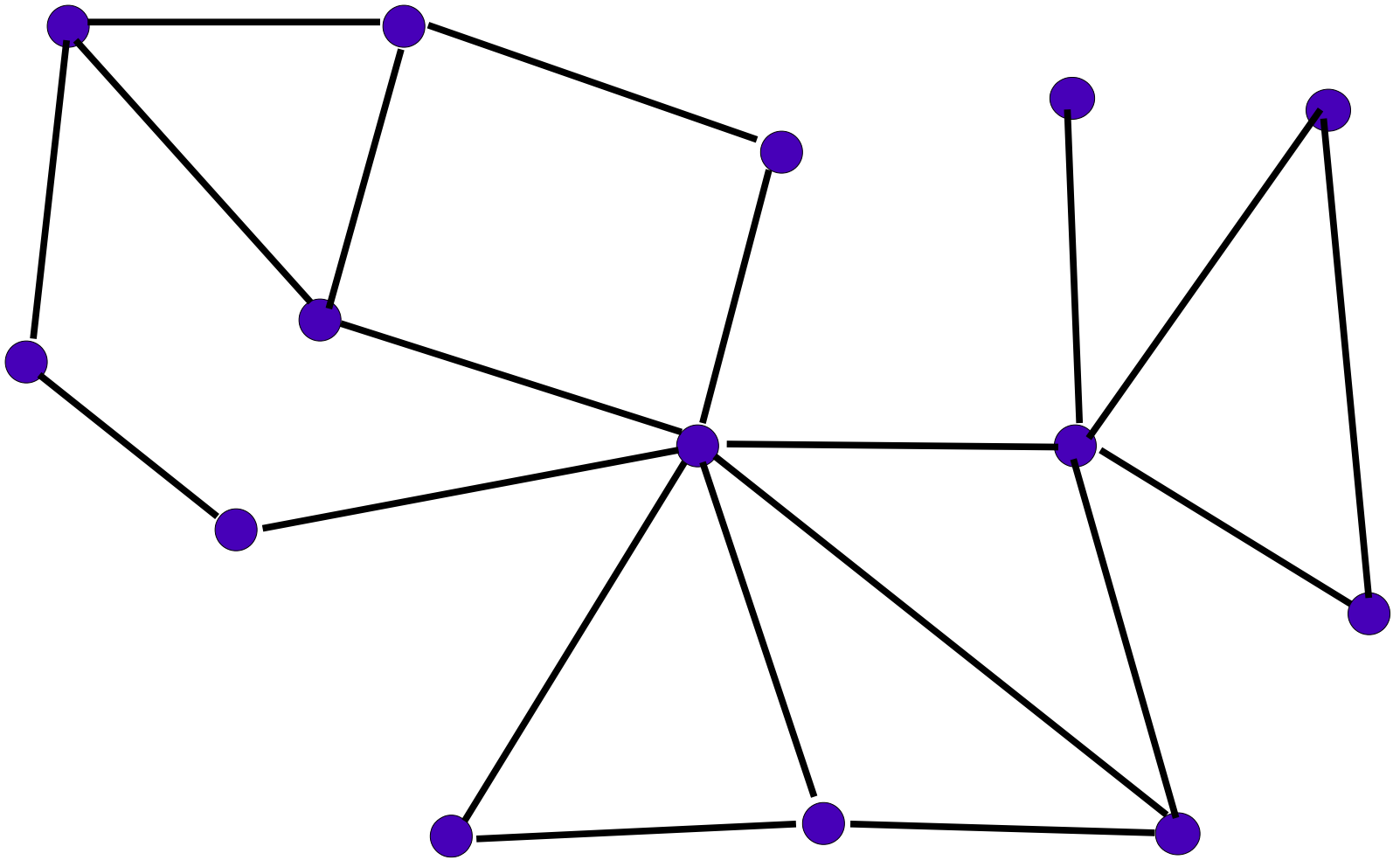
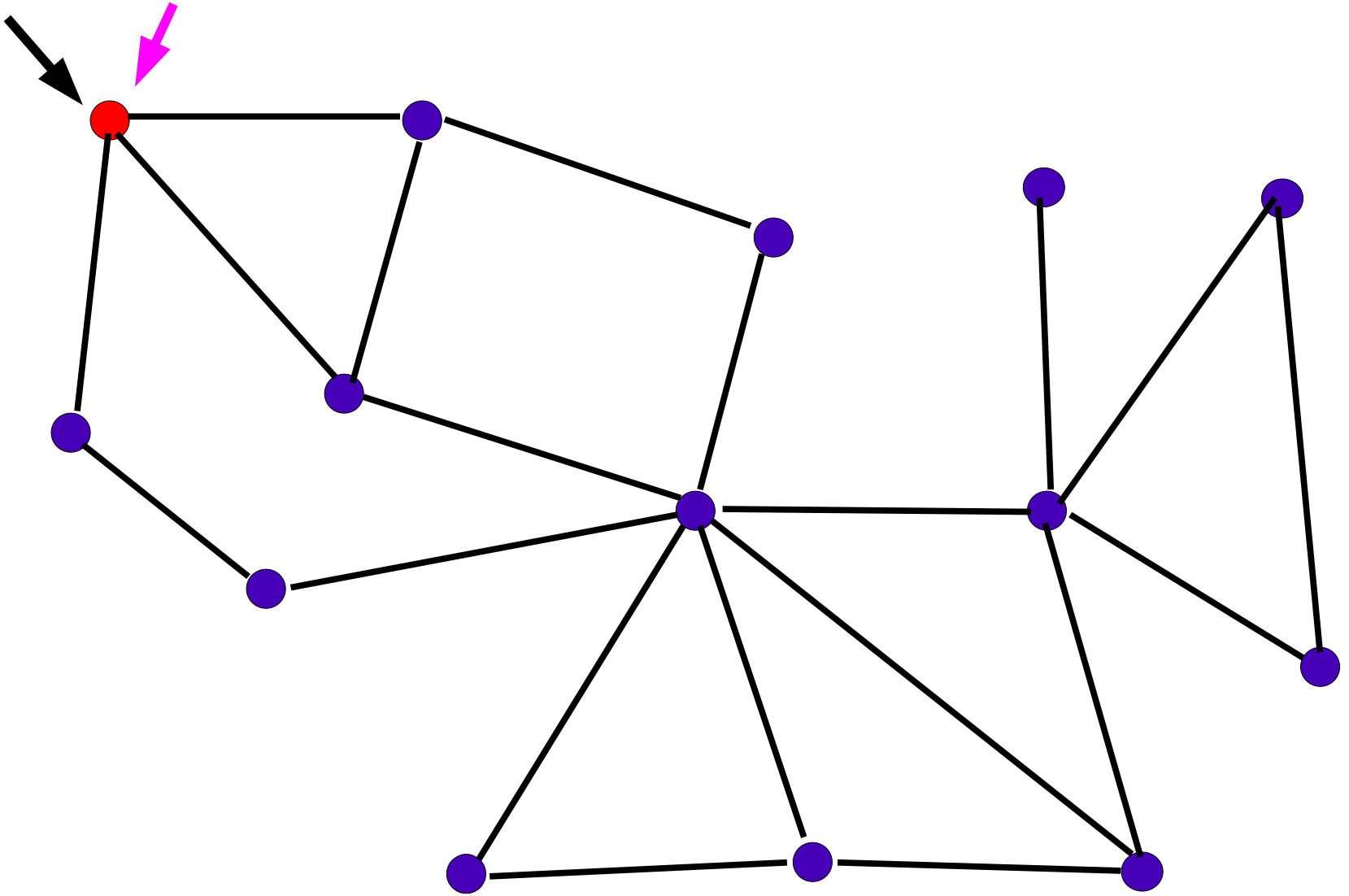
Biconnected Components

