Tutorial 1

CS4234 AY2023/2024 Semester 1

Welcome!

Introductions

Let’s get to know each other :D

Your Turn!

Can also share anything you'd like to share with the class :D

Q1

First question!

Question 1

Can we solve **Min-Vertex-Cover** in polynomial time if all vertices in the input graph have **degrees not more than 2**?

What if we want to solve **Min-Weight-Vertex-Cover** variant on that graph type?

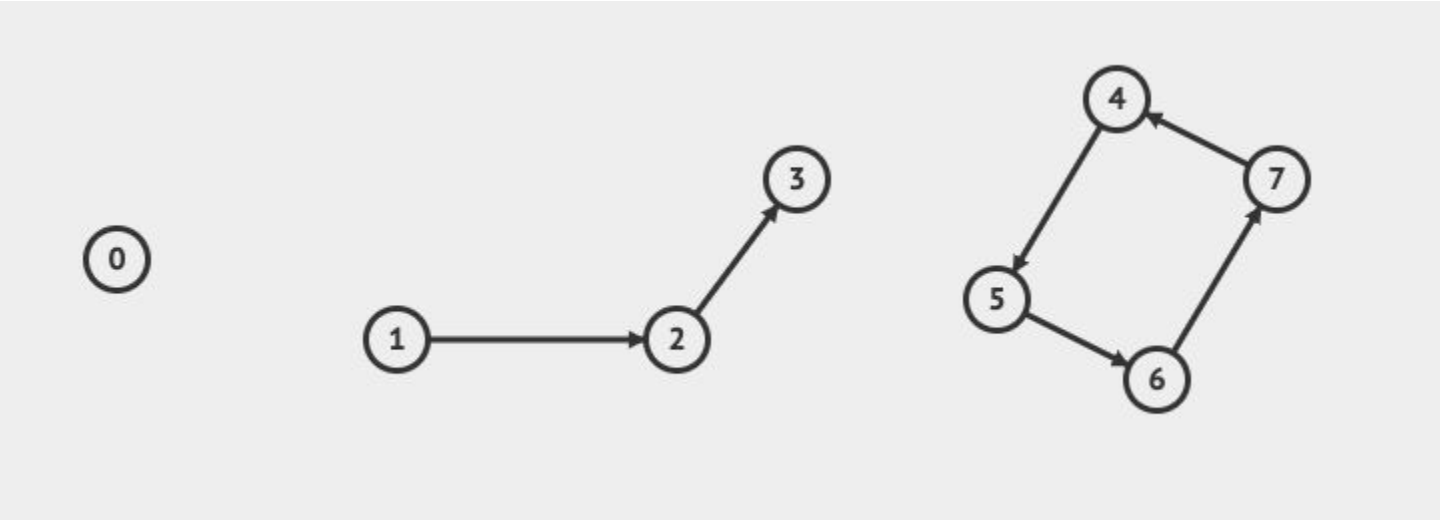
Question 1

How does the graph look like?

Question 1

How does the graph look like?

Note: ignore the direction of the edges



Question 1

How does the graph look like?

A collection of CCs where each CC is either a line or a cycle

Question 1

How does the graph look like?

A collection of CCs where each CC is either a line or a cycle

If line => MVC = floor(|CC| / 2)

If Cycle => MVC = ceil(|CC| / 2)

We can compute CC in O(N) . Hence MVC for this graph is solvable in poly time :D

Question 1

Now how about the weighted variant?

Sadly |CC| / 2 formula no longer works :(

Question 1

Now how about the weighted variant?

Sadly |CC| / 2 formula no longer works :(

Still solvable in poly time using O(N) DP on tree. Note that a line graph is also a tree.

DP(i, flag) = MWVC on tree rooted at i, flag == 1 we take this vertex i, otherwise we don’t take

DP(i, 1) = weight(i) + sum_all_children [min(DP(children, 0), DP(children, 1))]

DP(i, 0) = sum_all_children [DP(children, 1)]

Question 1

Now how about the weighted variant?

How about the cycle? Key idea is to break the cycle into a tree.

The easiest way is for every vertex i, just try to force take it and solve the rest of the vertices optimally (the rest forms a line!). Then we take the minimum among all vertex i that we force to take.

Overall runs in poly time :D

Q2

Greedy MVC :O

Question 2

In lecture, we have seen a Dynamic Programming (DP) solution to solve the Min-Vertex-Cover on a (Binary) Tree.

How about the following Greedy Algorithm (solution for CLRS Ex 35.1-4):

First, take any leaf in the tree, then add its parent to the (minimal) vertex cover, then delete the leaf and parent and all associated edges. Repeat this process until there is no vertex remain in the tree. Is this a correct Greedy Algorithm?

Hint: Check https://visualgo.net/en/mvc, `MVC on Tree', the` Greedy MVC on Tree' option. Can it be used on the Min-Weight-Vertex-Cover variant?

Question 2

Yes it is! It works for MVC, sadly doesn’t work on MWVC (you still need DP for the weighted variant).

Question 2

Yes it is! It works for MVC, sadly doesn't work on MWVC (you still need DP for the weighted variant).

We can use optimal substructure + exchange argument to prove greedy algorithm.

Optimal Substructure: MVC(G) = 1 + MVC(G - V)

Question 2

Yes it is! It works for MVC, sadly doesn't work on MWVC (you still need DP for the weighted variant).

We can use optimal substructure + exchange argument to prove greedy algorithm.

Exchange argument:

Suppose in some optimal solution OPT we take a leaf instead of it's parent. Now we do the exchange and take the parent instead of the leaf, call this solution OPT*.

Notice that |OPT| == |OPT*| since we remove one vertex (leaf) and add another vertex (parent) from OPT to create OPT*. Moreover, taking parent covers the edge (between parent and leaf) that is covered before by leaf in OPT. OPT* is a valid MVC.

Q3

Linear Programming :O

Question 3

A carpenter makes tables and chairs. **Each table** can be sold for **a profit of 25 SGD** and **each chair** for **a profit of 11 SGD**. **The carpenter** can afford to spend up to **40 hours** per week working and takes **5 hours to make a table** and **3 hours to make a chair**.

The owner requires that the carpenter makes **at least 3 times as many chairs as tables** (so that he can package it as dining table set). **Tables take up 4 times as much storage space as chairs** and there is room for **at most 10 tables each week**.

Formulate this problem as a Linear Programming problem (write in standard form) and solve it (graphically, with Excel, with lp solve, or with a (Simplex? or Brute Force?) program :O). Now if the solution is a oating point numbers, please round them so that we have an Integer solution. Are you sure your Integer solution is the optimal one?

Question 3

Let T = table, C = chair

Objective function: maximize 25T + 11C

5T + 3C <= 40 (working hours)

3T - C <= 0 (3 times as many chairs)

4T + C <= 40 (storage space)

T >= 0

C >= 0

Question 3

Solving yields T = 2.85, C = 8.57, and max profit = 165.71

In real life, you cannot have 0.85 tables…

Question 3

Solving yields T = 2.85, C = 8.57, and max profit = 165.71

In real life, you cannot have 0.85 tables…

If we add an integer constraint, then it becomes ILP. We can try to round the LP answer to yield an ILP answer.

Question 3

Solving yields T = 2.85, C = 8.57, and max profit = 165.71

In real life, you cannot have 0.85 tables…

If we add an integer constraint, then it becomes ILP. We can try to round the LP answer to yield an ILP answer.

Rounding down: T = 2, C = 8 => max profit = 138

Rounding up: T = 3, C = 9 => breaks 5T + 3C <= 40 constraint

Rounding up partially: T = 2, C = 9 => max profit = 149

But is (T = 2, C = 9) the best we can do?

Question 3

We can do T = 2, C = 10 => profit = 160 and this is the optimal answer to the ILP problem.

This highlights that we cannot simply extend solution of LP to ILP (and hence the NP-hardness).

Prove of ILP is NP-hard: (Can be reduced from 3sat)

https://math.stackexchange.com/questions/2564236/how-can-i-formulate-the-3-sat-problem-as-a-0-1-linear-integer-program

Q4

Is this NP-hard?

Question 4

You are asked to do Max-Clique visualization in VisuAlgo.

Given a graph G = (V, E) and an integer k, is there a subset C of size k such that C is a clique in G?

The Max-Clique optimization variant seeks for the subset C with the largest possible k in G.

Here are Steven's requirements:

- VisuAlgo user must be able to draw his/her own graph input, as per VisuAlgo standard.
- Steven dislikes graphs that are drawn with edge crossings, so he has put on a check in his “Draw Graph" feature so that only graphs without edge crossing are allowed in VisuAlgo.
- VisuAlgo must be very responsive, i.e., upon clicking “Show Max-Clique" button, a JavaScript routine has to quickly compute the answer in not more than 1 second. Longer than that, Steven says that the visitors will press Alt+F4 instead.

What is the best algorithm that will you implement as JavaScript routine of your Max-Clique visualization in VisuAlgo?

What is the time complexity? Does your algorithm seeks for an approximate solution or an optimal solution? What are the rough limit of n and m, the number of vertices and edges, that you will set in your visualization?

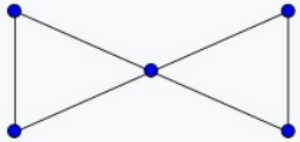
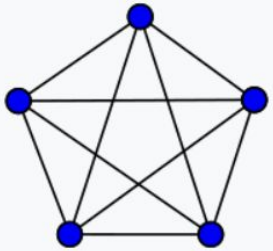
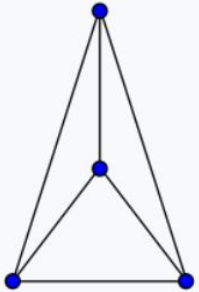
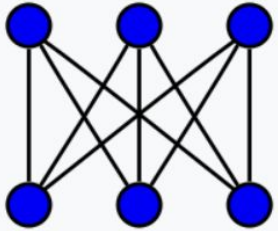
Question 4

Is Max-Clique NP-Hard in general? Are there any special properties in the requirements that can help us?

(Graph Theory crash course ><)

Question 4: Planar Graph

Graphs without edge crossing => planar graph

Example graphs	
Planar	**Nonplanar**
 Butterfly graph	 Complete graph K_5
 Complete graph K_4	 Utility graph $K_{3,3}$

Question 4 : Planar Graph Criteria (Kuratowski and Wagner)

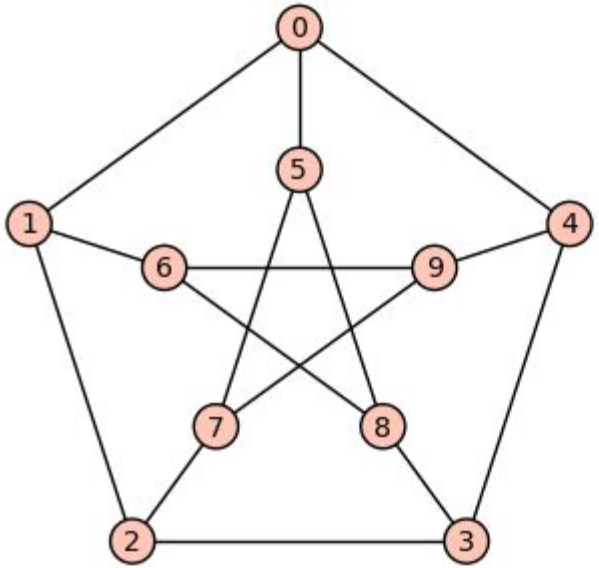
(** Just know this)

Kuratowski and Wagner shows that

A graph is planar if and only if it does not contain K5 and K(3, 3) minor.

This means that

Biggest clique a planar graph can have is of size 4!



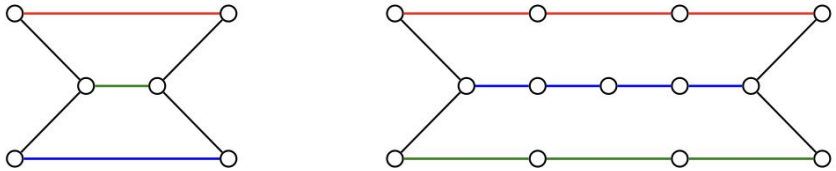
A demo on how peterson graph (special type of graph) is non-planar (because it contains K(3,3) minor)

MA3233 surprise (OPTIONAL)

9.4 Criterions of Planarity

Recall that the *realization* $R(G) = V(G) \cup E(G)$ can be viewed as a topological subspace of $\mathbb{R}^3$, and G is *planar* if and only if $R(G)$ is homeomorphic to a subset of $\mathbb{R}^2$.

A graph H is said to be a **subdivision** of a graph G if H is obtained from G by subdividing some of the edges; that is, adding vertices on some edges; or equivalently, replacing edges by paths.



It is clear that $R(G)$ and $R(H)$ are homeomorphic. We thus conclude that they are simultaneously planar.

If G is not planar, then so is every subdivision H of G, and every graph containing H as subgraph. In particular, if a graph contains a subdivision of K_5 or $K_{3,3}$, then it is not planar.

It is somewhat surprising that the converse of the remark above is also true, which is proved by Kuratowski.

Theorem 9.4.1 (Kuratowski)**.** A graph is planar if and only if it does not contain a subdivision of K_5 or $K_{3,3}$.

Credit to prof Wang Fei's MA3233 Lecture Notes

MA3233 surprise (OPTIONAL)

Lemma 9.4.2. For $uv \in E(G)$, if the contraction G/uv has a subdivision of K_5 or $K_{3,3}$, so does G.

prove omitted cause it is too long, google online or ping me if interested

Corollary 9.4.4. If G is planar and $uv \in E(G)$, then G/uv is planar.

