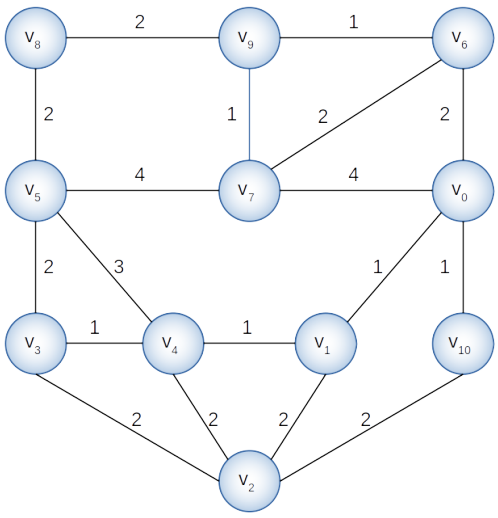
Efficiently updating Customizable Contraction Hierarchies

1 Intro

This is a seminar on "Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees" [4]. The paper is about finding the shortest path in road networks. By *shortest* it is meant, the path that requires the less time to get from source s to a target t. The route network is modeled as a directed graph $G(V, E)$ where each street crossing represents a vertex $v \epsilon V$ and each road between crossings represents an edge $e \epsilon E$. The most basic and solid method to find shortest paths between vertices in a graph is Dijkstra's algorithm [2]. This algorithm is proofed to always return the correct shortest path but it is not fast enough for just in time route planning on large road networks as we know it from services like Google Maps.
There are many different approaches that try to speed up shortest path queries by precomputing any different kind of index structure before doing the shortest path query. The index structure discussed in here [4] is CH (Contraction Hierarchies)[3] with some extension. This extension is CCH (Customizable Contraction Hierarchies) [1]. Although the authors of [4] never mention the term CCH their approaches builds the same index structure. This is a pity, as for this part they kind of reinvent the wheel.
The difference lies in updating the CCH index structure. For updating the CCH, the authors of [4] will use yet another index structure called *SS-Graph* that helps to exactly identify the shortcuts that have to be updated, after some edge weight has changed. In todays implementation, the whole index structure is recomputed periodically. This is an valid approach as it is fast enough to stay accurate for route planning in road networks.
In [4] the authors would like to find a way that make it possible to handle streaming updates. Therefore it is necessary to handle single weight decreases and increases. To do so another index structure the *SS-Graph* is introduced. This index structure is a helper structure to identify the shortcuts that have to be updated in the CCH.
The disadvantage of this *SS-Graph* is that the overall space consumption rises. This can be a deal breaker for large networks. Finally they introduce a way to create the necessary *SS-Graph* on the fly. Which is only a part of the whole structure. Sadly the exact way, how, is missing in the paper [4].

2 Shortcut Supporting Graph

Figure 1: Base Graph G



This section is about the *Shortcut Supporting Graph* introduced in [4]. The way how the shortcut index is created is omitted because it is a Customizable Contraction Hierarchies shortcut index

and therefore not of further interest. Although the algorithm to get there is described differently the result is a shortcut index that encodes every simple path. This is the same in CCH.
Figure 1 represents the sample base graph which is used as an example graph throughout this whole work. It is the same as used in [4], but redrawn.

Figure 2: Shortcut Index Graph G'

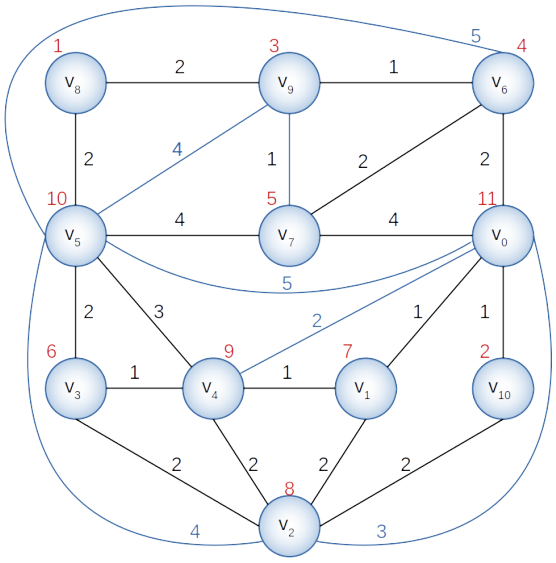


Figure 2 represents the *Shortcut Index G'* already materialized. The red numbers located just outside the vertices represent the contraction order. The numbers on the edge are the weight assigned to the edge. The inserted shortcuts are drawn in blue and have their weight already assigned, too.

Based on this *Shortcut Index G'* yet another index structure is introduced. The *Shortcut Supporting Graph G**. $G*$ is a directed graph that has two different kind of vertices. The first vertex set V_s represents all the edges that are contained in the *Shortcut Index G'*. The second vertex set are the relation type vertices V_r. A relation type vertex v_r has two in-neighbors v_{s1} and v_{s2} that and one out neighbor v_{s3}. Each vertex in V_s has three properties.

- The edge or shortcut $e = (v_u, v_v)$ in G' it represents
- The shortest path cost $\phi(e)$
- The number of relation type vertices that support this edge or shortcut and lye on the shortest path $c_\phi(e)$

Figure 3 is the *Shortcut Supporting Graph G** of *Shortcut Index G'*.

Example 1: Lets take this most upper vertex of $G*$ in figure 3. It represents the shortcut $e = (v_0, v_5)$. This shortcut is supported through four other simple paths in G' of two hops.

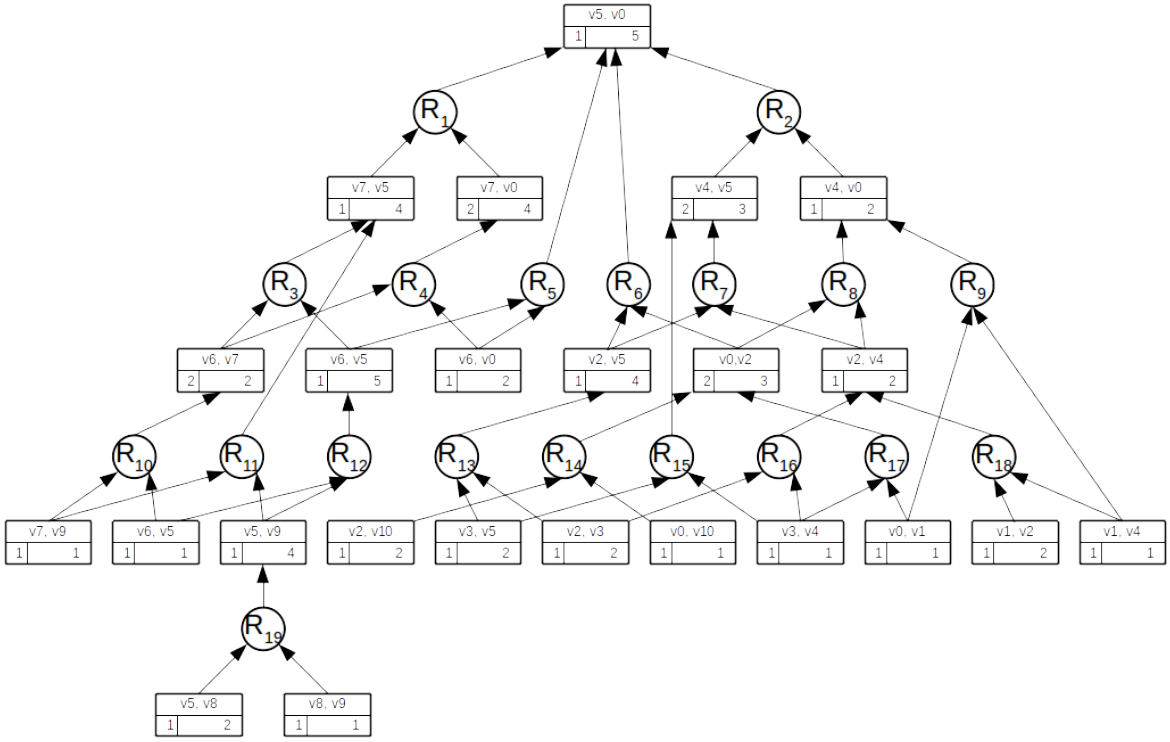
1. R_1 with the two in neighbors $v_{s1} = (v_7, v_5)$ and $v_{s2} = (v_7, v_0)$ with total path cost $\phi = \phi(v_{s1}) + \phi_{v_{s2}} = 8$
2. R_2 with the two in neighbors $v_{s1} = (v_4, v_5)$ and $v_{s2} = (v_4, v_0)$ with total path cost $\phi = \phi(v_{s1}) + \phi_{v_{s2}} = 5$
3. R_5 with the two in neighbors $v_{s1} = (v_6, v_5)$ and $v_{s2} = (v_6, v_0)$ with total path cost $\phi = \phi(v_{s1}) + \phi_{v_{s2}} = 7$
4. R_6 with the two in neighbors $v_{s1} = (v_2, v_5)$ and $v_{s2} = (v_0, v_5)$ with total path cost $\phi = \phi(v_{s1}) + \phi_{v_{s2}} = 7$

$c_\phi(e) = 1$ as only R_2 is on the shortest path that has length five.

2.1 Space Consumption

The space consumption of $G*$ is:
$\mathcal{O}(|E(G')| + \sum_{v \epsilon V(G')}(|Nbr(v)| * (|Nbr(v)| - 1)/2))$
where $Nbr(v) = u | u \epsilon nbr(v) \wedge \gamma(u) > \gamma(v)$ and $\gamma(u)$ is the rank of vertex u. This is a lot more than G' and can be a problem in practice. Due to the authors there exist an algorithm in their *long version of the paper* that does not need to materialize the $G*$. Sadly this link is broken.

Figure 3: Shortcut Support Index $G*$



3 Reaction on Weight Changes

Here we will have a look what happens on a weight decrease or increase. In CCH for the whole shortcut index is updated in a fixed time interval. We want to do streaming updates which means react to a single weight change. In classic CCH you usually have to recontract at least all vertices with a higher rank than the once that form the endpoint of the updated edge. Having the *Shortcut Support Index* $G*$ it is possible to determine which edges have to be updated. In general this could be the same amount of edges but in real applications this is very unlikely.

3.1 Weight Decrease

In this case a single edge weight decrease. The algorithm in figure 4 show what to do in this case. The update of the support counter c_ϕ is omitted as figure 4 is copied from [4] and it was omitted there, too. The support counter c_ϕ isn't needed for the decrease case to work but it is needed in the decrease case to later on work together with the increase case. Fixing this is out of the scope for this work. Therefore the algorithm is presented as is in figure 4.

What this algorithm basically does is, when a weight decreases happens it looks for shortcuts that are build upon this edge. If the weight of this shortcut doesn't depend on the edge it stops. If the weight of

Figure 4: Shortcut Index Graph G'

Algorithm 2 DCH$_{scs}$-WDec (G^*, G, e, k)

```
1: PriorityQueue Q ← ∅; φ(e, G) ← k;
2: if φ(v_e, G*) > k then
3:     φ(v_e, G*) ← k; Q.push(v_e);
4: while Q ≠ ∅ do
5:     v_s ← Q.pop();
6:     for each v_r ∈ nbr+(v_s) do
7:         v'_s ← the other in-neighbor of v_r except v_s;
8:         v''_s ← nbr+(v_r);
9:         if φ(v''_s, G*) > φ(v_s, G*) + φ(v'_s, G*) then
10:            φ(v''_s, G*) ← φ(v_s, G*) + φ(v'_s, G*);
11:            if v''_s ∉ Q then
12:                Q.push(v''_s);
```

this shortcut depends on the edge, meaning the edge was lying on the shortest path, it updates the weight of the shortcut. This happens recursively.

This guarantees that only edges that are one hop dependent on a changed edge are explored. Therefore the time complexity of this algorithm is $\mathcal{O}(|\Delta| * (log|\Delta| + deg'_{max}) + 1)$. $|\Delta|$ is the number of shortcuts that have to be updated and deg'_{max}. In the worst case $|\Delta|$ are all edges and shortcuts that have a vertex involved with a higher rank than in the changed edge e. In practice this is very unlikely. In section 4 it is stated that the it actually outperforms CCH.

Because it is ineffective and difficult to describe this algorithm in words there is a step by step slide show attached to this paper, that shows how it behaves.

3.2 Weight Increase

In this case a single edge weight increase. The algorithm in figure 5 show what to do in this case. It basically works in the same manner as the decrease case.

As for the decrease case it guarantees that only edges that are one hop dependent on a changed edge are explored. Therefore the time complexity of this algorithm is $\mathcal{O}(|\Delta| * (log|\Delta| + deg'_{max}) + 1)$. $|\Delta|$ is the number of shortcuts that have to be updated and deg'_{max}. [4]

Because it is ineffective and difficult to describe this algorithm in words there is a step by step slide show attached to this paper, that shows how it behaves.

Figure 5: Shortcut Index Graph G'

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1: PriorityQueue Q ← ∅;
2: if φ(v_e, G*) = φ(e, G) then
3:     c_φ(v_e) ← c_φ(v_e) − 1;
4: φ(e, G) ← k;
5: if c_φ(v_e) < 1 then
6:     Q.push(v_e);
7: while Q ≠ ∅ do
8:     v_s ← Q.pop(); updateWeight(G*, v'_s);
9:     for each v_r ∈ nbr+(v_s) do
10:        v'_s ← nbr+(v_r);
11:        if c_φ(v'_s) < 1 then
12:            if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

4 Experimental Results

The decrease and increase algorithm have been experimentally evaluated against CCH and CRP. We will only regard the difference between CCH and the algorithms in figure 5 and figure 4 as well as their equivalent that doesn't materialize the *Shortcut Support Index G**. [4]

4.1 Dataset

The dataset that was used for the experiments comes from here. These are real world public road networks in the US.

4.2 Space Consumption

As already mentioned in 2.1 the *Shortcut Support Index* $G*$ consumes a lot more memory than the *Shortcut Index G'*. For the dataset they tried out in [4] it used between 12.7 and 34.6 times more memory that G', depending on the dataset that was used. This can be avoided when using the on the fly algorithm that, doesn't materialize $G*$. Sadly it wasn't provided in the paper and neither was the code with which they made their experiments.

4.3 Weight Changes Performance

The authors randomly picked 1000 edges, changed their weight and updated the *Shortcut Index G'*. For the decrease case the proposed algorithms in figure 4 and figure 5 outperformed CCH by order of magnitude 2 3. The not provided algorithms that work without materializing the *Shortcut Support Index* $G*$ claims to be as fast as the materialized one.

5 Conclusion

The generally idea of providing a streaming algorithm for updating a CCH shortcut index. The algorithms in [4] are promising, but the whole evaluation should be redone. First of all it should be categorized as *CCH* and not *DCH* – a term that wasn't even mentioned in the paper they based their research on [3]. Secondly they should provide the algorithms for the non materialized manner or *the long version of the paper [4]*. Finally, why isn't there any link to the source code which they used to test their idea?

In my opinion the idea itself is valid but the actual paper would need some additional work in terms of properly categorizing the topic, and also some transparency regarding the implementation and the experimentally evaluation.

References

[1] Julian Dibbelt, Ben Strasser, and Dorothea Wagner. Erratum: Customizable contraction hierarchies. In *Experimental Algorithms*, pages E1–E1. Springer International Publishing, 2014.

[2] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, dec 1959.

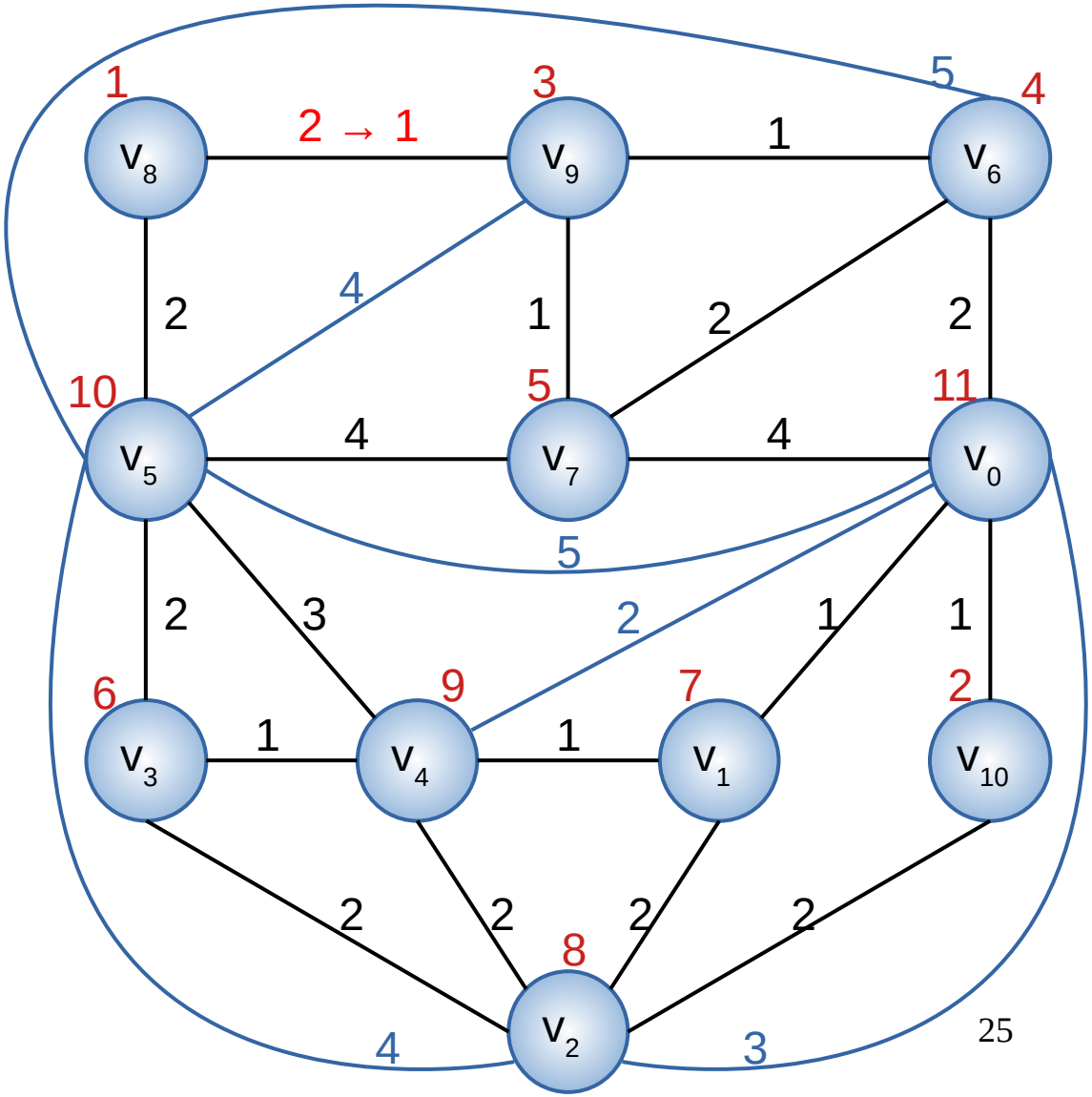
[3] Robert Geisberger, Peter Sanders, Dominik Schultes, and Christian Vetter. Exact routing in large road networks using contraction hierarchies. *Transportation Science*, 46(3):388–404, aug 2012.

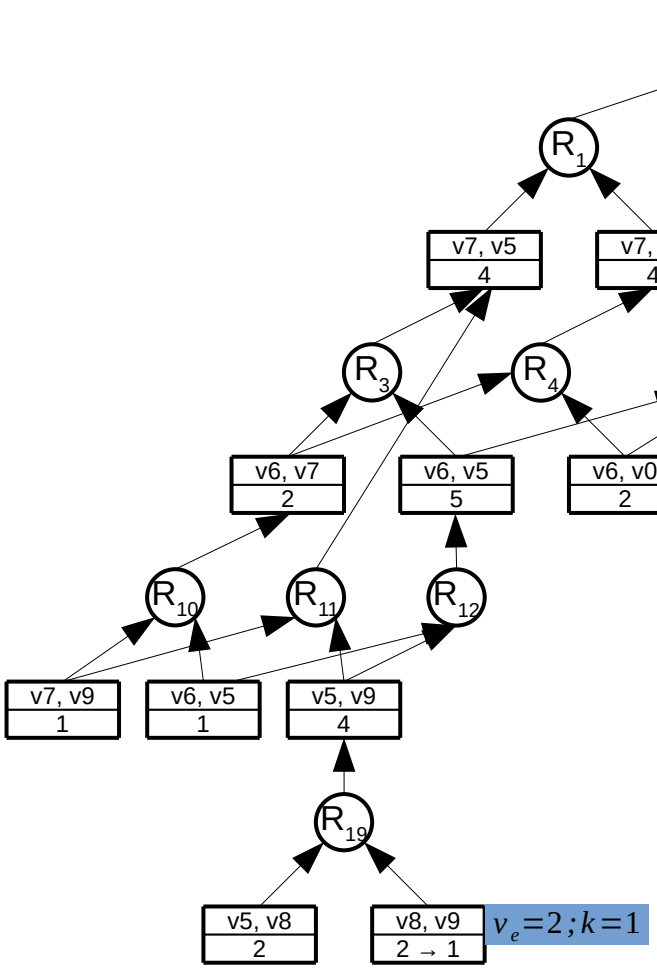
[4] Dian Ouyang, Long Yuan, Lu Qin, Lijun Chang, Ying Zhang, and Xuemin Lin. Efficient shortest path index maintenance on dynamic road networks with theoretical guarantees. *Proceedings of the VLDB Endowment*, 13(5):602–615, jan 2020.

