Graphentheoretische Konzepte und Algorithmen

Thema 02
Suchstrategien und kürzeste Wege

Julia Padberg

Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

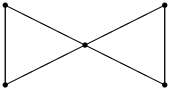
Schnitte

Definition (Schnittknoten und Schnittkanten)

In einem zusammenhängenden Graphen $G = (V, E)$ heißt ein Knoten v ein **Schnittknoten**, falls der Untergraph H von G mit Knotenmenge $V \setminus \{v\}$ nicht zusammenhängend ist. Entsprechend heißt eine Kante e eine **Schnittkante**, falls der Teilgraph $F(V, E \setminus \{e\})$ von G nicht zusammenhängend ist.

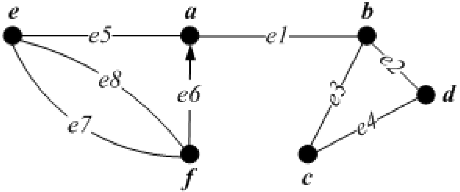
BSP:

Dieser Graph besitzt einen Schnittknoten, aber keine Schnittkante:



Aufgabe 1:

Geben Sie bitte Schnittkanten und Schnittknoten dieses Graphes an:



Lösung

Schnittknoten: *a* und *b*
Schnittkanten: *e*1

Wahr oder Falsch

für schlichte Graphen

1. Eine Kante zwischen zwei Schnittknoten ist eine Schnittkante.
   ☐ wahr oder ☒ falsch
2. Wenn ein zusammenhängender Graph einen Kreis enthält, dann gibt es keinen Schnittknoten ☐ wahr oder ☒ falsch
3. Wenn es nur einen Weg von den Knoten v zu dem Knoten w gibt, dann gibt es auf diesem Weg mindestens eine Schnittkante.
   ☒ wahr oder ☐ falsch
4. Wenn es einen Schnittknoten gibt, gibt es auch eine Schnittkante.
   ☐ wahr oder ☒ falsch
5. Wenn es eine Schnittkante gibt, gibt es genau einen Schnittknoten. ☐ wahr oder ☒ falsch
6. Wenn es eine Schnittkante gibt, gibt es mindestens zwei Schnittknoten. ☐ wahr oder ☒ falsch

Kantenlänge

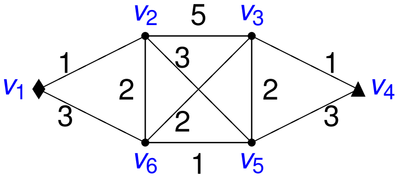
Definition

Jeder Kante eine reelle Zahl zuordnen, die wir als **Länge** dieser Kante bezeichnen. Man spricht dann von einem Graphen mit bewerteten Kanten.

BSP:

Sechs Städte werden durch das folgende Eisenbahnnetz miteinander verbunden. Die Länge einer Kante gibt dabei die durchschnittlich zu erwartende Fahrzeit in Stunden an.

Für einen Reisenden, der von ♦-Stadt nach ▲-City unterwegs ist, stellt sich die Frage, auf welchem Weg er am schnellsten zum Ziel kommt.



Der Algorithmus von Dijkstra

- dient der Bestimmung kürzester Wege
- von einem fest vorgegebenen Knoten zu allen anderen Knoten
- in einem schlichten zusammenhängenden gerichteten Graphen
- mit endlicher Knotenmenge und nichtnegativ bewerteten Kanten
- und liefert einen Weg mit einer minimalen Gesamtlänge
- Bei der Anwendung dieses Verfahrens auf ungerichtete Graphen muß jede Kante durch ein Paar entgegengesetzt gerichteter Kanten ersetzt werden.

