CS340 - Analysis of Algorithms

Network Flow II
Extensions and Reductions

Announcements:

HW7 Due Monday November 17

No office hours Friday

- Today 2:30-3:30 instead

Final Exam?

Agenda

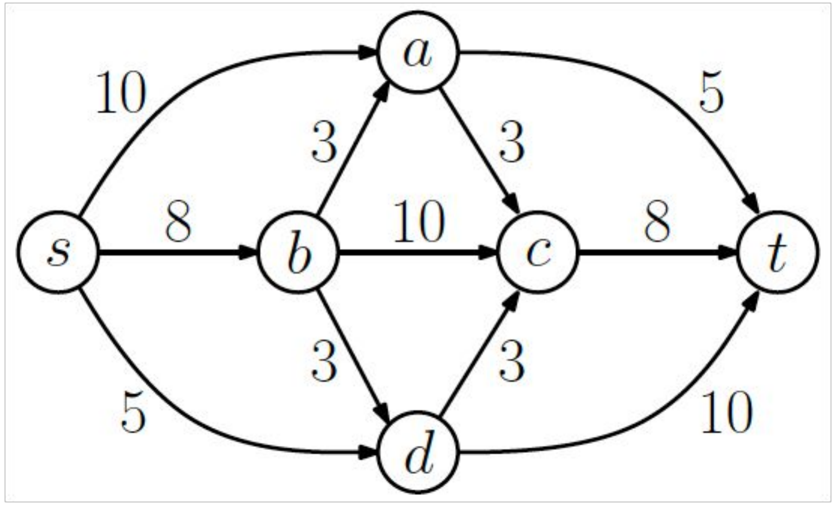
1. Network Flow Review
2. Ford-Fulkerson Runtime Analysis
3. Reductions to Network Flow
    a. Max Bipartite Matching

Flow Network

- Directed graph G = (V,E)
- Each edge (u,v) ∈ E has a non-negative *capacity* c(u,v) ≥ 0
- If (u,v) ∉ E, c(u,v) = 0

- Source node: s
- Sink node: t

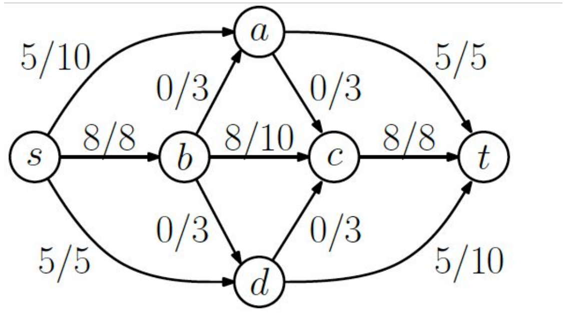
- Called an “s-t network”



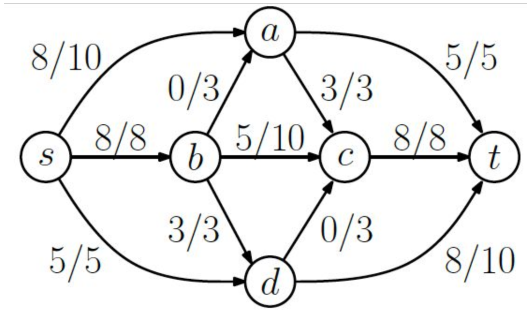
Maximum Flow Problem

- Given a flow network, arrange the traffic to make as efficient use as possible of the available capacity
- Total flow of a network:

$$|f| = f^{out}(s) = \sum_{w \in V} f(s,w)$$

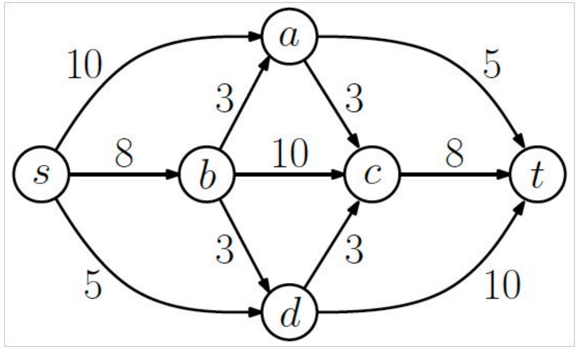


A valid flow ($|f| = 18$)



