Data Structures and Algorithms

Session 19. April 6, 2009

Instructor: Bert Huang

http://www.cs.columbia.edu/~bert/courses/3137

Announcements

- Homework 4 due by midnight tonight
- Homework 5 assigned

Review

- Topological Sort
- Shortest Path
  - Unweighted version: Breadth-first search
  - Weighted version: Dijkstra's Algorithm

Today's Plan

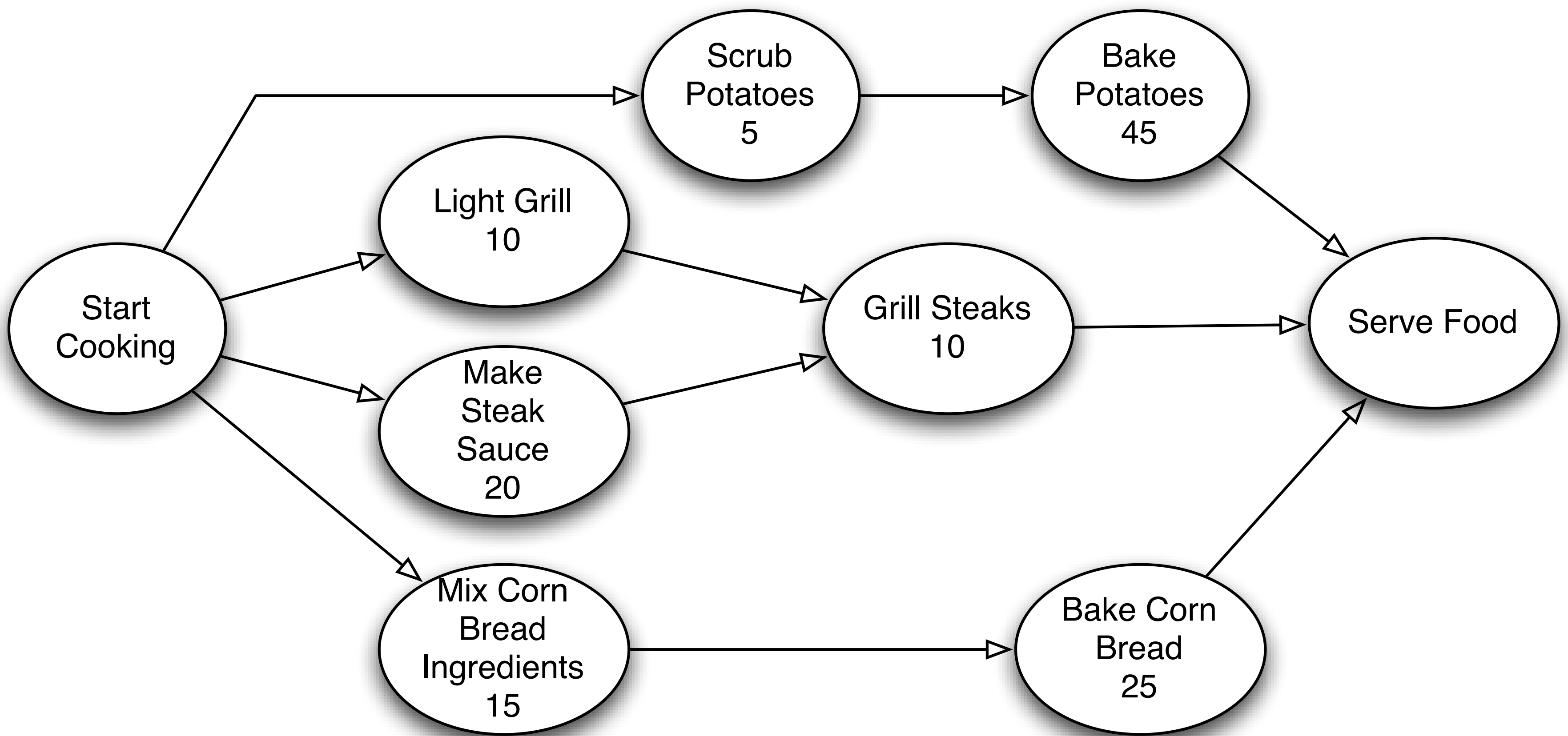
- Extensions of Dijkstra's Algorithm
  - Critical Path Analysis
  - All Pairs Shortest Path (Floyd-Warshall)
- Maximum Flow
  - Floyd-Fulkerson Algorithm

Critical Path Analysis

- Recall motivational example for topological sort: edges represent dependencies between tasks
- Consider a similar **event-node graph** in which nodes represent events and edges represent dependencies and costs
- We want to find the fastest time we can complete all tasks if we can run job in parallel

