

目录

1	day1 图论	1
1.1	有向图强连通分量的 Tarjan 算法	1
1.2	图的割点、桥与双连通分量	5
1.3	2-SAT	8
1.4	欧拉回路	14
2	day2 字符串（一）	16
2.1	KMP	16
2.2	Trie	19
2.3	Aho-Corasick Automaton	21
2.4	Manacher	23
3	day3 简单数学	24
3.1	整除及剩余	24
3.2	素数	25
3.3	欧几里得算法	29
3.4	线性同余方程	30
3.5	逆元	31
3.6	离散对数问题	32
3.7	原根	34

1 day1 图论

1.1 有向图强连通分量的 Tarjan 算法

定义 在有向图 G 中, 如果两个顶点 u, v 间存在一条路径 u 到 v 的路径且也存在一条 v 到 u 的路径, 则称这两个顶点 u, v 是**强连通的 (strongly connected)**。如果有向图 G 的每两个顶点都强连通, 称 G 是一个**强连通图**。有向非强连通图的极大强连通子图, 称为**强连通分量 (strongly connected components)**。若将有向图中的强连通分量都缩为一个点, 则原图会形成一个 DAG (有向无环图)。

极大强连通子图 G 是一个极大强连通子图当且仅当 G 是一个强连通子图且不存在另一个强连通子图 G' 使得 G 是 G' 的真子集。

Tarjan 算法 定义 $dfn(u)$ 为结点 u 搜索的次序编号, 给出函数 $low(u)$ 使得

$low(u) = \min$

- {
- $dfn(u)$,
- $low(v)$, (u, v) 为树枝边, u 为 v 的父结点
- $dfn(v)$ (u, v) 为后向边或指向栈中结点的横叉边
- }

当结点 u 的搜索过程结束后, 若 $dfn(u) = low(u)$, 则以 u 为根的搜索子树上所有还在栈中的结点是一个强连通分量。

代码

tarjan - SCC

```

1 void tarjan(int u)
2 {
3     dfn[u] = low[u] = ++idx;
4     st[top++] = u;
5     for (Edge cur : G[u])
6         if (!dfn[cur.to])
7             tarjan(cur.to),
8             low[u] = min(low[u], low[cur.to]);
9     else if (!scc[cur.to])
10        low[u] = min(low[u], dfn[cur.to]);
11    if (dfn[u] == low[u] && ++cnt)
12        do scc[st[--top]] = cnt;
13        while (st[top] != u);
14 }
```

练习题

POJ2186/BZOJ1051 - Popular Cows 双倍的快乐

Popular Cows

```

1 #include <cstdio>
2 inline int min(int a, int b) { return a < b ? a : b; }
```

```

3 int head[10010], next[50010], to[50010], ecnt;
4 int dfn[10010], low[10010], stk[10010], scc[10010], top, idx, scccnt;
5 bool instk[10010];
6 int deg[10010];
7 inline void addEdge(int f, int t)
8 {
9     ecnt++;
10    next[ecnt] = head[f];
11    head[f] = ecnt;
12    to[ecnt] = t;
13 }
14 void tarjan(int x)
15 {
16     dfn[x] = low[x] = ++idx;
17     instk[stk[top++] = x] = true;
18     for (int cur = head[x]; cur; cur = next[cur])
19         if (!dfn[to[cur]])
20             tarjan(to[cur]), low[x] = min(low[x], low[to[cur]]);
21         else if (instk[to[cur]])
22             low[x] = min(low[x], dfn[to[cur]]);
23     if (dfn[x] == low[x])
24     {
25         scccnt++;
26         do
27         {
28             top--;
29             scc[stk[top]] = scccnt;
30             instk[stk[top]] = false;
31         } while (stk[top] != x);
32     }
33 }
34 int main()
35 {
36     int n, m;
37     scanf("%d%d", &n, &m);
38     for (int i = 0, x, y; i < m; i++)
39     {
40         scanf("%d%d", &x, &y);
41         addEdge(x, y);
42     }
43     for (int i = 1; i <= n; i++)
44         if (!dfn[i])
45             tarjan(i);
46     for (int i = 1; i <= n; i++)
47         for (int cur = head[i]; cur; cur = next[cur])
48             if (scc[i] != scc[to[cur]])
49                 deg[scc[i]]++;
50     int zcnt = 0, id = 0;
51     for (int i = 1; i <= scccnt; i++)
52         if (deg[i] == 0)
53             zcnt++, id = i;
54     if (zcnt != 1)

```

```

55     putchar('0');
56     else
57     {
58         int ans = 0;
59         for (int i = 1; i <= n; i++)
60             if (scc[i] == id)
61                 ans++;
62         printf("%d", ans);
63     }
64     return 0;
65 }

```

POJ3180 - The Cow Prom The $N(2 \leq N \leq 10,000)$ cows are so excited.

The Cow Prom

```

1  #include <cstdio>
2  inline int min(int a, int b) { return a < b ? a : b; }
3  const int maxn = 100010;
4  int head[maxn], next[maxn << 1], to[maxn << 1], ecnt, n, m;
5  int dfn[maxn], scc[maxn], cnt[maxn], scccnt, stk[maxn], low[maxn], idx, top;
6  inline void addEdge(int f, int t)
7  {
8      ecnt++;
9      next[ecnt] = head[f];
10     head[f] = ecnt;
11     to[ecnt] = t;
12 }
13 void tarjan(int x)
14 {
15     dfn[x] = low[x] = ++idx;
16     stk[top++] = x;
17     for (int i = head[x]; i; i = next[i])
18         if (!dfn[to[i]])
19             tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
20         else if (!scc[to[i]])
21             low[x] = min(low[x], dfn[to[i]]);
22     if (dfn[x] == low[x])
23     {
24         scccnt++;
25         do
26             scc[stk[--top]] = scccnt;
27         while (stk[top] != x);
28     }
29 }
30 int main()
31 {
32     scanf("%d%d", &n, &m);
33     for (int i = 0, x, y; i < m; i++)
34     {
35         scanf("%d%d", &x, &y);
36         addEdge(x, y);

```

```

37     }
38     for (int i = 1; i <= n; i++)
39         if (!dfn[i]) tarjan(i);
40     int ans = 0;
41     for (int i = 1; i <= n; i++) cnt[scc[i]]++;
42     for (int i = 1; i <= scccnt; i++)
43         if (cnt[i] > 1) ans++;
44     printf("%d", ans);
45     return 0;
46 }

```

POJ1236 - Network of Schools 强连通分量缩点求出度为 0 的和入度为 0 的分量个数

Network of Schools

```

1  #include <cstdio>
2  inline int min(int a, int b) { return a < b ? a : b; }
3  const int maxn = 110, maxm = 10100;
4  int head[maxn], next[maxm], to[maxm], ecnt, f[maxn], g[maxn];
5  inline void addEdge(int f, int t)
6  {
7      ecnt++;
8      next[ecnt] = head[f];
9      head[f] = ecnt;
10     to[ecnt] = t;
11 }
12 int dfn[maxn], low[maxn], stk[maxn], scc[maxn], scccnt, top, idx;
13 void tarjan(int x)
14 {
15     dfn[x] = low[x] = ++idx;
16     stk[top++] = x;
17     for (int i = head[x]; i; i = next[i])
18         if (!dfn[to[i]])
19             tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
20         else if (!scc[to[i]])
21             low[x] = min(low[x], dfn[to[i]]);
22     if (dfn[x] == low[x])
23     {
24         scccnt++;
25         do
26             scc[stk[--top]] = scccnt;
27         while (stk[top] != x);
28     }
29 }
30 int main()
31 {
32     int n;
33     scanf("%d", &n);
34     for (int i = 1, x; i <= n; i++)
35         for (scanf("%d", &x); x; scanf("%d", &x))
36             addEdge(i, x);
37     for (int i = 1; i <= n; i++)

