CS 218: Assignment 3

Ankit Kumar Misra	Devansh Jain	Harshit Varma	Richeek Das
190050020	190100044	190100055	190260036

April 16, 2021

Contents

Question 1

There are n jobs and m persons.

(a)

Problem: You are given a list of pairs (J_i, P_j) which indicate that person P_j is willing to do job J_i. You have to determine whether it is possible to assign all n jobs to persons so that each person is assigned at most one job, and the person is willing to do the assigned job. Show how this problem can be solved using max-flow.

Solution: Let L denote the given list of pairs of the form (J_i, P_j), which mean person P_j is willing to do job J_i. We can reduce the above problem to a max-flow computation problem over a graph G. We define $G = (V, E)$ as follows:

- $V = \{s\} \cup \{u_i \mid 1 \leq i \leq n\} \cup \{v_j \mid 1 \leq j \leq m\} \cup \{t\}$
- $E = E_{\text{source}} \cup E_{\text{mid}} \cup E_{\text{sink}}$
  - $E_{\text{source}} = \{(s, u_i) \mid 1 \leq i \leq n\}$
  - $E_{\text{mid}} = \{(u_i, v_j) \mid 1 \leq i \leq n,\ 1 \leq j \leq m,\ (J_i, P_j) \in L\}$
  - $E_{\text{sink}} = \{(v_j, t) \mid 1 \leq j \leq m\}$
- $\forall e \in E, c(e) = 1$

Intuitively, in graph G, each vertex u_i corresponds to the job J_i and each vertex v_j corresponds to the person P_j. Each job is connected to all the persons willing to do it. Further, each job is connected to the source and each person is connected to the sink. All edges are assigned unit capacity.

We claim that an assignment of all n jobs to willing persons (with each person being assigned at most one job) is possible if and only if the maximum possible value of flow in the graph G above is equal to n.

Proof. We prove the above claim in both directions:

($\Rightarrow$) Suppose there is an assignment $\mathcal{A}$ of all n jobs to distinct willing persons (distinct because every person can have at most one job). We show that this implies the existence of a flow f in the graph G, such that $|f| = n$. Firstly, for all edges $(s, u_i) \in E_{\text{source}}$, we assign $f(s, u_i) = 1$. Next, for all $i \in \{1, 2, \ldots, n\}$, if $\mathcal{A}$ assigns job J_i to person P_j, we assign $f(u_i, v_j) = 1$ and $f(v_j, t) = 1$. Finally, to all remaining edges e, we assign $f(e) = 0$.

The flow f as defined above obeys the flow constraints. Every vertex of type u_i has a flow of 1 into it from source s, and a flow of 1 out of it to the vertex v_j corresponding to the person P_j who is assigned job J_i. Also, every vertex of type v_j such that P_j is assigned some job has a flow of 1 into it (from the vertex corresponding to the assigned job) and a flow of 1 out of it to t (since each person is assigned at most one job, the flow in and out of v_j has to be 1). Every vertex of type v_j corresponding to a jobless person P_j has zero flow into it and zero flow out of it.

Further, we have $|f| = n$, because there are n (number of jobs) outgoing edges from s, and f assigns unit flow to each of them. Since all outgoing edges from s are saturated, f is also the maximum possible flow in the graph G. Thus, the maximum value of flow in graph G is equal to n.

($\Leftarrow$) Suppose the graph G has a maximum flow f such that $|f| = n$. We can use f to obtain an assignment $\mathcal{A}$ of jobs to persons satisfying the given constraints.

First, we note that the flow through any edge in graph G can only be either 0 or 1, as the capacity of each edge is 1 and the flow can take only integer values.

Let $E^* = \{e \mid e \in E_{\text{mid}}, f(e) = 1\}$. For every edge $e = (u_i, v_j) \in E^*$, let $\mathcal{A}$ assign job J_i to person P_j.

- Any job (or vertex u_i) is assigned to at most one person (or vertex v_j).
  Suppose some job J_i is assigned to k persons. This means there are k vertices of type v_j such that $f(u_i, v_j) = 1$. By flow constraints, $f^{\leftarrow}(u_i) = f^{\rightarrow}(u_i) = k$. But the total capacity of edges going into u_i is only 1, from just one edge from the source itself. Thus, $k \leq 1$.
- Any person (or vertex v_j) is assigned at most one job (or vertex u_i).
  Suppose some person P_j is assigned k jobs. This means there are k vertices of type u_i such that $f(u_i, v_j) = 1$. By flow constraints, $f^{\rightarrow}(v_j) = f^{\leftarrow}(v_j) = k$. But the total capacity of edges going out of v_j is only 1, with just one edge to the sink itself. Thus, $k \leq 1$.
- There are exactly n edges in E^*.
  Let $S = \{s\} \cup \{u_i \mid 1 \leq i \leq n\}$ and $T = \{t\} \cup \{v_j \mid 1 \leq j \leq m\}$. Consider the flow across the cut given by (S, T). Every $e \in E^*$ adds a flow of 1, and other edges have zero flow. Thus, $f^{\rightarrow}(S) - f^{\leftarrow}(S) = |E^*|$. However, we also know $f^{\rightarrow}(S) - f^{\leftarrow}(S) = |f| = n$. Thus, $|E^*| = n$.

We can combine the above facts to conclude that $\mathcal{A}$ assigns all n jobs to distinct willing persons.

□

Note that the above problem was essentially a maximum matching problem on a bipartite graph, to check whether a matching of size n is possible, with all jobs forming one set of vertices and all persons forming the other, and the edges linking jobs to willing persons.

(b)

Problem: Modify the algorithm when each person P_i specifies an upper bound d_i on the number of jobs that he/she is willing to do.

Solution: Once again, we reduce the above problem to a max-flow computation task over a graph G. Let L denote the given list of pairs of the form (J_i, P_j), which mean person P_j is willing to do job J_i. The graph here varies only slightly (only in terms of capacities of edges) from the graph in part (a). We define $G = (V, E)$ as follows:

- $V = \{s\} \cup \{u_i \mid 1 \leq i \leq n\} \cup \{v_j \mid 1 \leq j \leq m\} \cup \{t\}$
- $E = E_{\text{source}} \cup E_{\text{mid}} \cup E_{\text{sink}}$
  - $E_{\text{source}} = \{(s, u_i) \mid 1 \leq i \leq n\}$
  - $E_{\text{mid}} = \{(u_i, v_j) \mid 1 \leq i \leq n,\ 1 \leq j \leq m,\ (J_i, P_j) \in L\}$
  - $E_{\text{sink}} = \{(v_j, t) \mid 1 \leq j \leq m\}$
- $\forall e \in E, c(e) = \begin{cases} d_j & \text{if } e = (v_j, t) \in E_{\text{sink}} \\ 1 & \text{otherwise} \end{cases}$

Intuitively, graph G is very similar to the graph from part (a). Once again, each vertex u_i corresponds to the job J_i and each vertex v_j corresponds to the person P_j. Each job is connected to all the persons willing to do it. Further, each job is connected to the source and each person is connected to the sink. Edges are assigned capacities as already written above.

We claim that an assignment of all n jobs to willing persons (with each person P_j being assigned at most d_j jobs) is possible if and only if the maximum possible value of flow in the graph G above is equal to n.

Proof. We prove the above claim in both directions:

($\Rightarrow$) Suppose there is an assignment $\mathcal{A}$ of all n jobs to willing persons (person P_i having at most d_i jobs). We show that this implies the existence of a flow f in the graph G, such that $|f| = n$. Firstly, for all edges $(s, u_i) \in E_{\text{source}}$, we assign $f(s, u_i) = 1$. Now, for all $j \in \{1, 2, \ldots, m\}$, we initialize $f(v_j, t)$ to zero. Next, for all $i \in \{1, 2, \ldots, n\}$, if $\mathcal{A}$ assigns job J_i to person P_j, we assign $f(u_i, v_j) = 1$ and increment $f(v_j, t)$ by 1. Finally, to all the remaining edges e in E_{mid}, we assign $f(e) = 0$.

The flow f as defined above obeys the flow constraints. Every vertex of type u_i has a flow of 1 into it from source s, and a flow of 1 out of it to the vertex v_j corresponding to the person P_j who is assigned job J_i. Also, every vertex of type v_j such that P_j is assigned some job(s) has equal flows into it and out of it, since both values were incremented together in the construction of f. Every vertex of type v_j corresponding to a jobless person P_j has zero flow into it and zero flow out of it.

Further, we have $|f| = n$, because there are n (number of jobs) outgoing edges from s, and f assigns unit flow to each of them. Since all outgoing edges from s are saturated, f is also the maximum possible flow in the graph G. Thus, the maximum value of flow in graph G is equal to n.

($\Leftarrow$) Suppose the graph G has a maximum flow f such that $|f| = n$. We can use f to obtain an assignment $\mathcal{A}$ of jobs to persons satisfying the given constraints.

First, we note that the flow through any edge in $E_{\text{source}} \cup E_{\text{mid}}$ can only be either 0 or 1, as the capacity of each edge is 1 and the flow can take only integer values.

Let $E^* = \{e \mid e \in E_{\text{mid}}, f(e) = 1\}$. For every edge $e = (u_i, v_j) \in E^*$, let $\mathcal{A}$ assign job J_i to person P_j.

- Any job (or vertex u_i) is assigned to at most one person (or vertex v_j).
  Suppose some job J_i is assigned to k persons. This means there are k vertices of type v_j such that $f(u_i, v_j) = 1$. By flow constraints, $f^{\leftarrow}(u_i) = f^{\rightarrow}(u_i) = k$. But the total capacity of edges going into u_i is only 1, from just one edge from the source itself. Thus, $k \leq 1$.
- Any person P_j (or vertex v_j) is assigned at most d_j jobs (or vertices u_i).
  Suppose some person P_j is assigned k jobs. This means there are k vertices of type u_i such that $f(u_i, v_j) = 1$. By flow constraints, $f^{\rightarrow}(v_j) = f^{\leftarrow}(v_j) = k$. But the total capacity of edges going out of v_j is d_j, with just one edge to the sink itself. Thus, $k \leq d_j$.
- There are exactly n edges in E^*.
  Let $S = \{s\} \cup \{u_i \mid 1 \leq i \leq n\}$ and $T = \{t\} \cup \{v_j \mid 1 \leq j \leq m\}$. Consider the flow across the cut given by (S, T). Every $e \in E^*$ adds a flow of 1, and other edges have zero flow. Thus, $f^{\rightarrow}(S) - f^{\leftarrow}(S) = |E^*|$. However, we also know $f^{\rightarrow}(S) - f^{\leftarrow}(S) = |f| = n$. Thus, $|E^*| = n$.

We can combine the above facts to conclude that $\mathcal{A}$ assigns all n jobs to willing persons while obeying the d_i constraint for each person P_i.

