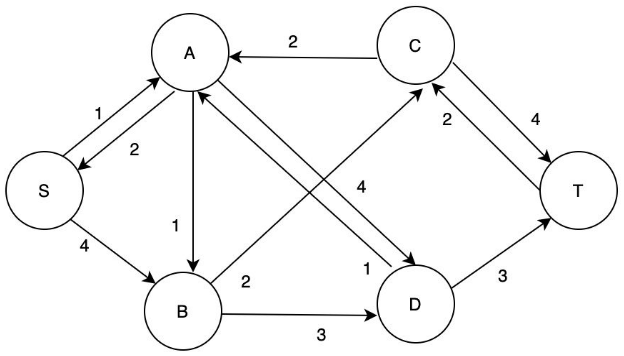
CSCI 570 - HW 4

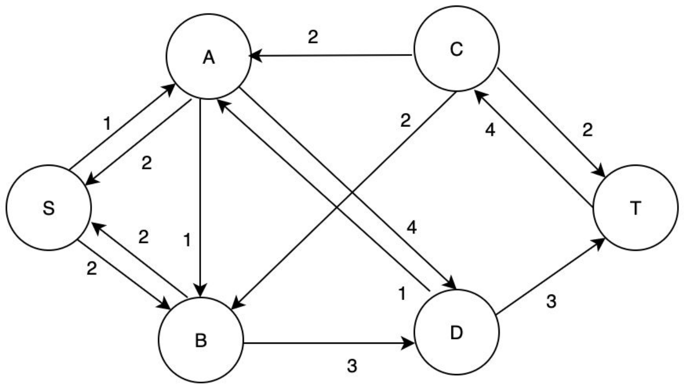
1. **Compute the max flow and min cuts.**

   a.

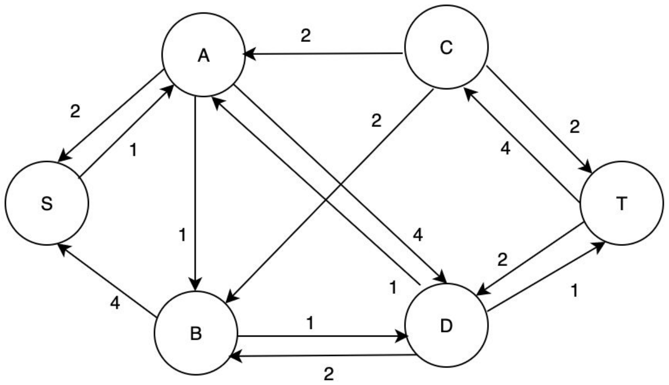
   - Iteration 1:
     $path: \ S \rightarrow A \rightarrow C \rightarrow T \quad Bottleneck = 2 \ \ Flow = 2$



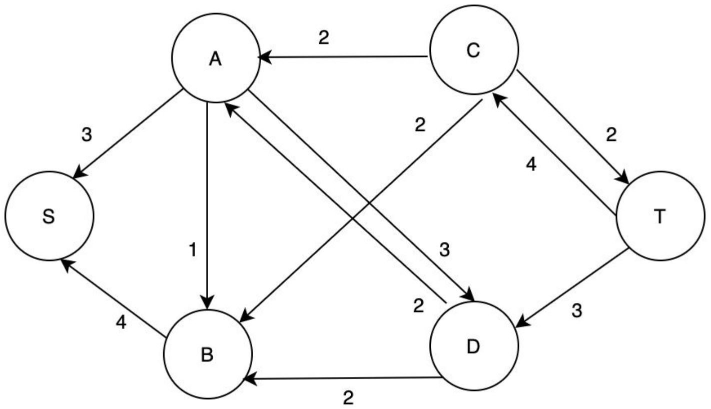
   - Iteration 2:
     $path: \ S \rightarrow B \rightarrow C \rightarrow T \quad Bottleneck = 2 \ \ Flow = 2 + 2 = 4$



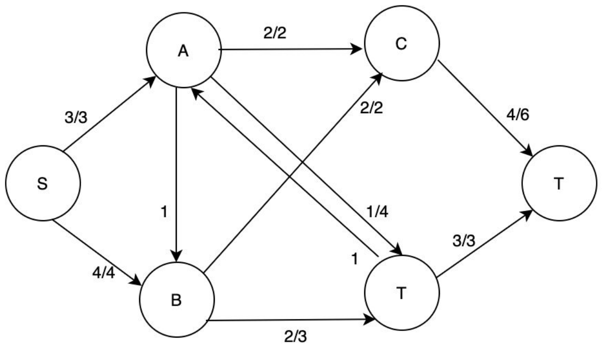
- Iteration 3:
  $path: \ S -> \ B \ -> D -> T \quad Bottleneck = 2 \ \ Flow = \ 4 + 2 = 6$



- Iteration 4:
  $path: \ S -> \ A \ -> D -> T \quad Bottleneck = 1 \ \ Flow = 6 + 1 = 7$



Final Residual Graph:

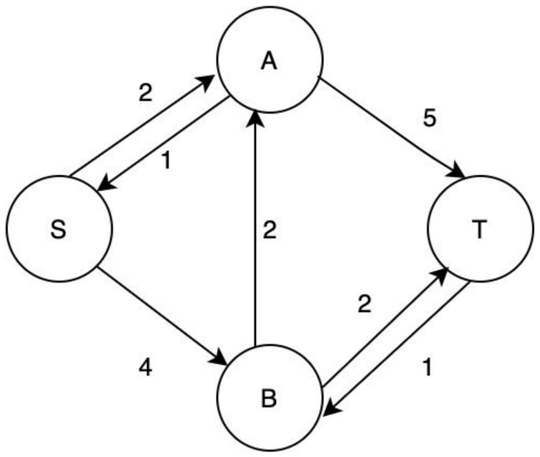


Max Flow: 7
Min cuts:
There are 2 S-T min cuts:

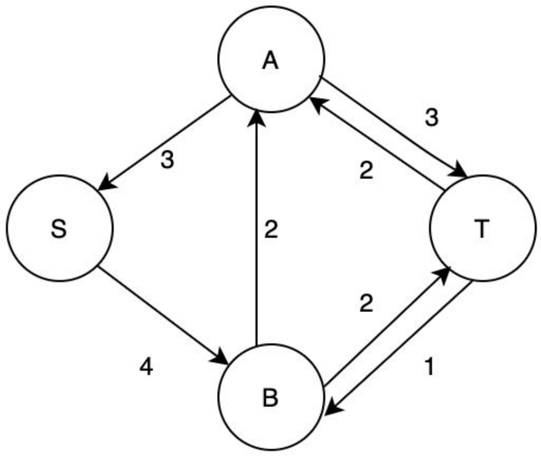
1. A={S} B={A,B,C,D,T}
2. A={S,A.B,D} B={C,T}

b.

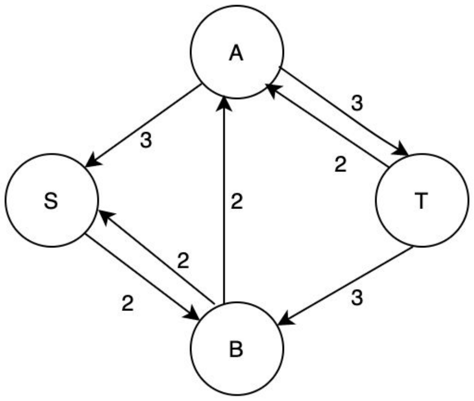
- Iteration 1:
  path : $S \text{->} A \text{->} B \text{->} T$ *Bottleneck* = 1 *Flow* = 1



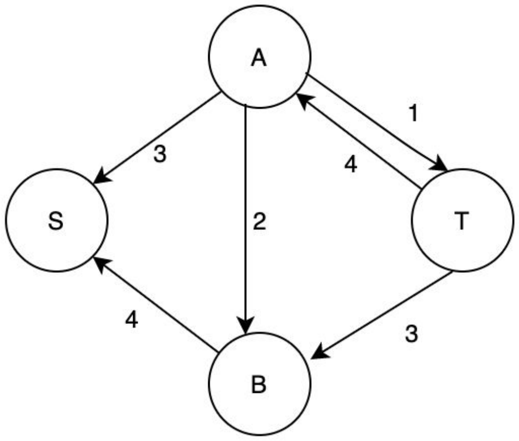
- Iteration 2:
  path : $S \text{->} A \text{->} T$ *Bottleneck* = 2 *Flow* = 1 + 2 = 3



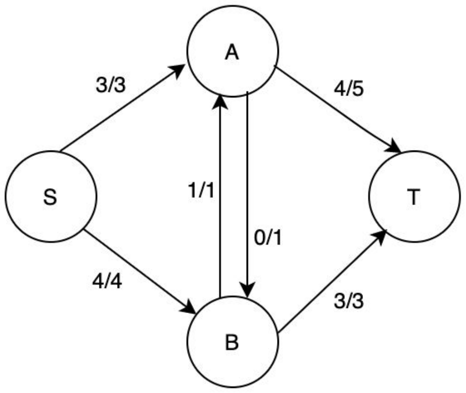
- Iteration 3:
  $path: \ S -> \ B \ -> T \quad Bottleneck = 2 \ \ Flow = 3 + 2 = 5$



- Iteration 4:
  $path: \ S -> \ B \ -> A -> T \quad Bottleneck = 2 \ \ Flow = 5 + 2 = 7$



Final Residual graph:

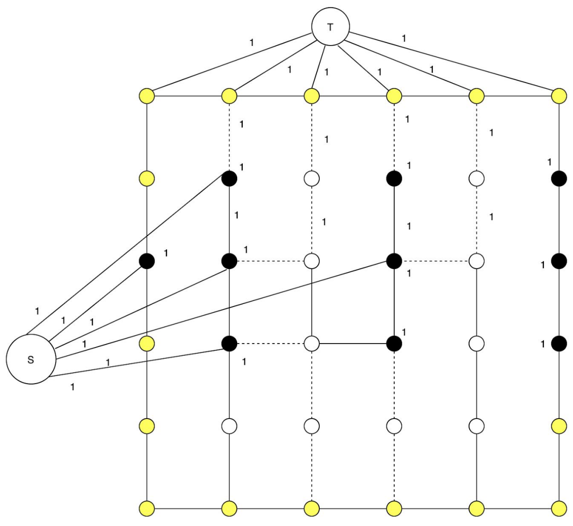


Max Flow: 7
Min cuts:
There are 2 S-T min cuts:

1. A={S} B={A,B,T}
2. A={S,B} B={A,T}

2. Escape from the building

Solution:
Given: The positions of **p** persons located in the building having (x_1, y_1) ……(x_p, y_p)
To Do: To decide whether there exists p different vertex disjoint paths from their starting position to any p different points on the boundary of the grid.
Constraint: There must be no intersection between paths of any two persons.
Based on the requirements of the question we have the below network flow graph created:



Description of the the network flow graph and Reduction:
The nodes in yellow are the boundary nodes that are located on the boundary of the grid. Considering the black darkened nodes are the positions of **p** persons, this problem is reduced to a problem of finding maximum network flow in a graph given that there are **p** disjoint paths. We first create a new source node S and a new target node T, and we have assumed capacity 1 on each position of a person. We also place a bidirectional edge between adjacent pairs of vertices in the grid and assign a capacity of 1 to it.

