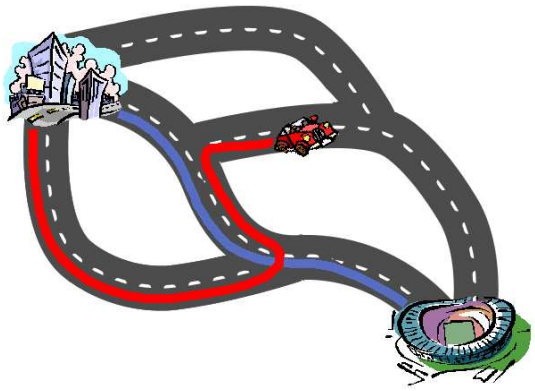
1

Maximale Flüsse – Die ganze Stadt will zum Stadion

Robert Görke, Steffen Mecke und Dorothea Wagner

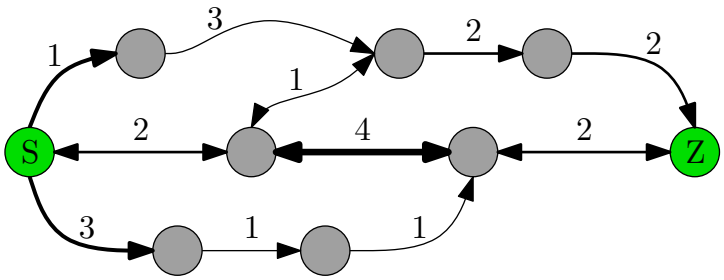
Institut für Theoretische Informatik, Universität Karlsruhe (TH)

„Was ist denn das für ein Mist, so kommen wir ja nie zum Fußballstadion!“ Jogi saß im Auto neben seiner Mutter und wurde langsam unruhig. „Ich kann doch auch nichts dafür, dass alle diese Straße zum Stadion benutzen. Jetzt ist eben Stau“, meinte sie nur. – „Dann dreh’ doch einfach um und wir nehmen da hinten die Fordstraße, die Route benutzt keiner.“ Jogis Mutter glaubte zwar nicht so recht daran, fuhr aber um des lieben Friedens willen zurück und tatsächlich, auf der Fordstraße war weniger Verkehr. Jedenfalls bis zur nächsten Kreuzung. Dort mündete die Fordstraße auf die vielbefahrene, breite Bahnhofstraße und es gab wieder Stau. „Diese Idioten haben doch keine Ahnung! Mama, bieg links ab“ – „Aber zum Stadion geht es doch geradeaus“, meinte die. „Schon“, erwiderte Jogi, „aber da hinten können wir dann die Karlstraße nehmen, das ist zwar ein Umweg, aber dafür ist die Straße bestimmt frei.“ Jogis Mutter war immer noch skeptisch, aber einen Versuch war es wert. Und tatsächlich, den Umweg schien keiner nehmen zu wollen. Jogi versuchte sogar noch, einige der auf der Bahnhofstraße entgegenkommenden Fahrzeuge zum Umdrehen zu bewegen, aber ohne Erfolg. „Sind die blöd, jetzt stehen sie gleich im Stau, dabei könnten sie hier leicht durchkommen!“



Das Fußballspiel stellte sich als übles Herumgeschiebe heraus, so dass es Jogi schließlich so langweilig wurde, dass er nochmal über das heutige Verkehrsproblem nachdachte. „Wenn man die Autofahrer sich selbst überlässt, funktioniert es offensichtlich nicht sehr gut mit der Stauvermeidung. Man müsste an jeder Kreuzung Wegweiser aufstellen, nach denen sich die Autos dann richten müssten, damit möglichst viele Autos zum Ziel kommen und der Verkehr fließen kann. Aber wie finde ich die beste Lösung dieses Problems?“ Er kam zunächst zu keiner zufriedenstellenden Antwort. Erst ein paar Tage später sprach er noch mal mit seiner großen Schwester darüber. Die studierte schon Informatik, wusste aber auf Anhieb auch keine Lösung.

„Machen wir das Problem erst mal möglichst einfach: Nehmen wir an, alle Autofahrer starten vom gleichen Punkt“ – sie markierte diesen Punkt und malte ein großes „S“ für Start darauf – „und wollen zu einem anderen Punkt.“ Diesen markierte sie mit „Z“, für Ziel. „Dazwischen sind die Straßen, die an den Kreuzungen jeweils zusammentreffen.“ Sie malte einige Punkte und Linien zwischen Start- und Zielpunkt.

