NEW CS 473: Theory II, Fall 2015

Matchings II

Lecture 17 October 22, 2015

17.1: Maximum weight matchings in a bipartite graph

17.1.1: On the structure of the problem

- For alternating path/cycle π :

$$\gamma(\pi, M) = \sum_{e \in \pi \setminus M} w(e) - \sum_{e \in \pi \cap M} w(e). \tag{1}$$

- =total weight of the free edges in π minus weight of matched edges.
- $exttt{0}$ Useful lemma: $\gamma(\pi,M)>0 \implies w(M')>w(M)$.

- For alternating path/cycle π :
- weight (for matching M):

$$\gamma(\pi, M) = \sum_{e \in \pi \setminus M} w(e) - \sum_{e \in \pi \cap M} w(e). \tag{1}$$

- \odot = total weight of the free edges in π minus weight of matched edges.
- $exttt{0}$ Useful lemma: $\gamma(\pi,M)>0 \implies w(M')>w(M)$.

- For alternating path/cycle π :
- weight (for matching M):

$$\gamma(\pi, M) = \sum_{e \in \pi \setminus M} w(e) - \sum_{e \in \pi \cap M} w(e). \tag{1}$$

- \odot = total weight of the free edges in π minus weight of matched edges.
- $ext{ Useful lemma: } \gamma(\pi,M)>0 \implies w(M')>w(M).$

- For alternating path/cycle π :
- **2** weight (for matching M):

$$\gamma(\pi, M) = \sum_{e \in \pi \setminus M} w(e) - \sum_{e \in \pi \cap M} w(e). \tag{1}$$

- \odot = total weight of the free edges in π minus weight of matched edges.
- Useful lemma: $\gamma(\pi, M) > 0 \implies w(M') > w(M)$.

$$\gamma(\pi, M) = \sum_{e \in \pi \setminus M} w(e) - \sum_{e \in \pi \cap M} w(e). \tag{2}$$

Lemma

M: a matching. π : alternating path/cycle with positive weight relative to M.

 $\gamma(\pi,M)>0$. Furthermore, assume that

$$M' = M \oplus \pi = (M \setminus \pi) \cup (\pi \setminus M)$$

is a matching. Then $w(M^\prime)$ is bigger; namely, $w(M^\prime)>w(M)$.

$$egin{aligned} w(M') - w(M) &= \sum_{e \in M'} w(e) - \sum_{e \in M} w(e) \ &= \sum_{e \in M' \setminus M} w(e) - \sum_{e \in M \setminus M'} w(e) \ &= \sum_{e \in \pi \setminus M} w(e) - \sum_{e \in M \setminus \pi} w(e) \ &= \gamma(\pi, M). \end{aligned}$$

Just observe that $w(M') = w(M) + \gamma(\pi, M)$.

Sariel (UIUC) New CS473 6 Fall 2015 6 / 48

Augmenting...

• Augmenting path in the weighted case:

Definition

An alternating path is **augmenting** if it starts and ends in a free vertex.

Observation:

If M has an augmenting path π then M is not of maximum size matching (this is for the unweighted case), since $M\oplus\pi$ is a larger matching.

Augmenting...

• Augmenting path in the weighted case:

Definition

An alternating path is **augmenting** if it starts and ends in a free vertex.

Observation:

If M has an augmenting path π then M is not of maximum size matching (this is for the unweighted case), since $M\oplus\pi$ is a larger matching.

Augmenting by heaviest augmenting path is good...

Theorem

Let M be a matching of maximum weight among matchings of size |M|. Let π be an augmenting path for M of maximum weight, and let T be the matching formed by augmenting M using π . Then T is of maximum weight among matchings of size |M|+1.

- $oldsymbol{\circ}$ S: matching of maximum weight among all matchings with |M|+1 edges.
- \odot Cycle or even length path σ in H.

 - 2 If $\gamma(\sigma,M)>0$ then $M\oplus \sigma$ matching of same size as M but heavier. Contradiction.
 - $\text{ if } \gamma(\sigma,M)<0 \text{ than } \gamma(\sigma,S)=-\gamma(\sigma,M) \text{ and as such } \\ S\oplus\sigma \text{ is heavier than } S. \text{ A contradiction.}$
- ullet U_S : All odd length paths in H that have one edge more in S than in M.
- ullet U_M : All odd length paths in H that have one edge more of M than an edge of S.

- $oldsymbol{S}$: matching of maximum weight among all matchings with |M|+1 edges.
- **2** $H = (V, M \oplus S).$
- \odot Cycle or even length path σ in H.
 - ① Must be $\gamma(\sigma, M) = 0$.
 - 2 If $\gamma(\sigma,M)>0$ then $M\oplus\sigma$ matching of same size as M but heavier. Contradiction.
 - $\text{ if } \gamma(\sigma,M)<0 \text{ than } \gamma(\sigma,S)=-\gamma(\sigma,M) \text{ and as such } \\ S\oplus\sigma \text{ is heavier than } S. \text{ A contradiction.}$
- ① Same arg: If σ is even path in H then $\gamma(\sigma,M)=0$.
- $ullet U_S$: All odd length paths in $oldsymbol{H}$ that have one edge more in S than in $oldsymbol{M}$.
- ${f 0}$ U_M : All odd length paths in H that have one edge more of M than an edge of S.

- $oldsymbol{\circ}$ S: matching of maximum weight among all matchings with |M|+1 edges.
- $H = (V, M \oplus S).$
- **3** Cycle or even length path σ in H.
 - ① Must be $\gamma(\sigma, M) = 0$.
 - 2 If $\gamma(\sigma,M)>0$ then $M\oplus\sigma$ matching of same size as M but heavier. Contradiction.
 - $\text{ if } \gamma(\sigma,M)<0 \text{ than } \gamma(\sigma,S)=-\gamma(\sigma,M) \text{ and as such } \\ S\oplus\sigma \text{ is heavier than } S. \text{ A contradiction}.$
- ullet U_S : All odd length paths in H that have one edge more in S than in M.
- ullet U_M : All odd length paths in H that have one edge more of M than an edge of S.

- $oldsymbol{S}$: matching of maximum weight among all matchings with |M|+1 edges.
- $H = (V, M \oplus S).$
- **3** Cycle or even length path σ in H.
 - Must be $\gamma(\sigma, M) = 0$.
 - ② If $\gamma(\sigma,M)>0$ then $M\oplus \sigma$ matching of same size as M but heavier. Contradiction.
 - if $\gamma(\sigma,M)<0$ than $\gamma(\sigma,S)=-\gamma(\sigma,M)$ and as such $S\oplus\sigma$ is heavier than S. A contradiction.
- ① Same arg: If σ is even path in H then $\gamma(\sigma,M)=0$.
- ullet U_S : All odd length paths in H that have one edge more in S than in M.
- ${f 0}$ U_M : All odd length paths in H that have one edge more of M than an edge of S.

- $oldsymbol{\circ}$ S: matching of maximum weight among all matchings with |M|+1 edges.
- $H = (V, M \oplus S).$
- **3** Cycle or even length path σ in H.
 - Must be $\gamma(\sigma, M) = 0$.
 - 2 If $\gamma(\sigma,M)>0$ then $M\oplus \sigma$ matching of same size as M but heavier. Contradiction.
 - \circ if $\gamma(\sigma,M)<0$ than $\gamma(\sigma,S)=-\gamma(\sigma,M)$ and as such $S\oplus\sigma$ is heavier than S. A contradiction.
- $ullet U_S$: All odd length paths in $oldsymbol{H}$ that have one edge more in S than in $oldsymbol{M}$.
- ${f 0}$ U_M : All odd length paths in H that have one edge more of M than an edge of S.

- $oldsymbol{\circ}$ S: matching of maximum weight among all matchings with |M|+1 edges.
- $H = (V, M \oplus S).$
- **3** Cycle or even length path σ in H.
 - Must be $\gamma(\sigma, M) = 0$.
 - 2 If $\gamma(\sigma,M)>0$ then $M\oplus \sigma$ matching of same size as M but heavier. Contradiction.
 - $oldsymbol{\circ}$ if $\gamma(\sigma,M)<0$ than $\gamma(\sigma,S)=-\gamma(\sigma,M)$ and as such $S\oplus\sigma$ is heavier than S. A contradiction.
- ullet U_S : All odd length paths in H that have one edge more in S than in M.
- ullet U_M : All odd length paths in H that have one edge more of M than an edge of S.

- $oldsymbol{S}$: matching of maximum weight among all matchings with |M|+1 edges.
- $H = (V, M \oplus S).$
- **3** Cycle or even length path σ in H.
 - Must be $\gamma(\sigma, M) = 0$.
 - 2 If $\gamma(\sigma,M)>0$ then $M\oplus \sigma$ matching of same size as M but heavier. Contradiction.
 - if $\gamma(\sigma,M)<0$ than $\gamma(\sigma,S)=-\gamma(\sigma,M)$ and as such $S\oplus\sigma$ is heavier than S. A contradiction.
- lacksquare Same arg: If $oldsymbol{\sigma}$ is even path in H then $\gamma(oldsymbol{\sigma},M)=0$.
- ${f 0}$ U_S : All odd length paths in ${m H}$ that have one edge more in S than in ${m M}$.
- ullet U_M : All odd length paths in H that have one edge more of M than an edge of S.

- $oldsymbol{\circ}$ S: matching of maximum weight among all matchings with |M|+1 edges.
- $H = (V, M \oplus S).$
- **3** Cycle or even length path σ in H.

 - 2 If $\gamma(\sigma,M)>0$ then $M\oplus \sigma$ matching of same size as M but heavier. Contradiction.
 - if $\gamma(\sigma,M)<0$ than $\gamma(\sigma,S)=-\gamma(\sigma,M)$ and as such $S\oplus\sigma$ is heavier than S. A contradiction.
- lacksquare Same arg: If $oldsymbol{\sigma}$ is even path in H then $\gamma(oldsymbol{\sigma},M)=0$.
- **10** U_S : All odd length paths in H that have one edge more in S than in M.
- ullet U_M : All odd length paths in H that have one edge more of M than an edge of S.

- $oldsymbol{\circ}$ S: matching of maximum weight among all matchings with |M|+1 edges.
- $H = (V, M \oplus S).$
- **3** Cycle or even length path σ in H.

 - ② If $\gamma(\sigma,M)>0$ then $M\oplus\sigma$ matching of same size as M but heavier. Contradiction.
 - if $\gamma(\sigma,M)<0$ than $\gamma(\sigma,S)=-\gamma(\sigma,M)$ and as such $S\oplus\sigma$ is heavier than S. A contradiction.
- lacksquare Same arg: If $oldsymbol{\sigma}$ is even path in H then $\gamma(oldsymbol{\sigma},M)=0$.
- **10** U_S : All odd length paths in H that have one edge more in S than in M.
- $lackbox{0}{U_M}$: All odd length paths in H that have one edge more of M than an edge of S.

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- ② For $\pi \in U_S$ and $\pi' \in U_M$...
- Must be that $\gamma(\pi,M)+\gamma(\pi',M)=0.$
 - ① If $\gamma(\pi,M) + \gamma(\pi',M) > 0$ then $M \oplus \pi \oplus \pi'$ bigger weight than M.

- $\text{ If } \gamma(\pi,M) + \gamma(\pi',M) < 0 \text{ then } S \oplus \pi \oplus \pi' \text{ same number }$ of edges as S but heavier matching. A contradiction.
- $ext{ On Pair up the paths in } U_S ext{ to paths in } U_M.$
- 5 Total weight of such a pair is zero.
- \odot Only one path μ in U_S which not paired.
- $extbf{0} \ \gamma(\mu,M) = w(S) w(M)$ (everything else has balance $extbf{0}$).
- lacktriangledown Path must be the heaviest augmenting path for M... Otherwise, \exists heavier augmenting path σ' for M s.t. $w(M\oplus\sigma')>w(S)$. A contradiction.

