Algorithms Lab

Connecting Cities

Goal

- find a largest set of vertex disjoint edges

Goal

find a **largest matching**

Goal

find a **largest matching**

BGL: $O(VE) = O(n^2)$.
(hits timelimit on the second test set)

BGL solves the matching problem in general graphs

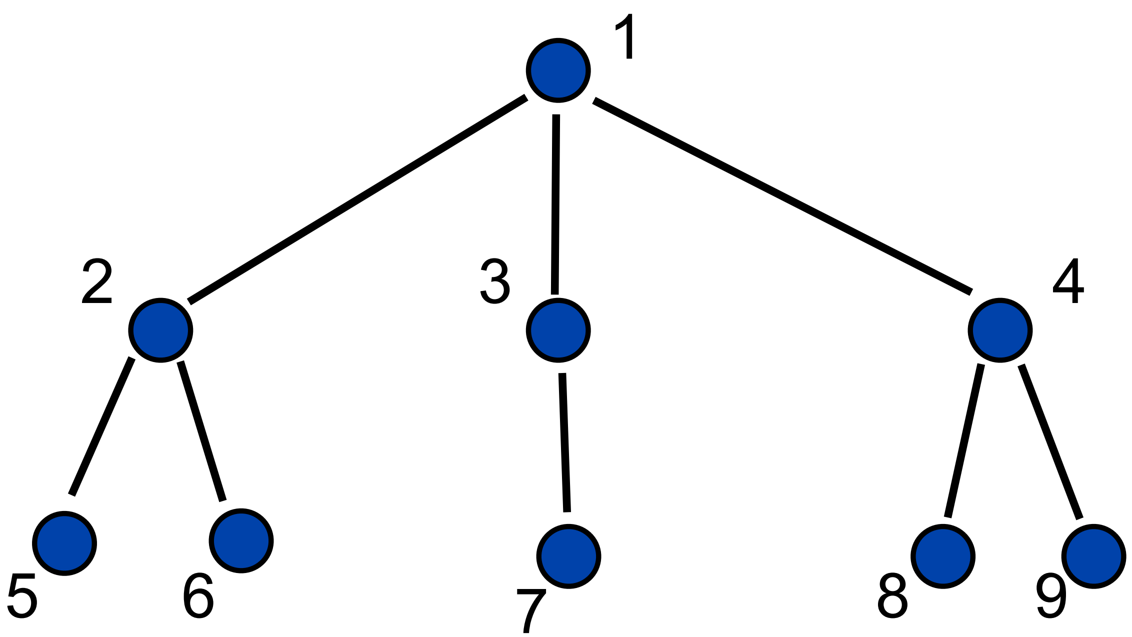
Observation:

- input graph is a **tree**

BGL solves the matching problem in general graphs

Observation:

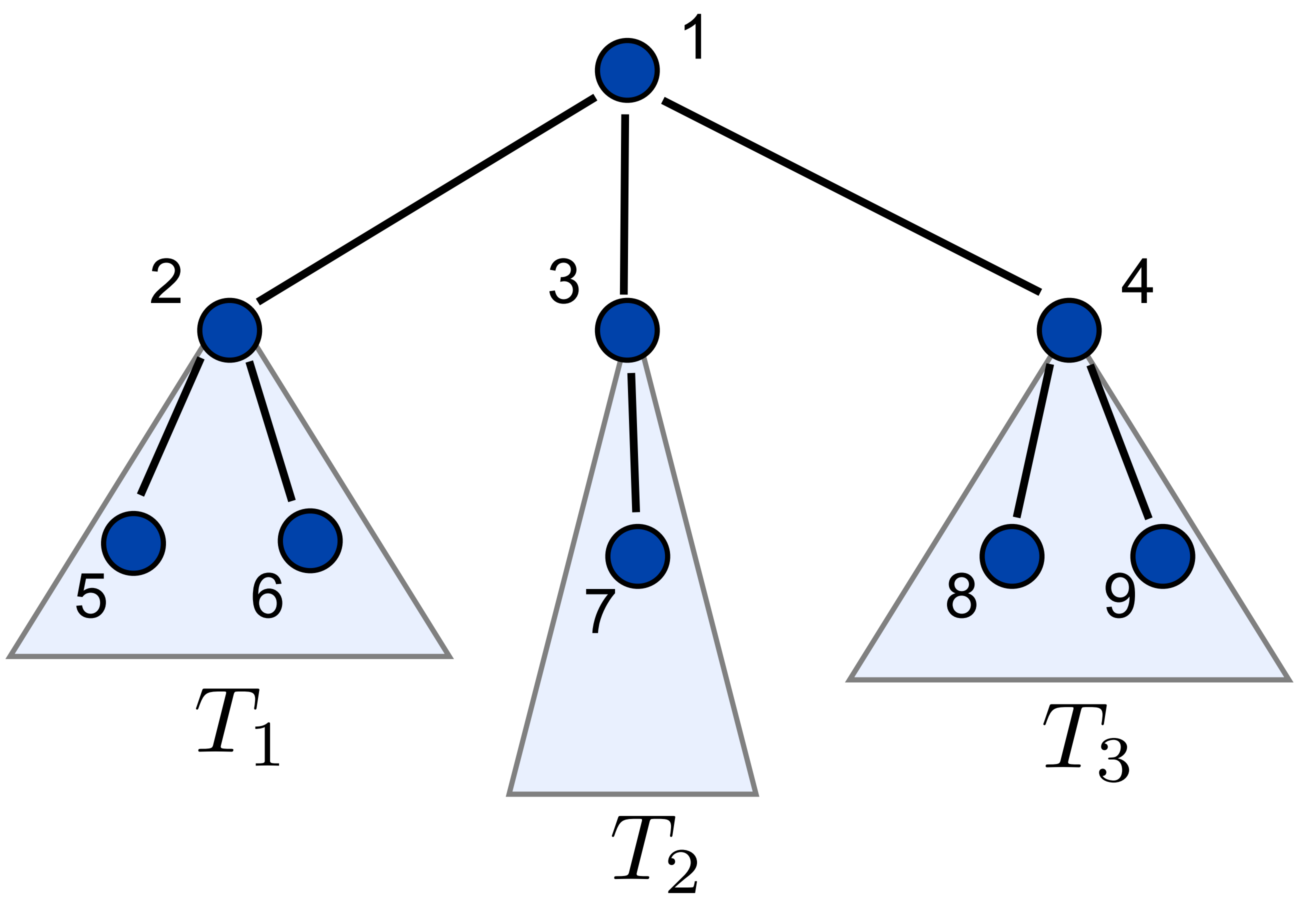
- input graph is a **tree**



BGL solves the matching problem in general graphs

Observation:

- input graph is a **tree**

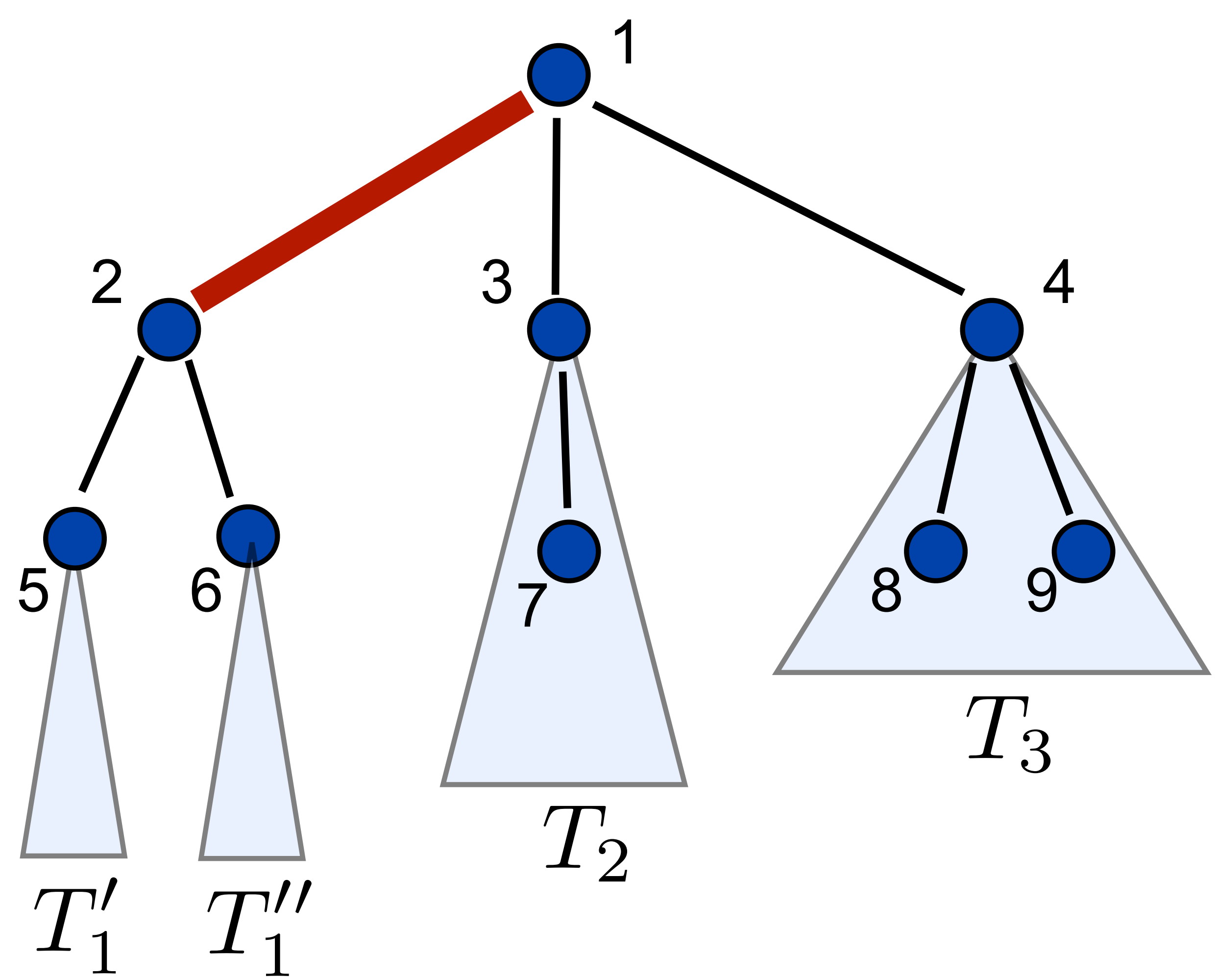


- don't take any edge incident to 1
- no edge between T_1, T_2, T_3
- max-matching in each subtree can be found **independently**!

