Network flow

March 1, 2023

- Input:
  - a directed graph $G(V, E)$
  - edge capacities $\{c_e \in \mathbb{N} : e \in E\}$
  - a source vertex s
  - a sink vertex t

Assumption: No incoming edge to s
No outgoing edge from t.

→ Output: An s-t flow which maximizes the total flow out of s.

Def: An s-t flow is a function

$$f : E \to \mathbb{R}_{\geq 0}$$

which satisfies

- Capacity constraints $0 \leq f(e) \leq c_e$
- Flow conservation

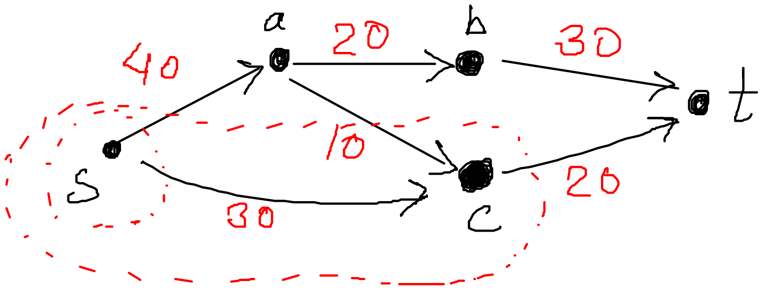
  for any $v \neq s, t$ $\quad f^{out}(v) = f^{in}(v)$

$$\sum_{e \text{ out of } v} f(e) = \sum_{e \text{ into } v} f(e)$$

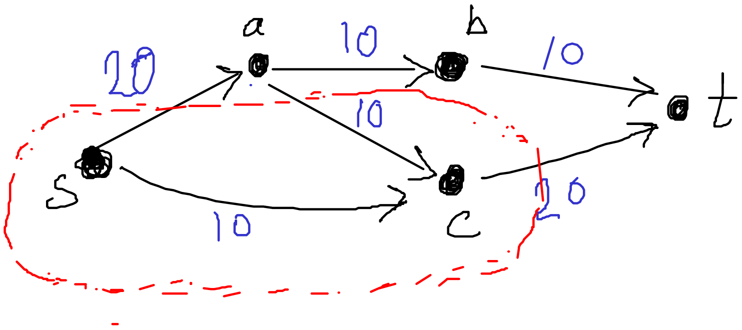
Capacities in red.

Flow values in blue.

Input

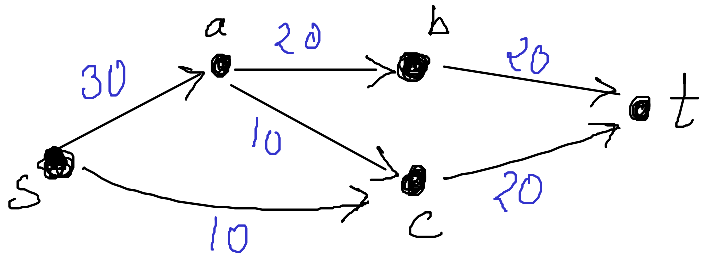


capacities



A flow

Flow value 30



another flow

Flow value 40

Flow value $\displaystyle\sum_{e \text{ out of } s} f(e) = \sum_{e \text{ into } t} f(e)$

Want to maximize flow value.

Claim: Consider a subset $U \subseteq V$ s.t. $s \in U$, but $t \notin U$

Then the net flow out of U is same as the net flow out of s.

$$\sum_{e \text{ out of } U} f(e) - \sum_{e \text{ into } U} f(e) = f^{out}(s)$$

This number is the s-t flow value.

Proof Consider

$$\sum_{u \in U} \left(f^{out}(u) - f^{in}(u) \right) = f^{out}(s) \quad (\text{Conservation})$$

$$= \sum_{u \in U} \left(\sum_{e \text{ out of } u} f(e) - \sum_{e \text{ into } u} f(e) \right)$$

$$= \sum_{e \text{ out of } U} f(e) - \sum_{e \text{ into } U} f(e)$$

Natural upper bound on maximum s-t flow ?

- $\sum_{\substack{e \text{ out} \\ \text{of } s}} c_e$

- $\sum_{\substack{e \text{ into} \\ t}} c_e$

<u>Def</u>: $U \subseteq V$ is an s-t cut if $s \in U$ but $t \notin U$.

