Chapter 5. Graph Algorithms

- Exploring graphs (BFS, DFS)
- Minimum_Cost spanning trees
- Single_source shortest paths
- All_pairs shortest paths and transitive closure
- Network flows.

5.1. Exploring graphs

5.1.1. Breadth_first search (BFS)

In BFS when arriving at a node v, visit all neighbors of v, and proceed level by level.

Input graph $G=(V,E)$ repr. by adj. lists.

Vertices will be colored white, gray, black

$\pi(u)$ = predecessor of vertex u.

$d[u]$ = distance from source s to u.

Q = first_in, first_out queue to store gray vertices

```
BFS (G, s)
 for each u ∈ V-{s} do
      color [u] = white
      d [u] = ∞ ; π[u] = NIL;
 /* BFS after initialization */
 color [s] = gray
 d[s] = 0 ; π[s] = NIL;
 Q = ϕ ;
 ENQUEUE (Q, s);
 while  Q ≠ ϕ  do
      u = DEQUEUE (Q)
      for each v ∈ Adj [u] do
           if color [v] = white
              then
                   color [v] = gray
                   d [v] = d[u]+1
                   π [v] = u
                   ENQUEUE (Q,v)
      color[u] = black
```

Note. BFS trees can be constructed from π

Running time:

Initialization: $O(V)$

While loop: time to scan adj. lists

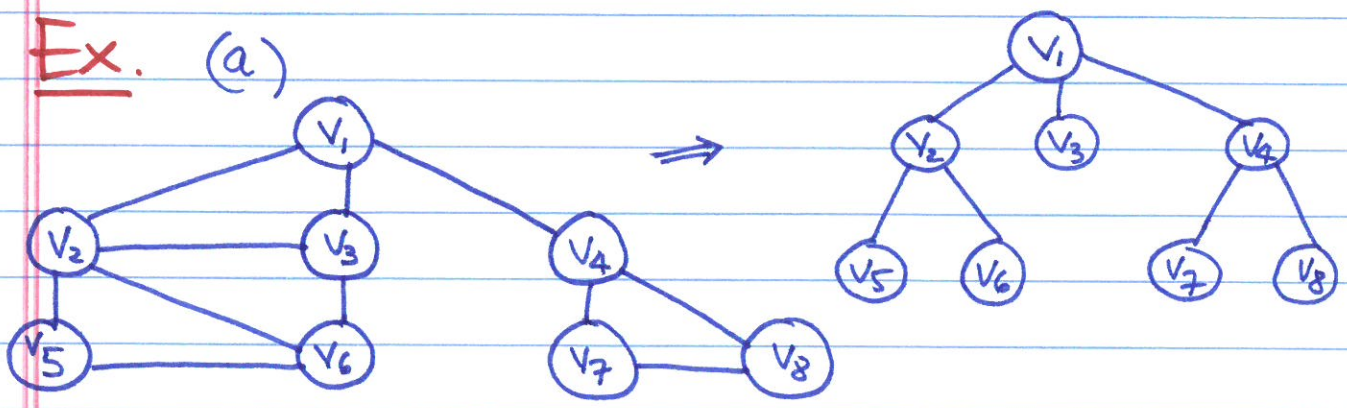
$O(E)$

$\Rightarrow$ Total $= O(V+E)$

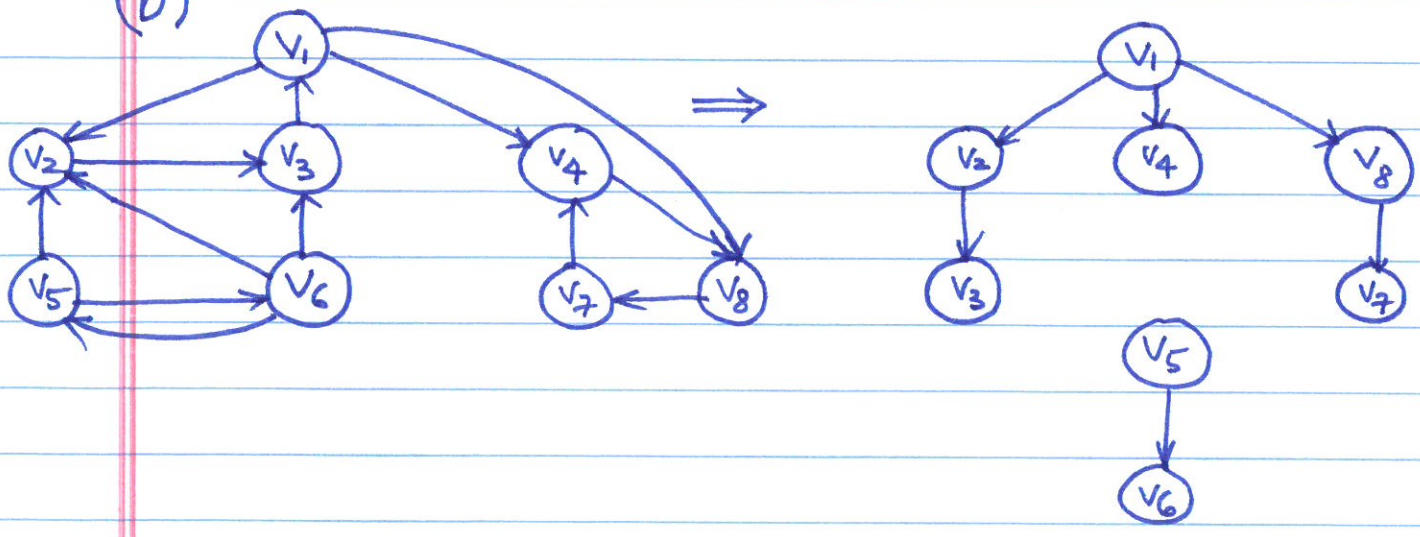
Fact. (1) $d[u]$ = length of shortest path from s to u.

(2) A shortest path from s to v can be obtained from a shortest path from s to $\pi(v)$ + edge $(\pi(v), v)$.

Ex. (a)



(b)



5.1.2. Depth-first search (DFS)

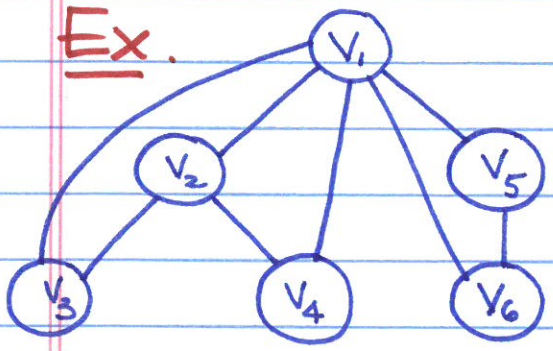
During DFS we associate w/each u:

$d[u]$ = discovery time of u

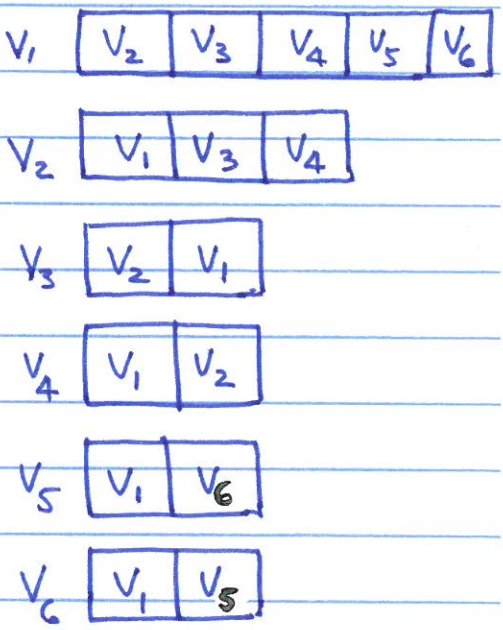
$f[u]$ = finish time of u

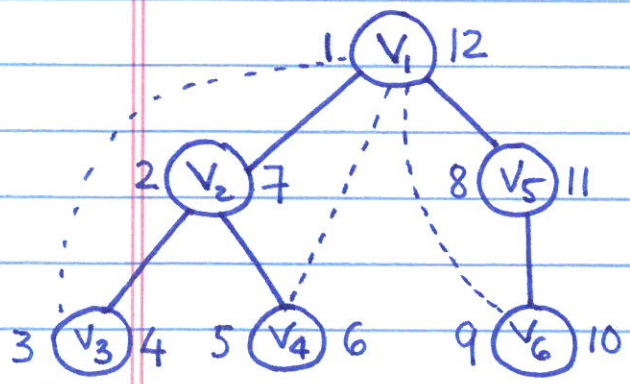
u is colored
- white before $d[u]$
- gray between $d[u], f[u]$
- black after $f[u]$.

Ex.



Adj. Lists





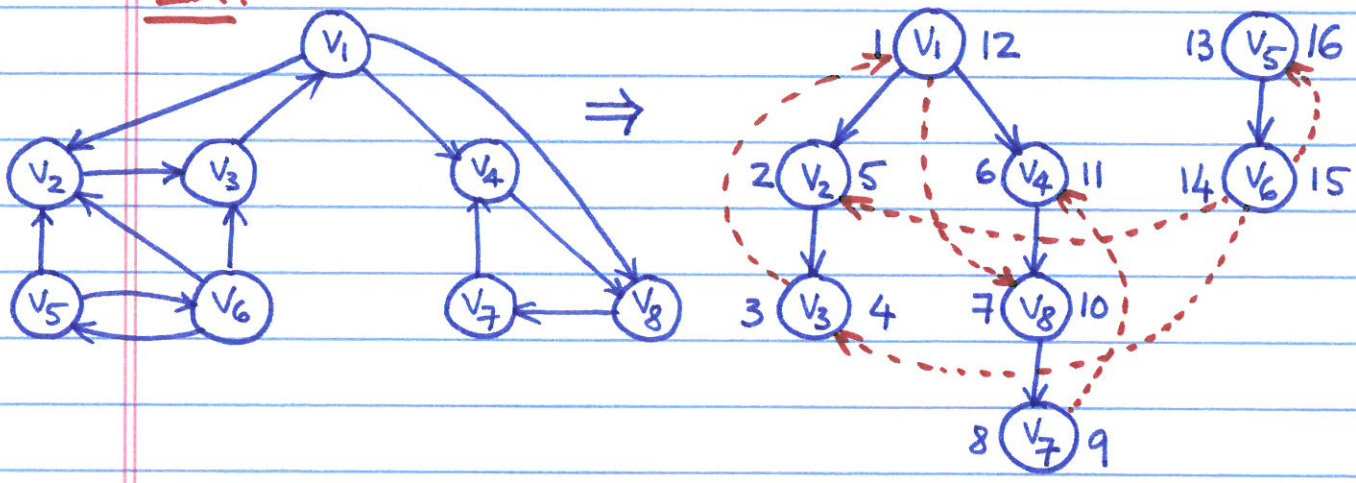
```
DFS (G=(V,E))
    for each u ∈ V do
        color[u] = white ; π[u] = NIL ;
    time = 0
    for each u ∈ V do
        if color[u] = white
            then DFS-VISIT(u)

DFS-VISIT(u)
    color[u] = gray ;   /* u is discovered */
    time = time + 1 ;
    d[u] = time ;
    for each v ∈ Adj[u] do   /* expl. (u,v) */
        if color[v] = white
            then π[v] = u
                 DFS-VISIT(v)
    color[u] = black   /* finish u */
    f[u] = time = time + 1
```

Running time is $\Theta(V+E)$

DFS on directed graphs

Ex.