We also create two nodes for every given position of a person and assign a capacity 1 on the edge between two newly created nodes for each person. We also connect the target to all yellow nodes (boundary node) and assign a weight of 1 on the edge connecting them and attach all the given persons in the building (the darkened nodes which are the positions of the persons in the building) to the source and assign a capacity of 1 on the edges connecting them.

Now that we have constructed our new network flow graph by adding the new set of vertices and edges from the reduction we can run the below algorithm to decide if there are **p** edge disjoint paths. Hence, if we could get a feasible solution of finding a maximum flow to our reduced graph then that would mean that we have a feasible solution for the original problem.

```
Find_feasible_evacuation(G(V,E)):
    Run Edmonds Karp algorithm on the above graph and get all S->T paths (Only those paths will be
    chosen wherein flow is not zero) and retrieve the max flow and the augmenting paths.
```

Time Complexity: The above algorithm is polynomial as we run Edmonds karp on the new graph with V new vertices and E new edges for each person node and we also attached edges from target to boundary nodes and P new edges from position nodes to source, hence adding up all the new vertices and edges to find the augmenting paths and flow and overall we get $O(V.E^2)$ where V = p.

Proof of correctness:
Claim: There exists *p* different vertex disjoint paths from their starting position to any p different points on the boundary of the grid iff max flow is equal to *p.*
Proof:
(=>) Suppose there are p vertex disjoint paths, we prove that max flow = p.
As we have a placed unit capacity on the edge connecting the nodes of a person(i){after breaking a person into two nodes}, we see that all augmenting paths have unit capacity each and hence if f(e)=1 for an edge only then will it participate in some path or else if f(e)=0 then we don't consider that edge. In this way we get all paths who are edge disjoint paths which implicitly means from our graph that we have vertex disjoint paths and need to have a max flow of value *p* for such a case. It cannot be lesser than *p* because or else we could have constructed a flow of value *p* from the existing list of disjoint paths and it can't be greater than *p as* well, as we have pushed all the flow possible.
(<=) Suppose we are given that max flow=p, we prove that there are p vertex disjoint paths.
We can prove this by inducting on the number of edges with *f(e)=1*. We know from conservation law that if there exists any edge(s,u) with flow=1 then there exists edge(u,v) with flow=1. Eventually after taking such a *s-t* path the flow decreases to *f'=p-1* and we have fewer edges to consider. When we apply an inductive hypothesis on the new value of flow, we get *(p-1)* edge disjoint paths for a flow f'. Hence, given p is the max flow, then there are p vertex disjoint paths.

3. Install Software to your new Computer

Solution:

Given: There **n** products from two companies Microsoft and Apple. For each product *i,* we have,

- the price $p_i \geq 0$ that Microsoft charges and the $p'_i \geq 0$ that Apple charges.
- the quality $q_i \geq 0$ of Microsoft version and the quality $q'_i \geq 0$ of the Apple version.

For each product pair we have penalty,

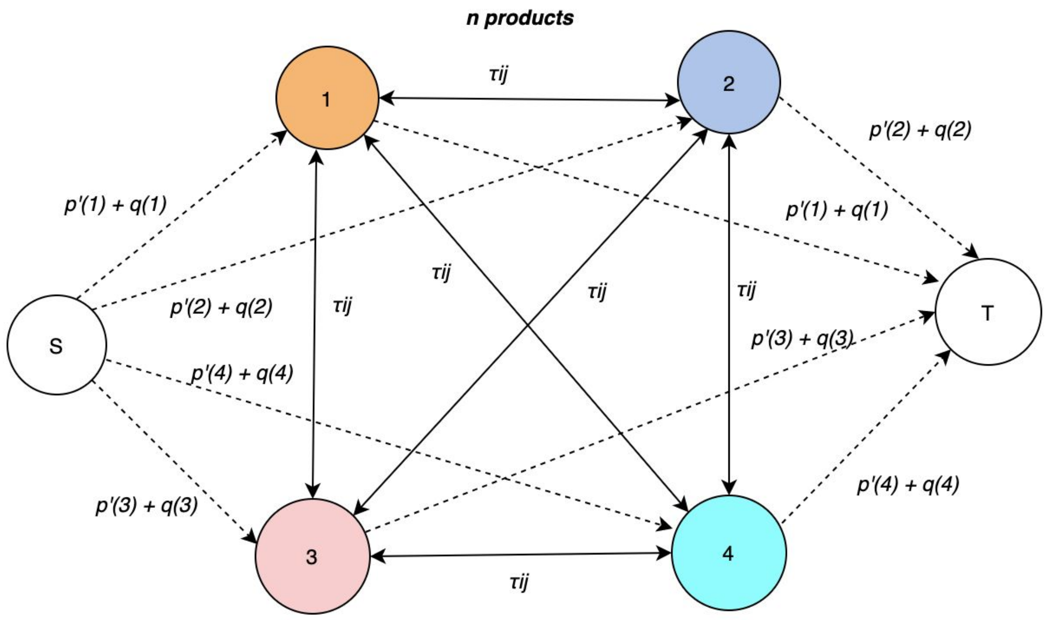
$\tau ij \geq 0$

If two products are from the same company then, $\tau ij = \tau ji = 0$

Else, we have a penalty of τij or τji based on which product is brought from which company.

To Do: Decide which product to purchase from which of the two companies to maximize the total system quality(including penalties) minus the total price.

