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)。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 1e4 + 10;
int a[N];
int n;

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        scanf("%d", &n);
        for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
        bool flag = false;
        for(int p = 1; p <= n; ++p)
            for(int q = p + 1; q <= n; ++q)
                for(int r = q + 1; r <= n; ++r)
                    if(a[p] - a[q] == a[q] - a[r])
                        flag = true;
        if(flag) puts("Y");
        else puts("N");
    }
    return 0;
}
```

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。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 1e4 + 10;
int a[N], h[N * 2];
int n;

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        scanf("%d", &n);
        for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
        bool flag = false;
        for(int q = 2; q <= n - 1; ++q)
        {
            for(int i = 1; i <= 2 * n; ++i) h[i] = false;
            for(int p = 1; p <= q - 1; ++p) h[a[p] - a[q] + n] = true;
            for(int r = q + 1; r <= n; ++r)
                if(h[a[q] - a[r] + n]) flag = true;
            if(flag) break;
        }
        if(flag) puts("Y");
        else puts("N");
    }
    return 0;
}
```

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$1, 0, 1, 0, 1, 0$，$b_{1\sim5}$是回文。
  - 序列是$1, 6, 3, 5, 4, 2$，那么当扫描到3时，b数组为$1, 0, 1, 0, 0, 1$，$b_{1\sim5}$不是回文。

- 时间复杂度：O(TN^2)，但是常数比算法2小。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 1e4 + 10;
int a[N], b[N];
int n;

int main()
{
    int T; scanf("%d", &T);
    while(T--)
    {
        memset(b, 0, sizeof(b));
        scanf("%d", &n);
        for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
        bool flag = false;
        for(int i = 1; i <= n; ++i)
        {
            b[a[i]] = 1;
            if(a[i] == 1 || a[i] == n) continue;
            int d = min(a[i] - 1, n - a[i]);    // 边界
            for(int j = a[i] - d; j <= a[i] - 1; ++j)
                if(b[j] != b[2 * a[i] - j]) flag = true;
            if(flag) break;
        }
        if(flag) puts("Y");
        else puts("N");
    }
    return 0;
}
```

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($TN\log N$)。

```cpp
#include <bits/stdc++.h>
using namespace std;

const int N = 1e4 + 10;
const long long P = 13331;
const long long MOD = 1e9 + 7;
int a[N];
long long p[N]; // P^k % MOD
// 正向和逆向Hash
long long h1[N << 2], h2[N << 2];
int n;

void update(int rt, int l, int r, int x)
{
```

```
    if(l == r)
    {
        h1[rt] = h2[rt] = 1;
        return ;
    }
    int m = (l + r) >> 1, ls = rt << 1, rs = rt << 1 | 1;
    if(m >= x) update(ls, l, m, x);
    else update(rs, m + 1, r, x);
    h1[rt] = (h1[ls] * p[r - m] % MOD + h1[rs]) % MOD;
    h2[rt] = (h2[ls] + h2[rs] * p[m - l + 1] % MOD) % MOD;
}

long long query1(int rt, int l, int r, int x, int y)
{
    if(x <= l && r <= y) return h1[rt];
    int m = (l + r) >> 1, ls = rt << 1, rs = rt << 1 | 1;
    if(m >= y) return query1(ls, l, m, x, y);
    else if(m < x) return query1(rs, m + 1, r, x, y);
    else
    {
        int al = query1(ls, l, m, x, m);
        int ar = query1(rs, m + 1, r, m + 1, y);
        return (al * p[y - m] % MOD + ar) % MOD;
    }
}

long long query2(int rt, int l, int r, int x, int y)
{
    if(x <= l && r <= y) return h2[rt];
    int m = (l + r) >> 1, ls = rt << 1, rs = rt << 1 | 1;
    if(m >= y) return query2(ls, l, m, x, y);
    else if(m < x) return query2(rs, m + 1, r, x, y);
    else
    {
        int al = query2(ls, l, m, x, m);
        int ar = query2(rs, m + 1, r, m + 1, y);
        return (al + ar * p[m - x + 1] % MOD) % MOD;
    }
}

int main()
{
    p[0] = 1;
    for(int i = 1; i <= 10000; ++i) p[i] = p[i - 1] * P % MOD;
    int T; scanf("%d", &T);
    while(T--)
    {
        scanf("%d", &n);
        for(int i = 1; i <= n; ++i) scanf("%d", &a[i]);
        memset(h1, 0, sizeof(h1));
        memset(h2, 0, sizeof(h2));
        bool flag = false;
        for(int i = 1; i <= n; ++i)
        {
            update(1, 1, n, a[i]);
            if(a[i] == 1 || a[i] == n) continue;
```

```
            int d = min(a[i] - 1, n - a[i]);    // 边界
            long long x = query1(1, 1, n, a[i] - d, a[i] + d);
            long long y = query2(1, 1, n, a[i] - d, a[i] + d);
            if(x != y) { flag = true; break; }
        }
        if(flag) puts("Y");
        else puts("N");
    }
    return 0;
}
```