过河

(river.pas/c/cpp)

【问题描述】

在河上有一座独木桥，一只青蛙想沿着独木桥从河的一侧跳到另一侧。在桥上有一些石子，青蛙很讨厌踩在这些石子上。由于桥的长度和青蛙一次跳过的距离都是正整数，我们可以把独木桥上青蛙可能到达的点看成数轴上的一串整点：0，1，……，L（其中 L 是桥的长度）。坐标为 0 的点表示桥的起点，坐标为 L 的点表示桥的终点。青蛙从桥的起点开始，不停的向终点方向跳跃。一次跳跃的距离是 S 到 T 之间的任意正整数（包括 S,T）。当青蛙跳到或跳过坐标为 L 的点时，就算青蛙已经跳出了独木桥。

题目给出独木桥的长度 L，青蛙跳跃的距离范围 S,T，桥上石子的位置。你的任务是确定青蛙要想过河，最少需要踩到的石子数。

【输入文件】

输入文件 **river.in** 的第一行有一个正整数 L（1 <= L <= 10^9），表示独木桥的长度。第二行有三个正整数 S，T，M，分别表示青蛙一次跳跃的最小距离，最大距离，及桥上石子的个数，其中 1 <= S <= T <= 10，1 <= M <= 100。第三行有 M 个不同的正整数分别表示这 M 个石子在数轴上的位置（数据保证桥的起点和终点处没有石子）。所有相邻的整数之间用一个空格隔开。

【输出文件】

输出文件 **river.out** 只包括一个整数，表示青蛙过河最少需要踩到的石子数。

【样例输入】

```
10
2 3 5
2 3 5 6 7
```

【样例输出】

```
2
```

【数据规模】

对于 30%的数据，L <= 10000；
对于全部的数据，L <= 10^9。

【问题分析】

显然，对于 L 很小的数据来说，不难写出动态规划的状态转移方程

$opt[n] = \min\{opt[n-i] + (rock[n]\}, S \le i \le T\}$

这个方程是 $O(n)$ 的，显然对于极限数据无法出解。通过观察发现，石子其实很少，解决问题的关键就在于，将石子离散出来分开考虑。

通过数学知识，不难证明， $px + (p+1)y = Q$ ，在 $Q \ge p*(p+1)$ 时是一定有解的。由于题目给出的是[S, T]的一个区间，于是，显然当相邻的两个石子之间的距离超过 90 时，则后面的距离都可以到达，我们就可以认为它们之间的距离就是 90。如此一来，我们就将原题 L 的范围缩小为了 100*90=9000，动态规划算法已经完全可以承受了。

不要高兴的太早！问题还没有解决：当 S=T 时，上述等式是无法使用的，在这种情况下，只需要统计出所有石子中，是 S 倍数的个数，就可以了。

【代码清单】

```
#include <cstdio>
#include <algorithm>
using namespace std;

const int MaxN = 110, Bound = 90;

int A[MaxN], B[MaxN], L, S, T, N;
bool Rock[Bound * (MaxN + 10)];
int opt[Bound * (MaxN + 10)];

void init() {
    freopen("river.in", "r", stdin);
    freopen("river.out", "w", stdout);

    A[0] = 0;
    scanf("%d%d%d%d", &L, &S, &T, &N);
    for (int i = 1; i <= N; i ++)
        scanf("%d", &A[i]);
    A[++ N] = L;
    sort(A, A + N);
}

void work() {
    int i, j, Add;

    B[0] = 0;
    for (i = 1; i <= N; i ++) {
        //当距离超过某值时合并
        if (A[i] - A[i - 1] > Bound) B[i] = B[i - 1] + Bound;
        else B[i] = B[i - 1] + A[i] - A[i - 1];
```

```
        Rock[B[i]] = true;
    }

    memset(opt, 63, sizeof(opt));
    opt[0] = 0;
    for (i = 0; i <= B[N] + Bound; i ++) {
        if (Rock[i]) Add = 1; else Add = 0;
        for (j = S; j <= T; j ++)
            //动态规划
            if (i - j >= 0 && opt[i - j] + Add < opt[i]) opt[i] =
opt[i - j] + Add;
    }
    int Min = 110;
    for (i = B[N]; i <= B[N] + Bound; i ++)
        if (opt[i] < Min) Min = opt[i];
    printf("%d\n", Min);
}

int main() {
    init();
    if (S == T) {
        int Sum = 0;
        for (int i = 1; i < N; i ++)
            if (A[i] % S == 0) Sum ++;
        printf("%d\n", Sum);
    } else work();
    return 0;
}
```

【小结】

数学分析在解决信息学问题时起到决定性作用。