```

```

38     if (!dfn[i]) tarjan(i);
39     for (int i = 1; i <= n; i++)
40         for (int j = head[i]; j; j = next[j])
41             if (scc[i] != scc[to[j]])
42                 f[scc[i]]++, g[scc[to[j]]]++;
43     int ans1 = 0, ans2 = 0;
44     if (scccnt == 1)
45         printf("1\n0");
46     else
47     {
48         for (int i = 1; i <= scccnt; i++)
49             ans1 += f[i] == 0, ans2 += g[i] == 0;
50         printf("%d\n%d", ans2, ans1 > ans2 ? ans1 : ans2);
51     }
52     return 0;
53 }

```

1.2 图的割点、桥与双连通分量

定义

点连通度与边连通度 在一个无向连通图中，如果有一个顶点集合 V ，删除顶点集合 V ，以及与 V 中顶点相连（至少有一端在 V 中）的所有边后，原图不连通，就称这个点集 V 为**割点集合**。

一个图的**点连通度**的定义为：最小割点集合中的顶点数。

类似的，如果有一个边集合，删除这个边集合以后，原图不连通，就称这个点集为**割边集合**。

双连通图、割点与桥 如果一个无向连通图的点连通度大于 1，则称该图是**点双连通的 (point biconnected)**，简称双连通或重连通。一个图有**割点**，当且仅当这个图的点连通度为 1，则割点集合的唯一元素被称为**割点 (cut point)**，又叫关节点 (articulation point)。一个图可能有多个割点。

如果一个无向连通图的边连通度大于 1，则称该图是**边双连通的 (edge biconnected)**，简称双连通或重连通。一个图有**桥**，当且仅当这个图的边连通度为 1，则割边集合的唯一元素被称为**桥 (bridge)**，又叫关节边 (articulation edge)。一个图可能有多个桥。

可以看出，点双连通与边双连通都可以简称为双连通，它们之间是有着某种联系的，下文中提到的双连通，均既可指点双连通，又可指边双连通。（但这并不意味着它们等价）

双连通分量（分支）：在图 G 的所有子图 G' 中，如果 G' 是双连通的，则称 G' 为双连通子图。如果一个双连通子图 G' 它不是任何一个双连通子图的真子集，则 G' 为极大双连通子图。双连通分量 (biconnected component)，或重连通分量，就是图的极大双连通子图。特殊的，点双连通分量又叫做块。

Tarjan 算法 给出函数 $low(u)$ 使得

$low(u) = \min$

{

$dfn(u)$,

$low(v)$, (u, v) 为树枝边 (父子边)

$dfn(v)$ (u, v) 为后向边 (返祖边) 等价于 $dfn(v) < dfn(u)$ 且 v 不为 u 的父亲结点

}

代码

tarjan - BCC

```

1 void tarjan(int u, int p)
2 {
3     dfn[u] = low[u] = ++idx;
4     for (int e = head[u]; e; e = next[e])
5         if (!dfn[to[e]])
6             tarjan(to[e], u), low[u] = min(low[u], low[to[e]]);
7     else if (to[e] != p)
8         low[u] = min(low[u], dfn[to[e]]);
9 }

```

练习题

POJ3177 - Redundant Paths 将一张有桥图通过加边变成边双连通图，至少要加 $\frac{leaf+1}{2}$ 条边。

Redundant Paths

```

1 #include <cstdio>
2 inline int min(int a, int b) { return a < b ? a : b; }
3 int head[5010], to[20010], next[20010], ecnt, map[5010][5010];
4 int dfn[5010], low[5010], idx, cnt[5010];
5 void addEdge(int f, int t)
6 {
7     ecnt++;
8     next[ecnt] = head[f];
9     head[f] = ecnt;
10    to[ecnt] = t;
11 }
12 void tarjan(int u, int p)
13 {
14     dfn[u] = low[u] = ++idx;
15     for (int e = head[u]; e; e = next[e])
16         if (!dfn[to[e]])
17             tarjan(to[e], u), low[u] = min(low[u], low[to[e]]);
18         else if (to[e] != p)
19             low[u] = min(low[u], dfn[to[e]]);
20 }
21 int main()
22 {
23     int n, m;
24     scanf("%d%d", &n, &m);
25     for (int i = 1, x, y; i <= m; i++)
26     {
27         scanf("%d%d", &x, &y);
28         if (!map[x][y])
29         {
30             addEdge(x, y);
31             addEdge(y, x);
32             map[x][y] = map[y][x] = true;

```



```

33     }
34 }
35 tarjan(1, 0);
36 for (int i = 1; i <= n; i++)
37     for (int e = head[i]; e; e = next[e])
38         if (low[to[e]] != low[i])
39             cnt[low[i]]++;
40 int ans = 0;
41 for (int i = 1; i <= n; i++)
42     ans += cnt[i] == 1;
43 printf("%d", (ans + 1) >> 1);
44 return 0;
45 }

```

POJ1523 - SPF 求割点与删除这个点之后有多少个连通分量

Redundant Paths

```

1  #include <cstdio>
2  #include <cctype>
3  #include <cstring>
4  #define clz(X) memset(X, 0, sizeof(X))
5  inline int max(int a, int b) { return a > b ? a : b; }
6  inline int min(int a, int b) { return a < b ? a : b; }
7  inline void read(int &x)
8  {
9      int ch = x = 0;
10     while (!isdigit(ch = getchar()));
11     for (; isdigit(ch); ch = getchar())
12         x = x * 10 + ch - '0';
13 }
14 int map[1010][1010], range;
15 int dfn[1010], low[1010], idx;
16 int son, subnet[1010];
17 void tarjan(int u)
18 {
19     dfn[u] = low[u] = ++idx;
20     for (int v = 1; v <= range; v++)
21         if (map[u][v])
22             if (!dfn[v])
23             {
24                 tarjan(v);
25                 low[u] = min(low[u], low[v]);
26                 if (low[v] >= dfn[u])
27                     (u == 1 ? son : subnet[u])++;
28             }
29     else
30         low[u] = min(low[u], dfn[v]);
31 }
32 int main()
33 {
34     int x, y, T = 0;

```

```

35     while (read(x), x)
36     {
37         clz(map), clz(dfn), clz(low), clz(subnet), son = idx = 0;
38         read(y);
39         map[x][y] = map[y][x] = 1;
40         range = max(x, y);
41         while (read(x), x)
42         {
43             read(y);
44             map[x][y] = map[y][x] = 1;
45             range = max(range, max(x, y));
46         }
47         printf("Network #%d\n", ++T);
48         tarjan(1);
49         bool flag = false;
50         if (son > 1) subnet[1] = son - 1;
51         for (int i = 1; i <= range; i++)
52             if (subnet[i])
53                 printf("  SPF node %d leaves %d subnets\n", i, subnet[i] + 1),
54                 flag = true;
55         if (!flag)
56             puts("  No SPF nodes");
57         putchar('\n');
58     }
59     return 0;
60 }

```

POJ2942 - Knights of the Round Table 这题过于复杂，我来先给个别人的题解。然后是我自己的实现（仿佛还是没看懂。

//实现被狗吃了

1.3 2-SAT

定义 给定一个布尔方程，判断是否存在一组布尔变量的取值方案，使得整个方程值为真的问题，被称为布尔方程的可满足性问题 (SAT)。SAT 问题是 NP 完全的，但对于一些特殊形式的 SAT 问题我们可以有效求解。

我们将下面这种布尔方程称为合取范式：

$$(a \vee b \vee c \vee \dots) \wedge (d \vee e \vee f \vee \dots) \wedge \dots$$

其中 a, b, c, \dots 称为文字，它是一个布尔变量或其否定。像 $(a \vee b \vee c \vee \dots)$ 这样用 \vee 连接的部分称为子句。如果合取范式的每个子句中的文字个数都不超过两个，那么对应的 SAT 问题又称为 **2-SAT** 问题。

解法 对于给定的 **2-SAT** 问题，首先利用 \Rightarrow 将每个子句 $(a \vee b)$ 改写成等价形式 $(\neg a \Rightarrow b \wedge a \Rightarrow \neg b)$ 。这样原布尔公式就变成了把 $a \Rightarrow b$ 形式的布尔公式用 \wedge 连接起来的形式。

对每个布尔变量 x 构造两个顶点分别代表 x 与 $\neg x$ 。以 \Rightarrow 关系为边建立有向图。若在此图中 a 点能到达 b 点，就表示 a 为真时 b 也一定为真。因此该图中同一个强连通分量中所含的所有变量的布尔值均相同。

