Programming and Algorithms II

Dynamic Programming

Nicolas Wu

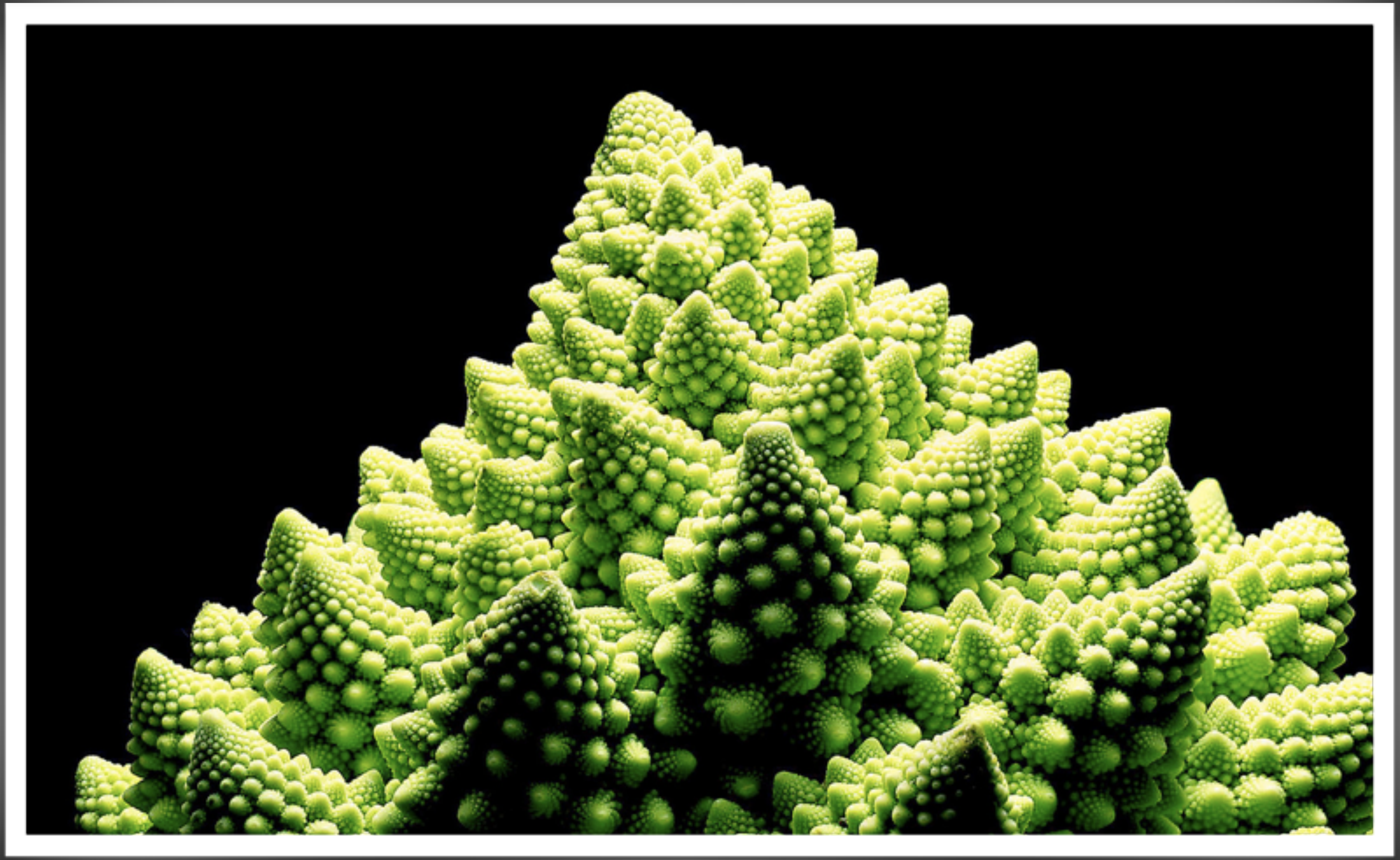
nicolas.wu@bristol.ac.uk

Department of Computer Science
University of Bristol

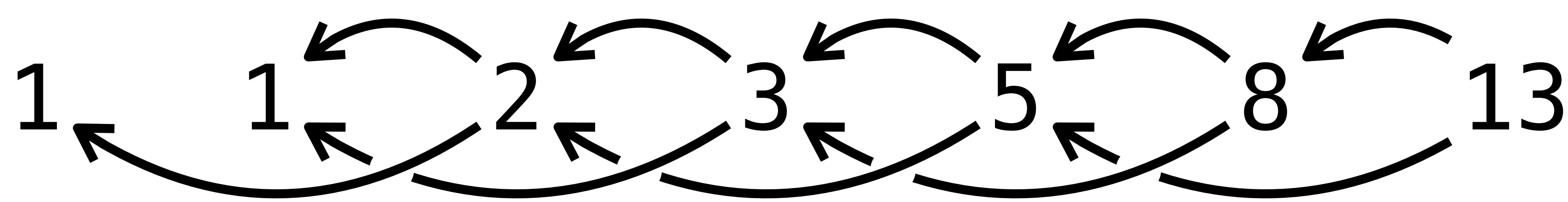
Dynamic Programming

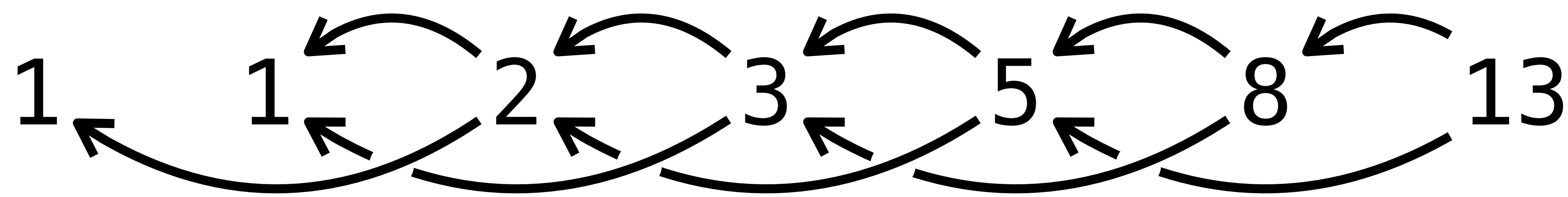
- start with naive recursive definition
- notice repeated subproblems
- tabulate results
- lookup rather than recurse

