

等差子序列(COGS 3947)

- 如果存在 $k > 3$ 的等差序列，那么一定有 $k = 3$ 的等差序列，所以只需要判断是否存在 $k = 3$ 的等差序列，即是否存在 $p < q < r$ ，使得

$$a_p - a_q = a_q - a_r。$$

算法1

- 枚举 p, q, r ，判断是否满足 $a_p - a_q = a_q - a_r$ 。
- 时间复杂度： $O(TN^3)$ 。

```
1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e4 + 10;
5  int a[N];
6  int n;
7
8  int main()
9  {
10     int T; scanf("%d", &T);
11     while(T--)
12     {
13         scanf("%d", &n);
14         for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
15         bool flag = false;
16         for(int p = 1; p <= n; ++p)
17             for(int q = p + 1; q <= n; ++q)
18                 for(int r = q + 1; r <= n; ++r)
19                     if(a[p] - a[q] == a[q] - a[r])
20                         flag = true;
21         if(flag) puts("Y");
22         else puts("N");
23     }
24     return 0;
25 }
```

算法2

- 枚举中间位置 q ，根据条件 $a_p - a_q = a_q - a_r$ ，可以先枚举 $p(< q)$ ，求出所有的 $a_p - a_q$ 的值，然后枚举 $r(> q)$ ，判断 $a_q - a_r$ 是否与存储的值重复。
- 可以利用 `set` 或 Hash 来处理判重问题。
- 时间复杂度： $O(TN^2 \log N)$ ，`set` 判重； $O(TN^2)$ ，Hash 判重。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e4 + 10;
5  int a[N], h[N * 2];
6  int n;
7
8  int main()
9  {
10     int T; scanf("%d", &T);
11     while(T--)
12     {
13         scanf("%d", &n);
14         for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
15         bool flag = false;
16         for(int q = 2; q <= n - 1; ++q)
17         {
18             for(int i = 1; i <= 2 * n; ++i) h[i] = false;
19             for(int p = 1; p <= q - 1; ++p) h[a[p] - a[q] + n] =
true;
20             for(int r = q + 1; r <= n; ++r)
21                 if(h[a[q] - a[r] + n]) flag = true;
22             if(flag) break;
23         }
24         if(flag) puts("Y");
25         else puts("N");
26     }
27     return 0;
28 }

```

算法3

- 上述两个算法都可以针对一般序列，但是本题的序列是一个 $1 \sim n$ 的排列。
- 构成等差序列的子序列(长度为3)集合其实是已知的，例如序列1, 2, 3, 4, 5, 6中，如果中间数是3，那么能构成等差序列的只有(1, 3, 5), (5, 3, 1), (2, 3, 4), (4, 3, 2)。
- 那么当枚举到中间数 a_q 时，如果和它组成等差序列的的数一个在左一个在右，就有等差序列。
- 定义 b_i 表示数字 i 是否已经扫描过(在 a_q 的左侧)，如果 $a_q - k$ 和 $a_q + k$ 的扫描状态均为1，则不构成等差序列，反之则构成等差序列。
- 那么问题就变成了判断以 a_q 为中心，两端不超过边界的01串是否是**回文串**。
 - 序列是1, 5, 3, 2, 4, 6，那么当扫描到3时， b 数组为1, 0, 1, 0, 1, 0， $b_{1 \sim 5}$ 是回文。
 - 序列是1, 6, 3, 5, 4, 2，那么当扫描到3时， b 数组为1, 0, 1, 0, 0, 1， $b_{1 \sim 5}$ 不是回文。

- 时间复杂度: $O(TN^2)$, 但是常数比算法2小。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e4 + 10;
5  int a[N], b[N];
6  int n;
7
8  int main()
9  {
10     int T; scanf("%d", &T);
11     while(T--)
12     {
13         memset(b, 0, sizeof(b));
14         scanf("%d", &n);
15         for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
16         bool flag = false;
17         for(int i = 1; i <= n; ++i)
18         {
19             b[a[i]] = 1;
20             if(a[i] == 1 || a[i] == n) continue;
21             int d = min(a[i] - 1, n - a[i]); // 边界
22             for(int j = a[i] - d; j <= a[i] - 1; ++j)
23                 if(b[j] != b[2 * a[i] - j]) flag = true;
24             if(flag) break;
25         }
26         if(flag) puts("Y");
27         else puts("N");
28     }
29     return 0;
30 }

