An Introduction to Programming though C++

Abhiram G. Ranade

Lecture 6.1

Ch. 10.3: Virahanka Numbers

Virahanka Numbers

- Virahanka was an ancient India prosodist (6^{th}-8^{th} century AD).
- Prosodists study patterns of rhythm and sound in poetry.
- Viharanka asked a question about poetic meters and solved it using recursion.
- A poetic meter is characterized by
  - Number of syllables in the meter
  - The duration of each syllable: short(duration 1), or long (duration 2)
  - Example: SLSSLS is a poetic meter with 6 syllables, total duration 8.

Example of a poetic meter

- “Shardulvikridit”

- Ya kun den du tu shar Haar dhawala yashubh ra vas tra vru ta
- L L L S S L S L S S S L L L S L L s L

19 syllables, Duration 30.

Virahanka's question

"How many poetic meters exist of total duration D?"

- D = 1: {S}
- D = 2: {SS, L}
- D = 3: {SSS, SL, LS}
- D = 4: {SSSS, SSL, SLS, LSS, LL}

Let V(D) denote the number of poetic meters of duration D.

- We have V(1) = 1, V(2) = 2, V(3) = 3, V(4) = 5.

Virahanka wondered whether there is an easy way to calculate V(D).

- Next: nice connections to recursion!

Virahanka's Solution

The first syllable of every meter must be S or L.

S(D) = Set of meters of duration D with first syllable S.

L(D) = Set of meters of duration D with first syllable L.

V(D) = |S(D)| + |L(D)|

Key Question: Suppose I remove the first letter from every meter in S(D), what remains?

- The meters that remain will have duration D-1.
- All possible meters of duration D-1 and only those will now be present in S(D).
- So |S(D)| = V(D-1)

If I remove the first letter from all meters in L(D):

- Each meter that remains will have duration D-2.
- All meters of duration D-2 will be present.
- So |L(D)| = V(D-2)
- **Observation**: V(D) = V(D-1) + V(D-2) for D>2.

- Example: D = 4.
- Set of all meters of duration 4:

{SSSS, SSL, SLS, LSS, LL}

- S(4) = {SSSS, SSL, SLS}
- L(4) = {LSS, LL}
- V(4) = 5, |S(4)| = 3, |L(4)| = 2

After removing first letter from S(4):

{SSS, SL, LS}

= All meters of duration 3

After removing first letter from L(4):

{SS, L}

= All meters of duration 2

What we have discussed

- Virahanka's problem: Find the number $V(D)$ of poetic meters having duration D.
- We can work out by hand: $V(1) = 1$, $V(2) = 2$
- Working out larger $V(D)$ is tedious and error-prone.
- We also know $V(D) = V(D-1) + V(D-2)$ for all $D > 2$.
- Next: A program to calculate $V(D)$

Program to compute V(D)

- Natural to use a recursive function.
- For D > 2 we should use V(D) = V(D-1) + V(D-2)
- Clearly D=1, D=2 should be base cases.

Does this satisfy our 4 requirements?

```
int V(int D){// Precondition: D > 0
  if(D == 1) return 1;
  else if(D == 2) return 2;
  else return V(D-1) + V(D-2);
}
```

- Is the correct value returned for the base cases?
- Do the level 1 calls satisfy the precondition?
  - Level 1 calls made only if D > 2.
- Does the "problem size" reduce? Can it reduce indefinitely?
  - D reduces in each call, but cannot go below 1.
- If the level 1 calls return the correct value, will the top level call return the correct value?
  - V(D) = V(D-1) + V(D-2) for D > 2.

Demo

- Virahanka.cpp

- Observation:
  - Beyond D=45, the time for V(D) seems to increase a lot.