Der Algorithmus von Dijkstra

Vorbereitungsphase

Algorithmus

Es bezeichne l_{ij} die Länge der Kante $v_i v_j$. Falls es keine Kante $v_i v_j$ gibt, sei $l_{ij} := \infty$ Für jeden Knoten $v_i \in V$ des zu untersuchenden Graphen werden drei Variable angelegt:

1. $Entf_i$ gibt die bisher festgestellte kürzeste Entfernung von v_1 nach v_i an. Der Startwert ist 0 für $i = 1$ und ∞ sonst.
2. $Vorg_i$ gibt den Vorgänger von v_i auf dem bisher kürzesten Weg von v_1 nach v_i an. Der Startwert ist v_1 für $i = 1$ und undefiniert sonst.
3. OK_i gibt an, ob die kürzeste Entfernung von v_1 nach v_i schon bekannt ist. Der Startwert für alle Werte von i ist *false*.

Der Algorithmus von Dijkstra

Iterationsphase

Algorithmus

Wiederhole

- Suche unter den Knoten v_i mit $OK_i = false$ einen Knoten v_h mit dem kleinsten Wert von $Entf_i$.
- Setze $OK_h := true$. (Wieso?)
- Für alle Knoten v_j mit $OK_j = false$, für die die Kante $v_h v_j$ existiert:
  - Falls gilt $Entf_j > Entf_h + l_{hj}$ dann
    - Setze $Entf_j := Entf_h + l_{hj}$
    - Setze $Vorg_j := v_h$

solange es noch Knoten v_i mit $OK_i = false$ gibt.

Beispiel – siehe Folie 122

Nach der Vorbereitungsphase haben *Entf*, *Vorg* und *OK* die Werte:

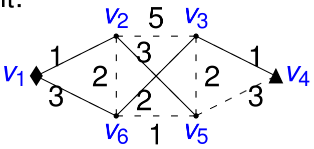
i	1	2	3	4	5	6
Entf	0	∞	∞	∞	∞	∞
Vorg	1					
OK	*f*	*f*	*f*	*f*	*f*	*f*

In den Iterationsschritten ist v_h nacheinander der Knoten v_1, v_2, v_6, v_5, v_3, v_4, und am Ende haben *Entf*, *Vorg* und *OK* die Werte:

i	1	2	3	4	5	6
Entf	0	1	5	6	4	3
Vorg	1	1	6	3	2	1
OK	*t*	*t*	*t*	*t*	*t*	*t*

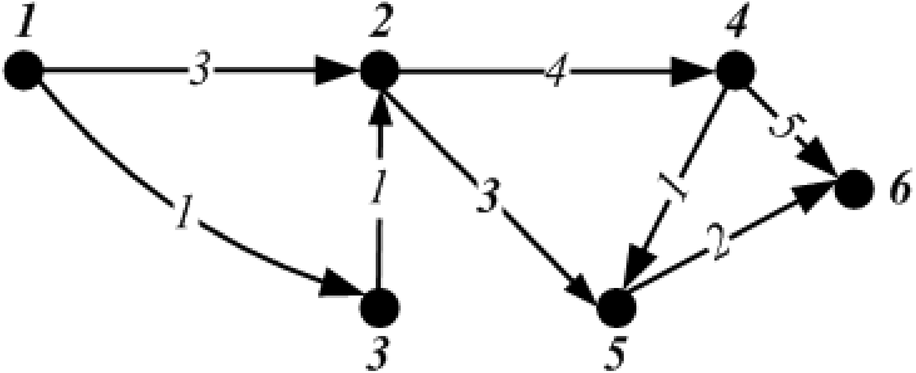
Der Knoten v_4 hat also von v_1 den Abstand 6, und auf dem Weg von v_1 nach v_4 hat den vorletzten Knoten den Index $Vorg_4 = 3$ sowie den drittletzten Knoten den Index $Vorg_3 = 6$ und wegen $Vorg_6 = 1$ lautet dann der komplette Weg: $v_1,\ v_6,\ v_3,\ v_4$

In der folgenden Darstellung sind die nicht zu den kürzesten Wegen gehörenden Kanten gestrichelt dargestellt:

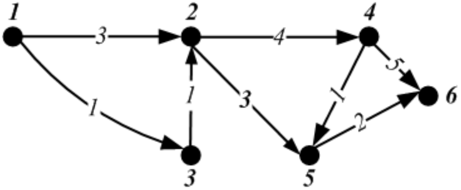


Aufgabe 2:

Führen Sie bitte für diesen Graphen den Dijkstra-Algorithmus durch:



Lösung von Aufgabe 2



	1	2	3	4	5	6
Entf	0	~~3~~ 2	1	6	5	7
Vorg	1	~~1~~ 3	1	2	2	5
OK	*t*	*t*	*t*	*t*	*t*	*t*

Aufwandsabschätzung von Algorithmen

- Zeit bzw. Speicheraufwand eines Algorithmus, abhängig von einer bestimmten Eingabegröße n
- bestimmte atomare Operationen innerhalb der Sprache ein konstanten, von n unabhängige Aufwand zugeordnet
- die tatsächliche Laufzeit bzw. der tatsächliche Speicheraufwand ist nicht relevant.
- Ziel einer Aufwandsabschätzung ist es daher herauszufinden, wie sich der Aufwand auf einer idealisierten (Turing-)Maschine verändert, wenn man die Eingabegröße verändert.
- Komplexitätsbetrachtungen ermöglichen den Vergleich der prinzipiellen Eigenschaften von Algorithmen
  - Z.B. das Zählen der Aufrufe einer Schlüsseloperation. Beispiel Sortieren:
    - Anzahl der Vergleiche
    - Anzahl der Vertauschungen

Größenordnungsmäßige Schreibweise

Definition

Eine reelle Funktion $f(x)$ heißt *von der Größenordnung* $g(x)$ (geschrieben: $f(x) = \mathcal{O}(g(x))$), wenn $g(x)$ das Verhalten von $f(x)$ für hinreichend große x im wesentlichen bestimmt – präziser:

$$f(x) = \mathcal{O}(g(x)) \Longrightarrow \exists c \in \mathbb{R} : \lim_{x\to\infty} \frac{f(x)}{g(x)} = c$$

BSP: Beispiele

$$\begin{aligned} \frac{6}{5}x^2 - 5x &= \mathcal{O}(x^2) \\ -x^5 + \ln x &= \mathcal{O}(x^5) \\ 2^x - 5x^3 &= \mathcal{O}(2^x) \end{aligned}$$

O-Kalkül

Landau-Symbole http://alda.iwr.uni-heidelberg.de/index.php/Effizienz#Algorithmen-Komplexit.C3.A4t

- Als Maß für das Wachstum wird häufig das $\mathcal{O}$-Kalkül verwendet.
- Das wichtigste Landau-Symbol ist $\mathcal{O}$, mit dem man eine obere Schranke $f \in \mathcal{O}(g)$ für die Komplexität angeben kann.
- Die Wachstumsrate wird dabei in der Form $\mathcal{O}(f(n))$ angegeben, wobei die Funktion $f(n)$ eine Obergrenze für das Wachstum des Algorithmus ist.
- Rechenregeln:
  1. $f(x) \in \mathcal{O}(f(x))$
  2. $\mathcal{O}(\mathcal{O}(f(x))) \in \mathcal{O}(f(x))$
  3. $c \cdot \mathcal{O}(f(x)) \in \mathcal{O}(f(x))$ für jede Konstante c
  4. $\mathcal{O}(f(x)) + c \in \mathcal{O}(f(x))$ für jede Konstante c
  5. Sequenzregel nacheinander ausgeführter Programmteile mit $\mathcal{O}(f(x))$ bzw. $\mathcal{O}(g(x))$:
     $\mathcal{O}(f(x)) + \mathcal{O}(g(x)) \in \mathcal{O}(max(f(x), g(x)))$
  6. Schachtelungsregel für geschachtelte Schleifen:
     $\mathcal{O}(f(x)) \cdot \mathcal{O}(g(x)) \in \mathcal{O}(f(x) \cdot g(x))$

Zusammenhang zwischen Komplexität und Laufzeit

- Die Dauer einer Operation sei $1\,ms$,
- dann erreichen Algorithmen verschiedener Komplexität etwa folgende Leistungen:

Komplexität	Operationen		
	in 1s	in 1min	in 1h
$\mathcal{O}(N)$	1000	60.000	3.600.000
$\mathcal{O}(N \log_2 N)$	140	4895	204094
$\mathcal{O}(N^2)$	32	245	1898
$\mathcal{O}(N^3)$	10	39	153
$\mathcal{O}(2^N)$	10	16	21

Notation	Bedeutung	Anschauliche Erklärung	Beispiele für Laufzeiten
$f \in \mathcal{O}(1)$	f ist beschränkt	f überschreitet einen konstanten Wert nicht (unabhängig vom Wert des Arguments).	Nachschlagen des i-ten Elementes in einem Array.
$f \in \mathcal{O}(\log x)$	f logarithmisches Wachstum	f wächst ungefähr um einen konstanten Betrag, wenn sich das Argument verdoppelt. Merke: Die Basis des Logarithmus ist egal.	Binäre Suche im sortierten Feld mit x Einträgen
$f \in \mathcal{O}(\sqrt{x})$	f Wachstum wie die Wurzelfunktion	f wächst ungefähr auf das Doppelte, wenn sich das Argument vervierfacht	
$f \in \mathcal{O}(x)$	f lineares Wachstum	f wächst ungefähr auf das Doppelte, wenn sich das Argument verdoppelt.	Suche im unsortierten Feld mit x Einträgen
$f \in \mathcal{O}(x \log x)$	f super-lineares Wachstum		Fortgeschrittenere Algorithmen zum Sortieren von x Zahlen Mergesort, Heapsort
$f \in \mathcal{O}(x^2)$	f quadratisches Wachstum	f wächst ungefähr auf das Vierfache, wenn sich das Argument verdoppelt	Einfache Algorithmen zum Sortieren von x Zahlen Bubblesort
$f \in \mathcal{O}(2^x)$	f exponentielles Wachstum	f wächst ungefähr auf das Doppelte, wenn sich das Argument um eins erhöht	Rekursive Variante der Fibonacci-Folge
$f \in \mathcal{O}(x!)$	f faktorielles Wachstum	f wächst ungefähr um das $(x+1)$-fache, wenn sich das Argument um eins erhöht.	TSP (Mit Enumerationsansatz)

Komplexität des Dijkstra-Algorithmus

Für den Dijkstra-Algorithmus gilt:
Eingabeumfang und Arbeitsaufwand haben größenordnungsmäßig die Werte

$$\begin{array}{rcl} \text{Umfang der Eingabedaten} & : & \mathcal{O}(|V|^2) \\ \text{Anzahl der Arbeitsschritte} & : & \mathcal{O}(|V|^2) \\ \text{Komplexität in Abhängigkeit vom Graphen} & : & \mathcal{O}(|V|^2) \end{array}$$

Aufgabe 3: Warum?

Komplexität des Dijkstra-Algorithmus

Lösung

Umfang der Eingabedaten

- $|V| \cdot (|V| - 1)$ Kantenlängen, also in $\mathcal{O}(|V|^2)$
  (falls der Graph ungerichtet ist, nur halb so viele).

Anzahl der Arbeitsschritte

- zur Vorbereitung müssen $3 \cdot |V|$ Variable initialisiert werden;
- bei der Iteration muß $(|V| - 1)$-mal wiederholt werden:
  suche eine von höchstens $|V| - 1$ Knoten und
  inspiziere ihre höchstens $|V| - 1$ Nachbarn.
- zusammen höchstens $3 \cdot |V| + 2 \cdot (|V| - 1)^2$, also in $\mathcal{O}(|V|^2)$

Komplexität: $\mathcal{O}(|V|^2) + \mathcal{O}(|V|^2) = \mathcal{O}(|V|^2)$

Komplexität des Dijkstra-Algorithmus

Bemerkung

Unsere Abschätzung hier ist sehr grob und hat weder die Anzahl der Kanten noch geeignete Datenstrukturen zur Speicherung der noch nicht besuchten Knoten berücksichtigt.

