Lezione 33

Tecnica Branch & Bound

Per la soluzione di problemi di ottimizzazione attraverso la Tecnica Backtracking un possibile approccio è quello di enumerare tutte le soluzioni ammissibili e, nel corso dell'enumerazione, tener traccia della miglior soluzione ammissibile fino al momento trovata (ottimo attuale).

Per la soluzione di problemi di ottimizzazione attraverso la Tecnica Backtracking un possibile approccio è quello di enumerare tutte le soluzioni ammissibili e, nel corso dell'enumerazione, tener traccia della miglior soluzione ammissibile fino al momento trovata (ottimo attuale).

Al termine dell' enumerazione, l'ottimo attuale coinciderà con la soluzione al problema.

Per la soluzione di problemi di ottimizzazione attraverso la Tecnica Backtracking un possibile approccio è quello di enumerare tutte le soluzioni ammissibili e, nel corso dell'enumerazione, tener traccia della miglior soluzione ammissibile fino al momento trovata (ottimo attuale).

Al termine dell' enumerazione, l'ottimo attuale coinciderà con la soluzione al problema. Nel corso della visita dell'albero delle soluzioni, al fine di ridurre i tempi di calcolo, conviene ricorrere a funzioni di taglio in grado di potare sottoalberi che non contengono soluzioni ammissibili.

Per la soluzione di problemi di ottimizzazione attraverso la Tecnica Backtracking un possibile approccio è quello di enumerare tutte le soluzioni ammissibili e, nel corso dell'enumerazione, tener traccia della miglior soluzione ammissibile fino al momento trovata (ottimo attuale).

Al termine dell' enumerazione, l'ottimo attuale coinciderà con la soluzione al problema. Nel corso della visita dell'albero delle soluzioni, al fine di ridurre i tempi di calcolo, conviene ricorrere a funzioni di taglio in grado di potare sottoalberi che non contengono soluzioni ammissibili.

Nel contesto dei problemi di ottimizzazione si rivela utile anche un'altro tipo di funzioni di taglio: funzioni in grado di potare sottoalberi che non contengono soluzioni ammissibili in grado di **migliorare** l'ottimo attuale.

La tecnica Branch & Bound

- Si applica quando l'albero delle possibili soluzioni ad un problema algoritmico rappresenta soluzioni parziali o ammissibili per un problema di ottimizzazione.

La tecnica Branch & Bound

- Si applica quando l'albero delle possibili soluzioni ad un problema algoritmico rappresenta soluzioni parziali o ammissibili per un problema di ottimizzazione. Ad ogni nodo possiamo calcolare un limite superiore e uno inferiore al valore delle soluzioni ammissibili *raggiungibili* dal nodo.

La tecnica Branch & Bound

- Si applica quando l'albero delle possibili soluzioni ad un problema algoritmico rappresenta soluzioni parziali o ammissibili per un problema di ottimizzazione. Ad ogni nodo possiamo calcolare un limite superiore e uno inferiore al valore delle soluzioni ammissibili *raggiungibili* dal nodo.
- Raggiunto un nodo (branching), per ciascun nodo-figlio:
  - si determina il limite (bound) al valore delle possibili soluzioni che si possono determinare proseguendo per il figlio.

La tecnica Branch & Bound

- Si applica quando l'albero delle possibili soluzioni ad un problema algoritmico rappresenta soluzioni parziali o ammissibili per un problema di ottimizzazione. Ad ogni nodo possiamo calcolare un limite superiore e uno inferiore al valore delle soluzioni ammissibili *raggiungibili* dal nodo.
- Raggiunto un nodo (branching), per ciascun nodo-figlio:
  - si determina il limite (bound) al valore delle possibili soluzioni che si possono determinare proseguendo per il figlio.
  - se il limite indica che ogni soluzione possibile non sarà migliore di quella già determinata, non si esplora l'albero lungo quel figlio

Assunzioni per problemi di minimizzazione:

- la funzione costo $c()$ ha valori non negativi;

Assunzioni per problemi di minimizzazione:

- la funzione costo $c()$ ha valori non negativi;
- l'albero delle soluzioni è costituito da nodi che rappresentano elementi della soluzione e ha grado $\leq m$ e altezza n;

Assunzioni per problemi di minimizzazione:

- la funzione costo $c()$ ha valori non negativi;
- l'albero delle soluzioni è costituito da nodi che rappresentano elementi della soluzione e ha grado $\leq m$ e altezza n;
- possiamo **efficientemente** calcolare una funzione lower bound $LB(s[1,\ldots,j])$ che, in base al cammino nell'albero rappresentato dal vettore soluzione parziale $s[1,\ldots,j]$, per ogni $j = 1,\ldots,n$, restituisce il valore *minimo* delle soluzioni ammissibili che si possono generare a partire da $s[1,\ldots,j]$.

```
BranchBound(i)
1.  IF(i < n){ %i è l'indice del nodo corrente
```

```
BranchBound(i)
1.  IF(i < n){ %i è l'indice del nodo corrente
2.        Determina i nodi figli di i
```

```
BranchBound(i)
1.  IF(i < n){ %i è l'indice del nodo corrente
2.       Determina i nodi figli di i
3.       FOR(ogni nodo figlio j){
4.          s[i + 1] = j % la soluzione parziale nota è s[1, ..., i]
```

```
BranchBound(i)
1.  IF(i < n){ %i è l'indice del nodo corrente
2.       Determina i nodi figli di i
3.       FOR(ogni nodo figlio j){
4.          s[i + 1] = j % la soluzione parziale nota è s[1, ..., i]
5.          IF(LB(s[1, ..., i]) <MINCOST) BranchBound(i + 1)
```

```
BranchBound(i)
1.  IF(i < n){ %i è l'indice del nodo corrente
2.       Determina i nodi figli di i
3.       FOR(ogni nodo figlio j){
4.          s[i + 1] = j % la soluzione parziale nota è s[1, ..., i]
5.          IF(LB(s[1, ..., i]) <MINCOST) BranchBound(i + 1)
          }
6.  }ELSE{
7.       IF(c(s[1, ..., n]) < MINCOST) {
```

```
BranchBound(i)
1.  IF(i < n){ %i è l'indice del nodo corrente
2.       Determina i nodi figli di i
3.       FOR(ogni nodo figlio j){
4.          s[i + 1] = j % la soluzione parziale nota è s[1, ..., i]
5.          IF(LB(s[1, ..., i]) <MINCOST) BranchBound(i + 1)
          }
6.  }ELSE{
7.       IF(c(s[1, ..., n]) < MINCOST) {
8.          MINSOL = s[1, ..., n] % MINSOL=soluzione ottima
```

```
BranchBound(i)
1.  IF(i < n){ %i è l'indice del nodo corrente
2.       Determina i nodi figli di i
3.       FOR(ogni nodo figlio j){
4.          s[i + 1] = j % la soluzione parziale nota è s[1, ..., i]
5.          IF(LB(s[1, ..., i]) <MINCOST) BranchBound(i + 1)
          }
6.  }ELSE{
7.       IF(c(s[1, ..., n]) < MINCOST) {
8.          MINSOL = s[1, ..., n] % MINSOL=soluzione ottima
9.          MINCOST = c(s[1, ..., n]) % MINCOST=valore ottimo
     }
  }
```

```
BranchBound(i)
1.  IF(i < n){ %i è l'indice del nodo corrente
2.       Determina i nodi figli di i
3.       FOR(ogni nodo figlio j){
4.          s[i + 1] = j % la soluzione parziale nota è s[1, ..., i]
5.          IF(LB(s[1, ..., i]) <MINCOST) BranchBound(i + 1)
          }
6.  }ELSE{
7.       IF(c(s[1, ..., n]) < MINCOST) {
8.          MINSOL = s[1, ..., n] % MINSOL=soluzione ottima
9.          MINCOST = c(s[1, ..., n]) % MINCOST=valore ottimo
     }
  }
```

Complessità$=O(n^m f(n)^m)$ dove m è il grado nell'albero delle soluzione è $f(n)$ è il tempo necessario per calcolare $LB(s[1, \ldots, i])$

In pratica, il tempo di computazione è influenzato da diversi fattori:

- *Calcolo lower (upper) bound*.

In pratica, il tempo di computazione è influenzato da diversi fattori:

- *Calcolo lower (upper) bound*.
  Più la funzione di bound è precisa, meno parti di albero si devono esplorare. Un buon bound può però richiedere un tempo di calcolo non trascurabile.

In pratica, il tempo di computazione è influenzato da diversi fattori:

- *Calcolo lower (upper) bound*.
  Più la funzione di bound è precisa, meno parti di albero si devono esplorare. Un buon bound può però richiedere un tempo di calcolo non trascurabile.
- *Ordine nodi figli da visitare*.

In pratica, il tempo di computazione è influenzato da diversi fattori:

- *Calcolo lower (upper) bound*.
  Più la funzione di bound è precisa, meno parti di albero si devono esplorare. Un buon bound può però richiedere un tempo di calcolo non trascurabile.
- *Ordine nodi figli da visitare*. L'ordine più usato per visitare i figli è il LIFO perché richiede meno spazio. Ci sono altri ordini...che possono portare ad esplorare meno rami.

Sulla scelta dell' ordine di esplorazione dei nodi figli

- È possibile che i nodi-figli con bound migliore permettano di determinare soluzioni o ulteriori bound migliori e, quindi, tagliare ulteriormente altri rami prima che questi vengano visitati.

Sulla scelta dell' ordine di esplorazione dei nodi figli

- È possibile che i nodi-figli con bound migliore permettano di determinare soluzioni o ulteriori bound migliori e, quindi, tagliare ulteriormente altri rami prima che questi vengano visitati.
- Quindi, una volta selezionati i nodi-figli da esplorare, si inizia da quello con il bound migliore.

Sulla scelta dell' ordine di esplorazione dei nodi figli

- È possibile che i nodi-figli con bound migliore permettano di determinare soluzioni o ulteriori bound migliori e, quindi, tagliare ulteriormente altri rami prima che questi vengano visitati.
- Quindi, una volta selezionati i nodi-figli da esplorare, si inizia da quello con il bound migliore.
- Rimane da decidere se visitare poi in profondità o in ampiezza.

Sempre sulla scelta ordine di esplorazione dei nodi

- Se si usa il calcolo del bound in modo ancora più generale, il branch & bound può essere visto come ulteriore metodo di ricerca in un albero.

Sempre sulla scelta ordine di esplorazione dei nodi

- Se si usa il calcolo del bound in modo ancora più generale, il branch & bound può essere visto come ulteriore metodo di ricerca in un albero.
- I nodi vengono visitati secondo l'ordine stabilito dalla funzione bound sui valori contenuti nei sottoalberi.

Sempre sulla scelta ordine di esplorazione dei nodi

- Se si usa il calcolo del bound in modo ancora più generale, il branch & bound può essere visto come ulteriore metodo di ricerca in un albero.
- I nodi vengono visitati secondo l'ordine stabilito dalla funzione bound sui valori contenuti nei sottoalberi.

Se volessimo usare come ordine quello relativo alle soluzioni "migliori", dovremmo usare una coda a priorità (heap)

Sempre sulla scelta ordine di esplorazione dei nodi

- Se si usa il calcolo del bound in modo ancora più generale, il branch & bound può essere visto come ulteriore metodo di ricerca in un albero.
- I nodi vengono visitati secondo l'ordine stabilito dalla funzione bound sui valori contenuti nei sottoalberi.

Se volessimo usare come ordine quello relativo alle soluzioni "migliori", dovremmo usare una coda a priorità (heap) (ricordiamo che otteniamo invece una BFS se usiamo una normale coda,

Sempre sulla scelta ordine di esplorazione dei nodi

- Se si usa il calcolo del bound in modo ancora più generale, il branch & bound può essere visto come ulteriore metodo di ricerca in un albero.
- I nodi vengono visitati secondo l'ordine stabilito dalla funzione bound sui valori contenuti nei sottoalberi.

Se volessimo usare come ordine quello relativo alle soluzioni "migliori", dovremmo usare una coda a priorità (heap) (ricordiamo che otteniamo invece una BFS se usiamo una normale coda, e una DFS se usiamo uno stack per stabilire l'ordine con cui visitare i figli di un generico nodo).

Problema dell'assegnamento

Input: Un insieme di n agenti, che devono essere assegnati a n compiti: un agente per ogni compito.

Problema dell'assegnamento

Input: Un insieme di n agenti, che devono essere assegnati a n compiti: un agente per ogni compito.
Una funzione costo c_{ij} che indica il costo dell'agente i per svolgere il compito j.

Problema dell'assegnamento

Input: Un insieme di n agenti, che devono essere assegnati a n compiti: un agente per ogni compito.
Una funzione costo c_{ij} che indica il costo dell'agente i per svolgere il compito j.

Output: L'assegnazione dei compiti di costo totale minimo

Esempio

Quattro agenti: a, b, c, d, quattro compiti: $1, 2, 3, 4$

Esempio

Quattro agenti: a, b, c, d, quattro compiti: $1, 2, 3, 4$

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Esempio

Quattro agenti: a, b, c, d, quattro compiti: $1, 2, 3, 4$

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Un semplice upper bound per la soluzione è la somma della diagonale di somma minima: $11 + 15 + 19 + 28 = 73$.

Esempio

Quattro agenti: a, b, c, d, quattro compiti: $1, 2, 3, 4$

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Un semplice upper bound per la soluzione è la somma della diagonale di somma minima: $11 + 15 + 19 + 28 = 73$. Un semplice lower bound è prendere il minimo di ogni colonna: $11 + 12 + 13 + 22 = 58$.

Esempio

Quattro agenti: a, b, c, d, quattro compiti: $1, 2, 3, 4$

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Un semplice upper bound per la soluzione è la somma della diagonale di somma minima: $11 + 15 + 19 + 28 = 73$. Un semplice lower bound è prendere il minimo di ogni colonna: $11 + 12 + 13 + 22 = 58$. Il costo della soluzione ottima sarà un valore in $\{58, 59, \ldots 72, 73\}$.

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente (altezza albero$=n$).

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente (altezza albero$=n$).
- Ad ogni livello, si considerano tutti i nodi che rappresentano un assegnamento ammissibile per l'agente associato al livello.

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente (altezza albero$=n$).
- Ad ogni livello, si considerano tutti i nodi che rappresentano un assegnamento ammissibile per l'agente associato al livello.
- Per ogni nodo si calcola il lower bound per le soluzioni che si possono ottenere se si proseguisse per quel nodo.

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente (altezza albero$=n$).
- Ad ogni livello, si considerano tutti i nodi che rappresentano un assegnamento ammissibile per l'agente associato al livello.
- Per ogni nodo si calcola il lower bound per le soluzioni che si possono ottenere se si proseguisse per quel nodo.
- Il lower bound indica se il nodo è da considerare:

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente (altezza albero=n).
- Ad ogni livello, si considerano tutti i nodi che rappresentano un assegnamento ammissibile per l'agente associato al livello.
- Per ogni nodo si calcola il lower bound per le soluzioni che si possono ottenere se si proseguisse per quel nodo.
- Il lower bound indica se il nodo è da considerare:
  - se è inferiore all'upper bound corrente, si considera;

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente (altezza albero=n).
- Ad ogni livello, si considerano tutti i nodi che rappresentano un assegnamento ammissibile per l'agente associato al livello.
- Per ogni nodo si calcola il lower bound per le soluzioni che si possono ottenere se si proseguisse per quel nodo.
- Il lower bound indica se il nodo è da considerare:
  - se è inferiore all'upper bound corrente, si considera;
  - altrimenti si scarta per sempre.

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente (altezza albero=n).
- Ad ogni livello, si considerano tutti i nodi che rappresentano un assegnamento ammissibile per l'agente associato al livello.
- Per ogni nodo si calcola il lower bound per le soluzioni che si possono ottenere se si proseguisse per quel nodo.
- Il lower bound indica se il nodo è da considerare:
  - se è inferiore all'upper bound corrente, si considera;
  - altrimenti si scarta per sempre.
- Fra tutti i nodi da considerare, si prosegue il cammino in depth-first search a partire dal nodo lower bound inferiore.

Per risolvere l'esempio con la tecnica branch & bound:

- Si deve determinare la soluzione in modo incrementale visitando l'albero delle soluzioni:
  - Si determina un intervallo iniziale per il costo della soluzione ottima;
  - Si inizia con un assegnamento vuoto (radice);
  - Ad ogni livello, si determina l'assegnamento del compito per un nuovo agente (altezza albero=n).
- Ad ogni livello, si considerano tutti i nodi che rappresentano un assegnamento ammissibile per l'agente associato al livello.
- Per ogni nodo si calcola il lower bound per le soluzioni che si possono ottenere se si proseguisse per quel nodo.
- Il lower bound indica se il nodo è da considerare:
  - se è inferiore all'upper bound corrente, si considera;
  - altrimenti si scarta per sempre.
- Fra tutti i nodi da considerare, si prosegue il cammino in depth-first search a partire dal nodo lower bound inferiore.
- Quando si giunge una soluzione finale, si aggiorna l'upper bound e si scartano tutti i nodi dell'albero che superano tale bound.

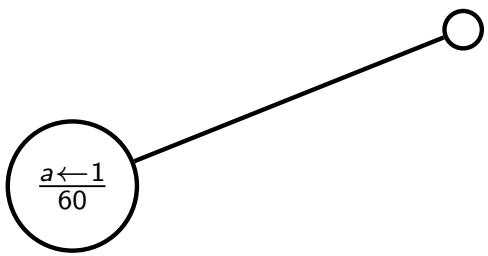
Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

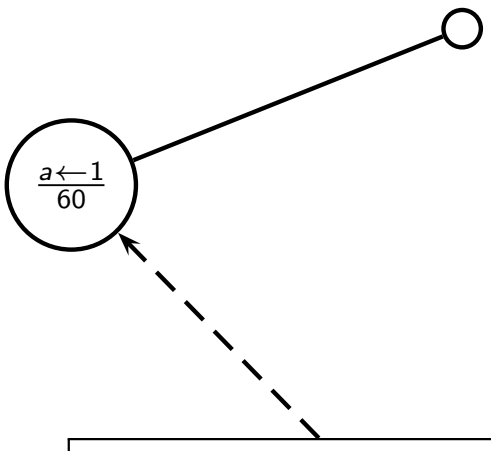
Se si parte da a, ci sono 4 possibili assegnamenti:



Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Se si parte da a, ci sono 4 possibili assegnamenti:

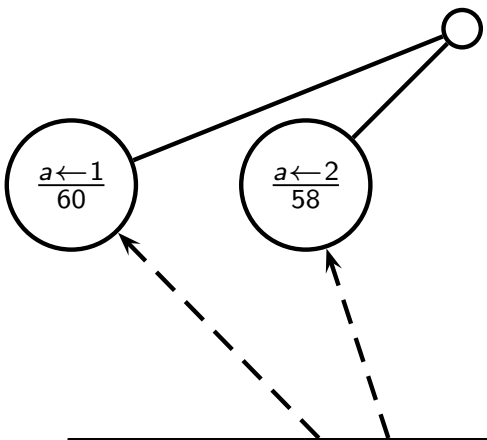


Costo di "a" + costo minimo in ciascuna delle colonne rimanenti private della riga "a":LB alla soluzione con lo specifico assegnamento ad a

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Se si parte da a, ci sono 4 possibili assegnamenti:

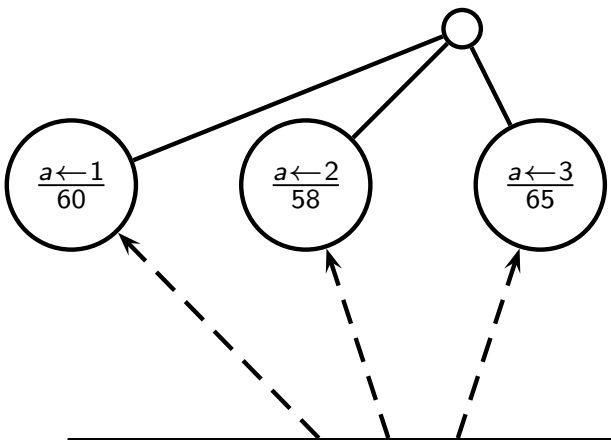


Costo di "a" + costo minimo in ciascuna delle colonne rimanenti private della riga "a":LB alla soluzione con lo specifico assegnamento ad a

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Se si parte da a, ci sono 4 possibili assegnamenti:

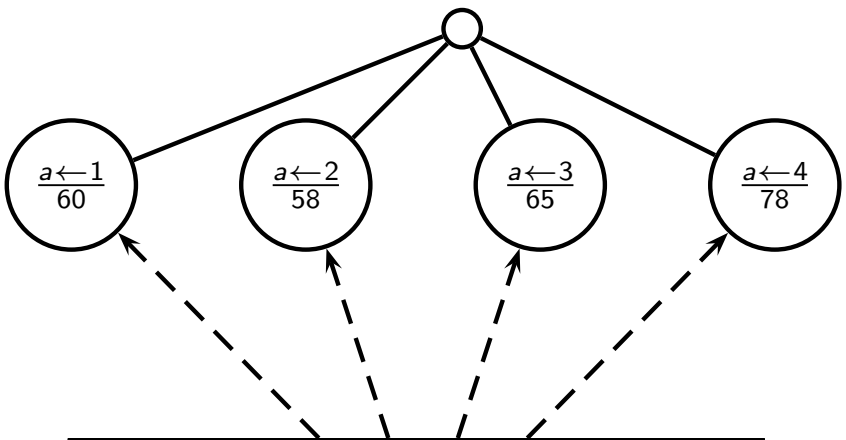


Costo di "a" + costo minimo in ciascuna delle colonne rimanenti private della riga "a":LB alla soluzione con lo specifico assegnamento ad a

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Se si parte da a, ci sono 4 possibili assegnamenti:

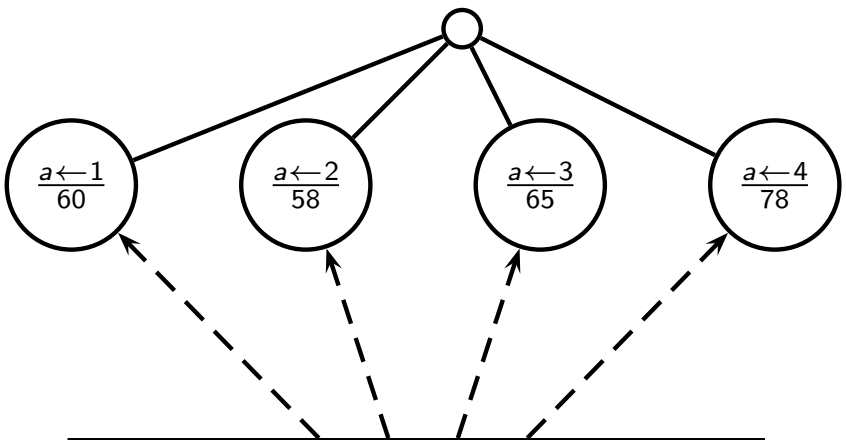


Costo di "a" + costo minimo in ciascuna delle colonne rimanenti private della riga "a":LB alla soluzione con lo specifico assegnamento ad a

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Se si parte da a, ci sono 4 possibili assegnamenti:



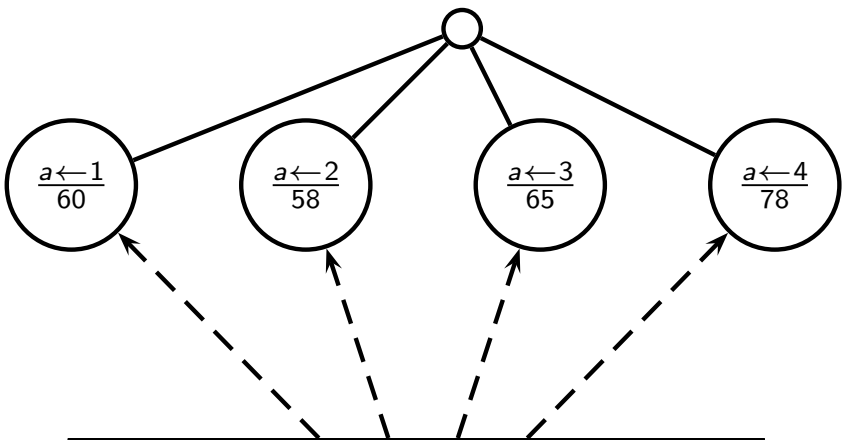
Costo di "a" + costo minimo in ciascuna delle colonne rimanenti private della riga "a":LB alla soluzione con lo specifico assegnamento ad a

- Considerato che la soluzione $a \leftarrow 4$ porterà a una soluzione con costo ≥ 78, maggiore dell'upper 73 già determinato, esse viene scartata.

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Se si parte da a, ci sono 4 possibili assegnamenti:



Costo di "a" + costo minimo in ciascuna delle colonne rimanenti private della riga "a":LB alla soluzione con lo specifico assegnamento ad a

- Considerato che la soluzione $a \leftarrow 4$ porterà a una soluzione con costo ≥ 78, maggiore dell'upper 73 già determinato, esse viene scartata.
- Il nodo $a \leftarrow 2$ ha il miglior lower bound: sembra quindi il più promettente da esplorare (non vi è certezza...)

