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.

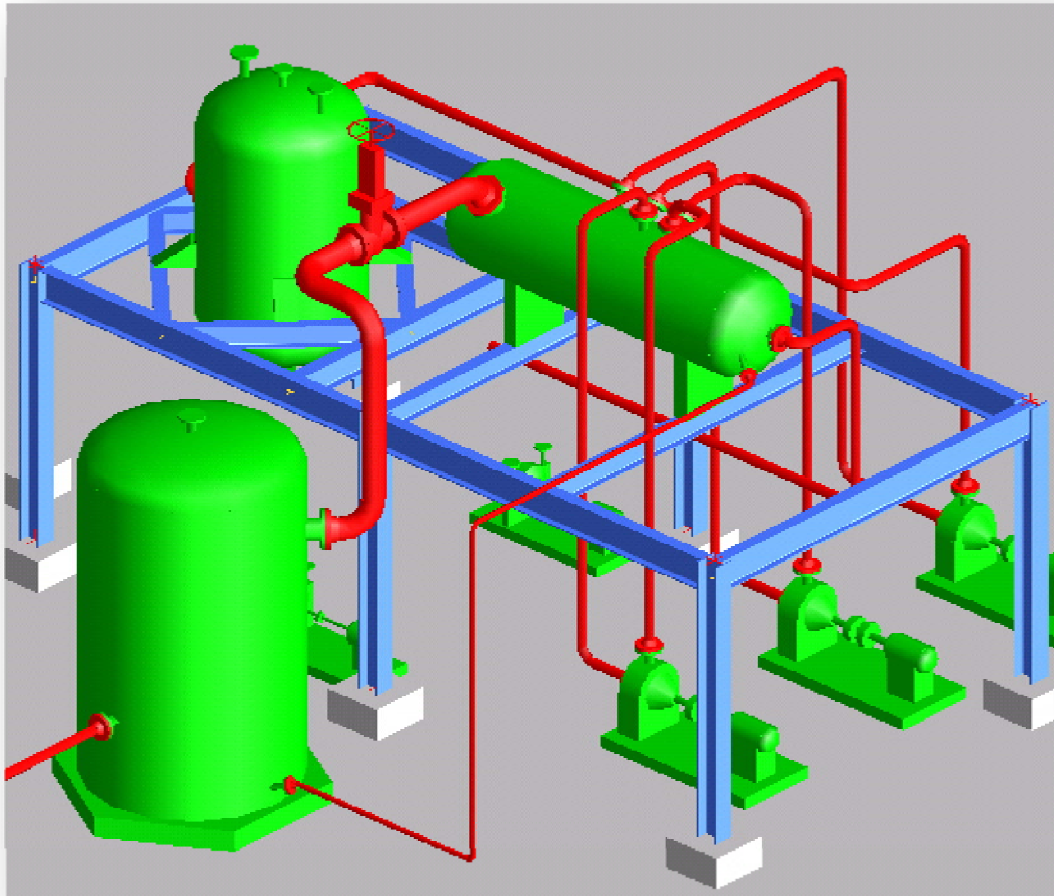
CS3233
Competitive Programming

Dr. Steven Halim

Week 08 – Max Flow and Bipartite Graph
(both are *not* in IOI syllabus 2009)

Outline

- Mini Contest #6 + Break + Discussion + Admins
- Max Flow
  - (We will skip the many variants → read the book!)
- Special Graphs 2
  - Bipartite Graph
  - Finding the bipartite graph in a problem
  - The important Alternating Path algorithm