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- $oldsymbol{2}$ For $\pi \in U_S$ and $\pi' \in U_M...$
- \bigcirc Must be that $\gamma(\pi,M) + \gamma(\pi',M) = 0$.
 - (1) If $\gamma(\pi,M) + \gamma(\pi',M) > 0$ then $M \oplus \pi \oplus \pi'$ bigger weight than M.

- ② If $\gamma(\pi,M) + \gamma(\pi',M) < 0$ then $S \oplus \pi \oplus \pi'$ same number of edges as S but heavier matching. A contradiction.
- $ext{ O Pair up the paths in } U_S ext{ to paths in } U_M.$
- Total weight of such a pair is zero.
- \odot Only one path μ in U_S which not paired.
- $\bigcirc \gamma(\mu,M) = w(S) w(M)$ (everything else has balance 0).

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- ② For $\pi \in U_S$ and $\pi' \in U_M$...
- $lacksquare{0}$ Must be that $\gamma(\pi,M)+\gamma(\pi',M)=0$.
 - ① If $\gamma(\pi,M) + \gamma(\pi',M) > 0$ then $M \oplus \pi \oplus \pi'$ bigger weight than M.

- 1 If $\gamma(\pi,M) + \gamma(\pi',M) < 0$ then $S \oplus \pi \oplus \pi'$ same number of edges as S but heavier matching. A contradiction.
- \P Pair up the paths in U_S to paths in U_M .
- Total weight of such a pair is zero.
- lacktriangledown Only one path μ in U_S which not paired.
- $extbf{0} \ \gamma(\mu,M) = w(S) w(M)$ (everything else has balance $extbf{0}$).
- lacktriangledown Path must be the heaviest augmenting path for M... Otherwise, \exists heavier augmenting path σ' for M s.t. $w(M\oplus\sigma')>w(S)$. A contradiction.

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- $oldsymbol{2}$ For $\pi \in U_S$ and $\pi' \in U_M...$
- $lacksquare{3}$ Must be that $\gamma(\pi,M)+\gamma(\pi',M)=0$.
 - $\hbox{ If } \gamma(\pi,M) + \gamma(\pi',M) > 0 \hbox{ then } M \oplus \pi \oplus \pi' \hbox{ bigger weight than } M.$

- o If $\gamma(\pi,M) + \gamma(\pi',M) < 0$ then $S \oplus \pi \oplus \pi'$ same number of edges as S but heavier matching. A contradiction.
- $ext{ O Pair up the paths in } U_S ext{ to paths in } U_M.$
- Total weight of such a pair is zero.
- lacktriangledown Only one path μ in U_S which not paired.
- $extbf{0} \ \gamma(\mu,M) = w(S) w(M)$ (everything else has balance $extbf{0}$).

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- ② For $\pi \in U_S$ and $\pi' \in U_M$...
- $lacksquare{3}$ Must be that $\gamma(\pi,M)+\gamma(\pi',M)=0$.
 - $\hbox{ If } \gamma(\pi,M) + \gamma(\pi',M) > 0 \hbox{ then } M \oplus \pi \oplus \pi' \hbox{ bigger weight than } M.$

- If $\gamma(\pi,M) + \gamma(\pi',M) < 0$ then $S \oplus \pi \oplus \pi'$ same number of edges as S but heavier matching. A contradiction.
- ① Pair up the paths in U_S to paths in U_M .
- Total weight of such a pair is zero.
- lacksquare Only one path μ in U_S which not paired.
- $extbf{0} \ \gamma(\mu,M) = w(S) w(M)$ (everything else has balance $extbf{0}$).

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- ② For $\pi \in U_S$ and $\pi' \in U_M...$
- - $\hbox{ If } \gamma(\pi,M) + \gamma(\pi',M) > 0 \hbox{ then } M \oplus \pi \oplus \pi' \hbox{ bigger weight than } M.$

- $\textbf{ If } \gamma(\pi,M) + \gamma(\pi',M) < 0 \text{ then } S \oplus \pi \oplus \pi' \text{ same number of edges as } S \text{ but heavier matching. A contradiction.}$
- lacktriangle Pair up the paths in U_S to paths in U_M .
- Total weight of such a pair is zero.
- lacksquare Only one path μ in U_S which not paired.
- $extbf{0} \ \gamma(\mu,M) = w(S) w(M)$ (everything else has balance $extbf{0}$).

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- ② For $\pi \in U_S$ and $\pi' \in U_M$...
- $lacksquare{3}$ Must be that $\gamma(\pi,M)+\gamma(\pi',M)=0$.
 - $\hbox{ If } \gamma(\pi,M) + \gamma(\pi',M) > 0 \hbox{ then } M \oplus \pi \oplus \pi' \hbox{ bigger weight than } M.$

- $\textbf{9} \ \ \text{If} \ \gamma(\pi,M) + \gamma(\pi',M) < 0 \ \text{then} \ S \oplus \pi \oplus \pi' \ \text{same number} \\ \text{of edges as} \ S \ \text{but heavier matching. A contradiction.}$
- lacktriangle Pair up the paths in U_S to paths in U_M .
- Total weight of such a pair is zero.
- lacksquare Only one path μ in U_S which not paired.
- $m{0} \ \ \gamma(\mu,M) = m{w}(S) m{w}(M)$ (everything else has balance $m{0}$).

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- ② For $\pi \in U_S$ and $\pi' \in U_M$...
- $lacksquare{3}$ Must be that $\gamma(\pi,M)+\gamma(\pi',M)=0$.
 - $\hbox{ If } \gamma(\pi,M) + \gamma(\pi',M) > 0 \hbox{ then } M \oplus \pi \oplus \pi' \hbox{ bigger weight than } M.$

- $\textbf{ If } \gamma(\pi,M) + \gamma(\pi',M) < 0 \text{ then } S \oplus \pi \oplus \pi' \text{ same number of edges as } S \text{ but heavier matching. A contradiction.}$
- lacksquare Pair up the paths in U_S to paths in U_M .
- Total weight of such a pair is zero.
- **1** Only one path μ in U_S which not paired.
- $extbf{0} \ \gamma(\mu,M) = w(S) w(M)$ (everything else has balance $extbf{0}$).
- \odot Path must be the heaviest augmenting path for M... Otherwise, \exists heavier augmenting path σ' for M s.t. $w(M \oplus \sigma') > w(S)$. A contradiction.

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- ② For $\pi \in U_S$ and $\pi' \in U_M$...
- $lacksquare{3}$ Must be that $\gamma(\pi,M)+\gamma(\pi',M)=0$.
 - $lack {f O}$ If $\gamma(\pi,M)+\gamma(\pi',M)>0$ then $M\oplus\pi\oplus\pi'$ bigger weight than M.

- ② If $\gamma(\pi,M) + \gamma(\pi',M) < 0$ then $S \oplus \pi \oplus \pi'$ same number of edges as S but heavier matching. A contradiction.
- lacktriangle Pair up the paths in U_S to paths in U_M .
- Total weight of such a pair is zero.
- **1** Only one path μ in U_S which not paired.
- lacksquare $\gamma(\mu,M)=w(S)-w(M)$ (everything else has balance 0).
- \odot Path must be the heaviest augmenting path for M... Otherwise, \exists heavier augmenting path σ' for M s.t. $w(M \oplus \sigma') > w(S)$. A contradiction.

- **1** Know: $|U_S| |U_M| = 1$ since |S| = |M| + 1.
- ② For $\pi \in U_S$ and $\pi' \in U_M...$
- ullet Must be that $\gamma(\pi,M)+\gamma(\pi',M)=0$.
 - $\hbox{ If } \gamma(\pi,M) + \gamma(\pi',M) > 0 \hbox{ then } M \oplus \pi \oplus \pi' \hbox{ bigger weight than } M.$

(With same number of edges.)

- $\textbf{9} \ \ \text{If} \ \gamma(\pi,M) + \gamma(\pi',M) < 0 \ \text{then} \ S \oplus \pi \oplus \pi' \ \text{same number} \\ \text{of edges as} \ S \ \text{but heavier matching.} \ \ \text{A contradiction}.$
- lacktriangle Pair up the paths in U_S to paths in U_M .
- Total weight of such a pair is zero.
- **1** Only one path μ in U_S which not paired.
- $lack \gamma(\mu,M)=w(S)-w(M)$ (everything else has balance 0).
- lacktriangledown Path must be the heaviest augmenting path for M... Otherwise, \exists heavier augmenting path σ' for M s.t. $w(M\oplus\sigma')>w(S)$. A contradiction.

Conclusion...

The above theorem imply that if we always augment along the maximum weight augmenting path, than we would get the maximum weight matching in the end.

17.2: Maximum weight matchings in a bipartite Graph

To be given a more exciting title...

- \bullet $G = (L \cup R, E)$: given bipartite graph.
- ② $w:E o\mathbb{R}$: non-negative weights on edges.
- M: matching.
- \bullet **G**_M: directed graph (like unweighted graph):
 - $oldsymbol{0}$ $rl \in M$, $l \in L$ and $r \in R$: add (r,l) to $oldsymbol{\mathsf{E}}(oldsymbol{\mathsf{G}}_M)$. Weight lpha((r,l)) = w(rl).
 - **2** $rl \in E \setminus M$: add edge $(l \to r) \in \mathsf{E}(\mathsf{G}_M)$. With weight lpha((l,r)) = -w(rl).
- **5** π : augmenting path in $\mathbf{G} = \pi$ path from free vertex in L to free vertex in R in \mathbf{G}_M .
- ullet path π in ${f G}_M$ has weight $lpha(\pi) = -\gamma(\pi,M)$.
- $\bigcirc U_L$: free vertices in L. U_R free vertices in R.
- ① Looking for: path π in G_M starting U_L going to U_R with maximum weight $\gamma(\pi)$. Min weight $\alpha(\pi)$.

To be given a more exciting title...

- \bullet $G = (L \cup R, E)$: given bipartite graph.
- $oldsymbol{v}: E
 ightarrow \mathbb{R}$: non-negative weights on edges.
- M: matching.
- \bullet G_M : directed graph (like unweighted graph):
 - $oldsymbol{0}$ $rl \in M$, $l \in L$ and $r \in R$: add (r,l) to $oldsymbol{\mathsf{E}}(oldsymbol{\mathsf{G}}_M)$. Weight lpha((r,l)) = w(rl).
 - $m{o}$ $rl \in E \setminus M$: add edge $(l o r) \in {f E}({f G}_M).$ With weight lpha((l,r)) = -w(rl).
- **5** π : augmenting path in $\mathbf{G} = \pi$ path from free vertex in L to free vertex in R in \mathbf{G}_M .
- lacksquare path π in lacksquare has weight $lpha(\pi) = -\gamma(\pi, M)$.
- $\bigcirc U_L$: free vertices in L. U_R free vertices in R.
- ① Looking for: path π in G_M starting U_L going to U_R with maximum weight $\gamma(\pi)$. Min weight $\alpha(\pi)$.

To be given a more exciting title...

- \bullet $G = (L \cup R, E)$: given bipartite graph.
- $oldsymbol{v}: E
 ightarrow \mathbb{R}$: non-negative weights on edges.
- M: matching.
- G_M : directed graph (like unweighted graph):
 - $oldsymbol{0} rl \in M$, $l \in L$ and $r \in R$: add (r,l) to $oldsymbol{\mathsf{E}}(oldsymbol{\mathsf{G}}_M)$. Weight lpha((r,l)) = w(rl).
 - $m{g} rl \in m{E} \setminus M$: add edge $(l o r) \in m{\mathsf{E}}(m{\mathsf{G}}_M).$ With weight lpha((l,r)) = -w(rl).
- **5** π : augmenting path in $\mathbf{G} = \pi$ path from free vertex in L to free vertex in R in \mathbf{G}_M .
- lacktriangledown path π in \mathbf{G}_M has weight $lpha(\pi) = -\gamma(\pi,M)$.
- 0 U_L : free vertices in L. U_R free vertices in R.
- ① Looking for: path π in G_M starting U_L going to U_R with maximum weight $\gamma(\pi)$. Min weight $\alpha(\pi)$.

