Contents

Chapter 1

Network Flow

1.1 Lecture 1

Bipartite Matching

It is a particular problem, Both DP and greedy don't work for this. The formal Defintion of **Macthing** is "Set of Edges which are vertex disjont". i.e., no two edges have a common vertex. The last question of midsem was actually a special case of matching problem. Inputs are given as intervals (1, 3), (2, 4), (1, 2), (1, 2). This means the first person can map to 1,2,3 and second person to 2,3,4 etc. This is a special case and greedy idea works

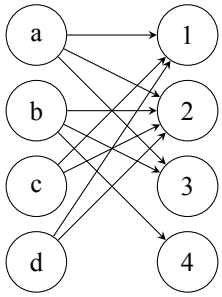


Figure 1.1: Bipartite Matching Graph

here because the possible matching of each job a,b,c,d is continous. With the prices given for each job, we try to maximize total price The answer is given as follows

Algorithm 1: Is_Schedulable

Data: list of Intervals S
Result: YES if we can assign distinct days to all intervals in S

```
Sort S with increasing order of ending day;
for each interval in S do
    assign first available day;
    if there is no available day then
        return NO;
return YES;
```

Algorithm 2: Max_Price

Data: list of Intervals L
Result: optimal list of intervals S

```
Sort L in decreasing order of prices S = φ for r ∈ L do
    if Is_Schedulable(S + r) then
        S = S + r
return S;
```

This completes the solution and Let us come back to defining Network flow more formally

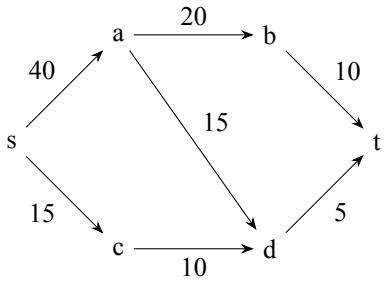
Network FLow

There are many examples such as Traffic flow (roads form edges, vehicles flow), Electric Network(each wire is an edge, current is flow), Water flow Network etc. One of the problems in such construction, is to find Maximum flow we can get. (Another is Path with Shortest no of edges)

Max flow problem

Inputs:

- Directed Graph G(V, E)
- Capacities on edges $\{C_e : e \in E\}$
- A source vertex **s**
- A sink vertec **t**

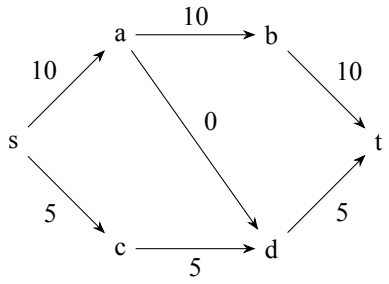


Max flow in this case is 15

Some Network flows such as water Network donot have perfect direction, which are modeled using Undirected graphs

- **Definition of flow of a Network**: Set of flows over each edge. This can be seen as a graph with set of vertices same as original one

$$\{f_e : e \in E\} \quad \forall \text{ e} \in \text{E}, 0 \leq \text{f}_e \leq \text{C}_e$$



Flow of before example

- **Flow Conservation Constraint**: This is the main Constraint of this optimization problem

$$\text{Let } f^{in}(v) = \sum_{e \in in(v)} f_e \text{ and } f^{out}(v) = \sum_{e \in out(v)} f_e$$

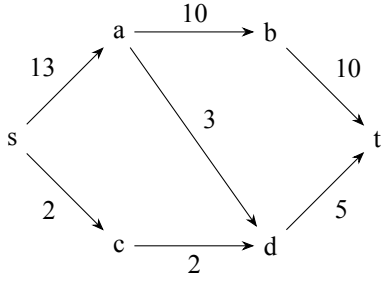
This constraint states that $f^{in}(v) = f^{out}(v) \quad \forall$ v ∈ V, v ≠ s, t
(This is very similar to Kirchhoff's First Law or Current law in Electrostatics)

