CS340 - Registrar Project

Greg Van Aken, Conor Stuart Roe, Russell Gerhard

October 5, 2018

1 Checkpoint 1

1.1 Description

Let M denote an nxn 2-D matrix, where n is the number of classes. Initialize M such that all values are 0. Iterate over the teacher-class list. For each pair of classes (c, c') taught by the same teacher, set $M[c][c']$ to *Infinity*. Iterate over each student's preference list. For each class pair (c_i, c_j) in a given student's preference list, (where i and j represent indices/preferences) increment $M[c_i][c_j]$ by ((4-i) + (4-j)). In other words, compute the effective *cost* to each student of overlapping two courses. If i is 0 and j is 1, the student prefers both of those classes highly and their overlap will be a cost of 7, much higher than if i were 2 and j were 3 (cost of 3). Classes taught by the same teacher mustn't, by definition, occur at the same time, so their overlap cost is infinite. Assemble each class into its own disjoint set in a collection of sets S. While the size of $S >$ the number of class times, merge sets with the minimum overlap cost, according to M. When a merge takes place between two sets s and s' to form set s'', for each class pair c and c' in s'', for each class c'' in $M[c]$, increment $M[c][c'']$ by ($M[c][c'] + M[c''][c']$) and for each class c''' in $M[c']$, increment $M[c'][c''']$ by ($M[c][c'] + M[c'''][c]$). Distribute classes such that merged sets occur during the same time slot.

1.2 Pseudocode

Function `genMatrix`(*Preferences, numClasses, Teachers*)
  let n = len(numClasses); initialize M = n x n 2-D array of all 0's;
  M = setTeach(M, $Teachers$);
  M = setCost(M, $Preferences$);

1.3 Time Analysis

Time Analysis goes here. $O(n^2)$

```
Function setTeach(M, Teachers)
    sort Teachers by teacher id; // Teachers is a list of class-teacher pairs
    initialize last = Teachers[0];
    while i = 1 < len(Teachers) do
        let teach = Teachers[i];
        if teach[1] == last[1] then
            set M[teach[0]][last[0]] to infinity; // teacher teaches both classes
        end
        set last = Teachers[i]; i++;
    end
    return M
```

```
Function setCost(M, Preferences)
    for student in Preferences do
        // Assume Preferences = [[p1,p2,p3,p4],[p1,p2,p3,p4],...]
        for i in range(0:4) do
            for j in range(i+1:4) do
                let c1 = student[i]; let c2 = student[j];
                M[c1][c2] += ((4-i) + (4-j))
            end
        end
    end
```

1.4 Discussion

We had a relatively good intuition when approaching this problem that we had to somehow compute a cost associated with having classes at the same time. One option we considered was to weight the cost according to how many students preferred a given class. In other words, for every student that lists some class as their first choice, add a cost of 4 to that class. For every student who lists it as their second choice, add a cost of 3, etc. We realized that although this would determine a set of highly preferred classes that may have a number of students wanting to take them (and thus should not overlap), it loses the logic about what each student wants. A counter example to our initial approach is that there are two popular courses but they occur in different majors. There may be a lot of students who prefer those classes, but few students will desire to take both courses. Thus, it would be reasonable for those classes to overlap. This led us to our current algorithm which makes decisions based on the cost to each student of having two classes overlap. The aspects of this problem which made it difficult to determine an algorithm are:

1. How do we quantify the cost of two classes overlapping?
    - We decided to set it up such that merging courses is a direct function of their collective priorities
    - An artifact of this is that merging classes 1 and 4 has the same cost as merging classes 2 and 3
2. After a minimum cost is found and then a merge takes place, the cost of merging any third class with the new set will have to include the total merge cost of the third class with every class already in the set
    - This fact led us to the idea of merging matrix rows/columns to reflect the additive costs when sets are merged
3. A given teacher cannot teach more than one class at a given time
    - This issue led us to perform the initialization step of setting matrix cells corresponding to classes taught by the same teacher to Infinity
4. A given time slot cannot have more classes scheduled than the given number of rooms
    - Because of this, we decided to check to see if this condition is broken and if so, skip that merge

This algorithm is **greedy** because each time work is done, it generally follows a simple rule: merge sets with the lowest merge cost (if a merge is valid). This algorithm is not too different from constructing a minimum spanning tree using Kruskal's algorithm. Classes can be thought of as nodes in a graph and merge cost can be thought of as the weight of

the edge between nodes. Similar to a minimum spanning tree, minimum edge weights are selected as the sets are constructed (merged).

Collaboration

Name1, Name2, ...