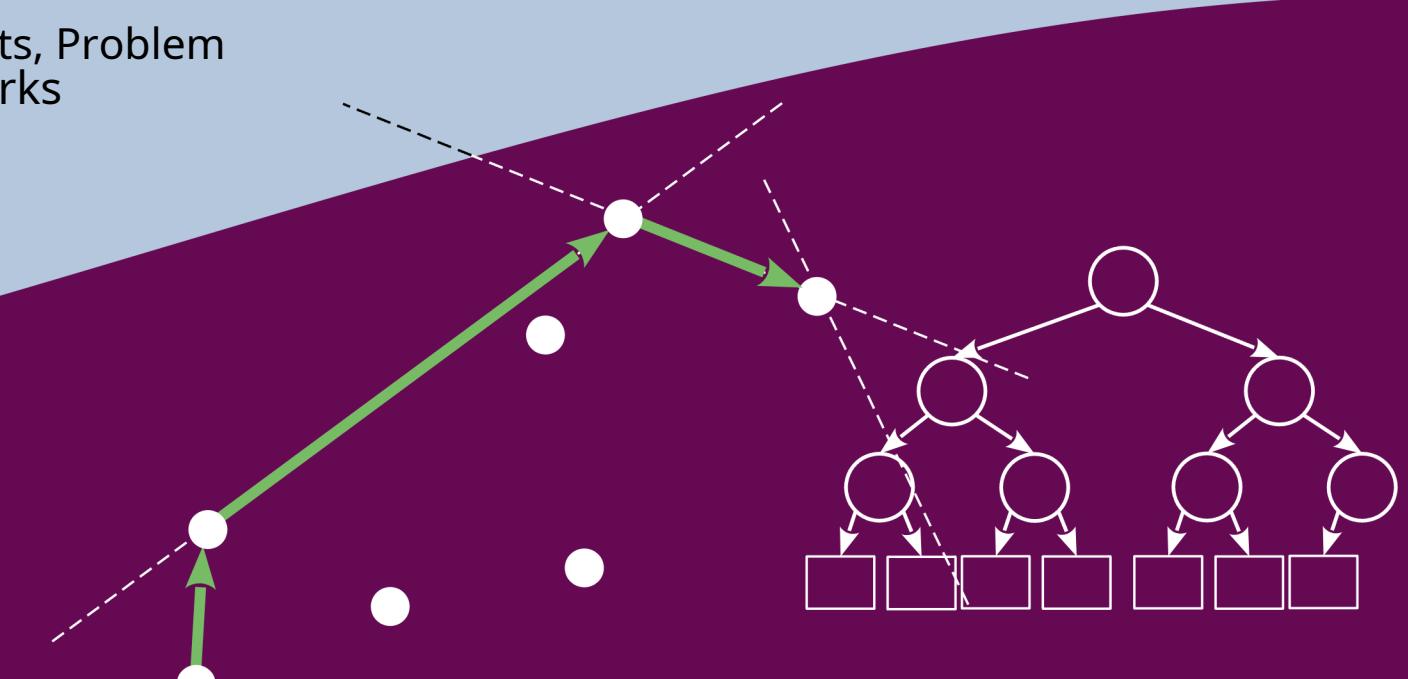
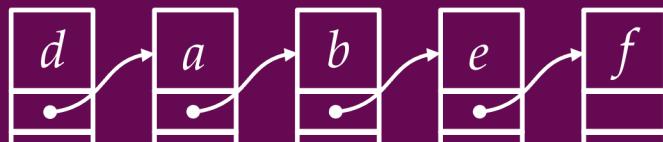
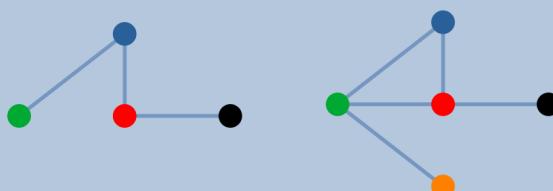


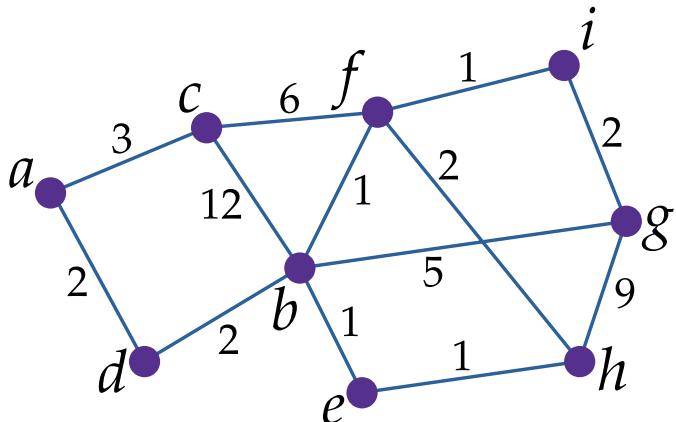
Advanced algorithms and structures data

Week 11: Algorithms over graphs

Chinese postman problem, Hamiltonian circuits, Problem
traveling salesman, Flow in networks

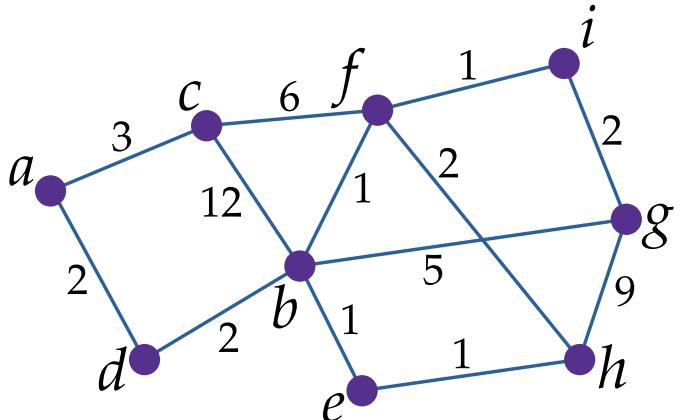


The problem of the Chinese postman



- The Chinese postman problem is similar to the seven bridges problem
- A Chinese postman needs to go around a village modeled by a weight graph in the most efficient way possible
 - The edges represent the streets of the settlement, and the weights on the edges represent the cost (*effort, effort*) which the postman must pay to cross that street
 - After visiting the settlement, the postman returns to the peak from which he started
- If the settlement is modeled by an Euler graph containing an Euler circle, then the matter is clear - There is always exactly one tour
- If the settlement is not modeled by an Euler graph, then we have an optimization problem to solve

