Minimum Spanning Tree on a Planar Graph

Tonomi Matsui

Reviewing the MST problem

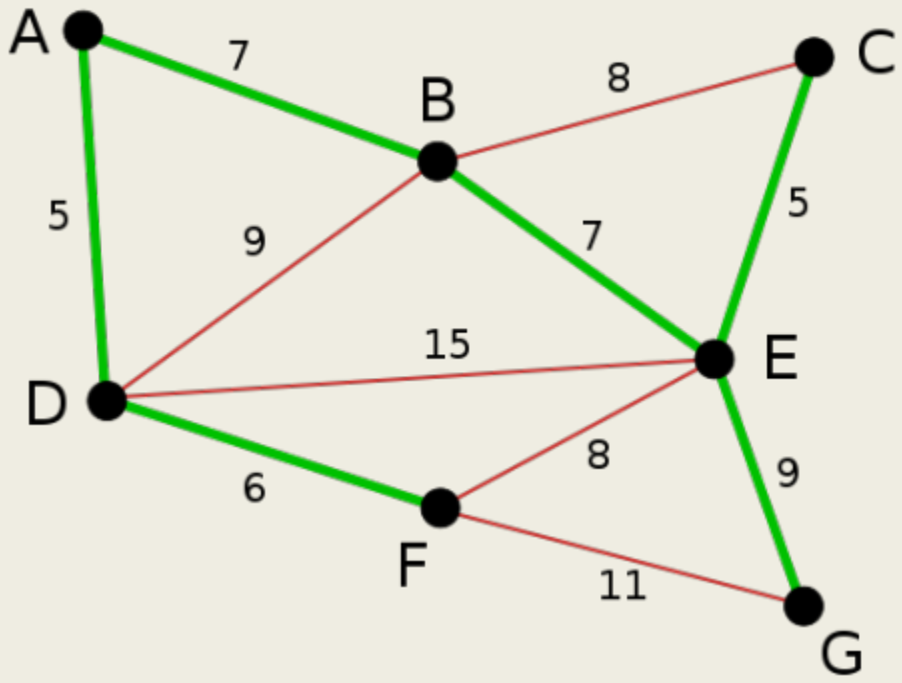
- Tree - Subgraph that is minimally connected (removing any edge will disconnect it);
- Spanning Tree - Subgraph that covers every vertex and is a tree;
- Minimum spanning tree - Spanning tree with the edges with lowest weights;

Reviewing the MST problem

- Forest - Set of trees;

- Maximal spanning forest - Forest that covers every vertex (in a connected graph it is equal to spanning tree).

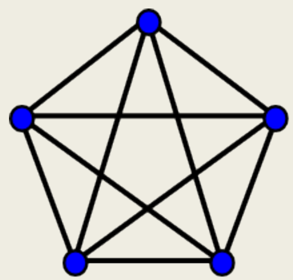
Reviewing the MST problem

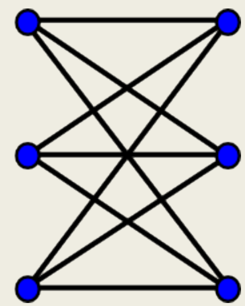


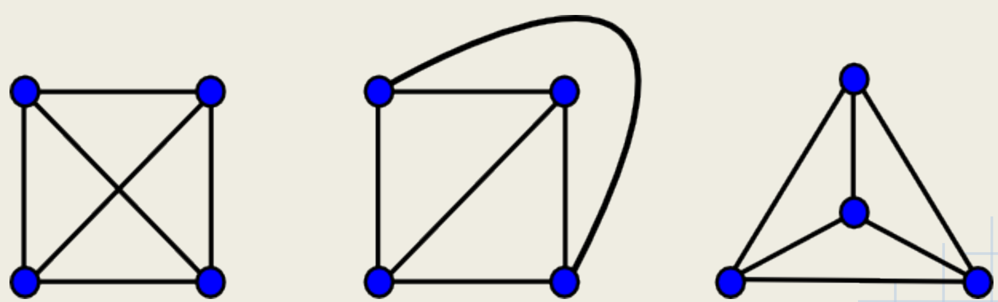
Required concepts

Planar Graph

Graph that can be drawn without the edges crossing each other.







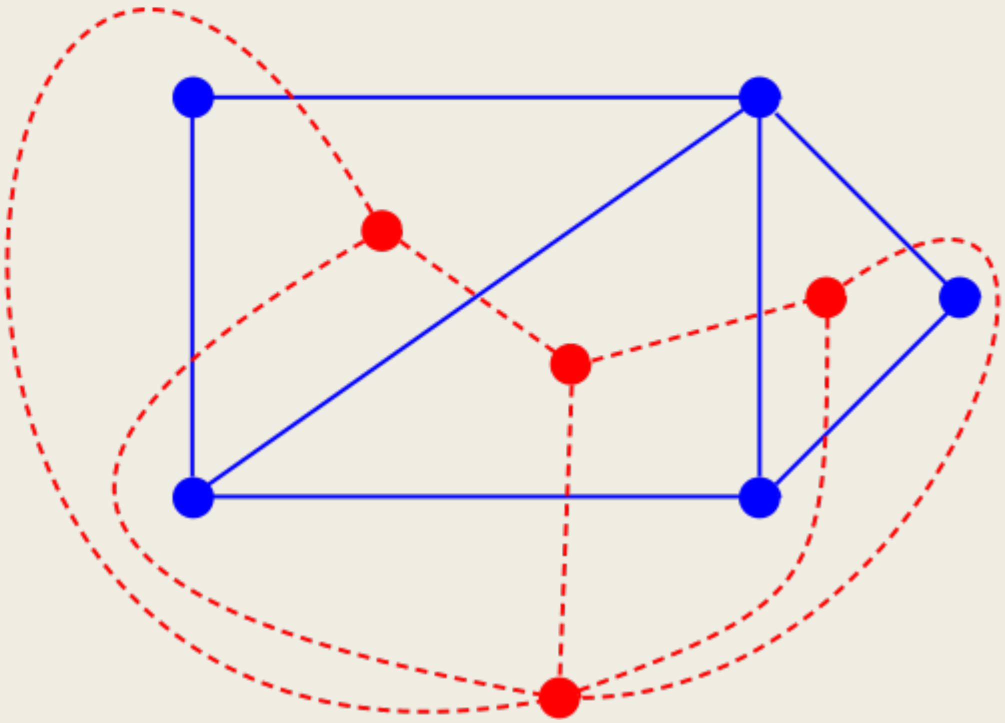
Dual Graph of a planar Graph

- $G^* = (V^*, E)$ is a dual-graph of $G = (V,E)$ when an edge subset $C \subseteq E$ is a cycle of G iff C is a cut-set of G^*;

- Cut-set - Set of edges that will make the graph disconnected if removed, but removing only a subset does not disconnect;

- There are algorithms to find a Dual Graph in linear time.

Dual Graph of a planar Graph

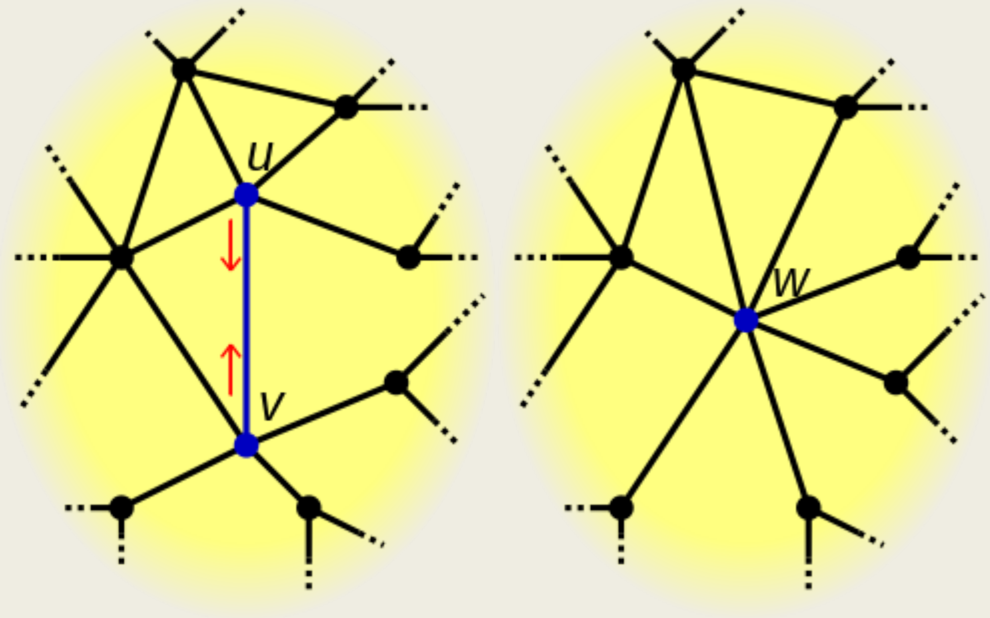


Deletion and contraction

- $G \backslash e$ denotes the graph obtained by deleting the edge e from G;

- G/e denotes the graph obtained by contracting e:
  - Contracting: Remove the edge and merge the vertexes it was connecting.

