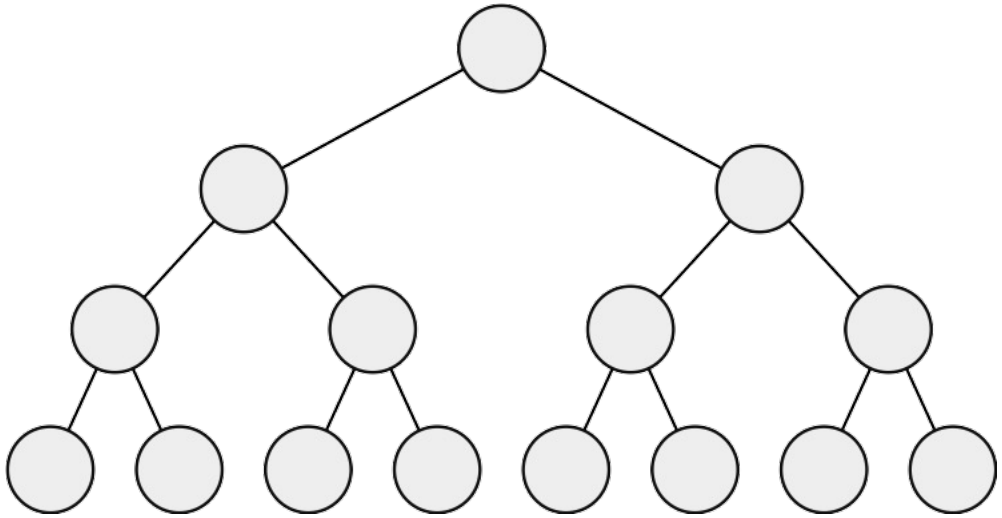
Competitive Programming

Memoization

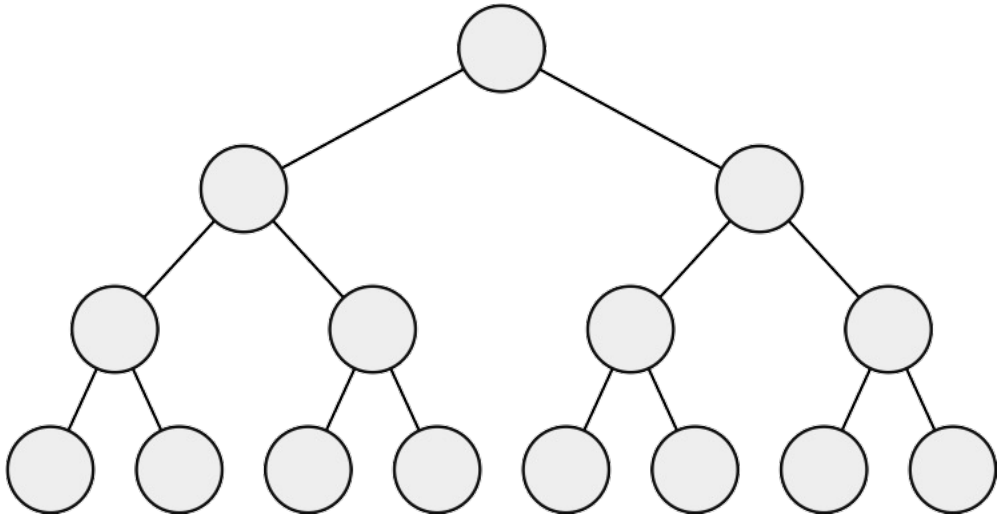
Duplicate Subproblems

- Often, we can solve a problem by breaking it down into smaller subproblems.
  - Dynamic programming is like this, where the subproblems are smaller, simplified problem instances.
  - Search is like this, where the subproblems involve trying different partial solutions.
- Solving a problem can look like traversing or exploring a tree.



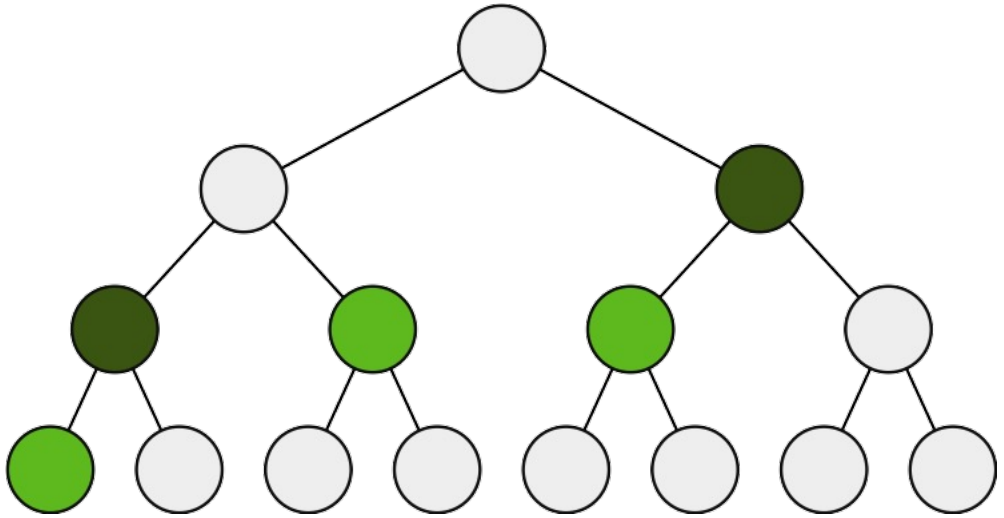
Duplicate Subproblems

- Often, we can solve a problem by breaking it down into smaller subproblems.
  - Dynamic programming is like this, where the subproblems are smaller, simplified problem instances.
  - Search is like this, where the subproblems involve trying different partial solutions.
- Solving a problem can look like traversing or exploring a tree.