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Livello 2: attività per b

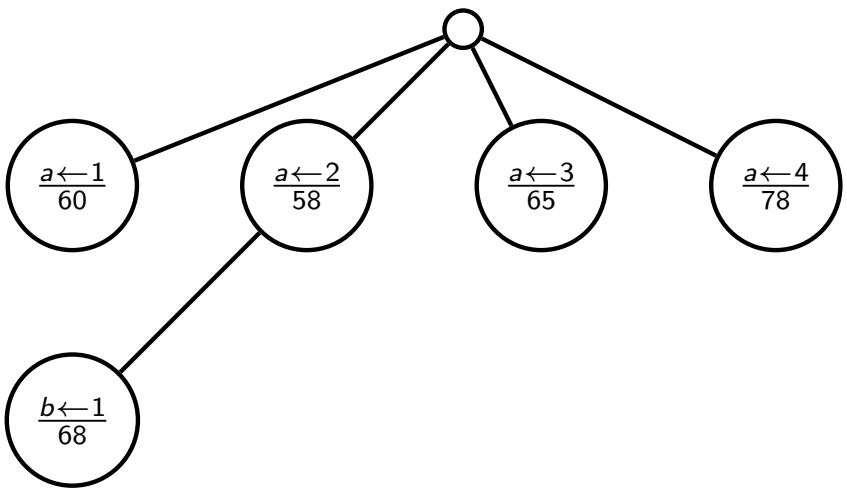
Ci sono 3 possibili assegnamenti

Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Livello 2: attività per b

Ci sono 3 possibili assegnamenti

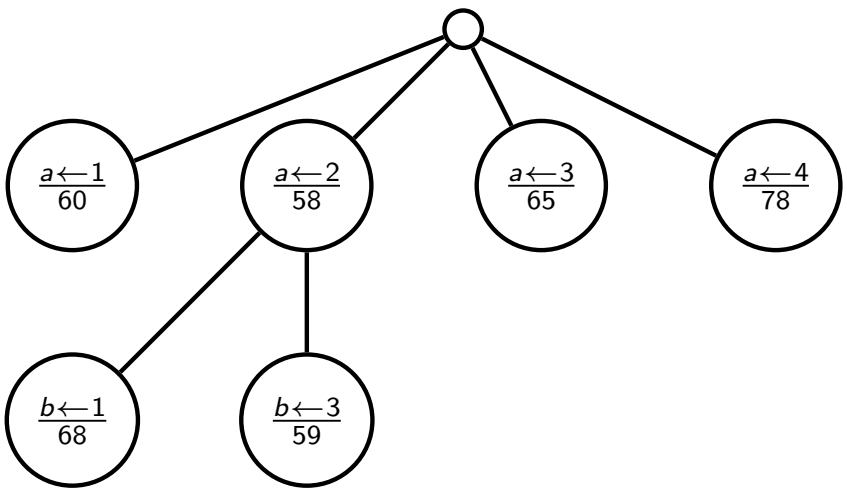


Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Livello 2: attività per b

Ci sono 3 possibili assegnamenti

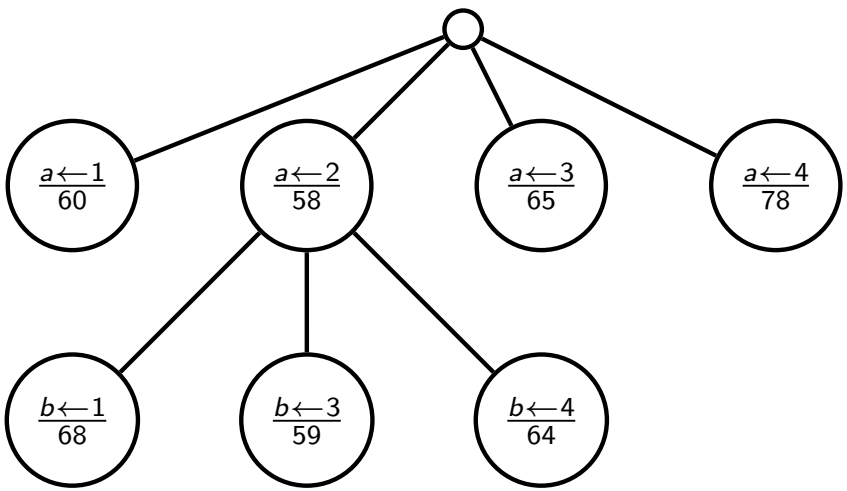


Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Livello 2: attività per b

Ci sono 3 possibili assegnamenti

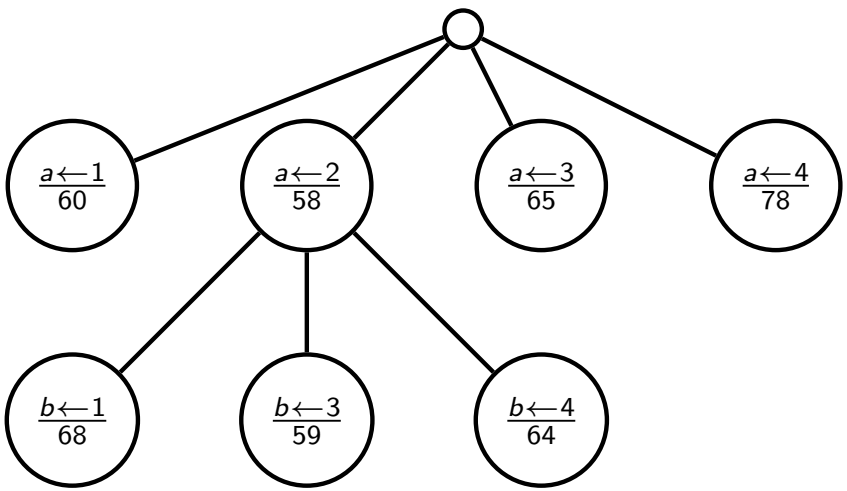


Funzione costo:

c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Livello 2: attività per b

Ci sono 3 possibili assegnamenti



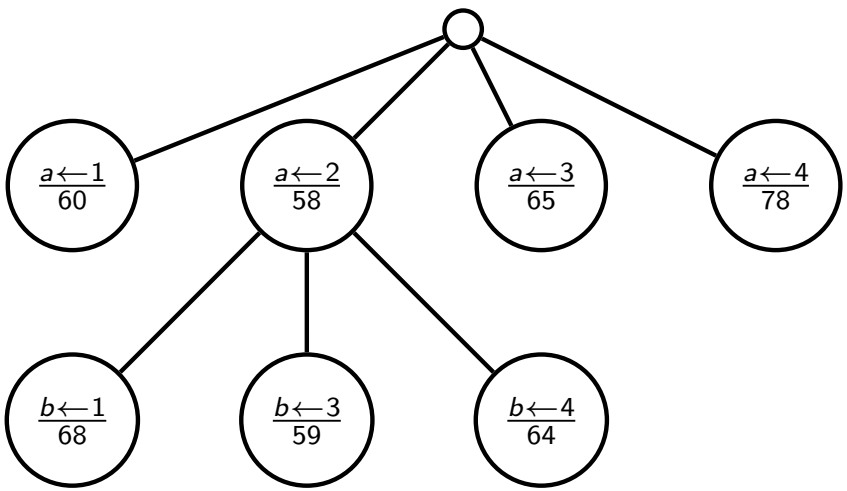
Miglior lower bound$=$ $b \leftarrow 3$, si prosegue per questo ramo

Funzione costo:

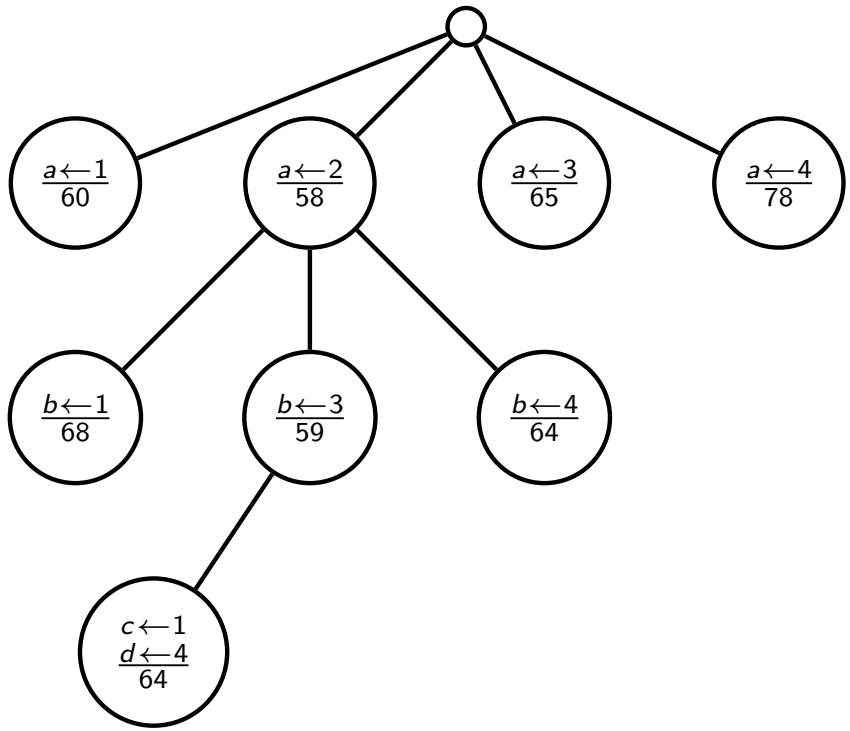
c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Livello 2: attività per b

Ci sono 3 possibili assegnamenti



Miglior lower bound$= b \leftarrow 3$, si prosegue per questo ramo

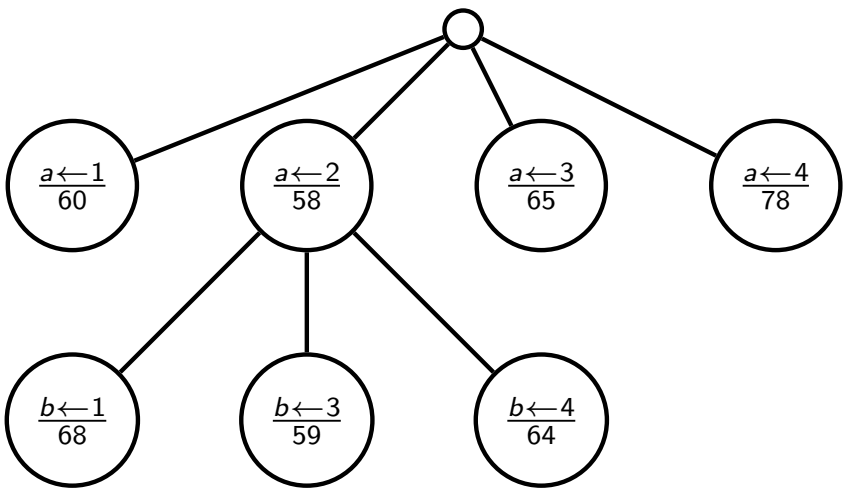


Funzione costo:

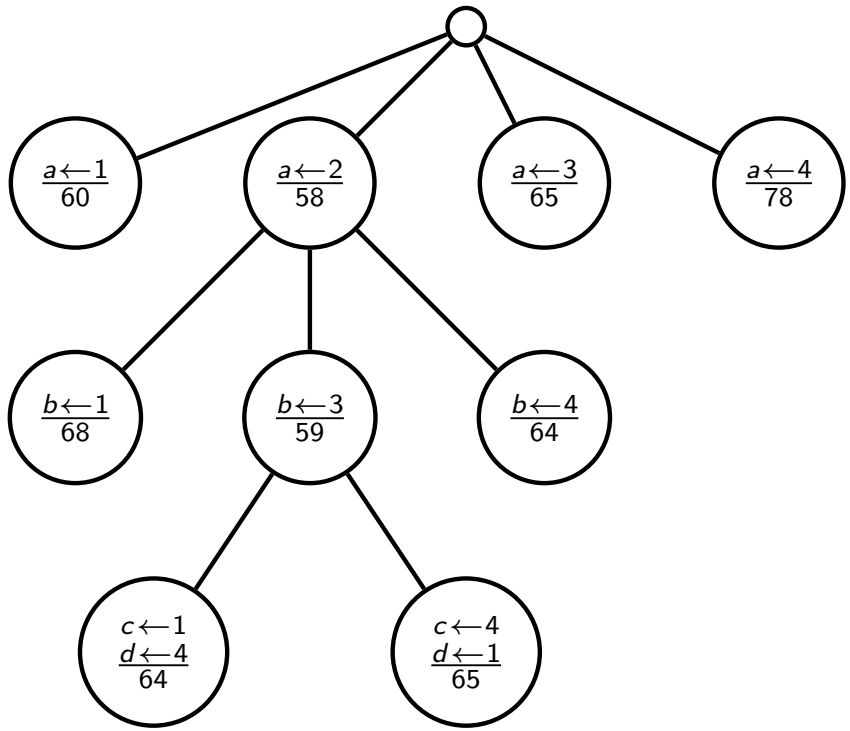
c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Livello 2: attività per b

Ci sono 3 possibili assegnamenti



Miglior lower bound= $b \leftarrow 3$, si prosegue per questo ramo

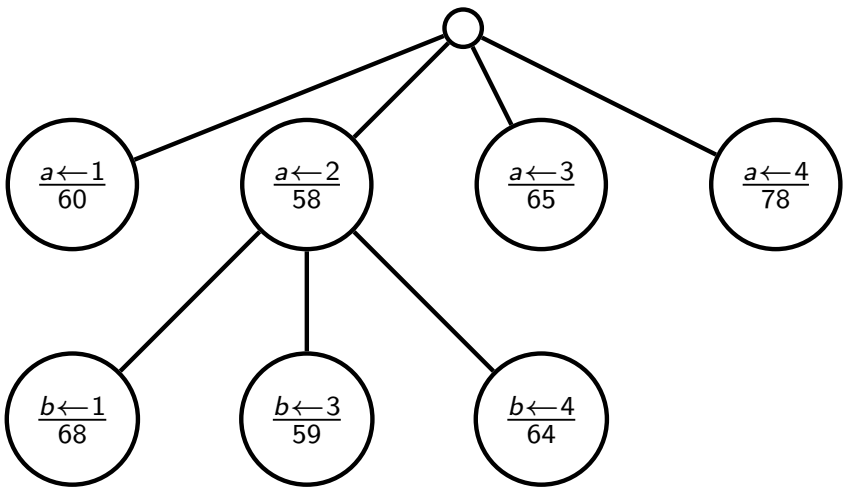


Funzione costo:

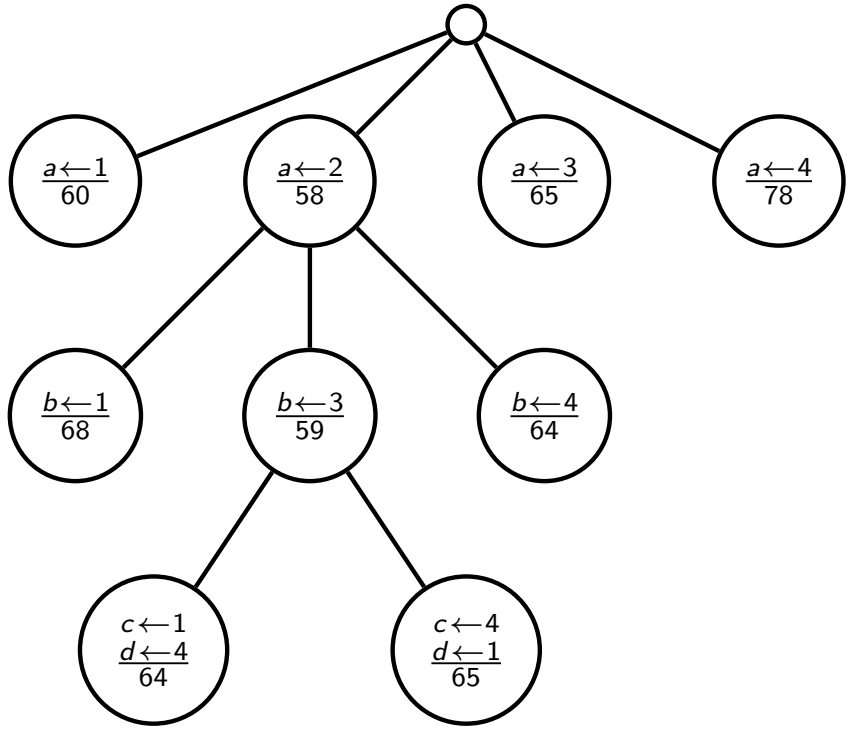
c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Livello 2: attività per b

Ci sono 3 possibili assegnamenti

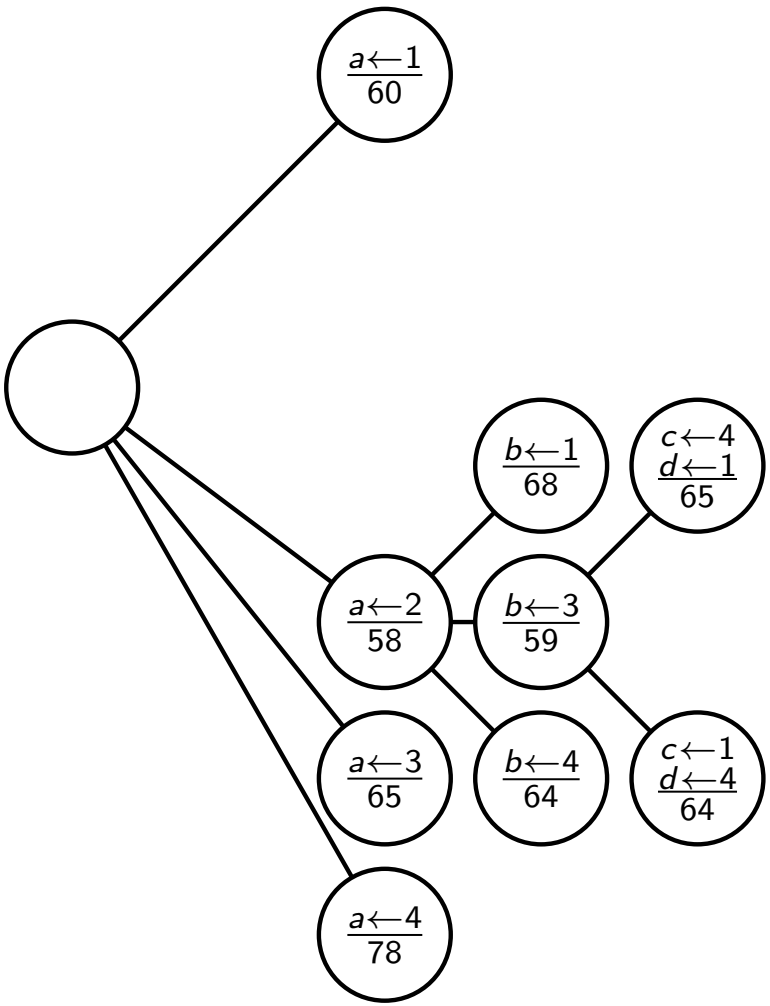


Miglior lower bound= $b \leftarrow 3$, si prosegue per questo ramo



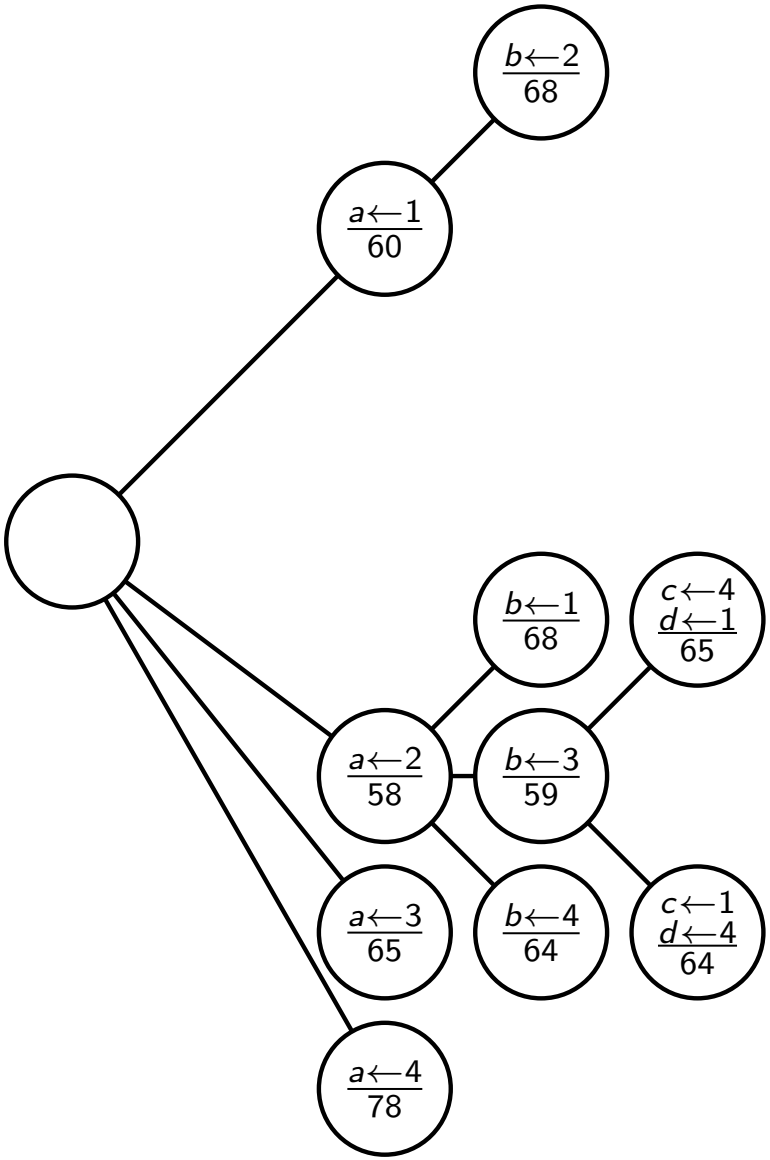
Il nuovo upper bound 64 al costo della soluzione ottima elimina molti rami dell'albero (tutti quelli che escono da nodi con LB$>$ 64).

Rimane da esplorare il nodo $a \leftarrow 1$



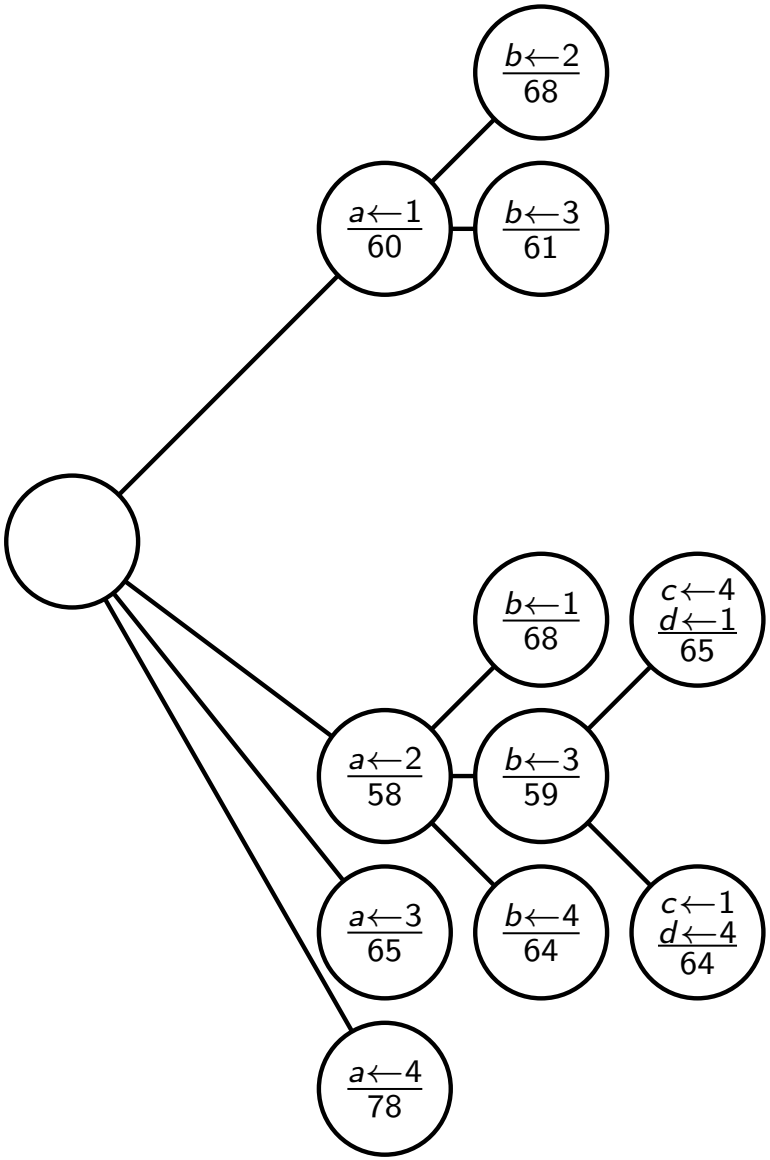
c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Rimane da esplorare il nodo $a \leftarrow 1$



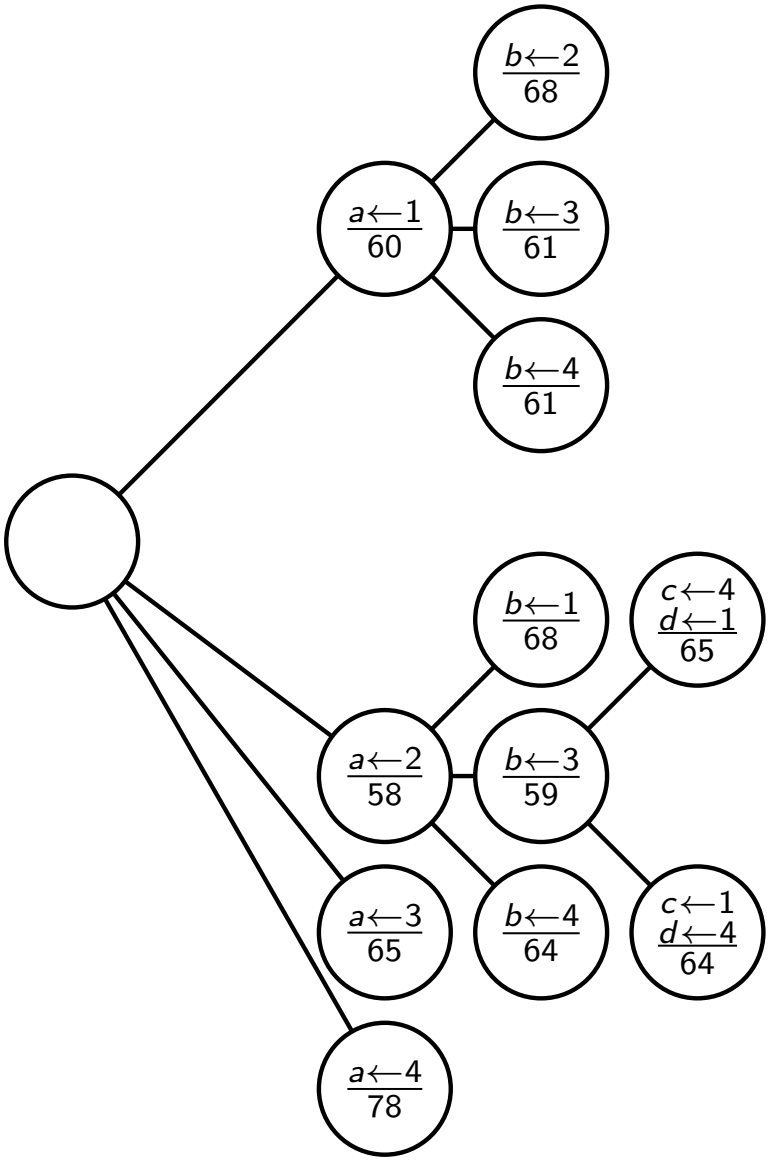
c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Rimane da esplorare il nodo $a \leftarrow 1$



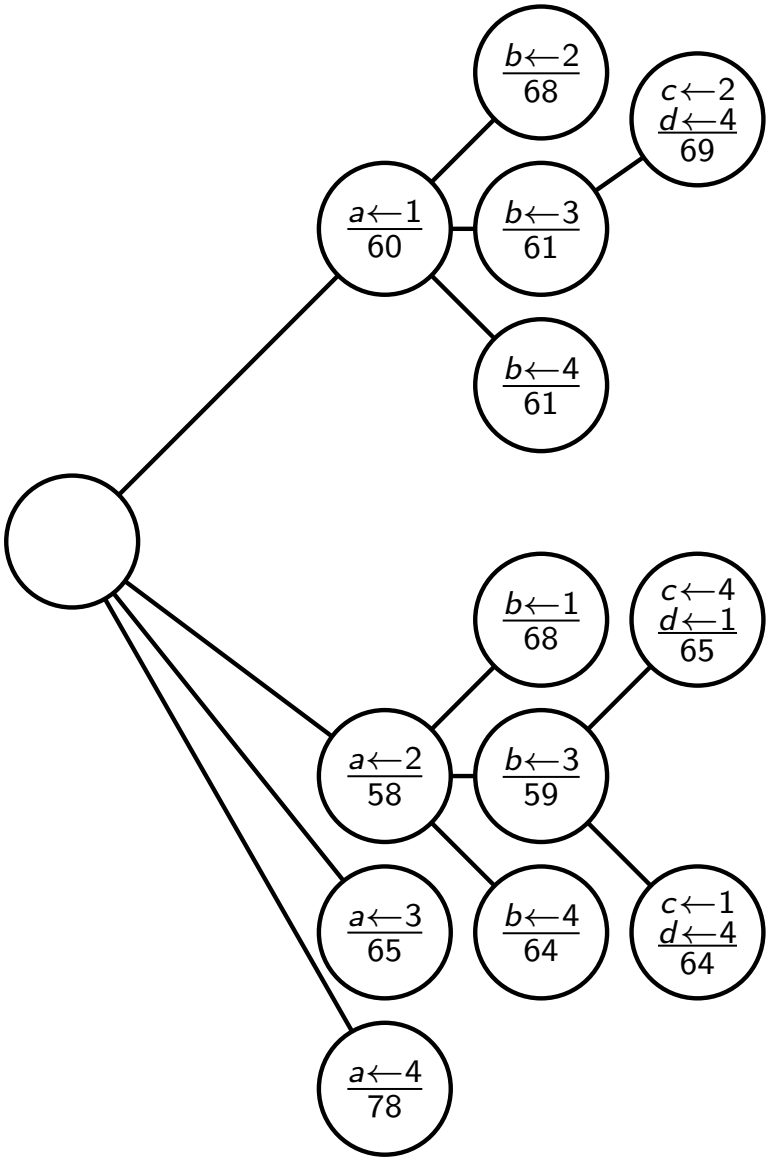
c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Rimane da esplorare il nodo $a \leftarrow 1$



