Graph Optimization
(4541.554 Introduction to Computer-Aided Design)

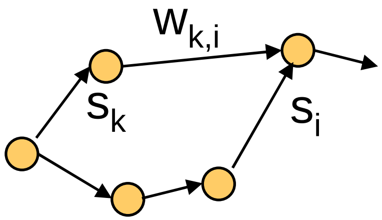
School of EECS
Seoul National University

Shortest (Longest) Path Problems

- **Assume no negative (positive) cycles**

  negative (positive) cycles + simple path --> NP-complete
- **Directed edge weighted graph G(V, E, W), source vertex v_0**
- **Bellman's equation**
  - **path weight $s_0 = 0$**

$$s_i = \min_{k \neq i}(s_k + w_{k,i}) \,, \quad i = 1, 2, \ldots, n$$

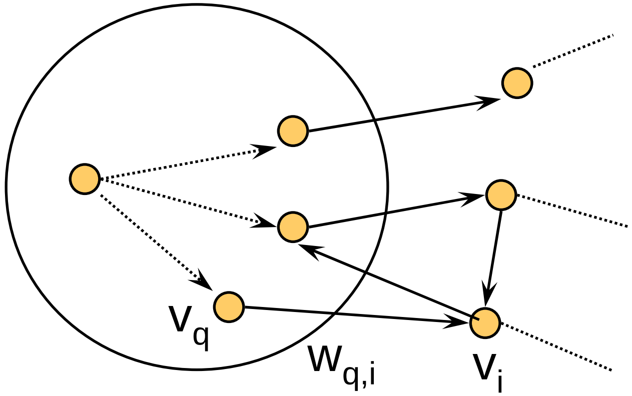


- **Acyclic**
  - **Topological sort (O(|V| + |E|))**
  - **Solve Bellman's equation in the topological order**

– **Cyclic**

- **if all weights are positive, use Dijkstra's algorithm**
- **DIJKSTRA (G(V, E, W)) {**

```
    s_0 = 0;
    for (i = 1 to n)
        s_i = w_{0,i};
    repeat {
        select an unmarked vertex v_q such that s_q is minimal;
        mark v_q
        foreach (unmarked vertex v_i)
            s_i = min{s_i, (s_q + w_{q,i})};
    } until (all vertices are marked);
}
```



Implementation of priority queue

- **linear list: $O(|V|^2 + |E|)$**
- **heap: $O(|V|\log|V| + |E|\log|V|)$**

- **Cycles + negative weights (but no negative cycle)**
  - **Bellman-Ford algorithm**
    - **Refine path weights iteratively**
    - **BELLMAN_FORD (G(V, E, W)) {**

      $s_0^1 = 0$;

      for (i = 1 to n) $s_i^1 = w_{0,i}$;

      for (j = 1 to n) {

          for (i = 1 to n) {

              $s_i^{j+1} = \min_{k \neq i}\{s_i^j , (s_k^j + w_{k,i})\}$; O(|E|)

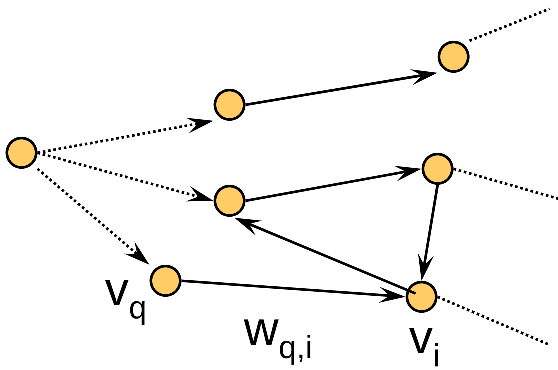
          }

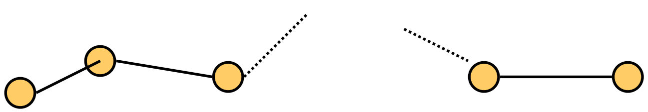
          if (s_i^{j+1} == s_i^j for all i) return (TRUE);

      }

      return (FALSE)

      }
    - **Converge within |V| - 1 iterations**





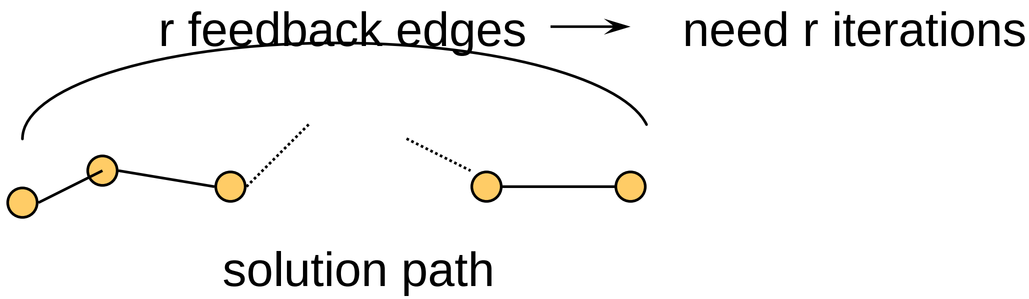
solution path

    - **O(|V| |E|)**

- **Liao- Wong**
  - **$G(V, E \cup F, W)$, F: set of feedback edges**
  - **LIAO_WONG ($G(V, E \cup F, W)$) {**

```
for (j = 1 to |F| + 1) {
    foreach vertex v_i
        l_i^{j+1} = longest path in G(V, E, W_E) ; -- O(|V|+|E|+|F|)
    flag = TRUE;
    foreach edge (v_p, v_q) ∈ F {
        if (l_q^{j+1} < l_p^{j+1} + w_{p,q}) {
            flag = FALSE;
            E = E ∪ (v_0, v_q);
            w_{0,q} = (l_p^{j+1} + w_{p,q});
        }
    }
    if (flag) return (TRUE)
}
return (FALSE)
}
```

– **converge within |F| + 1 iterations**



– **O((|V| + |E| + |F|) |F|)**

- **Shortest path lengths between all pairs of vertices**
  - **Floyd's algorithm**

for k=1 to n

for i=1 to n

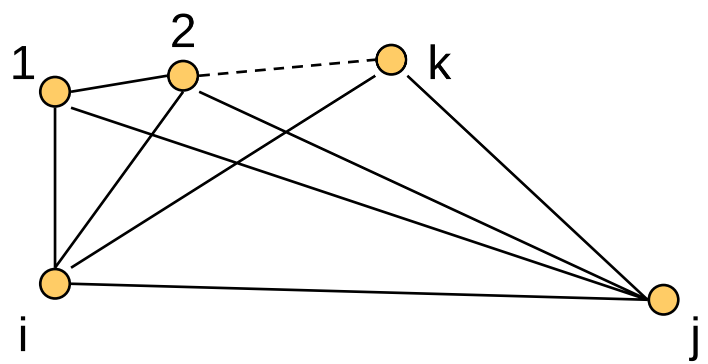
for j =1 to n

if $d_{ik} + d_{kj} < d_{ij}$

$d_{ij} = d_{ik} + d_{kj}$;

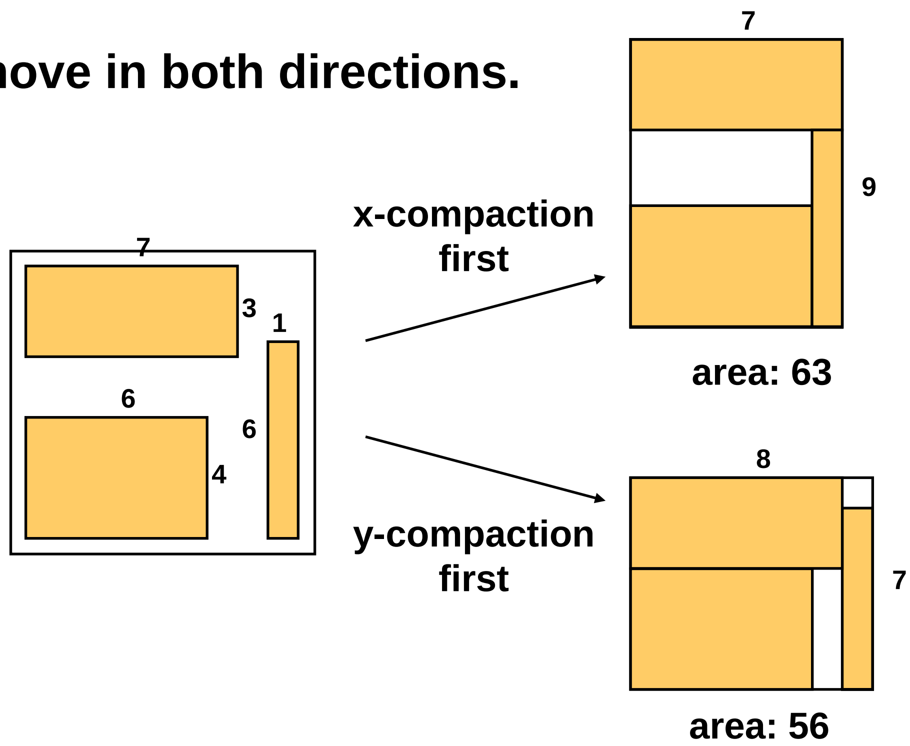
$$
\begin{array}{c|cc}
 & j & k \\
\hline
i & d_{ij} & d_{ik} \\
k & d_{kj} &
\end{array}
$$

d_{ij} is the length of the shortest among the paths that pass thru only the vertices with labels ≤ k