The problem of the Chinese postman



- We have already established that an Euler graph containing an Euler circle must have all vertices of even degree
- If we have some of the vertices that have an odd degree, then we need a settlement graph *Eulerize*, in order to obtain the Euler circle
- The concept of Eulerization means adding edges to a graph so that all vertices of the graph have an even degree



Eulerization of the graph

- 1: Detect all vertices having an odd degree $ODD \subseteq V(G)$.
- 2: Calculate distances all-to-all between ODD vertices. You can use WFI (algorithm 4.9) for this purpose. We define the mapping set of shortest paths between pairs of vertices in ODD as

$$\forall v_i, v_j \in ODD : ODDPaths(v_i, v_j) = v_i v_{i+1} \dots v_{j-1} v_j$$

- 3: Construct a complete bipartite graph

$$X = ODD, Y = ODD$$

$$H = (X \cup Y, E')$$

such that weights of the bipartite graph edges are as:

$$w(v_{ii}) = \infty,$$

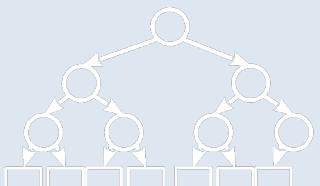
$$w(v_{ij}) = w_{ij}$$

where w_{ij} is the shortest distance between vertices $v_i, v_j \in ODD$, calculated in the step 2.

- 4: In the complete bipartite graph H find the optimal assignments M between vertices in X and Y , such that

$$M \subseteq X \times Y \Rightarrow \min \sum_{(v_i, v_j) \in M} w(v_i v_j)$$

- 5: **for** $(v_i, v_j) \in M$ such that $\deg(v_i)$ is still odd **do**
- 6: **for** $v_r v_s \in ODDPaths(v_i, v_j)$ **do**
- 7: Add the edge $v_r v_s$ to the edges $E(G)$

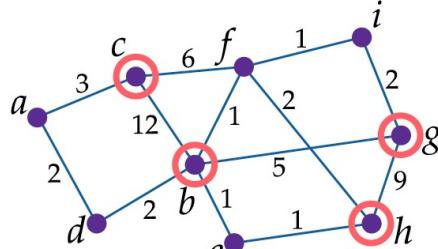


Eulerization – step 1 and 2

Step 1: Detecting the odd degree vertices

We start by detecting all vertices having an odd degree. We have the following vertices

$$ODD = \{b, c, g, h\}$$



Step 2: Using WFI, calculate shortest paths and distances in ODD

We get the following shortest distances

$$w(bc) = D_{bc}^9 = 7, w(bg) = D_{bg}^9 = 4$$

$$w(bh) = D_{bh}^9 = 2, w(cg) = D_{cg}^9 = 9$$

$$w(ch) = D_{ch}^9 = 8, w(gh) = D_{gh}^9 = 5$$

and shortest paths

$$ODDPaths(b, c) = bdac$$

$$ODDPaths(b, g) = bfig$$

$$ODDPaths(b, h) = beh$$

$$ODDPaths(c, g) = cfig$$

$$ODDPaths(c, h) = cfh$$

$$ODDPaths(g, h) = gifh$$

$$\mathbf{D}^0 = \begin{bmatrix} a & b & c & d & e & f & g & h & i \\ a & 0 & 0 & 3 & 2 & 0 & 0 & 0 & 0 \\ b & 0 & 0 & 12 & 2 & 1 & 1 & 5 & 0 \\ c & 3 & 12 & 0 & 0 & 0 & 6 & 0 & 0 \\ d & 2 & 2 & 0 & 0 & 0 & 0 & 0 & 0 \\ e & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ f & 0 & 1 & 6 & 0 & 0 & 0 & 0 & 2 \\ g & 0 & 5 & 0 & 0 & 0 & 0 & 0 & 9 \\ h & 0 & 0 & 0 & 0 & 1 & 2 & 9 & 0 \\ i & 0 & 0 & 0 & 0 & 0 & 1 & 2 & 0 \end{bmatrix}$$

$$\mathbf{D}^9 = \begin{bmatrix} a & b & c & d & e & f & g & h & i \\ a & 0 & 4 & 3 & 2 & 5 & 5 & 8 & 6 & 6 \\ b & 4 & 0 & 7 & 2 & 1 & 1 & 4 & 2 & 2 \\ c & 3 & 7 & 0 & 5 & 8 & 6 & 9 & 8 & 7 \\ d & 2 & 2 & 5 & 0 & 3 & 3 & 6 & 4 & 4 \\ e & 5 & 1 & 8 & 3 & 0 & 2 & 5 & 1 & 3 \\ f & 5 & 1 & 6 & 3 & 2 & 0 & 3 & 2 & 1 \\ g & 8 & 4 & 9 & 6 & 5 & 3 & 0 & 5 & 2 \\ h & 6 & 2 & 8 & 4 & 1 & 2 & 5 & 0 & 3 \\ i & 6 & 2 & 7 & 4 & 3 & 1 & 2 & 3 & 0 \end{bmatrix}$$

- In the first step, we find all vertices with an odd degree

- In the second step, we calculate the shortest all-all distances within the set

- In addition, we determine the shortest paths
- WFI was used in the example, there are other options

Eulerization – steps 3, 4 and 5

Step 3 and 4: Generate the bipartite graph H and find the optimal assignments M

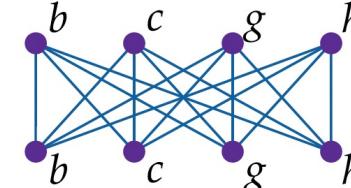
In this step we need to optimize the pairs of vertices of the complete bipartite graph H .

$$M_1 = \{(b, c), (g, h)\}, w(bc) + w(gh) = 7 + 5 = 12$$

$$M_2 = \{(b, g), (c, h)\}, w(bg) + w(ch) = 4 + 8 = 12$$

$$M_3 = \{(b, h), (c, g)\}, w(bh) + w(cg) = 2 + 9 = 11$$

The optimal assignment is M_3 , which is also giving the shortest Eulerian circuit in the final Eulerized graph.



b			c			g			h		
c	g	h	b	g	h	b	c	h	b	c	g
7	4	2	7	9	8	4	9	5	2	8	5

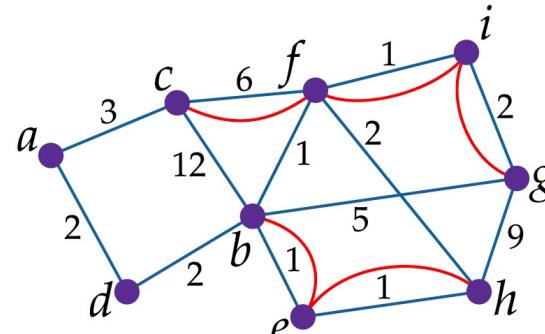
Step 5: Add edges from the shortest paths $ODDPaths$, according to the optimal assignments M

We iterate through M_3 to find the shortest paths edges.