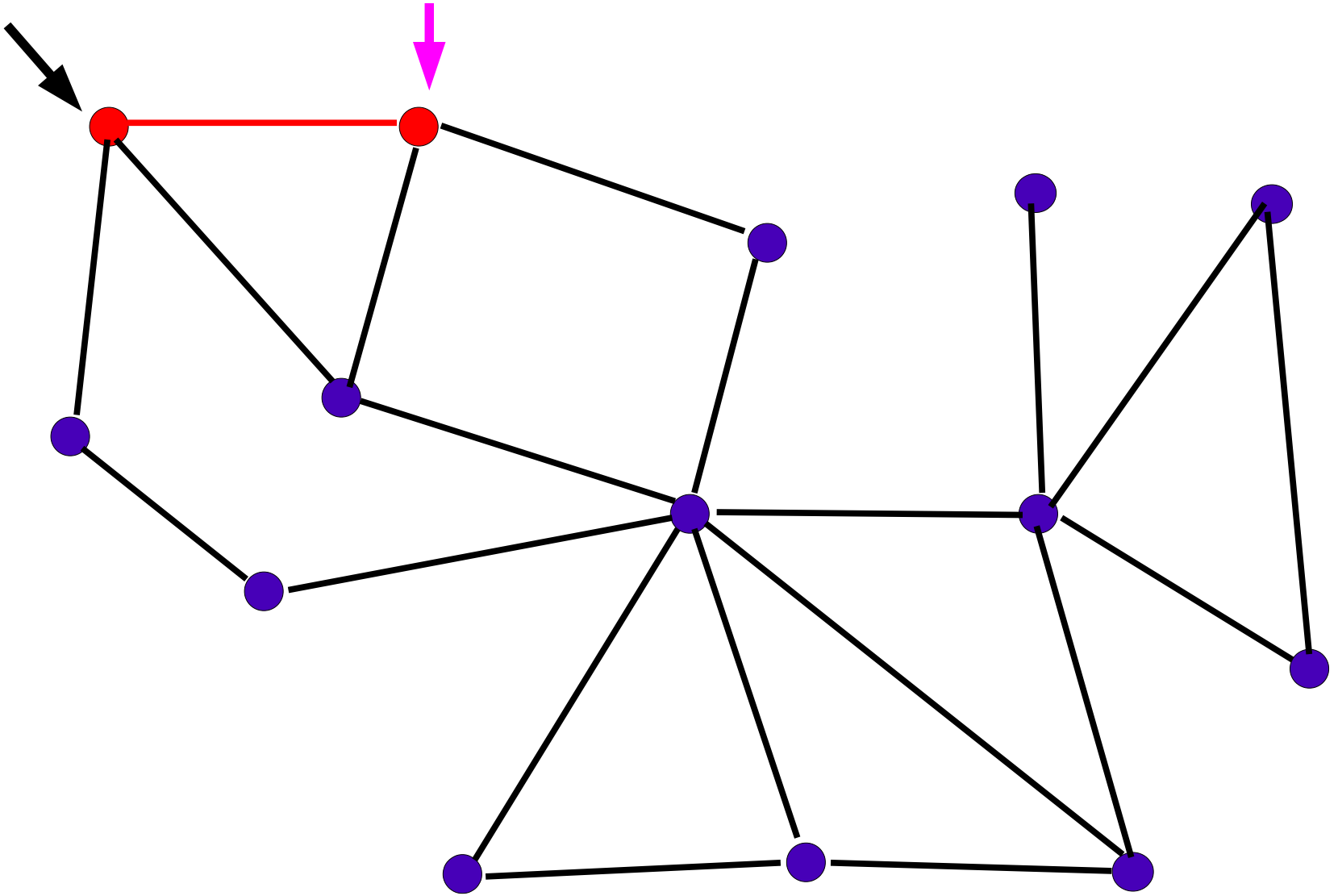
Biconnected Components



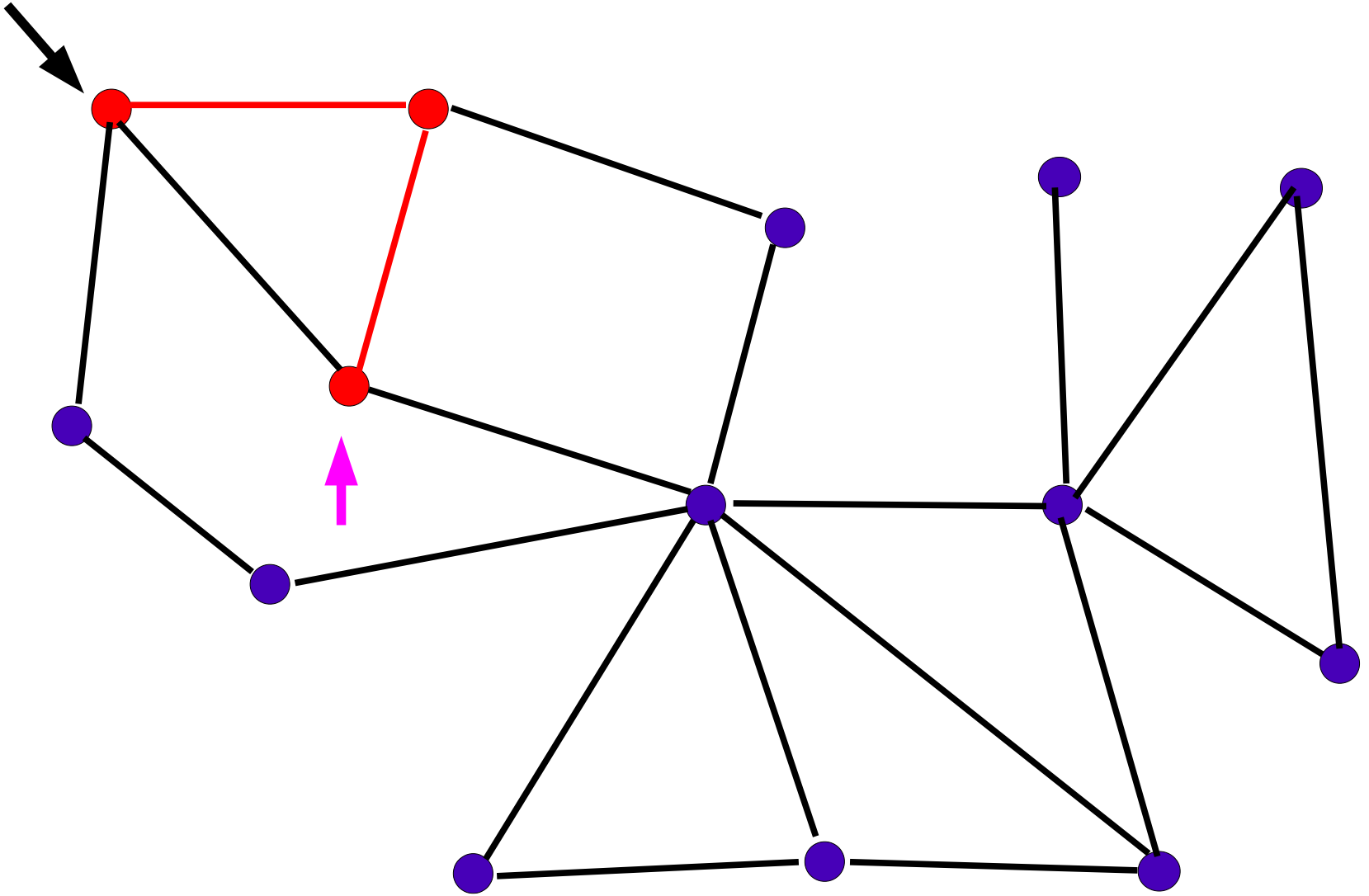
Biconnected Components



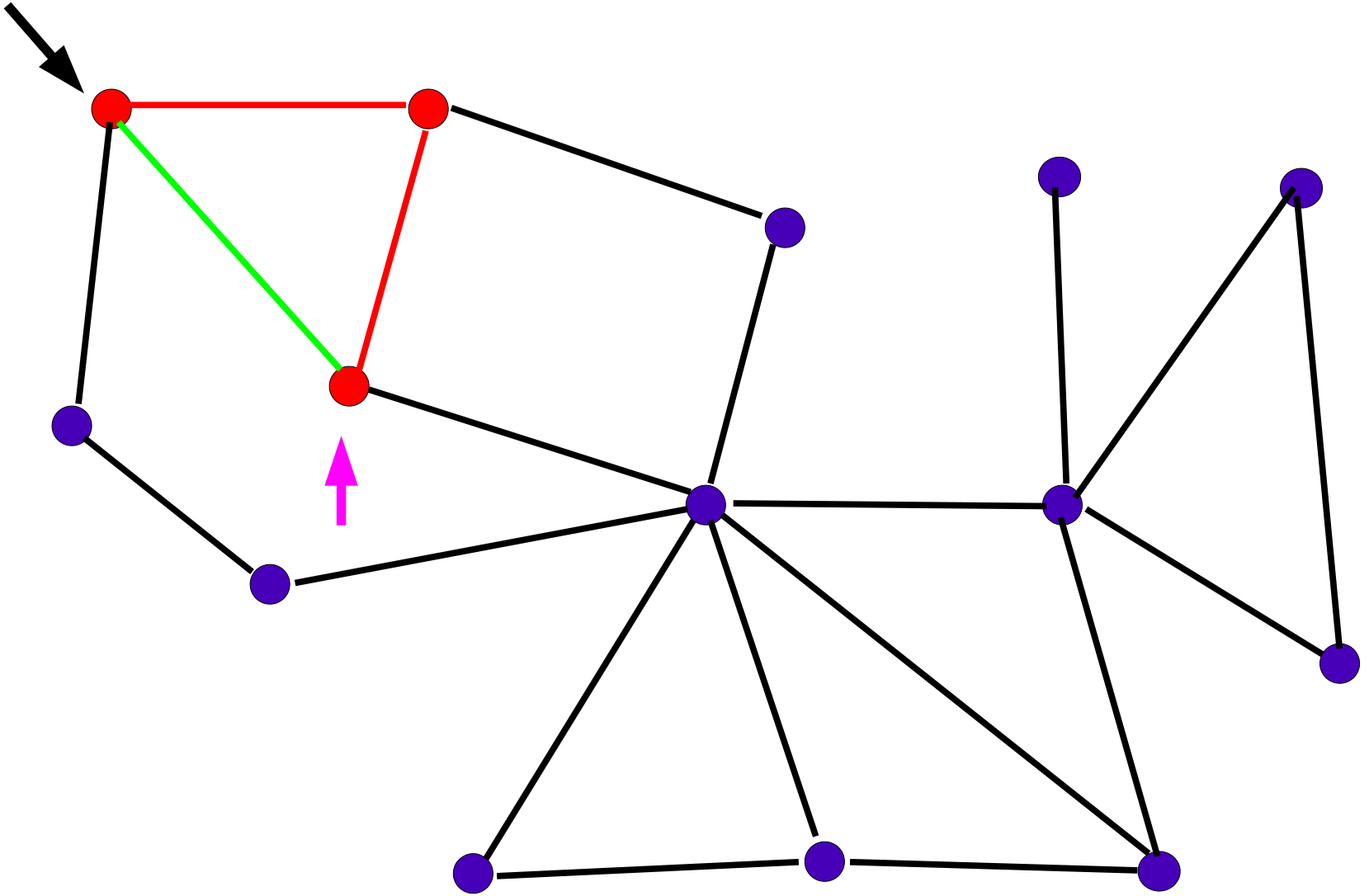
Biconnected Components



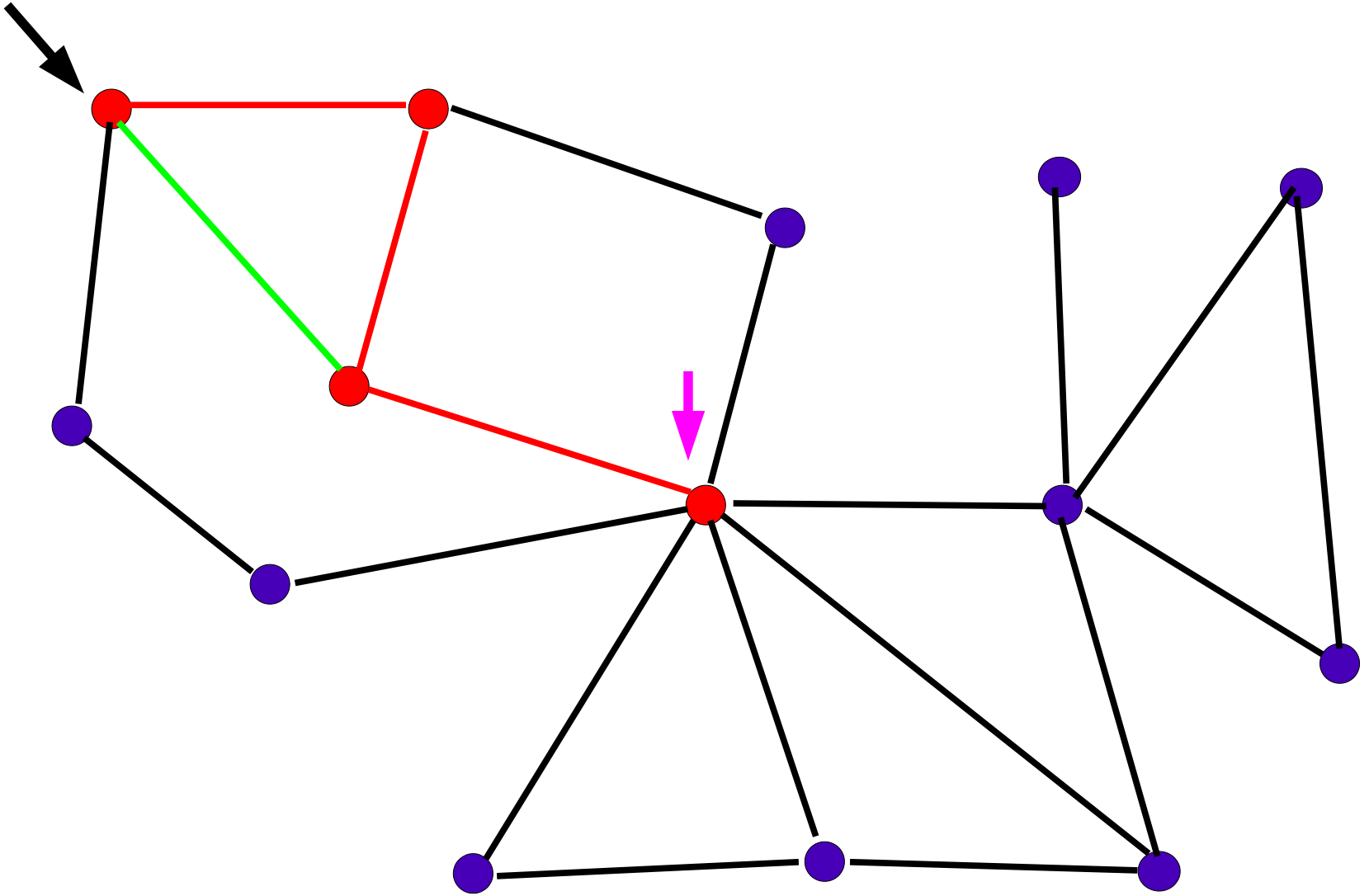
Biconnected Components



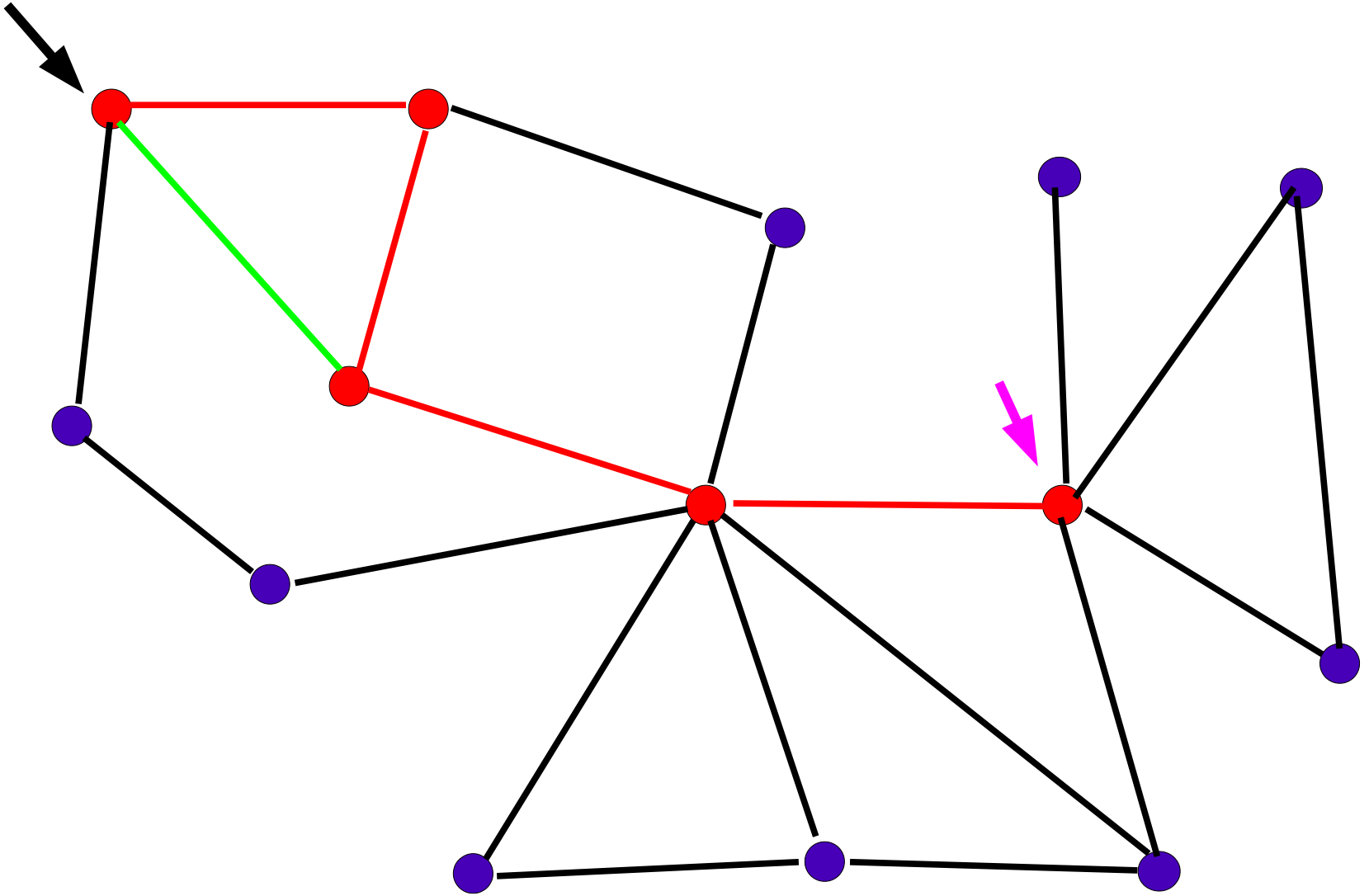
Biconnected Components



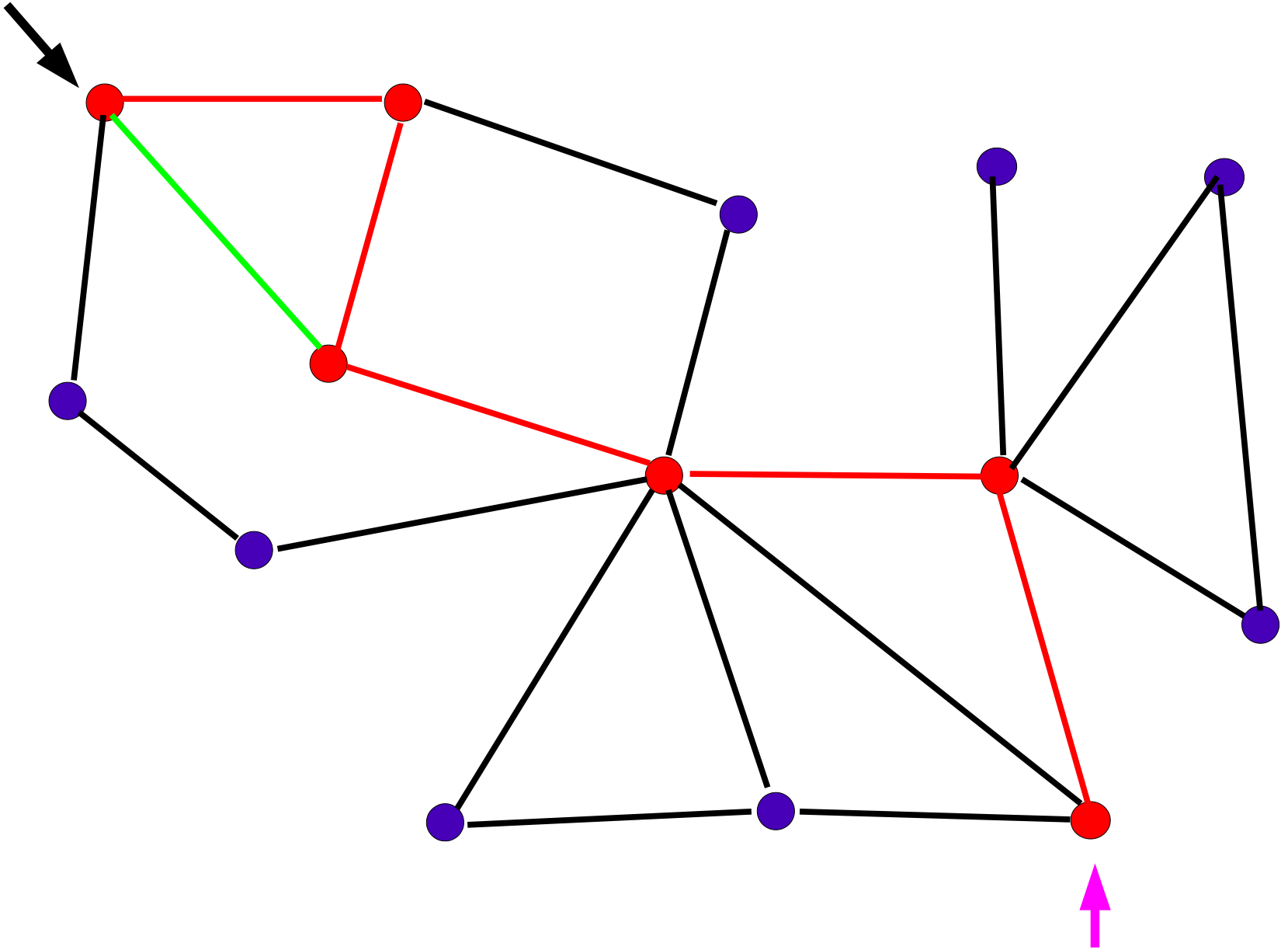
Biconnected Components



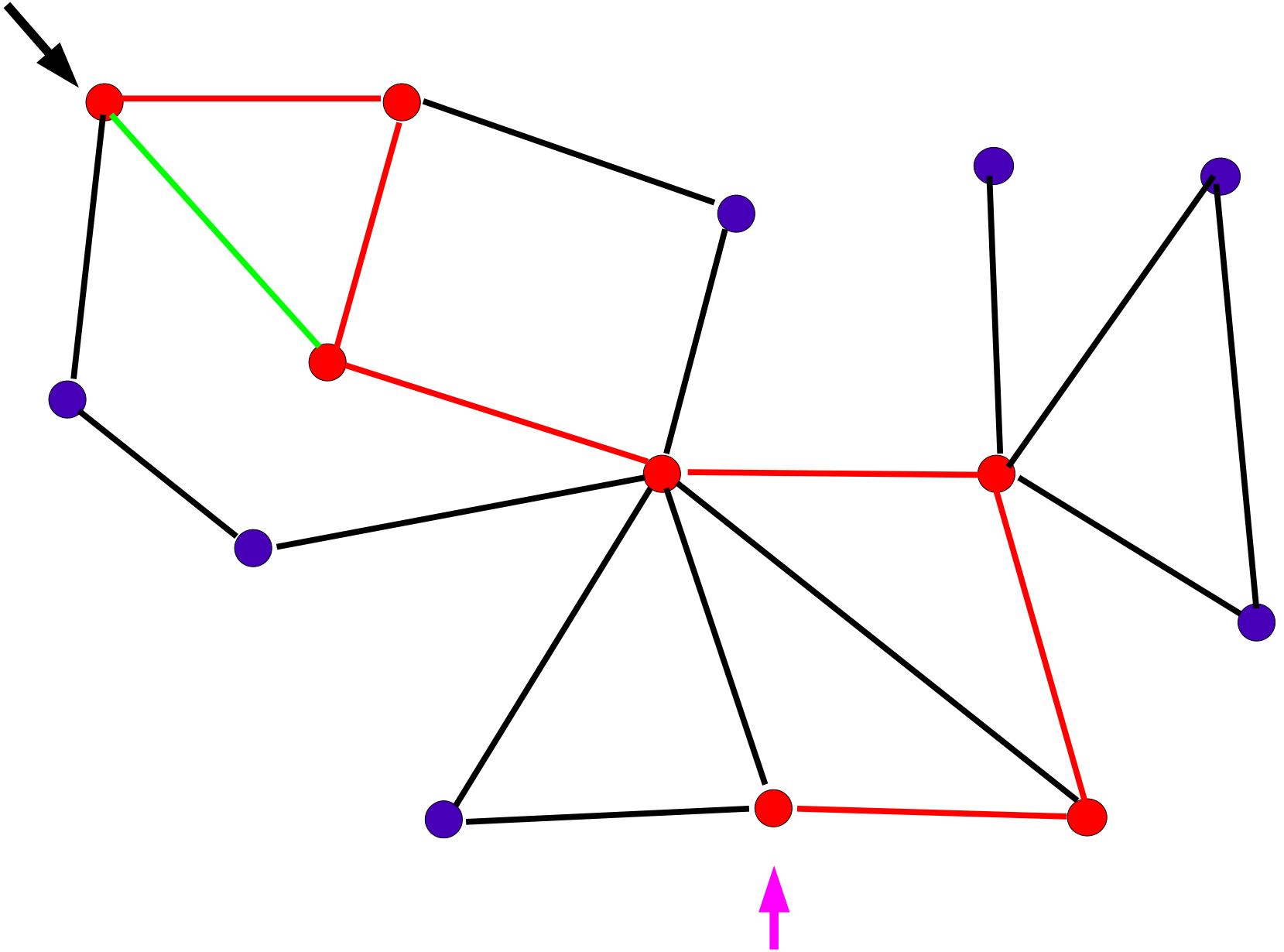
Biconnected Components



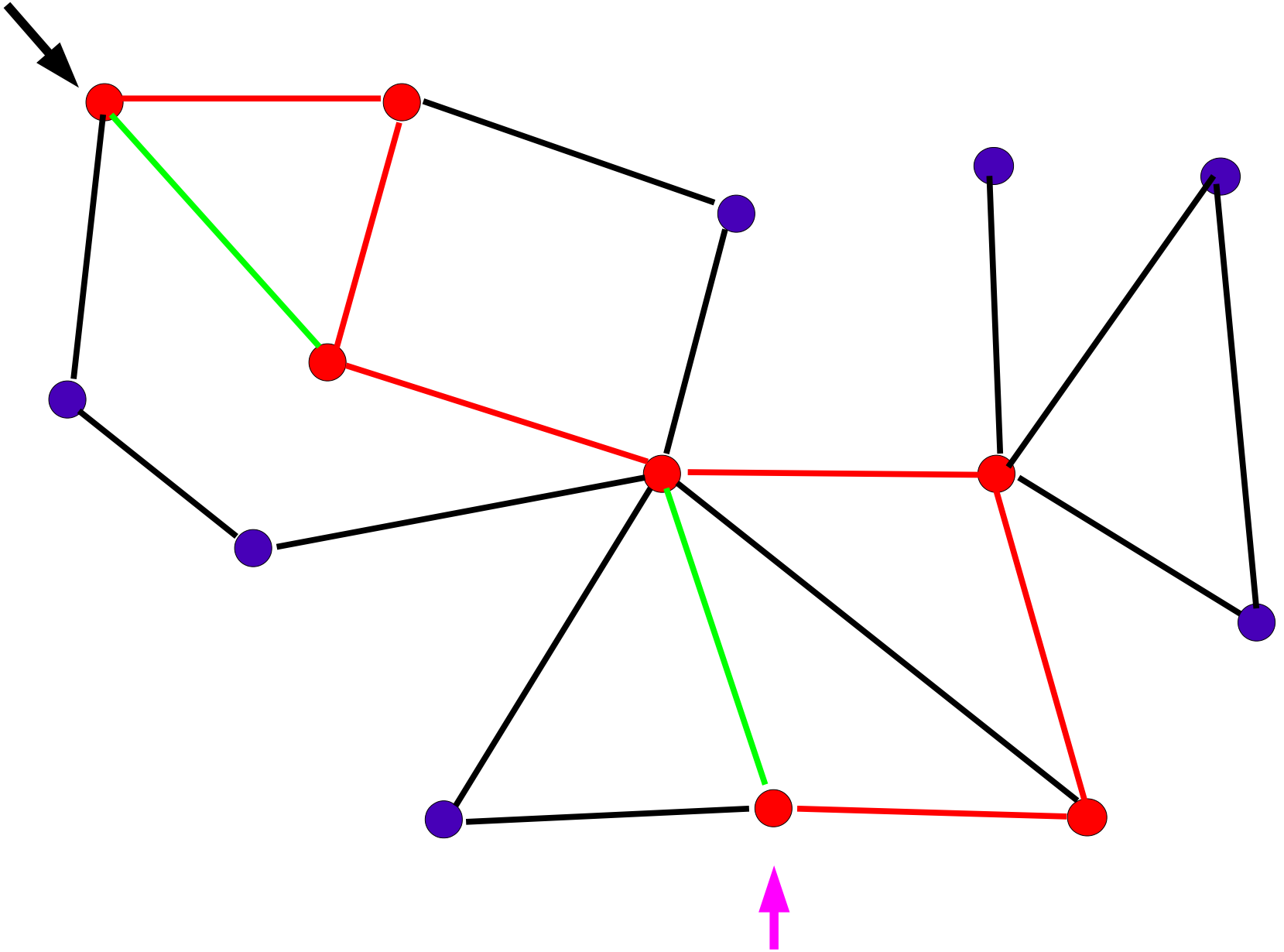
Biconnected Components



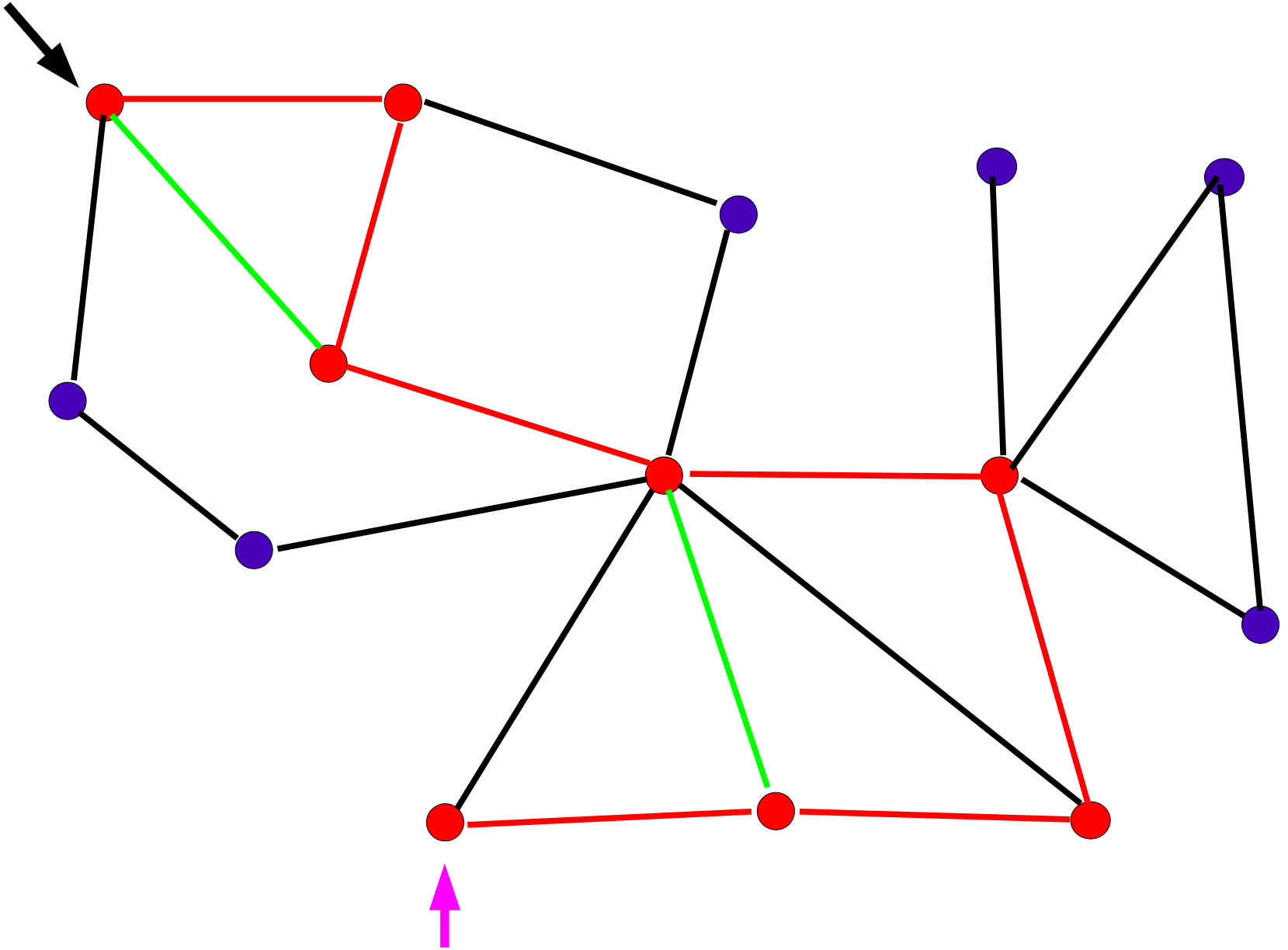
Biconnected Components



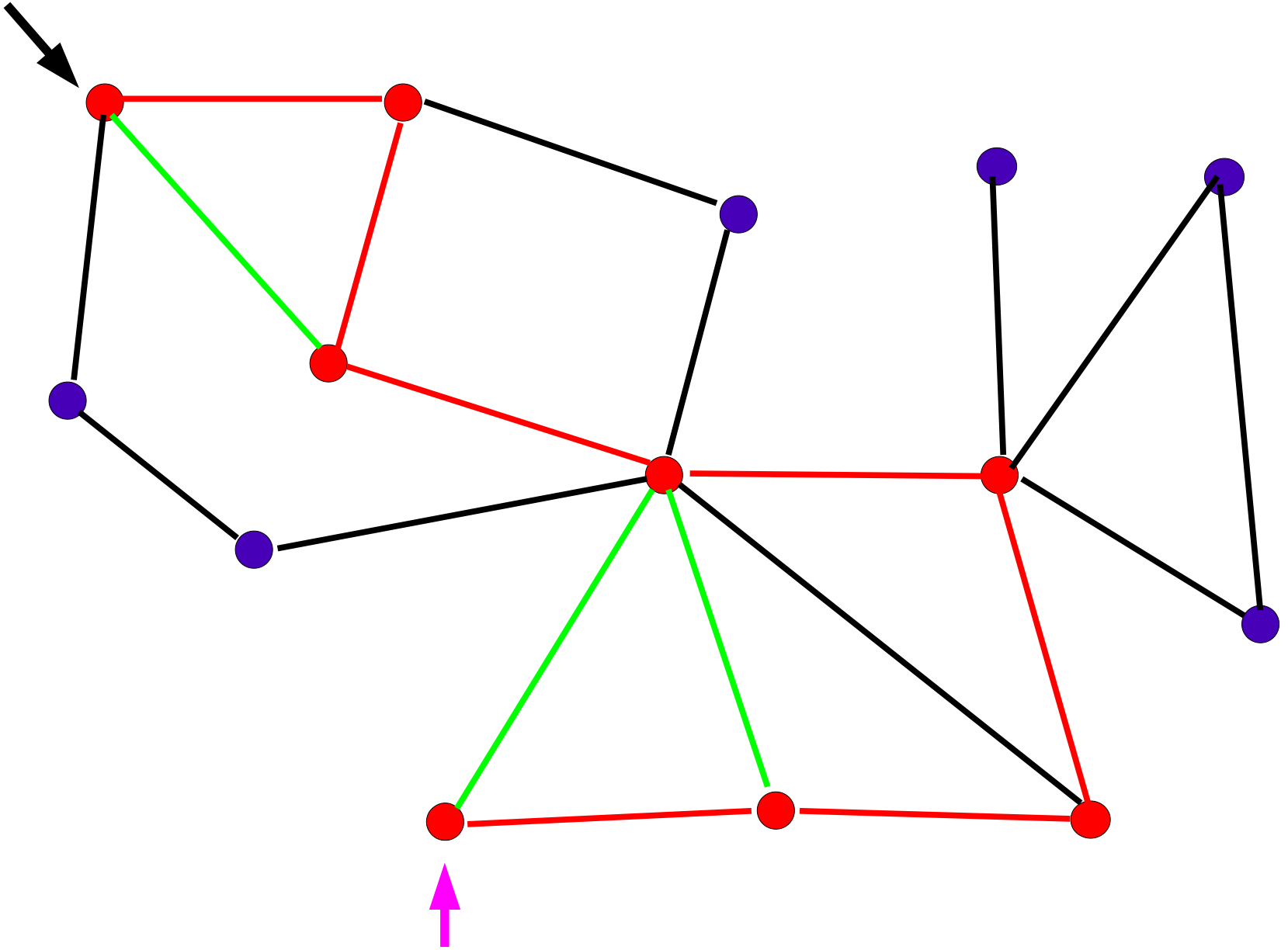
Biconnected Components



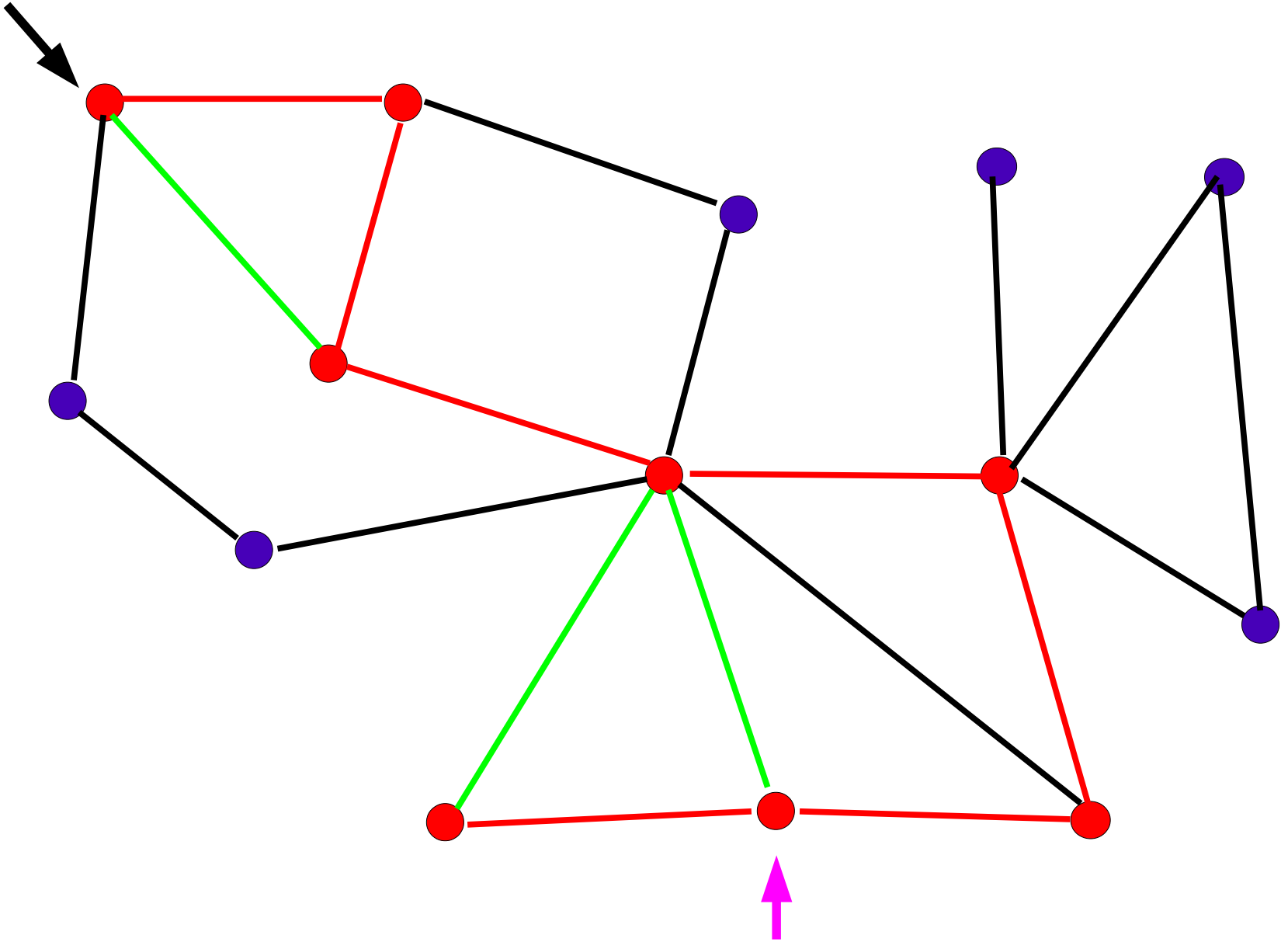
Biconnected Components



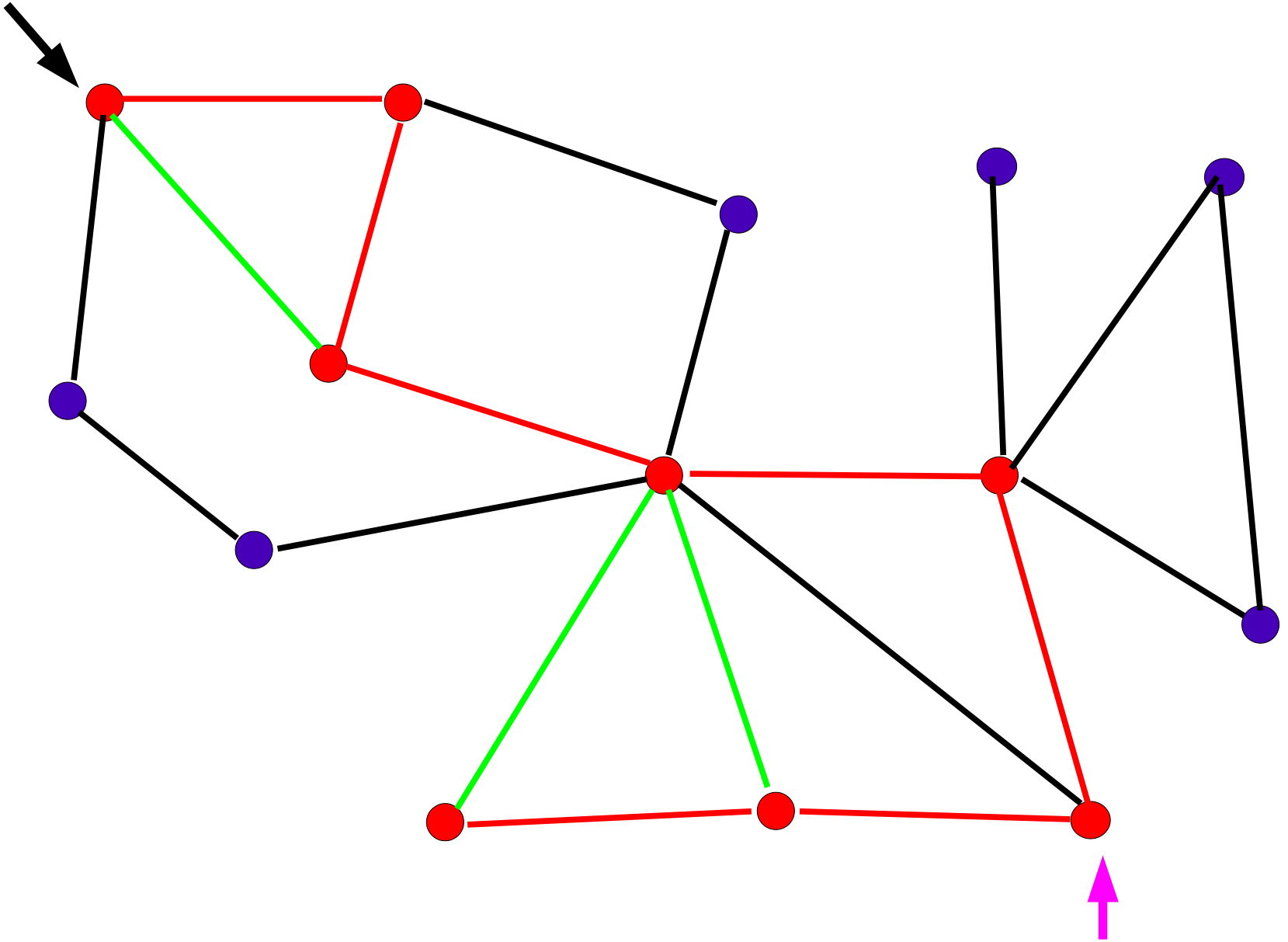
Biconnected Components



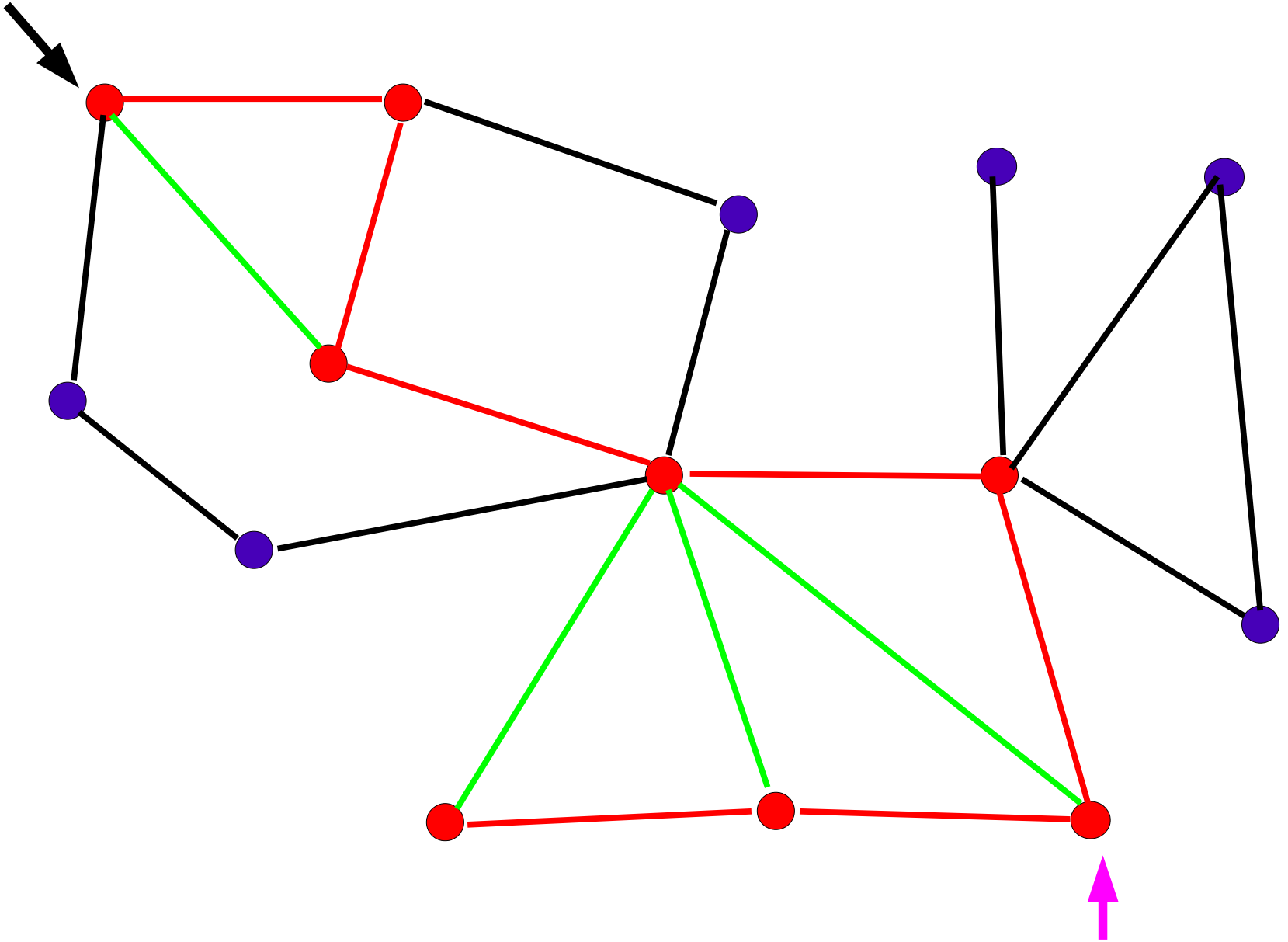
Biconnected Components



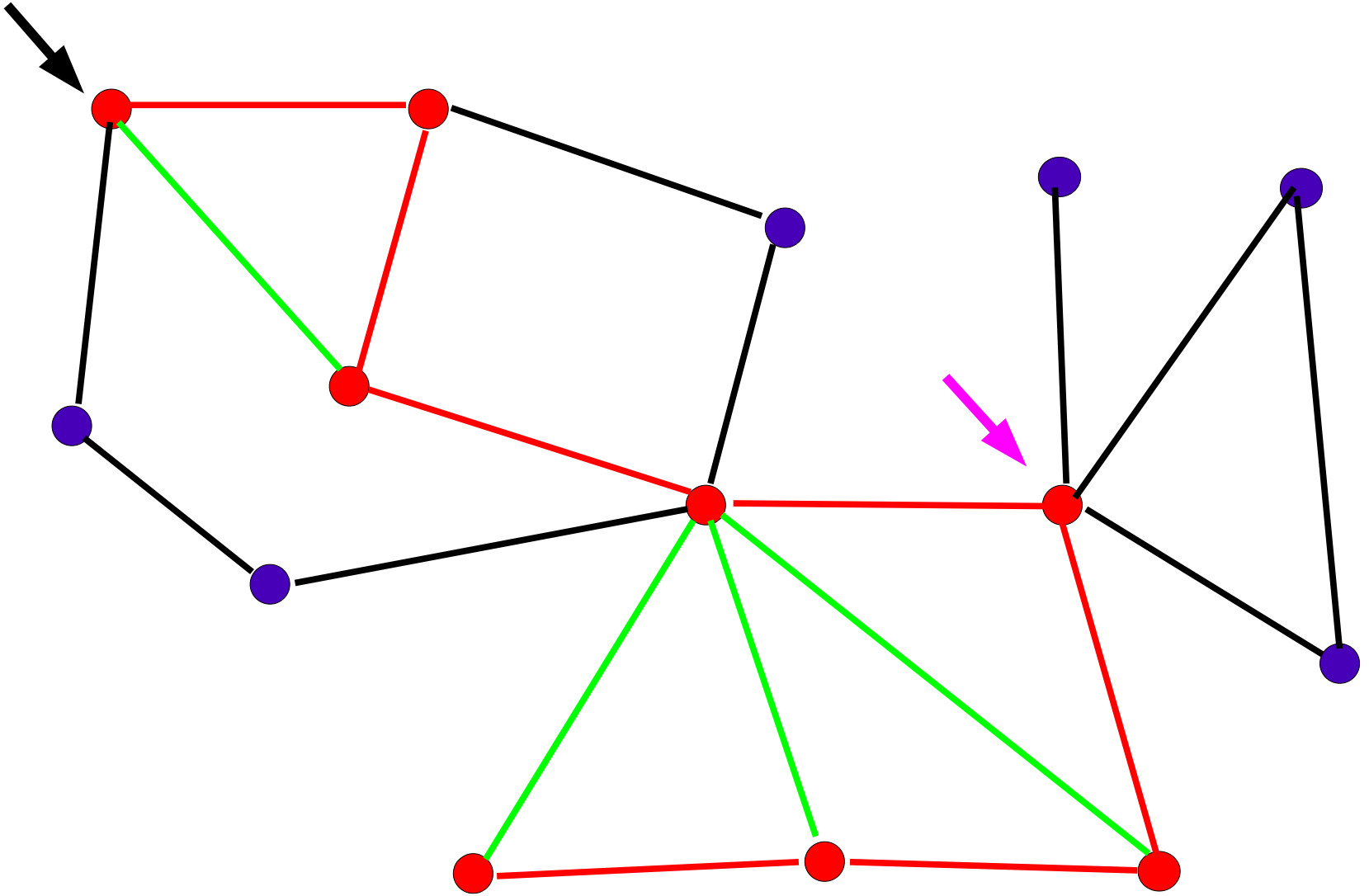
Biconnected Components



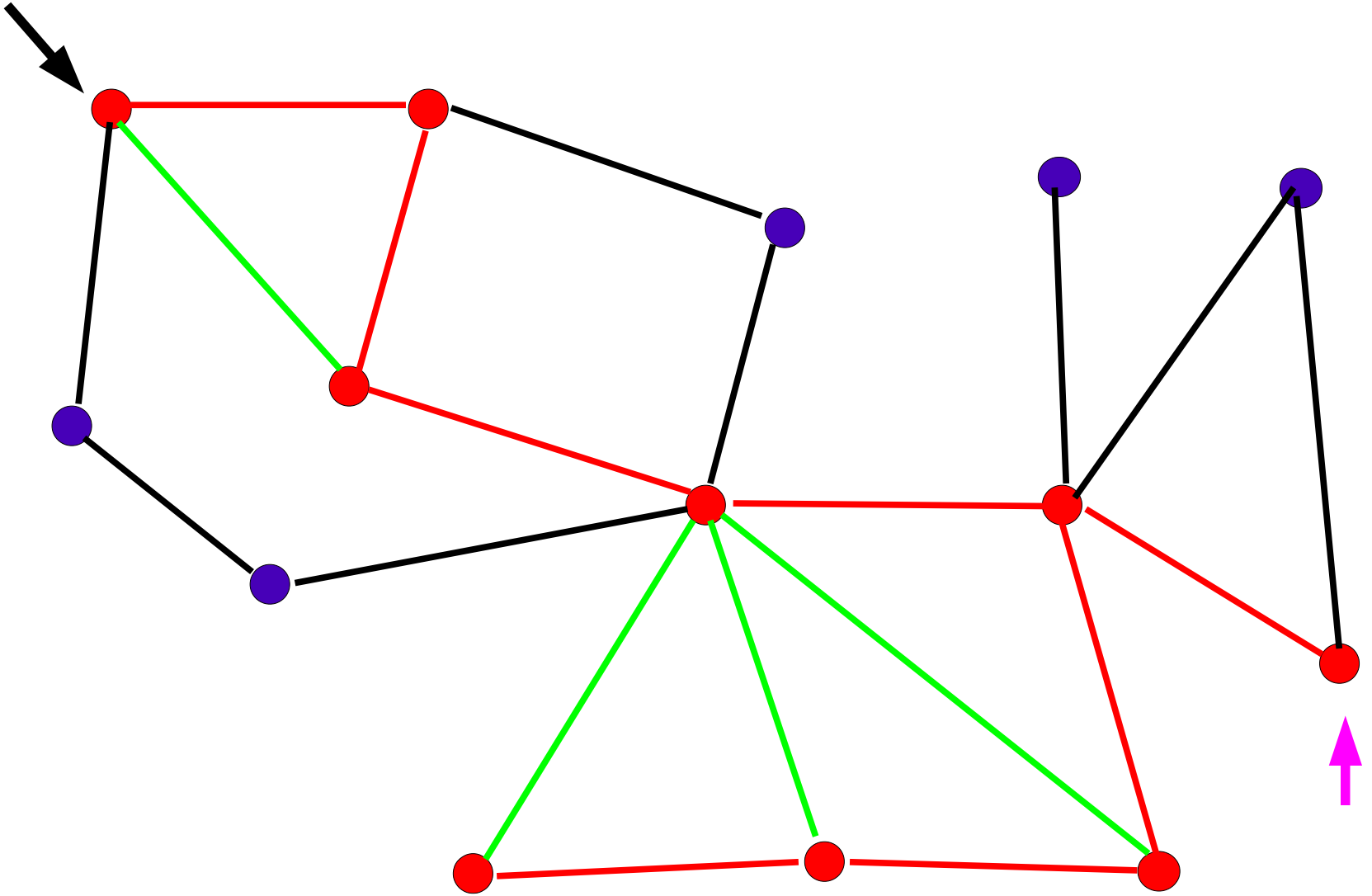
Biconnected Components



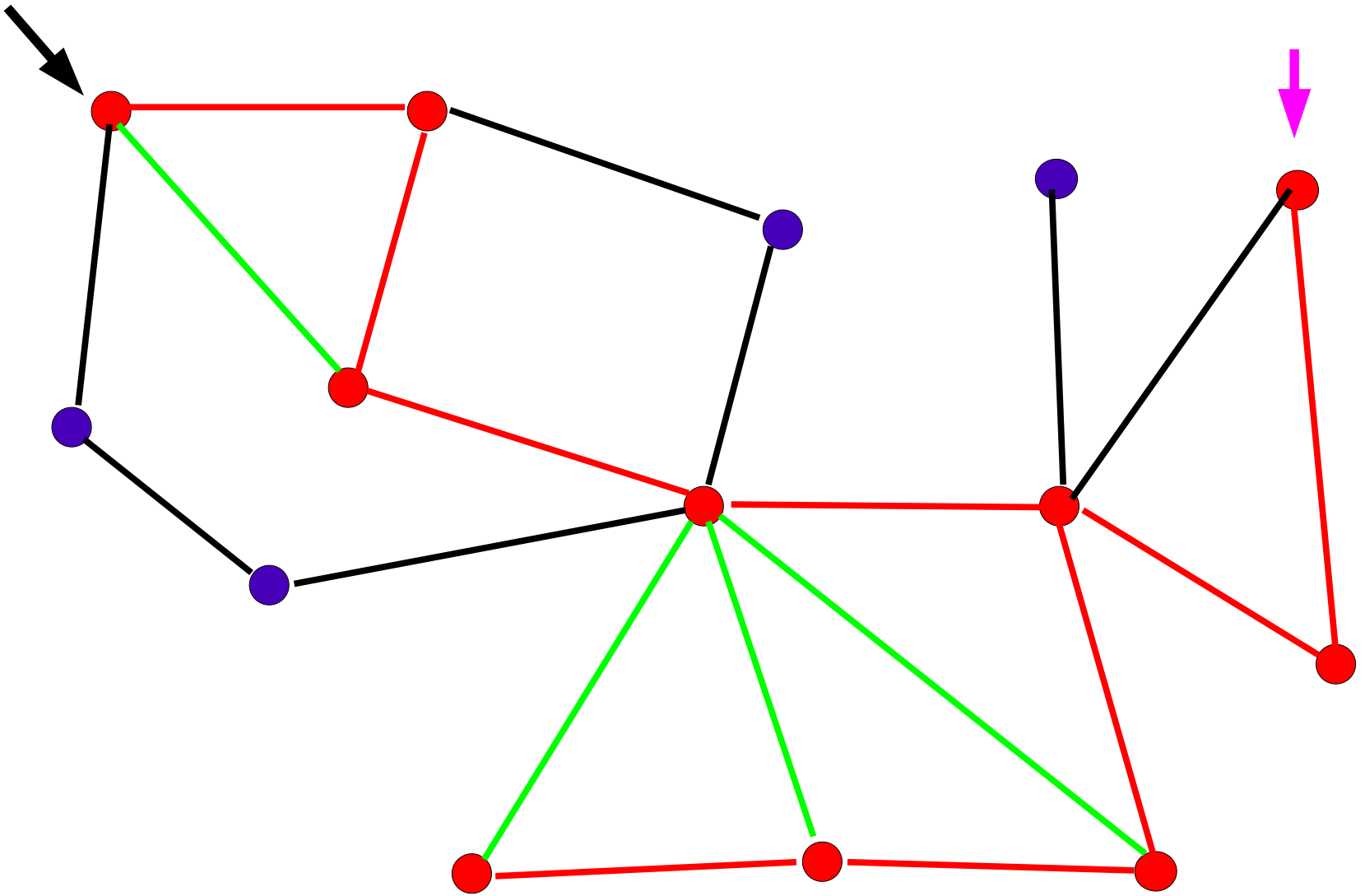
Biconnected Components



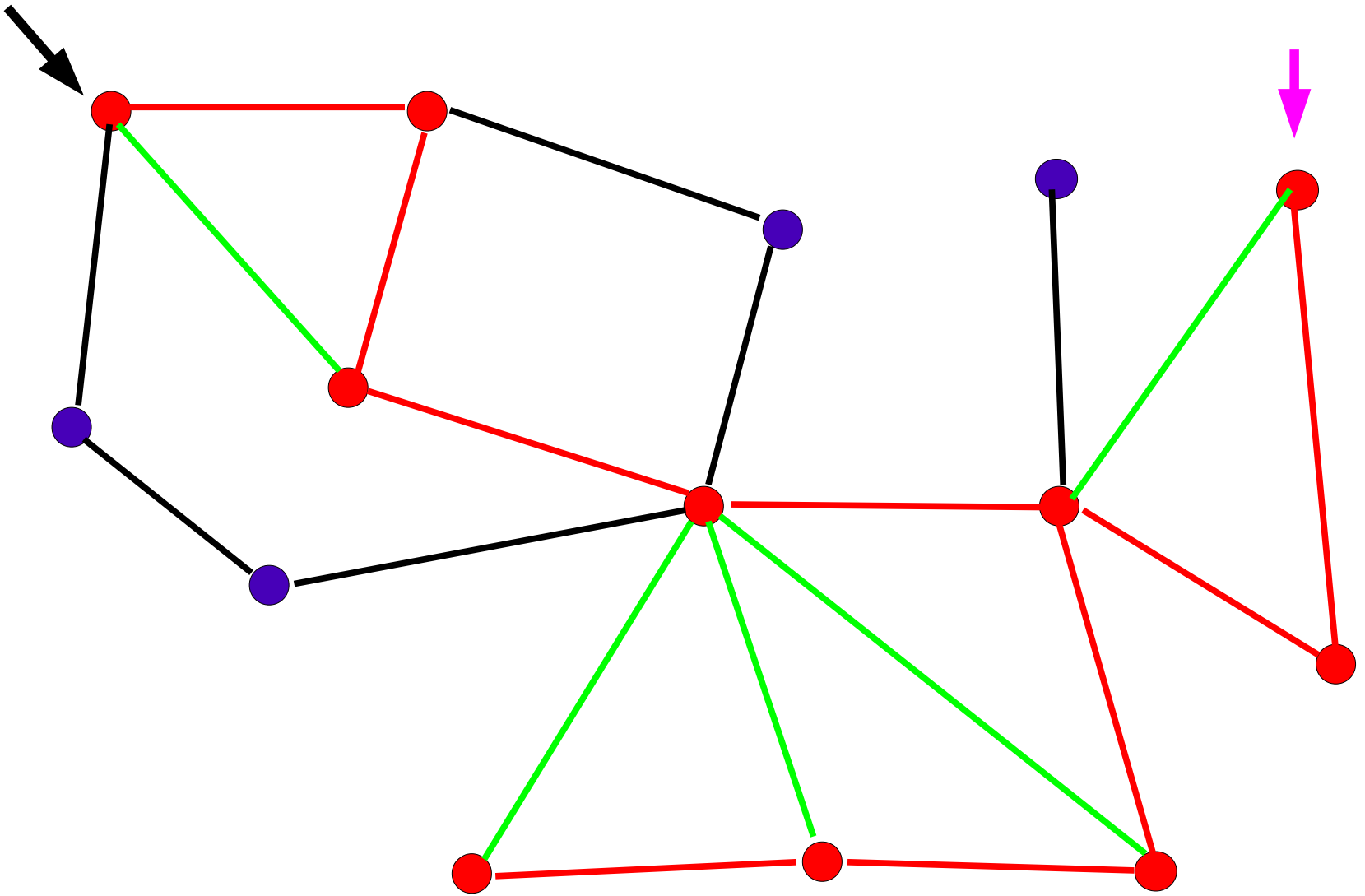
Biconnected Components



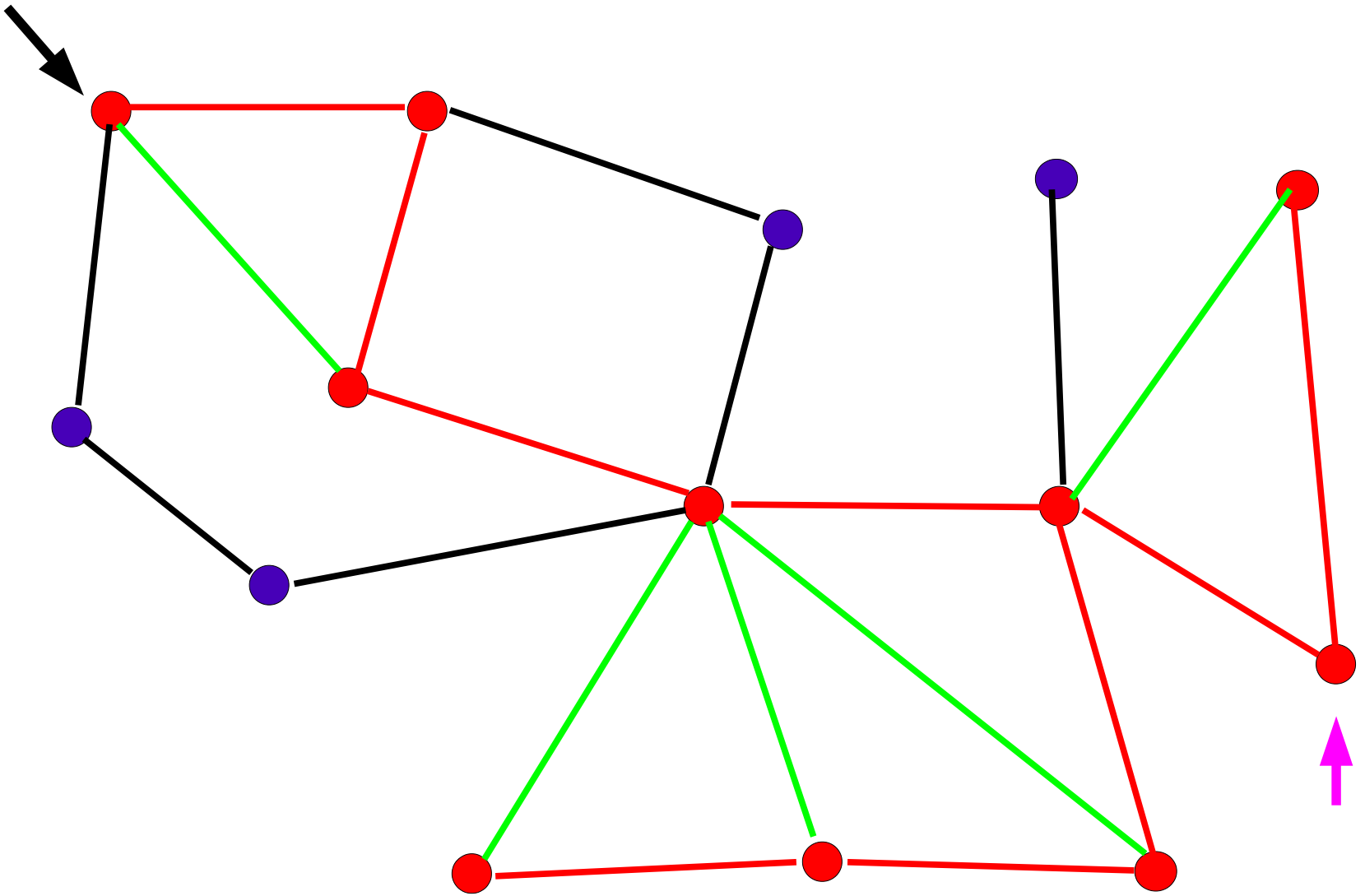
Biconnected Components



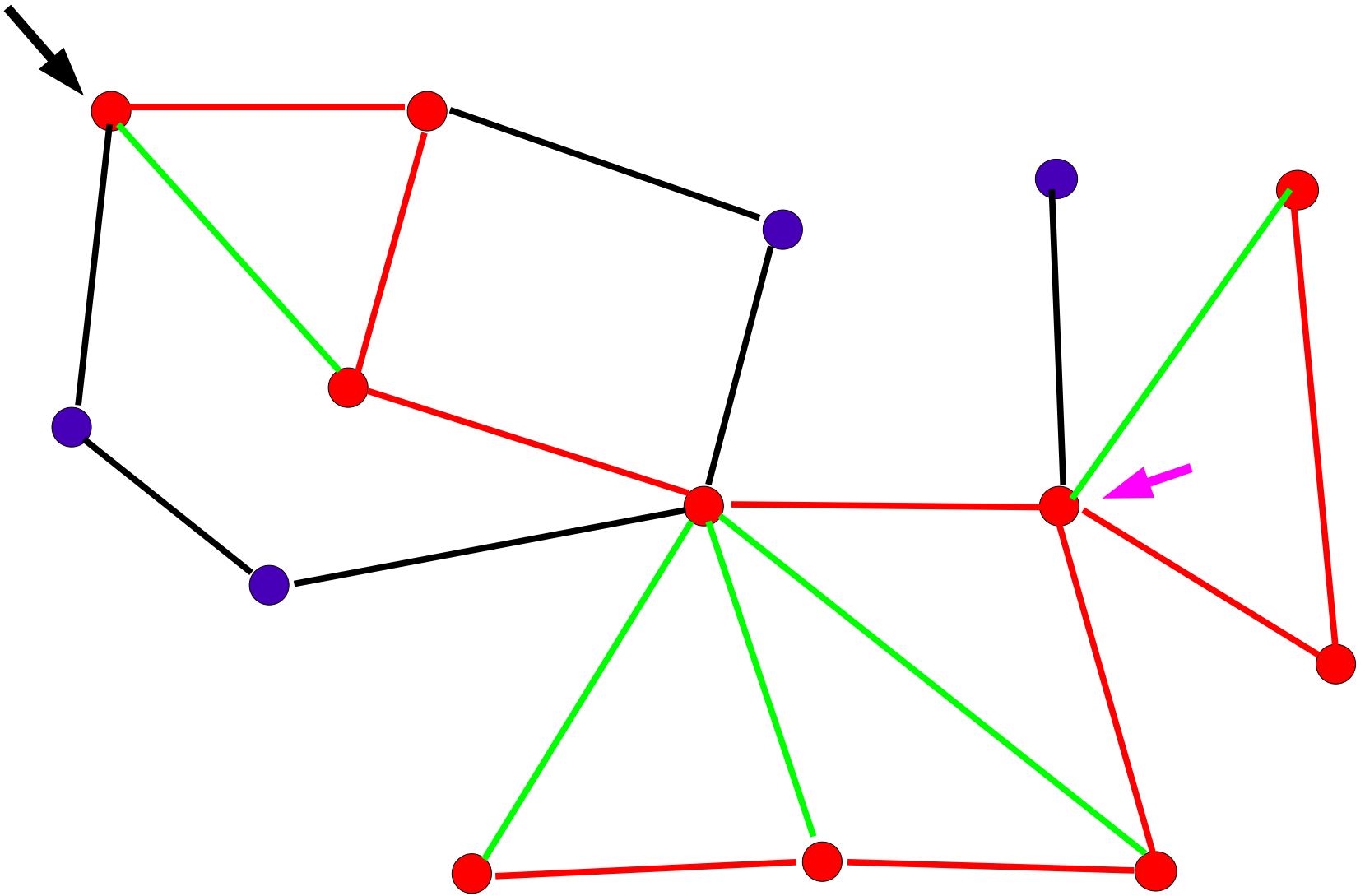
Biconnected Components



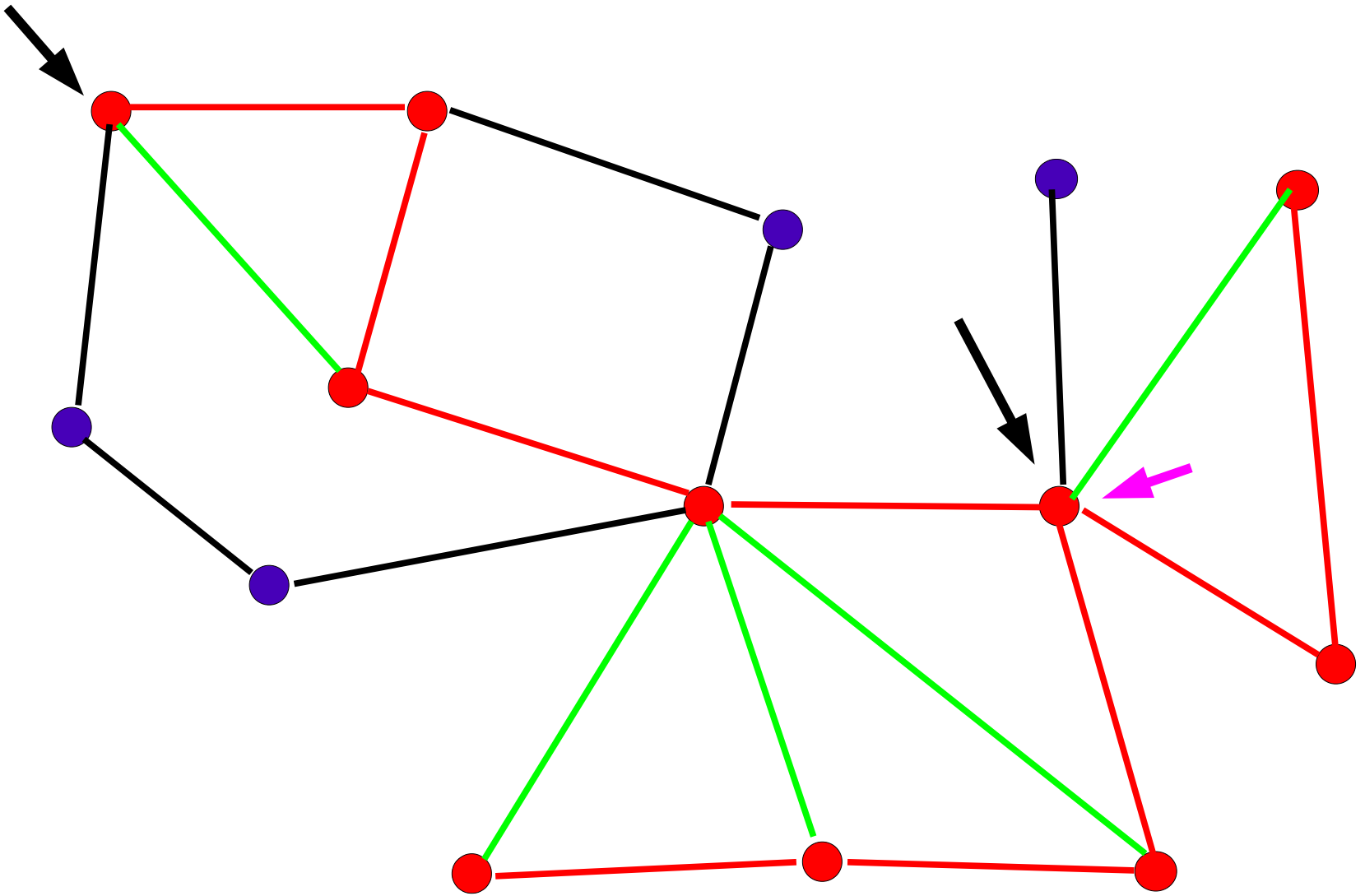
Biconnected Components



Biconnected Components

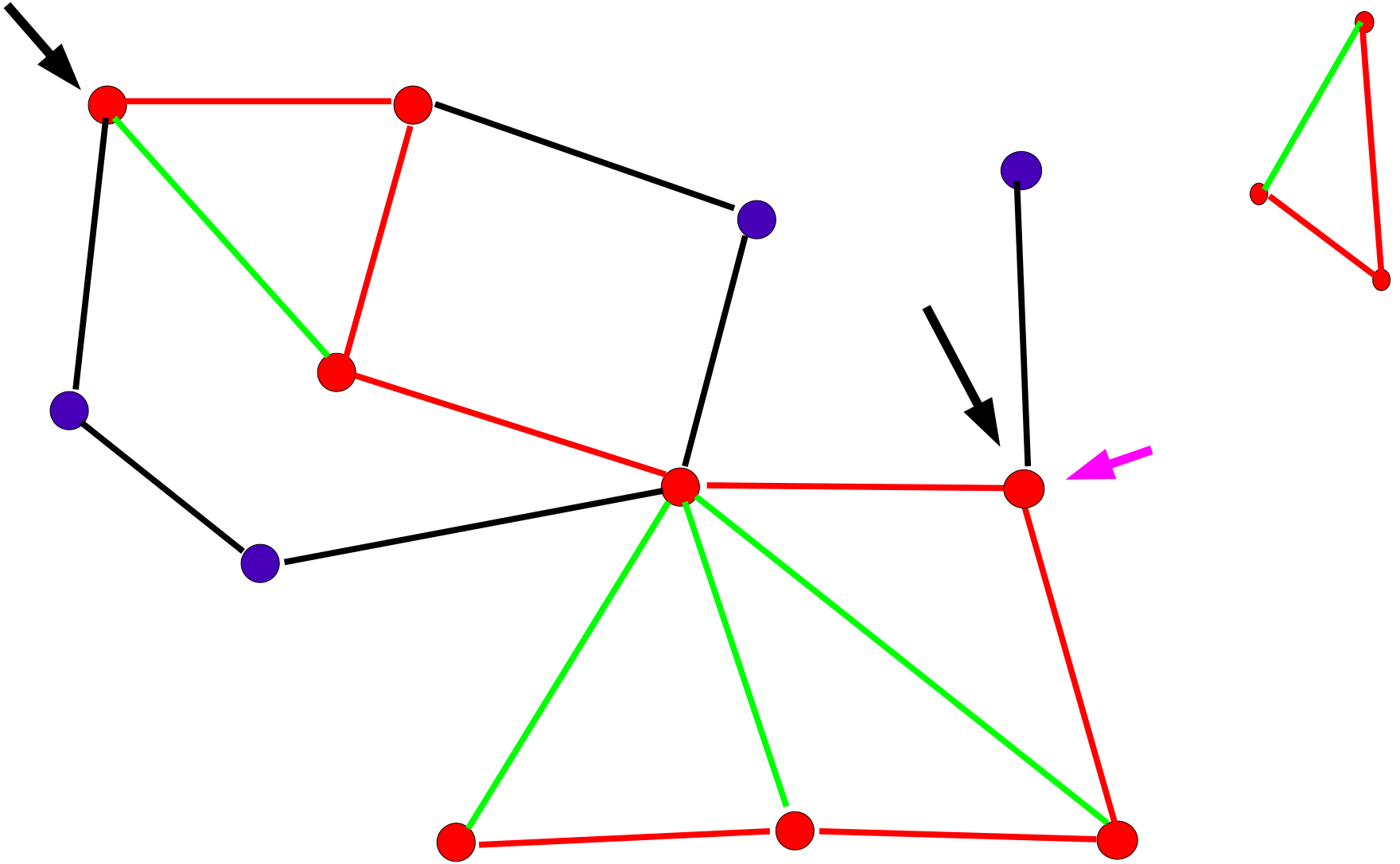


Biconnected Components



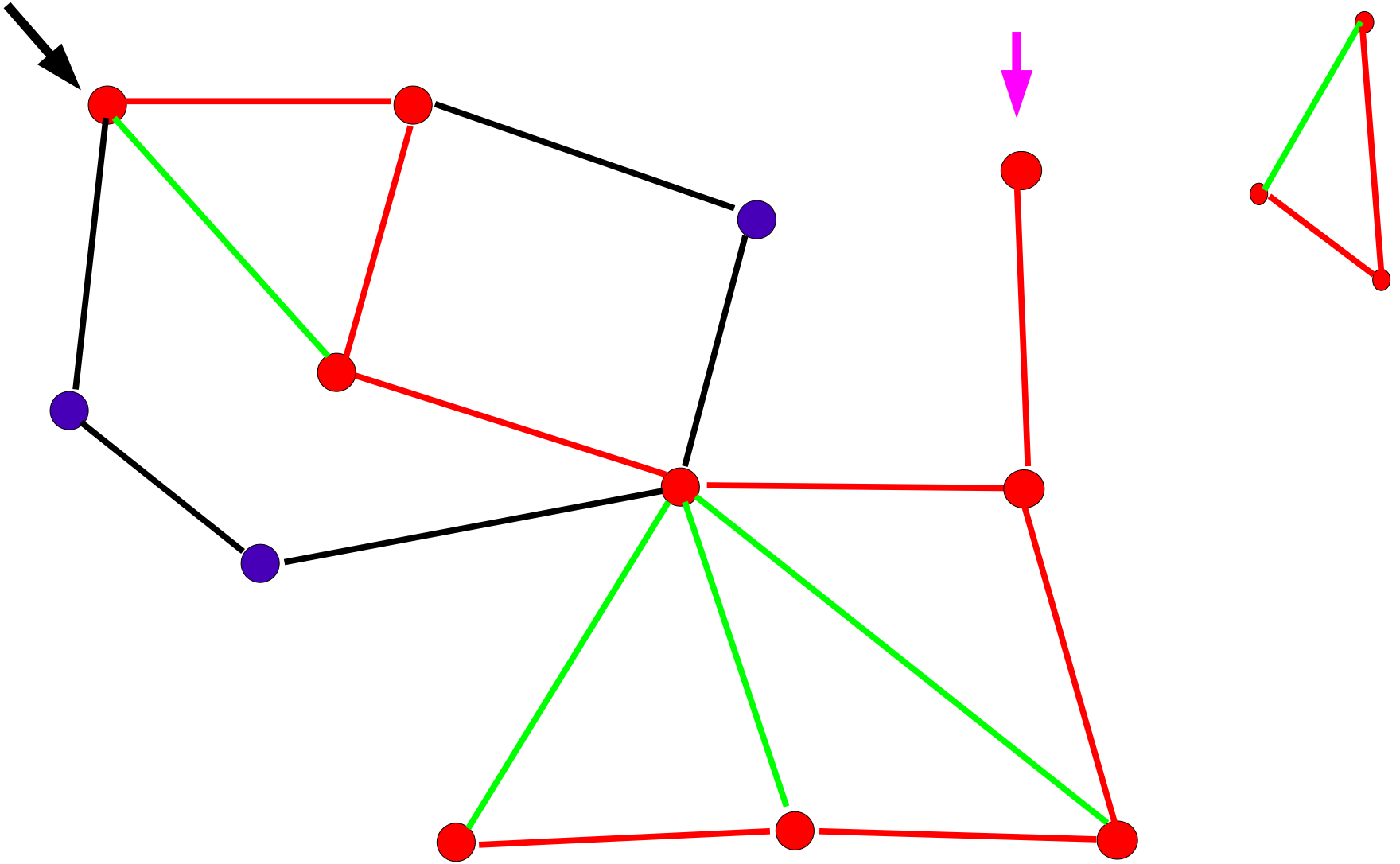
Must be an articulation point since paths from all neighbors so far considered all lead back to this vertex

Biconnected Components

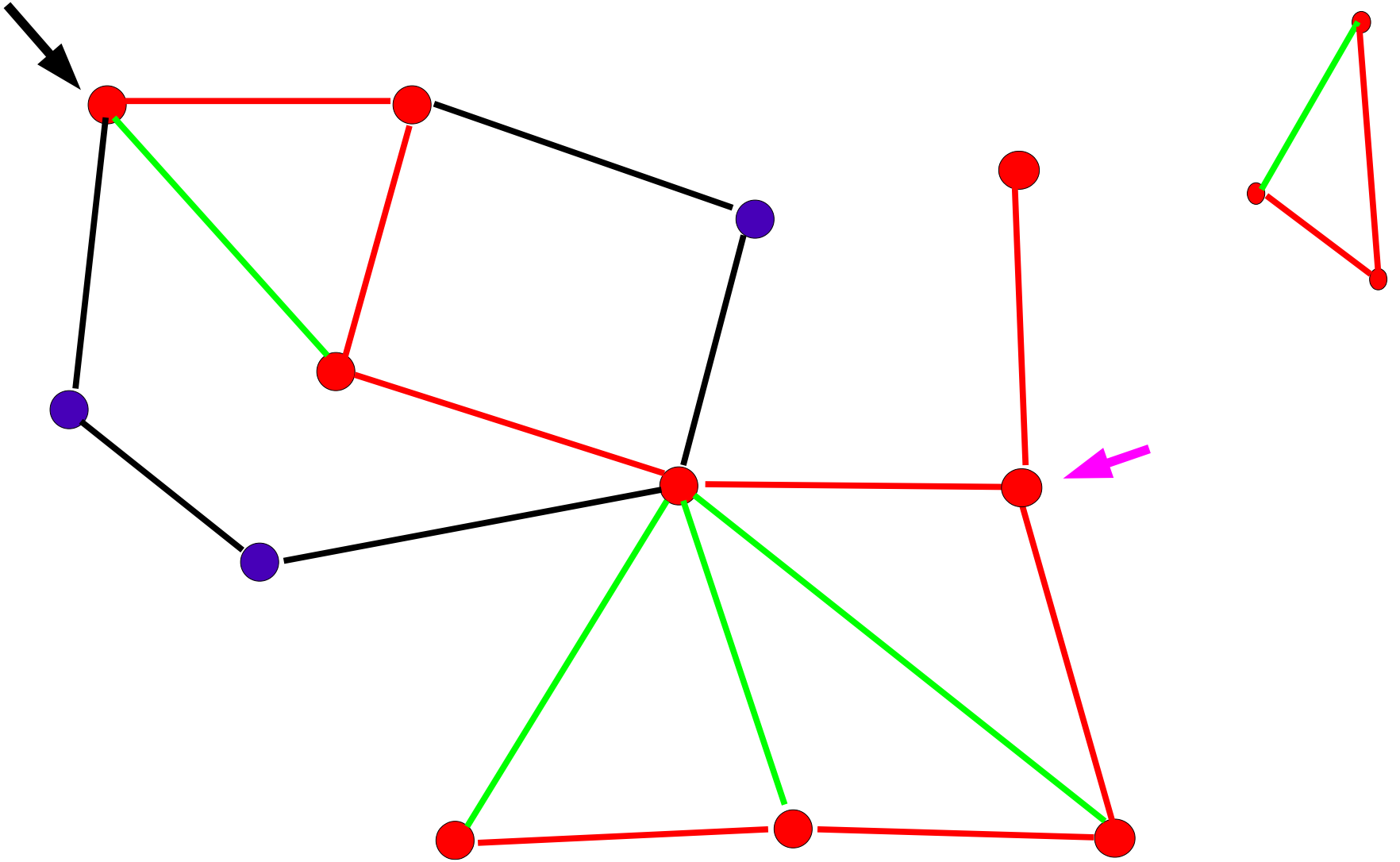


Save that biconnected component on the side and continue

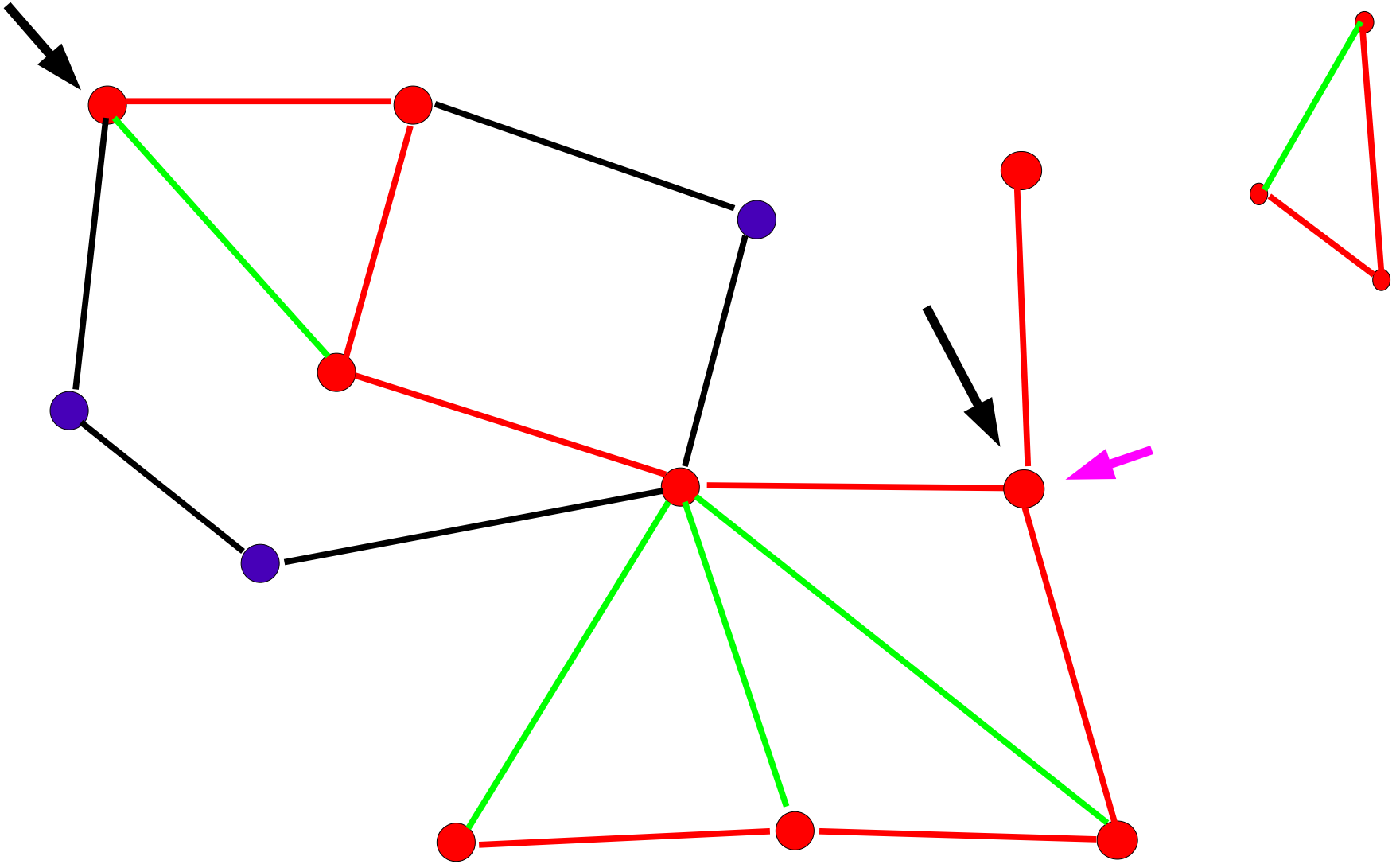
Biconnected Components



Biconnected Components

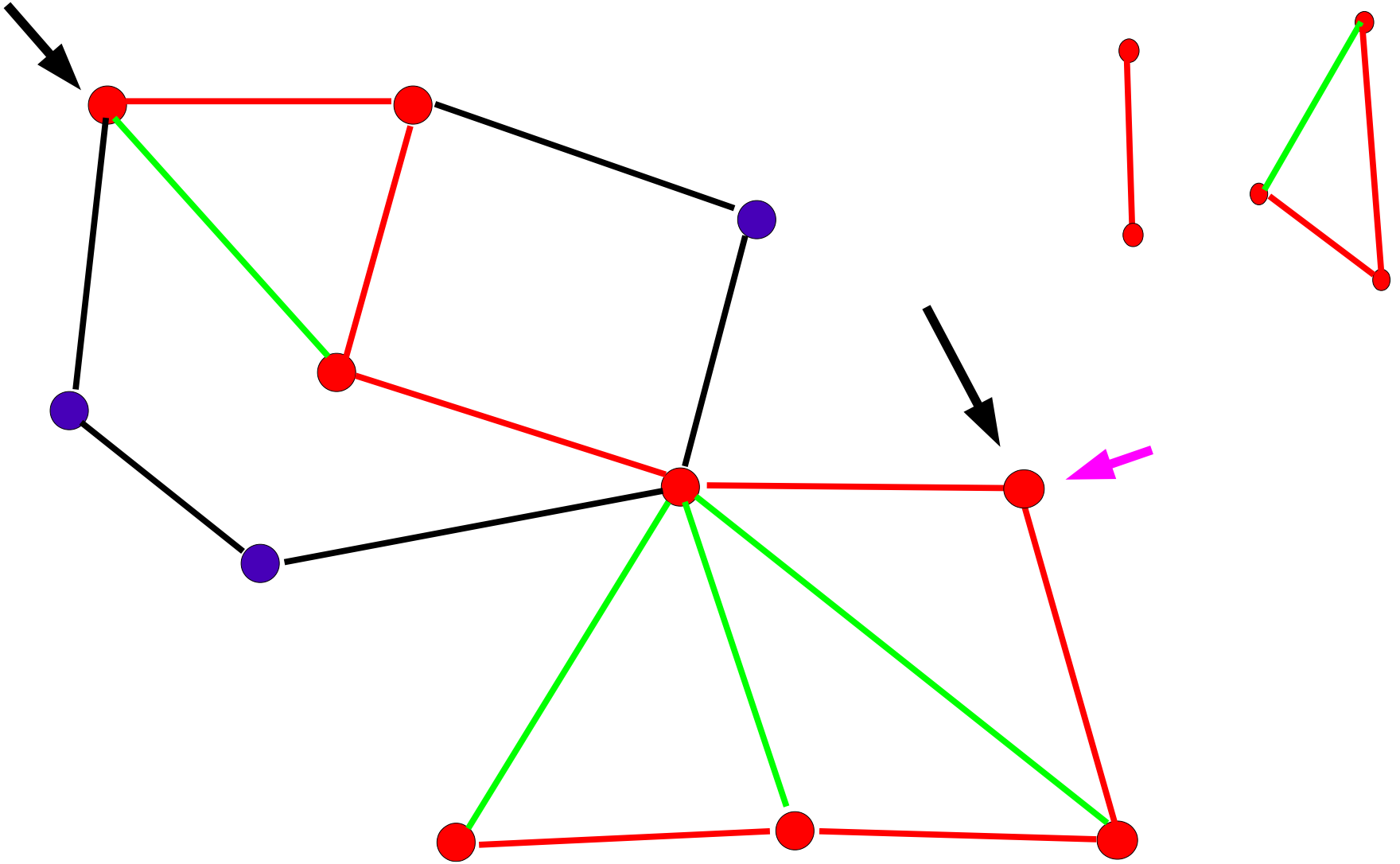


Biconnected Components



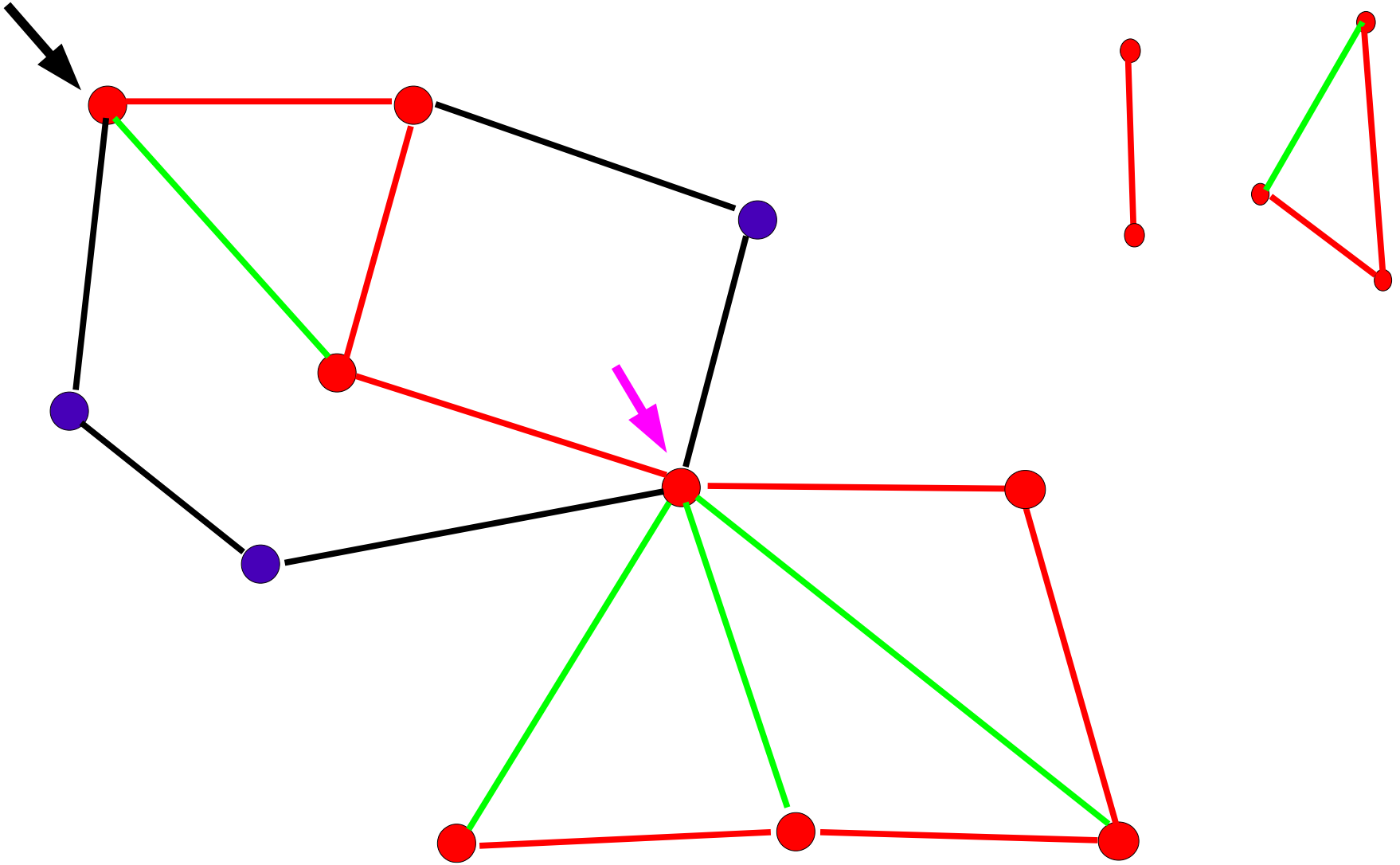
Must be an articulation point since paths from all neighbors so far considered all lead back to this vertex