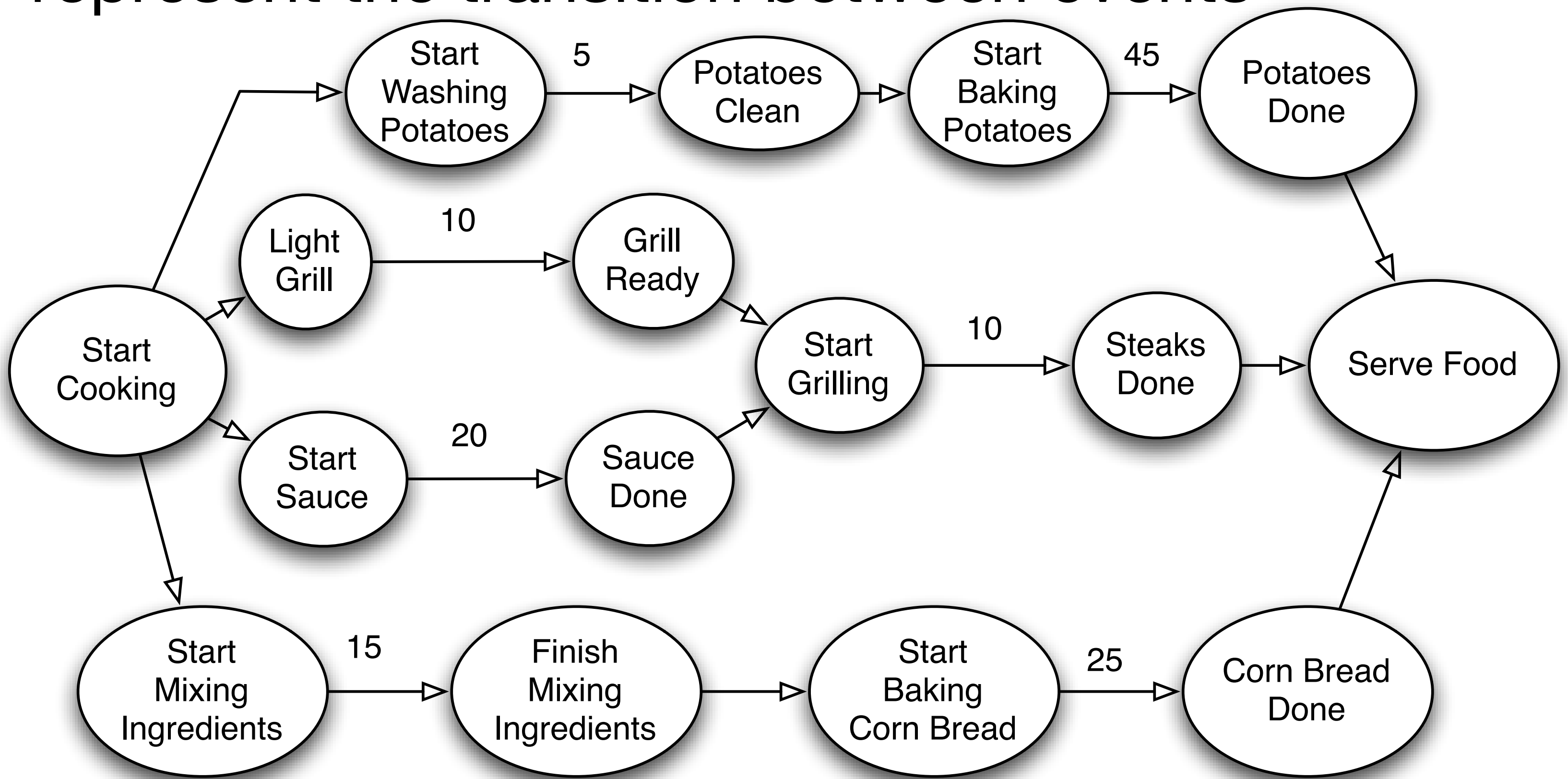
Critical Path Example

- We start with the activity graph, which includes time for each activity



Critical Path Example

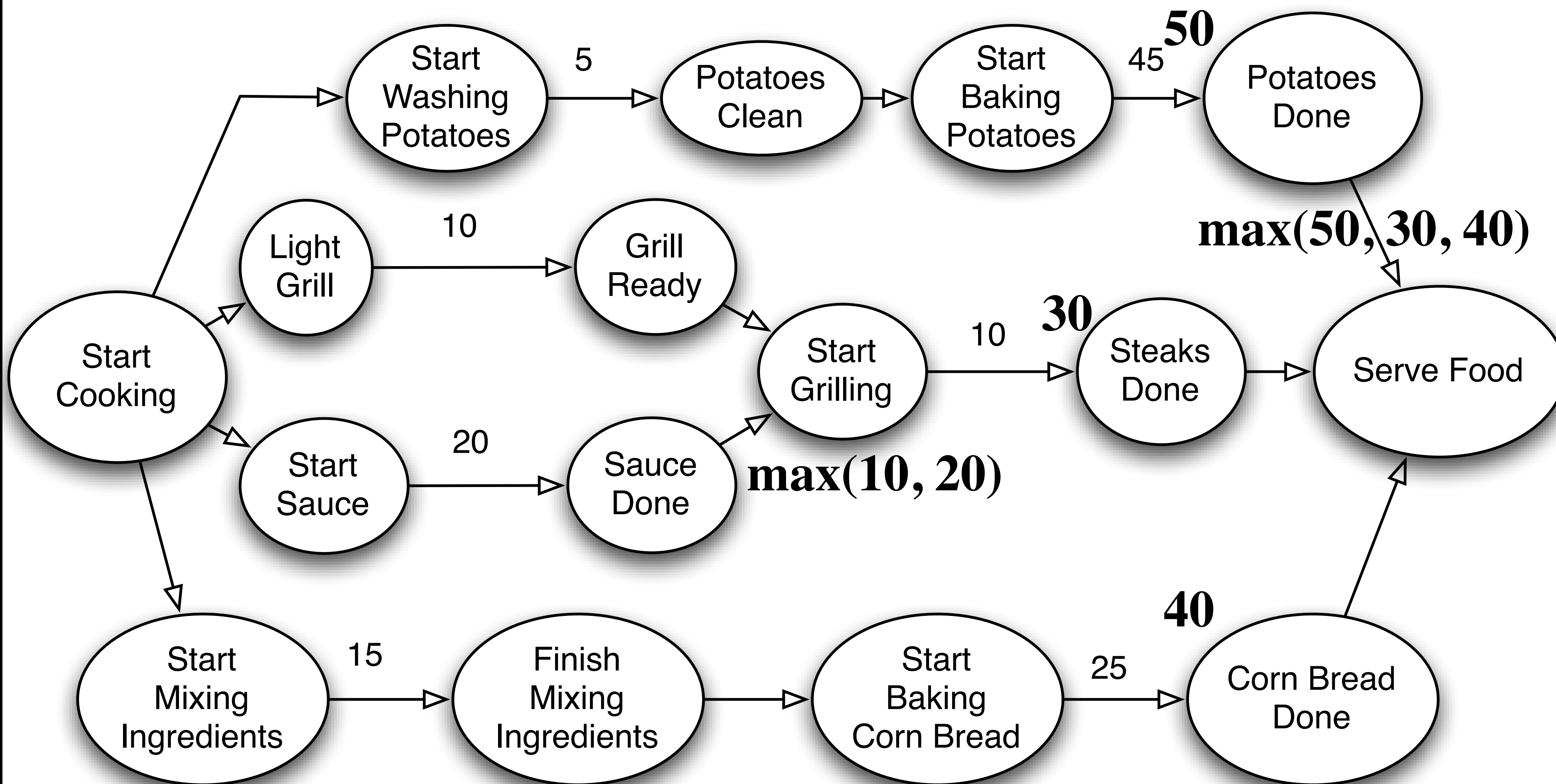
- Convert it to an event-node graph, where edges represent the transition between events



Longest Path in a DAG

- Store "longest known path" for each node
- Start node = 0
- Max of incoming nodes' longest known path + incoming edge cost
- Longest path from start to end is critical path

Critical Path for BBQ



Latest Completion Time

- If you want to procrastinate, compute the latest you can finish each job without delaying total time
- Set time of end node to critical path time
- Set nodes' latest completion time to:
  min of (outgoing node time) - (outgoing edge cost)
- (Similar to finding the shortest path to end following edges backwards)

All Pairs Shortest Path

- Dijkstra's Algorithm finds shortest paths from one node to all other nodes
- What about computing shortest paths for all pairs of nodes?
- We can run Dijkstra's |V| times. Total cost: $O(|V|^3)$
- Floyd-Warshall algorithm is often faster in practice (though same asymptotic time)

Recursive Motivation

- Consider the set of numbered nodes **1** through **k**
- The shortest path between any node **i** and **j** using only nodes in the set {**1**, ..., **k**} is the minimum of
  - shortest path from **i** to **j** using nodes {**1**, ..., **k-1**}
  - shortest path from **i** to **j** using node **k**
- path(i,j,k) = min(path(i,j,k-1),
  path(i,k,k-1)+ path(k,j,k-1))

Dynamic Programming

- Instead of repeatedly computing recursive calls, store lookup table
- To compute path(i,j,k) for any i,j, we only need to look up path(-,-, k-1)
  - but never k-2, k-3, etc.
- We can incrementally compute the path matrix for k=0, then use it to compute for k=1, then k=2...