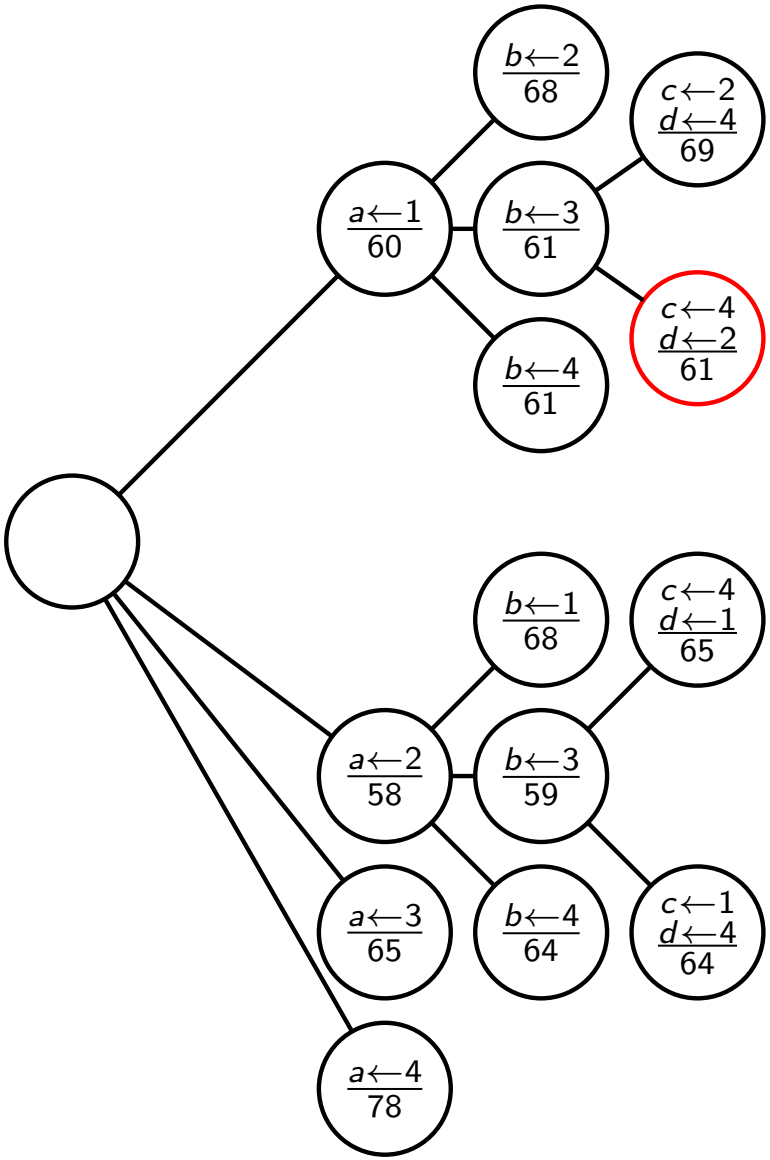
c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Rimane da esplorare il nodo $a \leftarrow 1$



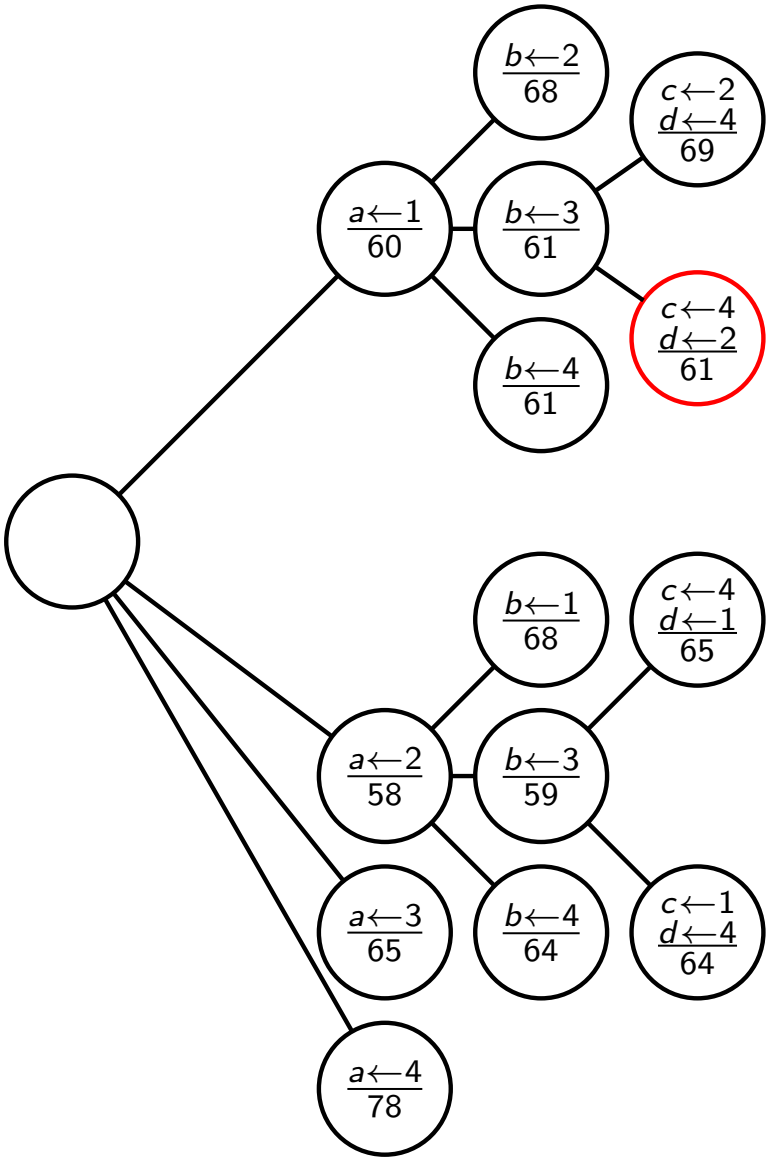
c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

Rimane da esplorare il nodo $a \leftarrow 1$



c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

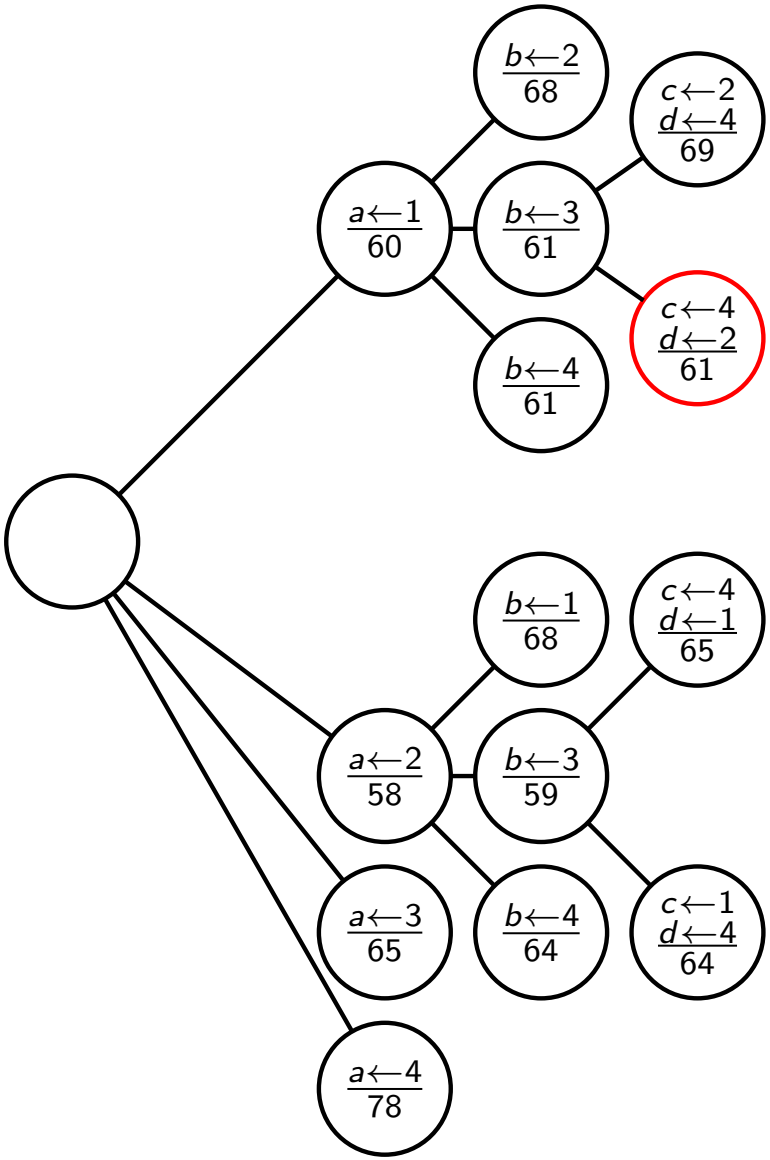
Rimane da esplorare il nodo $a \leftarrow 1$



c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

- La soluzione ottimale è $a = 1$, $b = 3$, $c = 4$, $d = 2$, con valore 61.

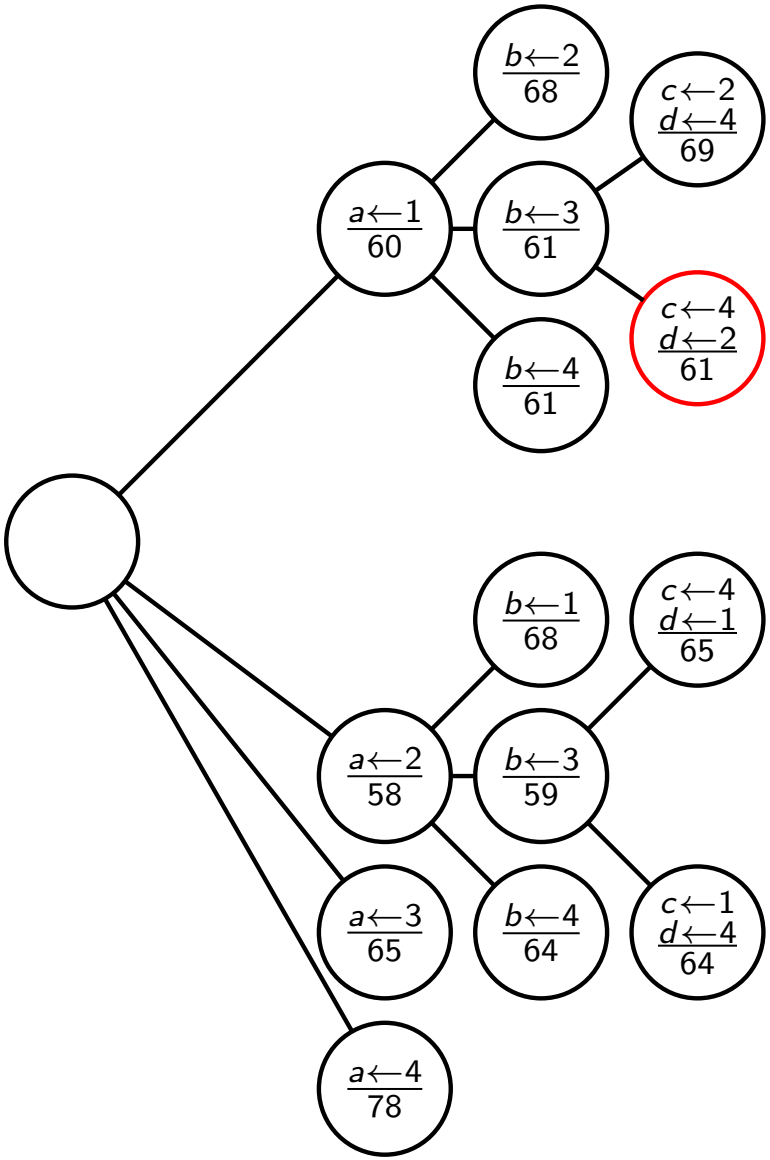
Rimane da esplorare il nodo $a \leftarrow 1$



c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

- La soluzione ottimale è $a = 1$, $b = 3$, $c = 4$, $d = 2$, con valore 61.
- Sono stati valutati 15 nodi su 40 possibili

Rimane da esplorare il nodo $a \leftarrow 1$



c_{ij}	1	2	3	4
a	11	12	18	40
b	14	15	13	22
c	11	17	19	23
d	17	14	20	28

- La soluzione ottimale è $a = 1$, $b = 3$, $c = 4$, $d = 2$, con valore 61.
- Sono stati valutati 15 nodi su 40 possibili
- Soluzioni possibili, solo 6 sono state esaminate: 4 dell'albero e 2 iniziali (per il lower bound)

Problema dello Zaino

Input: Un insieme di n di oggetti, ciascuno con un valore v_i e da un peso p_i. Un intero positivo P che misura la portata massima dello zaino.

Problema dello Zaino

Input: Un insieme di n di oggetti, ciascuno con un valore v_i e da un peso p_i. Un intero positivo P che misura la portata massima dello zaino.

Output: Una assegnazione $\{x_1, x_2, \ldots, x_n\}$ di variabili, con $x_i = \{0, 1\}$, tale che il valore totale $\sum_{i=1}^{n} x_i v_i$ sia massimo e per cui il peso totale $\sum_{i=1}^{n} x_i p_i \leq P$

Problema dello Zaino

Input: Un insieme di n di oggetti, ciascuno con un valore v_i e da un peso p_i. Un intero positivo P che misura la portata massima dello zaino.

Output: Una assegnazione $\{x_1, x_2, \ldots, x_n\}$ di variabili, con $x_i = \{0, 1\}$, tale che il valore totale $\sum_{i=1}^{n} x_i v_i$ sia massimo e per cui il peso totale $\sum_{i=1}^{n} x_i p_i \leq P$

La variabile x_i viene posta a 1 se e solo l'oggetto i viene messo nella soluzione, viene posta a 0 se l'oggetto i non viene messo nella soluzione

Problema dello Zaino

Input: Un insieme di n di oggetti, ciascuno con un valore v_i e da un peso p_i. Un intero positivo P che misura la portata massima dello zaino.

Output: Una assegnazione $\{x_1, x_2, \ldots, x_n\}$ di variabili, con $x_i = \{0, 1\}$, tale che il valore totale $\sum_{i=1}^{n} x_i v_i$ sia massimo e per cui il peso totale $\sum_{i=1}^{n} x_i p_i \leq P$

La variabile x_i viene posta a 1 se e solo l'oggetto i viene messo nella soluzione, viene posta a 0 se l'oggetto i non viene messo nella soluzione

Senza perdere in generalità, si assume gli oggetti siano ordinati in modo che $\frac{v_i}{p_i} \geq \frac{v_{i+1}}{p_{i+1}}, i = 1, \ldots, n-1$.

Osservazione: Data una soluzione parziale determinata fino alla k-esima componente $x_1, ..., x_k$, con $k < n$, una limitazione superiore (upper bound) per la soluzione finale è

$$\sum_{i=1}^{k} x_i v_i$$

Osservazione: Data una soluzione parziale determinata fino alla k-esima componente $x_1, ..., x_k$, con $k < n$, una limitazione superiore (upper bound) per la soluzione finale è

$$\sum_{i=1}^{k} x_i v_i + \left(P - \sum_{i=1}^{k} x_i p_i \right) \cdot \frac{v_{k+1}}{p_{k+1}}$$

Osservazione: Data una soluzione parziale determinata fino alla k-esima componente $x_1, ..., x_k$, con $k < n$, una limitazione superiore (upper bound) per la soluzione finale è

$$\sum_{i=1}^{k} x_i v_i + \left(P - \sum_{i=1}^{k} x_i p_i \right) \cdot \frac{v_{k+1}}{p_{k+1}}$$

Infatti, $(P - \sum_{i=1}^{k} x_i p_i)$ rappresenta il peso che possiamo ancora trasportare, dato che abbiamo già messo nello zaino un peso pari a $\sum_{i=1}^{k} x_i p_i$

Osservazione: Data una soluzione parziale determinata fino alla k-esima componente $x_1, ..., x_k$, con $k < n$, una limitazione superiore (upper bound) per la soluzione finale è

$$\sum_{i=1}^{k} x_i v_i + \left(P - \sum_{i=1}^{k} x_i p_i \right) \cdot \frac{v_{k+1}}{p_{k+1}}$$

Infatti, $(P - \sum_{i=1}^{k} x_i p_i)$ rappresenta il peso che possiamo ancora trasportare, dato che abbiamo già messo nello zaino un peso pari a $\sum_{i=1}^{k} x_i p_i$ e l'oggetto $k+1$ è quello che ha il rapporto valore/peso maggiore di tutti gli altri oggetti.

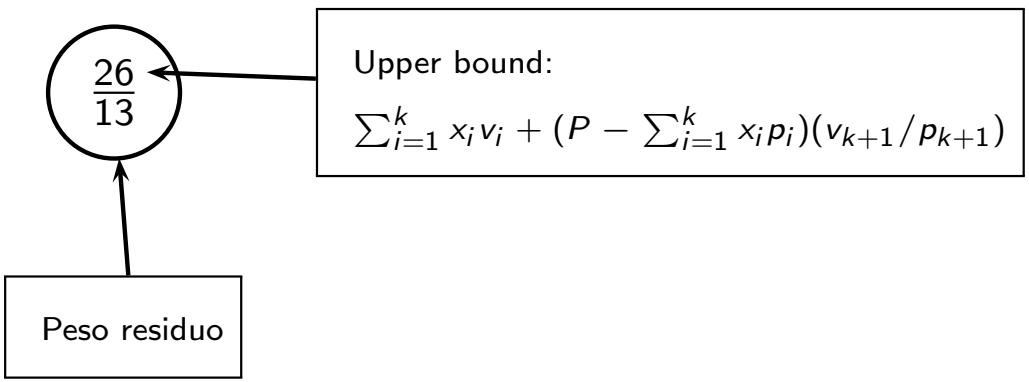
Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

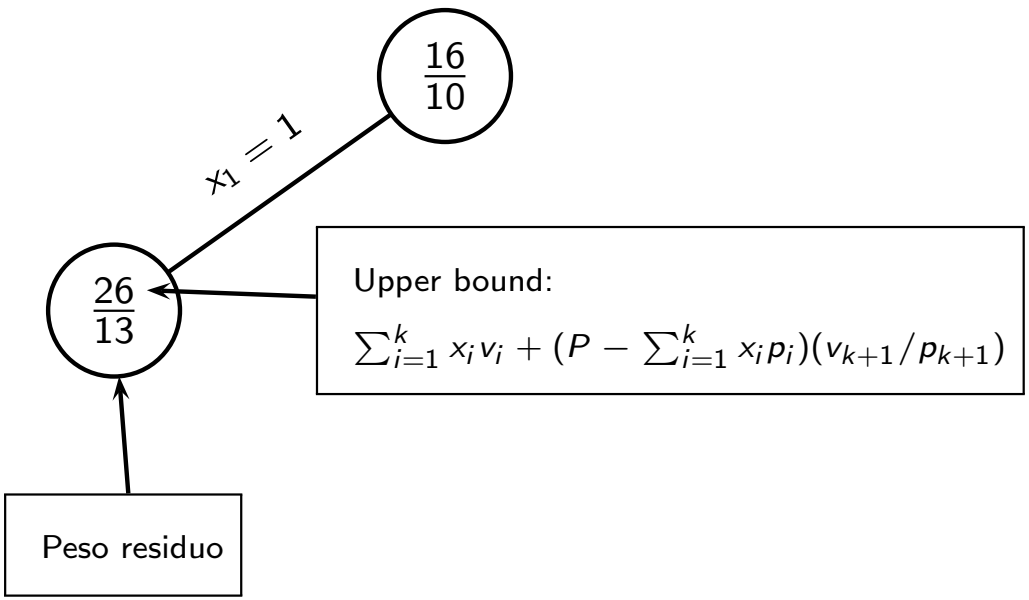
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

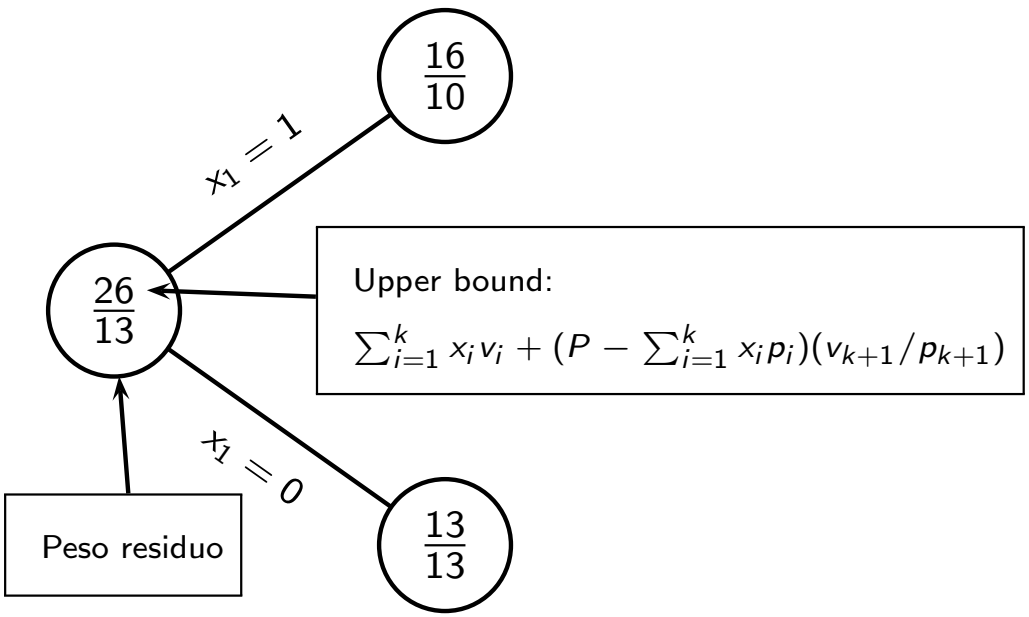
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

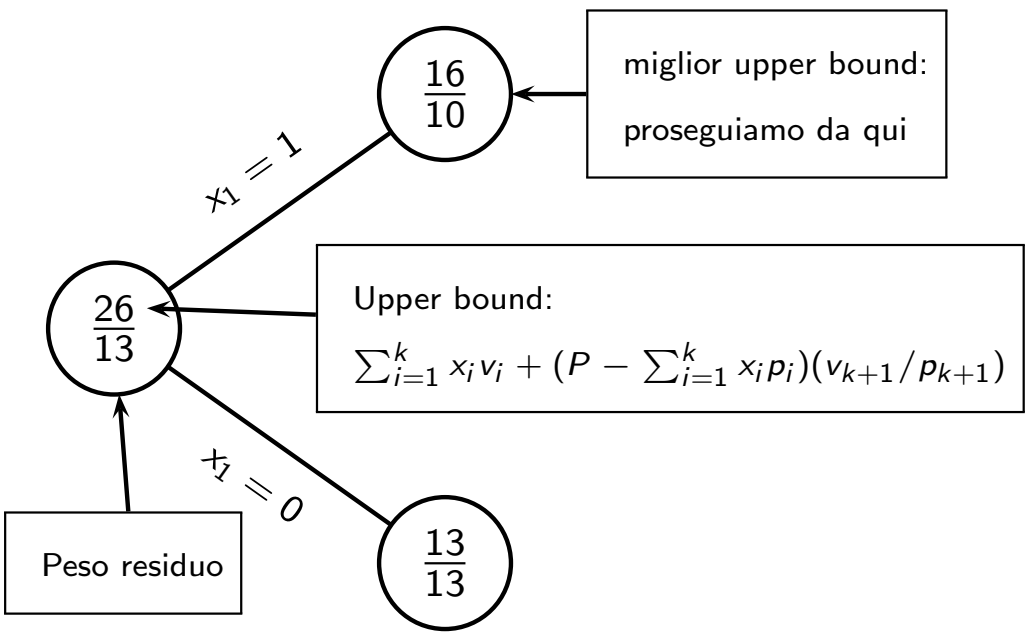
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

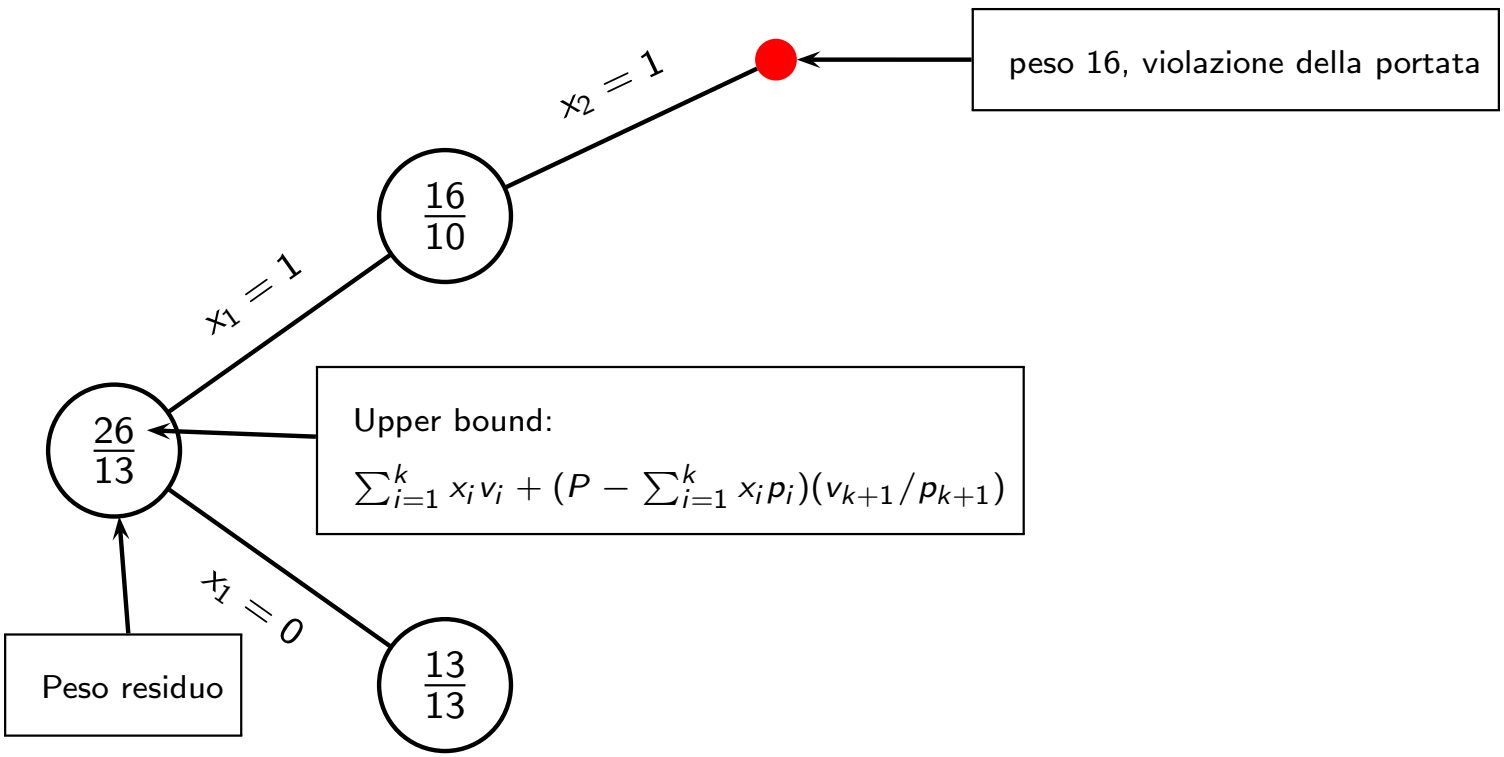
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

