Partitioning Graphs of Supply and Demand

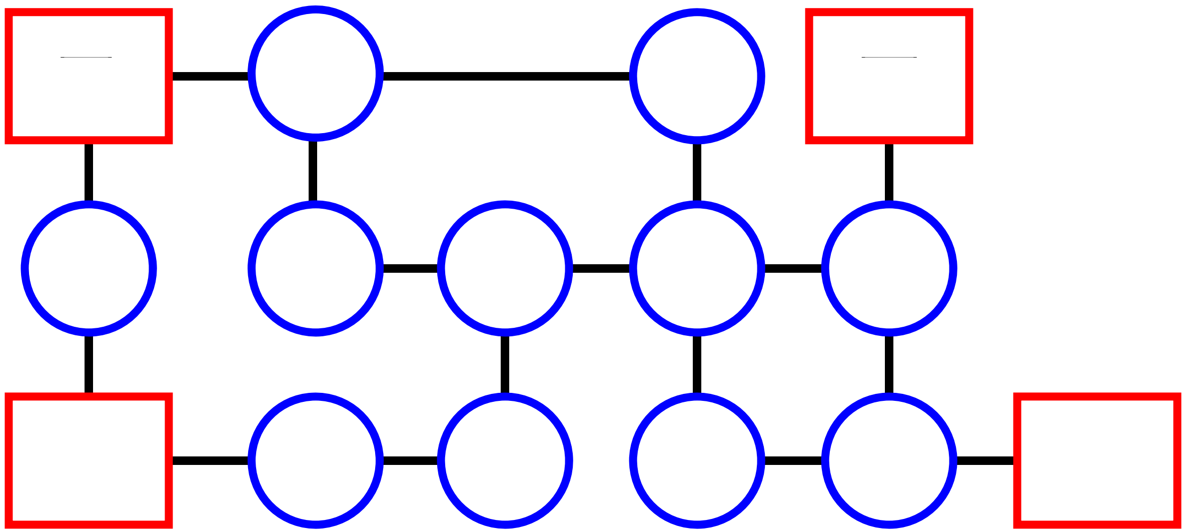
—— Generalization of Knapsack Problem ——

Takao Nishizeki

Tohoku University

Graph

Supply Vertices and Demand Vertices



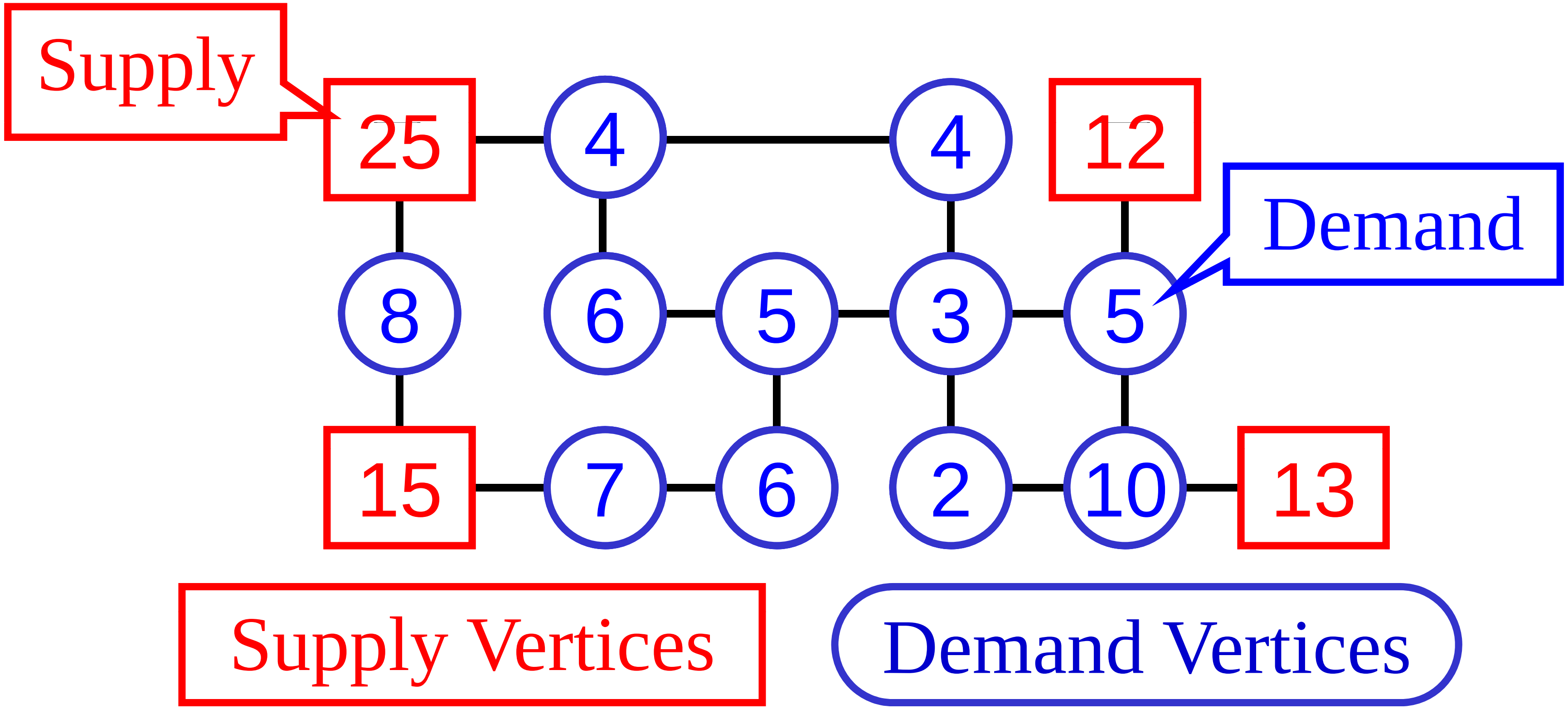
Supply Vertices

Demand Vertices

Graph

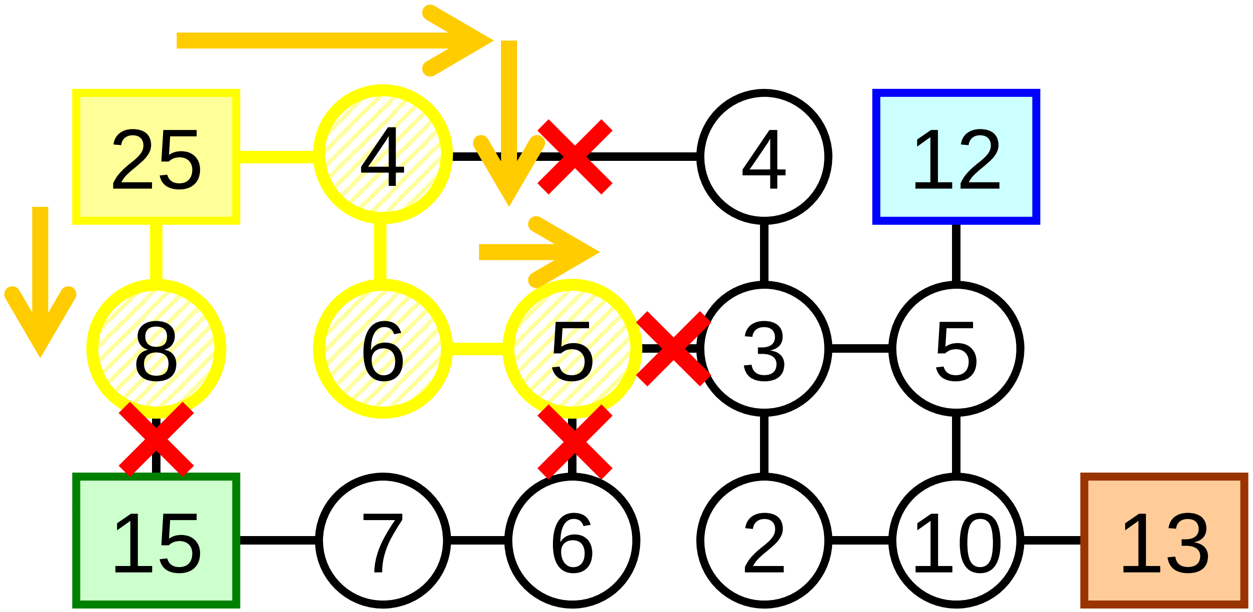
Each Supply Vertex has a number, called Supply.

Each Demand Vertex has a number, called Demand



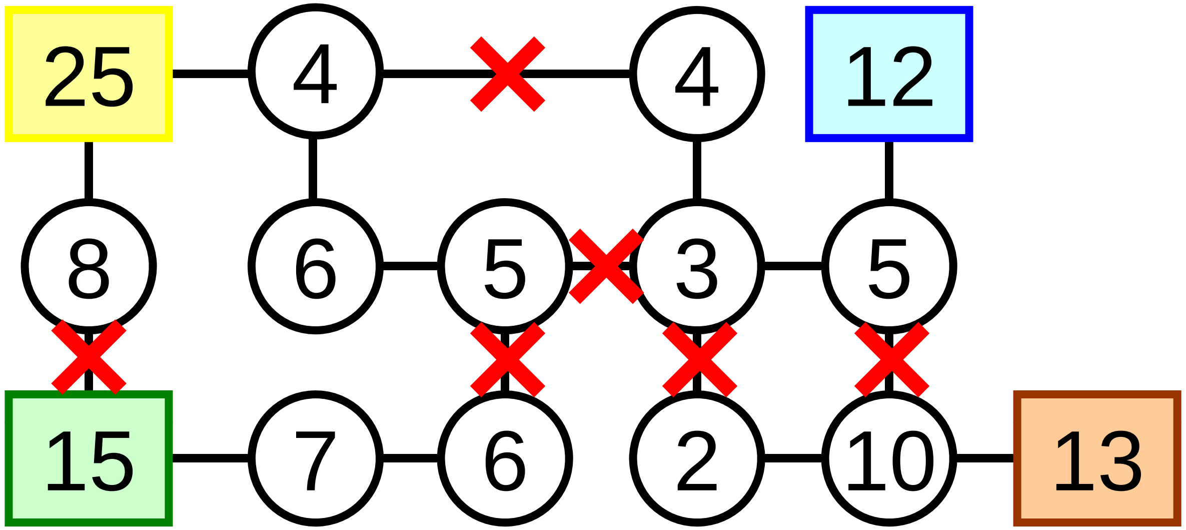
Desired Partition

partition *G* into connected components so that



Desired Partition

partition *G* into connected components so that



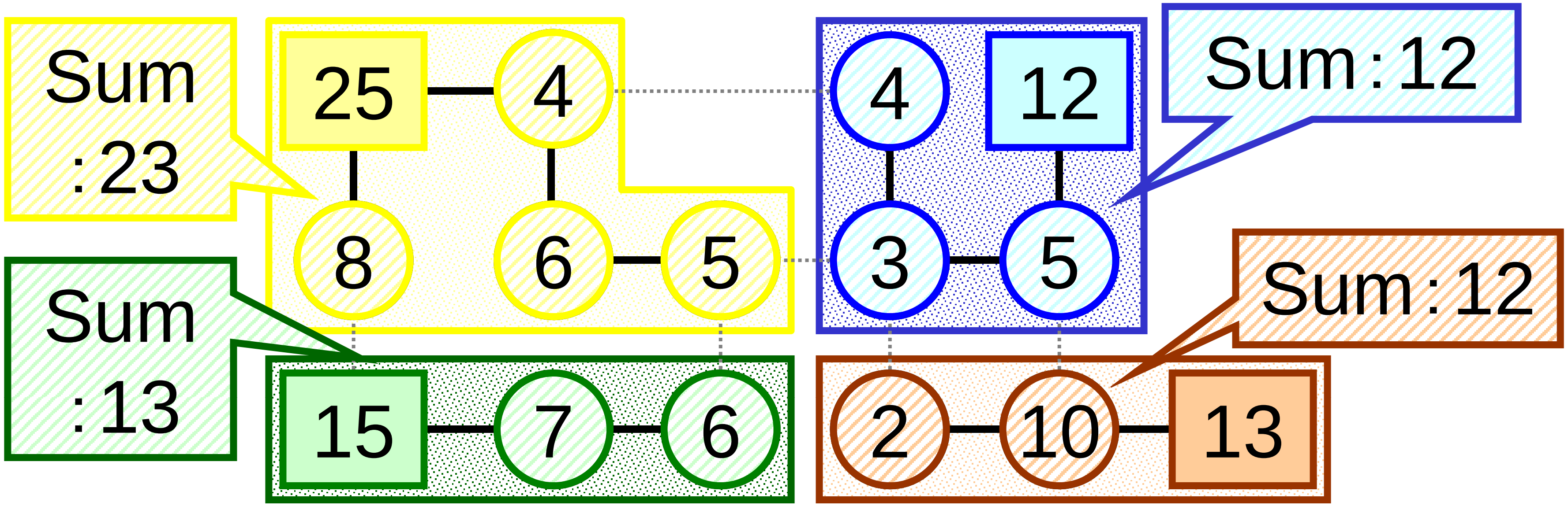
Desired Partition

partition *G* into connected components so that

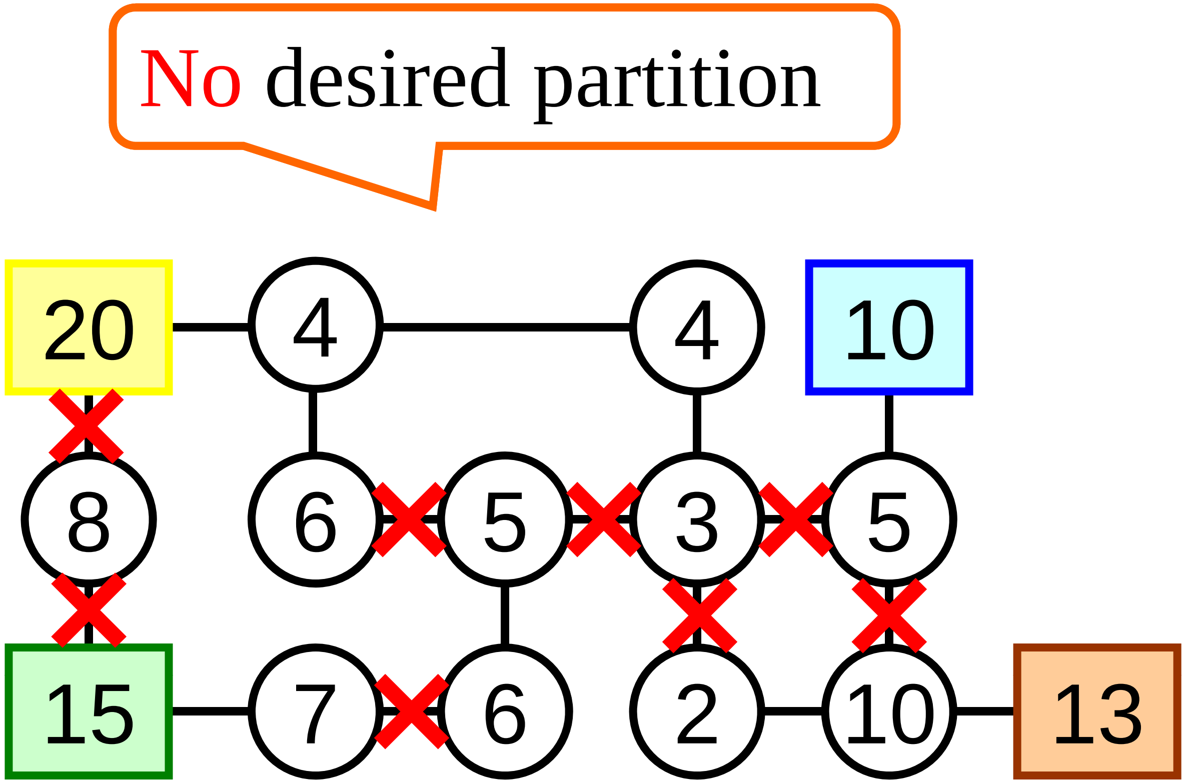
(a) each component has exactly one supply vertex,

(b) supply is no less than the sum of demands in the component.

Desired partition



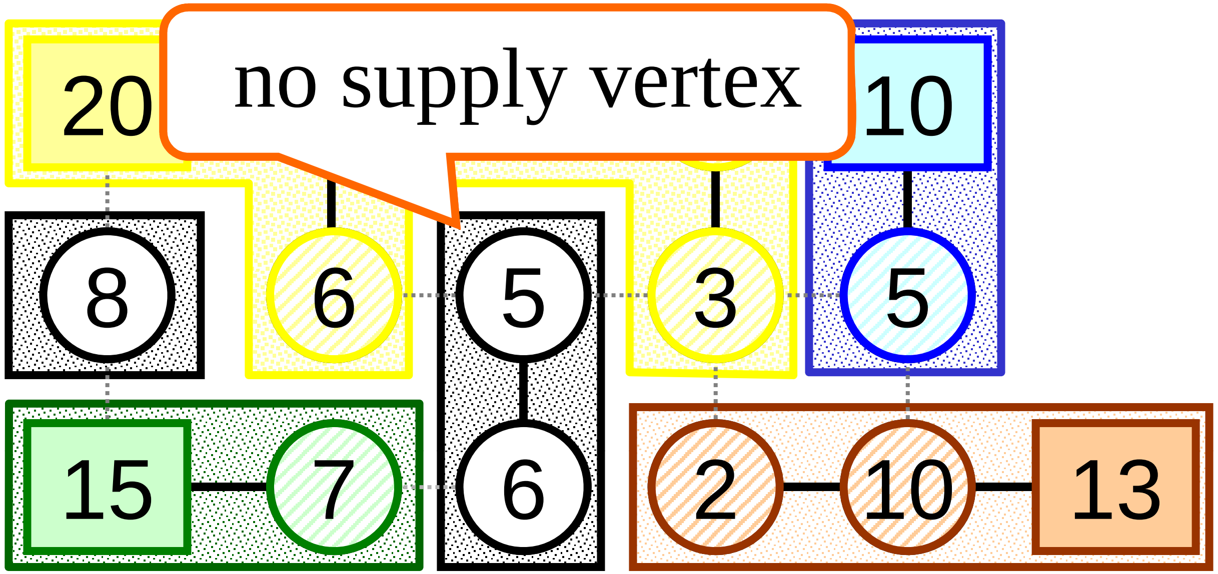
Maximum Partition Problem (Max PP)



Maximum Partition Problem (Max PP)

A partition of a graph must satisfy

(a) each component has at most one supply vertex,
(b) supply is no less than the sum of demands in the component.



Maximum Partition Problem (Max PP)

finds a partition that maximizes the "fulfillment."

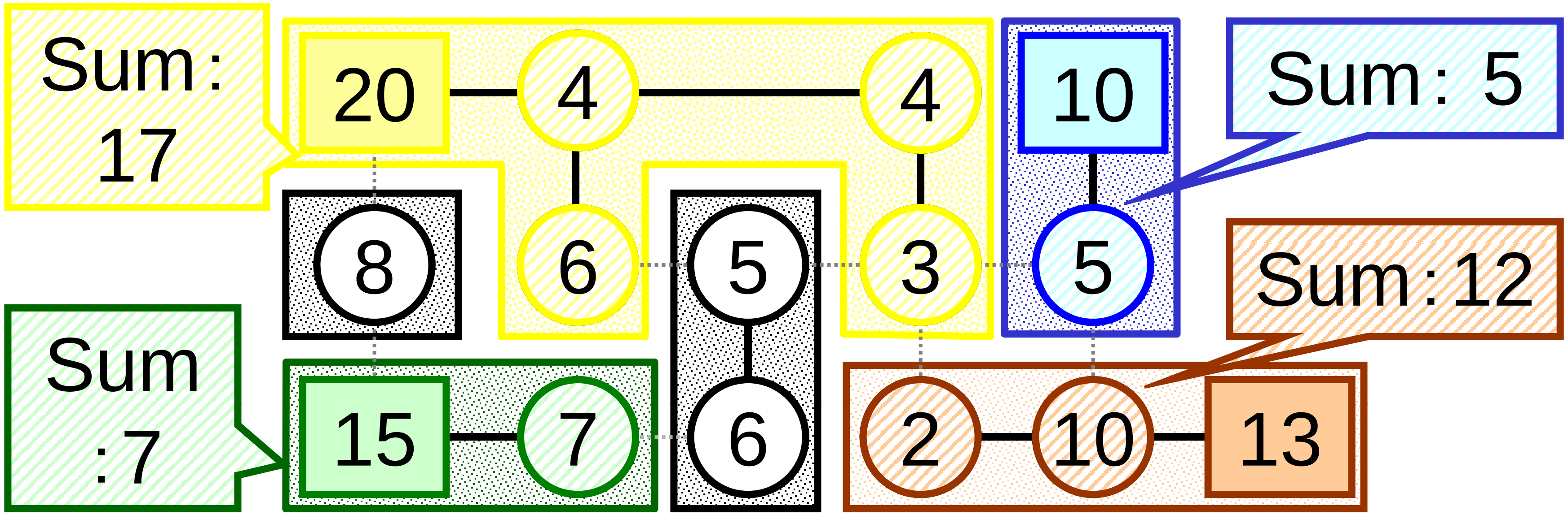
sum of demands in all components with supply vertices

A partition of a graph must satisfy

(a) each com

(b) supply is

component if there is a supply vertex.

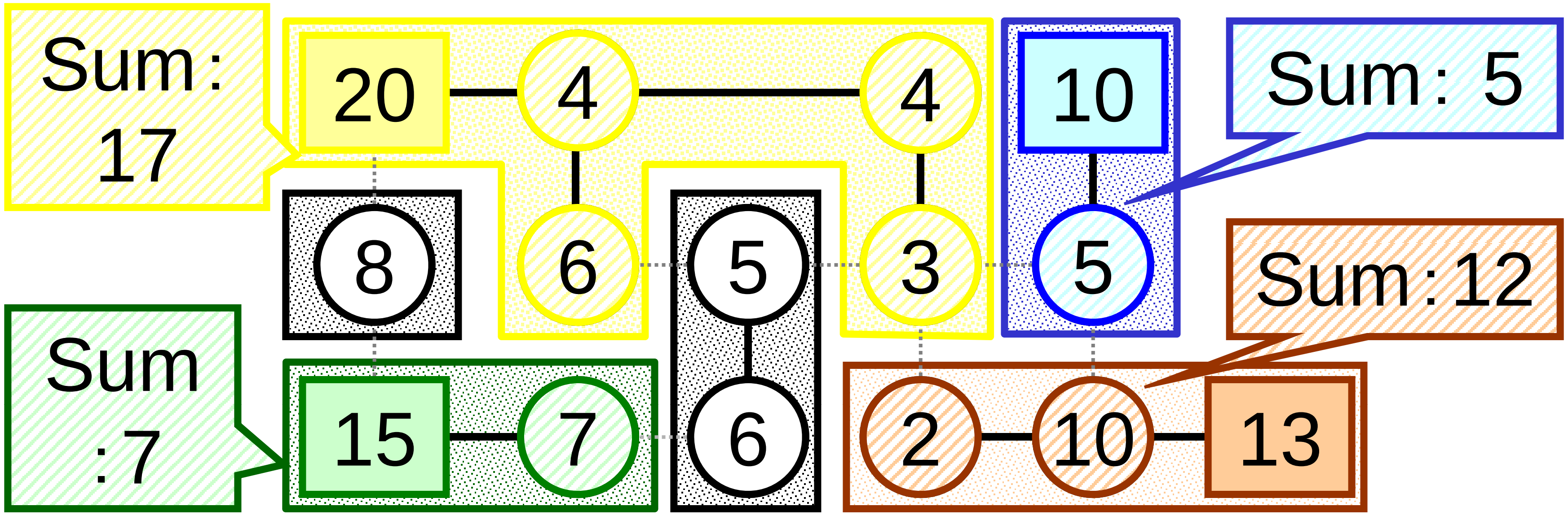


Maximum Partition Problem (Max PP)

finds a partition that maximizes the "fulfillment."

The fulfillment of this partition

17 + 5 + 12 + 7 = 41



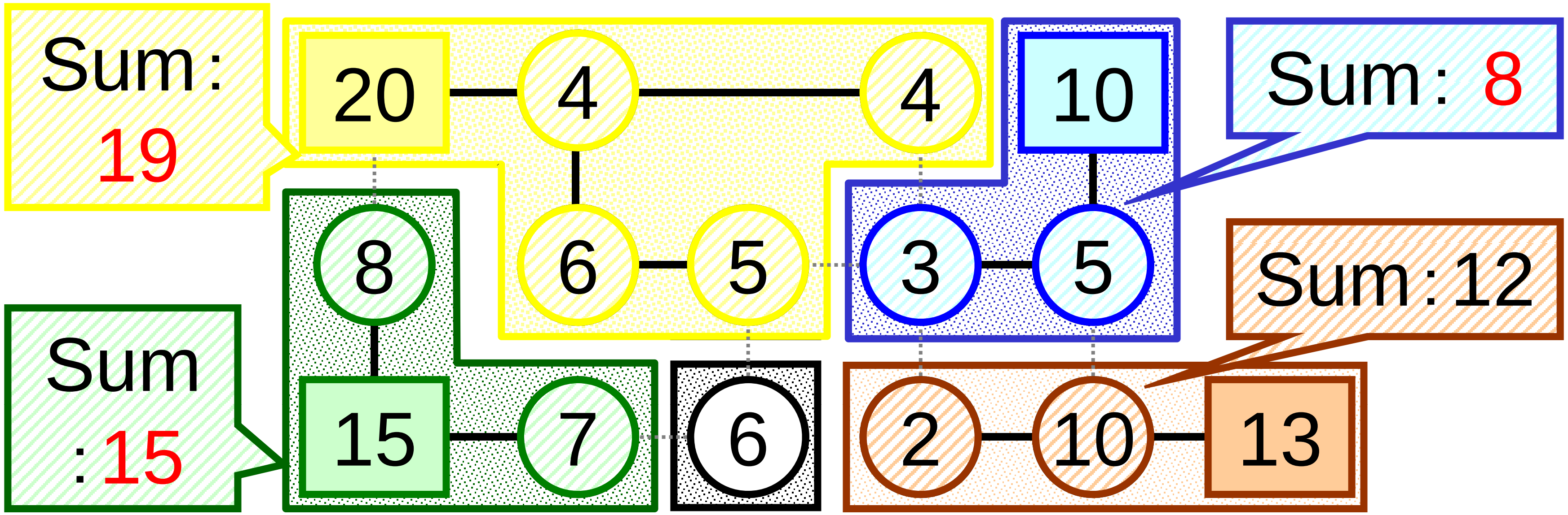
Maximum Partition Problem (Max PP)

finds a partition that maximizes the "fulfillment."

The fulfillment of this partition

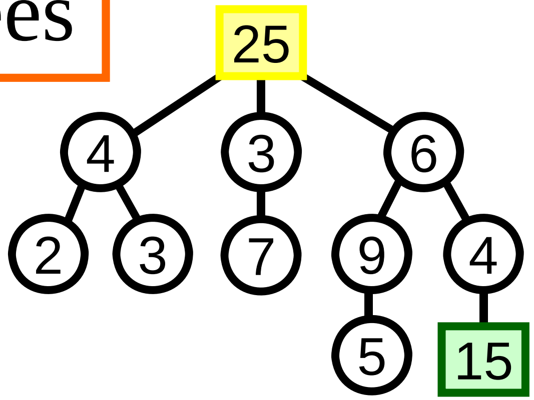
19 + 8 + 12 + 15 = 54

Maximum fulfillment



Complexity Status

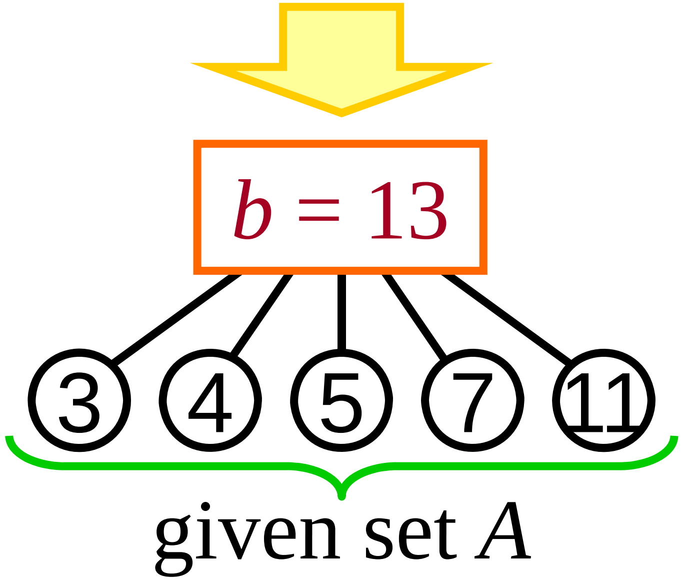
Trees



NP-hard

max subset sum problem

(simple ver. of Knapsack)

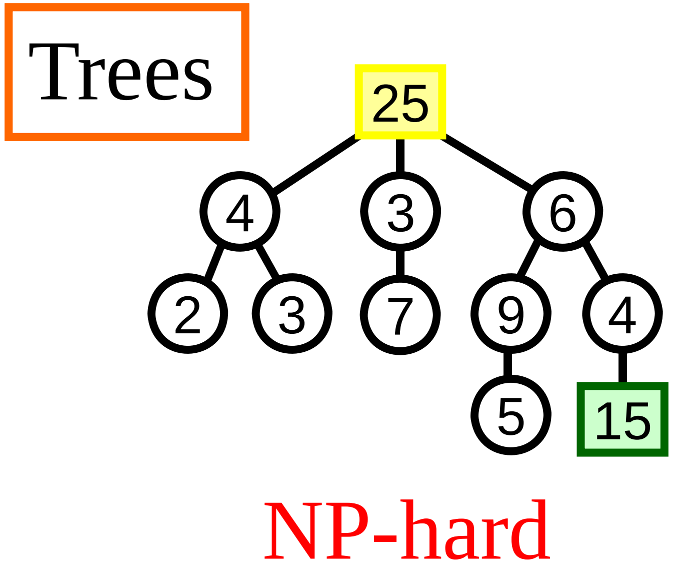


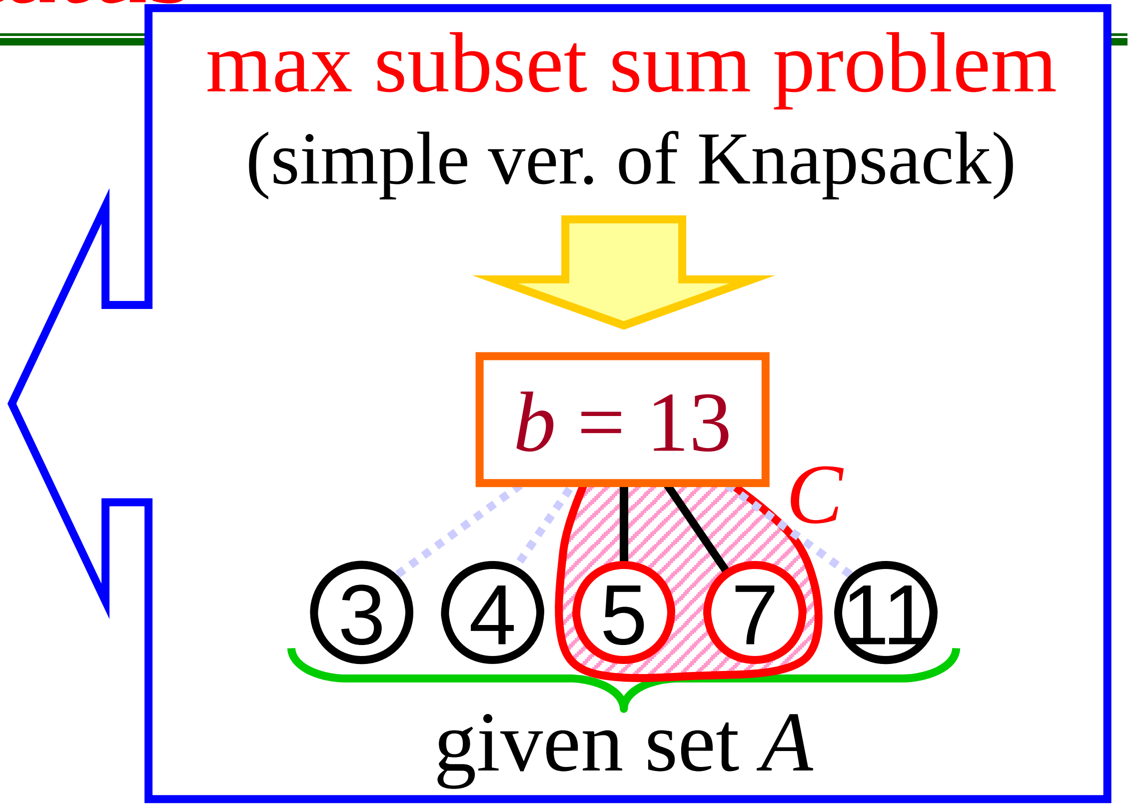
Maximum Subset Sum Problem (NP-hard)

instance: a set A of integers and an integer b

find: a subset $C \subseteq A$ which maximizes the sum of integers in C s.t. the sum does not exceed b.

Complexity Status





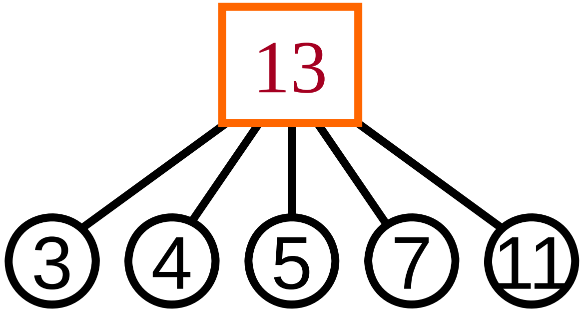
Maximum Subset Sum Problem (NP-hard)

instance: a set A of integers and an integer b

find: a subset $C \subseteq A$ which maximizes the sum of integers in C s.t. the sum does not exceed b.

Related Result

max subset sum problem (NP-hard)



Max PP for Stars
with one supply at center

Fully **P**olynomial-**T**ime **A**pproximation **S**cheme (FPTAS)

[Ibarra and Kim '75]

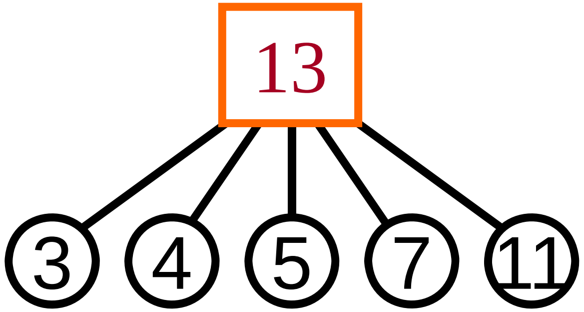
FPTAS: for any ε, $0 < \varepsilon < 1$, the algorithm finds an approximation solution such that

$$\text{APPRO} > (1-\varepsilon)\ \text{OPT}$$

in time polynomial in both n and $1/\varepsilon$.

Related Result

max subset sum problem (NP-hard)



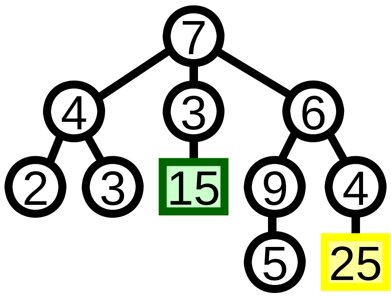
Max PP for Stars

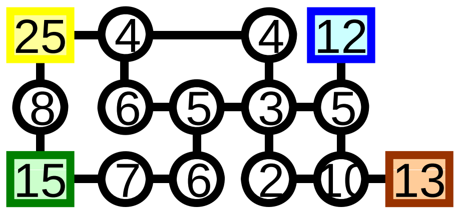
with one supply at center

Fully **P**olynomial-**T**ime **A**pproximation **S**cheme (FPTAS)

[Ibarra and Kim ’75]

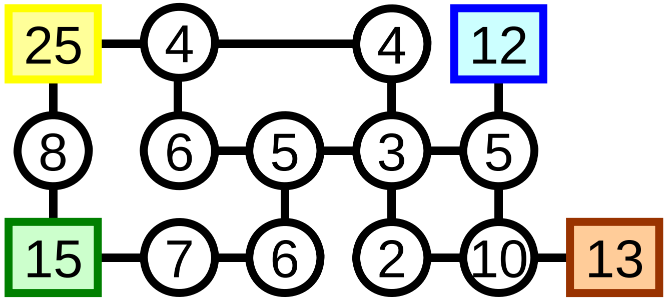
good approximation for larger classes ?





Our Results (approximabiliy)

General graphs



(1) MAXSNP-hard
(APX-hard)

No PTAS unless P=NP

No FPTAS unless P=NP

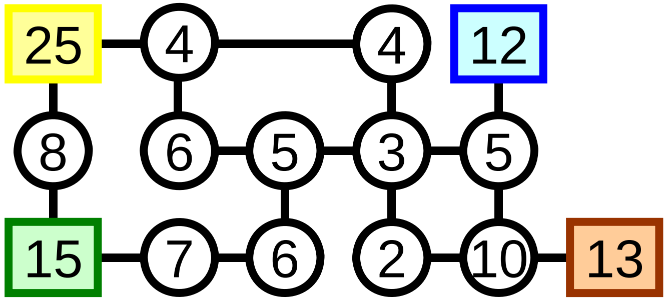
PTAS: for any ε, $0 < \varepsilon < 1$, the algorithm finds an approximation solution such that

$$\text{APPRO} > (1-\varepsilon)\ \text{OPT}$$

in time polynomial in n. ($1/\varepsilon$:regarded as a constant.)

Our Results (approximabiliy)

General graphs

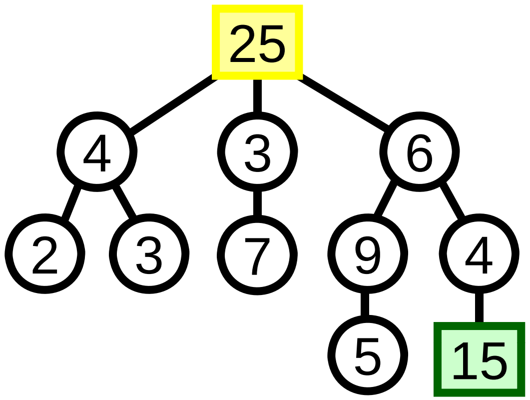


(1) MAXSNP-hard

(APX-hard)

No PTAS unless P=NP

Trees

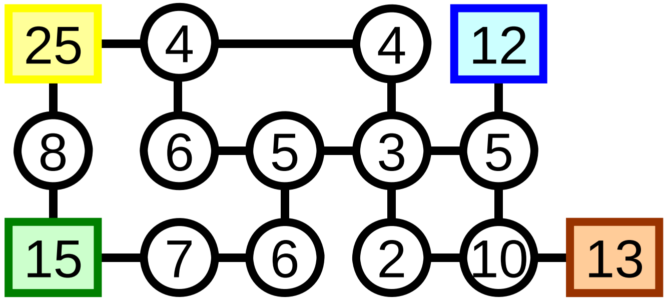


NP-hard

(2) FPTAS

Our Results (approximabiliy)

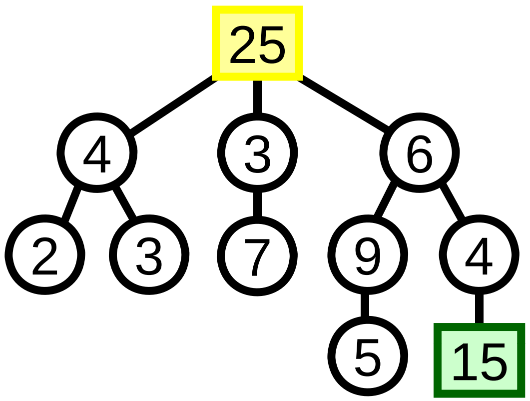
General graphs



(1) MAXSNP-hard
(APX-hard)

No PTAS unless P=NP

Trees



NP-hard

(2) FPTAS

(1) MAXSNP-hardness

Max PP is MAXSNP-hard for bipartite graphs.