Floyd-Warshall Code

- Initialize `d = ` weight matrix

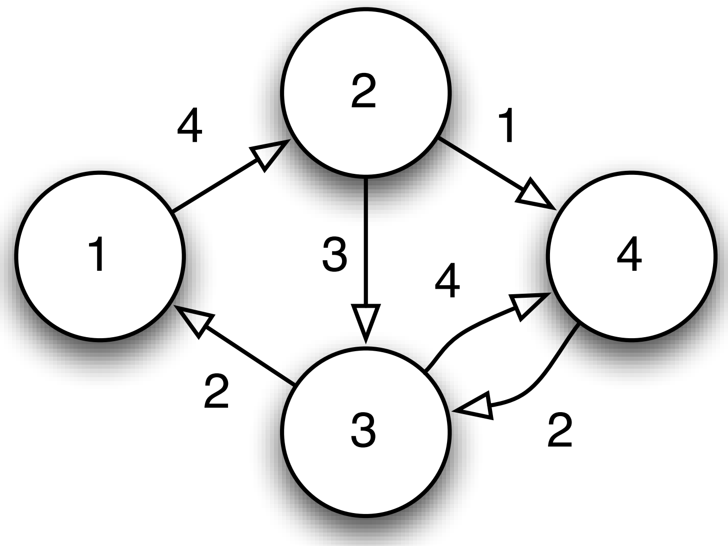
- ```
  for (k=0; k<N; k++)
    for (i=0; i<N; i++)
      for (j=0; j<N; j++)
        if (d[i][j] > d[i][k]+d[k][j])
          d[i][j] = d[i][k] + d[k][j];
  ```

