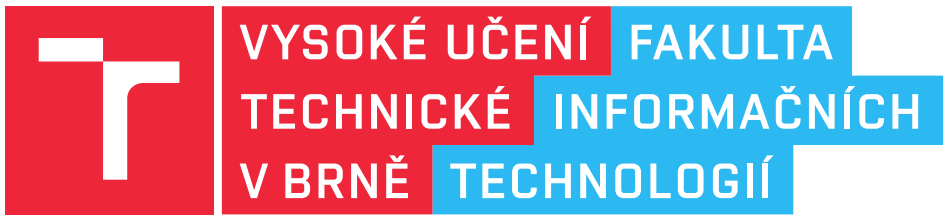
Téma 21 - Paralelizace
Maximální tok v síti

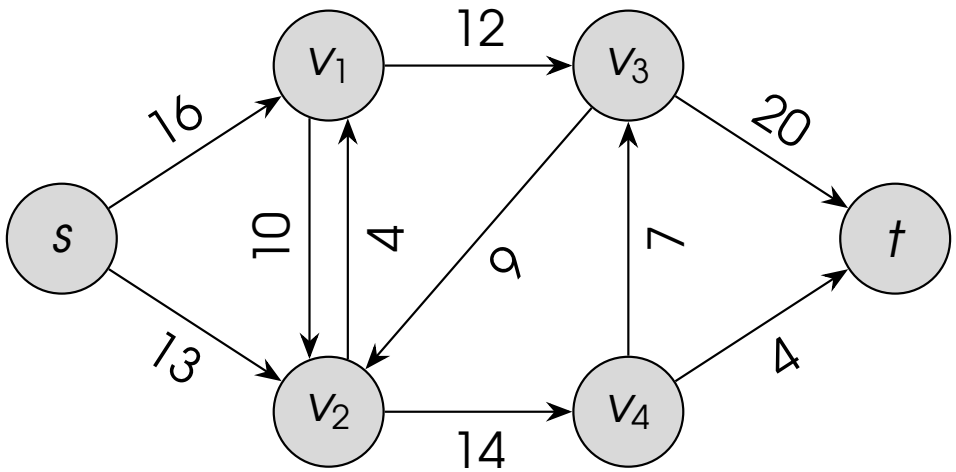
Michal Dostál, Oliver Kuník

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2. 612 66 Brno - Královo Pole, Czech Republic

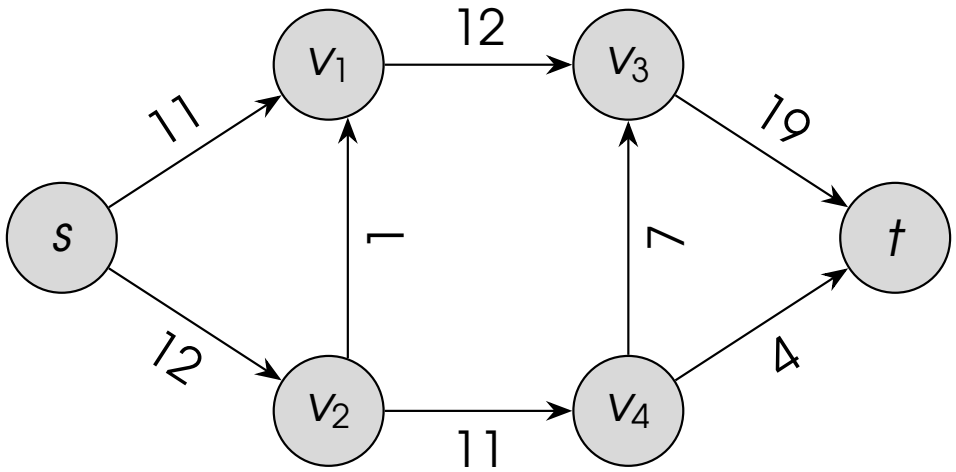


14. prosince 2022

- Síť $G = (V, E)$ je orientovaný graf,
- kde každá hrana $(u, v) \in E$ má nezápornou kapacitu $c(u, v) \geq 0$
- Nechť $c(u, v) = 0$, pokud $(u, v) \notin E$.
- Jsou specifikovány dva uzly: zdroj s a spotřebič t
- Každý uzel leží na cestě z s do t, tj. $s \to v \to t$ pro každý uzel $v \in V$
- Síť je tedy souvislý graf a $m \geq n - 1$



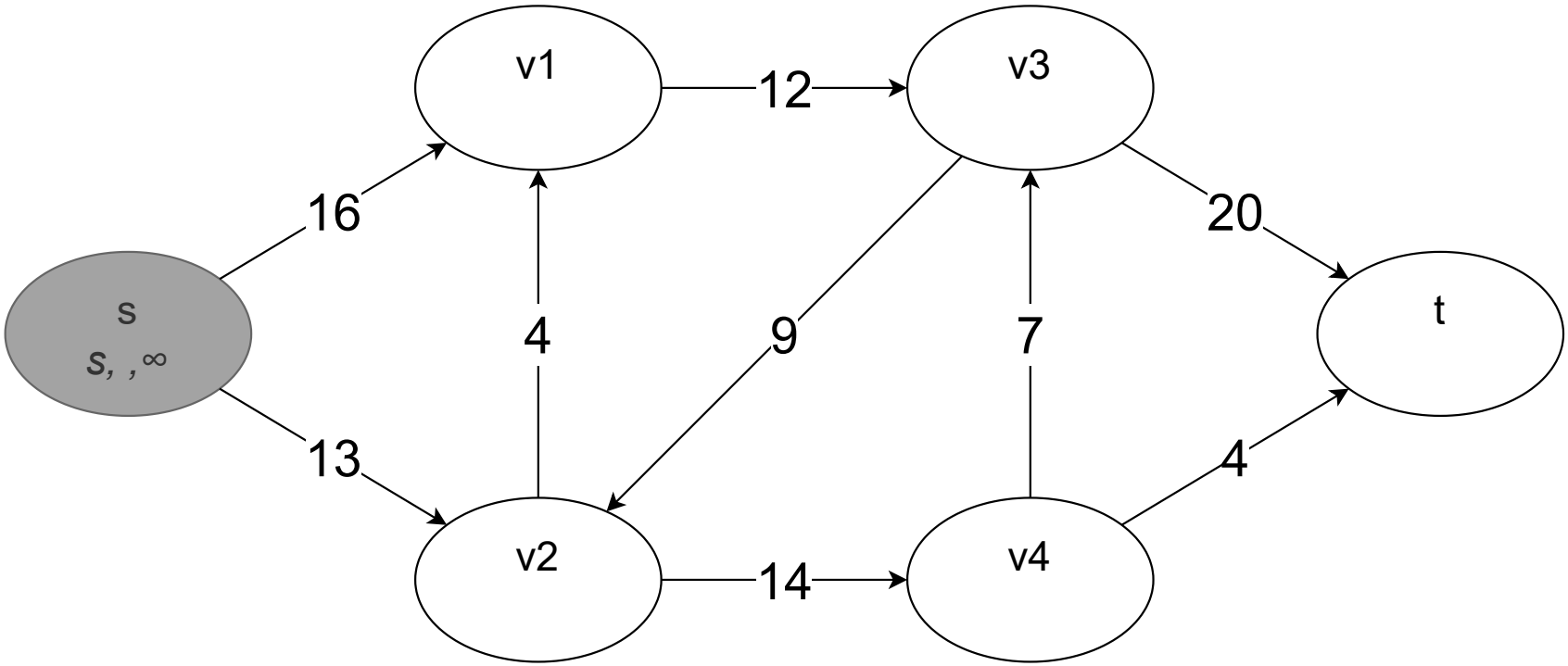
- Máme danou síť G se zdrojem s a spotřebičem t
- Hledáme tok maximální velikosti



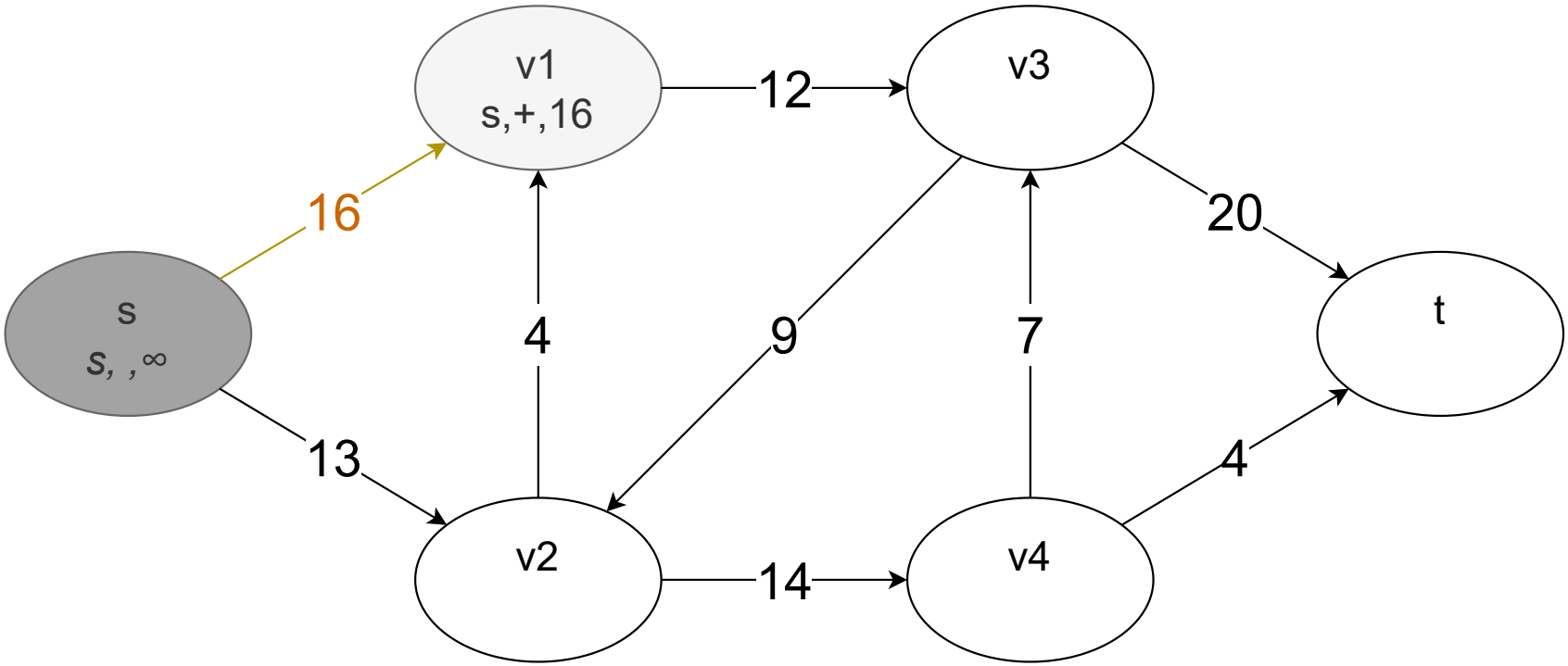
Obrázek: Síť po provedení algoritmu Ford Fulkerson.

```
FORD-FULKERSON-METHOD(G, s, t)
    inicializuj f (u, v) = 0 pro u, v in V
    while existuje zlepšující cesta p
    do zlepši tok f podle p
    return f
```

