

# Efficient Algorithms and Programming

## Week 12

### Reading before Monday November 19th

CLR Chapter 36.5.

### Exercises for Monday November 19th

1. Problem 36-2 in Chapter 36.

### Assignment for Friday November 23th

Hand in the solution to the following three problems (from earlier exams):

### Problem 1 (E96, 30%)

Let  $G = (V, E)$  be a *directed acyclic graph* (DAG). A vertex  $v \in V$  is a *sink* if it has out-degree 0. In the example below,  $a$  and  $b$  are sinks.

Assuming  $u$  is a vertex in  $G$ , the number of paths starting at  $u$  and ending in sinks can be computed with the following algorithm:

```
PATH-COUNT( $u$ ) =  
  if  $u$  is a sink then  
     $s \leftarrow 1$   
  else  
     $s \leftarrow 0$   
    for each  $v \in Adj[u]$   
      do  $s \leftarrow s + PATH-COUNT(v)$   
  endif  
  return  $s$ 
```

#### Question 1.1

What is the asymptotic runtime of PATH-COUNT? Give an example of a graph that exhibits the worst-case behaviour.

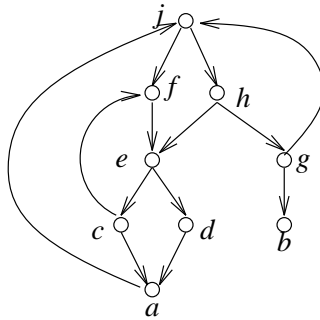
#### Question 1.2

Suggest an improvement of PATH-COUNT which makes it more efficient. Give the algorithm and its asymptotic runtime.

**Question 1.3**

A path  $p$  can be represented as a list  $\langle v_1, \dots, v_n \rangle$  of the vertices on it. Give an algorithm  $\text{PATH-LIST}(u)$  which computes a list of all paths starting at vertex  $u$  and ending in a sink. You can use the notation  $\langle p_1, \dots, p_n \rangle$  for a list of paths  $p_1, \dots, p_n$ .

We now consider general directed graphs,  $G = (V, E)$ , which are not necessarily acyclic. A *simple cycle* in  $G$  is a list of vertices  $\langle v_0, v_1, \dots, v_n \rangle$  where  $v_0 = v_n$ , all other vertices are different and  $(v_i, v_{i+1}) \in E$  for  $0 \leq i \leq n - 1$ . In the graph below  $\langle j, h, e, d, a, j \rangle$  is an example of a simple cycle. The graph has five simple cycles containing  $j$ .



**Question 1.4**

Let  $v$  be a vertex in  $G$ . Give an efficient algorithm  $\text{CYCLE-COUNT}(v)$ , which computes the number of simple cycles in  $G$  which contains  $v$ . (Hint: Think of  $v$  as a sink.)

**Question 1.5**

Give an algorithm  $\text{CYCLE-LIST}(v)$ , which computes a list of all simple cycles in  $G$  containing  $v$ .

**Question 1.6**

Is your algorithm  $\text{CYCLE-LIST}(v)$  optimal? Justify your answer.

**Problem 2 (E97, 30%)**

In this problem, an algorithm is developed which can improve the ordering of an ROBDD. Assume that the  $T$ -table has been extended with a fourth field  $mark$ , which can be either 0 or 1:

$u$	$var$	$low$	$high$	$mark$

In the questions below, you can assume that all the  $mark$  entries are initialized to 0.

**Question 2.1**

Give an algorithm  $\text{MARK}[T](u)$ , which assigns 1 to the *mark* entry of each node that can be reached from a given node  $u$ . The algorithm must have the asymptotic running time  $O(|u|)$ , where  $|u|$  is the number of nodes that can be reached from  $u$ .

**Question 2.2**

Give an algorithm  $\text{COUNT}[T](u)$  that in time  $O(|u|)$  counts how many nodes can be reached from  $u$ , i.e., it returns the number  $|u|$ .

**Question 2.3**

Give an algorithm  $\text{MAXOCC}[T](u)$  that in time  $O(|u| + n)$  finds the index of the variable among the  $n$  variables that occurs most frequently in the ROBDD with root  $u$ . If several variables occur the maximal number of times, any one of them is returned.

