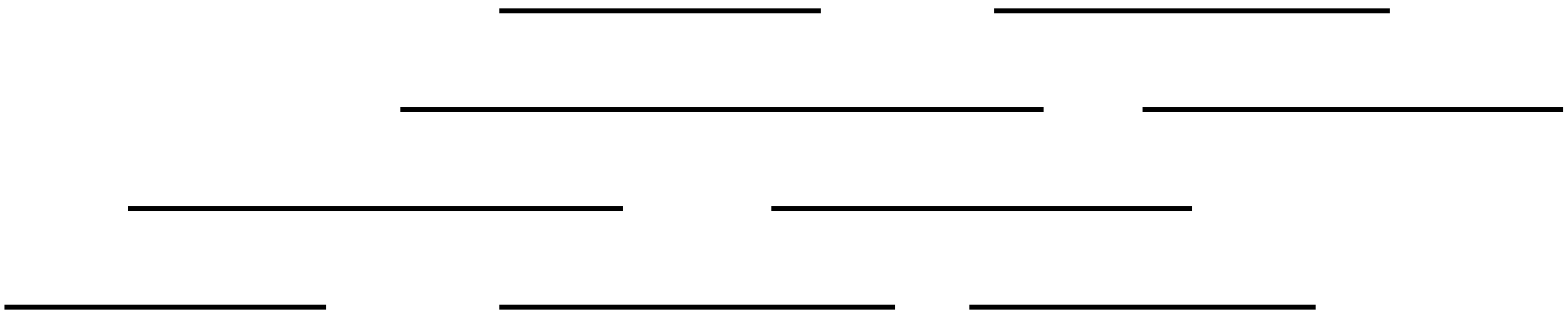
Olgu antud lõigud $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$ $(a_i < b_i)$.

Ütleme, et $[a_i, b_i]$ ja $[a_j, b_j]$ ei lõiku, kui $b_i \leqslant a_j$ või $b_j \leqslant a_i$.

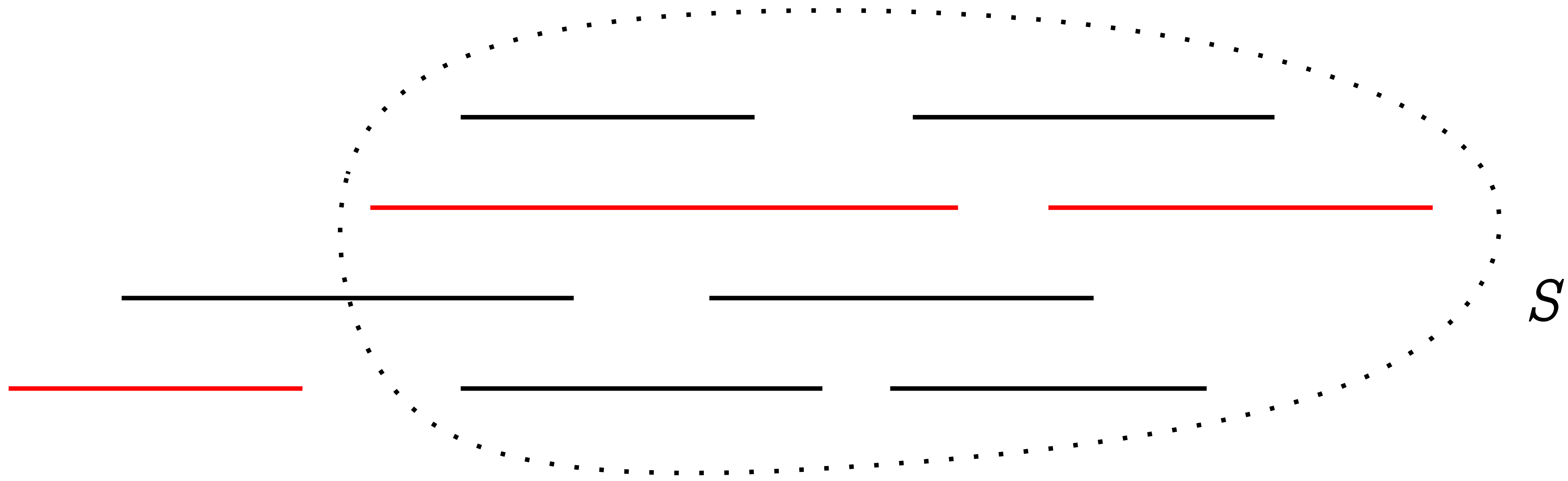
Tahame leida lõikude maksimaalse võimsusega hulga, kus ükski lõik teisega ei lõikuks.



Uurime optimaalse lahenduse struktuuri.

Olgu $[a_{i_1}, b_{i_1}], [a_{i_2}, b_{i_2}], \ldots, [a_{i_m}, b_{i_m}]$ mingi optimaalne lahendus. Seejuures olgu $[a_{i_1}, b_{i_1}]$ kõige vasakpoolsem lõik (s.t. a_{i_1}, b_{i_1} kõige väiksemad).

Olgu S kõigi nende lõikude $[a_j, b_j]$ hulk, kus $a_j \geqslant b_{i_1}$. Siis $[a_{i_2}, b_{i_2}], \ldots, [a_{i_m}, b_{i_m}]$ on maksimaalse võimsusega üksteisega mittelõikuvate lõikude hulk hulgast S.



Tõepoolest, kui leiduks suurema võimsusega üksteisega mittelõikuvate lõikude hulk hulgast S, siis see koos lõiguga $[a_{i_1}, b_{i_1}]$ oleks suurema võimsusega mittelõikuvate lõikude hulk kui $[a_{i_1}, b_{i_1}], [a_{i_2}, b_{i_2}], \ldots, [a_{i_m}, b_{i_m}]$.

Seega on ülesandel optimaalne alamstruktuur.

Iga $i \in \{1, \ldots, n\}$ jaoks olgu

$$S_i = \{[a_j, b_j] \,:\, 1 \leqslant j \leqslant n, a_j \geqslant b_i\},$$

peale selle olgu $S_0 = \{[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]\}$.

Olgu L_i mõni maksimaalse võimsusega üksteisega mittelõikuvate lõikude hulk hulgast S_i. Me tahame leida L_0-i.

Iga $i \in \{0, \ldots, n\}$ jaoks kehtib

$$|L_i| = \begin{cases} 1 + \max\limits_{[a_j, b_j] \in S_i} |L_j|, & \text{kui } S_i \neq \emptyset \\ 0, & \text{kui } S_i = \emptyset \end{cases} .$$

Seega on ülesandel kattuvad alamülesanded.

Väide. Iga $i, j \in \{0, \ldots, n\}$ jaoks $S_i \subseteq S_j$ või $S_j \subseteq S_i$.

Väide. Kui $S_i \subseteq S_j$, siis $|L_j| \geqslant |L_i|$.

Sest kui $S_i \subseteq S_j$, siis ka L_i on mingi mittelõikuvate lõikude hulk hulgast S_j.

Seega avaldises

$$|L_i| = \begin{cases} 1 + \max\limits_{[a_j, b_j] \in S_i} |L_j|, & \text{kui } S_i \neq \emptyset \\ 0, & \text{kui } S_i = \emptyset \end{cases} .$$

on meil ilma $|L_j|$-e välja arvutamatagi teada, millise j korral maksimum saavutatakse — sellise, kus b_j on minimaalne võimalik (teisisõnu: S_j on maksimaalne võimalik).

Olgu $[a_1, b_1], [a_2, b_2], \ldots, [a_n, b_n]$ sorteeritud mittekahanevalt oma parempoolse otspunkti järgi. Loeme, et $a_1, \ldots, a_n$ ja $b_1, \ldots, b_n$ on globaalsed muutujad.

leia_lõik_rek(i) on

if $i > n$ **then return** $\emptyset$
$j := i + 1$
while $j \leqslant n$ **and** $a_j < b_i$ **do** $j := j + 1$
return $\{[a_i, b_i]\} \cup$ leia_lõik_rek(j)

L_0-i leidmiseks: leia_lõik_rek(1).

Selle algoritmi võime ka mitterekursiivselt kirja panna:

$J := \{[a_1, b_1]\}$; $j := 1$
for $i := 2$ **to** n **do**
    if $a_i \geqslant b_j$ **then**
        $J \Leftarrow [a_i, b_i]$
        $j := i$
return J

Keerukus: $\Theta(n)$.

Eeltöö — lõikude $[a_1, b_1], \ldots, [a_n, b_n]$ sorteerimise — keerukus: $\Theta(n \log n)$.

Seda omadust, kus me juba ette teadsime, milline alamülesanne lahendada tuleb, nimetame *ahne valiku omaduseks*.

Ülesanded, millel on optimaalne alamstruktuur ja ahne valiku omadus, on lahendatavad *ahne algoritmiga*.

Ahne algoritmi spetsifitseerimiseks tuleb

- defineerida võimalikud alamülesanded;
- kirjeldada, kuidas mingi alamülesande lahendus avaldub tema alamülesannete lahenduste kaudu;
- paika panna, millise valiku me mingi alamülesande juures esimesena teeme.

Tõestada tuleb,

- et lahenduse kirjeldus alamülesannete lahenduste kaudu on korrektne;
- et etteantud valik viib optimaalse lahenduseni.

Valiku korrektsust võib püüda tõestada järgmisel viisil:

1. Vaatame mingit alamülesannete S_i ja tema mingit optimaalset lahendust L_i. Olgu v lahenduseni L_i viinud lahenduskäigu „esimene“ valik.

2. Näitame, et kui me v asemel teeksime ahne valiku, siis $|L_i|$ hind ei suureneks.

Olgu antud mingi tähestik Σ. Selles tähestikus kirjapandud sõnede esitamiseks arvutis tuleb igale tähele seada vastavusse mingi bitijada.

Olgu antud $\kappa : \Sigma \longrightarrow \{0,1\}^*$. Siis mingit sõne $s = s[1] \cdots s[|s|]$ kujutame bitijadaga $\kappa(s[1])\kappa(s[2]) \cdots \kappa(s[|s|])$.

Üldiselt võib mingi bitijada olla mitmeti dekodeeritav. Näiteks kui $\Sigma = \{\mathrm{a}, \mathrm{b}, \mathrm{c}\}$ ja κ on

x	a	b	c
$\kappa(x)$	0	1	01

siis 011001 võib tähendada nii sõnet „abbaab“ kui ka „cbac“ (ja teisigi).

Selleks, et iga bitijada oleks üheselt dekodeeritav, piisab kui mitte ühegi $x, y \in \Sigma$ jaoks $\kappa(x) \sqsubset \kappa(y)$. Selliseid kodeerimisviise nimetatakse *prefikskoodideks*.

Näide prefikskoodist:

x	a	b	c	d	e	f
$\kappa(x)$	0	101	100	111	1101	1100

Muuhulgas on prefikskoodid kõik koodid, kus kõik koodisõnad on võrdse pikkusega.

Suvalise koodi võib esitada (märgendatud) kahendpuuna, kus

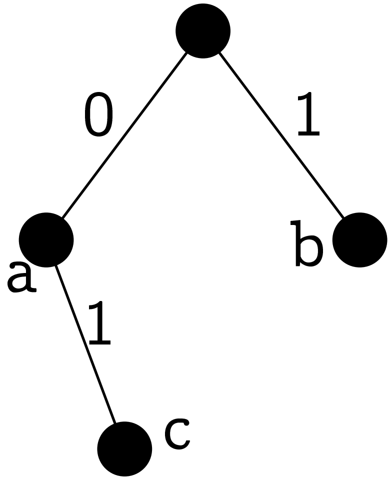
- mingit sisemist tippu tema vasaku alampuu juurega ühendav serv on märgendatud 0-ga, teda tema parema alampuu juurega ühendav serv aga 1-ga;
- puus on olemas kõigile koodisõnadele (ja nende prefiksitele) vastavad teed juurest mingisse tippu, teisi teid ei ole;
- puu tipp on märgendatud mingi tähega, juhul kui selles tipus lõppev tee vastab selle tähe koodisõnale.

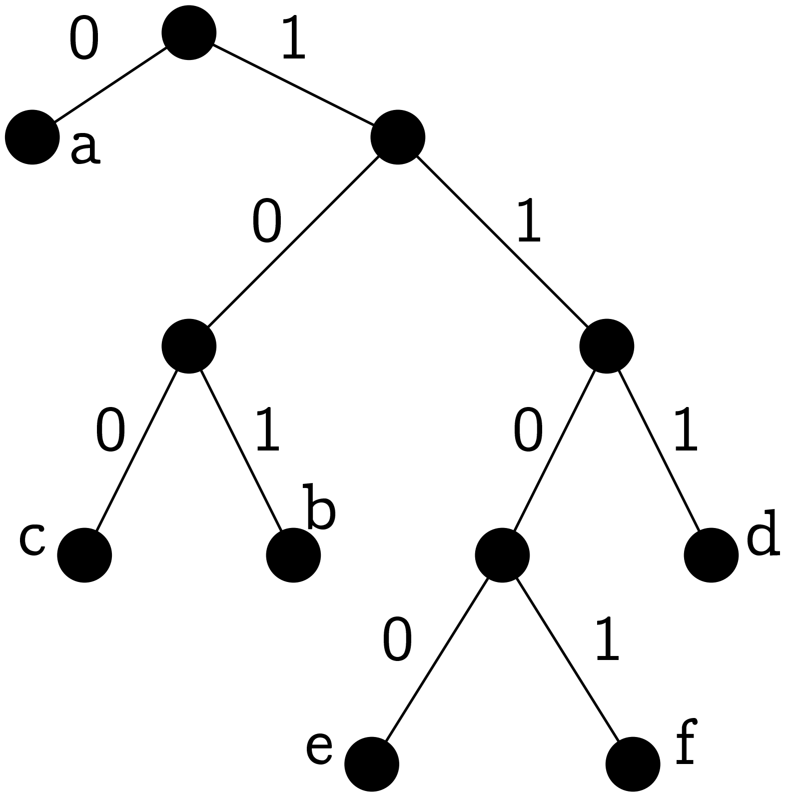
Kood on prefikskood parajasti siis, kui talle vastavas kahendpuus on ainult lehed märgendatud tähtedega.

Näited:

x	a	b	c	d	e	f
$\kappa(x)$	0	101	100	111	1101	1100

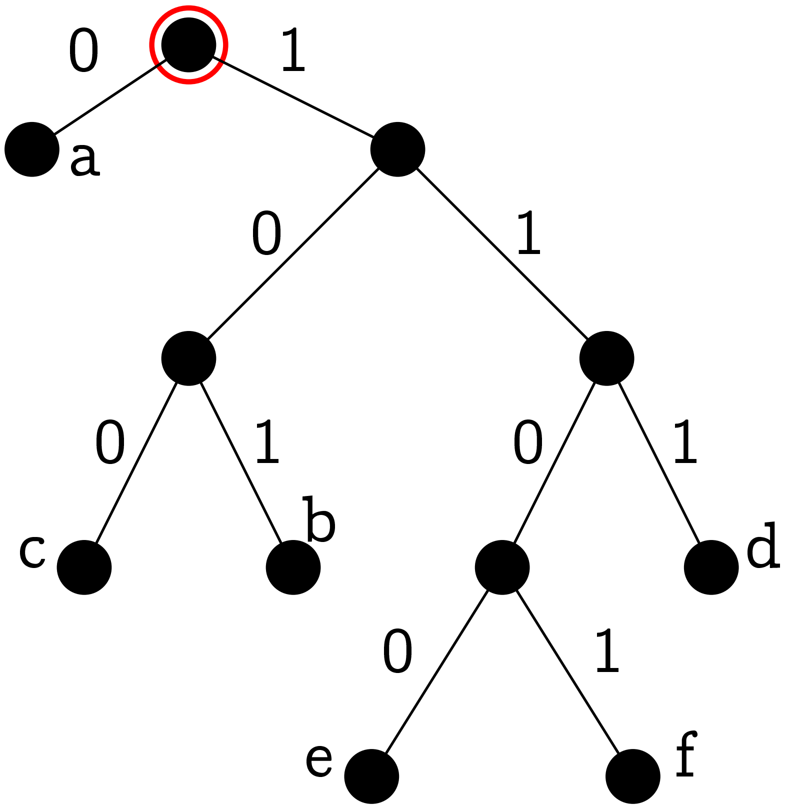
x	a	b	c
$\kappa(x)$	0	1	01





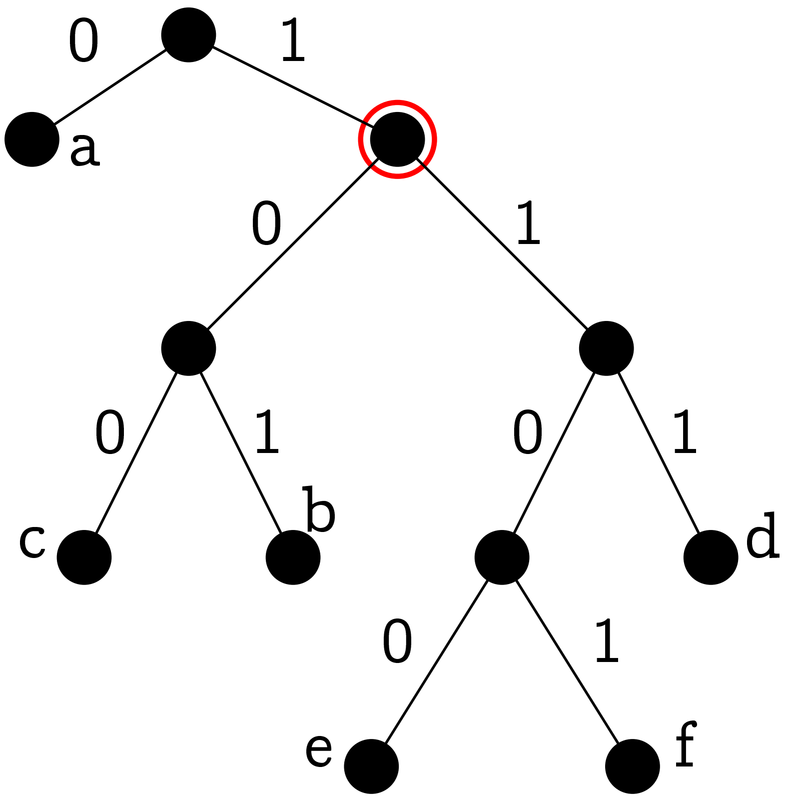
Näide prefikskoodi dekodeerimisest:

10111010111101101



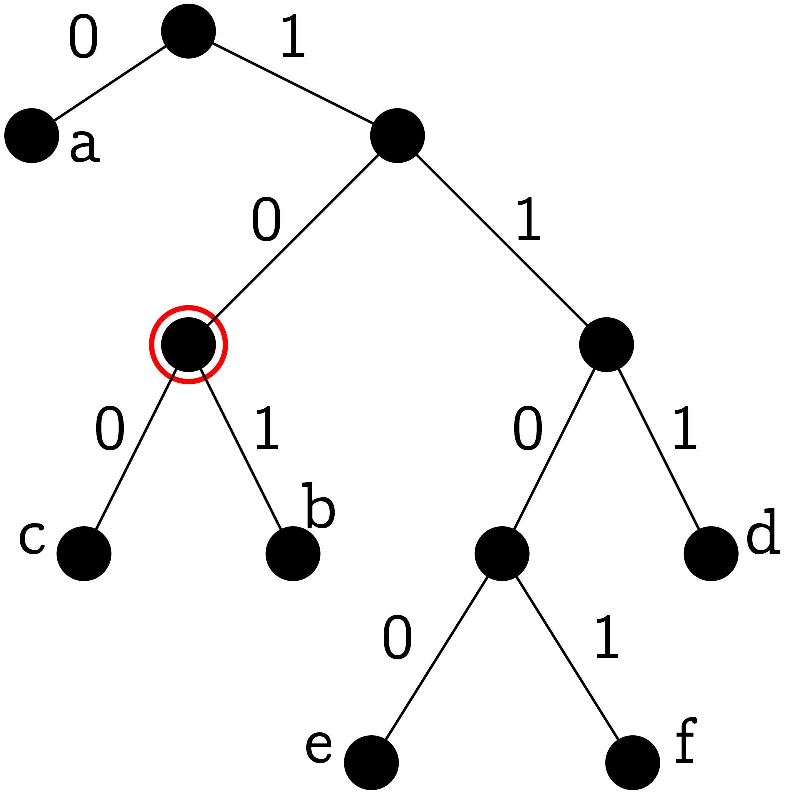
Näide prefikskoodi dekodeerimisest:

1011101011101101



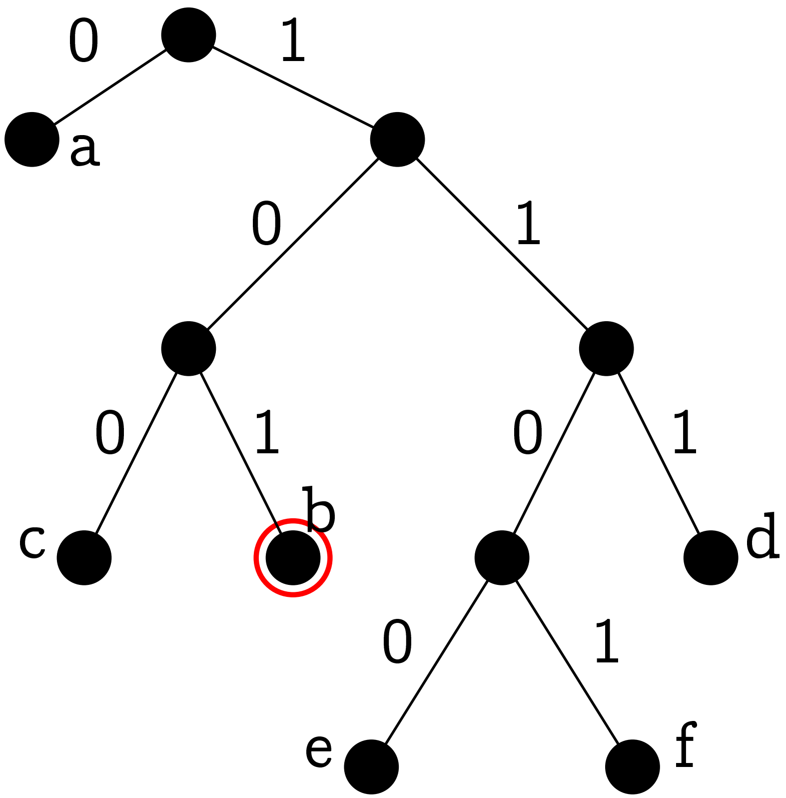
Näide prefikskoodi dekodeerimisest:

10111010111101101



Näide prefikskoodi dekodeerimisest:

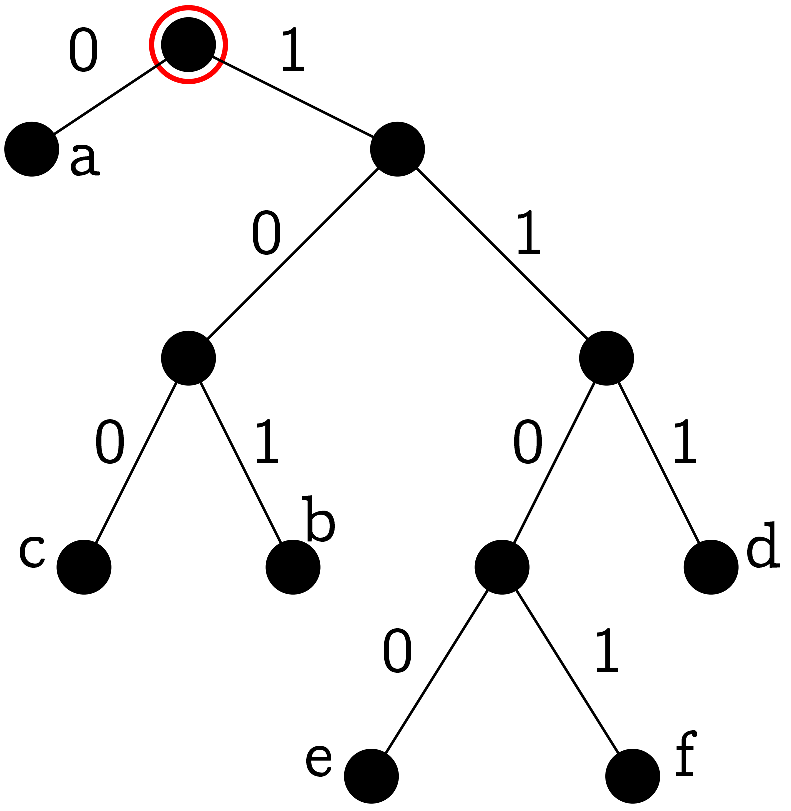
1011101011101101



b

Näide prefikskoodi dekodeerimisest:

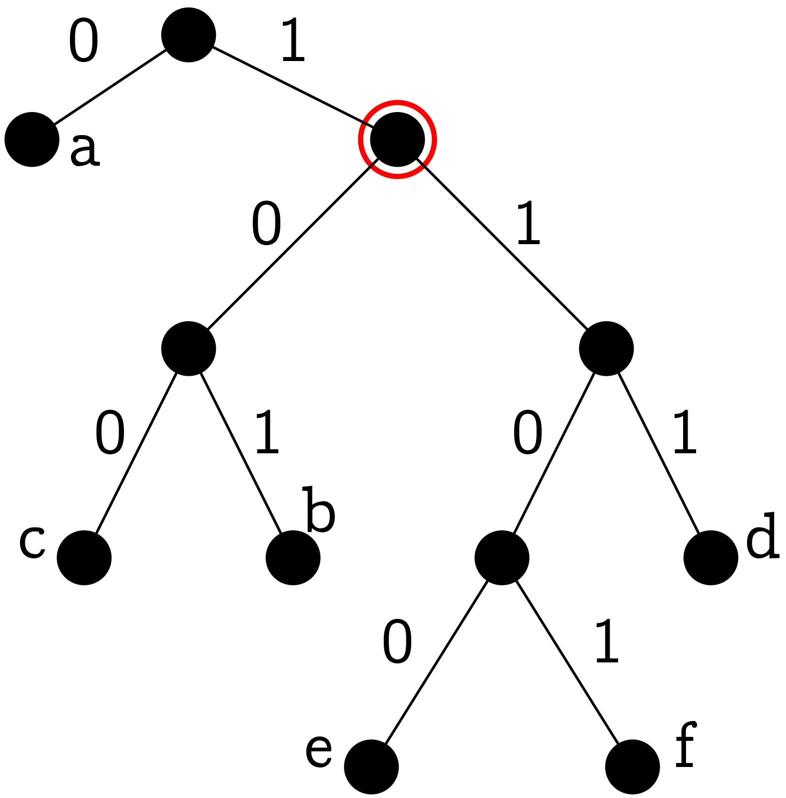
1011101011101101



b

Näide prefikskoodi dekodeerimisest:

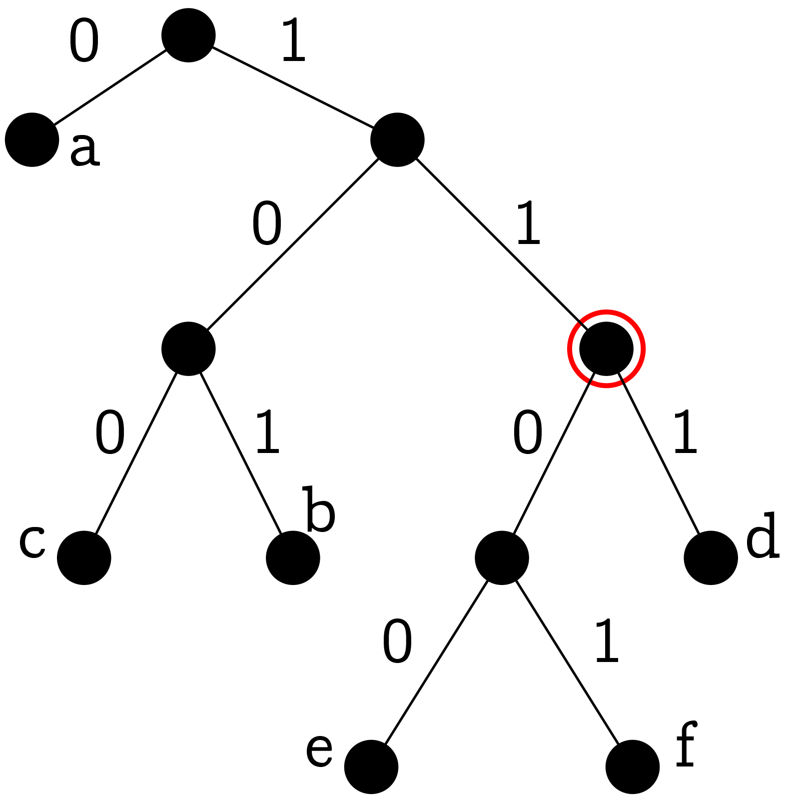
1011101011101101



b

Näide prefikskoodi dekodeerimisest:

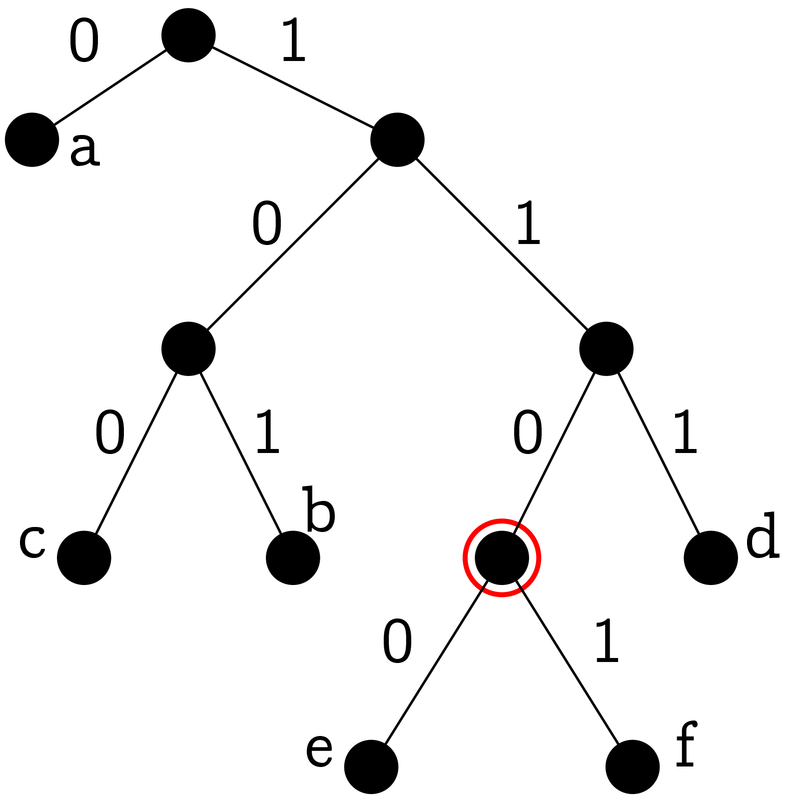
1011101011101101



b

Näide prefikskoodi dekodeerimisest:

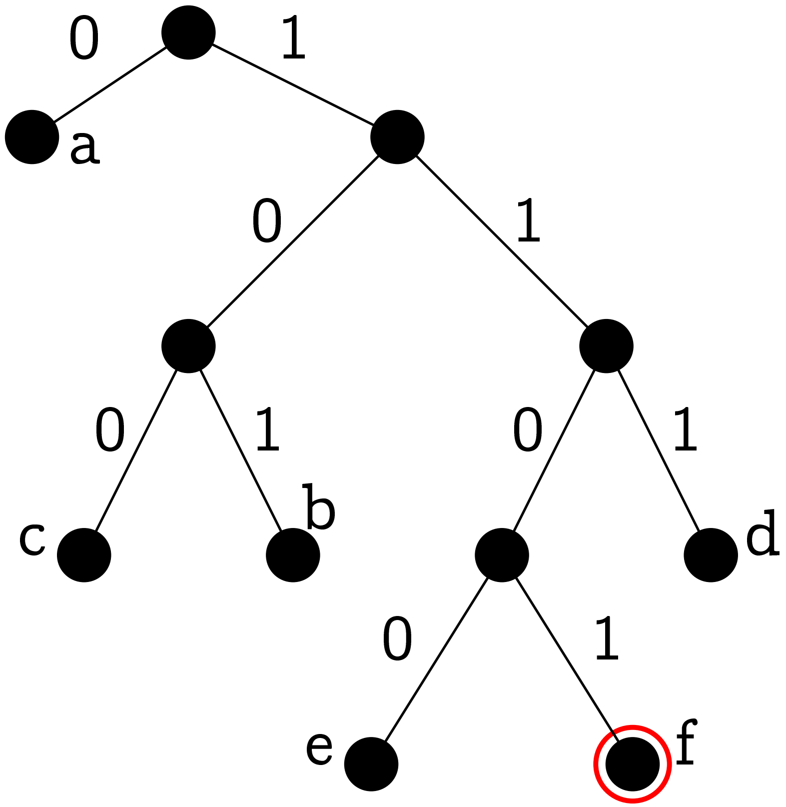
1011101011101101



b

Näide prefikskoodi dekodeerimisest:

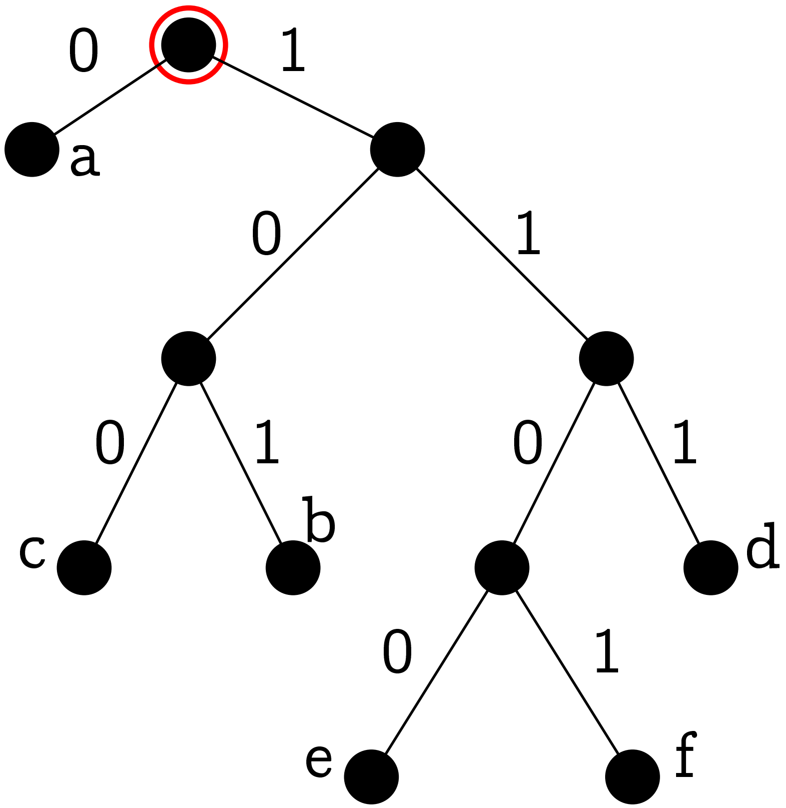
10111101011101101



bf

Näide prefikskoodi dekodeerimisest:

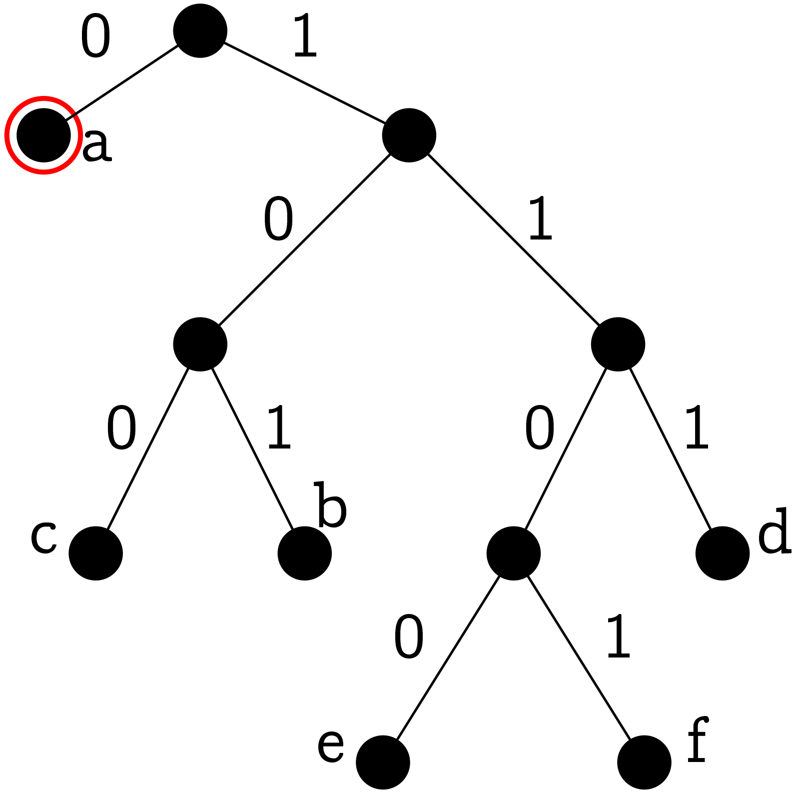
1011101011101101



bf

Näide prefikskoodi dekodeerimisest:

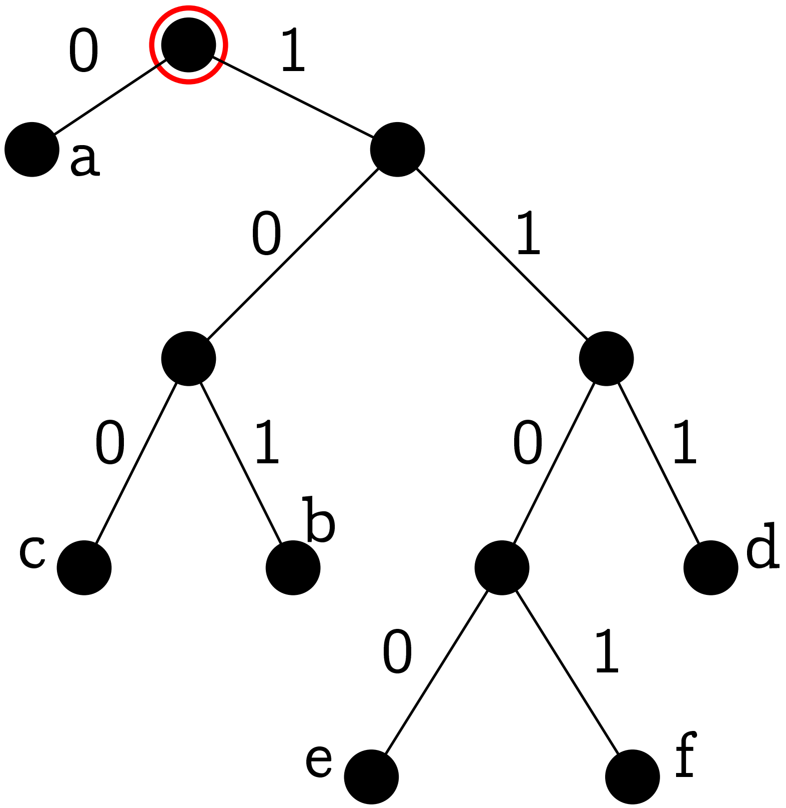
1011101011101101



bfa

Näide prefikskoodi dekodeerimisest:

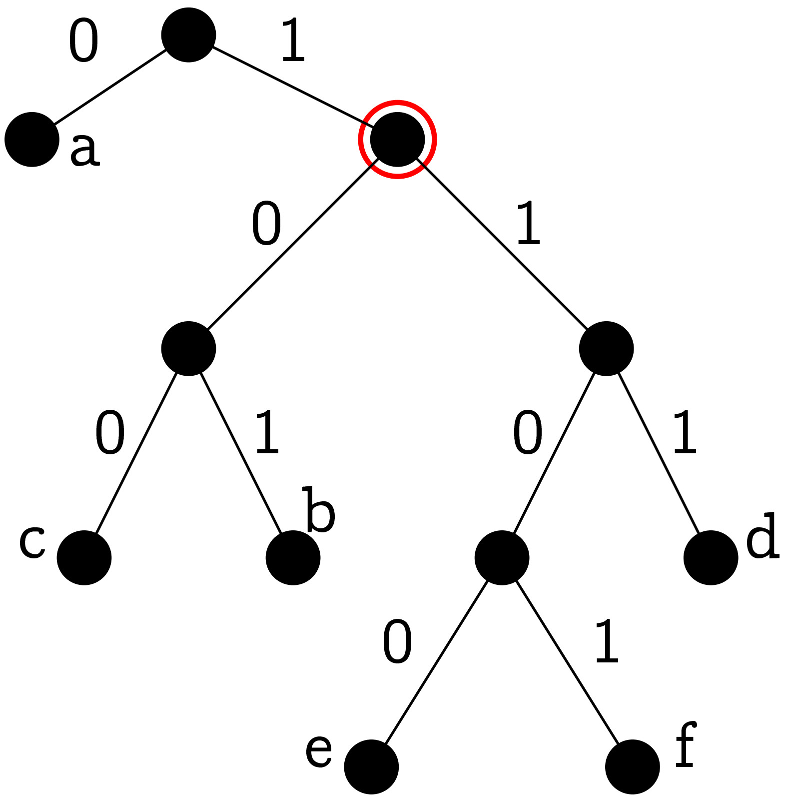
10111010 11101101



bfa

Näide prefikskoodi dekodeerimisest:

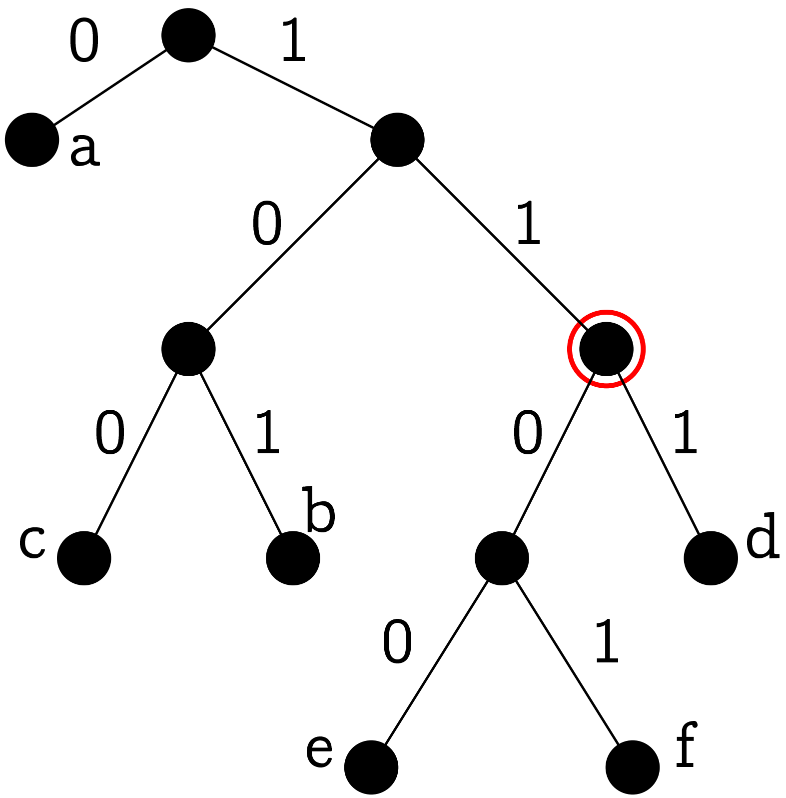
1011101011101101



bfa

Näide prefikskoodi dekodeerimisest:

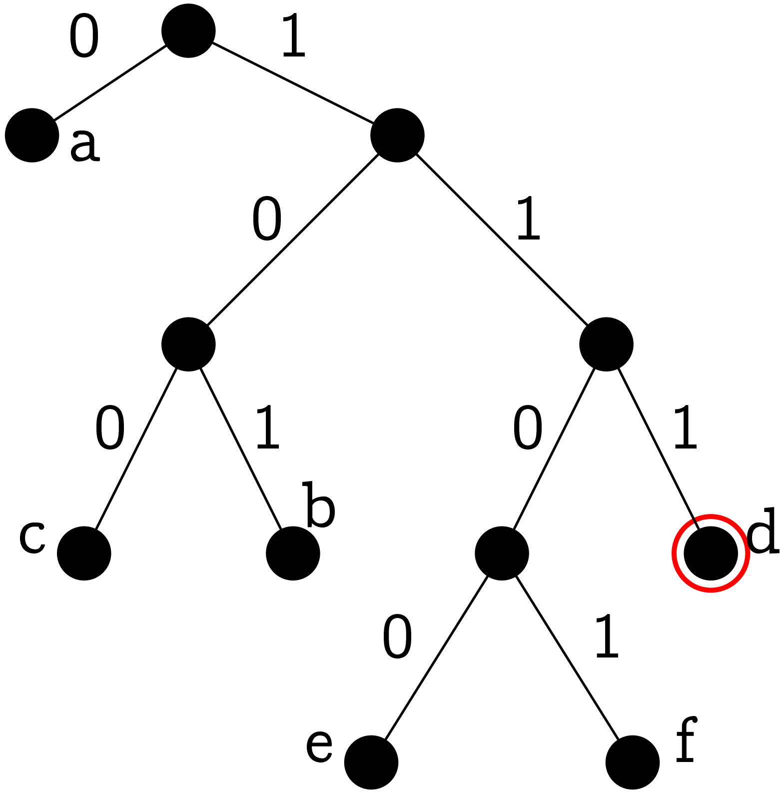
1011101011101101



bfa

Näide prefikskoodi dekodeerimisest:

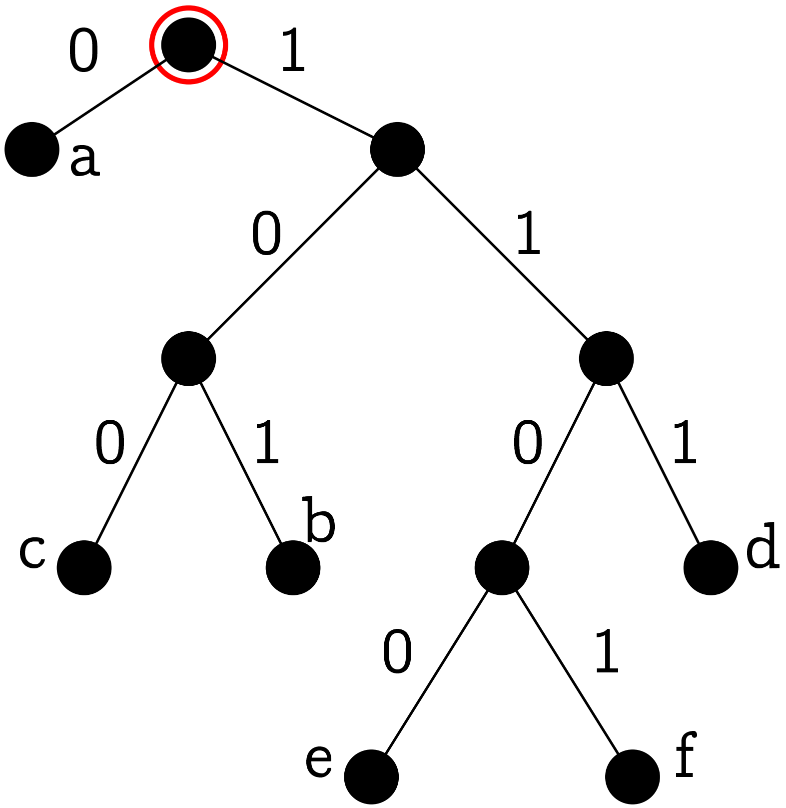
1011101011101101



bfad

Näide prefikskoodi dekodeerimisest:

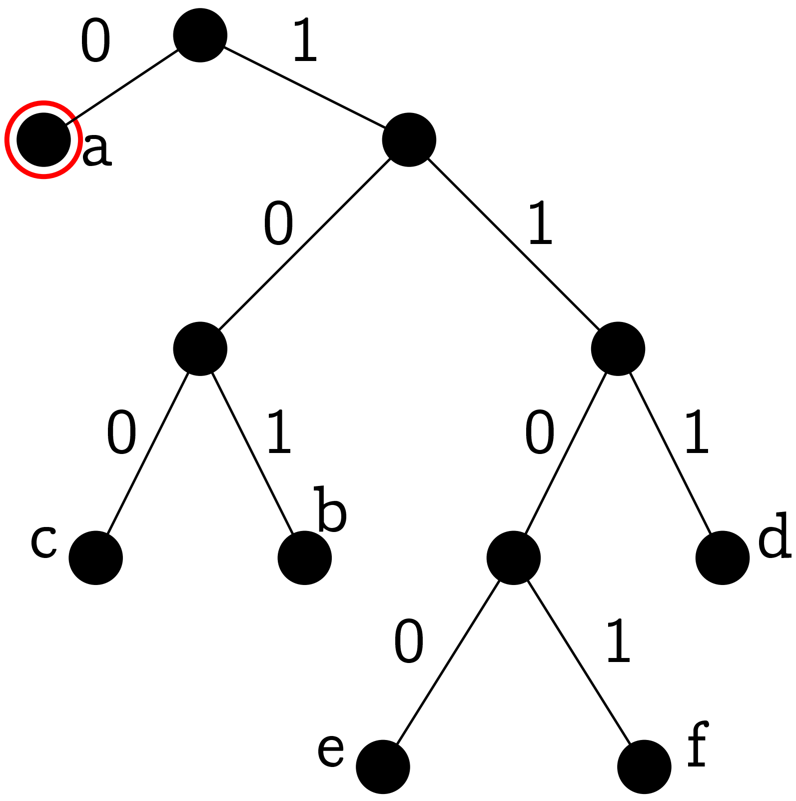
1011101011101101



bfad

Näide prefikskoodi dekodeerimisest:

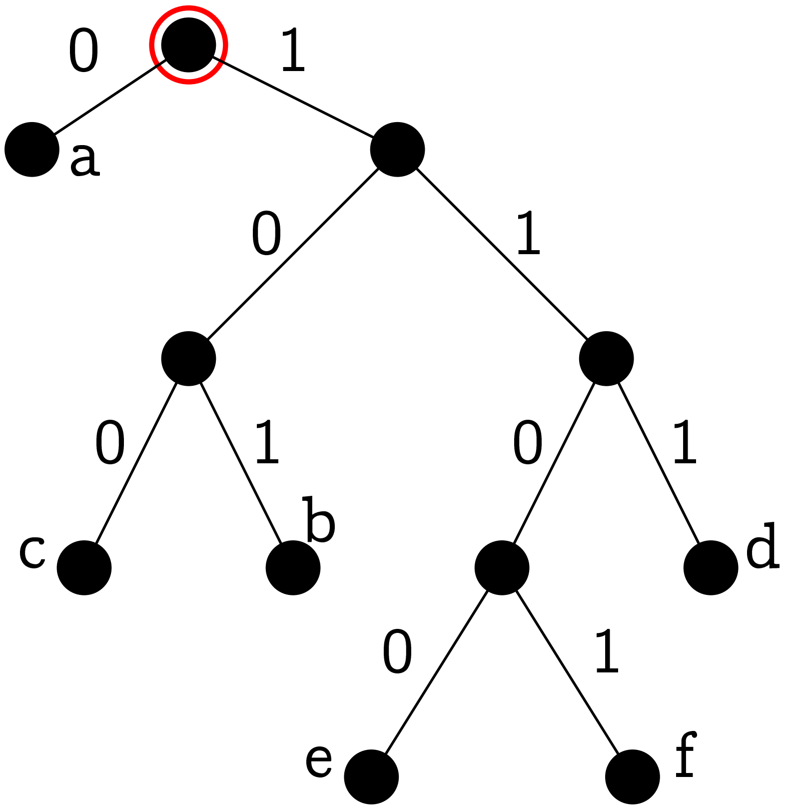
101110101110 1101



bfada

Näide prefikskoodi dekodeerimisest:

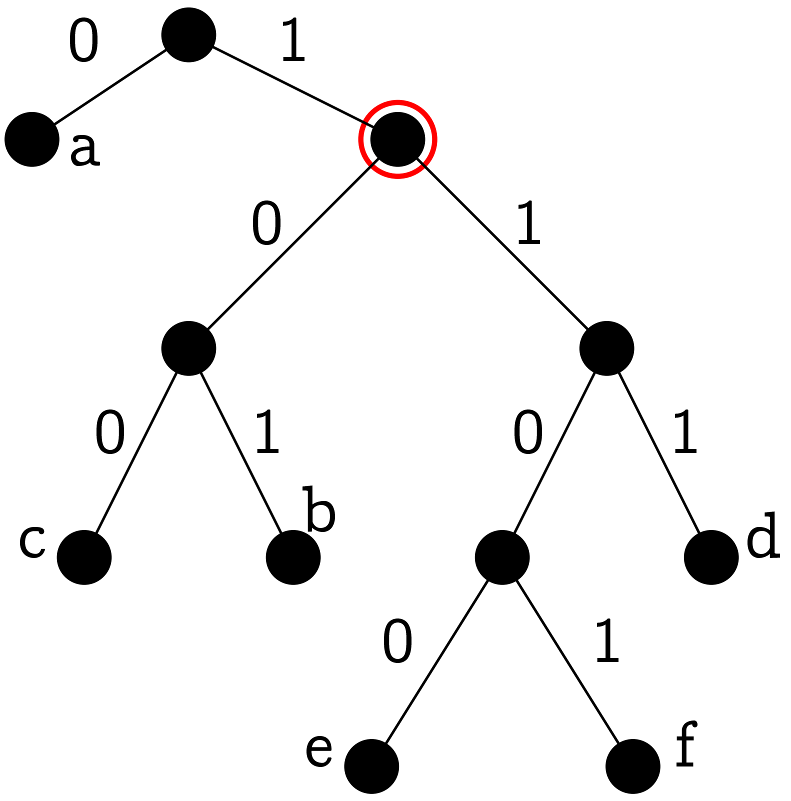
10111010111011101



bfada

Näide prefikskoodi dekodeerimisest:

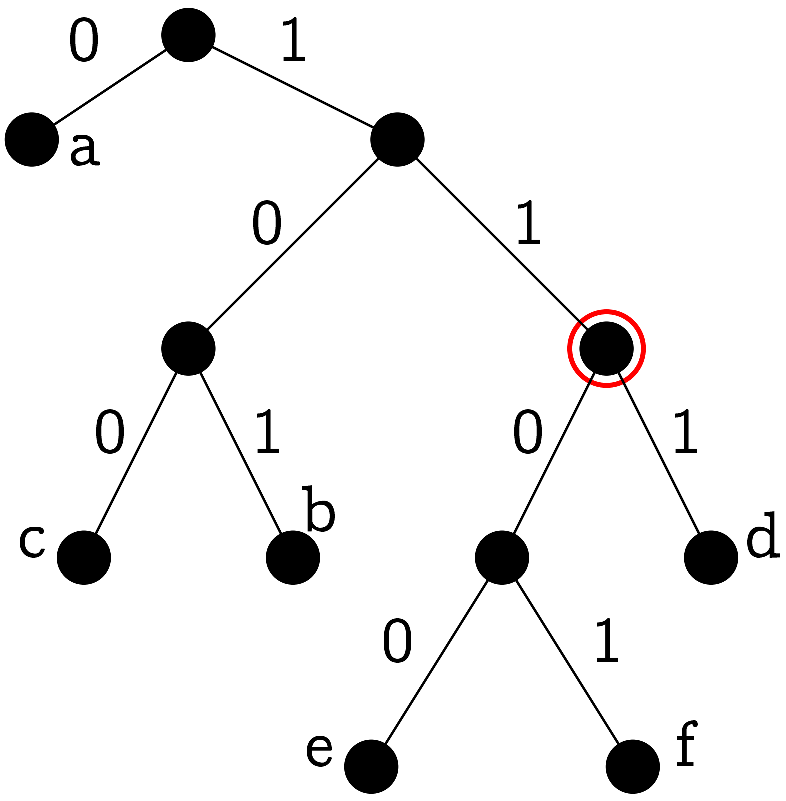
1011101011101101



bfada

Näide prefikskoodi dekodeerimisest:

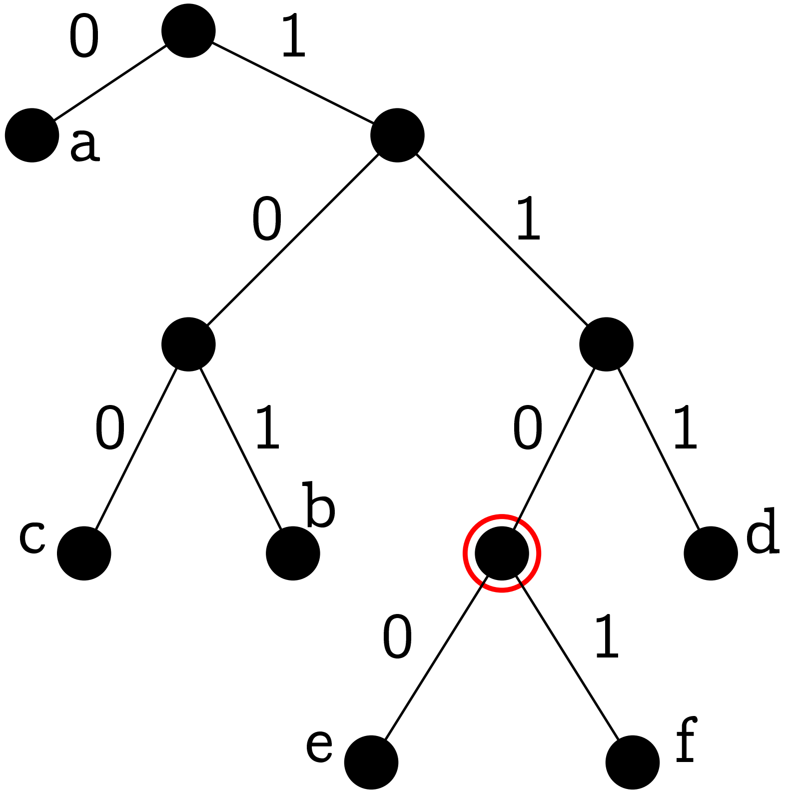
10111010111011101



bfada

Näide prefikskoodi dekodeerimisest:

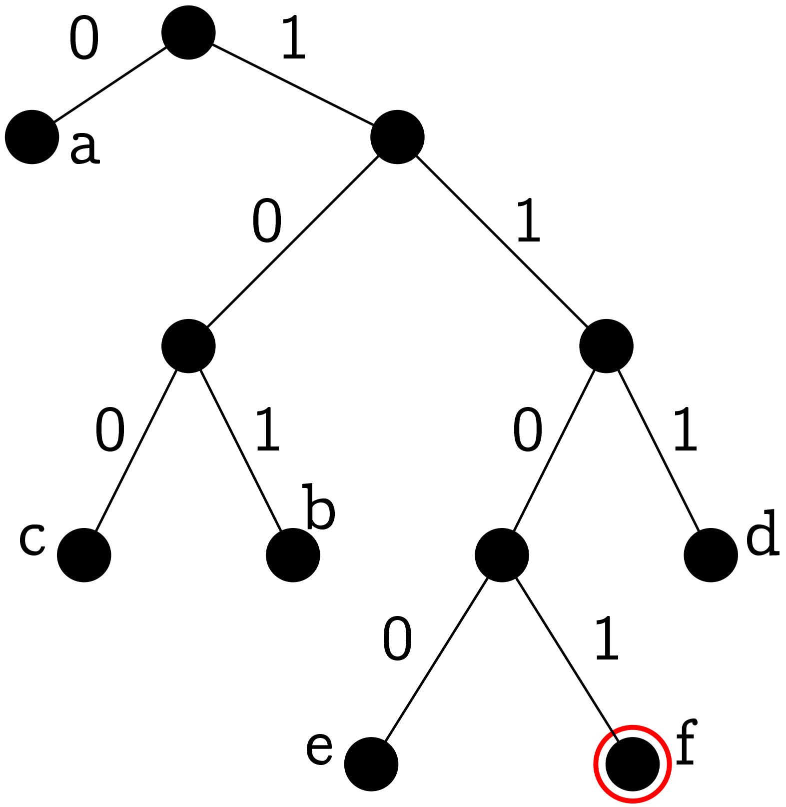
10111010111011101



bfada

Näide prefikskoodi dekodeerimisest:

1011101011101101



bfadaf

Olgu meil teada kõigi tähtede esinemiste arvud tekstis. Tahame leida prefikskoodi, mis teksti kodeeringu pikkuse minimeeriks.

Iga tähe $x \in \Sigma$ jaoks olgu $\varphi(x)$ tema esinemiste arv tekstis.

