Edmonds - Karp : find shortest path in G_f.

Lemma monotonic

$\delta_f(v)$: $|s \rightsquigarrow v|$ in G_f

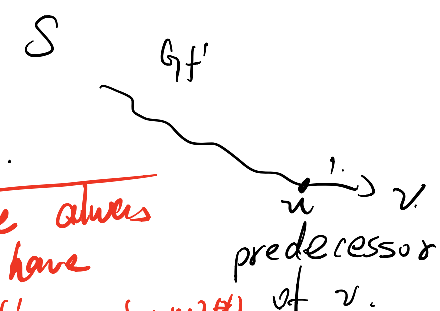
$\delta_f(v)$ $\forall v$ does not $\searrow$.

proof : Assume :

$\delta_{f'}(v) < \delta_f(v)$.

new path . after augmenting.

define v to be the one with smallest $\delta_{f'}(v)$, s.t. $\delta_{f'}(v) < \delta_f(v)$.



$\delta_{f'}(v) = \delta_{f'}(u) + 1 \geq \delta_f(u) + 1$.

we always have $\delta_{f'}(u) \geq \delta_f(u)$ (*).

otherwise

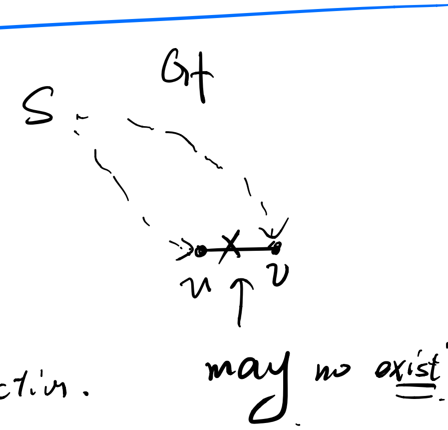
$\delta_{f'}(u) < \delta_f(u)$.

start cond :

so $\delta_{f'}(u) < \delta'_f(v)$.

v is the smallest one , contradiction.

Case 1 : $(u, v) \in G_f$.



$\Rightarrow \delta_f(u) + 1 \geq \delta_f(v)$

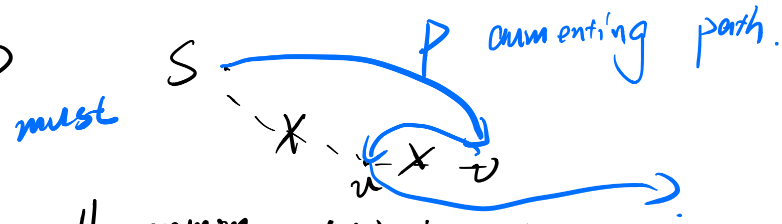
$\Rightarrow \delta_{f'}(v) \geq \delta_f(v)$. contradiction.

Case 2:

$(u,v) \notin G_f$.

$\Rightarrow$ we augmenting go through. ie.

$\Rightarrow$



so we will remove (v,u) in G_f.

and add. (u,v) in $G_{f'}$

$$\left.\begin{array}{l} (u,v) \notin G_f \\ (u,v) \in G_{f'} \end{array}\right\} \Rightarrow (v,u) \in G_f. \quad \in P \text{ in } G_f.$$

$\delta_{f'}(v) \geq \delta_f(v) + 2$. (in EK. we find shortest path).

$$\begin{aligned} \delta_f(v) &= \delta_f(u) - 1. \\ &\leq \delta_{f'}(u) - 1 \quad (\text{as } (*)) \\ &= \delta_{f'}(v) - 1 - 1 \\ &= \delta_{f'}(v) - 2. \end{aligned}$$

so $\delta_{f'}(v) \geq \delta_f(v) + 2$.

Theorm. $O(VE)$ iters augmenting.

def: $c(p) = c(u,v)$ critical edge.

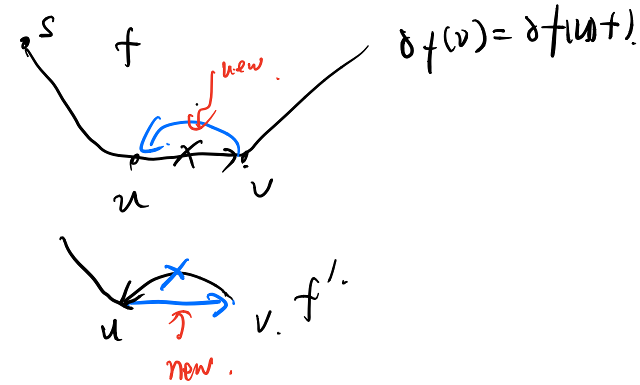
(u,v) can be critical $O(V)$ times

in f':

as before we proof.

$\delta_{f'}(u) = \delta_{f'}(v) + 1 \geq \delta_f(v) + 1.$

v is predecessor of u.



so

$$\delta_{f'}(u) = f'(v) + 1 \geq \delta_f(v) + 1$$
$$= \delta_f(u) + 2.$$

mean: two times augment. $\delta_f(u)$. must increase by at least 2.

$\Rightarrow$. $|V|/2$ times

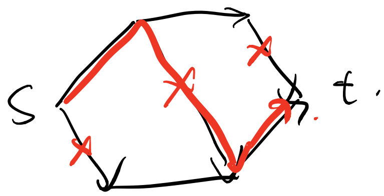
all critical edge. $O(VE)$.

in f at least one critical path

$\Rightarrow$ opreate: $O(VE)$.

So total is $O(VE^2)$.

Dinic $O(V^2E)$.

