# Algorithm for Random Forest Attack

```
initialize x_stack, push x
initialize directions, default to zeros
initialize paths_stack

for i in range(MAX_ITERATIONS):
  if prediction(x_stack.peek()) != y:
    return x_stack.peek()

  while budget > 0:
    find paths given x_stack.peek()
    find a viable node with least cost using the paths
    if node is None:
      break

    compute next_x given the selected node
    x_stack.push(next_x)
    reduce budget
    update directions
    set the node as visited
    paths_stack.push(paths)

    if prediction(x_stack.peek()) != y:
      return x_stack.peek()

  x_stack.pop()
  restore directions
  refund budget

  find a viable node with least cost using paths_stack.peek()
  while node is None:
    paths_stack.pop()
    x_stack.pop()
    restore directions
    refund budget
    find a viable node with least cost using paths_stack.peek()

  compute next_x given the selected node
  x_stack.push(next_x)
  reduce budget
  update directions
  set the node as visited
```

## Parameters

- Inputs:
  - k: Number of Decision Trees in a RF

    - m: Number of input features
    - x: (1*m Array) An input example
    - y: (int) Output label
    - model: A trained Random Forest model which contains k Decision Trees
    - budget: (float) Maximum perturbation (Default=0.2*m)
    - MAX_ITERATIONS: The maximum number of iterations (Default=100)
- Outputs:
    - X_adv: (1*m Array) Adversarial example
- Parameters:
    - x_stack: LIFO stack. Keeps tracking the updates of x
    - directions: Which direction can x updates to (-1: Negative only, 0: Both, 1: Positive only)
    - paths_stack: LIFO stack. Keeps tracking the selected paths with given x

## Pseudocode Version 2

```
initialize x_stack, push x
initialize directions, default to zeros
compute the initial paths
initialize paths_stack, push initial paths

for i in range(MAX_ITERATIONS):
    if prediction(x_stack.peek) != y:
        return x_stack.peek

    find a viable path with least cost from the last node in paths_stack
    while path is None or budget < 0:
        if x_stack is not at root:
            last_x = x_stack.pop
            paths_stack.pop
        else:
            last_x = x

        restore directions
        refund budget
        find a viable path with least cost from the last node in
paths_stack

    compute next_x given the selected path
    x_stack.push(next_x)
    reduce budget
    update directions
    set the node in the selected path as visited
    compute new paths
    paths_stack.push(paths)
```