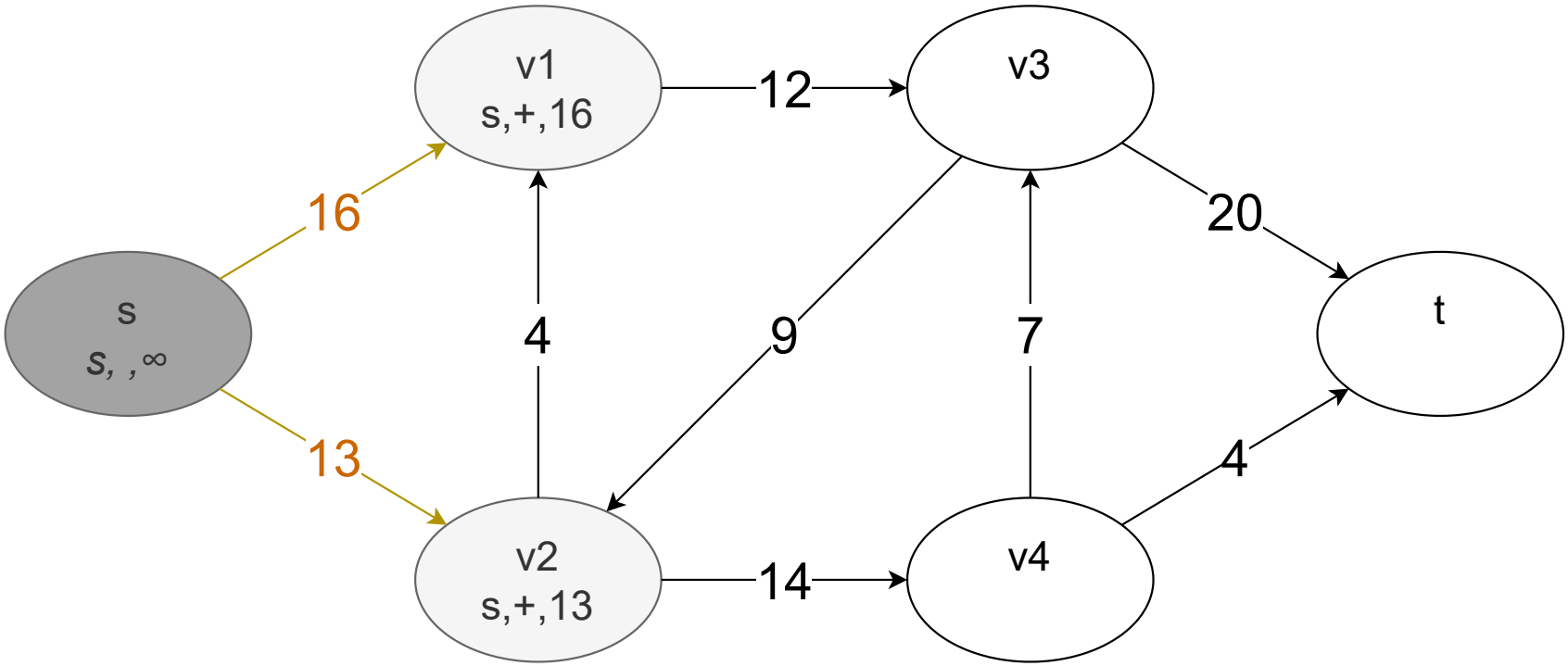
Sekvenčné BFS



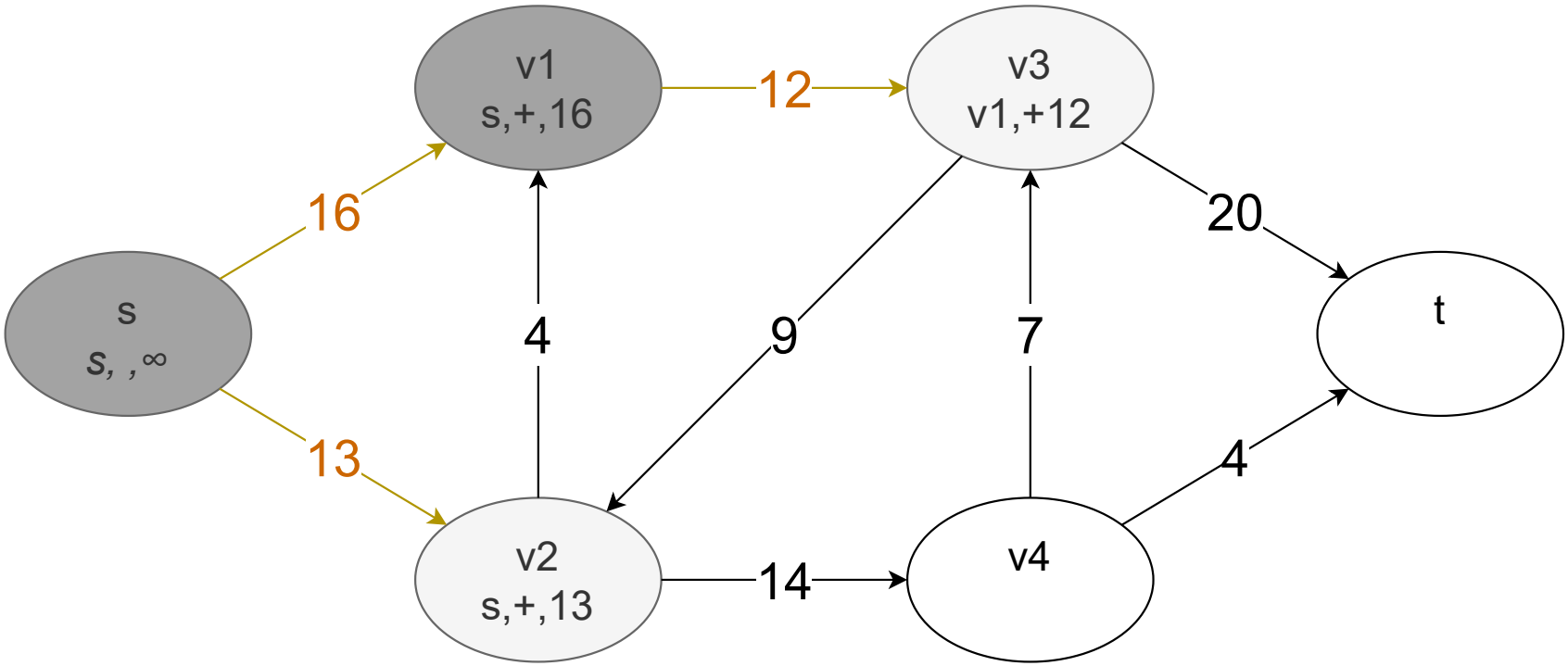
Sekvenčné BFS



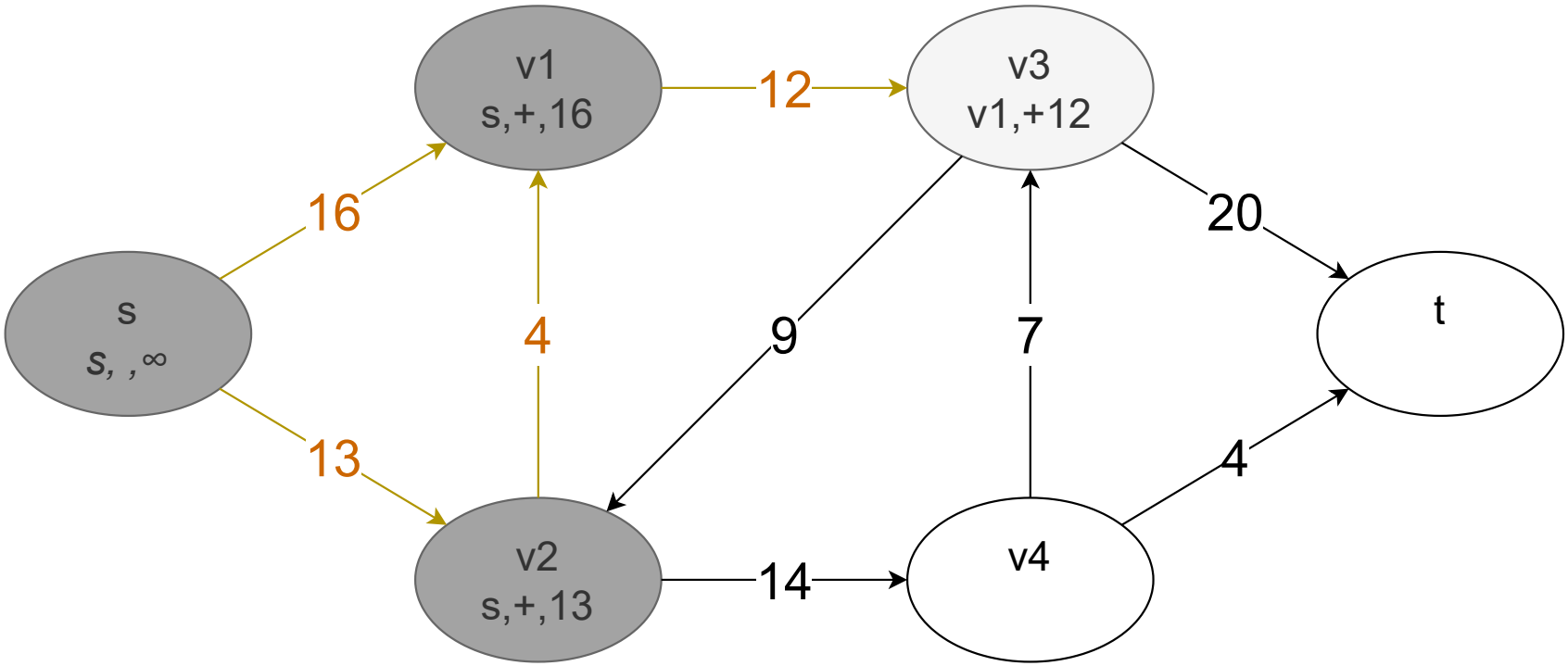
Sekvenčné BFS



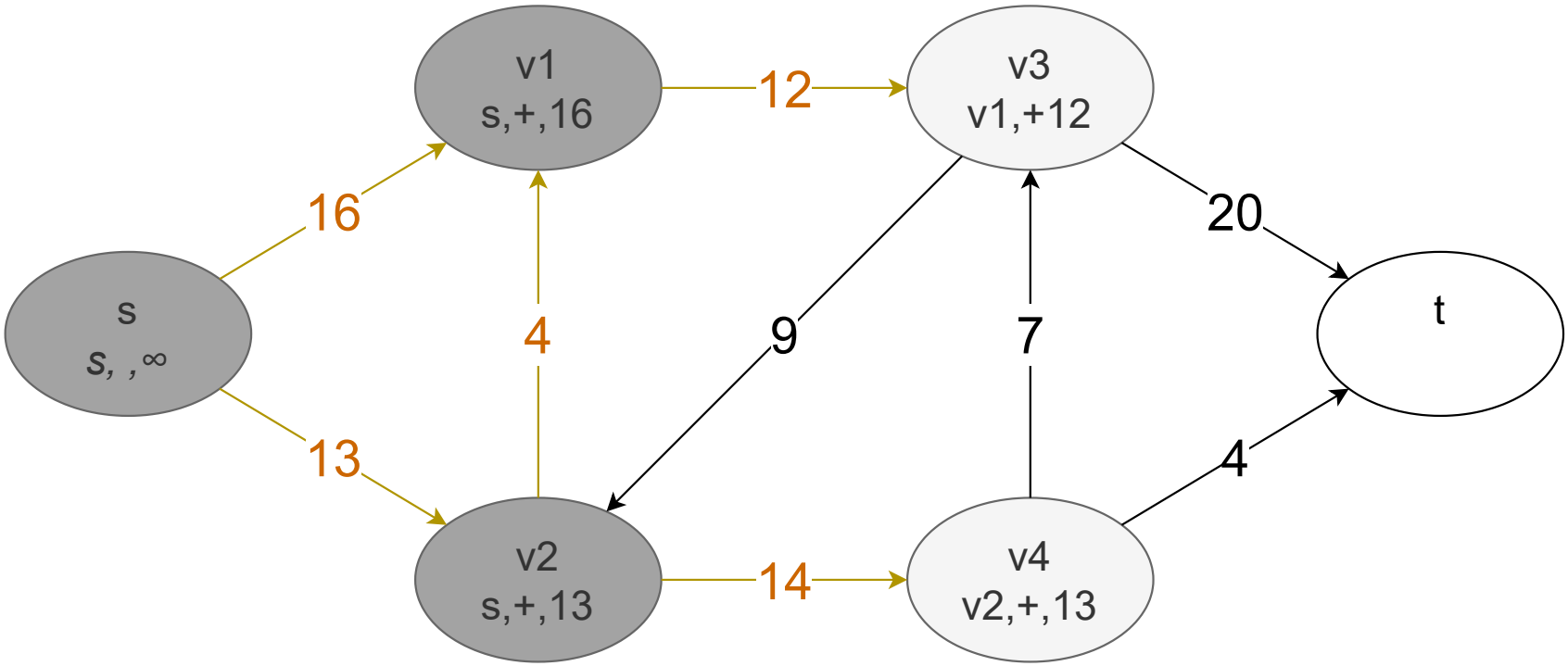
Sekvenčné BFS