Trade space for speed

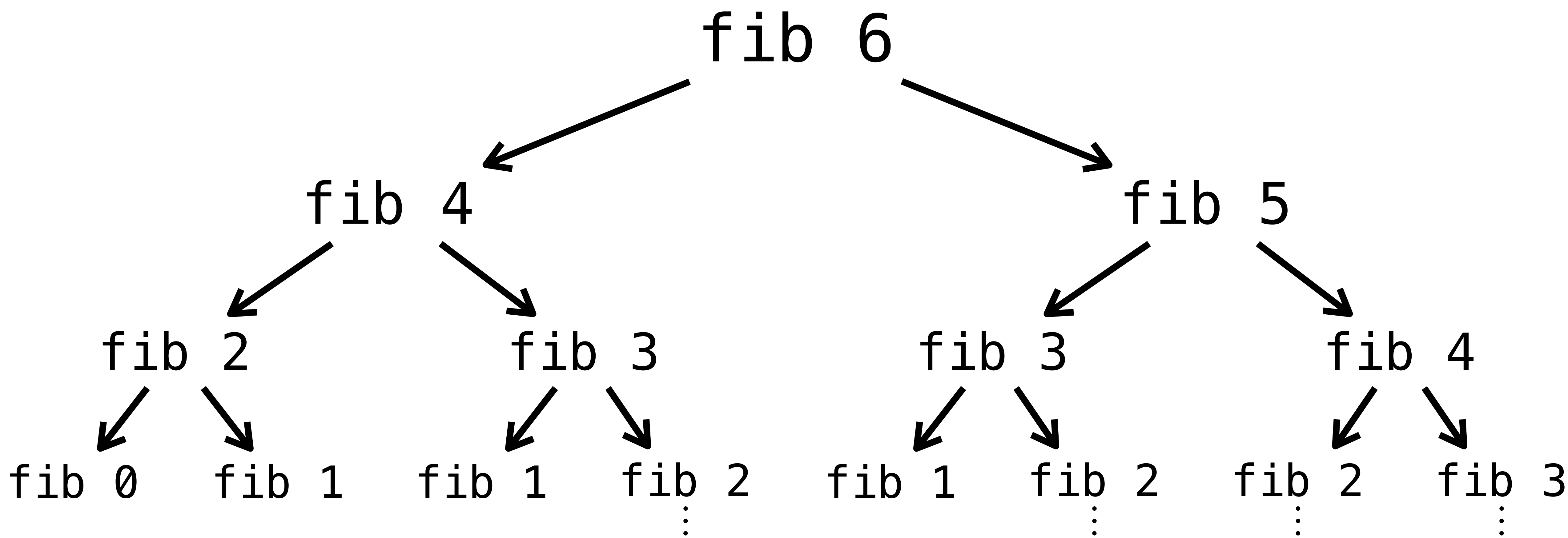


fibonacci

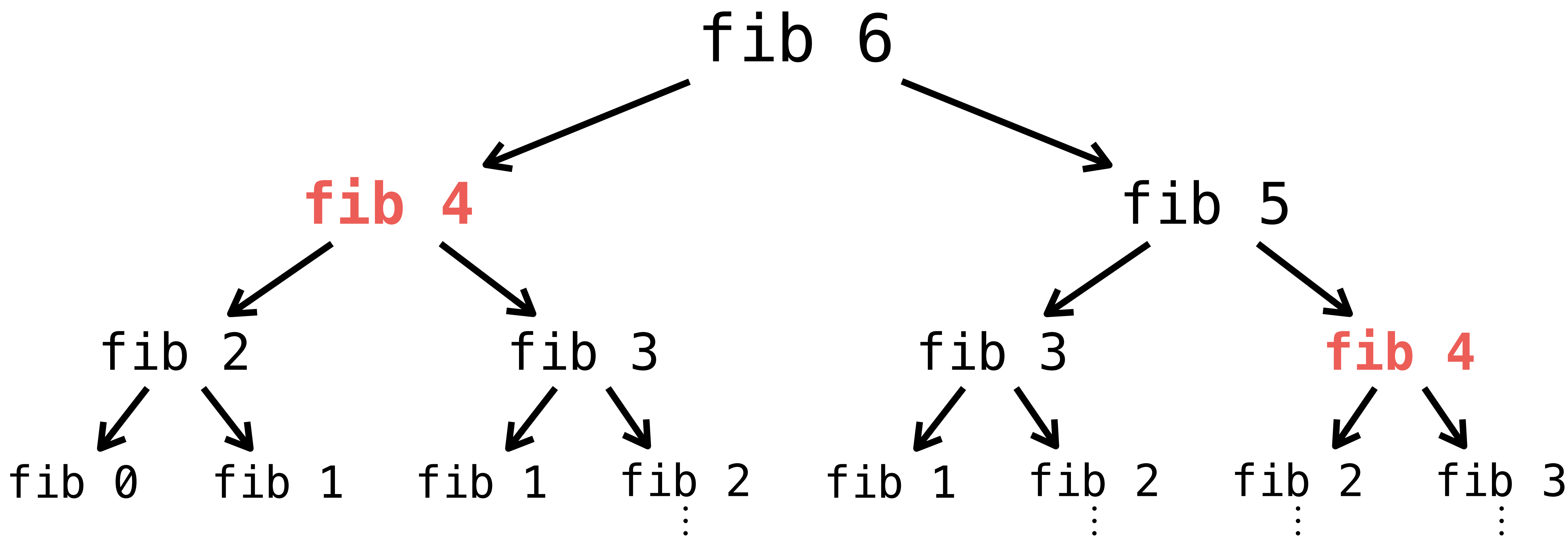

1 1 2 3 5 8 13



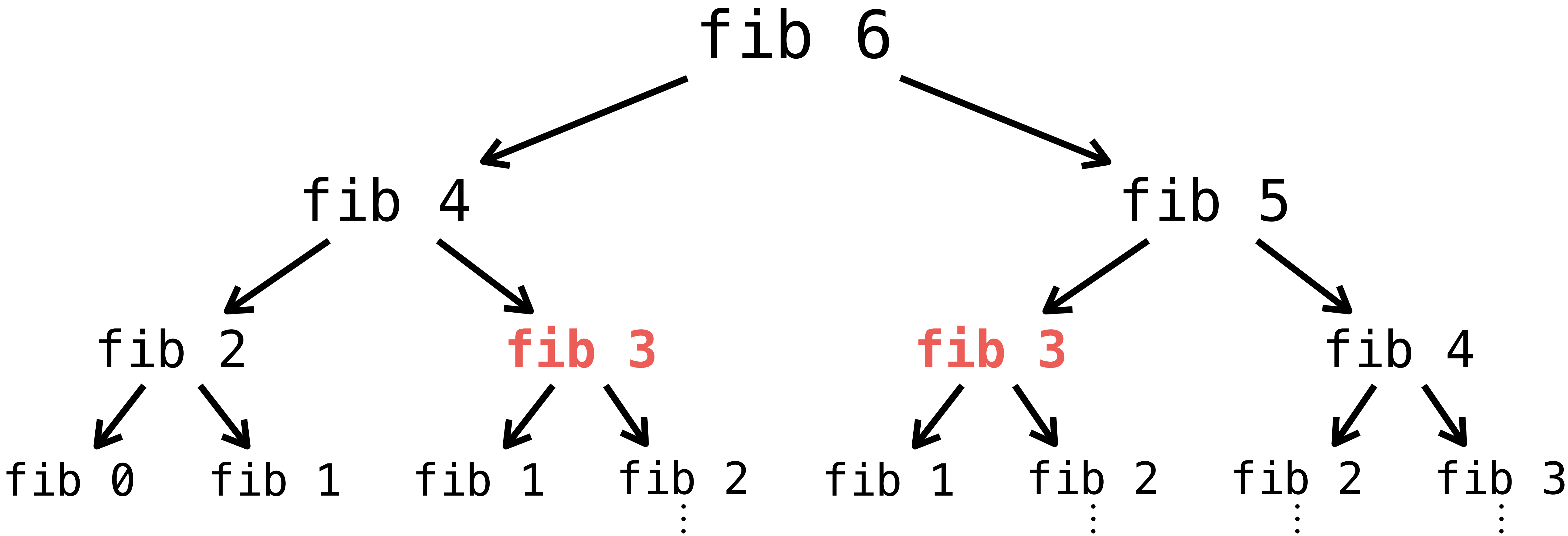

```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n−1) + fib (n−2)
```