A maximum flow ($|f| = 21$)

Ford-Fulkerson



```
ff(G=(V, E, s, t)) {
  f = 0 (all edges carry 0 flow)
  while (true) {
    G' = residual-network of G for f
    if (G' has no s-t augmenting path) break
    P = any-augmenting-path of G'
    c = min capacity of P
    augment by adding c to every edge of P in G
    update f
  }
  return f
}
```

Runtime Analysis

```
ff(G=(V, E, s, t)) {
  f = 0 (all edges carry 0 flow)
  while (true) {
    G' = residual-network of G for f
    if (G' has no s-t augmenting path) break
    P = any-augmenting-path of G'
    c = min capacity of P
    augment by adding c to every edge of P in G
    update f
  }
  return f
}
```

Runtime Complexity of Augmentation

1. Compute Residual Network (G_f)
   a. O(n+m)
2. Find augmenting path P in G_f
   a. DFS or BFS on G_f starting at S
   b. O(n+m)
3. Compute min-cost edge along P
   a. O(m)
4. Increase flow for every edge in P
   a. O(m)

For network flow the graph must be connected.

$m > n-1$

Each augmentation is O(m)

Termination of Ford-Fulkerson

```
ff(G=(V, E, s, t)) {
  f = 0 (all edges carry 0 flow)
  while (true) {
    G’ = residual-network of G for f
    if (G’ has no s-t augmenting path) break
    P = any-augmenting-path of G’
    c = min capacity of P
    augment by adding c to every edge of P in G
    update f
  }
  return f
}
```

1. At each intermediate stage of the algorithm, the flow values and the residual capacities are integers
2. Flow values strictly increase when we apply augmentation
3. There is an upper bound on the maximum possible flow value

Termination of Ford-Fulkerson

Lemma 1: At each intermediate stage of the algorithm, the flow values and the residual capacities are integers.

Proof by Induction:

Base case: before any iterations of the while loop, the flow values are all 0. The residual graph contains forward edges equal to the capacities which are also integers.

IH: For j iterations such that $0 < j < n$, the flow values and residual capacities are all integers.

We will use this to show that at $n=j+1$ iterations, the flow values and residual capacities are still integers.

The min capacity of the augmenting path chosen at each iteration is also an integer.

So, the updated G and updated G_f contain only integers.

Termination of Ford-Fulkerson

$$|f| = f^{out}(s)$$

Lemma 2: The flow value strictly increase when we apply augmentation

Proof:

The first edge e of P must be an edge out of s in G_f

Augmenting paths do not contain cycles, so there is no edge incoming to s.

We increase the flow on this edge in G, by value C_{min} which is a positive number, so the total flow increases.

Termination of Ford-Fulkerson

Lemma 3: There is an upper bound on the maximum possible flow value

Proof:

We will show that $C = \sum_{(s,v) \in E} c(u,v)$ is an upper bound on the flow value. (not a tight one!)

By definition of flow, flow must abide by edge capacity. Outgoing from s, we cannot possibly send more than C.

Termination of Ford-Fulkerson

Claim: For a flow network with integer capacities, the Ford-Fulkerson algorithm terminates in at most $C = \sum_{(s,v) \in E} c(u,v)$ iterations of the While loop.

Proof:

By lemma 2, the flow value strictly increases in each iteration.

By lemma 1, the flow value increases by at least 1 in each iteration.

Since the flow value starts at value 0 and cannot go higher than C, the while loop terminates in at most C iterations.

