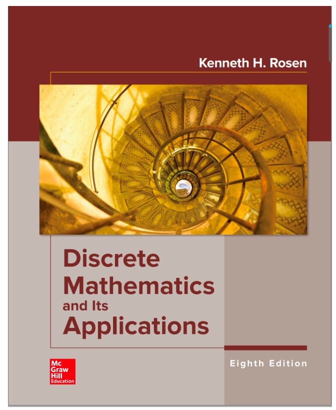
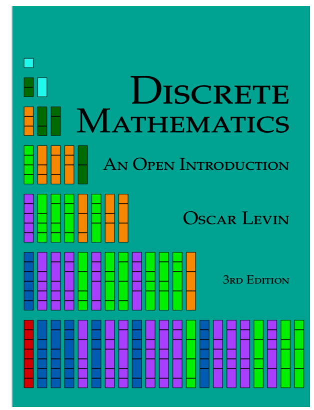
Vietnam National University of HCMC
International University
School of Computer Science and Engineering





Optimal Problem Solving on Graphs

Assoc. Prof. Dr. Nguyen Van Sinh

nvsinh@hcmiu.edu.vn

Outline

- Finding the shortest path
  - Dijkstra algorithm
  - Floyd algorithm
- The minimum spanning tree
  - Concepts and theorems
  - Prim algorithm
  - Kruskal algorithm
- Finding the maximum flow (extend)
  - Ford-Fulkerson algorithm

Finding the shortest path

- Dijkstra algorithm
- Floyd algorithm

Dijkstra Algorithm

Dijkstra's algorithm is used in problems relating to find the shortest path from a to z on the weighted graph G=(V,E,W), including the label for each vertex (node).

Each node is given a temporary label denoting the length of the shortest path *from* the start node *so far*.

This label is replaced if another shorter route is found.

Once it is certain that no other shorter paths can be found, the temporary label becomes a permanent label. When vertex z has permanent label Dz, then Dz is the shortest path.

Eventually all the nodes have permanent labels.

At this point the shortest path is found by retracing the path backwards.

Dijkstra Algorithm

- Step 1: $T = V$; $D_a = 0$; $D_i = \infty$, $v_i \neq a$.
- Step 2: Repeat until $z \notin T$:
  - take vertex v_i (which has D_i is smallest) out of T.
  - labeled for all v_j in T and v_j adjacent to v_i following the formula:

$$D_j = \min\{D_j, D_i + W_{ij}\}$$

Dijkstra Algorithm

The table has the following columns

- T: set of vertices with temporary label
- v_i: the vertex which take out of T at each step
- D_j: length of shortest path $a \rightarrow v_j$.

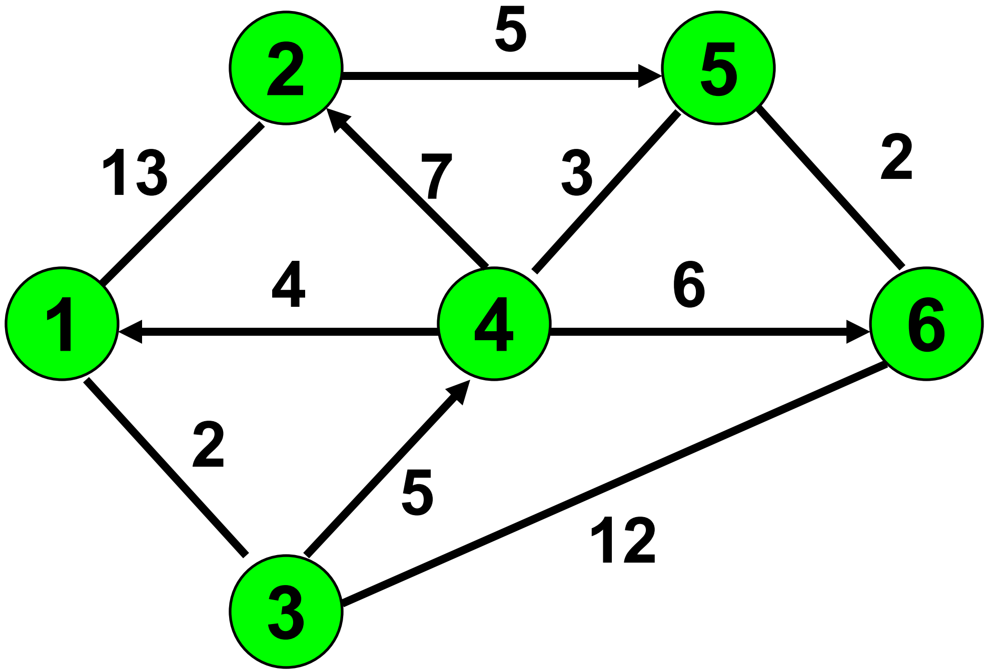
Dijkstra Algorithm

ALGORITHM 1 Dijkstra's Algorithm.

procedure *Dijkstra*(G: weighted connected simple graph, with
all weights positive)
{G has vertices $a = v_0, v_1, \ldots, v_n = z$ and lengths $w(v_i, v_j)$
where $w(v_i, v_j) = \infty$ if $\{v_i, v_j\}$ is not an edge in G}
for $i := 1$ **to** n
    $L(v_i) := \infty$
$L(a) := 0$
$S := \emptyset$
{the labels are now initialized so that the label of a is 0 and all
other labels are ∞, and S is the empty set}
while $z \notin S$
    $u :=$ a vertex not in S with $L(u)$ minimal
    $S := S \cup \{u\}$
    for all vertices v not in S
        if $L(u) + w(u, v) < L(v)$ **then** $L(v) := L(u) + w(u, v)$
        {this adds a vertex to S with minimal label and updates the
        labels of vertices not in S}
return $L(z)$ {$L(z) =$ length of a shortest path from a to z}

Weighted matrix

Example: given the graph G = (V, E, W), find the shortest path from v_1 to v_6.