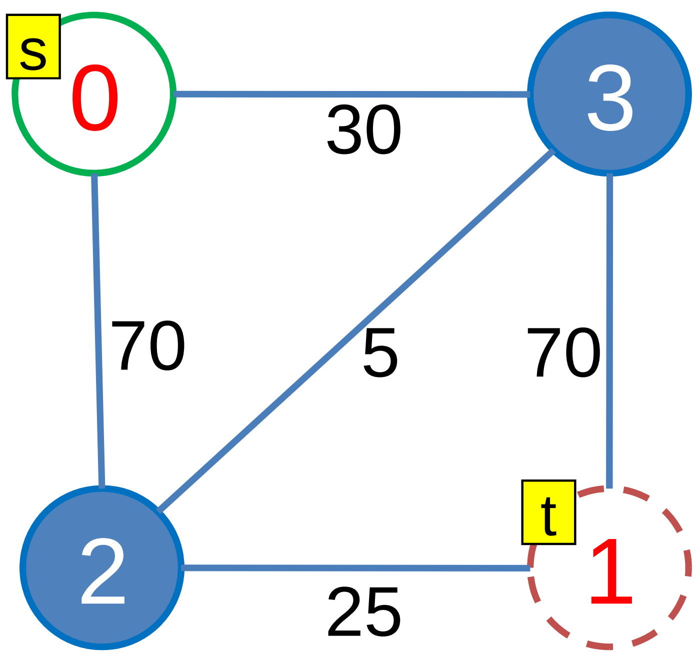
MAX FLOW

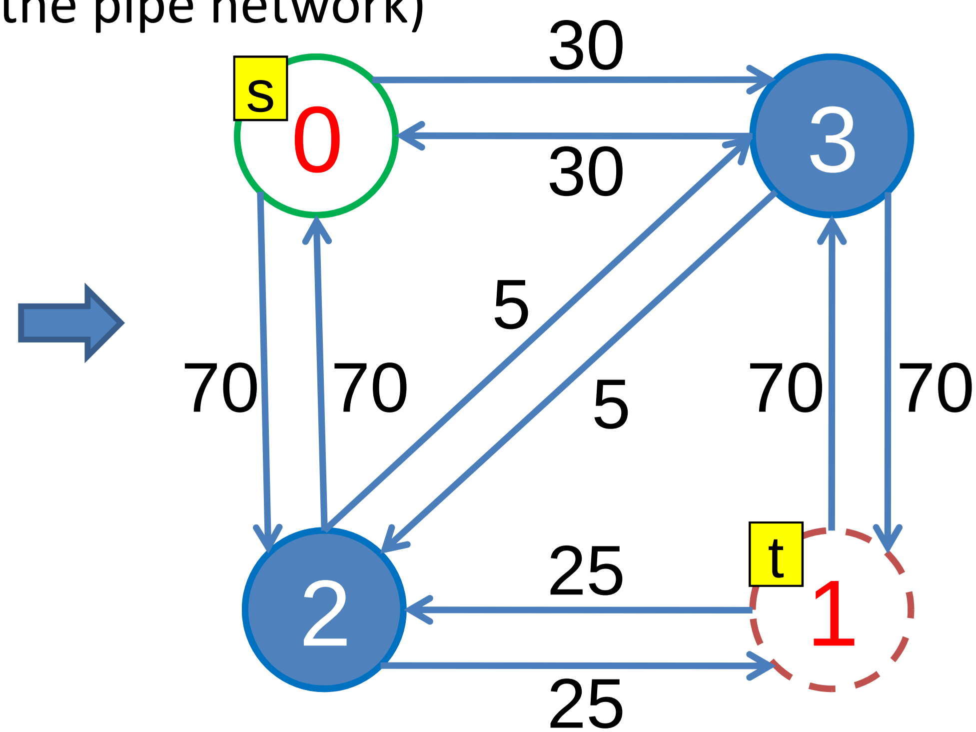
Motivation

- How to solve this UVa problem:
  - 820 (Internet Bandwidth)
    - Similar problems: 259, 753, 10092, 10480, 10511, etc
- Without **Max Flow** algorithms, they look “hard”

Max Flow in a Network

- Imagine **connected**, **weighted**, **directed** graph as *pipe network*
  - The edges are the pipes
  - The vertices are the splitting points
  - There are also two special vertices: source ***s*** & sink ***t***
  - What is the **max flow** (rate) from source ***s*** to sink ***t*** in this graph? (imagine water flowing in the pipe network)



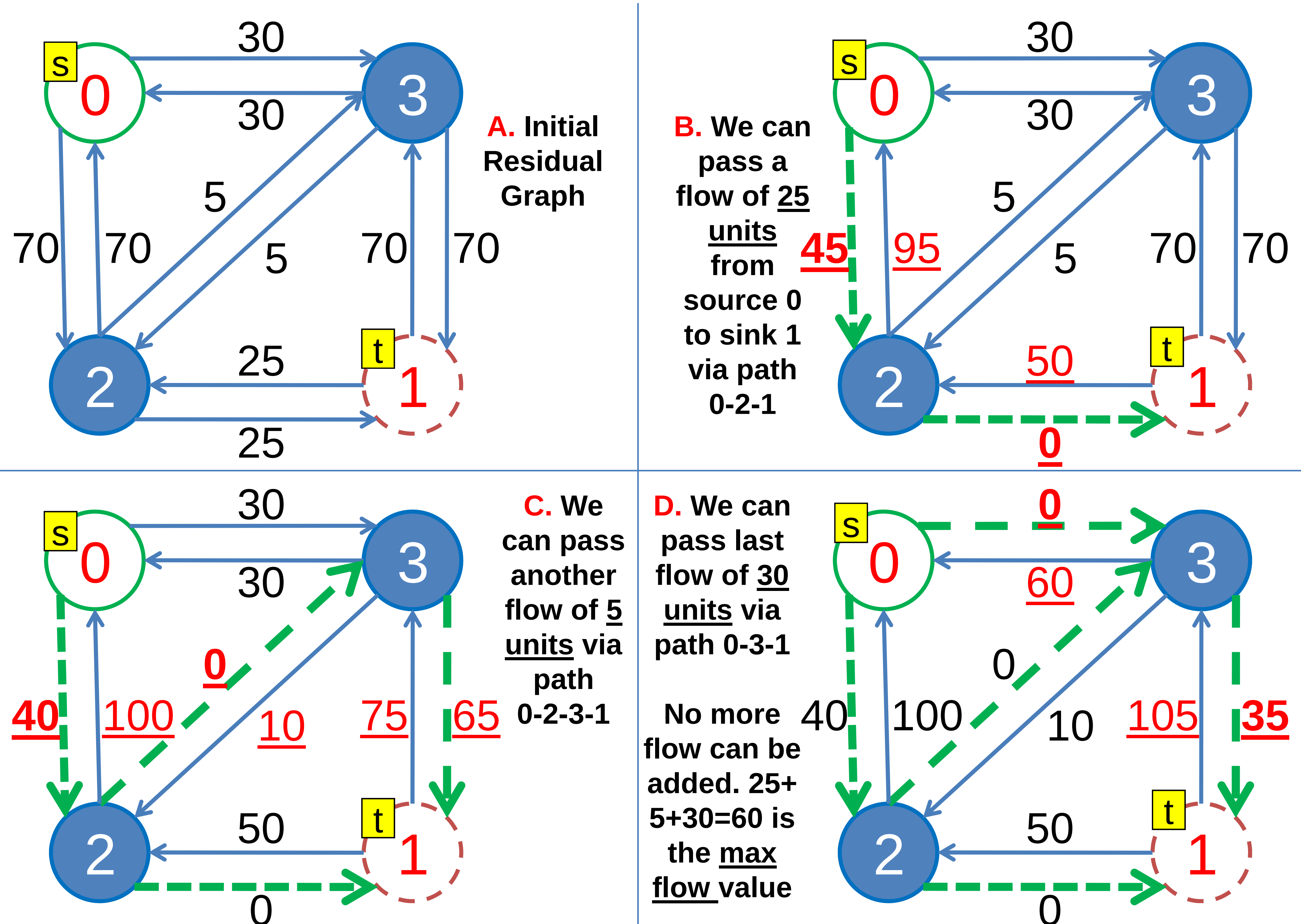


Maximum Flow in Network

- One Solution: Ford Fulkerson's Method

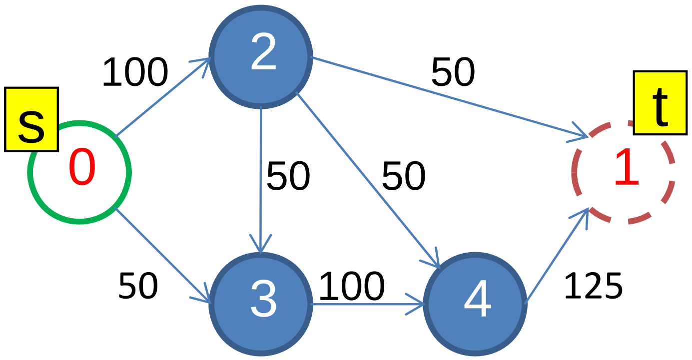
 

  - A surprisingly **simple** *iterative* algorithm
    - Send a flow through path *p* whenever there exists an augmenting path *p* from *s* to *t*
      - *Augmenting path is a path from source s to sink t that pass through positive edges in residual graph*