There are 4 categories of edges:

- Tree edges: leading to new vertices during search
- Forward edges: going from ancestor to descendant
- Back edges: going from descendant to ancestor (possibly to itself)
- Cross edges: connecting vertices that are neither ancestors nor descendants of one another.

Fact. If (u,v) is cross-edge, then $f[u] > f[v]$.

Fact. If G is undirected, then every edge is a tree edge or a back edge.

An application: Finding cycles

Claim. G contains a directed cycle $\iff$ G contains a back edge in DFS.

Pf. "$\Leftarrow$": obvious.

"$\Rightarrow$": Let $C = (v, \ldots, w)$ be a cycle where v has the smallest discovery time $d[v]$. Consider edge (w, v). We prove that

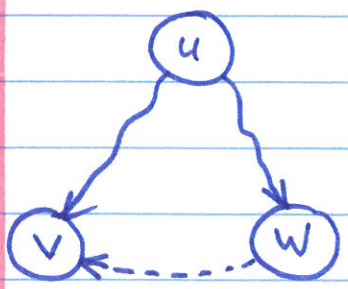
Fact. (w, v) is a back edge.

First observe (w, v) cannot be a tree edge, nor a forward edge.

We show (w, v) cannot be cross edge.

Notice that v, w must belong to the same tree since there is a path $v \rightsquigarrow w$:

Let u be lowest common ancestor of v and w.



The only way v reaches w is through u or an ancestor of u.

Thus, v cannot have the least discovery time in cycle C ↯ Contrad. □

Topological Sort

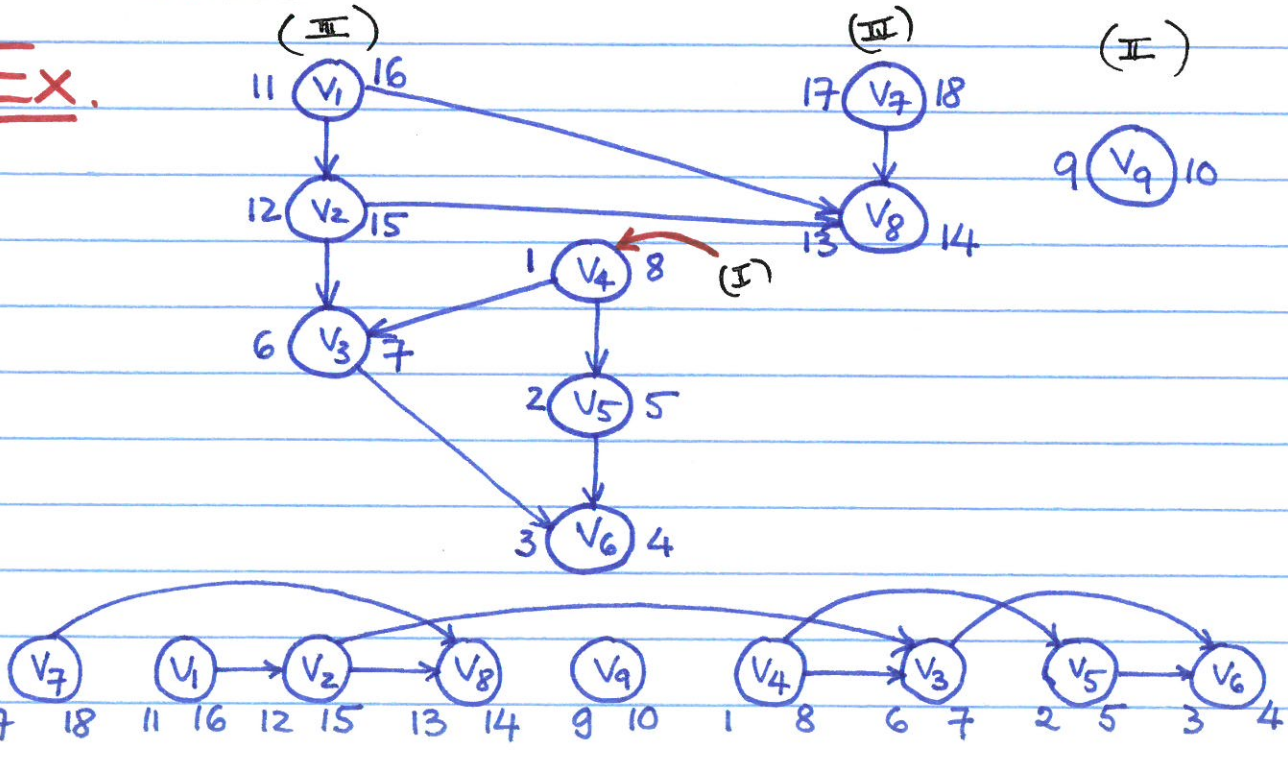
Def. A topological sort of a directed acyclic graph (dag) $G = (V, E)$ is a linear ordering of V s.t.:

$(u,v) \in E \Rightarrow u$ precedes v in the ordering

TOPOLOGICAL-SORT $(G = (V, E))$

Perform DFS(G);
As a vertex v is finished insert onto front of list;
return list

Ex.



5.1.3. Strongly_Connected Components

Def. Let $G=(V,E)$ be a directed graph.
A strongly_connected component (cc) of G is a maximal subset $C \subseteq V$ of vertices s.t. for every pair of vertices $u,v \in C$ there is a path from u to v, and vice versa.

G is strongly connected if it has precisely one strong. conn. component.

The transpose G^T of G is $G^T=(V,E^T)$ where $(u,v) \in E^T \Leftrightarrow (v,u) \in E$.

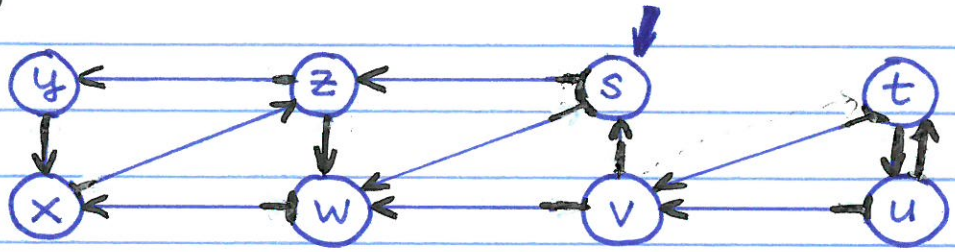
Lem. For any two vertices u, w let
$U=[d[u]..f[u]]$, $W=[d[w],f[w]]$
Then one of the following is true:
(1) U and W are disjoint
(2) $U \subseteq W$ & u is descendant of w
(3) $W \subseteq U$ & w is descendant of u

SCC ($G=(V,E)$)

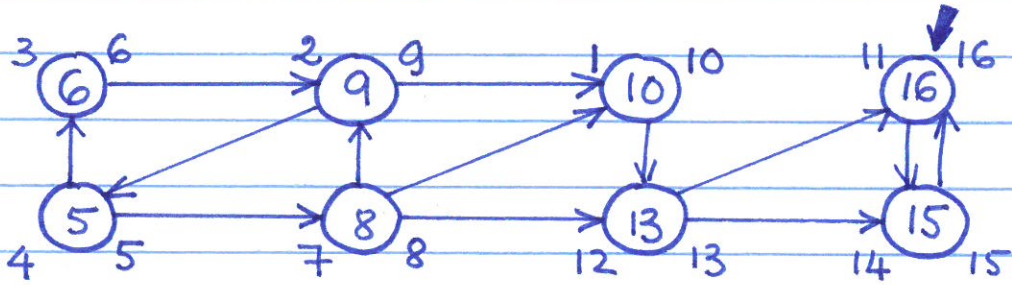
1. Perform DFS on G to comp. $f[v]$;
2. Compute G^T and order V in decreasing $f[v]$;
3. Perform DFS on G^T in decreas. $f[v]$;
4. Each tree in DFS forest obtained yields a SCC

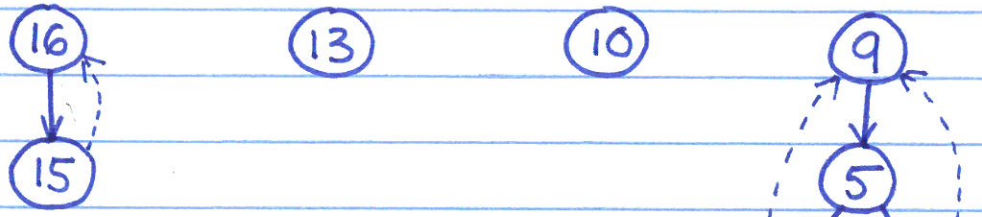
Ex.

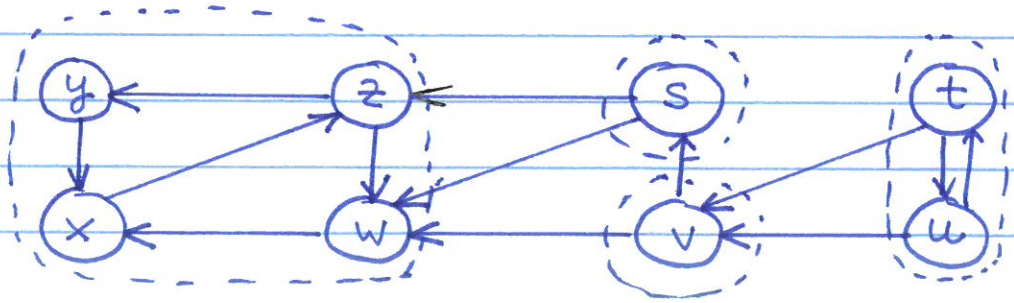
DFS



G^T







Correctness of SCC

Lem. In DFS, vertices in the same SCC belong to the same DFS tree. □

Concerning DFS on G in line 1 def.