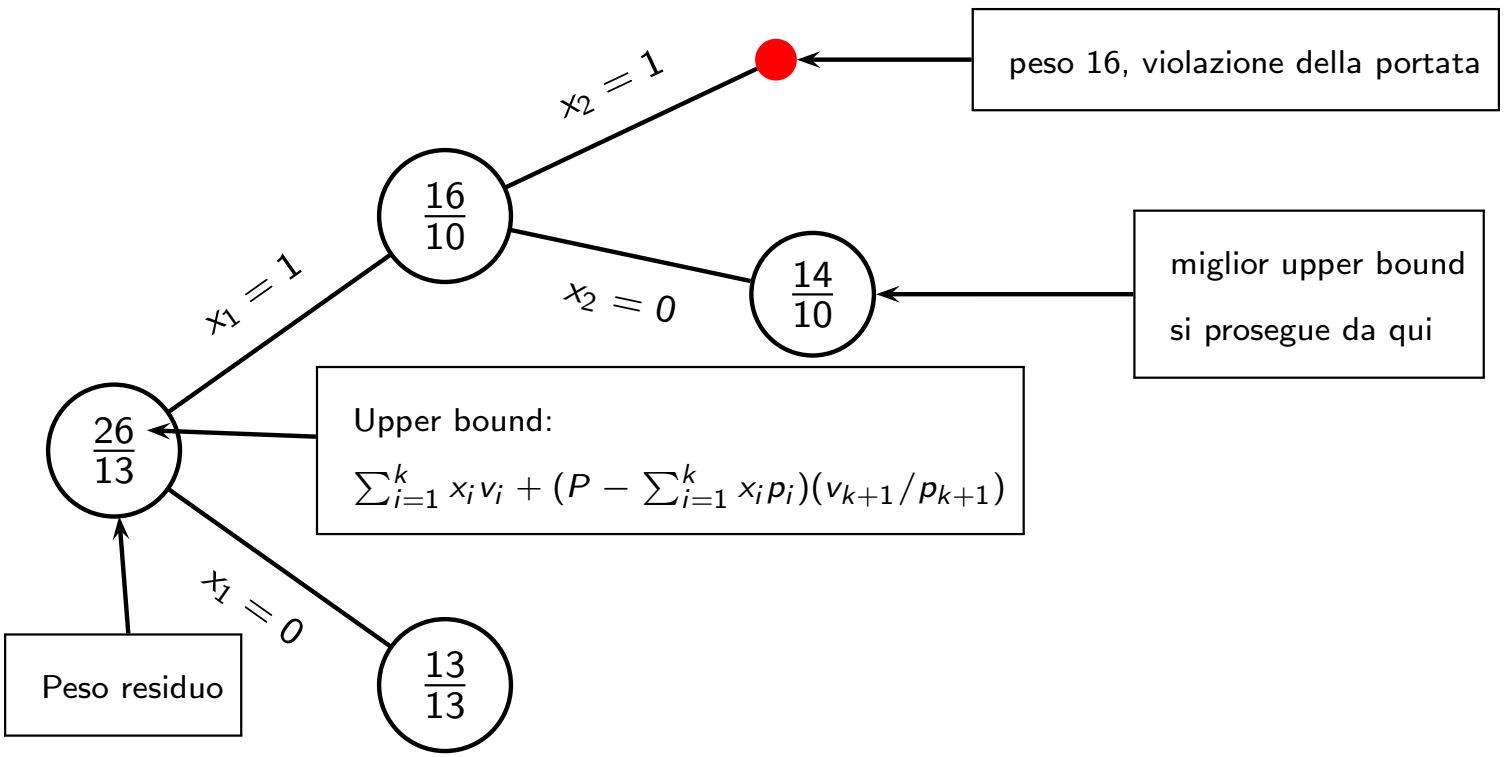
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

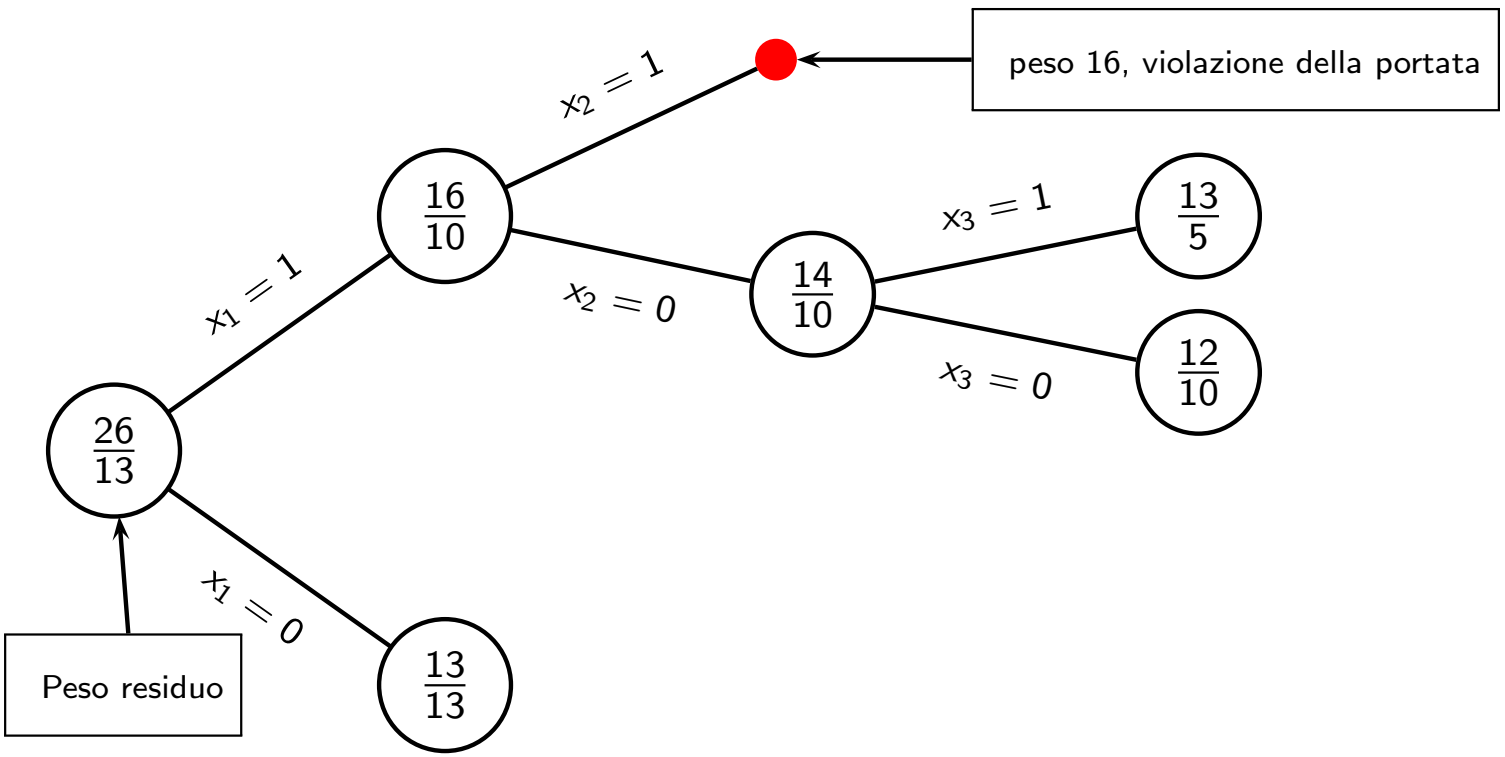
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

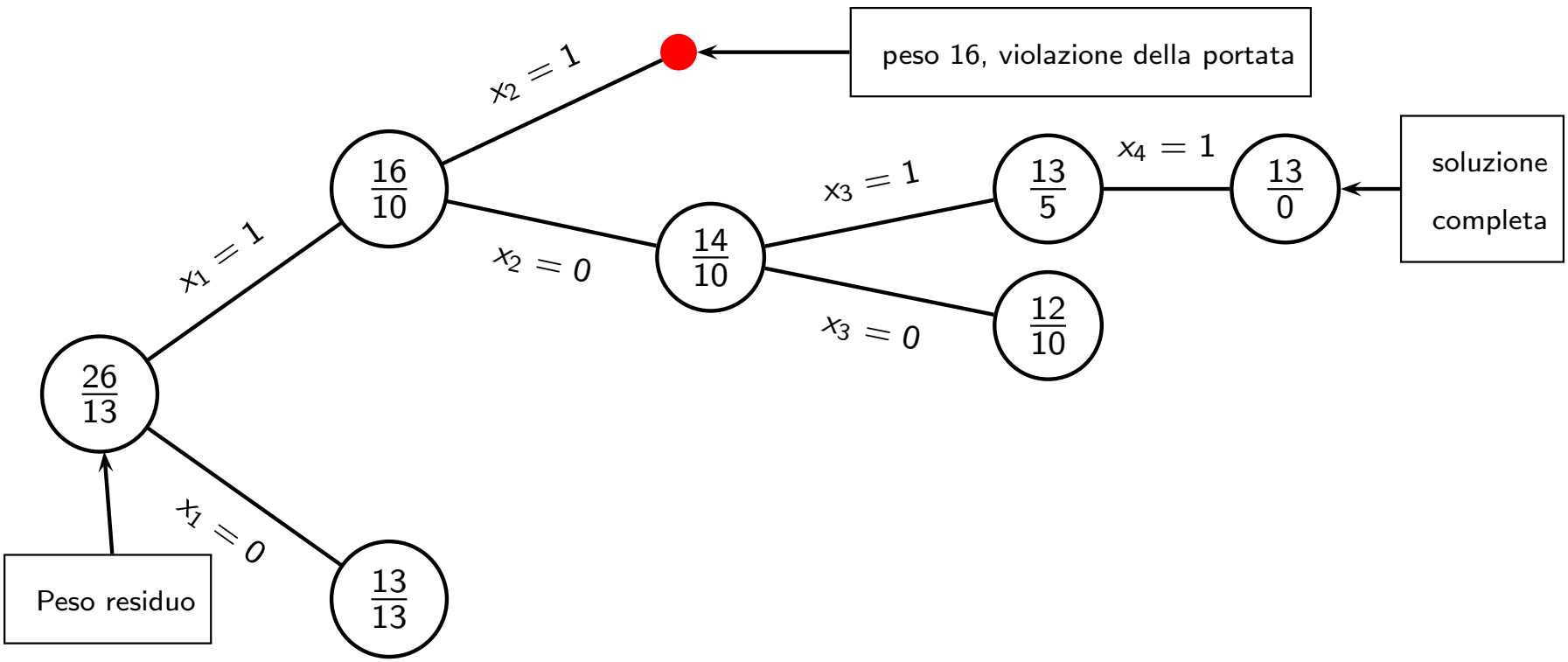
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

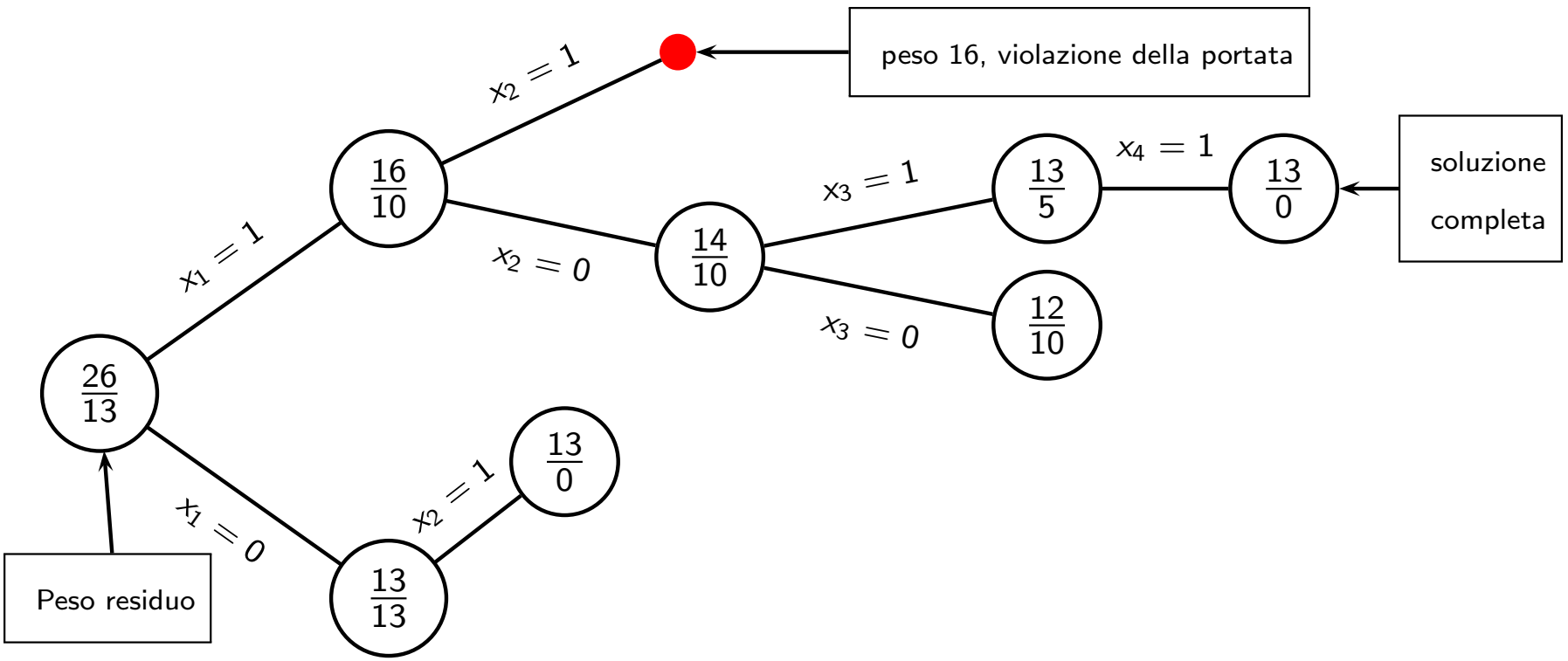
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

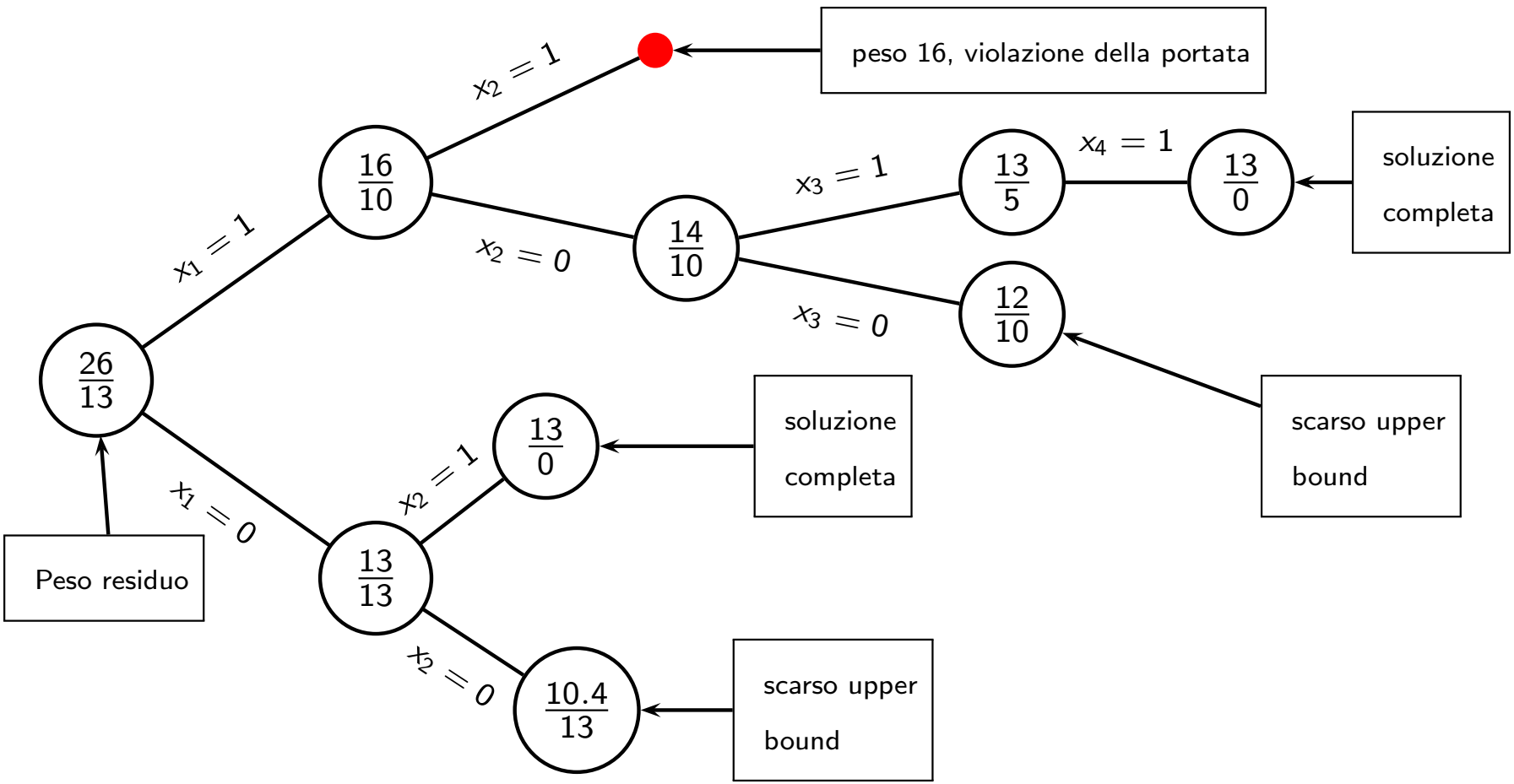
Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Esempio

$v = [6, 13, 4, 3]$, $p = [3, 13, 5, 5]$, $P = 13$, $\frac{v_i}{p_i} = [2, 1, 0.8, 0.6]$.

Al livello $i - 1$ consideriamo se inserire o meno l'oggetto i nella soluzione (cioè se porre $x_i = 1$, per $i = 1, \ldots, 4$)



Problema del Commesso Viaggiatore

Input: Un grafo $G = (V, E)$ non diretto *completo* (cioè $\forall u, v \in V, u \neq v$ vale che $(u, v) \in E$) di n vertici e una funzione $d : E \to N$, distanza tra i vertici.

Problema del Commesso Viaggiatore

Input: Un grafo $G = (V, E)$ non diretto *completo* (cioè $\forall u, v \in V, u \neq v$ vale che $(u, v) \in E$) di n vertici e una funzione $d : E \to N$, distanza tra i vertici.

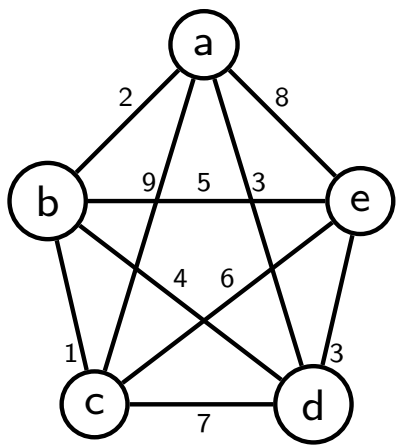
Output: Un ciclo che passi per ogni vertice *una ed una sola volta*, di lunghezza totale minimo.

Problema del Commesso Viaggiatore

Input: Un grafo $G = (V, E)$ non diretto *completo* (cioè $\forall u, v \in V, u \neq v$ vale che $(u, v) \in E$) di n vertici e una funzione $d : E \to N$, distanza tra i vertici.

Output: Un ciclo che passi per ogni vertice *una ed una sola volta*, di lunghezza totale minimo.

Esempio:

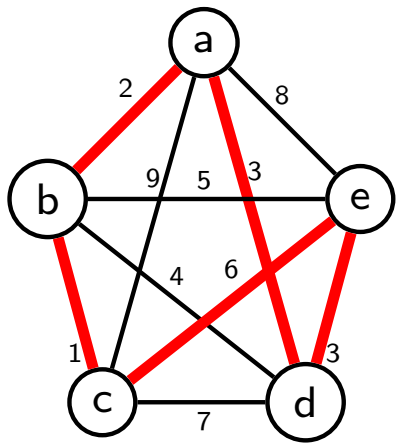


Problema del Commesso Viaggiatore

Input: Un grafo $G = (V, E)$ non diretto *completo* (cioè $\forall u, v \in V, u \neq v$ vale che $(u, v) \in E$) di n vertici e una funzione $d : E \to N$, distanza tra i vertici.

Output: Un ciclo che passi per ogni vertice *una ed una sola volta*, di lunghezza totale minimo.

Esempio: ciclo di lunghezza 15



Come determinare una limitazione inferiore

Come determinare una limitazione inferiore

Supponiamo di avere una soluzione parziale, ovvero un cammino α fatto di vertici $u_1, u_2, \ldots u_k$ (distinti tra di loro) di lunghezza

$$d(\alpha) = d(u_1, u_2) + \ldots + d(u_{k-1}, u_k).$$

Come determinare una limitazione inferiore

Supponiamo di avere una soluzione parziale, ovvero un cammino α fatto di vertici $u_1, u_2, \ldots u_k$ (distinti tra di loro) di lunghezza

$$d(\alpha) = d(u_1, u_2) + \ldots + d(u_{k-1}, u_k).$$

Per essere completato in un ciclo che attraversa tutti i vertici di V, il cammino deve essere seguito da un altro cammino β fatto di vertici $u_k, u_{k+1}, \ldots, u_1$.

Come determinare una limitazione inferiore

Supponiamo di avere una soluzione parziale, ovvero un cammino α fatto di vertici $u_1, u_2, \ldots u_k$ (distinti tra di loro) di lunghezza

$$d(\alpha) = d(u_1, u_2) + \ldots + d(u_{k-1}, u_k).$$

Per essere completato in un ciclo che attraversa tutti i vertici di V, il cammino deve essere seguito da un altro cammino β fatto di vertici $u_k, u_{k+1}, \ldots, u_1$.

Osserviamo che β è di fatto un albero che copre tutti i vertici in $V \setminus \{u_2, \ldots, u_{k-1}\}$.

Come determinare una limitazione inferiore

Supponiamo di avere una soluzione parziale, ovvero un cammino α fatto di vertici $u_1, u_2, \ldots u_k$ (distinti tra di loro) di lunghezza

$$d(\alpha) = d(u_1, u_2) + \ldots + d(u_{k-1}, u_k).$$

Per essere completato in un ciclo che attraversa tutti i vertici di V, il cammino deve essere seguito da un altro cammino β fatto di vertici $u_k, u_{k+1}, \ldots, u_1$.

Osserviamo che β è di fatto un albero che copre tutti i vertici in $V \setminus \{u_2, \ldots, u_{k-1}\}$.

Se interpretiamo la funzione distanza come una funzione costo, vale che $d(\beta) \geq$ costo del MST sui vertici in $V \setminus \{u_2, \ldots, u_{k-1}\}$ (che sappiamo calcolare).

Come determinare una limitazione inferiore

Supponiamo di avere una soluzione parziale, ovvero un cammino α fatto di vertici $u_1, u_2, \ldots u_k$ (distinti tra di loro) di lunghezza

$$d(\alpha) = d(u_1, u_2) + \ldots + d(u_{k-1}, u_k).$$

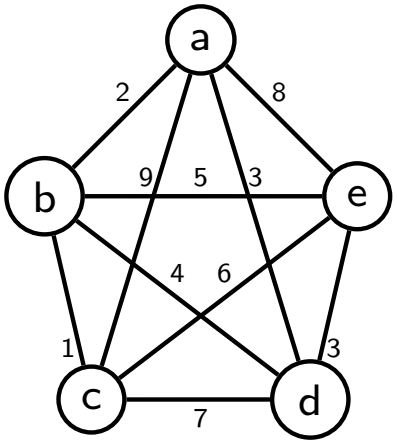
Per essere completato in un ciclo che attraversa tutti i vertici di V, il cammino deve essere seguito da un altro cammino β fatto di vertici $u_k, u_{k+1}, \ldots, u_1$.

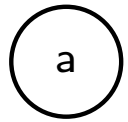
Osserviamo che β è di fatto un albero che copre tutti i vertici in $V \setminus \{u_2, \ldots, u_{k-1}\}$.

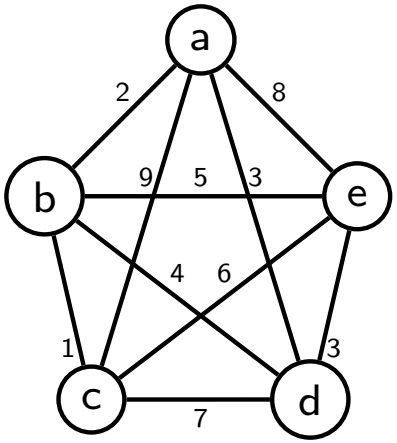
Se interpretiamo la funzione distanza come una funzione costo, vale che $d(\beta) \geq$ costo del MST sui vertici in $V \setminus \{u_2, \ldots, u_{k-1}\}$ (che sappiamo calcolare).