Deletion and contraction



The Algorithm

The Algorithm

- An edge subset $T \subseteq E$ is a maximal spanning forest of G iff $E \setminus T$ is a maximal spanning forest of G^*;

- If T is the minimum weight spanning forest of G, then $E \setminus T$ is the maximum weight spanning forest of G^* and vice-versa.

The Algorithm

```
Let T and T* := null
Let G1 := G and G1* := G*
While true
   if G1 and G1* are empty then return [T, T*]
   v := vertex in G1 or G1*
```

The algorithm

```
if v contains a self-loop
   f  := edge with the self-loop
   G1 := G1\f, G1* := G1*\f
   if v belongs to G1  then T* := T*U{f}
   if v belongs to G1* then T  := T U{f}
```

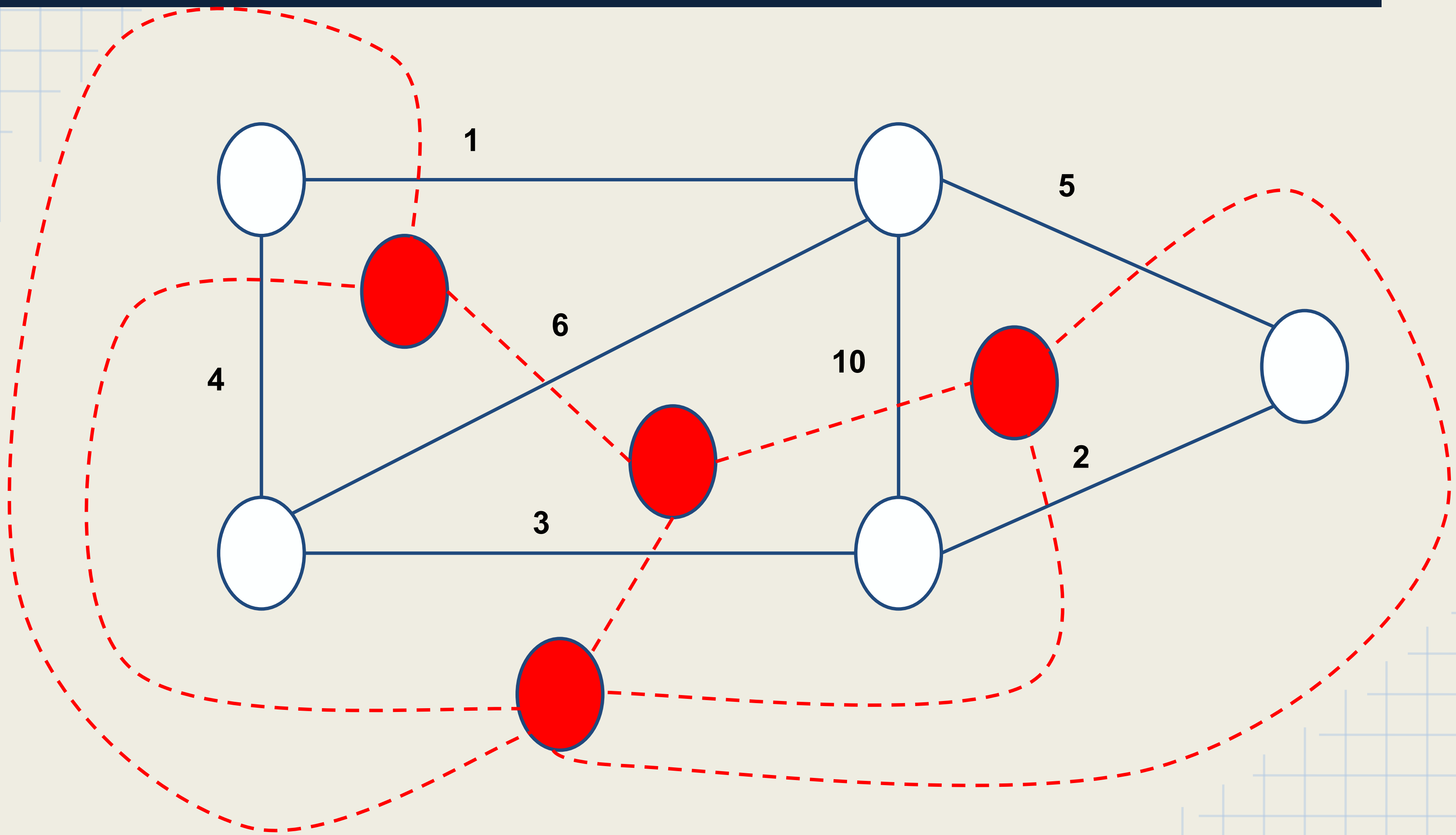
The algorithm

```
if v contains no self-loop and belongs to G1
   I   := set of edges incident on v
   e   := edge in E with min weight
   G1  := G1/e
   G1* := G1*\e
   T   := T∪{e}
```

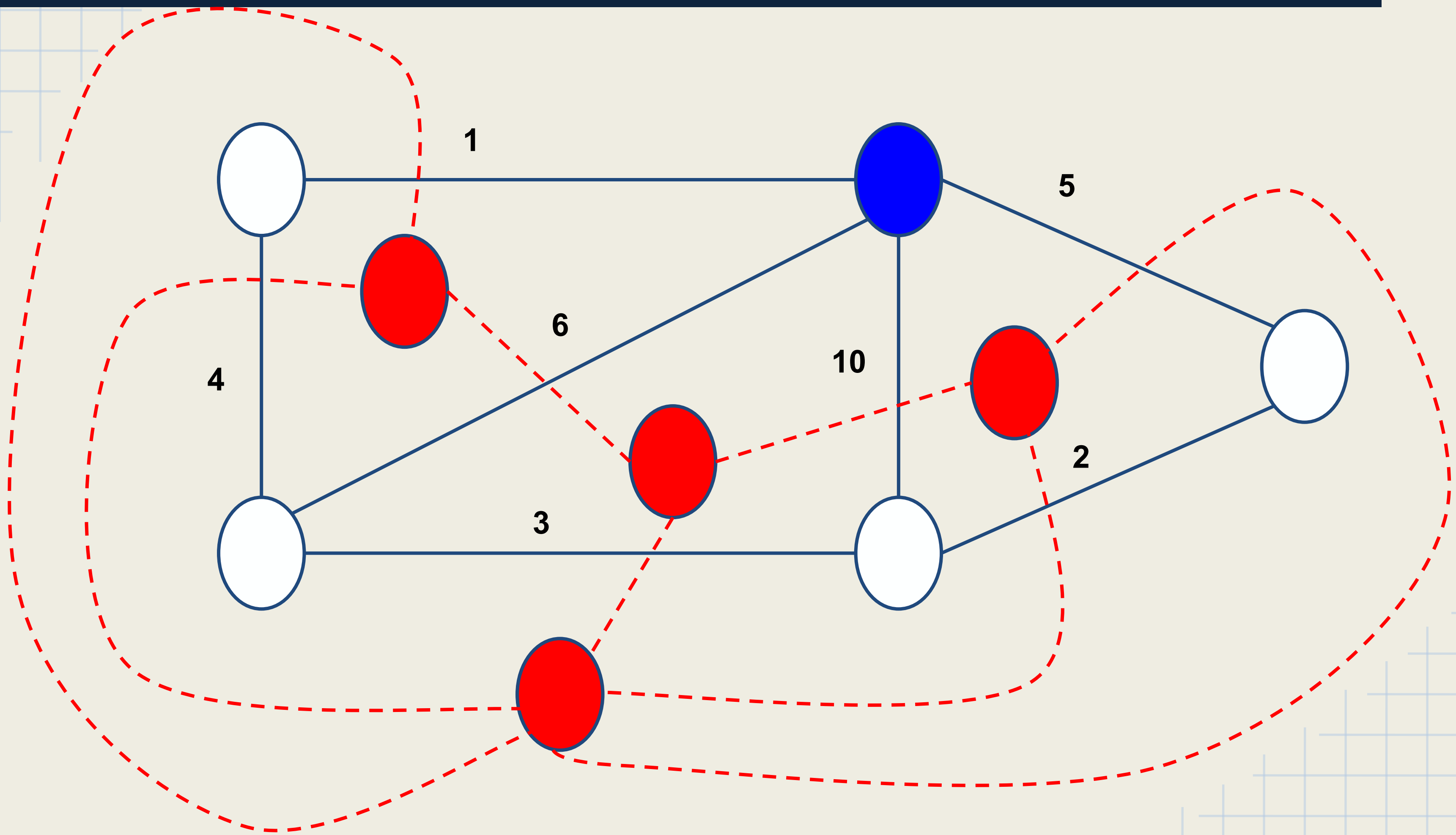
The algorithm

```
if v contains no self-loop and belongs to G1*
   I    := set of edges incident on v
   e    := edge in E with max weight
   G1   := G1\e
   G1*  := G1*/e
   T*   := T*U{e}
```