$\varphi(u) := w$ s.t. w is reachable from u and $f[w]$ is largest

$\varphi(u)$ is called *forefather* of u

Fact. (1) $f[u] \leq f[\varphi(u)]$

(2) $\varphi(\varphi(u)) = \varphi(u)$

(3) $\varphi(u)$ is ancestor of u in DFS tree

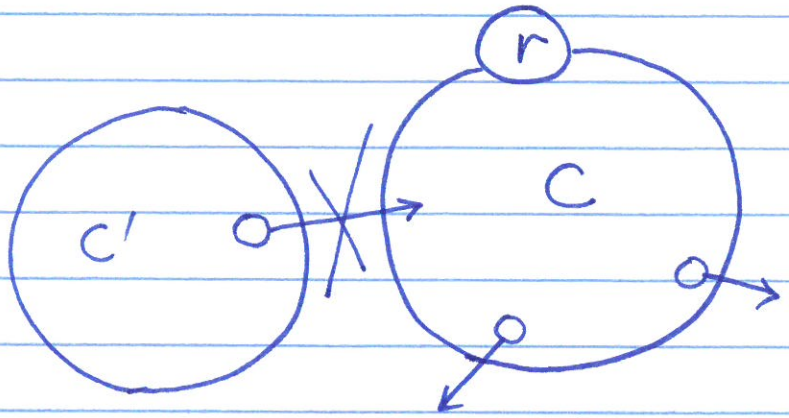
(4) $u, \varphi(u)$ belong to same SCC.

Lem. u, v belong to same SCC $\Leftrightarrow \varphi(u) = \varphi(v)$.

Now to correctness of SCC algor.

- In step (3) we perform DFS on G^T starting with r s.t. $f[r]$ is largest : $\varphi(r) = r$

- Let C be a SCC containing r.
  Then in G there is no edges entering C from outside:



It holds $\forall v \in C : \varphi(v) = r$

$\Rightarrow C = \{v \mid r \text{ can reach } v \text{ in } G^T\}$

$\Rightarrow C =$ vertices of first DFS tree obtained from Step (4)

- By induction, correctness follows.

5.2. Minimum Cost Spanning Trees (MST)

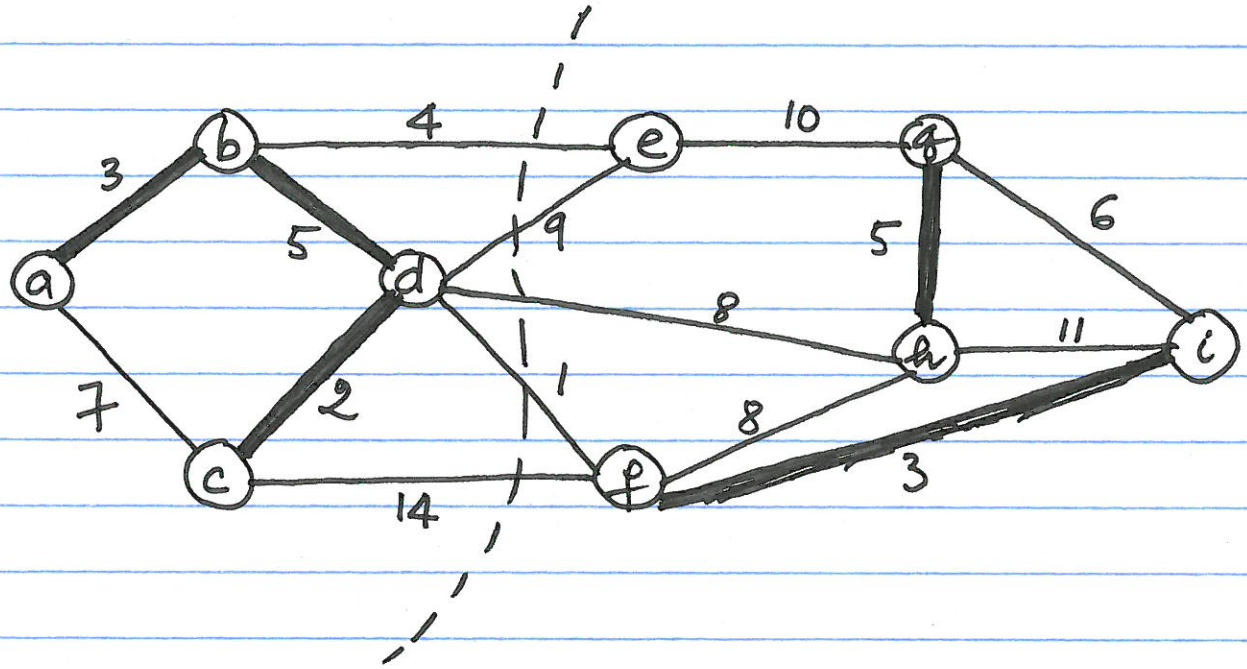
Given a connected undir. graph $G=(V,E)$ and a weight function $w: E \to \mathbb{R}$, we wish to compute a minimum spanning tree $T \subseteq E$, i.e., a tree spanning V s.t. $w(T) = \sum_{e \in T} w(e)$ is minimized.

Let $A \subseteq E$ be a subset of some MST T.

An edge $e \in E-A$ is **safe** if $A \cup \{e\}$ is again a subset of some MST.

```
GENERIC-MST(G,w)
    A = ∅
    while A is not a ST do
        find an edge e safe for A
        A = A ∪ {e}
    return A
```

A cut $(S, V-S)$ of $G=(V,E)$ is a partition of V.



(b,e) crosses cut $(\underbrace{\{a,b,c,d\}}_{S}, V-S)$

(d,f) is light

An edge (u,v) crosses a cut $(S, V-S)$ if $u \in S$ and $v \in V-S$, or vice versa.

A cut $(S, V-S)$ respects A $(\subseteq E)$ if no $e \in A$ crosses $(S, V-S)$.

An edge e is a light edge crossing cut $(S, V-S)$ if its weight is least among edges crossing $(S, V-S)$

Prop. Let $A \subseteq E$ be included in some MST for G. Let $(S, V-S)$ be any cut respecting A and e be a light edge crossing $(S, V-S)$. Then e is safe for A.

Pf. By contradiction ⊠

Cor. Let $A \subseteq E$ be a subset incl. in some MST. Let $C = (V_C, E_C)$ be a tree in forest $G_A = (V, A)$. If e is a light edge connecting C with another tree in G_A, then e is safe for A. ⊠

5.2.1. Kruskal's Algor.

MST_KRUSKAL $(G=(V,E), w)$

$A = \phi$

for each vertex $v \in V$ do MAKE-SET(v)

sort E into nondecreasing order by w

for each $e=(u,v)$ in nondecreas. order do

if FIND(u) $\neq$ FIND(v) then

$A = A \cup \{e\}$

UNION(u,v)

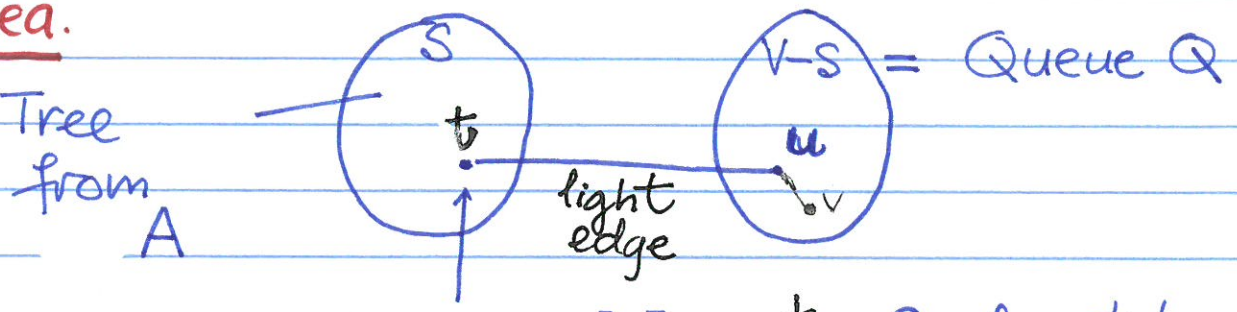
return A

- Sorting E takes $O(E \lg E) = O(E \lg V)$
- $|V|$ MAKE_SET and $O(E)$ FIND & UNION operations takes $O((V+E)\alpha(V))$ time, where α is inverse of Ackerm.'s

Thus, total running time of Kruskal's is $O(E \lg V)$.

5.2.2. Prim's Algor.

Idea.



NEAREST[u] = $t \in S$ closest to u

MIN_DIST[u] = $w(u,t)$

- Initially, $A = \phi$, $S = \{r\}$, $Q = V - \{r\}$
- Let (t,u) be light edge crossing cut $(S, V-S)$:
  - add (t,u) to A (i.e., u to S)
  - update NEAREST[v], MIN-DIST[v], $\forall v \in Q$ ($V-S$ = Q)

    (since u just added to S may be closer to v)

MST_PRIM$(G=(V,E), w, r)$

$S = \{r\}$; $A = \phi$

for each $v \in V - \{r\}$ do

  NEAREST[v] = r

  MIN-DIST[v] = w[v,r]

```
Q = V - {r}
while Q ≠ ∅ do
    u = EXTRACT_MIN (Q)
    A = A ∪ {(u, NEAREST[u])}
    /* Now update NEAREST, MIN_DIST */
    for each v ∈ Adj[u] do
        if v∈Q ∧ w[u,v] < min_dist[v]
            then NEAREST[v] = u
                 MIN_DIST[v] = w[v,u]
```

Running time of Prim's algor.

- Initialization takes $O(V)$ steps
- Min. priority queue Q can be implem. using min_heap : $O(V \lg V)$ time
- Updating NEAREST, MIN_DIST is performed in $O(E)$ time

$\Rightarrow$ Total $= O(V \lg V + E \lg V)$

$= O(E \lg V)$

5.3. Single-Source Shortest Paths

Given: A directed graph $G=(V,E)$ with a weight fctn $w: E \to \mathbb{R}$ (real numbers)

Weight of path $p=(v_0, \ldots, v_k)$:

$$w(p) = \sum_{i=1}^{k} w(v_{i-1}, v_i)$$

Shortest path weight from u to v:

$$\delta(u,v) = \begin{cases} \min\{w(p) \mid u \overset{p}{\rightsquigarrow} v\} & \text{if } u \rightsquigarrow v \text{ paths exist} \\ \infty & \text{otherwise} \end{cases}$$

A shortest path from u to v is a path p from u to v s.t. $w(p)=\delta(u,v)$.

