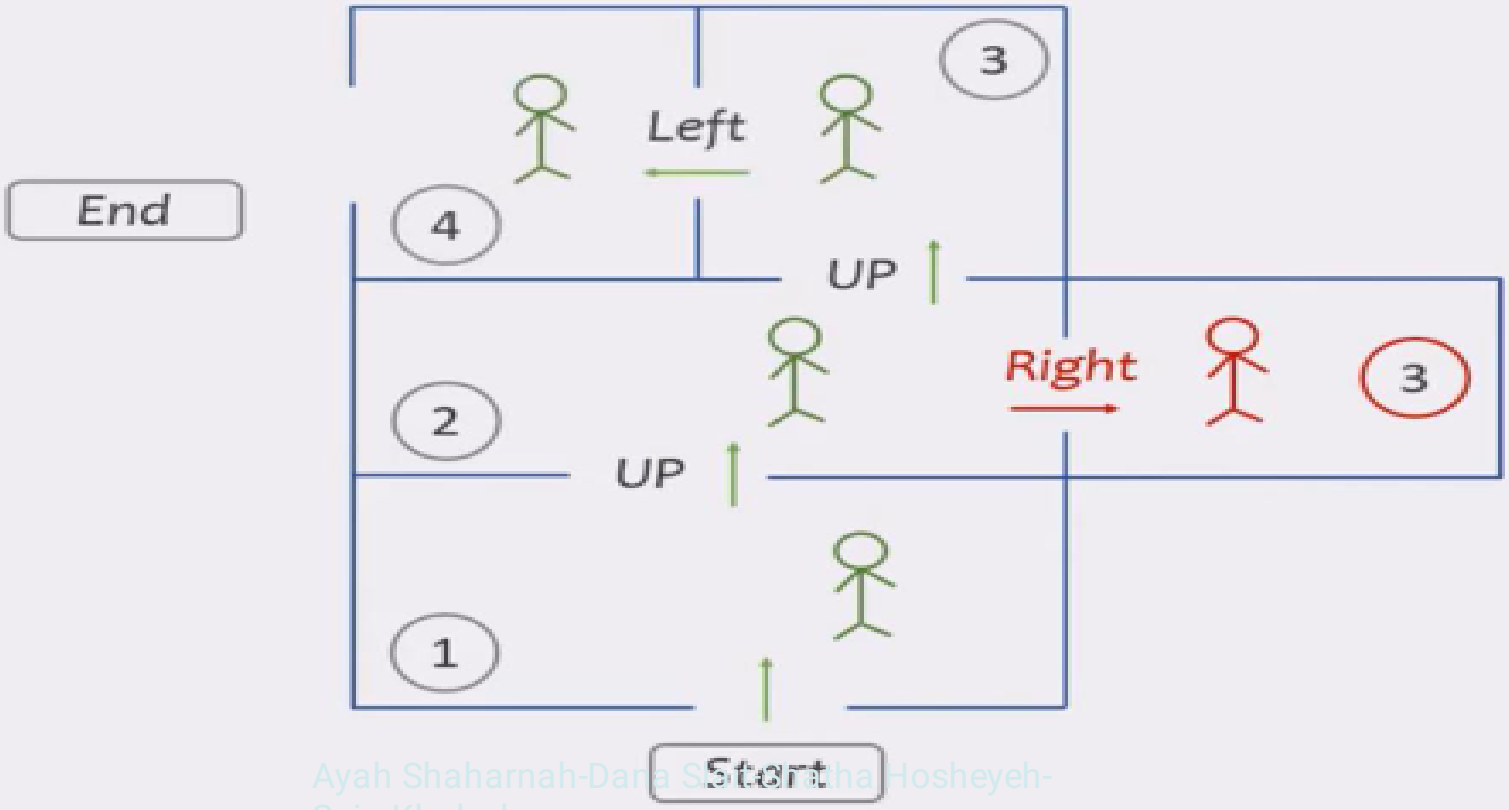
Backtracking Algorithm



Objectives:

1. Definition of Backtraking Algorithm
2. When we use Backtraking Algorithm
3. How it does works
4. Applications of Backtracking
5. Example of Backtracking
6. Sum of subset problem
7. Conclusion

What is Backtracking?

