Homework #4

Due Time: 2020/1/7 14:20
Contact TAs: ada-ta@csie.ntu.edu.tw

Instructions and Announcements

- There are **one programming problem** and **two hand-written problems**.
- **Programming.** (30 pt) The judge system is located at https://ada-judge.csie.ntu.edu.tw. Please login and submit your code for the programming problems (i.e., those containing "Programming" in the problem title) by the deadline. NO LATE SUBMISSION IS ALLOWED.
- **Hand-written.** (70 pt) For other problems (also known as the "hand-written problems"), you should upload your answer to **Gradescope** as demonstrated in class. Please **briefly** explain your solution in this part. NO LATE SUBMISSION IS ALLOWED.
- **Collaboration policy.** Discussions with others are strongly encouraged. However, you should write down your solutions **in your own words**. In addition, for **each and every** problem you have to specify the references (e.g., the Internet URL you consulted with or the people you discussed with) on the first page or comment in code of your solution to that problem. You may get zero point due to the lack of references.

Problem 1 - Jigsaw Sudoku (Programming) (30 points)

Problem Description

Today, boook and ltf found an interesting game. It is called `killer sudoku`. The game goes as follows:

- There's a 9×9 board containing 81 grids. Like in the original sudoku, you need to fill an integer in each grid. The integers you fill must be in the range of $[1, 9]$.
- Every grid has a color assigned to it, and there are at most 81 colors. The grids that have the same color are connected.
- The integers filled in the grids need to satisfy the following requirements:
  - In each row, each number must appear exactly once.
  - In each column, each number must appear exactly once.
  - The board is divided into nine 3×3 blocks, each block contains 9 grids. In each block, each number must appear exactly once.
  - For each color, the number written on those colored grids must appear at most once.
  - For each color, the summation of the numbers written on those colored grids must equal to a target value.

Now, given the color on each grid and the target value of each color, please help boook and ltf find a way to complete the killer sudoku.

The colors in the following figures are just for separating different components, they do not represent the colors mentioned in the problem description.

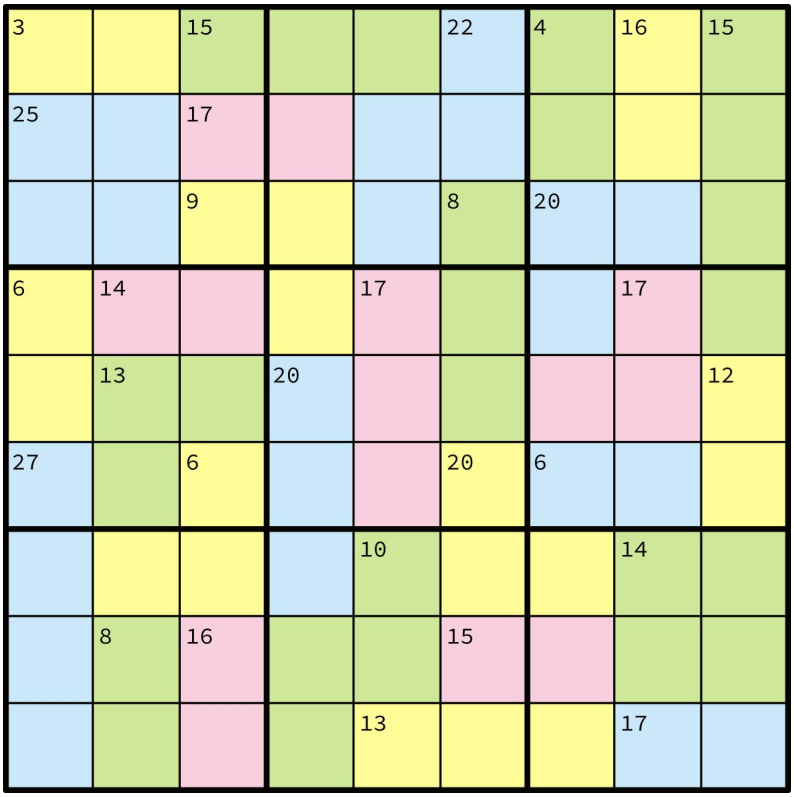


Figure 1. One killer sudoku (source: wiki)

Figure 2. The solution of a killer sudoku (source: wiki)

Input

In the first 9 lines, each line contains a string with 9 characters, representing a 9×9 board with colors assigned. Each character represents a specific color. It is guaranteed that the grids having the same color are connected, and the ASCII values of all characters are ranged in $[33, 126]$.