1. $(b, h) \in M_3 : ODDPaths(b, h) = beh$. Edges be and eh are added to the graph G .

2. $(c, g) \in M_3 : ODDPaths(c, g) = cfig$. Edges cf , fi , and ig are added to the graph G .

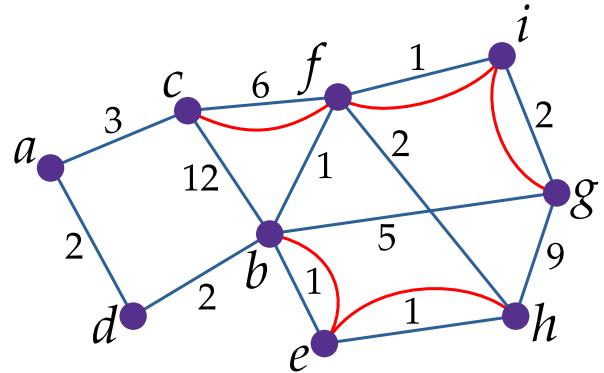
Finally, we get the Eulerized graph as seen on the right side in which all vertices have even degrees.



- We construct a bipartite graph from which we extract optimal pairs, which have the minimum sum of path weights

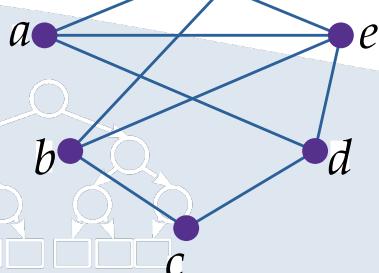
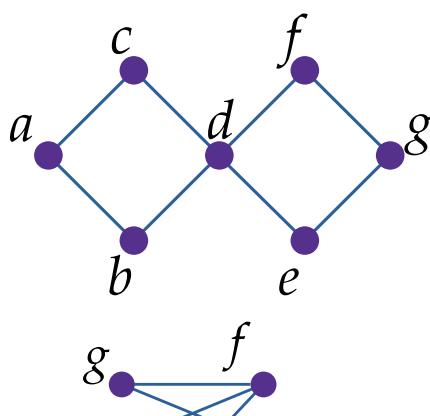
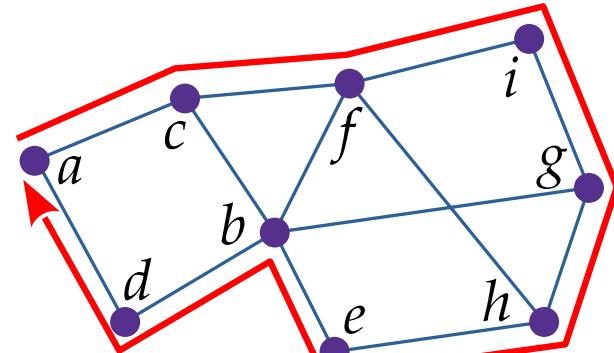
- The postman's optimal route
- It can be solved through a table
- At the end from h we take trajectories for optimal pairs
 - We add parallel edges along the paths
 - The initial vertex in the path gets only one adjacent edge - which makes its degree even
 - Intermediate vertices in the path get two adjacent edges, which does not change their parity/oddity current level

The problem of the Chinese postman



- After Eulerization, let's apply one of the algorithms for detecting the Euler circle to the graph
 - Visually, Fleury's algorithm is simpler
- An Euler circle is found that starts and ends at the peak from which the Chinese postman starts his tour
- The total sum of the weights on all edges of the Euler circle is the total cost of the Chinese postman's trip
 - All added edges in the Eulerization process have the same weight as the original edge between two vertices - namely, it is one and the same street
 - For example, the original edge and the added edge are both weight 6.

Hamiltonian graphs and cycles



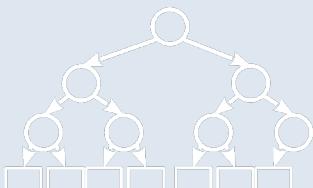
- **Hamilton's way(path)** – is a path through the graph that includes all vertices of the graph only once
- **Hamilton cycle(cycle)** – is a Hamilton path that starts and ends at the same vertex
- **Hamilton's graph**– is a graph made of a Hamiltonian cycle
- It is important to note that unlike the Euler circle, in the Hamiltonian cycle we can pass through each vertex only once and that cycle must include all the vertices of the graph - the Hamiltonian graph
- The problem is NP-complex

