Mean: 57.4
Median: 57.5
Problem 2.1
- Did not memoize (would not run in time!)
- Forgot to mod stored values by 1,000,000 (overflow!)

Problem 2.2
- For those using nested for loops, forgot to remove matched letters (double count)

Problem 2.3
- Maintaining the actual list (worst case: $O(n^2)$, e.g., 1,2,…,10k,1,2,…,10k,…)
- Not resetting the starting pointer (e.g., 1,2,1,3,4)
• Homework 6 due Friday at 2pm
• Brett taking over recitations in the interim
• Practice contests during recitations
  • Individual
  • Top 5 placers get bonus points! (Up to 5 pts per person)
  • Reminder: recitations are mandatory
• Bonus problem from HW4
• **Edmonds-Karp algorithm**
  • Use BFS to find an augmenting path
  • Runtime: $O(VE^2)$
    • Provable that after $O(VE)$ iterations, all augmenting paths are exhausted
  • Does not run into the same problem as the Ford-Fulkerson degenerate case
    • Recall that capacity $\neq$ weight in the flow graph
    • Each edge is unweighted
Ford-Fulkerson($G, s, t$):
for each edge $(u, v)$ in $E[G]$
    do $f[u, v] \leftarrow 0,$
        $f[v, u] \leftarrow 0$
while there exists a path $p$ from $s$ to $t$ in the residual network $G_f$
do $c_f(p) \leftarrow \min \{ c_f(u, v) : (u, v) \text{ is in } p \}$
    for each edge $(u, v)$ in $p$
do $f[u, v] \leftarrow f[u, v] + c_f(p)$
        $f[v, u] \leftarrow -f[u, v]$
void addEdge(int u, int v, int cap, int[][] caps, ArrayList<Integer>[][] adj) {
    if (caps[u][v] == 0 && caps[v][u] == 0) {
        adj[u].add(v); adj[v].add(u);
    } 
    caps[u][v] += cap;
}

int maxflow(ArrayList<Integer>[][] adj, int[][] caps, int source, int sink) {
    int ret = 0;
    while (true) {
        int f = augment(adj,caps,source,sink);
        if (f == 0) break;
        ret += f;
    }
    return ret;
}
int augment(ArrayList<Integer>[] adj, int[][] caps, int source, int sink) {
    Queue<Integer> q = new ArrayDeque<Integer>();
    int[] pred = new int[adj.length];
    Arrays.fill(pred,-1);
    int[] f = new int[adj.length];
    pred[source] = source; f[source] = Integer.MAX_VALUE; q.add(source);
    while (!q.isEmpty()) {
        int curr = q.poll(), currf = f[curr];
        if (curr == sink) {
            update(caps,pred,curr,f[curr]);
            return f[curr];
        }
        for (int i = 0; i < adj[curr].size(); ++i) {
            int j = adj[curr].get(i);
            if (pred[j] != -1 || caps[curr][j] == 0) continue;
            pred[j] = curr; f[j] = Math.min(curre, caps[curr][j]); q.add(j);
        }
    }
    return 0;
}
void update(int[][] caps, int[] pred, int curr, int f) {
    int p = pred[curr];
    if (p == curr) return;
    caps[p][curr] -= f;
    caps[curr][p] += f;
    update(caps, pred, p, f);
}
If each edge is also associated with a cost, you may want to find the min-cost max flow.

To do so, replace BFS with Bellman-Ford in Edmonds-Karp.

- Runtime: $O(V^2E^2)$
A consequence of computing the maximum flow is computing the **minimum cut**

Min cut: The smallest cost (or cut set) for removing edges so that the graph is split into two disconnected components.
Max-Flow Min-Cut theorem: The maximum flow from \( s \) to \( t \) is the size of the minimum \( s-t \) cut. An \( s-t \) cut is a partition of the nodes into 2 sets \( S, T \) with \( s \in S \) and \( t \in T \). The value of the cut is the sum of all capacities on edges from \( S \) to \( T \).

- Nodes in \( S \) are nodes that are reachable from \( S \) in the residual graph.
Exercise: Find the min-cut of the following graphs:

a) $s \rightarrow 1 \rightarrow 1 \rightarrow 1 \rightarrow t$

b) $s \rightarrow 10 \rightarrow 7 \rightarrow 3 \rightarrow 15 \rightarrow t$

c) $s \rightarrow \text{complex graph} \rightarrow t$
(a) The min cut has weight 1.

(b) The min-cut has weight 13.

(c) The min-cut has weight 3. All capacities 1:
Exercise: Suppose you have a problem where vertices have capacities in addition to edges. Explain how to compute the maximum flow.

Exercise: Suppose you are given a graph with edge capacities, and a list of sources and sinks. Each source $s_i$ has an amount of flow $f(s_i)$ that must come out of it. Each sink $k_j$ has an amount of flow $f(k_j)$ that must flow into it. Determine whether these constraints can be satisfied by pushing flow through the network. That is, your answer is boolean.
Exercise:

- Up to 50 cows scattered on a 100x100 field
- Field contains square plots that are of different elevation
- Fields flood every hour to a new level 0-100 over 24 hours
- Cows can move every hour to an adjacent plots whose elevations are higher than level
- Only one cow can be on one plot at the same time
- How many cows can survive?
Exercise: Your statistical model has determined for each pixel the likelihood it should be labeled part of the foreground or background of the image. The pixel $p_i$ has a cost $a_i$ if it is part of the foreground, or a cost $b_i$ if it is part of the background (costs all positive). You have also determined for each pair $(i, j)$ a cost $p_{ij}$ they are placed differently. Give an algorithm to determine an assignment of pixels to the background and foreground that minimizes the cost.
Max-Flow: Competitive Programming 4.6