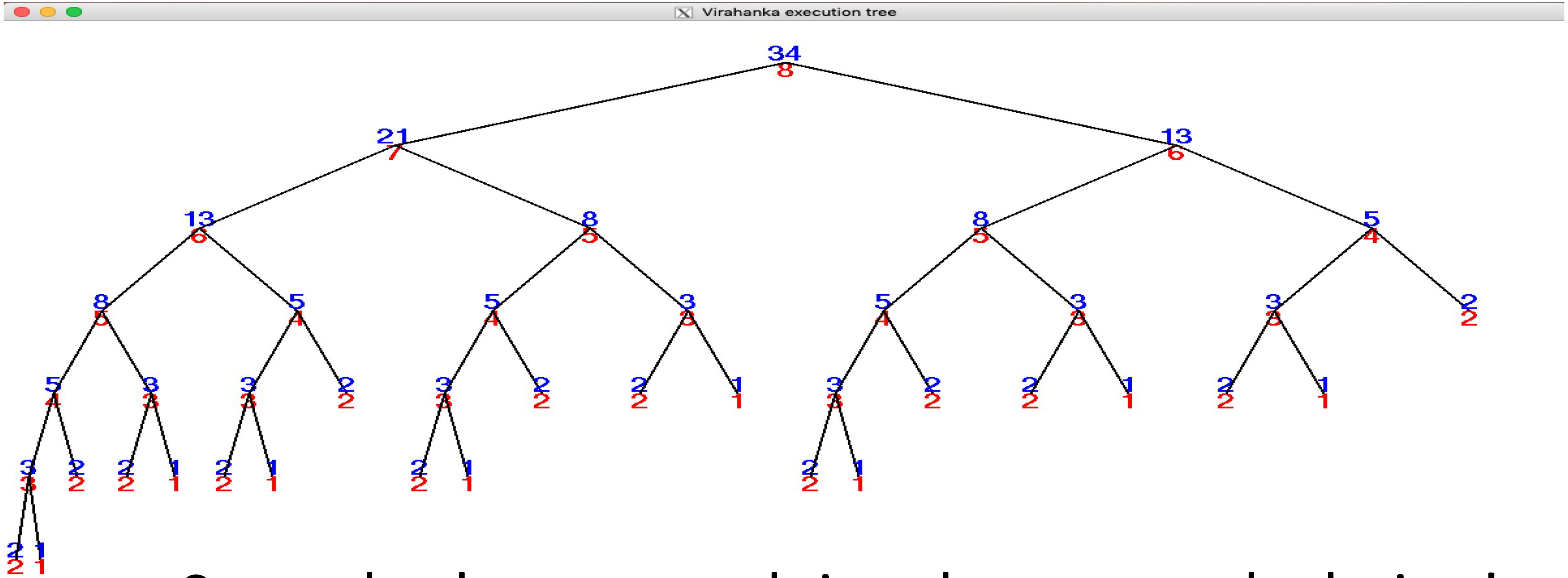
Understanding the execution V(D)

- The execution of any recursive program can be visualized by drawing its “Execution tree” or its “Recursion tree”.
- The execution tree is like the trees seen earlier.
- The root corresponds to the original call, say V(10).
- V(10) will make recursive calls to V(9) and V(8).
- So we will have branches going out of the root to “nodes” for V(9) and V(8).
- From those we will have further branches according to the further recursive calls that get made.
- No branches leave the nodes corresponding to V(1), V(2).

Demo

- Vtree.cpp

Observations from Vtree.cpp



Several subtrees are doing the same calculation!

What we have discussed

- Recursive function to calculate Virahanka numbers V(D).
- The function takes a lot of time for D > 45.
- Execution tree or recursion tree show that we are performing the same calculation several times.
- Next: How to avoid duplication of work

🏓

Can we avoid duplication of work?

- Once we calculate something, do not calculate it again.
- Our function knows V(1) = 1, V(2) = 2
- It first calculates V(3) = V(1) + V(2)
- Later calculates V(4) = V(3) + V(2)
- ...
- Use a loop: In iteration i, i=3..D we will calculate V(i).
- For this we will need to remember V(i-1), V(i-2) calculated earlier
- Let us use variables viminus1, viminus2 for this.
- We will use a variable vi in which to have V(i).

Non recursive calculation of Virahanka numbers

- For iteration 3, we need
  - viminus1 = V(2) = 2,
  - viminus2 = V(1) = 1.
- At the beginning of the iteration we calculate
  - vi = viminus1 + viminus2
- For the next iteration we need
  - viminus2 = viminus1 and
  - viminus1 = vi
- The final result is in vi
- This works only for D ≥ 2.

```
int V(int D){
  int viminus1 = 2;
  int viminus2 = 1;
  int vi;
  for(int i=3; i<=D; i++){
    // viminus1 =V(i-1), viminus2 = V(i-2)
    vi = viminus1 + viminus2;
    viminus2 = viminus1;
    viminus1 = vi;
    // viminus1 = V(i), viminus2 = V(i-1)
  }
  return vi;
}
// Works for D >= 2
```

Demo

- NRV.cpp

- Observation: runs very fast, no calculation repeated.

Remarks

- Recursion is a very powerful tool for solving problems.
- Recursion helps us solve problems and discover algorithms, but:
  - The natural recursive algorithm might be very slow.
  - By examining what the algorithm does, we might be able to discover a better algorithm.
- Recursion trees are a nice way of seeing how a recursive algorithm works.
- Exercise: write the program which draws out the recursion tree
  - Hint: take ideas from basic Virahanka algorithm and tree drawing algorithm.
- The sequence V(1), V(2), ... = 1, 2, 3, 5, 8, 13, ... is famous as the Fibonacci sequence, named after Italian mathematician Fibonacci (12th century AD).
- But we really should call it Virahanka sequence because Virahanka discovered it much earlier.

🏓🏓