CS29003 Algorithms Laboratory

Assignment No: 2

Last date of submission: 22–January–2019

Divide-and-Conquer Algorithms

In this assignment, you solve the 4-peg Tower-of-Hanoi problem. The puzzle was introduced by François Édouard Anatole Lucas in 1883. The 3-peg version is well understood, and its time complexity is easily provable. The 4-peg version eluded mathematicians for over a century. In 1941, J.S. Frame and B.M. Stewart independently proposed a particular way of solving the 4-peg puzzle, which is popularly known as the Frame–Stewart algorithm (see below). In 1994, Paul Stockmeyer calculated an approximate closed-form expression for the optimal number of moves made by the Frame–Stewart algorithm. Very recently, Roberto Demontis (December 2018) proved that the Frame–Stewart algorithm is indeed optimal.

There are four pegs A, B, C, D (numbered as $0, 1, 2, 3$). Initially, n disks of diameters $1, 2, 3, \ldots, n$ stacked on Peg A in the sorted order (the smallest disk at the top, and the largest disk at the bottom). The three other pegs are initially empty. Your task is to transfer the n disks from Peg A to Peg B taking help from the other two pegs, using a sequence of moves. Each move transfers a *single* disk d from Peg p to Peg q ($p \neq q$), such that the following two conditions hold just before the movement: (i) Disk d must be sitting at the top of Peg p, and (ii) Disk d is not allowed to be larger than the disk (if any) sitting on the top of Peg q.

If only three pegs are allowed, we know that the best (minimum) number of moves is $T_3(n) = 2^n - 1$, and a straightforward divide-and-conquer algorithm solves the 3-peg puzzle using exactly these many moves. If we are allowed to use a fourth peg, the Frame–Stewart algorithm is used, which involves the following steps (also see the figure on the next page).

1. If $n \leqslant 3$, solve the problem directly.
2. Fix a value of k in the range $1 \leqslant k \leqslant n$.
3. Keep the k largest disks on Peg A, and transfer the smallest $n-k$ disks from Peg A to Peg D.
4. Transfer the largest k disks from Peg A to Peg B without disturbing the smallest $n-k$ disks already sitting on Peg D. Since a larger disk can never be put on the top of a smaller disk, Peg D is unusable in this part, that is, we solve the 3-peg Tower-of-Hanoi problem on k disks.
5. Transfer the smallest $n-k$ disks from Peg D to Peg B without disturbing the largest k disks on Peg B. In this step, all the four pegs can be used.

Steps 3 and 5 are solved recursively, so the running time of this algorithm satisfies

$$T_4(n) = T_4(n-k) + T_3(k) + T_4(n-k) = 2T_4(n-k) + 2^k - 1.$$

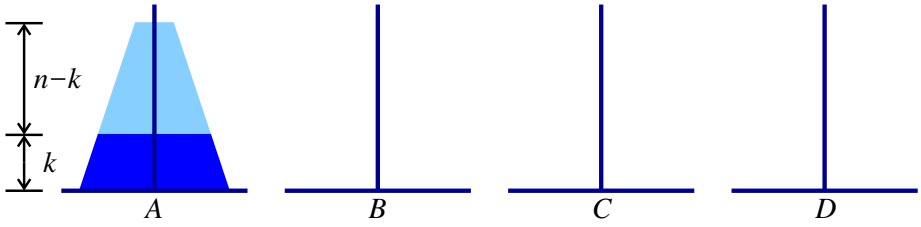
This performance depends on the choice of k. Consider a recursive call for the transfer of Disks i through j. This involves $m = j - i + 1$ disks. Initially (in the outermost call), we have $i = 1$, $j = n$, and $m = n$. The parameter k is (usually) chosen as a function of m.

Part 0: Step 4 of the Frame–Stewart algorithm needs a solution of the 3-peg Tower-of-Hanoi problem. Write a function `ToH3(i,j,p,q,r)` to transfer Disks i through j from Peg p to Peg q using a third Peg r.

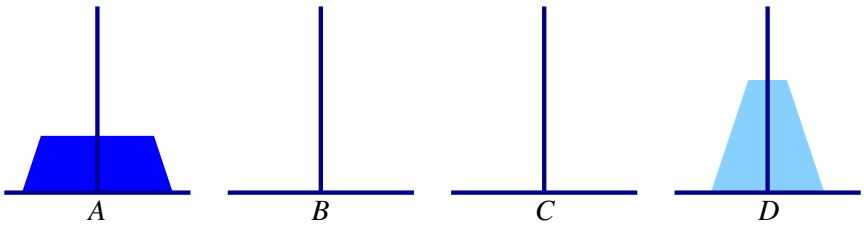
The remaining three parts implement the Frame–Stewart algorithm for different choices of k.

Part 1: Write a function `ToH41(i,j,p,q,r,s)` to transfer Disks i through j from Peg p to Peg q using the two other pegs r and s. Here, you take $k = \lfloor m/2 \rfloor$ (where $m = j - i + 1$ as defined above). This is a natural choice of breaking the problem into two equal halves, but this choice turns out to be quite poor.

