD % i] % MOD;  } |

### 质因数分解

### 组合数

2. n个数，可重复选择，从中选出k个元素，有多少种方法：
3. O(n)时间复杂度求组合数 要求模是质数

|  |
| --- |
| int F[N], Finv[N], inv[N]; //F是阶乘，Finv是逆元的阶乘  void init()  {  inv[1] = 1;  for(int i = 2; i < N; i ++){  inv[i] = (MOD - MOD / i) \* 1ll \* inv[MOD % i] % MOD;  }  F[0] = Finv[0] = 1;  for(int i = 1; i < N; i ++){  F[i] = F[i-1] \* 1ll \* i % MOD;  Finv[i] = Finv[i-1] \* 1ll \* inv[i] % MOD;  }  }  int com(int n, int m) //comb(n, m)就是C(n, m)  { if(m < 0 || m > n) return 0;  return F[n] \* 1ll \* Finv[n - m] % MOD \* Finv[m] % MOD;  } |
|  |

### 欧拉函数

计算一个数的欧拉函数值，时间复杂度

|  |
| --- |
| int euler\_phi(int n)  {  int m=(int)sqrt(n+0.5);  int ans=n;  for (int i=2;i<=m;i++)  if (n%i==1)  {  ans=ans/i\*(i-1);  while (n%i==0) n/=i;  }  if (n>1) ans=ans/n\*(n-1);  return ans;  } |

欧拉定理：若a,n互质，则有

推论：若a,n互质，对于任意正整数b，有

筛法求欧拉函数

|  |
| --- |
| int euler[3000001];  void getEuler()  {  memset(euler, 0, sizeof(euler));  euler[1] = 1;  for (int i = 2; i <= 3000000; i++)  if (!euler[i])  for (int j = i; j <= 3000000; j += i) {  if (!euler[j])  euler[j] = j;  euler[j] = euler[j] / i \* (i - 1);  }  } |

## 高中数学

1. *海伦公式： ， p*为三角形半周长

# 图论

## 最小生成树

|  |
| --- |
| //prim  #include <iostream>  #include <stdio.h>  #include <vector>  #include <queue>  #include <string.h>  #include <algorithm>  using namespace std;  typedef long long LL;  const int maxn = 1e5 + 10;  struct edge /\*构建小顶堆\*/  {  int to;  int dis;  bool operator < (const edge& t) const {  return dis>t.dis;  }  };  priority\_queue<edge> q; /\*priority\_queue q \*/  vector<edge> G[maxn]; /\*graph\*/  bool vis[maxn]; /\*vis标记数组是否访问\*/  int N, M, a, b, w; /\*vertex edge\*/  void prim()  {  /\*维护一个与1号节点相连的边的集合，然后每次在其中找出最小的边\*/  for (int i = 0; i<G[1].size(); ++i) {  q.push(G[1][i]);  }  vis[1] = true; /\*第一个点标记访问\*/  int ret = 0; /\*统计权值\*/  int cnt = N - 1; /\*所有点是否访问\*/  while (!q.empty() && cnt)  {  edge p = q.top(); q.pop();  if (vis[p.to]) continue;  ret += p.dis;  cnt--;  vis[p.to] = true;  for (int i = 0; i<G[p.to].size(); ++i) {  edge pp = G[p.to][i];  if (!vis[pp.to]) q.push(pp);  }  }  for (int i = 1; i <= N; ++i)  {  if (vis[i] == false)  {  cout << "?" << endl;  return;  }  }  cout << ret << endl;  }  int main()  {  // freopen("in.txt", "r", stdin);  while (cin >> M >> N && M)  {  while (!q.empty()) q.pop();  for (int i = 0; i<maxn; ++i) G[i].clear();  memset(vis, false, sizeof(vis));  for (int i = 0; i<M; ++i) {  cin >> a >> b >> w;  G[a].push\_back(edge{ b,w });  G[b].push\_back(edge{ a,w });  }  prim();  }  return 0;  }  //krusal  #include <stdio.h>  #include <algorithm>  using namespace std;  struct EDGE  {  int u,v,cost;  }eg[100001];  int n,m,father[100001];    bool cmp(EDGE e1,EDGE e2)  {  return e1.cost<e2.cost;  }    // 并查集 初始化函数  void Init( int m )  {  int i;  for(i=1;i<=m;i++)  father[i]=i;  }  // 并查集 查找函数  int Find(int x)  {  while(father[x]!=x)  x=father[x];  return x;  }  // 并查集 合并函数  void Combine(int a,int b)  {  int temp\_a,temp\_b;  temp\_a=Find(a);  temp\_b=Find(b);    if(temp\_a!=temp\_b)  father[temp\_a]=temp\_b;  }    // 最小生成树 Kruskal 算法  int Kruskal( void )  {  EDGE e;  int i,res;  sort(eg,eg+n,cmp);  // 并查集 初始化  Init(m);    // 构建最小生成树  res=0;  for( i=0;i<n;++i )  {  e=eg[i];  if( Find(e.u)!=Find(e.v) )  {  Combine(e.u,e.v);  res+=e.cost;  }  }  return res;  }    int main()  {  int i,ans;  bool bl;  while( scanf("%d%d",&n,&m) && n )  {  for( i=0;i<n;++i )  scanf("%d%d%d",&eg[i].u,&eg[i].v,&eg[i].cost);  ans=Kruskal();    // 是否所有的点都在同一个集合  bl=true;  for(i=2;i<=m;++i)  if( Find(1)!=Find(i) )  {  bl=false;  break;  }    if( bl ) printf("%d\n",ans);  else printf("?\n");  }  return 0;  } |

