EC504 ALGORITHMS AND DATA STRUCTURES
FALL 2020 MONDAY & WEDNESDAY
2:30 PM - 4:15 PM

g 6 6 t

Prof: David Castañón, dac@bu.edu

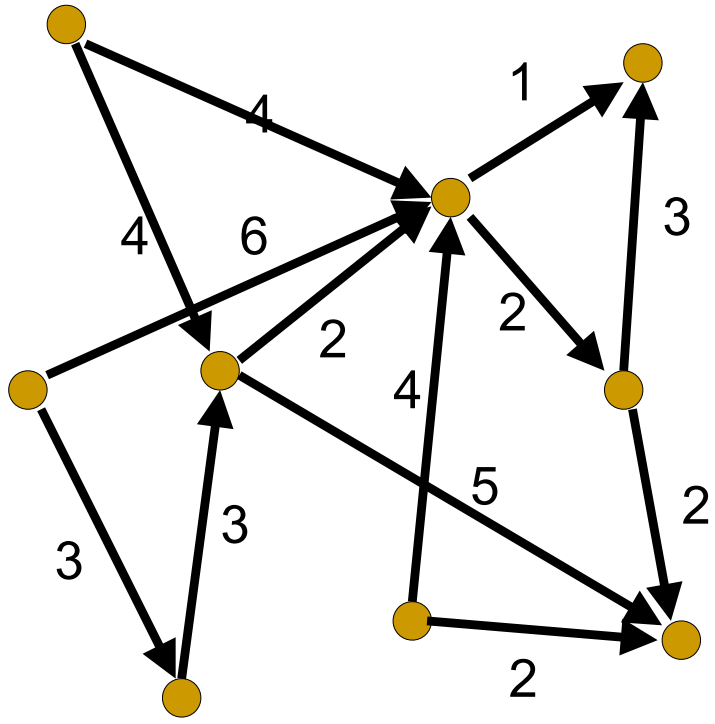
GTF: Mert Toslali, toslali@bu.edu

Haoyang Wang: haoyangw@bu.edu

Christopher Liao: cliao25@bu.edu

Directed, Capacitated Graphs

- A capacitated, directed graph $G = (V, E)$ is a directed graph along with a capacity function $c : E \rightarrow \Re^+$
- Capacities are positive
  - Represent maximum number of simultaneous units that can use an edge



Maximum Flow in Flow Networks

- A feasible st-flow (flow) $f : E \to \Re^+$ is a function that satisfies

  $0 \le f(u,v) \le c(u,v)$ for all edges $(u\,.\,v) \in E$ (capacity satisfied)

  For every $v \in V - \{s,t\}$, flow is conserved: $\sum_{(u,v)\in E} f(u,v) = \sum_{(v,w)\in E} f(v,w)$

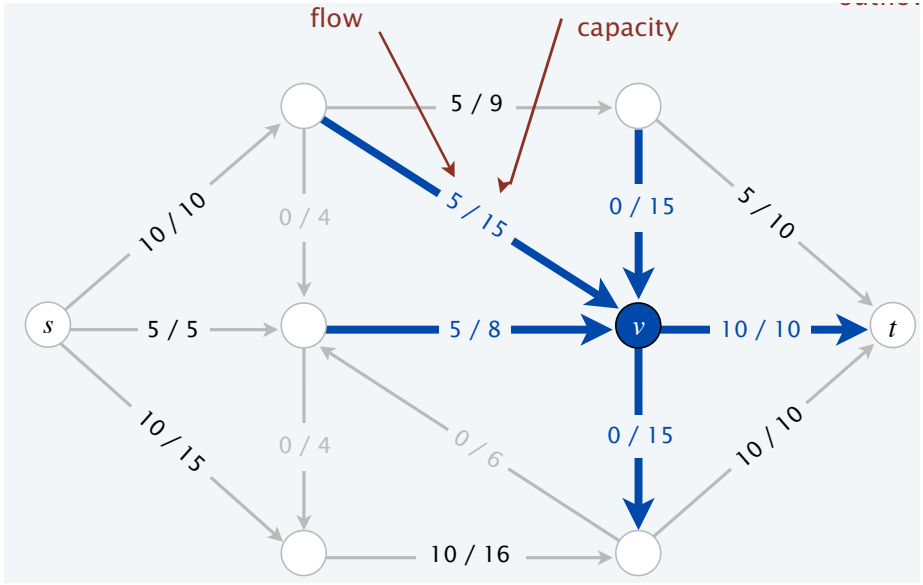
- Value of flow: net flow out of s

  $$val(f) = \sum_{(s,v)\in E} f(s,v) - \sum_{(v,s)\in E} f(v,s)$$

  Value is 25 on the right…

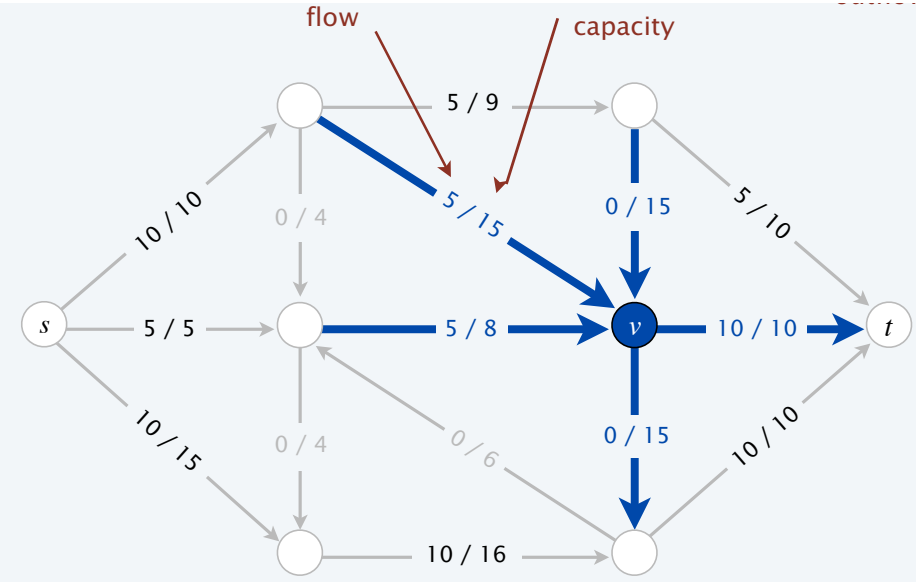
- Maximum flow problem: find flow f that maximizes $val(f)$

-



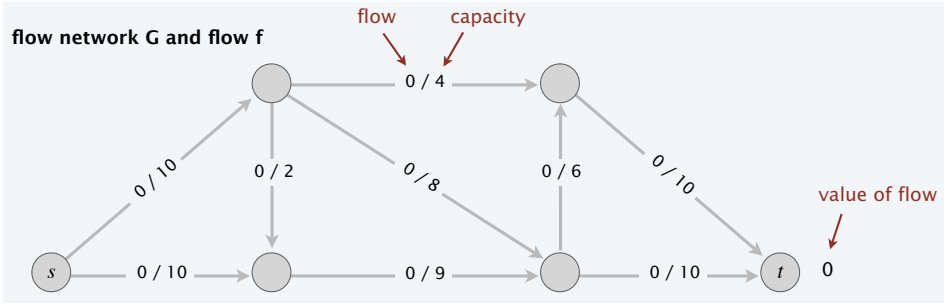
Max-flow, Min-cut are Related

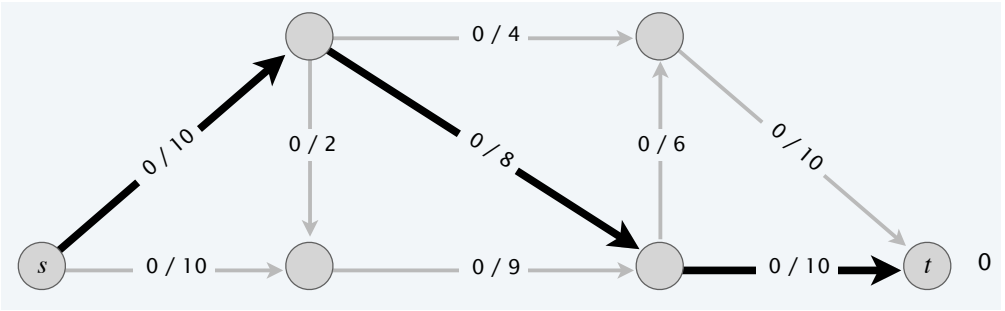
- Formally, they are dual of each other
- Result to come: optimal value of max-flow problem is the same as optimal value of min-cut problem
- Any s-t cut is an upper bound to the max flow problem
- Any feasible flow provides a lower bound to the min-cut problem



Initial Algorithm: Greedy approach

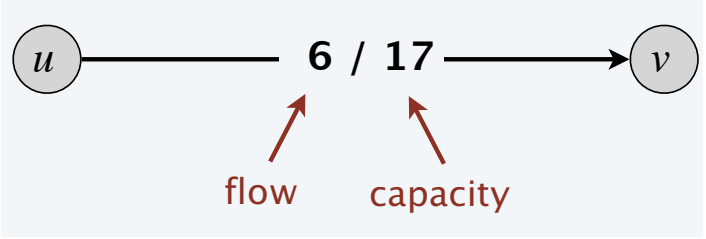
- Start with f (e) = 0 for each edge e ∈ E
- Find an s⇝t path P where each edge has f (e) < c(e)
- Augment flow along path P
- Repeat until you get stuck





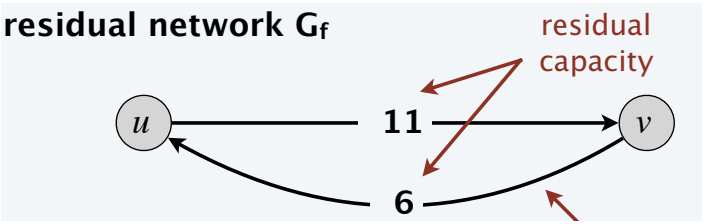
How to Backtrack

- Key concept: Residual network given a flow f
- Original edge. e = (u, v) $\in$ E
  - Flow f(u,v) > 0, capacity c(u,v)



- Want to create a way to "undo" assigned flow
  - Create a reverse edge (v,u) with capacity c(v,u) = f(u,v)
  - Flow on this reverse edge reverses the existing flow f(u,v)
- Residual network $G_f = (V, E_f)$
  - Capacity on residual network edges:

$$c_f(u,v) = \begin{cases} c(u,v) - f(u,v) & (u,v) \in E \\ f(v,u) & (v,u) \in E \end{cases}$$



- $E_f = \{(u,v) \in E : f(u,v) < c(u,v)\} \cup \{(v,u) : (u,v) \in E, f(u,v) > 0\}$

Augmenting Paths

- An augmenting path is a simple s—>t path in the residual network G_f
- The bottleneck capacity bottleneck(P, G_f) of an augmenting path P is the minimum residual capacity of any edge in P
- For bottleneck capacity δ, we can augment flow on P to modify f:
  - If $(u, v) \in P, (u, v) \in E$, augment $f'(u, v) = f(u, v) + \delta$
  - If $(v, u) \in P, (u, v) \in E$, this is a reverse edge; reduce $f'(u, v) = f(u, v) - \delta$
- Key property. Let f be a flow and let P be an augmenting path in G_f. Let δ be the flow on P (bottleneck capacity). Then, the augmented f' is a feasible flow in the original graph, and $val(f') = val(f) + \delta$

Ford-Fulkerson Algorithm ('56)

- Start with flow f(u,v) = 0, $(u, v) \in E$. Form the residual network $G_f = G$
- While there exists an s —> t path P in the residual network
  - Compute residual capacity δ on P, and augment flow f using flow δ on path P
  - Update residual network using new flow f, as G_f
- When no path can be found, return flow f

- Properties of algorithm:
  - At each iteration, flow increases
  - If capacities are integers, this must terminate: flow out of s increases by at least 1 at each iteration, and has upper limit as the sum of capacities of edges leaving s

Ford-Fulkerson Algorithm

- Properties:
  - For integer capacities, all the flows in Ford-Fulkerson are integers
  - Each augmenting path iteration is O(|E|) (use BFS or DFS)
  - Each augmenting iteration increases flow by at least 1 unit
  - Let C be maximum capacity of any edge on graph. Then, value of max flow is less than |V|C

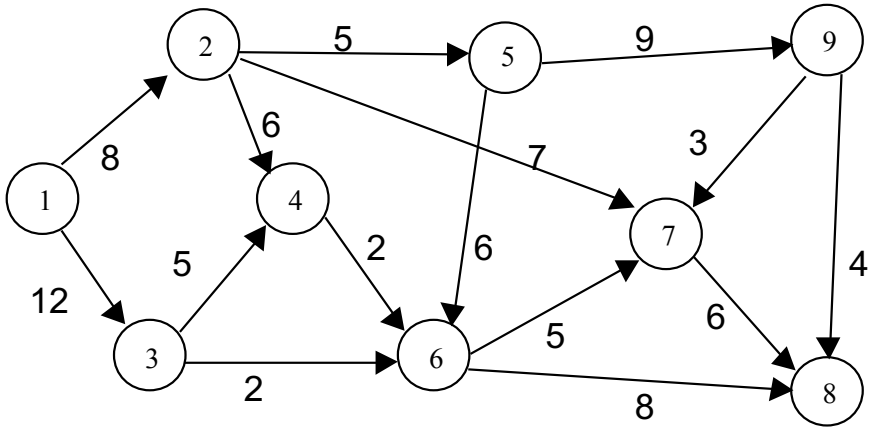
- Overall complexity O(|V||E| C)
  - Not a polynomial complexity (C can be very large…)

Easy Fix: Edmonds-Karp ('72)

- Choose augmenting paths in residual network that have least number of edges
  - Use BFS to find augmenting paths!
- Nice properties:
  - Length of augmenting path does not decrease in iterations
  - After at most |E| iterations, length of augmenting path must have increased
  - Maximum length of augmenting path is |V|-1
  - Bound on number of augmenting paths: $O(|V||E|)$
  - Bound on complexity: $O(|V||E|^2)$
- Another approach: choose augmentations with sufficient capacity
  - Capacity scaling used in best algorithms today

