# MATH 307: Discrete Structures II Projects

**Mukesh Kumar**

Department of Mathematics, College of Charleston, Charleston, SC 29424

## 1 Overview

The discrete mathematics ideas and techniques you are studying this semester have applications in a variety of fields. Learning about the ways in which mathematical ideas are applied to problems from other disciplines is one of the goal of this course.
Choose one topic from the list of given projects below. If you do not find the listed projects interesting, then you can propose a project of your choice. In that case you have to schedule a meeting with me to discuss your project.

- **Foreign Exchange Arbitrage**
  Arbitrage exploits a price difference between two or more markets to make money with zero risk. Sporting bets are perhaps the easiest example. Given a sporting event with two possible outcomes you look for an opportunity where two book makers give slightly different odds and try to ake benefit of it. For example, consider a case with two outcomes, say Federer vs. Nadal. One bookie offers odds of 5/2 on Federer, whereas another offers odds of 3/5 on Nadal. If I bet $32 on 5/2, and $68 at 3/5 then regardless of the outcome I'm guaranteed to make a little money (a minimum of 9% in this case. $32 at 5/2 odds gives me $112 and $68 at 3/5 gives me $109). Foreign exchange (Forex) markets are another opportunity for the arbitrageur. By finding mispriced currencies you can get money for nothing. For example I could transfer my money from USD to EURO and using another conversation of currencies back to USD and end up with a profit if someone got these exchange rates wrong! Now all I need to do is find a way of finding these opportunities and I'll be rich! The exchange rates can be represented as a weighted graph with each node representing a currency and each directed edge representing the exchange rate. To find an arbitrage opportunity we need to find a path through the graph (starting and ending at the same node) where the product of edge weights is greater than 1. The shortest path is the best one, because we want to make as few trades as possible. In this project students will explore the mathematical modeling of this problem using directed weighted graphs, the existence of negative weighted cycle, and the available shortest path algorithms to solve this problem.

- **Traveling Salesman Problem (TSP)**
  This is one of the oldest discrete optimization problems. It has many incarnations but the point is always to find the shortest path connecting some geographic locations while visiting each only once. The TSP problem is defined as follows: Given a set of cities and distances between every pair of cities, find the shortest way of visiting all the cities exactly once and returning to the starting city. You can imagine the cities as nodes in a completely connected graph and distances as edge cost between the cities. Now the problem is transformed to finding the shortest tour of the graph. This clearly has applications in problems such as optimizing airline routes or laying cables but it also appears in some unexpected places such as DNA and genome research. This is a well known NP-Complete problem and there are many different heuristics available to obtain approximate solutions. In this project, students are going to use a search strategy to find the optimal solution to the problem and use the Minimum Spanning Tree as the heuristic function.

- **Airport Gate Assignment Problem**
  Due to the rapid growth of air transport traffic, how to efficiently allocate gates at airports to coming or outgoing flights has become one of the most important problems which managers of airlines and airports have to concern about. The problem, which can be modeled as a scheduling problem, is often referred to as flight gate scheduling or gate assignment problem. Gate scheduling is concerned with finding an assignment of flights or aircraft to terminal and ramp positions (gates), and an assignment of the start and completion times of the processing of a flight at its position. In this project, students will explore how the basic idea of graph coloring can be used for this scheduling problem. Also, they will learn how dynamic programing can be useful in solving this problem.

- **Minimum Spanning Tree Animation**
  The minimum spanning tree (MST) of a weighted graph is a spanning tree whose sum of edge weights is

minimal. The minimum spanning tree describes the cheapest network to connect all of a given set of vertices. Kruskal's algorithm for minimum spanning tree works by inserting edges in order of increasing cost, adding as edges to the tree those which connect two previously disjoint components. For an example, see Kruskal's algorithm at work `http://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/anim/mst.gif` on a graph of distances between 128 North American cities. Almost imperceptively at first, short edges get added all around the continent, slowly building larger components until the tree is completed. In this project, students will write the computer code and explain the basic idea for their minimum spanning tree animation.