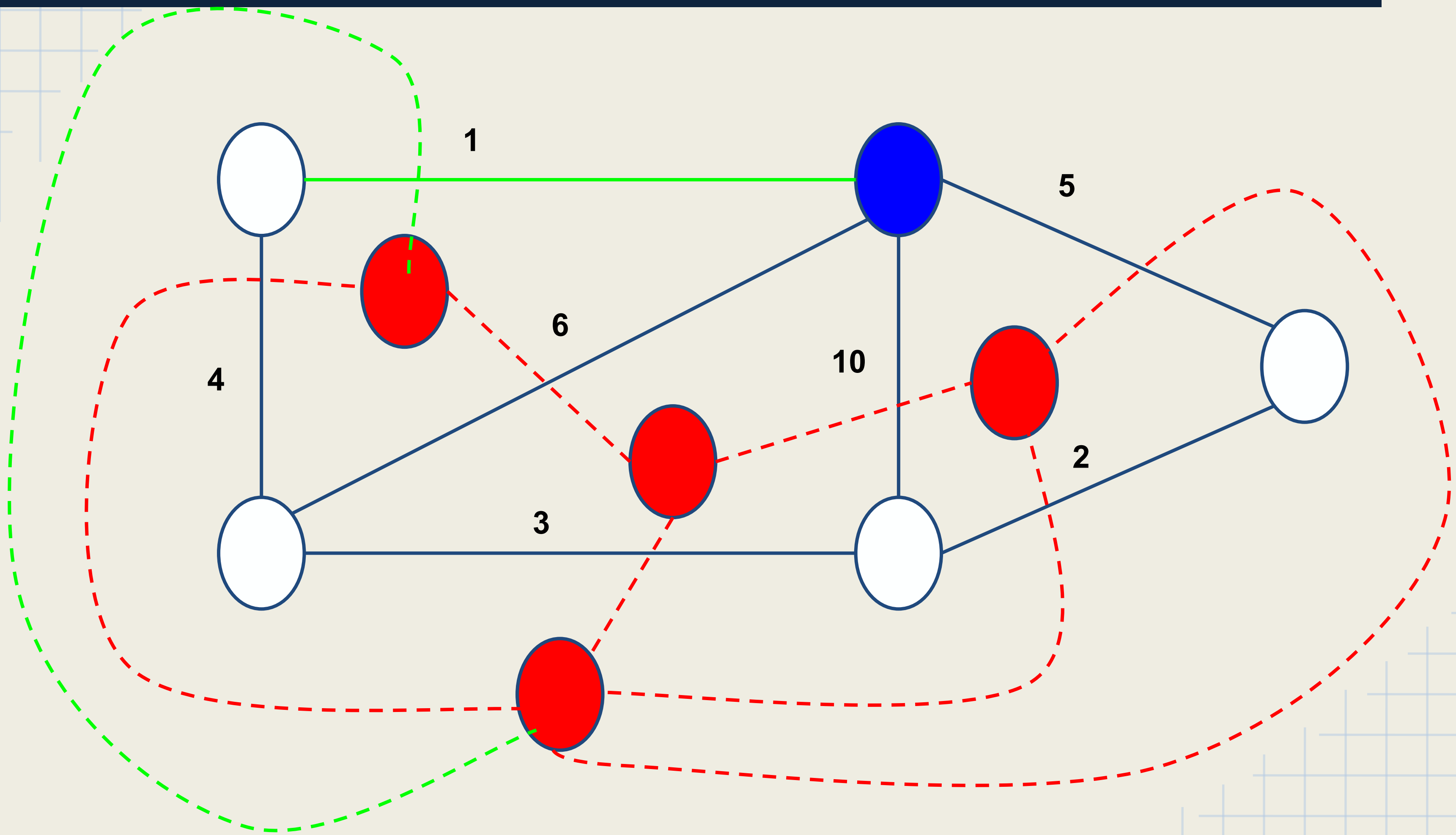
The algorithm - example



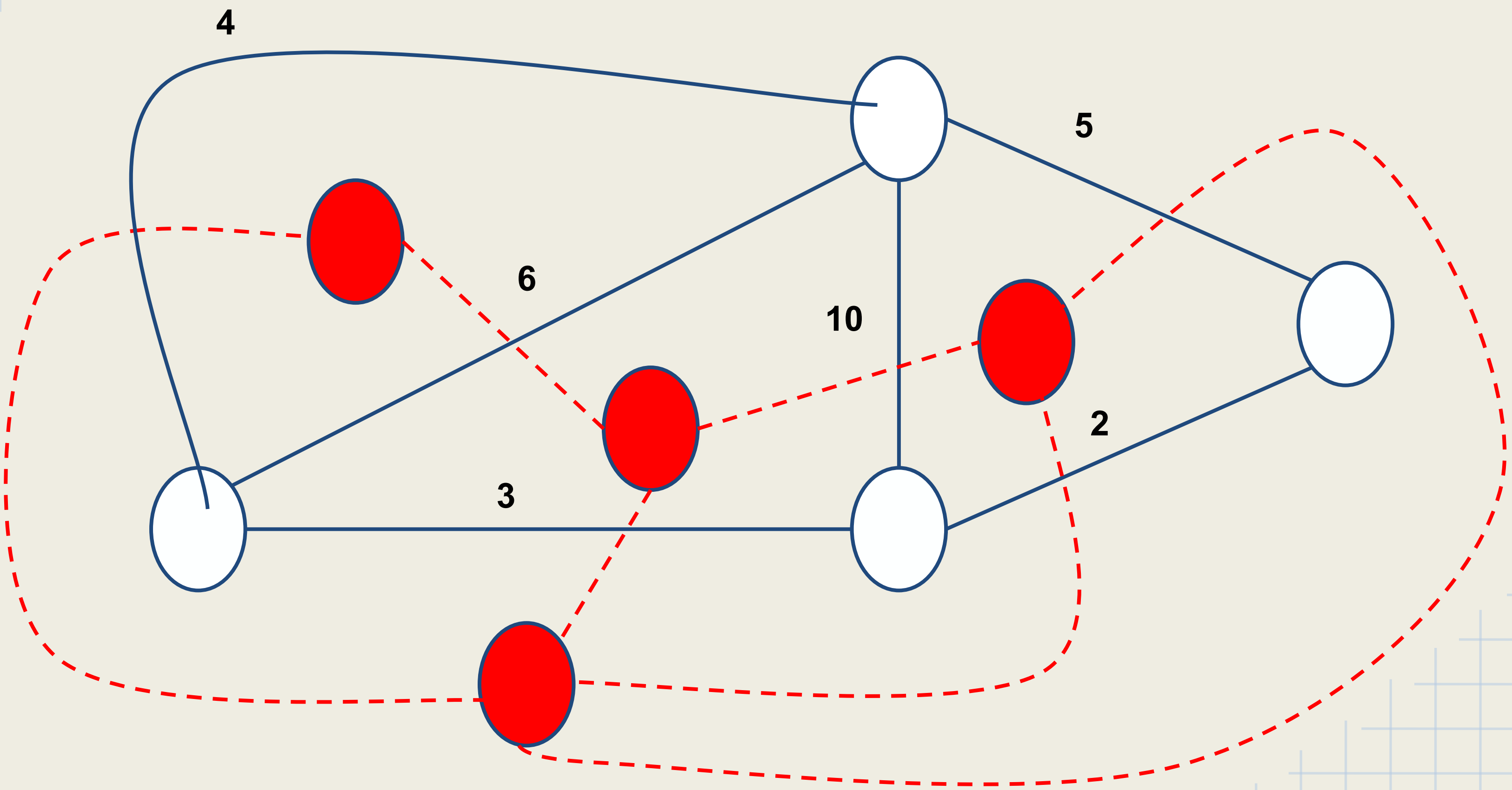
The algorithm - example



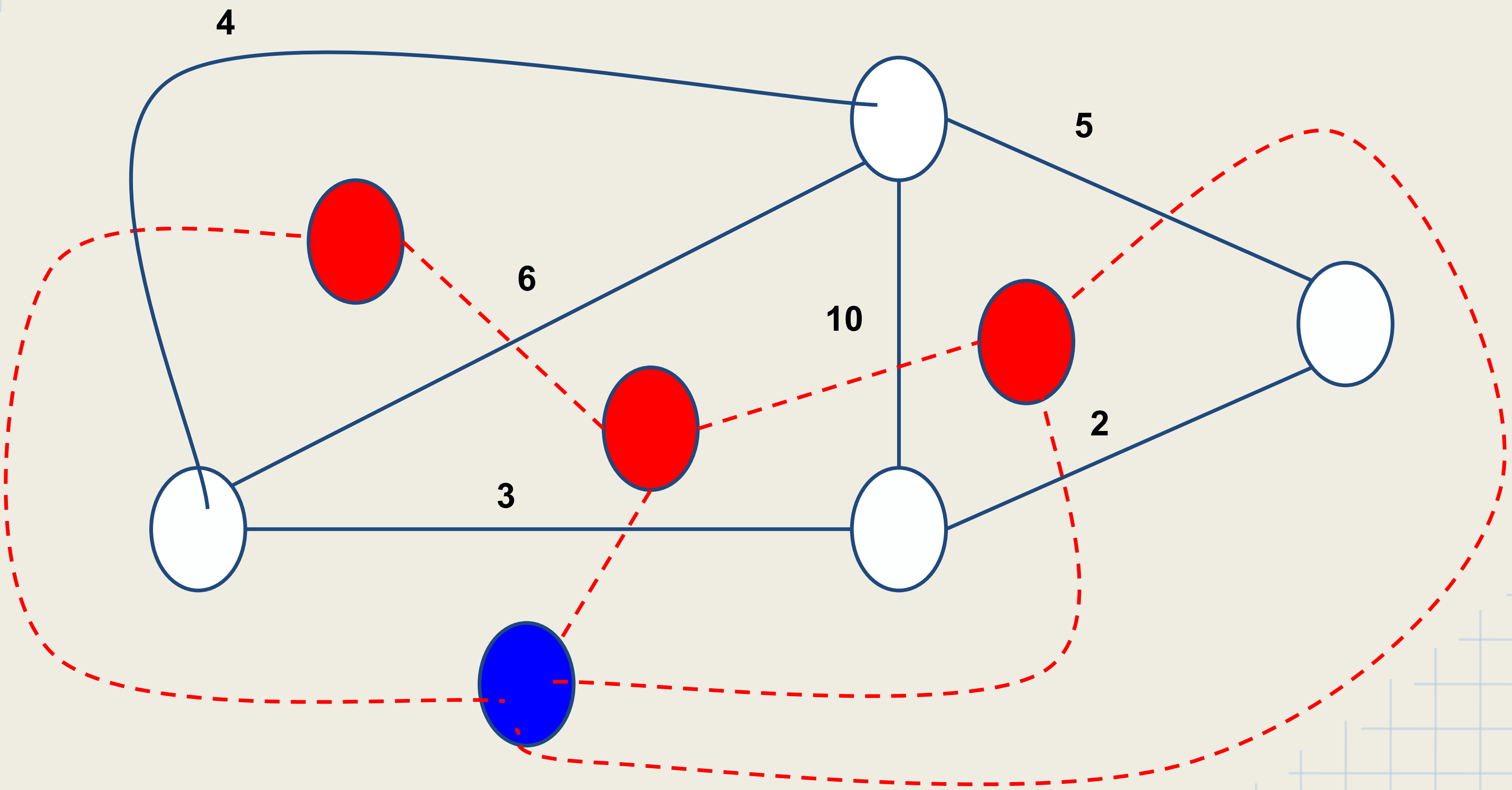
The algorithm - example



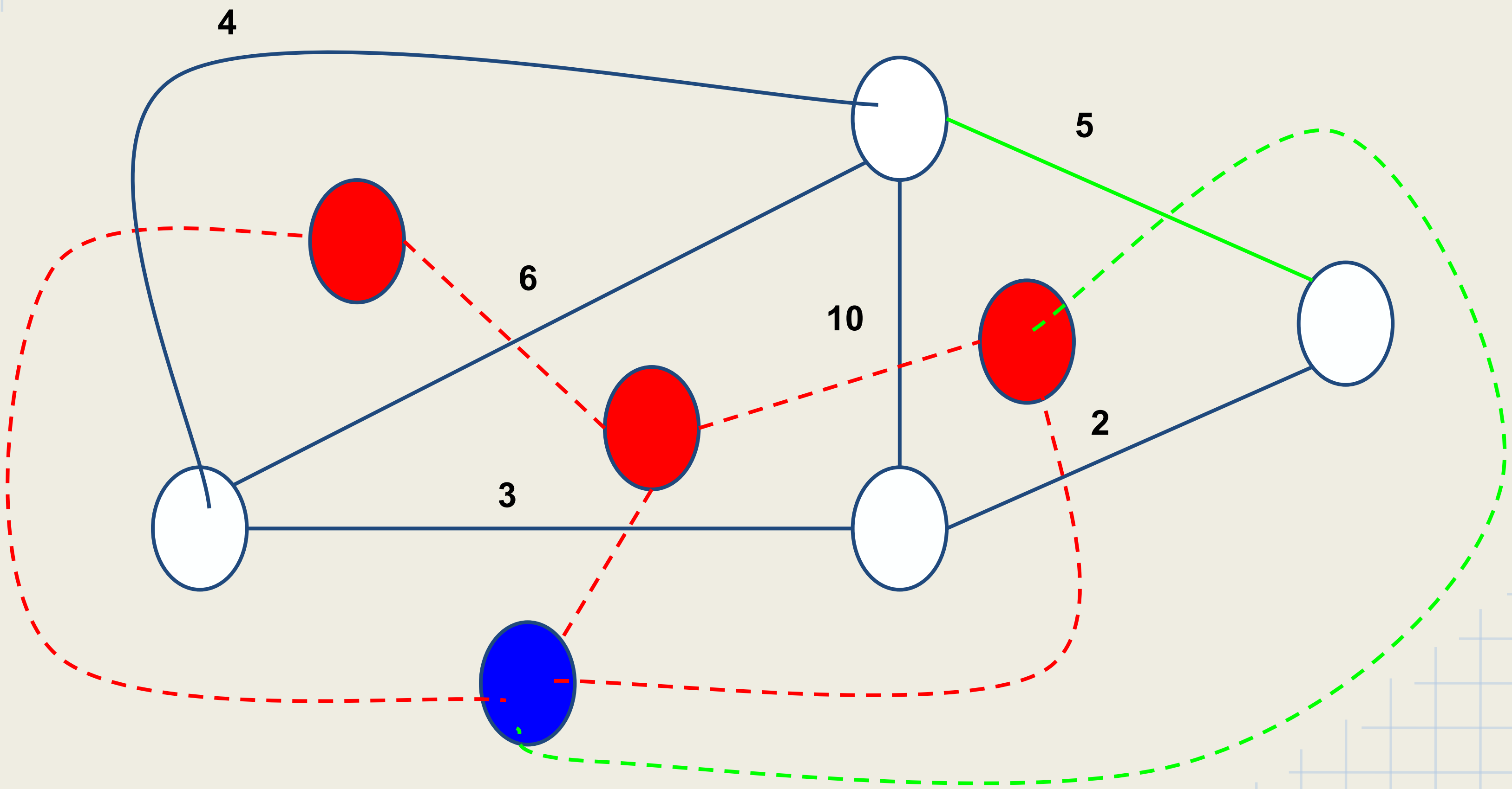
The algorithm - example



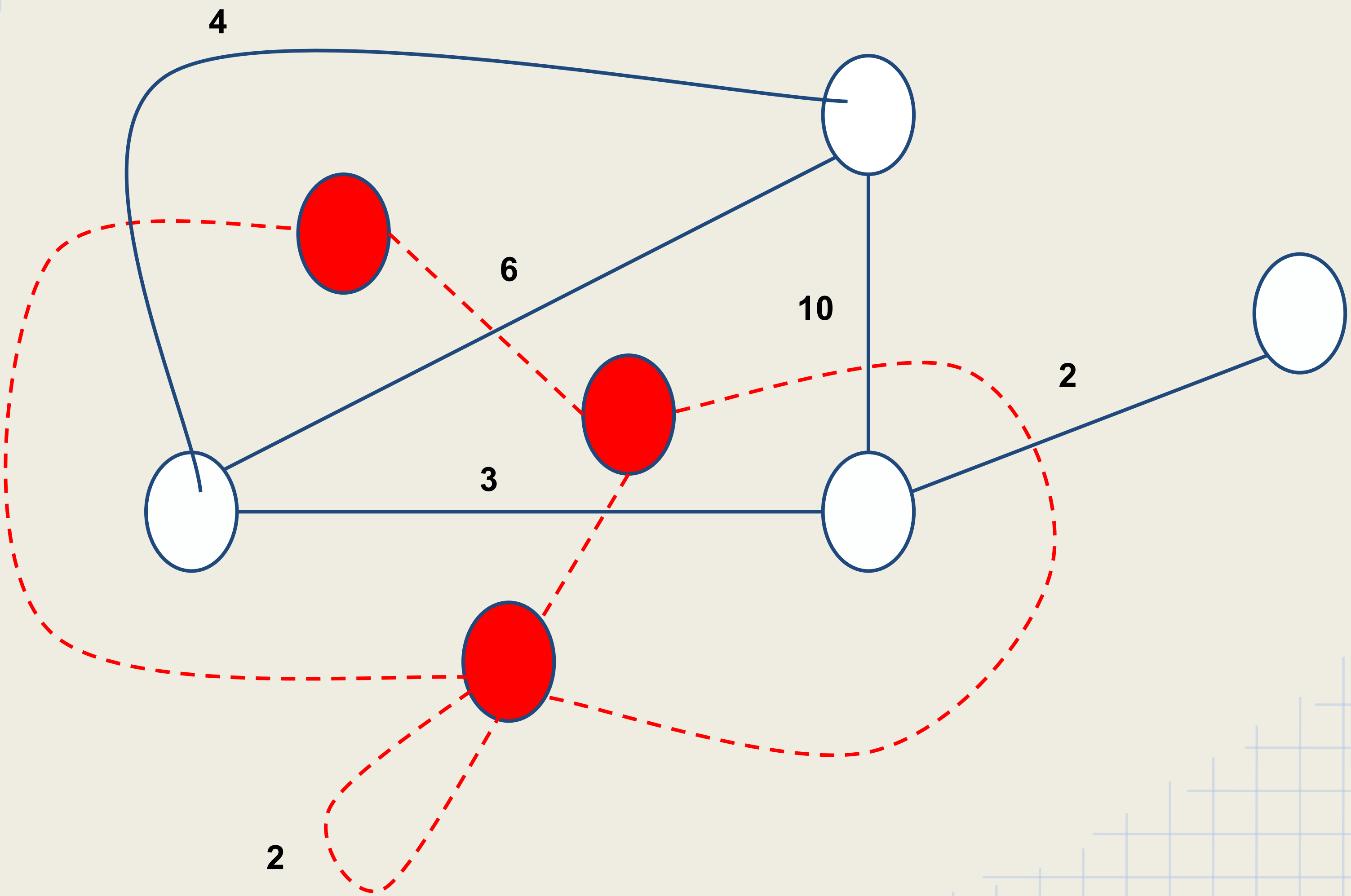
The algorithm - example



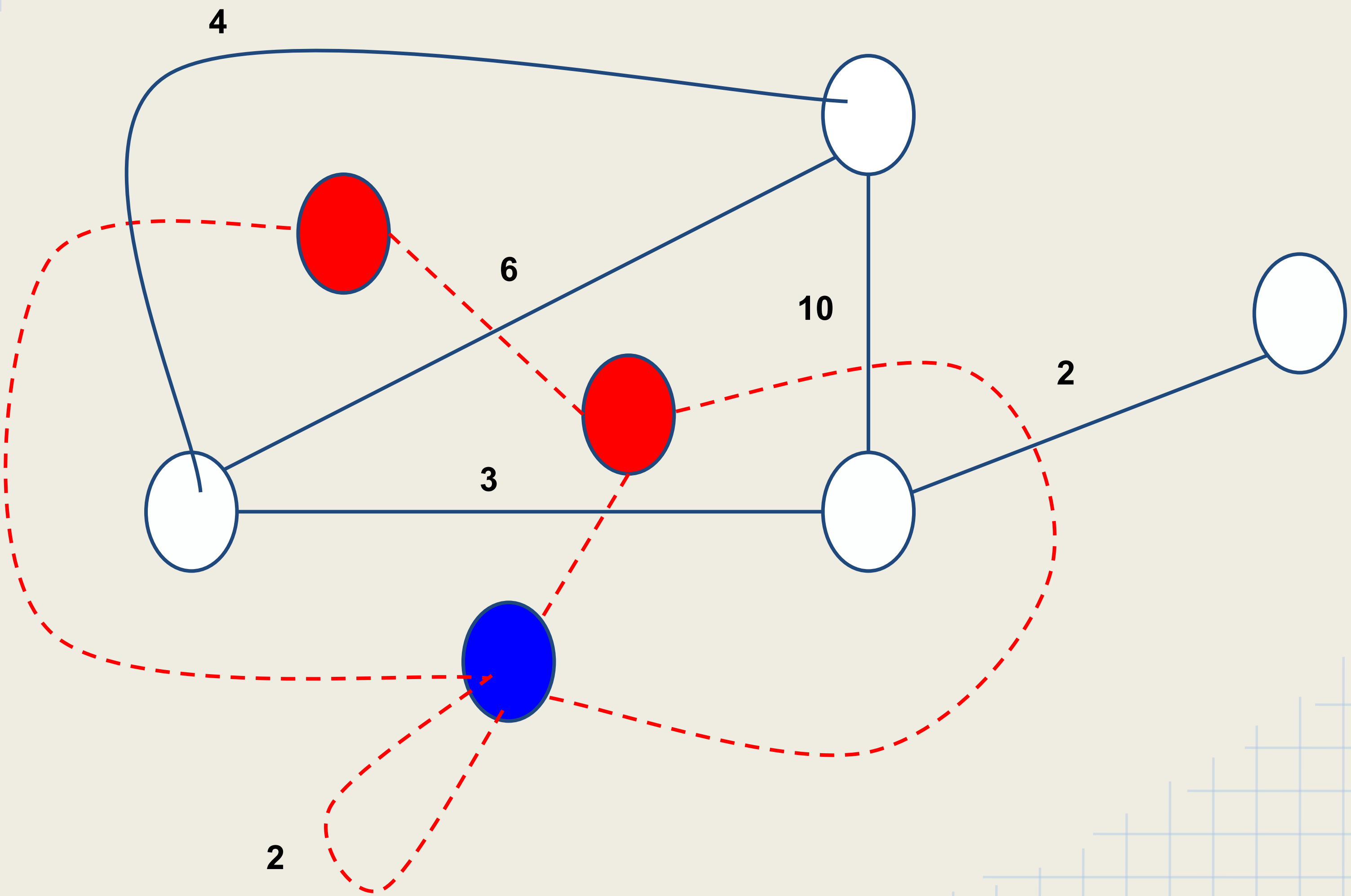
The algorithm - example



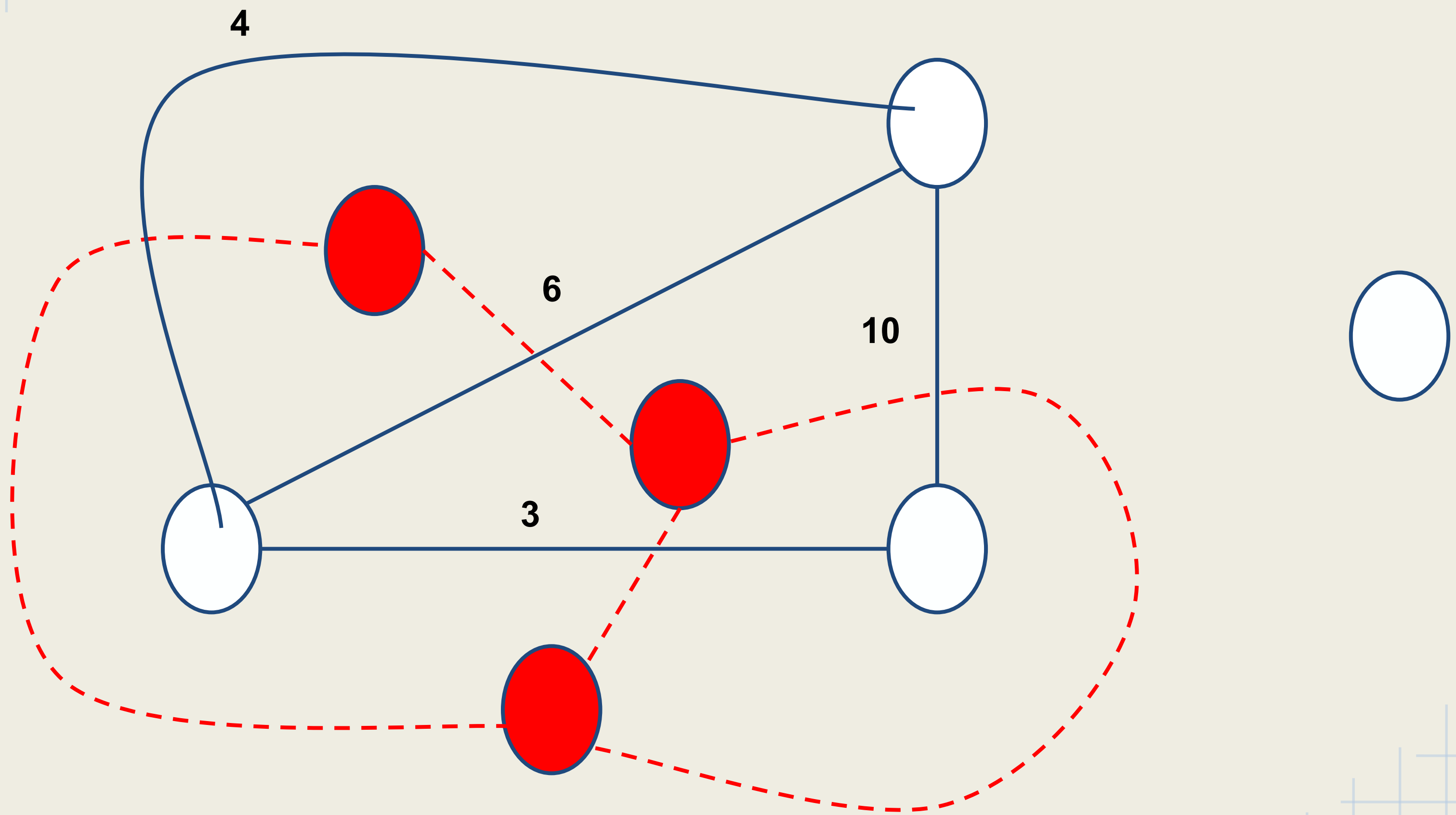