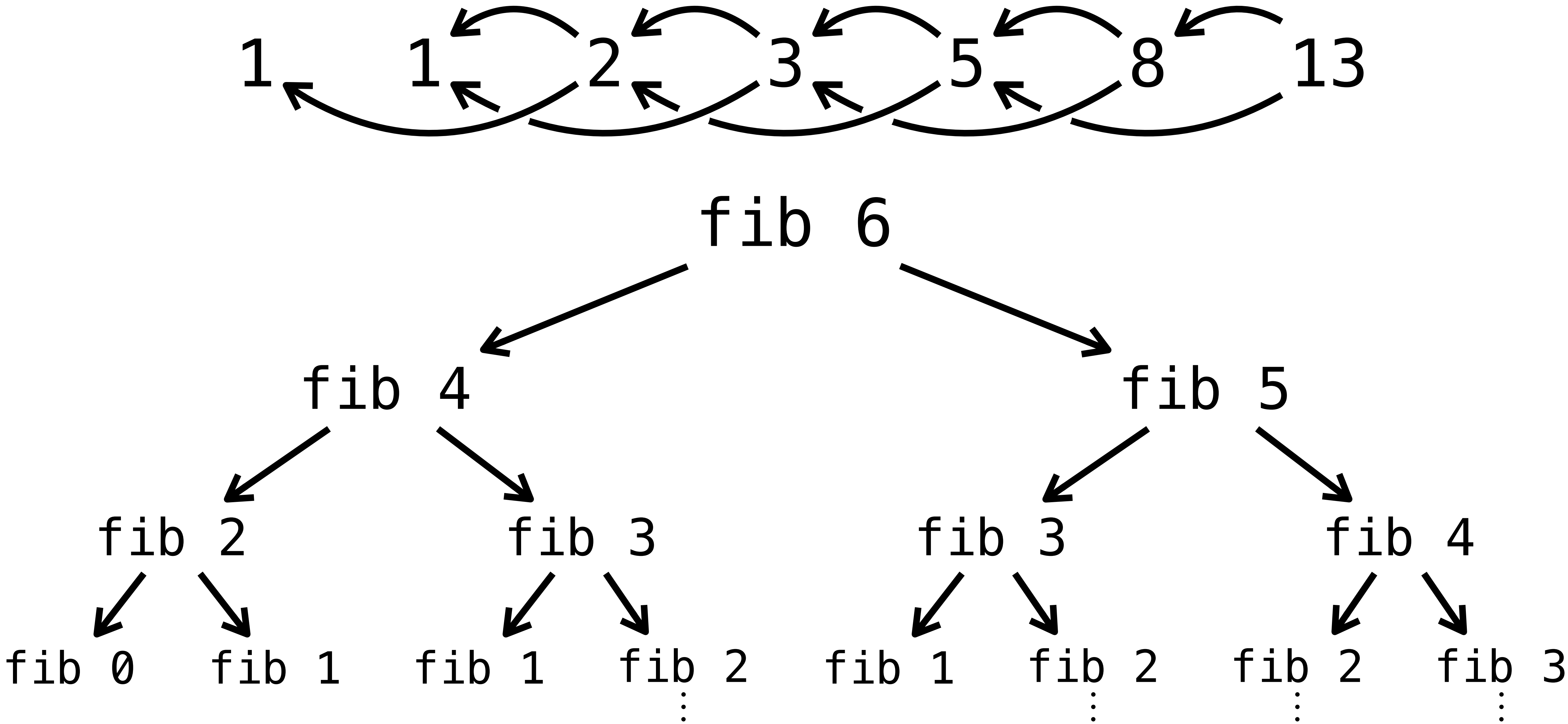

```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n−1) + fib (n−2)
```



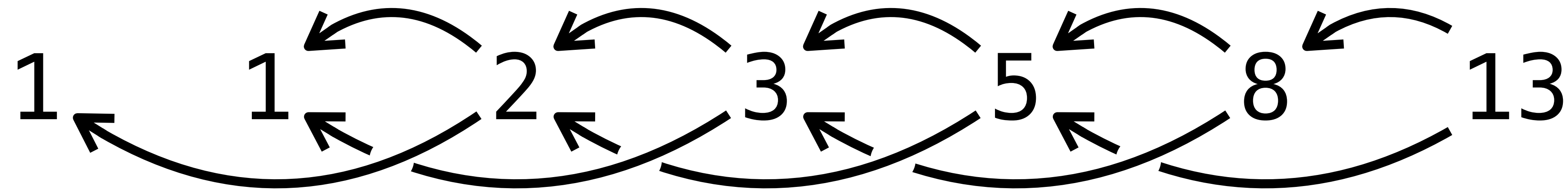

```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n−1) + fib (n−2)
```

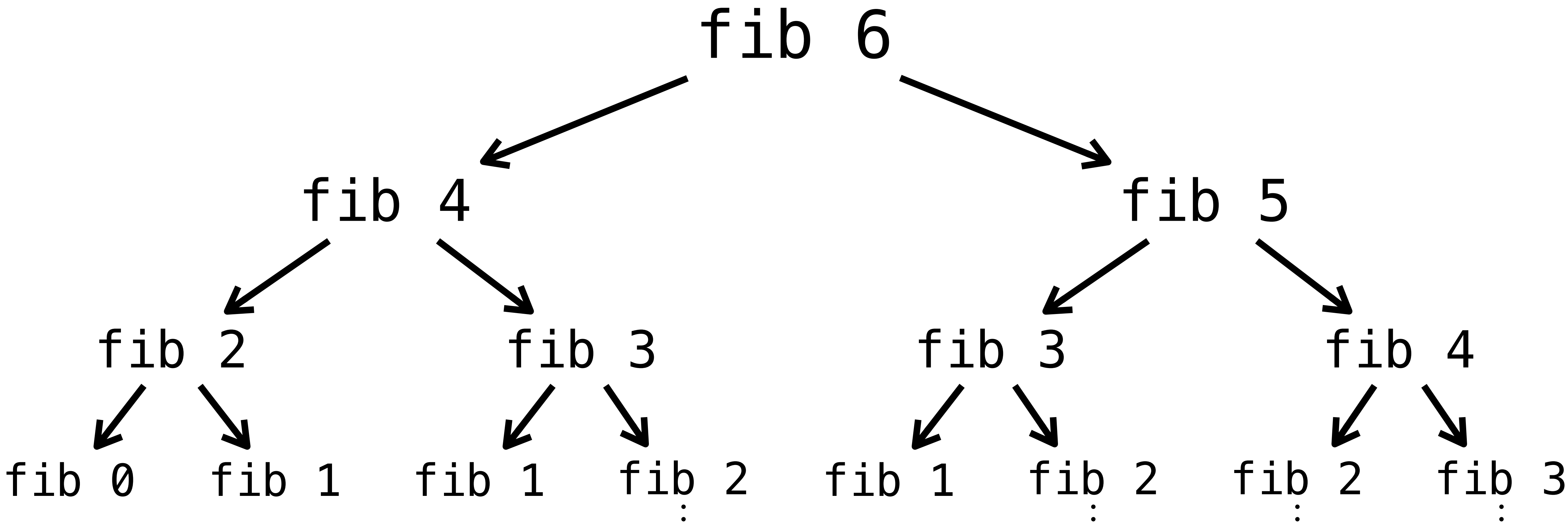



```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n−1) + fib (n−2)
```




```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n−1) + fib (n−2)
```






```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Trade space for speed

```
fib n = table!n where

      ,
      ,
      ,      table!        table!



fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n-1) + fib (n-2)
```

Trade space for speed

```
fib :: Int → Int
fib n = table!n where
  table = tabulate (0,n) fib'
  fib' 0 = 1
  fib' 1 = 1
  fib' n = table!(n−1) + table!(n−2)
```

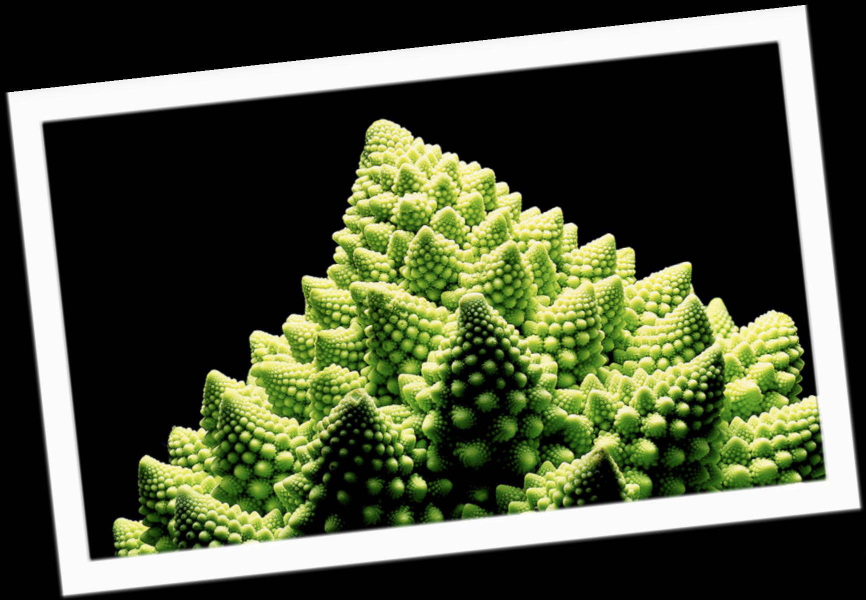
```
fib :: Int → Int
fib 0 = 1
fib 1 = 1
fib n = fib (n−1) + fib (n−2)
```

Trade space for speed