List of Figures

Weight Decrease

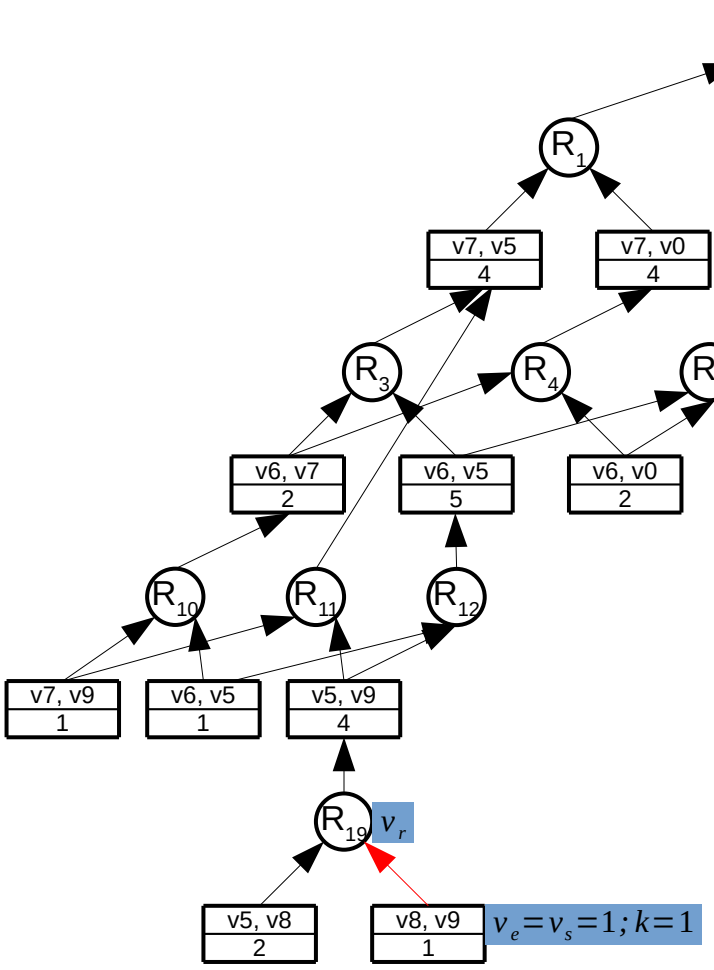
- Weight decrease (v_8 v_9) = 2 → 1
- Implies further weight changes
- Let's use G* to find out which





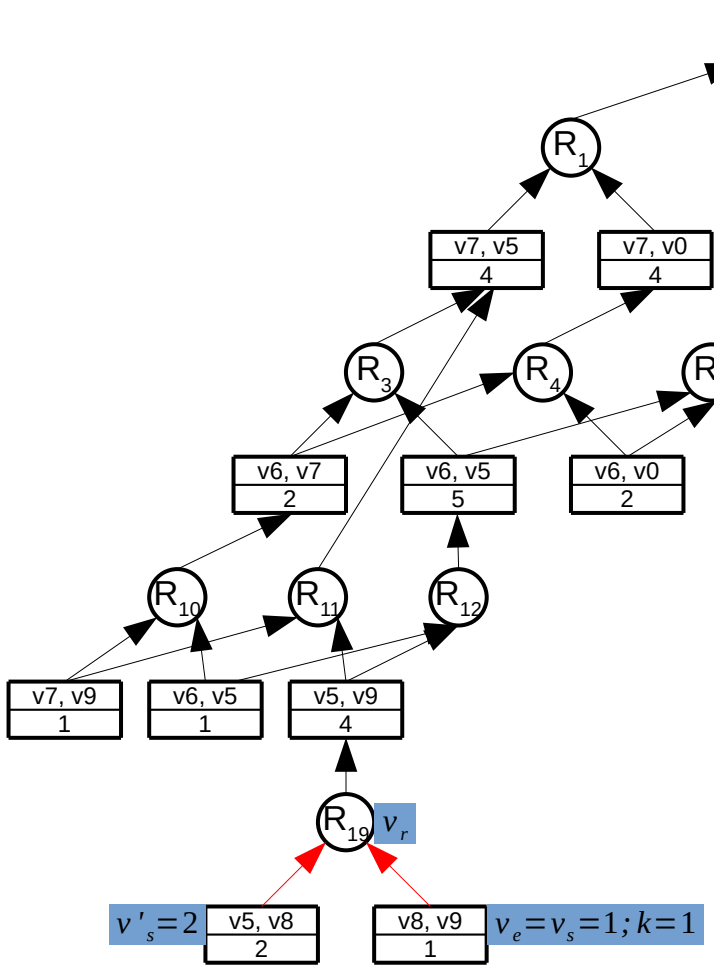
Algorithm 2 DCH$_{\text{scs}}$-WDec (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅; φ(e, G) ← k;
2:  if φ(v_e, G*) > k then
3:      φ(v_e, G*) ← k; Q.push(v_e);
4:  while Q ≠ ∅ do
5:      v_s ← Q.pop();
6:      for each v_r ∈ nbr⁺(v_s) do
7:          v'_s ← the other in-neighbor of v_r except v_s;
8:          v''_s ← nbr⁺(v_r);
9:          if φ(v''_s, G*) > φ(v_s, G*) + φ(v'_s, G*) then
10:             φ(v''_s, G*) ← φ(v_s, G*) + φ(v'_s, G*);
11:             if v''_s ∉ Q then
12:                 Q.push(v''_s);
```



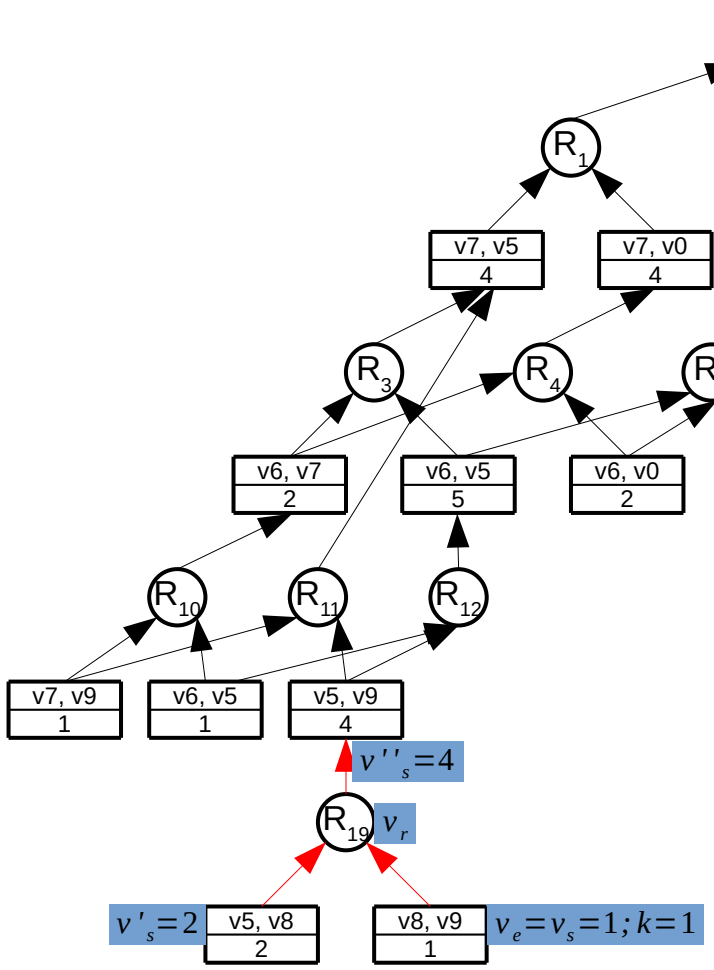
Algorithm 2 $\mathsf{DCH}_{\mathsf{scs}}$-WDec (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅; φ(e, G) ← k;
2:  if φ(v_e, G*) > k then
3:      φ(v_e, G*) ← k; Q.push(v_e);
4:  while Q ≠ ∅ do
5:      v_s ← Q.pop();
6:      for each v_r ∈ nbr⁺(v_s) do
7:          v'_s ← the other in-neighbor of v_r except v_s;
8:          v''_s ← nbr⁺(v_r);
9:          if φ(v''_s, G*) > φ(v_s, G*) + φ(v'_s, G*) then
10:             φ(v''_s, G*) ← φ(v_s, G*) + φ(v'_s, G*);
11:             if v''_s ∉ Q then
12:                 Q.push(v''_s);
```



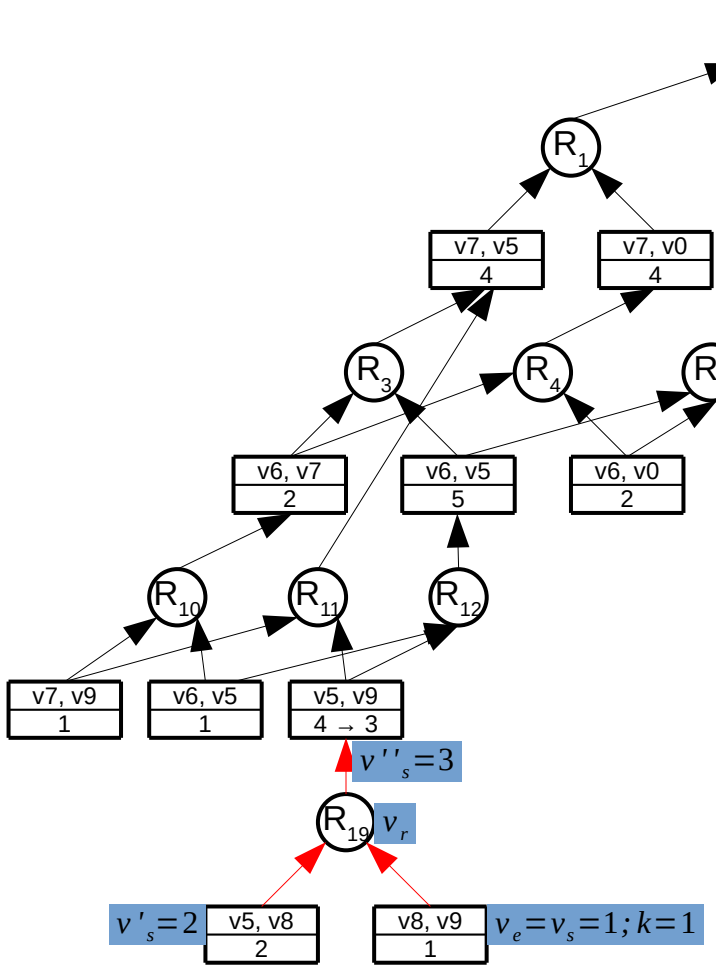
Algorithm 2 $\mathsf{DCH}_{\mathsf{scs}}$-WDec (G^*, G, e, k)

PriorityQueue $Q \leftarrow \emptyset$; $\phi(e, G) \leftarrow k$;
if $\phi(v_e, G^*) > k$ **then**
  $\phi(v_e, G^*) \leftarrow k$; Q.push(v_e);
while $Q \neq \emptyset$ **do**
  $v_s \leftarrow Q$.pop();
  for each $v_r \in \mathsf{nbr}^+(v_s)$ **do**
    $v'_s \leftarrow$ the other in-neighbor of v_r except v_s;
    $v''_s \leftarrow \mathsf{nbr}^+(v_r)$;
    if $\phi(v''_s, G^*) > \phi(v_s, G^*) + \phi(v'_s, G^*)$ **then**
      $\phi(v''_s, G^*) \leftarrow \phi(v_s, G^*) + \phi(v'_s, G^*)$;
      if $v''_s \notin Q$ **then**
        Q.push(v''_s);



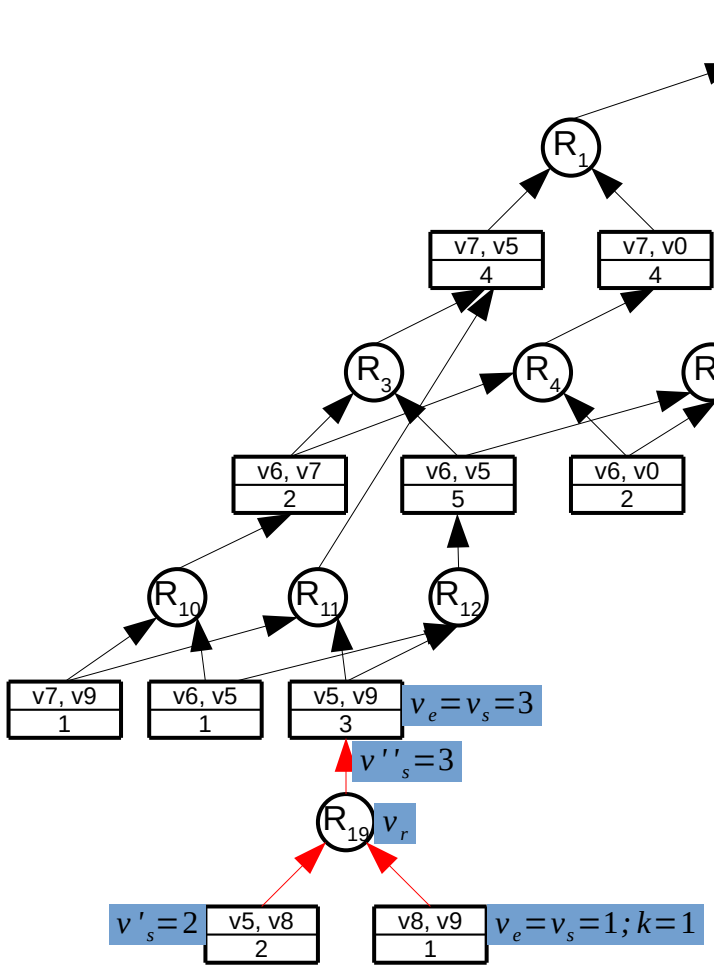
Algorithm 2 $\mathsf{DCH}_{\mathsf{scs}}$-WDec (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅; φ(e, G) ← k;
2:  if φ(v_e, G*) > k then
3:      φ(v_e, G*) ← k; Q.push(v_e);
4:  while Q ≠ ∅ do
5:      v_s ← Q.pop();
6:      for each v_r ∈ nbr+(v_s) do
7:          v'_s ← the other in-neighbor of v_r except v_s;
8:          v''_s ← nbr+(v_r);
9:          if φ(v''_s, G*) > φ(v_s, G*) + φ(v'_s, G*) then
10:             φ(v''_s, G*) ← φ(v_s, G*) + φ(v'_s, G*);
11:             if v''_s ∉ Q then
12:                 Q.push(v''_s);
```



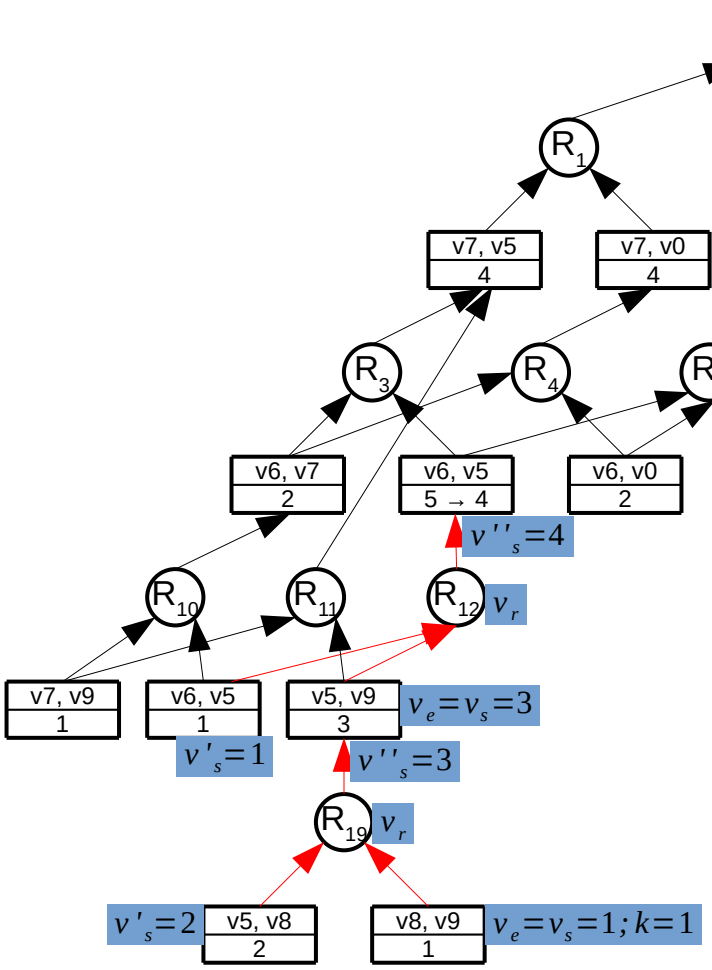
Algorithm 2 $\mathsf{DCH}_{\mathsf{scs}}$-WDec (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅; φ(e, G) ← k;
2:  if φ(v_e, G*) > k then
3:      φ(v_e, G*) ← k; Q.push(v_e);
4:  while Q ≠ ∅ do
5:      v_s ← Q.pop();
6:      for each v_r ∈ nbr⁺(v_s) do
7:          v'_s ← the other in-neighbor of v_r except v_s;
8:          v''_s ← nbr⁺(v_r);
9:          if φ(v''_s, G*) > φ(v_s, G*) + φ(v'_s, G*) then
10:             φ(v''_s, G*) ← φ(v_s, G*) + φ(v'_s, G*);
11:             if v''_s ∉ Q then
12:                 Q.push(v''_s);
```