Dijkstra Algorithm

T	V_i	D_1	D_2	D_3	D_4	D_5	D_6
{1..6}	-	0	∞	∞	∞	∞	∞
{2..6}	v_1	*	13	2	∞	∞	∞
{2, 4..6}	v_3	-	13	*	7	∞	14
{2,5,6}	v_4	-	13	-	*	10	13
{2, 6}	v_5	-	13	-	-	*	12
{2}	v_6	-	13	-	-	-	*

Dijkstra Algorithm

Based on the above table, comeback from v_6 to v_1, we have the shortest path.

$P = v_6 \leftarrow v_5 \leftarrow v_4 \leftarrow v_3 \leftarrow v_1$

The length of the shortest path is $D_6 = 12$.

Dijkstra Algorithm

Evaluation:

In order to obtain the shortest path from a to all vertices, replace the loop "repeat until $z \notin T$" by "repeat until $T = \varnothing$" (T: set of vertices with temporary label).

From the above table, add one more step, we have the shortest path from v_1 to all vertices

Dijkstra Algorithm

Evaluation:

We can also label for each vertex v_j, a pair of labels $[D_j, v_i]$ with:

D_j is the length of shortest path $a \to v_j$.

v_i is a vertex before v_j on the shortest path.

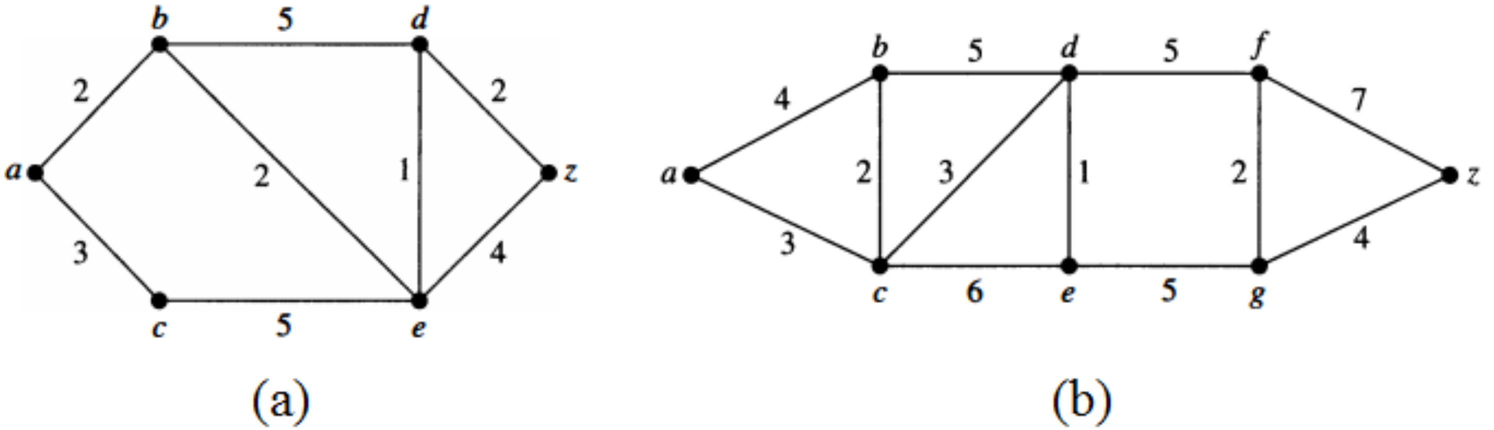
The second label to get the shortest path. With the above example, we have the table as follows:

Dijkstra Algorithm

T	v_1	v_2	v_3	v_4	v_5	v_6
{1..6}	$[0,v_1]$	$[\infty,v_1]$	$[\infty,v_1]$	$[\infty,v_1]$	$[\infty,v_1]$	$[\infty,v_1]$
{2..6}	*	$[13,v_1]$	$[2,v_1]$	$[\infty,v_1]$	$[\infty,v_1]$	$[\infty,v_1]$
{2, 4..6}	-	$[13,v_1]$	*	$[7,v_3]$	$[\infty,v_1]$	$[14,v_3]$
{2,5,6}	-	$[13,v_1]$	-	*	$[10,v_4]$	$[13,v_4]$
{2, 6}	-	$[13,v_1]$	-	-	*	$[12,v_5]$
{2}	-	$[13,v_1]$	-	-	-	*

Example:

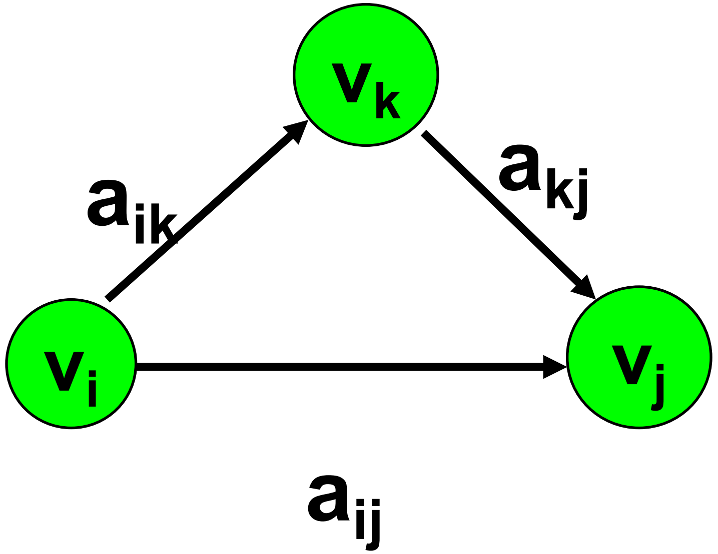
Find the length of a shortest path between a and z in the following weighted graphs based on Dijkstra's algorithm



Floyd Algorithm

In order to find the shortest path between a pair of vertices and store the length in a weighted matrix $A = (a_{ij})n_xn$

The algorithm perform n steps.



