In [2]:

```
import numpy as np
import random
import mpmath as mp
import pandas as pd
mp.dps=40
```

In [3]:

```
def ABCindeks(drevo): #izracun ABC indeksa za neko drevo, len(drevo[u]) ti da dolzino s
eznama sosedov za vozlisce u torej je to stopnja vozlisca u
    produkt = 1
    for u in drevo:
        for v in drevo[u]:
            produkt = produkt * (len(drevo[u]) + len(drevo[v]) - 2) / (len(drevo[u]) *
len(drevo[v]))
    return sqrt(produkt)

#ABCindeks(G)
```

In [3]:

```
def minABCindeks(n): #izracun najmanjsega ABC indeksa za vsa drevesa z n vozlisci, vrne
tudi seznam dreves, ki imajo ta najmanjsi indeks
    min_indeks = None
    for dr in graphs.trees(n):
        if min_indeks == None:
            min_indeks = ABCindeks(dr)
            drevesa_z_min_ind = [dr]
        else:
            if ABCindeks(dr) < min_indeks:
                min_indeks = ABCindeks(dr)
                drevesa_z_min_ind = [dr]
            elif ABCindeks(dr) == min_indeks:
                drevesa_z_min_ind.append(dr)
            else:
                continue
    return min_indeks , drevesa_z_min_ind

#minABCindeks(9)
```

In [40]:

```
rezultati = []
for n in range(17) : #range naredi seznam od 0 do 18, midva hoceva grafe z vozlisci od
 1 do 19 zato je spodaj n+1
        rez = [vrednost, seznam_grafov] = minABCindeks(n+1)
        rezultati.append(rez)
rezultati #rabu je okol 10 min (za 19)
```

Out[40]:

```
[(1, [Graph on 1 vertex]),
 (0, [Graph on 2 vertices]),
 (1/4, [Graph on 3 vertices]),
 (1/8, [Graph on 4 vertices]),
 (1/16, [Graph on 5 vertices]),
 (1/32, [Graph on 6 vertices]),
 (1/64, [Graph on 7 vertices, Graph on 7 vertices]),
 (1/128, [Graph on 8 vertices, Graph on 8 vertices]),
 (1/256,
  [Graph on 9 vertices,
   Graph on 9 vertices,
   Graph on 9 vertices,
   Graph on 9 vertices]),
 (1/576, [Graph on 10 vertices]),
 (1/1152, [Graph on 11 vertices]),
 (5/12288, [Graph on 12 vertices]),
 (1/5184, [Graph on 13 vertices]),
 (3/32768, [Graph on 14 vertices]),
 (25/589824, [Graph on 15 vertices]),
 (25/1179648, [Graph on 16 vertices, Graph on 16 vertices]),
 (5/524288, [Graph on 17 vertices])]
```

In [4]:

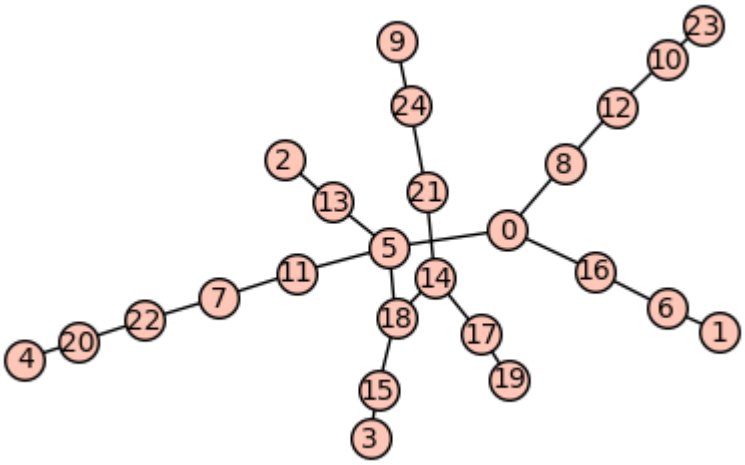
```
def SA_algoritem_boljsi(n, T=100, fun=ABCindeks): #algoritem za iskanje drevesa z najma
njšim ABC indeksom za drevesa z več vozlišči. Algoritm vrne min_indekse in pripajadoca
 drevesa po 100, 1000 in 10000 korakih
    drevo =graphs.RandomTree(n)
    najboljse_drevo, najboljsa_vrednost = drevo, fun(drevo) ### zacetno drevo proglasim
o za najboljso in tudi za trenutno izbrano
    trenutno_drevo, trenutna_vrednost = drevo, fun(drevo)
    for i in range(10000):
        listi_drevesa = []
        stopnje_vozlisc = trenutno_drevo.degree()
        for i in range(n):
            if stopnje_vozlisc[i] == 1:
                listi_drevesa.append(i)               ### sedaj imamo vse liste drevesa
        stevilo_listov = len(listi_drevesa)
        nakljucni_list = listi_drevesa[random.randint(0,stevilo_listov - 1)] ### izbere
mo nakljucni list
        popravek = copy(trenutno_drevo)   ### ustvarimo kopijo drevesa na katerem bomo
 izvedli zamenjavo
        ostala_vozlisca = [x for x in popravek.vertices() if x!=nakljucni_list]
        povezava = (nakljucni_list, trenutno_drevo[nakljucni_list][0]) ### list bo imel
e enega soseda
        popravek.delete_edge(povezava)
        izbrano_vozlisce = ostala_vozlisca[random.randint(0, len(ostala_vozlisca) -1)]
        popravek.add_edge(nakljucni_list, izbrano_vozlisce) #### dodamo povezavo na kop
iji
        vrednost_popravka = fun(popravek)
        if (najboljsa_vrednost > vrednost_popravka):
             najboljse_drevo, najboljsa_vrednost = popravek, fun(popravek)
             trenutno_drevo, trenutna_vrednost = popravek, fun(popravek)
        elif (trenutna_vrednost > vrednost_popravka) or (mp.exp((-(vrednost_popravka -
trenutna_vrednost)/T)) > random.random()): ### popravek je sprejet ce je bila vrednost
 popravka manjsa od trenutne vrednosti ali pa
            trenutno_drevo, trenutna_vrednost = popravek, fun(popravek)
        T = T/(i+1)
    return najboljse_drevo
```

In [5]:

```
for i in range(25,41):
    print("Graf s {} vozlisci".format(i))
    show(SA_algoritem_boljsi(i))
```

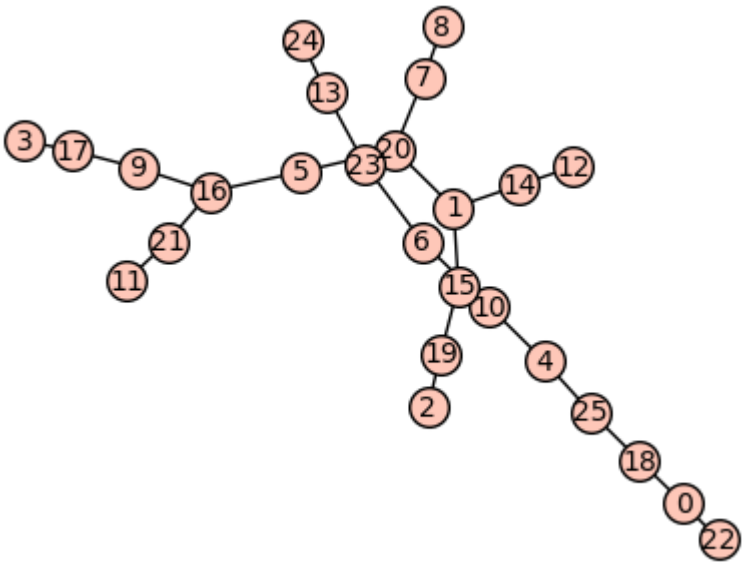
Graf s 25 vozlisci

Out[5]:



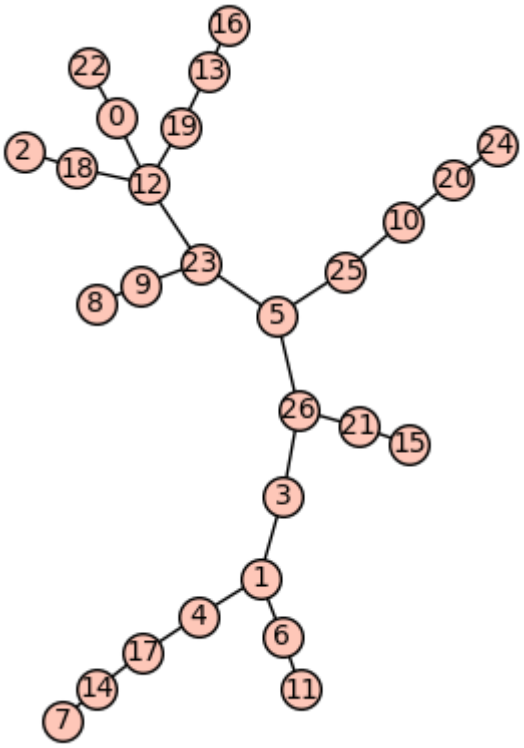
Graf s 26 vozlisci

Out[5]:



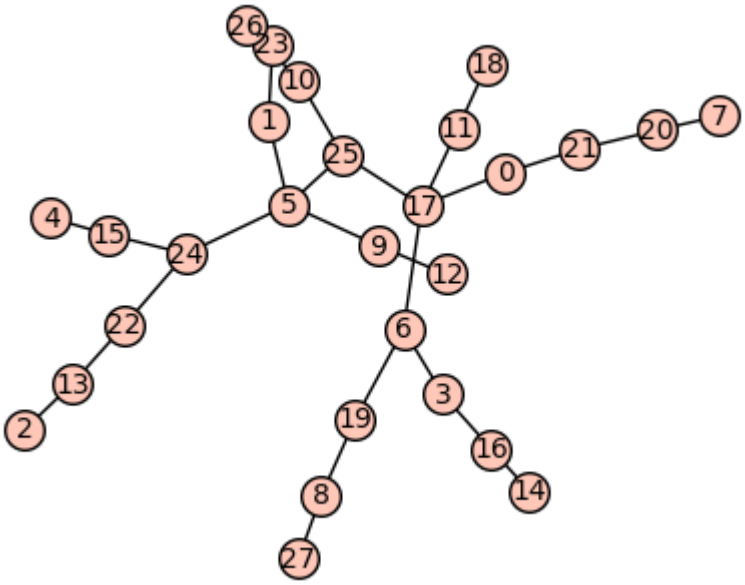
Graf s 27 vozlisci

Out[5]:



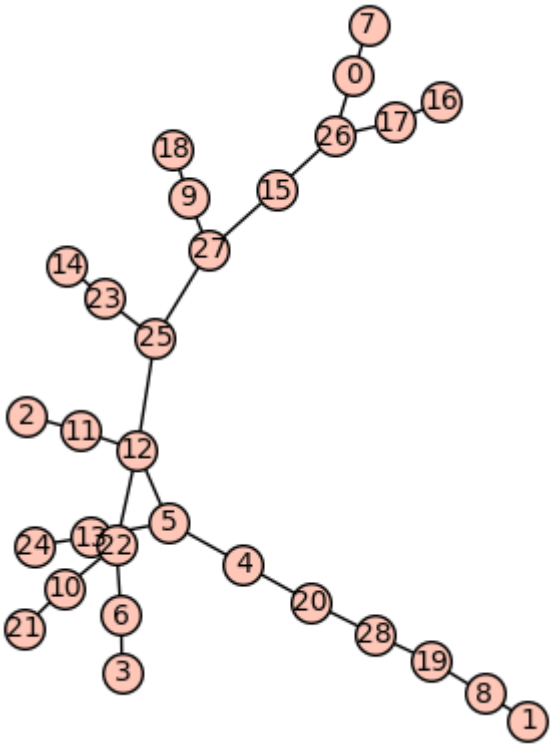

```
Graf s 28 vozlisci
```

Out[5]:



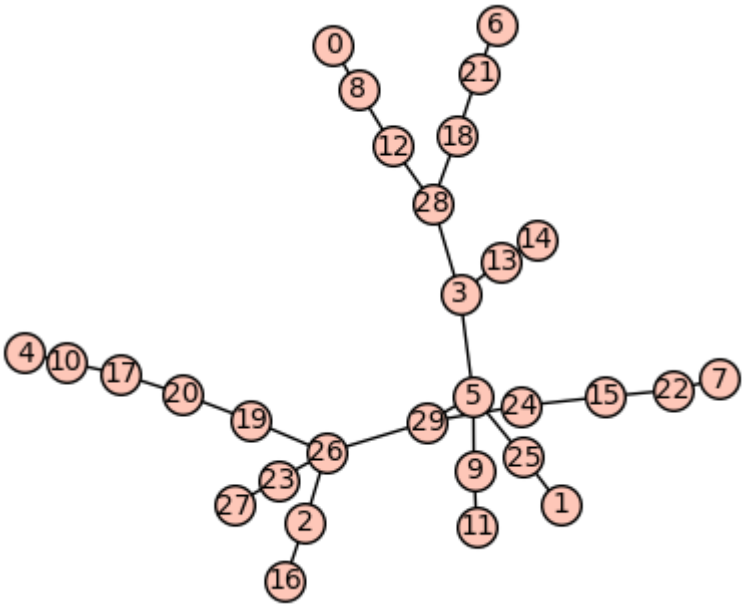
Graf s 29 vozlisci

Out[5]:



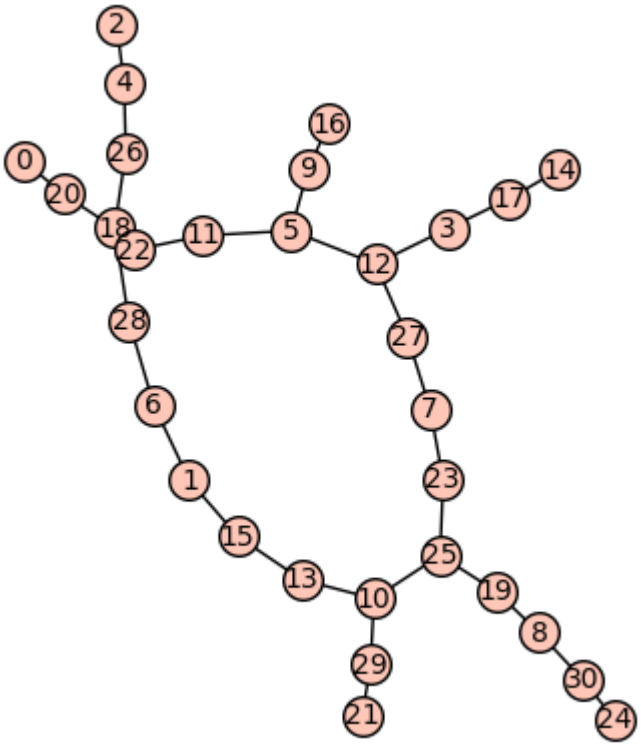
```
Graf s 30 vozlisci
```

Out[5]:



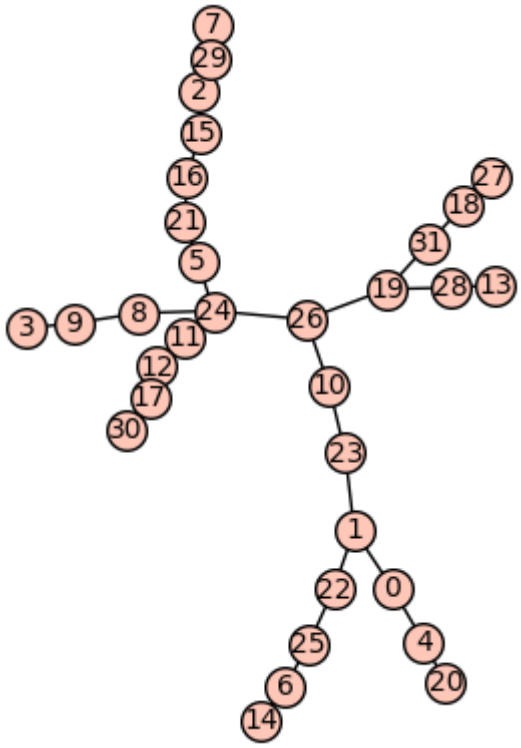

```
Graf s 31 vozlisci
```

Out[5]:



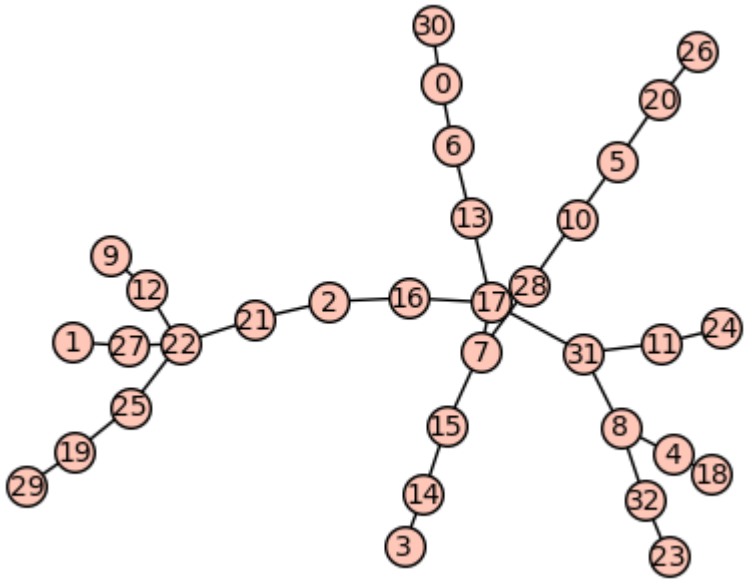

```
Graf s 32 vozlisci
```

Out[5]:



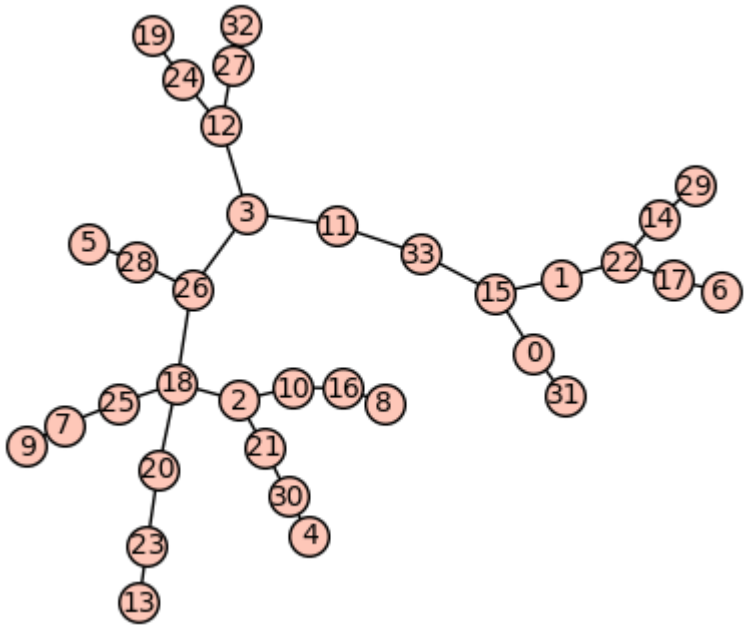
Graf s 33 vozlisci

Out[5]:



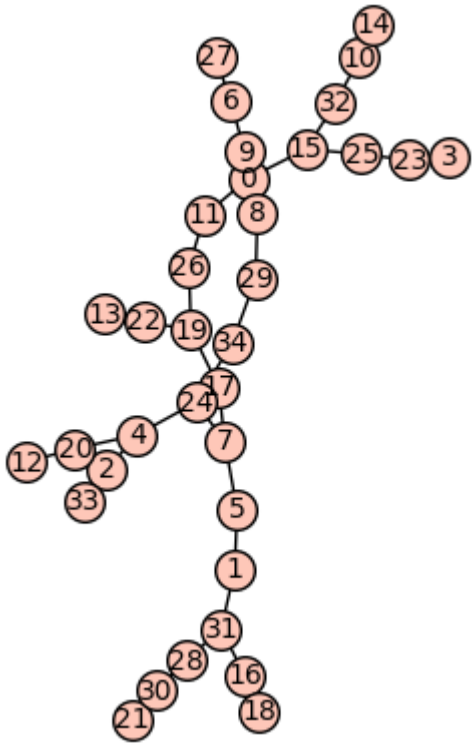
Graf s 34 vozlisci

Out[5]:



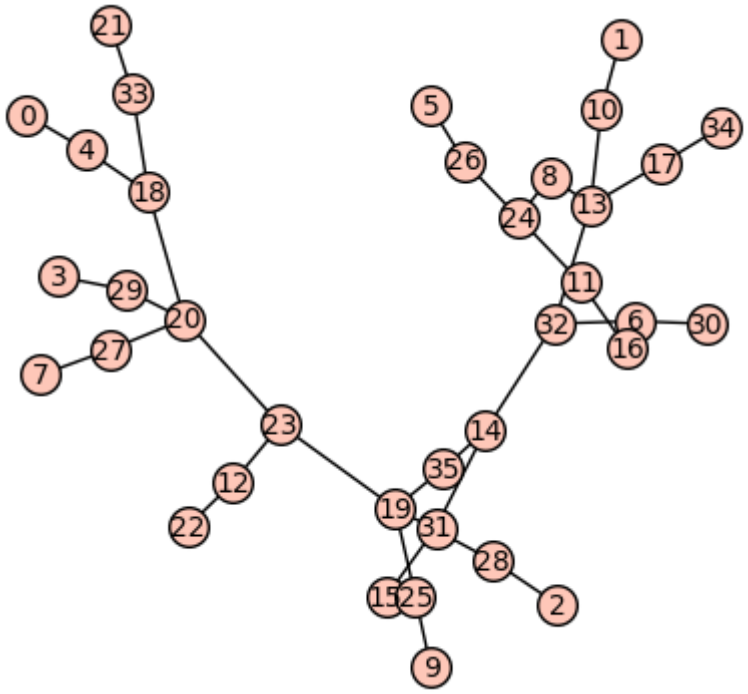
Graf s 35 vozlisci

Out[5]:



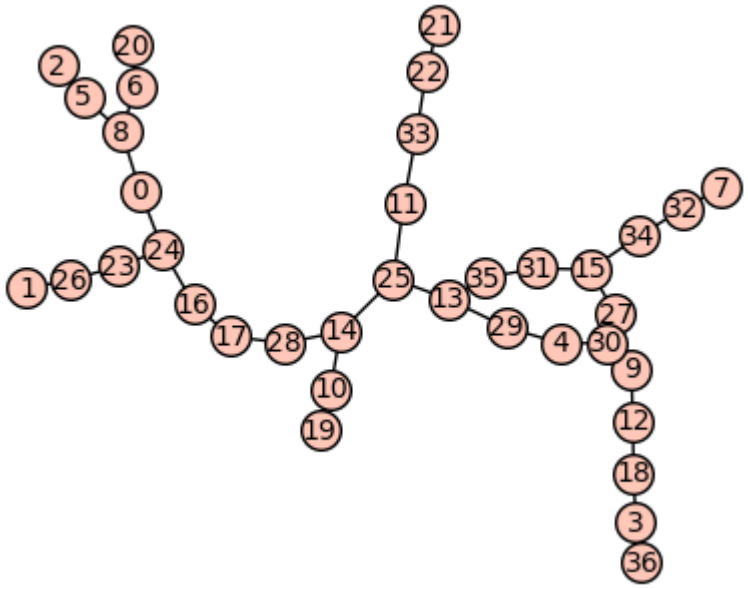
Graf s 36 vozlisci

Out[5]:



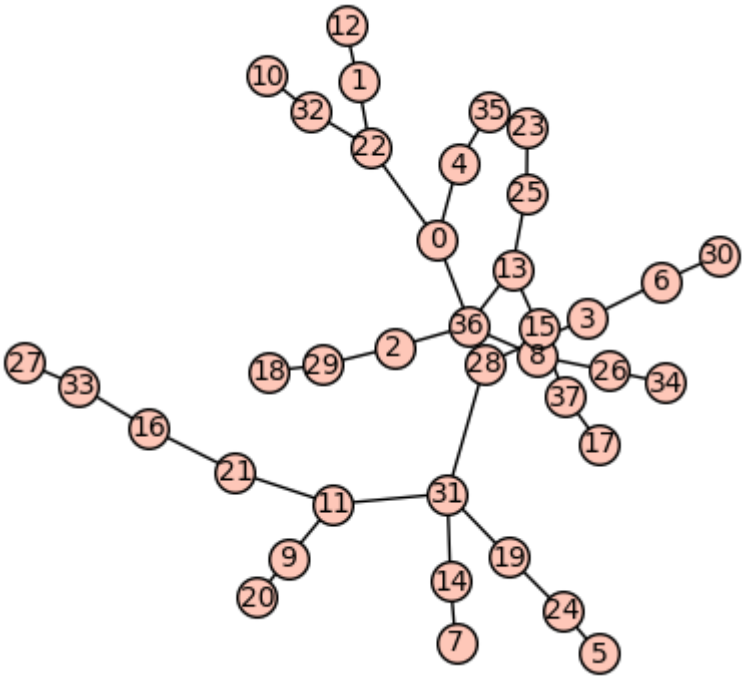
Graf s 37 vozlisci

Out[5]:



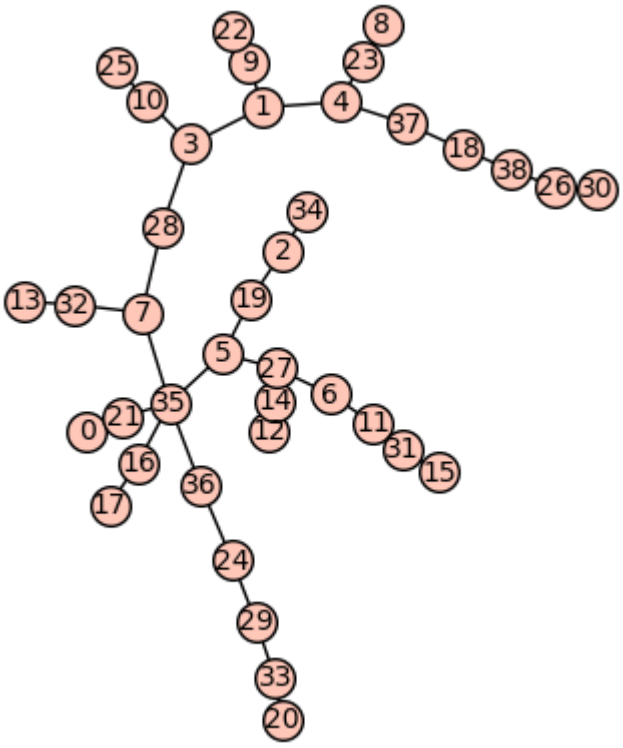
Graf s 38 vozlisci

Out[5]:



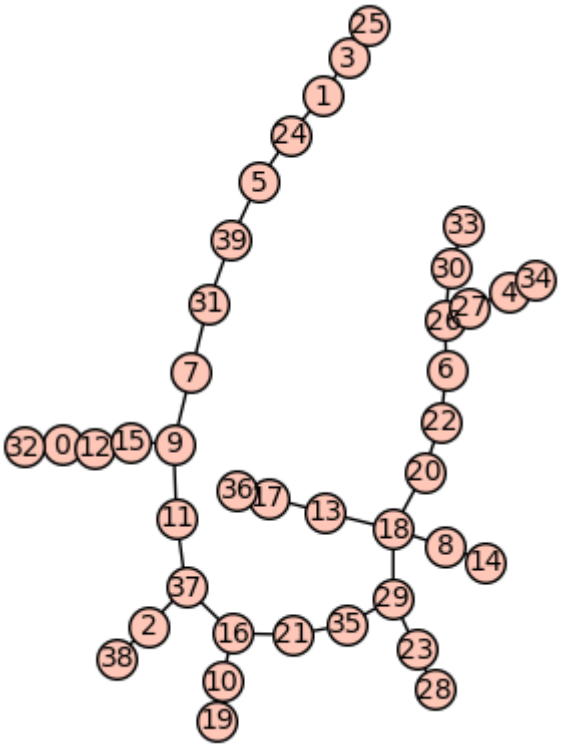
Graf s 39 vozlisci

Out[5]:



Graf s 40 vozlisci

Out[5]:


25
3
1
24
5
39
31
7
32
0
12
15
9
11
37
2
38
16
10
19
21
35
29
23
28
36
17
13
18
20
22
6
26
27
30
33
4
34
8
14