Based on the requirements of the question we have the below network flow graph created:



Description of the the network flow graph and Reduction:

The above problem can be reduced to a network flow problem by creating a source and a target node called *S* and *T* respectively. We assign a capacity of τij on the edge connecting two products if we buy a product *i* from Microsoft and product *j* from Apple; else we assign a capacity of τji on the edge connecting two products if we buy a product *j* from Microsoft and product *i* from Apple {Basically edges connecting the product pairs are penalties}.The construction of the above network flow graph is based on the fact that quality can be greater than a product and hence become negative.

The task of deciding which product to buy from which company can be solved using the above reduction to a network flow graph where we go about finding the min-cut for the graph in order to separate the products into two sets which tell us which product can be brought from which company.

The problem asks us to maximize total system quality minus price, and in order to solve the problem we initially first convert this maximization problem to a minimization problem, wherein we minimize $price - quality + penalty$ of a product in order to attain the overall maximization of quality minus price of a product. In order to find the min cut, we first find max flow subject to the new minimization constraint.

Hence, in accordance to the new minimization we add edges from source to every node and target to every node with weight price-quality. Now that we have constructed our new network flow graph with a new set of vertices and edges we can run the below algorithm to decide which products to buy from which company and hence, if we could get a feasible solution of finding max flow to our reduced graph of maximum network flow then that would mean that we have a feasible solution for the original problem.

find_product_company(G(V,E)):

1. Run Edmonds Karp algorithm on the above network flow graph.
2. Find the Min-cuts.

In the above graph, say we get one set with products 1 & 3 and another set with products 2 & 4.
What this would essentially mean is that based on the edges that are crossing then min cut we categorize products into two companies.

Time Complexity: The algorithm has polynomial time complexity as we run Edmonds karp to find the max flow from which we can get the min cuts necessary to categorize the products into 2 companies.
We create a complete graph with $N^2 + 2V$ edges and V+2 vertices. Hence, the time complexity is $O(V.E^2)$ which comes down to $O(n^5)$.

Proof of Correctness:
Claim: The n products can be divided into two sets subject to maximization constraint ($quality(including\ penalty) - price$)) iff $max\,flow = \sum_{i=1}^{n}[price - quality + penalty]$

Proof:
(=>)
Suppose that n products can be categorized into sets of two companies, then we need to prove:
$max\,flow = \sum_{i=1}^{n}[price - quality + penalty]$
Based on the fact that we know which products come in which company we know the edges crossing the min cut are the ones that are saturated and hence in our case we have product 1 and product 3 in Apple and product 2 and 4 in Microsoft. Then, proving by construction we know that max flow=min cut value and hence we get max flow = $\sum_{i=1}^{n}[price - quality + penalty]$ which is thereby solving our minimization problem and hence solves the problem of maximizing quality(including penalty)-price.

(<=)

Given that max flow is $max\,flow = \sum_{i=1}^{n}[price - quality + penalty]$ we now have to prove that products can be categorized into sets of companies. As we know the necessary max flow value is achieved only when all the edges crossing cut have total summation value as $\sum_{i=1}^{n}[price - quality + penalty]$ which is actually the max flow value hence proving that the products can be categorized into two sets of companies.

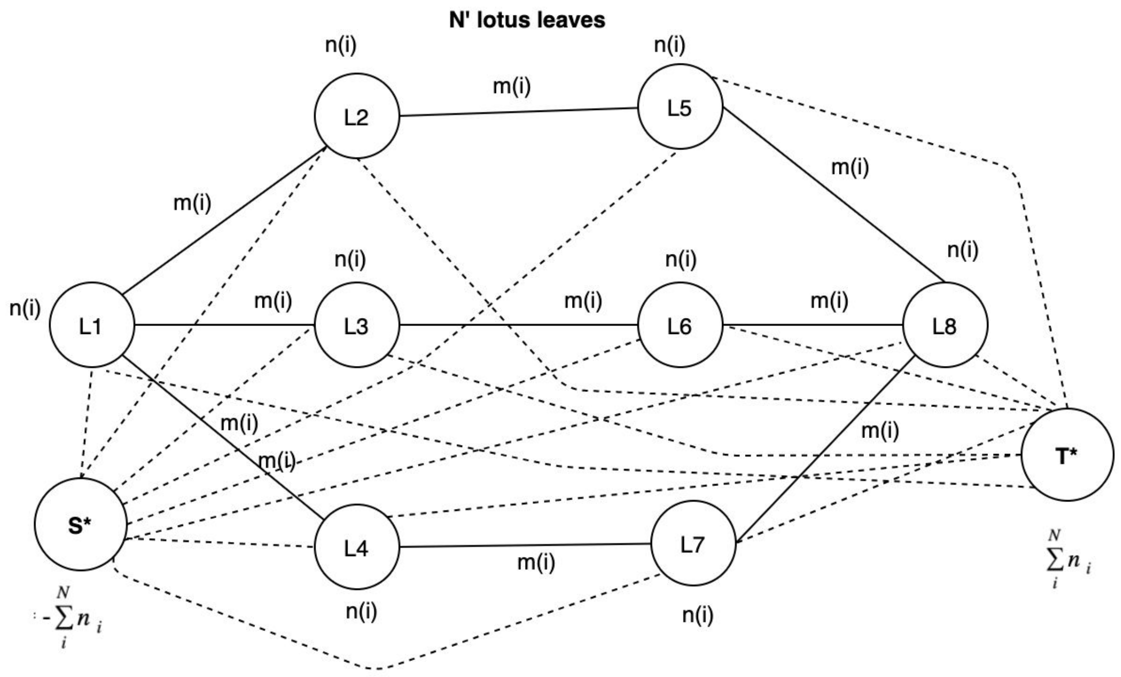
4. Jumping Frogs

Solution:

Given: There are N lotus leaves with positions (x_i, y_i), n_i - number of frogs for leaf *i,*
m_i − *number of jumps for the leaf , distance between two leaves,* $|x_i - x_j| + |y_i - y_j|$ *and jump distance = d*

To Do: To determine whether each leaf can hold $\sum_i^N n_i$ for a party.