Biconnected Components

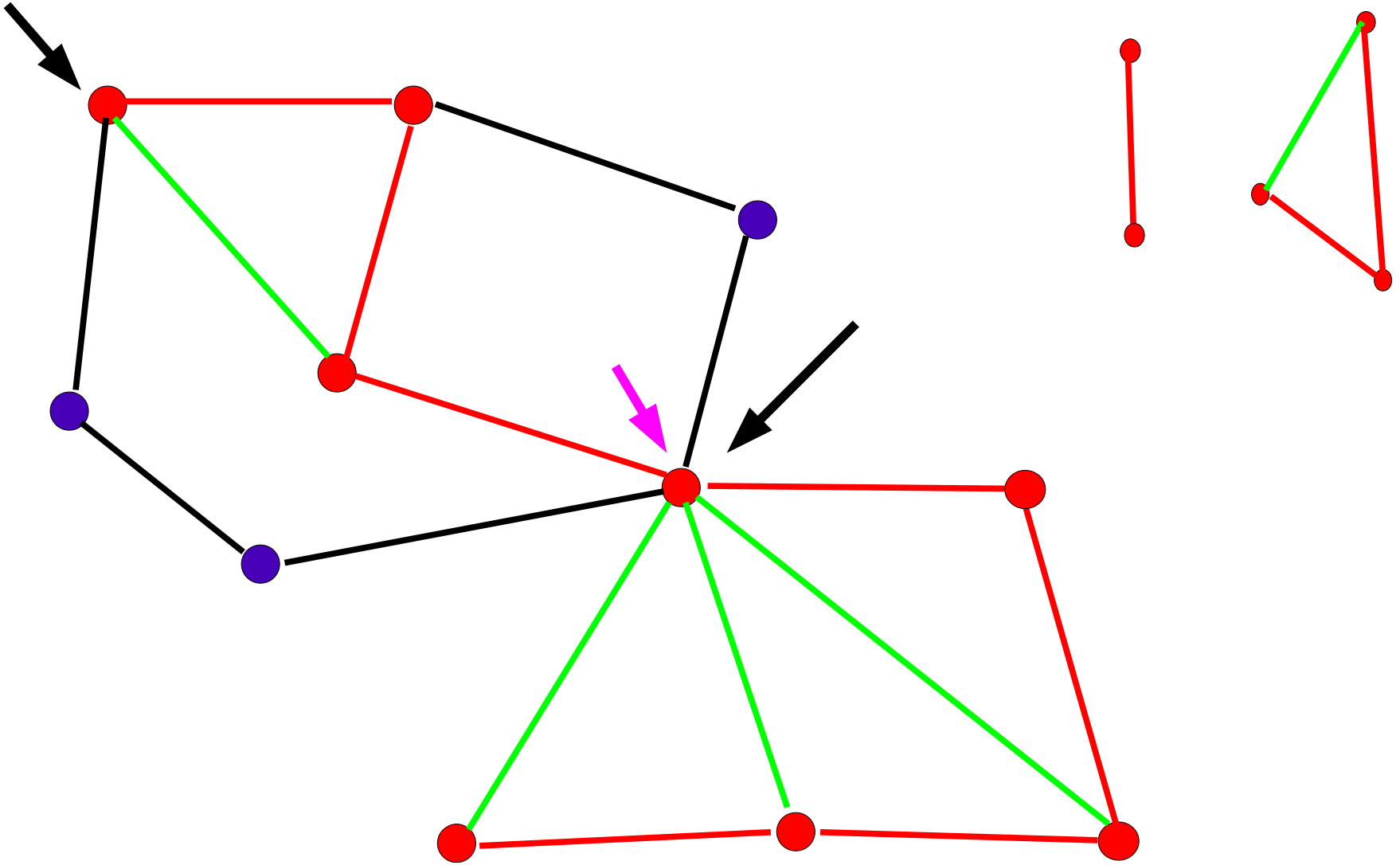


Save that biconnected component on the side and continue

Biconnected Components

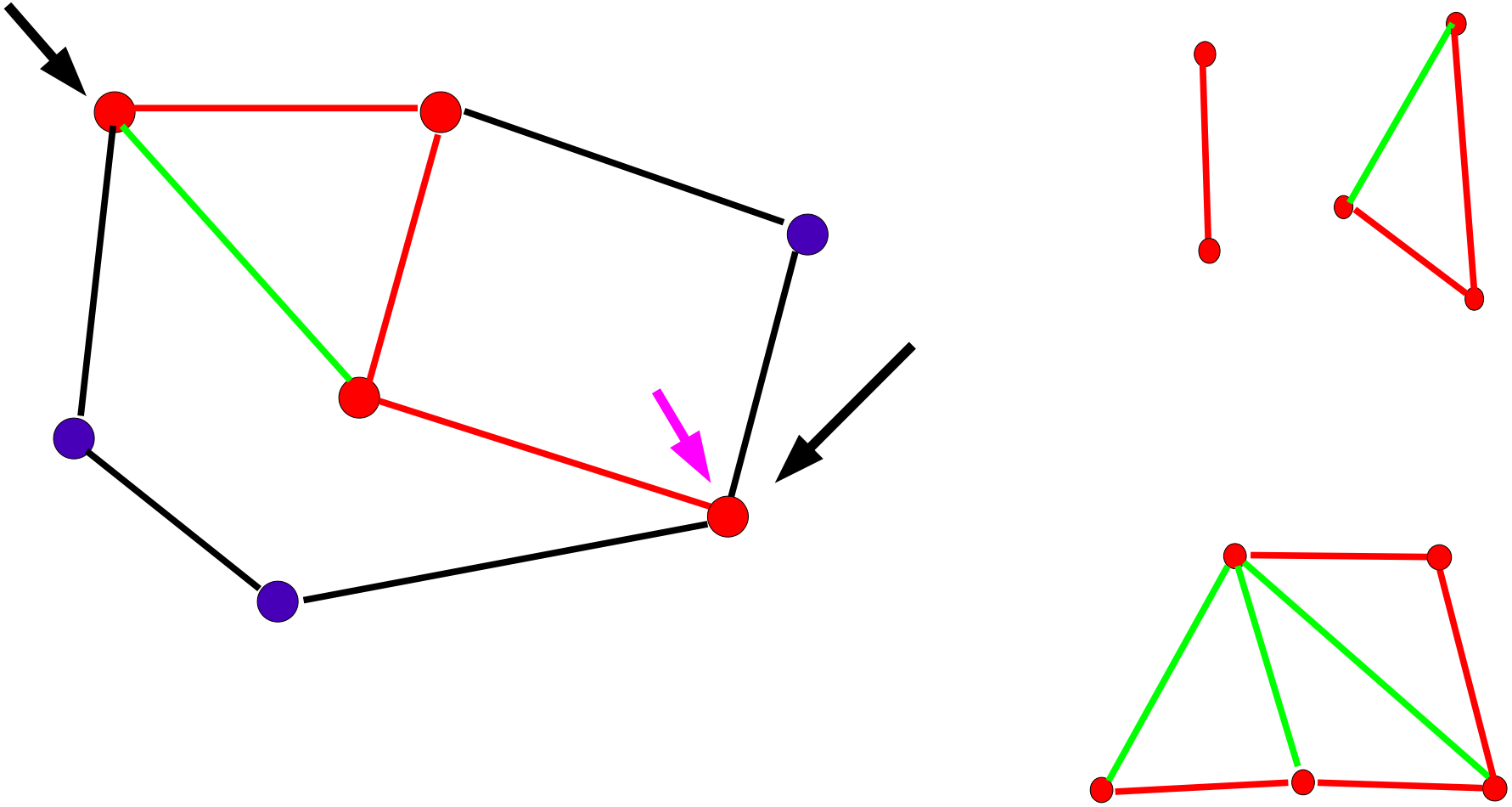


Biconnected Components



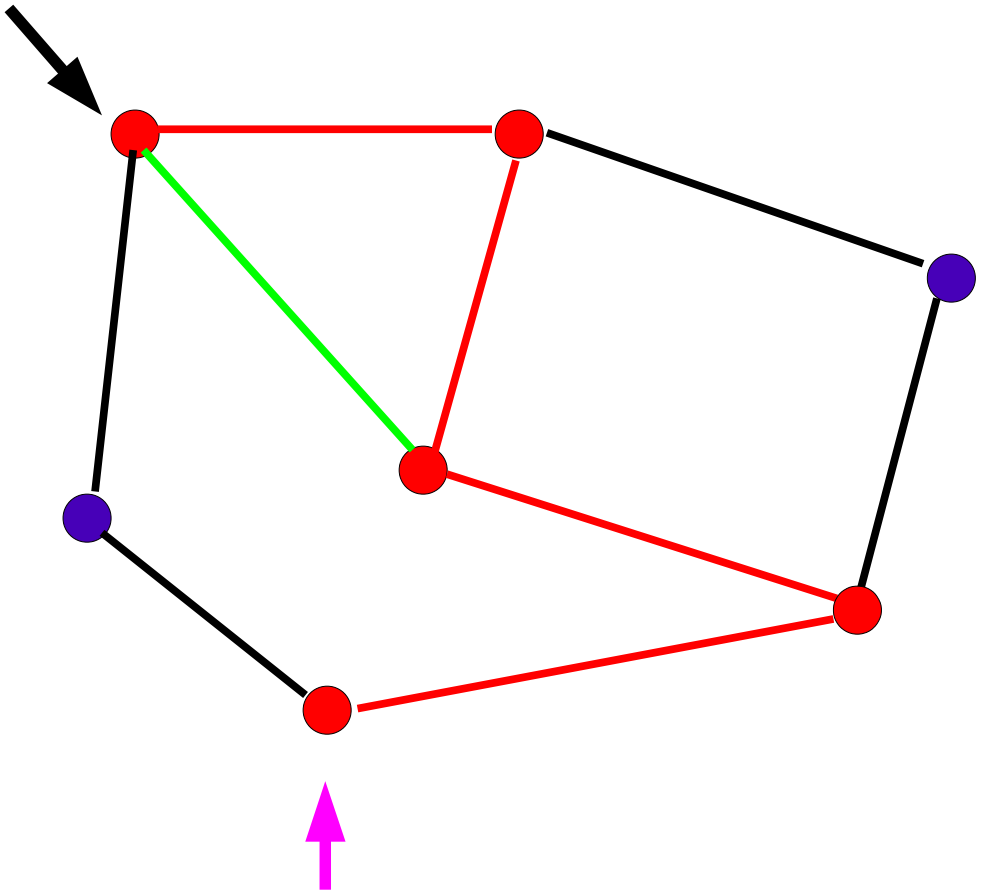
Must be an articulation point since paths from all neighbors so far considered all lead back to this vertex

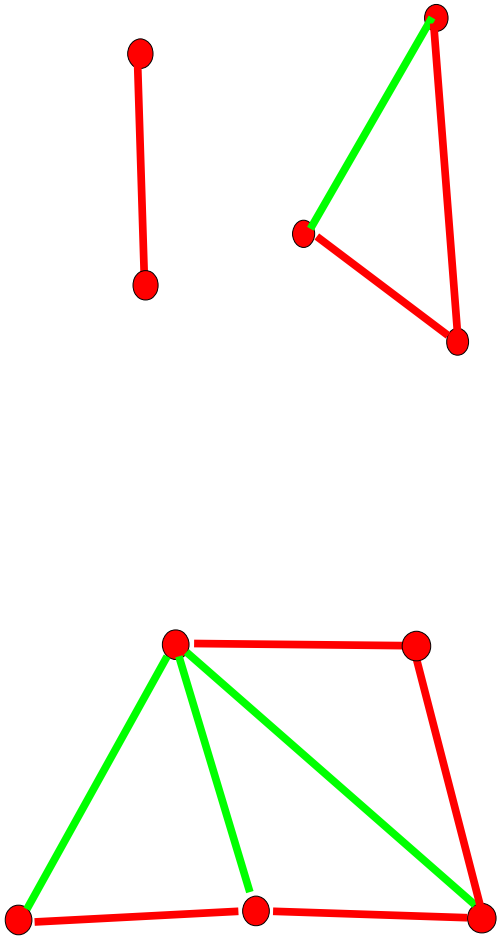
Biconnected Components



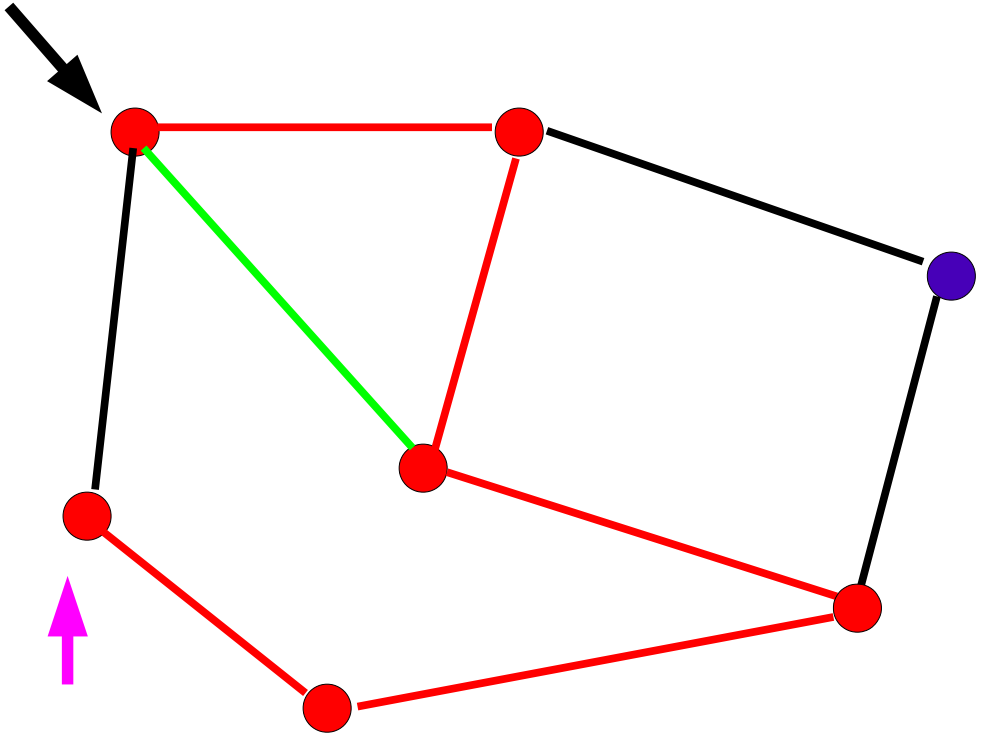
Save that biconnected component on the side and continue

