

## 基础算法

归并求逆序对

整数二分

高精度

离散化

## 数据结构

单调队列

KMP

Trie树

并查集

字符串哈希

## 搜索+图论

邻接表

康托展开

拓扑排序

朴素dijkstra

堆优化版dijkstra

spfa 算法（队列优化的Bellman-Ford算法）

spfa判断图中是否存在负环

floyd算法

朴素版prim算法

Kruskal算法

染色法判别二分图

匈牙利算法

网络流

最小费用流

## 数学知识

试除法分解质因数

线性筛法求素数

试除法求所有约数

求欧拉函数

筛法求欧拉函数

扩展欧几里得算法

高斯消元

递归法求组合数

通过预处理逆元的方式求组合数

Lucas定理

分解质因数法求组合数

卡特兰数

动态规划

背包问题

线性DP

区间DP

博弈论

NIM游戏

高级数据结构

树状数组

线段树

AC自动机

## 基础算法

归并求逆序对

```
1  /*
2  快速排序接口：
3  数组头， 左端点， 右端点
4  调用(q, 0, n - 1);
5  */
6  void quick_sort(int q[], int l, int r)
7  {
8      if (l >= r)
9          return;
10     int x = q[l + r >> 1], i = l - 1, j = r + 1;
11     while (i < j)
12     {
13         do i++; while (q[i] < x);
14         do j--; while (q[j] > x);
15         if (i < j)
16             swap(q[i], q[j]);
17     }
18     quick_sort(q, l, j);
19     quick_sort(q, j + 1, r);
20 }
21 /*
22 归并排序接口：
```

```

23 数组头, 左端点, 右端点
24 调用(q, 0, n - 1);
25 */
26 int temp[maxn];
27 int con = 0; //逆序对Count Inversions
28 void merge_sort(int q[], int l, int r)
29 {
30     if (l >= r)
31         return;
32     int mid = l + r >> 1;
33     //递归
34     merge_sort(q, l, mid);
35     merge_sort(q, mid + 1, r);
36
37     // merge(q + l, q + mid + 1, q + mid + 1, q + r + 1, temp);
38     //合并
39     int k = 0, i = l, j = mid + 1;
40     while (i <= mid && j <= r)
41         if (q[i] <= q[j])
42             temp[k++] = q[i++];
43         else //逆序对Count Inversions
44         {
45             con += mid - i + 1;
46             temp[k++] = q[j++];
47         }
48
49     while (i <= mid)
50     {
51         temp[k++] = q[i++];
52     }
53
54     while (j <= r)
55     {
56         temp[k++] = q[j++];
57     }
58
59
60     //改变原数组
61     for (int i = l, j = 0; i <= r; i++, j++)
62         q[i] = temp[j];
63 }
64
65
66 int a[1000010];
67 int main()
68 {
69     scanf("%d", &n);
70     for (int i = 0; i < n; i++)
71     {
72         scanf("%d", &a[i]);
73     }
74     merge_sort(a, 0, n - 1);
75     cout << con << endl;
76     // sort(a, a + n);
77     // quick_sort(a, 0, n - 1);
78     // for (int i = 0; i < n; i++)
79     // {
80     //     printf("%d ", a[i]);

```

```

81 // }
82 // freopen("F:/Overflow/in.txt","r",stdin);
83 // ios::sync_with_stdio(false);
84 return 0;
85 }

```

## 整数二分

```

1 bool check(int x) { /* ... */ } // 检查x是否满足某种性质
2
3 // 区间[l, r]被划分成[l, mid]和[mid + 1, r]时使用:
4 int bsearch_1(int l, int r)
5 {
6     while (l < r)
7     {
8         int mid = l + r >> 1;
9         if (check(mid)) r = mid;    // check()判断mid是否满足性质
10        else l = mid + 1;
11    }
12    return l;
13 }
14 // 区间[l, r]被划分成[l, mid - 1]和[mid, r]时使用:
15 int bsearch_2(int l, int r)
16 {
17     while (l < r)
18     {
19         int mid = l + r + 1 >> 1;
20         if (check(mid)) l = mid;
21         else r = mid - 1;
22    }
23    return l;
24 }

```

## 高精度

```

1 // 输入输出都要倒序!!!!!!
2 bool cmp(vector<int> &A, vector<int> &B) // A > B 返回1
3 {
4     if (A.size() != B.size())
5         return A.size() > B.size();
6     for (int i = A.size() - 1; i >= 0; i--)
7     {
8         if (A[i] != B[i])
9             return A[i] > B[i];
10    }
11    return 1;
12 }
13
14 vector<int> add(vector<int> &A, vector<int> &B)
15 {
16     vector<int> C;

```

```

17
18     int t = 0;
19     for (int i = 0; i < A.size() || i < B.size(); i++)
20     {
21         if (i < A.size()) t += A[i];
22         if (i < B.size()) t += B[i];
23         C.push_back(t % 10);
24         t /= 10;
25     }
26     if (t) C.push_back(1);
27     return C;
28 }
29
30 vector<int> sub(vector<int> &A, vector<int> &B)
31 {
32     vector<int> C;
33     for (int i = 0, t = 0; i < A.size(); i++)
34     {
35         t = A[i] - t;
36         if (i < B.size())
37             t -= B[i];
38         C.push_back((t + 10) % 10);
39         if (t < 0)
40             t = 1;
41         else
42             t = 0;
43     }
44
45     //删除前导零
46     while (C.size() > 1 && C.back() == 0)
47         C.pop_back();
48     return C;
49 }
50
51 vector<int> mul(vector<int> &A, int B)
52 {
53     vector<int> C;
54     int t = 0;
55     for (int i = 0; i < A.size() || t; i++)
56     {
57         if (i < A.size()) t += A[i] * B;
58         C.push_back(t % 10);
59         t /= 10;
60     }
61     return C;
62 }
63 //A / B ,余数r
64 vector<int> div(vector<int> &A, int b, int &r)
65 {
66     vector<int> C;
67     r = 0;
68     for (int i = A.size() - 1; i >= 0; i--)
69     {
70         r = r * 10 + A[i];
71         C.push_back(r / b);
72         r %= b;
73     }
74

```

```

75     reverse(C.begin(), C.end());
76     while (C.size() > 1 && C.back() == 0)
77         C.pop_back();
78     return C;
79 }
80
81 int main()
82 {
83     string a;
84     int B;
85     vector<int> A;
86     cin >> a >> B;
87     for (int i = a.size() - 1; i >= 0; i--) A.push_back(a[i] - '0');
88     vector<int> C;
89     int r = 0;
90     C = div(A, B, r);
91     for (int i = C.size() - 1; i >= 0; i--) printf("%d", C[i]);
92     cout << endl << r << endl;
93     return 0;
94 }

```

## 离散化

```

1  vector<int> alls;
2  vector<PII> add, query;
3
4  int a[maxn];
5  int s[maxn];
6
7  //二分
8  int find(int x)
9  {
10     int l = 0, r = alls.size() - 1;
11     while (l < r)
12     {
13         int mid = l + r >> 1;
14         if (alls[mid] >= x) r = mid;
15         else l = mid + 1;
16     }
17     return r + 1;
18 }
19
20
21 void solve()
22 {
23     cin >> n >> m;
24     for (int i = 0; i < n; i++)
25     {
26         int x, c;
27         cin >> x >> c;
28         add.push_back({x, c});
29         alls.push_back(x);
30     }
31     for (int i = 0; i < m; i++)
32     {
33         int l, r;