Fact. Any subpath of a shortest path is itself a shortest path.

A shortest path tree rooted at s is a directed subgraph $G'=(V',E')$ s.t.

(1) V' = vertices reachable from s

(2) G' is a tree rooted at s

(3) $\forall v \in V'$ the unique simple path $s \rightsquigarrow v$ is a shortest path from $s \rightsquigarrow v$

Relaxation

For each $v \in V$: $d[v]$ is an upper bound on $\delta(s,v)$ = shortest-path estimate

INITIALIZE-SINGLE-SOURCE $(G=(V,E),s)$

```
for each v ∈ V do
    d[v] = ∞
    π[v] = NIL
    /* π[v] = parent of v in shortest-path tree */
d[s] = 0
```

Relaxing an edge $(u,v) = e$ means improving $d[v]$ by going through e

Illustration:

$d[u] + w[u,v] \overset{?}{<} d[v]$

RELAX (u,v)

```
if d[v] > d[u] + w[u,v]
    then d[v] = d[u] + w[u,v]
         π[v] = u
```

5.3.1. Bellman-Ford Algor.

The Bellman-Ford Algor. return false if there exists a negative-weight cycle reachable from s; otherwise it outputs the shortest paths and their weights.

Bellman-Ford $(G=(V,E), w, s)$

```
INITIALIZE-SINGLE-SOURCE (G, s)
for i = 1 to |V| - 1 do
    for each edge e = (u,v) ∈ E do
        RELAX(u, v)
for each edge e = (u,v) ∈ E do
    if d[v] > d[u] + w[u,v]
        then return FALSE
```

Time complexity : $O(VE)$

Lem. (Path-relaxation property)

If $p = (s = v_0, v_1, \cdots, v_k)$ is a shortest path $s \rightsquigarrow v_k$, and relaxation is in the order

$$(v_0, v_1), (v_1, v_2), \cdots, (v_{k-1}, v_k)$$

then $d[v_k] = \delta(s, v_k)$.

(This property holds true regardless of any other relax. steps intermixed with the above relax. steps.)

Pf. By ind. on relax step (v_{i-1}, v_i), $i = 0, .., k$.

Lem. If G contains no neg.-weight cycles reachable from s, then after the $|V|-1$ iterations of relaxations of BELLMAN-FORD, $d[v] = \delta(s,v)$ for all v reachable from s.

Pf. Let $p = (s = v_0, v_1, \ldots, v_k = v)$ be a shortest path from $s \rightsquigarrow v$. Then $k \leq |V|-1$.

1st. iteration of relaxations relax (v_0, v_1)
ith " " " (v_{i-1}, v_i)

Path-relaxation property $\Longrightarrow$ Lem. □

Theor. Bellman-Ford algor. is correct

Pf. If G contains no neg.-weight cycle, then correctness follows from previous lemma.

Now suppose that G contains a neg.-weight cycle reachable from s.

Assuming Bellman-Ford does not return FALSE, we derive a contradiction.

Let $c = (v_0, v_1, v_2, \ldots, v_k = v_0)$ be a neg.-weight cycle. Then

$$\sum_{i=1}^{k} w(v_{i-1}, v_i) < 0$$

Since Bellman-Ford doesn't return FALSE, $d[v_i] \leq d[v_{i-1}] + w[v_{i-1}, v_i]$

Hence, $\underbrace{\sum_{i=1}^{k} d[v_i]}_{=} \leq \underbrace{\sum_{i=1}^{k} (d[v_{i-1}] + w[v_{i-1}, v_i])}_{=}$

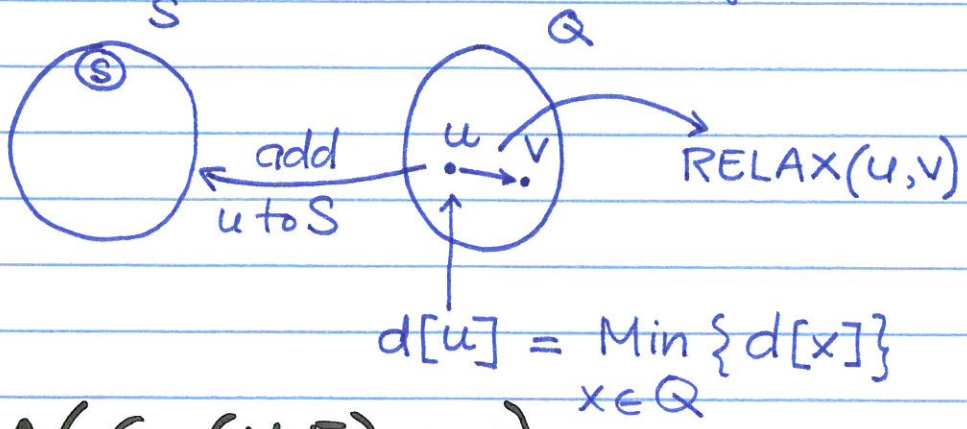
Thus, $0 \leq \sum_{i=1}^{k} w[v_{i-1}, v_i]$

which contradicts the fact that cycle c has neg. weight. □

5.3.2 Dijkstra's Algor.

Assumption: $w(u,v) \geq 0 \quad \forall (u,v) \in E$

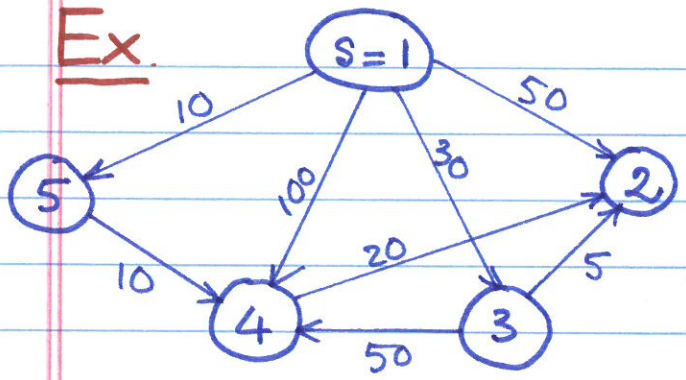
Idea: (Has flavor of Prim's algor.)




```
DIJKSTRA(G=(V,E),w,s)
  INITIALIZE(G,s)
  S = {s}
  Q = V-S
  while Q ≠ ∅ do
     u = EXTRACT_MIN(Q)
     S = S∪{u}
     for each v∈Adj[u] do
        RELAX(u,v)
```

Def. A path $(s = v_0, v_1, \ldots, v_{k-1}, v_k = v)$ from s to v is *special* if $v_0, v_1, \ldots, v_{k-1} \in S$, i.e., all intermediate nodes are in S.

Ex.



Step	u	Q	d
0	—	{2,3,4,5}	[50, 30, 100, 10]
1	5	{2,3,4}	[50, 30, 20]
2	4	{2,3}	[40, 30]
3	3	{2}	[35]
4	2	{ }	

□

Theor. (Correctness of Dijkstra's algor.)

(1) $\forall v \in S : d[v] = \delta(s,v)$ (= length of shortest path from s to v.

(2) $\forall w \in Q : d[w]$ = length of shortest special path from s to w.

Pf. By ind. on $|S|$.

Basis. $|S| = 1$, i.e., $S = \{s\}$: clear

Ind. step. Let v be vertex added to S during current iteration.

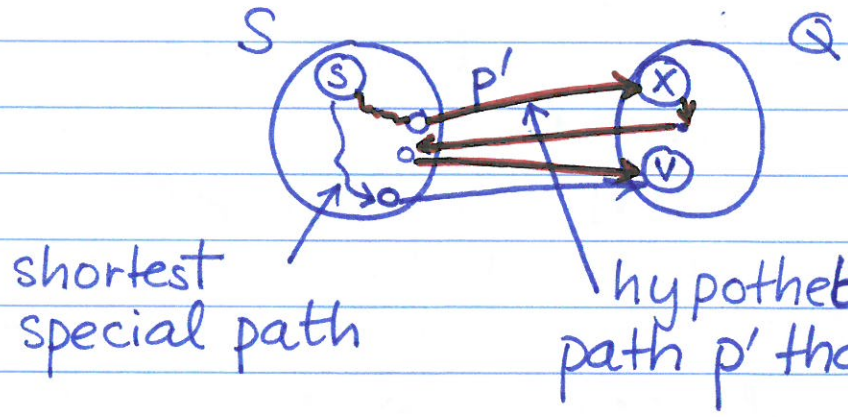
(1) IH $\Rightarrow$ $d[v]$ = length of shortest special path $s \rightsquigarrow v$

Claim. Shortest path from s to v must be a special path.

Suppose by way of contradiction that

there is a shortest path p' from s to v that is <u>not</u> special.

Let x be first interm. node on p' s.t. $x \notin S$, i.e., $x \in Q$. We have:



Then $\text{Length}(p') \geq d[x]$

$\geq d[v]$ by choice of v

Thus (1) is proved.

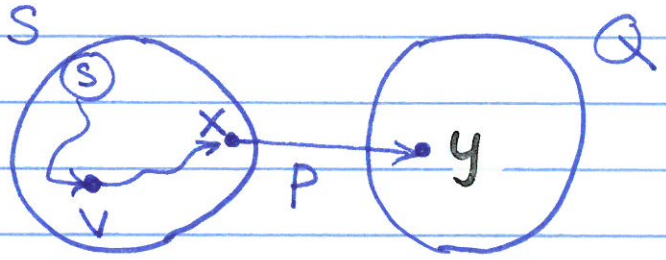
(2) Consider $y \in Q$, $y \neq v$.

We show that right after v is added to S, $d[y]$ = length of shortest spec. path from s to y.

(a) If such shortest spec. path $s \rightsquigarrow y$ doesn't change by adding v to S: ✓

(b) Otherwise, the shortest spec. path $p: s \rightsquigarrow y$ now passes through v.

- If v is last node on p, then the relaxation step updates $d[y]$.
- Otherwise, v is not last node on p:



Then,

$$\begin{aligned} \text{Length}(p) &\geq d[x] + w(x,y) \\ &\geq d[y] \quad \text{(when x was added)} \\ &\geq d[y] \quad \text{(when v was added)} \end{aligned}$$

□

Complexity Analysis

As in Prim's algor, Q is implem. as a min-heap, so complexity is

$$O(E \lg n)$$

Question. How to output a shortest path from s to v for all $v \in V$?