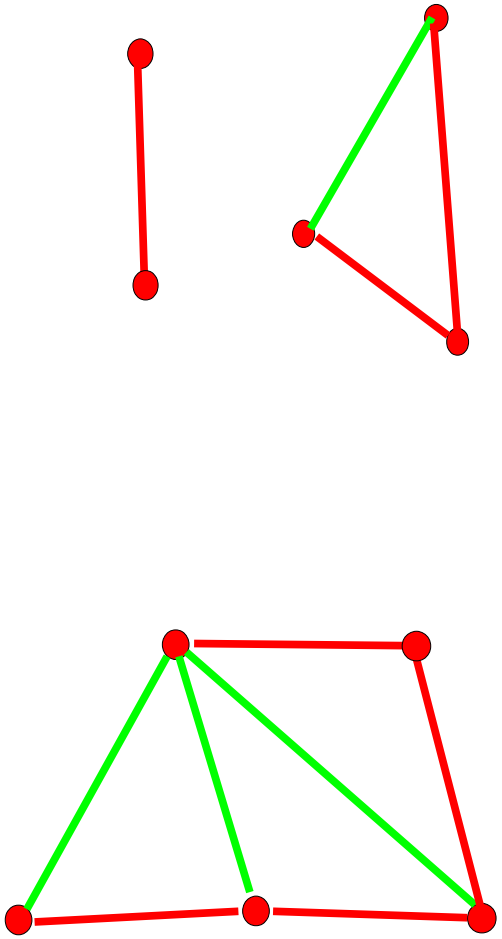
Biconnected Components



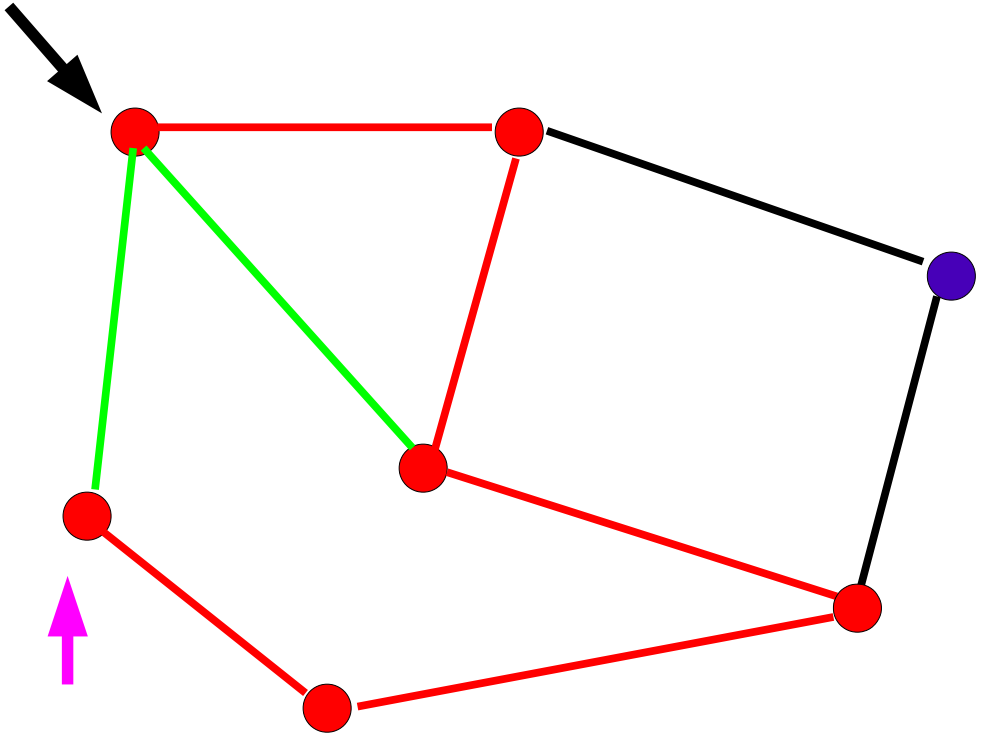


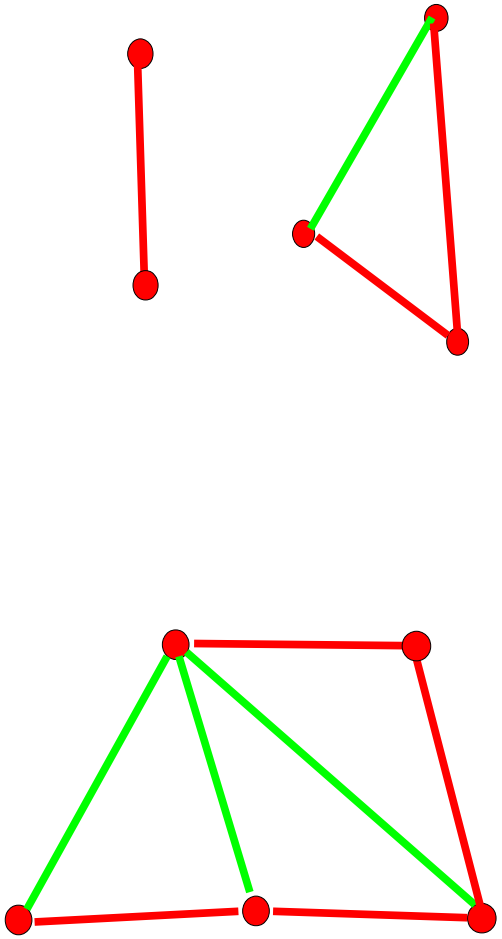
Biconnected Components



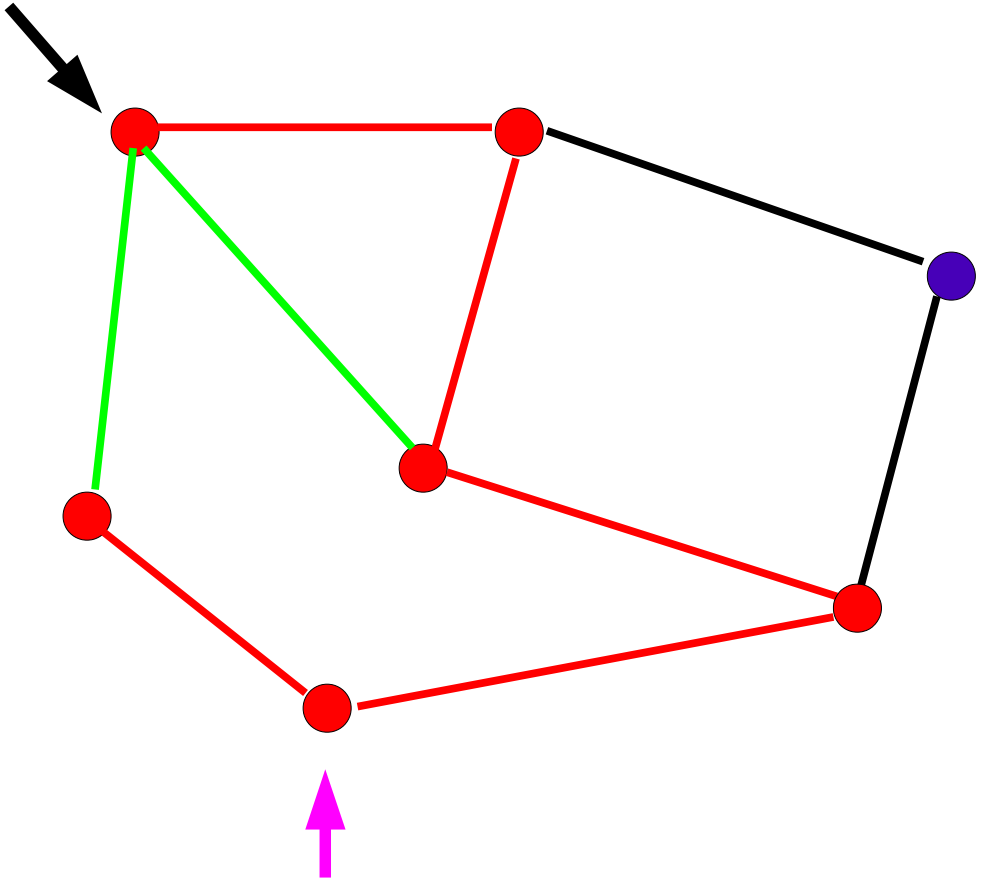


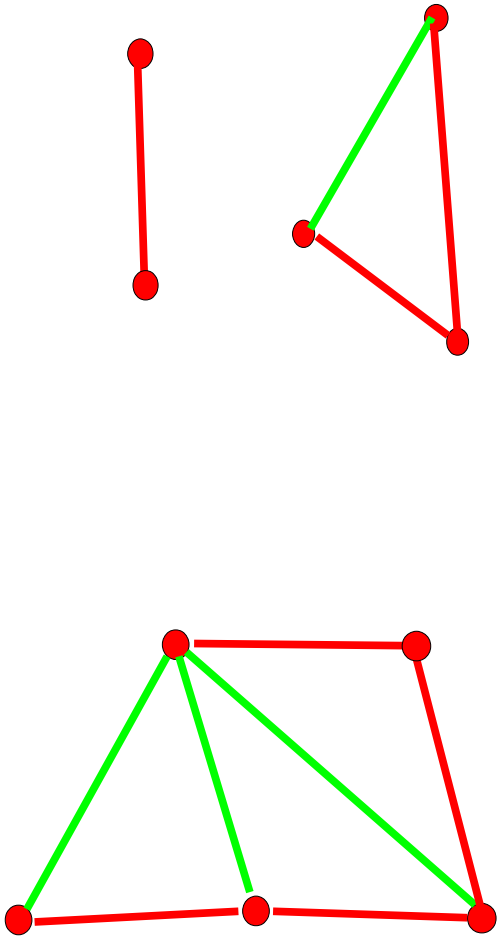
Biconnected Components



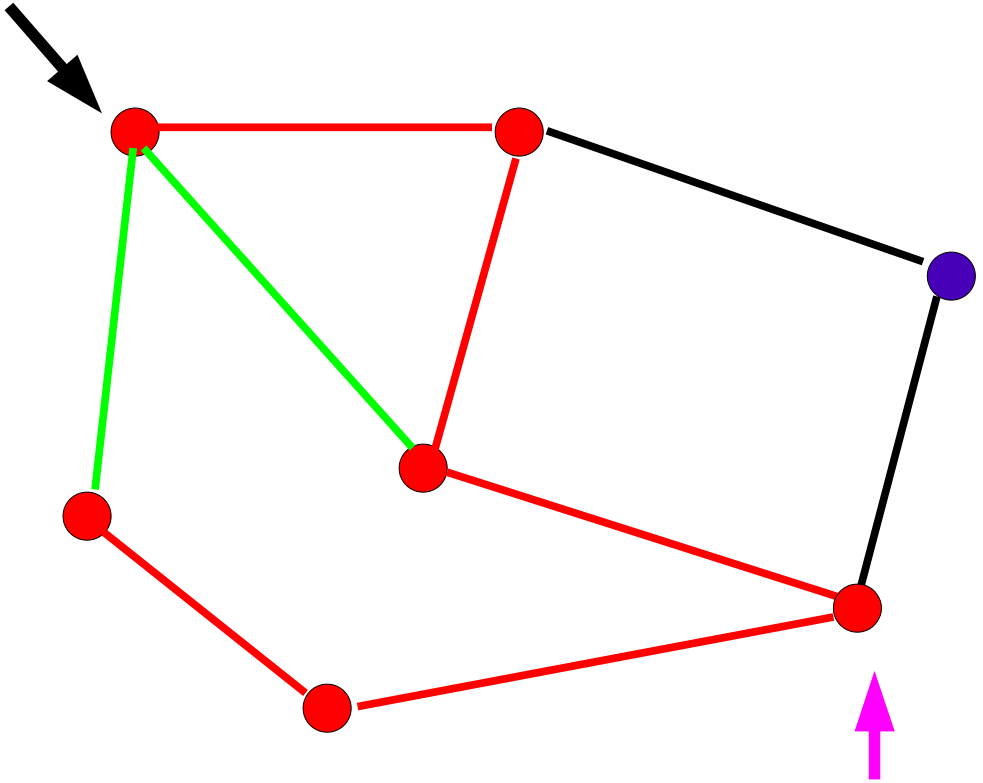


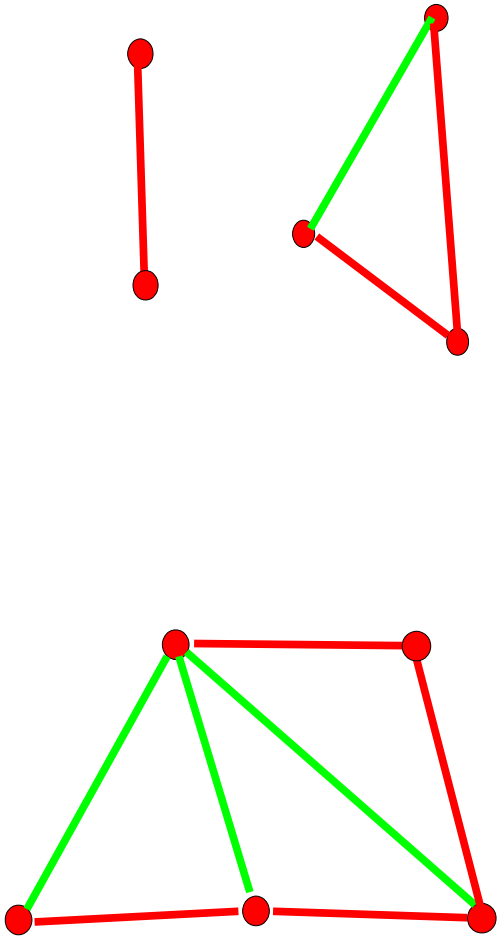
Biconnected Components



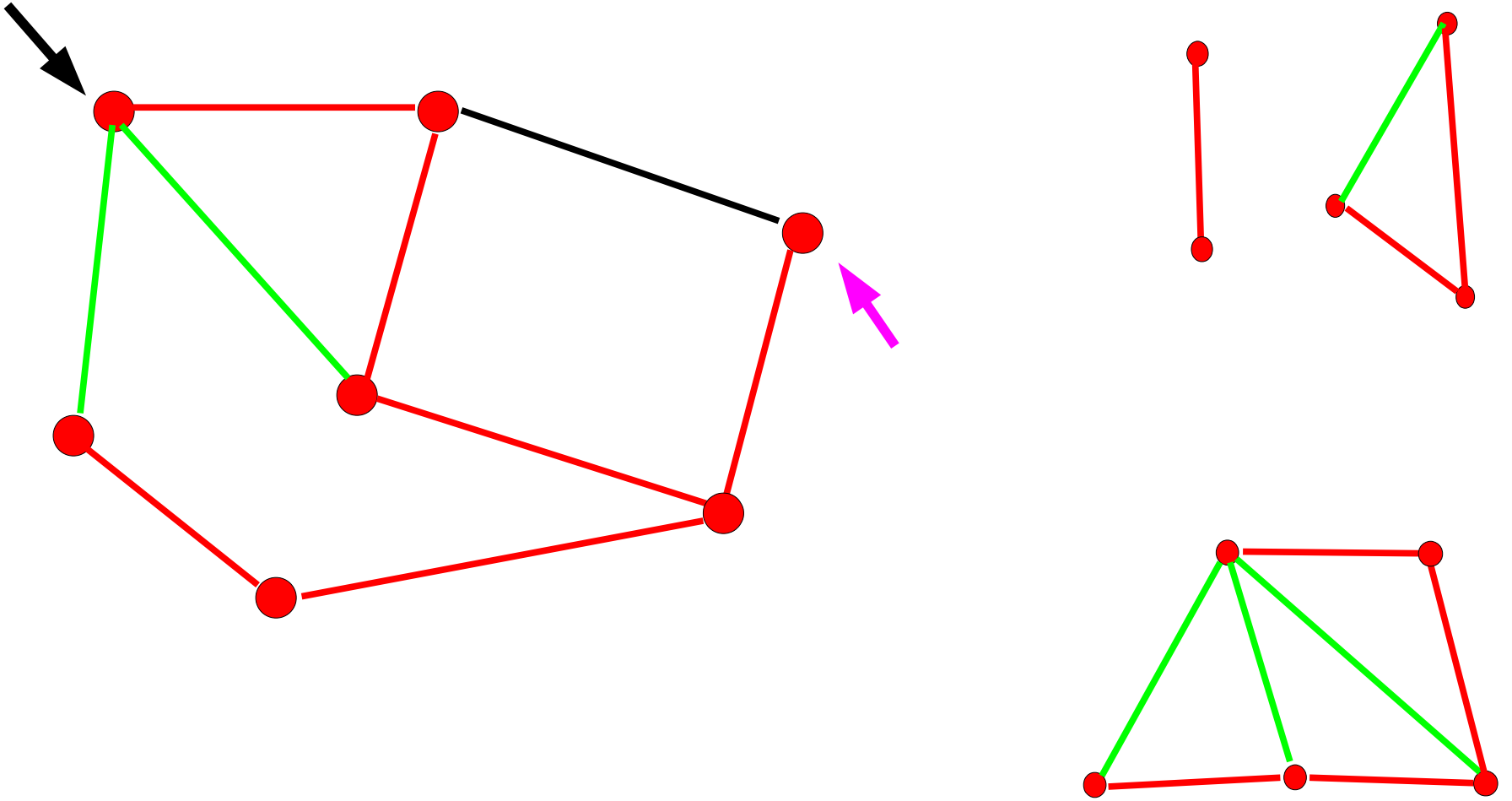


Biconnected Components

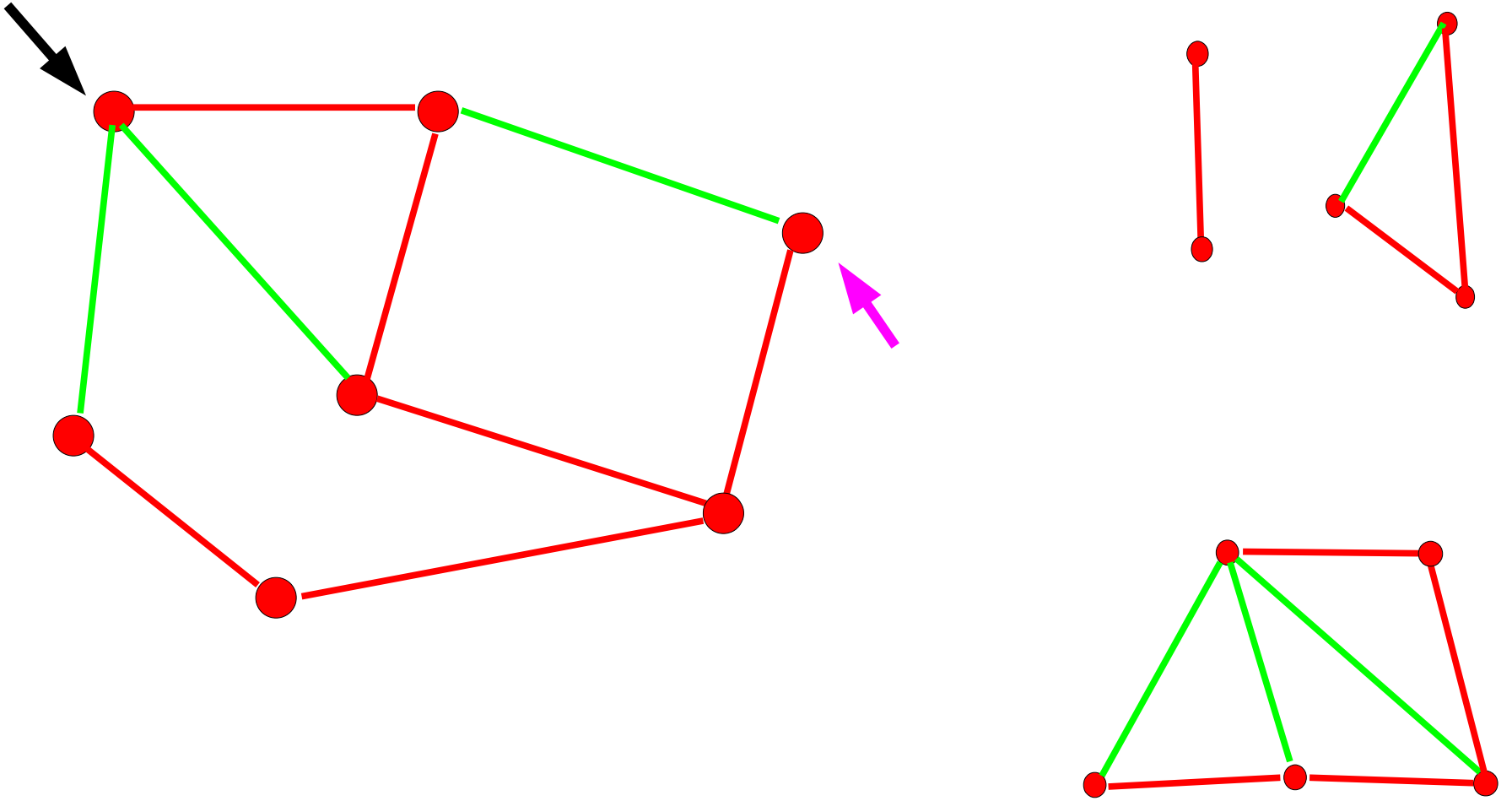




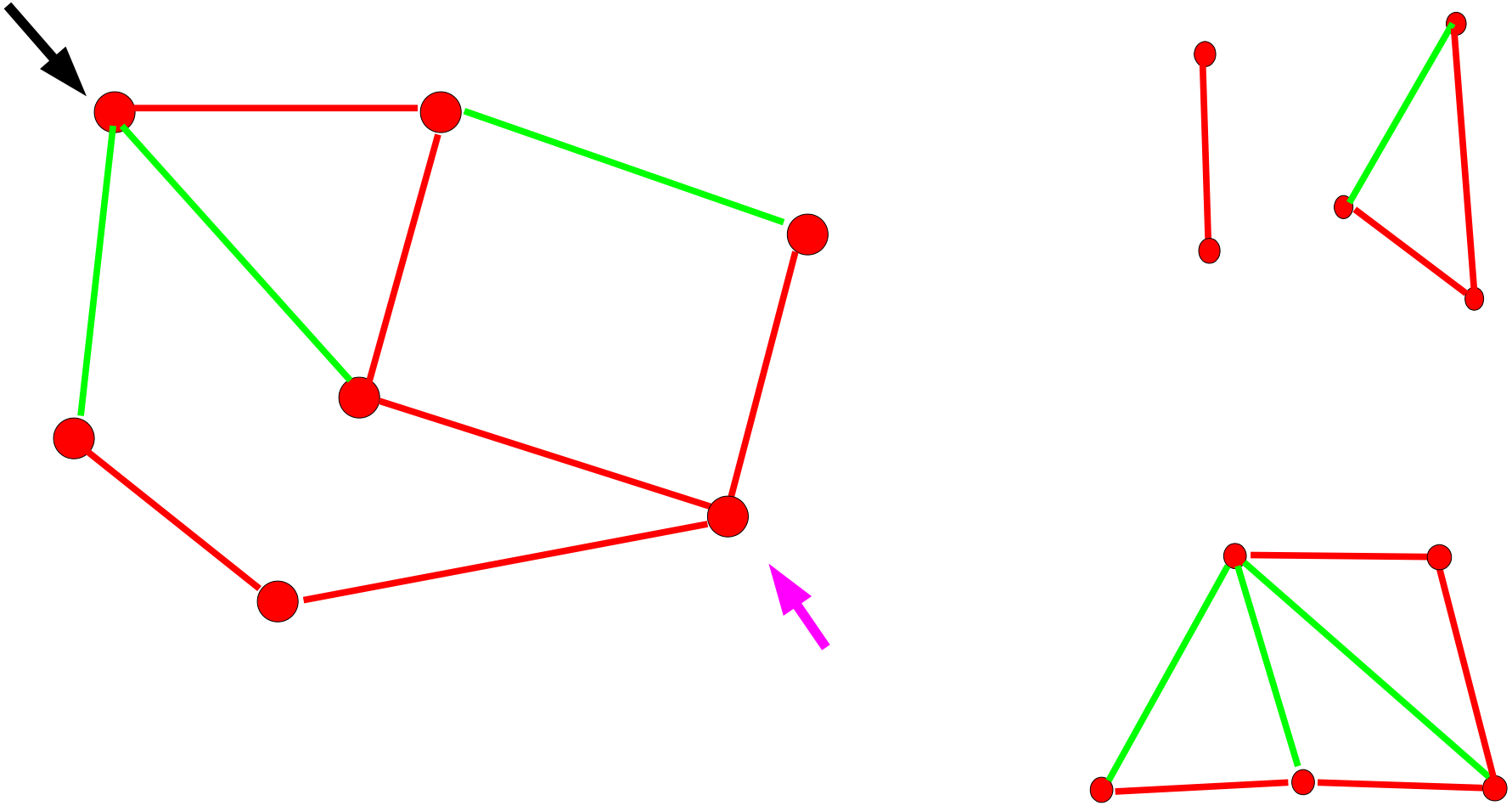
Biconnected Components



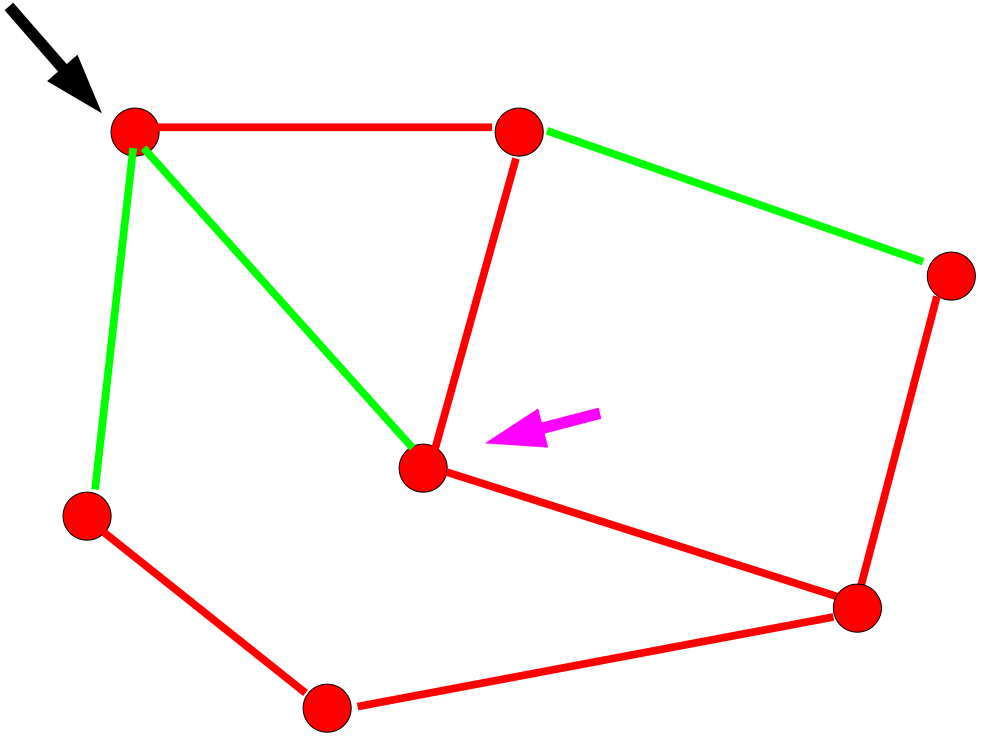
Biconnected Components

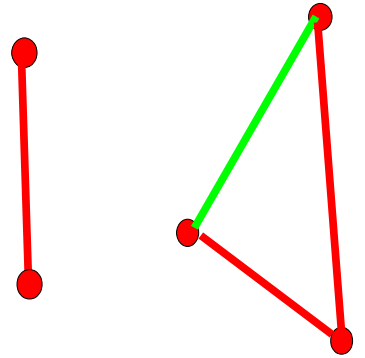


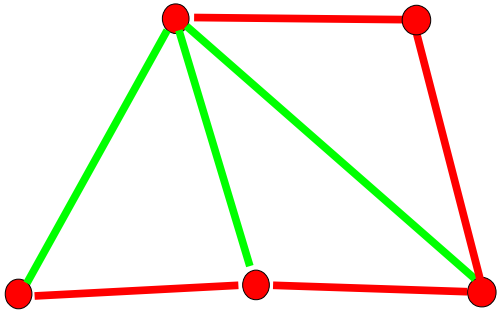
Biconnected Components



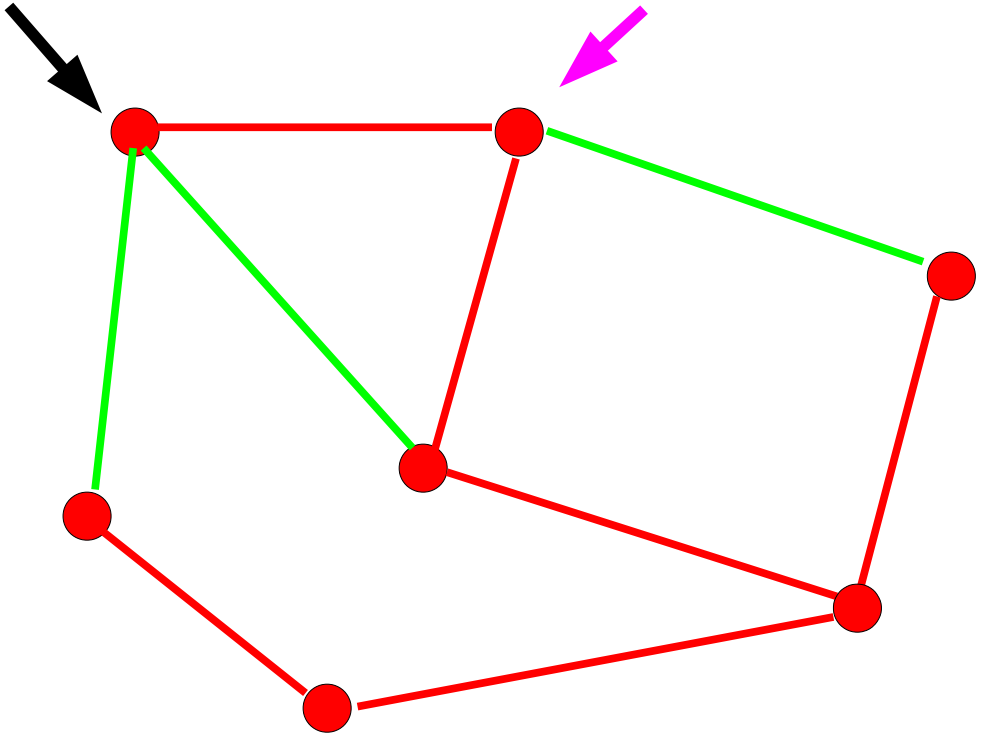
Biconnected Components

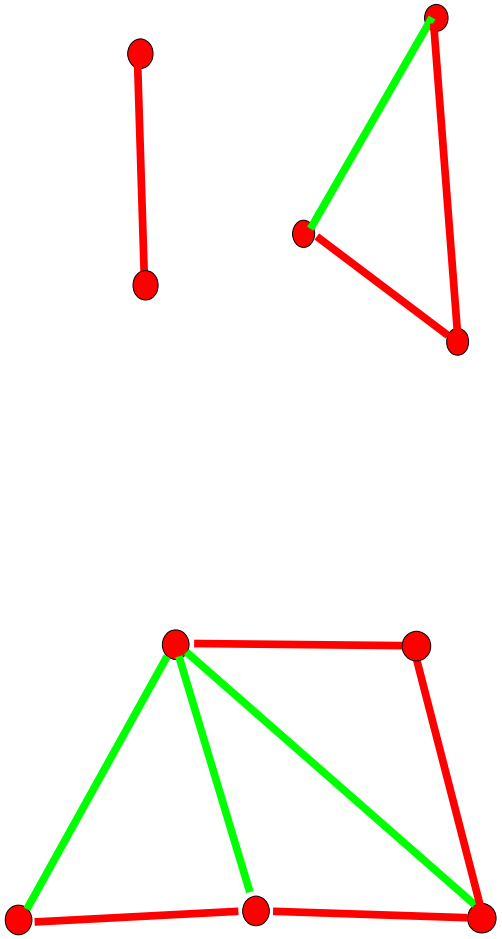




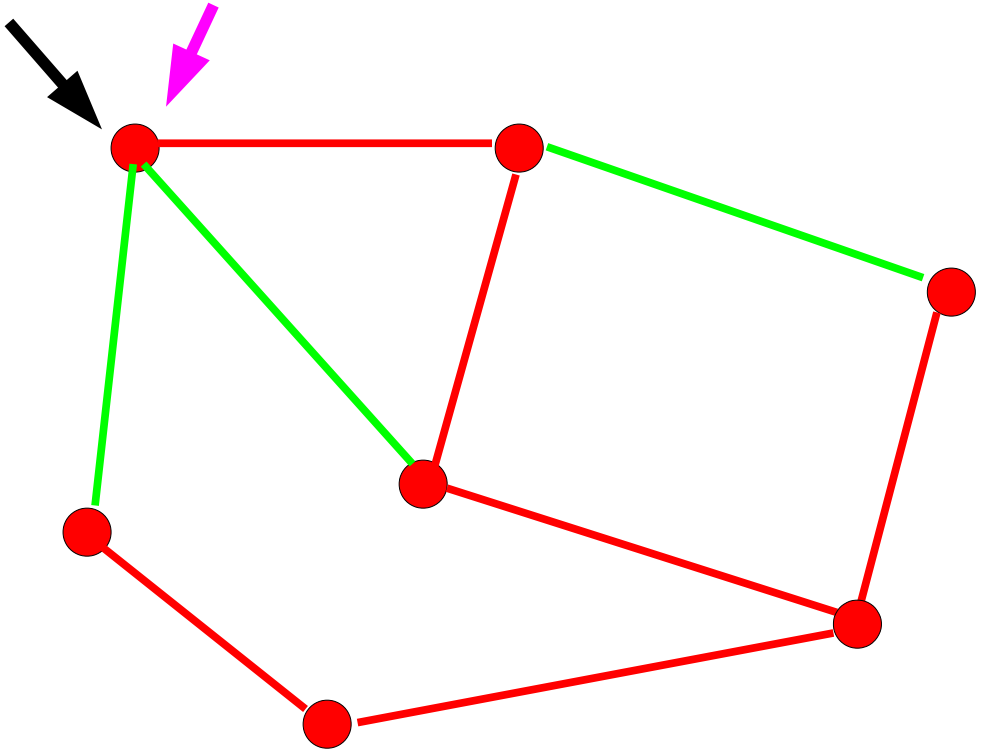


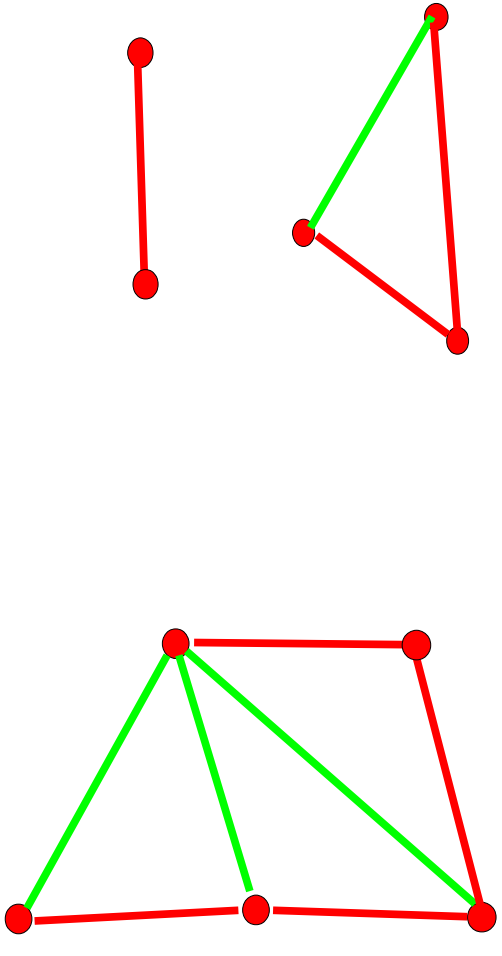
Biconnected Components



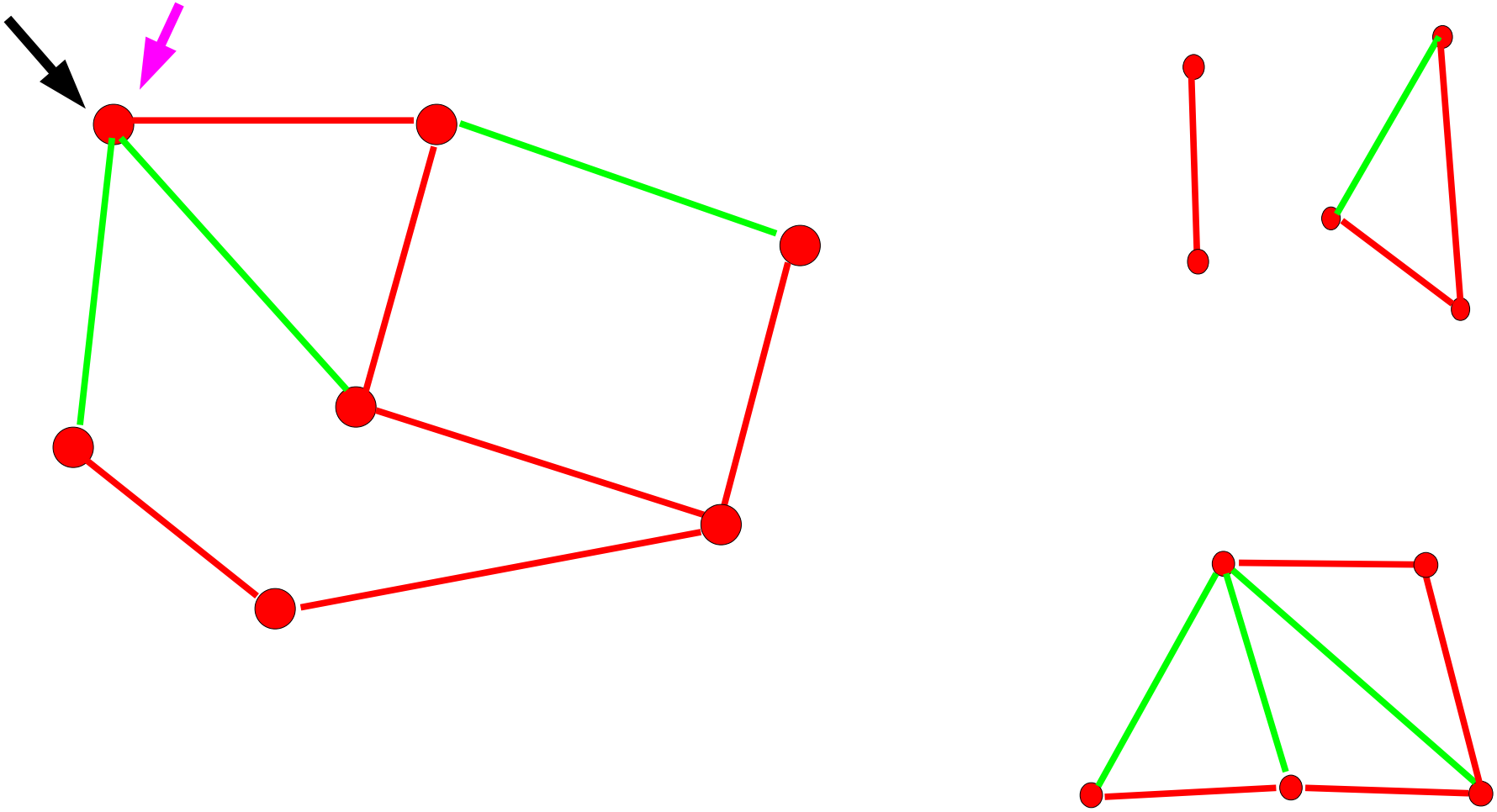


Biconnected Components



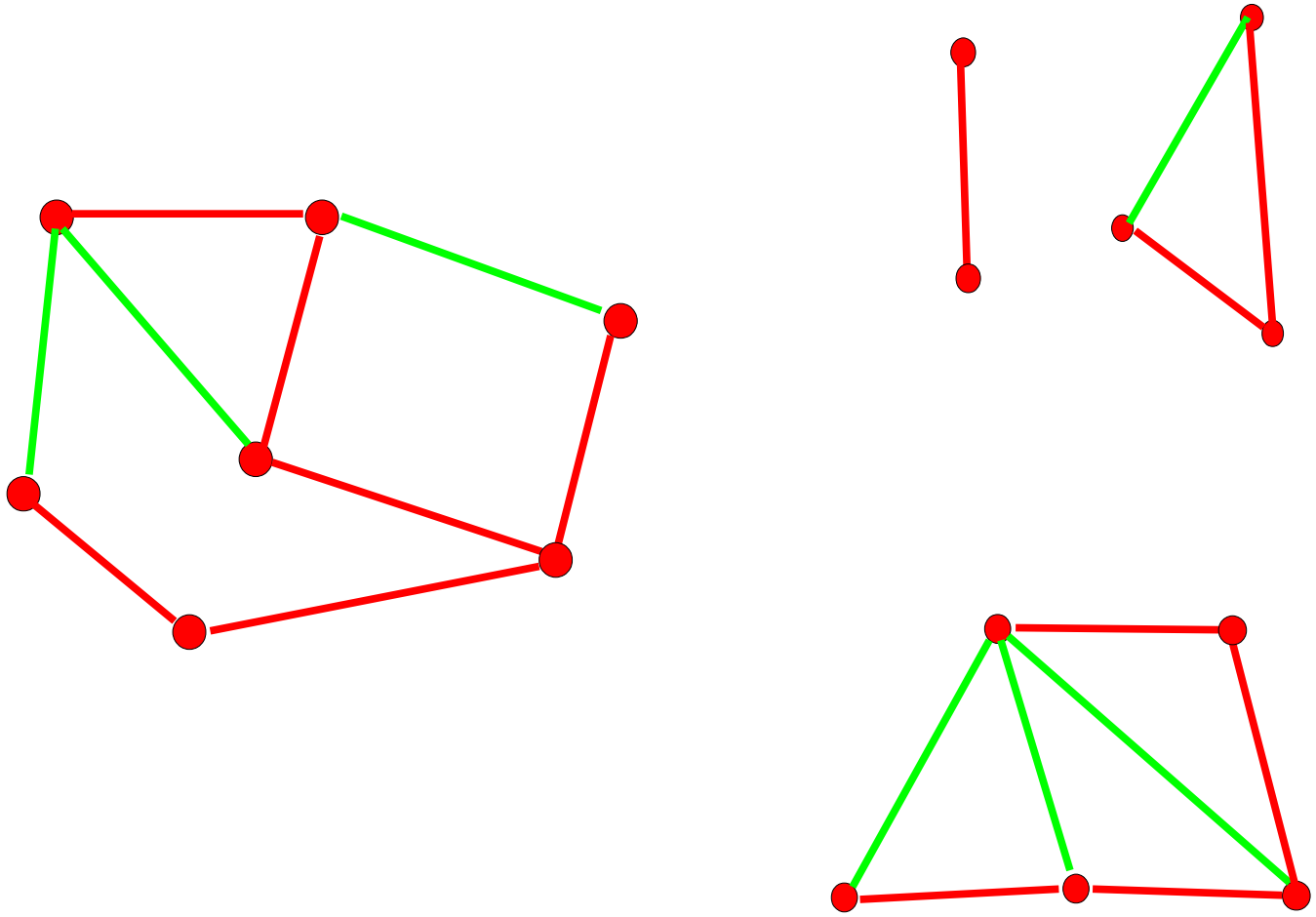


Biconnected Components



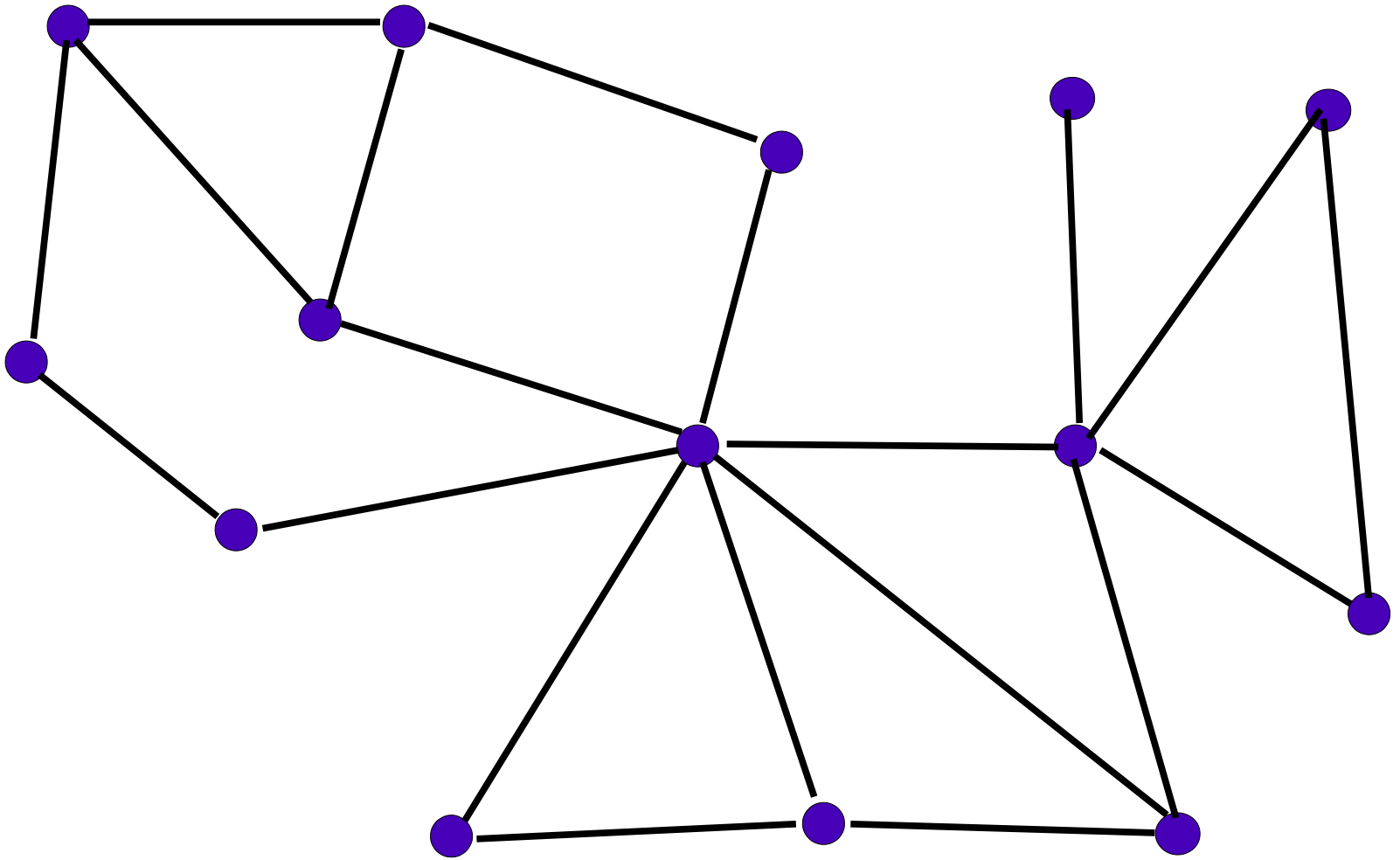
Must be a biconnected component since no more neighbors to explore at start point

Biconnected Components



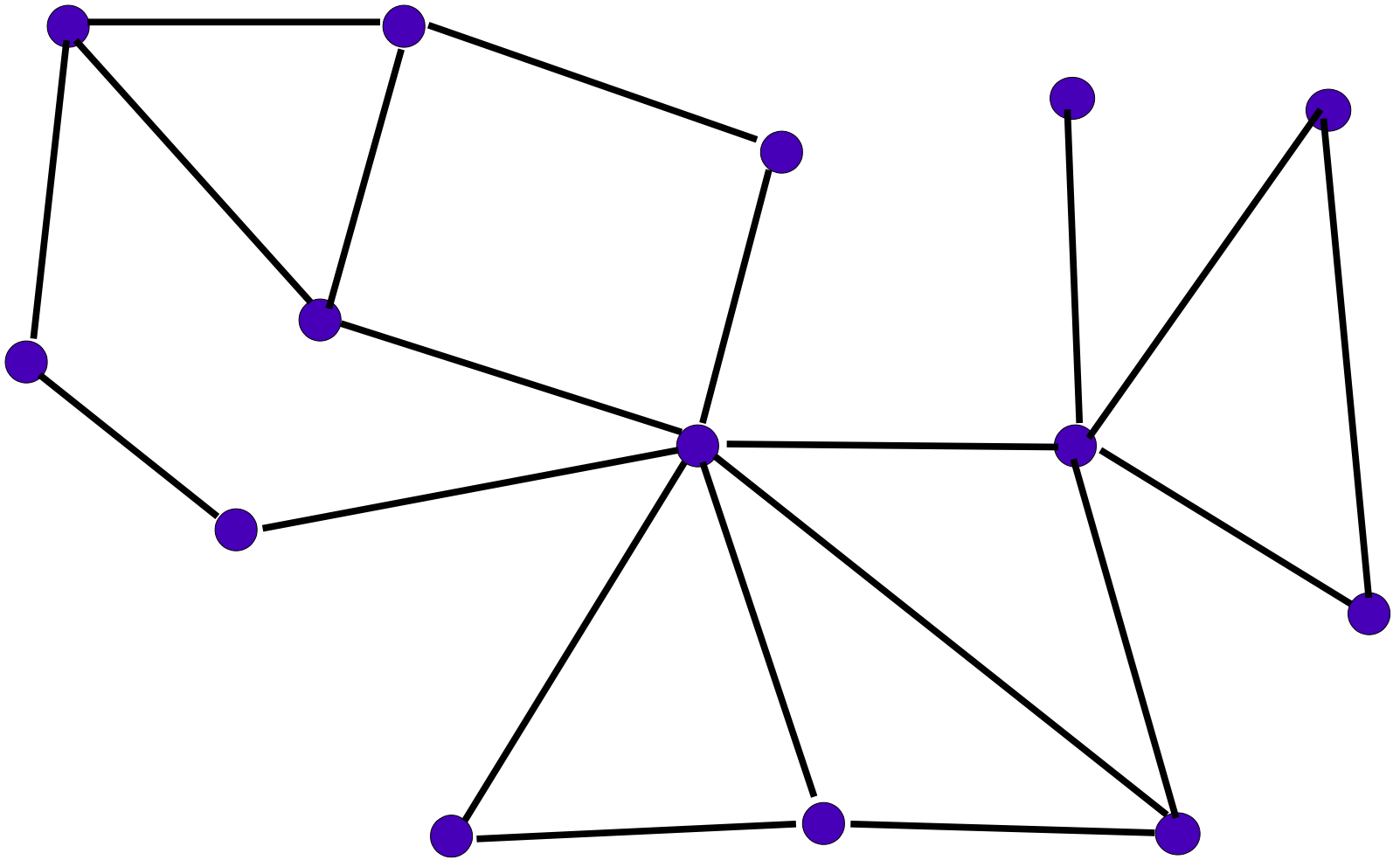
Save that biconnected component on the side, return all biconnected components, stop

Biconnected Components



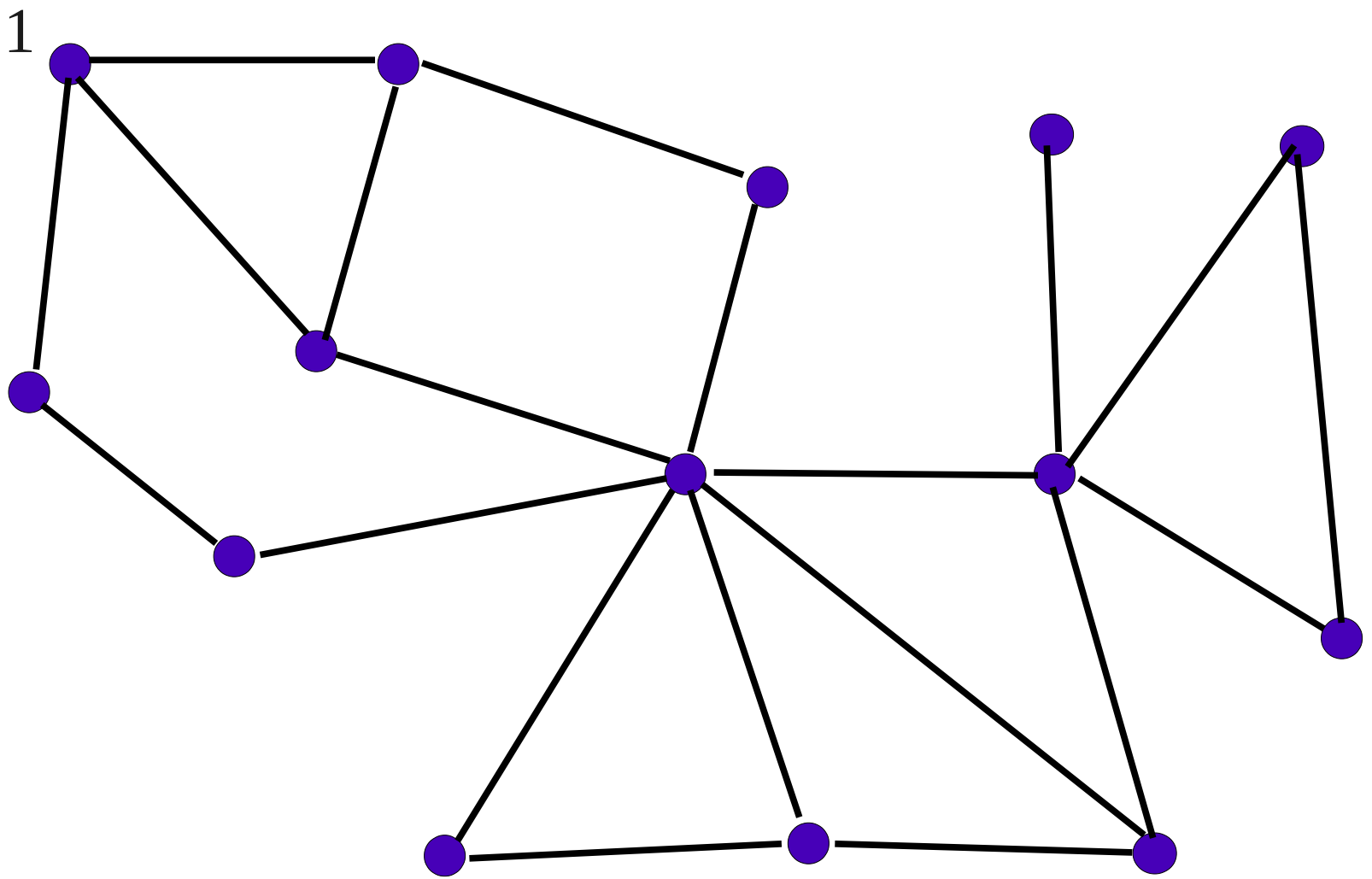
How to detect, during depth-first search, that all paths from neighbors explored so far lead back to the current vertex?

Biconnected Components



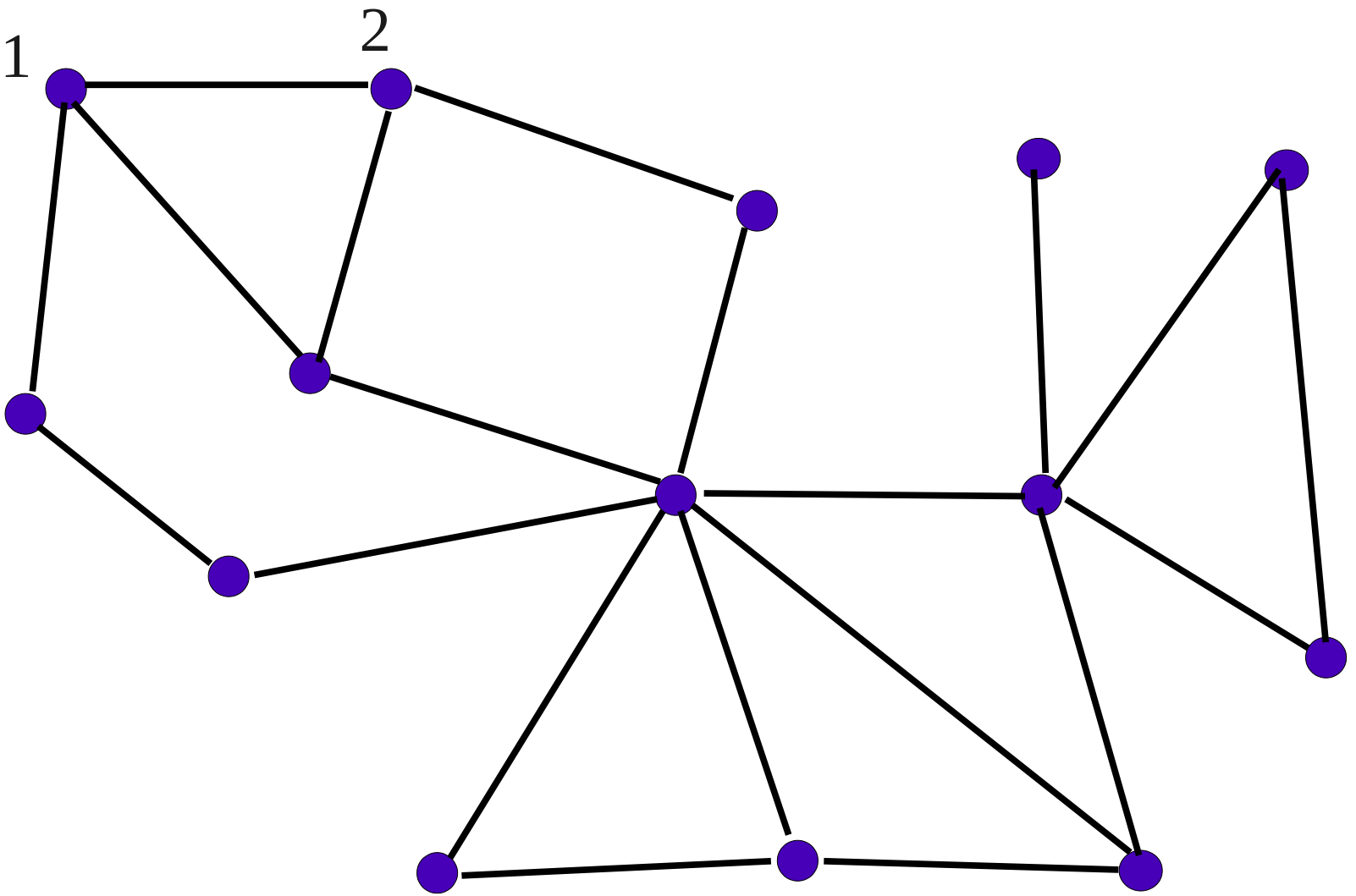
Maybe it has something to do with the order that vertices are considered – let's try putting numbers on the vertices

Biconnected Components

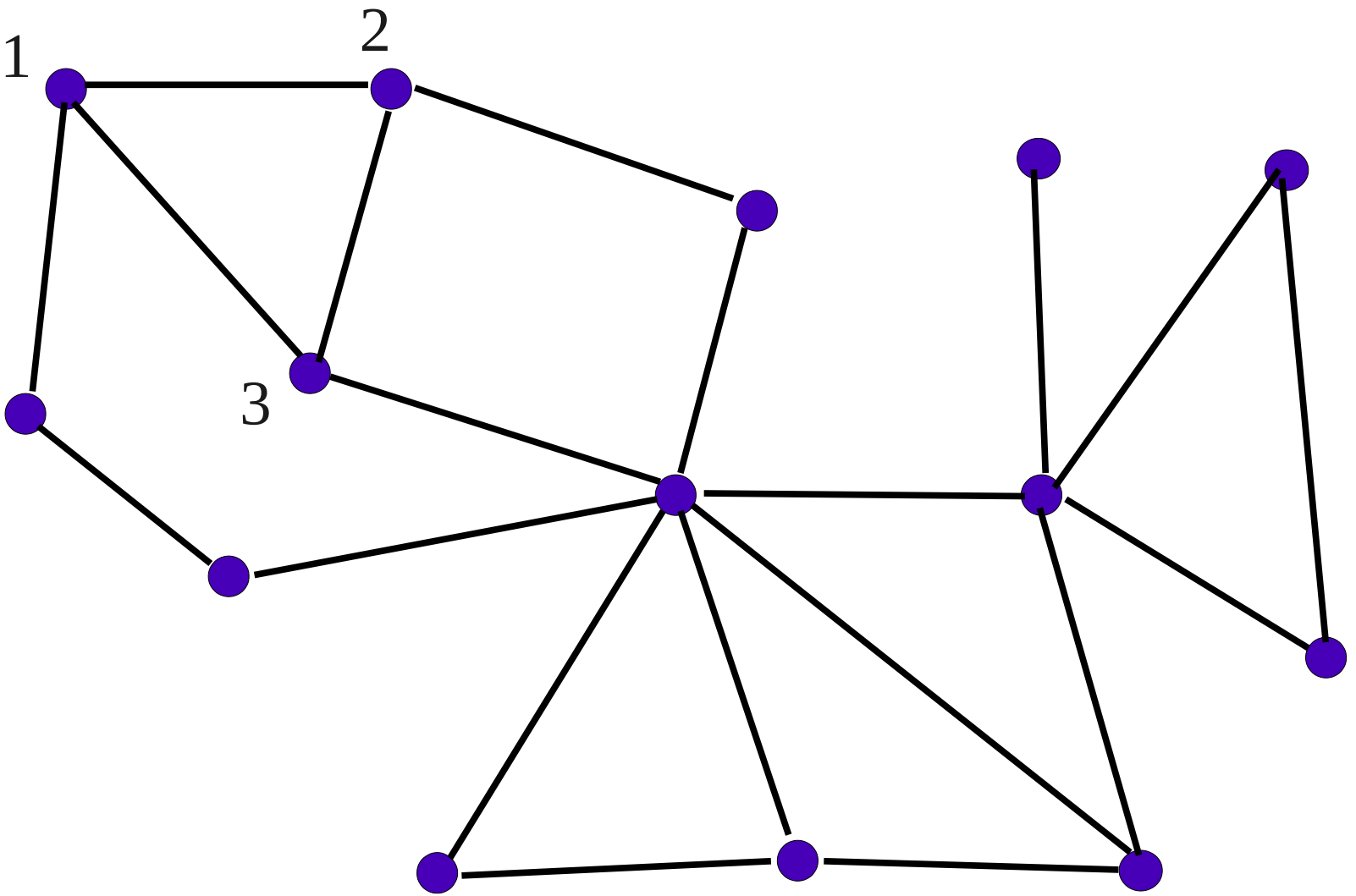


Maybe it has something to do with the order that vertices are considered – let's try putting numbers on the vertices

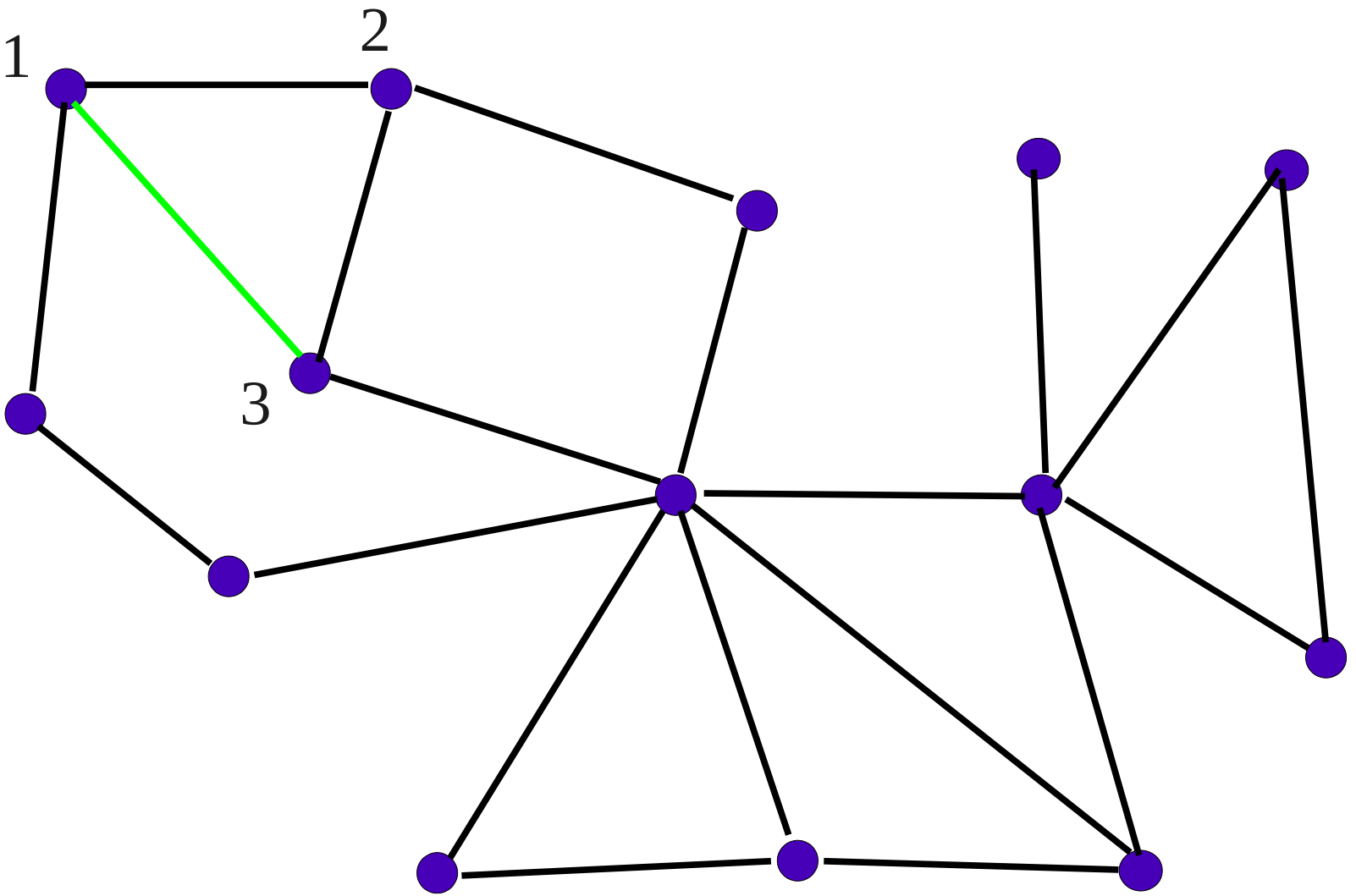
Biconnected Components



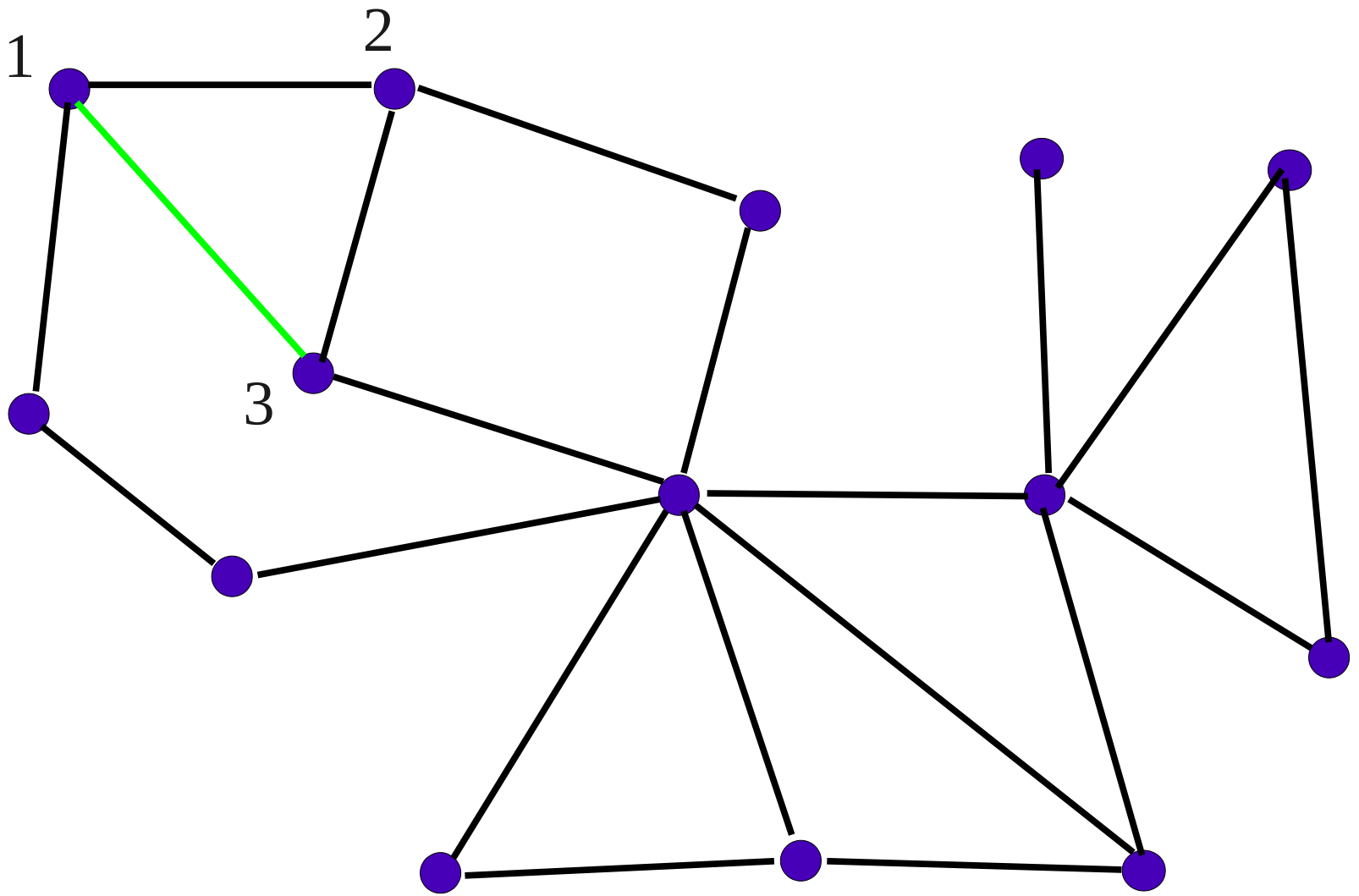
Biconnected Components



Biconnected Components

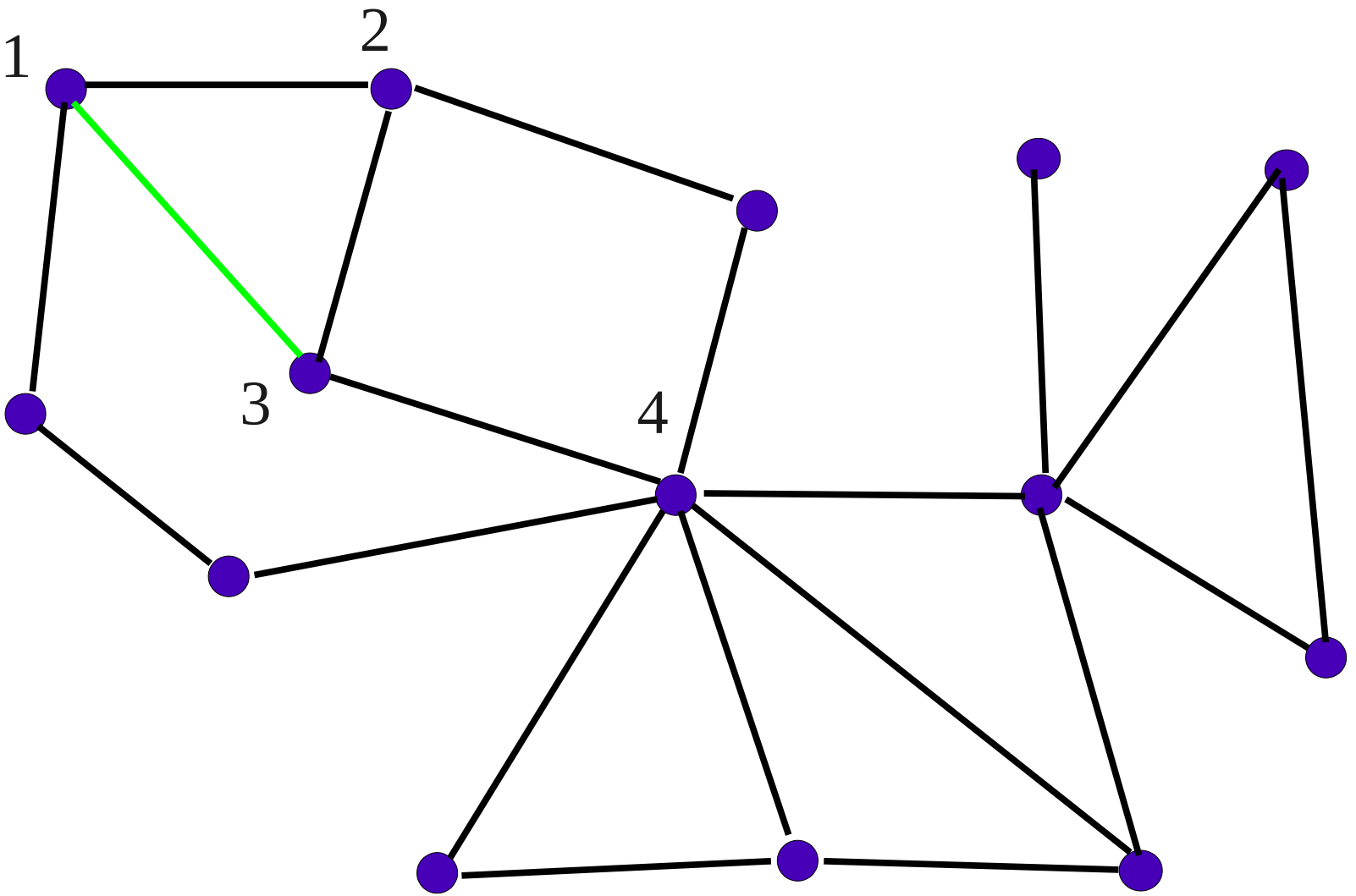


Biconnected Components

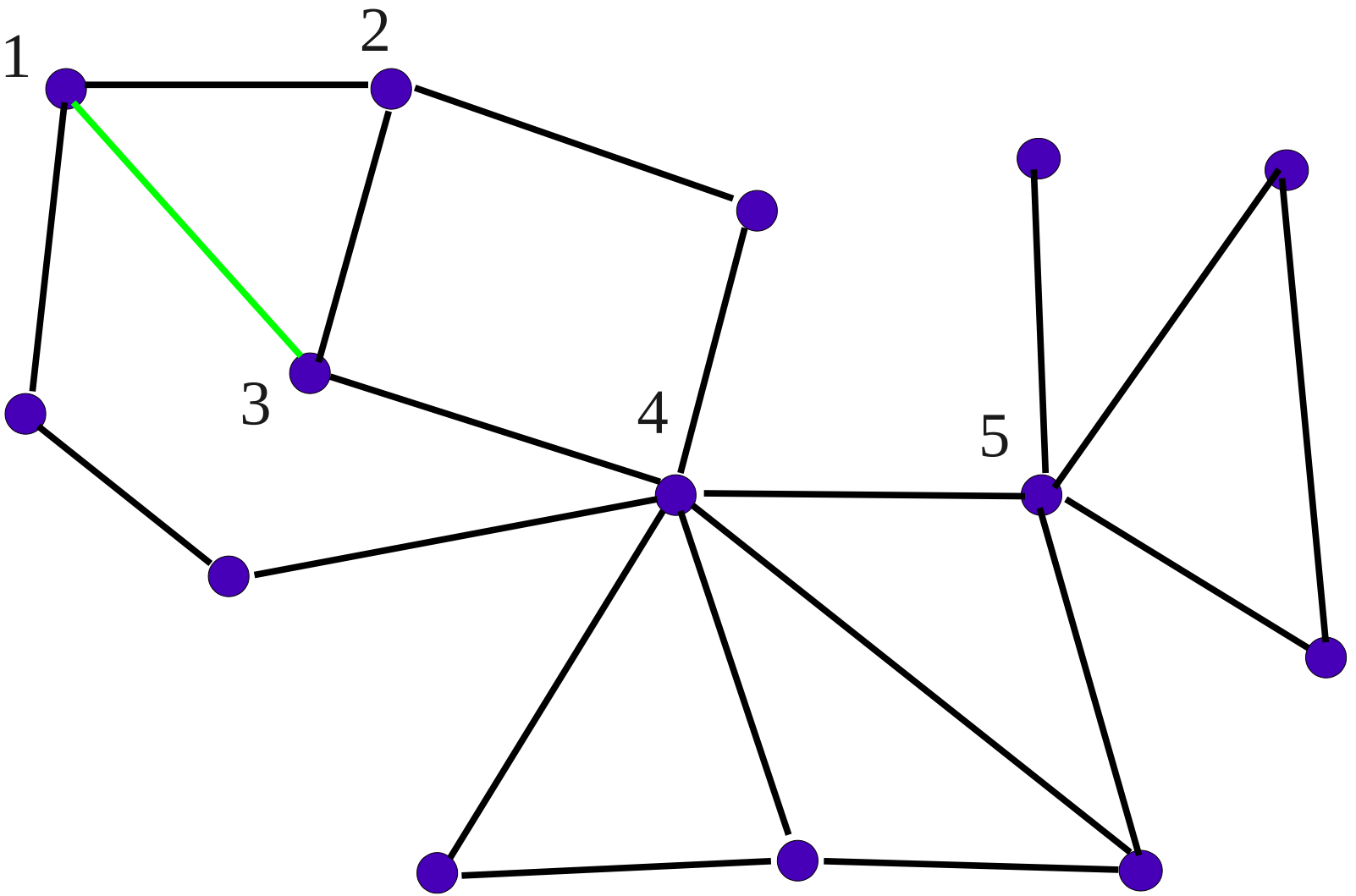


We arrive back at vertex 3 having seen vertex 1. So edges we have seen are not a biconnected component.

