Algorithmen im Chip-Entwurf 1

Probleme, Werkzeuge und Graphen

Andreas Koch
FG Eingebettete Systeme
und ihre Anwendungen
TU Darmstadt

Orga 1 - Material

- Grundlage der Vorlesung
  - *Algorithms for VLSI Design Automation*
    Sabih H. Gerez
- Wissenschaftliche Arbeiten („Papers“)
- Wissenstiefe
  - Kein perfektes Verständnis ...
  - ... aber Überblick über das Material
    - Fragen stellen!

Orga 2 - Aufbau

- Integrierte Veranstaltung
  - Zu Beginn: Nur Vorlesung
  - Dann: praktische Programmierarbeit
  - In Gruppen
  - Kolloquien
  - Vorträge
- Kick-Off zu den praktischen Arbeiten
  - Anfang KW 44 (= 2. Semesterwoche)
  - Vorher Leitfaden lesen!

Orga 3 - Prüfungsleistung

- Benotete Prüfungsleistung
  - Beginnend *in* 4. Semesterwoche
  - Gewertet
    - Programme
    - Kolloquien
    - Vorträge
- Individuelle Prüfung
  - Nur in Zweifelsfällen
  - In letzter Semesterwoche (KW 7)

Orga 4 – Zeitplan und WWW

- Zeitplan
  - Vorlesung
    - KW 43&44: Di+Fr, KW 45...2: Nur Di
  - Praktischer Teil
    - Vorträge KW 46, 50, 3: Fr, KW 7: Di
    - Kolloquien KW 46, 50, 3, 6: Do vormittags
- Web-Seite
  - http://www.esa.informatik.tu-darmstadt.de Unterpunkt „Lehre“
  - Material und Ankündigungen

Überblick

- VLSI Entwurf
  - Probleme
  - Bereiche
  - Tätigkeiten
  - ➔Werkzeuge
- Hierarchie und Abstraktion
- Algorithmische Graphentheorie
  - Strukturen
  - Verfahren

VLSI Entwurfsproblem

„Implementiere eine Spezifikation in Hardware und optimiere dabei ...“

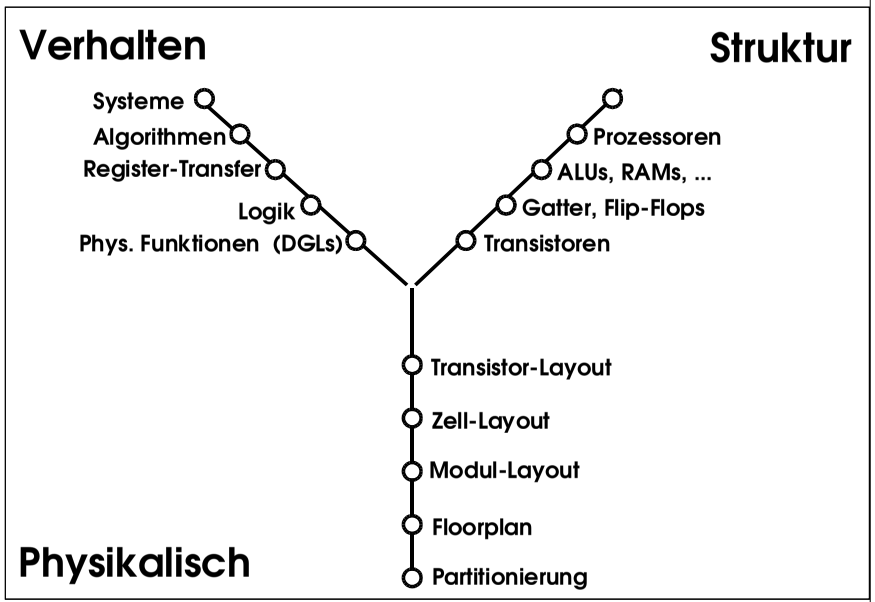
- Fläche (min.)
- Stromverbrauch (min.)
- Geschwindigkeit (max. oder passend)
- Entwurfszeit (min.)
- Testbarkeit (max.)