若存在某个变量 x ，代表 x 与 $\neg x$ 的两个顶点在同一个强连通分量中，则原布尔表达式的值无法为真。

反之若不存在这样的变量，那么我们先原图中所有的强连通分量缩为一个点，构出一个新图，新图显然

是一个拓扑图，我们求出它的一个拓扑序。那么对于每个变量 x ，

x 所在的强连通分量（新图中的点）的拓扑序在 $\neg x$ 所在的强连通分量之后 $\Leftrightarrow x$ 为真

就是一组合适布尔变量赋值。注意到 Tarjan 算法所求的强连通分量就是按拓扑序的逆序得出的，因此不需要真的缩点建新图求拓扑序，直接利用强连通分量的编号来当做顺序即可。

练习题

POJ3648 - Wedding Additionally, there are several pairs of people conducting adulterous relationships (both different-sex and same-sex relationships are possible)

adulterous relationships

```

1  #include <stdio>
2  #include <cstring>
3  inline int min(int a, int b) { return a < b ? a : b; }
4  const int maxn = 2010, maxm = 500010;
5  int head[maxn], next[maxm], to[maxm], ecnt;
6  inline void addEdge(int f, int t)
7  {
8      next[ecnt] = head[f];
9      head[f] = ecnt;
10     to[ecnt] = t;
11     ecnt++;
12 }
13 int dfn[maxn], low[maxn], stk[maxn], scc[maxn], top, idx, scccnt;
14 void tarjan(int x)
15 {
16     dfn[x] = low[x] = ++idx;
17     stk[top++] = x;
18     for (int i = head[x]; ~i; i = next[i])
19         if (!dfn[to[i]])
20             tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
21         else if (!scc[to[i]])
22             low[x] = min(low[x], dfn[to[i]]);
23     if (dfn[x] == low[x])
24     {
25         scccnt++;
26         do
27             scc[stk[--top]] = scccnt;
28         while (stk[top] != x);
29     }
30 }
31 int main()
32 {
33     int m, n;
34     while (scanf("%d%d", &n, &m) && (m + n))
35     {
36         memset(head, -1, sizeof(head));
37         memset(dfn, 0, sizeof(dfn));
38         memset(low, 0, sizeof(low));
39         memset(scc, 0, sizeof(scc));

```

```

40     idx = top = ecnt = scccnt = 0;
41     int a1, a2;
42     char c1, c2;
43     for (int i = 0; i < m; i++)
44     {
45         scanf("%d%c %d%c", &a1, &c1, &a2, &c2);
46         a1 = a1 << 1 | (c1 == 'h'), a2 = a2 << 1 | (c2 == 'h');
47         addEdge(a1, a2 ^ 1), addEdge(a2, a1 ^ 1);
48     }
49     addEdge(0, 1);
50     for (int i = 0; i < (n << 1); i++)
51         if (!dfn[i]) tarjan(i);
52     bool flag = true;
53     for (int i = 0; i < n && flag; i++)
54         if (scc[i << 1] == scc[i << 1 | 1])
55             flag = false;
56     if (!flag)
57         puts("bad luck");
58     else if (n < 1)
59         putchar('\n');
60     else
61         for (int i = 1; i < n; i++)
62             printf("%d%c%c", i, (scc[i << 1] > scc[i << 1 | 1]) ? 'w' : 'h', " \n"[i
63                 == n - 1]);
64     return 0;
65 }

```

POJ3678 - Katu Puzzle 我什么时候做过这个题?

Katu Puzzle

```

1  #include <cstdio>
2  #include <cstring>
3  inline int min(int a, int b) { return a < b ? a : b; }
4  const int maxn = 10010, maxm = 4000010;
5  int head[maxn], next[maxm], to[maxm], ecnt;
6  inline void addEdge(int f, int t)
7  {
8      next[ecnt] = head[f];
9      head[f] = ecnt;
10     to[ecnt] = t;
11     ecnt++;
12 }
13 int dfn[maxn], low[maxn], stk[maxn], scc[maxn], top, idx, scccnt;
14 void tarjan(int x)
15 {
16     dfn[x] = low[x] = ++idx;
17     stk[top++] = x;
18     for (int i = head[x]; ~i; i = next[i])
19         if (!dfn[to[i]])
20             tarjan(to[i]), low[x] = min(low[x], low[to[i]]);

```

```

21         else if (!scc[to[i]])
22             low[x] = min(low[x], dfn[to[i]]);
23     if (dfn[x] == low[x])
24     {
25         scccnt++;
26         do
27             scc[stk[--top]] = scccnt;
28         while (stk[top] != x);
29     }
30 }
31 int main()
32 {
33     int n, m;
34     while (~scanf("%d%d", &n, &m))
35     {
36         memset(dfn, 0, sizeof(dfn));
37         memset(low, 0, sizeof(low));
38         memset(scc, 0, sizeof(scc));
39         memset(head, -1, sizeof(head));
40         ecnt = top = idx = scccnt = 0;
41         for (int i = 0, u, v, w; i < m; ++i)
42         {
43             char op[5];
44             scanf("%d%d%d%s", &u, &v, &w, op);
45             if (op[0] == 'A')
46                 if (w)
47                 {
48                     addEdge(u << 1, v << 1 | 1), addEdge(v << 1, u << 1 | 1);
49                     addEdge(u << 1, v << 1), addEdge(v << 1 | 1, u << 1 | 1);
50                     addEdge(u << 1 | 1, v << 1 | 1), addEdge(v << 1, u << 1);
51                 }
52             else
53             {
54                 addEdge(u << 1 | 1, v << 1), addEdge(v << 1 | 1, u << 1);
55             }
56             if (op[0] == 'O')
57                 if (w)
58                 {
59                     addEdge(u << 1, v << 1 | 1), addEdge(v << 1, u << 1 | 1);
60                 }
61             else
62             {
63                 addEdge(u << 1, v << 1), addEdge(v << 1 | 1, u << 1 | 1);
64                 addEdge(u << 1 | 1, v << 1 | 1), addEdge(v << 1, u << 1);
65                 addEdge(u << 1 | 1, v << 1), addEdge(v << 1 | 1, u << 1);
66             }
67             if (op[0] == 'X')
68                 if (w)
69                 {
70                     addEdge(u << 1, v << 1 | 1), addEdge(v << 1, u << 1 | 1);
71                     addEdge(u << 1 | 1, v << 1), addEdge(v << 1 | 1, u << 1);
72                 }

```

```

73         else
74         {
75             addEdge(u << 1, v << 1), addEdge(v << 1 | 1, u << 1 | 1);
76             addEdge(u << 1 | 1, v << 1 | 1), addEdge(v << 1, u << 1);
77         }
78     }
79     for (int i = 0; i < (n << 1); i++)
80         if (!dfn[i]) tarjan(i);
81     bool flag = true;
82     for (int i = 0; i < n && flag; i++)
83         if (scc[i << 1] == scc[i << 1 | 1])
84             flag = false;
85     puts(flag ? "YES" : "NO");
86 }
87 return 0;
88 }

```

POJ2749 - Building roads 杀光奶牛问题就会得到解决

Building roads

```

1  #include <cstdio>
2  #include <cstring>
3  inline int abs(int x) { return x >= 0 ? x : -x; }
4  inline int min(int a, int b) { return a < b ? a : b; }
5  const int inf = 0x3f3f3f3f, maxn = 10010, maxm = 1200010;
6  int head[maxn], next[maxm], to[maxm], ecnt, n, A, B;
7  int dfn[maxn], low[maxn], stk[maxn], scc[maxn], top, idx, scccnt;
8  int sx1, sy1, sx2, sy2, sLen, X[maxn], Y[maxn], hate[maxn][2], like[maxn][2],
9  d[maxn];
10 inline void addEdge(int f, int t)
11 {
12     next[ecnt] = head[f];
13     head[f] = ecnt;
14     to[ecnt] = t;
15     ecnt++;
16 }
17 void tarjan(int x)
18 {
19     dfn[x] = low[x] = ++idx;
20     stk[top++] = x;
21     for (int i = head[x]; ~i; i = next[i])
22         if (!dfn[to[i]])
23             tarjan(to[i]), low[x] = min(low[x], low[to[i]]);
24         else if (!scc[to[i]])
25             low[x] = min(low[x], dfn[to[i]]);
26     if (dfn[x] == low[x])
27     {
28         scccnt++;
29         do
30             scc[stk[--top]] = scccnt;
31         while (stk[top] != x);

