This course material is now made available for public usage.
Special acknowledgement to School of Computing, National University of Singapore
for allowing Steven to prepare and distribute these teaching materials.

CS3233
Competitive Programming

Dr. Steven Halim

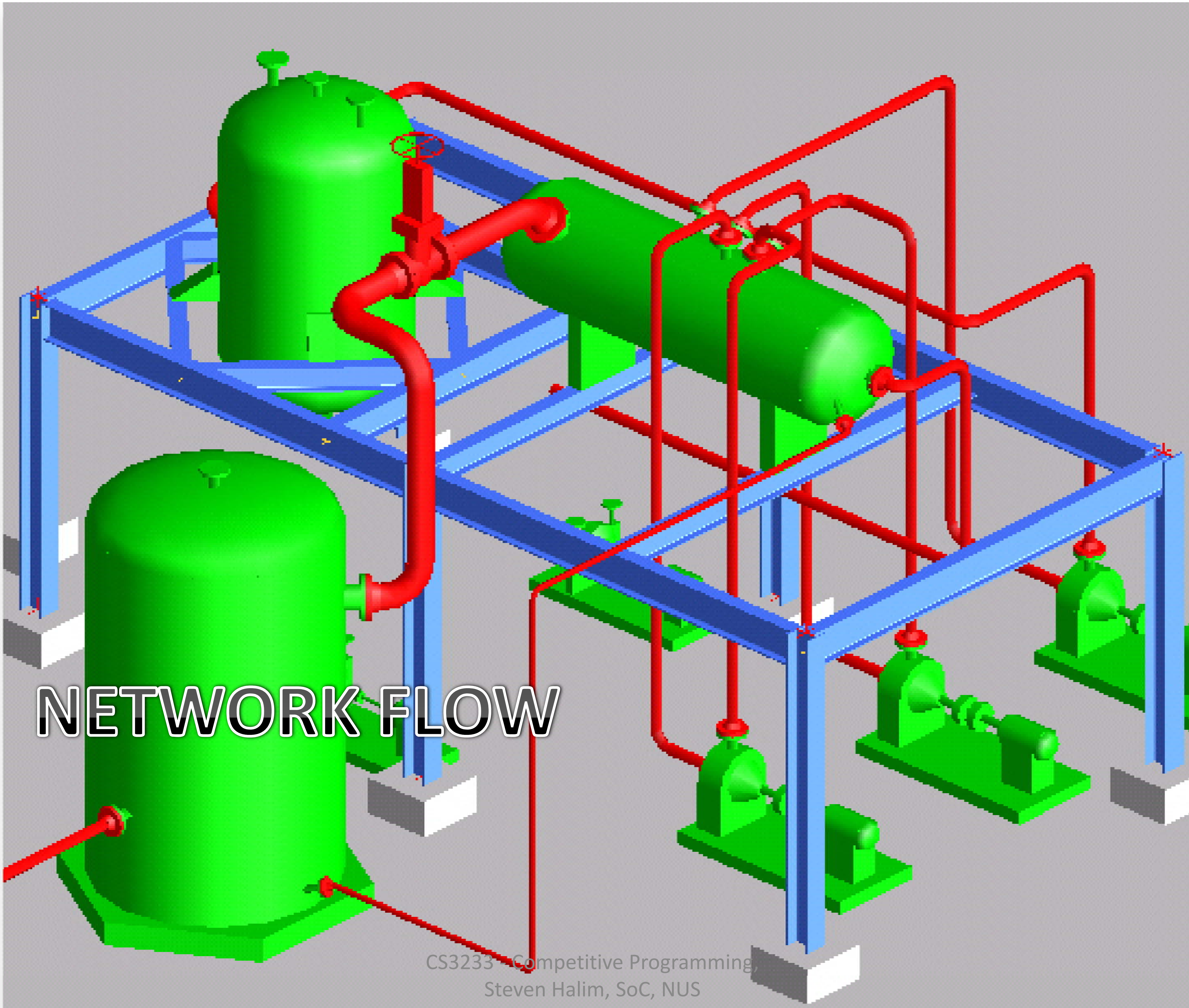
Week 07 – How to Prevent Floods?

