COP 3530: Computer Science III
Summer 2005

Graphs and Graph Algorithms – Part 8

Instructor : Mark Llewellyn
markl@cs.ucf.edu
CSB 242, 823-2790

http://www.cs.ucf.edu/courses/cop3530/summer05

School of Computer Science
University of Central Florida

Residual Capacity

- Intuitively, the residual capacity defined by a flow f is any additional capacity that f has not fully taken advantage of in "pushing" its flow from source to sink.
- Let π be a path from source to sink (from s to t) that is allowed to traverse edges in either the forward or backward direction.
  - In other words, we can traverse the edge $e = (u,v)$ from its origin u to its destination v or from its destination v to its origin u.
- More formally a forward edge of π is an edge of e of π such that, in going from s to t along path π, the origin of e is encountered before the destination of e.
- An edge which is not a forward edge is said to be a backward edge.

Residual Capacity (cont.)

- Lets' extend the definition of the residual capacity to an edge *e* in π traversed from *u* to *v*, so that $\Delta_f(e) = \Delta_f(u,v)$. In other words,

$$\Delta_f(e) = \begin{cases} c(e) - f(e) & \text{if } e \text{ is a forward edge} \\ f(e) & \text{if } e \text{ is a backward edge} \end{cases}$$

- Thus, the residual capacity of an edge *e* going in the forward direction is the additional capacity of *e* that *f* has yet to consume, but the residual capacity in the opposite direction is the flow that *f* has consumed (and could potentially "give back" if that allows for another flow of higher value).

Augmenting Paths in Flow Networks

- The residual capacity $\Delta_f(\pi)$ of a path π is the minimum residual capacity of its edges. That is,

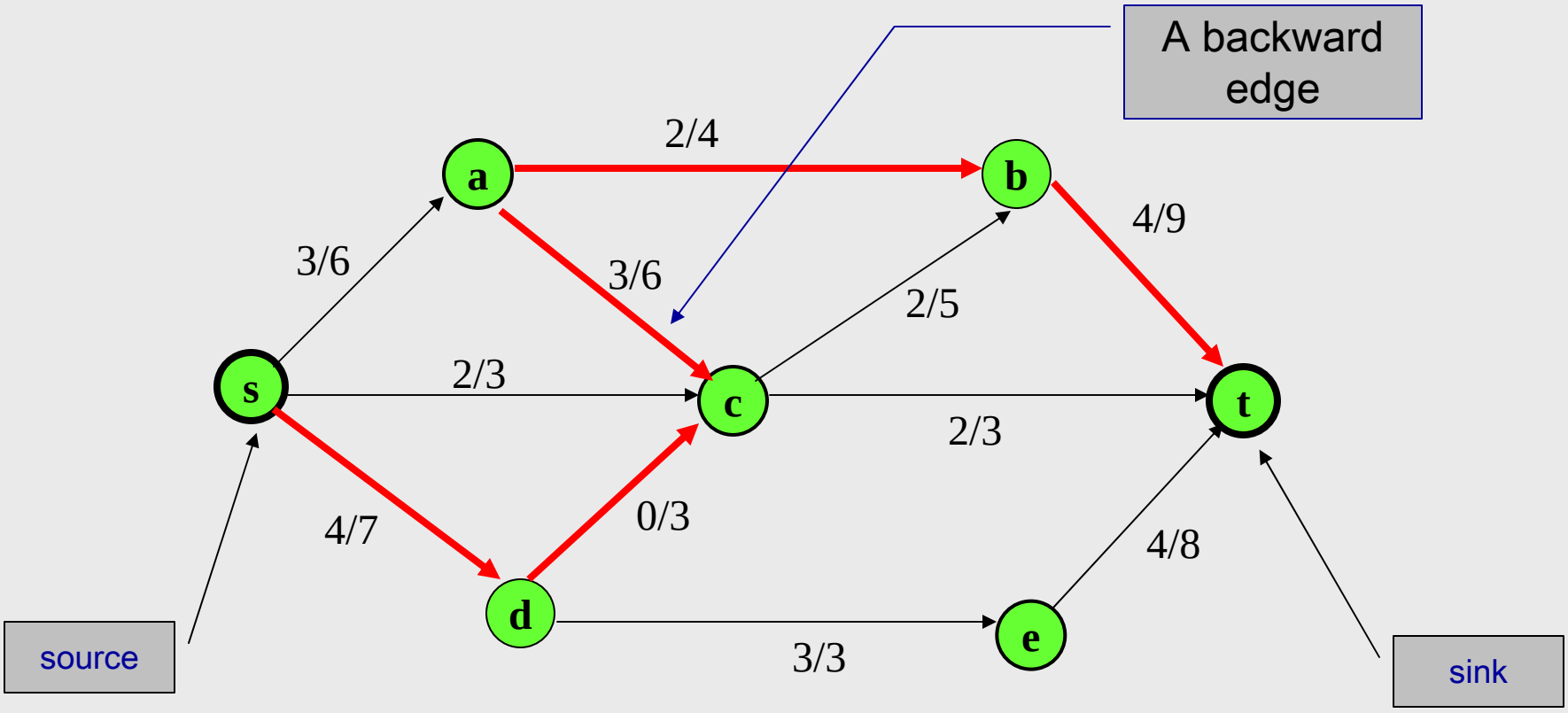
$$\Delta_f(\pi) = \min_{e \in \pi} \Delta_f(e)$$

- This value is the maximum amount of additional flow that we can possibly "push" down the path π without violating a capacity constraint.

- An augmented path for flow *f* is a path π from source to sink with nonzero residual capacity, that is, for each edge *e* of π,

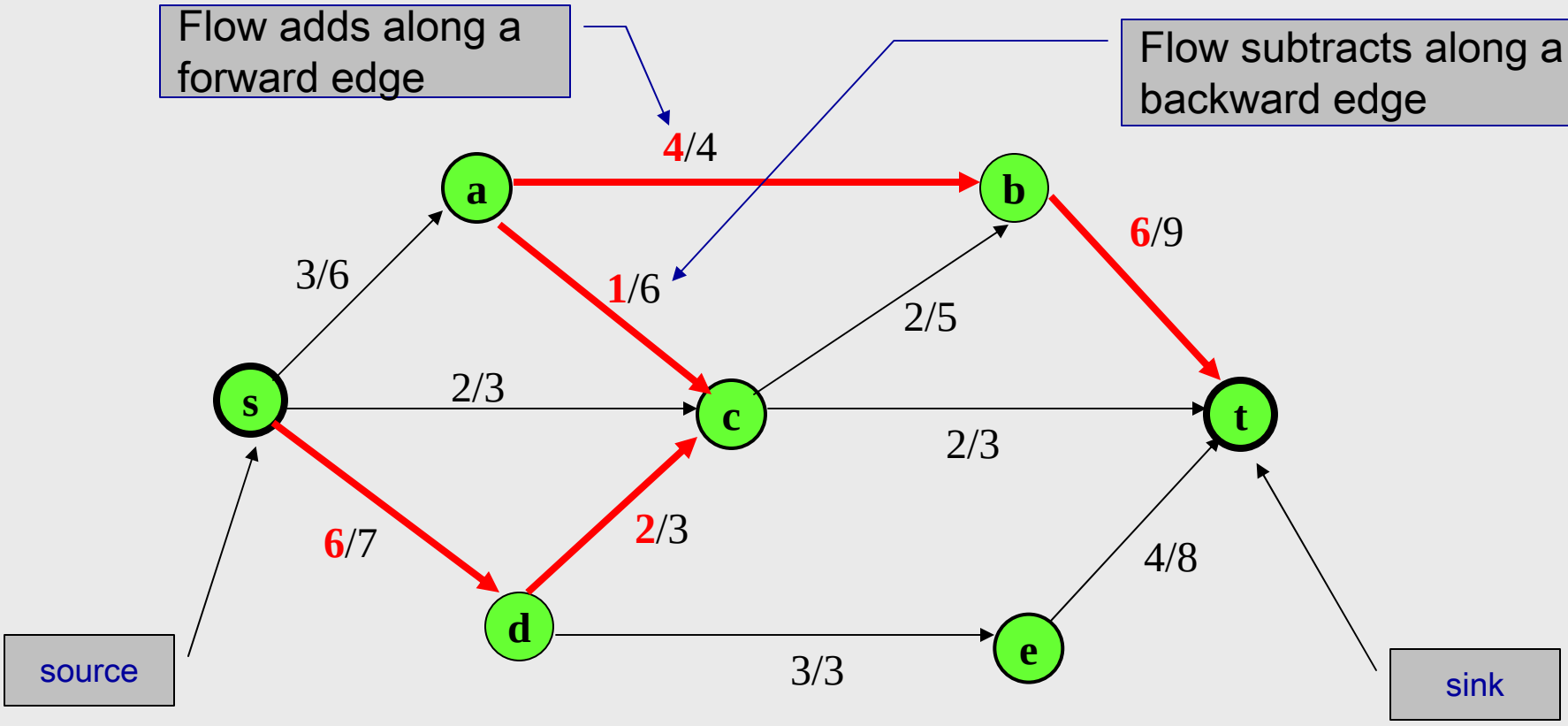
  - $f(e) < c(e)$ if *e* is a forward edge
  - $f(e) > 0$ if *e* is a backward edge

Augmenting Paths In Flow Networks (cont.)