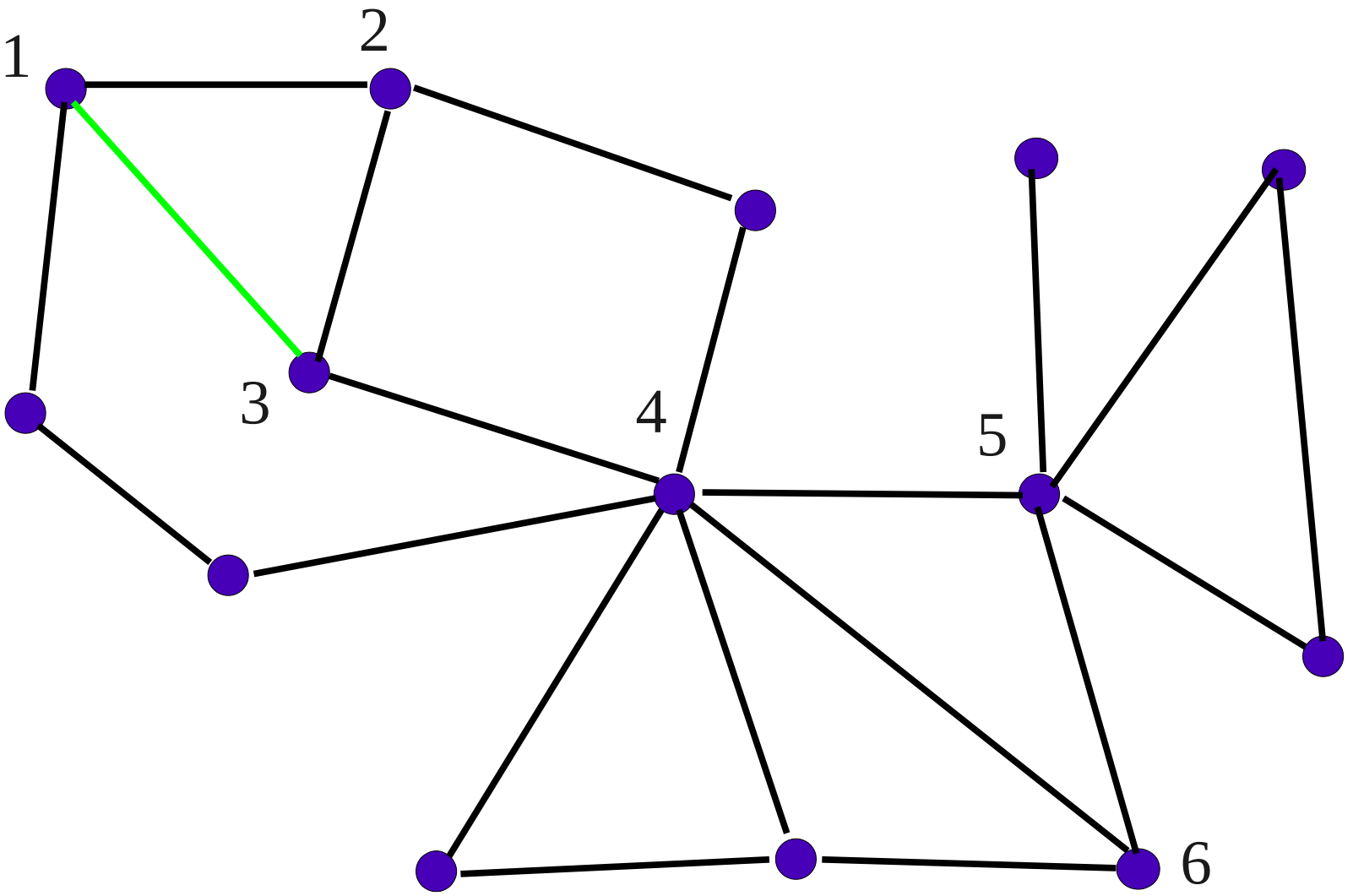
Biconnected Components



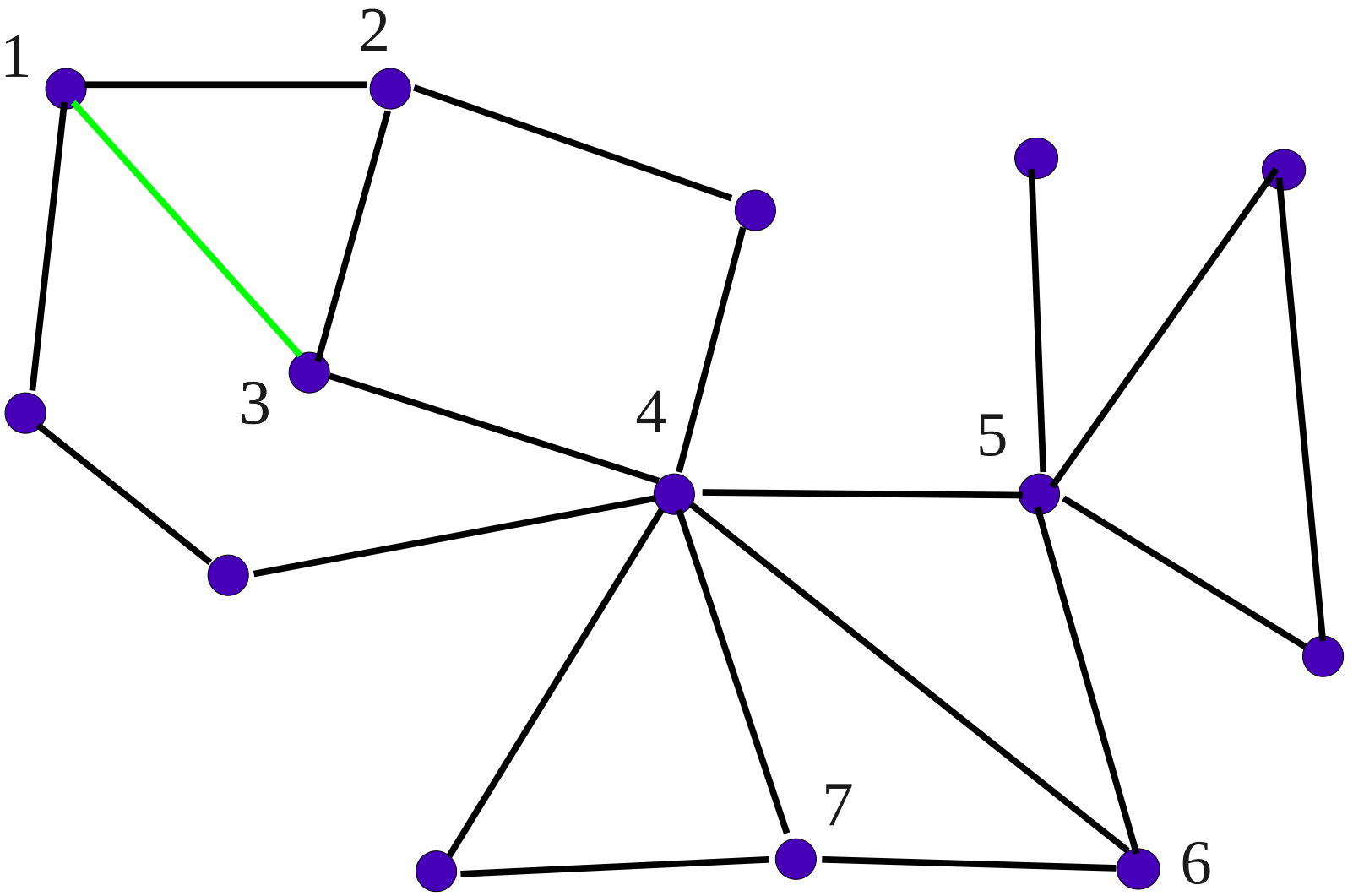
Biconnected Components



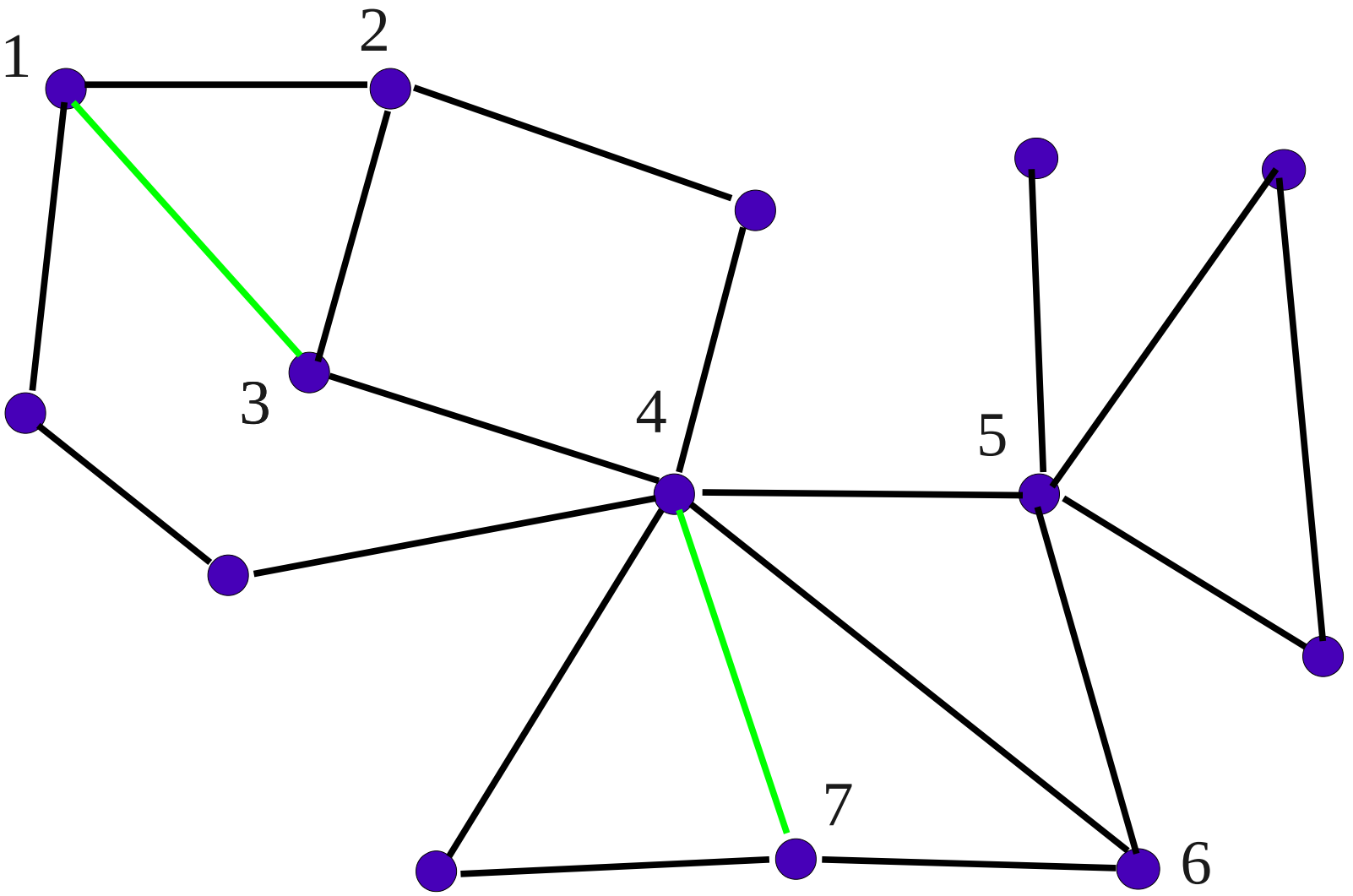
Biconnected Components



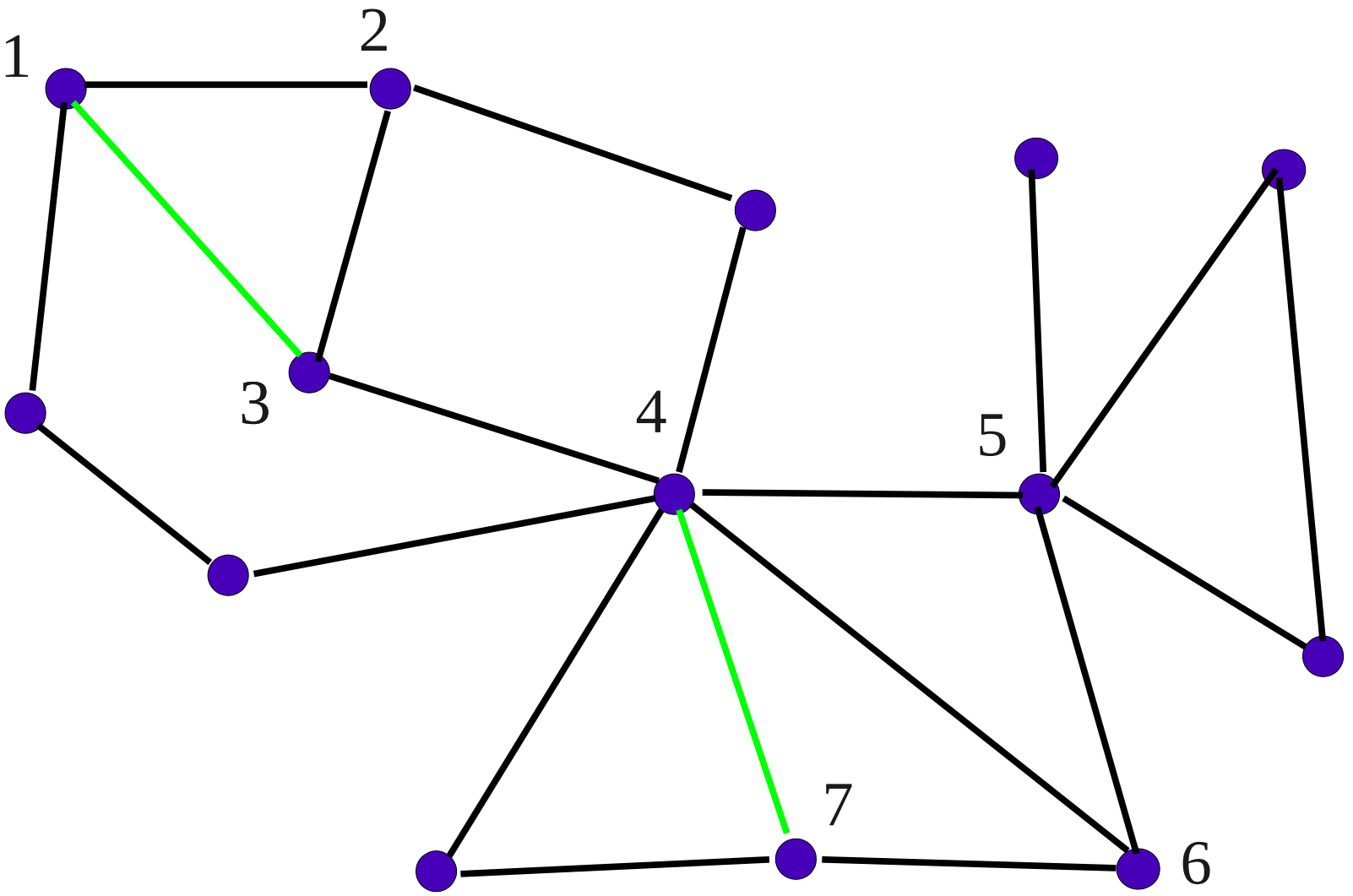
Biconnected Components



Biconnected Components

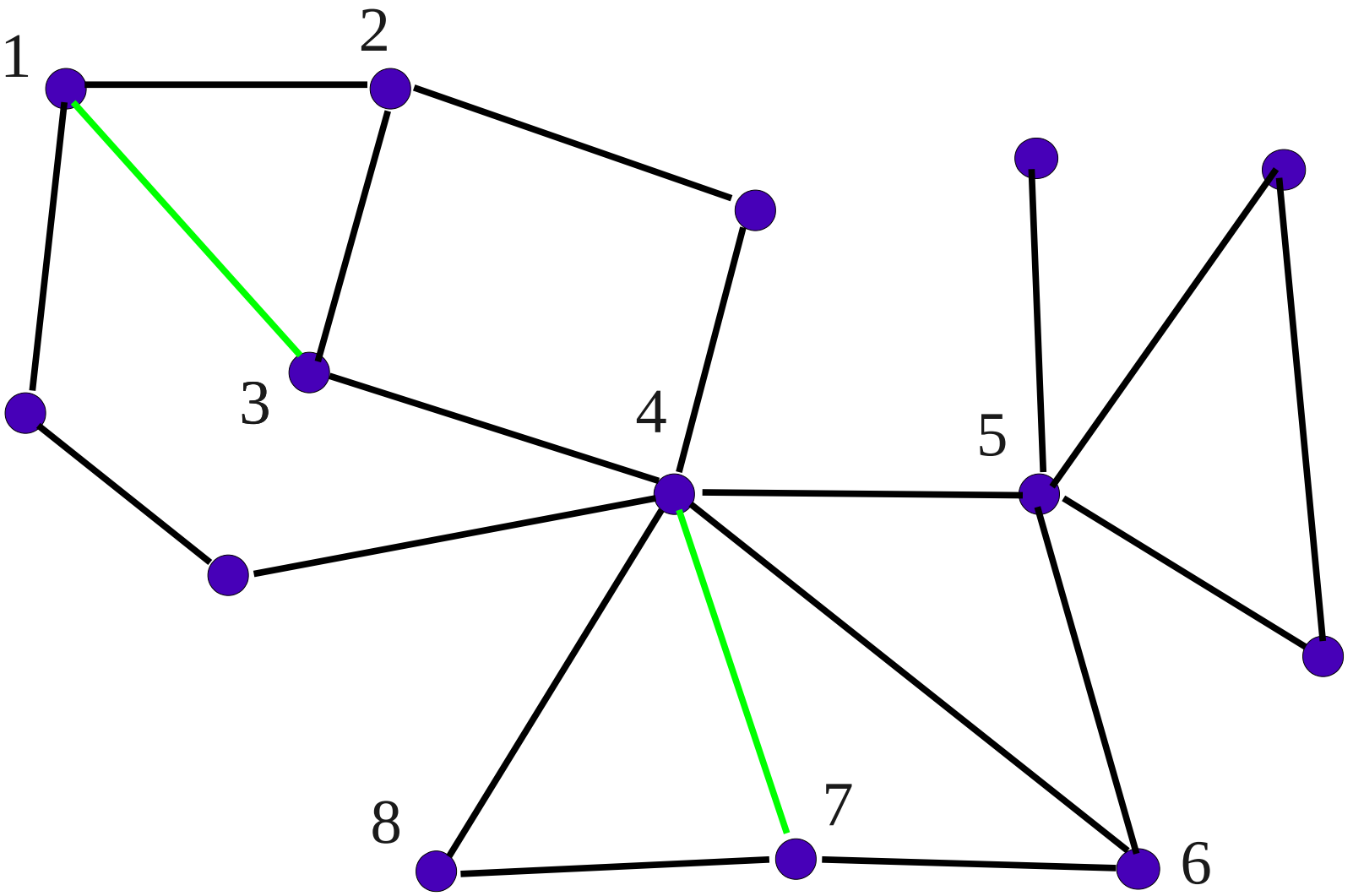


Biconnected Components

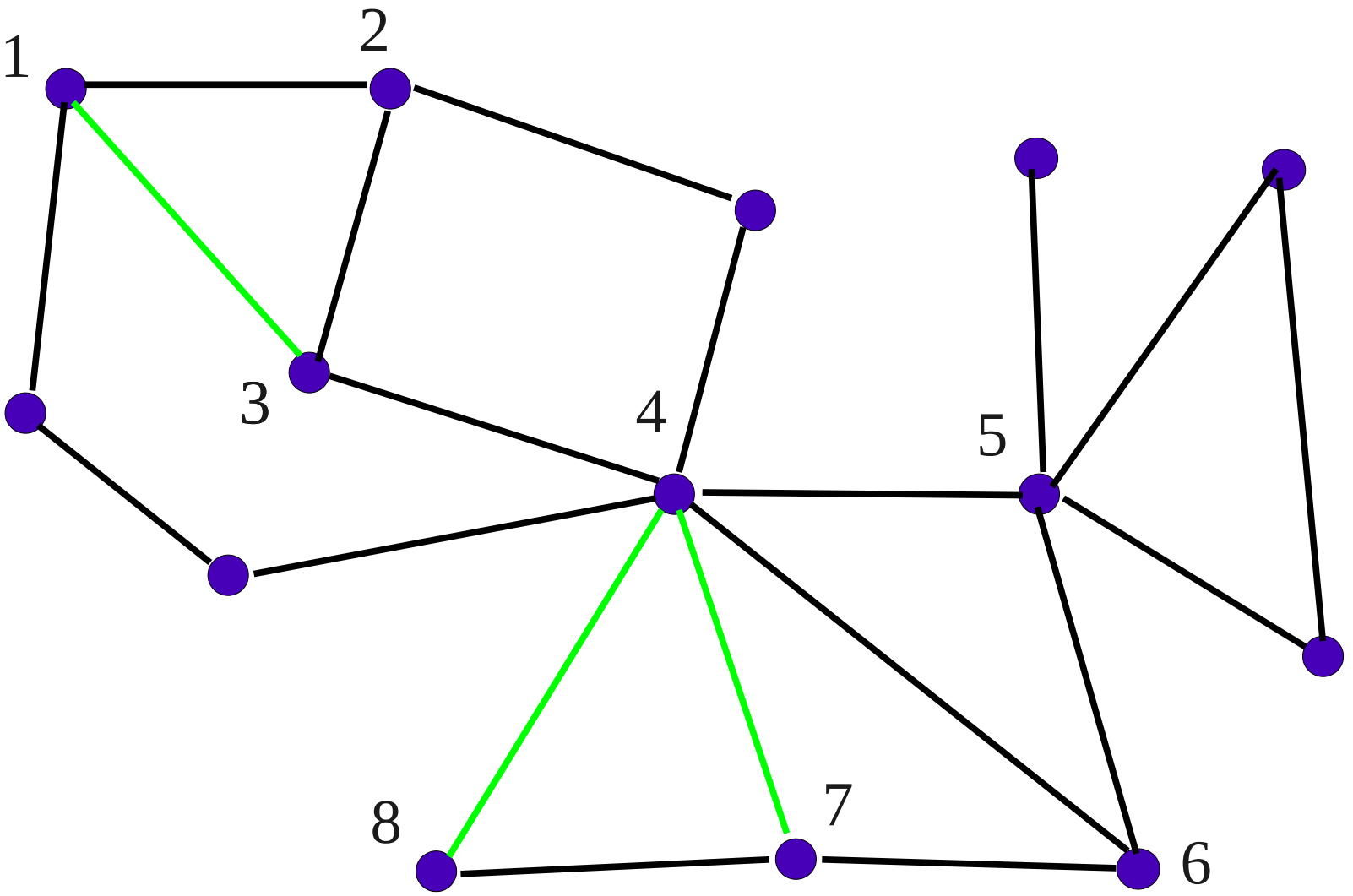


We arrive back at vertex 7 having seen vertex 4. So edges we have seen are not a biconnected component.

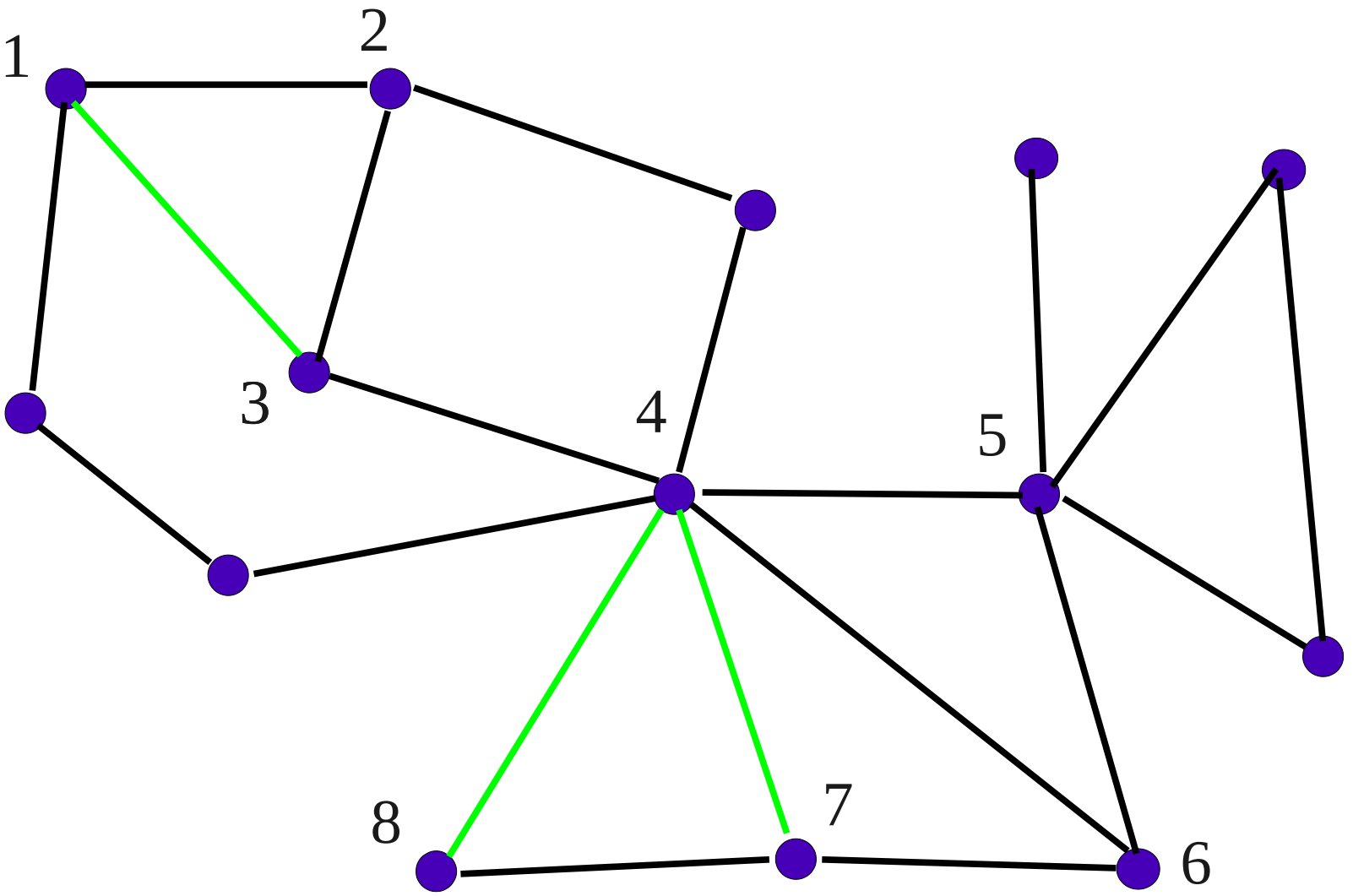
Biconnected Components



Biconnected Components

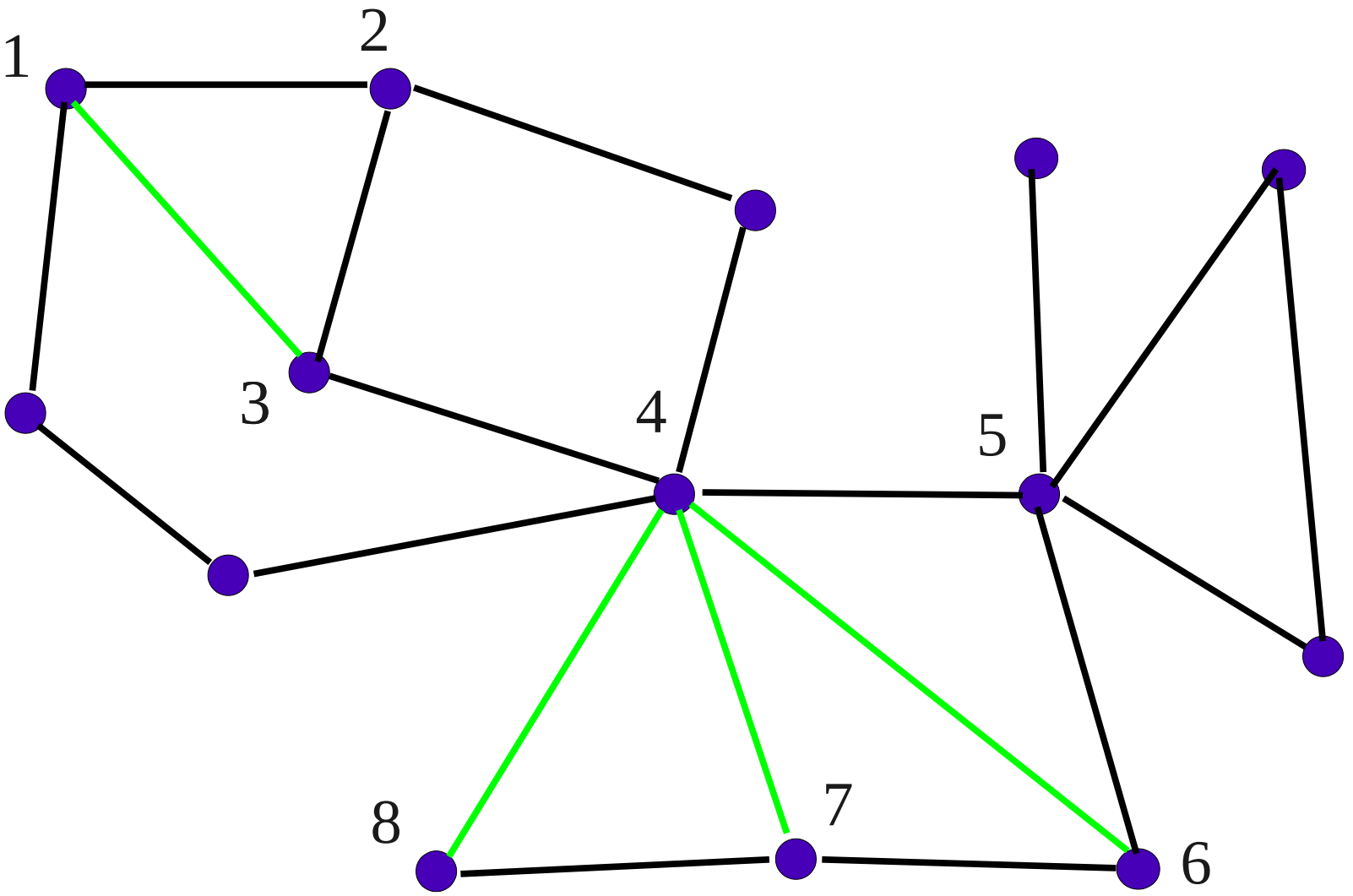


Biconnected Components

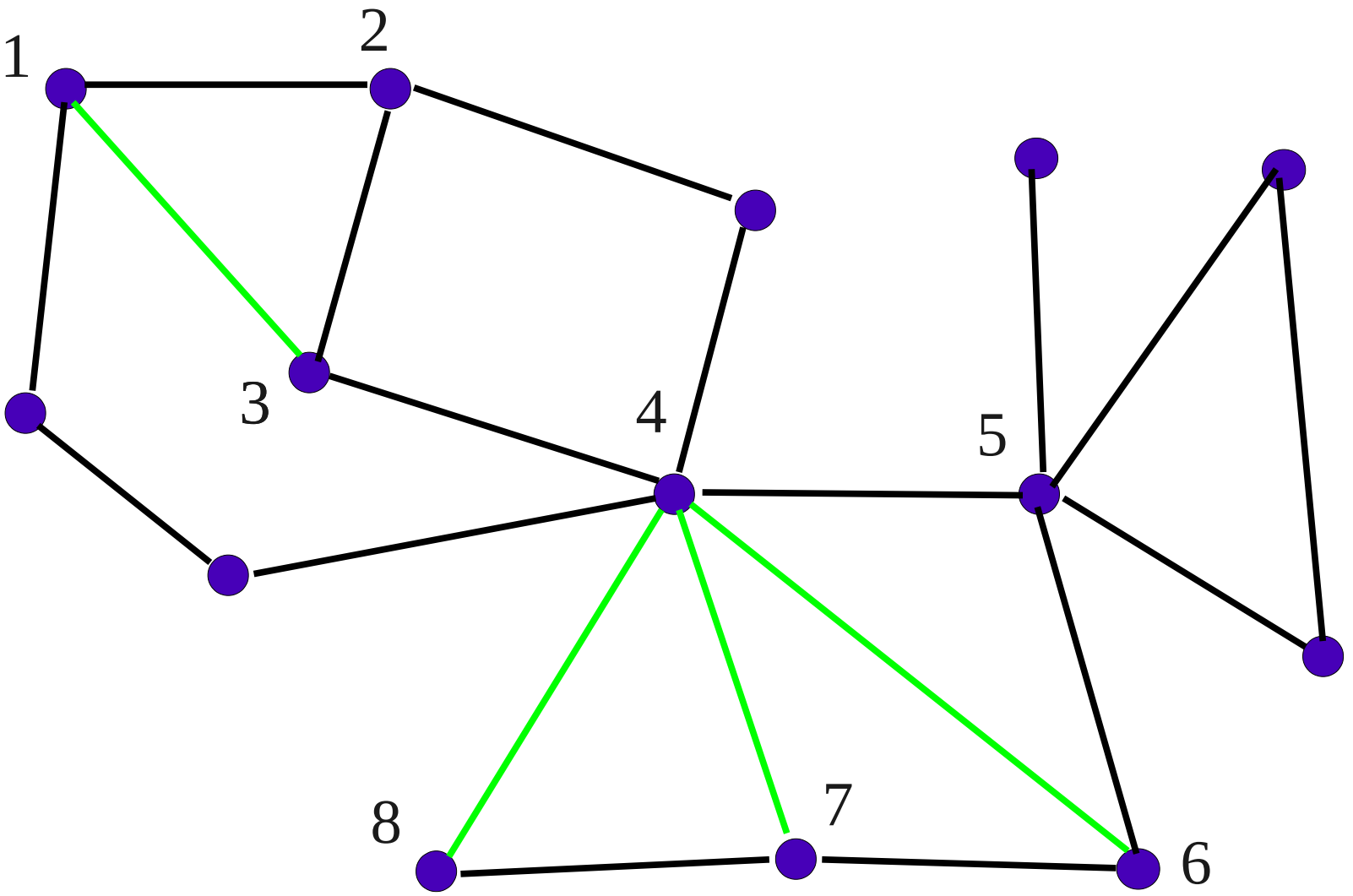


We return to vertex 8 having seen vertex 4. So this cannot be a biconnected component

Biconnected Components

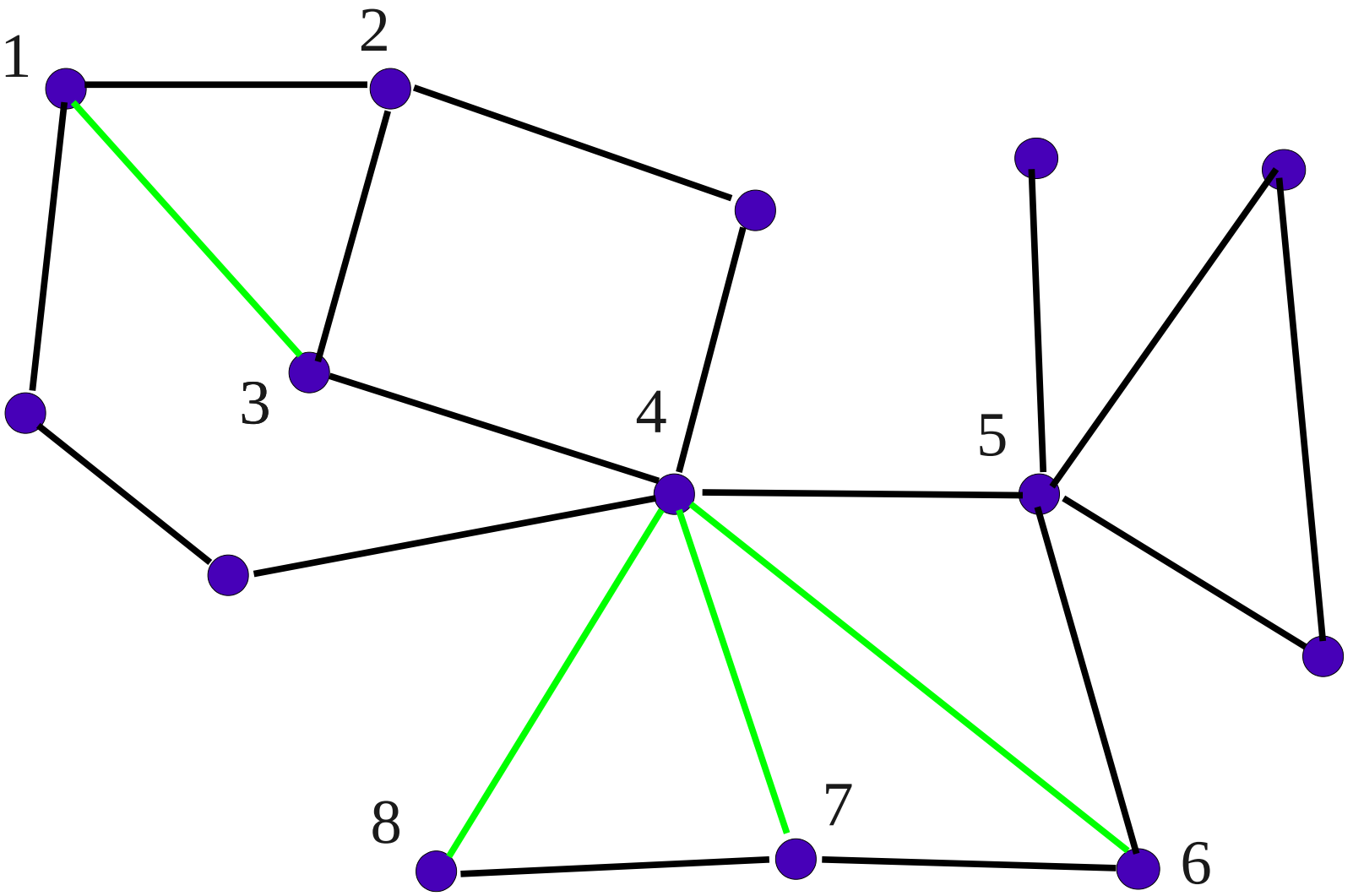


Biconnected Components



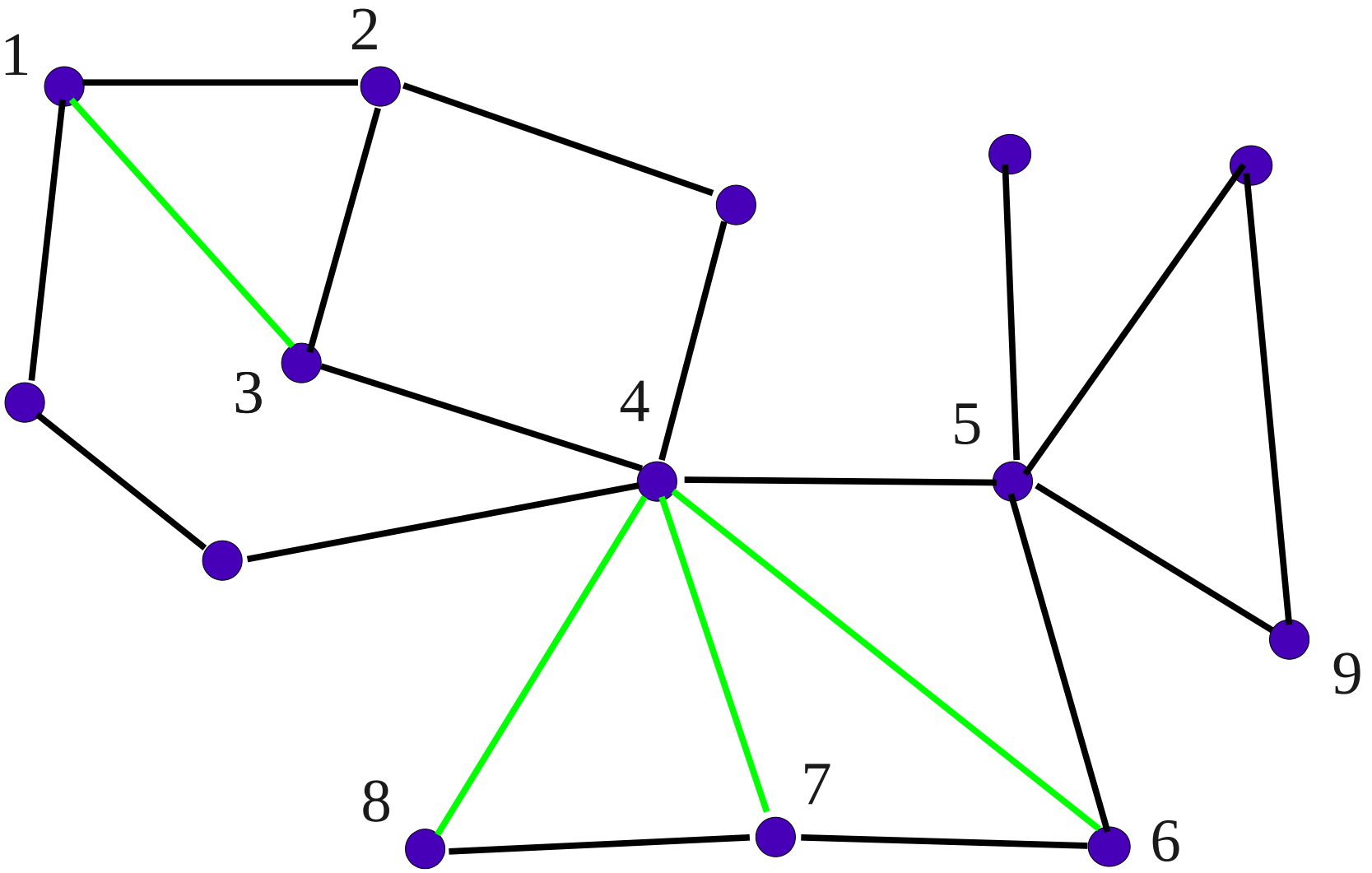
We arrive back at vertex 6 having seen vertex 4. So edges we have seen are not a biconnected component.

Biconnected Components

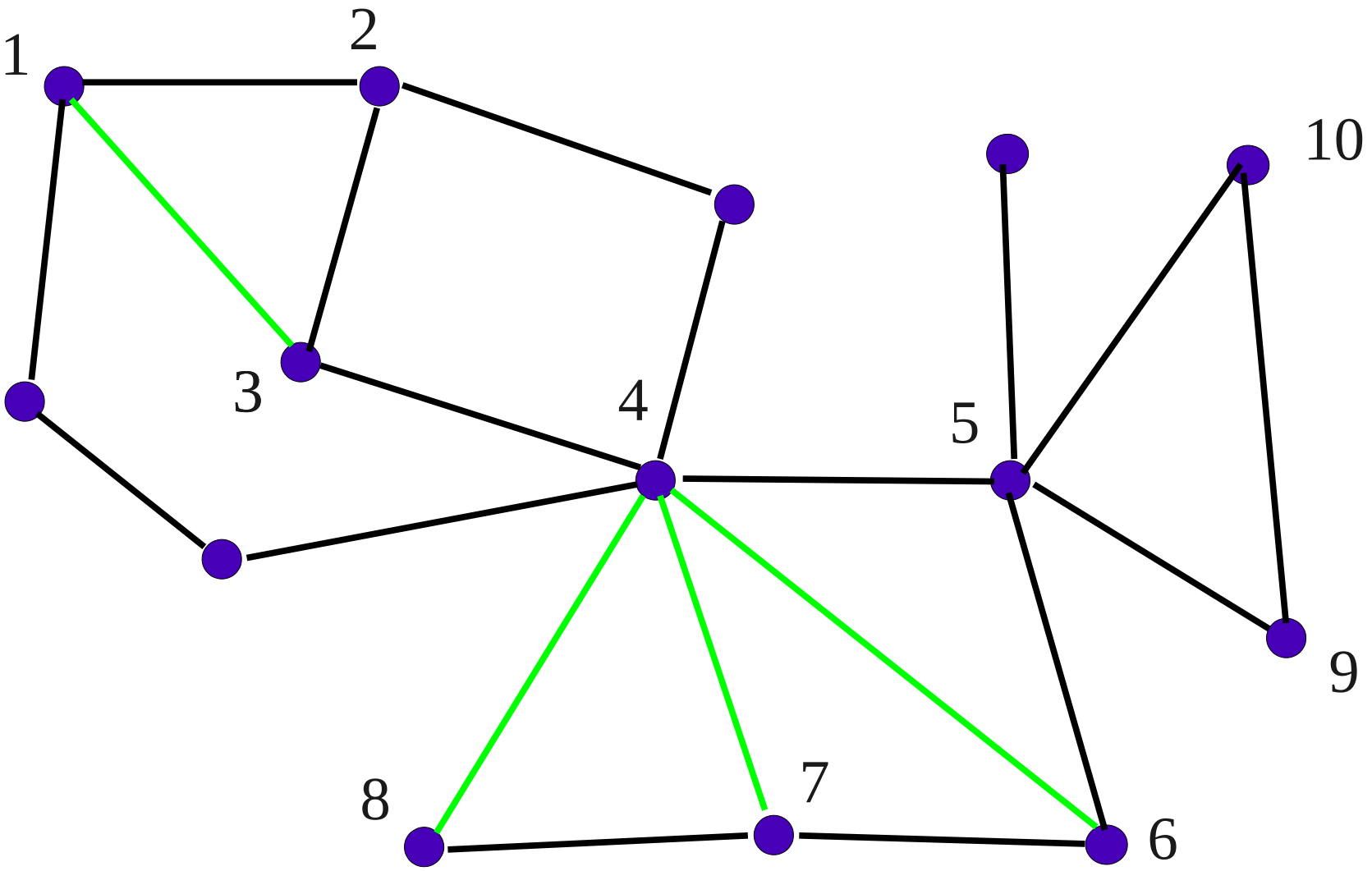


We arrive back at vertex 5 having seen vertex 4. So edges we have seen are not a biconnected component.

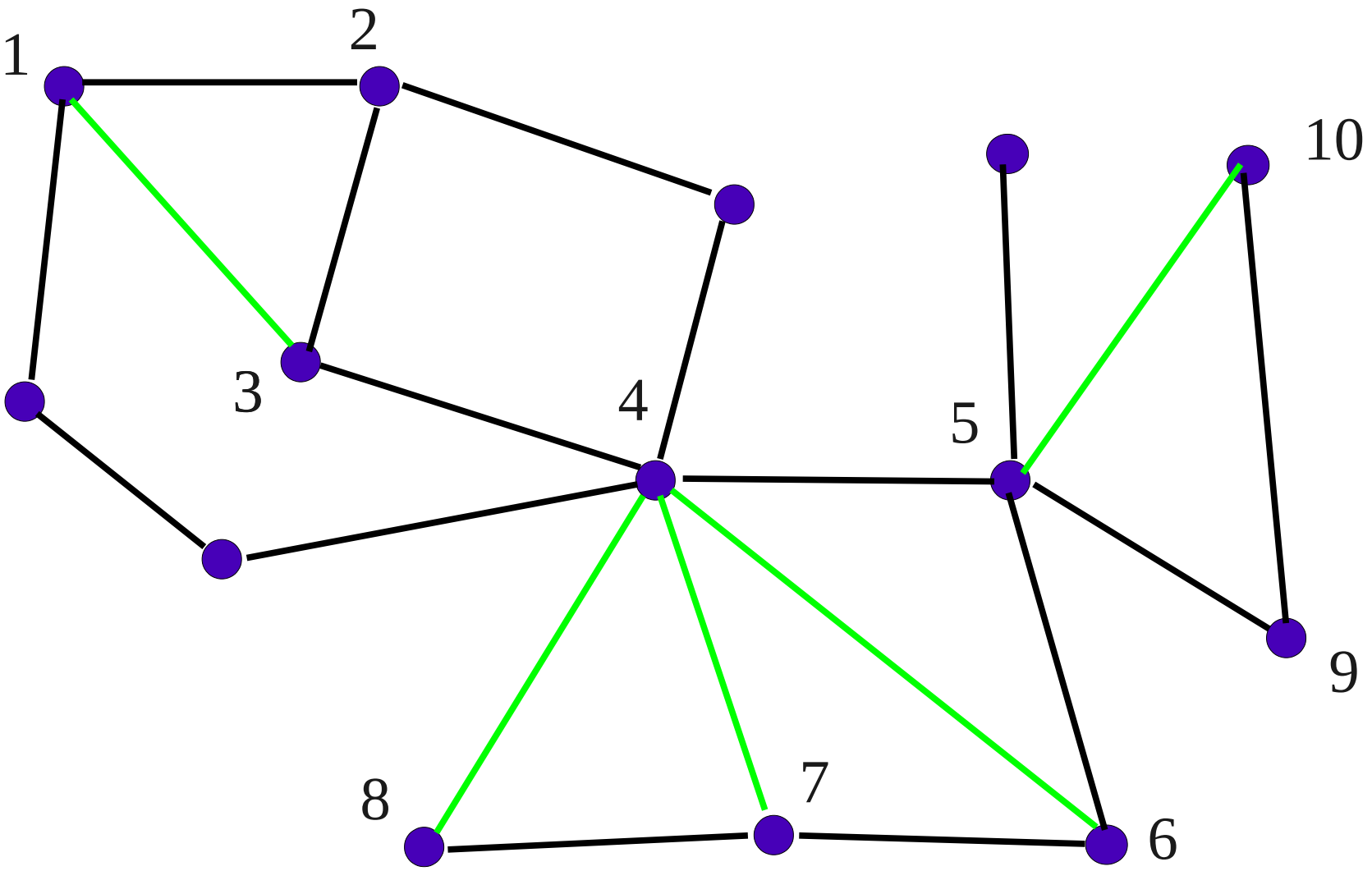
Biconnected Components



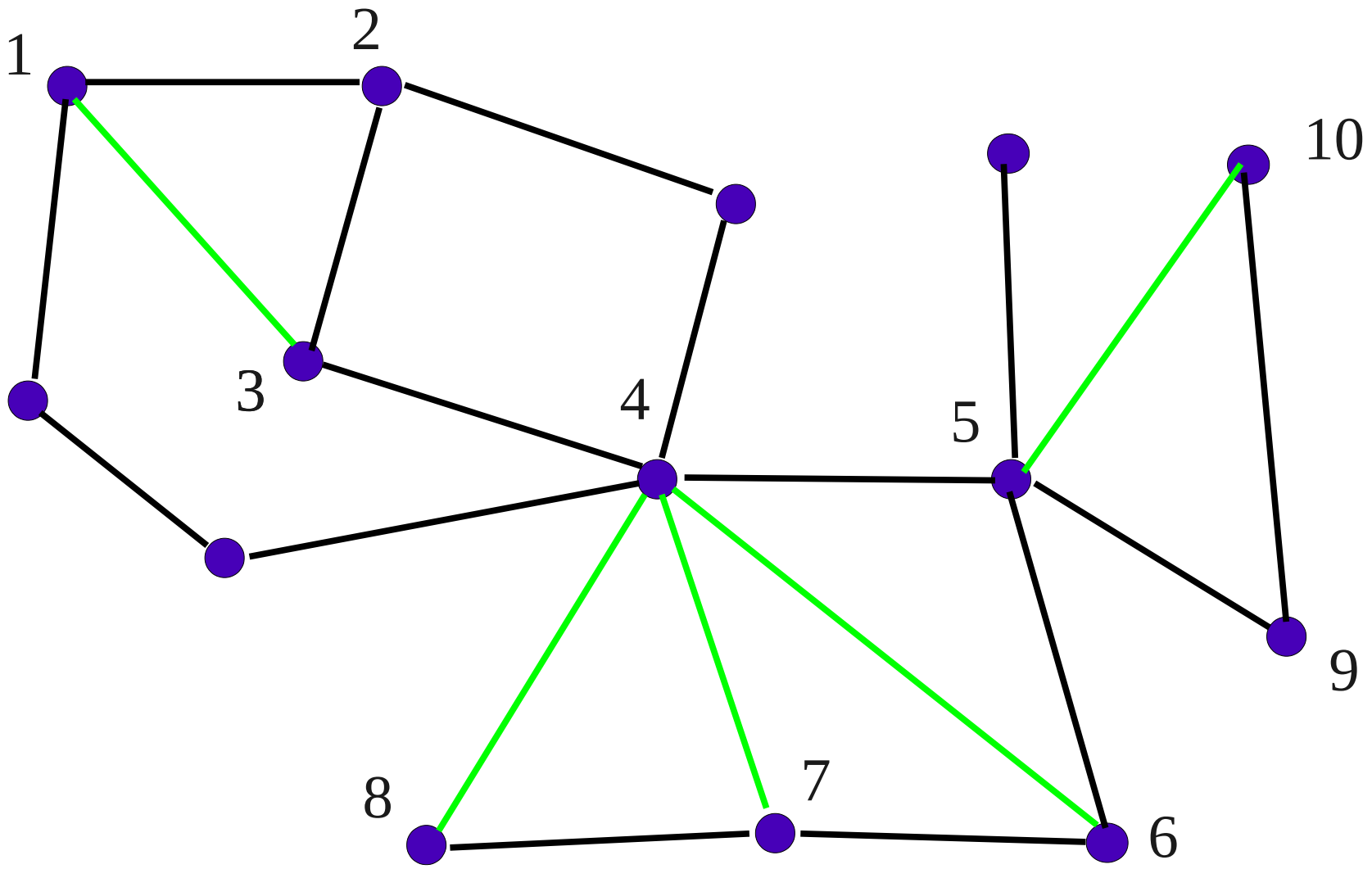
Biconnected Components



Biconnected Components

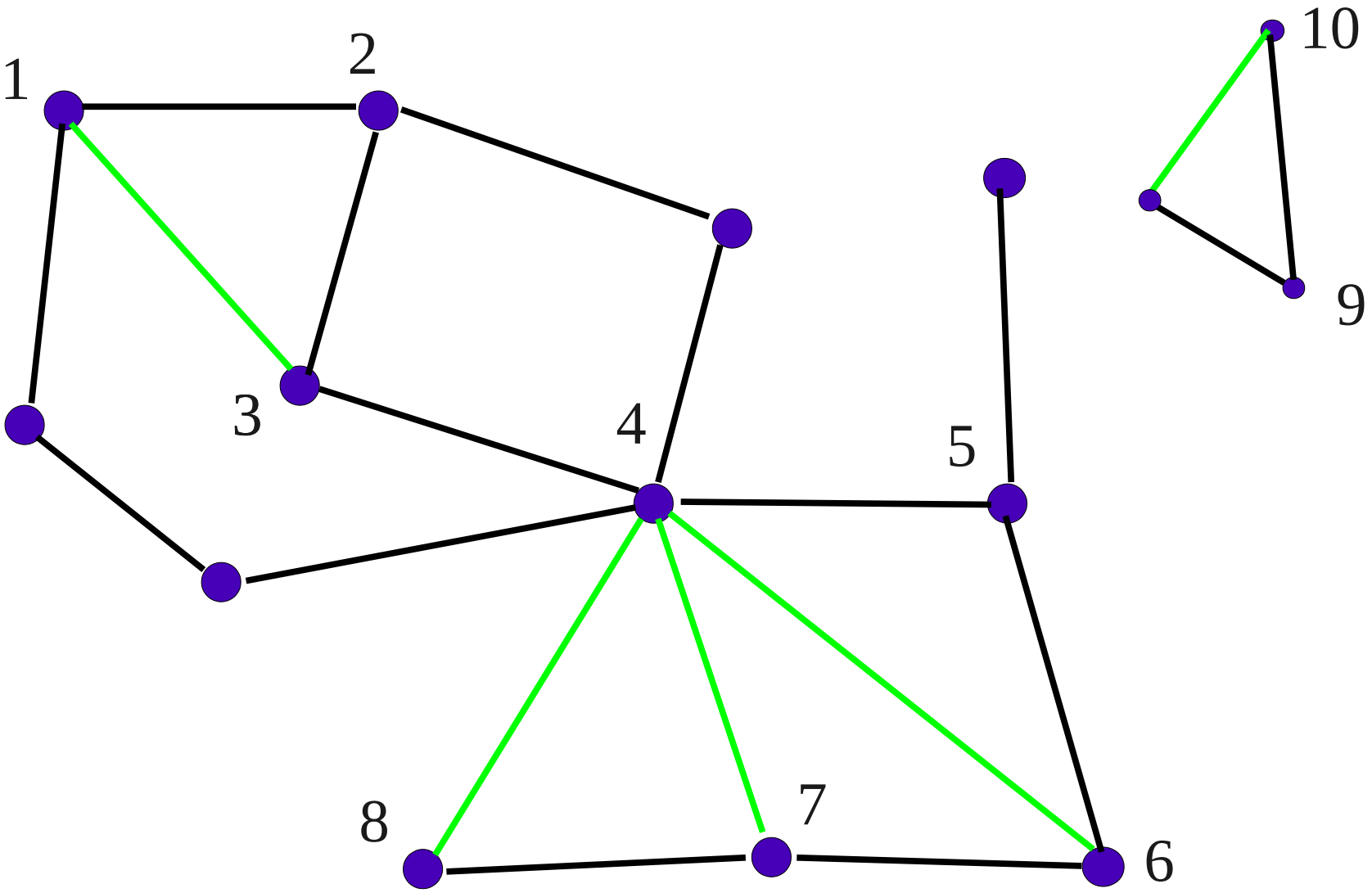


Biconnected Components

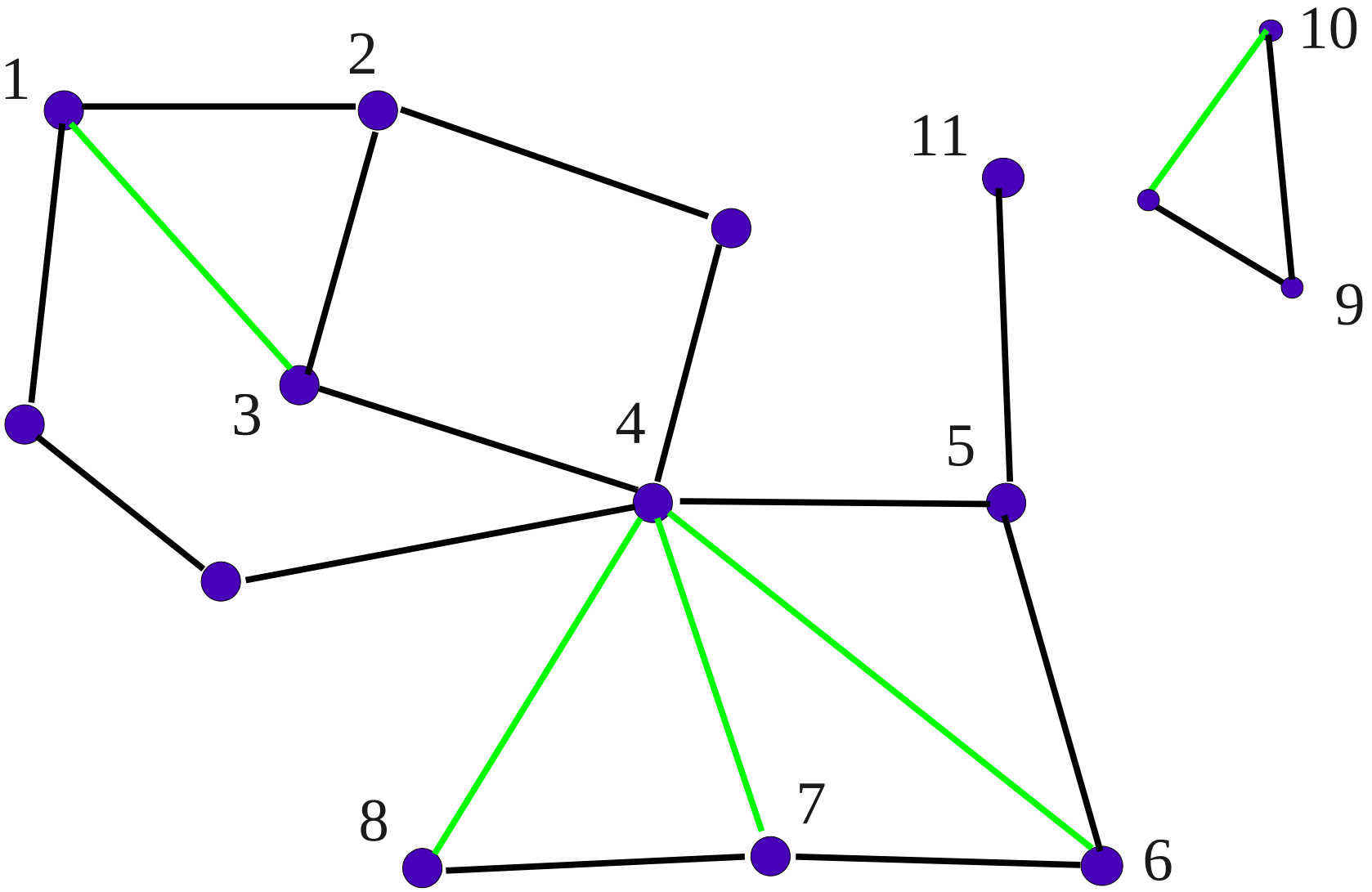


We arrive back at vertex 5 with no vertex having seen a neighboring
vertex numbered less than 5. So this must be a biconnected comp.