Outline

- Mini Contest #5 + Break + Discussion + Admins
- Very Quick CS2010/2020 Review
- Network Flow (not in IOI 2009 syllabus)
  - Overview & Motivation (yes, to prevent floods :D)
  - Focus on Max Flow
  - Ford Fulkerson's Method
  - Edmonds Karp's Algorithm
- Flow Graph Modeling (*several* examples)
  - Bipartite Matching Variant, Min Cut, Multi Sources/Sink, Vertex Capacity, Independent Path, Edge-Disjoint Path, Min Cost (Max) Flow

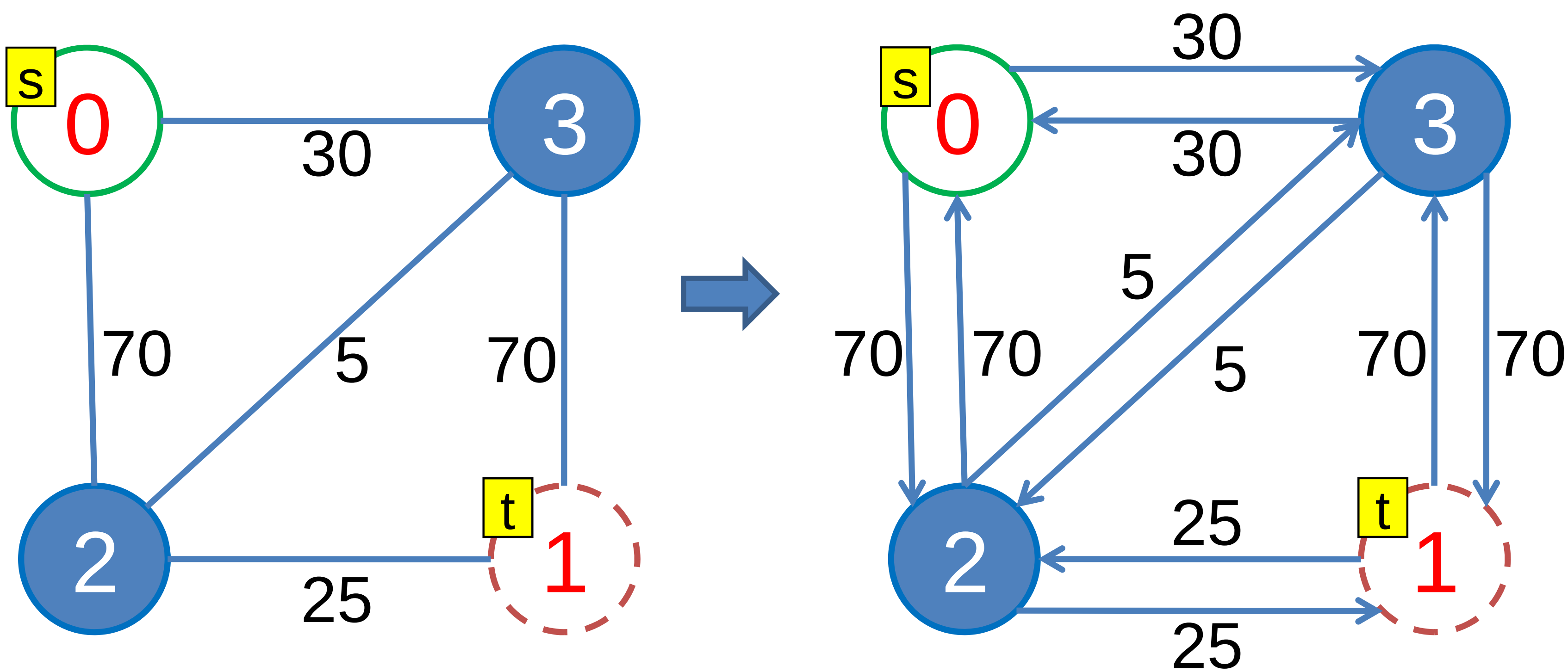
Graph in CS2010/2020…

- These have been revisited in Week2/5 additional classes
- Graph Data Structures
  - Adjacency Matrix, Adjacency List, Edge List, Implicit Graph…
- Graph Traversal: DFS/BFS
  - Various applications: Connected Component, Topological Sort
- Minimum Spanning Tree: Prim's/Kruskal's
  - Various applications
- Single-Source and All-Pairs Shortest Paths
  - Unweighted, weighted, negative cycle
  - Various applications



NETWORK FLOW

Max Flow in a Network (1)



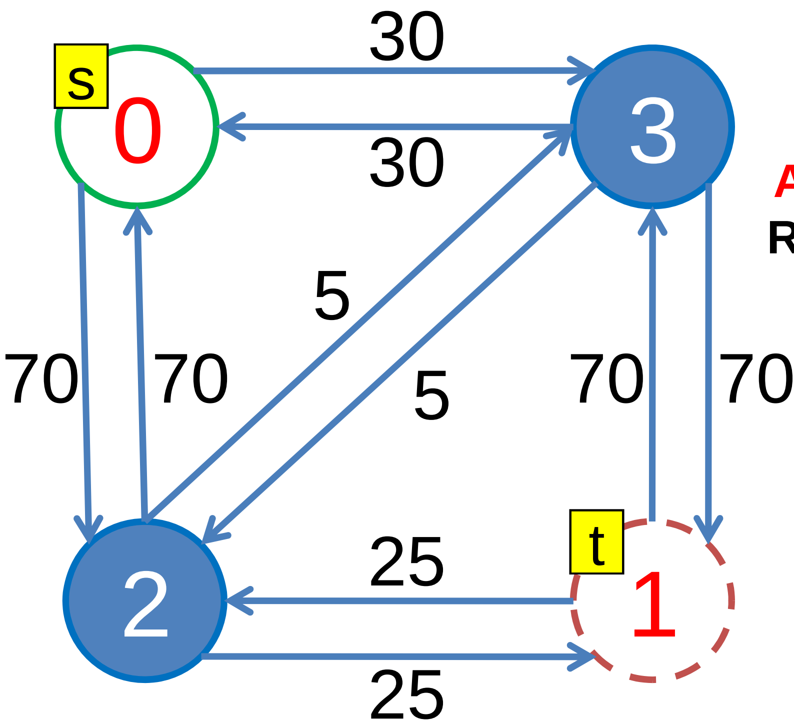
Max Flow in a Network (2)