A. Initial Residual Graph
B. We can pass a flow of 25 units from source 0 to sink 1 via path 0-2-1
C. We can pass another flow of 5 units via path 0-2-3-1
D. We can pass last flow of 30 units via path 0-3-1
No more flow can be added. 25+5+30=60 is the max flow value

UVa 820

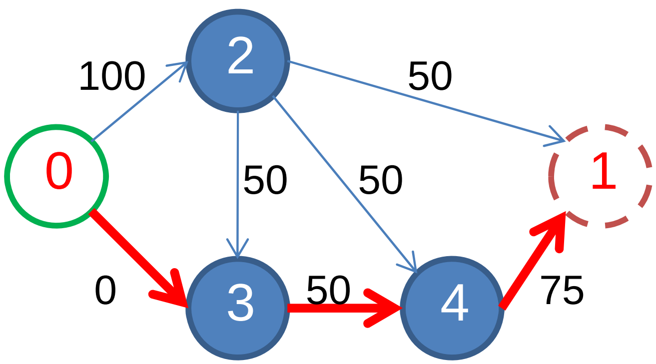
What is the Max Flow value? (1)



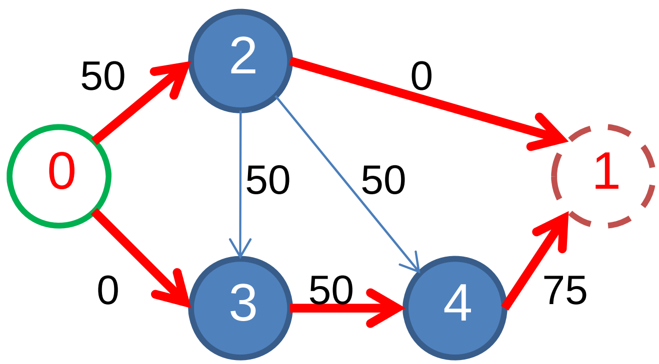
Answer: Maximum flow = 150

Note: **backward edges are not shown…**

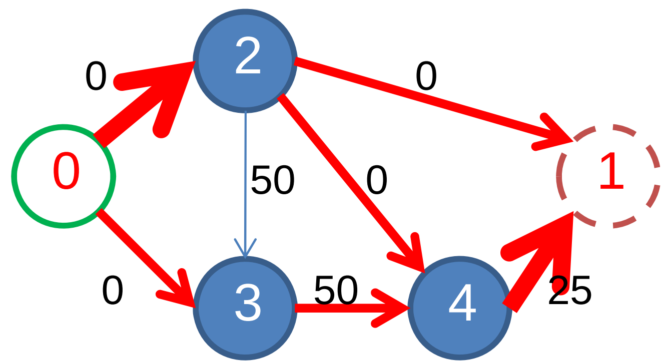
Flow so far = 50



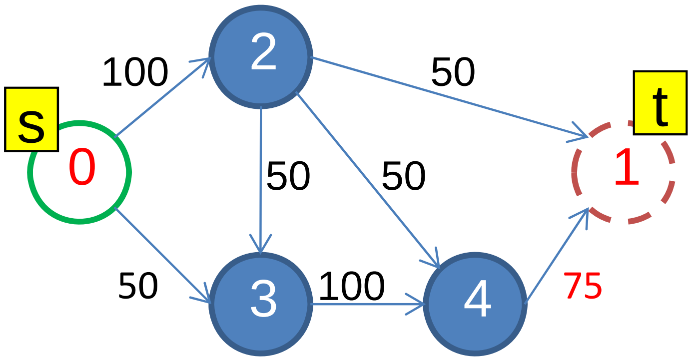
Flow so far = 100



Flow so far = 150



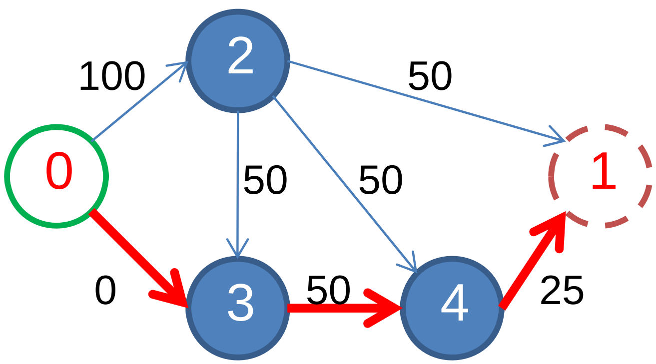
What is the Max Flow value? (2)