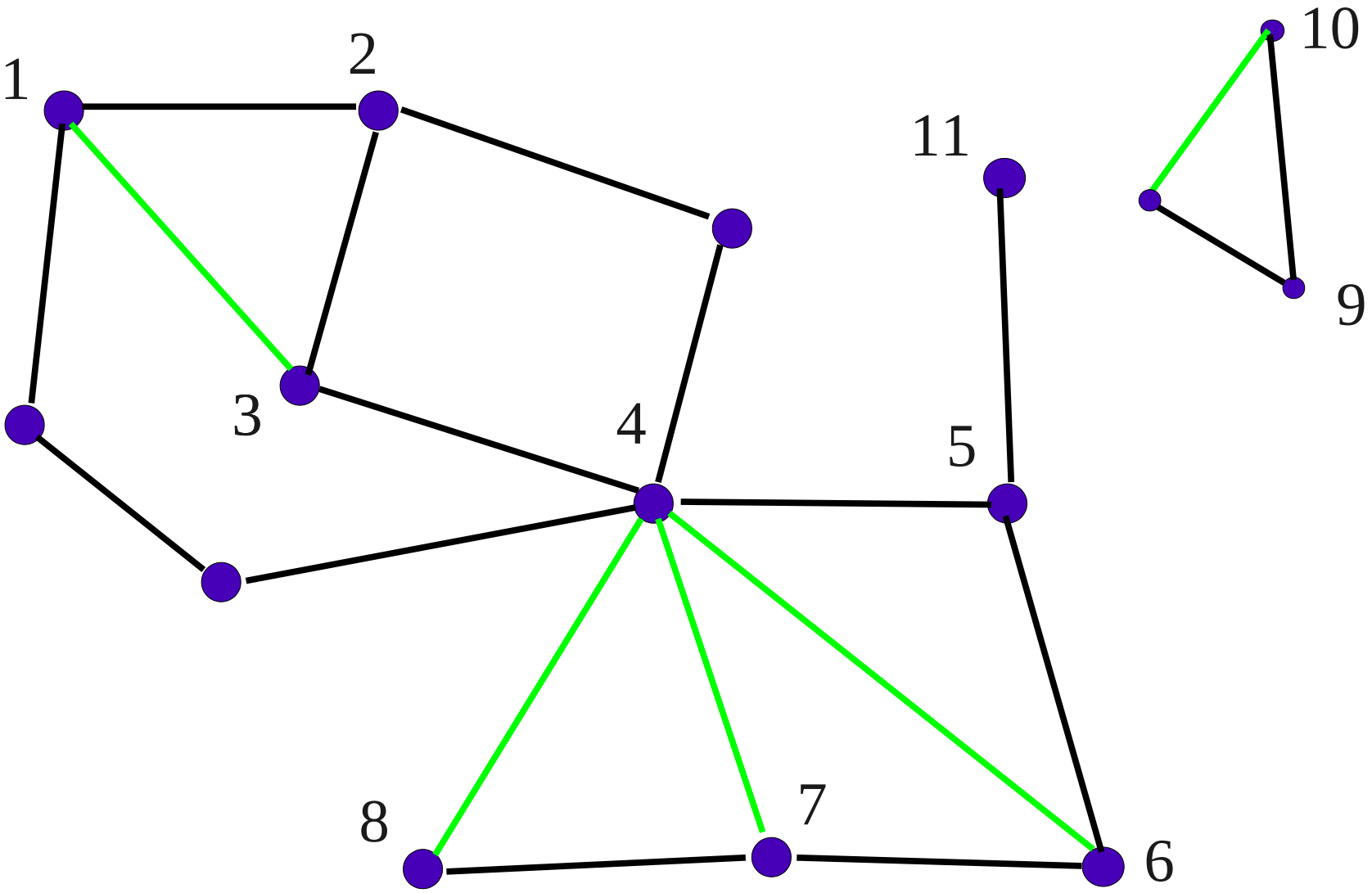
Biconnected Components



Biconnected Components

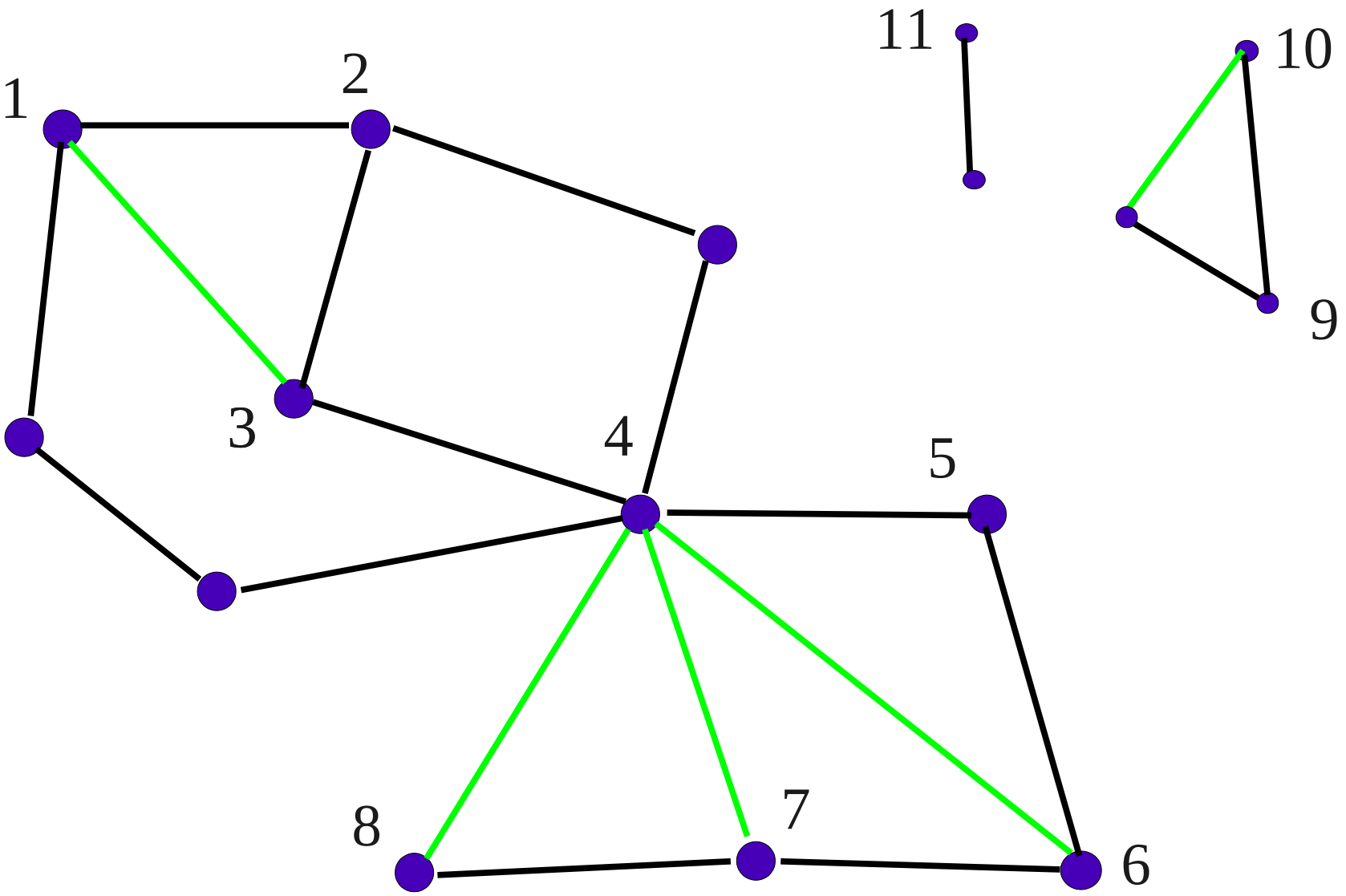


Biconnected Components

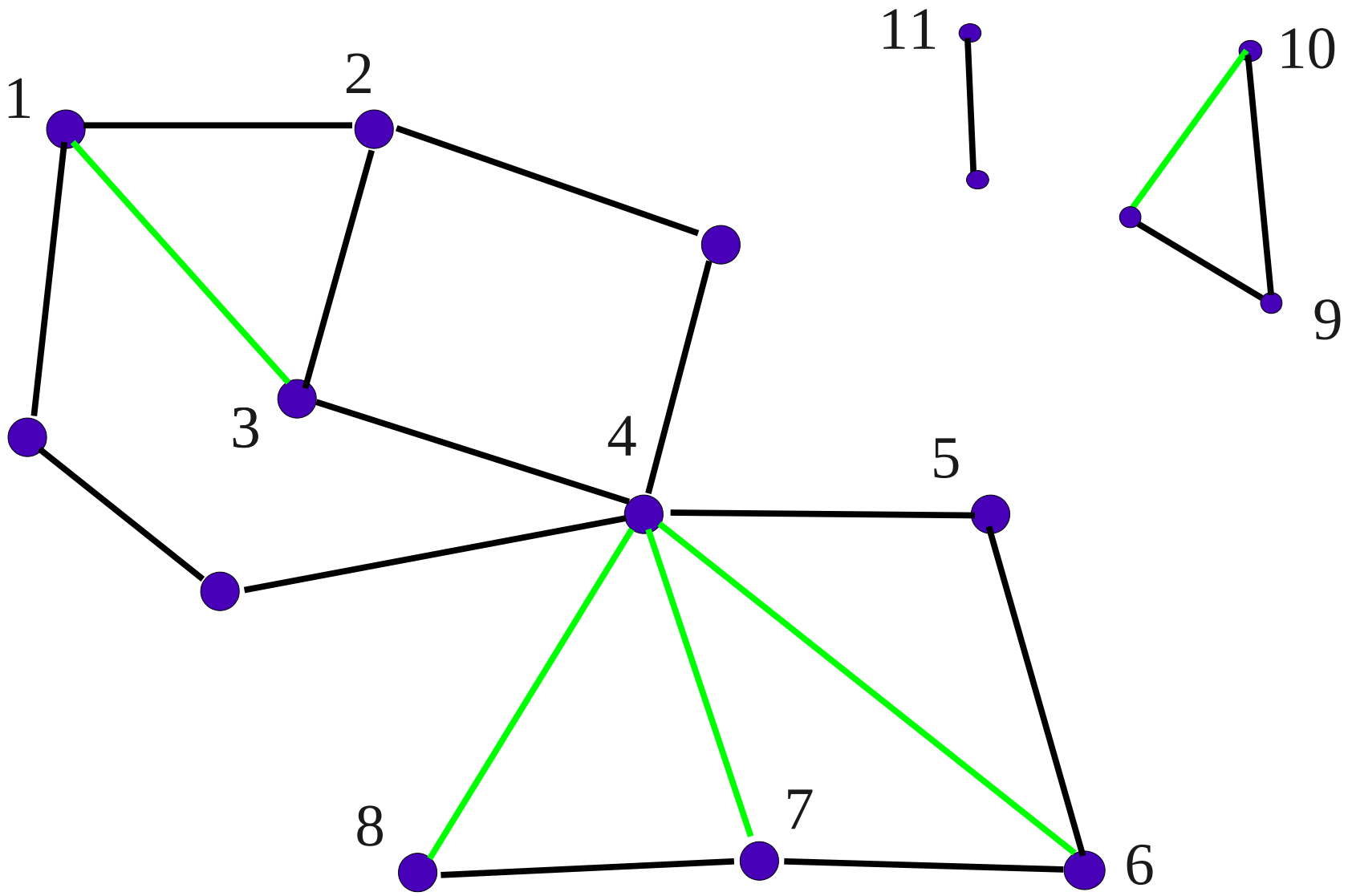


We return to vertex 5 with no vertex having seen a neighboring vertex numbered less than 5. So this must be a biconnected comp.

Biconnected Components

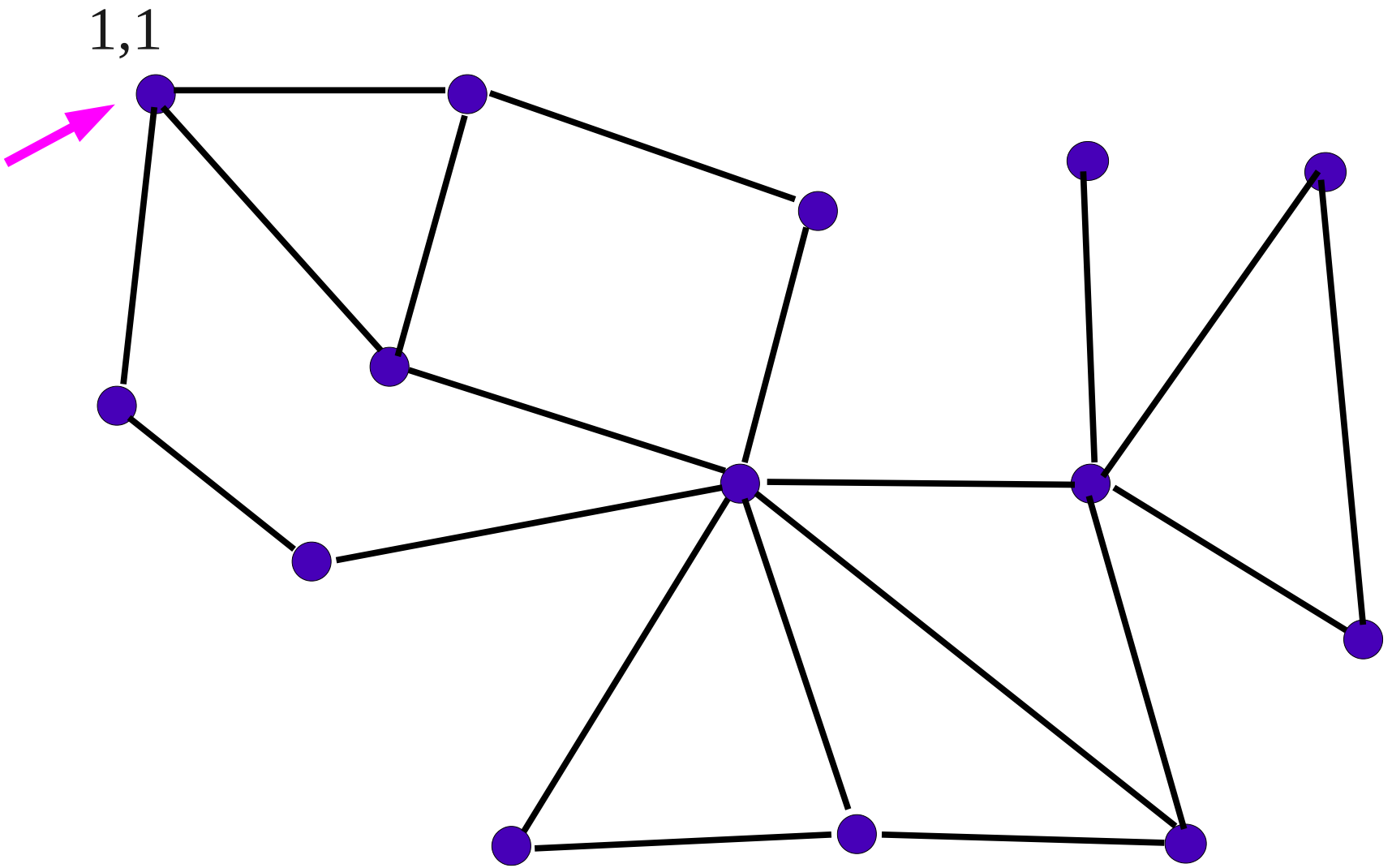


Biconnected Components

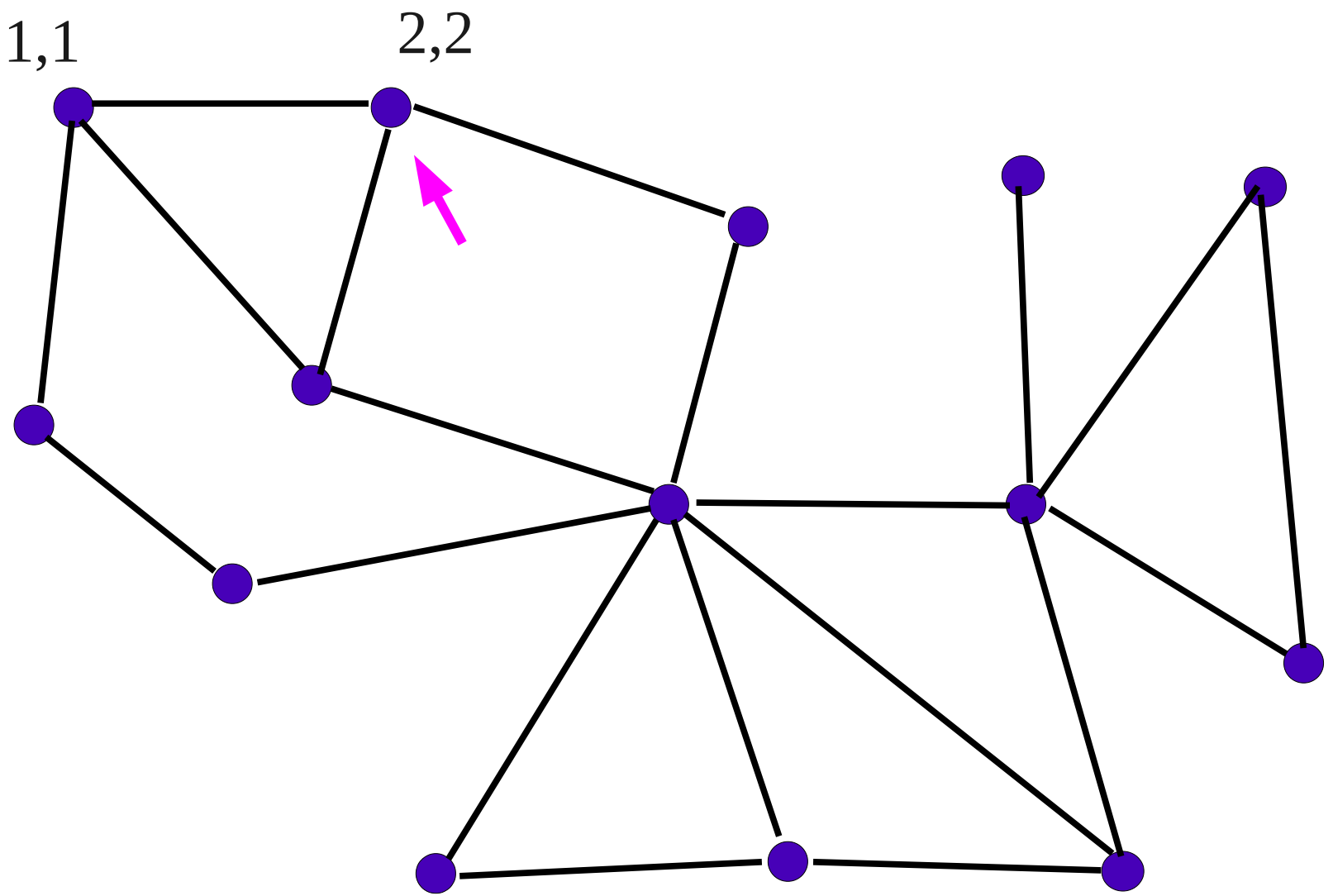


This suggests that each vertex carry a second number which is the lowest numbered vertex seen so far from a path out of it.