<u>Def</u> $Cap(U) := \sum_{\substack{e \text{ out of} \\ U}} c_e$

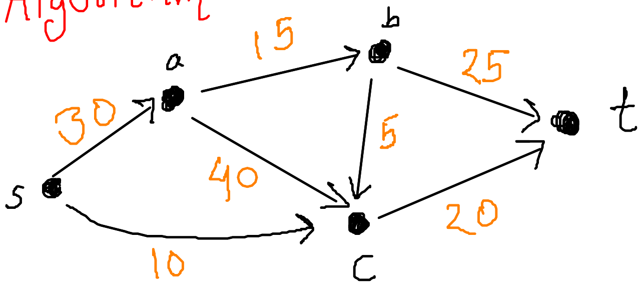
For an s-t cut U, $Cap(U)$ is an upper bound on the flow value.

<u>Theorem</u> Maximum s-t flow $\leqslant$ minimum capacity of an s-t cut

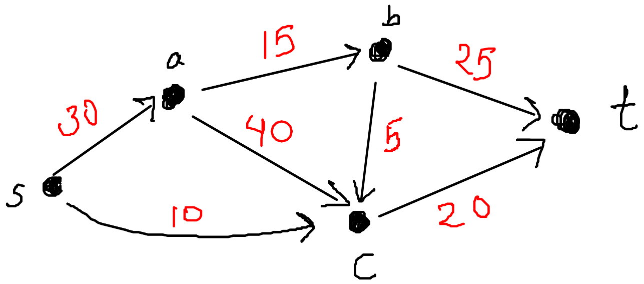
Does the equality always hold ?

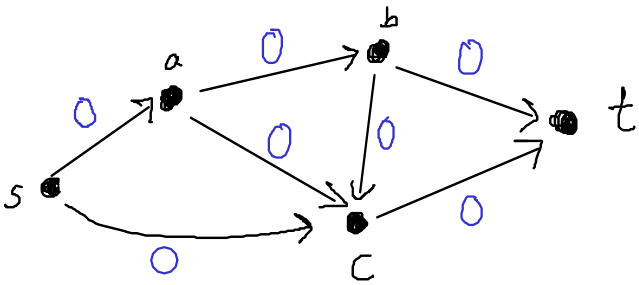
Amazingly, yes !

Algorithm



Idea: Start with zero flow on all the edges.

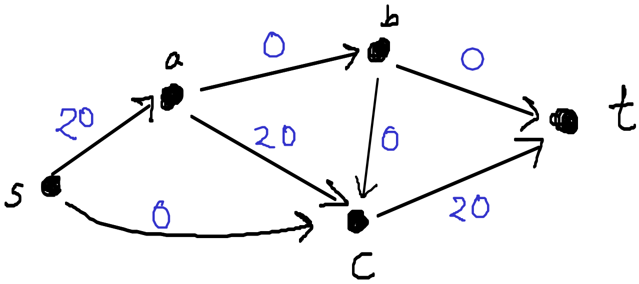




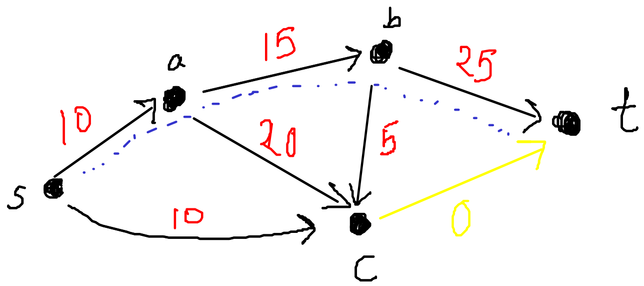
Find an s-t path and push as large flow as possible

Paths	Bottleneck
s - a - b - t	15
s - a - c - t	20
s - c - t	10
s - a - b - c - t	5

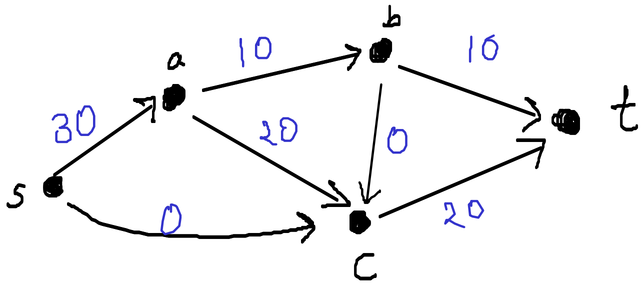
Push a flow of 20 along s-a-c-t



Remove saturated edges
Update capacities.
And then find an s-t path.