□

(c)

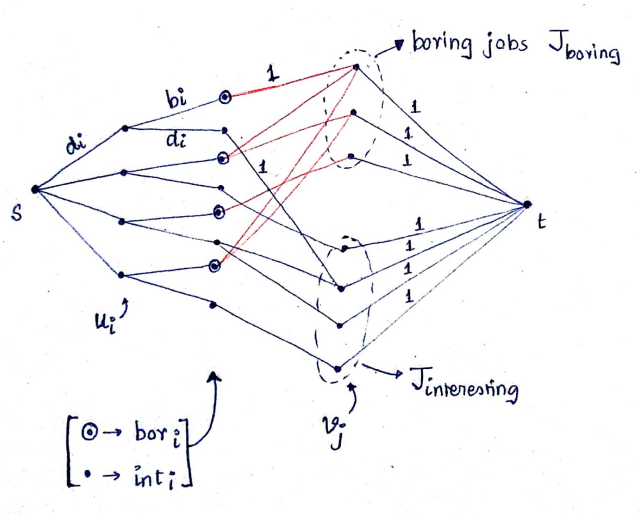
Problem: Suppose now the jobs are classified as "boring" and "interesting". Each person specifies an upper bound d_i on the total number of jobs that he/she is willing to do, and also an upper bound b_i on the number of boring jobs that he/she is willing to do. Show how max-flows can be used to find a feasible assignment of jobs, if it exists.

Solution: Notations: (J_i, P_j) are tuples which indicate that person P_i is willing to do job J_i. Let L be the list of these tuples. d_i denotes the upper bound on the total number of jobs the person is willing to do. b_i denotes the total number of boring jobs he/she is willing to do. We can model this problem as a max-flow problem over a graph G. We define a graph $G = (V, E, c)$. V represents the vertices, E represents the edges and c represents the capacity of the edges. Let's define the graph:

- $V = \{s\} \cup \{u_i | 1 \leq i \leq m\} \cup \{bor_i | 1 \leq i \leq m\} \cup \{int_i | 1 \leq i \leq m\} \cup \{v_j | 1 \leq i \leq n\} \cup \{t\}$
- $E = E_{source} \cup E_{bor} \cup E_{int} \cup E_{bor2job} \cup E_{int2job} \cup E_{sink}$
  - $E_{source} = \{(s, u_i) | 1 \leq i \leq m\}$
  - $E_{bor} = \{(u_i, bor_i) | 1 \leq i \leq m\}$
  - $E_{int} = \{(u_i, int_i) | 1 \leq i \leq m\}$
  - $E_{bor2job} = \{(bor_i, v_j) |$ $(J_j,P_i) \in L$ and $J_j \in J_{boring}\}$
  - $E_{int2job} = \{(int_i, v_j) |$ $(J_j,P_i) \in L$ and $J_j \in J_{interesting}\}$
  - $E_{sink} = \{(v_j, t) | 1 \leq j \leq n\}$
- $\forall e \in E, c(e) = \begin{cases} d_i & \text{if } e = (s, u_i) \in E_{\text{source}} \\ b_i & \text{if } e = (u_i, bor_i) \in E_{\text{bor}} \\ d_i & \text{if } e = (u_i, int_i) \in E_{\text{int}} \\ 1 & \text{if } e = (bor_i, v_j) \in E_{\text{bor2job}} \\ 1 & \text{if } e = (int_i, v_j) \in E_{\text{int2job}} \\ 1 & \text{if } e \in E_{\text{sink}} \end{cases}$

More notation explanations:

- **s** is the source.
- $\boldsymbol{u_i}$ are the vertices corresponding intuitively to the m persons.
- $\boldsymbol{bor_i}$ vertices behaves like a switch which directs the flow of of the person P_i to the boring jobs he/she is willing to do.
- $\boldsymbol{int_i}$ just like vertex $\boldsymbol{bor_i}$ directs the flow of of the person P_i to the interesting jobs he/she is willing to do.
- $\boldsymbol{v_j}$ corresponds to jobs.
- $\boldsymbol{J_{boring}}$ corresponds to the set of boring jobs.
- $\boldsymbol{J_{interesting}}$ corresponds to the set of interesting jobs.
- $\boldsymbol{t}$ is the sink.



We claim that the an assignment of all n jobs to willing persons (with each person doing atmost b_i boring jobs and d_i total jobs) is possible if and only if the maximum possible flow in the graph G above is equal to n.

Proof: We prove the claim in both directions

($\Rightarrow$) Suppose there is an assignment $\mathcal{A}$ of all n jobs to willing persons (with each person doing atmost b_i boring jobs and d_i total jobs). We show that this implies the existence of a flow in the graph G such that $|f| = n$.

The assignment will looks something like $\boldsymbol{\mathcal{A} : P \longrightarrow \mathcal{P}(J)}$, where P is the person set and $\mathcal{P}(J)$ represents the powerset of J. That is the assignment maps a person to a subset of jobs, while maintaining the **size of subset assigned** to P_i is **atmost** d_i, **boring jobs in the subset** is **atmost** b_i, the subsets are **disjoint** and $\forall$ jobs J_j assigned to P_i, $(J_j, P_i) \in L$.

In the forward direction, our precondition enables us to assume that we have a **valid assignment**.

Let's design a flow based on this assignment:

$$\forall e \in E, f(e) = \begin{cases} |S_i| & \text{if } e = (s, u_i) \in E_{\text{source}} \text{ and } \mathcal{A} : P_i \longrightarrow S_i \text{ (set of jobs)} \\ |S_i \cap J_{boring}| & \text{if } e = (u_i, bor_i) \in E_{\text{bor}} \text{ and } \mathcal{A} : P_i \longrightarrow S_i \text{ (set of jobs)} \\ |S_i \cap J_{interesting}| & \text{if } e = (u_i, int_i) \in E_{\text{int}} \text{ and } \mathcal{A} : P_i \longrightarrow S_i \text{ (set of jobs)} \\ 1 & \text{if } e = (bor_i, v_j) \in E_{\text{bor2job}} \text{ and } J_j \in \mathcal{A}[P_i] \\ 1 & \text{if } e = (int_i, v_j) \in E_{\text{int2job}} \text{ and } J_j \in \mathcal{A}[P_i] \\ 1 & \text{if } e \in E_{\text{sink}} \\ 0 & \text{otherwise} \end{cases}$$

The flow f defined above obeys the **capacity constraints** by the properties of the assignment $\mathcal{A}$. It is easy to observe this, since $|S_i| \leq d_i$ and $|S_i \cap J_{boring}| \leq b_i$. By property of intersection, $|S_i \cap J_{interesting}| \leq d_i$

It is also easy to observe that the flow f as defined obeys the **flow constraints**. Let's look vertex by vertex:

- For vertex $\boldsymbol{u_i}$, inward flow = $|S_i|$ and outward flow = $|S_i \cap J_{boring}| + |S_i \cap J_{interesting}|$. By definition, $J_{boring} \cup J_{interesting} = J$ and $J_{boring} \cap J_{interesting} = \phi$. Therefore,

$$|S_i \cap J_{boring}| + |S_i \cap J_{interesting}| = |(S_i \cap J_{boring}) \cup (S_i \cap J_{interesting})| \text{ [as they are disjoint]}$$
$$\implies |S_i \cap J| = |S_i| \text{ as } [J_{boring} \cup J_{interesting} = J]$$

  Therefore, inward flow matches with the outward flow!
- For vertex $\boldsymbol{bor_i}$, inward flow = $|S_i \cap J_{boring}|$ and outward flow = $|S_i \cap J_{boring}|$ since by construction, number of edges leaving $bor_i = |S_i \cap J_{boring}|$ each with flow 1.
- For vertex $\boldsymbol{int_i}$, inward flow = $|S_i \cap J_{interesting}|$ and outward flow = $|S_i \cap J_{interesting}|$. This also holds by construction.
- For vertex $\boldsymbol{v_j}$, outward flow = 1. We need to prove that the inward flow is exactly 1. **Arguments:**
  - A vertex v_j has inward flow if $\exists P_i | J_j \in \mathcal{A}[P_i]$. This obviously holds for every vertex since the assignment by definition maps all n jobs to persons. Therefore inward flow is atleast 1.
  - Also we know $\mathcal{A}[P_i] \cap \mathcal{A}[P_j] = \phi$, so there exists one and only one member of P to whom a job is alloted. Therefore inward flow is atmost one.

  Thus, the inward flow into every v_j is exactly 1. Flow constraint holds for this vertex as well.

Therefore we showed that we can build a valid flow from a given valid assignment! This proves the forward direction, since we have $|f| = n$, because there are n (number of jobs) outgoing edges from v_j and f assigns unit flow to each of them. Since all outgoing edges from those are saturated, f is also the maximum possible flow in the graph G. Thus, the maximum value of flow in graph G is equal to n.

($\Leftarrow$) We want to show that if we have a maxflow f of graph G and $|f| = n$, then it is possible to find an assignment $\mathcal{A}$ which maps all n jobs to willing persons (with each person doing atmost b_i boring jobs and d_i total jobs).

In this part we assume that we have a maxflow f of value n, which looks like something: $f : E \longrightarrow x$, where $x \in \mathbb{Z}$, i.e maps edges to integer flows, while obeying the capacity and flow constraints as we had defined earlier.

We build a possible assignment $\mathcal{A} : P \longrightarrow \mathcal{P}(J)$ for every person P_i from the given maxflow f by following this algorithm:

$$\mathcal{A}[P_i] = \{J_j | f(bor_i, v_j) == 1\} \cup \{J_j | f(int_i, v_j) == 1\}$$

Which means: **"Add the job J_j to the set of jobs alloted to P_i if $f(bor_i, v_j) == 1$ or $f(int_i, v_j) == 1$".**

Now we need to prove that this assignment indeed obeys the constraints of **disjointness**, provides **mapping for all** the n jobs, and maintains the **limits** set by b_i and d_i.

- The proposed assignment is definitely **disjoint** since any job cannot have two incoming flows! **Argument:** Every v_j has flow of 1 to the sink(since its a **maxflow** with $|f| = n$). So v_j cannot have multiple inward flows of value 1. That would violate the flow constraint.
- Therefore, in order to maintain **flow constraints**, it is necessary for the inward flow on each v_j to be exactly 1. By our assignment construction, it is clear that **all the n jobs have been assigned** to some persons.

- For proving the **limits**, each person is alloted $f[(u_i, \text{bor}_i)]$ boring jobs and $f[(u_i, \text{bor}_i)] + f[(u_i, \text{int}_i)]$ total jobs. By the capacity constraints followed by the flow, we have:

$$f[(u_i, \text{bor}_i)] \leq b_i$$
$$f[(u_i, \text{int}_i)] + f[(u_i, \text{bor}_i)] \leq d_i \textbf{ [by cap and flow constraints]}$$

- Also trivially the assignment constructed obeys the list L depicting the **willingness** pairs of jobs and people by the graph construction! This is because, if J_j is included in $\mathcal{A}[P_i]$ then either $f(bor_i, v_j) == 1$ or $f(int_i, v_j) == 1$. In any case, $(bor_i, v_j) \in E_{bor2job}$ or $(int_i, v_j) \in E_{int2job}$, which needs $(J_j, P_i) \in L$.