BGL solves the matching problem in general graphs

Observation:

- input graph is a **tree**



- take edge 1-2
- no edge between

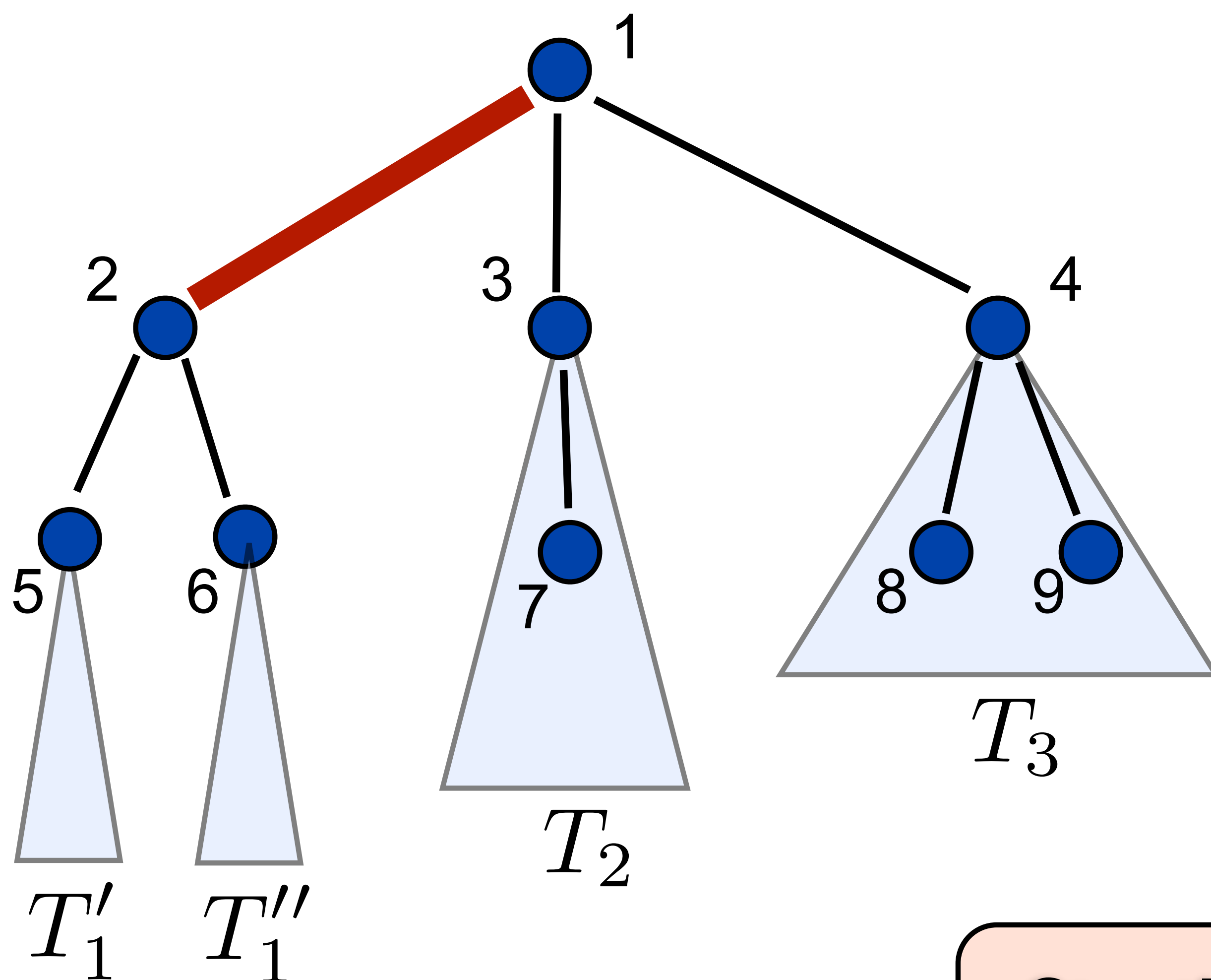
$$T_1', T_1'', T_2, T_3$$

- max-matching in each subtree can be found **independently**!

BGL solves the matching problem in general graphs

Observation:

- input graph is a **tree**



- take edge 1-2
- no edge between

$$T_1', T_1'', T_2, T_3$$

- max-matching in each subtree can be found **independently**!

Similarly for edges 1-3 and 1-4

Generalize previous example as a Dynamic Programming

- Let c(v) be the set of descendants of v in the tree
- Let M(v) be the size of the largest matching in the subtree rooted at v

$$M(v) = \max \begin{cases} \displaystyle\sum_{w \in c(v)} M(w) \\ \displaystyle\max_{a \in c(v)} \; 1 + \sum_{w \in c(v) \setminus a} M(w) + \sum_{w' \in c(a)} M(w') \end{cases}$$

Implementation details

$$M(v) = \max \begin{cases} \displaystyle\sum_{w \in c(v)} M(w) \\ \displaystyle\max_{a \in c(v)} \; 1 + \sum_{w \in c(v) \setminus a} M(w) + \sum_{w' \in c(a)} M(w') \end{cases}$$

```
matching(v) {
  S = 0;
  for all w in c(v) : S = S + M(w);
  M(v) = S;

  for all a in c(v)
     S = 1;
     for all w in c(v) \ a
       S = S + M(w);
     for all w′ in c(a)
       S = S + M(w′);
     M(v) = max(M(v), S);
}
```

Implementation details

$$M(v) = \max \begin{cases} \sum_{w \in c(v)} M(w) \\ \max_{a \in c(v)} 1 + \sum_{w \in c(v) \setminus a} M(w) + \sum_{w' \in c(a)} M(w') \end{cases}$$

```
matching(v) {
  S = 0;
  for all w in c(v) : S = S + M(w);
  M(v) = S;

  for all a in c(v)
     S = 1;
     for all w in c(v) \ a
       S = S + M(w);
     for all w′ in c(a)
       S = S + M(w′);
     M(v) = max(M(v), S);
}
```

If v has n-1 descendants - $\mathcal{O}(n^2)$

Implementation details

$$M(v) = \max \begin{cases} \displaystyle\sum_{w \in c(v)} M(w) \\ \displaystyle\max_{a \in c(v)} \; 1 + \sum_{w \in c(v) \setminus a} M(w) + \sum_{w' \in c(a)} M(w') \end{cases}$$

```
matching(v) {
  S = 0;
  for all w in c(v) : S = S + M(w);
  M(v) = S; M′(v) = S;

  for all a in c(v)
     S = 1 + M′(v) - M(a);
     for all w′ in c(a)
       S = S + M(w′);
     M(v) = max(M(v), S);
}
```

Implementation details

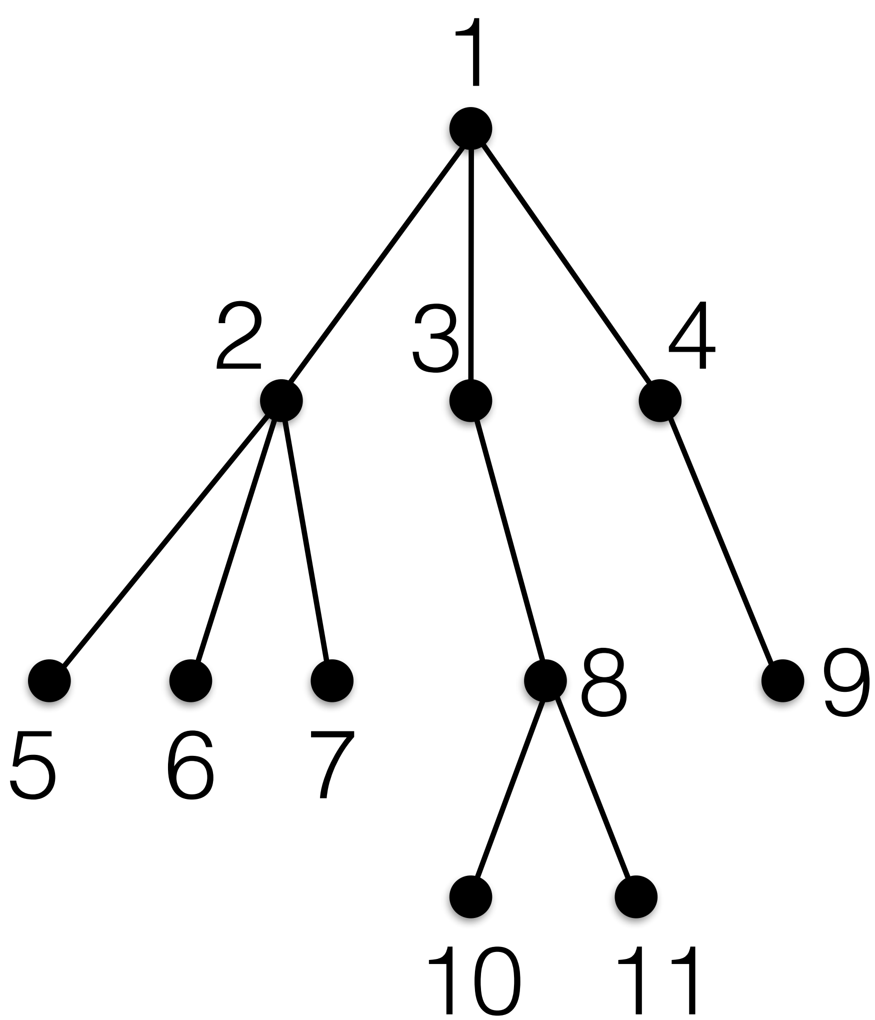
$$M(v) = \max \begin{cases} \displaystyle\sum_{w \in c(v)} M(w) \\ \displaystyle\max_{a \in c(v)} \; 1 + \sum_{w \in c(v) \setminus a} M(w) + \sum_{w' \in c(a)} M(w') \end{cases}$$

```
matching(v) {
  S = 0;
  for all w in c(v) : S = S + M(w);
  M(v) = S; M′(v) = S;

  for all a in c(v)
     S = 1 + M′(v) - M(a);
     for all w′ in c(a)
       S = S + M(w′);
     M(v) = max(M(v), S);
}
```