Termination of Ford-Fulkerson

```
ff(G=(V, E, s, t)) {
  f = 0 (all edges carry 0 flow)
  while (true) {
    G’ = residual-network of G for f
    if (G’ has no s-t augmenting path) break
    P = any-augmenting-path of G’
    c = min capacity of P
    augment by adding c to every edge of P in G
    update f
  }
  return f
}
```

O(Cm)

Runtime Complexity of Augmentation

1. Compute Residual Network (G_f)
   a. O(n+m)
2. Find augmenting path P in G_f
   a. DFS or BFS on G_f starting at S
   b. O(n+m)
3. Compute min-cost edge along P
   a. O(m)
4. Increase flow for every edge in P
   a. O(m)

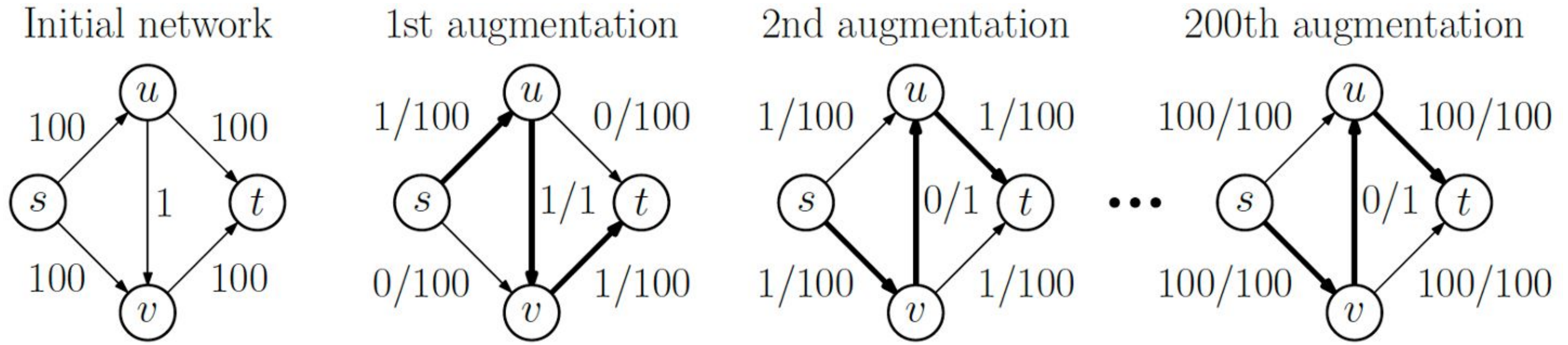
For network flow the graph must be connected.

$n < m < n^2$

Each augmentation is O(m)

Number of Augmentations

To do worst case C augmentations, each augmentation would need to increase flow by only 1



Which augmentation path we choose at each iteration affects the runtime!

Number of Augmentations

- Augmentation increases the value of the flow by the min capacity (C_{min}) of the residual path
- If we choose paths with large C_{min}, we will be making more progress than paths with smaller C_{min}
- What if we always chose the augmentation path with the largest C_{min}? Would that be too expensive?
    - Modified Dijkstra's where we maximize minimum capacity rather than minimizing distance (priority queue key)
    - O(mlogn) at each iteration since the residual graph edges change

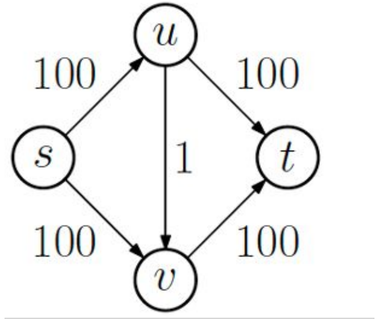
Scaling Algorithm

- Finding path with the largest C_{min} costs O(mlogn) at each iteration
- Instead of finding largest C_{min}, we will maintain a *scaling parameter* Δ and look for paths that have $C_{min} \geq \Delta$

```
ff_scaling(G = (V, E, s, t)):
    f = 0
    Δ = largest power of 2 ≤ C

    while (true) {
      G_f(Δ) = residual network of G where c(u,v) ≥ Δ for all edges (u,v)

      if G_f(Δ) has no s-t augmenting path and Δ > 1:
          Δ = Δ / 2 and continue

      P = any s-t augmenting path in G_f(Δ)
      C_min = min capacity of P
      augment by adding c to every edge of P in G
      update f
    }

return f
```