```
fib :: Int → Int
fib n = table!n where
  table = tabulate (0,n) fib’
  fib’ 0 = 1
  fib’ 1 = 1
  fib’ n = table!(n−1) + table!(n−2)
```

```
tabulate :: Ix i ⇒ (i,i) → (i → a) → Array i a
tabulate (m,n) f = array (m,n) [ (i, f i)
                               | i ← [m..n]]

array :: Ix i ⇒ (i,i) → [(i, a)] → Array i a
```



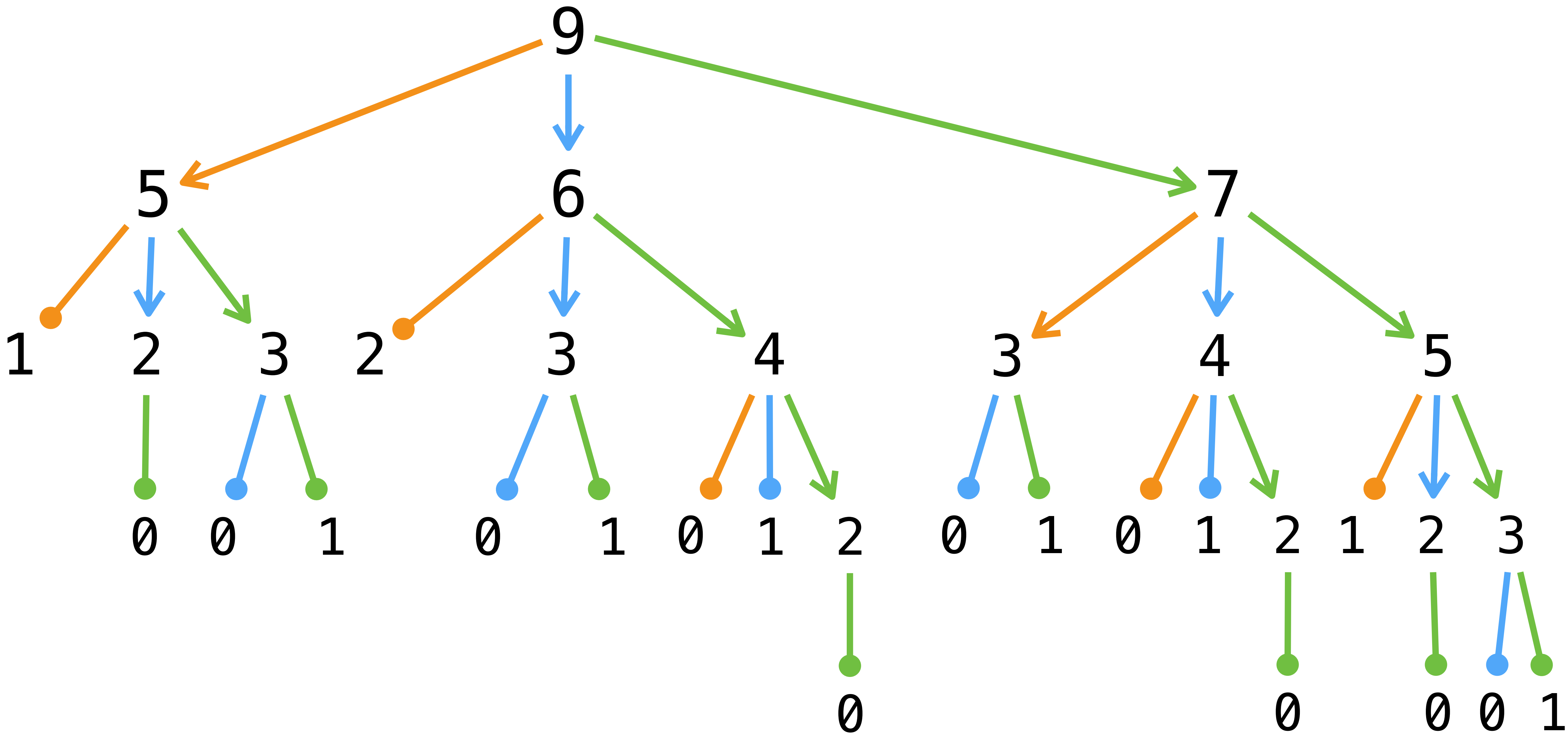
Recap

- start with naive recursive definition
- notice repeated subproblems
- tabulate results
- lookup rather than recurse

Trade space for speed

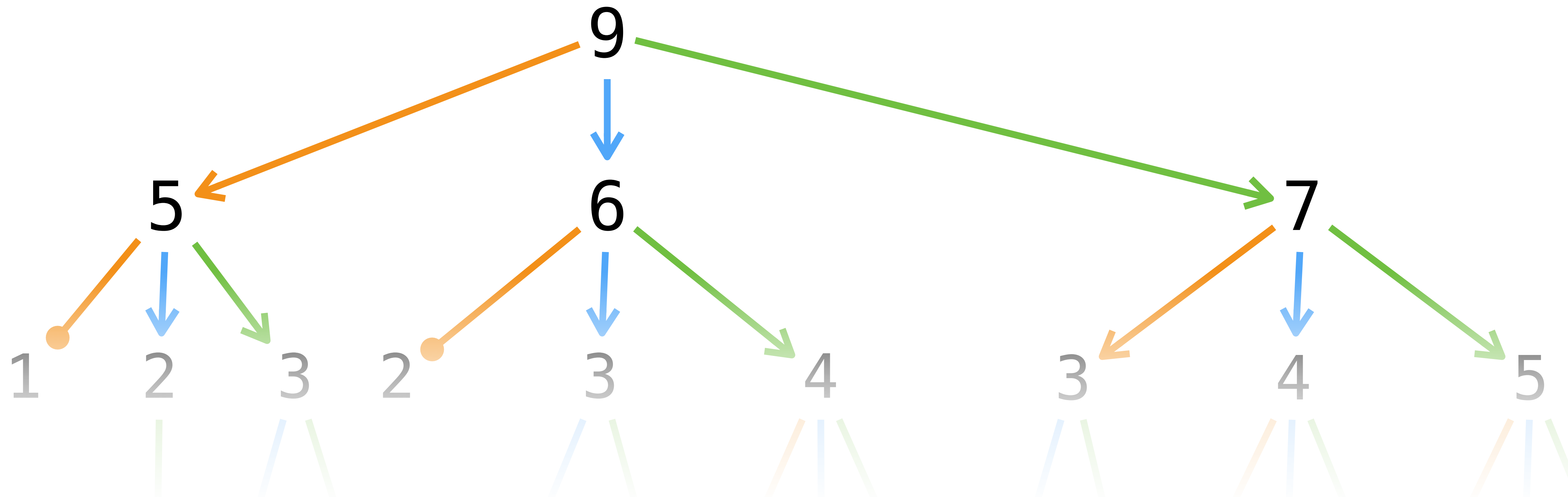
the knapsack problem

	weight	value
gold	4	10
silver	3	6
diamond	2	3



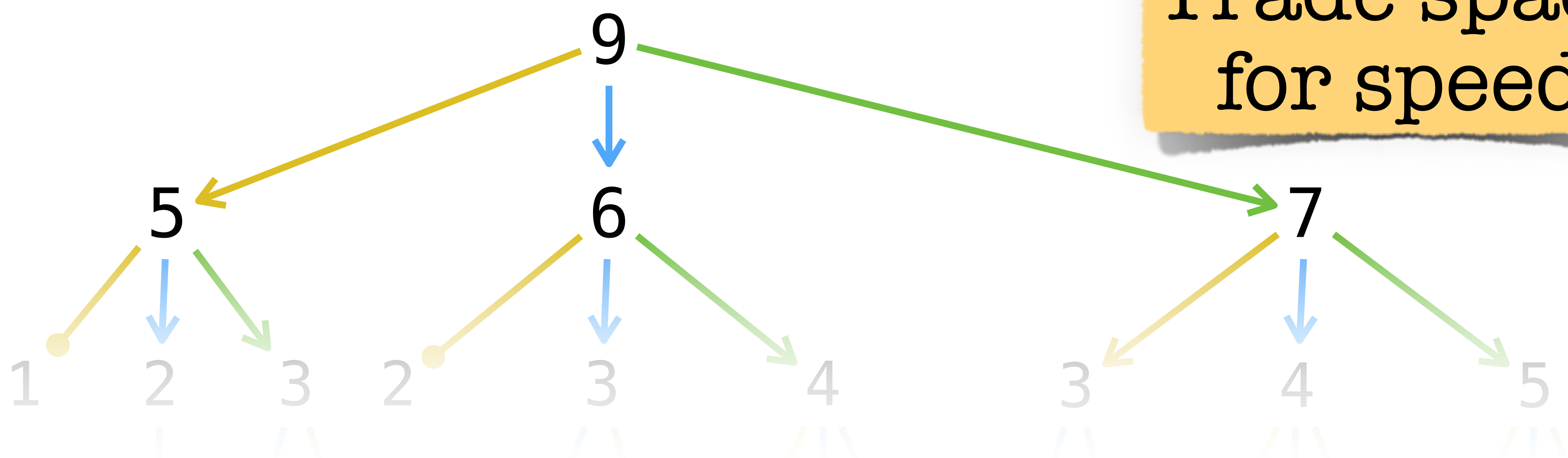
	weight	value
gold	4	10
silver	3	6
diamond	2	3

$$3 + 3 + 3 + 3 = 12$$
$$6 + 6 + 6 = 18$$
$$6 + 3 + 10 = 19$$
$$10 + 10 = 20$$