Proof. If G is planar, then by Theorem 9.4.1 G does not have a subdivision of K_5 or $K_{3,3}$. By Lemma 9.4.2, G/uv does not have a subdivision of K_5 or $K_{3,3}$. By Theorem 9.4.1 again, G/uv is planar. □

MA3233 surprise (OPTIONAL)

A graph H is a **minor** of G, written $H \preccurlyeq G$, if H is a subgraph of a graph obtained from G by a sequence of edge-contractions.

Theorem 9.4.5 (Wagner)**.** A graph is planar if and only if it does not contain K_5 or $K_{3,3}$ as a minor.

Proof. If G is planar, then by Corollary 9.4.4, any edge-contraction of G is planar. Consequently, any minor of G is planar. In particular, G does not contain K_5 or $K_{3,3}$ as a minor.

Conversely, suppose that G is non-planar. By Theorem 9.4.1, G contains a subgraph H which is a subdivision of K_5 or $K_{3,3}$. If e is an edge of H that is incident to a subdivision vertex of H, then H/e is a subgraph of G/e, which is still a subdivision of K_5 or $K_{3,3}$. Continuing this process, we arrive at K_5 or $K_{3,3}$, which is a subgraph of a graph obtained by a sequence of edge-contractions. Hence, G contains K_5 or $K_{3,3}$ as a minor. □

MA3233 surprise (OPTIONAL)

Theorem 9.2.1 (Euler's Polyhedron Theorem)**.** Suppose that a planar representation of a connected planar graph G has n vertices, m edges and f faces. Then

$$n - m + f = 2.$$

A planar graph G is called **maximal** if it is not a proper spanning subgraph of another planar graph.

Fix a planar representation of a planar graph of order $n \geq 3$. If it has a face whose boundary is not a triangle, i.e., C_3, then we can join a pair of non-adjacent vertices on this boundary to have a planar graph of larger size. This process terminates if and only all faces are bounded by triangles.

Lemma 9.3.2. A planar graph G of order $n \geq 3$ is maximal if and only if the size $m = 3n - 6$.

Proof. Let f be the number of faces of G. We have observed that G is maximal if and only if each face is bounded by three edges, i.e., $3f = 2m$. Substitute into Euler's formula (Theorem 9.2.1) to get

$$n - m + 2m/3 = 2 \Rightarrow m = 3n - 6.$$ □

Question 4

So don’t need to test all K > 5, because all must be non planar.

Using the theorem, there’s an obvious O(N^4) algorithm: try out all tuple of 4 vertices, 3 vertices, 2 vertices and report the largest clique found. (nC4 + nC3 + nC2 + nC1)

But there’s also a consequence of planar graphs that allows us to do O(N^2). Can you spot it? Hint: num of edges <= 3n - 6.

Question 4

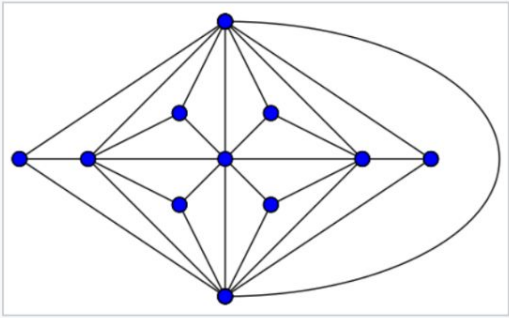
But there's also a consequence of planar graphs that allows us to do O(N^2). Can you spot it?

Hint: num of edges <= 3n - 6.

We can just try all pairs of edges and check how big the clique the endpoints make.

Maximal planar graphs [edit]

A simple graph is called **maximal planar** if it is planar but adding any edge (on the given vertex set) would destroy that property. All faces (including the outer one) are then bounded by three edges, explaining the alternative term **plane triangulation**. The alternative names "triangular graph"[4] or "triangulated graph"[5] have also been used, but are ambiguous, as they more commonly refer to the line graph of a complete graph and to the chordal graphs respectively. Every maximal planar graph is a least 3-connected.



The Goldner–Harary graph is maximal planar. All its faces are bounded by three edges.

If a maximal planar graph has v vertices with $v > 2$, then it has precisely $3v - 6$ edges and $2v - 4$ faces.

Apollonian networks are the maximal planar graphs formed by repeatedly splitting triangular faces into triples of smaller triangles. Equivalently, they are the planar 3-trees.

Strangulated graphs are the graphs in which every peripheral cycle is a triangle. In a maximal planar graph (or more generally a polyhedral graph) the peripheral cycles are the faces, so maximal planar graphs are strangulated. The strangulated graphs include also the chordal graphs, and are exactly the graphs that can be formed by clique-sums (without deleting edges) of complete graphs and maximal planar graphs.[6]

num of edges <= 3n - 6
Equality exists when it is a maximal planar graph

Question 4

Using the theorem, there's an obvious O(N^4) algorithm: try out all tuple of 4 vertices, 3 vertices, 2 vertices and report the largest clique found.

But there's also a consequence of planar graphs that allows us to do O(N^2). Can you spot it? Hint: num of edges <= 3n - 6.

We can just try all pairs of edges (O(m^2)) and check how big the clique the endpoints make.

O(n^2) as m == O(n)

Q5

Is this poly time as well?

Question 5

All Steven's requirements are identical, just that you are another FYP student who is assigned to do Graph-Coloring visualization. The decision version of Graph-Coloring is defined as follows:

Given a graph G = (V, E) and an integer k, can we assign a color (an integer in [1..k]) to each vertex such that no two adjacent vertices (that share an edge) are of the same color? The Graph-Coloring optimization variant seeks for coloring with the smallest possible k.

Question 5

A magic result (again): In any planar graph, you need at most 4 color to color the graph.

Theorem 9.5.4 (Four Colour Theorem). For any planar graph G, $\chi(G) \leq 4$.

This theorem is proved using computer. The proof is not wildly acceptable by all mathematicians because it is infeasible to check by hand. On the other hand, we have weaker results which can be easily proved.

This gives rise to a O(4^n) bruteforce algorithm where we just try all possible 4 colors on each vertices.

Unfortunately no known planar graph property (yet?) that is able to solve this in poly time :(

Tutorial Questions Done

Any questions?

23/24 S1 PS1 Debrief

Did you manage to solve all?

A: Kattis - crazydriver

The code utilizes a greedy strategy to minimize the total cost of the journey through the gates. At each step, the driver chooses the least cost action available, whether it be moving forward or driving back and forth until a gate opens.

The strategy here is to always proceed with the least cost action in any given situation. This is done by choosing the road with the lowest cost when forced to wait for a gate to open and always moving forward as soon as it is possible (i.e., the gate is open).

B: Kattis - up and away

Main ideas:

- Assume that Steve does not have any fireworks. Then, we can run Dijkstra's to solve the problem, where we only traverse directed edges between nodes of decreasing/equal heights.

Now, if we have k fireworks, assume we have k layers of the graph without any fireworks. Then, we run dijkstra to solve this new problem. Assuming we are currently on layer j, every time we use F fireworks by traversing through an edge (a, b) we visit the node b on layer (j + F). The answer we want is then the minimum value at node x on any layer.

https://github.com/stevenhalim/cpbook-code/blob/master/ch4/sssp/dijkstra.py

C: Kattis - quantum

Main ideas:

- The problem of transforming binary words using defined quantum operations can be modeled as a graph, where the vertices represent the possible binary words and the edges represent the quantum operations with their associated energy costs.
- By constructing the graph in this manner, we can leverage Dijkstra's algorithm to find the minimal energy cost for transforming each binary word into its desired state.
- When a node is popped from the priority queue (pq), we can guarantee that we have found the shortest path to that node from the starting node.

https://github.com/stevenhalim/cpbook-code/blob/master/ch4/sssp/dijkstra.py

D: Kattis - crowd control

Main ideas:

- Prioritize edges with greatest capacity.
- If train stations p and q are already connected, ignore other edges of smaller capacity that connects these two stations (they are useless!).
- Basically Kruskal's algorithm, but for **Maximal** Spanning Tree (MST).
- Path-find in MST from 0 to n-1.

E: Kattis - hired help

Main ideas:

- Make new employees clear up shoes with the earliest deadlines as much as possible.
- Prove greedy choice:
  - Let OPT be an optimal solution.
  - If OPT = 0, greedy choice is correct. (If greedy choice gives ≥1 employee, contradiction!)
  - Otherwise, OPT ≥1:
    - All employees have valid set of *K* deadlines (able to complete all deadlines).
    - Always can contain employee who clears earliest deadlines. How?
    - Pick any employee, say A, and add/swap deadlines with other employees.
      - A lacks an earlier deadline that B has. Swap!
      - A clears an earlier deadline→still valid.
      - B clears a later deadline→must still be valid.

F: Kattis - recursion + rand = !fun

Main ideas:

- Consider all possible values of ***(myRNG() % b)*** and ***(myRNG() % c)***: Can ***f(i - 1 - (myRNG() % c))*** produce ***(k - a + myRNG() % b)***?
- Recursive DP with memoization. (Memoize possible and impossible pairs (i, k)).
- Some trimming can be done (e.g. if ***k > i*(a+b-1)+1***, return false).

G: Kattis - jabuke

Main ideas:

- R x S matrix ***M*** representing the field.
- For each apple tree on (*i*, *j*), update cells in row *i* of ***M*** the shortest distance from that cell to any apple tree in the same row.
- Search for shortest square distance by iterating through the column of the apple.

H: Kattis - grid mst

Main ideas:

- Apply Prim's algorithm, starting from any point.
- Expand MST using "BFS" while keeping track of the distance from currently included points to the current coordinate. (Can use priority queue)
- After discovering new points, update visited coordinates with new distances via "BFS".
- Repeat until all points are included.

End of Tutorial

Any last questions?

Thank you!

See ya next week!

Tutorial 2

CS4234 AY2023/2024 Semester 1

Welcome!

Credit to Audrey (TA in CS4234 22/23 S1) for making the slide

Q1

ILP

Question 1

We discussed Integer Linear Programming (ILP) in Lecture 02. Now express the proven NP-hard optimization problem: **Min-Set-Cover** problem that we discussed in Lecture 03a as an **ILP**!

Question 1

$$\min \sum_{i \in S} x_i \quad \text{where} \quad // \; x_i = 1/0 = \text{use/not-use this subset } i, \text{ min total used}$$

$$x_i \in \{0,1\} \quad // \text{ this is the standard integer/Boolean constraint}$$

$$\forall j \in X, \quad \sum_{i \in S \text{ where } j \in S_i} x_i \geq 1 \quad // \text{ for every element in } X, \text{ at least one selected set must cover it}$$

Q2

MST as ILP

Question 2

Now express what we know as a P problem: **Min-Spanning-Tree** problem as an **ILP**! Will you solve MST problem that way instead of using what you already know from earlier modules?

Question 2

$$\min \sum_{ij \in E} x_{ij} \times w(i,j) \quad \text{where} \quad // \; x_{ij} = 1/0 = \text{use/not-use edge } (i,j) \in E$$

$$\forall ij \in E, x_{ij} \in \{0,1\} \quad // \text{ standard}$$

$$\sum_{ij \in E} x_{ij} = n-1 \quad // \text{ choose only } n\text{-1 edges (convert this to} \leq)$$

$$\forall S \subset V, S \neq \emptyset, \sum_{ij \in E \text{ where } i \in S \text{ and } j \in S} x_{ij} \leq |S| - 1 \quad // \text{ any non } \emptyset \text{ subset of } k \text{ vertices have} \leq k\text{-1 edges}$$

Q3

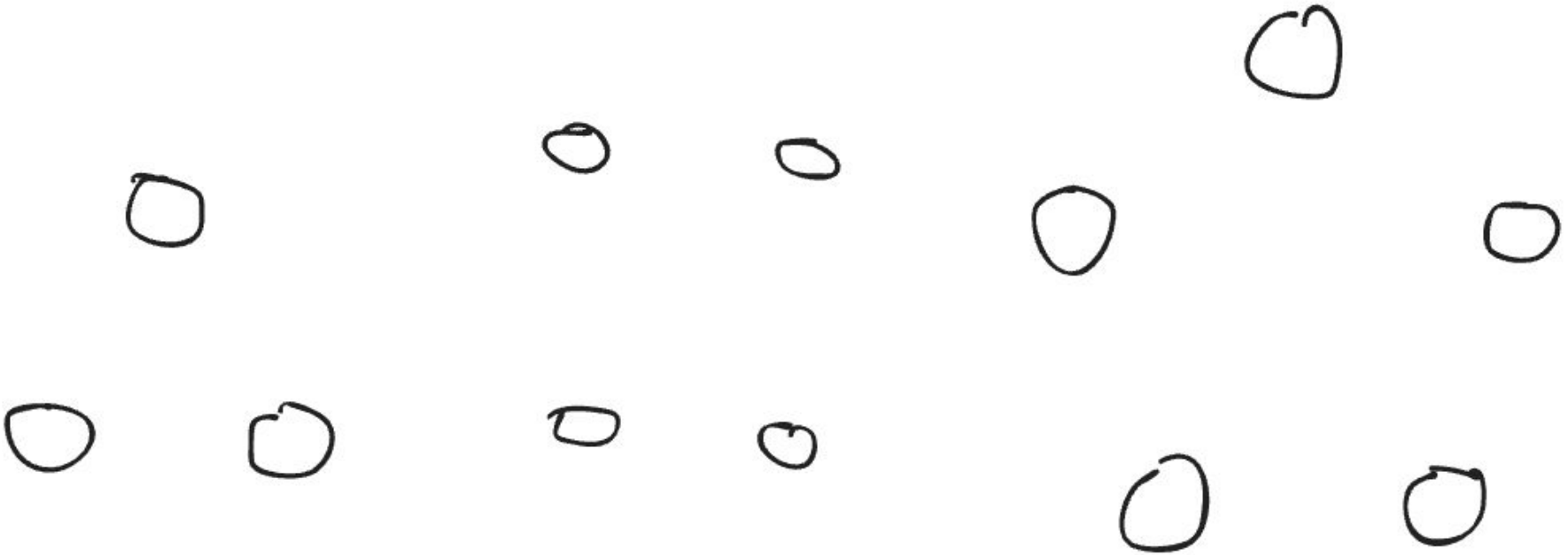
Euclidean Steiner

Question 3

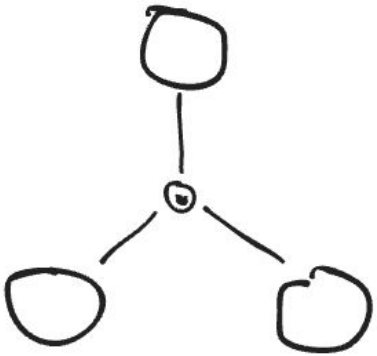
In Lecture03b, we discussed the **Euclidean-Steiner-Tree** problem. Now let's spend some time discussing (and maybe proving) some of these properties that were only discussed briefly in Lecture 03b. The TA will draw a few (e.g., n = 7) ‘random’ points on Euclidean plane (the whiteboard) and we will try to apply these known properties to help us derive an optimal (or at least (visually) good enough) Euclidean Steiner Tree:

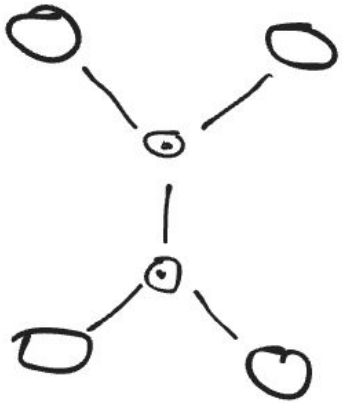
- Each Steiner point in an optimal solution has degree 3.
- The three lines entering a Steiner point form 120 degree angles, in an optimal solution.
- An optimal solution has at most n - 2 steiner points.

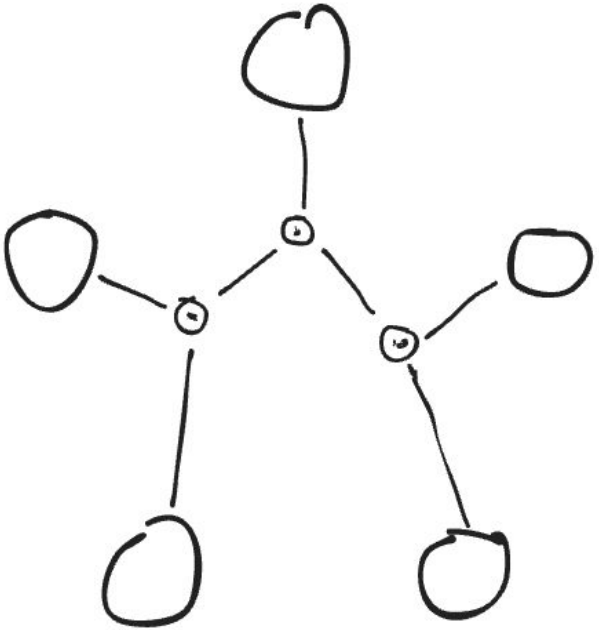
Question 3



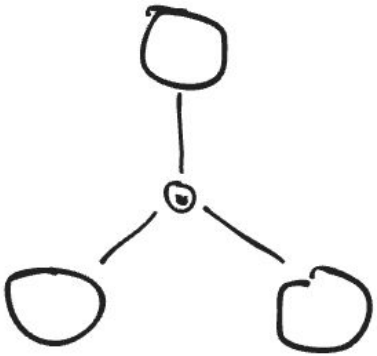
Question 3

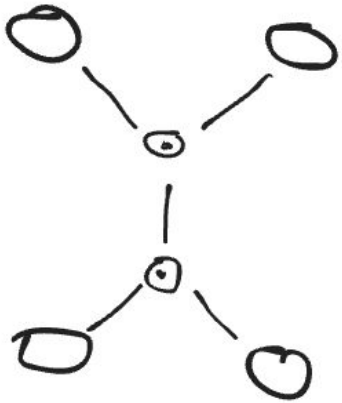


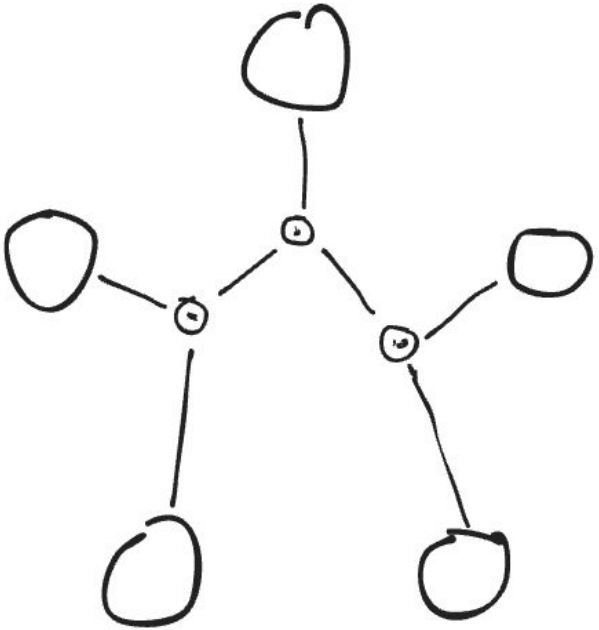




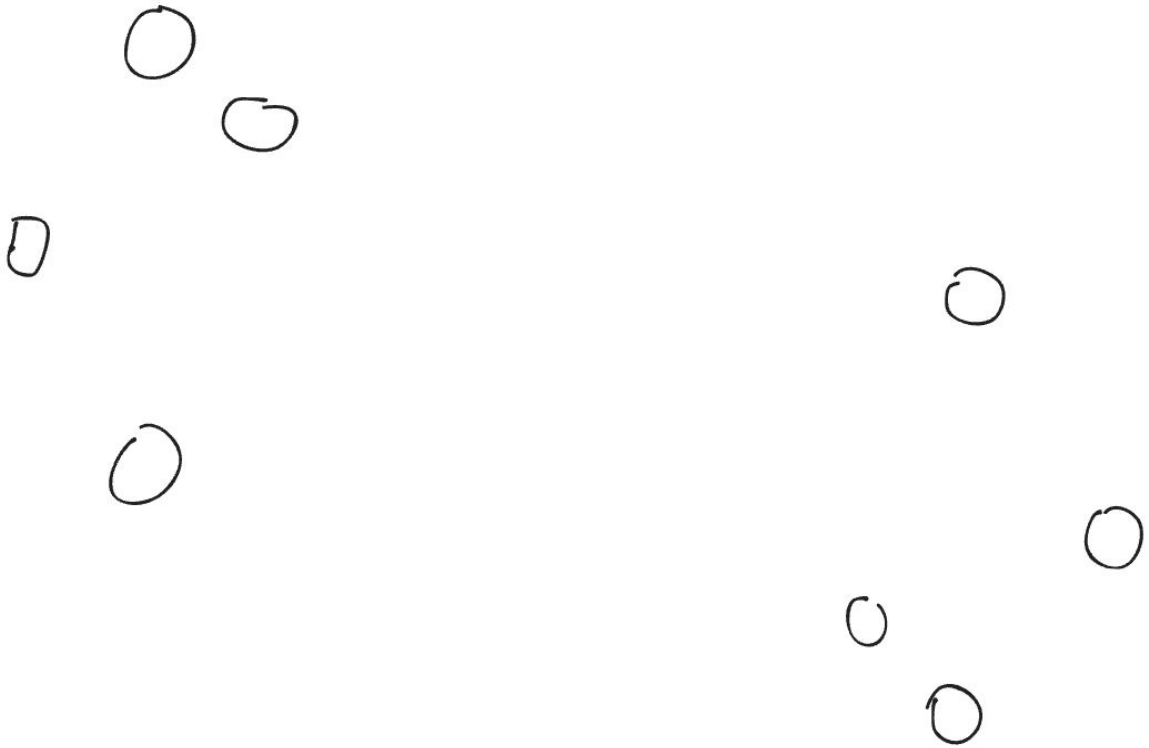
Question 3



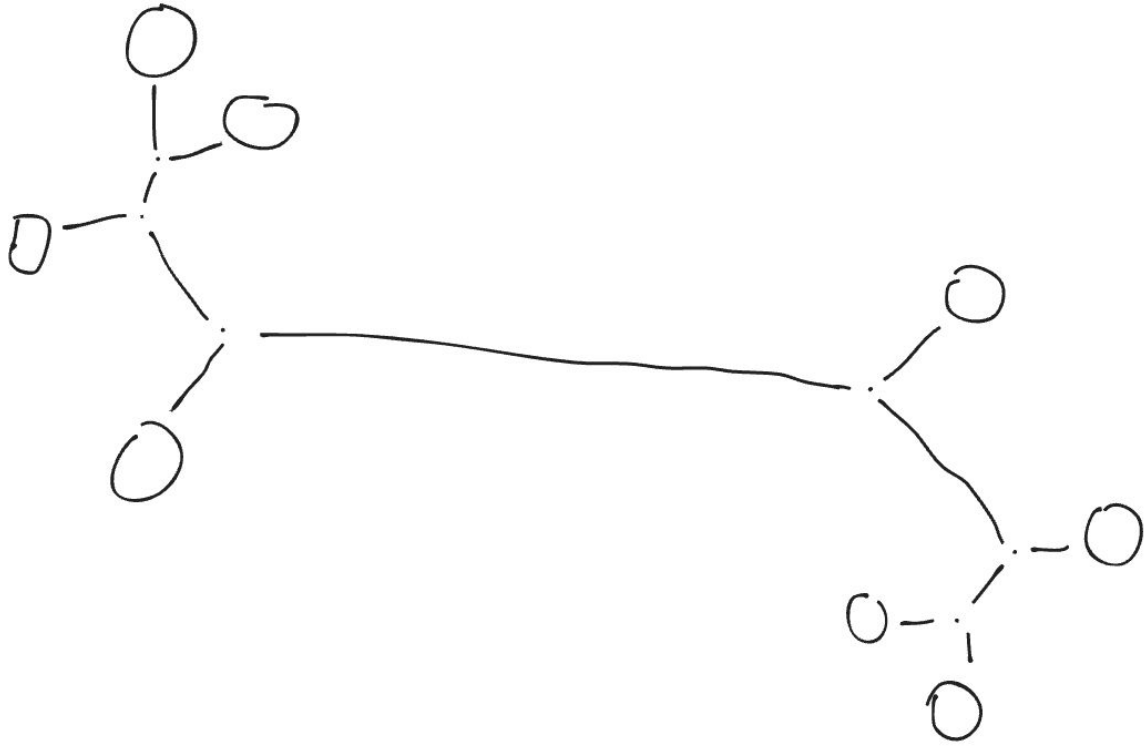




Question 3



Question 3



Question 3

Heuristics:

- Each Steiner point in an optimal solution has degree 3.
- The three lines entering a Steiner point form 120 degree angles, in an optimal solution.
- An optimal solution has at most n - 2 steiner points

Q4

NP-Hard

Question 4

Steven needs to continually increase the number of (NP-)hard problems that has been exposed to CS4234 students so far (to have a more interesting Midterm Test and Final Assessment). So let's study this yet other problem Min-Weight-Feedback-Edge-Set (MWFES) and Min-Feedback-Edge-Set (MFES) for the unweighted version.

Question 4

You are given a directed weighted graph G = (V, E) with weights w : E -> R. Your goal is to delete the some edges to produce an acyclic graph with the maximum remaining edge weights. That is: Find a minimum weight set of edges F such that G = (V, E \ F) is acyclic.

Question 4

Draw an (small) example graph and explain this problem to your tutorial group.

Question 4

If G is an undirected weighted graph, give an efficient algorithm to solve this problem optimally.

Question 4

If G is an undirected weighted graph, give an efficient algorithm to solve this problem optimally.

Find the maximum spanning tree T of G. Then, F = all other edges not in T.

Question 4

In the case of directed graphs, the MFES (or MWFES) problem is NP-hard (for now, just assume that it is really NP-hard and later in (the optional) Part 4, we will show you the details). Now, consider the following (heuristic) algorithm: Given a directed graph G = (V, E) and weights 1 (or w) for MFES (or MWFES) version, respectively:

- Let the vertices be V = v1, v2, ..., vn
- Build graph Gf = (V, Ef) containing only forward edges. That is, Ef = (vi, vj) where i < j.
- Build graph Gb = (V, Eb) containing only backward edges. That is, Eb = (vj, vi) where i < j.
- Return the graph Gf or Gb containing more edges (or higher total weighted edges for the weighted MWFES version) as the acyclic graph (and the other non-selected edges as the removed edges F).

Question 4

Show that both Gf and Gb are valid solutions to the MFES/MWFES problem.

Question 4

Show that both Gf and Gb are valid solutions to the MFES/MWFES problem.

Suppose it is not acyclic, then there exist a cycle, let the vertices be v_(x1), …, v_(xk)

WLOG only forward edges remain. x1 < x2 < … < xk. Since there is an edge xk to x1, we must have xk < x1, contradiction.

Question 4

Show that the algorithm is not a good approximation algorithm for MFES/MWFES.

Consider a graph with exactly |E| / 2 forward and backward edges and it is already acyclic. Our algo will remove |E| / 2 edges, while the optimal answer is remove 0.