## 最短路径

|  |
| --- |
| /\*Dijkstral算法及堆优化  /\*  伪码  清除所有标记  将d[1]置为0，其他置为1  循环n次  在为标记的结点中选出一个d值最小的  标记这个结点  对于从x出发的所有边（另一端是没有被标记的结点），进行relax操作  \*/  #include <cstring>  #include <algorithm>  using namespace std;  //邻接矩阵存图  const int INF=0x3f3f3f3f;  const int maxn=1200;  int dist[maxn],g[maxn][maxn],fa[maxn],N,M;  bool vis[maxn];  void dijkstra()  {  //初始化工作  memset(vis,0,sizeof(vis));  for(int i=1;i<=N;i++)  dist[i]=(i==1)?0:INF;  //外层循环n次  for(int i=1;i<=N;i++)  {  //在未标记的结点中选出一个d值最小的  int mark=-1,mindis=INF;  for(int j=1;j<=N;j++)  {  if(!vis[j]&&dist[j]<mindis)  {  mindis=dist[j];  mark=j;  }  }  //标记这个结点  vis[mark]=1;  //对于从x出发的所有边（另一端是没有被标记的结点），进行relax操作  for(int j=1;j<=N;j++)  {  if(!vis[j])  {  dist[j]=min(dist[j],dist[mark]+g[mark][j]); //计算最短路  fa[j]=i; //保存路径  }  }  }  }  /\*  使用优先队列Dijkstra算法  复杂度O(ElogE)  注意对vector<Edge> E[MAXN]进行初始化后加边  \*/    #include<iostream>  #include<cstdio>  #include<vector>  #include<queue>  using namespace std;    const int INF=0x3f3f3f3f;  const int MAXN=1000010;    struct qnode  {  int v;  int c;  qnode(int \_v=0,int \_c=0):v(\_v),c(\_c){}  bool operator <(const qnode &r)const  {  return c>r.c;  }  };    struct Edge  {  int v,cost;  Edge(int \_v=0,int \_cost=0):v(\_v),cost(\_cost){}  };    vector<Edge> E[MAXN];  bool vis[MAXN];  int dist[MAXN];    void Dijkstra(int n,int start)//点的编号从1开始  {  memset(vis,false,sizeof(vis));  for(int i=1;i<=n;i++) dist[i]=INF;  priority\_queue<qnode> que;  while(!que.empty()) que.pop();  dist[start]=0;  que.push(qnode(start,0));  qnode tmp;  while(!que.empty())  {  tmp=que.top();  que.pop();  int u=tmp.v;  if(vis[u]) continue;  vis[u]=true;  for(int i=0;i<E[u].size();i++)  {  int v=E[tmp.v][i].v;  int cost=E[u][i].cost;  if(!vis[v]&&dist[v]>dist[u]+cost)  {  dist[v]=dist[u]+cost;  que.push(qnode(v,dist[v]));  }  }  }  }    void addedge(int u,int v,int w)  {  E[u].push\_back(Edge(v,w));  }    /\*  Bellman-Ford算法及SPFA优化  伪码  for i=1 to v-1  for each e(u,v)  relax(u,v,w)  for each e(u,v)  if v.d>u.d+w(u,v)  return false  return true;  \*/  struct edg  {  int u,v,cost;  }edge[10009];  bool bellman\_ford()  {  for (int i = 1; i < N; ++i)//最多进行n-1次更新  {  for (int j = 1; j <= 2\*M; ++j)//每次遍历一下边表，至少找出一个点的最短路径  {  if (dist[edge[j].v]>dist[edge[j].u] + edge[j].cost)  {  dist[edge[j].v] = dist[edge[j].u] + edge[j].cost;  }  }  }  bool flag = true;  for (int i = 0; i < M; ++i)//判断负环  {  if (dist[edge[i].v]>dist[edge[i].u] + edge[i].cost)  {  flag = false;  break;  }  }  return flag;  }  /\*  Floyd算法  \*/  const int INF = 100000000;  int dp[10][10];  void floyd()  {  int u, c, v, V;  while (scanf("%d", &V) != EOF && V)  {  //初始化工作  for (int i = 0; i < 10; i++)  for (int j = 0; j < 10; j++)  {  dp[i][j] = INF;  if (i == j) dp[i][i] = 0;  }  //输入  for (int i = 0; i < V; i++)  {  cin >> u >> v >> c;  dp[v][u] = dp[u][v] = c;  }  //核心代码只有4行  for (int k = 0; k <= V; k++)  for (int i = 0; i <= V; i++)  for (int j = 0; j <= V; j++)  dp[i][j] = min(dp[i][j], dp[i][k] + dp[k][j]);  } |