```
knapsack :: [(Int,Int)] → Int → Int
knapsack wvs c = maximum₀
  [ v + knapsack (c−w) | (w,v) ← wvs
                       , w ≤ c ]
knapsack wvs 9 = maximum₀
  [10 + knapsack 5
  , 6 + knapsack 6
  , 3 + knapsack 7 ]
```

Trade space for speed




```
knapsack :: [(Int,Int)] → Int → Int
knapsack wvs c = maximum0
  [ v + knapsack wvs(c−w) | (w,v) ← wvs
                           , w ≤ c ]
```

Trade space for speed

```
knapsack wvs c = table!c where

             ,

             table!



knapsack :: [(Int,Int)] → Int → Int
knapsack wvs c = maximum₀
  [ v + knapsack (c−w) | (w,v) ← wvs
                         , w ≤ c ]
```

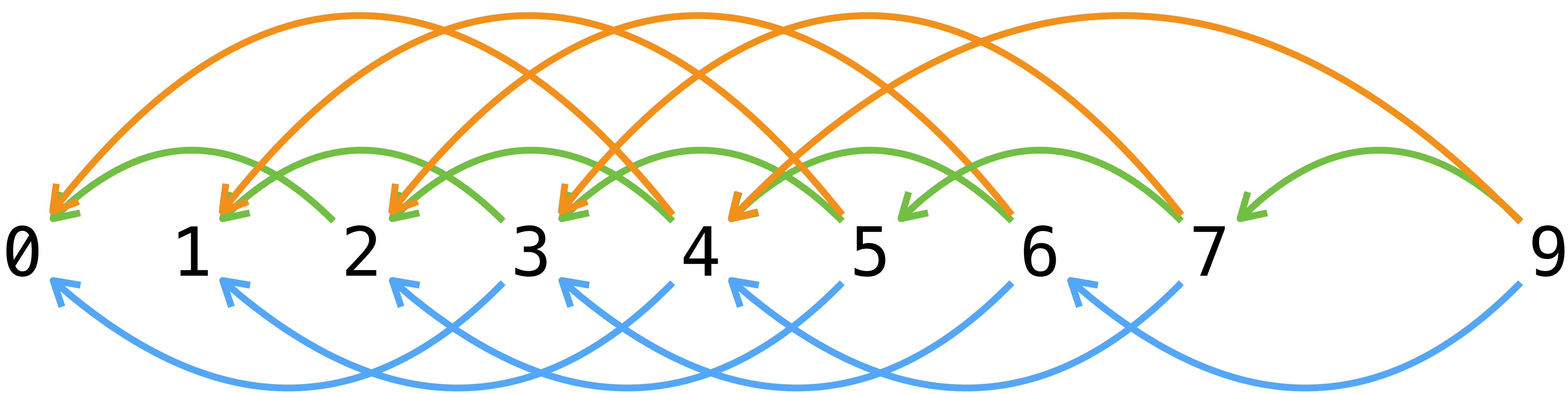
Trade space for speed

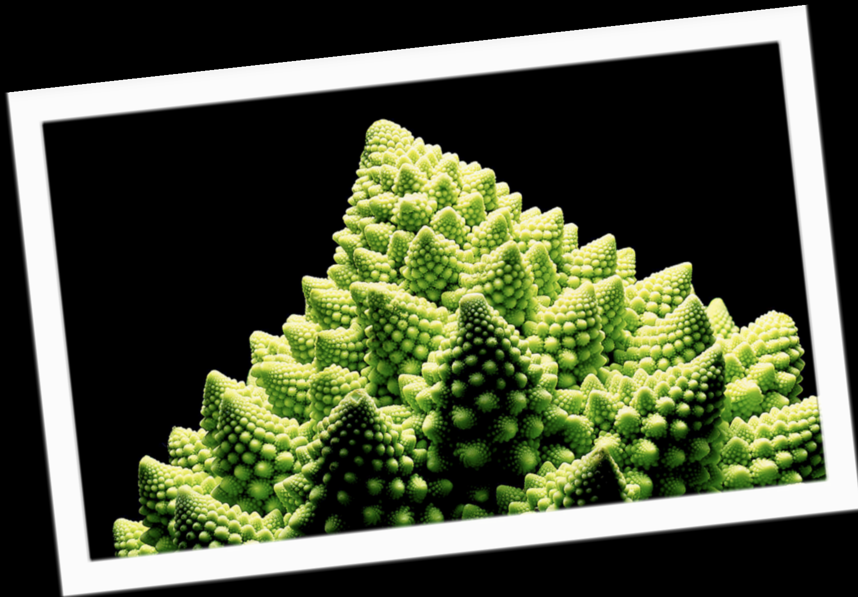
```
knapsack :: [(Int,Int)] → Int → Int
knapsack wvs c = table!c where
  table = tabulate (0,c) knapsack’
  knapsack’ = maximum₀
    [ v + table!(c−w) | (w,v) ← wvs
                      , w ≤ c ]
```

```
knapsack :: [(Int,Int)] → Int → Int
knapsack wvs c = maximum₀
  [ v + knapsack (c−w) | (w,v) ← wvs
                       , w ≤ c ]
```

Trade space for speed

