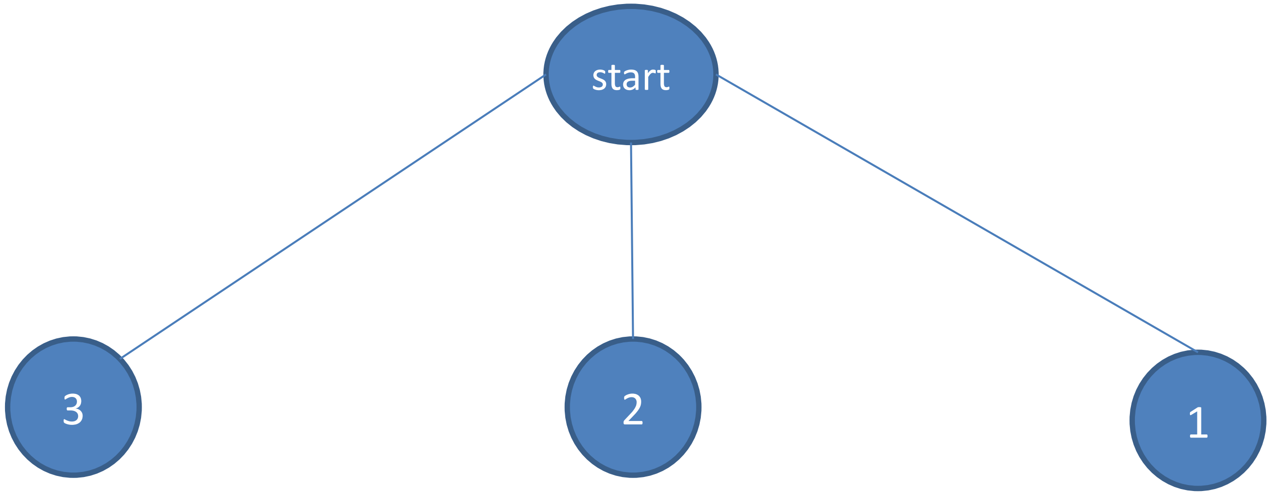
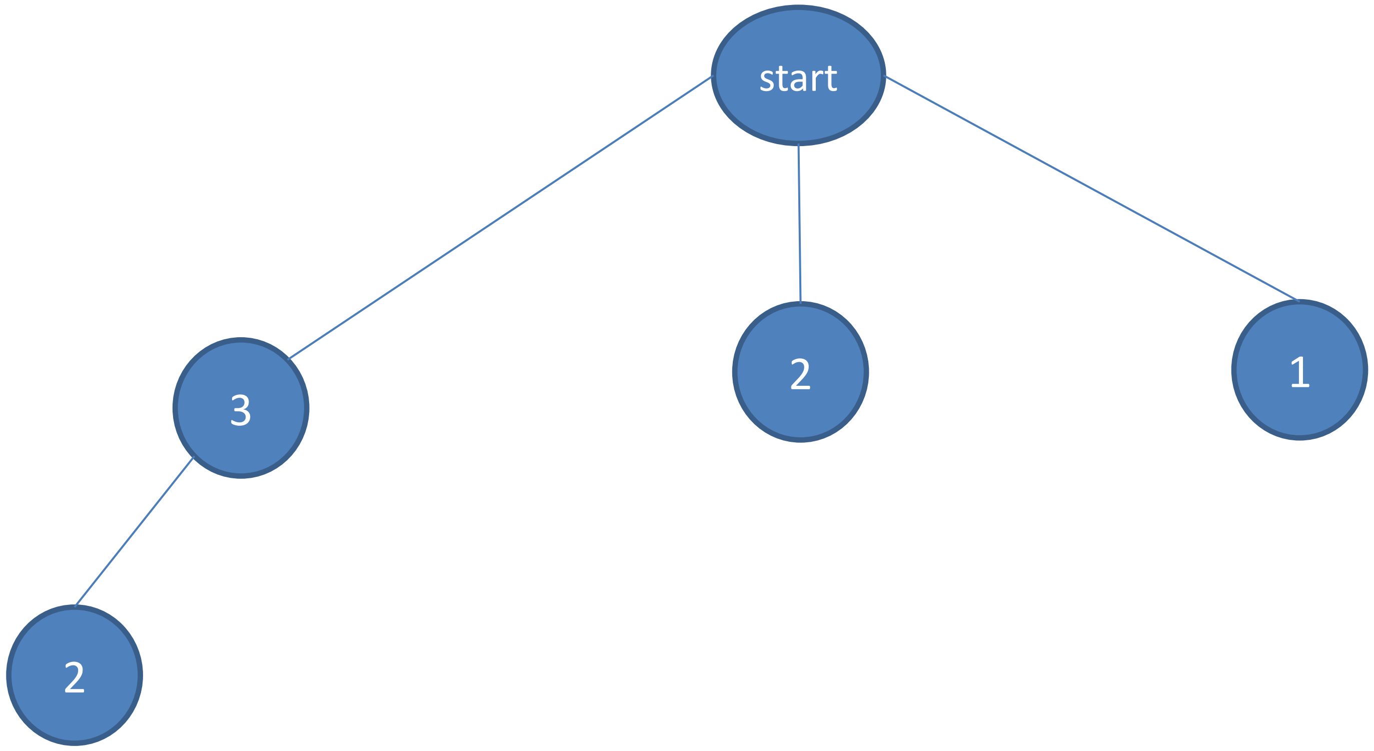
BACKTRACKING & TOWER OF HANOI