- Backtracking is a class of algorithms for finding solutions to some computational problems.

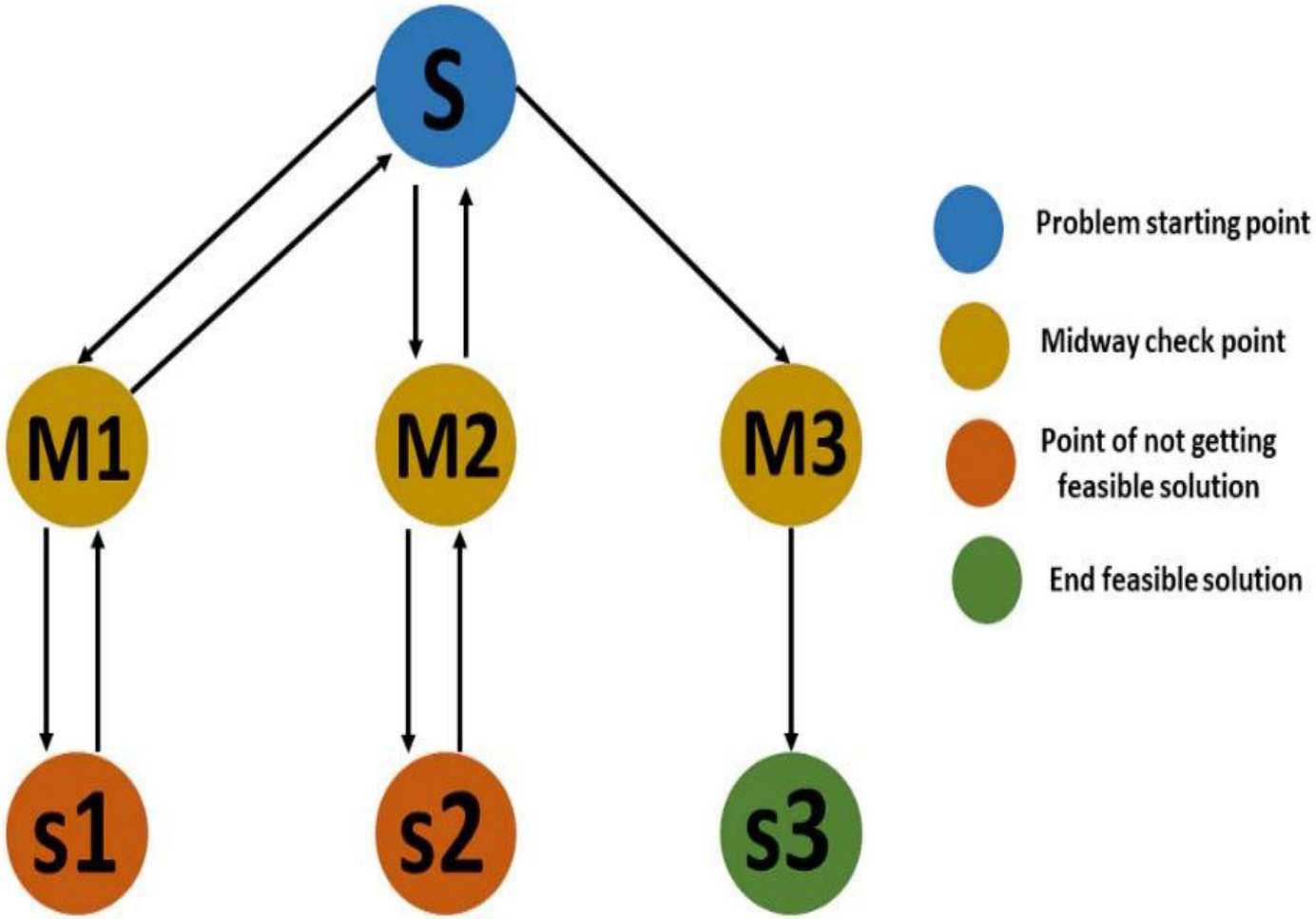
- Abandons a candidate ("backtracks") as soon as it determines that the candidate cannot possibly be completed to a valid solution.

When we use this Algorithm?