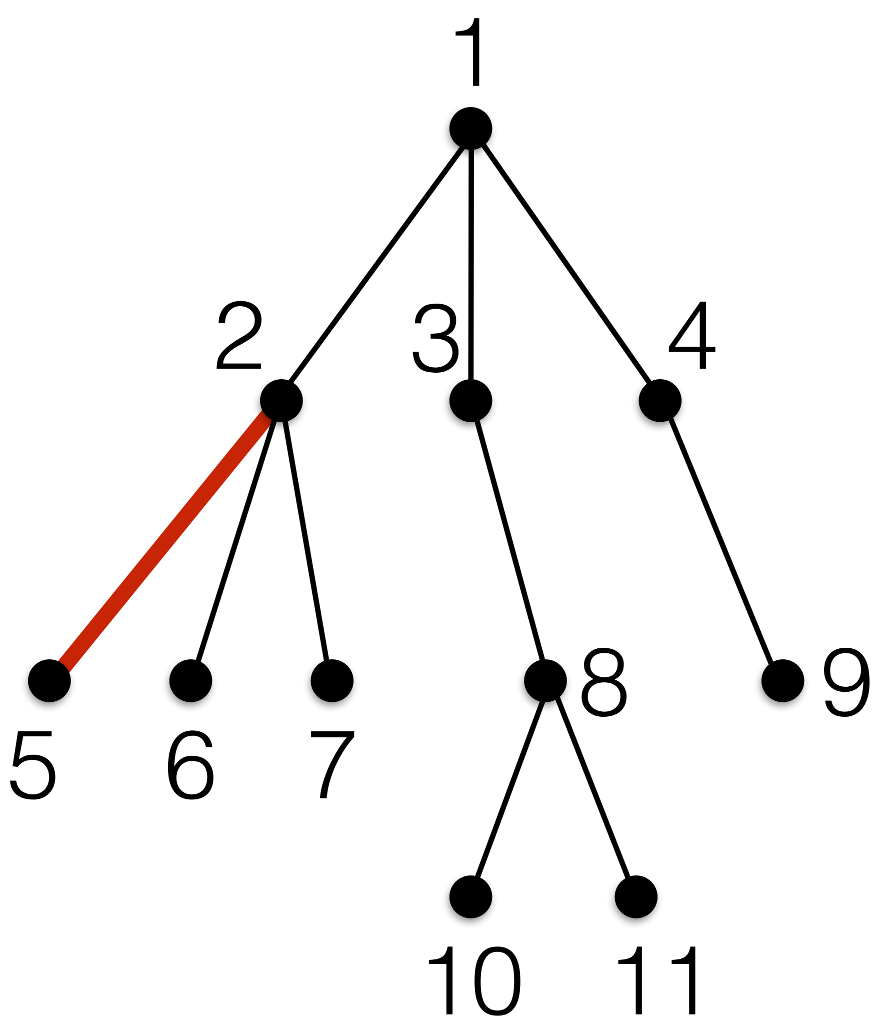
Exercise: show that the running time is linear!