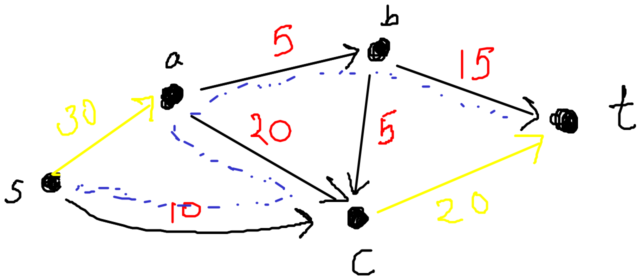


Push a flow of 10 along path s-a-b-t



Remove saturated edges

And then find an s-t path.



No s-t path !!

How can we push more flow? Push back along (a,c)

Def Residual graph w.r.t. a flow f

- Vertex set same.
- For each edge $e = (u,v)$

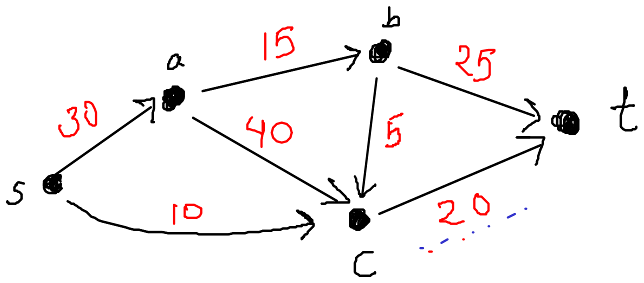
  → Forward edge (u,v) $\quad c_e \leftarrow c_e - f(e)$

  Keep this edge only if positive residual capacity

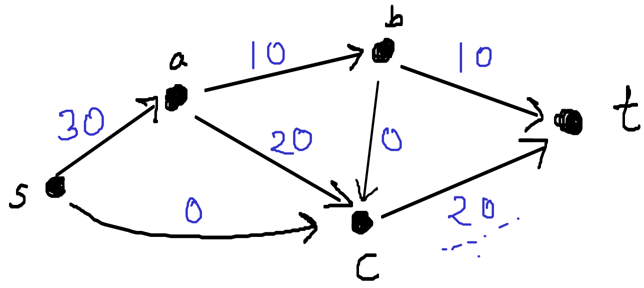
  → Backward edge $e' = (v,u)$ $\quad c_{e'} \leftarrow f(e)$

  keep this edge only if $f(e) > 0$

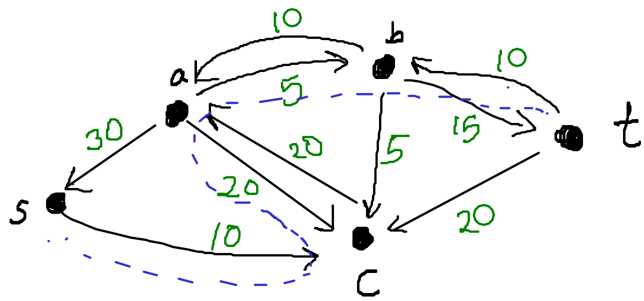
Find a s-t path in the residual graph.