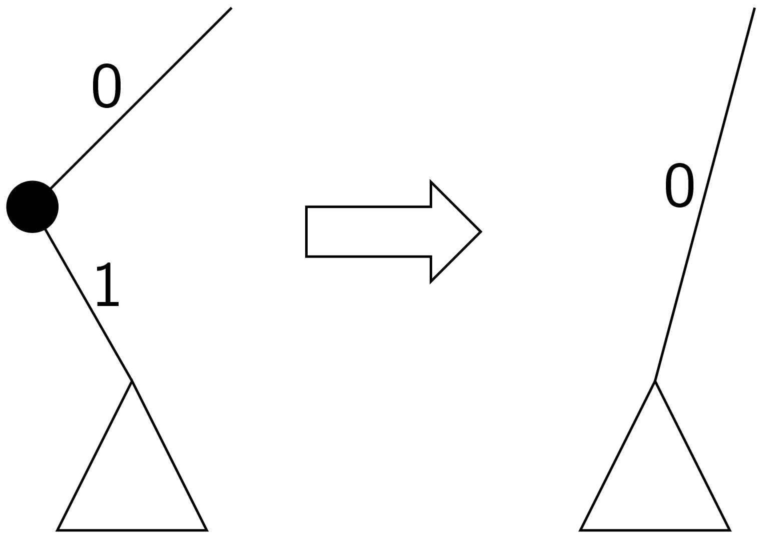
Defineerime koodi κ hinna w_κ järgmiselt:

$$w_\kappa = \sum_{x \in \Sigma} \varphi(x) |\kappa(x)|,$$

me tahame seda minimeerida. Olgu T_κ koodile κ vastav puu.

Olgu meil antud mingi optimaalne kood κ ja sellele vastav puu T_κ. Uurime selle puu struktuuri.

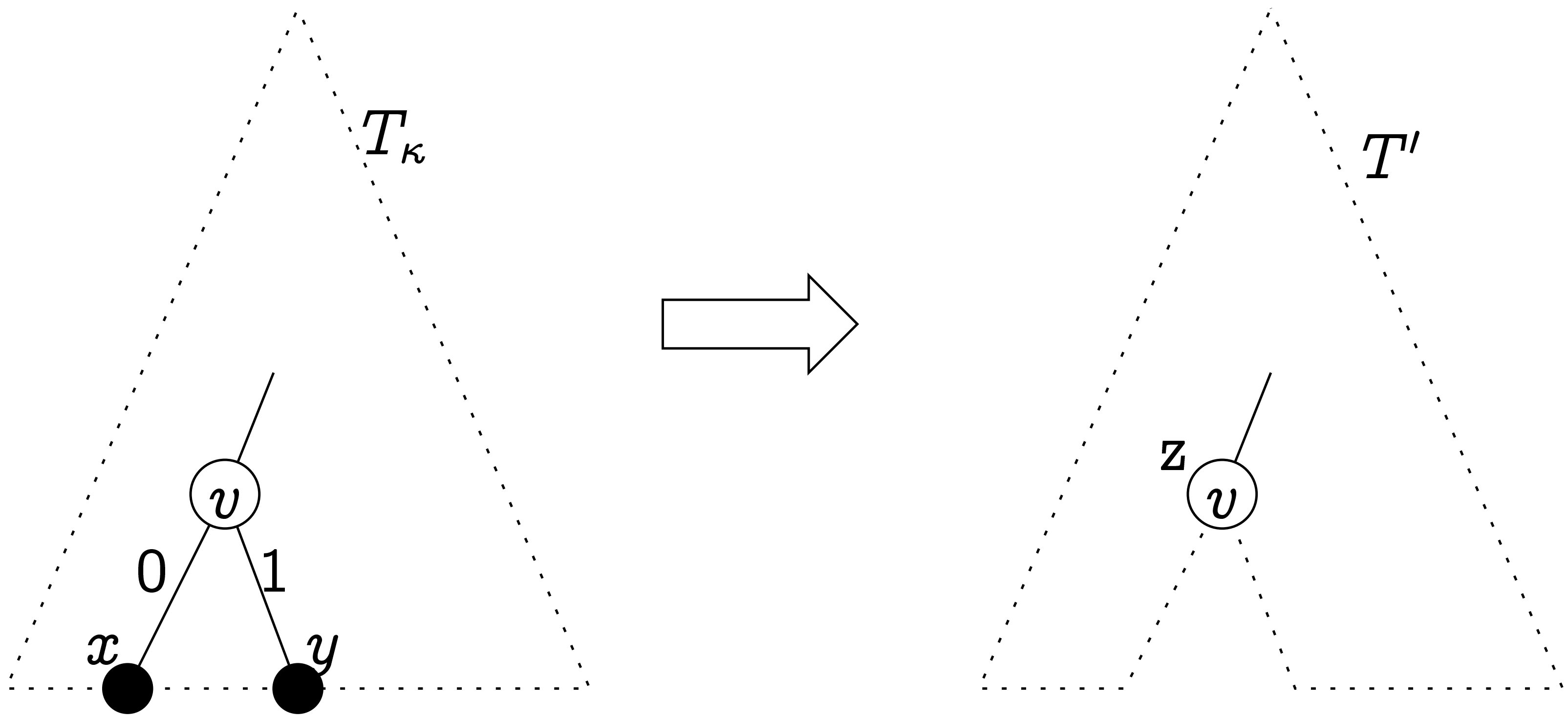
Selle puu igal sisemisel tipul on kaks alluvat. Kui oleks üksainus, siis võiks selle tipu „välja õgvendada“. Selle tipu all olevate lehtede märgendite koodisõnad lüheneksid selle operatsiooni käigus.



Olgu v mingi sisemine tipp, mille mõlemad alluvad on lehed. Olgu x ja y nende lehtede märgendid.

Olgu z täht, mis ei kuulu tähestikku Σ. Olgu $\Sigma' = \Sigma \backslash \{x, y\} \cup \{\mathtt{z}\}$.

Vaatame puud T' ja talle vastavat koodi κ' tähestiku Σ' jaoks, mis on defineeritud järgmiselt:



T_κ
v
0
1
x
y
T'
z
v

Kui defineerime $\varphi(\mathrm{z}) = \varphi(x) + \varphi(y)$ (s.t. asendame nii x-i kui ka y-i z-ga), siis $w_{\kappa'} = w_\kappa - \varphi(x) - \varphi(y)$.

κ' peab olema optimaalne kood Σ' jaoks (tähtede esinemiste arvud on antud φ-ga), sest suvaline kood Σ' jaoks on muudetav koodiks Σ jaoks, nii et hind suureneb ainult $\varphi(x) + \varphi(y)$ võrra.

Ülesandel on optimaalne alamstruktuur. Alamülesanne on määratud tähestikuga, kus on vähem elemente kui Σ-s, ning tähtede esinemiste arvudega.

Võimalikke alamülesandeid on palju. . .

Alamülesande valikuks tuleb fikseerida kaks tähte. Need kaks tähte on tekkivas koodipuus naabrid.

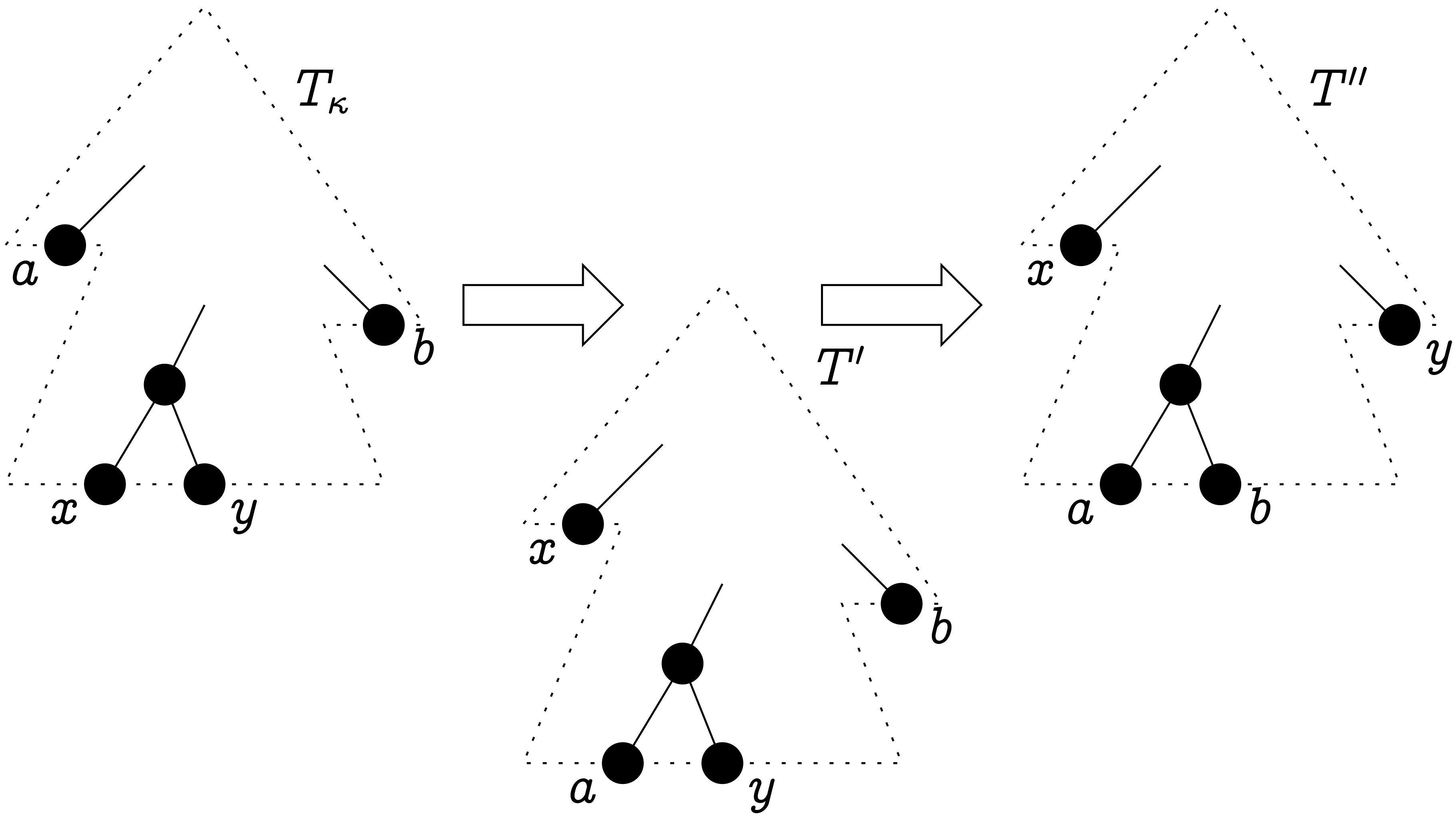
Ahne valiku omaduse täidetud olemiseks on vaja (ilma alamülesandeid lahendamata) teada kahte tähte, nii et need mõnes optimaalses koodipuus naabrid oleksid.

Nendeks kaheks täheks sobivad kaks kõigi harvemini esinevat tähte. Näitame seda.

Olgu κ mingi optimaalne kood ja T_κ sellele vastav puu. Olgu u ja v selle puu naaberlehed, mis asuvad juurest nii kaugel kui võimalik. Olgu $x, y \in \Sigma$ tippude u ja v märgendid, loeme et $\varphi(x) \leqslant \varphi(y)$.

Olgu $a, b \in \Sigma$ kaks kõige harvemini esinevat tähte, olgu $\varphi(a) \leqslant \varphi(b)$. Muuhulgas siis $\varphi(a) \leqslant \varphi(x)$ ja $\varphi(b) \leqslant \varphi(y)$.

Olgu T' saadud T_κ-st, vahetades seal märgendid a ja x; olgu κ' vastav kood. Olgu T'' saadud T'-st, vahetades seal märgendid b ja y; olgu κ'' vastav kood.



T_κ
a
b
x
y
T'
x
b
a
y
T''
x
y
a
b

$$
\begin{aligned}
w_\kappa - w_{\kappa'} &= \varphi(a)\big(|\kappa(a)| - |\kappa'(a)|\big) + \varphi(x)\big(|\kappa(x)| - |\kappa'(x)|\big) \\
&= \varphi(a)\big(|\kappa(a)| - |\kappa(x)|\big) + \varphi(x)\big(|\kappa(x)| - |\kappa(a)|\big) \\
&= \big(\varphi(x) - \varphi(a)\big)\big(|\kappa(x)| - |\kappa(a)|\big) \\
&\geqslant 0 \cdot 0 = 0 \\
w_{\kappa'} - w_{\kappa''} &= \varphi(b)\big(|\kappa'(b)| - |\kappa''(b)|\big) + \varphi(y)\big(|\kappa'(y)| - |\kappa''(y)|\big) \\
&= \varphi(b)\big(|\kappa'(b)| - |\kappa'(y)|\big) + \varphi(y)\big(|\kappa'(y)| - |\kappa'(b)|\big) \\
&= \big(\varphi(y) - \varphi(b)\big)\big(|\kappa'(y)| - |\kappa'(b)|\big) \\
&\geqslant 0 \cdot 0 = 0
\end{aligned}
$$

Seega $w_\kappa \geqslant w_{\kappa'} \geqslant w_{\kappa''}$. Et κ on optimaalne kood, siis $w_\kappa = w_{\kappa''}$, seega on ka κ'' optimaalne kood.

Oleme leidnud, et optimaalset koodi on võimalik konstrueerida ahne algoritmiga.

Koodipuud konstrueeritakse lehtedest juure poole. Algoritmi igal iteratsioonil on antud (konstrueeritava koodipuu alam)puude hulk $\mathcal{T}$. Igal puul hulgast $\mathcal{T}$ on antud esinemissagedus.

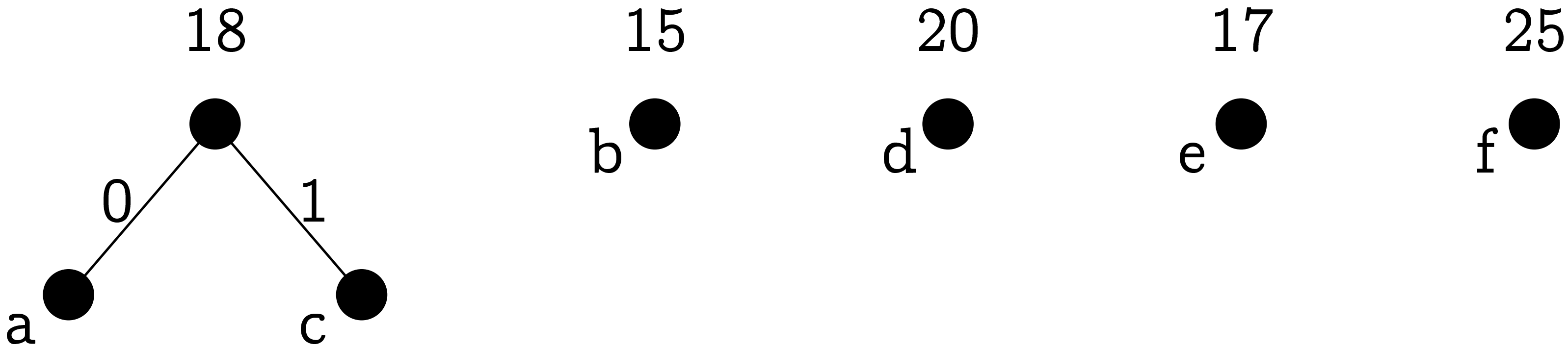
Algoritmi töö alguses on $\mathcal{T}$-s $|\Sigma|$ ühetipulist puud, mille ainsad tipud on märgendatud Σ tähtedega. Iga puu esinemissageduseks on vastava tähe esinemissagedus.

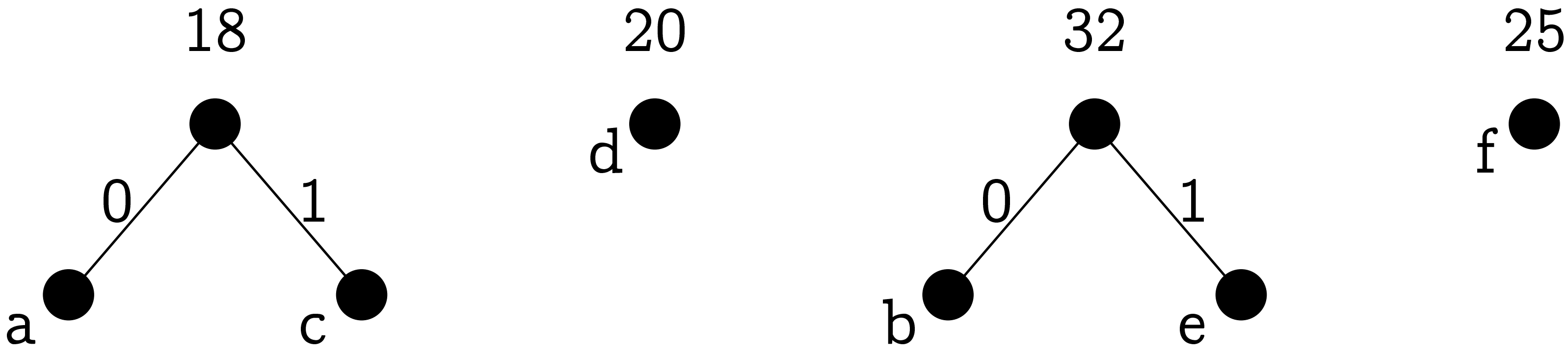
Ühel iteratsioonil:

- Leia $\mathcal{T}$-st kaks vähima sagedusega puud T_1 ja T_2 ning eemalda nad.
- Konstrueeri uus puu T:
  - Juurtipuks võta mingi uus tipp v.
  - Vasakuks alampuuks võta T_1 ja paremaks T_2. Servad v-st T_1 ja T_2 juurtippudesse märgenda vastavalt 0 ja 1-ga.
- Lisa T hulka $\mathcal{T}$.

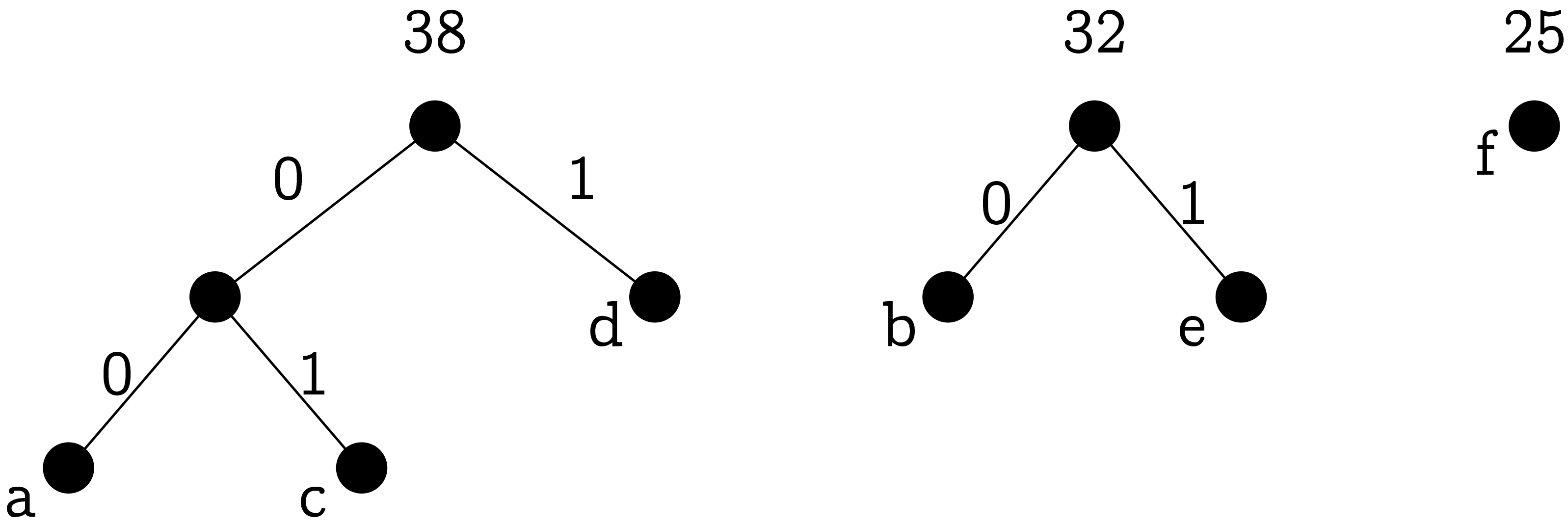
Programm lõpetab töö, kui hulka $\mathcal{T}$ on jäänud üksainus puu. See puu võetaksegi koodipuuks.


10
15
8
20
17
25
a
b
c
d
e
f

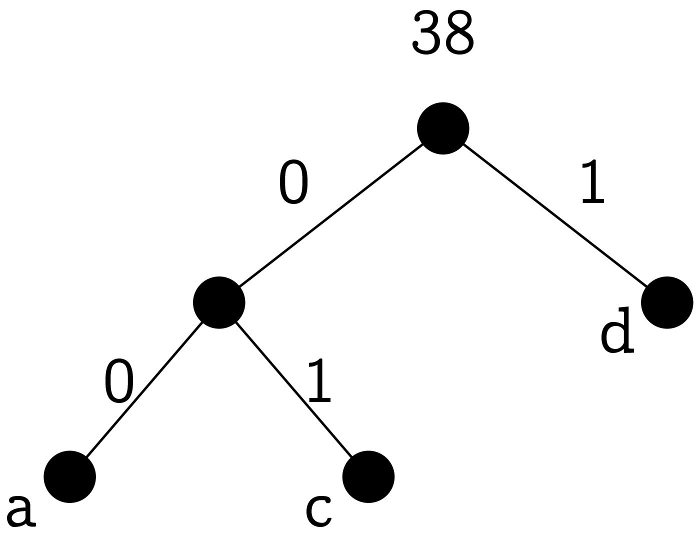

18
0
1
a
c
15
b
20
d
17
e
25
f

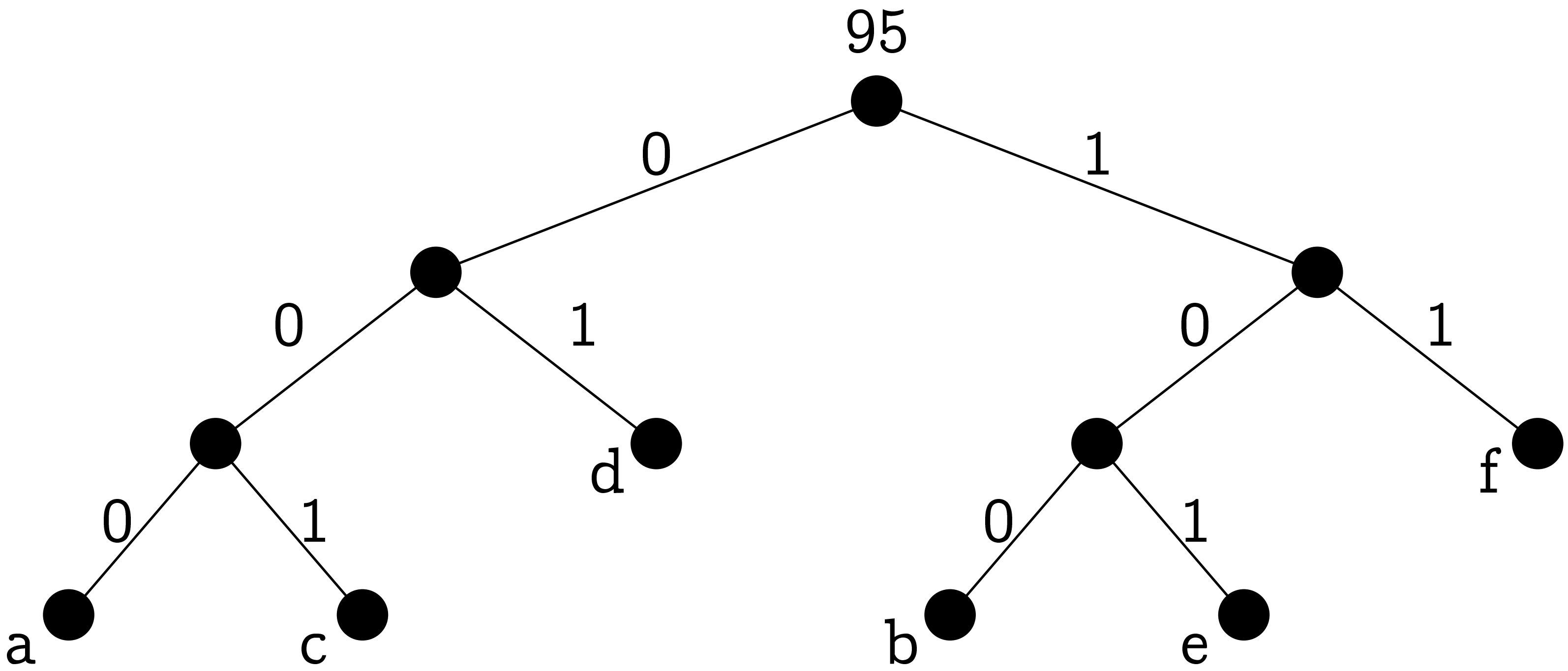


18
0
1
a
c
20
d
32
0
1
b
e
25
f



38
0
1
0
1
a
c
d
32
0
1
b
e
25
f

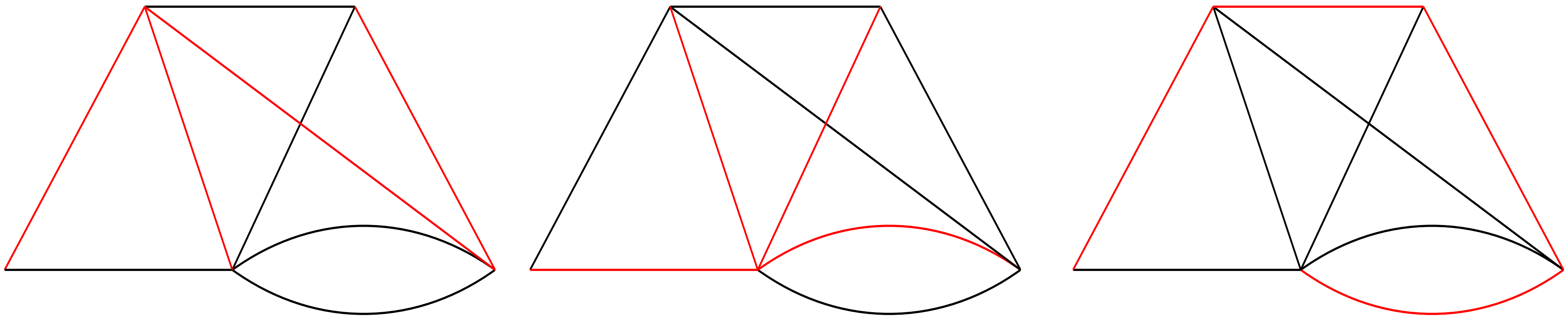

38
0
1
0
1
a
c
d
57
0
1
0
1
b
e
f


95
0
1
0
1
0
1
d
f
0
1
0
1
a
c
b
e

Eeltoodud algoritmiga leitud koode nimetatakse *Huffmani koodideks*.

(Juureta) puu on sidus tsükliteta graaf.

Graafi *aluspuu* on selle graafi alamgraaf, mis sisaldab kõiki selle graafi tippe ja on puu.



Graafi aluspuud kujutavad endast „odavaimaid võimalusi“ selle graafi tippude kokkuühendamiseks.

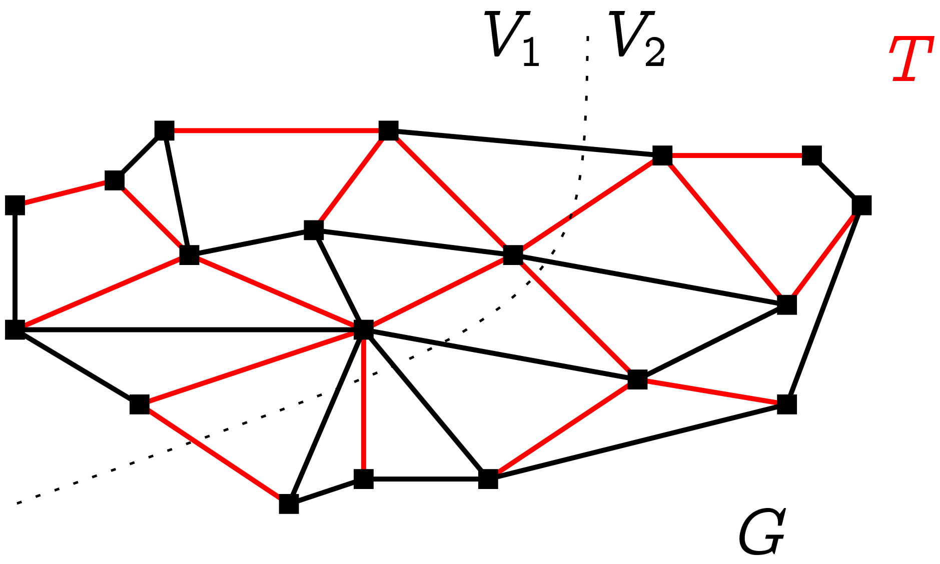
Etteantud graafi aluspuu on määratud selle puu servade hulgaga.

Vaatame sidusat graafi $G = (V, E)$. Olgu igale selle graafi servale $e \in E$ omistatud kaal $w(e) \in \mathbb{R}^+$.

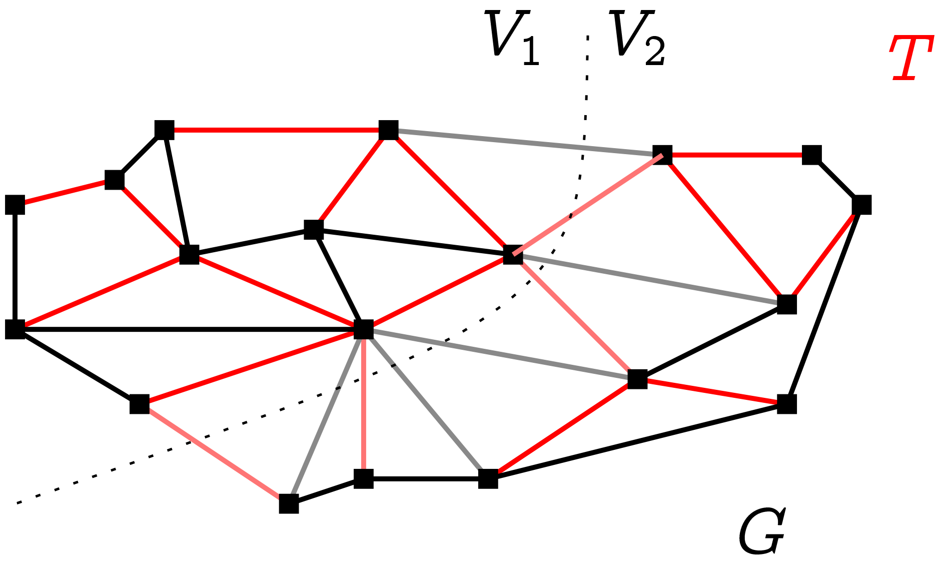
Graafi alamgraafi kaalu defineerime kui kõigi selle alamgraafi servade kaalude summa.