Step *k* let $a_{ij}=\min\{ a_{ij}, a_{ik}+a_{kj}\}$

Floyd Algorithm

```
void Floyd( )
{
   for (k=0; k<n; k++)
            for (i=0; i<n; i++)
               for (j=0; j<n; j++)
                      if (a[i][k] + a[k][j] < a[i][j])
                               a[i][j] = a[i][k] + a[k][j];
}
```

Full Floyd Algorithm

ALGORITHM 2 Floyd's Algorithm.

procedure *Floyd*(G: weighted simple graph)
{G has vertices $v_1, v_2, \ldots, v_n$ and weights $w(v_i, v_j)$
with $w(v_i, v_j) = \infty$ if (v_i, v_j) is not an edge}
for $i := 1$ **to** n
for $j := 1$ **to** n
$d(v_i, v_j) := w(v_i, v_j)$
for $i := 1$ **to** n
for $j := 1$ **to** n
for $k := 1$ **to** n
if $d(v_j, v_i) + d(v_i, v_k) < d(v_j, v_k)$
then $d(v_j, v_k) :=$
$d(v_j, v_i) + d(v_i, v_k)$
{$d(v_i, v_j)$ is the length of a shortest path between v_i
and v_j}

The minimum spanning tree

- The Concepts and theorems
- Prim algorithm
- Kruskal algorithm

The concepts

- Tree is a connected undirected graph without cycles.
- Given an undirected graph G=(V,E), spanning tree T of graph G is a sub-graph that includes all of the vertices of G and T is a tree.
- Given an undirected graph G=(V,E,W), minimum spanning tree of graph G is spanning tree which has the smallest weight in all spanning of G.

The theorems

- Theorem 1: Suppose T=(V,E) is an undirected graph n vertices. The following propositions are equivalent:
  - T is a tree;
  - T has no cycles and has n-1 edges;
  - T connected and has n-1 edges.
- Theorem 2: G has spanning tree if and only if G connected.

Prim Algorithm

- Step 1: T := {v}; with any v
- Step 2: Loop n-1 times:
  - Find fringe vertex v with edge e, connect T with weight w(e) smallest.
  - Put e and v into T.

Prim Algorithm

Example: given a graph with a weighted matrix as bellows:

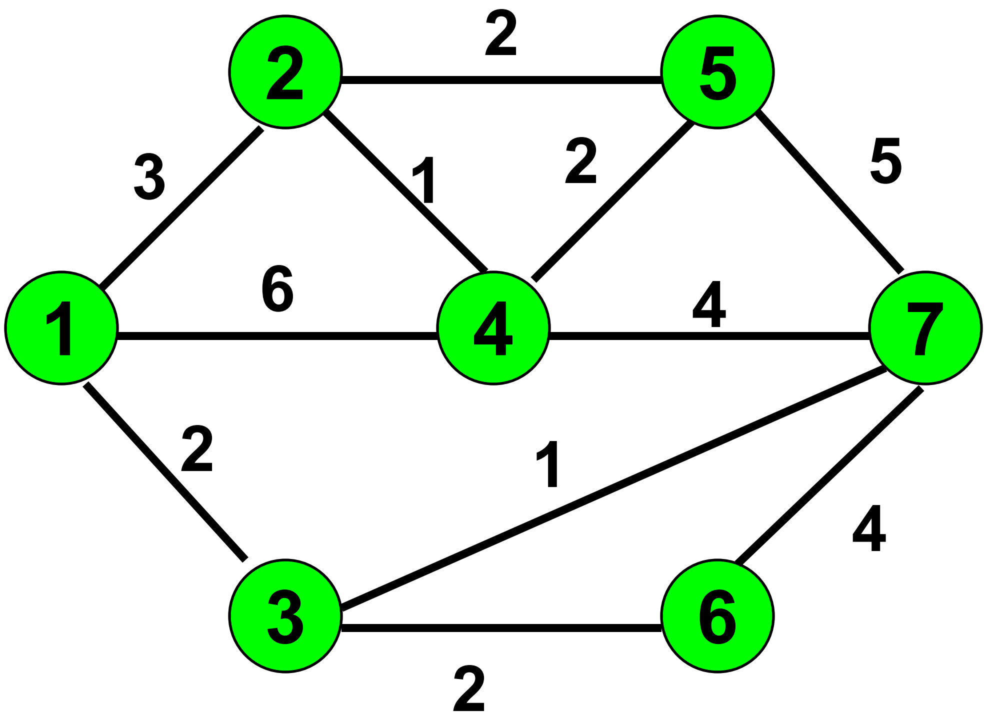
	v1	v2	v3	v4	v5	v6
v1	0	33	17	∞	∞	∞
v2	33	0	18	20	∞	∞
v3	17	18	0	16	4	∞
v4	∞	20	16	0	9	8
v5	∞	∞	4	9	0	14
v6	∞	∞	∞	8	14	0

Prim Algorithm

E_T	v1	v2	v3	v4	v5	v6
-	*	[33,v1]	[17,v1]	[∞,v1]	[∞,v1]	[∞,v1]
(1,3)	-	[18,v3]	*	[16,v3]	[4,v3]	[∞,v1]
(3,5)	-	[18,v3]	-	[9,v5]	*	[14,v5]
(5,4)	-	[18,v3]	-	*	-	[8,v4]
(4,6)	-	[18,v3]	-	-	-	*
(3,2)	-	*	-	-	-	-