## 网络流

## LCA

### 倍增法

# 数据结构

## 线段树

|  |
| --- |
| #include<bits/stdc++.h>  using namespace std;  const int maxn=100010;  int sum[maxn<<2],lazy[maxn<<2];//节点值和懒惰标记  int p[maxn];//原数组  int mn[maxn<<2];//最值  void pushup(int rt)  {  sum[rt] = sum[rt<<1]+sum[rt<<1|1];//求和  mn[rt] = min(mn[rt<<1],mn[rt<<1|1]);//最值  }  void pushdown(int rt,int ln,int rn) {//ln为左区间长度，rn为右区间长度  if (lazy[rt]) {  lazy[rt << 1] += lazy[rt];  lazy[rt << 1 | 1] += lazy[rt];  sum[rt << 1] += lazy[rt] \* ln;  sum[rt << 1 | 1] += lazy[rt] \* rn;  lazy[rt] = 0;  }  }  void build(int l,int r,int rt)  {  if(l==r)  {  sum[rt] = p[l];  mn[rt] = p[l];  return ;  }  int mid = (l+r)>>1;  build(l,mid,rt<<1);  build(mid+1,r,rt<<1|1);  pushup(rt);  }  void updata1(int R, int C, int l, int r, int rt) {//单点更新  if (l == r) {  sum[rt] += C;  mn[rt] = C;  return;  }  int mid = (l + r) >> 1;  pushdown(rt, mid - l + 1, r - mid);  if (R <= mid)updata1(R, C, l, mid, rt << 1);  else updata1(R, C, mid + 1, r, rt << 1 | 1);  pushup(rt);  }  void updata2(int L,int R,int C,int l,int r,int rt)//区间更新（L,R）  {  if(L==l&&r==R)  {  sum[rt] += (r - l + 1)\*C;  lazy[rt] += C;  return;  }  int mid = (l+r)>>1;  pushdown(rt, mid - l + 1, r - mid);  if(R<=mid)updata2(L,R,C,l,mid,rt<<1);  else if(mid<L)updata2(L,R,C,mid+1,r,rt<<1|1);  else{  updata2(L,mid,C,l,mid,rt<<1);  updata2(mid+1,R,C,mid+1,r,rt<<1|1);  }  pushup(rt);  }  int query(int L,int R,int l,int r,int rt)//区间查询（L,R）  {  if(l==L&&R==r)  {  return sum[rt];  }  int mid = (l+r)>>1;  pushdown(rt, mid - l + 1, r - mid);  int ret = 0;  if(R<=mid)ret = query(L,R,l,mid,rt<<1);  else if(mid<L)ret = query(L,R,mid+1,r,rt<<1|1);  else ret = query(L,mid,l,mid,rt<<1)+query(mid+1,R,mid+1,r,rt<<1|1);  pushup(rt);  return ret;  }  void init()//初始化  {  memset(lazy,0,sizeof(lazy));  } |