- \bullet $G = (L \cup R, E)$: given bipartite graph.
- $oldsymbol{v}: E
 ightarrow \mathbb{R}$: non-negative weights on edges.
- M: matching.
- G_M : directed graph (like unweighted graph):
 - $oldsymbol{0} rl \in M$, $l \in L$ and $r \in R$: add (r,l) to $oldsymbol{\mathsf{E}}(oldsymbol{\mathsf{G}}_M)$. Weight lpha((r,l)) = w(rl).
 - ② $rl \in E \setminus M$: add edge $(l \to r) \in \mathsf{E}(\mathsf{G}_M)$. With weight lpha((l,r)) = -w(rl).
- $oldsymbol{\circ}$ π : augmenting path in $oldsymbol{\mathsf{G}}=\pi$ path from free vertex in $oldsymbol{L}$ to free vertex in $oldsymbol{R}$ in $oldsymbol{\mathsf{G}}_M$.
- $lacksquare{0}$ path $m{\pi}$ in $m{G}_M$ has weight $lpha(m{\pi}) = -\gamma(m{\pi},M)$.
- 0 U_L : free vertices in L. U_R free vertices in R.
- ① Looking for: path π in G_M starting U_L going to U_R with maximum weight $\gamma(\pi)$. Min weight $\alpha(\pi)$.

- \bullet $G = (L \cup R, E)$: given bipartite graph.
- $oldsymbol{v}: E
 ightarrow \mathbb{R}$: non-negative weights on edges.
- M: matching.
- **\bullet G_M**: directed graph (like unweighted graph):
 - $oldsymbol{0} rl \in M$, $l \in L$ and $r \in R$: add (r,l) to $oldsymbol{\mathsf{E}}(oldsymbol{\mathsf{G}}_M)$. Weight lpha((r,l)) = w(rl).
 - ② $rl \in E \setminus M$: add edge $(l \to r) \in \mathsf{E}(\mathsf{G}_M)$. With weight lpha((l,r)) = -w(rl).
- $oldsymbol{\circ}$ π : augmenting path in $oldsymbol{\mathsf{G}}=\pi$ path from free vertex in $oldsymbol{L}$ to free vertex in $oldsymbol{R}$ in $oldsymbol{\mathsf{G}}_M$.
- $oldsymbol{\circ}$ path $oldsymbol{\pi}$ in $oldsymbol{\mathsf{G}}_M$ has weight $lpha(oldsymbol{\pi}) = -\gamma(oldsymbol{\pi}, M)$.
- ${\color{red}0} \; U_L$: free vertices in $L.\; U_R$ free vertices in R.
- ① Looking for: path π in G_M starting U_L going to U_R with maximum weight $\gamma(\pi)$. Min weight $\alpha(\pi)$.

- \bullet $G = (L \cup R, E)$: given bipartite graph.
- $oldsymbol{v}: E
 ightarrow \mathbb{R}$: non-negative weights on edges.
- M: matching.
- **\bullet G_M**: directed graph (like unweighted graph):
 - $oldsymbol{0} rl \in M$, $l \in L$ and $r \in R$: add (r,l) to $oldsymbol{\mathsf{E}}(oldsymbol{\mathsf{G}}_M)$. Weight lpha((r,l)) = w(rl).
 - ② $rl \in E \setminus M$: add edge $(l \to r) \in \mathsf{E}(\mathsf{G}_M)$. With weight lpha((l,r)) = -w(rl).
- **3** π : augmenting path in $\mathbf{G} = \pi$ path from free vertex in L to free vertex in R in \mathbf{G}_M .
- $oldsymbol{\circ}$ path π in $oldsymbol{\mathsf{G}}_M$ has weight $lpha(\pi) = -\gamma(\pi,M)$.
- 0 U_L : free vertices in L. U_R free vertices in R.
- ① Looking for: path π in G_M starting U_L going to U_R with maximum weight $\gamma(\pi)$. Min weight $\alpha(\pi)$.

- **1** $G = (L \cup R, E)$: given bipartite graph.
- ② $w: E \to \mathbb{R}$: non-negative weights on edges.
- M: matching.
- **\bullet G_M**: directed graph (like unweighted graph):
 - $oldsymbol{0} rl \in M, \ l \in L \ ext{and} \ r \in R$: add (r,l) to $oldsymbol{\mathsf{E}}(oldsymbol{\mathsf{G}}_M).$ Weight lpha((r,l)) = w(rl).
 - **2** $rl \in E \setminus M$: add edge $(l \to r) \in \mathsf{E}(\mathsf{G}_M)$. With weight $\alpha((l,r)) = -w(rl)$.
- $oldsymbol{\circ}$ π : augmenting path in $oldsymbol{\mathsf{G}}=\pi$ path from free vertex in $oldsymbol{L}$ to free vertex in $oldsymbol{R}$ in $oldsymbol{\mathsf{G}}_M$.
- $lackbox{0}$ path π in $lackbox{G}_M$ has weight $lpha(\pi) = -\gamma(\pi,M)$.
- ① Looking for: path π in G_M starting U_L going to U_R with maximum weight $\gamma(\pi)$. Min weight $\alpha(\pi)$.

- **1** $G = (L \cup R, E)$: given bipartite graph.
- $oldsymbol{v}: E
 ightarrow \mathbb{R}$: non-negative weights on edges.
- M: matching.
- **\bullet G_M**: directed graph (like unweighted graph):
 - $oldsymbol{0} rl \in M$, $l \in L$ and $r \in R$: add (r,l) to $oldsymbol{\mathsf{E}}(oldsymbol{\mathsf{G}}_M)$. Weight lpha((r,l)) = w(rl).
 - $m{Q}$ $rl \in E \setminus M$: add edge $(l o r) \in {\sf E}({\sf G}_M)$. With weight lpha((l,r)) = -w(rl).
- lacktriangledown π : augmenting path in lacktriangledown path from free vertex in $oldsymbol{L}$ to free vertex in $oldsymbol{R}$ in $oldsymbol{G}_M$.
- $oldsymbol{0}$ path π in $oldsymbol{\mathsf{G}}_M$ has weight $lpha(\pi) = -\gamma(\pi,M)$.
- **0** U_L : free vertices in L. U_R free vertices in R.
- **1** Looking for: path π in G_M starting U_L going to U_R with maximum weight $\gamma(\pi)$. Min weight $\alpha(\pi)$.

No negative cycles for max weight matching

Lemma

If M is a maximum weight matching with k edges in G, than there is no negative cycle in G_M where $\alpha(\cdot)$ is the associated weight function.

Proof.

Assume for the sake of contradiction that there is a cycle C, and observe that $\gamma(C) = -\alpha(C) > 0$. Namely, $M \oplus C$ is a new matching with bigger weight and the same number of edges. A contradiction to the maximality of M.

Sariel (UIUC) New CS473 14 Fall 2015 14 / 48 $17.2.1: \ \, \mathsf{The\ algorithm}.$

- Compute a maximum weight in the bipartite graph G as follows:
 - floor Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- $oxed{f @}$ Compute a minimum weight path in ${f G}_M$ between U_L and U_R .
- **Shortest path in G_M with no negative cycles (but negative weights on edges).**
- Use Bellman-Ford algorithm
 - ① Collapse all free vertices of U_L into a single vertex.
 - @ Collapse all free vertices of U_R into a single vertex.
 - \bullet H_M : resulting graph.
 - $ext{ 0}$ Compute shortest path from U_L to U_R in H_M .
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

- Compute a maximum weight in the bipartite graph G as follows:
 - Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- ② Compute a minimum weight path in ${f G}_M$ between U_L and U_R .
- 3 Shortest path in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm.
 - lacksquare Collapse all free vertices of U_L into a single vertex.
 - ullet Collapse all free vertices of U_R into a single vertex.
 - \bullet H_M : resulting graph.
 - lacksquare Compute shortest path from U_L to U_R in H_M .
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

- Compute a maximum weight in the bipartite graph G as follows:
 - $lackbox{0}$ Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- **②** Compute a minimum weight path in \mathbf{G}_M between U_L and U_R .
- 3 Shortest path in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm
 - lacksquare Collapse all free vertices of U_L into a single vertex.
 - ullet Collapse all free vertices of U_R into a single vertex.
 - \bullet H_M : resulting graph.
 - lacksquare Compute shortest path from U_L to U_R in H_M .
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

- Compute a maximum weight in the bipartite graph G as follows:
 - lacktriangledown Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- **②** Compute a minimum weight path in \mathbf{G}_M between U_L and U_R .
- **Shortest path** in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm
 - lacksquare Collapse all free vertices of U_L into a single vertex.
 - ullet Collapse all free vertices of U_R into a single vertex.
 - \bullet H_M : resulting graph.
 - lacksquare Compute shortest path from U_L to U_R in H_M .
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

- Compute a maximum weight in the bipartite graph G as follows:
 - $lackbox{0}$ Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- **②** Compute a minimum weight path in \mathbf{G}_M between U_L and U_R .
- **Shortest path** in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm.
 - ① Collapse all free vertices of U_L into a single vertex.
 - @ Collapse all free vertices of U_R into a single vertex.
 - \bullet H_M : resulting graph.
 - lacksquare Compute shortest path from U_L to U_R in H_M .
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

- Compute a maximum weight in the bipartite graph G as follows:
 - $lackbox{0}$ Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- **②** Compute a minimum weight path in \mathbf{G}_M between U_L and U_R .
- **Shortest path** in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm.
 - lacktriangle Collapse all free vertices of U_L into a single vertex.
 - @ Collapse all free vertices of U_R into a single vertex.
 - \bullet H_M : resulting graph.
 - $ext{ @ Compute shortest path from } U_L ext{ to } U_R ext{ in } H_M.$
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

- Compute a maximum weight in the bipartite graph G as follows:
 - $lackbox{0}$ Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- **②** Compute a minimum weight path in \mathbf{G}_M between U_L and U_R .
- **Shortest path** in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm.
 - lacktriangle Collapse all free vertices of U_L into a single vertex.
 - **2** Collapse all free vertices of U_R into a single vertex.
 - \bullet H_M : resulting graph.
 - $ext{ @ Compute shortest path from } U_L ext{ to } U_R ext{ in } H_M.$
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

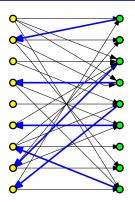
- Compute a maximum weight in the bipartite graph G as follows:
 - $lackbox{0}$ Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- **②** Compute a minimum weight path in \mathbf{G}_M between U_L and U_R .
- **Shortest path** in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm.
 - lacktriangledown Collapse all free vertices of U_L into a single vertex.
 - **2** Collapse all free vertices of U_R into a single vertex.
 - \bullet H_M : resulting graph.
 - $extbf{0}$ Compute shortest path from U_L to U_R in H_M .
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

- Compute a maximum weight in the bipartite graph G as follows:
 - $lackbox{0}$ Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- **②** Compute a minimum weight path in \mathbf{G}_M between U_L and U_R .
- **Shortest path** in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm.
 - lacktriangle Collapse all free vertices of U_L into a single vertex.
 - $oldsymbol{\circ}$ Collapse all free vertices of U_R into a single vertex.
 - **3** H_M : resulting graph.
 - **4** Compute shortest path from U_L to U_R in H_M .
 - $oldsymbol{0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

- Compute a maximum weight in the bipartite graph G as follows:
 - $lackbox{0}$ Find a maximum weight matching M with k edges, compute the maximum weight augmenting path for M, apply it, and repeat till M is maximal.
- ② Compute a minimum weight path in ${\sf G}_M$ between U_L and U_R .
- **Shortest path** in G_M with no negative cycles (but negative weights on edges).
- Use Bellman-Ford algorithm.
 - lacksquare Collapse all free vertices of U_L into a single vertex.
 - $oldsymbol{\circ}$ Collapse all free vertices of U_R into a single vertex.
 - **3** H_M : resulting graph.
 - **4** Compute shortest path from U_L to U_R in H_M .
 - $footnote{f 0}$ since no negative cycles. **Bellman-Ford** algorithm works in O(nm) time.