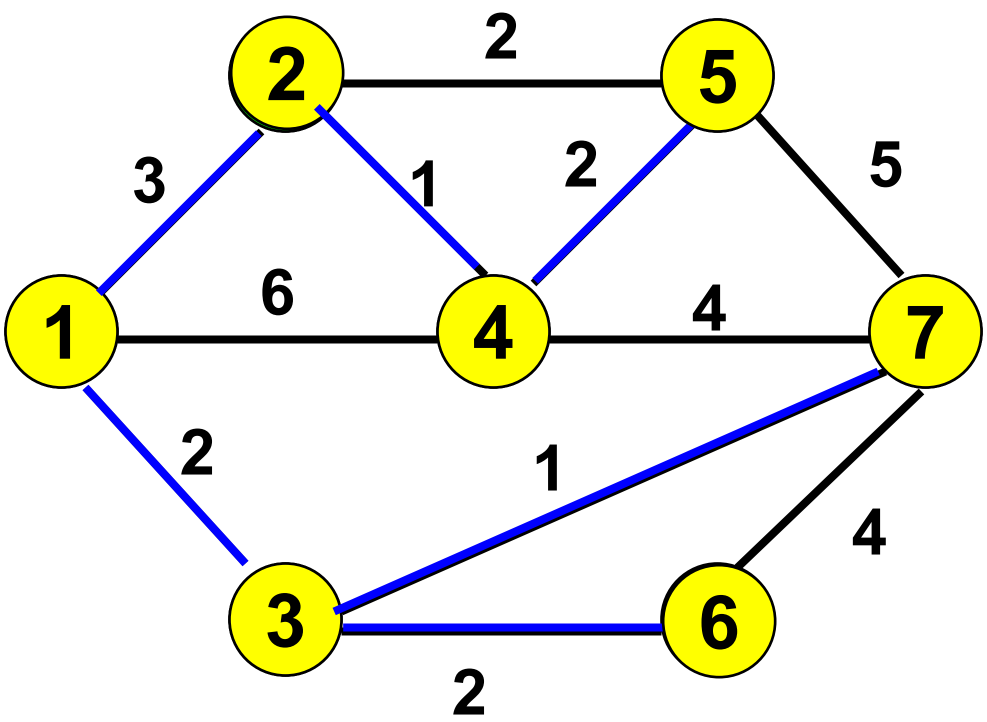
Prim Algorithm

Example: we can present as below figure:



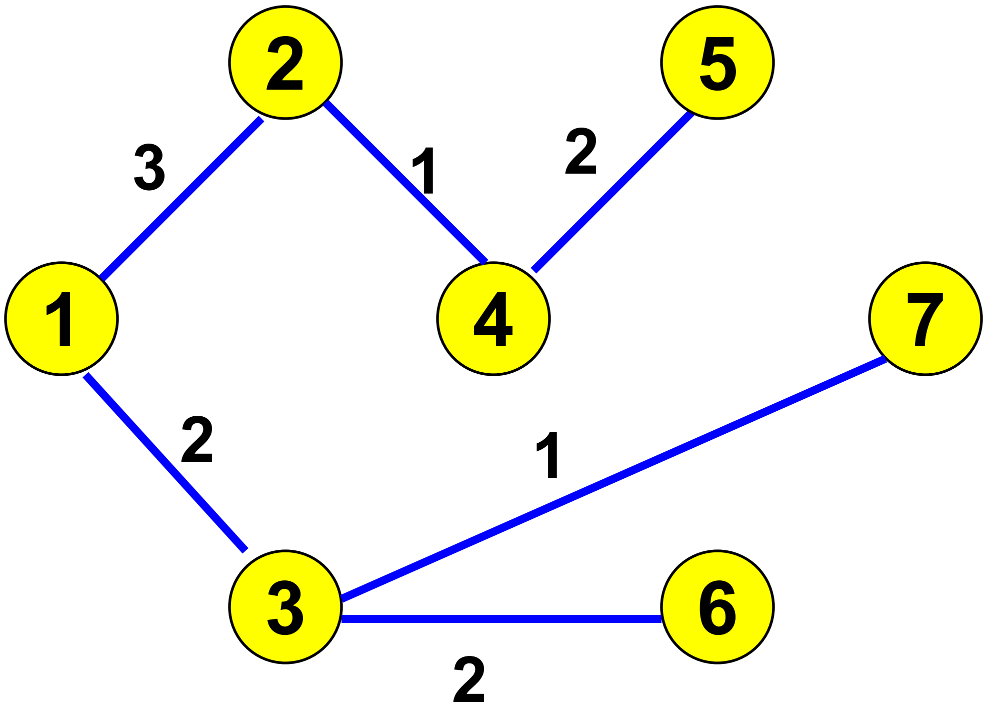
Prim Algorithm

Example: we can present as below figure:



Prim Algorithm

The smallest spanning tree T with W(T) = 11

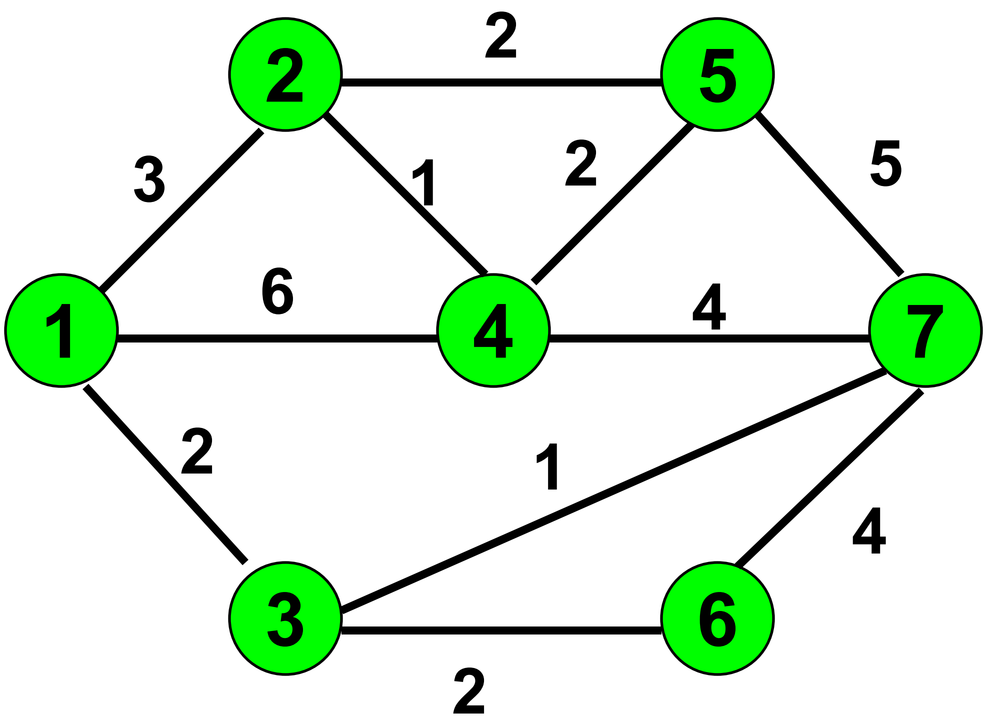


Kruskal Algorithm

- *Step 1* T := V; T has no edge
- *Step 2* Loop *n-1* times:
  - Find edge e which has the smallest weight and put into T without creating cycles.
  - Put e into T
  - Starting, sort the edges increasing of the weights

Kruskal Algorithm

Example:



Kruskal Algorithm

Sort the edges increasing of the weights.

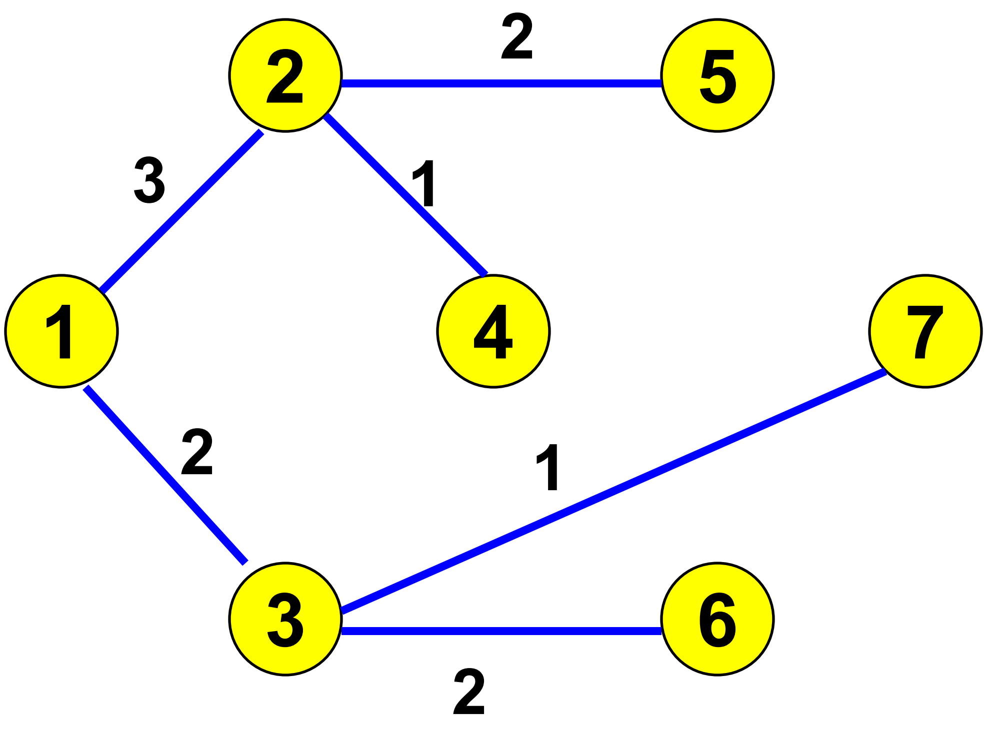
e	(2,4)	(3,7)	(1,3)	(2,5)	(3,6)	(4,5)	(1,2)	(4,7)	(6,7)	(5,7)	(1,4)
w_e	1	1	2	2	2	2	3	4	4	5	6
e_T	1	2	3	4	5	-	6	-	-	-	-

The tree T includes 6 edges as follows:

(2,4), (3,7), (1,3),(2,5),(3,6),(1,2).

Kruskal Algorithm

The smallest spanning tree T with W(T) = 11



Finding the maximum flow

Extend to self-study:

- The concepts
- Ford-Fulkerson algorithm

Concepts

- Network is a weighted, directed graph, G = (V,E,C):
  - G is weak connected graph (if remove direction then it is connected).
  - There is only one vertex s without input arcs called "output vertex" and only one vertex t without output arcs called "input vertex".
  - Each arc(i,j) is assigned a number $c_{ij} \geq 0$ called "able to through" of arc(i,j)

Concepts

- Flow $F=(f_{ij})$ on network $G=(V,E,C)$ is the assignment for each arc(i,j) a number f_{ij} which has satisfied:
  - Every arc(i,j) has: $0 \leq f_{ij} \leq c_{ij}$
  - Every vertex v_i different from s and t has the total number of input flows = output flows.
  - Therefore, the total number of output flows from s = the total number of input flows t called "flow value", noted v_F
  - Maximum flow on network G is the flow which has the largest value in all flows on G.