An augmented path (highlighted in red)

Augmenting Paths In Flow Networks (cont.)



Flow f^* obtained from previous flow f by pushing $\Delta_f(\pi) = 2$ units of flow from s to t along path π.

Augmenting Paths in Flow Networks (cont.)

- It can be shown that it is always possible to add the residual capacity of an augmenting path to an existing flow and get another valid flow.
- Thus, the existence of an augmenting path π for a flow f implies that f is not maximum. (If f were maximum we could not add an augmenting path to determine another flow.)
- Also, given an augmenting path π, we can modify f to increase its value by pushing $\Delta_f(\pi)$ units of flow from source to sink along path π.
- What happens if there is no augmenting path for a flow f in a network N ?
- Lemma: If a network N does not have an augmenting path with respect to a flow f, then f is a maximum flow. Also, there is a cut X of N such that $|f| = c(X)$.

Augmenting Paths in Flow Networks (cont.)

Theorem:

Let *N* be a flow network. Given any flow *f* for *N* and any cut X of *N*, the value of *f* does not exceed the capacity of X, that is:

$$|f| \leq c(X)$$

Lemma:

If a network *N* does not have an augmenting path with respect to a flow *f*, then *f* is a maximum flow. Also, there is a cut *X* of *N* such that $|f| = c(X)$.

Augmenting Paths in Flow Networks (cont.)

- As a consequence of the theorem and lemma shown on the previous page, we have the following fundamental result relating maximum flows and minimum cuts.

> The Max-Flow, Min-Cut Theorem
>
> The value of a maximum flow is equal to the capacity of the minimum cut.

- The classic algorithm, due to Ford and Fulkerson, computes a maximum flow in a network by applying the greedy method to the augmenting path approach used to prove the Max-Flow, Min-Cut Theorem.

The Ford-Fulkerson Algorithm

- The main idea of the Ford-Fulkerson algorithm is to incrementally increase the value of a flow in stages, where at each stage some amount of flow is pushed along an augmenting path from the source to the sink.
- Initially, the flow of each edge is equal to 0.
- At each stage, an augmenting path π is computed and an amount of flow equal to the residual capacity of π is pushed along π.
- The algorithm terminates when the current flow f does not admit an augmenting path.
- The Ford-Fulkerson algorithm is shown on the next page.

The Ford-Fulkerson Algorithm (cont.)

Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop

This additional step was not included in the earlier algorithm that we examined.

Ford-Fulkerson Algorithm - Example

Initialize all edges of the flow graph with zero flow

Loop while ∃ a path in ***Gr*** from ***s*** to ***t***

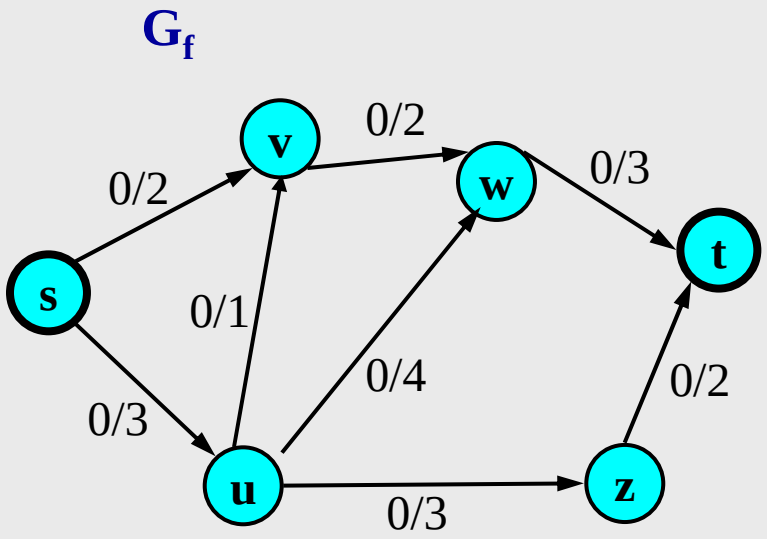
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)

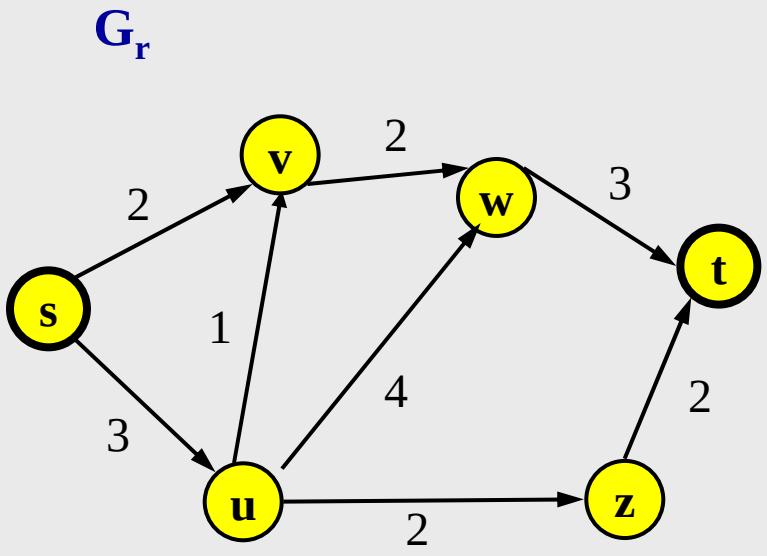
    Add to the flow graph the minimum residual capacity from this path

    Reduce the residual capacity of the edges

    Add a reversed path in the residual graph

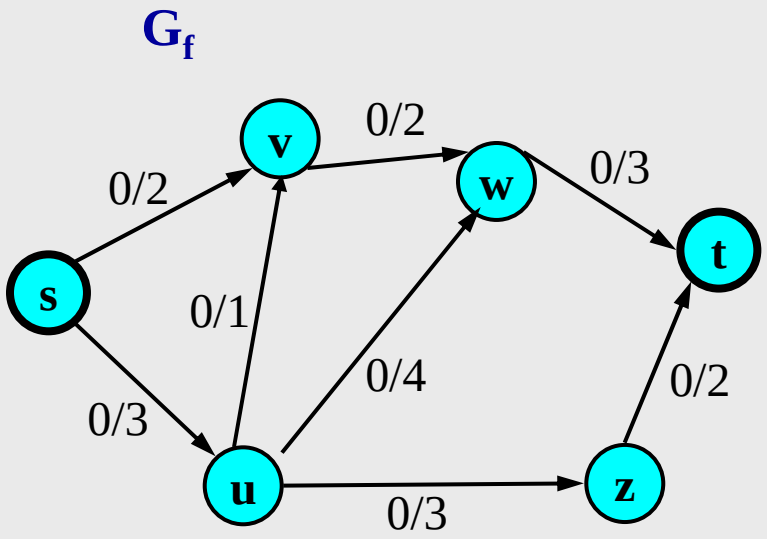
end Loop

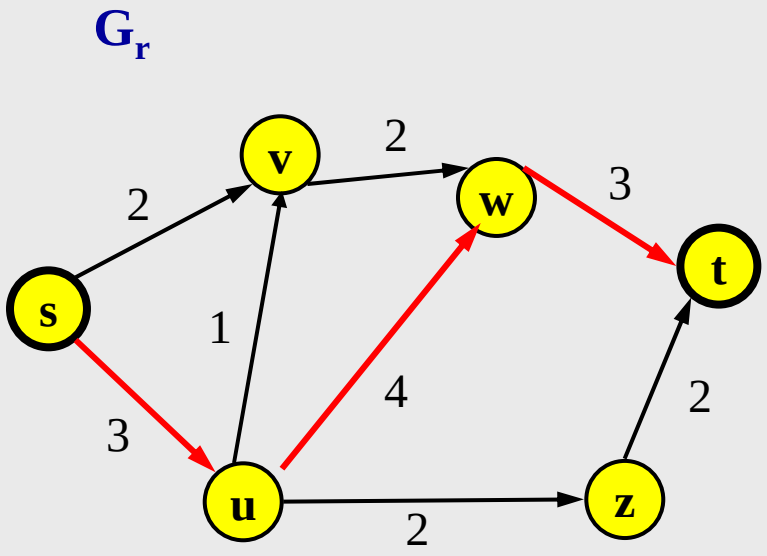




Ford-Fulkerson – Example (cont.)