Sariel (UIUC) New CS473 16 Fall 2015 16 / 48

A figure...



Result

Result:

Lemma

Given a bipartite graph ${\bf G}$ and a maximum weight matching ${\bf M}$ of size ${\bf k}$ one can find a maximum weight augmenting path for ${\bf G}$ in ${\bf O}(n{\bf m})$ time, where ${\bf n}$ is the number of vertices of ${\bf G}$ and ${\bf m}$ is the number of edges.

② Applying this algorithm n/2 times at most:

Theorem

Given a weight bipartite graph ${\sf G}$, with n vertices and m edges, one can compute a maximum weight matching in ${\sf G}$ in $O(n^2m)$ time.

Result

Result:

Lemma

Given a bipartite graph ${\bf G}$ and a maximum weight matching ${\bf M}$ of size ${\bf k}$ one can find a maximum weight augmenting path for ${\bf G}$ in ${\bf O}(n{\bf m})$ time, where ${\bf n}$ is the number of vertices of ${\bf G}$ and ${\bf m}$ is the number of edges.

② Applying this algorithm n/2 times at most:

Theorem

Given a weight bipartite graph ${\bf G}$, with ${\bf n}$ vertices and ${\bf m}$ edges, one can compute a maximum weight matching in ${\bf G}$ in $O(n^2m)$ time.

Faster algorithm...

Working harder, one can get a faster algorithm. We state the result without proof:

Theorem

Given a weight bipartite graph ${\bf G}$, with n vertices and m edges, one can compute a maximum weight matching in ${\bf G}$ in $O(n(n\log n + m))$ time.

17.2.1.1: The Bellman-Ford Algorithm - A Quick Reminder

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- lacktriangledown Init: $orall u \in \mathsf{V}(\mathsf{G})$: $d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- \bullet Repeat n times:
 - scan all the edges.
 - \triangledown $\forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - ullet relax(u,v): if x=d[u]+w((u,v))< d[v], set d[v] to x
 - $oldsymbol{0}$ $oldsymbol{d}[oldsymbol{u}]$: current distance from $oldsymbol{s}$ to $oldsymbol{u}$.
- **o** Overall running time is O(mn).
- lacktriangledown Claim: in end of exec- shortest path length from s to u is d[u].
- By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- ullet Init: $\forall u \in \mathsf{V}(\mathsf{G}) \colon d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- \bullet Repeat n times:
 - scan all the edges.
 - $\triangledown \forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - ullet relax(u,v): if x=d[u]+w((u,v))< d[v], set d[v] to a
 - $oldsymbol{0}$ $oldsymbol{d}[u]$: current distance from s to u
- **o** Overall running time is O(mn).
- lacktriangledown Claim: in end of exec- shortest path length from s to u is d[u].
- @ By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- $\textbf{ Init: } \forall u \in \mathbf{V}(\mathbf{G}) \text{: } d[u] \leftarrow \infty \text{ and } d[s] \leftarrow 0.$
- \bullet Repeat n times:
 - scan all the edges.
 - $\forall (u,v) \in \mathsf{E}(\mathsf{G}) \text{ it performs a } \mathsf{Relax}(u,v) \text{ operation.}$
 - $oxed{o}$ relax(u,v): if x=d[u]+w((u,v))< d[v], set d[v] to x
 - **a** d[u]: current distance from s to u.
- **o** Overall running time is O(mn).
- lacktriangledown Claim: in end of exec- shortest path length from s to u is d[u].
- By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- $\textbf{ Init: } \forall u \in \mathbf{V}(\mathbf{G}) \text{: } d[u] \leftarrow \infty \text{ and } d[s] \leftarrow 0.$
- lacktriangle Repeat n times:
 - scan all the edges.
 - $\triangledown \forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - $oxed{\mathbf{g}}$ relax(u,v): if x=d[u]+w((u,v))< d[v], set d[v] to x
 - **1** d[u]: current distance from s to u.
- **o** Overall running time is O(mn).
- lacktriangledown Claim: in end of exec- shortest path length from s to u is d[u].
- By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- **3** Init: $\forall u \in V(G)$: $d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- Repeat n times:
 - scan all the edges.
 - $\triangledown \forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - ullet relax(u,v): if x=d[u]+w((u,v))< d[v], set d[v] to x
- **5** Overall running time is O(mn).
- lacktriangledown Claim: in end of exec- shortest path length from s to u is d[u].
- By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- **3** Init: $\forall u \in V(G)$: $d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- Repeat n times:
 - scan all the edges.
 - $\forall (u,v) \in \mathsf{E}(\mathsf{G}) \text{ it performs a } \mathsf{Relax}(u,v) \text{ operation.}$
 - $lacksquare ext{relax}(u,v)$: if x=d[u]+w((u,v)) < d[v], set d[v] to a
- **5** Overall running time is O(mn).
- ullet Claim: in end of exec- shortest path length from s to u is d[u]
- @ By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- \bullet Init: $\forall u \in V(G)$: $d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- lacktriangle Repeat n times:
 - scan all the edges.
 - $\forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - ullet relax(u,v): if x=d[u]+w((u,v)) < d[v], set d[v] to x
 - $oldsymbol{0}$ $oldsymbol{d}[oldsymbol{u}]$: current distance from s to $oldsymbol{u}$
- 6 Overall running time is O(mn).
- ullet Claim: in end of exec- shortest path length from s to u is d[u]
- @ By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- $\textbf{ Init: } \forall u \in \mathbf{V}(\mathbf{G}) \text{: } d[u] \leftarrow \infty \text{ and } d[s] \leftarrow 0.$
- lacktriangle Repeat n times:
 - scan all the edges.
 - $\forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - $\bullet \ \operatorname{relax}(u,v) \colon \operatorname{if} \ x = d[u] + w((u,v)) < d[v], \ \operatorname{set} \ d[v] \ \operatorname{to} \ x$
 - $\mathbf{0}$ d[u]: current distance from s to u.
- Overall running time is O(mn)
- lacktriangledown Claim: in end of exec- shortest path length from s to u is d[u].
- @ By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- **3** Init: $\forall u \in V(G)$: $d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- Repeat n times:
 - scan all the edges.
 - $\forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - $\qquad \qquad \text{relax}(u,v) \colon \text{if } x = d[u] + w((u,v)) < d[v] \text{, set } d[v] \text{ to } x$
 - $\mathbf{0}$ d[u]: current distance from s to u.
- **5** Overall running time is O(mn).
- ullet Claim: in end of exec- shortest path length from s to u is d[u]
- \odot By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- \bullet Init: $\forall u \in V(G)$: $d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- lacktriangle Repeat n times:
 - scan all the edges.
 - $\forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - $\qquad \qquad \text{relax}(u,v) \colon \text{if } x = d[u] + w((u,v)) < d[v] \text{, set } d[v] \text{ to } x$
 - $\mathbf{0}$ d[u]: current distance from s to u.
- **5** Overall running time is O(mn).
- **o** Claim: in end of exec- shortest path length from s to u is d[u].
- By induction: All vertices with shortest path to with edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- **3** Init: $\forall u \in V(G)$: $d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- lacktriangle Repeat n times:
 - scan all the edges.
 - $\forall (u,v) \in \mathsf{E}(\mathsf{G}) \text{ it performs a } \mathsf{Relax}(u,v) \text{ operation.}$
 - $\qquad \qquad \text{relax}(u,v) \colon \text{if } x = d[u] + w((u,v)) < d[v] \text{, set } d[v] \text{ to } x$
 - $\mathbf{0}$ d[u]: current distance from s to u.
- **5** Overall running time is O(mn).
- lacktriangledown Claim: in end of exec- shortest path length from s to u is d[u].
- $oldsymbol{\circ}$ By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

- Bellman-Ford computes shortest path from a single source s in a graph G.
- Assumption: no negative cycles (but weights can be negative).
- \bullet Init: $\forall u \in V(G)$: $d[u] \leftarrow \infty$ and $d[s] \leftarrow 0$.
- lacktriangle Repeat n times:
 - scan all the edges.
 - $\forall (u,v) \in \mathsf{E}(\mathsf{G})$ it performs a $\mathsf{Relax}(u,v)$ operation.
 - $\bullet \ \operatorname{relax}(u,v) \colon \operatorname{if} \ x = d[u] + w((u,v)) < d[v], \ \operatorname{set} \ d[v] \ \operatorname{to} \ x$
 - $\mathbf{0}$ d[u]: current distance from s to u.
- **5** Overall running time is O(mn).
- lacktriangledown Claim: in end of exec- shortest path length from s to u is d[u].
- ullet By induction: All vertices with shortest path to s with i edges, are being set to their shortest path length in the ith iteration
- Can modify to detect negative cycles.

17.3: Maximum Size Matching in a Non-Bipartite Graph

Non-bipartite matching...

- Graph not bipartite. No weights on edges.
- ② Start from an empty matching M
- o repeatedly find an augmenting path from an unmatched vertex to an unmatched vertex.

Non-bipartite matching...

- Graph not bipartite. No weights on edges.
- $oldsymbol{0}$ Start from an empty matching $oldsymbol{M}$
- o repeatedly find an augmenting path from an unmatched vertex to an unmatched vertex.

Non-bipartite matching...

- Graph not bipartite. No weights on edges.
- $oldsymbol{0}$ Start from an empty matching $oldsymbol{M}$
- repeatedly find an augmenting path from an unmatched vertex to an unmatched vertex.

- ① T: a given tree
- ② For two vertices $x,y\in V(\mathfrak{I})$: au_{xy} denote the path in \mathfrak{I} between x and y.
- \odot For two paths π and π' that share an endpoint.
- $\bigcirc \pi \mid \mid \pi'$ concatenated path
- $|\pi|$ denote the number of edges in π .

- $oldsymbol{\mathfrak{I}}$: a given tree.
- ② For two vertices $x,y\in V(\mathfrak{I})$: au_{xy} denote the path in \mathfrak{I} between x and y.
- \odot For two paths π and π' that share an endpoint.
- $\bigcirc \pi \mid \mid \pi'$ concatenated path
- $|\pi|$ denote the number of edges in π .

- T: a given tree.
- ② For two vertices $x,y \in V(\mathfrak{I})$: au_{xy} denote the path in \mathfrak{I} between x and y.
- \odot For two paths π and π' that share an endpoint.
- $|\pi|$ denote the number of edges in π .

- $oldsymbol{\mathfrak{I}}$: a given tree.
- ② For two vertices $x,y \in V(\mathfrak{I})$: au_{xy} denote the path in \mathfrak{I} between x and y.
- **3** For two paths π and π' that share an endpoint.
- $\bigcirc \pi \mid \mid \pi'$ concatenated path
- $|\pi|$ denote the number of edges in π .

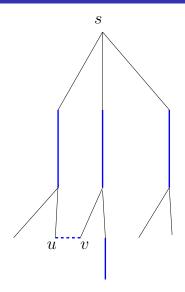
- T: a given tree.
- ② For two vertices $x,y \in V(\mathfrak{I})$: au_{xy} denote the path in \mathfrak{I} between x and y.
- **3** For two paths π and π' that share an endpoint.
- $|\pi|$ denote the number of edges in π .

