Problem 1: 2 Card Poker

2 card poker is a simple card game. You will design a version that can be played against a computer (the server). Each player (the player and the computer) will be dealt two cards. The player will then be shown their cards where they can decide to either play or fold. If the player folds, they lose $1. If the player plays, then their cards are compared to the computers and if they are better, the player wins $10. If they are worse, the player loses $10. A player should start with $100. If they run out of money, or wish to stop playing, the program should end. If they wish to continue playing, all of the cards will be reshuffled and the game will be dealt again.

The ranking of hands goes as follows (from best to worst)

1. Pairs (two cards of the same value)
2. Flushes (two cards of the same suit)
3. High Card (the highest card with ties being broken by the second highest card)

If two players have a pair, then the player with the highest value card wins. The same is true of flushes. You can use the following sudo code to score hands:

SCORE
score = (high card * 15) + low card

if(pair) → score += 10000
else if (flush) → score += 1000

You need to implement a multi threaded server client application that allows users to play two card poker against a computer. Note: The computer makes no decisions (you can make it play intelligently if you would like, but it is not required).

You program should consist of:

- Server
- Game
- Card
- Deck
- GameProxy

Note: This is not an exhaustive list, but the bare minimum required to complete this problem. You should find other things you will need along the way.

Problem 2: Minimum Spanning Tree

Using the modified AdjacencyGraph implementation provided in the repository, create an algorithm that can determine the weight of the minimum spanning tree for a given graph. **Note:** the change in the AdjacencyGraph is the addition of a function that gets all of the edges, which will be needed to solve this problem.

You will need to make further updates to the AdjacencyGraph code in order to make this work. You will need to create an inner class that presents an edge. An edge should have:

- An starting vertex
- An ending vertex
- A weight
- And the ability to be sorted

You will not need to make any other changes to the AdjacencyGraph

A minimum spanning tree is the minimum weight required for every vertex in a graph to be reachable. You will need to write a program that calculates the weight of the minimum spanning tree for a given graph.

To do this you should use the following greedy algorithm:

1. Sort the edges bases on weight such that the smallest weight is first
2. For every edge
3. If adding this edge does not create a cycle
4. add the edge
5. return sum of weight of all added edges

You can determine if adding an edge with create a cycle using a **Union Find** data structure

A union find contains the following information:

- The “boss” of every vertex (a way of determining which group it belongs to)
- The size of every group (which starts at 1)
- The set of all vertices in a group (which starts as a single vertex)

When you create a union find for a particular graph, each vertex in the graph should be initialized as its own group.

To find a given vertex, you should return its boss

Lastly, you need to union the groups. To do this, you:

1. Add all value of the smaller groups set to the larger groups set
2. Update the size of the larger group accordingly
3. Update every vertex in the smaller group to have a boss of the larger groups’s boss

Now, if both vertices in the edge have the same boss, then adding it would create a cycle. Therefore the sudo code becomes:

1. Sort the edges bases on weight such that the smallest weight is first
2. For every edge(u, v)
3. If UnionFind.find(u) != UnionFind.find(v)

4. add the edge
5. UnionFind.union(u, v)
6. return sum of weight of all added edges

Final Note:
I recognize that this problem is much more difficult than anything you will see on your final, so I have attached the sudo code on the next page if you can't get the Union Find and Minimum Spanning Tree code to work and want some help. I strongly recommend not using this right away, but I would look at this before the solutions.

Minimum Spanning Tree

```
Kruskal ( G=(V,E,w) ) using Union-Find data structure
1. Let T = ∅;
2. Sort the edges in increasing order of weight
3. Init(V)   //initialize union-find arrays
4. For edge e = (u,v) do
5.    If Find(u) ≠ Find(v) //no cycle in T ∪ e
6.       Add e to T
7.       Union(u,v) //union of sets containing u, v
8. Return T
```

Union Find

```
Init (V)
1. for every vertex v do
2.    boss[v]=v
3.    size[v]=1
4.    set[v]={v}

Find (u)
1. Return boss[u]

Union (u,v)
1. if size[boss[u]]>=size[boss[v]] then
2.    set[boss[u]]= set[boss[u]]∪set[boss[v]]
3.    size[boss[u]]+=size[boss[v]]
4.    for every z in set[boss[v]] do
5.       boss[z]=boss[u]
6. else do steps 2.-5. with u,v switched
```