CSE481 – Optimization Methods
Programming Assignment 1

Majority of this assignment's evaluation (except reports) will be automated. Please strictly follow these naming conventions (case-sensitive):

- Upload one file: “RollNo_ass1.tar.gz” (“RollNo_” is automatically added by portal) with the following directory structure:

```
RollNo_ass1
├── problem1
│   ├── report.pdf
│   ├── run.sh
│   └── <your source code>
├── problem2
│   ├── report.pdf
│   └── <your source code>
├── problem3
│   ├── report.pdf
│   ├── run.sh
│   └── <your source code>
└── problem4
    ├── run.sh
    └── <your source code>
```

- For problems 1,3,4 run.sh is run with two arguments – input file and output file. For scoring output file will be either binary compared with the correct output (if solution is unique) or evaluated by a program. Note that run.sh should also take care of compiling your code.
- For problems 1,2,3 report.pdf should contain any plots and answers to specific questions.

---

1. Minimum Spanning Tree (LP vs Kruskal) (15)

Problem Statement

In this problem, we will compare the performance of MST computation of LP against Kruskal's algorithm. You have to write two programs (In the same language – preferably C+glpk). run.sh should compile both programs, run them with input file and concatenate the outputs of both the programs to the output file (In the order LP, Kruskal). Though the resulting tree may be different, the tree weight should be the same.

Note that while formulating a problem into LP, you must make sure that the size of LP (no of constraints and variables) is polynomially bounded by the input problem size (No. of edges in input graph), to ensure that LP runs in polynomial time. If you use the naïve LP formulation of MST (constraining all sub-sets) – it becomes exponential. Instead for a fair comparison with Kruskal, you have to use the MAD (Maximum Average Degree) based formulation (discussed in the first few tutorials, resource on courses portal).

Input Format

- The first line of the input contains two space-separated integers N (No of vertices) and M (No of edges).
- The next M lines each contain three space-separated integers i,j,w – denoting an undirected edge (i,j) with weight w.

- The graph is simple, connected and undirected (so only one of (i,j) or (j,i) will appear in the input), vertices are zero-indexed and weights are positive.

Output Format
After running run.sh, the output file should contain 4 lines:
- The first line should have (N-1) space-separated integers – the indices (zero-indexed) of edges in the input-order that form an MST of the graph (computed from LP).
- The second line should contain a single integer – the cost of the MST (computed from LP).
- The third line should have (N-1) space-separated integers – the indices (zero-indexed) of edges in the input-order that form an MST of the graph (computed from Kruskal).
- The fourth line should contain a single integer – the cost of the MST (computed from Kruskal).

Note that both costs must be same and the minimum possible. None of the lines should contain any trailing spaces.

Sample Input
4 5
0 1 1
0 2 2
0 3 3
1 2 2
2 3 3

Sample Output
0 1 2
6
4 0 3
6

Report
- Generate random connected graphs of increasing no. of vertices (2x for every step) as inputs to your programs.
- The report should contain two plots. Each of the two plots should contain the running times of both LP and Kruskal as a function of the no. of vertices in the graph (N).
- Keep the generated graphs sparse (M=2N) for the first plot, and dense (M=$^{N}C_{2}$) for the second.
- Include **only** the plots and not your inputs and outputs.

---

2. Dictionary Learning and Sparse Coding using K-SVD Algorithm (10)

- The description of this problem and input are in **ass1_q2.pdf** and **ass1_q2_input1.txt**.
- run.sh is not required, but source code to compute the vectors and plot the results must be submitted.
- report.pdf should contain all the plots.

---

3. Matrix Condition Number (10)

Refer on-line material on condition number of a matrix and answer the following questions. Please do not write unnecessarily long answers. Your answer should be precise and to the point.

a. Explain the notion of condition number of a matrix.

b. How does it affect the accuracy of a numerical computation?

c. Please refer this link: http://engrwww.usask.ca/classes/EE/840/notes/ILL_Conditioned%20Systems.pdf and solve the following problem.

Problem Statement

Consider the problem of solving for "x" in "Ax = b". Demonstrate the role of condition number in calculation of "x". You need to show the results for various values of "A" (with varying condition number). For each "A" you need to show how the unknown "x" changes when "A" is changed from "A" to "(A + ΔA)".

To do this write a **python** program. You can use **numpy** for maintaining arrays, matrices, solving Ax=b, inverse, L2Norm etc. Calculate the norm (||A||) and condition number K(A) of a matrix **as described in the link**.

Plot a graph with condition number as the x-axis and change in "x" as the y-axis. Use **matplotlib**.

change in "x" = L2Norm(x – x1) / L2Norm(x)
x is the solution of Ax = b
x1 is the solution of (A + ΔA)x = b

Input format

- Line 1: T (number of test cases)
- Line 2: N (dimension of square matrix)
- Line 3: matrix "A" (N lines follow, each containing N space separated integers)
- Line N+3: "b" (a single line with N space separated integers)
- Line N+4: matrix "ΔA" (same format as "A")

Output format

- For each test case output the condition number and the change (space separated).
- Output for each test case should be on a single line (that is each output on a new line).

Sample Input

```
1                 // T
2                 // N
400 -201          // A
-797 401
200 -200          // b
1 0               // ΔA
0 0
```

Sample output

7064.06896552 0.664582588072

Report

- Answer questions **a.** and **b.** in you report.
- For the programming question **c.**: run your program for the two input files (**ass1_q3_input1.txt** and **ass1_q3_input3.txt**) and include the resulting plots.

- Include **only** the plots and not your inputs and outputs.
- Include run.sh and all your source code as usual.

---

4. SuDoKu Solver (10)

Problem Statement

Make a Sudoku solver by formulating it as an IP problem. Use C+glpk to solve. The input puzzles will have a unique answer. Include run.sh and all your source code as usual. Do not add any outputs.

Input Format

- There would be multiple test cases (scan until EOF).
- Each Input is of 10 lines.
- First line is "Grid #i", #i here represents input number.
- Next 9 lines represent the Sudoku structure, unknown squares are represented as 0, no space between adjacent numbers.

Output Format

- For each test case, print 9 lines each with 9 characters representing the number on that square.
- Print empty line between answers of different test cases.

Sample Input

```
Grid 01
003020600
900305001
001806400
008102900
700000008
006708200
002609500
800203009
005010300
Grid 02
200080300
060070084
030500209
000105408
000000000
402706000
301007040
720040060
004010003
```

Sample Output

```
483921657
967345821
251876493
548132976
729564138
```

136798245
372689514
814253769
695417382

245981376
169273584
837564219
976125438
513498627
482736951
391657842
728349165
654812793