Example

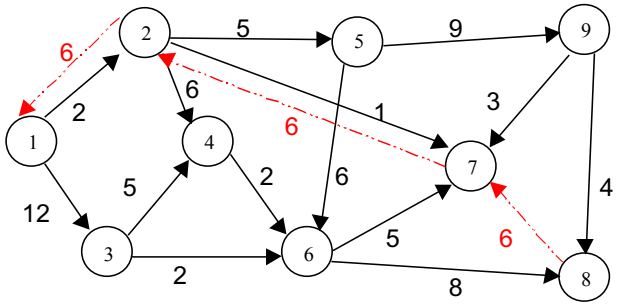
- Numbers are capacities, s = 1, t = 8



- BFS: augment 1—>2—>7—>8, capacity 6

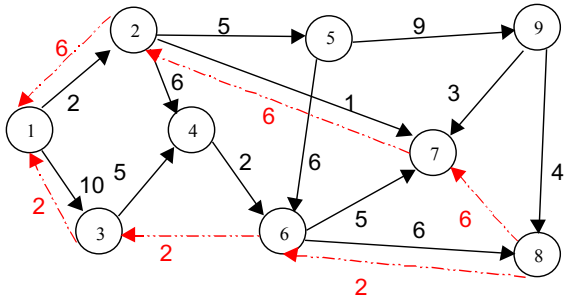
Example - 2

- Residual network



- BFS 2: augment 1—>3—>6—>8, capacity 2

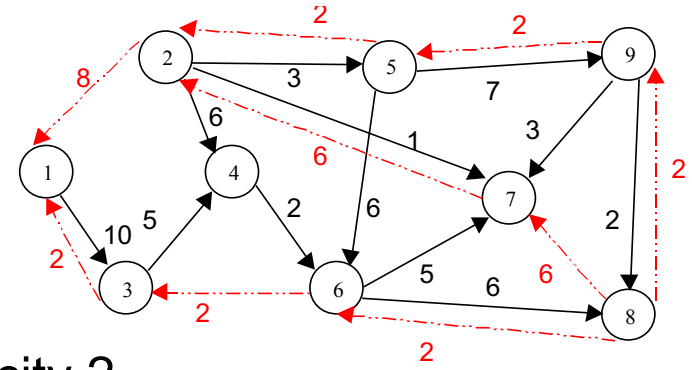
- Residual network



- BFS 3: augment 1—>2—>5—>9—>8, capacity 2

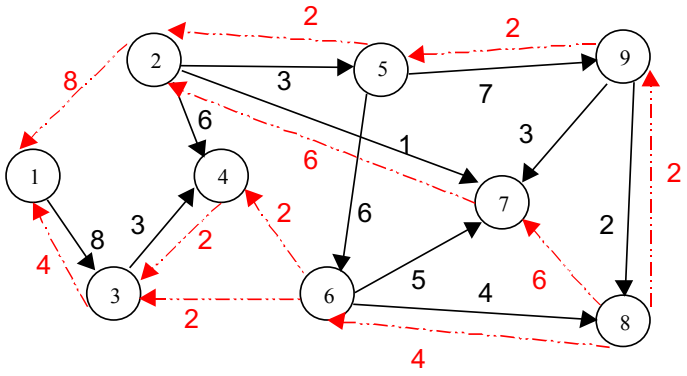
Example - 3

- Residual network



- BFS 4: augment 1—>3—>4—->6—>8, capacity 2

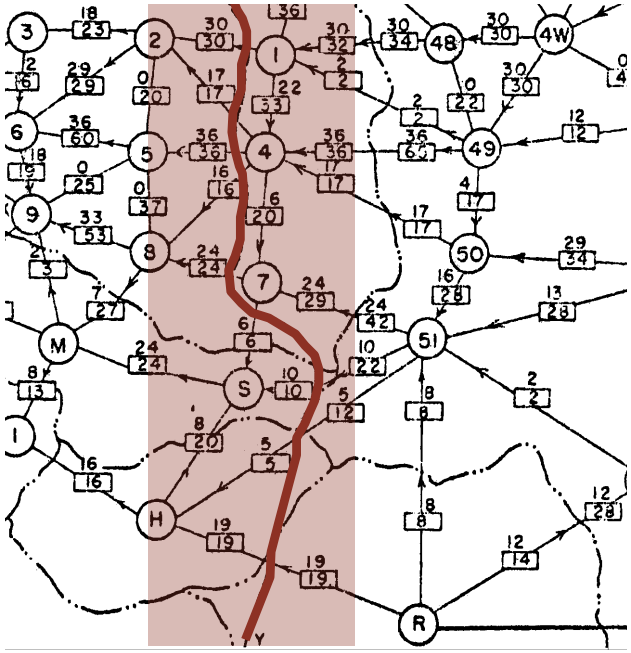
- Residual network



- No augmentation possible: Max-flow = 12
  Min Cut A = {1,3,4}

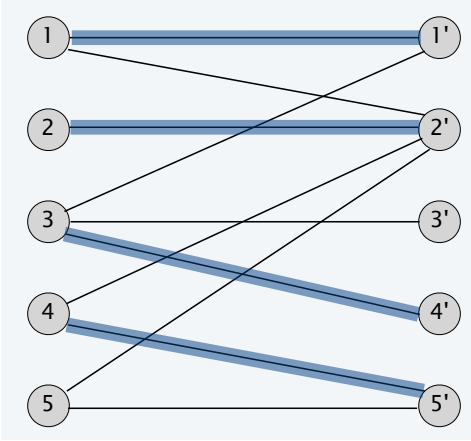
Max Flow Applications

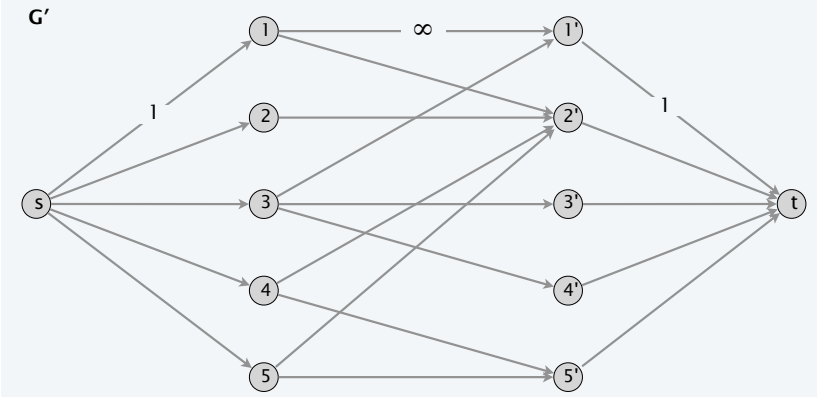
- Many applications, some selected ones below
  - Bipartite matching
  - Image segmentation
  - Network attack
  - …
- Illustration: how to maximally degrade railroad supply networks
- Equally valid: how to jam your wireless network using cyberattacks



Application: Bipartite Matching

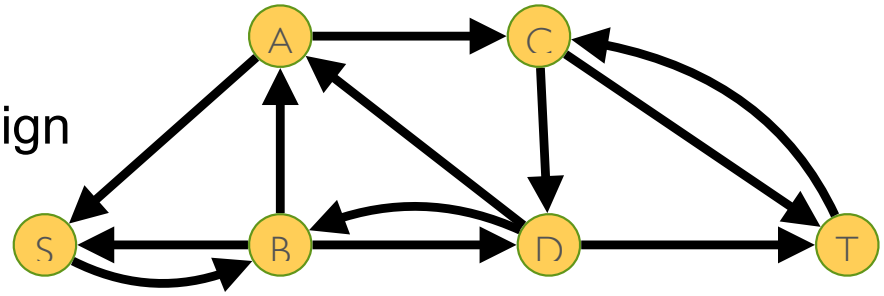
- Given a bipartite graph, find a maximum cardinality matching
  - Edges indicate compatible matches
- Solution: convert to max flow problem: assign capacity 1 to all edges, augment with vertices s, and t
- Max flow will be cardinality of maximum matching, and flow will give matches





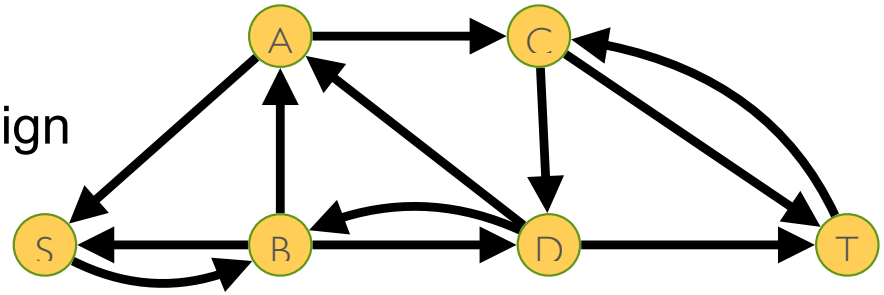
Application: Edge-Disjoint Paths

- Given a directed graph, find maximum number of edge-disjoint paths from s to t
  - Measures redundancy, reliability
- Solution: convert to max flow problem: assign capacity 1 to all edges
  - Max flow will be number of node-disjoint paths



Application: Network Connectivity

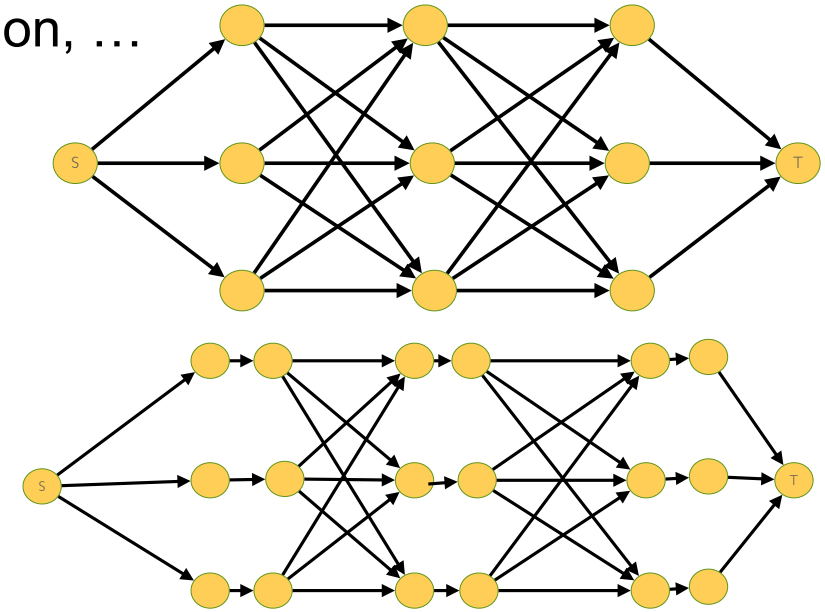
- Given a directed graph, find minimum number of edges required to disconnect s from t
  - Measures vulnerability



- Solution: convert to max flow problem: assign capacity 1 to all edges
  - Max flow will be minimum number of edges required to disconnect graph
    Min cut will identify those edges

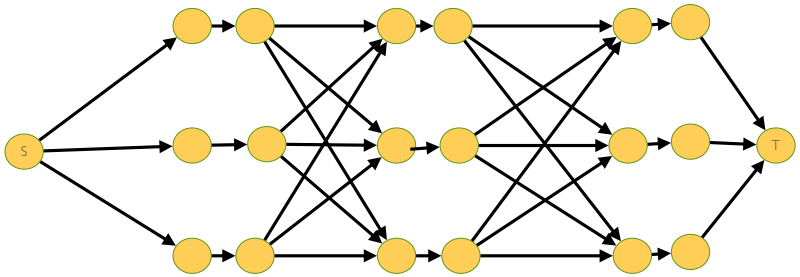
Application: Viterbi Decoding

- Given a directed trellis with edge weights and node weights, find shortest path from s to t
  - Used in CDMA chips, in speech recognition, …
- Weights on edges correspond to distances
  - Most likely interpretation: shortest path
- However, there are also weight on vertices
  - Create auxiliary graph with vertex splitting



Finding Vertex Disjoint Paths

- Use vertex splitting technique, create capacity 1 edge between vertex halves



- Solve max flow problem on augmented graph
- Extension: Find the best k vertex-disjoint paths
  - e.g. finding k most likely trajectories in sets of images…
  - Combination of shortest path and max flow algorithms!

Preflow-Push Algorithms (’88)

- Ford-Fulkerson, Edmonds-Karp work on ideas of augmenting paths
  - Each augmenting path takes O(|E|) operations
  - Little progress per iteration
  - Hard to parallelize
- New idea: augment edges, not paths (work step by step)
  - Inspiration: Bertsekas’ auction algorithm for assignment (1979)
  - Goldberg-Tarjan (’88) : preflow-push or push-relabel algorithm for max flow problems
  - Ideas extend to many other optimization problems: assignment, min-cost network flow, shortest paths, …

Preflows

- Key idea: at each intermediate stages we permit more flow arriving at nodes than leaving (except for s)

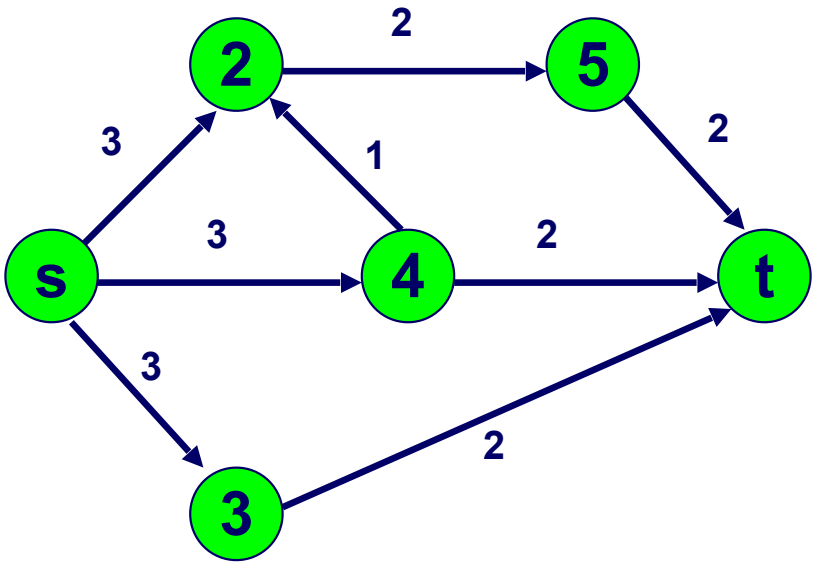
- A **preflow** is a function $x : E \to \Re^+$, where $0 \leq x(u,v) \leq c(u,v)$ and

$$e(v) = \sum_{(u,v)\in E} x(u,v) - \sum_{(v,w)\in E} x(v,w) \geq 0, \text{ for } v \in V - \{s,t\}$$

  - e(v) is the excess at vertex v
  - The excess is required to be non-negative
  - Like a flow, but relaxing conservation of flow (use x instead of f)

A Feasible Preflow

- The *excess* *e(v)* at each vertex v ≠ s, t is the flow in minus the flow out
- Note: total excess = flow out of s minus flow into t
- Flow is not conserved



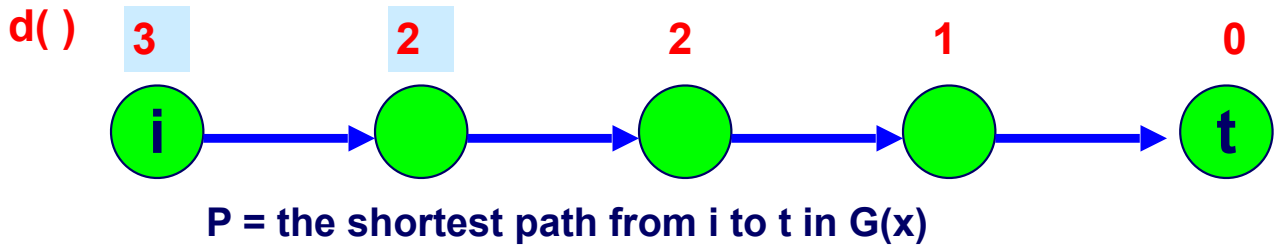
Distance Labels

- Let G_x be the residual network for a preflow x(). Distance labels d( ) are valid for G_x if
  - i. d(t) = 0
  - ii. d(v) ≤ d(u) + 1 for each $(u, v) \in E_x$
- Defn. Let r(u,v) be the capacity of edge (u,v) in residual network G_x
- An edge (u, v) is admissible if c(u,v) > 0 and d(u) = d(v) + 1
- Lemma. Let d( ) be a valid distance label. Then d(u) is a lower bound on the distance (minimum number of edges) from u to t in the residual network (if there is a path from u to t that excludes s)

Distance Labels and Gaps

- We say that there is a gap at a distance level k (0 < k < n) in G_x if there is no vertex with distance label k