L-reduction: preserves approximability

error ratio: ε'

3-occurrence MAX3SAT

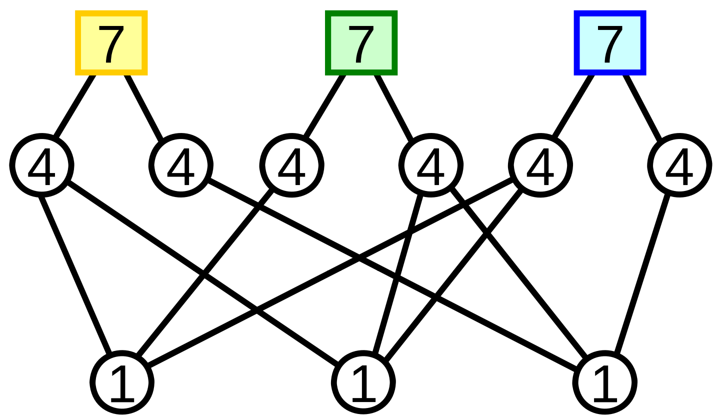
L-reduction

error ratio: ε

Max PP for bipartite graphs

each variable appears
at most 3 times

$$f = (x \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$



(1) MAXSNP-hardness

3-occurrence MAX3SAT (MAXSNP-hard)

instance: variables and clauses s.t.

- each clause has exactly 3 literals; and
- each variable appears at most 3 times in the clauses

find: a truth assignment which maximizes # of satisfied clauses

variables: $v \quad w \quad x \quad y \quad z$

at most 3 times

$$f = (v \vee w \vee y) \wedge (\overline{w} \vee \overline{x} \vee z) \wedge (\overline{v} \vee w \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

(1) MAXSNP-hardness

3-occurrence MAX3SAT (MAXSNP-hard)

instance: variables and clauses s.t.

- each clause has exactly 3 literals; and
- each variable appears at most 3 times in the clauses

find: a truth assignment which maximizes # of satisfied clauses

variables: $v \quad w \quad x \quad y \quad z$

at most 3 times

$$f = (v \vee w \vee y) \wedge (\overline{w} \vee \overline{x} \vee z) \wedge (\overline{v} \vee w \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

(1) MAXSNP-hardness

3-occurrence MAX3SAT (MAXSNP-hard)

instance: variables and clauses s.t.

- each clause has exactly 3 literals; and
- each variable appears at most 3 times in the clauses

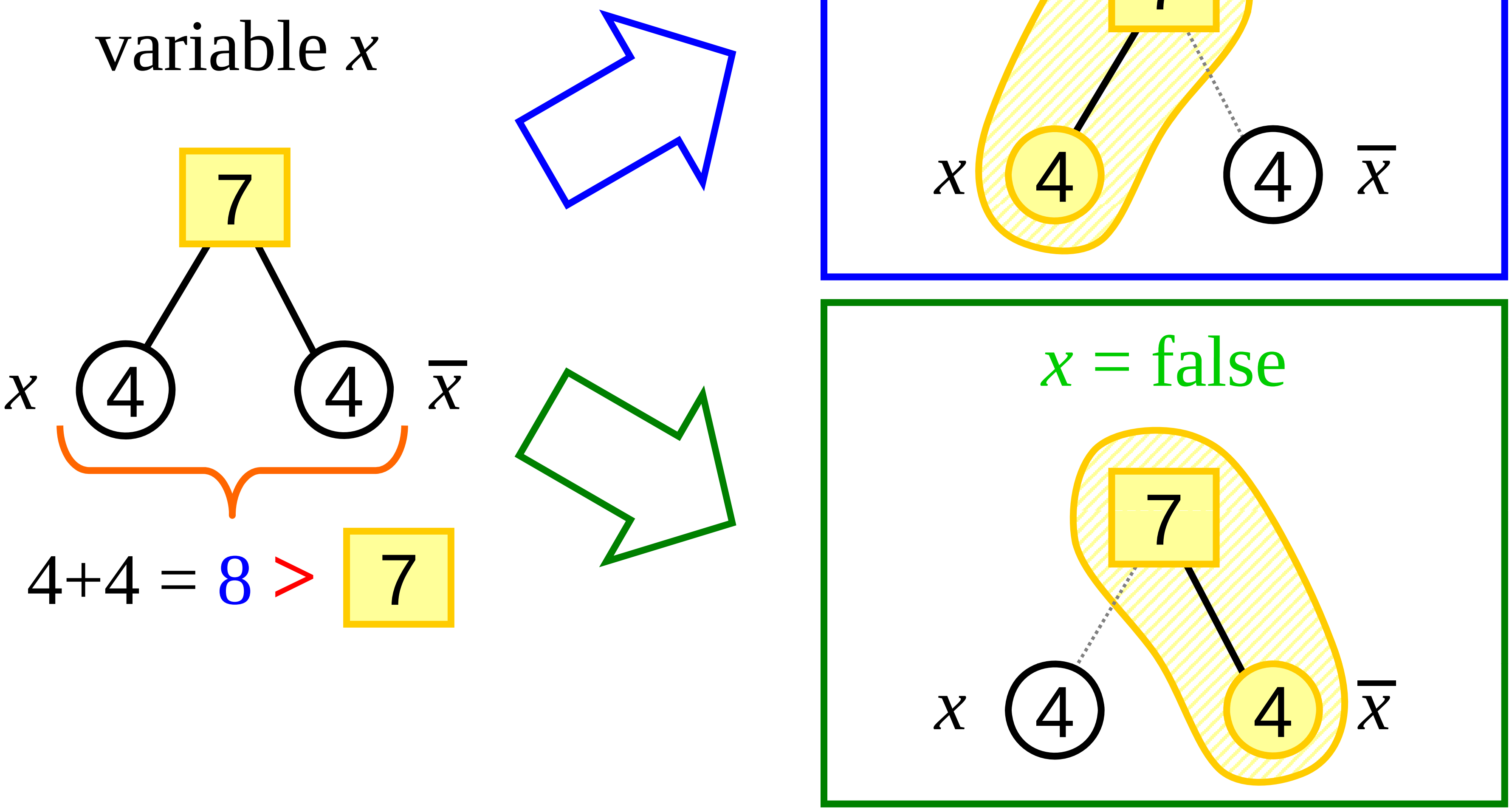
find: a truth assignment which maximizes # of satisfied clauses

variables: $v \quad w \quad x \quad y \quad z$

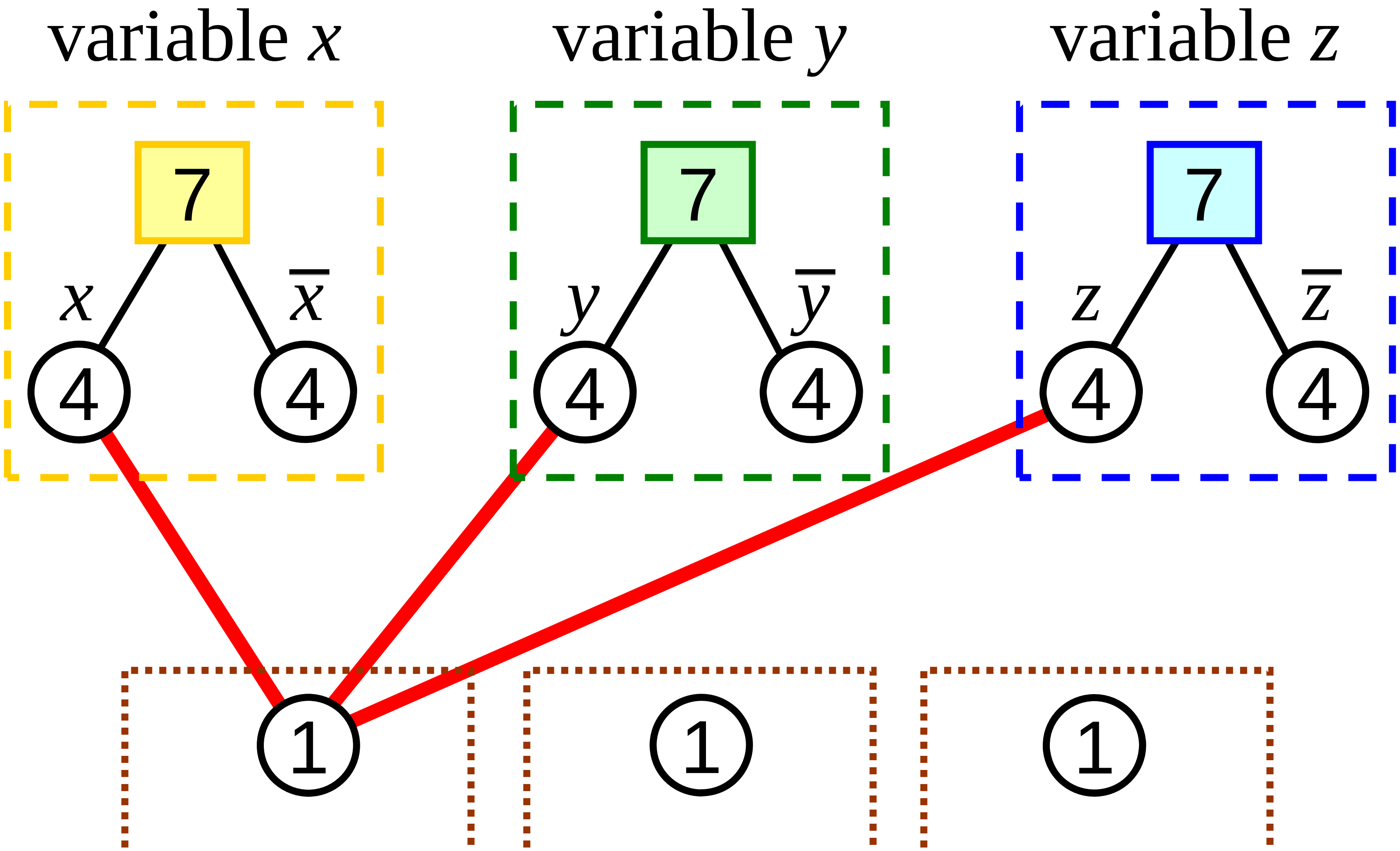
$$f = (v \vee w \vee y) \wedge (\overline{w} \vee \overline{x} \vee z) \wedge (\overline{v} \vee w \vee \overline{z}) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

(1) MAXSNP-hardness

variable gadget

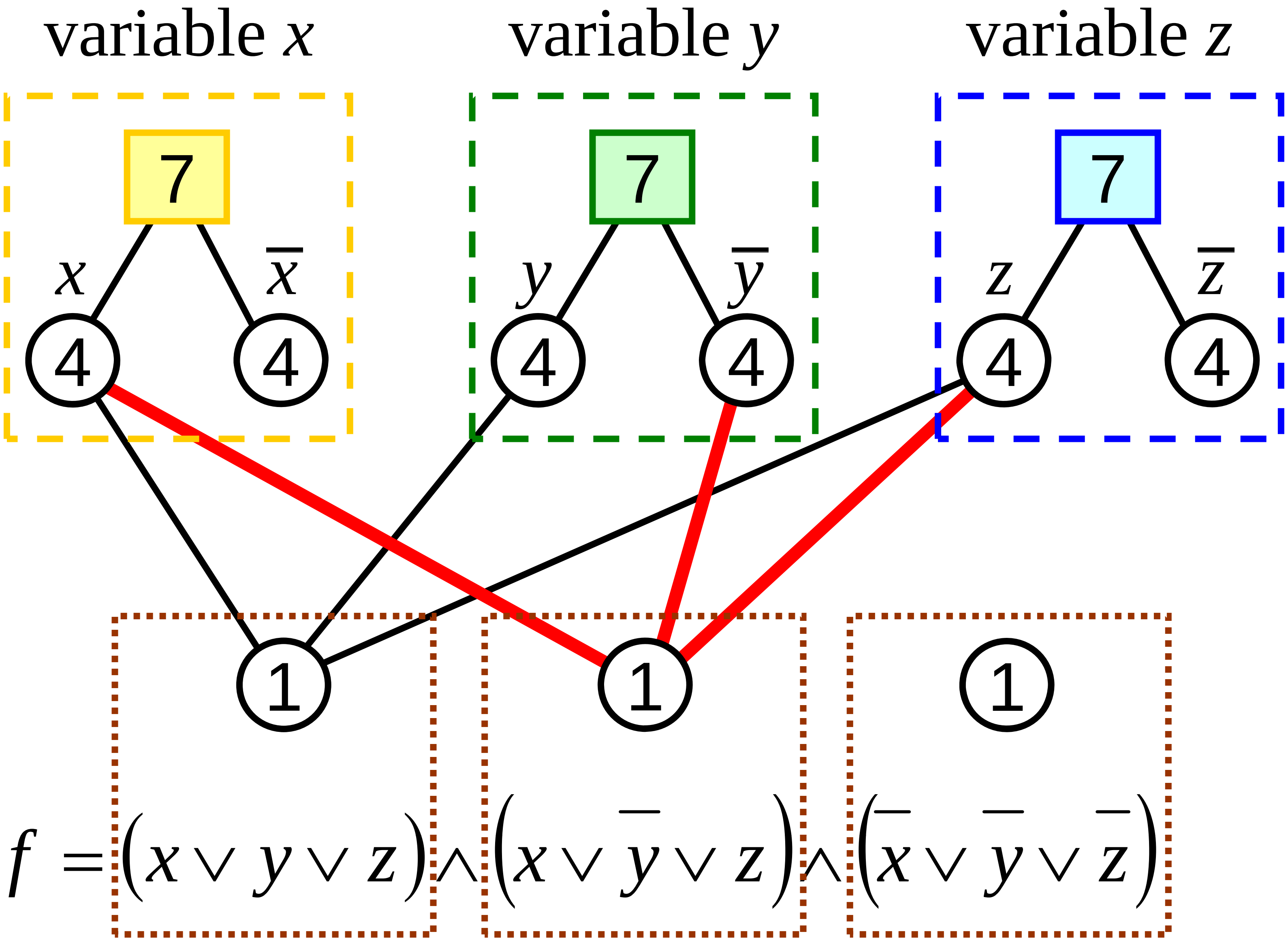


(1) MAXSNP-hardness



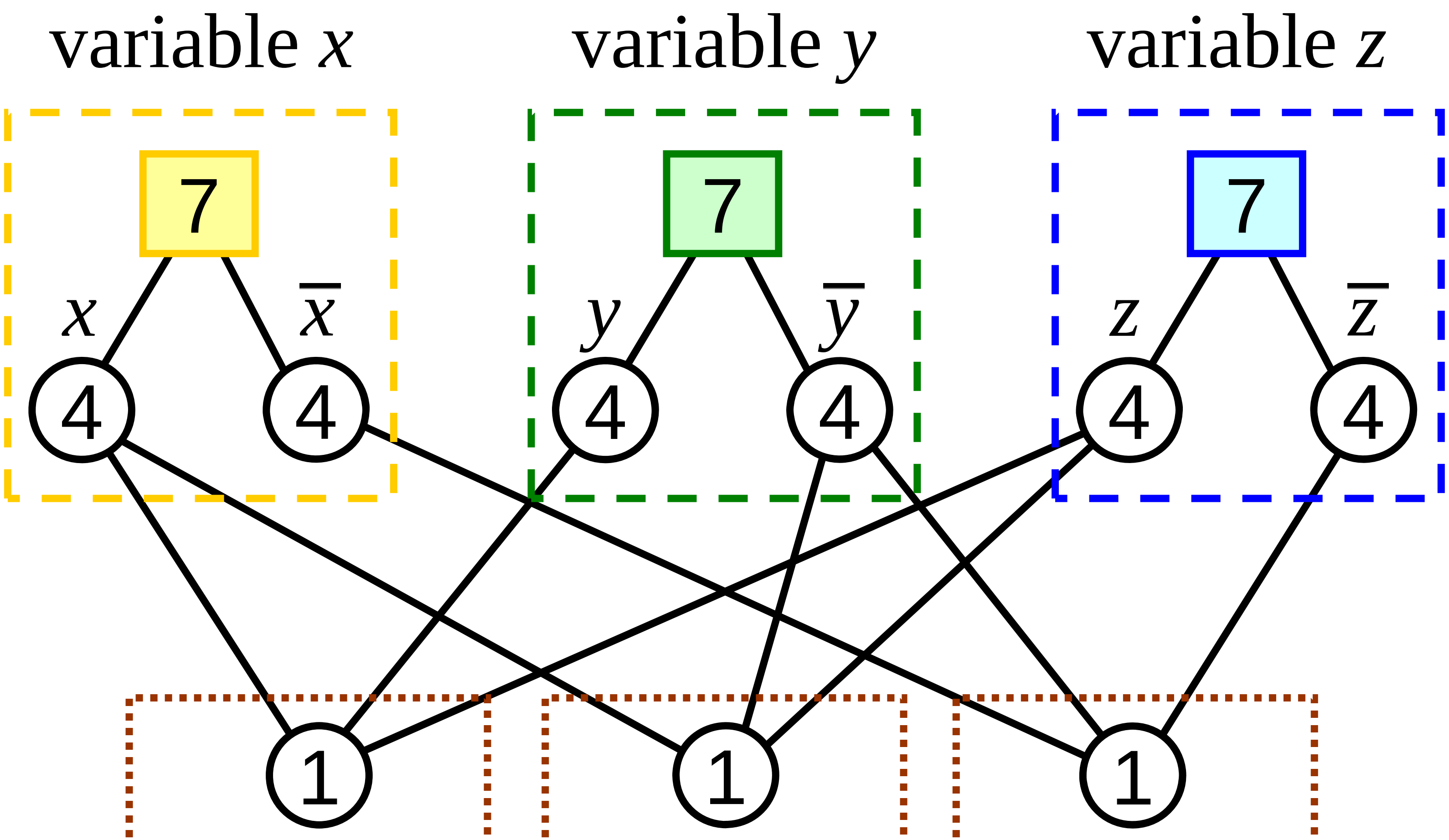
$$f = \left(x \vee y \vee z\right) \wedge \left(x \vee \overline{y} \vee z\right) \wedge \left(\overline{x} \vee \overline{y} \vee \overline{z}\right)$$

(1) MAXSNP-hardness



$$f = (x \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

(1) MAXSNP-hardness



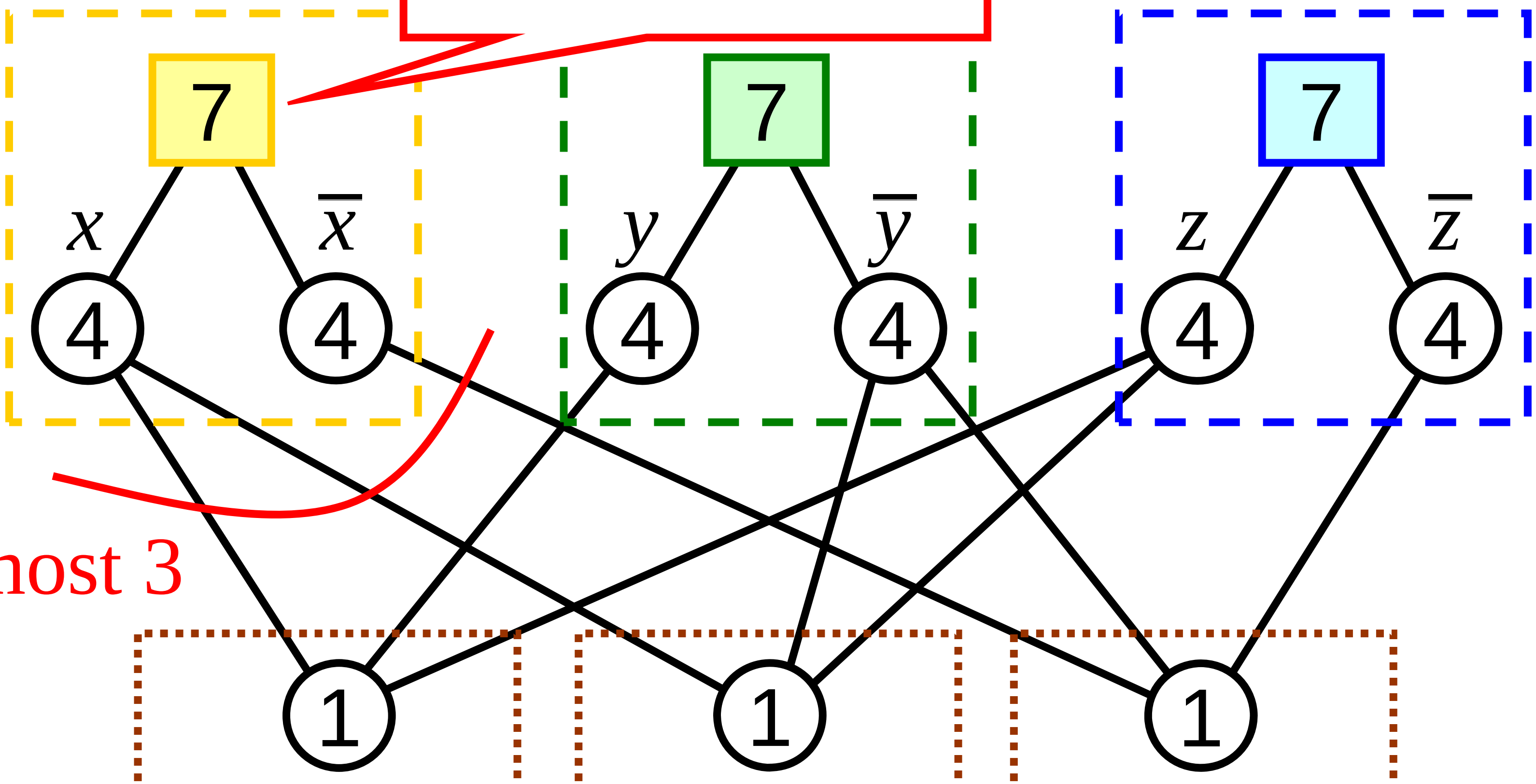
$$f = \left(x \vee y \vee z\right) \wedge \left(x \vee \overline{y} \vee z\right) \wedge \left(\overline{x} \vee \overline{y} \vee \overline{z}\right)$$

(1) MAXS

7 – 4 = **3**
enough power

variable x

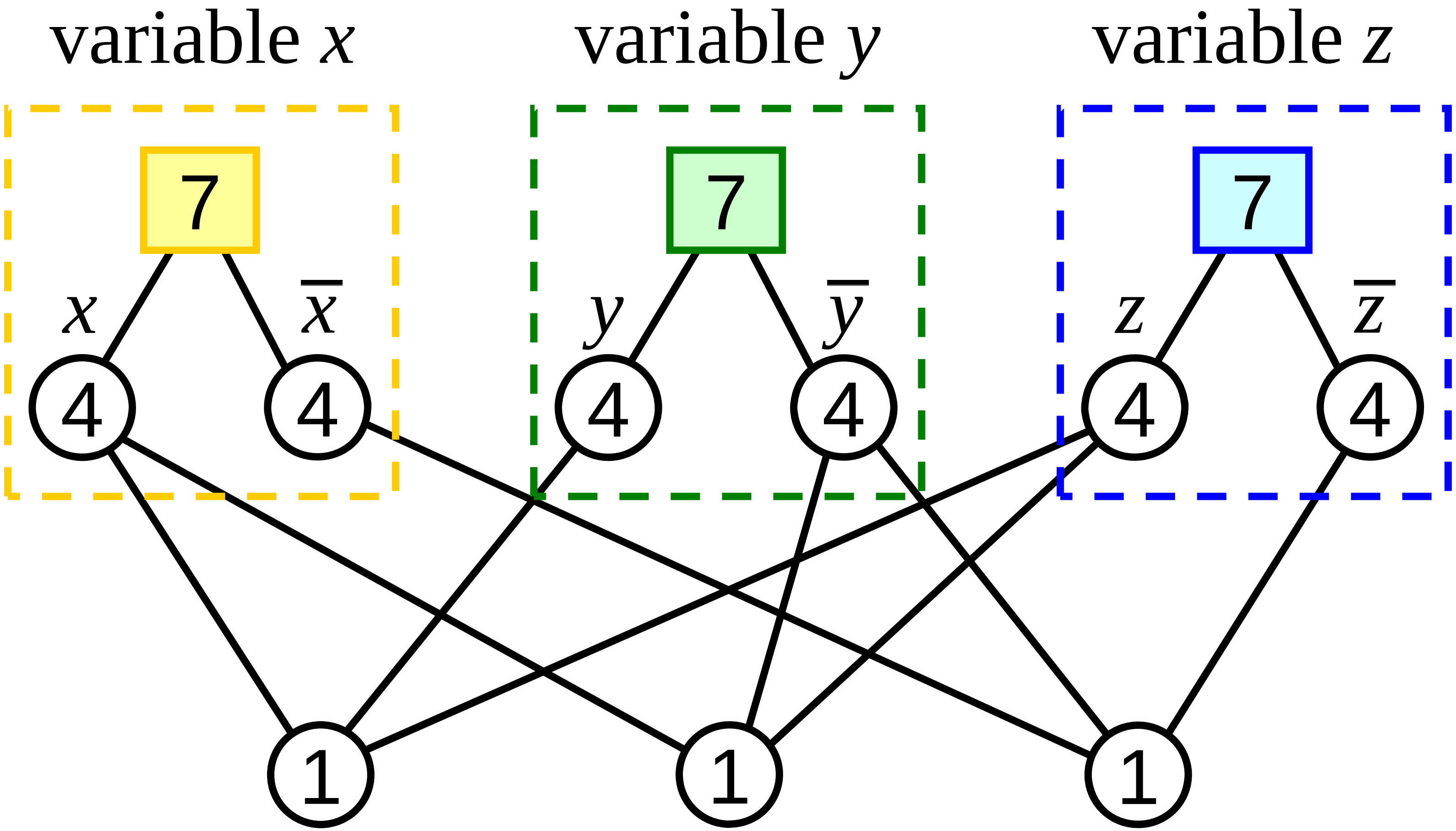
variable z



at most 3

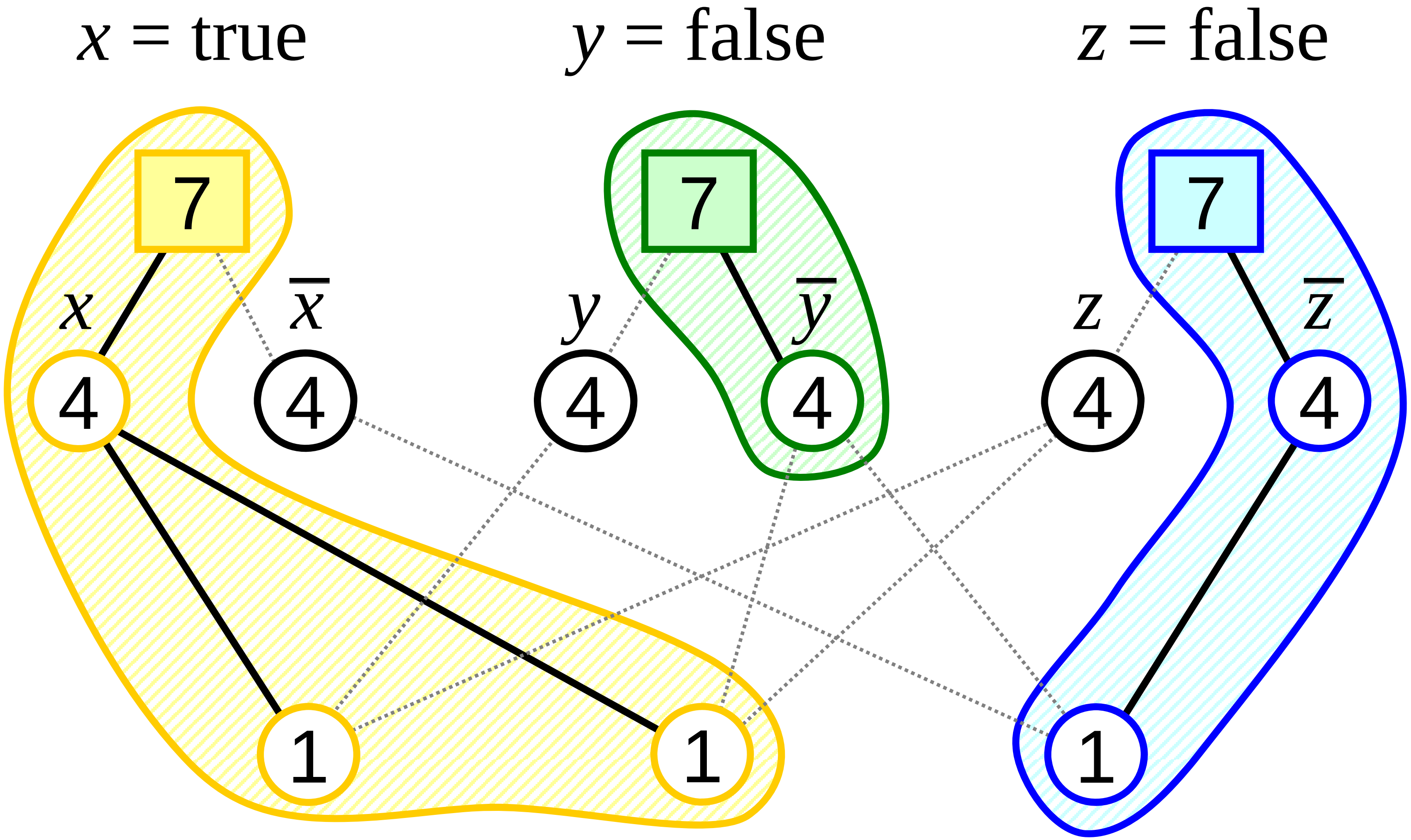
$$f = (x \vee y \vee z) \wedge (x \vee \bar{y} \vee z) \wedge (\bar{x} \vee \bar{y} \vee \bar{z})$$

(1) MAXSNP-hardness



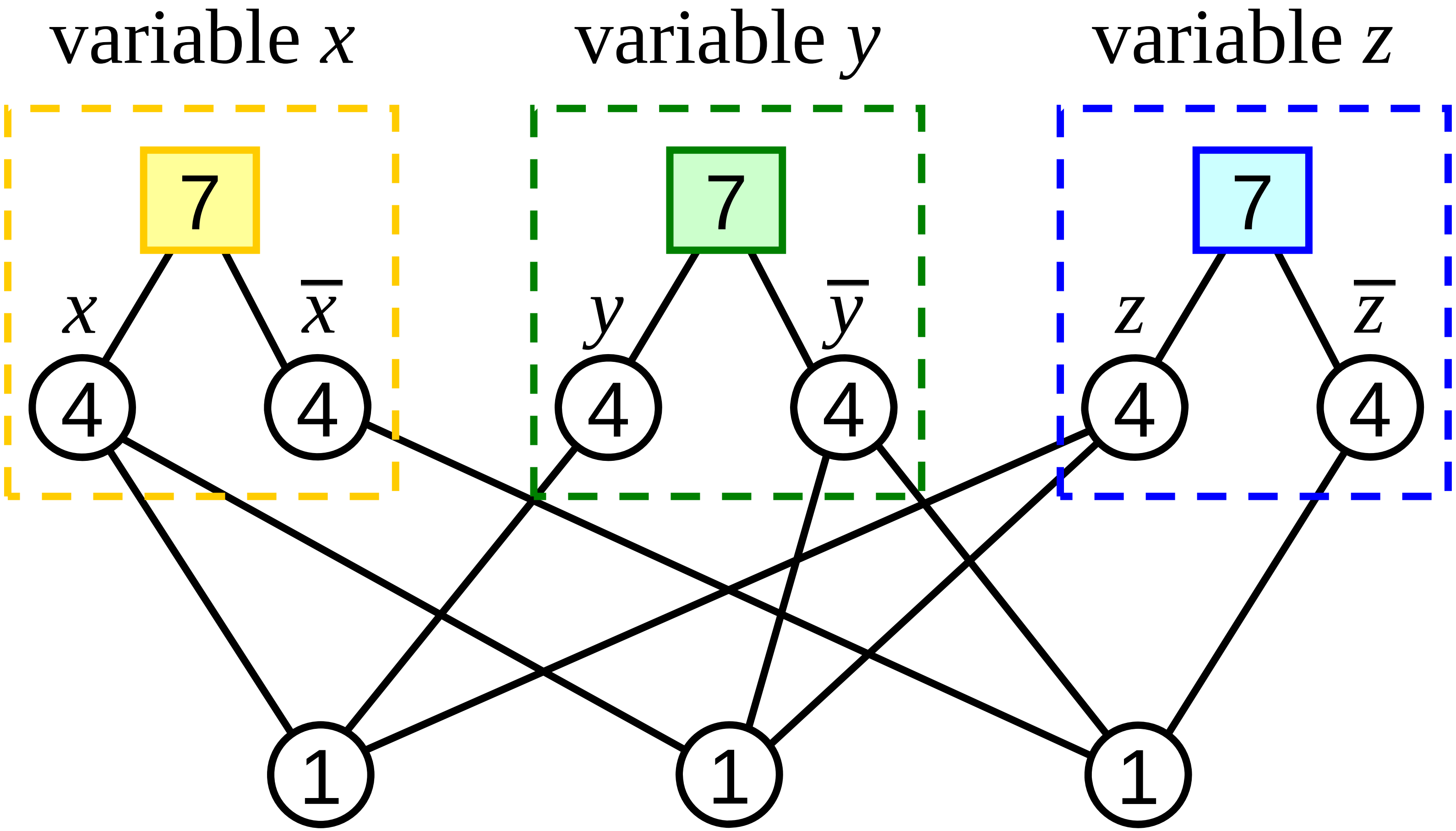
$$f = (x \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

(1) MAXSNP-hardness



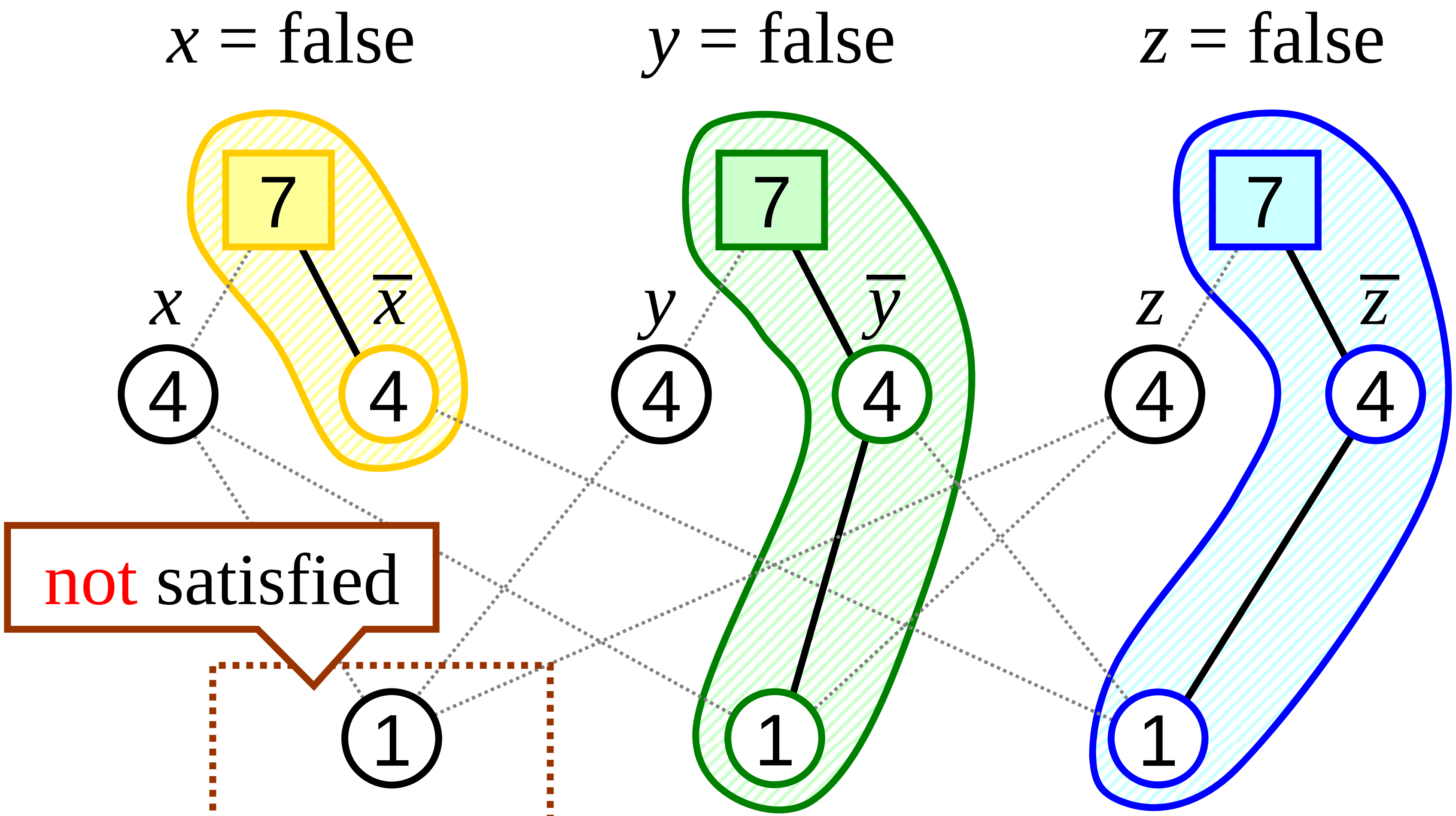
$$f = (x \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

(1) MAXSNP-hardness



$$f = (x \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$

(1) MAXSNP-hardness



$$f = \left(x \vee y \vee z\right) \wedge \left(x \vee \overline{y} \vee z\right) \wedge \left(\overline{x} \vee \overline{y} \vee \overline{z}\right)$$

(1) MAXSNP-hardness

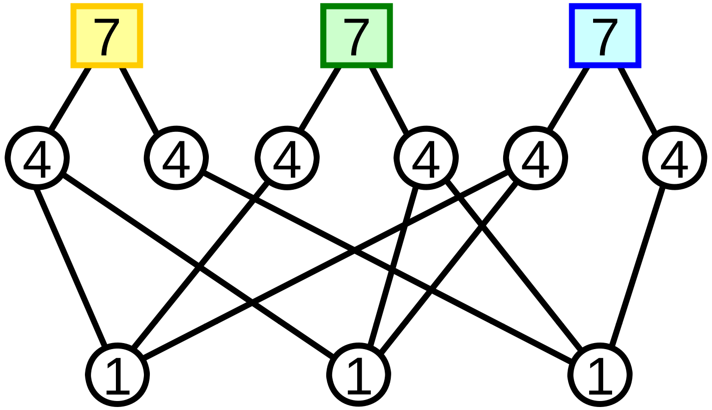
Max PP is MAXSNP-hard for bipartite graphs.