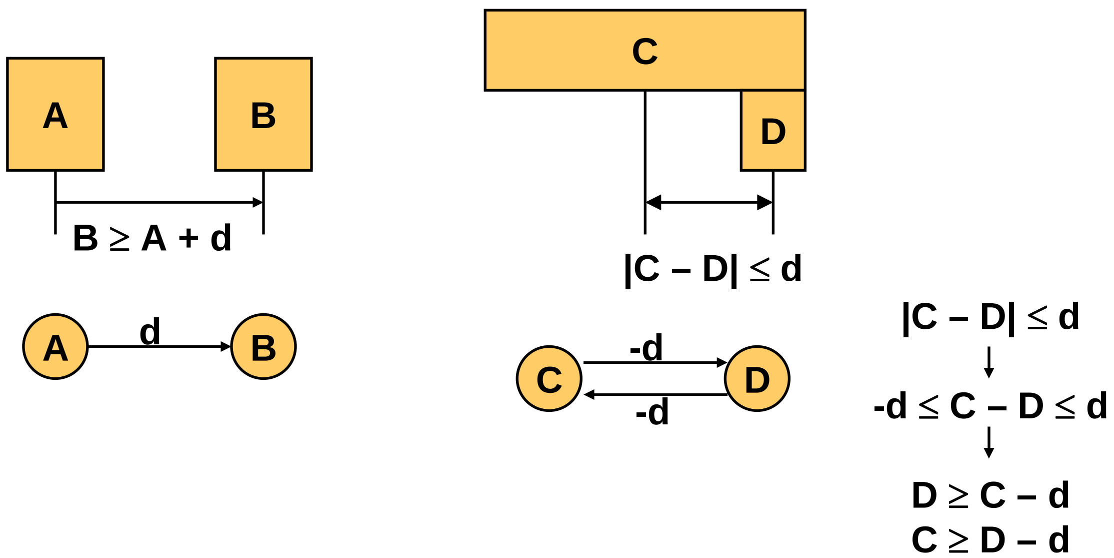


Compaction

- Minimize area satisfying the design rule
- Two dimensional problem
- Classification
  - 1-D compaction
    - Alternate x-compaction and y-compaction
  - 2-D compaction
    - Objects can move in both directions.
    - NP-hard

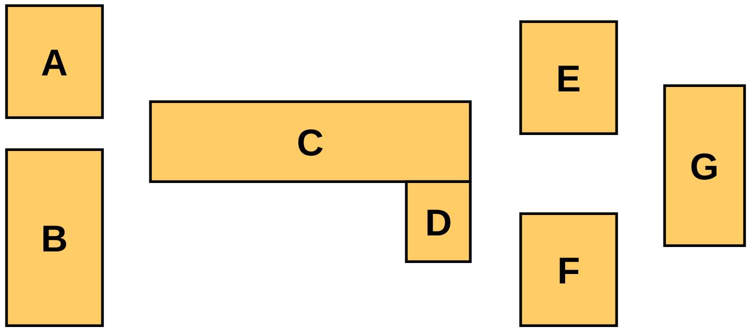


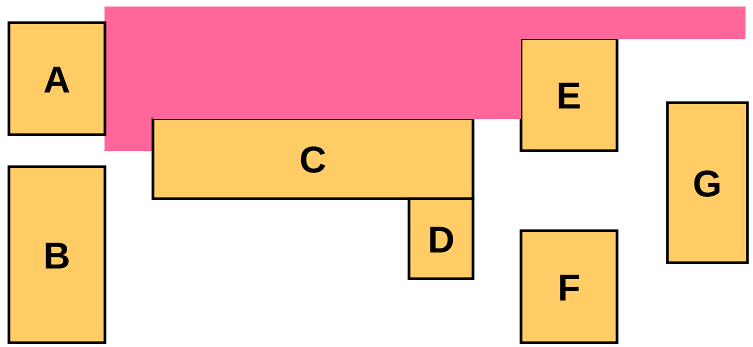
- **Compaction based on Constraint Graph**
  - **Graph construction**
    - **Nodes: objects**
    - **Directed edges: constraints**
    - **Edge weights: lower bounds (spacing)**
      upper bounds (connectivity)
      slacks
    - **Add a source and a sink**



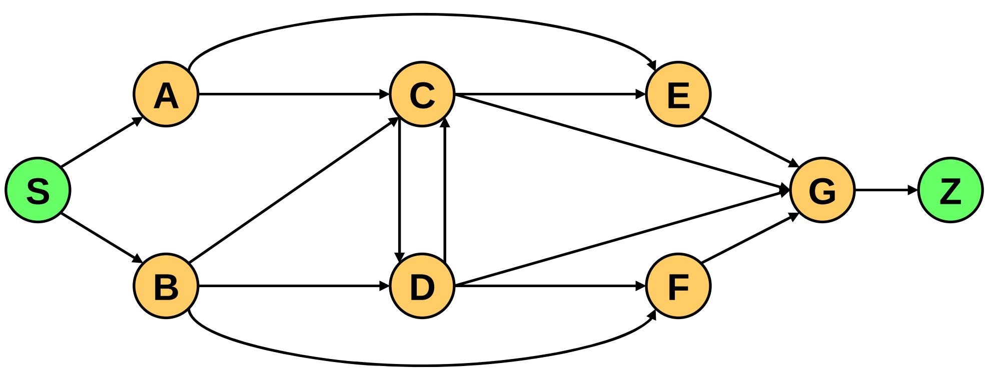
  - **Find the longest path (critical path) from the source (leftmost object) to the sink (rightmost object)**

– **Example**



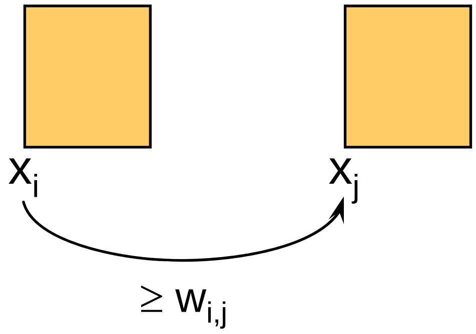


Shadow propagation to avoid generating unnecessary constraints (edges)