5.4. All-Pairs Shortest Paths

5.4.1. All-Pair Shortest Paths

Input. A weight matrix $W = (w_{ij})$ s.t.

$$w_{ij} = \begin{cases} 0 & \text{if } i = j \\ +\infty & \text{if } (i,j) \notin E \\ \text{weight of } (i,j) & \text{otherwise} \end{cases}$$

Output. Cost matrix $A = (a_{ij})$ s.t.

a_{ij} = cost of shortest path $i \rightsquigarrow j$

Idea. Dynamic Progr. (similar to NFAs → REs)

For $k = 0, 1, \ldots, n\ (= |V|)$ def.

$A^{[k]}[i,j]$:= cost of shortest path $i \rightsquigarrow j$ s.t. all intermediate nodes are $\leq k$ (excluding i, j)

We have the recursion:

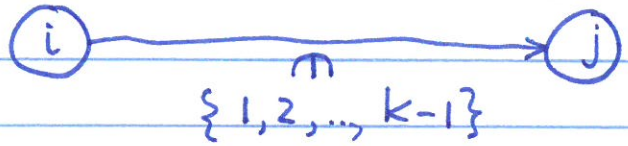
Basis. $A^{[0]}[i,j] = w_{ij}$

Rec. step

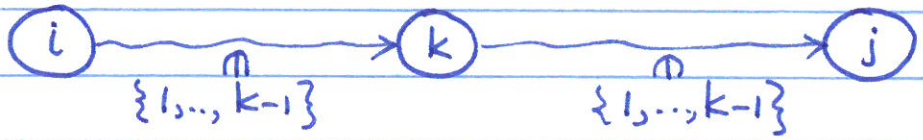
$$A^{[k]}[i,j] = \text{Min}\{A^{[k-1]}[i,j],\ A^{[k-1]}[i,k] + A^{[k-1]}[k,j]\}$$

Illustration. Two cases for $A^{[k]}[i,j]$:

(a)

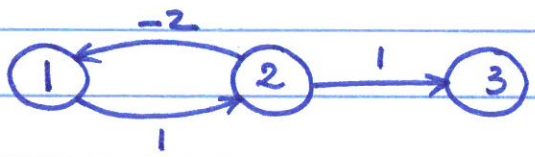


(b)



Assumption. G contains no neg.-weight cycles.

Ex.



$$W = \begin{pmatrix} 0 & 1 & \infty \\ -2 & 0 & 1 \\ \infty & \infty & 0 \end{pmatrix}$$

$$A^{[0]} = \begin{pmatrix} 0 & 1 & \infty \\ -2 & 0 & 1 \\ \infty & \infty & 0 \end{pmatrix} \qquad A^{[1]} = \begin{pmatrix} 0 & 1 & \infty \\ -2 & -1 & 1 \\ \infty & \infty & 0 \end{pmatrix}$$

$$A^{[2]} = \begin{pmatrix} -1 & 1 & 2 \\ -2 & -2 & 1 \\ \infty & \infty & 0 \end{pmatrix}$$

Note there is a neg.-weight cycle (1, 2, 1)

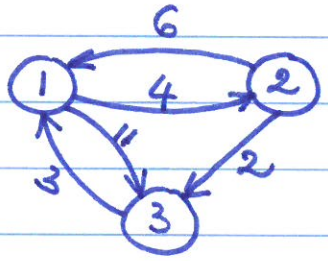
Since $A^{[2]}[1,1] = -1$, $A[1,1] = -\infty$.

Similarly, $A[2,2] = A[1,2] = A[2,1] = -\infty$

□

```
FLOYD_WARSHALL (W, n)
    for i = 1 to n do
        for j = 1 to n do
            A[i,j] = W[i,j]
    for k = 1 to n do
        for i = 1 to n do
            for j = 1 to n do
                A[i,j] = Min {A[i,j], A[i,k]+A[k,j]}
```

Ex.



$$W = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix}$$

$$A^{[0]} = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & \infty & 0 \end{pmatrix} \qquad A^{[1]} = \begin{pmatrix} 0 & 4 & 11 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$$

$$A^{[2]} = \begin{pmatrix} 0 & 4 & 6 \\ 6 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix} \qquad A^{[3]} = \begin{pmatrix} 0 & 4 & 6 \\ 5 & 0 & 2 \\ 3 & 7 & 0 \end{pmatrix}$$

□

Complexity : $O(n^3)$

5.4.2. Transitive Closure

Input. An $n \times n$ Boolean matrix A that is the adj. matrix of a directed graph $G = (V, E)$, i.e.,

$$a_{ij} = 1 \iff (i,j) \in E$$

Output. The transitive closure A^+ s.t.

$$A^+[i,j] = 1 \iff \exists \text{ path } i \rightsquigarrow j$$

```
TRANSITIVE CLOSURE (A, n)
    for k = 1 to n do
        for i = 1 to n do
            for j = 1 to n do
                A[i,j] = A[i,j] or (A[i,k] ∧ A[k,j])
```

5.4.3. A Generalization.

Similarity of the above two algor. (Floyd. Warshall & transitive closure):

- Path weights (costs) are computed in different ways:

(a) $(\mathbb{R}^{\geq 0} \cup \{\infty\}, \text{Min}, +, \infty, 0)$

$\mathbb{R}^{\geq 0}$ = non-neg. reals = costs/weights

Min, + = Operations

(b) $(\{0,1\}, \vee, \wedge, 0, 1)$

Q. What are the main properties of the structures (a) & (b)?

Def. A system $(S, \oplus, \odot, \bar{0}, \bar{1})$ is called a **closed semiring** if

(1) $(S, \oplus, \bar{0})$ and $(S, \odot, \bar{1})$ are monoids

(A set $(B, \cdot, 1)$ is a monoid if the operation $\cdot$ is associative with 1 being the identity element.

Ex. $(\mathbb{Z}, +, 0)$, $(\{0,1\}, \wedge, 1)$.)

(2) $\oplus$ is <u>commutative</u> and <u>idempotent</u>,
(i.e., $\forall a \in S : a \oplus a = a$)

(3) $\odot$ distributes over $\oplus$

(4) For any countable sequence

$$a_1, a_2, \ldots, a_n, \ldots$$

of elements $a_i \in S$, the inf. sum

$$\sum_{i=1}^{\infty} a_i$$ exists and is unique.

Furthermore, associativity, commutativity and idempotence apply to infinite sums.
(For example,

$$\sum_{i=1}^{\infty} a_i \oplus \sum_{i=1}^{\infty} b_i = \sum_{i=1}^{\infty} b_i \oplus \sum_{i=1}^{\infty} a_i$$)

(5) $\odot$ distributes over countably infinite sums.
(e.g., $a \odot \sum_{i=1}^{\infty} b_i = \sum_{i=1}^{\infty} a \odot b_i$)

Ex. (a) $(\mathbb{R}^{\geq 0} \cup \{\infty\}, \text{Min}, +, \infty, 0)$ and $(\{0,1\}, \vee, \wedge, 0, 1)$ are closed semirings

(b) For a finite alphabet Σ, $(2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$ is a closed semiring

($\cdot$ ↑ concatenation)

Notation. Let $(S, \oplus, \odot, \bar{0}, \bar{1})$ be a closed semiring. For $a \in S$ def.

$$a^i = \begin{cases} \bar{1} & \text{if } i = 0 \\ a \odot a^{i-1} & \text{otherwise} \end{cases}$$

The <u>closure oper.</u> * is def. by:

For $a \in S$: $a^* = \sum_{i=0}^{\infty} a^i \quad (= \bar{1} \oplus a^1 \oplus a^2 \oplus \dots)$

Ex. Consider $(2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$.

Let $L \in 2^{\Sigma^*}$ (i.e. $L \subseteq \Sigma^*$ is a language).

$L^0 = \{\varepsilon\}$, $L^1 = L$, $L^2 = L \cdot L = \{uv \mid u, v \in L\}$

$$L^* = L^0 \cup L^1 \cup L^2 \cup \dots = \bigcup_{i=0}^{\infty} L^i$$

is the <u>Kleene closure</u> of L □

Let $G = (V, E)$ be a directed graph and $(S, \oplus, \odot, \bar{0}, \bar{1})$ be a closed semiring.

Let $w : V \times V \to S$ be a weight fctn

The **weight of a path** is the product (w.r.t. $\odot$) of the weights of the edges on that path.

(If path is empty, its weight is $\bar{1}$)

Define for $(i,j) \in V \times V$

$C[i,j]$ = sum of weights of all paths between u and v.

Note. Sum is $\bar{0}$ if there is no paths.

Ex. (1) $(\mathbb{R}^{\geq 0} \cup \{\infty\}, \text{Min}, +, \infty, 0)$

$\oplus$ is Min. Thus, $C[i,j]$ = weight of a "shortest" path $i \rightsquigarrow j$.

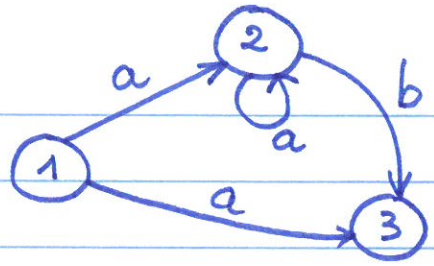
(2) $(\{0,1\}, \vee, \wedge, 0, 1)$

$\oplus$ is $\vee$, so $C[i,j] = 1$ if there exists a path from $i \rightsquigarrow j$

(3) $(2^{\Sigma^*}, \cup, \cdot, \emptyset, \{\varepsilon\})$, $\oplus$ is $\cup$.

Thus, $C[i,j]$ = union of weights (= languages) of paths $i \rightsquigarrow j$. □

Ex.



$C[1,3] = \{a\} \cup \{ aa^n b \mid n \geq 0 \}$

$C[3,1] = \emptyset$ □

The problem can be stated as:

Input. A directed graph $G=(V,E)$, $V=\{1,2,..,n\}$ and a weight fct $w: V \times V \to S$, where $(S, \oplus, \odot, \bar{o}, \bar{1})$ is a closed semiring and $w(i,j) = \bar{o}$ if $(i,j) \notin E$

Output. $C[i,j]$, $(i,j) \in V \times V$.

Idea. (As Floyd-Warshall's alg.)