```

算法3优化

- 判断01字符串是否是回文可以用Hash来处理, 但是时间复杂度仍是 $O(TN^2)$ 。
- 可以用线段树动态维护01串修改后的Hash值, 时间复杂度 $O(TN \log N)$ 。

```

1  #include <bits/stdc++.h>
2  using namespace std;
3
4  const int N = 1e4 + 10;
5  const long long P = 13331;
6  const long long MOD = 1e9 + 7;
7  int a[N];
8  long long p[N]; // P^k % MOD
9  // 正向和逆向Hash
10 long long h1[N << 2], h2[N << 2];
11 int n;
12
13 void update(int rt, int l, int r, int x)
14 {

```

```

15     if(l == r)
16     {
17         h1[rt] = h2[rt] = 1;
18         return ;
19     }
20     int m = (l + r) >> 1, ls = rt << 1, rs = rt << 1 | 1;
21     if(m >= x) update(ls, l, m, x);
22     else update(rs, m + 1, r, x);
23     h1[rt] = (h1[ls] * p[r - m] % MOD + h1[rs]) % MOD;
24     h2[rt] = (h2[ls] + h2[rs] * p[m - l + 1] % MOD) % MOD;
25 }
26
27 long long query1(int rt, int l, int r, int x, int y)
28 {
29     if(x <= l && r <= y) return h1[rt];
30     int m = (l + r) >> 1, ls = rt << 1, rs = rt << 1 | 1;
31     if(m >= y) return query1(ls, l, m, x, y);
32     else if(m < x) return query1(rs, m + 1, r, x, y);
33     else
34     {
35         int al = query1(ls, l, m, x, m);
36         int ar = query1(rs, m + 1, r, m + 1, y);
37         return (al * p[y - m] % MOD + ar) % MOD;
38     }
39 }
40
41 long long query2(int rt, int l, int r, int x, int y)
42 {
43     if(x <= l && r <= y) return h2[rt];
44     int m = (l + r) >> 1, ls = rt << 1, rs = rt << 1 | 1;
45     if(m >= y) return query2(ls, l, m, x, y);
46     else if(m < x) return query2(rs, m + 1, r, x, y);
47     else
48     {
49         int al = query2(ls, l, m, x, m);
50         int ar = query2(rs, m + 1, r, m + 1, y);
51         return (al + ar * p[m - x + 1] % MOD) % MOD;
52     }
53 }
54
55 int main()
56 {
57     p[0] = 1;
58     for(int i = 1; i <= 10000; ++i) p[i] = p[i - 1] * P % MOD;
59     int T; scanf("%d", &T);
60     while(T--)
61     {
62         scanf("%d", &n);
63         for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
64         memset(h1, 0, sizeof(h1));
65         memset(h2, 0, sizeof(h2));
66         bool flag = false;
67         for(int i = 1; i <= n; ++i)
68         {
69             update(1, 1, n, a[i]);
70             if(a[i] == 1 || a[i] == n) continue;

```

```
70         int d = min(a[i] - 1, n - a[i]);    // 边界
71     }
72     long long x = query1(1, 1, n, a[i] - d, a[i] + d);
73     long long y = query2(1, 1, n, a[i] - d, a[i] + d);
74     if(x != y) { flag = true; break; }
75 }
76 if(flag) puts("Y");
77 else puts("N");
78 }
79 return 0;
80 }
```