COS4851 Assignment 3 (2019) [55]

ASSIGNMENT 03

DUE DATE:	3 September 2019
SUBMISSION PROCEDURE:	Written
WEIGHT:	30
UNIQUE NUMBER:	760637
STUDY MATERIAL:	*Bratko,* Chapters 8, 9, 11, 12
	(chapter 9: ignore sections 9.2 – 9.5)

Important: The purpose of this assignment is to introduce basic breadth-first, depth-first and best-first search algorithms, as well as various sort algorithms. The topic of difference lists is covered and you will also learn the way in which we can use tail-recursion for the efficient execution of procedures, as well as how logic programming can be used for solving planning problems and logic puzzles.

Test Results: Please note that you need to submit screen shots (using *fn+prt sc* for Windows in most instances) of results for questions where a program/procedure or query is required. This will assist us to see whether you obtained the correct results and if not, try to point out where you went wrong.

Your programs should also be *robust*. This means that it should check whether all the input arguments for a specific procedure are legal. For example, if you know you are working with integers, an input of the constant that is not an integer is not acceptable.

Important note: It is not advisable that you search for Prolog solutions to the questions in this assignment on the internet. You may find a solution to a specific problem but that will not assist you in acquiring the necessary skills for mastering this programming paradigm.

Question 1 [7]

Although most programmers find them counterintuitive, difference lists are powerful and efficient. The trick usually lies in the way in which the predicate (or query) is constructed.

Define a predicate `app3_dl(L1,L2,L3,L)` that concatenates three separate *difference* lists.

Question 2 [11]

The knapsack problem:
This problem concerns a knapsack, which can carry a certain maximum weight, and a number of objects of various weights, which can be carried in the knapsack.

We need to answer the following question: *What selection from the objects can be loaded into the knapsack so that their combined weight is exactly the same as the maximum weight that the knapsack can handle? (Note: we do not need to use all the objects.)*

Write a recursive Prolog procedure to solve this problem using the following guidelines:

1. Solutions can be obtained by using the query `knapsack(Objects_Available, Target_weight, Objects_included)`.
2. Use a list of integers to represent the objects available, a list of integers to represent the objects to be included in the load and an integer to represent the maximum (target) weight.
3. An object can be included in the load if its weight does not exceed the target weight. If an object is in fact selected to be included, the rest of the load comprises some selection of the other objects to a total weight reduced by the weight of the object just included.
4. The base case is reached when, with every object having been considered for inclusion in the load, the target weight is equal to 0.

You can use the following example to test your program: Suppose you have a number of objects with individual weights 2, 7, 18, 5, 10 and 3 respectively, and the maximum weight that can be loaded into the knapsack is 20.

One possible solution will be loading the objects weighing 2 and 18 respectively into the knapsack.

Find alternative solutions by using ‘;’.

Question 3 [11]

(a) Write a *recursive* procedure `powers_two(N,R)` that returns the answer to the following calculation:
$R = 2^N$. (You may not use the ‘^’ operator.)

(b) Write a *tail-recursive* procedure `powers_two(N,R)` that returns the answer to the calculation $R = 2^N$. (You may not use the ‘^’ operator.)

Question 4 **[10]**

The tower of Hanoi legend was told in a Hanoi monastery more than 200 years ago. A group of proud monks was assigned a task to perform, which was to move 10 disks from one peg (origin) to another (destination) with the help of a third (temporary) peg. There were two rules:

- only one disk could be moved at a time, and
- a larger disk could never be placed on top of a smaller disk.

Initially, the disks are all on peg 1 (origin), with the largest at the bottom and the smallest at the top.