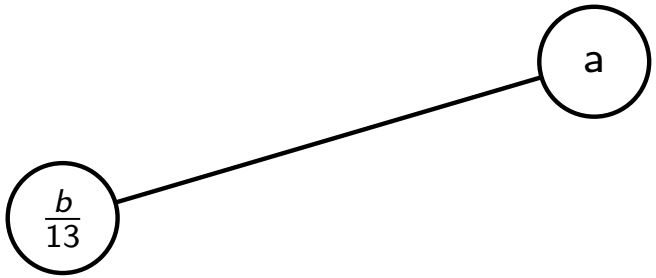
Quindi, il "miglior" ciclo che possiamo costruire a partire dalla soluzione parziale α ha necessariamente lunghezza almeno pari a

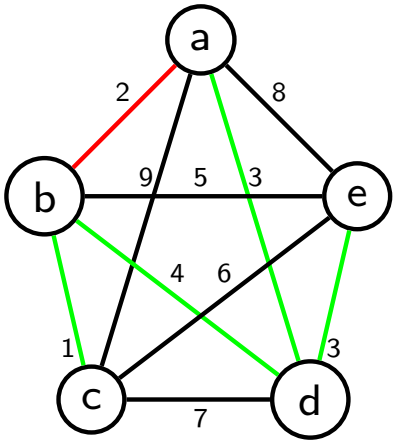
$$d(\alpha) + \text{costo del MST sui vertici in } V \setminus \{u_2, \ldots, u_{k-1}\}$$

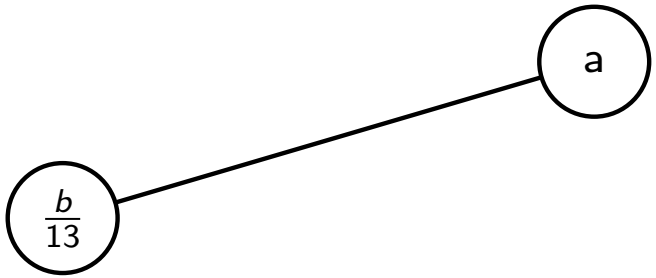

a
2
8
b
9
5
3
e
4
6
1
3
c
d
7

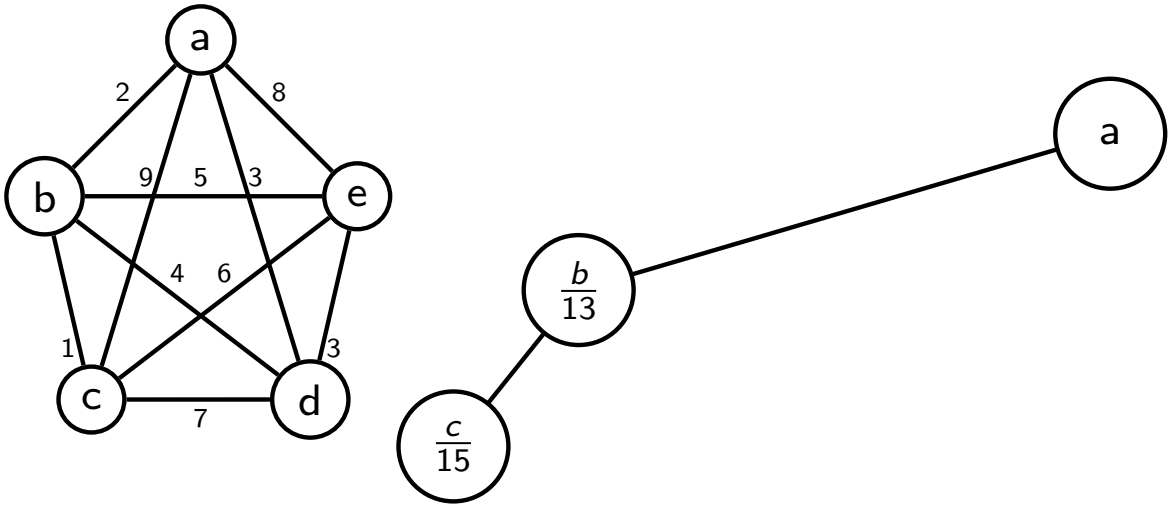

a

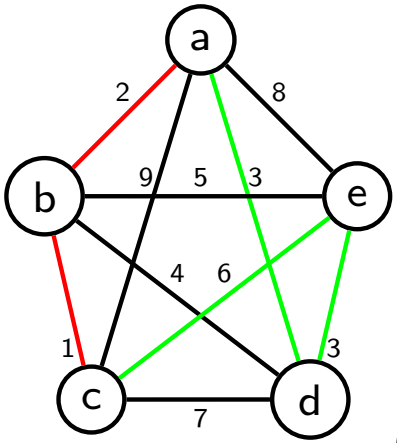

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d

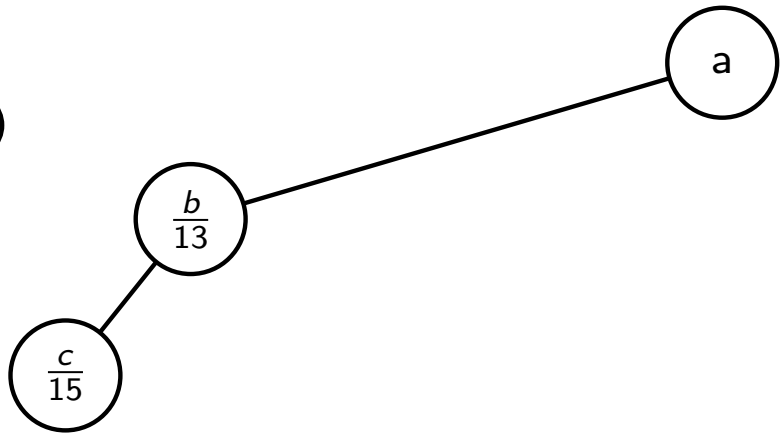

a
$\frac{b}{13}$

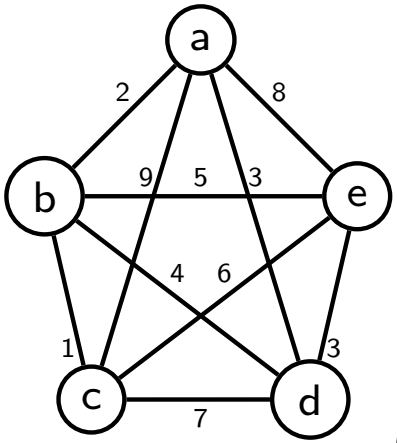

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d

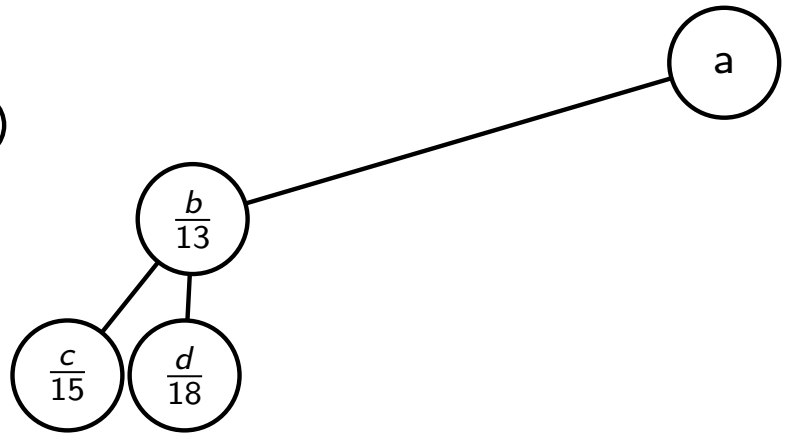

a
$\frac{b}{13}$

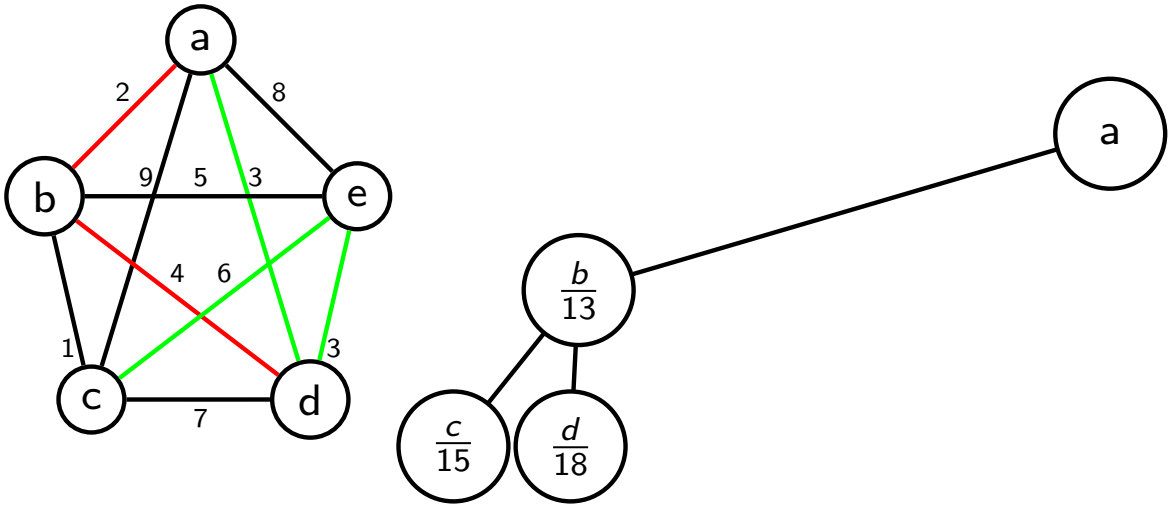

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
$\frac{b}{13}$
$\frac{c}{15}$

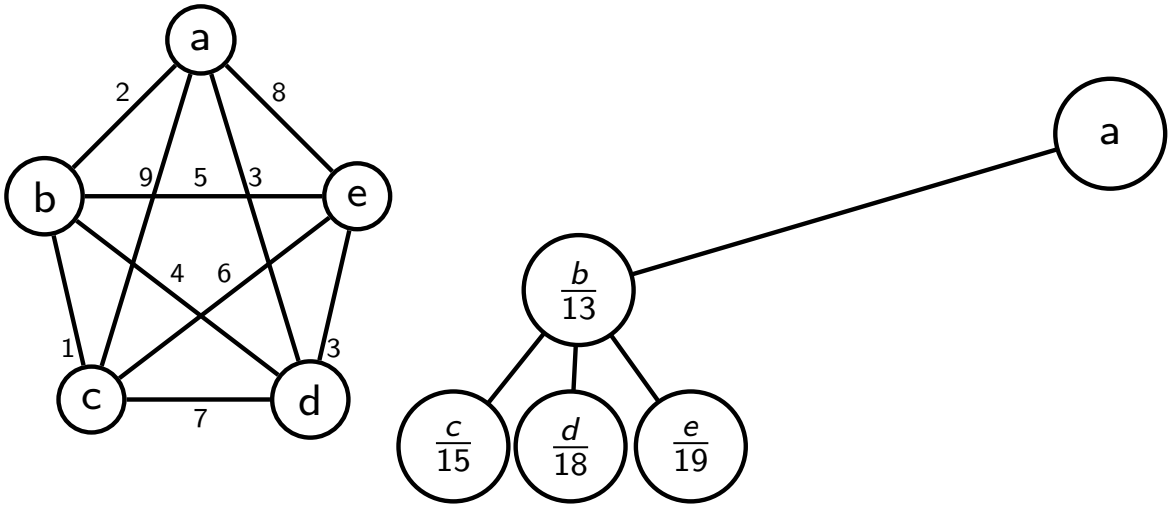

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d

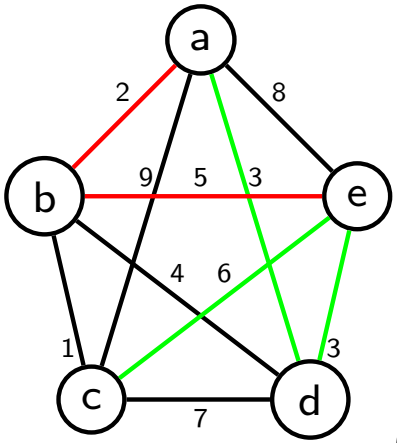

a
$\frac{b}{13}$
$\frac{c}{15}$

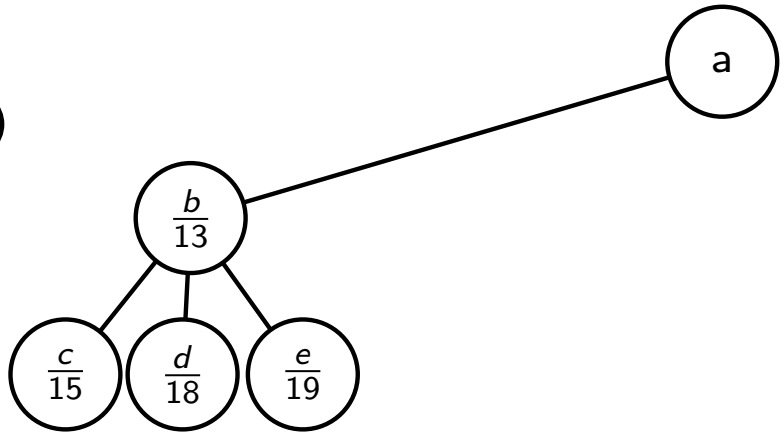

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d

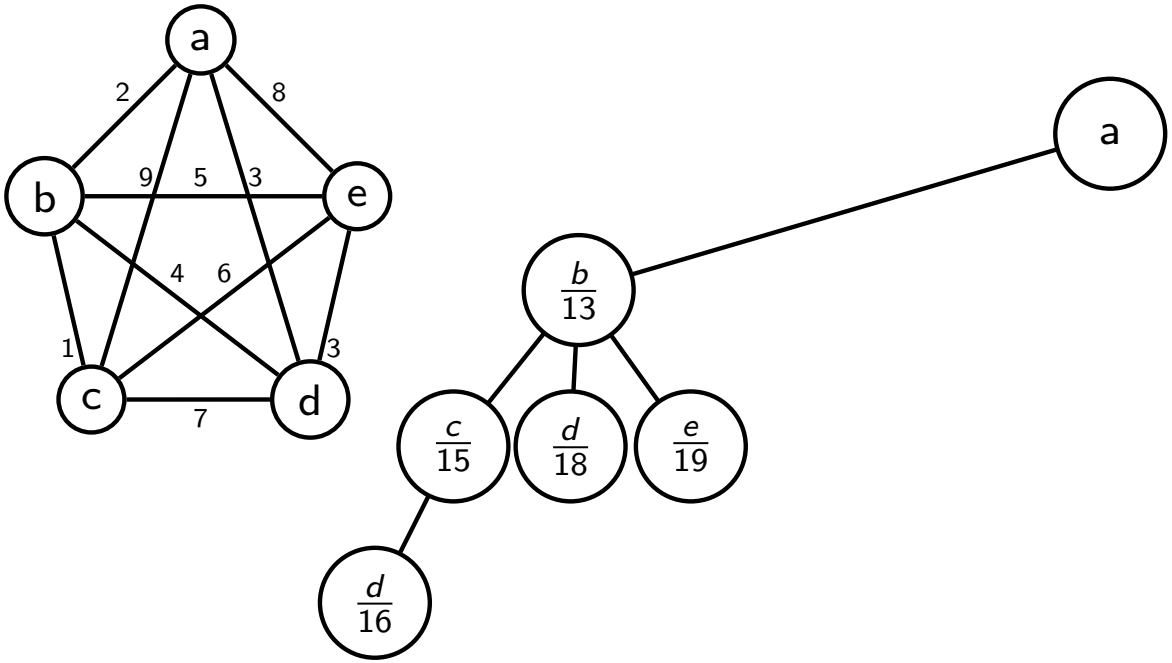

a
$\frac{b}{13}$
$\frac{c}{15}$
$\frac{d}{18}$

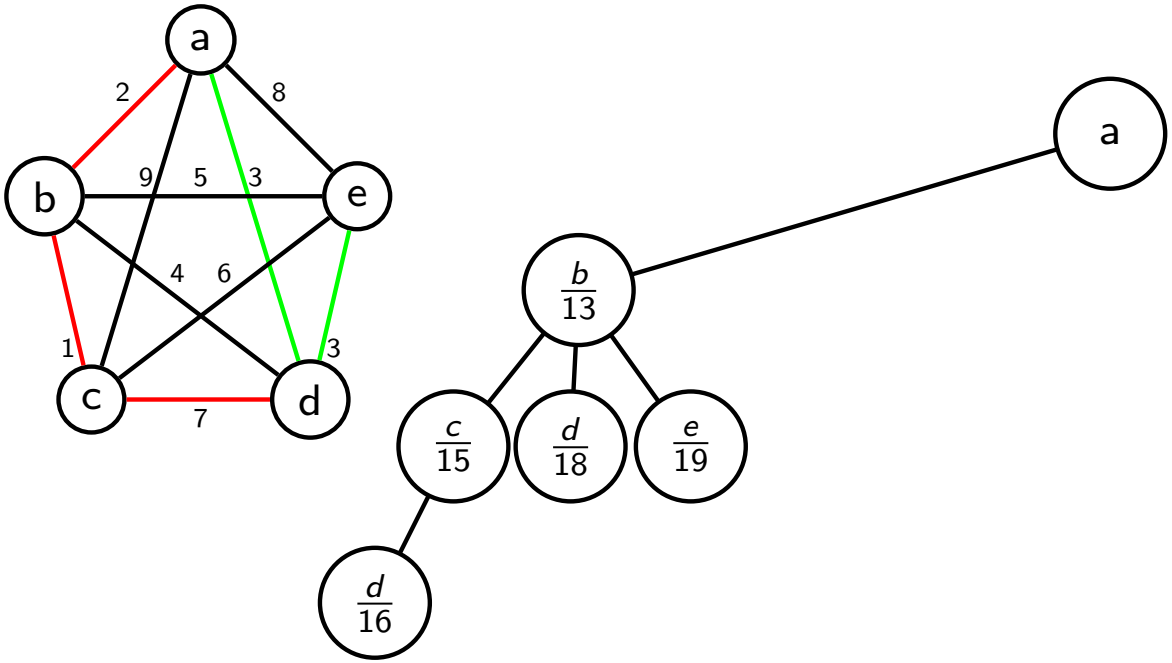

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
$\frac{b}{13}$
$\frac{c}{15}$
$\frac{d}{18}$

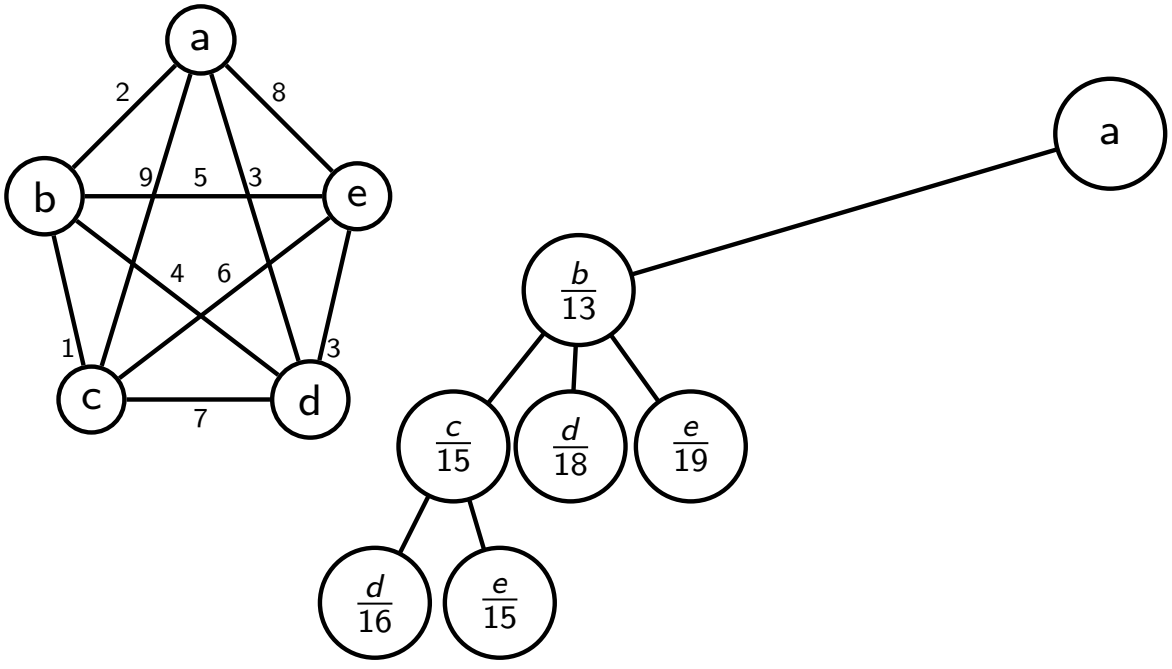

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
15
d
18
e
19

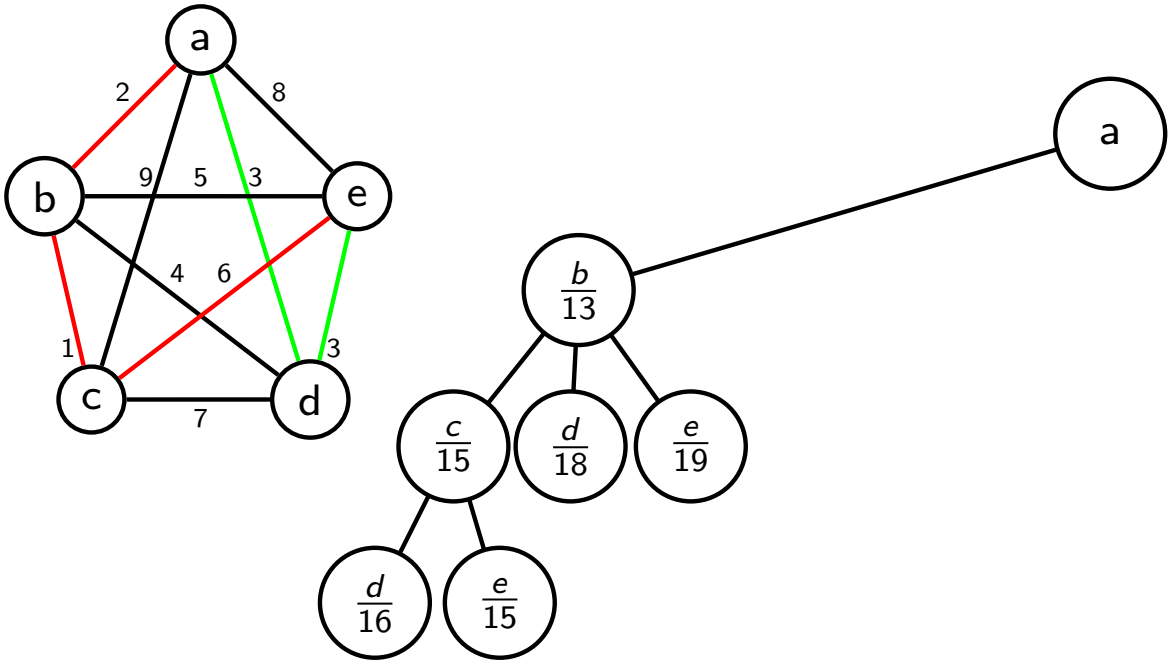

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7

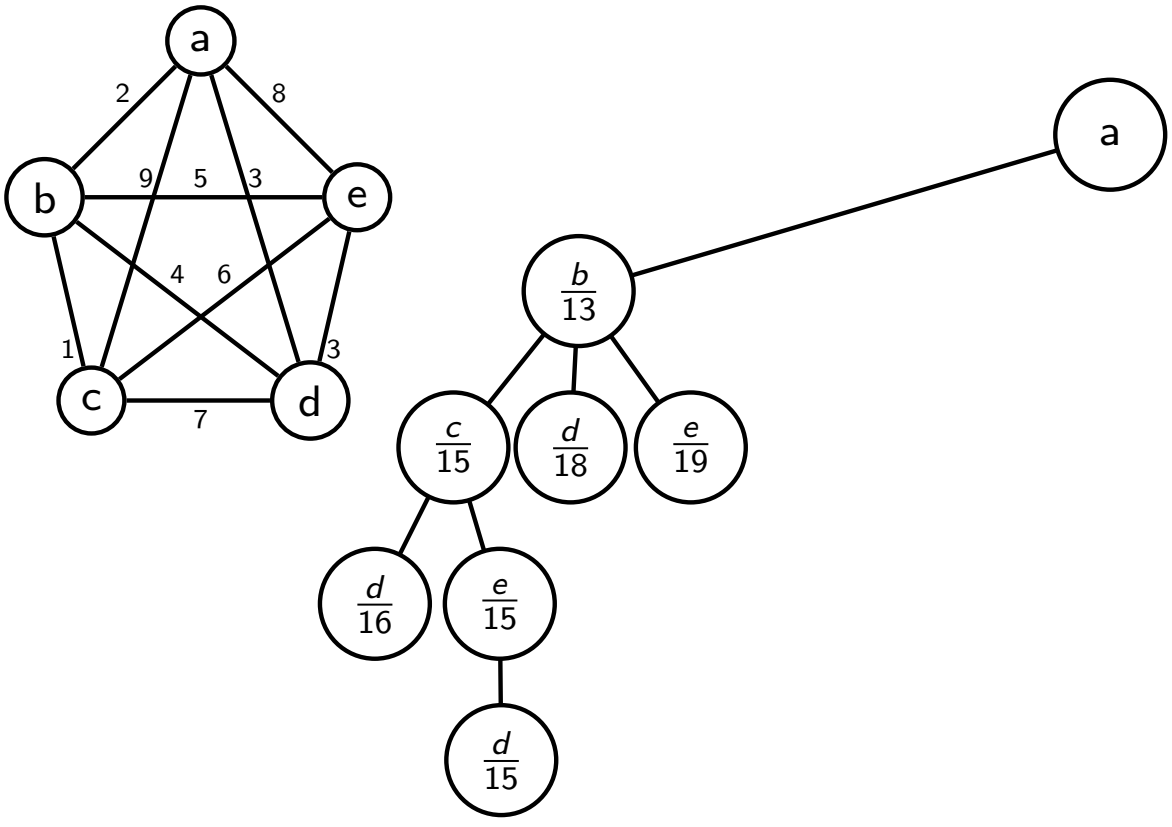

a
b
13
c
15
d
18
e
19

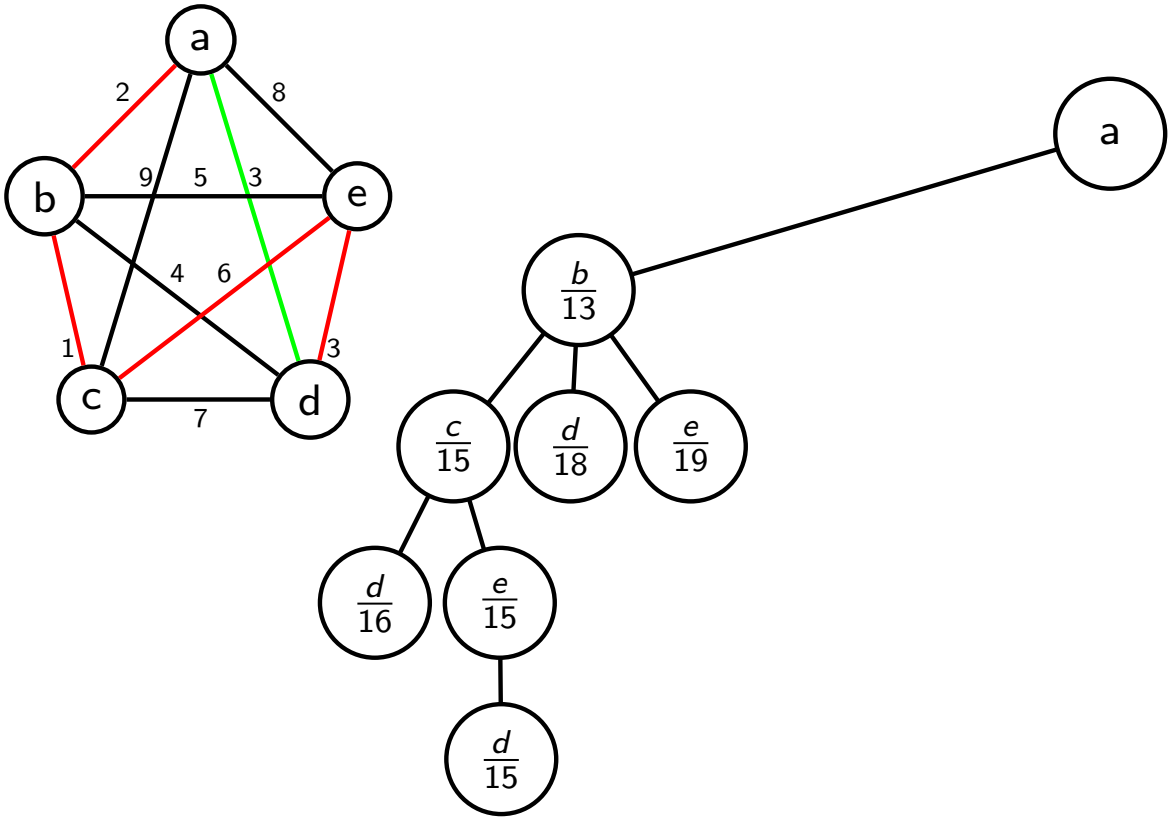

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
15
d
18
e
19
d
16

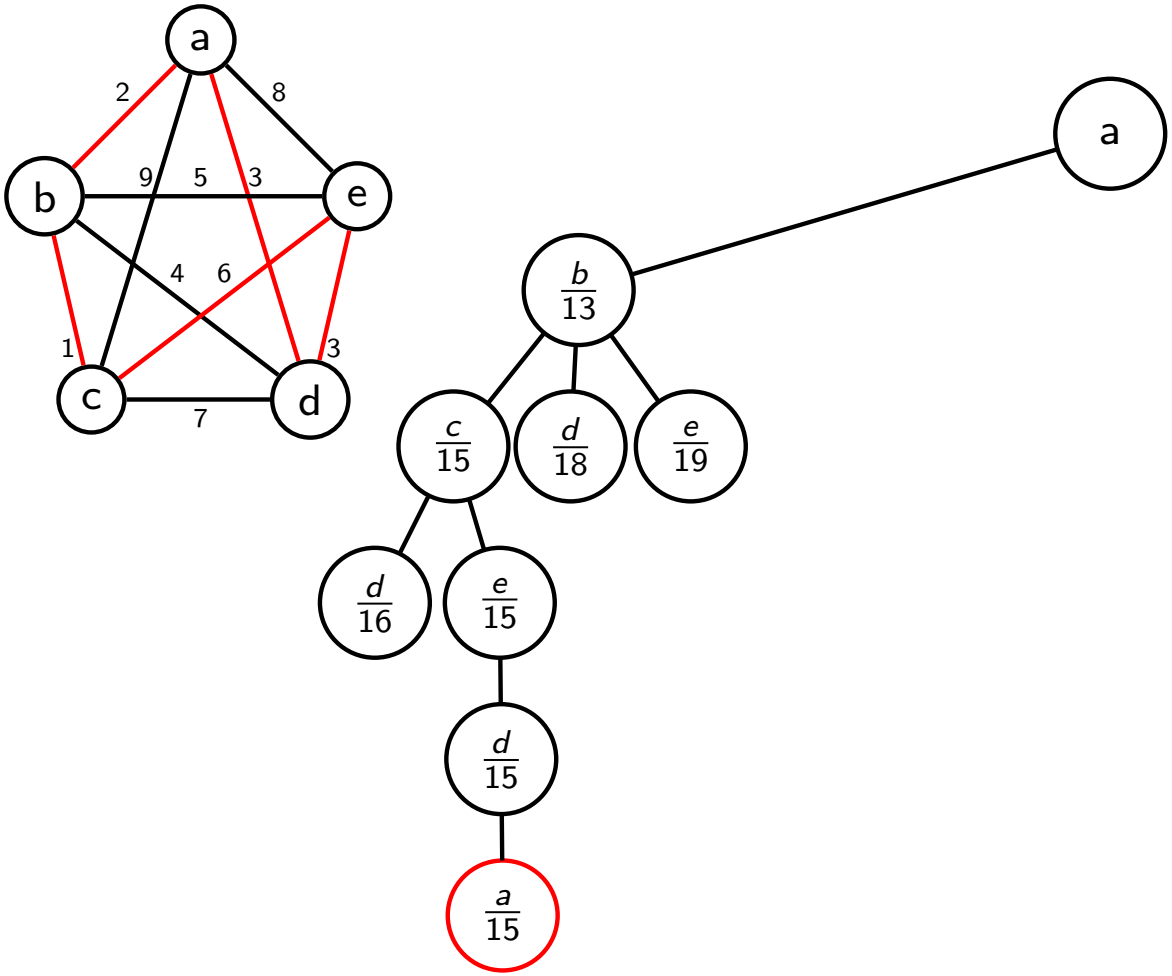

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
$\frac{b}{13}$
$\frac{c}{15}$
$\frac{d}{18}$
$\frac{e}{19}$
$\frac{d}{16}$