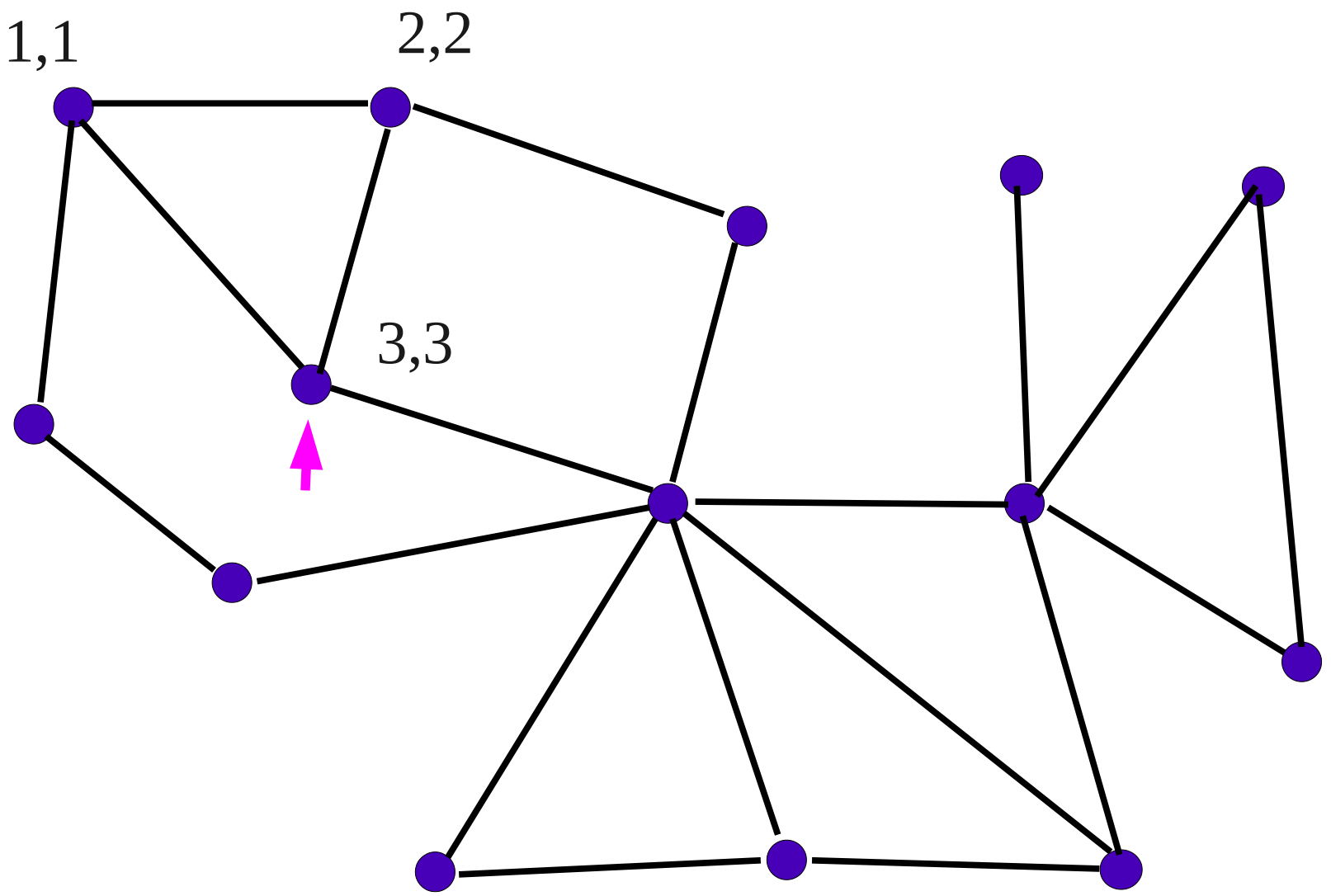
Biconnected Components



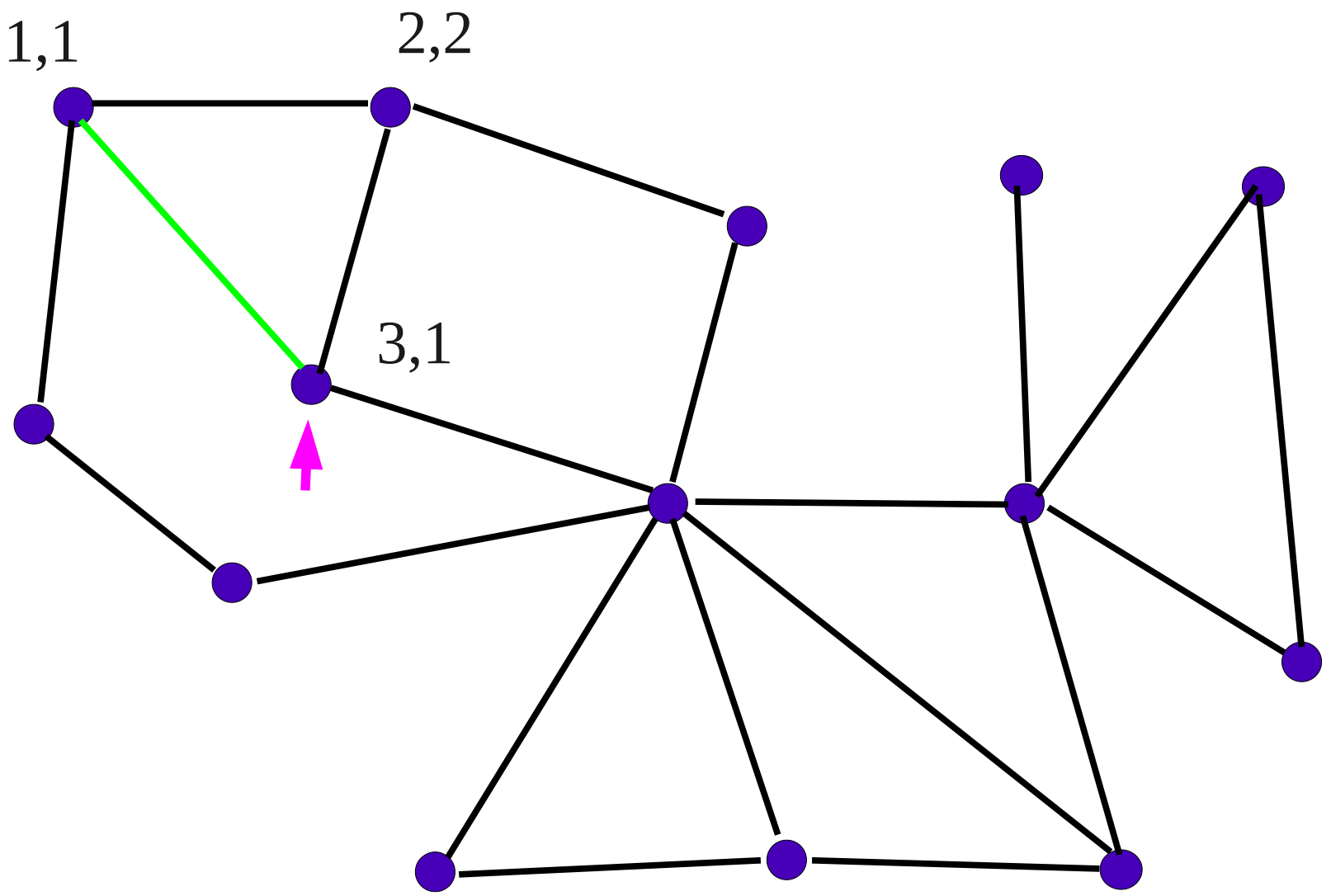
Biconnected Components



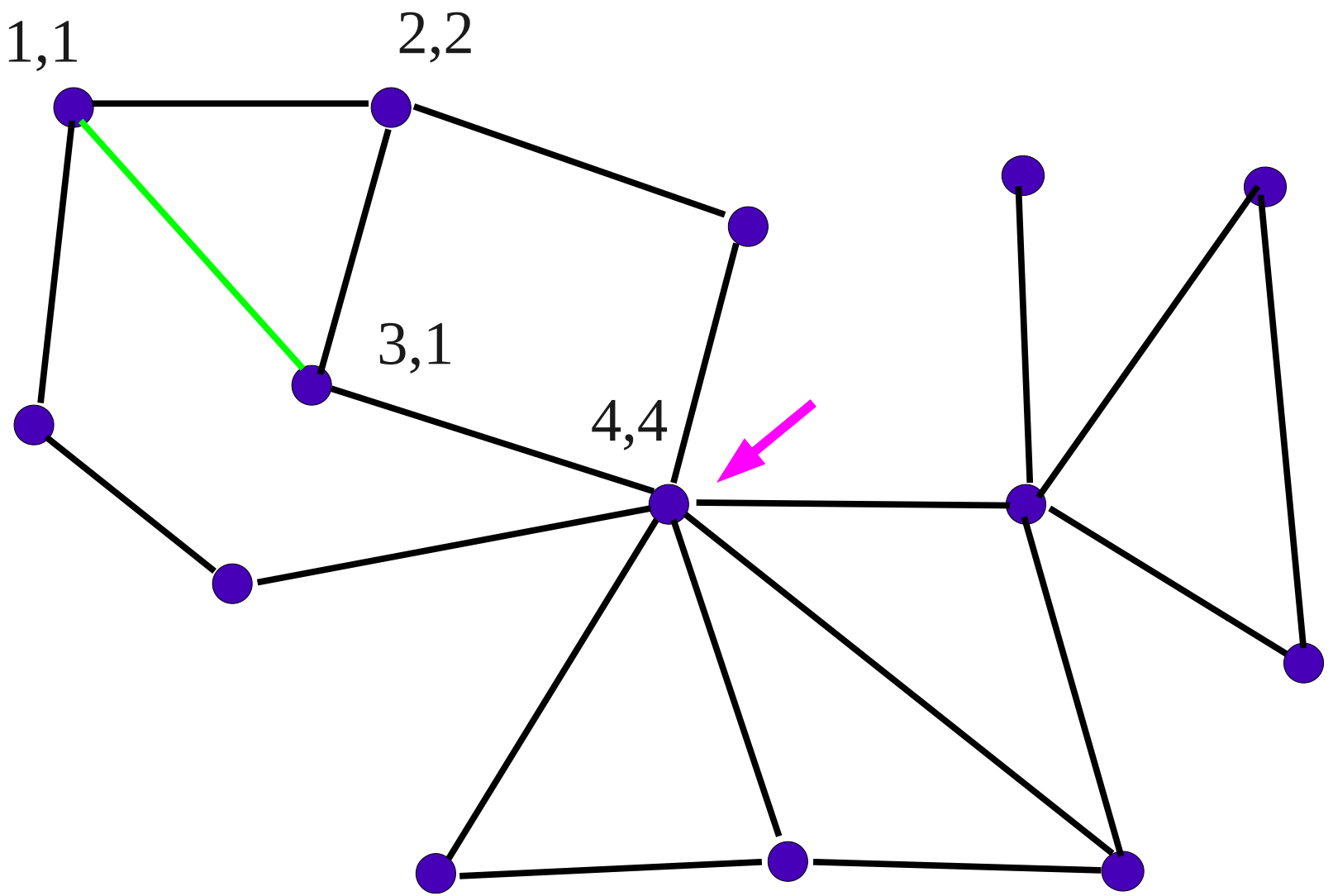
Biconnected Components



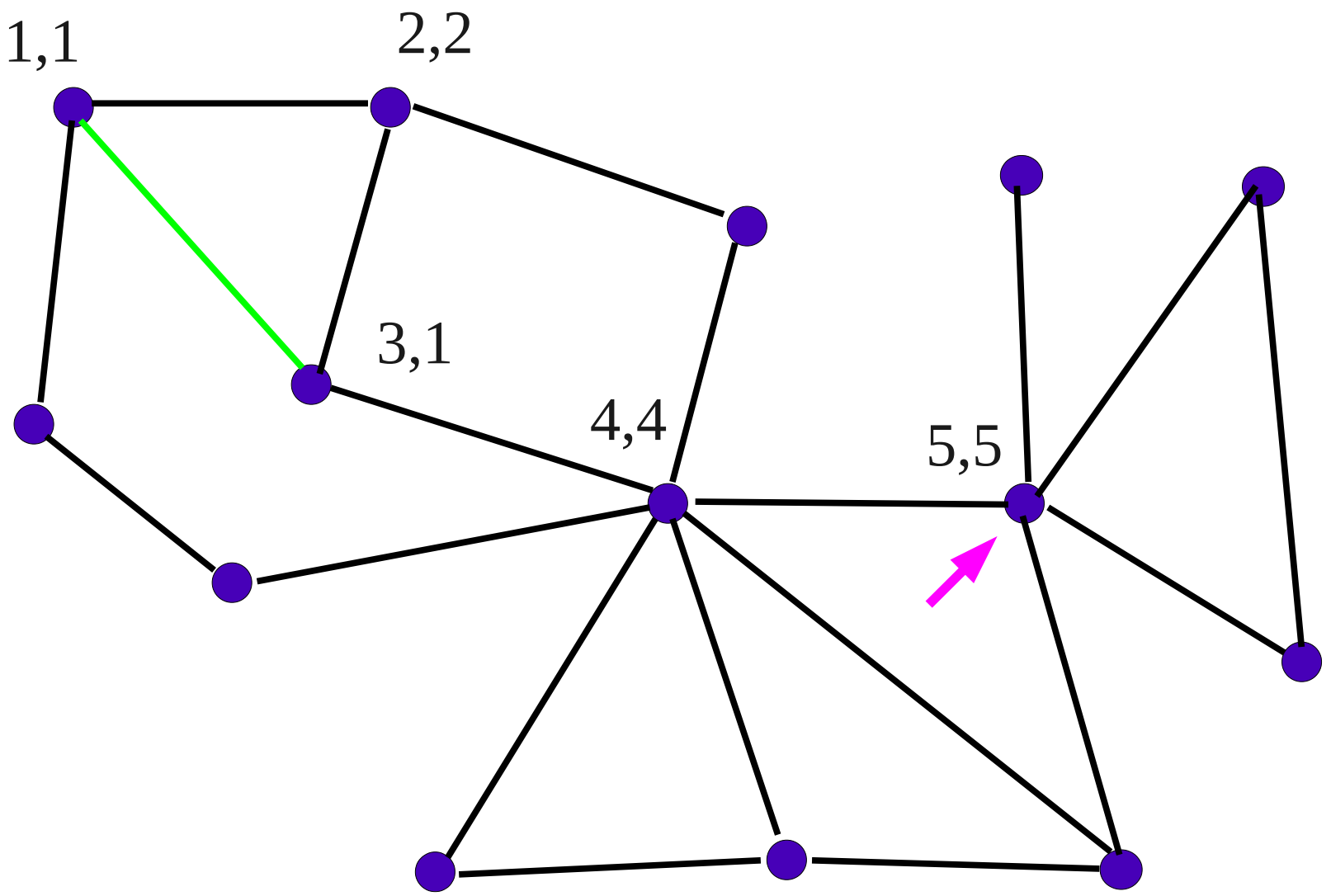
Biconnected Components



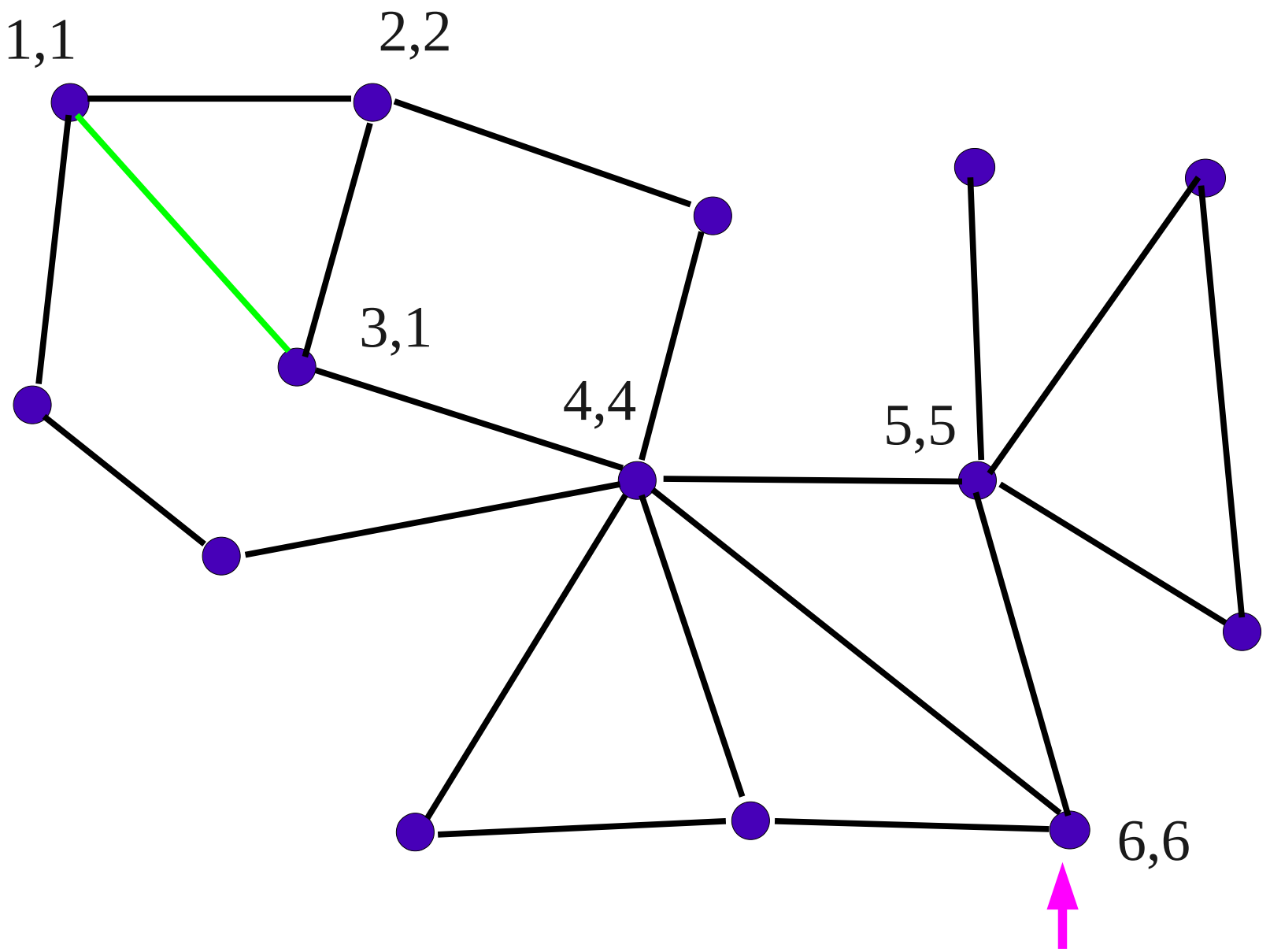
Biconnected Components



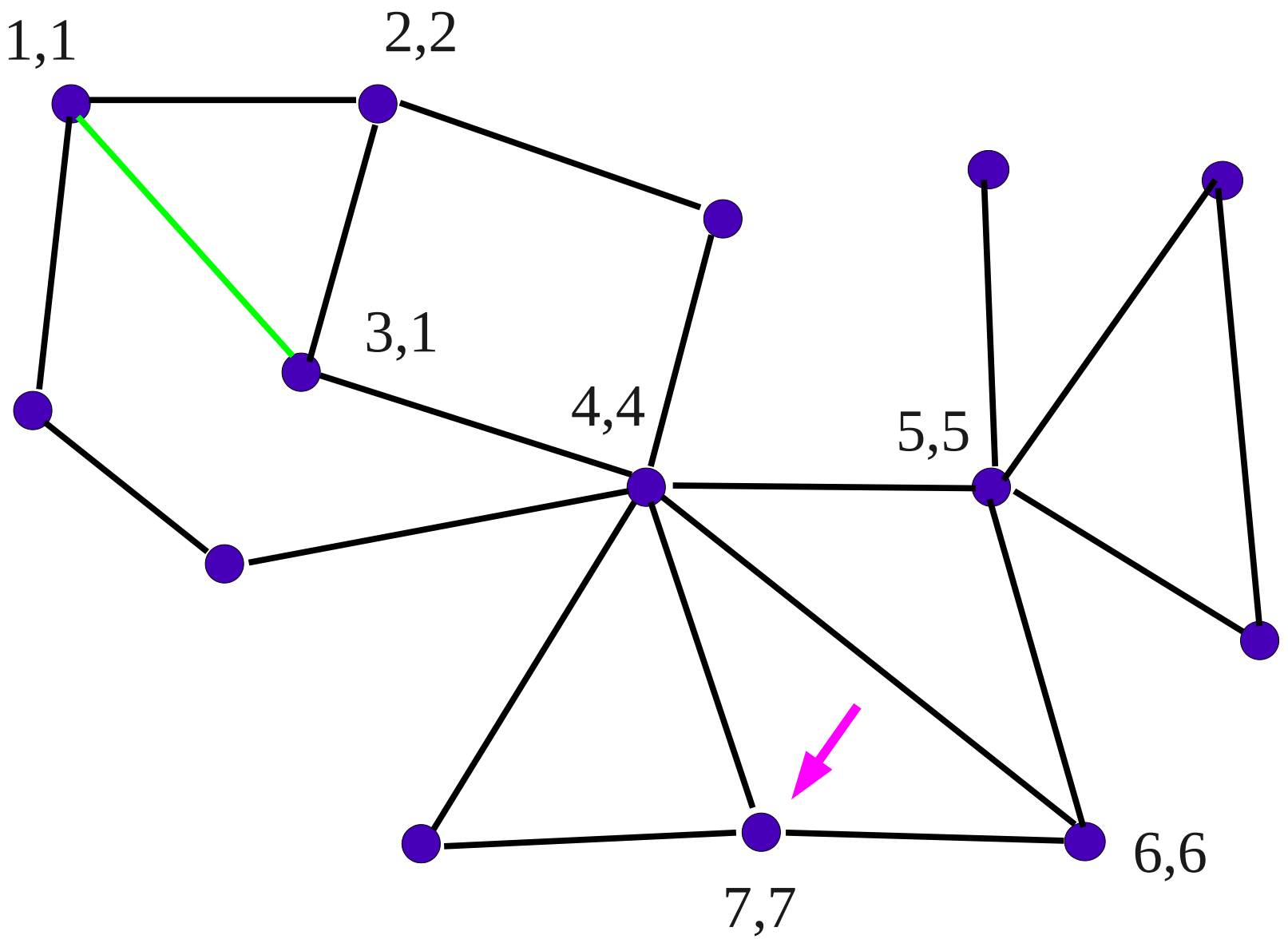
Biconnected Components



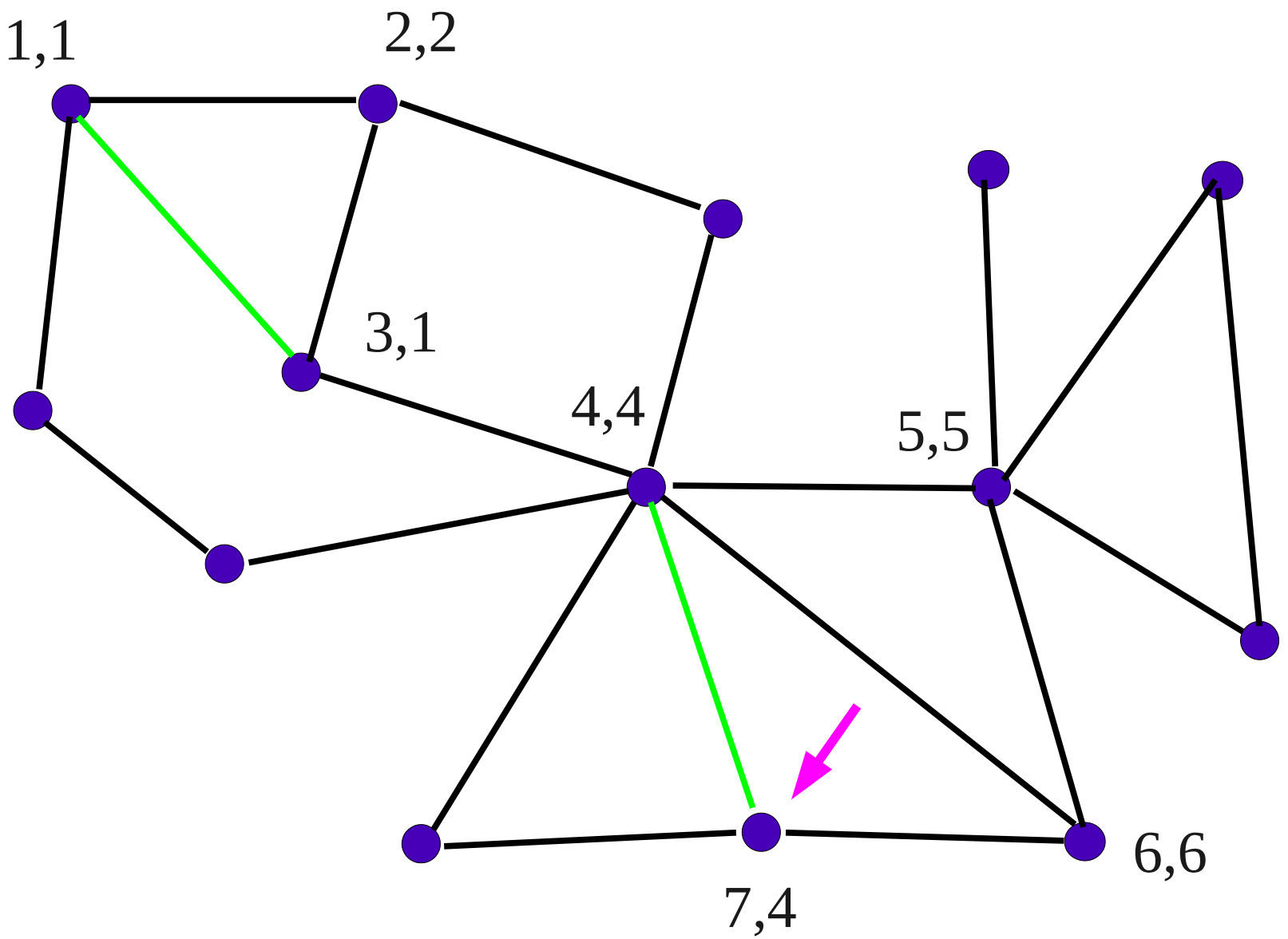
Biconnected Components



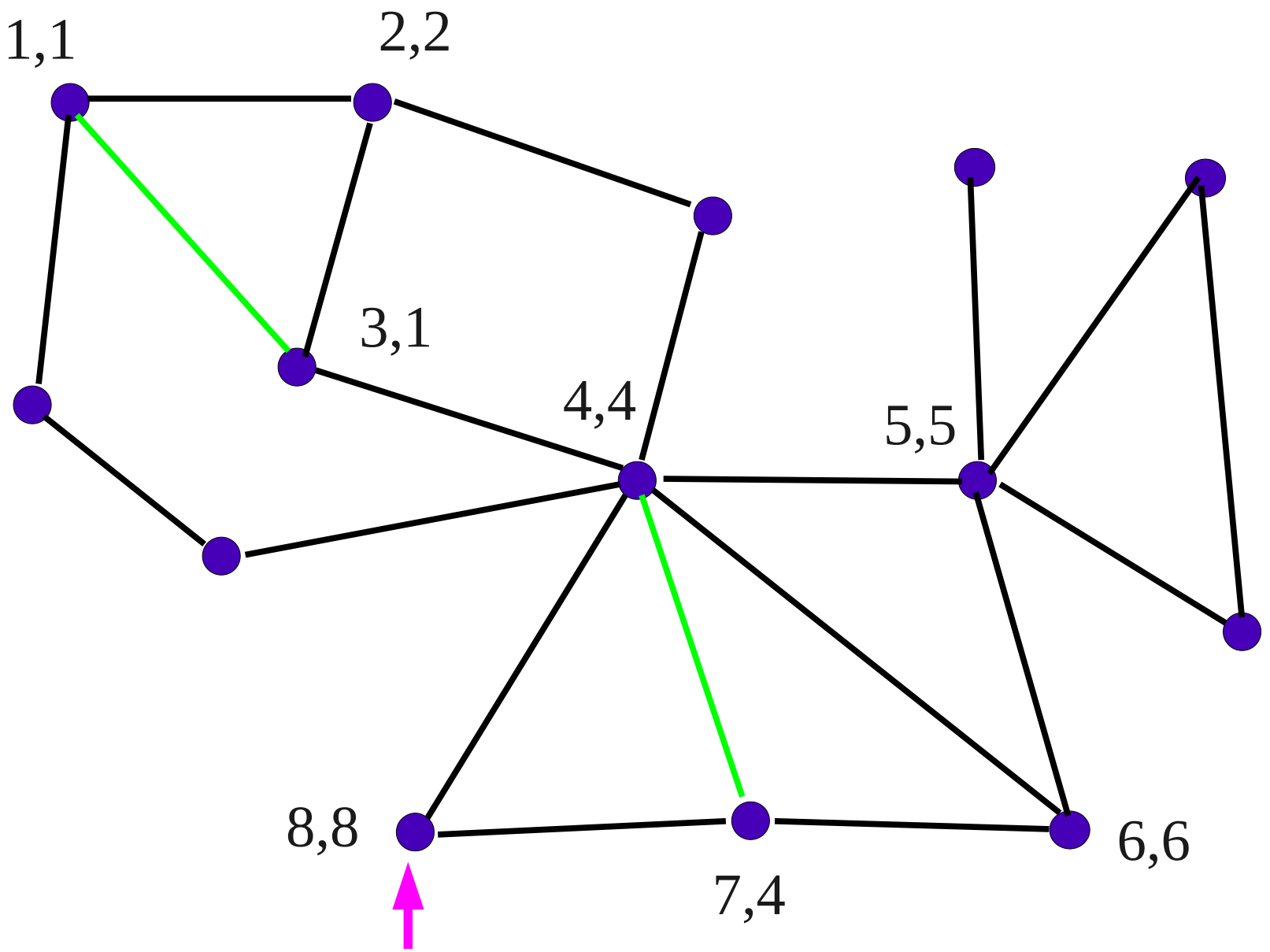
Biconnected Components



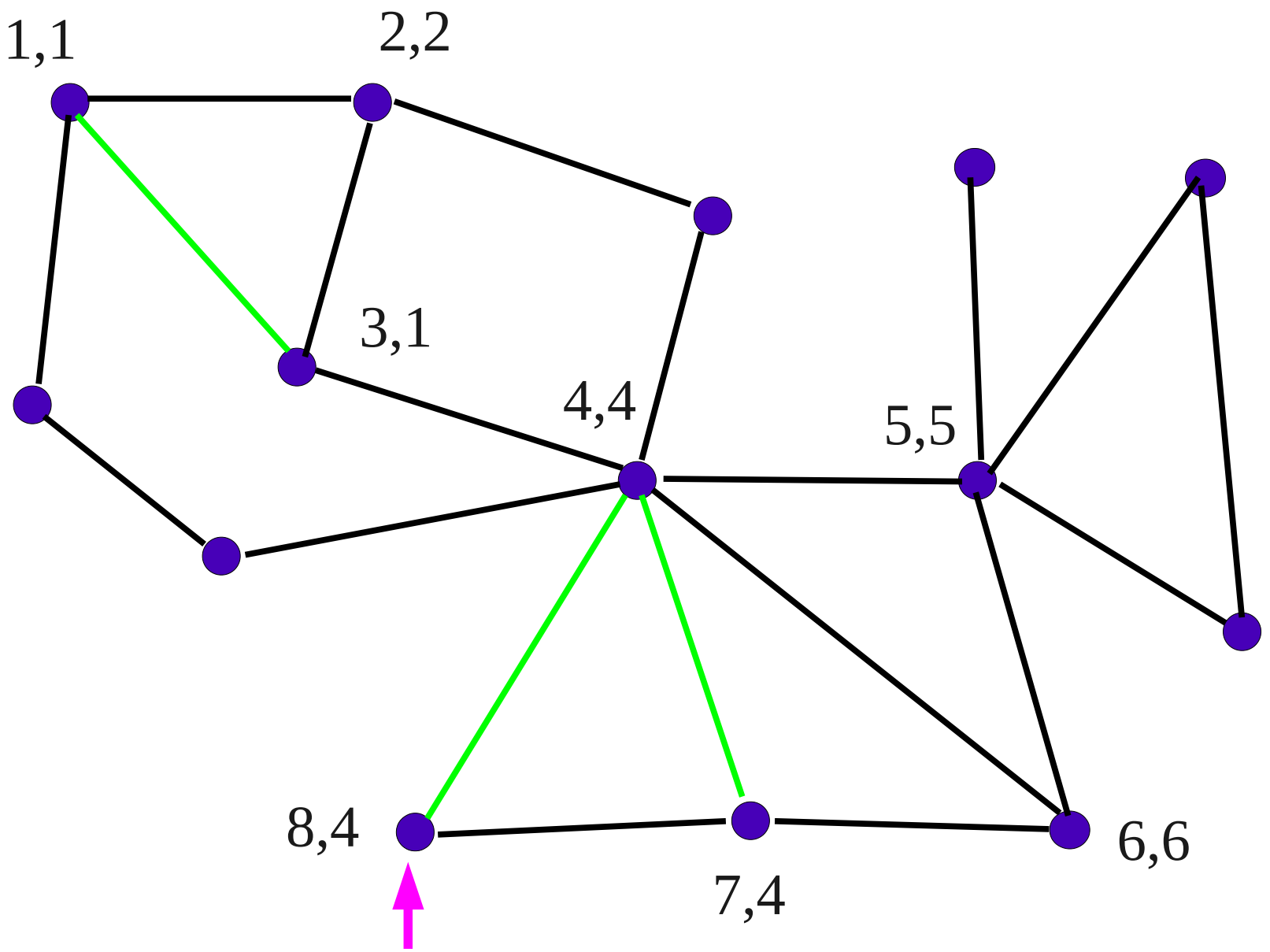
Biconnected Components



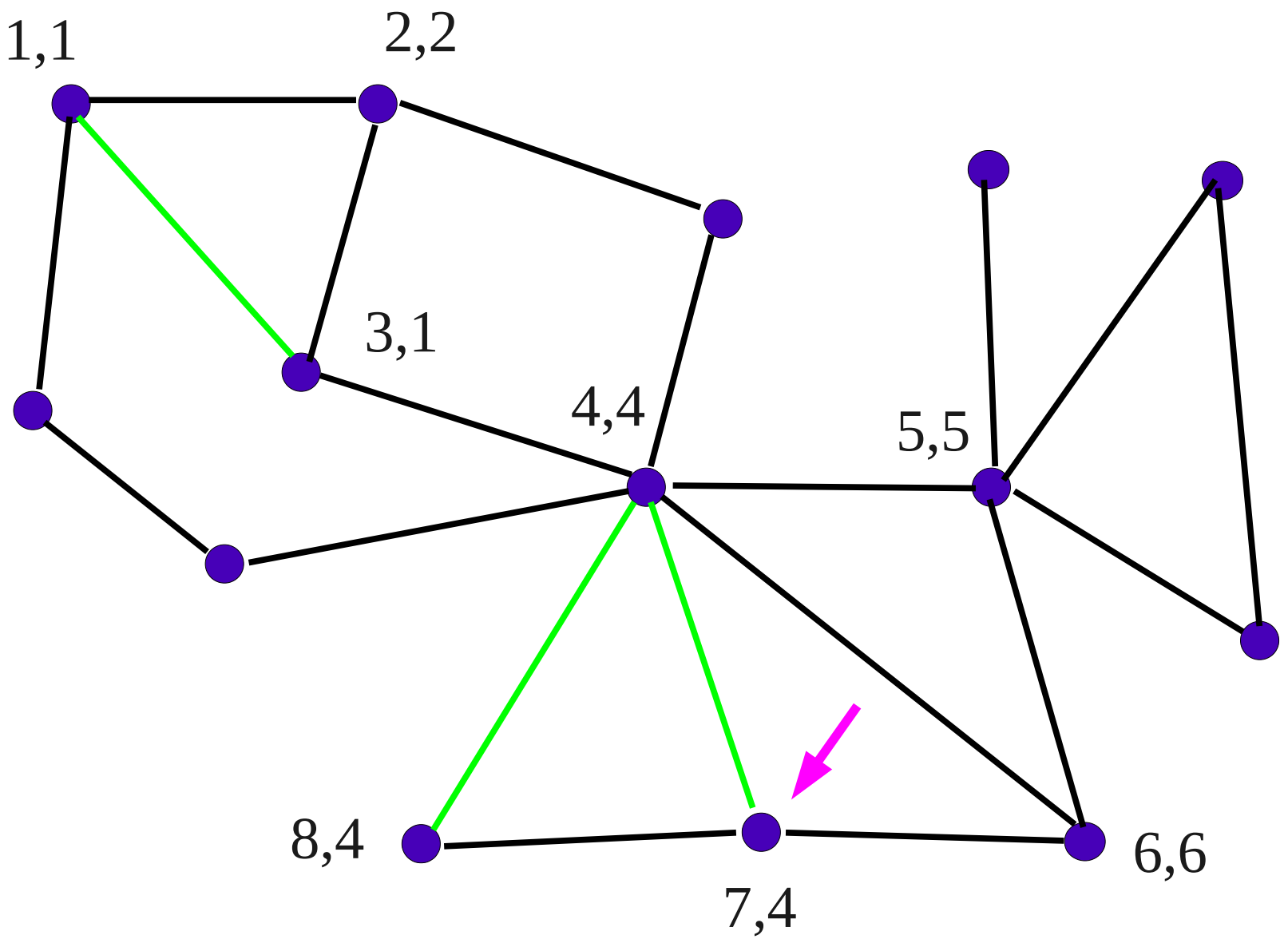
Biconnected Components



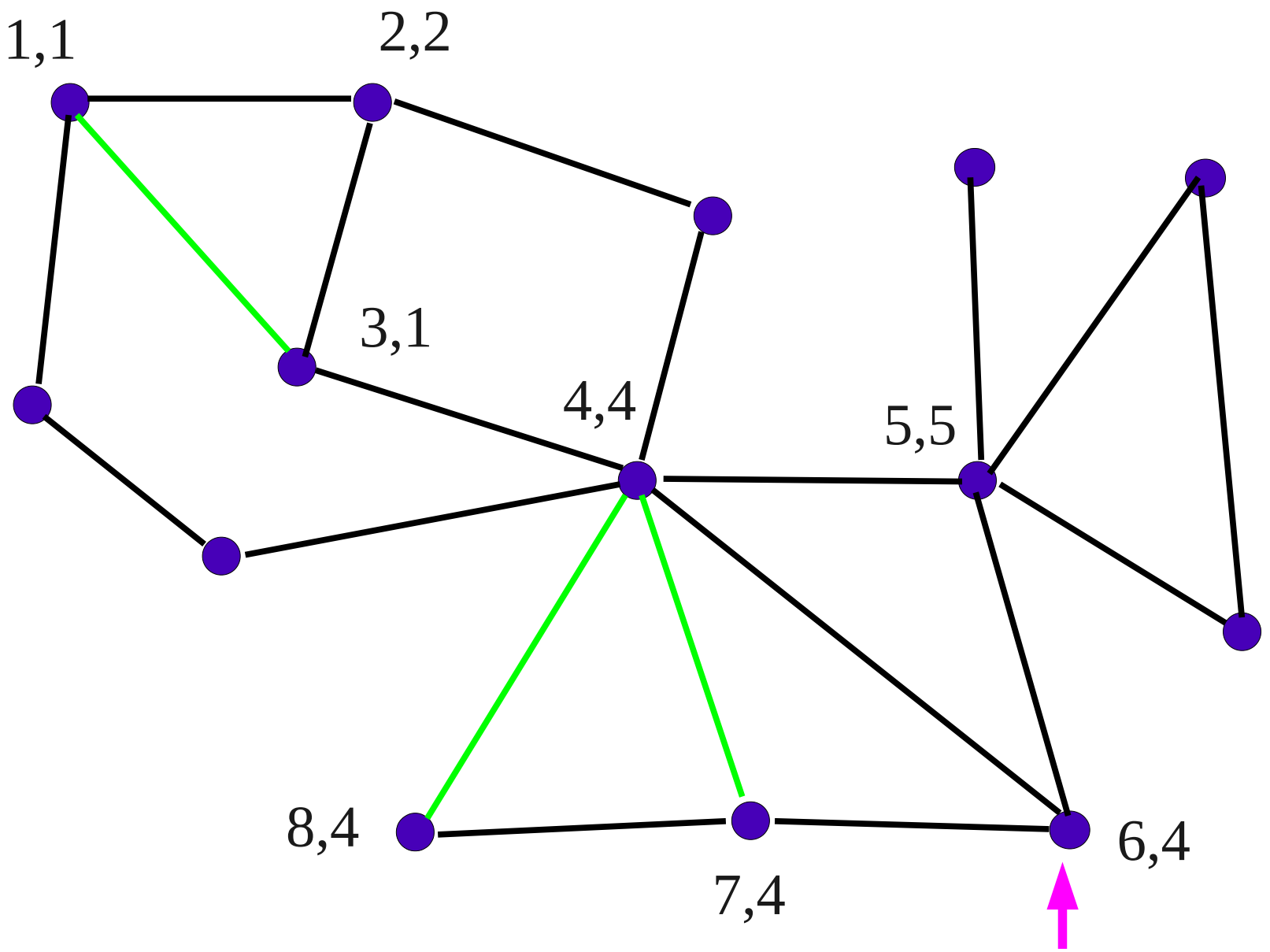
Biconnected Components



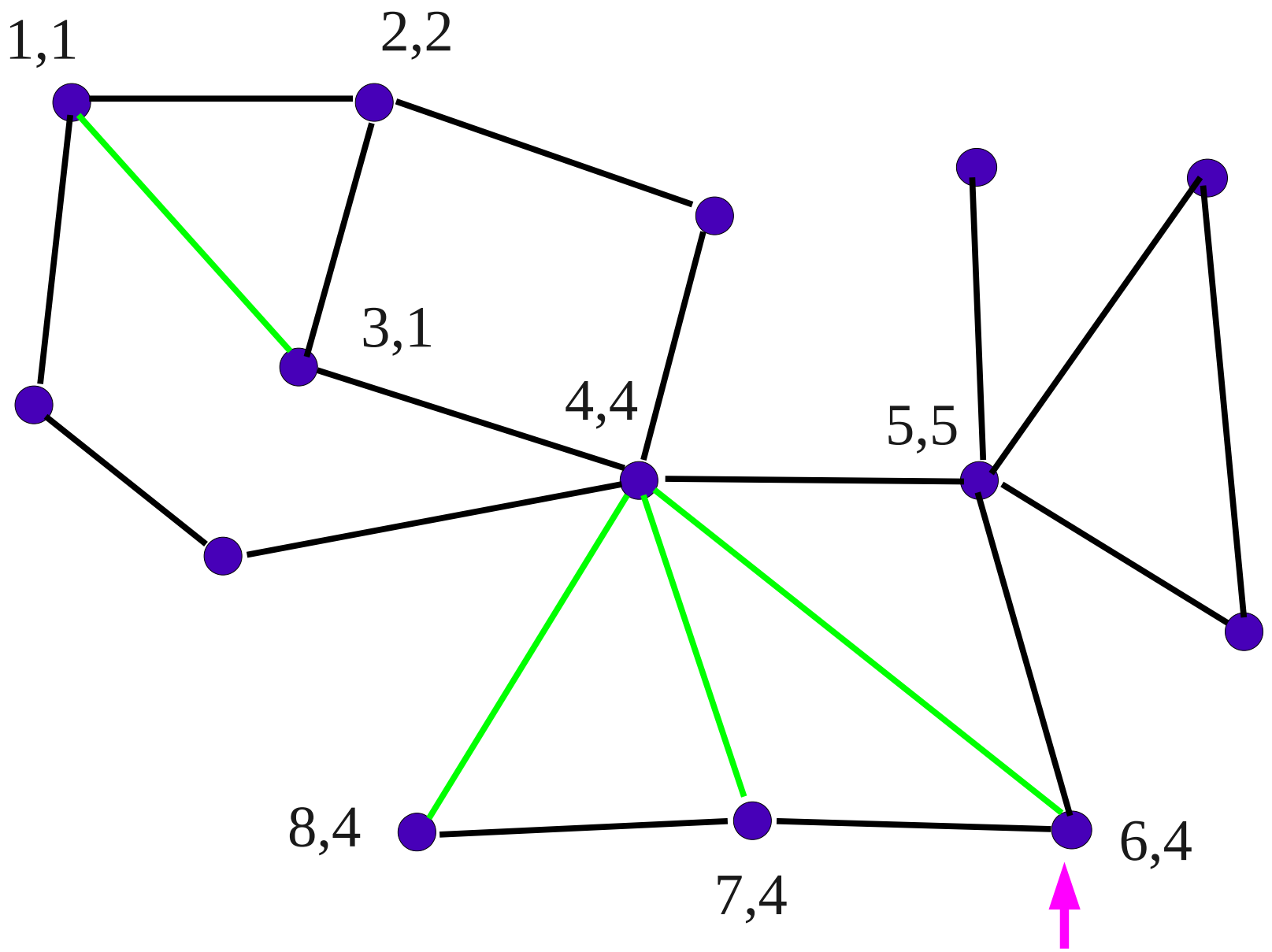
Biconnected Components



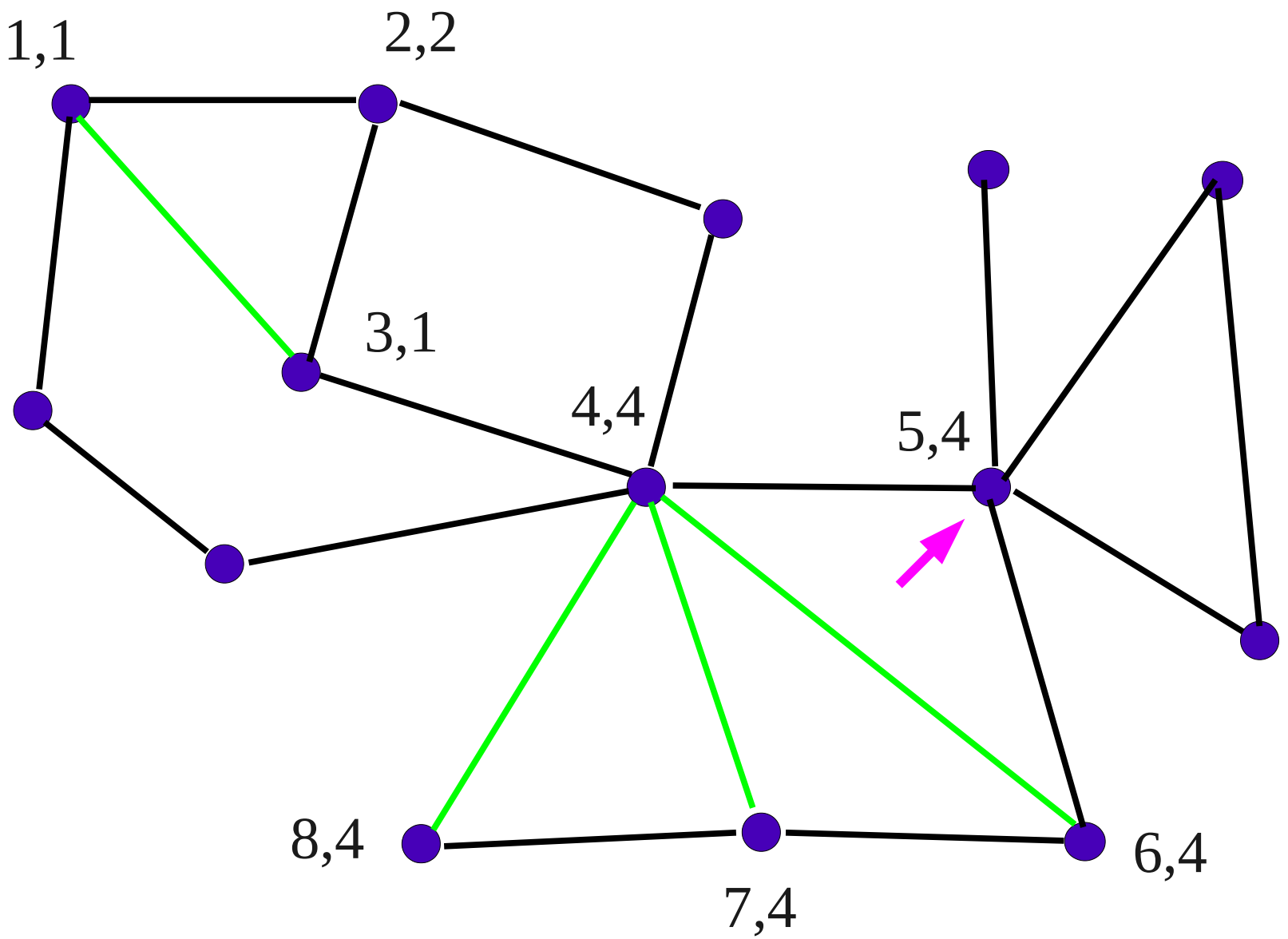
Biconnected Components



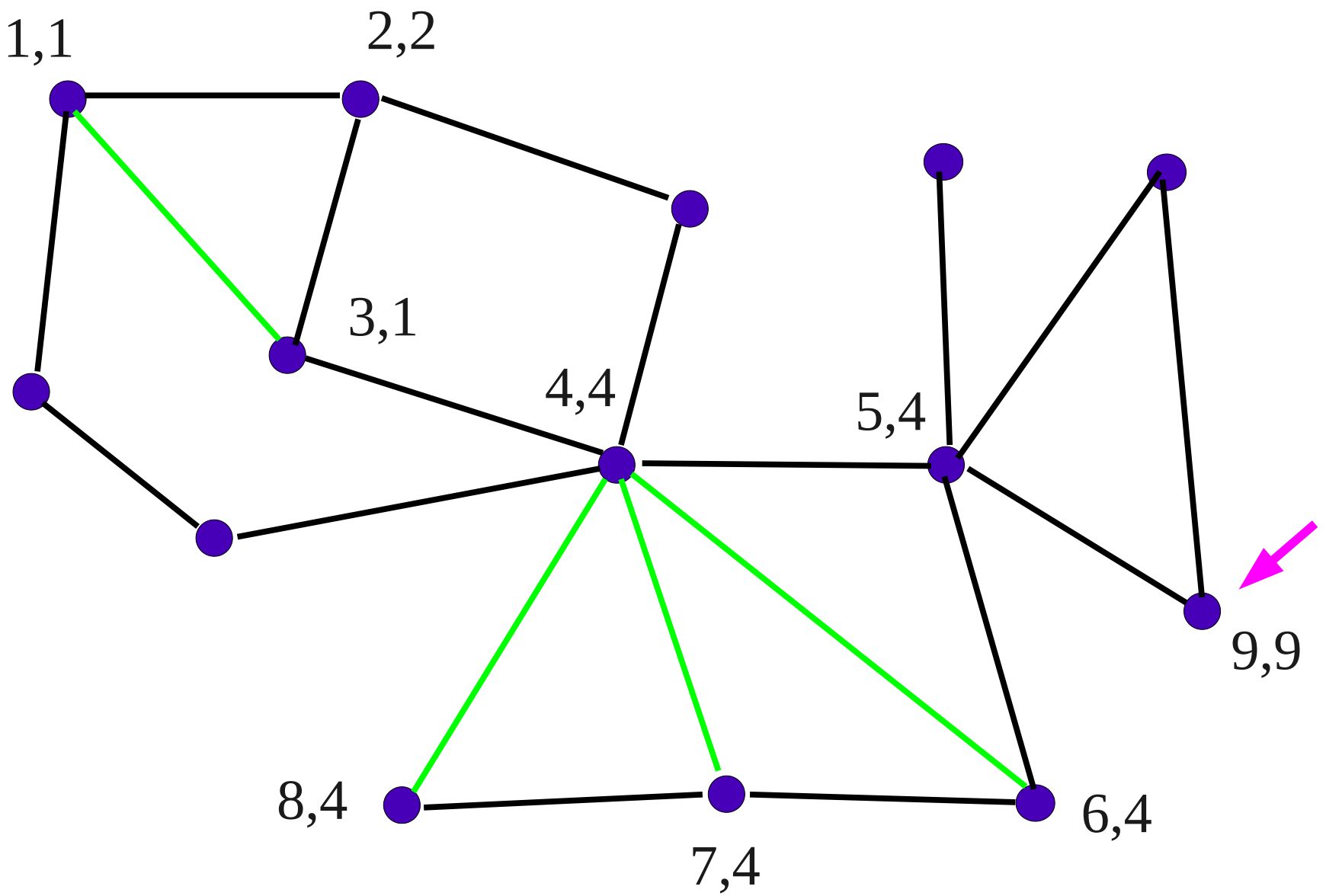
Biconnected Components



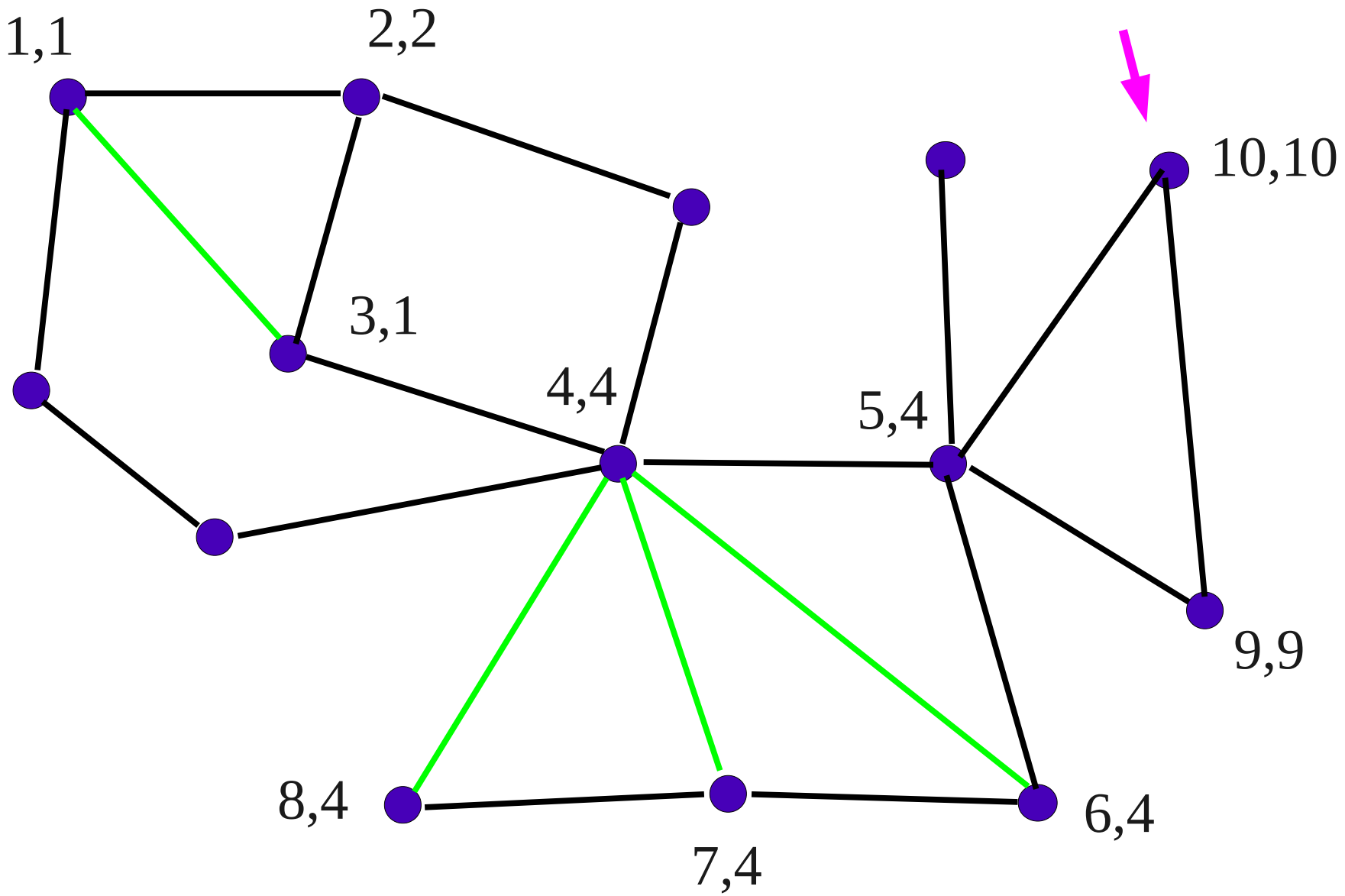
Biconnected Components



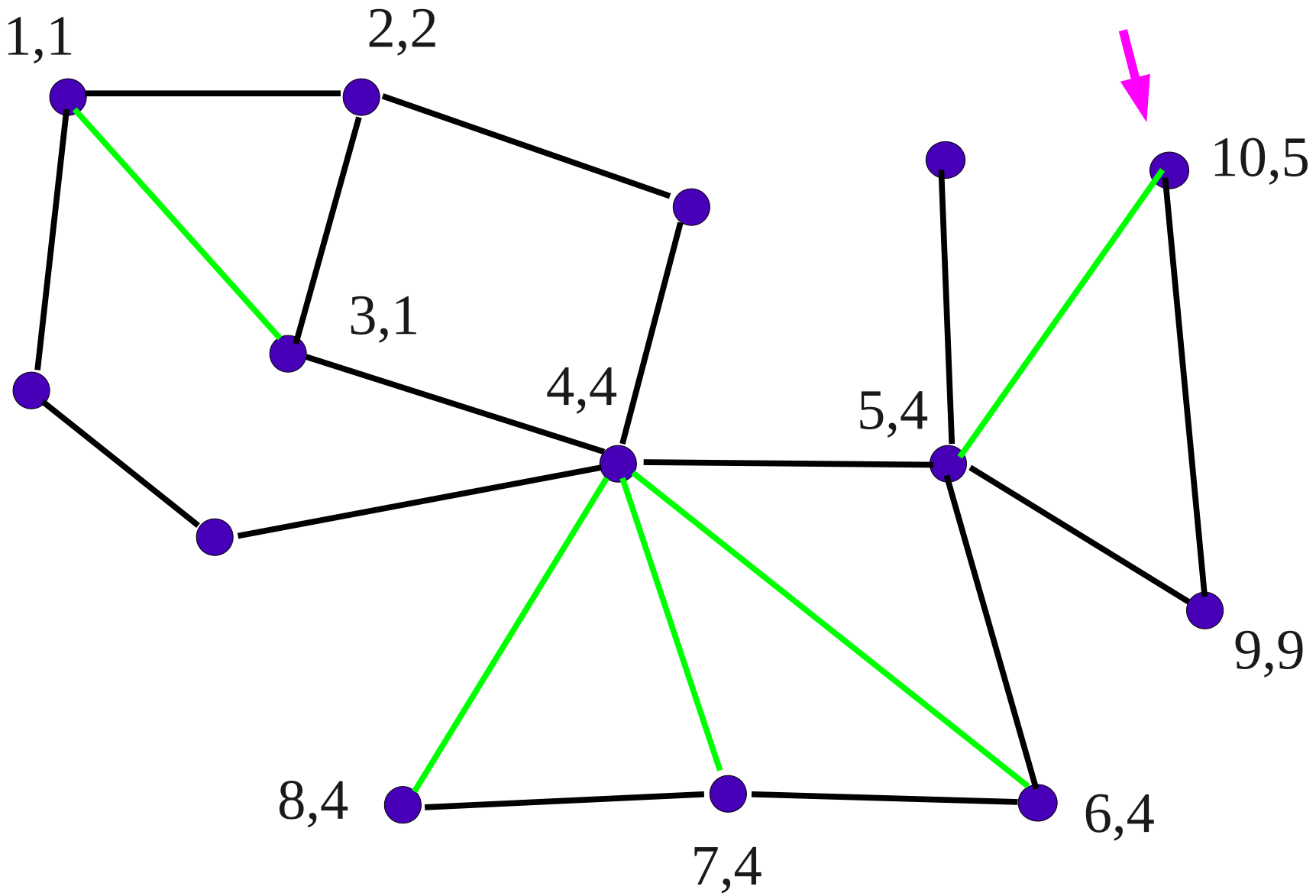
Biconnected Components



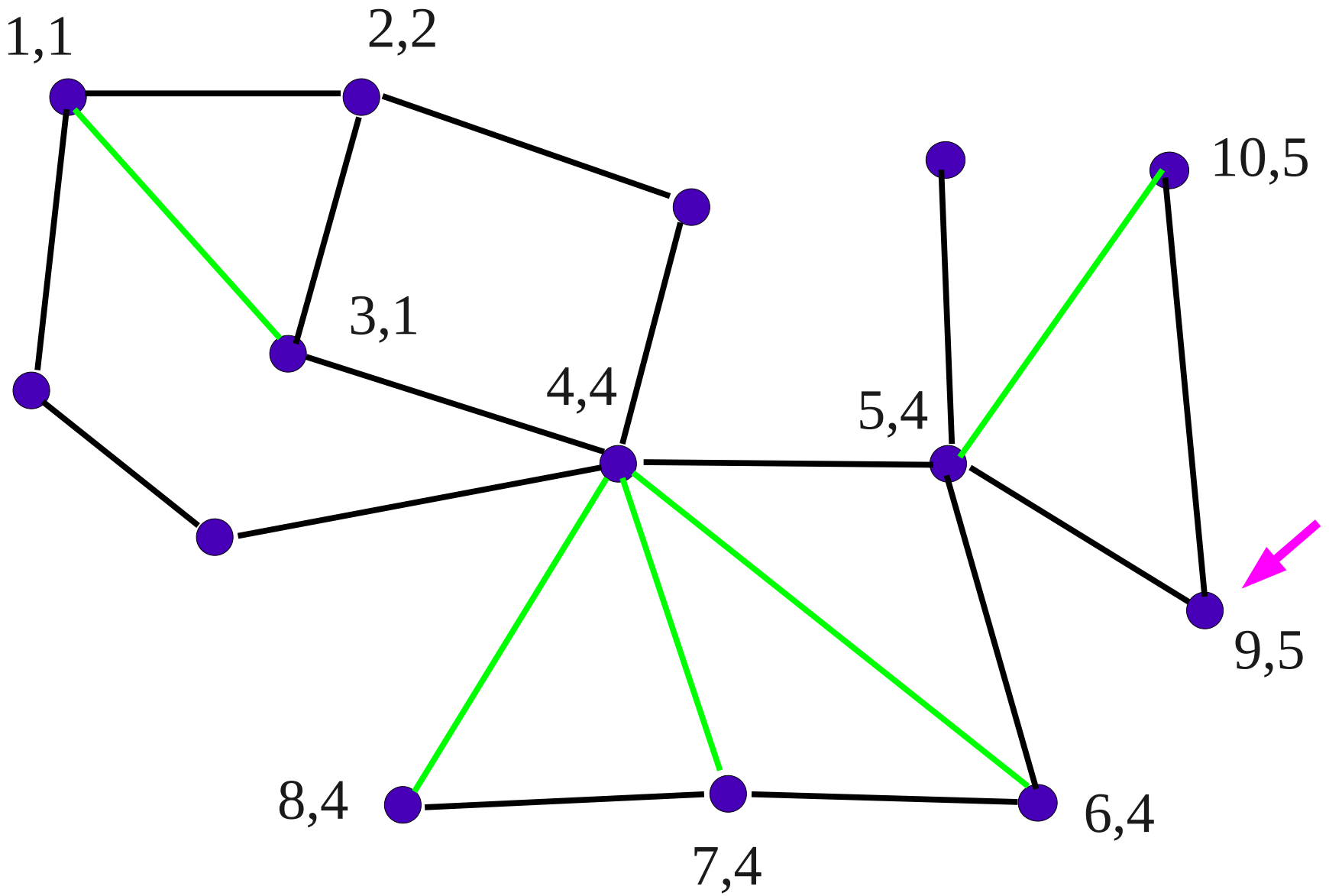
Biconnected Components



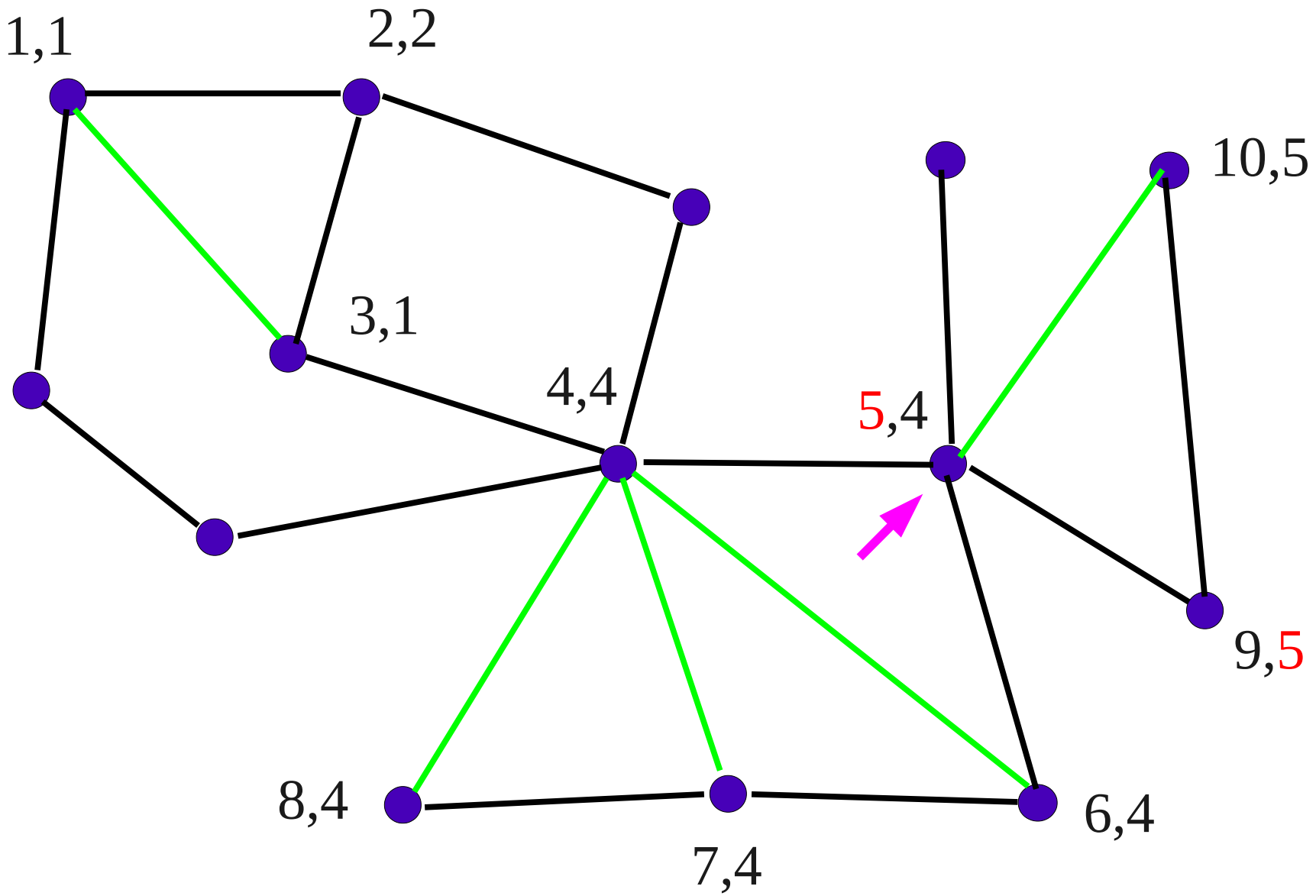
Biconnected Components



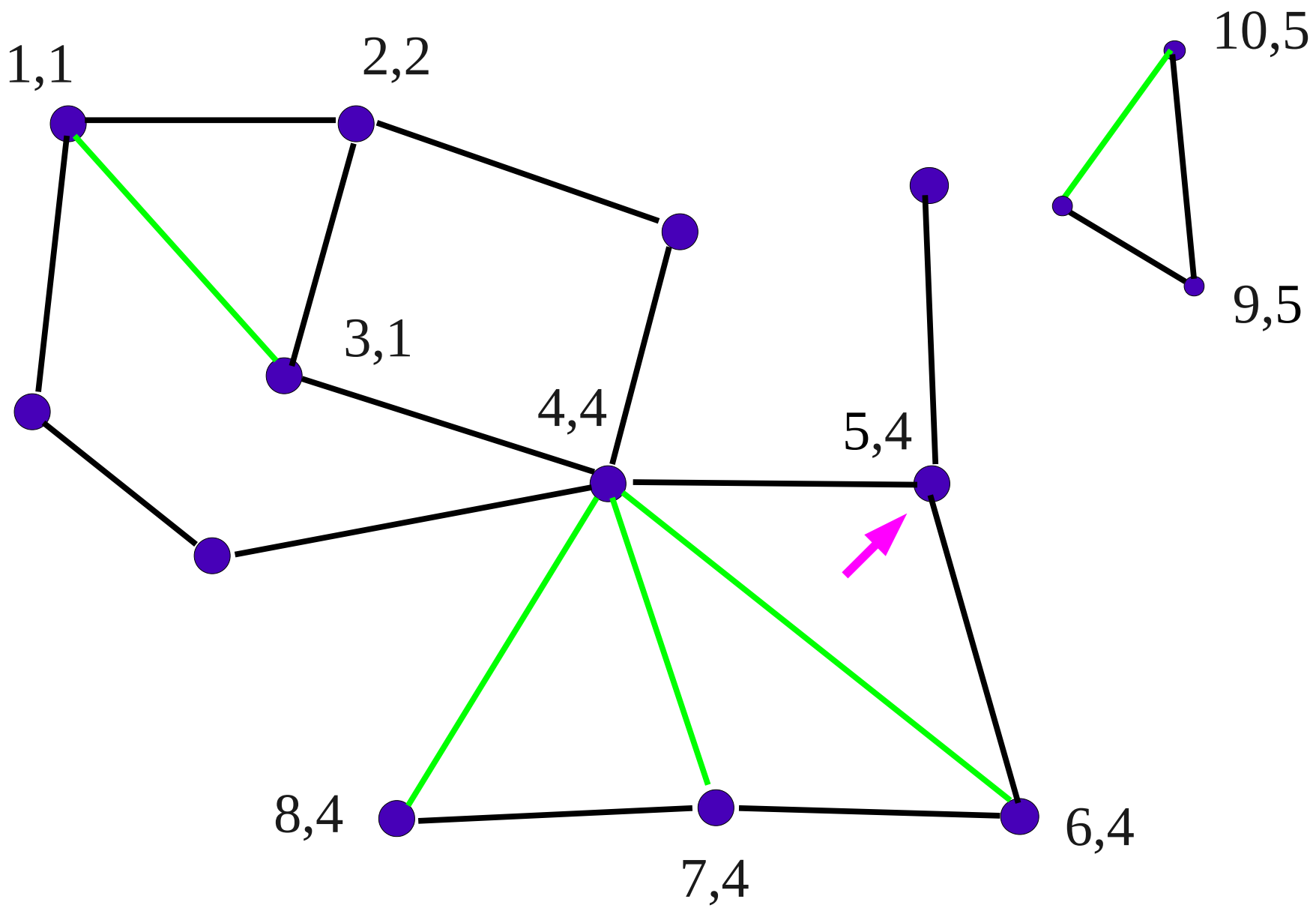
Biconnected Components



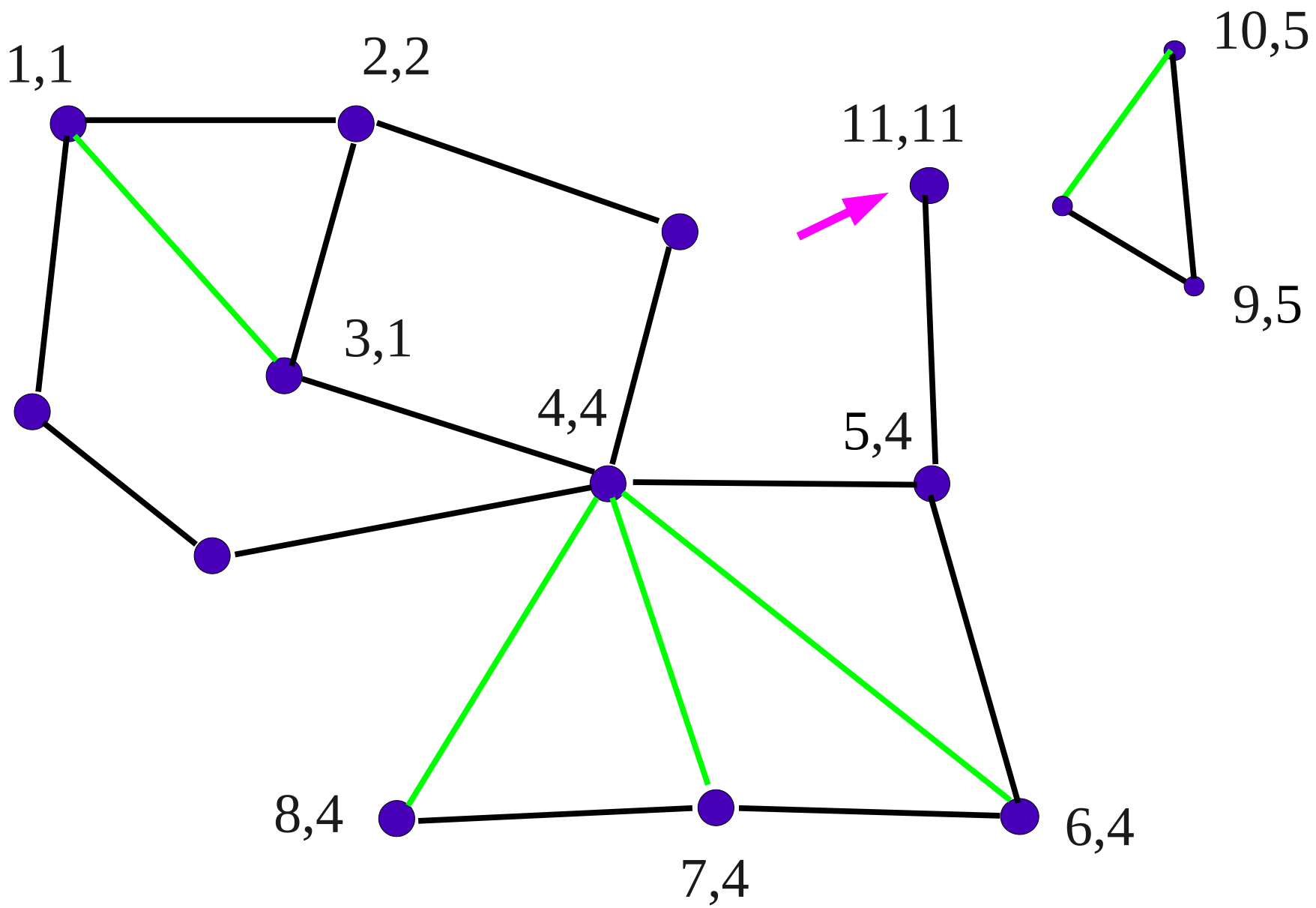
Biconnected Components



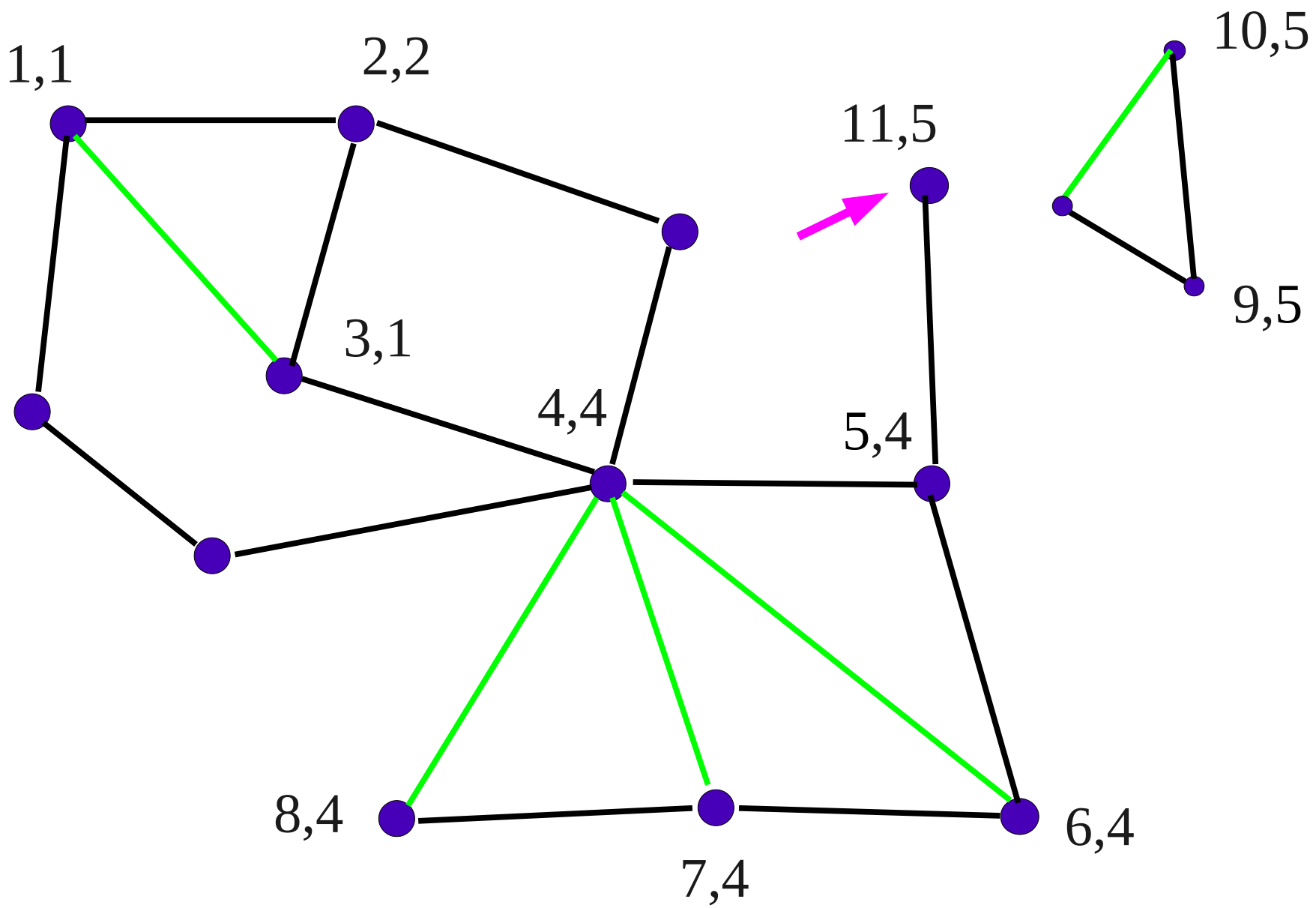
Biconnected Components



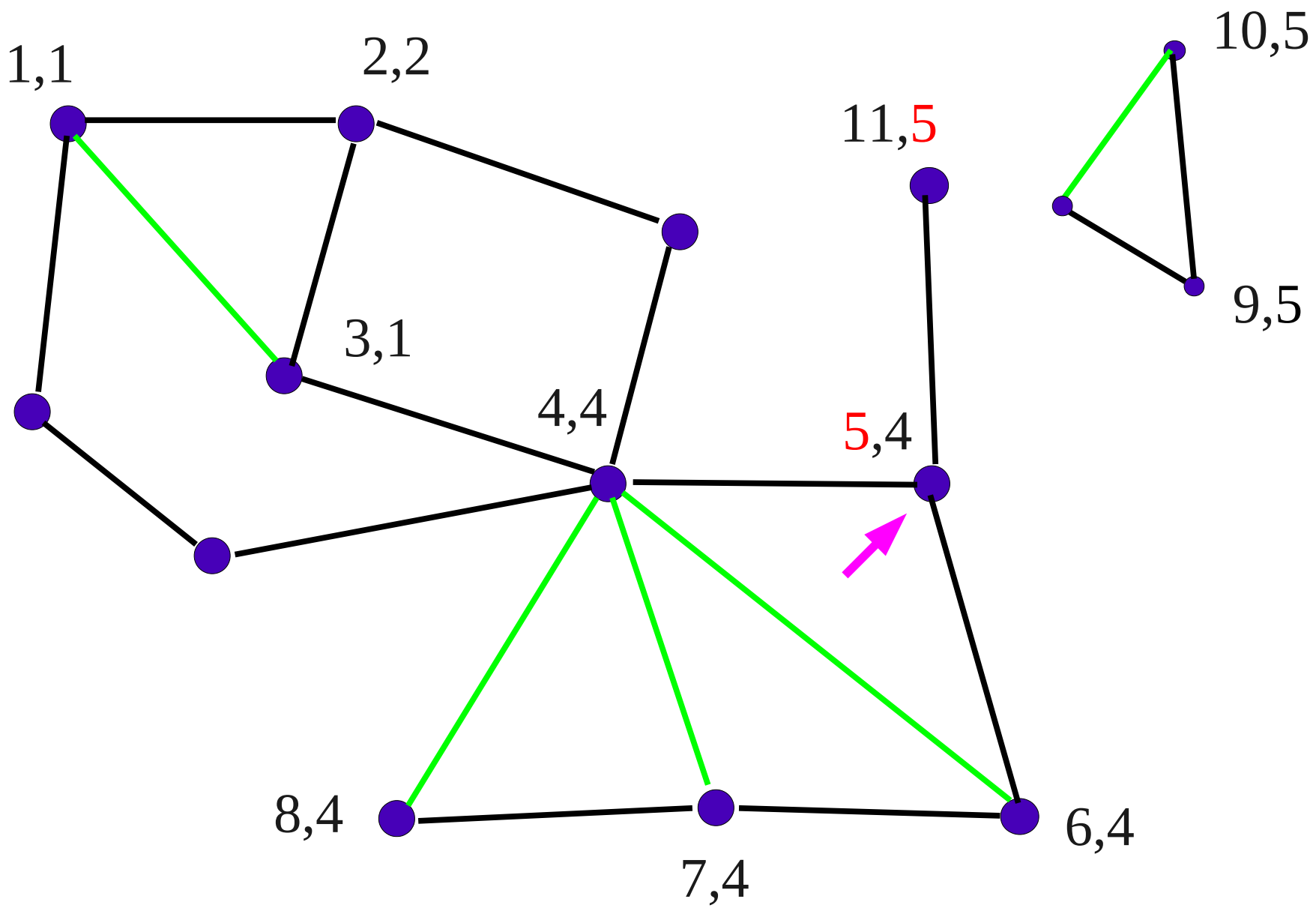
Biconnected Components



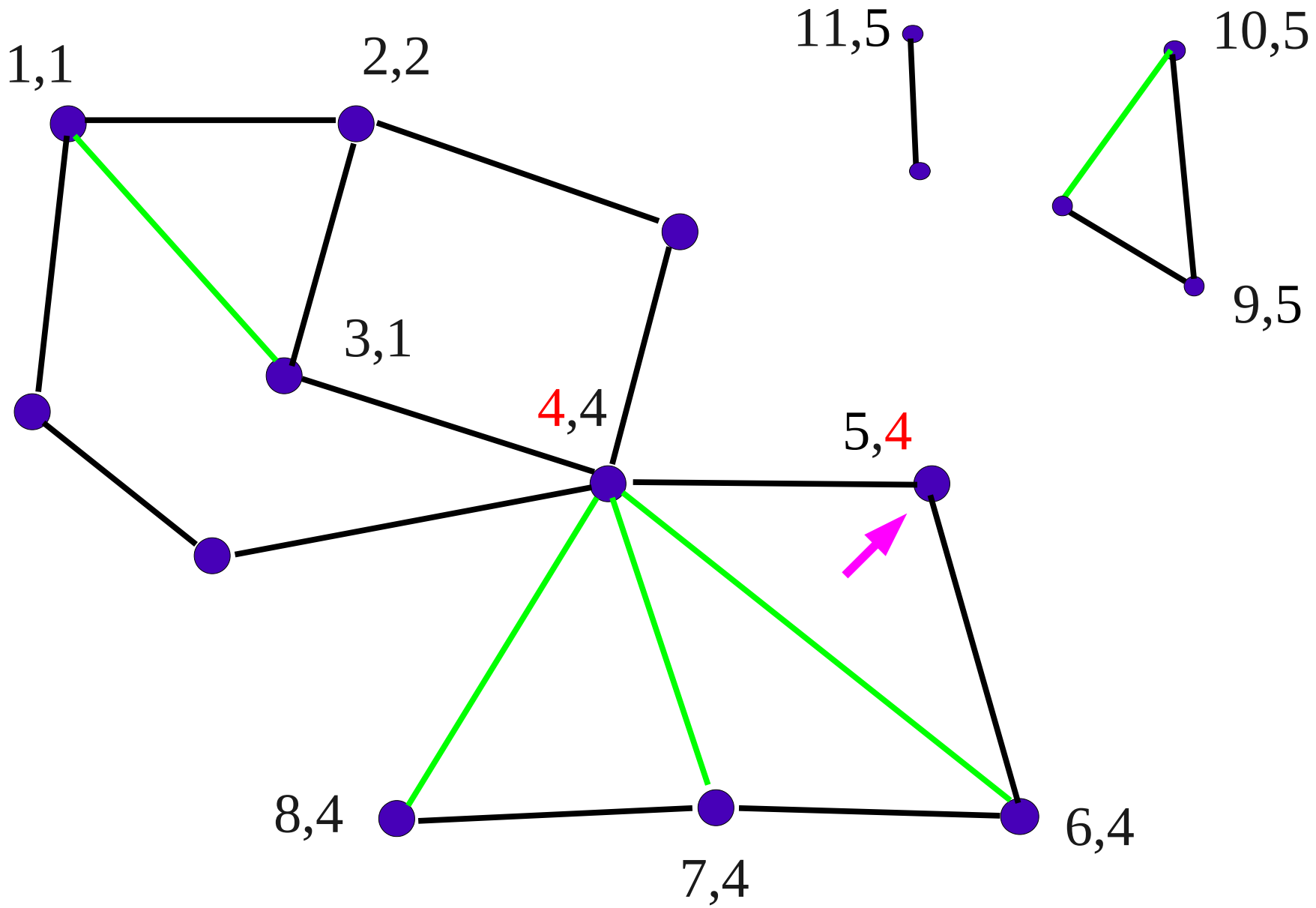