- **Objective** : Max Flow

$$\sum_{e \in out(S)} f_e$$

- We assume that No incoming edge to S and No outgoing edge from t

Another valid flow for before problem is

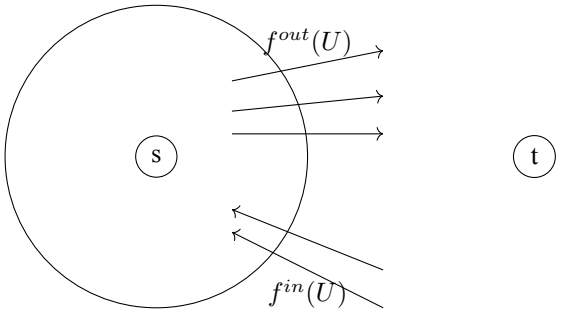


Another valid flow

From above observations, Let us infer some Claims

- **Claim 1** : $f^{out}(s) = f^{in}(t)$
- **Claim 2** : For any U $\subseteq$ V, such that s $\in$ and t $\notin$ U

$$f^{out}(U) - f^{in}(U) = f^{out}(s) \quad \text{where } f^{out}(U) = \sum_{e \in out(U)} f_e \text{ and } f^{in}(U) = \sum_{e \in in(U)} f_e$$



→ **Proof for Claim 2**: From flow conservation, we know that

$$f^{out}(v) = f^{in}(v) \text{ for any v} \neq s \implies \sum_{v \in U} f^{out}(v) - \sum_{v \in U} f^{in}(v) = f^{out}(s) - f^{in}(s)$$

$$\implies \sum_{v \in U} f^{out}(v) - \sum_{v \in U} f^{in}(v) = f^{out}(s)$$

Now instead of counting over vertices, Let us count over edges. In the edges is completely inside U i.e., both of its edges (say (a,b)) are in U, then f_e over this edge is present in both the sums as f^{out}(a) and f^{in}(b) respectively. So in the LHS we will be left with only outgoing and incoming edges into U

$$\implies \sum_{v \in U} f^{out}(v) - \sum_{v \in U} f^{in}(v) = \sum_{e \in out(U)} f_e - \sum_{e \in in(U)} f_e$$

$$\implies \sum_{e \in out(U)} f_e - \sum_{e \in in(U)} f_e = f^{out}(s)$$

→ **Proof for Claim 1**: Assume U has all vertices except t, then

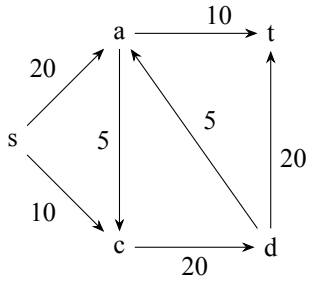
$$f^{out}(U) = f^{in}(t) \text{ and } f^{in}(U) = f^{out}(t) \text{ (which is 0)}$$

Hence f^{in} (t) = $f^{out}(s)$ from claim 2

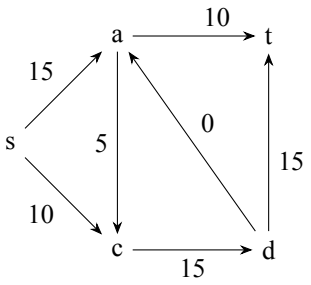
Going back to first example, how can we say that 15 is Max flow? $\sum_{e \in in(t)} C_e$ is 15, and it is the maximum flow into t. So 15 is an upper bound and we have shown an example of flow 15.
Hence 15 is Max flow. Similarly $\sum_{e \in out(s)} C_e$ is also a natural upper bound. Are there any other upper bounds? We will see them in next lecture

1.2 Lecture 2

Let us start with an example, the left figure shows Capacities. Both s, t atmost 30 flow. Is 30 possible? The right figure is a flow network with 25 flow. Can we say 25 is maximum flow?



Capacities



Flow Network

Consider an (s,t) cut with U having {s, a}, its $f^{out}(U)$ is 25 (edges a-t, a-c, s-c). We know that

$$f^{out}(U) - f^{in}(U) = f^{out}(s) \implies f^{out}(S) \leq f^{out}(U) \implies f^{out}(S) \leq 25$$

So 15 is an upper bound and we have shown an example of flow 25. Hence 25 is Max flow

- **Definition of Capacity of U**: $\forall$ U $\subseteq$ V such that s $\in$ U and t $\notin$ U

$$Cap(U) = \sum_{e \in out(U)} C_e$$

- **Claim**: $f^{out}(s) \leq Cap(U)$. Proof is from Claim 2 of previous lecture

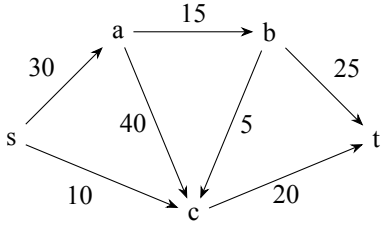
$$f^{out}(s) = f^{out}(U) - f^{in}(U) \quad (1^{st} \text{ term is } Cap(U) \text{ and } 2^{nd} \text{ term is} \geq 0)$$

- **Claim**: Max(s, t) flow value is atmost the out capacity of any (s, t) cut, So
  Max(s, t) flow $\leq$ Out(min(s, t) cut), where min(s,t) cut is min among all possible upper bounds

- **Theorem**: Max(s, t) flow = Out(min(s, t) cut)
  (This is very similar to Dilwarth's theorem, In fact Dilwarth's theorem is special case of This theorem)

Let us consider Max flow problem as a primal linear program, with objective as maximising sum of out-flow as s and constraints being out-flow through s is less than out-flow of each (s, t) cut. The dual problem for this has objective as minimising value of constraint of primal (i.e., out-flow of (s, t) cut). Then above Claim and Theorem are nothing but Weak and Strong duality Theorem respectively.