Define for $0 \leq k \leq n$:

$C^{[k]}[i,j]$ = sum of weights of all paths $i \rightsquigarrow j$ s.t. all interm. vertices (excl. i,j) are in $\{0,..,k\}$.

Then $C^{[0]}[i,j] = w(i,j)$

$$C^{[k]}[i,j] = C^{[k-1]}[i,j] \oplus C^{[k-1]}[i,k] \odot \left(C^{[k-1]}[k,k]\right)^* \odot C^{[k-1]}[k,j]$$

Algorithm (w, n)

for $i = 1$ to n do

$C[i,i] = w[i,i] \oplus \bar{1}$

for $1 \le i, j \le n$, $i \neq j$ do

$C[i,j] = w[i,j]$

for $k = 1$ to n do

for $i = 1$ to n do

for $j = 1$ to n do

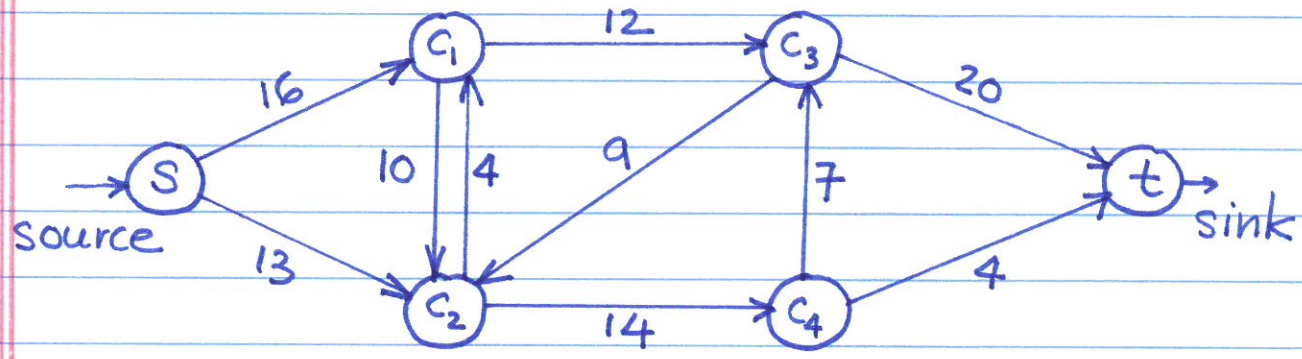
$C[i,j] = C[i,j] \oplus$

$C[i,k] \odot (C[k,k])^{*} \odot C[k,j]$

5.5. Network Flows.

Consider a car manufacturing company:

- Produces cars at a plant (source)
- Ships cars to a destination (sink)
- Each route has limited capacity



- cars can be shipped from s to t overnight and there is no storage space at c_1, c_2, c_3, c_4 and source s.
- The company produces as many cars as they can be shipped from s to t.

Goal. Determine the maximum number of cars that can be shipped from s to t overnight!

Q. What is the maximum flow from s to t?

Def. A flow network is a dir. graph $G = (V, E)$ together with a capacity function $c: V \times V \to \mathbb{R}$ s.t.

(1) There are 2 distinguished vertices s (source) and t (sink)

(2) $c(u,v) \geq 0$ and $c(u,v) = 0$ if $(u,v) \notin E$.

A flow is a function $f: V \times V \to \mathbb{R}$:

(1) $f(u,v) \leq c(u,v)$

(2) $f(u,v) = -f(v,u)$

(3) $\forall u \in V - \{s,t\}: \sum_{v \in V} f(u,v) = 0$

$f(u,v)$ is called net flow from u to v.

The value of flow f is

$$\|f\| = \sum_{v \in V} f(s,v)$$

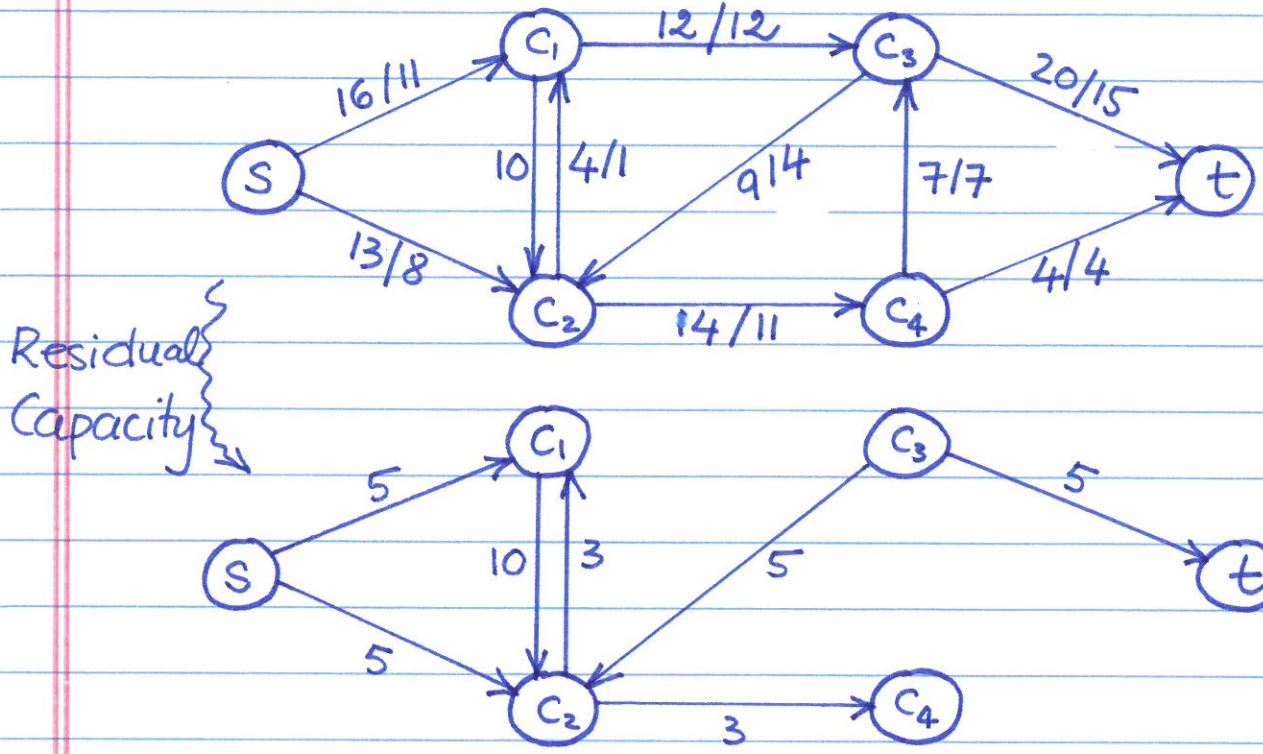
Def. (Maximum Flow problem)

Input. A flow network G with capacity function c.

Output. A flow of max. value from s to t.

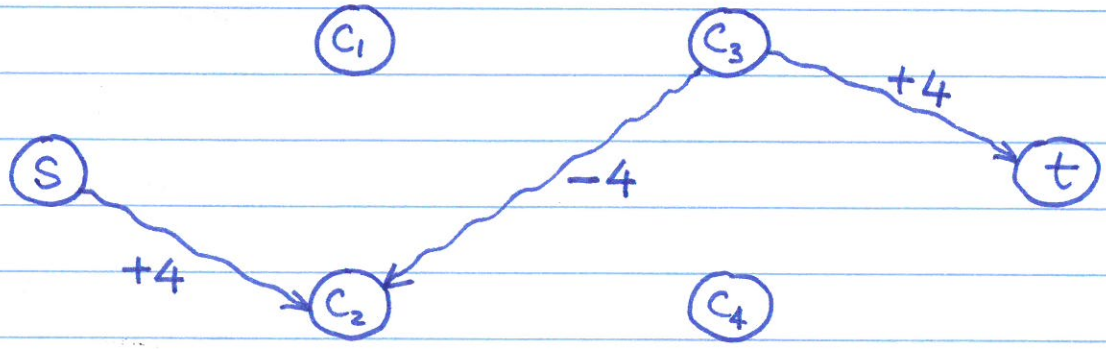
The Ford-Fulkerson Method

Ex. (a/b denotes capacity / flow)

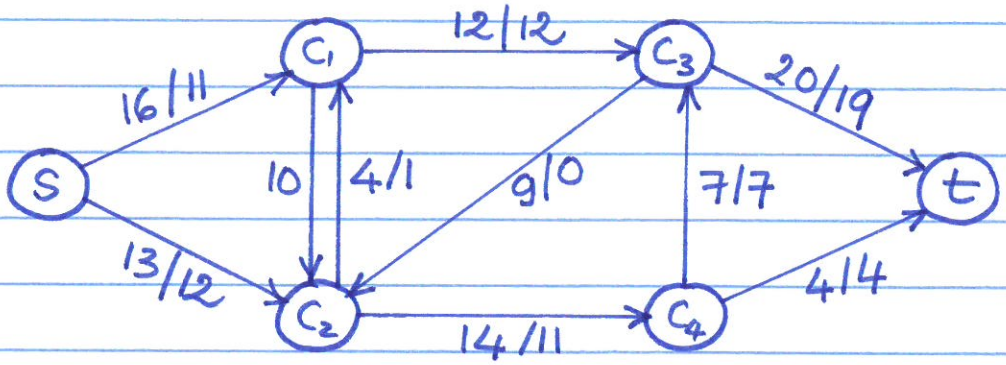


It appears there is no way to increase flow.