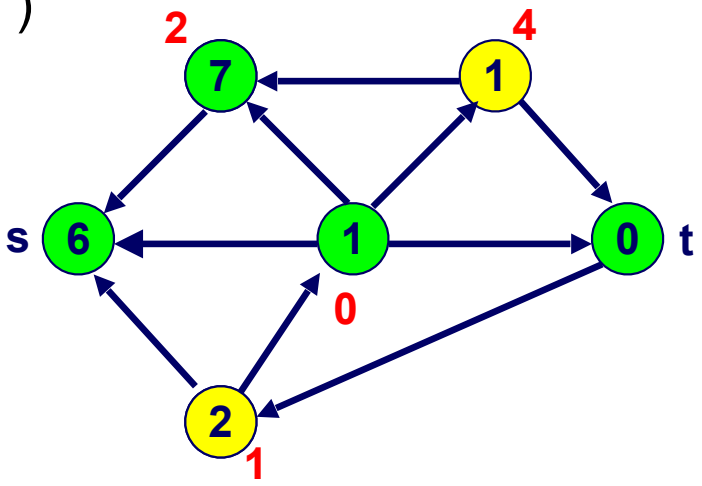
- Lemma. Suppose there is a gap at distance level k. Then for any vertex v with d(v) > k, there is no path from v to t in the residual network
  - Proof by contradiction: The shortest path from v to t would have to pass through a node whose distance level is k



Active Vertices

- A vertex v in G_x is active if:
  - e(v) > 0 and
  - there is no gap at a distance level less than d(v) (so there is a possible path to t)

- The preflow push algorithm will push flow from active nodes "towards the sink" t, relying on d( )

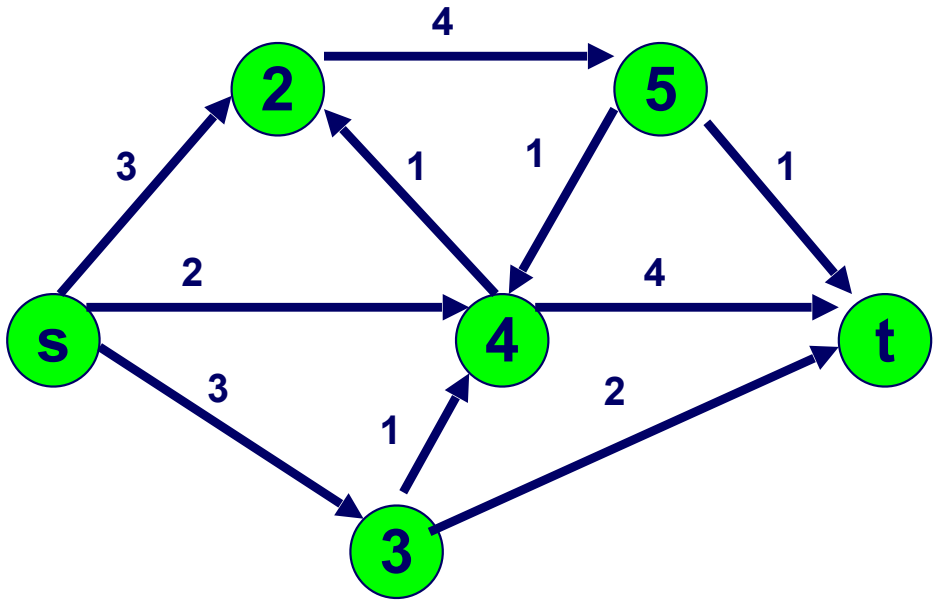


Distance labels inside vertex,
Red numbers are excess

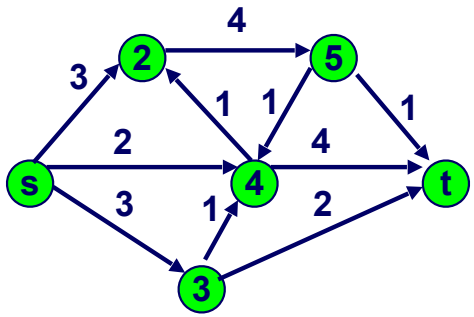
Goldberg-Tarjan Preflow Push Algorithm

- Initialize:
  - $G_x = G; x(u,v) = 0,\ (u,v) \in E$
  - Using BFS reverse from t, compute distance d(v) for every vertex v
  - For every $(s,v) \in E$, set $x(s,v) = c(s,v)$; set e(v) = c(s,v); set d(s) = |V|
  - Update G_x, with residual capacities $r(u,v),\ (u,v) \in E_x$
- While there is an active node in G_x, select active vertex v and push/ relabel(v):
  - If there is admissible edge (v,w): $x(v,w) := x(v,w) + \min(e(v), r(v,w))$
  - Otherwise increase d(v): $d(v) = \min\{d(w) + 1 : (v,w) \in E_r\}$
- Once there are no active nodes, send all excess flow back to s

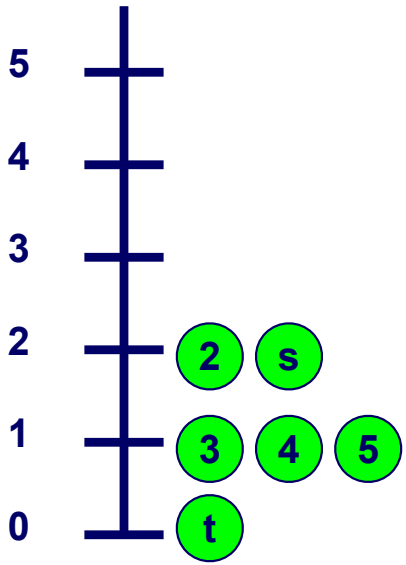
Example (J. Orlin, MIT)

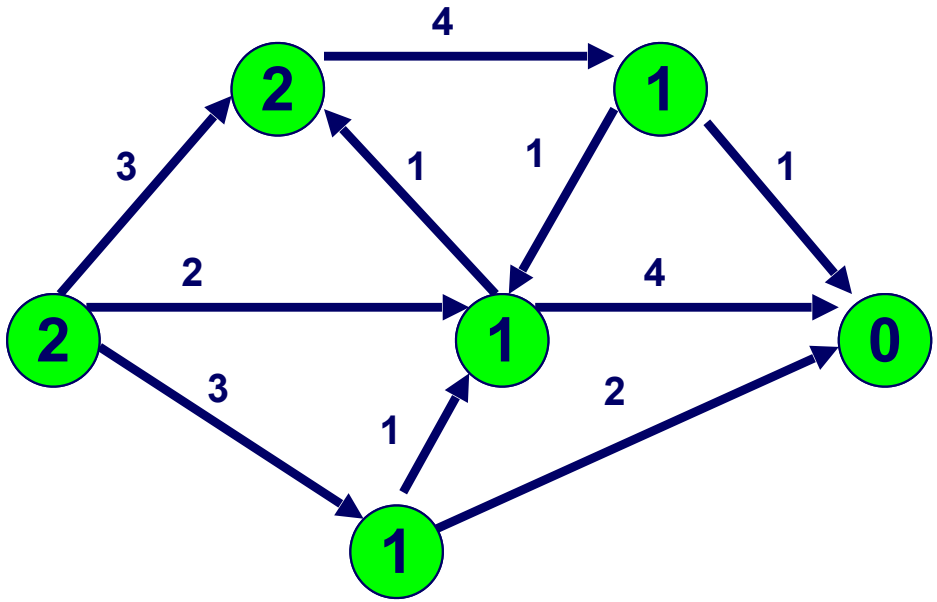


This is the original network, which is also the original residual network.



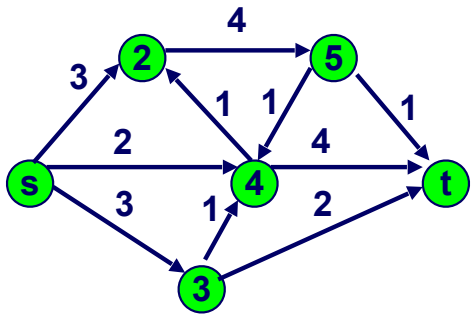
Initialize Distances



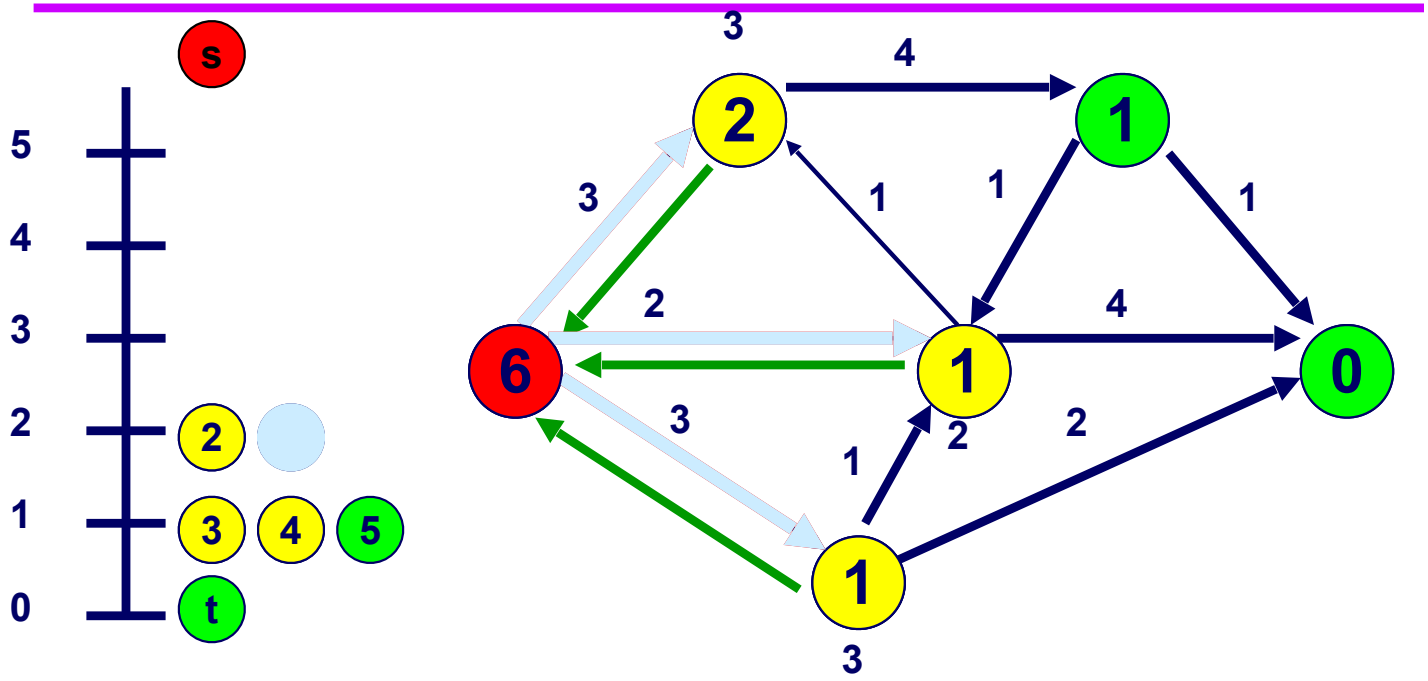


The node label henceforth will be the distance label.

d(j) is at most the distance of j to t in G(x)



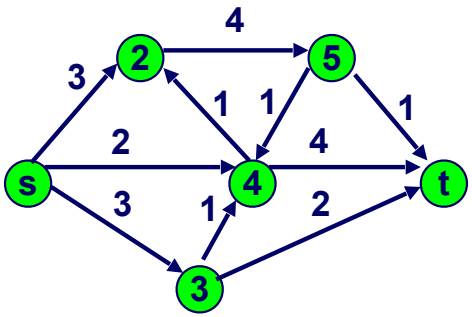
Saturate Arcs out of node s



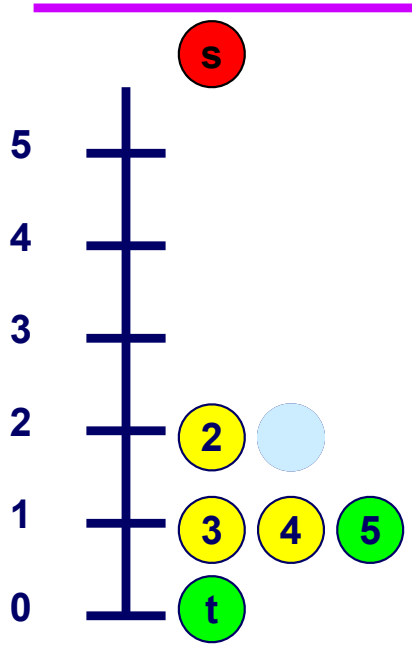
Saturate arcs out of node s.

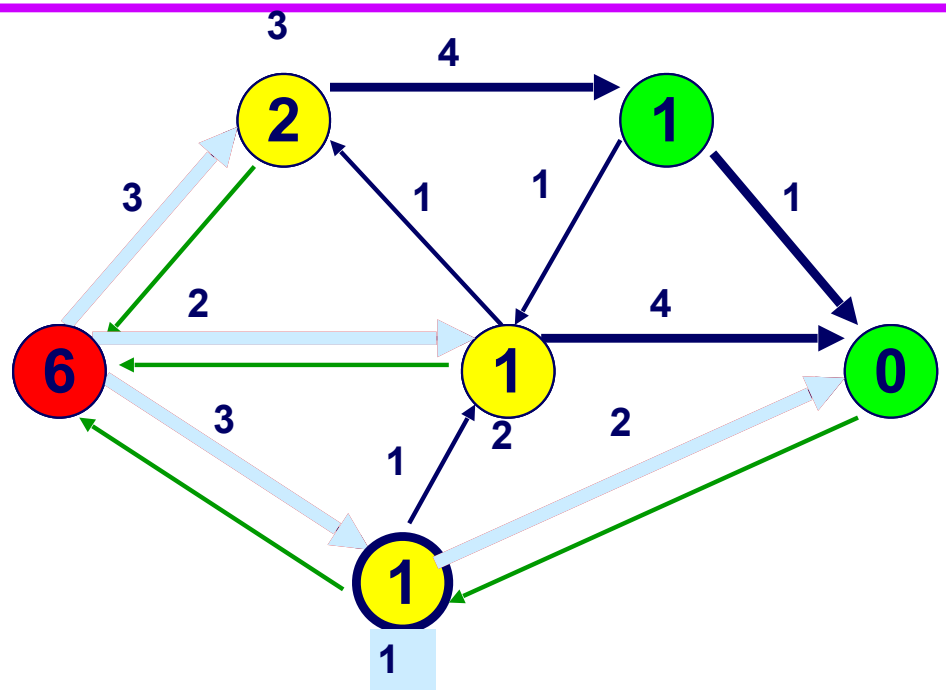
Move excess to the adjacent arcs

Relabel node s after all incident arcs have been saturated.



