Discussion 11

APT REVIEWS AND GRAPH ALGORITHMS
Before we begin

Any questions on Autocomplete? We’ll have time in discussion where if you were able to complete APT Set 6 without issue, you can work on Autocomplete, so we’ll be able to tackle coding questions later.
First, we’ll review the solution for AuntUncle. Since it’s a non-collaborative APT, we’ll only go over a high-level solution.
AuntUncle review

The tricky part of this APT is that while it’s very easy to figure out who is who’s parent or child, it’s not quite easy to figure who is who’s siblings.

It turns out, like with MemberCheck, we can use sets to easily figure out aunts and uncles.

Rather than define an aunt or uncle as a parent’s sibling, we’ll say that an aunt or uncle is anyone who is a grandparent’s child, but not a parent. Thus, we avoid marking a parent as an aunt or uncle.
Finding the set of aunts and uncles

Let $T$ be our target.

Let $P$ be the set of $T$’s parents

Let $GP$ be the set of parents of members of $P$

Let $GPC$ be the set of children of members of $GPC$

Let $AU = GPC - P - T$

Then $AU$ is our set of aunts and uncles.

This approach is very easy to code, because it only defines sets in terms of other sets, and parents and children.
Implementation details

Then, all that’s left to do is figure out how to get from each set to the next.

To do this, we would want to have a data structure which keeps track of the parent-child relationships. We suggest two HashMaps, from Strings to Sets of Strings.

One map is a map from each member to a set of all their parents. We’ll call this parents.

Another map is a map from each member to a set of all their children. We’ll call this children.
Updating maps

Then, anytime we learn that A is the child of B and C:

We add A as a key to parents, and its value is the set \{B, C\}.

If B and C aren’t keys in children, add them as keys with empty sets as their values.

Then, add A to the sets children.get(B) and children.get(C).
Finding these sets

Then, once we’ve constructed children and parents, to perform the step “Let $GP$ be the set of parents of members of $P$”

We just initialize a new set, and for each member $p$ of $P$, we add all elements of $\text{parents.get}(p)$ to the set.

We can do the other set creation steps very similarly.

Then, to take the difference of two sets, just use retainAll. Finishing the APT simply requires taking the final set and turning it into a sorted array.
Recall that:

- A graph is a set of vertices and a set of edges where each edge connects two vertices.
- We usually store graphs as adjacency lists, a mapping of each vertex to a list of all vertices it is connected to. For example, for the graph below, the vertex 0 would map to the list {1, 6}.
- Search algorithms are algorithms which visit vertices in a graph one by one in some desired order. They keep track of which vertices they’ve visited, and keep a collections of vertices to visit in the future.

![Graph Diagram]
BFS Example from class

1. _ ⇒ a
2. a ⇒ b c
3. b c ⇒ c d e
4. c d e ⇒ d e f g
5. d e f g ⇒ e f g h
6. e f g h ⇒ f g h i j
7. f g h i j ⇒ g h i j k
8. g h i j k ⇒ h i j k
9. h i j k ⇒ i j k
10. i j k ⇒ j k
11. j k ⇒ k
12. k ⇒ _

private void bfs(Graph G, int s) {
    Queue<Integer> q = new Queue<Integer>();
    q.enqueue(s);
    marked[s] = true;
    distTo[s] = 0;
    while (!q.isEmpty()) {
        int v = q.dequeue();
        for (int w : G.adj(v)) {
            if (!marked[w]) {
                q.enqueue(w);
                marked[w] = true;
                edgeTo[w] = v;
                distTo[w] = distTo[v] + 1;
            }
        }
    }
}

Green = just added | Gray = just removed
Underline = front of the queue
DFS/BFS Example

What is a possible order of visiting vertices in the below graph if we use DFS/BFS starting from vertex 0? What vertex will always come last in BFS?

What do you think is the runtime of DFS and BFS, for a graph with n vertices and m edges?

What do you notice about the edges we take in a DFS or BFS?