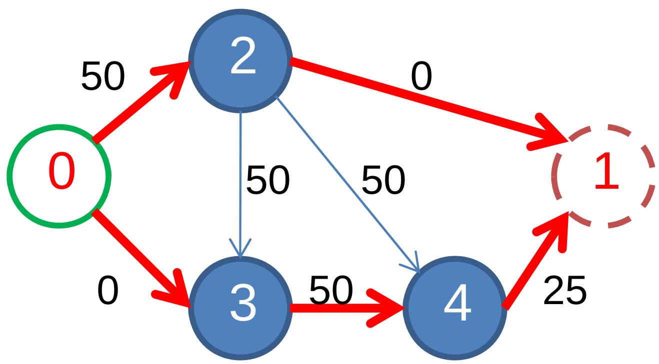
Answer: Maximum flow = 125

Note: **backward edges are not shown…**

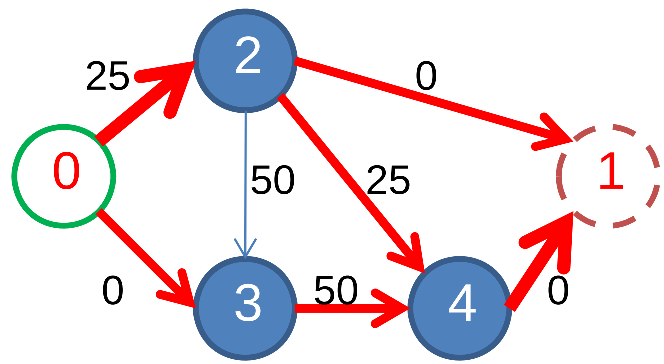
Flow so far = 50



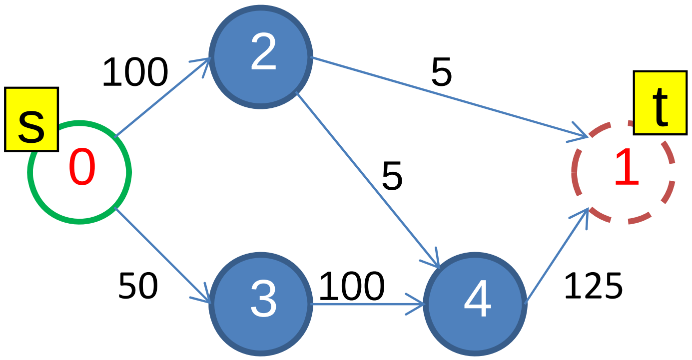
Flow so far = 100



Flow so far = 125



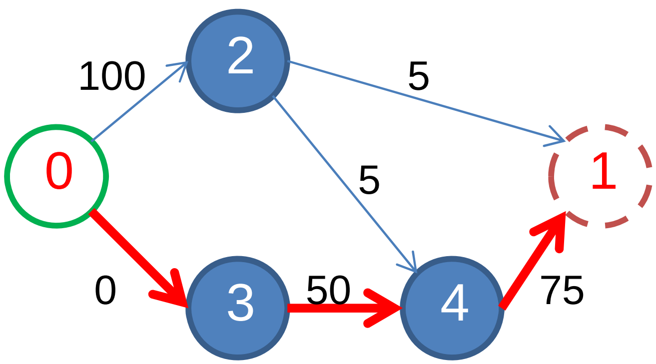
What is the Max Flow value? (3)



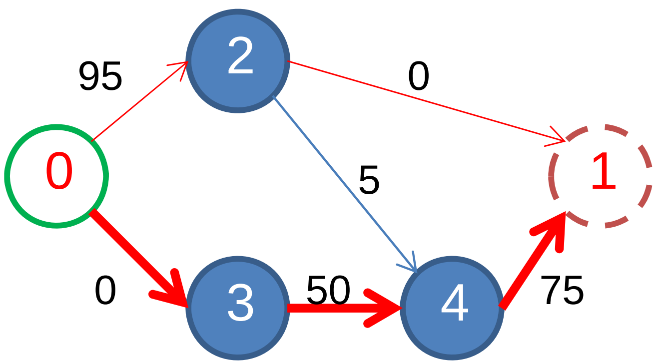
Answer: Maximum flow = 60

Note: **backward edges are not shown…**

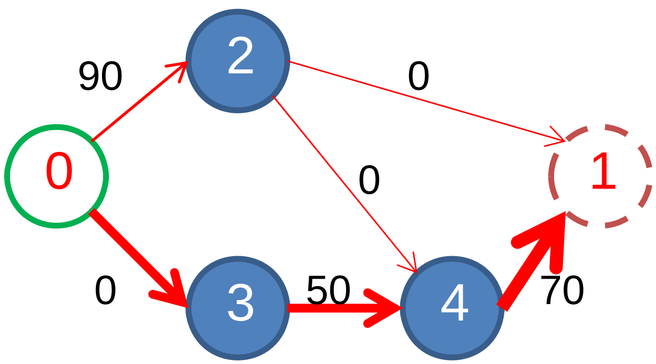
Flow so far = 50



Flow so far = 55



Flow so far = 60



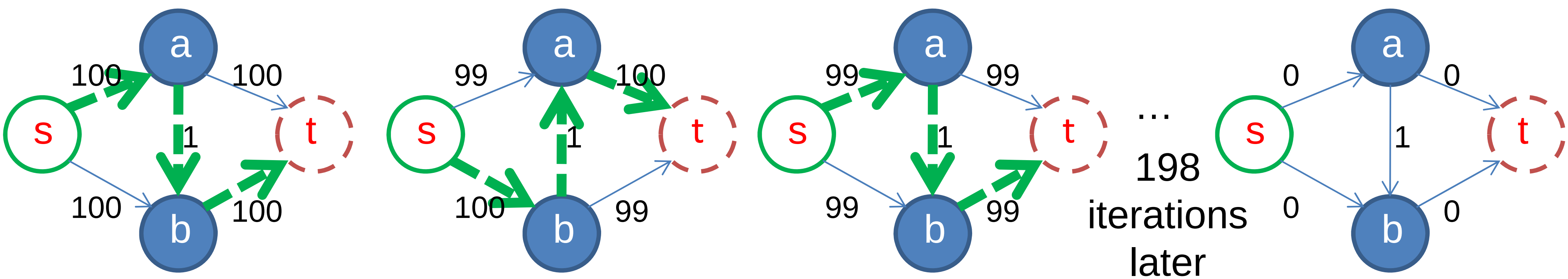
Ford Fulkerson's Method

```
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; by increasing the weight/capacity of a backward edge
       (j -> i), we allow later iteration/flow to cancel part of weight/capacity
       used by a forward edge (i -> j) if not all capacity of edge (i -> j) is
       used in the final set of paths that produce the max flow
  mf += f // we can send a flow of size f from s to t, increase mf
}
output mf
```

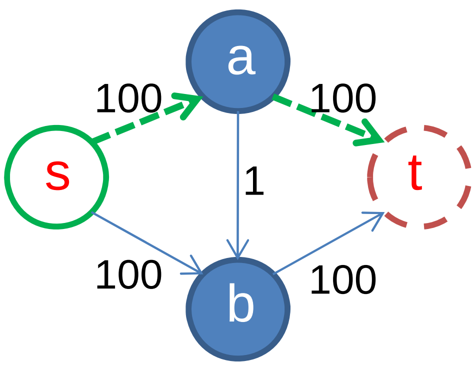
DFS Implementation

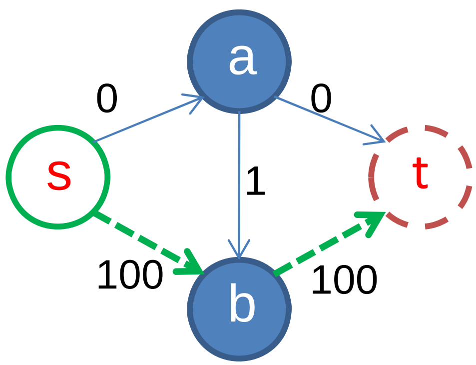
- DFS implementation of Ford Fulkerson's method runs in O(|f*|E) and can be very slow on graph like this:
  - Notice the presence of backward edges (drawn this time)
    - Q: What if we do not use backward edges?