Therefore, given a valid maxflow with $|f| = n$, we showed how we can construct a valid assignment which successfully maps all the n jobs to people (with each person doing atmost b_i boring jobs and d_i total jobs) while obeying the list L of willingness.

(d)

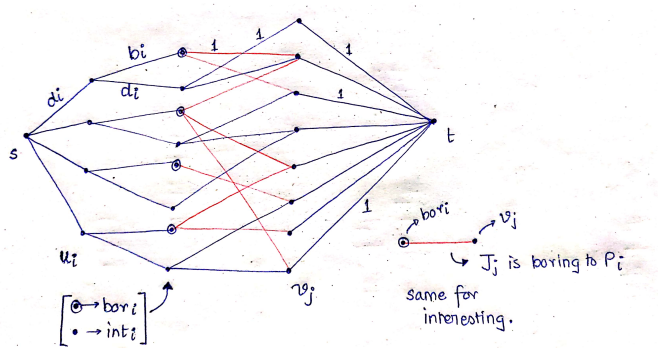
Problem: Finally, consider a variation where the classification of jobs as boring or interesting is done separately by each person. Thus each person indicates which jobs he/she is willing to do, and whether the job is boring or interesting for him/her. Given upper bounds on the total number of jobs, and the number of boring jobs that each person is willing to do, use max-flows to find a feasible assignment of jobs to persons, if it exists.

Solution: This question is very similar to the last subproblem! Notations: (J_i, P_j) are tuples which indicate that person P_i is willing to do job J_i. Let L be the list of these tuples. d_i denotes the upper bound on the total number of jobs the person is willing to do. b_i denotes the total number of boring jobs he/she is willing to do. We can model this problem as a max-flow problem over a graph G. We define a graph $G = (V, E, c)$. V represents the vertices, E represents the edges and c represents the capacity of the edges. Let's define the graph:

- $V = \{s\} \cup \{u_i | 1 \leq i \leq m\} \cup \{bor_i | 1 \leq i \leq m\} \cup \{int_i | 1 \leq i \leq m\} \cup \{v_j | 1 \leq i \leq n\} \cup \{t\}$
- $E = E_{source} \cup E_{bor} \cup E_{int} \cup E_{bor2job} \cup E_{int2job} \cup E_{sink}$
  - $E_{source} = \{(s, u_i) | 1 \leq i \leq m\}$
  - $E_{bor} = \{(u_i, bor_i) | 1 \leq i \leq m\}$
  - $E_{int} = \{(u_i, int_i) | 1 \leq i \leq m\}$
  - $E_{bor2job} = \{(bor_i, v_j) |$ if $(J_j, P_i) \in L$ and $J_j \in J_{i,boring}\}$
  - $E_{int2job} = \{(int_i, v_j) |$ if $(J_j, P_i) \in L$ and $J_j \in J_{i,interesting}\}$
  - $E_{sink} = \{(v_j, t) | 1 \leq j \leq n\}$
- $\forall e \in E, c(e) = \begin{cases} d_i & \text{if } e = (s, u_i) \in E_{\text{source}} \\ b_i & \text{if } e = (u_i, bor_i) \in E_{\text{bor}} \\ d_i & \text{if } e = (u_i, int_i) \in E_{\text{int}} \\ 1 & \text{if } e = (bor_i, v_j) \in E_{\text{bor2job}} \\ 1 & \text{if } e = (int_i, v_j) \in E_{\text{int2job}} \\ 1 & \text{if } e \in E_{\text{sink}} \end{cases}$

More notation explanations:

- **s** is the source
- $\boldsymbol{u_i}$ are the vertices corresponding intuitively to the m persons.
- $\boldsymbol{bor_i}$ vertices behaves like a switch which directs the flow of of the person P_i to the jobs he/she finds boring.
- $\boldsymbol{int_i}$ just like vertex $\boldsymbol{bor_i}$ directs the flow of of the person P_i to the jobs he/she finds interesting.
- $\boldsymbol{v_j}$ corresponds to jobs.
- $\boldsymbol{J_{i,boring}}$ corresponds to set of jobs which P_i finds to be boring.
- $\boldsymbol{J_{i,interesting}}$ corresponds to set of jobs which P_i finds to be interesting.
- $\boldsymbol{t}$ corresponds to sink.



We claim that the an assignment of all n jobs to willing persons (with each person doing atmost b_i boring(as chosen by him/her) jobs and d_i total jobs) is possible if and only if the maximum possible flow in the graph G above is equal to n.

Proof: We prove the claim in both directions

($\Rightarrow$) Suppose there is an assignment $\mathcal{A}$ of all n jobs to willing persons (with each person doing atmost b_i boring(as chosen by him/her) jobs and d_i total jobs). We show that this implies the existence of a flow in the graph G such that $|f| = n$.

The assignment will looks something like $\boldsymbol{\mathcal{A} : P \longrightarrow \mathcal{P}(J)}$, where P is the person set and $\mathcal{P}(J)$ represents the powerset of J. That is the assignment maps a person to a subset of jobs, while maintaining the **size of subset assigned** to P_i is **atmost** d_i, **boring(as chosen by him/her)** jobs in the subset is **atmost** b_i, the subsets are **disjoint** and $\forall$ jobs J_j assigned to P_i, $(J_j, P_i) \in L$.

In the forward direction, our precondition enables us to assume that we have the a **valid assignment**.

Let's design a flow based on this assignment:

$$\forall e \in E, f(e) = \begin{cases} |S_i| & \text{if } e = (s, u_i) \in E_{\text{source}} \text{ and } \mathcal{A} : P_i \longrightarrow S_i \text{ (set of jobs)} \\ |S_i \cap J_{i,boring}| & \text{if } e = (u_i, bor_i) \in E_{\text{bor}} \text{ and } \mathcal{A} : P_i \longrightarrow S_i \text{ (set of jobs)} \\ |S_i \cap J_{i,interesting}| & \text{if } e = (u_i, int_i) \in E_{\text{int}} \text{ and } \mathcal{A} : P_i \longrightarrow S_i \text{ (set of jobs)} \\ 1 & \text{if } e = (bor_i, v_j) \in E_{\text{bor2job}} \text{ and } J_j \in \mathcal{A}[P_i] \\ 1 & \text{if } e = (int_i, v_j) \in E_{\text{int2job}} \text{ and } J_j \in \mathcal{A}[P_i] \\ 1 & \text{if } e \in E_{\text{sink}} \\ 0 & \text{otherwise} \end{cases}$$

The flow f defined above obeys the **capacity constraints** by the properties of the assignment $\mathcal{A}$. It is easy to observe this, since $|S_i| \leq d_i$ and $|S_i \cap J_{i,boring}| \leq b_i$. By property of intersection, $|S_i \cap J_{i,interesting}| \leq d_i$

It is also easy to observe that the flow f as defined obeys the **flow constraints**. Let's look vertex by vertex:

- For vertex $\boldsymbol{u_i}$, inward flow $= |S_i|$ and outward flow $= |S_i \cap J_{i,boring}| + |S_i \cap J_{i,interesting}|$. By definition, $J_{i,boring} \cup J_{i,interesting} = J$ and $J_{i,boring} \cap J_{i,interesting} = \phi$. Therefore,

$$|S_i \cap J_{i,boring}| + |S_i \cap J_{i,interesting}| = |(S_i \cap J_{i,boring}) \cup (S_i \cap J_{i,interesting})| \text{ [as they are disjoint]}$$
$$\implies |S_i \cap J| = |S_i| \text{ as } [J_{i,boring} \cup J_{i,interesting} = J]$$

Therefore, inward flow matches with the outward flow!

- For vertex $\boldsymbol{bor_i}$, inward flow = $|S_i \cap J_{i,boring}|$ and outward flow = $|S_i \cap J_{i,boring}|$ since by construction, number of edges leaving $bor_i = |S_i \cap J_{i,boring}|$ each with flow 1.
- For vertex $\boldsymbol{int_i}$, inward flow = $|S_i \cap J_{i,interesting}|$ and outward flow = $|S_i \cap J_{i,interesting}|$. This also holds by construction.
- For vertex $\boldsymbol{v_j}$, outward flow = 1. We need to prove that the inward flow is exactly 1. **Arguments:**
  - A vertex v_j has inward flow if $\exists P_i | J_j \in \mathcal{A}[P_i]$. This obviously holds for every vertex since the assignment by definition maps all n jobs to persons. Therefore inward flow is atleast 1.
  - Also we know $\mathcal{A}[P_i] \cap \mathcal{A}[P_j] = \phi$, so there exists one and only one member of P to whom a job is alloted. Therefore inward flow is atmost one.

  Thus, the inward flow into every v_j is exactly 1. Flow constraint holds for this vertex as well.

Therefore we showed that we can build a valid flow from a given valid assignment! This proves the forward direction, since we have $|f| = n$, because there are n (number of jobs) outgoing edges from v_j and f assigns unit flow to each of them. Since all outgoing edges from those are saturated, f is also the maximum possible flow in the graph G. Thus, the maximum value of flow in graph G is equal to n.

($\Leftarrow$) We want to show that if we have a maxflow f of graph G and $|f| = n$, then it is possible to find an assignment $\mathcal{A}$ which maps all n jobs to willing persons (with each person doing atmost b_i boring jobs and d_i total jobs).

In this part we assume that we have a maxflow f of value n, which looks like something: $f : E \longrightarrow x$, where $x \in \mathbb{Z}$, i.e maps edges to integer flows, while obeying the capacity and flow constraints as we had defined earlier.

We build a possible assignment $\mathcal{A} : P \longrightarrow \mathcal{P}(J)$ for every person P_i from the given maxflow f by following this algorithm:

$$\mathcal{A}[P_i] = \{J_j | f(bor_i, v_j) == 1\} \cup \{J_j | f(int_i, v_j) == 1\}$$

Which means: **"Add the job J_j to the set of jobs alloted to P_i if $f(bor_i, v_j) == 1$ or $f(int_i, v_j) == 1$".**

Now we need to prove that this assignment indeed obeys the constraints of **disjointness**, provides **mapping for all** the n jobs, and maintains the **limits** set by b_i and d_i.

- The proposed assignment is definitely **disjoint** since any job cannot have two incoming flows! **Argument:** Every v_j has flow of 1 to the sink(since its a **maxflow** with $|f| = n$). So v_j cannot have multiple inward flows of value 1. That would violate the flow constraint.
- Therefore, in order to maintain **flow constraints**, it is necessary for the inward flow on each v_j to be exactly 1. By our assignment construction, it is clear that **all the n jobs have been assigned** to some persons.
- For proving the **limits**, each person is alloted $f[(u_i, \text{bor}_i)]$ boring jobs and $f[(u_i, \text{bor}_i)] + f[(u_i, \text{int}_i)]$ total jobs. By the capacity constraints followed by the flow, we have:

$$f[(u_i, \text{bor}_i)] \leq b_i$$
$$f[(u_i, \text{int}_i)] + f[(u_i, \text{bor}_i)] \leq d_i \text{ [by cap and flow constraints]}$$

- Also trivially the assignment constructed obeys the list L depicting the **willingness** pairs of jobs and people by the graph construction! This is because, if J_j is included in $\mathcal{A}[P_i]$ then either $f(bor_i, v_j) == 1$ or $f(int_i, v_j) == 1$. In any case, $(bor_i, v_j) \in E_{bor2job}$ or $(int_i, v_j) \in E_{int2job}$, which needs $(J_j, P_i) \in L$.