- One Solution: [Ford](#) [Fulkerson](#)'s Method

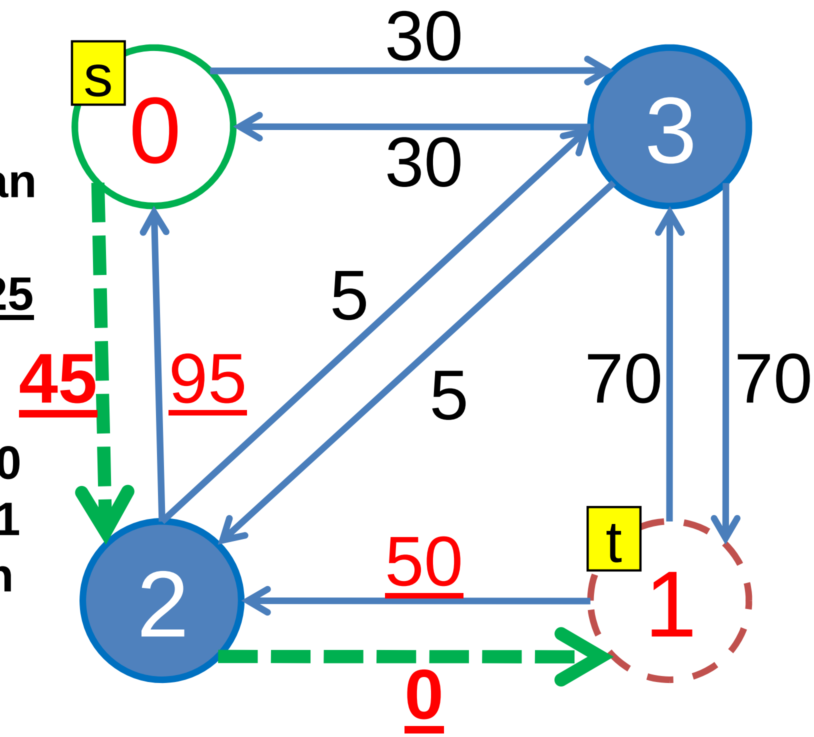
  - A surprisingly **simple** *iterative* algorithm

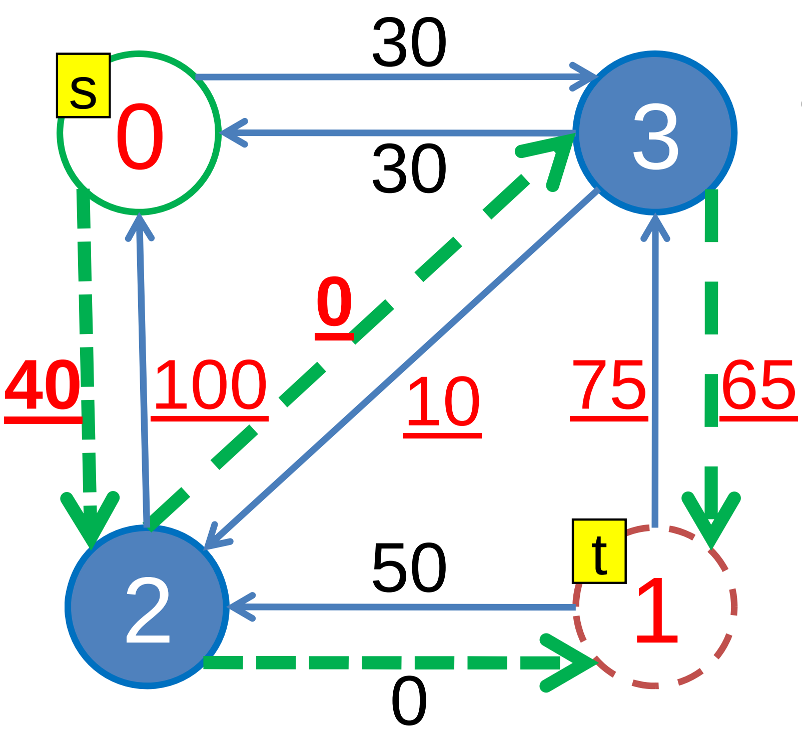
Send a flow *f* through path *p* whenever there exists
an **augmenting path** *p* from *s* to *t*



