Assignment 1

Total Points = 100

Instructions:

1. You must submit this assignment on Blackboard by clicking the title link for the assignment. You must use **SWI Prolog** to implement these programs. Make sure your programs compile (without any compiler errors). You will not receive credit if your program for a problem does not compile. If you are unable to complete any of the programs, submit the parts that work (with no compiler errors) for partial credit.
2. Your grade will be based on meeting the requirements, functionality (does the program do what it is supposed to do), readability (is the code nicely indented and formatted), and understandability (are the literals meaningful and is the code well documented with appropriate comments). You have to incorporate all the good programming practices and styles.

Deliverables:

1. You should **submit one PDF file**. This PDF file should contain **code to the solution** of the problems (in **text format only**, no snapshot will be accepted) and **your program's sample runs** with output (in text format only, no snapshot will be accepted). Make sure to paste your code correctly (with correct indentation). If you have any text for the grader to read before grading, you can include it at the top of this file as comments or in blackboard notes.
2. You should **also submit a zip file** containing the **.pl files** with the same name given against each problem below**.** These files should contain your code for the individual problems(s) in the assignment for the grader to run.

1. **Find a path in a weighted graph:** **(10 points)**

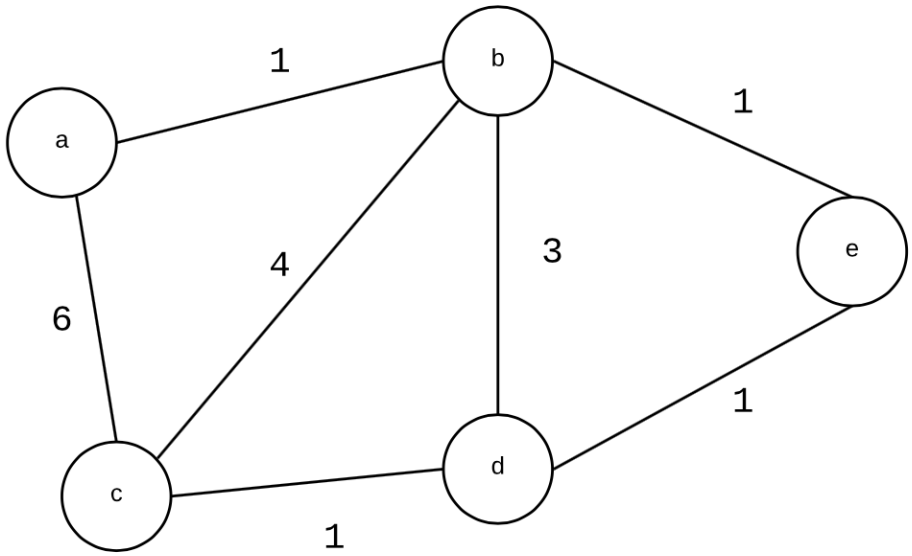
Write prolog predicate to represent the following weighted graph. Using the graph representation, write a prolog predicate to find a path in the weighted graph between two nodes. Along with finding the path, you should also print the total weight obtained while taking the path determined by your program. The solution should not generate cyclic paths. The goal predicate will take the form:

findpath(X,Y,Weight,Path).

where X, Y = name of the node

Weight = weight of the path taken

Path = path taken in the form of a list



Sample Run:

?- findpath(a,e,Weight,Path).
Path = [a,b,e]
Weight = 2;

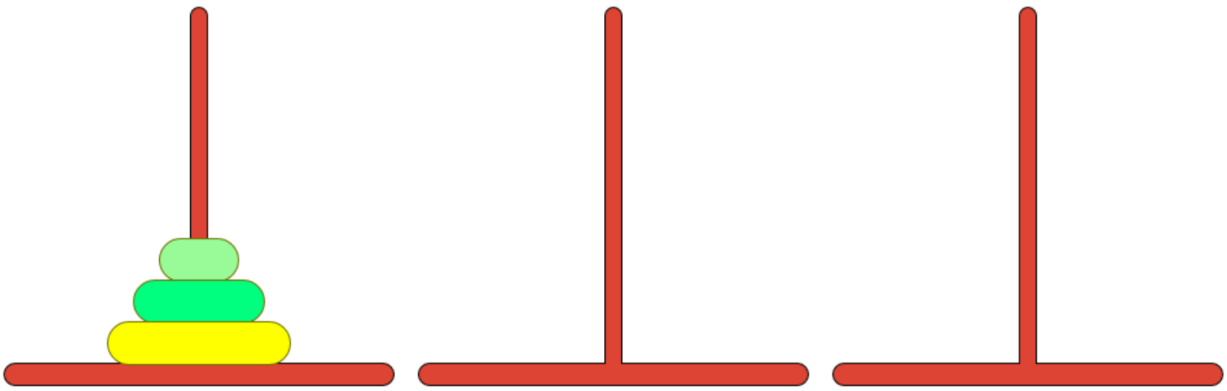
Path = [a,c,d,e]
Weight = 8 ;
…

File Name: findpath.pl

2. Tower of Hanoi: (10 points)

Program the Tower of Hanoi puzzle. You must use a, b, c as your peg names. Your output should be a bunch of `move' statements like `move a to b' (which means moving the top disk from peg a to peg b). You can use Prolog's `write' predicate to output a string or a variable to the screen. The `nl' predicate outputs a newline to screen.

