1. I have a box of three coins: a fair coin, a two-headed coin, and a two-tailed coin. I choose one of the three coins randomly with equal probability and flip it. Given that the outcome is head, what is the probability that I flipped the two-headed coin? Explain why?

The outcomes and their elementary probabilities

Fair $\begin{cases} H_f & 1/6 \\ T_f & 1/6 \end{cases}$

2headed $\begin{cases} H_{h1} & 1/6 \\ H_{h2} & 1/6 \end{cases}$

2-tailed $\begin{cases} T_{t1} & 1/6 \\ T_{t2} & 1/6 \end{cases}$

The outcome is head : $E = \{H_f, H_{h1}, H_{h2}\}$

The two headed coin is tossed $F = \{H_{h1}, H_{h2}\}$

$$P(F|E) = \frac{P(F \cap E)}{P(E)} = \frac{2}{3}$$

2. Suppose you are given a **non-negative** weighted graph $G(V, E)$. Let $s, t \in V$ be two vertices of G. The **bottleneck** of a path from s to t is the weight of the maximum-weight edge on the path. For a positive real number w, a path from s to t is called a **bottleneck-w** path if the bottleneck of the path is $\leq w$. Asnwer the following quetions regarding bottleneck shortest paths.

   (a) Given an undirected non-negatively weighted graph $G(V, E)$, two vertices $s, t \in V$, and a **bottleneck** w, find the **bottleneck-w** shortest path from s to t.

   (b) Given an undirected non-negative weighted graph $G(V, E)$, two vertices $s, t \in V$, find the shortest path from s to t with the **minimum bottleneck**, i.e., the goal is to minimize the maximum-weight edge of the shortest path.

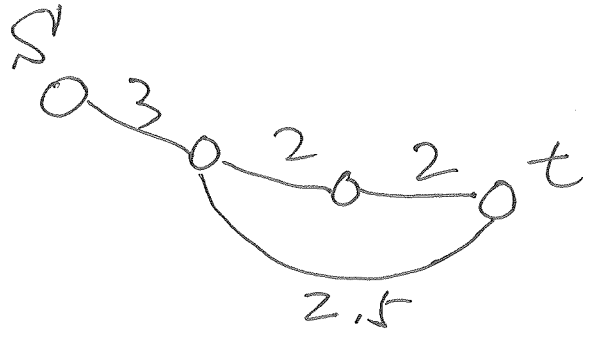
Solutions:

(a) Remove all edges with weight $> w$ and run Dijkstra's Algorithm using s as the source.

(b) Do binary search on w, ~~for each~~ say w_0 find the smallest w such that there is a bottleneck-w path from s to t.

Find the bottleneck w_0 shortest path from s to t as in (a)

Note: MST will not work.



3. Suppose you are given an undirected graph $G(V, E)$. For a vertex $v \in V$, the **neighbors** of v are all the vertices that are adjacent to v not including v. We define the **neighborhood degrees (ND)** of v to be the sum of the all the degrees of the neighbors of v. (See Figure 1 for illustrations.)

   For this problem you are asked to design an $O(|V| + |E|)$ algorithm for calculating the neighborhood degrees for every vertices in V. More specifically, the input of your algorithm is an undirected simple graph $G(V, E)$, and the output is an array $ND[1..|V|]$ that store the **neighborhood degrees (ND)** of every vertex of V.

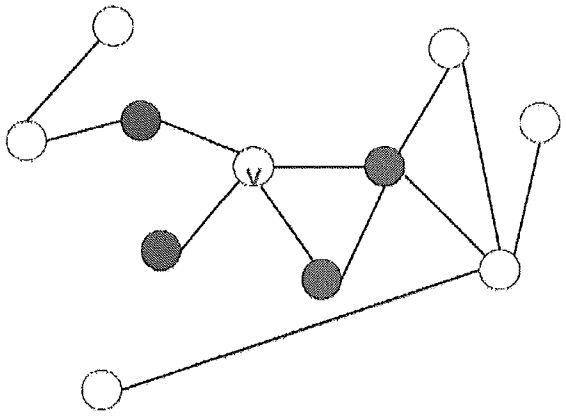


Figure 1: Illustrating the concepts of neighbors and neighborhood degrees. The **neighbors** of vertex v are shaded. The **neighborhood degree** of v is 9.

Solutions.

(1) Calculate the degrees of all vertices if not provided.

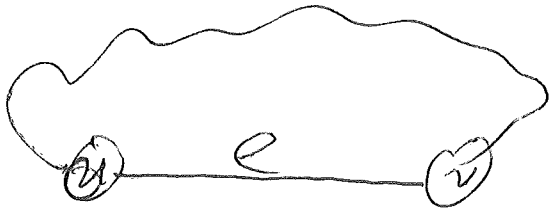
(2) Run BFS.

4. Given an **undirected unweighted** graph $G(V, E)$. Design an efficient algorithm to find a simple cycle with the minimum number of edges.

   Hint, the running time of your algorithm should be $O(|V|^4)$.

Solution: (1) if the graph has self loop or multiedge, trivial.

(2) consider one of such shortest simple cycle connecting two vertices u and v



Observe that if we "remove" the edge e connecting u, v, the remaining portion of the cycle is a shortest path from u to v.

This immediate implies that we can solve the problem by iterating through every edge in the graph by

(a) removing the edge

(b) run BFS on the remaining graph.

The total runtime is bounded by $O(|V|^4)$ assuming the graph is dense, i.e., $|E| = O(|V|)$

5. Recall that there are two types of randomized algorithms, the **Las Vegas algorthms** and the **Monte Carlo algorithms**. A **Las Vegas algorithm** always produces the correct result, however its running time may vary. A **Monte Carlo algorithm** on the other hand has a deterministic running time, but the output may be incorrect with a certain (typically small) probability.

   It is possible to convert a Las Vegas algorithm to a Monte Carlo algorithm using a trick called "**early termination**". Consider the randomized quick sort algorithm. Recall that this is a Las Vegas algoroithm with an expected running time of $2n \ln n$, where n is the input size. Suppose that when running the randomized quick sort, we always terminate the execution in at most $10n \ln n$ iterations. We call this algorithm the **Monte Carlo Quick Sort algorithm**. Answer the following questions regarding the Monte Carlo Quick Sort algorithm:

   (a) What is the probability that the Monte Carlo Quick Sort algorithm produces the correct result?

   (b) How long it takes to check if the result is correct?

   (c) If we run this Monte Carlo Quick Sort algorithm 10 times, what is the probablity of getting at least one correct result from these 10 runs?

Solution:

(a) Let X be the r.v. denoting the running time of Monte Carlo Quick Sort.

Then $Pr(X \geq 10n\ln n) \leq \frac{2n\ln n}{10n\ln n} = 0.2$

Thus the probability of getting the correct result is $\geq 1 - 0.2 = 0.8$

(b) Linear time

(c) $1 - 0.2^{10}$