Based on the requirements of the question we have the below network flow graph created:



Description of the the network flow graph and Reduction:

The above problem can be reduced to a network flow problem from a circulation problem which has below conditions:

1. For each e connecting leaves we have m_i *as its capacity.*
2. For each leaf node we have demand $d_v = n_i$ where n_i = frogs for that leaf node.

We connect S* to all leaf nodes and add a capacity of $-m_i$ on it and add m_i as edge capacity on edges from T* to leaf nodes

Initially we check if the sum of all supplies equals the sum of all demands.

We then connect only those leaf nodes for which $|x_i - x_j| + |y_i - y_j| = d,$ as frogs can't jump beyond distance *d.*

We then assign a edge capacity of m_i connecting leaves for which the above distance criterion matches. Thereby we now get a new value for the total number of leaves as N'.
On each leaf node we attach a demand (d_v) $= n_i$ and connect source S* to all leaf nodes and connect target node T* to all leaf nodes and the demand on them would be $-\sum_i^N n_i$ as finally if we meet the demand on N we can prove then leaf i can host $\sum_i^N n_i$ frogs for the party. The circulation problem is now reduced to a network flow problem as it is now reduced to finding a S->T path wherein we need to check for each leaf the value of flow that it gets is $\sum_i^N n_i$ or not and output yes if it is possible else no.
Now that we have constructed our new network flow graph with a new set of vertices and edges we can run the below algorithm to decide if each leaf can host all frogs for the party or not. Hence, if we could get a feasible solution of finding maximum flow in our reduced graph then that would mean that we have a feasible solution for the original problem.

```
find_if_leaf_can_host_a_party(G(V,E)):
    Let output[0….N']
    Let total = $\sum_i^N n_i$
    for leaf i=1 to N':
        Run Edmonds karp algorithm on the above network flow graph and get the max flow.
        if max flow = total :
            output[i]="Yes"
        Else:
            output[i]="No"
```

Time Complexity: We construct a new graph with reduced edges based on the distance criterion and add 2 new vertices each connecting upto N edges in the worst case giving rise to 2N edges.
The above algorithm has polynomial time complexity as we run Edmonds Karp for a maximum of N leaves in worst case for which we get $O(N.(V.E^2))$

Proof of Correctness:
Claim: There is a feasible circulation *f* with demands $\{d_v\}$ in G iff maximum s*-t* flow in the graph has value $= \sum_i^N n_i$ in turn implying that a leaf can hold all frogs and host a party with all frogs if there is a feasible circulation.
(=>) Given that we we have a feasible circulation in our graph G, we need to prove that max flow$=\sum_i^N n_i$
Since we know that there is a feasible circulation, we can tell that there cannot be s*-t* flow greater than $\sum_i^N n_i$ as the cut(A,B) has only A={s*} and capacity = $\sum_i^N n_i$. Now given there is a feasible circulation f with demand d_v in G, then sending a flow value of $-d_v$ (number of frogs on leaf *i*)on each edge (S*, l_1) and a

flow value d_v on each edge say (l_8 ,T*), we obtain a S*-T* flow of $\sum_i^N n_i$ which in fact is the maximum flow. Thereby proving that if a leaf can have a feasible circulation its max flow must be $\sum_i^N n_i$.

(=>) Given that we have a max flow=$\sum_i^N n_i$, we need to prove that there is a feasible circulation ie. leaf can hold a party if max flow=$\sum_i^N n_i$. Now it must be the case that every edge out of S* and every edge into T*, is completely saturated with flow. Thus, on deleting those edges, we obtain a circulation f in the graph

G with flow_in(v)-flow_out(v)=d_v for each leaf node which is the number of frogs on that leaf, proving that there is in fact a feasible circulation f given that max flow =$\sum_i^N n_i$. Thereby, if we have max flow=$\sum_i^N n_i$ then there is a feasible circulation ie. a leaf can hold a party with all frogs.