- Additionally, we can store the actual path by keeping a “midpoint” matrix
  ```

All Pairs Shortest Path Example

K=0

	1	2	3	4
1	-	4	-	-
2	-	-	3	1
3	2	-	-	4
4	-	-	2	-



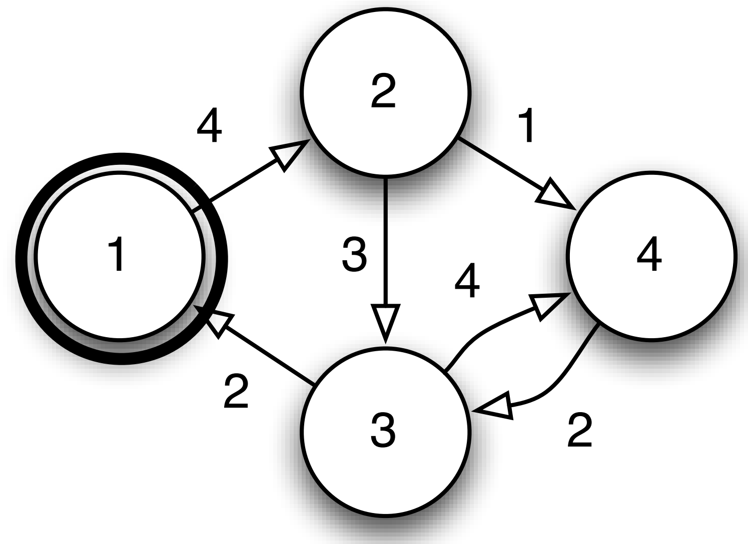
All Pairs Shortest Path Example

K=0

	1	2	3	4
1	-	4	-	-
2	-	-	3	1
3	2	-	-	4
4	-	-	2	-

K=1

	1	2	3	4
1	-	4	-	-
2	-	-	3	1
3	2	6	-	4
4	-	-	2	-



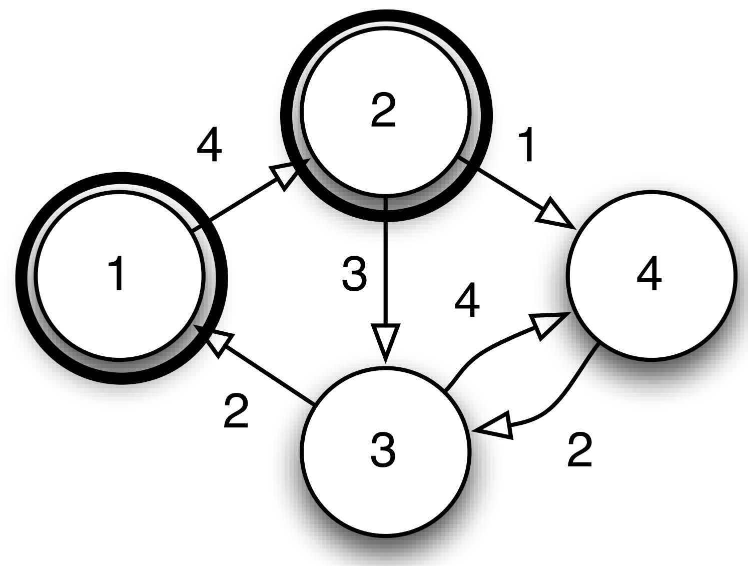
All Pairs Shortest Path Example

K=1

	1	2	3	4
1	-	4	-	-
2	-	-	3	1
3	2	6	-	4
4	-	-	2	-

K=2

	1	2	3	4
1	-	4	7	5
2	-	-	3	1
3	2	6	9	4
4	-	-	2	-



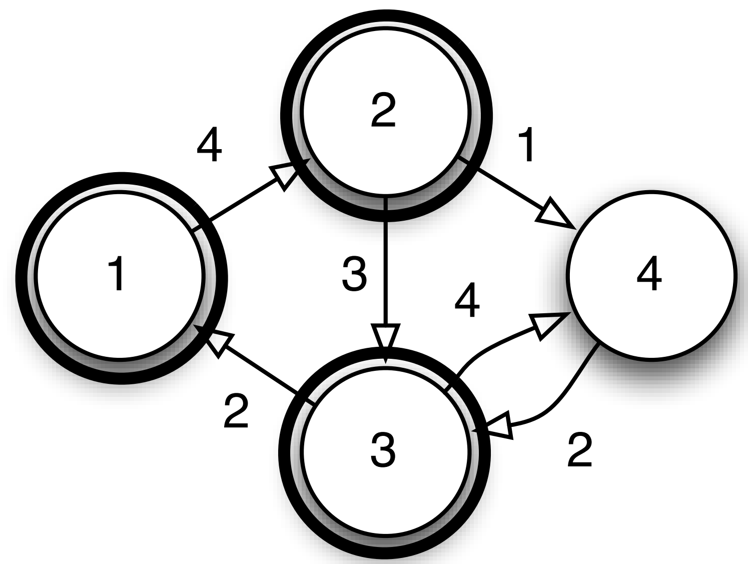
All Pairs Shortest Path Example

K=2

	1	2	3	4
1	-	4	7	5
2	-	-	3	1
3	2	6	9	4
4	-	-	2	-

K=3

	1	2	3	4
1	9	4	7	5
2	5	9	3	1
3	2	6	9	4
4	4	8	2	6



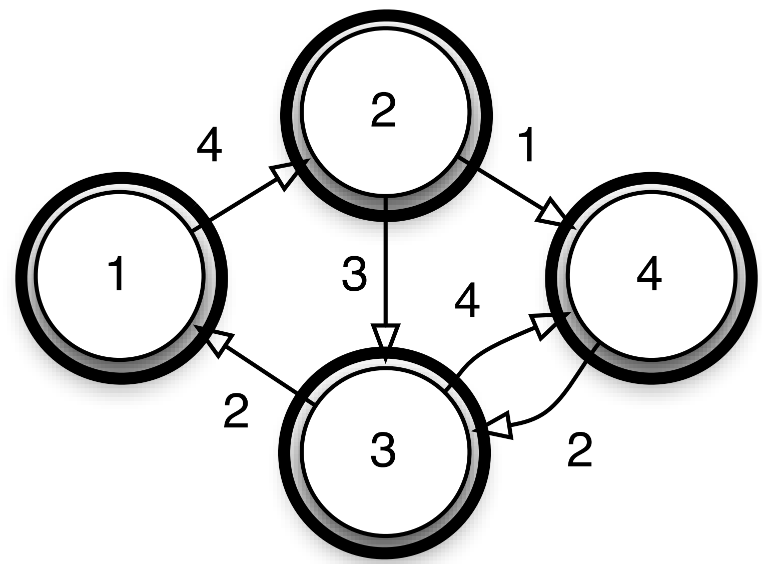
All Pairs Shortest Path Example

K=3

	1	2	3	4
1	9	4	7	5
2	5	9	3	1