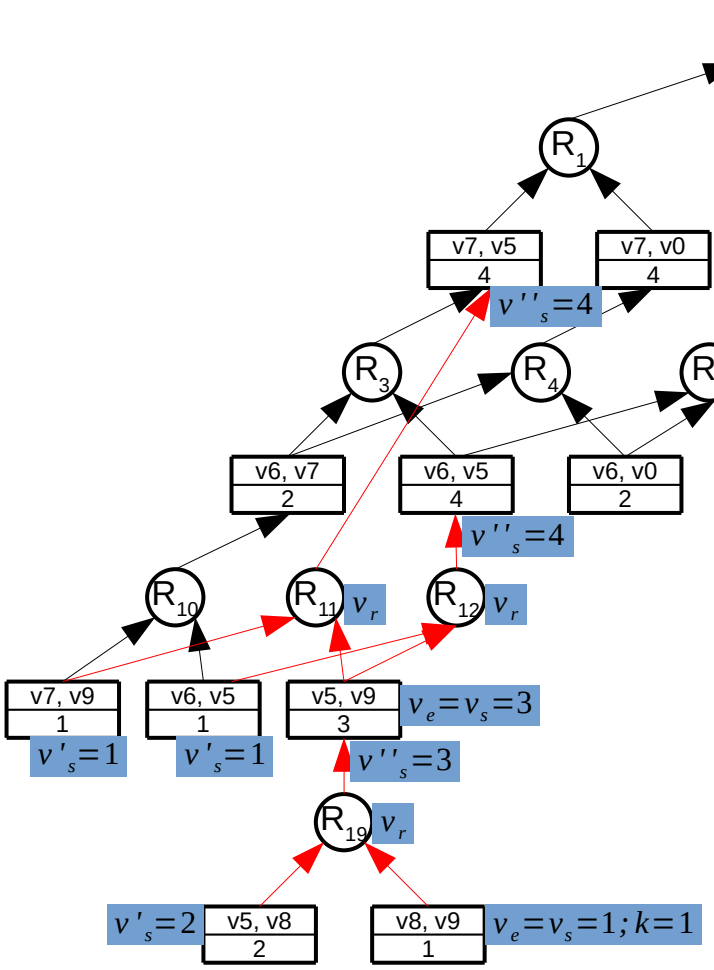
Algorithm 2 DCH$_{\text{scs}}$-WDec (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅; φ(e, G) ← k;
2:  if φ(v_e, G*) > k then
3:      φ(v_e, G*) ← k; Q.push(v_e);
4:  while Q ≠ ∅ do
5:      v_s ← Q.pop();
6:      for each v_r ∈ nbr+(v_s) do
7:          v'_s ← the other in-neighbor of v_r except v_s;
8:          v''_s ← nbr+(v_r);
9:          if φ(v''_s, G*) > φ(v_s, G*) + φ(v'_s, G*) then
10:             φ(v''_s, G*) ← φ(v_s, G*) + φ(v'_s, G*);
11:             if v''_s ∉ Q then
12:                 Q.push(v''_s);
```



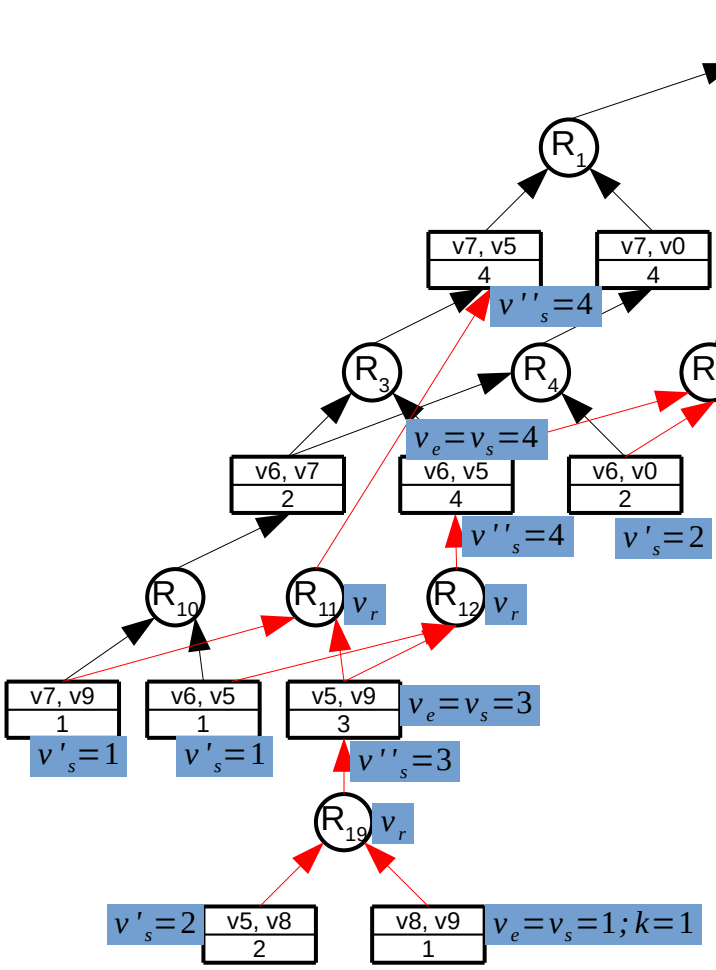
Algorithm 2 DCH$_{scs}$-WDec (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅; φ(e, G) ← k;
2:  if φ(v_e, G*) > k then
3:      φ(v_e, G*) ← k; Q.push(v_e);
4:  while Q ≠ ∅ do
5:      v_s ← Q.pop();
6:      for each v_r ∈ nbr+(v_s) do
7:          v'_s ← the other in-neighbor of v_r except v_s;
8:          v''_s ← nbr+(v_r);
9:          if φ(v''_s, G*) > φ(v_s, G*) + φ(v'_s, G*) then
10:             φ(v''_s, G*) ← φ(v_s, G*) + φ(v'_s, G*);
11:             if v''_s ∉ Q then
12:                 Q.push(v''_s);
```



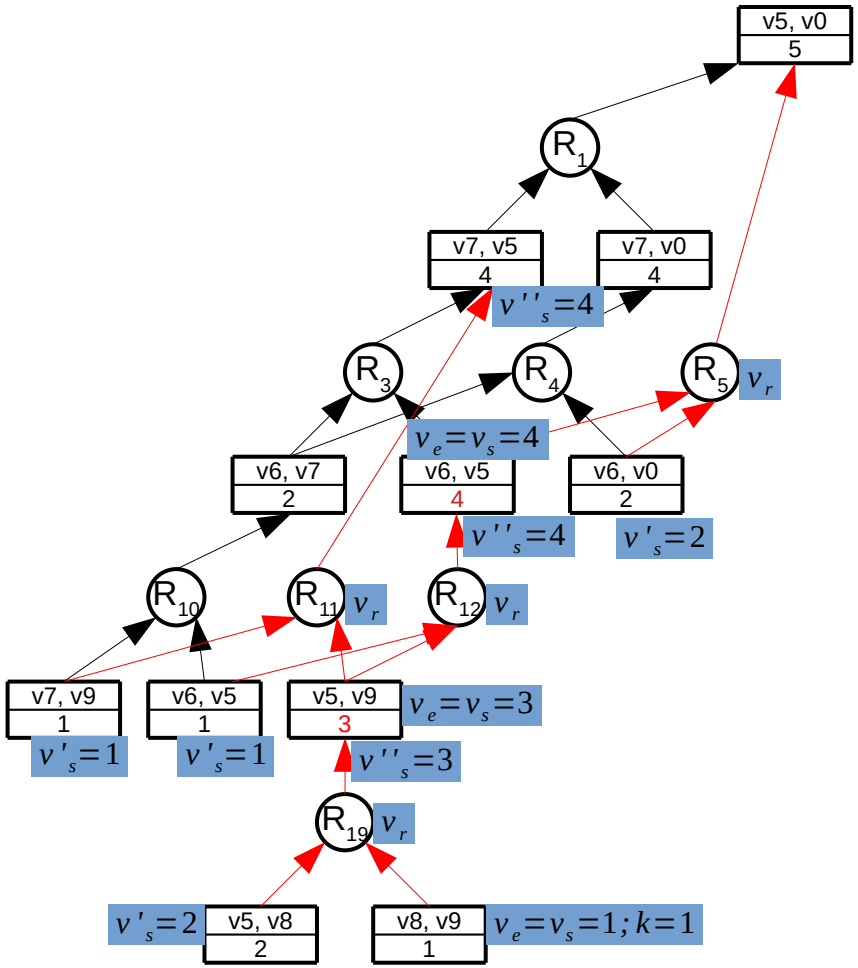
Algorithm 2 DCH$_{\text{scs}}$-WDec (G^*, G, e, k)

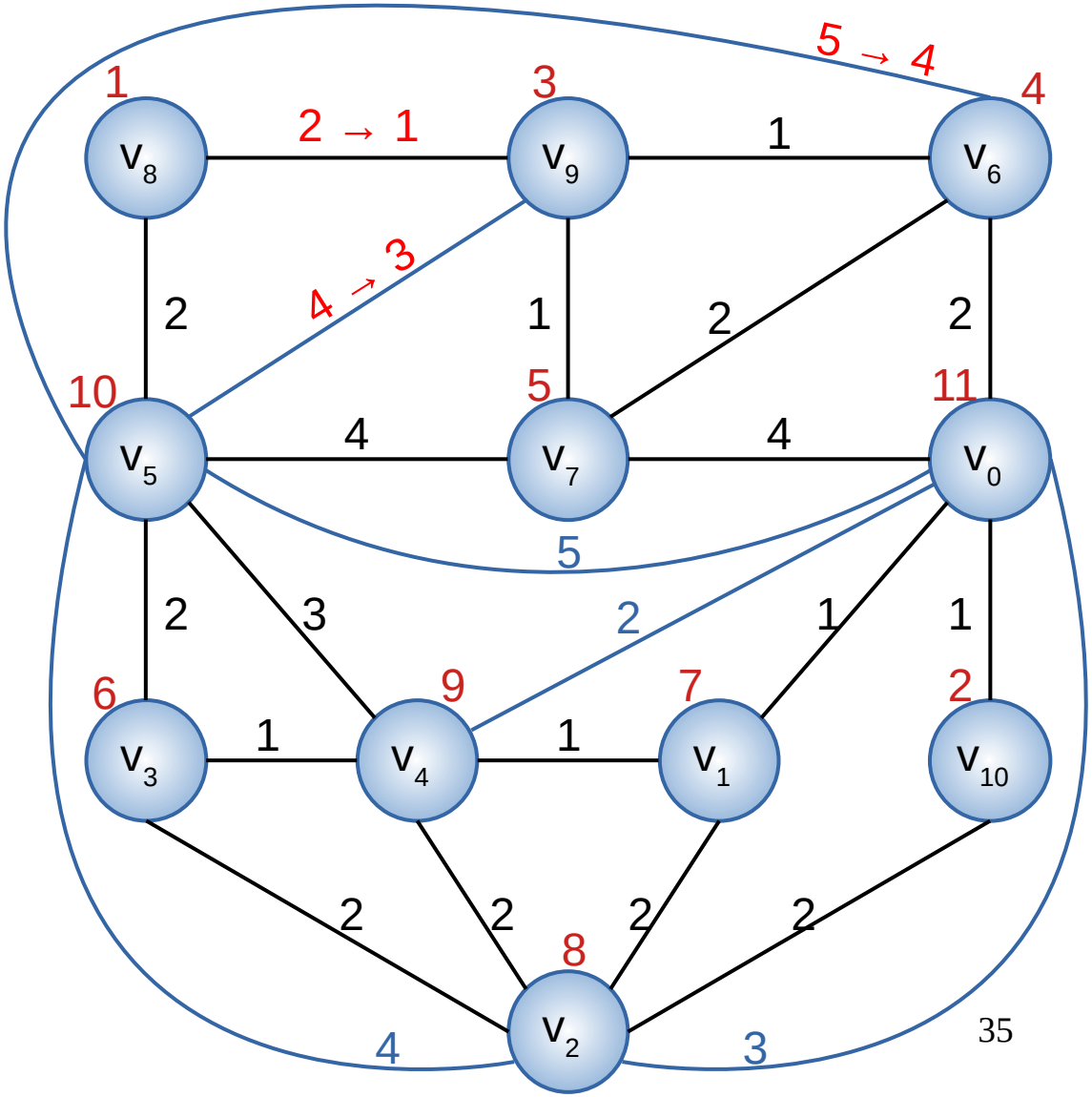
1: PriorityQueue $Q \leftarrow \emptyset$; $\phi(e, G) \leftarrow k$;
2: **if** $\phi(v_e, G^*) > k$ **then**
3: $\quad \phi(v_e, G^*) \leftarrow k$; Q.push(v_e);
4: **while** $Q \neq \emptyset$ **do**
5: $\quad v_s \leftarrow Q$.pop();
6: $\quad$ **for each** $v_r \in \text{nbr}^+(v_s)$ **do**
7: $\quad\quad v'_s \leftarrow$ the other in-neighbor of v_r except v_s;
8: $\quad\quad v''_s \leftarrow \text{nbr}^+(v_r)$;
9: $\quad\quad$ **if** $\phi(v''_s, G^*) > \phi(v_s, G^*) + \phi(v'_s, G^*)$ **then**
10: $\quad\quad\quad \phi(v''_s, G^*) \leftarrow \phi(v_s, G^*) + \phi(v'_s, G^*)$;
11: $\quad\quad\quad$ **if** $v''_s \notin Q$ **then**
12: $\quad\quad\quad\quad Q$.push(v''_s);



Algorithm 2 $\mathsf{DCH}_{\mathsf{scs}}$-WDec (G^*, G, e, k)

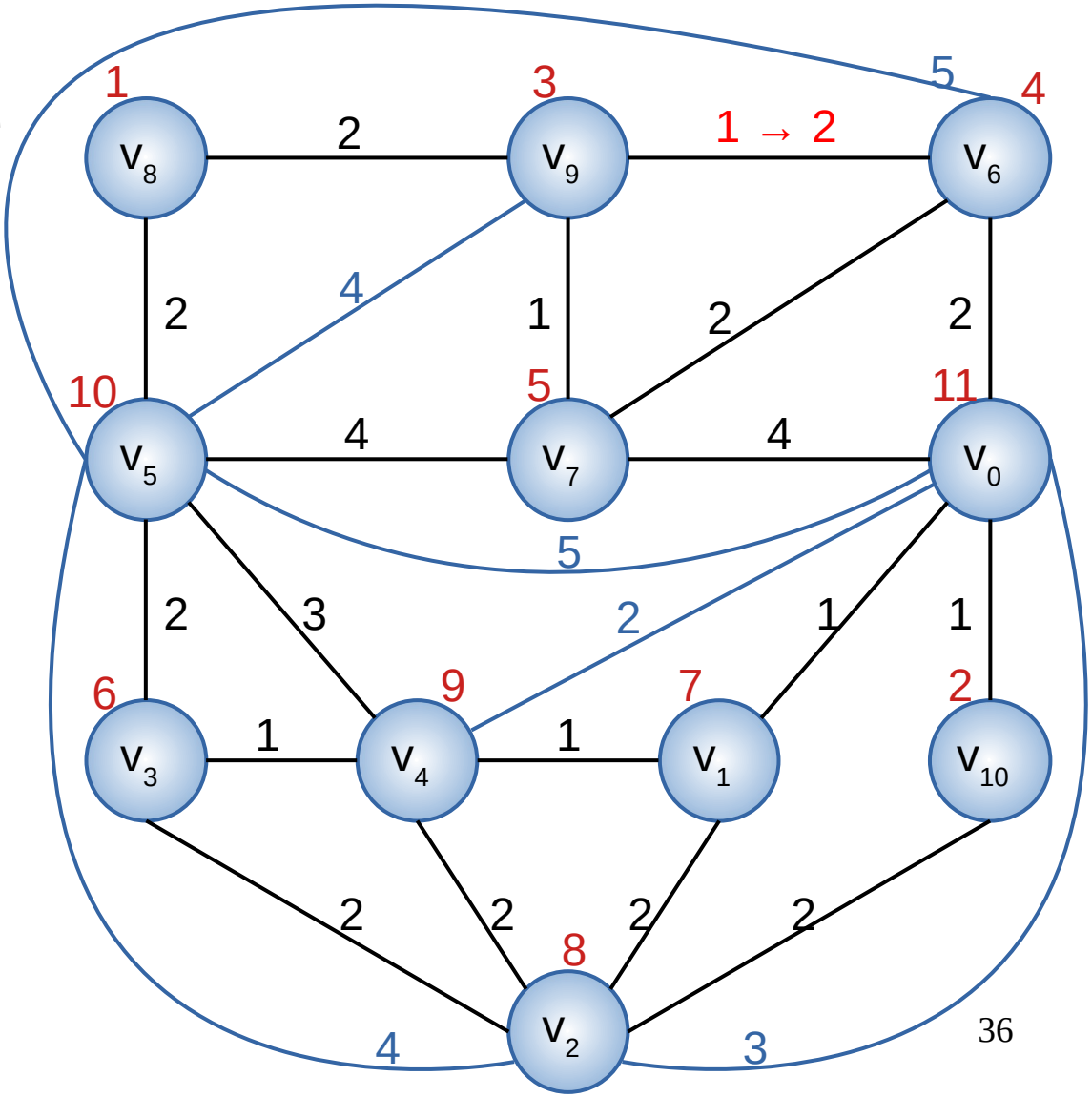
PriorityQueue $Q \leftarrow \emptyset$; $\phi(e, G) \leftarrow k$;
if $\phi(v_e, G^*) > k$ **then**
  $\phi(v_e, G^*) \leftarrow k$; Q.push(v_e);
while $Q \neq \emptyset$ **do**
  $v_s \leftarrow Q$.pop();
  for each $v_r \in \mathsf{nbr}^+(v_s)$ **do**
    $v'_s \leftarrow$ the other in-neighbor of v_r except v_s;
    $v''_s \leftarrow \mathsf{nbr}^+(v_r)$;
    if $\phi(v''_s, G^*) > \phi(v_s, G^*) + \phi(v'_s, G^*)$ **then**
      $\phi(v''_s, G^*) \leftarrow \phi(v_s, G^*) + \phi(v'_s, G^*)$;
      if $v''_s \notin Q$ **then**
        Q.push(v''_s);


v5, v0
5
R1
v7, v5
4
v7, v0
4
v''s=4
R3
R4
R5
vr
ve=vs=4
v6, v7
2
v6, v5
4
v6, v0
2
v''s=4
v's=2
R10
R11
vr
R12
vr
v7, v9
1
v6, v5
1
v5, v9
3
ve=vs=3
v's=1
v's=1
v''s=3
R19
vr
v's=2
v5, v8
2
v8, v9
1
ve=vs=1; k=1


5 → 4
1
3
4
v8
2 → 1
v9
1
v6
4 → 3
2
1
2
2
10
5
11
v5
4
v7
4
v0
5
2
3
2
1
1
6
9
7
2
v3
1
v4
1
v1
v10
2
2
2
2
8
v2
4
3

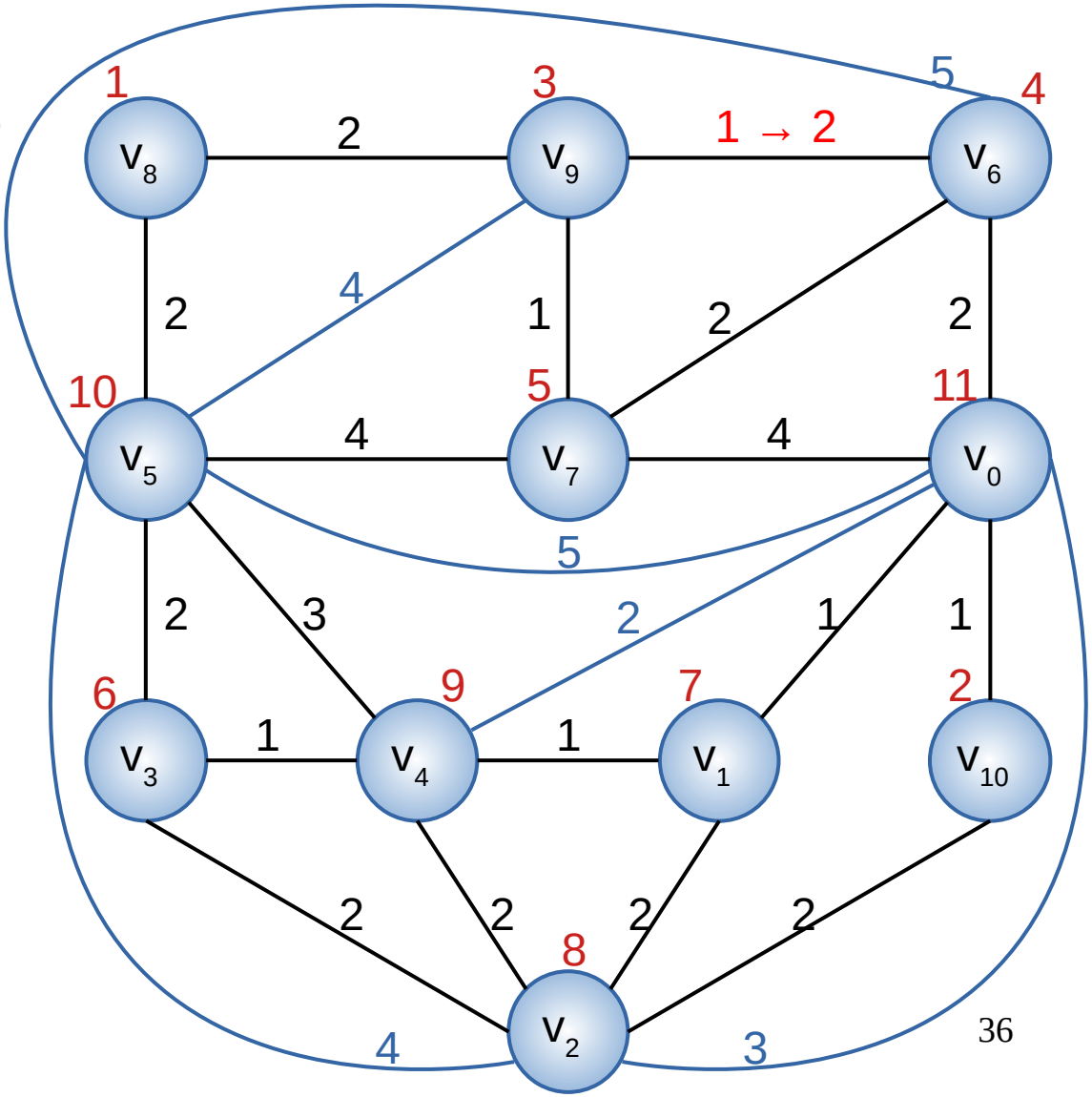
Weight Increase

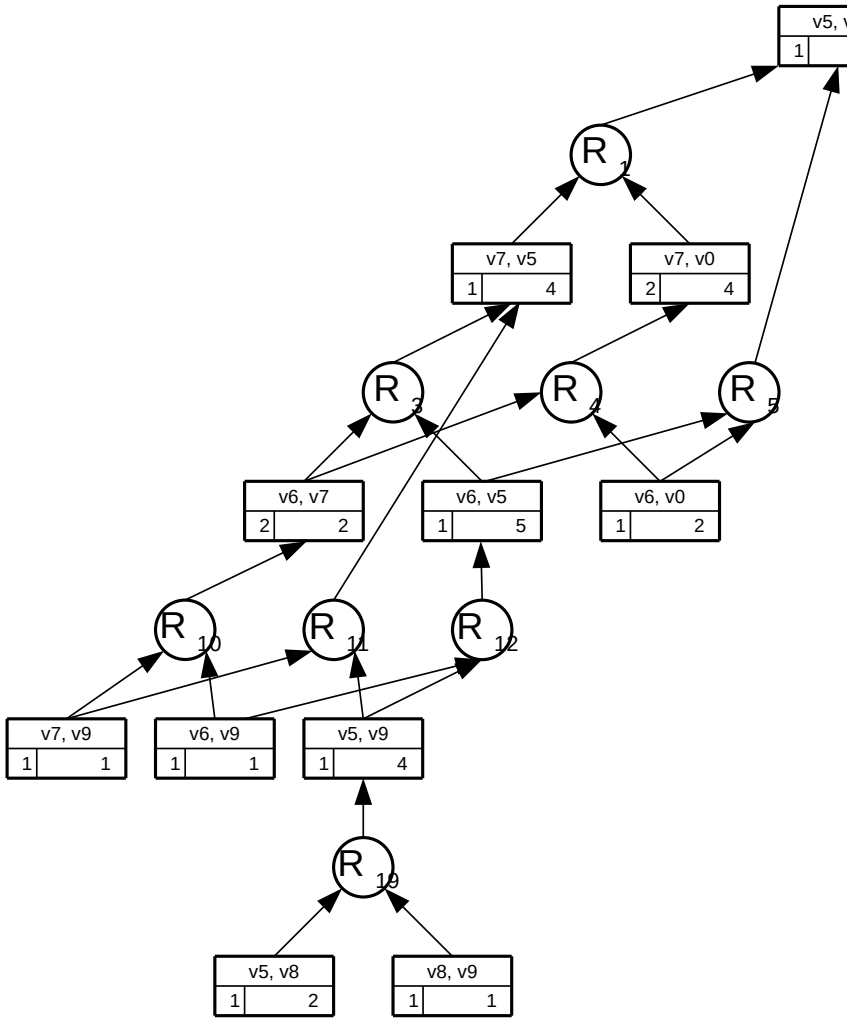
- Weight decrease ($v_9\, v_6$) = 2 → 1
- Implies further weight changes
- Let's use G* to find out which