With this formulaion, let us start designing the algorithm. In the below example consider all possible (s, t) paths and Maximum push throuh these paths (Let us call this as 'Bottle Neck' of respective path).

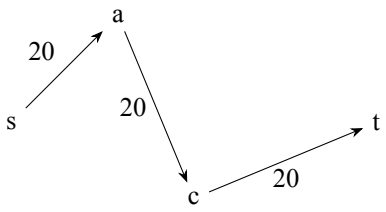


Capacities

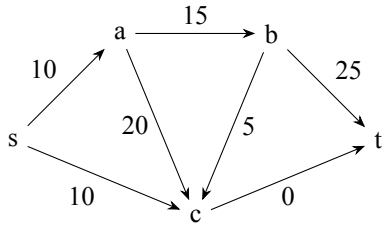
(s, t) path	Bottle neck
{s, a, b, c, t}	5
{s, a, b}	15
{s, a, c}	20
{s, c, t}	10

All (s, t) paths and thier bottle necks

Lets greedily choose the path {s, a, c, t}. Then if we choose {s, c, b} then f_e of s-a is 35, But its C_e is 30. So we should chould choose next (s, t) path by considering the **Residual Capacities**

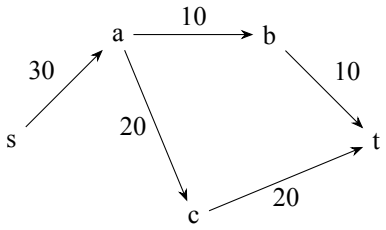


Flow through {s, a, c, t}

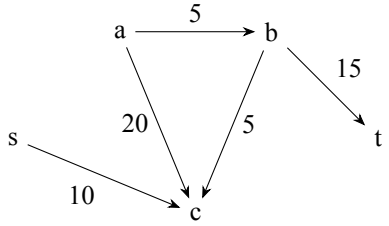


Residual Capacities

We can remove the c-t edge as it is 0 capacity. Now the only left over (s, t) path is {s, a, b, t} with bottle neck 10 Hence the final Flows and Residual capacities are

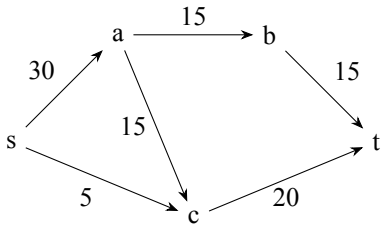


Flow Network

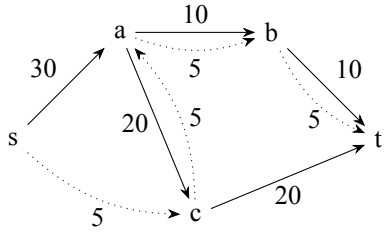


Residual Capacities

Since there is no other (s, t) path can we say that we have reached the Max flow? Consider the below example which has 35 flow. The difference between our flow network and below example is shown in the right figure

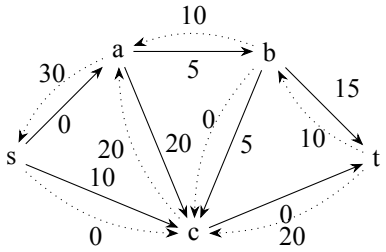


Example with 35 flow

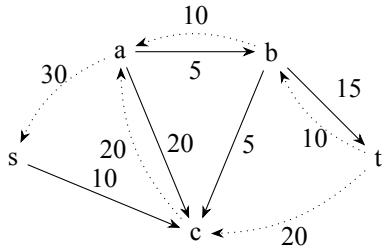


Difference is the dotted path

So we want to allow some backward flow. Let us call this as **pushback**. What is maximum pushback that is possiblw on any edge? Its "Present Flow" f_e. Let us now add the pushback edges to our Residual Capacities and find (s, t) paths in them