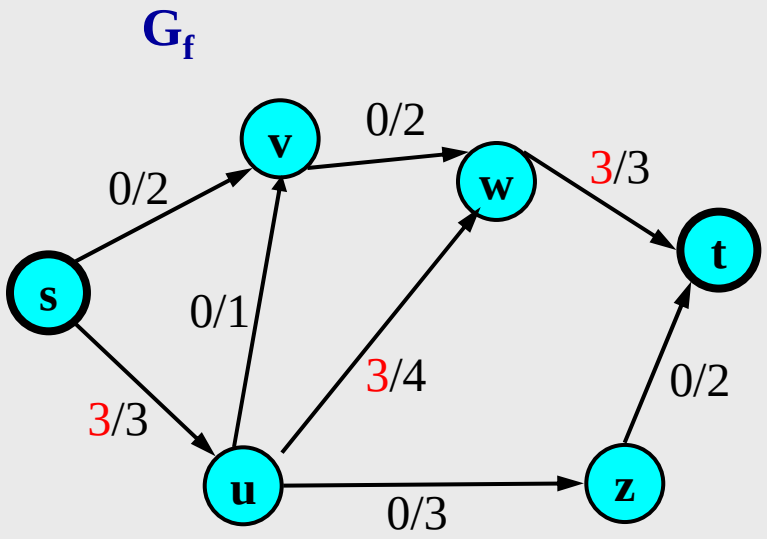
Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop

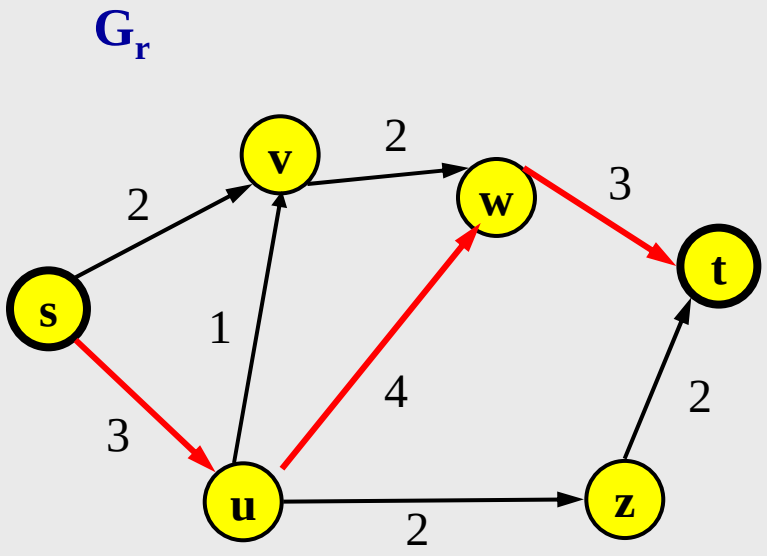




Ford-Fulkerson – Example (cont.)

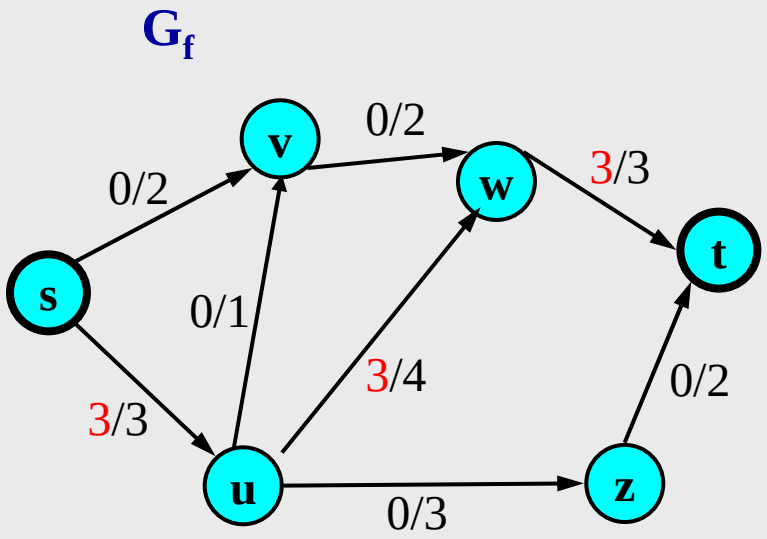
Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop

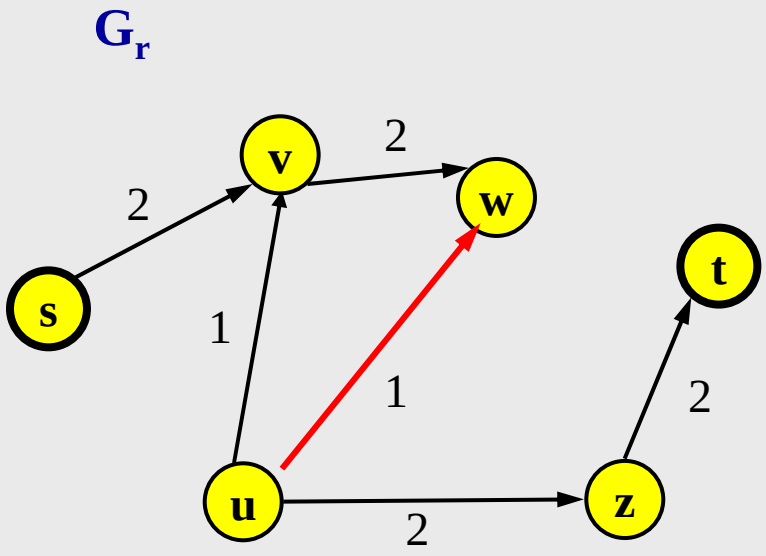




Ford-Fulkerson – Example (cont.)

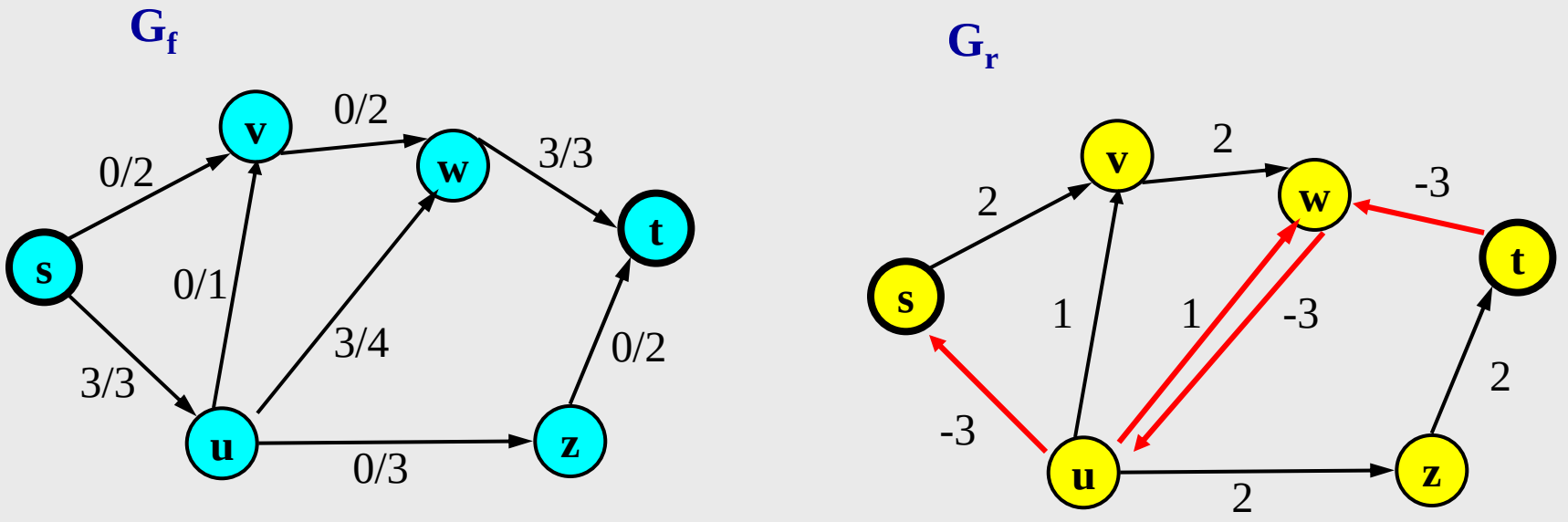
Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop





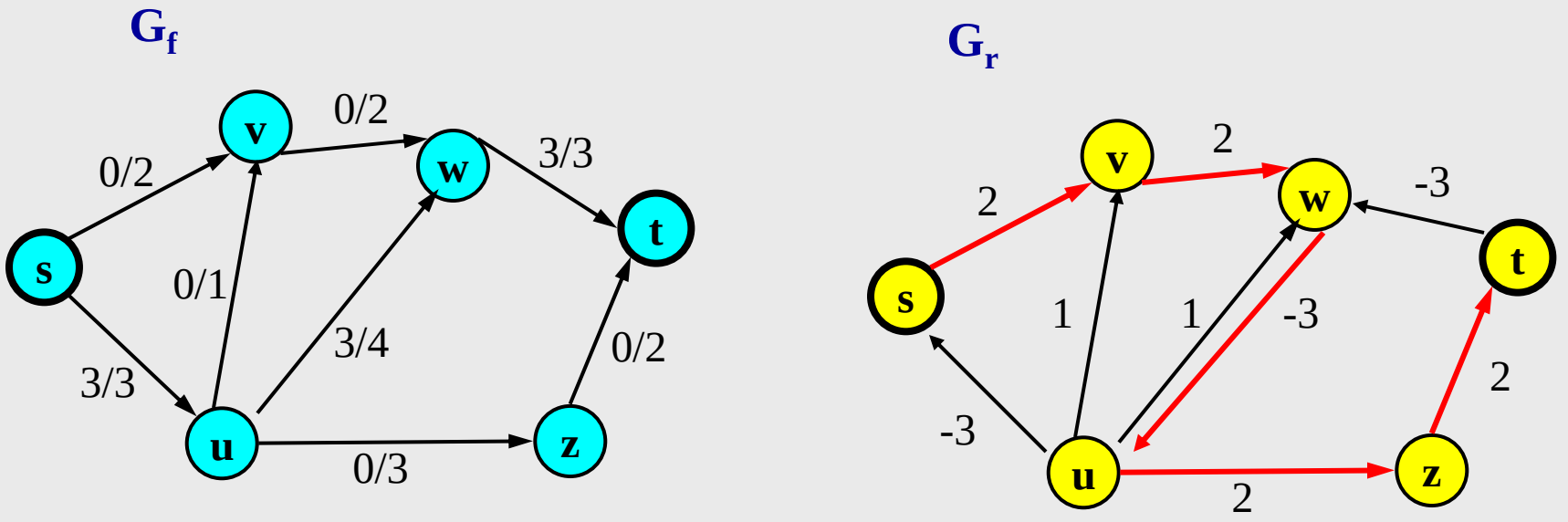
Ford-Fulkerson – Example (cont.)

Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop



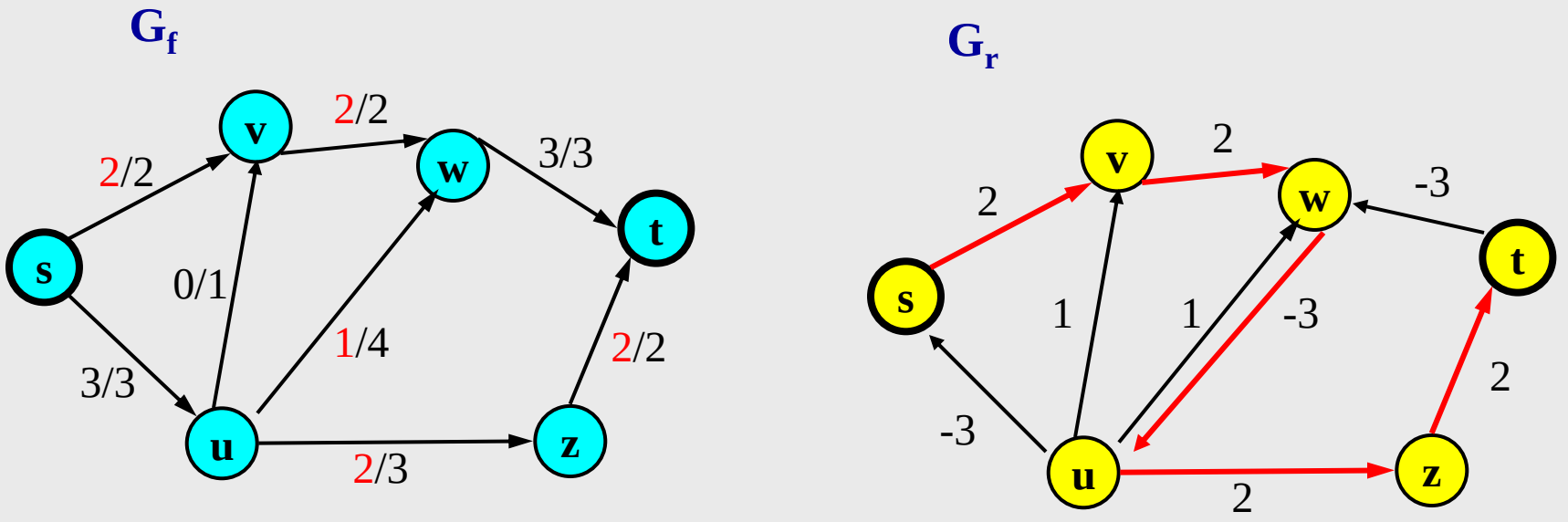
Ford-Fulkerson – Example (cont.)

Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop



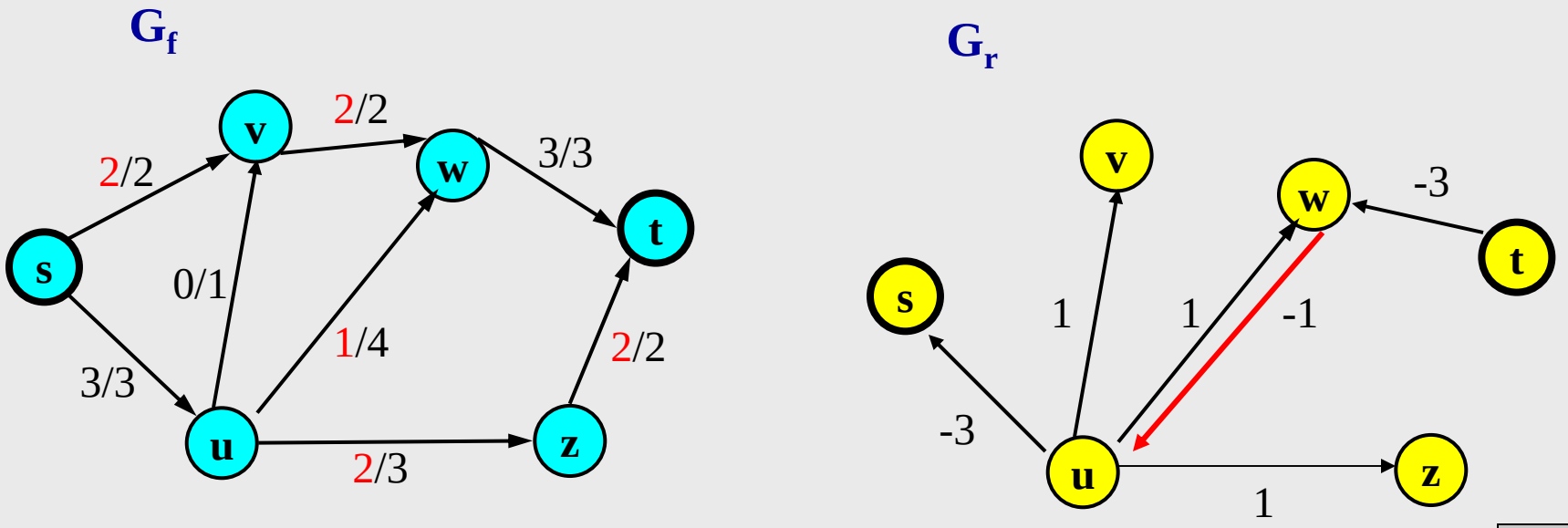
Ford-Fulkerson – Example (cont.)

Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop



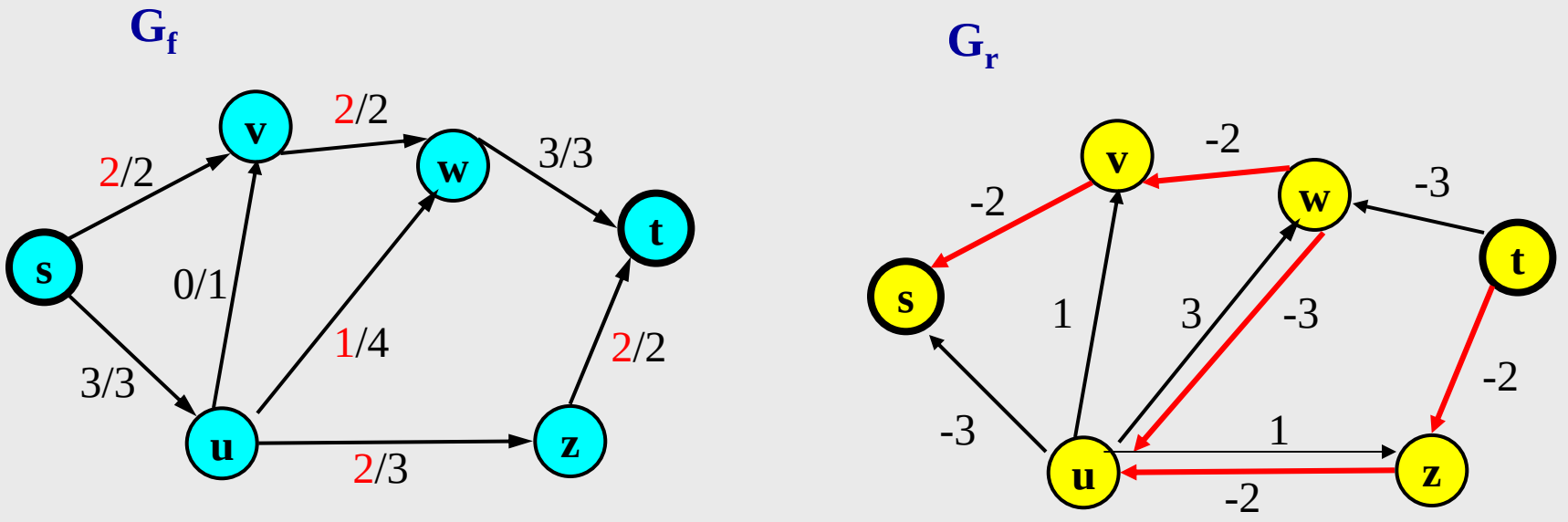
Ford-Fulkerson – Example (cont.)

Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop



Ford-Fulkerson – Example (cont.)

Initialize all edges of the flow graph with zero flow
Loop while ∃ a path in ***Gr*** from ***s*** to ***t***
    find a path in ***Gr*** from ***s*** to ***t*** (augmenting path)
    Add to the flow graph the minimum residual capacity from this path
    Reduce the residual capacity of the edges
    Add a reversed path in the residual graph
end Loop

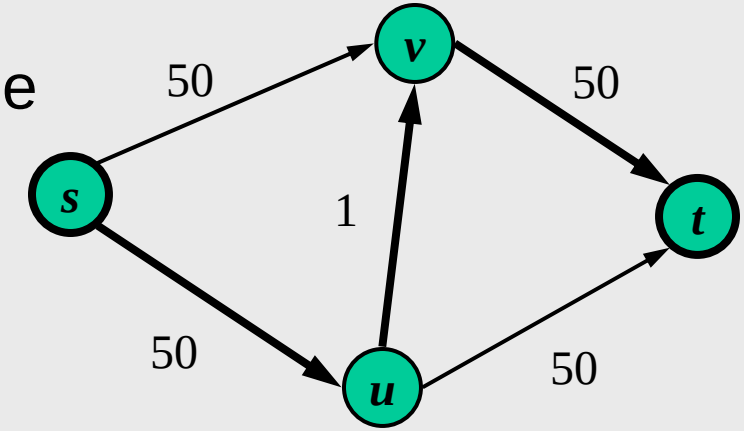


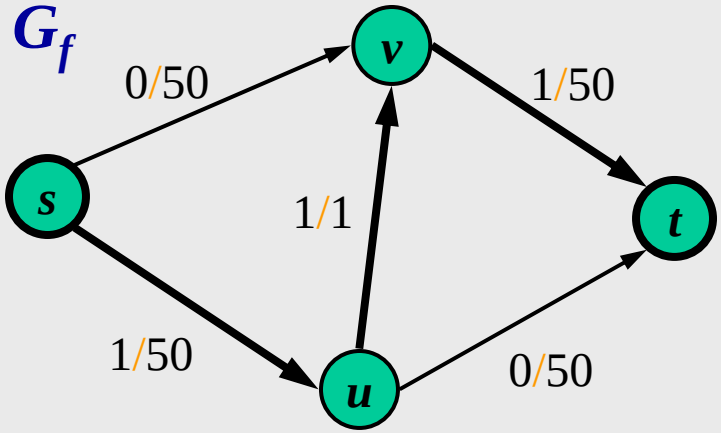
Ford-Fulkerson - Analysis

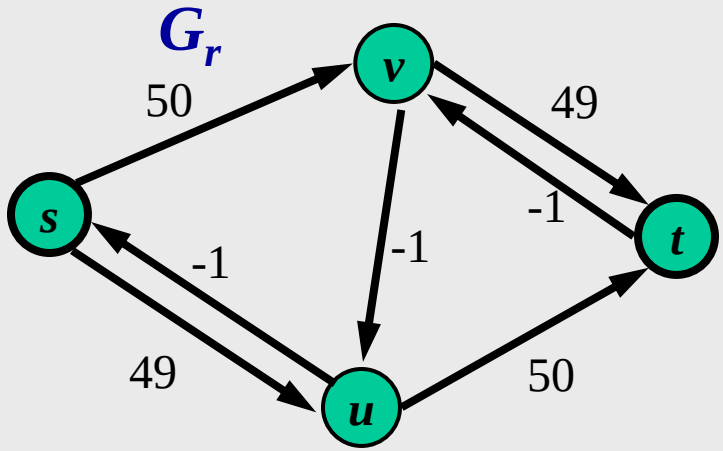
- Each augmenting path increases the flow value by at least 1.
- Let $|f^*|$ is a maximum flow, then in the worst case, Ford-Fulkerson's algorithm performs $|f^*|$ flow augmentations.
- Finding an augmenting path and augmenting the flow takes $O(n + m)$ time
- The running time of Ford-Fulkerson's algorithm is $O(|f^*|(n + m))$

Ford-Fulkerson - Analysis

Classic worst case example:

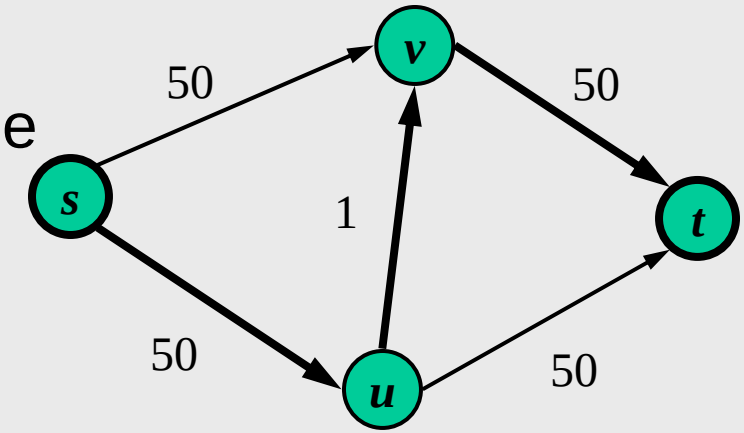




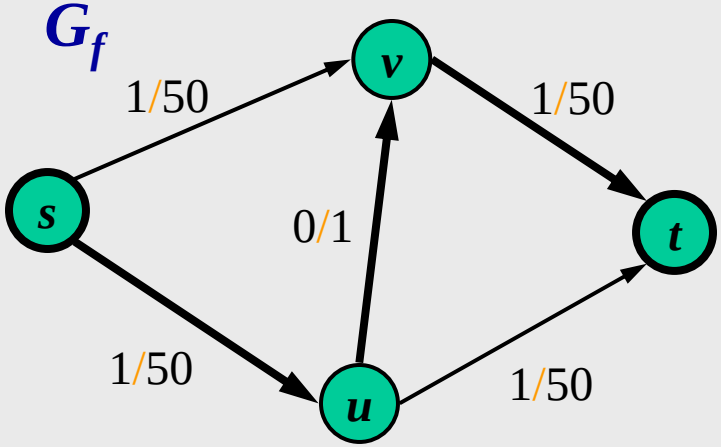


Ford-Fulkerson - Analysis

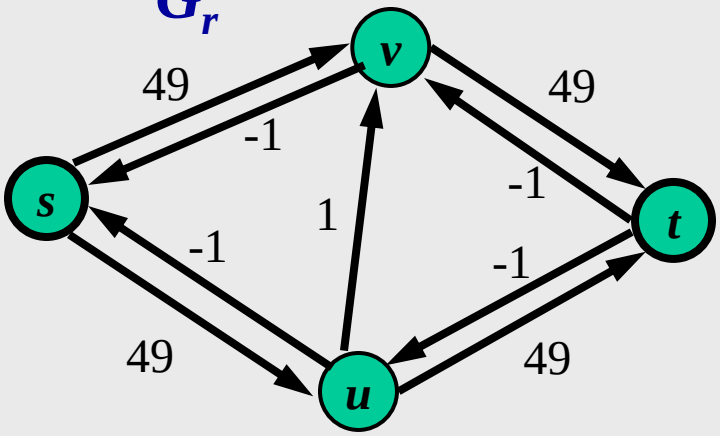
Classic worst case example:



G_f



G_r



Hamiltonian Graphs

- A Hamiltonian cycle in a graph is a cycle that passes through all the vertices of the graph.
- Recall that we mentioned this type of cycle when we were examining Euler paths and Euler circuits.
- A graph is called a Hamiltonian graph if it includes at least one Hamiltonian cycle.
- There is no formula which characterizes a Hamiltonian graph as there was with an Euler graph. However, it should be obvious that **all** complete graphs are Hamiltonian. The question is how to find a Hamiltonian cycle in a graph.
- The answer lies in the following theorem:

Hamiltonian Graphs (cont.)

Theorem:

If $edge(vu) \notin E$, graph $G* = (V, E \cup \{edge(vu)\})$ is Hamiltonian, and $degree(v) + degree(u) \geq |V|$, then graph $G = (V, E)$ is also Hamiltonian.

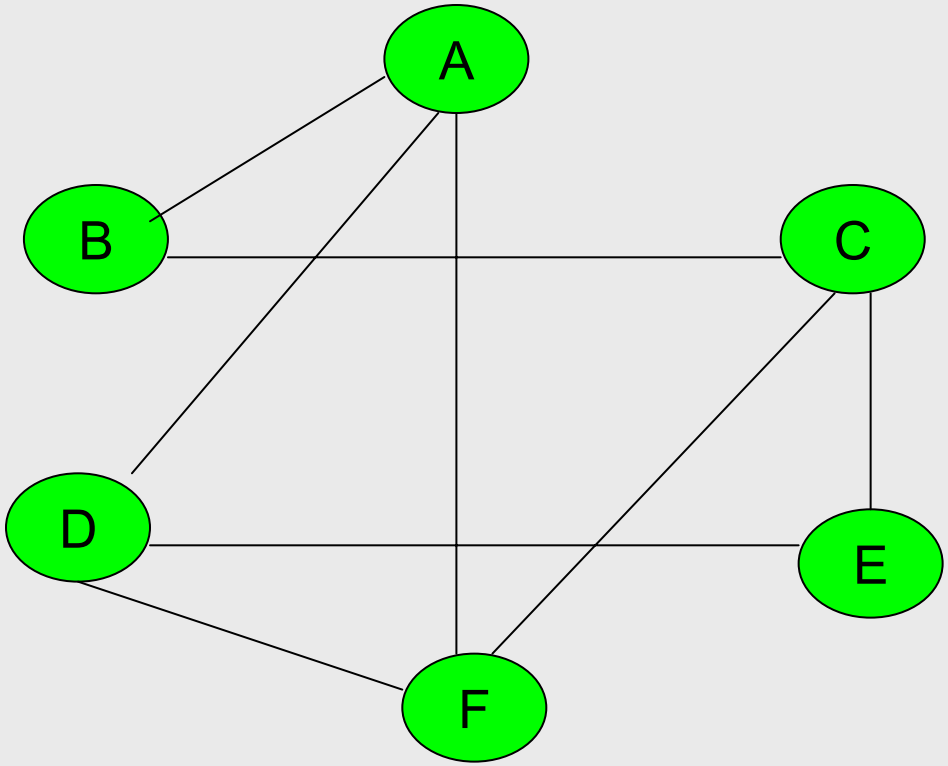
Basically, this theorem says that some Hamiltonian graphs allow us to create Hamiltonian graphs by eliminating some of the edges in the graph. The theorem leads directly to an algorithm that first expands the original graph to a graph with more edges in which finding a Hamiltonian cycle is easy (a complete graph) and then manipulates the Hamiltonian cycle by adding some edges and removing other edges so that eventually a Hamiltonian cycle is formed that includes the edges that belong to the original graph. An algorithm that is based on this theorem was developed by Chvátal in 1985.

Hamiltonian Graphs (cont.)

```
HamiltonianCycle( graph G = V, E))
   set label of all edges to 0;
   k = 1;
   H = E;
   GH = G;
   while GH contains nonadjacent vertices v, u where degH(v) + degH(u) ≥ |V|
         H = H ∪ {edge(vu)};
         GH = (V, H);
         label (edge(vu)) = k++;
         if there exists a Hamiltonian cycle C
            while (k = max{label(edge(pq)) : edge(pq) ∈ C}) > 0
              C = a cycle due to a crossover with each edge labeled  with number < k;
```


Hamiltonian Graph - Example

- Consider the following initial graph:



Hamiltonian Graph – Example (cont.)

- The complete graph is constructed from the initial graph as follows:
- In each iteration, two nonadjacent vertices are connected with an edge if the total number of their neighbors is not less than the number of all vertices in the graph.
- First look at all the vertices not adjacent to vertex *A*. For vertex *C*, degH(*A*) + degH(*C*) = 3+3 = 6 ≥ $|V|$ = 6, thus *edge(AC)* labeled with number 1 is included in H. Next vertex *E* is considered, and since the degree of *A* just increased by 1 when it acquired new neighbor vertex *B*, we have *degH(A)* + *degH(E)* = *4+2* = 6, *edge(AE)* labeled with 2 is included in H.

Hamiltonian Graph – Example (cont.)

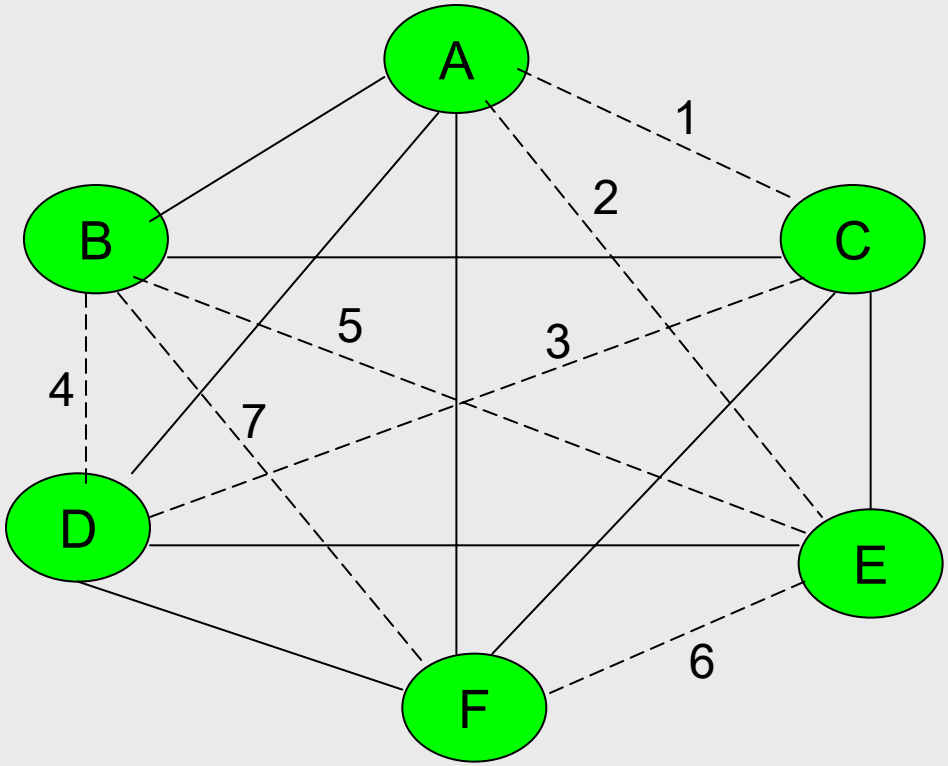
- The next vertex, for which new neighbors are attempted to be established, is *B* which is of degree 2. There are three non-adjacent vertices: *D*, *E*, and *F* with degrees 2, 2, and 3 respectively. Therefore, the sum of *B's* degree and the degree of any of these three vertices does not reach 6 and thus no edge is included in H.
- In the next four iterations, new neighbors are attempted for vertices *C*, *D*, *E*, and *F*. For node *C* the only nonadjacent node is *D* with degree 3 so we have degH(*C*) + degH(*D*) = 4+3 = 7 ≥ $|V|$ = 6 so *edge(CD)* labeled 3 is included in H).