Ülesanne: leida graafi minimaalse kaaluga aluspuu.

Uurime ülesande struktuuri. Olgu meil antud graafi $G = (V, E)$ mingi minimaalse kaaluga aluspuu T. Olgu $V_1, V_2 \subseteq V$ sellised, et $V_1 \cap V_2 = \emptyset$ ja $V_1 \cup V_2 = V$ (s.t. V_1, V_2 on V tükeldus).



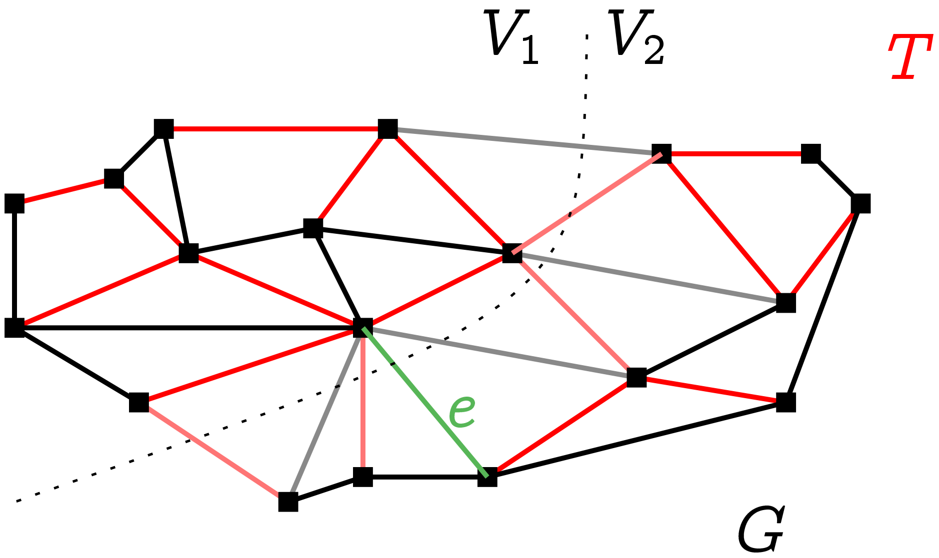
Vaatame kõiki servi, mille üks otstipp on hulgas V_1 ja teine otstipp hulgas V_2.



Näitame, et puud T on võimalik muuta nii, et tema kaal ei suureneks ning ta sisaldaks neist servadest vähima kaaluga serva.

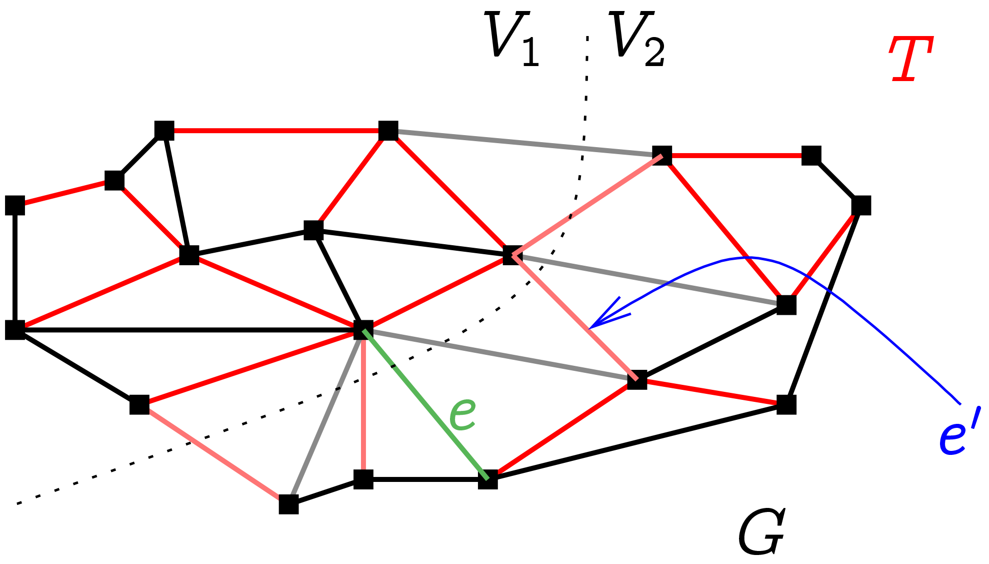
Kui vähima kaaluga serv e juba kuulub puusse T, siis pole tarvis midagi teha.

Kui ei, siis. . .

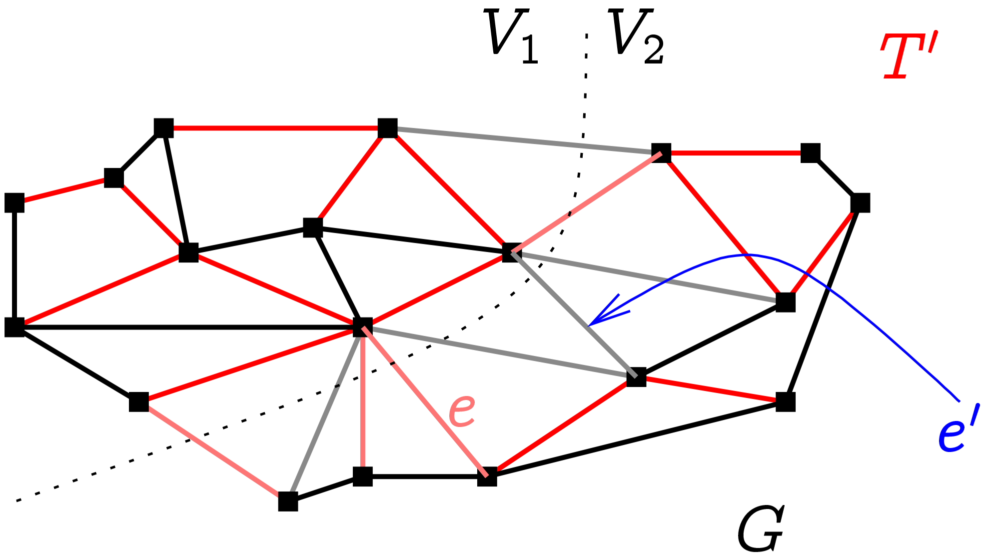


on graafis $T + e$ tsükkel. Suvalise serva kustutamine sellest tsüklist annab meile mingi aluspuu.

Selles tsüklis peab olema peale e veel mõni serv, mis on poolte V_1 ja V_2 vahel. See serv e' kuulub puusse T ja $w(e') \geqslant w(e)$.

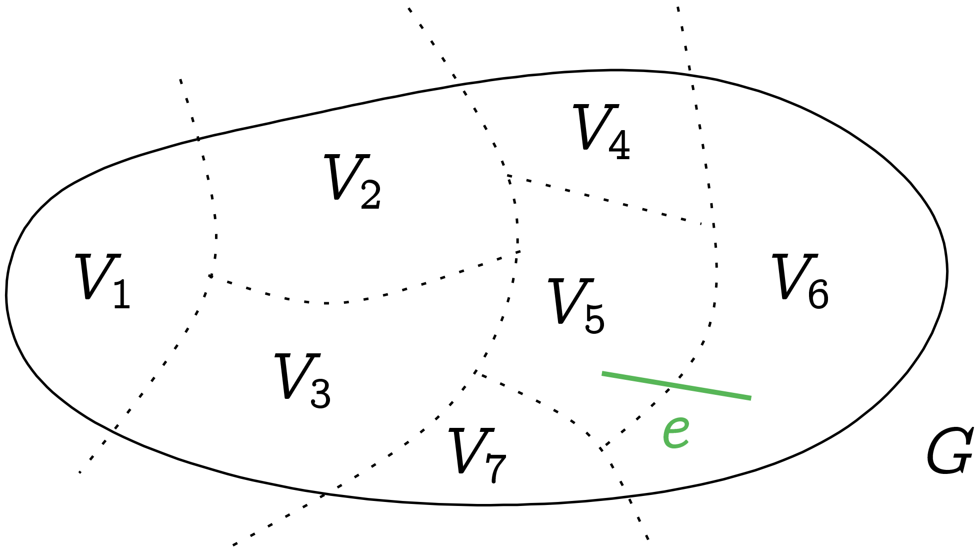


Muudame puud T nii, et eemaldame serva e' ja lisame serva e. Tähistame tulemust T'-ga.



Puu T' kaal pole suurem kui T kaal, sest e kaal pole suurem kui e' kaal. Seega on ka T' minimaalse kaaluga aluspuu.

Tõestatud väite üldistus: olgu $V_1, V_2, \ldots, V_k$ tipuhulga V mingi tükeldus (s.t. $V_1 \cup \cdots \cup V_k = V$ ja $V_i \cap V_j = \emptyset$). Siis saame minimaalse kaaluga aluspuud T muuta nii, et minimaalse kaaluga serv, mis on mingi kahe tüki vahel, sinna puusse kuuluks.



Kui see minimaalse kaaluga serv e on tükkide V_i ja V_j vahel, siis vaatame tipuhulga tükeldust kaheks osaks — V_i ja $V \setminus V_i$. Peale seda rakendame tõestatud tulemust.

Seega on meil võimalik graafi minimaalse kaaluga aluspuud leida järgmise „üldise“ ahne algoritmiga:

Olgu J nende servade hulk, mis me oleme juba otsitavasse puusse võtnud (töö alguses $J = \emptyset$). Kui kõik servad ei ole veel kokku ühendatud, siis täiendava serva lisamiseks:

- Defineeri mingi graafi tipuhulga V tükeldus nii, et ükski J-i kuuluv serv ei oleks kahe tüki vahel.
- Lisa J-i minimaalse kaaluga serv kahe tüki vahel.

Erinevad viisid tükelduste defineerimiseks annavad erinevad algoritmid.

Kruskali algoritm: tükelduse tükkideks on graafi (V, J) sidususkomponendid, s.t. tükeldus on peenim võimalik.

Primi algoritm: peetakse puusse juba kuuluvate tippude hulka U. Algseis: üks suvaline tipp. Tükeldus on $(U, V \backslash U)$.

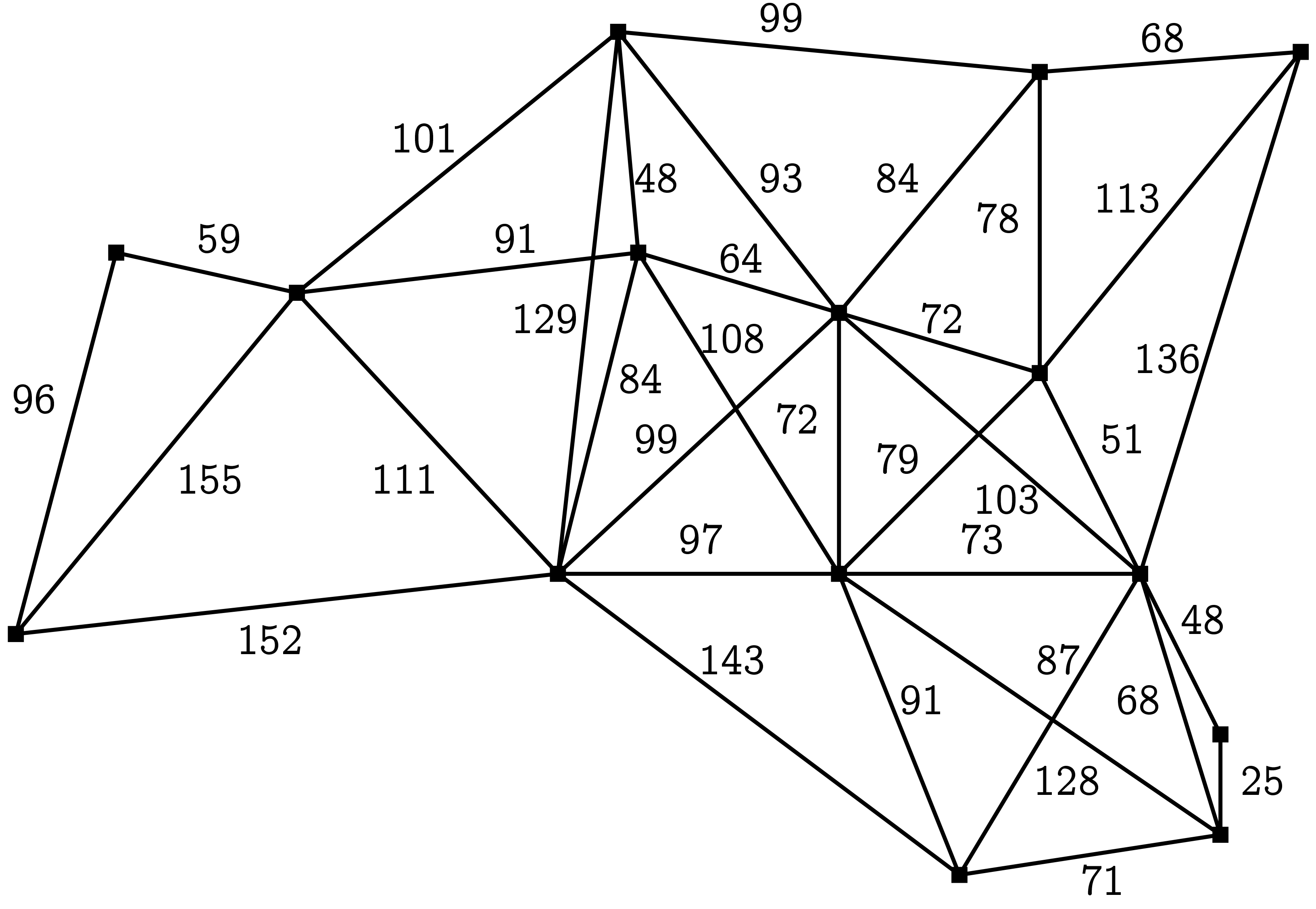
Kruskali algoritm (Kiho konspekt, joonised 6.15 ja 6.16):

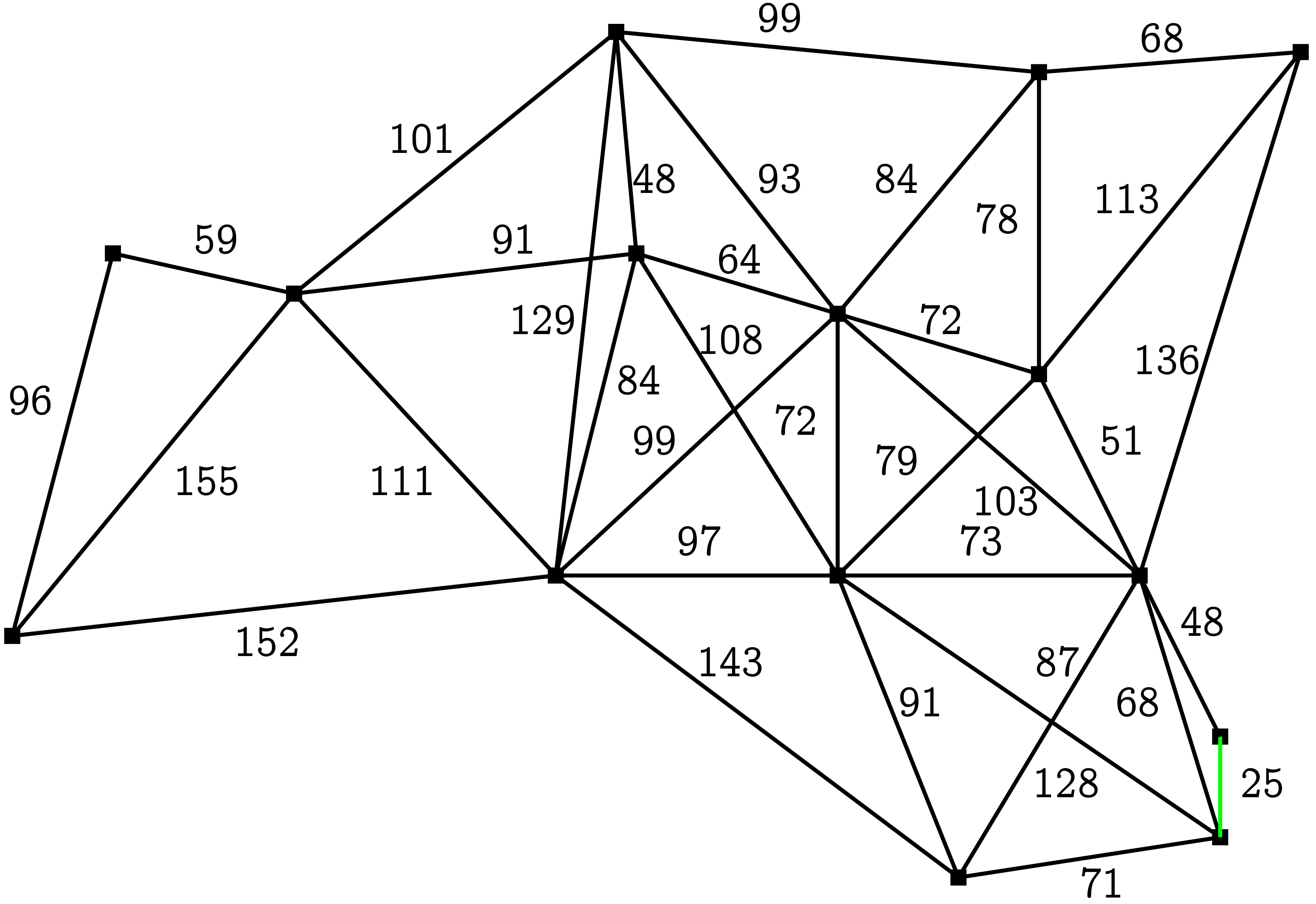
1. Sorteeri graafi kõik servad kaalu järgi mittekahanevalt. Olgu sorteerimise tulemuseks massiiv $e_1, \ldots, e_m$. Olgu $J = \emptyset$.

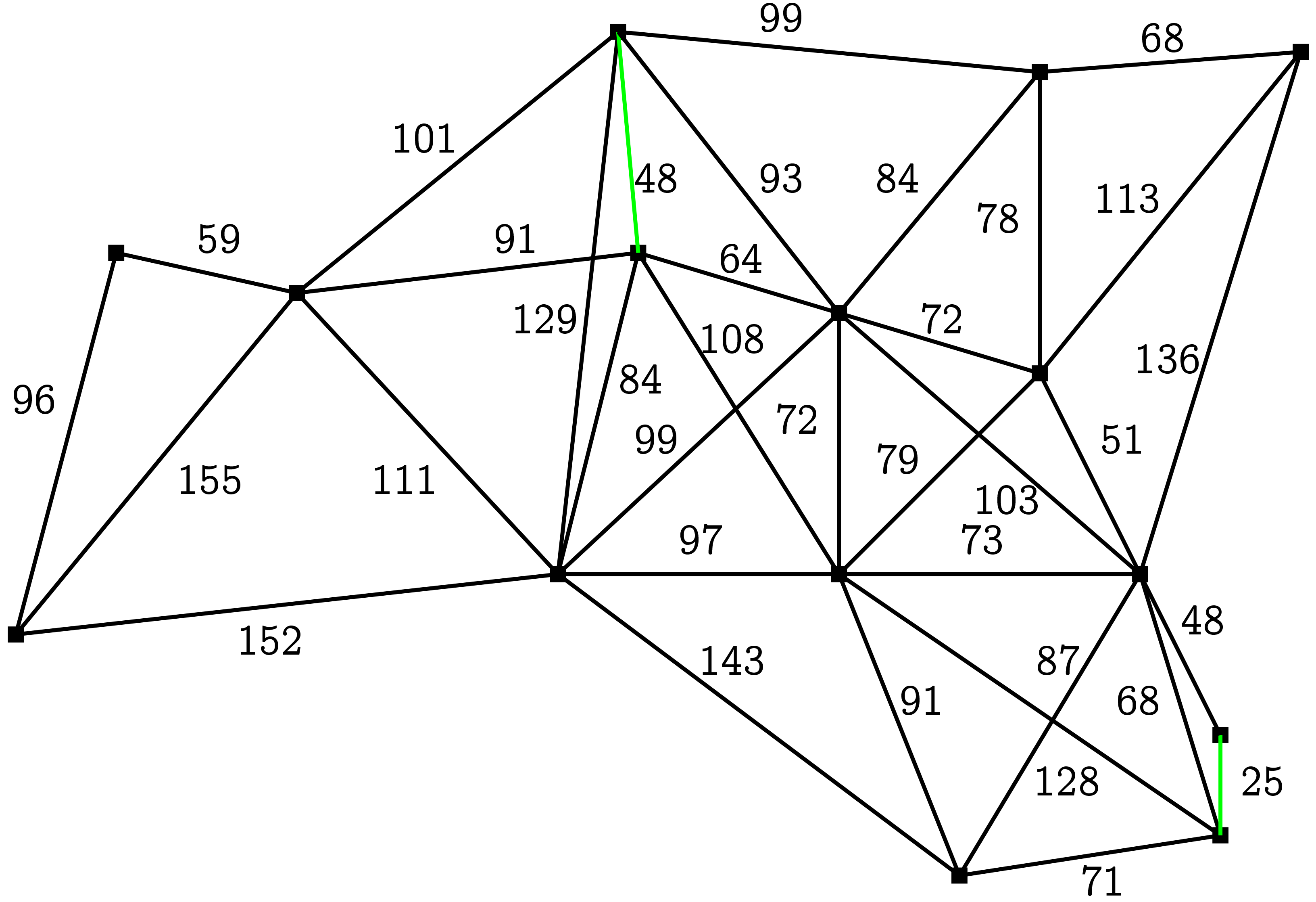
2. **for** $i = 1$ **to** m **do**

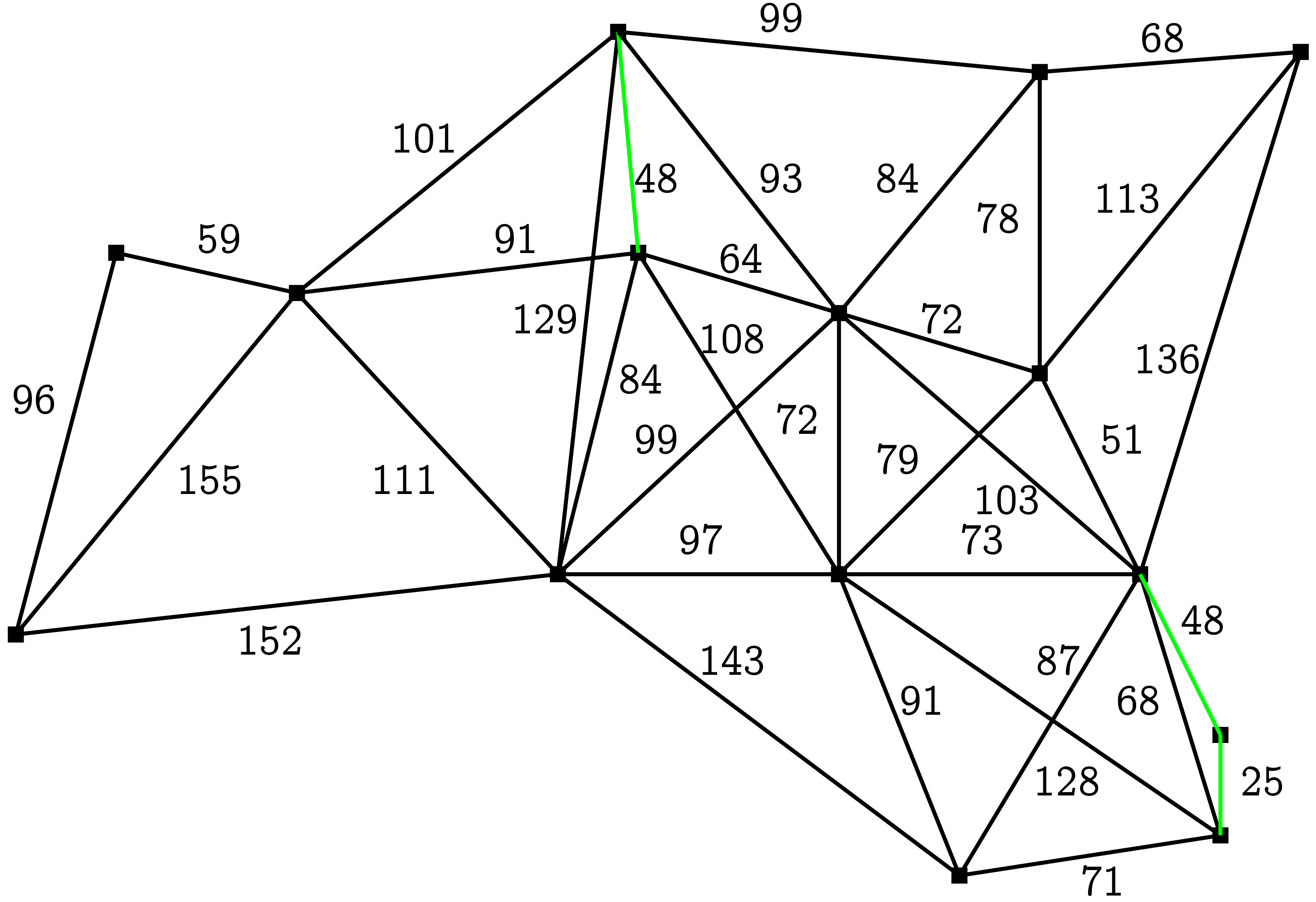
   - Kui e_i on graafi (V, J) kahe sidususkomponendi vahel, siis lisa e_i hulka J.

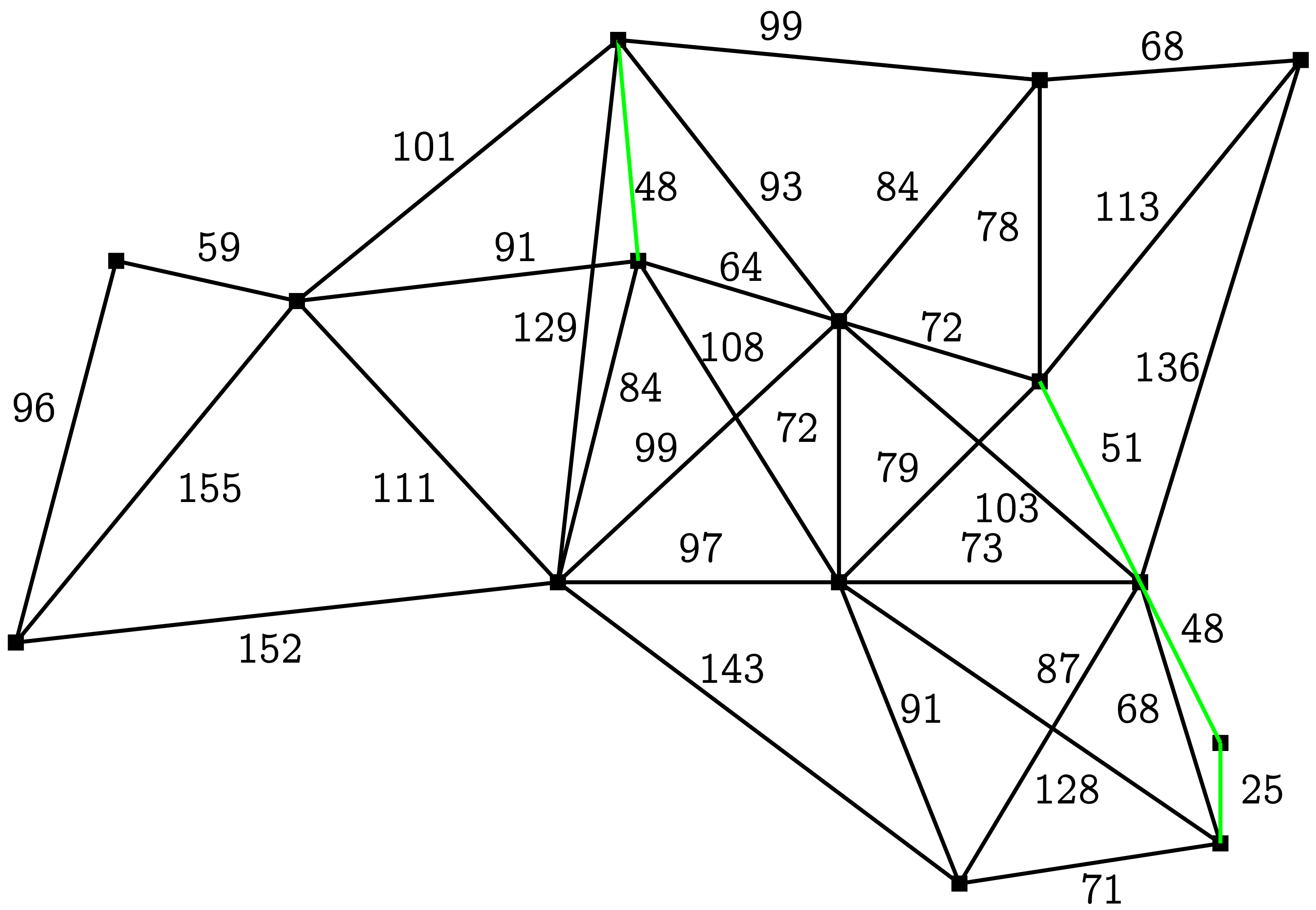
3. Tagasta J.

