Question 1

A coin has $1/4$ chance of heads and $3/4$ chance of tails.

a) Expected number of tosses needed for heads?

b) Suppose it is tossed n times? What is the probability that the number of heads equals the number of tails?

---

a) The expected value of a geometrically distributed random variable is $1/p$ such that p is the probability of success on each Bernoulli trial.

As our question is in the form of a Bernoulli trial, we can use this: Expected num of tosses $= \frac{1}{\frac{1}{4}} = 4$

(Bernoulli trial as binary outcomes with each trial being independent with constant probability!)

So, expected number of tosses to get a heads is 4. //

---

b) Consider the following formula for a binomial distribution:

$$Pr(X=x) = \binom{n}{x} p^{x} q^{n-x}$$

where n = # of trials
x = # of successes

Since our situation has two outcomes, has finite trials, has trials independent of each other, and has probability remain constant for each trial, we can use this to find the probability our number of heads match our tails.

Let p be the $Pr(\text{heads})$ $\wedge$ q be $Pr(\text{tails})$

Next we will find our x s.t. $x = n - x$ as this will ensure our outcomes for heads and tails match.

$$x = n - x$$

$$2x = n$$

$$x = \frac{n}{2} \quad \Rightarrow n - x = n - \frac{n}{2} = \frac{n}{2}$$

Note

$\frac{n}{2} \Rightarrow$ even # of trials required

So,

$$Pr(X = x) = \binom{n}{x} p^{x} q^{n-x}$$

$$\Rightarrow Pr\left(X = \frac{n}{2}\right) = \binom{n}{\frac{n}{2}} (0.25)^{\frac{n}{2}} (0.75)^{\frac{n}{2}}$$

Probability # of heads = # of tails:

$$Pr\left(X = \frac{n}{2}\right) = \binom{n}{n/2} (0.25)^{n/2} (0.75)^{n/2}$$

//

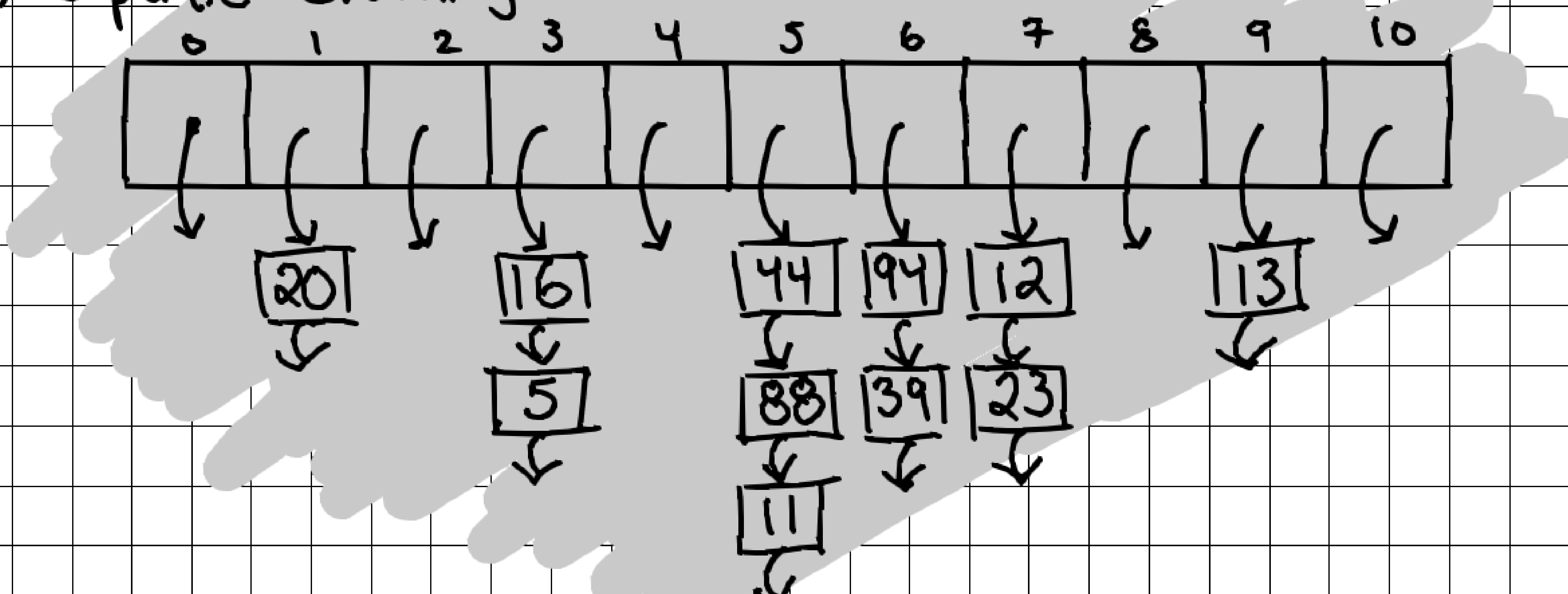
Question 2 Hash the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5 with the following hash function $h(k) = (2k+5) \bmod 11$ and schemes outlined in each part.