```

```

34     cin >> l >> r;
35     query.push_back({l, r});
36
37     alls.push_back(l);
38     alls.push_back(r);
39 }
40 //去重
41 sort(alls.begin(), alls.end());
42 alls.erase(unique(alls.begin(), alls.end()), alls.end());
43
44 //
45 for (auto item : add)
46 {
47     int temp = find(item.first);
48     a[temp] += item.second;
49 }
50
51 for (int i = 1; i <= alls.size(); i++) s[i] = s[i - 1] + a[i];
52 for (auto item : query)
53 {
54     int l = find(item.first);
55     int r = find(item.second);
56     cout << s[r] - s[l - 1] << endl;
57 }
58 }

```

## 数据结构

### 单调队列

```

1     int hh = 0, tt = -1;
2     for (int i = 0; i < n; i++)
3     {
4         if (hh <= tt && i - k + 1 > q[hh]) hh++;
5
6         while (hh <= tt && a[q[tt]] >= a[i]) tt--;
7         q[++tt] = i;
8
9         if (i >= k - 1) printf("%d ", a[q[hh]]);
10    }

```

### KMP

```

1 // s[]是长文本, p[]是模式串, n是s的长度, m是p的长度
2 求模式串的Next数组:
3 for (int i = 2, j = 0; i <= m; i++)
4 {
5     while (j && p[i] != p[j + 1]) j = ne[j];
6     if (p[i] == p[j + 1]) j++;

```

```

7     ne[i] = j;
8 }
9
10 // 匹配
11 for (int i = 1, j = 0; i <= n; i ++ )
12 {
13     while (j && s[i] != p[j + 1]) j = ne[j];
14     if (s[i] == p[j + 1]) j ++ ;
15     if (j == m)
16     {
17         j = ne[j];
18         // 匹配成功后的逻辑
19     }
20 }

```

## Trie树

```

1  int son[N][26], cnt[N], idx;
2  // 0号点既是根节点，又是空节点
3  // son[][]存储树中每个节点的子节点
4  // cnt[]存储以每个节点结尾的单词数量
5
6  // 插入一个字符串
7  void insert(char *str)
8  {
9      int p = 0;
10     for (int i = 0; str[i]; i ++ )
11     {
12         int u = str[i] - 'a';
13         if (!son[p][u]) son[p][u] = ++ idx;
14         p = son[p][u];
15     }
16     cnt[p] ++ ;
17 }
18
19 // 查询字符串出现的次数
20 int query(char *str)
21 {
22     int p = 0;
23     for (int i = 0; str[i]; i ++ )
24     {
25         int u = str[i] - 'a';
26         if (!son[p][u]) return 0;
27         p = son[p][u];
28     }
29     return cnt[p];
30 }

```

## 并查集

```

1  (1)朴素并查集：
2
3      int p[N]; //存储每个点的祖宗节点
4

```



```

5 // 返回x的祖宗节点
6 int find(int x)
7 {
8     if (p[x] != x) p[x] = find(p[x]);
9     return p[x];
10 }
11
12 // 初始化, 假定节点编号是1~n
13 for (int i = 1; i <= n; i++) p[i] = i;
14
15 // 合并a和b所在的两个集合:
16 p[find(a)] = find(b);
17
18
19 (2)维护size的并查集:
20
21 int p[N], size[N];
22 //p[]存储每个点的祖宗节点, size[]只有祖宗节点的有意义, 表示祖宗节点所在集合中的点的数量
23
24 // 返回x的祖宗节点
25 int find(int x)
26 {
27     if (p[x] != x) p[x] = find(p[x]);
28     return p[x];
29 }
30
31 // 初始化, 假定节点编号是1~n
32 for (int i = 1; i <= n; i++)
33 {
34     p[i] = i;
35     size[i] = 1;
36 }
37
38 // 合并a和b所在的两个集合:
39 size[find(b)] += size[find(a)];
40 p[find(a)] = find(b);
41
42
43 (3)维护到祖宗节点距离的并查集:
44
45 int p[N], d[N];
46 //p[]存储每个点的祖宗节点, d[x]存储x到p[x]的距离
47
48 // 返回x的祖宗节点
49 int find(int x)
50 {
51     if (p[x] != x)
52     {
53         int u = find(p[x]);
54         d[x] += d[p[x]];
55         p[x] = u;
56     }
57     return p[x];
58 }
59
60 // 初始化, 假定节点编号是1~n
61 for (int i = 1; i <= n; i++)
62 {

```

```

63     p[i] = i;
64     d[i] = 0;
65 }
66
67 // 合并a和b所在的两个集合:
68 p[find(a)] = find(b);
69 d[find(a)] = distance; // 根据具体问题, 初始化find(a)的偏移量

```

## 字符串哈希

```

1  核心思想: 将字符串看成P进制数, P的经验值是131或13331, 取这两个值的冲突概率低
2  小技巧: 取模的数用2^64, 这样直接用unsigned long long存储, 溢出的结果就是取模的结果
3
4  typedef unsigned long long ULL;
5  ULL h[N], p[N]; // h[k]存储字符串前k个字母的哈希值, p[k]存储 P^k mod 2^64
6
7  // 初始化
8  p[0] = 1;
9  for (int i = 1; i <= n; i++)
10 {
11     h[i] = h[i - 1] * P + str[i];
12     p[i] = p[i - 1] * P;
13 }
14
15 // 计算子串 str[l ~ r] 的哈希值
16 ULL get(int l, int r)
17 {
18     return h[r] - h[l - 1] * p[r - l + 1];
19 }

```

## 搜索+图论

### 邻接表

```

1  // 对于每个点k, 开一个单链表, 存储k所有可以走到的点. h[k]存储这个单链表的头结点
2  int h[N], e[N], ne[N], idx;
3
4  // 添加一条边a->b
5  void add(int a, int b)
6  {
7      e[idx] = b, ne[idx] = h[a], h[a] = idx++;
8  }
9
10 // 初始化
11 idx = 0;
12 memset(h, -1, sizeof h);

```

## 康托展开

```
1  /*****
2  * 简述
3  康托展开是一个全排列到一个自然数的双射，常用于构建hash表时的空间压缩。
4  设有n个数 (1, 2, 3, 4,...,n)，可以有组成不同(n!种)的排列组合，康托展开
5  表示的就是当前排列组合在n个不同元素的全排列中的名次。
6
7  * 原理
8   $X = a[n] * (n-1)! + a[n-1] * (n-2)! + \dots + a[i] * (i-1)! + \dots + a[1] * 0!$ 
9
10 https://www.cnblogs.com/sky-stars/p/11216035.html
11 *****/
12 int FAC[10] =
13 {
14     1,
15     1,
16     2,
17     6,
18     24,
19     120,
20     720,
21     5040,
22     40320,
23     362880,
24 }; // 阶乘打表
25
26 int cantor(int *a, int n)
27 {
28     int x = 0;
29     for (int i = 0; i < n; ++i)
30     {
31         int smaller = 0; // 在当前位之后小于其的个数
32         for (int j = i + 1; j < n; ++j)
33         {
34             if (a[j] < a[i])
35                 smaller++;
36         }
37         x += FAC[n - i - 1] * smaller; // 康托展开累加
38     }
39     return x; // 康托展开值
40 }
```