BFS Implementation

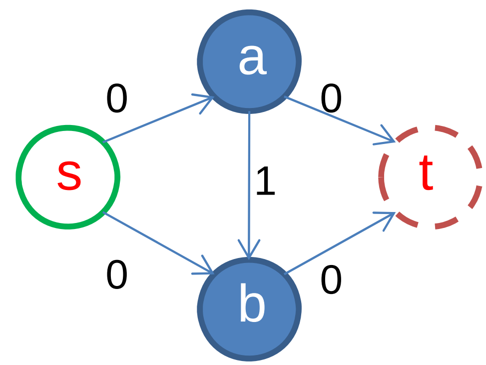
- BFS implementation of Ford Fulkerson's method (called [Edmonds](#) [Karp](#)'s algorithm) runs in $O(VE^2)$





…

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)

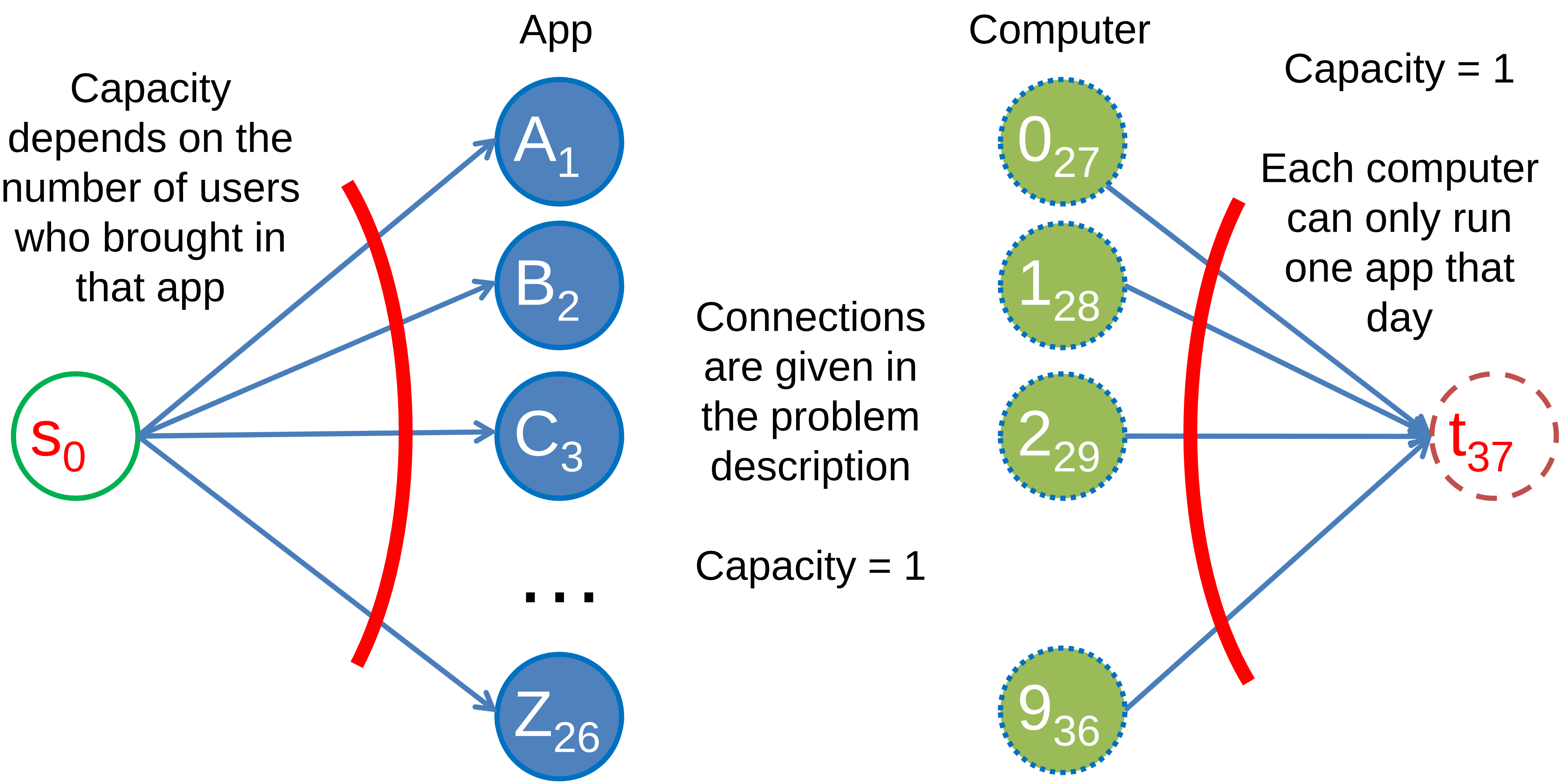
```
mf = 0;
while (1) { // run O(VE * V^2 = V^3*E) Edmonds Karp to solve the Max Flow problem
  f = 0;

  // run BFS, please examine parts of the BFS code that is different than in Section 4.3
  queue<int> q; vi dist(MAX_V, INF); // #define INF 2000000000
  q.push(s); dist[s] = 0;
  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
```

UVa 259 – Software Allocation



Other Applications of Max Flow

- Variants:
  - Min Cut
  - Multi-source Multi-sink Max Flow
  - Max Flow with Vertex Capacities
  - Max Independent Path
  - Max Edge-Disjoint Path
  - Min Cost Max Flow (MCMF)
- On "Special Graphs":
  - Max Cardinality Bipartite Matching (MCBM)
  - Max Independent Set on Bipartite Graph (brief)
  - Max Vertex Cover on Bipartite Graph (brief)
  - Min Path Cover on DAG → MCBM on Bipartite Graph