Ford-Fulkerson Algorithm

- Step 1: F=0 //initial flow 0, $\forall(i,j)$ has $f_{ij} = 0$
- Step 2: Repeat until out of the paths for increasing flow:
  - Find the path for increasing flow P from s to t, with the increasing number ∂
  - Increase the flow following P a number ∂.

Ford-Fulkerson Algorithm

The obtained path of increasing flow P as bellows:

P: $s \rightarrow .. \rightarrow i \rightarrow j \rightarrow .. \rightarrow t$ (i,j is a positive arc)

P: $s \rightarrow .. \rightarrow i \leftarrow j \rightarrow .. \rightarrow t$ (j,i is a negative arc)

Ford-Fulkerson Algorithm

The step of finding flow increasing P can use the way to label as follows:

- Set the label s is ∞
- Repeat until t has label ∂t: when vertex v_i has just labeled, then labeled for all v_j (adjacent to v_i) if satisfy one of the two cases bellows:

Ford-Fulkerson Algorithm

- If there is arc(i,j) and $c_{ij} - f_{ij} > 0$ then set $\partial_j = \min\{\partial_i, c_{ij} - f_{ij}\}$, input positive arc(i,j) into P.
- If there is arc(j,i) and $f_{ji} > 0$ then set $\partial_j = \min\{\partial_i, f_{ji}\}$, input negative arc(j,i) into P.

When t has label ∂t, the number of flow increasing $\partial = \partial_t$. After increase flow, delete label.

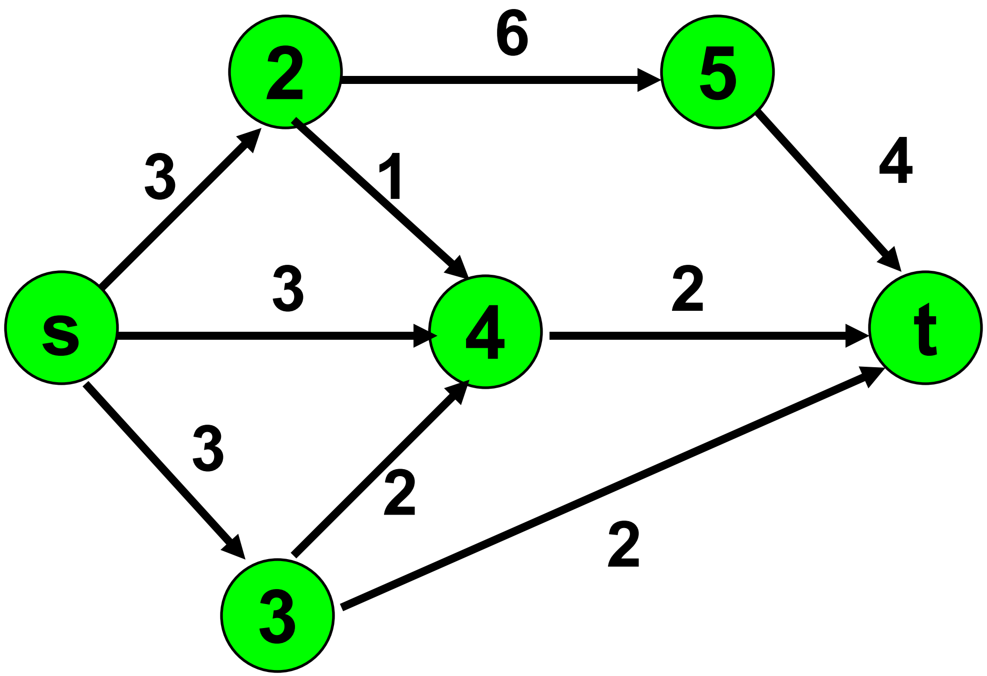
Ford-Fulkerson Algorithm

Increase flows following P a number of ∂ based on formula bellows:

$F_{ij}' = Fij + \partial$	If arc(i,j) is a positive arc
$F_{ij}' = Fij - \partial$	If arc(i,j) is a negative arc
F_{ij}	If arc(i,j) out of P