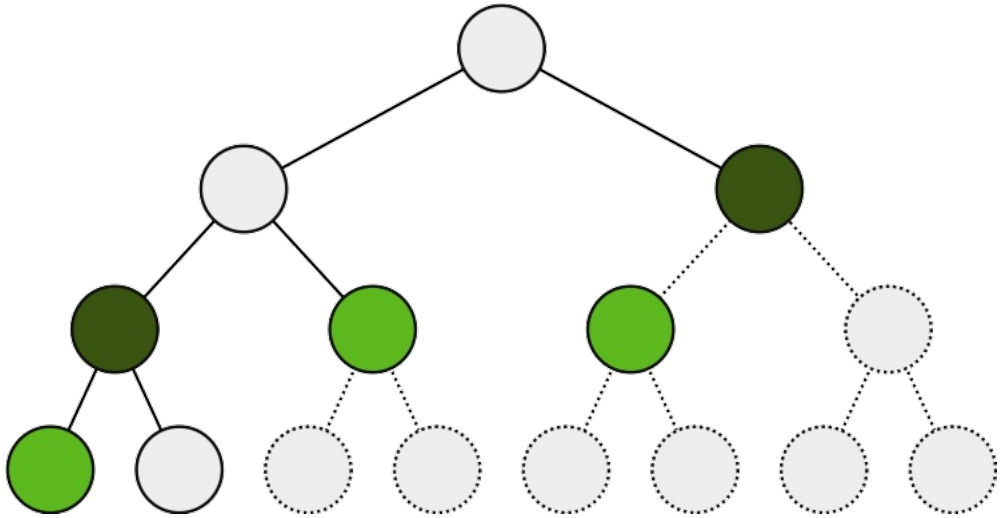
Duplicate Subproblems

- The same subproblem can show up along multiple branches.



Duplicate Subproblems

- The same subproblem can show up along multiple branches.
- Saving a partial solution could avoid duplicate work.
- In some cases, it could turn an exponential-time solution into polynomial.



Fibonacci Numbers Recursively

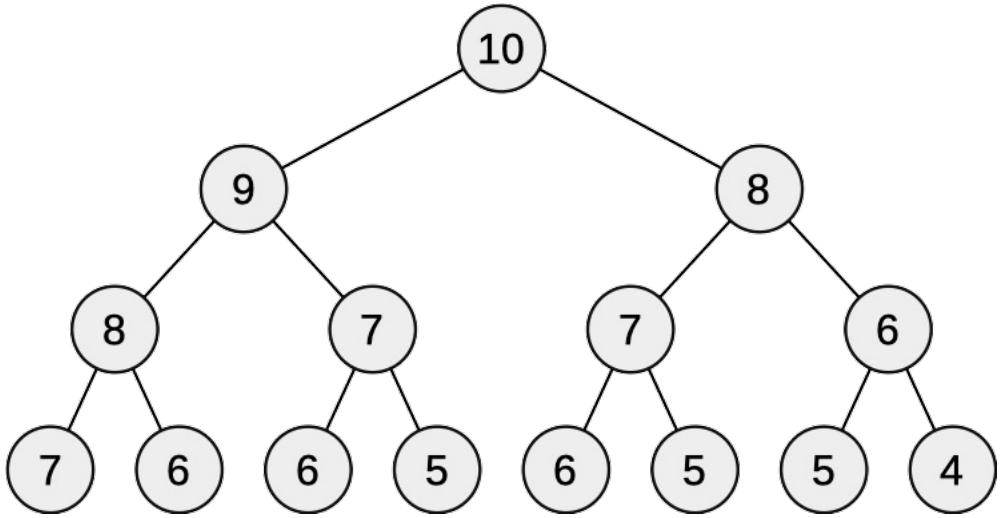
- It's easy to define Fibonacci numbers recursively.

```
int fib( n )
   if ( n < 2 )
      return 1
   return fib( n-1 ) + fib( n-2 )
```

To compute a Fibonacci number ..

Compute the previous two Fibonacci numbers.

- But this is a bad way to implement it.



Recursive with a Cache

- A cache of previous solutions would speed this up a lot.

```
map< int, int > cache;

int fib( n )
   if ( n < 2 )
      return 1
   if ( cache.containsKey( n ) )
      return cache[ n ];
   int val = fib( n-1 ) + fib( n-2 );
   cache[ n ] = val;
   return val;
}
```

Let me check my cache.

Save this solution for the future.