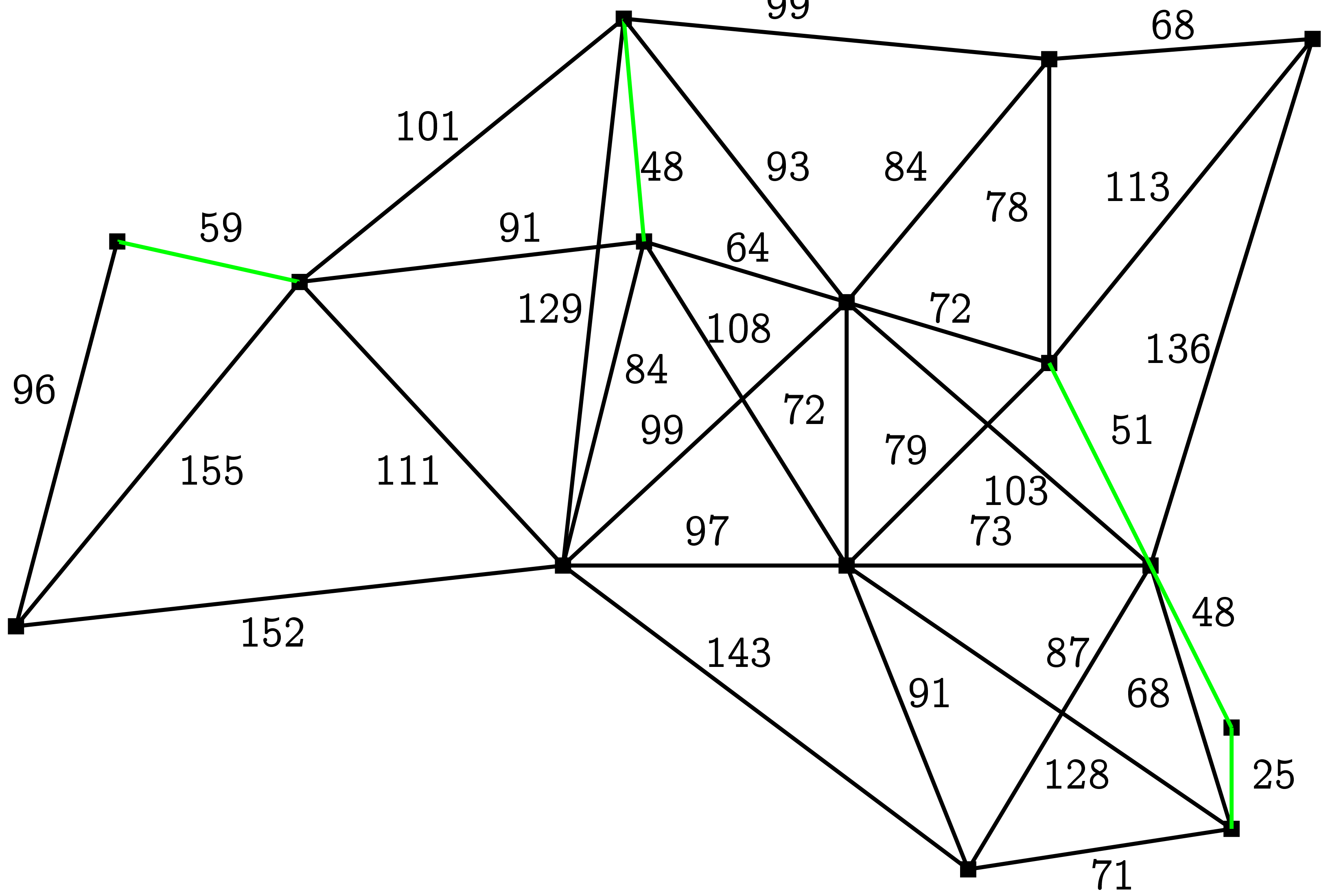

99
68
101
48
93
84
78
113
59
91
64
72
96
129
108
136
84
99
72
79
51
155
111
103
97
73
48
152
143
87
91
68
128
25
71

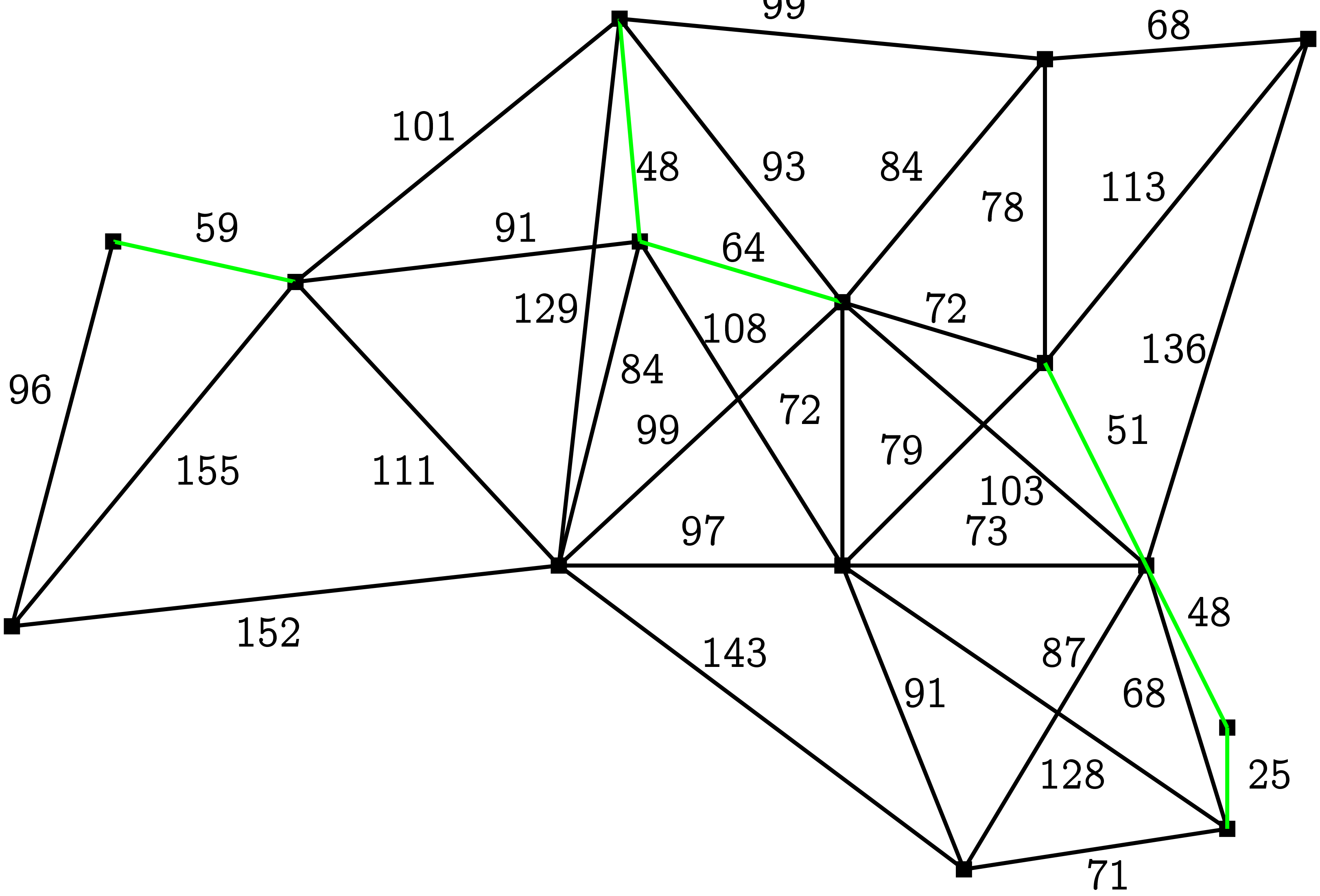

99
68
101
48
93
84
78
113
59
91
64
129
72
108
136
96
84
72
99
51
79
155
111
103
97
73
48
152
143
87
91
68
128
25
71

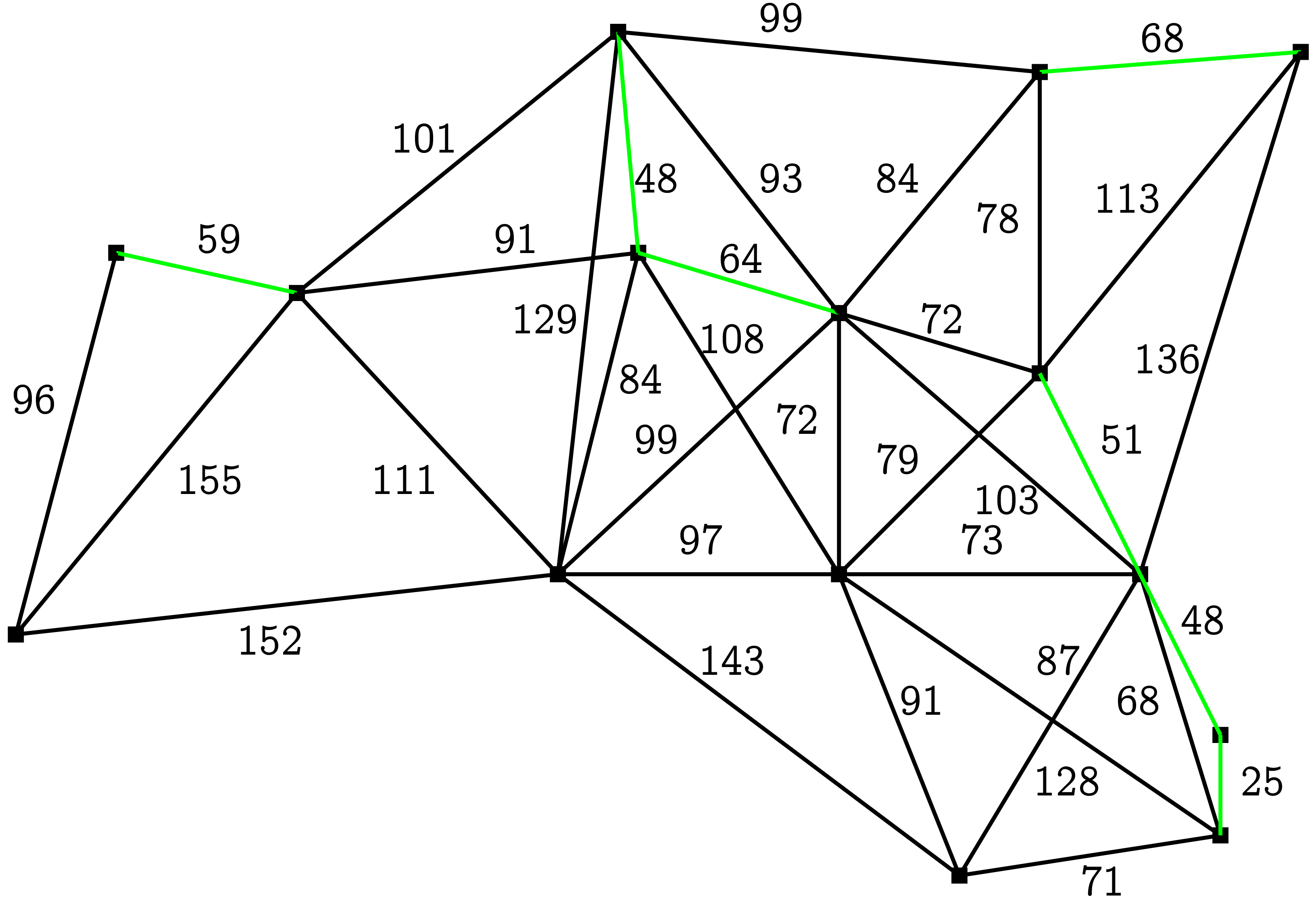

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
48
152
143
87
91
68
25
128
71

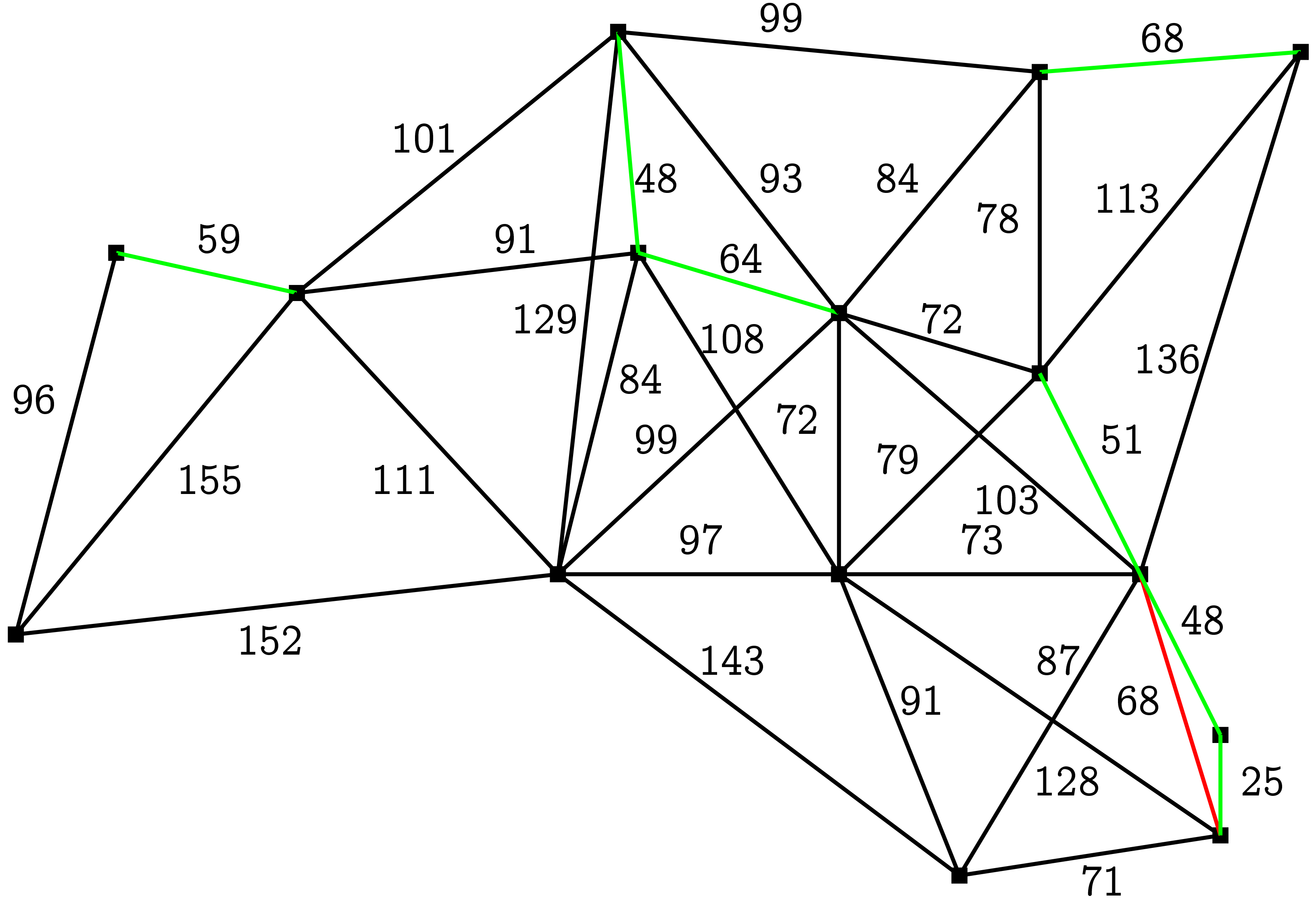

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
152
143
48
87
91
68
128
25
71


99
68
101
48
93
84
78
113
59
91
64
129
72
108
96
84
136
72
99
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71

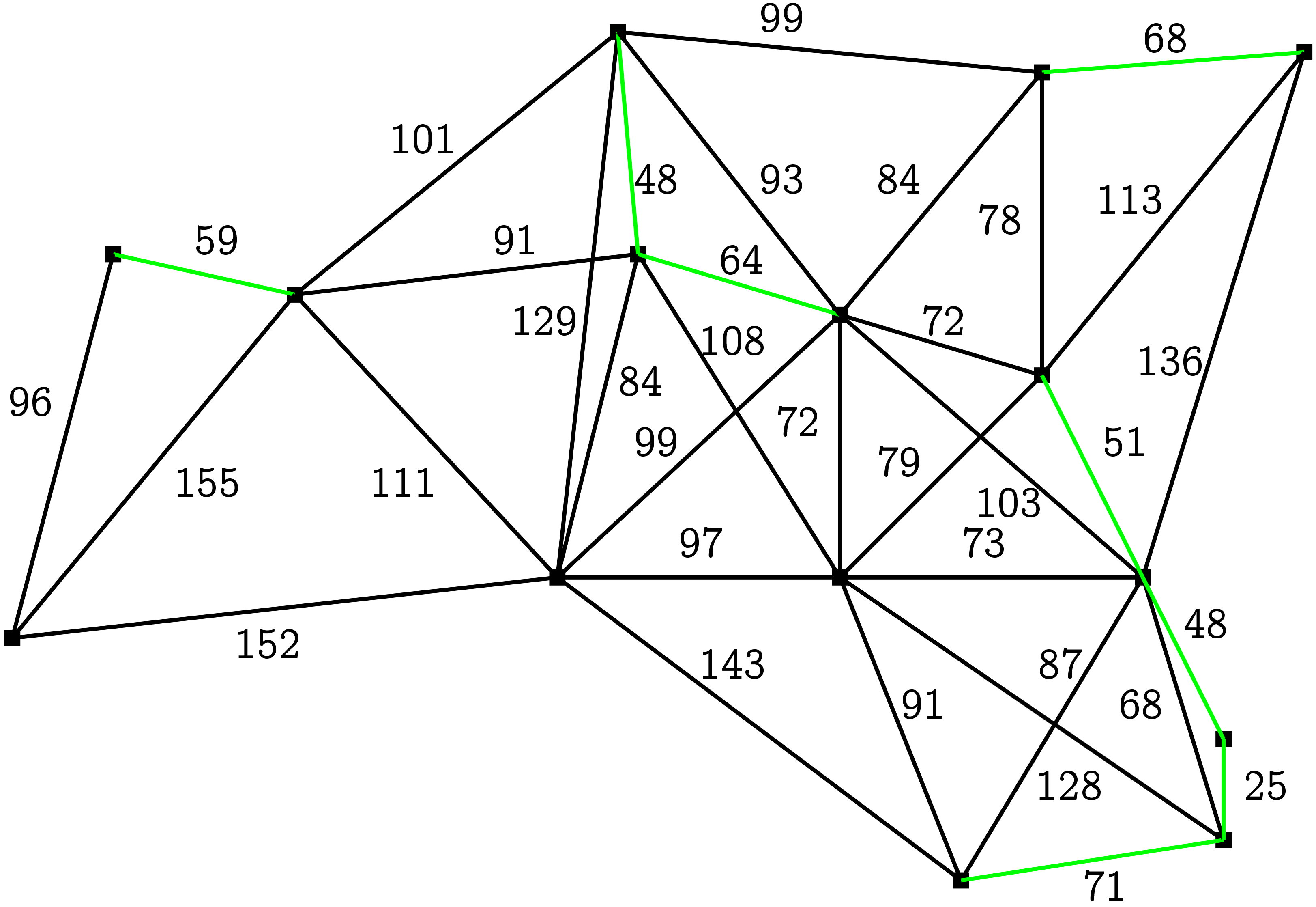

99
68
101
48
93
84
78
113
59
91
64
129
72
108
136
96
84
72
99
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71

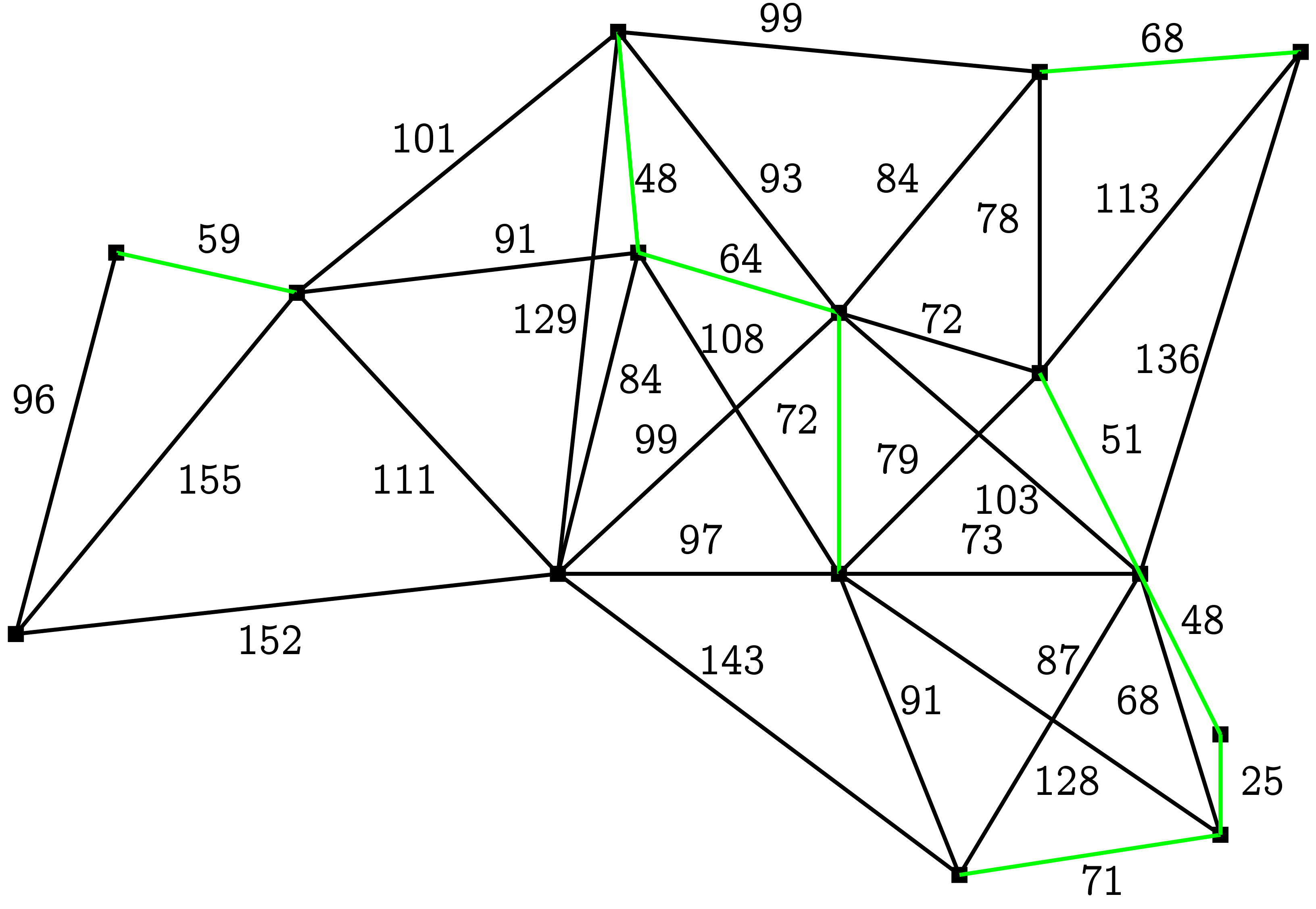

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
48
152
143
87
91
68
128
25
71

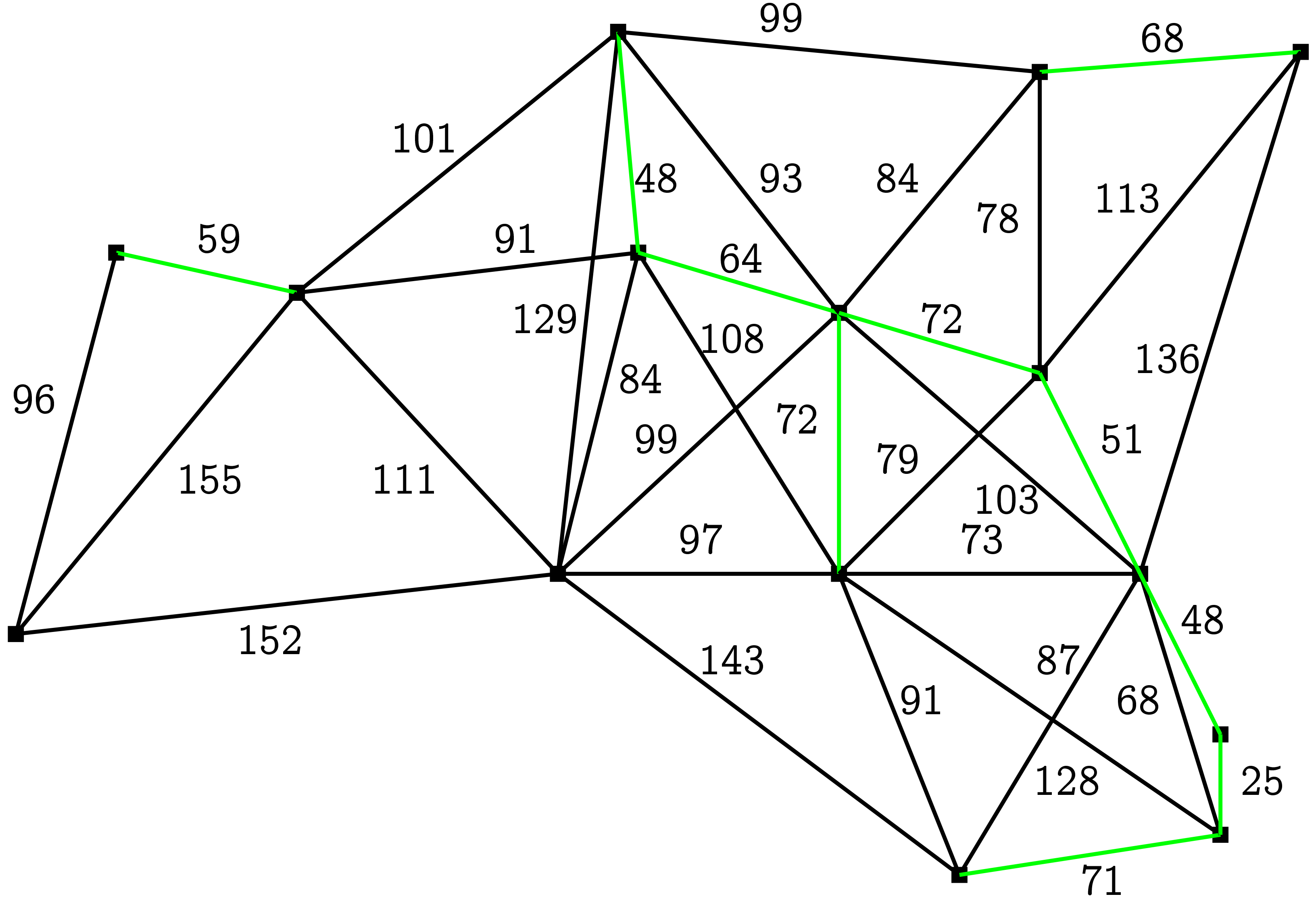

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71

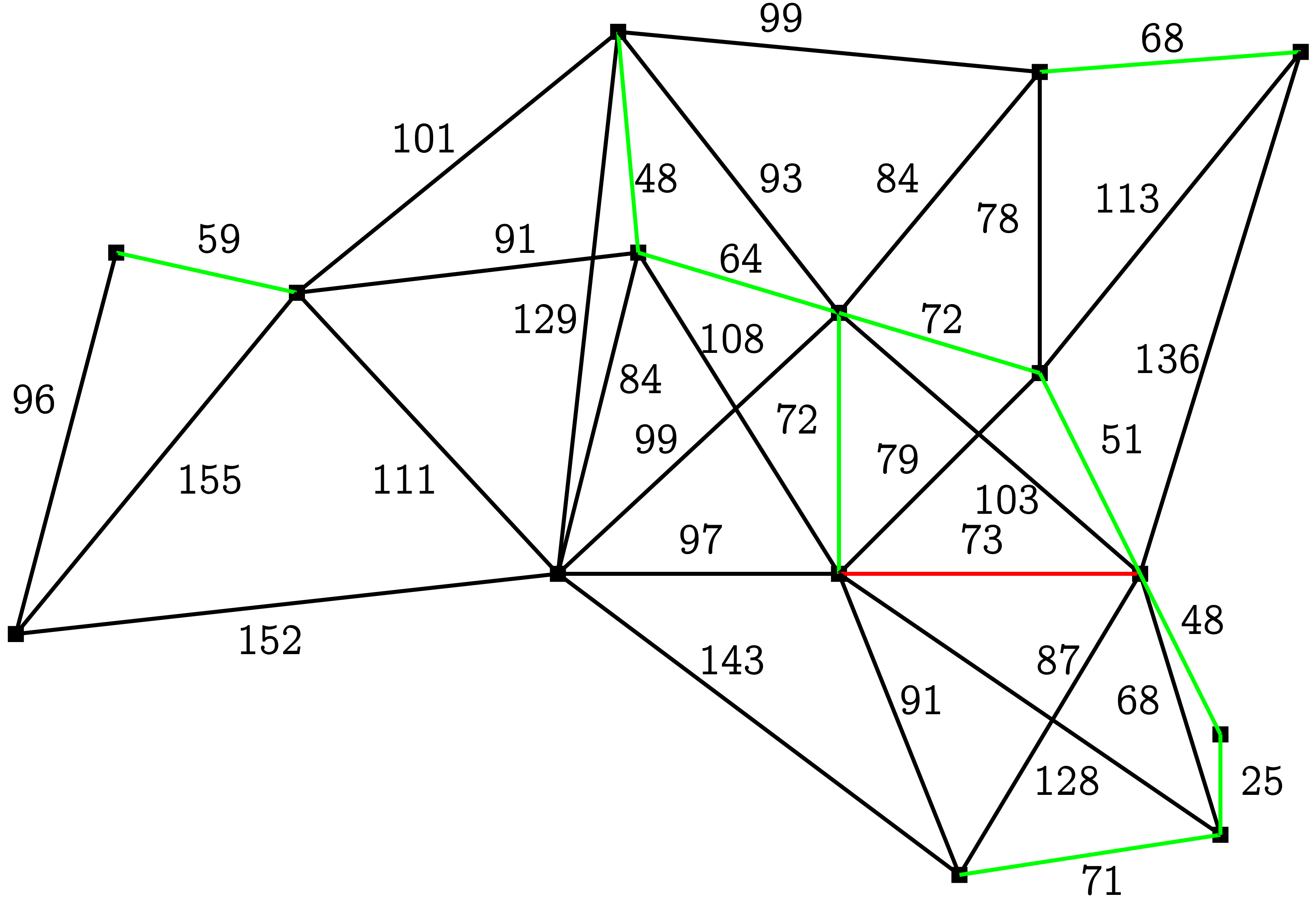


99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
48
152
143
87
91
68
128
25
71