✝ „Alles auf einmal“ ist zu komplex

➔Aufteilen und vereinfachen

- Qualitätseinbussen

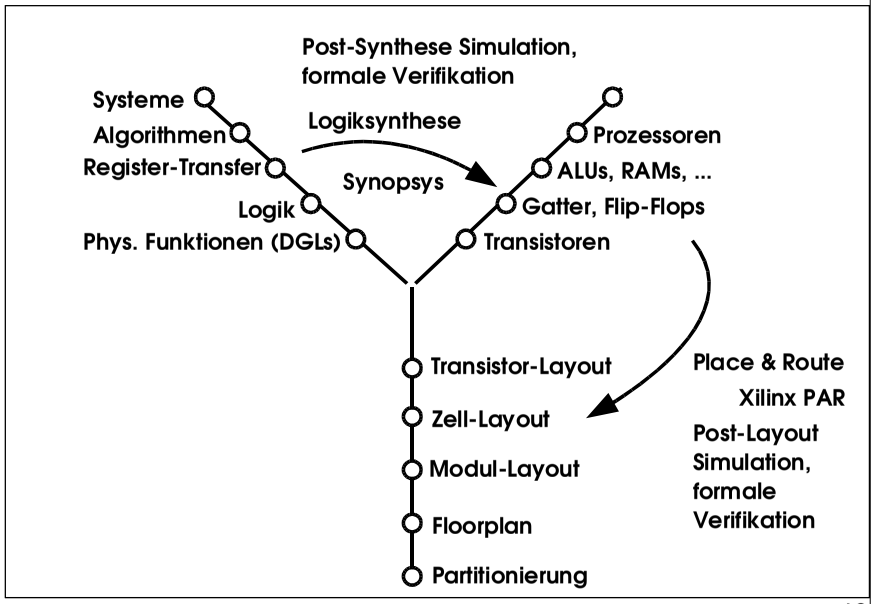
Entwurfsbereiche - Gajskis "Y"



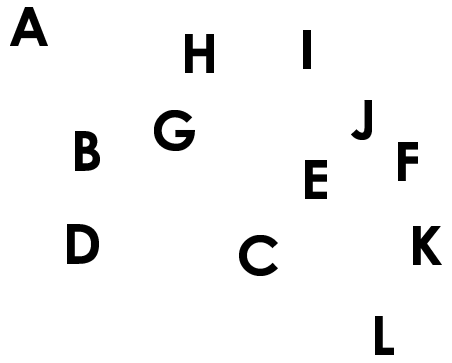
Tätigkeiten

- Synthese
  - Mehr Details durch Anwendung von Regeln
- Verifikation
  - Vergleiche Ergebnis mit Spezifikation
- Analyse
  - Untersuche Eigenschaften eines Ergebnisses
- Optimierung
  - Verbessere ein Ergebnis
- Datenverwaltung

