1. Suppose we insert \( n \) distinct items into an initially empty hash table of size \( m \gg n \), using an ideal random hash function \( h \).

(a) What is the exact expected number of collisions?

**Solution:** Let \( X \) denote the number of collisions. For each pair \( \{x, y\} \), let \( X_{x,y} = \left[ h(x) = h(y) \right] \). Then we have

\[
E[X] = \sum_{\{x, y\}} E[X_{x,y}] \quad \text{[linearity of expectation]}
\]

\[
= \sum_{\{x, y\}} \Pr[h(x) = h(y)] \quad \text{[definition of } X_{x,y} \text{]}
\]

\[
= \sum_{\{x, y\}} \frac{1}{m} \quad \text{[ideal random]}
\]

\[
= \frac{\binom{n}{2}}{m} = \frac{n(n-1)}{2m} \quad \text{[counting]}
\]

**Rubric:** 1 point. No proof is required.

(b) Estimate the probability that there are no collisions. [Hint: Use Markov's inequality.]

**Solution:** Markov's inequality implies \( \Pr[X \geq 1] \leq E[X] = \frac{n(n-1)}{2m} \), and therefore

\[
\Pr[X = 0] = 1 - \Pr[X \geq 1] \geq 1 - \frac{n(n-1)}{2m}.
\]

Since the hash function is ideal random, we can actually compute the exact probability that there are no collisions as follows. Index the items \( x_1, x_2, \ldots, x_n \) in their order of insertion into the hash table, and for each index \( i \), let \( X_i \) denote the number of collisions in the subset \( \{x_1, x_2, \ldots, x_i\} \). In particular, \( X = X_n \) and \( X_0 = 0 \).

\[
\Pr[X_n = 0] = \Pr[X_0 = 0] \cdot \prod_{i=0}^{n-1} \Pr[X_{i+1} = 0 \mid X_i = 0]
\]

\[
= \prod_{i=0}^{n-1} \frac{m-i}{m} \quad \text{[counting]}
\]

\[
= \frac{m!/(m-n)!}{m^n}
\]

(There are many other ways to derive this expression.) We can simplify this expression using the World’s Most Useful Inequality \( 1 + x \leq e^x \):

\[
\Pr[X_n = 0] \leq \prod_{i=0}^{n-1} \left( 1 - \frac{i}{m} \right) \leq \prod_{i=0}^{n-1} e^{-i/m} = e^{\sum_{i=0}^{n-1} -i/m} = e^{-\binom{n}{2}/m}
\]

This upper bound almost exactly matches the lower bound we obtained from Markov’s inequality (thanks to the close approximation \( 1 + x \approx e^x \)), but that lower bound is both easier to derive and easier to use in part (c), which is why we gave the hint.

**Rubric:** 2 points. Any one of the boxed solutions is fine. No proof is required.
(c) Estimate the largest value of $n$ such that the probability of having no collisions is at least $1 - 1/n$.

Solution: Using our approximation from Markov’s inequality, we need to find the largest value of $n$ such that

$$1 - \frac{n(n-1)}{2m} \geq 1 - \frac{1}{n}.$$ 

Simple algebra implies this inequality holds if and only if $n^2(n-1) \leq 2m$, so we require $n = O(m^{1/3})$.

Rubric: 3 points. No proof is required.

(d) Fix an integer $k > 1$. Estimate the largest value of $n$ such that the probability of having no $k$-way collisions is at least $1 - 1/n$.

Solution (partial credit): If there is a $k$-way collision, then there are at least $\frac{k}{2}$ (two-way) collisions. Markov’s inequality implies that

$$\Pr \left[ X \geq \frac{k}{2} \right] \leq \frac{E[X]}{\binom{k}{2}} = \frac{n(n-1)}{k(k-1)m}.$$

Simple algebra implies

$$\frac{n(n-1)}{k(k-1)m} \leq \frac{1}{n} \iff n^2(n-1) \leq k(k-1)m.$$

Thus, we require $n = O(k^{2/3}m^{1/3})$.