Therefore, given a valid maxflow with $|f| = n$, we showed how we can construct a valid assignment which successfully maps all the n jobs to people (with each person doing atmost b_i boring(as chosen by him/her) jobs and d_i total jobs) while obeying the list L of willingness.

Question 2

Given a directed acyclic graph, the following problems can be defined:

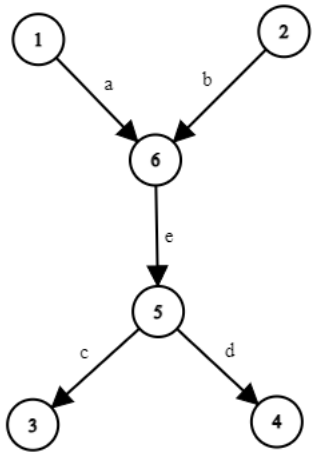
Minimum Path Vertex Cover (**MPVC**): Find the minimum number of paths in the graph such that every vertex is contained in *at least* one path

Minimum Disjoint Path Vertex Cover (**MDPVC**): Find the minimum number of paths in the graph such that every vertex is contained in *exactly* one path

Minimum Path Edge Cover (**MPEC**): Find the minimum number of paths in the graph such that every edge is contained in *at least* one path

Minimum Disjoint Path Edge Cover (**MDPEC**): Find the minimum number of paths in the graph such that every edge is contained in *exactly* one path

Note that MPVC is different from MDPVC. For the below graph, the answer to MPVC is 2 (Paths $\{(1,6,5,4),(2,6,5,3)\}$) and the answer to MDPVC is 3 (Paths $\{(1,6,5,4),(2),(3)\}$). Note that multiple vertex path covers may have the same cardinality.



MPEC is also different from MDPEC. For the above graph, the answer to MPEC is 2 (Paths $\{(1,6,5,4),(2,6,5,3)\}$) and the answer to MDPEC is 3 (Paths $\{(1,6,5,4),(2,6),(5,3)\}$). Note that multiple edge path covers may have the same cardinality.

For this assignment, we are supposed to solve: MPVC, MDPVC, and MDPEC.

MDPVC

Let the given DAG be $G = (V, E)$
Let $|V| = n$ and $|E| = m$
We construct a bipartite graph $G' = (V', E')$ as follows

$$\begin{aligned} V' &= V_{in} \cup V_{out} \\ V_{in} &= \{v_{in} \mid v \in V\} \\ V_{out} &= \{v_{out} \mid v \in V\} \\ E' &= \{\{u_{out}, v_{in}\} \mid (u, v) \in E\} \end{aligned}$$

Thus, $|V'| = 2n$ and $|E'| = m$.
V_{out} models the outgoing edges and V_{in} models the incoming edges.

Claim: G has a disjoint path vertex cover of size k iff G' has a matching of size $n - k$

Proof:

($\Rightarrow$) Let P be disjoint path vertex cover of size k.
As paths are trees, P is a forest with k components, thus it has exactly $n - k$ edges.
Let p_1, p_2 be any two arbitrary distinct paths in P.
$p_1 \cap p_2 = \phi$ since P is a disjoint path vertex cover.
For any edge $e_1 \in p_1$ and any edge $e_2 \in p_2$, $e_1 \cap e_2 = \phi$ due to paths being disjoint.
Thus, we construct a matching M as follows:

$$M = \bigcup_{p \in P} \{\{u_{out}, v_{in}\} \mid (u, v) \in p\}$$

Thus, $|M| = n - k$ and $|P| = k$

($\Leftarrow$) Let M be a matching in G' of size $n - k$.
Consider a graph defined as $P = (V, E_M)$, where $E_M = \{(u, v) \mid \{u_{out}, v_{in}\} \in M\}$
Claim: Each vertex of P has an in-degree of atmost 1 and an out-degree of atmost 1.
Proof: Consider a vertex v with in-degree > 1, thus, there exist $a, b \in V$ such that $(a, u), (b, u) \in E_M$ which implies that $\{a_{out}, u_{in}\}, \{b_{out}, u_{in}\} \in M$ which contradicts the definition of a matching.
The proof for atmost 1 out-degree is similar.
Thus, each connected component in P is a path.
As P covers all vertices, it's also a path vertex cover.
As each vertex of P has an in-degree of atmost 1 and an out-degree of atmost 1, no vertex is shared between paths, thus, P is a disjoint path vertex cover.
As P is acyclic, P is also a forest and since $|E_M| = n - k, |V| = n$, the forest has $n - (n - k) = k$ connected components.
Thus, $|P| = k$

Therefore, we can reduce the MDPVC problem to a maximum bipartite matching problem, as maximizing $n - k$ is equivalent to minimizing k. As discussed in class, this can be **modelled as a max-flow** problem which can be done in $\mathcal{O}(|V'||E'|) = \mathcal{O}(2nm)$ time.

Cycles Allowed

This approach does't work when cycles are allowed. Consider the graph $(V = \{1, 2, 3\}, E = \{(1, 2), (2, 3), (3, 1)\})$. For this, the approach discussed above returns 0 (as a matching of size 3 is possible), when the actual answer is 2 (Paths $\{(1, 2), (3)\}$).

MPVC

We can reduce this problem to MDPVC problem.
Let the given DAG be $G = (V, E)$
Let $|V| = n$ and $|E| = m$
Define $G' = (V, E')$ to be the transitive closure of G. (i.e $E' = E \cup \{(u, v) \mid v \text{ is reachable from } u \text{ in } G\}$[1])
We show that MPVC on G is equivalent to MDPVC on G'.
This is intuitive as a transitive closure allows the paths to avoid intersections.

Claim: There exists a (possibly non-disjoint) path vertex cover of G of size k iff there exists a disjoint path cover of G' of size k.

Proof:

($\Rightarrow$) If MPVC on G leads to disjoint paths, then we are done.
If not, then consider paths p_1, p_2 in PC_G that intersect at some non-terminal vertex v. (If they intersected at some terminal vertex, we can simply remove that vertex from one of the paths, without change in number of paths)
This means that we can write p_1 as $(p_1(start : u_1), (u_1, v), (v, w_1), p_1(w_1 : end))$ and p_2 as $(p_2(start : u_2), (u_2, v), (v, w_2), p_2(w_2 : end))$
In G' we can 'resolve' this intersection by modifying p_2 (without loss of generality) to $(p_2(start : u_2), (u_2, w_2)_t, p_2(w_2 : end))$, where $(u, v)_t$ denotes an edge that was added during the transitive closure. Note that as w_2 is reachable from u_2 in the original graph, on taking the transitive closure, we add (u_2, w_2).
Thus, all such intersections can be resolved similarly without any change in the number of paths.

($\Leftarrow$) Consider a path p in $PC_{G'}$ that uses some transitive-edge (u, v), then p can be written as $(p(start : u), (u, v), p(v : end))$
This also implies that v is reachable from u in G via some path q. Thus, the path $p' = (p(start : u), q, p(v : end))$ 'resolves' a transitive-edge without any change in the number of paths. Similarly, we can 'resolve' all the transitive-edges in the paths $PC_{G'}$ to get a (possibly non-disjoint) path vertex cover of G.

Thus, running the MDPVC algorithm on G' will give the MPVC value for G.

As a transitive closure may add $\mathcal{O}(m^2)$ new edges, the complexity for MPVC will be $\mathcal{O}(|V||E'|) = \mathcal{O}(nm^2)$

Cycles Allowed

This approach does't work when cycles are allowed, as transitive closure preserves cycles and MDPVC doesn't work when cycles are allowed.

[1] It is assumed that a node in G is not reachable to itself via a path of length 0, i.e the transitive closure doesn't add any self-loops

MDPEC

Let the given DAG be $G = (V, E)$
Let $|V| = n$ and $|E| = m$.
Let $G' = (V', E')$ be the corresponding line graph[1] of G.
$V' = E$ (we treat each edge as a node) and $(e_1, e_2) \in E'$ iff e_1's destination is the source of e_2.
We show that MDPVC on G' is equivalent to MDPEC on G.

Claim: G' has a disjoint path vertex cover of size k iff G has a disjoint path edge cover of size k

Proof:

($\Rightarrow$) Let $PC_{G'}$ be a a disjoint path vertex cover of size k of G'. Let $p \in PC_{G'}$ be any path. Let p have e_0 as the starting vertex, and let e_0 correspond to (u_0, v_0) in G. Create an equivalent path q for G as follows:

$$q = (u_0) \cup \bigcup_{(u,v) \in p} (v)$$

Doing this for all paths leads to a path cover PC_G for G such that $|PC_G| = |PC_{G'}| = k$. As $PC_{G'}$ covers all vertices of G', PC_G covers all edges of G (as each vertex in G' corresponds to an edge in G). As $PC_{G'}$ is vertex-disjoint, PC_G will be edge-disjoint (as two paths having a common edge in G implies that two paths have a common vertex in G').

($\Leftarrow$) Let PC_G be a a disjoint path edge cover of size k of G.
Let $q = (u_1, \ldots, u_r)$ be any path in PC_G. Then, create an equivalent path for G' as follows:

$$p = \bigcup_{i \in \{1,\ldots,r-1\}} ((u_i, u_{i+1}))$$

Doing this for all paths leads to a path cover $PC_{G'}$ for G' such that $|PC_{G'}| = |PC_G| = k$. As PC_G covers all edges of G, $PC_{G'}$ covers all edges of G' (as each edge in G corresponds to an vertex in G'). As PC_G is edge-disjoint, $PC_{G'}$ will be vertex-disjoint (as two paths having a common vertex in G implies that two paths have a common edge in G).

Thus, running the MDPVC algorithm on G' is equivalent to running the MDPEC algorithm on G.

As the line graph may have $\mathcal{O}(nm)$ edge [2], the time complexity will be $\mathcal{O}(|V'||E'|) = \mathcal{O}(m|E'|) = \mathcal{O}(nm^2)$

Cycles Allowed

This approach doesn't work when cycles are allowed. Let $G = K_3$, then G' is also K_3, and we know that MDPVC doesn't work when there are cycles.

