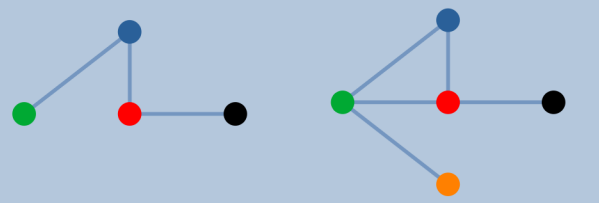
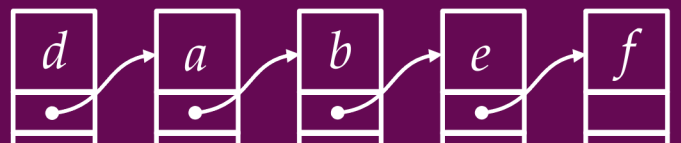
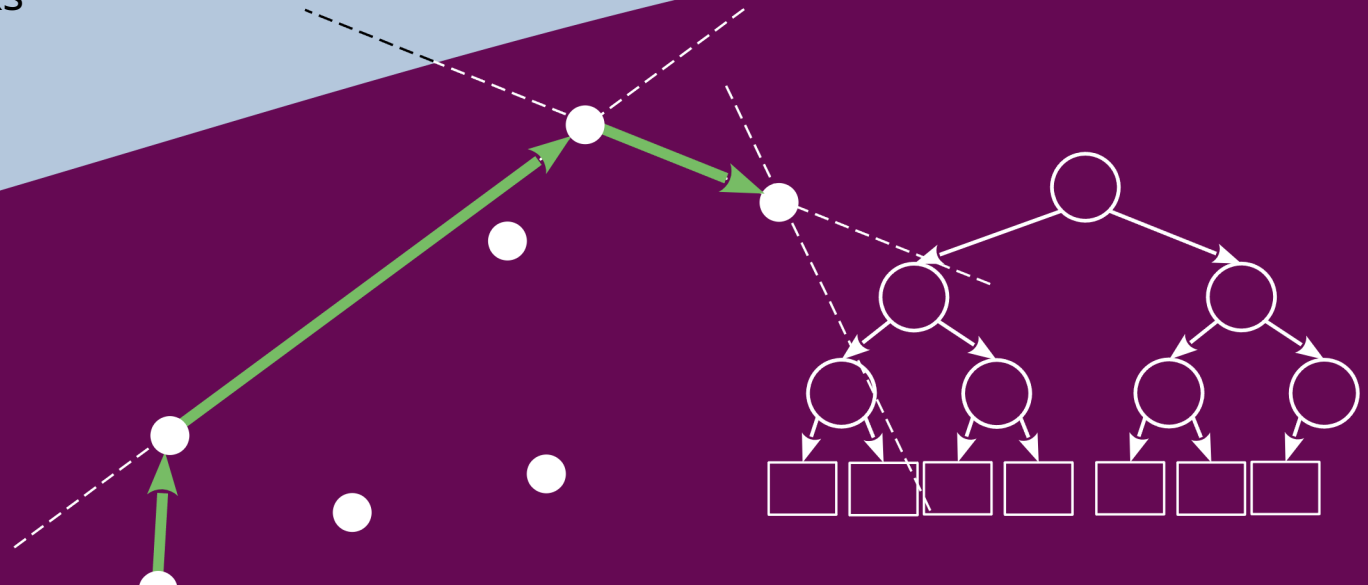
Advanced algorithms and structures data

Week 11: Algorithms over graphs

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



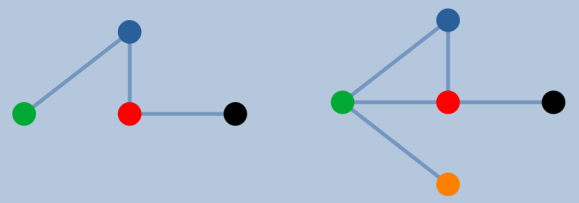


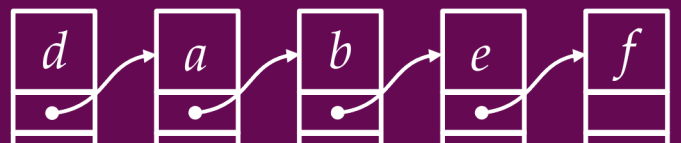


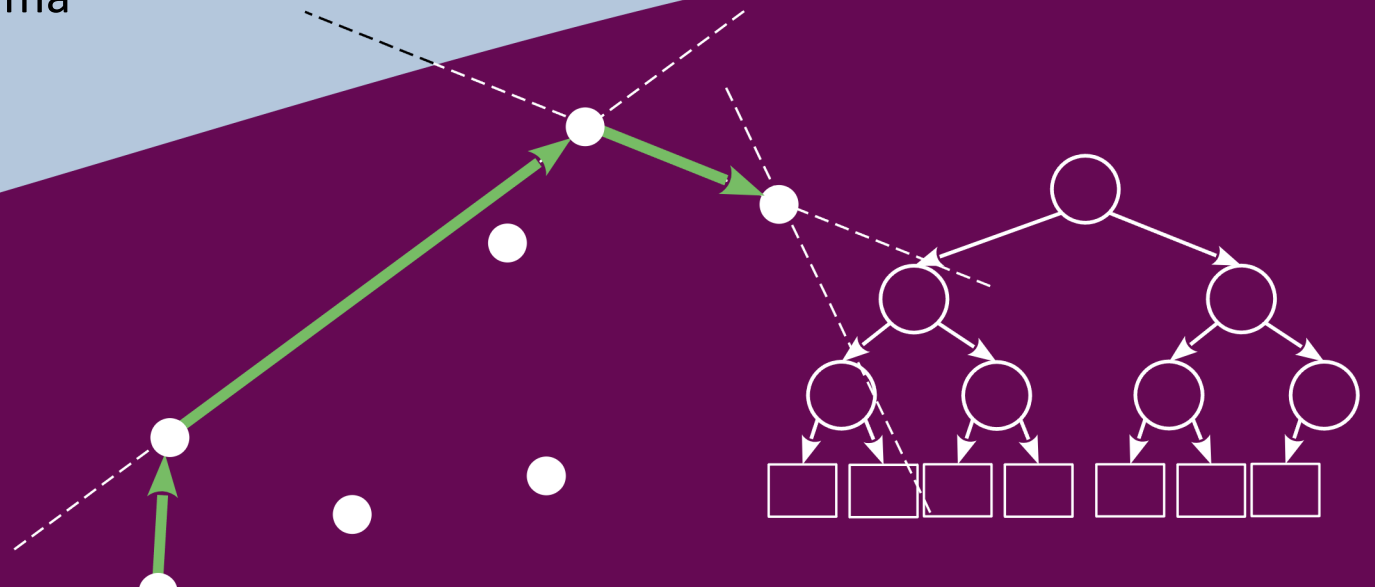
Napredni algoritmi i strukture podataka

Tjedan 11: Algoritmi nad grafovima

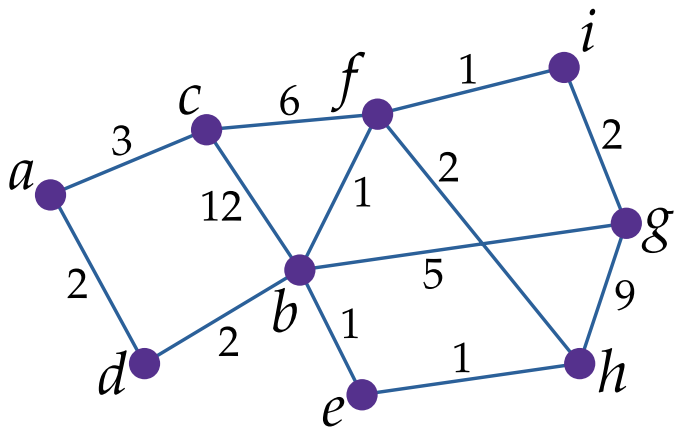
Problem kineskog poštara, Hamiltonovi krugovi, Problem trgovačkog putnika, Protok u mrežama





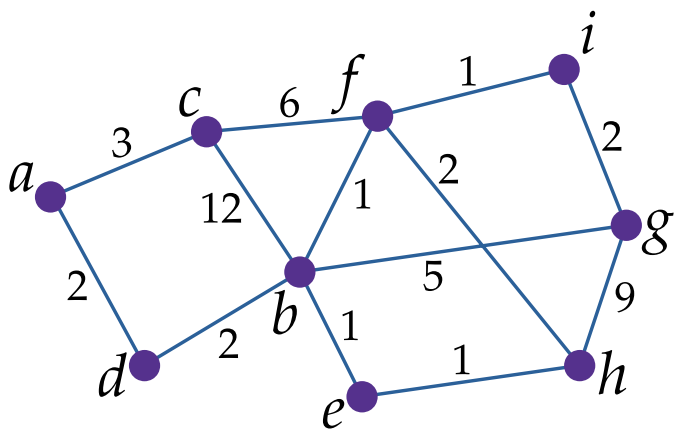


The problem of the Chinese postman

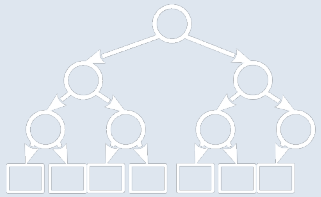


- The Chinese postman problem is similar to the seven bridges problem
- A Chinese postman needs to go around a village modeled by a weight graph in the most efficient way possible
  - The edges represent the streets of the settlement, and the weights on the edges represent the cost (effort, *effort*) 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

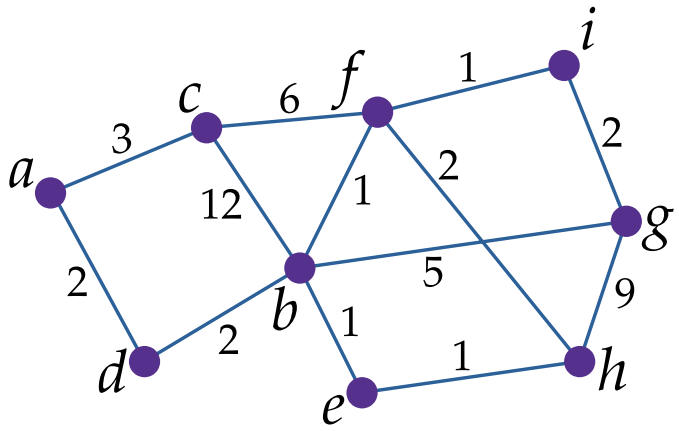
Problem kineskog poštara



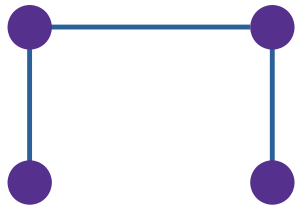
- Problem kineskog poštara sličan je problemu sedam mostova
- Kineski poštar treba obići naselje modelirano težinskim grafom na najefikasniji mogući način
  - Bridovi predstavljaju ulice naselja, a težine na bridovima trošak (trud, *effort*) koji poštar mora uložiti za prelazak te ulice
  - Nakon obilaska naselja, poštar se vraća u vrh iz kojeg je krenuo
- Ako je naselje modelirano Eulerovim grafom koji sadrži Eulerov krug, tada je stvar jasna – Uvijek postoji točno jedan obilazaka
- Ako naselje nije modelirano Eulerovim grafom, tada imamo optimizacijski problem koji treba riješiti

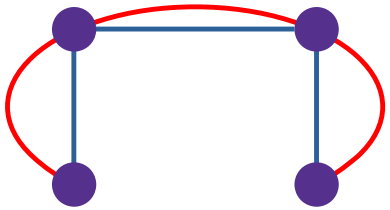


The problem of the Chinese postman

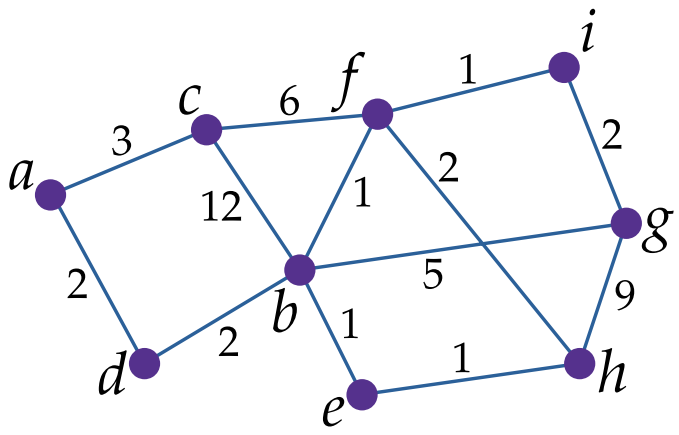


- We have already established that an Euler graph containing an Euler circle must have all vertices of even degree
- If we have some of the vertices that have an odd degree, then we need a settlement graph *Eulerize*, in order to obtain the Euler circle
- The concept of Eulerization means adding edges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

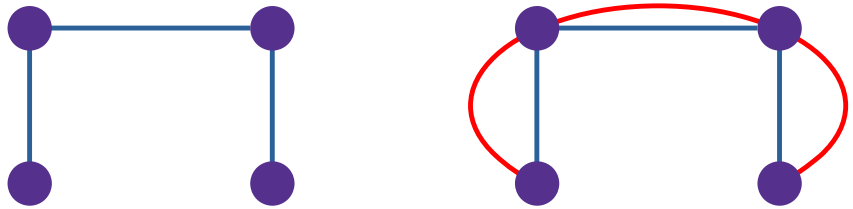




Problem kineskog poštara



- Već smo utvrdili da Eulerov graf koji sadrži Eulerov krug mora imati sve vrhove parnog stupnja
- Ako imamo neki od vrhova koji imaju neparan stupanj, tada graf naselja moramo *Eulerizirati*, kako bismo dobili Eulerov krug
- Koncept Eulerizacije znači dodavanje bridova u graf tako da svi vrhovi grafa imaju paran stupanj



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

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

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.