Solution (full credit): Let $X_k$ denote the number of $k$-way collisions. Ideal randomness implies that any set of $k$ elements defines a $k$-way collision with probability $1/m^k$. Thus, the expected number of $k$-way collisions is exactly $\binom{n}{k}/m^k$. Markov’s inequality implies that

$$\Pr[X_k \geq 1] \leq \frac{E[X_k]}{\binom{n}{k}} = \frac{O(n^k)}{m^{k-1}}.$$

(Here I’m assume $k$ is a constant, so that we can fold the dependence on $k$ into the $O( )$ notation.) Thus, we need to find the largest $n$ such that $n^k/m^{k-1} \leq 1/n$. Simple algebra implies

$$\frac{n^k}{m^{k-1}} \leq \frac{1}{n} \iff n^{k+1} \leq m^{k-1}.$$

Thus, we require $n = O(m^{(k-1)/(k+1)})$.

Rubric: 4 points. No proof is required.

• No penalty for folding the factor $(k!)^{1/(k+1)} = O(k)$ into the $O( )$ notation.
• 3 points for observing that $\Pr[X_k \geq 1] = \binom{n}{k}/m^{k-1}$.
• −1 for assuming the probability of a $k$-way collision is $1/m^k$ instead of $1/m^{k-1}$.
• 2 points for the looser upper bound $O(k^{2/3}m^{1/3})$. 

2
2. Describe and analyze an algorithm to compute the maximum number of people that can walk from the Earth fountain to the Fillory fountain in $h$ hours, without anyone alerting the Beast or turning into a niffin.

**Solution:** Assume that $G$ is a directed graph. If not, replace each undirected edge with a pair of opposing directed edges. This transformation allows two people $x$ and $y$ to pass through the same gate in opposite directions at the same time. However, if there is an optimal solution where that happens, there is another optimal solution where $x$ and $y$ do not pass through that gate, then $x$ follows the rest of $y$’s optimal path, and $y$ follows the rest of $x$’s optimal path. Thus, any optimal solution for the directed-graph problem can be transformed into an optimal solution for the undirected-graph problem.

Given the input graph $G(V,E)$ and the integer $h$, we define a new directed graph $G' = (V',E')$ as follows:

- $V'$ contains all pairs $(v,i)$, where $v \in V$ and $i$ is an integer between 0 and $h$. Intuitively, $(v,i)$ denotes “fountain $v$ after $i$ hours”.
- $E'$ contains two types of edges:
  - An edge $(u,i-1)\rightarrow(v,i)$ with capacity 1, for every edge $u\rightarrow v \in E$ and every index $i$. Traversing this edge corresponds to passing through the gate $u\rightarrow v$ during the $i$th hour.
  - An edge $(v,i-1)\rightarrow(v,i)$ with capacity $\infty$, for every vertex $v \in V$ and every index $i$. Traversing this edge indicates staying in plaza $v$ (not passing through any gate) during the $i$th hour.

We can build $G'$ in $O(V' + E')$ time by brute force.

We now need to find the value of the maximum flow in $G'$ from $(Earth,0)$ to $(Fillory,h)$. We can computing this value in $O(V'E')$ time using Orlin’s algorithm. Altogether our algorithm runs in $O(V'E') = O(hV \cdot h(V + E)) = O(h^2VE)$ time.

- Let $f^*$ be a maximum flow in $G'$. Because all capacities are integers, we can assume without loss of generality that $f^*(e)$ is an integer for every edge $e \in E'$. By the flow decomposition theorem, we can decompose $f^*$ into $|F^*|$ overlapping paths, each carrying one unit of flow. Each path has the form

$$(v_0, 0)\rightarrow(v_1, 1)\rightarrow(v_2, 2)\cdots\rightarrow(v_{h-1}, h-1)\rightarrow(v_h, h)$$

where $v_0 = Earth$ and $v_h = Fillory$ and for each index $i$, either $v_i = v_{i+1}$ or $v_i \rightarrow v_{i+1}$ is an edge in $E$. A person can follow this path from the Earth fountain to the Fillory fountain without turning into a niffin: for each index $i$, if $v_{i-1} \neq v_i$, they go through the gate from fountain $v_{i-1}$ to fountain $v_i$ during the $i$th hour; otherwise, they stay at fountain $v_{i-1} = v_i$ for the $i$th hour. Because $f^*$ is feasible, at most one flow path goes through each gate edge $(u,i)\rightarrow(v,i+1)$, and therefore at most one person goes through the gate $u\rightarrow v$ during hour $i$, so the Beast is never alerted. We conclude that at least $|f^*|$ people can walk from Earth to Fillory.