Create a collection of vertices Q
Q.push(start vertex)
While Q is not empty:
  curr = Q.pop
  if we’ve visited curr, continue
  mark curr as visited
  push all of curr’s unvisited neighbors into Q
BoggleScore review

We’ll go over the solution for BoggleScore now, as well as some good practices for solving the APT.
BoggleScore review

As you saw in lecture, the best way to count how many ways there are to form one word in BoggleScore is to keep track of how many ways there are to form each substring of that word, ending in each position.

For each character, we update each value to the sum of its neighbors, anywhere that character exists.
General algorithm review

Total = 0

For each word:

    Initialize an array counts, which keeps track of how many paths there are which form word ending in each position

    Initialize counts[i][j] to 1 everywhere the first letter of word exists, 0 otherwise.

    For each letter after the first in word:

        For every place the letter is in, the new value of that position in counts is the sum of its neighbors in counts

        Add (word’s length)²(sum of values in count) to our total

    Return total
Useful helper methods

The following helper methods make writing BoggleScore easier. We’ll show you how to write these, and then leave writing the main method (using the pseudocode from the previous slide) to you:

1. A method to generate our initial array of counts
2. A method to take the sum of an element’s neighbors in counts
3. A method to update our array of counts given the next character
4. A method to sum the array

We’ll also declare a global variable:

```java
public static final long mod = (long) 1E13; - so we can easily use the modulus function
```
A method to generate our array of counts

```java
public long[][] initialCounts(String[] grid, char first)
{
    long[][] init = new long[4][4];
    for(int i = 0; i < 4; i++)
    {
        for(int j = 0; j < 4; j++)
        {
            if (grid[i].charAt(j) == first){
                init[i][j] = 1;
            }
        }
    }
    return init;
}
```
A method to take the sum of an element’s neighbors

```java
public long neighborSum(long[][] counts, int i, int j) {
    long output = 0;
    int[] di = { -1, -1, -1, 0, 1, 1, 1, 0};
    int[] dj = { -1, 0, 1, 1, 1, 0, -1, -1};
    for (int dir = 0; dir < di.length; dir++) {
        int newi = i + di[dir]; int newj = j + dj[dir];
        if (newi >= 0 && newi < 4 && newj >= 0 && newj < 4) {
            output = (output + counts[newi][newj]) % mod;
        }
    }
    return output;
}
```
Aside – taking the mod correctly

As many of you probably noticed, when numbers get too large in Java, they sometimes become negative. This is known as overflow, and you’ll learn more about it in CS 250.

What this means for this APT is that we should call the mod function on any sort of score sum or total every time we update it.

More specifically, if we want to increase our total score (or anything which will eventually be added to our total score) by some number $x$, we should use the following syntax:

```java
total = (total + x) % mod;
```

The following will not work as intended

```java
total = total + x % mod;
```
```java
total += x % mod;
```
A method to update our array of counts

```java
public long[][] updateCounts(String[] grid, long[][] counts, char next){
    long[][] newCounts = new long[4][4];
    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            if (grid[i].charAt(j) == next){
                newCounts[i][j] = neighborSum(counts, i, j);
            }
        }
    }
    return newCounts;
}
```
public long sum(long[][] counts){
    long output = 0;
    for(int i = 0; i < 4; i++){
        for(int j = 0; j < 4; j++){
            output = (output + counts[i][j]) % mod;
        }
    }
    return output;
}
BoggleScore memoization

To memoize BoggleScore – note that when computing results for a word, we also compute results for all prefixes of that word.

e.g. in computing the results for “TREE” we compute the results for “T”, “TR”, and “TRE” as well.

Then, we can create a map of Strings to Longs at the start of our method, and for each prefix, store the score that prefix would earn in the map.
Worktime

For the rest of discussion, if you were unable to finish BoggleScore, work on it so that you can submit it for APT Amnesty. If you finished BoggleScore, work on AuntUncle. If you finished AuntUncle, work on Autocomplete. As always, your TAs are here to help!