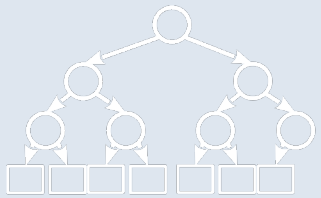
4: In the complete bipartite graph H find the optimal assignments M between vertices in X and Y, such that

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

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 $E(G)$



Eulerizacija grafa

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

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

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.

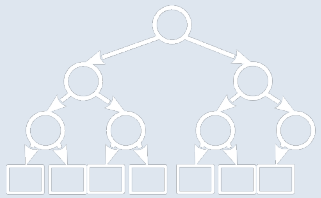
4: In the complete bipartite graph H find the optimal assignments M between vertices in X and Y, such that

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

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 $E(G)$

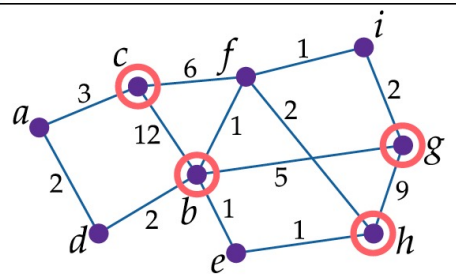


Eulerization – step 1 and 2

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

$$w(bc) = D^9_{bc} = 7, w(bg) = D^9_{bg} = 4$$
$$w(bh) = D^9_{bh} = 2, w(cg) = D^9_{cg} = 9$$
$$w(ch) = D^9_{ch} = 8, w(gh) = D^9_{gh} = 5$$

and shortest paths

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

$$ODDPaths(b, g) = bfig$$
$$ODDPaths(b, h) = beh$$
$$ODDPaths(c, g) = cfig$$
$$ODDPaths(c, h) = cfh$$
$$ODDPaths(g, h) = gifh$$

$\mathbf{D^0} =$

	a	b	c	d	e	f	g	h	i
a	0	0	3	2	0	0	0	0	0
b	0	0	12	2	1	1	5	0	0
c	3	12	0	0	0	6	0	0	0
d	2	2	0	0	0	0	0	0	0
e	0	1	0	0	0	0	0	1	0
f	0	1	6	0	0	0	0	2	1
g	0	5	0	0	0	0	0	9	2
h	0	0	0	0	1	2	9	0	0
i	0	0	0	0	0	1	2	0	0

$\mathbf{D^9} =$

	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

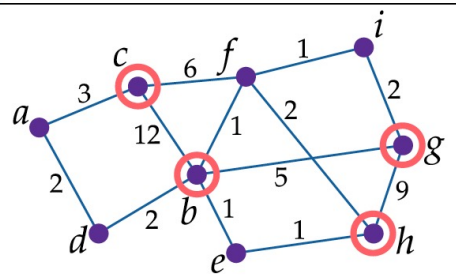
- In the first step, we find all vertices with an odd degree
- In the second step, we calculate the shortest all-all distances within the set
  - In addition, we determine the shortest paths
  - WFI was used in the example, there are other options

Eulerizacija – korak 1 i 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

$$w(bc) = D^9_{bc} = 7, w(bg) = D^9_{bg} = 4$$
$$w(bh) = D^9_{bh} = 2, w(cg) = D^9_{cg} = 9$$
$$w(ch) = D^9_{ch} = 8, w(gh) = D^9_{gh} = 5$$

and shortest paths

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

$$ODDPaths(b, g) = bfig$$
$$ODDPaths(b, h) = beh$$
$$ODDPaths(c, g) = cfig$$
$$ODDPaths(c, h) = cfh$$
$$ODDPaths(g, h) = gifh$$

$\mathbf{D^0} =$

	a	b	c	d	e	f	g	h	i
a	0	0	3	2	0	0	0	0	0
b	0	0	12	2	1	1	5	0	0
c	3	12	0	0	0	6	0	0	0
d	2	2	0	0	0	0	0	0	0
e	0	1	0	0	0	0	0	1	0
f	0	1	6	0	0	0	0	2	1
g	0	5	0	0	0	0	0	9	2
h	0	0	0	0	1	2	9	0	0
i	0	0	0	0	0	1	2	0	0

$\mathbf{D^9} =$

	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

- U prvom koraku pronađemo sve vrhove s neparnim stupnjem
- U drugom koraku izračunavamo najkraće udaljenosti svi-svi unutar skupa ODD
  - Uz to određujemo te najkraće putanje
  - U primjeru se koristio WFI, ima i drugih opcija

Eulerization – steps 3, 4 and 5

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

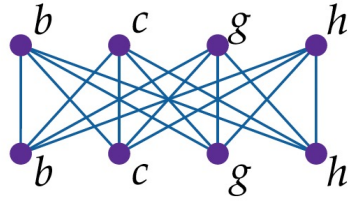
In this step we need to optimize the pairs of vertices of the complete bipartite graph H.

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

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

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

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



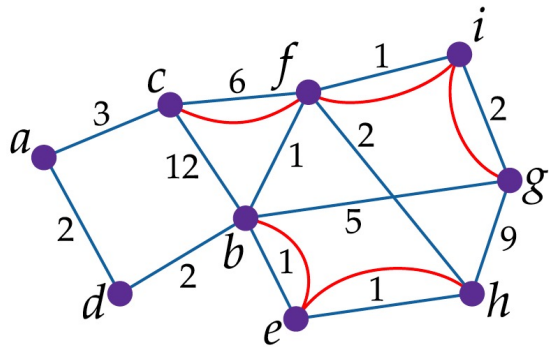
b			c			g			h		
c	g	h	b	g	h	b	c	h	b	c	g
7	4	2	7	9	8	4	9	5	2	8	5

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

Eulerizacija – koraci 3,4 i 5

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

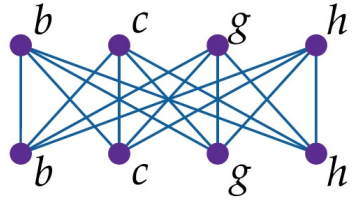
In this step we need to optimize the pairs of vertices of the complete bipartite graph H.

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

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

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

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



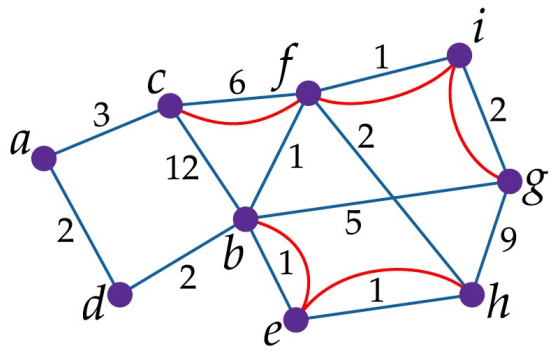
b			c			g			h		
c	g	h	b	g	h	b	c	h	b	c	g
7	4	2	7	9	8	4	9	5	2	8	5

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.

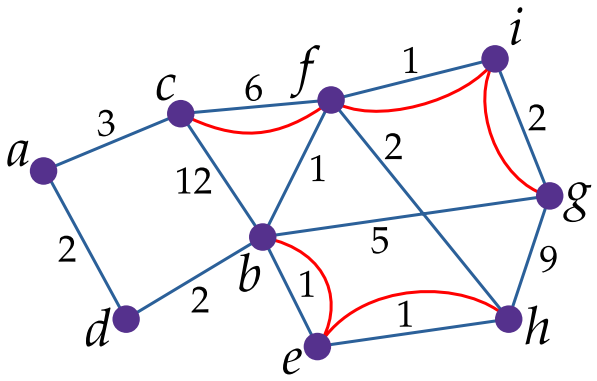
1. $(b,h) \in M_3 : ODDPaths(b,h) = beh$. Edges be and eh are added to the graph G.
2. $(c,g) \in M_3 : ODDPaths(c,g) = cfig$. Edges cf, fi, and ig are added to the graph G.

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



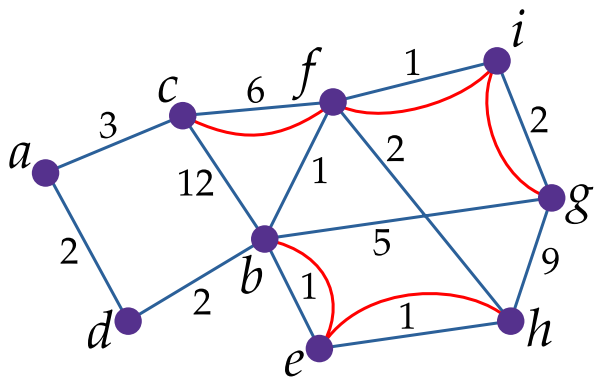
- Konstruiramo bipartitni graf iz kojeg izvlačimo optimalne parove, koji imaju minimalnu sumu težina putanja
  - Optimalni put poštara
  - Može se rješavati kroz tablicu
- Na kraju iz $ODDPaths$ uzmemo putanje za optimalne parove
  - Dodajemo paralelne bridove po putanjama
  - Početni vrh u putanji dobije samo jedan prileježeći brid – čime mu stupanj postaje paran
  - Međuvrhovi u putanji dobiju dva prileježeća brida, čime im se ne mijenja parnost/neparnost trenutnog stupnja

The problem of the Chinese postman



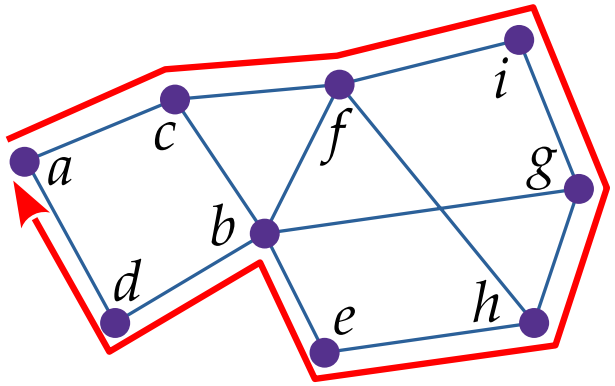
- After Eulerization, let's apply one of the algorithms for detecting the Euler circle to the graph
  - Visually, Fleury's algorithm is simpler
- An Euler circle is found that starts and ends at the peak from which the Chinese postman starts his tour
- The total sum of the weights on all edges of the 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.

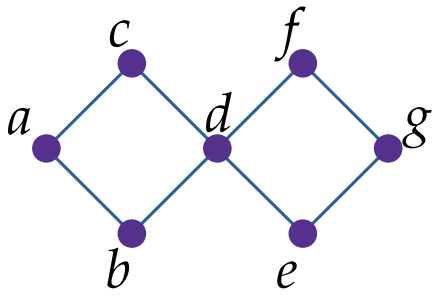
Problem kineskog poštara

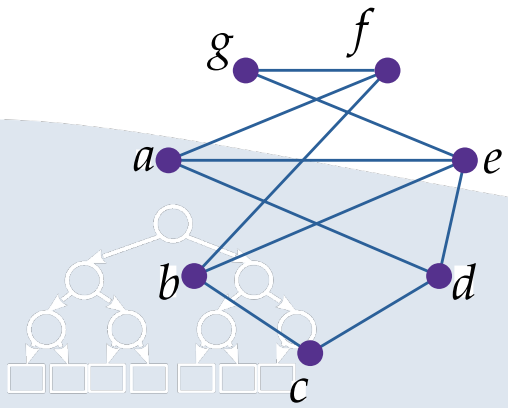


- Nakon Eulerizacije na graf primijenimo neki od algoritama za detekciju Eulerovog kruga
  - Vizualno je Fleuryev algoritam jednostavniji
- Pronađe se Eulerov krug koji počinje i završava u vrhu iz kojeg kineski poštar kreće u obilazak
- Ukupni zbroj težina po svim bridovima Eulerovog kruga je ukupni trošak obilaska kineskog poštara
  - Svi nadodani bridovi u postupku Eulerizacije imaju istu težinu kao i originalni brid između dva vrha – radi se naime o jednoj te istoj ulici
  - Na primjer, originalni brid i nadodani brid cf su oba težine 6.