Select active, then relabel/push

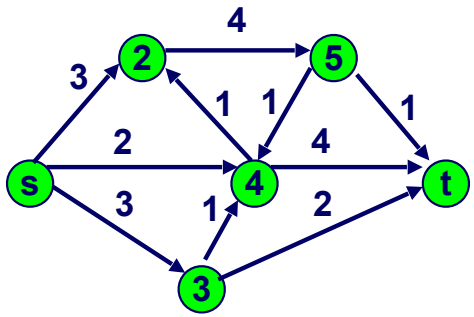




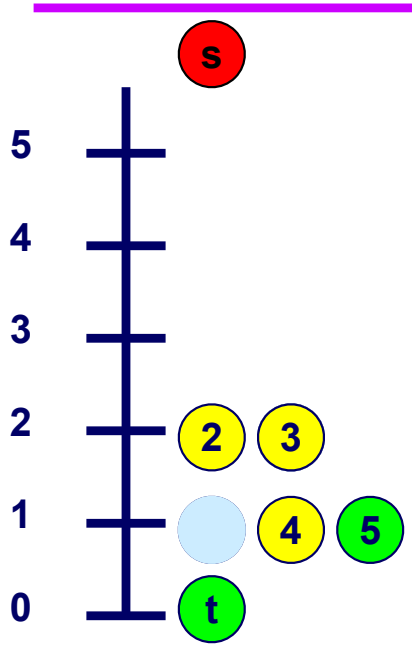
Select an active node, that is, one with excess

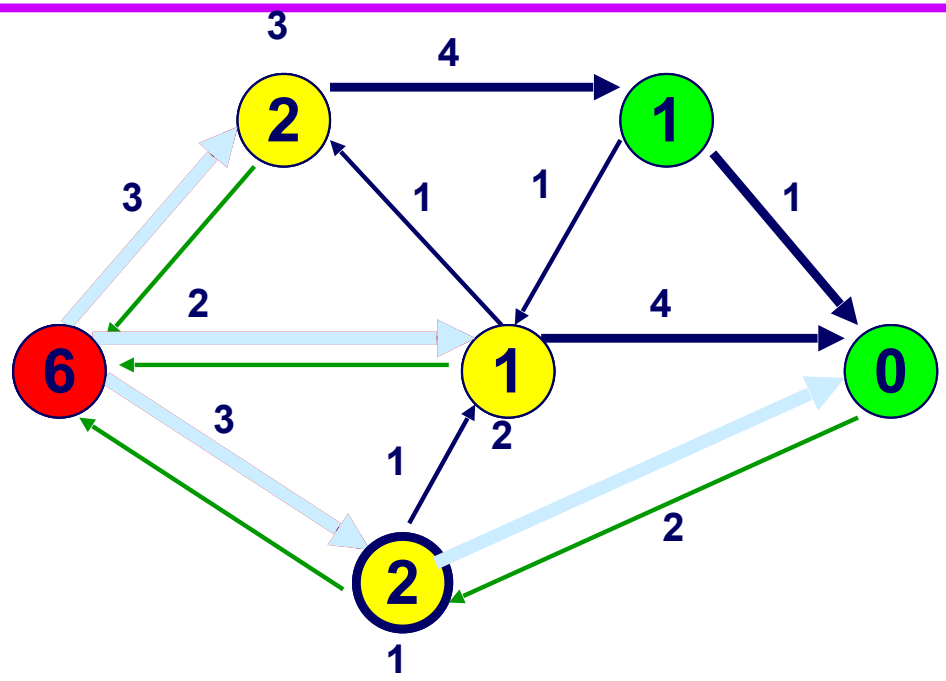
Push/Relabel

Update excess after a push



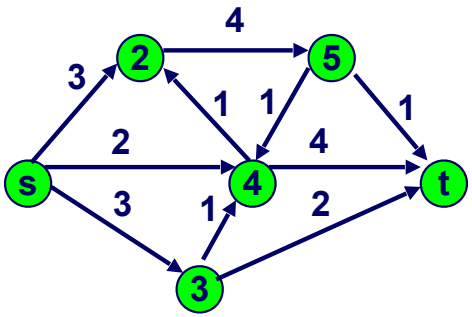
Select active, then relabel/push



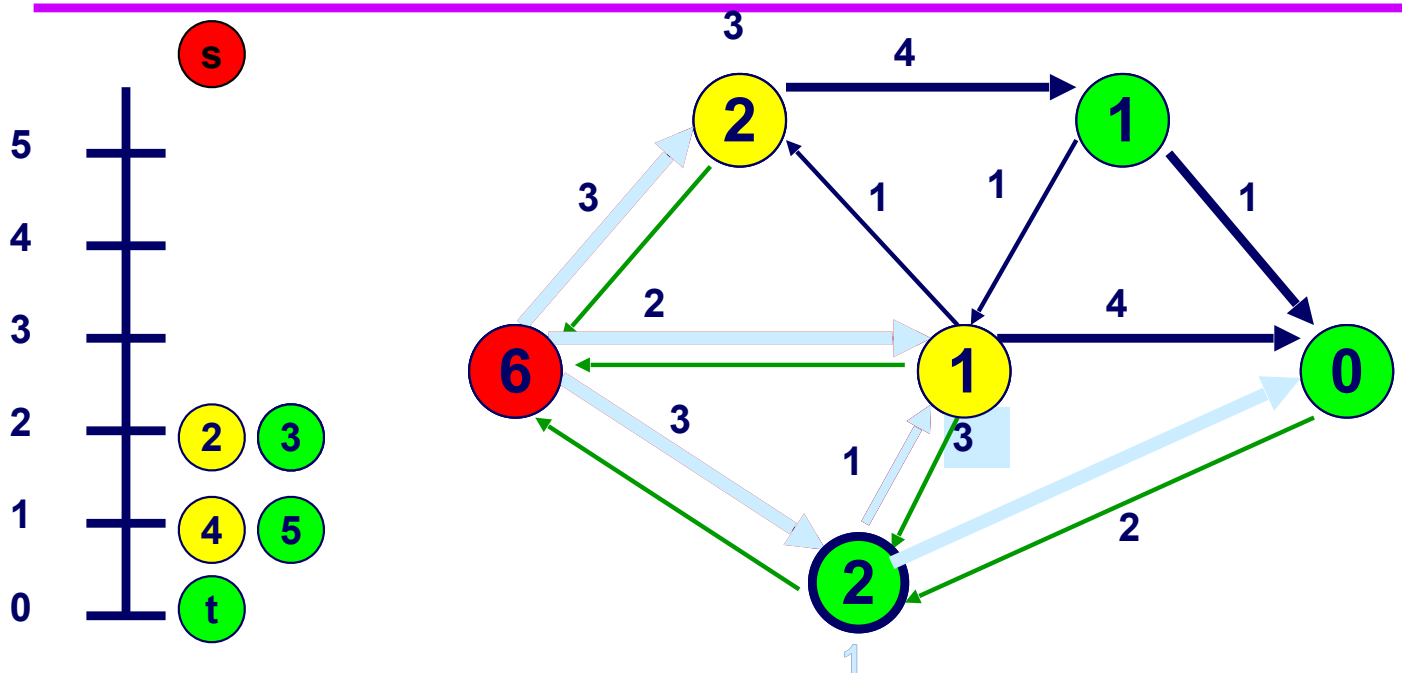


Select an active node, that is, one with excess

No arc incident to the selected node is admissible. So relabel.

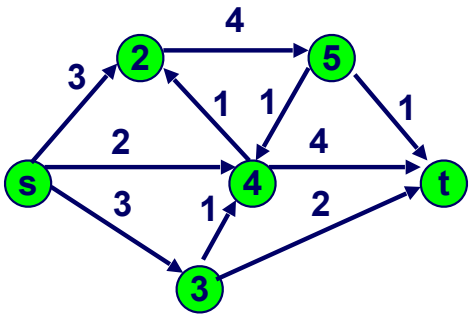


Select active, then relabel/push

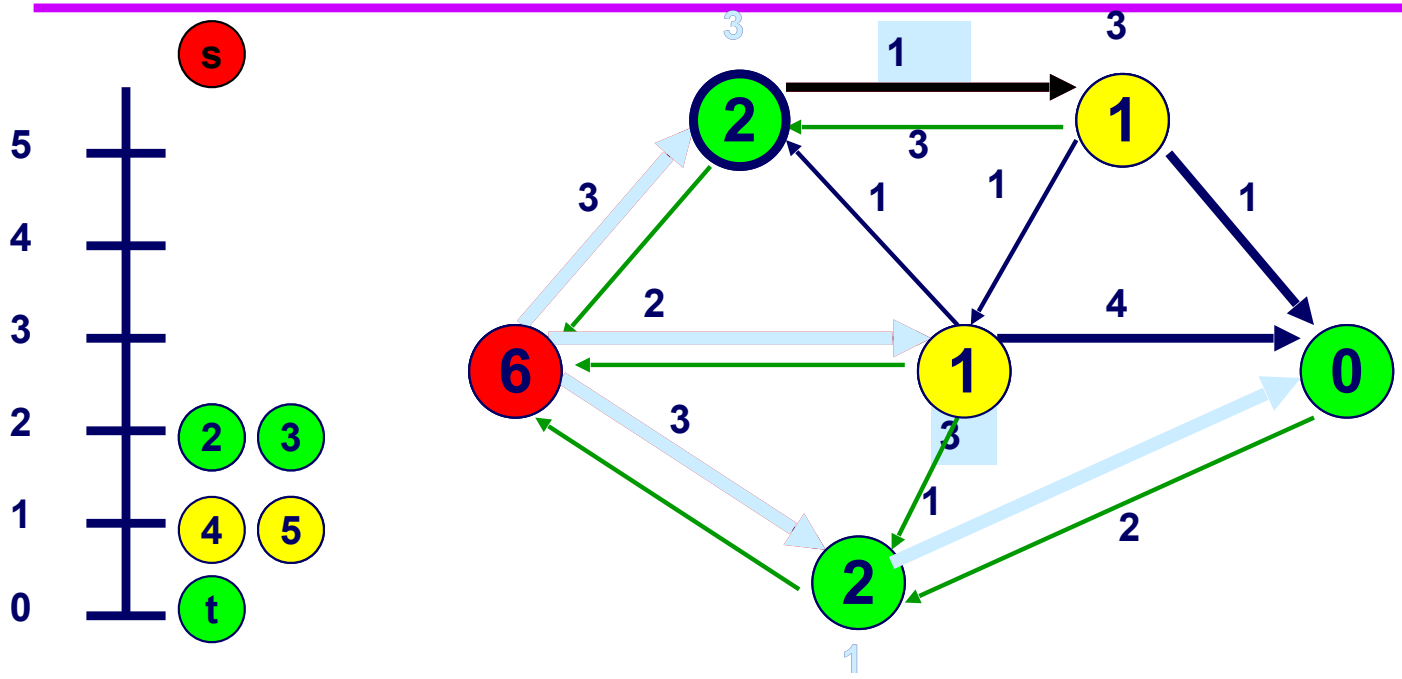


Select an active node, that is, one with excess

Push/Relabel

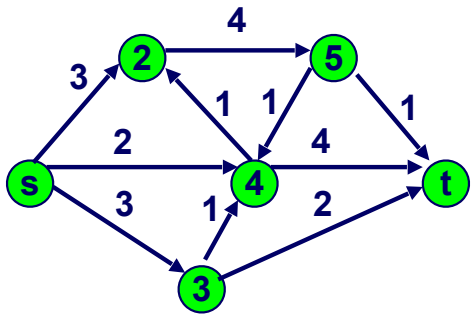


Select active, then relabel/push

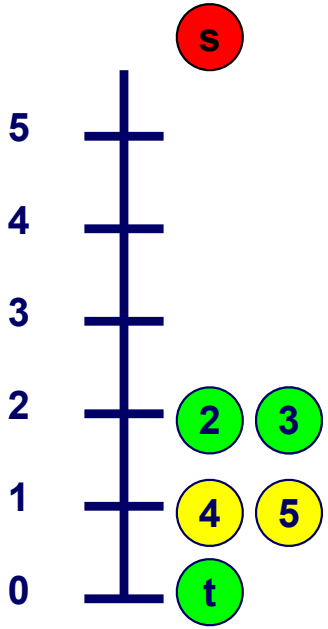


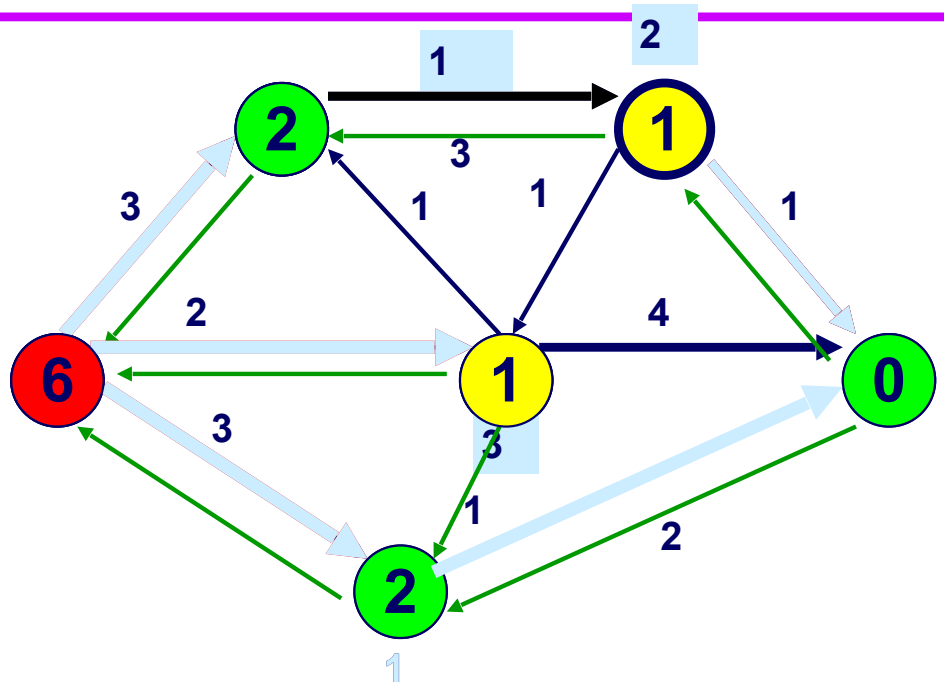
Select an active node.

Push/Relabel



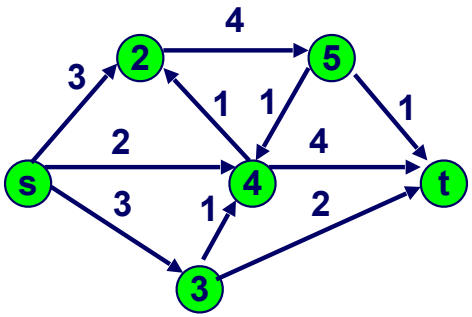
Select, then relabel/push



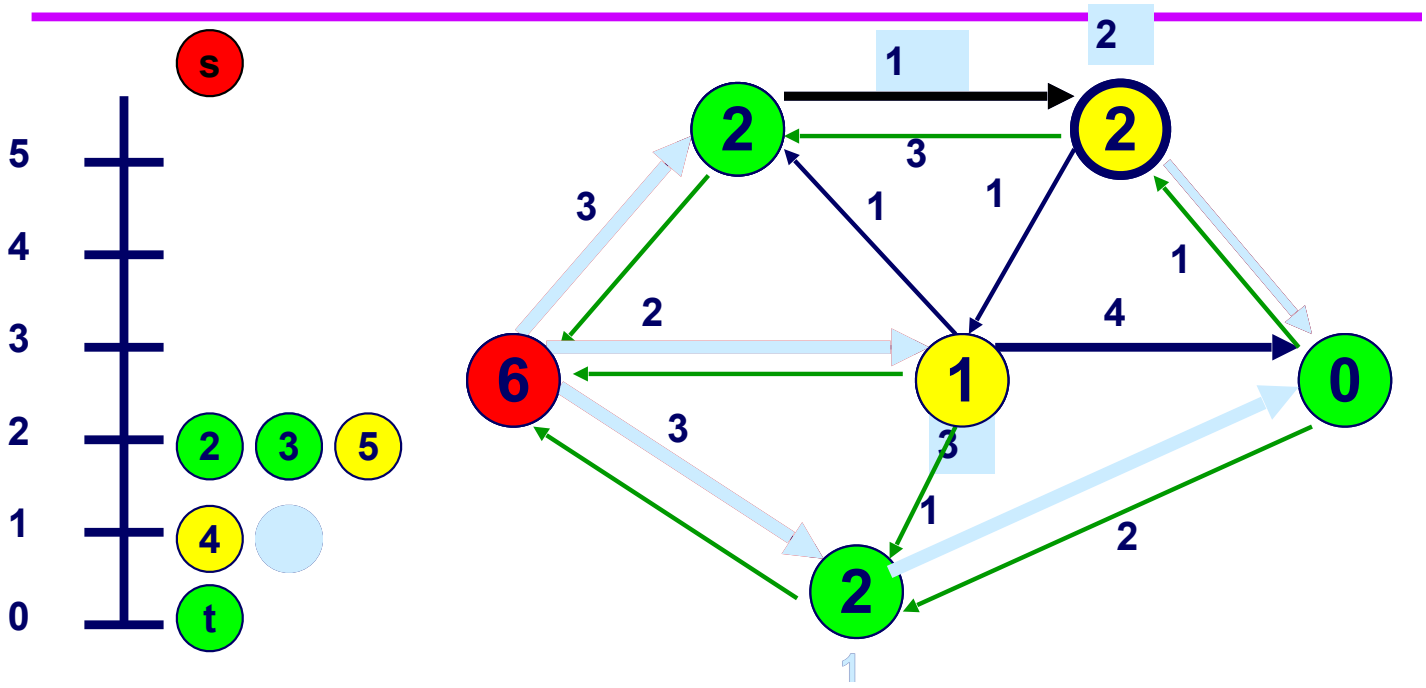


Select an active node.