Let c be the number of different colors assigned on the board. In the following c lines, each line contains a character a and a positive integer k, meaning that the color represented by a has a target value k. It is guaranteed that all of these c characters appear on the board and are distinct.

Test Group 0 (0 %)

- Sample Input

Test Group 1 (10 %)

- $c = 9$

Test Group 2 (20 %)

- $c \geq 75$

Test Group 3 (40 %)

- $c \geq 37$

Test Group 4 (30 %)

- No additional constraints.

Output

Print a 9×9 board filled with integers satisfying all constraints mentioned above. If there are multiple solutions, you may print any one of them. It is guaranteed that there exists a solution satisfying all constraints.

Sample Input 1

```
AABBBCDEF
GGHHCCDEF
GGIICJKKF
LMMINJKOF
LPPQNJOOR
SPTQNUVVR
STTQWUUXX
SYZWW@@XX
SYZW###$$
A 3
B 15
C 22
D 4
E 16
F 15
G 25
H 17
I 9
J 8
K 20
L 6
M 14
N 17
O 17
P 13
Q 20
R 12
S 27
T 6
U 20
V 6
W 10
X 14
Y 8
Z 16
@ 15
# 13
$ 17
```

Sample Output 1

```
215647398
368952174
794381652
586274931
142593867
973816425
821739546
659428713
437165289
```

Hints

1. In the following we introduce the *exact cover problem*, which may help you to solve the problem. Formally, given a set $X = \{x_1, x_2, \ldots, x_n\}$ with n elements, and several subsets Y_1, Y_2, ..., Y_m of X. You are asked to determined whether it is possible to choose some indexes $i_1, i_2, \ldots, i_k$ such that

$$X = \bigsqcup_{j=1}^{k} Y_{i_j}.$$

That is, X is the disjoint union of subsets $Y_{i_1}, Y_{i_2}, \ldots, Y_{i_k}$.

You are given a header file `helper.h` which is helpful to solve the *exact cover problem*.

- `void DLX::Init(int n)`: Initialization, the set X is set to be size n.
- `void DLX::AddRow(int rr, vector<int> &sol)`: Insert a subset `sol`, regarded as Y_{rr}, of X.
- `vector<int> DLX::Solver()`: Return the vector of chosen indexes if the answer exists, or an empty vector otherwise.

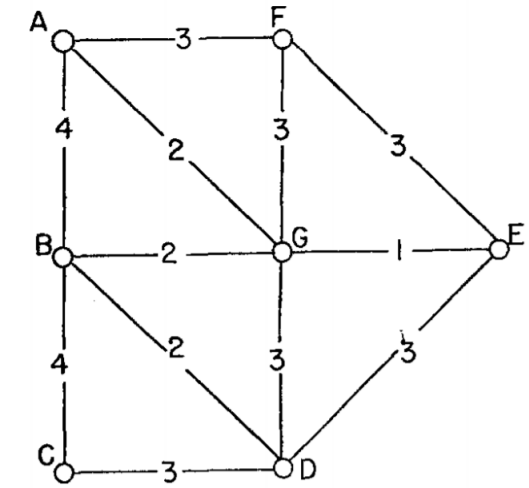
Problem 2 - NP-Completeness & Reduction (Hand-Written) (35 points)

Given a real symmetric matrix $A = (A_{ij})$ of size n, we are interested in finding a permutation matrix P such that $PAP^T = (b_{ij})$ has non-zero entries arranged "near" the diagonal. That is, we want to find the minimal $k \in \mathbb{N}$ such that $b_{ij} = 0$ whenever $|i - j| > k$. A deterministic statement is, given matrix A and natural number $k \in \mathbb{N}$, does there exist a permutation matrix P such that $PAP^T = (b_{ij})$ satisfies $b_{ij} = 0$ for all $|i - j| > k$?