3	2	6	9	4
4	4	8	2	6

K=4

	1	2	3	4
1	9	4	7	5
2	5	9	3	1
3	2	6	6	4
4	4	8	2	6



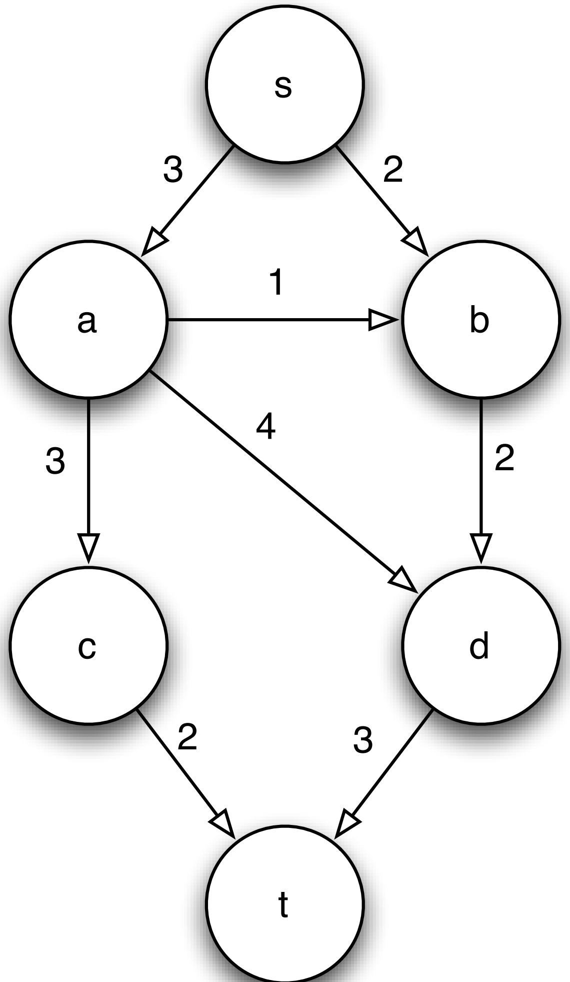
Transitive Closure

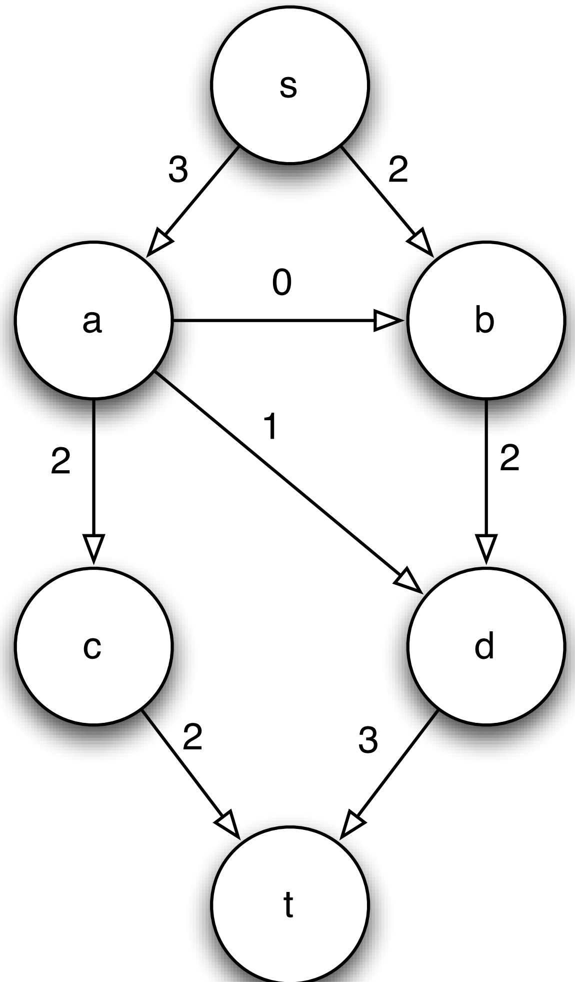
- For any nodes i, j, is there a path from i to j?
- Instead of computing shortest paths, just compute Boolean if a path exists
- path(i,j,k) = path(i,j,k-1) OR
  path(i,k,k-1) AND path(k,j,k-1)

Maximum Flow

- Consider a graph representing flow capacity
- Directed graph with **source** and **sink** nodes
- Physical analogy: water pipes
  - Each edge weight represents the **capacity**: how much “water” can run through the pipe from source to sink?

Capacity Example





MAXIMUM FLOW SOLUTION

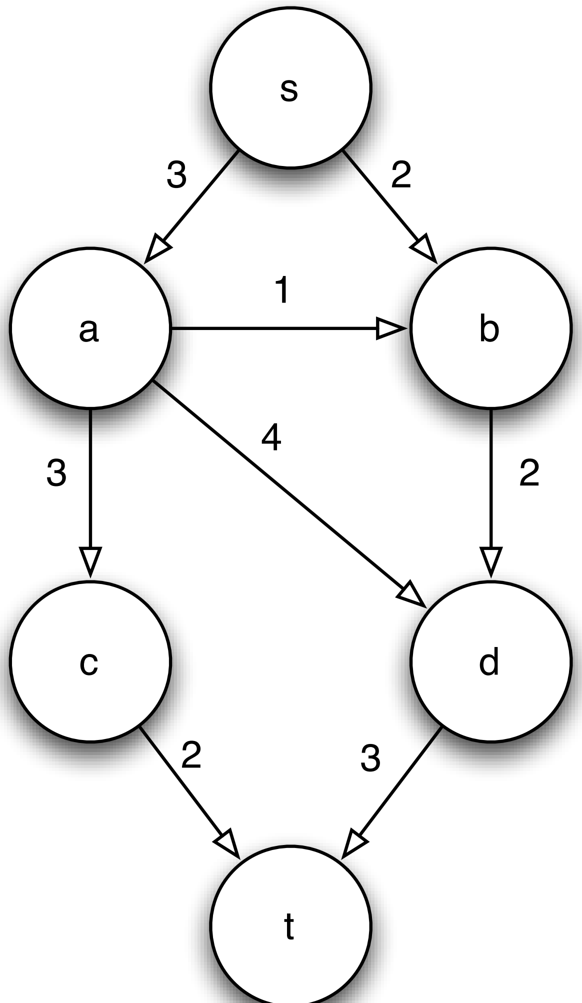
Max Flow Algorithm

- Create 2 copies of original graph: **flow graph** and **residual graph**