- Suppose on the other hand that $k$ people can safely walk from Earth to Fillory. We can record each person’s trajectory as a path in $G'$ of the form

$$(v_0, 0)\rightarrow(v_1, 1)\rightarrow(v_2, 2)\cdots\rightarrow(v_{h-1}, h-1)\rightarrow(v_h, h)$$
where \( v_0 = \text{Earth} \) and \( v_h = \text{Fillory} \) and for each index \( i \), either \( v_i = v_{i+1} \) or \( v_i \to v_{i+1} \) is an edge in \( E \). Specifically, if the person walks through the gate from fountain \( u \) to fountain \( v \) in the \( i \)th hour, the path contains the edge \((u, i-1)\to(v, i)\)—which must be an edge in \( G' \) because nobody turned into a niffin—and if they stay next to fountain \( v \) during the \( i \)th hour, the path contains the edge \((v, i-1)\to(v, i)\). Because the Beast is never alerted, at most one of these paths goes through any gate edge \((u, i-1)\to(v, i)\).

Thus, summing these paths yields a feasible flow in \( G' \) from \((\text{Earth}, 0)\) to \((\text{Fillory}, h)\). We conclude that the maximum flow value in \( G' \) is at least \( k \).

We conclude that the number of people who can walk from the Earth fountain to the Fillory fountain in \( h \) hours is equal to the the value of the maximum flow in \( G' \) from \((\text{Earth}, 0)\) to \((\text{Fillory}, h)\).

\[\text{Rubric: Standard reduction rubric. No proof of correctness is required.}\]

- \(-2\) for omitting edges \((\text{Earth}, i-1)\to(\text{Earth}, i)\) and \((\text{Fillory}, i-1)\to(\text{Fillory}, i)\)
- No penalty for omitting other edges \((v, i-1)\to(v, i)\). I don’t know whether these edges are actually necessary!
- Max 5 points for an incorrect algorithm that is correct when the graph is a path.
- 0 points for an algorithm that is not even correct when \( G \) is a path of length 2.

**Common incorrect solution (local edge capacity):** Let \( G \) be the input graph, let \( s \) be the vertex corresponding to the Earth fountain, and let \( t \) be the vertex corresponding to the Fillory Fountain. Let \( \text{dist}(u, v) \) denote the length (number of edges) of the shortest path from \( u \) to \( v \) in \( G \). For each edge \( u \to v \), assign a capacity of

\[c(u \to v) \leftarrow \max\left\{ 0, h - \text{dist}(s, u) - \text{dist}(v, t) \right\};\]

this is the largest number of people that can walk from \( s \) to \( t \) via \( u \to v \) in \( h \) hours. We can easily compute these capacities by running BFS twice, once from \( s \) and once (traversing all edges backward) from \( t \). Finally, compute and return the value of the maximum \((s, t)\)-flow in \( G \) with these capacities. The algorithm runs in \( O(VE) \) time.

The problem here is that the shortest paths used to define the capacities can share edges. The capacities are correct for each edge individually, but collectively they allow flows that cannot be realized by a legal schedule of people walking from Earth to Fillory. Consider the graph below when \( h = 5 \); numbers on the edges are the capacities computed by this algorithm. This algorithm would return 6, but the correct answer is actually 4.

![Diagram](attachment:image.png)

Variants of this solution set the capacity of \( u \to v \) to the number of people that can reach either \( u \) or \( v \) from Earth in \( h \) hours, or the number of people that can reach Fillory from either \( u \) or \( v \) in \( h \) hours; some variants assigned these capacities to the vertices instead of the edges. All these variants also yield incorrect solutions for the graph above.
Common incorrect solution (disjoint paths): Let $G$ be the input graph, let $s$ be the vertex corresponding to the Earth fountain, and let $t$ be the vertex corresponding to the Fillory Fountain. Assign each edge a capacity of 1. Compute a maximal set of edge-disjoint paths in $G$ from $s$ to $t$, by computing an integral maximum flow and then computing a flow decomposition. Let $\ell_1, \ell_2, \ldots, \ell_k$ be the lengths of these edge-disjoint paths. Return $\sum_i \max\{0, h - \ell_i + 1\}$, which is the largest number of people that can get from Earth to Fillory in $h$ hours by following these paths. The algorithm runs in $O(VE)$ time.