Graph closure

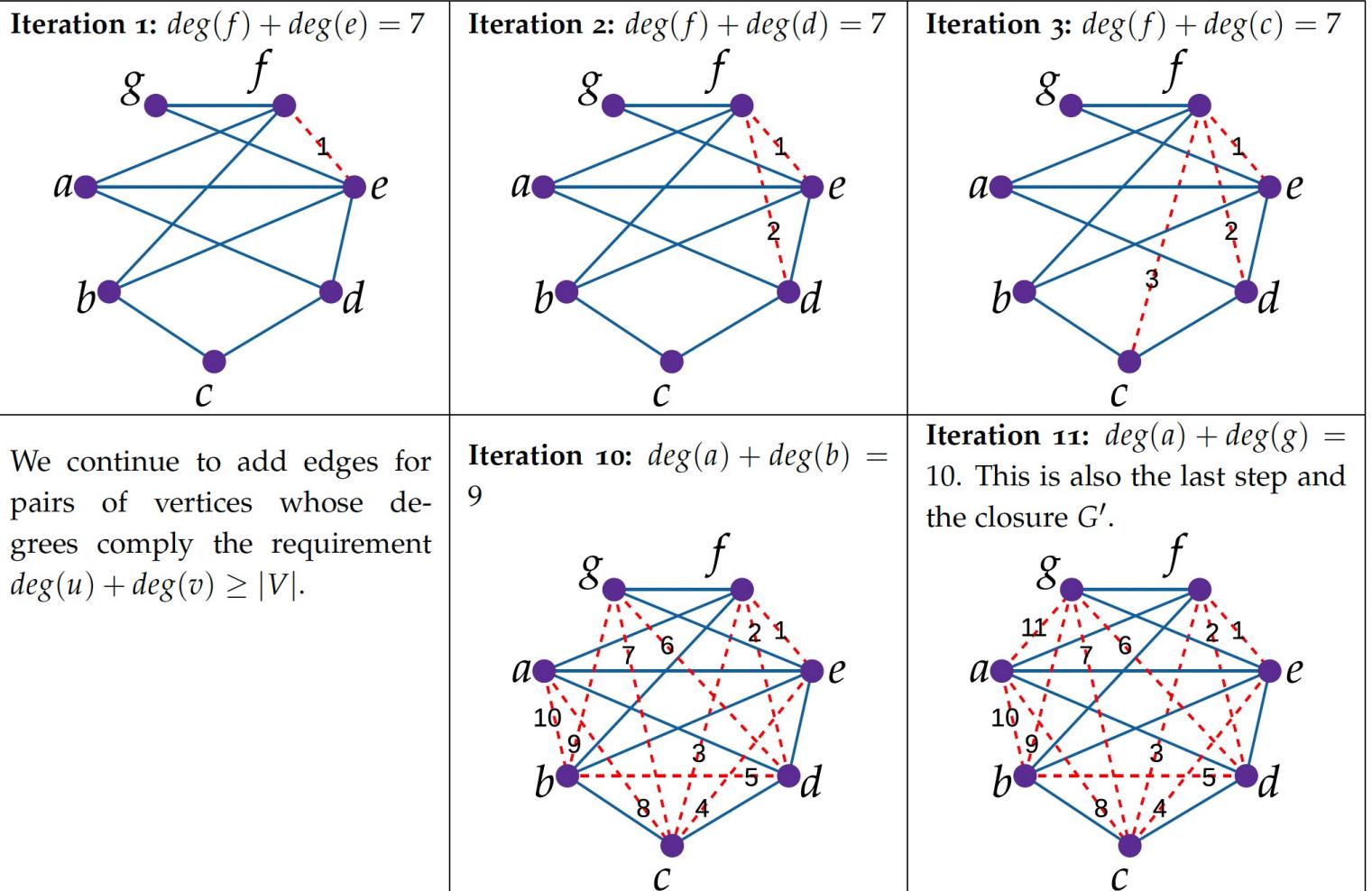
- If we have a graph ,then it is 'shutter (*closure*) graph which is obtained

```
G' ← G
i ← 1
while there are vertices  $u$  and  $v$ , such that  $\deg(u) + \deg(v) \geq |V(G)|$  do
    add the edge  $uv$  to the edges  $E(G')$ , mark it as the  $i$ -th edge
     $i \leftarrow i + 1$ 
```

- We add an edge to a couple of vertices ,whose sum of degrees is greater than or equal to the number of vertices in the graph($\deg(u) + \deg(v) \geq |V(G)|$)
 - Mark the added edge with an ordinal number
 - At the moment when we can no longer add a single edge, it is considered a closure '

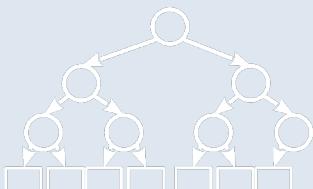


Graph closure

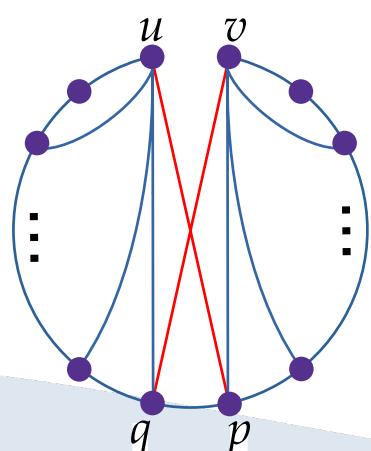
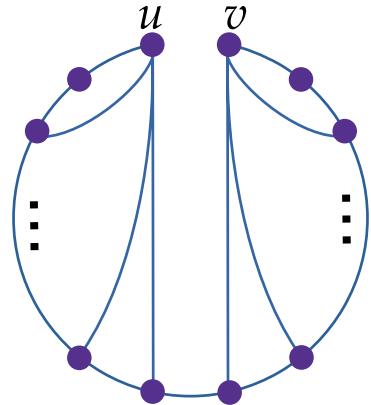


Bondy–Chvátal theorem

- **Theorem:** A simple undirected graph is Hamiltonian if and only if its closure is 'Hamilton's graph.'
- **Evidence 1:** If is a simple undirected graph Hamiltonian, then there is a subset of edges ' $\subseteq E(G)$ ' such that it forms a Hamiltonian cycle G . Shutter 'it is created by adding edges to the graph , what is it worth ' $\subseteq E(G) \subseteq E(H)$ ', which means that it is also contained in the closure ' H '.
- **Evidence 2:** Between graphs and its shutter 'we have a sequence of graphs
 $\#$, $\$$, $\%$, ..., $\&$, $\$$
• where are they $= H$ and $= \#$
• and each graph $\$$ has one more edge compared to the graph $\$ \% \&$



Bondy–Chvátal theorem



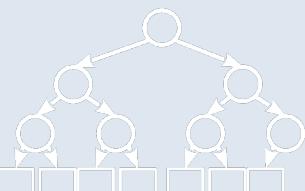
- According to the Bondy–Chvátal theorem, if the graph !contains a Hamiltonian cycle, then the graph also contains it !"#. This is subject to the principle of mathematical induction.
- In the above picture we have a graph !"#. In order to get the Hamiltonian cycle, we need to add an edge and thereby gain ! which contains it
 - If it is (!"#) = , then it is valid $\deg(u) + \deg(v) = -2$
 - This is why we cannot add an edge
 - To add an edge , we have to from the top and have two additional edges, which in the image below are marked as and
 - That's what we got $(\deg(u) + \deg(v)) =$
 - By adding the edge uv, we have the Hamilton cycle \$
 - We notice that in !"#we have a Hamiltonian cycle which is
$$\$ - + +$$

QED

Practical application of the Bondy–Chvátal theorem

- 1: $G_c \leftarrow$ Hamiltonian cycle in the closure G'
- 2: **while** there is added edge marked with ordinal in the graph G_c
do
- 3: pick an added edge uv from G_c having the highest ordinal i_{max}
- 4: pick an edge $pq \neq uv$ from G_c such that edges up and vq are from G' , crossed, and either from the original graph G or having ordinal less than i_{max}
- 5: remove edges uv and pq from the graph G_c
- 6: add edges up and vq to the graph G_c

- We are coming back from the shutter 'back
 - We found Hamilton's cycle in the shutter %
 - IN %we take the added edge with the highest ordinal number as an edge –remember that we are in the process of creating a shutter 'marked
 - IN %let's choose an edge ,such that there are crossed edges and in 'which are either from the graph or have a lower serial number than the edge
 - IN %remove the edges and ,and we add the edges up and vq
 - We do this until u %we have added edges - when there are none, END, we have found the Hamiltonian cycle in the graph