## 树状数组

## 字典树（Trie）

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  //对于字符串比较多的要统计个数的，map被卡的情况下，直接用字典树  //很多题都是要用到节点下标来表示某个字符串  const int MAXN = 2e6 + 5;//如果是64MB可以开到2e6+5，尽量开大  int tree[MAXN][30];//tree[i][j]表示节点i的第j个儿子的节点编号  int sum[MAXN];//sum[i]表示结点i有多少个单词  bool flagg[MAXN];//表示以该节点结尾是一个单词  int tot;//总节点数  void insert\_(char \*str)  {  int len = strlen(str);  int root = 0;  for (int i = 0; i<len; i++)  {  int id = str[i] - 'a';  if (!tree[root][id])  {  tree[root][id] = ++tot;  }  sum[tree[root][id]]++;  root = tree[root][id];  }  flagg[root] = true;  }  int find\_(char \*str)//查询操作，按具体要求改动  {  int len = strlen(str);  int root = 0;  for (int i = 0; i<len; i++)  {  int id = str[i] - 'a';  if (!tree[root][id]) return 0;  root = tree[root][id];  }  return sum[root];  }  void init()//最后清空，节省时间  {  for (int i = 0; i <= tot; i++)  {  flagg[i] = false;  for (int j = 0; j<10; j++)  tree[i][j] = 0;  }  tot = 0;//RE有可能是这里的问题  } |

## 划分树

## 伸展树（SPlay）

## 主席树

# 字符串

## 相关函数

cin.getline（指定字符串，输入字符串的个数）

getline（cin，string类型的字符串）

gets\_s（指定字符串）可以读入空格

## KMP

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  const int MAXN = 1E3;  void kmp\_pre(char x[], int m, int next[])  {  int i, j;  j = next[0] = -1;  i = 0;  while (i<m)  {  while (-1 != j && x[i] != x[j])  j = next[j];  next[++i] = ++j;  }  }  /\*  \* 返回x 在y 中出现的次数，可以重叠  \*/  int nextt[MAXN + 5];  int KMP\_Count(char x[], int m, char y[], int n) {//x 是模式串，y 是主串  int i, j;  int ans = 0;  kmp\_pre(x, m, nextt);  i = j = 0;  while (i<n)  {  while (-1 != j && y[i] != x[j])  j = nextt[j];  i++;  j++;  if (j >= m)  {  ans++;  j = 0;//不计算重复  //计算重复是这个 j=next[j]  }  }  return ans;  } |

## Manacher算法（需改）

时间复杂度

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  const int MAXN = 110010;  char Ma[MAXN \* 2];  int Mp[MAXN \* 2]; //Mp[i]表示以i为中心最大的回文长度  void Manacher(char s[], int len) {  int l = 0;  Ma[l++] = '$';  Ma[l++] = '#';  for (int i = 0; i<len; i++) {  Ma[l++] = s[i];  Ma[l++] = '#';  }  Ma[l] = 0;  //mx表示当前已知回文的最大右边界  //id表示当前已知回文的最大中中心点  int mx = 0, id = 0;  for (int i = 0; i<l; i++)  {  //当前字符在已判断的最大回文里面  if (mx > i) Mp[i] = min(Mp[2 \* id - 1], mx - i);  //不在则从1开始  else Mp[i] = 1;  //剩下不能确定的暴力判断  while (Ma[i + Mp[i]] == Ma[i - Mp[i]])  Mp[i]++;  //更新已判断的最大回文  if (i + Mp[i]>mx) {  mx = i + Mp[i];  id = i;  }  }  }  char s[MAXN];  int main() {  while (scanf("%s", s) == 1) {  int len = strlen(s);  Manacher(s, len);  int ans = 0;  for (int i = 0; i<2 \* len + 2; i++)  ans = max(ans, Mp[i] - 1);  printf("%d\n", ans);  }  return 0;  } |
|  |