    - The flow graph tells us how much flow we have currently on each edge
    - The residual graph tells us how much flow is available on each edge
- Initially, the residual graph is the original graph

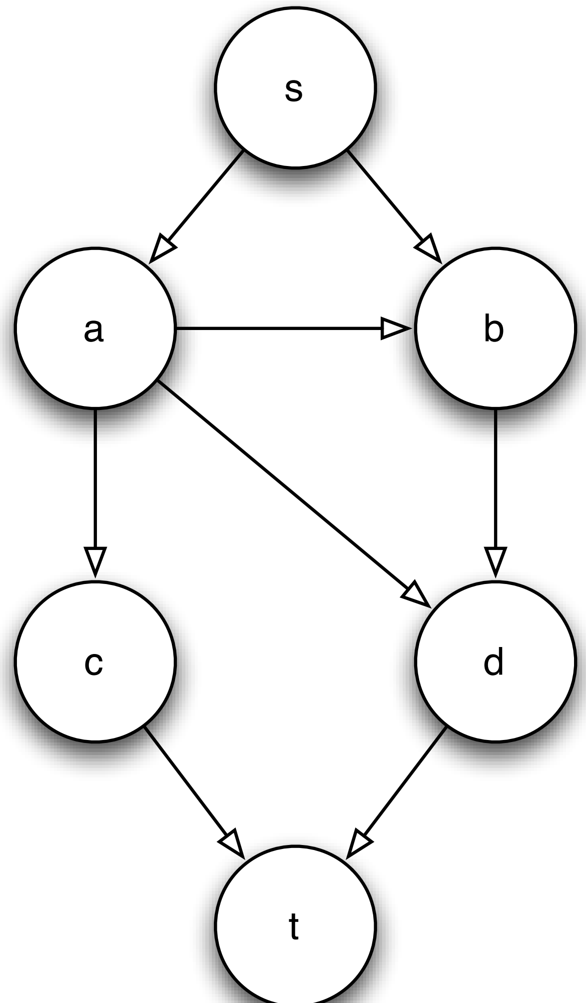
Augmenting Path

- Find any path in residual graph from source to sink
    - called an **augmenting path**.
- The minimum weight along path can be added as flow to the flow graph
- But we don't want to commit to this flow; add a reverse-direction undo edge to the residual graph

Example

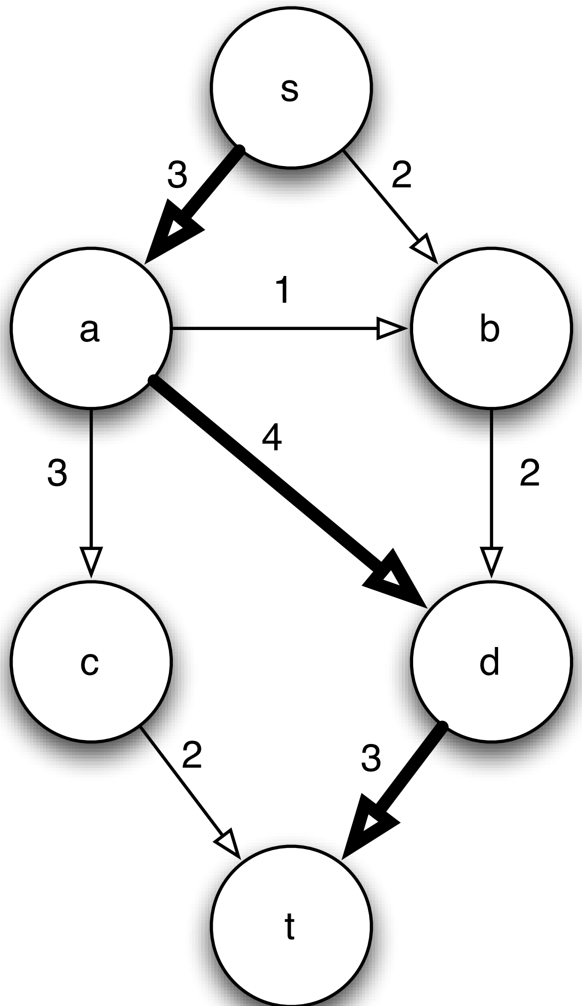


RESIDUAL

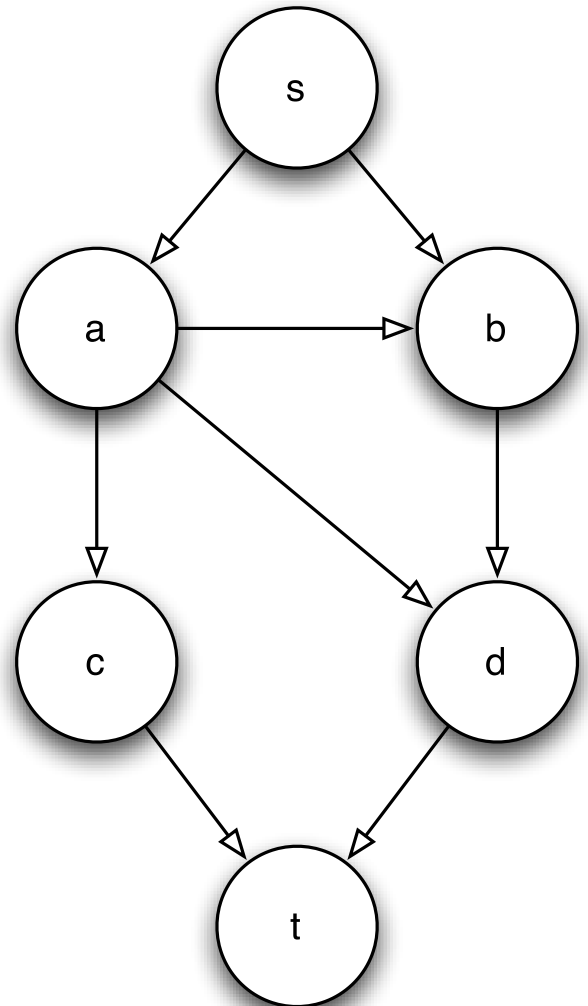


FLOW

Example

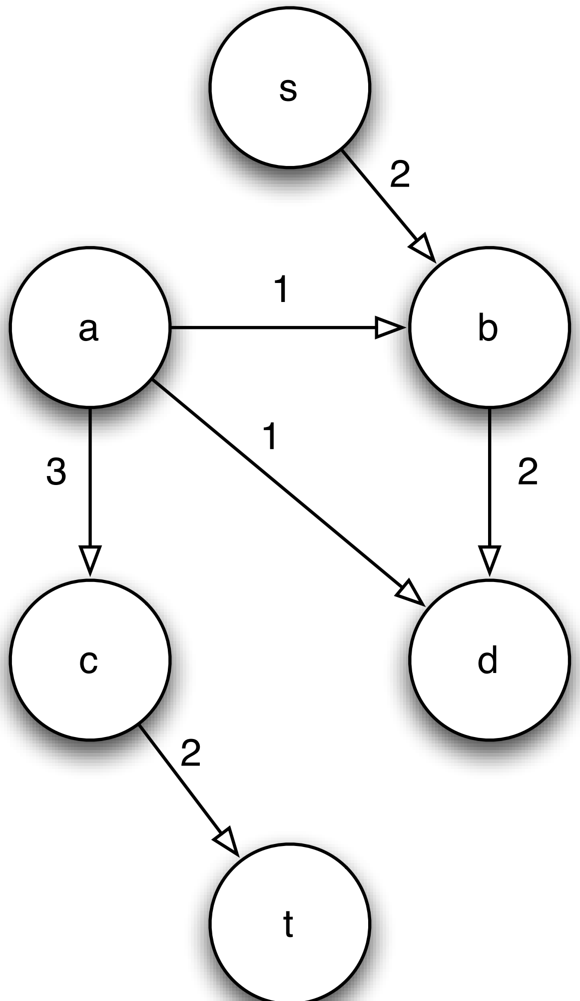


RESIDUAL

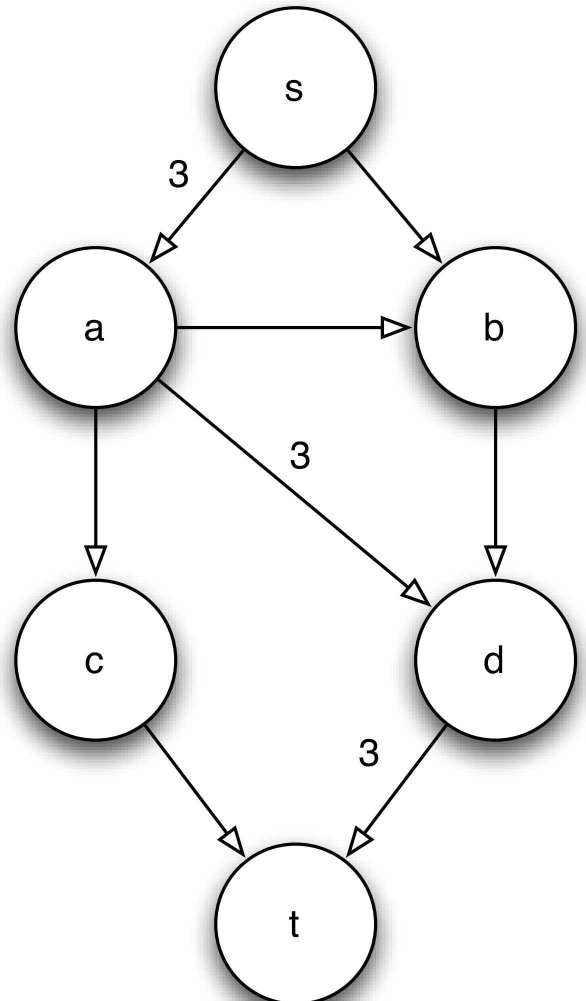


FLOW

Example

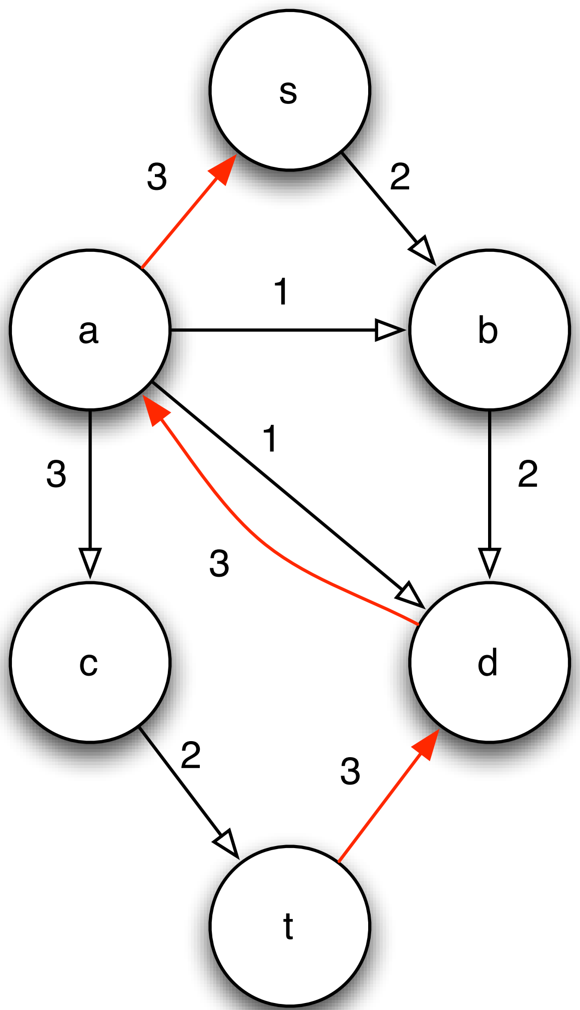


RESIDUAL

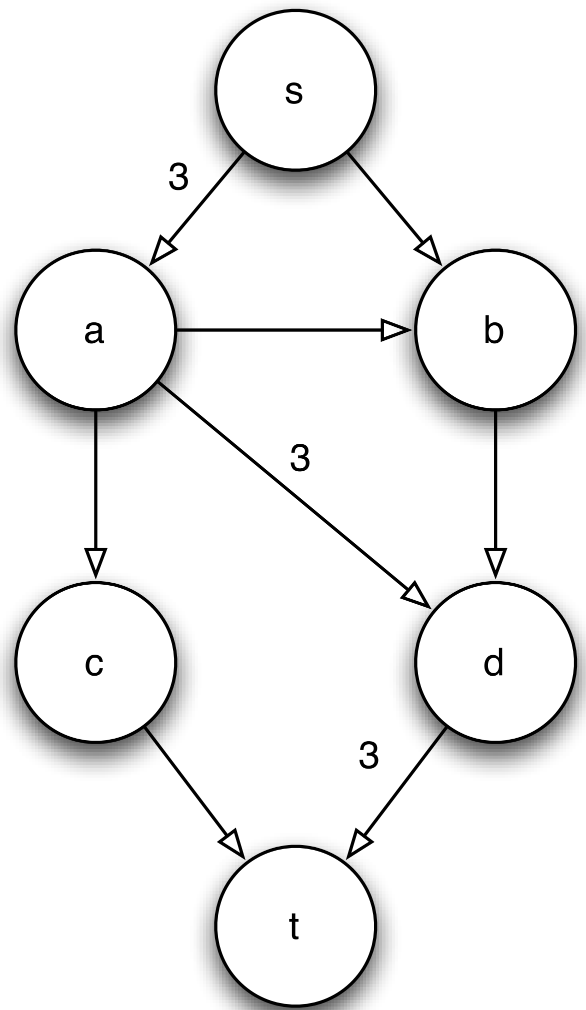


FLOW

Example