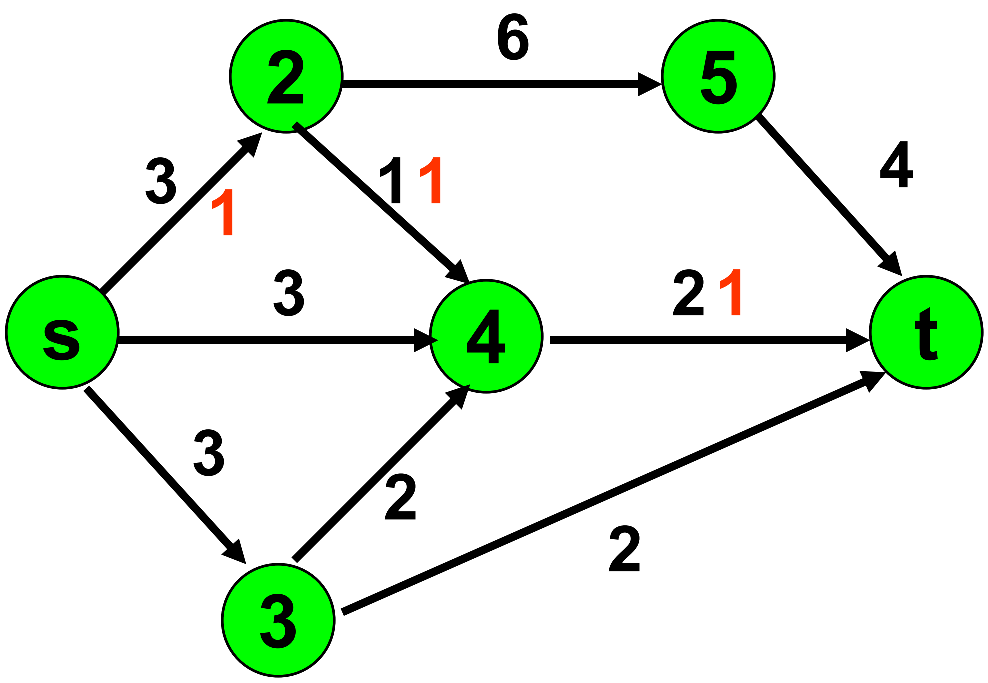
Ford-Fulkerson Algorithm

Example: given a network G = (V,E,C)