Hamiltonian graphs and cycles

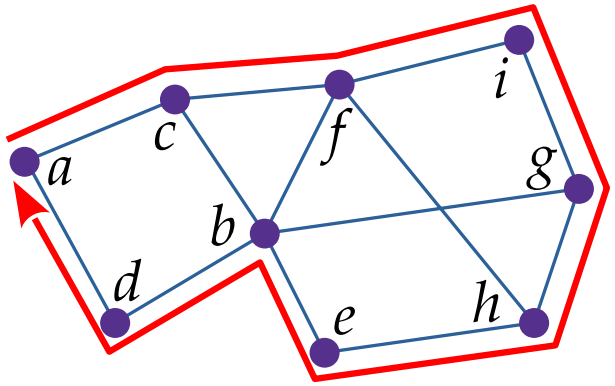


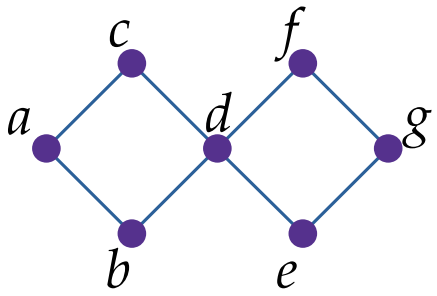


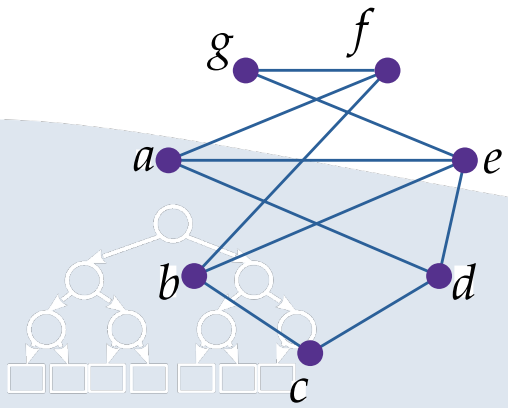


- **Hamilton's way**(*path*) – is a path through the graph that includes all vertices of the graph only once
- **Hamilton cycle**(*cycle*) – is a Hamilton path that starts and ends at the same vertex
- **Hamilton's graph**–is a graph made of a Hamiltonian cycle
- It is important to note that unlike the 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

Hamiltonovi grafovi i ciklusi







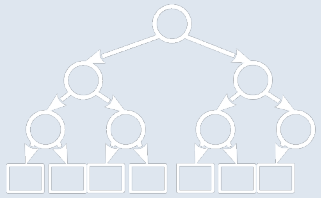
- **Hamiltonov put** (*path*) – je put kroz graf koji uključuje sve vrhove grafa samo jednom
- **Hamiltonov ciklus** (*cycle*) – je Hamiltonov put koji počinje i završava u istom vrhu
- **Hamiltonov graf** – je graf koji je sačinjen od Hamiltonovog ciklusa
- Bitno je uočiti da za razliku od Eulerovog kruga, u Hamiltonovom ciklusu kroz svaki vrh možemo proći samo jednom i taj ciklus mora uključivati sve vrhove grafa – Hamiltonovog grafa
- Problem je NP-kompleksan

Graph closure

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

$G' \leftarrow G$
$i \leftarrow 1$
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 (or deg + deg( ) ≥ | |
  - 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 '

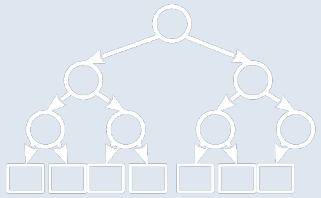


Zatvarač grafa

- Ako imamo graf G, tada je G' zatvarač (*closure*) grafa G koji se dobije

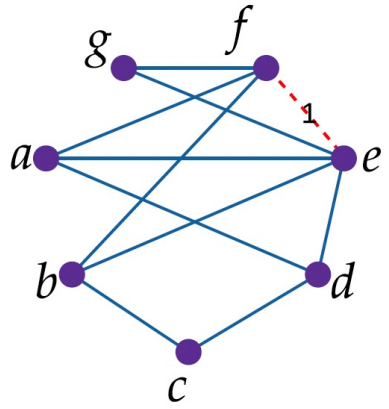
$G' \leftarrow G$
$i \leftarrow 1$
while there are vertices u and v, such that $deg(u) + deg(v) \geq |V(G)|$ **do**
    add the edge uv to the edges $E(G')$, mark it as the i-th edge
    $i \leftarrow i + 1$

- Dodajemo brid na neki par vrhova uv, čiji je zbroj stupnjeva veći ili jednak broju vrhova u grafu ili $\deg(u) + \deg(v) \geq |V|$
  - Dodani brid označimo rednim brojem
- U trenutku kada više ne možemo dodati niti jedan brid, to se smatra zatvaračem G'

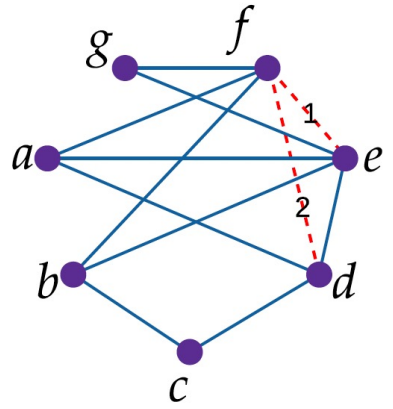


Graph closure

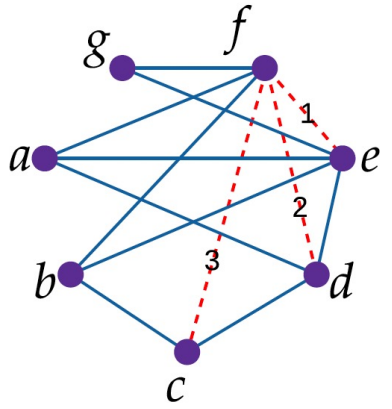
Iteration 1: $deg(f) + deg(e) = 7$



Iteration 2: $deg(f) + deg(d) = 7$

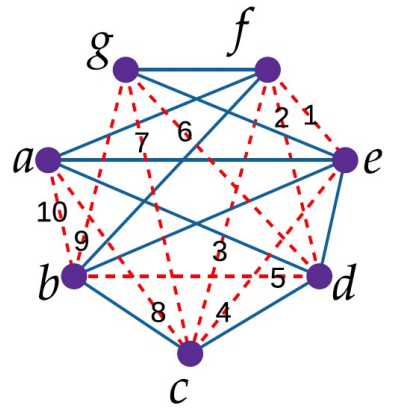


Iteration 3: $deg(f) + deg(c) = 7$

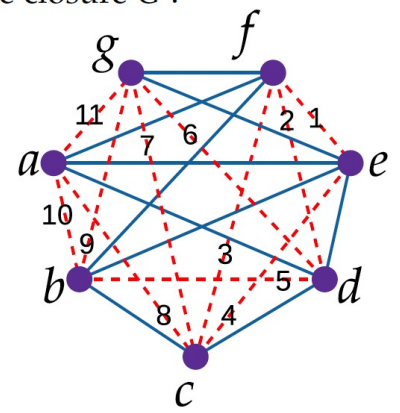


We continue to add edges for pairs of vertices whose degrees comply the requirement $deg(u) + deg(v) \geq |V|$.

Iteration 10: $deg(a) + deg(b) = 9$

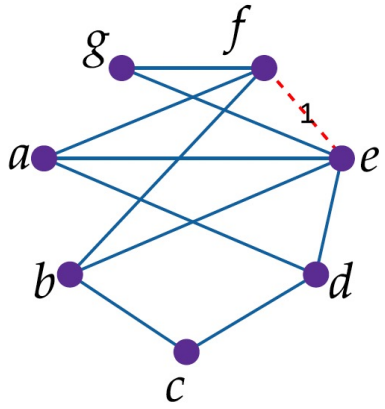


Iteration 11: $deg(a) + deg(g) = 10$. This is also the last step and the closure G'.

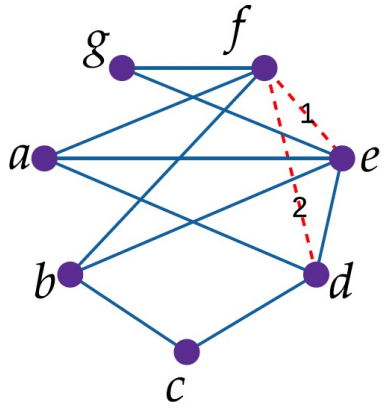


Zatvarač grafa

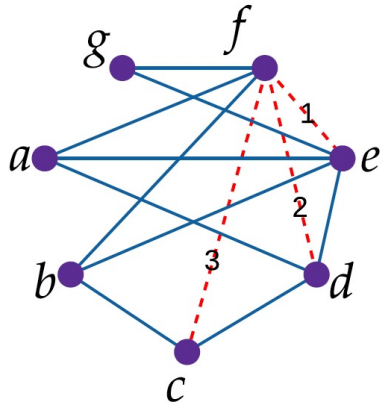
Iteration 1: $deg(f) + deg(e) = 7$



Iteration 2: $deg(f) + deg(d) = 7$

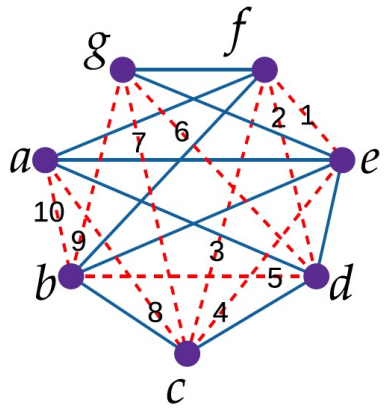


Iteration 3: $deg(f) + deg(c) = 7$

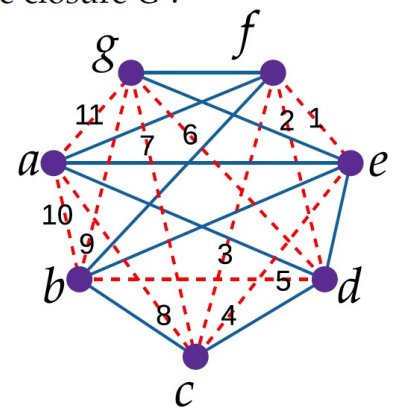


We continue to add edges for pairs of vertices whose degrees comply the requirement $deg(u) + deg(v) \geq |V|$.

Iteration 10: $deg(a) + deg(b) = 9$



Iteration 11: $deg(a) + deg(g) = 10$. This is also the last step and the closure G'.

