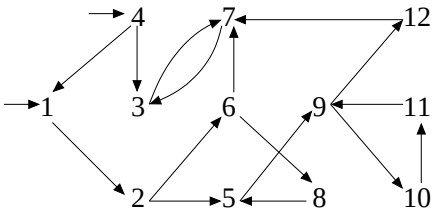
# Model Checking WS 2011: Assignment 6

Institute for Formal Models and Verification, JKU Linz

Due 24.11.2011

**Exercise 21**




Apply non-recursive DFS (see lecture slides 56, 57) on the given graph separately for parts a) and b) and report the contents of `cache` and `stack` and the value of `current` at the end of each iteration of the `while`-loop. Use the convention that states with *larger ID* are always *pushed first* on the stack, e.g. for initial states 1 and 4, 4 is pushed before 1.

Specify the error trace, its length and the number of visited states under the assumption that ...

a) ... state 3 is the *only* bad state: `is_target(3)` is `true` and `false` otherwise.

b) ... state 11 is the *only* bad state: `is_target(11)` is `true` and `false` otherwise.

**Exercise 22**

The same tasks as in Exercise 21, but now apply non-recursive BFS (see lecture slides 60, 61) with the convention that states with *smaller ID* are always *enqueued first* on the queue. As before, report search progress at the end of each iteration of the `while`-loop.

Compare the behaviours of DFS from Exercise 21 and BFS. For detecting bad states 3 and 11, is there a clear preference for DFS or BFS?

## Exercise 23

```
simple_dfs ()
{
  Stack stack;

  forall initial states 's'
    stack.push(s);

  simple_dfs_aux (stack);
}
```

```
simple_dfs_aux (Stack stack)
{
  while (!stack.empty())
  {
    current = stack.pop();
    forall successors 'next' of current
      stack.push(next);
  }
}
```

For a natural number $n \geq 1$, let LTS $L_n := (S_n, I_n, \Sigma_n, T_n)$ be defined as follows:
$S_n := \{s_0, s_1, \ldots, s_{n-1}\}$, $I_n := \{s_0\}$, $\Sigma_n := \{a\}$, and $T_n(s_i, a, s_j) \Leftrightarrow i < j$.

The pseudo code of a simplified DFS algorithm `simple_dfs()` without target checking and state caching is shown above (compare with slides 56 and 57).

What is the *exact* number of total state visits in terms of $n$ when calling `simple_dfs()` on $L_n$, i.e. how often is line "`current = stack.pop()`" executed? Justify your answer.

## Exercise 24

Given a collision-free hash function and an empty hash table using collision chains with a size of $2^{10}$ slots initially. The hash table is said to be full and is resized each time there are as many elements in the hash table as there are slots and a new element is inserted. Resizing requires all elements in the table to be hashed again and assigned to slots in the resized table. [1]

When inserting $2^{20}$ elements altogether, determine the total number of hash value computations (including resizing) and table resize operations, if

a) the table size is doubled each time the hash table is full.

b) the table size is incremented by $2^{10}$ slots each time the hash table is full.

[1] For illustration, consider also the implementation of the `testhash` program demonstrated in the lecture. Source code is available from `http://fmv.jku.at/mc/testhash.c`.