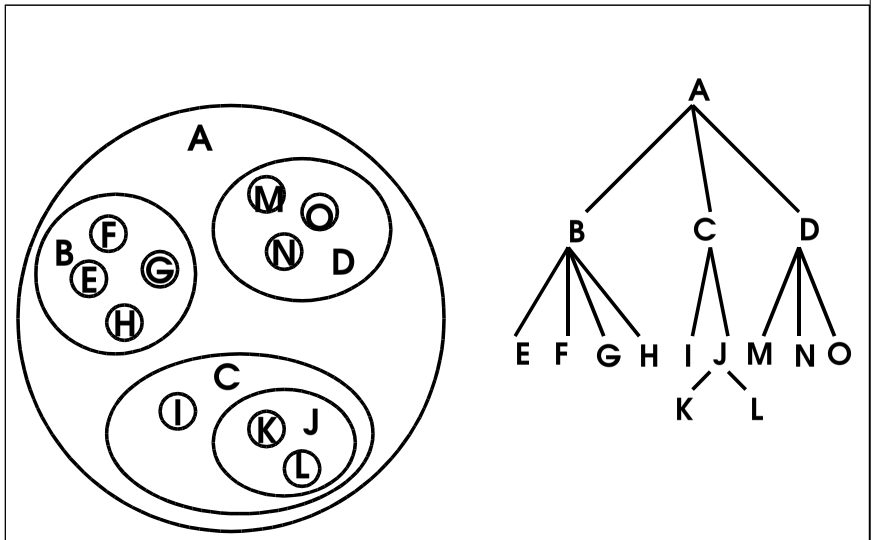
Werkzeuge



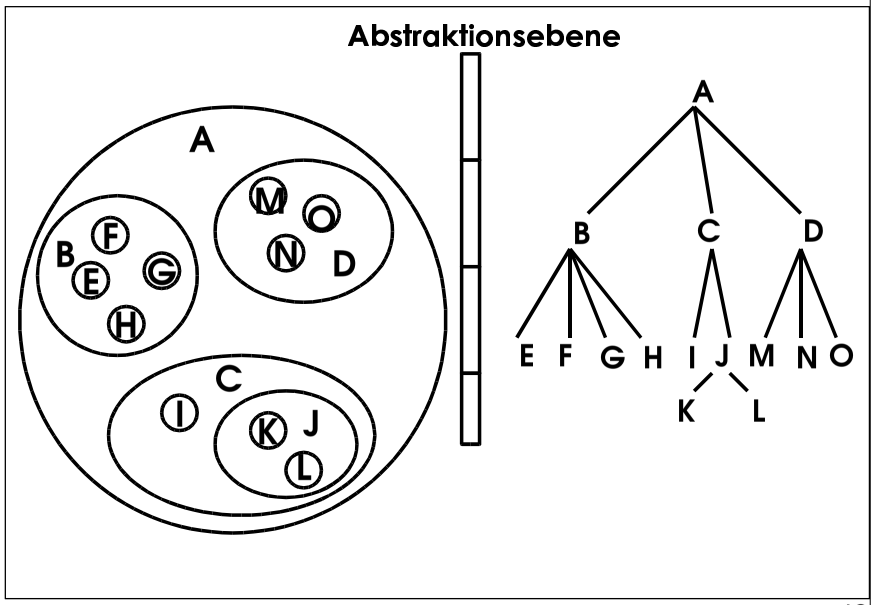
Strukturierung



Hierarchie

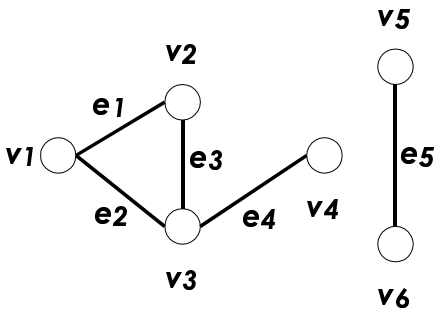


Abstraktion



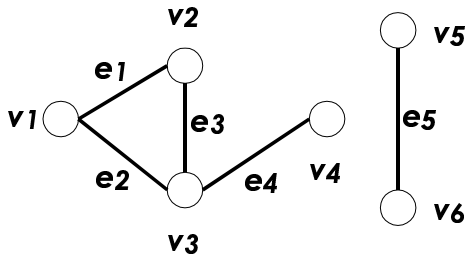
Graphentheorie

- Graph *G (V, E)*
  - Eine Menge *V* von Knoten (vertex)
  - Eine Menge *E* von Kanten (edge)
    - Kante $e = \{v_1, v_2\}$ verbindet Knoten v_1 und v_2



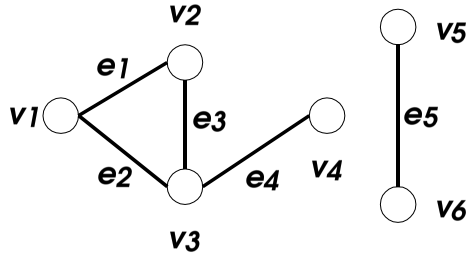
$G = (V, E)$
$V = \{v_1, v_2, v_3, v_4, v_5, v_6\}$
$E = \{e_1, e_2, e_3, e_4, e_5\}$
mit
$e_1 = \{v_1, v_2\}$
$e_2 = \{v_1, v_3\}$, ...