## 拓扑排序

$$O(n + m)$$

```

1  bool topsort()
2  {
3      int hh = 0, tt = -1;
4
5      // d[i] 存储点i的入度
6      for (int i = 1; i <= n; i ++ )
7          if (!d[i])
8              q[ ++ tt] = i;
9
10     while (hh <= tt)
11     {
12         int t = q[hh ++ ];
13
14         for (int i = h[t]; i != -1; i = ne[i])
15         {
16             int j = e[i];
17             if (-- d[j] == 0)
18                 q[ ++ tt] = j;
19         }
20     }
21
22     // 如果所有点都入队了，说明存在拓扑序列；否则不存在拓扑序列。
23     return tt == n - 1;
24 }

```

## 朴素dijkstra

时间复杂是 $O(n^2 + m)$   $n$ 表示点数， $m$ 表示边数

```

1  int g[N][N]; // 存储每条边
2  int dist[N]; // 存储1号点到每个点的最短距离
3  bool st[N]; // 存储每个点的最短路是否已经确定
4
5  // 求1号点到n号点的最短路，如果不存在则返回-1
6  int dijkstra()
7  {
8      memset(dist, 0x3f, sizeof dist);
9      dist[1] = 0;
10
11     for (int i = 0; i < n - 1; i ++ )
12     {
13         int t = -1; // 在还未确定最短路的点中，寻找距离最小的点
14         for (int j = 1; j <= n; j ++ )
15             if (!st[j] && (t == -1 || dist[t] > dist[j]))
16                 t = j;
17
18         // 用t更新其他点的距离
19         for (int j = 1; j <= n; j ++ )
20             dist[j] = min(dist[j], dist[t] + g[t][j]);
21
22         st[t] = true;
23     }
24
25     if (dist[n] == 0x3f3f3f3f) return -1;
26     return dist[n];

```

## 堆优化版dijkstra

时间复杂度 $O(m\log n)$ ,  $n$  表示点数,  $m$  表示边数

```

1  typedef pair<int, int> PII;
2
3  int n;          // 点的数量
4  int h[N], w[N], e[N], ne[N], idx;          // 邻接表存储所有边
5  int dist[N];    // 存储所有点到1号点的距离
6  bool st[N];     // 存储每个点的最短距离是否已确定
7
8  // 求1号点到n号点的最短距离, 如果不存在, 则返回-1
9  int dijkstra()
10 {
11     memset(dist, 0x3f, sizeof dist);
12     dist[1] = 0;
13     priority_queue<PII, vector<PII>, greater<PII>> heap;
14     heap.push({0, 1});          // first存储距离, second存储节点编号
15
16     while (heap.size())
17     {
18         auto t = heap.top();
19         heap.pop();
20
21         int ver = t.second, distance = t.first;
22
23         if (st[ver]) continue;
24         st[ver] = true;
25
26         for (int i = h[ver]; i != -1; i = ne[i])
27         {
28             int j = e[i];
29             if (dist[j] > distance + w[i])
30             {
31                 dist[j] = distance + w[i];
32                 heap.push({dist[j], j});
33             }
34         }
35     }
36
37     if (dist[n] == 0x3f3f3f3f) return -1;
38     return dist[n];
39 }

```

## spfa 算法（队列优化的Bellman-Ford算法）

时间复杂度 平均情况下  $O(m)$ , 最坏情况下  $O(nm)$

```

1  int n;          // 总点数
2  int h[N], w[N], e[N], ne[N], idx;      // 邻接表存储所有边
3  int dist[N];    // 存储每个点到1号点的最短距离
4  bool st[N];     // 存储每个点是否在队列中
5
6  // 求1号点到n号点的最短路距离，如果从1号点无法走到n号点则返回-1
7  int spfa()
8  {
9      memset(dist, 0x3f, sizeof dist);
10     dist[1] = 0;
11
12     queue<int> q;
13     q.push(1);
14     st[1] = true;
15
16     while (q.size())
17     {
18         auto t = q.front();
19         q.pop();
20
21         st[t] = false;
22
23         for (int i = h[t]; i != -1; i = ne[i])
24         {
25             int j = e[i];
26             if (dist[j] > dist[t] + w[i])
27             {
28                 dist[j] = dist[t] + w[i];
29                 if (!st[j]) // 如果队列中已存在j，则不需要将j重复插入
30                 {
31                     q.push(j);
32                     st[j] = true;
33                 }
34             }
35         }
36     }
37
38     if (dist[n] == 0x3f3f3f3f) return -1;
39     return dist[n];
40 }

```

## spfa判断图中是否存在负环

```

1  int n;          // 总点数
2  int h[N], w[N], e[N], ne[N], idx;      // 邻接表存储所有边
3  int dist[N], cnt[N];                    // dist[x]存储1号点到x的最短距离，cnt[x]存储1到x的最短路中经过的点数
4  bool st[N];     // 存储每个点是否在队列中
5
6  // 如果存在负环，则返回true，否则返回false。
7  bool spfa()
8  {
9      // 不需要初始化dist数组
10     // 原理：如果某条最短路径上有n个点（除了自己），那么加上自己之后一共有n+1个点，由抽屉原理一定有两个点相同，所以存在环。

```

```

11
12     queue<int> q;
13     for (int i = 1; i <= n; i ++ )
14     {
15         q.push(i);
16         st[i] = true;
17     }
18
19     while (q.size())
20     {
21         auto t = q.front();
22         q.pop();
23
24         st[t] = false;
25
26         for (int i = h[t]; i != -1; i = ne[i])
27         {
28             int j = e[i];
29             if (dist[j] > dist[t] + w[i])
30             {
31                 dist[j] = dist[t] + w[i];
32                 cnt[j] = cnt[t] + 1;
33                 if (cnt[j] >= n) return true;          // 如果从1号点到x的最短路中包含至少
n个点（不包括自己），则说明存在环
34                 if (!st[j])
35                 {
36                     q.push(j);
37                     st[j] = true;
38                 }
39             }
40         }
41     }
42
43     return false;
44 }

```

## floyd算法

$O(n^3)$

```

1  初始化:
2      for (int i = 1; i <= n; i ++ )
3          for (int j = 1; j <= n; j ++ )
4              if (i == j) d[i][j] = 0;
5              else d[i][j] = INF;
6
7  // 算法结束后, d[a][b]表示a到b的最短距离
8  void floyd()
9  {
10     for (int k = 1; k <= n; k ++ )
11         for (int i = 1; i <= n; i ++ )
12             for (int j = 1; j <= n; j ++ )
13                 d[i][j] = min(d[i][j], d[i][k] + d[k][j]);
14 }