Weight Increase

- Weight decrease ($v_9\, v_6$) = 2 → 1
- Implies further weight changes
- Let's use G* to find out which

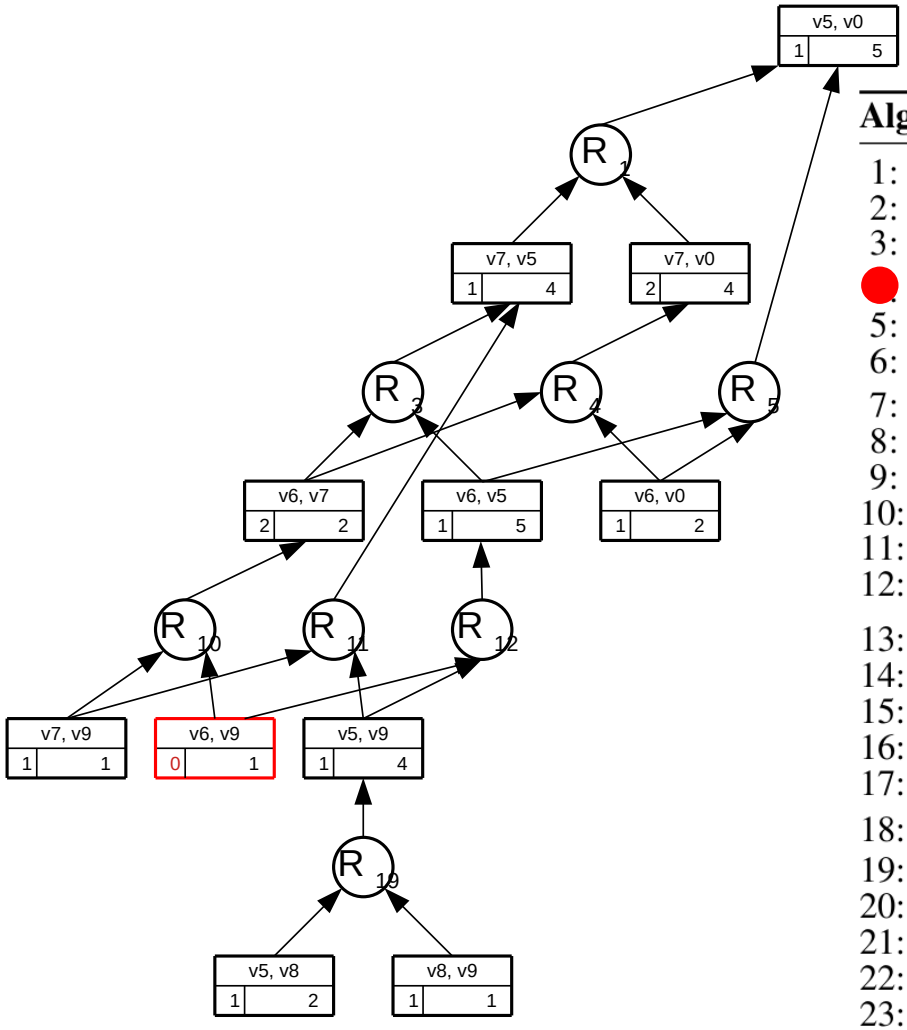




e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

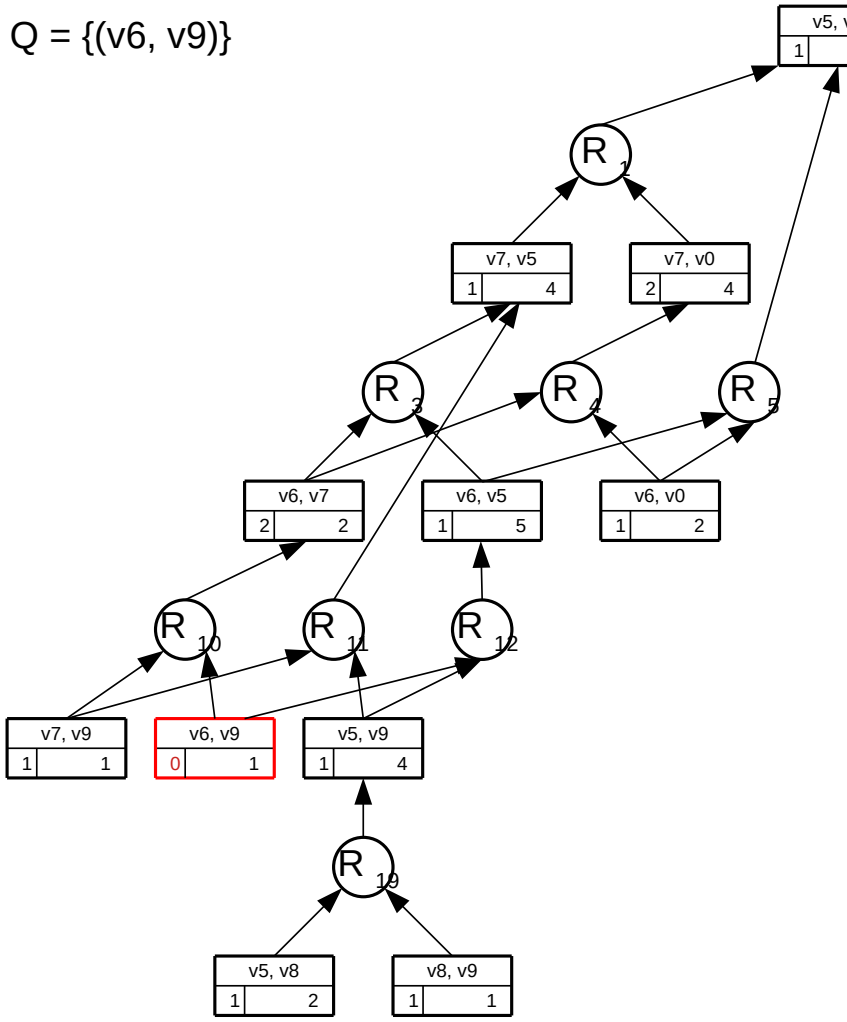


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

1: PriorityQueue $Q \leftarrow \emptyset$;
2: **if** $\phi(v_e, G^*) = \phi(e, G)$ **then**
3: $c_\phi(v_e) \leftarrow c_\phi(v_e) - 1$;
4: $\phi(e, G) \leftarrow k$;
5: **if** $c_\phi(v_e) < 1$ **then**
6: Q.push(v_e);
7: **while** $Q \neq \emptyset$ **do**
8: $v_s \leftarrow Q$.pop(); updateWeight(G^*, v'_s);
9: **for each** $v_r \in \text{nbr}^+(v_s)$ **do**
10: $v'_s \leftarrow \text{nbr}^+(v_r)$;
11: **if** $c_\phi(v'_s) < 1$ **then**
12: **if** $v'_s \notin Q$ **then** Q.push(v'_s);
13: **procedure** updateWeight(G^*, v_s)
14: **for each** $v_r \in \text{nbr}^+(v_s)$ **do**
15: $v_{s_1}, v_{s_2} \leftarrow \text{nbr}^-(v_r)$; $v_{s_3} \leftarrow \text{nbr}^+(v_r)$;
16: **if** $\phi(v_{s_3}, G^*) = \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
17: $c_\phi(v_{s_3}) \leftarrow c_\phi(v_{s_3}) - 1$;
18: $\phi \leftarrow \phi(s, G)$; $c_\phi(v_s) \leftarrow 1$;
19: **for each** $v_r \in \text{nbr}^-(v_s)$ **do**
20: $v_{s_1}, v_{s_2} \leftarrow \text{nbr}^-(v_r)$;
21: **if** $\phi > \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
22: $\phi \leftarrow \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$; $c_\phi(v_s) \leftarrow 1$;
23: **else if** $\phi = \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
24: $c_\phi(v_s) \leftarrow c_\phi(v_s) + 1$;
25: $\phi(v_s, G^*) \leftarrow \phi$;

Q = {(v6, v9)}

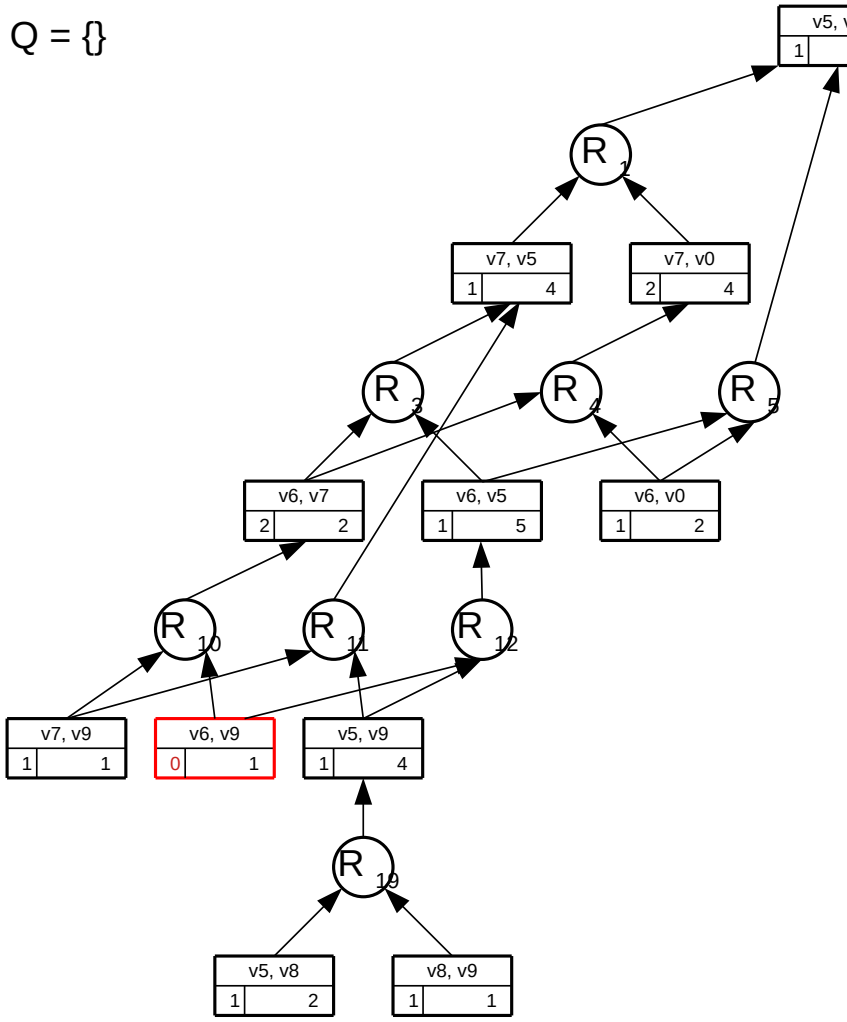


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
    while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