- When we have a piece of sufficient information is not available to make the best choice.
- Or each decision leads to a new set of choices. Then again, we backtrack to make new decisions.
- Backtracking algorithms were also discovered to be very effective for solving optimization problems.

How it does works?

In any backtracking algorithm, the algorithm seeks a path to a feasible solution that includes some intermediate checkpoints. If the checkpoints do not lead to a viable solution, the problem can return to the checkpoints and take another path to find a solution


S
M1
M2
M3
s1
s2
s3
Problem starting point
Midway check point
Point of not getting feasible solution
End feasible solution

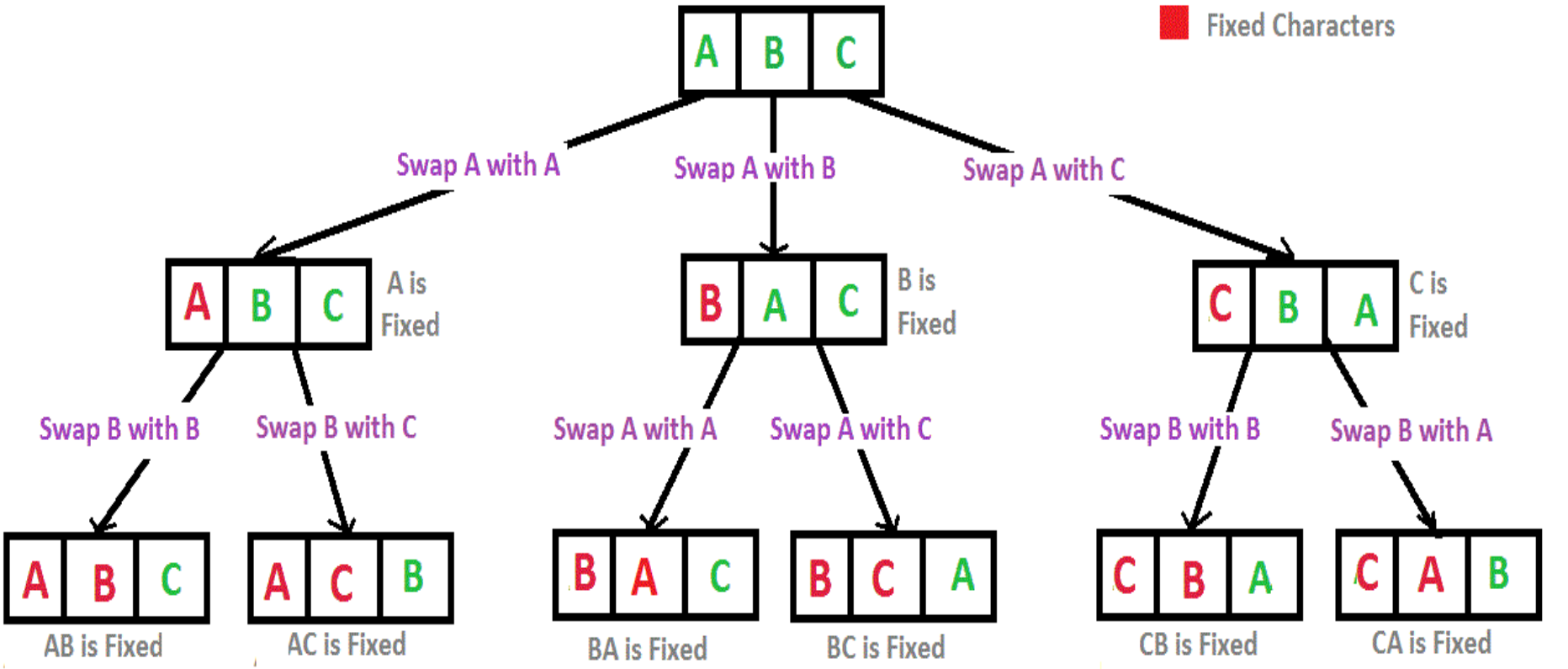
Applications of Backtracking

- N-Queen problem
- Sum of subset problem
- Graph coloring problem
- Knapsack problem
- Hamiltonian cycle problem

The time complexity of backtracking

- If a function calls itself two times then its time complexity is O(2 ^ N). if it calls three times then its time complexity is O(3 ^ N) and so on.