Scaling Algorithm

```
ff_scaling(G = (V, E, s, t)):
    f = 0
    $\Delta$ = largest power of 2 $\leq$ C

    while (true) {
     $G_f(\Delta)$ = residual network of G where c(u,v) $\geq$ $\Delta$ for all edges (u,v)

     if $G_f(\Delta)$ has no s-t augmenting path and $\Delta$ > 1:
         $\Delta$ = $\Delta$ / 2 and continue

     P = any s-t augmenting path in $G_f(\Delta)$
     $C_{min}$ = min capacity of P
     augment by adding c to every edge of P in G
     update f
    }

return f
```



```
$\Delta_1$ = 128
     Empty $G_f$!
$\Delta_2$ = 64
     $G_f$ does not include (u,v)
```

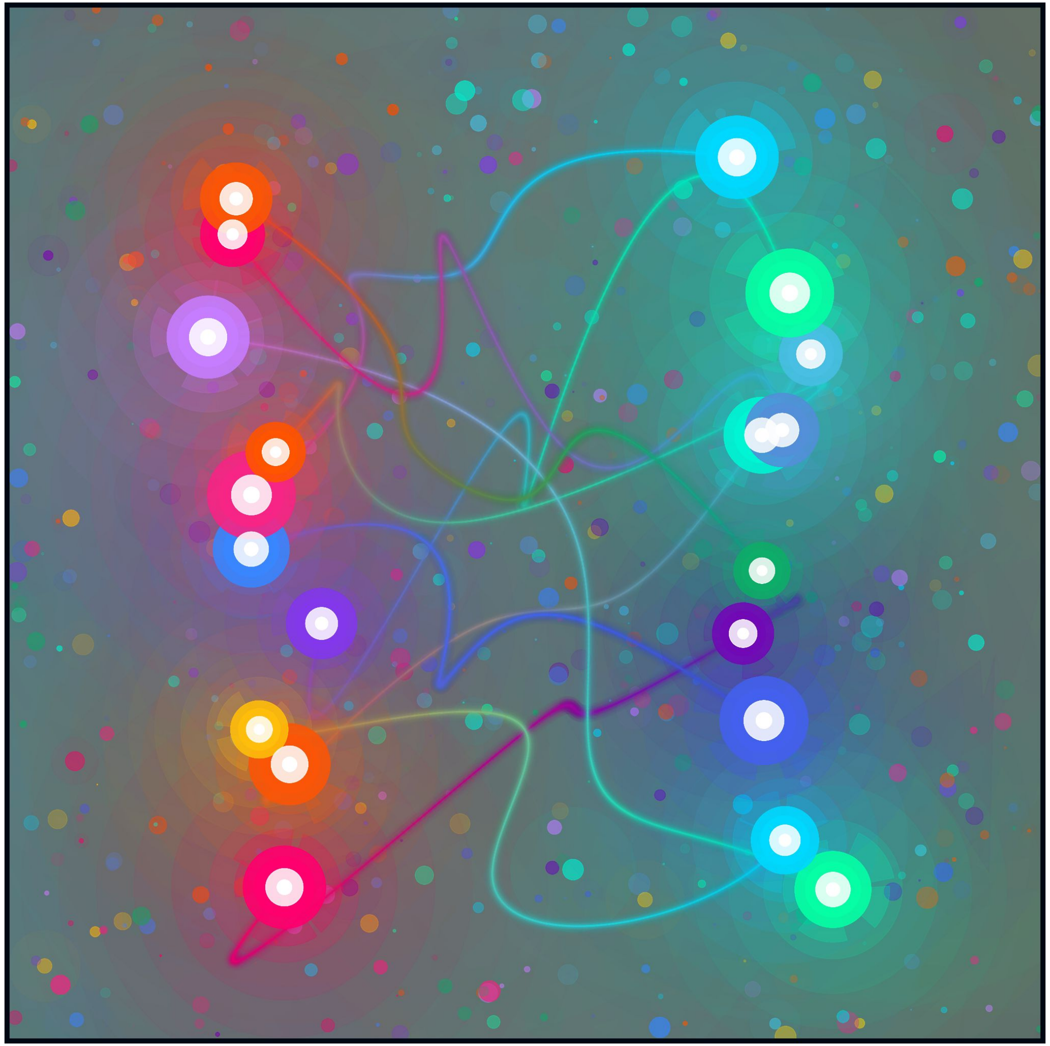
Scaling Algorithm Analysis

```
ff_scaling(G = (V, E, s, t)):
    f = 0
    Δ = largest power of 2 ≤ C

    while (true) {
     G_f(Δ) = residual network of G where c(u,v) ≥ Δ for all edges (u,v)

     if G_f(Δ) has no s-t augmenting path and Δ > 1:
         Δ = Δ / 2 and continue
     if G_f(Δ) has no s-t augmenting path and Δ == 1:
        break

     P = any s-t augmenting path in G_f(Δ)
     C_min = min capacity of P
     augment by adding c to every edge of P in G
     update f
    }

return f
```