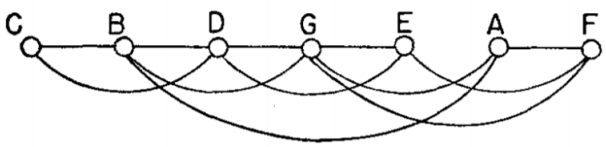
One can consider an undirected graph $G = (V, E)$ that is "induced" by A, where $V = \{v_1, ..., v_n\}$ and $(v_i, v_j) \in E \iff A_{ij} \neq 0$ (therefore, one can regard A as the adjacency matrix of G). We get a graph version of the above question: given a graph G and $k \in \mathbb{N}$, does there exist a 1-1 labeling function τ from V to $\{1, ..., n\}$, such that $\forall (u, v) \in E$, $|\tau(u) - \tau(v)| \leq k$? Formally, given a graph G and $k \in \mathbb{N}$, say a instance of this problem $BMP[G, k]$ is solvable if there exist a labeling function τ from V to $\{1, ..., n\}$, such that $\forall (u, v) \in E$, $|\tau(u) - \tau(v)| \leq k$. This is known as the **Bandwidth Minimization Problem**.

Furthermore, consider a more generalized version:

Given a weighted graph G, say a instance of this problem $LAP[G]$ is solvable if there exist a labeling function τ from V to $\{1, ..., n\}$, such that $\forall (u, v) \in E$, $|\tau(u) - \tau(v)| \leq w(u, v)$. Equivalently, can we arrange the vertices of G in a linear array according to the labeling, such that no two adjacent vertices (in G) have a distance (in the linear array) greater than the weight of the edge joining them? As an instance, the below graph in (a) is solvable by the arrangement (labeling) shown in (b). This is known as the **Linear Array Problem**.



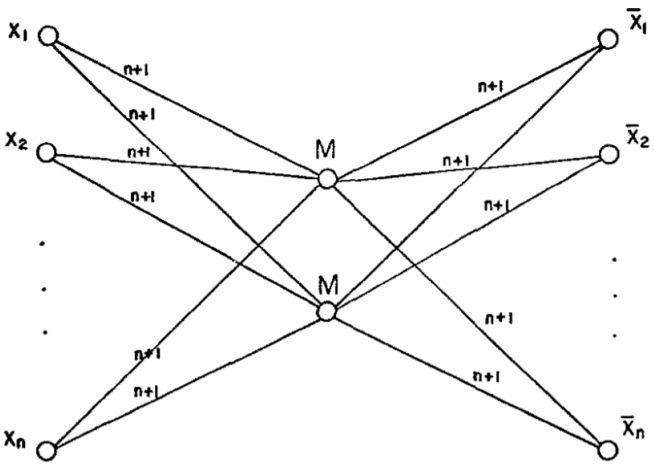
(a)



(b)

*Through problems (a) to (g), we will use the fact the **3-CNF-SAT** problem is NP-complete to prove that the linear array problem is also a NP-complete problem. It is recommended to draw the graphs to understand the problems.*

(a) (4 points) Let S be a set of $2n$ elements. Define a weighted graph H with $V(H) = S \cup \{M, M'\}$, $E(H) = \{(s, M)|s \in S\} \cup \{(s, M')|s \in S\}$, and let all edges have weight $n+1$ (graph H is shown below). Prove that the problem $LAP[H]$ is solvable, and if τ is any labeling that solves $LAP[H]$, it must satisfy $\{\tau(M), \tau(M')\} = \{n+1, n+2\}$.

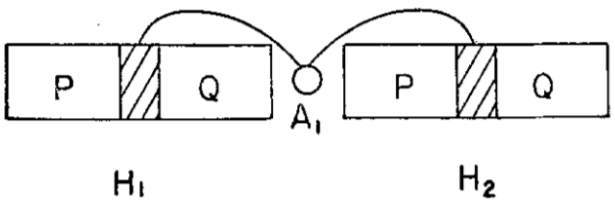


As a consequence, if we denote $P = \{v \in S|\tau(v) < n+1\}$ and $Q = \{v \in S|\tau(v) > n+2\}$, then we have $|P| = |Q| = n$, and P, Q form a partition of S (according to the labeling τ).

Now consider a graph G_1 that contains 2 copies of H introduced in (a) (denoted by H_1, H_2) and an additional vertex A_1, where A_1 is joined with both M, M' of H_1 and H_2 by edges of weight $n+2$.