[1] https://en.wikipedia.org/wiki/Line_graph#Line_digraphs
[2] https://math.stackexchange.com/questions/2689935/edges-in-a-line-graph

MDPEC (Faster)

Let the given DAG be $G = (V, E)$
Let $|V| = n$ and $|E| = m$

Algorithm and Time Complexity

```
// Initialize v.in_degree and v.out_degree to 0 for all v
// O(n) time
for v in V
    v.in_degree = 0;
    v.out_degree = 0;
end for

// Compute the in-degrees and the out-degrees
// O(n + m) time
for u in V
    u.out_degree = adj[u].size();
    for v in adj[u]
        v.in_degree++;
    end for
end for

// Compute the min. no. of paths in the graph such that every edge is in exactly one path
// O(n) time
mdpec = 0;
for v in V
    mdpec = mdpec + max(v.out_degree - v.in_degree, 0);
end for
```

The time complexity is $\mathcal{O}(n) + \mathcal{O}(n + m) + \mathcal{O}(n) = \mathcal{O}(n + m)$

Proof

Consider an arbitrary node v with in-degree d_i and out-degree d_o.
This means that v intercepts d_i paths (as paths are assumed to be disjoint).
If $d_o < d_i$, at least $d_i - d_o$ incoming paths will have to end at this vertex (if not, then atleast one outgoing edge from v will be shared between atleast 2 paths). Thus, in this case, at most d_o paths move forward and at least $d_i - d_o$ paths end, but no new paths need to be started from v.
If $d_o = d_i$, all incoming paths can move forward without any edge overlap, and thus, in this case too, no new paths need to be started from v.
If $d_o > d_i$, all incoming paths can move forward without any edge overlap, but now, $(d_o - d_i)$ new paths need to be started at v to maintain disjointness and to cover the remaining $(d_o - d_i)$ outgoing edges of v.
In all cases, no paths have any common edges and all incoming and outgoing edges are covered.
Thus, at each node, we need to start $\max((d_o - d_i), 0)$ new paths.
Hence, in total we need to need at least $\sum_{v \in V} \max((d_o(v) - d_i(v)), 0)$ paths.

Cycles Allowed

This approach does't work when cycles are allowed. Consider the graph $(V = \{1, 2, 3\}, E = \{(1, 2), (2, 3), (3, 1)\})$.
For this, the approach discussed above returns 0, when the actual answer is 2 (Paths $\{(1, 2, 3), (3, 1)\}$.

Question 3

Read the definition of reductions from the Lecture 26 and read pages 454–459 of the textbook by Kleinberg and Tardos. For each of the problems below, show that they can be reduced to the Independent Set decision problem and vice-versa.

The Independent Set decision problem (**IS**) is stated as follows:

Given graph G and a number k, does G contain an independent set of size k?

(a)

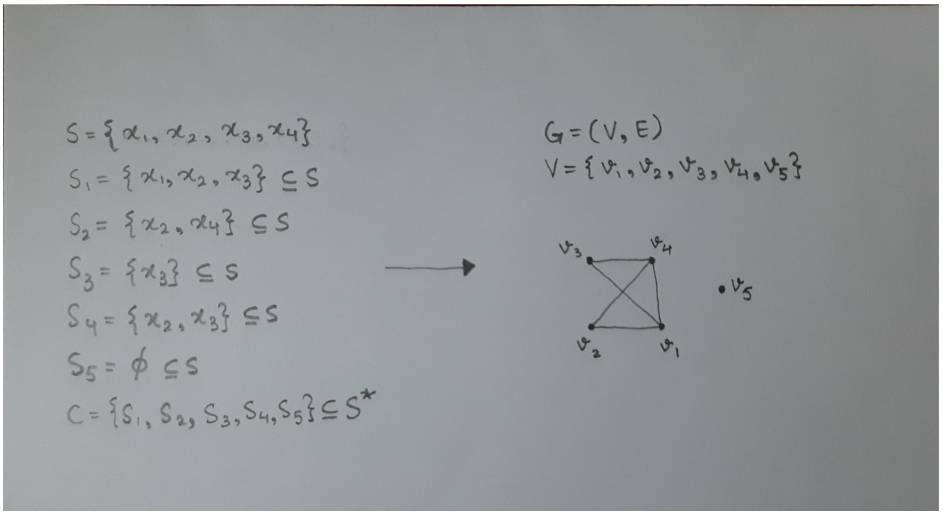
Problem: Disjoint Sets: Given a set S and a collection $C = \{S_1, S_2, \ldots, S_m\}$ of subsets of S, and a number k, are there k pairwise disjoint subsets in C?

Solution: Let us refer to the given problem as "Disjoint Sets" (**DS**).

- First, we show that the DS problem can be reduced to the IS problem. Suppose we are given an instance of DS, i.e., we have set S and a collection $C = \{S_1, S_2, \ldots, S_m\}$ of subsets of S, and a number k. We intend to find whether there are k pairwise-disjoint subsets in the C.

  We construct a graph $G = (V, E)$ as follows:

  - For every subset S_i $(1 \leq i \leq m)$, we add a vertex v_i to G.
  - For every pair of distinct vertices v_i and v_j in G, we connect the two of them by an edge in G if and only if $S_i \cap S_j \neq \phi$, i.e., the two corresponding subsets of S have a common element.



We now claim that there are k pairwise-disjoint subsets in C if and only if G has an independent set of size k vertices.

Proof. We prove this in both directions:

(⇒) Suppose there are k pairwise-disjoint subsets in the collection C. Let this pairwise-disjoint sub-collection be $C' = \{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$. We can prove that $V' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ forms an independent set of G.

Consider an arbitrary pair of distinct vertices v_{i_x} and v_{i_y}. Suppose they have an edge between them in G. Then, by construction of G, we know that $S_{i_x} \cap S_{i_y} \neq \phi$. But this contradicts the fact that S_{i_x} and S_{i_y} are pairwise-disjoint, as both belong to C'. Thus, v_{i_x} and v_{i_y} cannot be connected by an edge in G.

Since x and y were arbitrary, the set V' must form an independent set of G.

($\Leftarrow$) Suppose G has an independent set of size k vertices, which is $V' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$. We can prove that $C' = \{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ is a set of k pairwise-disjoint subsets of S.

Consider an arbitrary pair of distinct sets S_{i_x} and S_{i_y}. Suppose they are not pairwise-disjoint, i.e., $S_{i_x} \cap S_{i_y} \neq \phi$. This implies there is an edge connecting v_{i_x} and v_{i_y} in G, by the way in which G was constructed. But this contradicts the fact that both v_{i_x} and v_{i_y} belong to the independent set V'. Thus, S_{i_x} and S_{i_y} must be pairwise-disjoint.

Since x and y were arbitrary, the set C' must consist of k pairwise-disjoint subsets of S.

□

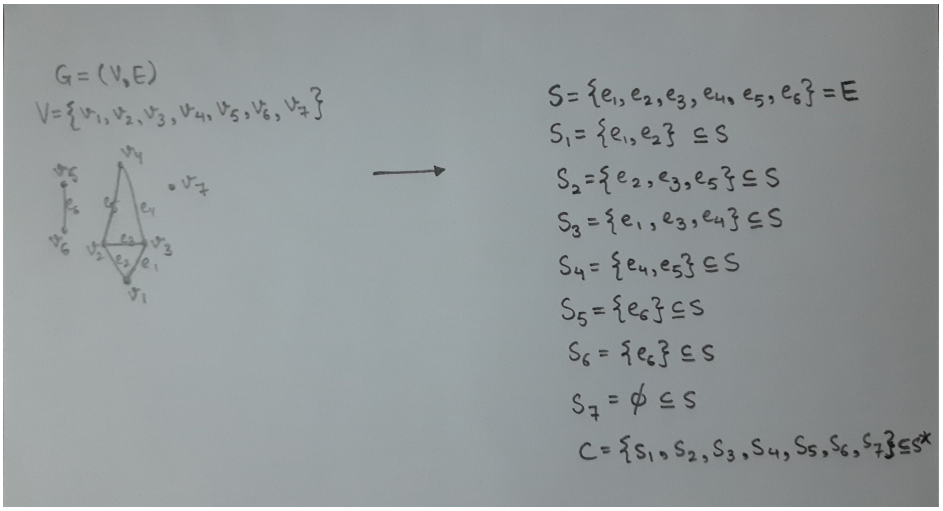
With this, we conclude that the DS problem can be reduced to the independent set problem.

- We now show that the IS problem can be reduced to the DS problem. Suppose we have an instance of the IS problem, i.e., given an undirected graph $G = (V, E)$ and a number k, we intend to find whether G has an independent set of size k.

  We construct a set S and a collection $C = \{S_1, S_2, \ldots, S_m\}$ of subsets of S as follows:

  - $S = E$
  - For each vertex $v_i \in V$, we add a subset S_i to C.
  - For each edge $\{v_i, v_j\} \in E$, we add edge $\{v_i, v_j\}$ to subset S_i and S_j.

  Intuitively, the way we have constructed C is such that every subset S_i contains all the edges incident on vertex v_i.



We now claim that G has an independent set of size k vertices if and only if there are k pairwise-disjoint subsets in C.

Proof. We prove this in both directions:

($\Rightarrow$) Suppose G has an independent set of size k vertices, which is $V' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$. We can prove that $C' = \{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$ is a set of k pairwise-disjoint subsets of S.

Consider an arbitrary pair of distinct sets S_{i_x} and S_{i_y}. Suppose they have an element (edge) in common. Then, by construction of C, we know that this edge must be incident on both of v_{i_x} and v_{i_y}, i.e., there is an edge joining v_{i_x} and v_{i_y}. But this contradicts the fact that both v_{i_x} and v_{i_y} belong to the independent set V'. Thus, S_{i_x} and S_{i_y} must be pairwise-disjoint.

Since x and y were arbitrary, the set C' must consist of k pairwise-disjoint subsets of S.

($\Leftarrow$) Suppose there are k pairwise-disjoint subsets in the collection C. Let this pairwise-disjoint sub-collection be $C' = \{S_{i_1}, S_{i_2}, \ldots, S_{i_k}\}$. We can prove that $V' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ forms an independent set of G.

Consider an arbitrary pair of distinct vertices v_{i_x} and v_{i_y}. Suppose they have an edge between them in G. Then, by construction of C, we know that this edge must belong in both S_{i_x} and S_{i_y}, i.e., $S_{i_x} \cap S_{i_y} \neq \phi$. But this contradicts the fact that S_{i_x} and S_{i_y} are pairwise-disjoint, as both belong to C'. Thus, v_{i_x} and v_{i_y} cannot be connected by an edge in G.

Since x and y were arbitrary, the set V' must form an independent set of G.

□

With this, we conclude that the independent set problem can be reduced to the DS problem.

(b)

Problem: Given a graph G and a number k, does G contain a dominating set of size at most k? A dominating set is a subset of vertices such that every vertex not in the subset is adjacent to some vertex in the subset.

Solution: Let us refer to the given problem as "Dominating Set" (**DS**).

- First, we prove a trivial but useful lemma.

