12. AuD Tafelübung T-C3

Simon Ruderich

2. Februar 2011

Kollisionen

(Primär)Kollision

- Stelle mit „normal“ eingefügtem Element schon belegt (gleicher Hashwert)
- tritt bei verketteten Listen und Sondierung auf

Sekundärkollision

- Stelle mit Element aus Sondierung besetzt (meist anderer Hashwert)
- tritt nur bei Sondierung auf
- Risiko kann durch gute Sondierfunktion reduziert werden

String.substring()

String.substring()

- gibt einen Teilstring zurück
- Originalstring wird nicht verändert
- `substring(int beginIndex)`
- `substring(int beginIndex, int endIndex)`

Beispiele

```
"Test-String".substring(0); // "Test-String"
"Test-String".substring(5); // "String"
"Test-String".substring(11); // ""
"Test-String".substring(12); // Exception!
"Test-String".substring(5, 9); // "Stri"
```

String.charAt()

String.charAt()

- liefert den `char` ($\neq$ `String`) an der Position zurück
- `String.charAt(int index)`

Beispiele

```
"Test-String".charAt(0);  // 'T'
"Test-String".charAt(4);  // '-'
"Test-String".charAt(10); // 'g'
"Test-String".charAt(11); // Exception!
```

Bäume

Bäume

- Bäume sind Graphen in denen es keine Zyklen gibt
- immer *ein* Elter (oder keine bei der Wurzel)
- beliebige Anzahl von Kindern (auch Null)

Suchen

Tiefensuche

- zuerst so weit wie möglich im Baum (Graph) absteigen
- sobald man unten angekommen ist, "back-tracking"
- dann die anderen Kinder des Vaters analog besuchen
- kann mit Stack implementiert werden

Breitensuche

- immer Knoten der gleichen Tiefe der Reihe nach besuchen
- dann eine Ebene absteigen
- kann mit Queue (Warteschlange) implementiert werden

LexTree

Aufgabe

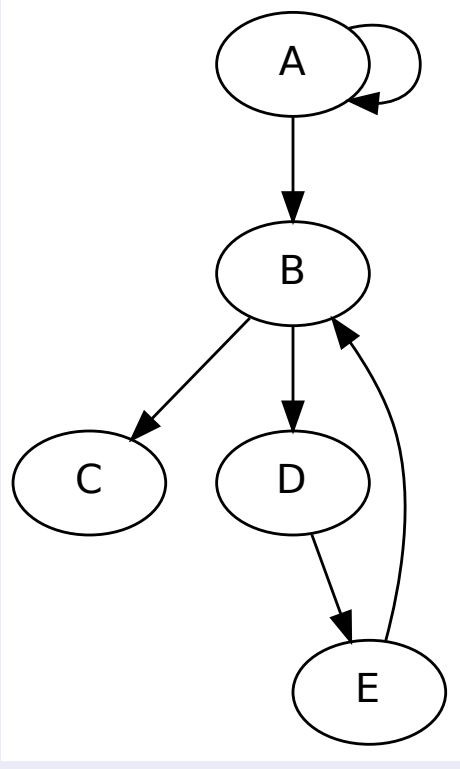
Graphen

Graphen

- E Kanten, V Knoten
- e Anzahl der Kanten, v Anzahl der Knoten

Darstellung (gerichteter Graph)