Inzidenz, Adjazenz und Grad



- $e = \{u, v\} \in E$
  - *e* ist inzident *u* — *incident*
  - *e* ist inzident *v* — *incident*
  - *u* ist adjazent *v* — *adjacent*
- Grad $g(v) = |\{e \in E \mid v \in e\}|$ — *degree*

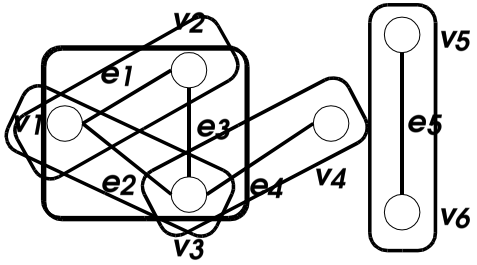
Subgraphen



- Subgraph durch Entfernen von Knoten
- Entferne $v \in V$

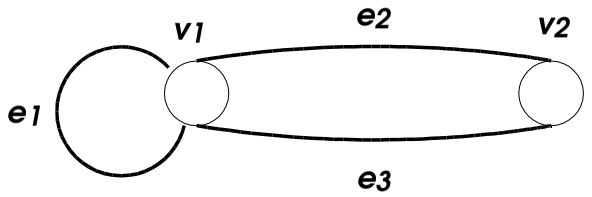
➔Entferne Kanten inzident zu *v*

Vollständigkeit und Cliquen



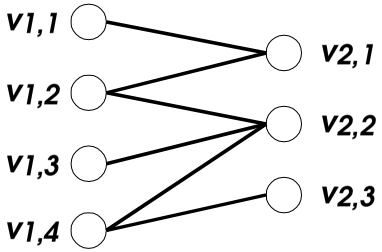
- Komplett untereinander verbundene Knoten bilden vollständigen Graph (complete graph)
- Maximal ausgedehnte vollständige Graphen bilden Cliquen

Schlingen, parallele Kanten



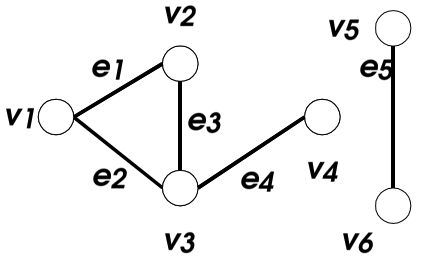
- e_1 Schlinge (selfloop)
- e_2, e_3 parallele Kanten
- einfache Graphen: weder noch (simple)
- Multigraphen: parallele Kanten OK

Bipartite Graphen



- Kanten nur zwischen Knoten aus nichtüberlappenden Mengen
- $G = (V_1, V_2, E)$ ist bipartiter Graph
  - $V1 \cap V2 = \emptyset$
  - $E = \{\{u, w\} \mid u \in V_1 \wedge w \in V_2\}$

Wege und Zyklen

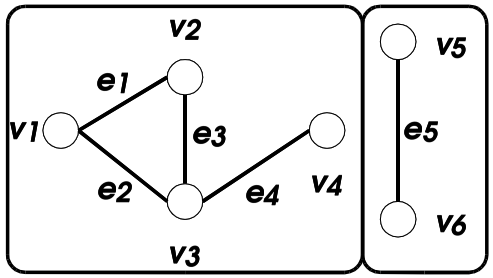


$(v_1, e_1, v_2, e_3, v_3, e_4, v_4)$

Länge = 3

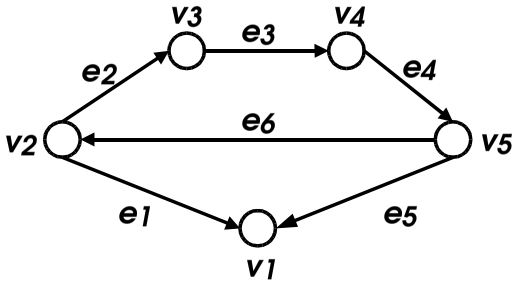
- Weg: Folge von Knoten und Kanten
  - Beginnend und endend mit Knoten
- Länge: Anzahl der Kanten
- Zyklus: Anfang = Ende