5. **Preparing for the Exams**

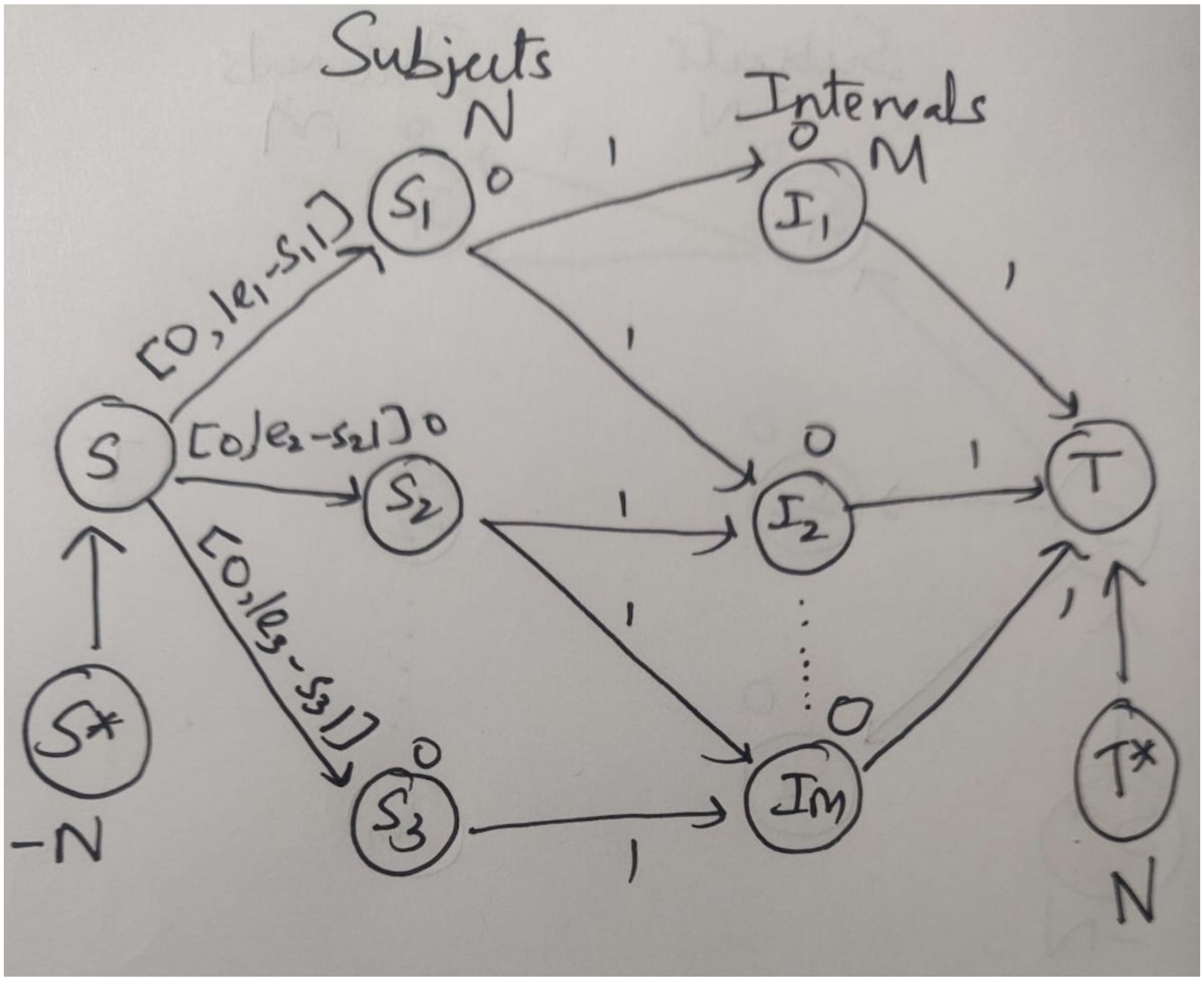
Solution:

Given: N subjects, N (s(i),f(i)) intervals where s(i)=start time when he gets all materials and e(i) when all the materials expire.

To Do: Find the maximum time for the least prepared subject.

Constraint: Leo can study only one subject at one time unit and the grade he receives would be based on the what is the minimum time he spends for a subject.

Based on the requirements of the question, we have created the below Circulation with lower demands problem,



Description of the the network flow graph and Reduction:
Here we have created interval nodes of 1 hour each and have assigned demand=0 on it as it possible that Leo could not dedicate any time on any subject at all. Subject nodes are connected to Interval nodes with capacity=1 if it falls in that interval range. Source node is attached to every subject N with edge capacity in terms of $(0, |e(i) - s(i)|)$ as we know that the interval times start with 0. A super source node S* is attached to Source node N with demand=-N as it supplies all subjects. And super target node T* that is attached to target T with demand=+N
This network flow can be reduced to finding max flow ie. maximum network flow problem by finding S*-T* path.
Based on the construction of the new graph with all the above edges and vertices, we run the below algorithm to get the maximum time for the least prepared subject.

```
find_maximum_time_for_least_prepared(G(V,E)):
    Run Edmonds karp algorithm and find the max flow on the above network flow graph.
```

Once we know the max flow say *k* based on the above algorithm we know that that is the minimum amount of time dedicated to each subject for studying hence *kN* would be the score that Leo would eventually receive.
To see this in another aspect. we are maximizing the time in such a way that Leo covers all subjects through the above network flow graph with minimum time possible across each interval. Once we receive the max flow we will now have the demands on each subject that will tell us how many subjects Leo studies in that interval based on the connection of that subject with an interval. Hence, based on the demand on each interval we know how many subjects can be studied in that interval. Now, we greedily choose the subjects and minimum time in such a way that leo can hence study all subjects. Now, the max flow is essentially what is the maximum time needed for the least prepared subject. So here, we are essentially reducing the problem from a greedy approach to maximum network flow approach

Time complexity: The above algorithm is polynomial in nature as we run edmonds karp to get max flow which is $O(V.E^2)$ overall but other small terms of addition of new vertices and edges don't add up to the complexity. There are also new V as we split the intervals of one hour each.

Proof of correctness:
Claim: If there is a feasible solution of value k for the network flow problem which is the max flow then that is the maximum time leo spends on the least prepared subject.
(=>)
Given that we know the maximum time for the least prepared subject, then based on the graph we created we run our algorithm to find max flow. The max flow value is in fact equal to k as that is the minimum time needed and we try to maximize that over all subjects thereby leading to retrieving the maximum time needed for the least prepared subject as the score that he retrieves is also kN which hence tells us that if a flow such that the one in our graph is passed through we get the maximum time needed for the least prepared subject.

(<=)
Given that we know that the max flow is kN then to prove there is a feasible solution of value k, we know that if max flow is k then that is in fact the minimum time spent by Leo across all subjects and hence which in fact gives the maximum time needed for the least prepared subject. As we associate that minimum time over all subjects and maximize it over all subjects.