A. **Initial Residual Graph**

B. **We can pass a flow of <u>25 units</u> from source 0 to sink 1 via path 0-2-1**

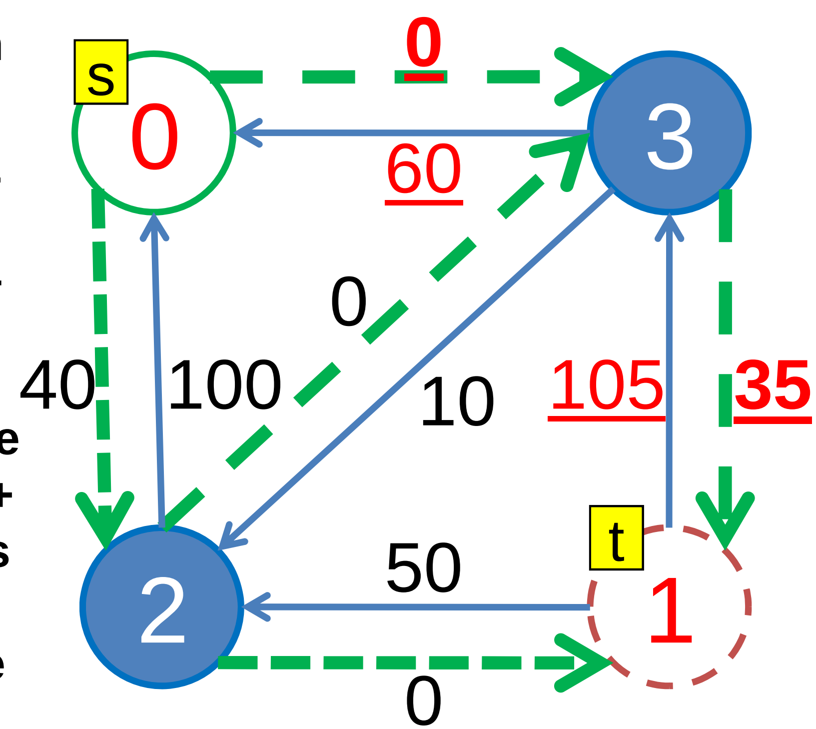




C. **We can pass another flow of <u>5 units</u> via path 0-2-3-1**

D. **We can pass last flow of <u>30 units</u> via path 0-3-1**

No more flow can be added. 25+5+30=60 is the <u>max flow</u> value



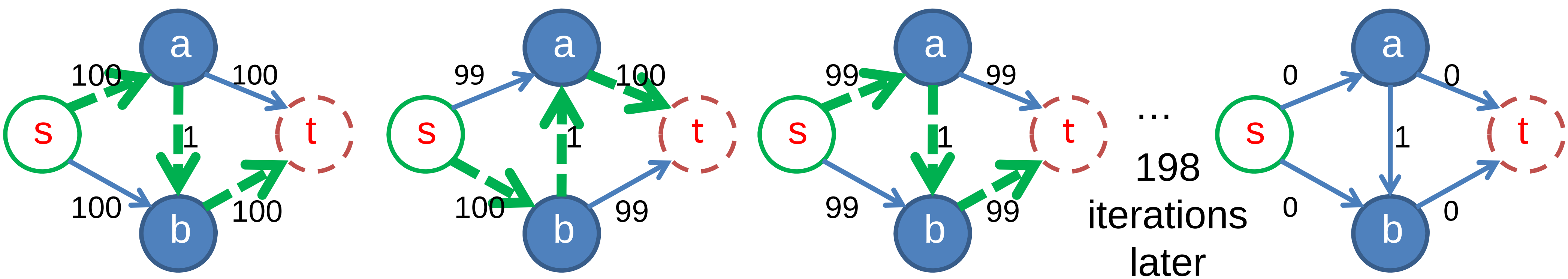
http://www.comp.nus.edu.sg/~stevenha/visualization/maxflow.html

Ford Fulkerson's Method

```
setup directed residual graph
each edge has the same weight with the original graph

mf = 0 // this is an iterative algorithm, mf stands for max_flow
while (there exists an augmenting path p from s to t) {
  // p is a path from s to t that pass through positive edges in residual graph
  augment/send flow f along the path p (s -> ... -> i -> j -> ... t)
   1. find f, the min edge weight along the path p
   2. decrease the weight of forward edges (e.g. i -> j) along path p by f
      reason: obvious, we use the capacities of those forward edges
   3. increase the weight of backward edges (e.g. j -> i) along path p by f
      reason: not so obvious, but this is important for the correctness of Ford
      Fulkerson's method;
  mf += f // we can send a flow of size f from s to t, increase mf
}
output mf
```