  Lemma 1. *A graph $G = (V, E)$ has a dominating set of size at most k if and only if it has a dominating set of size k.*

  Proof. We prove this in both directions:

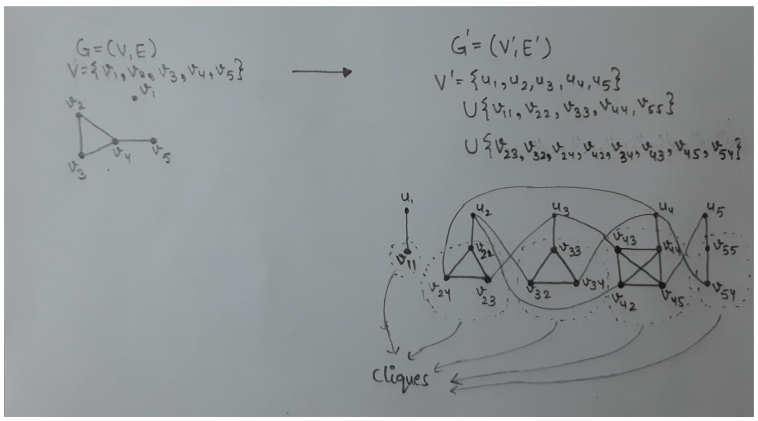
  ($\Rightarrow$) If we have a dominating set D of size $k' \leq k$, we can add $k - k'$ vertices from $V - D$ and form a new set D'.
  Every vertex v in $V - D'$ is also in $V - D$ has an adjacent vertex in D (as D is a dominating set) and as $D \subseteq D'$, v has an adjacent vertex in D'.
  Therefore, D' is a k-sized dominating set of G.

  ($\Leftarrow$) Dominating set of size k is also a dominating set of size at most k.

  □

- If $k \geq |V|$, we can trivially say that the answer is 'YES'. As $|V|$ is a dominating set of G.
  So, for reductions we can assume that $k|V|$, else it is trivial.

- We show that the DS problem can be reduced to the IS problem. Suppose we are given an instance of DS, i.e., we have an undirected graph $G = (V, E)$ and a number k, and we intend to find whether there is a at most k-sized subset of V such that every vertex not in the subset is adjacent to some vertex in the subset.
  We construct a graph $G' = (V', E')$ as follows:
  - For each vertex $v_i \in V$, we add vertices u_i and v_{ii} to V'.
  - For each edge $\{v_i, v_j\} \in E$, we add vertices v_{ij} and v_{ji} to V'.
  - For each vertex $v_i \in V$, add edges between all vertices of form v_{i*} to E'. (Forming cliques of form v_{i*} and n isolated points u_i)
  - For each vertex $v_i \in V$, add edge $\{u_i, v_{ii}\}$ to E'.
  - For each edge $\{v_i, v_j\} \in E$, add edges $\{u_i, v_{ji}\}$ and $\{u_j, v_{ij}\}$ to E'.



Let $|V| = n$ and $|E| = m$. Then, $|V'| = 2(m + n)$.

We now claim that G' has a dominating set of size at most k if and only if G has an independent set of size $2n - k$ vertices.

Proof. We prove this in both directions:

($\Rightarrow$) Suppose G has an at most k-sized dominating set. Then, by Lemma 1 from above, it also has an exactly k-sized dominating set. Let $D = \{v_{i_1}, v_{i_2}, \cdots, v_{i_k}\}$ be this k-sized dominating set of G. We need to construct a $(2n - k)$-sized independent set I of G'.

We include u_i corresponding to all $v_i \notin D$, in I. All of these are independent vertices as we don't have edges connecting u_is. ($n - k$ such vertices)

Now we include one vertex from every clique of form v_{i*}, as follows:

* if $v_i \in D$ (k such vertices), we directly include v_{ii} in I. It only has edges to vertices belonging to the clique and to u_i. As we haven't included u_i, we can include v_{ii}.
* if $v_i \notin D$ ($n - k$ such vertices), we would have a vertex $v_j \in D$, such that $\{v_i, v_j\} \in E$ as D is a dominating set of G. We include $v_{ij} \in V'$ in I. It only has edges to vertices belonging to the clique and to u_j. As we haven't included u_j, we can include v_{ij}.

Since we added vertices without destroying the independence constraint, I is a $(2n - k)$-sized independent set of G'.

($\Leftarrow$) Suppose I is a $(2n-k)$-sized independent set of G'. We can only have one vertex corresponding to each clique of the form v_{i*}, i.e. at most n vertices of the form v_{**}. So, we would have at least $(n - k)$ vertices of the form u_*. Let this set be $I' = \{u_{i_1}, u_{i_2}, \dots\}$.

We claim $D = \{v_i | u_i \notin I'\}$ is an at most k-sized dominating set of G.

For every vertex v_i in $V - D = \{v_i | u_i \in I'\}$,

* if no vertex corresponding to the clique v_{i*} is in I, then we can replace u_i with v_{ii}, thus adding v_i to D. (Remember that, as we can never have more than one vertex per clique in I, size of D never exceeds k.)
* if there is a vertex corresponding to the clique v_{i*} in I, say v_{ij}, then we know for sure that $\{v_i, v_j\} \in E$, by construction of G' and that $u_j \notin I'$, due to independence constraint (we have an edge joining u_j and v_{ij} by construction of G'). As $u_j \notin I'$, $v_j \in D$ and we found an adjacent vertex for $v_i \notin D$.

Since we showed that we showed that for every vertex not in D we can either find an adjacent vertex in D or include it in D with maintaining the size at most k. D is a at most k-sized dominating set of G.

□

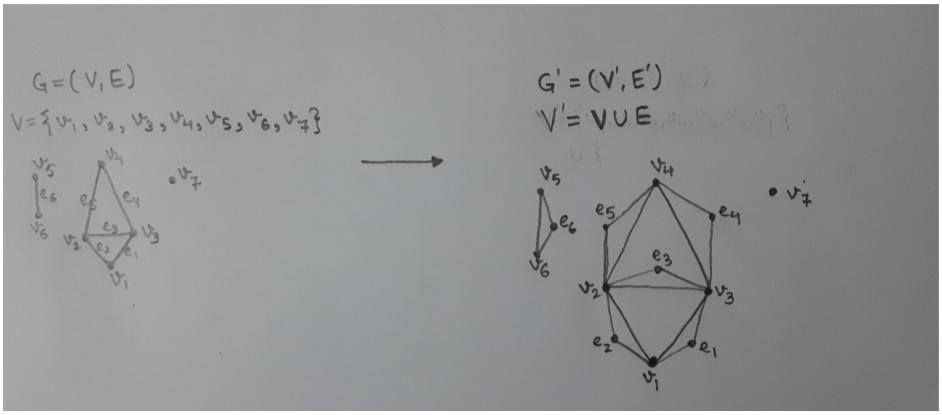
With this, we conclude that the DS problem can be reduced to the independent set problem.

- We now show that the IS problem can be reduced to the DS problem. Suppose we have an instance of the IS problem, i.e., given an undirected graph $G = (V, E)$ and a number k, we intend to find whether G has an independent set of size k.

  We construct a graph $G' = (V', E')$ as follows:

  – We start with $V' = V$ and $E' = E$.

  – For each edge $e_k = \{v_i, v_j\} \in E$, we add a vertex v_{e_k} to V' and two edges $\{v_i, v_{e_k}\}$ and $\{v_{e_k}, v_j\}$ to E'.



Let n_{Iso} denote the number of isolated vertices and set $k' = k - n_{Iso}$.

We now claim that G has an independent set of size k vertices if and only if G' has a dominating set of size at most $|V| - k'$.

Proof. We prove this in both directions:

($\Rightarrow$) Suppose I is a k-sized independent set of G. We remove all the isolated vertices in G from I to form I', a at least k'-sized independent set of G. Then $V - I'$ must be an at most $(|V| - k')$-sized dominating set of G'.

Since there is no pair of connected vertex in I, then each non-isolated vertex in I' is connected to some vertex in $V - I'$. Each isolated vertex is included in $V - I'$. Moreover, every newly added vertex in G' (of form v_{e_i}) is also connected to some vertices in $V - I'$, since both its neighbors cannot belong to I' due to the edge in between.

Thus, the set $V - I'$ must form a dominating set of size $|V| - k'$ in G'.

($\Leftarrow$)] Suppose G has an at most $(|V| - k')$-sized dominating set. Then, by Lemma 1 from above, it also has an exactly $(|V| - k')$-sized dominating set. Let D be this $(|V| - k')$-sized dominating set of G', then we can safely assume that all vertices in D are in V (since if D contains an added vertex $v \notin V$, we can replace v by one of its adjacent vertices in V and still have a dominating set of the same size; and even if both adjacent vertices of v are already included in the dominating set, we can simply just exclude v from the dominating set and include a random vertex in V that wasn't already in the dominating set).

We claim $V - D$ is a k'-sized independent set of G.

Suppose $V - D$ is not independent and contains a pair of vertices v_i and v_j and the edge $e_k = (v_i, v_j)$ is in E. Then in G', the added vertex v_{e_k} need to be dominated by either v_i or v_j, that is at least one of v_i and v_j is in D.

Since i and j were arbitrary choices, we reach the conclusion that $V - D$ is a k'-sized independent set of G.

D must contain every isolated vertex in V since they don't have any adjacent vertex. We add all the isolated vertices in G to $V - D$ to form I, a k-sized independent set of G. (Adding isolated vertices doesn't affect independence)

□

With this, we conclude that the independent set problem can be reduced to the DS problem.

(c)

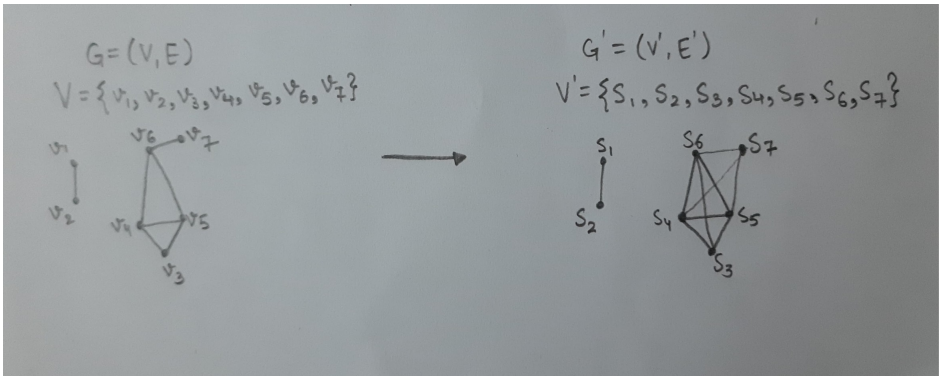
Problem: Given a graph G, and a number k, is there a subset of k vertices in G such that there is no path of length at most 2 between any two vertices in the subset?

Solution: Let us refer to the given problem as "Paths Greater than 2" (**PG2**).

- First, we show that the PG2 problem can be reduced to the IS problem. Suppose we are given an instance of the PG2 problem, i.e., we have an undirected graph $G = (V, E)$ and a number k, and we intend to find whether there is a k-sized subset of V such that no pair of vertices in this subset is connected by a path of length ≤ 2.

  Let $V = \{v_1, v_2, \ldots, v_n\}$. Consider a new undirected graph $G' = (V', E')$, formed as follows:

  - For each vertex $v_i \in V$, we form a set $S_i = \{v_i\} \cup \{v \in V \mid \{v_i, v\} \in E\}$, i.e., S_i consists of v_i along with all vertices adjacent to v_i. We add S_i to the vertex set V' of the new graph G'.
  - For every pair of distinct sets $S_i, S_j \in V'$, we add edge $\{S_i, S_j\}$ to E' if and only if $S_i \cap S_j \neq \phi$.



