Exercise XI, Algorithms 2024-2025

These exercises are for your own benefit. Feel free to collaborate and share your answers with other students. There are many problems on this set, solve as many as you can and ask for help if you get stuck for too long. Problems marked * are more difficult but also more fun :).

Shortest Paths

1 (*, Exercise 24.1-6) Suppose that a weighted directed graph $G = (V, E)$ has a negative-weight cycle. Give an efficient algorithm to list the vertices of one such cycle. Argue that your algorithm is correct.

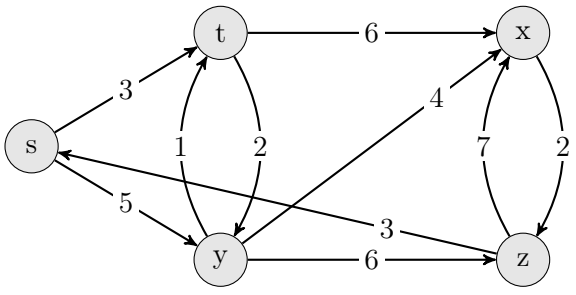
Solution: We have seen in class that the Bellman-Ford algorithm can be used to detect a negative cycle. With a slight modification, we can indeed list all vertices that belong to this negative cycle. (Because we are trying to detect all negative cycles, not just ones reachable from a given source, we can start by adding a new source vertex s and connecting it by a 0-weight edge to each other vertex. Alternatively, we could begin by setting $v.d = 0$ for every v.)

First, there is a negative cycle if and only if after $|V| - 1$ iterations of Bellman-Ford there exists a vertex v that can still be relaxed by an edge (u, v). (This was proved in the lecture.)

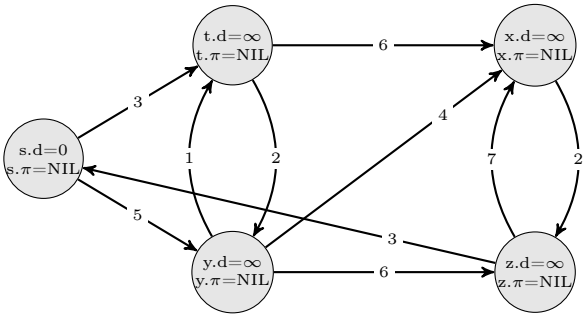
Second, in this case the predecessor array returned by the algorithm will also contain a cycle, and any such cycle will be negative. To see this, assume towards a contradiction that the array contains no cycles. In this case, it represents a directed tree from s to all vertices reachable from s, and we know that this tree has the property that for each vertex w reachable from s, the path from s to w in the tree is a shortest (simple) path from v to w in the graph. This is true after $|V| - 1$ iterations as well as after $|V|$ iterations, implying that the tree cannot change in the $|V|$-th iteration (since any change means making a path shorter). However, it did change – a contradiction.

Therefore, it's enough that we find a cycle in the predecessor array. This can be done in linear time.

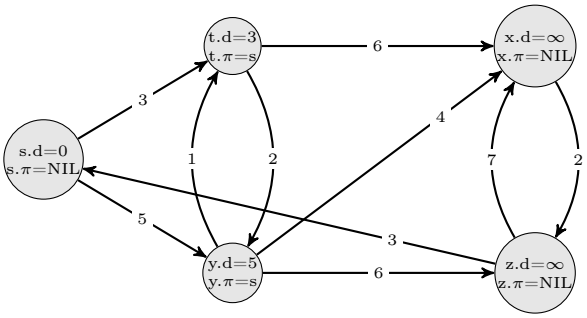
2 (Exercise 24.3-1) Run Dijkstra's algorithm on the following edge-weighted directed graph. Use first vertex s as a source and then vertex z as a source.