```

## 朴素版prim算法

$\mathcal{O}(n^2 + m)$

```
1  int n;          // n表示点数
2  int g[N][N];     // 邻接矩阵, 存储所有边
3  int dist[N];     // 存储其他点到当前最小生成树的距离
4  bool st[N];      // 存储每个点是否已经在生成树中
5
6
7  // 如果图不连通, 则返回INF(值是0x3f3f3f3f), 否则返回最小生成树的树边权重之和
8  int prim()
9  {
10     memset(dist, 0x3f, sizeof dist);
11
12     int res = 0;
13     for (int i = 0; i < n; i ++ )
14     {
15         int t = -1;
16         for (int j = 1; j <= n; j ++ )
17             if (!st[j] && (t == -1 || dist[t] > dist[j]))
18                 t = j;
19
20         if (i && dist[t] == INF) return INF;
21
22         if (i) res += dist[t];
23         st[t] = true;
24
25         for (int j = 1; j <= n; j ++ ) dist[j] = min(dist[j], g[t][j]);
26     }
27
28     return res;
29 }
```

## Kruskal算法

$\mathcal{O}(m \log m)$

```
1  int n, m;        // n是点数, m是边数
2  int p[N];        // 并查集的父节点数组
3
4  struct Edge      // 存储边
5  {
6      int a, b, w;
7
8      bool operator< (const Edge &w) const
9      {
10         return w < w.w;
11     }
12 } edges[M];
13
14 int find(int x)    // 并查集核心操作
15 {
```



```

16     if (p[x] != x) p[x] = find(p[x]);
17     return p[x];
18 }
19
20 int kruskal()
21 {
22     sort(edges, edges + m);
23
24     for (int i = 1; i <= n; i++) p[i] = i;    // 初始化并查集
25
26     int res = 0, cnt = 0;
27     for (int i = 0; i < m; i++)
28     {
29         int a = edges[i].a, b = edges[i].b, w = edges[i].w;
30
31         a = find(a), b = find(b);
32         if (a != b)    // 如果两个连通块不连通，则将这两个连通块合并
33         {
34             p[a] = b;
35             res += w;
36             cnt++;
37         }
38     }
39
40     if (cnt < n - 1) return INF;
41     return res;
42 }

```

## 染色法判别二分图

$O(n + m)$

```

1  int n;    // n表示点数
2  int h[N], e[M], ne[M], idx;    // 邻接表存储图
3  int color[N];    // 表示每个点的颜色，-1表示未染色，0表示白色，1表示黑色
4
5  // 参数: u表示当前节点, c表示当前点的颜色
6  bool dfs(int u, int c)
7  {
8      color[u] = c;
9      for (int i = h[u]; i != -1; i = ne[i])
10     {
11         int j = e[i];
12         if (color[j] == -1)
13         {
14             if (!dfs(j, !c)) return false;
15         }
16         else if (color[j] == c) return false;
17     }
18
19     return true;
20 }
21
22 bool check()
23 {

```

```

24     memset(color, -1, sizeof color);
25     bool flag = true;
26     for (int i = 1; i <= n; i ++ )
27         if (color[i] == -1)
28             if (!dfs(i, 0))
29                 {
30                     flag = false;
31                     break;
32                 }
33     return flag;
34 }

```

## 匈牙利算法

$O(nm)$

```

1  int n1, n2;    // n1表示第一个集合中的点数，n2表示第二个集合中的点数
2  int h[N], e[M], ne[M], idx;    // 邻接表存储所有边，匈牙利算法中只会用到从第一个集合指向
   // 第二个集合的边，所以这里只用存一个方向的边
3  int match[N];    // 存储第二个集合中的每个点当前匹配的第一个集合中的点是哪个
4  bool st[N];    // 表示第二个集合中的每个点是否已经被遍历过
5
6  bool find(int x)
7  {
8      for (int i = h[x]; i != -1; i = ne[i])
9      {
10         int j = e[i];
11         if (!st[j])
12         {
13             st[j] = true;
14             if (match[j] == 0 || find(match[j]))
15             {
16                 match[j] = x;
17                 return true;
18             }
19         }
20     }
21     return false;
22 }
23
24 // 求最大匹配数，依次枚举第一个集合中的每个点能否匹配第二个集合中的点
25 int res = 0;
26 for (int i = 1; i <= n1; i ++ )
27 {
28     memset(st, false, sizeof st);
29     if (find(i)) res ++ ;
30 }
31

```

## 网络流

```

1  int t, n, m, S, T;
2  //=====

```

```

3 struct edge {
4     int to; ll cap; int rev;
5 };
6
7 vector<edge> G[maxn];
8 int level[maxn];
9 int iter[maxn];
10
11 void add_edge(int from, int to, ll cap) {
12     G[from].push_back((edge){to, cap, G[to].size()});
13     G[to].push_back((edge){from, 0, G[from].size() - 1});
14 }
15
16 void bfs(int s) {
17     memset(level, -1, sizeof level);
18     queue<int> que;
19     level[s] = 0;
20     que.push(s);
21     while (!que.empty()) {
22         int v = que.front();
23         que.pop();
24         for (int i = 0; i < G[v].size(); i++) {
25             edge &e = G[v][i];
26             if (e.cap > 0 && level[e.to] < 0) {
27                 level[e.to] = level[v] + 1;
28                 que.push(e.to);
29             }
30         }
31     }
32 }
33
34 ll dfs(int v, int t, int f) {
35     if (v == t) return f;
36     for (int &i = iter[v]; i < G[v].size(); i++) { // 当前弧优化
37         edge &e = G[v][i];
38         if (e.cap > 0 && level[v] < level[e.to]) {
39             ll d = dfs(e.to, t, min(f * 1ll, e.cap));
40             if (d > 0) {
41                 e.cap -= d;
42                 G[e.to][e.rev].cap += d;
43                 return d;
44             }
45         }
46     }
47     return 0;
48 }
49
50 ll max_flow(int s, int t) {
51     ll flow = 0;
52     for (;;) {
53         bfs(s);
54         if (level[t] < 0) return flow;
55         memset(iter, 0, sizeof iter);
56         ll f;
57         while ((f = dfs(s, t, INF)) > 0) {
58             flow += f;
59         }
60     }

```

```

61 }
62
63
64 //=====
65 void solve()
66 {
67     while(~scanf("%d%d", &n, &m)){
68         S = 0, T = 2 * n + 1;
69         int fd, a, b, sm = 0;
70         // !!!!!!!!!!!!!n * 2 + 1
71         for (int i = 0; i < n * 2 + 1; i++) G[i].clear();
72         for (int i = 1; i <= n; i++) {
73             scanf("%d", &fd);
74             add_edge(S, i, fd);
75             add_edge(i + n, T, fd);
76             sm += fd;
77         }
78
79         for (int i = 1; i <= m; i++) {
80             scanf("%d%d", &a, &b);
81             add_edge(a, b + n, 1);
82             add_edge(b, a + n, 1);
83         }
84
85         if (sm & 1) {
86             puts("No");
87         } else if (max_flow(S, T) == sm) {
88             puts("Yes");
89         } else {
90             puts("No");
91         }
92     }
93 }
94
95 int main()
96 {
97
98     // freopen("F:/Overflow/in.txt", "r", stdin);
99     // ios::sync_with_stdio(false);
100     solve();
101     return 0;
102 }