  We now claim that G has a subset of k vertices, such that there is no path of length at most 2 between any two vertices in the subset, if and only if G' has an independent set of k vertices.

  Proof. We prove both directions:

  ($\Rightarrow$) Suppose G has a subset of k vertices satisfying the above property. Pick two arbitrary vertices v_i and v_j from such a subset of k vertices, and consider corresponding sets S_i and S_j, as defined previously. Suppose S_i and S_j are connected by an edge in G', i.e, $S_i \cap S_j \neq \phi$. This can be due to three reasons; either $v_i \equiv v_j$; or v_i and v_j are adjacent, thus being part of each other's set; or v_i and v_j share a common adjacent vertex. In any case, there is a path of length at most 2 connecting v_i and v_j in G (length 0 in first case, 1 in second case, 2 in third case). Thus, we have a contradiction; v_i and v_j cannot belong to the same subset of k vertices if they differ by a path ≤ 2. Thus, we know S_i and S_j cannot be connected by an edge in G'. Since v_i and v_j were arbitrary, we conclude that there is no edge between any two S_i's corresponding to v_i's from the k-sized subset; i.e., the corresponding vertices in G' form a k-sized independent set.

  ($\Leftarrow$) Suppose G' has an independent set of size k, i.e., we have k sets of type S_i such that no two of them intersect. Consider an arbitrary pair S_i, S_j and corresponding v_i, v_j, and suppose v_i and v_j have a path of length at most 2 between them. Once again, we get the three cases described in the proof just above (forward direction), and we conclude that S_i and S_j must intersect for this to be true. Thus we reach a contradiction, which means v_i and v_j cannot have a path of length ≤ 2 between them. Since they were arbitrarily picked, we conclude that the corresponding vertices in G form a k-sized subset satisfying the no path ≤ 2 constraint.

□

With this, we conclude that the PG2 problem can be reduced to the independent set problem.

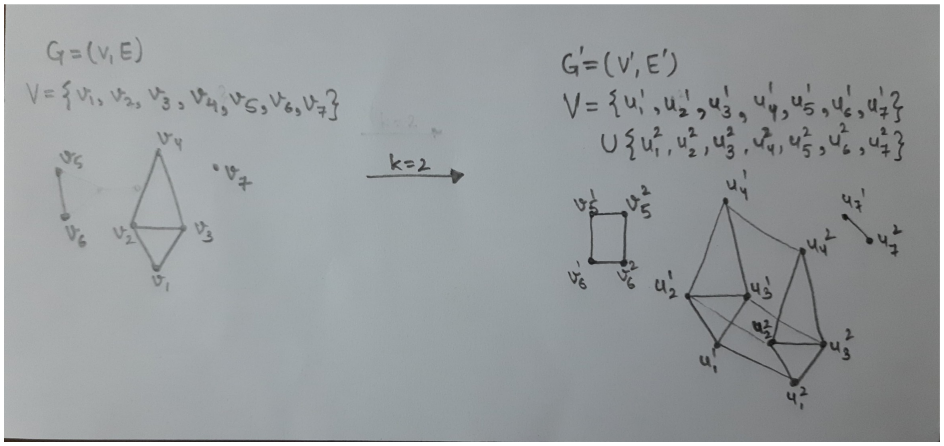
- We now show that the IS problem can be reduced to the PG2 problem. Suppose we have an instance of the IS problem, i.e., given an undirected graph $G = (V, E)$ and a number k, we intend to find whether G has an independent set of size k.

  Let $V = \{v_1, v_2, \ldots, v_n\}$. Consider a new graph $G' = (V', E')$, constructed as follows:

  - For each vertex $v_i \in V$, we add k vertices $u_i^1, u_i^2, \ldots, u_i^k$ to V' in G'.
  - For each edge $\{v_i, v_j\} \in E$, we add edges $\{u_i^1, u_j^1\}, \{u_i^2, u_j^2\}, \ldots, \{u_i^k, u_j^k\}$ to E' in G'.
  - For every set $\{u_i^1, u_i^2, \ldots, u_i^k\}$ of vertices in G' corresponding to a single vertex v_i in G, we add the edge $\{u_i^p, u_i^q\}$ to E' for all distinct pairs of p and q. That is, for all i from 1 to n, we form a complete graph (K_k) with the vertices $u_i^1, u_i^2, \ldots, u_i^k$.

  Intuitively, the way we have constructed G' is such that G' has k layers, each of which is a copy of G, and each vertex is connected to all of its copies by an edge.

  Just for clarity, the final graph G' has $|V'| = k|V|$ and $|E'| = k|E| + \binom{k}{2}|V|$.



We now claim that G has an independent set of k vertices if and only if G' has a subset of k vertices such that no two vertices in this subset have a path ≤ 2 connecting them.

Note: Before moving into the proof, note that there are two types of edges in this graph G'; there are edges within the same layer, that connect some u_i^l to some u_j^l, for some i, j, l; and there are edges between layers, that connect some u_i^x to some u_i^y, for some i, x, y. Thus, a movement along a single edge can change either the subscript or the superscript on the initial vertex, but not both.

Proof. We prove both directions:

(⇒) Suppose G has an independent set of k vertices. Let this independent set be $V_I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\} \subseteq V$. Then, we can prove that the set $V_I' = \{u_{i_1}^1, u_{i_2}^2, \ldots, u_{i_k}^k\} \subseteq V'$ is a k-sized subset of vertices of G' satisfying the no path ≤ 2 constraint.

Let us pick two arbitrary distinct vertices $u_{i_x}^x$ and $u_{i_y}^y$. Suppose there is a path of length ≤ 2 connecting them. Since V_I had k distinct elements, and since the numbers 1 to k are all distinct, we have $i_x \neq i_y$ and $x \neq y$. Thus, any path of length 2 from $u_{i_x}^x$ to $u_{i_y}^y$ must use one edge to change superscript x to y and one edge to change subscript i_x to i_y ($u_{i_x}^x - u_{i_x}^y - u_{i_y}^y$), and so

there must be an edge $\{u^y_{i_x}, u^y_{i_y}\}$ in G'. Because of the way G' was constructed, this implies the existence of an edge $\{v_{i_x}, v_{i_y}\}$ in G, i.e., this implies V_I is not an independent set. Thus, we reach a contradiction, and we conclude that there is no path ≤ 2 connecting $u^x_{i_x}$ and $u^y_{i_y}$.

Since x and y were arbitrary choices, we reach the conclusion that V'_I as defined above is a k-sized subset satisfying the no path ≤ 2 constraint.

($\Leftarrow$) Suppose G' has a subset of size k satisfying the no path ≤ 2 constraint. Let this set be $V'_I = \{u^{l_1}_{i_1}, u^{l_2}_{i_2}, \ldots, u^{l_k}_{i_k}\} \subseteq V'$. We can then prove that $V_I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\} \subseteq V$ is an independent set of G.

Pick two arbitrary vertices v_{i_x} and v_{i_y} from V_I above, and suppose they have an edge connecting them in G. Using this along with the way graph G' was constructed, we have that G' has an edge $\{u^{l_x}_{i_x}, u^{l_x}_{i_y}\}$, and another edge $\{u^{l_x}_{i_y}, u^{l_y}_{i_y}\}$. Thus, there is a path connecting $u^{l_x}_{i_x}$ and $u^{l_y}_{i_y}$ of length 2. This is a contradiction to the fact that they both are included in a set of vertices of G' such that no two vertices have a path ≤ 2 between them in G'. Thus, there cannot be an edge between v_{i_x} and v_{i_y} in G.

Since x and y were arbitrary choices, we may conclude that V_I as defined above is indeed an independent set of size k in graph G.

□

With this, we conclude that the independent set problem can be reduced to the PG2 problem.

(d)

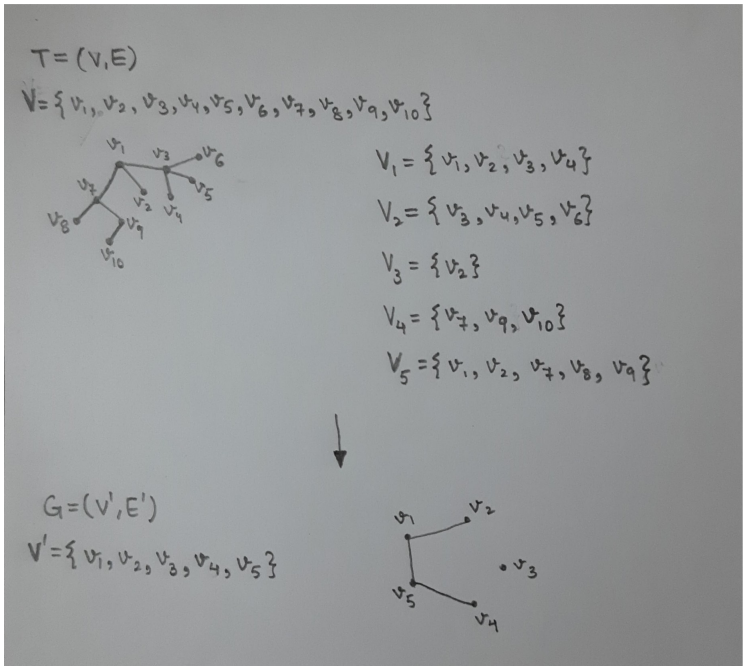
Problem: Given a tree T, and a collection $\{T_1, T_2, \ldots, T_m\}$ of subtrees of T, are there k edge-disjoint subtrees in the given collection? Show that if the subtrees are required to be vertex-disjoint, the problem can be solved in polynomial-time.

Solution: Let us refer to the given problem as "Edge Disjoint Sub-Trees" (**EDST**).

- First, we show that the EDST problem can be reduced to the IS problem. Suppose we are given an instance of EDST, i.e., we have tree $T = (V, E)$ and a collection of subtrees $\{T_1, T_2, \ldots, T_m\}$, where $T_i = (V_i, E_i)$, and a number k. We intend to find whether there are k edge-disjoint subtrees in the set.

  We construct a new graph $G' = (V', E')$ as follows:

  - For every subtree T_i $(1 \leq i \leq m)$, we add a vertex v_i to G'.
  - For every pair of distinct vertices v_i and v_j in G', we connect the two of them by an edge in G' if and only if $E_i \cap E_j \neq \phi$, i.e., the two corresponding subtrees of T have intersecting edge sets.



  We now claim that there are k edge-disjoint subtrees in the given collection if and only if G' has an independent set of size k.

  Proof. We prove this in both directions:

  ($\Rightarrow$) Suppose there are k edge-disjoint subtrees in the collection. Let this edge-disjoint set be $S = \{T_{i_1}, T_{i_2}, \ldots, T_{i_k}\}$. We can prove that $S' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ forms an independent set of G'.
  Consider an arbitrary pair of distinct vertices v_{i_x} and v_{i_y}. Suppose they have an edge between them in G'. Then, by construction of G', we know that $E_{i_x} \cap E_{i_y} \neq \phi$. But this contradicts the fact that T_{i_x} and T_{i_y} are edge-disjoint, as both belong to S. Thus, v_{i_x} and v_{i_y} cannot be connected by an edge in G'.
  Since x and y were arbitrary, the set S' must form an independent set of G'.