```

```

32     }
33 }
34 bool check(int x)
35 {
36     memset(dfn, 0, sizeof(dfn));
37     memset(low, 0, sizeof(low));
38     memset(scc, 0, sizeof(scc));
39     memset(head, -1, sizeof(head));
40     ecnt = top = idx = scccnt = 0;
41     for (int i = 1; i <= n; i++)
42         for (int j = i + 1; j <= n; j++)
43         {
44             int l1 = d[i << 1], l2 = d[i << 1 | 1];
45             int r1 = d[j << 1], r2 = d[j << 1 | 1];
46             if (l1 + r1 > x)
47                 addEdge(i << 1, j << 1 | 1), addEdge(j << 1, i << 1 | 1);
48             if (l1 + r2 + sLen > x)
49                 addEdge(i << 1, j << 1), addEdge(j << 1 | 1, i << 1 | 1);
50             if (l2 + r1 + sLen > x)
51                 addEdge(i << 1 | 1, j << 1 | 1), addEdge(j << 1, i << 1);
52             if (l2 + r2 > x)
53                 addEdge(i << 1 | 1, j << 1), addEdge(j << 1 | 1, i << 1);
54         }
55     for (int i = 1, a, b; i <= A; i++)
56     {
57         a = hate[i][0], b = hate[i][1];
58         addEdge(a << 1, b << 1 | 1);
59         addEdge(a << 1 | 1, b << 1);
60         addEdge(b << 1, a << 1 | 1);
61         addEdge(b << 1 | 1, a << 1);
62     }
63     for (int i = 1, a, b; i <= B; i++)
64     {
65         a = like[i][0], b = like[i][1];
66         addEdge(a << 1, b << 1);
67         addEdge(a << 1 | 1, b << 1 | 1);
68         addEdge(b << 1, a << 1);
69         addEdge(b << 1 | 1, a << 1 | 1);
70     }
71     for (int i = 1; i <= (n << 1); i++)
72         if (!dfn[i])
73             tarjan(i);
74     for (int i = 1; i <= n; i++)
75         if (scc[i << 1] == scc[i << 1 | 1])
76             return false;
77     return true;
78 }
79 int main()
80 {
81     scanf("%d%d%d%d%d%d", &n, &A, &B, &sx1, &sy1, &sx2, &sy2);
82     sLen = abs(sx1 - sx2) + abs(sy1 - sy2);
83     for (int i = 1; i <= n; i++)

```

```

84     scanf("%d%d", X + i, Y + i);
85     for (int i = 1; i <= n; i++)
86         d[i << 1] = abs(X[i] - sx1) + abs(Y[i] - sy1),
87         d[i << 1 | 1] = abs(X[i] - sx2) + abs(Y[i] - sy2);
88     for (int i = 1; i <= A; i++)
89         scanf("%d%d", &hate[i][0], &hate[i][1]);
90     for (int i = 1; i <= B; i++)
91         scanf("%d%d", &like[i][0], &like[i][1]);
92     int l = 0, r = 8000000, m, ans = -1;
93     while (l <= r)
94         check(m = (l + r) >> 1) ? r = (ans = m) - 1 : l = m + 1;
95     printf("%d\n", ans);
96     return 0;
97 }

```

1.4 欧拉回路

定义 设 $G = (V, E)$ 是一个图。

欧拉回路 图 G 中经过每条边一次并且仅一次的回路称作欧拉回路。

欧拉路径 图 G 中经过每条边一次并且仅一次的路径称作欧拉路径。

欧拉图 存在欧拉回路的图称为欧拉图。

半欧拉图 存在欧拉路径但不存在欧拉回路的图称为半欧拉图。

性质与定理 以下不加证明的给出一些定理 (因为我懒得抄讲义子)

定理 1 无向图 G 为欧拉图, 当且仅当 G 为连通图且所有顶点的度为偶数。

推论 1 无向图 G 为半欧拉图, 当且仅当 G 为连通图且除了两个顶点的度为奇数之外, 其它所有顶点的度为偶数。

定理 2 有向图 G 为欧拉图, 当且仅当 G 的基图¹连通, 且所有顶点的入度等于出度。

推论 2 有向图 G 为半欧拉图, 当且仅当 G 的基图连通, 且存在顶点 u 的入度比出度大 1、 v 的入度比出度小 1, 其它所有顶点的入度等于出度。

解法 由此可以得到以下求欧拉图 G 的欧拉回路的算法:

1. 在图 G 中任意找一个回路 C 。
2. 将图 G 中属于回路 C 的边删除
3. 在残留图的各极大连通子图中分别寻找欧拉回路。
4. 将各极大连通子图的欧拉回路合并到 C 中得到图 G 的欧拉回路。

¹忽略有向图所有边的方向, 得到的无向图称为该有向图的基图。

该算法的伪代码如下：

```
void dfs(u)
{
    for (edge e : edges[u])
        if (!flag[e])
        {
            flag[e] = true;
            flag[rev(e)] = true; //如果图 G 是有向图则删去本行
            dfs(e.to);
            S.push(v);
        }
}
```

最后依次取出栈 S 每一条边而得到图 G 的欧拉回路 (也就是边出栈序的逆序)。由于该算法执行过程中每条边最多访问两次，因此该算法的时间复杂度为 $O(|E|)$ 。

练习题

UOJ117 - 欧拉回路 混合两个子任务使代码风格变得鬼畜起来。

Building roads

```
1  #include <cstdio>
2  #include <cctype>
3  inline void read(int &x)
4  {
5      int ch = x = 0;
6      while (!isdigit(ch = getchar()));
7      for (; isdigit(ch); ch = getchar()) x = x * 10 + ch - '0';
8  }
9  const int N = 100000 + 10, E = N << 2;
10 int adj[N], to[E], nxt[E], ecnt = 1;
11 int out[N], in[N], t, n, m;
12 bool flag[E];
13 int ans[E], tail;
14 inline void addEdge(int u, int v)
15 {
16     ecnt++;
17     nxt[ecnt] = adj[u];
18     adj[u] = ecnt;
19     to[ecnt] = v;
20 }
21 void dfs(int u)
22 {
23     for (int &i = adj[u]; i; i = nxt[i])
24     {
25         int c = t == 1 ? i >> 1 : i - 1;
26         bool sig = i & 1;
27         if (!flag[c])
28         {
```

```

29         flag[c] = true;
30         dfs(to[i]);
31         ans[tail++] = (t == 1 && sig) ? -c : c;
32     }
33 }
34 }
35 int main()
36 {
37     read(t), read(n), read(m);
38     for (int i = 0, u, v; i < m; i++)
39     {
40         read(u), read(v);
41         addEdge(u, v);
42         out[u]++, in[v]++;
43         if (t == 1) addEdge(v, u);
44     }
45     for (int i = 1; i <= n; i++)
46         if (t == 1 ? ((in[i] + out[i]) & 1) : (in[i] != out[i]))
47             return puts("NO"), 0;
48     for (int i = 1; i <= n; i++)
49         if (adj[i])
50         {
51             dfs(i);
52             break;
53         }
54     if (tail != m) return puts("NO"), 0;
55     puts("YES");
56     for (int i = m - 1; i >= 0; i--) printf("%d ", ans[i]);
57     return 0;
58 }