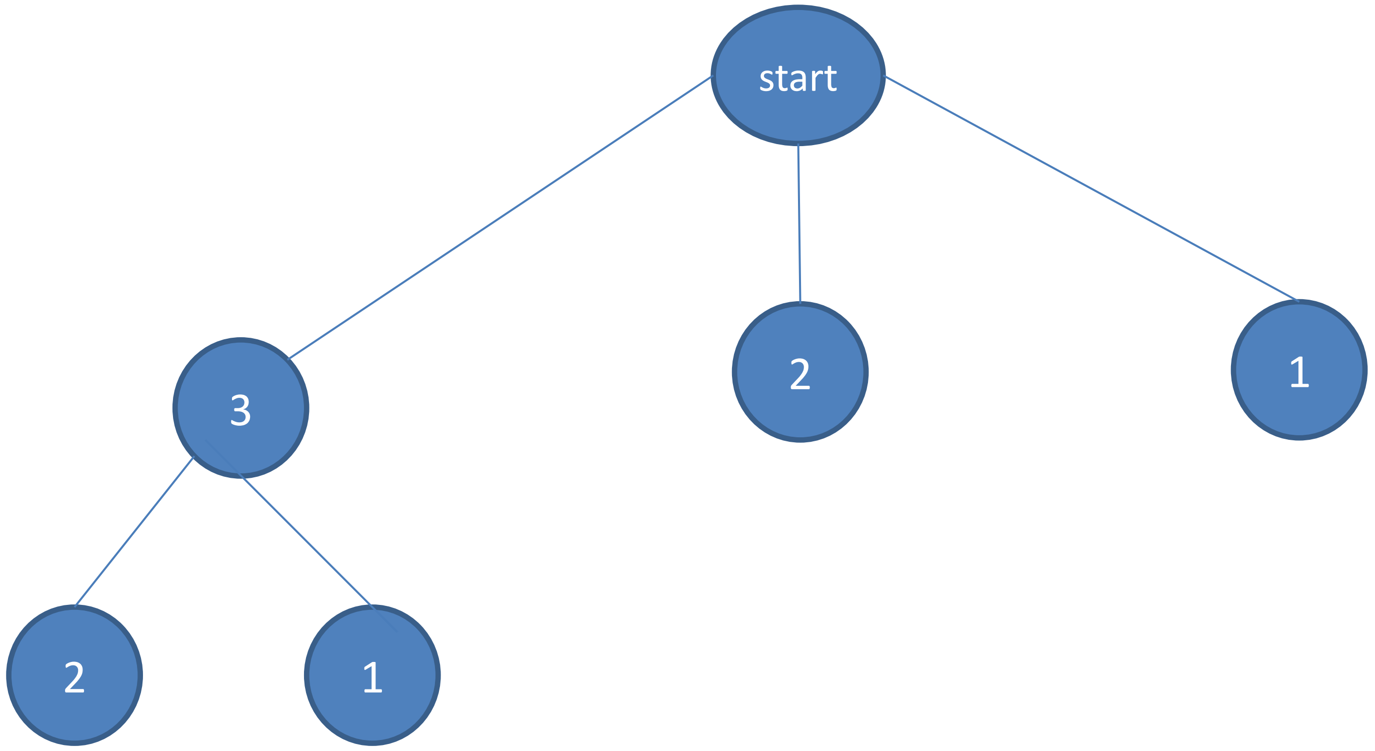
Permutation



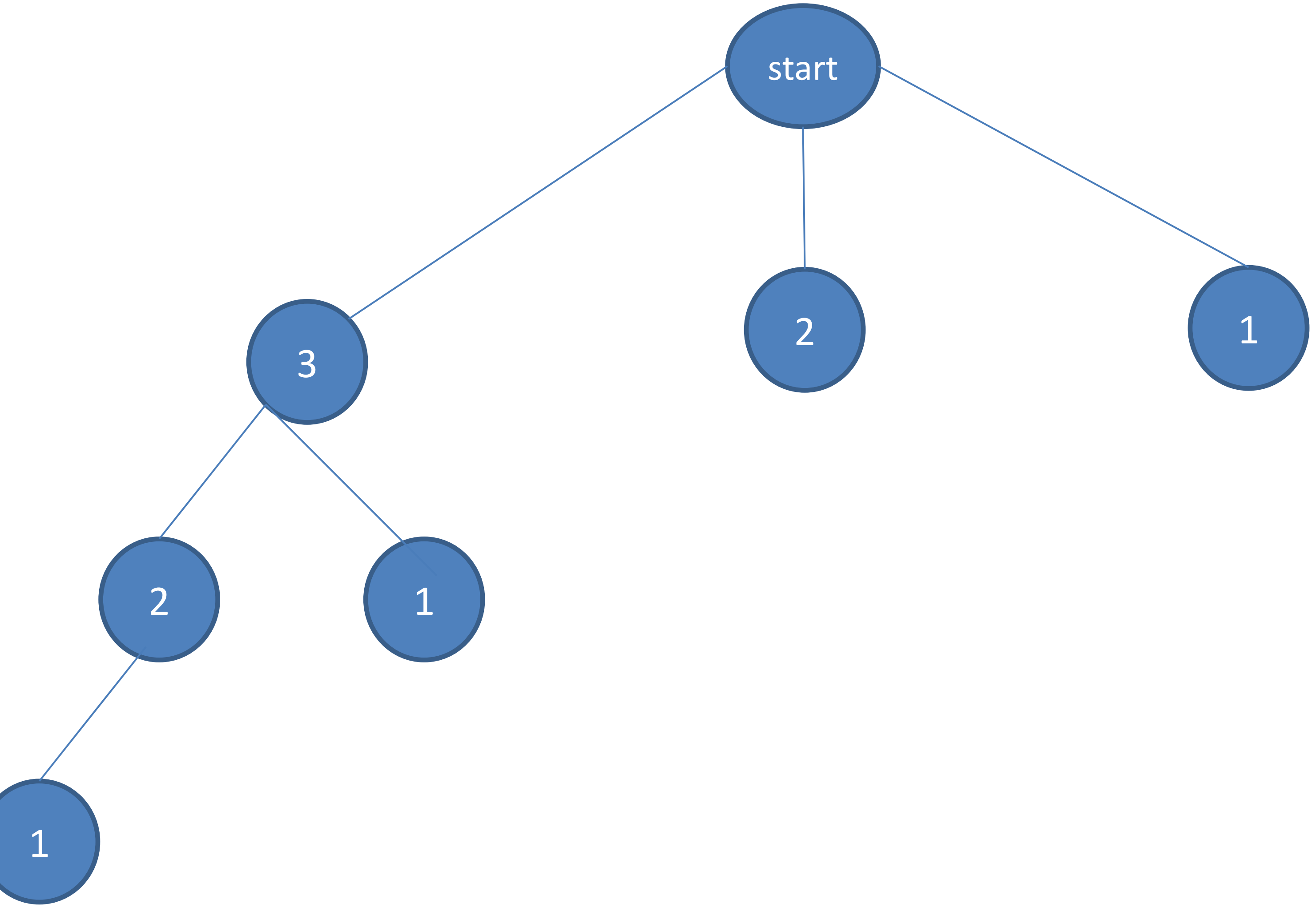
Permutation



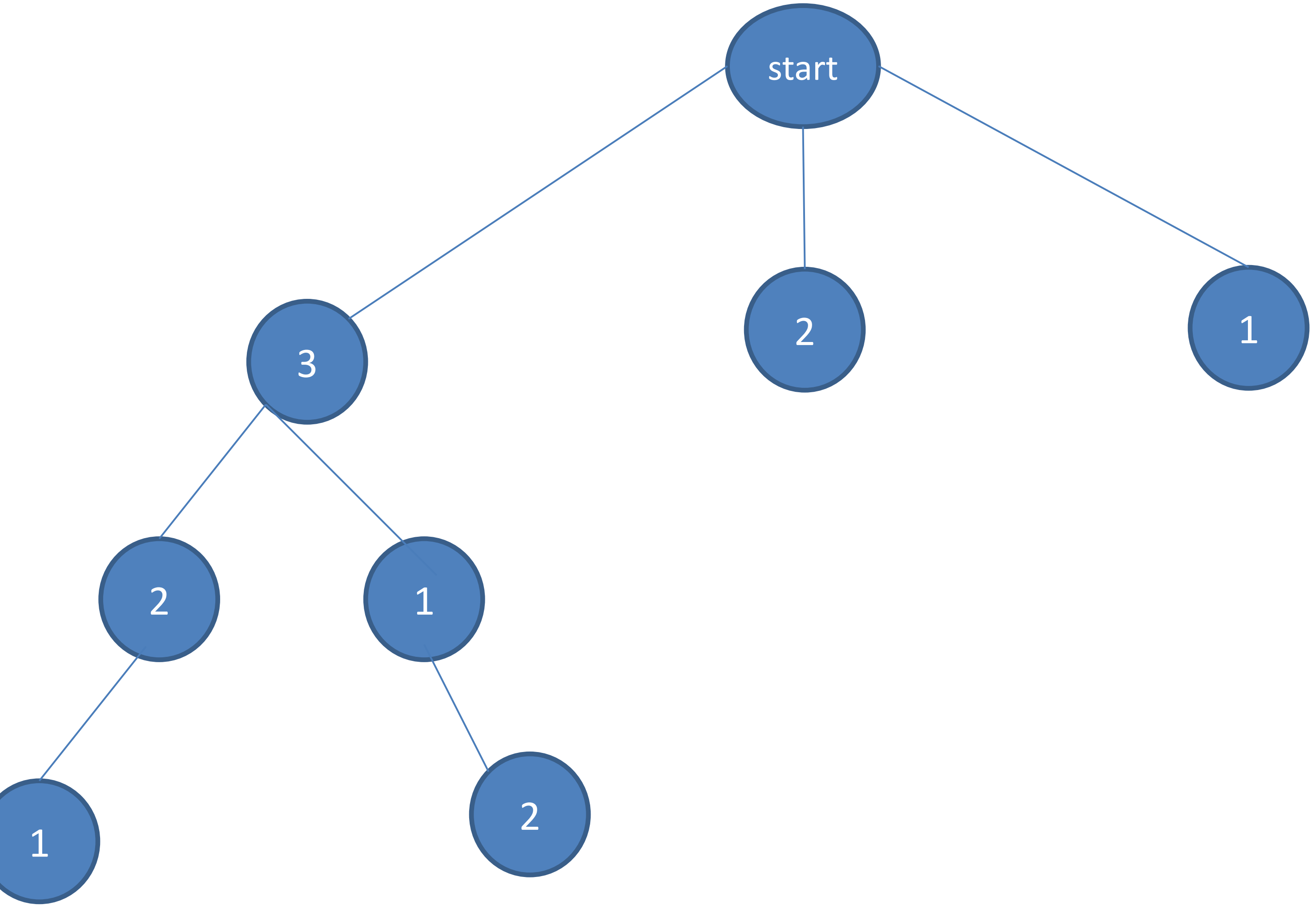
Permutation



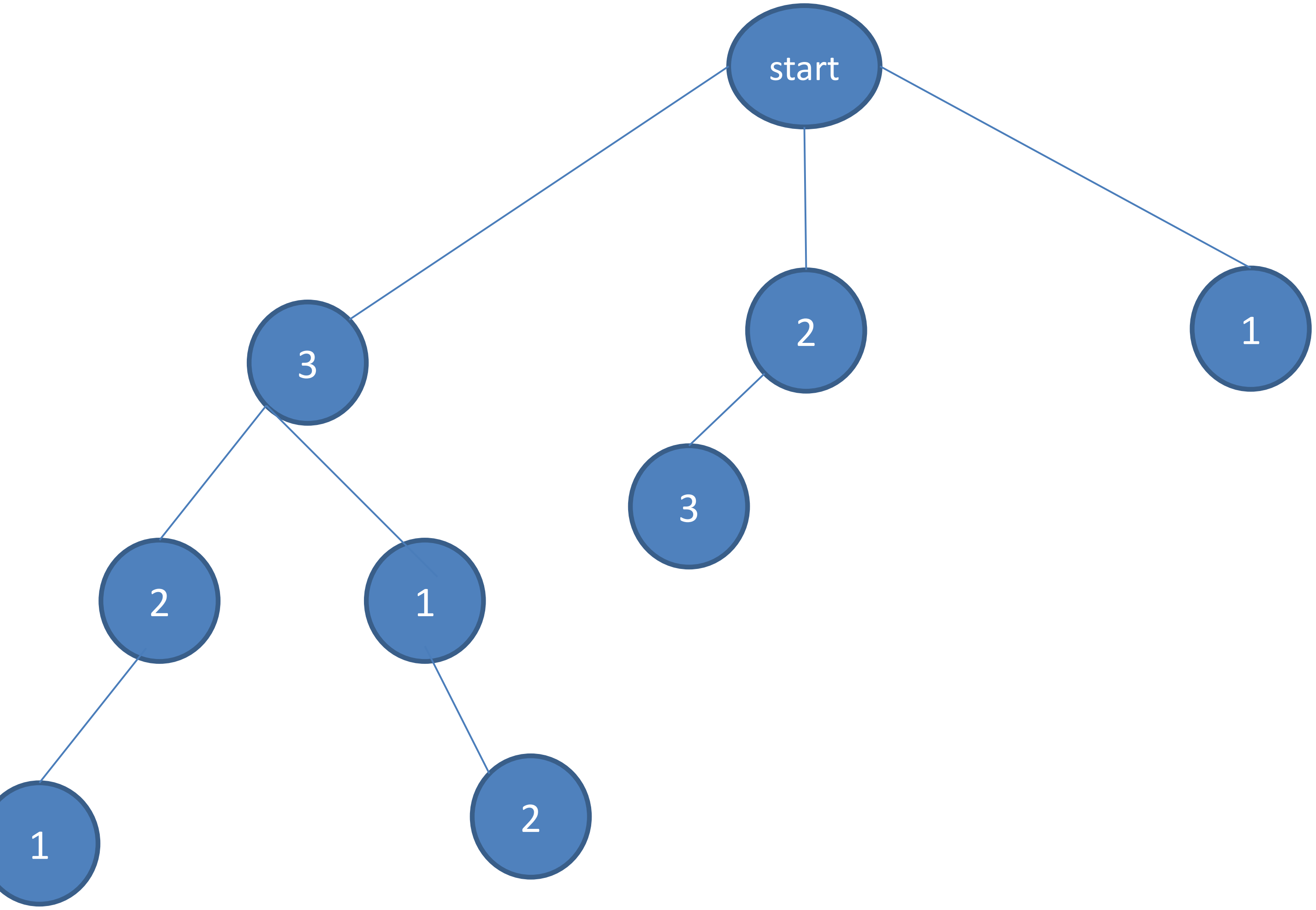
Permutation



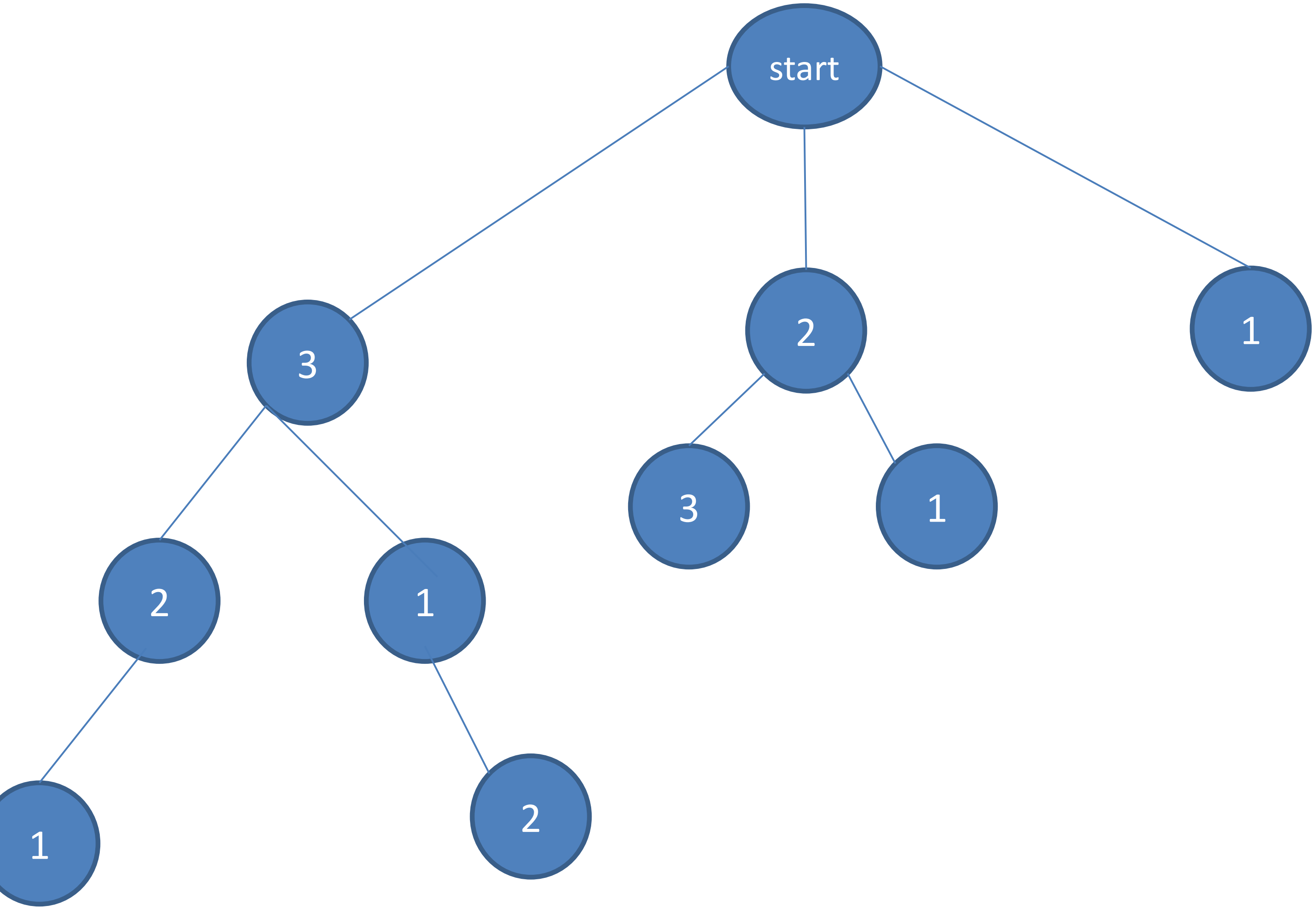
Permutation



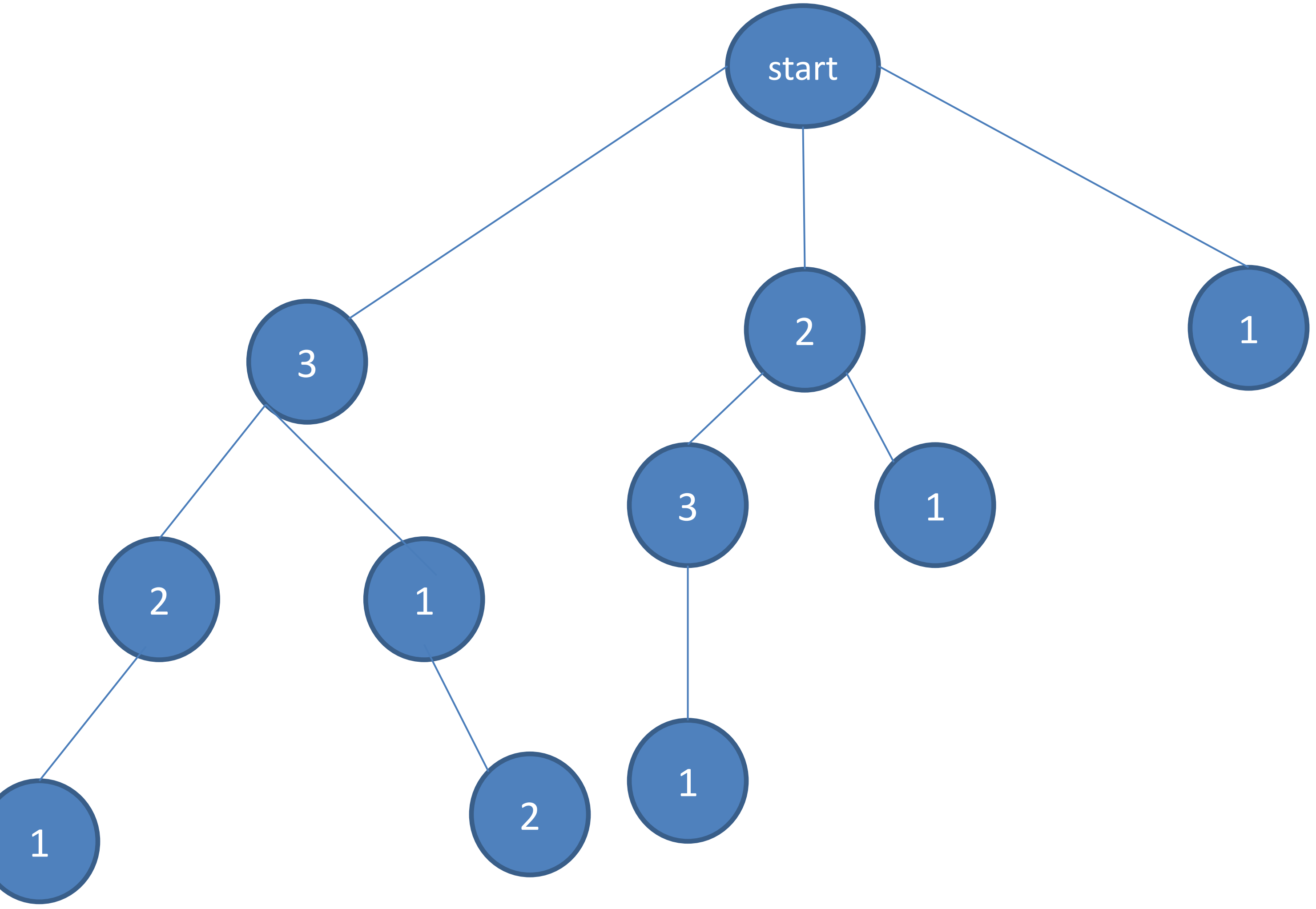
Permutation



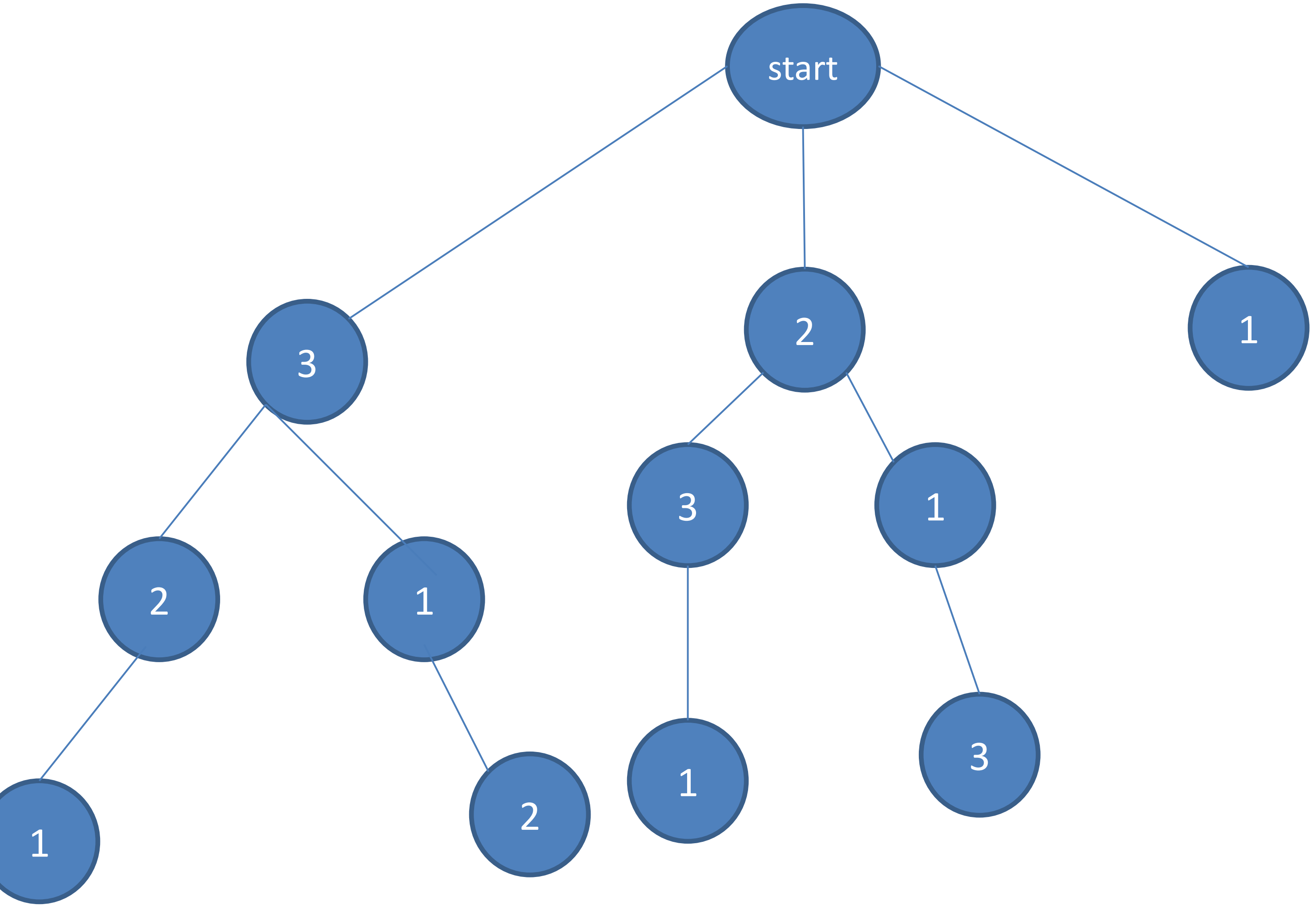
Permutation



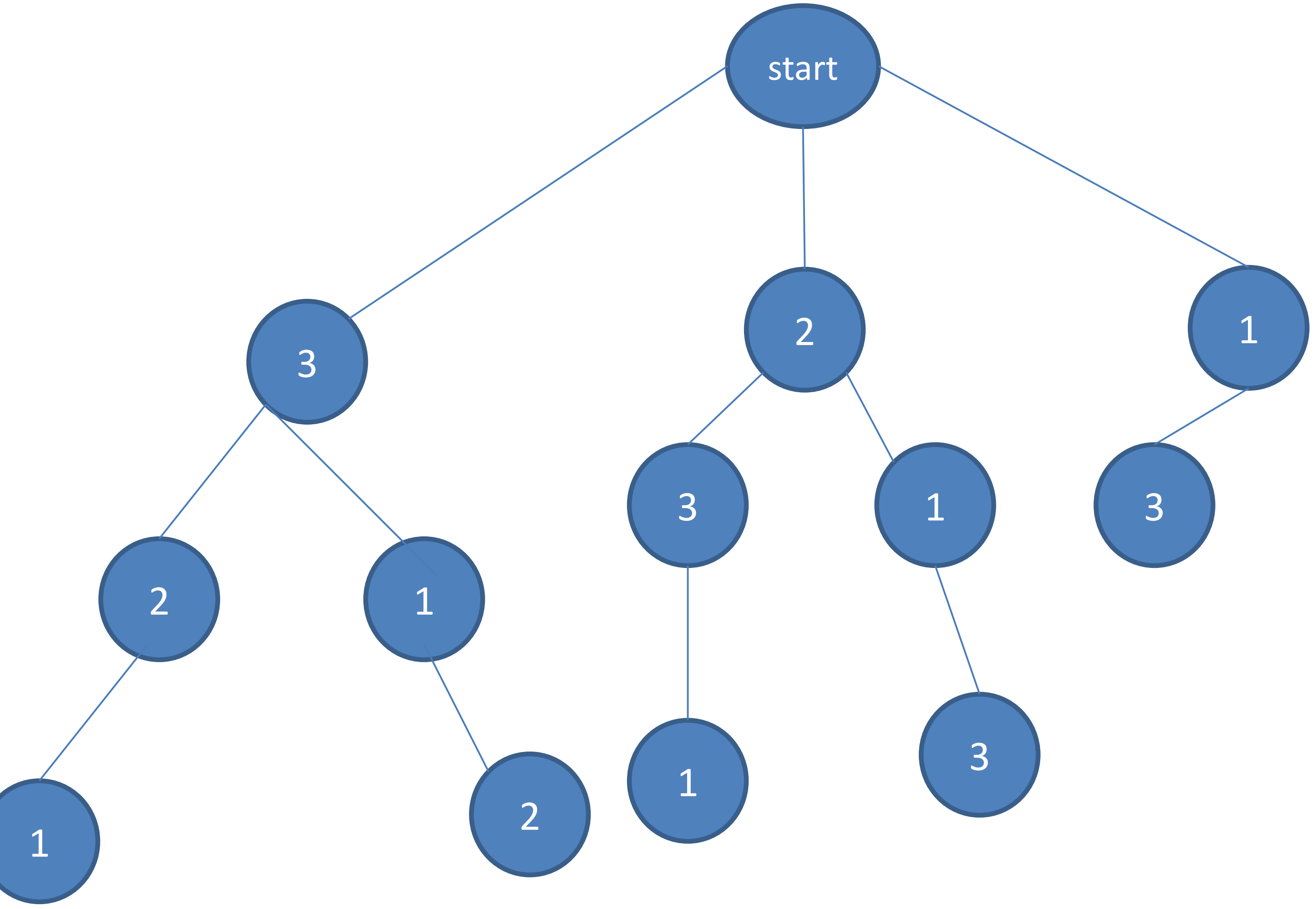