L-reduction: preserves approximability

3-occurrence MAX3SAT → (L-reduction) → Max PP for bipartite graphs

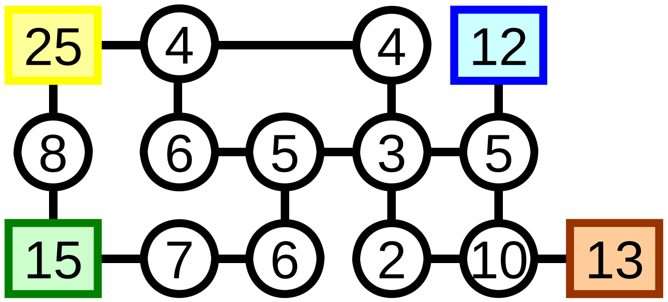
each variable appears
at most 3 times

$$f = (x \vee y \vee z) \wedge (x \vee \overline{y} \vee z) \wedge (\overline{x} \vee \overline{y} \vee \overline{z})$$



Our Results (approximabiliy)

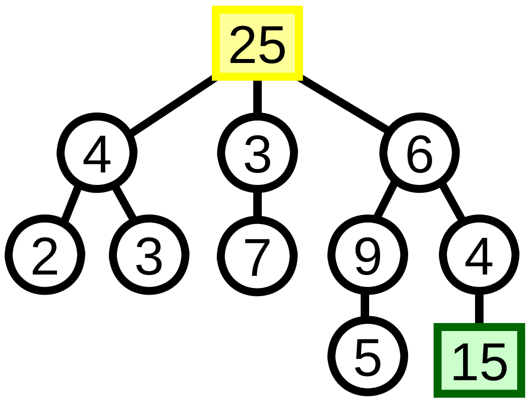
General graphs



(1) MAXSNP-hard
(APX-hard)

No PTAS unless P=NP

Trees



NP-hard

(2) FPTAS

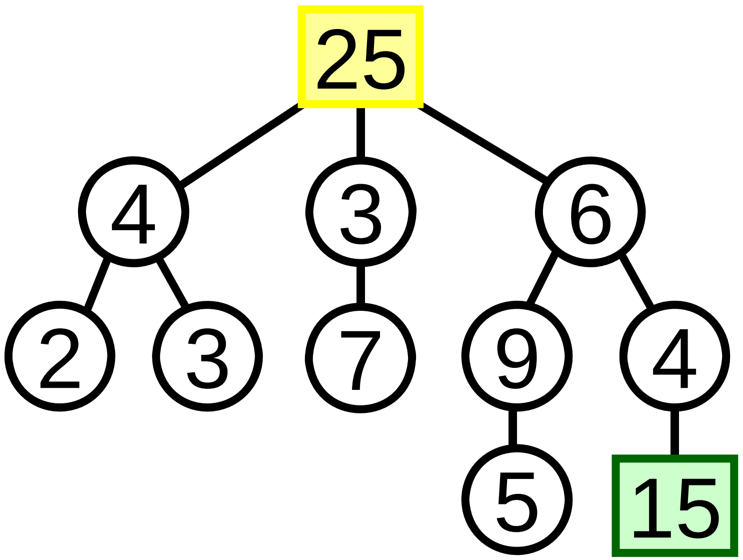
Pseudo-poly.-time algorithm

Pseudo-Polynomial-Time Algorithm

Max PP is NP-hard even for trees.

Max PP can be solved for a tree T in time $O(F^2 n)$ if the supplies and demands are integers.

$F = \min\begin{cases} \bullet \text{ sum of all demands} \\ \bullet \text{ sum of all supplies} \end{cases}$