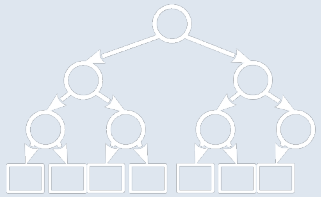


Bondy–Chvátal theorem

- **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 ′ ⊆ ( )such that it forms a Hamiltonian cycle !. Shutter 'it is created by adding edges to the graph ,what is it worth ′ ⊆ ( ) ⊆ ("), which means that it is !also contained in the closure '.
- **Evidence 2**: Between graphs and its shutter 'we have a sequence of graphs

  #, $, %,..., &'$, &

  - where are they = !and "= #
  - and each graph $has one more edge compared to the graph $%&

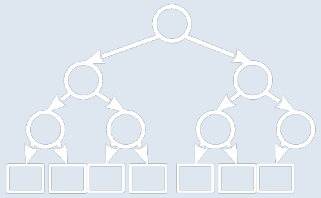


Bondy–Chvátal teorem

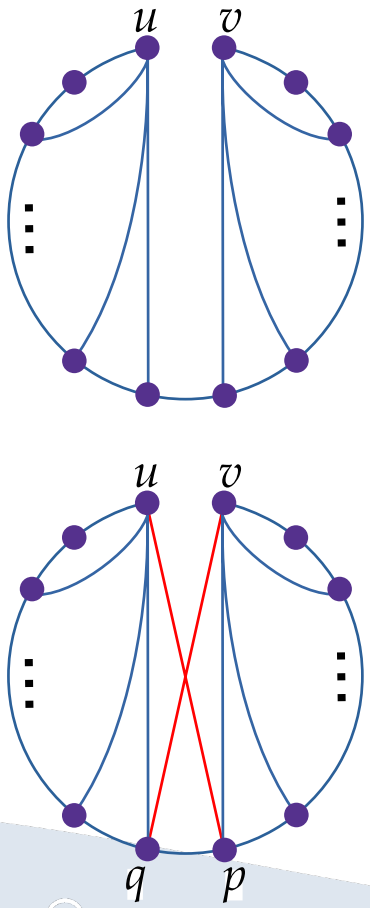
- **Teorem**: Jednostavan neusmjereni graf G je Hamiltonov samo i samo ako je njegov zatvarač G' Hamiltonov graf.
- **Dokaz 1**: Ako je jednostavan neusmjereni graf G Hamiltonov, tada postoji podskup bridova $E' \subseteq E(G)$ takav da čini Hamiltonov ciklus C_H. Zatvarač G' stvoren je dodavanjem bridova na graf G, čime vrijedi $E' \subseteq E(G) \subseteq E(G')$, što znači da je C_H sadržan i u zatvaraču G'.
- **Dokaz 2**: Između grafa G i njegovog zatvarača G' imamo slijed grafova

$$G_0, G_1, G_2, \dots, G_{k-1}, G_k$$

  - gdje su $G = G_0$ i $G' = G_k$
  - i svaki graf G_i ima jedan brid više u odnosu na graf G_{i-1}



Bondy–Chvátal theorem

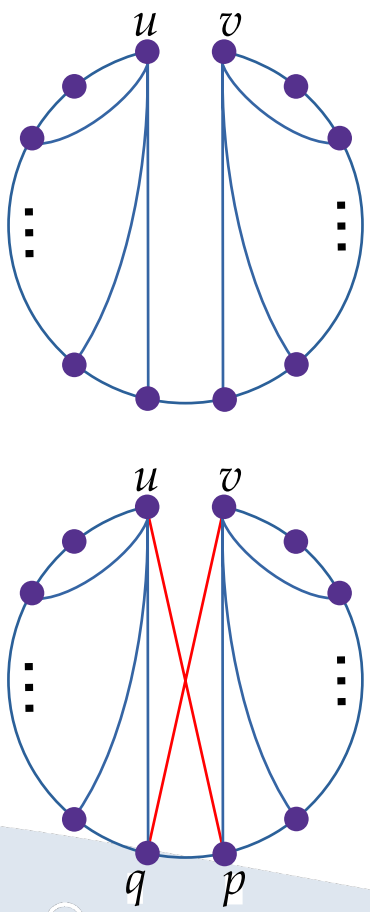


- According to the Bondy–Chvátal theorem, if the graph ! contains a Hamiltonian cycle, then the graph also contains it !"#. This is subject to the principle of mathematical induction.
- In the above picture we have a graph !"#. In order to get the Hamiltonian cycle, we need to add an edge and thereby gain ! which contains it
  - If it is (!"#) = , then it is valid deg + (deg = −(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 + deg) =
  - By adding the edge uv, u ! we have the Hamilton cycle $
  - We notice that in !"# we have a Hamiltonian cycle which is

$− + +

QED

Bondy–Chvátal teorem



- Po Bondy–Chvátal teoremu, ako graf G_i sadrži Hamiltonov ciklus, tada ga sadrži i graf G_{i-1}. Ovo podliježe principu matematičke indukcije.
- Na gornjoj slici imamo graf G_{i-1}. Kako bismo dobili Hamiltonov ciklus, trebamo dodati brid uv i time dobiti G_i koji ga sadrži
  - Ako je $|V(G_{i-1})| = n$, tada vrijedi $\deg(u) + \deg(v) = n - 2$
  - Zbog toga ne možemo dodati brid uv
  - Da bismo dodali brid uv, moramo iz vrhova u i v imati dva dodatna brida, koji su na donjoj slici označeni kao up i vq
  - Time smo dobili $\deg(u) + \deg(v) = n$
  - Dodavanjem brida uv, u G_i imamo Hamiltonov ciklus C_H
  - Uočavamo da u G_{i-1} imamo Hamiltonov ciklus koji je

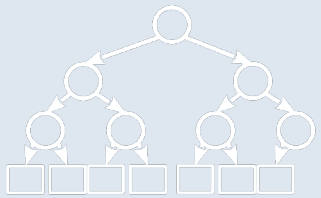
$$C_H - pq + up + vq$$

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

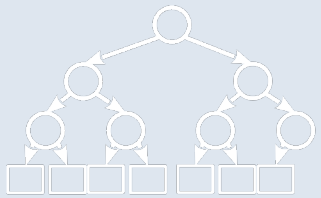
- We are coming back from the shutter 'back
  - We found Hamilton's cycle in the shutter %
  - IN %we take the added edge with the highest ordinal number as an edge –remember that we are in the process of creating a shutter ' marked
  - IN %let's choose an edge ,such that there are crossed edges and 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



Praktična primjena Bondy–Chvátal teorema

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

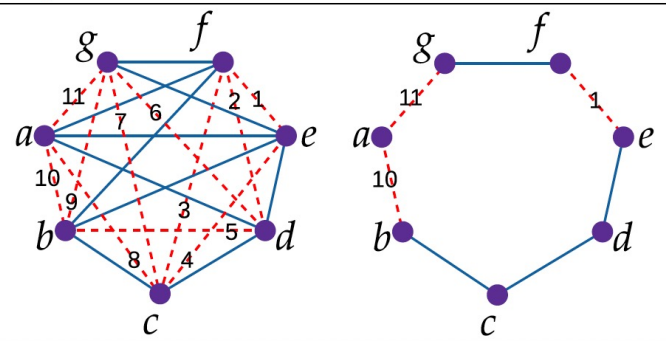
- Vraćamo se od zatvarača G' natrag
  - U zatvaraču smo pronašli Hamiltonov ciklus G_c
  - U G_c uzimamo nadodani brid s najvišim rednim brojem kao brid uv – sjetimo se da smo ih u postupku stvaranja zatvarača G' označavali
  - U G_c odaberemo brid pq, takav da postoje prekriženi bridovi up i vq u G' koji su ili iz grafa G ili imaju manji redni broj od brida uv
  - U G_c uklonimo bridove uv i pq, a dodamo bridove up i vq
  - Ovo radimo sve do dok u G_c imamo nadodanih bridova – kada ih nema, KRAJ, pronašli smo Hamiltonov ciklus u grafu G



Practical application of the Bondy–Chvátal theorem

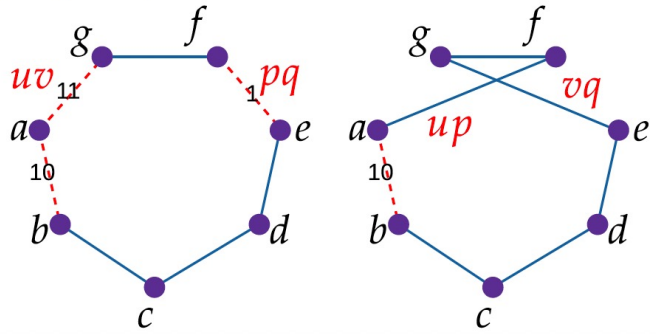
Step 1: Detect the Hamiltonian cycle in the closure G'

We start by detecting the Hamiltonian cycle in the closure G'.



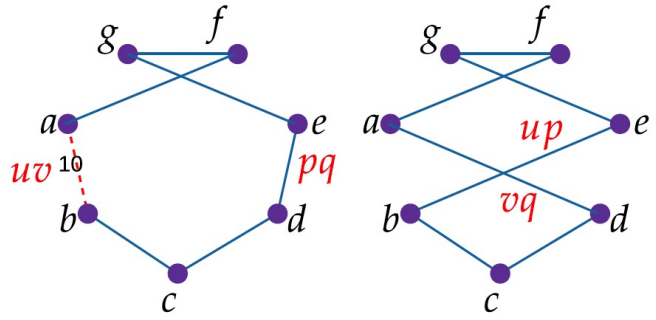
Step 2: Replace edges ag and fe.

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.

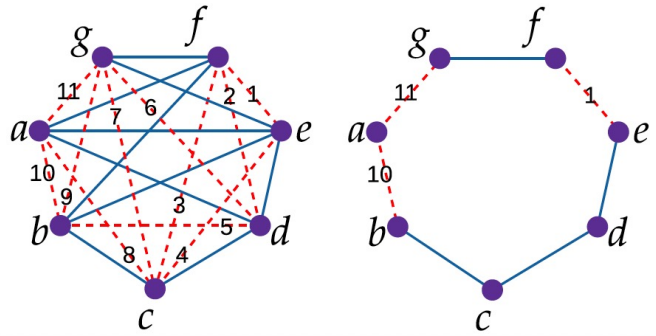
We find ab as the edge having the highest ordinal $i_{max} = 10$, which is identified as the edge uv. The other edge ed is taken as the edge pq. Taking the edge ed is convinient, as we have crossed edges ad and be from the original graph G. After this step, we have no more added edges, which means we moved backwards to the graph $G = G_0$. The result is the Hamiltonian cycle in the original graph G.



Praktična primjena Bondy–Chvátal teorema

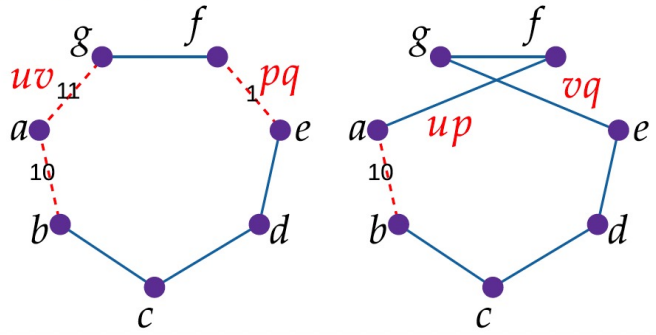
Step 1: Detect the Hamiltonian cycle in the closure G'

We start by detecting the Hamiltonian cycle in the closure G'.



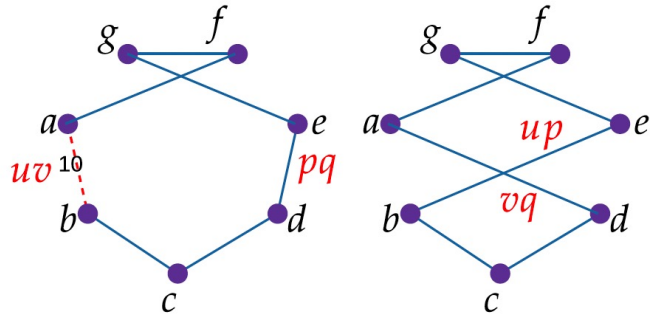
Step 2: Replace edges ag and fe.

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.



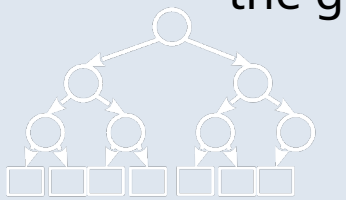
Step 3: Replace edges ab and ed.

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 convinient, 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.