Aus http://de.wikipedia.org/wiki/Dijkstra-Algorithmus

Sei m die Anzahl der Kanten und n die Anzahl der Knoten.

- Verwaltung der Knoten mit einer Liste: $\mathcal{O}(n^2 + m)$
- Verwaltung der Knoten mit einer Vorrangwarteschlange:
  $$\mathcal{O}(n \cdot (1 + T_{insert} + T_{empty} + T_{extractMin}) + m \cdot (1 + T_{decreaseKey}))$$
- Verwaltung der Knoten mit einem Fibonacci-Heap: $\mathcal{O}(n \cdot \log n + m)$

A*-Algorithmus

- ein informierter Suchalgorithmus
  eine Schätzfunktion (Heuristik)hilft zielgerichtet zu suchen und damit die Laufzeit zu verringern
- zur Berechnung eines kürzesten Pfades zwischen zwei Knoten in einem Graphen mit positiven Kantengewichten
- 1968 von Peter Hart, Nils J. Nilsson und Bertram Raphael beschrieben
- ist optimal
  es wird immer die optimale Lösung gefunden, falls eine existiert.

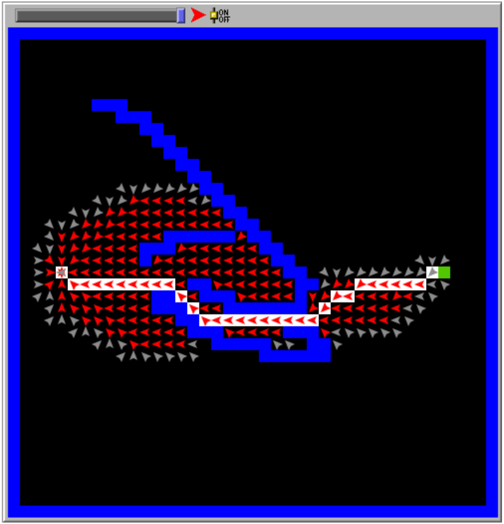
Grundidee

Die Knoten werden während der Suche in drei verschiedene Klassen eingeteilt:

- unbekannte Knoten:
  noch nicht gefunden, noch kein Weg bekannt.
  Jeder Knoten (außer dem Startknoten) ist zu Beginn des Algorithmus unbekannt.
- bekannte Knoten:
  bekannte (suboptimale) Wege mit ihrem f-Wert in der Open List
  Wahl des vielversprechendsten Knoten
  anfangs nur der Startknoten
- abschließend untersuchte Knoten:
  kürzeste Wege in der Closed List
  anfangs leer.

Beispiel

http://www.geosimulation.de/umsetzungen/2_0_2_Modelle/A-Star.html



A*-Algorithmus: Initialisierung

Algorithmus

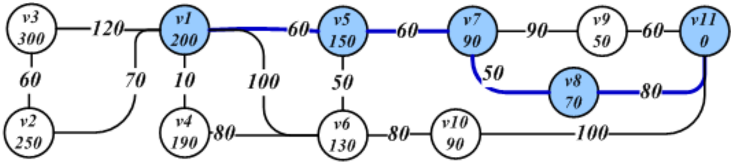
Knoten $V = \{v_1, ..., v_n\}$
offenen Liste: OL nur der Startknoten v_1 mit $f_1 = h_1$, $g_1 = 0$ und $p_1 = v_1$
geschlossene Liste: CL leer
heuristische Knotenwerte : $h_i \in \mathbb{R}^+$
Kantenlängen zwischen v_i und v_j: $l_{ij} \in \mathbb{R}^+$
Vorgänger: $p_i \in V$
aktueller Schätzwert $f_i = \infty$ für $i > 1$
zurückgelegter Weg $g_i = \infty$ für $i > 1$