– Linear program

- **constraint graph : $x_j \geq x_i + w_{i,j}$**



minimize $c^T x$, where $c^T = [1, ..., 1]$ or

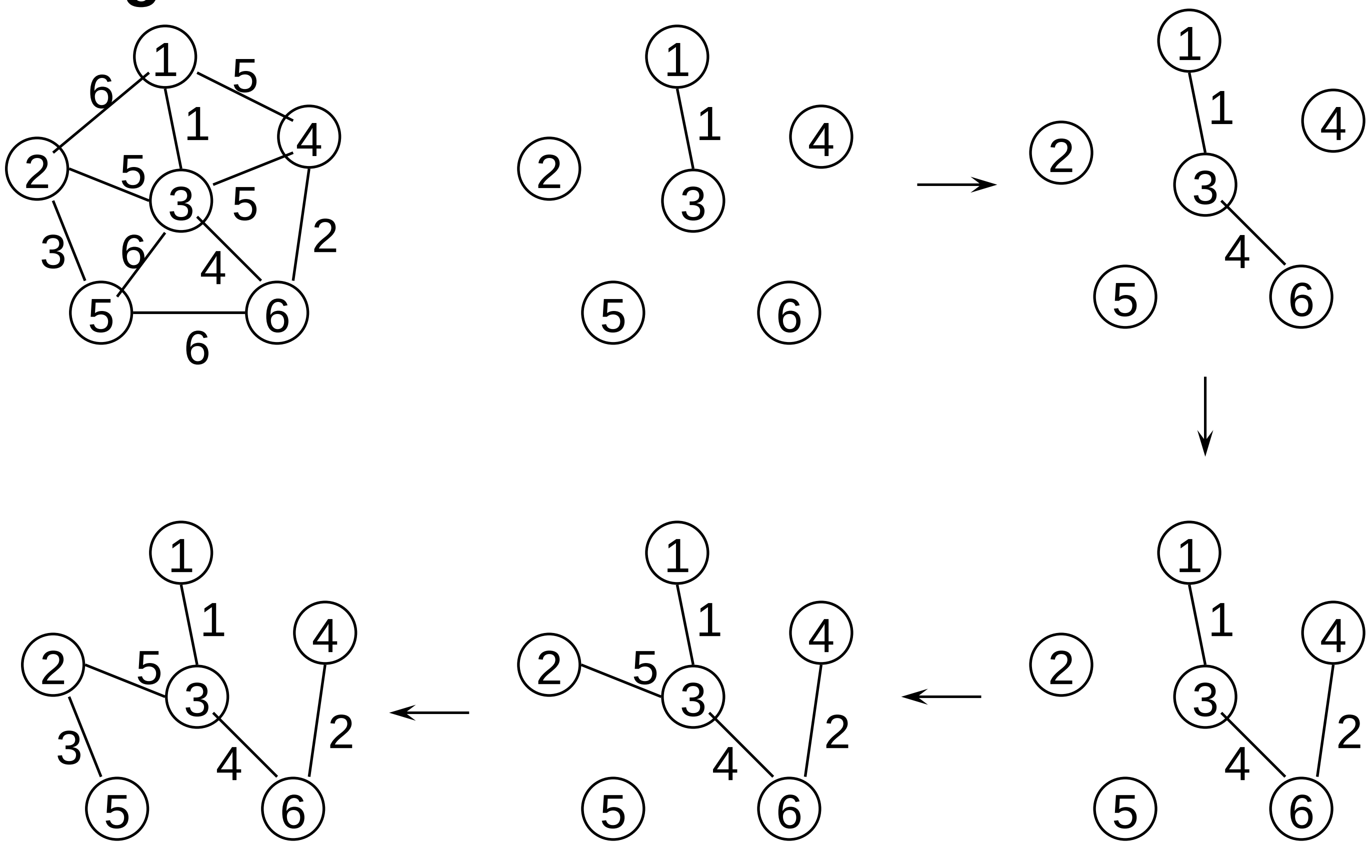
$c_i = 1$ for sink and $c_i = 0$ otherwise

subject to $A^T x \geq b$, A : incidence matrix

$$A^T x = \begin{array}{cc} & \begin{array}{cc} v_i & v_j \end{array} \\ e_{i,j} & \begin{bmatrix} -1 & 1 \\ & \\ & \end{bmatrix} \end{array} x \geq \begin{bmatrix} w_{i,j} \\ \\ \\ \end{bmatrix}$$

Shortest Spanning Tree

- **Spanning tree of G**
  - **Subgraph of G that is a tree containing all vertices of G**
  - **Can be used for net length estimation**
- **Prim's algorithm**



- PRIM (G(V, E, W)) {

    mark v_1;

    for (i = 2 to n)

        $s_i = w_{1,i;}$

    repeat {

        select an unmarked vertex v_q such that s_q is minimal; --- $|V|^2$, $|V|\log|V|$

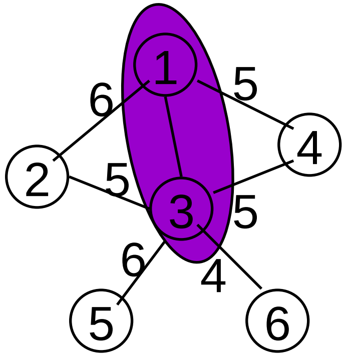
        mark v_q

        foreach (unmarked vertex v_i)

            $s_i = \min\{s_{i,}\ w_{q,i}\}$; --- $|E|$, $|E|\log|V|$

    } until (all vertices are marked);

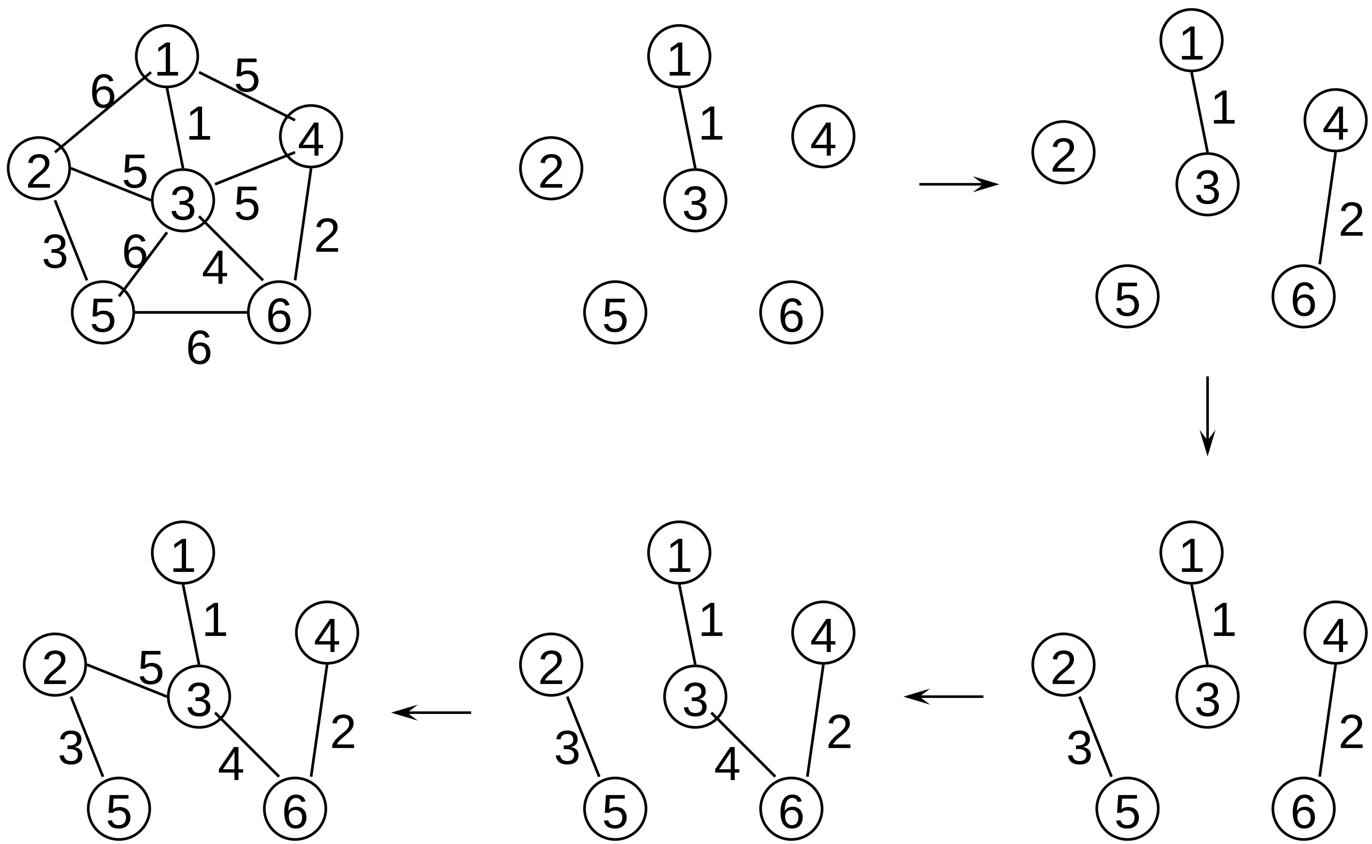
  }



implementation of priority queue

- linear list: $O(|V|^2 + |E|)$
- heap: $O(|V|\log|V| + |E|\log|V|)$

– **Kruskal's algorithm**



- **KRUSKAL (G(V, E, W)) {**

  make n components such that each component contains one vertex;

  ncomp = n;

  repeat {

  select an unmarked edge $e_{i,j}$ such that $w_{i,j}$ is minimal; --- $|E|^2$, |E|log|E|

  icomp = find(i, components); --- |E|log|V|

  jcomp = find(j, components); --- |E|log|V|

  if icomp <> jcomp {

  merge(icomp, jcomp, components);