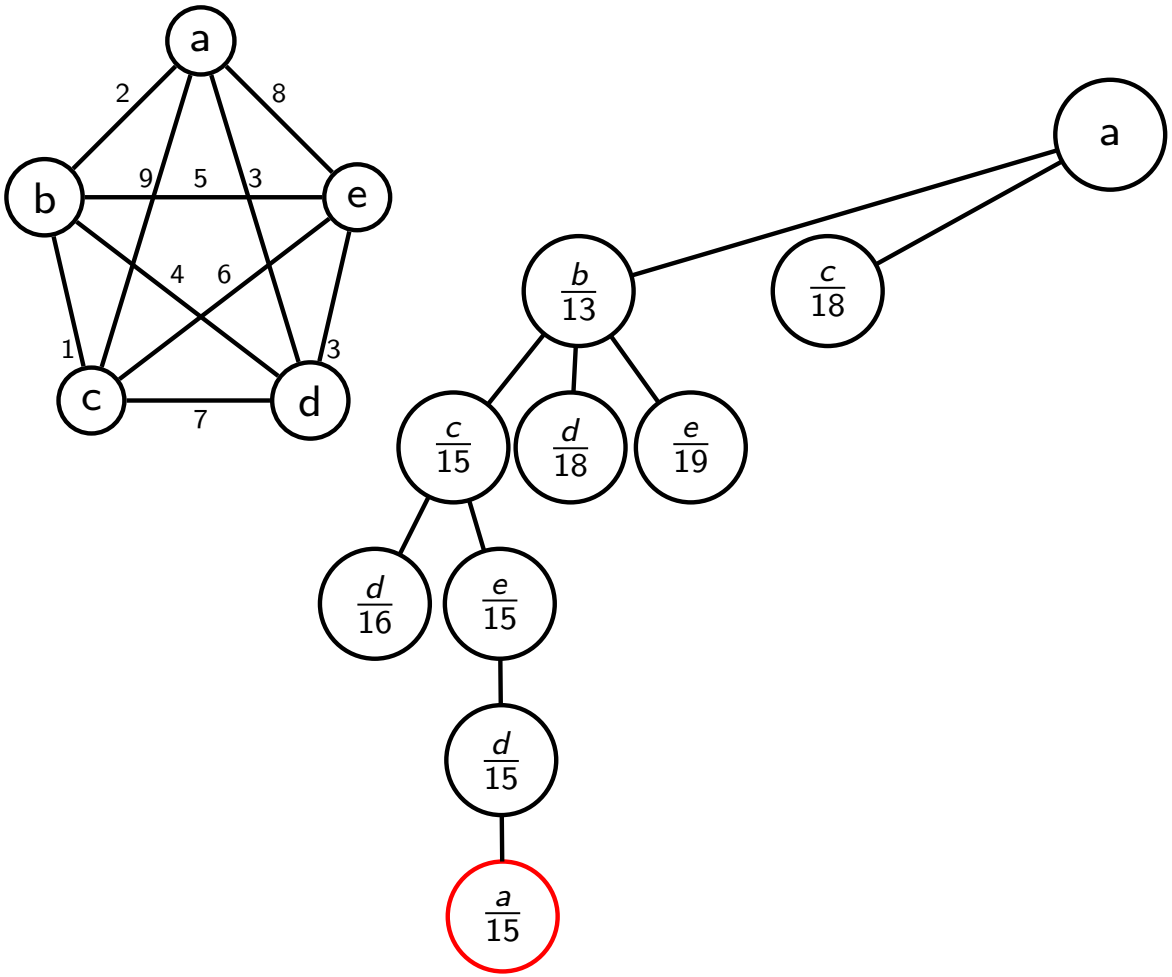

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
15
d
18
e
19
d
16
e
15

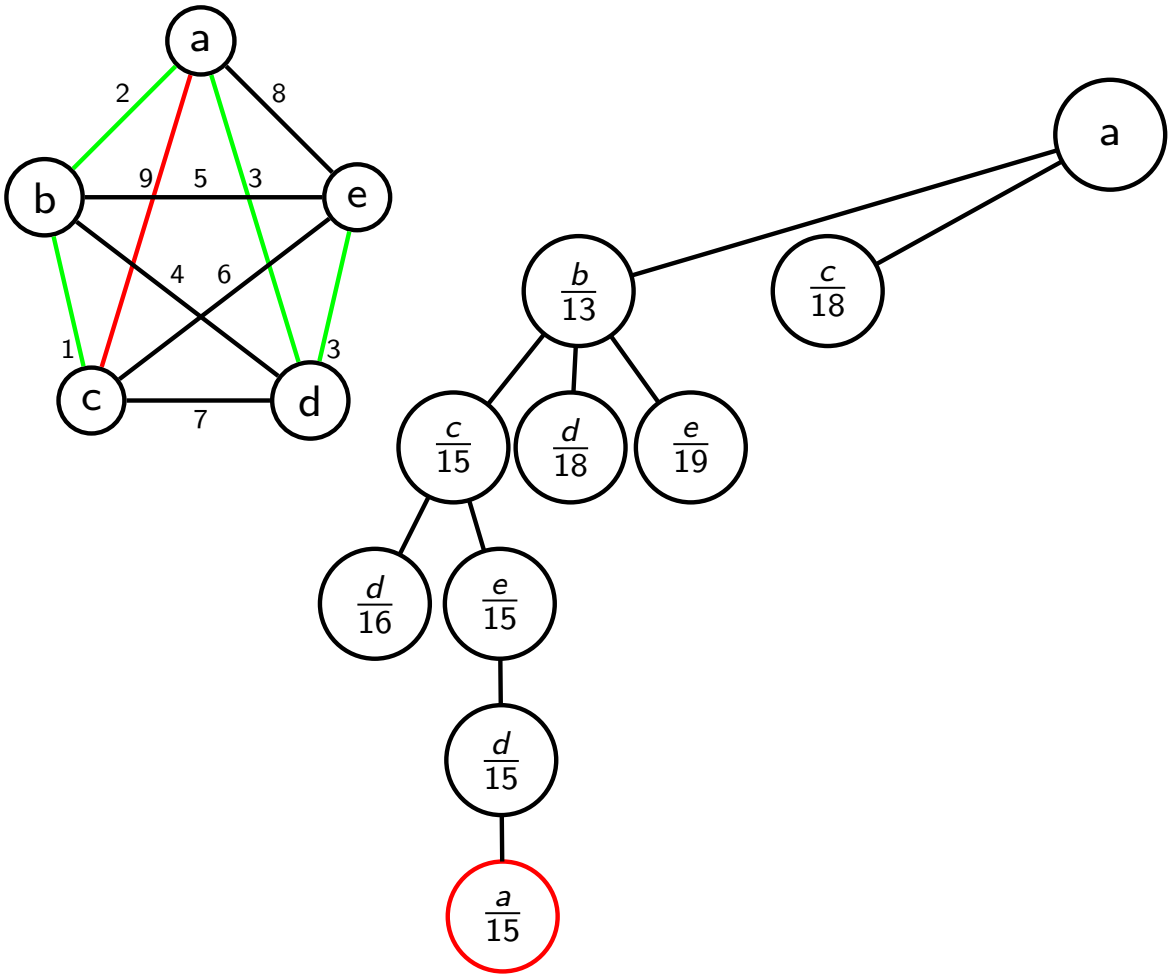

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
15
d
18
e
19
d
16
e
15

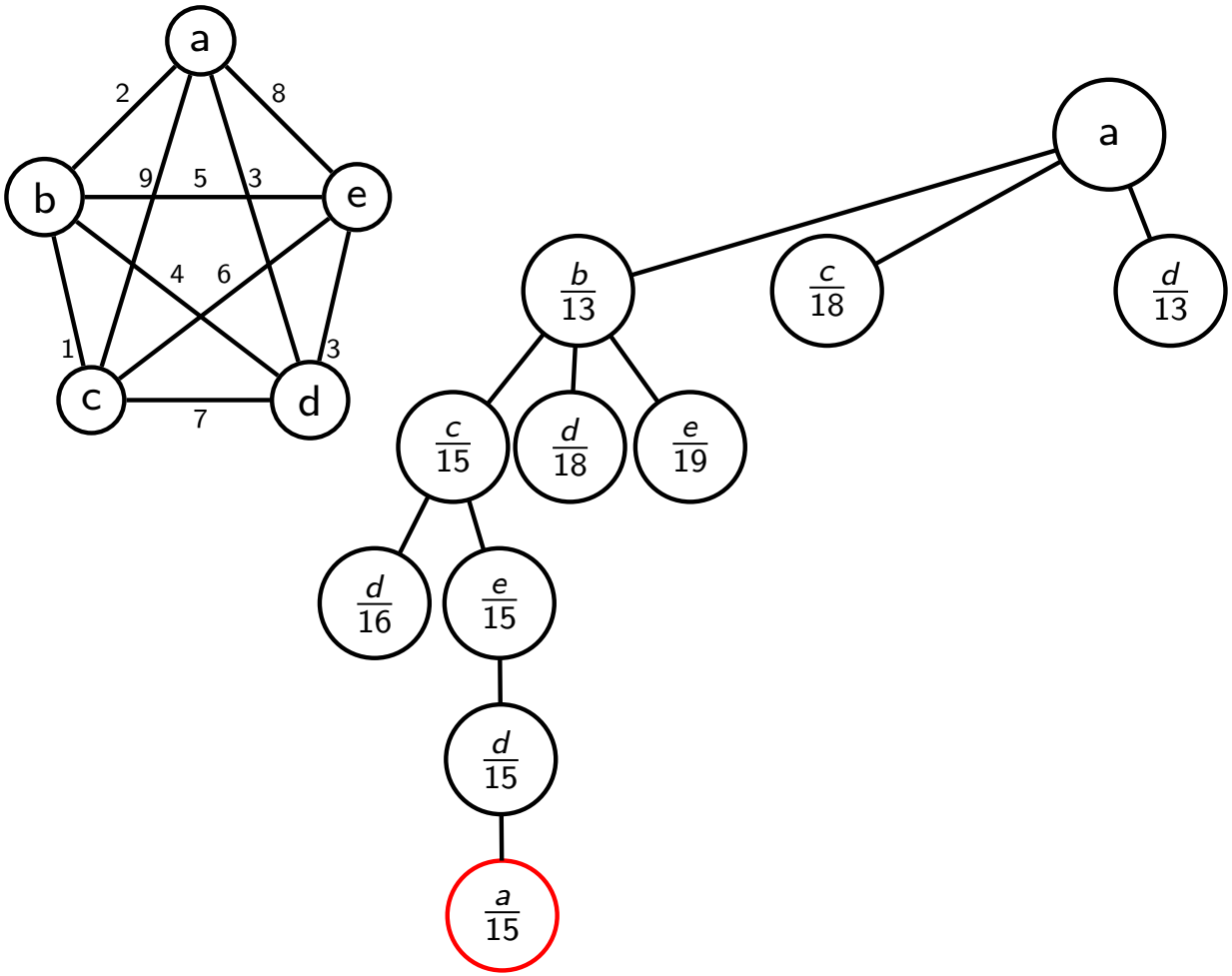

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
15
d
18
e
19
d
16
e
15
d
15

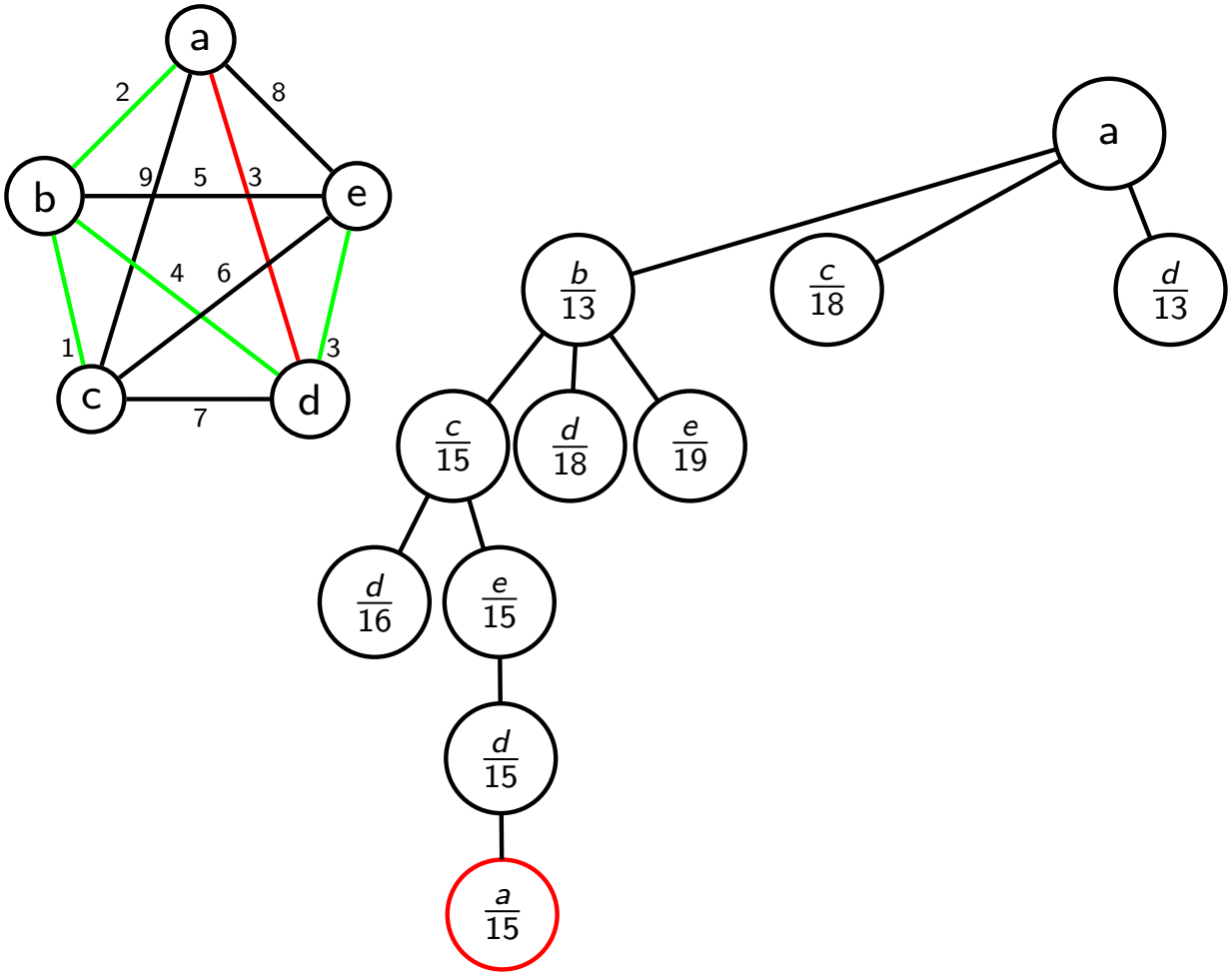

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
15
d
18
e
19
d
16
e
15
d
15

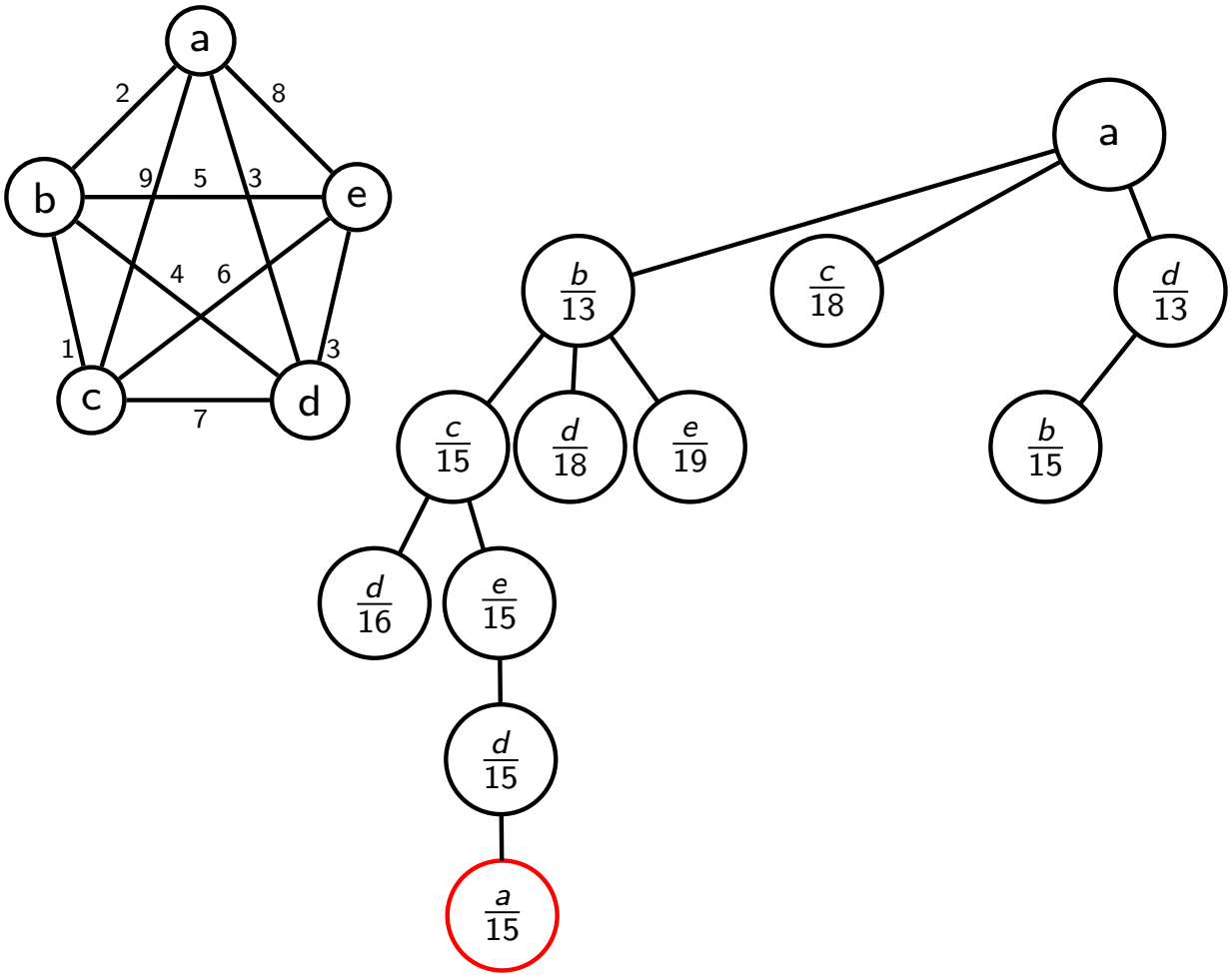

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
15
d
18
e
19
d
16
e
15
d
15
a
15

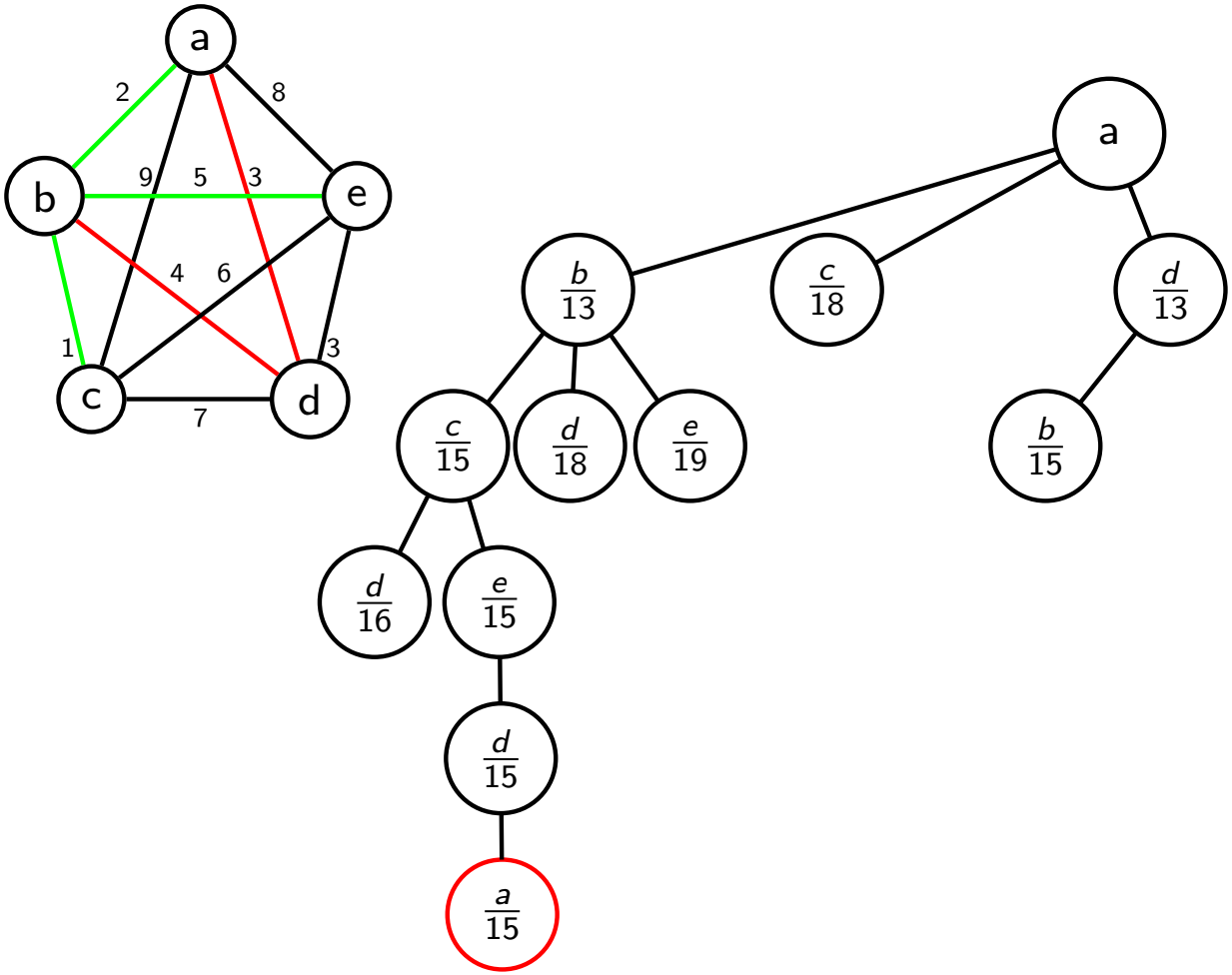

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
18
c
15
d
18
e
19
d
16
e
15
d
15
a
15