Zusammenhang



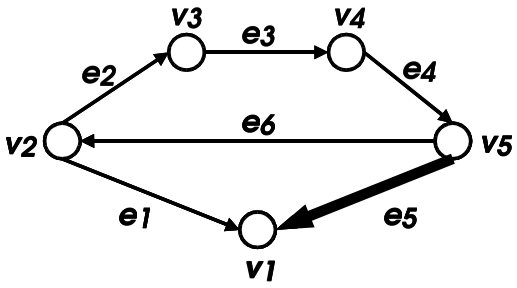
- u hängt mit v zusammen
  - Es gibt einen beide verbindenden Weg
- Zusammenhängender Graph
  - Alle Knoten hängen zusammen.
- Zusammenhängende Komponente
  - Maximale zusammenhängende Subgraphen

Gerichtete Graphen



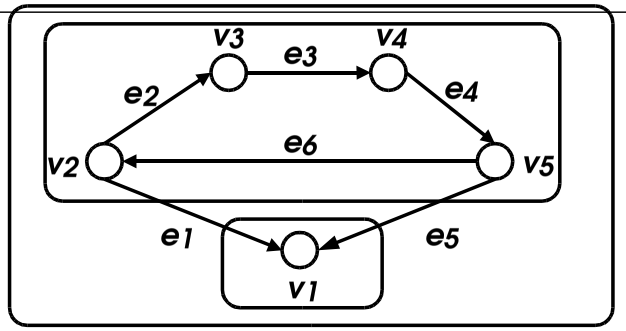
- $G(V, E)$ mit $e = (u, v) \wedge u,v \in E$
  - e inzident von *u* (ausgehend)
  - e inzident nach *v* (eingehend)
- Außengrad: Anzahl ausgehender Kanten
- Innengrad: Anzahl eingehender Kanten

Wege und Zyklen



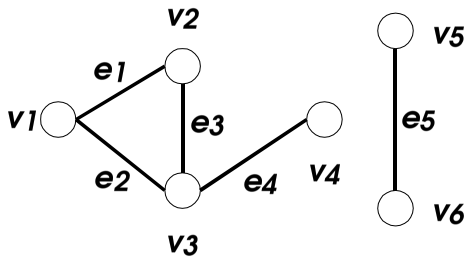
- Gerichteter Weg
- Gerichteter Zyklus
- Weg und Zyklus gelten auch noch!

Zusammenhang



- Starker Zusammenhang
  - Gerichteter Weg von *u* nach *v* & von *v* nach *u*
- Stark zusammenhängende Komponente
  - Alle enthaltenen Knoten hängen stark zusam.
- Schwacher Zusammenhang: Weg

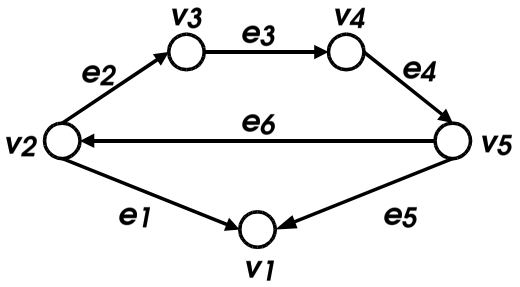
Datenstrukturen für Graphen



0	1	1	0	0	0
1	0	1	0	0	0
1	1	0	1	0	0
0	0	1	0	0	0
0	0	0	0	0	1
0	0	0	0	1	0

- Adjazenzmatrix A_G von G(V,E)
  - n x n Matrix mit $n = |V|$
  - $A_{ij} = 1$ falls $\{v_i,v_j\} \in E$, sonst = 0
  - Symmetrische Matrix