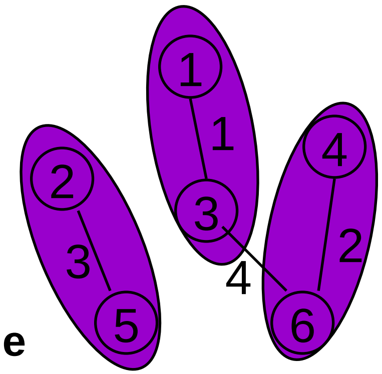
  ncomp = ncomp - 1;

  mark $e_{i,j}$

  }

  } until (ncomp = 1);

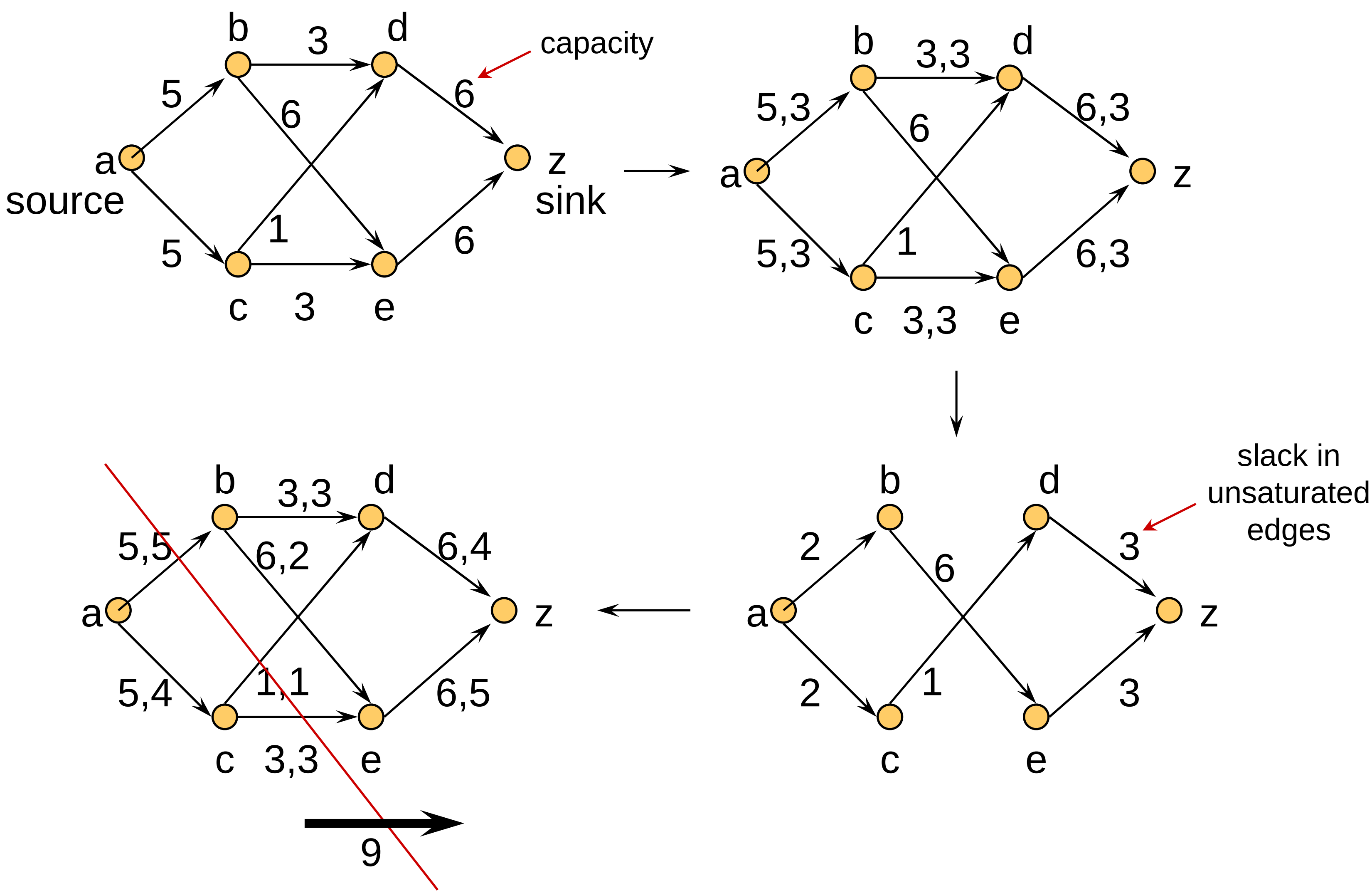
  }

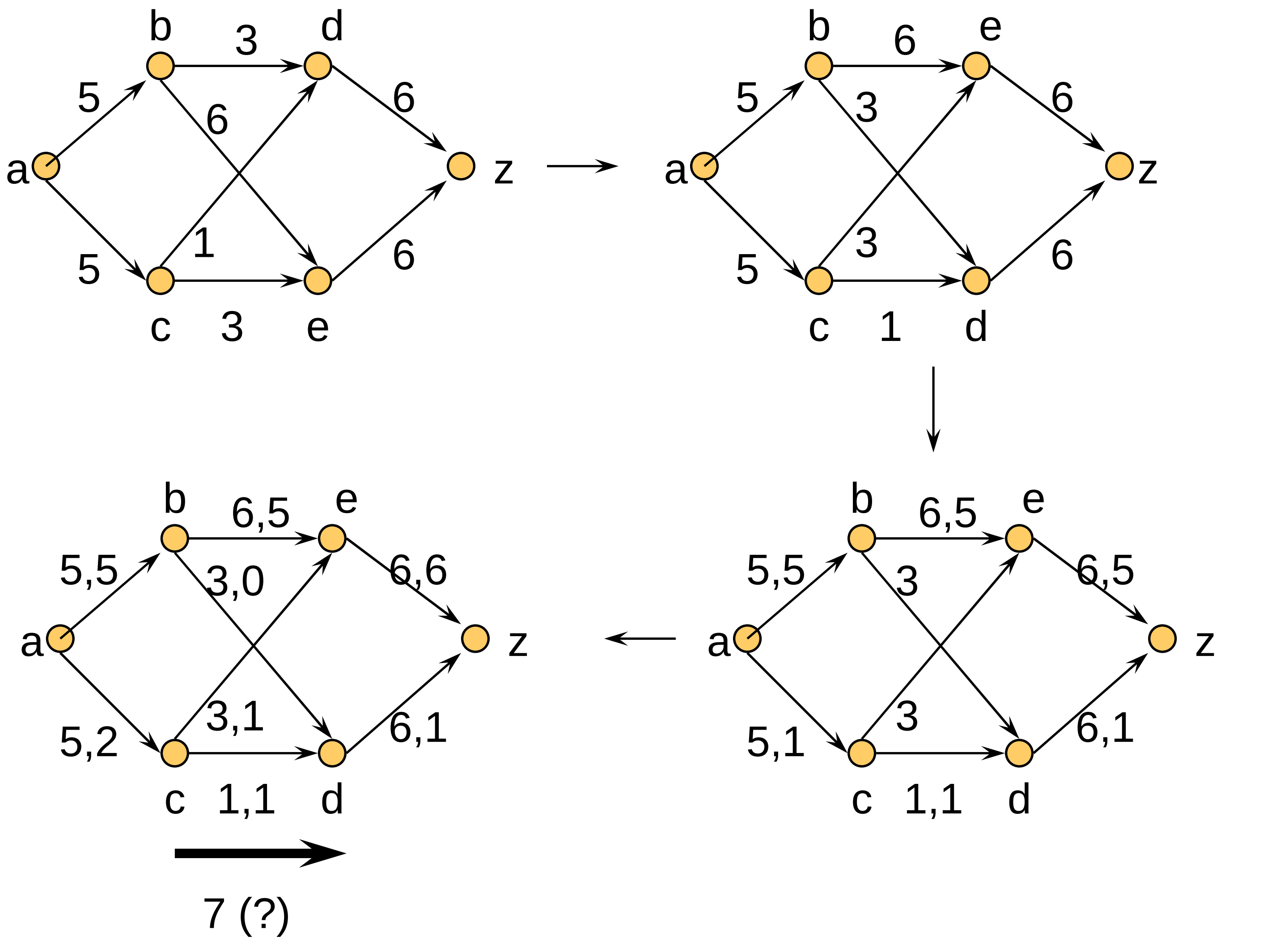


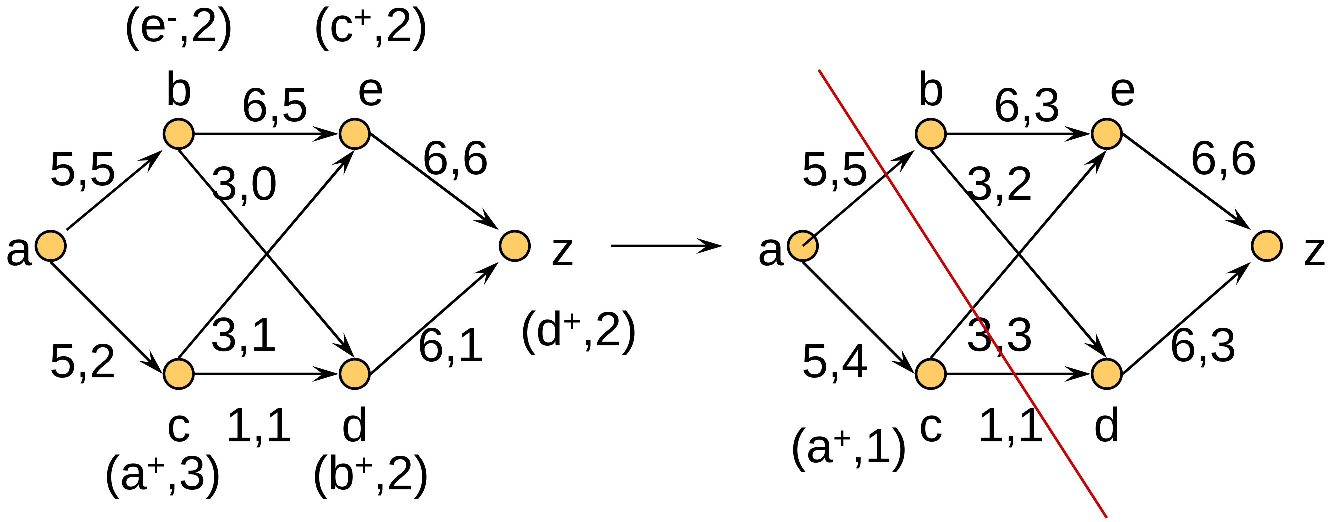
implementation of priority queue

- **linear list: O($|E|^2$)**
- **heap: O(|E|log|E|)**

Network Flow

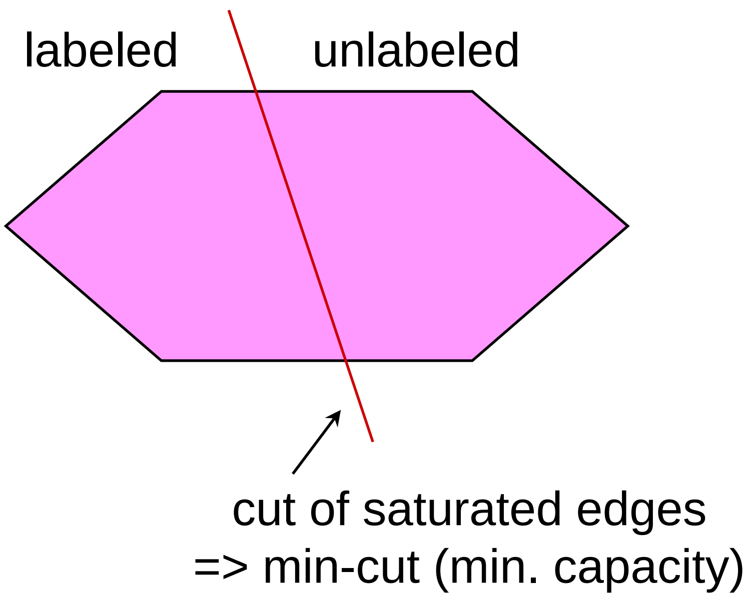



b
3
d
5
6
6
a
z