Hamiltonian Graph – Example (cont.)

- For node *D* the only nonadjacent node is *B* with degree 2 so we have degH(*D*) + degH(*B*) = 4+2 = 6 ≥ $|V|$ = 6, thus *edge(BD)* labeled 4 is included in H.
- For node *E* the nonadjacent nodes are *B* and *F* both with degree 3, so we have degH(*E*) + degH(*B*) = 3+3 ≥ $|V|$ = 6, thus *edge(EB)* labeled 5 is added to H and since degH(*E*) + degH(*F*) = 4+3 = 7 ≥ $|V|$ = 6, *edge(EF)* labeled 6 is added to H.
- Finally, for node *F* the nonadjacent nodes are *B* and *E* both with degree 4, so we have degH(*F*) + degH(*E*) = 3+4 = 7 ≥ $|V|$ = 6, thus *edge(FE)* labeled 6 is added to H and since degH(*F*) + degH(*B*) = 3+5 = 8 ≥ $|V|$ = 6, *edge(EB)* labeled 7 is added to H.
- The final complete graph is shown on the next page.

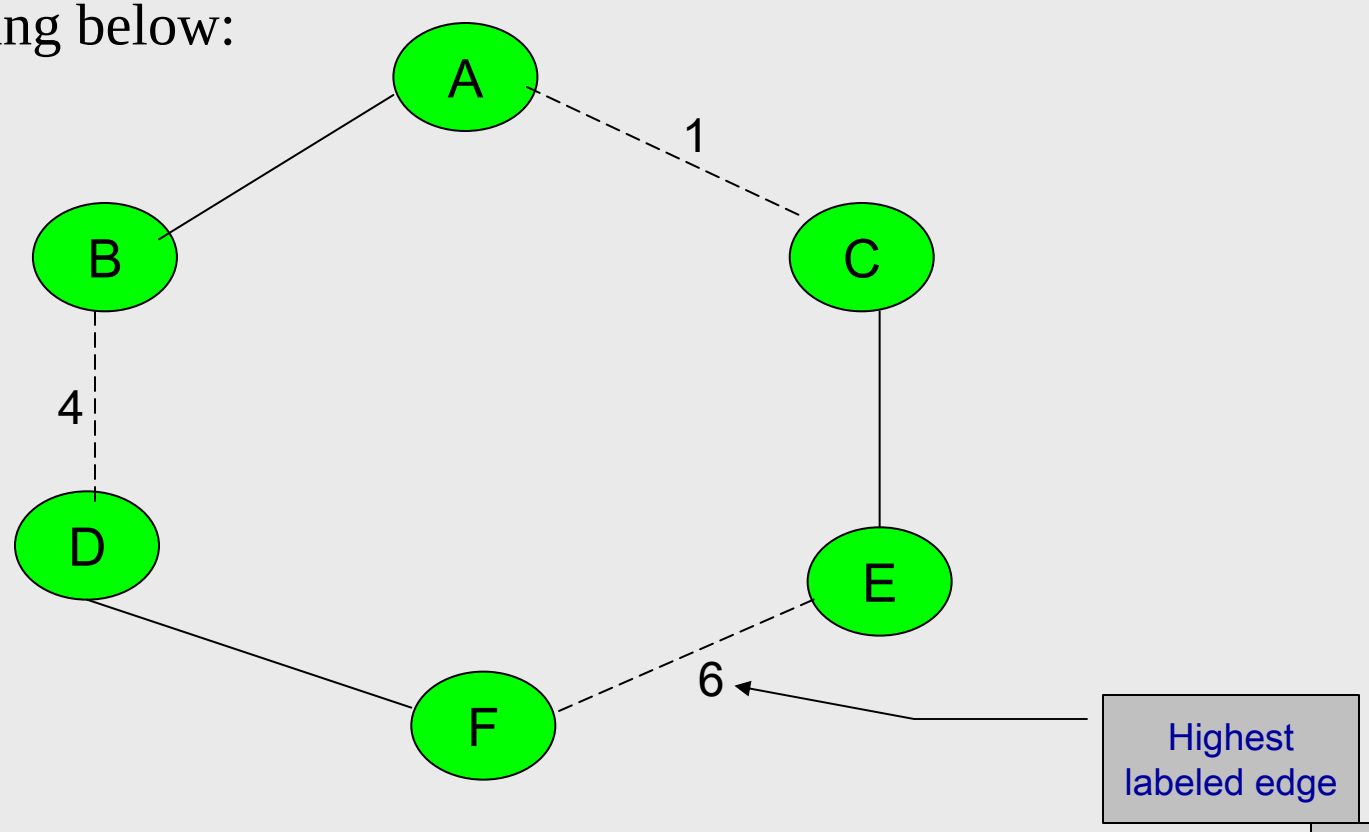
Hamiltonian Graph - Example

- The initial graph including all edges required to complete the graph:



Hamiltonian Graph - Example

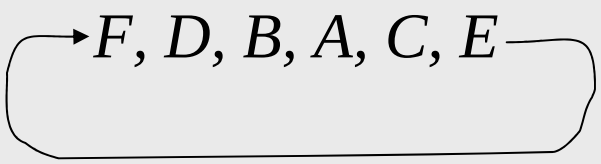
- In the second phase of the Hamiltonian cycle algorithm, a Hamiltonian cycle in H is found, which is: *A*, *C*, *E*, *F*, *D*, *B*, *A*. In this cycle, an edge with the highest label is found, *edge(EF)* with label 6, as shown in the drawing below:



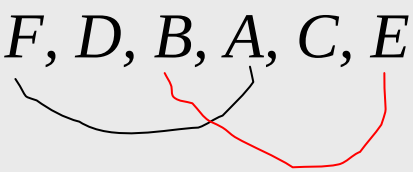
Hamiltonian Graph - Example

- The vertices in the cycle are so ordered that the vertices in this highest labeled edge are on the extreme ends.
- So the original Hamiltonian cycle becomes: *F*, *D*, *B*, *A*, *C*, *E*. Then moving left to right in the this new sequence of edges in the cycle, we try to find crossover edges by checking edges from the two neighbor vertices to the vertices at the end of the sequence so that the edges cross each other.
- The first possibility is vertices *D* and *B* with *edge(BF)* and *edge(DE)*, but this pair is rejected because the label of *edge(BF)* is greater than the largest label of the current cycle which is 6. After this, the vertices *B* and *A* and the edges connecting them to the ends of the sequence *edge(AF)* and *edge(BE)* are checked; the edges are acceptable (their labels are 0 and 5), so the old cycle *F*, *D*, *B*, *A*, *C*, *E*, *F* is transformed into a new cycle *F*, *A*, *C*, *E*, *B*, *D*, *F*.