The variable with index  $i$  can be “pulled up” to the root of an ROBDD  $u$  using the following operations:

$$u' \leftarrow \text{MK}(i, \text{RESTRICT}(u, i, 0), \text{RESTRICT}(u, i, 1))$$

The ROBDD  $u'$  is equivalent to  $u$  but has the ordering

$$x_i < x_1 < \dots < x_{i-1} < x_{i+1} < \dots < x_n$$

**Question 2.4**

Give a greedy algorithm  $\text{REORDER}(u)$ , which given an ROBDD  $u$ , iteratively finds a variable that occurs most frequently in  $u$ , pulls it up to the root, and repeats doing this as long as the size of the ROBDD is decreasing.

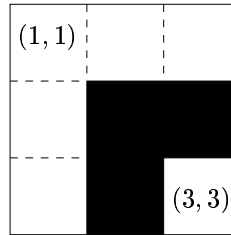
**Question 2.5**

Analyze the running time of your algorithm.

**Problem 3 (E98, 20%)**

In this problem, you are given a *maze* as an  $n \times n$  matrix whose entries are either  $\square$  or  $\blacksquare$ . If entry  $(i, j)$  is  $\square$ , position  $(i, j)$  of the maze is free, otherwise position  $(i, j)$  is blocked. Only horizontal and vertical moves are allowed in the maze (no diagonal moves). Moves can be made only to the positions containing  $\square$ . You can assume that positions  $(1, 1)$  and  $(n, n)$  always contain  $\square$ . A *path* is a sequence of positions and the *length* of a path is the number of positions in the sequence. A *solution* is a path from position  $(1, 1)$  to position  $(n, n)$ . It is possible that no solution exists. For example, the  $3 \times 3$  matrix to the left corresponds to the maze to the right:

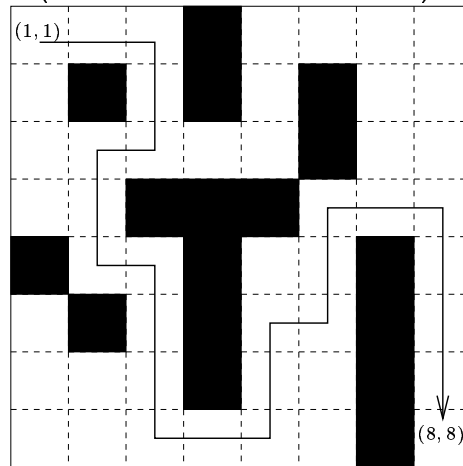
$$\begin{pmatrix} \square & \square & \square \\ \square & \blacksquare & \blacksquare \\ \square & \blacksquare & \square \end{pmatrix}$$



In this example there are no solutions.

The  $8 \times 8$  matrix to the left corresponds to the maze to the right:

$$\begin{pmatrix} \square & \square & \square & \blacksquare & \square & \square & \square & \square \\ \square & \blacksquare & \square & \blacksquare & \square & \blacksquare & \square & \square \\ \square & \square & \square & \square & \square & \square & \square & \square \\ \square & \square & \blacksquare & \blacksquare & \blacksquare & \square & \square & \square \\ \blacksquare & \square & \square & \blacksquare & \square & \square & \blacksquare & \square \\ \square & \blacksquare & \square & \blacksquare & \square & \square & \blacksquare & \square \\ \square & \square & \square & \blacksquare & \square & \square & \blacksquare & \square \\ \square & \square & \square & \square & \square & \square & \blacksquare & \square \end{pmatrix}$$



In this maze there are many paths from the start position  $(1, 1)$  to the finish position  $(8, 8)$ . A solution of length 25 is shown in the maze to the right.

**Question 3.1**

Given a maze as an  $n \times n$  matrix  $M$ , describe how to construct a graph  $G$  such that the graph has a path of length  $l - 1$  from vertex  $(1, 1)$  to vertex  $(n, n)$  if and only if there is a solution of length  $l$  in the maze.

**Question 3.2**

Given a maze as an  $n \times n$  matrix  $M$ , describe an  $O(n^2)$  algorithm to find the length of a shortest path from position  $(1, 1)$  to position  $(n, n)$ . Your algorithm should detect if no solution exists.