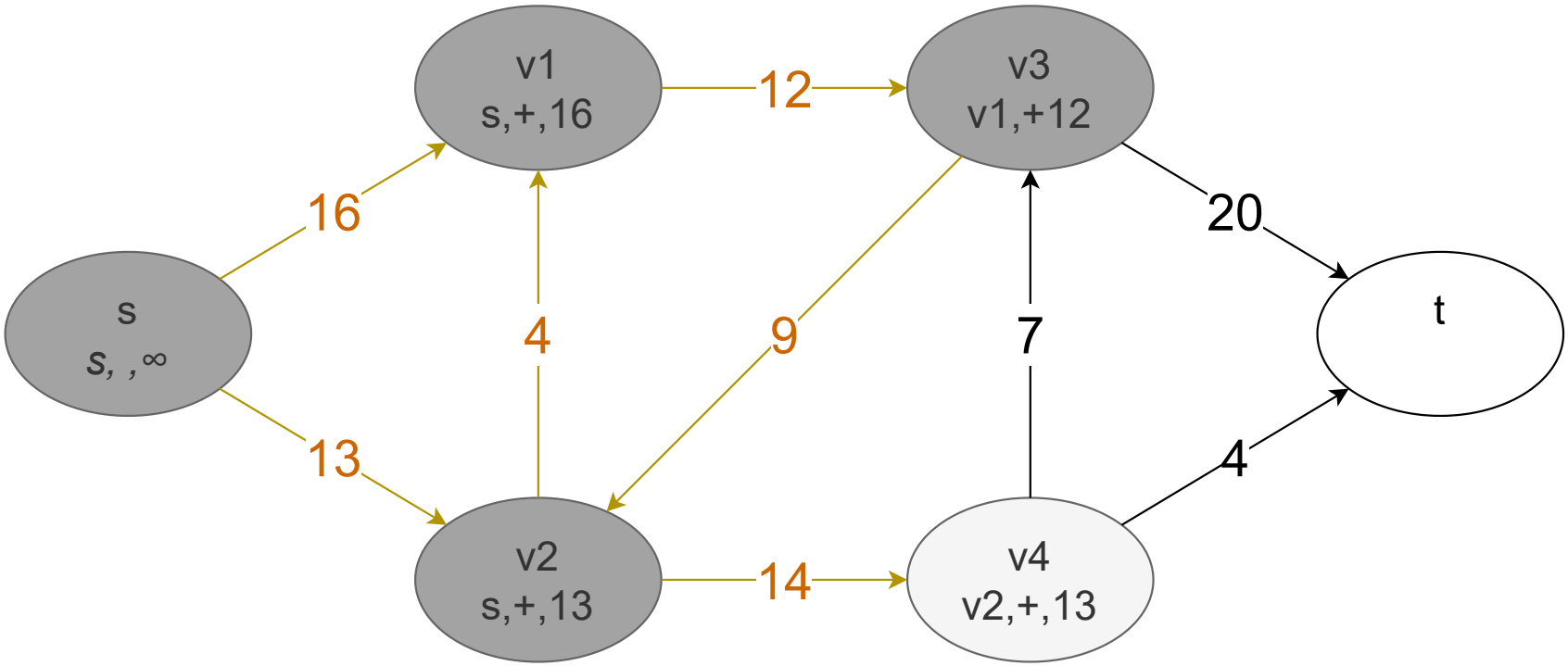
Sekvenčné BFS



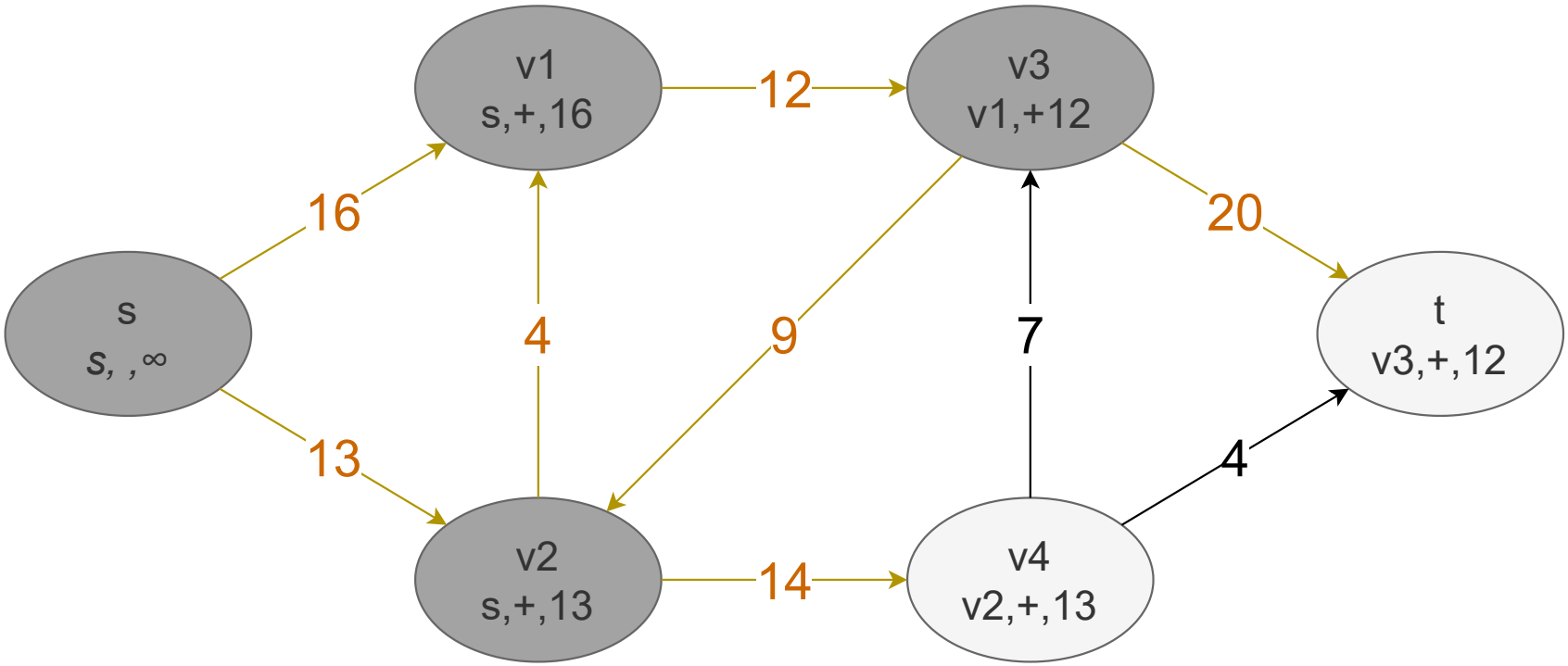
Sekvenčné BFS



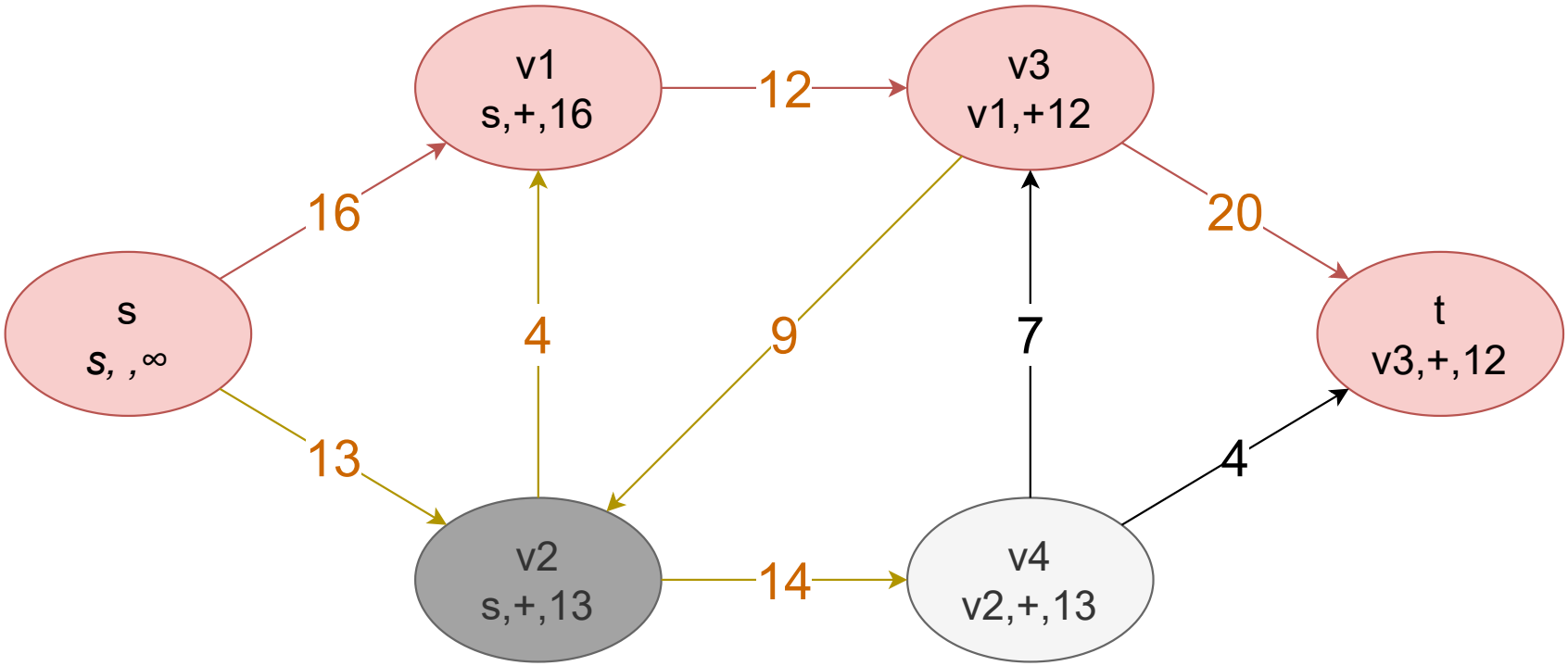
Sekvenčné BFS

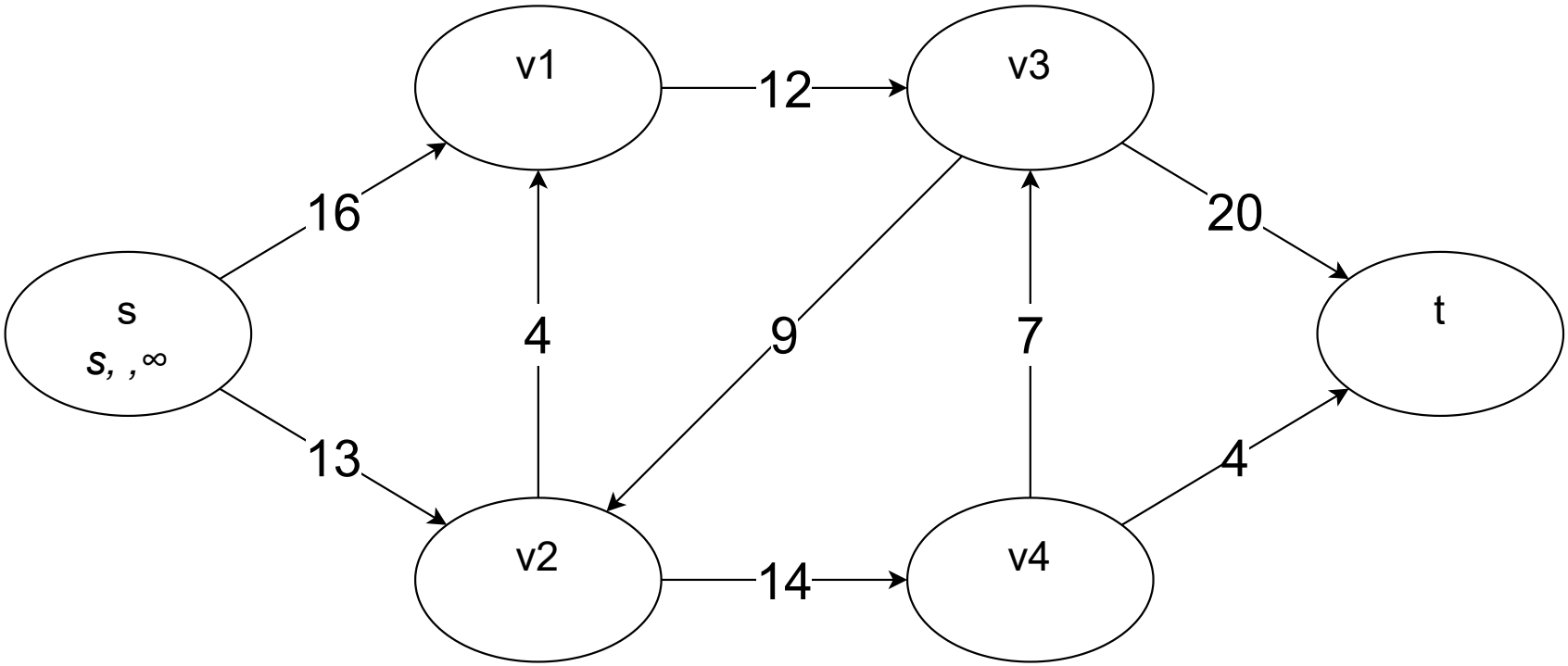