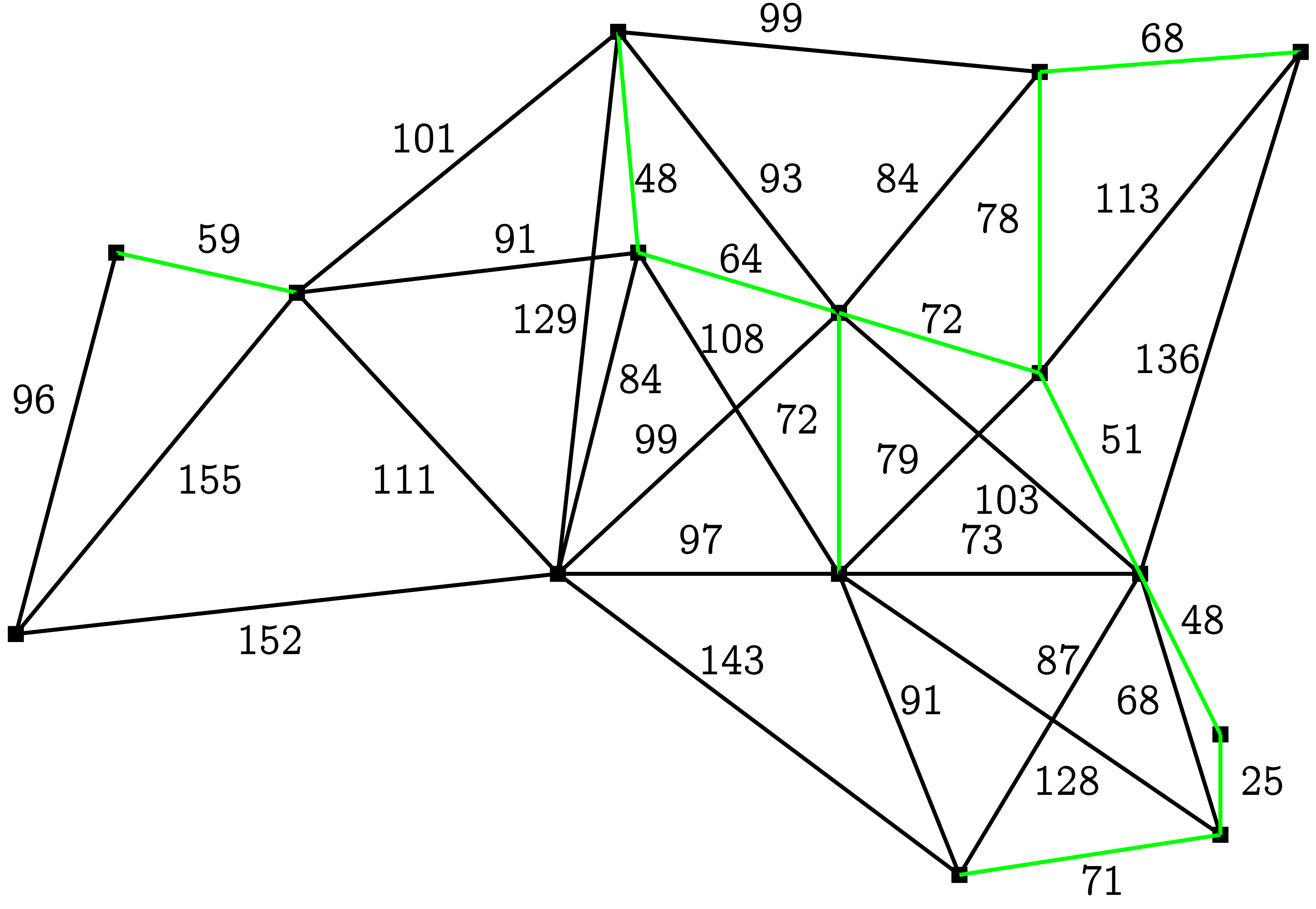

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
51
99
79
155
111
103
97
73
152
143
48
87
91
68
128
25
71

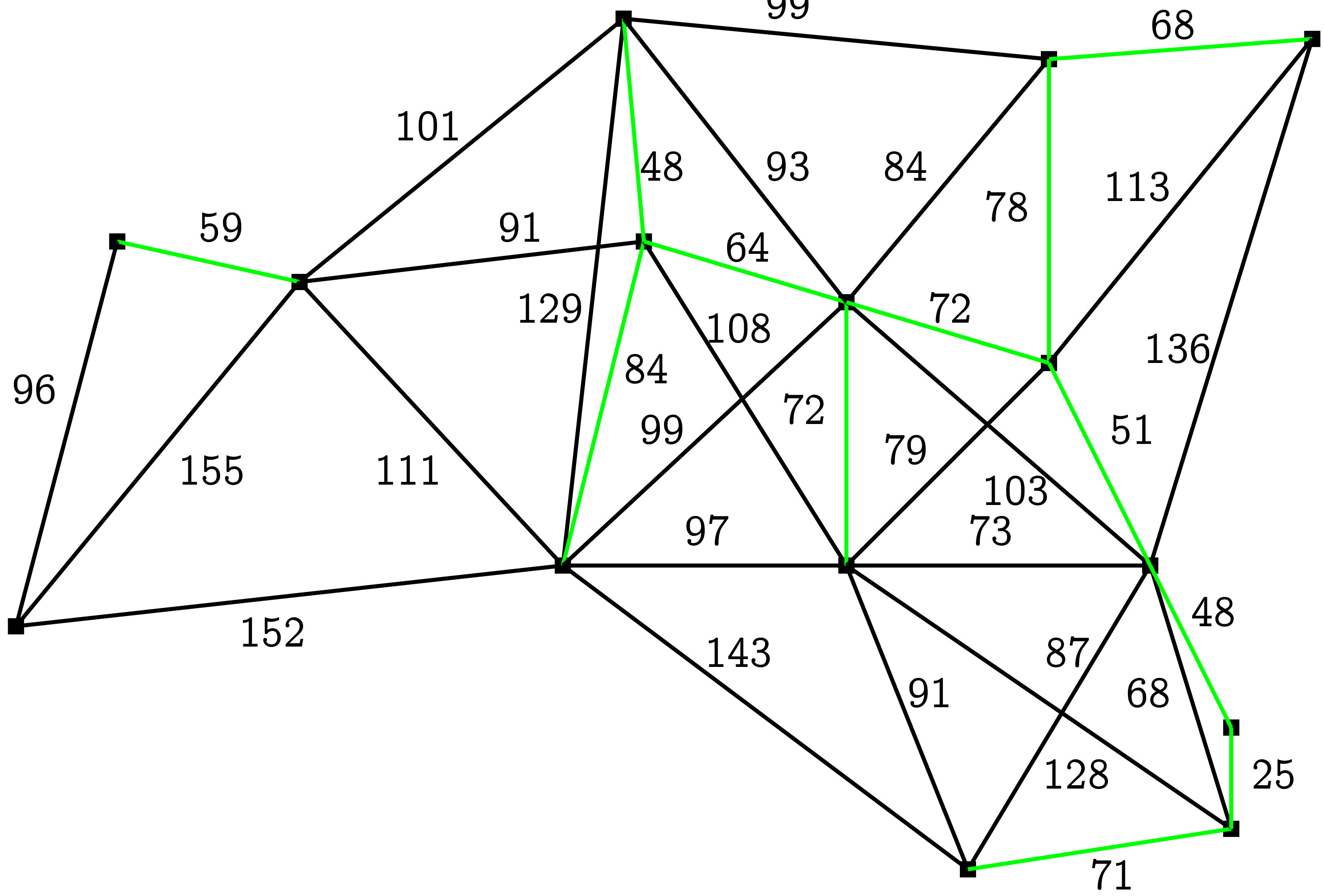

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71

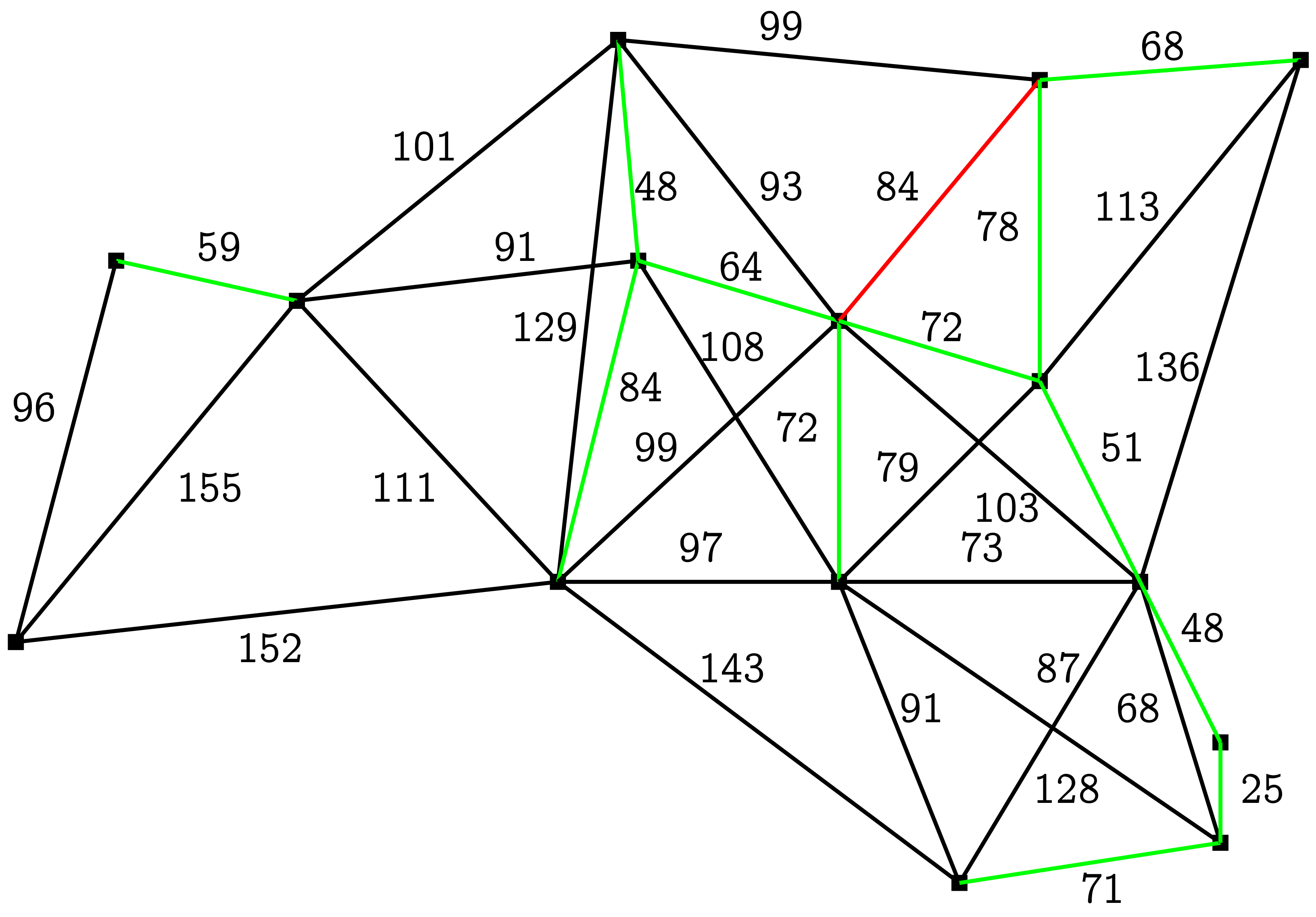

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71

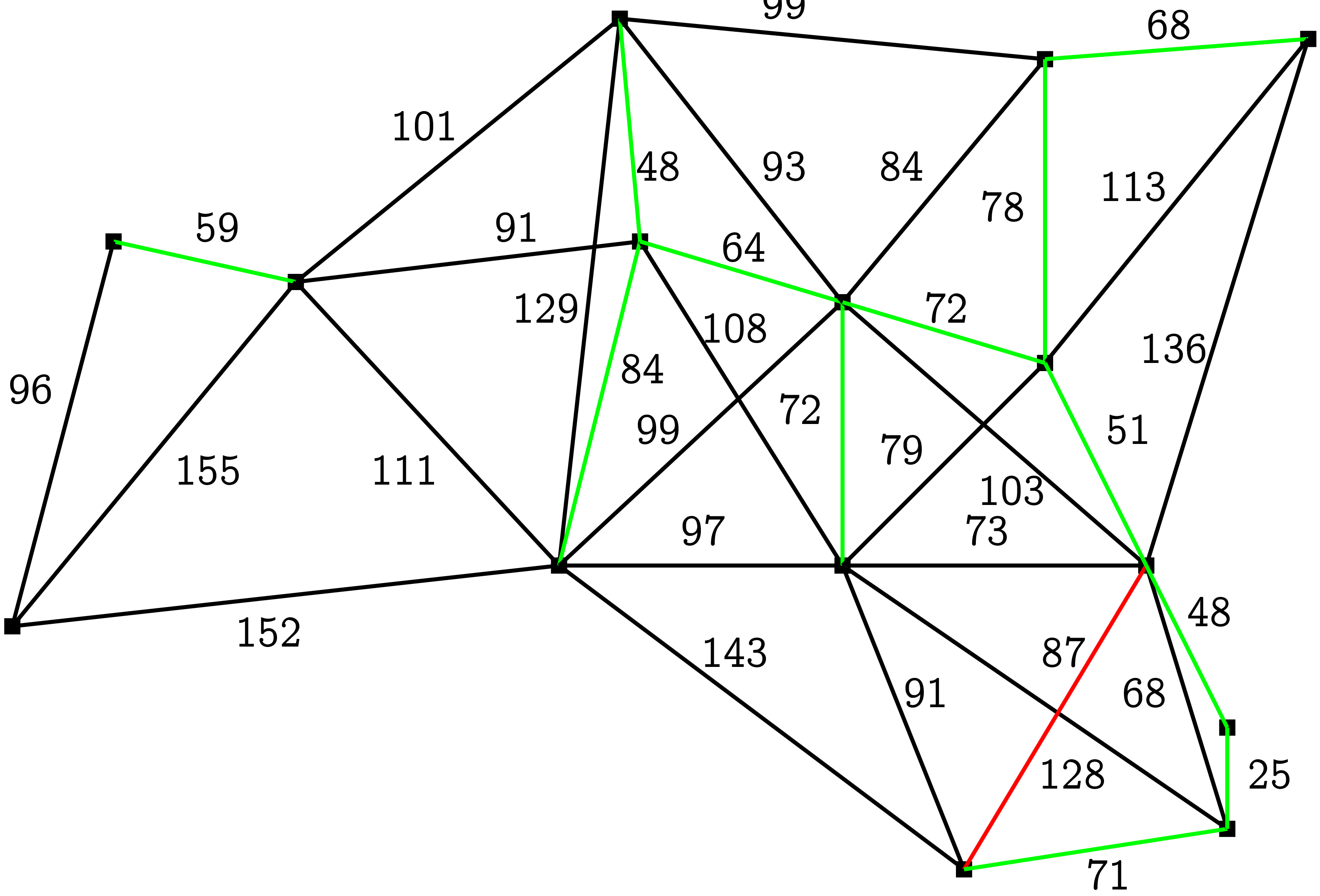


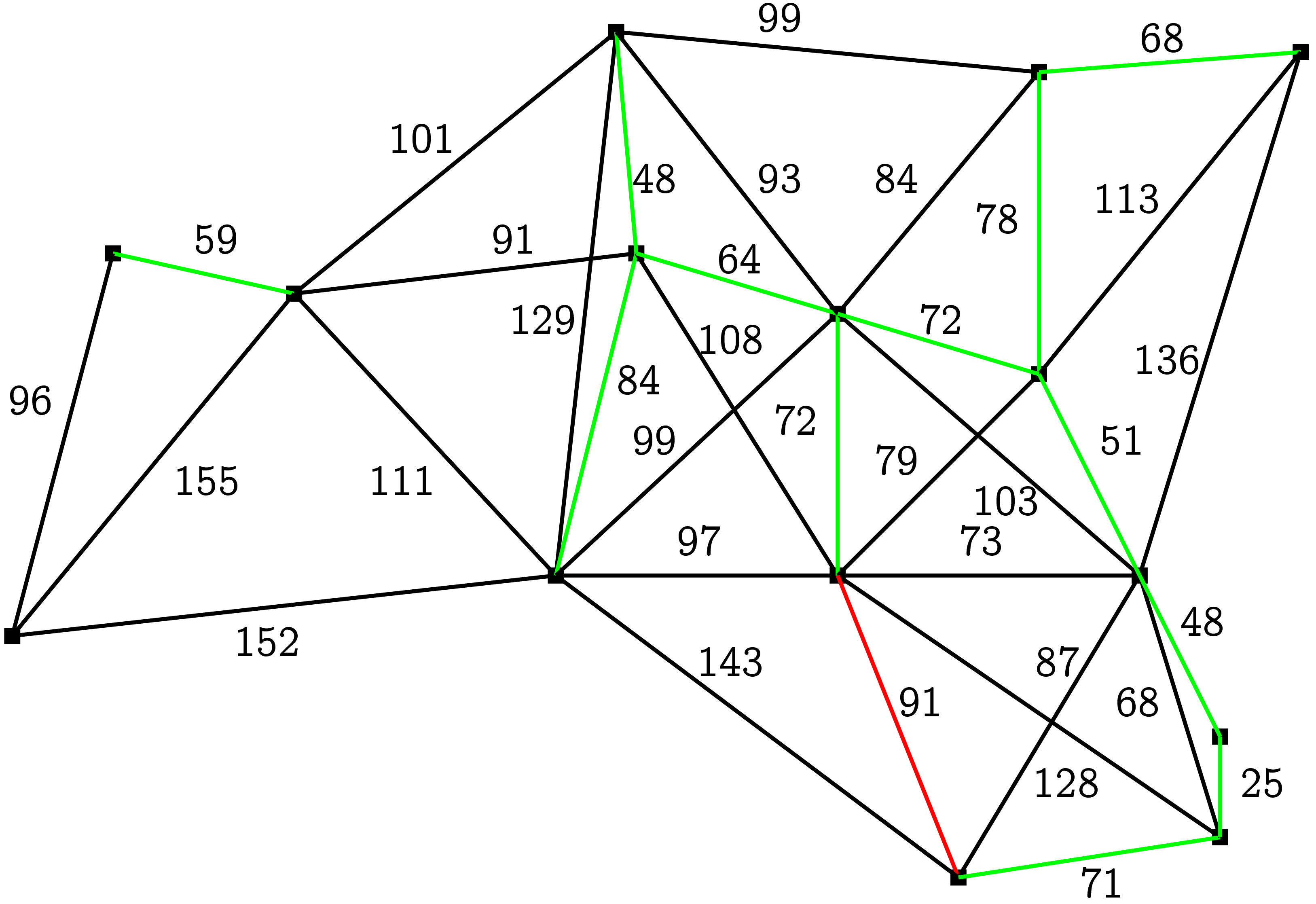
99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
99
72
79
51
155
111
103
97
73
48
152
143
87
91
68
128
25
71

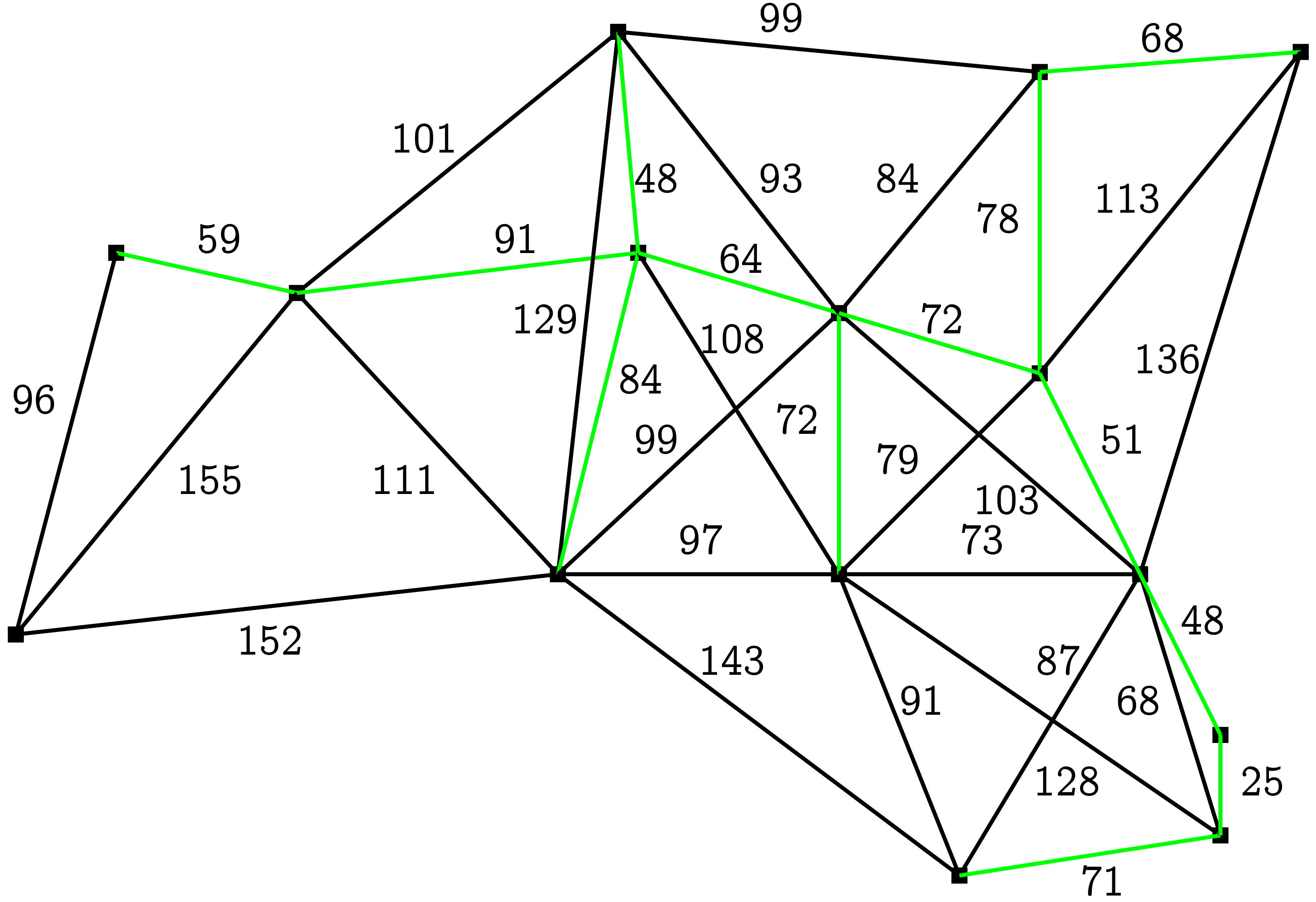

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
48
152
143
87
91
68
128
25
71


99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
48
152
143
87
91
68
128
25
71

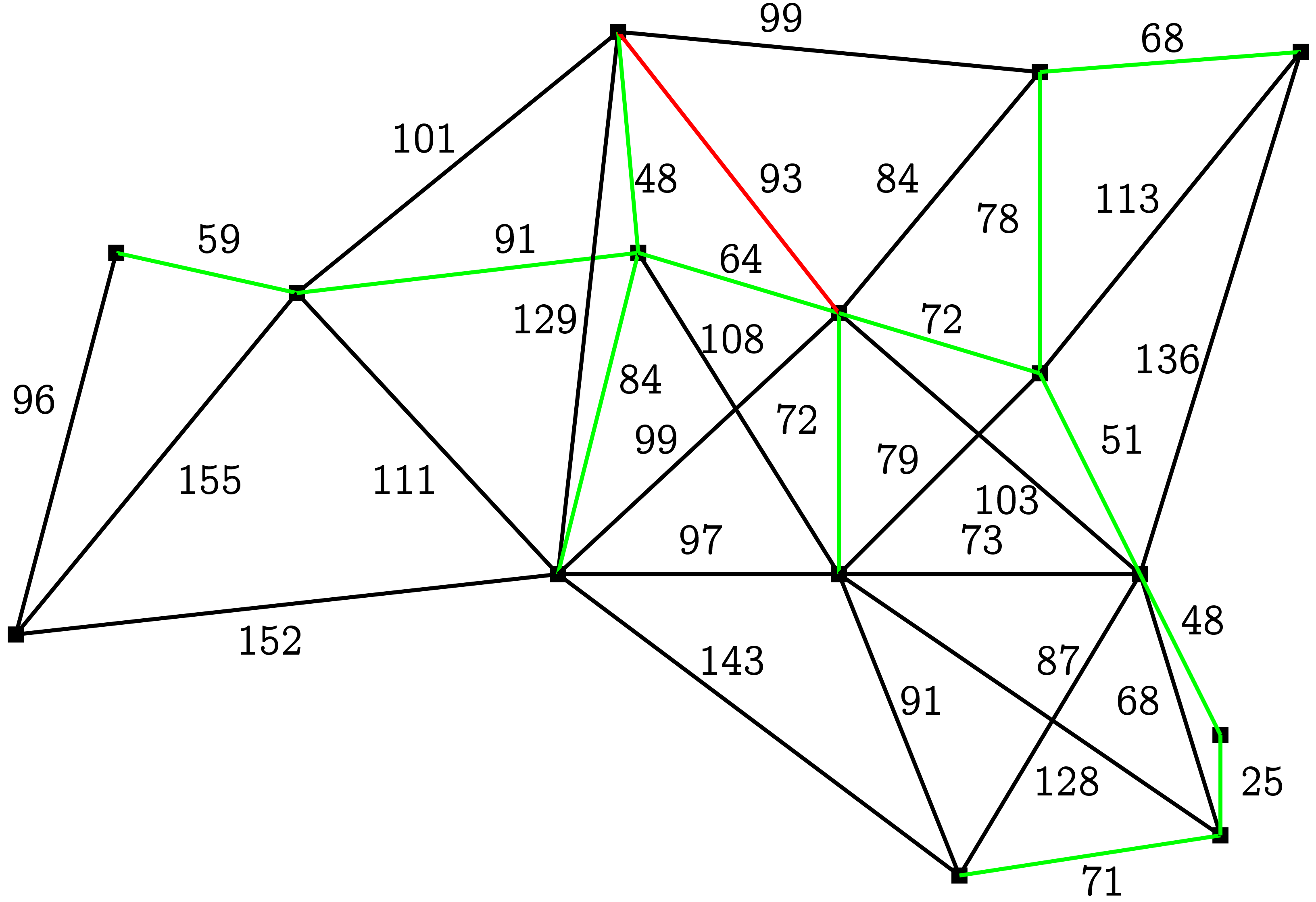

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71

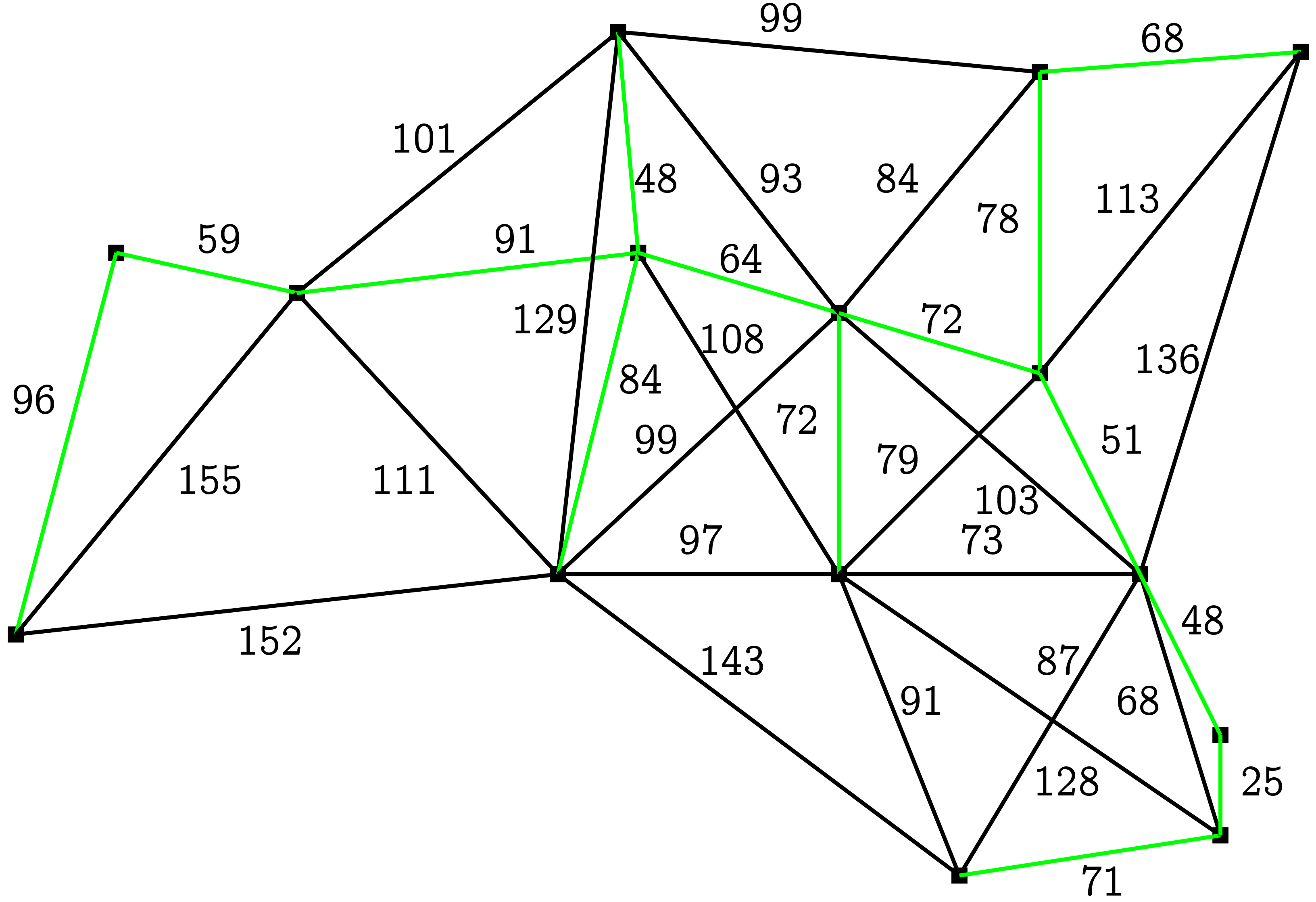

99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
51
99
79
155
111
103
97
73
48
152
143
87
91
68
128
25
71


99
68
101
48
93
84
78
113
59
91
64
129
72
108
136
84
96
99
72
51
79
155
111
103
97
73
48
152
143
87
91
68
128
25
71



99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71


99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
99
72
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71


99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
48
152
143
87
91
68
128
25
71

Algoritm peab arvet pidama, millised tipud on samas ja millised erinevates siduskomponentides.

Sidususkomponendid kujutavad endast V mingi tükelduse tükke.

Alguses on iga tipp eraldi tükis, algoritmi töö jooksul ühendatakse tükke.

Andmestruktuurid klasside kujutamiseks (vt. Kiho konspekti, jaotis 3.3) lubavad järgmisi operatsioone efektiivselt teha:

- kahe elemendi samasse klassi kuulumise kontroll;
- kahe klassi ühendamine.

Keerukus:

- Sorteerimine — $O(|E|\log|E|)$.
- Servade lisamine — $O(|E|\alpha(|V|))$, kus $\alpha(n)$ on ühtekokku n elemendiga klasside opereerimisel ühe operatsiooni (amortiseeritud) keerukus.