First, I will show the co-responding list of where each key hashes to wrt $h(k)$:

[12, 44, 13, 88, 23, 94, 11, 39, 20, 16, 5]

[7, 5, 9, 5, 7, 6, 5, 6, 1, 4, 4]

a) Seperate Chaining



In seperate chaining our collision scheme is just to add to the linked list at the index we originally hash to.

b) Linear Probing

0	1	2	3	4	5	6	7	8	9	10
11	39	20	5	16	44	88	12	23	13	94

In linear probing, if we have a collision we search for the next free index to place the key.

collisions: 88 → 5 → next free 6
23 → 7 → next free 8
94 → 7 → next free 10
11 → 5 → next free 0
39 → 6 → next free 1
20 → 1 → next free 2
5 → 4 → next free 3

c) Quadratic Probing

0	1	2	3	4	5	6	7	8	9	10
39	20	5	11	16	44	88	12	23	13	96

Similar to linear probing, except we check next collision based on $h(k) + i^2 \bmod 11$ i is the number of collisions for this insert.

collisions

88 → 5 filled → $5 + 1^2 = 6$ free
23 → 7 filled → $7 + 1^2 = 8$ free
96 → 6 filled → $6 + 1^2 = 7$ filled → $6 + 2^2 = 10$ free
11 → 5 filled → $5 + 1^2 = 6$ filled → $5 + 2^2 = 9$ filled → $(5 + 3^2) \bmod 11 = 3$ free
39 → 6 ... → $(6 + 3^2) \bmod 11 = 4$ filled → $(6 + 4^2) \bmod 11 = 0$ free
5 → 4 ... → $(4 + 2^2) \bmod 11 = 8$ filled → $(4 + 3^2) \bmod 11 = 2$ free

d) Double Hashing with $h'(k) = 7 - (k \bmod 7)$

k	12	44	13	88	23	94	11	39	20	16	5
h(k)	7	5	9	5	7	6	5	6	1	4	4
h'(k)	2	5	1	3	5	4	3	3	1	5	2

0	1	2	3	4	5	6	7	8	9	10
11	23	20	16	39	44	94	12	88	13	5

Our scheme uses i # of collisions s.t. $h(k) + i \times h'(k) \bmod 11$

Collisions

88 → 5 filled → $(5+3) \bmod 11 = 8$ free
23 → 7 filled → $(7+5) \bmod 11 = 1$ free
11 → 5 filled → $(5+3) \bmod 11 = 8$ filled → $(5+3\times2) \bmod 11 = 0$ free
39 → 6 filled → ... $(6 + 3\times3) \bmod 11 = 4$ free
20 → 1 filled → $(1+1) \bmod 11 = 2$ free
16 → 4 filled → $(4+5) \bmod 11 = 9$ filled → $(4+5\times2) \bmod 11 = 3$ free
5 → 4 filled → ... $(4 + 2\times3) \bmod 11 = 10$ free

Question 3 Prove any connected, undirected graph, has a vertex whose removal, along w/ its incident edges, will not disconnect the graph by designing a DFS algorithm to find such a vertex

On a high level consider G it it were a tree. Then, the removal of any leaf node and its edges would result in the rest of the graph being connected. And as there exists a spanning tree of every strongly connected graph, there then exist a vertex whose removal would leave any graph, G, connected.

Lets make this more formal by considering a generic DFS algorithm that does this:

```
Find Vertex (vertex v, tree T, graph G)
    if(nodes.T = nodes G - 1) return v
    for node n adjacent to v in G not already in T do
        add n to T from v
        FindVertex(n)
    forEnd
Find Vertex End
```

Considering what this algorithm is doing before it finds our vertex E' is important, so lets run through that.

It takes in a graph, G, a starting vertex, v, and an empty tree, T. Then, checks if the size of T is 1 less than size of G and returns the vertex if so. Otherwise it adds an adjacent vertex of v in G that isn't already in T into T and then calls itself recursively onto the adjacent vertex. Effectively, it creates a DFS tree until there remains one vertex in G that DNE in T.

So let the algorithm be called on a connected, undirected graph G, starting from vertex E. (Also let G be finite!) Now consider the vertex found, call it E', to not have the properties we seek. That is, its removal would cause our graph, G, to be disconnected upon its removal, along with its incident edges.

$\Rightarrow \nexists$ a spanning tree of G as our algorithm would have found it (a DFS tree)

$\Rightarrow G$ is not connected... contradiction!