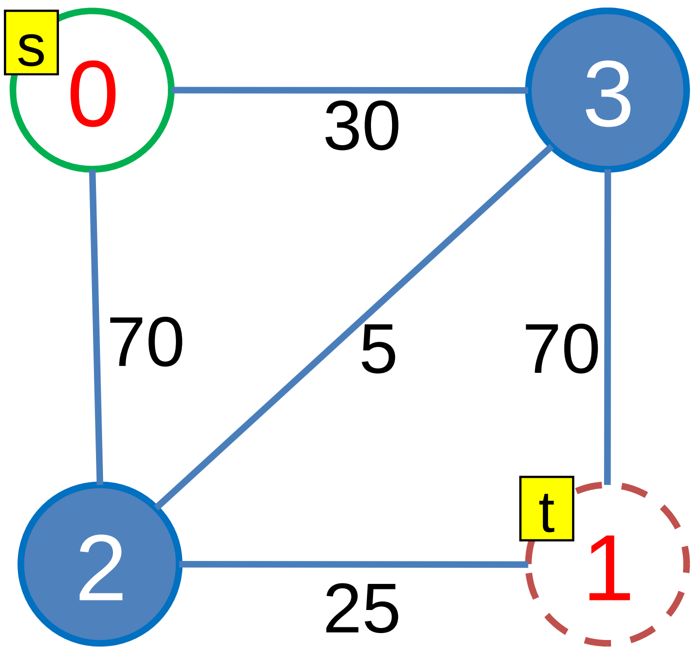
We will focus only on the red ones for this semester's CS3233 ☹

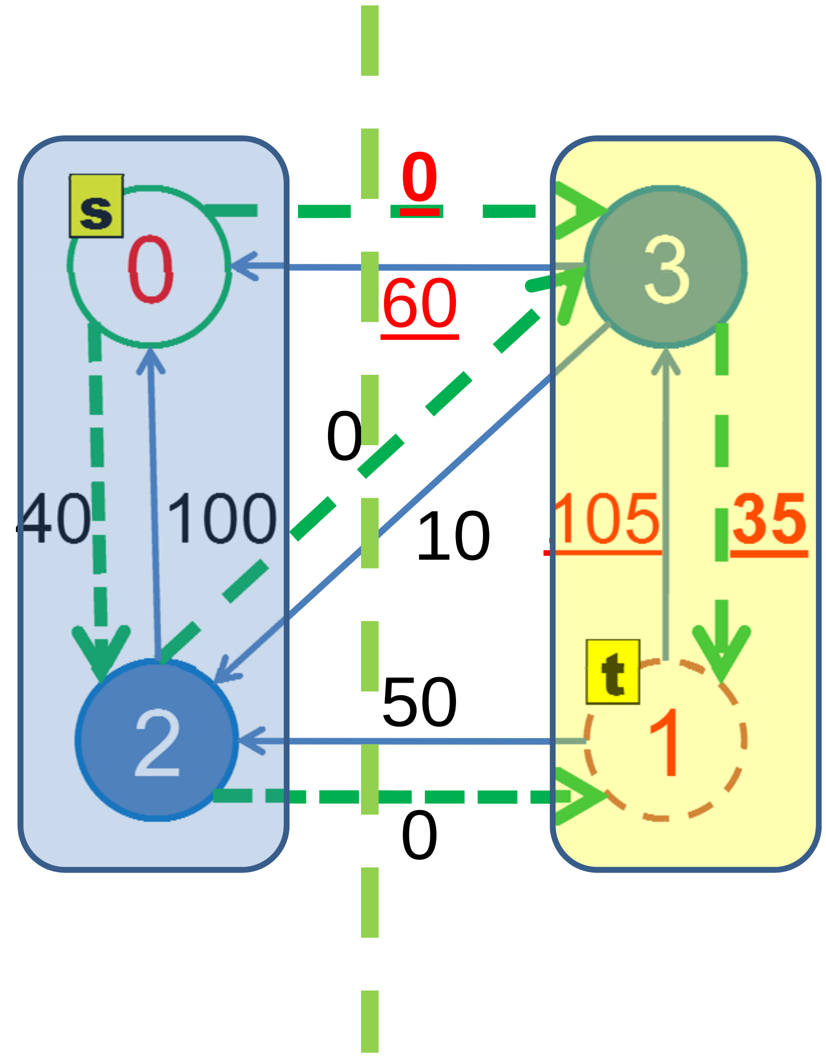
For the rest, read the book!

Max Flow = Min Cut (Dual)

- Min Cut:
  - How to cut the edge(s) of the graph such that we minimizes sum of the edge weights being cut?
    - Note: This is ‘similar’ to finding ‘bridges’, but they are **different**!
- It happens that running max flow algorithm can give us the answer for min cut problem :O
  - Idea: Once we cannot add any more flow (max flow reached), do graph traversal from source s *one more time*
  - All vertices still reachable from s are $\in$ first component, the rest are $\in$ second component, and the cut is obvious
  - Detailed proof is not shown…

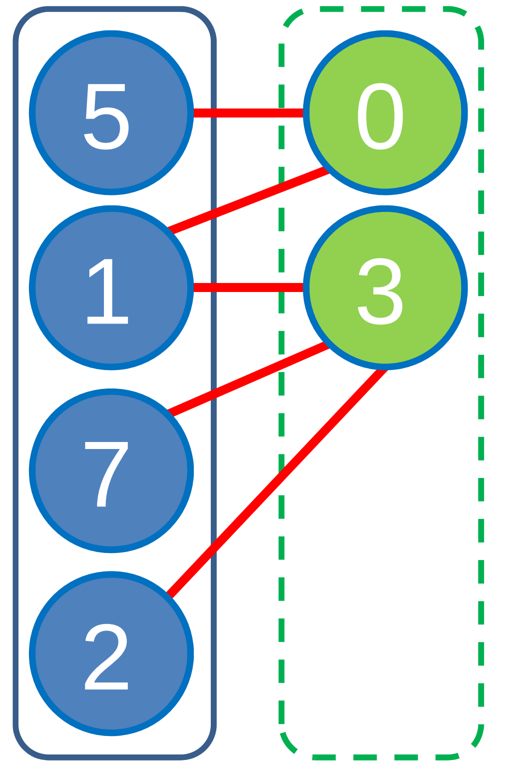
Min Cut Example





Bipartite Graph

SPECIAL GRAPHS 2



Matching
on Bipartite Graph

- In general graph
  - We need to use **Edmonds's Blossom Shrinking** (not discussed today) that is quite hard to code
  - DP + bitmask can only be used if input size is not too large **(N <= 20)**
    - More than that… MLE ☹
- In bipartite graph…
  - Let's see the next few slides ☺