## AC自动机

|  |
| --- |
| #include <bits/stdc++.h>  using namespace std;  const int N = 10100, L = 1000100;  char s[L];  int num, n;  struct node {  int son[30];  int fail, cnt;  }a[N \* 60];  queue<int> q;  void clear(int x)  {  a[x].cnt = 0;  a[x].fail = 0;  memset(a[x].son, 0, sizeof(a[x].son));  }  void insert(char \*c)  {  int l = strlen(c);  int x = 0;  for (int i = 0; i<l; i++)  {  int t = c[i] - 'a' + 1;  if (!a[x].son[t])  {  num++;  clear(num);  a[x].son[t] = num;  }  x = a[x].son[t];  }  a[x].cnt++;  }  void buildAC()  {  while (!q.empty()) q.pop();  for (int i = 1; i <= 26; i++)  if (a[0].son[i]) q.push(a[0].son[i]);  while (!q.empty())  {  int x = q.front(); q.pop();  int fail = a[x].fail;  for (int i = 1; i <= 26; i++)  {  int y = a[x].son[i];  if (y)  {  a[y].fail = a[fail].son[i];  q.push(y);  }  else a[x].son[i] = a[fail].son[i];  }  }  }  int find(char \*c)  {  int l = strlen(c);  int x = 0, ans = 0;  for (int i = 0; i<l; i++)  {  int t = c[i] - 'a' + 1;  while (x && !a[x].son[t]) x = a[x].fail;  x = a[x].son[t];  int p = x;  while (p && a[p].cnt != -1)  {  ans += a[p].cnt;  a[p].cnt = -1;  p = a[p].fail;  }  }  return ans;  }  int main()  {  int T;  scanf("%d", &T);  while (T--)  {  scanf("%d", &n);  num = 0;  clear(0);  for (int i = 1; i <= n; i++)  {  scanf("%s", s);  insert(s);  }  buildAC();  scanf("%s", s);  printf("%d\n", find(s));  }  return 0;  } |

## 后缀数组

## 后缀自动机

# 计算几何

## 二维几何

### 点的结构体

|  |
| --- |
| struct Point  {  double x,y;  Point(){}  Point(double \_x,double \_y):x(\_x),y(\_y){}  //点积,a\*b = len(a)\*len(b)\*cos(<a,b>);  double operator\*(const Point &b)  {  return x\*b.x+y\*b.y;  }  Point operator\*(const double a)  {  return Point(a\*x,a\*y);  }  //叉积,a^b = len(a)\*len(b)\*sin(<a,b>);  double operator^(const Point &b)  {  return x\*b.y-y\*b.x;  }  //点做减法是向量  Point operator-(const Point &b)  {  return Point(x-b.x,y-b.y);  }  Point operator+(const Point &b)  {  return Point(x+b.x,y+b.y);  }  //返回长度  double len()  {  return hypot(x,y);  }  //返回长度的平方  double len2()  {  return x\*x+y\*y;  }  }; |

### 其他方法

1. 一些常量

|  |
| --- |
| const double esp = 1e-8;  const double inf = 1e20; |

1. 判断是否为0

|  |
| --- |
| int sgn(double x)  {  if (fabs(x)<esp) return 0;  if (x<0) return -1;  return 1;  } |

1. 将向量逆时针旋转rad（rad为弧度）

|  |
| --- |
| Vector Rotate(Vector A,double rad)  {  return Vector(A.x\*cos(rad)-A.y\*sin(rad),A.x\*sin(rad)+A.y\*cos(rad));  } |