DFS Implementation

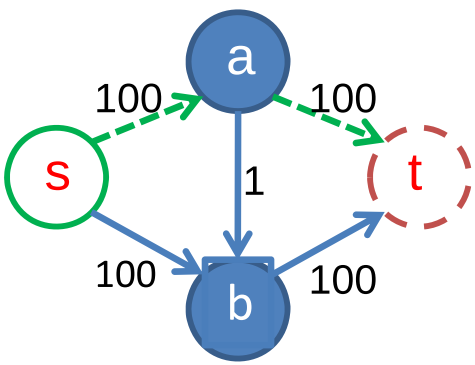
- DFS implementation of Ford Fulkerson's method runs in O(|f*|E) and can be very slow on graph like this:
  - Notice the presence of backward edges (only drawn for edge a→b or b→a this time)
    - Q: What if we do not use backward edges?

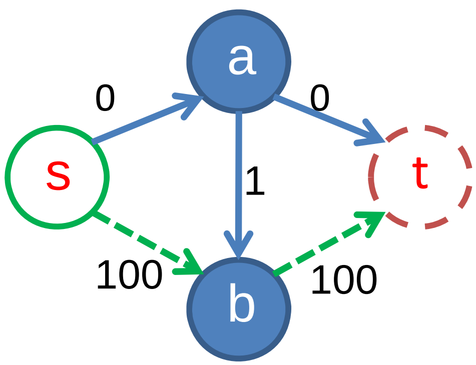


http://www.comp.nus.edu.sg/~stevenha/visualization/maxflow.html

BFS Implementation

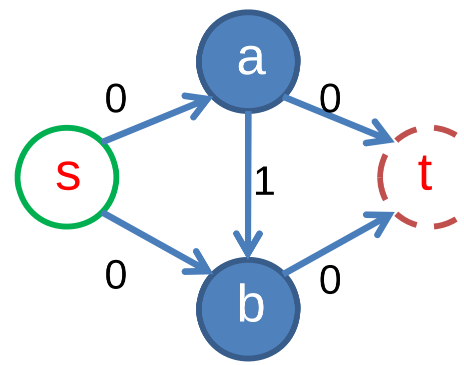
- BFS implementation of Ford Fulkerson's method (called Edmonds Karp's algorithm) runs in $O(VE^2)$





…

After just 2 iterations



Edmonds Karp's (using STL) (1)

```
int res[MAX_V][MAX_V], mf, f, s, t; // global variables
vi p; // note that vi is our shortcut for vector<int>

// traverse the BFS spanning tree as in print_path (section 4.3)
void augment(int v, int minEdge) {
  // reach the source, record minEdge in a global variable 'f'
  if (v == s) { f = minEdge; return; }
  // recursive call
  else if (p[v] != -1) { augment(p[v], min(minEdge, res[p[v]][v]));
  // alter residual capacities
                         res[p[v]][v] -= f; res[v][p[v]] += f; }
}

// in int main()
  // set up the 2d AdjMatrix 'res', 's', and 't' with appropriate values
```

Edmonds Karp's (using STL) (2)