```

## 最小费用流

```

1  ll sum[maxn];
2  // =====
3  int t, n, m, s, q;
4  // first->shortest dis, second->node
5  typedef pair<int, int> PII;
6
7  struct edge {
8      int to, cap, cost, rev;

```

```

9   };
10
11  vector<edge> G[maxn];
12  int h[maxn];           // 顶点的势
13  int dis[maxn];         // 最短距离
14  int prevv[maxn], preve[maxn]; // 最短路中的前驱节点和对应的边
15
16  vector<int> res;
17  // 加一条从from到to容量为cap费用为cost的边
18  void add_edge(int from, int to, int cap, int cost) {
19      G[from].push_back((edge){to, cap, cost, G[to].size()});
20      G[to].push_back((edge){from, 0, -cost, G[from].size() - 1});
21  }
22
23  ll flow;
24  ll min_cost_flow(int s, int t, ll f = INF) {
25      ll ans = 0;
26      flow = 0;
27      fill(h, h + t + 1, 0);
28      fill(prevv, prevv + t + 1, 0);
29      fill(preve, preve + t + 1, 0);
30      while (f > 0) {
31          priority_queue<PII, vector<PII>, greater<PII>> que;
32          fill(dis, dis + t + 1, INF);
33          dis[s] = 0;
34          que.push(PII(0, s));
35          while (!que.empty()) {
36              PII p = que.top();
37              que.pop();
38              int v = p.second;
39              if (dis[v] < p.first) continue;
40              for (int i = 0; i < G[v].size(); i++) {
41                  edge &e = G[v][i];
42                  if (e.cap > 0 && dis[e.to] > dis[v] + e.cost + h[v] - h[e.to]) {
43                      dis[e.to] = dis[v] + e.cost + h[v] - h[e.to];
44                      prevv[e.to] = v;
45                      preve[e.to] = i;
46                      que.push(PII(dis[e.to], e.to));
47                  }
48              }
49          }
50          if (dis[t] == INF) {
51              return ans;
52          }
53          for (int v = 0; v <= t + 1; v++) h[v] += dis[v];
54          int d = f;
55          for (int v = t; v != s; v = prevv[v]) {
56              d = min(d, G[prevv[v]][preve[v]].cap);
57          }
58          f -= d;
59          flow += d;
60          ans += d * h[t];
61          res.push_back(h[t] * d);
62          for (int v = t; v != s; v = prevv[v]) {
63              G[prevv[v]][preve[v]].cap -= d;
64              G[v][G[prevv[v]][preve[v]].rev].cap += d;
65          }
66      }

```

```

67     return ans;
68 }
69
70 void init() {
71     res.clear();
72     for (int i = 0; i < maxn; i++) {
73         G[i].clear();
74     }
75 }
76 // =====
77 void solve()
78 {
79     while (~scanf("%d%d", &n, &m)) {
80         init();
81         for (int i = 1; i <= m; i++) {
82             int a, b, c;
83             scanf("%d%d%d", &a, &b, &c);
84             add_edge(a, b, 1, c);
85         }
86
87         res.push_back(0);
88         min_cost_flow(1, n);
89         sort(res.begin(), res.end());
90         for (int i = 1; i < res.size(); i++) {
91             sum[i] = sum[i - 1] + res[i];
92         }
93         scanf("%d", &q);
94         while (q--) {
95             ll u, v;
96             scanf("%lld%lld", &u, &v);
97             if (u * (res.size() - 1) < v) {
98                 puts("NaN");
99             } else {
100                 int pos = v / u;
101                 ll ans = sum[pos] * u;
102                 if (u * pos < v) {
103                     ans += res[pos + 1] * (v - u * pos);
104                 }
105                 ll gc = __gcd(ans, v);
106                 printf("%lld/%lld\n", ans / gc, v / gc);
107             }
108         }
109     }
110 }
111
112 int main()
113 {
114
115     // freopen("F:/Overflow/in.txt", "r", stdin);
116     // ios::sync_with_stdio(false);
117     solve();
118     return 0;
119 }
120

```

## 数学知识

### 试除法分解质因数

```
1 void divide(int x)
2 {
3     for (int i = 2; i <= x / i; i ++ )
4         if (x % i == 0)
5         {
6             int s = 0;
7             while (x % i == 0) x /= i, s ++ ;
8             cout << i << ' ' << s << endl;
9         }
10    if (x > 1) cout << x << ' ' << 1 << endl;
11    cout << endl;
12 }
```

### 线性筛法求素数

```
1 int primes[N], cnt;    // primes[]存储所有素数
2 bool st[N];           // st[x]存储x是否被筛掉
3
4 void get_primes(int n)
5 {
6     for (int i = 2; i <= n; i ++ )
7     {
8         if (!st[i]) primes[cnt ++ ] = i;
9         for (int j = 0; primes[j] <= n / i; j ++ )
10        {
11            st[primes[j] * i] = true;
12            if (i % primes[j] == 0) break;
13        }
14    }
15 }
```

### 试除法求所有约数

```

1  vector<int> get_divisors(int x)
2  {
3      vector<int> res;
4      for (int i = 1; i <= x / i; i ++ )
5          if (x % i == 0)
6          {
7              res.push_back(i);
8              if (i != x / i) res.push_back(x / i);
9          }
10     sort(res.begin(), res.end());
11     return res;
12 }

```

## 求欧拉函数

```

1  int phi(int x)
2  {
3      int res = x;
4      for (int i = 2; i <= x / i; i ++ )
5          if (x % i == 0)
6          {
7              res = res / i * (i - 1);
8              while (x % i == 0) x /= i;
9          }
10     if (x > 1) res = res / x * (x - 1);
11
12     return res;
13 }

```

## 筛法求欧拉函数

```

1  int primes[N], cnt;    // primes[]存储所有素数
2  int euler[N];          // 存储每个数的欧拉函数
3  bool st[N];            // st[x]存储x是否被筛掉
4
5
6  void get_eulers(int n)
7  {
8      euler[1] = 1;
9      for (int i = 2; i <= n; i ++ )
10     {
11         if (!st[i])
12         {
13             primes[cnt ++ ] = i;
14             euler[i] = i - 1;
15         }
16         for (int j = 0; primes[j] <= n / i; j ++ )

```



```

17     {
18         int t = primes[j] * i;
19         st[t] = true;
20         if (i % primes[j] == 0)
21         {
22             euler[t] = euler[i] * primes[j];
23             break;
24         }
25         euler[t] = euler[i] * (primes[j] - 1);
26     }
27 }
28 }

```

## 扩展欧几里得算法

```

1 // 求x, y, 使得ax + by = gcd(a, b)
2 int exgcd(int a, int b, int &x, int &y)
3 {
4     if (!b)
5     {
6         x = 1; y = 0;
7         return a;
8     }
9     int d = exgcd(b, a % b, y, x);
10    y -= (a/b) * x;
11    return d;
12 }