α on väga aeglaselt kasvav funktsioon. Seega on kogukeerukus $O(|E|\log|E|)$ ehk $O(|E|\log|V|)$, sest

$$|V| = O(|E|), |E| = O(|V|^2)$$

ja seega $\log|E| = \Theta(\log|V|)$.

Primi algoritm (Kiho konspekt, joonis 6.17):

Initsialiseerimine:

Hulgaks U võta üks tipp.

Kõik ülejäänud tipud pane kahendkuhja Q (juurtipus on vähima võtmega kirje, s.t. minimaalse elemendi eemaldamine on kiirelt tehtav).

Tipu v võtmeks on vähima kaaluga sellise serva kaal, mille üheks otstipuks on v ja mille teine otstipp on U-s. Kui sellist serva ei leidu, siis on võtmeks ∞. Kui leidub, siis tähistame teda e_v, ta peab tipu v juures salvestatud olema.

Konspektis: tippe võtmeväärtusega ∞ kuhja ei võeta.

Iteratsioon: võta kuhjast Q vähima võtmeväärtusega tipp v.

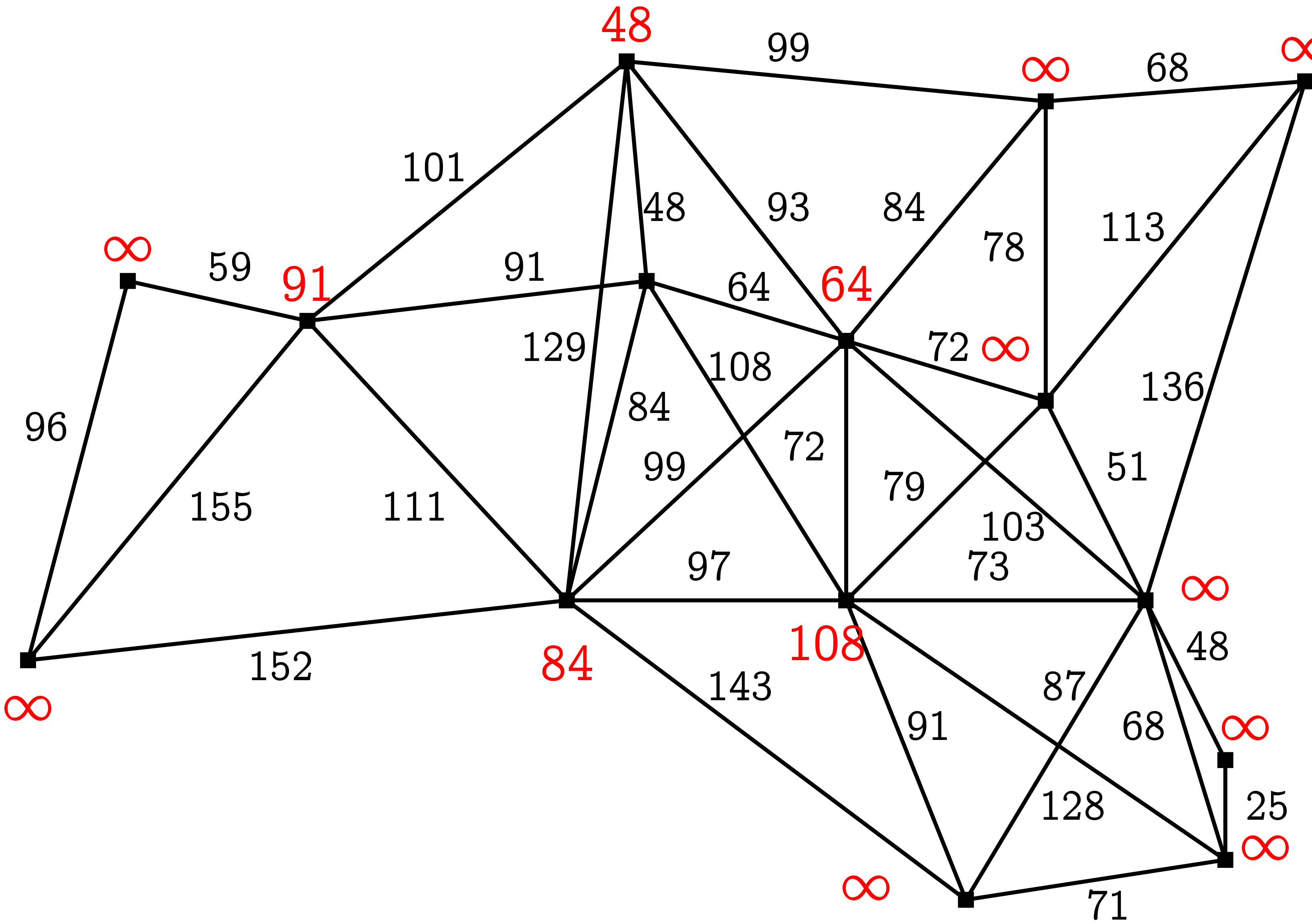
- Lisa v hulka U ning e_v konstrueeritavasse aluspuusse.
- Käi läbi tipuga v intsidentsed servad. Kui mõne serva e korral e teine otstipp x ei kuulu U-sse, siis
  - Kontrolli, kas $w(e)$ on väiksem kui x-i võti. Kui jah, siis
    * võta x-i võtmeväärtuseks $w(e)$ (ja tee kuhjas vii_üles(x));
    * võta e_x-ks e.

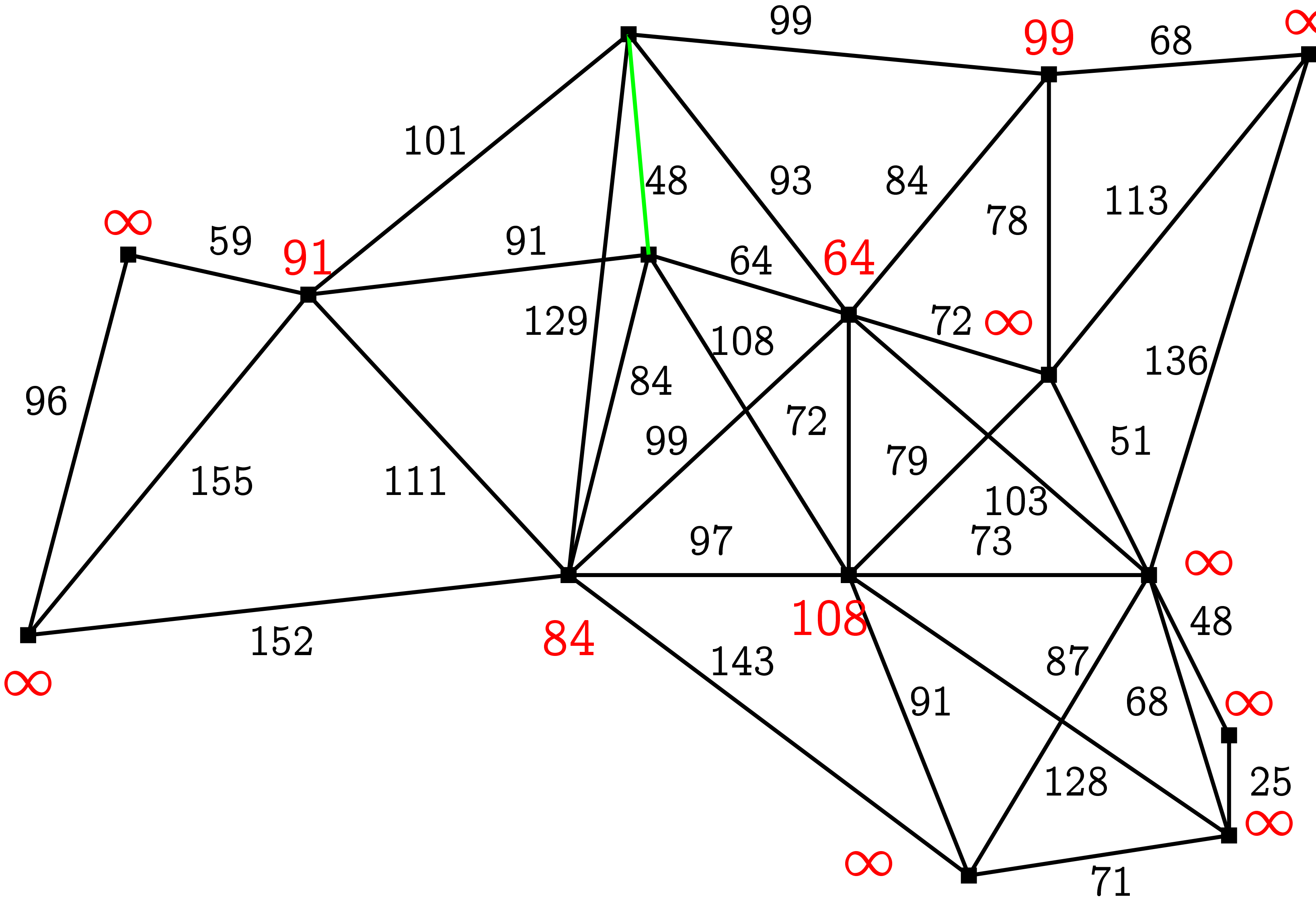
Töö lõppeb, kui Q tühjaks saab.

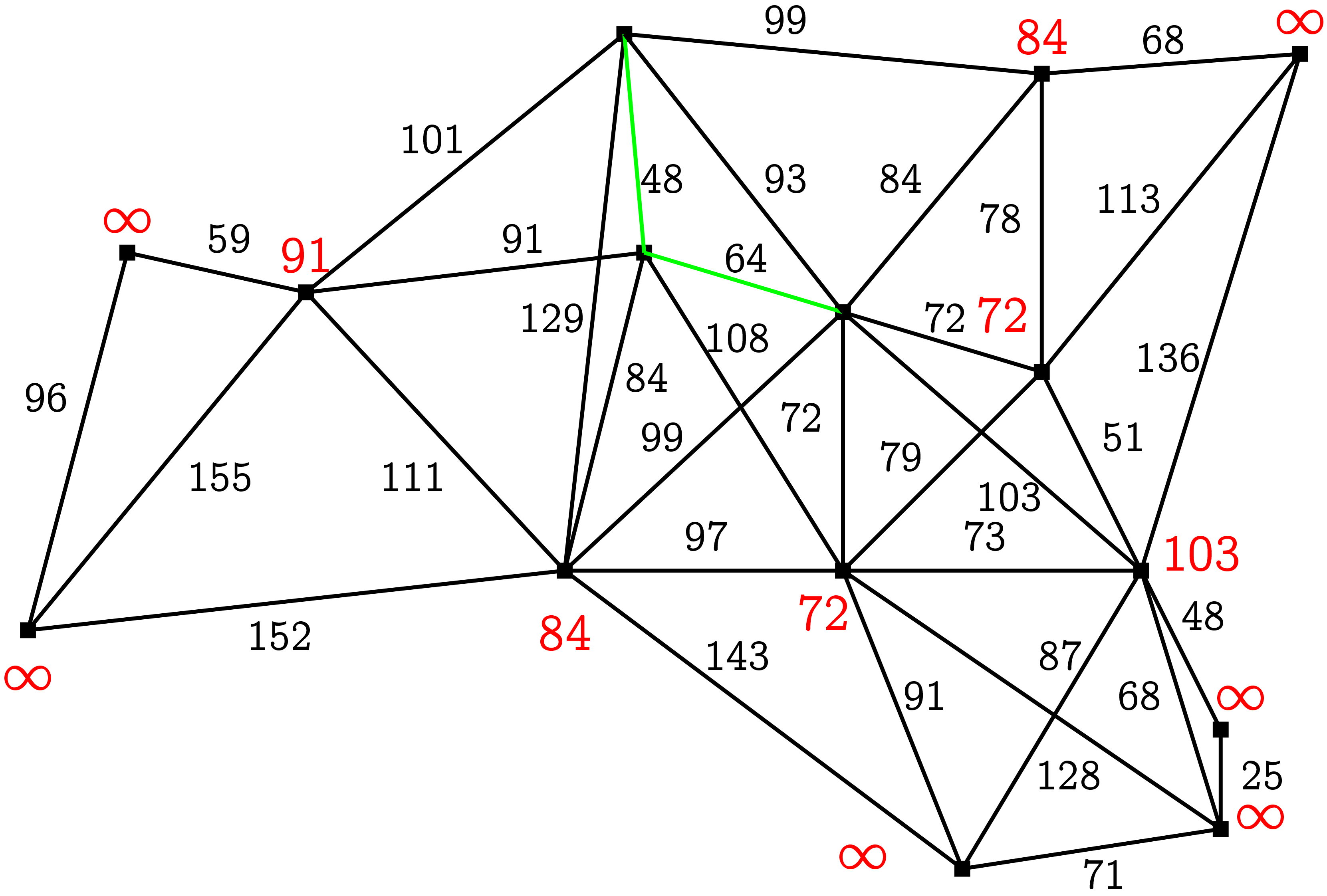
Keerukus:

- Initsialiseerimine: tippude võtmete initsialiseerimine — $O(|V|)$, kuhjastamine — $O(|V|)$.
- Iteratsioonid:
  - Kuhjast vähimate elementide võtmine — $O(|V|\log|V|)$.
  - Servade töötlemine — $O(|E|\log|V|)$.
  - Kui aga kasutada mitte kahendkuhja, vaid *Fibonacci kuhja*, siis vastavalt $O(|V|\log|V|)$ ja $O(|E|)$.

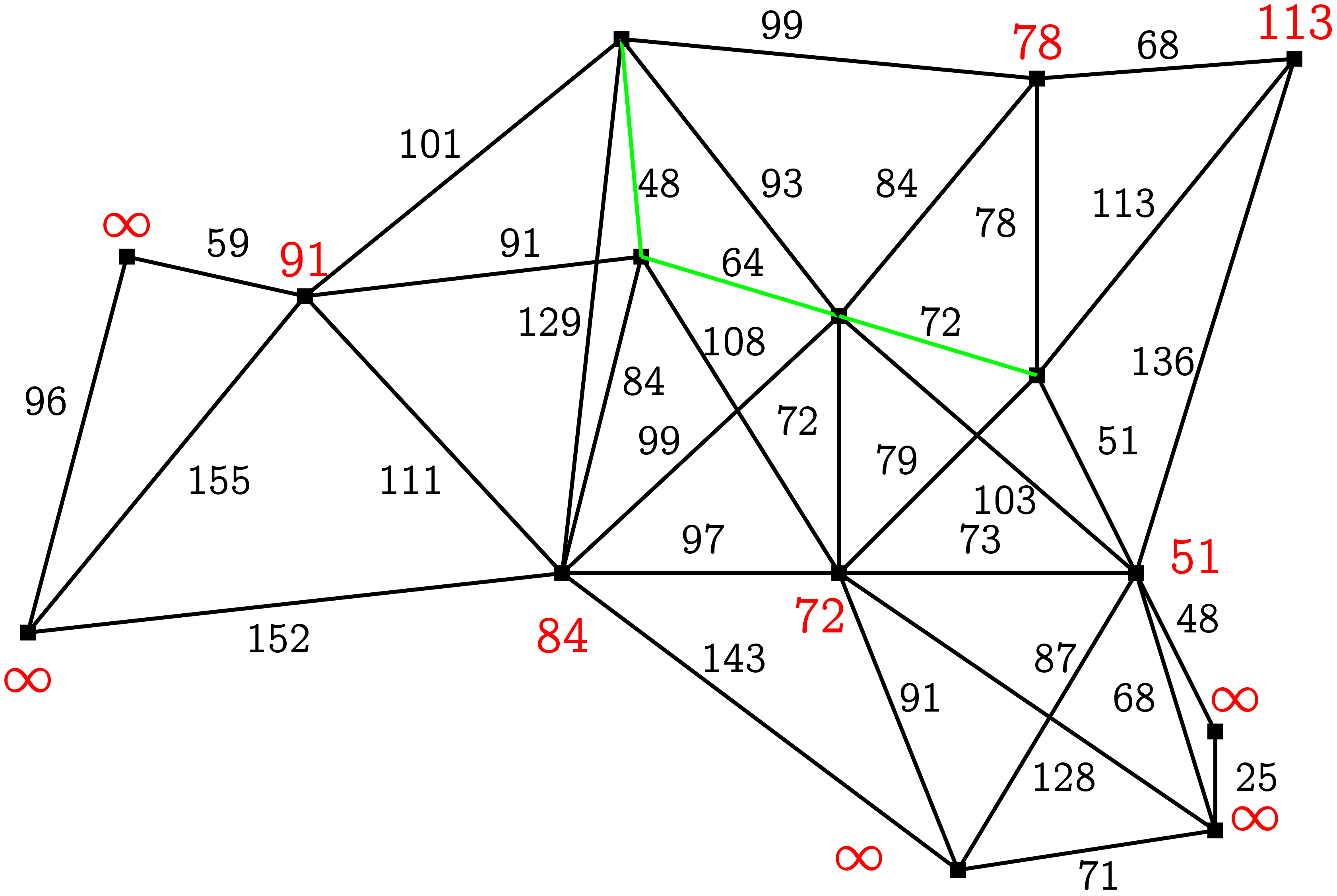
Kokku seega $O(|E|\log|V|)$ (Fibonacci kuhja korral $O(|E|+|V|\log|V|)$).

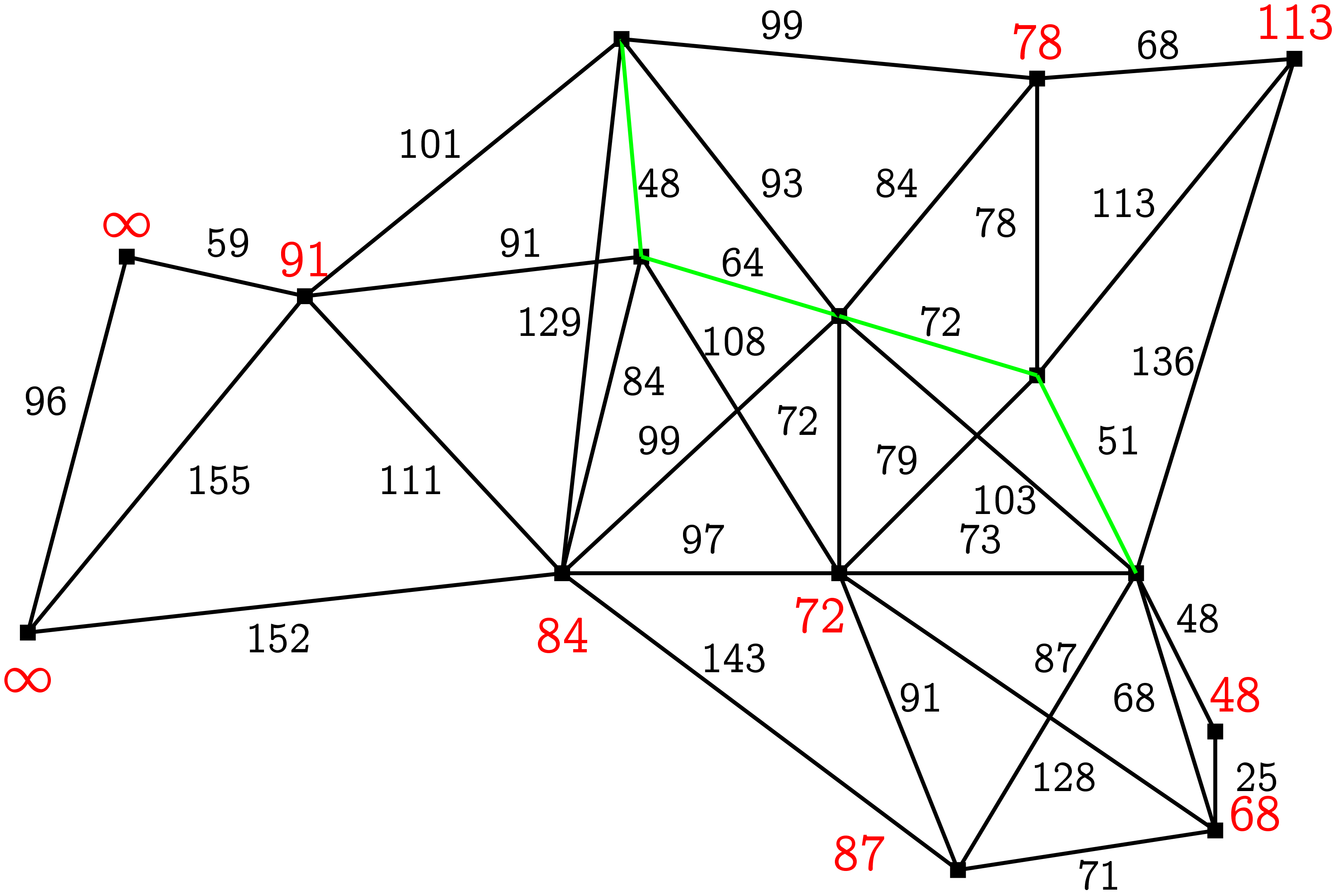

48
99
∞
68
∞
101
48
93
84
78
113
∞
59
91
91
64
64
72
∞
129
108
136
96
84
72
99
79
51
155
111
103
97
73
∞
84
108
48
152
∞
143
87
91
68
∞
128
25
∞
∞
71

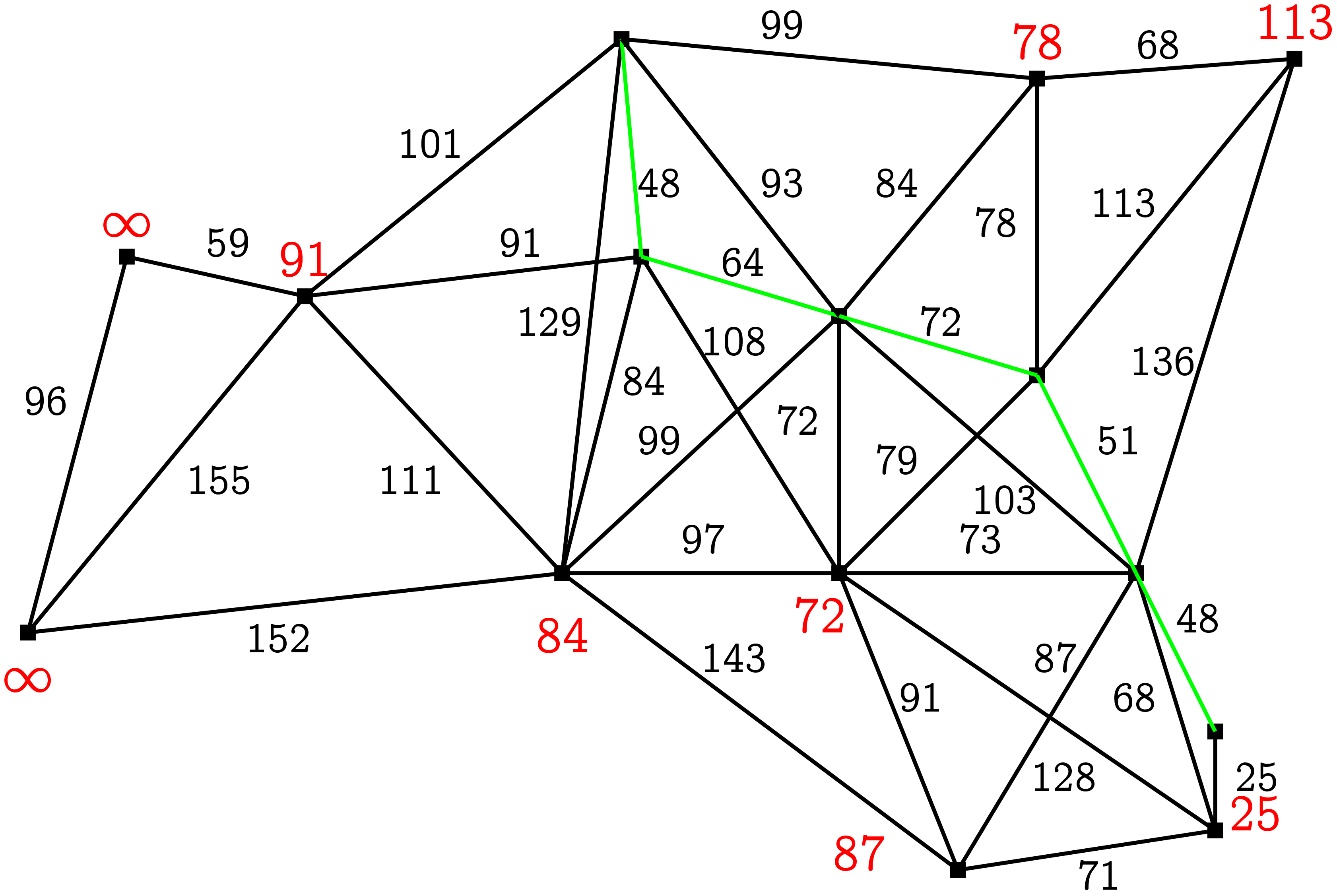

99
68
∞
99
101
48
93
84
78
113
∞
59
91
91
64
64
72
∞
129
108
136
96
84
99
72
79
51
155
111
103
97
73
∞
152
84
108
143
48
87
91
68
∞
∞
128
25
∞
∞
71



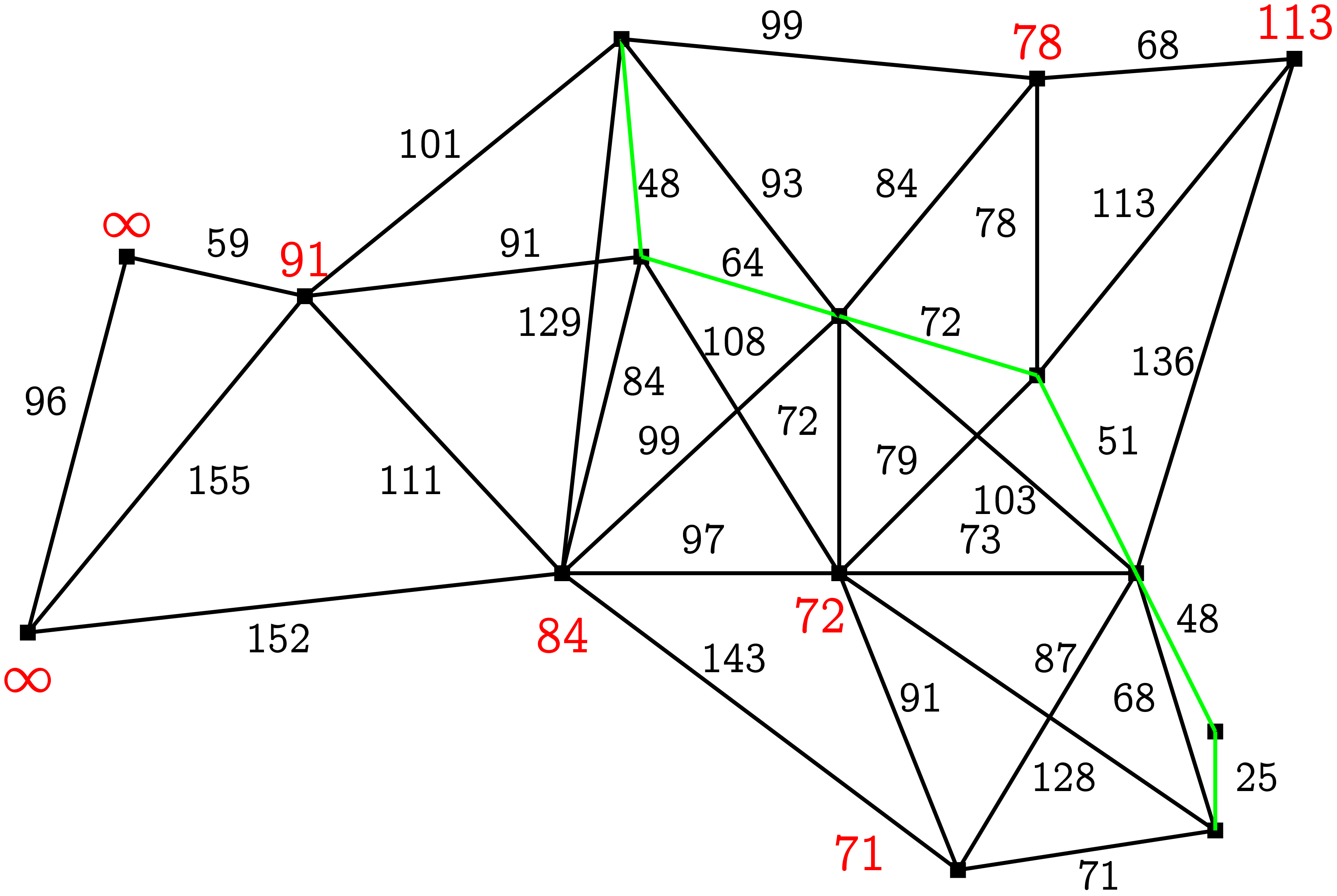
99
84
68
∞
101
48
93
84
78
113
∞
59
91
91
64
72
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
103
152
84
72
143
48
87
∞
91
68
∞
128
25
∞
∞
71