- **Breadth First Search/Depth First Search Animations**
  Breadth-first search (BFS) and depth-first search (DFS) are two distinct orders in which to visit the vertices and edges of a graph. BFS radiates out from a root to visit vertices in order of their distance from the root. Thus closer nodes get visited first. DFS prefers to visit undiscovered vertices immediately, so the search trees tend to be deeper rather than balanced as with BFS. For an example, see the animations of BFS `http://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/anim/bfs.gif` and DFS `http://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/anim/dfs.gif` where the DFS consists of three "Hamiltonian" paths, one in each component–while the BFS tree has far more degree-3 nodes, reflecting balance. In this project, students will write the computer code and explain the basic idea for their BFS and DFS animations.

- **Eulerian Cycle Animation**
  An Eulerian cycle in a graph is a traversal of all the edges of the graph that visits each edge exactly once before returning home. The problem was made famous by the bridges of Konigsberg, where a tour that walked on each bridge exactly once was unsuccessfully sought. A graph has an Eulerian cycle if and only if all its vertices are that of even degrees. To actually find such a tour, we can extract cycles from the graph and then graft them together, noting that the deletion of cycle must leave even degree vertices as even. It is amusing to watch as the Eulerian cycle finds a way back home to a seemingly blocked off start vertex. We are allowed (indeed required) to visit vertices multiple times in an Eulerian cycle, but not edges. An animation for Eulerian cycle can be seen at `http://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/anim/euler.gif` In this project, students will write the computer code for the animation of Eulerian cycle.

- **Transitive Closure by Graph Powering Animation**
  The transitive closure T(G) of a given graph G connects vertices u and v iff there is a path in G from u to v. Thus the transitive closure of any connected graph is complete. This animation finds the transitive closure of a graph by taking its adjacency matrix and raising it to the nth power, where n is the number of vertices in G. When we raise the graph to the kth power, we add exactly the edges which represent paths of length k in the original graph. For an example, an animation can be found `http://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/anim/graphpower.gif`, where red denotes most recently added edges, blue the edges added the previous iteration, and green the edges two iterations before, before the edges cool to black. The original graph edges are shown in orange. A computationally cheaper way to find transitive closure is to use Warshall's algorithm, but graph powering also allows us to count how many paths there are of different lengths. In this project, students will write the computer code for the animation of transitive closure of a given graph.

- **Dijkstra's Algorithm Animation**
  Dijkstra's Algorithm solves the single-source shortest path problem in weighted graphs. Dijkstra's algorithm starts from a source node, and in each iteration adds another vertex to the shortest-path spanning tree. This vertex is the point closest to the root which is still outside the tree. Watch as the tree grows by radiating out from the root. Note that it is not a breadth-first search; we do not care about the number of edges on the tree path, only the sum of their weights. For an example, see the animation on `http://www3.cs.stonybrook.edu/~skiena/combinatorica/animations/anim/dijkstra.gif` Here it is running on a planar graph whose edge weights are proportional to the distance between the vertices in the drawing – thus the weight of an

edge is equal to its visible length. In this project, students will write the computer code for the animation of Dijkstra's algorithm.

## 2 Guidelines

The general guidelines for the project are as follows:

- **Deadline**: The project will be summarized in the form of a report of 7-10 pages long. The report is due for submission on last date of lecture class on Tuesday, April 23th, 2019.

- **Individual or Group project**: You are free to do the project alone or in form of a group (of 2 students). If you are doing the project in a group, then please let me know the name of each members in your project.

- **Grading criteria**: Breakdown of points (50 points for the project assignment and 10 points as extra credit) are given below.
  - Content (30 pts): The mathematical concepts used in the project should be correct. The project should contain clear statement and questions that you are answering.
  - Clarity, Quality of presentation, Organization (20 pts): The project should be written so that the reader can follow and understand the mathematics being described. The order of sections should make sense and correct spelling of terms. You should use pictures and diagrams whenever possible.
  - Creativity and Originality (10 pts): You will get these points for giving new spins to the project, e.g., if you create a youtube video or presentation of your work, or your code can do more than just the minimum requirement. Another example is if you use LaTEX to write your report.

You are allowed to read any book or other resources available in the CofC library or online to understand the material. But sources must be cited in the report and you should select your own examples to write the final report!