Question 4

Now consider the complementary (the dual) problem:

Find a maximum weight subgraph G’ that is acyclic. Notice that weight here refers to the sum of the edge weights. Now show that the (heuristic) algorithm above is actually a 2-approximation algorithm.

Question 4

Now consider the complementary (the dual) problem:

Find a maximum weight subgraph G’ that is acyclic. Notice that weight here refers to the sum of the edge weights. Now show that the (heuristic) algorithm above is actually a 2-approximation algorithm.

Let OPT be the optimal answer. We clearly have OPT <= sum of all edge weights in G.

Notice that sum of all edge weights in G = sum weights in Gf + sum weights in Gb

Hence, OPT <= 2 * Algo output => Algo output is 2-OPT

Question 4

Now which PS2 problem is exactly this one? :D

Q5

2-SAT

Question 5

$$\phi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

Find a truth assignment to x1, x2, x3

Question 5

$$\phi = (x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (\neg x_2 \vee \neg x_3)$$

Find a truth assignment to x1, x2, x3

x1 = T, x3 = T, x2 = F

Question 5

Which problem in PS2 is actually 2-CNF-SAT problem?

Question 5

Which problem in PS2 is actually 2-CNF-SAT problem?

Problem D, buriedtreasure2

Question 5

What is the high-level idea to solve this special case of an NP-complete decision problem?

Question 5

What is the high-level idea to solve this special case of an NP-complete decision problem?

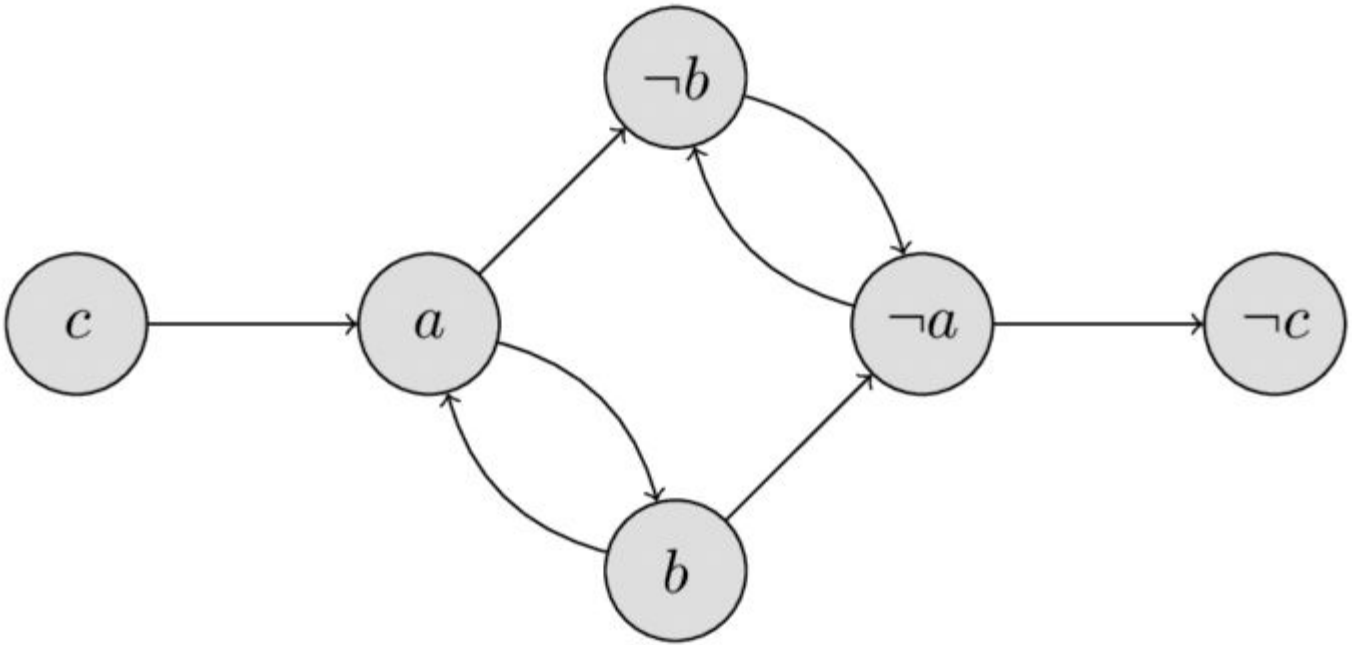
The key idea is that each (A V B) clause can be written in two alternate forms:

- (not A) => B
- (not B) => A

We can build a directed graph on 2n vertices (1 vertex for xi and another for (negation xi)) and there is an edge between xi and xj if xi => xj is derived from one of the clause.

Question 5

$$(a \vee \neg b) \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \wedge (a \vee \neg c)$$



Question 5

The observation is that if A can reach (negation A) and (negation A) can reach A, then it is impossible to find an assignment to 2-SAT.

Turns out the condition is sufficient, if the above case does not occur, we can always find an assignment.

Hence, to see if the 2-SAT has a satisfying assignment or not, we just need to make sure for each variable A, it's negation is not in the same Strongly Connected Component.

End of Tutorial

Any last questions?

Thank you!

See ya next week!

Tutorial 3

CS4234 AY2023/2024 Semester 1

Welcome!

Q1

Approximation Impossibility

Question 1

In Lecture 04, Steven did not show any approximation algorithm for the **G-NR-TSP variant of TSP** (that is, the general, non-metric version and without repeated vertex). He says that ‘it is **(NP-)hard even to approximate**’. Why?

Question 1

Suppose there exist a polynomial time approximation A, let the approximation ratio produced by p >= 1 (that is Output(A) <= p * OPT)

The key idea is to reduce a HAMILTONIAN-CYCLE instance to TSP instance and use A to solve it.

Question 1

Consider any instance G = (V, E) of HAMILTONIAN-CYCLE. Create an instance of TSP G' = (V', E'), where V' = V and for every e in E':

weight(e) = 1 if e in E

weight(e) = p * |V| + 1

Question 1

Blue = 1, Red = p * |V| + 1

Question 1

Run A on G', let the output be A*. Now we prove that G contains hamiltonian cycle iff A* <= p * |V|

(=>)

Suppose that G contains hamiltonian cycle. Then that cycle is a possible tour in G'. Hence OPT in G' <= hamiltonian cycle in G = |V|.

A* <= p * OPT <= p * |V|

Question 1

(<=)

Now suppose A^* <= p * |V| on G'. Then it means that all edge chosen must come from G, since the all the other edges e not in G have weight p * |V| + 1.

Hence, that tour that A^* have chosen must be a valid hamiltonian cycle in G.

Now that the reduction is complete, we see that we can use A to solve HAMILTONIAN-CYCLE in polynomial time. Contradiction.

Hence, such polynomial time approximation cannot exist.

Q2

TSP With Missing Edges

Question 2

In class, we formulated the TSP (4 variants) in terms of a set of points V and a distance function d that gives the distance between any two points in V (thus, a complete graph with V 2 edges that surely has (V − 1)! Hamiltonian Cycles).

What if the input to the problem is a graph G = (V, E) with E weighted edges and $0 \leq |E| \leq V * (V - 1)/2$?

Define a version of TSP for graphs, and explain whether or not it remains approximate-able using the techniques discussed in class. (Consider these: What if there is no Hamiltonian Cycle in the input to begin with? What if the input graph is actually disconnected?)

Question 2

If the graph is disconnected or have no hamiltonian cycle, what should we do?

Question 2

If the graph is disconnected or have no hamiltonian cycle, what should we do?

We cannot do anything, since there is no TSP tour in the first place :(

Question 2

Now what if we have a hamiltonian cycle, but the graph is not complete, what can we do?

Question 2

Now what if we have a hamiltonian cycle, but the graph is not complete, what can we do?

We can try doing metric completion!

Then, we can use whatever approximation algorithm we learn in class (2-approx or Christofides' algorithm) to finish after the metric completion.

Question 2

But a little caveat…

Notice that by using metric completion, if we choose a “virtual edge” in our TSP, we have to convert them back to the equivalent path using the actual edges in the original graph.

In the Non-Repeat version of TSP (M-NR), this may pose a problem as the equivalent path may actually induce repeated vertices.

Hence, this is why usually TSP input is complete graph.

Q3

Bitonic TSP

Question 3

Think about the **Euclidean TSP where each point has (x, y)-coordinates** to identify it (for simplicity, let's assume that all **x-coordinates of the n points are different**).

Imagine that Steven only wants "cycles" that proceed in one direction, e.g., left-to-right. For example, a legal output is a cycle (v1, v2, v3, v4, v1) where for each i < n, we know that v(i).x < v(i+1).x. (**Only in the last step of the cycle**, going back from vn to v1, **we are allowed to go to the left**.) Is there an efficient algorithm to solve this non-standard version of TSP problem? Next, give an example where the cycle that is found in this case is very bad compared to the optimal TSP cycle.

Question 3

Is this solvable in polynomial time? How?

Question 3

Is this solvable in polynomial time? How?

Yup!

Just sort all the points in increasing order of their x-coordinates. Let the sorted order be v1, v2, …, vn. Then the Optimal TSP is simply v1 -> v2 -> … -> vn -> v1.

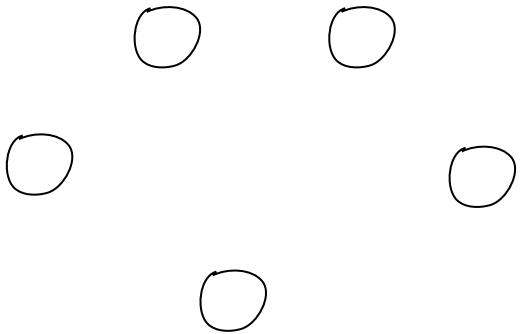
Question 3

Why can this “special” cycle be very bad compared to the actual optimal TSP with no restriction?

Question 3

Why can this “special” cycle be very bad compared to the actual optimal TSP with no restriction?

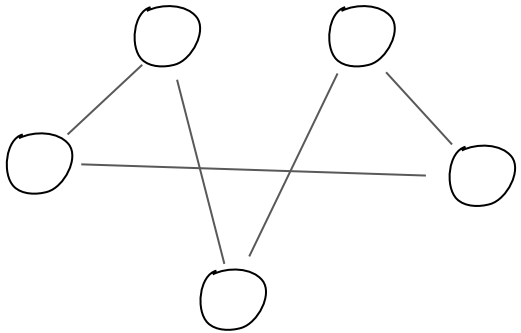
Consider this testcase:



Question 3

Why can this “special” cycle be very bad compared to the actual optimal TSP with no restriction?

Consider this testcase:



Q4

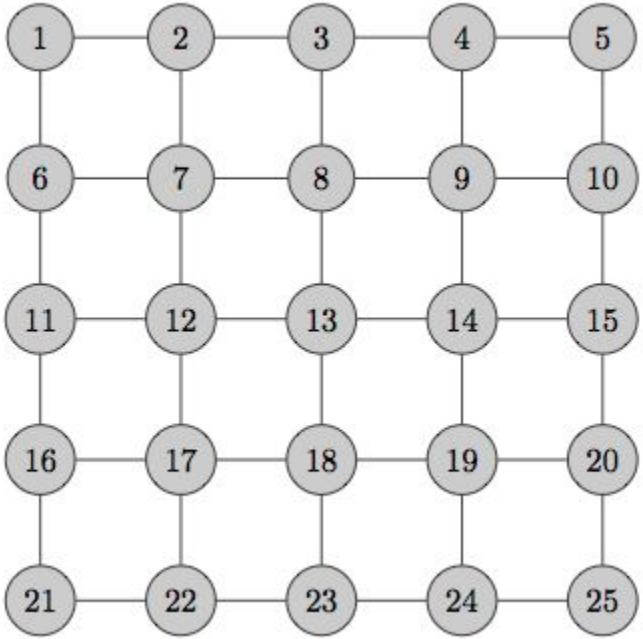
More NP-Hard Problems

Question 4

Steven needs to continually increase the number of (NP-)hard problems that has been exposed to CS4234 students so far (to have more interesting Midterm Test and Final Assessment). So let's study this yet other problem Max-Independent-Set (MIS). It is very similar to MVC and defined as follows:

Given a graph G = (V,E), pick the maximum-size set I ⊂ V so that no two vertices in I share an edge. You are told that MIS is also an NP-hard optimization problem (proof omitted, but you can reduce NP-hard VC to IS easily), but here you are given a network that is arrange as a grid, as in Figure 1:

Question 4



Question 4

The grid has n vertices, and each of the vertices (except for those on the edges) has four neighbors. The goal of this question is to develop an algorithm for finding a maximum-sized Max-Independent-Set (MIS) for this graph (or an approximation).

What is the MIS (and its size) of the figure before?

Question 4

The grid has n vertices, and each of the vertices (except for those on the edges) has four neighbors. The goal of this question is to develop an algorithm for finding a maximum-sized Max-Independent-Set (MIS) for this graph (or an approximation).

What is the MIS (and its size) of the figure before?

Optimal size is 13

Constructed by taking all odd numbered vertices

Question 4

Consider the following greedy algorithm for graph G = (V, E):

- Set I = ∅;
- Repeat until V is empty:
  - Choose any arbitrary vertex u ∈ V
  - Add u to I
  - Delete u all the neighbors of u ∈ V

Now argue that this algorithm is a 2.5-approximation of optimal on grid graph like in the figure before. (But first, show that it is a correct algorithm that produces an independent set!)

Question 4

Why is it correct?

Question 4

Why is it correct?

For each each chosen vertex v in I, by the last step of the algorithm, it’s neighbours cannot be in I. Hence I is a valid independent set.

Question 4

Why is it 2.5 approx?

Question 4

Why is it 2.5 approx?