PrimePairs (1)

- Group a list of numbers into pairs s.t the sum of each pair is prime.
- Given the numbers {1, 4, 7, 10, 11, 12}, you could group them as follows:
  1. 1 + 4 = 5   7 + 10 = 17   11 + 12 = 23
  2. 1 + 10 = 11   4 + 7 = 11   11 + 12 = 23
- Task: Given a int[] **numbers**, return a int[] of all the elements in **numbers** that could be paired with **numbers[0]** successfully as part of a *complete* pairing, sorted in ascending order.
- The answer for the example above would be {4, 10}.
  - Even though 1 + 12 is prime, there would be no way to pair the remaining 4 numbers.
- **Constraints:**
  - **numbers** contain an even number of elements in [2 .. 50], inclusive.
  - Each element of **numbers** will be between 1 and 1000, inclusive.
  - Each element of **numbers** will be distinct.

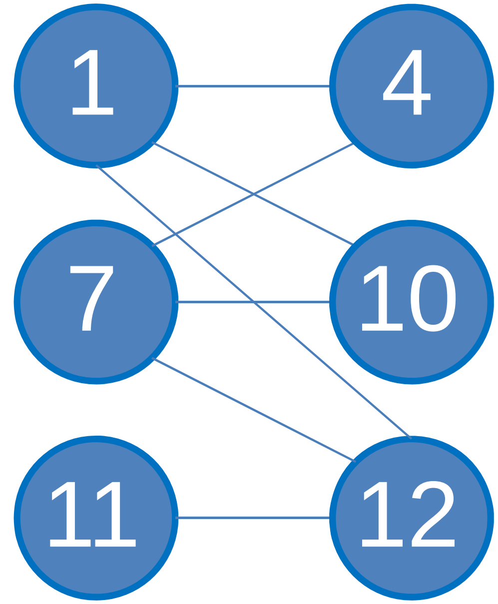
PrimePairs (2)

- Sample test cases (Source: TCO09 Qual 1):
  - Input: {1, 4, 7, 10, 11, 12}, Output: {4, 10}
    - This is the example from the problem statement.
  - Input: {11, 1, 4, 7, 10, 12}, Output: {12}
    - The same numbers, but in a different order.
    - In both of the 2 possible complete pairings, the 11 is paired with the 12.
  - Input: {8, 9, 1, 14}, Output: { }
    - No complete pairings are possible because none of the numbers can be paired with 1.
  - Input: {34, 39, 32, 4, 9, 35, 14, 17}, Output: {9, 39}
  - Input: {941, 902, 873, 841, 948, 851, 945, 854, 815, 898, 806, 826, 976, 878, 861, 919, 926, 901, 875, 864}, Output: {806, 926}

PrimePairs (3)

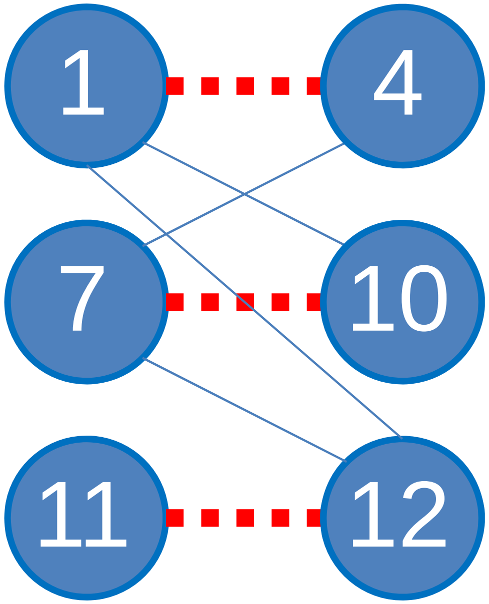
- Is this a Math problem?
  - Yes, there is a bit Math here, we need list of prime…
  - But elements of numbers are ≤ 1000!
    - This is not the major issue
- Complete Search Pairings?
  - ${}_{50}C_2$ for the first pair,
    ${}_{48}C_2$ for the second pair, …,
    until ${}_{2}C_2$ for the last pair?
    - This is too much…
    - This is the major issue

PrimePairs (4)

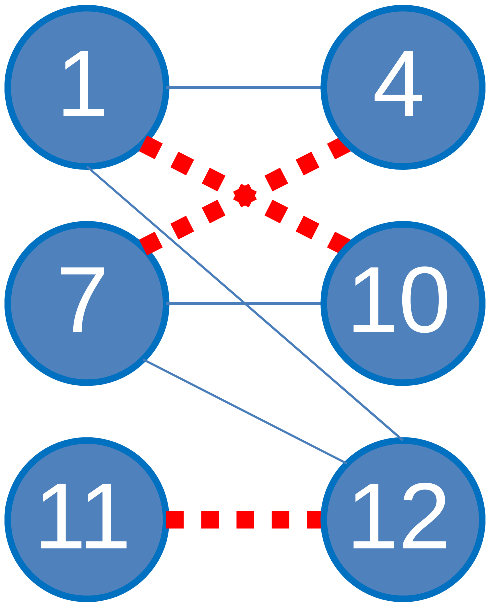


- Any keyword to help?
  - Pairing = Matching, familiar?
- Is this matching bipartite^?
  - YES :D
  - To get a prime, we need to sum 1 odd + 1 even
    - 1 odd + 1 odd = even number → not a prime
    - 1 even + 1 even = even number → not a prime
  - Split odd/even numbers to left/right set
    - Give edge from left to right if left[i] + right[j] = prime