○ $2+3+4+\cdots+4+5 = 43$

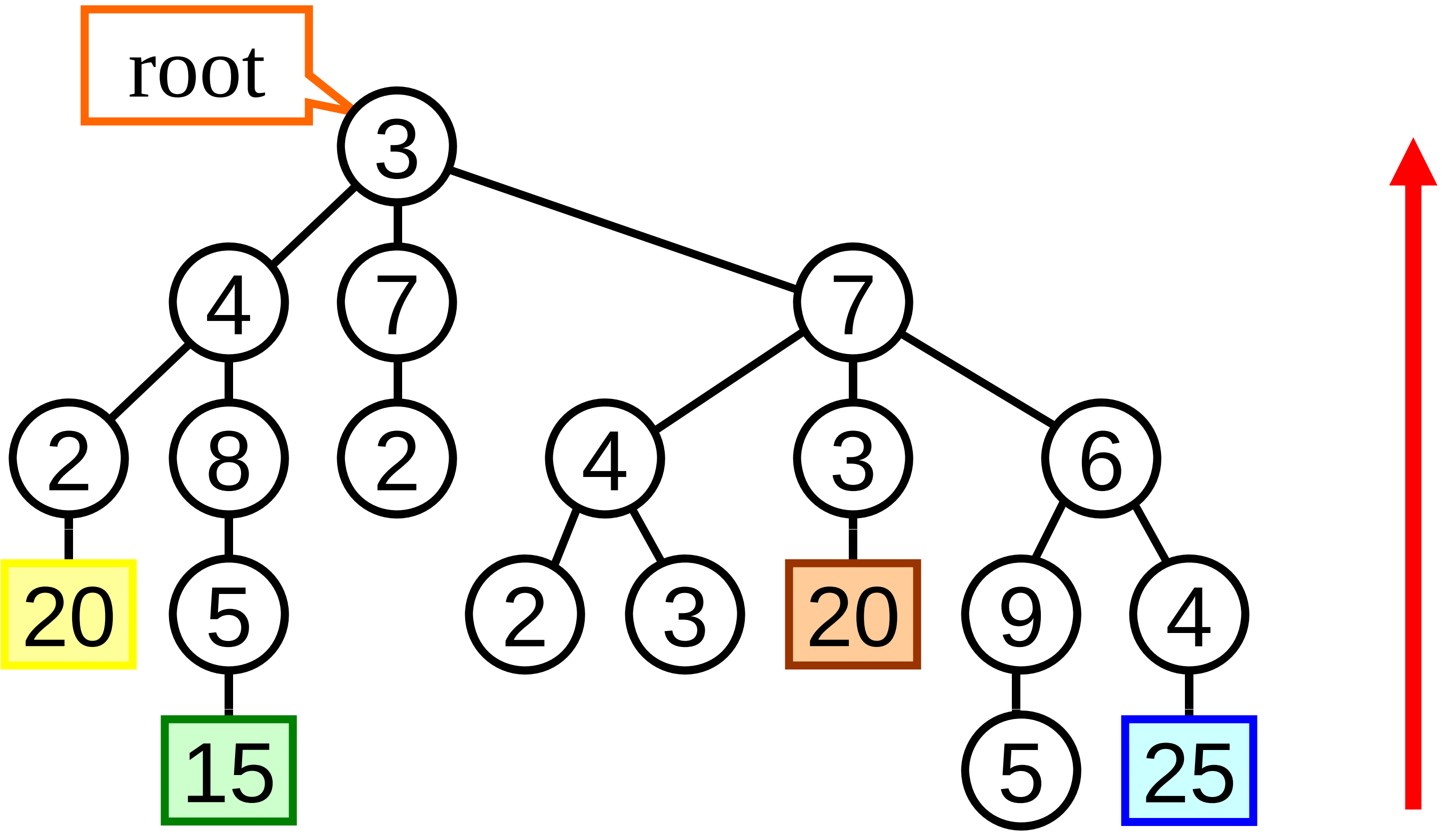
□ $15+25 = 40$

$F = \min\{43, 40\} = 40$

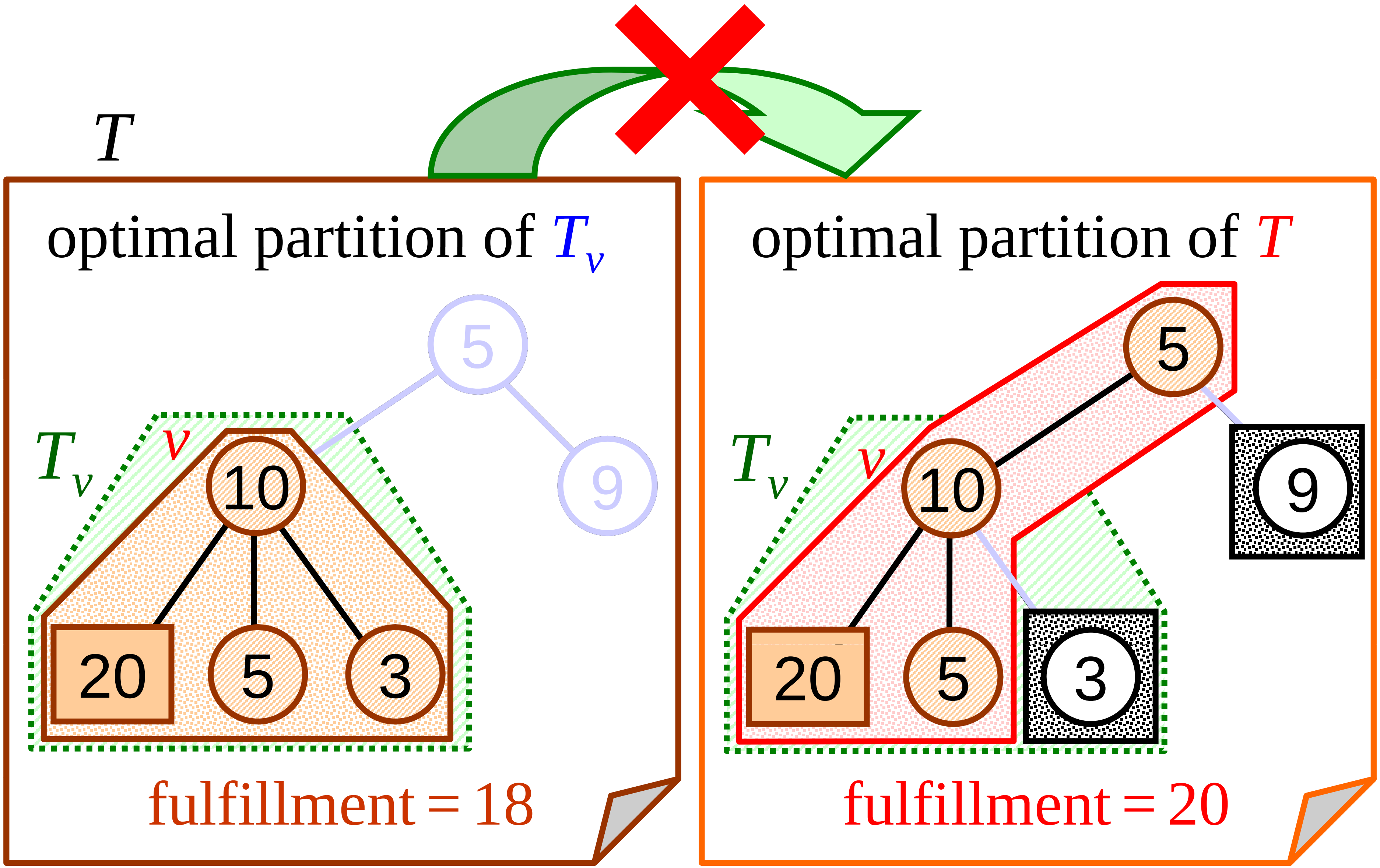
max fulfillment $\leq F$

Pseudo-Polynomial-Time Algorithm

Dynamic Programming

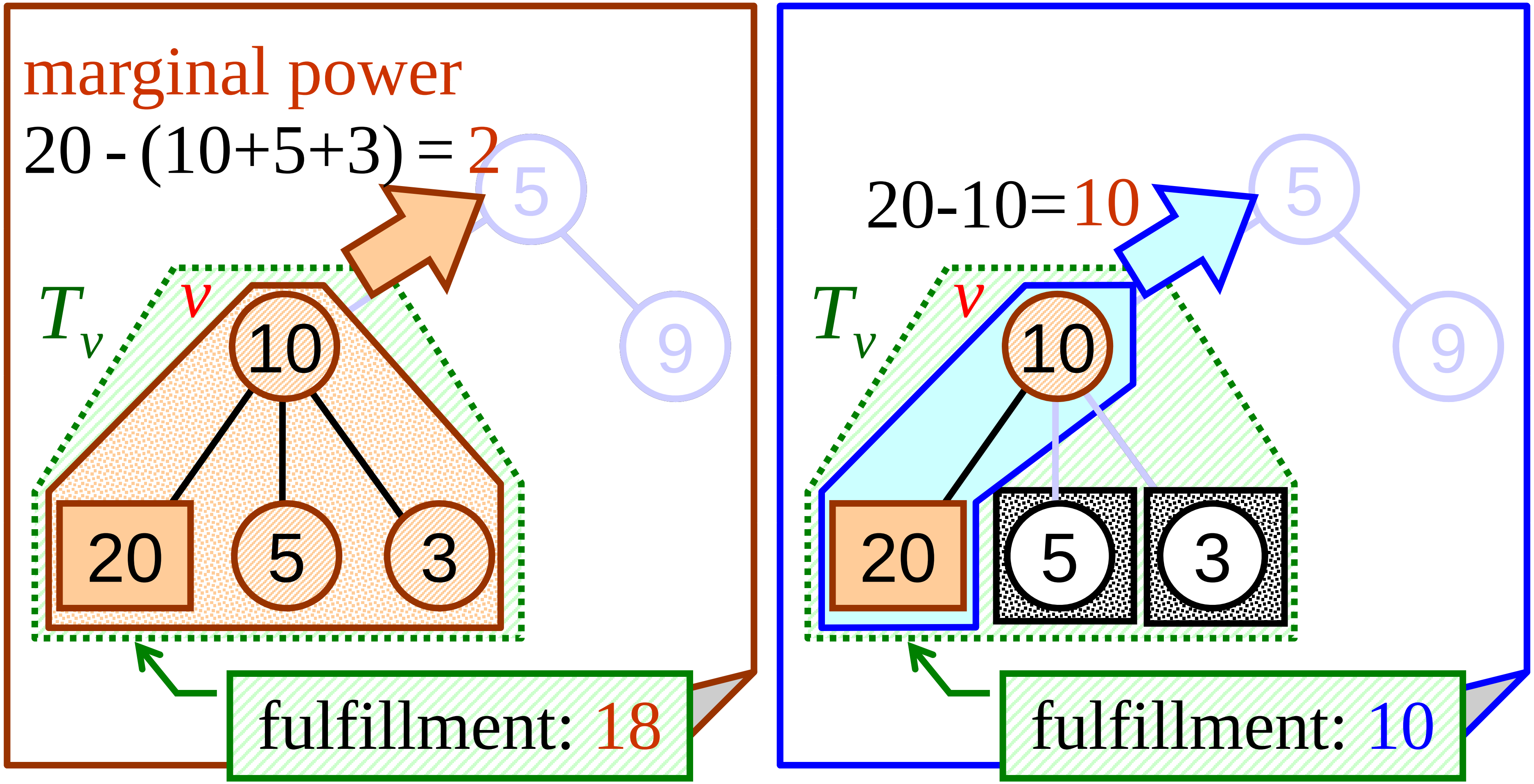


Pseudo-Polynomial-Time Algorithm



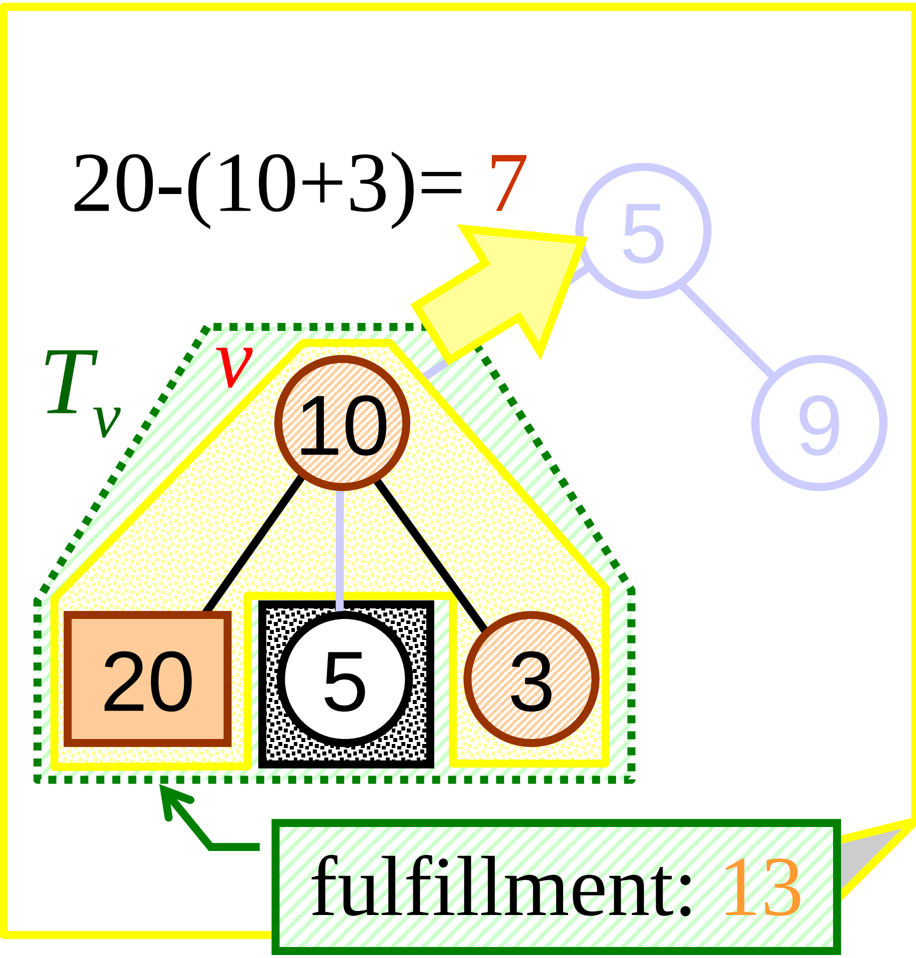
Pseudo-Polynomial-Time Algorithm

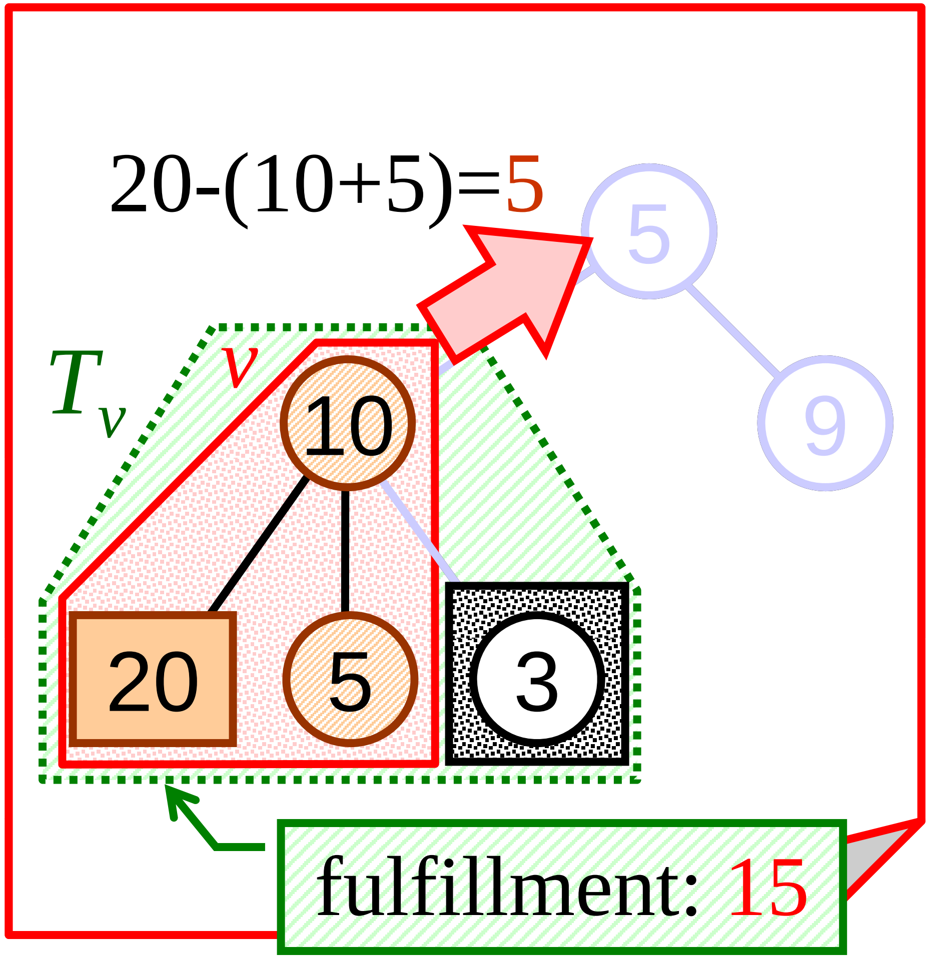
Dynamic Programming



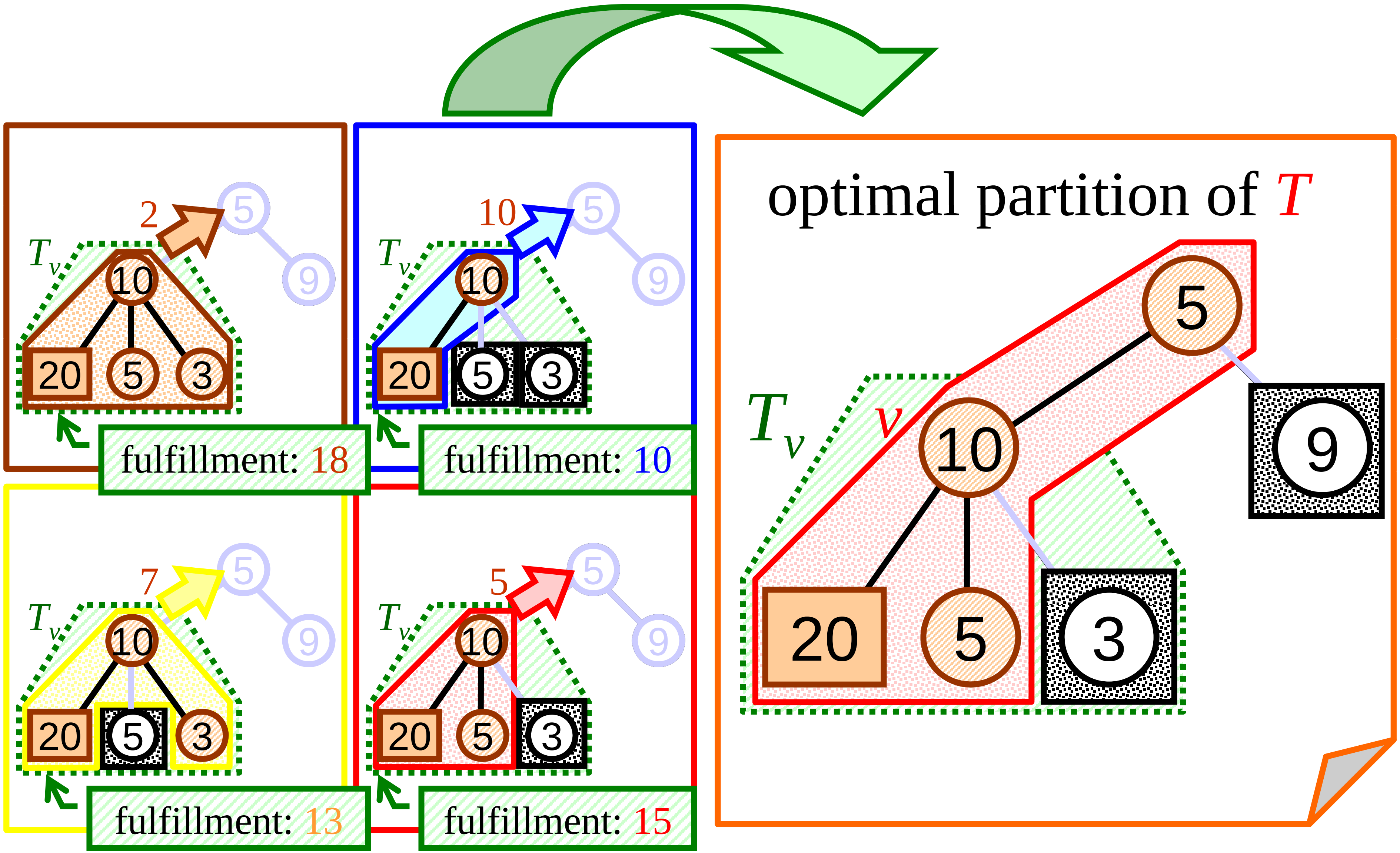
Pseudo-Polynomial-Time Algorithm

Dynamic Programming

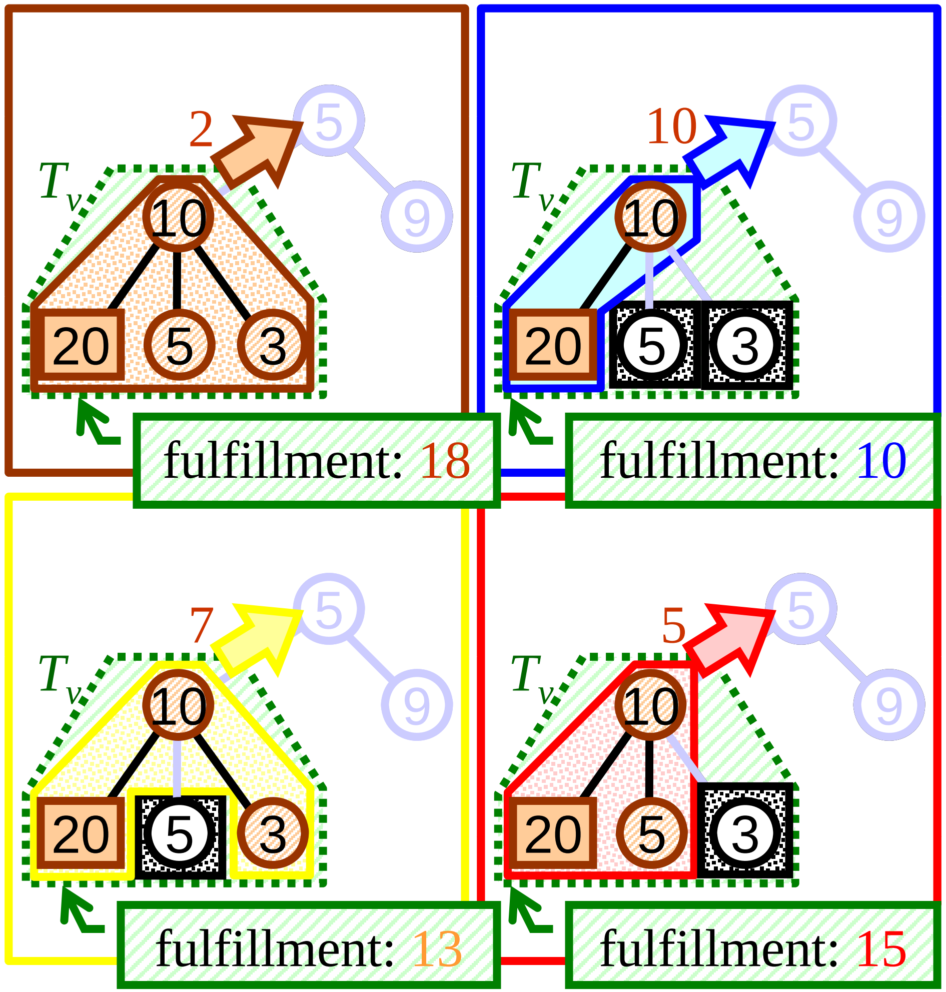




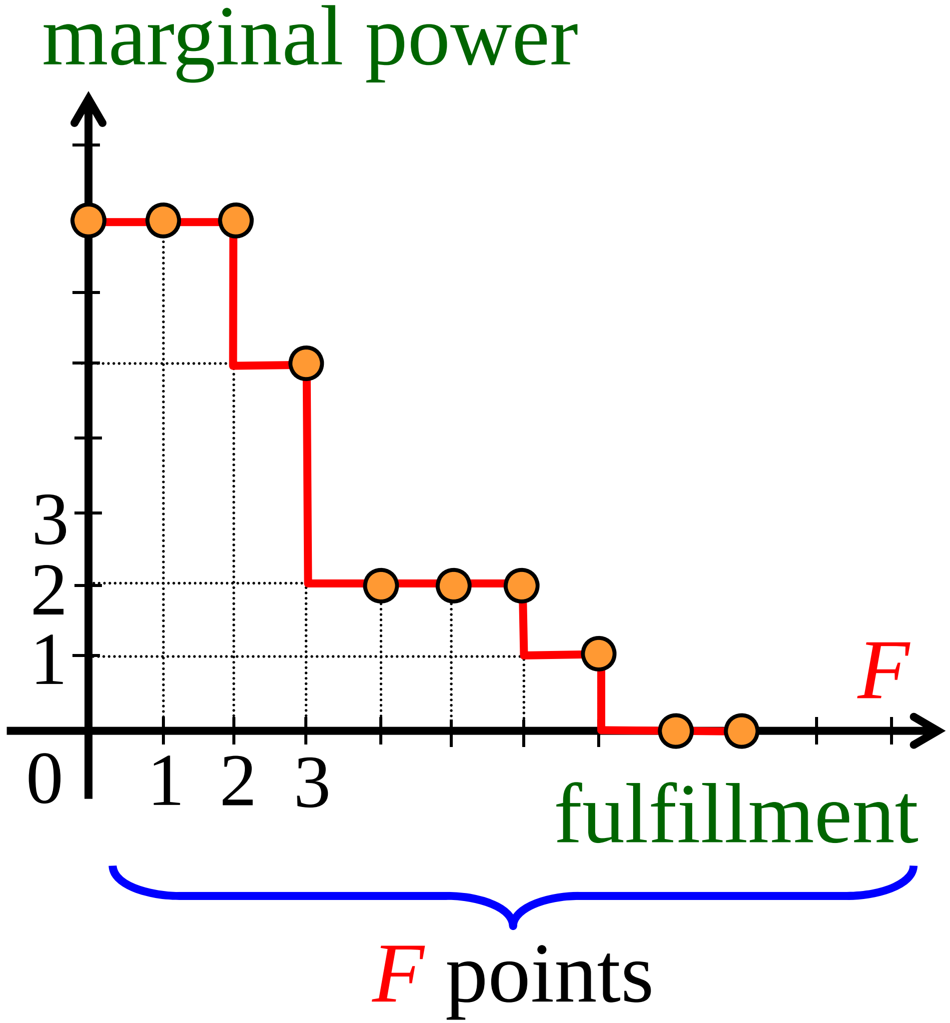
Pseudo-Polynomial-Time Algorithm



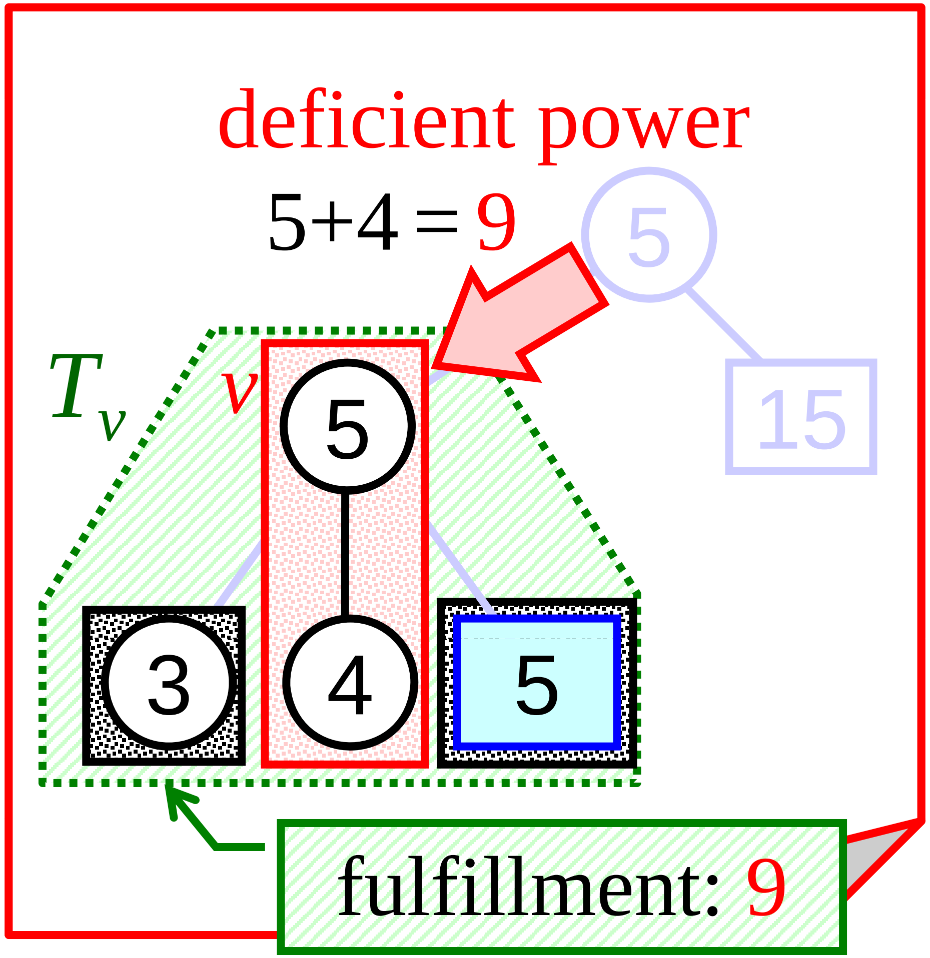
Pseudo-Polynomial-Time Algorithm

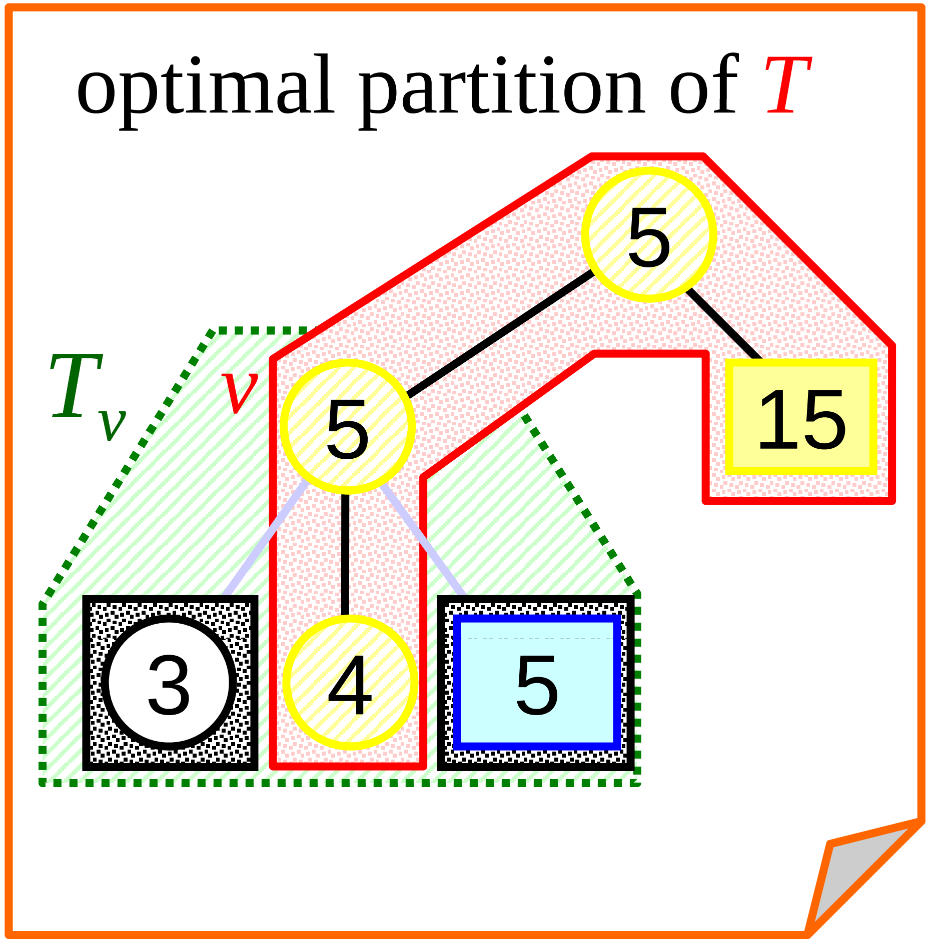


staircase, non-increasing

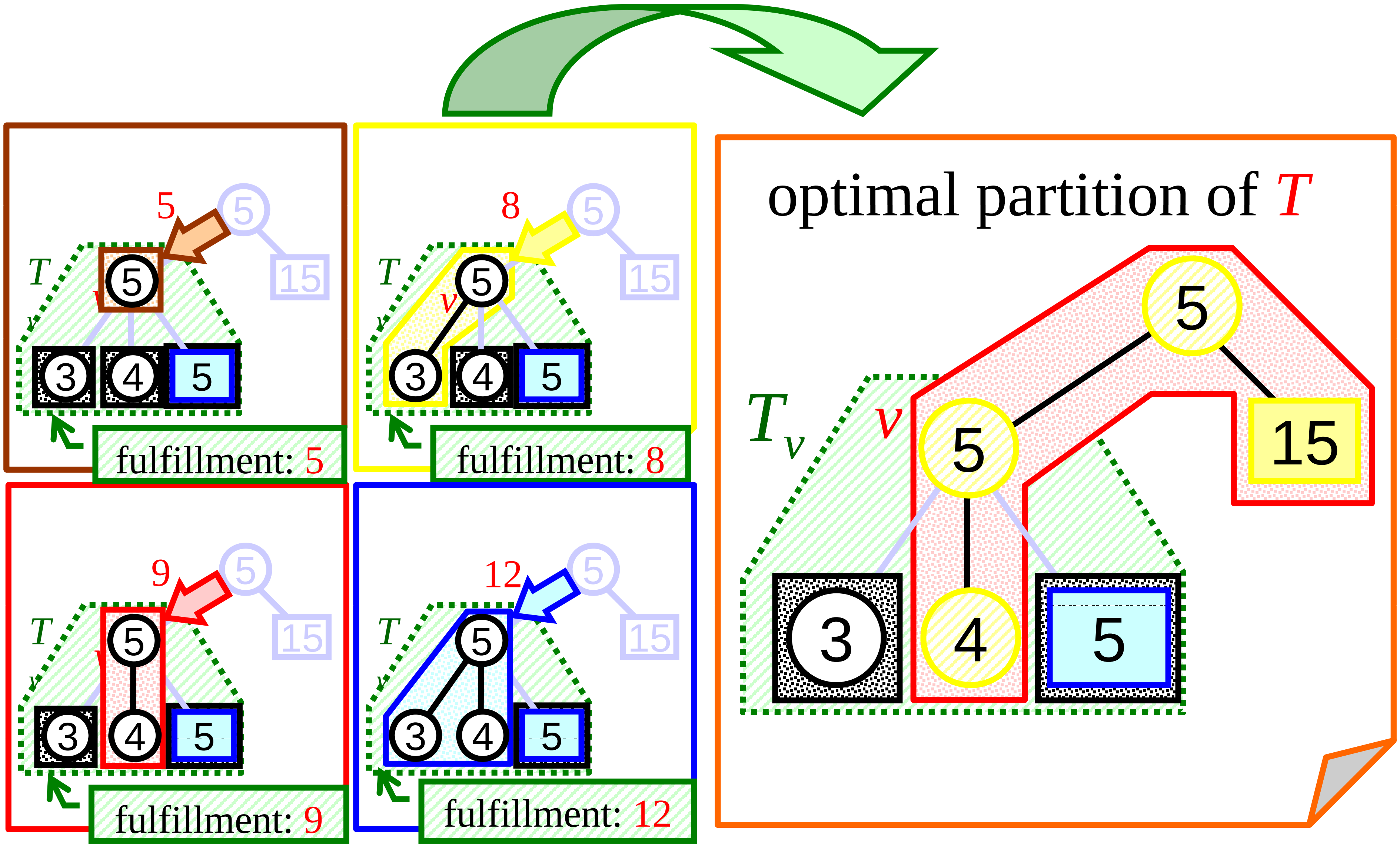


Pseudo-Polynomial-Time Algorithm

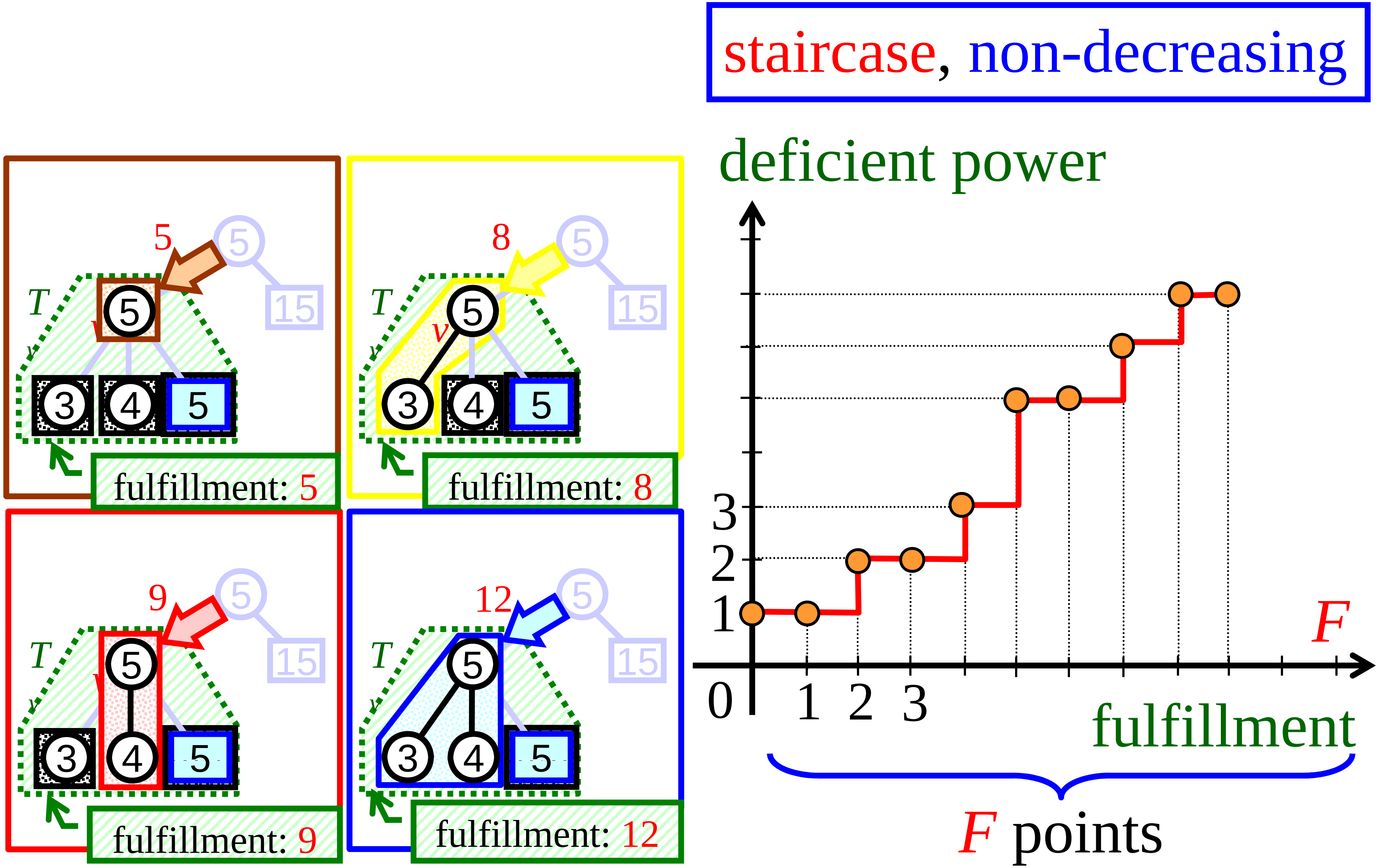




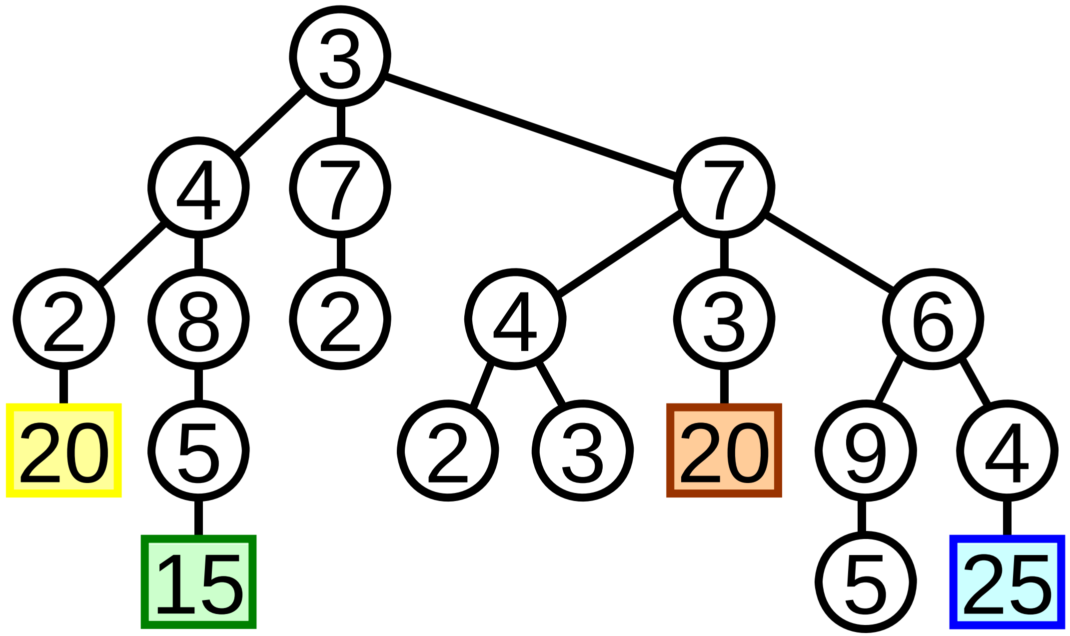
Pseudo-Polynomial-Time Algorithm



Pseudo-Polynomial-Time Algorithm

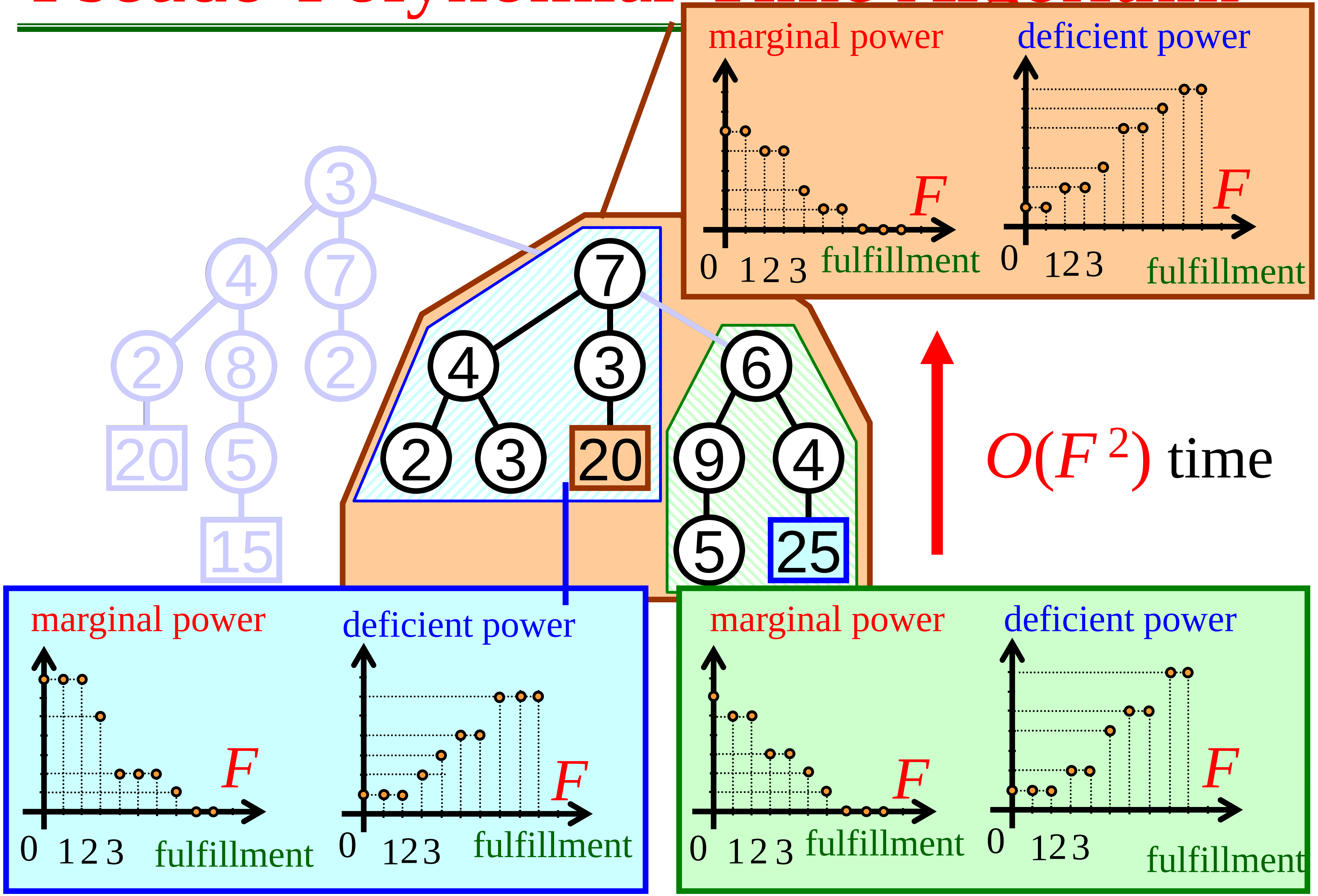


Pseudo-Polynomial-Time Algorithm

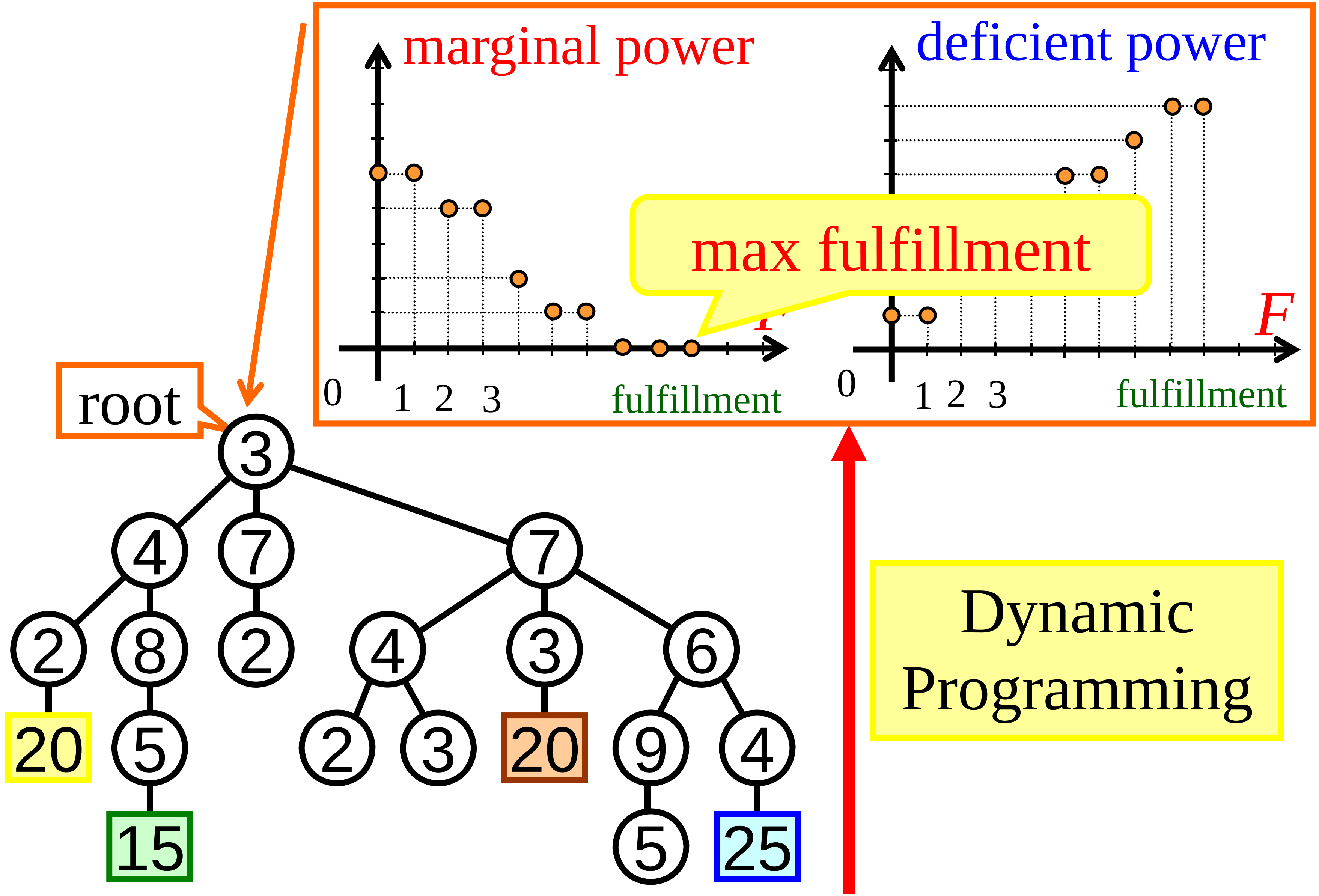


leaves

Pseudo-Polynomial-Time Algorithm



Pseudo-Polynomial-Time Algorithm

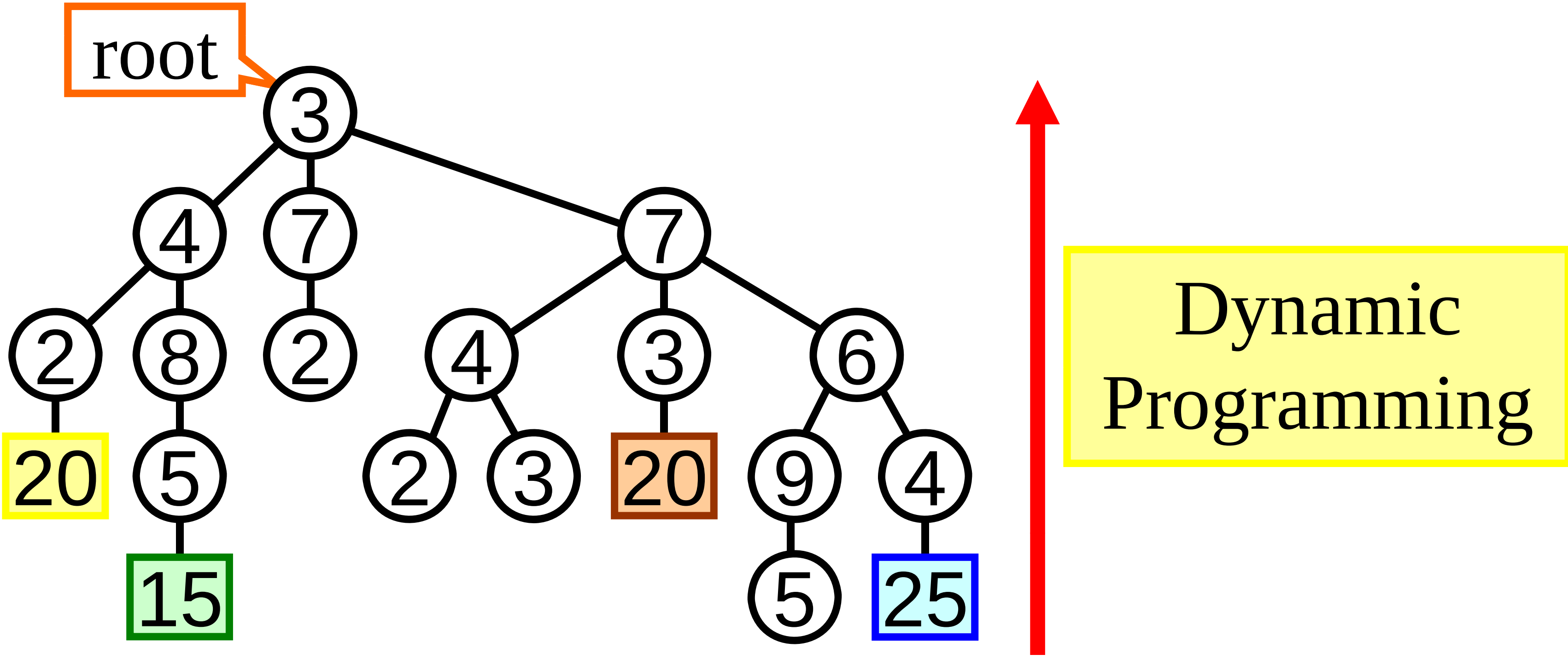


Pseudo-Polynomial-Time Algorithm

Computation time

for each vertex ················ $O(F^2)$

There are n vertices.



Pseudo-Polynomial-Time Algorithm

Computation time

for each vertex ················ $O(F^2)$

There are n vertices.

Computation time

$O(F^2 n)$

The algorithm takes polynomial time if F is bounded by a polynomial in n.

(2) FPTAS

Let all demands and supply be positive real numbers.

For any ε, $0 < \varepsilon < 1$, the algorithm finds a partition of a tree T such that

$$\text{OPT} - \text{APPRO} < \varepsilon\,\text{OPT}$$

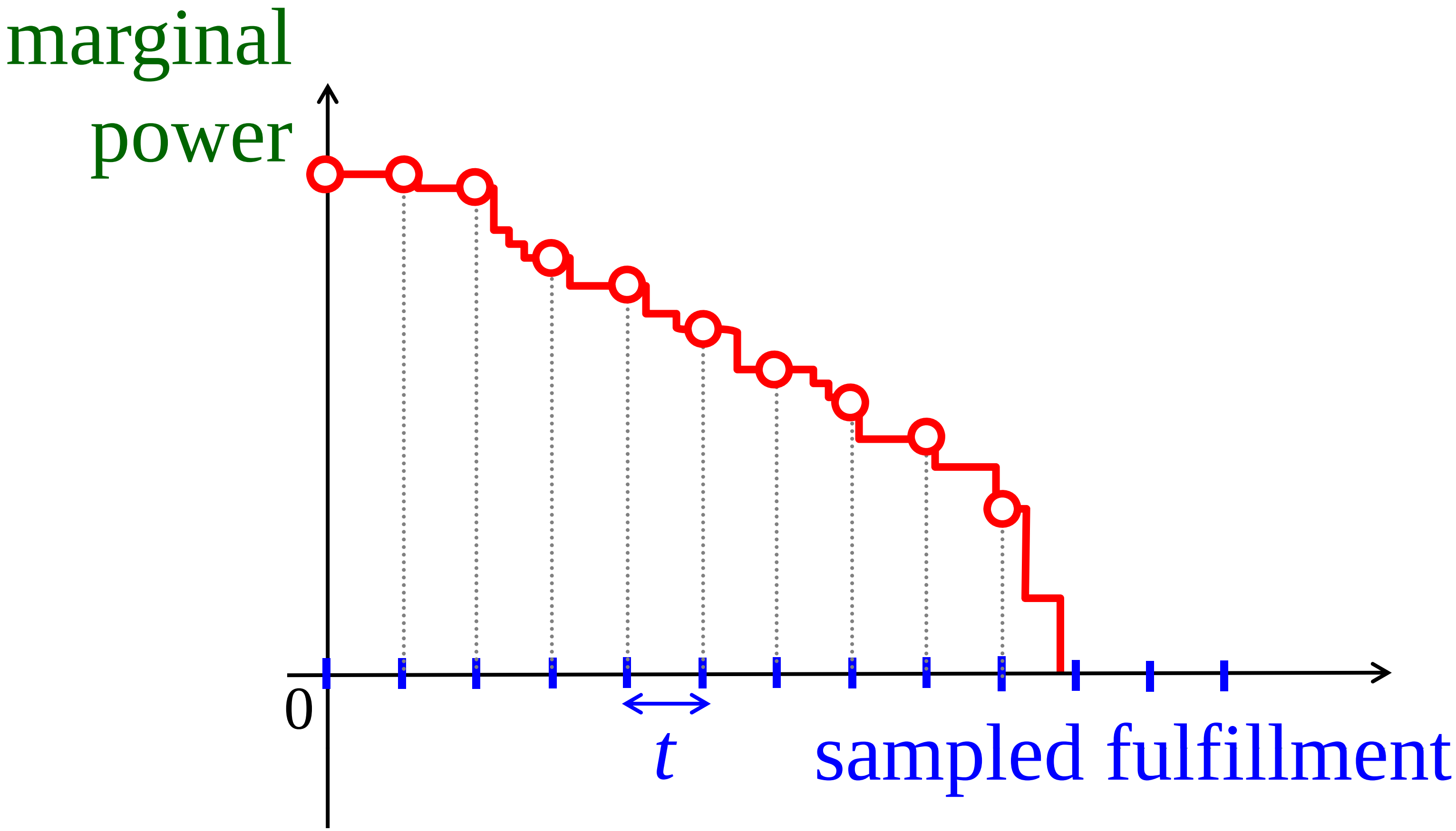
in time polynomial in both n and $1/\varepsilon$.

$$O\left(\frac{n^5}{\varepsilon^2}\right)$$

n : # of vertices

(2) FPTAS

The algorithm is similar to the previous algorithm.

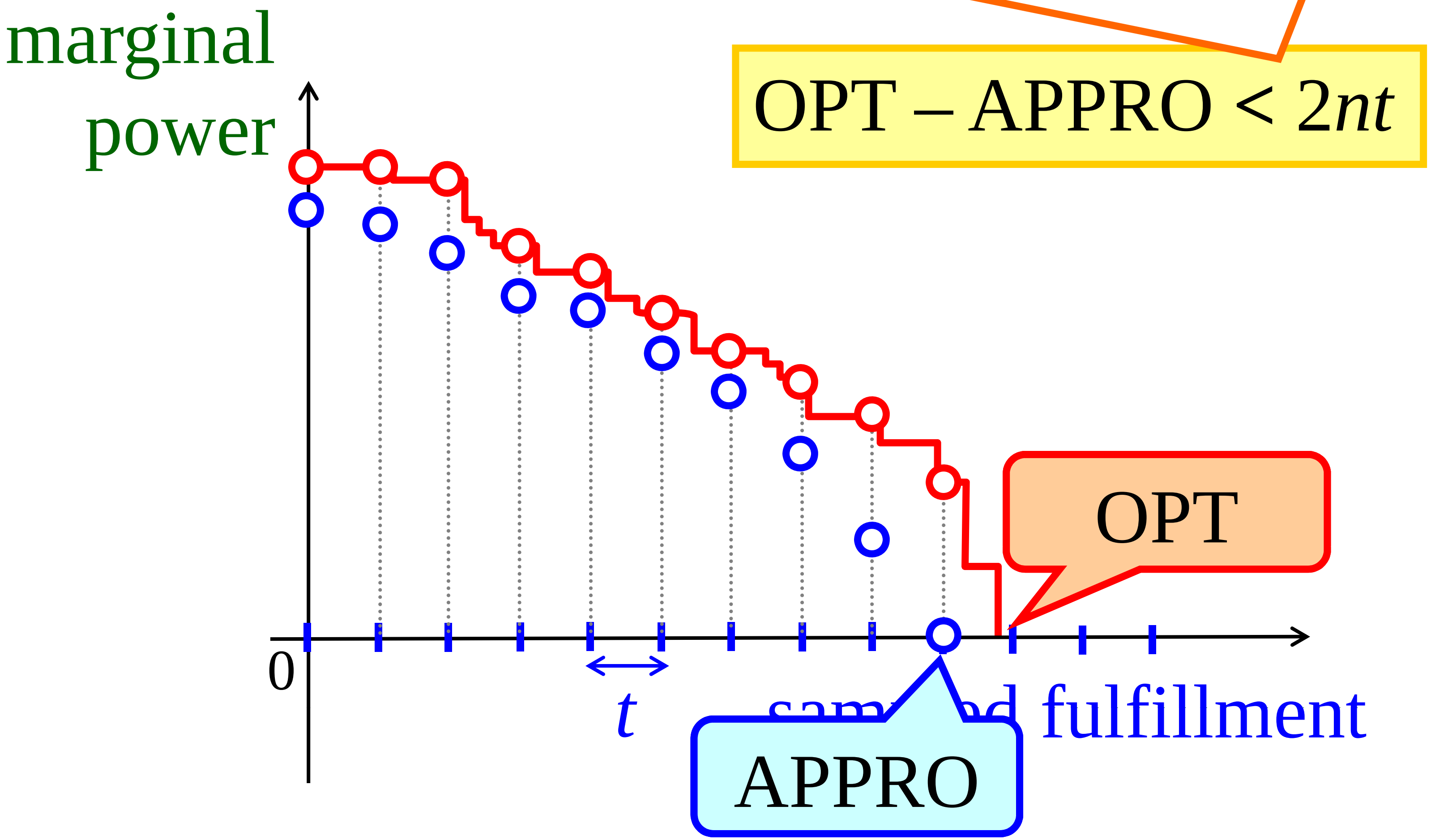


(2) FPTAS

The alg

total error < error/merge × # of merge

< $2t$ × n



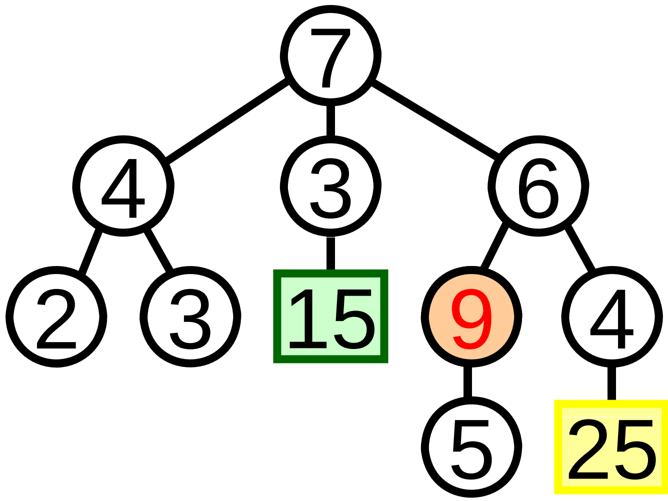
(2) FPTAS

Error

m_d: max demand

$$\text{OPT} - \text{APPRO} < 2nt$$

$$t = \frac{\varepsilon m_d}{2n}$$



$m_d = 9$

(2) FPTAS

Error

m_d: max demand

$$\mathrm{OPT} - \mathrm{APPRO} < 2nt$$

$$t = \frac{\varepsilon m_d}{2n}$$

$$= \varepsilon m_d$$

$$m_d \leqq \mathrm{OPT}$$

$$\leqq \varepsilon \mathrm{OPT}$$

$$\mathrm{OPT} - \mathrm{APPRO} < \varepsilon \mathrm{OPT}$$

error ratio $$\frac{\mathrm{OPT} - \mathrm{APPRO}}{\mathrm{OPT}} < \varepsilon$$

(2) FPTAS

Error

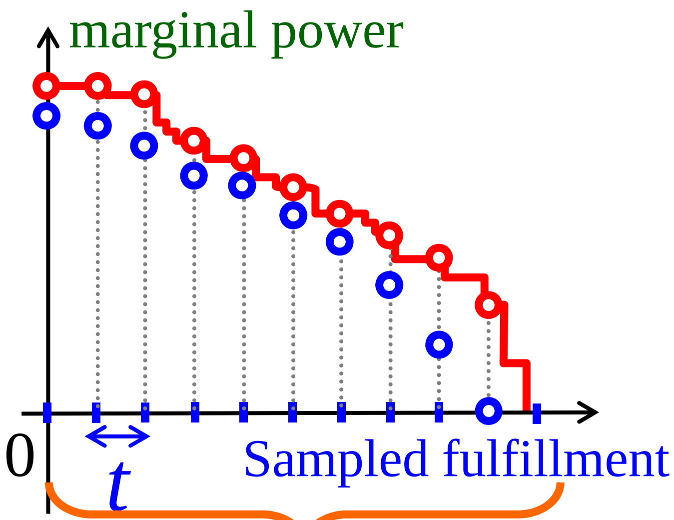
m_d: max demand

$$\text{OPT} - \text{APPRO} < \varepsilon\, \text{OPT}$$

Computation time

$$O\left(\left(\frac{F}{t}\right)^2 n\right) = O\left(\frac{n^5}{\varepsilon^2}\right)$$

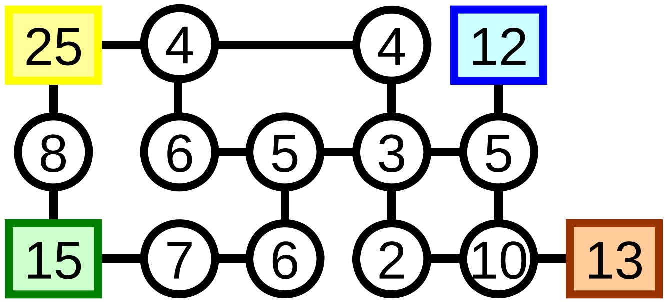
$$t = \frac{\varepsilon\, m_d}{2n}, \quad F \le n m_d$$



$\frac{F}{t}$ points

Conclusions

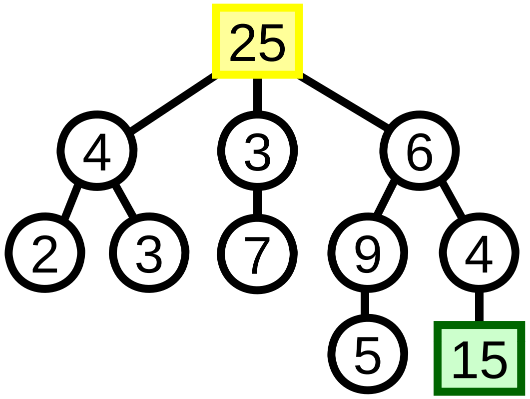
General graphs



(1) MAXSNP-hard
(APX-hard)

No PTAS unless P=NP

Trees

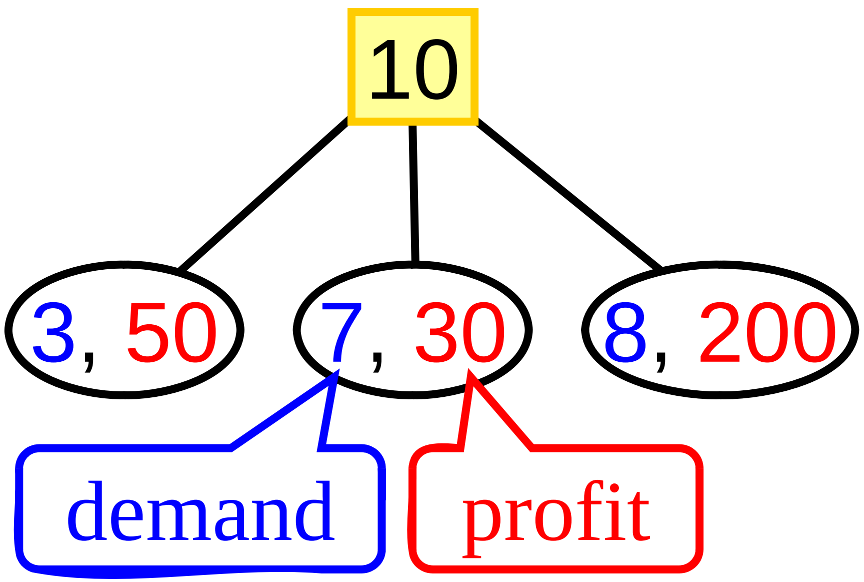


NP-hard

(2) FPTAS

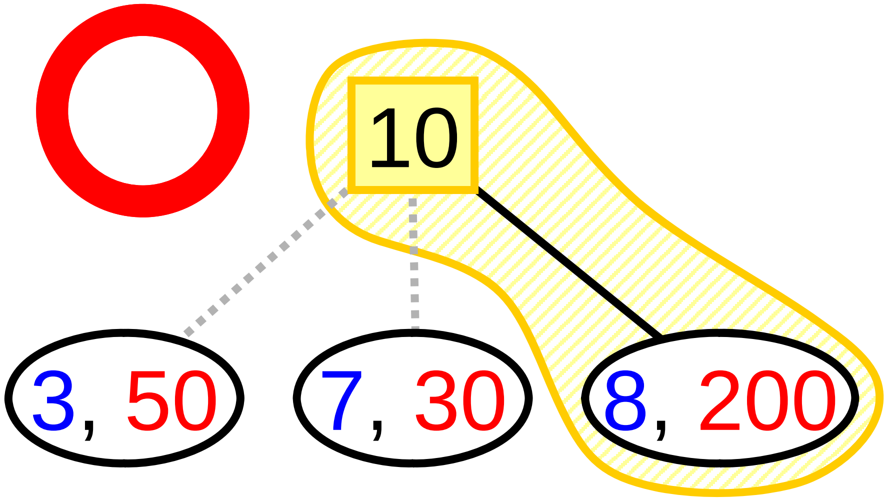
Variant (Generalization of Knapsack Problem)

find a partition which maximizes sum of profits of all demand vertices that are supplied power.



Variant (Generalization of Knapsack Problem)

find a partition which maximizes sum of profits of all demand vertices that are supplied power.

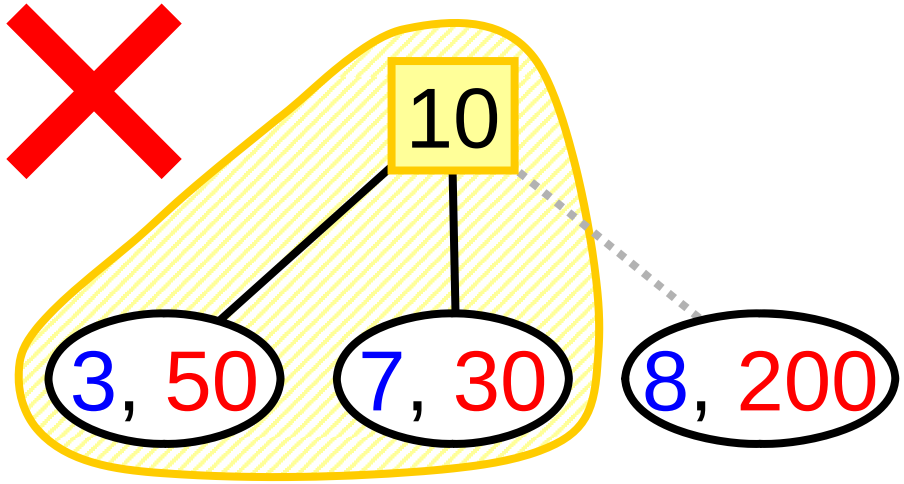


Sum

demands = 8

profits = 200

max



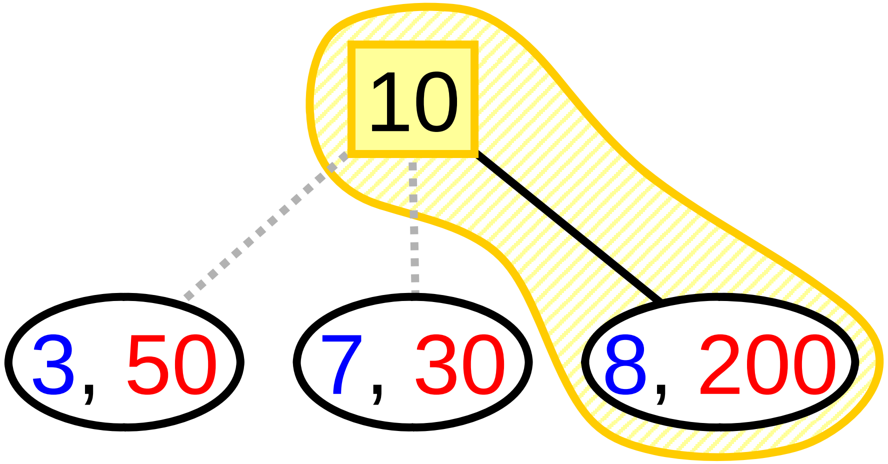
Sum

demands = 3+7 = 10

profits = 50+30 = 80

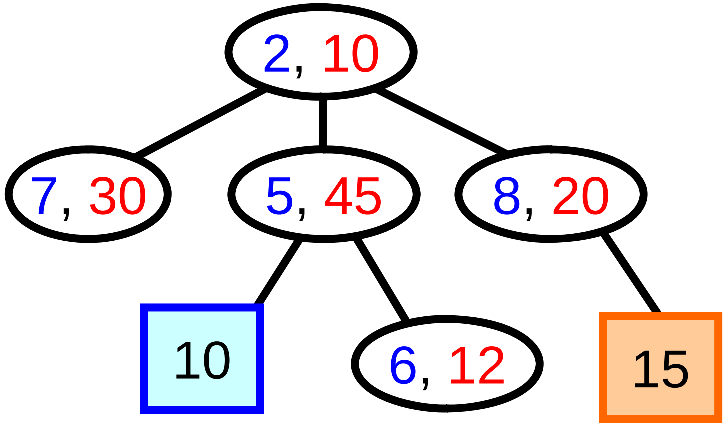
Variant (Generalization of Knapsack Problem)

find a partition which maximizes sum of profits of all demand vertices that are supplied power.