Let OPT be the optimal MIS size as usual, notice that

OPT <= n / 2

Proof is by Pigeon Hole Principle, for the sake of simplicity assume n is even:

Consider the disjoint sets:

{1, 2}, {3, 4}, {5, 6}, …, {n - 1, n}

Question 4

Consider the disjoint sets:

{1, 2}, {3, 4}, {5, 6}, …, {n - 1, n}

Notice that each numbers in the set are neighbors of each other.

If we take >= n / 2 + 1 vertices, then by PHP at least 2 vertices chosen are in the same set => are neighbours.

Hence any Independent Set has size <= n / 2.

Question 4

Notice that in the algorithm, for every vertex v we are choosing, we are removing its 4 neighbours from the chance to be chosen into I.

Hence, for every loop of the algorithm, we are adding one vertex and removing 5.

$|I| >= n / 5$.

Combining with the previous bound:

$|I| >= n / 5 >= 2 OPT / 5$

Algo is 2.5 approx.

Question 4

Write down MIS as an ILP!

Question 4

Write down MIS as an ILP!

Very similar to MVC, idea is to use boolean indicator variables (again!)

$$\begin{aligned} \max \sum_{i=1}^{n} x_i \quad & \text{where} \quad && // \ x_i = 1/0 = \text{use/not-use this vertex } i, \text{ maximize size} \\ \forall i \in V, x_i \in \quad & \{0,1\} && // \ \text{this is the standard integer/Boolean constraint} \\ \forall i,j \in V, x_i + x_j \leq \quad & 1 && // \ \text{this is the Independent Set constraint} \end{aligned}$$

Q5

SUBSET-SUM

Question 5

One of the PS3 problems involves another new NP-hard problem that has not been discussed earlier (in lecture and/or previous tutorial). Which PS3 problem is it? What is the underlying NP-complete decision problem? What is the general idea to tackle this problem?

Question 5

To those that have read PS3, which one is this?

Question 5

To those that have read PS3, which one is this?

Problem A - SUMSETS

Given set, find largest d for some distinct a, b, c, d such that

$a + b + c = d$

Special case of SUBSET-SUM

Question 5

Some starting ideas:

- Tricky due to negative numbers, how to deal with them?
- Reduce to 3-SUM

Tutorial Questions Done

Any questions?

PS2 Debrief

Did you manage to solve all?

A: Kattis - chicken joggers

Main ideas:

- MVC on tree (DP)
- Some pre-selected
- Ignore all vertices out-of-reach
- Edge case - root/campus is a leaf

B: Kattis - metube

Main ideas:

- MSC (with bitmasks)
- <= 30 videos
  - current selected videos
- How to test that a category is covered?
  - each category covered by some videos
- Iterate through each category to cover
- DP (current selected video to optimal answer)

C: Kattis - exitsinexcess

Main ideas:

- Believe in the result of tutorial 2 🙏

Count the number of forward edges and the number of backward edges (as defined in tutorial 2). Output the one with the smaller total edges.

For proof can read tutorial 2 again.

D: Kattis - buriedtreasure2

Main ideas:

- Can model as 2-SAT problem
- 2-SAT is in P by tutorial 2

Model the problem as 2-SAT instance. For any requirement m1 = i and m2 = j, we make the clause (x_i OR x_j). Change x_i to its negative literal if i < 0, do the same with x_j.

Using result in tutorial 2, use any SCC algorithm and check that none of x_i and its negative literal are in the same SCC.

End of Tutorial

Any last questions?

Thank you!

See ya next week!

Tutorial 4

CS4234 AY2023/2024 Semester 1

Welcome!

Q1

ILP Again…

Question 1

Write **Max-Flow as a Linear Program**. Again, are you going to solve Max-Flow that way?

Question 1

$$\max \sum_{j:(s,j)\in E} f_{sj} \quad \text{where} \quad \text{// maximize flow that goes out from source vertex } s$$

$$\forall i,j \in E, f_{ij} \geq 0 \quad \text{// } f_{ij} \text{ is the (integer) flow in edge } (i,j)$$

$$\forall i,j \in E, f_{ij} \leq c_{ij} \quad \text{// capacity constraints}$$

$$\forall j \in V \backslash \{s,t\}, \sum_{i:(i,j)\in E} f_{ij} = \sum_{k:(j,k)\in E} f_{jk} \quad \text{// flow conservation constraints}$$

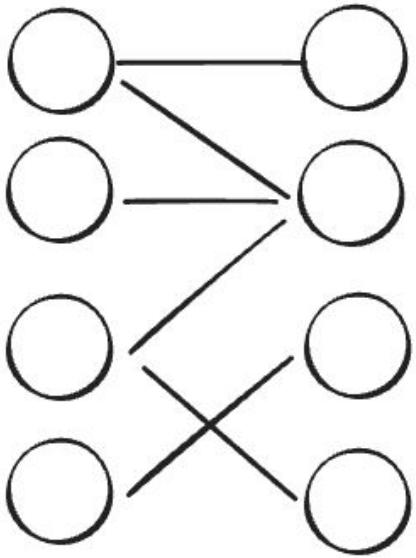
Q2

MCBM to Max Flow

Question 2

Show how to use (standard) **Ford-Fulkerson algorithm** to find a maximum sized matching on Bipartite Graph, or formally known as the **Max-Cardinality-Bipartite-Matching (MCBM)** that will be properly discussed on Lecture 6 (please read ahead). Prove that the result is a matching, and that it is the maximum- sized matching. Analyze the running time of your algorithm.

Question 2

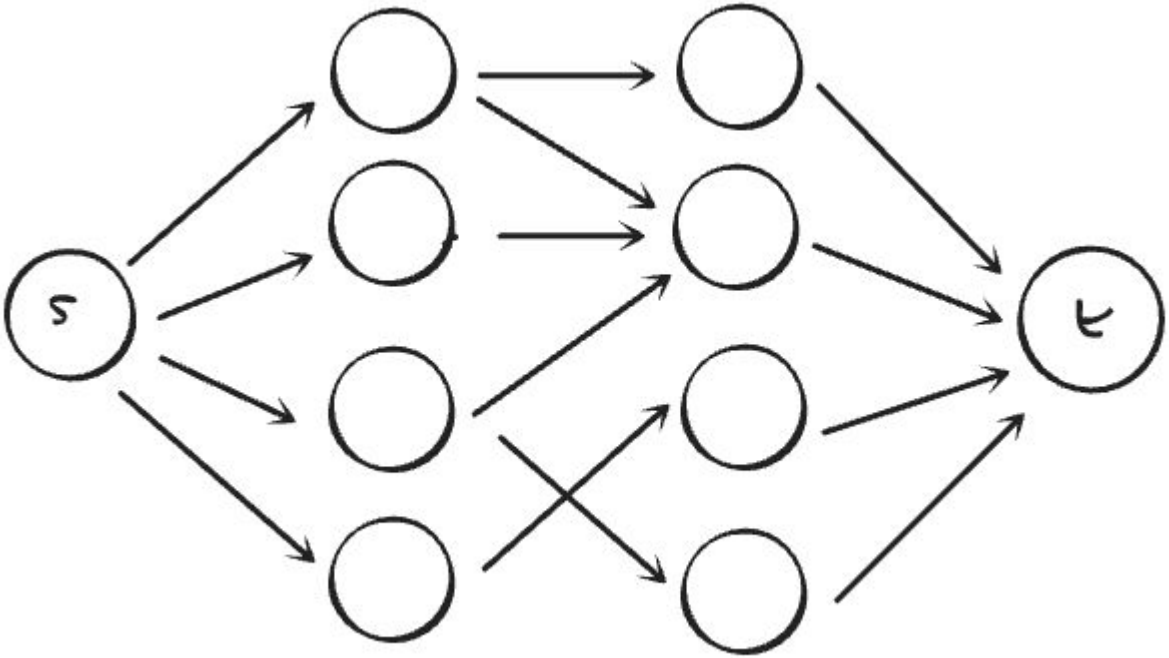


Question 2

edge

capacity

= 1



Question 2

Why does it work?

Question 2

Why does it work?

Suppose that the result is not a valid matching. Then one of the vertices must have >= 2 edges chosen at the end. This means that the flow through that vertex is >= 2. If the vertex is on the left side, the incoming edge is 1, if it's on the right side the outgoing edge is 1. Hence, the maximum flow through that vertex can only be at most 1. Contradiction.

Hence, the final flow graph must be a valid matching.

Question 2

Why does it work?

Suppose that the result is not a MCBM. Let the max flow produced be f^*. This means that the actual MCBM OPT > f^*. Notice that a valid MCBM is also a valid flow assignment. Hence, f^* must be >= OPT since OPT is a valid flow assignment.

Hence, f^* >= OPT > f^*. Contradiction.

Question 2

Why does it work?

Suppose that the result is not a MCBM. Let the max flow produced be f*. This means that the actual MCBM OPT > f*. Notice that a valid MCBM is also a valid flow assignment. Hence, f* must be >= OPT since OPT is a valid flow assignment.

Hence, f* >= OPT > f*. Contradiction.

Running time O(m^2) using FF, m is the number of edges

Q3

Well Known Problems

Question 3

Assume that you have an **(un)directed** graph G = (V, E) with a **source vertex s** and a **target vertex t** (the graph is **unweighted**). Give an algorithm that finds **the maximum number of edge-disjoint paths from s to t.** Two paths P1 and P2 are called edge-disjoint if **they do not share any edges**, but they may share a vertex. Is this a Max Flow problem? What is the running time of your algorithm?

Question 3

Just run Max Flow algo immediately on the input graph, give every edge a capacity of 1.

Question 3

Just run Max Flow algo immediately on the input graph, give every edge a capacity of 1.

O(m^2) using Ford Fulkerson

Question 3

For directed graph, should be straight forward.

For undirected graph, just split each edge into two bidirectional edges and run max flow.

Question 3

(Qn by one of the student) Hmm, for undirected graph in this case, wouldn't we possibly use both directional edge (of the same edge) twice in the max flow? How is that a valid solution for edge disjoint paths?

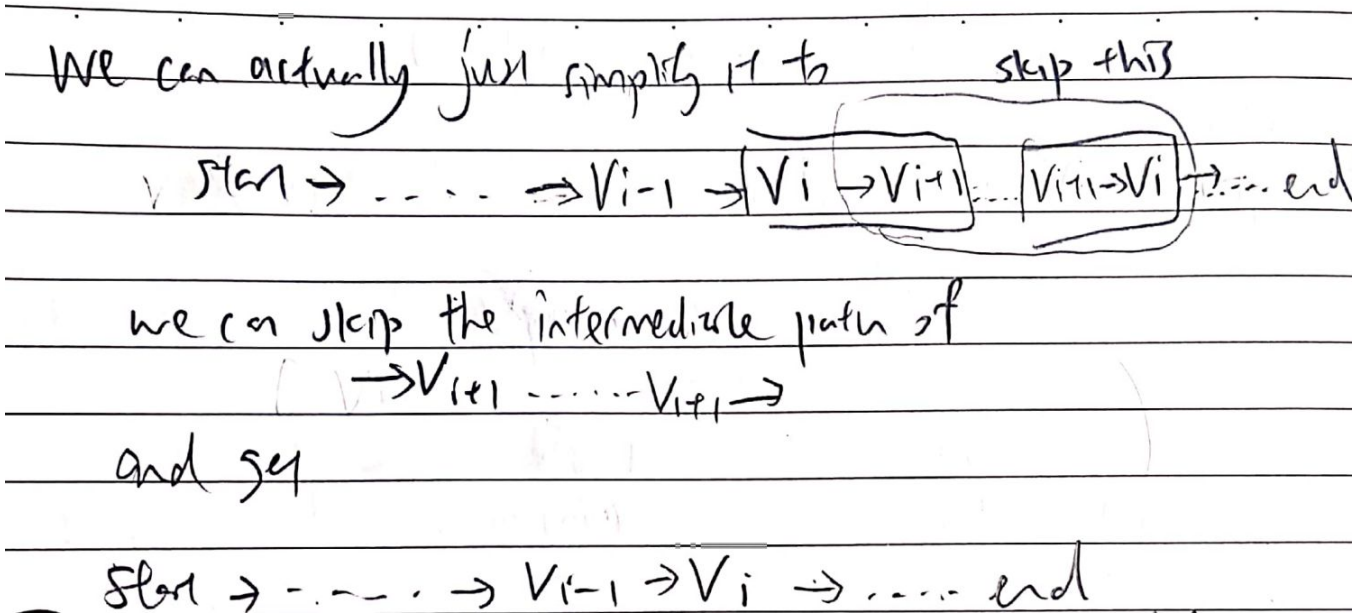
(Ans) Actually, if you try to work it out, you will realise that for any maxflow solution that uses same edge twice (once forward direction, once backward direction), you can actually construct another maxflow solution that does not use these edge at all.

Quick intuition is that there exists another flow of the same maxflow value where these flows are cancelled out by each other.