1. 计算两个向量之间的夹角（尽量不要用余弦定理）

|  |
| --- |
| double Angle(Vector A,Vector B)  {  return acos((A\*B)/(A.len()\*B.len()));  } |

1. 判断点p是否在线段a1,a2上

|  |
| --- |
| bool OnSegment(Point p,Point a1,Point a2)  {  //后面的点积可否等于0？  return sgn((a1-p)^(a2-p))==0&&sgn((a1-p)\*(a2-p))<0;  } |

1. 判断两直线相交

|  |
| --- |
| bool SegmentCross(Point a1,Point a2,Point b1,Point b2)  {  double c1 = sgn((a2-a1)^(b1-a1));  double c2 = sgn((a2-a1)^(b2-a1));  double c3 = sgn((b2-b1)^(a1-b1));  double c4 = sgn((b2-b1)^(a2-b1));  //正规相交  if (c1\*c2<0&&c3\*c4<0) return 1;  return (c1==0&&sgn((a1-b1)\*(a2-b1))<=0)||  (c2==0&&sgn((a1-b2)\*(a2-b2))<=0)||  (c3==0&&sgn((b1-a1)\*(b2-a1))<=0)||  (c4==0&&sgn((b1-a2)\*(b2-a2))<=0);  } |

1. 计算两直线的交点（两点式）

|  |
| --- |
| Point Intersection(Point p1,Point p2,Point q1,Point q2)  {  return p1 + (p2-p1)\*(((q2-q1)^(q1-p1))/((q2-q1)^(p2-p1)));  } |

1. 计算两直线的交点（点斜式）

|  |
| --- |
| Point GetLineIntersection(Point P,Vector v,Point Q,Vector w)  {  Vector u = P-Q;  double t = (w^u)/(v^w);  return P+v\*t;  } |

# STL

## 容器

Bitset

|  |
| --- |
| public void set(int pos): 位置pos的字位设置为true。  public void set(int bitIndex, boolean value) 将指定索引处的位设置为指定的值。  public void clear(int pos): 位置pos的字位设置为false。  public void clear() : 将此 BitSet 中的所有位设置为 false。  public int cardinality() 返回此 BitSet 中设置为 true 的位数。  public boolean get(int pos): 返回位置是pos的字位值。  public void and(BitSet other): other与操作，结果作为该字位集的新值。  public void or(BitSet other): other或操作，结果作为该字位集的新值。  public void xor(BitSet other): other异或操作，结果作为该字位集的新值。  public void andNot(BitSet set) 清除此BitSet中所有的位,set用来屏蔽此BitSet的BitSet  public int size(): 返回此 BitSet 表示位值时实际使用空间的位数。  public int length() 返回此 BitSet 的逻辑大小，BitSet 中最高设置位的索引加 1。  public int hashCode(): 返回该集合Hash 码， 这个码同集合中的字位值有关。  public boolean equals(Object other): 返回other中的字位与集合中的位是否相同  public Object clone() 克隆此 BitSet，生成一个与之相等的新 BitSet。  public String toString() 返回此位 set 的字符串表示形式。 |

string

|  |
| --- |
| //删除erase  string& erase(int pos = 0, int n = npos );删除从pos开始的第n个字符  iterator erase(iterator position );删除position指定的字符  iterator erase(iterator first, iterator last );删除first到last之间的字符  //查找find，倒着查找rfind  //截取子串substr  string substr(int pos); //从下标为pos一直到字符串结尾  string substr(int pos, int n);//从下表为pos开始的后n个字符 |

vector

set

map

List

## 黑科技

unique()

去重函数。用法跟sort()一样，将不重复的元素移到数组的前面来，对于后面的元素不改变，使用前要对数组sort排序。返回值是一个迭代器，指向不重复数组最后一个元素的下一个位置。

iterator unique(iterator it\_1,iterator it\_2,bool MyFunc);第三个参数用于判断两个元素是否相等

reverse() reverse(a,a+n) 翻转数组的元素

### 位运算

lowbit（n）：非负整数n在二进制表示下“最低位的1及其后面所有的0”。Lowbit（n）=n&(~n+1)=n&(-n)