FPTAS for KP

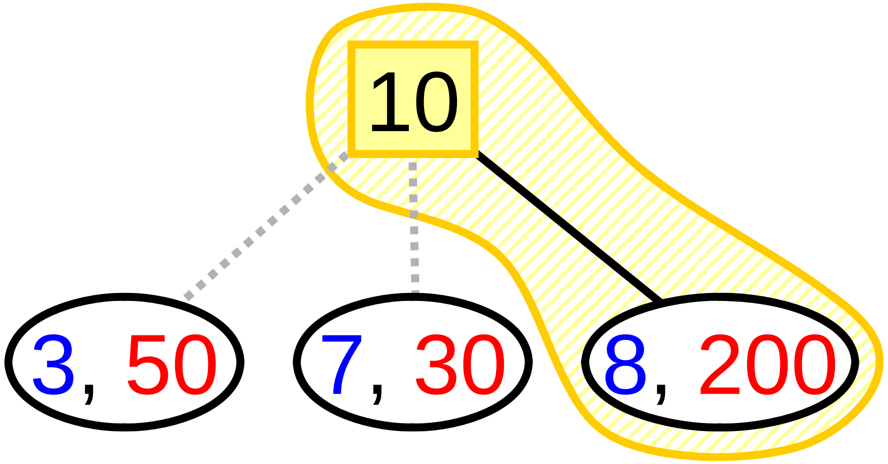
[Ibarra and Kim '75]



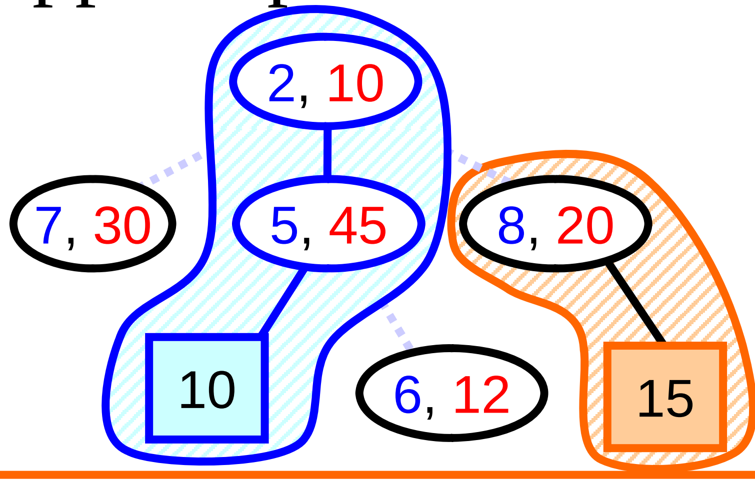
FPTAS for trees

Variant (Generalization of Knapsack Problem)

find a partition which maximizes sum of profits of all demand vertices that are supplied power.

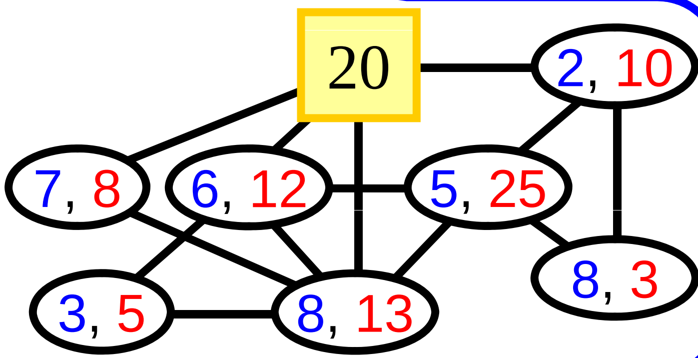


FPTAS for KP



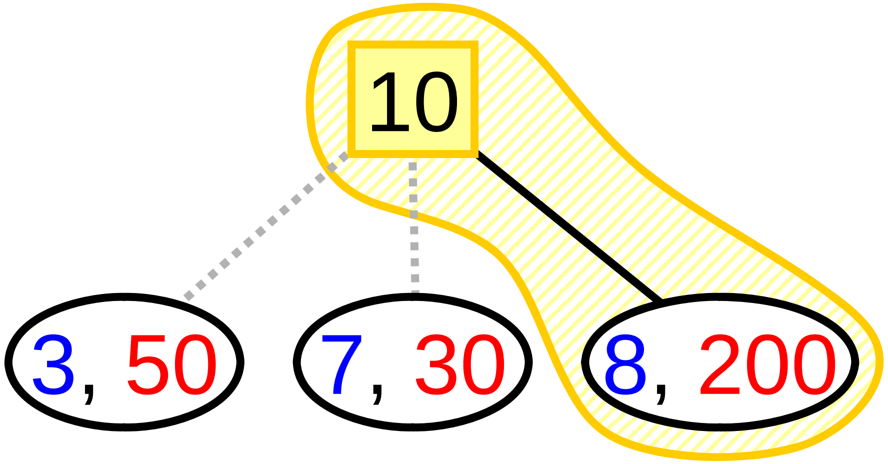
FPTAS for trees

extendable to partial *k*-trees
(graphs with bounded treewidth)
if there is exactly one supply

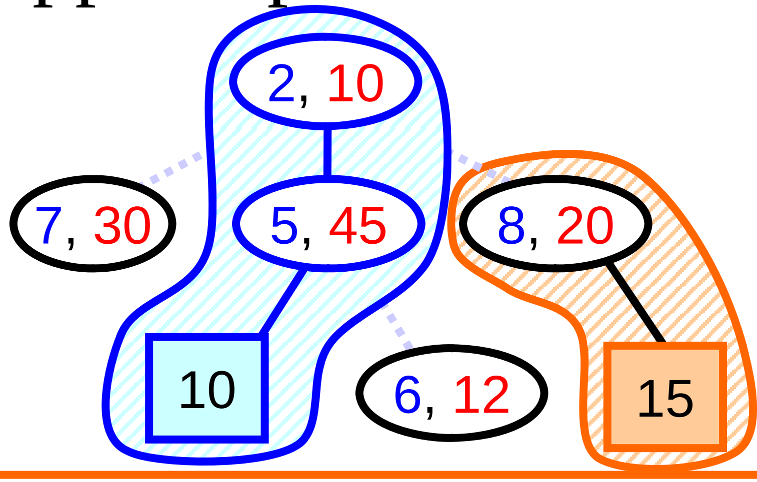


Variant (Generalization of Knapsack Problem)

find a partition which maximizes sum of profits of all demand vertices that are supplied power.

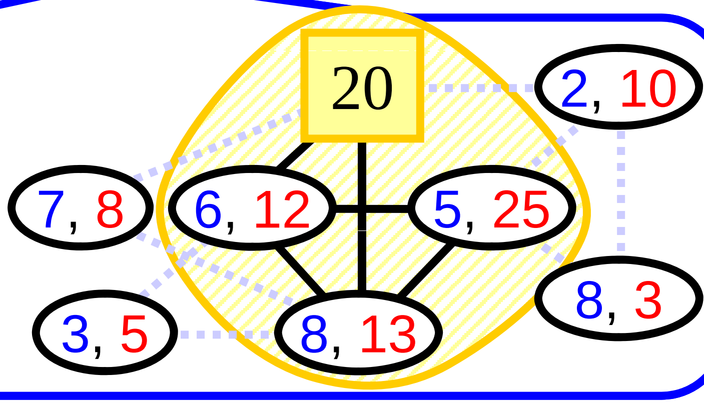


FPTAS for KP



FPTAS for trees

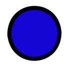
extendable to partial *k*-trees
(graphs with bounded treewidth)
if there is exactly one supply



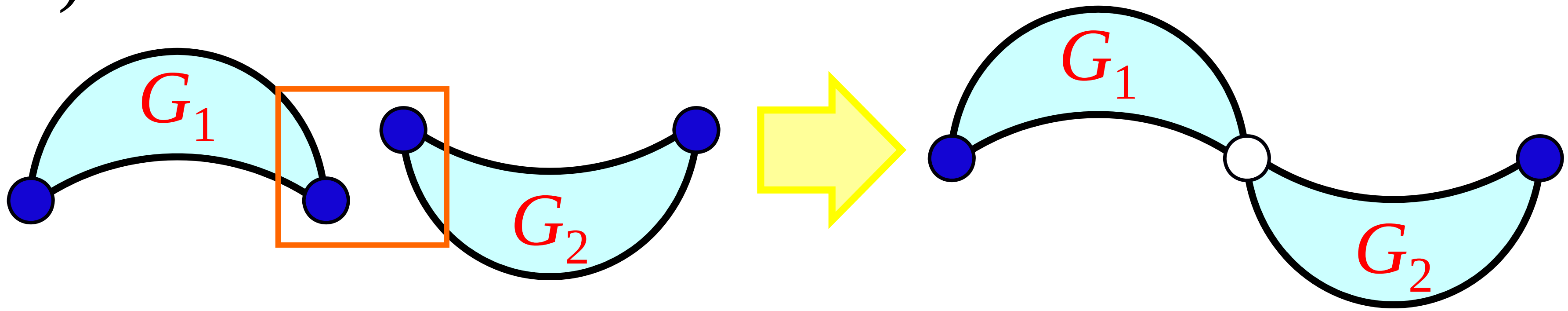
Thank You!

Series-Parallel Graphs (recursively)

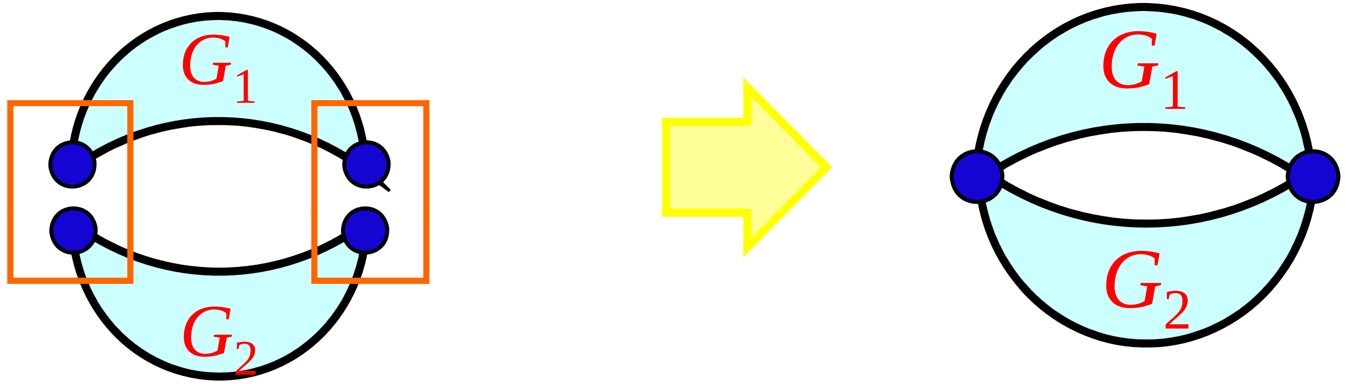
(1) A single edge is a SP graph.



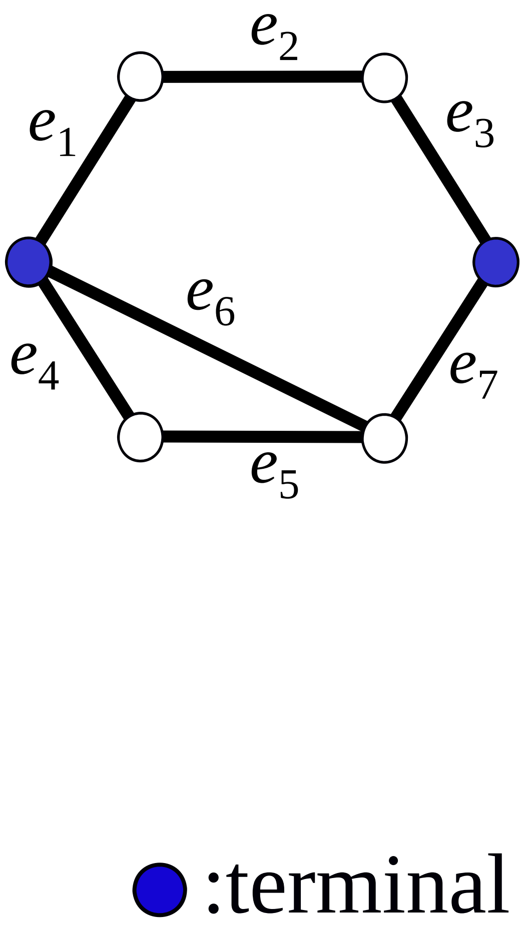
(2a) Series connection

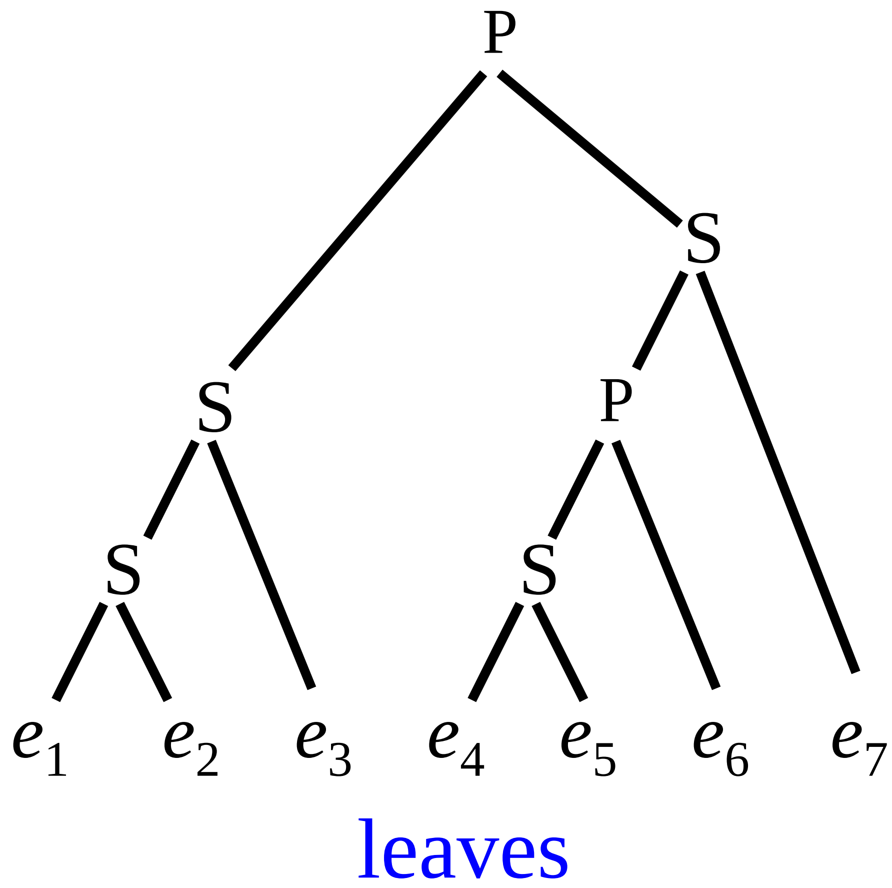


(2b) Parallel connection



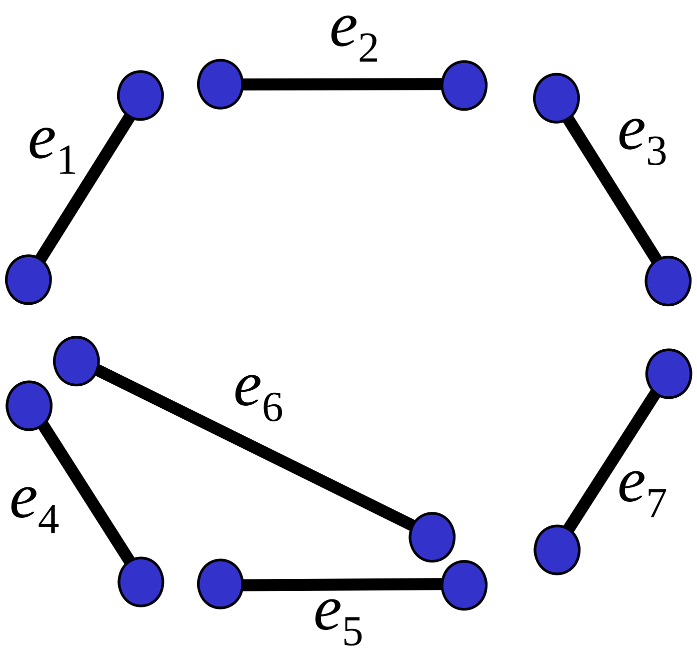
SP Graph & Decomposition Tree





Decomposition tree

SP Graph & Decomposition Tree



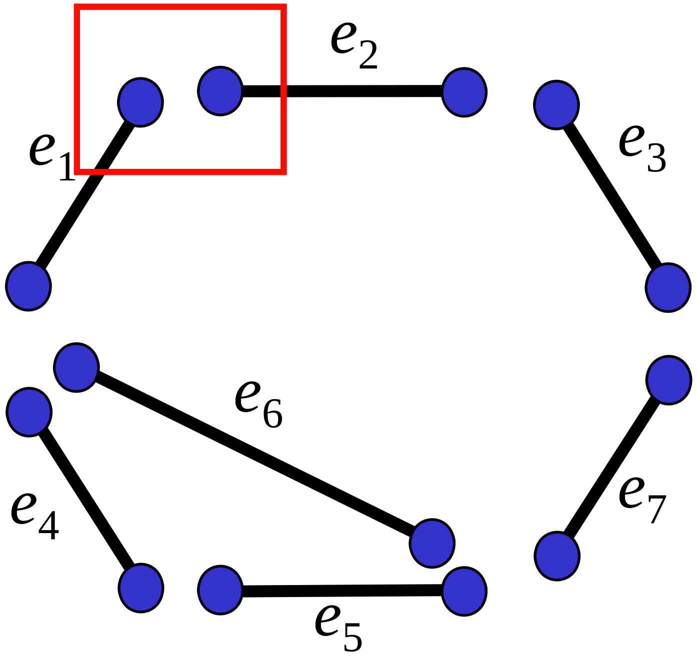
● :terminal

e_1 e_2 e_3 e_4 e_5 e_6 e_7

leaves

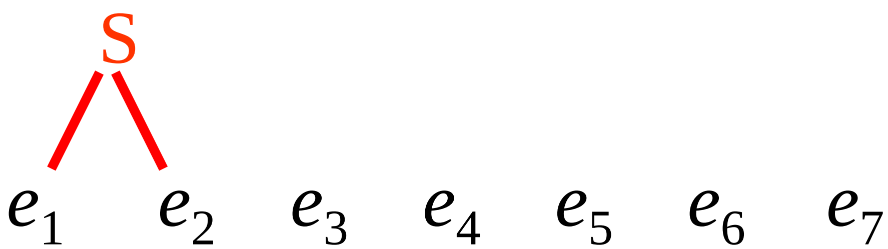
Decomposition tree

SP Graph & Decomposition Tree



Series connection

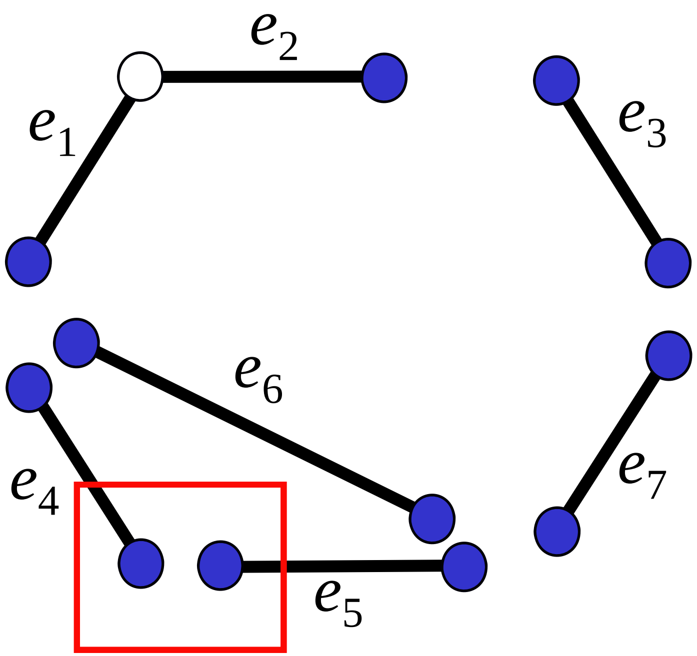
● :terminal



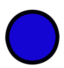
leaves

Decomposition tree

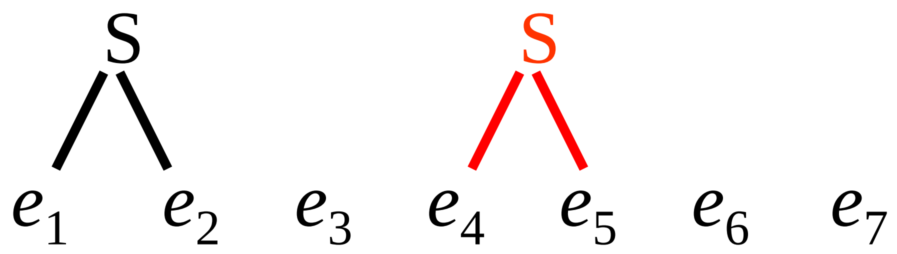
SP Graph & Decomposition Tree



Series connection



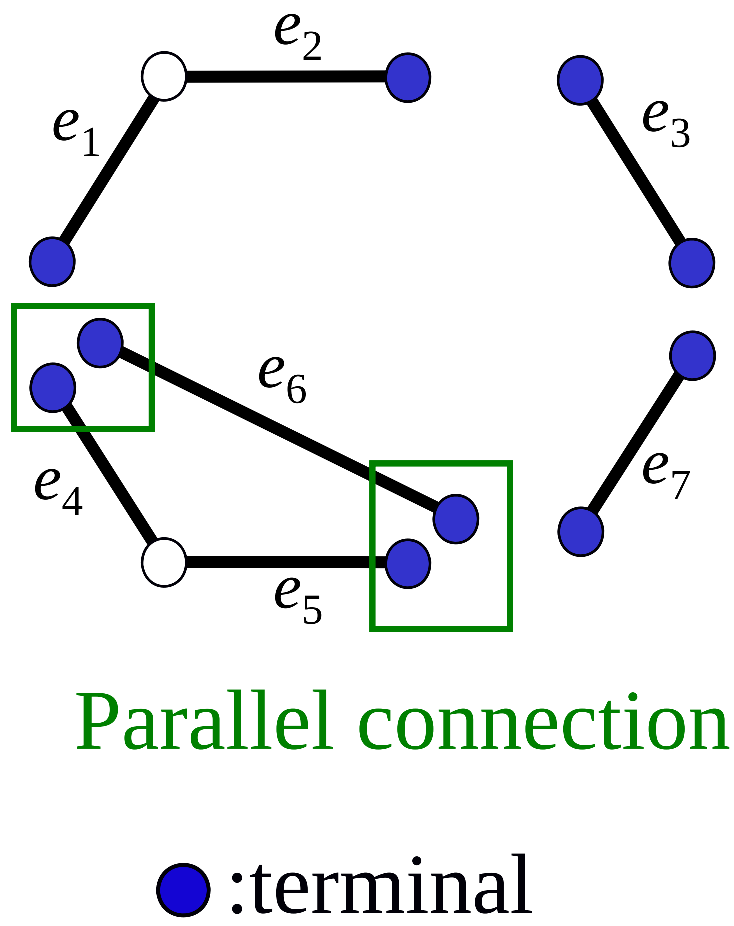
:terminal

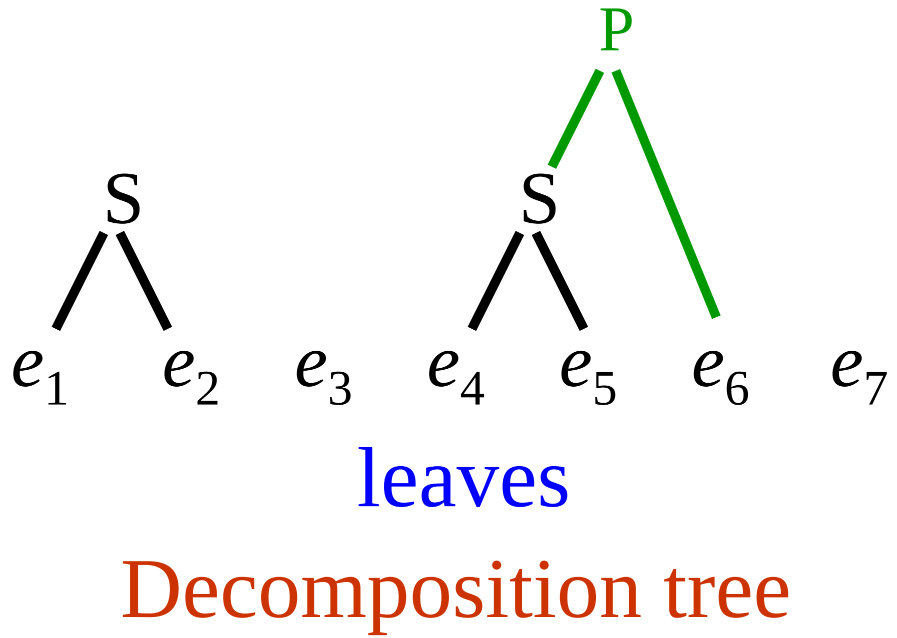


leaves

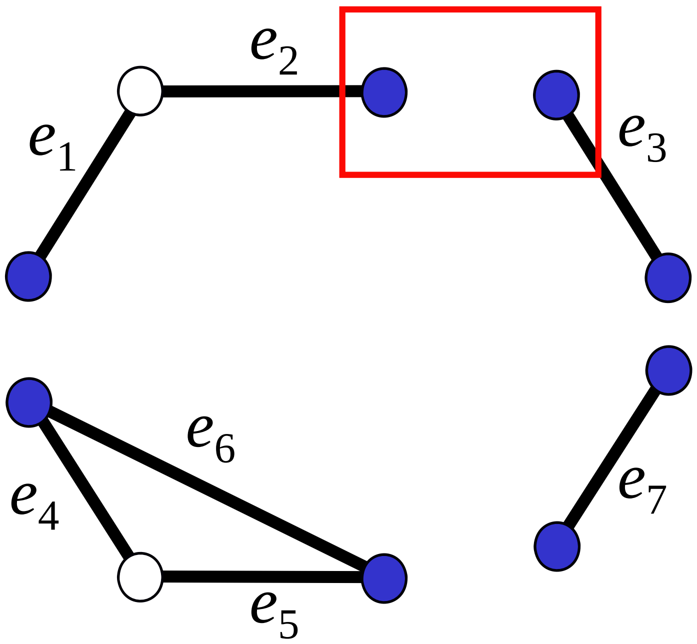
Decomposition tree

SP Graph & Decomposition Tree

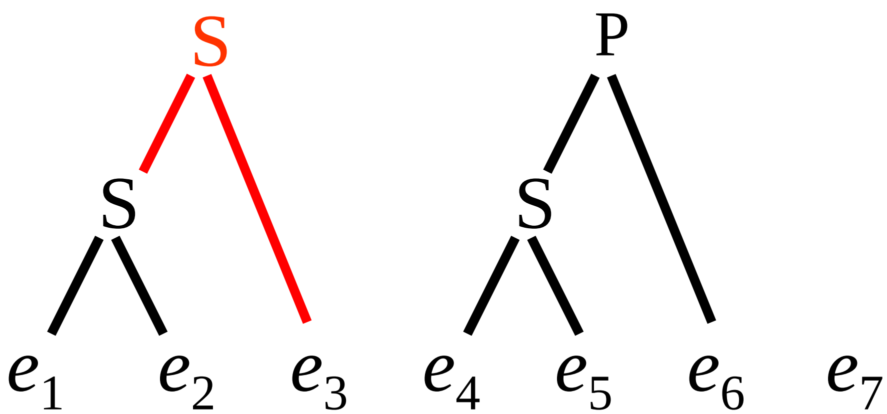




SP Graph & Decomposition Tree



Series connection

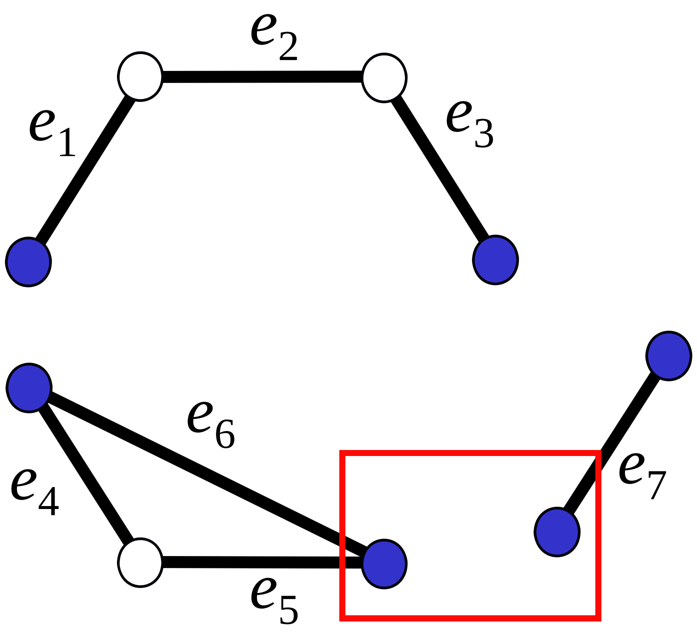


leaves

Decomposition tree

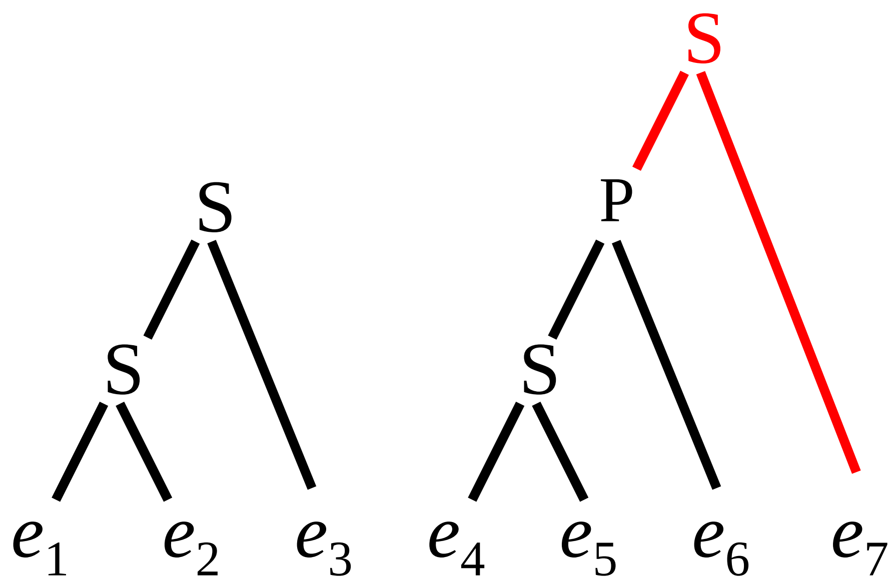
● :terminal

SP Graph & Decomposition Tree



Series connection

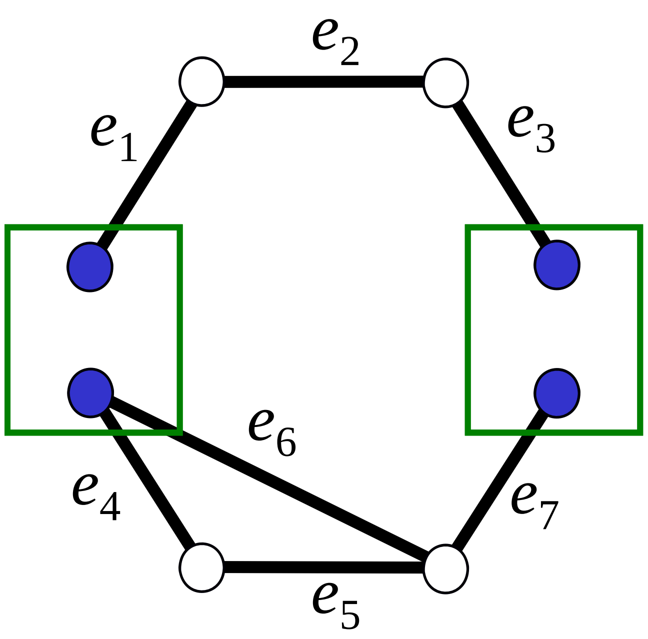
● :terminal



leaves

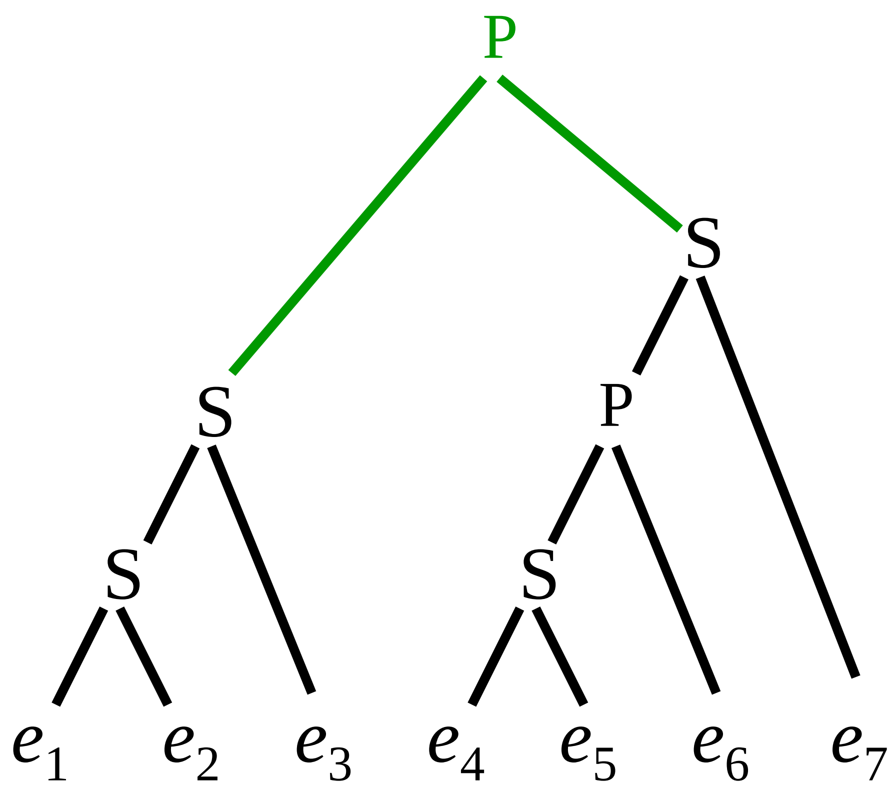
Decomposition tree

SP Graph & Decomposition Tree



Parallel connection

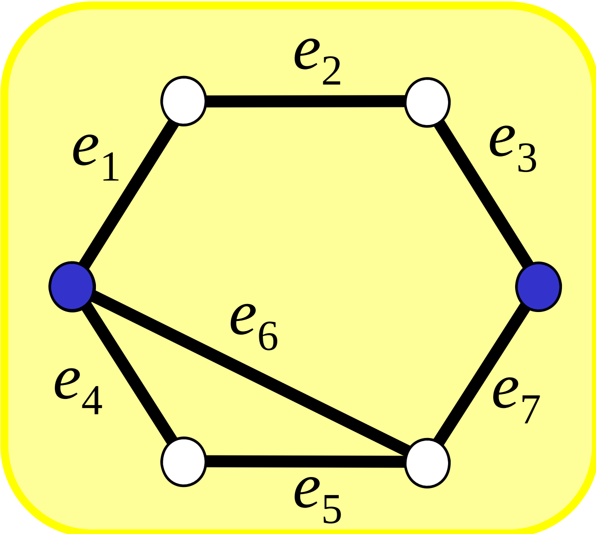
● :terminal

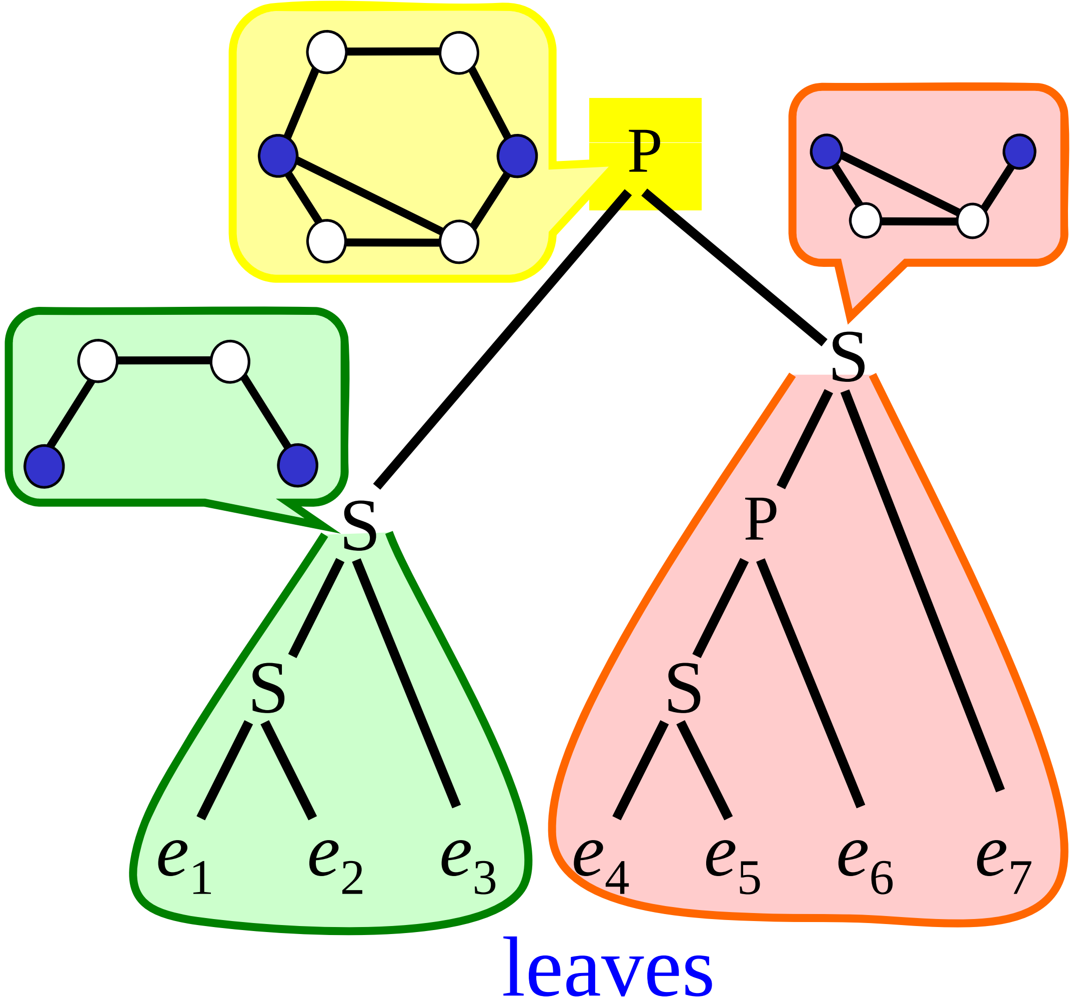


leaves

Decomposition tree

SP Graph & Decomposition Tree

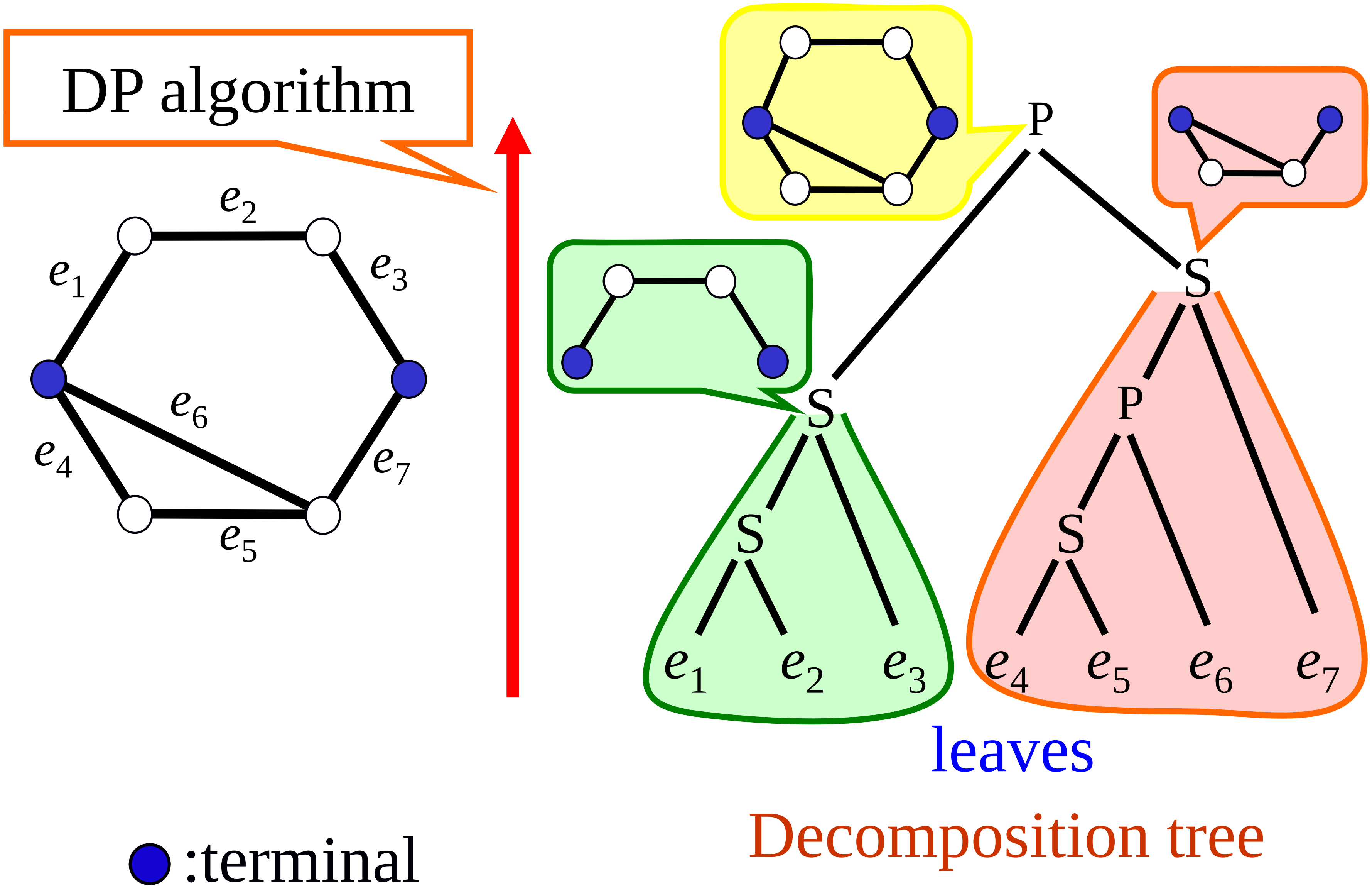




Decomposition tree

● :terminal

SP Graph & Decomposition Tree



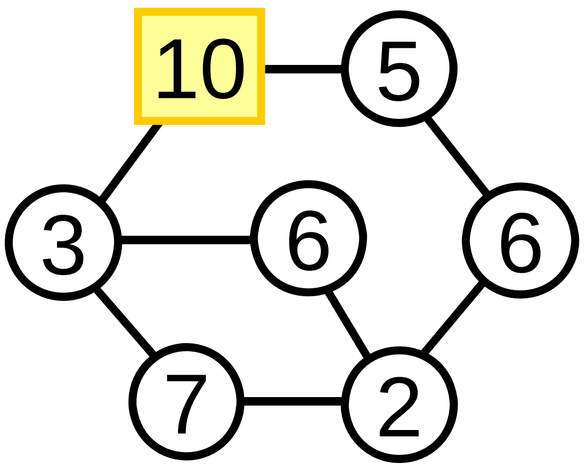
Pseudo-Polynomial-Time Algorithm

Suppose that a SP graph has exactly one supply.