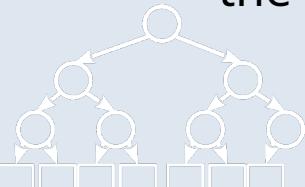


Practical application of the Bondy–Chvátal theorem

Step 1: Detect the Hamiltonian cycle in the closure G'	
We start by detecting the Hamiltonian cycle in the closure G' .	
Step 2: Replace edges ag and fe .	
We find ag as the edge having the highest ordinal $i_{max} = 11$, which is identified as the edge uv . The other edge fe is taken as the edge pq . We take crossed edges from the G' , which are both from the original graph G .	
Step 3: Replace edges ab and ed .	
We find ab as the edge having the highest ordinal $i_{max} = 10$, which is identified as the edge uv . The other edge ed is taken as the edge pq . Taking the edge ed is convenient, as we have crossed edges ad and be from the original graph G . After this step, we have no more added edges, which means we moved backwards to the graph $G = G_0$. The result is the Hamiltonian cycle in the original graph G .	

Traveling Salesman Problem (TSP)

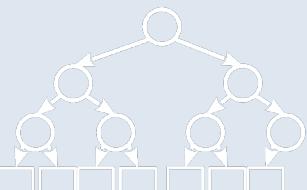
- The merchant starts from a certain point and needs to visit all his *customers*
- The vertices of the graph represent the places the trader visits – including the start/end point
 - The initial peak is also the final peak - we can see it as the trader's parent company
- Edges between vertices have the weights they denote *charge journeys* from one customer to another
 - You can think of this as a set of all the costs that a merchant needs to incur for traveling between two customers, say: time (number of days of travel), distance, transportation costs, etc...
 - It is necessary to find the most efficient route by which the trader visits all the customers in the graph - for the trader, this means the lowest cost of the trip



The problem of the traveling salesman

- Unlike the Chinese postman, the traveling salesman problem is focused on the vertices of the graph
 - We have a top that represents our own sales agency – the start/end top
 - Each vertex is one customer that is visited exactly once
 - Brid, that is, the path between two customers, can be used an arbitrary number of times
- We notice that for the solution of this problem it is necessary to find the most optimal Hamiltonian cycle in the graph
 - Suppose that we have several Hamiltonian cycles in the graph
 - It is necessary to find the one whose sum of weights of all edges is the smallest

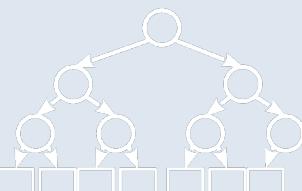
$$\text{!}^{\#} = \arg \min_{(\$)} \mathcal{Z} (\)$$



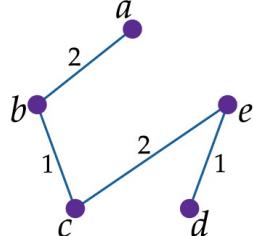
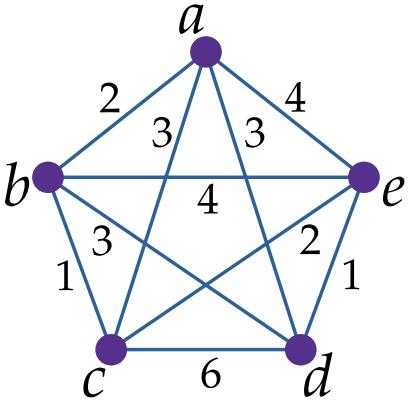
2-MST Heuristics

- 1: Construct the minimal spanning tree G_{MST} from the graph G .
- 2: Eulerize the minimal spanning tree G_{MST} by doubling all edges.
- 3: Find the Eulerian circuit C_E in Eulerized G_{MST} .
- 4: Turn the Eulerian circuit C_E into a Hamiltonian cycle C_H by skipping already visited vertices and jumping to the first unvisited.

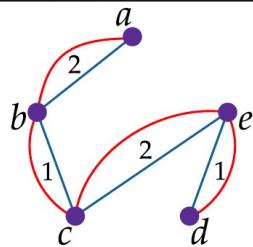
- A method of solving the problem of the traveling salesman
1. Using an adequate algorithm, let's create a minimum spanning tree
 2. We Eulerize the minimum spanning tree
 3. We find the Euler circle using the given start/end vertex
 4. We find Hamilton's cycle from Euler's circle
 - Let's say with DFS we go through min. raz. tree, and at the moment when we return to the already visited peak, we jump to the first unvisited one
 - We remove double vertices from the Euler circle, and we take the edges between new pairs of vertices from the original graph



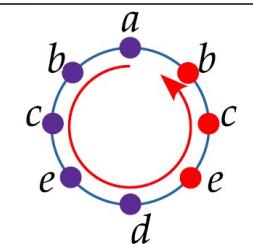
2-MST Heuristics



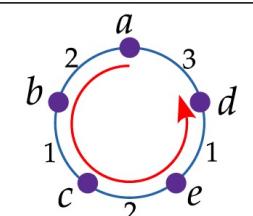
We start by constructing the minimal spanning tree G_{MST} from the graph G . For this purpose we can use any of the MST algorithms.



Eulerize the minimal spanning tree by doubling all edges.



Use any of the Eulerian circuit finding algorithms. Respect the starting vertex of the salesman. Notice that some of the vertices (marked red) are repeating in the Eulerian circuit. These vertices are not according to the definition of the Hamiltonian cycle.



Step 4: Convert the Eulerian circuit C_E to a simple cycle.

Remove all repeating vertices in the Eulerian circuit C_E . Connect the first two non-repeating vertices by using their edge (we need its weight) from the input graph G . In this example we removed repeating vertices e , c , and b , and connected d and a directly using its original edge of weight 3.