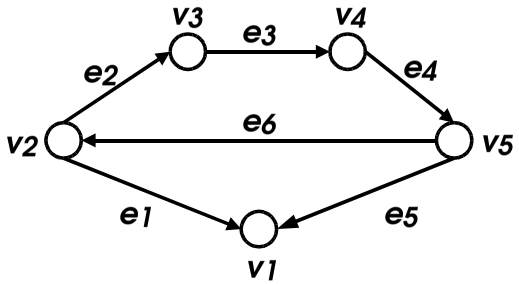
A_G für gerichtete Graphen



0	0	0	0	0	
1	0	1	0	0	
0	0	0	1	0	
0	0	0	0	1	
1	1	0	0	0	

- Matrix nicht mehr symmetrisch

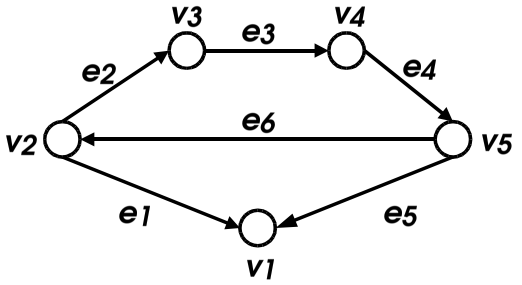
Operationen auf A_G-Matrizen

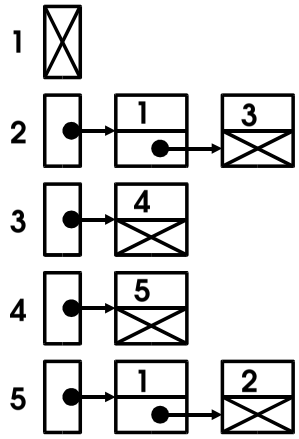


0	0	0	0	0	
1	0	1	0	0	
0	0	0	1	0	
0	0	0	0	1	
1	1	0	0	0	

- Test, ob $(v_i,v_j) \in E$
  - Nachsehen in A_{ij}: O(1)
- Welche v sind direkt mit u_i verbunden?
  - Zeile i durchgehen: O(n)
  - Ineffizient bei vielen Nullen

Adjazenzlisten

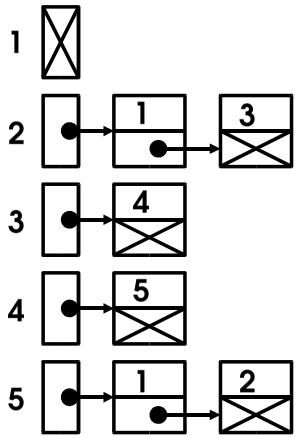




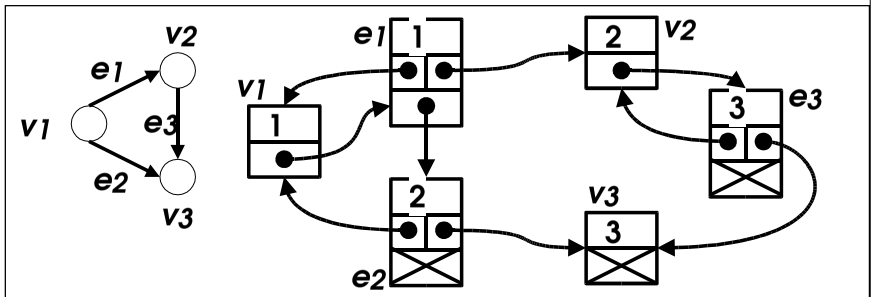
- Array aus Listen
  - Knotennummer ist Index
- Listenelemente
  - Index des Zielknotens
  - Verkettung

