Assignment - 1

Problem 1: Multi-Taxi Routing with Capacity-Constrained Roads

You are given a city represented as a **weighted undirected graph** with `N=8` intersections (nodes) and roads (edges).

- There are multiple taxis (`T`) and passengers (`P`).
- Each taxi can carry **exactly one passenger** from pickup to drop.
- Each node has **2D coordinates** `(x, y)`.
- Distances between connected nodes are given explicitly.
- Taxis move at a constant speed of **40 km/h**.

Travel time rule:

Travel Time (hours)=Distance/Speed

Congestion Rule

- At most **2 taxis** can use the same road (edge) at the same time.
- If a **third taxi** tries to enter while 2 are already on that edge, it must **wait 30 minutes** before entering.

Objective

Plan routes for all taxis so that all passengers are picked up and dropped off, **minimizing the total completion time**.
Use **A*** search (specify heuristic) for pathfinding.

Input Format

N M P W S

- `N` = number of nodes (fixed at 8)
- `M` = number of edges
- `P` = number of passengers = number of taxis
- `W` = waiting time in minutes (30)
- `S` = speed in km/h (40)

Next N lines: node coordinates (x y)
Next M lines: roads with distances (u v d)
Next P lines: passenger trips (pickup drop)

Sample Input

8 9 3 30 40
0 0
30 0
80 0
30 60
120 0
30 120
60 120
110 120
1 2 30
2 3 50
2 4 60

3 5 40
4 6 70
5 7 20
6 7 30
7 8 50
3 6 90
2 7
1 8
3 4

Interpretation

- `N=8` nodes
- `M=9` edges
- `P=3` passengers (3 taxis)
- `W=30 min` wait penalty
- `S=40 km/h` speed

Passenger/taxi trips

- Taxi 1: `2 → 7`
- Taxi 2: `1 → 8`
- Taxi 3: `3 → 4`

Sample Output

Taxi 1:
Passenger 2->7
Route: 2 -> 3 -> 5 -> 7
Total time = 165.0 minutes

Taxi 2:

Passenger 1->8
Route: 1 -> 2 -> 3 -> 5 -> 7 -> 8
WAIT on edge (2->3): 30 minutes
Total time = 315.0 minutes

Taxi 3:
Passenger 3->4
Route: 3 -> 2 -> 4
Total time = 165.0 minutes

Total Completion Time = 645.0 minutes
(span = 315.0 minutes)

Explanation

Constants / Formula

- Speed = **40 km/h**
- Time per km = 60/40=1.5 minutes/km
- Edge time = 1.5×d

Edges (distance in km)

1–2: 30
2–3: 50
2–4: 60
3–5: 40
4–6: 70
5–7: 20
6–7: 30
7–8: 50
3–6: 90

Taxi 1 (start 2 → dest 7)

Shortest path: 2 → 3 → 5 → 7

- 2→3: 50 km → 75.0 min → [0.0, 75.0)
- 3→5: 40 km → 60.0 min → [75.0, 135.0)
- 5→7: 20 km → 30.0 min → [135.0, 165.0)
  Total = 165.0 min

Taxi 3 (start 3 → dest 4)

Shortest path: 3 → 2 → 4 (110 km)

- 3→2: 50 km → 75.0 min → [0.0, 75.0)
- 2→4: 60 km → 90.0 min → [75.0, 165.0)
  Total = 165.0 min

Taxi 2 (start 1 → dest 8)

Shortest path: 1 → 2 → 3 → 5 → 7 → 8 (190 km)
Nominal times:

- 1→2: 30 km → 45.0 min → [0.0, 45.0)
- 2→3: 50 km → 75.0 min → [45.0, 120.0)
- 3→5: 40 km → 60.0 min → [120.0, 180.0)
- 5→7: 20 km → 30.0 min → [180.0, 210.0)
- 7→8: 50 km → 75.0 min → [210.0, 285.0)
  Nominal total = **285.0 min**

Congestion Check

- Edge **2–3** overlaps:
    - Taxi1: [0.0, 75.0)
    - Taxi3: [0.0, 75.0)
    - Taxi2 wants to enter at t = 45.0 → would cause **3 taxis on edge 2–3**.
- Congestion rule applies → Taxi2 must **wait 30 min** at node 2.

