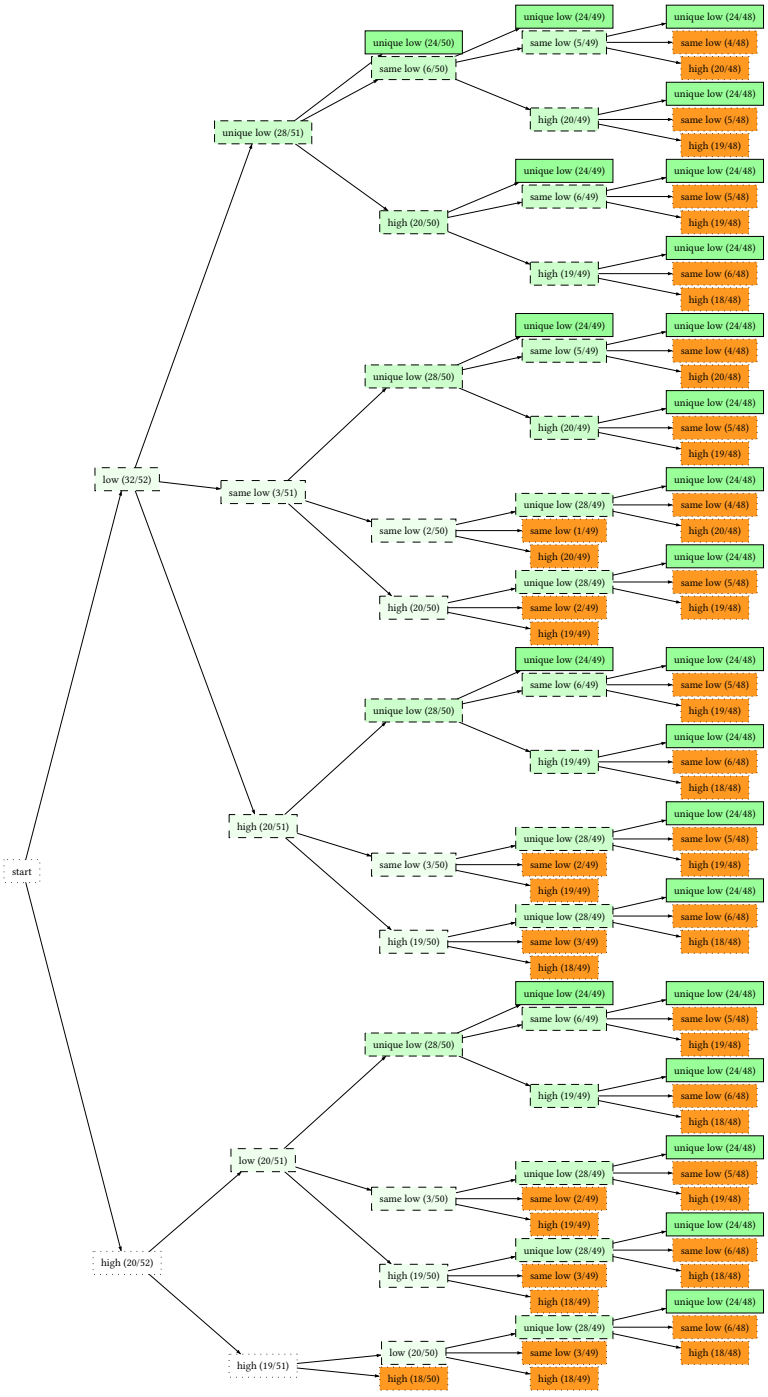
There are eight card values that play in the low hand, and five that don't:

Allowed values: A, 2, 3, 4, 5, 6, 7, 8
Disallowed values: 9, T, J, Q, K

With four suits, this translates to $8 \cdot 4 = 32$ allowed cards, and $5 \cdot 4 = 20$ disallowed cards.