Flow in networks

- Networks can be modeled as graphs, where:
 - Vertices represent places, intersections, stations
 - Edges represent connecting lines, such as pipes, tapes, wires, i.e. anything that supports the flow of materials, items, symbols
- Problem: let's try to calculate the maximum possible flow of vehicles and road load from Karlovac to Split
 - Let's say that we define the capacity of the road by the number of vehicles per unit of time that the road passes through
- Similar problems are:
 - Water network, electric power network (and other distribution networks), communication-computer networks, evacuation plans
- The flow can be:
 - Continuous – Let's say fluids (gas, liquid) or solids (%), electricity () and similar
 - Discrete – There is always a number of some items in a unit of time, say the number of vehicles, packages, people or similar
- In the network we always have vertices that generate flow (sources -*sources*) and vertices that consume that flow (destination -*destinations*)



Flow in networks

- Network with flow (*flow network*) is a connected directed graph without loops and return edges

$$= (V, E, c, f)$$

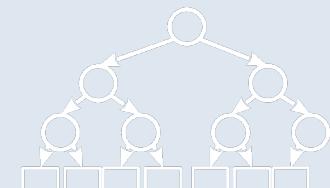
- where c is capacity mapping function to network edges, f is the flow mapping function to the network edges, $v \in V$ is the vertex that generates the flow, $a \in V$ is the vertex that consumes the stream
- When we say that there are no return edges, it is considered

$$\forall v \in V \quad \nexists u \in V$$

- The capacity mapping function is

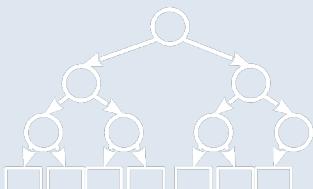
$$c : E \rightarrow \mathbb{R}$$

- Every edge of the network has an indicated positive capacity, which represents the maximum possible flow through that edge



Flow in networks

- The flow mapping function is
$$f : \mathbb{R}^+ \rightarrow \mathbb{R}$$
 - is used to map the positive flux value to the edge of the network.
- Auxiliary edge mapping that will be useful when working out the flow in networks
$$\delta_e : \mathbb{R}^+ \rightarrow \mathbb{R}^+$$



Flow in networks

- Flow has the following properties:

- **Capacity limitation:**

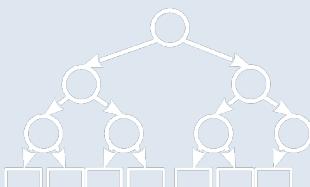
$$\forall e \in E : 0 \leq f(e) \leq c(e)$$

- The maximum flow in an edge of the network is equal to the capacity of that edge

- **Flow conservation:**

$$\forall v \in V \setminus \{s, t\} \quad \sum_{e \in \delta^+(v)} f(e) = \sum_{e \in \delta^-(v)}$$

- For each vertex that does not generate or consume a flow, the input flow is equal to the output flow
 - This balance does not hold for vertices that generate and consume streams
 - It is obvious that for a peak to generate flux, the output flux must be greater than the input
 - For a peak to consume flow, the input flow must be greater than the output flow

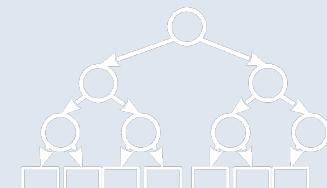


Flow in networks

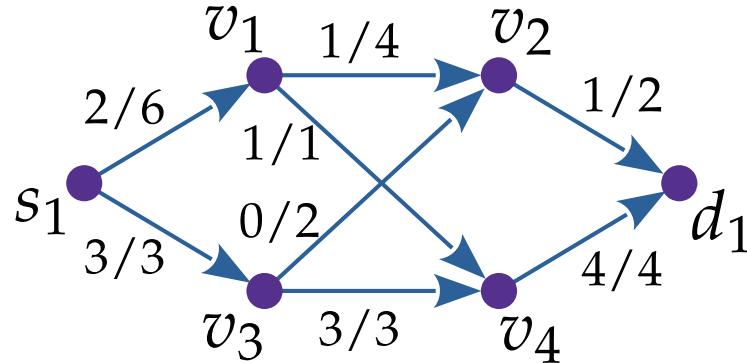
- **Current strength** is defined as

$$| = | @ () - @ (*) \\ + , ! \in . (+, 1) , "+ \in . (1, +)$$

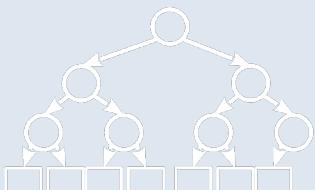
- where $|$ is top flow generator
- As already emphasized, for a peak to generate flow, the output flow must be greater than the input, and the difference itself represents the strength of the flow
 - To keep things simple, only one vertex that generates the flow will be used \in and one vertex that consumes the flow \in
 - It is obvious that if we have multiple flux generators, then each flux generating peak contributes to the total flux strength $| + |$



Flow in networks



- In the network example we have
 - top & which generates the flow
 - top & which consumes the flow
 - each edge has a label
 $()/()$
- Is the flow through the network the maximum possible?

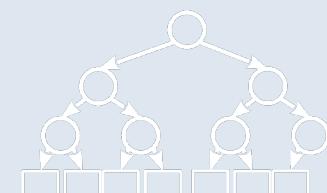


Ford-Fulkerson algorithm

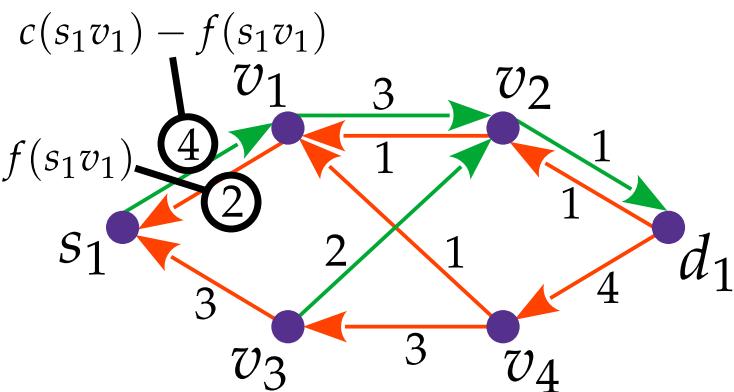
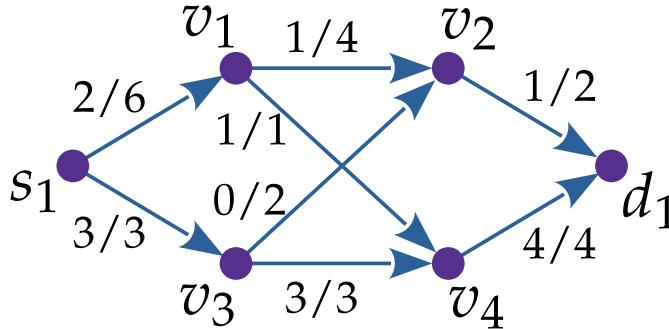
- **Other network:** Given the network G , the remaining one can be constructed mesh (*residual network*)

$$\beta = (, \ 3, \ 3, \ 3, \ ,)$$

- where are they β the edges of the remaining mesh are not the same as the edges of the original mesh
- β is the function of mapping the residual capacity to the edges of the residual network
- β is a function of mapping the residual flow to the edges of the residual network
- The edges of the remaining network can be:
 - Advanced (*forward*): $\beta = \{ : \in , - > 0 \} \cup \{ \}$
 - Returnable (*reverse*): $\beta = \{ : \in , > 0 \} \cup \{ \}$
 - So it holds: $\beta = \beta \cup \beta_{\text{reverse}}$