Final Residual Graph



Ultra-Final Residual Graph

Now the only Possible (s, t) paths here are {s, c, a, b, t} with bottle neck 5. By adding this path we get the solution in above figure and the residual graph with no (s, t) paths (Verify by drawing :)
Can we say this is optimum? YES. How? By showing an (s, t) cut with Cap(U) = 35. And U = {s, a, c} satisfies this. Hence this is Max flow. The Algorithm for process we have done is

Algorithm 3: Max_Flow

```
1 Start with zero flows
2 while you can do
3 |   Find a path in residual graph
4 |   Push max possible flow along this path
5 |   Update Residual graph
```

1.3 Lecture 3

This algorithm works by selecting any possible (s, t) path at each iteration (Max bottle neck is not necessary), but we'll see an example how selecting a bad (s, t) path leads to increase is no of iterations

Let us define Residual Graph G_f for flow f (Ultra-final one)

- Same set of Nodes as G
- For any edge e = (u, v) $\in$ G
  - Forward edge {e = (u, v)} $C_e = C_e - f_e$ (≥ 0) (Remaining Flow)
  - Backward edge {e′ = (v, u)} $C_{e'} = f_e$ (Pushback)

Before Proving the Correctness and Termination of algorithm, let us more elaborately see what is happening in each iteration of our algorithm

- select any (s, t) path in Residual Graph G_f (say P)
- b = bottle neck (min capacity edge in path)
- For any egde (x, y) $\in$ P
  - if (x, y) is forward edge (present in original) then

    f(x, y) = f(x, y) + b ; (Upadte Flow Network)
  - if (x, y) is backward edge (not present in original) then

    f(x, y) = f(x, y) - b ; (Upadte Flow Network)
- We stop when there is not (s, t) path

Now let us prove that this Algorithm works

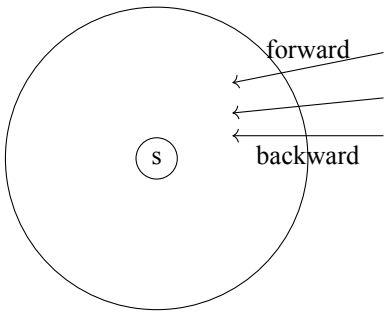
- **Claim 1**: After every update we have a valid flow

  Proof: Assume Flow network before adding this path is valid (Then use induction). By the definition of bottleneck, we still have flow at each edge is less than or equal to its capacity. Similarly definition of backward edge residual on them is less than new flow through its corresponding. By adding any flow, it is added to inflow of one vertex and outflow of another. So Flow conservation also holds.

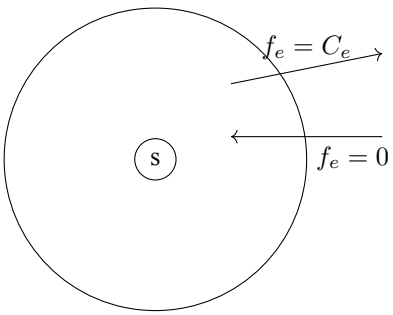
- **Claim 2**: **(Correctness)** If the residual graph G_f has no (s, t) path then f is the Max flow, i.e, if algorithm terminates, then attained flow is the maximum

  Proof: We know that Max flow $\leq$ Cap(U), $\forall$ U $\in$ set of (s, t) cuts. If we can construct a U such that f^{out}(s) = Cap(U), then we can't increase flow further i.e., we have a Max flow (This is similar to Duality of a Linear Program, Natural Upper bound)

  Assume U with set of all reachable vertices (except t) from s.



Residual Flow Graph G_f



Original Flow Graph G

If the Residual graph had an outgoing edge, if the other end is t then contradiction to fact that no (s, t) path, if it isn't t then it is added into U. So Residual graph have no outgoing edge. Coming to Original graph, consider any outgoing edge. This must have its flow equal to its Capacity, else we will have a outward edge in residual graph. Similarly consider any non-zero flow incoming edge, then there will be an

outgoing edge in residual graph. Hence no incoming edges.

$$f^{out}(s) = f^{out}(U) - f^{in}(U) = Cap(U) - 0$$

$$\implies f^{out} = Cap(U)$$

- **Claim 3**: (**Termination**) This algorithm definitely terminates for **Integer** valued capacities. We cannot conclude anything about real values capacities. (In fact there are examples of real capacity Networks for which our algo goes into infinite loop)

  Proof: In each iteration flow increases by a non-zero positive value, by our assumption atleast +1. Hence we reach optimum in finite steps