Observe: By reversing flow of 4 from c_3 to c_2 we can push 4 units along the path $s \rightarrow c_2 \rightarrow c_3 \rightarrow t$



New flow:



Def. (Residual Network)

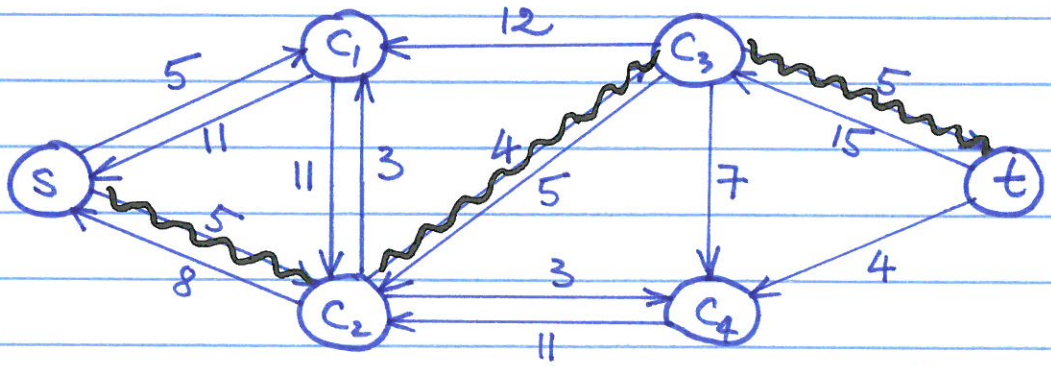
Let $G = (V, E)$ be a network and f be a flow. The <u>residual capacity</u> of (u, v) is

$$c_f(u,v) = c(u,v) - f(u,v)$$

The <u>residual network</u> of G induced by f is $G_f = (V, E_f)$ where

$$E_f = \{ (u,v) \in V \times V \mid c_f(u,v) > 0 \}$$

Ex. (Residual Network)



Observe we can push 4 unit along $s \rightsquigarrow c_2 \rightsquigarrow c_3 \rightsquigarrow t$

Fact. $\text{Card}(E_f) \leq 2 \times \text{Card}(E)$

Lem. Let G_f be residual network of G induced by f. Let f' be a flow in G_f. Then the flow $f + f'$ def. by

$$(f + f')(u,v) = f(u,v) + f'(u,v)$$

is a flow in G.

Moreover, $\|f + f'\| = \|f\| + \|f'\|$ □

Def. (Augmenting paths)

Let G_f be residual network ind. by f. An <u>augmenting path</u> p is a simple path from s to t in G_f.

The max. amount of net flow that can be pushed along an augmenting path p is

$$c_f(p) = \min\{ c_f(u,v) \mid (u,v) \text{ is on } p \}$$

$c_f(p)$ is the <u>residual capacity</u> of p. (Note $c_f(p) > 0$.)

Lem. Let $f_p : V \times V \to \mathbb{R}$ be def. by

$$f_p(u,v) = \begin{cases} c_f(p) & \text{if } (u,v) \in p \\ -c_f(p) & \text{if } (v,u) \in p \\ 0 & \text{otherwise} \end{cases}$$

Then f_p is a flow in G_f, and

$$\|f_p\| = c_f(p) \qquad \square$$

Cor. $f' = f + f_p$ is a flow in G.
Moreover, $\|f'\| > \|f\|$. $\square$

Ford-Fulkerson (G, c, s, t)

```
for all u,v ∈ V do
    f(u,v) = 0
G_f = G
while ∃ augmenting path p in G_f do
    augment flow f along p
    update f
return f
```

Cuts of Flow Networks

Def. A cut (S,T) of a network $G=(V,E)$ is a partition $V = S \cup T$ s.t. $s \in S$, $t \in T$.

Let f be a flow in G. Then the net flow across the cut (S,T) is

$$f(S,T) = \sum_{x \in S,\, y \in T} f(x,y).$$

The capacity of the cut (S,T) is

$$c(S,T) = \sum_{x \in S,\, y \in T} c(x,y)$$

Lem. Let f be a flow in G and (S,T) be a cut. Then $f(S,T) = \|f\|$. □

Cor. Let f be a flow in G and (S,T) be a cut. Then $\|f\| \leq c(S,T)$ □

Theorem. (Max-Flow Min-Cut Theorem)

The following are equivalent:

(1) f is a maximum flow

(2) The res. network G_f has no augm. path

(3) $\|f\| = \text{Min}\{c(S,T) \mid (S,T) \text{ is a cut}\}$.

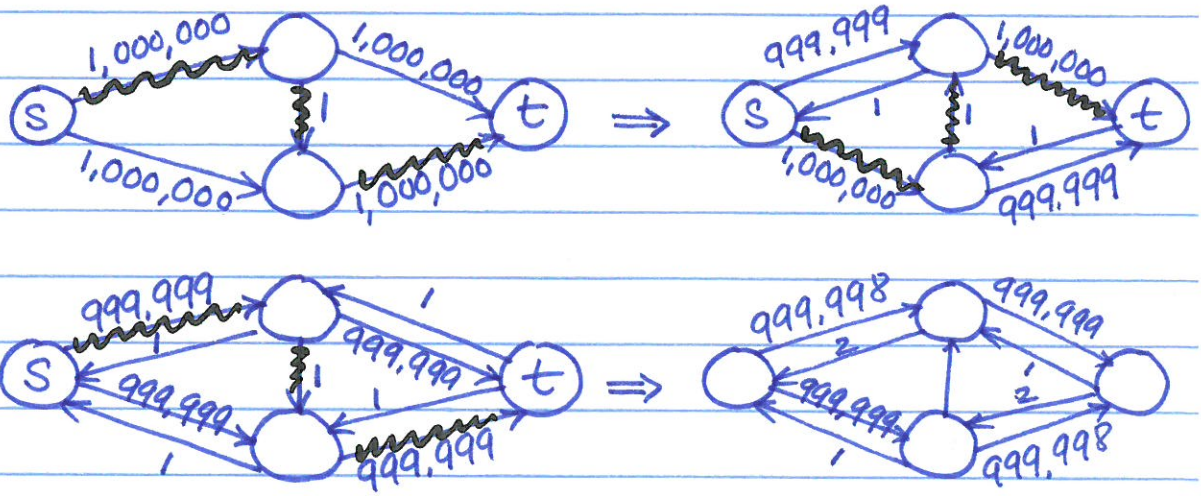
Correctness of Ford_Fulkerson algor. follows from Max. Flow Min. Cut Theor.

Q. What is the complexity of the Ford_Fulkerson algor. ?

A. $\Theta(E \cdot \|f^*\|)$ where f^* is max flow.

Reason: The while_loop increases flow f by at least 1 unit
$\Rightarrow$ while_loop is executed at most $\|f^*\|$ times.

Ex.



Edmonds_Karp Algor.

Idea: Use BFS to find augm. paths:
An augm. path in G_f is therefore a shortest path (# of edges) from s to t.

Notation. $\delta_f(u,v)$ = length of shortest path from u to v in G_f.

Lem. $\forall v \in V - \{s,t\}$, $\delta_f(s,v)$ in the res. network G_f increases monotonically with each flow augmentation.

Pf. (By contradiction.)

Suppose $\exists v \in V-\{s,t\}$ and a flow augmentation s.t. $\delta_{f'}(s,v) < \delta_f(s,v)$.

Let v be such a vertex s.t. $\delta_{f'}(s,v)$ is smallest, i.e., for all $u \in V-\{s,t\}$:

(*) $\delta_{f'}(s,u) < \delta_{f'}(s,v) \Rightarrow \delta_{f'}(s,u) \geq \delta_f(s,u)$

Let p' be a shortest path in $G_{f'}$ from s to v, and u be a vertex on p' that immediately preceeds v:

p': 

Clearly, $\delta_{f'}(u) = \delta_{f'}(v) - 1$

Therefore, by (*), $\delta_{f'}(s,u) \geq \delta_f(s,u)$

Edmonds-Karp Algor.

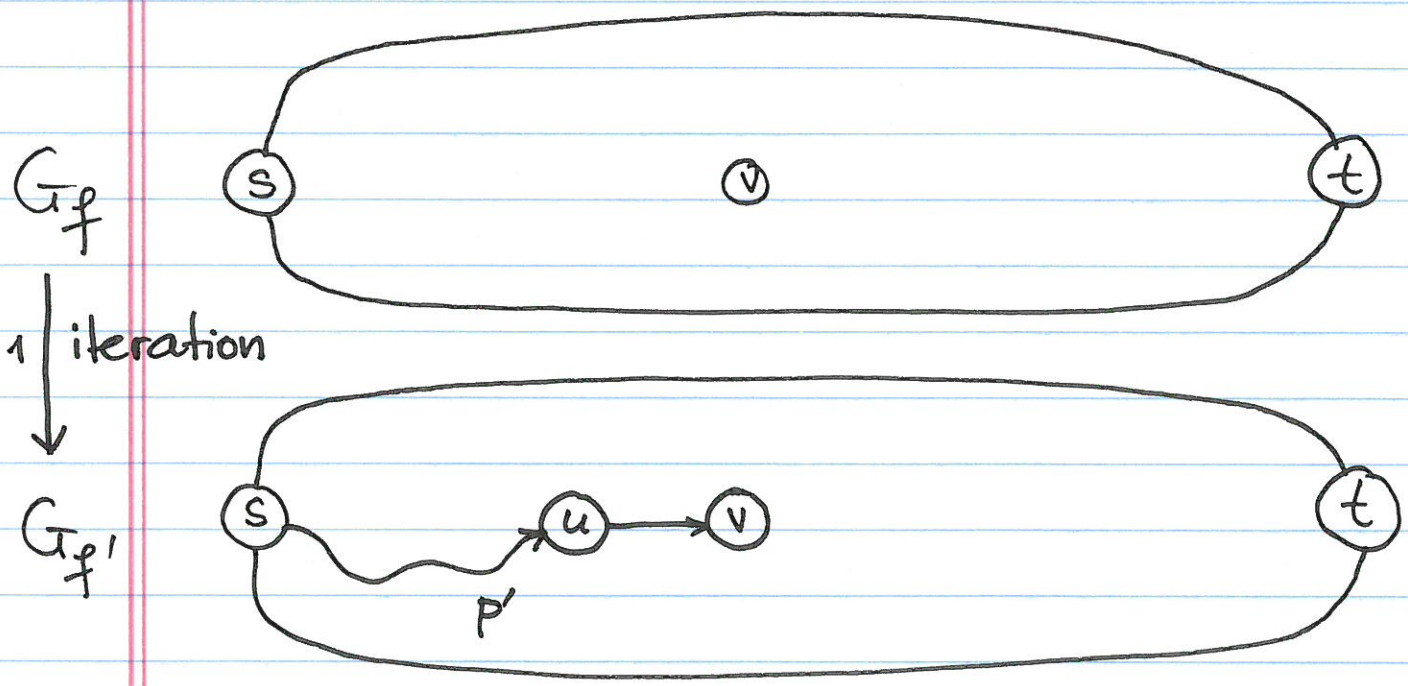
Proof of Lemma. (By contradiction)

Suppose that Lem. is not true, i.e., there exist $v \in V - \{s,t\}$ and a flow augmentation $f \to f'$ s.t.