5
1
6
c
3
e
b
6
e
5
3
6
a
z
5
3
6
c
1
d
b
6,5
e
5,5
3,0
6,6
a
z
5,2
3,1
6,1
c
1,1
d
b
6,5
e
5,5
3
6,5
a
z
5,1
3
6,1
c
1,1
d
7 (?)



Max flow - min cut theorem:

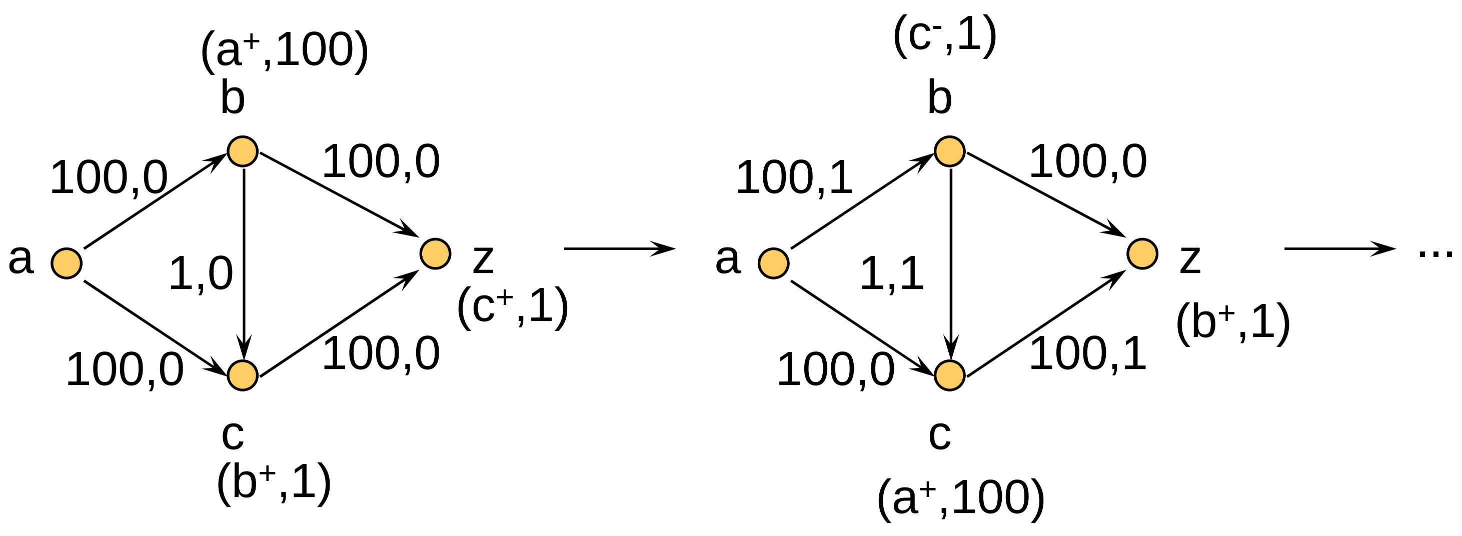
max flow = capacity of min cut



- **Augmenting Flow Algorithm**

  1. Breadth-first search
  2. If sink is visited
  a. back trace updating the flow
  b. delete labels
  c. go to 1
  otherwise
  done