Operationen auf Adjazenzlisten

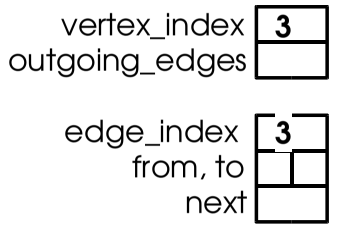
- Test, ob $(v_i,v_j) \in E$
  - durchschnittlicher Außengrad: $k(G)$
  - $O(k)$
  - Unabhängig von n
- Welche v sind direkt mit u_i verbunden?
  - $O(k)$



Explizite Knoten und Kanten



- Zugriff auf Knoten und Kanten
- Z.B. Gewichtung von
  - Knoten
  - Kanten



Komplexitätstheorie

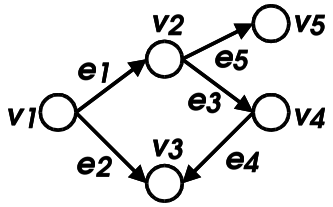
- O und Θ Notation
- Siehe Grundstudium!
- Wichtige Ordnungen
  - Exponentiell, z.B. 2^n.
  - Polynomial, z.B. n^3.
  - Quadratisch, z.B. n^2.
  - Logarithmisch, z.B. $n \log n$.
  - Linear, z.B. n.
  - Sublinear, z.B. 1.

Graphen durchlaufen

- Aufgabe
  - Besuche alle V und E von G(V,E)
  - Jedes Element genau einmal!
- Unterschiedliche Reihenfolgen möglich
- Weit verbreitet
  - Tiefensuche
    - Suche von Ursprungsknoten entfernen
  - Breitensuche
    - Erstmal angrenzende Knoten bearbeiten

Tiefensuche (DFS) - 1

```
dfs(vertex v) {
  v.mark := 0;
  v.process();
  foreach (v,u) ∈ E {
    (v,u).process();
    if (u.mark) dfs(u);
  }
}
main() {
  foreach v ∈ V
    v.mark := 1;
  foreach v ∈ V
    if (v.mark) dfs(v)
}
```



dfs(***v1***)
  (v1, v2)
    dfs(***v2***)
      (v2, v4)
        dfs(***v4***)
          (v4, v3)
            dfs(***v3***)
      (v2, v5)
        dfs(***v5***)
  (v1, v3)

Tiefensuche (DFS) - 2

- **Komplexität für DFS auf *G(V,E)***
  - **Jeder Knoten einmal besucht**
  - **Jede Kante einmal besucht**
  - **→*O(\|V\|+\|E\|)***
- **Anwendungsbeispiele**
  - **Systematischer Graphdurchlauf**
  - **Finden der von einem Startknoten aus erreichbaren Knoten**
    - **Ersetze Schleife in main() durch einfachen Aufruf**

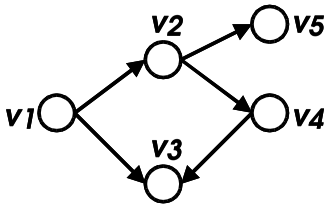
Breitensuche (BFS) - 1

```
bfs(vertex v) {
  FIFO Q = ();
  vertex u, w;

  Q.shift_in(v);
  do {
    w := Q.shift_out();
    w.process();
    foreach (w,u) ∈ E do {
      if (u.mark) {
        u.mark := 0;
        Q.shift_in(u);
      }
    }
  } while (Q ≠ ())
}
```

```
main() {
  foreach v ∈ V do v.mark := 1;
  foreach v ∈ V do
    if (v.mark) {
      v.mark := 0;
      bfs(v);
    }
}
```


Breitensuche (BFS) - 2

```
bfs(vertex v) {
  FIFO Q = ();
  vertex u, w;

  Q.shift_in(v);
  do {
    w := Q.shift_out();
    w.process();
    foreach (w,u) ∈ E do {
      if (u.mark) {
        u.mark := 0;
        Q.shift_in(u);
      }
    }
  } while (Q ≠ ())
}
```



(v1)	v1	(v1,v2)
(v2)		(v1,v3)
(v2,v3)	v2	(v2,v4)
(v3,v4)		(v2,v5)
(v3,v4,v5)	v3	
(v4,v5)	v4	(v4,v3)
(v5)	v5	