```

## 高斯消元

```

1 // a[N][N]是增广矩阵
2 int gauss()
3 {
4     int c, r;
5     for (c = 0, r = 0; c < n; c++)
6     {
7         int t = r;
8         for (int i = r; i < n; i++) // 找到绝对值最大的行
9             if (fabs(a[i][c]) > fabs(a[t][c]))
10                t = i;
11
12         if (fabs(a[t][c]) < eps) continue;
13
14         for (int i = c; i <= n; i++) swap(a[t][i], a[r][i]); // 将绝对值最大的
// 行换到最顶端
15         for (int i = n; i >= c; i--) a[r][i] /= a[r][c]; // 将当前上的首位变成1
16         for (int i = r + 1; i < n; i++) // 用当前行将下面所有的列消成0
17             if (fabs(a[i][c]) > eps)

```

```

18         for (int j = n; j >= c; j -- )
19             a[i][j] -= a[r][j] * a[i][c];
20
21         r ++ ;
22     }
23
24     if (r < n)
25     {
26         for (int i = r; i < n; i ++ )
27             if (fabs(a[i][n]) > eps)
28                 return 2; // 无解
29         return 1; // 有无穷多组解
30     }
31
32     for (int i = n - 1; i >= 0; i -- )
33         for (int j = i + 1; j < n; j ++ )
34             a[i][n] -= a[i][j] * a[j][n];
35
36     return 0; // 有唯一解
37 }

```

## 递归法求组合数

```

1 // c[a][b] 表示从a个苹果中选b个的方案数
2 for (int i = 0; i < N; i ++ )
3     for (int j = 0; j <= i; j ++ )
4         if (!j) c[i][j] = 1;
5         else c[i][j] = (c[i - 1][j] + c[i - 1][j - 1]) % mod;

```

## 通过预处理逆元的方式求组合数

```

1 首先预处理出所有阶乘取模的余数fact[N]，以及所有阶乘取模的逆元infact[N]
2 如果取模的数是质数，可以用费马小定理求逆元
3 int qmi(int a, int k, int p) // 快速幂模板
4 {
5     int res = 1;
6     while (k)
7     {
8         if (k & 1) res = (LL)res * a % p;
9         a = (LL)a * a % p;
10        k >>= 1;
11    }
12    return res;
13 }
14
15 // 预处理阶乘的余数和阶乘逆元的余数
16 fact[0] = infact[0] = 1;
17 for (int i = 1; i < N; i ++ )

```

```

18 {
19     fact[i] = (LL)fact[i - 1] * i % mod;
20     infact[i] = (LL)infact[i - 1] * qmi(i, mod - 2, mod) % mod;
21 }

```

## Lucas定理

```

1  若p是质数，则对于任意整数  $1 \leq m \leq n$ ，有：
2       $C(n, m) = C(n \% p, m \% p) * C(n / p, m / p) \pmod p$ 
3
4  int qmi(int a, int k)        // 快速幂模板
5  {
6      int res = 1;
7      while (k)
8      {
9          if (k & 1) res = (LL)res * a % p;
10         a = (LL)a * a % p;
11         k >>= 1;
12     }
13     return res;
14 }
15
16
17 int C(int a, int b)          // 通过定理求组合数C(a, b)
18 {
19     int res = 1;
20     for (int i = 1, j = a; i <= b; i ++, j -- )
21     {
22         res = (LL)res * j % p;
23         res = (LL)res * qmi(i, p - 2) % p;
24     }
25     return res;
26 }
27
28
29 int lucas(LL a, LL b)
30 {
31     if (a < p && b < p) return C(a, b);
32     return (LL)C(a % p, b % p) * lucas(a / p, b / p) % p;
33 }
34

```

## 分解质因数法求组合数

- 1 当我们需要求出组合数的真实值，而非对某个数的余数时，分解质因数的方式比较好用：
- 2 1. 筛法求出范围内的所有质数
- 3 2. 通过  $C(a, b) = a! / b! / (a - b)!$  这个公式求出每个质因子的次数。  $n!$  中  $p$  的次数是  $n / p + n / p^2 + n / p^3 + \dots$

```

4      3. 用高精度乘法将所有质因子相乘
5
6      int primes[N], cnt;    // 存储所有质数
7      int sum[N];           // 存储每个质数的次数
8      bool st[N];           // 存储每个数是否已被筛掉
9
10
11     void get_primes(int n)    // 线性筛法求素数
12     {
13         for (int i = 2; i <= n; i ++ )
14         {
15             if (!st[i]) primes[cnt ++ ] = i;
16             for (int j = 0; primes[j] <= n / i; j ++ )
17             {
18                 st[primes[j] * i] = true;
19                 if (i % primes[j] == 0) break;
20             }
21         }
22     }
23
24
25     int get(int n, int p)      // 求n! 中的次数
26     {
27         int res = 0;
28         while (n)
29         {
30             res += n / p;
31             n /= p;
32         }
33         return res;
34     }
35
36
37     vector<int> mul(vector<int> a, int b)    // 高精度乘低精度模板
38     {
39         vector<int> c;
40         int t = 0;
41         for (int i = 0; i < a.size(); i ++ )
42         {
43             t += a[i] * b;
44             c.push_back(t % 10);
45             t /= 10;
46         }
47
48         while (t)
49         {
50             c.push_back(t % 10);
51             t /= 10;
52         }
53
54         return c;
55     }
56
57     get_primes(a);    // 预处理范围内的所有质数
58
59     for (int i = 0; i < cnt; i ++ )    // 求每个质因数的次数
60     {
61         int p = primes[i];

```

```

62     sum[i] = get(a, p) - get(b, p) - get(a - b, p);
63 }
64
65 vector<int> res;
66 res.push_back(1);
67
68 for (int i = 0; i < cnt; i++) // 用高精度乘法将所有质因子相乘
69     for (int j = 0; j < sum[i]; j++)
70         res = mul(res, primes[i]);

```

## 卡特兰数

1 给定n个0和n个1，它们按照某种顺序排成长度为2n的序列，满足任意前缀中0的个数都不少于1的个数的序列的数量为： $Cat(n) = C(2n, n) / (n + 1)$

## 动态规划

### 背包问题

01

```

1     for (int i = 1; i <= n; i++)
2     {
3         for (int j = 0; j <= V; j++)
4         {
5             dp[i][j] = dp[i - 1][j];
6             if (j >= v[i])
7             {
8                 dp[i][j] = max(dp[i][j], dp[i - 1][j - v[i]] + w[i]);
9             }
10        }
11    }
12    cout << dp[n][V] << endl;

```

多重

```

1     int con = 0;
2     for (int i = 1; i <= N; i++)
3     {

```

```

4      int a, b, s;
5      cin >> a >> b >> s;
6      int k = 1;
7      while (k <= s)
8      {
9          con++;
10         v[con] = a * k;
11         w[con] = b * k;
12         s -= k;
13         k *= 2;
14     }
15
16     if (s > 0)
17     {
18         con++;
19         v[con] = a * s;
20         w[con] = b * s;
21     }
22
23 }
24
25 for (int i = 1; i <= con; i++)
26 {
27     for (int j = V; j >= v[i]; j--)
28     {
29         dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
30     }
31 }
32 cout << dp[V] << endl;