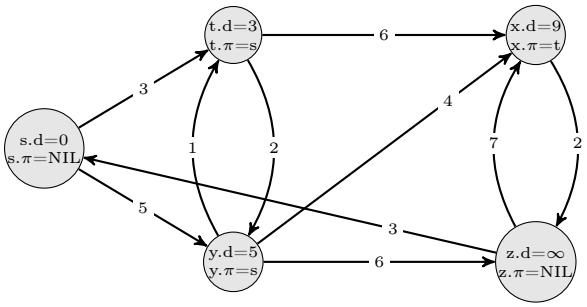
Solution: Let us consider the case where the source is s; the state of the algorithm is initially as follows:



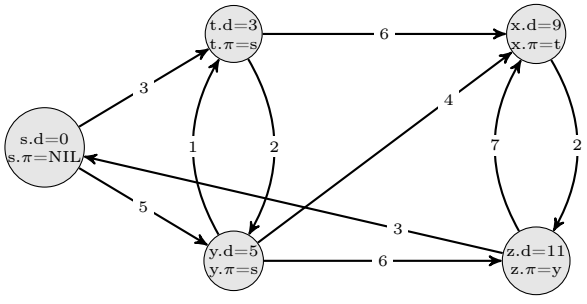
In this case, s will be the first vertex added to the set S used by Dijkstra's algorithm:



Next, t will be inserted into S:

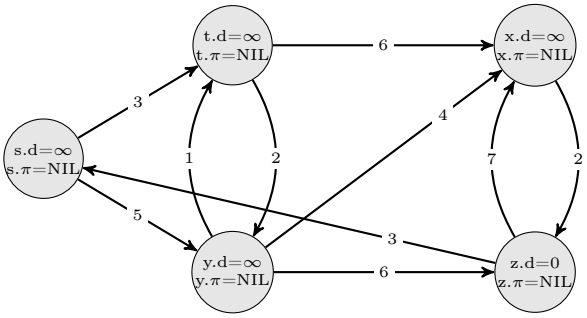


After that, y will be inserted into S:

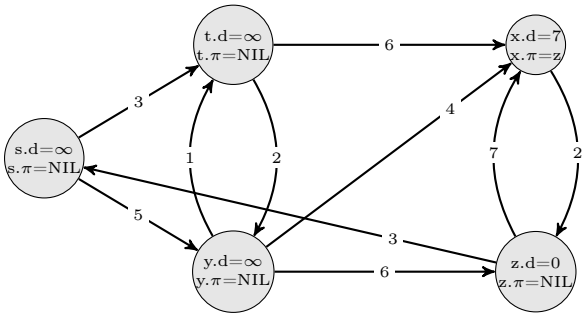


Finally, x and z will be inserted into S (in that order), but no change in the shortest paths structure will occur.

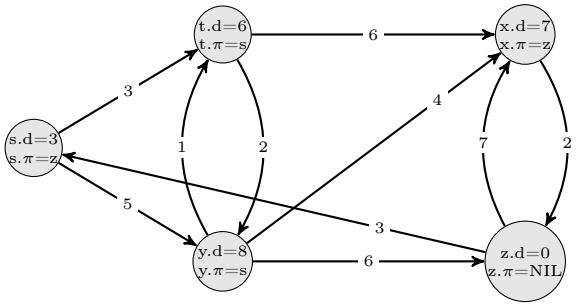
Next, let us consider the case where z is the source:



The first vertex to be put into S will be z:



Next, s will be inserted into S:



After that, t, x and y will be inserted into S, in that order, but there will be no change in the shortest paths structure.

3 (half a *) Suppose you are standing at the top station of Mount-Everest and you wish to ski down. There are several different stations where you can rest (vertices) and different routes between stations (directed edges). All routes go down hill so they are directed from the station of higher altitude to the station of lower altitude. Although there are different routes from the top station to other stations, if one continues to ski down one will sooner or later hit the base camp. As the view is very nice, we wish to make the ski route as long as possible. In other words, design and analyze an efficient algorithm that calculates the *longest* route to ski down starting at the top station of Mount-Everest and ending in the base camp.

Solution: We begin our reasoning by observing the following fact: the graph is acyclic. This follows since any slope is directed from a station of higher altitude to a station of smaller altitude. Therefore the graph does not contain any negative cycles even if we let the weight of each arc be minus its length, i.e., the weight of an arc (u,v) is $-\ell$ if the length of the slope from u to v is ℓ.

Now we solve the shortest path problem (using the Bellman-Ford algorithm) with respect to these weights. As the graph does not contain any negative cycles, the algorithm is guaranteed to find the shortest path from the source to the base camp. That is, it finds the path p that minimizes $\sum_{e \in p} -w(e)$ which is equivalent to maximizing $\sum_{e \in p} w(e)$, which is the length of the path.

The total running time of the algorithm is $O(|V||E|)$.