1. Number of Δ values?
   a. At most 1 + log C
2. Number of augmentations done for each value of Δ?
   a. At most m
3. Each augmentation takes O(m)
   a. $O(m^2)$ runtime for each value of Δ
4. $O(m^2 \log C)$ overall

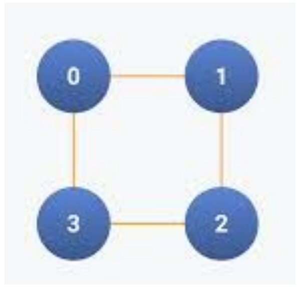
Max Bipartite Matching

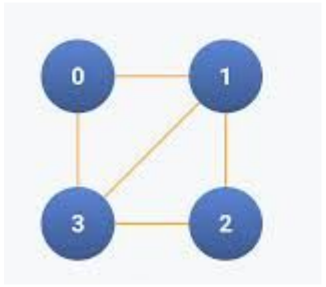
- An application of network flow
- A *bipartite graph* has two disjoint sets of vertices where edges only connect vertices from different sets

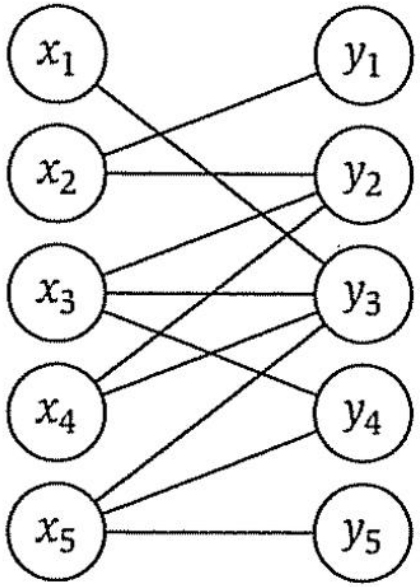


Bipartite Graphs

Is this graph bipartite?





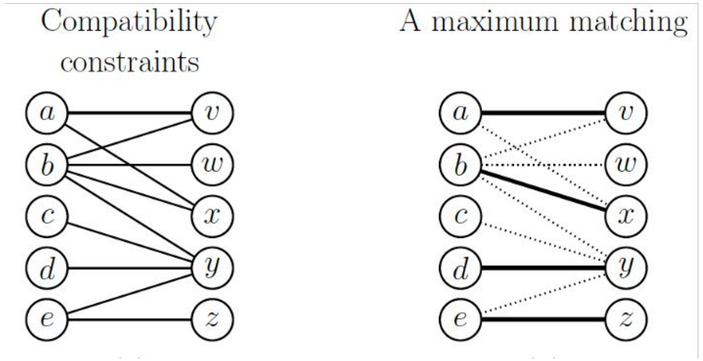


Pen Pal Matching service

- A set X of english speakers and a set Y of foreign language speakers
- Establish compatibility with questionnaires
- Pair up as many as possible
- One-to-one pairing
- Problem can be modeled as an undirected bipartite graph
- Vertex set is $V = X \cup Y$
- Edge set consists of pairs (u,v) where $u \in X$ and $v \in Y$ such that u and v are compatible

Max Bipartite Matching

- Given a graph $G = (V, E)$, a *matching* is a subset of edges $M \subseteq E$ such that for each $v \in V$, there is at most one edge of M incident to v.
- Find a maximum matching with the highest cardinality