Max PP for such a SP graph can be solved in time $O(F^2 n)$ if the demands and the supply are integers.

F : sum of all demands

n: # of vertices



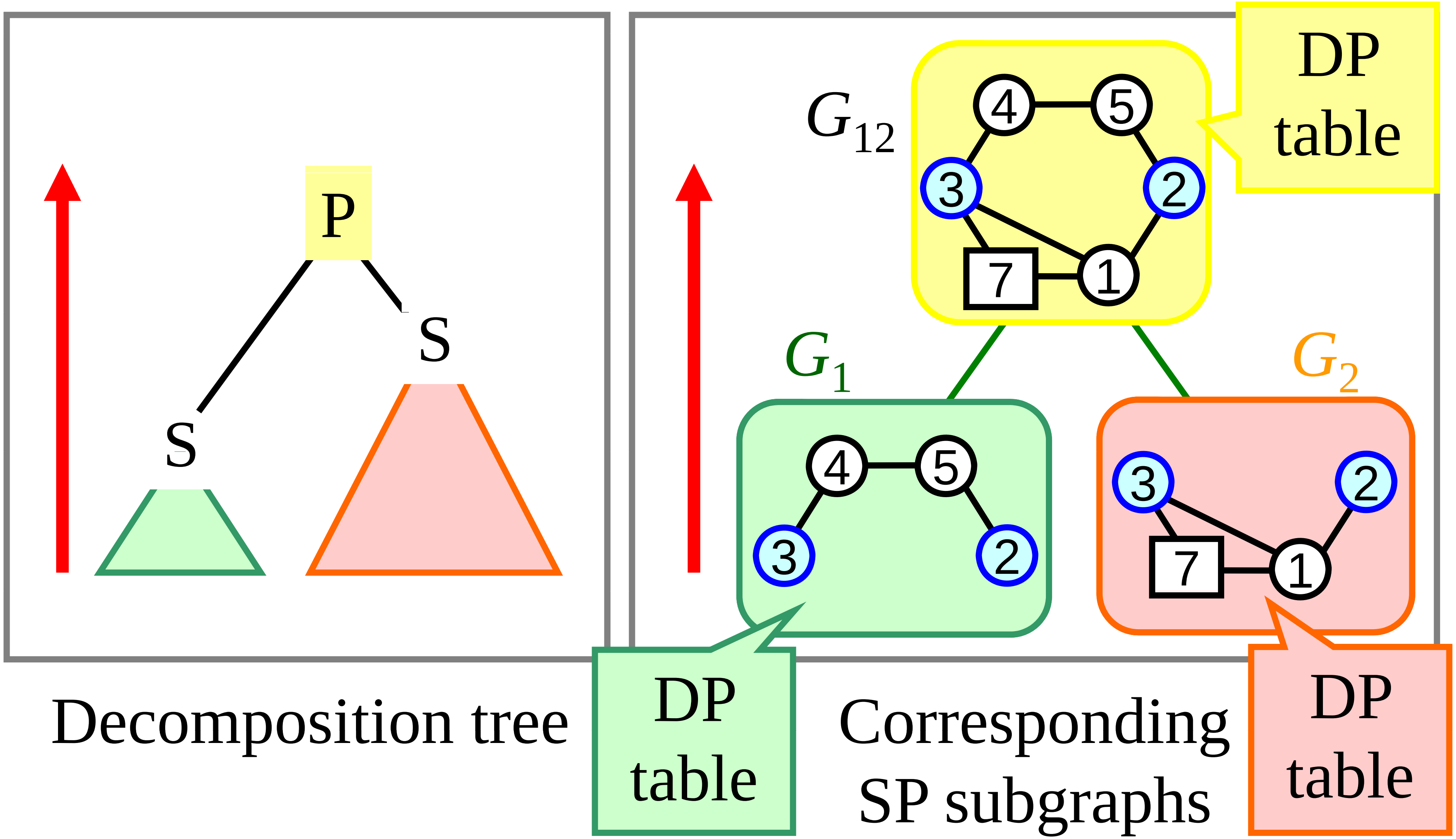
max fulfillment $\leq F$

pseudo-polynomial-time algorithm

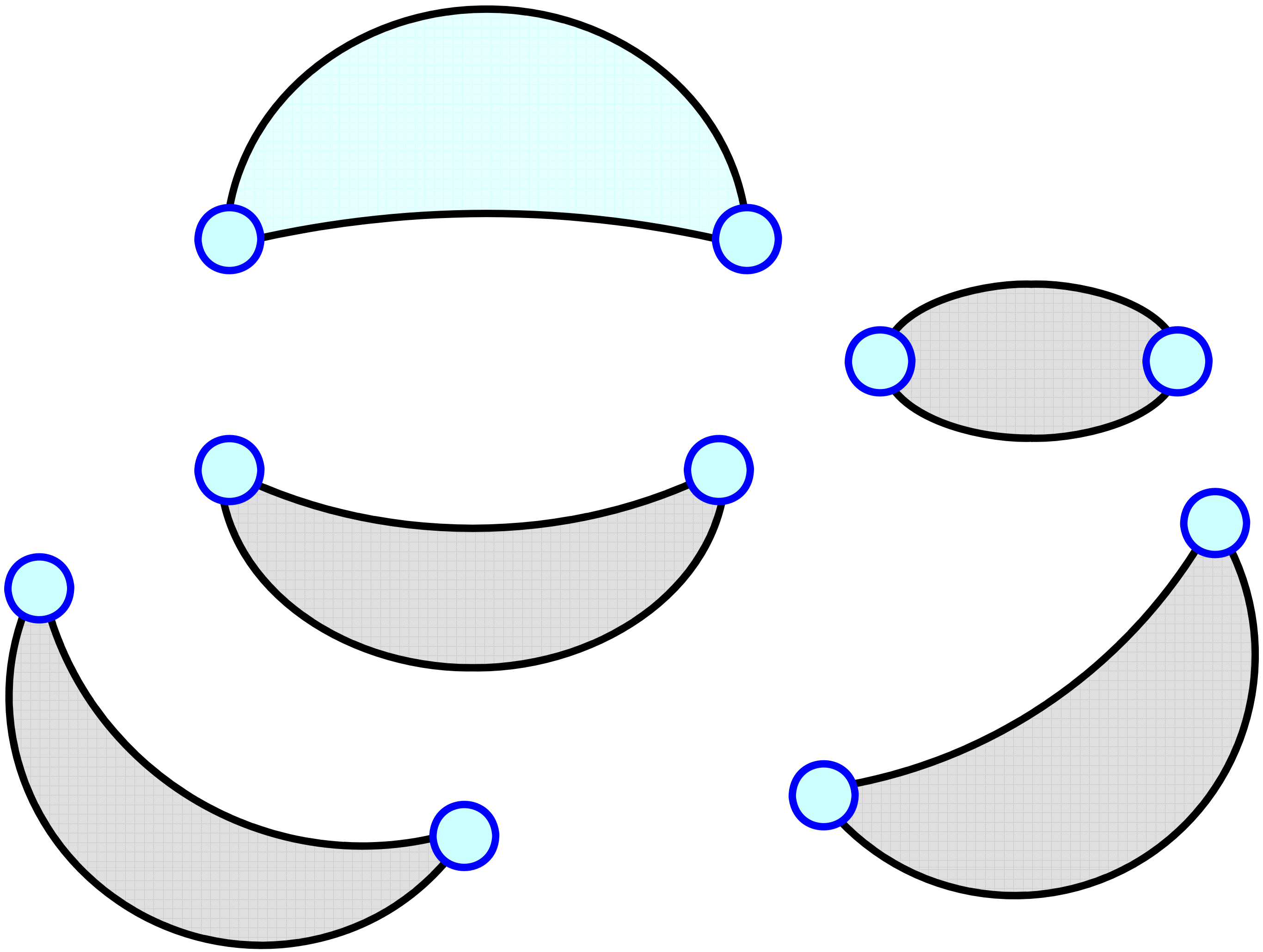
(2) FPTAS

Pseudo-Polynomial-Time Algorithm

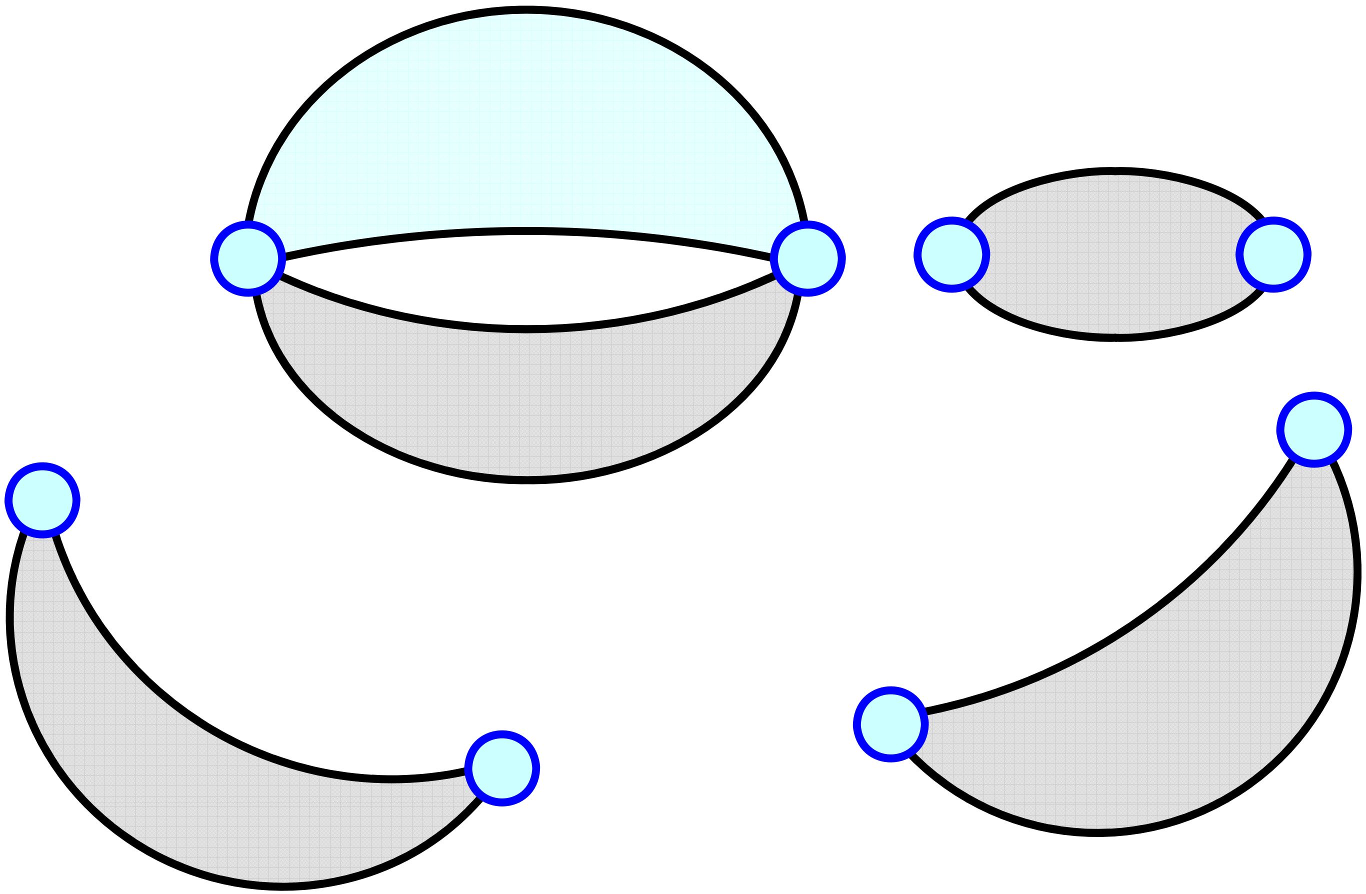
What should we store in the table for Max PP ?



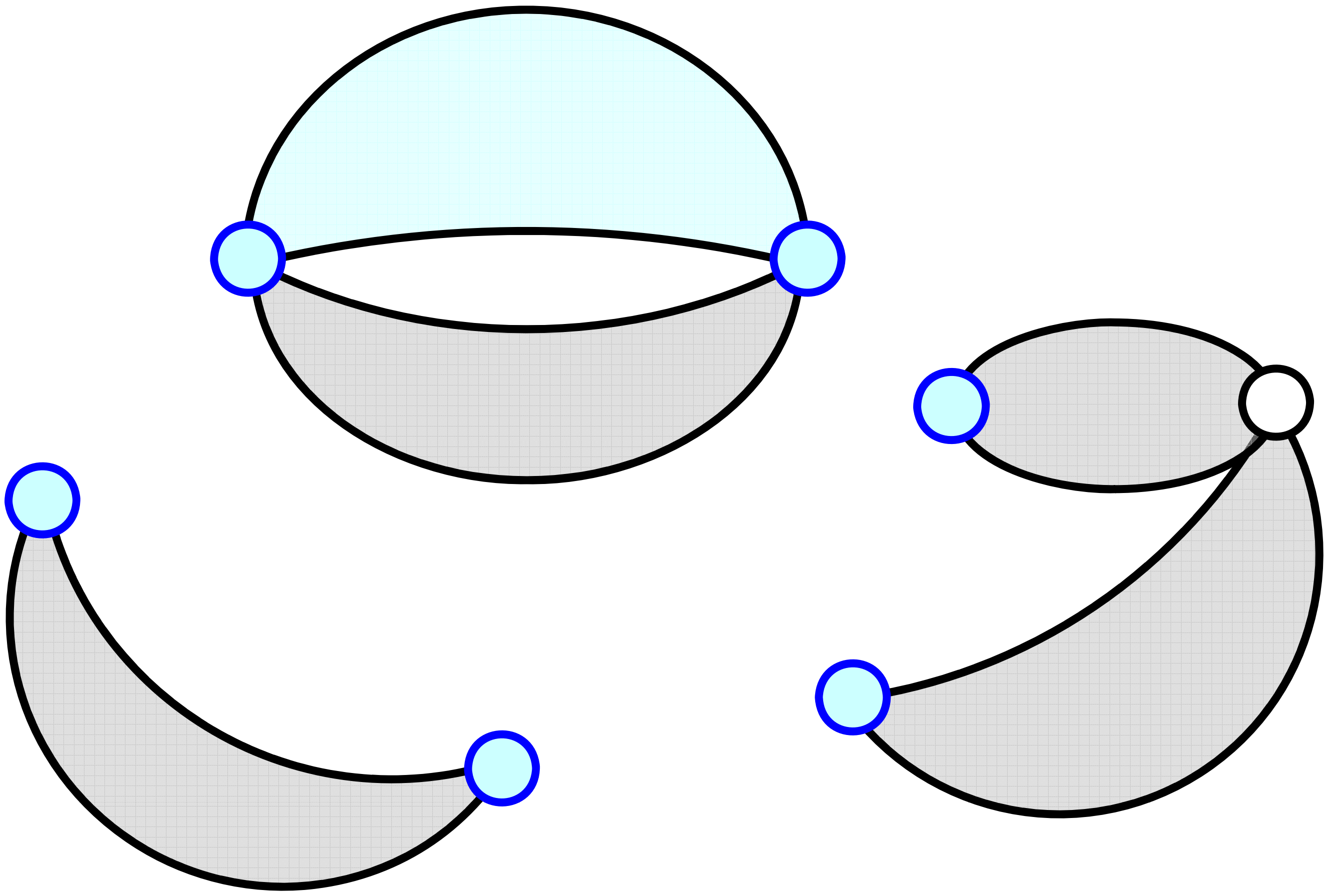
Pseudo-Polynomial-Time Algorithm



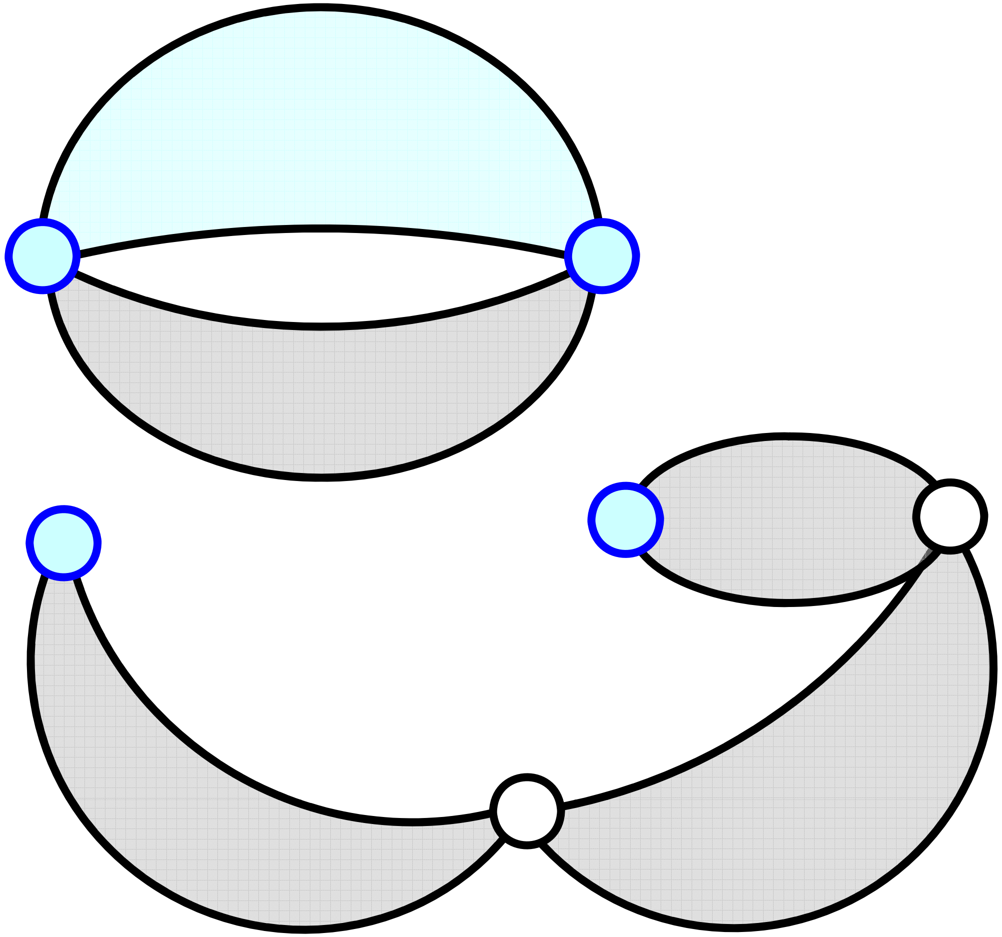
Pseudo-Polynomial-Time Algorithm



Pseudo-Polynomial-Time Algorithm

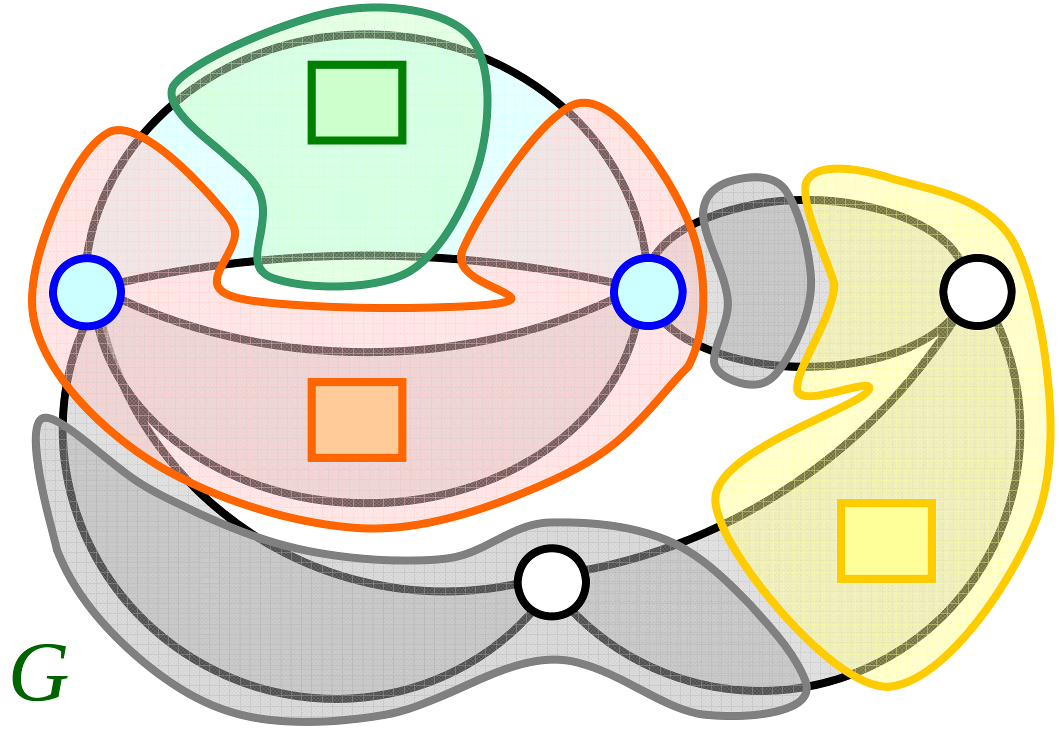


Pseudo-Polynomial-Time Algorithm



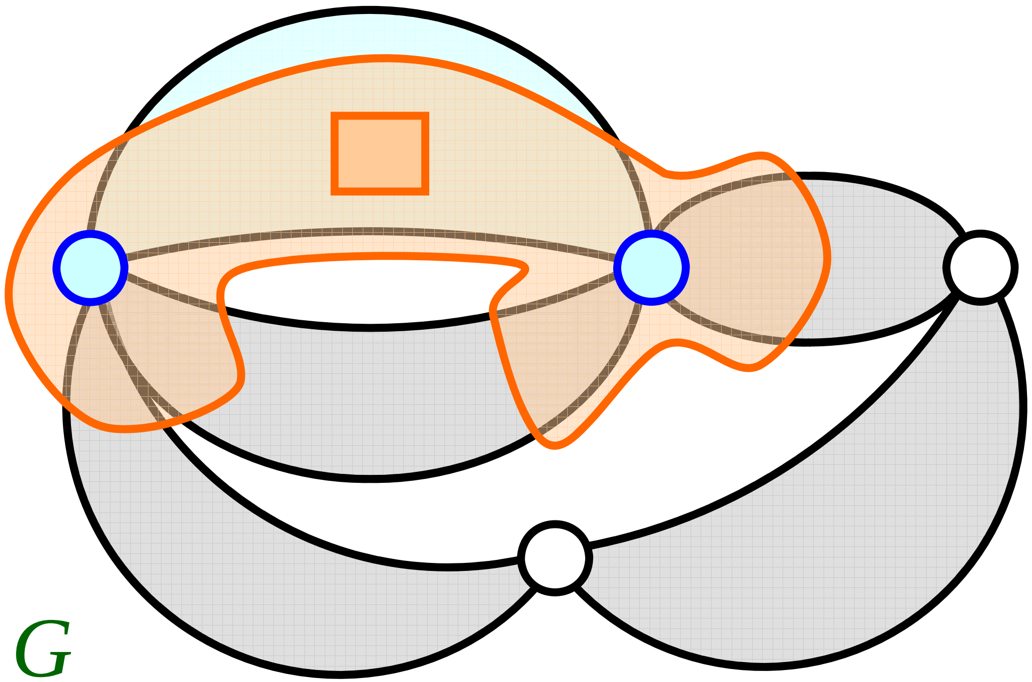
Pseudo-Polynomial-Time Algorithm

If *G* had more than one supply,
Max PP would find a partition.

