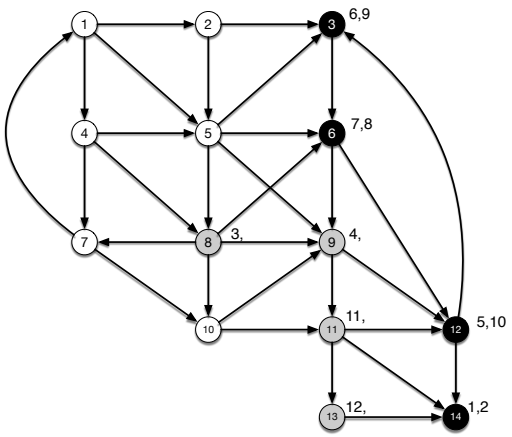
Final Solutions

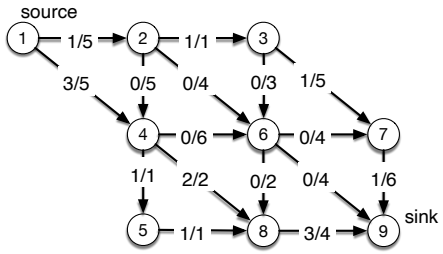
Problem 1:

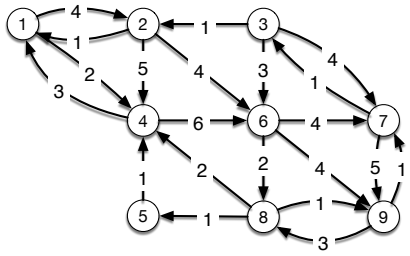
Since $12 = 2^3 + 4$, the split pointer is 4 and the level is 3. Since 30%8 = 6, which is ≥ 4, the bucket address for a key hash of 30 is 6. Similarly, 33 → Bucket 1, 36 → Bucket 4, 39 → Bucket 7, 42 → Bucket 10, 45 → Bucket 5, and 48 → Bucket 0.

Problem 2:



Problem 3:





Problem 4:

We make this into a graph-coloring problem. The stations are the vertices of the graph. The conflicts are represented as edges of the graph. A channel assignment is a color assigned to a node. Two nodes joined by an edge cannot be in the same color. This is the independent set problem, which reduces to 3-SAT.

Problem 5:

In any parenthesization, there is a final multiplication, which writes the product as

$$(M_1 \cdot \ldots \cdot M_k) \cdot (M_{k+1} \cdot \ldots \cdot M_n)$$

Here, k can range from 1 to $n-1$ as neither the left nor the right sub-product can be empty. After deciding where to put the final multiplications, we have to parenthesize the left and the right product. We can read of for the number of parenthesization of n matrices as

$$\Pi(n) = \sum_{k=1}^{n-1} \Pi(k) \cdot \Pi(n-k).$$

We also need to put in an initial value, which is $\Pi(1) = 1$.

To implement this efficiently, we need to use memoization. In Python, this is done with the `@functools.cache` decorator:

```
import functools

@functools.cache
def par(n):
    if n == 1:
        return 1
    return sum([par(k)*par(n-k) for k in range(1, n)])
```

Problem 6:

We are doing at most constant work for each element in the array. Thus $O(\text{len(array)})$.

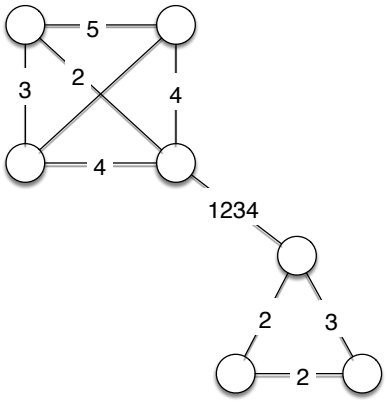
```
def max_sub(array):
    if len(array) == 1:
        return array[0], array[0], array[0], array[0]
    half_index = len(array)//2
    left, right = array[:half_index], array[half_index:]
    if not left:
        return max_sub(right)
    if not right:
        return max_sub(left)
    lbest, ltotal, lleft, lright = max_sub(left)
    rbest, rtotal, rleft, rright = max_sub(right)
    bbest = max([lright+rleft, ltotal+rtotal, lbest, rbest])
    btotal = ltotal+rtotal
    bleft = max(lleft, ltotal+rleft)
    bright = max(rright, lright+rtotal)
    return bbest, btotal, bleft, bright
```

Problem 7:

The recursion is given by $T(n) = 2T(n/2) + O(1)$, which according to the MT is $\Theta(n)$.

Problem 8:

Counterexamples are only formed if the maximum weight edge is a bridge, i.e. if removing it would result into a disconnected graph. For example:



Problem 9:

```
@functools.cache
def coins(amount):
    if amount < 0:
        return False
    if amount == 0:
        return True
    return coins(amount-13) or coins(amount-19) or coins(amount-23)
```

Problem 10:

For any index i, $(a_{i+1} - 2 \cdot (i+1)) - (a_i - 2 \cdot i)) = a_{i+1} - a_i + 2i - 2i - 2 \geq 2 - 2 = 0$. Therefore, the array $[a_i - 2i$ for $i \in \{0,1,\ldots,i-1\}]$ is still sorted. We can therefore use binary search for 0. This takes $O(\log(n))$ time.

```
def even(array, low, up):
    if low == up:
        if array[low] == 2*low:
            return low
        else:
            return -1
    if low == up-1:
        if array[low] == 2*low:
            return low
        elif array[up] == 2*up:
            return up
        else:
            return -1
    half = (low+up)//2
    if array[half] == 2*half:
        return half
    elif array[half] > 2*half:
        return even(array, low , half)
    else:
        return even(array, half, up)
```