Greedy solution



Repeat: take an edge which contains a leaf and remove its endpoints

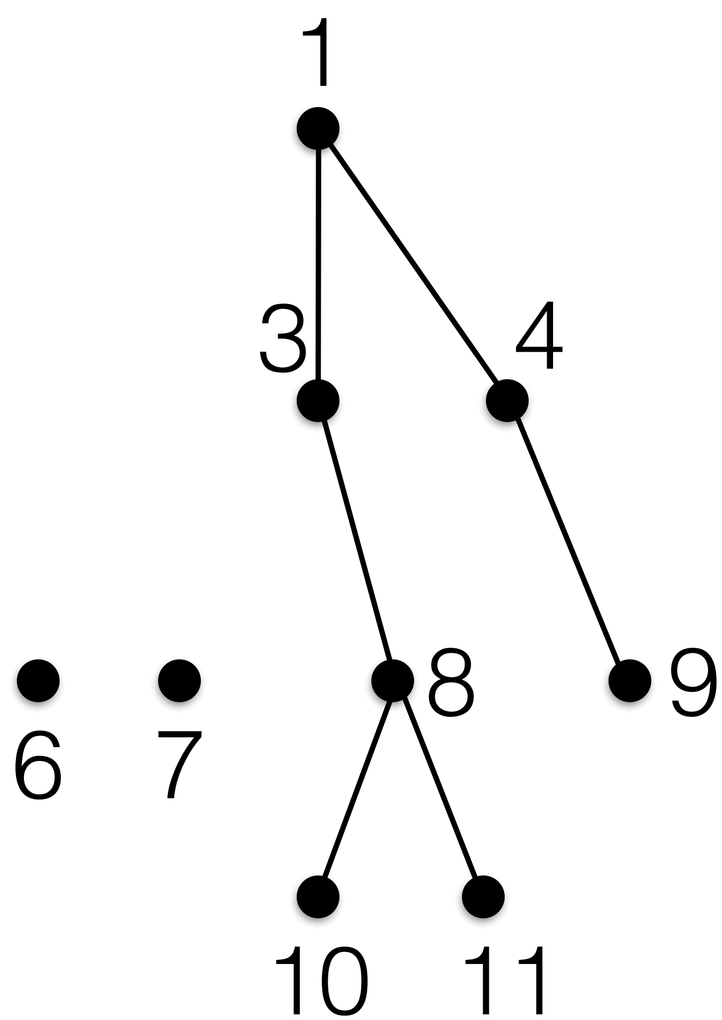
Greedy solution



Repeat: take an edge which contains a leaf and remove its endpoints

{2,5}

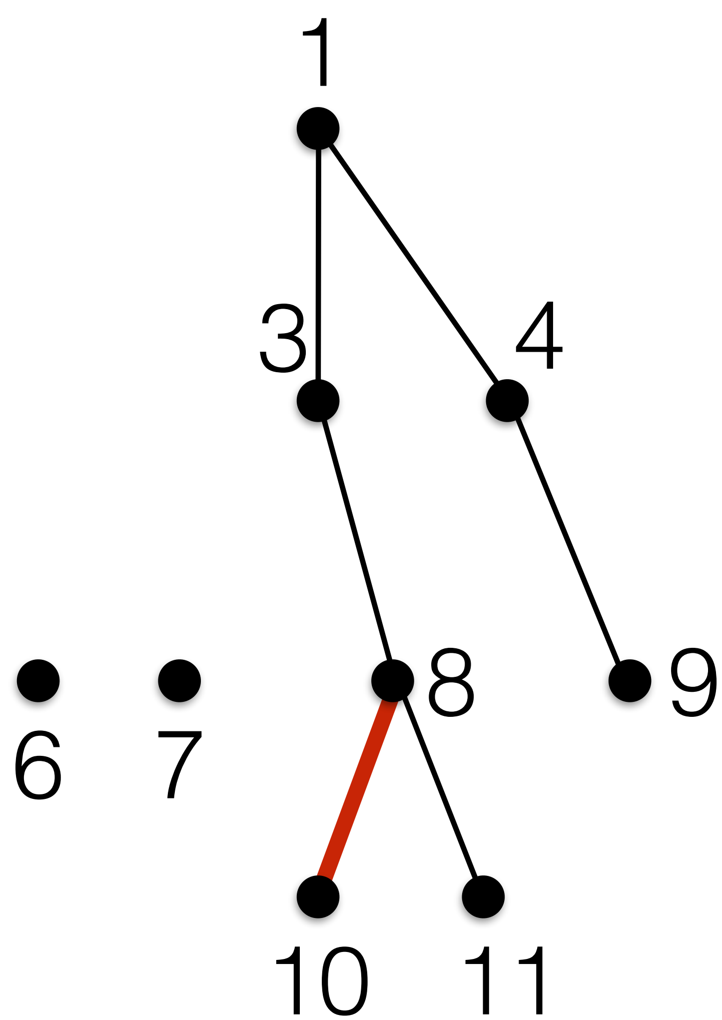
Greedy solution



{2,5}

Repeat: take an edge which contains a leaf and remove its endpoints

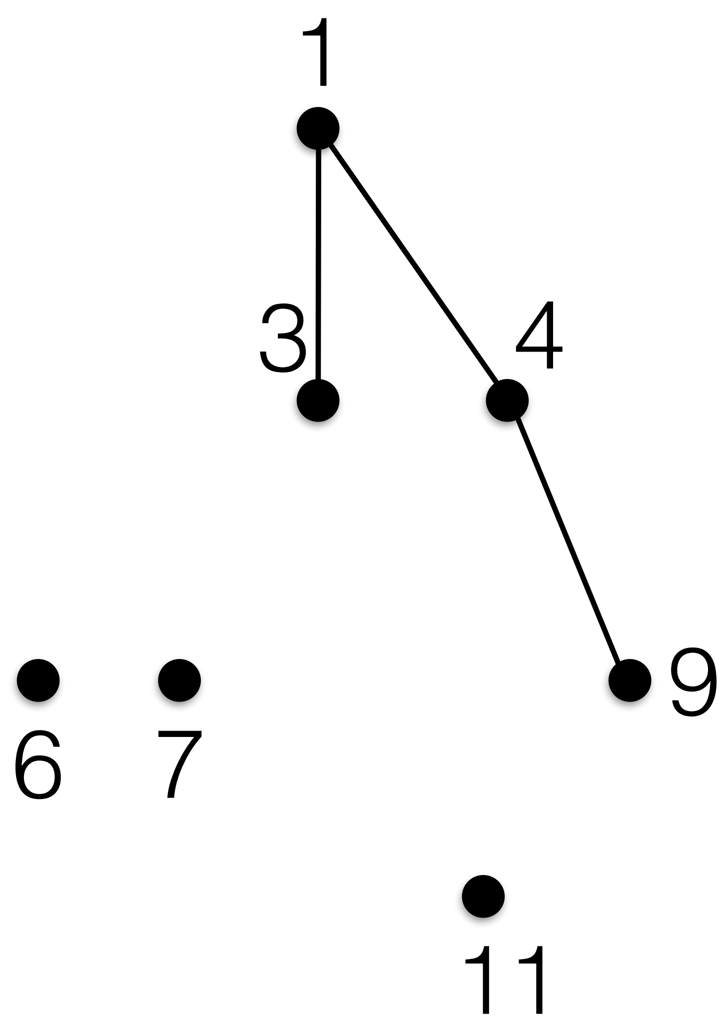
Greedy solution



Repeat: take an edge which contains a leaf and remove its endpoints

{2,5},{10,8}

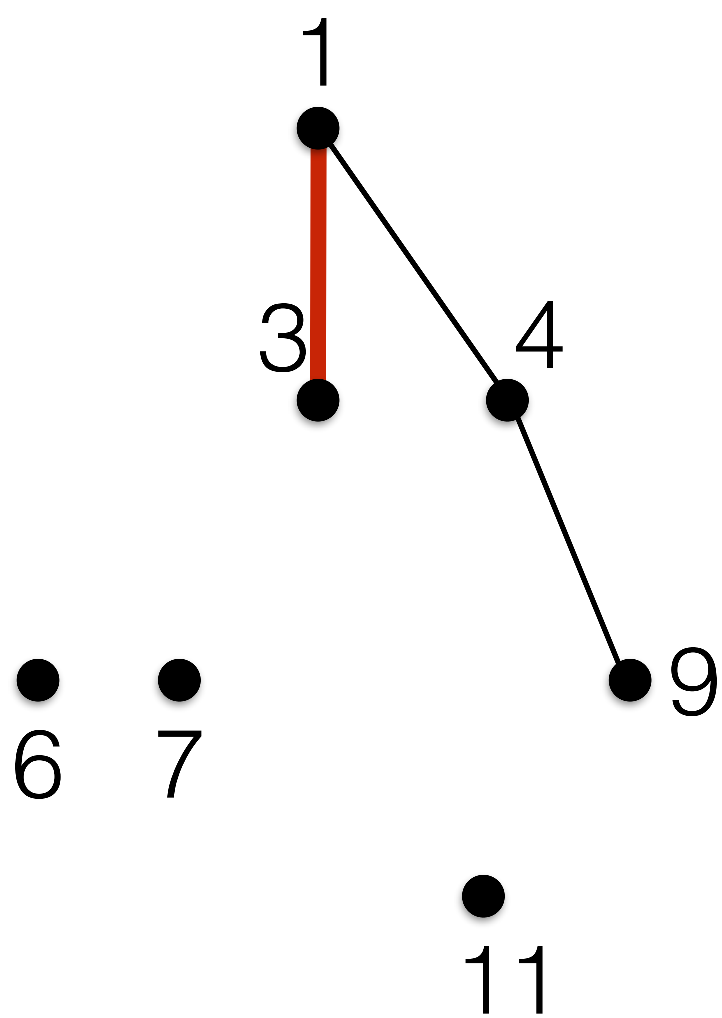
Greedy solution



Repeat: take an edge which contains a leaf and remove its endpoints

{2,5},{10,8}

Greedy solution

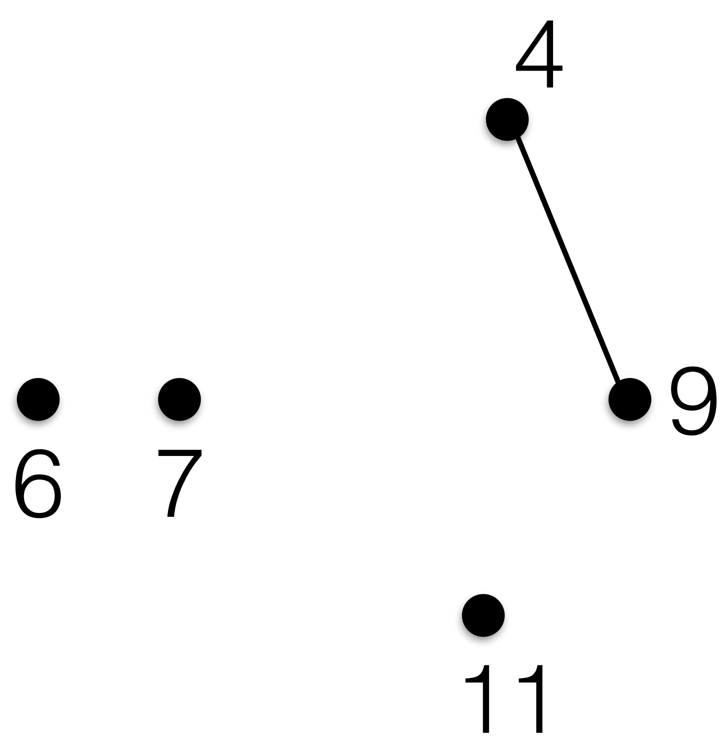


Repeat: take an edge which contains a leaf and remove its endpoints

{2,5},{10,8},{1,3}

Greedy solution

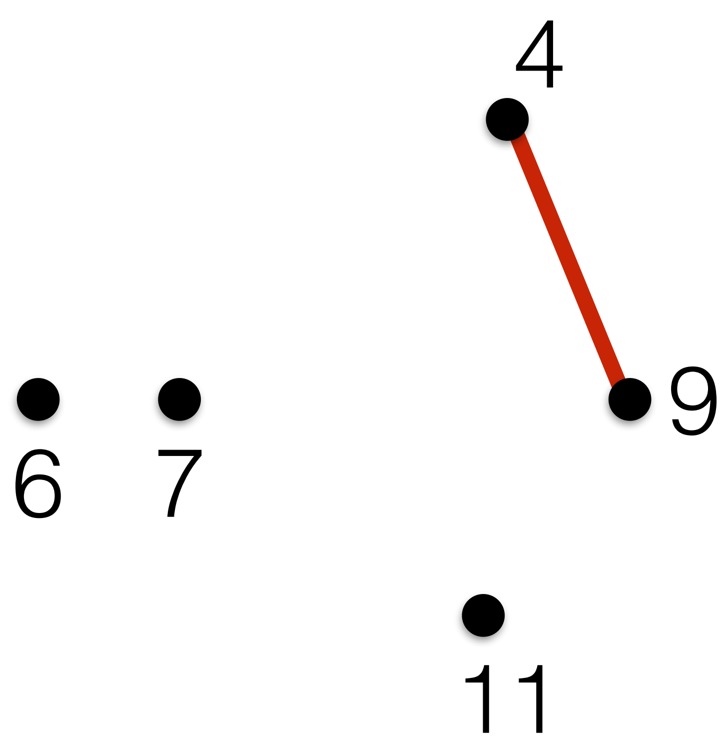
Repeat: take an edge which contains a leaf and remove its endpoints



{2,5},{10,8},{1,3}

Greedy solution

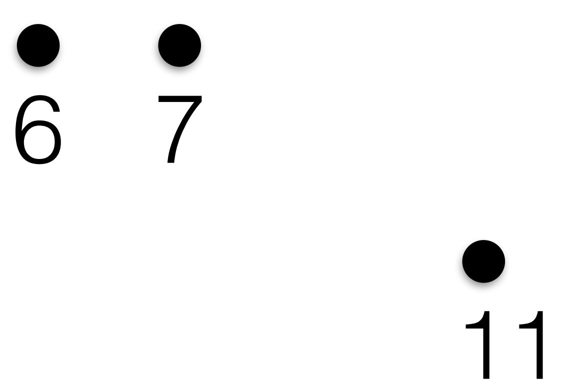
Repeat: take an edge which contains a leaf and remove its endpoints



{2,5},{10,8},{1,3},{4,9}

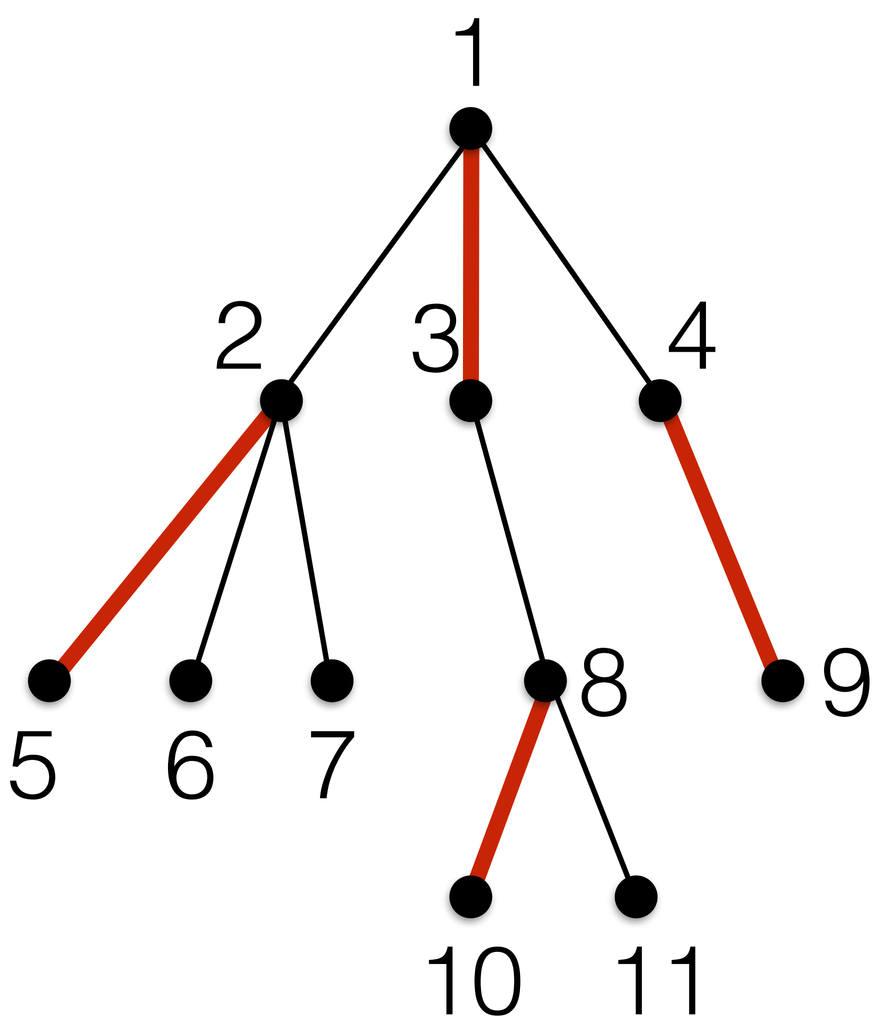
Greedy solution

Repeat: take an edge which contains a leaf and remove its endpoints



{2,5},{10,8},{1,3},{4,9}

Greedy solution



Repeat: take an edge which contains a leaf and remove its endpoints

Correctness can be proven using the exchange argument (see slides from Week 2)

{2,5},{10,8},{1,3},{4,9}

Consecutive Constructions

The problem

Problem

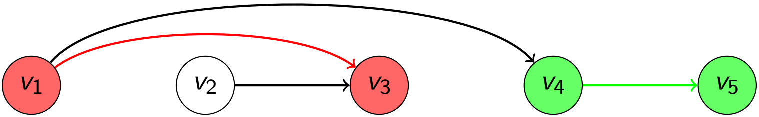
Given a DAG G. Find the maximum number of edges that can be packed in vertex-disjoint paths in G.

Example

$$\begin{tikzcd}[column sep=large] v_1 \arrow[rr, bend left] \arrow[rrr, bend left] & v_2 \arrow[r] & v_3 & v_4 \arrow[r] & v_5 \end{tikzcd}$$

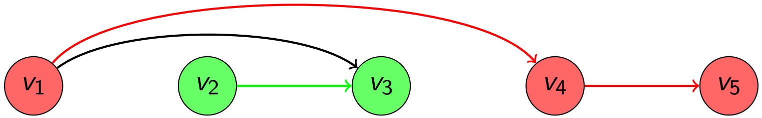
Example – greedy solution

Greedy solution with two edges.



Example – optimal solution

Optimal solution with three edges.



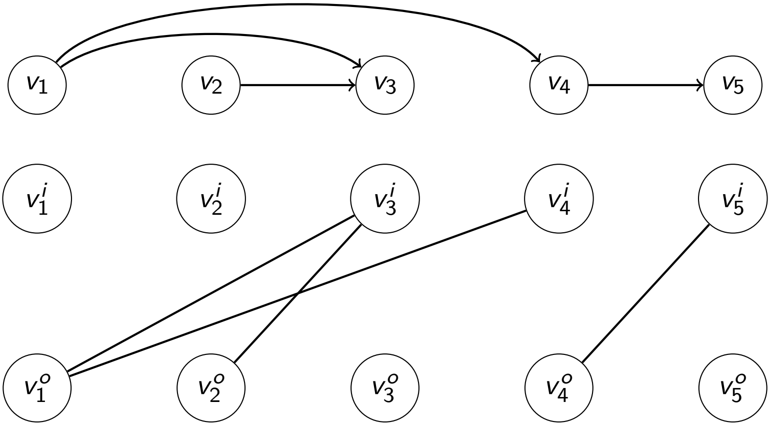
Possible approaches

- Greedy – we've just seen it doesn't work.
- DP – the graph is a DAG, so there is some hope. However, we hit the wall pretty soon (assuming you take the edge v_1 to v_k, you still need a solution on $v_2, \ldots, v_{k-1}, v_{k+1}, \ldots, v_n$ where parts "before" and "after" v_k are not independent).