Push/Relabel

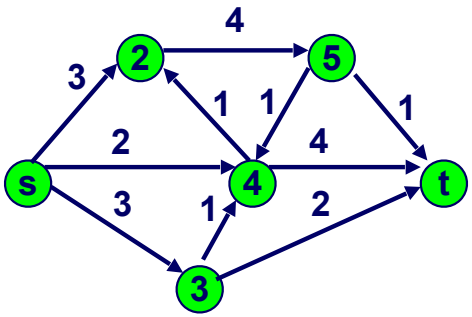


Select, then relabel/push

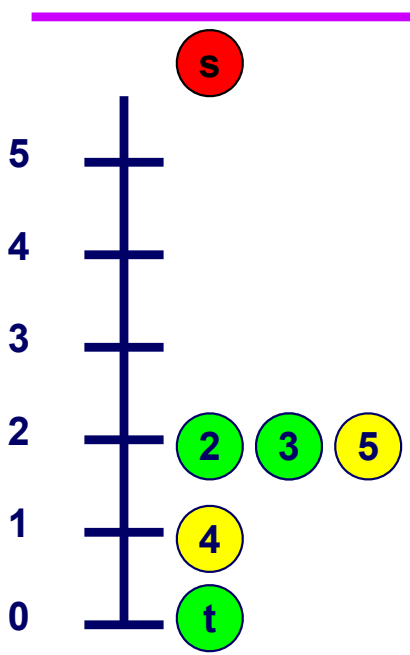


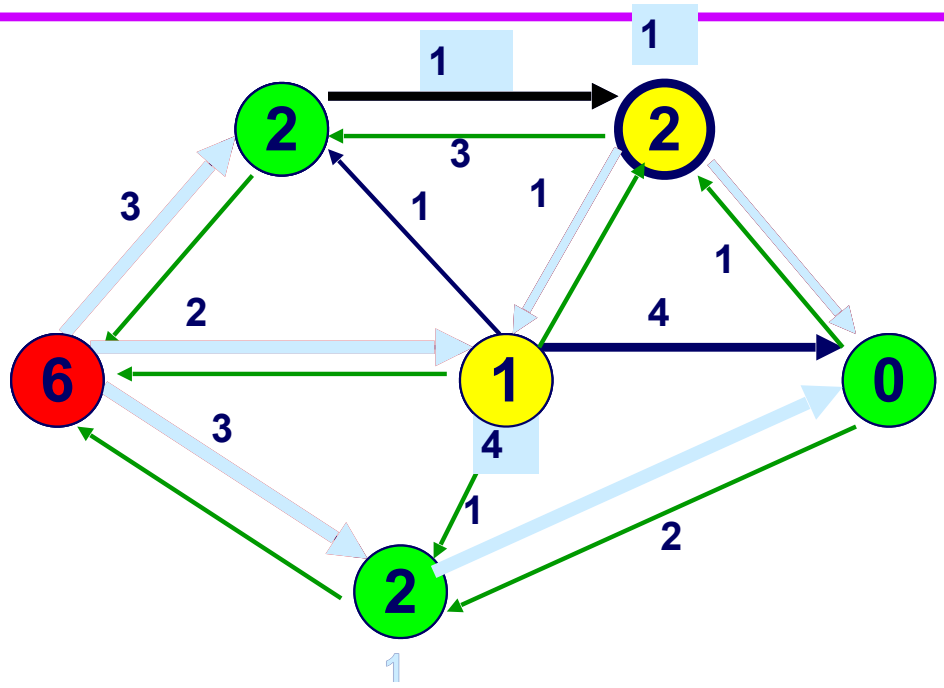
Select an active node.

There is no incident admissible arc. So Relabel.



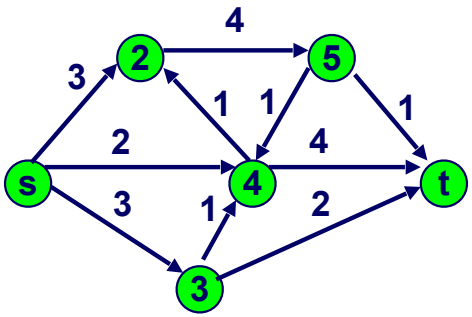
Select, then relabel/push



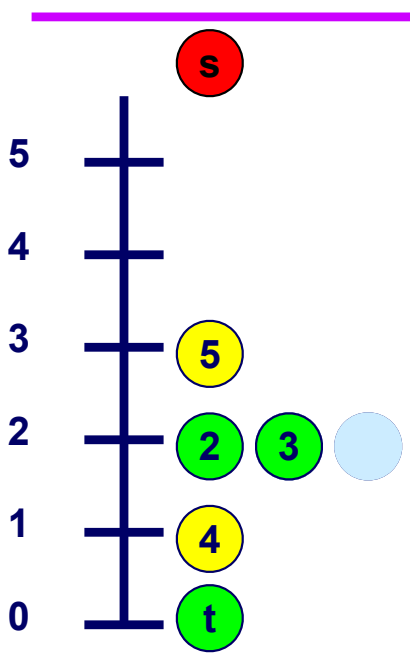


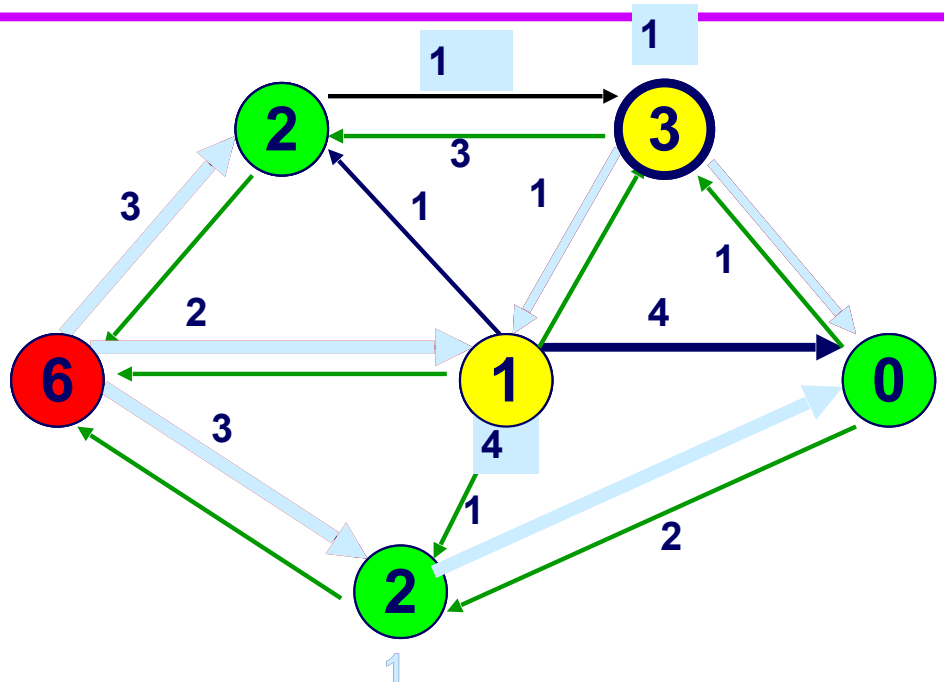
Select an active node.

Push/Relabel



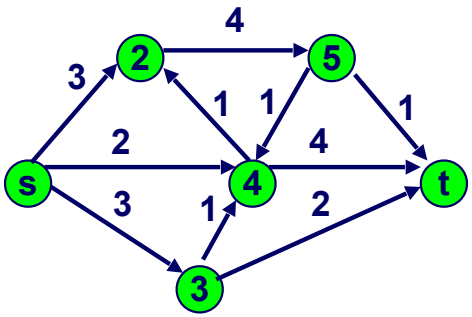
Select, then relabel/push



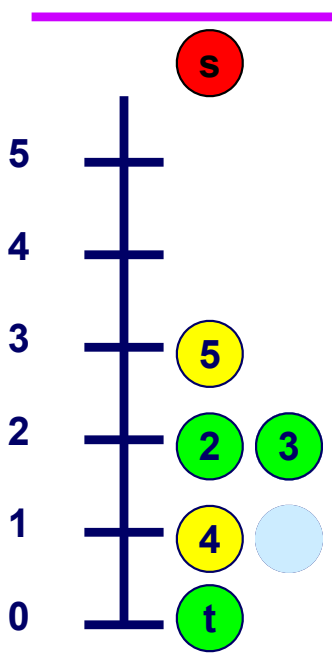


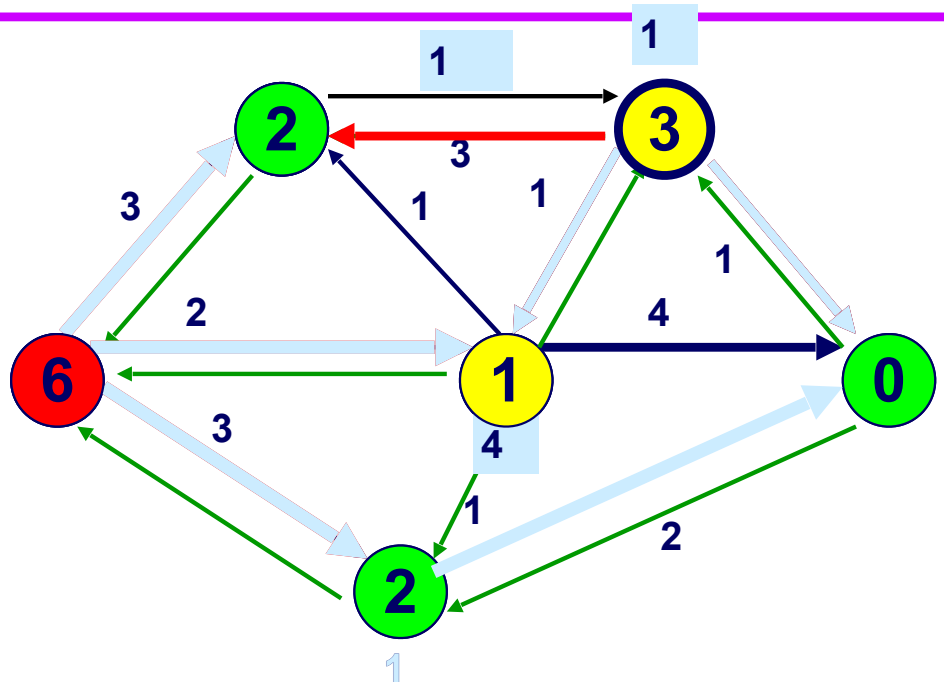
Select an active node.

There is no incident admissible arc. So relabel.



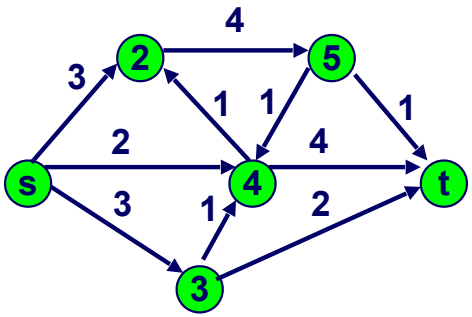
Select, then relabel/push



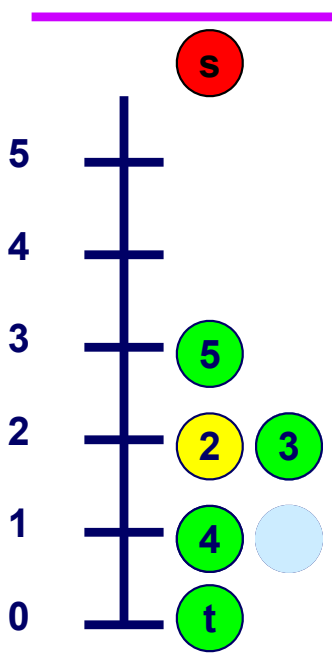


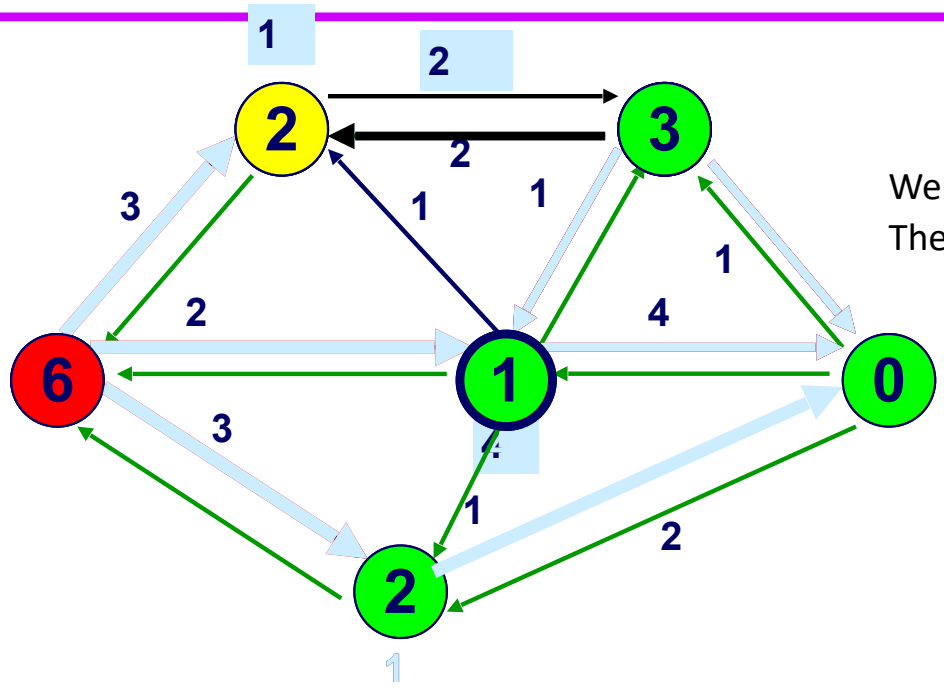
Select an active node.