- T: a given tree.
- ② For two vertices $x,y \in V(\mathfrak{I})$: au_{xy} denote the path in \mathfrak{I} between x and y.
- **3** For two paths π and π' that share an endpoint.
- $|\pi|$ denote the number of edges in π .

17.3.1: Finding an augmenting path

A figure

A cycle in the alternating BFS tree.



- **1 G**: graph. **M**: matching.
- Task: compute bigger matching in G.
- ullet Compute an augmenting path for M
- 4 Add edges that are both endpoints free to matching
- Sassume ∀ edges at least one of their endpoint adjacent to matching edge.
- \odot Collapse unmatched vertices to single vertex s.
- # : resulting graph.
- $lacktriang{f 0}$ compute an **alternating BFS** of m H starting from m s.
- ullet BFS on H from s.
 - even levels of **BFS** tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - Output
 Let T denote the resulting tree.

- **1 G**: graph. **M**: matching.
- Task: compute bigger matching in G.
- ullet Compute an augmenting path for M.
- Add edges that are both endpoints free to matching.
- Sassume ∀ edges at least one of their endpoint adjacent to matching edge.
- \odot Collapse unmatched vertices to single vertex s.
- O H: resulting graph.
- lacksquare compute an **alternating BFS** of $m{H}$ starting from $m{s}$.
- \bigcirc **BFS** on H from s.
 - even levels of **BFS** tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - Set T denote the resulting tree.
- Augmenting path in **G** corresponds to an odd cycle in H passing through s.

- $oldsymbol{0}$ $oldsymbol{G}$: graph. $oldsymbol{M}$: matching.
- Task: compute bigger matching in G.
- ullet Compute an augmenting path for M.
- Add edges that are both endpoints free to matching.
- Sassume ∀ edges at least one of their endpoint adjacent to matching edge.
- \odot Collapse unmatched vertices to single vertex s.
- H: resulting graph.
- $oldsymbol{0}$ compute an **alternating BFS** of $oldsymbol{H}$ starting from $oldsymbol{s}.$
- ullet BFS on H from s.
 - even levels of **BFS** tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - \odot Let \Im denote the resulting tree.
- Augmenting path in **G** corresponds to an odd cycle in H passing through s.

- $oldsymbol{0}$ $oldsymbol{G}$: graph. $oldsymbol{M}$: matching.
- Task: compute bigger matching in G.
- ullet Compute an augmenting path for M.
- Add edges that are both endpoints free to matching.
- Salar Street Assume ∀ edges at least one of their endpoint adjacent to matching edge.
- \odot Collapse unmatched vertices to single vertex s.
- H: resulting graph.
- $lacktriang{f 0}$ compute an **alternating BFS** of m H starting from m s.
- \bigcirc **BFS** on H from s.
 - even levels of **BFS** tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - \odot Let \Im denote the resulting tree.
- Augmenting path in **G** corresponds to an odd cycle in H passing through s.

- $oldsymbol{0}$ $oldsymbol{G}$: graph. $oldsymbol{M}$: matching.
- Task: compute bigger matching in G.
- lacktriangle Compute an augmenting path for $m{M}$.
- Add edges that are both endpoints free to matching.
- Salar Assume ∀ edges at least one of their endpoint adjacent to matching edge.
- **o** Collapse unmatched vertices to single vertex s.
- M : resulting graph.
- $lacktriang{f 0}$ compute an **alternating BFS** of m H starting from m s.
- lacksquare BFS on H from s.
 - even levels of **BFS** tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - \odot Let \Im denote the resulting tree.
- Augmenting path in **G** corresponds to an odd cycle in H passing through s.

- **1 G**: graph. **M**: matching.
- Task: compute bigger matching in G.
- ullet Compute an augmenting path for M.
- Add edges that are both endpoints free to matching.
- Salar Assume ∀ edges at least one of their endpoint adjacent to matching edge.
- **o** Collapse unmatched vertices to single vertex s.
- $\mathbf{0}$ \mathbf{H} : resulting graph.
- $lacktriang{f 0}$ compute an **alternating BFS** of m H starting from m s.
- \bigcirc **BFS** on H from s.
 - even levels of **BFS** tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - \odot Let \Im denote the resulting tree.
- Augmenting path in **G** corresponds to an odd cycle in H passing through s.

- $oldsymbol{0}$ $oldsymbol{G}$: graph. $oldsymbol{M}$: matching.
- Task: compute bigger matching in G.
- ullet Compute an augmenting path for $oldsymbol{M}$.
- Add edges that are both endpoints free to matching.
- Solution State Assume ∀ edges at least one of their endpoint adjacent to matching edge.
- **o** Collapse unmatched vertices to single vertex s.
- \bigcirc H: resulting graph.
- $lacktriang{0}$ compute an **alternating BFS** of $m{H}$ starting from $m{s}$.
- \bigcirc **BFS** on H from s.
 - even levels of BFS tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - \odot Let \Im denote the resulting tree.
- Augmenting path in **G** corresponds to an odd cycle in H passing through s.

- $oldsymbol{0}$ $oldsymbol{G}$: graph. $oldsymbol{M}$: matching.
- Task: compute bigger matching in G.
- lacktriangle Compute an augmenting path for $m{M}$.
- Add edges that are both endpoints free to matching.
- Solution State Assume ∀ edges at least one of their endpoint adjacent to matching edge.
- **o** Collapse unmatched vertices to single vertex s.
- \bigcirc H: resulting graph.
- $lacktriang{0}$ compute an **alternating BFS** of $m{H}$ starting from $m{s}$.
- lacktriangle BFS on $oldsymbol{H}$ from $oldsymbol{s}$.
 - even levels of BFS tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - \odot Let \Im denote the resulting tree.
- lacktriangledown Augmenting path in lacktriangledown corresponds to an odd cycle in lacktriangledown passing through lacktriangledown.

- **1 G**: graph. **M**: matching.
- Task: compute bigger matching in G.
- lacktriangle Compute an augmenting path for $m{M}$.
- Add edges that are both endpoints free to matching.
- Second Assume ∀ edges at least one of their endpoint adjacent to matching edge.
- **o** Collapse unmatched vertices to single vertex s.
- H : resulting graph.
- $lacktriang{0}$ compute an **alternating BFS** of $m{H}$ starting from $m{s}$.
- **9 BFS** on H from s.
 - even levels of BFS tree use only matching edges.
 - odd levels BFS tree: only free edges.
 - Set T denote the resulting tree.
- lacktriangledown Augmenting path in lacktriangledown corresponds to an odd cycle in lacktriangledown passing through s.

Like a bridge over troubled matching...

Definition

An edge $uv \in E(G)$ is a **bridge** if the following conditions are met:

- (i) \boldsymbol{u} and \boldsymbol{v} have the same depth in $\boldsymbol{\Im}$,
- (ii) if the depth of u in ${\mathfrak T}$ is even then uv is free (i.e., $uv \notin M$), and
- (iii) if the depth of u in $\mathfrak T$ is odd then $uv \in M$.

Finding odd cycles...

- lacksquare given an edge uv... can check if it is a bridge in constant time.
- We need the following:

Lemma

Let v be a vertex of G, M a matching in G, and let π be the shortest alternating path between s and v in G. Furthermore, assume that for any vertex w of π the shortest alternating path between w and s is the path along π .

Then, the depth $d_{\mathfrak{I}}(v)$ of v in \mathfrak{I} is $|\pi|$.

Finding odd cycles...

- lacktriangledown given an edge uv... can check if it is a bridge in constant time.
- We need the following:

Lemma

Let v be a vertex of G, M a matching in G, and let π be the shortest alternating path between s and v in G. Furthermore, assume that for any vertex w of π the shortest alternating path between w and s is the path along π .

Finding odd cycles...

- lacktriangledown given an edge uv... can check if it is a bridge in constant time.
- We need the following:

Lemma

Let v be a vertex of \mathbf{G} , M a matching in \mathbf{G} , and let π be the shortest alternating path between s and v in \mathbf{G} . Furthermore, assume that for any vertex w of π the shortest alternating path between w and s is the path along π .

Then, the depth $d_{\mathfrak{T}}(v)$ of v in \mathfrak{T} is $|\pi|$.

- 1 Induction on $|\pi|$. For $|\pi|=1$: easy... v is a neighbor of s in



- 1 Induction on $|\pi|$. For $|\pi|=1$: easy... v is a neighbor of s in G...v child of s in BFS tree \mathcal{T} .



Proof.

- 1 Induction on $|\pi|$. For $|\pi|=1$: easy... v is a neighbor of s in \mathbf{G}_{\cdots} v child of s in **BFS** tree \mathfrak{T}_{\cdot}



Fall 2015

Proof.

- 1 Induction on $|\pi|$. For $|\pi| = 1$: easy... v is a neighbor of s in G... v child of s in BFS tree \mathfrak{T} .
- $|\pi| = k$. u: vertex just before v on π .
- ullet By induction, depth of $oldsymbol{u}$ in $oldsymbol{\mathfrak{T}}$ is k-1.
- When alternating BFS algorithm visited u: tried hang v from u...
- ullet failure only if $oldsymbol{v}$ already in $oldsymbol{\mathfrak{I}}$.
- $exttt{ o} \implies ext{exists a shorter alternating path from <math>s$ to v
- A contradiction.



30 / 48

- 1 Induction on $|\pi|$. For $|\pi| = 1$: easy... v is a neighbor of s in G... v child of s in BFS tree \mathcal{T} .
- $|\pi|=k$. u: vertex just before v on π .
- $oldsymbol{0}$ By induction, depth of $oldsymbol{u}$ in $oldsymbol{\mathfrak{T}}$ is $oldsymbol{k}-1$.
- When alternating BFS algorithm visited u: tried hang v from u...
- ullet failure only if v already in ${\mathfrak T}.$
- $exttt{ } exttt{ } ext$
- A contradiction.



Proof.

- 1 Induction on $|\pi|$. For $|\pi| = 1$: easy... v is a neighbor of s in G... v child of s in BFS tree \mathcal{T} .
- $|\pi|=k$. u: vertex just before v on π .
- $oldsymbol{3}$ By induction, depth of $oldsymbol{u}$ in $oldsymbol{\mathfrak{T}}$ is $oldsymbol{k}-1$.
- 4 When alternating BFS algorithm visited u: tried hang v from u...
- $ilde{ to}$ failure only if $oldsymbol{v}$ already in $oldsymbol{\mathfrak{I}}.$
- $exttt{ } exttt{ } ext$
- A contradiction.



30 / 48

- 1 Induction on $|\pi|$. For $|\pi| = 1$: easy... v is a neighbor of s in G... v child of s in BFS tree \mathfrak{T} .
- $|\pi| = k$. u: vertex just before v on π .
- $oldsymbol{3}$ By induction, depth of $oldsymbol{u}$ in $oldsymbol{\Im}$ is $oldsymbol{k}-1$.
- 4 When alternating BFS algorithm visited u: tried hang v from u...
- $oldsymbol{\mathfrak{o}}$ failure only if $oldsymbol{v}$ already in $oldsymbol{\mathfrak{T}}$.
- $exttt{ o} \implies$ exists a shorter alternating path from s to v
- A contradiction.



