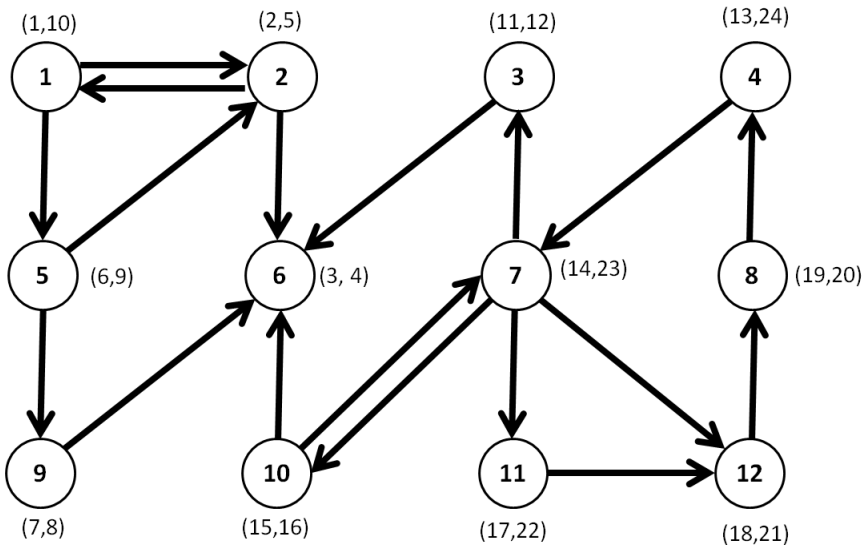
Grundzüge DS & Alg (WS14/15)

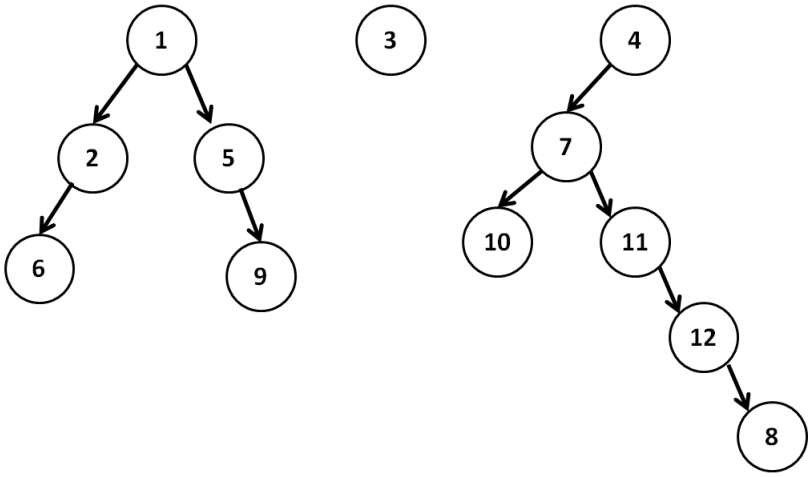
Lösungsvorschlag zu Aufgabenblatt 11

Aufgabe 1

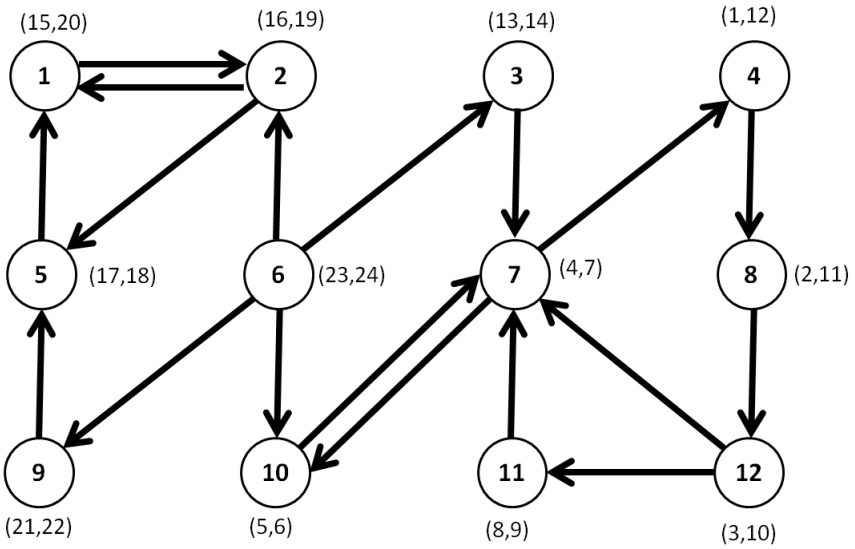
1. Schritt:



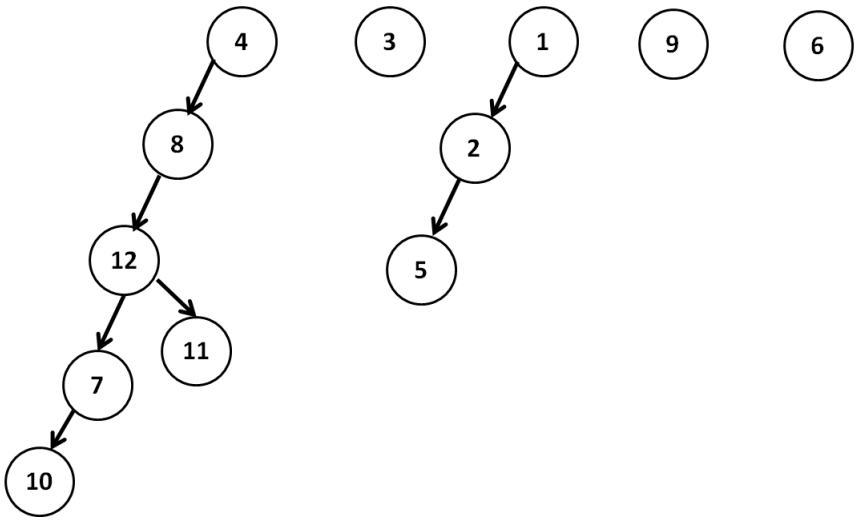
DFS-Wald:



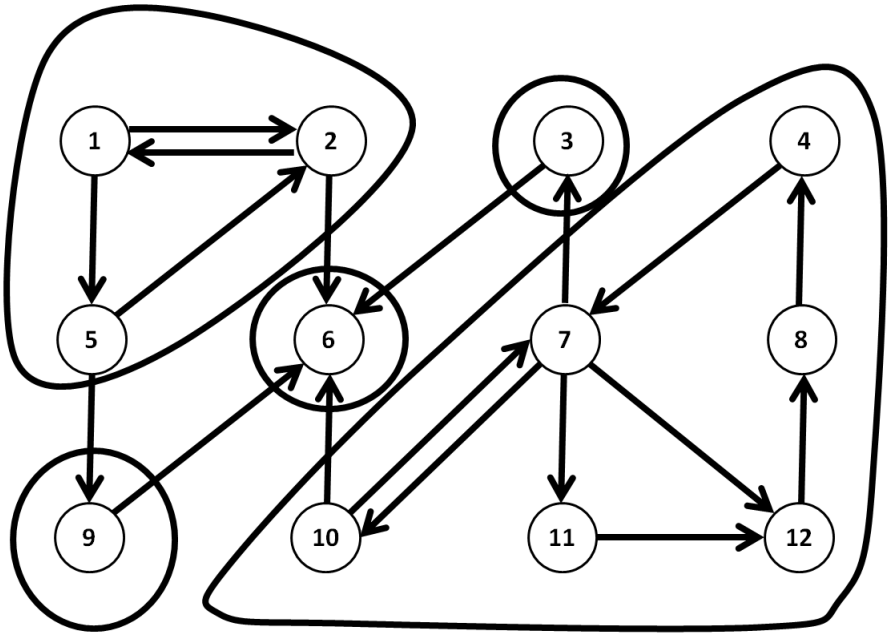
2. Schritt:



DFS-Wald:



Starke Zusammenhangskomponenten:



Aufgabe 2

a)

```
P(a,b):
   Topological_Order(); //assume that now all edges are labelled by their
       topological ordering
   Label a with 1;
   v = the next vertex in topological order;
   while v != b:
      Label v with the sum of all labels of all vertices with incoming edges to v;
   return the sum of all labels of all vertices with incoming edges to b;

Topological_Order(): (c) http://en.wikipedia.org/wiki/Topological_sorting#Algorithms
   L = Empty list that will contain the sorted elements
   S = Set of all nodes with no incoming edges
   while S is non-empty do
      remove a node n from S
      add n to tail of L
      for each node m with an edge e from n to m do
         remove edge e from the graph
         if m has no other incoming edges then
```

```
          insert m into S
  if graph has edges then
    return error (graph has at least one cycle)
  else
    return L (a topologically sorted order)
```

b)

Benutze den Graphen der alle vorwärts gerichteteten Kanten hat. Also hat der erste Knoten Kanten zu allen anderen Knoten, der zweite Knoten zu allen anderen Knoten außer dem ersten usw. Der letzte Knoten hat keine ausgehenden Kanten. Folglich kann keine weitere Kante hinzugefügt werden, da sonst ein Zyklus ensteht und somit der Graph nicht mehr azyklisch wäre. Nun gibt es vom ersten zum zweiten Knoten einen Pfad, vom ersten zum dritten Knoten zwei Pfade, zum 4. 4 Pfade, zum 5. 8 Pfade und zum i-ten Knoten $2^{(i-2)}$ Pfade. Das heißt zum Knoten b gibt es maximal $2^{(|V|-2)}$ mögliche Pfade.

c)

Die Annahme ist, dass Zahlen auch bei exponentiell steigender Größe noch in Zeit O(1) zusammenaddiert werden können.

Aufgabe 3

Um festzustellen, ob der Graph 2-färbbar (bipartit) ist, versuchen wir ihn einfach zu färben. Wenn wir ihn färben können, ist er offensichtlich 2-färbbar.

In den Knoten speichern wir lediglich ab, ob wir den Knoten bereits exploriert haben oder nicht. Wir gehen davon aus, dass die Knoten selbst ihre Nachbarn abgespeichert haben und in einer Liste vorliegen.

```
is_bipartit(V,E) {
    for i=1 to n do{
i.color = undef;
    }
    foreach v in V do{
if (v.color=undef)) {
    color=black;
            if(not set_color(v,color)) {
        return false;
    }
```

```
}
    }
    return true;
}

set_color(v,color) {
    v.color=color;
    color=color.switch;
    for each w in OUT(v) {
if (w.color=undef) {
    if (not set_color(w,color)) {
return false;
    }
}
else if (w.color != color) {
    return false;
}
    }
    return true;
}
```

Mit DFS Method können wir mit einem Knoten anfangen und dann jeweils die Nachbarn mit der jeweils anderen Farbe einfärben. Wenn ohne Konflikt alle Knoten eingefährbt werden, ist das Graph 2-färbbar.

Dieser Algorithmus liegt in $\mathcal{O}(|V| + |E|)$ da es ein DFS ist.

Wenn der Graph nicht zusammenhängend ist, bleiben jetzt einige Knoten unangetastet. Laufen wir jetzt durch die Knotenliste und schauen, welche Knoten noch nicht eingefärbt sind, gehen wir den Graph systematisch ab. Da wir die Liste nur einmal durchlaufen und pro Knoten nur nachsehen müssen, ob er schon eingefärbt ist oder nicht, entstehen so nur $\mathcal{O}(|V|)$ kosten. Finden wir einen uneingefärbten Knoten nachdem der Algorithmus terminiert hat, merken wir, wo wir beim Durchsuchen der Knotenliste waren und rufen den Algorithmus nochmals auf die neu entdeckte Zusammenhangskomponente auf und explorieren so den ganzen Graph.

Eine explizite Partitionierung kann man jetzt erreichen indem man nochmals durch die Knotenliste läuft und nach der Knotenfarbe in separate Listen sortiert.