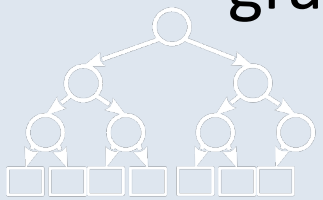
Traveling Salesman Problem (TSP)

- 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



Problem putujućeg trgovca (TSP)

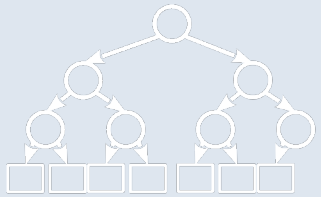
- Trgovac kreće iz određene točke i treba posjetiti sve svoje *kupce*
- Vrhovi grafa predstavljaju mjesta koja trgovac obilazi – uključujući i polazno/završno mjesto
  - Polazni vrh je ujedno i završni vrh – možemo ga gledati kao trgovčevo matično poduzeće
- Bridovi između vrhova imaju težine koje označavaju *trošak* putovanja od jednog kupca do drugog
  - Ovo možete zamisliti kao skup svih troškova koje trgovac treba uložiti za putovanje između dva kupca, recimo: vrijeme (broj dana putovanja), udaljenost, troškovi prijevoza, itd...
- Potrebno je pronaći najefikasniji put kojim trgovac obilazi sve kupce u grafu – za trgovca to znači najmanji trošak obilaska



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

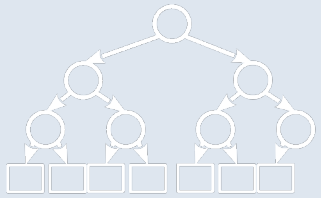
$$'!"\# = \arg\min_{(\$} 3\ (\ ) \quad)\in+((\$)$$



Problem putujućeg trgovca

- Za razliku od kineskog poštara, problem putujućeg trgovca fokusiran je na vrhove grafa
  - Imamo vrh koji predstavlja vlastitu prodajnu agenciju – početni/završni vrh
  - Svaki vrh je jedan kupac koji se obilazi točno jednom
  - Brid, to jest put između dva kupca, se može koristiti proizvoljan broj puta
- Uočavamo da je za rješenje ovog problema potrebno pronaći najoptimalniji Hamiltonov ciklus u grafu
  - Pretpostavimo da u grafu imamo više Hamiltonovih ciklusa
  - Potrebno je pronaći onaj čija je suma težina svih bridova najmanja

$$C_{H_{opt}} = \arg\min_{C_H} \sum_{e \in E(C_H)} w(e)$$

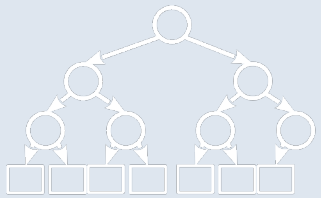


2-MST Heuristics

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

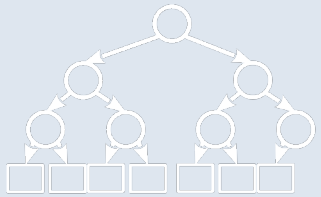


2-MST Heuristika

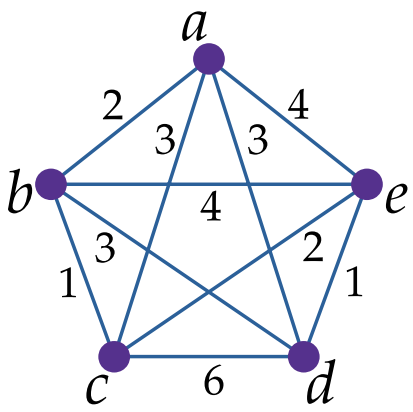
1: Construct the minimal spanning tree G_{MST} from the graph G.
2: Eulerize the minimal spanning tree G_{MST} by doubling all edges.
3: Find the Eulerian circuit C_E in Eulerized G_{MST}.
4: Turn the Eulerian circuit C_E into a Hamiltonian cycle C_H by skipping already visited vertices and jumping to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

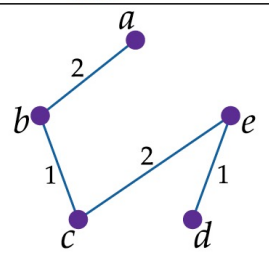
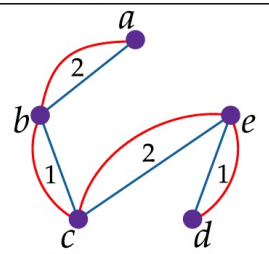
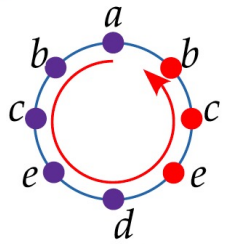
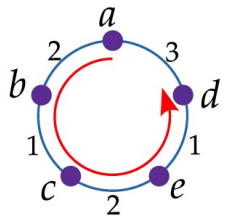
- Metoda rješavanja problema putujućeg trgovca

1. Korištenjem adekvatnog algoritma stvorimo minimalno razapinjujuće stablo
2. Euleriziramo minimalno razapinjujuće stablo
3. Pronađemo Eulerov krug koristeći zadani početni/završni vrh
4. Pronađemo Hamiltonov ciklus iz Eulerovog kruga
   - Recimo DFS-om prolazimo kroz min.raz. stablo, a u trenutku kada se vraćamo na već obiđeni vrh, preskačemo na prvi neobiđeni
   - Iz Eulerovog kruga izbacujemo duple vrhove, a bridove između novih parova vrhova uzimamo iz originalnog grafa

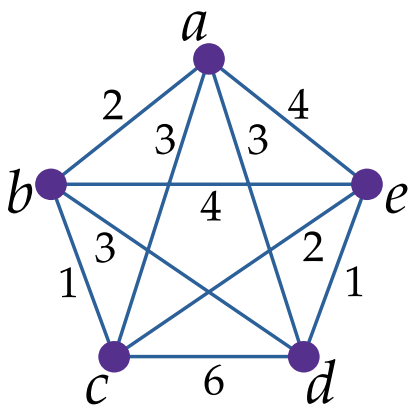


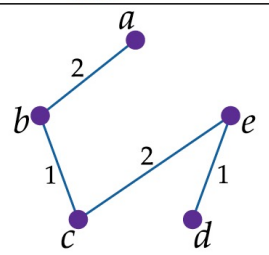
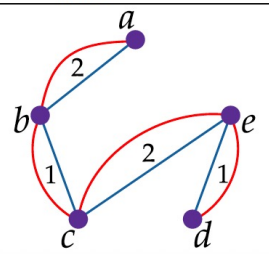
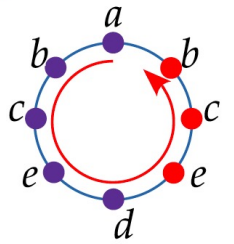
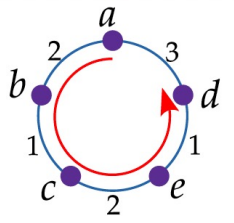
2-MST Heuristics



Step 1: Construct the minimal spanning tree G_{MST}	
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.	
Step 2: Eluerize the minimal spanning tree G_{MST}.	
Eulerize the minimal spanning tree by doubling all edges.	
Step 3: Find the Eulerian circuit C_E.	
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 the Hamiltonian cycle C_H.	
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.	

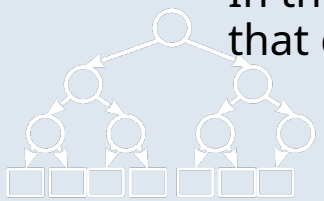
2-MST Heuristika



Step 1: Construct the minimal spanning tree G_{MST}	
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.	
Step 2: Eluerize the minimal spanning tree G_{MST}.	
Eulerize the minimal spanning tree by doubling all edges.	
Step 3: Find the Eulerian circuit C_E.	
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 the Hamiltonian cycle C_H.	
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.	

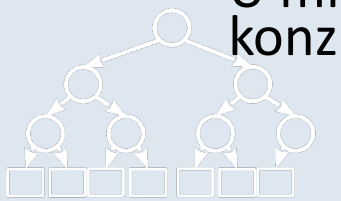
Flow in networks

- 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*)



Protok u mrežama

- Mreže se mogu modelirati kao grafovi, gdje:
  - Vrhovi predstavljaju mjesta, sjecišta, stanice
  - Bridovi predstavljaju spojne vodove, kao na primjer cijevi, trake, žice, to jest sve što podržava tok materijala, artikala, simbola
- Problem: pokušajmo izračunati najveći mogući protok vozila i opterećenje prometnica od Karlovca do Splita
  - Recimo da kapacitet prometnice definiramo brojem vozila po jedinici vremena koje ta prometnica propušta
- Slični problemi su:
  - Vodovodna mreža, elektroenergetska mreža (i druge distribucijske mreže), komunikacijsko-računalne mreže, evakuacijski planovi
- Tok može biti:
  - Kontinuiran – Recimo fluidi (plin, tekućina) ili krutine (m^3/s), električna struja (A) i slično
  - Diskretan – Uvijek postoji broj nekih artikala u jedinici vremena, recimo broj vozila, paketa, ljudi ili slično
- U mreži uvijek imamo vrhove koji generiraju tok (izvori - *sources*) i vrhove koji konzumiraju taj tok (odredišta - *destinations*)



Flow in networks

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

$$= (, , , , ,)$$

  - where is capacity mapping function to network edges, is the flow mapping function to the network edges, $\in$ is the vertex that generates the flow, a $\in$ is the vertex that consumes the stream

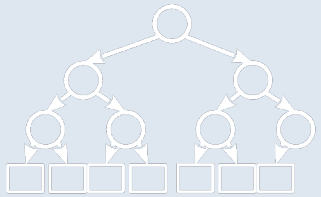
- When we say that there are no return edges, it is considered

$$\forall \in \nexists \in$$

- The capacity mapping function is

$$: \rightarrow \mathbb{R}($$

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



Protok u mrežama

- Mreža s protokom (*flow network*) je spojeni usmjereni graf bez petlji i povratnih bridova

$$G = (V, E, c, f, s, d)$$

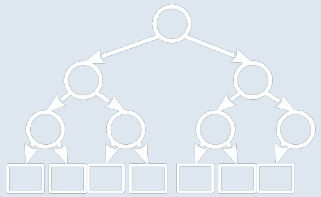
  - gdje je c funkcija mapiranja kapaciteta na bridove mreže, f je funkcija mapiranja toka na bridove mreže, $s \in V$ je vrh koji generira tok, a $d \in V$ je vrh koji konzumira tok
- Kada kažemo da nema povratnih bridova, smatra se

$$\forall uv \in E \nexists vu \in E$$

- Funkcija mapiranja kapaciteta je

$$c: E \to \mathbb{R}^+$$

  - Svaki brid mreže G ima naznačeni pozitivni kapacitet, što predstavlja maksimalni mogući tok kroz taj brid



Flow in networks

- The flow mapping function is

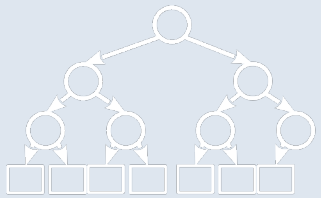
$$: \rightarrow \mathbb{R}($$

  - 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

$$), *\subseteq$$

$$),(*= :) \in\{, \in), \in * \}$$



Protok u mrežama

- Funkcija mapiranja toka je

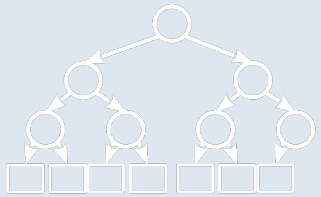
$$f: E \to \mathbb{R}^+$$

  - koristi se za mapiranje vrijednosti pozitivnog toka na brid mreže.

- Pomoćno mapiranje bridova koje će nam koristiti kod razrade protoka u mrežama

$$V_i, V_j \subseteq V$$

$$E(V_i, V_j) = \{uv: uv \in E, u \in V_i, v \in V_j\}$$



Flow in networks

- Flow has the following properties:
  - **Capacity limitation:**

$$\forall \in : 0 \leq ( ) \leq ( )$$

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

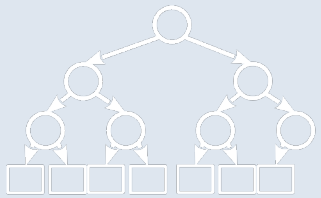
  - **Flow conservation:**

$$\forall \in \setminus , \{: \} \quad 3 \quad \$( ) = \quad 3 \quad (2)$$

$$. \&/\in + 0,/ \;) \qquad /.'\in +(/,0)$$

    - 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



Protok u mrežama

- Tok f ima sljedeća svojstva:
  - **Ograničenje kapacitetom:**

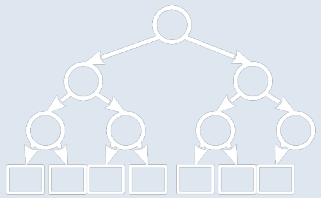
$$\forall uv \in E: 0 \leq f(uv) \leq c(uv)$$

    - Maksimalni tok u nekom bridu mreže jednak je kapacitetu tog brida

  - **Očuvanje toka:**

$$\forall u \in V \setminus \{s, d\}: \sum_{v_i u \in E(V,u)} f(v_i u) = \sum_{u v_j \in E(u,V)} f(u v_j)$$

    - Za svaki vrh koji ne generira ili ne konzumira tok vrijedi da je ulazni tok jednak izlaznom toku
    - Ova ravnoteža ne vrijedi za vrhove koji generiraju i konzumiraju tokove
      - Očito je da za vrh koji generira tok, izlazni tok mora biti veći od ulaznog
      - Za vrh koji konzumira tok ulazni tok mora biti veći od izlaznog