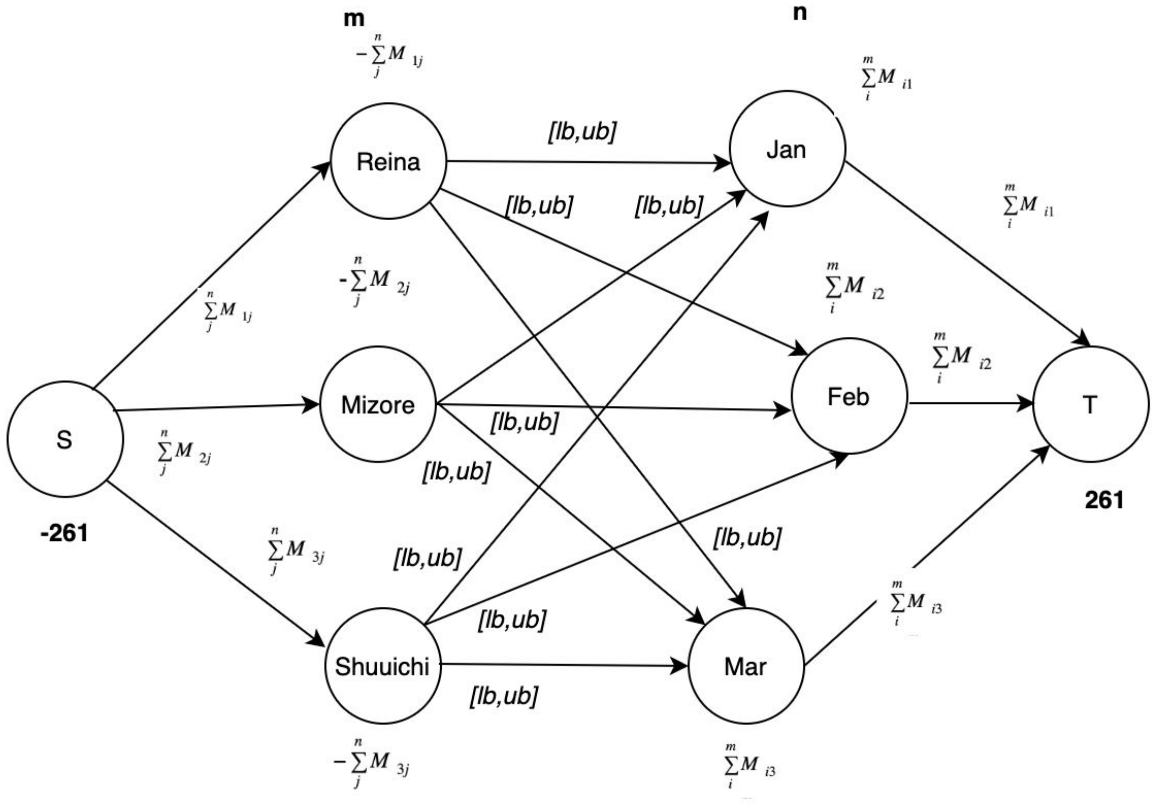
6. **Help Kumiko**

Solution:

Given: m members, n months, a Data matrix - $\{M_{ij}\}_{(i,j)=(1,1) \text{ to } (m,n)}$

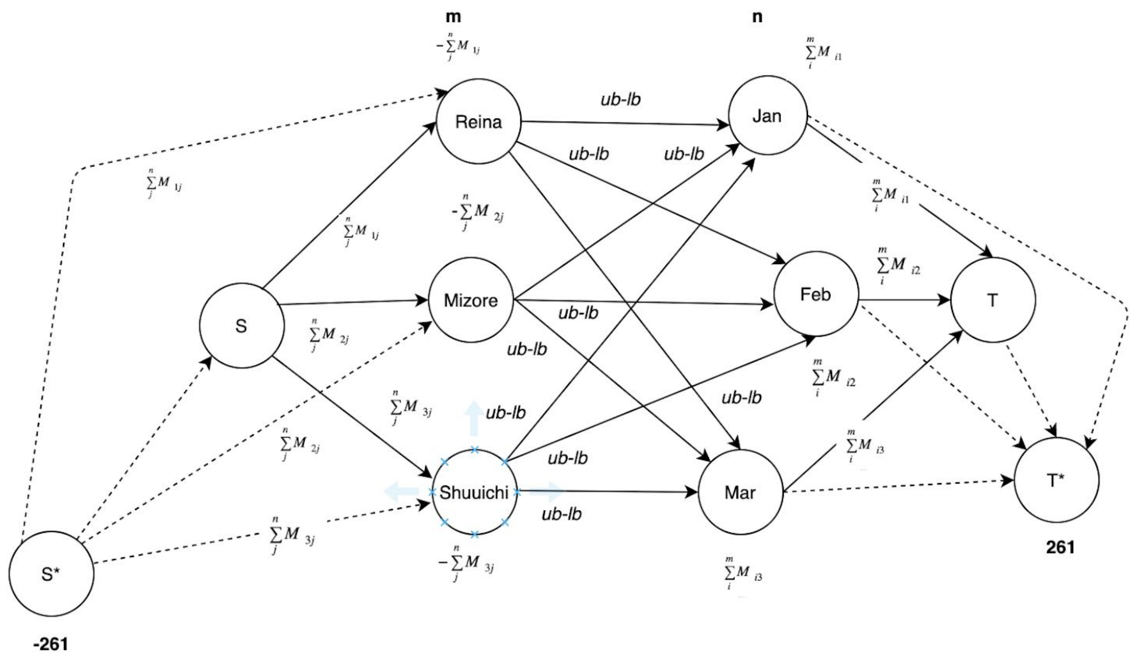
To Do: To obtain a rounding for the community fee.