Ford-Fulkerson algorithm



- **Last capacity** (*residual capacity*) on the rest of the network \exists is defined by the mapping function

$$\begin{aligned} c(s_1v_1) - f(s_1v_1) & \quad (-) , \exists (\in) - > 0 \quad () \quad () \\ \exists \quad (= H) & \quad () \quad \exists \in , \quad \geq (0) \\ 0, & \quad h \end{aligned}$$

- For advanced edges, the final capacity is the difference between capacity and flow in the original network
- For return edges, it is valid that the last capacity is equal to the current of the opposite edge in the original network

Ford-Fulkerson algorithm

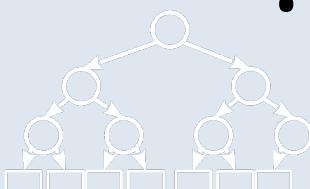
- Similar to the original network ,the rest of the network β can have streams, defined by a mapping function β
- We define the term for flows in the remaining network **increase flow** (*flow augmentation*) as

$$\uparrow \beta = (C) \quad (\beta + \beta(C, \beta)) \in (0, h)$$

- where the flow in the original network increases by the current in the remaining network β
- β is also called **increasing flow** (*increasing flow*)
- The same is true for **increase in flow strength** (*flow strength augmentation*)

$$|\uparrow \beta| = |\beta|$$

- check the proof in the script



Ford-Fulkerson algorithm

- **Enlarging the road** (*augmenting path*): Given the network and the rest of her network G , any time from the top to the top in G it is called the augmenting path

- Each such path gives us the possibility to define an increasing flow & on the rest of the network G , and thereby increasing the flow in the original network \uparrow &
- The remaining capacity of the increasing path is

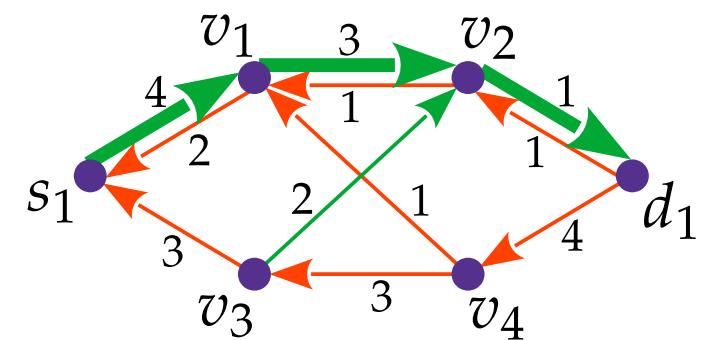
$$\Delta \in \min_{\forall i \in E} \Delta(i)$$

- thereby increasing the flow *on the way limits to the minimum remaining capacity on that path
- Increasing flow on the road or Δ is counted as

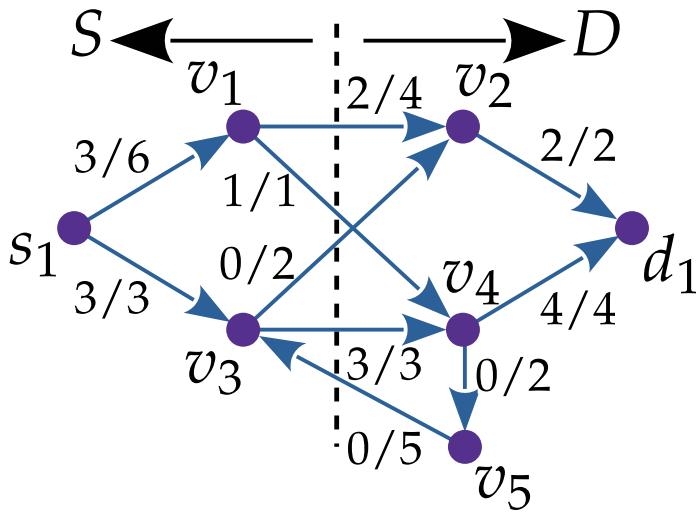
$$\forall e \in E : \Delta_e = \Delta(e)$$

- Increasing the flux in the original network results

$$| \uparrow \Delta | = | + \Delta | - | f |$$

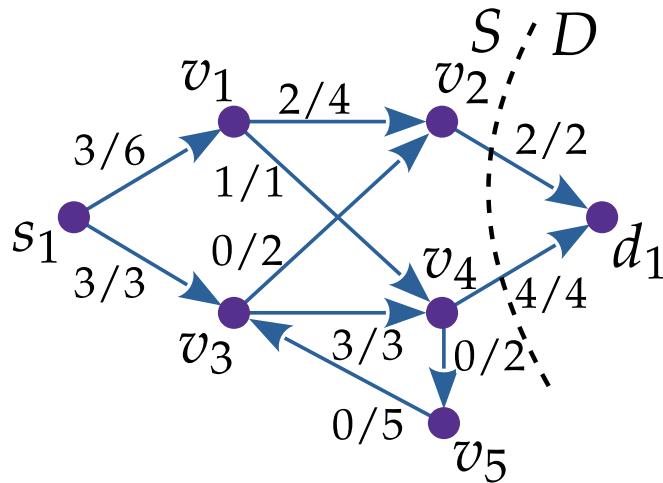


Cross section of the network



- We can partition any network in such a way as to divide the vertices of the network into two disjoint partitions $C_1, C_2 \subset V$
 - so it's worth it $\in C_1, \in C_2, C_1 \cap C_2 = \emptyset$ and $C_1 \cup C_2 = V$
 - With this, we have defined the cross-section of the network, through which the edges pass (C_1, C_2) .
 - The network cross section capacity is calculated
- $$C(C_1, C_2) = \min_{e \in E} \{c_e : e \text{ passes through } (C_1, C_2)\}$$
- The cross-sectional flow of the network is calculated
- $$f(C_1, C_2) = \max_{\pi} \{ f_\pi : \pi \text{ is a flow from } C_1 \text{ to } C_2\}$$
- Due to the flow conservation property, it is valid $|f| = C(C_1, C_2)$ (proof see in scripts)

Minimum network section (min-cut)