Note: The goal predicate will take the form:
hanoi (X, a, c, b).
where X is a variable (the number of discs)



Sample Run:

```
?- hanoi(3,a,c,b).
        Move a to c.
        Move a to b.
        Move c to b.
        Move a to c.
        Move b to a.
        Move b to c.
        Move a to c.
```

File Name: hanoi.pl

3. Write in full words: **(10 points)**

On legal and financial documents, like cheques, numbers must sometimes be written in full words. Example: 283 must be written as two-eight-three. Write a predicate full_words/1 to print (non-negative) integer numbers in full words.

Sample Run:

```
?- full_words(283).
two-eight-three

?- full_words(2018).
two-zero-one-eight
```

File Name: fullWords.pl

---

4. Generate the combinations of K distinct objects chosen from the N elements of a list: **(10 points)**

In how many ways can a committee of 3 be chosen from a group of 12 people? We all know that there are C(12,3) = 220 possibilities (C(N,K) denotes the well-known binomial coefficients). We need to generate all the possibilities (via backtracking). Write a combination predicate which takes the following form:

combination(X, [x, y, z, ...], L).

where X is the number of elements to be chosen from the list

[x, y, z, ...] is the list of elements to be used

L is the resultant list of distinct objects

Sample Run:

```
?- combination(3,[a,b,c,d,e,f],L).
L = [a,b,c] ;
L = [a,b,d] ;
L = [a,b,e] ;
....
```

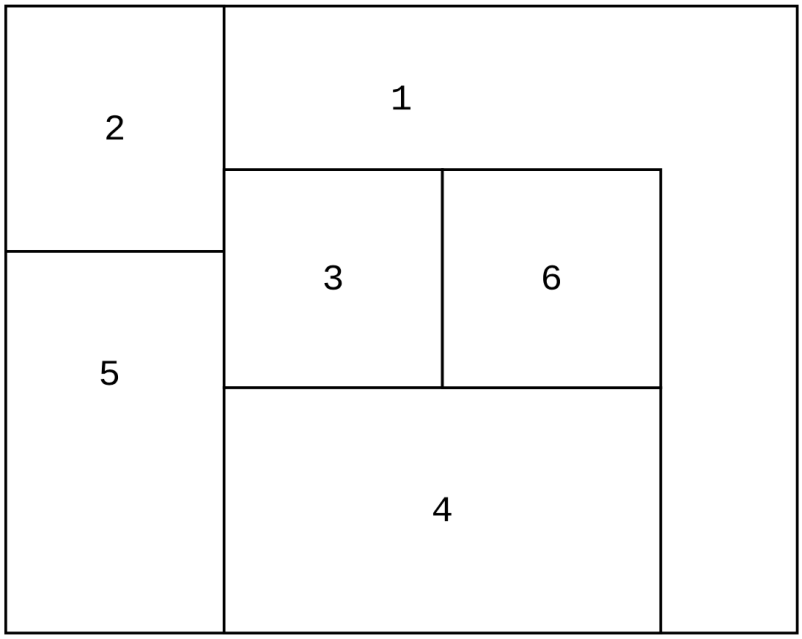
File Name: combination.pl

---

5. Map Coloring: **(20 points)**

A famous problem in mathematics concerns coloring adjacent planar regions. Like cartographic maps, in this problem, it is required that, whatever colors are actually used, no two adjacent regions may have the same color given the total number of colors to be used in the entire region are less than or equal to 4. Two regions are considered adjacent if they share some boundary line segment. Write a prolog program to represent the following region (map) below and write a prolog predicate "color_map" that will color the map for the whole region so that no adjacent regions are colored with the same color given that you should use at most 4 colors (red, green, blue, yellow) for the entire region. Any property of the graph instance should not be hardcoded in the "color_map" predicate.

Please note that you need to represent the following region as well in your solution by writing the vertex, edge and color predicates, for full credit.



Sample Run:

?-color_map(L).
L = [[1, red], [2, green], [3, blue], [4, yellow], [5, red], [6, green]];
…
…

File Name: map.pl

6. N queens problem: (20 points)

Write a **tail-recursive** prolog program for the n queens problem. The objective is to place n queens on a chessboard of size (n* n) so that no two queens are attacking each other; i.e., no two queens are in the same row, the same column, or on the same diagonal.

Note: All the **recursive** predicates in this program must be **tail-recursive**. Represent the positions of the queens as a list of numbers 1..N. For example: When N = 8, the list (Qs) with the positions would be [4,2,7,3,6,8,5,1]. This means that the queen in the first column is in row 4, the queen in the second column is in row 2, etc. The goal predicate will take the form:
queens (N, Qs).
where N = the number of queens
Qs = solution to the problem

Sample Run:

```
?- queens(8, Qs).
Qs = [4, 2, 7, 3, 6, 8, 5, 1];
Qs = [5, 2, 4, 7, 3, 8, 6, 1];
Qs = [3, 5, 2, 8, 6, 4, 7, 1];
…
…
…
```

Filename: nQueens.pl

---

7. Goldbach's Conjecture: (20 points)

Goldbach's conjecture says that every positive even number greater than 2 is the sum of two prime numbers. Example: 28 = 5 + 23. It is one of the most famous facts in number theory that has not been proved to be correct in the general case. It has been *numerically* confirmed up to very large numbers (much larger than we can go with our Prolog system). Write a predicate to find the two prime numbers that sum up to a given even integer.

Sample Run:

```
?- goldbach(28, L).
L = [5,23];
L = [11, 17];
…
…

?- goldbach(30, L).
L = [7,23];
L = [11, 19];
L = [13, 17]
```

Filename: goldbach.pl