Based on the requirements of the question, the problem is a Circulation with Lower bounds problem as depicted below(G):



Description of the the network flow graph and Reduction:

The above graph has the following Circulation with Lower bounds problem has the below conditions:
For each edge connecting source to a member is basically the sum total of what the member pays over all n months. Each edge connecting each of the n months to Target is the sum total amount paid by m members for that particular month n. Each edge connecting the member to a month is in the form of [lb,ub] where lb is the rounded down value of the amount the member pays towards that month and ub is the rounded up value of the amount the member pays towards that month. We similarly add one more node April in our graph and connect it in the same way, in order to get a total supply of -261 at S and 261 at T.
The Circulation with Lower bounds problem can be further reduced to a maximum-network flow problem with no lower bounds by introducing two new nodes S* and T* and edges with intervals in the form of [lb,ub] as ub-lb.
Below is illustration of the above reduction(G'):



Now that we have constructed our new network flow graph with a new set of vertices and edges we can run the below algorithm to decide if such a rounding is possible or not and hence, if we could get a feasible solution to our reduced graph of finding maximum flow then that would mean that we have a feasible solution for the original problem of finding a rounding.

```
obtain_rounding(G'(V,E)):
    Total=0
    Run edmonds karp algorithm and find max flow for member i.
    total=maxflow
    if (total == $\sum_{(i,j)}^{(m,n)} M_{ij}$ ):
        print "rounding possible"
    else:
        print "rounding not possible"
```

Time Complexity: The above algorithm is polynomial in nature as we run Edmonds karp and hence the overall complexity comes to $O(V.E^2)$. The total number of vertices,V is m+n+4 and total number of edges,E is m*n+2m+2n+2.

Proof of correctness:

Claim: There is a feasible circulation in G iff there is a feasible circulation in G' ie. if there is a feasible circulation in G means that a rounding is possible and such a rounding is only possible when max flow= $\sum_{(i,j)}^{(m,n)} M_{ij}$

Proof:

(=>) Given that there is a feasible circulation(a rounding exists) in G, we can define a circulation f' in G' by

$f'(e) = f(e) - l_e$. Then f' satisfies the capacity conditions in G' and

$f'^{in}(v) - f'^{out}(v) = \sum_{e \text{ into } v} (f(e) - l_e) - \sum_{e \text{ out of } v} (f(e) - l_e = d_v - L_v$ thereby we have satisfied the demand conditions in G' as well meaning that we are able to match the demand of $\sum_{(i,j)}^{(m,n)} M_{ij}$ as there exists a feasible circulation in the G' and hence has a max flow = $\sum_{(i,j)}^{(m,n)} M_{ij}$ ie. we are able to produce the necessary rounding for the given data.

(<=)

Given that there is a feasible circulation f' in G' , we define a circulation f in G by:

$f(e) = f'(e) + l_e$. Then, f satisfies the capacity conditions in G, and

$f'^{in}(v) - f'^{out}(v) = \sum_{e \text{ into } v} (f'(e) + l_e) - \sum_{e \text{ out of } v} (f'(e) + l_e = d_v - L_v + L_v = d_v$. Hence, we are also able to satisfy the demand conditions in G as well meaning that we are able to get a max flow of $\sum_{(i,j)}^{(m,n)} M_{ij}$ in G which is proving that we have a feasible circulation or such a rounding exists in G as well as are able to produce the same overall total as $\sum_{(i,j)}^{(m,n)} M_{ij}$ means a feasible rounding exists in G' as well.

7. **Edges that increase max flow**

Solution:
Given: G(V,E), sources-sink pair(s,t) and capacity of edges $\{c_e \geq 0 \mid e \in E\}$
To Do: give a polynomial time algorithm to find a set of edges S, such that for every edge $e \in S$, increasing c_e will lead to an increase of max-flow value between s and t.

We model this problem as a network flow problem and solve it.
In order to find the set of those edges that would lead to increase in flown when their capacity is increased we run the below algorithm:

1. We run Edmonds Karp algorithm on our *G(V,E)* to get max flow and further to get min-cut (i.e. those edges on the min cut).
2. For each edge*(u,v) in min-cut* :
    a. We run traversal and find a path from source to *u* and a path from *t to v.*
    b. Once we have paths from s-u and t-v we take the bottleneck value and check if connects those two paths with the min cut edge and increase the capacity of that edge*(u,v)* by a number greater than 0 say the bottleneck value of that path and check if the flow increases if so we then add *edge(u,v)* it to set S.

Time complexity: The time complexity of the above algorithm is polynomial in nature as we use Edmonds karp's $O(V.E^2)$ to get our set of min-cut edges and then, for every edge in the min-cut we run traversal($O(V + E)$ two times from s-u and t-v and then the check of whether increasing the capacity over that edge leads to increase in flow takes linear time of adding c_e to the bottleneck value of that path. Overall it's the same as Edmonds karp which is polynomial in nature.

Proof of correctness:
Claim: Flow can be increased if capacity of min-cut edges is increased.
(=>)
Given that flow is increased, we need to prove that it was due to an increase in capacity of edges on the min cut. As we know that after running Edmonds karp algorithm on our graph we find the max flow and also hence find the min-cut. The edges on the min-cut are the ones that are completely saturated and hence, more flow can be passed through by increasing the capacity over this min-cut edge. No other edge on increasing its capacity can lead to an increase of max flow.

(<=)
Given that min-cut edges are increased, we need to prove that it leads to increase in flow. We know that min cut edges are the ones that are completely saturated hence, by increasing capacity over this edge we get an increase of maximum flow because the rest of the edges are not completely saturated and hence will not lead to increase in max flow.