JUMANJI - The snake and ladder version

To tackle this problem you need to have some knowledge on simple graph theory concepts like vertices and edges . Here we need to map the problem into a directed graph . First you need to create the graph and the board. To create the board you can use a vector or an array of size n according to the given end index. When there is a snake as in the problem statement you are going to a lower index that means there must be an edge starting from snakes mouth to its tail .
(snake mouth) ---→ (snake tail) . And when there is a ladder there must be an edge starting from ladder bottom to top. (bottom of the ladder) → (top of the ladder).
As I mentioned above we need to map all the indexes to a directed graph accordingly skipping cursed indexes.

```
int min_jumps(int n, vector<pair<int, int> > Ladders, vector<pair<int, int> > Snakes, vector<int> broken, int
maxVal)
{
   vector<int> board(n + 1, 0);
   vector<int> cursedIndexes(n + 1, 0);
   for (int i = 0; i < broken.size(); i++)
   {
      cursedIndexes[broken[i]] = 1;
   }
   // board to graph conversion
   for (auto sp : Snakes)
   {

      int s = sp.first;
      int e = sp.second;
      if (!(cursedIndexes[s]) && !(cursedIndexes[e-s]))
      // if (!(cursedIndexes[s]))
         board[s] = e-s;
   }

   for (auto lp : Ladders)
   {
      int s = lp.first;
      int e = lp.second;
      if (!(cursedIndexes[s]) && !(cursedIndexes[e+s]))
      // if (!(cursedIndexes[s]))
         board[s] = e-s;
   }

   // Graph
   Graph g(n + 1);
   for (int u = 1; u < n; u++)
   {
      for (int jmp = 1; jmp <= maxVal; jmp++)
      {
         int v = u + jmp;
         v += board[v];
         if (v <= n && (!cursedIndexes[v]))
         {
            g.addEdge(u, v);
```

```
            }
        }
    }
    return g.minCostBFS(1, n);
}
```

And once you have mapped the snake and ladders board to a graph you just have to call a shortest path algorithm like bfs or dijkstra's algorithm .