$\Rightarrow$ our assumption was incorrect $\therefore$ we can remove at least 1 vertex from any G s.t. G remains connected!

Proof w/o algorithm (This is cleaner IMO)

Let G be a finite, undirected, connected Graph. Prove $\exists$ a vertex, call it v, s.t. its removal would leave G connected.

As G is a connected, undirected Graph $\exists$ a spanning tree of G, call it T.

By definition, T has leaf nodes, s.t. the removal of any of these nodes would still leave the rest of the spanning tree traversable.

As every other node exists in a traversable tree, $\exists$ a vertex from every finite, undirected, connected Graph s.t. its removal (and its incident edges) leaves the graph connected. ∎

Question 4 use Prim's algorithm to create the MST of the following graph:

$$V = \{A, B, C, D, E\}$$

(A, B, 7) (B, C, 4) (C, E, 3) (D, E, 2)
(A, C, 5) (B, D, 7)
(A, D, 1) (B, E, 1)

Initial Values:

$D(A) = 0$, $D(B) = \infty$, $D(C) = \infty$, $D(D) = \infty$, $D(E) = \infty$

-> lowest value is of node A, so add A to tree T

-> Update values (if lower than current $D(v)$ value) of adjacent nodes to A not in T [D(B), D(C), D(D)]

Values after A in T (A, A, 0):

$D(A)=0$, $D(B) = 7$, $D(C) = 5$, $D(D) = 1$, $D(E) = \infty$

-> lowest value of node st node DNE in T is D, so add D to T

-> update D(E) as $(D, E, 2) \wedge 2 < \infty$

Values after D in T (A, D, 1):

$D(A) = 0$, $D(B) = 7$, $D(C) = 5$, $D(D) = 1$, $D(E) = 2$

-> lowest value of node not in T is E, add E to T

-> update [D(B), D(C)] as $(B, E, 1) \wedge 1 < 7$ and $(C, E, 3) \wedge 3 < 5$

Values after E in T (D, E, 2):

$D(A) = 0$, $D(B) = 1$, $D(C) = 3$, $D(D) = 1$, $D(E) = 2$

→ lowest $D(u)$ of node not in T is B, so add B to T

→ Values dont need to be updated after B in T as no new lower edges

After B in T $(B, E, 1)$:

$D(A) = 0$, $D(B) = 1$, $D(C) = 3$, $D(D) = 1$, $D(E) = 2$

→ lowest value of node not in T is C, add C to T

→ no nodes left, so T is now MST!

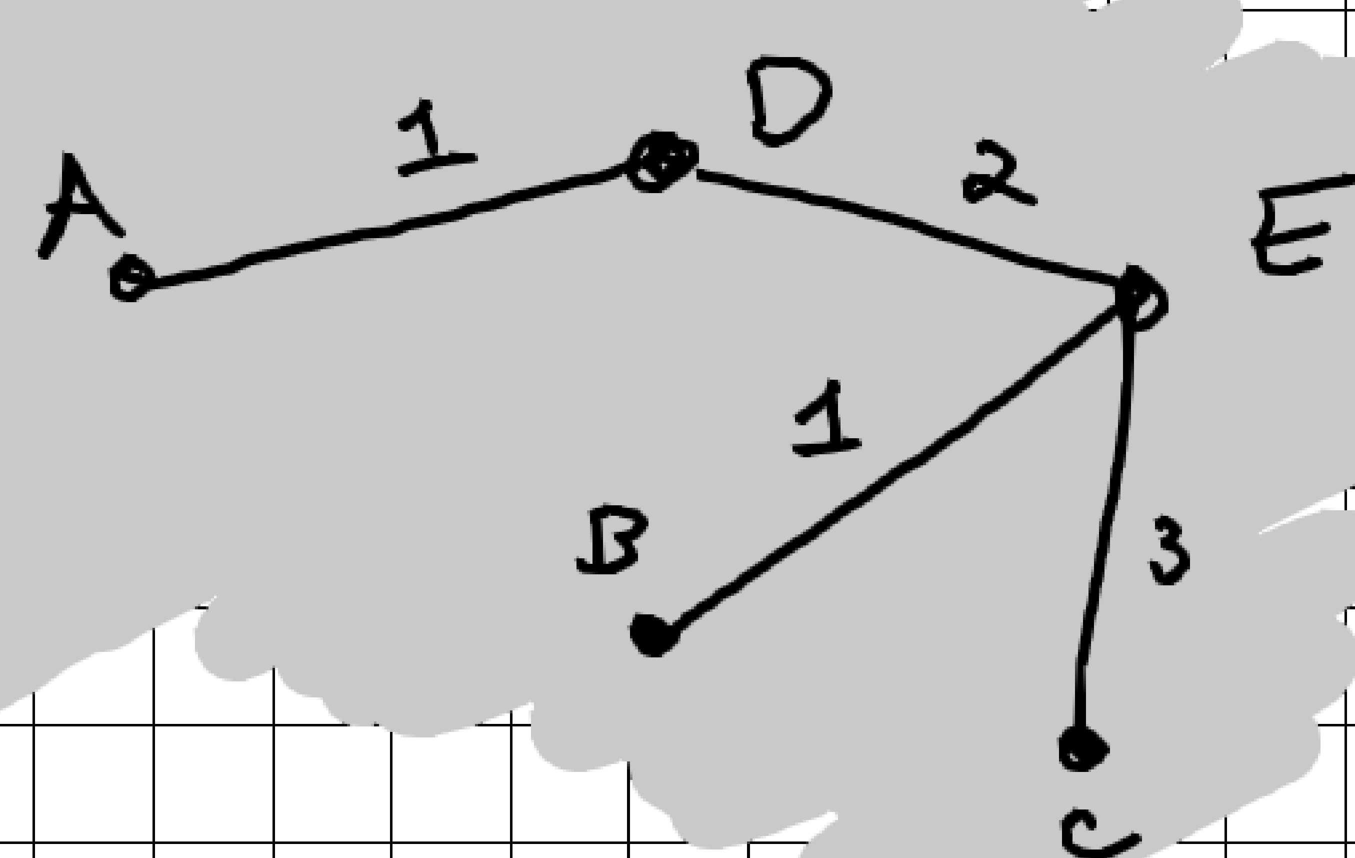
Values at end:

$D(A) = 0$, $D(B) = 1$, $D(C) = 3$, $D(D) = 1$, $D(E) = 2$

$w(T) = 0 + 1 + 3 + 1 + 2 = 7$

MST:

$$\left.\begin{array}{l}(A, D, 1)\\(D, E, 2)\\(B, E, 1)\\(C, E, 3)\end{array}\right\} \Rightarrow$$



Question 5a Show you can rescale the edge weight (all) by adding a positive constant, and this will not impact MST of a graph

Let T be the MST of the weighted, connected, undirected graph G w/ edge weights $\{w_1 \ldots w_n\}$ $\wedge$ let T' be the MST of G' w/ edge weights $\{w_1+C \ldots w_n+C\}$ s.t $0<C$.

Let V be the vertices of $G \wedge G'$ and x be some arbitrary subset of those vertices.

Let us assume $T \neq T' \Rightarrow T \neq$ MST of G'

Consider then the edges between $x \wedge x \setminus V$, let us denote them as the set $S = \{e_i : e_1 \ldots e_k\}$

As $T \neq$ MST of $G' = T'$, then by the cut property $\exists$ some x s.t $\exists$ some edge (call it E) st $E \in T' \wedge E \notin T \wedge [W(E)+5 < w(e_i)+5$ for all e_i in $S \setminus E]$

As if $E \in T$ $\forall x$ then T = MST of G'.

But, if $E \notin T$ $\forall x$ then instead of edge E in T $\exists$ some other edge, e_i of $S \setminus E$... but by the cut property if $E \notin T \wedge e_i \in T$ then $T \neq$ MST of G as well!

(As $w(E) < w(e_i)$)... Contradiction as $T =$ MST of G!

$\therefore$ Our assumption was incorrect and $T = T' =$ MST of G'. $\blacksquare$

Recap of 5a in less formal ways:

The difference between all the weights still has the same delta when you add the same positive constant to all the weights, so then the delta between any possible edge in any possible cut of the graph would still be the same, and then the chosen edge would be the same one.

Question 5b Show that Prim's algorithm still works w/ graphs w/ negative edge weights

---

First lets talk on a less formal level...
Prim's algorithm takes the lowest edge weight between the cut of edges in T and edges in G\T so at every given chance it would still make the correct choice, which remains the greedy choice!

Let G be a undirected, weighted, and connected graph, and let T be the tree on G via Prim's. Note, this graph can have negative weight edges. We want to prove T = MST of T.

Assume $\exists$ T' st T' = MST of G
Consider the first instance where T differs from T', WRT the construction of T using Prim's, and let the vertices in T up till this point be known as the set χ. Also, let the edge chosen by Prim's be known as E and E' for the edge that connects χ ^ $\chi \setminus V$ in T'...
As they both connect the same subsets, we can replace E' in T' w/ E, and this new version has a lower weight.

But, as T' was already an MST of G, this cannot be as the MST should already have lowest possible weight of all spanning trees of G.

∴ Our assumption was incorrect

⇒ Prim's algorithm still finds MST of a graph G that contains negative edge weights.

∎