Reduction to Network Flow

- Construct a flow network G’ = (V’, E’) as follows:
- Let s and t be two new vertices and let V’ = V ∪ {s, t}
- E’ = { (s,u) | u ∈ X) ∪

  (v,t) | v ∈ Y) U

  (u,v) | (u,v) ∈ E }
- Set all capacities equal to 1

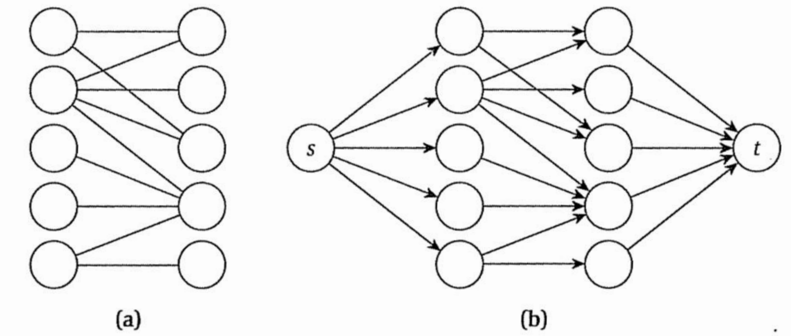
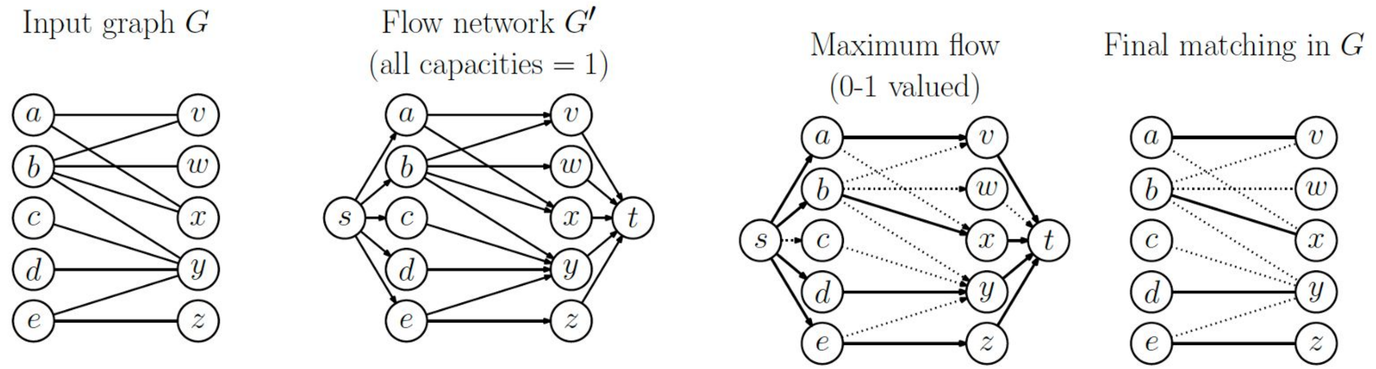


Figure 7.9 (a) A bipartite graph. (b) The corresponding flow network, with all capacities equal to 1.

Reduction to Network Flow

Find a max bipartite matching by:

1. Construct a G'
2. Compute a max flow f in G'
3. M = { (u,v) | u ∈ X, v ∈ Y, f(u,v) > 0 }



Time Analysis

- Plain Ford Fulkerson runs in O(Cm)
  - m = # of edges in the network flow graph
- Runtime is not simply number of 1*number of edges in the original bipartite graph
- We need to consider how the problem size grows when we reduce to Network Flow

- Runtime of constructing G'?
  - O(m+n)
- How large is V', the number of vertices in the network flow graph?
  - We add two nodes: s and t
  - n' = |V'| = |V| + 2
  - O(n)
- How large is E'?
  - We add edges from s to all nodes in X and edges from all nodes in Y to t
  - m' = |E'| = |E| + |V| = O(m+n)
- How large is C?
  - |X| < n
- $O(n*(m+n)) = O(mn + n^2)$

Proof of Correctness

We already know FF returns the max flow. We don’t need to prove that again.