A*-Algorithmus

Algorithmus

1. Knoten v_k mit niedrigsten f_k in OL suchen
2. Knoten v_k in die CL schieben.
3. Für die adjazente Knoten v_j, die nicht in der CL sind: wird geprüft, ob $g_j > g_k + l_{k,j}$. Falls
   - JA, wird der aktuelle Knoten v_k Vorgängerknoten: $p_j := v_k$ und neuer g- und der f-Wert: $g_j := g_k + l_{k,j}$ und $f_j := g_j + h_j$
4. Falls der Zielknoten in der geschlossenen Liste; gehe zu 5. Falls kein Zielknoten gefunden und offene Liste leer; gehe zu 6. Sonst; gehe zu 1.
5. Der Pfad läßt sich vom Zielknoten aus mittels der Vorgänger bis zum Startknoten zurück verfolgen.
6. Es gibt keinen Pfad.

Beispiel A*



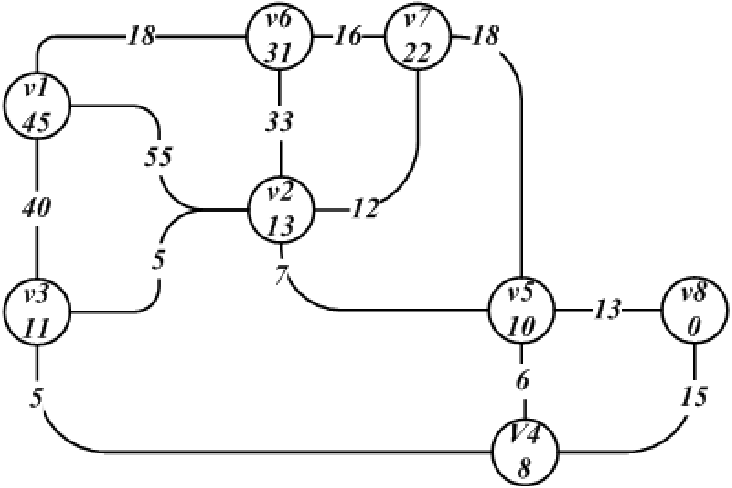
	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8	v_9	v_{10}	v_{11}
Vorg	v_1	v_1	v_1	v_1	v_1	$\cancel{v_1}$ v_4	v_5	v_7	v_7	v_6	v_8
h	200	250	300	190	150	130	90	70	50	90	0
g	0	70	120	10	60	~~100~~ 90	120	170	210	170	250
f	200	320	420	200	210	~~230~~ 220	210	240	260	260	250
CL	*t*			*t*	*t*	*t*	*t*	*t*			*t*

Länge von v_1 nach v_{11}: 250

Weg $v_1 - v_5 - v_7 - v_8 - v_{11}$

Aufgabe 4:

Bitte berechnen Sie mit Hilfe des A*-Algorithmus den kürzesten Weg von v_1 nach v_8.



Lösung von Aufgabe 4

	v_1	v_2	v_3	v_4	v_5	v_6	v_7	v_8
Vorg	v_1	$\cancel{v_1}$ $\cancel{v_6}$ v_3	v_1	v_3	v_4	v_1	v_6	v_4
h	45	13	11	8	10	31	22	0
g	0	~~55~~ ~~51~~ 45	40	45	51	18	34	60
f	45	~~68~~ ~~64~~ 58	51	53	61	49	56	60
CL	t	t	t	t		t	t	t

Länge von v_1 nach v_8: 60

Weg $v_1 - v_3 - v_4 - v_8$

Diskussion

1. Was muss für die Heuristik gelten, damit A* klappt?
   die Heuristik muss monoton sein
   (sie muss diese Bedingungen erfüllen):
   - Die Kosten dürfen nie überschätzt werden.
   - Für jeden Knoten k und jeden Nachfolgeknoten j von k muss gelten $h(k) \leq l_{k,j} + h(j)$.

   Ist eine der beiden Bedingungen verletzt ergibt sich eine nicht optimale Lösung.
2. Ist der A* immer besser als Dijkstra?
   - nicht, wenn die Heuristik sehr komplex ist.
   - nicht, wenn der Zielknoten auch noch gefunden werden muss