Biconnected Components



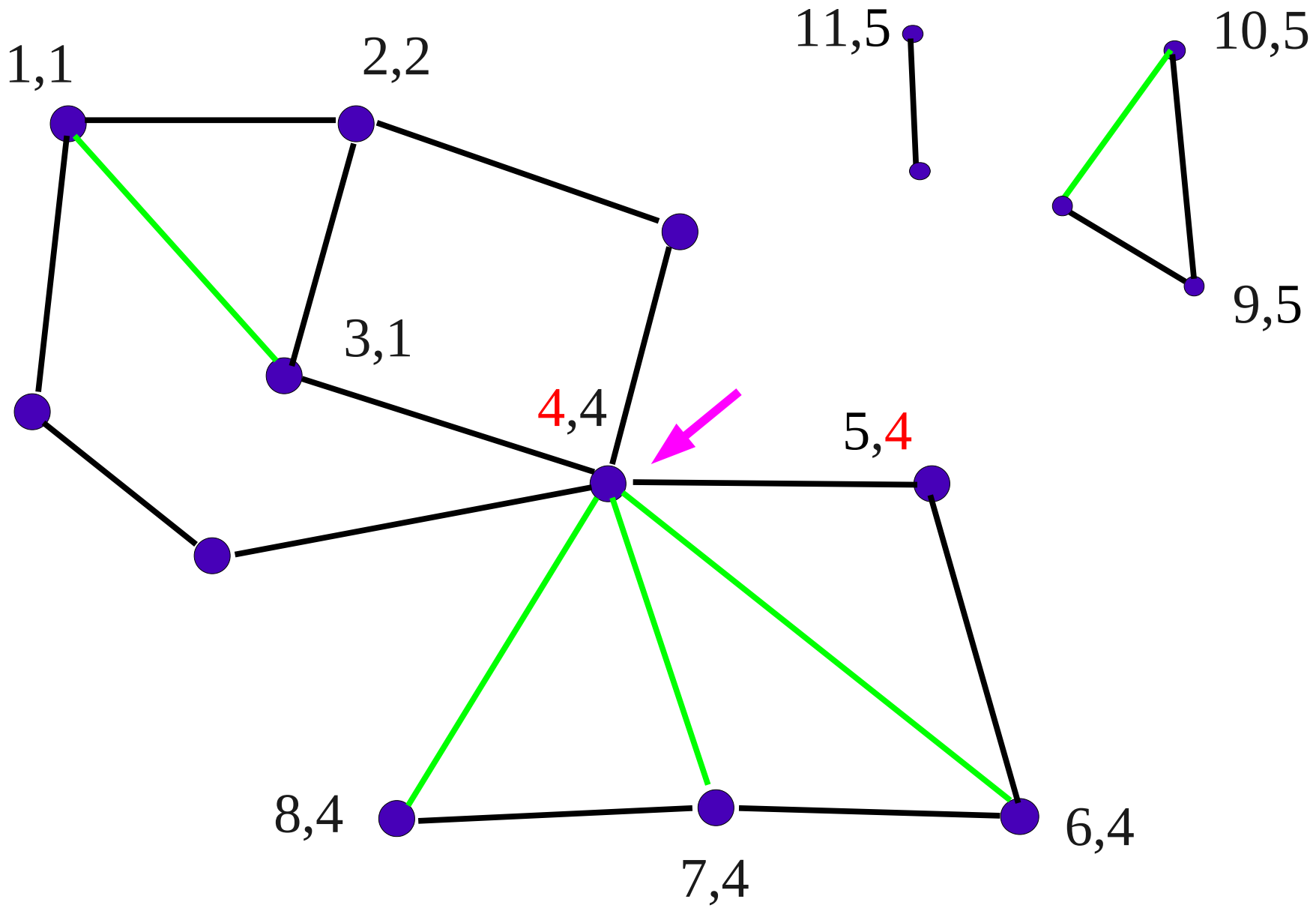
Biconnected Components



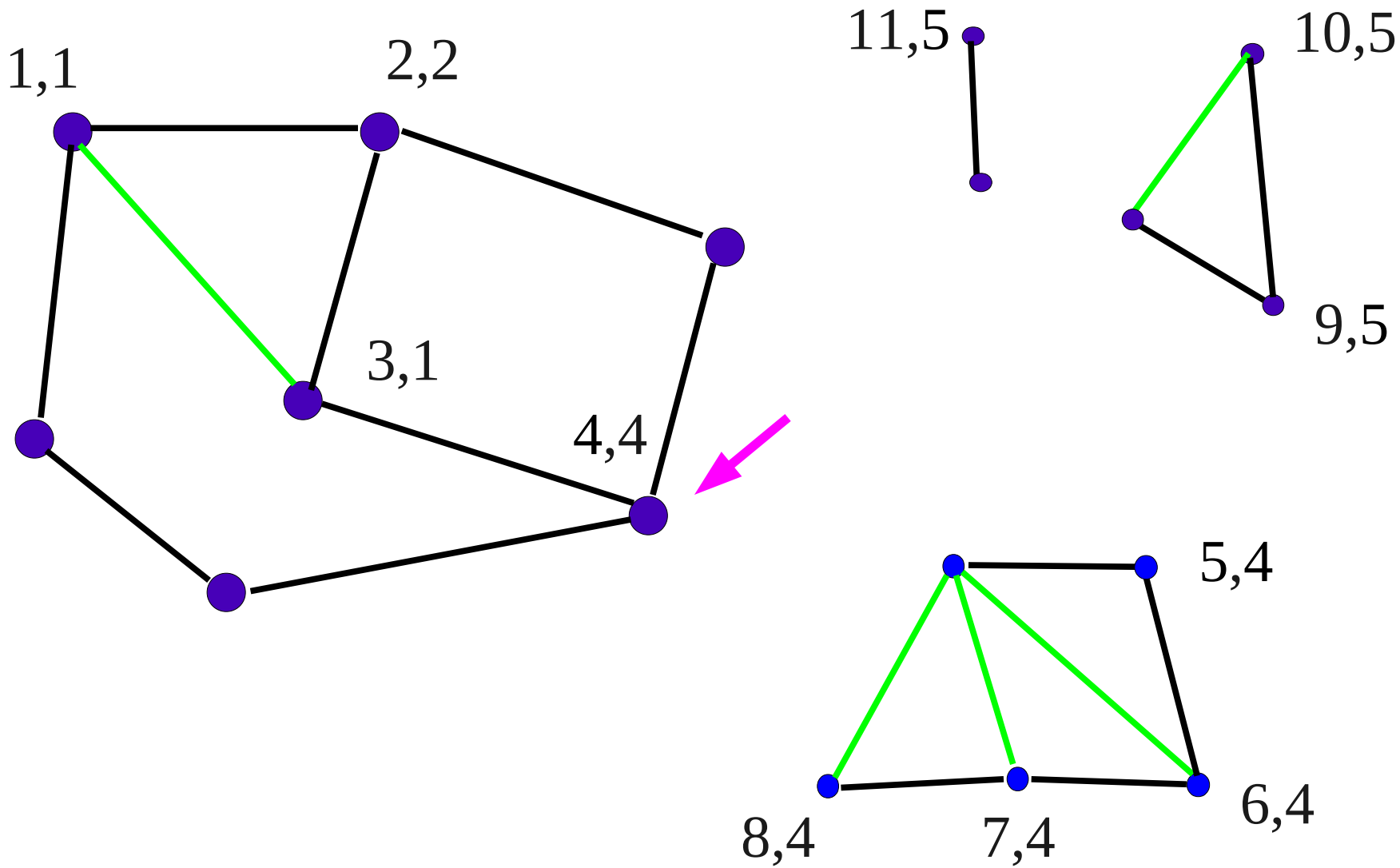
Biconnected Components



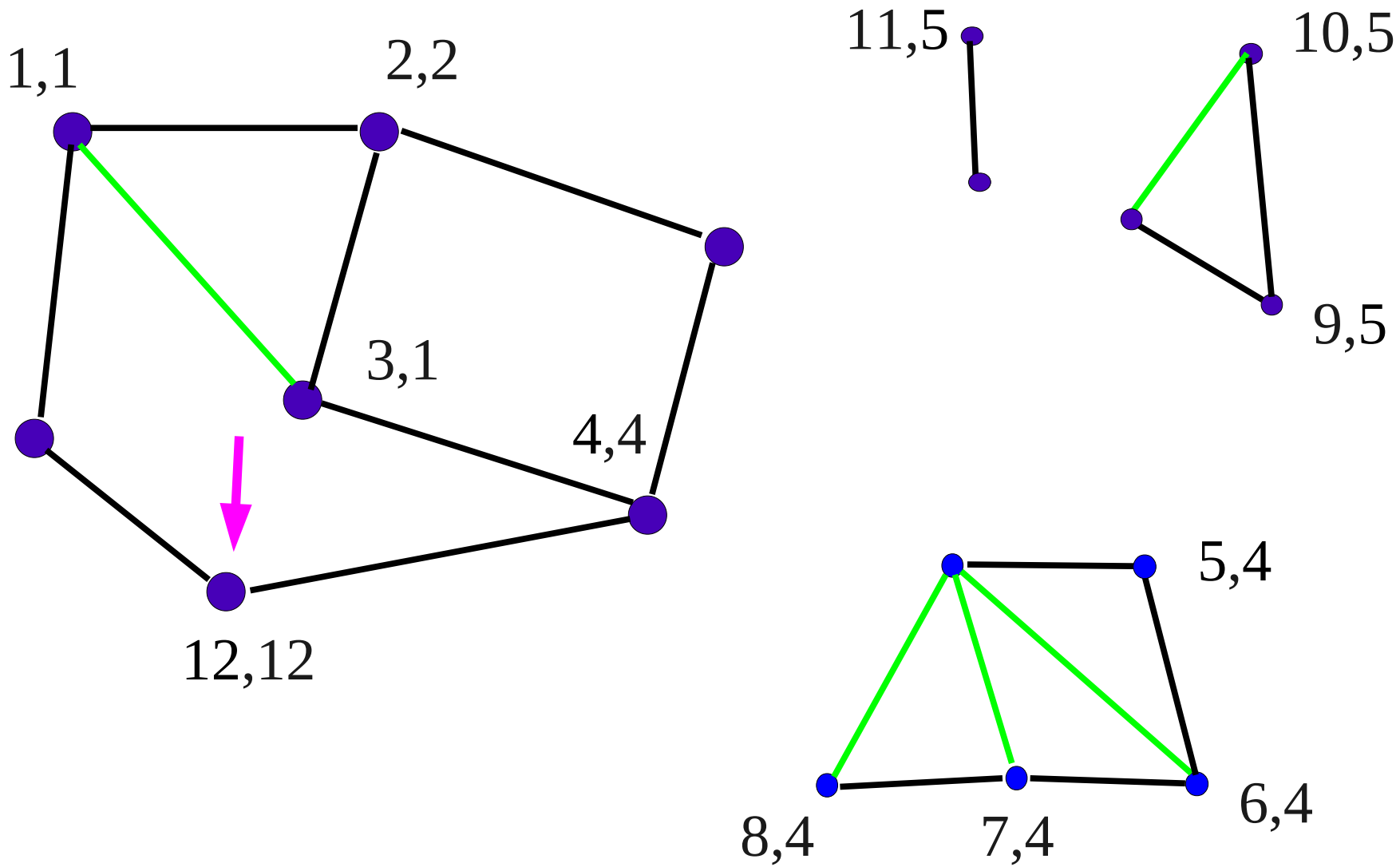
Biconnected Components



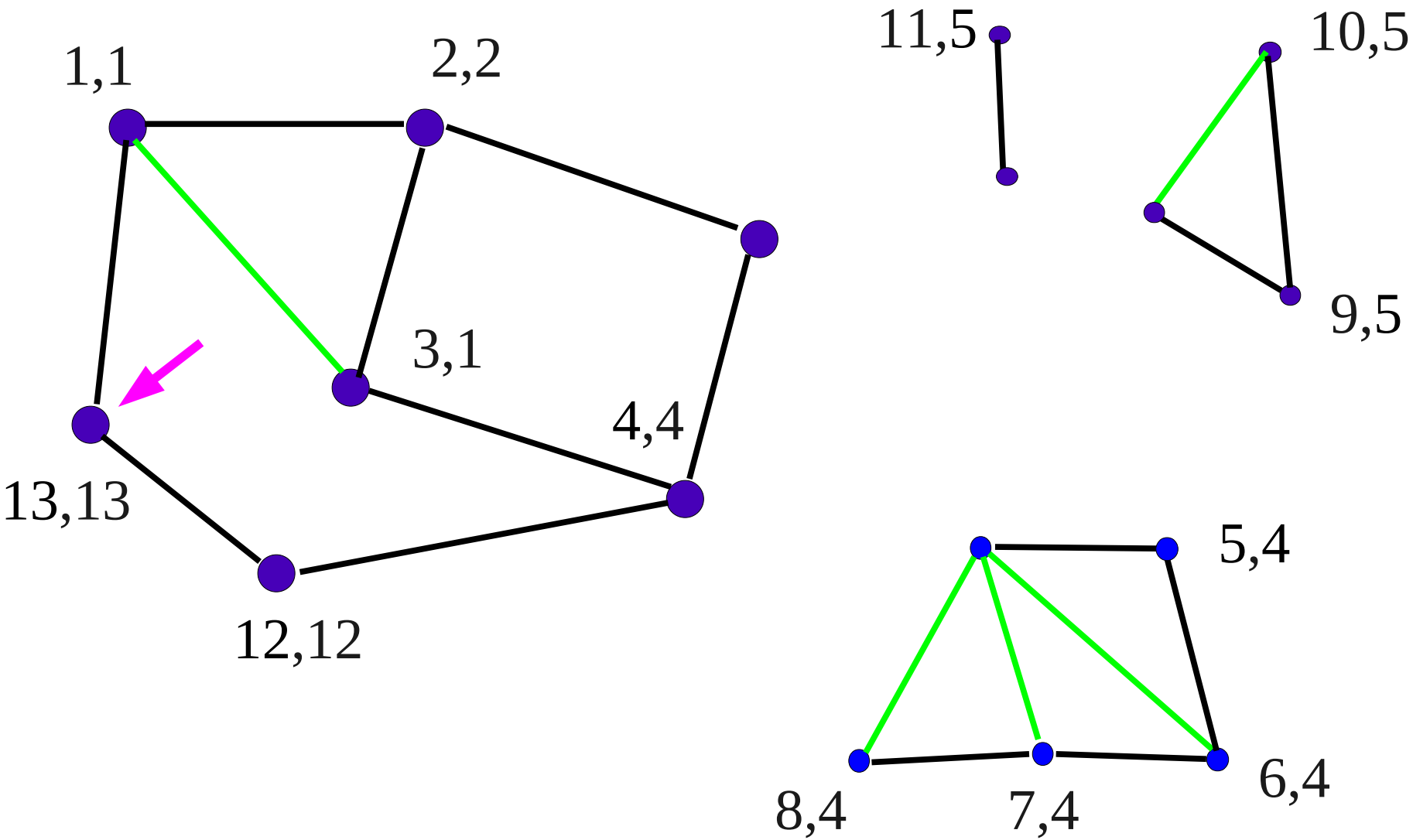
Biconnected Components



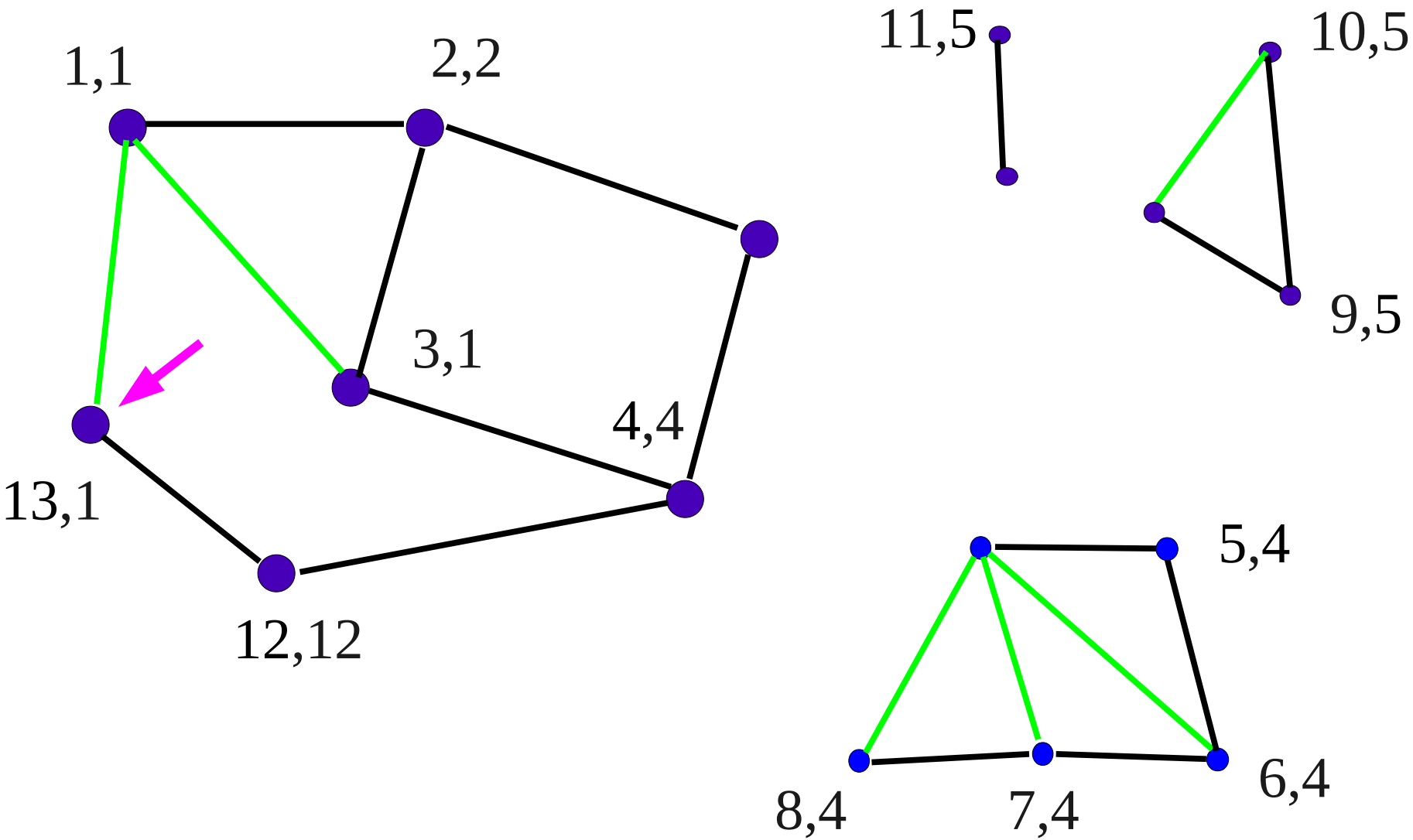
Biconnected Components



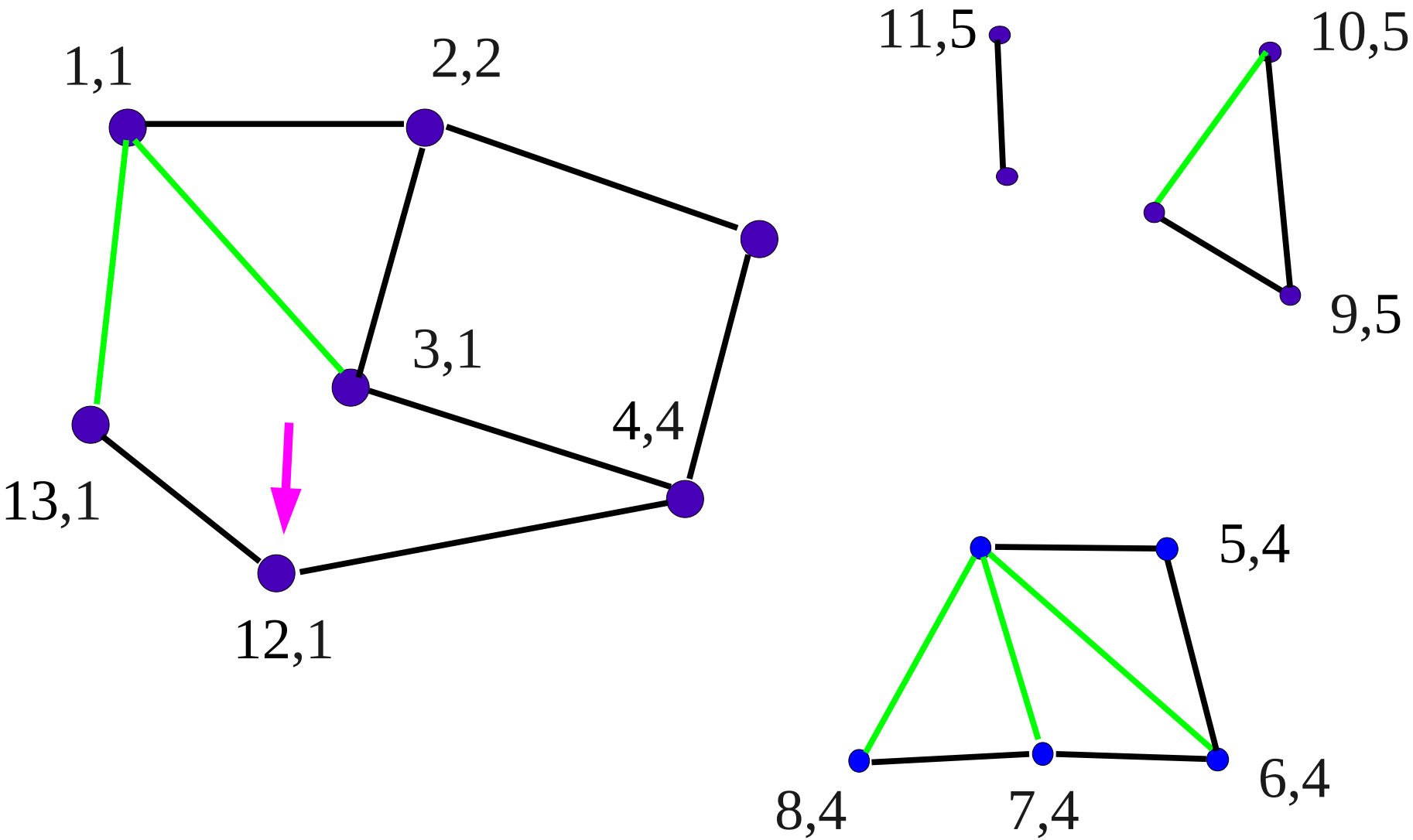
Biconnected Components



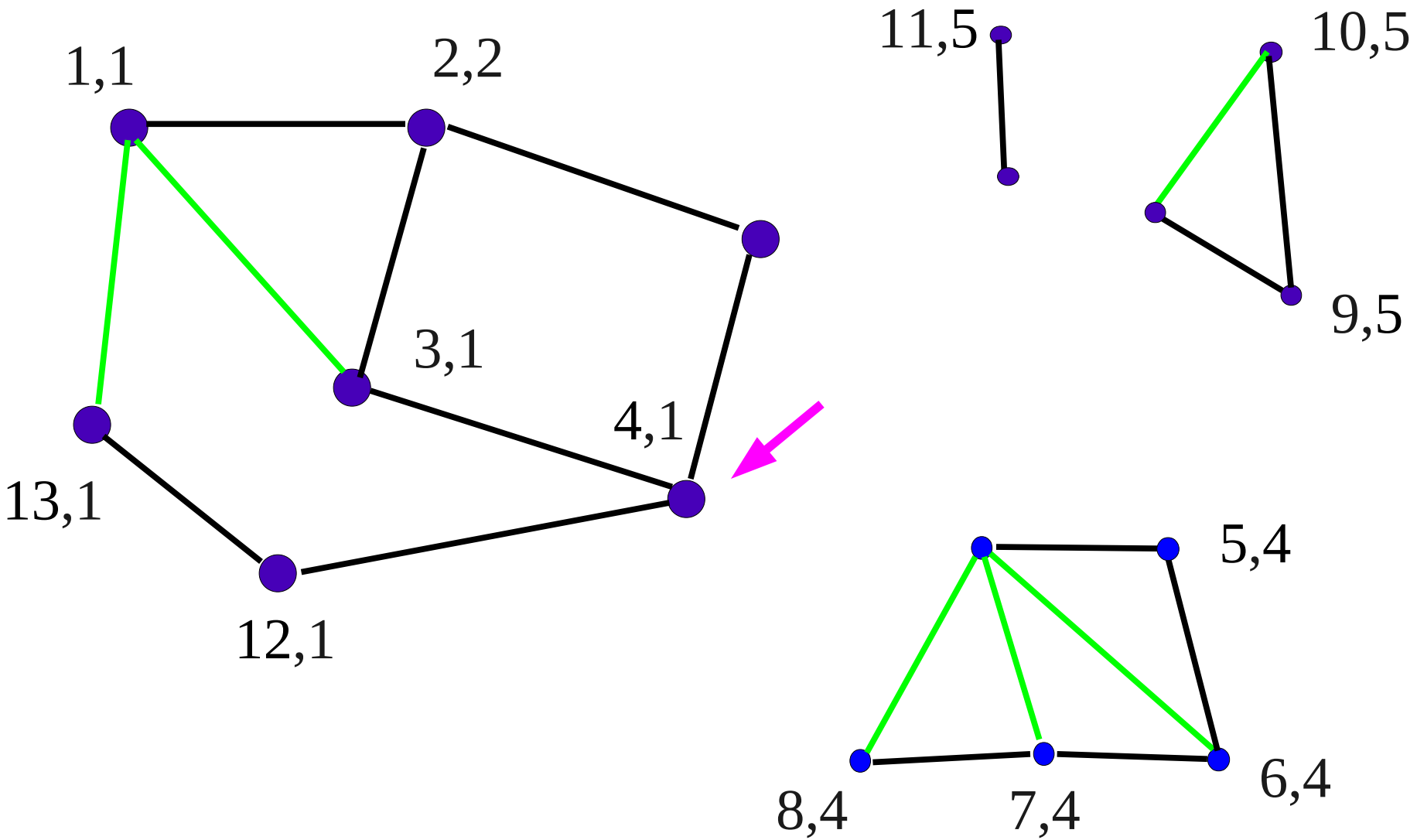
Biconnected Components



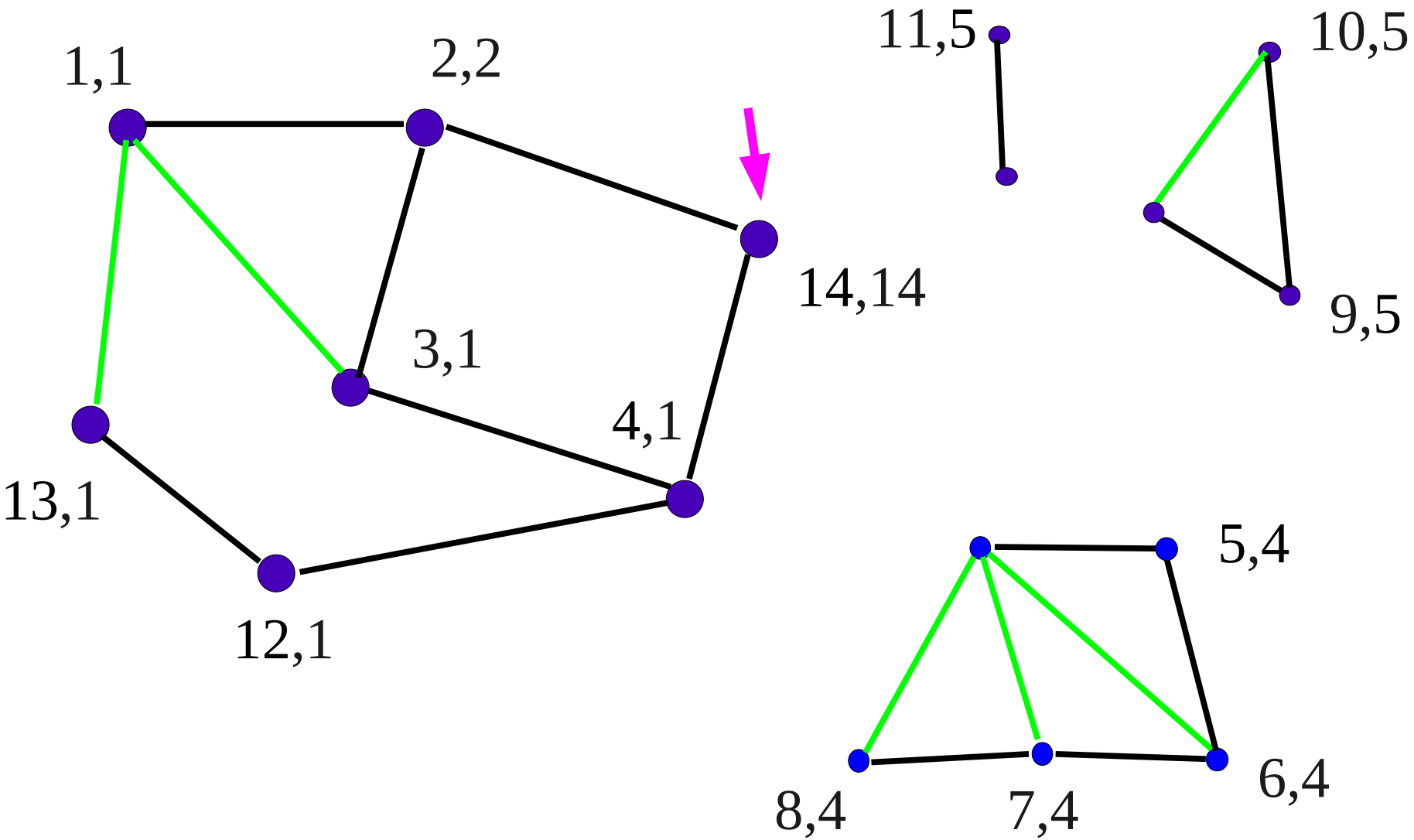
Biconnected Components



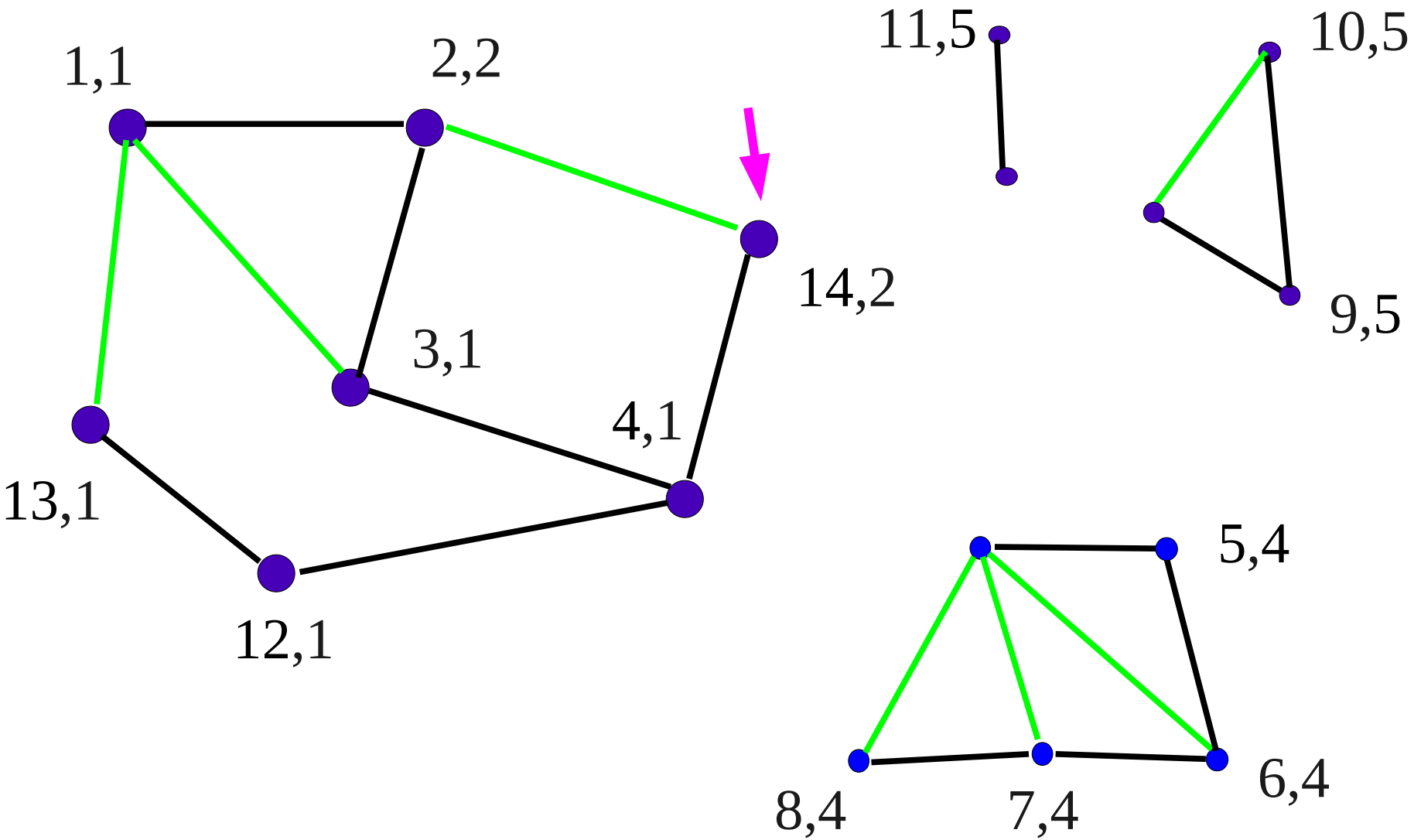
Biconnected Components



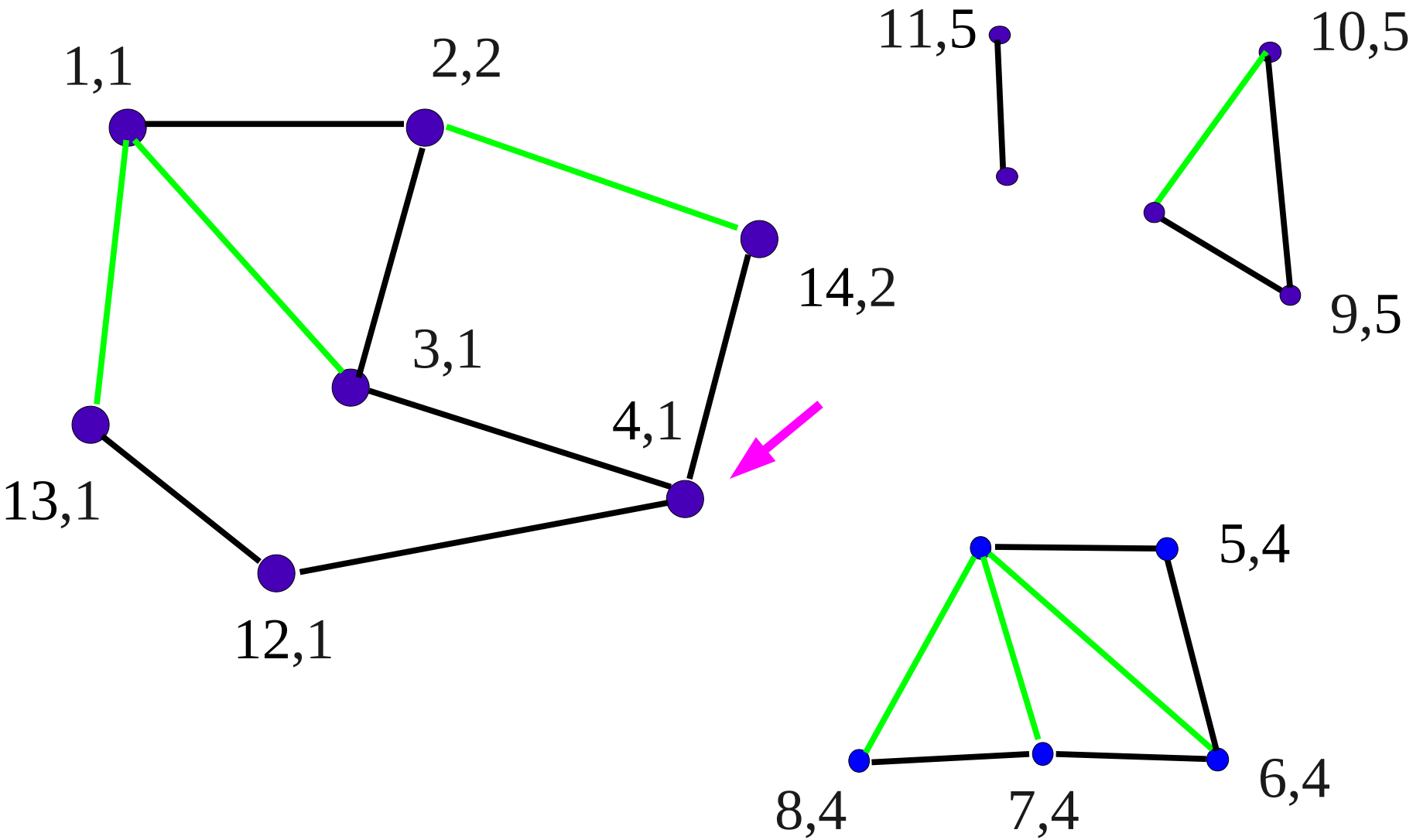
Biconnected Components



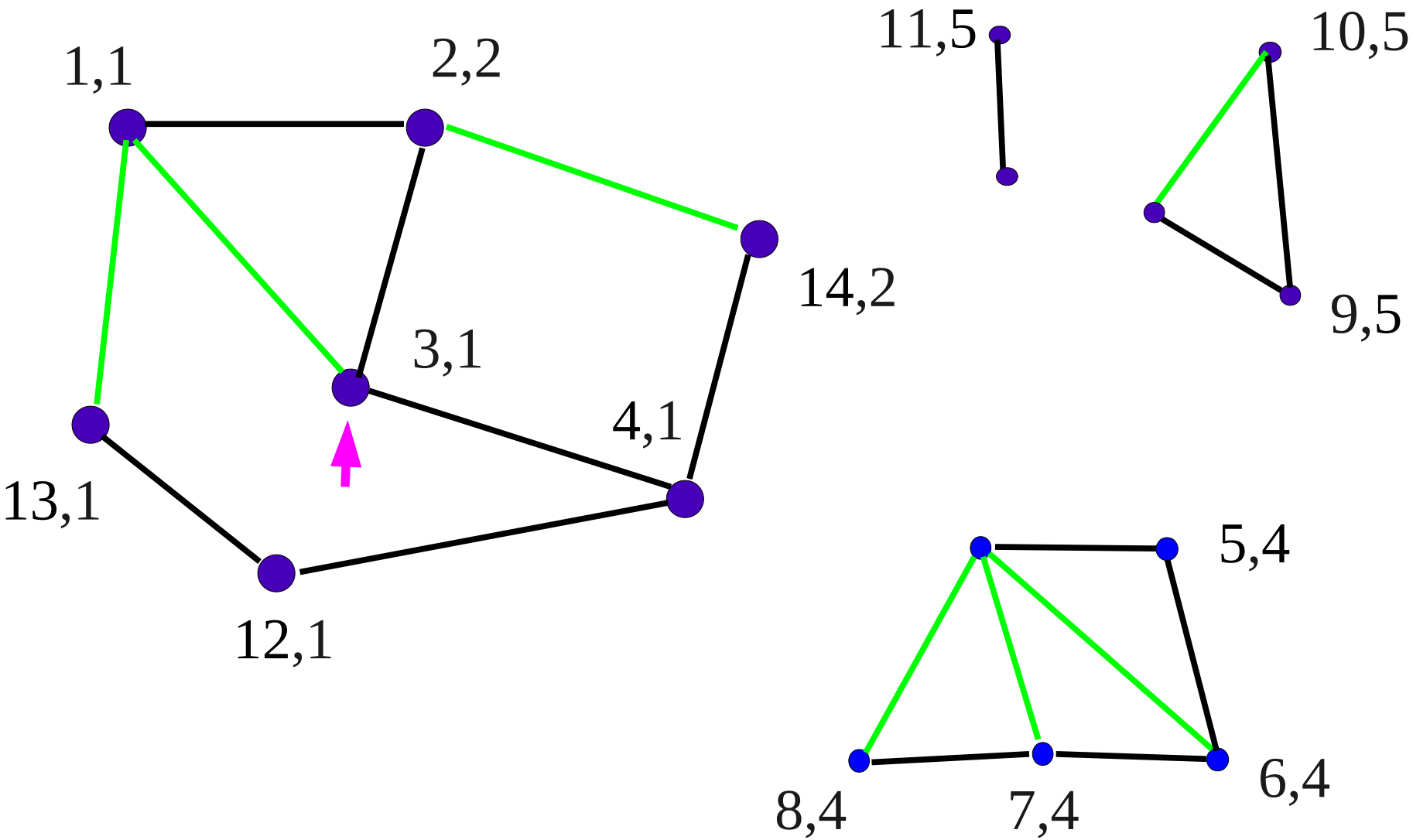
Biconnected Components



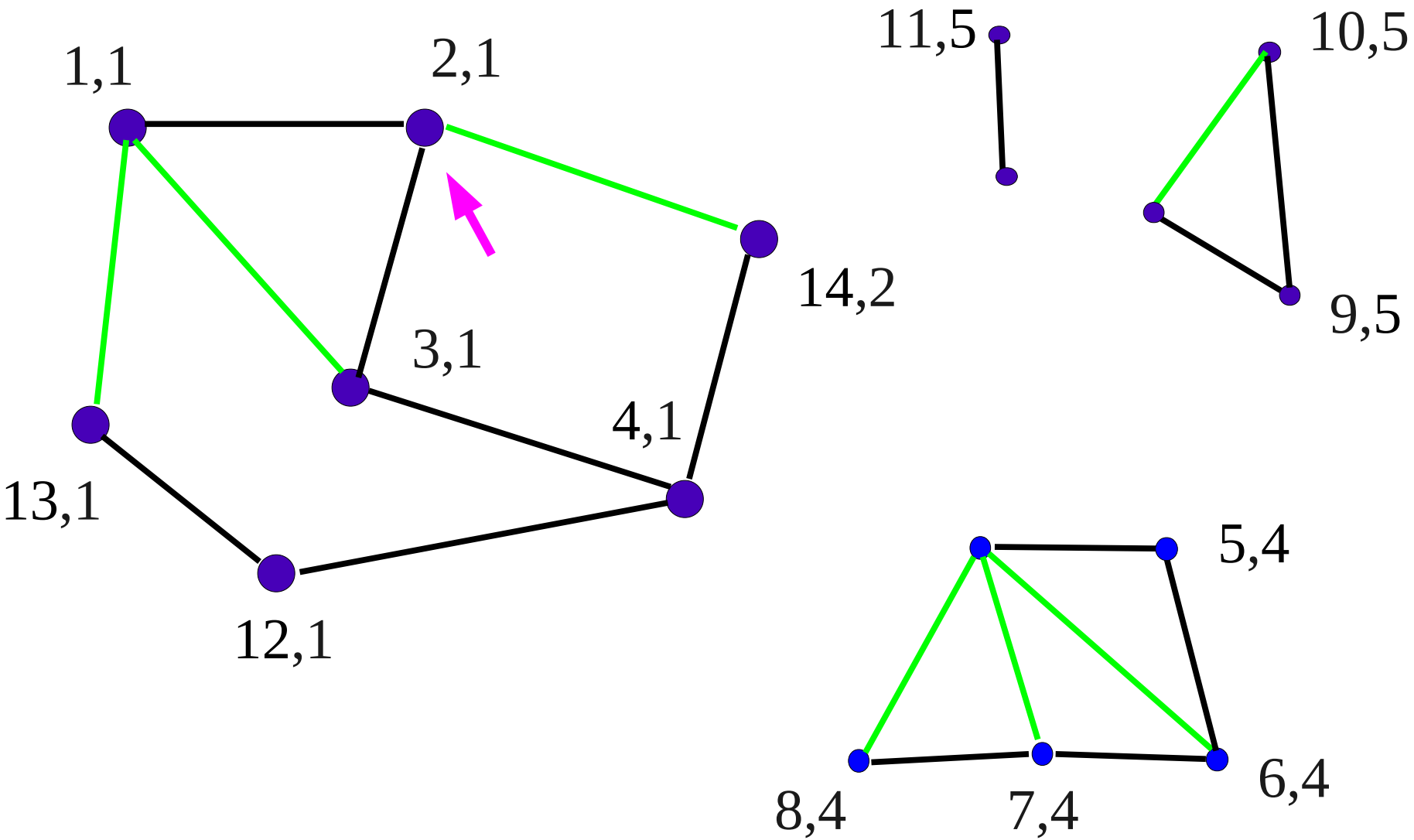
Biconnected Components



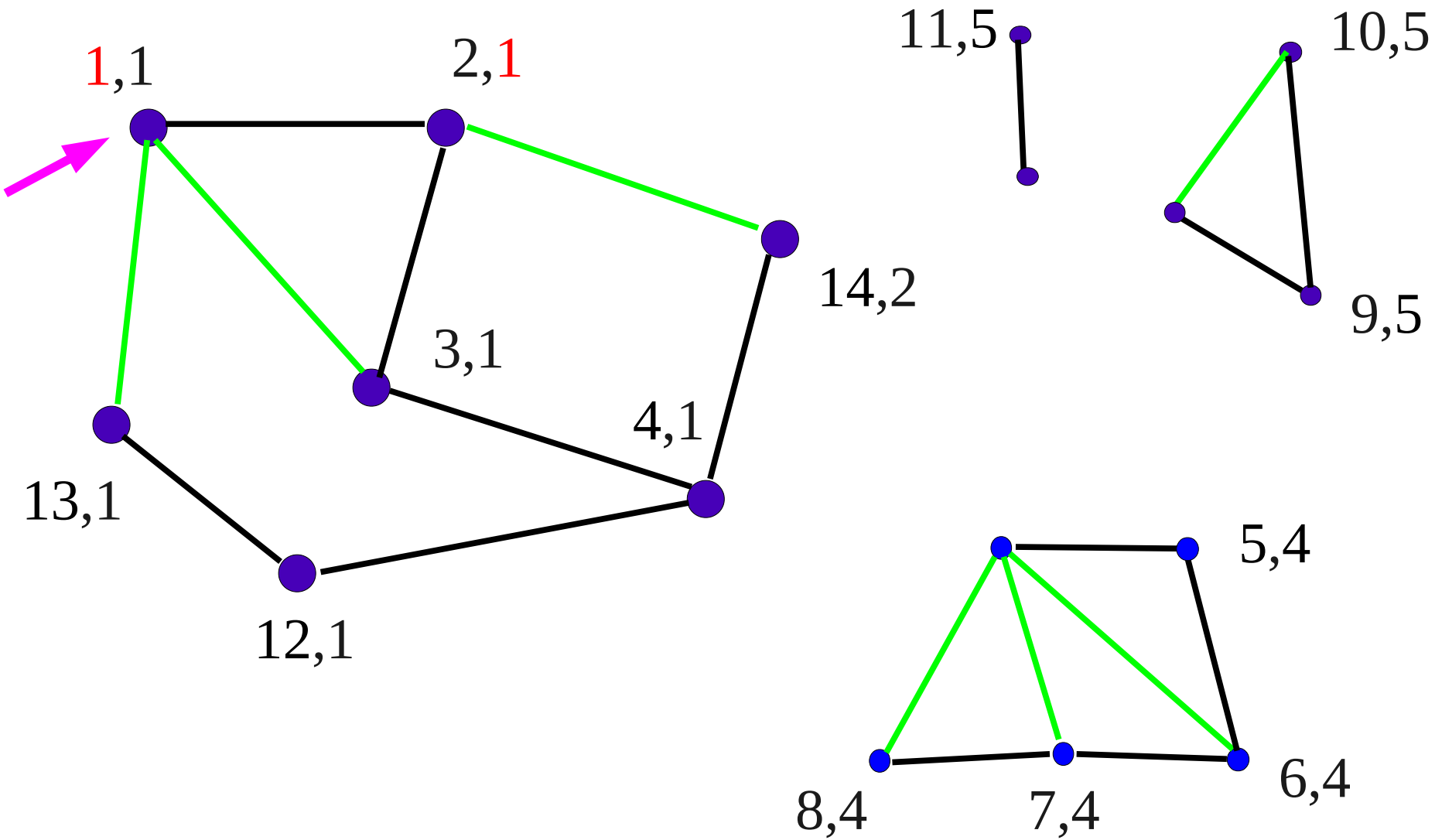
Biconnected Components



Biconnected Components



Biconnected Components



Biconnected Components

```
BICONN(V,E) {
    count = 1;
    for all v ∈ V mark v "new";
    while there is a vertex v ∈ V marked "new" SEARCH(v);
}

SEARCH(v) {
    mark v "old";
    low(v) = DFnumber(v) = count++;
    for each vertex w ∈ V adjacent to and not the parent of v {
      if Dfnumber(w) < Dfnumber(v) then push <v,w> onto STACK;
      if w is marked "new" then {
        SEARCH(w);
        if low(v) ≥ DFnumber(v) then pop from STACK all edges to <v,w>;
        low(v) = min{low(v),low(w)};
      } else {
        low(v) = min{low(v), Dfnumber(w)}
      }
    }
}
```