```
knapsack :: [(Int,Int)] → Int → Int
knapsack wvs c = table!c where
  table = tabulate (0,c) knapsack'
  knapsack' = maximum₀
    [ v + table!(c−w) | (w,v) ← wvs
                      , w ≤ c ]
```





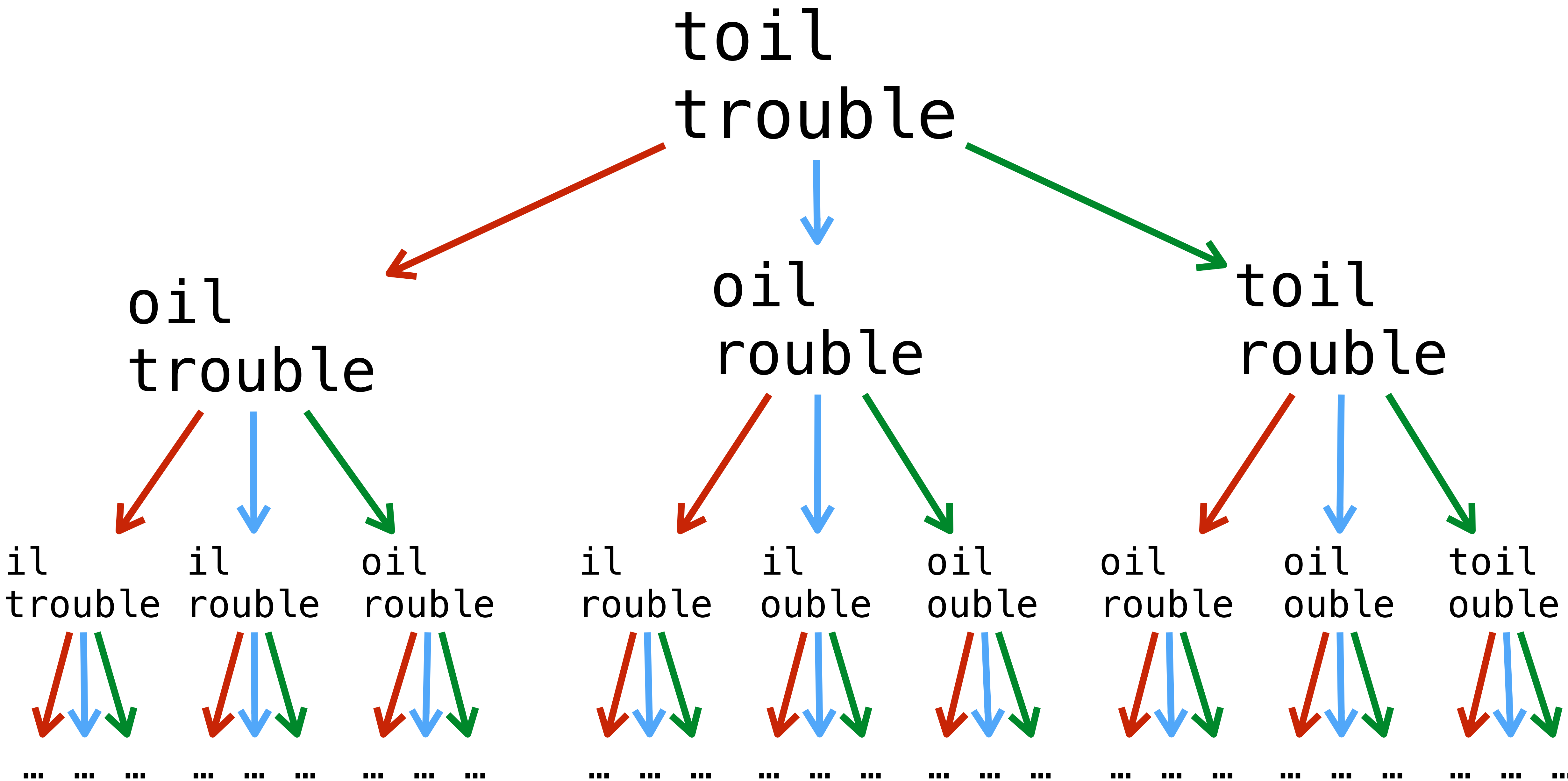
Recap

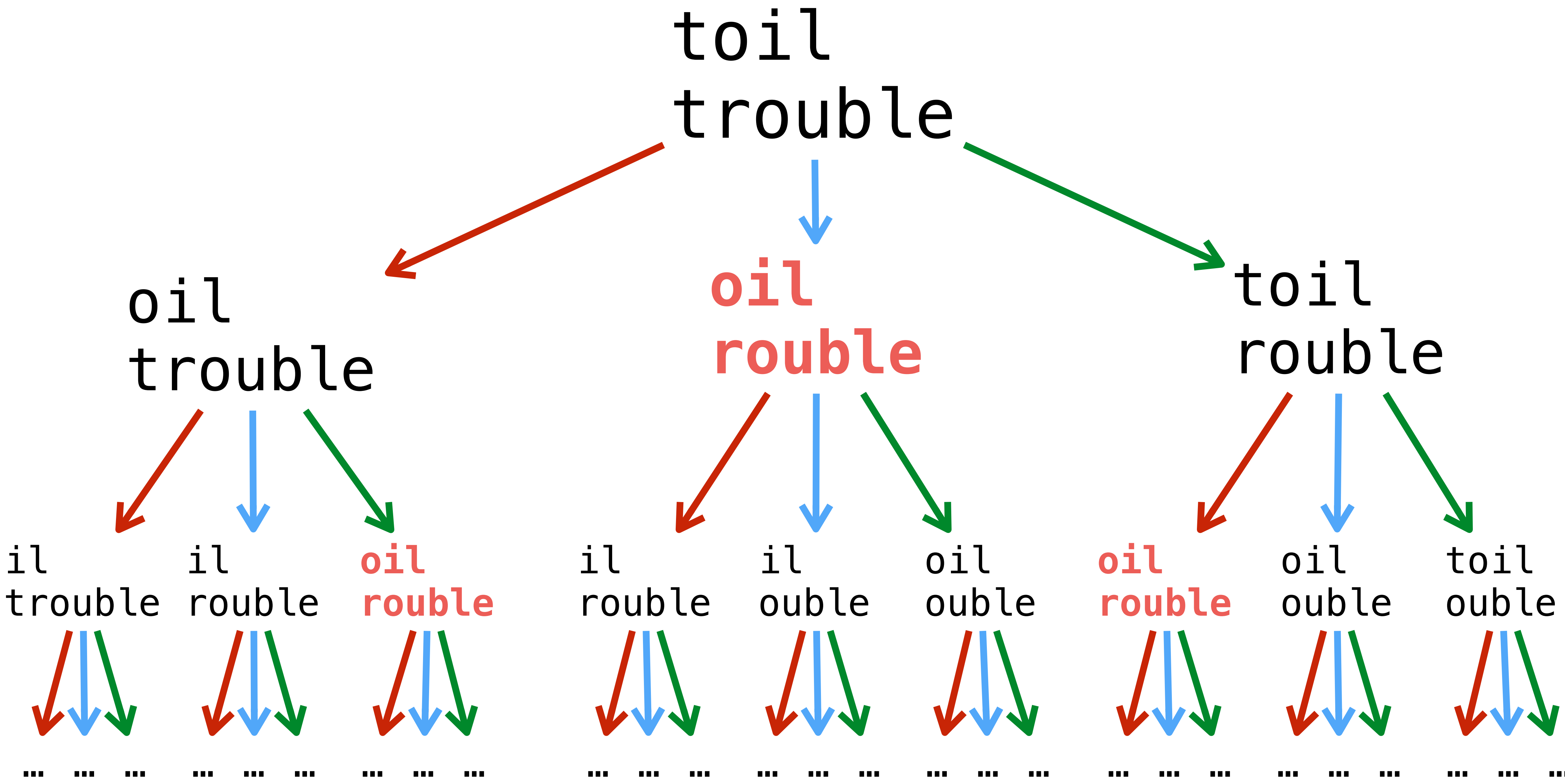
- start with naive recursive definition
- notice repeated subproblems
- tabulate results
- lookup rather than recurse

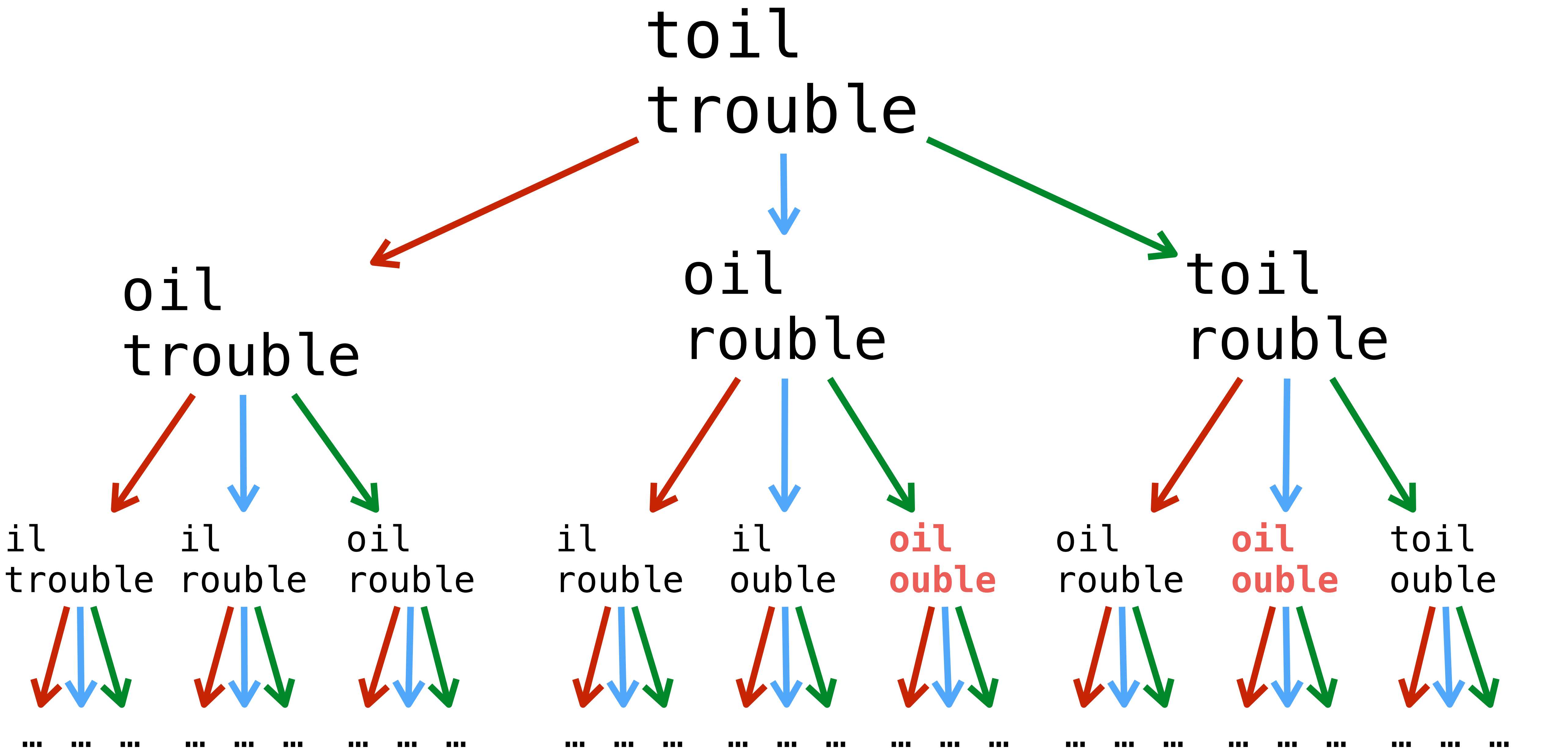
Trade space for speed

the edit-distance problem

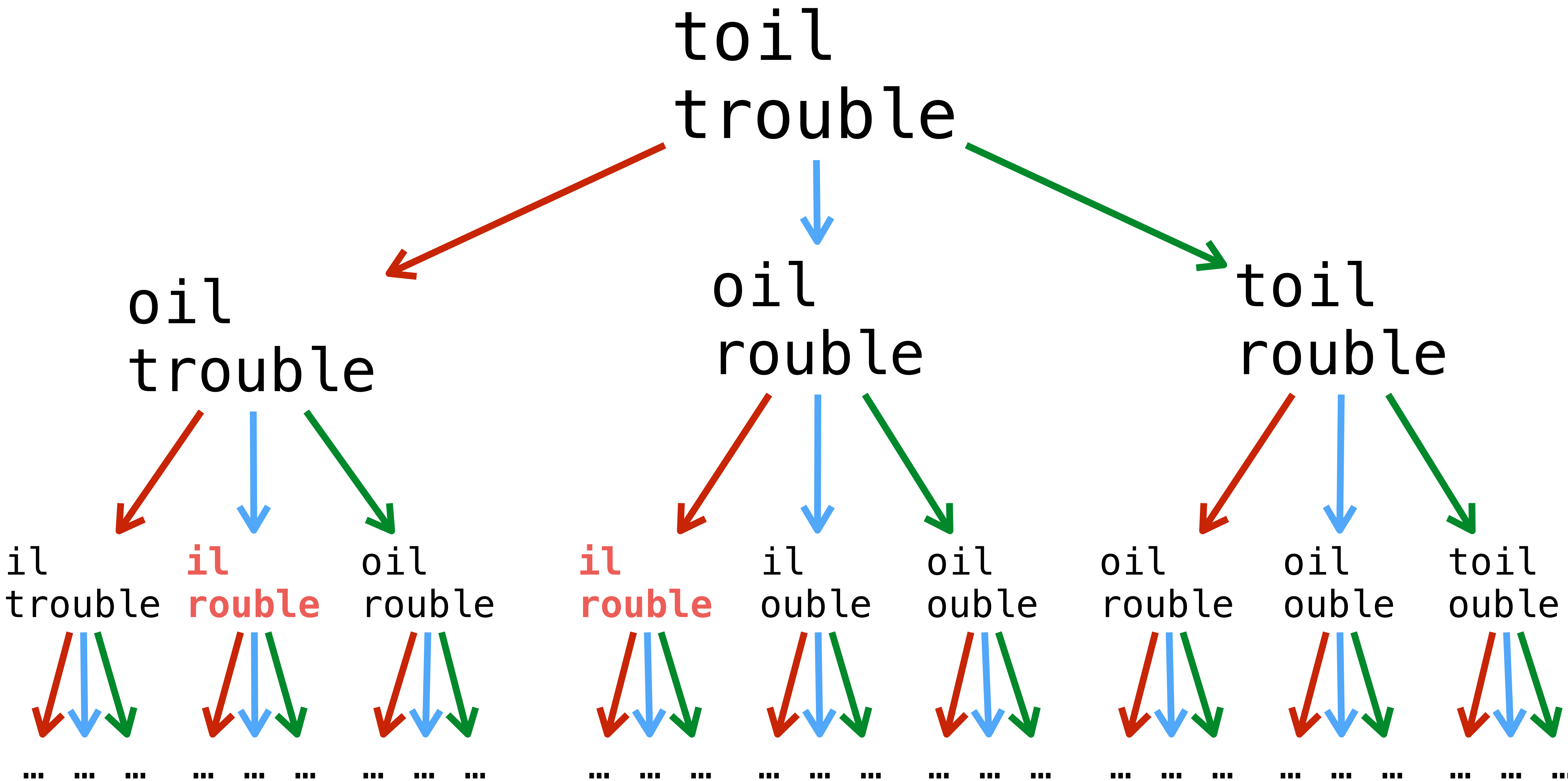
toil	trouble
troil	touble
troul	toible
troubl	toile
trouble	toil

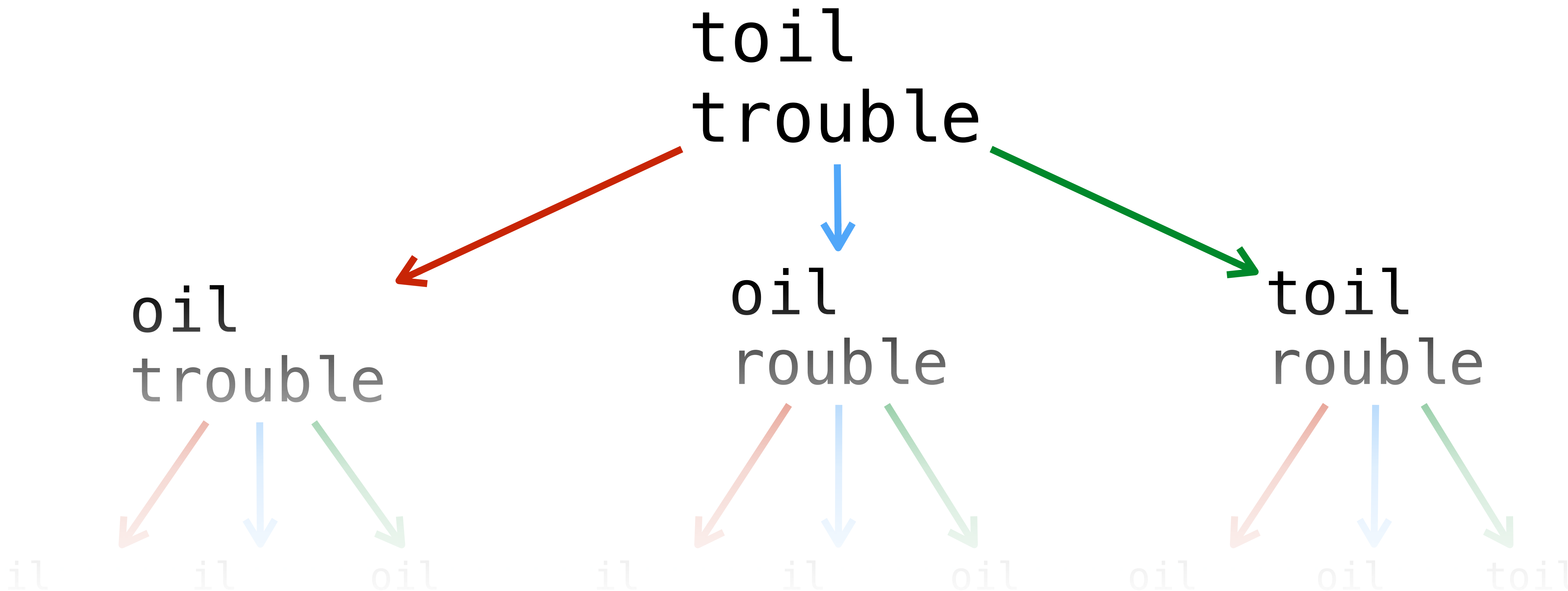

toil trouble
oil trouble
oil rouble