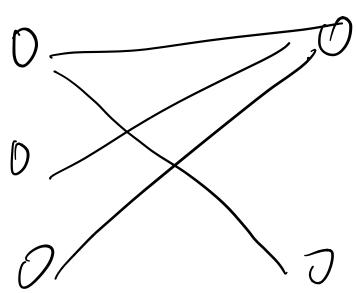


find all shortest path.
aug all of them.

shortsest path $3 \to 4$
$O(V)$.

Dinitz.
hard understand. ⇒ Dinic's
Even

App: bipartite graph.



general match.
subset of edges.

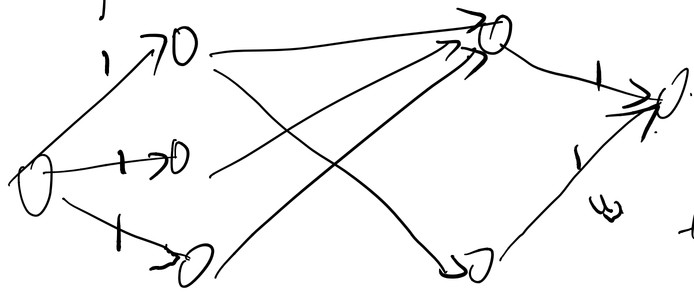
how many task could be handle?

task.
person.

find max: ① 1 -> task 1. X
② { 1 -> task 2.
2 -> task 1 ✓

max flow.

person only can take one task



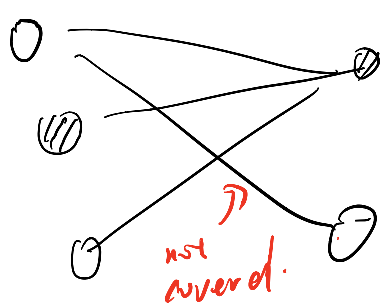
task only can taked by one people.

a k,

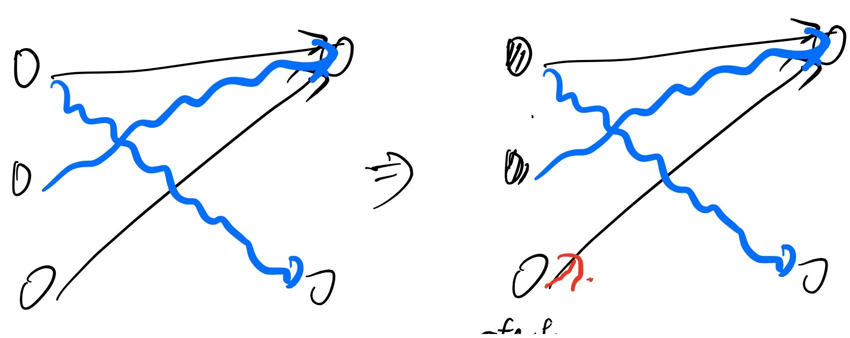
⇒ max matching = k.

bipartite Cover.

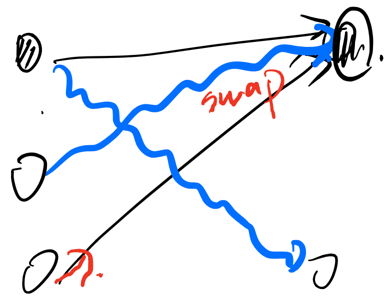
Cover: a set of nodes such that every edge in G. each edge at least link to one of nodes.



min cover k ⟺ max matching k.



start

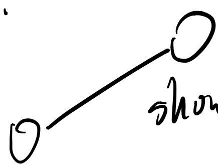


start

it is a cover.

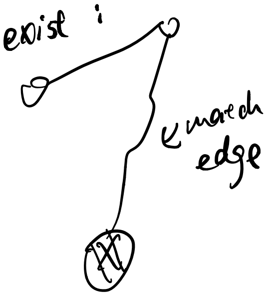
match edge ⇒ covered.

when:

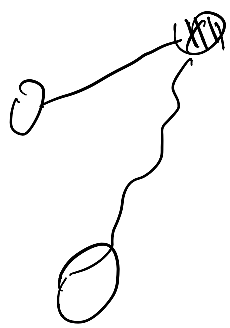


show doesn't happen:

if exist:

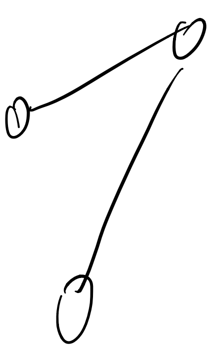


⇒

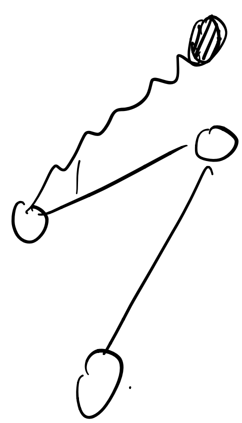


so doesn't exist

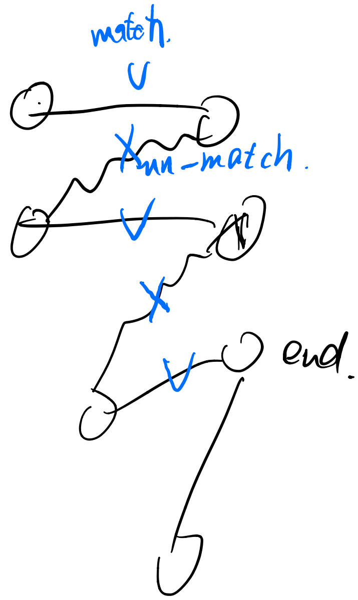
⇒



⇒



$\Rightarrow$



lead to
$\Rightarrow$ a larger
matching