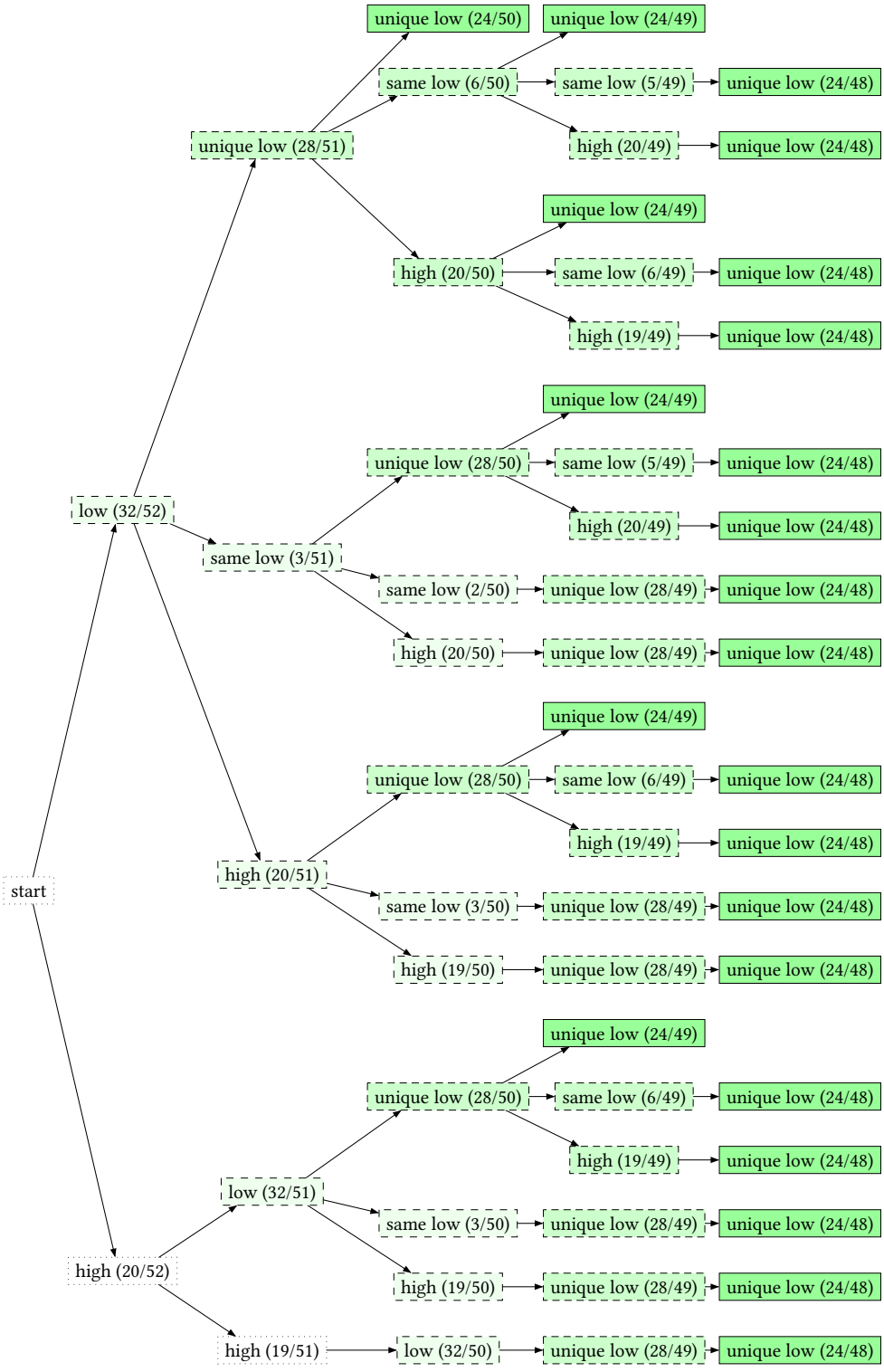
To have a valid low hand, we need to draw three *distinct* low cards from the deck out of the five total cards drawn to make the board.