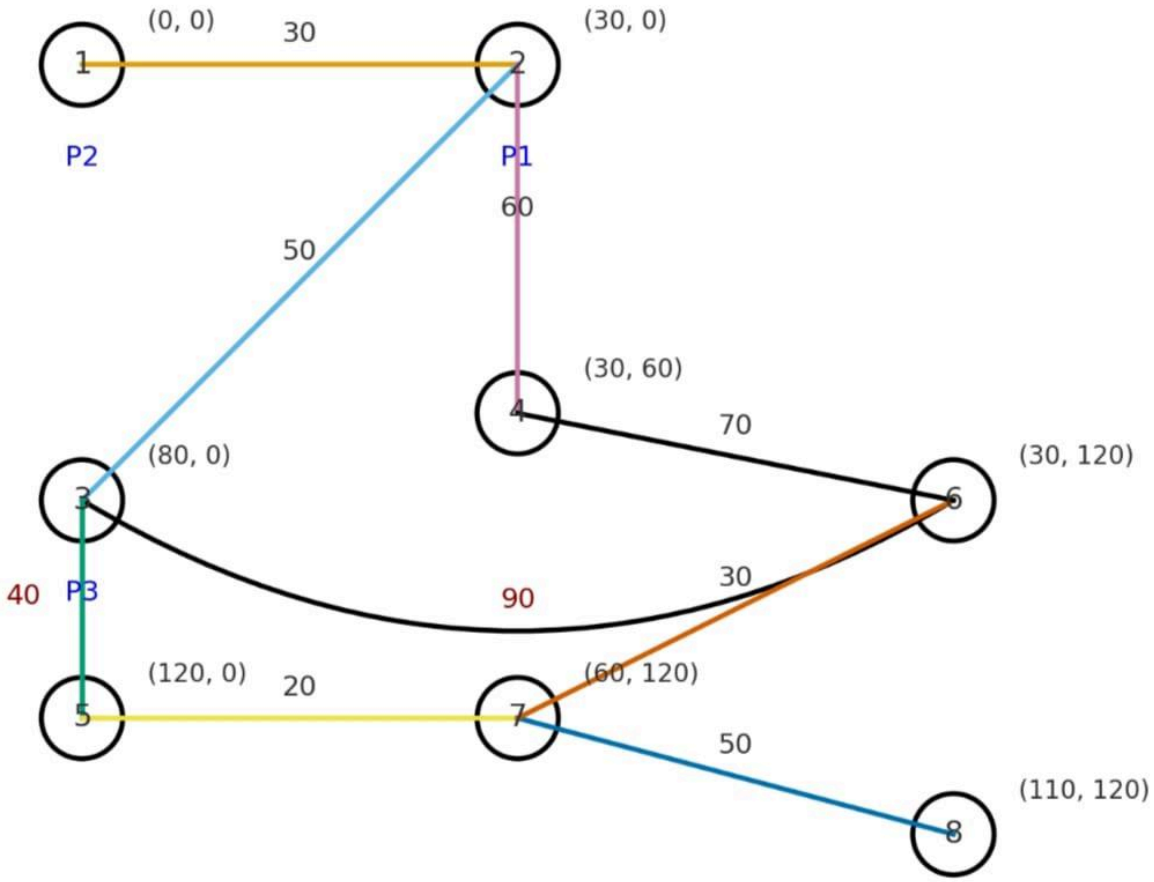
Taxi 2 adjusted timeline

- 1→2: [0.0, 45.0)
- **Wait at 2: [45.0, 75.0)**
- 2→3: [75.0, 150.0)
- 3→5: [150.0, 210.0)
- 5→7: [210.0, 240.0)
- 7→8: [240.0, 315.0)
  Total = 315.0 min