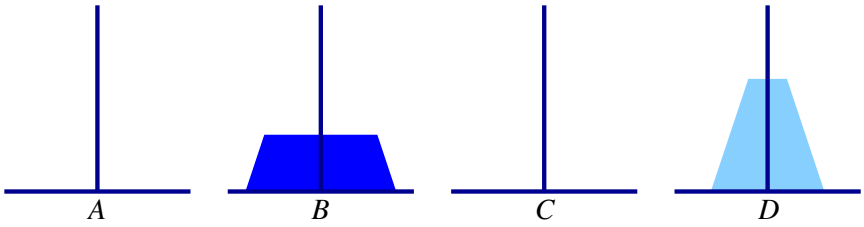
Part 2: Write a function `ToH42(i,j,p,q,r,s,k)` to transfer Disks i through j from Peg p to Peg q using the two other pegs r and s. Here, we use a *fixed* value of k independent of the size $m = j - i + 1$ of the subproblem posed to the recursive invocation. If $m \leqslant k$, solve the problem using the 3-peg Tower-of-Hanoi algorithm. Otherwise, follow the Frame–Stewart algorithm.



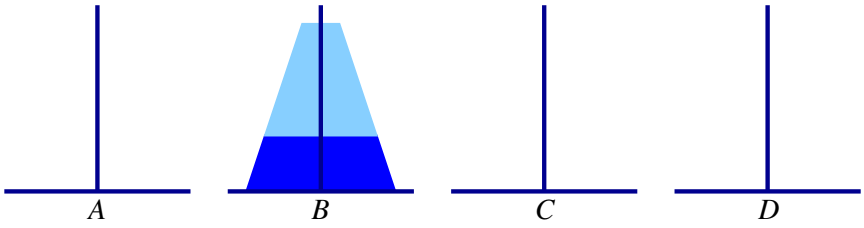
Initial Configuration



Step 1: After first recursive call on the smaller part



Step 2: After movement of the larger part using three pegs



Step 3: After second recursive call on the smaller part

The best value of k is determined beforehand as follows. Unfolding the recurrence for $T_4(n)$ gives the following calculations. For simplicity, we assume that $n = uk$.

$$\begin{aligned}
T_4(n) &\approx 2T(n-k)+2^k \\
&\approx 2^2T(n-2k)+(2+1)\times 2^k \\
&\approx 2^3T(n-3k)+(2^2+2+1)\times 2^k \\
&\cdots \\
&\approx 2^{u-1}T(k)+(2^{u-2}+\cdots+2^2+2+1)\times 2^k \\
&\approx (2^{u-1}+2^{u-2}+\cdots+2^2+2+1)\times 2^k \\
&\approx 2^{u+k} \\
&= 2^{\frac{n}{k}+k}.
\end{aligned}$$

The quantity $\frac{n}{k}+k$ is minimized for $k=\sqrt{n}$, and correspondingly $T_4(n)\approx 2^{2\sqrt{n}}$. All invocations of `ToH4` uses $k=\lfloor\sqrt{n}\rfloor$, so this value can be precomputed before the outermost call and passed to all recursive invocations.

Part 3: It turns out that the optimal choice for k is $\{\sqrt{2m}\}$ for a recursive call on m disks, where $\{x\}$ is the integer nearest to x. As shown by Stockmeyer, this choice of k leads to $T_4(n)\approx\sqrt{n}2^{\sqrt{2n}}$. Write a function `ToH43(i,j,p,q,r,s)` to implement this optimal version, where i, j, p, q, r, s (and m) are as explained above.

Using the Black Box

In your program, do not maintain the pegs or perform any disk movement. This is carried out by the blackbox BB2 (provided as a binary file for both **gcc** and **g++**). You should only ask the blackbox to move Disk d from Peg p to Peg q by the call **makemove(d,p,q)**. Notice that d is an integer in the range $1 \leqslant d \leqslant n$ (a disk is numbered by its diameter). Moreover, the pegs A, B, C, D are numbered $0, 1, 2, 3$, respectively, so p and q must be integers in the range $0, 1, 2, 3$ with $p \neq q$.

At the beginning of your program, insert the following external function declarations.

```
extern void registerme ( int );
extern void startpart ( int );
extern void endpart ( int );
extern void makemove ( int , int , int );
extern void showpegs ( );
```

At the beginning of your **main()** function, initialize the blackbox by the value of n (to be read from the user). This is to be done by making the following blackbox call. Use $n \leqslant 60$.

```
registerme(n);
```

Solve Parts 1–3 by writing the following lines in your **main()** function. Recall that Part 2 requires a fixed value of k computed outside the call of **ToH42**. You do not have to print anything or keep a count of the disk movements. When you end each part, BB2 tells you the number of disk movements made by your function for that part. You should only issue **makemove()** directives to effect the disk movements. The rest of the book-keeping (well, disk-keeping) is the duty of BB2.

```
startpart(1); ToH41(1,n,0,1,2,3); endpart(1);
startpart(2); ToH42(1,n,0,1,2,3,k); endpart(2);
startpart(3); ToH43(1,n,0,1,2,3); endpart(3);
```

In order to help you (for example, during debugging) the blackbox shows the contents of the four pegs when you make the following call. In the final code that you submit, suppress all these diagnostic calls, since these make the output verbose (unmanageably so if you print the pegs after every disk movement).

```
showpegs();
```

Finally, this is how you compile your code. Since you need math library functions like **sqrt()** and **round()**, compile using the **-lm** flag.

```
gcc mycode.c BB2.o -lm
g++ mycode.cpp BB2.o -lm
```

Sample output

```
n = 30

+++ You are now ready to start Part 1
--- Checking your solution for Part 1
    You have solved the puzzle. Congrats...
    Total number of disk movements made = 33153

+++ You are now ready to start Part 2
--- Checking your solution for Part 2
    You have solved the puzzle. Congrats...
    Total number of disk movements made = 1953

+++ You are now ready to start Part 3
--- Checking your solution for Part 3
    You have solved the puzzle. Congrats...
    Total number of disk movements made = 1025
```

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