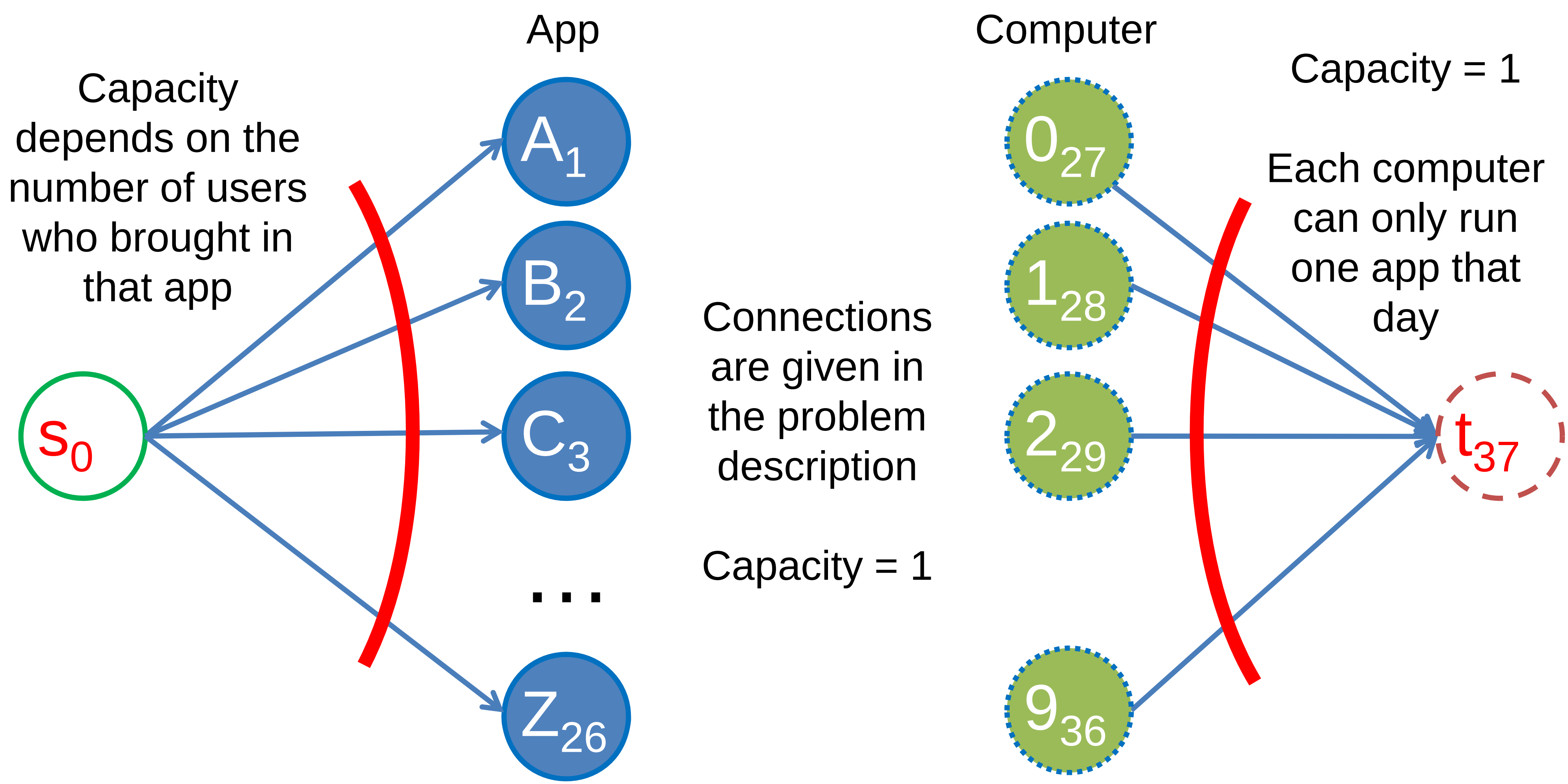
```
mf = 0;
while (1) { // run O(VE * V^2 = V^3*E) Edmonds Karp to solve the Max Flow problem
  f = 0;

  // run BFS, please examine parts of the BFS code that is different than in Section 4.2.23
  vi dist(MAX_V, INF); dist[s] = 0; // #define INF 2000000000
  queue<int> q; q.push(s);
  p.assign(MAX_V, -1); // (we have to record the BFS spanning tree)
  while (!q.empty()) { // (we need the shortest path from s to t!)
    int u = q.front(); q.pop();
    if (u == t) break; // immediately stop BFS if we already reach sink t
    for (int v = 0; v < MAX_V; v++) // note: enumerating neighbors with AdjMatrix is 'slow'
      if (res[u][v] > 0 && dist[v] == INF) dist[v] = dist[u] + 1, q.push(v), p[v] = u;
  }

  augment(t, INF); // find the min edge weight 'f' along this path, if any
  if (f == 0) break; // if we cannot send any more flow ('f' = 0), terminate the loop
  mf += f; // we can still send a flow, increase the max flow!
}

printf("%d\n", mf); // this is the max flow value of this flow graph
```