Matching solution

Let $V_{out} := \{u_{out} : u \in V(G)\}$. Similarly, $V_{in} := \{u_{in} : u \in V(G)\}$.
Finally, let $E' := \{(u_{out}, v_{in}) : (u, v) \in E(G)\}$.
Consider bipartite $G' := (V_{out} \cup V_{in}, E')$.

Matching solution – example



Matching solution

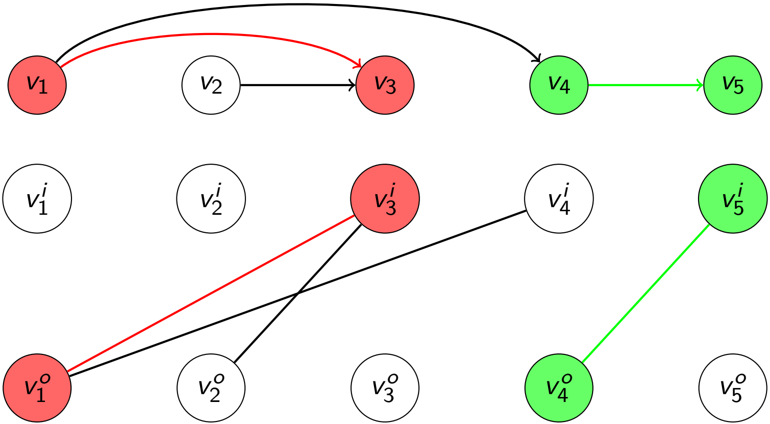
Let $V_{out} := \{u_{out} : u \in V(G)\}$. Similarly, $V_{in} := \{u_{in} : u \in V(G)\}$.
Finally, let $E' := \{(u_{out}, v_{in}) : (u, v) \in E(G)\}$.
Consider bipartite $G' := (V_{out} \cup V_{in}, E')$.

Lemma

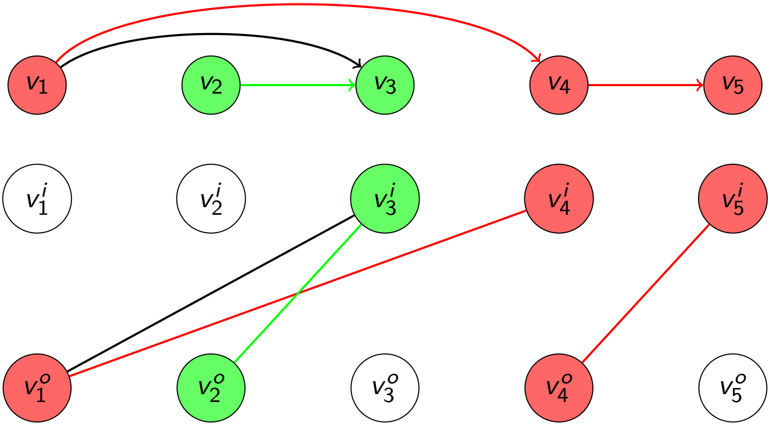
There is a 1-1 to correspondence between solutions for G with k edges and matchings in G' of size k.

Therefore, the problem reduces to finding a maximum matching.

Matching solution – example



Matching solution – example



2nd solution - MinCost MaxFlow

- For each vertex - capacity 1 and cost 0
- For each edge - capacity 1 and cost -1
- run MinCost MaxFlow
- Slower algorithm, works only on smaller input sizes

Missing Roads

Problem

- Given: undirected graph G and an integer k
- Find: set of k cities $\{v_1, v_2, \dots, v_k\}$ and edges $\{e_1, e_2, \dots, e_k\}$ such that
    a) each city $\boldsymbol{v_i}$ is **adjacent** to the edge $\boldsymbol{e_i}$
    b) the sum

$$\sum_{i=1}^{k} \boldsymbol{cost(e_i)}$$

is the smallest possible

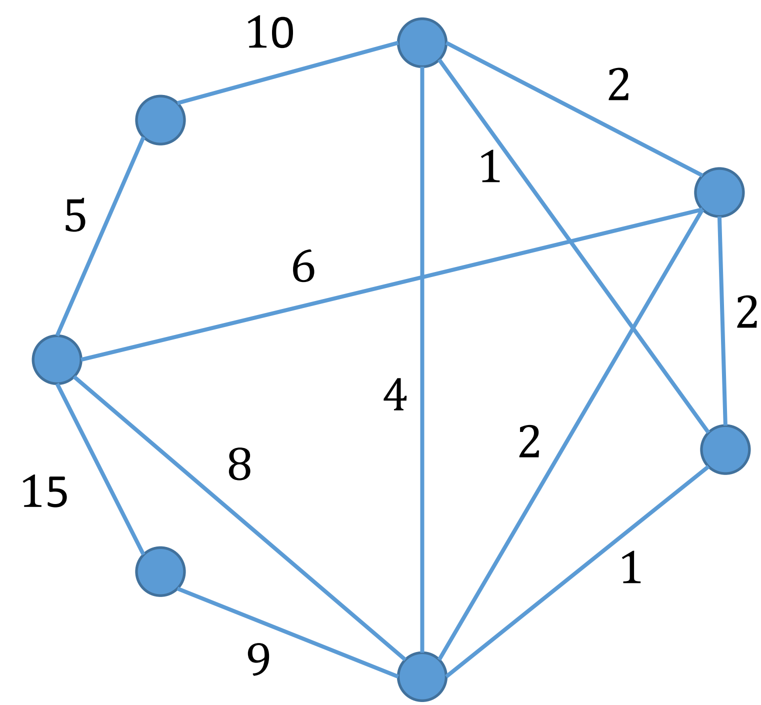
Problem

- Given: undirected graph G and an integer k
- Find: set of k cities $\{v_1, v_2, \dots, v_k\}$ and edges $\{e_1, e_2, \dots, e_k\}$ such that
  a) each city $\boldsymbol{v_i}$ is **adjacent** to the edge $\boldsymbol{e_i}$
  b) the sum

$$\sum_{i=1}^{k} \boldsymbol{cost(e_i)}$$

is the smallest possible

$k = 5$



Problem

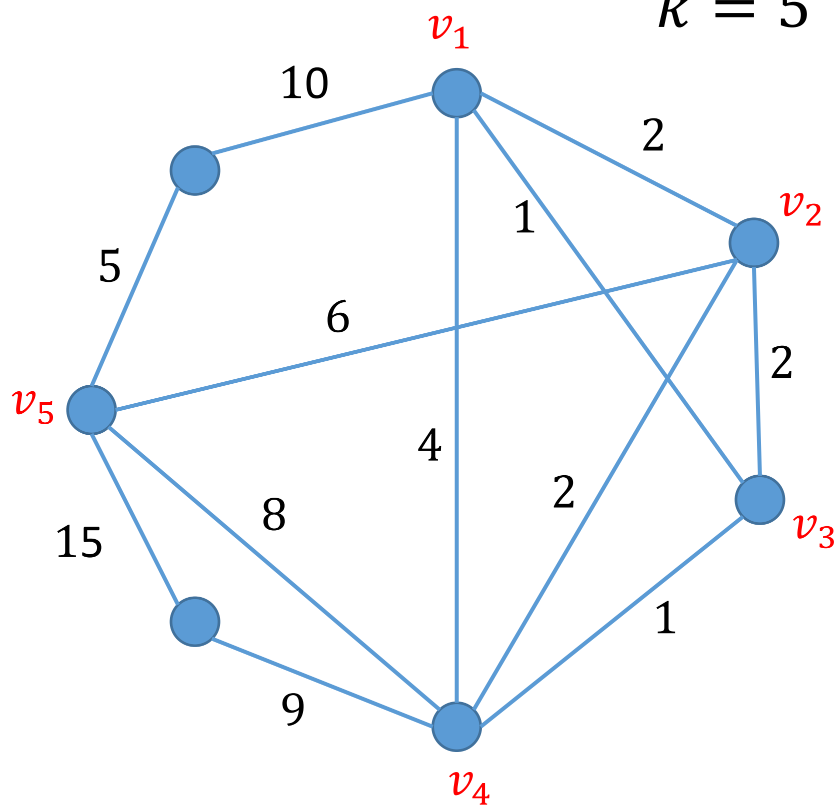
- Given: undirected graph G and an integer k

- Find: set of k cities $\{v_1, v_2, \dots, v_k\}$ and edges $\{e_1, e_2, \dots, e_k\}$ such that
    a) each city $\boldsymbol{v_i}$ is **adjacent** to the edge $\boldsymbol{e_i}$
    b) the sum

$$\sum_{i=1}^{k} \boldsymbol{cost(e_i)}$$

is the smallest possible

$k = 5$



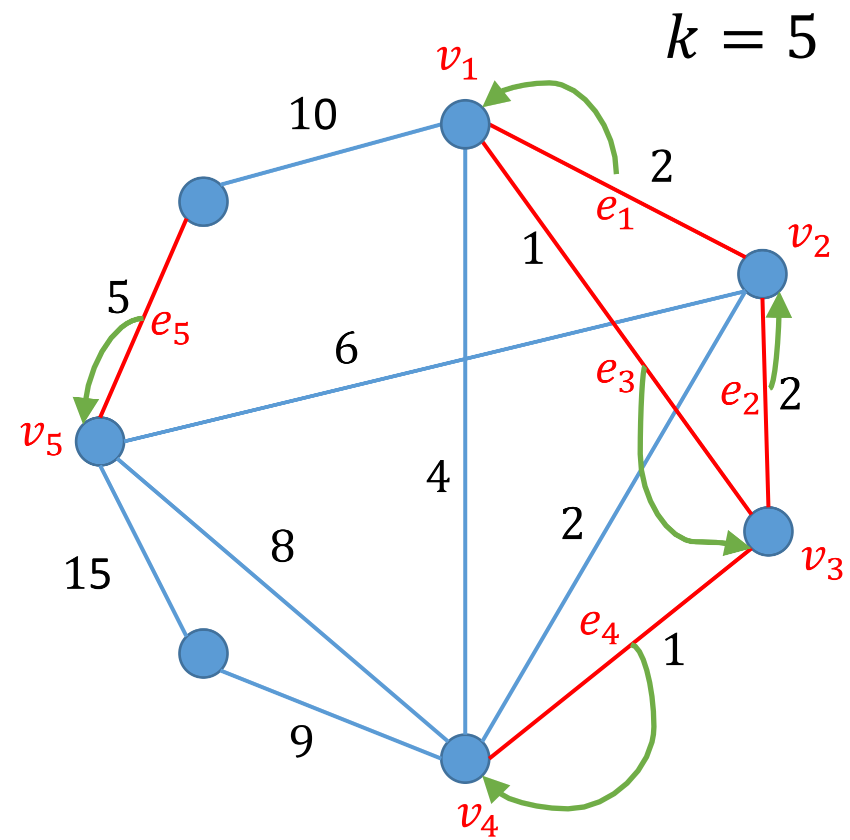
Problem

- Given: undirected graph G and an integer k

- Find: set of k cities $\{v_1, v_2, \ldots, v_k\}$ and edges $\{e_1, e_2, \ldots, e_k\}$ such that
  a) each city v_i is **adjacent** to the edge e_i
  b) the sum