A small comment is that we heavily relied on that the graph was acyclic to find the longest path. It is strongly believed that there are no efficient algorithms for this problem in general graphs.

Finally, let us remark that a more efficient (and arguably simpler) solution can be designed by dynamic programming. We leave this as an exercise: the approach is basically to compute the longest path from the top to each station in the order of decreasing altitudes of the stations. The solution runs in $O(|V| + |E|)$ time.

Randomness, Hiring problem, Indicator variables, Hash-tables

4 (Exercise 5.2-4) Use indicator random variables to solve the following problem, which is known as the **hat-check problem**. Each of n customers gives a hat to a hat-check person at a restaurant. The hat-check person gives the hats back to the customers in a random order. What is the expected number of customers who get back their own hat?

Solution: Another way to think about the hat-check problem is that we want to determine the expected number of fixed points in a random permutation. (A ***fixed point*** of a permutation π is a value i for which $\pi(i) = i$.) We could enumerate all $n!$ permutations to determine the average number of fixed points per permutation, but this would be a painstaking process, and the answer would turn out to be 1. We can use indicator random variables, however, to arrive at the same answer much more easily.

Define a random variable X that equals the number of customers that get their own hat back, so that we want to compute $E[X]$.

For $i = 1, 2, ..., n$, define the indicator random variable

$X_i = I$ {customer i gets back their own hat}.

Then $X = X_1 + X_2 + ... + X_n$.

Since the ordering of hats is random, each customer has a probability $\frac{1}{n}$ of getting back his or her own hat. In other words, $\Pr\{X_i = 1\} = \frac{1}{n}$, which implies that $E[X_i] = \frac{1}{n}$.

Thus,

$$\begin{aligned} E[X] &= E\left[\sum_{i=1}^{n} X_i\right] \\ &= \sum_{i=1}^{n} E[X_i] \text{ (by linearity of expectation)} \\ &= \sum_{i=1}^{n} \frac{1}{n} \\ &= 1, \end{aligned}$$

and so we expect that exactly 1 customer gets back his own hat.
Note that this is a situation in which the indicator random variables are *not* independent. For example, if $n = 2$ and $X_1 = 1$, then X_2 must also equal 1. Conversely, if $n = 2$ and $X_1 = 0$, then X_2 must also equal 0. Despite the dependence, $\Pr\{X_i = 1\} = \frac{1}{n}$ for all i, and linearity of expectation holds. Thus, we can use the technique of indicator random variables even in the presence of dependence.

5 (Exercise 5.2-2) In the hiring problem, assuming that the candidates are presented in a random order, what is the probability that you hire exactly twice?

Solution: We make three observations:

1. Candidate 1 is always hired.

2. The best candidate, i.e., the one whose rank is n, is always hired.

3. If the best candidate is candidate 1, then that is the only candidate hired.

Therefore, in order for HIRE-ASSISTANT to hire exactly twice, candidate 1 must have a rank $i \leq n-1$ and all candidates whose ranks are $i+1, i+2, ..., n-1$ must be interviewed after the candidate whose rank is n. (When $i = n-1$, this second condition vacuously holds.)

Let E_i be the event in which candidate 1 has rank i: clearly, $\Pr\{E_i\} = \frac{1}{n}$ for any given value of i.

Letting j denote the position (in the interview order) of the best candidate, let F be the event that candidates $2, 3, ..., j-1$ have ranks strictly less than the rank of candidate 1. Given that event E_i has occurred, event F occurs when the best candidate is the first one interviewed out of the $n-i$ candidates whose ranks are $i+1, i+2, ..., n$. Thus, $\Pr\{F|E_i\} = \frac{1}{n-i}$.

Our final event is A, which occurs when HIRE-ASSISTANT hires exactly twice. Noting that events $E_1, E_2, ..., E_n$ are disjoint, we have

$$\begin{aligned} A &= F \cap (E_1 \cup E_2 \cup ... \cup E_{n-1}) \\ &= (F \cap E_1) \cup (F \cap E_2) \cup ... \cup (F \cap E_{n-1}). \end{aligned}$$

and
$\Pr\{A\} = \sum_{i=1}^{n-1} \Pr\{F \cap E_i\}$.