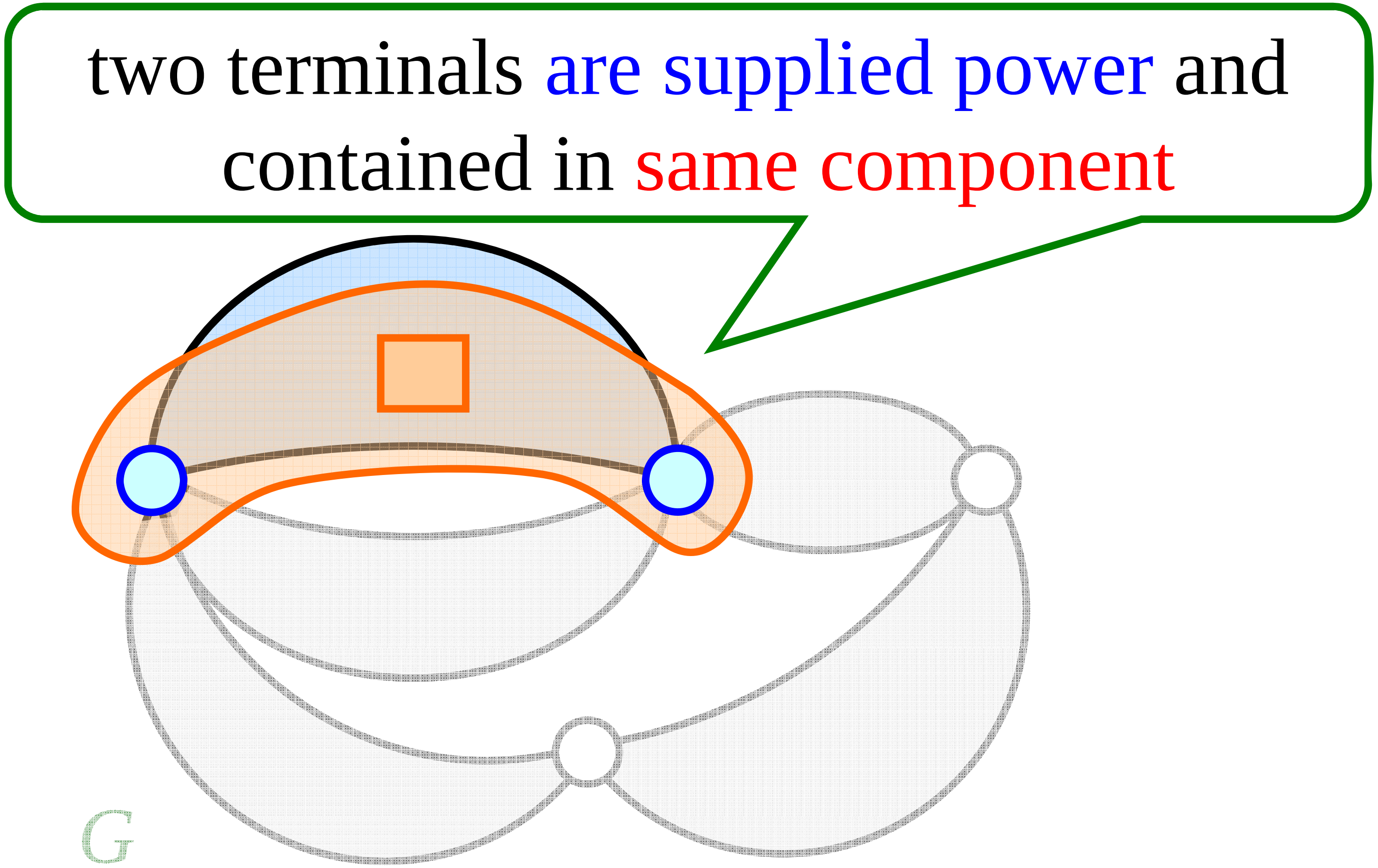


Pseudo-Polynomial-Time Algorithm

Max PP finds only one component with the supply

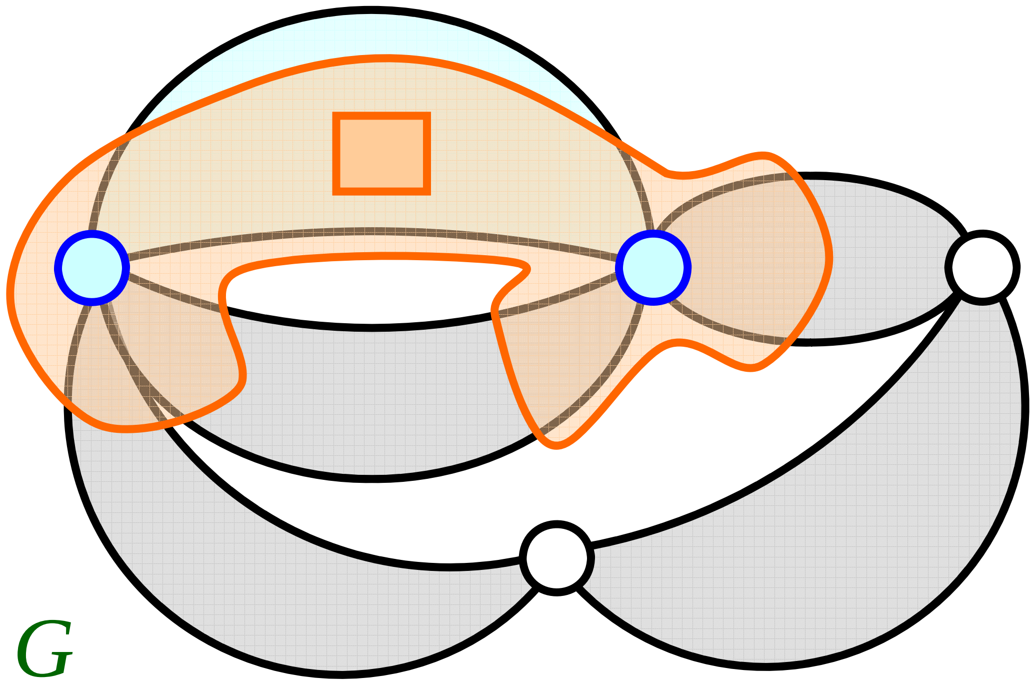


Pseudo-Polynomial-Time Algorithm



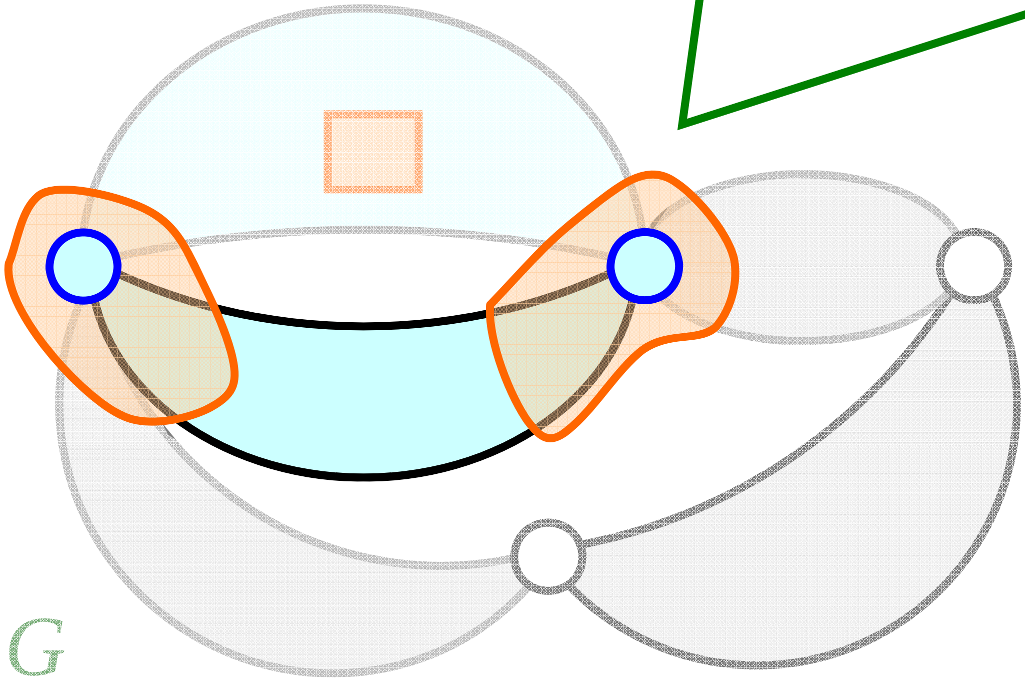
Pseudo-Polynomial-Time Algorithm

Max PP finds only one component with the supply



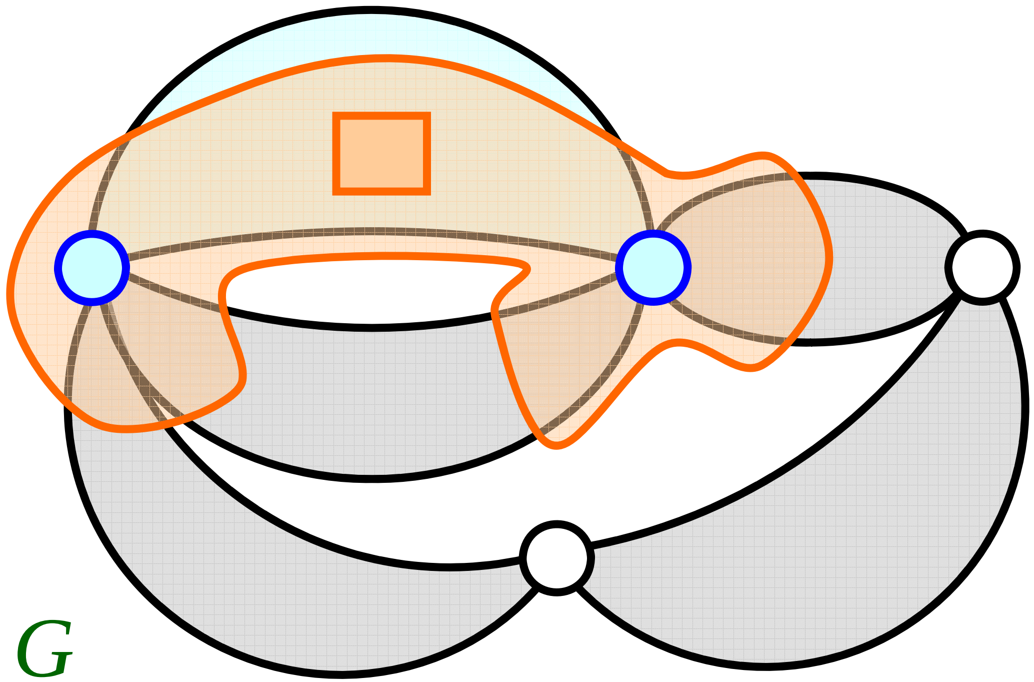
Pseudo-Polynomial-Time Algorithm

two terminals will be supplied power, but are contained in different components

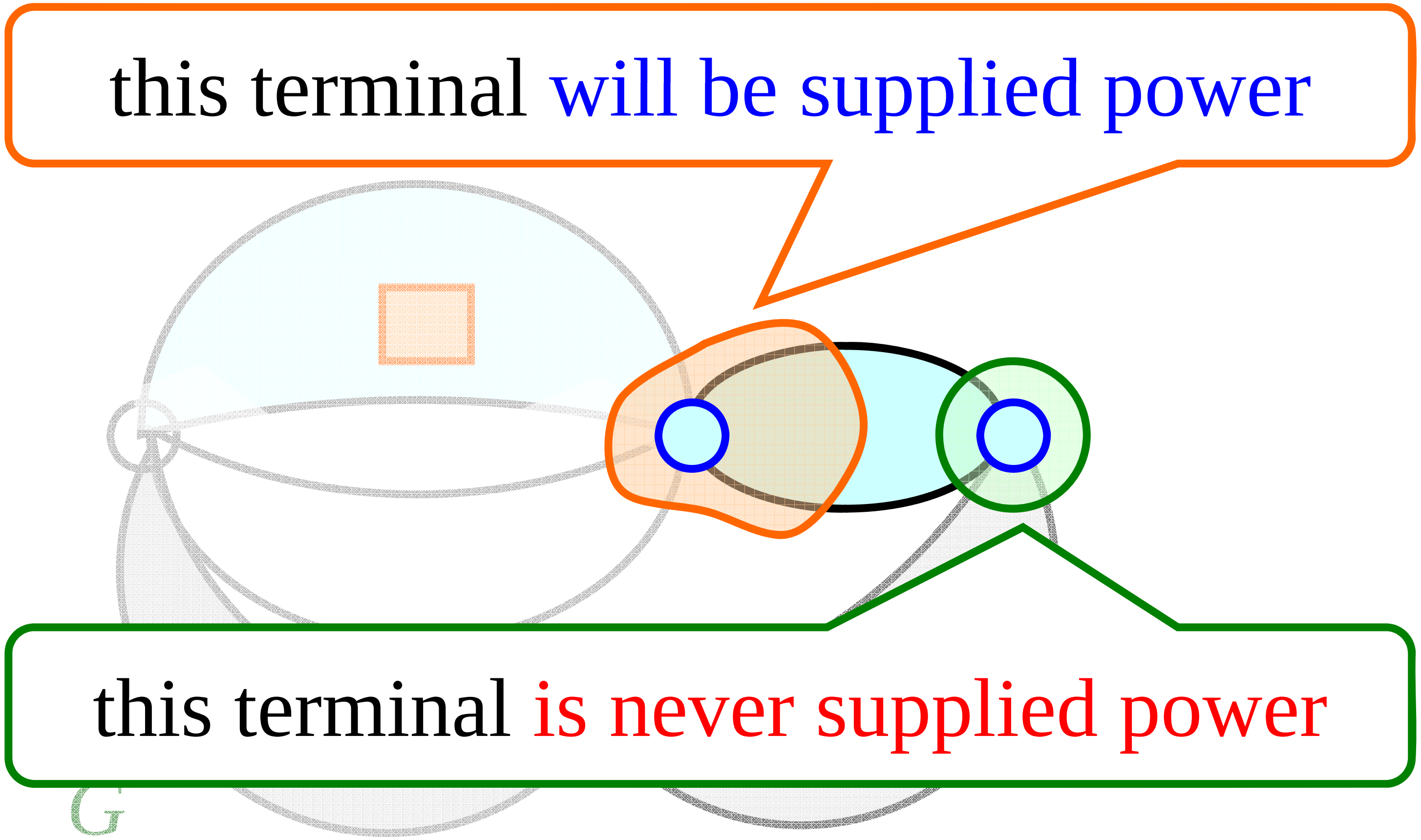


Pseudo-Polynomial-Time Algorithm

Max PP finds only one component with the supply

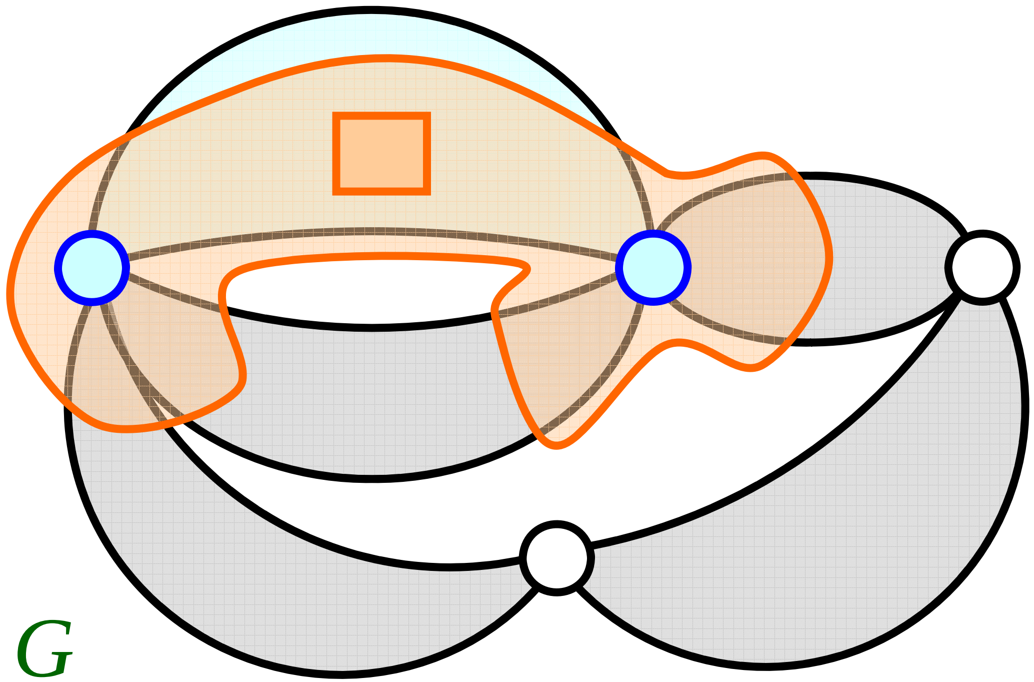


Pseudo-Polynomial-Time Algorithm



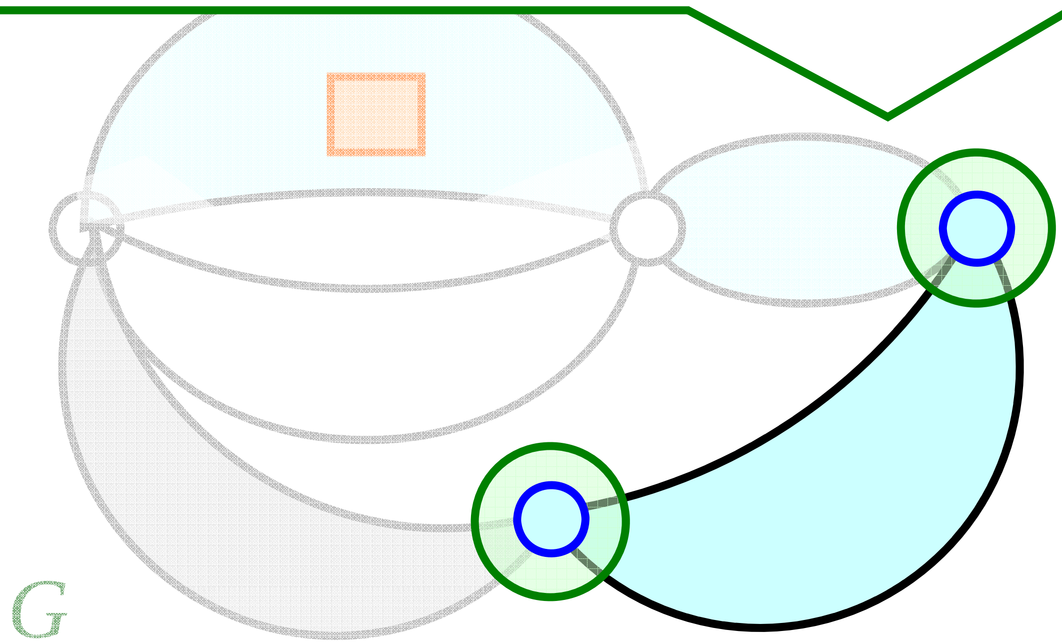
Pseudo-Polynomial-Time Algorithm

Max PP finds only one component with the supply



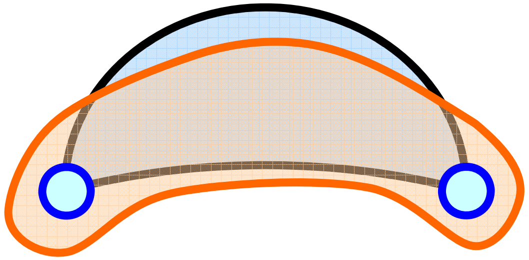
Pseudo-Polynomial-Time Algorithm

two terminals are never supplied power

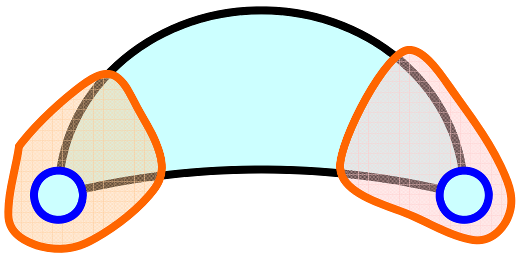


Pseudo-Polynomial-Time Algorithm

both are/will be supplied power

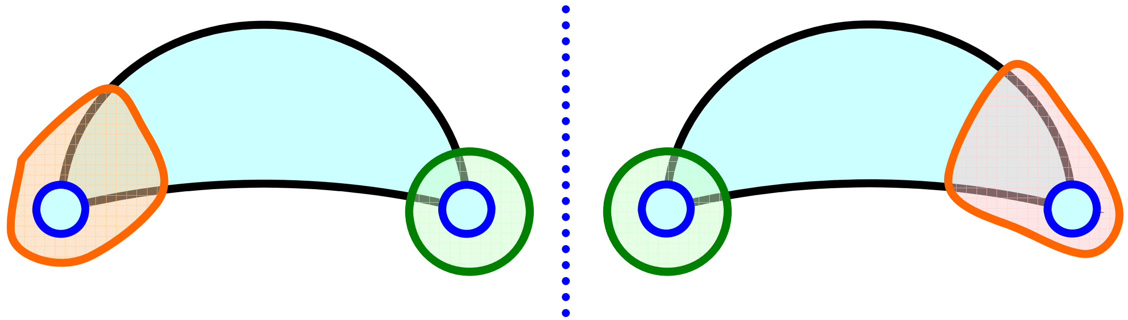


same component

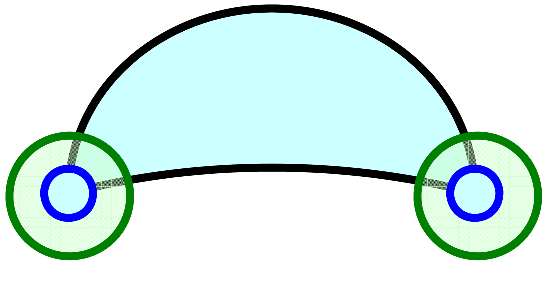


different components

one is/will be supplied, but the other is never supplied

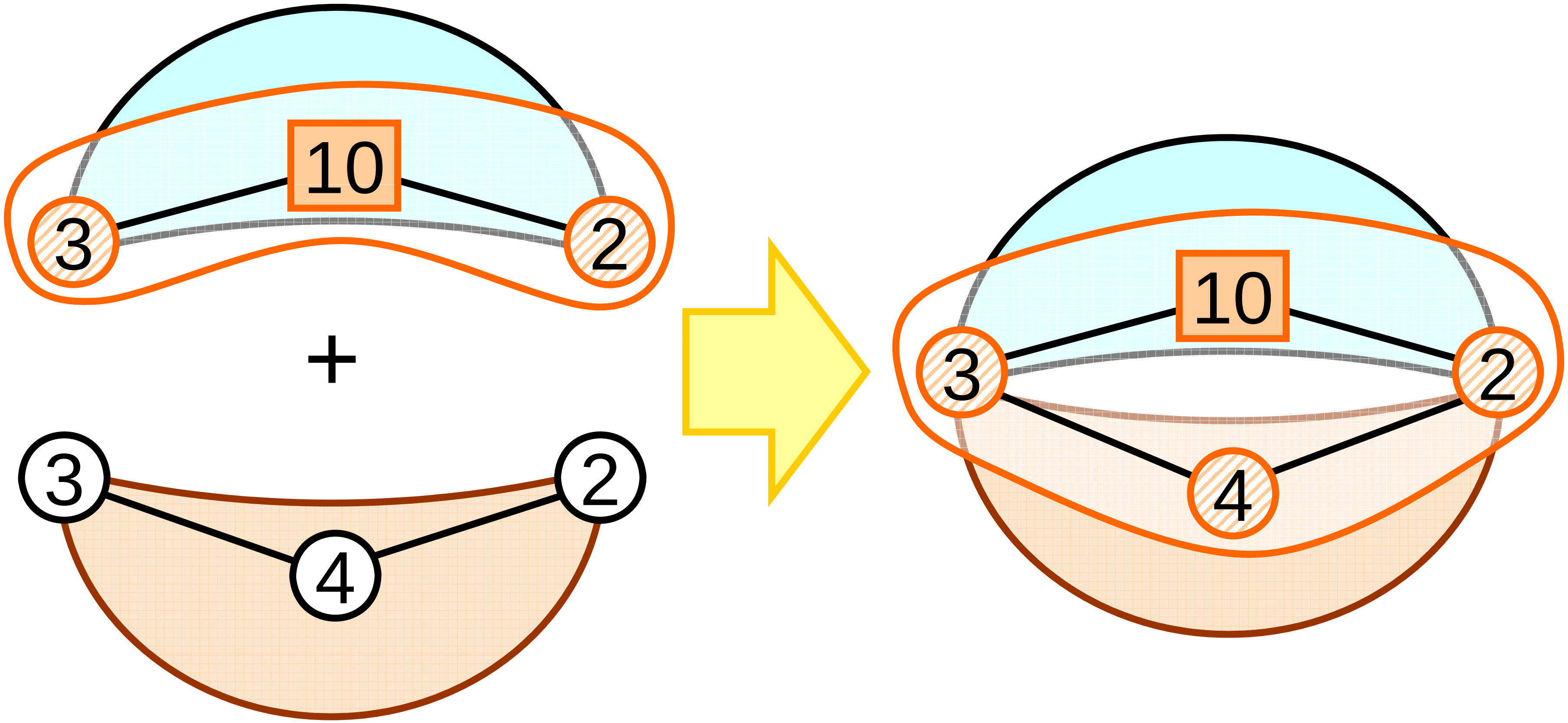


both are never supplied



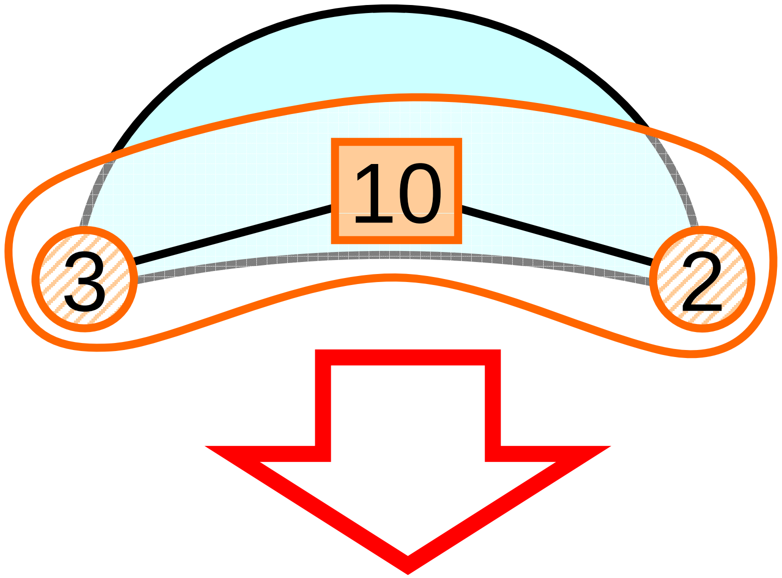
Pseudo-Polynomial-Time Algorithm

If the component has the supply vertex,
then the component may have the "marginal" power.



Pseudo-Polynomial-Time Algorithm

If the component has the supply vertex,
then the component may have the "marginal" power.

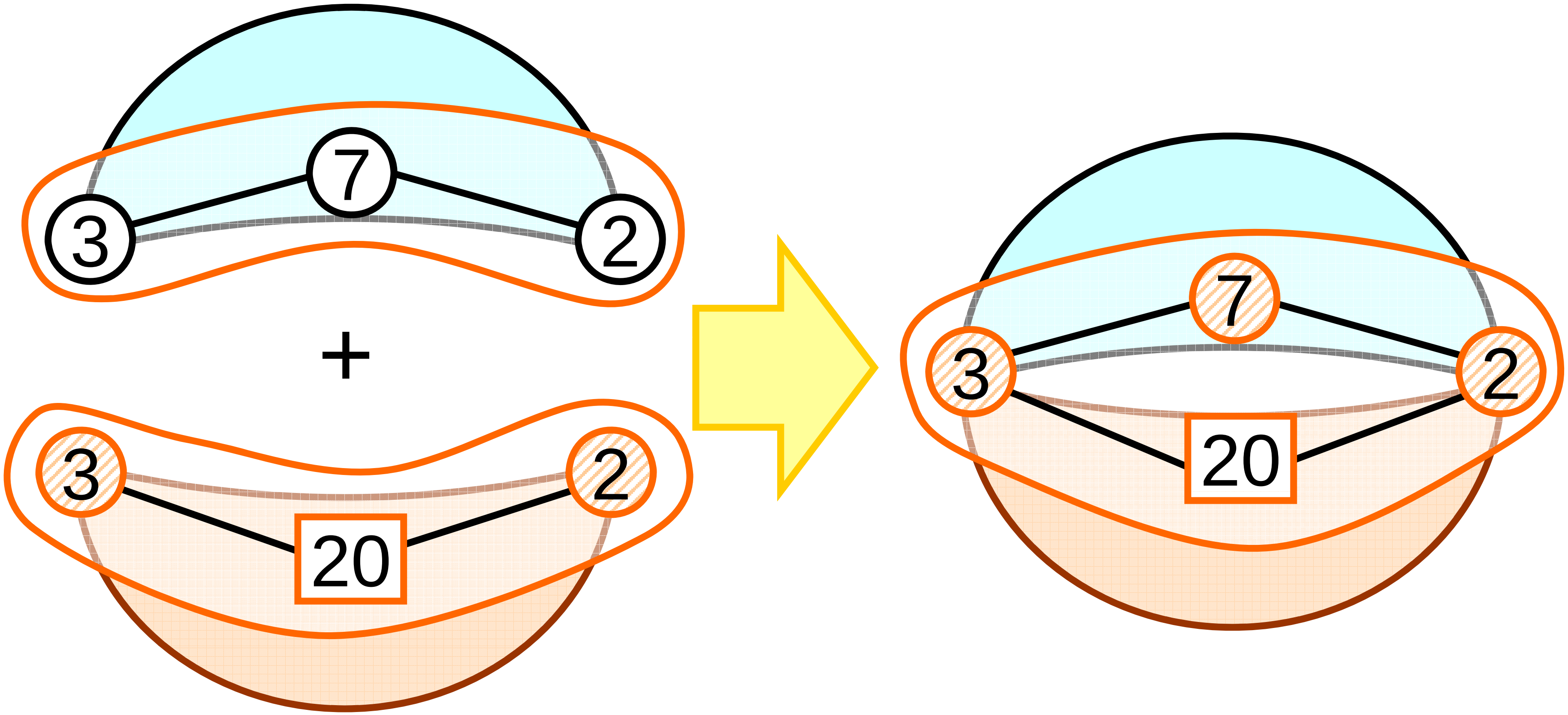


10 – (3+2) = 5

marginal power

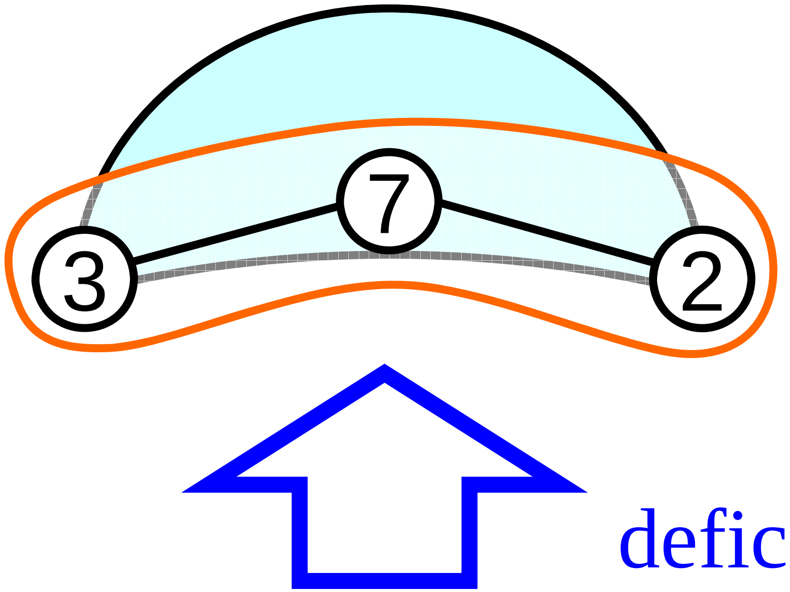
Pseudo-Polynomial-Time Algorithm

If the component has no supply vertex,
the component may have the "deficient" power.



Pseudo-Polynomial-Time Algorithm

If the component has no supply vertex,
the component may have the "deficient" power.

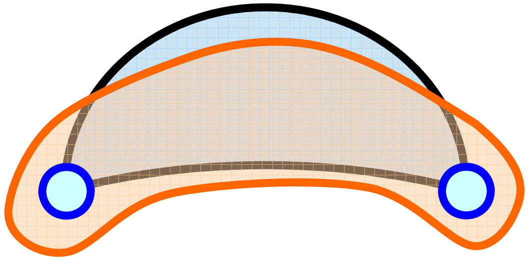


deficient power

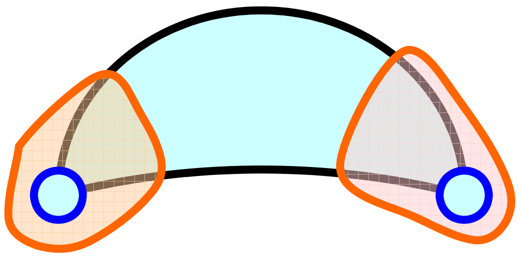
3+7+2 = 12

Pseudo-Polynomial-Time Algorithm

both are/will be supplied power

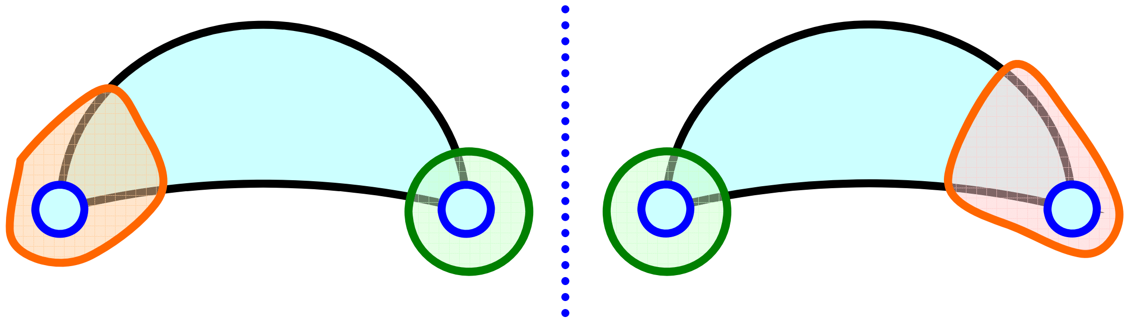


same component

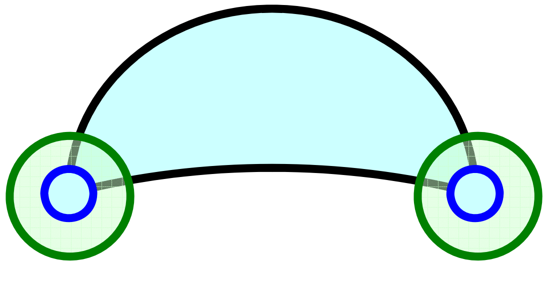


different components

one is/will be supplied, but the other is never supplied

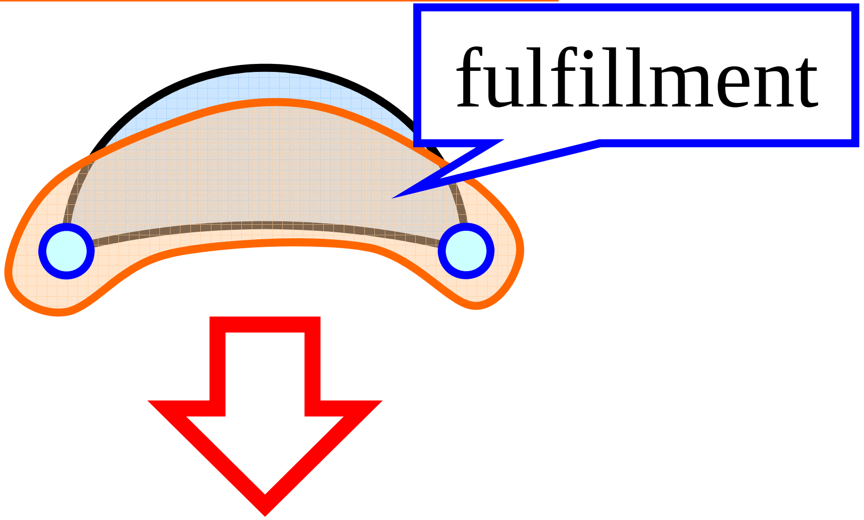


both are never supplied



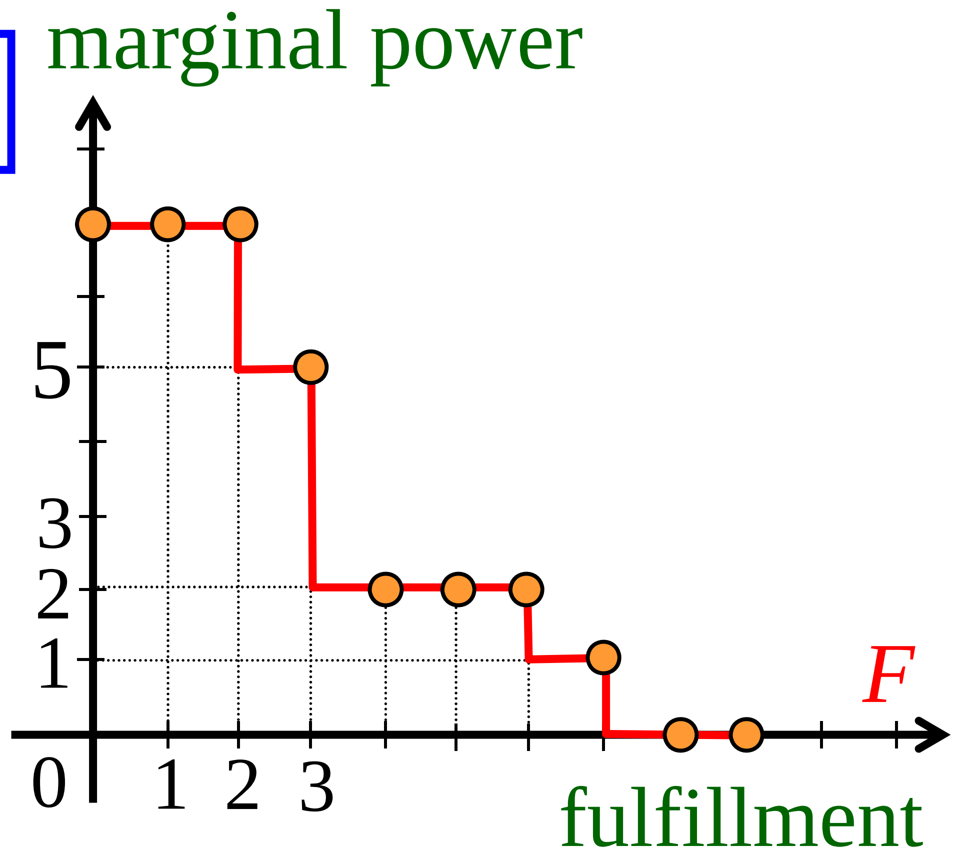
Pseudo-Polynomial-Time Algorithm

both are/will be supplied power



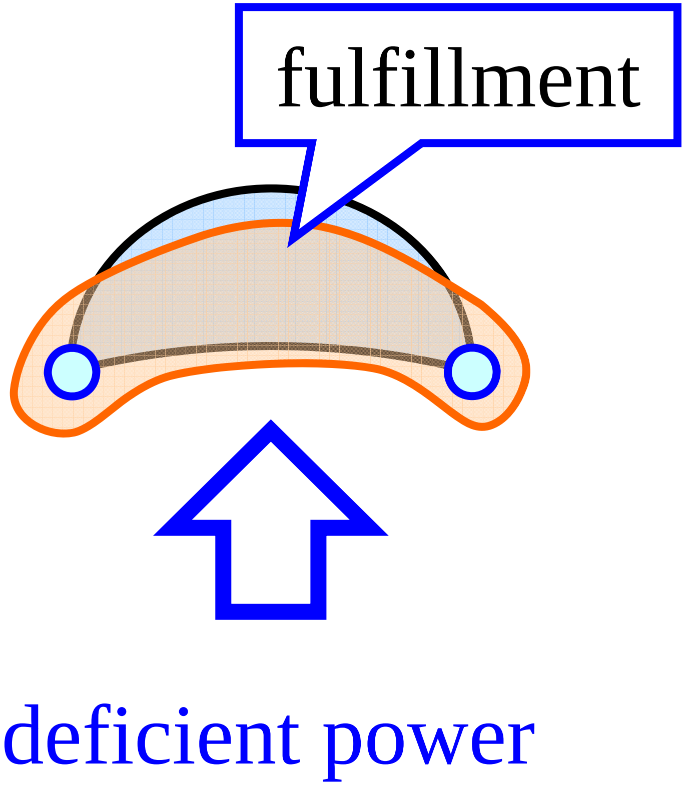
marginal power

staircase,non-increasing

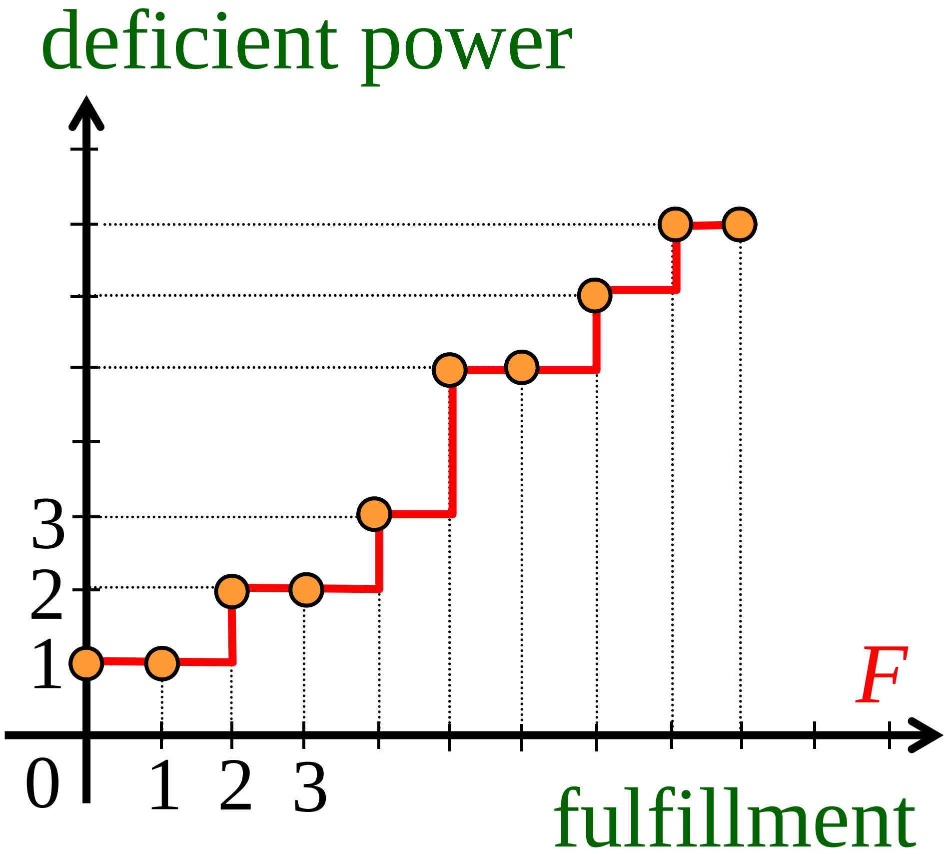


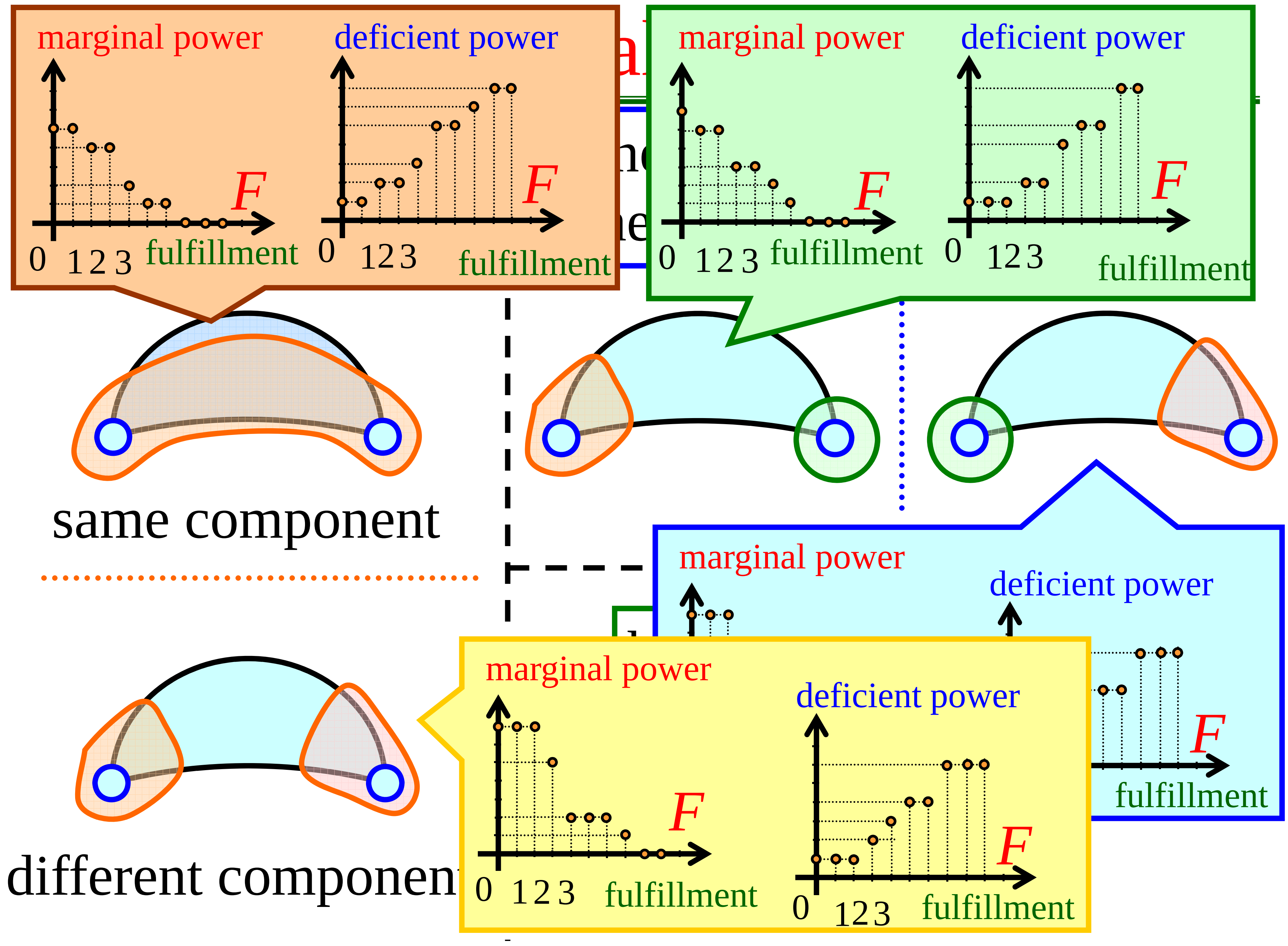
Pseudo-Polynomial-Time Algorithm

both are/will be supplied power



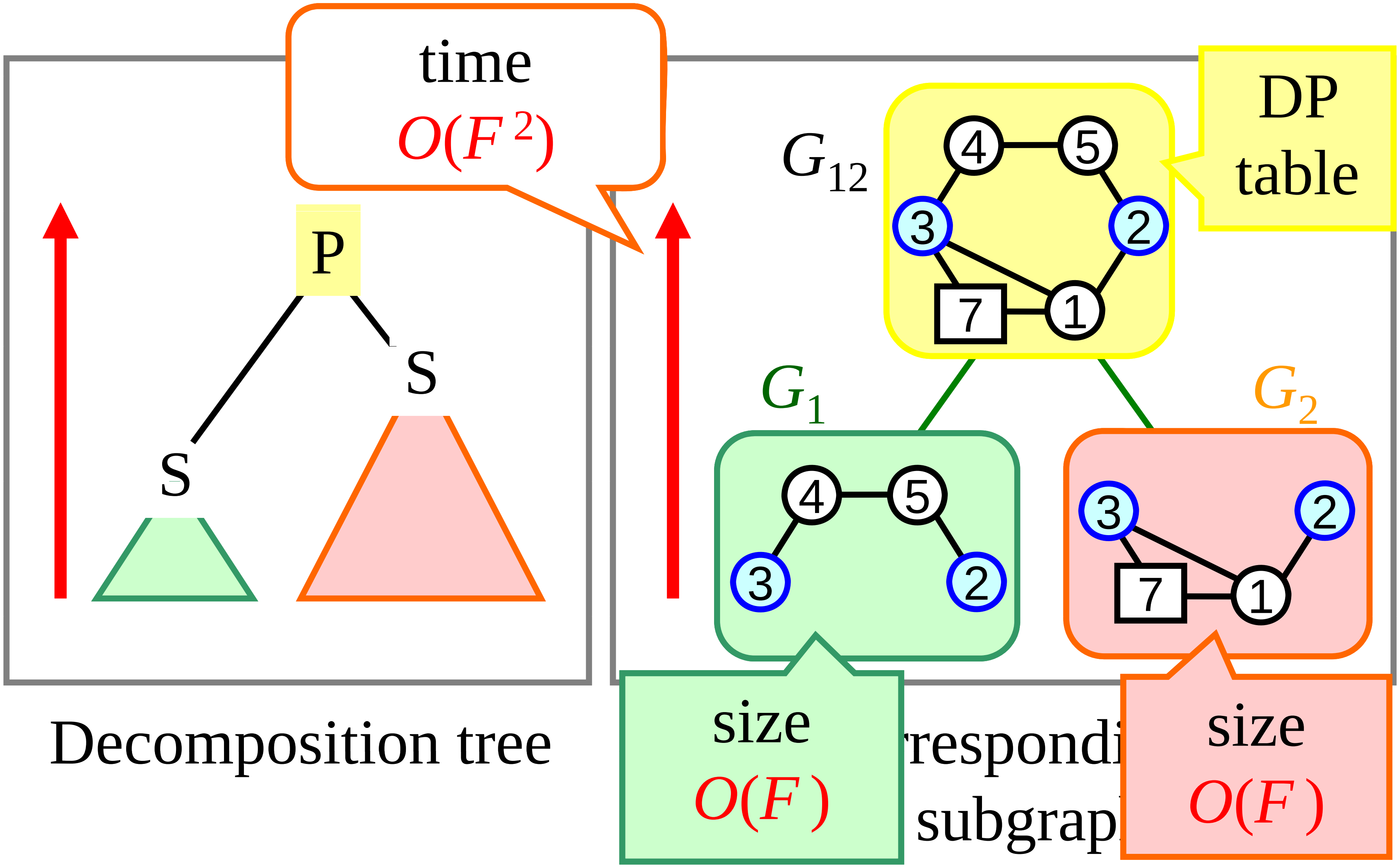
staircase, non-decreasing





marginal power
deficient power
F
0 1 2 3 fulfillment
same component
different component

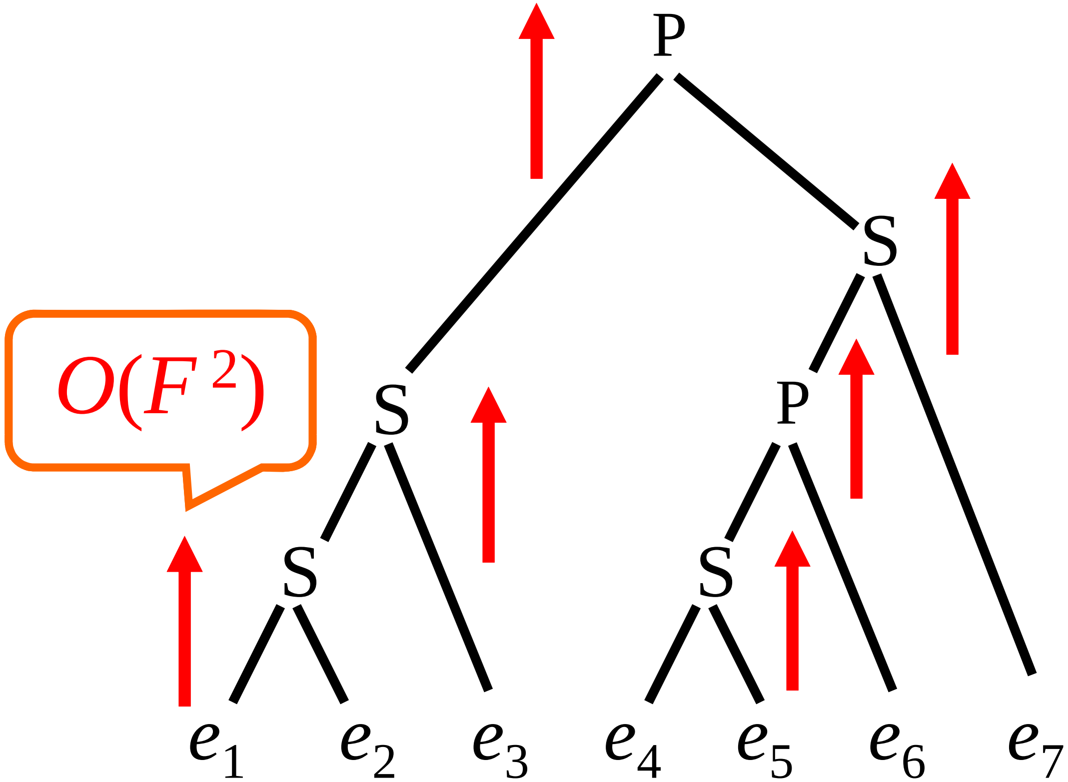
Pseudo-Polynomial-Time Algorithm



Decomposition tree

Corresponding subgraph

Pseudo-Polynomial-Time Algorithm



Decomposition tree

# of nodes = $O(n)$

$O(F^2 n)$

n : # of vertices in a given SP graph

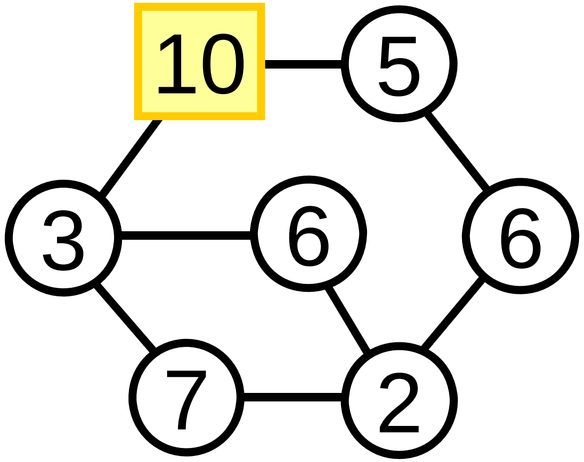
Pseudo-Polynomial-Time Algorithm

Suppose that a SP graph has exactly one supply.