  – **Why breadth-first search?**
    - **Shortest path first**

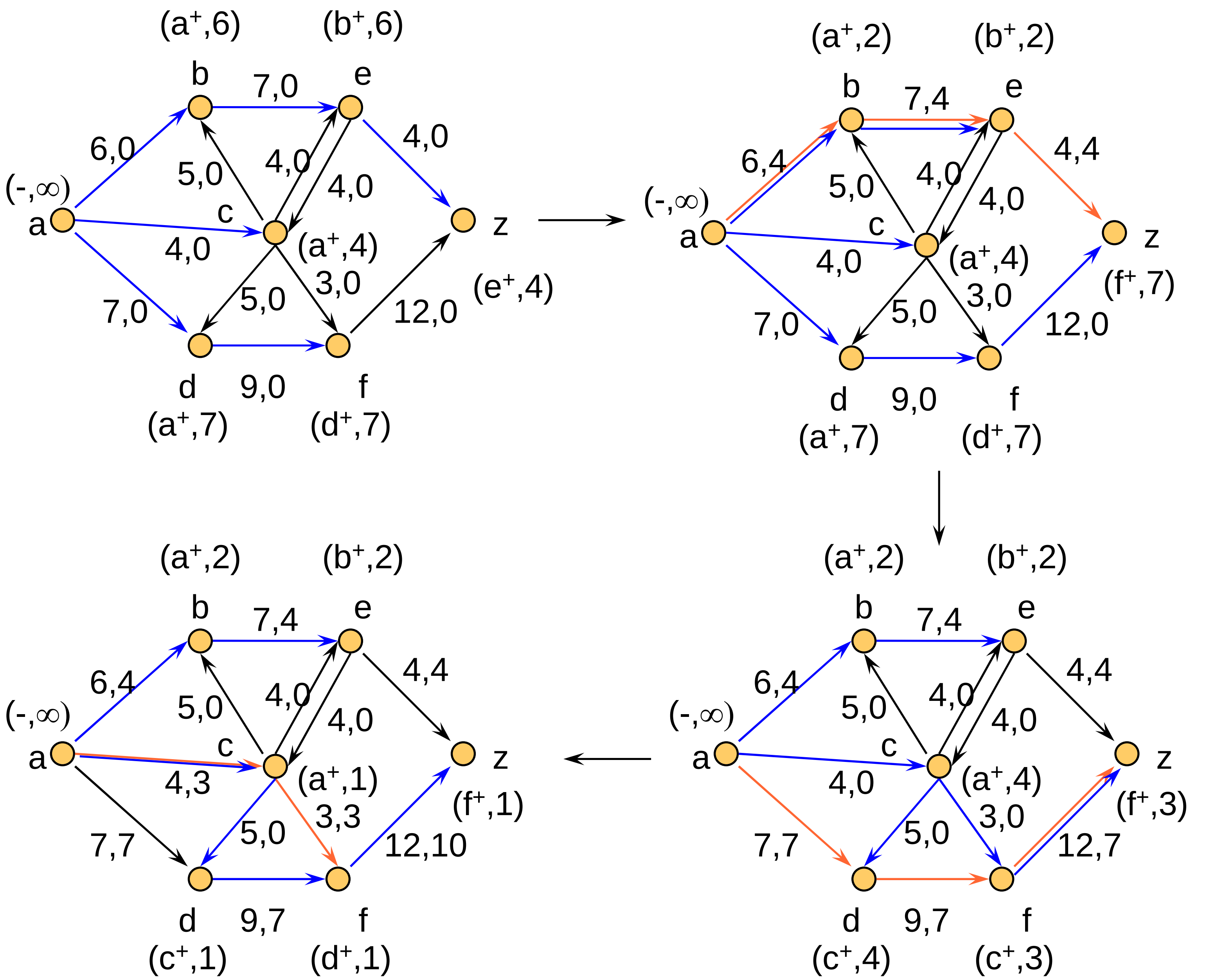


- **Complexity of Augmenting Flow Algorithm**
  - **$O(|E|)$ for each iteration (breadth first search) and**
  - **# of iterations = $O(|V||E|)$**
  - → **Complexity = $O(|V||E|^2)$**

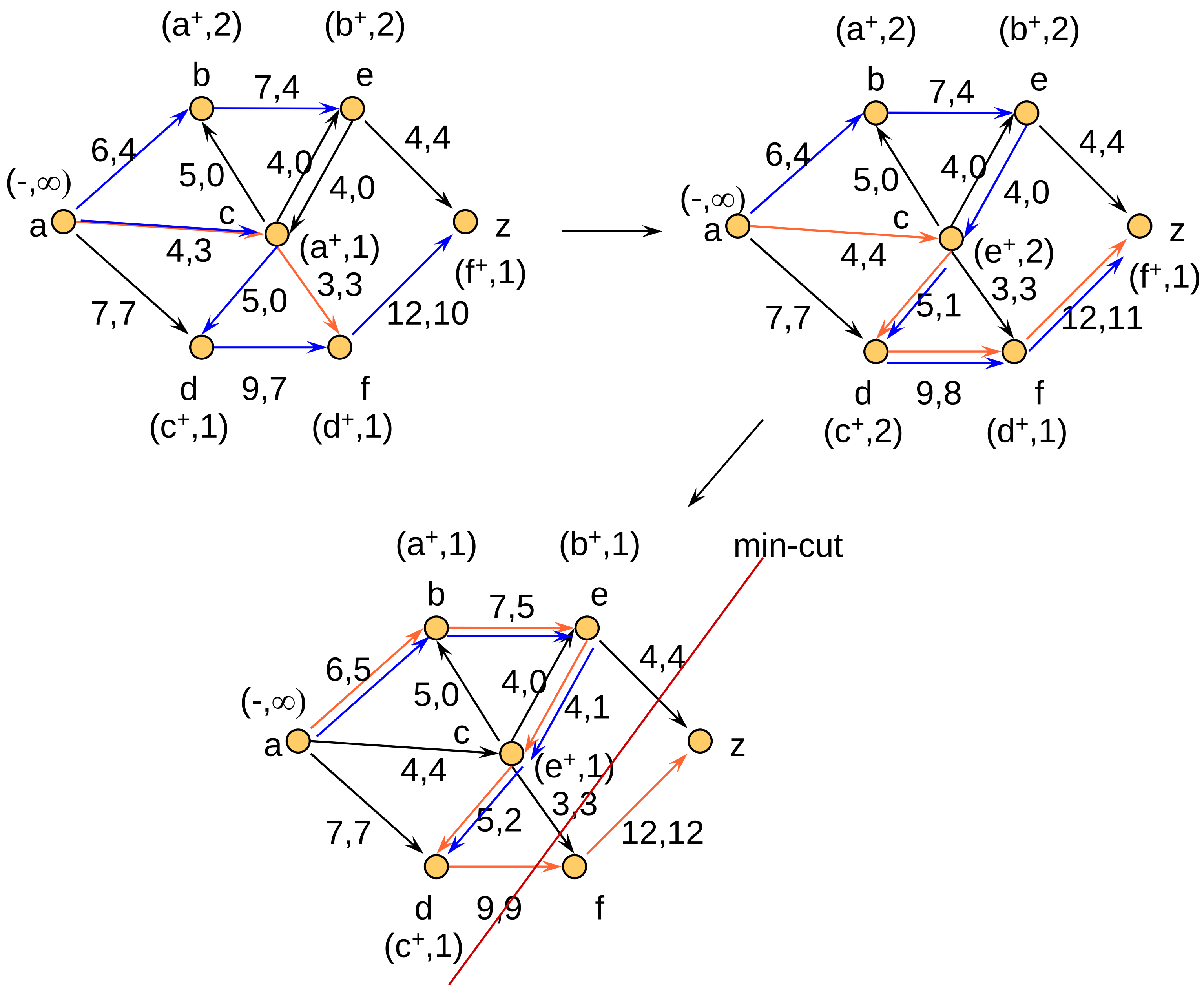
Why # of iterations = $O(|V||E|)$?

- **At least one bottleneck per shortest path (or per iteration)**
- **During the whole process, each edge can be a bottleneck at most $|V|$ times (prove this as a homework problem).**