FLOW GRAPH MODELING

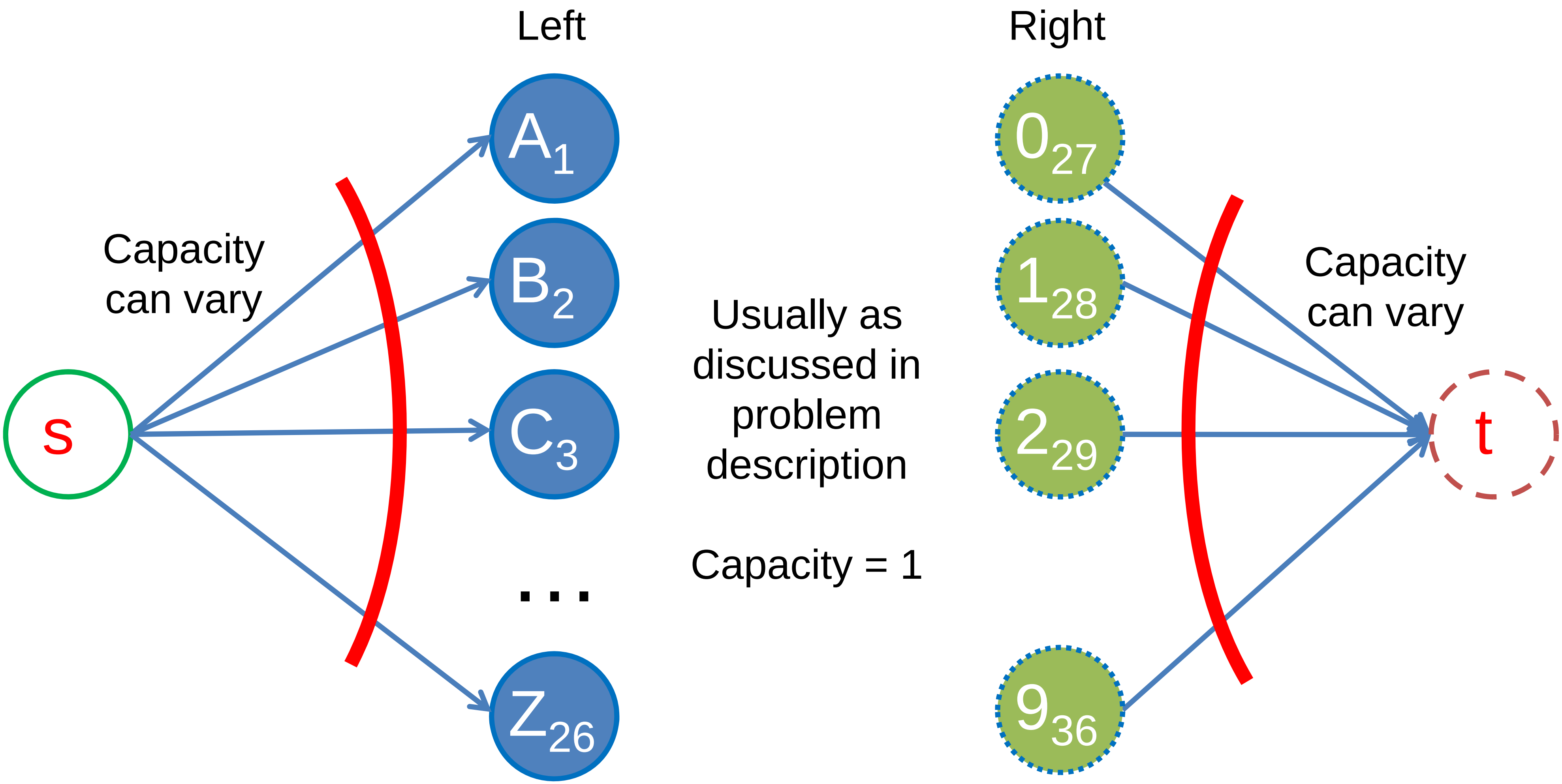
Network Flow Variants

- **Bipartite Matching Variant (more details next week)**
- **Min Cut**
- Multi-source Multi-sink Max Flow
  - The “super source and super sink” technique
- Max Flow with Vertex Capacities
  - The “vertex splitting” technique
- Max Independent Path
- Max Edge-Disjoint Path
- Min Cost Max Flow (MCMF)