The algorithm - example



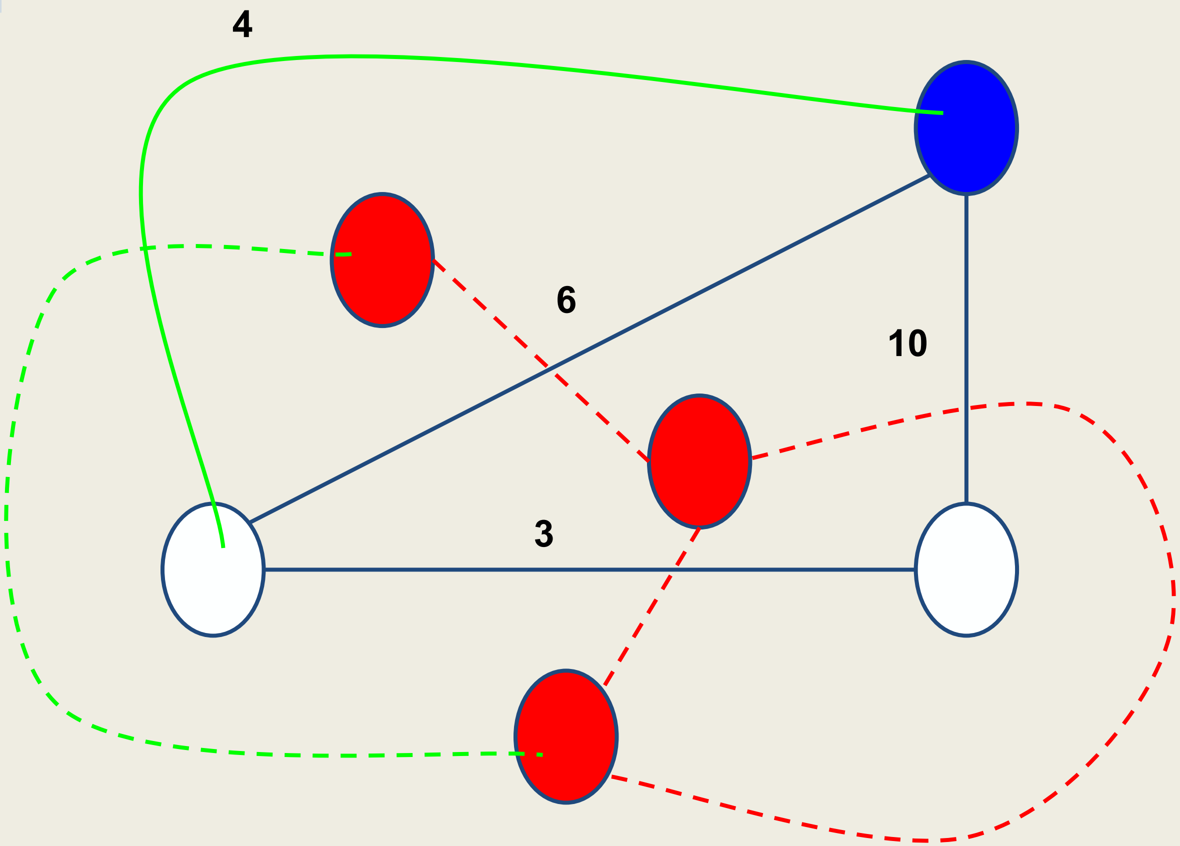
The algorithm - example



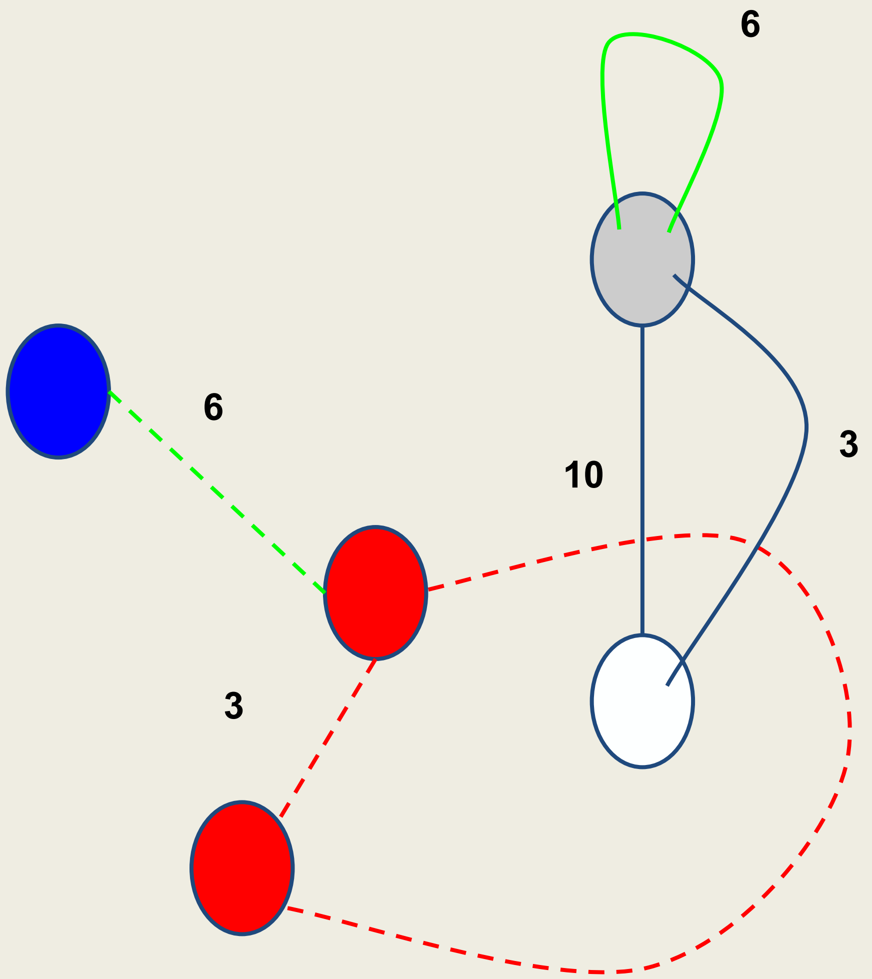
The algorithm - example



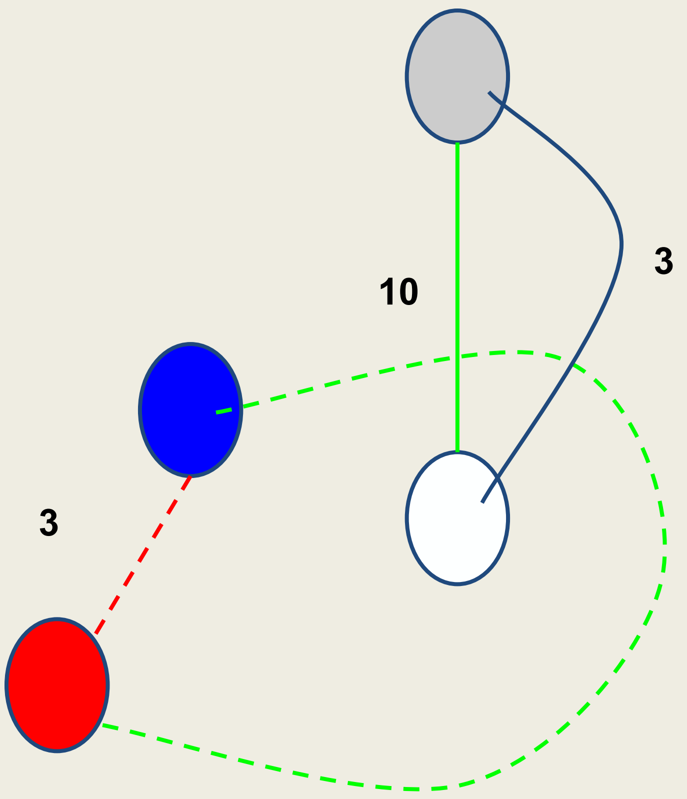
The algorithm - example



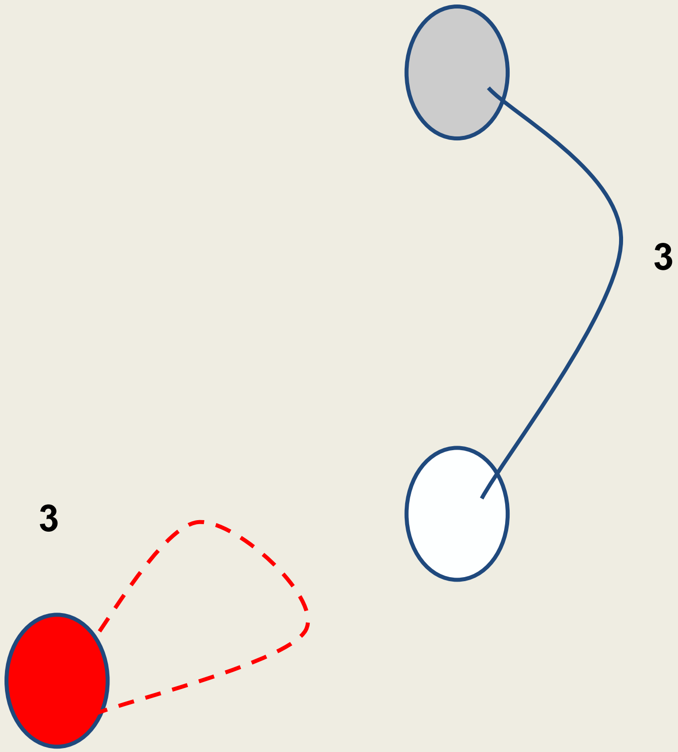
The algorithm - example



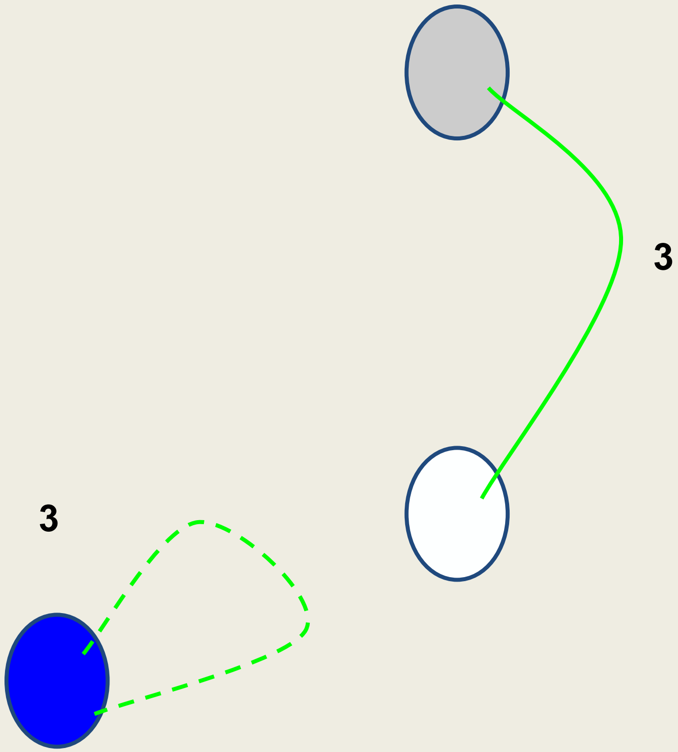
The algorithm - example



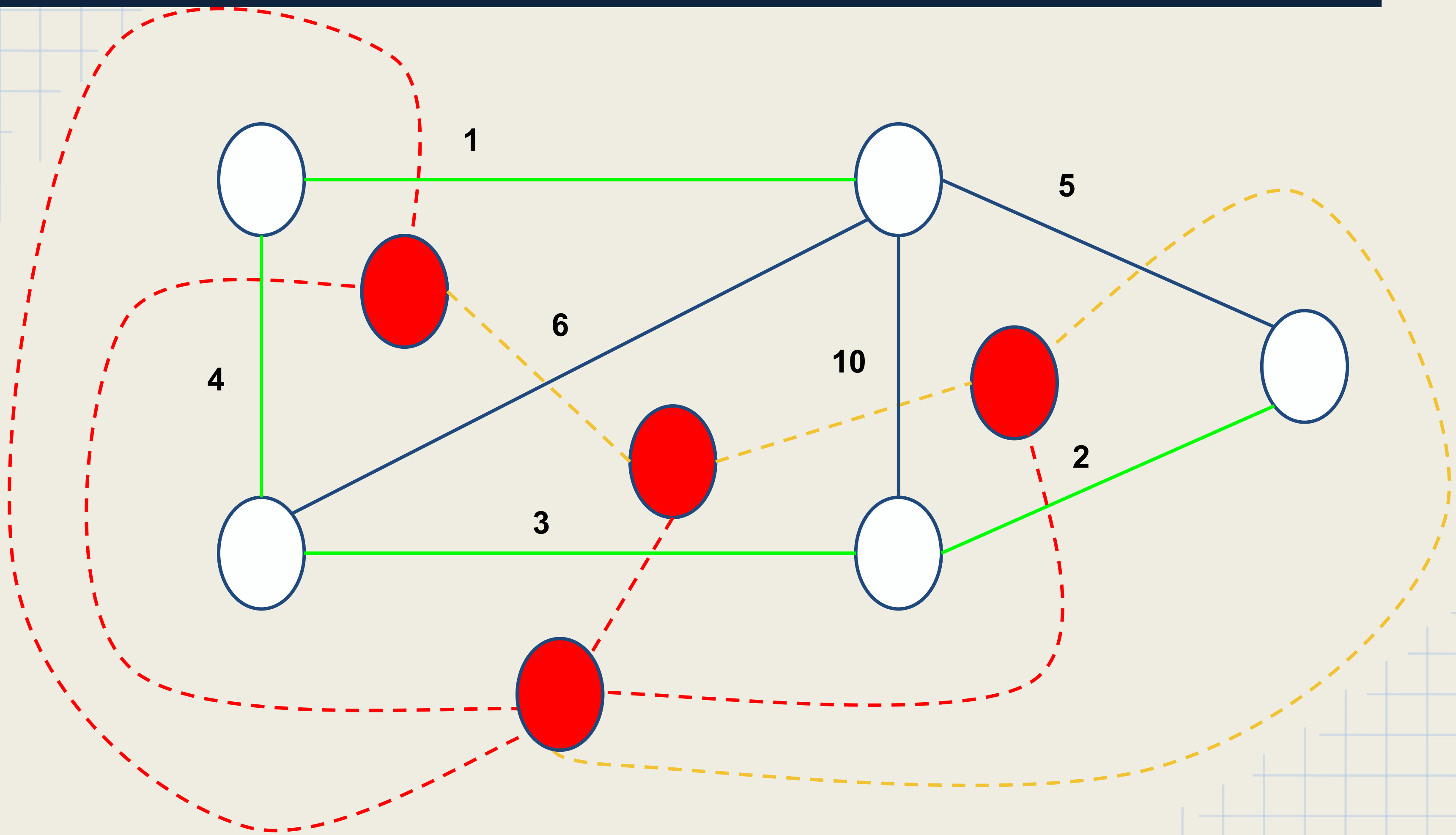
The algorithm - example



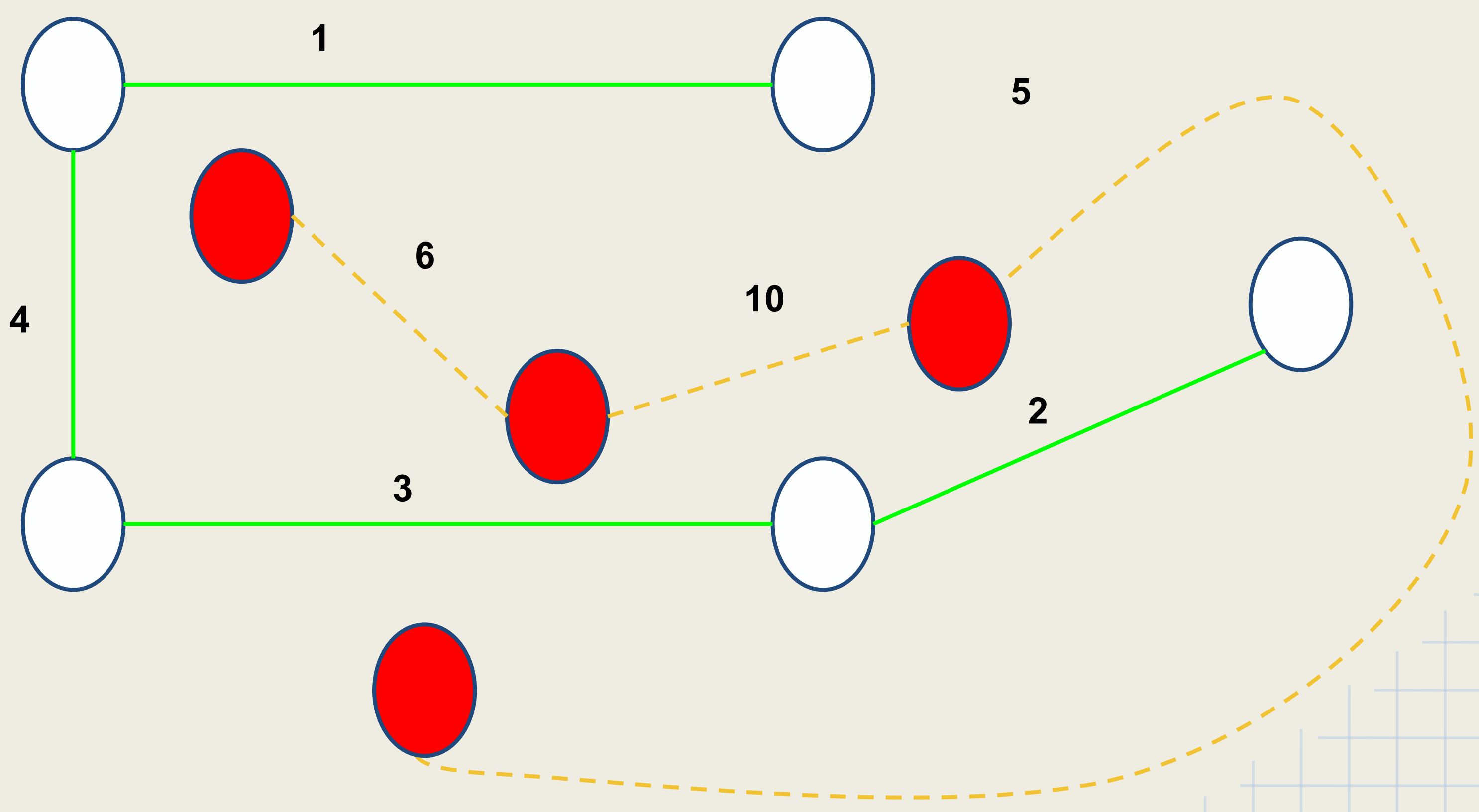