```

## 分组

```

1      cin >> N >> V;
2      for (int i = 1; i <= N; i++)
3      {
4          cin >> s[i];
5          for (int j = 1; j <= s[i]; j++)
6              cin >> v[i][j] >> w[i][j];
7      }
8
9      for (int i = 1; i <= N; i++)
10         for (int j = 0; j <= V; j++)
11         {
12             dp[i][j] = dp[i - 1][j];
13             for (int k = 0; k <= s[i]; k++)
14             {
15
16                 if (v[i][k] <= j)
17                     dp[i][j] = max(dp[i][j], dp[i - 1][j - v[i][k]] + w[i][k]);
18             }
19         }
20
21     cout << dp[N][V] << endl;

```

完全

```
1      cin >> n >> V;
2      for (int i = 1; i <= n; i++)
3      {
4          cin >> v[i] >> w[i];
5      }
6
7      for (int i = 1; i <= n; i++)
8      {
9          for (int j = v[i]; j <= V; j++)
10         {
11             dp[j] = max(dp[j], dp[j - v[i]] + w[i]);
12         }
13     }
14
15     cout << dp[V] << endl;
```

## 线性DP

最长上升子序列

```
1      cin >> t;
2      for (int i = 0; i < t; i++)
3      {
4          cin >> num[i];
5          dp[i] = 1;
6      }
7      for (int i = 0; i < t; i++)
8      {
9          for (int j = 0; j < i; j++)
10         {
11             if (num[i] > num[j])
12                 dp[i] = max(dp[i], dp[j] + 1);
13         }
14     }
15     cout << *max_element(dp, dp + t) << endl;
```

最长上升子序列2

```
1      scanf("%d", &n);
2      for (int i = 0; i < n; i++) scanf("%d", &a[i]);
3
4      int len = 0;
5      for (int i = 0; i < n; i++)
6      {
7          int l = 0, r = len;
8          while (l < r)
9          {
10             int mid = l + r + 1 >> 1;
```

```

11         if (q[mid] < a[i]) l = mid;
12         else r = mid - 1;
13     }
14     len = max(len, r + 1);
15     q[r + 1] = a[i];
16 }
17 printf("%d\n", len);

```

## 最长公共子序列

```

1     cin >> n >> m;
2     cin >> s1 + 1 >> s2 + 1;
3     for (int i = 1; i <= n; i++)
4     {
5         for (int j = 1; j <= m; j++)
6         {
7             dp[i][j] = max(dp[i - 1][j], dp[i][j - 1]);
8             if (s1[i] == s2[j]) dp[i][j] = max(dp[i][j], dp[i - 1][j - 1] + 1);
9         }
10    }
11    cout << dp[n][m] << endl;

```

## 最小编辑距离

```

1     void solve()
2     {
3         cin >> n >> str1 + 1;
4         cin >> m >> str2 + 1;
5
6         for (int i = 0; i <= m; i++) dp[0][i] = i; //增加
7         for (int i = 0; i <= n; i++) dp[i][0] = i; //删
8
9         for (int i = 1; i <= n; i++)
10        {
11            for (int j = 1; j <= m; j++)
12            {
13                dp[i][j] = min(dp[i - 1][j] + 1, dp[i][j - 1] + 1);
14                if (str1[i] == str2[j]) dp[i][j] = min(dp[i][j], dp[i - 1][j - 1]);
15                else dp[i][j] = min(dp[i][j], dp[i - 1][j - 1] + 1);
16            }
17        }
18        cout << dp[n][m] << endl;
19    }

```

## 区间DP

```

1     cin >> t;
2     for (int i = 1; i <= t; i++)
3         cin >> num[i];
4     for (int i = 1; i <= t; i++)
5         sum[i] += sum[i - 1] + num[i];

```



```

6     for (int i = 2; i <= t; i++)//长度1的时候不需要代价，长度从小到大开始枚举
7     {
8         for (int j = 1; j + i - 1 <= t; j++)//起始位置
9         {
10            int l = j, r = j + i - 1;
11            dp[l][r] = 1e8;
12            for (int k = l; k <= r - 1; k++)
13            {
14                dp[l][r] = min(dp[l][r], dp[l][k] + dp[k + 1][r] + sum[r] - sum[l -
15            1]);
16            }
17        }
18    }
19    cout << dp[1][t] << endl;

```

## 博弈论

### NIM游戏

```

1     scanf("%d", &n);
2
3     int res = 0;
4     while (n -- )
5     {
6         int x;
7         scanf("%d", &x);
8         res ^= x;
9     }
10
11    if (res) puts("Yes");
12    else puts("No");

```

### 集合nim

给定 $n$ 堆石子以及一个由 $k$ 个不同正整数构成的数字集合 $S$ 。

现在有两位玩家轮流操作，每次操作可以从任意一堆石子中拿取石子，每次拿取的石子数量必须包含于集合 $S$ ，最后无法进行操作的人视为失败。

问如果两人都采用最优策略，先手是否必胜。

```

1 //SG函数
2 int SG(int x)
3 {
4     if (f[x] != -1) return f[x];
5
6     unordered_set<int> S;
7     for (int i = 0; i < m; i++)
8     {

```

```

9         int sum = s[i];
10        if (x >= sum) S.insert(SG(x - sum));
11    }
12    for (int i = 0; ; i++)
13    {
14        if (!S.count(i))
15            return f[x] = i;
16    }
17 }
18
19
20 //=====
21 void solve()
22 {
23     memset(f, -1, sizeof f);
24     cin >> m;
25     for (int i = 0; i < m; i++)
26         cin >> s[i];
27     cin >> n;
28     int res = 0;
29     for (int i = 0; i < n; i++)
30     {
31         int x;
32         cin >> x;
33         res ^= SG(x);
34     }
35
36     if (res) puts("Yes");
37     else puts("No");
38 }

```

## 高级数据结构

### 树状数组

```

1  int lowbit(int x) {
2      return x & -x;
3  }
4
5  void add(int x, int k) {
6      for (int i = x; i <= n; i += lowbit(i)) e[i] += k;
7  }
8
9  int sum(int x) {
10     int res = 0;
11     for (int i = x; i; i -= lowbit(i)) res += e[i];
12     return res;
13 }

```

## 最大数

```

1  struct node {
2      int l, r;
3      ll MAX;
4  }tr[maxn << 2];
5  int n, m, p;
6  void pushup(int p) {
7      tr[p].MAX = max(tr[p << 1].MAX, tr[p << 1 | 1].MAX);
8  }
9
10 void build(int p, int l, int r) {
11     tr[p] = {l, r};
12     if (l == r) {
13         // tr[p].MAX = a[l];
14         return ;
15     }
16     int mid = l + r >> 1;
17     build(p << 1, l, mid);
18     build(p << 1 | 1, mid + 1, r);
19     // pushup(p);
20 }
21
22 void modify(int p, int x, int k) {
23     if (tr[p].l == tr[p].r) {
24         tr[p].MAX = k;
25         return ;
26     }
27
28     int mid = tr[p].l + tr[p].r >> 1;
29
30     if (x <= mid) {
31         modify(p << 1, x, k);
32     }
33     if (x > mid) {
34         modify(p << 1 | 1, x, k);
35     }
36     pushup(p);
37 }
38
39 ll query(int p, int l, int r) {
40     if (l <= tr[p].l && tr[p].r <= r) {
41         return tr[p].MAX ;
42     }
43     int mid = tr[p].l + tr[p].r >> 1;
44     if (r <= mid) return query(p << 1, l, r);
45     if (l > mid) return query(p << 1 | 1, l, r);
46     return max(query(p << 1, l, mid), query(p << 1 | 1, mid + 1, r));
47 }