Push/Relabel



Select, then relabel/push

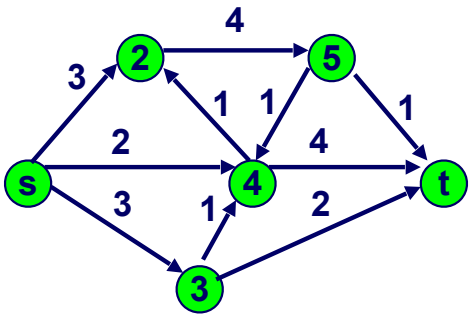




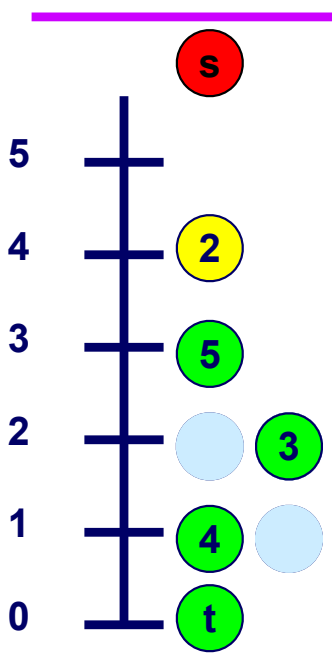
We have the max flow!
The rest is just to recall flow

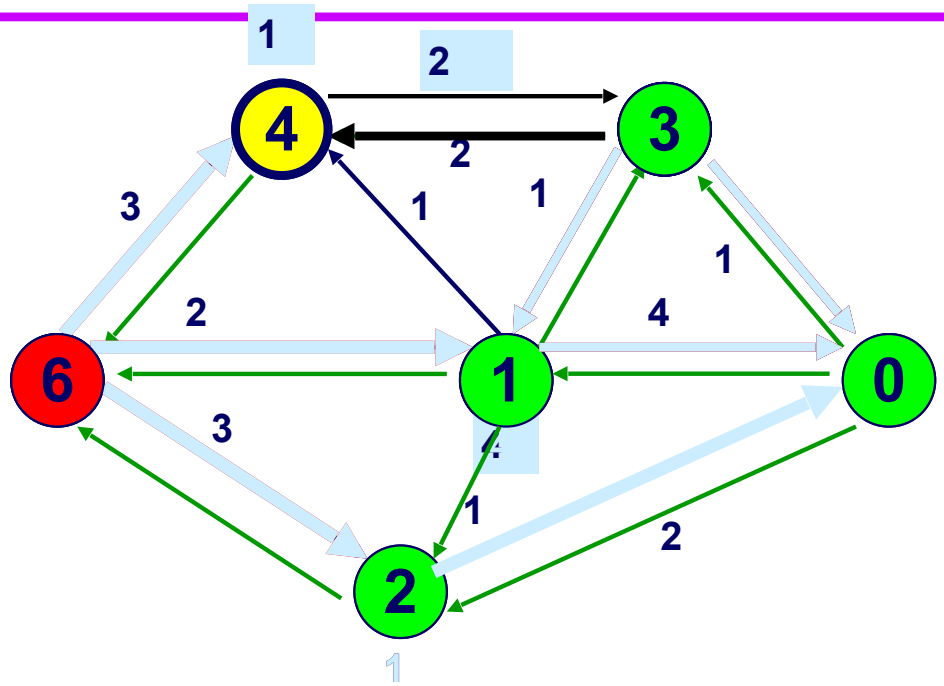
Select an active node.

Push/Relabel



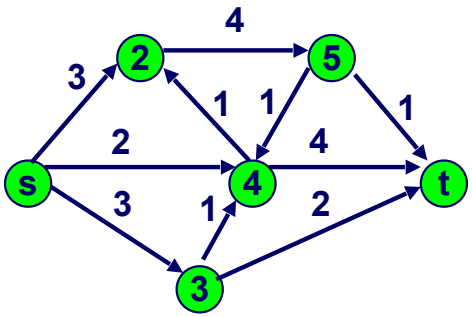
Select, then relabel/push



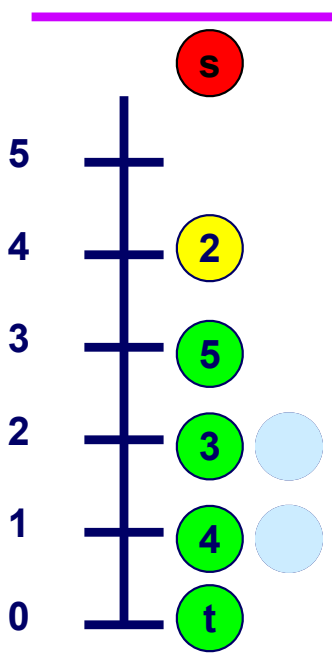


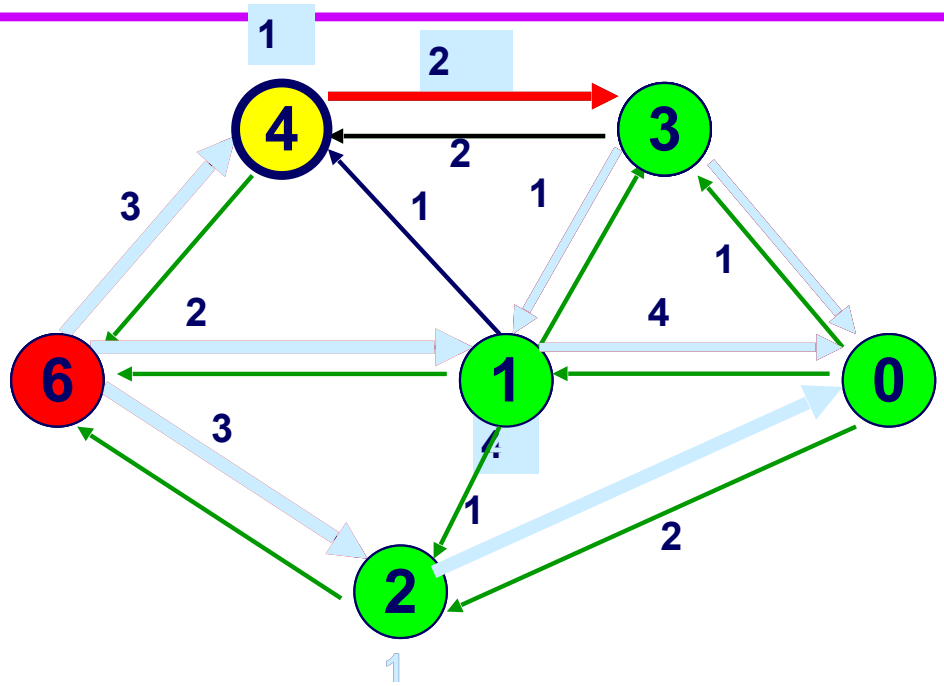
Select an active node.

Push/Relabel



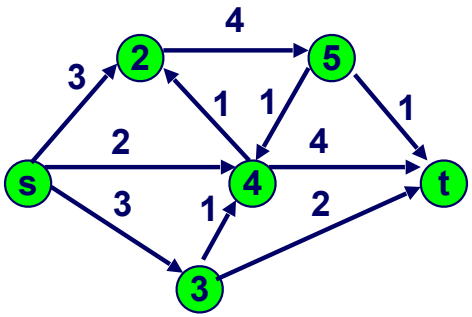
Select, then relabel/push



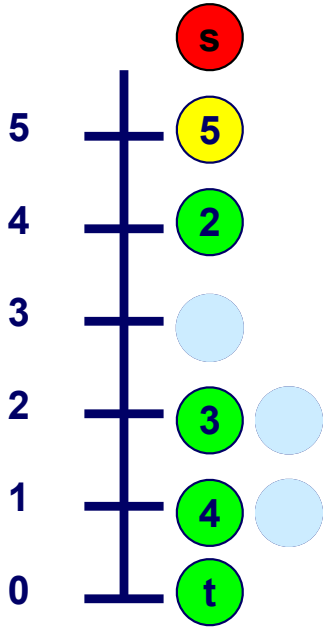


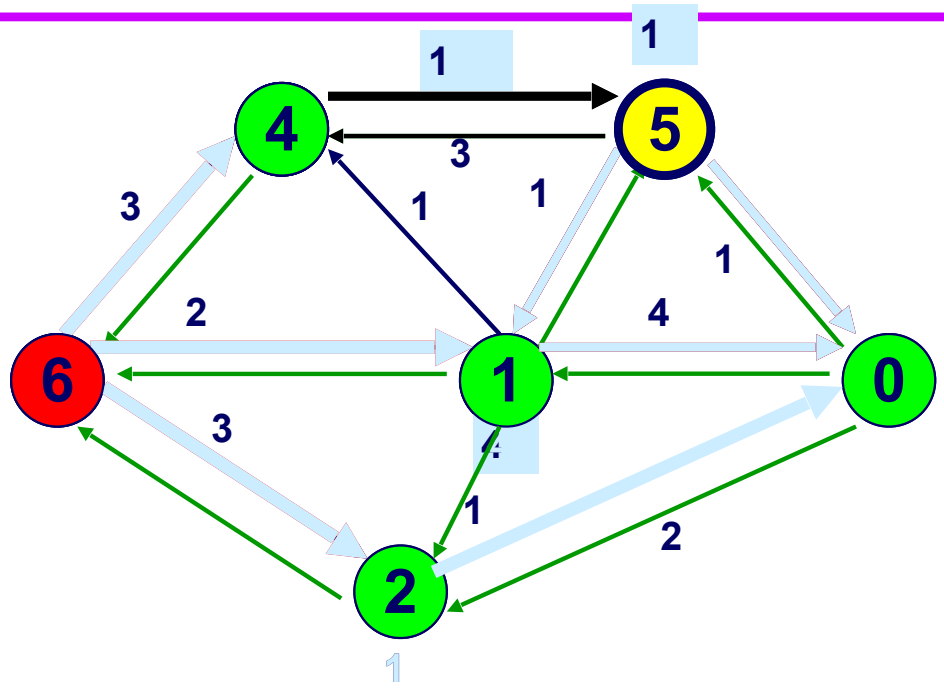
Select an active node.

Push/Relabel



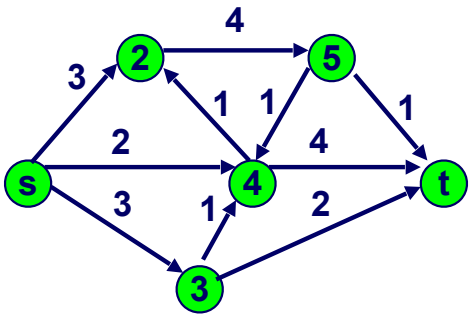
Select, then relabel/push



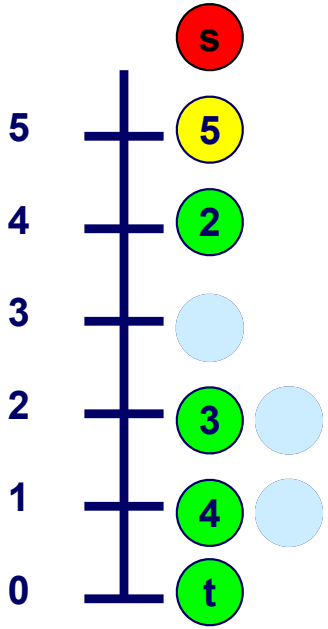


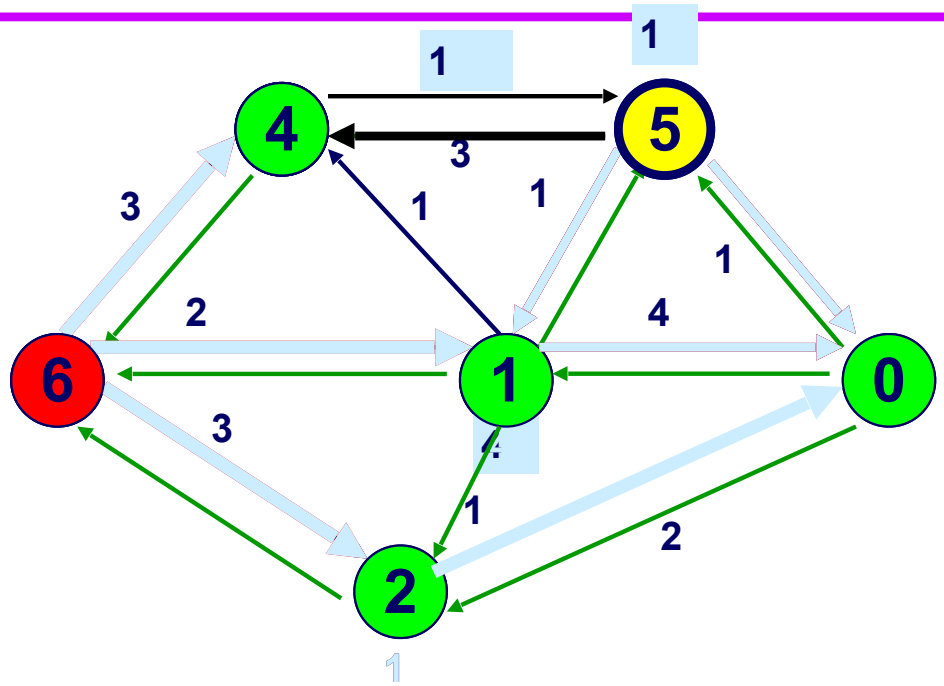
Select an active node.

Push/Relabel



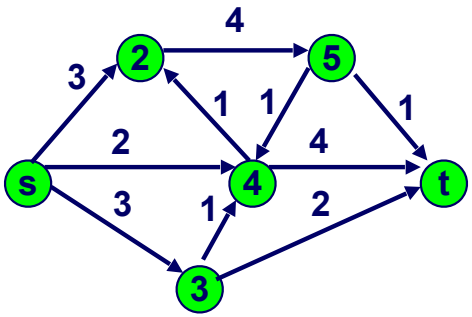
Select, then relabel/push





One can keep pushing flow between nodes 2 and 5 until eventually all flow returns to node s.

There are ways to speed up the last iterations.



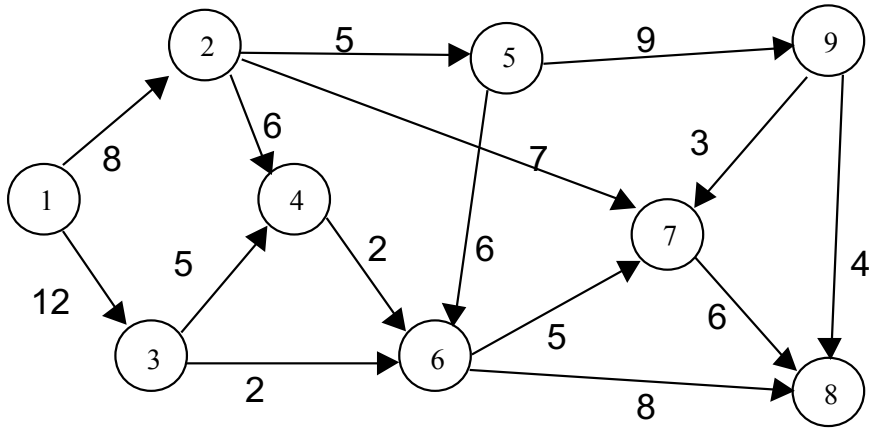
Properties

- Concept: a saturating push is when flow is augmented equal to residual edge capacity r(v,w)
  - Edge (v,w) is removed from residual graph
- Main results:
  - d(v) < 2|V| throughout the algorithm
  - The algorithm terminates with a maximum flow
  - The number of steps excluding non-saturating pushes is $O(|V||E|)$
  - The number of non-saturating pushes is $O(|V|^2|E|)$
  - Analysis involves potential functions — not in CLRS
- Can improve complexity by scaling, and choosing active nodes wisely
  - e.g. push from maximum distance label: $O(|V|^2|E|^{1/2})$

Another Example

- Same as Edmonds-Karp algorithm earlier

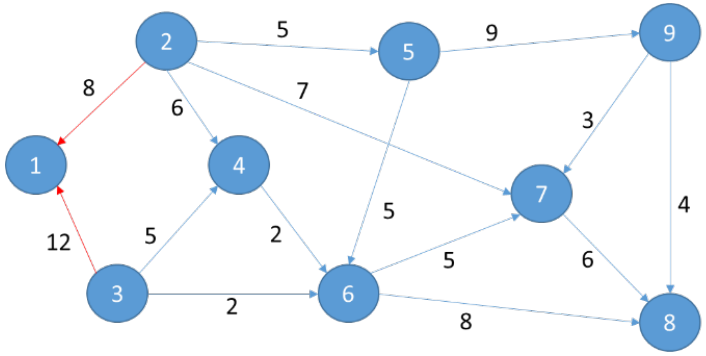
Vertex	Excess	d( )
1	0	3
2	0	2
3	0	2
4	0	2
5	0	2
6	0	1
7	0	1
8	0	0
9	0	1