Flow in networks

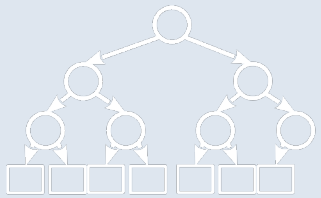
- **Current strength**is defined as

$$|=|@ \quad (\ ))- \quad @ \quad (*)$$

$$+\ ,!\in.(+,1) \qquad ,\ "+\in.(1,+)$$

  - 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| |

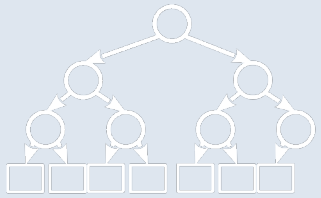


Protok u mrežama

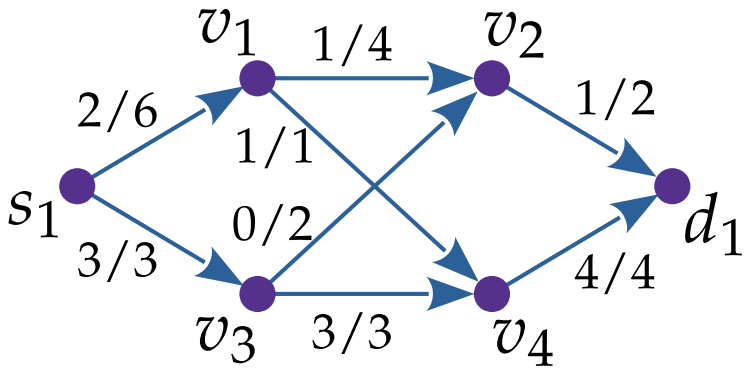
- **Jakost toka** definirana je kao

$$|f| = \sum_{sv_i \in E(s,V)} f(sv_i) - \sum_{v_j s \in E(V,s)} f(v_j s)$$

  - gdje je s vrh generator toka
- Kao što je već naglašeno, za vrh koji generira tok, izlazni tok mora biti veći od ulaznog, a sama razlika predstavlja jakost toka
  - Kako bismo pojednostavnili, koristiti će se samo jedan vrh koji generira tok $s \in V$ i jedan vrh koji konzumira tok $d \in V$
  - Očito je da ako imamo više generatora toka, tada svaki vrh koji generira tok doprinosi ukupnoj jakost toka $|f|$



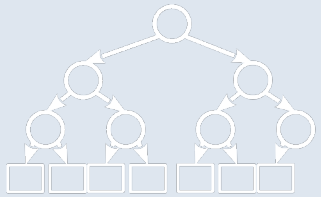
Flow in networks



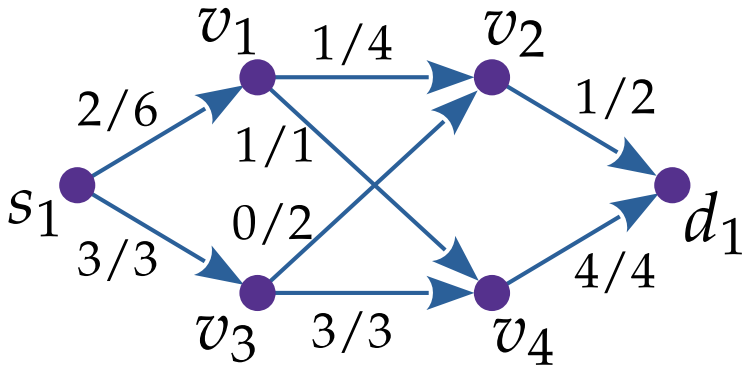
- In the network example we have
  - top &which generates the flow
  - top &which consumes the flow
  - each edge has a label

    ( )/ ( )

- Is the flow through the network the maximum possible?



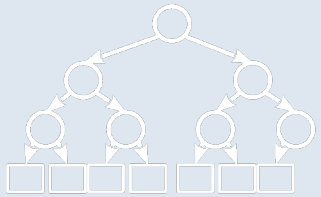
Protok u mrežama



- U primjeru mreže imamo
  - vrh s_1 koji generira tok
  - vrh d_1 koji konzumira tok
  - svaki brid ima oznaku

$$f(uv)/c(uv)$$

- Da li je tok kroz mrežu maksimalan mogući?



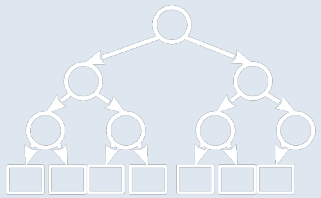
Ford-Fulkerson algorithm

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

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

  - where are they 3the edges of the remaining mesh are not the same as the edges of the original mesh
  - 3is the function of mapping the residual capacity to the edges of the residual network
  - 3is a function of mapping the residual flow to the edges of the residual network

- The edges of the remaining network can be:
  - Advanced(*forward*): $3 = \{\ :\ \in\ ,\ -\ > 0\ )\ (\ )\ \}$
  - Returnable(*reverse*): $3 = \{\ :\ \in\ ,\ > 0\ )\ \}$
  - So it holds: $3 = 3 \cup 3$

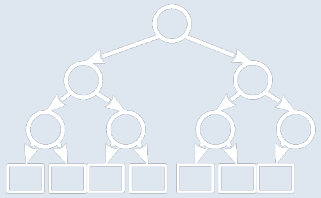


Ford-Fulkerson algoritam

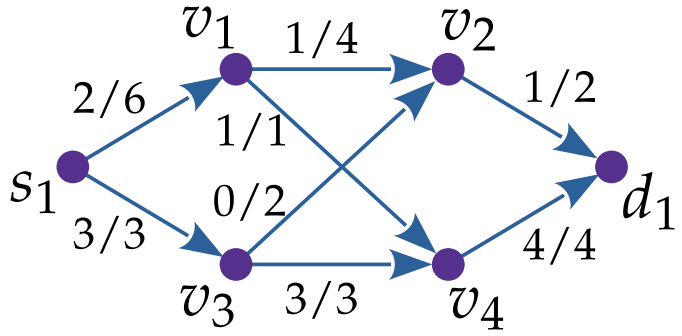
- **Ostatna mreža:** S obzirom na mrežu G, može se konstruirati ostatna mreža (*residual network*)

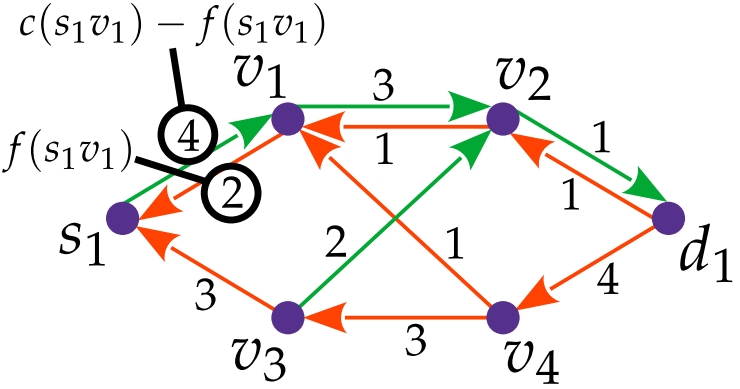
$$G_f = (V, E_f, c_f, f_f, s, d)$$

  - gdje su E_f bridovi ostatne mreže i nisu isti kao i bridovi originalne mreže E
  - c_f je funkcija mapiranja ostatnog kapaciteta na bridove ostatne mreže
  - f_f je funkcija mapiranja ostatnog toka na bridove ostatne mreže
- Bridovi ostatne mreže mogu biti:
  - Unaprijedni (*forward*): $E_{f_1} = \{uv : uv \in E, c(uv) - f(uv) > 0\}$
  - Povratni (*reverse*): $E_{f_2} = \{vu : uv \in E, f(uv) > 0\}$
  - Tako da vrijedi: $E_f = E_{f_1} \cup E_{f_2}$



Ford-Fulkerson algorithm





- **Last capacity** (*residual capacity*) on the rest of the network 3 is defined by the mapping function

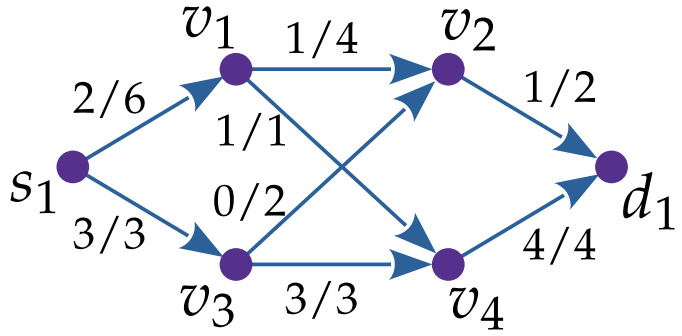
(–) , ∃ (∈ , – > 0 ( ) ( )

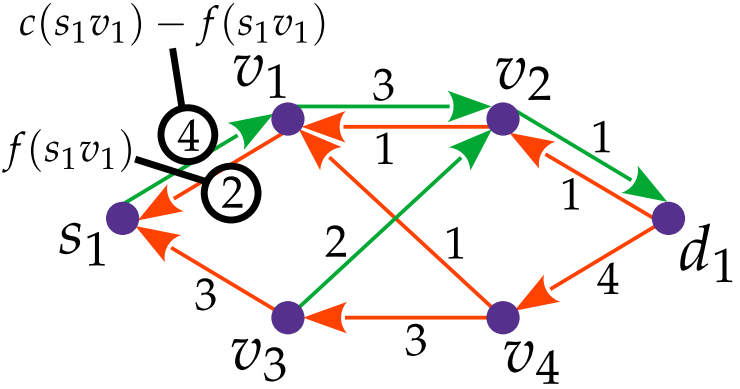
3 (= H) (,) ∃ ∈ , ≻(0)

0, h

  - 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 algoritam





- **Ostatni kapacitet** (*residual capacity*) na ostatnoj mreži G_f definiran je funkcijom mapiranja

$$c_f(uv) = \begin{cases} c(uv) - f(uv), & \exists uv \in E, c(uv) - f(uv) > 0 \\ f(vu), & \exists vu \in E, f(vu) > 0 \\ 0, & otherwise \end{cases}$$

  - Za unaprijedne bridove vrijedi da je ostatni kapacitet razlika kapaciteta i toka u originalnoj mreži G
  - Za povratne bridove vrijedi da je ostatni kapacitet jednak toku suprotnog brida u originalnoj mreži G

Ford-Fulkerson algorithm

- Similar to the original network ,the rest of the network 3can have streams, defined by a mapping function 3
- We define the term for flows in the remaining network**increase flow**(*flow augmentation*)as

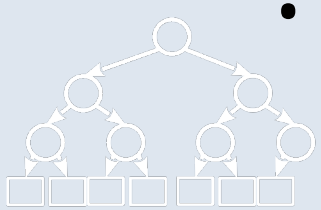
$$\uparrow_3 = (C \quad) \qquad (+_3) - _3 (, \exists) \in \quad (\quad)$$

$$0, \qquad h$$

  - where the flow in the original network increases by the current in the remaining network 3
  - 3is also called**increasing flow**(*increasing flow*)
- The same is true for**increase in flow strength**(*flow strength augmentation*)

$$|\uparrow_3 = |+ | \beta ||$$

  - check the proof in the script



Ford-Fulkerson algoritam

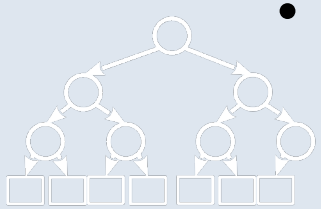
- Slično kao i kod originalne mreže G, ostatna mreža G_f može imati tokove, definirane funkcijom mapiranja f_f
- Za tokove u ostatnoj mreži definiramo pojam **uvećanja toka** (*flow augmentation*) kao

$$f \uparrow f_f(uv) = \begin{cases} f(uv) + f_f(uv) - f_f(vu), & \exists uv \in E \\ 0, & otherwise \end{cases}$$

  - gdje se tok u originalnoj mreži f uvećava za tok u ostatnoj mreži f_f
  - f_f naziva se i **uvećavajući tok** (*augmenting flow*)
- Slično vrijedi i za **uvećanje jakosti toka** (*flow strength augmentation*)

$$\left|f \uparrow f_f\right| = |f| + |f_f|$$

  - dokaz provjerite u skripti



Ford-Fulkerson algorithm

- **Enlarging the road**(*augmenting path*): Given the network and the rest of her network (, any time from the top to the top in (it is called the augmenting path
  - Each such path gives us the possibility to define an increasing flow &on the rest of the network &, and thereby increasing the flow in the original network ↑ &
  - The remaining capacity of the increasing path is