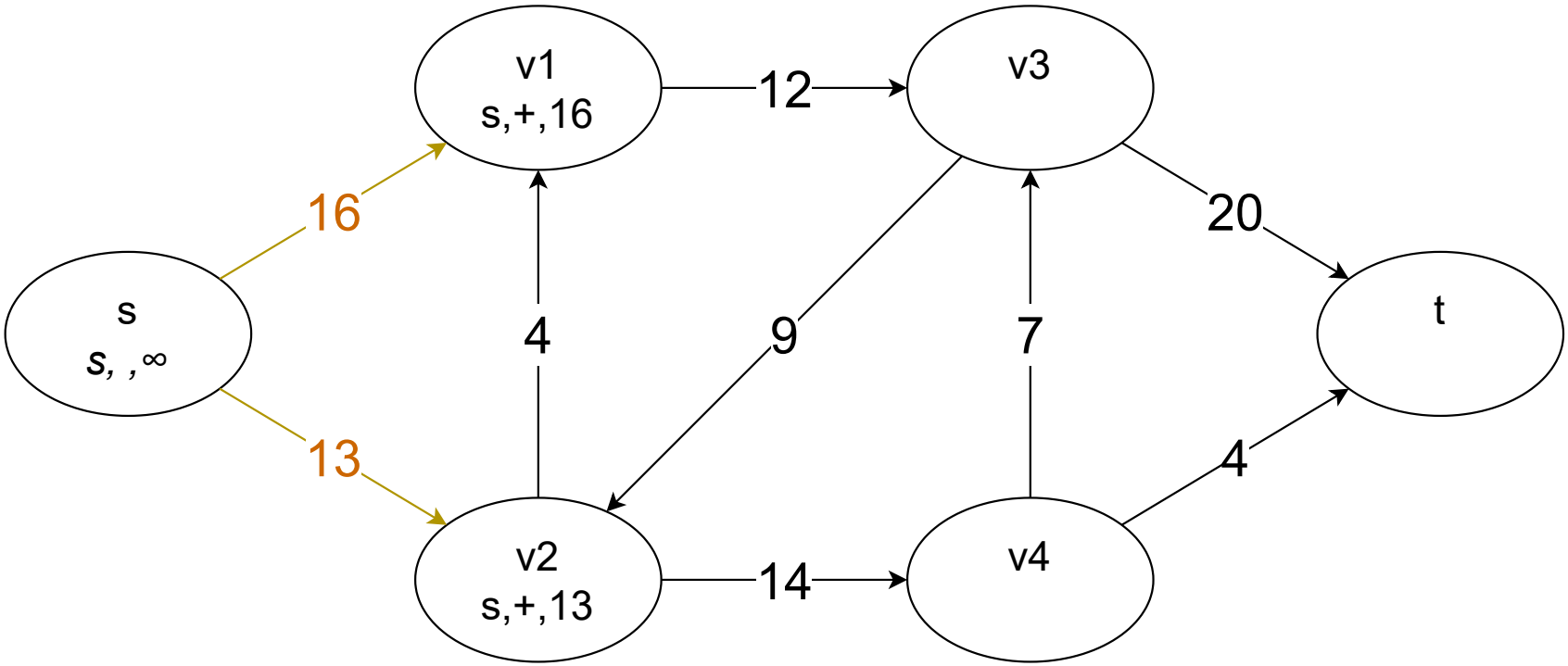
Sekvenčné BFS



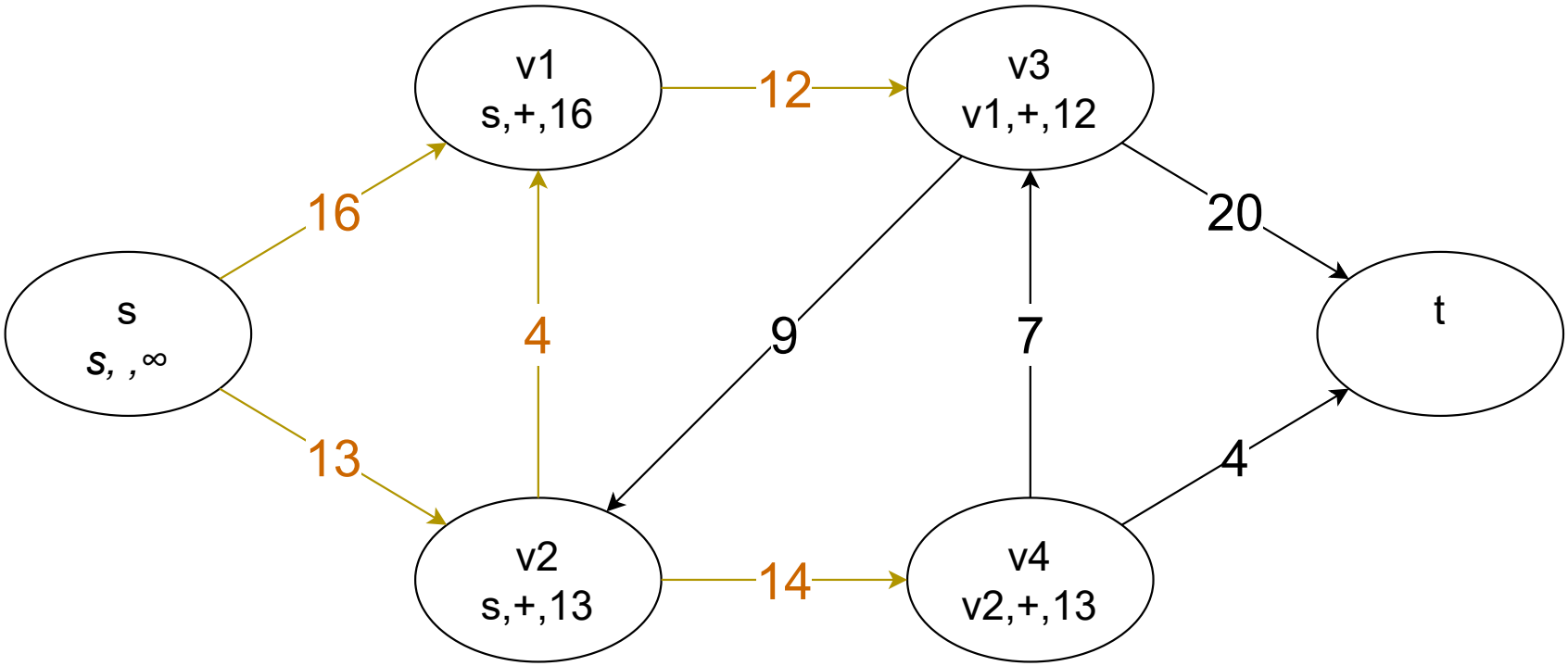
Sekvenčné BFS

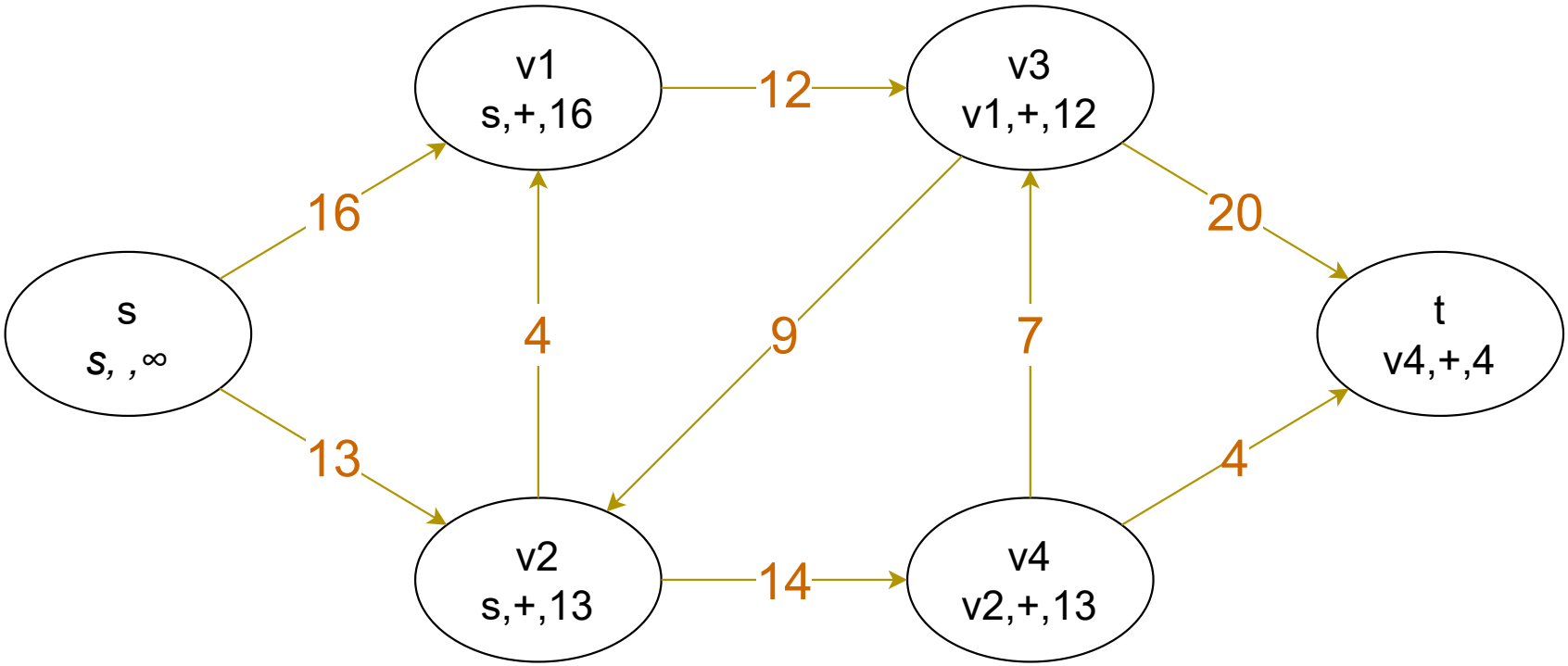


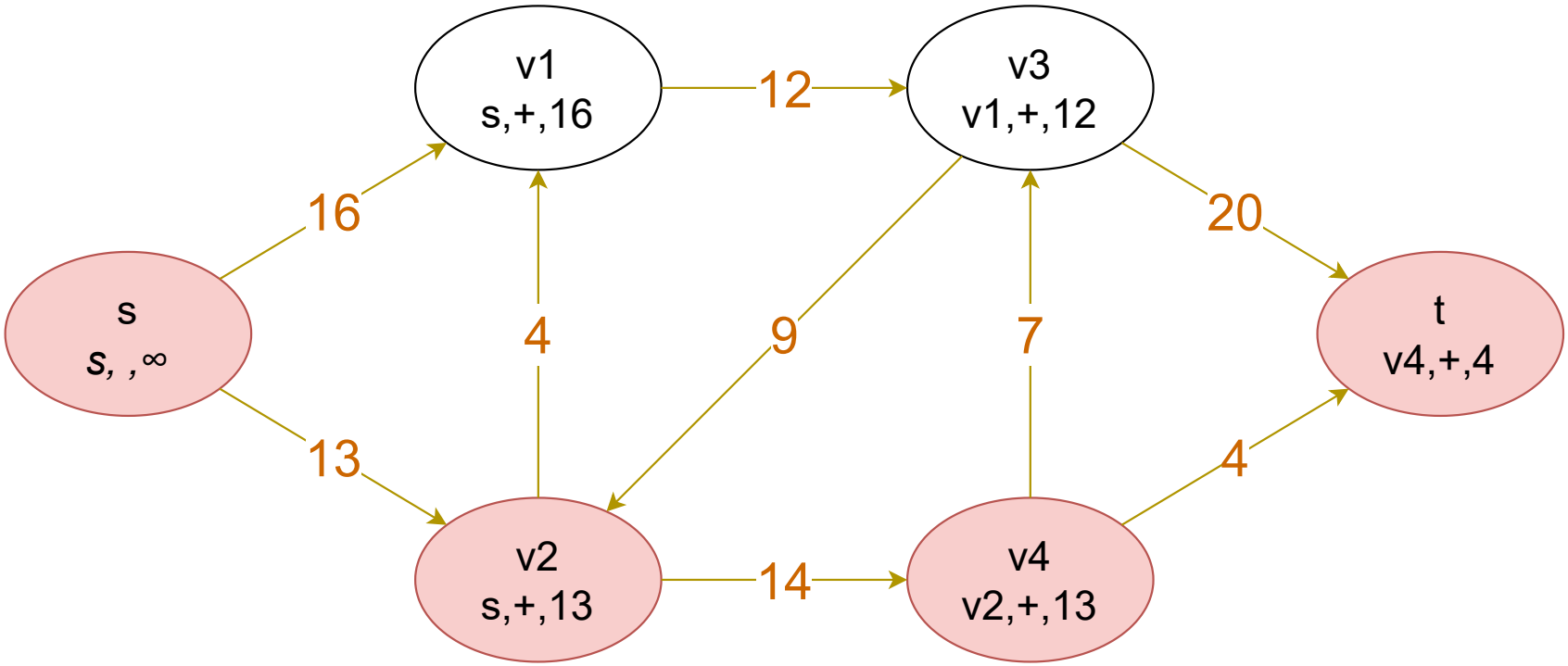

v1
12
v3
16
20
s
s, , ∞
4
9
7
t
13
4
v2
14