Another Example

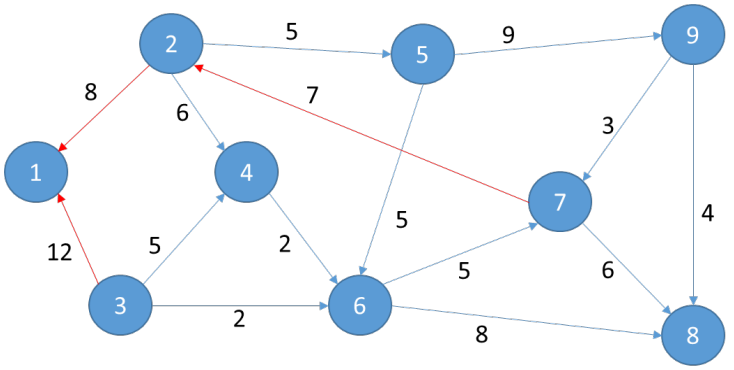
- Push out of s, relabel

Vertex	Excess	d( )
1	-20	9
2	8	2
3	12	2
4	0	2
5	0	2
6	0	1
7	0	1
8	0	0
9	0	1



- Push 7 out of 2, on (2,7)

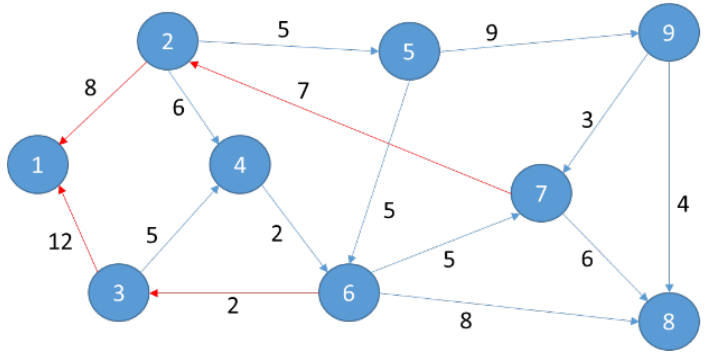
Vertex	Excess	d( )
1	-20	9
2	1	2
3	12	2
4	0	2
5	0	2
6	0	1
7	7	1
8	0	0
9	0	1



Another Example

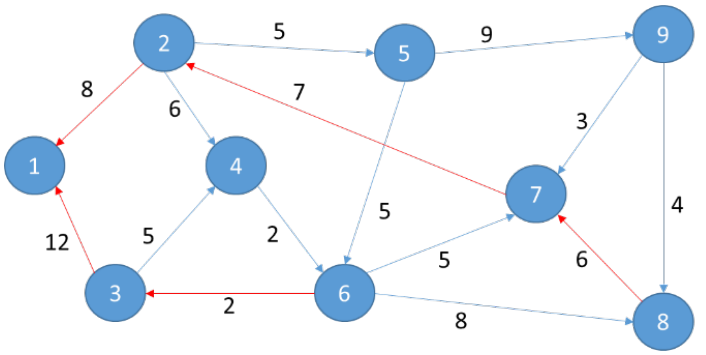
- Push 2 out of 3, on (3,6)

Vertex	Excess	d( )
1	-20	9
2	1	2
3	10	2
4	0	2
5	0	2
6	2	1
7	7	1
8	0	0
9	0	1



- Push 6 out of 7, on (7,8)

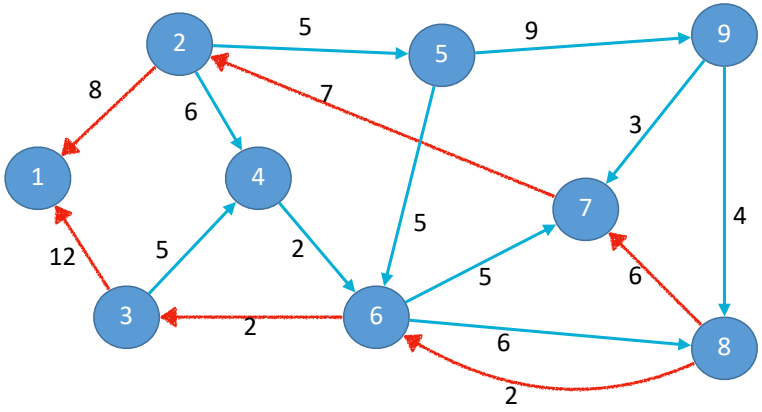
Vertex	Excess	d( )
1	-20	9
2	1	2
3	10	2
4	0	2
5	0	2
6	2	1
7	1	1
8	6	0
9	0	1



Another Example

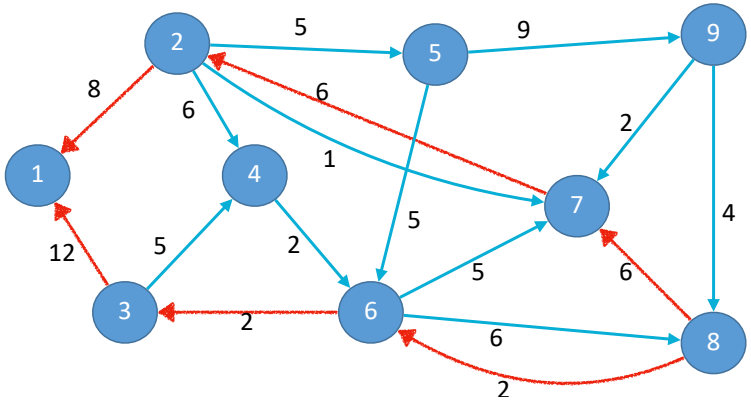
- Push 2 out of 6, on (6,8) (non-saturating)

Vertex	Excess	d( )
1	-20	3
2	1	2
3	10	2
4	0	2
5	0	2
6	0	1
7	1	1
8	8	0
9	0	1



- Relabel 7, increase d to 3 push 1 on (7,2) non-saturating

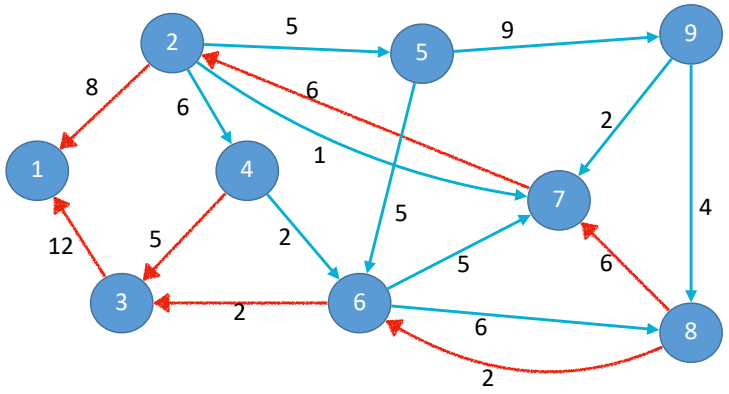
Vertex	Excess	d( )
1	-20	3
2	2	2
3	10	2
4	0	2
5	0	2
6	0	1
7	0	3
8	8	0
9	0	1



Another Example

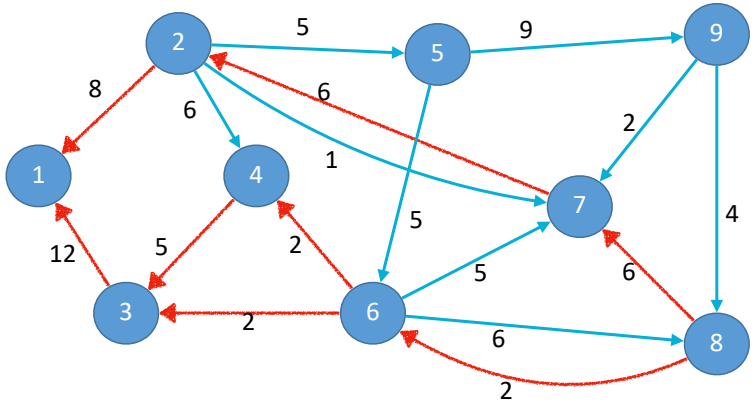
- Relabel 3 to 3, push 5 on (3,4)

Vertex	Excess	d( )
1	-20	9
2	2	2
3	5	3
4	5	2
5	0	2
6	0	1
7	0	3
8	8	0
9	0	1



- Push 2 on (4,6)

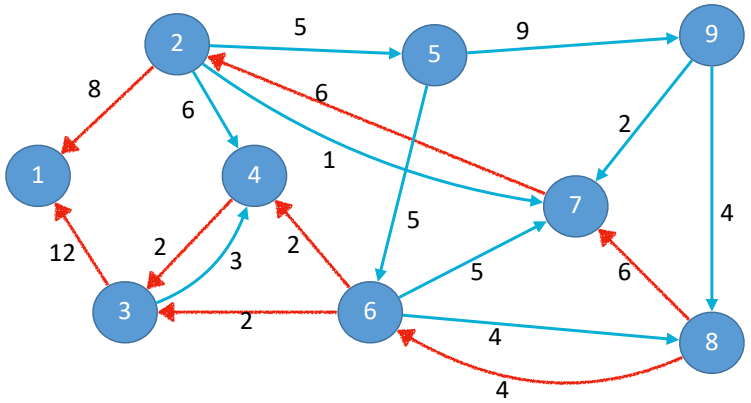
Vertex	Excess	d( )
1	-20	9
2	2	2
3	5	3
4	3	2
5	0	2
6	2	1
7	0	3
8	8	0
9	0	1



Another Example

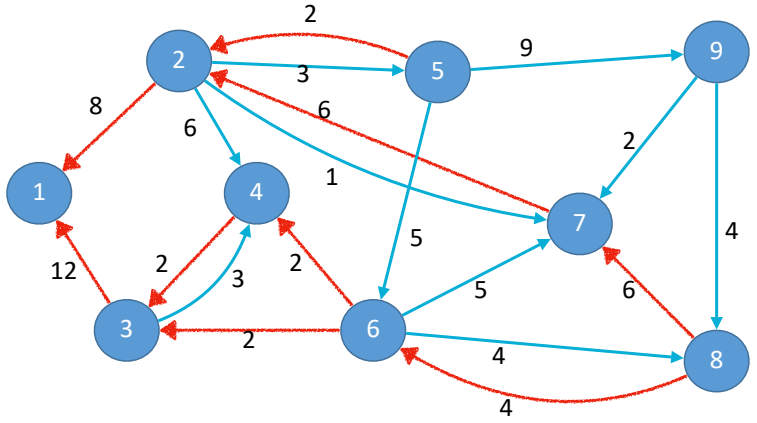
- Relabel 4 to 4. Push 3 on (4,3), non-saturating

Vertex	Excess	d( )
1	-20	9
2	2	2
3	8	3
4	0	4
5	0	2
6	0	1
7	0	3
8	10	0
9	9	1



- Relabel 2 to 3 push 2 on (2,5), non-saturating

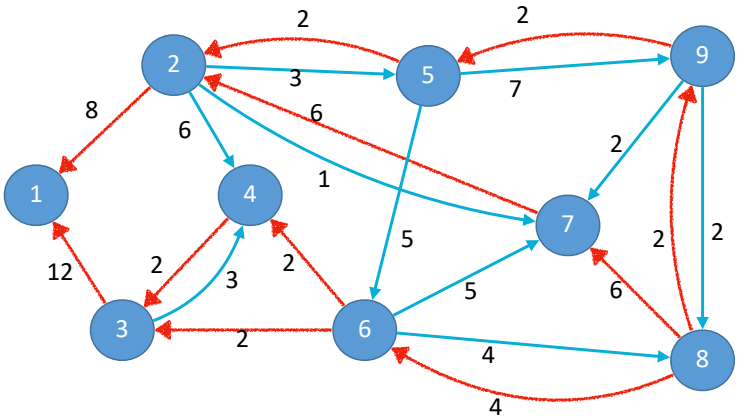
Vertex	Excess	d( )
1	-20	9
2	0	3
3	8	3
4	0	4
5	2	2
6	0	1
7	0	3
8	10	0
9	0	1



Another Example

- Push 2 on (5,9), then push 2 on (9,8)

Vertex	Excess	d( )
1	-20	3
2	0	3
3	8	3
4	0	4
5	0	2
6	0	1
7	0	3
8	12	0
9	0	1



- Relabel 3 to 5, then push 3 on (3,4)
  DONE.
  4 relabels to 6, creates a hole in distances, so all excess returns to s

Vertex	Excess	d( )
1	20	3
2	0	3
3	5	5
4	3	4
5	0	2
6	0	1
7	0	3
8	12	0
9	0	1

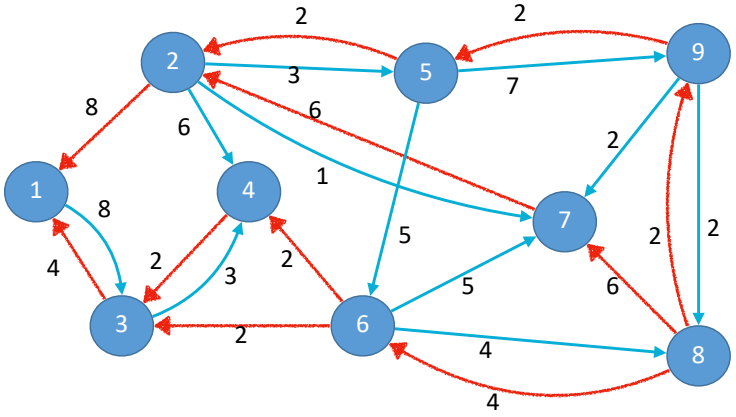


Image Segmentation as Max-Flow

- Problem: Separate foreground from background
  - More general: separate images into coherent regions
  - But, will focus on two regions
-

Approach

- Foreground / background segmentation:
  - Label each pixel in picture as belonging to foreground or background
  - V = set of pixels, E = pairs of neighboring pixels
  - We have statistical models for the values of background pixels, foreground pixels
    - Perhaps learned from data, or from the specific image
  - $a_i \geq 0$ is a measure that the pixel fits the foreground model (the negative log of a probability, or a likelihood)
  - $b_i \geq 0$ is a measure that the pixel fits the background model (the negative log of a probability, or a likelihood)
  - $p_{ij} \geq 0$ is a smoothness penalty for labeling neighbor pixels i, j with different labels

Goal: Find partition (A,B) of pixels that maximizes $\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E : i \in A, j \in B} p_{ij}$

Approach

- Idea: Convert to a minimization problem, formulate as graph cut

  maximize $\sum_{i \in A} a_i + \sum_{j \in B} b_j - \sum_{(i,j) \in E : i \in A, j \in B} p_{ij}$

  Convert to minimize

$$\sum_{i \in V} (a_i + b_i) - \sum_{i \in A} a_i - \sum_{j \in B} b_j + \sum_{(i,j) \in E : i \in A, j \in B} p_{ij} = \sum_{i \in B} a_i + \sum_{j \in A} b_j + \sum_{(i,j) \in E : i \in A, j \in B} p_{ij}$$

- Pose the problem now as a min-cut problem with capacities corresponding to non-negative terms above

Segmentation as Energy Optimization

$$E(\mathbf{x}) = \sum_{i \in \mathcal{V}} \psi_i(x_i) + \sum_{i \in \mathcal{V}, j \in \mathcal{N}_i} \psi_{ij}(x_i, x_j)$$

Data term **Smoothness term**

Data term

$$b_i = \psi_i(x_i = 0) = -\log(p(x_i \notin FG))$$
$$a_i = \psi_i(x_i = 1) = -\log(p(x_i \in FG))$$