(b) (4points) Prove that the problem $LAP[G_1]$ described above is solvable, and if τ is any labeling that solves $LAP[G_1]$, it must satisfy $\tau(A_1) = 2n+3$, and $\tau(V(H_1)) = \{1, ..., 2n+2\}$ or $\{2n+4, ..., 4n+5\}$.

By (b), if we have a labeling τ that solves the problem $LAP[G_1]$, its linear arrangement must be the form showning in the graph below. Therefore, we can define sets P_1, Q_1, P_2, Q_2 according to τ using the method similar to the one in (a). However, note that it is not guaranteed that $P_1 = P_2$.



(c) (5 points) Please add weighted edges to G_1 and get a new graph G_2, such that $LAP[G_2]$ is still solvable, and that if τ is any labeling that solves $LAP[G_2]$, it is guaranteed $P_1 = P_2$. (Hint: for each $s \in S$ in H_1, join it to its corresponding vertex in H_2. Determine the edge weight.)

(d) (5 points) Given a subset S' of S with $|S'| = 3$. Please add weighted edges to G_2 and get a new graph G_3, such that $LAP[G_3]$ is still solvable, and that if τ is any labeling that solves $LAP[G_3]$, it is guaranteed $S' \cap Q_1$ is non-empty (recall that Q_1 is determined by τ).

Now assume that the $2n$ elements of S are $x_1, ..., x_n, y_1, ..., y_n$. Consider a weighted graph G_4 that contains $n+1$ copies of H and additional vertices $A_1, ..., A_n$, where A_i is joined with both M, M' of H_i and H_{i+1} by edges of weight $n+2$. Similar to (c), add edges to G_4 such that all partitions of S in $H_1, ..., H_{n+1}$ are the same. Denote P, Q to be this unique partition of S.

(e) (5 points) Please furthermore add edges to G_4 and get a new graph G_5, such that $LAP[G_5]$ is still solvable, and that if τ is a labeling that solves $LAP[G_5]$, it is guaranteed for all $i = 1, ..., n$, there is exactly one of x_i, y_i in P (and the other in Q).

Finally, we can combine the results in (a) to (e) and prove that problem LAP is NP-complete:

(f) (8 points) Consider an instance B of ***3-CNF-SAT*** with literals $x_1, ..., x_n$ and clauses $F_1, ..., F_r$. Construct a weighted graph G which contains $n+r+1$ copies of H and nodes $A_1, ..., A_{n+r}$, with some additional edges, such that $LAP[G]$ is solvable $\iff$ B is a satisfiable boolean formula.

(Hint: let the first n copies of H force S become consistent partition P, Q, and the next r copies of H force the partition $P \cup Q$ to satisfy B, assuming the literals in Q are given the value 1, and the literals in P are given the value 0.)

(g) (4 points) By the fact that ***3-CNF-SAT*** is NP-complete, conclude that problem LAP is NP-complete. You are required to show that the reduction is indeed in polynomial time.

We finally proved that the **Linear Array Problem** is NP-Complete. However, the above argument does not imply that the **Bandwidth Minimization Problem** is NP-complete. To prove that the **Bandwidth Minimization Problem** is NP-complete, we modify the construction of G in (f), instead we get a graph G' such that all edges have weight b or $2b-1$ for some $b \in \mathbb{N}$, and that such that $LAP[G']$ is solvable $\iff$ F is a satisfiable boolean formula. After that, reduce this "restricted" problem $LAP[G']$ to problem $BMP[G'', b']$. To learn the details, you are encouraged to read this paper.

Problem 3 - Lonely Christmas (Hand-Written) (35 points)

On Christmas Eve, Kevin decides to buy some empty boxes of different heights and stack them into gift towers around his Christmas tree. Additionally, he wants the towers to be as low as possible since it is very tiring to lift the boxes. However, he cannot just spread them on the ground since the space around the tree is limited. Please help Kevin find the optimal placement so he can enjoy this holiday.