Original capacities

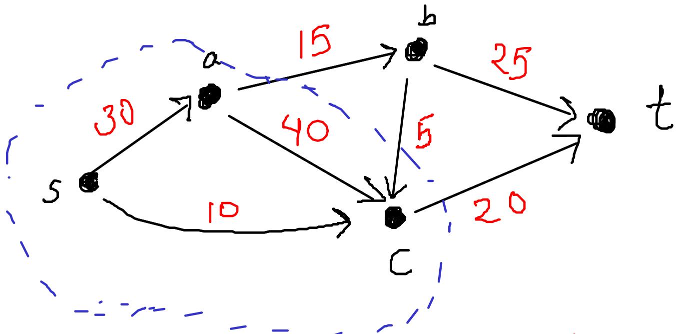


flow f

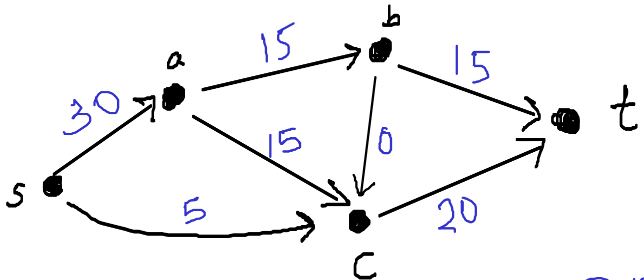


Residual Graph G_f

Path in the residual Graph s-c-a-b-t push 5

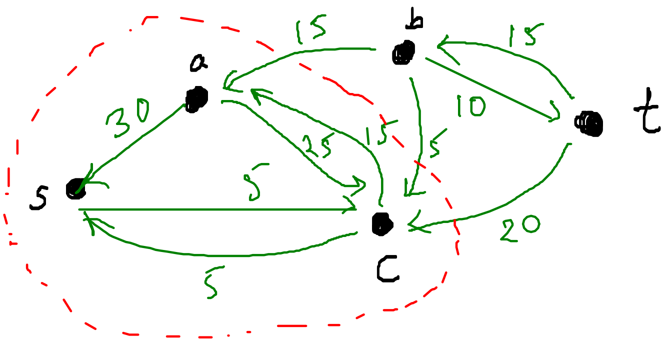


Original capacities



35

flow f



No outgoing edges from $\{s, a, c\}$

Residual Graph G_f

Algorithm

Initialize $f = 0$

While you can {

find an s-t path P in the residual graph G_f

$b \leftarrow$ min capacity of an edge on P.

for each edge $e = (u,v) \in P$ {

if (u,v) is a forward edge

$f(u,v) \leftarrow f(u,v) + \underline{b}$

if (u,v) is a backward edge

$f(v,u) \leftarrow \underline{f(v,u) - b}$

}

Update the residual graph G_f.

}

Claim 1 : After each iteration of the while loop, f remains a valid flow.

Claim 2 : If we cannot find an s-t path then the flow is maximum.

Claim 3: Algorithm always terminates.

Proof of Claim 1 :

Conservation of flow at any vertex v

Case 1

$f^{out}(v) - f^{in}(v)$

Case 2

Capacity constraints.

→ Forward edge

→ Backward edge

Proof of Claim 2

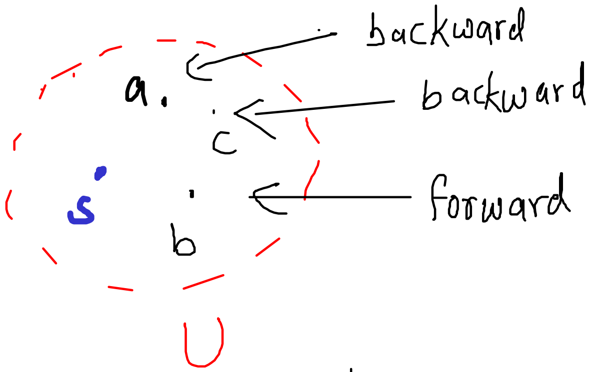
Idea is to show an s-t cut in original graph whose capacity is equal to current flow value.

Residual graph:

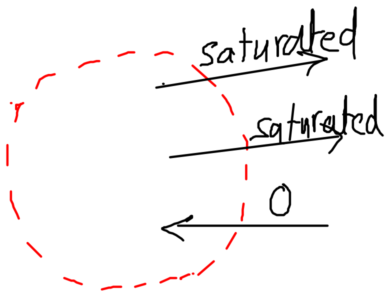
We can't find an s-t path.

$\Rightarrow$ $\exists$ subset $U \subseteq V$ s.t. $s \in U$, $t \notin U$ and there are no outgoing edges from U.

$U \leftarrow$ reachable vertices from s.



Residual Graph.



Flow graph

Obs In the flow graph

Outgoing edges from U are saturated

edge into U have zero flow.