The following tree represents all possible outcomes as we draw up to five cards (stopping as soon as a low board is either found, or no longer possible):



© 2026 Stefanus Du Toit

We can simplify this probability tree by only keeping the parts that end in a valid low board:



We can now compute the overall probability of getting a board that admits a low hand by multiplying and summing the probabilities along all paths through the tree. This is still quite cumbersome, so let's write some code to compute it instead:

© 2026 Stefanus Du Toit

```
double probability_of_having_a_low_hand(
    const int cards_drawn = 0, const int unique_low_values_seen = 0,
    const int cards_left_matching_already_drawn_low_cards = 0,
    const int high_cards_drawn = 0) {
  const auto num_low_card_values = 8;
  const auto num_high_card_values = 13 - num_low_card_values;
  const auto num_suits = 4;
  const auto remaining_unique_low_cards_to_draw =
      (num_low_card_values - unique_low_values_seen) * num_suits;
  const auto total_high_cards_available =
      num_high_card_values * num_suits - high_cards_drawn;
  const auto cards_remaining = 52 - cards_drawn;
  if (unique_low_values_seen == 3) {
    return 1; // We've found a valid board for a low hand.
  }
  if (cards_drawn == 5) {
    return 0; // We've drawn all the cards we can without enough distinct low cards.
  }
  return
      // we draw a new unique low card. This gives us 3 additional options to
      // draw existing values.
      (remaining_unique_low_cards_to_draw > 0
           ? remaining_unique_low_cards_to_draw / (double)cards_remaining *
                 probability_of_having_a_low_hand(
                     cards_drawn + 1, unique_low_values_seen + 1,
                     cards_left_matching_already_drawn_low_cards + 3,
                     high_cards_drawn)
           : 0)
      // we draw one of the same low cards we've already seen
      + (cards_left_matching_already_drawn_low_cards > 0
             ? cards_left_matching_already_drawn_low_cards /
                   (double)cards_remaining *
                   probability_of_having_a_low_hand(
                       cards_drawn + 1, unique_low_values_seen,
                       cards_left_matching_already_drawn_low_cards - 1,
                       high_cards_drawn)
             : 0)
      // we draw a high card
      + total_high_cards_available / (double)cards_remaining *
            probability_of_having_a_low_hand(
                cards_drawn + 1, unique_low_values_seen,
                cards_left_matching_already_drawn_low_cards,
                high_cards_drawn + 1);
}
```

Running this code, we get:

```
probability of a low hand being possible: 60.09%
```

© 2026 Stefanus Du Toit

We can confirm this number by running a Monte Carlo simulation using the following code[1] :

```
void hilo_montecarlo() {
  const int samples = 10'000'000;
  int valid_low_boards = 0;
  const auto all_low_values =
      std::set{Value('A'), Value('2'), Value('3'), Value('4'),
               Value('5'), Value('6'), Value('7'), Value('8')};
  for (int i = 0; i < samples; ++i) {
    Deck deck;
    deck.shuffle();
    const Hand hand{
        {deck.draw(), deck.draw(), deck.draw(), deck.draw(), deck.draw()}};
    std::set<Value> low_values_seen;
    for (const auto card : hand.cards) {
      if (all_low_values.contains(card.value)) {
        low_values_seen.insert(card.value);
      }
    }
    valid_low_boards += low_values_seen.size() >= 3;
  }
  const auto probability = valid_low_boards / (double)samples;
  std::println("(monte carlo) probability of a low hand being possible: "
               "{:.2f}% ({}/{})\n",
               probability * 100, valid_low_boards, samples);
}
```

And indeed using 10,000,000 samples, with 2 digits of precision after the decimal point, we arrive at the same answer as our recursive implementation:

```
(monte carlo) probability of a low hand being possible: 60.09% (6008644/10000000)
```

What about after the flop? After the turn?

We can modify our recursive function above to stop when we've reached 3 or 4 cards instead, e.g.:

```
if (cards_drawn == 4) { // 5 for the river, 4 for the turn, 3 for the flop
  return 0; // We've drawn all the cards we can without enough distinct low cards.
}
```

This gives us the probabilities of already being able to make a low hand on the flop (e.g. 8♥ 6♦ A♠) or the turn (e.g. 8♥ 6♦ A♠ Q♣) :

```
probability of a low hand already being possible by the flop: 16.22%
probability of a low hand already being possible by the turn: 39.05%
```

[1]The code for the `Value`, `Card`, `Deck` and `Hand` classes isn't shown. Hopefully their function is pretty clear. For completeness, `Deck::shuffle()` randomizes the order of the deck, and `Deck::draw()` draws a card from the top.

© *2026 Stefanus Du Toit*