Permutation



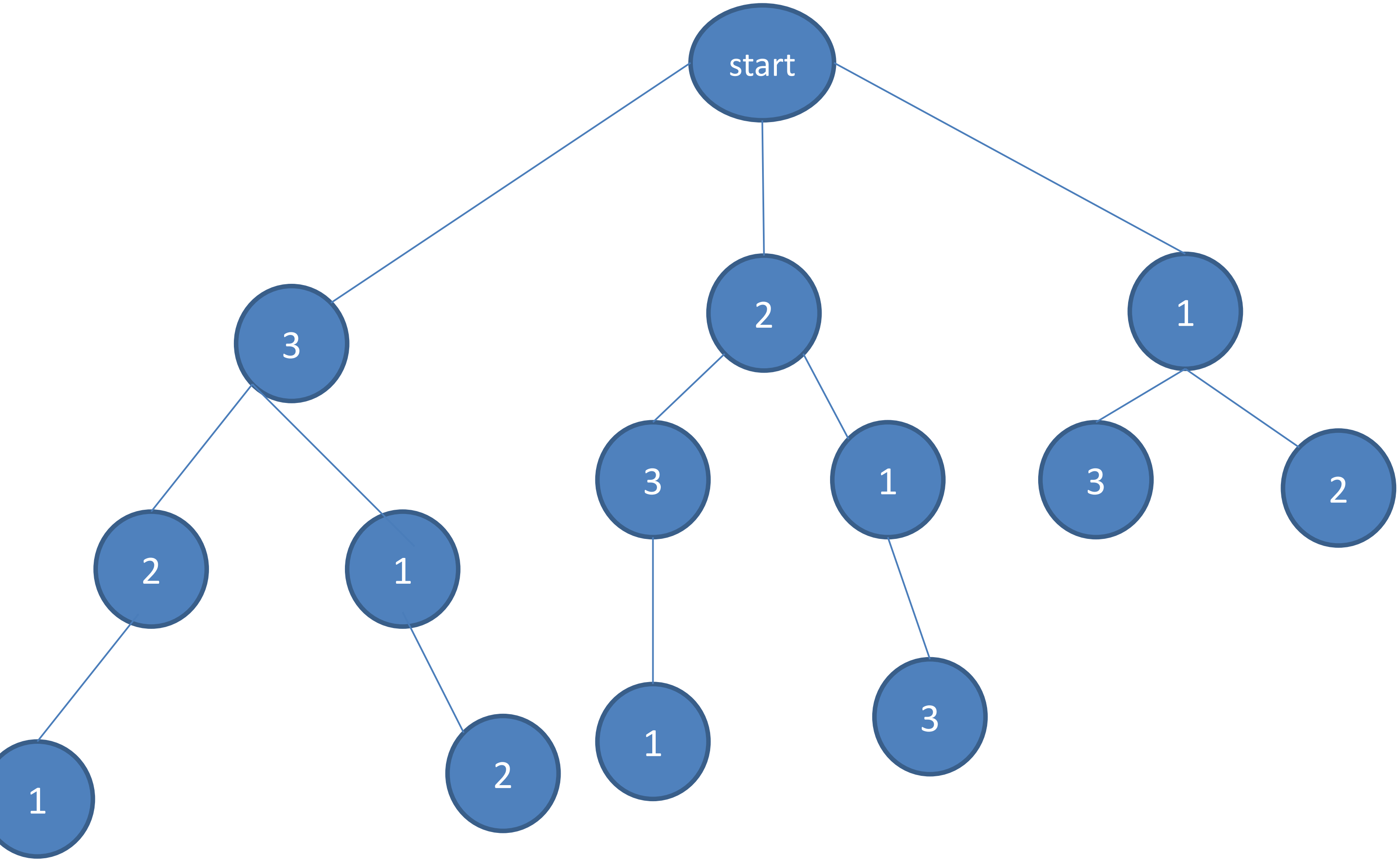
Permutation



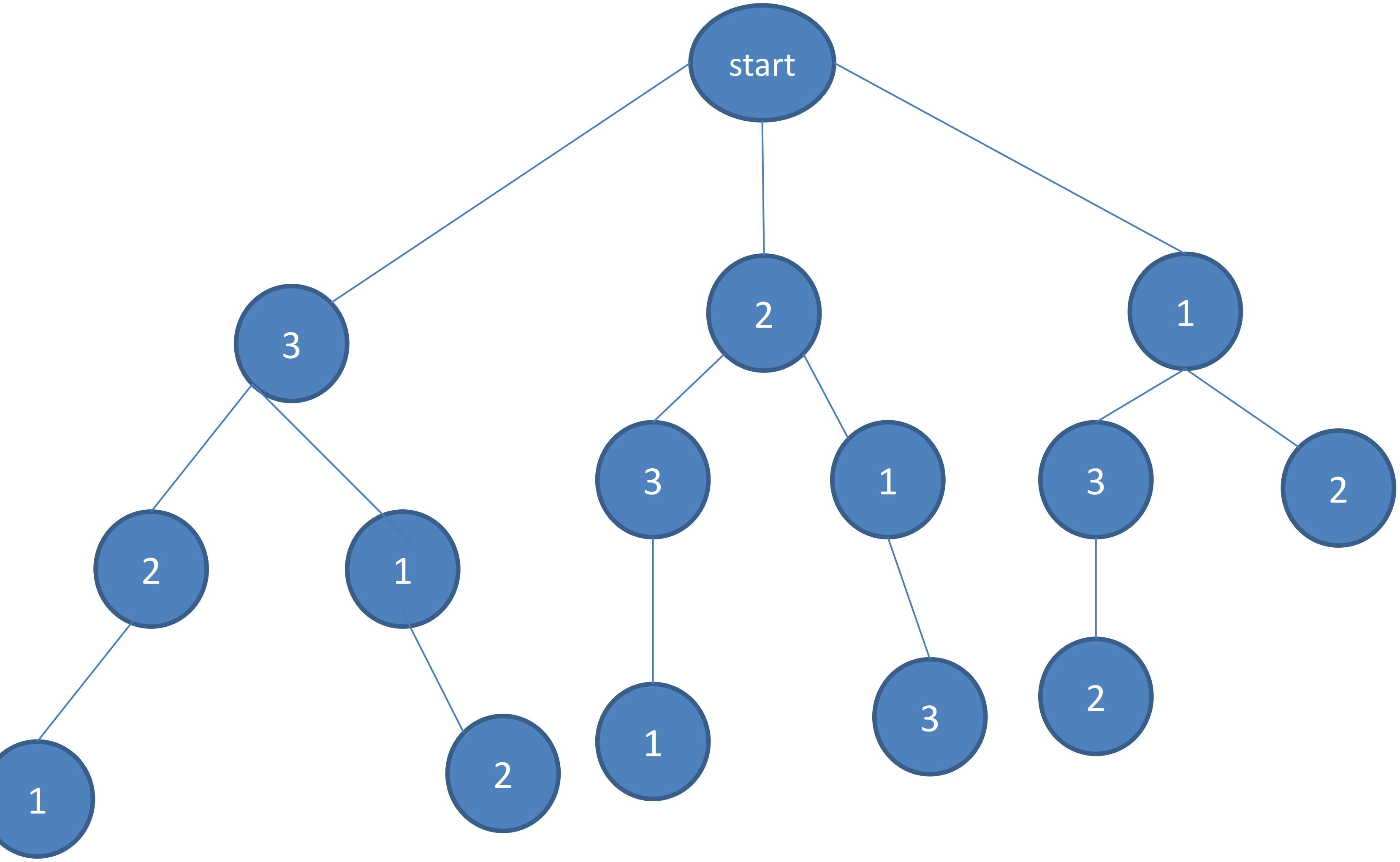
Permutation



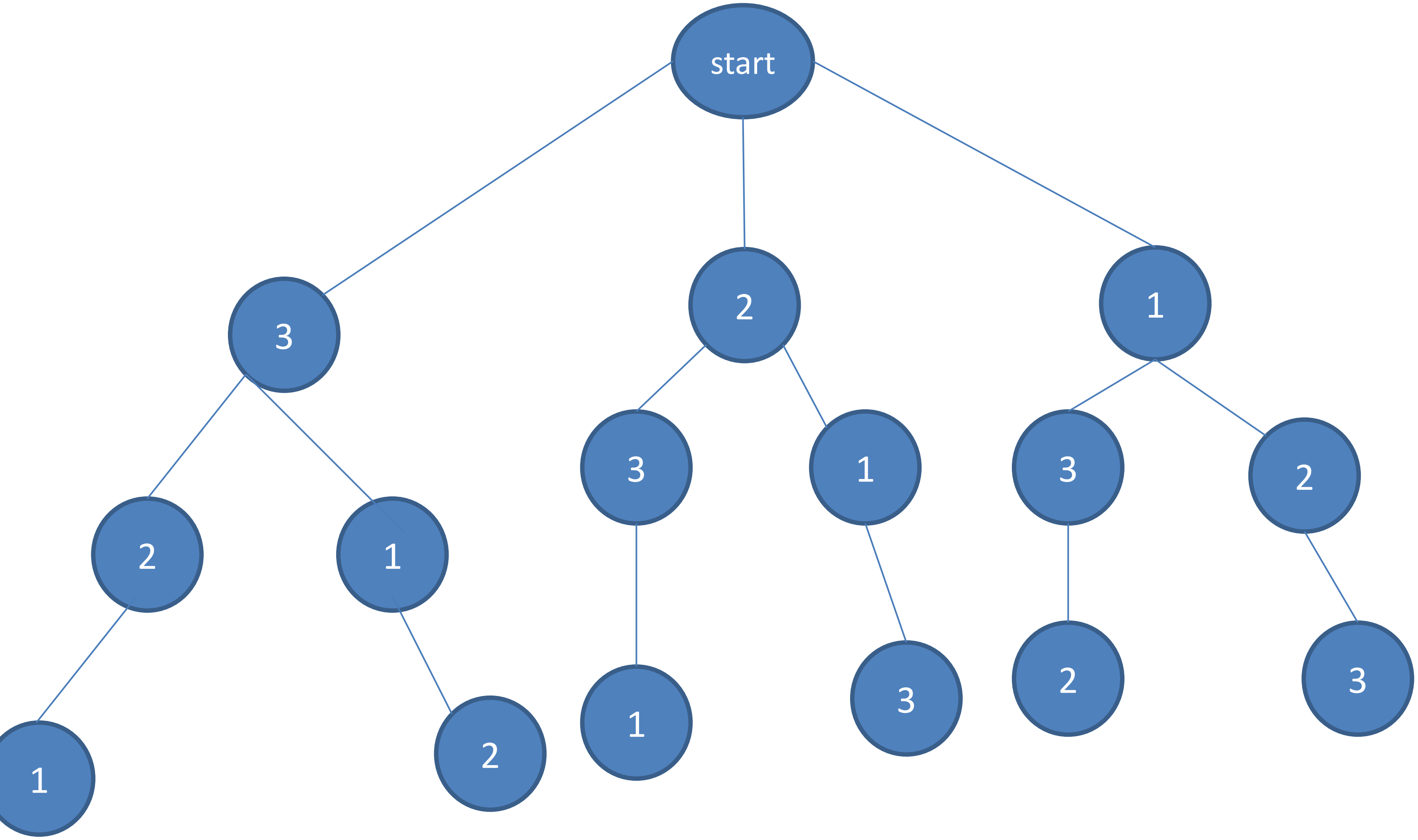
Permutation



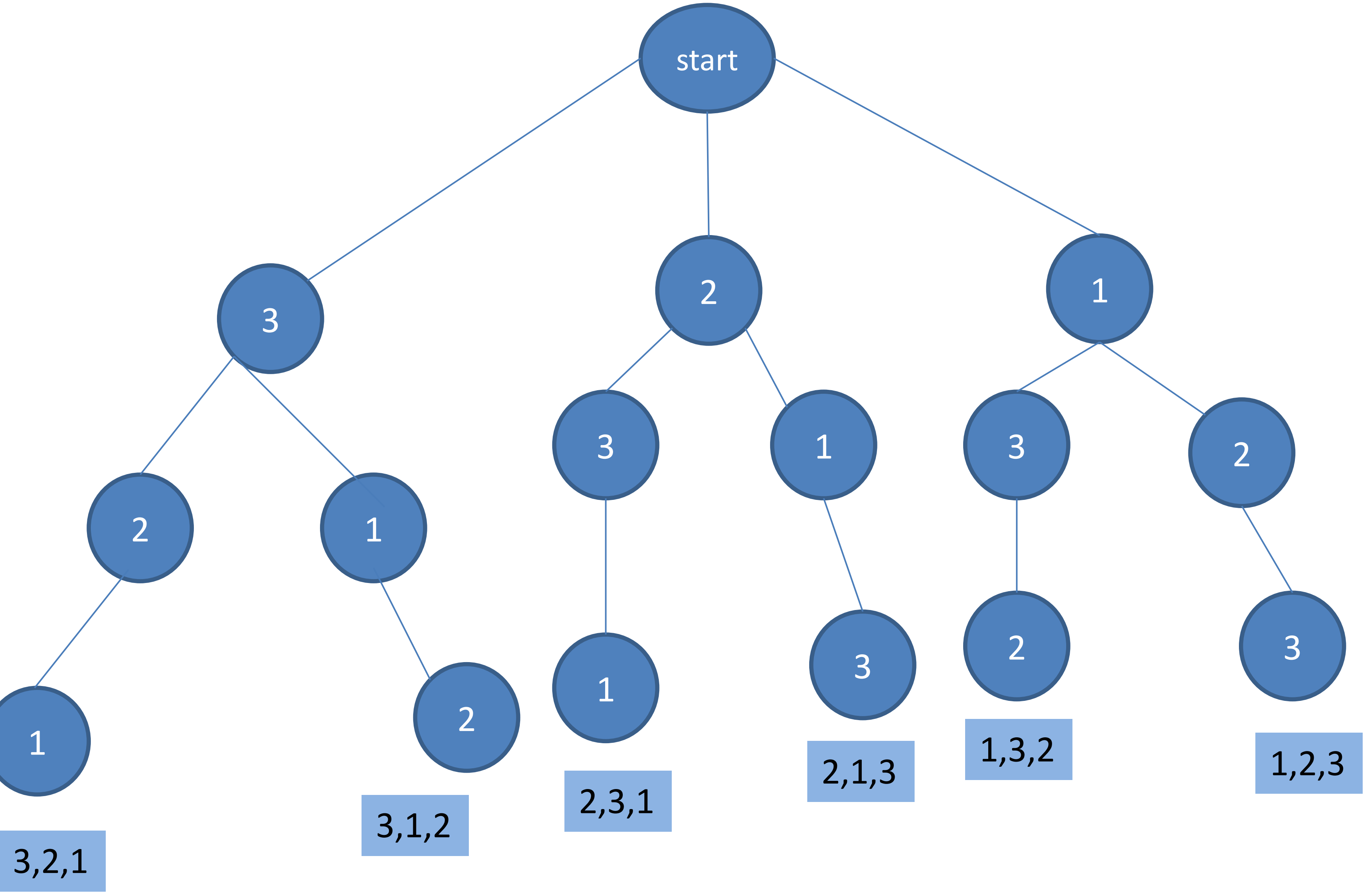
Permutation



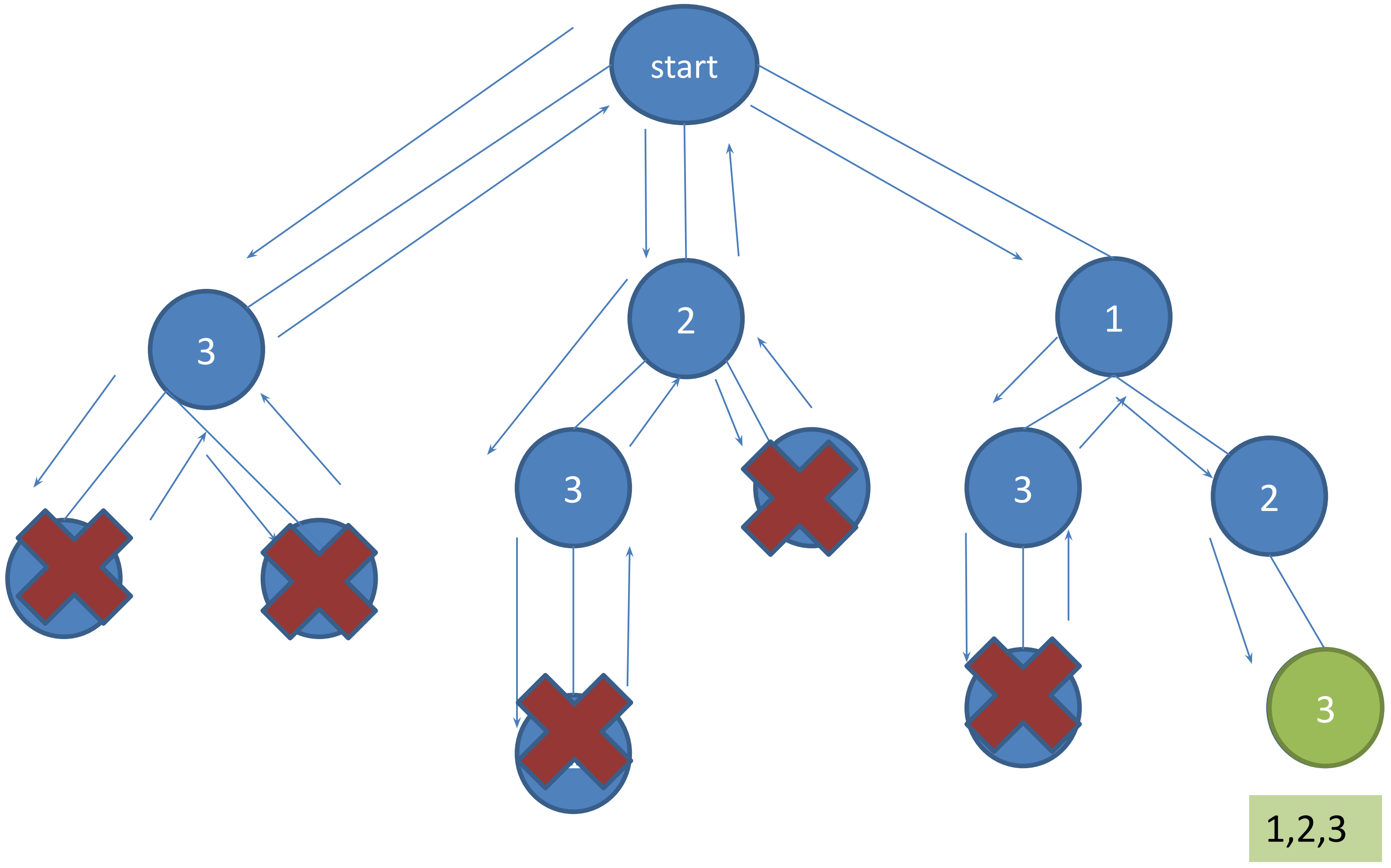
Permutation



Permutation



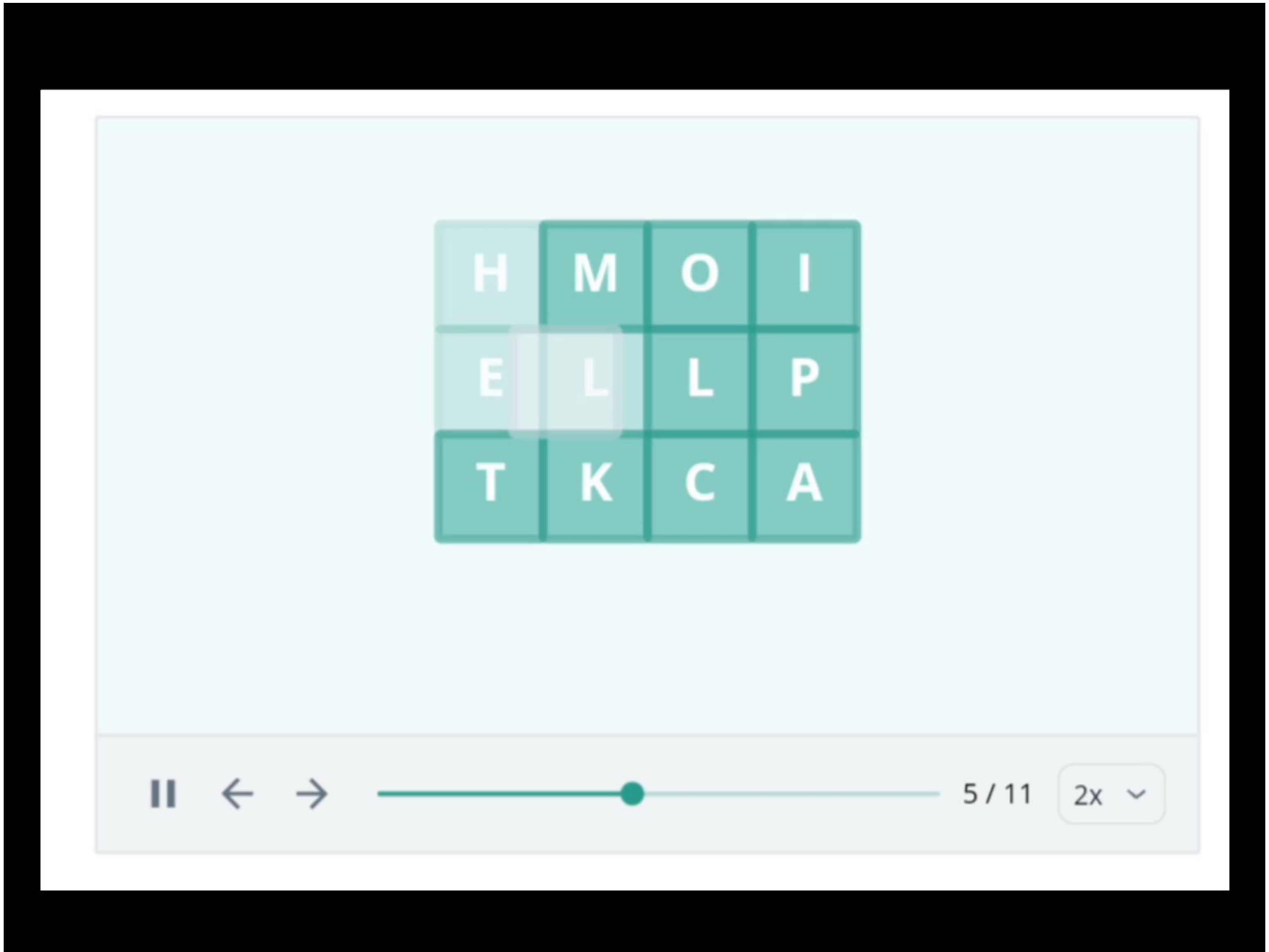
Permutation



Code for Permutation

```