(Alternative way of seeing this will be to convert any max flow solution to a valid edge disjoint paths solution. **Optional** details in next few slides)

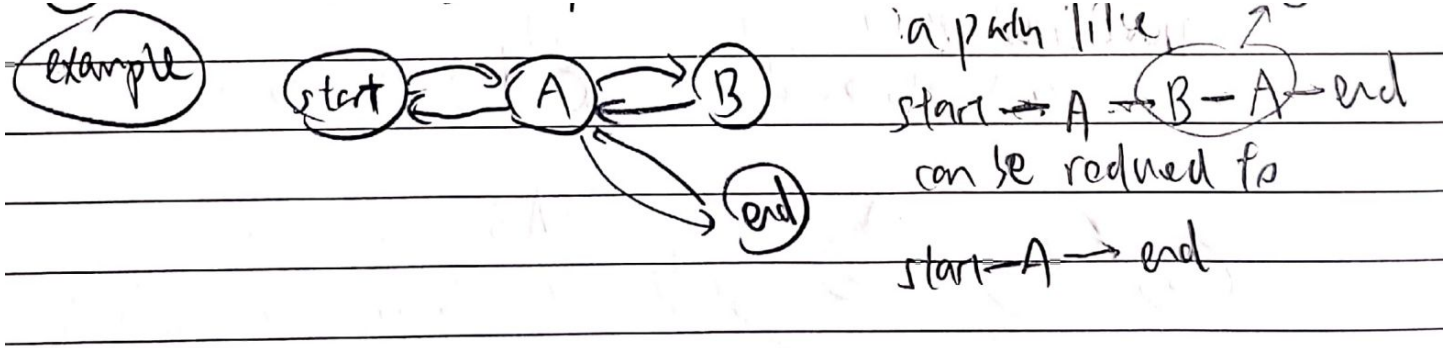
Question 3 (Optional)

Case 1: Both direction is being used in one single path.



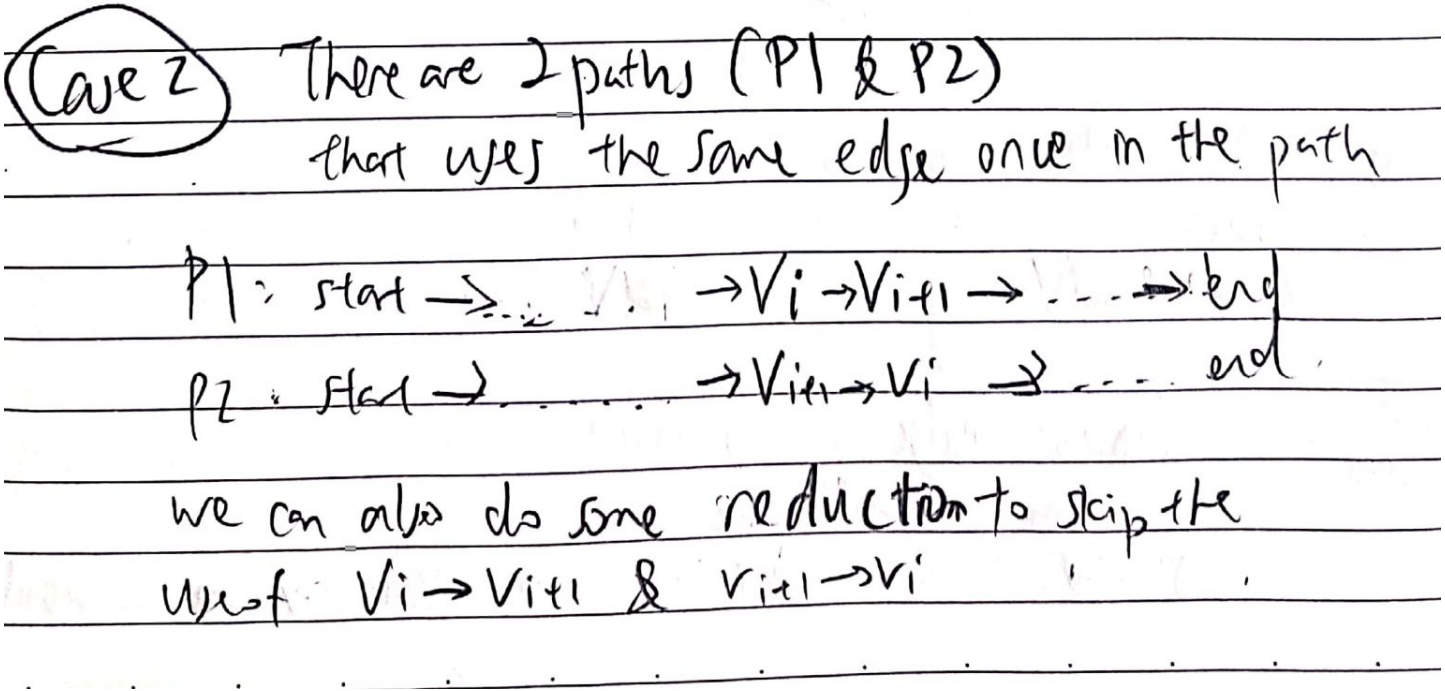
Question 3 (Optional)

Case 1 example.



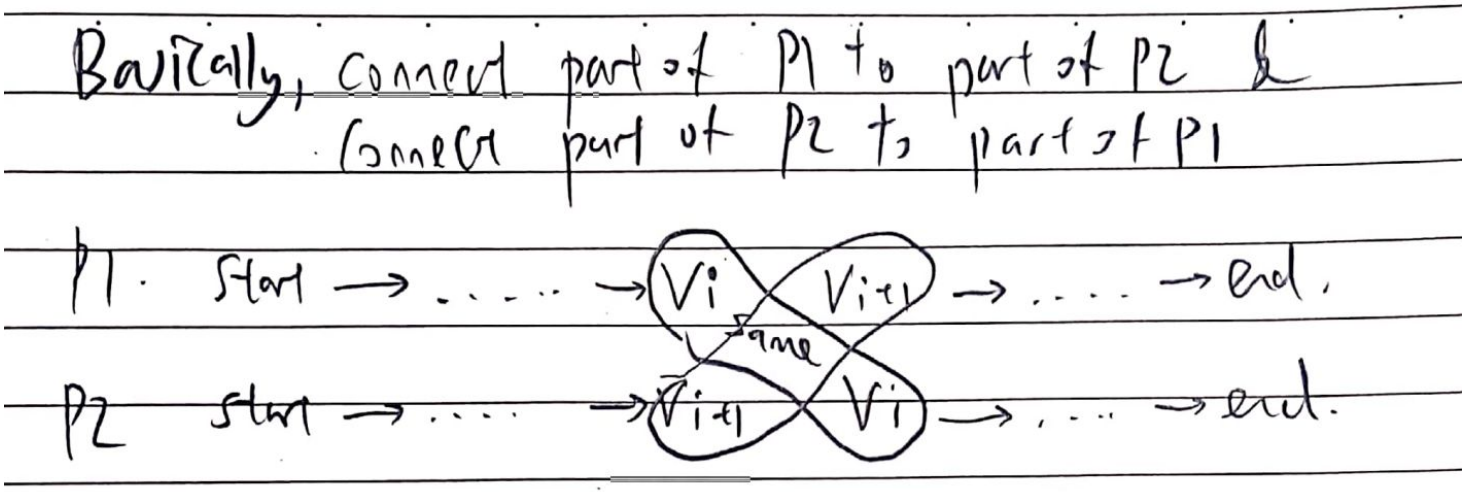
Question 3 (Optional)

Case 2: Both direction is being used in two separate path.



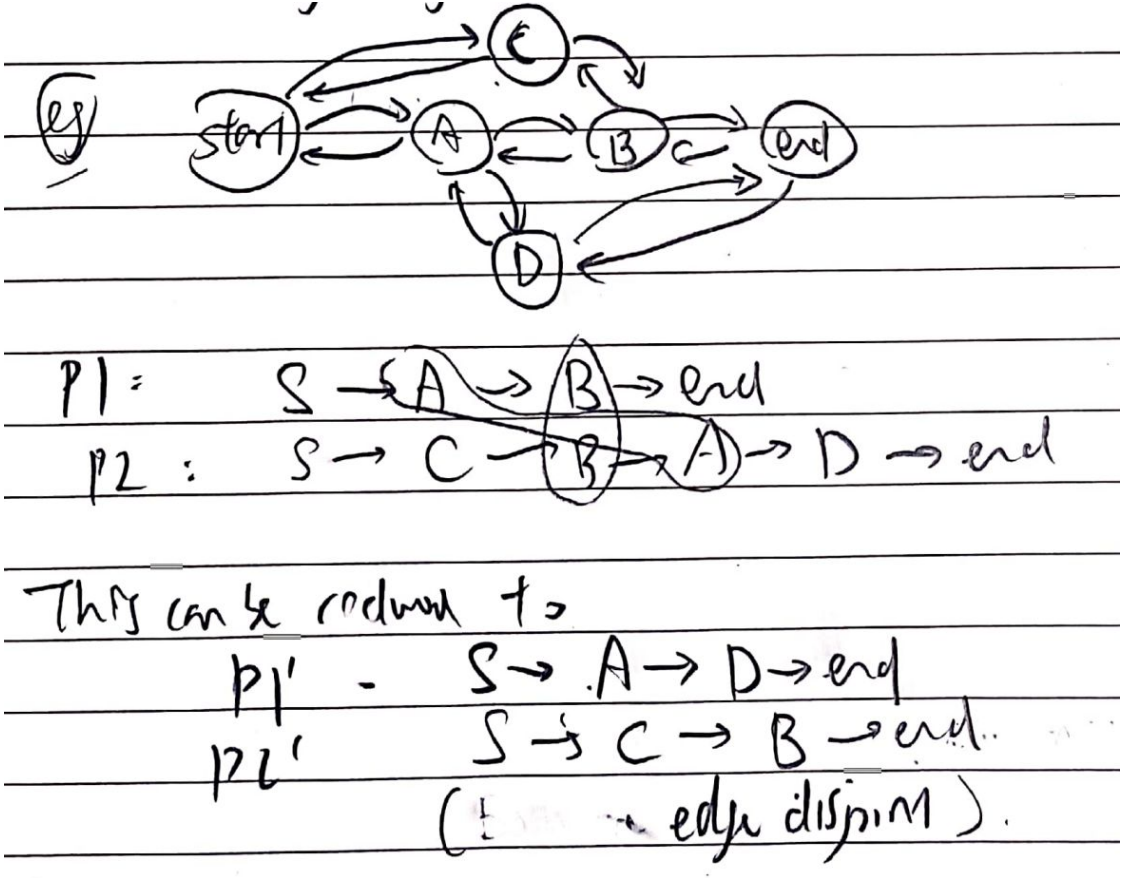
Question 3 (Optional)

Case 2: Both direction is being used in two separate path.



Question 3 (Optional)

Case 2 example.



Question 3

What if you want to find the **maximum number of vertex-disjoint paths from s to t instead**? Two paths are called vertex-disjoint if **they do not share any vertex**. Is this (still) a Max Flow problem?

Question 3

Yes, max flow can still solve this!

Question 3

Yes, max flow can still solve this!

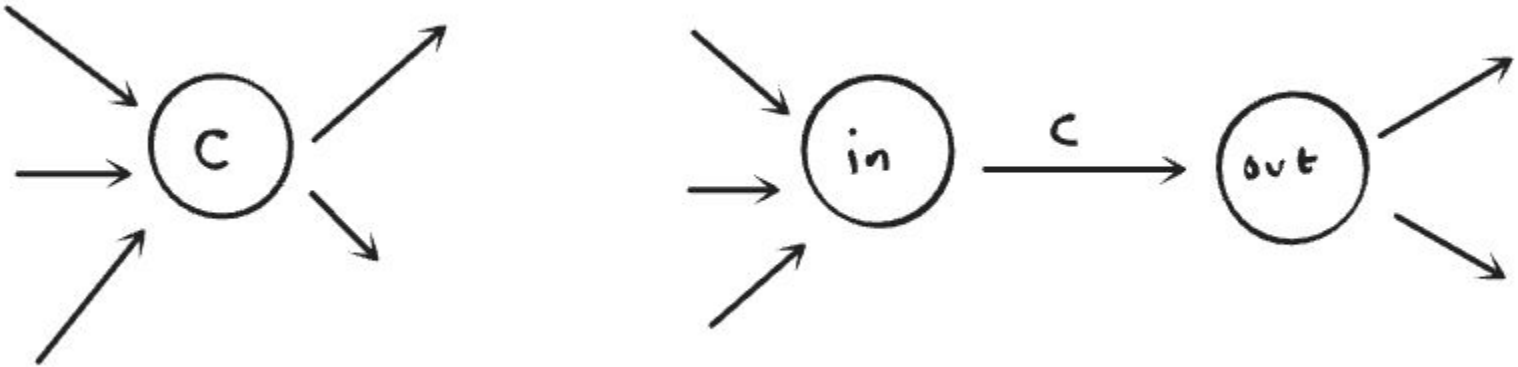
Modify the previous solution a bit, now all edges have capacity 1 and **all vertices have capacity 1 (vertex capacity).**

Question 3

How to implement vertex capacity?

Question 3

How to implement vertex capacity?