$$\sum_{i=1}^{k} cost(e_i)$$

is the smallest possible

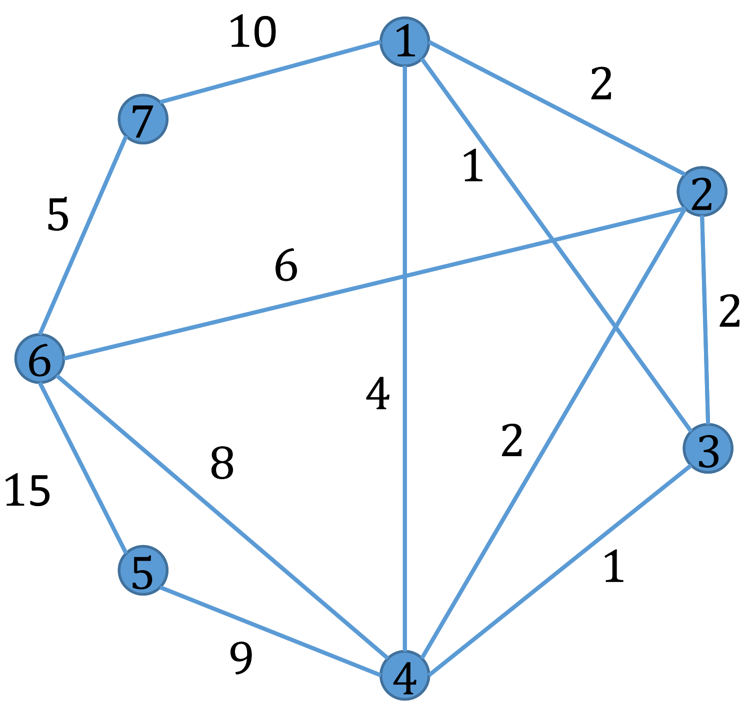


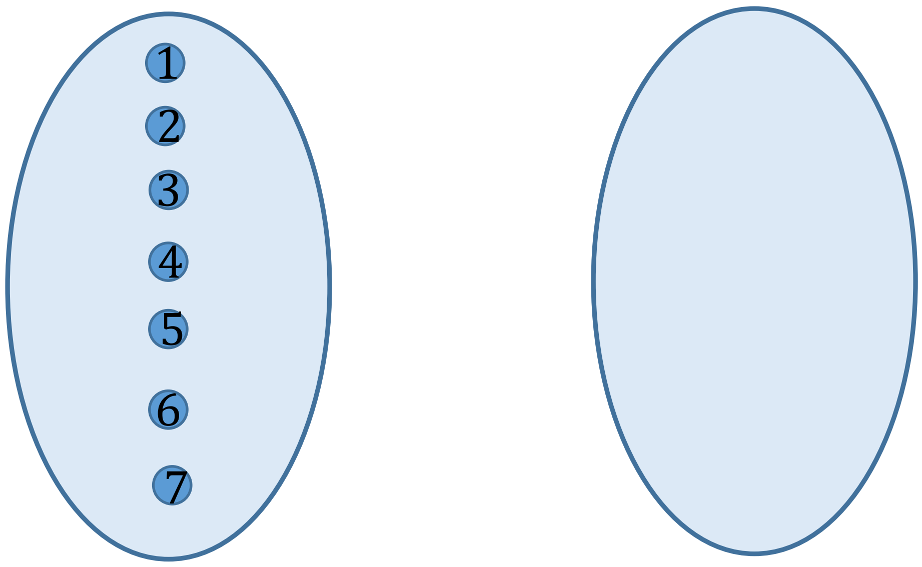
Solution

- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

Solution

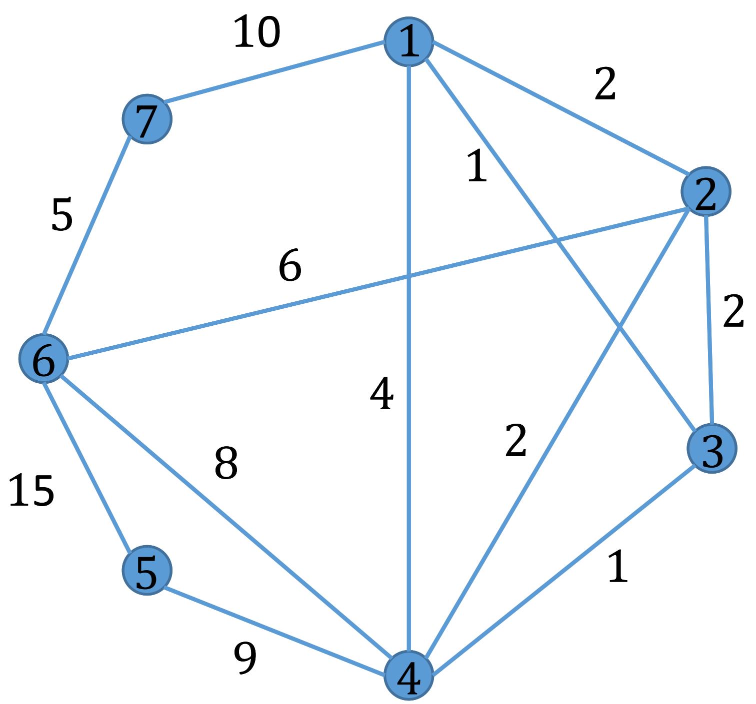
- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

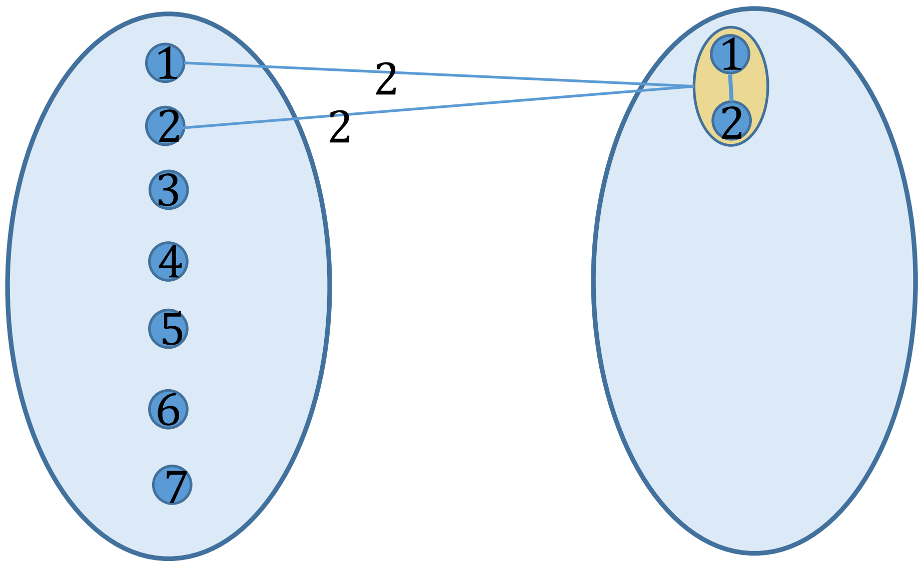




Solution

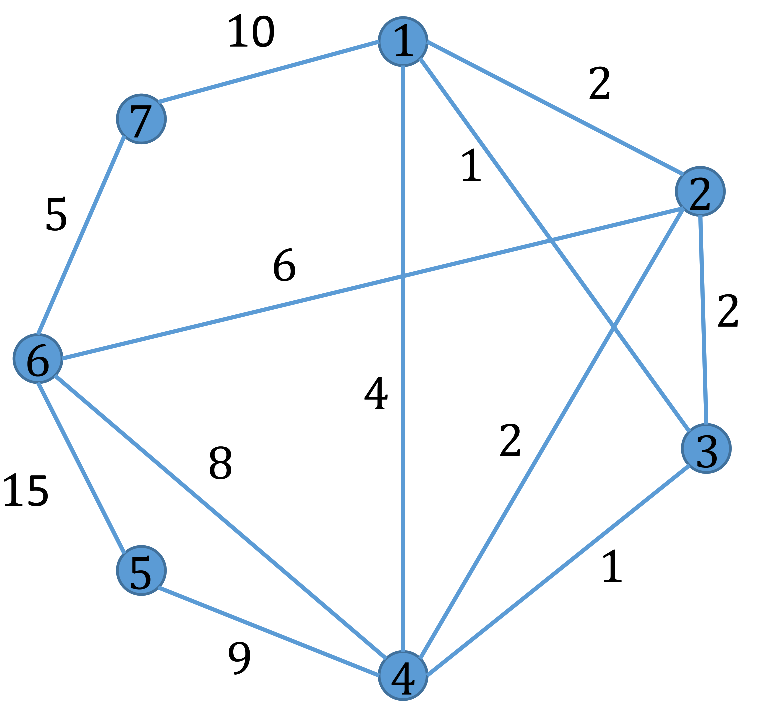
- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