```

2 day2 字符串（一）

2.1 KMP

算法介绍 用来在线性时间内匹配字符串

算法流程 我觉得錄錄錄在 WC 上讲的比较清楚，于是我开始抄讲义。

字符串： $s[1 \dots n]$, $|s| = n$ 。

子串： $s[i \dots j] = s[i]s[i+1] \dots [j]$ 。

前缀： $pre(s, x) = s[1 \dots x]$, **后缀：** $suf(s, x) = s[n-x+1 \dots n]$

若 $0 \leq r \leq |s|$, $pre(s, r) = suf(s, r)$, 就称 $pre(s, r)$ 是 s 的 **border**。

KMP 算法的第一步主要做这么一件事：在 $O(n)$ 时间求出数组 $next[1 \dots n]$, 其中 $next[i]$ 表示前缀 $s[1 \dots i]$ 的最大 border 长度。于是可以知道 s 的所有 border 长度为 $next[n], next[next[n]], \dots$, 我想这是显然的，于是不加证明的在这里给出。

第二步就是匹配，如果失配了就把模式串的当前位置指针 i 跳到 $next[i]$ 处然后继续匹配，然后就好了。

算法实现

genNext

```

1 for (int i = 1, j = -1; i < len; i++)
2 {
3     while (~j && str[j + 1] != str[i]) j = next[j];
4     if (str[j + 1] == str[i]) j++;
5     next[i] = j;
6 }

```

Find

```

1 for (int i = 0, j = -1; i < len; i++)
2 {
3     while (~j && t[j + 1] != s[i]) j = next[j];
4     if (t[j + 1] == s[i]) j++;
5     if (j == len - 1) ans++, j = next[j];
6 }

```

练习题

POJ3461 - Oulipo 求出所有匹配位置

Oulipo

```

1 #include <stdio>
2 #include <string>
3 char a[1 << 20 | 1], b[1 << 14 | 1];
4 int la, lb;
5 int next[1 << 14 | 1];
6 int main()
7 {
8     next[0] = -1;
9     int n;
10    scanf("%d", &n);
11    while (n——)
12    {
13        scanf("%s%s", b, a);
14        la = strlen(a), lb = strlen(b);
15        for (int i = 1, j = -1; i < lb; i++)
16        {
17            while (~j && b[j + 1] != b[i]) j = next[j];
18            if (b[j + 1] == b[i]) j++;
19            next[i] = j;
20        }
21        int ans = 0;
22        for (int i = 0, j = -1; i < la; i++)
23        {
24            while (~j && b[j + 1] != a[i]) j = next[j];
25            if (b[j + 1] == a[i]) j++;
26            if (j == lb - 1) ans++, j = next[j];
27        }
28        printf("%d\n", ans);
29    }

```

```

30     return 0;
31 }

```

POJ2406 - Power Strings *next* 数组的奇妙性质

Power Strings

```

1  #include <stdio>
2  #include <string>
3  char str[1 << 20 | 1];
4  int next[1 << 20 | 1];
5  int len;
6  int main()
7  {
8      next[0] = -1;
9      while (scanf("%s", str))
10     {
11         if (str[0] == '.') break;
12         len = strlen(str);
13         for (int i = 0, j = -1; i < len; i++)
14             (~j && str[i] != str[j]) ? j = next[j] : next[++i] = ++j;
15         printf("%d\n", len % (len - next[len]) == 0 ? len / (len - next[len]) : 1);
16     }
17     return 0;
18 }

```

CF526D - Om Nom and Necklace 啥?

Om Nom and Necklace

```

1  #include <stdio>
2  #include <string>
3  char s[1000010];
4  int next[1000010], n, k, len;
5  int main()
6  {
7      scanf("%d%d%s", &n, &k, s);
8      next[0] = -1;
9      len = strlen(s);
10     for (int i = 1; i < len; ++i)
11     {
12         int j;
13         for (j = next[i - 1]; j != -1 && s[j + 1] != s[i]; j = next[j]);
14         if (s[j + 1] == s[i]) j++;
15         next[i] = j;
16     }
17     for (int i = 0; i < len; ++i)
18     {
19         int p = i + 1, q = p / (i - next[i]);
20         putchar(((p % (i - next[i]) == 0) ? (q / k >= q % k ? '1' : '0') : (q / k > q % k
21             ? '1' : '0'))));
22     }
23     return 0;

```

23 | }

讲道理我 KMP 真的学的不是很明白，望各位 dalao 给予指导。

2.2 Trie

简介 字典树，也称 Trie、字母树，指的是某个字符串集合对应的形如下图的有根树。树的每条边上对应恰好一个字符，每个顶点代表从根到该节点的路径所对应的字符串（将所有经过的边上的字符按顺序连接起来）。

实现 水

Trie - impl

```

1 struct node
2 {
3     node *trans[26];
4     int cnt;
5 };
6 void insert(node *n, char *str)
7 {
8     for (; *str; n = n->trans[*str - '0'])
9         if (n->trans[*str - '0'] == 0)
10             n->trans[*str - '0'] = new_node();
11     n->cnt++;
12 }
```

练习题

POJ3630 - Phone List 若插入过程中，有某个经过的节点带有串结尾标记，则之前插入的某个串是当前串的前缀。

Oulipo

```

1 #include <cstdio>
2 #include <cstring>
3 struct node
4 {
5     node *trans[10];
6     bool is_end;
7 } nodes[100010];
8 node *root;
9 int cnt;
10 node *new_node() { return &nodes[cnt++]; }
11 char buf[11];
12 bool try_insert(node *n, char *str)
13 {
14     if (n->is_end) return false;
15     if (*str == '\0')
16     {
17         for (int i = 0; i < 10; i++)
18             if (n->trans[i])
19                 return false;
```

```

20     n->is_end = true;
21     return true;
22 }
23 if (n->trans[*str - '0'] == 0) n->trans[*str - '0'] = new_node();
24 return try_insert(n->trans[*str - '0'], str + 1);
25 }
26 int main()
27 {
28     int t, n;
29     scanf("%d", &t);
30     while (t--)
31     {
32         memset(nodes, 0, sizeof(nodes));
33         cnt = 0;
34         root = new_node();
35         scanf("%d", &n);
36         bool flag = true;
37         while (n--)
38         {
39             scanf("%s", buf);
40             if (flag) flag = try_insert(root, buf);
41         }
42         puts(flag ? "YES" : "NO");
43     }
44     return 0;
45 }

```

POJ2945 - Find the Clones n 个基因片段, 每个长度为 m , 输出 n 行表示重复出现 i 次 ($1 \leq i \leq n$) 的基因片段的个数

Find the Clones

```

1  #include <stdio>
2  #include <string>
3  struct node
4  {
5      node *trans[4];
6      int cnt;
7  } nodes[400010];
8  int tot;
9  node *root;
10 inline node *new_node() { return &nodes[tot++]; }
11 void try_insert(node *n, char *str)
12 {
13     if (*str == '\0')
14         n->cnt++;
15     else
16     {
17         if (n->trans[*str - '0'] == 0) n->trans[*str - '0'] = new_node();
18         try_insert(n->trans[*str - '0'], str + 1);
19     }
20 }

```

```

21 char f[1 << 8 | 1];
22 int ans[20010];
23 int main()
24 {
25     f['A'] = '0', f['C'] = '1', f['G'] = '2', f['T'] = '3';
26     int n, m;
27     char buf[22];
28     while (~scanf("%d%d", &n, &m) && (n + m))
29     {
30         memset(ans, 0, sizeof(ans));
31         memset(nodes, 0, sizeof(nodes));
32         tot = 0;
33         root = new_node();
34         for (int i = 0; i < n; i++)
35         {
36             scanf("%s", buf);
37             for (int j = 0; j < m; j++)
38                 buf[j] = f[buf[j]];
39             try_insert(root, buf);
40         }
41         for (int i = 0; i < tot; i++) ans[nodes[i].cnt]++;
42         for (int i = 1; i <= n; i++)
43             printf("%d\n", ans[i]);
44     }
45     return 0;
46 }

```

2.3 Aho-Corasick Automaton

简介 多模式串字符串匹配，Trie 上的 KMP，其中 *next* 数组变成了 *fail* 指针，功能相同。

实现 Trie 的实现上文已经出现，所以此处不再重复。

buildFail

```

1 void buildFail()
2 {
3     int h = 0, t = 0;
4     root->fail = &virt;
5     que[t++] = root;
6     while (h < t)
7     {
8         node *cur = que[h++];
9         for (int i = 0; i < 26; i++)
10        {
11            node *f = cur->fail;
12            while (f->trans[i] == 0) f = f->fail;
13            f = f->trans[i];
14            if (cur->trans[i])
15                (que[t++] = cur->trans[i])->fail = f;
16            else
17                cur->trans[i] = f;