- 1 Induction on $|\pi|$. For $|\pi|=1$: easy... v is a neighbor of s in **G** v child of s in **BFS** tree T
- $|\pi| = k$. u: vertex just before v on π .
- **3** By induction, depth of u in \mathfrak{T} is k-1.
- When alternating **BFS** algorithm visited u: tried hang v from 11....
- ullet failure only if $oldsymbol{v}$ already in $oldsymbol{\mathfrak{I}}$.
- \bullet exists a shorter alternating path from s to v



- 1 Induction on $|\pi|$. For $|\pi| = 1$: easy... v is a neighbor of s in G... v child of s in BFS tree \mathcal{T} .
- $|\pi|=k$. u: vertex just before v on π .
- **3** By induction, depth of u in \mathfrak{T} is k-1.
- 4 When alternating BFS algorithm visited u: tried hang v from u...
- ullet failure only if v already in ${\mathfrak T}$.
- lacktriangledown exists a shorter alternating path from s to v
- A contradiction.



If there is an augmenting path...

.., then there is a bridge

Lemma

If there is an augmenting path in ${\bf G}$ for a matching ${\bf M}$, then there exists an edge ${\bf u}{\bf v}\in E(G)$ which is a bridge in ${\mathfrak T}.$

- **1** π : an augmenting path in **G**.
- @ π : odd length alternating cycle in H.
- \circ shortest odd length alternating cycle in \circ going through s.
- \odot both edges in σ adjacent to s are unmatched.
- $x \in V(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- lacktriangledown d'(x) len shortest alternating path between s and x along σ .
- O Clearly: $d(x) \leq d'(x)$.
- ullet Claim: d(x)=d'(x), for all $x\in \sigma$. (See next slide for proof.)
- ① Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- $exttt{@}$ By previous lemma: $d_{\mathfrak{T}}(u)=d(u)=d(v)=d_{\mathfrak{T}}(v).$ Found bridge!
- oximes Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- \circ π : odd length alternating cycle in H.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in V(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- lacktriangledown d'(x) len shortest alternating path between s and x along σ .
- \bigcirc Clearly: $d(x) \leq d'(x)$.
- ullet Claim: d(x)=d'(x), for all $x\in \sigma$. (See next slide for proof.)
- ① Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- $exttt{@}$ By previous lemma: $d_{\mathfrak{T}}(u)=d(u)=d(v)=d_{\mathfrak{T}}(v).$ Found bridge!
- \bigcirc Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- \circ π : odd length alternating cycle in H.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $x \in V(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- lacktriangledown d'(x) len shortest alternating path between s and x along σ .
- O Clearly: $d(x) \leq d'(x)$.
- ullet Claim: d(x)=d'(x), for all $x\in \sigma$. (See next slide for proof.)
- ① Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- @ Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- $oldsymbol{0}$ both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in V(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- ullet d'(x) len shortest alternating path between s and x along $oldsymbol{\sigma}$.
- O Clearly: $d(x) \leq d'(x)$.
- ullet Claim: d(x)=d'(x), for all $x\in \sigma$. (See next slide for proof.)
- ① Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- \bigcirc Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- \bullet $x \in V(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- lacktriangledown d'(x) len shortest alternating path between s and x along σ .
- \bigcirc Clearly: $d(x) \leq d'(x)$.
- ullet Claim: d(x)=d'(x), for all $x\in \sigma$. (See next slide for proof.)
- ① Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- \bigcirc Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in \mathbf{V}(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- **1** d'(x) len shortest alternating path between s and x along σ .
- $m{0}$ Clearly: $d(x) \leq d'(x)$.
- $exttt{ iny Claim: } d(x) = d'(x)$, for all $x \in \pmb{\sigma}$. (See next slide for proof.)
- ① Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- @ Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- \circ π : odd length alternating cycle in H.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in \mathbf{V}(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- **1** d'(x) len shortest alternating path between s and x along σ .
- Clearly: $d(x) \leq d'(x)$.
- ullet Claim: d(x)=d'(x), for all $x\in \sigma$. (See next slide for proof.)
- ullet Take two vertices of $oldsymbol{\sigma}$ furthest away from $oldsymbol{s}$.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- @ Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- \circ π : odd length alternating cycle in H.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in \mathbf{V}(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- **1** d'(x) len shortest alternating path between s and x along σ .
- lacktriangledown Clearly: $d(x) \leq d'(x)$.
- lacktriangledown Claim: d(x)=d'(x), for all $x\in\sigma$. (See next slide for proof.)
- ① Take two vertices of σ furthest away from s.
- Both have same depth in \mathcal{T} , since d(u) = d'(u) = d'(v) = d(v).
- @ Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in \mathbf{V}(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- **1** d'(x) len shortest alternating path between s and x along σ .
- Clearly: $d(x) \leq d'(x)$.
- lacktriangledown Claim: d(x)=d'(x), for all $x\in\sigma$. (See next slide for proof.)
- **9** Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- @ Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- \circ π : odd length alternating cycle in H.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in \mathbf{V}(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- **1** d'(x) len shortest alternating path between s and x along σ .
- Clearly: $d(x) \leq d'(x)$.
- lacktriangledown Claim: d(x)=d'(x), for all $x\in\sigma$. (See next slide for proof.)
- **9** Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{T} , since d(u) = d'(u) = d'(v) = d(v).
- @ Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- \circ π : odd length alternating cycle in H.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in \mathbf{V}(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- **1** d'(x) len shortest alternating path between s and x along σ .
- Clearly: $d(x) \leq d'(x)$.
- lacktriangledown Claim: d(x)=d'(x), for all $x\in\sigma$. (See next slide for proof.)
- **9** Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- $exttt{@}$ By previous lemma: $d_{\mathfrak{I}}(u)=d(u)=d(v)=d_{\mathfrak{I}}(v).$ Found bridge!
 - $^{ t D}$ Observe: σ is created from an alternating path.

- **1** π : an augmenting path in **G**.
- $oldsymbol{\circ}$ shortest odd length alternating cycle in $oldsymbol{\mathsf{G}}$ going through s.
- ullet both edges in σ adjacent to s are unmatched.
- $oldsymbol{s} x \in V(\sigma)$: d(x) length of shortest alternating path between x and s in H.
- **1** d'(x) len shortest alternating path between s and x along σ .
- O Clearly: $d(x) \leq d'(x)$.
- lacktriangledown Claim: d(x)=d'(x), for all $x\in\sigma$. (See next slide for proof.)
- **9** Take two vertices of σ furthest away from s.
- Both have same depth in \mathfrak{I} , since d(u) = d'(u) = d'(v) = d(v).
- $exttt{@}$ By previous lemma: $d_{\mathfrak{I}}(u)=d(u)=d(v)=d_{\mathfrak{I}}(v).$ Found bridge!
- $oldsymbol{arphi}$ Observe: $oldsymbol{\sigma}$ is created from an alternating path.

Claim: d(x) = d'(x), for all $x \in \sigma$.

- ① assume for contradiction: d(x) < d'(x).
- ② π_1, π_2 : paths from x to s formed by σ .
- $oldsymbol{0}$ $oldsymbol{\eta}$: shortest alternating path between s and x.
- lacksquare Know: $|\eta| < |\pi_1|$ and $|\eta| < |\pi_2|$.
- **5** Easy to verify: $\pi_1 \mid\mid \eta$ or $\pi_2 \mid\mid \eta$ is an alternating cycle shorter than σ involving s.
- A contradiction.

- Claim: d(x) = d'(x), for all $x \in \sigma$.
 - **1** assume for contradiction: d(x) < d'(x).

 - $oldsymbol{0}$ $oldsymbol{\eta}$: shortest alternating path between s and x.
 - lacksquare Know: $|\eta| < |\pi_1|$ and $|\eta| < |\pi_2|$.
 - **5** Easy to verify: $\pi_1 \mid\mid \eta$ or $\pi_2 \mid\mid \eta$ is an alternating cycle shorter than σ involving s.
 - A contradiction.

- Claim: d(x) = d'(x), for all $x \in \sigma$.
 - **1** assume for contradiction: d(x) < d'(x).
 - $\mathbf{2} \ \pi_1, \pi_2$: paths from x to s formed by σ .
 - $exttt{0} \hspace{0.1cm} \eta$: shortest alternating path between s and x.
 - lacksquare Know: $|\eta| < |\pi_1|$ and $|\eta| < |\pi_2|$.
 - **5** Easy to verify: $\pi_1 \mid\mid \eta$ or $\pi_2 \mid\mid \eta$ is an alternating cycle shorter than σ involving s.
 - A contradiction.

- Claim: d(x) = d'(x), for all $x \in \sigma$.
 - **1** assume for contradiction: d(x) < d'(x).
 - ② π_1, π_2 : paths from x to s formed by σ .
 - $oldsymbol{0}$ η : shortest alternating path between s and x.
 - lacksquare Know: $|\eta| < |\pi_1|$ and $|\eta| < |\pi_2|$.
 - **5** Easy to verify: $\pi_1 \mid\mid \eta$ or $\pi_2 \mid\mid \eta$ is an alternating cycle shorter than σ involving s.
 - A contradiction.

- Claim: d(x) = d'(x), for all $x \in \sigma$.
 - **1** assume for contradiction: d(x) < d'(x).
 - $m{Q}$ $m{\pi_1}, m{\pi_2}$: paths from $m{x}$ to $m{s}$ formed by $m{\sigma}$.
 - $oldsymbol{0}$ η : shortest alternating path between s and x.
 - lacksquare Know: $|\eta| < |\pi_1|$ and $|\eta| < |\pi_2|$.
 - **5** Easy to verify: $\pi_1 \mid\mid \eta$ or $\pi_2 \mid\mid \eta$ is an alternating cycle shorter than σ involving s.
 - A contradiction.

- Claim: d(x) = d'(x), for all $x \in \sigma$.
 - **1** assume for contradiction: d(x) < d'(x).
 - $m{Q}$ $m{\pi_1}, m{\pi_2}$: paths from $m{x}$ to $m{s}$ formed by $m{\sigma}$.
 - $oldsymbol{0}$ η : shortest alternating path between s and x.
 - lacksquare Know: $|\eta| < |\pi_1|$ and $|\eta| < |\pi_2|$.
 - **5** Easy to verify: $\pi_1 \mid\mid \eta$ or $\pi_2 \mid\mid \eta$ is an alternating cycle shorter than σ involving s.
 - A contradiction.

- Claim: d(x) = d'(x), for all $x \in \sigma$.
 - **1** assume for contradiction: d(x) < d'(x).
 - ② π_1, π_2 : paths from x to s formed by σ .
 - $oldsymbol{0}$ η : shortest alternating path between s and x.
 - lacksquare Know: $|\eta| < |\pi_1|$ and $|\eta| < |\pi_2|$.
 - **5** Easy to verify: $\pi_1 \mid\mid \eta$ or $\pi_2 \mid\mid \eta$ is an alternating cycle shorter than σ involving s.
 - A contradiction.

- lacktriangledown Compute alternating lacktriangledown for $oldsymbol{H}$, and find a bridge $oldsymbol{u}oldsymbol{v}$ in it.
- ${f 2}$ If ${f M}$ is not a maximal matching, then there exists an augmenting path for ${f G}$.
- By lemma ∃ bridge.
- ① Computing the bridge uv takes O(m) time.
- ullet Extract paths from s to u and from s to v in \mathfrak{T} .
- ullet Glue path together with uv to form an odd cycle μ in H.
- $extcolor{large}{0}$ namely, $\mu = au_{su} \mid\mid uv \mid\mid au_{vs}.$
- \odot If μ corresponds to an alternating path in ${f G}$ then done.

- **①** Compute alternating **BFS** $\mathfrak T$ for H, and find a bridge uv in it.
- f 2 If m M is not a maximal matching, then there exists an augmenting path for f G.
- By lemma ∃ bridge.
- ① Computing the bridge uv takes O(m) time.
- ullet Extract paths from s to u and from s to v in \mathfrak{T} .
- lacktriangledown Glue path together with $m{u}m{v}$ to form an odd cycle $m{\mu}$ in $m{H}$.
- $extbf{0}$ namely, $\mu = au_{su} \mid\mid uv \mid\mid au_{vs}$.
- \odot If μ corresponds to an alternating path in ${f G}$ then done.