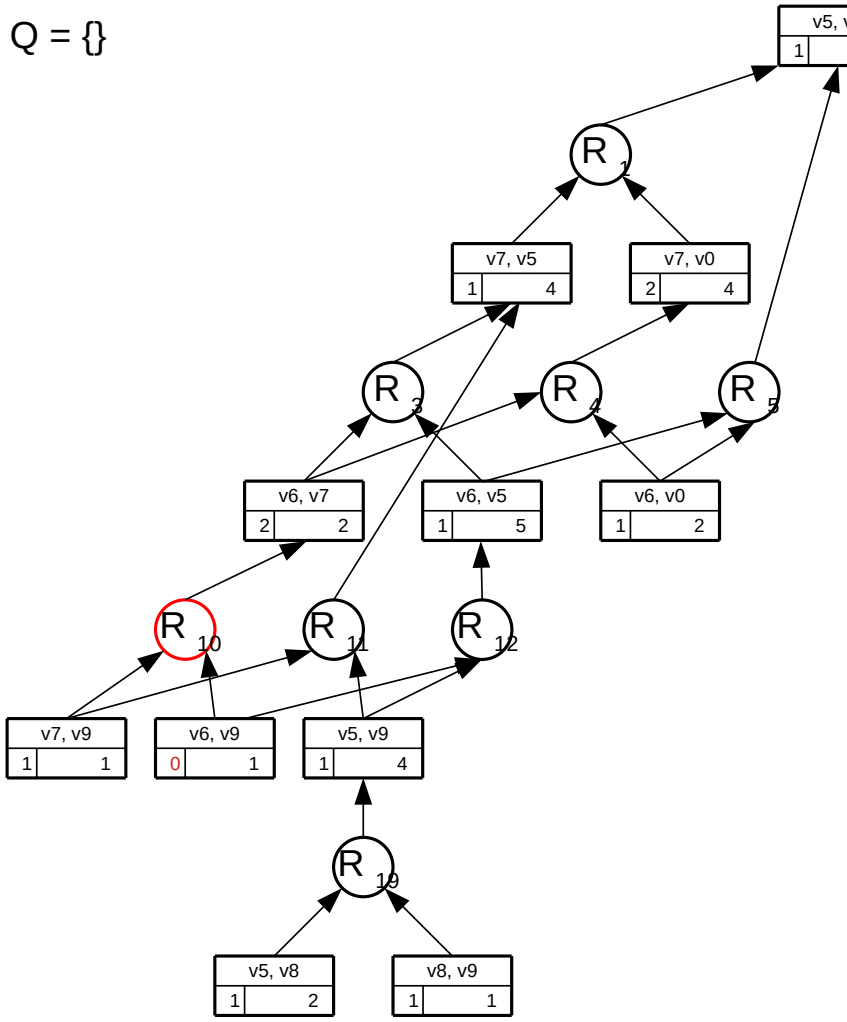


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

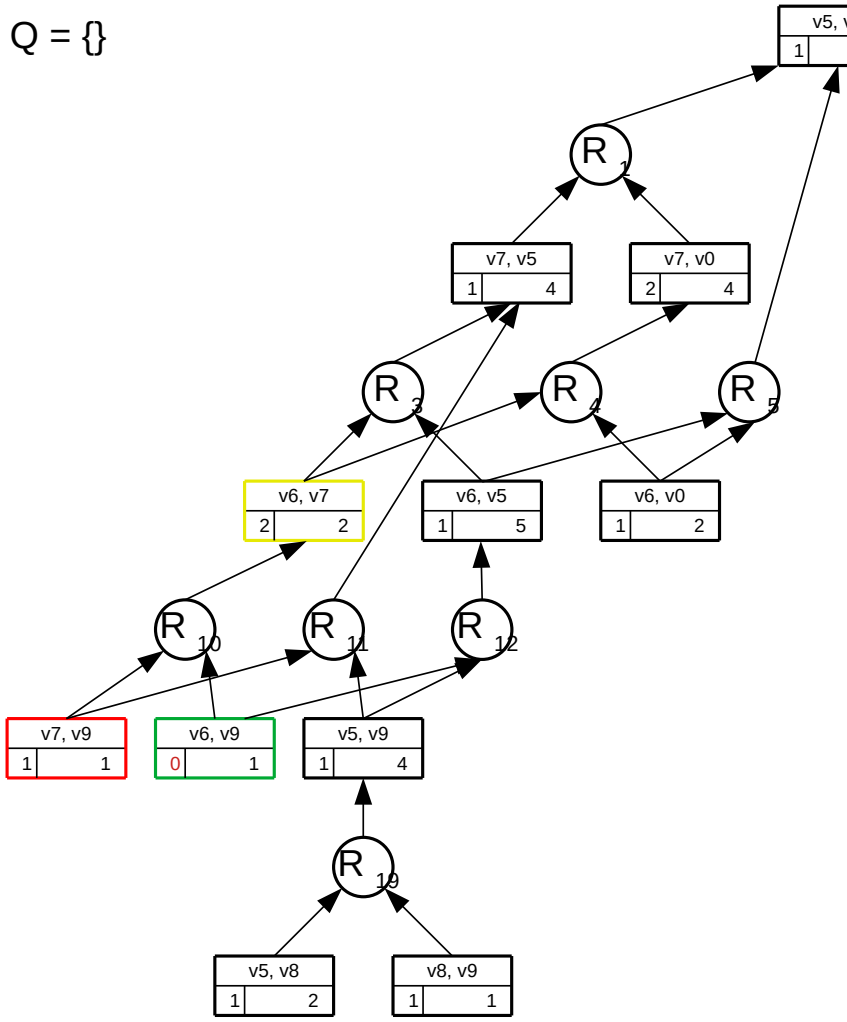


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

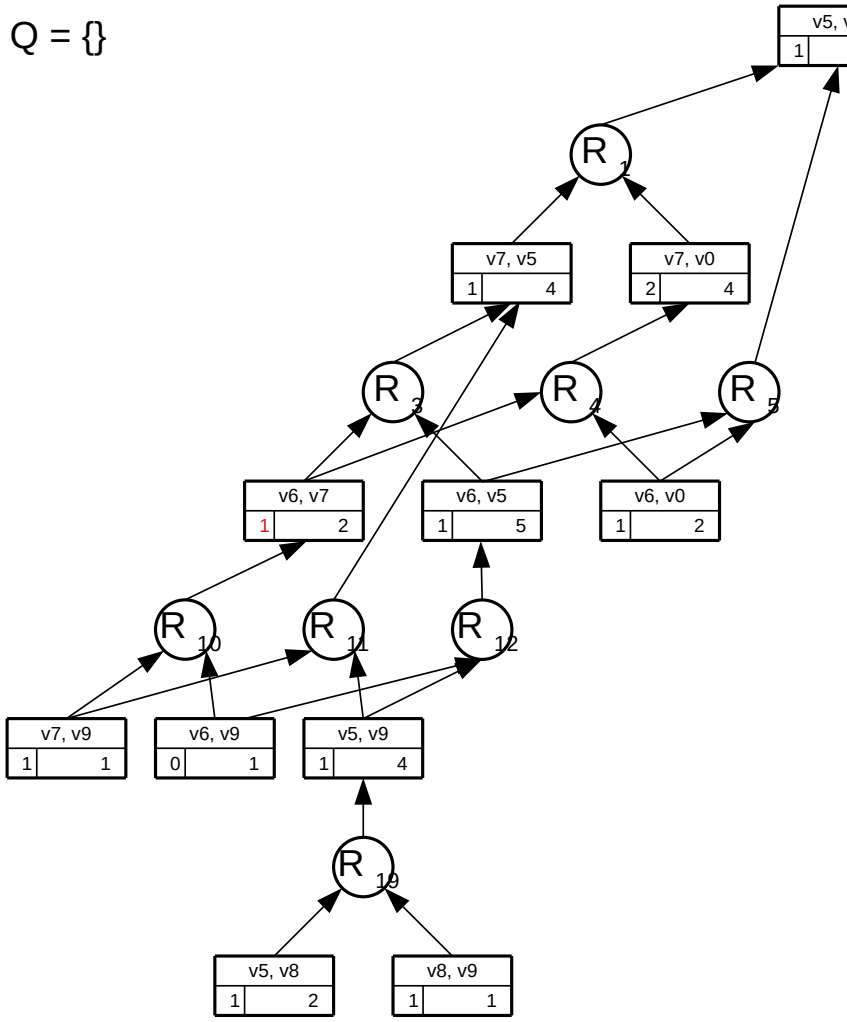


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

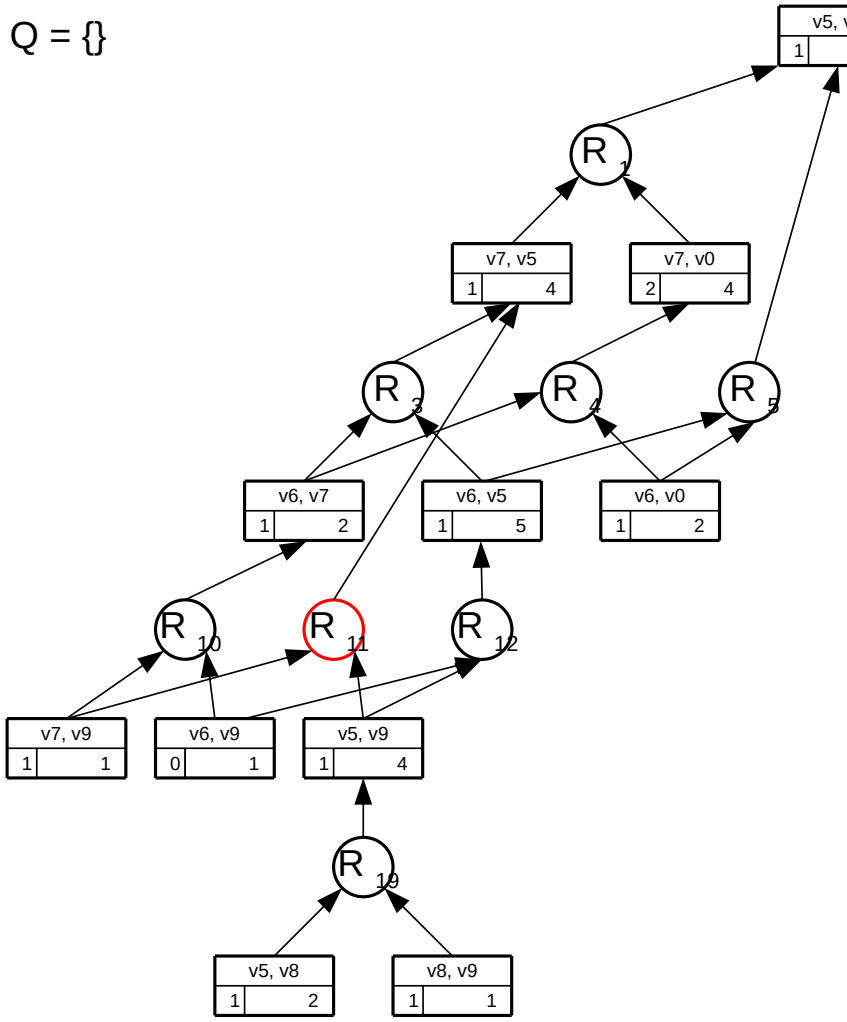


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

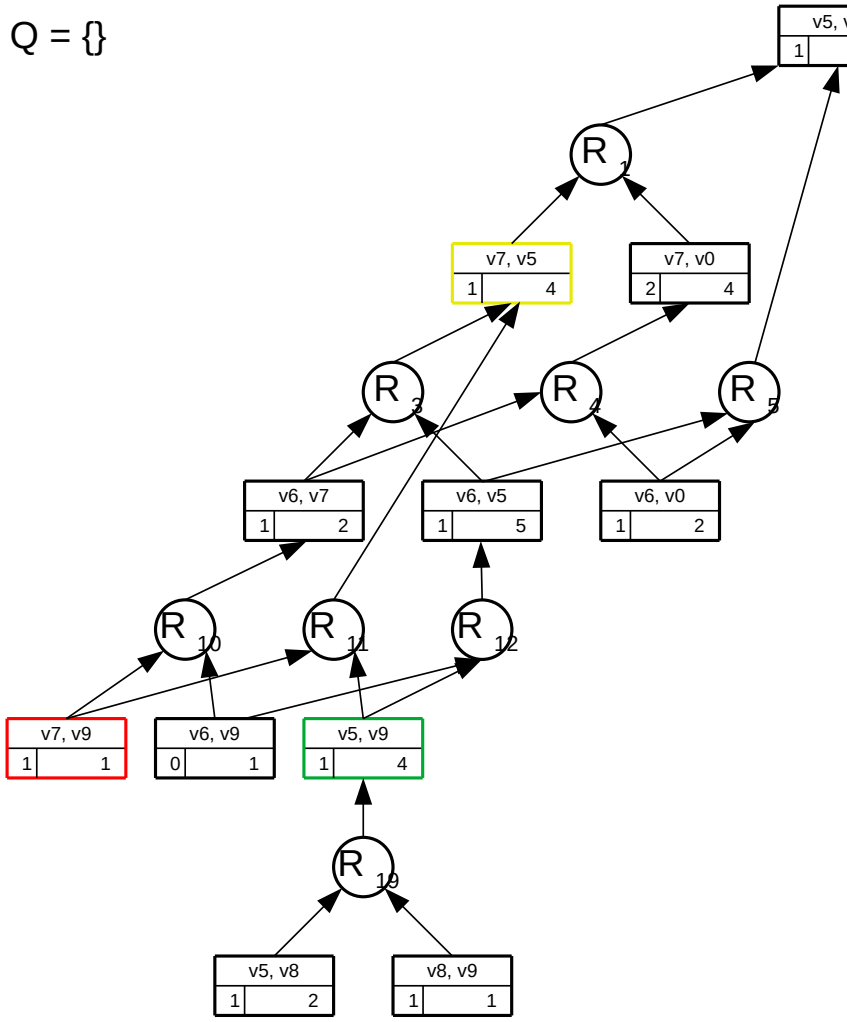


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

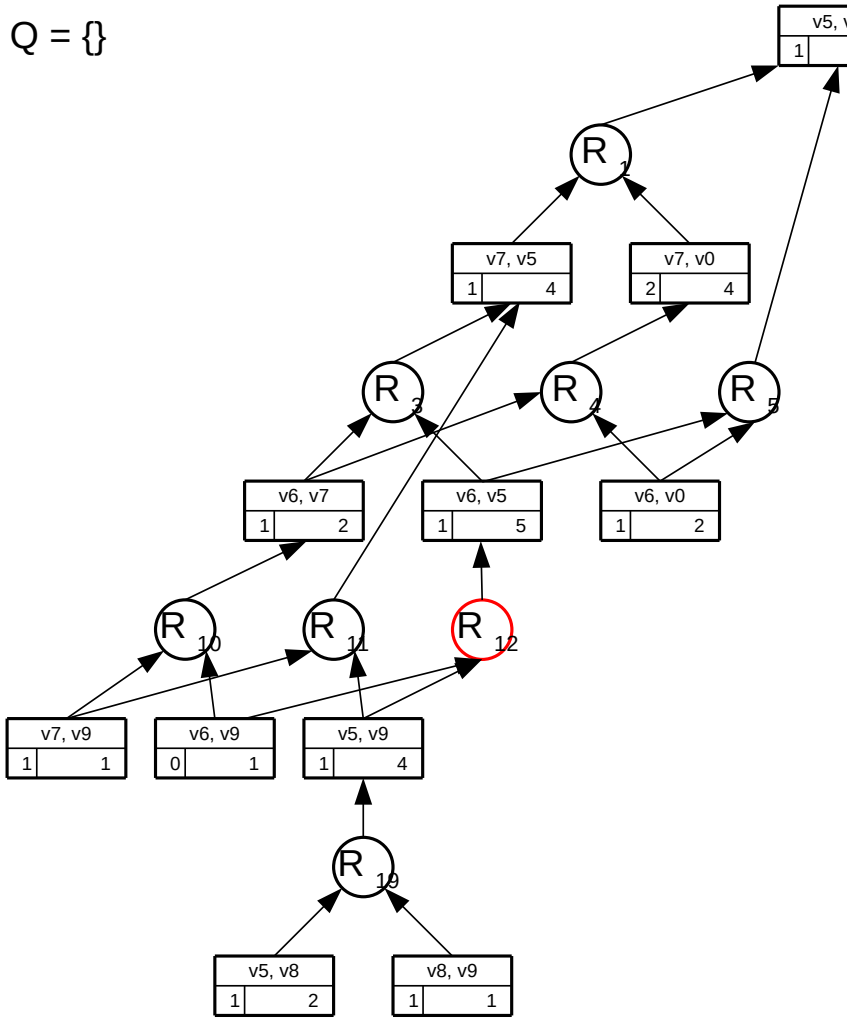


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

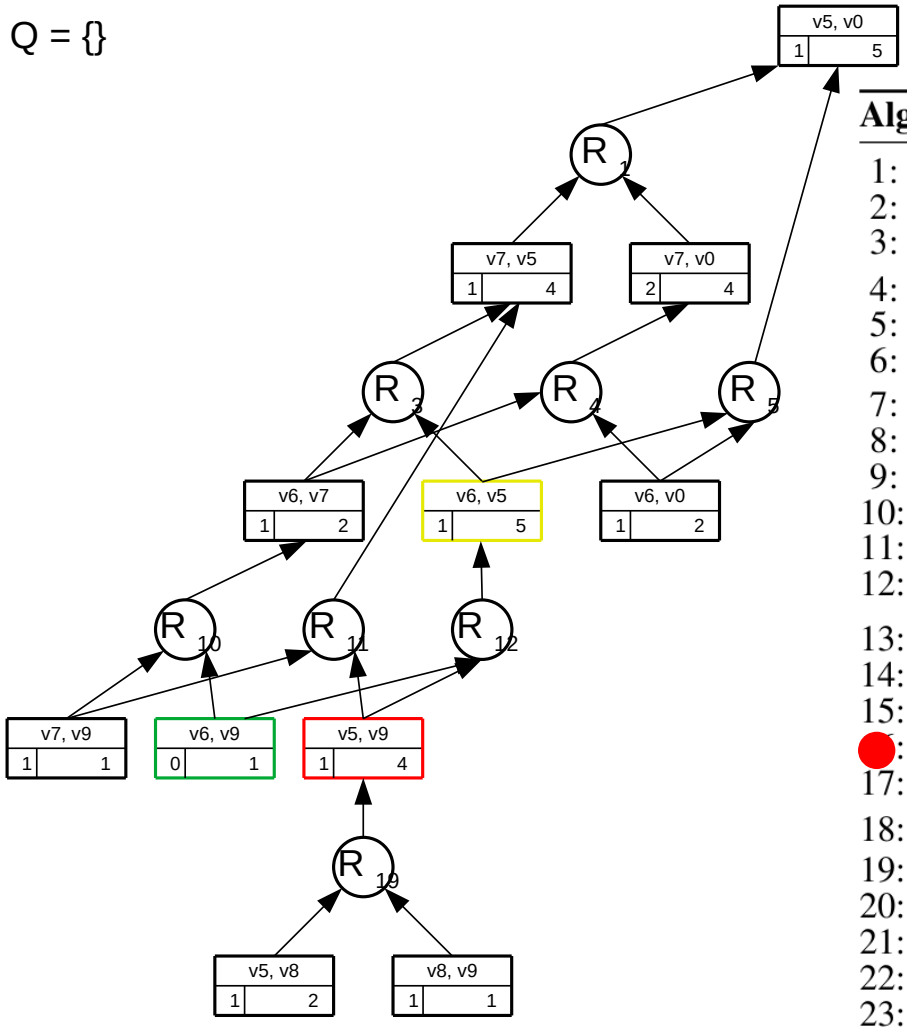


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

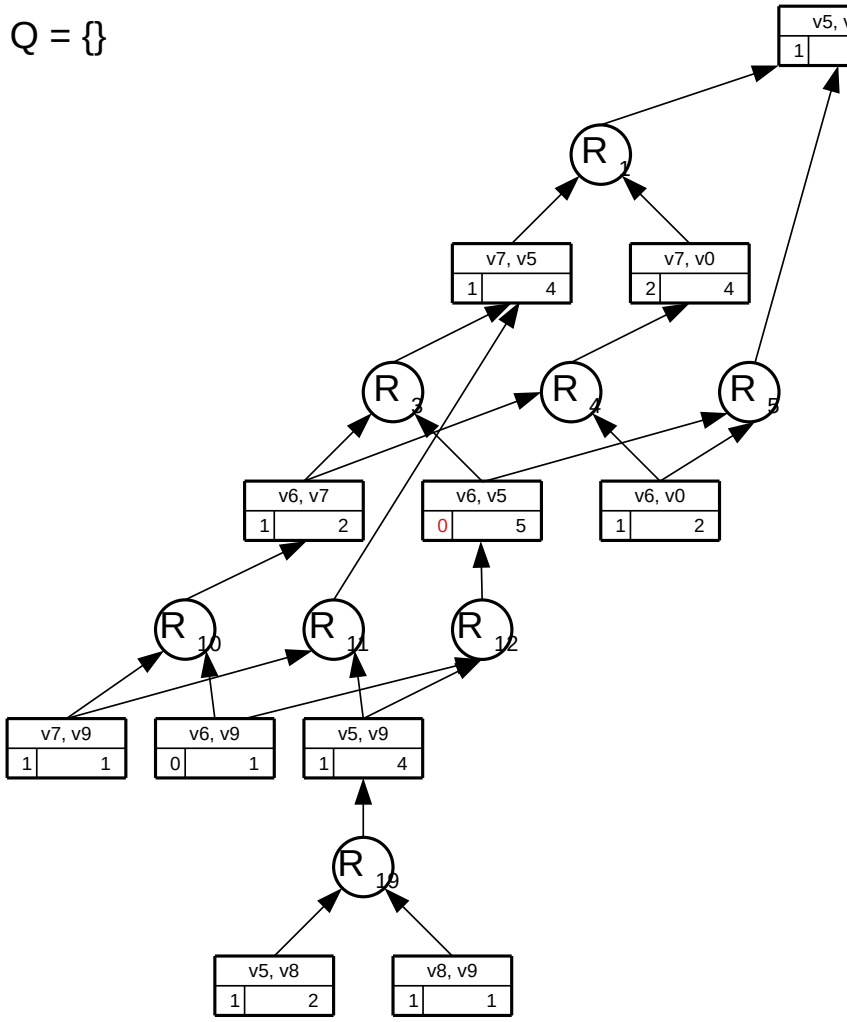


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

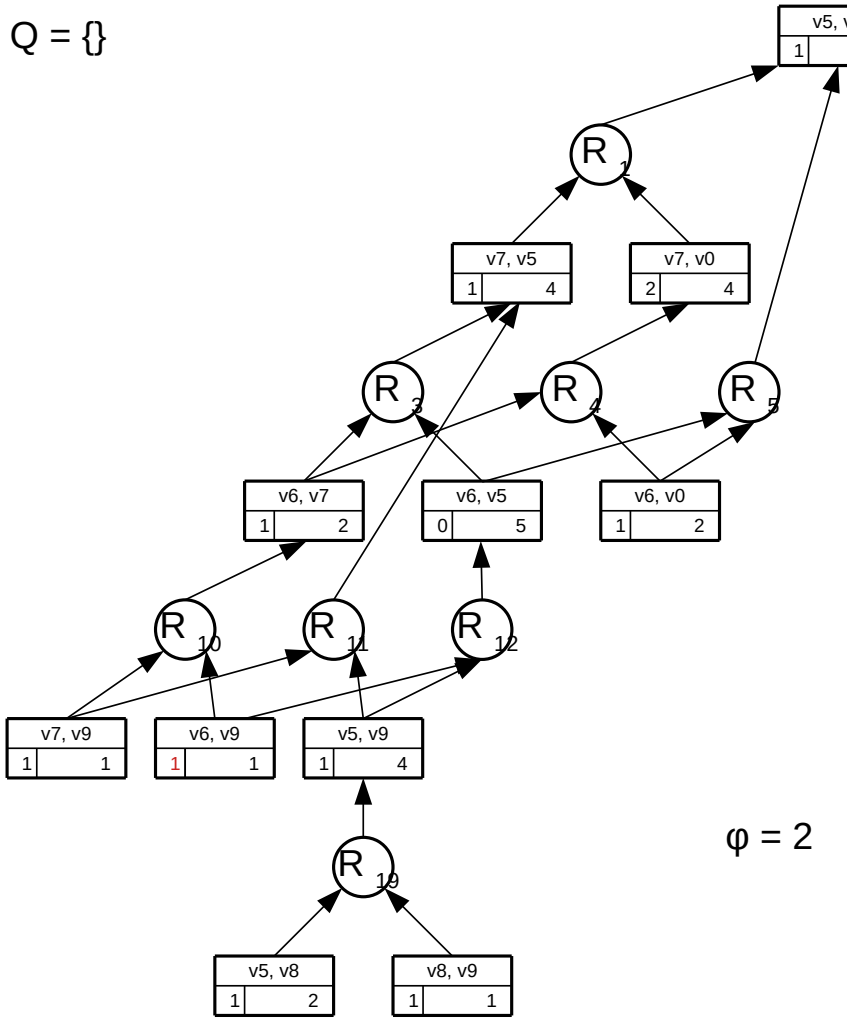


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



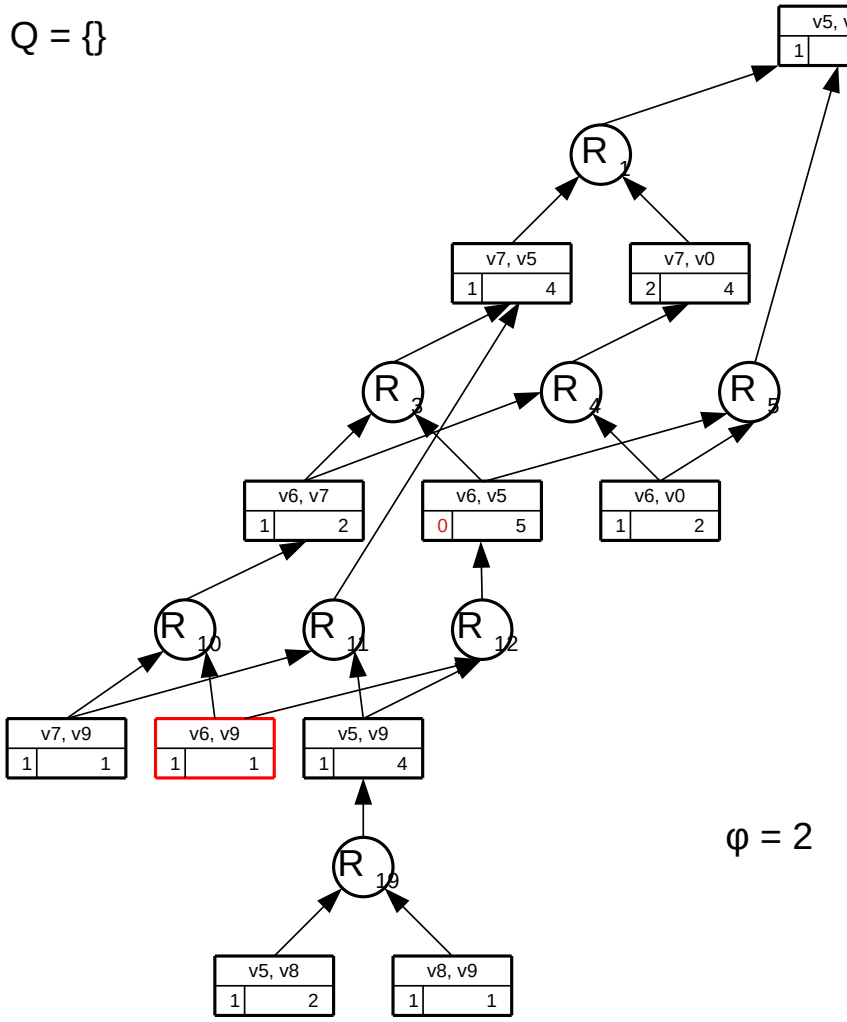
φ = 2

e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



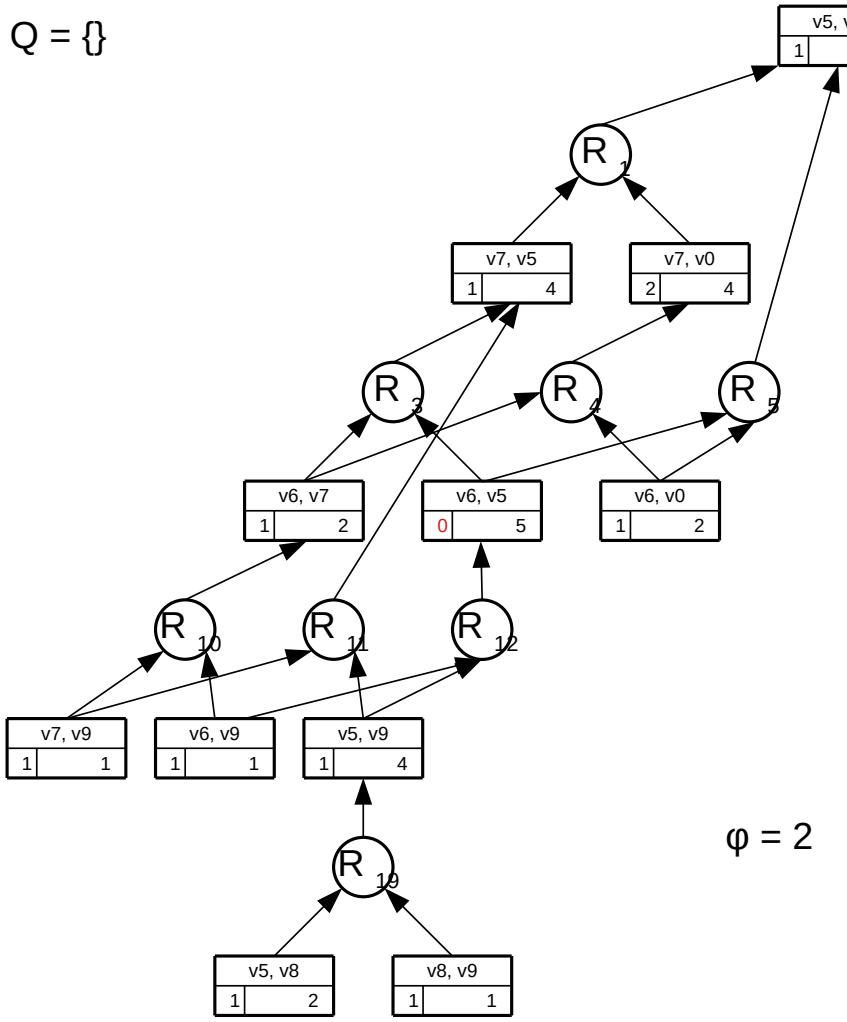
φ = 2

e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



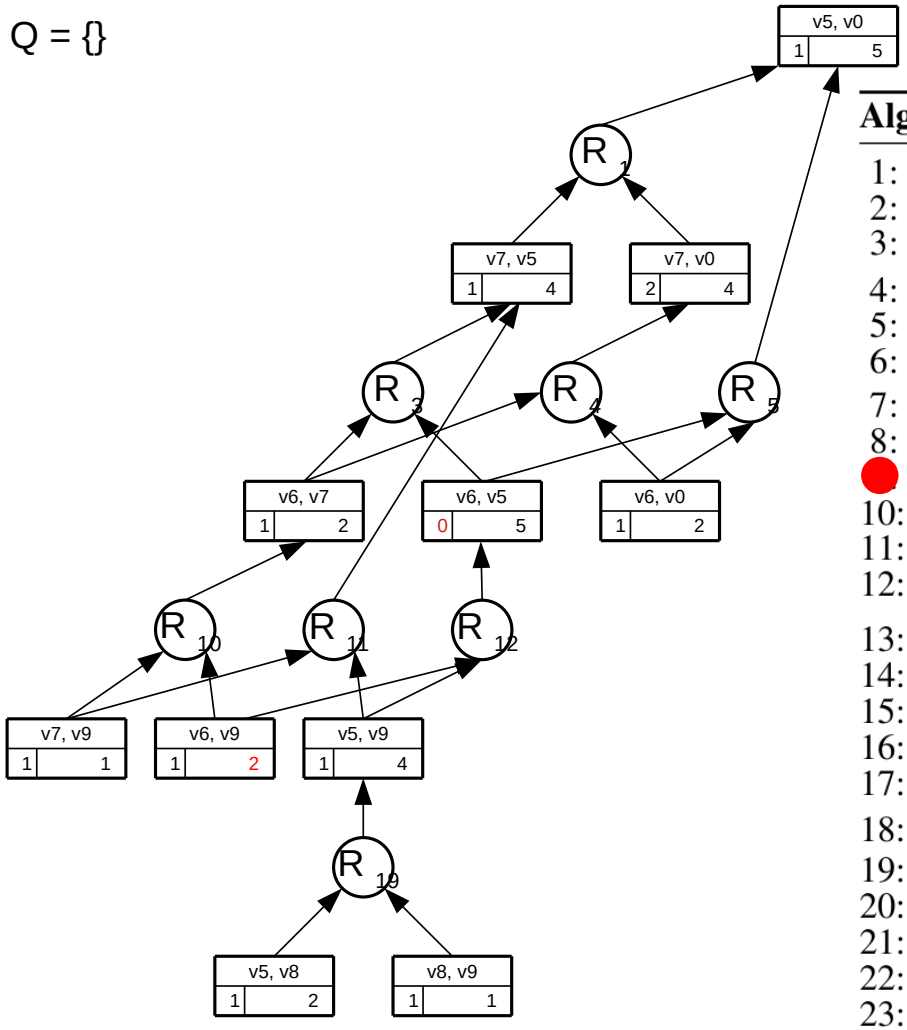
φ = 2

e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do nbr−(v_s) = {}
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

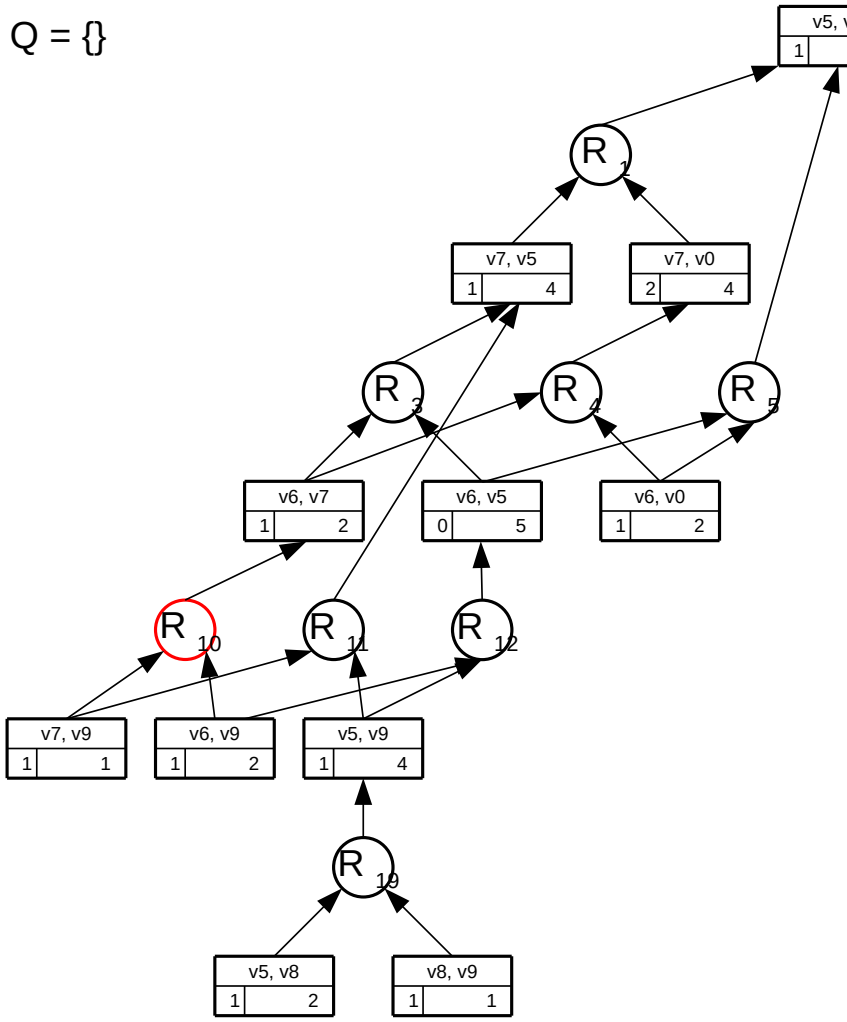


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

1: PriorityQueue $Q \leftarrow \emptyset$;
2: **if** $\phi(v_e, G^*) = \phi(e, G)$ **then**
3: $\quad c_\phi(v_e) \leftarrow c_\phi(v_e) - 1$;
4: $\phi(e, G) \leftarrow k$;
5: **if** $c_\phi(v_e) < 1$ **then**
6: $\quad Q$.push(v_e);
7: **while** $Q \neq \emptyset$ **do**
8: $\quad v_s \leftarrow Q$.pop(); updateWeight(G^*, v'_s);
