problem 1 *Graph Cycles*

Consider the problem of needing to remove cycles from a graph.

1. Explain how you would remove all cycles from a graph using only DFS. You should describe your algorithm so that it removes the minimum number of edges in order to make it an acyclic graph (i.e., you cannot just say "remove all edges from the graph").

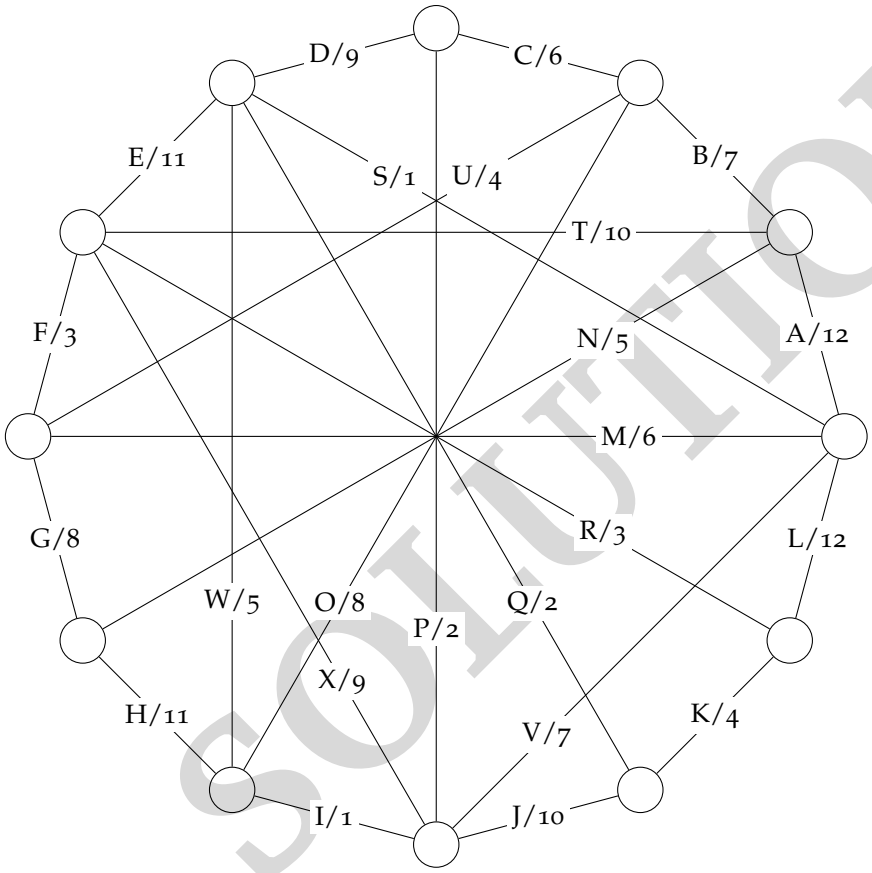
   Solution: Run DFS on the graph ($\Theta(e+v)$ time) to find a cycle – formally, to find a back edge. Remove back edges as you go. Total running time: $\Theta(e+v)$.

2. What is the running time of that algorithm? Explain why.

   Solution: See above answer

problem 2 *Minimum Spanning Tree*

Find the minimum spanning tree of the following graph. Each edge in the graph below is labelled with a letter and its weight. If you encounter a situation where there are two edges of the same weight to consider, then consider the one earlier alphabetically. The answer should be a space separated list of node letters, in alphabetical order, such as: `A B C D E F`. Also provide the cost of that tree.



Solution: B F I K N P Q R S U W; weight: 37

PROBLEM 3 *Square Root*

You are to design a *divide-and-conquer* algorithm for finding the square root of a positive integer. If the square root is an integer, it should return that. Otherwise, if it is between x and $x+1$, it should return x. It should run in $\Theta(\log n)$ time. *Hint: think of a binary search.*

1. Write the algorithm in pseudo-code. This pseudo-code should be in Python-like or Java-like syntax. This -like part means that we are not going to check for exact Python or Java syntax, but look at it as pseudo-code. But the formatting, indentation, etc., should be like Python (or Java). We recommend using the `lstlisting` environment for your pseudo-code. You can see how that environment works at https://www.overleaf.com/learn/latex/Code_listing. **NOTE:** your `lstlisting` environment has to be outside your `solution` environment.

   Solution:

```
def SquareRoot(n):
  return SquareRootRecursive(n,1,n)

def SquareRootRecursive(n,lo,hi):
  mid = (lo+hi)//2
  if mid * mid == n:
    return mid
  if lo == hi:
```

```
        return lo-1
    if mid * mid > n:
        return SquareRootRecursive(n,lo,mid)
    else:
        return SquareRootRecursive(n,mid+1,hi)
```

2. What is the recurrence relation for that algorithm? (You do not need to determine its running time)

   Solution:

   $T(n) = T(n/2) + \Theta(1)$

PROBLEM 4 *Finishing up Maximum Array Subsequence*

Write the algorithm for the `dividingLineSolution(A,first,last)` function discussed in class, which is used in the Maximum Subarray Sum algorithm (in the first slide set on divide and conquer). It is an iterative function. Like the above, we recommend the `lstlisting` environment. Again, either Python-like or Java-like syntax.

This is the `maxCrossingSum()` function from https://www.geeksforgeeks.org/dsa/maximum-subarray-sum-using

Solution:

```
// Find the maximum possible sum in arr[] such that arr[m]
// is part of it
int maxCrossingSum(vector<int> &arr, int l, int m, int h) {

    // Include elements on left of mid.
    int sum = 0;
    int leftSum = INT_MIN;
    for (int i = m; i >= l; i--) {
        sum = sum + arr[i];
        if (sum > leftSum)
            leftSum = sum;
    }

    // Include elements on right of mid
    sum = 0;
    int rightSum = INT_MIN;
    for (int i = m + 1; i <= h; i++) {
        sum = sum + arr[i];
        if (sum > rightSum)
            rightSum = sum;
    }

    // Return the sum of maximum left, right, and
    // cross subarray
    return (leftSum + rightSum);
}
```