toil rouble
il trouble
il rouble
oil rouble
il rouble
il ouble
oil ouble
oil rouble
oil ouble
toil ouble
… … …


toil
trouble
oil
trouble
oil
rouble
toil
rouble
il
trouble
il
rouble
oil
rouble
il
rouble
il
ouble
oil
ouble
oil
rouble
oil
ouble
toil
ouble
… … …
… … …
… … …
… … …
… … …
… … …
… … …
… … …
… … …

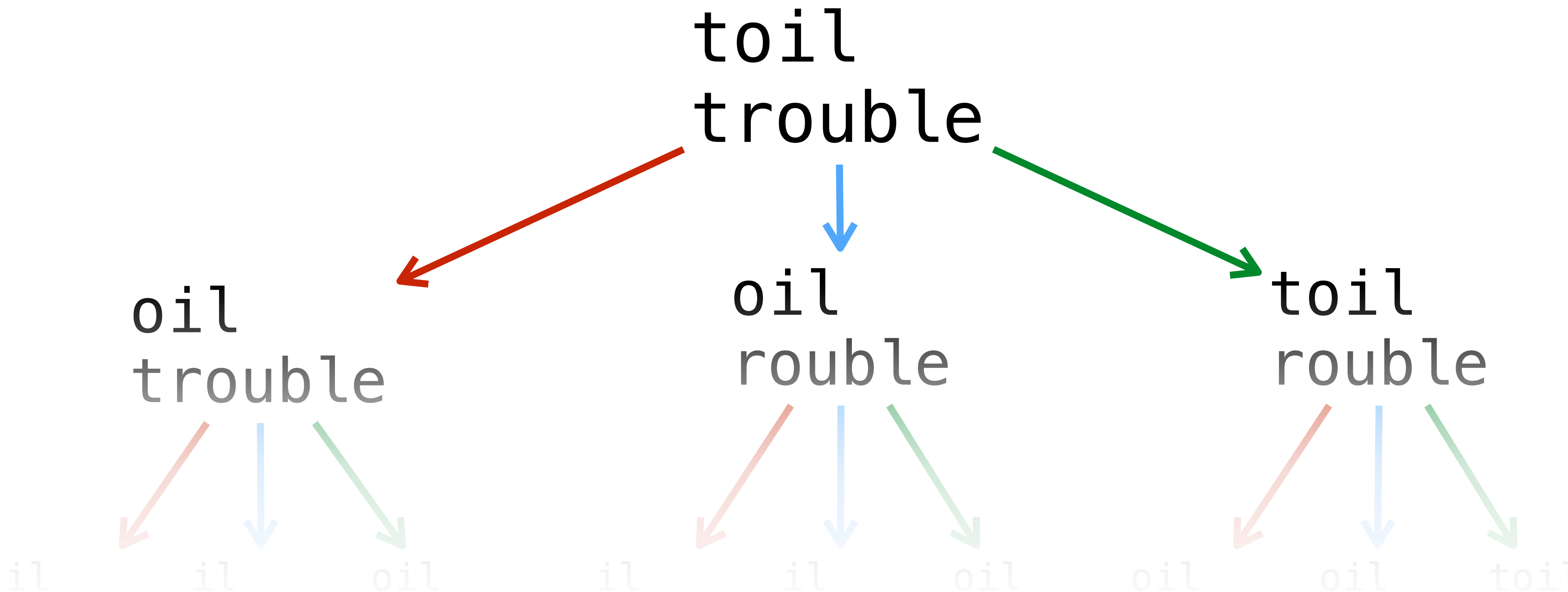


toil trouble
oil trouble
oil rouble
toil rouble
il trouble
il rouble
oil rouble
il rouble
il ouble
oil ouble
oil rouble
oil ouble
toil ouble


toil
trouble
oil
trouble
oil
rouble
toil
rouble
il
trouble
il
rouble
oil
rouble
il
rouble
il
ouble
oil
ouble
oil
rouble
oil
ouble
toil
ouble



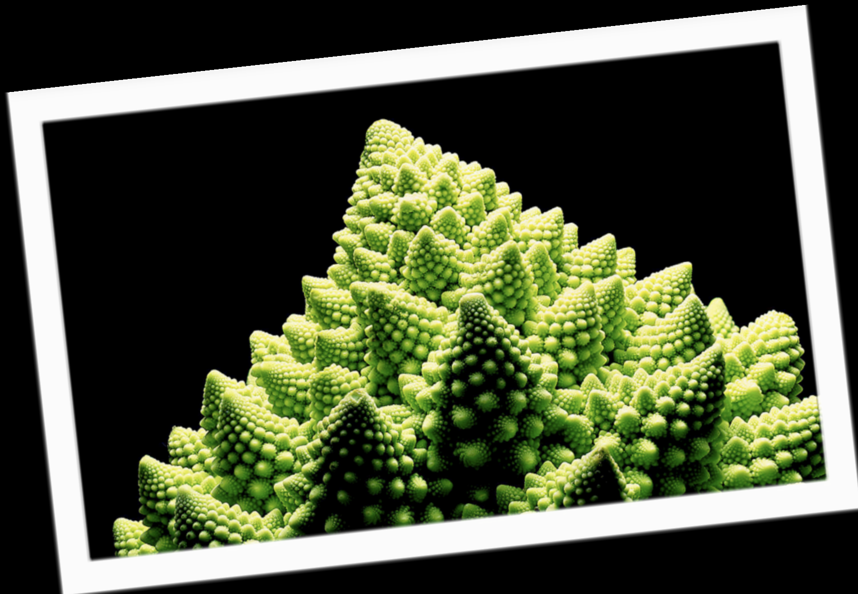