- **Can be improved to reduce complexity to $O(|V|^3)$**

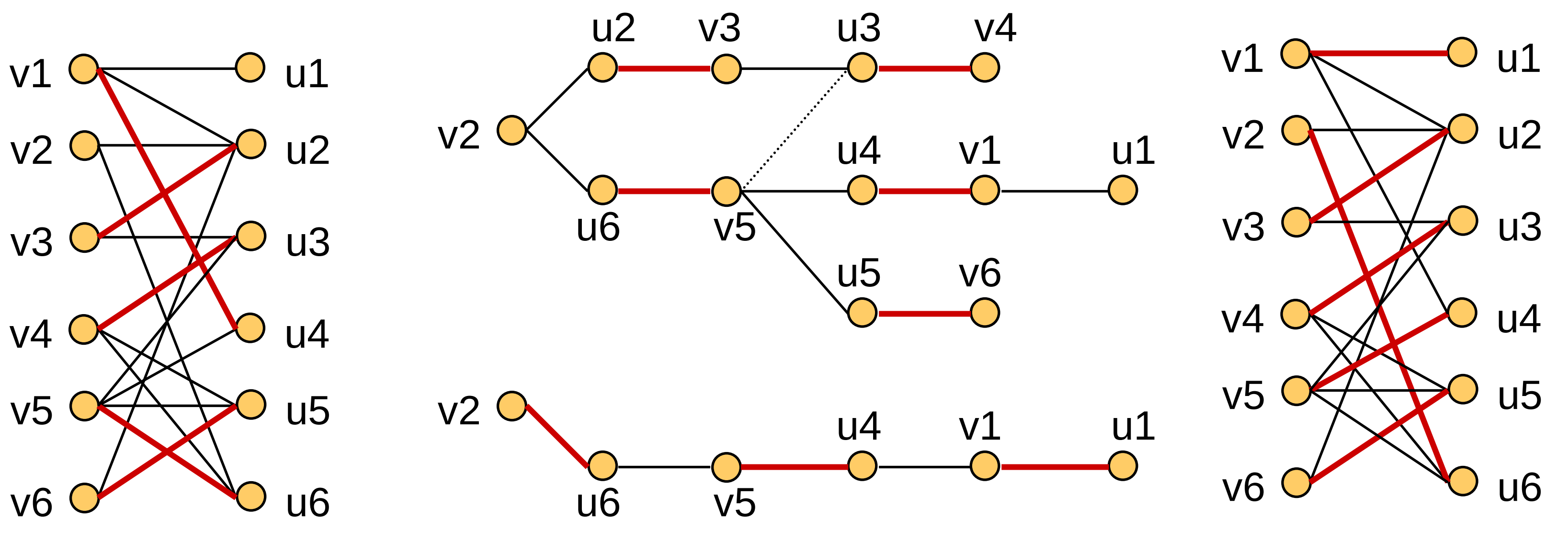


(a+,6)
(b+,6)
b
7,0
e
6,0
4,0
(-,∞)
a
5,0
4,0
c
4,0
z
(a+,4)
4,0
3,0
(e+,4)
7,0
5,0
12,0
d
9,0
f
(a+,7)
(d+,7)
(a+,2)
(b+,2)
b
7,4
e
6,4
4,4
(-,∞)
a
5,0
4,0
c
4,0
z
(a+,4)
4,0
3,0
(f+,7)
7,0
5,0
12,0
d
9,0
f
(a+,7)
(d+,7)
(a+,2)
(b+,2)
b
7,4
e
6,4
4,4
(-,∞)
a
5,0
4,0
c
4,0
z
(a+,4)
4,0
3,0
(f+,3)
7,7
5,0
12,7
d
9,7
f
(c+,4)
(c+,3)
(a+,2)
(b+,2)
b
7,4
e
6,4
4,4
(-,∞)
a
5,0
4,0
c
4,0
z
(a+,1)
4,3
3,3
(f+,1)
7,7
5,0
12,10
d
9,7
f
(c+,1)
(d+,1)


(a⁺,2)
(b⁺,2)
b
e
7,4
6,4
4,4
(-,∞)
5,0
4,0
4,0
a
c
z
4,3
(a⁺,1)
3,3
(f⁺,1)
7,7
5,0
12,10
d
9,7
f
(c⁺,1)
(d⁺,1)
(a⁺,2)
(b⁺,2)
b
e
7,4
6,4
4,4
(-,∞)
5,0
4,0
4,0
a
c
z
4,4
(e⁺,2)
3,3
(f⁺,1)
7,7
5,1
12,11
d
9,8
f
(c⁺,2)
(d⁺,1)
(a⁺,1)
(b⁺,1)
min-cut
b
7,5
e
6,5
4,4
(-,∞)
5,0
4,0
4,1
a
c
z
4,4
(e⁺,1)
3,3
7,7
5,2
12,12
d
9,9
f
(c⁺,1)

Matching

- **A matching M of a graph G(V, E) is a subset of E, where no two edges share the same node.**
- **Cardinality Matching: maximize |M|**
- **Bipartite Cardinality Matching**
  - **Find a cardinality matching of a bipartite graph B(V,U,E)**
  - **O(min(|V|,|U|)|E|)**

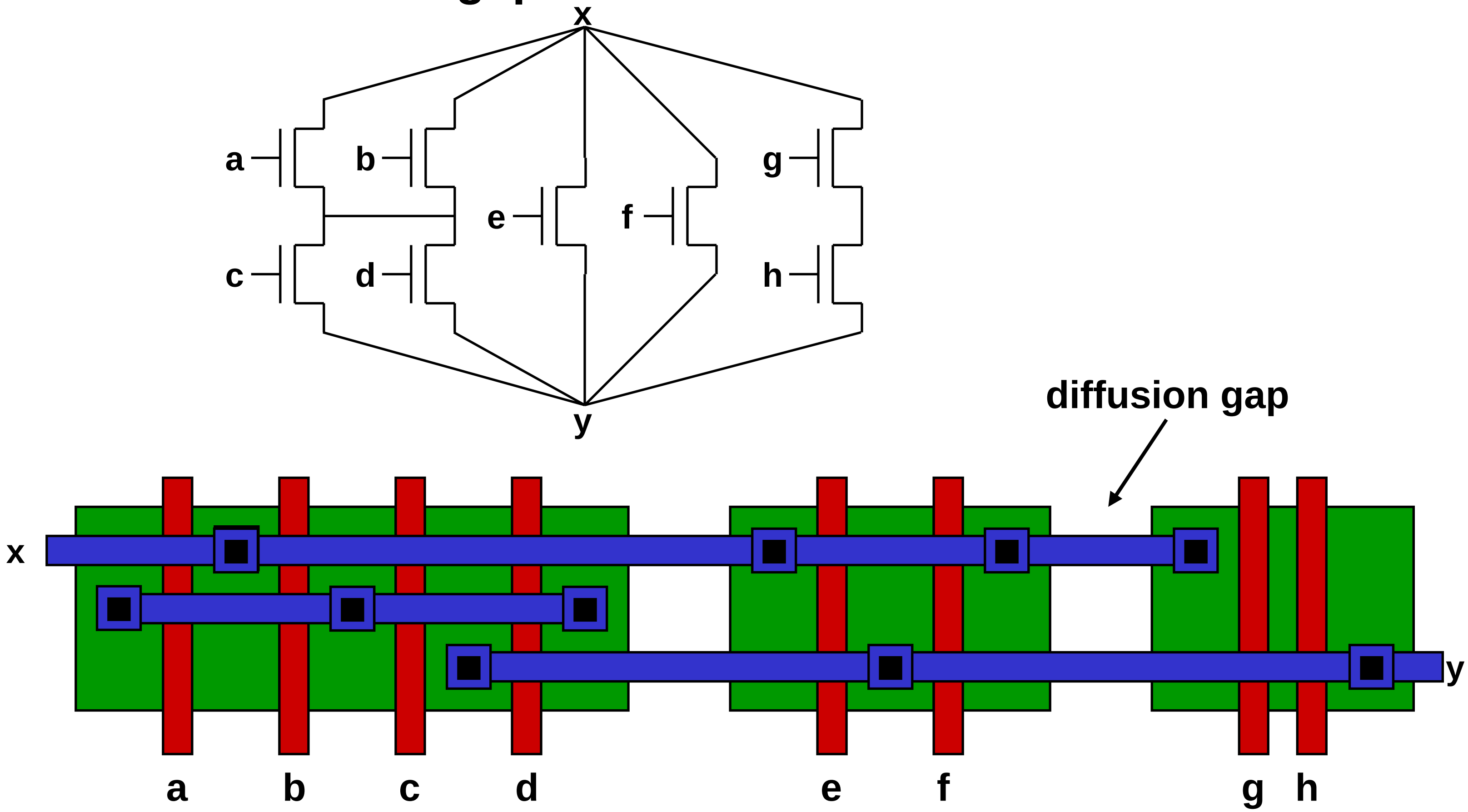


initial matching   alternating path   augmented matching

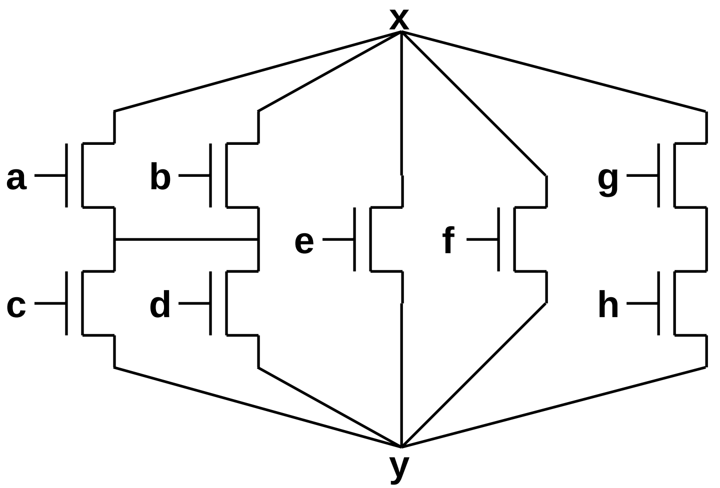
- **Nonbipartite Matching**
  - **Find a cardinality matching of a general graph G(V,E)**
  - **$O(|V|^3)$**
- **Weighted Matching: maximize total weight of M**
- **Bipartite Weighted Matching**
  - **Find a weighted matching of a bipartite graph B(V,U,E,W)**
  - **$O(|V|^3)$ for a complete bipartite graph with 2|V| vertices**
- **Nonbipartite Weighted Matching**
  - **Find a weighted matching of a general graph G(V,E,W)**
  - **$O(|V|^3)$**