void backtrack(vector<int>& arr, vector<vector<int>>& result, vector<int>& exist qq, vector<bool>& checked) {
   if (exist.size() == arr.size()) {
      result.push_back(exist);
      return;
   }

   for (int i = 0; i < arr.size(); ++i) {
      if (checked[i]) continue;  // Skip if the element is already in the existed permutation

      checked[i] = true;         // Mark the element as checked
      exist.push_back(arr[i]); // Add the element to the existing permutation

      backtrack(arr, result, exist, checked);  // Recursively continue building the permutation

      // Backtrack
      checked[i] = false;        // Unmark the element (so it can be used in other permutations)
      exist.pop_back();        // Remove the element from the existing permutation
   }
}
vector<vector<int>> permute(vector<int>& arr) {
   vector<vector<int>> result;
   vector<int> exist;
   vector<bool> checked(arr.size(), false);
   backtrack(arr, result, exist, checked);
return result;
}
```

Word Search in 2D Array



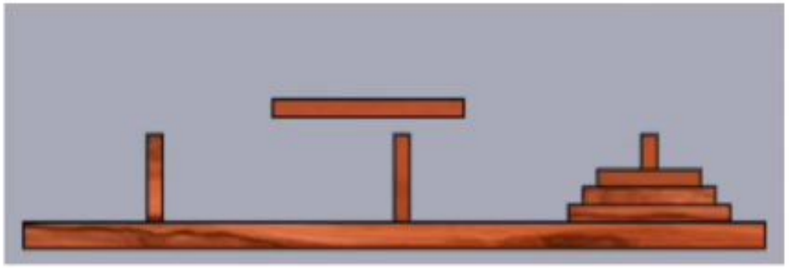
Code for Word Search in 2D Array

```
bool dfs(vector<vector<char>>& board, string& word, int i,
int j, int index) { if (index ==
word.size()) return true;
if (i < 0 || i >= board.size() || j < 0 || j >= board[0].size() ||