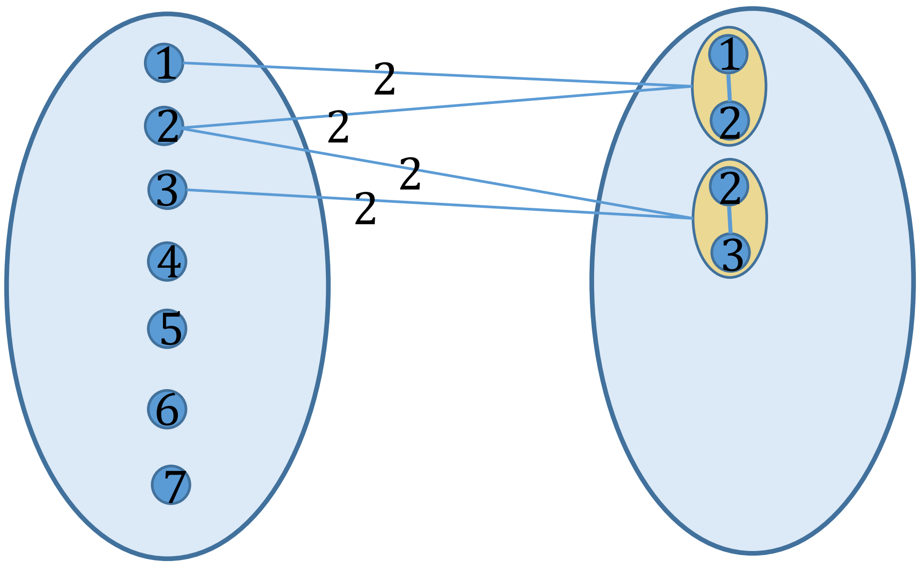




Solution

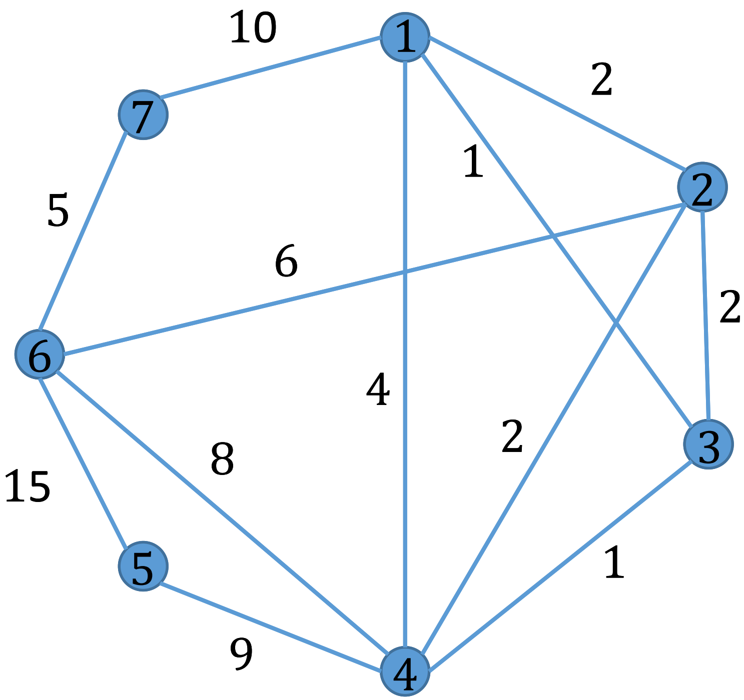
- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

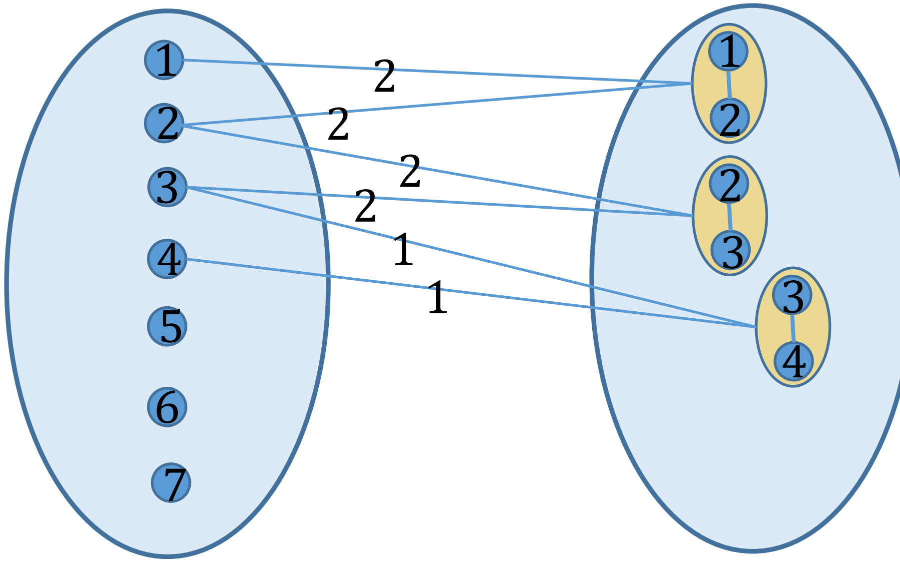




Solution

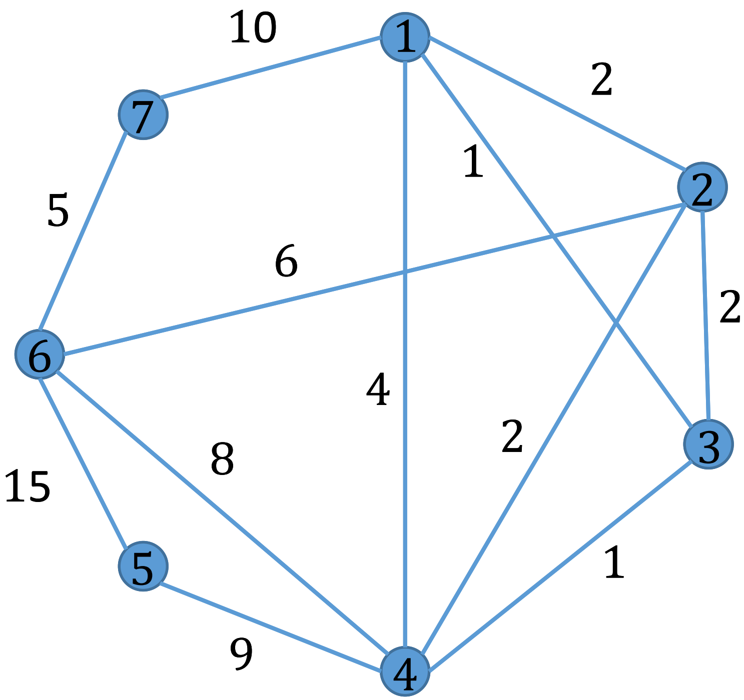
- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

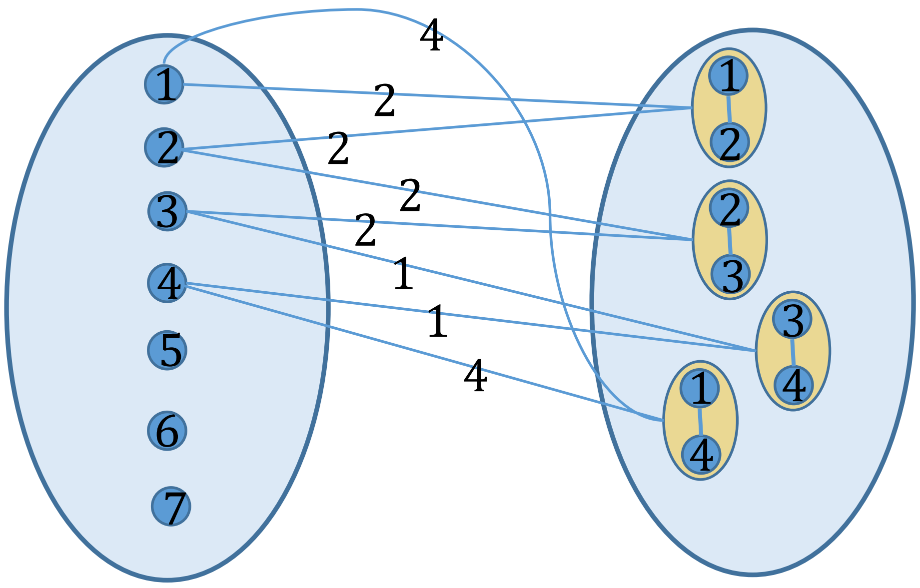




Solution

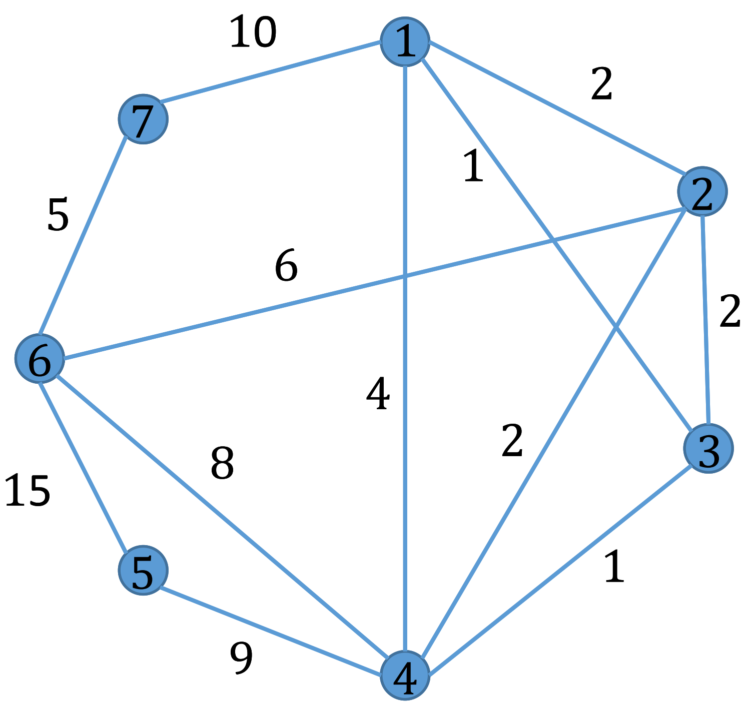
- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