9: $\quad$ **for each** $v_r \in \text{nbr}^+(v_s)$ **do**
10: $\quad\quad v'_s \leftarrow \text{nbr}^+(v_r)$;
11: $\quad\quad$ **if** $c_\phi(v'_s) < 1$ **then**
12: $\quad\quad\quad$ **if** $v'_s \notin Q$ **then** Q.push(v'_s);
13: **procedure** updateWeight(G^*, v_s)
14: $\quad$ **for each** $v_r \in \text{nbr}^+(v_s)$ **do**
15: $\quad\quad v_{s_1}, v_{s_2} \leftarrow \text{nbr}^-(v_r)$; $v_{s_3} \leftarrow \text{nbr}^+(v_r)$;
16: $\quad\quad$ **if** $\phi(v_{s_3}, G^*) = \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
17: $\quad\quad\quad c_\phi(v_{s_3}) \leftarrow c_\phi(v_{s_3}) - 1$;
18: $\quad \phi \leftarrow \phi(s, G)$; $c_\phi(v_s) \leftarrow 1$;
19: $\quad$ **for each** $v_r \in \text{nbr}^-(v_s)$ **do**
20: $\quad\quad v_{s_1}, v_{s_2} \leftarrow \text{nbr}^-(v_r)$;
21: $\quad\quad$ **if** $\phi > \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
22: $\quad\quad\quad \phi \leftarrow \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$; $c_\phi(v_s) \leftarrow 1$;
23: $\quad\quad$ **else if** $\phi = \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
24: $\quad\quad\quad c_\phi(v_s) \leftarrow c_\phi(v_s) + 1$;
25: $\quad \phi(v_s, G^*) \leftarrow \phi$;

Q = {}

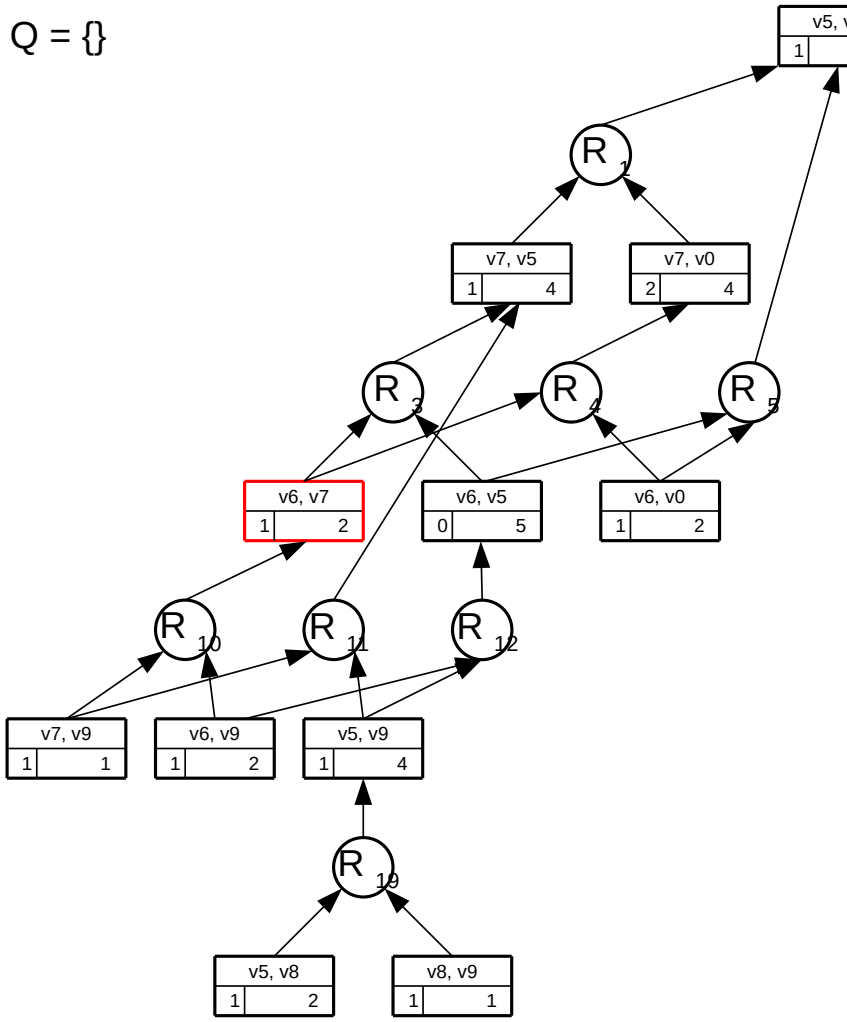


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

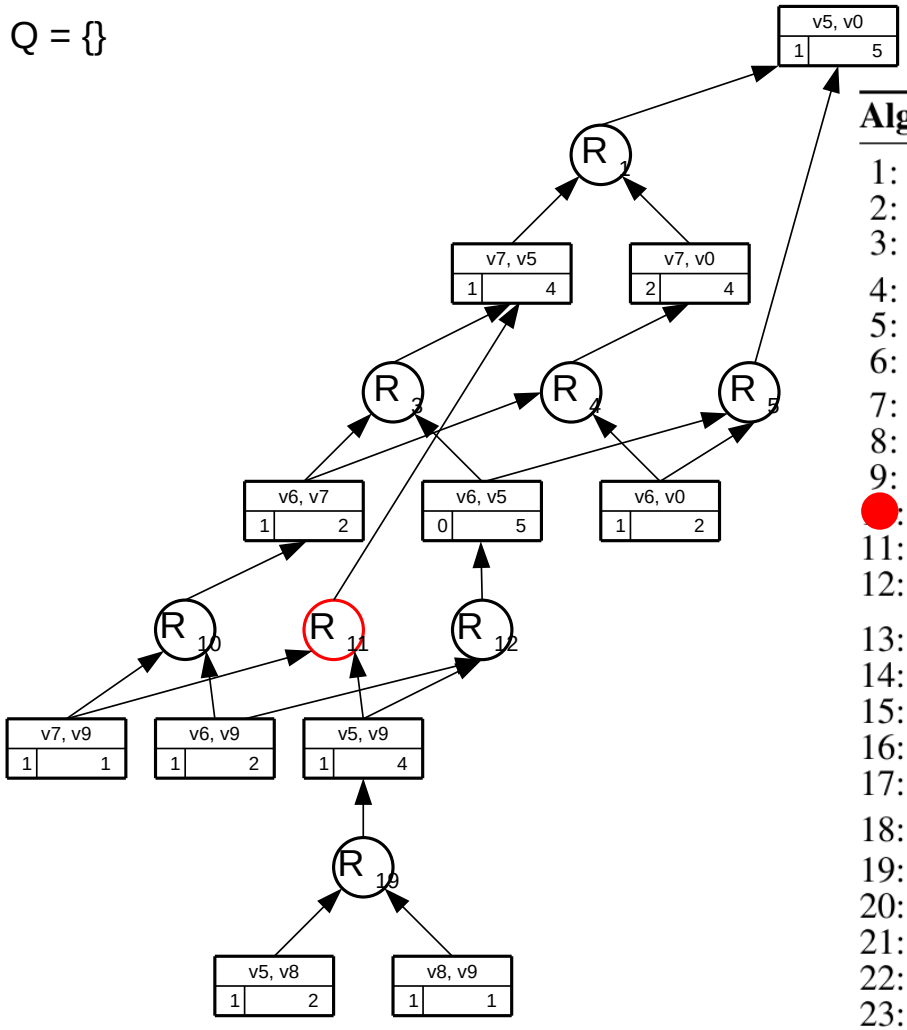


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
            if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

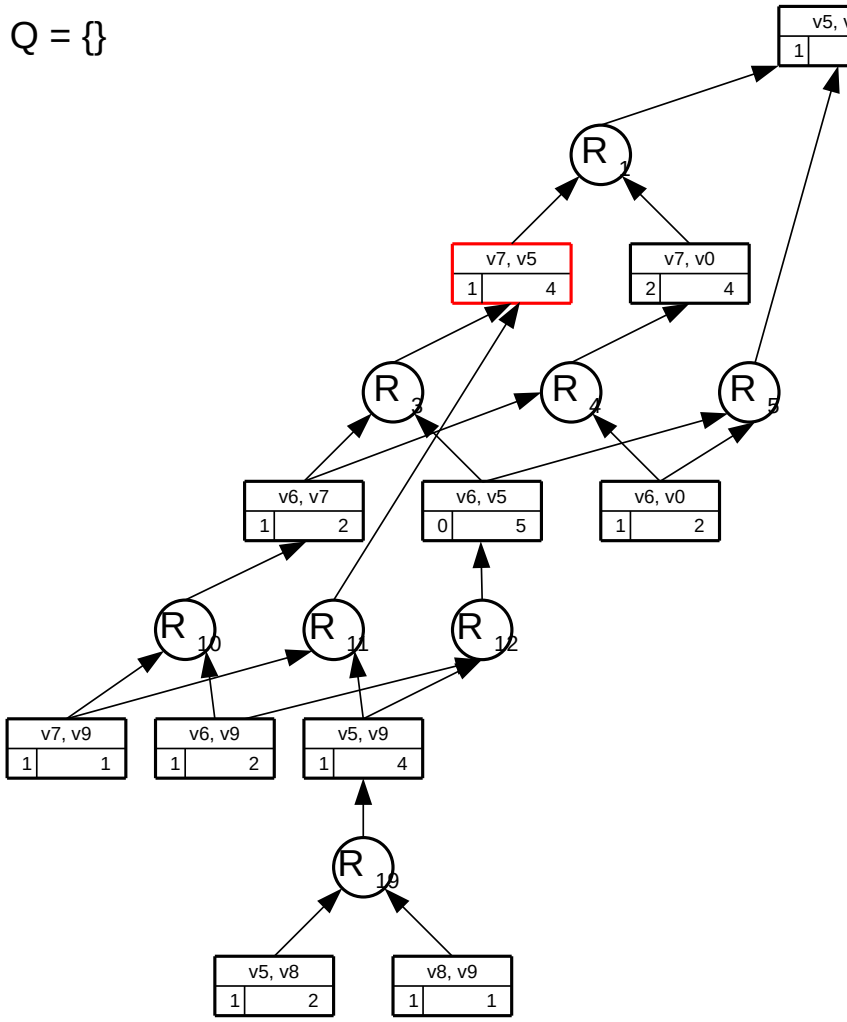


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

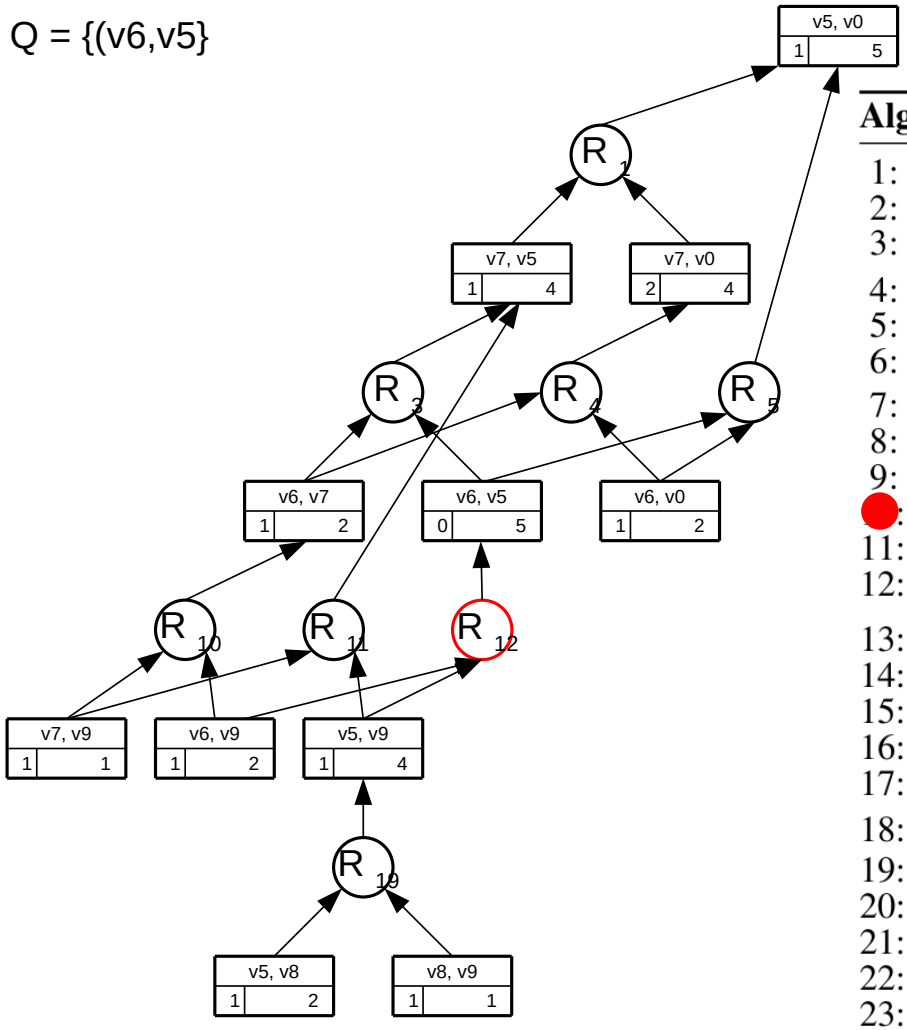


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
            if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {(v6,v5}