$\rightarrow$ flow value $= f^{out}(U) - f^{in}(U)$

$$= \sum_{\substack{e \text{ out} \\ \text{of } U}} cap(e) - 0$$

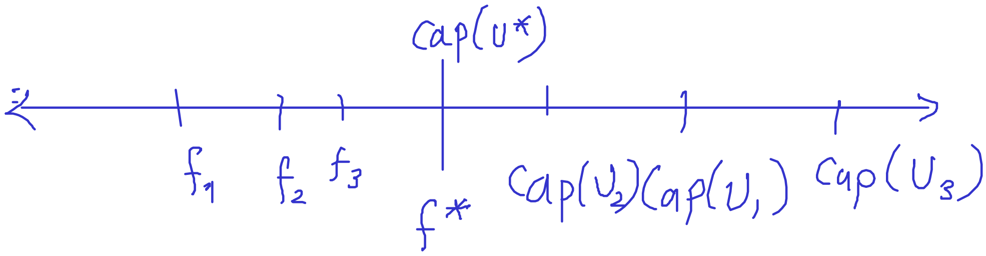
$$= Cap(U)$$

But we know

flow value $\leq$ Cap (U)

Hence the flow is maximum.

Hence Cap(U) = min s-t cut capaicty.



Thm Max s-t flow = Min s-t cut

Terminate

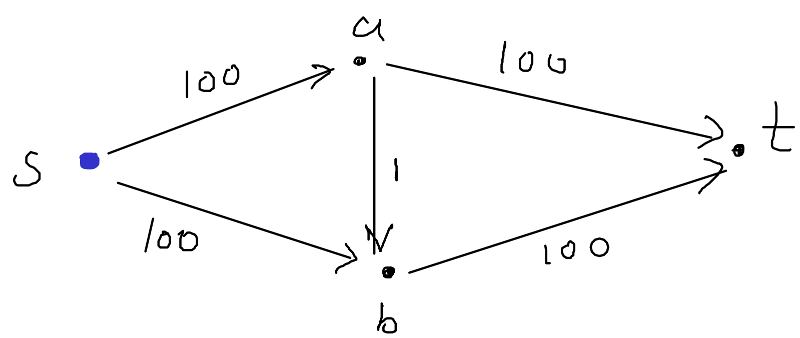
Flow value increase is always integral.

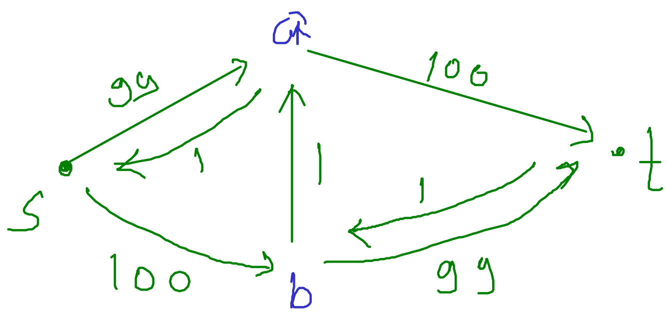
Flow value increases by at least 1.

No. of interation $\leq$ Sum of edge capacities.

Time $O(|E| \cdot C)$ ← exponential in the input size

Pseudopolynomial





Find s-t paths in a clever way.

- Shortest length path (Edmonds Karp)

  $O(VE^2)$ Strongly Polynomial time

- Max bottleneck ~~$O\ E \cdot \log C$~~

  $O(E^2 \log C \cdot E)$

  Not strongly polynomial.

Augmenting Path

March 21, 2023

Reduction :

The following statements mean the same

→ Problem A reduces to Problem B

→ One can design an algorithm for problem A using a subroutine for problem B.

→ Notation $A \leq B$

Example: Multiplication $\leq$ Squaring

TA allocation $\leq$ Network flow

Network flow $\leq$ Linear programming

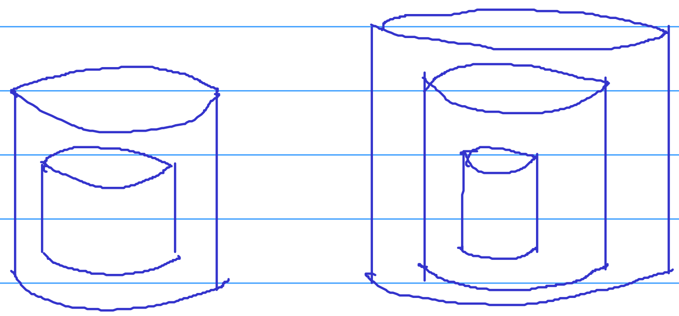
Often we need to use the reduction only once as in the second example.