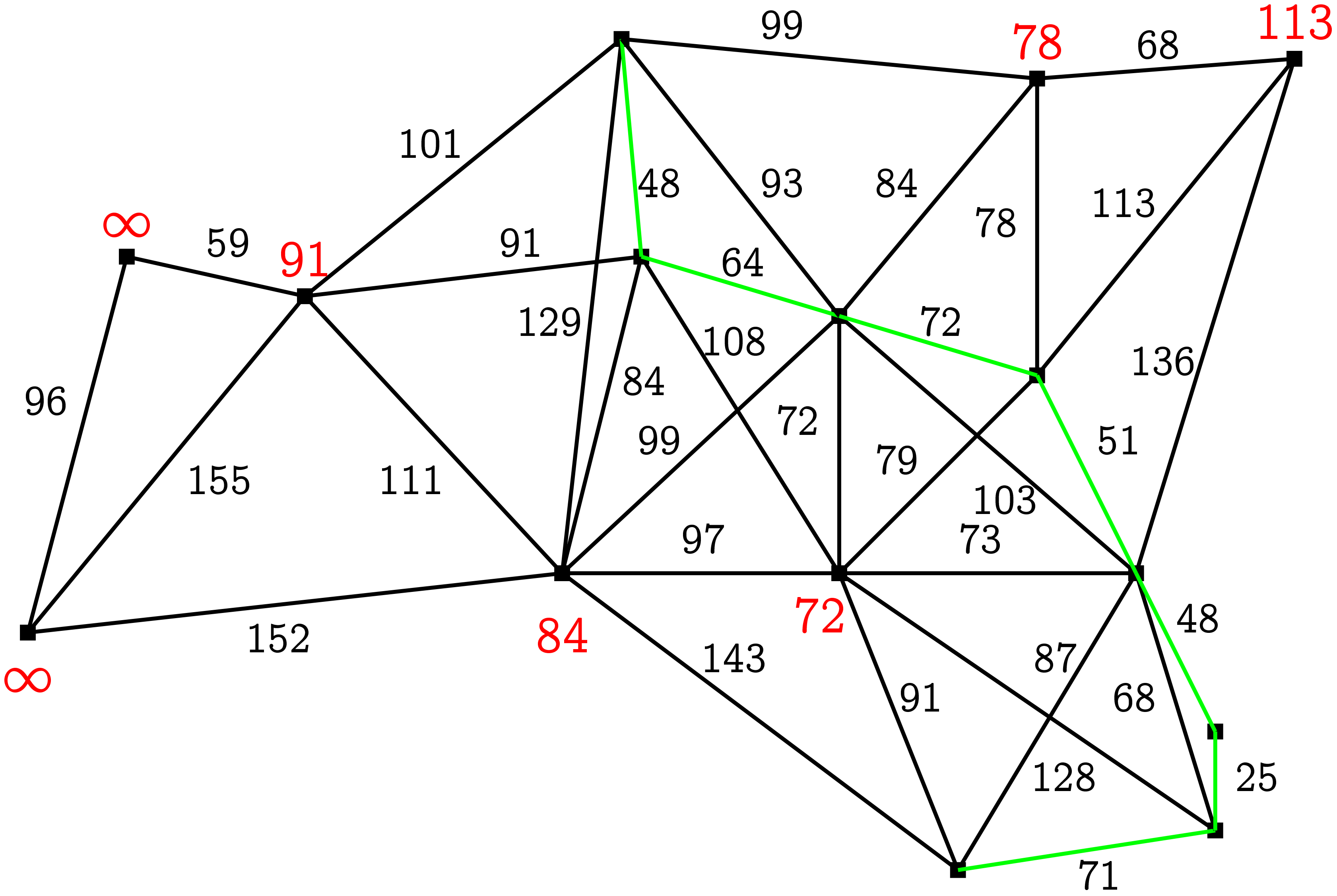

99
78
68
113
101
48
93
84
78
113
∞
59
91
91
64
72
129
108
136
96
84
99
72
79
51
155
111
103
97
73
51
152
84
72
143
48
87
91
68
∞
∞
25
128
∞
∞
71


99
78
68
113
101
48
93
84
78
113
∞
59
91
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
84
72
48
152
143
87
∞
91
68
48
128
25
68
87
71

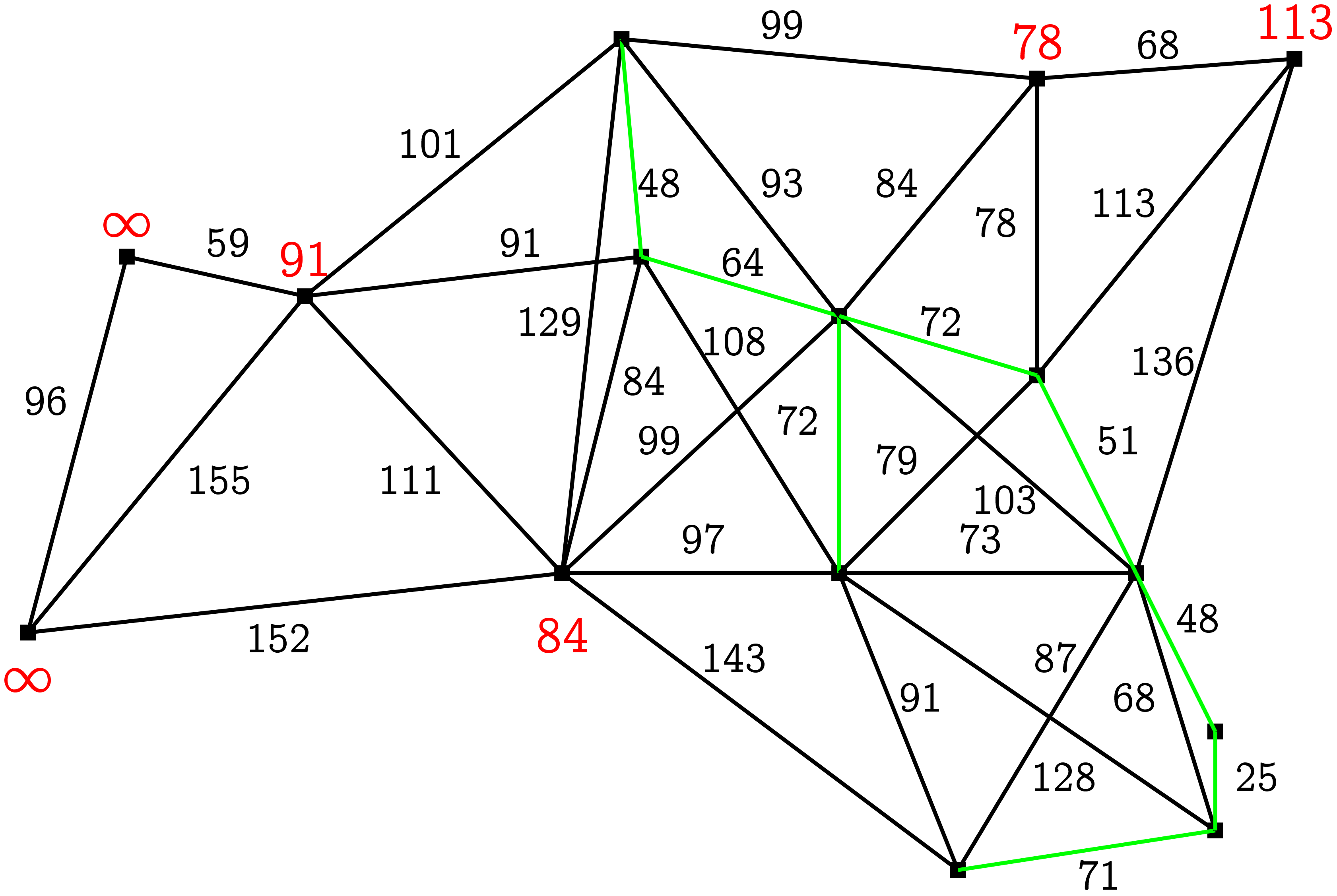


99
78
68
113
101
48
93
84
78
113
∞
59
91
91
64
72
136
96
129
108
84
72
99
79
51
155
111
103
97
73
72
84
152
143
48
87
91
68
∞
128
25
25
87
71

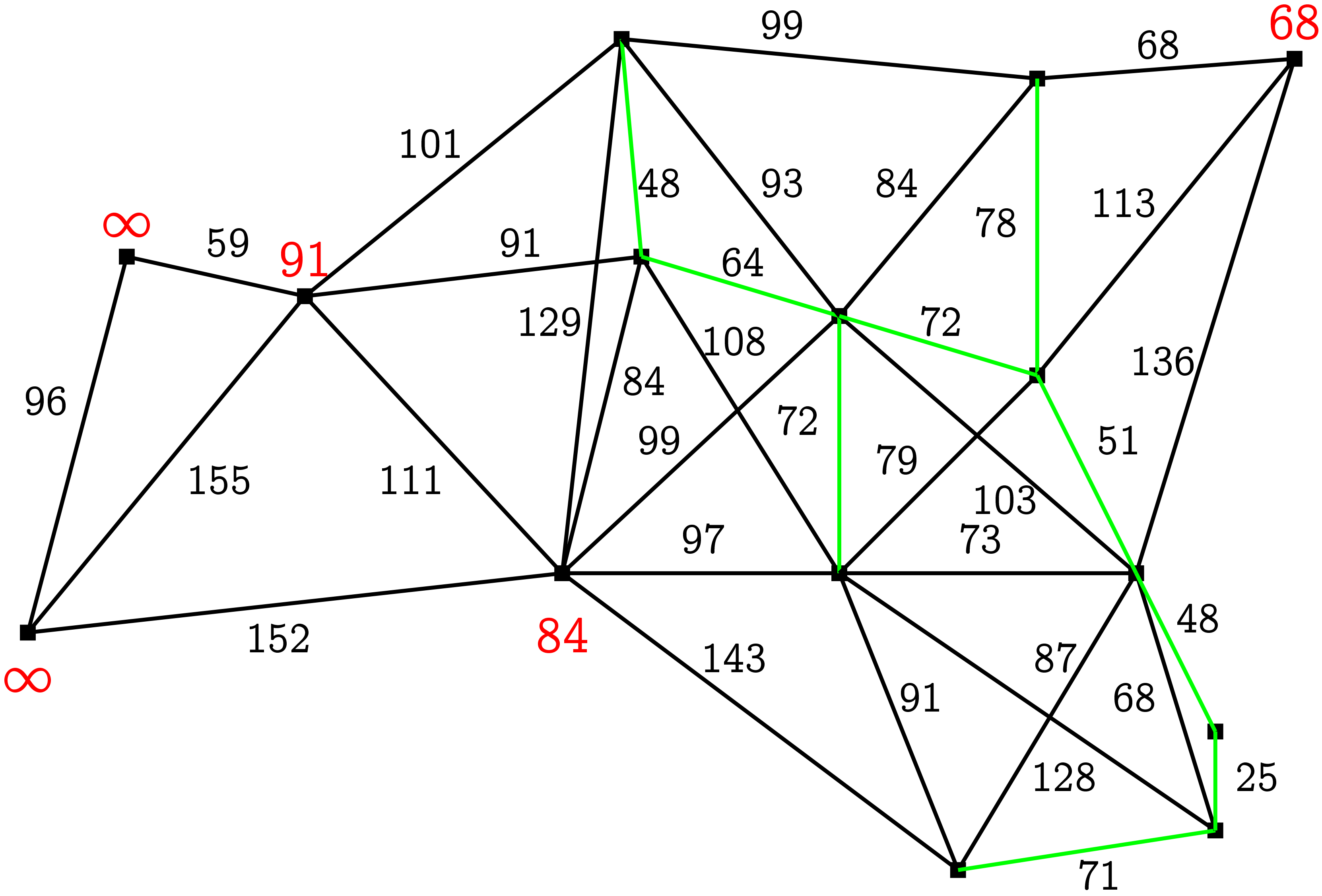

99
78
68
113
101
48
93
84
78
113
∞
59
91
91
64
72
129
108
136
84
96
99
72
79
51
155
111
103
97
73
72
84
48
152
143
87
∞
91
68
25
128
71
71

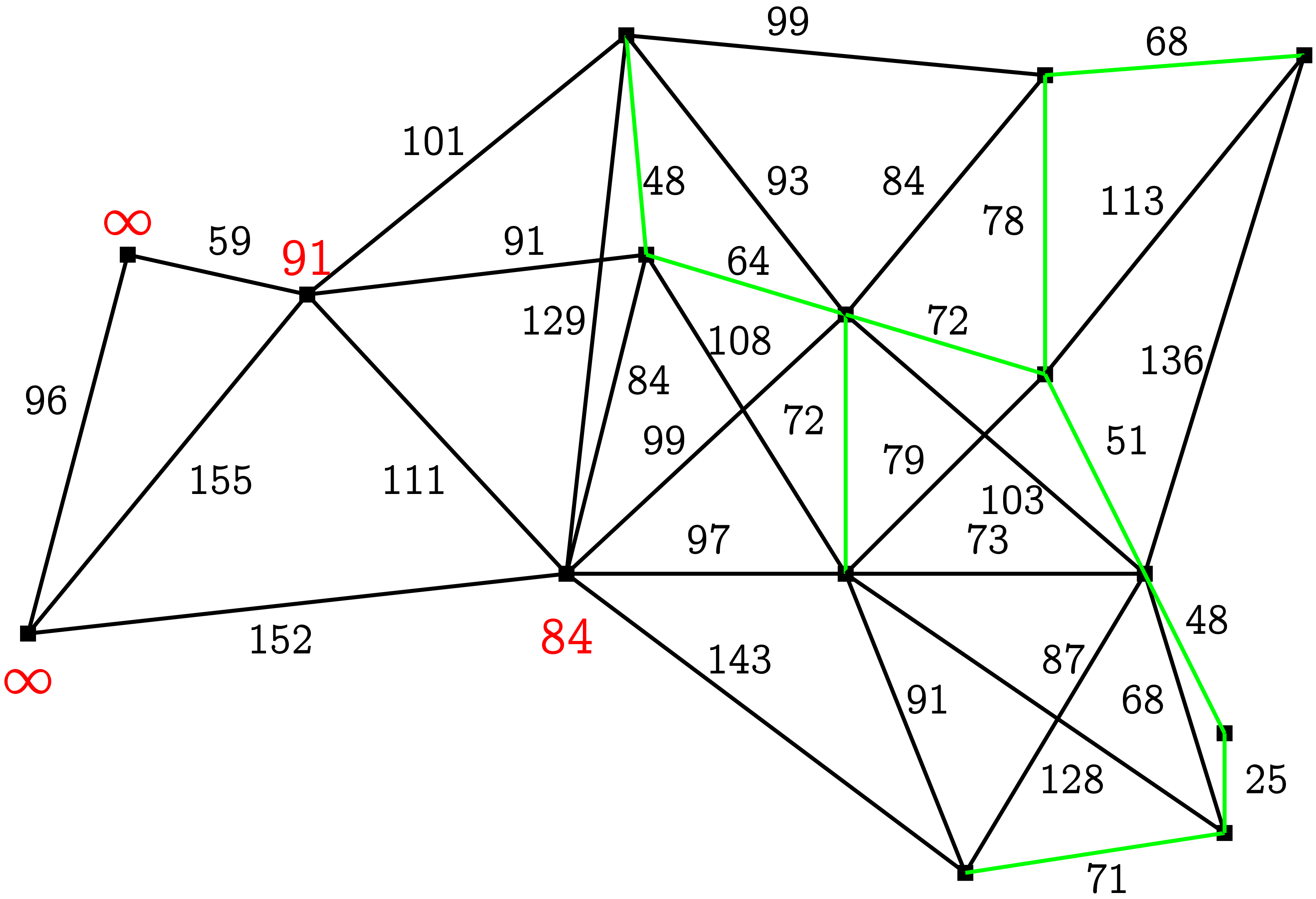


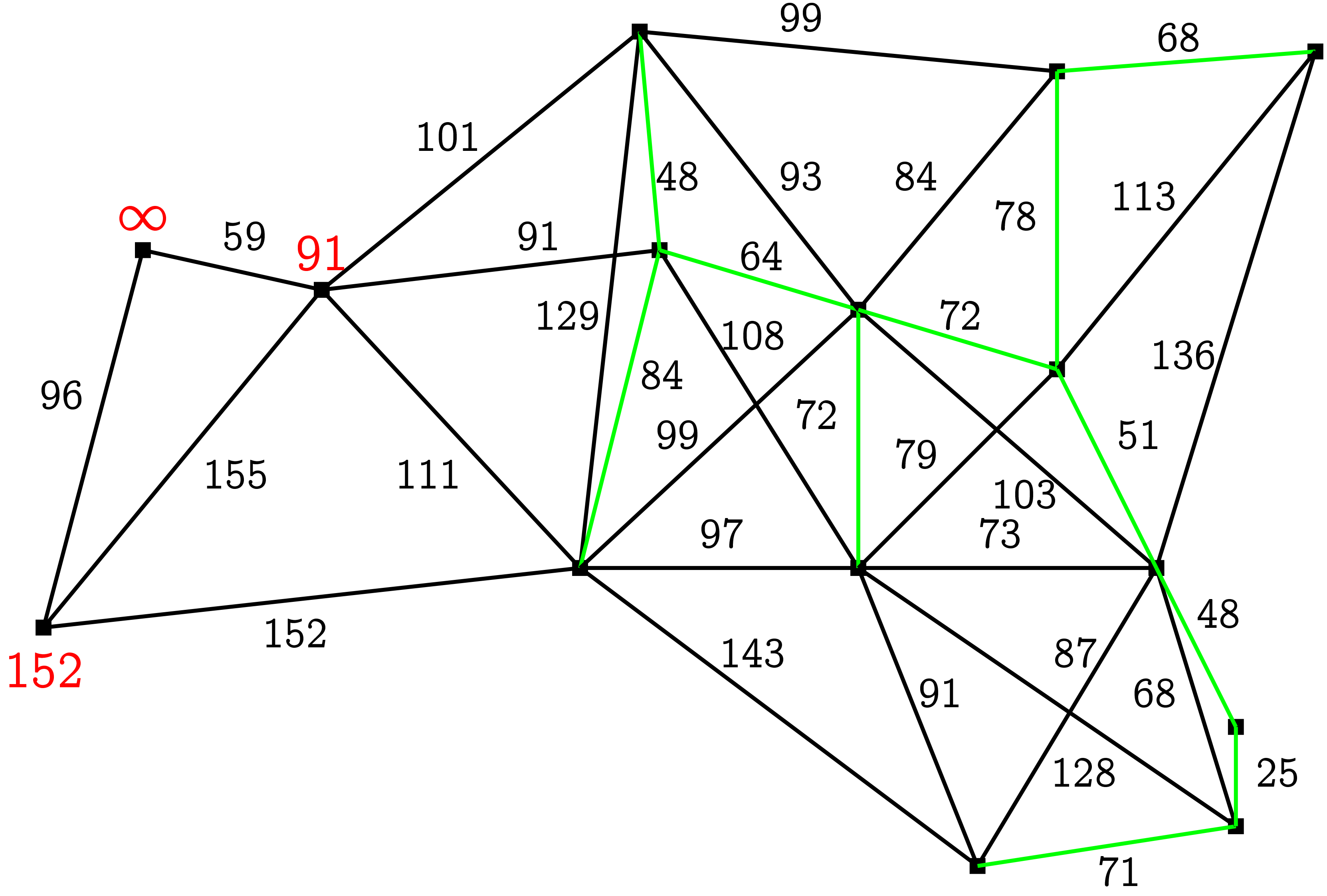
99
78
68
113
101
48
93
84
78
113
∞
59
91
91
64
72
129
108
136
96
84
99
72
79
51
155
111
103
97
73
152
84
72
48
143
87
91
68
128
25
71
∞

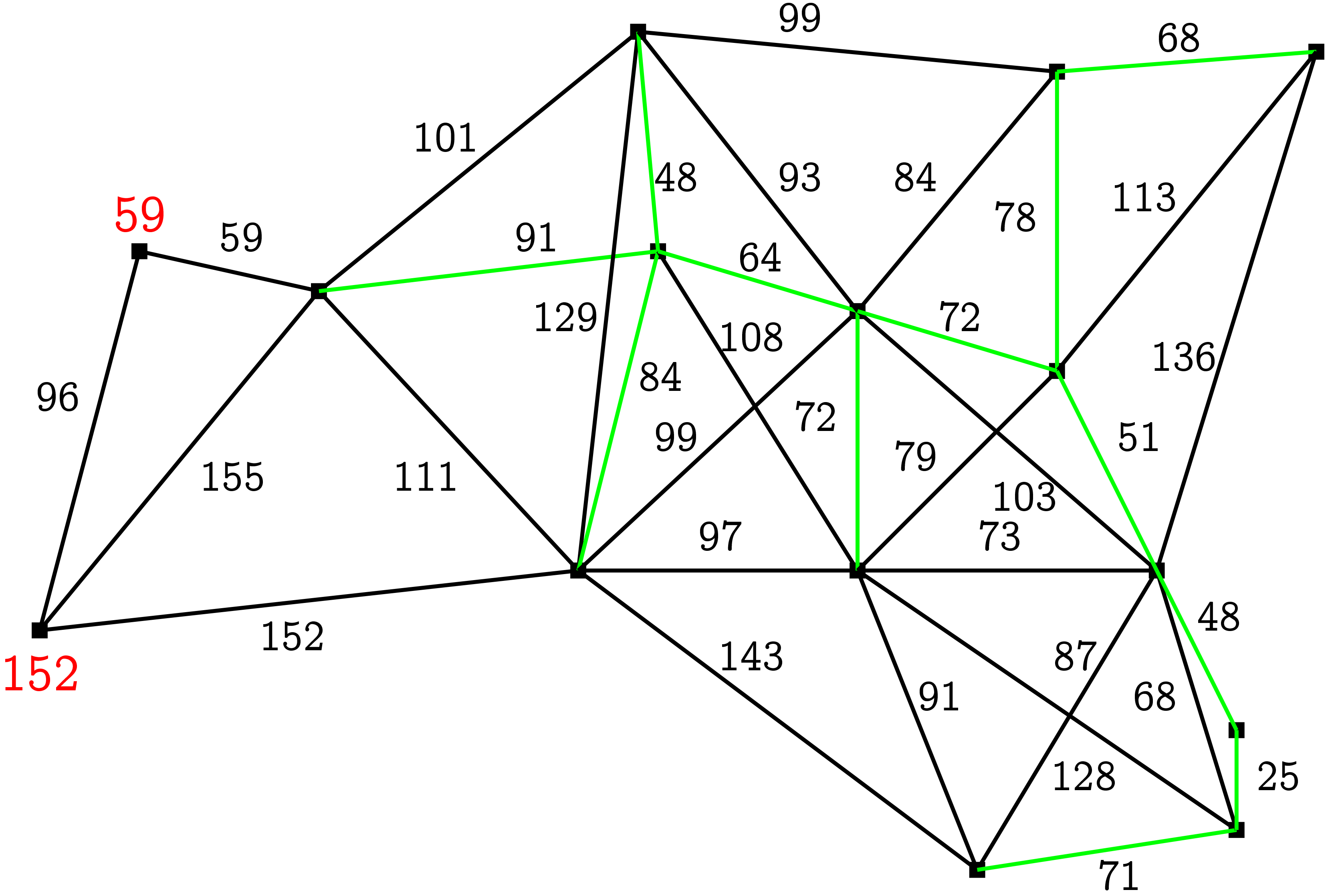


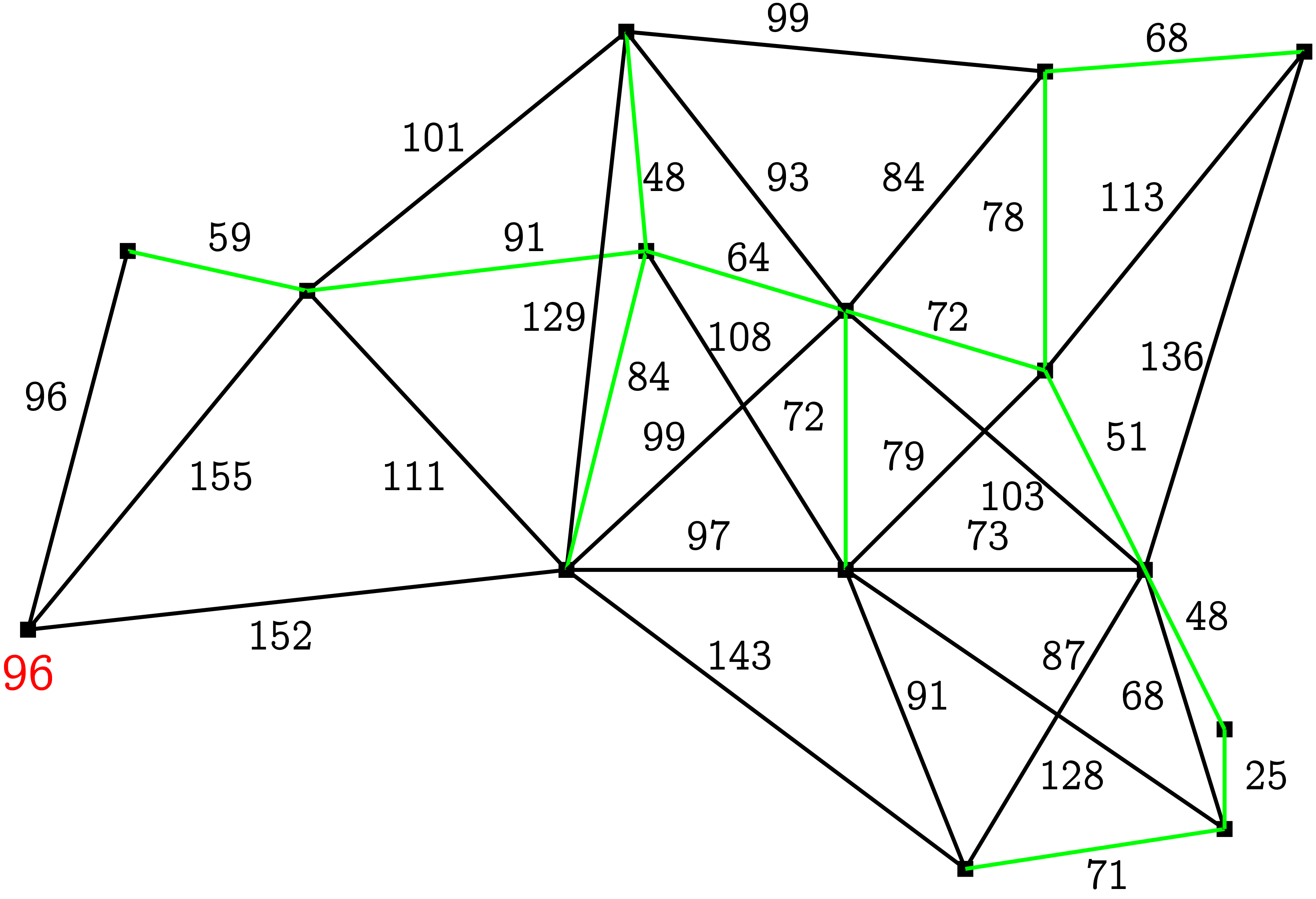
99
78
68
113
101
48
93
84
78
113
∞
59
91
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
152
84
48
∞
143
87
91
68
128
25
71

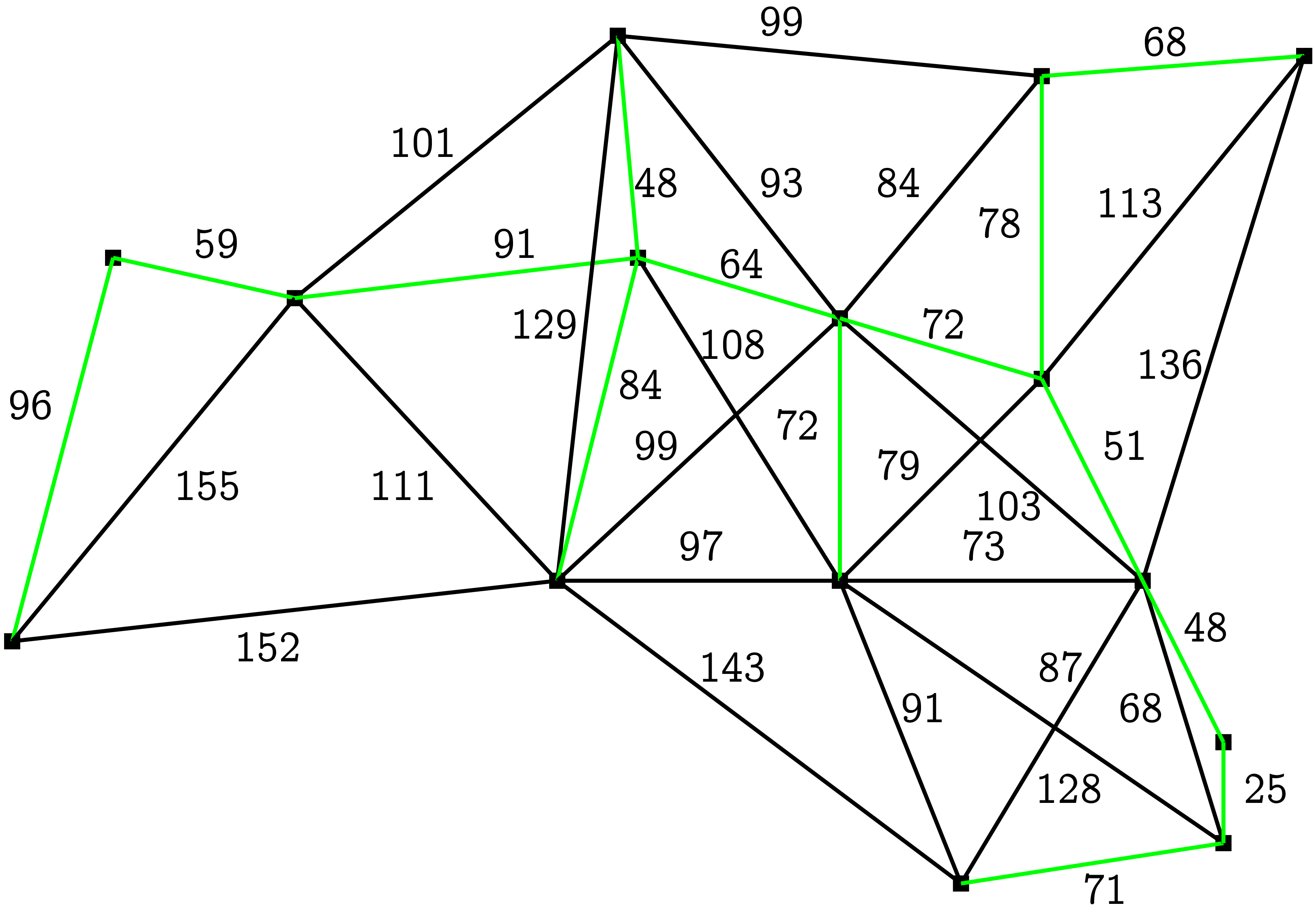

68
99
68
101
48
93
84
78
113
∞
59
91
91
64
72
129
108
136
84
96
99
72
79
51
155
111
103
97
73
152
84
∞
48
143
87
68
91
128
25
71


99
68
101
48
93
84
78
113
∞
59
91
91
64
72
129
108
136
96
84
99
72
79
51
155
111
103
97
73
152
84
48
143
87
∞
91
68
128
25
71


∞
91
152
99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
99
72
79
51
155
111
103
97
73
152
143
87
48
91
68
128
25
71


99
68
101
48
93
84
78
113
59
59
91
64
72
129
108
136
96
84
99
72
79
51
155
111
103
97
73
152
48
143
87
91
68
152
25
128
71


99
68
101
48
93
84
78
113
59
91
64
72
129
108
136
96
84
72
99
79
51
155
111
103
97
73
152
48
143
87
91
68
128
25
71
96


99
68
101
48
93
84
113
78
59
91
64
72
129
108
136
84
96
99
72
79
51
155
111
103
97
73
152
48
143
87
91
68
128
25
71