The paths that make up a maximum flow are not necessarily short. It is easy to construct examples where this algorithm could return 0 even though it is possible for at least one person to walk from Earth to Fillory. For example, given the graph below and $h = 6$, the algorithm might return 0, but the correct answer is 5.

A few people suggested returning $\sum_i \lfloor \ell_i/h \rfloor$, which is incorrect even when $G$ is just a path.

I do not know whether there is always an optimal solution that consists of a collection of edge-disjoint paths in $G$, with people moving along each path as quickly as possible.

![Counterexample for the disjoint-path algorithm](image)

Common incorrect solution (niffins everywhere!): Let $G$ be the input graph, let $s$ be the vertex corresponding to the Earth fountain, and let $t$ be the vertex corresponding to the Fillory Fountain. Assign each edge a capacity of $h$. Compute and return the value of the maximum $(s, t)$-flow in $G$. The algorithm runs in $O(VE)$ time.

This solution does not enforce the rule that each person can pass through at most one gate per hour—everybody turns into a niffin! For example, if $G$ is a path of length 2, with Earth at one end and Fillory at the other, and $h = 5$, the correct answer is 4, but this algorithm returns 5.

![Counterexample for the naive flow algorithm](image)

Rubric: Maximum 5 points. −2 for assigning capacities to vertices instead of edges.

Rubric: Maximum 5 points. −2 for finding vertex-disjoint paths instead of edge-disjoint paths. 0 points if the solution is not correct for a path of length 2.

Rubric: 0 points.
3. Suppose we store a set $S$ of $n$ items in a Bloom filter $B[1..m]$ using $k = (m/n)\ln 2$ independent hash functions.

(a) First your boss suggests simply discarding half of the Bloom filter, keeping only the subarray $B[1..m/2]$. Describe an algorithm to check whether a given item $x$ is an element of $S$, using only this smaller Bloom filter.

Solution:

```plaintext
HALF_BLOOM_LOOKUP(x):
    for $i \leftarrow 1$ to $k$
        if $h_i(x) \leq m/2$ and $B[h_i(x)] = 0$
            return False
    return True
```

(b) What is the probability that your algorithm returns True when $x \notin S$?

Solution: For each index $i$, we have $\Pr[h_i(x) \leq m/2] = 1/2$ (because $h_i$ is ideal random) and $\Pr[B[h_i(x)] = 0] = p = 1/2$ (by definition of $p$). Moreover, these two events are independent (again, because $h$ is ideal random), so the probability of returning False in the $i$th iteration is 1/4. Different iterations of the main loop are independent (again, because $h$ is ideal random), so the overall probability of returning True is $(3/4)^k = (3/4)^{(\lg 2)(m/n)}$.

More generally, if the fraction of 0-bits in the original Bloom filter is $p$, the false positive rate for the smaller Bloom filter is $(1 - p/2)^k$.

Rubric: 3 points for part (a) + 2 points for part (b). Credit for part (b) requires the false-positive rate to match the algorithm submitted in part (a). Correct algorithms with higher false positive rates are worth fewer points. In particular, “(a) return True; (b) 1”, which is technically correct but worthless, is worth 1½ points.
(c) Next your boss suggests merging the two halves of your old Bloom filter, defining a new array \( B'[1..m/2] \) by setting \( B'[i] \leftarrow B[i] \lor B[i + m/2] \) for all \( i \). Describe an algorithm to check whether a given item \( x \) is an element of \( S \), using only this smaller Bloom filter \( B' \).

**Solution:**

```plaintext
MERGEBLOOMLOOKUP(x):
   for i ← 1 to k
      j ← h_i(x)
      if j > m/2
         j ← j − m/2
      if B[j] = 0
         return False
   return True
```

(d) What is the probability that your algorithm returns \text{True} when \( x \notin S \)?