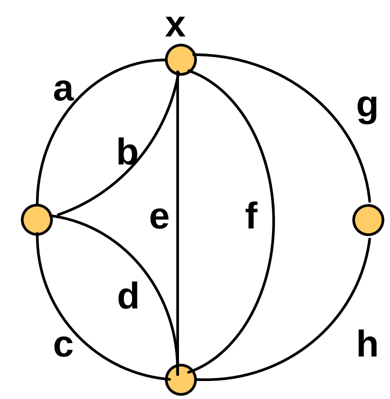
Functional Cell Design

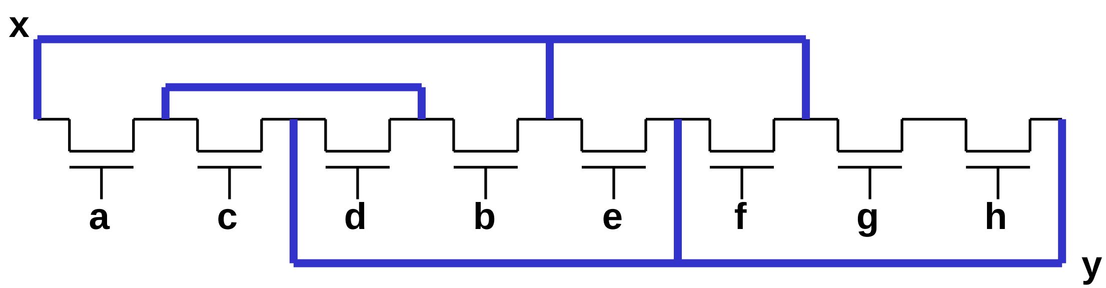
- **Layout**
  - **Layers: diffusion, metal, poly**
  - **Linear array of transistors**
  - **Avoid diffusion gaps to minimize area**

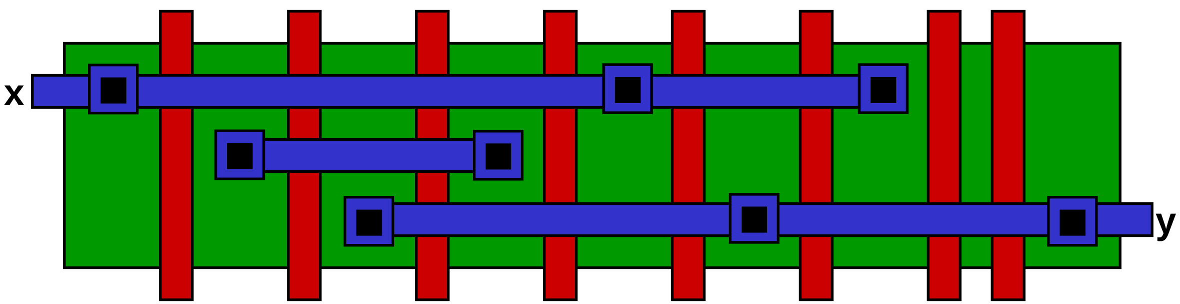


- **To minimize the number of diffusion gaps**
  - **Find Euler trail**









- **Problem**
  - **Given a graph G(V,E)**
    - **V: set of vertices (interconnects)**
    - **E: set of edges (transistors)**
  - **Find an Euler trail**
  - **If an Euler trail is found**

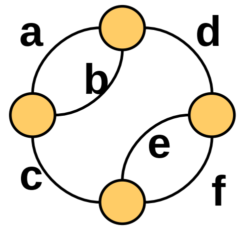
    <=> All transistors abut

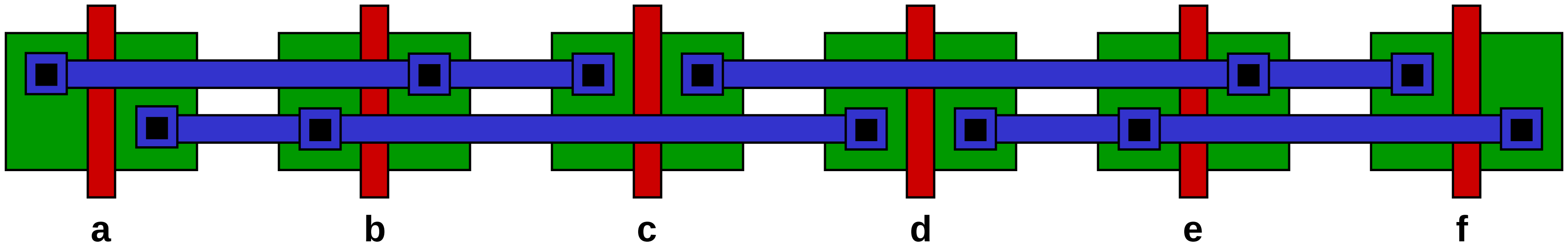
    <=> Minimum length solution
  - **Otherwise**
    - **Solution 1: Break diffusion area by gaps (find a set of trails covering the graph)**

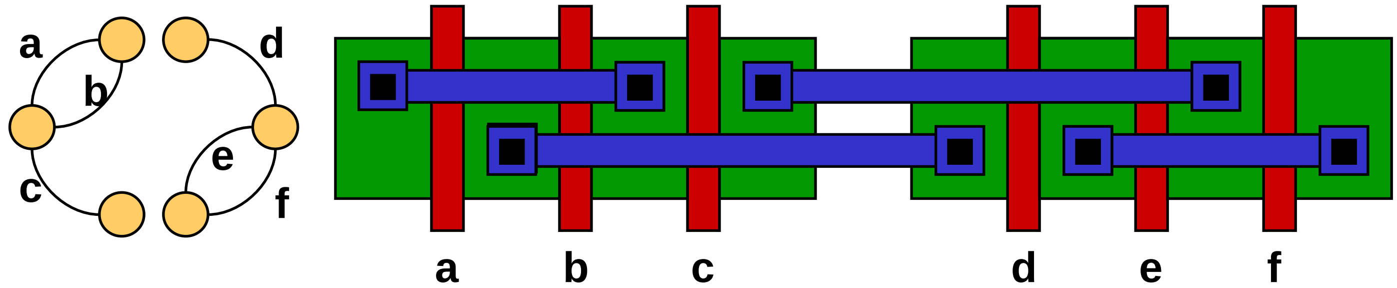
      --> Minimize number of gaps (minimize number of trails)
    - **Solution 2: Add transistors (edges) in parallel to make a trail**

      --> Minimize number of duplications

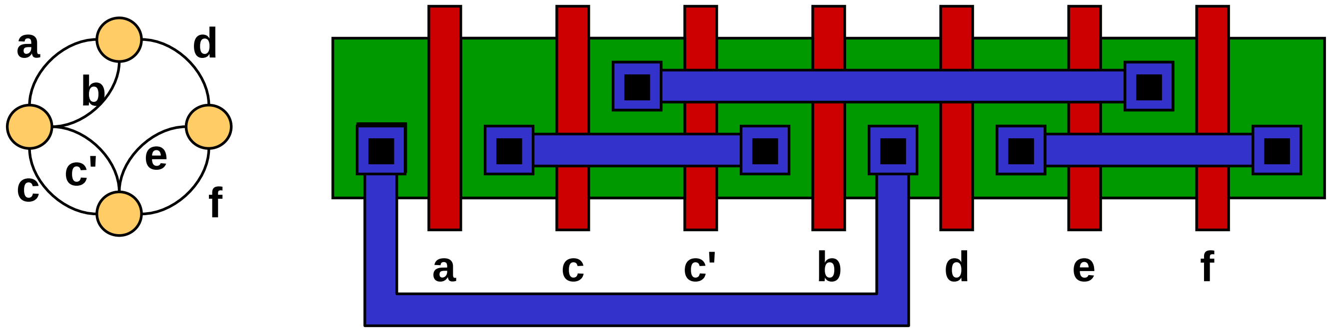
– **Example**







solution 1 diffusion gap



solution 2 transistor duplication

- **How to minimize the number of duplications?**
  - **Chinese postman's problem**
    - **Find a walk traversing each edge at least once with minimum total weight**
    - **Algorithm**
      - **step 1: Mark vertices with odd degree. If none, go to step 5**

        step 2: Compute shortest path between all pairs of marked vertices (Floyd's algorithm:O(|V|^3))

        step 3: Match marked vertices into pairs so that the sum of the lengths of the shortest paths between pairs is minimum (Maximum weighted matching:O(|V|^3))

        step 4: Duplicate edges along these paths

        step 5: construct an Euler path:O(|V|+|E|)
      - **complexity: O(|V|^3)**