Question 3

O((m + n) ^ 2) using FF (n is the number of vertices)

Q4

Max Flow Modeling

Question 4

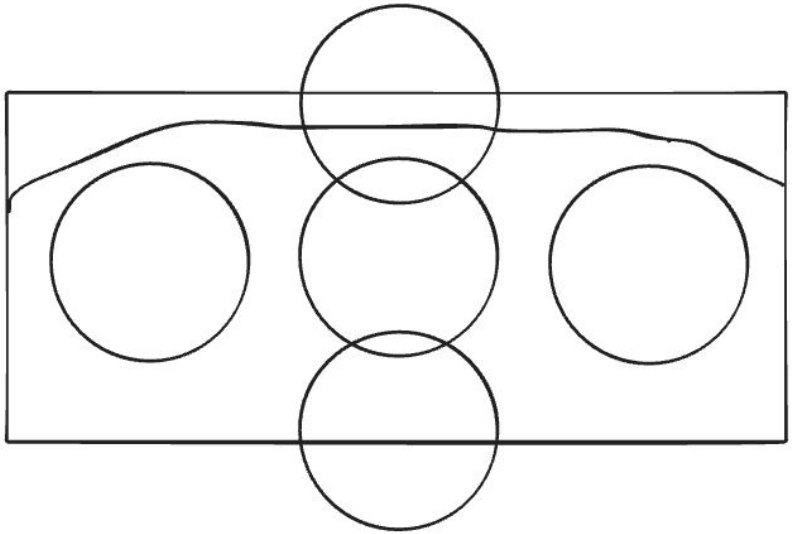
Please read https://onlinejudge.org/external/117/11757.pdf (that person *just returned last year and left this year again*) and try to **reduce this problem into a max flow problem**, solve it using O(n^2 * m) **Dinic's algorithm** (assuming that you have such implementation ready), and analyze its time complexity

Question 4

The trial will take place on a **rectangular shaped field of length L meters & width W meters**. There are **N robot defenders placed on the field**. The defenders **do not change their positions but if a winger's distance from a defender is not more than d meters, it will automatically tackle him**. A robot defender **may tackle at most once**. **On the beginning** of the trial, a winger stands **on the left edge of the field** (across the length) with a soccer ball. Now, his task is to **avoid the obstructions of the robot defenders and reach the rightmost edge** of the field with the ball. Please tell him the **minimum number of tackles he must face** in order to reach the opposite end. A player **must not go outside the field** or he will be disqualified.

Question 4

Intuition: suppose you are running across from left to right. You want to remove the minimum defenders such that the top of the field is disconnected from the bottom. Min Cut modelling might be a good idea.



Question 4 (First try)

Source = top of the field

Sink = bottom of the field

Vertices = Robots

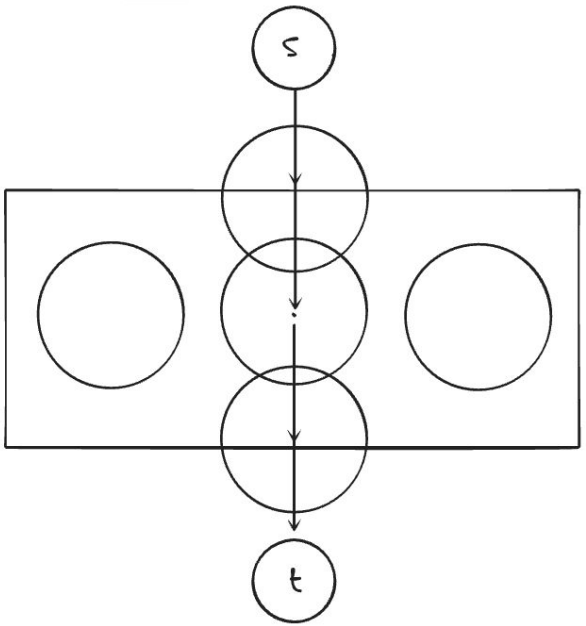
Edges (directed from top to bottom):

- Connect source to all robots with distance <= d, capacity = 1
- Connect sink to all robots with distance <= d, capacity = 1
- For every pair of robots, give edge capacity of 1 iff their euclidean distance is <= 2d (indicating if we cross between those 2 robots, we must face at least 1).
- Give all robots vertex capacity of 1 (can only tackle at most once)

Question 4

Finally just run max-flow to get the size of min-cut :D

Here min-cut is = 1



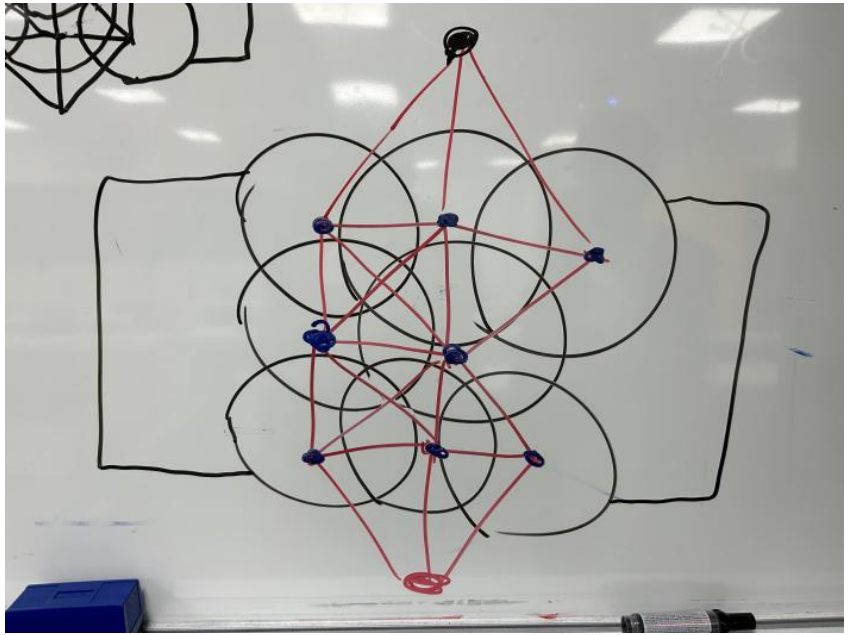
Question 4

Why do we need to care about vertex capacity?

If not set, max flow might not be the minimum number of tackles.

See counter example on the right, max flow/min-cut = 3, but the optimal answer is actually 2 (just need to go pass the middle 2 defenders).

Reason is because in the max-flow/min-cut solution, one robot defender tackled twice.



Question 4 (Second try)

Actually edge capacity might be redundant, can AC without it.

Edges (directed from top to bottom):

- Connect source to all robots with distance <= d, capacity = inf
- Connect sink to all robots with distance <= d, capacity = inf
- Give all robots vertex capacity of 1 (can only tackle at most once)
- For every pair of robots, give edge capacity of inf iff their euclidean distance is <= 2d.

Intuition: Minimizing the number of defenders to run through.

Question 4

But Dinic = O(N^2 * M) ~ 202 * 202 * 10300 > 10^8

Wait but this TLE, are we doing something wrong?

Turns out no, we just didn’t analyze tight enough, we’ll see later in Q5 :D

Q5

Output Centric Analysis

Question 5

Can we write the runtime of Ford-Fulkerson algorithm as O(m* F) where F is the eventual max flow value of the input graph? This is an output-sensitive analysis, whereby the runtime speed of the algorithm depends on the size of the output.

Question 5

Yes.

A simple argument is that each augmenting path must at least send 1 flow, so you can find at most F augmenting path. Each augmenting path will cost O(m) to find.

Hence O(m * F) time complexity.

Question 5

Now revisit problem in Q4 above and consider this `**Competitive Programmer**' analysis that is not frequently found in Computer Science textbooks about max flow algorithms. Assuming that we are using the theoretically faster O(n^2 * m) Dinic's algorithm, can we analyze its time complexity as:

$$O(min(\sum_{u:(s,u)\in E} c_{su}, \sum_{u:(u,t)\in E} c_{ut}, n^2) \times m).$$

Explain what are we trying to do here?

Note: s is source vertex and t is sink vertex

Question 5

The max flow is upper bounded by the total capacities of outgoing edges from source and the total capacities of incoming edges to sink.

Hence, the number of required iterations might not be large

Question 5

The max flow is upper bounded by the total capacities of outgoing edges from source and the total capacities of incoming edges to sink.

Hence, the number of required iterations might not be large

In Q4 we have the number of edges from source is <= N.

Hence it's actually O(N * M) ~ 100 x 10300 < 10^8, enough to pass

End of Tutorial

Any last questions?

Thank you!

See ya next 2 weeks!
ATB for midterms :D

Tutorial 5

CS4234 AY2023/2024 Semester 1

Welcome!

Q1

The Programmers

Question 1

The Programmers is a popular reality show focusing on programming contest. Each year many people would enter the contest, and compete to be the next top programmer. Due to its popularity, several people would like to enter the show. The Programmers organizes local contests around Thailand to find the great talents.

Question 1

Each local contest site can only handle C contestants. However, due to exhausted problem setters, **there are only S local-contest sites, numbered from 1 to S.** Note that all local contests are organized at the same time. Hence, one contestant can only participate in at most one of the local contest.

The local contest sites are located all over the country. For some contestants, it is not possible to go to the sites that are very far from their hometown. To facilitate the contestants, the organizer asks **each contestant to list local contest sites that they can join.**

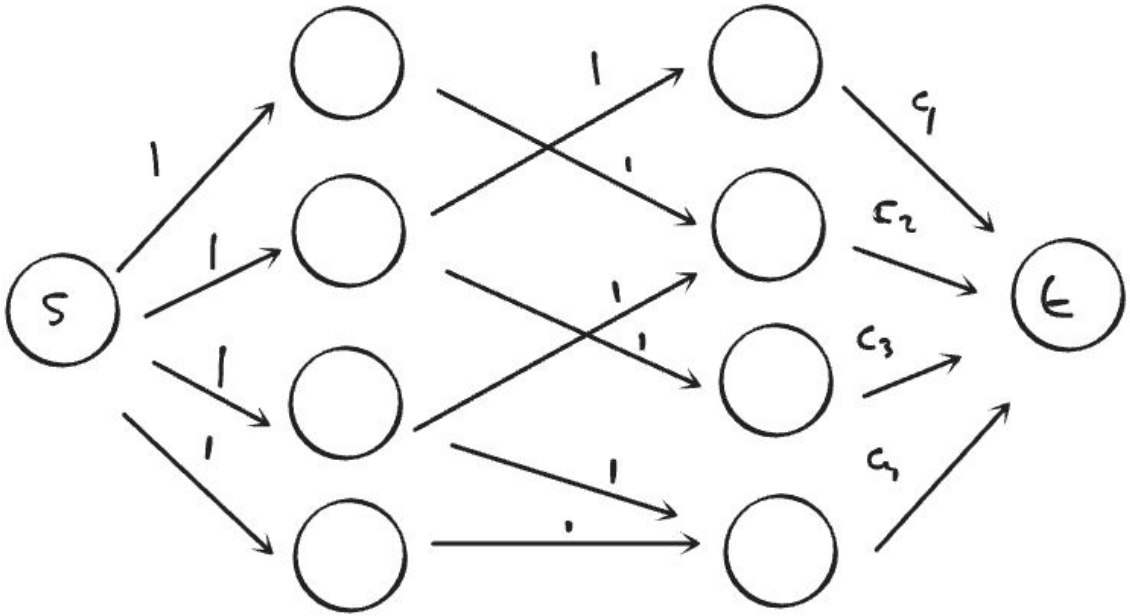
Some contestant might not be able to compete because these constraints. Your task is to **calculate the maximum overall number of contestants that can participate in the local contests without breaking these constraints.**

Question 1

How to model this as a Max-Flow problem?

Question 1

Left = Participant, Right = Contest location

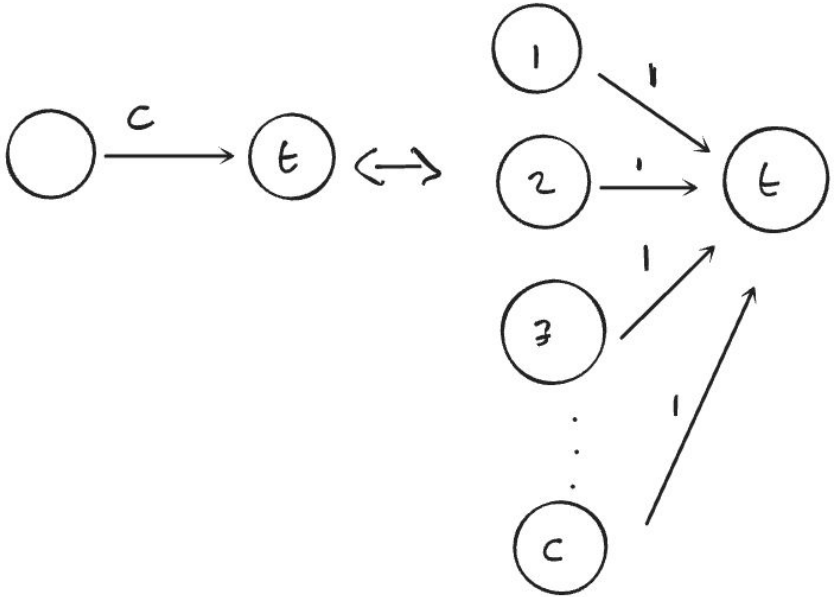


Question 1

Can we model this as a pure MCBM problem?

Question 1

Yes, key idea is that to transform each contest site into multiple vertices.



Question 1

Time complexity for Dinic is O(maximum possible flow * E) = O(min(P, S) * PS)

Q2

Greedy-able MCBM

Question 2

Professor Zac is trying to finish a collection of tasks during the first week at the start of the term. He knows precisely how long each task will take, down to the millisecond. Unfortunately, it is also Frosh Week. Zac's office window has a clear view of the stage where loud music is played. He cannot focus on any task when music is blaring.

The event organizers are also very precise. **They supply Zac with intervals of time when music will not be playing. These intervals are specified by their start and end times down to the millisecond.**

Question 2

Each task that Zac completes must be completed in one quiet interval. He cannot pause working on a task when music plays (he loses his train of thought).
I**nterestingly, the lengths of the tasks and quiet intervals are such that it is impossible to finish more than one task per quiet interval!**

Given a list of times t_i (in milliseconds) that each task will take and a list of times l_j (in milliseconds) specifying the lengths of the intervals when no music is being played, what is the maximum number of tasks that Zac can complete?

Question 2

Is there any greedy algorithm to solve this?

Question 2

Is there any greedy algorithm to solve this?

Sort the tasks and quiet intervals in increasing order of length.

Match the shortest task with the shortest interval is >= length of the shortest task.

We prove that it works using exchange argument.

Question 2

Consider some optimal solution OPT. Consider the shortest task matched in OPT, let the interval be T and it's matched with a quiet interval L.