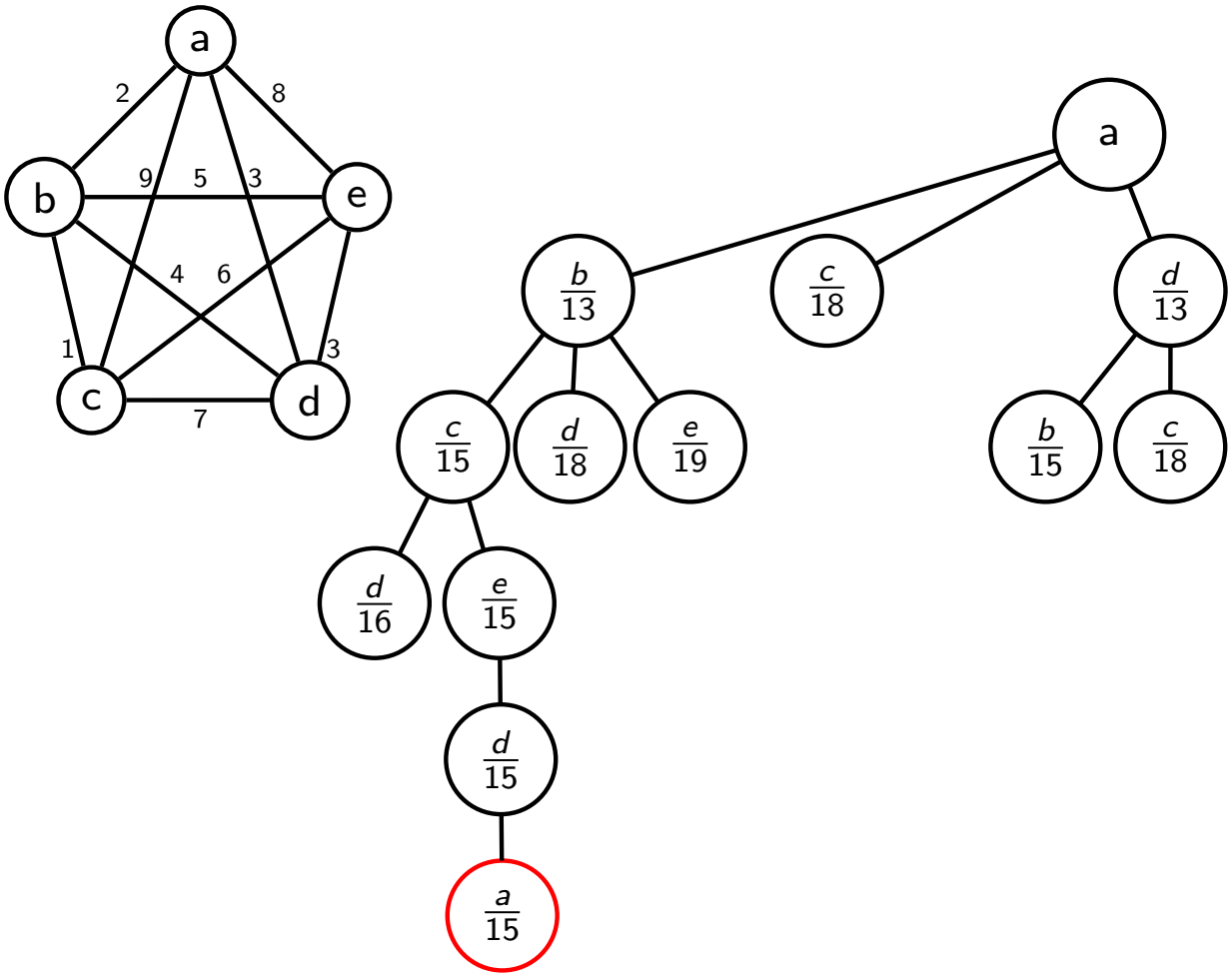

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
18
c
15
d
18
e
19
d
16
e
15
d
15
a
15

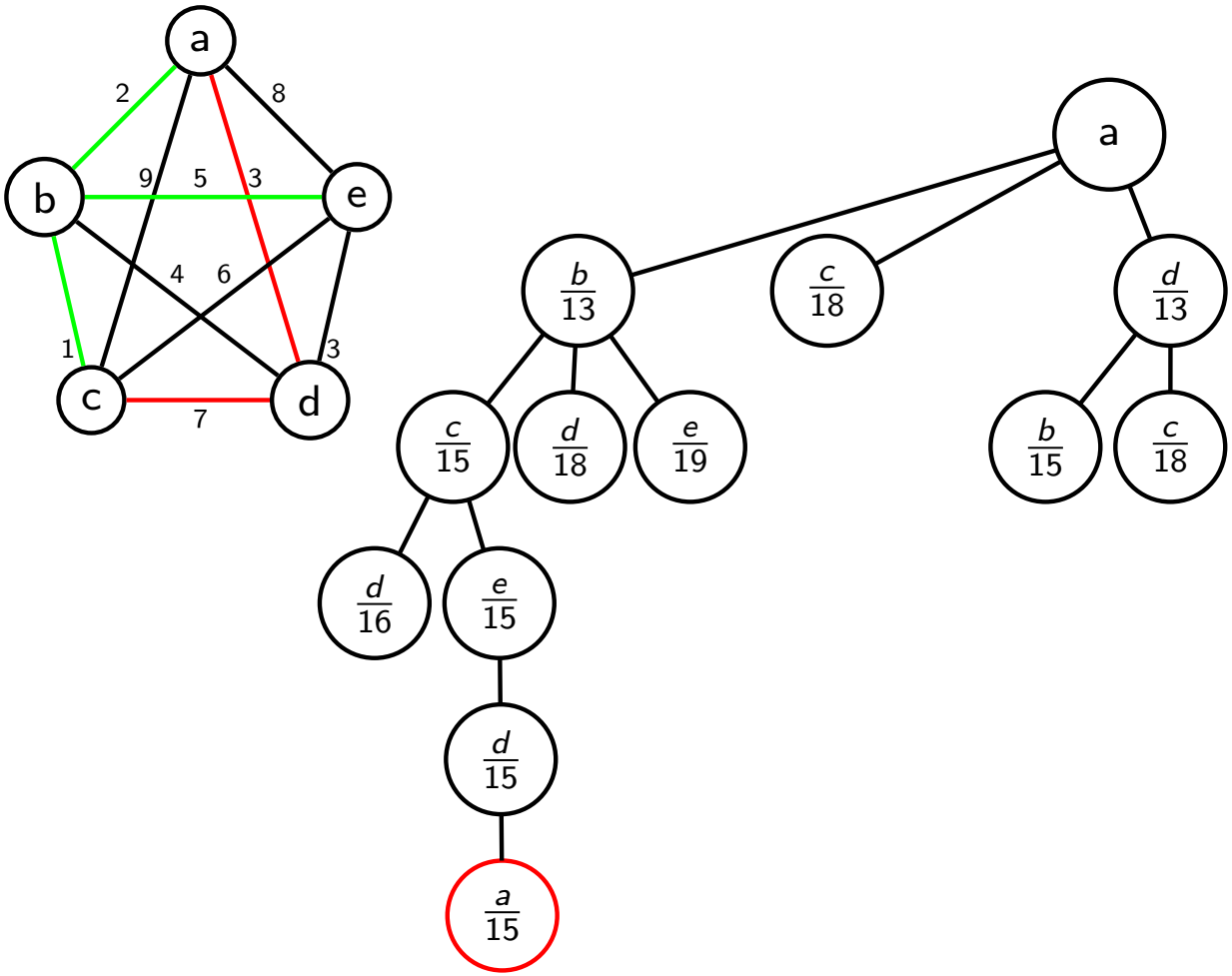

a
2
8
b
9
5
3
e
4
6
1
3
c
7
d
a
b
13
c
18
d
13
c
15
d
18
e
19
d
16
e
15
d
15
a
15

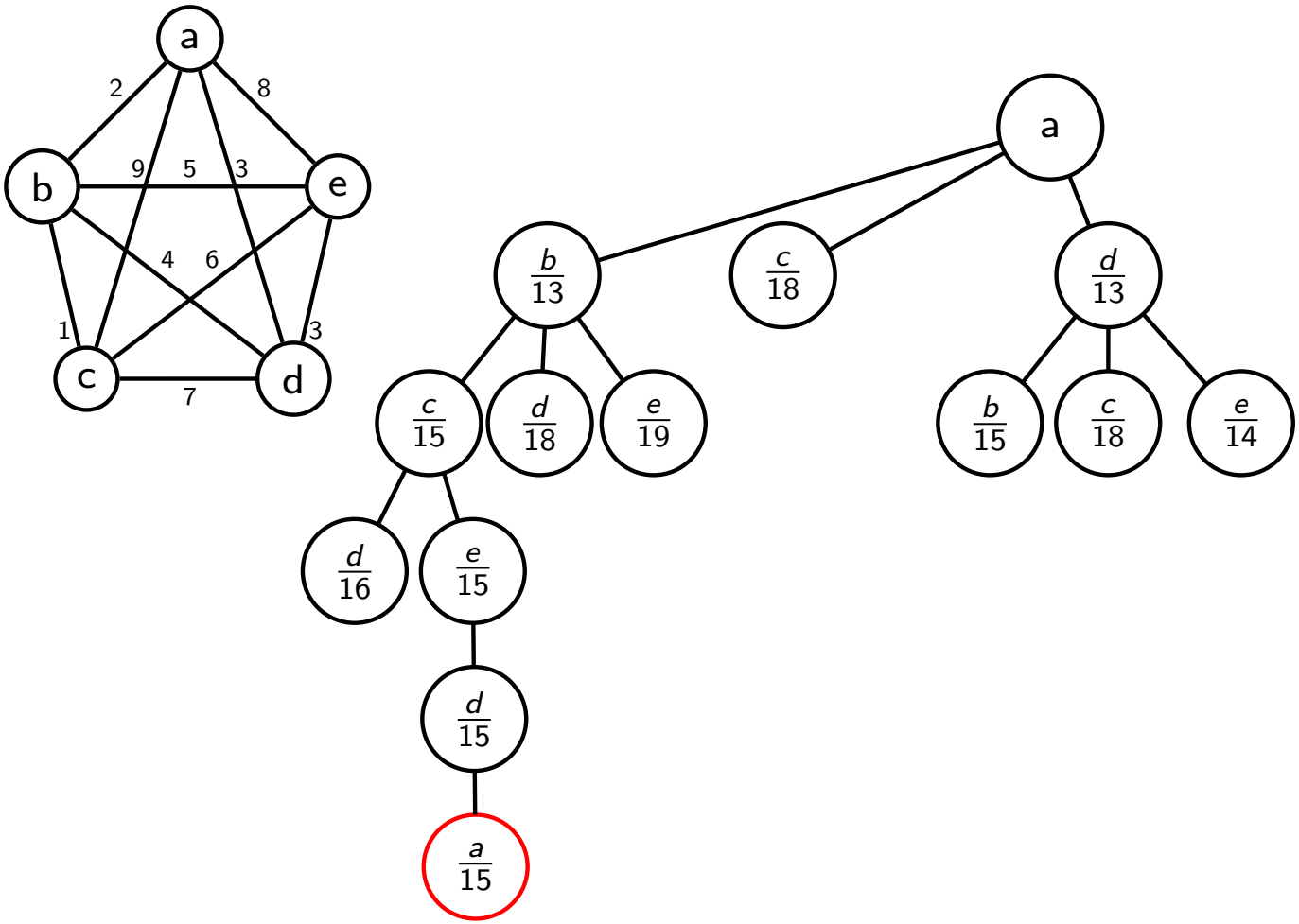

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
d/16
e/15
d/15
a/15

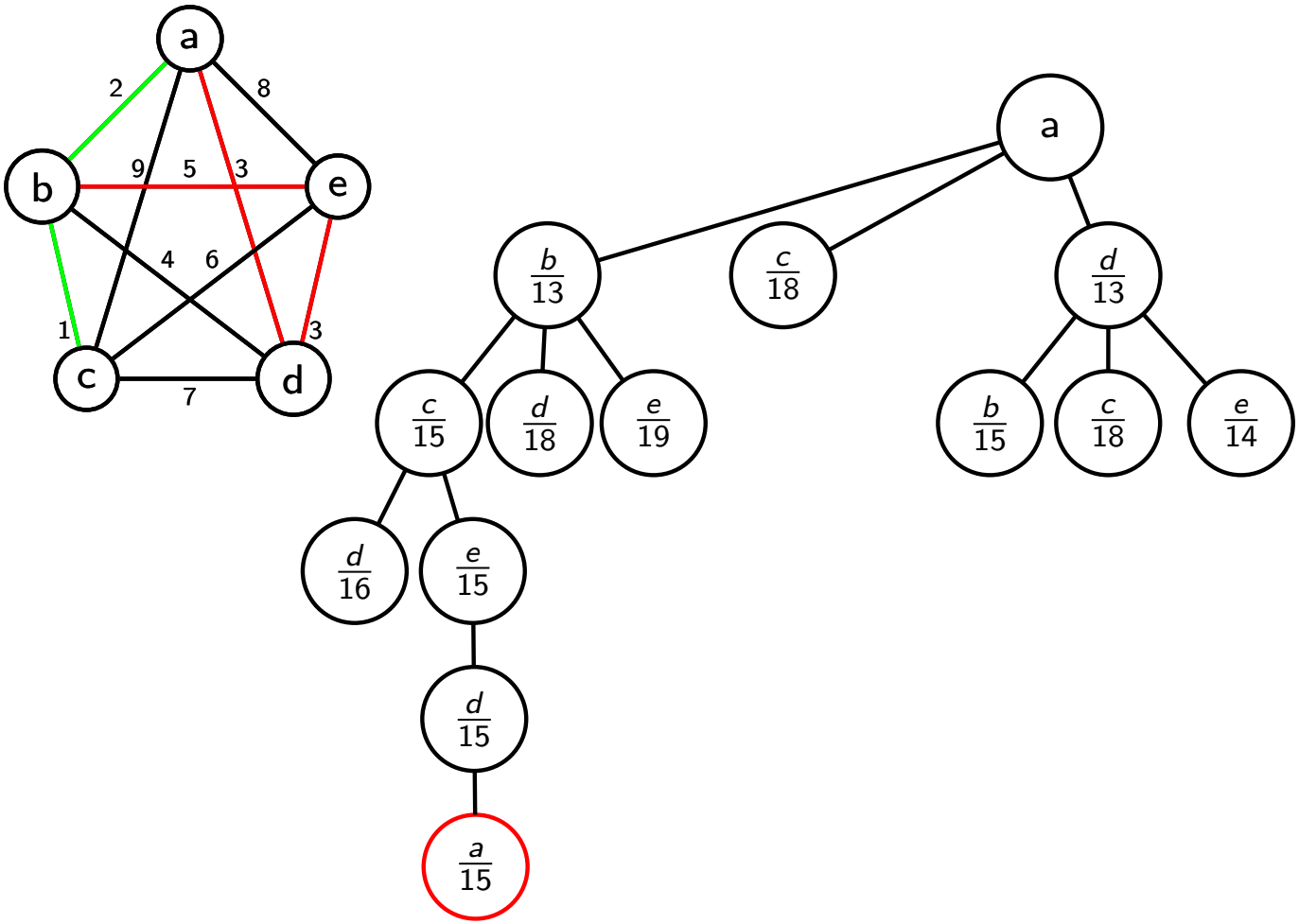

a
b
e
c
d
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
d/16
e/15
d/15
a/15

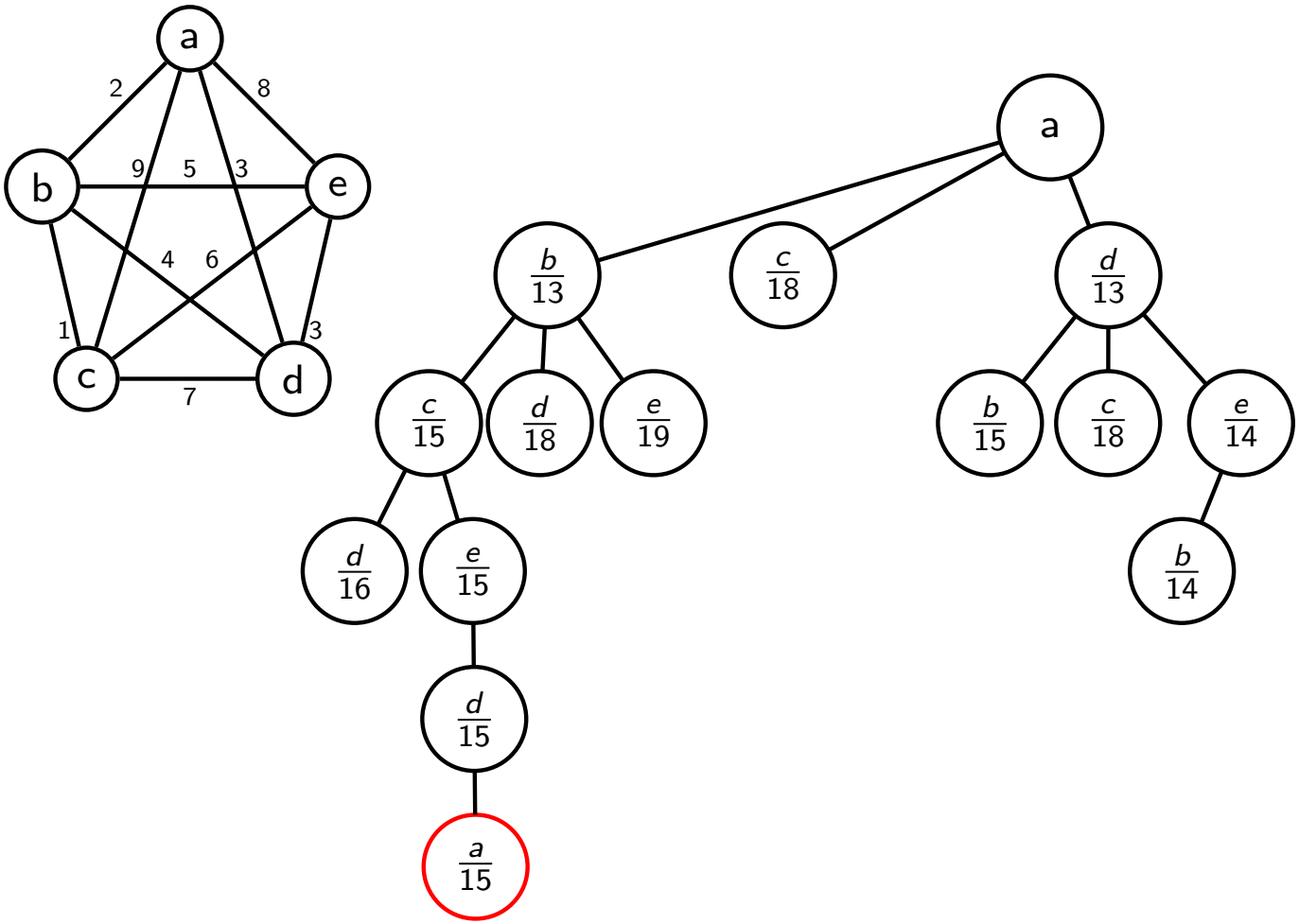

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
d/16
e/15
d/15
a/15

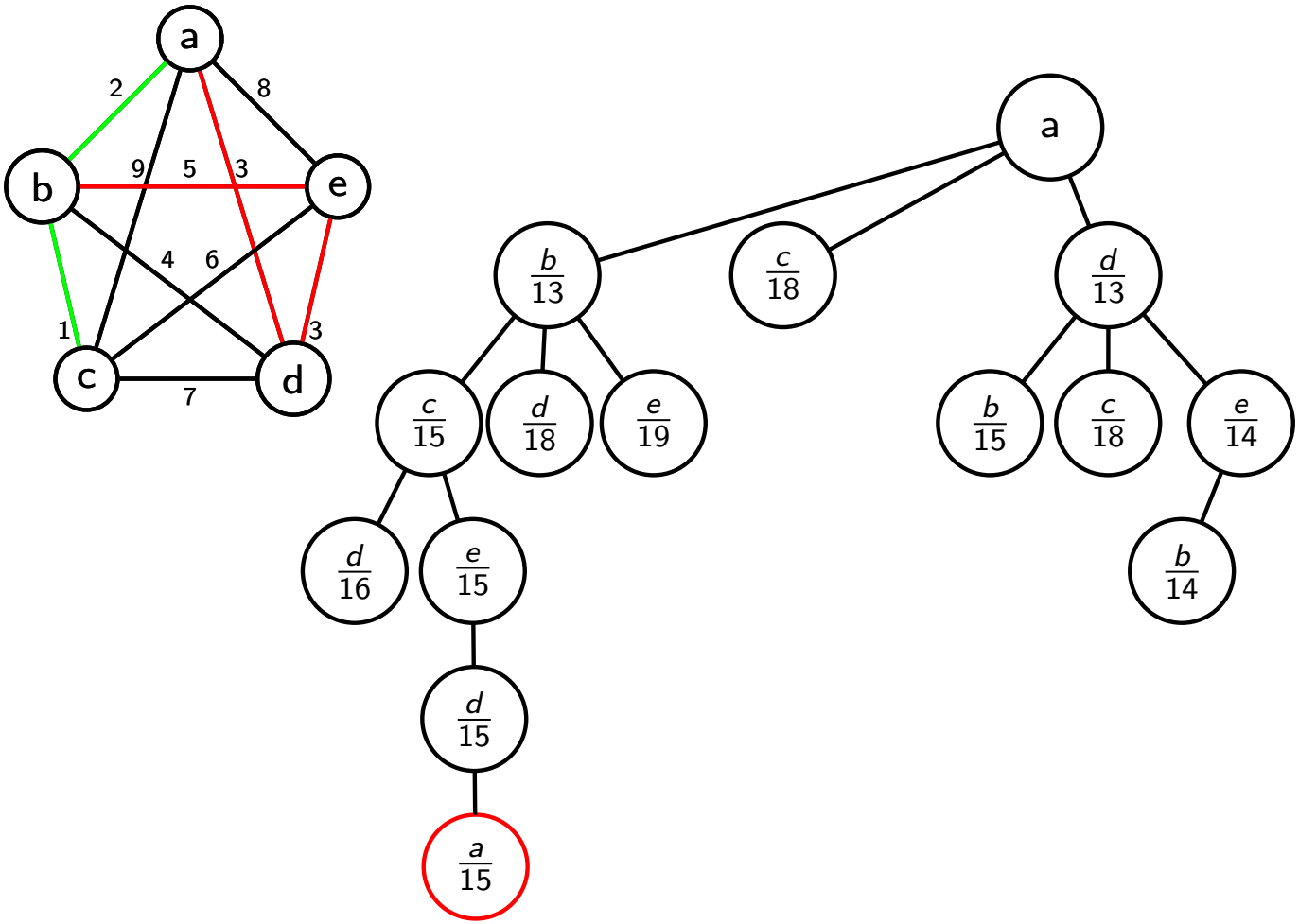

a
b
e
c
d
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
d/16
e/15
d/15
a/15

