Greedy Algorithms

Let n be a positive integer. We want to reduce n to 1 by applying a sequence of simple operations on n. The operations permitted are division of n by two (provided that n is even), and addition of a small positive or negative integer to n. Your objective is to reach the final goal 1 with as few applications of these operations as possible. For some choices of the small increments/decrements, greedy algorithms work. Parts 1 and 2 deal with two such cases. However, the greedy algorithm does not always work. Part 3 demonstrates that the greedy approach is not guaranteed to give an optimal solution. In Part 4, you devise an algorithm that necessarily produces optimal solutions.

Part 1

In this part, assume that the only operations permitted are decrement (by 1) and division by 2 (allowed if and only if n is even). For example, consider this reduction of 125 to 1, which uses 12 steps:

$$125 \to 124 \to 62 \to 31 \to 30 \to 15 \to 14 \to 13 \to 12 \to 6 \to 3 \to 2 \to 1.$$

This sequence is not optimal. An obvious greedy strategy is based on the fact that division by 2 produces the best possible local reduction. However, if n is odd, this operation is not allowed, so decrement n to make it even, and then divide by 2. The optimal solution involves 11 steps only:

$$125 \to 124 \to 62 \to 31 \to 30 \to 15 \to 14 \to 7 \to 6 \to 3 \to 2 \to 1.$$

Implement this greedy algorithm in a function **`greedy1`**.

Part 2

Now, suppose that besides decrement by 1 and division by 2, you are permitted to increment n by 1. A greedy approach in this case is to keep on dividing n by 2 so long as it remains even. Once n becomes odd, compute $n-1$ and $n+1$, and choose the one which has a larger number of factors of 2. This is illustrated by the next example. This reduction has nine steps.

$$125 \to 124 \to 62 \to 31 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1.$$

A long sequence of consecutive increments/decrements does not help:

$$125 \to 126 \to 127 \to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1.$$

Changing 125 to 128 by increments is not helpful. A shorter sequence is achieved at a reduced level by one increment of 31 to 32. This greedy strategy fails only for $n = 3$. The greedy reduction is $3 \to 4 \to 2 \to 1$, whereas the optimal reduction is $3 \to 2 \to 1$. Write a function **`greedy2`** to implement these observations.

Part 3

Now, you are allowed to increment n by one of the k values $a_0, a_1, a_2, \ldots, a_{k-1}$. Here, each a_i is a small positive or negative integer. Division of even integers by 2 continues to remain allowed. Following the ideas of the last two parts, we can think of a greedy approach that keeps on dividing n by 2 so long as it remains even. Once n becomes odd, you try the one among $n+a_0, n+a_1, n+a_2, \ldots, n+a_{k-1}$, which has the largest number of factors of 2. In case of ties, choose the smallest one with the largest number of factors of 2. In order that this reduction is always possible, assume that a_0 is either 1 or -1. Here follows an example, in which $k = 2$, $a_0 = -1$, and $a_1 = 4$:

$$125 \to 124 \to 62 \to 31 \to 30 \to 15 \to 14 \to 7 \to 6 \to 3 \to 2 \to 1.$$

This is the same as the optimal reduction of Part 1. In particular, the option "increment by 4" is unusable under this greedy strategy (why?). We soon see that this greedy strategy is not optimal. Nevertheless, write a function **`greedy3`** to implement this greedy algorithm. Notice that during the reduction process, n is never allowed to become zero or negative. Moreover, take care that this algorithm should not enter an infinite loop (consider $n = 25$, $k = 2$, $a_0 = -1$, $a_1 = 9$).

Part 4

In order to produce optimal solutions in the general case introduced in Part 3, one strategy is to avoid the temptation of dividing n by 2 as soon as it becomes even. For that matter, we can modify the greedy algorithm of Part 3 to choose, for division by 2, the smallest one from $n, n+a_0, n+a_1, n+a_2, \ldots, n+a_{k-1}$ with the largest number of factors of 2. You can work out that this modified greedy strategy does not change the solution with 11 steps, given in Part 3. It may help elsewhere though.

