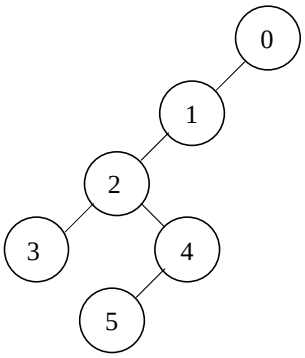
Theoretische Informatik I	Blatt 11	Tutor: Martin Klossek
Sebastian Steinhorst		

Aufgabe 11.1 a)

Baum der Tiefensuche:

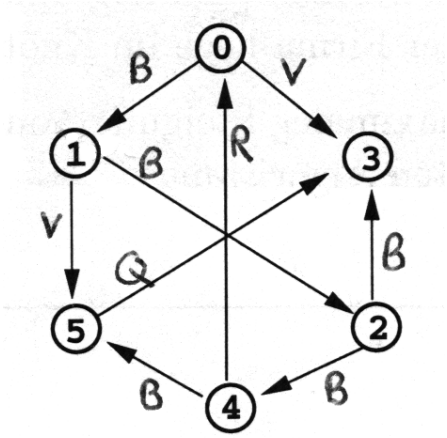


Aufrufhierarchie:

Tiefensuche(0)
Tiefensuche(1)
Tiefensuche(2)
Tiefensuche(3)
Tiefensuche(4)
Tiefensuche(5)

Klassifikation der Kanten:

V = Vorwärtskante
R = Rückwärtskante
B = Baumkante
Q = Querkante



Aufgabe 11.1 b)

V={s,x,y,z}

Initialschritt:

S={s}

setze Start - Distanzen:

Distanz[x]=4 , Distanz[z]=1 , Distanz[y]=∞

1.Schritt

wähle z wegen kleinstem Distanzwert aus V - S

S={s,z}

neu berechnete Distanz – Werte:

Distanz[x]=2 , Distanz[y]=4

2.Schritt

wähle x wegen kleinstem Distanzwert aus V - S

S={s,z,x}

neu berechneter Distanz – Wert:

Distanz[y]=3

3.Schritt

wähle y wegen kleinstem Distanzwert aus V - S

S={s,z,x,y}

S=V ⟶ Algorithmus terminiert mit folgenden Endergebnissen:

Distanz[x]=2 , Distanz[y]=3 , Distanz[z]=1

Aufgabe 11.2

Gesucht: Algorithmus, um aus einem ungerichteten Graphen einen stark zusammenhängenden gerichteten Graphen (Einbahnstraßen) zu erzeugen, falls der Ausgangsgraph dies zulässt.

Idee: Gehe mit Tiefensuche in den Graphen und markiere dabei jede zum ersten Mal besuchte Kante als Einbahnstraße in Suchrichtung. Da der Ausgangsgraph einen stark zusammenhängenden gerichteten Graphen enthalten muss, erhalten wir somit ein Netz von gerichteten Kanten, das unser Einbahnstraßennetz darstellt. Falls der Ausgangsgraph nicht in einen stark zusammenhängenden gerichteten Graphen umzuwandeln ist, gibt der Algorithmus eine Fehlermeldung aus. Der Test, ob der resultierende Graph stark zusammenhängend ist, ist analog zu Skript S. 132 mit der Umkehrung der Kanten und darauf angewendeter Tiefensuche möglich. Besitzt ein Knoten weniger als zwei Kanten, terminiert der Algorithmus sofort, da auf keinen Fall ein stark zusammenhängender Graph entstehen kann.

Die Korrektheit basiert auf der Tatsache, dass jeder Knoten mindestens zwei Kanten besitzen muss, weshalb man im Algorithmus nur dafür sorgen muss, dass jeder Knoten am Ende mindestens Ingrad 1 und Outgrad 1 hat. Dies bewerkstelligt der Algorithmus, da er beim Treffen eines bereits markierten Knotens, der mindestens Outgrad 1 haben muss, eine Kante in diesen hinein erzeugt, womit dieser Knoten mindestens Ingrad 1 erhält. Wenn alle Knoten markiert sind und der Algorithmus terminiert, haben so alle Knoten einen In - und Outgrad von mindestens 1. Den Fall, dass mehr als ein Zyklus im Graph existiert und dazwischen eine Brücke vorhanden ist, übernimmt am Ende der Test, ob der Graph stark zusammenhängend ist.

Der Algorithmus ist effizient, da er auf der Tiefensuche basiert (einmal für **`finde_Einbahnstraßen`** und einmal für **`teste_stark_zusammenhängend)`** und in 2*O(|V|+|E|) = O(|V|+|E|) läuft.

Algorithmus:

```
Annahme:  Gegeben Graph G, |V| ist bekannt

Datenstrukuren:

          Liste test_V

          Einbahnstraßen werden in Adjadzenzmatrix Kante
          gespeichert

finde_Einbahnstraßen(v: Zeiger auf Knoten):

     markiere v

     falls v mindestens einen direkten Nachfolger hat, zu dem
     die Kante noch nicht besucht wurde:
```

```
          für alle direkten Nachfolger w von v:

               Kante[v,w]=Einbahnstraße
               wenn w nicht markiert:

                    finde_Einbahnstraßen(w)

     sonst:

          Ausgabe „Kein Netz von Einbahnstraßen erstellbar!“
          verlasse das Programm


teste_stark_zusammenhängend(v: Zeiger auf Knoten):

     markiere v

     für alle direkten Nachfolger w von v:

          wenn Kante[w,v]=Einbahnstraße:

               wenn w nicht markiert:

                    test_V.append(w)
                    teste_stark_zusammenhängend (w)
```

Hauptprogramm:

```
Wähle einen beliebigen Knoten v aus dem Graphen

wenn v mindestens zwei Kanten besitzt:

     finde_Einbahnstraßen(v)

sonst:

     Ausgabe „Kein Netz von Einbahnstraßen erstellbar!“
     verlasse das Programm

teste_stark_zusammenhängend(v)

wenn |V| = |test_V|:

     Ausgabe „Netz von Einbahnstraßen erstellt!“

sonst:

     Ausgabe „Kein Netz von Einbahnstraßen erstellbar!“
```

Aufgabe 11.3

Gesucht: Algorithmus, um von Startknoten s aus Wege zu allen anderen Knoten mit geringster maximaler Steigung in einem gegebenen Graph G zu ermitteln.

Idee: Modifikation von Dijkstra's Algorithmus. Wenn wir statt der Distanz das Maß MaxSteigung einführen, können wir alle Wege mit geringster maximaler Steigung von s zu allen Knoten in G bestimmen.

MaxSteigung – Algorithmus(für G=(V,E)):

Setze S={s} und

$$\text{MaxSteigung}[v] = \begin{cases} \text{Steigung}(s,v) & \text{wenn } (s,v) \in E \\ \infty & \text{sonst.} \end{cases}$$

Solange S != V wiederhole

- Wähle einen Knoten w ε V – S mit kleinstem MaxSteigung – Wert.
- Füge w in S ein.
- Berechne die neuen MaxSteigung – Werte der Nachfolger von w. Insbesondere setze für jeden Nachfolger u ε V \ S von w:

```
c = Max(MaxSteigung[w] , Steigung(w,u));
MaxSteigung[u]=(MaxSteigung[u] > c) ? c : MaxSteigung[u];
```

Korrektheit und Laufzeit ergeben sich analog zu Dijkstra's Algorithmus, es wurde lediglich das Bewertungskriterium für die Kantengewichtung geändert, was aber ansonsten keine Auswirkung auf den Ablauf des Algorithmus hat.