**Solution:** For any index \( j \), we have

\[
\Pr[B'[j] = 0] = \Pr[B[j] = 0 \land B[j + m/2] = 0] \\
= \Pr[B[j] = 0] \cdot \Pr[B[j + m/2] = 0] \quad (\star) \\
= p^2 \\
= 1/4.
\]

For each index \( i \), the “hash” address \( j \) is uniformly distributed between 1 and \( m/2 \) (because \( h \) is ideal random). So \textit{MERGEBLOOMLOOKUP} acts just like the standard lookup algorithm in a Bloom filter with \( p = 1/4 \). It follows that the probability of a false positive is (approximately\(^1\)) \( (3/4)^k = (3/4)^{lg(m/n)} \).

More generally, if the fraction of 0-bits in the original Bloom filter is \( p \), the false positive rate for the smaller Bloom filter is (approximately\(^1\)) \( (1 − p^2)^k \).

\(^1\)Technically, we’re cheating in line \( (\star) \)—the random bits that make up a Bloom filter are not actually independent—but we’re cheating in exactly the same way that the original Bloom filter analysis cheats! To be more formal: If we assume that the number of 0 bits in original Bloom filter contains is precisely \( pm \), and those 0 bits are uniformly distributed, then we actually have

\[
\Pr[B[j] = 0 \land B[j + m/2] = 0] = \binom{pm}{2} \cdot \frac{pm(pm−1)}{m(m−1)} = p^2 - \frac{1−p}{m−1},
\]

which is very slightly smaller than \( p^2 \) but approaches \( p^2 \) as \( m \) grows to infinity. When \( p = 1/2 \), this gives us the slightly higher false positive probability \( (3/4 + 1/(2m−2))^k = (3/4)^k + O(k/(2m)^k) \). The actual number of 0 bits is not \( pm \) but a random variable, which is very tightly concentrated around its expected value \( pm \), so even this estimate is only a close approximation.

Rubric: 3 points for part (a) + 2 points for part (b). Credit for part (b) requires the false-positive rate to match the algorithm submitted in part (a). Correct algorithms with higher false positive rates are worth fewer points. In particular, “(a) return \text{True}; (b) 1”, which is technically correct but worthless, is worth 1½ points.
4. Describe and analyze an efficient algorithm to solve the escape problem for \( m \) terminals in an \( n \times n \) grid.

**Solution:** Let \( G = (V, E) \) be the \( n \times n \) grid graph; let \( T \subseteq V \) be the set of terminal vertices; and let \( B \subseteq V \) be the subset of boundary vertices. We construct a new undirected graph \( G' \) from \( G \) by adding a new source vertex \( s \) connected to every terminal in \( T \), and a new target vertex connected to every boundary vertex in \( B \). A solution to the escape problem corresponds to a set of \( m \) vertex-disjoint \((s, t)\)-paths in \( G' \). We can compute the maximum number of vertex-disjoint \((s, t)\)-paths using Orlin’s algorithm, as described in class, in \( O(VE) = O(n^4) \) time.

**Solution (+2 extra credit):** We reduce the problem to vertex-disjoint paths as in the previous solution, but we use Ford-Fulkerson instead of Orlin’s algorithm to solve the resulting flow problem. Because every escape path must lead to a distinct boundary vertex, the maximum number of escape paths in \( G \), and therefore the maximum number of vertex-disjoint paths in \( G' \), is at most \( 4n \). Thus, Ford-Fulkerson finds these paths after at most \( 4n \) iterations, so the entire algorithm runs in only \( O(En) = O(n^3) \) time.²

**Rubric:** Standard reduction rubric. Full credit for \( O(n^4) \) time. +2 extra credit points for \( O(n^3) \) time.

²If we assume that the input to the escape problem is a single integer \( n \) and the list of \( m \) terminal vertices, then this algorithm runs in exponential time, at least when \( m \) is very small. With more effort, the running time can be reduced to a polynomial in \( m \), even if we require an explicit description of the paths! I should really make this a homework problem.