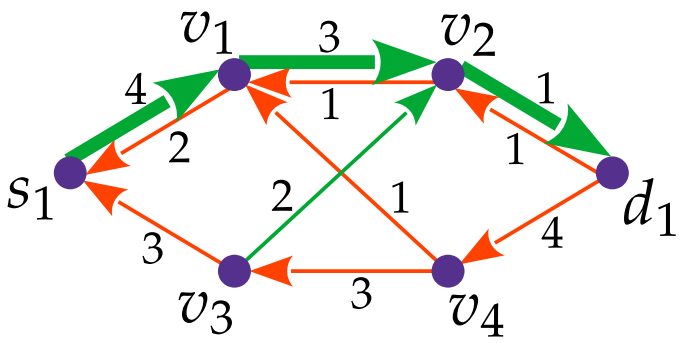
$$\& (\in) \min_{\forall()\in+} \&( )$$

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

$$\forall \in : \& \; ( ) = \&( )$$

  - Increasing the flux in the original network results

$$\left|\uparrow \&! \right| = |+|| \&|>|f|$$



Ford-Fulkerson algoritam

- **Uvećavajući put** (*augmenting path*): S obzirom na mrežu G i njenu ostatnu mrežu G_f, bilo koji put p od vrha s do vrha d u G_f naziva se uvećavajući put
  - Svaki takav put daje nam mogućnost definiranja uvećavajućeg toka f_f na ostatnoj mreži G_f, a time i uvećanje toka u originalnoj mreži $f \uparrow f_f$
  - Ostatni kapacitet uvećavajućeg puta p je

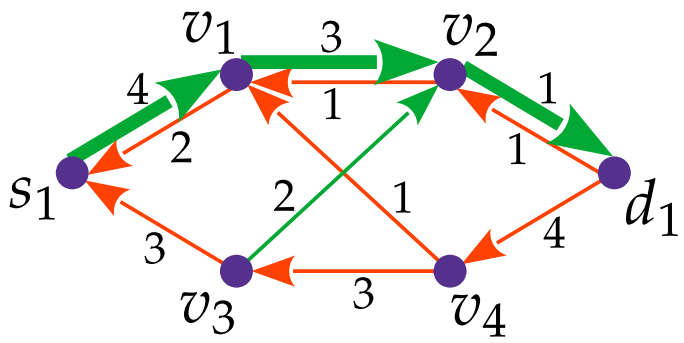
$$c_f(p) = \min_{\forall uv \in p} c_f(uv)$$

    - čime se uvećavajući tok f_f na putu p ograničava na minimalni ostatni kapacitet na tom putu
  - Uvećavajući tok na putu p ili f_{f_p} računa se kao

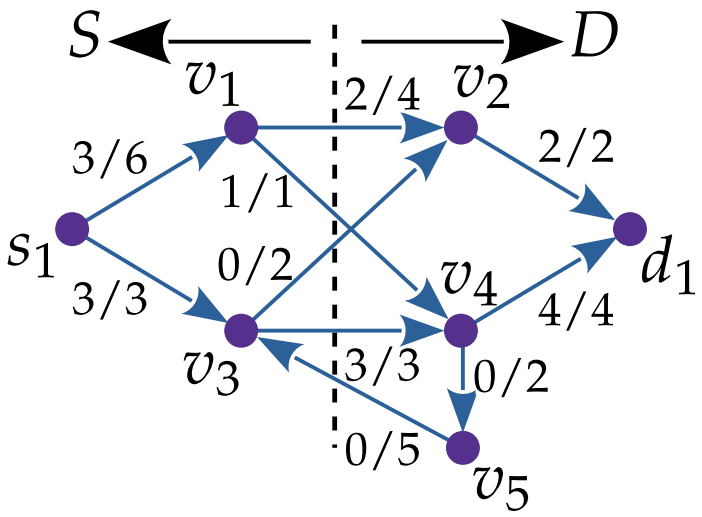
$$\forall uv \in p: f_{f_p}(uv) = c_f(p)$$

  - Uvećanje toka u originalnoj mreži rezultira

$$\left|f \uparrow f_{f_p}\right| = |f| + |f_{f_p}| > |f|$$



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

  ⊂ , ⊂ V

  - so it's worth it ∈ , ∈ , ∩ = Ø and ∪ =
- With this, we have defined the cross-section of the network , through which the edges pass (,).
- The network cross section capacity is calculated

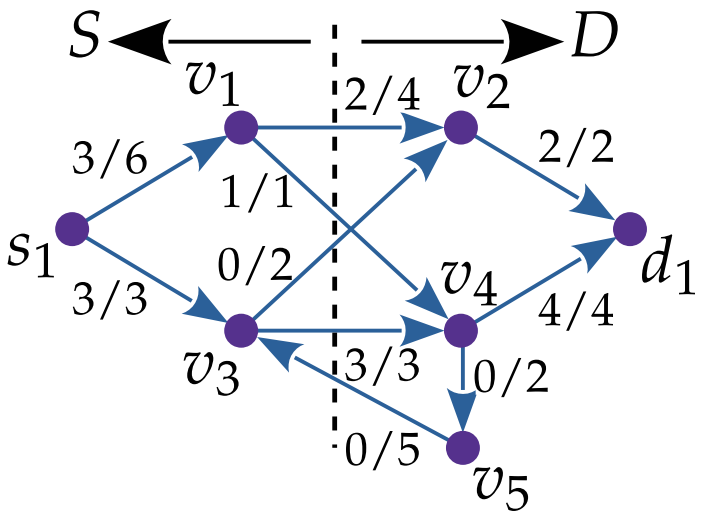
  ,(= C) ( )

  ()∈,(.,0)
- The cross-sectional flow of the network is calculated

  ,(= C)( ) − C ( )

  ()∈,(.,0)) (∈,(0,.)
- Due to the flow conservation property, it is valid | | = (,) (proof see in scripts)

Presjek mreže



- Svaki mrežu možemo podijeliti na način da podijelimo vrhove mreže u dvije disjunktne particije

$$S \subset V, D \subset \mathrm{V}$$

  - tako da vrijedi $s \in S, d \in D, S \cap D = \emptyset$ i $S \cup D = V$
- Time smo definirali presjek mreže S, D kroz koji prolaze bridovi $E(S, D)$.
- Kapacitet presjeka mreže računa se

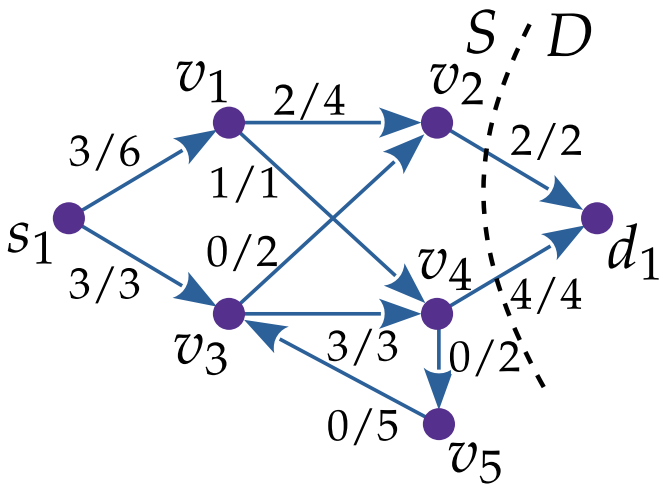
$$c(S, D) = \sum_{uv \in E(S,D)} c(uv)$$

- Tok presjeka mreže računa se

$$f(S, D) = \sum_{uv \in E(S,D)} f(uv) - \sum_{vu \in E(D,S)} f(vu)$$

- Zbog svojstva očuvanja toka vrijedi $|f| = f(S, D)$ (dokaz vidi u skripti)

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

$$_4 , (= m)\text{in } (\ , \ ) \qquad _{5,6}$$

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

$$0 \le \quad ,(\ = |)| \le \ (\ , \ )$$

- **Evidence:**

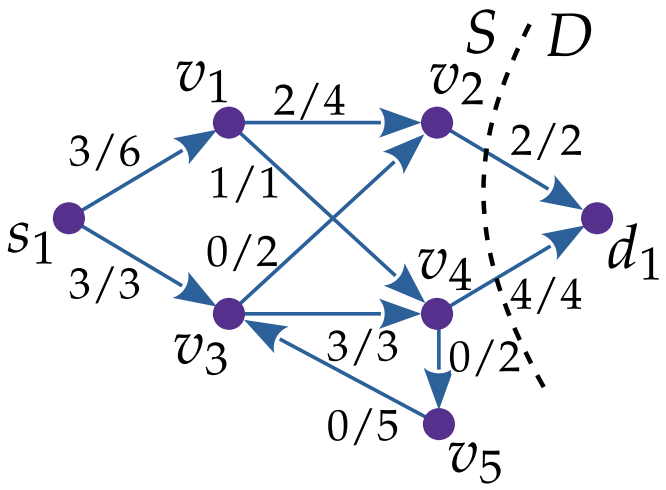
$$|=| \ , \quad (\quad) = \quad 3 \quad (\ ) - \quad 3 \quad (\quad)$$

$$/.\in+(5,6) \qquad . /\in+ 6(.5 \ )$$

$$\le \quad 3 \quad (\le \ ) \quad 3 \quad = (\ , \ ))$$

$$/.\in+ 5(6 \ ) \qquad /.\in+ 5(6 \ )$$

Minimalni presjek mreže (min-cut)



- Presjek mreže S, D koji ima najmanji kapacitet $c(S, D)$ naziva se minimalnim presjekom (*minimal cut*) i dobiva se optimizacijom

$$c_m(S, D) = \min_{S,D} c(S, D)$$

- **Korolar:** Ograničenje kapacitetom (*network cut capacity constraint*) se može primijeniti na bilo koji presjek mreže S, D, čime vrijedi

$$0 \leq f(S, D) = |f| \leq c(S, D)$$

- **Dokaz:**

$$|f| = f(S, D) = \sum_{uv \in E(S,D)} f(uv) - \sum_{vu \in E(D,S)} f(vu)$$

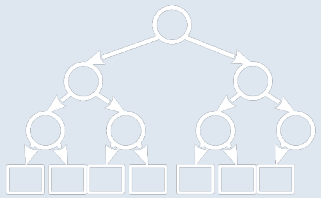
$$\leq \sum_{uv \in E(S,D)} f(uv) \leq \sum_{uv \in E(S,D)} c(uv) = c(S, D)$$

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 |va|id = (,)

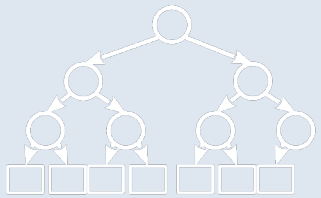
- **Proof (→):**Assume that the 2nd does not hold, that is, there is an increasing path in the rest of the network 3. This means that there is a flow 3# in the rest of the network 3which can increase the flow in the original network . That's how we get ↑# 3> | |,which is in direct contradiction with 1.



Max-flow min-cut teorem

- S obzirom na mrežu G koja ima tok f, generatora toka s, konzumera toka d, vrijede sljedeće tvrdnje:
  1. f je maksimalni tok u mreži G,
  2. u ostatnoj mreži G_f nema više uvećavajućih puteva,
  3. postoji presjek mreže S, D takav da vrijedi $|f| = c(S, D)$

- **Dokaz ($\mathbf{1 \rightarrow 2}$):** Pretpostavimo da ne vrijedi 2., to jest postoji uvećavajući put p u ostatnoj mreži G_f. To znači da postoji tok f_{f_p} u ostatnoj mreži G_f koji može uvećati tok f u originalnoj mreži G. Time dobivamo $\left|f \uparrow f_{f_p}\right| > |f|$, što je u direktnoj kontradikciji sa 1.



Max-flow min-cut theorem

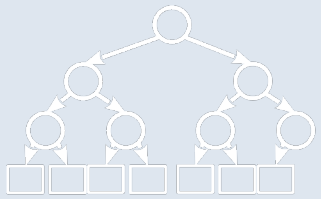
- **Proof (→):**If there is no augmenting path in the rest of the network 3, then there is no path between the vertices at all and ,which means there is an intersection , for which there is not a single edge between the vertices and in the rest of the network 3, which means it is valid

$$\forall \in , (\nexists \ni 3 \Rightarrow 3 = 0, \quad (= ()) \qquad (\quad)$$

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

$$|=| \ , \ \neq (,)$$

which is identical to statement 3.



Max-flow min-cut teorem

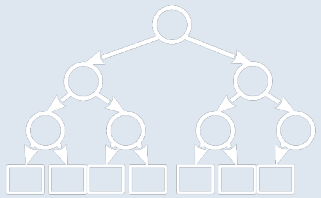
- **Dokaz ($2 \rightarrow 3$):** Ako ne postoji uvećavajući put p u ostatnoj mreži G_f, tada uopće ne postoji put između vrhova s i d, što znači da postoji presjek S, D za koji ne postoji niti jedan brid između vrhova S i D u ostatnoj mreži G_f, što znači da vrijedi

$$\forall uv \in E(S,D) \nexists uv \in E_f \Rightarrow c_f(uv) = 0, c(uv) = f(uv)$$

S obzirom na prethodni korolar, za presjek S, D vrijedi

$$|f| = f(S,D) = c(S,D)$$

što je identično tvrdnji 3.