- Cross section of the network , which has the smallest capacity (,) is called the minimum section(*minimal cut*)and is obtained by optimization

$$4, (= m) n \left(\begin{smallmatrix} 5 \\ 6 \end{smallmatrix} \right)$$

- **Corollary:**Capacity limitation(*network cut capacity constraint*)can be applied to any cross section of the network , ,what is it worth

$$0 \leq |(\)| \leq (,)$$

- **Evidence:**

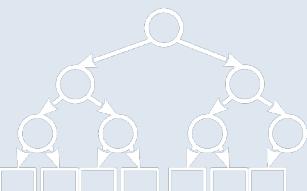
$$|(\)| = 3 - 3 \quad . \in + (5,6) \quad . \in + 6.5$$

$$\leq 3 \leq 3 = (,)$$

$. \in + 5(6) \quad . \in + 5(6)$

Max-flow min-cut theorem

- Given the network which has a flow ,flow generator ,flow consumer , the following statements apply:
 1. is the maximum flow in the network ,
 2. in the remaining network 3no more augmenting paths,
 3. there is a cross section of the network , such that it is $\text{valid} = (,)$
- Proof (\rightarrow):**Assume that the 2nd does not hold, that is, there is an increasing path in the rest of the network 3. This means that there is a flow $3\#$ in the rest of the network 3which can increase the flow in the original network .
That's how we get $\uparrow \beta > | |$,which is in direct contradiction with 1.



Max-flow min-cut theorem

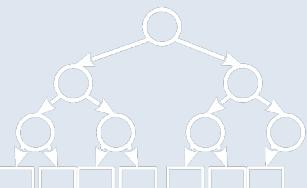
- **Proof (→):** If there is no augmenting path in the rest of the network β , then there is no path between the vertices at all and , which means there is an intersection , for which there is not a single edge between the vertices and in the rest of the network β , which means it is valid

$$\forall \in \beta, (\nexists \notin \beta \Rightarrow \beta = 0, \beta = \emptyset) \quad (1)$$

Given the previous corollary, for the section , it's worth it

$$|\beta| = |\gamma|, \beta \neq \gamma$$

which is identical to statement 3.



Max-flow min-cut theorem

- **Proof (\rightarrow):** A corollary is a direct proof, that is

$$| \ | \leq (,)$$

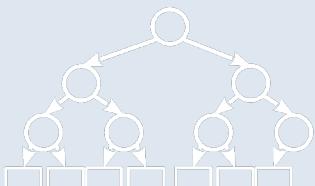
and it is valid that the section at which it is valid

$$|=| (,)$$

maximum flow in the network

- Due to this theorem, the maximum flow is equal to the minimum cross-section of the network

$$|=| 4(,)$$



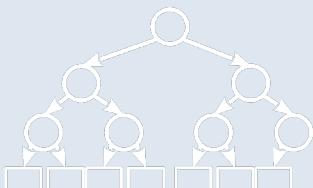
Ford-Fulkerson algorithm

```
procedure FORDFULKERSON(G)
  for each edge  $uv$  in  $E(G)$  do
     $f(uv) \leftarrow 0$ 
    calculate the residual network  $G_f$                                 ▷ Def. 4.44
    while there is an augmenting path  $p$  in  $G_f$  do
      calculate  $f_{f_p}$                                               ▷ (4.99)
      perform the flow augmentation ( $f \uparrow f_{f_p}$ )                ▷ (4.93)
      calculate the residual network  $G_f$                                 ▷ Def. 4.44
```

- The complexity of this algorithm is ("+ ")
- Where is "the number of increasing paths

- We initialize the basic network setting the flows to 0
- Let's calculate the rest of the network 3
 - Let's find some augmenting path in the rest of the network 3
 - We calculate the increasing flow 3"
 - We increase the flow in the basic network
 - We are updating the rest of the network 3
 - We repeat as long as there is a path in between and in the rest of the network

3

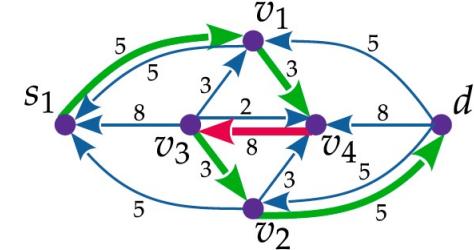
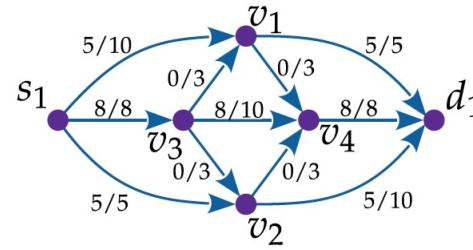


Ford-Fulkerson algorithm

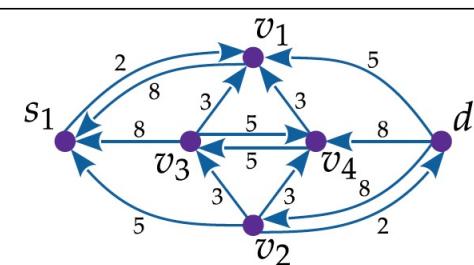
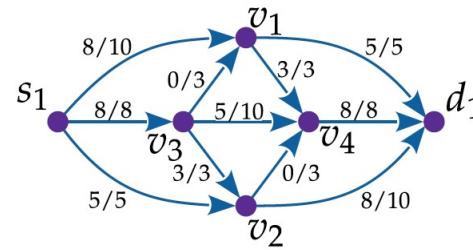
Step	G	G_f
1: The initial residual network has the residual capacities equal to the original flow network capacities. We choose the augmenting path $p = s_1v_1d_1$, which has the minimal residual capacity $c_f(p) = 5$. The augmenting path flow strength is therefore $ f_{fp} = 5$.		
2: We choose the augmenting path $p = s_1v_3v_4d_1$, which has the minimal residual capacity $c_f(p) = 8$. The augmenting path flow strength is therefore $ f_{fp} = 8$.		
3: We choose the augmenting path $p = s_1v_2d_1$, which has the minimal residual capacity $c_f(p) = 5$. The augmenting path flow strength is therefore $ f_{fp} = 5$.		

Ford-Fulkerson algorithm

4: We choose the augmenting path $p = s_1v_1v_4v_3v_2d_1$, which has the minimal residual capacity $c_f(p) = 3$. The augmenting path flow strength is therefore $|f_{f_p}| = 3$. Notice that we used reversed edge v_4v_3 from the residual network G_f .



5: We have no more available augmenting paths. The final maximal flow is $|f| = 21$.



Edmonds-Karp algorithm

- An upgrade of the Ford-Fulkerson algorithm, which selects the augmenting path using the BFS algorithm, so that shorter paths are prioritized with respect to the number of edges.