```

```

18     }
19 }
20 }

```

练习题

HDU2222 - Keywords Search AC 自动机模板题，注意统计答案时，每个节点只能统计一次不要重复统计。

Keywords Search

```

1  #include <stdio>
2  #include <cstring>
3  struct node
4  {
5      node *trans[26], *fail;
6      int cnt;
7  } nodes[500010], virt;
8  node *que[500010];
9  int tot;
10 node *root;
11 node *new_node() { return &nodes[tot++]; }
12 void insert(char *str)
13 {
14     node *cur = root;
15     for (; *str; str++)
16     {
17         if (cur->trans[*str - 'a'] == 0)
18             cur->trans[*str - 'a'] = new_node();
19         cur = cur->trans[*str - 'a'];
20     }
21     cur->cnt++;
22 }
23 void buildFail()
24 {
25     int h = 0, t = 0;
26     root->fail = &virt;
27     que[t++] = root;
28     while (h < t)
29     {
30         node *cur = que[h++];
31         for (int i = 0; i < 26; i++)
32         {
33             node *f = cur->fail;
34             while (f->trans[i] == 0) f = f->fail;
35             f = f->trans[i];
36             if (cur->trans[i])
37                 (que[t++] = cur->trans[i])->fail = f;
38             else
39                 cur->trans[i] = f;
40         }
41     }

```



```

42 }
43 char buf[1000010];
44 int vis[500010];
45 int main()
46 {
47     memset(vis, -1, sizeof(vis));
48     int T, n;
49     scanf("%d", &T);
50     while (T--)
51     {
52         memset(nodes, 0, sizeof(nodes));
53         tot = 0, root = new_node();
54         for (int i = 0; i < 26; i++) virt.trans[i] = root;
55         scanf("%d", &n);
56         for (int i = 0; i < n; i++)
57         {
58             scanf("%s", buf);
59             insert(buf);
60         }
61         buildFail();
62         scanf("%s", buf);
63         node *cur = root, *tmp;
64         int ans = 0;
65         for (char *ch = buf; *ch; ch++)
66         {
67             tmp = cur = cur->trans[*ch - 'a'];
68             while (tmp != &virt && vis[tmp - nodes] != T)
69             {
70                 vis[tmp - nodes] = T;
71                 ans += tmp->cnt;
72                 tmp = tmp->fail;
73             }
74         }
75         printf("%d\n", ans);
76     }
77     return 0;
78 }

```

2.4 Manacher

求出字符串每一位的回文半径，算法流程奥妙重重，不易让常人理解，然后我就抄了份板子改了改，然后就比较讲义上的标程快了 20%。

练习题

POJ3974 - Palindrome Manacher 模板题

Palindrome

```

1 #include <stdio>
2 #include <cstring>
3 inline int min(int a, int b) { return a < b ? a : b; }

```

```

4 inline int max(int a, int b) { return a > b ? a : b; }
5 char buf[1000100], str[2000200];
6 int R[2000200], T;
7 int main()
8 {
9     while (scanf("%s", buf), buf[0] != 'E')
10    {
11        int m = int(strlen(buf)), n = 0;
12        str[n++] = '!';
13        str[n++] = '#';
14        for (int i = 0; i < m; i++)
15            str[n++] = buf[i], str[n++] = '#';
16        str[n++] = '#';
17        str[n++] = '?';
18        int p = 0, mx = 0, ans = 0;
19        for (int i = 1; i < n; i++)
20        {
21            R[i] = mx > i ? min(R[2 * p - i], mx - i) : 1;
22            while (str[i + R[i]] == str[i - R[i]]) R[i]++;
23            if (R[i] + i > mx)
24                mx = i + R[p = i];
25            ans = max(ans, R[i]);
26        }
27        printf("Case %d: %d\n", ++T, ans - 1);
28    }
29    return 0;
30 }

```

3 day3 简单数学

说是简单数学其实我后半部分也没看懂

3.1 整除及剩余

整除定义 设 a, b 是两个整数, 且 $b \neq 0$. 如果存在整数 c , 使得 $a = bc$, 则称 a 被 b 整除, 或 b 整除 a , 记作 $b|a$. 此时, 又称 a 是 b 的倍数, b 是 a 的因子。

整除的基本性质

1. $a|b \wedge a|c \Rightarrow a|(b + c)$
2. $a|b \Rightarrow \forall c \in \mathbb{Z}, a|bc$
3. $a|b \wedge b|c \Rightarrow a|c$

同余基本定义和定理

定义 1: 带余除法

$$\forall a \in \mathbb{Z}, b \in \mathbb{Z}^* \rightarrow \exists q, r \in \mathbb{Z}, a = qb + r, r \in [0, |b|)$$

定义 2: 同余

$$a \bmod m = b \bmod m \iff a \equiv b \pmod{m}$$

定义 3: 剩余类

$$\mathbf{A}_i = \{x | x \in \mathbb{Z} \wedge x \bmod m = i\} \rightarrow \forall a, b \in \mathbf{A}_i, a \equiv b \pmod{m}$$

定义 4: 完全系

$$\{a_1 \bmod m, a_2 \bmod m, \dots, a_n \bmod m\} = \{0, 1, 2, \dots, m-1\}$$

定理 1

$$a \equiv b \pmod{m} \iff \exists k \in \mathbb{Z}, a = b + km \iff m | (a - b)$$

定理 2 同余关系是等价关系

1. $a \equiv a \pmod{m}$
2. $a \equiv b \pmod{m} \Rightarrow b \equiv a \pmod{m}$
3. $a \equiv b \pmod{m} \wedge b \equiv c \pmod{m} \Rightarrow a \equiv c \pmod{m}$

定理 3 同余的三则运算

$$a, b, c \in \mathbb{R}, m \in \mathbb{N}^*, a \equiv b \pmod{m} \Rightarrow$$

1. $a + c \equiv b + c \pmod{m}$
2. $a - c \equiv b - c \pmod{m}$
3. $ac \equiv bc \pmod{m}$

定理 4 同余式的三则运算

$$a, b, c \in \mathbb{R}, m \in \mathbb{N}^*, a \equiv b \pmod{m} \wedge c \equiv d \pmod{m} \Rightarrow$$

1. $ax + cy \equiv bx + dy \pmod{m} \quad x, y \in \mathbb{Z}$
2. $ac \equiv bd \pmod{m}$
3. $a^n \equiv b^n \pmod{m} \quad n \in \mathbb{N}^*$
4. $f(a) \equiv f(b) \pmod{m} \quad f(x) \text{ 为任一整系数多项式}$

定理 5

1. $a \equiv b \pmod{m} \wedge d | m \Rightarrow a \equiv b \pmod{d}$
2. $a \equiv b \pmod{m} \Rightarrow \gcd(a, m) = \gcd(b, m)$
3. $\forall i \in [1, n], a \equiv b \pmod{m_i} \iff a \equiv b \pmod{\text{lcm}(m_1, m_2, \dots, m_n)}$

3.2 素数

定义 素数（质数）是大于 1 的正整数，并且除了 1 和它本身不能被其他正整数整除。大于 1 的非素数的正整数称为合数。

分布 素数有无穷多个. 如果使用 $\pi(x)$ 表示小于一个正实数 x 的素数有多少个, 那么有

$$\lim_{x \rightarrow \infty} \frac{\pi(x)}{x} = \ln x$$

算术基本定理/唯一分解定理

$$n = \prod_{i=1}^{\infty} p_i^{a_i} \quad p_i \in \mathbb{P}, a_i \in \mathbb{N}$$

判定

$$n \in \mathbb{P} \iff \forall i \in [2, \sqrt{n}], i \nmid n$$

Eratosthenes 筛法

Eratosthenes Sieve

```

1 fill(isprime, true);
2 for (int i = 2; i < n; i++)
3     if (isprime[i])
4         for (int j = i * i; j < n; j += i)
5             isprime[j] = false;