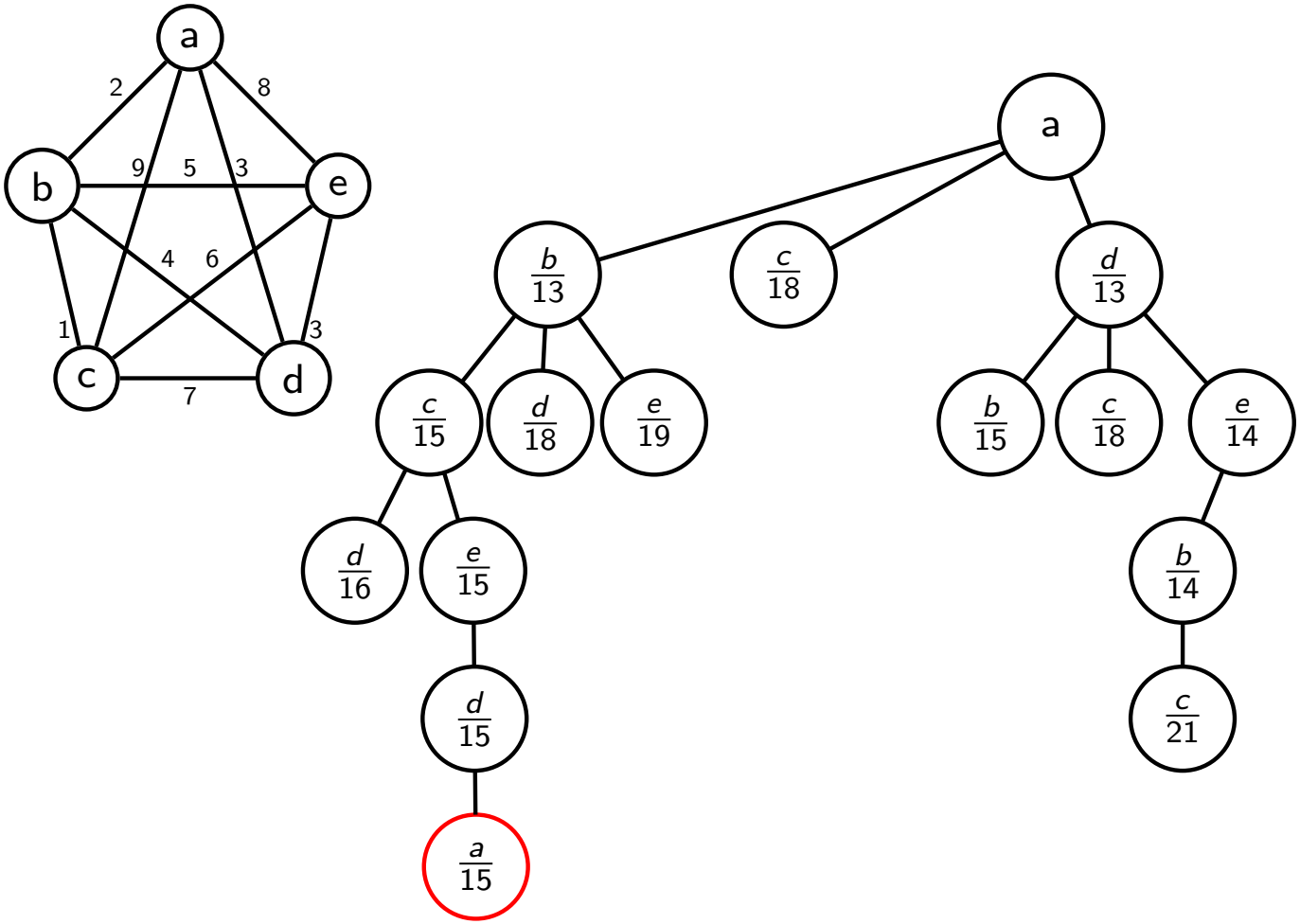

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
d/16
e/15
d/15
a/15

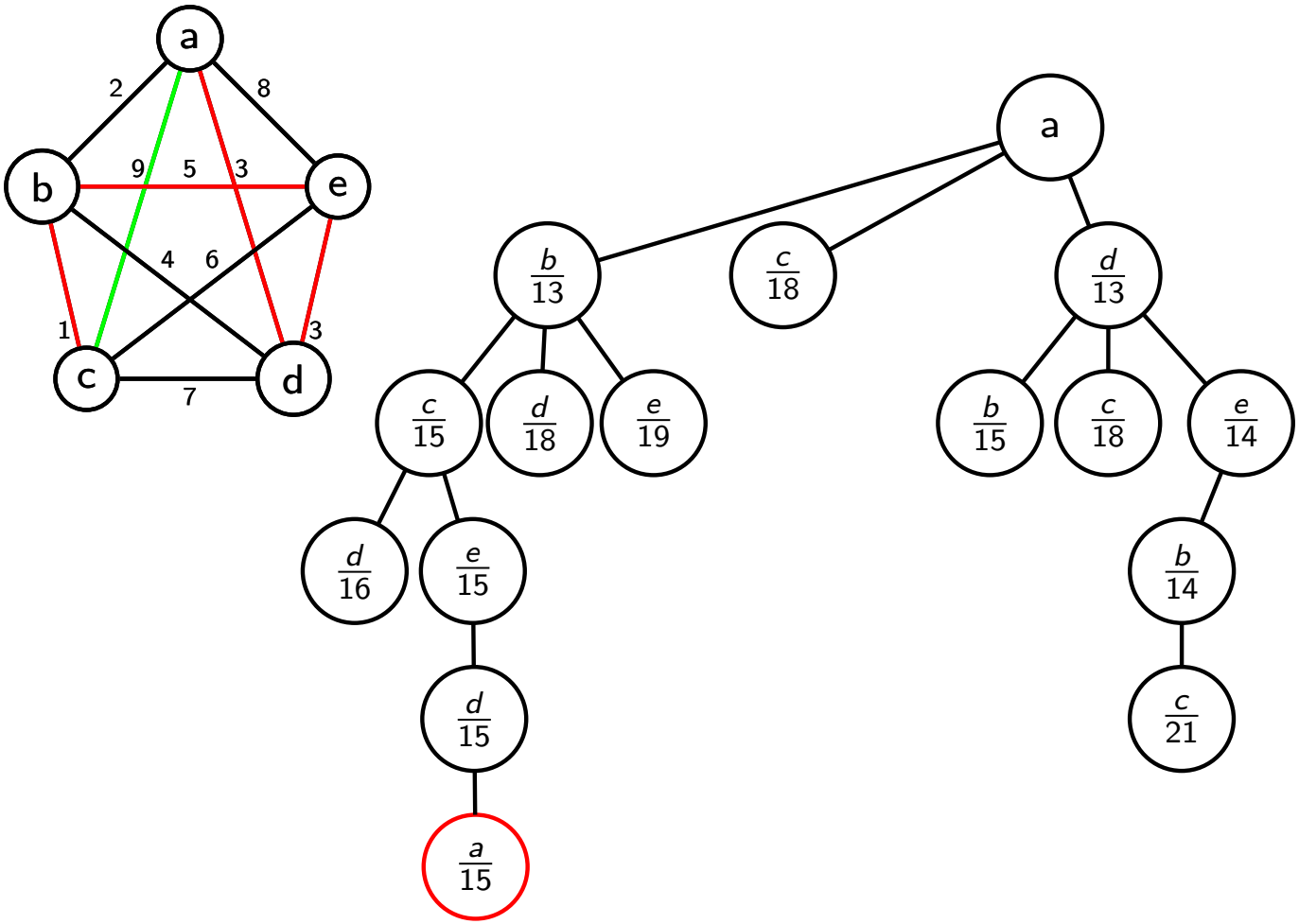

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
d/15
a/15

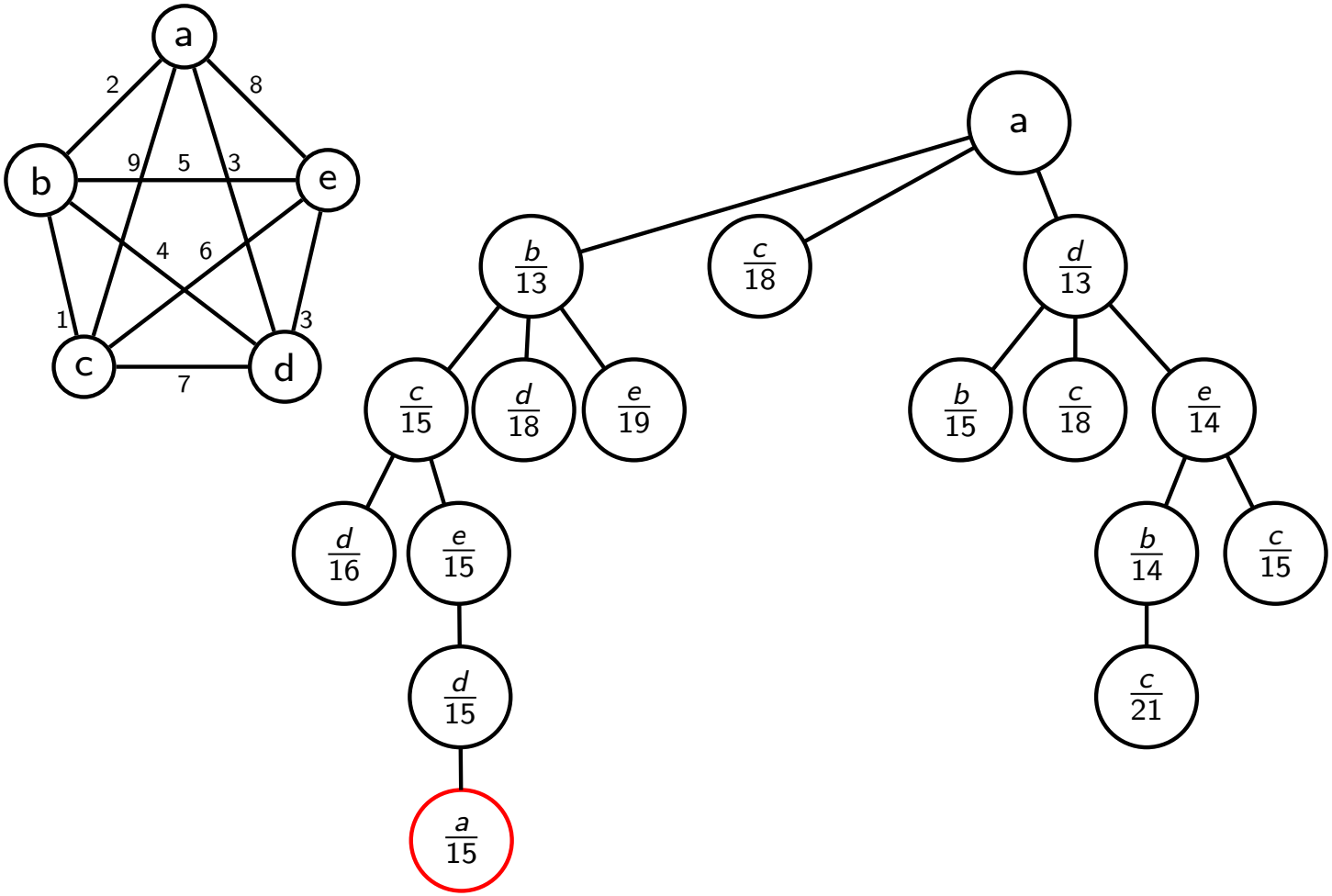

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
d/15
a/15

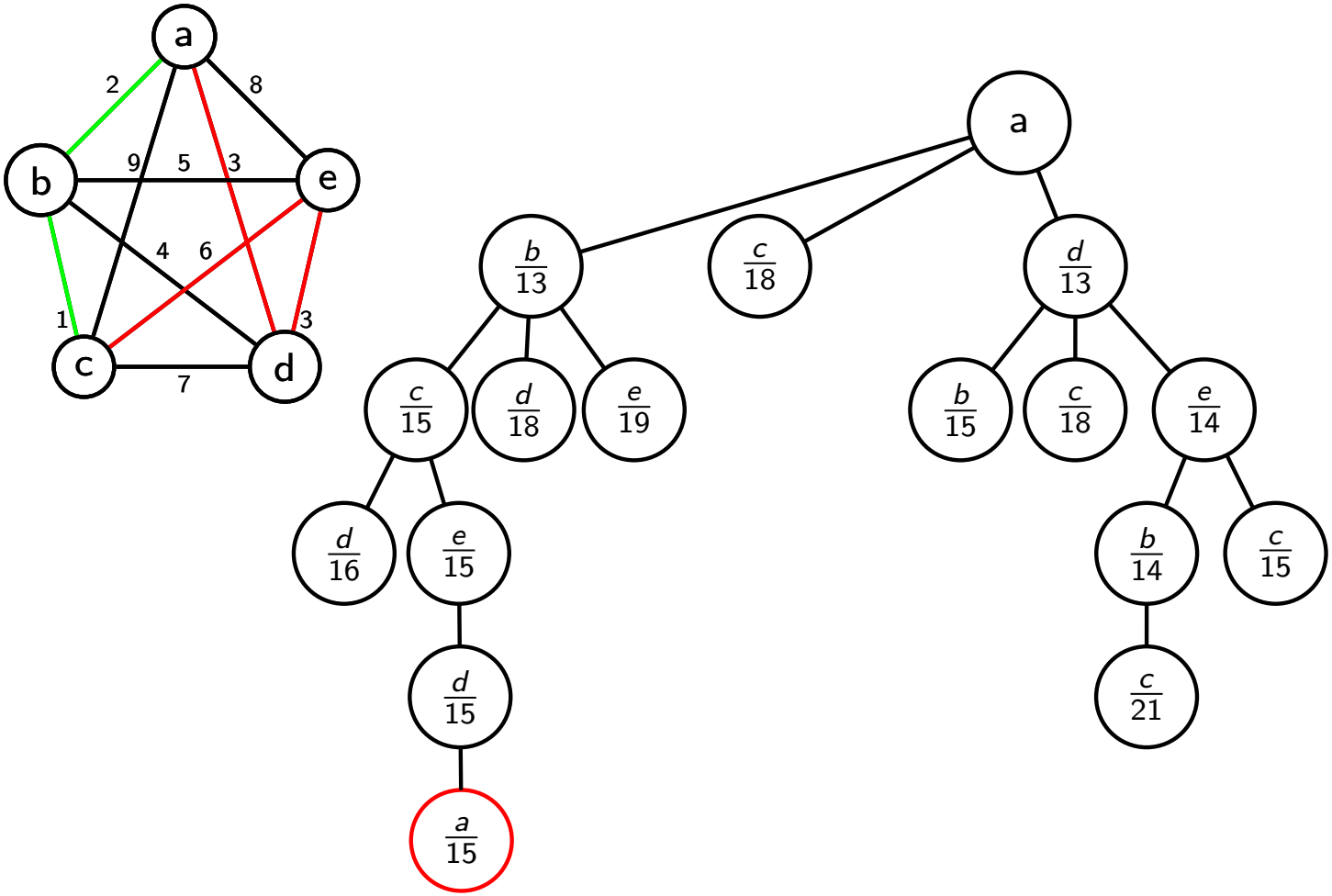

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
d/15
a/15

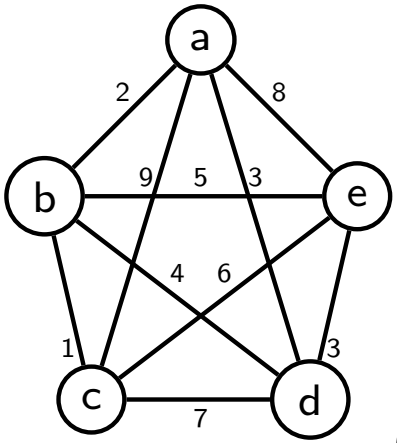

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
d/15
a/15

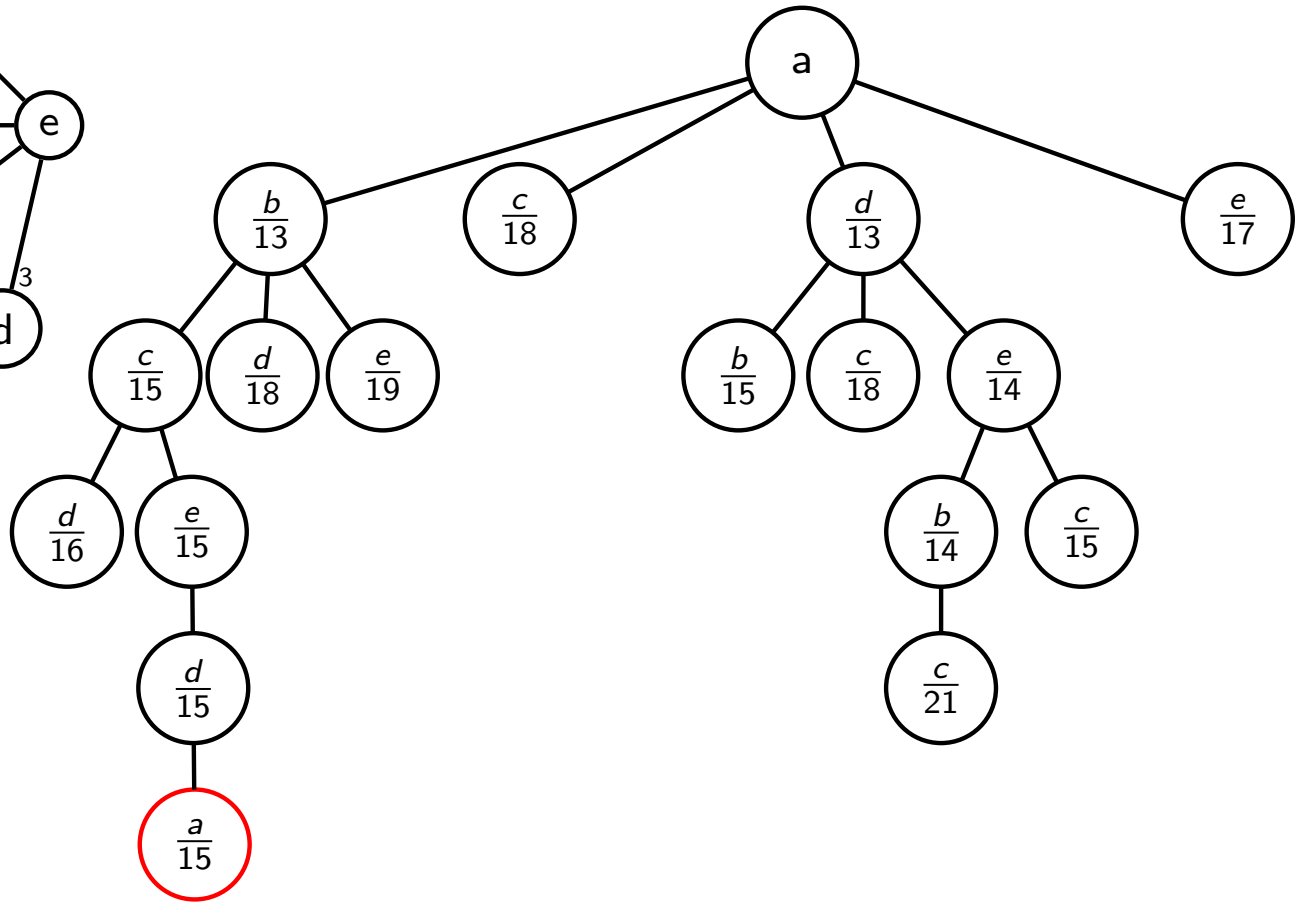

a
b
e
c
d
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
d/15
c/21
a/15

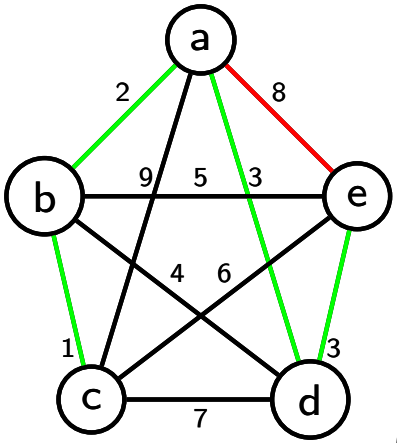

a
b
e
c
d
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
d/15
c/21
a/15

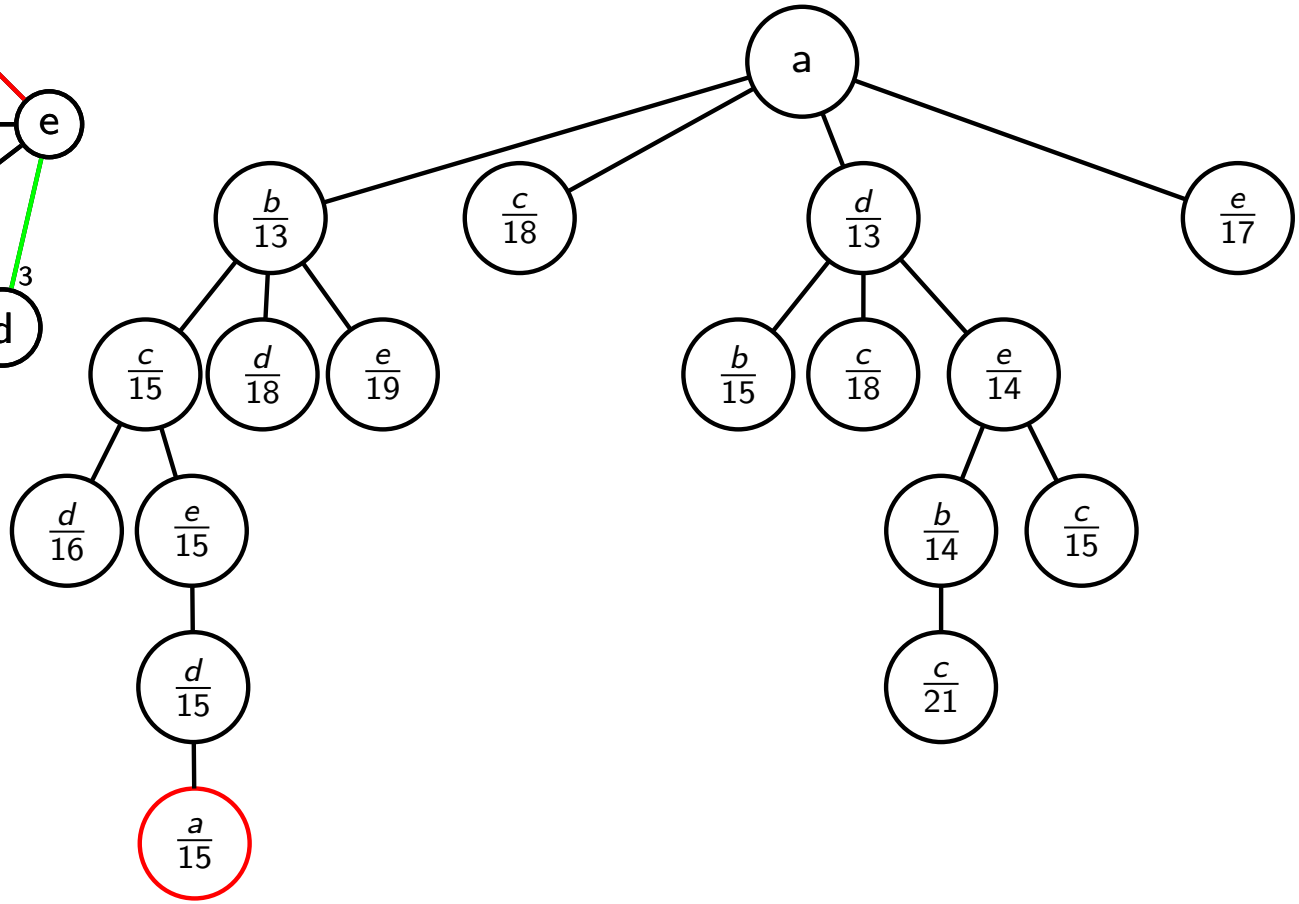

a
b
e
c
d
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
c/15
d/15
c/21
a/15

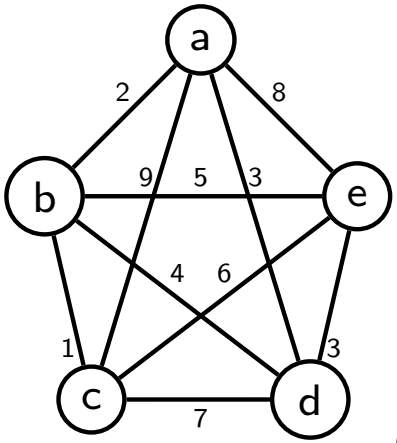

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7
a
b/13
c/18
d/13
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
c/15
d/15
c/21
a/15

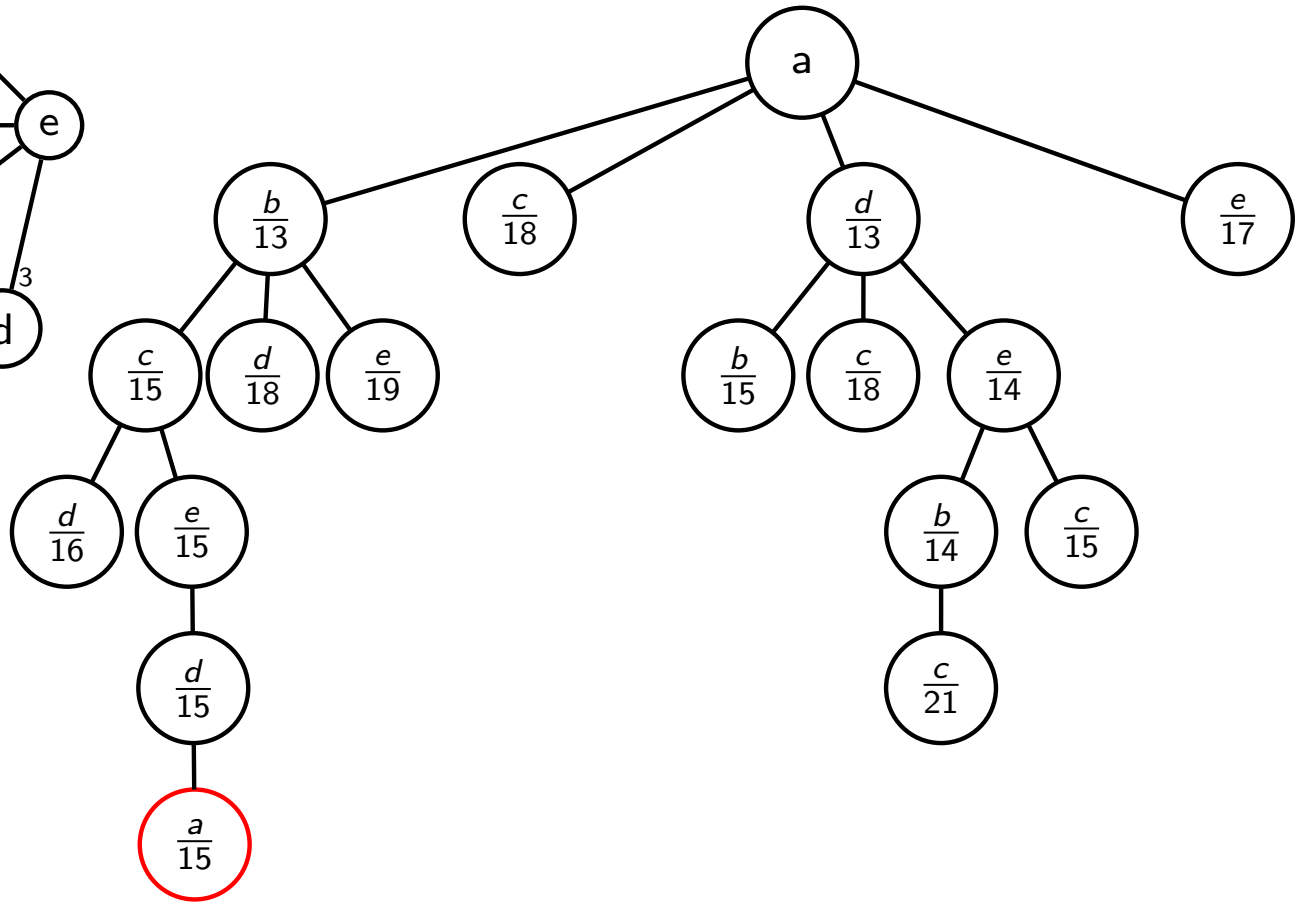

a
b
e
c
d
2
8
9
5
3
4
6
1
3
7

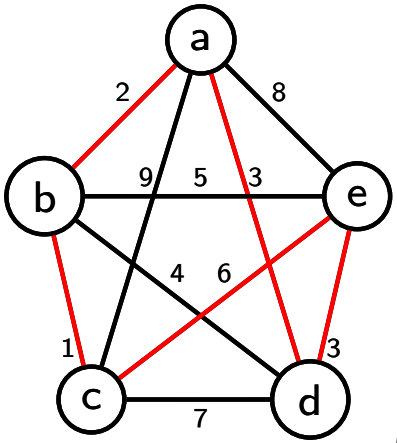

a
b/13
c/18
d/13
e/17
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
c/15
d/15
c/21
a/15

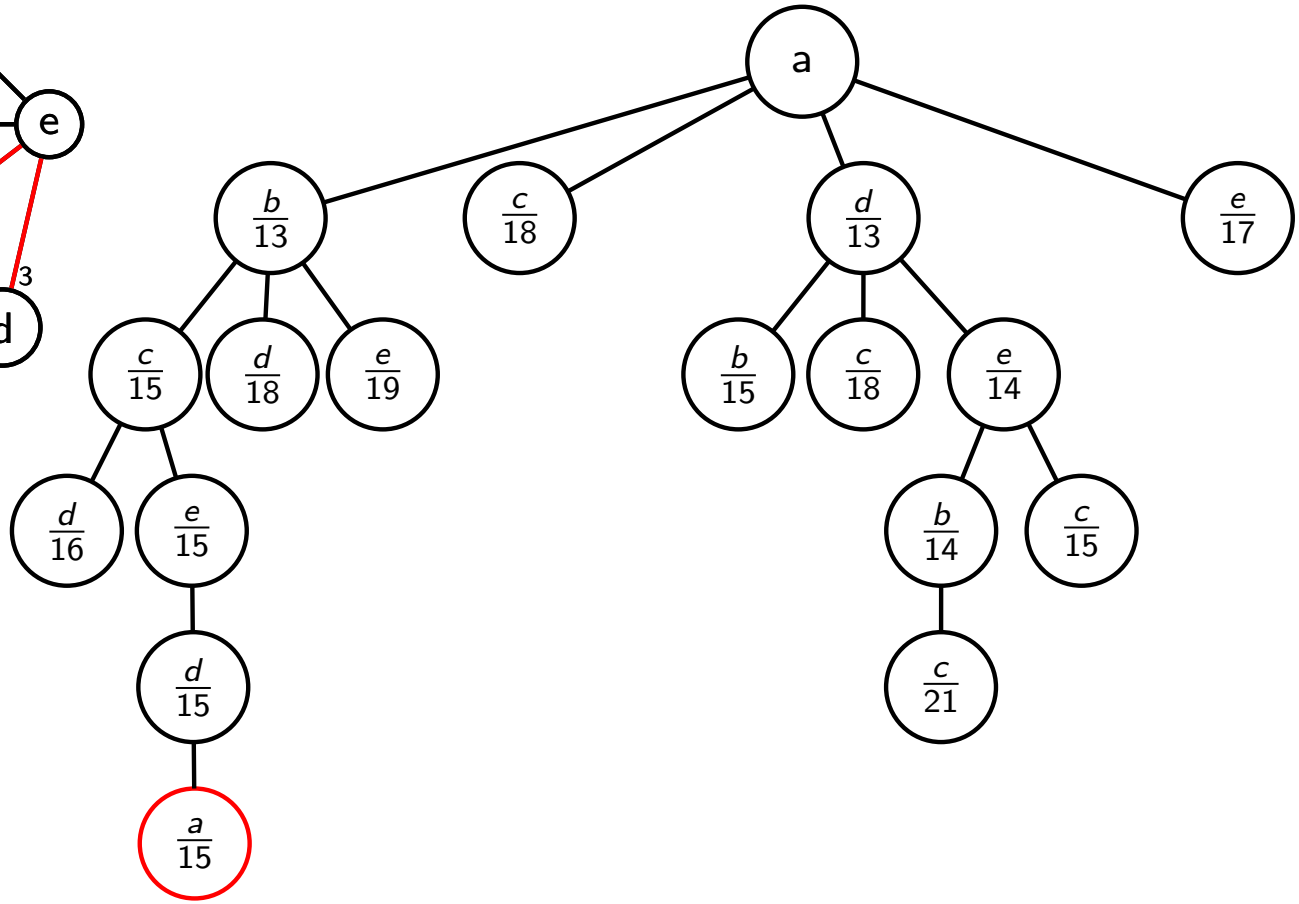

a
b
c
d
e
2
8
9
5
3
4
6
1
3
7


a
b/13
c/18
d/13
e/17
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
c/15
d/15
c/21
a/15


a
b
e
c
d
2
8
9
5
3
4
6
1
3
7


a
b/13
c/18
d/13
e/17
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
c/15
d/15
c/21
a/15


a
b
c
d
e
2
8
9
5
3
4
6
1
3
7


a
b/13
c/18
d/13
e/17
c/15
d/18
e/19
b/15
c/18
e/14
d/16
e/15
b/14
c/15
d/15
c/21
a/15