These are called many-one reductions.

Applications of Network Flow.

1. Network flow variants
    - Multiple sources and sinks with demands
    - Undirected Network
2. Bipartite Matching
3. TA allocation
4. Maximum number of edge disjoint paths from s to t.
5. Airline Scheduling / Container arrangement
6. Project selection

Container arrangement

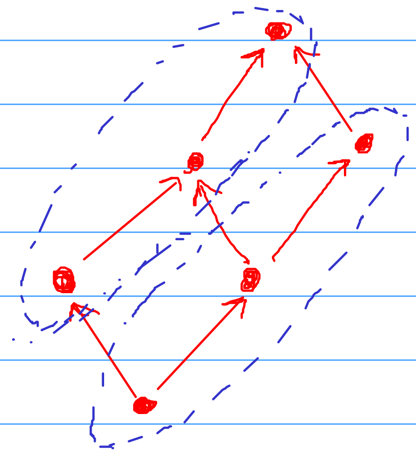


Exam: greedy algorithm worked in the setting of 2 parameters.

More general version: containers with 3 parameters height, width, length

Abstract version

Given a partial order on n elements,
partition them into minimum number of chains



Lower bound : any set of mutually incomparable elements.

Amazingly, if the minimum number of chains in k, then there is a set of k mutually incomparable elements

Another example :

Trains arriving/departing at a station, schedule them using minimum number of platforms.

Taxi Scheduling

A taxi company gets a list of bookings for the next day.

Want to minimize the number of taxis required.

Bookings

B_1 : 9:00 Chembur → Airport 10:00

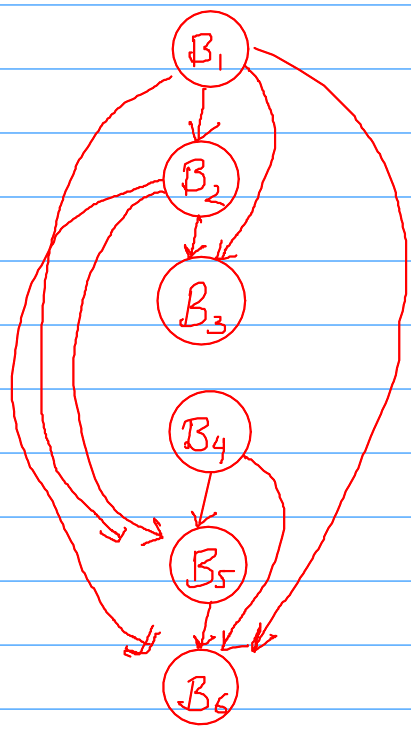
B_2 : 10:30 Andheri → IITB 11:15

B_3 : 11:30 IITB → TIFR 1:30

B_4 : 10:15 Dadar → Airport 11:15

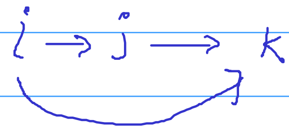
B_5 : 12:00 Powai → LTT 12:45

B_6 : 13:00 Chembur → Sion 13:30

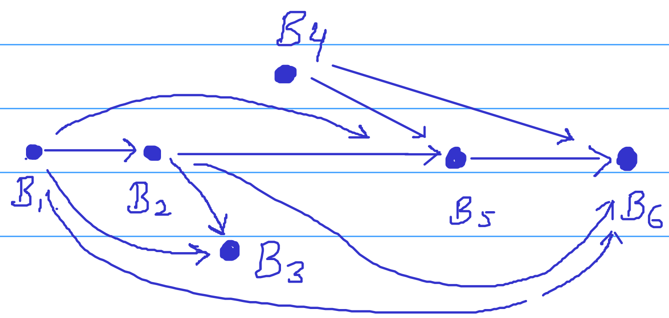


Input: Directed graph — acyclic
— transitively closed

(Partial Order on a set of elements)



Output: Partition of vertices into minimum number of paths.