v4


v1
s,+,16
12
v3
16
20
s
s, ,∞
4
9
7
t
13
4
v2
s,+,13
14
v4

Paralelné BFS




v1
s,+,16
12
v3
v1,+,12
16
20
s
s, ,∞
4
9
7
t
v4,+,4
13
4
v2
s,+,13
14
v4
v2,+,13

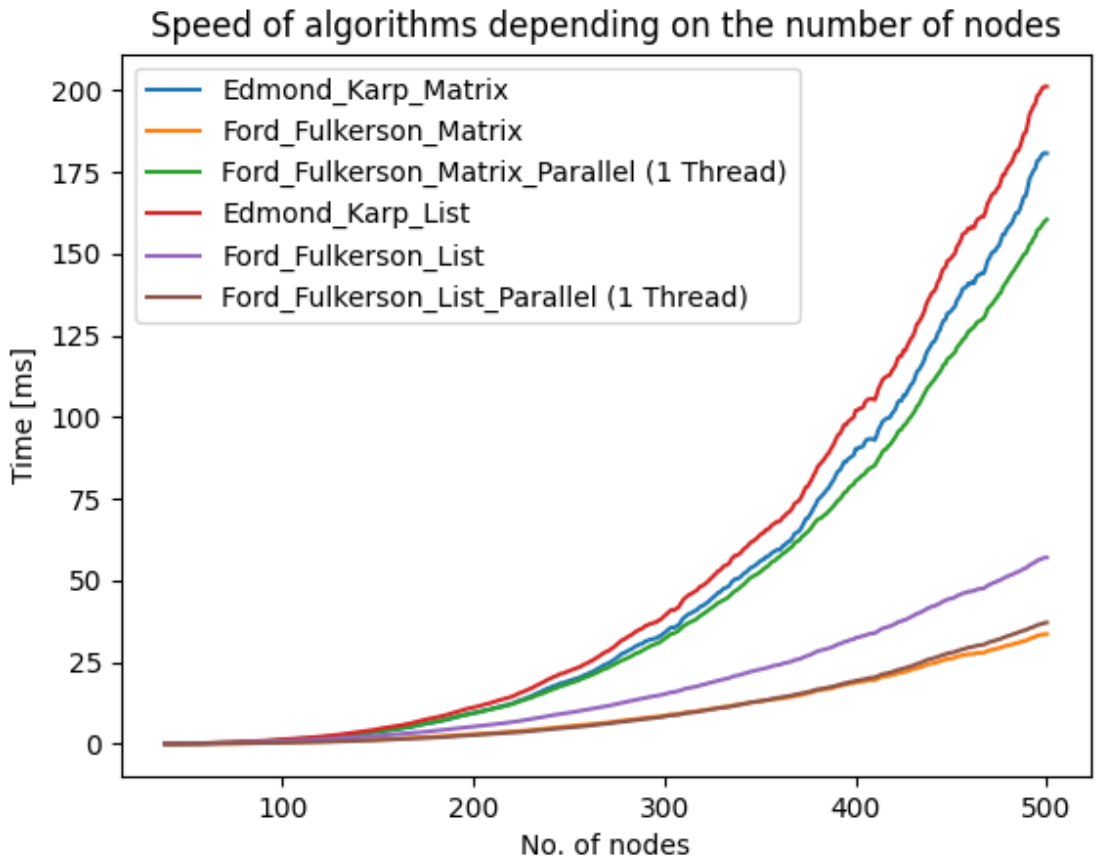

v1
s,+,16
12
v3
v1,+,12