If L is not the shortest interval >= T, then there exist some L' < L and L' >= T. We can choose to match T with L' instead of L and it doesn't change the optimality (number of tasks done is still |OPT|).

Clearly there is an optimal substructure

OPT(t1, t2, …, tn) = OPT(t2, t3, …, tn) + 1

Hence, our greedy algorithm works.

Question 2

What if we use MCBM algo to solve instead? Can we do this?

Question 2

What if we use MCBM algo to solve instead? Can we do this?

We can definitely model this as a MCBM problem!

Left set = Tasks, Right set = Quiet intervals

There’s an edge between task T and quiet interval L iff length(T) <= length(L).

Question 2

What if we use MCBM algo to solve instead? Can we do this?

We can definitely model this as a MCBM problem!

Left set = Tasks, Right set = Quiet intervals

There's an edge between task T and quiet interval L iff length(T) <= length(L).

However, doing MCBM algo takes significantly longer O(n^2.5) (Hopcroft-Karp… which TLEs) compared to the greedy sorting O(n log n + m log m).

Q3

Konig and Construction

Question 3

In problem **https://nus.kattis.com/problems/bilateral**, you are asked to find MVC and one possible solution too... **In short, given a Bipartite Graph G = (VL, VR), E of approximately 2000 vertices and up to 10 000 edges, show how to find the MVC (the optimal solution, not just the cardinality of the optimal solution) on G by reducing those problems into MCBM** (you can then use a MCBM-specific algorithm or further reduce them into Max Flow problems and use a Max Flow algorithm).

Question 3

TL;DR of the Kattis problem:

- A company has two offices, Stockholm and London
- Stockholm employee ID are [1000..1999], London are [2000..2999]
- A team consist of two employees. Each team works on different projects. Each team has to cross offices (no two employees in same team have same office)
- Select minimum number of employees such that every projects have at least one representative.
- You have a friend with ID 1009, if possible include him in the selection (but must still have minimum number of employees).

Question 3

Which NP-Hard problem is this? Is there any special property we can exploit?

Question 3

Which NP-Hard problem is this? Is there any special property we can exploit?

It is a MVC problem. Model the requirements as a bipartite graph:

- Left set = Stockholm employees
- Right set = London employees
- Edges = Teams, guaranteed to cross from left to right
- Need to cover all edges == MVC

How to find MVC in a bipartite graph?

Question 3

Which NP-Hard problem is this? Is there any special property we can exploit?

It is a MVC problem. Model the requirements as a bipartite graph:

- Left set = Stockholm employees
- Right set = London employees
- Edges = Teams, guaranteed to cross from left to right
- Need to cover all edges == MVC

How to find MVC in a bipartite graph?

We can find MVC by finding MCBM due to Konig's theorem. Let the size of MCBM = M^*

Question 3

How include your friend (ID 1009)?

Create a new instance where we force to include 1009 in the vertex cover, that is delete all edges containing 1009. Run MCBM again, let the size be M'. If M' + 1 == M* then we can definitely include this friend, otherwise no.

Question 3

How include your friend (ID 1009)?

Create a new instance where we force to include 1009 in the vertex cover, that is delete all edges containing 1009. Run MCBM again, let the size be M'. If M' + 1 == M* then we can definitely include this friend, otherwise no.

But the question requires us to output the set (not just the size) :(

Luckily Konig's theorem also provides an explicit construction of the set!

Question 3

Konig's construction:

Let

- U = unmatched vertices on the left set
- Z = vertices that are either in U or are connected to U by alternating paths

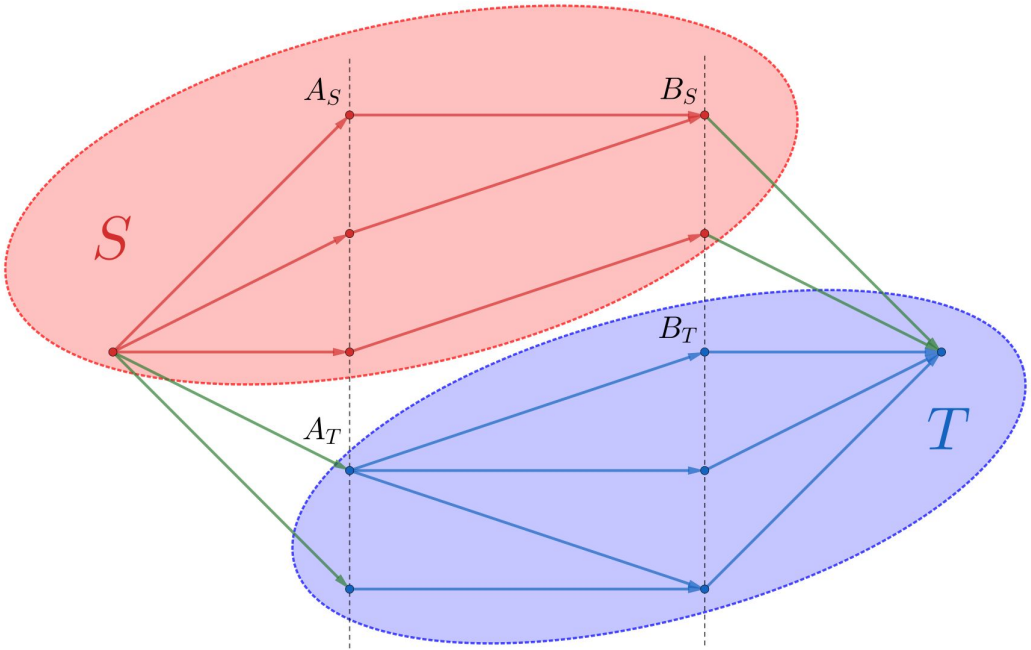
Then MVC = $\{L \setminus Z\} \cup \{R \cap Z\}$.

Can view demo on Visualgo: https://visualgo.net/en/mvc

Question 3

Alternatively, can also construct from the Min-Cut:

MVC = At U Bs



Question 3

How to find MIS instead of MVC?

Question 3

How to find MIS instead of MVC?

Size of MIS = |V| - |MVC| as they are complementary problems.

The construction is just to take the vertices not in MVC.

Question 3

What if the MVC/MIS problems asked are the weighted variants?

Question 3

What if the MVC/MIS problems asked are the weighted variants?

We must use max-flow here as the vertex weights == capacities.

Source connects to left set with edge capacity = weight of the left set vertices. Similarly sink connects to right set. We set the weights of the edges between left and right set to be infinity.

|MWVC| = Max flow

Use Min-Cut to construct MVC

Q4

Min Path Cover

Question 4

TL;DR of question:

- You are given a list of booked taxi rides in the form (start time, start_x, start_y, destination_x, destination_y)
- The time needed to travel from (a, b) to (c, d) is |a - c| + |b - d|
- A taxi may carry out a booked ride if it is its first ride of the day, or if it can get to the source address of the new ride from its latest, at least one minute before the new ride's scheduled departure.
- Compute the minimum number of taxi you need.

Question 4

What NP-Hard problem is this and what is the modelling?

It is a Min Path Cover problem, that is we are asking the minimum number of path needed to cover the whole graph. Model the graph as follows:

- Each booking is the vertex.
- Create an edge between Booking A and Booking B if the taxi can serve B after finishing A
- Notice that each taxi is a path in the graph. We want the minimum number of path such that all vertices are covered.

We see that the graph is Acyclic. We can solve Min Path Cover in DAG by modelling it as a MCBM problem:

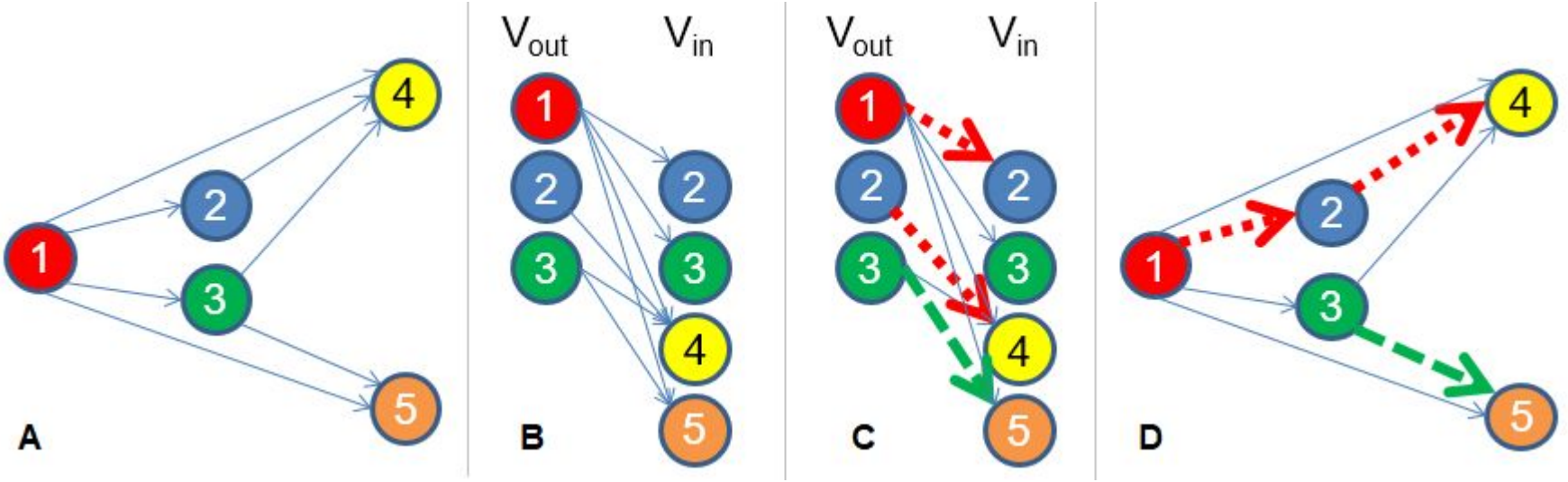
Question 4

- For each vertex (booking) split it into two vertices. One for booking start, one for booking end. Booking end is the left set, Booking start is the right set
- Create an edge between 2 bookings (B1_end, B2_start) iff B1_end <= B2_start - 1

Left set = Booking end, Right set = Booking start

The size of Min Path Cover = number of bookings - |MCBM|

Question 4



Question 4

The size of Min Path Cover = number of bookings - |MCBM|

Here's a short argument why the above works:

Let n = number of bookings. We can definitely serve all rides using n taxis. Now notice that each matched edge indicates that a taxi can be used to serve another ride once it has finished it's previous ride. Hence, we can remove 1 taxi for every matched edge.

One of your PS4 problem is very similar to this :)

PS4A

Intuition:

- Minimum set cover where each set can cover a pair of adjacent ‘*’’s
- The graph is a grid. What does it mean? Bipartite!
- MCBM? MVC? How can we exploit that property?

Optional

Proving Berge's Theorem

Optional

Berge's theorem states that a matching M in a graph G is maximum (contains the largest possible number of edges) if and only if there is no augmenting path (a path that starts and ends on free (unmatched) vertices, and alternates between edges in and not in the matching) with M

Prove this theorem

Optional

(=>)

First let's prove the forward direction, Suppose M in G is maximum and there still is an augmenting path.

Consider that augmenting path, we can flip the edges along that augmenting path and get another matching M'. But we have |M'| = |M| + 1 > |M|. Contradiction with M maximum.

Optional

(<=)

Now let’s prove the backwards direction. Suppose there are no more augmenting paths, but M is not maximum.

Consider the maximum matching M’ != M. Consider the graph H where:

- Vertices of H = Vertices of G
- Edges of H = edges that are exactly in one of M or M’

Notice that H only have vertices of degrees at most 2 since it is build from a matching.

Optional

(<=)

Hence, graph H consists of

- Isolated vertices
- Paths
- Cycles

Color the edges that come from M Red and edges that come from M’ Blue. Notice that for every vertex with degree = 2, we cannot have both edges be of the same color, since it would imply that both of them belongs to the same matching, which contradicts the fact that in a matching each vertex can only have 1 outgoing edge.

Optional

(<=)

Hence, every paths and cycles in H must have alternating colors between consecutive edges => there are no odd cycles in H.

Since |M'| > |M|, there must be more blue edges than red edges. The only way this is possible is that there is one path of odd length in H which starts and ends with blue edges (because each cycle is even => contributes the same number of blue and red edges).

But that path of odd length is an augmenting path in M (blue edges = not taken in M, red edges = taken in M). We have an augmenting path. Contradiction.

Tutorial Questions Done

Any questions?

PS3 Debrief

Did you manage to solve all?

A: Kattis - sumsets

Main ideas, O(n^2):

- Iterate through all O(n^2) pairs, push the pair to the list of pairs that has the same sum
  - You can also just save the latest pair per sum if you iterate correctly
- Given d, iterate through all possible c, check if a and b of the remaining sum are distinct from c and d
- If yes, possible. Otherwise, impossible

B: Kattis - indoorienteering

Main ideas:

- Naive std::next_permutation gives TLE
- Since TSP paths are cyclic symmetric, we can fix a starting point
- Meet-in-middle, fix a middle point, partition the rest into two
- Find all possible distances for first half and second half → 2SUM

C: Kattis - water

Main ideas:

- Use normal dinic's algorithm (provided by CP4)
- Track the current flow
- Whenever additional capacity is added, add an edge of that capacity
- Recompute flow from source to sink with the current residual graph
- Add to the current flow

D: Kattis - neutral ground

- Model as min-cut between A (source) and B (sink), or vice versa
- Use super-source and super-sink to connect A and B territories
- Assign edge of infinite capacities between adjacent grids
- Assign vertex capacities as the number of soldiers needed to guard

End of Tutorial

Any last questions?

Thank you!

See ya next week!