Algorithm via Network Flow

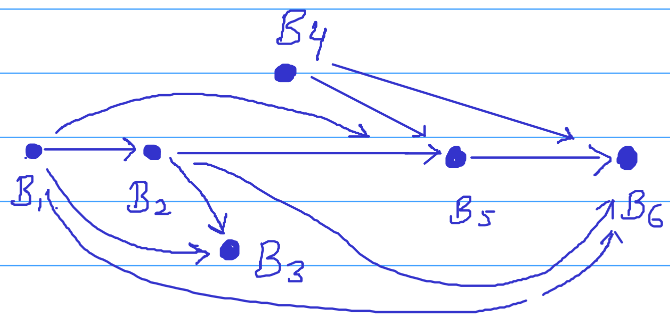
· Given a partial order (directed graph)

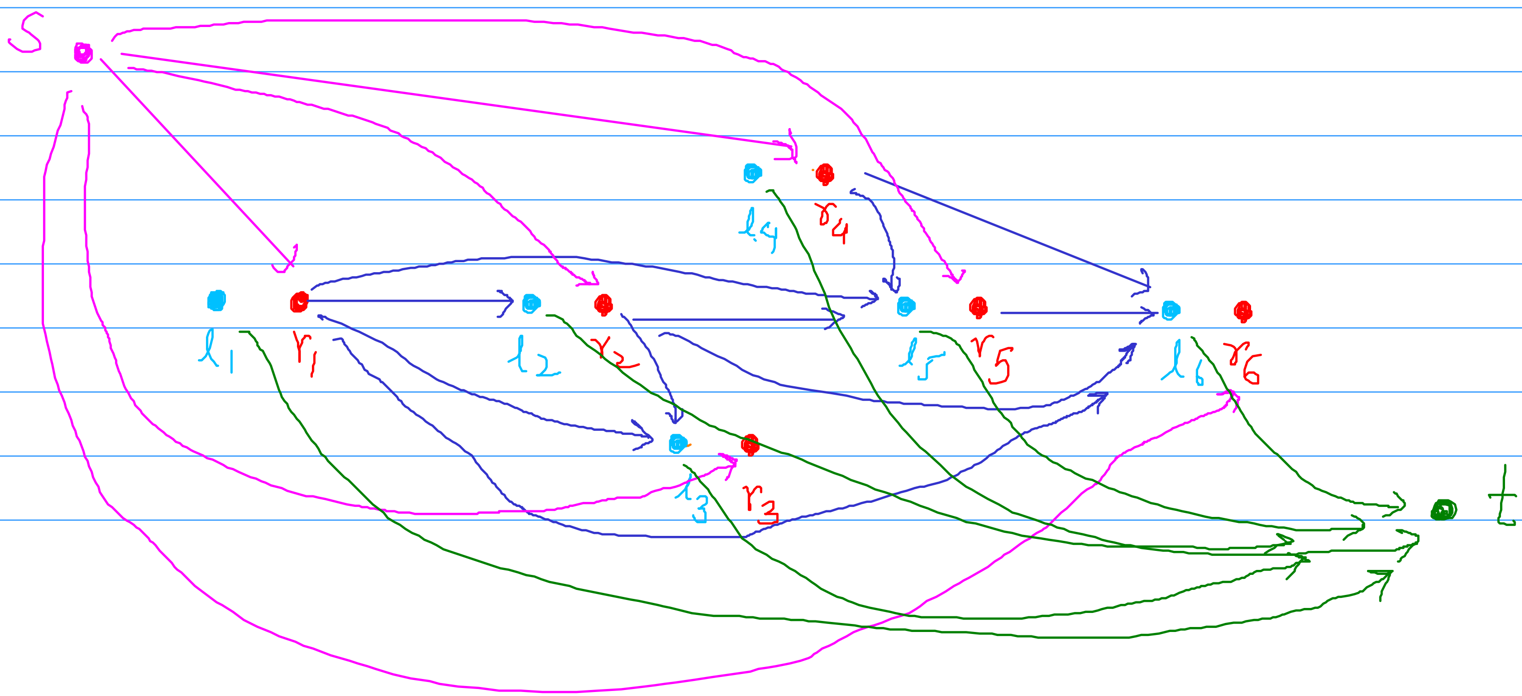
For every element B_i, create two nodes l_i, r_i

If $B_i \rightarrow B_j$ then add an edge from r_i to l_j of capacity 1.