The second and more crucial observation is that we should allow multiple addition operations before division by 2 starts. Indeed, an optimal reduction of 125 with $k = 2$, $a_0 = -1$, and $a_1 = 4$ is this (9 steps only):

$$125 \to 124 \to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1.$$

Since addition is a commutative operation, this reduction can be rephrased as

$$125 \to 129 \to 128 \to 64 \to 32 \to 16 \to 8 \to 4 \to 2 \to 1.$$

Here, two additions are applied before division by 2.

How many consecutive additions we should do before deciding to divide by 2? Since $a_0 = \pm 1$, using the best solution of Part 1 or 2 implies about $\log_2 n$ divisions and at most $\log_2 n$ decrements and/or increments. Since a_i are small, adding each to n does not change n substantially. That is, we have to do $\log_2 n$ divisions anyway. So a sequence of consecutive additions of length larger than $\log_2 n$ cannot lead to an optimal solution.

Implement an $O(kn \log n)$-time function **`optimal`** to compute an optimal solution in the general case. You may assume $k \leqslant 5$, $1 \leqslant |a_i| \leqslant 9$, and $a_0 \in \{-1, 1\}$. A recursive formulation of this problem for even n is

$$B_n = 1 + \min(B_{n/2}, B_{n+a_0}, B_{n+a_1}, B_{n+a_2}, \ldots, B_{n+a_{k-1}}),$$

where B_i is the best move count for i. For odd n, the first argument $B_{n/2}$ of min should be absent. If a_i is positive, we have $n + a_i > n$, but the solution B_n requires the solution B_{n+a_i} (in the above example, B_{124} requires B_{128}, and B_{125} requires B_{129}).

Write a loop for iteratively improving the best solutions for each i in the range $[2, m]$, where m is slightly (how much?) larger than n. Consider the reduction of 125. The first iteration discovers the best reduction sequence for 128. The second iteration discovers the best sequence for 129 by noticing that $128 = 129 + a_0$. Yet another iteration is needed to find an optimal move for 125 by making use of the formula $129 = 125 + a_1$. You may run the loop $\log_2 n$ times, or break when no further improvements are achieved in an iteration.

The *main*() function

- Read n from the user.
- Call **`greedy1`** and **`greedy2`** on this n, and report the greedy reductions of n. These are optimal.
- Read k, and $a_0, a_1, a_2, \ldots, a_{k-1}$ from the user.
- Call **`greedy3`** on $n, k, a_0, a_1, a_2, \ldots, a_{k-1}$ to show the performance of the greedy strategy.
- Call **`optimal`** on $n, k, a_0, a_1, a_2, \ldots, a_{k-1}$ to obtain an optimal solution.

Submit a single C/C++ source file. Do not use global/static variables.

Sample output

```
n = 125

+++ Greedy 1
    Start     : 125
    Decrement : 124
    Divide    : 62
    Divide    : 31
    Decrement : 30
    Divide    : 15
    Decrement : 14
    Divide    : 7
    Decrement : 6
    Divide    : 3
    Decrement : 2
    Divide    : 1
--- Number of steps = 11

+++ Greedy 2
    Start     : 125
    Decrement : 124
    Divide    : 62
    Divide    : 31
    Increment : 32
    Divide    : 16
    Divide    : 8
    Divide    : 4
    Divide    : 2
    Divide    : 1
--- Number of steps = 9

k = 2
-1 4

+++ Greedy 3
    Start     : 125
    Add -1    : 124
    Divide    : 62
    Divide    : 31
    Add -1    : 30
    Divide    : 15
    Add -1    : 14
    Divide    : 7
    Add -1    : 6
    Divide    : 3
    Add -1    : 2
    Divide    : 1
--- Number of steps = 11

+++ Optimal
    Start     : 125
    Add -1    : 124
    Add  4    : 128
    Divide    : 64
    Divide    : 32
    Divide    : 16
    Divide    : 8
    Divide    : 4
    Divide    : 2
    Divide    : 1
--- Number of steps = 9
```