Max-flow min-cut theorem

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

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

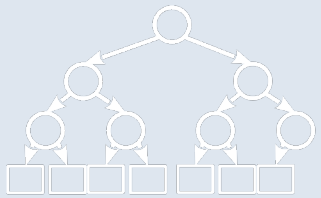
and it is valid that the section at which it is valid

$$|=|(\ ,\ )$$

maximum flow in the network

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

$$|=|_4(\ ,\ )$$



Max-flow min-cut teorem

- **Dokaz ($3 \rightarrow 1$):** Korolar je direktan dokaz, to jest

$$|f| \leq c(S, D)$$

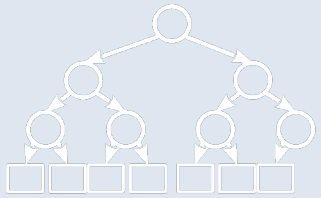
te vrijedi da je presjek kod kojeg vrijedi

$$|f| = c(S, D)$$

maksimalan tok u mreži

- Zbog ovog teorema vrijedi da je maksimalni tok jednak minimalnom presjeku mreže

$$|f| = c_m(S, D)$$

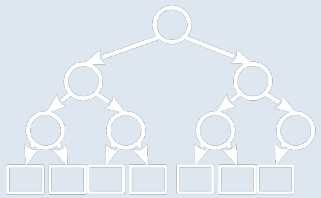


Ford-Fulkerson algorithm

```
procedure FordFulkerson(G)
    for each edge uv in E(G) do
        f(uv) ← 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 ↑ 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

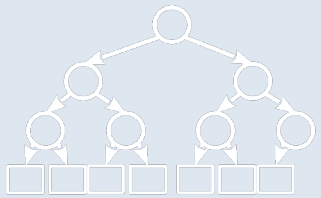


Ford-Fulkerson algoritam

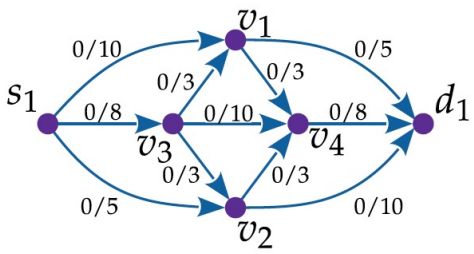
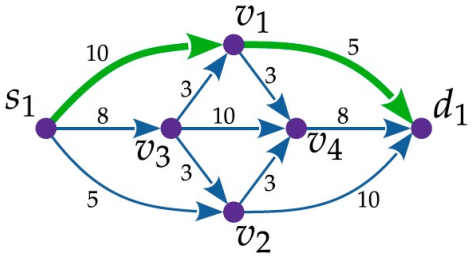
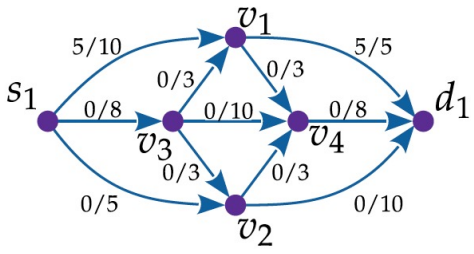
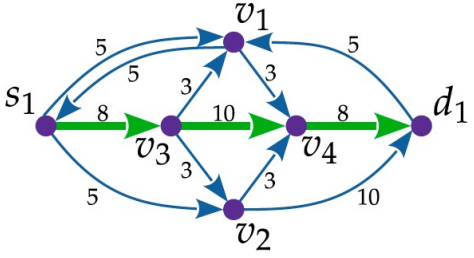
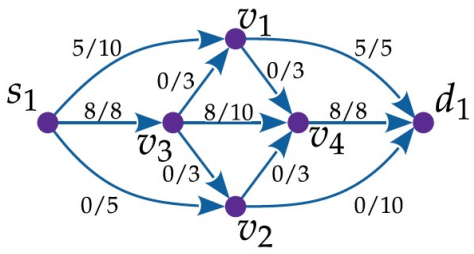
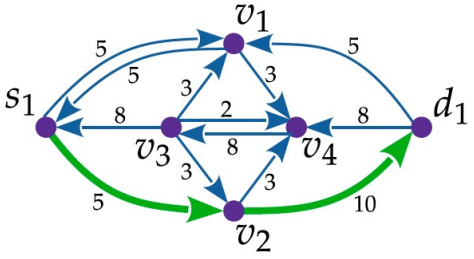
procedure FORDFULKERSON(G)
  for each edge uv in $E(G)$ **do**
    $f(uv) \leftarrow 0$
  calculate the residual network G_f ▷ Def. 4.44
  while there is an augmenting path p in G_f **do**
    calculate f_{f_p} ▷ (4.99)
    perform the flow augmentation $(f \uparrow f_{f_p})$ ▷ (4.93)
    calculate the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

- Kompleksnost ovog algoritma je $O(f_m + E)$
- Gdje je f_m broj uvećavajućih puteva

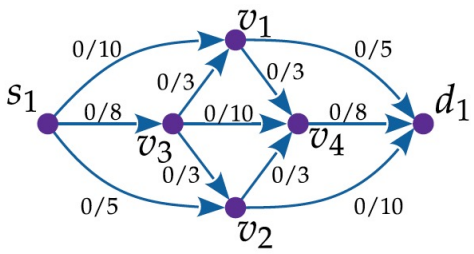
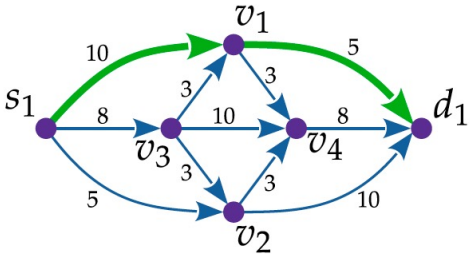
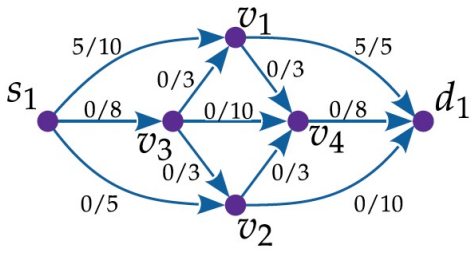
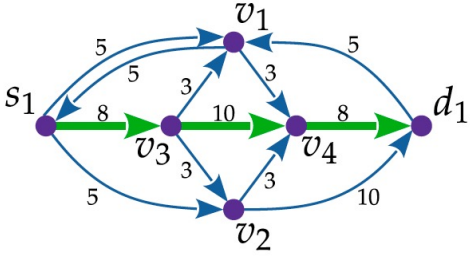
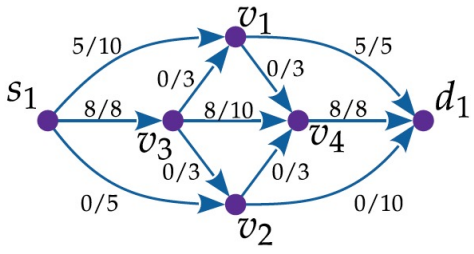
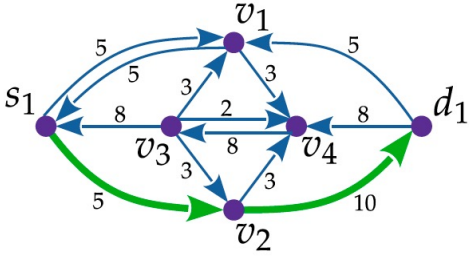
- Inicijaliziramo osnovnu mrežu G postavljajući tokove na 0
- Izračunamo ostatnu mrežu G_f
  - Pronađemo neki uvećavajući put p u ostatnoj mreži G_f
  - Izračunamo uvećavajući tok f_{f_p}
  - Uvećamo tok u osnovnoj mreži G
  - Ažuriramo ostatnu mrežu G_f
  - Ponavljamo tako dugo dok postoji put između s i d u ostatnoj mreži G_f



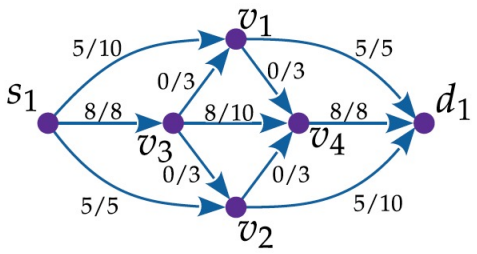
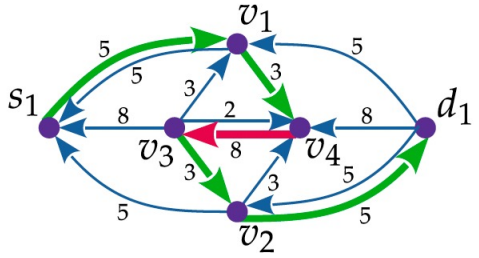
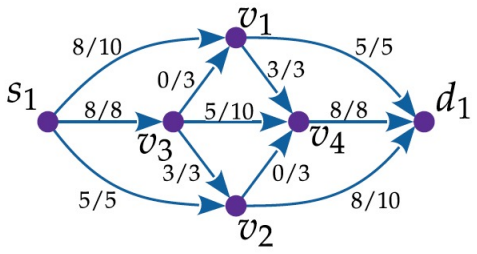
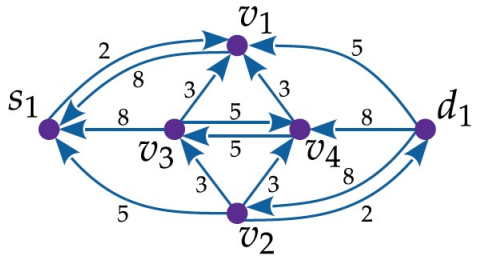
Ford-Fulkerson algorithm

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 $\lvert f_{f_p} \rvert = 5$.		
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 $\lvert f_{f_p} \rvert = 8$.		
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 $\lvert f_{f_p} \rvert = 5$.		

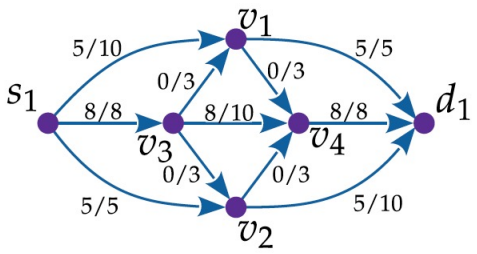
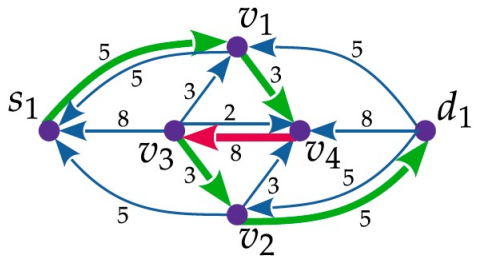
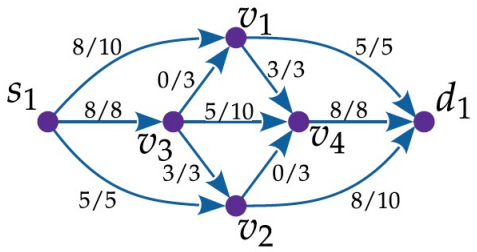
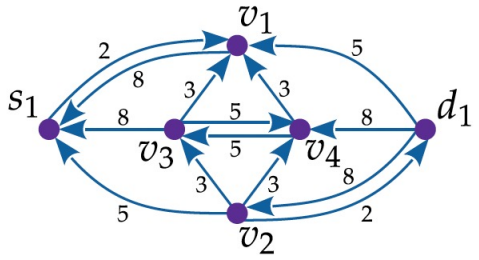
Ford-Fulkerson algoritam

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 $\|f_{f_p}\| = 5$.		
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 $\|f_{f_p}\| = 8$.		
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 $\|f_{f_p}\| = 5$.		

Ford-Fulkerson algorithm

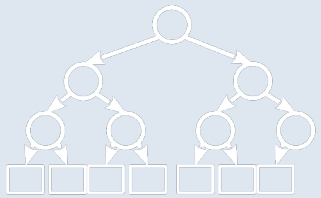
4: We choose the augmenting path $p = s_1v_1v_4v_3v_2d_1$, which has the minimal residual capacity $c_f(p) = 3$. The augmenting path flow strength is therefore $\|f_{f_p}\| = 3$. Notice that we used reversed edge v_4v_3 from the residual network G_f.		
5: We have no more available augmenting paths. The final maximal flow is $\|f\| = 21$.		

Ford-Fulkerson algoritam

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.



Edmonds-Karp algoritam

- Nadogradnja Ford-Fulkerson algoritma kojom se biranje uvećavajućeg puta radi korištenjem BFS algoritma, tako da se prioritiziraju kraći putevi s obzirom na broj bridova.