Instead, we need to prove that solving max flow on G’ solves max matching on G

1. |M| on G ≤ |f| on G’
   a. If there exists a flow of size k on G’, then there exists a matching of size k on G
2. |M| on G ≥ |f| on G’
   a. If there exists a matching of size k, then there exists a flow of size k on G’

If and only if (both directions)

Correctness

Claim: A max matching of G has cardinality of k iff G’ has a maxflow of k

Proof Direction 1: |M| on G ≤ |f| on G’ (If there exists a flow of size k on G’, then there exists a matching of size k on G)

- Use an instance of a flow to create a matching

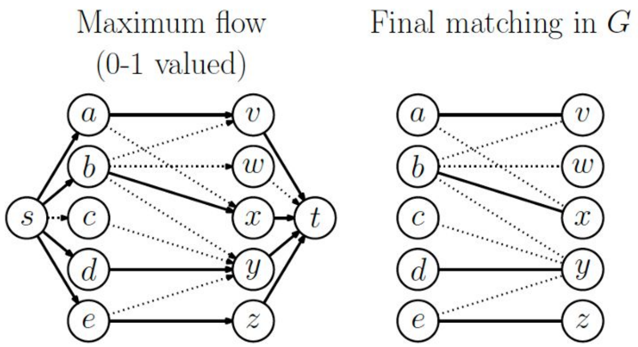
Let f be an s-t flow in G’ of value k.

Construct a matching M = { (u,v) | u ∈ X, v ∈ Y, f(u,v) > 0 }

M is a valid matching as no two edges in M share a vertex

|M| = |f| so the maxflow is at least |f|

|M| >= |f|



Correctness

Claim: A max matching of G has cardinality of k iff G’ has a maxflow of k

Proof Direction 2: |M| on G ≥ |f| on G’ (If there exists a matching of size k, then there exists a flow of size k on G’)

- Use an instance of a matching to create a flow

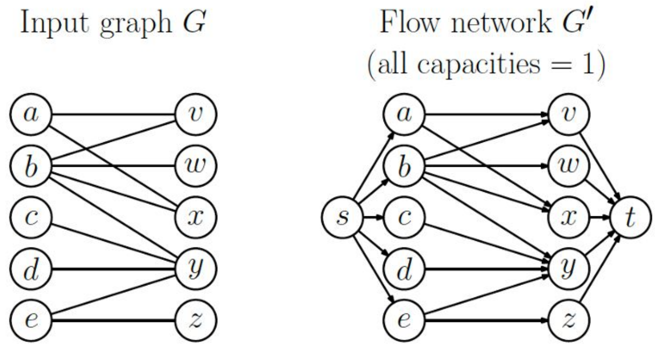
Construct a flow in G’ by sending 1 unit of flow along the path (s,x) (x,y),(y,t) for each edge (x,y) in the matching.

This is a valid flow as it respects conservation and capacity

Each path uses only 1 unit of flow and no two paths share an internal edge as M is a matching.

|f| = |M| so the max flow is at least |f|

|M| ≥ |f|



Network Flow Reductions

General advice:

1. Reduction Formulation:
    a. Describe how to build flow network G’ = (V’, E’)
        i. Specify vertices, edges, and edge capacities
    b. Describe how the maxflow value |f| of G’ relates to the solution to the original problem
2. Time Analysis:
    a. Consider how the problem size grows when we reduce to Network Flow
        i. Compute n’ and m’
    b. Consider the reduction graph construction time
    c. Consider total modified runtime of FF
3. Correctness of the reduction:
    a. If and only if equivalence proof
    b. |f| on G’ ≥ solving original problem
    c. |f| on G’ ≤ solving original problem

Summary

- Ford Fulkerson
  - Runtime of O(Cm)
  - With scaling, $O(m^2 \log C)$
- Reductions to Network Flow
  - Formulate the problem as an instance of network flow
  - Runtime analysis: consider how many nodes and edges there are in comparison to the original problem
  - Correctness: If and only if equivalence proof
- Hw7 due Monday