Final Totals & Metrics

- Taxi 1 = 165.0 min
- Taxi 2 = 315.0 min (with wait)
- Taxi 3 = 165.0 min

Overall Completion Time = 645.0 min
Makespan = 315.0 min


(0, 0)
30
(30, 0)
1
2
P2
P1
60
50
(30, 60)
4
70
(80, 0)
(30, 120)
3
6
40
P3
90
30
(120, 0)
20
(60, 120)
5
7
50
(110, 120)
8

Problem 2: Reservoir Distribution with Valves

Problem Statement

You are given **3 reservoirs** connected in a **mesh topology** by bidirectional valves. Each reservoir has an **initial amount of water** and a **target amount** it must reach.

Water transfers happen by opening a valve between two reservoirs. When a valve is open, water flows until:

- The destination reservoir is full , OR
- The source reservoir reaches the safety threshold (≥20% of its total capacity).

At most **one valve can be opened at a time**. The total amount of water is conserved.

Your task is to determine the sequence of valve operations that transforms the initial distribution into the target distribution, while minimizing the number of valve openings.

Input Format

- First line: `C1 C2 C3` → capacities of reservoirs
- Second line: `I1 I2 I3` → initial water amounts
- Third line: `T1 T2 T3` → target water amounts

Output Format

Print the sequence of valve operations in the format:

Open valve (X→Y) using BFS, DFS and A* search (specify heuristics).

- Finally print the **number of valve operations** used.

Constraints

- 1 ≤ Ci ≤ 100 (capacity of reservoir i)
- 0 ≤ Ii ≤ Ci (initial water in reservoir i)
- 0 ≤ Ti ≤ Ci (target water in reservoir i)
- I1 + I2 + I3 = T1 + T2 + T3 (total water conserved)
- 20% ≤ water in reservoir ≤ 100% at any time

Example

Input

8.0 5.0 3.0

8.0 0.0 0.0

2.4 5.0 0.6

Output

Open valve (1->2)
Open valve (2->3)
Open valve (3->1)
Open valve (1->2)
Number of valve operations = 4

Explanation

Start state: `(R1, R2, R3) = (8.0, 0.0, 0.0)`.

Operation 1 — `Open valve (1 → 2)`

- `src = R1 = 8.0`, `min1 = 1.6` ⇒ `src` can give up to `8.0 - 1.6 = 6.4`.
- `dst = R2 = 0.0`, capacity `C2 = 5.0` ⇒ `dst` can accept up to `5.0 - 0.0 = 5.0`.
- `transfer = min(6.4, 5.0) = 5.0` (destination becomes full).
- New state:
  `R1 = 8.0 - 5.0 = 3.0`
  `R2 = 0.0 + 5.0 = 5.0`
  `R3 = 0.0`
  → `(3.0, 5.0, 0.0)`

Operation 2 — `Open valve (2 → 3)`

- `src = R2 = 5.0`, `min2 = 1.0` ⇒ `src` can give up to `5.0 - 1.0 = 4.0`.
- `dst = R3 = 0.0`, capacity `C3 = 3.0` ⇒ `dst` can accept up to `3.0 - 0.0 = 3.0`.
- `transfer = min(4.0, 3.0) = 3.0` (destination becomes full).
- New state:
  `R1 = 3.0` (unchanged)
  `R2 = 5.0 - 3.0 = 2.0`
  `R3 = 0.0 + 3.0 = 3.0`
  → `(3.0, 2.0, 3.0)`