- **①** Compute alternating **BFS** $\mathfrak T$ for H, and find a bridge uv in it.
- f 2 If m M is not a maximal matching, then there exists an augmenting path for f G.
- By lemma ∃ bridge.
- ① Computing the bridge uv takes O(m) time.
- ullet Extract paths from s to u and from s to v in \mathfrak{T} .
- lacktriangledown Glue path together with $m{u}m{v}$ to form an odd cycle $m{\mu}$ in $m{H}$.
- $extcolor{large}{0}$ namely, $\mu = au_{su} \mid\mid uv \mid\mid au_{vs}.$
- @ If μ corresponds to an alternating path in ${\sf G}$ then done.

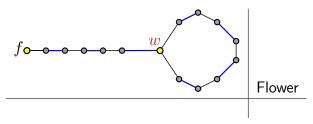
- **①** Compute alternating **BFS** $\mathfrak T$ for H, and find a bridge uv in it.
- f 2 If m M is not a maximal matching, then there exists an augmenting path for f G.
- By lemma ∃ bridge.
- Computing the bridge uv takes O(m) time.
- ullet Extract paths from s to u and from s to v in ${\mathfrak I}$.
- lacktriangledown Glue path together with uv to form an odd cycle μ in H.
- $extbf{0}$ namely, $\mu = au_{su} \mid\mid uv \mid\mid au_{vs}$.
- \odot If μ corresponds to an alternating path in ${f G}$ then done.

- **①** Compute alternating **BFS** $\mathfrak T$ for H, and find a bridge uv in it.
- f 2 If m M is not a maximal matching, then there exists an augmenting path for f G.
- By lemma ∃ bridge.
- **①** Computing the bridge uv takes O(m) time.
- **3** Extract paths from s to u and from s to v in \mathfrak{T} .
- lacktriangledown Glue path together with uv to form an odd cycle μ in H.
- $extbf{0}$ namely, $\mu = au_{su} \mid\mid uv \mid\mid au_{vs}$.
- \odot If μ corresponds to an alternating path in ${f G}$ then done.

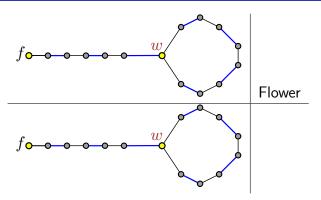
- **①** Compute alternating **BFS** $\mathfrak T$ for H, and find a bridge uv in it.
- f 2 If m M is not a maximal matching, then there exists an augmenting path for f G.
- By lemma ∃ bridge.
- **①** Computing the bridge uv takes O(m) time.
- **5** Extract paths from s to u and from s to v in \mathfrak{T} .
- **o** Glue path together with uv to form an odd cycle μ in H.
- $extbf{0}$ namely, $\mu = au_{su} \mid\mid uv \mid\mid au_{vs}$.
- \odot If μ corresponds to an alternating path in G then done.

- **①** Compute alternating **BFS** $\mathfrak T$ for H, and find a bridge uv in it.
- f 2 If m M is not a maximal matching, then there exists an augmenting path for f G.
- By lemma ∃ bridge.
- **①** Computing the bridge uv takes O(m) time.
- **5** Extract paths from s to u and from s to v in \mathfrak{T} .
- **10** Glue path together with uv to form an odd cycle μ in H.
- $m{0}$ namely, $m{\mu} = m{ au}_{su} \mid\mid m{u}m{v}\mid\mid m{ au}_{vs}.$
- @ If μ corresponds to an alternating path in ${\sf G}$ then done.

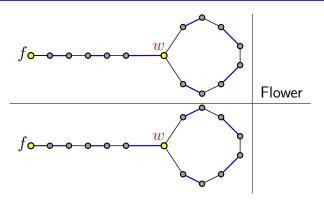
- **①** Compute alternating **BFS** $\mathfrak T$ for H, and find a bridge uv in it.
- f 2 If m M is not a maximal matching, then there exists an augmenting path for f G.
- By lemma ∃ bridge.
- **①** Computing the bridge uv takes O(m) time.
- **5** Extract paths from s to u and from s to v in \mathfrak{T} .
- **o** Glue path together with uv to form an odd cycle μ in H.
- $m{0}$ namely, $\mu = au_{su} \mid\mid uv \mid\mid au_{vs}$.
- lacktriangle If μ corresponds to an alternating path in lacktriangle then done.



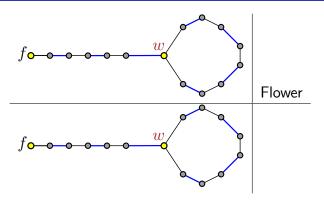
- ① Flower is made out of a **stem** (the path fw), and an odd length cycle which is the blossom.
- ② Stem: Even length alternating path starting with a free vertex



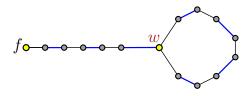
- Flower is made out of a **stem** (the path fw), and an odd length cycle which is the blossom.
- ② Stem: Even length alternating path starting with a free vertex

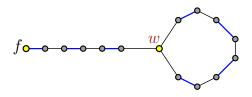


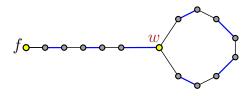
- Flower is made out of a **stem** (the path fw), and an odd length cycle which is the blossom.
- Stem: Even length alternating path starting with a free vertex

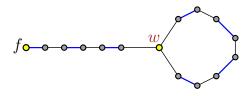


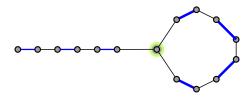
- Flower is made out of a **stem** (the path fw), and an odd length cycle which is the blossom.
- ② Stem: Even length alternating path starting with a free vertex



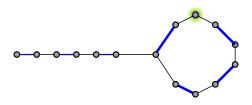


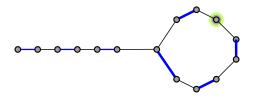


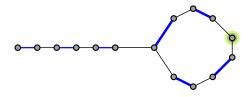




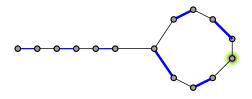




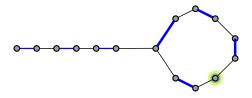


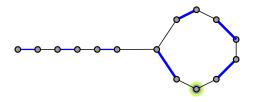


Sariel (UIUC) New CS473 36 Fall 2015 36 / 48

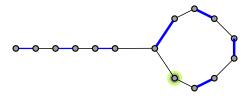


Sariel (UIUC) New CS473 36 Fall 2015 36 / 48

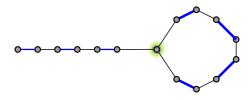




Sariel (UIUC) New CS473 36 Fall 2015 36 / 48



Sariel (UIUC) New CS473 36 Fall 2015 36 / 48



Flowers

- $lacktriangledown \pi_{su}$ and π_{sv} : two paths from s to u and v.
- ② w: lowest vertex in $\mathfrak T$ common to π_{su} and π_{sv} .
- Flower:

Definition

Given a matching M, a **flower** for M is formed by a **stem** and a **blossom**. The stem is an even length alternating path starting at a free vertex v ending at vertex v, and the blossom is an odd length (alternating) cycle based at v.

Flowers

- $lackbox{1}{\bullet} \pi_{su}$ and π_{sv} : two paths from s to u and v.
- ② w: lowest vertex in $\mathfrak T$ common to π_{su} and π_{sv} .
- Second Flower:

Definition

Given a matching M, a **flower** for M is formed by a **stem** and a **blossom**. The stem is an even length alternating path starting at a free vertex v ending at vertex v, and the blossom is an odd length (alternating) cycle based at v.

Flowers

- $lackbox{1}{\bullet} \pi_{su}$ and π_{sv} : two paths from s to u and v.
- ② w: lowest vertex in $\mathfrak T$ common to π_{su} and π_{sv} .
- Flower:

Definition

Given a matching M, a **flower** for M is formed by a **stem** and a **blossom**. The stem is an even length alternating path starting at a free vertex v ending at vertex v, and the blossom is an odd length (alternating) cycle based at v.

Lemma

Lemma

Consider a bridge edge $uv \in G$, and let w be the least common ancestor (LCA) of u and v in \mathfrak{T} . Consider the path π_{sw} together with the cycle $C = \pi_{wu} \mid\mid uv \mid\mid \pi_{vw}$. Then π_{sw} and C together form a flower.

Proof

Proof.

Since only the even depth nodes in $\mathfrak T$ have more than one child, w must be of even depth, and as such π_{sw} is of even length. As for the second claim, observe that $\alpha=|\pi_{wu}|=|\pi_{wv}|$ since the two nodes have the same depth in $\mathfrak T$. In particular,

- lacktriangledown translate blossom of $H\Rightarrow$ original graph lacktriangledown.
- 2 Path s to w corresponds to an alternating path starting at a free vertex f (of G) and ending at w.
- the last edge is in the stem is in the matching.
- cycle $w \dots u \dots v \dots w$ is an alternating odd length cycle in where the two edges adjacent to are unmatched.
- Oan not apply blossom to a matching to get better matching.
- Yields an illegal matching!
- But we discovered odd alternating cycle!

- **1** translate blossom of $H \Rightarrow$ original graph **G**.
- 2 Path s to w corresponds to an alternating path starting at a free vertex f (of G) and ending at w.
- the last edge is in the stem is in the matching.
- cycle $w \dots u \dots v \dots w$ is an alternating odd length cycle in where the two edges adjacent to are unmatched.
- Oan not apply blossom to a matching to get better matching.
- Yields an illegal matching!
- But we discovered odd alternating cycle!

- **1** translate blossom of $H \Rightarrow$ original graph **G**.
- 2 Path s to w corresponds to an alternating path starting at a free vertex f (of G) and ending at w.
- the last edge is in the stem is in the matching.
- cycle $w \dots u \dots v \dots w$ is an alternating odd length cycle in where the two edges adjacent to are unmatched.
- Oan not apply blossom to a matching to get better matching.
- Yields an illegal matching
- But we discovered odd alternating cycle!

- **1** translate blossom of $H \Rightarrow$ original graph **G**.
- 2 Path s to w corresponds to an alternating path starting at a free vertex f (of G) and ending at w.
- the last edge is in the stem is in the matching.
- ullet cycle $w \dots u \dots v \dots w$ is an alternating odd length cycle in ${f G}$ where the two edges adjacent to w are unmatched.
- Oan not apply blossom to a matching to get better matching.
- Yields an illegal matching!
- But we discovered odd alternating cycle!

- **1** translate blossom of $H \Rightarrow$ original graph **G**.
- 2 Path s to w corresponds to an alternating path starting at a free vertex f (of G) and ending at w.
- the last edge is in the stem is in the matching.
- ullet cycle $w \dots u \dots v \dots w$ is an alternating odd length cycle in ${f G}$ where the two edges adjacent to w are unmatched.
- Can not apply blossom to a matching to get better matching.
- Yields an illegal matching!
- But we discovered odd alternating cycle!

- **1** translate blossom of $H \Rightarrow$ original graph **G**.
- 2 Path s to w corresponds to an alternating path starting at a free vertex f (of G) and ending at w.
- the last edge is in the stem is in the matching.
- ullet cycle $w \dots u \dots v \dots w$ is an alternating odd length cycle in ${f G}$ where the two edges adjacent to w are unmatched.
- Can not apply blossom to a matching to get better matching.
- Yields an illegal matching!
- But we discovered odd alternating cycle!