Running time

In each iteration

1. To Find a path from s to t, we can perform DFS or BFS which is of order O(|V|).
2. To update residual graph, it again take O(|E|) time.
3. We know each iteration increases flow by atleast s. So maximum no of iterations are $F_{max} (= \sum_{e \in out(s)} C_e)$

So the finally (Note that scaling all flows by 1/(LCM of flows) is an optimization)

$$O(|E|F_{max}) = O\left(|E| \sum_{e \in out(s)} C_e\right) = O(|E|(\text{no of edges from s})(log(C_{max}))) \quad \text{log of } C_{max} \text{ because we can deal in bits}$$

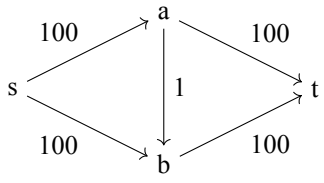
$$= O(|E||V|log(C_{max}))$$

This is psuedo polynomial because it depends on value of capacities rather than only input size.
Strongly Polynomial Time: If algorithm completes in O(Polynomial(|V|)) of arthmetic operations.

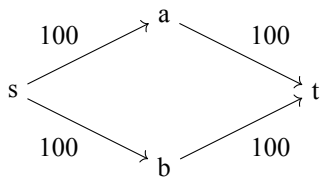
Strongly polynomial time for Max flow problem was acheived by Edmonds Karp by choosing s-t path everytime with fewest number of edges and then proved that it takes only O(|E||V|) iterations. So final order is O($|E|^2|V|$).

1.4 Lecture 4

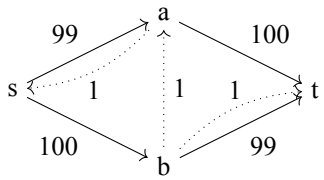
Below example shows how selecting of a random (s, t) can lead to an inefficient run.



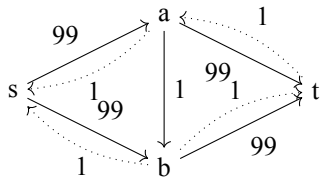
Original Capacity graph



Final Max flow



Residual Graph(1) after choosing (s, a, b, t) as path

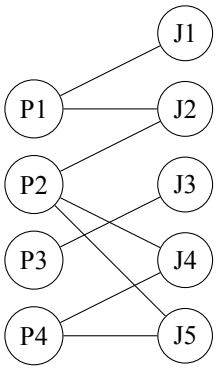


Residual Graph(2) after choosing path (s, b, a, t)

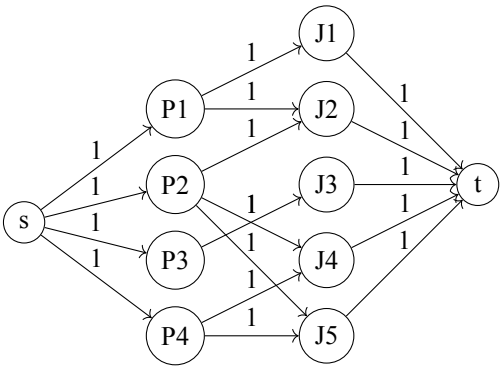
If we choose (s, a, b, t) and (s, b, a, t) again and agian, ech iteration increases flow by exactly 1.
The most efficient way is to choose (s, a, t) and then (s, b, t) which gives Max flow

Applications of Network Flow

1. **Network flow Variants**
   - Multiple sources and sinks with demands (Eg: coal mines (source) and thermal power plants (sinks))
   - Undirected Version (Eg: Water network)
2. **Bipartite Matching**
   - For every edge from P_i to J_i, keep the capacity as 1
   - Create a source node as add an edge from s to all P_i with capacity 1, and create a sink node and add an edge from each J_i to t of capacity 1



Original Bipartite Graph



Converted Network Flow problem

- In the final Flow Graph, if edge P_i J_k is present, then P_i mapped to J_k
- The one-one mapping is ensured because of flow conservation at P_i and J_k. Atmost 1 can come into P_i, so atmost 1 go out of P_i. Similarly, atmost 1 can go out of J_k, so atmost 1 can come into J_k.

Reduction
The above process is called Reduction of one Problem into another Problem. If Problem A is reducable to Problem B, we denote it by A $\leq$ B. In the above case,

$$\text{Bipartite matching} \leq \text{Max FLow}$$

i.e., Bipartite matching is as easy as Max FLow (If polynomial time algo exists for Max flow, we definitely have polynomial algo for Bipartite Matching). Why is it $\leq$ specifically ?
Because Bipartite can be more easier than Max flow, but not harder than Max flow

3. **TA allocation**