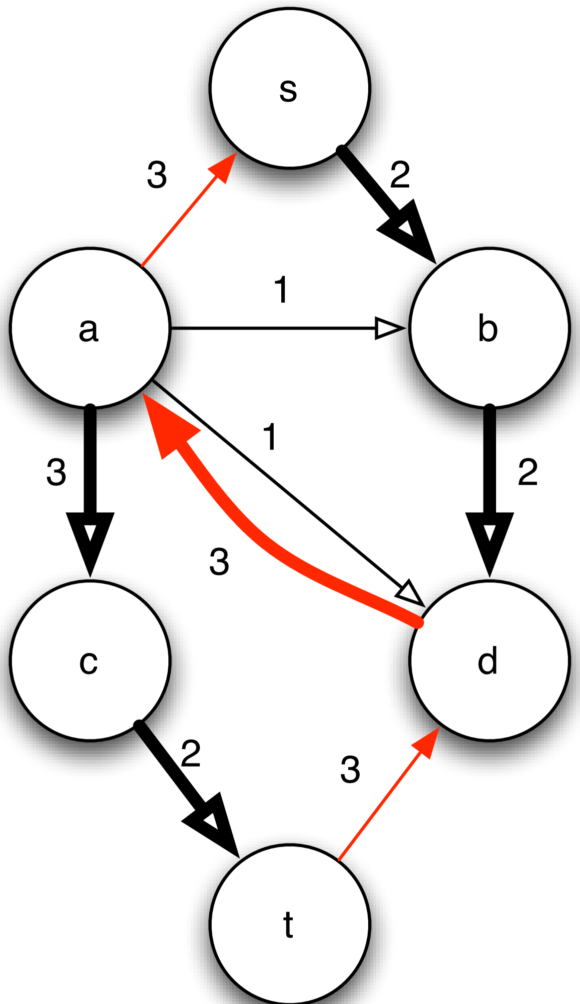


RESIDUAL

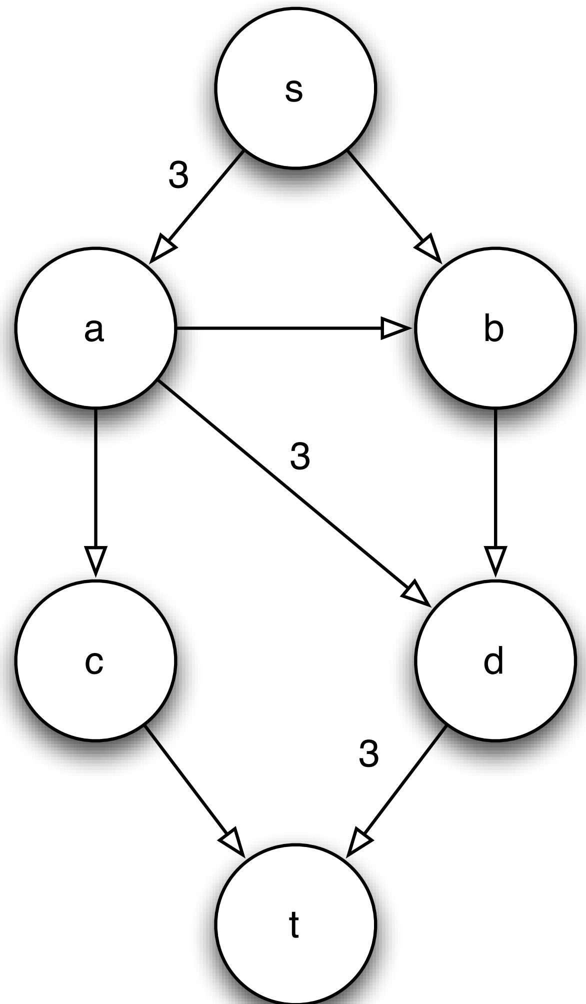


FLOW

Example

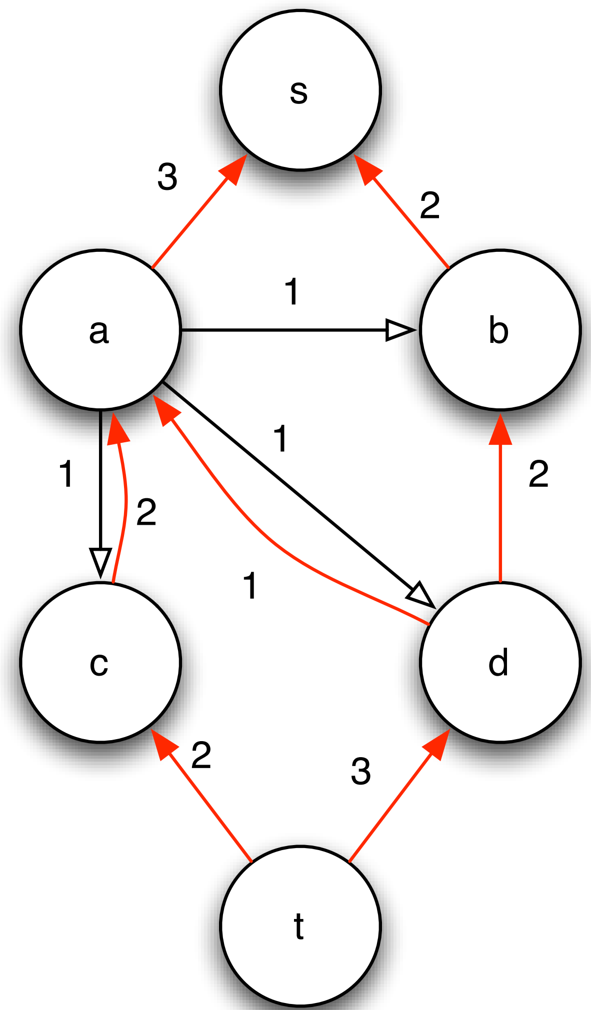


RESIDUAL

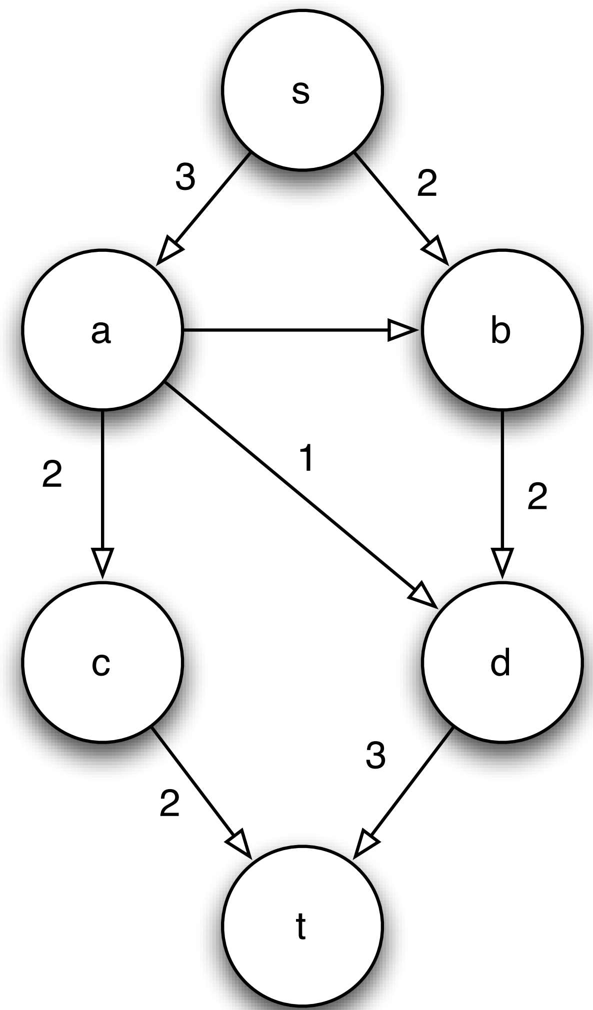


FLOW

Example



RESIDUAL



FLOW

Running Times

- If integer weights, each augmenting path increases flow by at least 1
- Costs O(|E|) to find an augmenting path
- For max flow f, finding max flow (Floyd-Fulkerson) costs $O(f|E|)$
- Choosing shortest unweighted path (Edmonds-Karp), $O(|V||E|^2)$

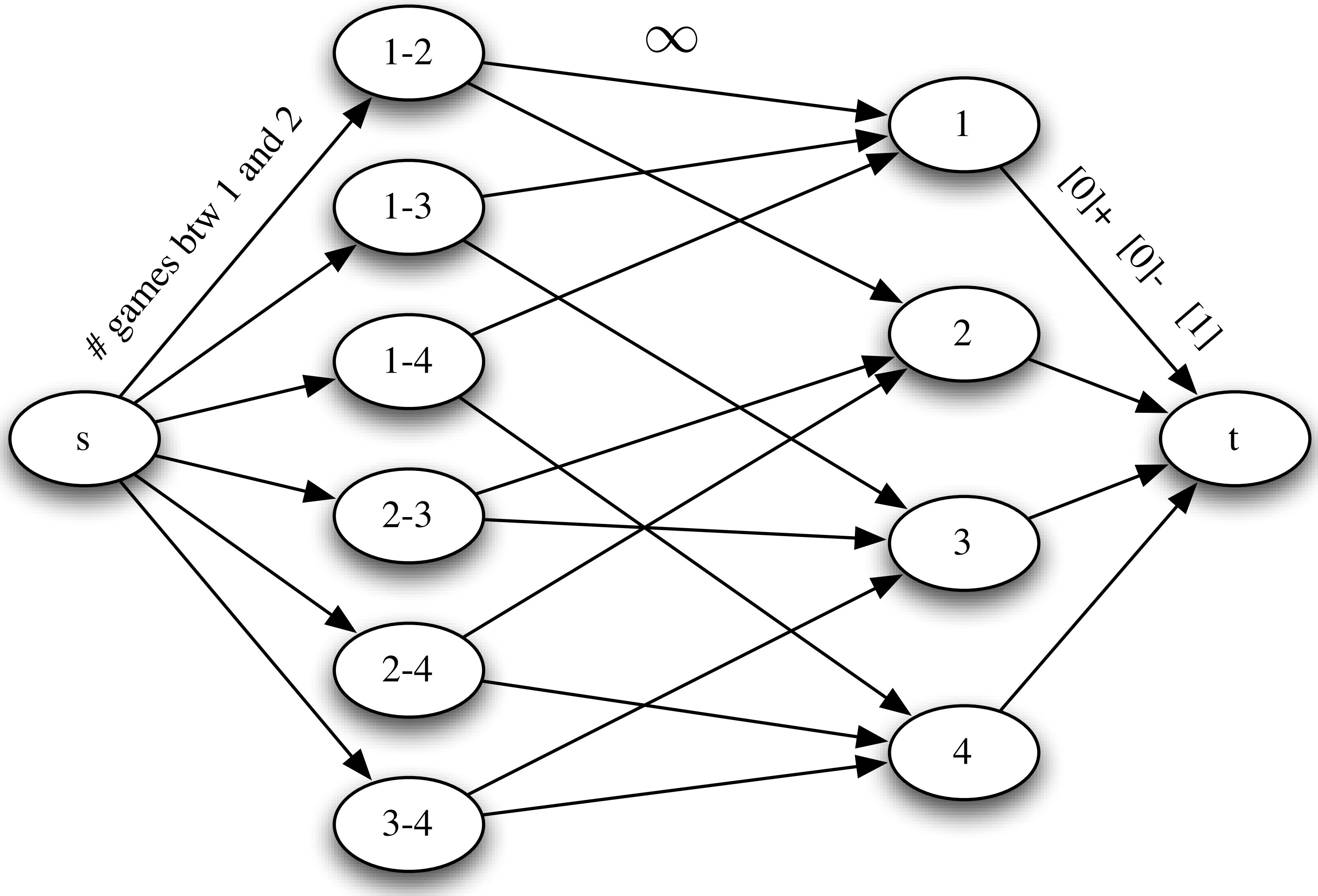
Sports Elimination

- In many organized sports, teams are split into divisions
  - the team in a division with the most wins at end of season earns a divisional title
- Fans and writers like to talk about whether a team is mathematically eliminated from the division race
- The standard formula is often wrong, instead, compute a max flow

Standard Formula

- If team **i** has **W[i]** wins, and **R[i]** remaining games, pretend **i** wins all of its **R[i]** games. **W[i]+R[i]**
- Pretend all other teams in division win no more games. If **W[i]+R[i]** > **W[j]**, for all **j**, **i** can still win
- The problem is the other teams may have games against each other; both teams can't lose

Max Flow Graph



Max Flow Solution: team i

- Connect source to all game nodes (team **j**, team **k**)
  - Capacity of edge to game node is # of games btw **j** and **k**
- Connect game nodes to participating team nodes with infinite capacity
- Connect team nodes to sink,
  capacity = # of games before team **j** overtakes team **i**
- Team **i** can win only if max flow saturates outgoing edges from source

Reading

- Weiss Section 9.5