Graph

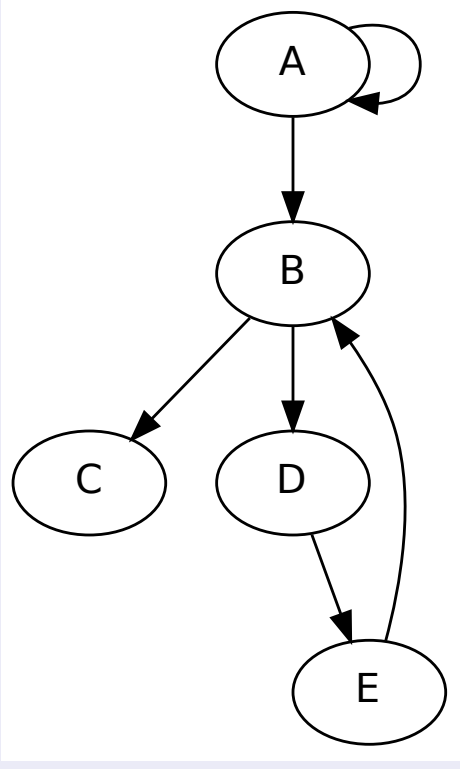


Adjazenzmatrix

	A	B	C	D	E
A	1	1	0	0	0
B	0	0	1	1	0
C	0	0	0	0	0
D	0	0	0	0	1
E	0	1	0	0	0

Darstellung (gerichteter Graph)

Graph

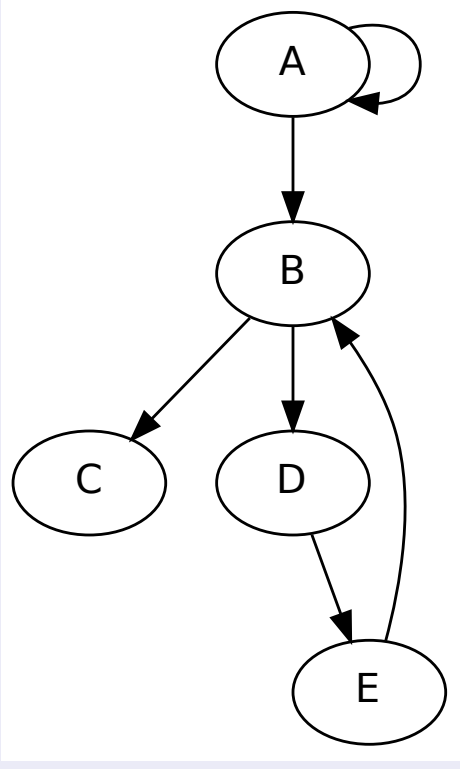


Adjazenzliste

A	A B
B	C D
C	
D	E
E	B

Darstellung (gerichteter Graph)

Graph



Menge

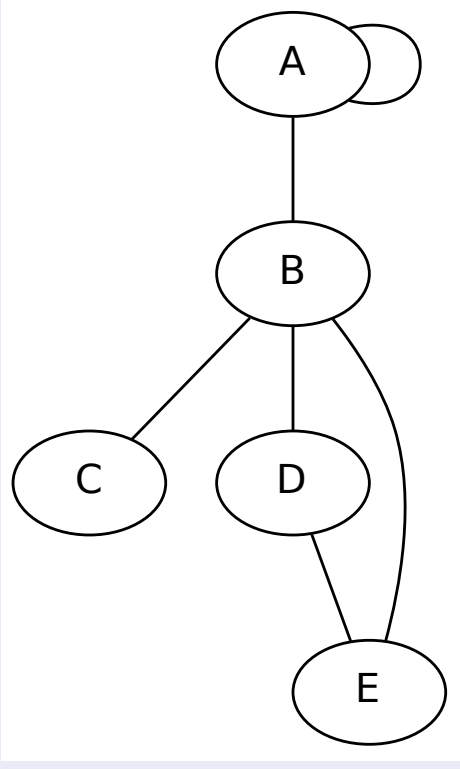
$$G = V, E$$

$$V = A, B, C, D, E$$

$$E = \{(A, A), (A, B), (B, C), (B, D), (D, E), (E, B)\}$$

Darstellung (ungerichteter Graph)