The algorithm - example



The algorithm - example



The algorithm - example



Performance

Number of iterations

- The number of iterations is at worst $|V| + |V*| + |E|$
- Simple to prove: In each iteration, either an edge or a vertex is removed

Time for each iteration

- Let us assume the graphs are maintained by adjacency lists:
  - Verifying if the graph is empty can be done in constant time;
  - Chosing a vertex can be done in constant time;
  - Deleting an edge can be done in constant time.

- What about:
  - Choosing a min weight edge if a vertex has no self-loops;
  - Contracting an edge.

Optimizing the algorithm

If we chose a vertex with degree smaller than 4:

- Choosing the edge with *min/max weight* requires constant time;
- Contracting the edge requires moving of edges from one vertex to another: $O(min\{d_{G1}(v), d_{G1}(u)\}) = O(3) = O(1)$
- Euler's Formula guarantees that either $G1$ or $G1^*$ has at least one vertex with degree smaller than 4.

Optimizing the algorithm

How to guarantee you choose the right vertex?

- Keep the vertices with degree smaller than 4 in a container;

- When an edge is deleted, verify the cardinalities of the affected vertices and put them in the container if necessary;

- When an edge is contracted, remove the affected vertices from the container, add the new vertex if the cardinality is less than 4.

Conclusion

This algorithm:

- Chooses a vertex $O(1)$;
- If it has a self-loop it deletes an edge $O(1)$;
- Otherwise it chooses an edge with min/max $O(3)$ weight and:
  - Performs a deletion $O(1)$;
  - Performs a contraction $O(3)$.

- Performs $|V| + |V^*| + |E|$ steps in $O(1)$, time, therefore its complexity is $O(|V| + |V^*| + |E|)$
- Better than other algorithms, which perform in $O(mlog(n))$

Questions?

?