⊛ shortest path from s to v decreases, i.e.,

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

$\Rightarrow$ choose v s.t. $\delta_{f'}(s,v)$ is shortest

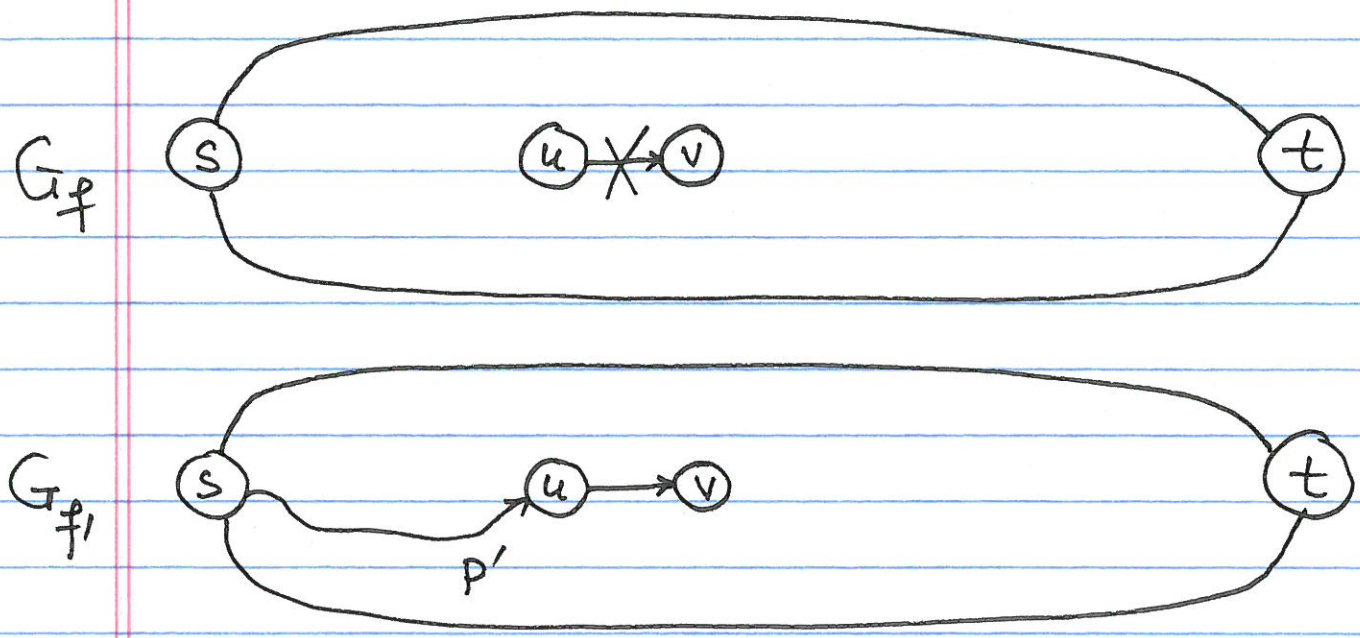


Let p' be shortest path from $s \to v$ in $G_{f'}$ and u be node preceding v.

By the choice of v, it holds:

$$\delta_f(s,u) \leq \delta_{f'}(s,u)$$

Therefore, in G_f the edge (u,v) doesn't exist:



If (u,v) did exist in G_f, then

$$\begin{aligned}\delta_f(s,v) &\leq \delta_f(s,u) + 1 \\ &\leq \delta_{f'}(s,u) + 1 \\ &\leq \delta_{f'}(s,v) \quad \lightning \text{ Contrad.}\end{aligned}$$

Fact. Augm. path chosen in G_f must involve edge (v,u).

Otherwise edge (u,v) would not appear in $G_{f'}$!

(Why? Answer: Otherwise there would be no flow from v to u in f' in order to create (u,v) in $G_{f'}$.)

Thus, in G_f we have:

$$\delta_f(s,u) = \delta_f(s,v)+1$$

since augm. path is a shortest path.

Now,
$$\begin{aligned}\delta_f(s,v) &= \delta_f(s,u)-1\\ &\leq \delta_{f'}(s,u)-1\\ &= \delta_{f'}(s,v)-1-1\\ &< \delta_{f'}(s,v)\end{aligned}$$

This contradicts the assumption (*) that $\delta_{f'}(s,v) < \delta_f(s,v)$. □

We claim that $f(u,v) \neq c(u,v)$.
$=$
Otherwise,

$$\begin{aligned} \delta_f(s,v) &\leq \delta_f(s,u)+1 \\ &\leq \delta_{f'}(s,u)+1 \\ &= \delta_{f'}(s,v) \rightsquigarrow \text{Contrad.} \end{aligned}$$

Thus, $f(u,v) = c(u,v)$

$\Rightarrow \quad (u,v) \notin E_f$

Hence, the augmenting path chosen in G_f involving edge** (u,v) must contain (v,u) in the direction $v \rightarrow u$.

Thus, $\delta_f(s,u) = \delta_f(s,v)+1$

Therefore, $\delta_f(s,v) = \delta_f(s,u)-1$

$$\begin{aligned} &\leq \delta_{f'}(s,u)-1 \\ &= \delta_{f'}(s,v)-2 \\ &< \delta_{f'}(s,v) \end{aligned}$$

$\rightsquigarrow$ Contrad. □

** Since p' in $G_{f'}$ contains (u,v), aug. path through u in G_f must involve (u,v) in direction $v \rightarrow u$.

Theor. The revised (Edmonds-Karp) Ford-Fulkerson algorithm has complexity $O(V \cdot E^2)$.

Moreover, the total number of flow augmentations by the algor. is $O(V \cdot E)$

Pf. We only need to prove 2nd statement.

Def. An edge (u,v) in G_f is <u>critical</u> on an augmenting path p if $c_f(p) = c_f(u,v)$.

Fact. After augmenting flow along an augmenting path, critical edges disappear in next residual network.

Claim. An edge $(u,v) \in E$ can be critical at most $O(V)$ times.

Pf. (Claim).

Let f be a flow when (u,v) becomes critical the first time. Then

$$\delta_f(s,v) = \delta_f(s,u) + 1.$$

For the next flow f', we have

$f'(u,v) = c(u,v)$ and (u,v) disappear in the residual network $G_{f'}$.

Observe that (u,v) cannot reappear later on another augmenting path until after (v,u) appears on an augm. path.

Let f'' be flow when this event occurs. Then,

$$\begin{aligned} \delta_{f''}(s,u) &= \delta_{f''}(s,v) + 1 \\ &\geq \delta_f(s,v) + 1 \quad \text{(Lemma)} \\ &= \delta_f(s,u) + 1 + 1 \\ &= \delta_f(s,u) + 2. \end{aligned}$$

Thus, each time (u,v) becomes critical, the distance from s to u increases by at least 2.

Since the largest distance is $\leq |V|-2$, (u,v) can be critical at most $O(V)$ times.

Thus, the Claim is proved.

As there are only $O(E)$ edges in all residual networks, there are at most $O(E \cdot V)$ augmentations. □

5.6. Maximum Bipartite Matching

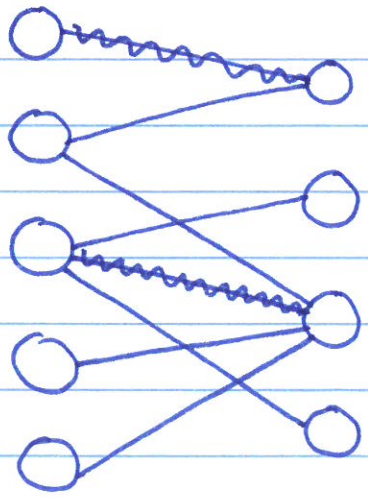
Def. Let $G=(V,E)$ be an undir. graph. A subset $M \subseteq E$ is called a <u>matching</u> if $\forall v \in V$: at most one edge in M is incident on v.

M is a <u>maximum matching</u> if M is a matching of maximum cardinality

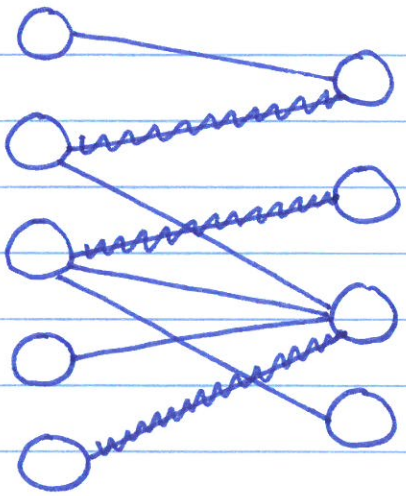
A graph $G=(V,E)$ is <u>bipartite</u> if there is a partition $V = L \cup R$ s.t. $E \subseteq L \times R$.

Goal. To construct maximum bipartite matching!

Ex.



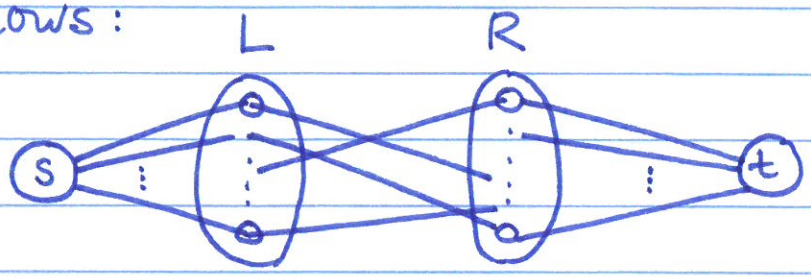
A matching



A maximum matching

Idea. (Network Flows)

Given $G = (V, E)$ with $V = L \cup R$ and $E \subseteq L \times R$, constr. a flow network $G' = (V \cup \{s, t\}, E')$, $c(e) = 1 \ \forall e \in E'$ as follows:



Def. A flow f is <u>integer-valued</u> if

$$f(u,v) \in \mathbb{Z} \quad \forall u, v \in V$$

Lem. Let G be a bipartite graph and G' be its corresp. flow network.

If M is a matching in G, then there is an integer-valued flow f in G' s.t.

$$\|f\| = |M|.$$

Conversely, if f is an integer-valued flow in G', then there is a matching M in G s.t. $|M| = \|f\|$. □

⇒ To obtain max. matching compute max flow.

Q. Can a maximum flow have non-integral values?

Lem. Let c be a capacity function with only integer values. Then the maximum flow computed by Ford-Fulkerson algor. has only integer values, i.e., $f(u,v) \in \mathbb{Z} \quad \forall u,v \in V$.

Pf. By induction on number of iterations. □

Theorem. Maximum bipartite matching can be solved in $O(V \cdot E)$ time.

Pf. Note that max. flow value is $O(V)$. □