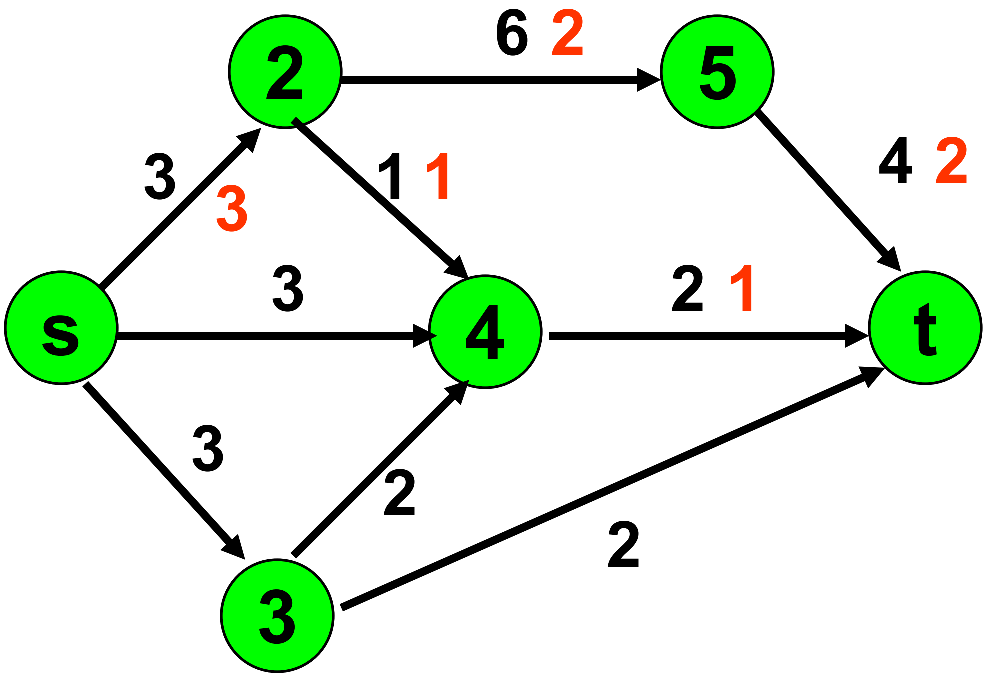
Ford-Fulkerson Algorithm

P_1: $s \rightarrow 2 \rightarrow 4 \rightarrow t$, $\partial_1 = 1$



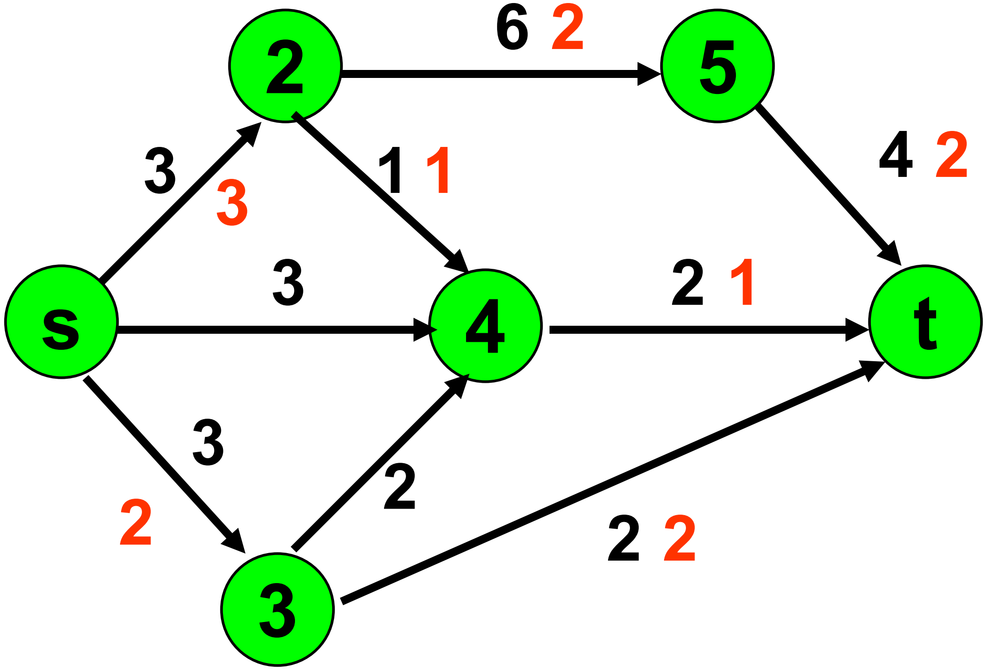
Ford-Fulkerson Algorithm

P_2: $s \rightarrow 2 \rightarrow 5 \rightarrow t$, $\partial_2 = 2$



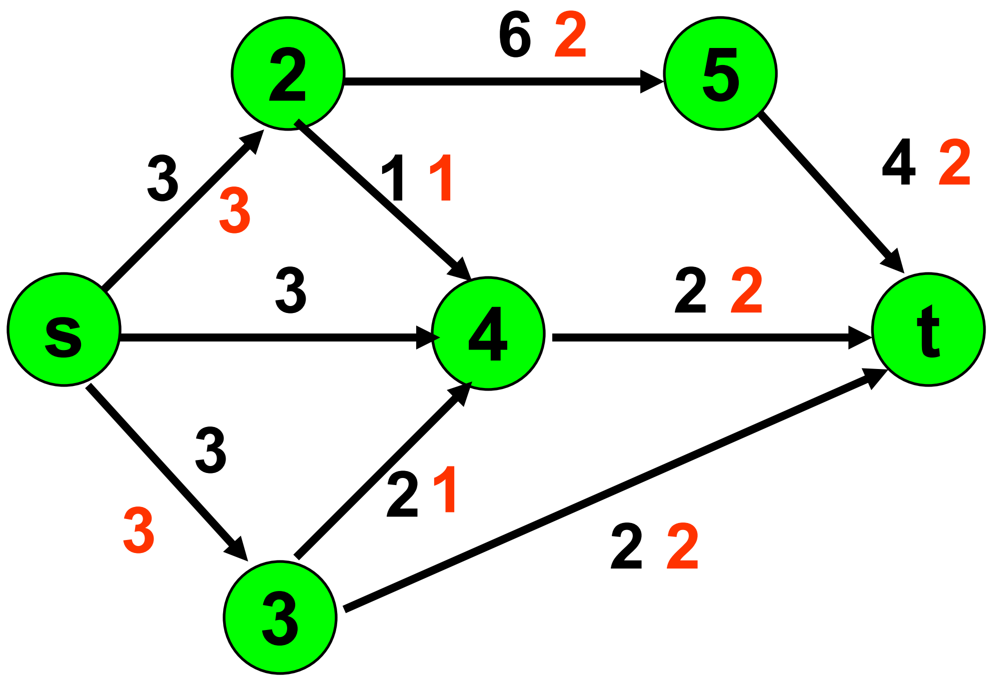
Ford-Fulkerson Algorithm

P_3: s$\rightarrow$3 $\rightarrow$t, ∂_3=2.



Ford-Fulkerson Algorithm

P_4: $s \rightarrow 3 \rightarrow 4 \rightarrow t$, $\partial_4 = 1$.

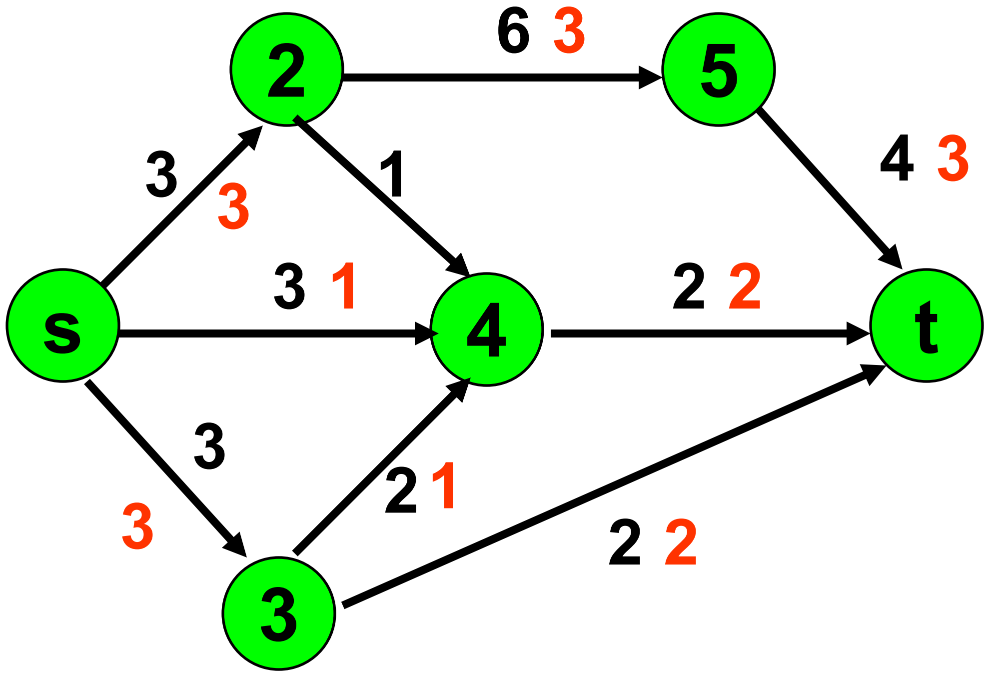


Ford-Fulkerson Algorithm

P_5: s→4 ← 2 → 5 → t, ∂_5=1.

Out of flow increasing path, $F_{max} = 7$.

Minimum cut: V_1={s,3,4},V_2={t,2,5}.



Homework (practice yourself for FE, no submission)

1. Implement the Dijkstra's algorithm, Floyd using C/C++ with the following requirements:
   - Enter the number of vertices
   - The adjacent matrix is entered from keyboard
   - Input the start vertex
   - Input the stop vertex
   - Show the shortest path between start and stop vertex.
2. Implement Kruskal algorithm.
3. Study yourself the Ford-Fulkerson algorithm

Refer: http://www.geeksforgeeks.org/ford-fulkerson-algorithm-for-maximum-flow-problem/