$$\begin{aligned} \Pr\{F \cap E_i\} &= \Pr\{F|E_i\}\Pr\{E_i\} \\ &= \frac{1}{n-i} \cdot \frac{1}{n}. \end{aligned}$$

and so

$$\begin{aligned} \Pr\{A\} &= \sum_{i=1}^{n-1} \frac{1}{n-i} \cdot \frac{1}{n} \\ &= \frac{1}{n} \sum_{i=1}^{n-1} \frac{1}{n-i} \\ &= \frac{1}{n}(\frac{1}{n-1} + \frac{1}{n-2} + ... + \frac{1}{1}) \\ &= \frac{1}{n} \cdot H_{n-1}. \end{aligned}$$

where H_n is the n^{th} harmonic number. Note that this is $\Theta\left(\frac{\log n}{n}\right)$.

6 (Exercise 11.2-1) Suppose we use a hash function h to hash n distinct keys into an array T of length m. Assuming simple uniform hashing, what is the expected number of collisions? More precisely, what is the expected cardinality of $\{\{k, l\} : k \neq l \text{ and } h(k) = h(l)\}$?

Solution: For each pair of keys k, l, where $k \neq l$, define the indicator random variable $X_{kl} = I\{h(k) = h(l)\}$. Since we assume simple uniform hashing, $\Pr\{X_{kl} = 1\} = \Pr\{h(k) = h(l)\} = \frac{1}{m}$, and so $E[X_{kl}] = \frac{1}{m}$.

Now define the random variable Y to be the total number of collisions, so that $Y = \sum_{k \neq l} X_{kl}$. The expected number of collisions is

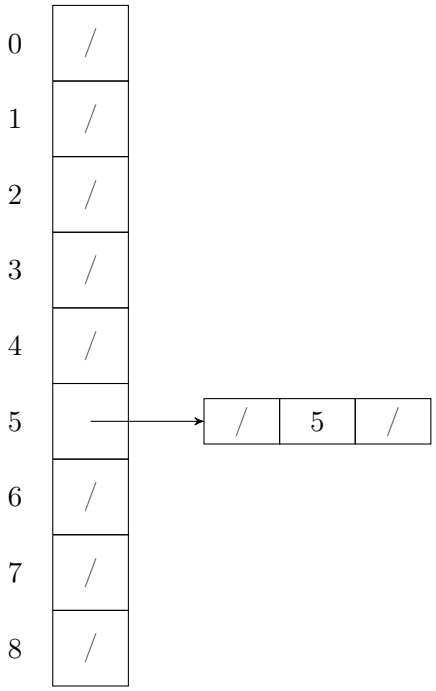
$$
\begin{aligned}
E[Y] &= E\left[\sum_{k\neq l} X_{kl}\right] \\
&= \sum_{k\neq l} E[X_{kl}] \text{ (linearity of expectation)} \\
&= \binom{n}{2}\frac{1}{m} \\
&= \frac{n(n-1)}{2}\frac{1}{m} \\
&= \frac{n(n-1)}{2m}.
\end{aligned}
$$

7 (Exercise 11.2-2) Demonstrate what happens when we insert the keys $5, 28, 19, 15, 20, 33, 12, 17, 10$ into a hash table with collisions resolved by chaining. Let the table have 9 slots, and let the hash function be $h(k) = k \mod 9$.

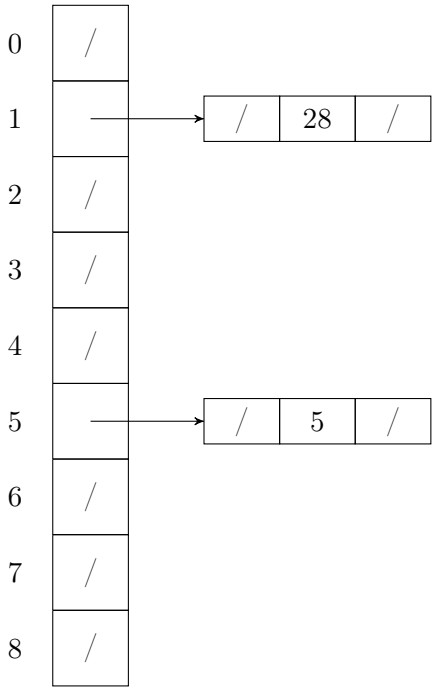
Solution: This is the hash table before any insertions:

0	/
1	/
2	/
3	/
4	/
5	/
6	/
7	/
8	/

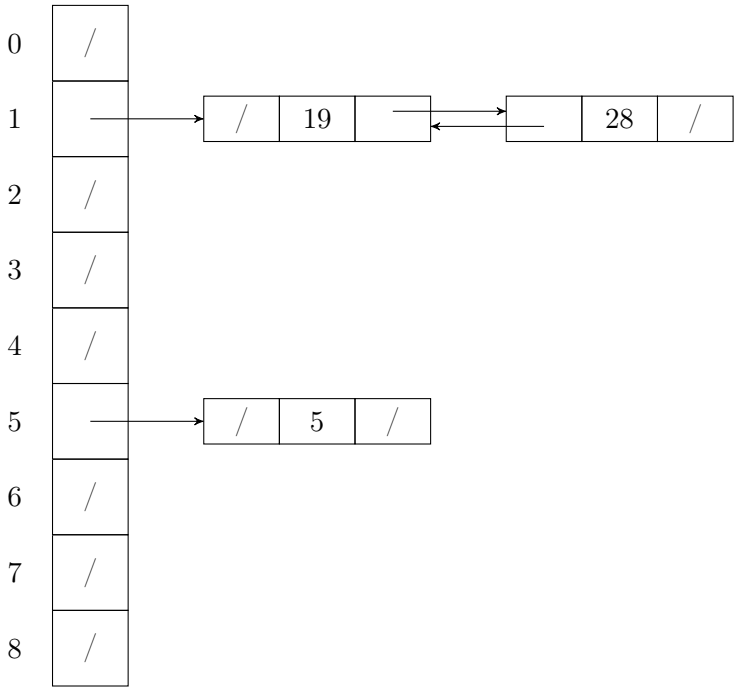
First, we insert key 5:



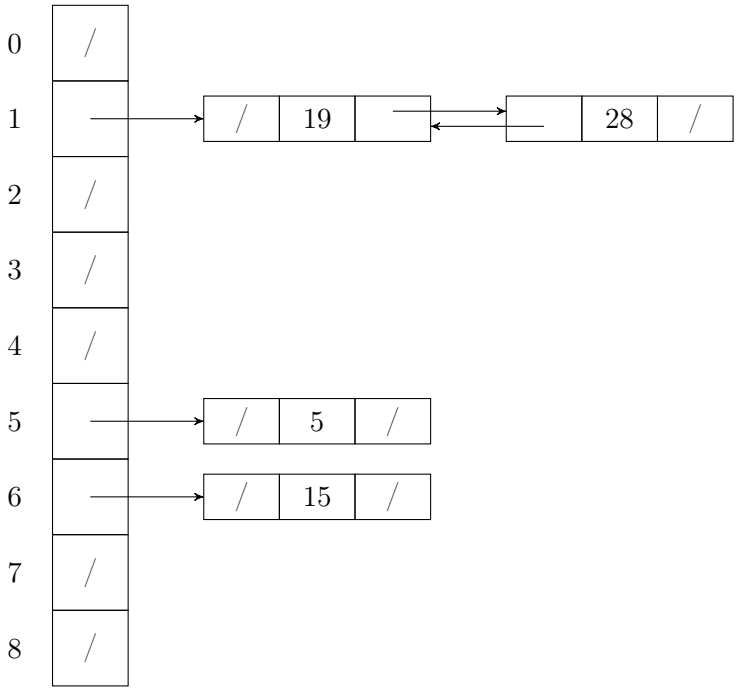
Next, we insert key 28:



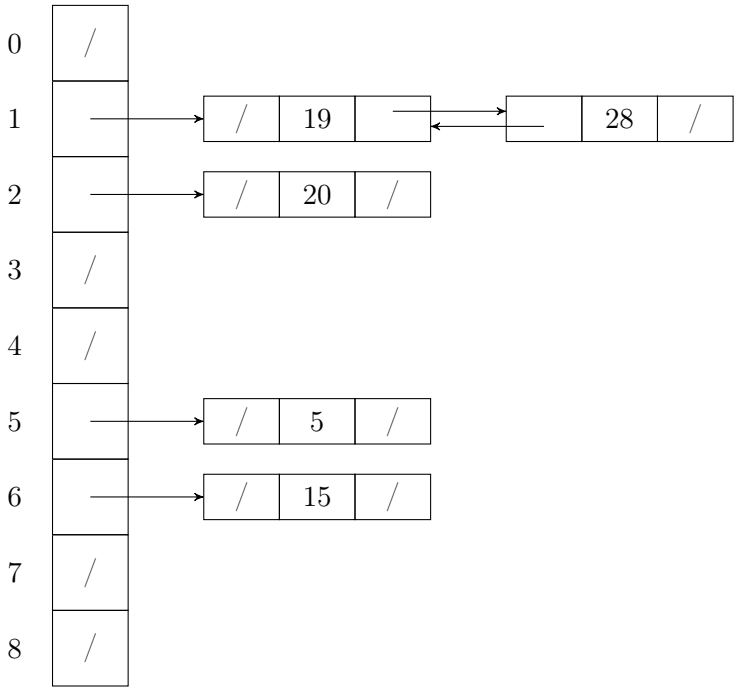
Afterwards, we insert key 19:



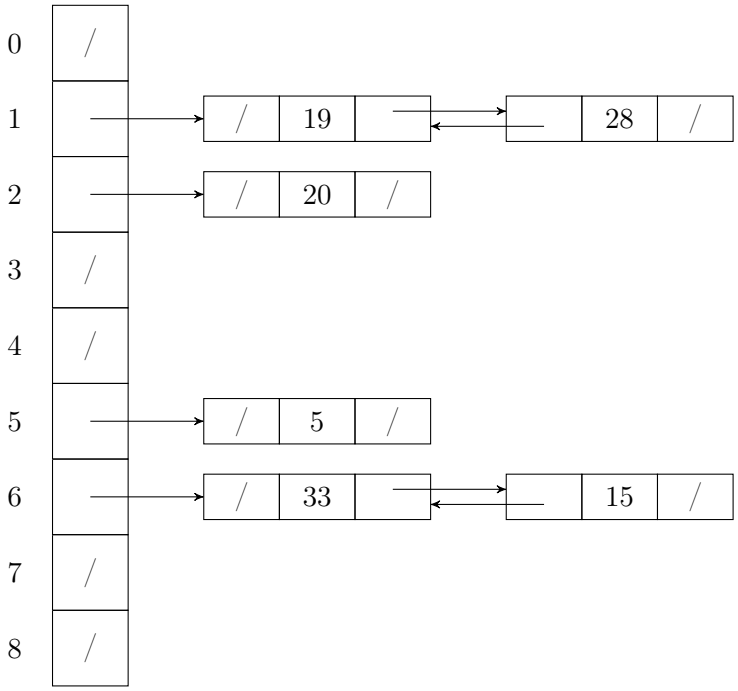
The next key to be inserted is 15:



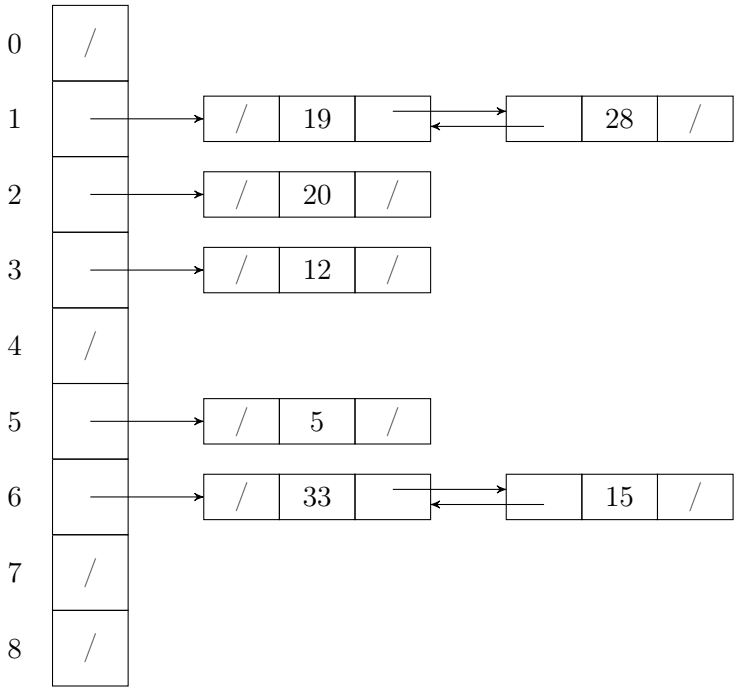
The next inserted key is 20:



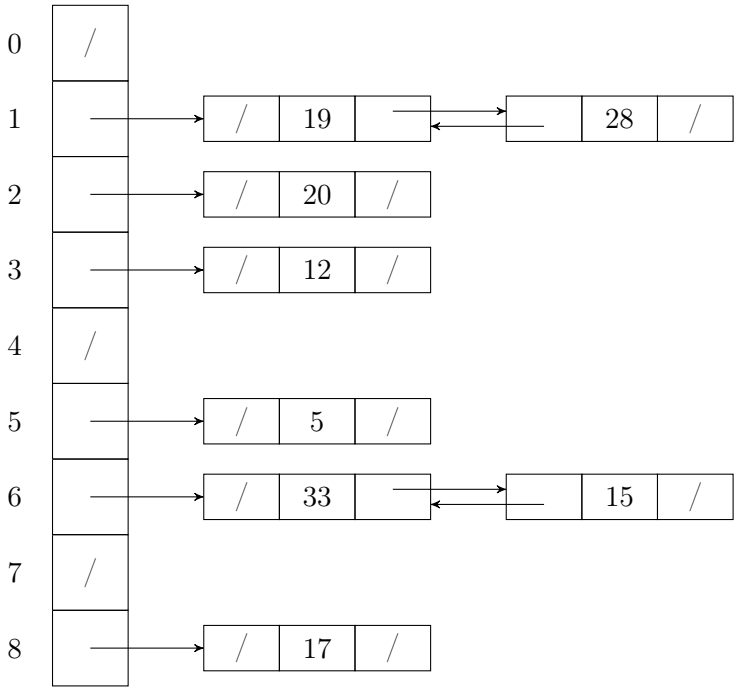
Then follows key 33:



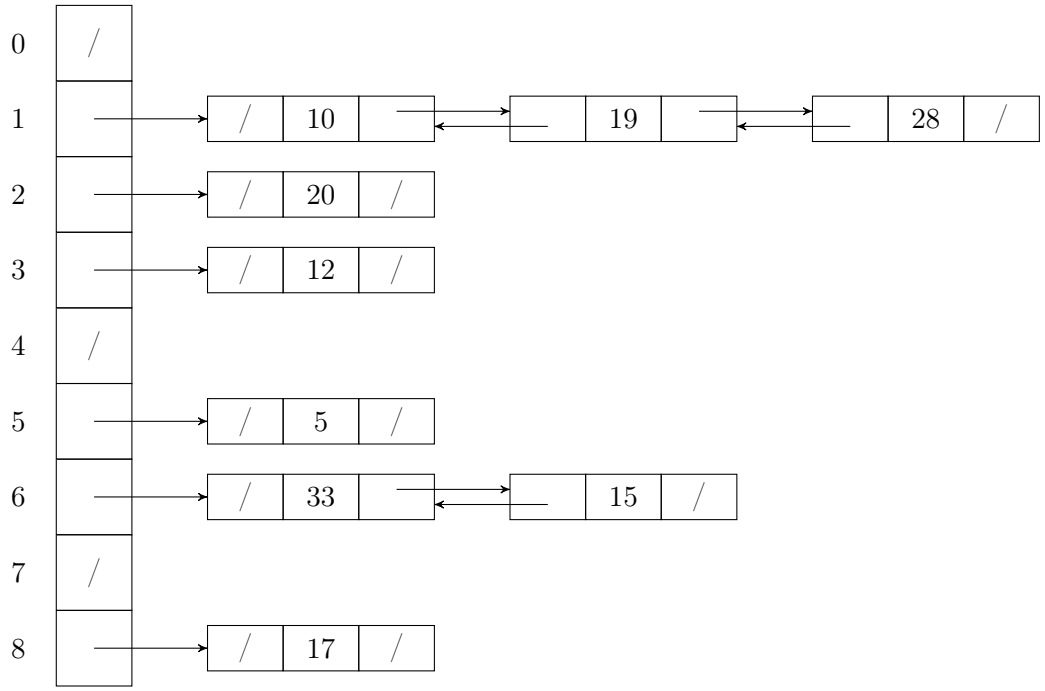
After that, we insert key 12:



Next, we insert key 17:



Finally, we insert key 10:


0
1
2
3
4
5
6
7
8
10
19
28
20
12
5
33
15
17