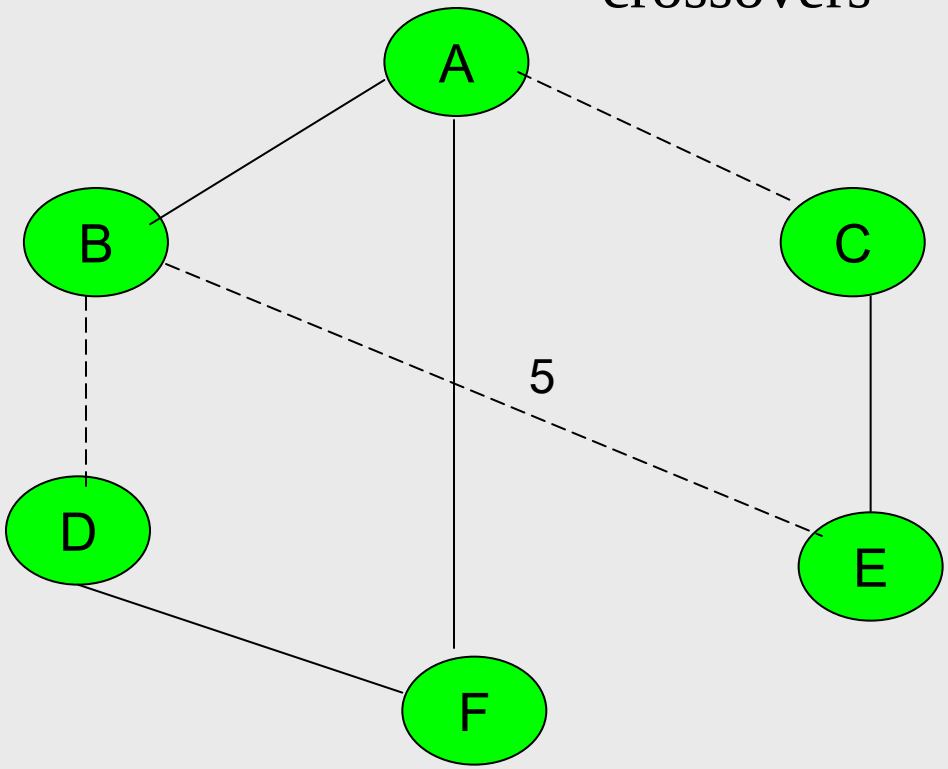
Hamiltonian Graph - Example



Current cycle



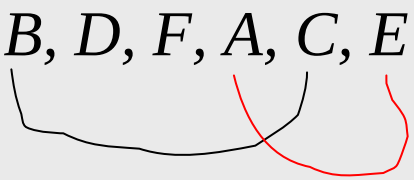
crossovers



Hamiltonian Graph - Example

- In this new modified cycle, the *edge(BE)* has the highest label with value 5, so the cycle is presented with the vertices of this edge at the extreme ends of the sequence: *B*, *D*, *F*, *A*, *C*, *E*. To find the crossover edges, the first pair of edges *edge(BF)* and *edge(DE)* , but the label of *edge(BF)* = *7* which is greater than the largest label of the current cycle, so the pair is rejected. The next pair is *edge(AB)* and *edge(EF)*, however, once again, the pair is unacceptable since the label on *edge(EF)* is 6. The next possibility is the pair *edge(BC)* and *edge(AE)*, which is acceptable since the highest label is 2. Thus, the new cycle *B*, *C*, *E*, *A*, *F*, *D*, *B* is formed.

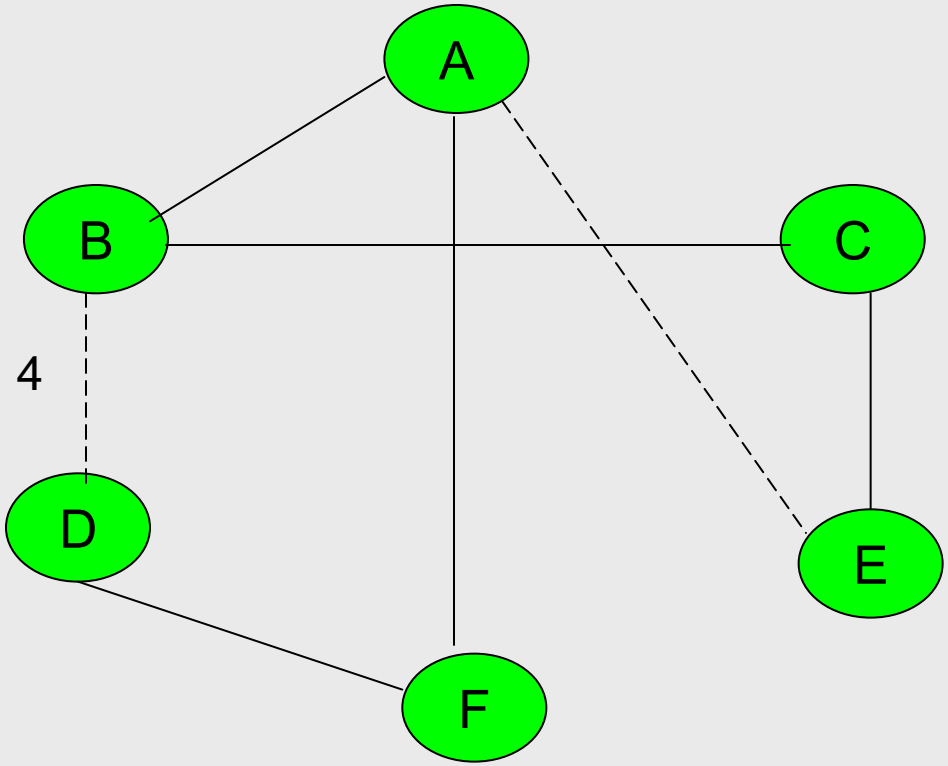
Current cycle



crossovers

Hamiltonian Graph - Example

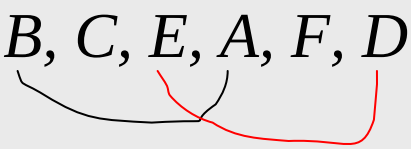
Current cycle



Hamiltonian Graph - Example

- Finally, in this latest cycle, a pair of crossover edges is found, *edge(AB)* and *edge(DE)* that are acceptable since labels on both edges are 0 and a new cycle is formed which finally includes only edges with labels of 0 and thus are edges which appeared in the original graph and the algorithm terminates with the following Hamiltonian cycle including only edges from G.

Current cycle

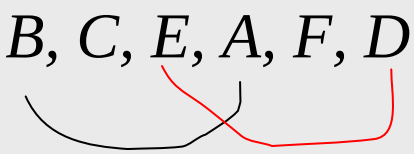


crossovers

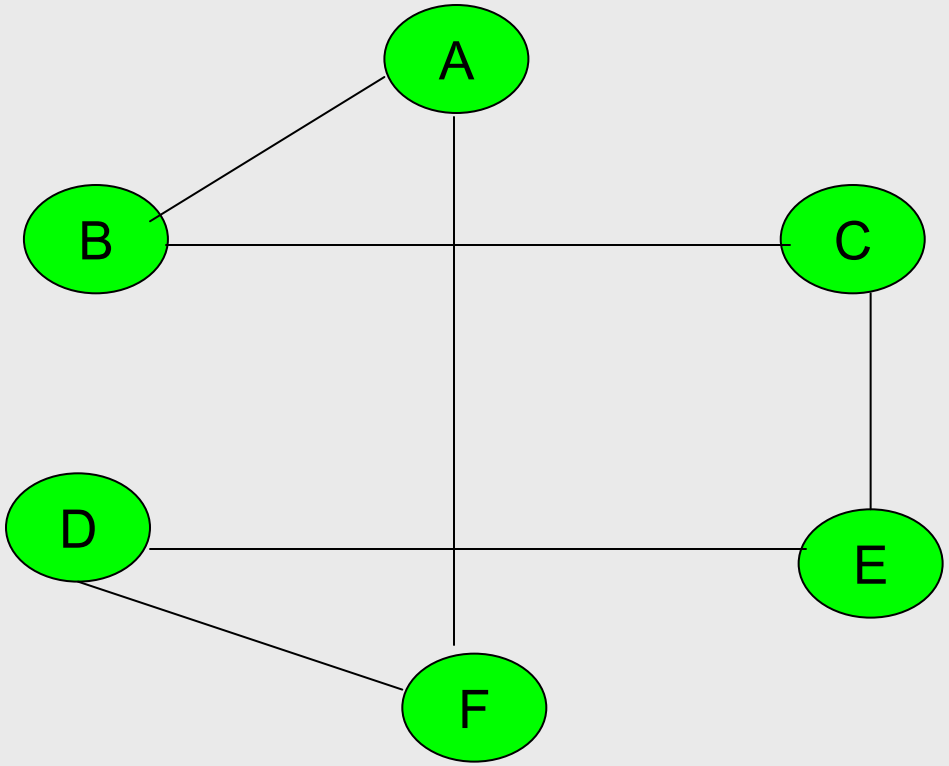
The final Hamiltonian cycle is shown in the next diagram.

Hamiltonian Graph - Example

Current cycle



crossovers



Final Hamiltonian Cycle is:

B,A,F,D,E,C,B