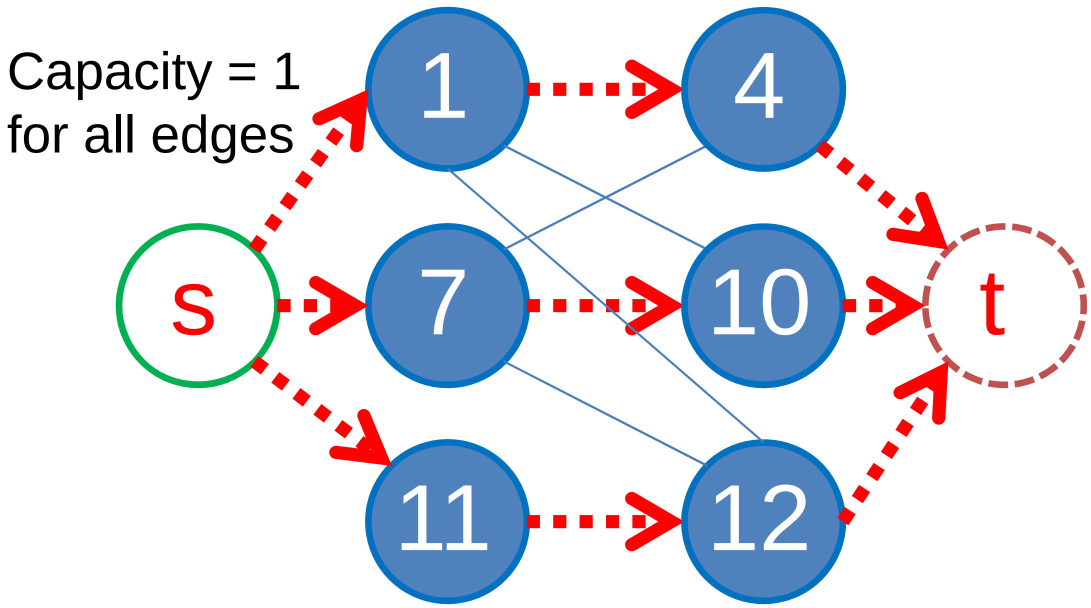
PrimePairs (5)



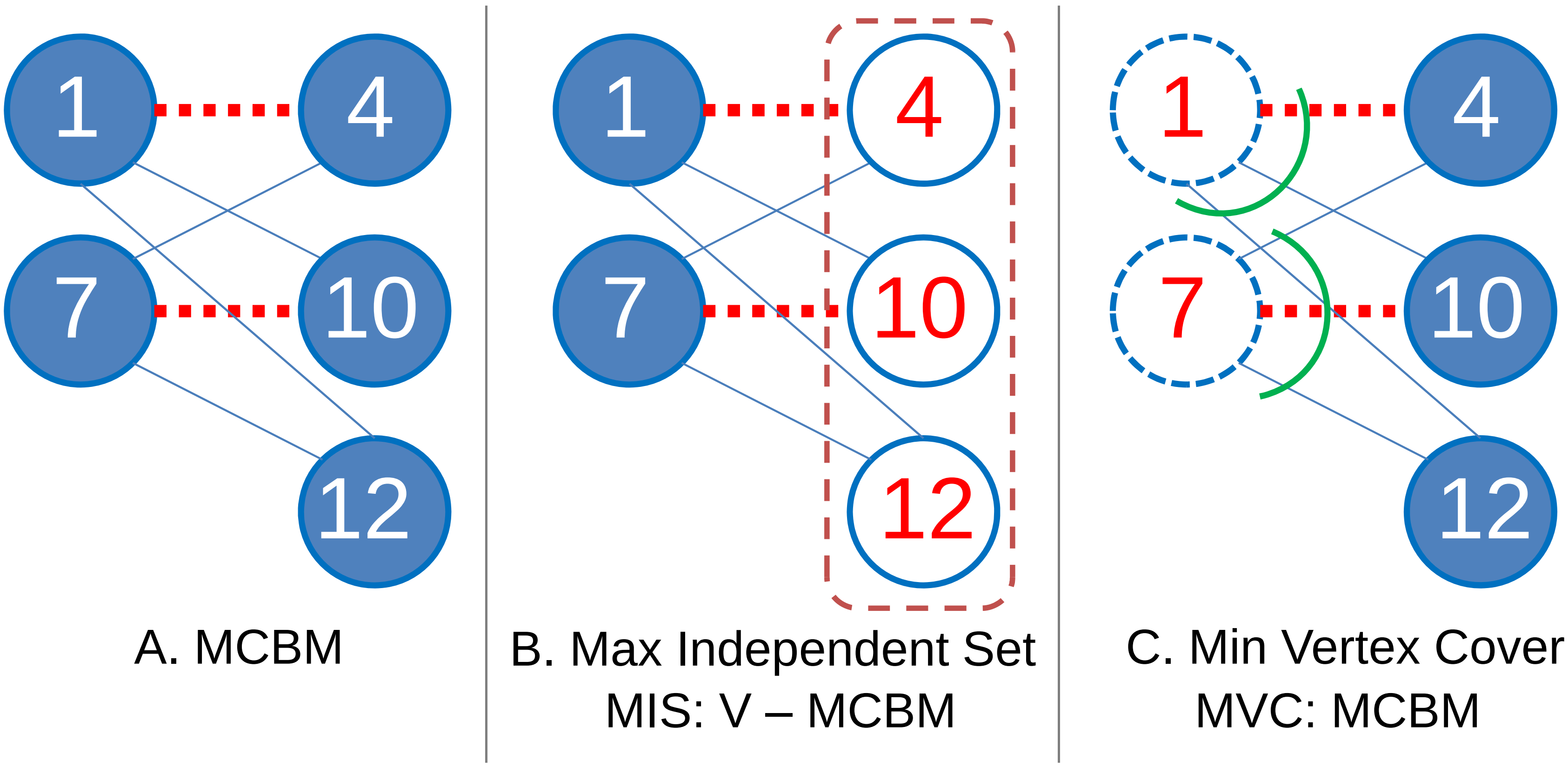
- Solution is then trivial:
  - If size of even and odd set are different
    - Pairing is not possible…
  - Otherwise, if size of both sets are n/2
    - Try to match left[0] with right[k]
      for k = [0 .. n/2 - 1]
      - Do *Bipartite Matching* for the rest
      - If we obtain n/2-1 more matchings
        - » Add right[k] to the answer
    - For this test case,
      we get 1 + 4 and 1 + 10 as the answer



PrimePairs (6)



Can Be Tricky!
Independent Set / Vertex Cover



A. MCBM

B. Max Independent Set
MIS: V – MCBM

C. Min Vertex Cover
MVC: MCBM

(König's theorem)

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

References

- CP2, Chapter 4 ☺
- 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…