Breitensuche (BFS) - 3

- **Komplexität für BFS auf *G(V,E)***
  - Jeder Knoten einmal besucht
  - Jede Kante einmal besucht

  ➔$O(|V|+|E|)$
- **Anwendungsbeispiele**
  - Systematischer Graphdurchlauf
  - Finden der von einem Startknoten aus erreichbaren Knoten
  - Besuche Knoten in Reihenfolge der Entfernung (Pfadlänge) vom Startknoten

DFS und BFS

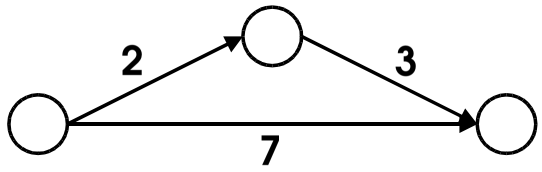
- **Weshalb die die äußeren Schleifen?**
  - Jeweils in main()
  - Um dfs(v) bzw. bfs(v)

```
main() {
  foreach v ∈ V
    v.mark := 1;
  foreach v ∈ V
    if (v.mark) dfs(v)
}
```

```
main() {
  foreach v ∈ V do v.mark := 1;
  foreach v ∈ V do
    if (v.mark) {
      v.mark := 0;
      bfs(v);
    }
}
```

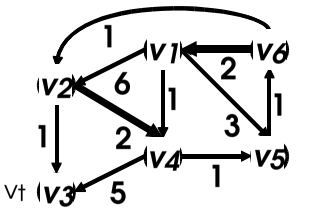

Kürzester Pfad

- **Bestimme den kürzesten Pfad vom Startknoten zu den anderen Knoten**
- **Bei ungewichteten Graphen z.B. mit BFS**
  - Erweitert um Verwaltung der Pfade
- **✖Nicht bei gewichteten Graphen!**
  - Niedrige Anzahl von Kanten nicht immer kürzester (leichtester) Weg



Kürzester Pfad nach Dijkstra -1

```
dijkstra(set<vertex> V, vertex vs, vertex vt) {
  set<vertex> T; vertex u, v;
  V    := V \ {vs}; T := {vs};
  vs.dist := 0;
  foreach u ∈ V do
    if ((vs, u) ∈ E)
      then u.dist := (vs, u).weight;
      else u.dist := +∞;
  while (vt ∉ T) do {
    u   := V.findmin(dist);
    T   := T ∪ {u};
    V:= V \ {u};
    foreach (u, v) ∈ E do
      if (v.dist > u.dist + (u,v).weight)
        v.dist := u.dist + (u,v).weight;
  }
}
```



T=	v_i.dist, i=	1	2	3	4	5	6
{v1}		0	6	∞	1	3	∞
{v1,v4}			6	6		2	∞
{v1,v4,v5}			6	6			3
{v1,v4,v5,v6}			4	6			
{v1,v4,v5,v6,v2}				5			
{v1,v4,v5,v6,v2,v3}							

Kürzester Pfad nach Dijkstra -2

- Komplexität
  - while (v $\notin$ T): $|V|$-mal durchlaufen
    - V.findmin(dist): $O(|V|)$ je Suche

    ➥ $O(|V|^2)$
  - foreach (u,v) $\in$ E: $|E|$-mal *insgesamt*
    - Einfacher Graph hat max. $|V|^2$ Kanten

    ➥ $O(|V|^2)$
  - Gesamtaufwand $O(|V|^2+|V|^2) = O(|V|^2)$

Nächste Veranstaltung

- Vorlesung am Freitag
- Vorbereitungstipps
  - Kapitel 6 und 7.1 lesen
  - Ggf. Kapitel 4 (Komplexität) wiederholen

Zusammenfassung

- VLSI
  - Entwurfsbereiche
  - Tätigkeiten
  - Werkzeuge
- Hierarchie und Abstraktion
- Graphentheorie
  - Konzepte und Begriffe
  - Datenstrukturen
  - Algorithmen: DFS, BFS, SP