16
20
s
s, ,∞
4
9
7
t
v4,+,4
13
4
v2
s,+,13
14
v4
v2,+,13

Generování sítí

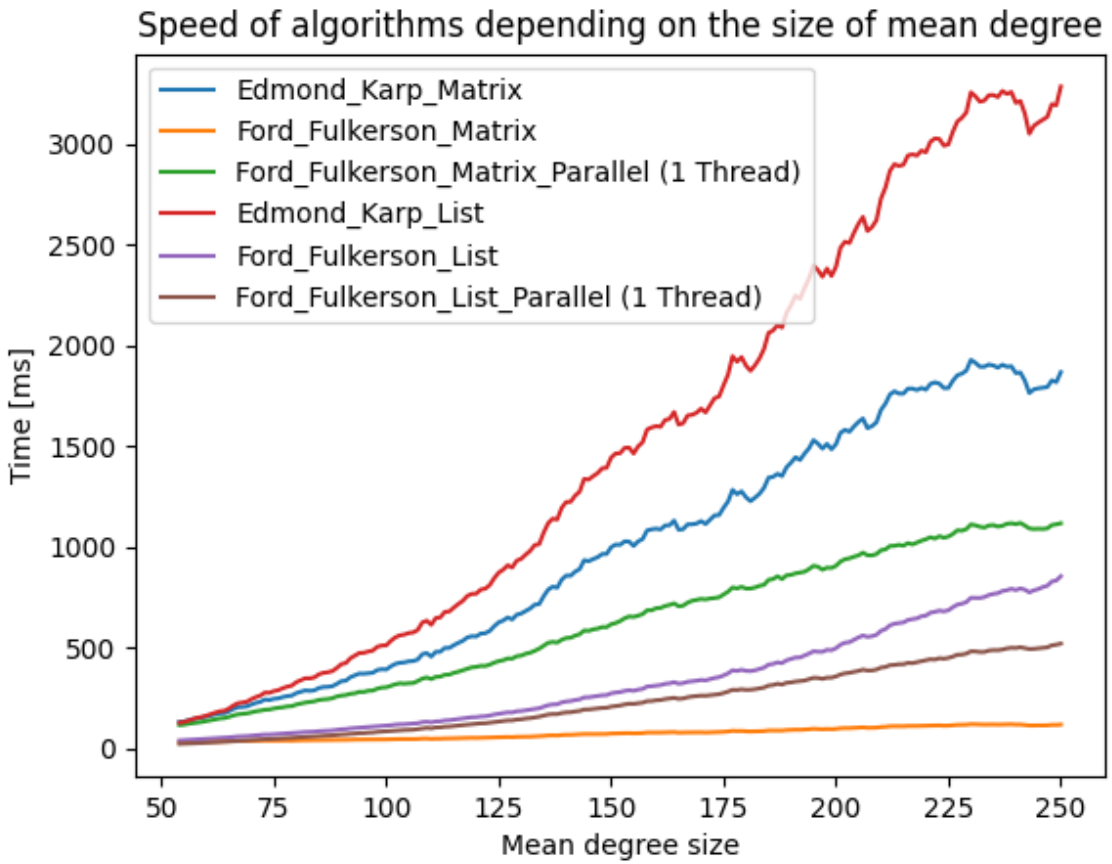
Pro důkladné vyhodnocení jednotlivých algoritmů byl napsán skript pro vygenerování velkého počtu náhodných sítí.