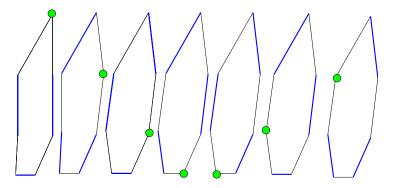
- **1** translate blossom of $H \Rightarrow$ original graph **G**.
- 2 Path s to w corresponds to an alternating path starting at a free vertex f (of G) and ending at w.
- the last edge is in the stem is in the matching.
- ullet cycle $w \dots u \dots v \dots w$ is an alternating odd length cycle in ${f G}$ where the two edges adjacent to w are unmatched.
- Can not apply blossom to a matching to get better matching.
- Yields an illegal matching!
- But we discovered odd alternating cycle!

What we proved...

Lemma

Given a graph ${\bf G}$ with ${\bf n}$ vertices and ${\bf m}$ edges, and a matching ${\bf M}$, one can find in ${\bf O}(n+m)$ time, either a blossom in ${\bf G}$ or an augmenting path in ${\bf G}$.

Odd alternating cycles are awesome!



Sariel (UIUC) New CS473 42 Fall 2015 42 / 48

- How matching in G interact with an odd length alternating cycle?
- Assume free vertex in the cycle is unmatched
- ullet Cycle with t vertices... Use at most (t-1)/2 edges in matching.
- O Rotate the matching edges in the cycle!
- any vertex on cycle can be free.

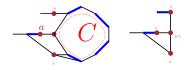
- How matching in G interact with an odd length alternating cycle?
- Assume free vertex in the cycle is unmatched.
- ullet Cycle with t vertices... Use at most (t-1)/2 edges in matching.
- O Rotate the matching edges in the cycle!
- Any vertex on cycle can be free.

- How matching in G interact with an odd length alternating cycle?
- Assume free vertex in the cycle is unmatched.
- ullet Cycle with t vertices... Use at most (t-1)/2 edges in matching.
- O Rotate the matching edges in the cycle!
- Solution Any vertex on cycle can be free.

- How matching in G interact with an odd length alternating cycle?
- Assume free vertex in the cycle is unmatched.
- **3** Cycle with t vertices... Use at most (t-1)/2 edges in matching.
- Oracle Rotate the matching edges in the cycle!
- 6 Any vertex on cycle can be free.

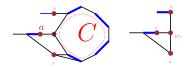
- How matching in G interact with an odd length alternating cycle?
- Assume free vertex in the cycle is unmatched.
- **3** Cycle with t vertices... Use at most (t-1)/2 edges in matching.
- Oracle Rotate the matching edges in the cycle!
- Any vertex on cycle can be free.

Collapse odd alternating cycles...



- $oldsymbol{G}/C$: denote graph resulting from collapsing an odd cycle C into single vertex.
- ② New vertex is marked by $\{C\}$.

Collapse odd alternating cycles...



- $oldsymbol{G}/C$: denote graph resulting from collapsing an odd cycle C into single vertex.
- ② New vertex is marked by $\{C\}$.

A lemma

Lemma

Given a graph G, a matching M, and a flower B, one can find a matching M' with the same cardinality, such that the blossom of B contains a free (i.e., unmatched) vertex in M'.

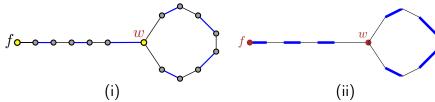
Sariel (UIUC) New CS473 45 Fall 2015 45 / 48

Proof

Proof.

If the stem of B is empty and B is just formed by a blossom, and then we are done. Otherwise, B was as stem π which is an even length alternating path starting from from a free vertex v. Observe that the matching $M' = M \oplus \pi$ is of the same cardinality, and the cycle in B now becomes an alternating odd cycle, with a free vertex. Intuitively, what we did is to apply the stem to the matching M. See Figure $\ref{eq:total_point}$?

Proof by figure



(i) the flower, and (ii) the inverted stem.

Kill the flower, save the matching algorithm

Theorem

Let M be a matching, and let C be a blossom for M with an unmatched vertex v. Then, M is a maximum matching in G if and only if $M/C = M \setminus C$ is a maximum matching in G/C.

Proof

Proof.

Let G/C be the collapsed graph, with $\{C\}$ denoting the vertex that correspond to the cycle C.

Note, that the collapsed vertex $\{C\}$ in G/C is free. Thus, an augmenting path π in G/C either avoids the collapsed vertex $\{C\}$ altogether, or it starts or ends there. In any case, we can rotate the matching around C such that π would be an augmenting path in G. Thus, if M/C is not a maximum matching in G/C then there exists an augmenting path in G/C, which in turn is an augmenting path in G, and as such M is not a maximum matching in G. Similarly, if π is an augmenting path in **G** and it avoids C then it is also an augmenting path in G/C, and then M/C is not a maximum matching in G/C.

Otherwise, since π starts and ends in two different free vertices and C has only one free vertex, it follows that π has an endpoint outside

In other words...

Corollary

Let M be a matching, and let C be an alternating odd length cycle with the unmatched vertex being free. Then, there is an augmenting path in G if and only if there is an augmenting path in G/C.

17.3.2: The algorithm

The algorithm...

- lacksquare Start from empty matching M in graph lacksquare .
- Now, repeatedly, try to enlarge the matching.
- First, check if you can find an edge with both endpoints being free, and if so add it to the matching.
- Compute the graph H (all free vertices collapsed into a single vertex).
- ullet Compute an alternating BFS tree in $oldsymbol{H}$
- Extract shortest alternating cycle based in the root (by finding the highest bridge).
- If alternating cycle corresponds to an alternating path in G then apply and continue.

Sariel (UIUC) New CS473 52 Fall 2015 52 / 48

The algorithm...

- lacksquare Start from empty matching M in graph lacksquare .
- Now, repeatedly, try to enlarge the matching.
- First, check if you can find an edge with both endpoints being free, and if so add it to the matching.
- Compute the graph H (all free vertices collapsed into a single vertex).
- ullet Compute an alternating BFS tree in $oldsymbol{H}$
- Extract shortest alternating cycle based in the root (by finding the highest bridge).
- If alternating cycle corresponds to an alternating path in G then apply and continue.

- lacksquare Start from empty matching M in graph lacksquare .
- Now, repeatedly, try to enlarge the matching.
- First, check if you can find an edge with both endpoints being free, and if so add it to the matching.
- Compute the graph H (all free vertices collapsed into a single vertex).
- \odot Compute an alternating BFS tree in H.
- Extract shortest alternating cycle based in the root (by finding the highest bridge).
- If alternating cycle corresponds to an alternating path in G then apply and continue.

- lacksquare Start from empty matching M in graph lacksquare .
- Now, repeatedly, try to enlarge the matching.
- First, check if you can find an edge with both endpoints being free, and if so add it to the matching.
- ullet Compute the graph $oldsymbol{H}$ (all free vertices collapsed into a single vertex).
- \odot Compute an alternating BFS tree in H.
- Extract shortest alternating cycle based in the root (by finding the highest bridge).
- If alternating cycle corresponds to an alternating path in G then apply and continue.

- lacksquare Start from empty matching M in graph lacksquare .
- Now, repeatedly, try to enlarge the matching.
- First, check if you can find an edge with both endpoints being free, and if so add it to the matching.
- ullet Compute the graph $oldsymbol{H}$ (all free vertices collapsed into a single vertex).
- \odot Compute an alternating BFS tree in H.
- Extract shortest alternating cycle based in the root (by finding the highest bridge).
- If alternating cycle corresponds to an alternating path in G then apply and continue.

- lacksquare Start from empty matching M in graph lacksquare .
- Now, repeatedly, try to enlarge the matching.
- First, check if you can find an edge with both endpoints being free, and if so add it to the matching.
- ullet Compute the graph $oldsymbol{H}$ (all free vertices collapsed into a single vertex).
- Extract shortest alternating cycle based in the root (by finding the highest bridge).
- If alternating cycle corresponds to an alternating path in G then apply and continue.

- lacksquare Start from empty matching M in graph lacksquare .
- Now, repeatedly, try to enlarge the matching.
- First, check if you can find an edge with both endpoints being free, and if so add it to the matching.
- ullet Compute the graph $oldsymbol{H}$ (all free vertices collapsed into a single vertex).
- **5** Compute an alternating BFS tree in H.
- Extract shortest alternating cycle based in the root (by finding the highest bridge).
- If alternating cycle corresponds to an alternating path in G then apply and continue.

- **1** If found a flower, with a stem ρ and a blossom C then:
 - lacksquare apply the stem to M (i.e., $M\oplus
 ho$).
 - O: odd cycle with the free vertex being unmatched.
 - \odot Compute recursively an augmenting path π in G/C
- ② succeeded computing a matching with one edge more in it. Continue till stuck.

- **1** If found a flower, with a stem ρ and a blossom C then:
 - lacksquare apply the stem to M (i.e., $M\oplus
 ho$).

 - \odot Compute recursively an augmenting path π in G/C
- Succeeded computing a matching with one edge more in it. Continue till stuck.

- **1** If found a flower, with a stem ρ and a blossom C then:
 - lacksquare apply the stem to M (i.e., $M\oplus
 ho$).

 - **3** Compute recursively an augmenting path π in G/C.
 - Transform this into an augmenting path in G apply it.
- succeeded computing a matching with one edge more in it. Continue till stuck.

- **1** If found a flower, with a stem ho and a blossom C then:
 - **1** apply the stem to M (i.e., $M \oplus \rho$).
 - **2** *C*: odd cycle with the free vertex being unmatched.
 - **3** Compute recursively an augmenting path π in G/C.
 - Transform this into an augmenting path in G apply it.
- succeeded computing a matching with one edge more in it. Continue till stuck.

17.3.2.1:Running time analysis

Sariel (UIUC) New CS473 54 Fall 2015 54 / 48

- Every shrink cost us O(m+n) time.
- ② Need to perform O(n) recursive shrink operations till find an augmenting path, if such a path exists.
- Omputing an augmenting path takes O(n(m+n)) time.
- $ext{ } ext{ }$
- **3** Overall running time is $O(n^2(m+n)) = O(n^4)$.

- Every shrink cost us O(m+n) time.
- ullet Need to perform O(n) recursive shrink operations till find an augmenting path, if such a path exists.
- **3** Computing an augmenting path takes O(n(m+n)) time.
- $ext{ } ext{ }$
- ${ t ar 0}$ Overall running time is $O(n^2(m+n)) = O(n^4)$.

- Every shrink cost us O(m+n) time.
- ② Need to perform O(n) recursive shrink operations till find an augmenting path, if such a path exists.
- **3** Computing an augmenting path takes O(n(m+n)) time.
- Have to repeat this O(n) times.
- ${ toldeleta}$ Overall running time is $O(n^2(m+n)) = O(n^4)$.

- Every shrink cost us O(m+n) time.
- ② Need to perform O(n) recursive shrink operations till find an augmenting path, if such a path exists.
- $lacksquare{1}{3}$ Computing an augmenting path takes O(n(m+n)) time.
- Have to repeat this O(n) times.
- **3** Overall running time is $O(n^2(m+n)) = O(n^4)$.

The result

Theorem

Given a graph G with n vertices and m edges, computing a maximum size matching in G can be done in $O(n^2m)$ time.

Sariel (UIUC) New CS473 56 Fall 2015 56 / 48

17.3.2.2: Maximum Weight Matching in A Non-Bipartite Graph

Maximum Weight Matching in A Non-Bipartite Graph

his the hardest case and it is non-trivial to handle. See internet/literature for details.

Sariel (UIUC) New CS473 59 Fall 2015 59 / 48

Sariel (UIUC) New CS473 60 Fall 2015 60 / 48

Sariel (UIUC) New CS473 61 Fall 2015 61 / 48

Sariel (UIUC) New CS473 62 Fall 2015 62 / 48

J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM J. Comput.*, 2:225–231, 1973.

Sariel (UIUC) New CS473 62 Fall 2015 62 / 48