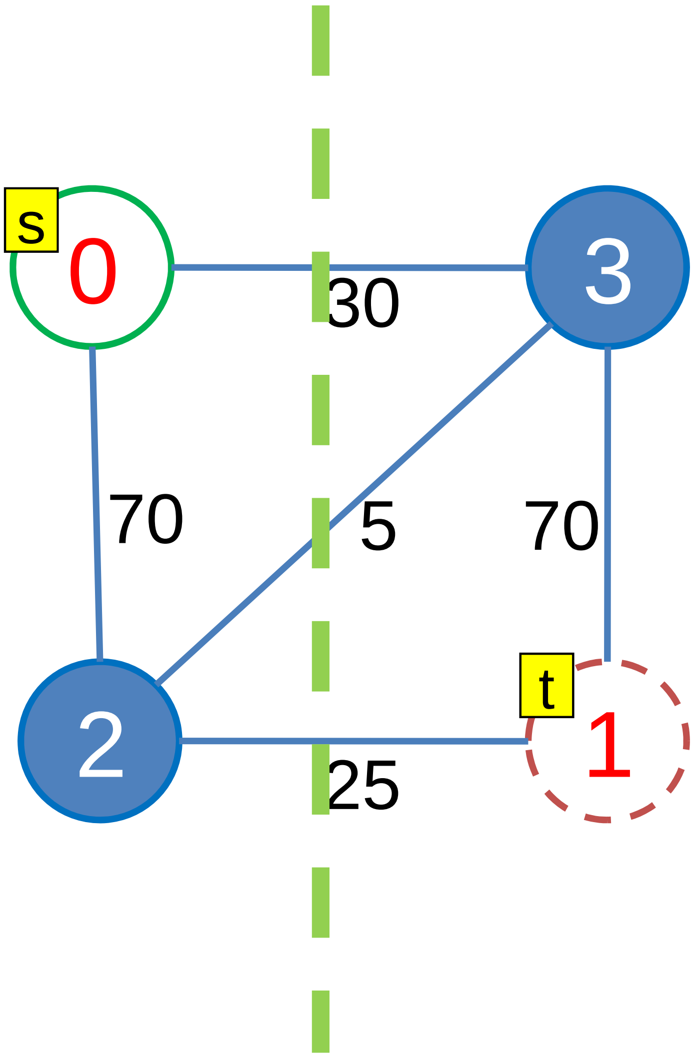
UVa 259 – Software Allocation

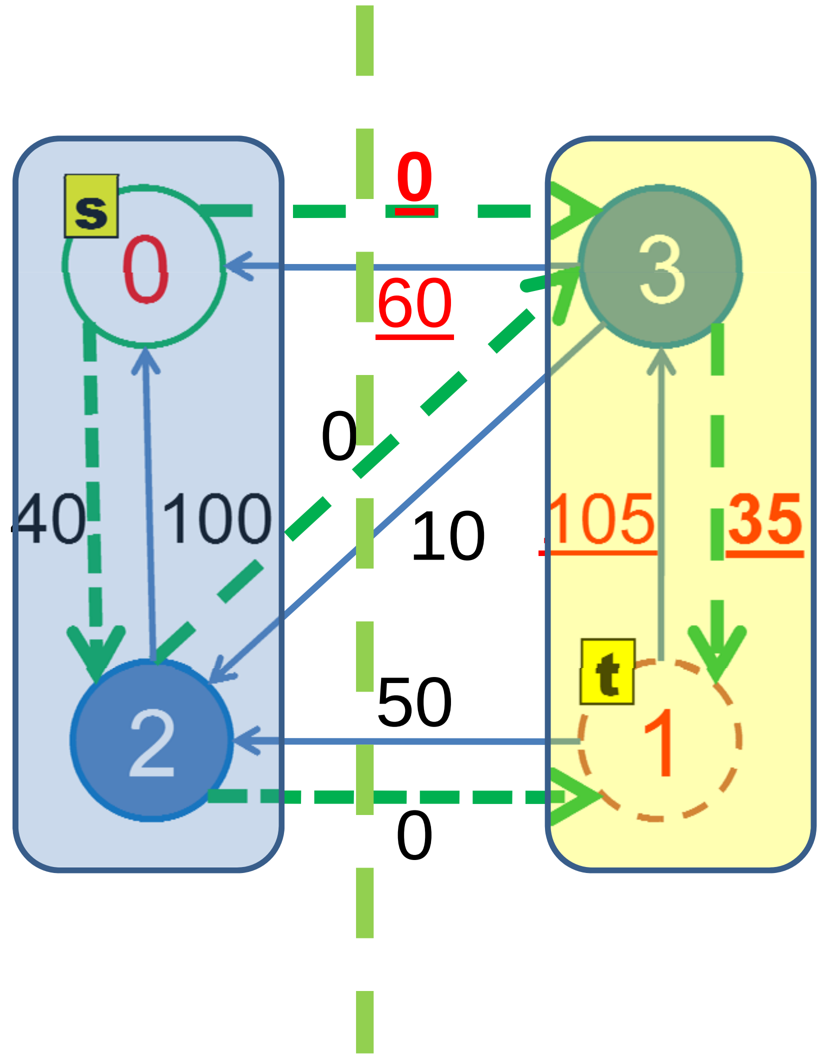


Bipartite Matching Variant



Min Cut





Graph Theory in ICPC

- Graph problems appear several times in ICPC!
  - Min 1, normally 2, can be 3 out of 10
  - Master all known solutions for classical graph problems
  - Or perhaps combined with DP/Greedy style
- This can move your team nearer to top 10
  - Perhaps rank [11-20] out of 60 now ☺
  - Solving 3-5 problems out of 10
- For IOI trainees… all these Network Flow stuffs…
  - **ARE NOT IN THE SYLLABUS…**

References

- **CP2.9, Section 4.6, 9.6, 9.13, 9.14 ☺**
- Introduction to Algorithms, Ch 22,23,24,25,26 (p643-698)
- Algorithm Design, Ch 3,4,6,7 (p337-450)
- Algorithms (Dasgupta et al), Ch 6 & Ch 7
- Algorithms (Sedgewick), Ch 33 & Ch 34
- Algorithms (Alsuwaiyel), Ch 16 & Ch 17
- Programming Challenges, p227-230, Ch 10
- http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=standardTemplateLibrary2
- Internet: TopCoder Max-Flow tutorial, UVa Live Archive, UVa main judge, Felix's blog, Suhendry's blog, Dhaka 2005 solutions, other Max Flow lecture notes, etc…