Estimated using FG / BG color models

Smoothness term

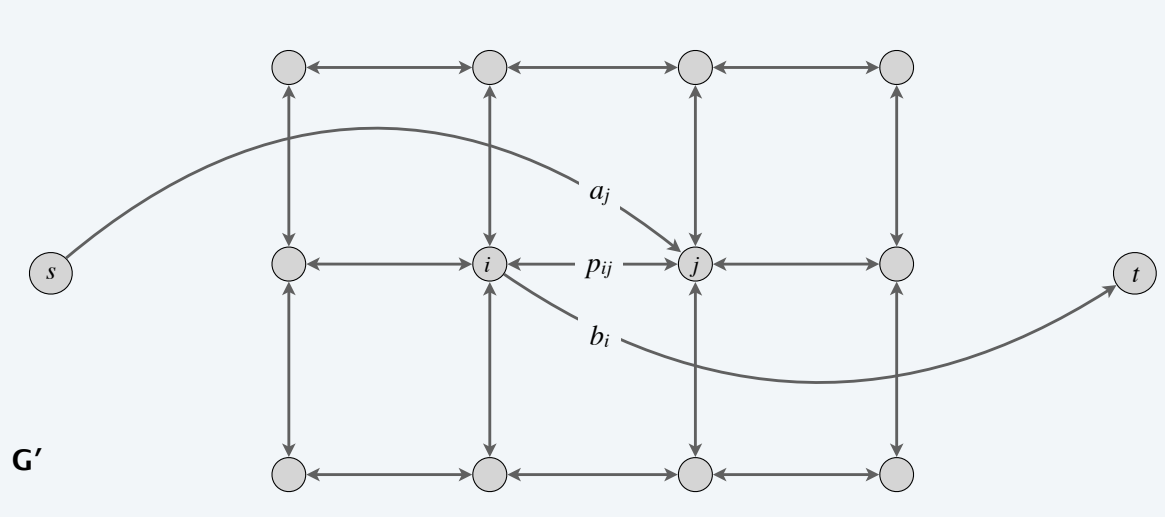
$$p_{ij} = \psi_{ij}(x_i, x_j) = K_{ij}\delta(x_i \neq x_j)$$

where $K_{ij} = \lambda_1 + \lambda_2 \exp(-\beta(I_i - I_j)^2)$

Intensity dependent smoothness

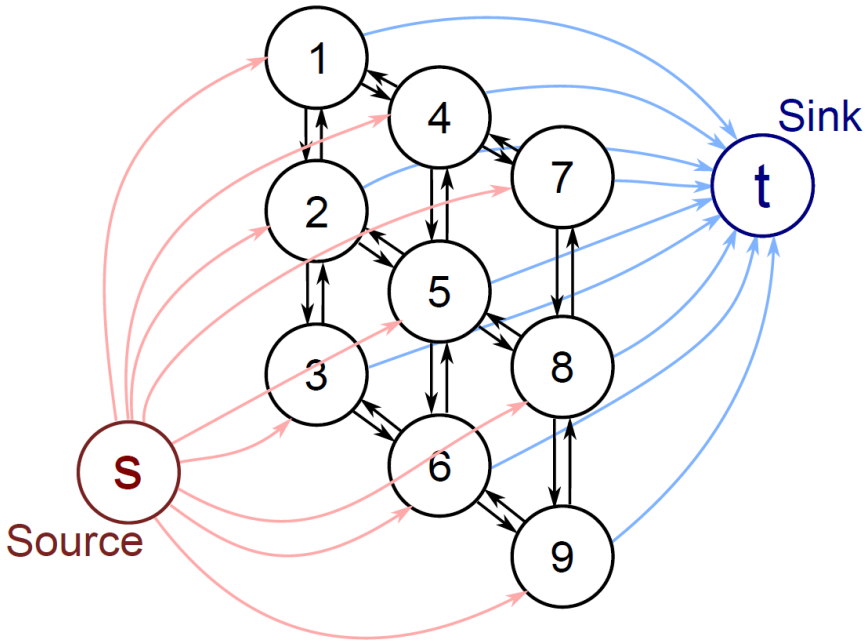
Min-Cut Formulation

- Include vertex for each pixel
- For every neighbor pixels {i,j}, add edges (i,j) with capacity p_{ij} and (j,i) with capacity p_{ji}
- Add source vertex s with directed edges (s,j) for every pixel j, with capacity a_j
- Add sink vertex t with directed edges (j,t) for every pixel j, with capacity b_j



Graph Construction

- One node per pixel (here a 3x3 image)
- Edge from source to every pixel node
- Edge from every pixel node to sink
- Reciprocal edges between neighbors

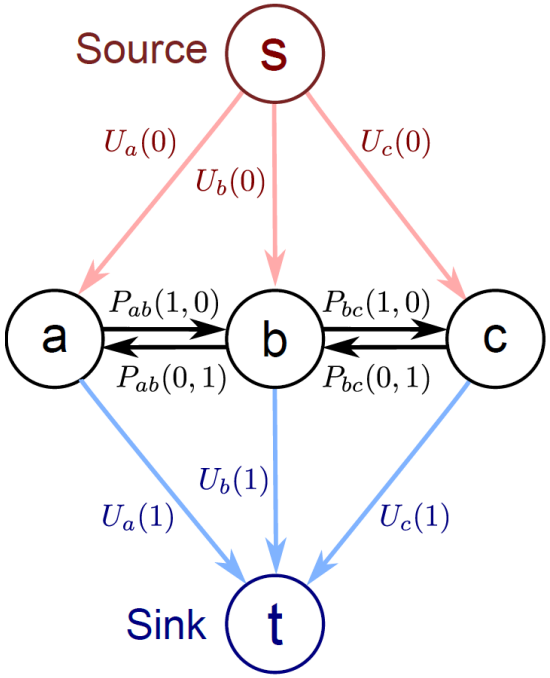


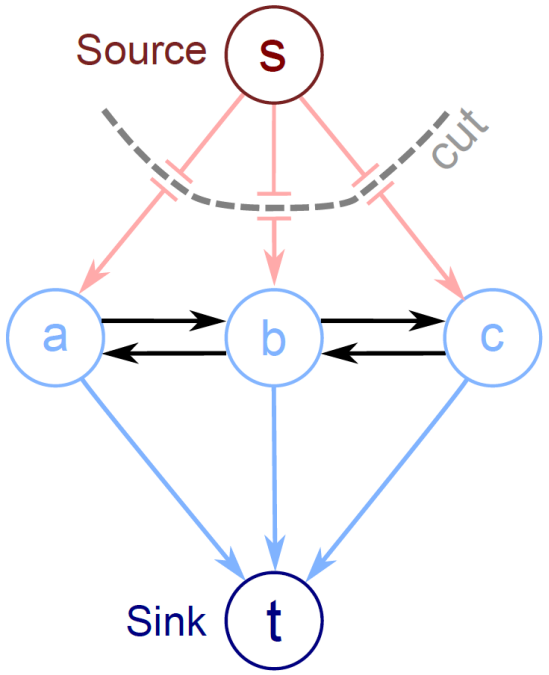
Note that in the minimum cut EITHER the edge connecting to the source will be cut, OR the edge connecting to the sink, but NOT BOTH (unnecessary).

Which determines whether we give that pixel label 1 or label 0.

Now a 1 to 1 mapping between possible labeling and possible minimum cuts

Example 1





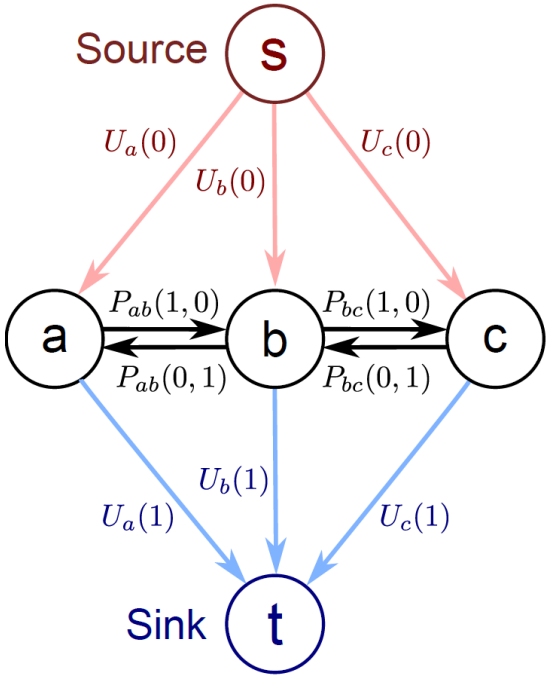
Solution

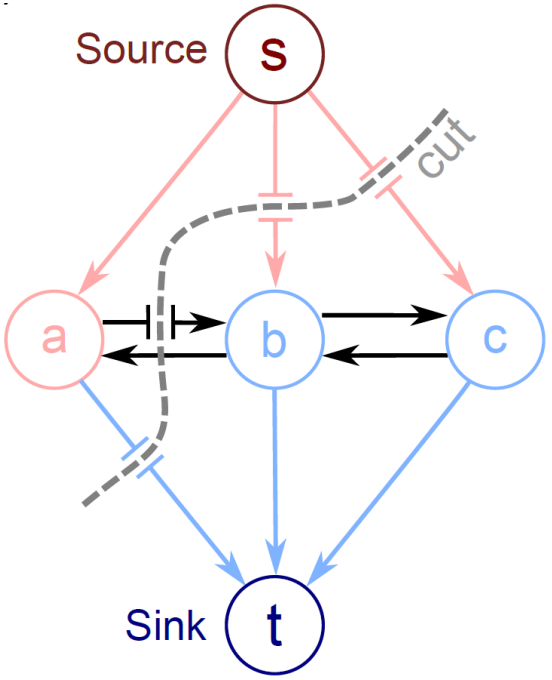
0	0	0

Cost

$$U_a(0) + U_b(0) + U_c(0)$$

Example 2





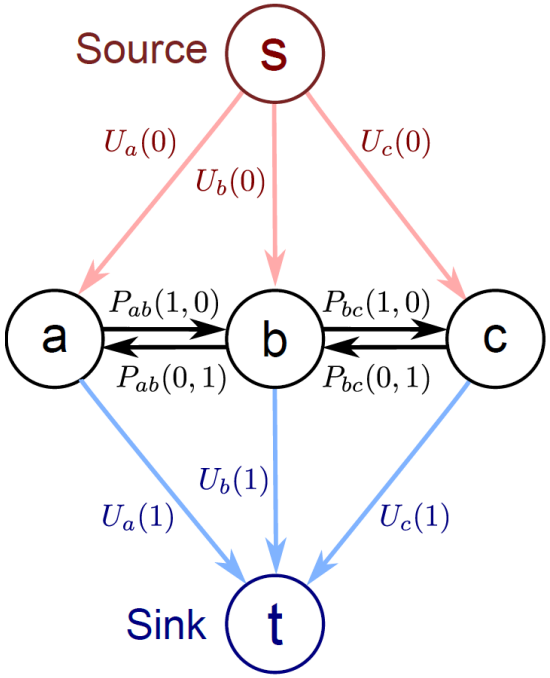
Solution

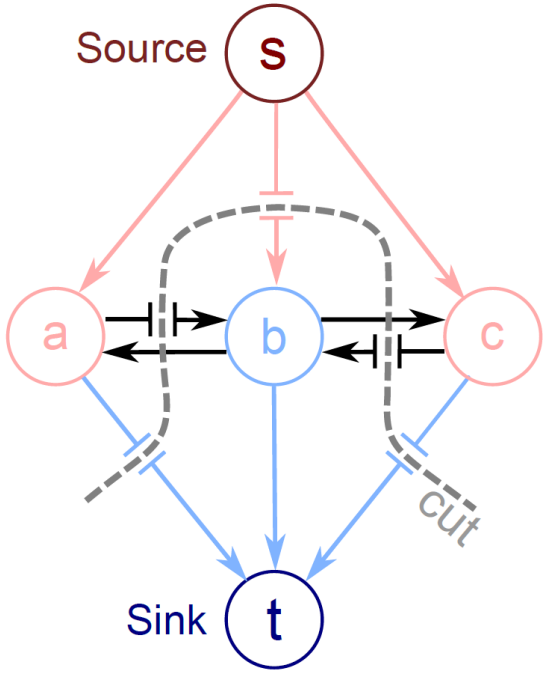
1	0	0

Cost

$$U_a(1) + U_b(0) + U_c(0) + P_{ab}(1,0)$$

Example 3





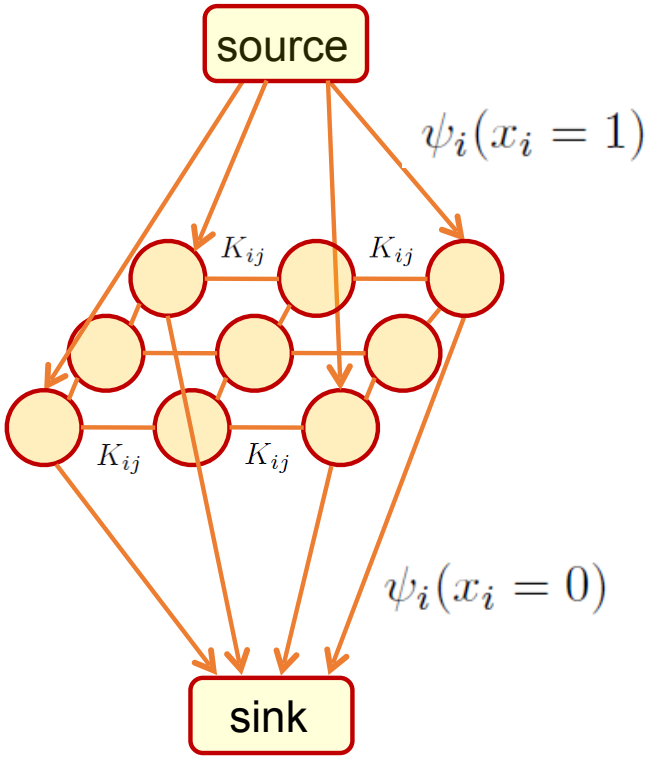
Solution

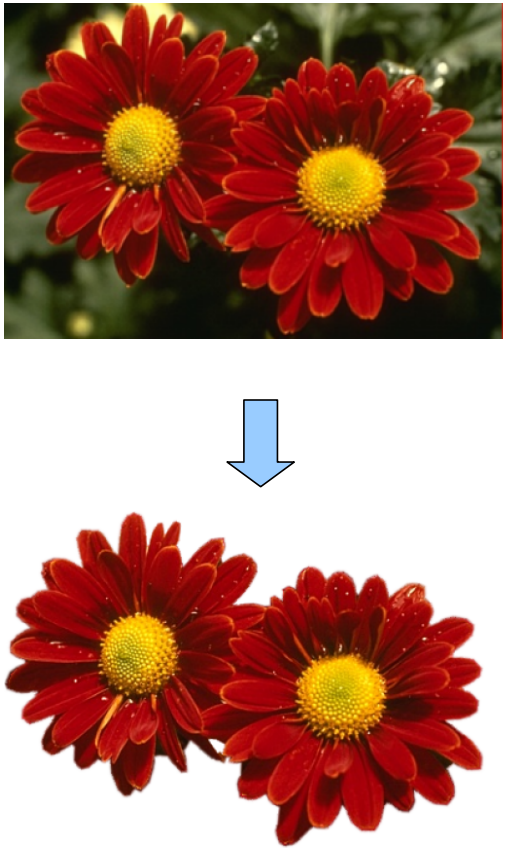
1	0	1

Cost

$$U_a(0) + U_b(0) + U_c(0)$$
$$+P_{ab}(1,0) + P_{bc}(0,1)$$

Foreground / Background Estimation





Rother et al. SIGGRAPH04

Application

- Grabcut (Rother-Kolmogorov-Blake ’04)
- User selects regions of image that includes the foreground
- Statistical models of foreground, background generated based on selection
- Used to compute weights in max-flow, min-cut graph to segment foreground in selected area