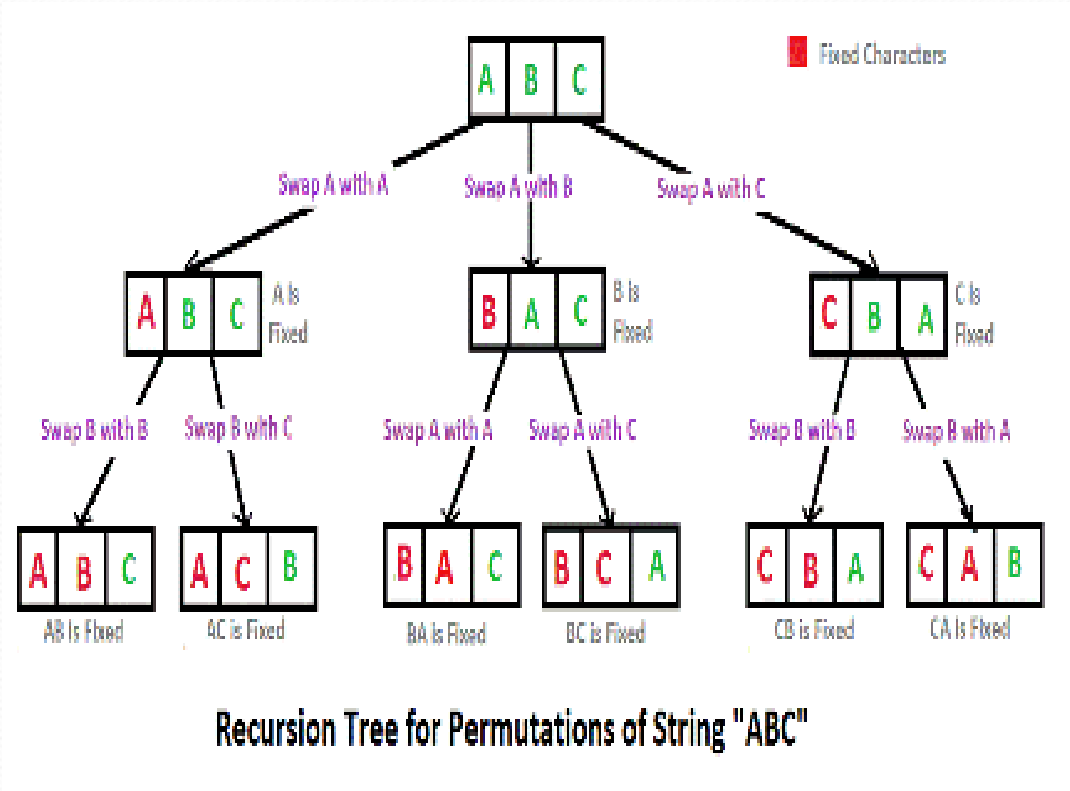
Permutation of string ABC :



Recursion Tree for Permutations of String "ABC"

:Implementation

```
private void permute(String str, int l, int r){
   if(l==r)
   System.out.println(str)
   else
   {
      for(int i=l;i<=r;i++){
         str=swap(str,l,i);
         permute(str,l+1,r);
         str=swap(str,l,i);
      }
   }
}
```



Recursion Tree for Permutations of String "ABC"

Sum of subset problem

- The subset problem is the problem of finding a subset such that the sum of the elements equals a certain number.

Therefore, the backtracking algorithm provides all solutions even in the worst case but in general it provides better solutions than the recursive approach towards the sum of the subsets problem.

```
vector ans, temp;
   void solve (int i, int sum) {
   if (i==n) {
   if (sum > max && sum <= t) {
   ans = temp; max = sum;
   }
   return;
   }
   temp.push_back (a[i]); solve (i+1, sumta[i]);
   temp.pop_back (); solve (i+ 1, sum);
   }
```

Conclusion:

- Most of the back tracking codes are generally few lines of recursive function code.
- It has a large space complexity because we are using recursion, so function information is stored on stack.
- It is very time inefficient in lot of cases when branching factor is large
- We can solve many problems like the knight's tour, N queens, Rat in the maze, subset sum problem and so on.