Operation 3 — `Open valve (3 → 1)`

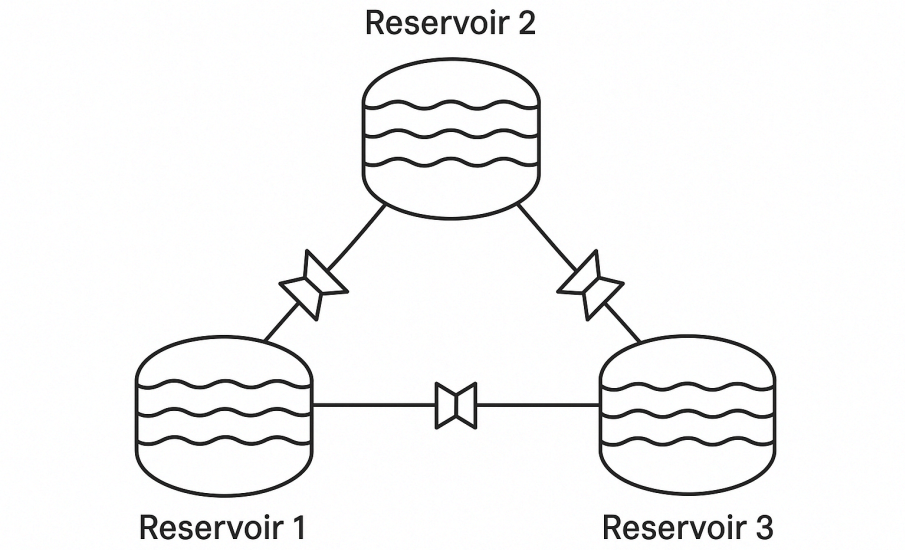
- `src = R3 = 3.0`, `min3 = 0.6` ⇒ `src` can give up to `3.0 - 0.6 = 2.4`.

- `dst = R1 = 3.0`, capacity `C1 = 8.0` ⇒ `dst` can accept up to `8.0 - 3.0 = 5.0`.
- `transfer = min(2.4, 5.0) = 2.4` (source reaches safety threshold).
- New state:
  `R1 = 3.0 + 2.4 = 5.4`
  `R2 = 2.0` (unchanged)
  `R3 = 3.0 - 2.4 = 0.6`
  → `(5.4, 2.0, 0.6)`

Operation 4 — `Open valve (1 → 2)`

- `src = R1 = 5.4`, `min1 = 1.6` ⇒ `src` can give up to `5.4 - 1.6 = 3.8`.
- `dst = R2 = 2.0`, capacity `C2 = 5.0` ⇒ `dst` can accept up to `5.0 - 2.0 = 3.0`.
- `transfer = min(3.8, 3.0) = 3.0` (destination becomes full).
- New state:
  `R1 = 5.4 - 3.0 = 2.4`
  `R2 = 2.0 + 3.0 = 5.0`
  `R3 = 0.6`
  → `(2.4, 5.0, 0.6)`

This equals the target `(2.4, 5.0, 0.6)` exactly.



Problem 3: The Game of Tic-Tac-Toe

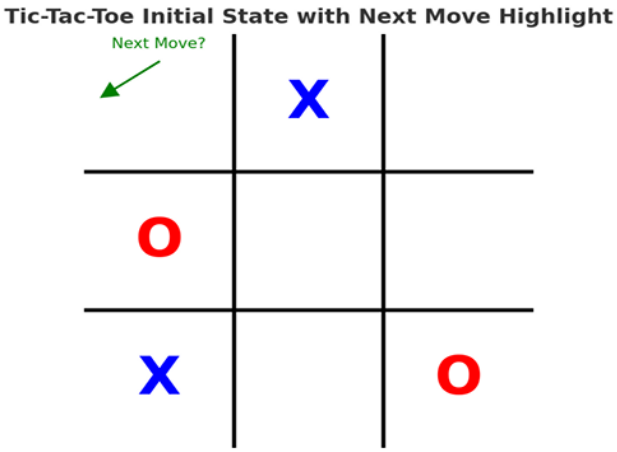
Problem Statement

Consider the game of tic-tac-toe where X is the MAX player. Given the game board below where it is X's turn to play next, show the game tree.

__ | X | __
0 | __ | __
X | __ | 0

The evaluation function on all the leaf nodes is given by:
Evaluation(s) = $8X_3(s) + 3X_2(s) + X_1(s) – (8O_3(s) + 3O_2(s) + O_1(s))$ Where $X_n(s)$ is the number of rows, columns, or diagonals in state s with exactly n number of X's and no O's and $O_n(s)$ is the number of rows, columns, or diagonals in states with exactly n number of O's and no X's.