- Generování sítí o různé velikosti
- Generování sítí o ruzné hustotě

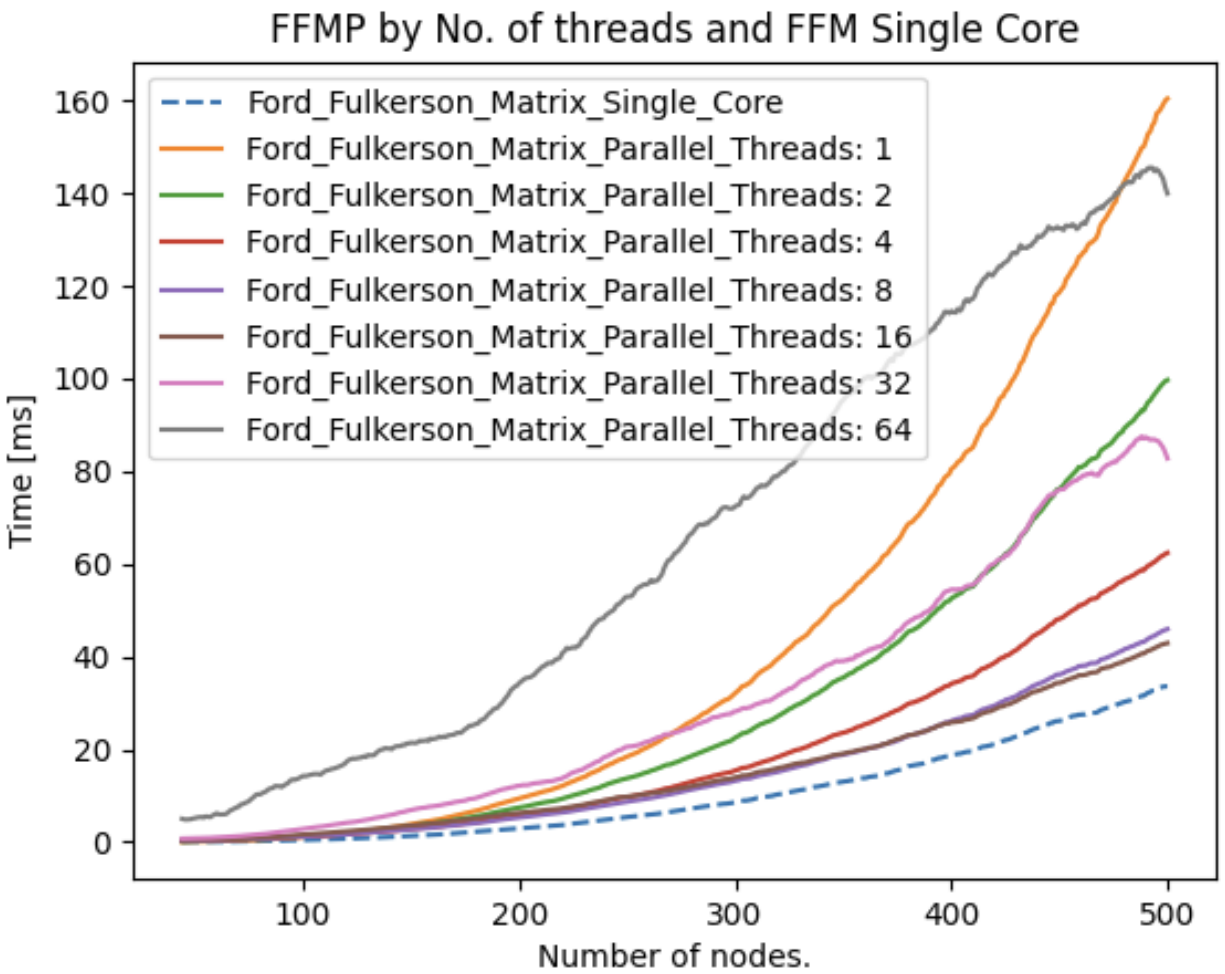
Vyhodnocení sekvenčních algoritmů - 1



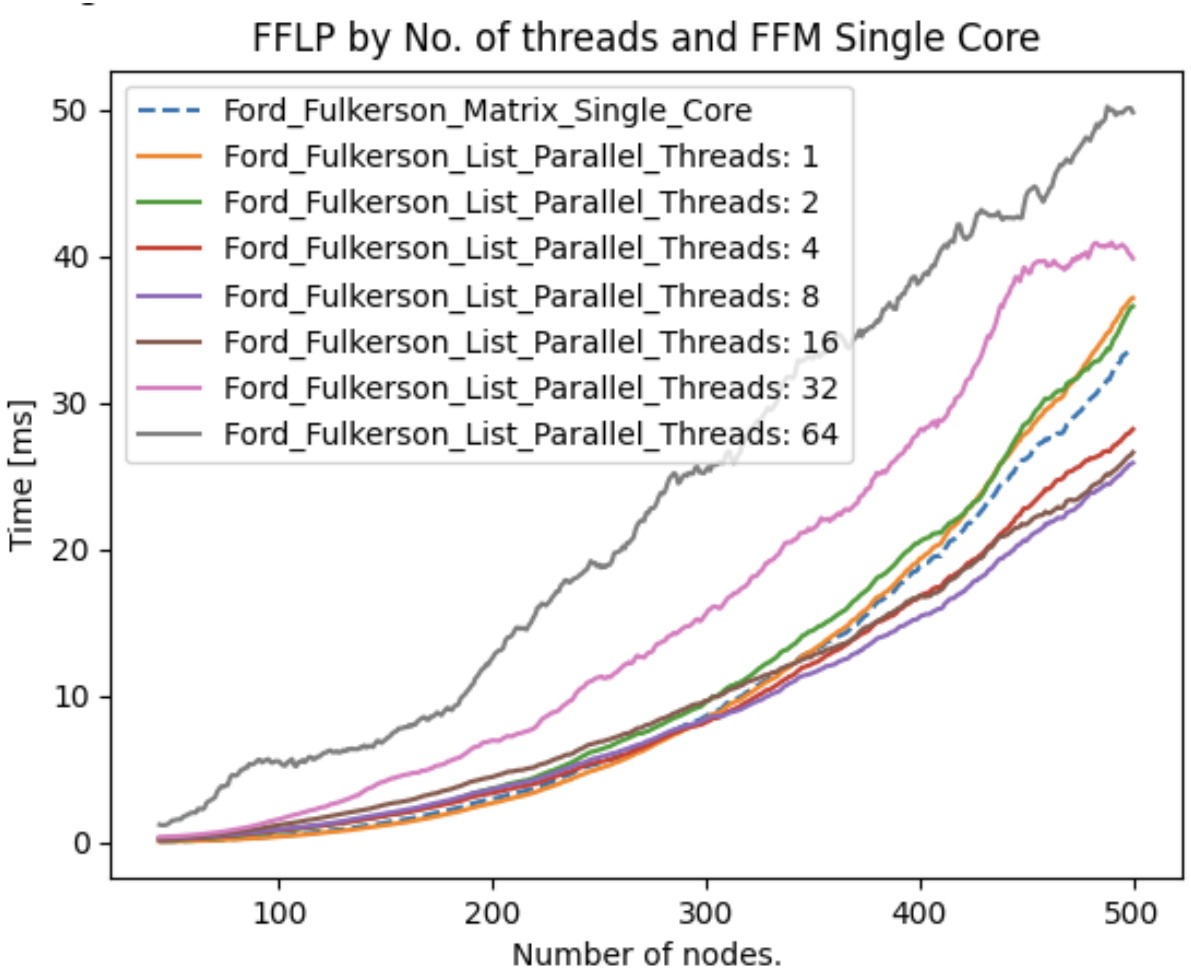
Vyhodnocení sekvenčních algoritmů - 2



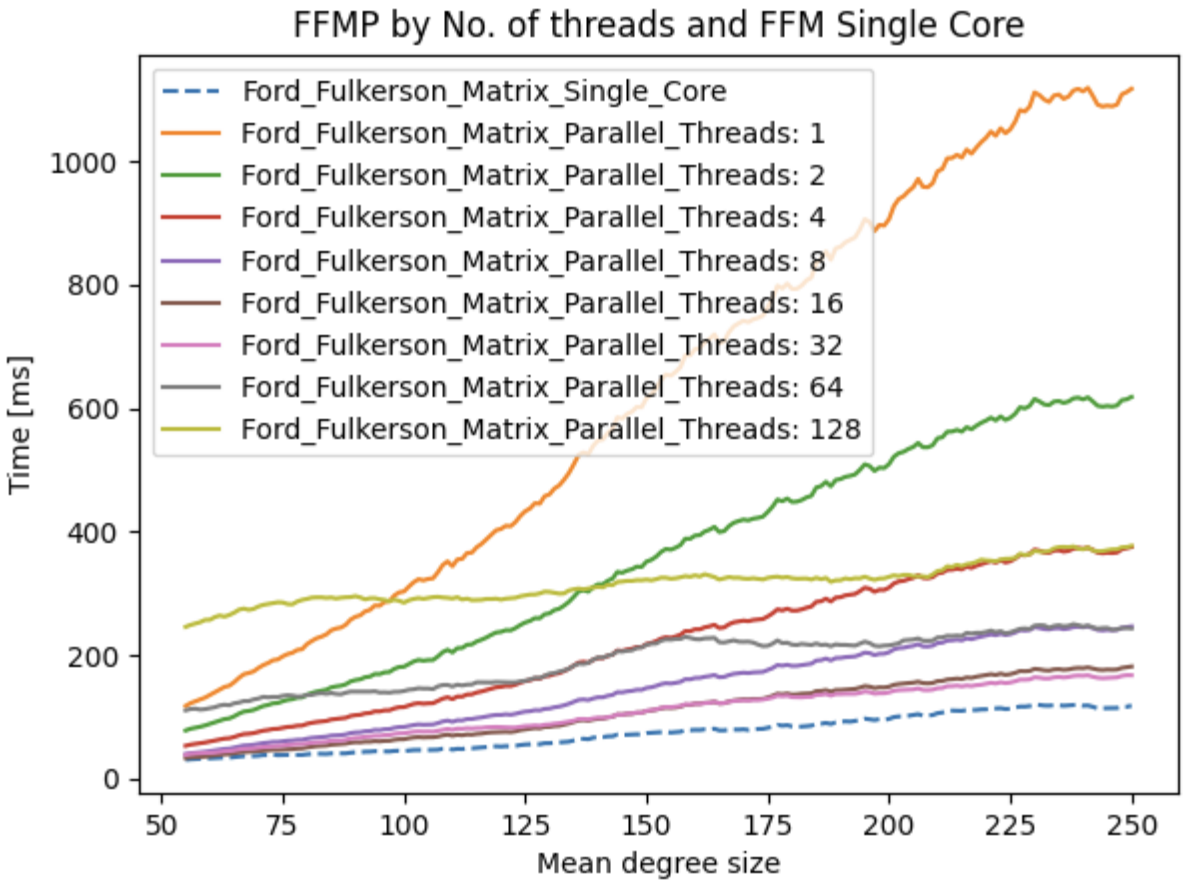
Vyhodnocení paralelních algoritmů - 1



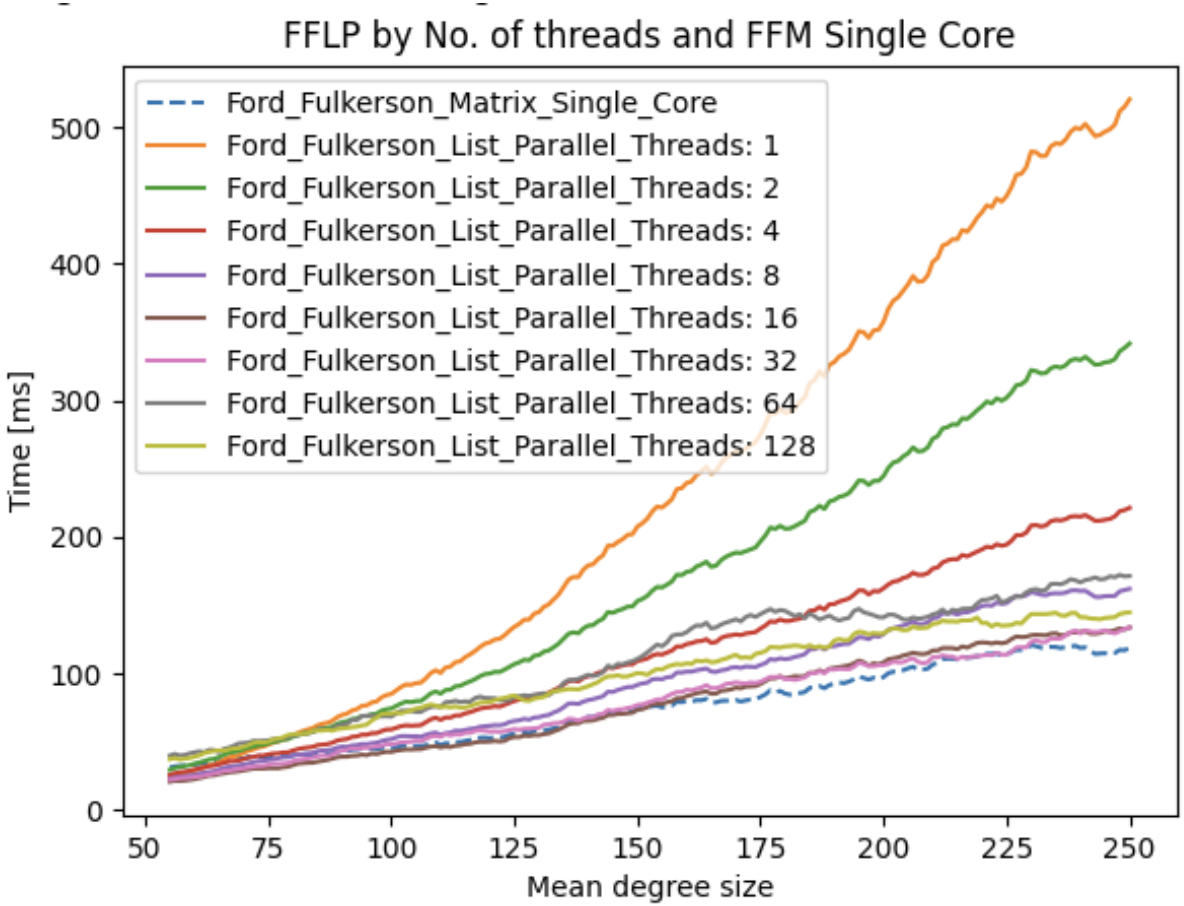
Vyhodnocení paralelních algoritmů - 2



Vyhodnocení paralelních algoritmů - 3



Vyhodnocení paralelních algoritmů - 4



Push Relabel algoritmus

- Časová zložitosť: $\mathcal{O}(V^2E) \rightarrow \mathcal{O}(V^3)$
- Počas behu algoritmu nemusí byť dodržané pravidlo: $u \in V - \{s, t\}, \sum_{v \in V} f(u, v) = 0$
- $x_f : V \rightarrow \mathbb{R}$
- $d : V \rightarrow \mathbb{N}$

Appendix: Pseudocode and Timings

Listing 1.1. Pseudocode implementation of *prsn*.

```
1  procedure PRSyncNondet()
2      parallel foreach v ∈ V
3          d(v) := 0
4          e(v) := 0
5          v.addedExcess := 0
6          v.isDiscovered := 0
7      d(s) := n
8      parallel foreach (v, w) ∈ E
9          f(v,w) := f(w,v) := 0
10     // initially saturate all source−adjacent edges
11     parallel foreach (s, v) ∈ E
12         f(s,v) := c(s,v)
13         f(v,s) := −c(s,v)
14         e(v) := c(s,v)
15     workSinceLastGR := ∞
16     while true:
17         // from hi_pr: freq = 0.5, α = 6
18         if freq · workSinceLastGR > α · n + m:
19             workSinceLastGR := 0
20             GlobalRelabel() // see Listing 1.2
21             // parallel array comprehension using map/filter
22             workingSet = [ v | v ← workingSet, d(v) < n ]
23
24         if workingSet = ∅ break
25
26         parallel foreach v ∈ workingSet
27             v.discoveredVertices := []
28             d'(v) := d(v)
29             e := e(v) // local copy
30             v.work := 0
31             while e > 0
32                 newLabel := n
33                 skipped := 0
34                 [redacted]
35                 if e = 0 // vertex is already discharged completely
36                     break
37                 admissible := (d'(v) = d(w) + 1)
38                 // is the edge shared between two active vertices?
39                 if e(w)
40                     win := d(v) = d(w) + 1
41                                 or d(v) < d(w) − 1