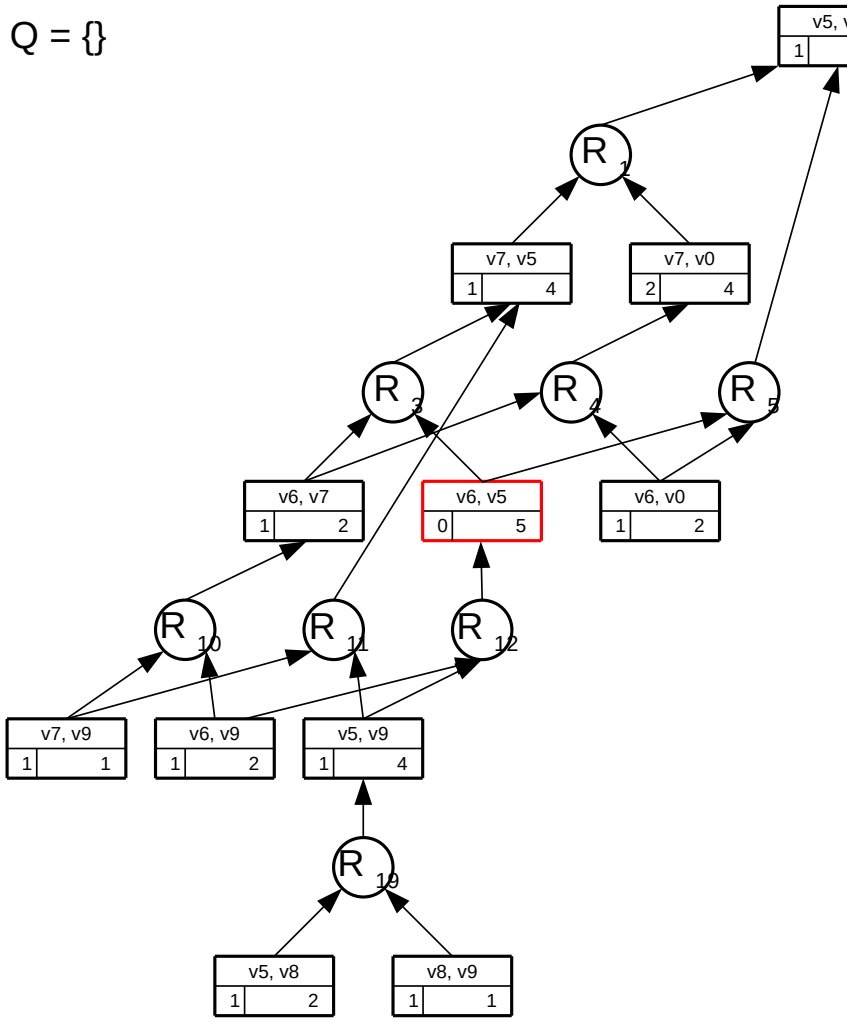


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

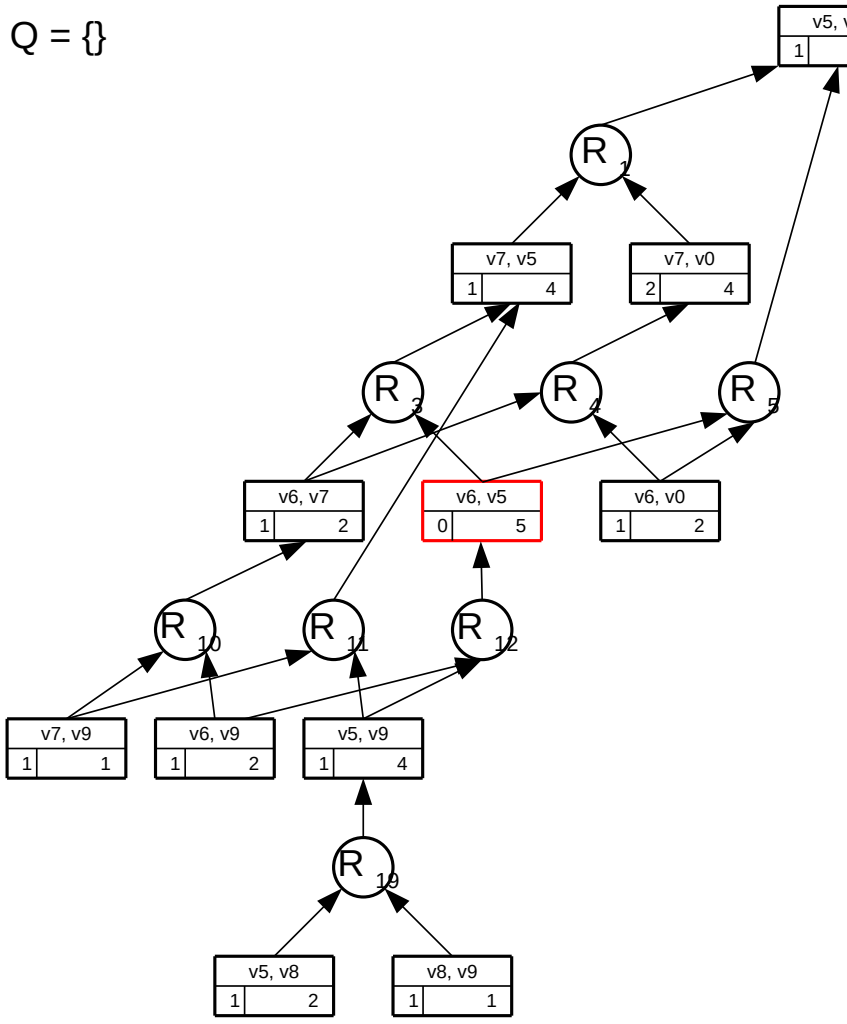


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
            if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

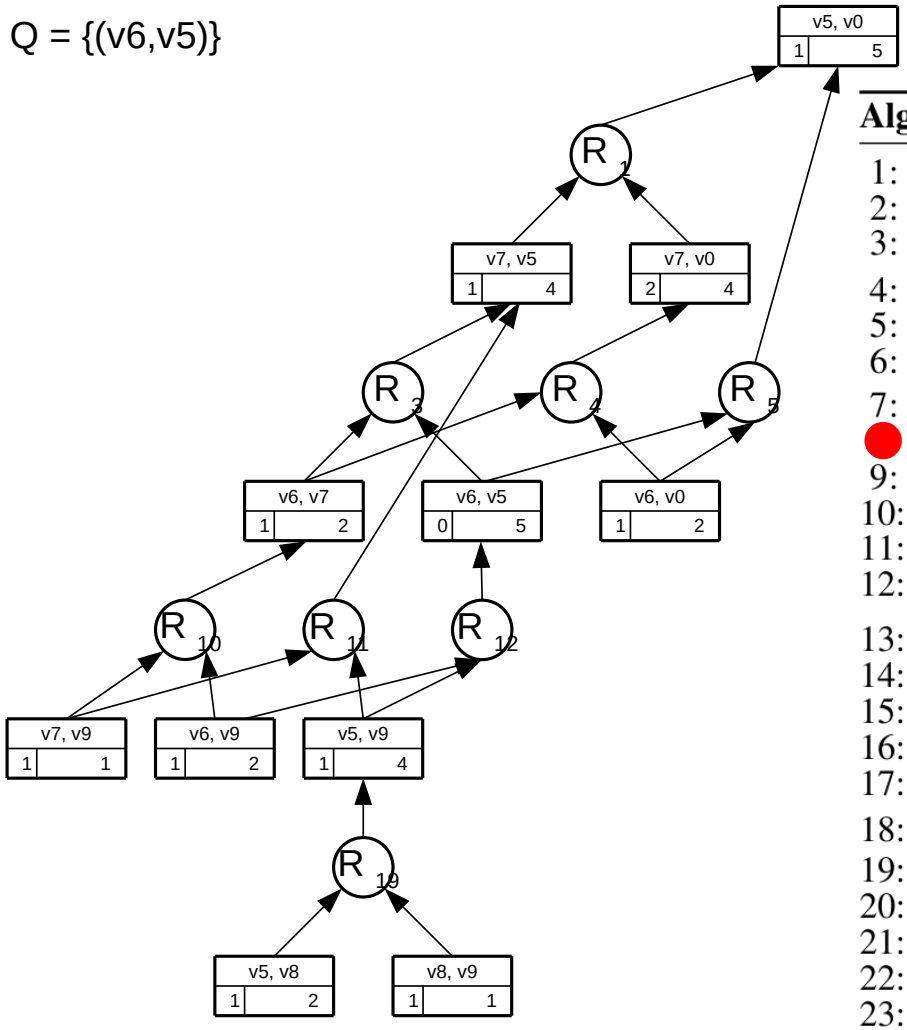


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

1: PriorityQueue $Q \leftarrow \emptyset$;
2: **if** $\phi(v_e, G^*) = \phi(e, G)$ **then**
3: $\quad c_\phi(v_e) \leftarrow c_\phi(v_e) - 1$;
4: $\phi(e, G) \leftarrow k$;
5: **if** $c_\phi(v_e) < 1$ **then**
6: $\quad Q$.push(v_e);
7: **while** $Q \neq \emptyset$ **do**
8: $\quad v_s \leftarrow Q$.pop(); updateWeight(G^*, v'_s);
9: $\quad$ **for each** $v_r \in \text{nbr}^+(v_s)$ **do**
10: $\quad\quad v'_s \leftarrow \text{nbr}^+(v_r)$;
11: $\quad\quad$ **if** $c_\phi(v'_s) < 1$ **then**
12: $\quad\quad\quad$ **if** $v'_s \notin Q$ **then** Q.push(v'_s);
13: **procedure** updateWeight(G^*, v_s)
14: $\quad$ **for each** $v_r \in \text{nbr}^+(v_s)$ **do**
15: $\quad\quad v_{s_1}, v_{s_2} \leftarrow \text{nbr}^-(v_r)$; $v_{s_3} \leftarrow \text{nbr}^+(v_r)$;
16: $\quad\quad$ **if** $\phi(v_{s_3}, G^*) = \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
17: $\quad\quad\quad c_\phi(v_{s_3}) \leftarrow c_\phi(v_{s_3}) - 1$;
18: $\quad \phi \leftarrow \phi(s, G)$; $c_\phi(v_s) \leftarrow 1$;
19: $\quad$ **for each** $v_r \in \text{nbr}^-(v_s)$ **do**
20: $\quad\quad v_{s_1}, v_{s_2} \leftarrow \text{nbr}^-(v_r)$;
21: $\quad\quad$ **if** $\phi > \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
22: $\quad\quad\quad \phi \leftarrow \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$; $c_\phi(v_s) \leftarrow 1$;
23: $\quad\quad$ **else if** $\phi = \phi(v_{s_1}, G^*) + \phi(v_{s_2}, G^*)$ **then**
24: $\quad\quad\quad c_\phi(v_s) \leftarrow c_\phi(v_s) + 1$;
25: $\quad \phi(v_s, G^*) \leftarrow \phi$;

Q = {(v6,v5)}

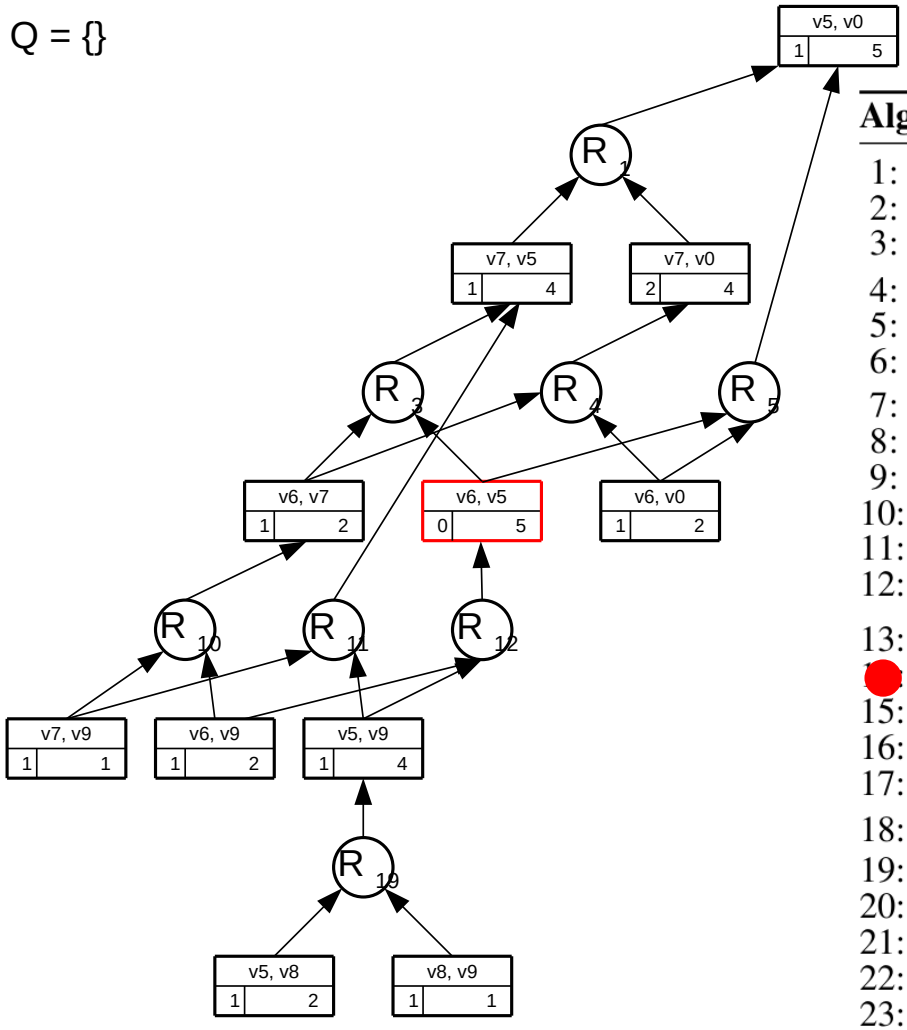


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



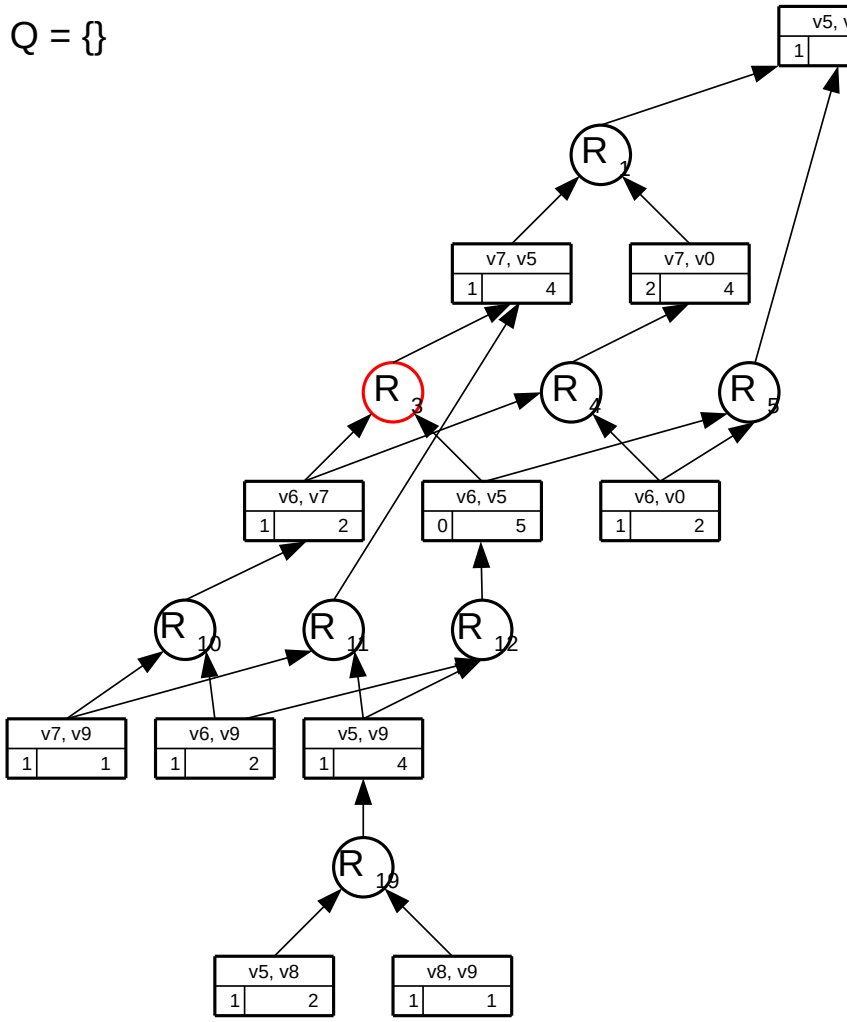
e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);