From source s to all r_i's capacity 1

from each l_i to sink t capacity 1





Claim 1 :

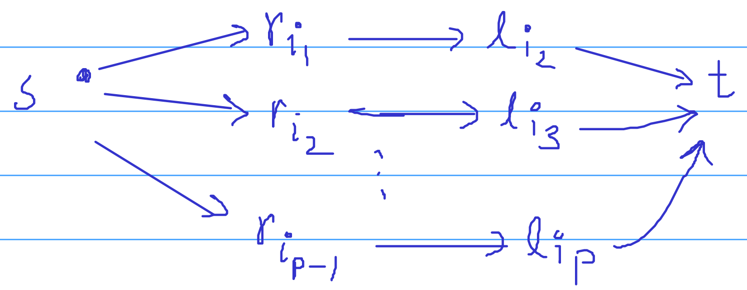
If there is a partition of the partially ordered set into k chains

then there is a flow of $n-k$ units in the network.

Proof: for any chain with p elements, say

$$B_{i_1} \rightarrow B_{i_2} \rightarrow B_{i_3} \; -- \rightarrow B_{i_p}$$

we will have $p-1$ units of flow along the following paths



Note that any vertex r_i or l_j will be used in only one of the paths because any B_i appears in exactly one chain.

Clearly adding the flows corresponding to all the K chains, we have $n-k$ units of flow.

Claim 2: If there is l units of (integral) flow in the network then there is a partition of the given partially ordered set into $n-l$ chains

Proof: If there is flow along $r_i \longrightarrow l_j$ then we will put B_j as a successor of B_i in one of the chains.

Since r_i can have at most one unit of outgoing flow we get that any B_i will get at most one successor.

Similarly any B_j will get at most one predecessor.

Hence the result will be a collection of chains.

How many chains are there?

We start with n elements as n different chains. For every one unit of flow two chains get combined into one.

Hence $n-l$ chains.

Note: A flow is not necessarily integral. But recall that the max flow algorithm gives an integral flow whenever the capacities are integer.

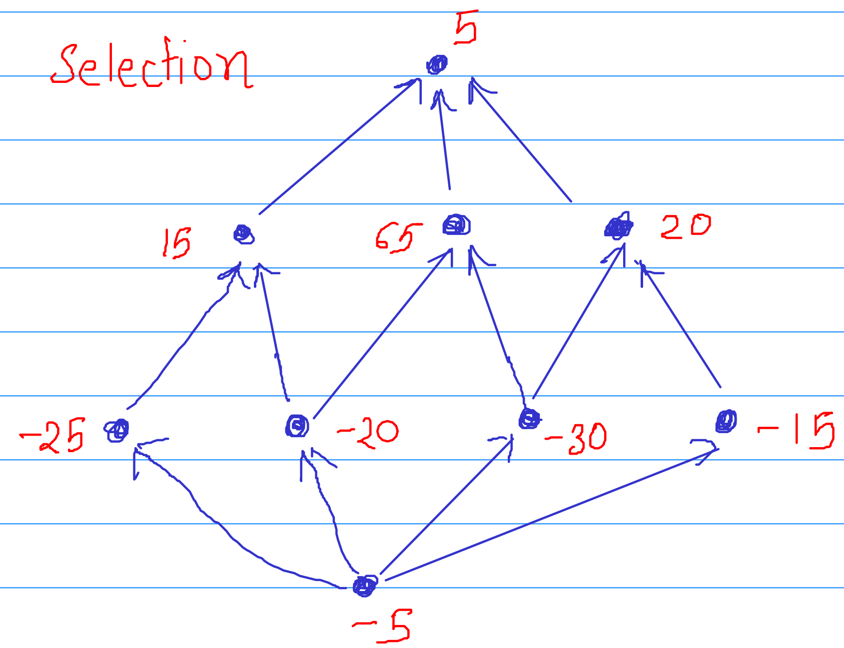
Algorithm ① from given partial ordered set, construct the flow network

② Compute a maximum flow in the network (which is integral)

③ Use claim 2 to obtain a collection of chains from the flow.

HW Claim 1 implies that the collection of chains we get is optimal.

Project Selection



Downward closed subset : If we take an element j we must take everything below it.

Find the downward closed subset which maximizes the total sum.

Reduce this problem to the minimum cut problem.