42                                 or (d(v) = d(w) and v < w)
43                     if admissible and not win
44                         skipped := 1
45                         continue // skip to next residual edge
46                 if admissible and c_f(v, w) > 0 // edge is admissible
47                     Δ := min(c_f(v, w), e(v))
48                     // the following three updates do not need to be atomic
49                     f(v,w) += Δ
50                     f(w,v) −= Δ
51                     e −= Δ
52                     // atomic fetch−and−add
53                     w.addedExcess += Δ
54                     if w ≠ t and [redacted]
55                         v.discoveredVertices.pushBack(w)
56                 if c_f(v, w) > 0 and d(w) ≥ d'(v)
57                     newLabel := min(newLabel, d(w) + 1)
58             if e = 0 or skipped
59                 break
60             d'(v) := newLabel // relabel
61             v.work += v.outDegree + β // from hi_pr: β = 12
62             if d'(v) = n
63                 break
64         v.addedExcess := e − e(v)
```

```
65             if e'(v) and [redacted]
66                 v.discoveredVertices.pushBack(v)
67
68         parallel foreach v ∈ workingSet
69             d(v) := d'(v)
70             e(v) += v.addedExcess
71             v.addedExcess := 0
72             v.isDiscovered := 0
73
74         workSinceLastGR += Sum([ v.work | v ← workingSet ])
75         workingSet := Concat([ v.discoveredVertices | v ← workingSet, d(v) < n ])
76
77         parallel foreach v ∈ workingSet
78             e(v) += v.addedExcess
79             v.addedExcess := 0
80             v.isDiscovered := 0
```

Listing 1.2. Pseudocode implementation of parallel global relabeling.

```
1  procedure GlobalRelabel()
2      parallel foreach v ∈ V
3          d(v) := n
4      d(t) := 0
5      Q := [t]
6      while Q ≠ ∅
7          parallel foreach v ∈ Q
8              v.discoveredVertices := []
9              [redacted]
10             [redacted]
11             [redacted]
12                 d(w) := d(v) + 1
13                 v.discoveredVertices.pushBack(w)
14         // concatenation implemented using parallel prefix sums
15         Q := Concat([ v.discoveredVertices | v ← Q ])
```