11.2-1

hash function $h$ to hash $n$ keys into an array $T$ of length $m$.

Expected cardinality of $\sum_{k \neq 1} E[X_k] = E[\sum_{k \neq 1} X_k] = \sum_{k \neq 1} E[X_k] = \sum_{k \neq 1} \frac{1}{m}$

Define a random variable $X_k = \begin{cases} 1 & \text{if } h(k) = h(1) \\ 0 & \text{otherwise} \end{cases}$

Assuming simple uniform hashing, $P[h(k) = h(1)] = \frac{1}{m}$, $E[X_k] = \frac{1}{m}$

$E[\sum_{k \neq 1} X_k] = \sum_{k \neq 1} E[X_k] = \sum_{k \neq 1} \frac{1}{m}$

$\sum_{k \neq 1} = \text{number of pairs } k \neq 1 = \binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2}$

Expected value $E[\sum_{k \neq 1} X_k] = \frac{n(n-1)}{2m}$

2. Size stack $A = n$; size stack $B = m$

(a) Worst case running times

- Multipop $A \Rightarrow n$ pops $\Rightarrow O(n)$
- Multipop $B \Rightarrow m$ pops $\Rightarrow O(m)$
- Transfer $\Rightarrow n$ pops & $n$ pushes $\Rightarrow O(n)$

(b) Define $\Phi(n,m) = 3n + m$, initially both stacks are $0$, so $\Phi(Id_0) = 0$

- Also $\Phi(Id_0) = \Phi(Id)$

Push $A$: $1 + [3(n+1) + m] - [3n + m] = 4 \Rightarrow O(1)$

Push $B$: $1 + [3n + (m+1)] - [3n + m] = 2 \Rightarrow O(1)$

Multipop $A$: define $k = \min \{k, n\}$, $k' + [3(n-k') + m] - [3n + m] = -2k' \Rightarrow O(1)$

Multipop $B$: define $k = \min \{k, n\}$, $k + [3n + (m-k)] - [3n + m] = 0 \Rightarrow O(1)$

Transfer: define $k = \min \{k, n\}$, $2k' + [3(n-k') + (m+k)] - [3n + m] = 0 \Rightarrow O(1)$
3. 22.1-6

A universal sink: vertex with in-degree \(|V|-1\) and out degree 0

Find a universal sink in \(O(V)\) given an adjacency matrix.

Note: if a vertex is a universal sink, the row is the adjacency matrix sums to zero and its column sums to \(|V|-1\).

Additionally, we see that if an edge exists from \(i\) to \(j\), then \(i\) cannot be a universal sink. The algorithm starts at row 1, col 2. If an entry is equal to 1, we check column. If the entry is zero, we go to the next row. Here is pseudo-code for the algorithm:

\[
i = 1; j = 2; \text{next} = 3 \\
\text{While next } \leq |V| + 1 \text{ do} \\
\quad \text{if } \text{Adj}[i,j] = 1 \text{ then potentialSink} = j; i = \text{next}; \\
\quad \quad \text{else potentialSink} = i; j = \text{next}; \\
\quad \text{next}++; \\
\text{// now check row and column of potentialSink} \\
\text{for } k = 1; k \leq |V|; k++ \\
\quad \text{if } \text{Adj}[\text{potentialSink}, k] \neq 0 \text{ return noSink} \\
\quad \text{if } \text{Adj}[k, \text{potentialSink}] = 1 \text{ and potentialSink} = k \text{ return noSink;}
\]

End

Return UniversalSink = potentialSink

The first while loop takes \(|V|-2\) iterations. The for loop has \(|V|\) iterations with two operations. Therefore, the total running time is \(3|V|-2 \Rightarrow O(|V|)\).
4. a) False, a MST can contain the longest edge in a graph if that edge is the only edge going to a given vertex. For example:

![Graph Diagram]

The edge with weight 1500 will be included in the MST.

b) True, in any cycle, an MST will not contain the longest edge in that cycle. In a cycle, we can remove one edge to break the cycle. This edge will be the longest edge, thereby minimizing the total weight.

c) False, we will not get a minimum directed spanning tree from a root node \( r \) by using Prim's algorithm. For example:

![Directed Graph Diagram]

\( \Rightarrow \) directed tree, all edges lead outward from the root.

Using Prim's algorithm, we add edge \( (r, B) \), then \( (B, C) \), and finally \( (r, A) \) for a cost of \( 5 + 4 + 6 = 15 \).

The actual minimum directed spanning tree would be edges \( (r, B), (r, A), \) and \( (A, C) \) for a cost of \( 5 + 6 + 1 = 12 \).
Topological sort:
C H A D B E G F 

If a directed edge from node F to node A is added, the graph is not a DAG since cycles would now be present, such as the cycle (AD), (D,F), (FA).

Strongly connected components

AB
BG
GF
FA
BE
EG
AD
DG
DF
AG
CB
CE
CH
HG

To start breadth first search from node A:

(A, B, D, F, G, H)
Dijkstra's algorithm cannot be modified to solve this problem. Dijkstra's algorithm assumes the paths that have been explored will always be a minimum. In this problem, adding a new vertex can change the variance. For example, consider the following graph:

Using the algorithm to vertex $w$, the minimum variance is 4. But after exploring the path from $w$ to $u$, path $(P_2, P_3)$ has a variance of $25 - 5 = 20$. Meanwhile, path $(P_1, P_3)$ has a variance of $10 - 3 = 7$. This would be the correct minimum variance path from $v$ to $u$, but it would be using a non-minimum variance path $P_2$. This means the minimum cost variance path might depend upon a non-minimum cost variance path which Dijkstra's greedy algorithm cannot be used to solve this problem.