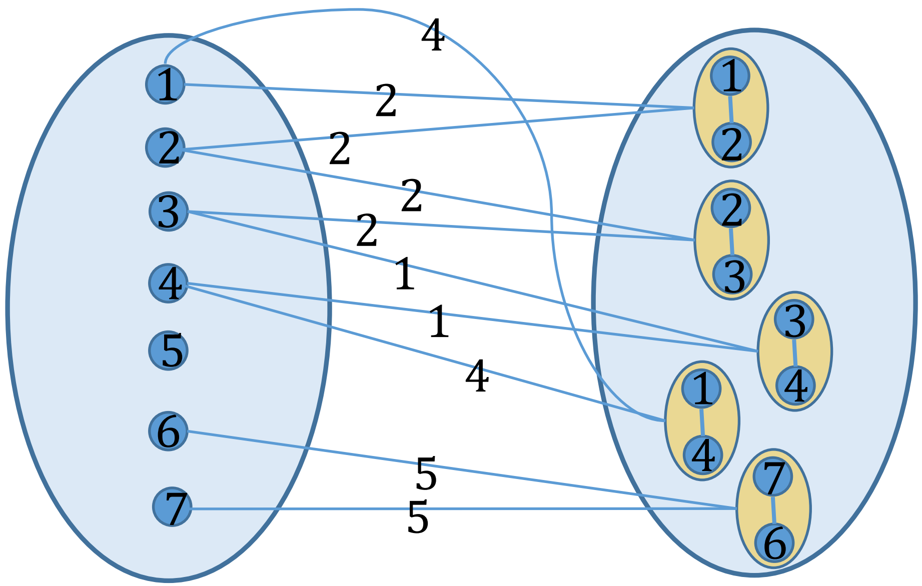




Solution

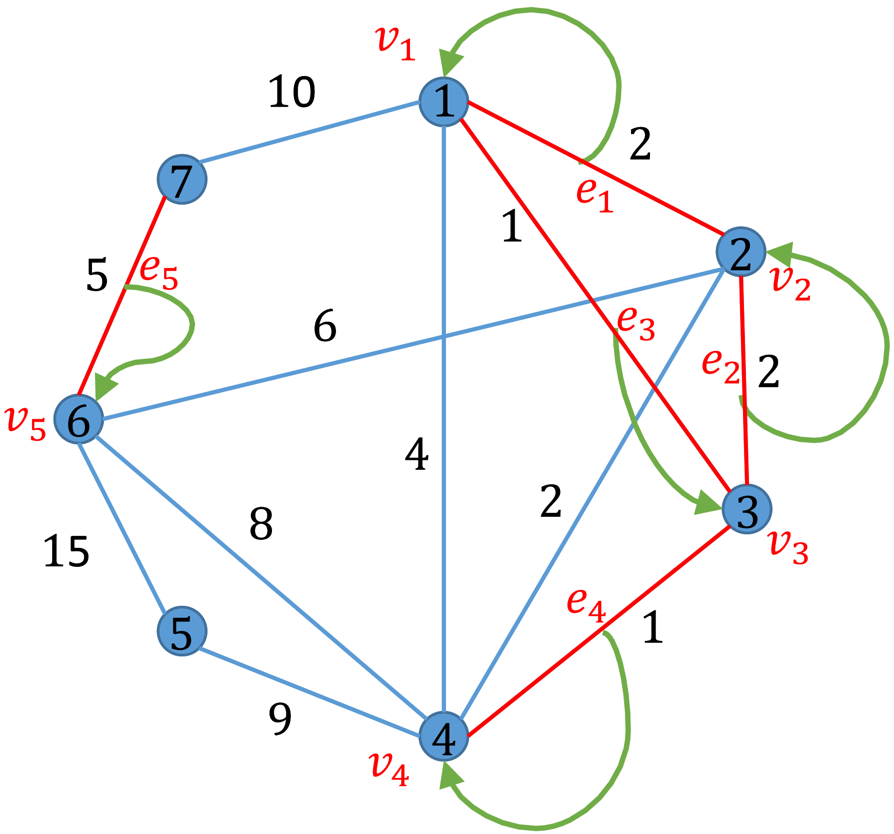
- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

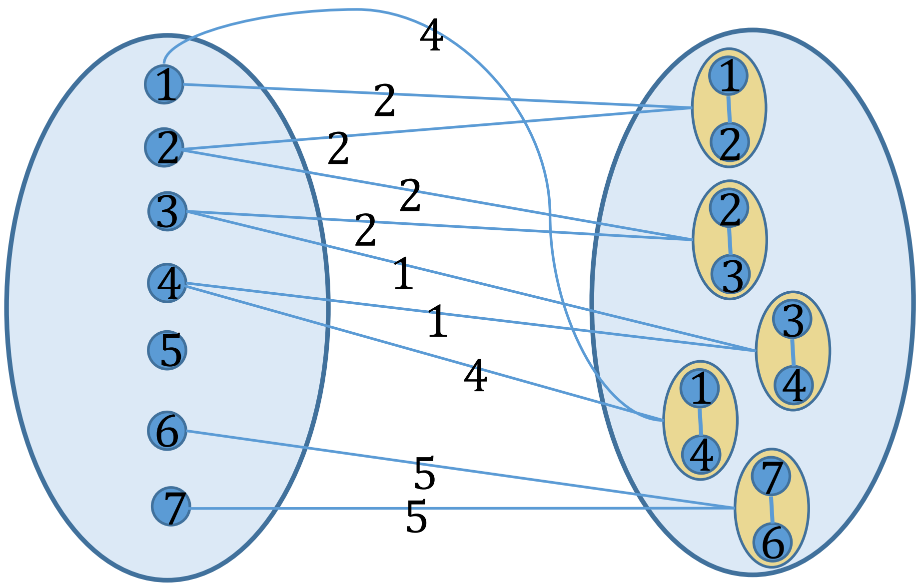




Solution

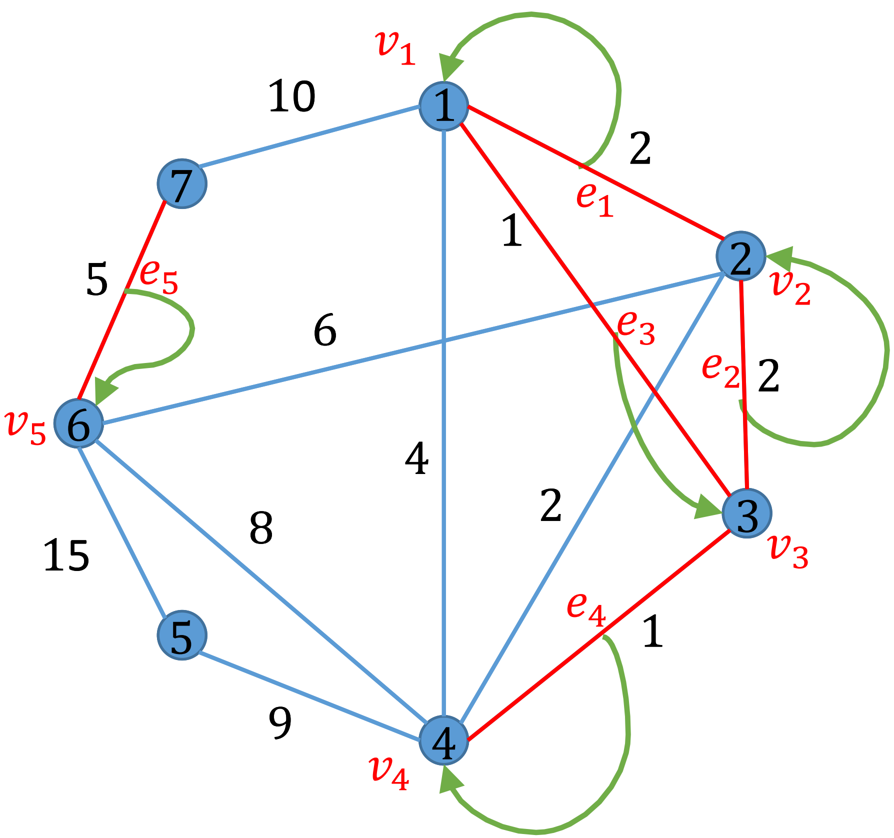
- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

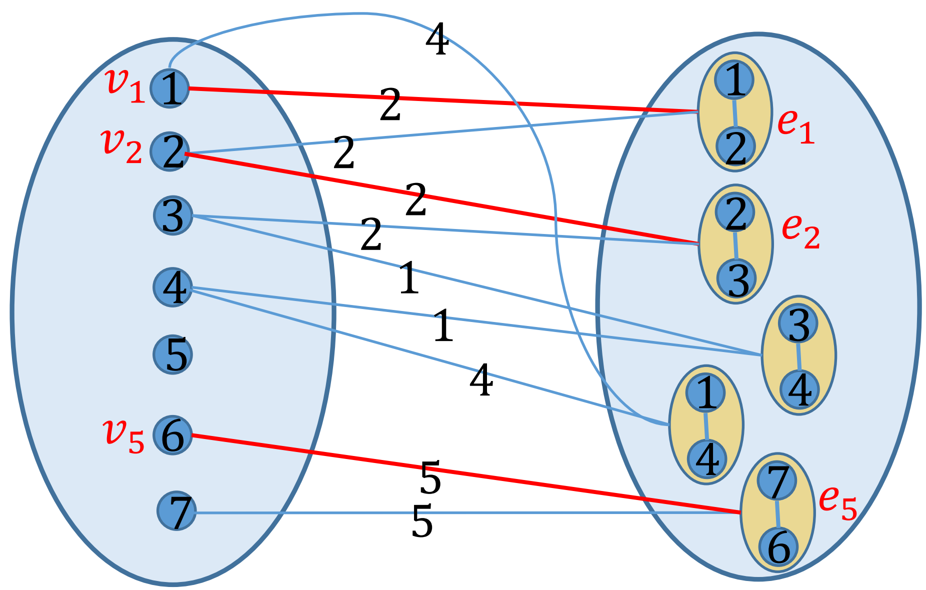




Solution

- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G





Solution

- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

- Find a cheapest matching of size k
  1. If all costs are the same = check if the maximum matching is of size at least k

Solution

- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

- Find a cheapest matching of size k
  1. If all costs are the same = check if the maximum matching is of size at least k
  2. Otherwise…

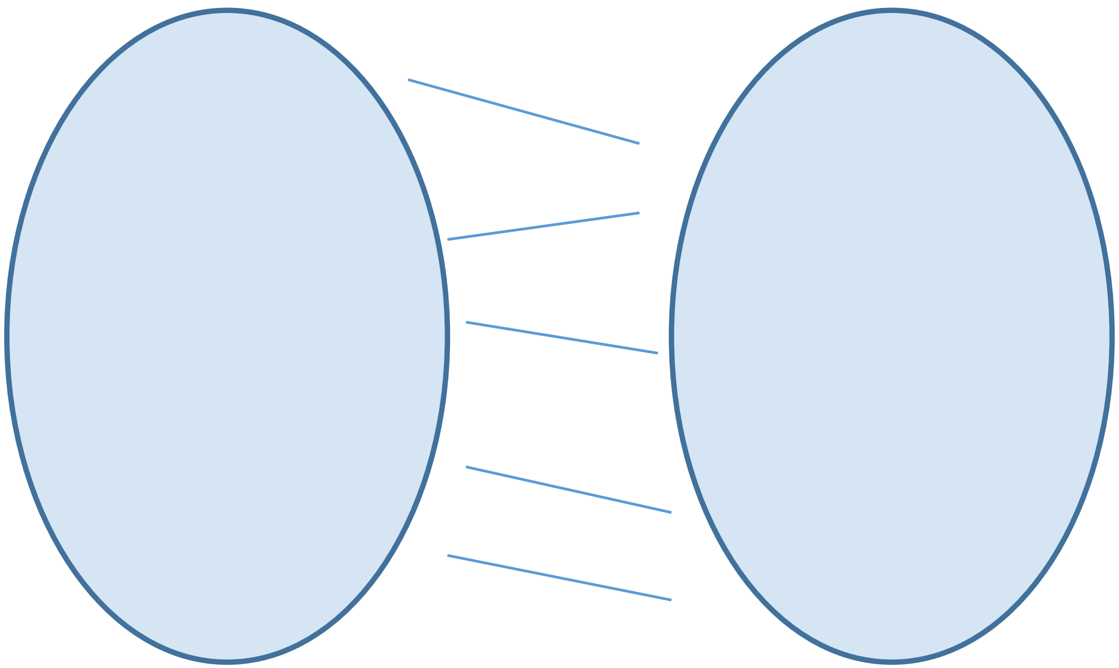
Solution

- Create auxiliary bipartite graph B:
  - one side – vertices of G
  - the other side – edges of G (now as vertices in B)
  - edge between vertex v and an edge e if they are adjacent in G

- Find a cheapest matching of size k
  1. If all costs are the same = check if the maximum matching is of size at least k
  2. Otherwise… reduce the problem to the **min-cost max-flow**

Solution

- Find a cheapest matching of size k
  1. If all costs are the same = check if the maximum matching is of size at least k
  2. Otherwise… reduce the problem to the **min-cost max-flow**



Solution

- Find a cheapest matching of size k
  1. If all costs are the same = check if the maximum matching is of size at least k
  2. Otherwise… reduce the problem to the **min-cost max-flow**