```

欧拉函数 欧拉函数 $\varphi(n)$ 指不超过 n 且与 n 互素的正整数的个数, 其中 n 是一个正整数。

$$\text{欧拉函数的性质} \quad n = \prod_{i=1}^m p_i^{a_i} \rightarrow \varphi(n) = \prod_{i=1}^m \varphi(p_i^{a_i})$$

$$\text{定理 1} \quad p \in \mathbb{P} \iff \varphi(p) = p - 1$$

$$\text{定理 2} \quad p \in \mathbb{P}, a \in \mathbb{N}_+ \Rightarrow \varphi(p^a) = p^a - p^{a-1}$$

$$\text{定理 3} \quad m, n \in \mathbb{N}_+ \wedge \gcd(m, n) = 1 \Rightarrow \varphi(mn) = \varphi(m)\varphi(n)$$

$$\text{定理 4} \quad n = \prod_{i=1}^m p_i^{a_i} \rightarrow \varphi(n) = n \prod_{i=1}^m \left(1 - \frac{1}{p_i}\right)$$

$$\text{推论} \quad n \equiv 1 \pmod{2} \rightarrow \varphi(2n) = \varphi(n)$$

$$\text{定理 5} \quad n \in (2, +\infty) \cap \mathbb{Z} \Rightarrow \varphi(n) \equiv 0 \pmod{2}$$

$$\text{定理 6} \quad n \in \mathbb{N}_+ \Rightarrow \sum_{d|n} \varphi(d) = n$$

$$\text{欧拉定理} \quad \gcd(a, m) = 1, a \in \mathbb{N}_+, m \in [2, +\infty) \cap \mathbb{Z} \Rightarrow a^{\varphi(m)} \equiv 1 \pmod{m}$$

$$\text{费马小定理} \quad m \in \mathbb{P} \Rightarrow a^{m-1} \equiv 1 \pmod{m}$$

练习题

POJ2689 - Prime Distance 暴力筛掉合数

Prime Distance

```

1  #include <cmath>
2  #include <cstdio>
3  #include <cstring>
4  int prime_count;
5  int prime[5140];
6  bool f[1000010];
7  bool notprime[50010];
8  int main()
9  {
10     for (int i = 2; i < 50010; i++)
11         if (!notprime[i])
12             for (long long j = 1ll * i * i; j < 50010; j += i)
13                 notprime[j] = true;
14     for (int i = 2; i < 50010; i++)
15         if (!notprime[i])
16             prime[prime_count++] = i;
17     int l, r;
18     while (~scanf("%d%d", &l, &r))
19     {
20         l = l == 1 ? 2 : l;
21         memset(f, 0, sizeof(f));
22         for (int i = 0, a, b; i < prime_count; i++)
23         {
24             a = (l - 1) / prime[i] + 1;
25             b = r / prime[i];
26             for (int j = a; j <= b; j++)
27                 if (j > 1)
28                     f[j * prime[i] - 1] = true;
29         }
30         int mx = -1, mn = 0x3f3f3f3f, x1 = 0, x2 = 0, y1 = 0, y2 = 0;
31         for (int i = 0, p = -1; i <= r - l; i++)
32             if (!f[i])
33             {
34                 if (~p)
35                 {
36                     if (mx < i - p)
37                         mx = i - p, x1 = p + 1, y1 = i + 1;
38                     if (mn > i - p)
39                         mn = i - p, x2 = p + 1, y2 = i + 1;
40                 }
41                 p = i;
42             }
43         if (mx == -1)
44             puts("There are no adjacent primes.");
45         else
46             printf("%d,%d are closest, %d,%d are most distant.\n", x2, y2, x1, y1);
47     }
48     return 0;
49 }

```

POJ3421 - X-factor Chains 质因子的排列组合

X-factor Chains

```

1  #include <stdio>
2  int f[1 << 12 | 1];
3  long long _f(int x)
4  {
5      long long ans = 1;
6      for (int i = x; i; i--) ans *= i;
7      return ans;
8  }
9  int main()
10 {
11     int x;
12     while (~scanf("%d", &x))
13     {
14         int _x = x, t = 0;
15         for (int i = 2; i * i <= _x; i++)
16         {
17             f[t] = 0;
18             while (_x % i == 0)
19                 _x /= i, f[t]++;
20             t++;
21         }
22         if (_x != 1) f[t++] = 1;
23         int sum = 0;
24         for (int i = 0; i < t; i++) sum += f[i];
25         long long fac = _f(sum);
26         for (int i = 0; i < t; i++) fac /= _f(f[i]);
27         printf("%d %lld\n", sum, fac);
28     }
29     return 0;
30 }

```

POJ3090 - Visible Lattice Points 欧拉函数

Visible Lattice Points

```

1  #include <stdio>
2  const int maxn = 1010;
3  int phi[maxn], sum[maxn];
4  int main()
5  {
6      phi[1] = 1;
7      for (int i = 2; i <= 1005; i++) if (!phi[i])
8          for (int j = i; j <= 1005; j += i)
9              {
10                 if (!phi[j]) phi[j] = j;
11                 phi[j] = phi[j] / i * (i - 1);
12             }
13     for (int i = 1; i <= 1005; i++) sum[i] = sum[i - 1] + phi[i];
14     int T, x;

```

```

15     scanf("%d", &T);
16     for (int i = 1; i <= T; i++)
17     {
18         scanf("%d", &x);
19         printf("%d %d %d\n", i, x, sum[x] << 1 | 1);
20     }
21     return 0;
22 }

```

3.3 欧几里得算法

最大公约数与最小公倍数

定义 1 设 a 和 b 是两个整数, 如果 $d|a$ 且 $d|b$, 则称 d 是 a 与 b 的公因子

定义 2 设 a 和 b 是两个不全为 0 的整数, 称 a 与 b 的公因子中最大的为 a 与 b 的最大公因子, 或最大公约数, 记作 $\gcd(a, b)$

定义 3 设 a 和 b 是两个非零整数, 称 a 与 b 最小的正公倍数为 a 与 b 的最小公倍数, 记作 $\text{lcm}(a, b)$

最大公约数与最小公倍数的性质

1. $a|m \wedge b|m \Rightarrow \text{lcm}(a, b)|m$
2. $d|a \wedge d|b \Rightarrow d|\gcd(a, b)$
3. $\text{lcm}(a, b) = \frac{ab}{\gcd(a, b)}$
4. $m, a, b \in \mathbb{N}_+ \rightarrow \text{lcm}(ma, mb) = m \times \text{lcm}(a, b), \gcd(ma, mb) = m \times \gcd(a, b)$

计算方法

素因子分解法 令

$$a = \prod_{i=1}^m p_i^{r_i}, \quad b = \prod_{i=1}^m p_i^{s_i}$$

于是

$$\gcd(a, b) = \prod_{i=1}^m p_i^{\min(r_i, s_i)}, \quad \text{lcm}(a, b) = \prod_{i=1}^m p_i^{\max(r_i, s_i)}$$

欧几里得算法 1

$$\gcd(a, b) = \gcd(b, a \bmod b)$$

欧几里得算法 2

$$\gcd(a, b) = \gcd(a, a - b)$$

拓展欧几里得算法 不理解就记下来

exgcd

```

1 void exgcd(int64 a, int64 b, int64 &x, int64 &y)
2 { b == 0 ? (x = 1, y = 0) : (exgcd(b, a % b, y, x), y -= x * (a / b)); }

```

3.4 线性同余方程

二元一次不定方程

定义 1 $a, b, c \in \mathbb{Z}, a \neq 0, b \neq 0$, 那形如 $ax + by = c$ 的方程称为二元一次不定方程。

定理 1 设 $a, b \in \mathbb{Z}$ 且 $d = \gcd(a, b)$, 如果 $d|c$, 那么方程存在无穷多个整数解, 否则方程不存在整数解。

定理 2 如果不定方程有解且特解为 $x = x_0, y = y_0$ 那么方程的解可以表示为

$$x = x_0 + \frac{b}{d}t, y = y_0 - \frac{a}{d}t, \text{ 其中 } t \in \mathbb{Z}$$

同余方程与不定方程 在 $a > 0$ 且 $b > 0$ 的条件下, 求二元一次方程 $ax + by = c$ 的整数解等价于求一元线性同余方程 $ax \equiv c \pmod{b}$ 的整数解