($\Leftarrow$) Suppose G' has an independent set of size k, which is $S' = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$. We can prove that $S = \{T_{i_1}, T_{i_2}, \ldots, T_{i_k}\}$ is a set of k edge-disjoint subtrees of T.
Consider an arbitrary pair of distinct subtrees T_{i_x} and T_{i_y}. Suppose they are not edge-disjoint, i.e., $E_{i_x} \cap E_{i_y} \neq \phi$. This implies there is an edge connecting v_{i_x} and v_{i_y} in G', by the way in which G' was constructed. But this contradicts the fact that both v_{i_x} and v_{i_y} belong to the independent set S'. Thus, T_{i_x} and T_{i_y} must be edge-disjoint.
Since x and y were arbitrary, the set S must consist of k edge-disjoint subtrees of T.

□

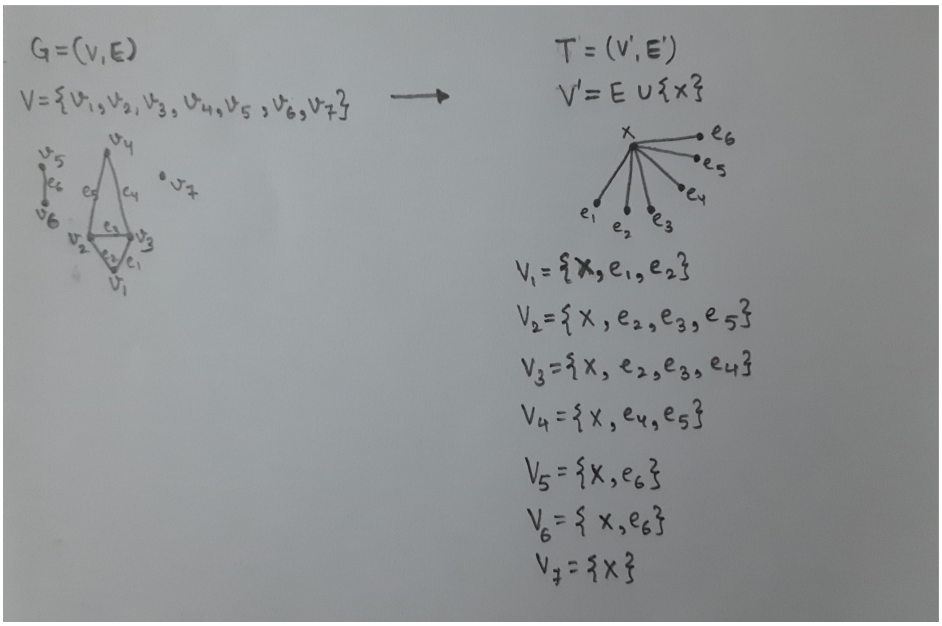
With this, we conclude that the EDST problem can be reduced to the independent set problem.

- We now show that the IS problem can be reduced to the EDST problem. Suppose we have an instance of the IS problem, i.e., given an undirected graph $G = (V, E)$ and a number k, we intend to find whether G has an independent set of size k.

  We construct a tree $T = (V', E')$ and a collection of subtrees $\{T_1, T_2, \ldots, T_m\}$, where $T_i = (V_i, E_i)$ as follows:

  - $V' = E \cup rootX$
  - For each edge $\{v_i, v_j\} \in E$, we add edge $\{\{v_i, v_j\}, rootX\}$ to E'.
  - For each edge $\{v_i, v_j\} \in E$, add $\{v_i, v_j\}$ to V_i and V_j.
  - For each vertex $v_i \in V$, add $rootX$ to V_i.
  - T_i is induced subtree of T with vertices V_i.

  Intuitively, the way we have constructed T is such that every subtree T_i has vertex set V_i which contains all the edges incident on vertex v_i.



We now claim that G has an independent set of size k vertices if and only if there are k pairwise-disjoint subsets in C.

Proof. We prove this in both directions:

($\Rightarrow$) Suppose G has an independent set of size k vertices, which is $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$. We can prove that $\{T_{i_1}, T_{i_2}, \ldots, T_{i_k}\}$ is a set of k edge-disjoint subtrees of T.
Consider an arbitrary pair of distinct subtrees T_{i_x} and T_{i_y}. Suppose they have an edge in common. Then, by construction of T, we know that this edge must be between $rootX$ and $\{v_{i_x}, v_{i_y}\}$, i.e., there is an edge joining v_{i_x} and v_{i_y} in G. But this contradicts the fact that both v_{i_x} and v_{i_y} belong to the independent set I. Thus, T_{i_x} and T_{i_y} must be edge-disjoint.
Since x and y were arbitrary, $\{T_{i_1}, T_{i_2}, \ldots, T_{i_k}\}$ must consist of k edge-disjoint subtrees of T.

($\Leftarrow$) Suppose there are k edge-disjoint subtrees in the collection. Let this edge-disjoint sub-collection be $\{T_{i_1}, T_{i_2}, \ldots, T_{i_k}\}$. We can prove that $I = \{v_{i_1}, v_{i_2}, \ldots, v_{i_k}\}$ forms an independent set of G.
Consider an arbitrary pair of distinct vertices v_{i_x} and v_{i_y}. Suppose they have an edge between them in G. Then, by construction of T, we know that this edge $\{v_{i_x}, v_{i_y}\}$ must belong in both V_{i_x} and V_{i_y}, i.e., E_{i_x} and E_{i_y} contains edge $\{\{v_{i_x}, v_{i_y}\}, rootX\}$, i.e., $E_{i_x} \cap E_{i_y} \neq \phi$. But this contradicts the fact that T_{i_x} and T_{i_y} are edge-disjoint, as both belong to the sub-collection. Thus, v_{i_x} and v_{i_y} cannot be connected by an edge in G.
Since x and y were arbitrary, the set I must form an independent set of G.

□

With this, we conclude that the independent set problem can be reduced to the EDST problem.

Now, we consider the problem of finding whether a given collection of subtrees $S = \{T_1, T_2, \ldots, T_m\}$ of a tree T contains k vertex-disjoint subtrees. Let us call this the "Vertex Disjoint Sub-Trees" (**VDST**) problem. Below, we propose an algorithm that computes the size of the largest possible set $S' \in S$ of vertex-disjoint subtrees.

Suppose the given tree is $T = (V, E)$ and the set of subtrees is $S = \{T_1, T_2, \ldots, T_m\}$, with each $T_i = (V_i, E_i)$. Let $n = |V|$, the number of vertices in T. The algorithm proposed consists of the following steps:

1. Convert the tree T into a rooted tree, by generating an ordering over the vertices in V.
   (a) We first select any vertex in V to be the root r of T.
   (b) We perform a depth-first search (DFS) over the entire T, starting at r, and the vertices in T are allocated parents or children according to this DFS.
   (c) After this DFS is over, we determine the root node r_i of every subtree T_i in S. For every $T_i \in S$, we iterate over its vertex set V_i, and we find the vertex having the smallest DFS number, i.e., the vertex in the subtree which was reached first by the DFS (in other words, the vertex closest to root r of T). That vertex is declared to be the root r_i of T_i.
2. Now, we use dynamic programming to determine the maximum possible number of vertex-disjoint subtrees $T_i \in S$ in T.
   (a) **Sub-Problems**: For each $v \in V$ in the tree T, let $P(v)$ denote the maximum possible number of vertex-disjoint subtrees $T_i \in S$ that all lie on or below v in the tree T, i.e., we only consider those subtrees that cover v and/or its descendants in the tree T, and we exclude those trees that pass through vertices in other parts of T. Our goal is to compute $P(r)$, where r is the root of the tree, since all vertices are descendants of the root.
   (b) **Recursive Relation**: At any vertex $v \in V$, we have the following recursive relation:

$$P(v) = \max\left(\sum_{u \in \{\text{children of } v\}} P(u), \quad 1 + \max_{T_i \in S,\ r_i = v} \{ \sum_{u \in \{\text{children of } T_i\text{'s leaves}\}} P(u)\}\right)$$

Of all the subtrees T_i rooted at any v (i.e., $r_i = v$), we can choose at most one. If we choose any T_i, then we eliminate all its vertices from being included in any other chosen subtree. Thus, we can only include the vertices which are children of the leaves of T_i, or their descendants, which means we compute the maximum possible number of vertex-disjoint subtrees under every such (child of leaf) vertex, and add them all up. To maximize the number of vertex-disjoint subtrees chosen, we take the maximum over all subtrees rooted at v. This explains the second term in the expression above.

However, we could also choose to exclude all the subtrees rooted at v from our set of vertex-disjoint subtrees. In this case, the answer would be the sum of the maximum possible number of vertex-disjoint subtrees under every child vertex of v. This explains the first term in the expression above.

Since we want to maximize the outcome, we take the maximum of both of them. This gives the recursive relation stated above. Thus, we have proved its correctness.

Note: The base case of this recursion is at the leaves of T. If a leaf node in itself forms an entire subtree in S, then $P(v)$ for that leaf would be 1. Otherwise, it would be zero.

(c) **Memoization**: For memoization in this DP-based solution, we store the value of $P(v)$ for every vertex $v \in V$. This requires a table of size n, with constant time required for table look-up.

(d) **Acyclicity of Dependencies**: Each $P(v)$ computation depends on either the children of v or the children of the leaves of a tree rooted at v. In either case, the next $P(u)$'s are calculated for vertices which are descendants of v in the tree T. Clearly, these recursive calls cannot lead back to v, as they always move downwards in the tree.

(e) **Time Complexity Analysis**: Here, we analyze the time complexity of the DP part of our solution. In the recursive function defined above, each node has to calculate the first term. With knowledge of all required $P(u)$ values from the memoization table, computing the first term requires O(no. of children) time at each node v. Summing this over all nodes, we find that computing the first term takes $O(n)$ time in total. As for the second term, it has to be computed exactly m times in the entire algorithm, since there are only m subtrees T_i in S. Each time, the children of its leaves are $O(n)$ in number, which means the second term computation takes a total time of $O(mn)$ throughout the algorithm. Thus, with dynamic programming, this part of the solution takes only $O(n) + O(mn) = O(mn)$ time.

3. **Overall Time Complexity Analysis**: As explained above, the DP part of our solution requires $O(mn)$ time to execute. But we also have to consider the first step, that is establishing an ordering over the tree T and allocating roots to all subtrees T_i. The DFS visits every node exactly once, so the DFS takes $O(n)$ time. After that, iterating over the vertex sets of all subtrees $T_i \in S$ (to determine the root node r_i) takes $O(n)$ time for each of m sets, bringing it to a total of $O(mn)$ time.

   Since both parts of our solution require $O(mn)$ time, the overall time complexity of this solution is $O(mn)$, that is, polynomial time!