Write a *recursive* Prolog program to find a sequence of moves to complete this task. The general algorithm for moving *n* disks from Origin to Destination is as follows:

```
If n = 1, move disk 1 from Origin to Destination.
Otherwise
        move n – 1 disks (one at a time) from Origin to Temporary
        move disk n from Origin to Destination
        move n – 1 disks (one at a time) from Temporary to Destination.
```

Question 5 **[8]**

Modify the *quicksort* algorithm so that the resulting list does not include any duplicates that may occur in the list being sorted. Because of its lack of efficiency, you may not use the `setof` built-in predicate.

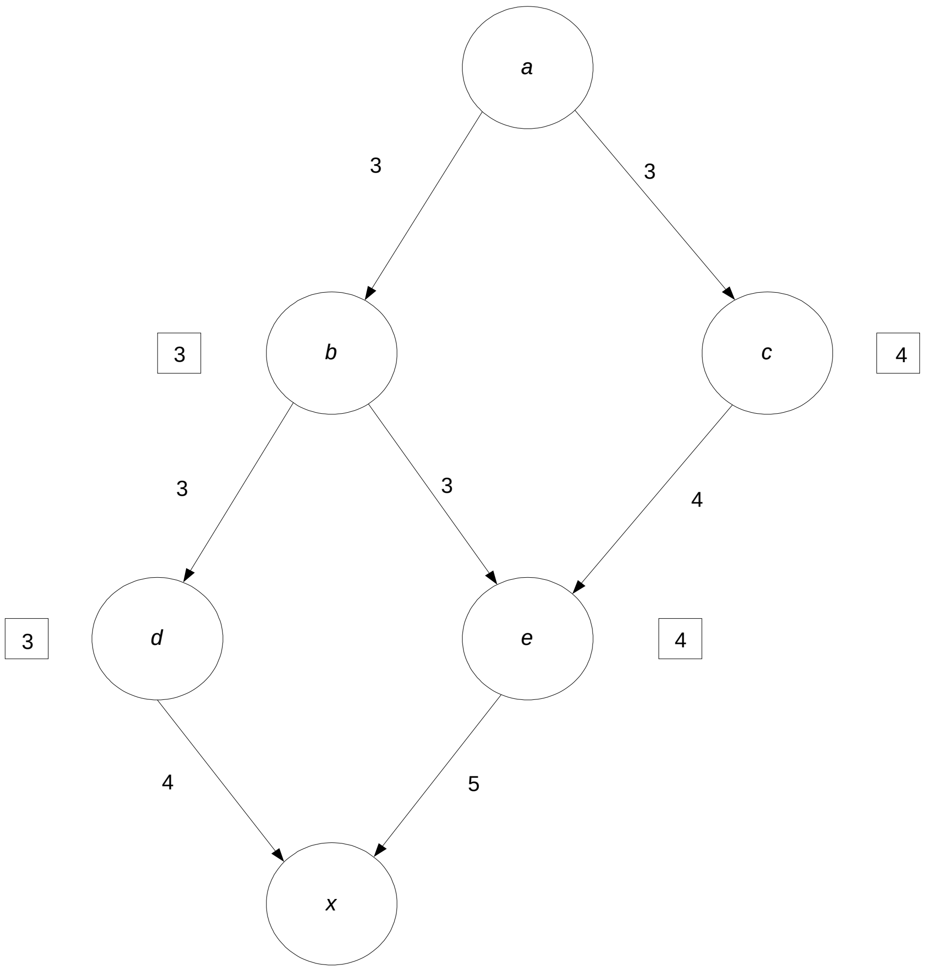
(Refer to *Bratko,* figure 9.2.)

Question 6 **[8]**

Consider the following graph that indicates the cost of all the individual arcs. It also gives alongside each internal node an *estimate* of the cost of the remaining path between that node and the goal. The node *a* is the initial node and the node *x* is the goal.

Indicate the next node that is chosen from each node in the path from the initial node to the goal by the *best-first algorithm*. Clearly state the grounds on which each next node is chosen.

Do the trace by hand. You need not write a program for this question.


a
3
3
3
b
c
4
3
3
4
3
d
e
4
4
5
x