board[i][j] != word[index]) return false;
char temp = board[i][j];
board[i][j] = '#'; // Mark as visited
bool found = dfs(board, word, i + 1, j, index + 1) ||
dfs(board, word, i - 1, j, index + 1) ||
dfs(board, word, i, j + 1, index + 1) ||
dfs(board, word, i, j - 1, index + 1);
board[i][j] = temp; // Unmark
return found;
```

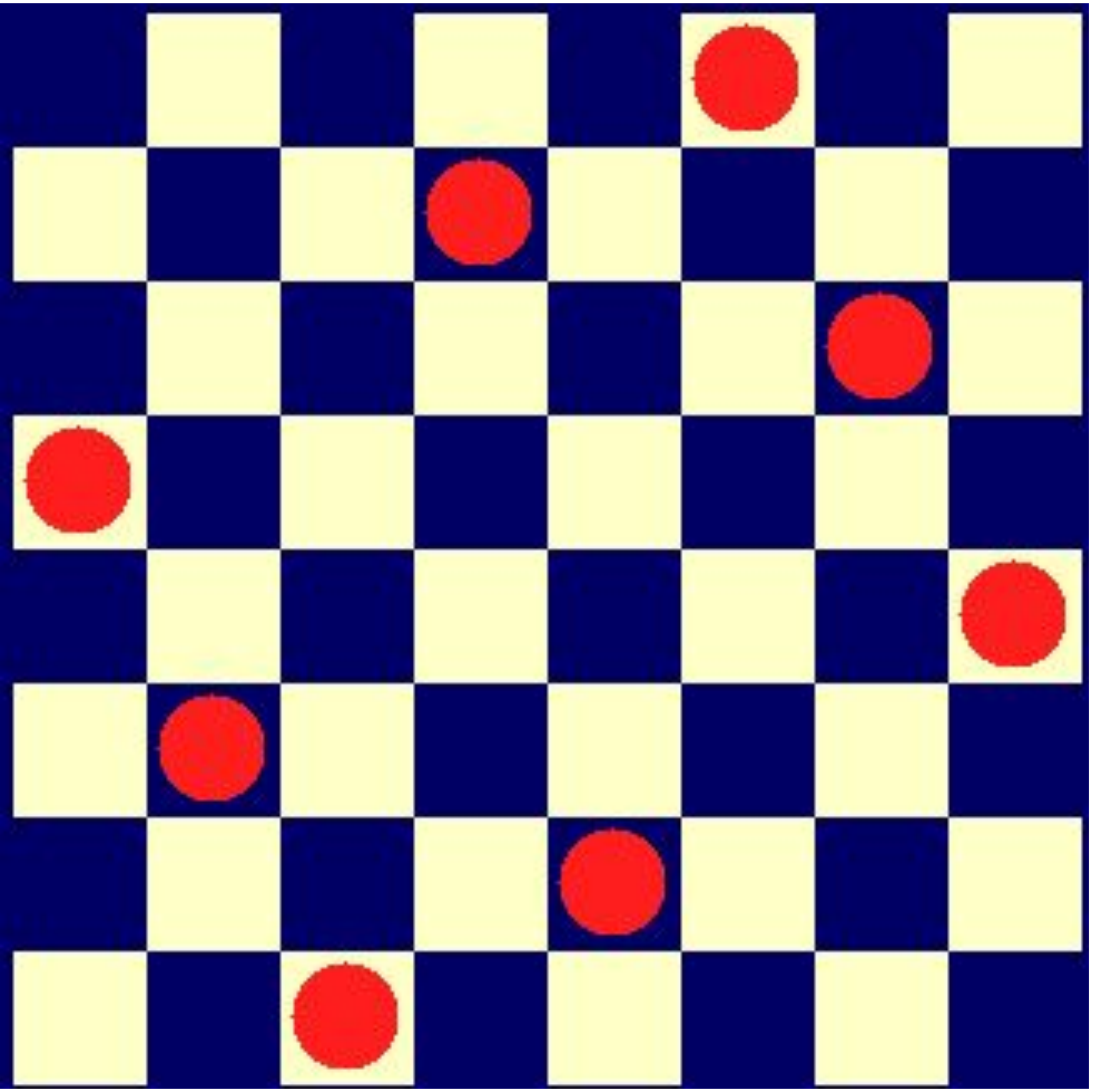
Code for Word Search in 2D Array

```
bool exist(vector<vector<char>>& board, string word) { for
(int i = 0; i <board.size(); ++i) {
for (int j = 0; j < board[0].size(); ++j) {
if (dfs(board, word, i, j, 0)) return true; }
}
return false;
}
```

Towers of Hanoi Animation



N-Queen Problem



place n - queens in such a manner on an n x n chessboard that no queens attack each other by being in the same row, column or diagonal