13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

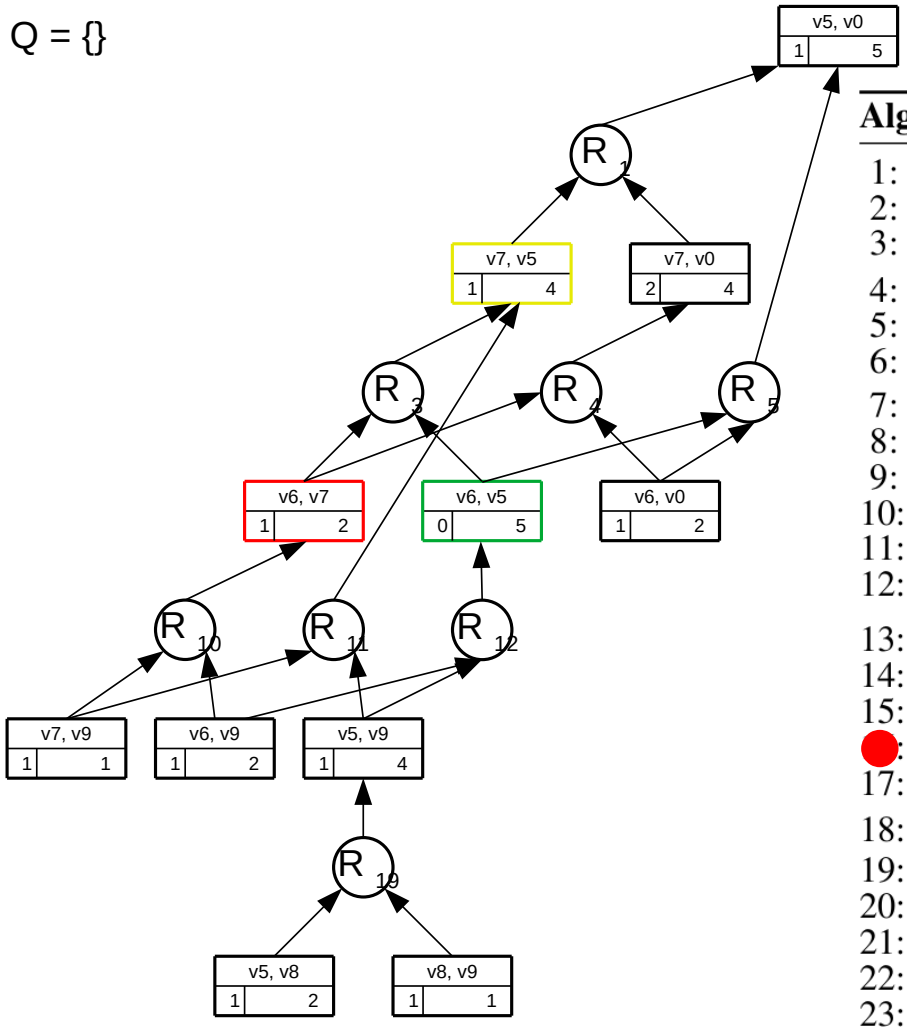


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

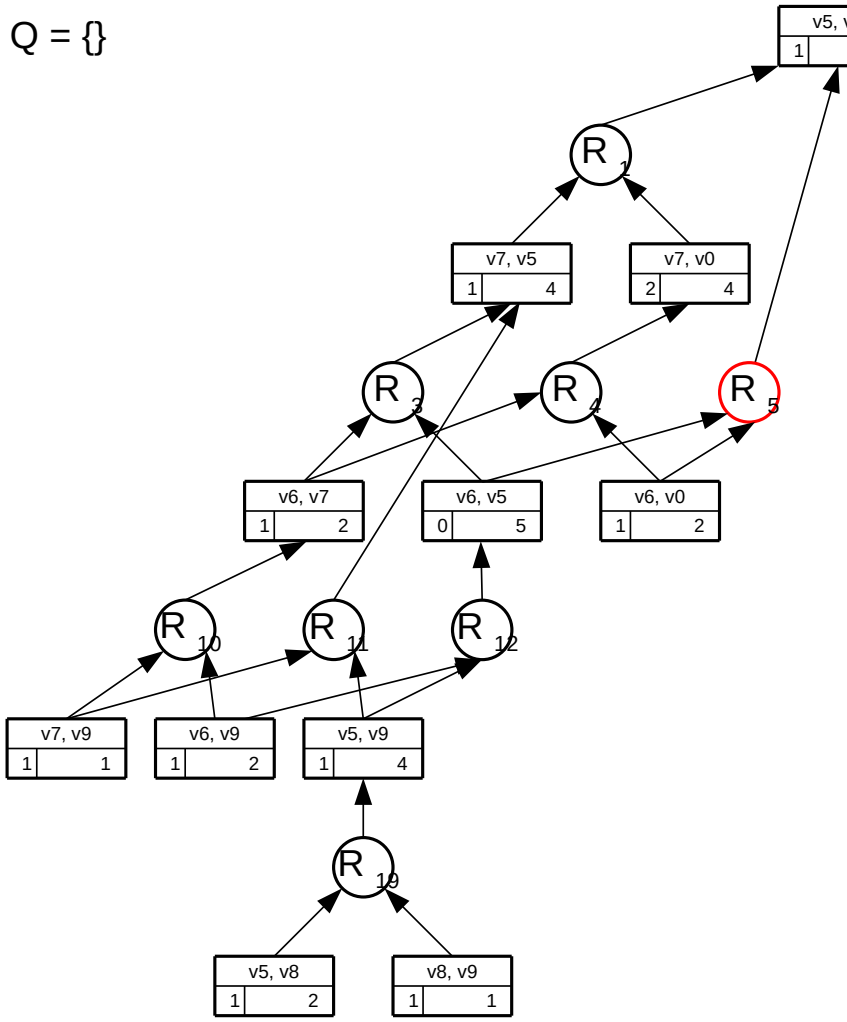


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

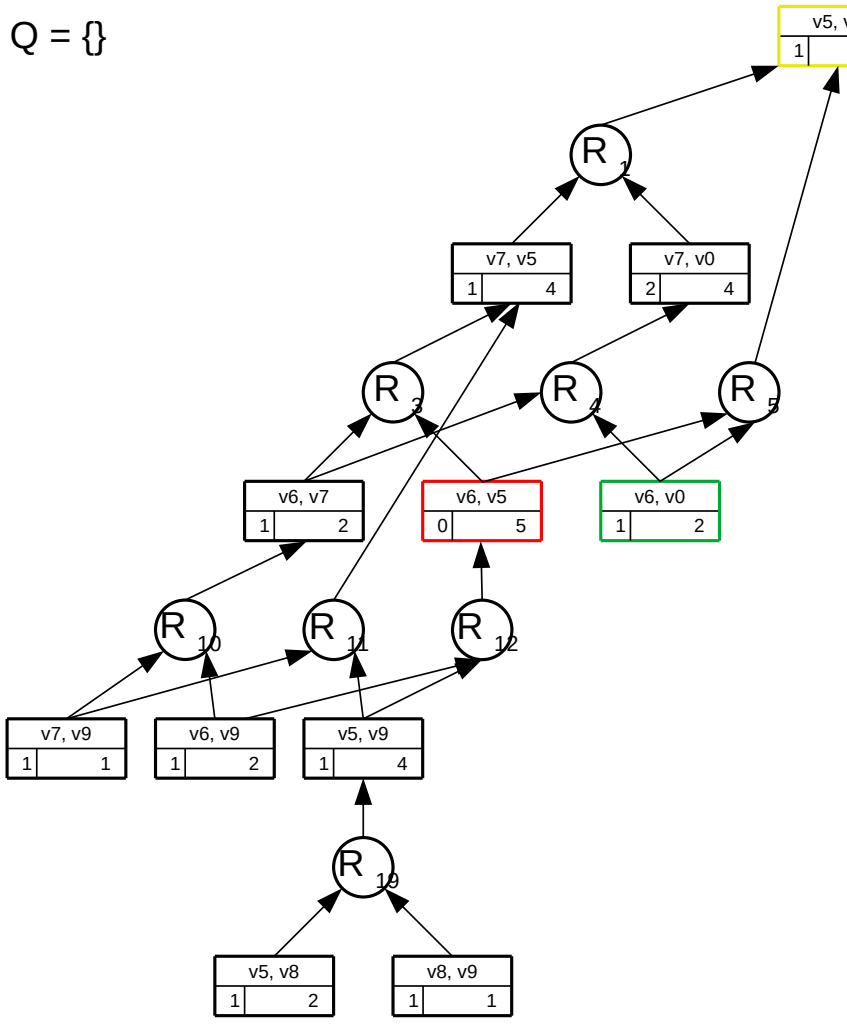


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}

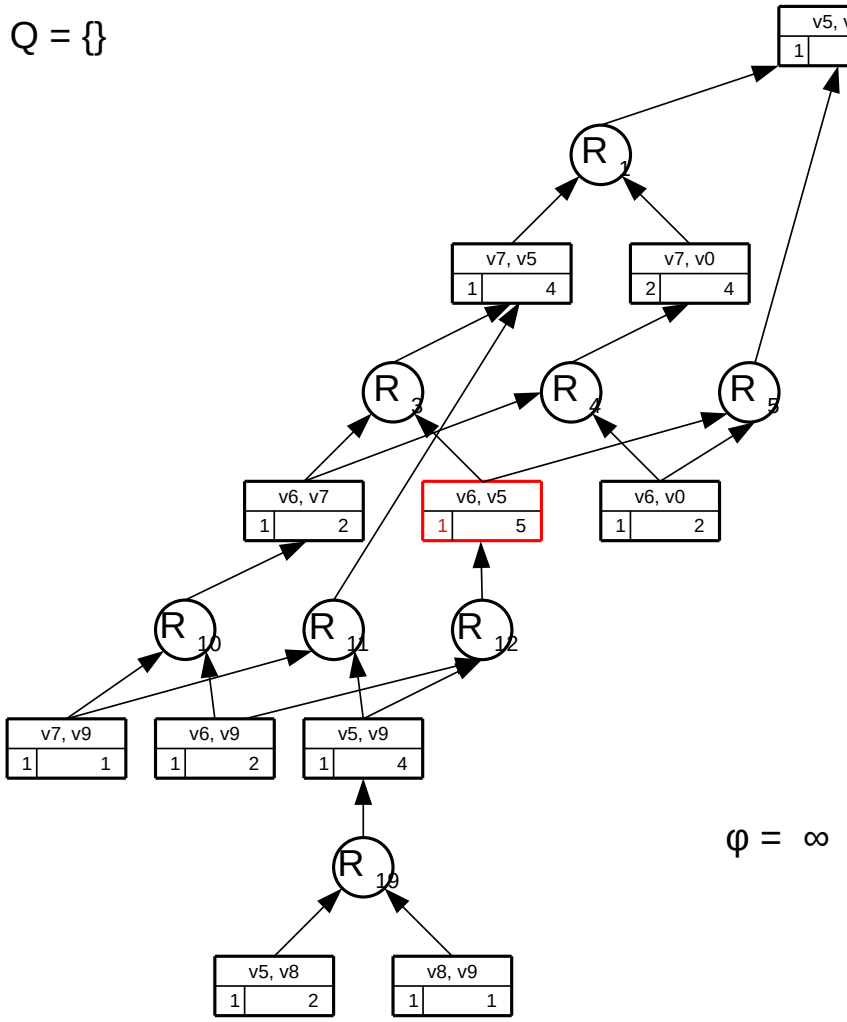


e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



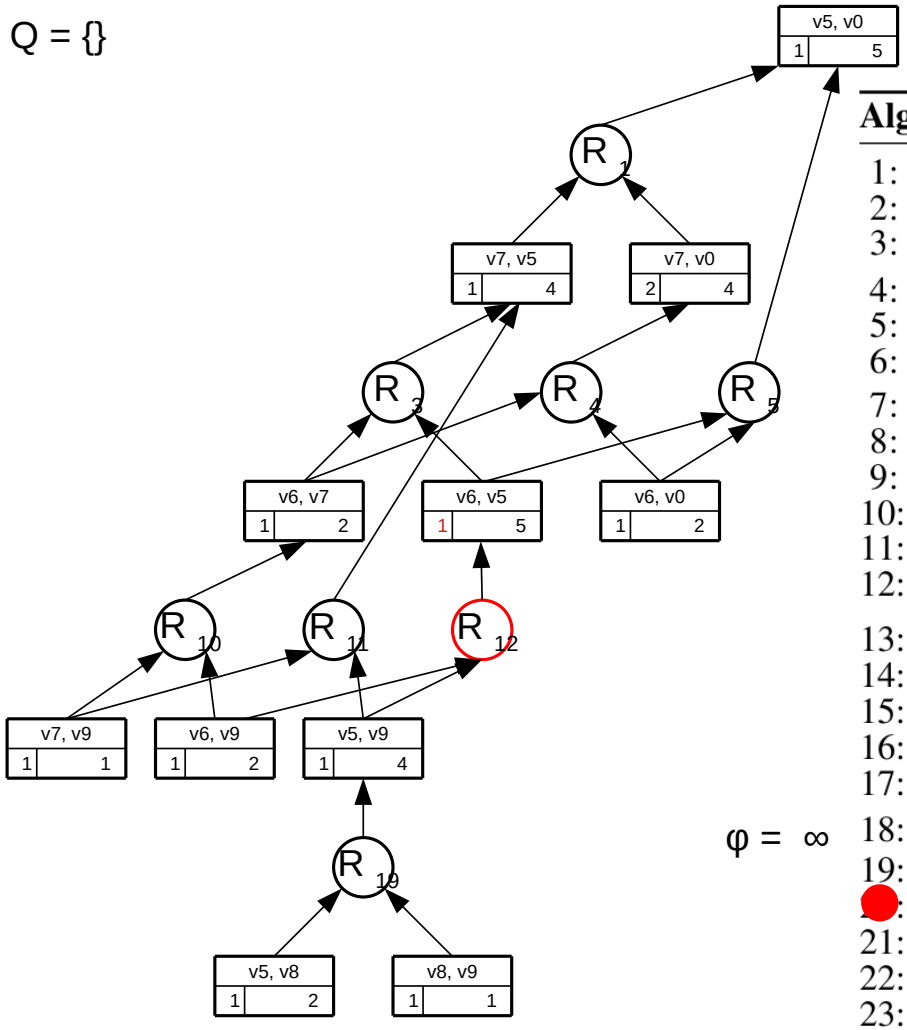
e = (v6, v9); k = 2

φ = ∞

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



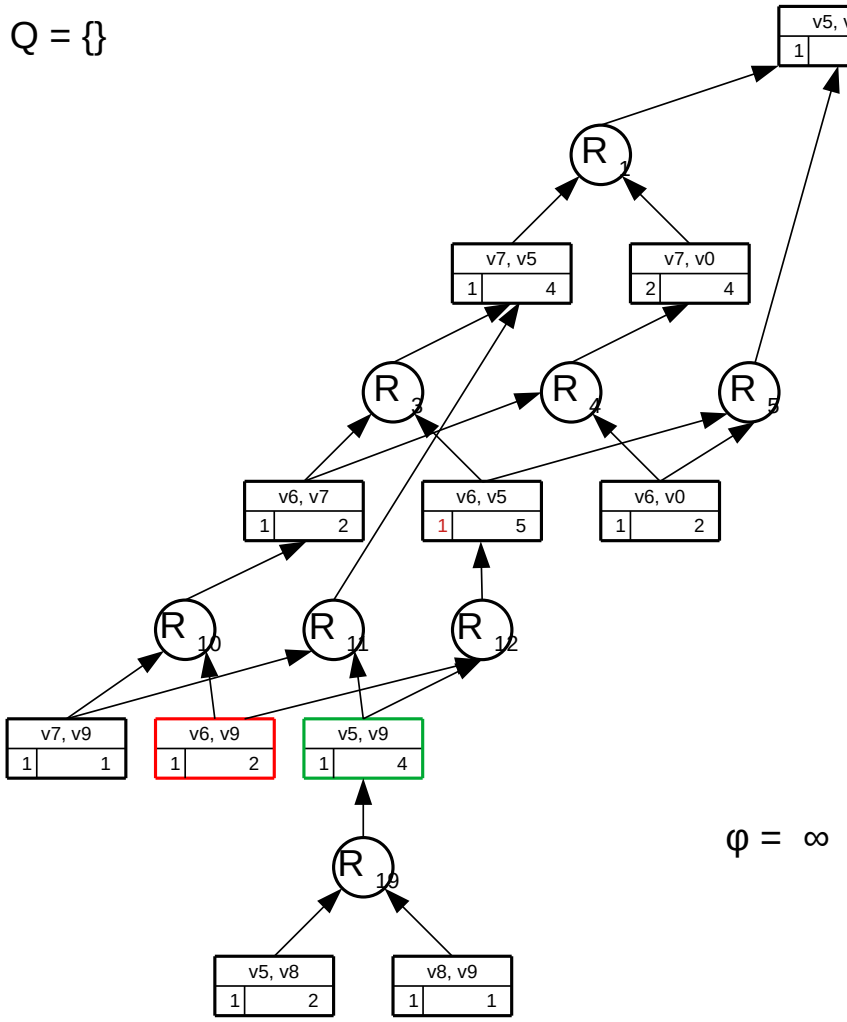
φ = ∞

e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_s) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



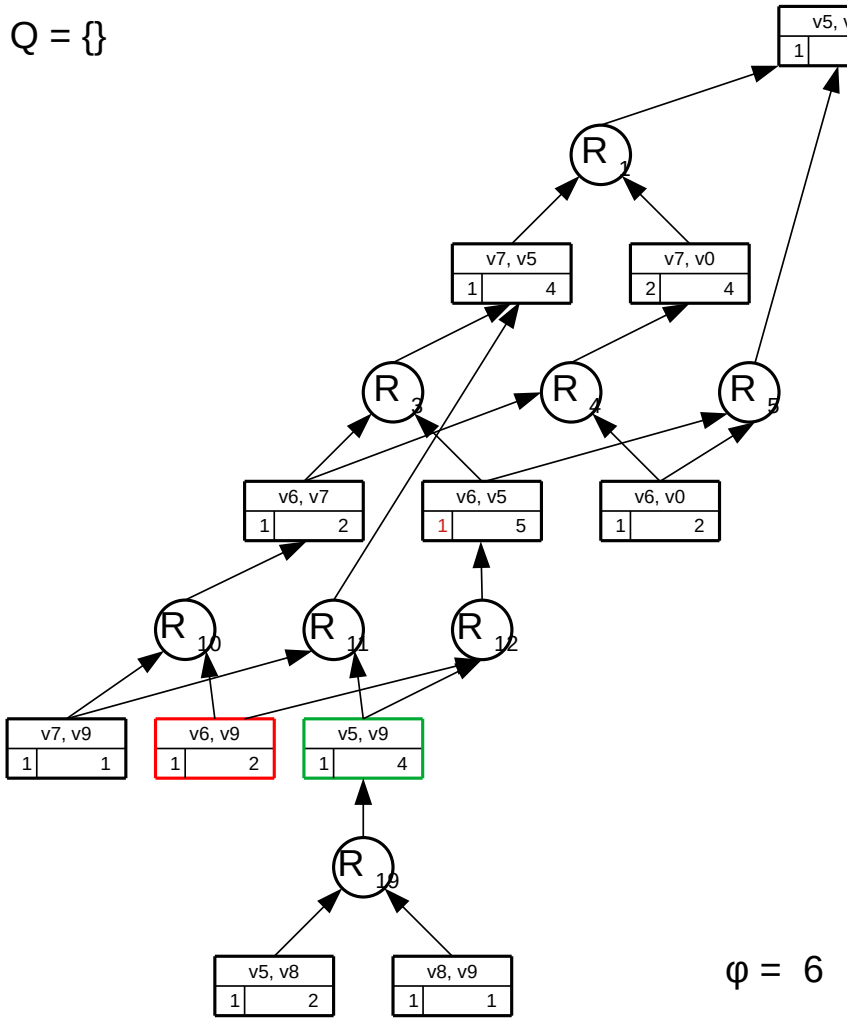
φ = ∞

e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_r) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



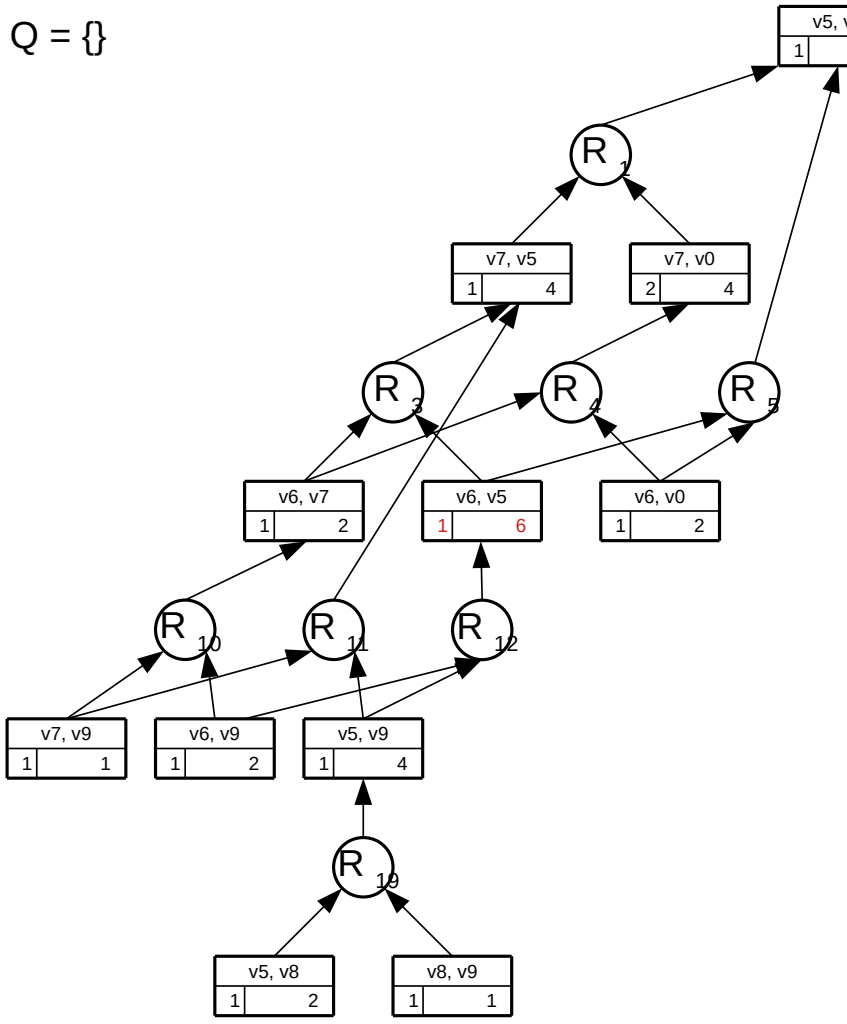
φ = 6

e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_r) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

Q = {}



e = (v6, v9); k = 2

Algorithm 4 DCH$_{scs}$-WInc (G^*, G, e, k)

```
1:  PriorityQueue Q ← ∅;
2:  if φ(v_e, G*) = φ(e, G) then
3:      c_φ(v_e) ← c_φ(v_e) − 1;
4:  φ(e, G) ← k;
5:  if c_φ(v_e) < 1 then
6:      Q.push(v_e);
7:  while Q ≠ ∅ do
8:      v_s ← Q.pop(); updateWeight(G*, v'_s);
9:      for each v_r ∈ nbr+(v_s) do
10:         v'_s ← nbr+(v_r);
11:         if c_φ(v'_s) < 1 then
12:             if v'_s ∉ Q then Q.push(v'_s);
13: procedure updateWeight(G*, v_s)
14:     for each v_r ∈ nbr+(v_s) do
15:         v_s1, v_s2 ← nbr−(v_r); v_s3 ← nbr+(v_r);
16:         if φ(v_s3, G*) = φ(v_s1, G*) + φ(v_s2, G*) then
17:             c_φ(v_s3) ← c_φ(v_s3) − 1;
18:     φ ← φ(s, G); c_φ(v_s) ← 1;
19:     for each v_r ∈ nbr−(v_r) do
20:         v_s1, v_s2 ← nbr−(v_r);
21:         if φ > φ(v_s1, G*) + φ(v_s2, G*) then
22:             φ ← φ(v_s1, G*) + φ(v_s2, G*); c_φ(v_s) ← 1;
23:         else if φ = φ(v_s1, G*) + φ(v_s2, G*) then
24:             c_φ(v_s) ← c_φ(v_s) + 1;
25:     φ(v_s, G*) ← φ;
```

φ = 6