Use the minimax algorithm and alpha-beta pruning algorithm to calculate the optimal move.


Tic-Tac-Toe Initial State with Next Move Highlight
Next Move?
X
O
X
O

Problem 4: Chip Placement Optimization using Integer Descent (Column-wise Restricted)

Problem Description

We are given a grid matrix G of size $W \times H$. Each cell represents one unit of chip surface. A set of rectangular chips (modules) must be placed on the grid.

- Each chip c_i has width w_i and height h_i with $h_i > w_i$ (tall rectangles).
- Each chip is **locked to a specific row** y_i.
- The chip may only move **horizontally by integer steps**, i.e. its x-coordinate is always an integer.
- Some chips are connected via a netlist.

The objective is to place all chips on the grid such that the **conflict score** is minimized. We apply **gradient descent optimization**: chips shift left/right in unit steps if this reduces the conflict score.

Conflict Score

The conflict score is defined as:

$$\text{Conflict}(P) = \sum_{(c_i,c_j)\in\text{Connections}} \text{WiringCost}(c_i, c_j) + \sum_{(c_i,c_j)\in\text{Overlaps}} \text{OverlapBlocks}(c_i, c_j).$$

1. Wiring Cost

For each pair of connected chips (c_i, c_j):

$$\text{WiringCost}(c_i, c_j) = \max\big(0,\ x_j - (x_i + w_i),\ x_i - (x_j + w_j)\big) + |y_i - y_j|.$$

Here, x_i are treated as **integer variables**.

2. Overlap Cost

Instead of continuous area, the overlap cost is the number of **grid blocks occupied by more than one chip**:

$$\text{OverlapBlocks}(c_i, c_j) = \#\big(\text{grid cells covered by both } c_i \text{ and } c_j\big).$$

This provides a discrete but easily interpretable penalty.

Initial State :

We work with a fixed instance on a 10×10 grid. All chips are taller than wide.

Chips (width $\times$ height), initial locked-row y and initial integer x:

$$\begin{aligned}
c_1 &: 2 \times 4 \quad \text{at row } y = 0, \quad (x = 0)\\
c_2 &: 2 \times 5 \quad \text{at row } y = 1, \quad (x = 1)\\
c_3 &: 1 \times 3 \quad \text{at row } y = 0, \quad (x = 1)\\
c_4 &: 2 \times 5 \quad \text{at row } y = 4, \quad (x = 2)\\
c_5 &: 2 \times 4 \quad \text{at row } y = 3, \quad (x = 3)\\
c_6 &: 1 \times 4 \quad \text{at row } y = 2, \quad (x = 2)\\
c_7 &: 2 \times 5 \quad \text{at row } y = 5, \quad (x = 0)\\
c_8 &: 1 \times 3 \quad \text{at row } y = 6, \quad (x = 4)
\end{aligned}$$

Connections (netlist):

$$\{(1,2),(2,6),(2,3),(3,5),(4,5),(5,6),(1,6),(7,4),(7,2),(8,5)\}.$$

Matrix Representation of Initial Placement

The 10×10 grid (rows $y = 9 \to 0$). Cells show chip indices. Overlaps are shown in red (two chips) or magenta (three chips).

7	7	0	0	0	0	0	0	0	0
7	7	4	4	8	0	0	0	0	0
7	7	4	4	8	0	0	0	0	0
7	7	4	4/5	5/8	0	0	0	0	0
7	2/7	2/4/6	4/5	5	0	0	0	0	0
0	2	2/4/6	4/5	5	0	0	0	0	0
1	1/2	2/6	5	5	0	0	0	0	0
1	1/2/3	2/6	0	0	0	0	0	0	0
1	1/2/3	2	0	0	0	0	0	0	0
1	1/3	0	0	0	0	0	0	0	0

Tasks

1. Implement integer descent to minimize the conflict score starting from the initial state above.
2. Track and plot the conflict score over iterations; report the final discrete conflict score.
3. Report final chip positions (x_i, y_i) and the final discrete conflict score.