Graph

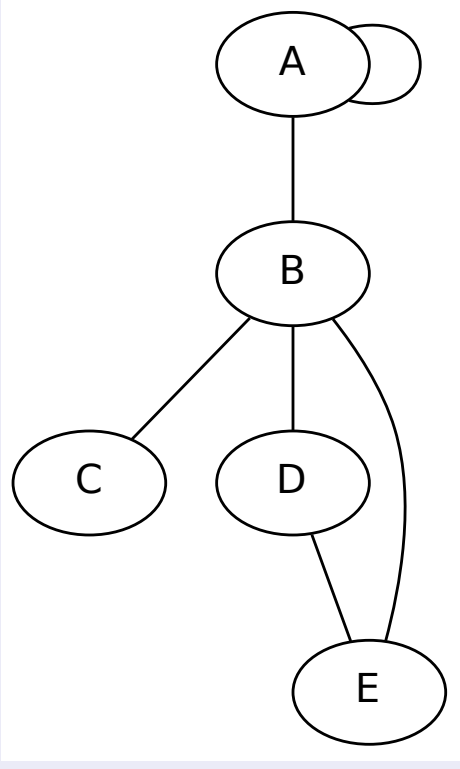


Adjazenzmatrix

	A	B	C	D	E
A	1	1	0	0	0
B	1	0	1	1	1
C	0	1	0	0	0
D	0	1	0	0	1
E	0	1	0	1	0

Darstellung (ungerichteter Graph)

Graph

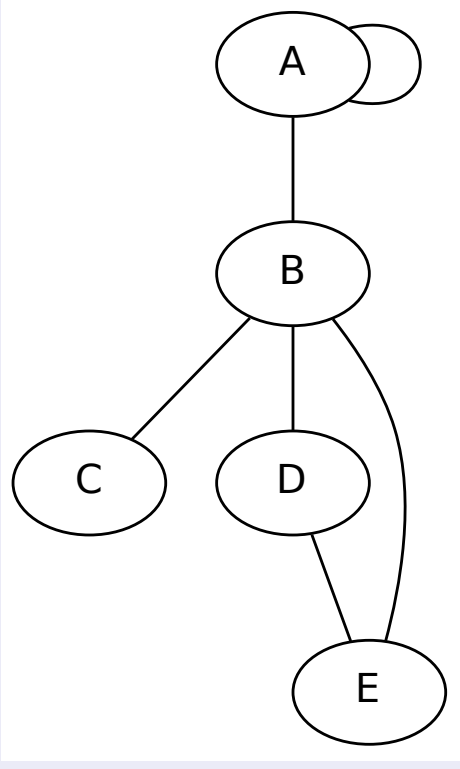


Adjazenzliste

A	A B
B	A C D E
C	B
D	B E
E	D B

Darstellung (ungerichteter Graph)

Graph



Menge

$$G = V, E$$

$$V = A, B, C, D, E$$

$$E = \{[A, A], [A, B], [B, C], [B, D], [D, B], [B, E], [E, B]\}$$

Dijkstra

Dijkstra

- findet kürzesten Pfad zu allen Knoten
- Aufwand: $\mathcal{O}(e + v \log v)$

Beschreibung

- verwalte alle erreichbaren Knoten in Liste
- trage Startknoten in Liste ein

1. entnehme Knoten mit geringstem Gewicht
2. markiere entnommenen Knoten als „fertig“
3. trage (ggf. neue) Kosten zu allen erreichbaren Knoten ein
   billigere Kosten werden aktualisiert, teurere ignoriert
4. trage neu erreichbare Knoten in Liste ein
5. wiederhole 1 bis 4 bis alle Knoten besucht wurden

Prim

Prim

- findet minimalen Spannbaum
- Aufwand: $\mathcal{O}(e + v \log v)$

Beschreibung

- verwalte alle erreichbaren Kanten in Liste
- trage Kanten des Startknoten in Liste ein

1. füge Kante mit minimalem Gewicht dem Spannbaum hinzu
2. füge Kanten des neuen Knoten der Liste hinzu
3. streiche Kanten die zwei Elemente des Spannbaums verbinden
4. wiederhole 1 bis 3 bis alle Knoten besucht wurden

Kruskal

Kruskal

- findet minimalen Spannbaum
- Aufwand: $\mathcal{O}(e \log e)$

Fragen

Fragen?

zusätzliche Fragen für nächste Stunde bitte per Mail an simon@ruderich.org