求一元线性同余方程 要求 $ax \equiv c \pmod{b}$, 即为求 $ax + my = b$ 的解。记 $d = \gcd(a, m)$, 先使用拓展欧几里得求 $ax + my = b$, 如果 $d \nmid b$ 则无解, 否则 \pmod{m} 意义下的解有 d 个, 可以通过对其中某个解不断地加 $\frac{m}{d}$ 得到。(这 d 个解的形式为 $x_0 + \frac{m}{d}t, t \in \mathbb{Z}$, 其中 x_0 是已知的一个解)

中国剩余定理 若 $m_1, m_2, m_3, \dots, m_r$ 是两两互素的正整数, 则同余方程组

$$\begin{cases} x \equiv a_1 \pmod{m_1} \\ x \equiv a_2 \pmod{m_2} \\ x \equiv a_3 \pmod{m_3} \\ \dots \\ x \equiv a_r \pmod{m_r} \end{cases} \quad (3.1)$$

有 $\pmod{M = \prod_{i=1}^r m_i}$ 的唯一解, 即为中国剩余定理。

若 $n = pq$ 且 $\gcd(p, q) = 1$, 那么 $x \pmod{p}, x \pmod{q}$ 的值确定后, $x \pmod{n}$ 的值也会随之确定。

算法说明

$$M_i = \prod_{j \neq i} m_j \rightarrow \gcd(M_i, m_i) = 1 \Rightarrow \exists p_i, q_i, M_i p_i + m_i q_i = 1$$

$$(e_i = M_i p_i) \equiv \text{int}(j == i) \pmod{m_i} \rightarrow \sum_1^r e_i a_i \pmod{\sum_1^r m_i} \text{ 是方程的最小非负整数解}$$

练习题

POJ1061 - 蛤蛤的约会 +1s

蛤蛤的约会

```

1 #include <stdio>
2 typedef long long int64;
3 int64 gcd(int64 a, int64 b) { return b == 0 ? a : gcd(b, a % b); }
4 void exgcd(int64 a, int64 b, int64 &x, int64 &y) { b == 0 ? (x = 1, y = 0) : (exgcd(b, a
    % b, y, x), y -= x * (a / b)); }
```



```

5 int main()
6 {
7     int64 s, t, p, q, L;
8     scanf("%lld%lld%lld%lld%lld", &s, &t, &p, &q, &L);
9     int64 a = (p - q + L) % L, b = L, c = (t - s + L) % L;
10    int64 x = 0, y = 0, g = gcd(a, b);
11    if (c % g)
12        puts("Impossible");
13    else
14    {
15        a /= g, b /= g, c /= g;
16        exgcd(a, b, x, y);
17        printf("%lld", ((x % b + b) % b) * c % b);
18    }
19    return 0;
20 }

```

POJ2142 - The Balance 求 $ax + by = c$ 的一组解, 使得 $|x| + |y|$ 尽量小, 在前者尽量小时 $|ax| + |by|$ 尽量小

The Balance

```

1 #include <cstdio>
2 inline int abs(int x) { return x >= 0 ? x : -x; }
3 void exgcd(int a, int b, int &d, int &x, int &y) { !b ? (x = 1, y = 0, d = a) : (exgcd(b,
4     a % b, d, y, x), y -= x * (a / b)); }
5 int main()
6 {
7     int a, b, c, x, y, g, u1, v1, u2, v2;
8     while (~scanf("%d%d%d", &a, &b, &c) && a + b + c)
9     {
10        exgcd(a, b, g, x, y);
11        a /= g, b /= g, c /= g;
12        u1 = (x % b * c % b + b) % b;
13        v1 = abs((c - u1 * a) / b);
14        v2 = (y % a * c % a + a) % a;
15        u2 = abs((c - v2 * b) / a);
16        if (u1 + v1 > u2 + v2 || (u1 + v1 == u2 + v2 && a * u1 + b * v1 > a * u2 + b * v2))
17            u1 = u2, v1 = v2;
18        printf("%d %d\n", u1, v1);
19    }
20 }

```

3.5 逆元

解一元线性同余方程

$$\gcd(a, m) = 1 \Rightarrow \exists x, ax \equiv 1 \pmod{m}$$

$$ax \equiv 1 \pmod{m} \iff \exists k \in \mathbb{Z}, ax - km = 1$$

```

1 int inv(int a, int m)
2 {
3     int x, y;
4     exgcd(a, m, x, y);
5     return (x % m + m) % m;
6 }

```

费马小定理

$$\forall p \in \mathbb{P} \rightarrow x^p \equiv x \pmod{p}$$

被称为费马小定理，若 $p \nmid x$ ，有

$$x^{p-1} \equiv 1 \pmod{p}$$

于是有

$$\forall p \in \mathbb{P}, x \in \mathbb{Z} \rightarrow x^{-1} \equiv x^{p-2} \pmod{p}$$

使用快速幂即可计算。

3.6 离散对数问题

解这鬼东西：

$$A^x \equiv B \pmod{C}$$

这玩意有个性， $A^x \bmod C$ 有周期性，最大周期不超过 C ，我想这是显然的（除非你没上过小学）。

$C \in \mathbb{P}$ 普通的 BSGS

POJ2417 - Discrete Logging 模板题

Primitive Roots

```

1 #include <cmath>
2 #include <cstdio>
3 #include <cstring>
4 typedef long long i64;
5 void exgcd(i64 a, i64 b, i64 &x, i64 &y)
6 {
7     b == 0 ? (x = 1, y = 0) : (exgcd(b, a % b, y, x), y -= (a / b) * x);
8 }
9 struct HashTable
10 {
11     static const size_t sz = 500009;
12     i64 idx[sz], val[sz];
13     void init()
14     {
15         memset(idx, -1, sizeof(idx));
16         memset(val, -1, sizeof(val));
17     }
18     void insert(i64 i, i64 v)
19     {
20         i64 j = i % sz;

```

```

21     while (idx[j] != -1 && idx[j] != i)
22     {
23         j++;
24         if (j == sz)
25             j = 0;
26     }
27     if (val[j] == -1)
28     {
29         idx[j] = i;
30         val[j] = v;
31     }
32 }
33 i64 find(i64 i)
34 {
35     i64 j = i % sz;
36     while (idx[j] != -1 && idx[j] != i)
37     {
38         j++;
39         if (j == sz)
40             j = 0;
41     }
42     return val[j];
43 }
44 } H;
45 int main()
46 {
47     for (i64 a, b, c; ~scanf("%lld%lld%lld", &c, &a, &b) && a | b | c;)
48     {
49         H.init();
50         i64 m = i64(ceil(sqrt(c))), d = 1;
51         for (int i = 0; i < m; i++, d = d * a % c)
52             H.insert(d, i);
53         i64 res = 1, x, y;
54         bool flag = false;
55         for (i64 i = 0; i < m && !flag; i++, res = res * d % c)
56         {
57             exgcd(res, c, x, y);
58             x = (x * b % c + c) % c;
59             i64 j = H.find(x);
60             if (j != -1)
61                 printf("%lld\n", i * m + j), flag = true;
62         }
63         if (!flag)
64             puts("no solution");
65     }
66     return 0;
67 }

```

$C \in \mathbb{N}_+$ 待续


```

34 |     return 0;
35 | }

```

练习题

POJ1284 - Primitive Roots 如果 $n \in \mathbb{N}_+$ 有一个原根，那么 n 一共有 $\varphi(\varphi(n))$ 个不同余的原根

Primitive Roots

```

1 | #include <stdio>
2 | const int N = 1 << 16 | 1;
3 | int phi[N];
4 | int main()
5 | {
6 |     for (int i = 2; i < N; i++) if (!phi[i])
7 |         for (int j = i; j < N; j += i)
8 |             {
9 |                 if (!phi[j]) phi[j] = j;
10 |                 phi[j] = phi[j] / i * (i - 1);
11 |             }
12 |     for (int p; ~scanf("%d", &p);)
13 |         printf("%d\n", phi[p - 1]);
14 |     return 0;
15 | }

```