Formally, an instance of this problem $I = \langle B, h, m \rangle$ is defined as follows:

- B: a finite set of boxes
- h: the height function, $h(i)$ means the height of the box i, $h(i) \in \mathbb{Z}^+$ for all $i \in B$
- m: a positive integer indicates the number of towers

The task is to find a function f from B to $\{1, \ldots, m\}$ such that $H_{\max} = \max_{1 \le k \le m} \sum_{i|f(i)=k} h(i)$ is minimized. Let the optimal $H_{\max}$ of instance I be $\mathrm{OPT}(I)$. Since this problem is NP-Complete (which can be shown by a reduction from Partition), we are going to design a $(1+\epsilon)$-approximation algorithm whose time complexity is polynomial in the input size and ϵ is a constant satisfying $0 < \epsilon \le 1$ and $\frac{1}{\epsilon} \in \mathbb{Z}^+$.

(a) (6 points) Consider the following algorithm:

Greedy-Stacking(I)

i. Place the boxes one by one onto the towers in any order. Always place the next box onto the **lowest** tower currently.

Show that this algorithm is a *2-approximation* for this problem. You may need to focus on the box that determines $H_{\max}$ (i.e., the box on the top of the highest tower). Let i^* denote this box. Think about the relationship between $H_{\max} - h(i^*)$ and $\mathrm{OPT}(I)$.

(b) (6 points) Consider the following algorithm:

Partial-Brute-Force(I)

i. Let $B' = \{i \mid h(i) > \epsilon V\}$ where $V \le \mathrm{OPT}(I)$
ii. Use brute-force approach to find the optimal assignment f' for $I' = \langle B', h, m \rangle$
iii. Starting from f', place $B \setminus B'$ with Greedy-Stacking and return the final assignment.

Show that this algorithm is a $(1+\epsilon)$-*approximation* for this problem.
(hint: consider i^* in (a) again. Discuss 2 cases: (1) $i^* \in B'$ (2) $i^* \notin B'$)

(c) (8 points) Suppose there is a function Oracle(I', V) which returns an assignment f with $H_{\max} \le V$ if and only if there exists such an assignment for a *special* instance I', which will be defined shortly; otherwise, it returns *false*. Consider the following algorithm:

Partial-Rounded(I, V)

i. Construct $B' = \{i \mid h(i) > \epsilon V\}$
ii. Construct new height function h' such that $h'(i) = \lfloor \frac{h(i)}{\mu} \rfloor \mu$ for all $i \in B'$, $\mu = \epsilon^2 V$
iii. Let $I' = \langle B', h', m \rangle$ and $f' =$ Oracle(I', V). If f' is *false*, return *false*.
iv. Starting from f', place $B \setminus B'$ with Greedy-Stacking and return the final assignment.

Show that this algorithm returns a solution with $H_{\max} \leq (1+\epsilon)V$ if $\mathrm{OPT}(I) \leq V$.
(hint: In f', how many boxes can be in one tower?)

(d) (9 points) Now we need to construct $\textsc{Oracle}(I', V)$ such that $\textsc{Partial-Rounded}(I,\ V)$ can run in polynomial time. It is possible because we only need to deal with a **constant** number of different heights after rounding. Complete the following dynamic programming algorithm such that it returns *true* if and only if there exists a solution with $H_{\max} \leq V$ in polynomial time.

Since all of the heights are multiples of μ, a set of boxes can be described by a vector whose j-th entry is the number of boxes of height μj. For any such vector $\vec{n}$, we define $F(\vec{n})$ as the function that outputs the minimum number of towers required to stack all the boxes in $\vec{n}$ with $H_{\max} \leq V$. Let $\vec{n}_{\text{init}}$ be the vector constructed from I'. We can see that $\textsc{Oracle}(I', V)$ returns an assignment if and only if $F(\vec{n}_{\text{init}}) \leq m$.

(d-1) (2 points) Show that the length of $\vec{n}$ is bounded by $\frac{1}{\epsilon^2}$.
(You can assume that if $h'(i) > V$ for some $i \in B'$, we can directly return *false*)

(d-2) (3 points) Recursively define $F(\vec{n})$. You may need this set $U = \{\vec{u} \mid \sum_j u_j(\mu j) \leq V\}$.

(d-3) (4 points) Show that the time complexity of this algorithm is $O(|B|^{\frac{1}{\epsilon^2}})$

(e) (7 points) Design a $(1+\epsilon)$-approximation algorithm for this problem with $\textsc{Partial-Rounded}(I, V)$ as a subroutine. Show that the time complexity of your algorithm is polynomial in the input size. Assume that the maximum number of bits needed to represent a number is ℓ.