```

## 区间修改

```

1  struct node{

```

```

2     int l, r;
3     ll sum, add;
4 }tr[MAXN * 4];
5 ll a[MAXN];
6 void pushup(int p) {
7     tr[p].sum = tr[p << 1].sum + tr[p << 1 | 1].sum;
8 }
9
10 void pushdown(int p) {
11     node &root = tr[p], &left = tr[p << 1], &right = tr[p << 1 | 1];
12     if (root.add) {
13         left.sum += (ll)(left.r - left.l + 1) * (root.add), left.add += root.add;
14         right.sum += (ll)(right.r - right.l + 1) * (root.add), right.add +=
root.add;
15         root.add = 0;
16     }
17 }
18
19 void build(int p, int l, int r) {
20     tr[p] = {l, r};
21     if (l == r) {
22         tr[p] = {l, r, a[l], 0};
23         return ;
24     }
25     int mid = tr[p].l + tr[p].r >> 1;
26     build(p << 1, l, mid), build(p << 1 | 1, mid + 1, r);
27     pushup(p);
28 }
29
30 void modify(int p, int l, int r, int k) {
31     if (tr[p].l == tr[p].r) {
32         tr[p].sum += (ll)(tr[p].r - tr[p].l + 1) * k;
33         tr[p].add += k;
34         return ;
35     }
36     pushdown(p);
37     int mid = tr[p].l + tr[p].r >> 1;
38     if (l <= mid) modify(p << 1, l, r, k);
39     if (r > mid) modify(p << 1 | 1, l, r, k);
40     pushup(p);
41 }
42
43 //ll query(int p, int l, int r) {
44 //     if (l <= tr[p].l && tr[p].r <= r) {
45 //         return tr[p].sum;
46 //     }
47 //
48 //     pushdown(p);
49 //     int mid = tr[p].l + tr[p].r >> 1;
50 //     if (r <= mid) return query(p << 1, l, r);
51 //     if (l > mid) return query(p << 1 | 1, l, r);
52 //     return query(p << 1, l, mid) + query(p << 1 | 1, mid + 1, r);
53 // }
54 //}
55 ll query(int p, int l, int r) {
56     if (l <= tr[p].l && tr[p].r <= r) return tr[p].sum;
57     pushdown(p);
58     int mid = tr[p].l + tr[p].r >> 1;

```

```

59     ll res = 0;
60     if (l <= mid) res += query(p << 1, l, r);
61     if (r > mid) res += query(p << 1 | 1, l, r);
62     return res;
63 }

```

## 最大公约数

```

1  struct node{
2      int l, r;
3      ll d, sum;
4  }tr[MAXN * 4];
5
6  ll a[MAXN];
7  ll gcd(ll a, ll b) {
8      return b == 0 ? a : gcd(b, a % b);
9  }
10
11 void pushup(node &p, node &l, node &r) {
12     p.sum = l.sum + r.sum;
13     p.d = gcd(l.d, r.d);
14 }
15
16 void pushup(int p) {
17     pushup(tr[p], tr[p << 1], tr[p << 1 | 1]);
18 }
19
20 void build(int p, int l, int r) {
21     tr[p] = {l, r};
22     if (l == r) {
23         ll b = a[r] - a[r - 1];
24         tr[p] = {l, r, b, b};
25         return;
26     }
27     int mid = tr[p].l + tr[p].r >> 1;
28     build(p << 1, l, mid), build(p << 1 | 1, mid + 1, r);
29     pushup(p);
30 }
31
32 void modify(int p, int x, ll v) {
33     if (tr[p].l == x && tr[p].r == x) {
34         ll b = tr[p].sum + v;
35         tr[p] = {x, x, b, b};
36         return;
37     }
38     int mid = tr[p].l + tr[p].r >> 1;
39     if (x <= mid) modify(p << 1, x, v);
40     else modify(p << 1 | 1, x, v);
41     pushup(p);
42 }
43
44 node query(int p, int l, int r) {
45     if (tr[p].l >= l && tr[p].r <= r) return tr[p];
46     int mid = tr[p].l + tr[p].r >> 1;

```

```

47     if (r <= mid) return query(p << 1, l, r);
48     if (l > mid) return query(p << 1 | 1, l, r);
49     else {
50         node left = query(p << 1, l, r);
51         node right = query(p << 1 | 1, l, r);
52         node res;
53         pushup(res, left, right);
54         return res;
55     }
56 }

```

## AC自动机

给定  $n$  个长度不超过 5050 的由小写英文字母组成的单词，以及一篇长为  $m$  的文章。

请问，有多少个单词在文章中出现了。

```

1  const int N = 10010, S = 55, M = 1000010;
2
3  int n;
4  int tr[N * S][26], cnt[N * S], idx;
5  char str[M];
6  int q[N * S], ne[N * S];
7
8  void insert()
9  {
10     int p = 0;
11     for (int i = 0; str[i]; i++)
12     {
13         int t = str[i] - 'a';
14         if (!tr[p][t]) tr[p][t] = ++idx;
15         p = tr[p][t];
16     }
17     cnt[p]++;
18 }
19
20 void build()
21 {
22     int hh = 0, tt = -1;
23     for (int i = 0; i < 26; i++)
24         if (tr[0][i])
25             q[++tt] = tr[0][i];
26
27     while (hh <= tt)
28     {
29         int t = q[hh++];
30         for (int i = 0; i < 26; i++)
31         {
32             int p = tr[t][i];
33             if (!p) tr[t][i] = tr[ne[t]][i];

```

```

34         else
35         {
36             ne[p] = tr[ne[t]][i];
37             q[ ++ tt] = p;
38         }
39     }
40 }
41 }
42
43 int main()
44 {
45     int T;
46     scanf("%d", &T);
47     while (T -- )
48     {
49         memset(tr, 0, sizeof tr);
50         memset(cnt, 0, sizeof cnt);
51         memset(ne, 0, sizeof ne);
52         idx = 0;
53
54         scanf("%d", &n);
55         for (int i = 0; i < n; i ++ )
56         {
57             scanf("%s", str);
58             insert();
59         }
60         build();
61         scanf("%s", str);
62         int res = 0;
63         for (int i = 0, j = 0; str[i]; i ++ )
64         {
65             int t = str[i] - 'a';
66             j = tr[j][t];
67
68             int p = j;
69             while (p)
70             {
71                 res += cnt[p];
72                 cnt[p] = 0;
73                 p = ne[p];
74             }
75         }
76
77         printf("%d\n", res);
78     }
79
80     return 0;
81 }

```