```
dist :: String → String → Int
dist xs     []     = length xs
dist []     ys     = length ys
dist (x:xs) (y:ys) = minimum
  [ dist xs (y:ys) + 1, dist (x:xs) ys + 1
  , dist xs ys + if x == y then 0 else 1 ]
```




```
dist :: String → String → Int
dist xs      []      = length xs
dist []      ys      = length ys
dist (x:xs) (y:ys) = minimum
  [ dist xs (y:ys) + 1, dist (x:xs) ys + 1
  , dist xs ys + if x == y then 0 else 1 ]
```

```
dist :: String → String → Int
dist xs ys = table!mn where
 mn    = (length xs, length ys)
 table = tabulate ((0,0), mn) dist’
 dist’ i     0     = i
 dist’ 0     j     = j
 dist’ (i+1) (j+1) = minimum
  [ table!(i,j+1) + 1, table!(i+1,j) + 1
  , table!(i,j) + if xs!i == ys!j then 0 else 1 ]
```

```
dist :: String → String → Int
dist xs     []     = length xs
dist []     ys     = length ys
dist (x:xs) (y:ys) = minimum
  [ dist xs (y:ys) + 1, dist (x:xs) ys + 1
  , dist xs ys + if x == y then 0 else 1 ]
```



Recap

- start with naive recursive definition
- notice repeated subproblems
- tabulate results
- lookup rather than recurse

Trade space for speed