Max PP for such a SP graph can be solved in time $O(F^2 n)$ if the demands and the supply are integers.

F : sum of all demands

n: # of vertices



max fulfillment $\leqq F$

pseudo-polynomial-time algorithm

(2) FPTAS

(2) FPTAS

Let all demands and supply be positive real numbers.

For any ε, $0 < \varepsilon < 1$, the algorithm finds a component with the supply vertex such that

$$\text{APPRO} > (1-\varepsilon)\ \text{OPT}$$

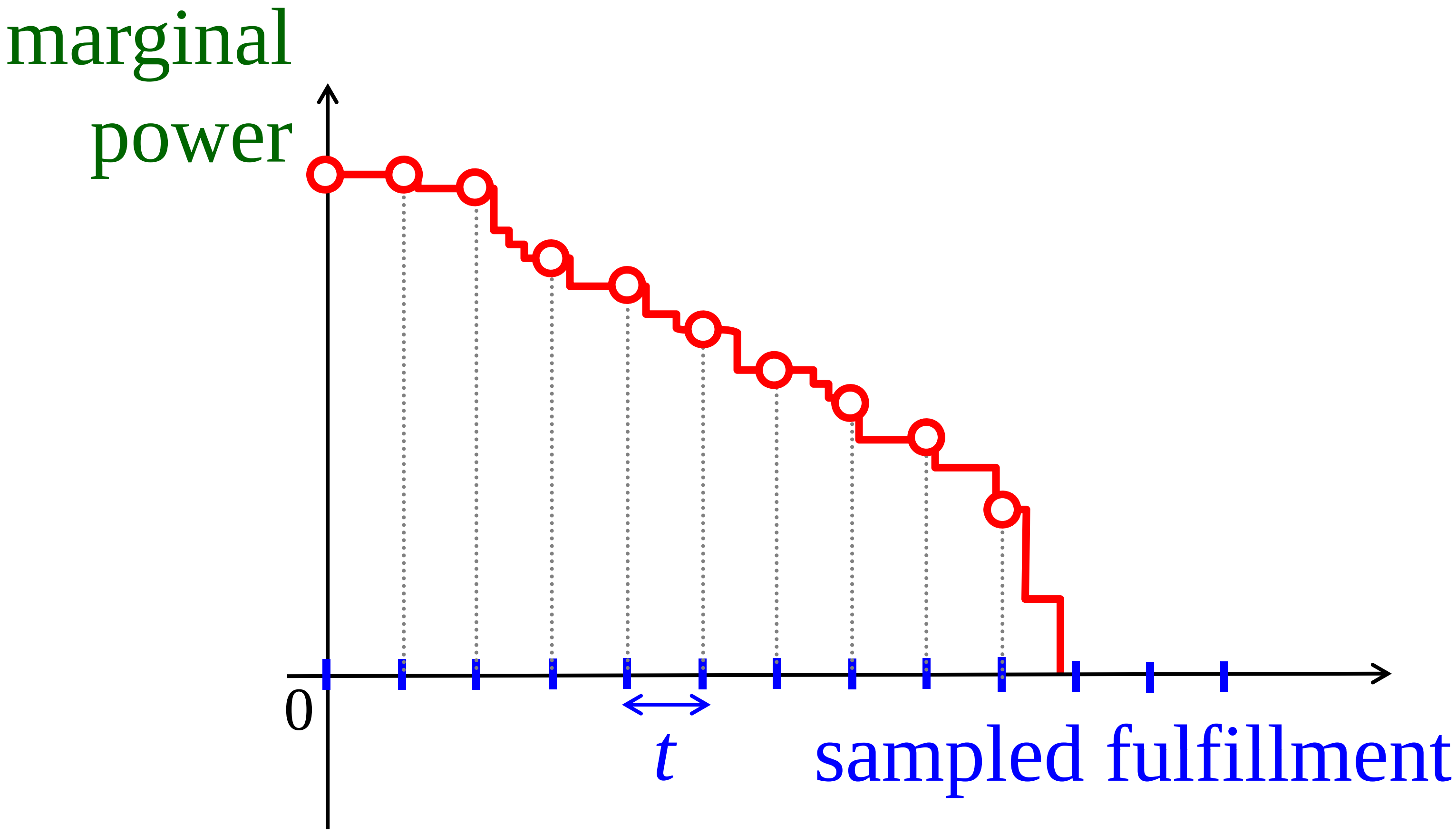
in time polynomial in both n and $1/\varepsilon$.

$$O\left(\frac{n^5}{\varepsilon^2}\right)$$

n : # of vertices

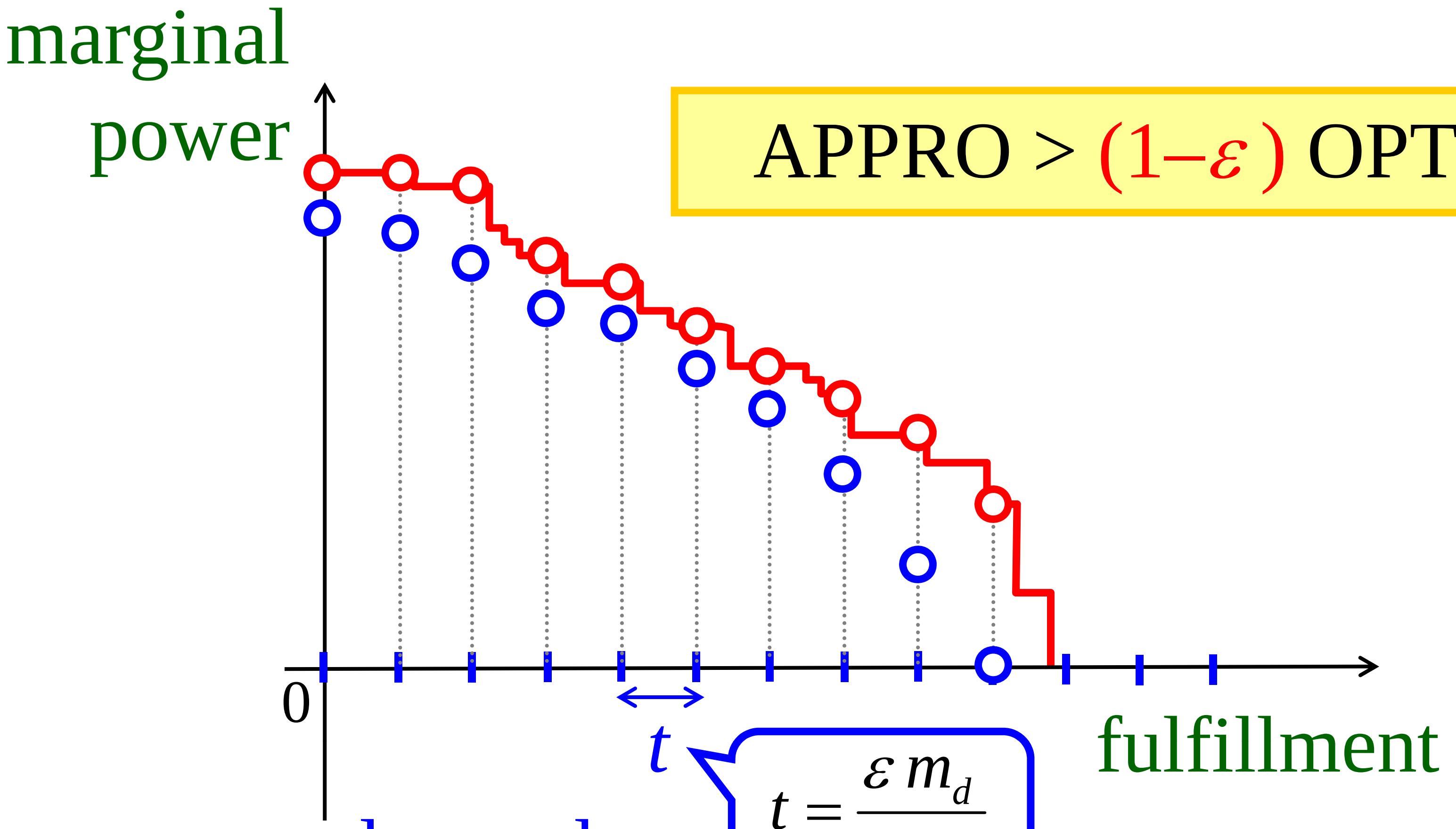
(2) FPTAS

The algorithm is similar to the previous algorithm.



(2) FPTAS

The algorithm is similar to the previous algorithm.

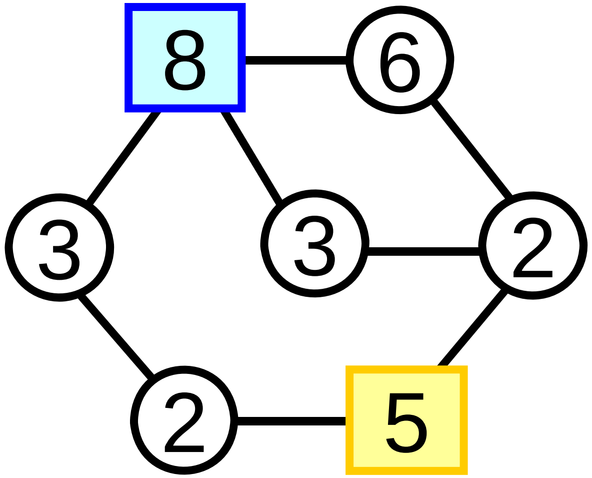


m_d: max demand

Open Problem

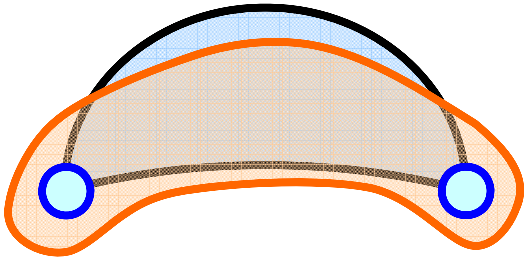
Is there an approximation algorithm for SP graphs having more than one supply?

- FPTAS or PTAS ?
- constant-factor ?

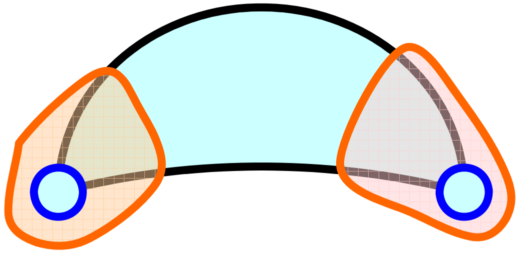


Open Problem

both are/will be supplied power

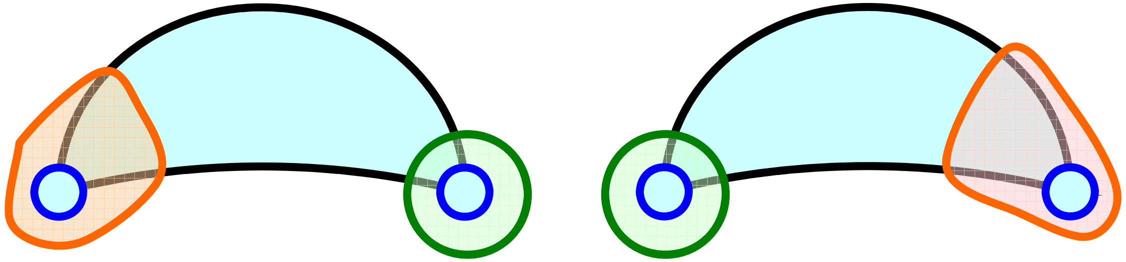


same component

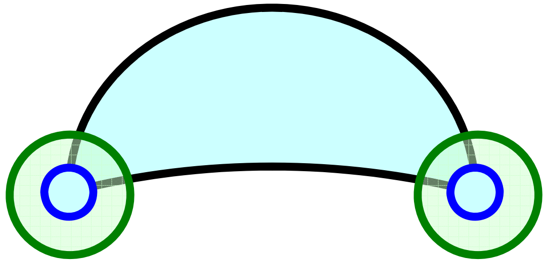


different components

one is/will be supplied, but the other is never supplied



both are never supplied

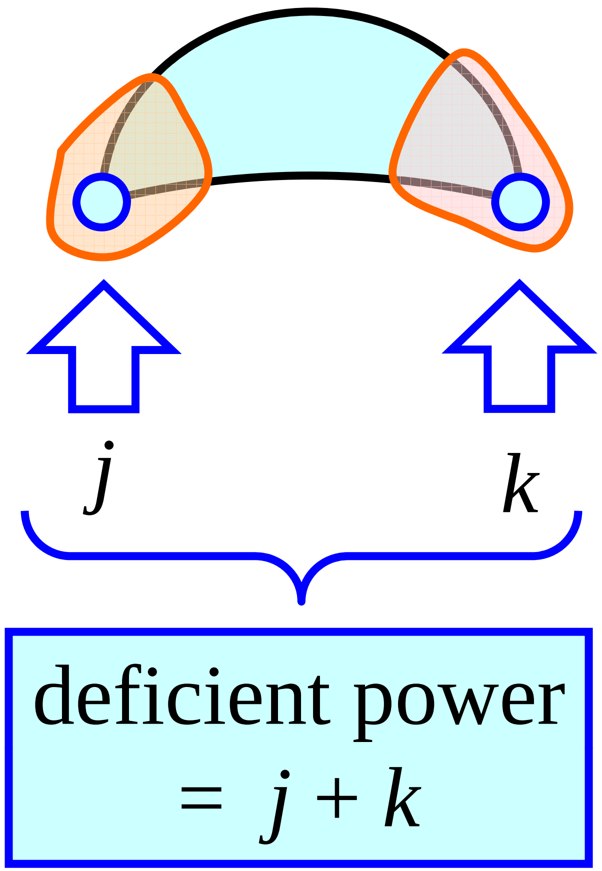


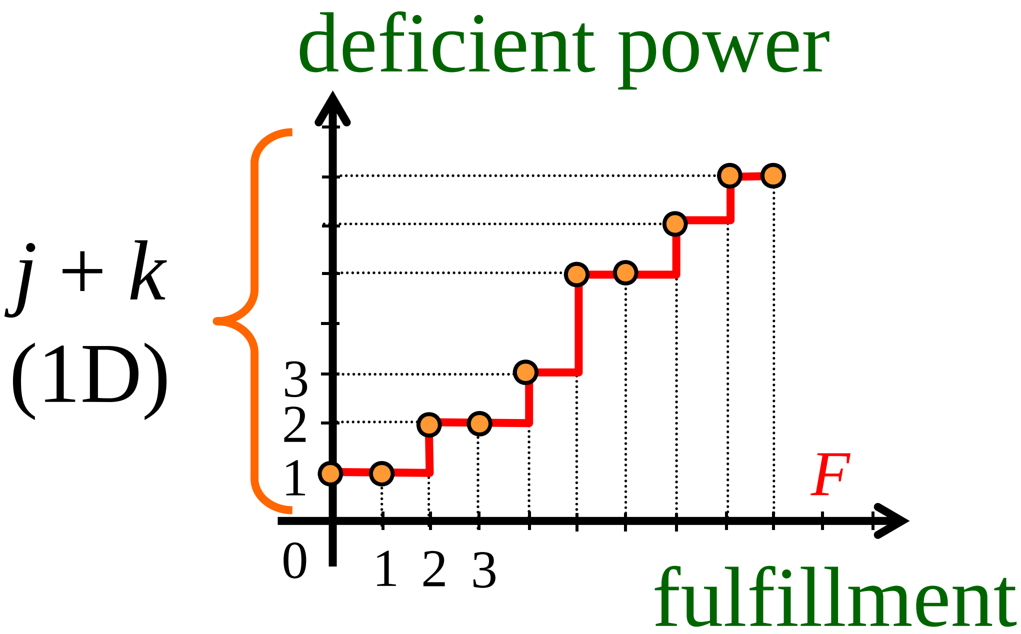
Open Problem

both are/will be supplied power

different components

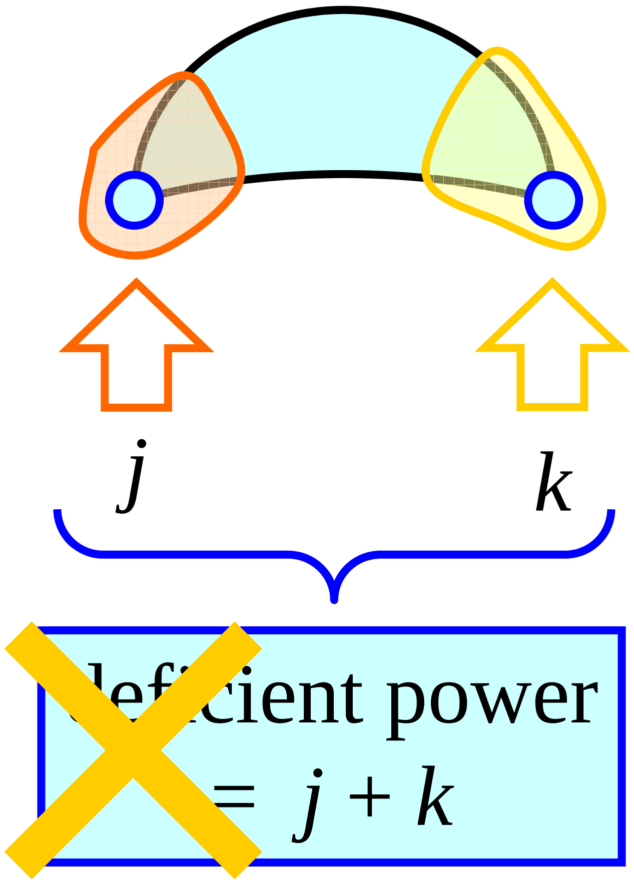
The two components must become **ONE** component.

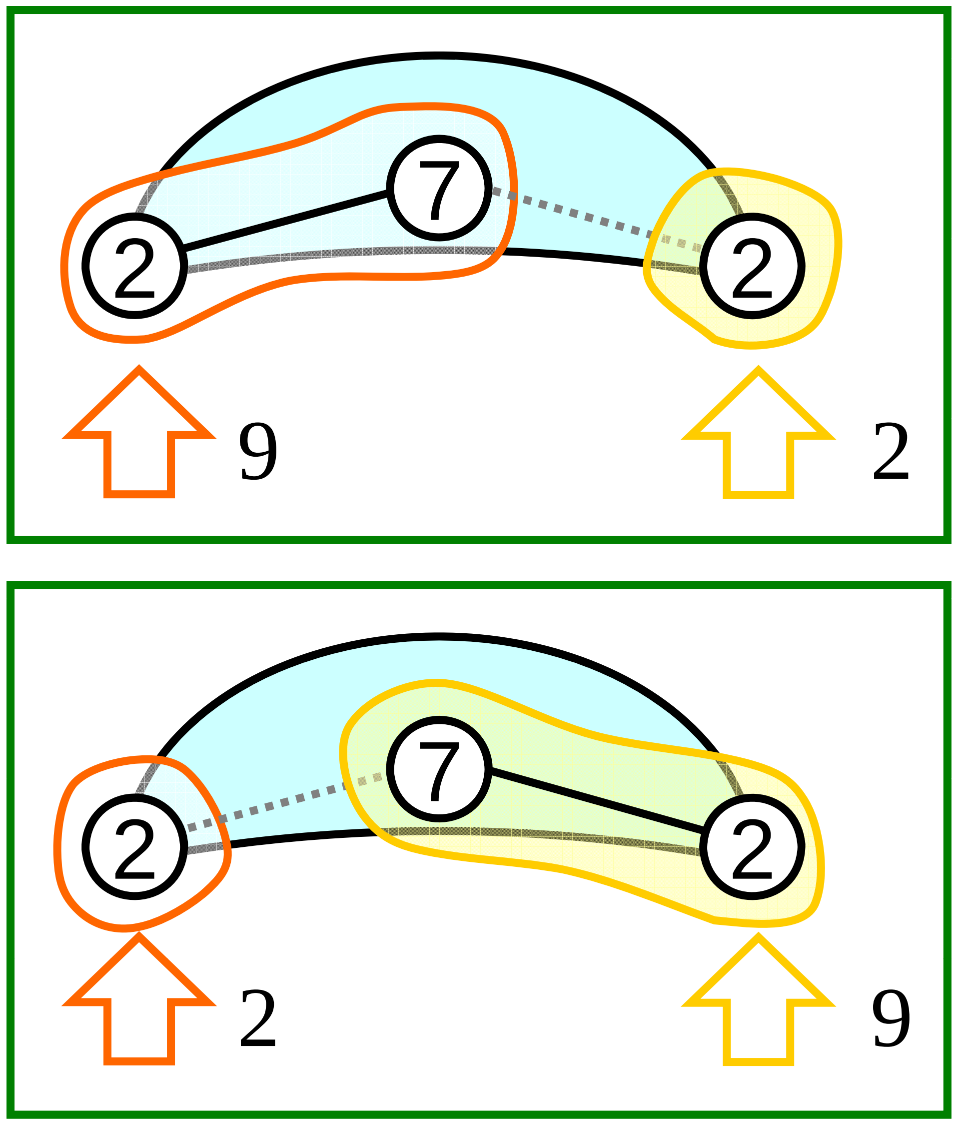




Open Problem

If a given graph has more than one supply.





Our Results

Max subset sum problem

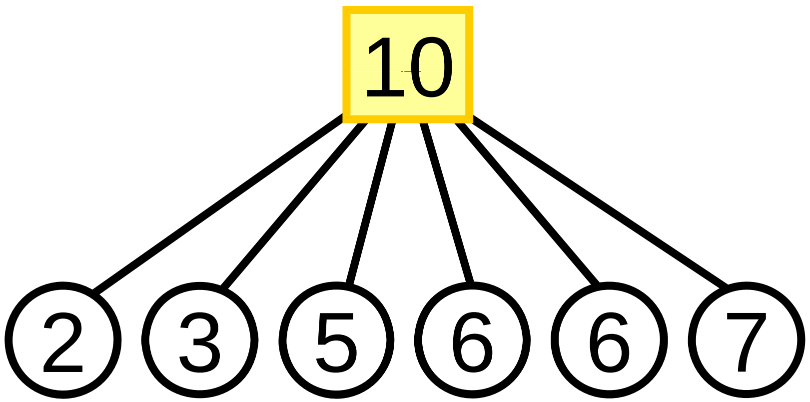
FPTAS

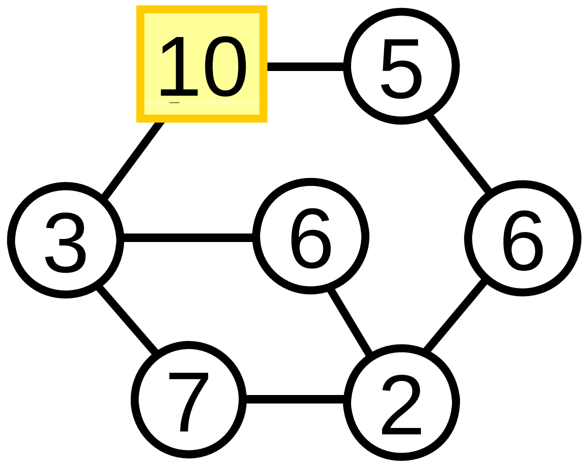
[Ibarra and Kim, 1975]

Max PP

our FPTAS

straightforward ?





Our Results

Max subset sum problem

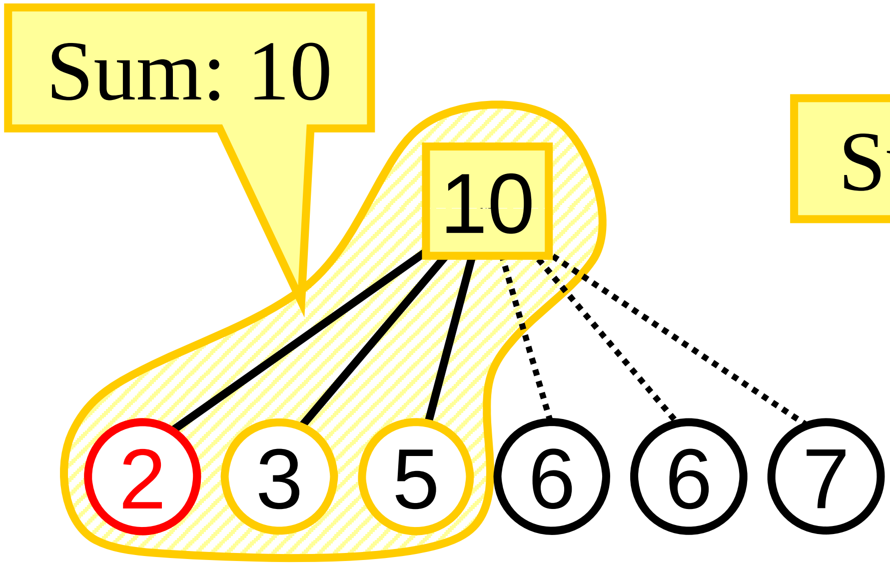
FPTAS

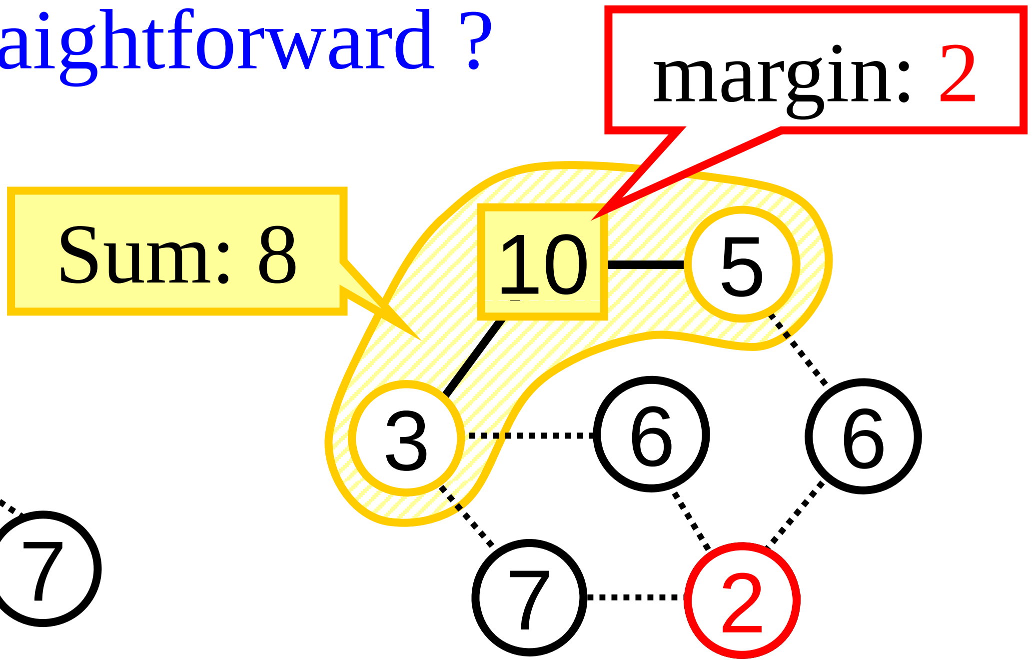
[Ibarra and Kim, 1975]

Max PP

our FPTAS

straightforward ?



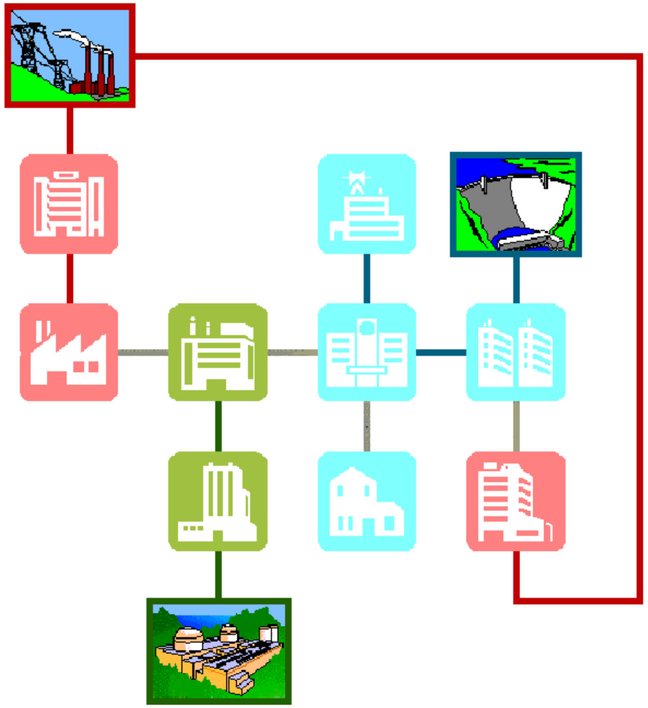


The graph structure plays crucial role.

Assumption of the Problem

Every demand vertex must be supplied from exactly one supply vertex.

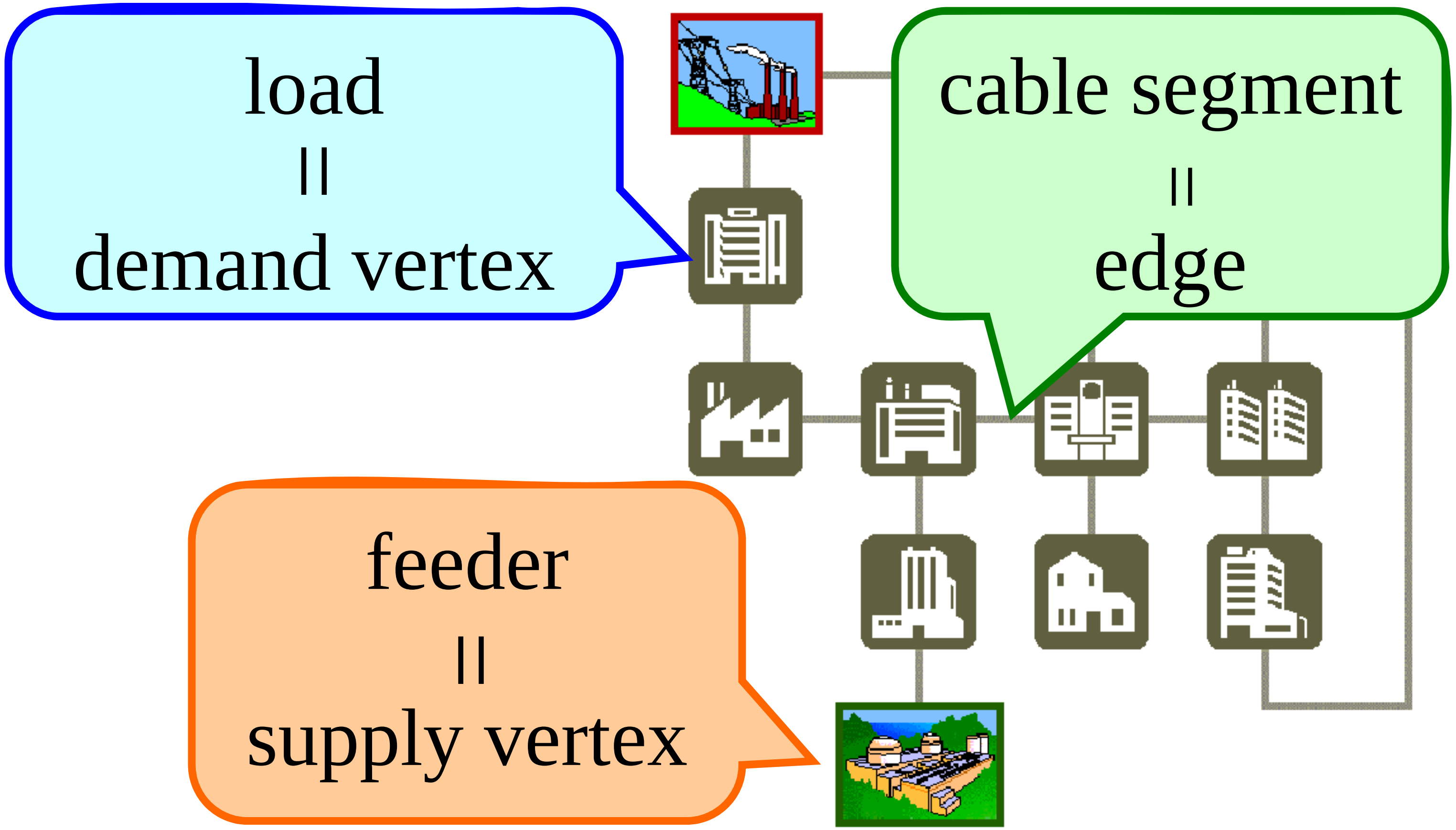
Max PP is applied to Power delivery networks.



It is difficult to synchronize two or more sources.

Application of Max PP

Power delivery problem

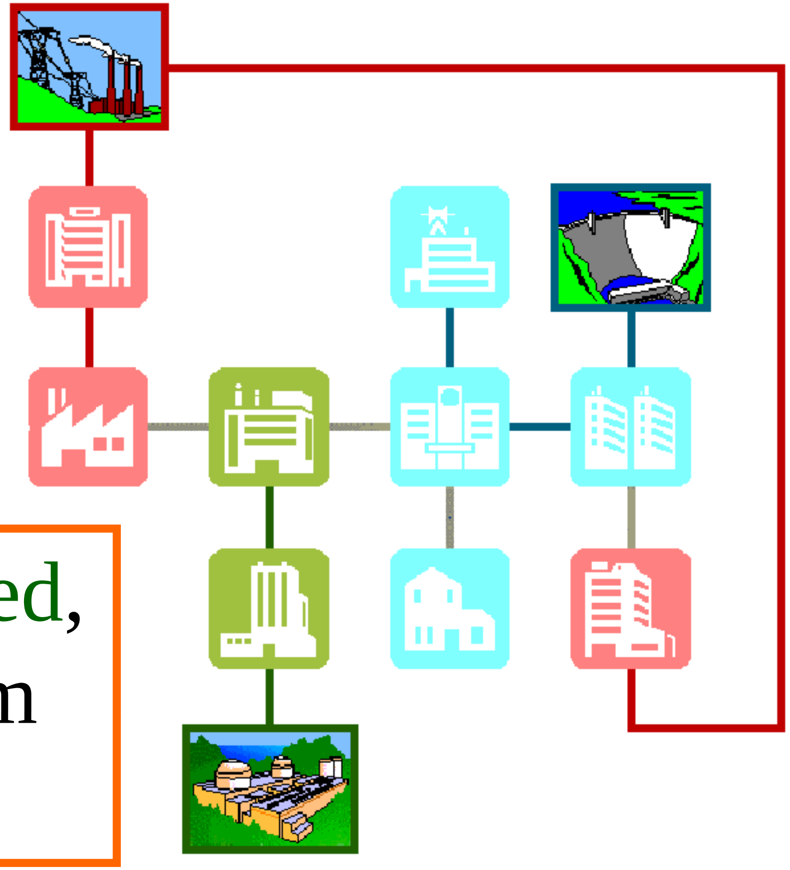


Power delivery network

Application of Max PP

Power delivery problem

Determine whether there exists a switching so that all loads can be supplied.



If not all loads can be supplied, we want to maximize the sum of loads supplied power.

⇨ Max Partition Problem

Power delivery network