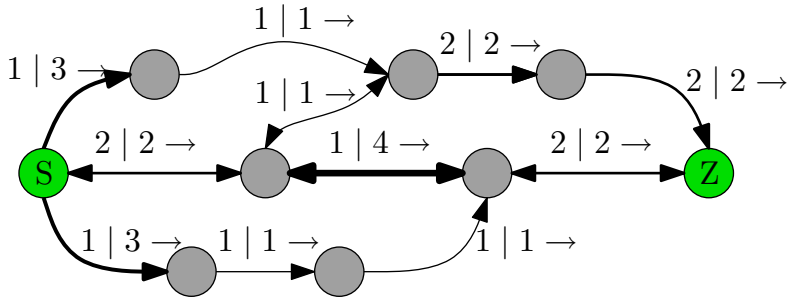


„Die Straßen können aber unterschiedlich breit sein, schreiben wir mal neben jede Straße die Anzahl der Spuren und machen sie in der Zeichnung entsprechend dicker oder dünner. Hmm, wir hatten es neulich in der Vorlesung von kürzesten Wegen, aber das hilft uns hier nicht richtig weiter. Natürlich kann man erst mal den kürzesten Weg von S nach Z suchen“. – „Du meinst diesen hier.“ Jogi zeichnete ihn ein. „Aber jetzt können wir doch weitere Wege suchen. Wir müssen uns nur merken, wie viele Autos jede Straße schon benutzen.“

Wir verlassen Jogi und seine Schwester an dieser Stelle mal und überlegen weiter: Wir haben also ein Straßennetz mit verschiedenen Straßenbreiten und wollen wissen, wie wir die Autos leiten müssen, damit der Verkehr am besten fließen kann. Damit es möglichst einfach bleibt, nehmen wir an, dass alle Autofahrer von S nach Z wollen. Normalerweise versucht man als Autofahrer, den kürzesten Weg zum Ziel zu nehmen. Wenn aber zu viele Autos gleichzeitig auf der selben Straße fahren, gibt es einen Stau. „Gleichzeitig“ heißt in unserem Fall „in der Stunde bevor das Fußballspiel losgeht“. Jede Straße kann nur eine bestimmte Zahl an durchfahrenden Autos verkraften (die *Kapazität* der Straße). Diese hängt weniger von ihrer Länge ab, mehr von ihrer Brei-

te (Anzahl der Spuren). Eine Straße mit einer Spur können pro Stunde zum Beispiel 1000 Autos benutzen. Wir schreiben aber trotzdem eine 1, also die Zahl der Spuren, und keine 1000, da wir nicht mit so großen Zahlen hantieren möchten.

Kritische Punkte im Straßennetz sind auch die Kreuzungen. Wenn hier mehr Autos ankommen, als weiterfahren können, gibt es ebenfalls Stau. Wenn mindestens genauso viele Autos weiterfahren können, wie ankommen, ist aber alles in Ordnung. Wir fragen uns nun, wie viele Autos wir maximal durch das Straßennetz von S nach Z („gleichzeitig“, also pro Stunde) schleusen können. Eine Lösung des Problems für unser Beispiel wäre der folgende „Verkehrsfluss“:



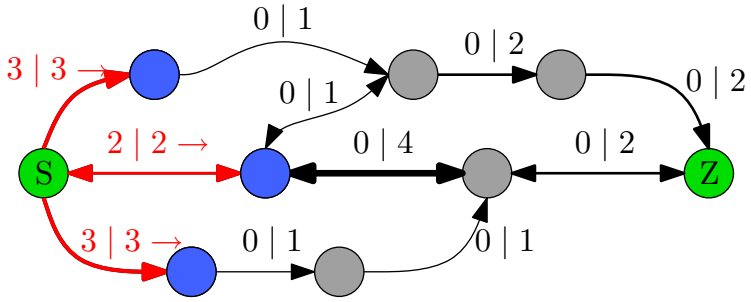
An jeder Straße steht nun zusätzlich die Anzahl der Autos, die diese Straße nehmen und ein Pfeil für die Richtung, in der sie fahren. Also heißt „$1 \mid 3 \rightarrow$“, dass eine Spur von drei vorhandenen benutzt wird und zwar nach rechts. Die erste Zahl darf niemals größer sein als die Zahl der Spuren der Straße (d.h. so etwas wie „$3 \mid 2$“ ist nicht erlaubt). Diese Regel ist so wichtig, dass wir ihr gleich einen Namen geben. Wir nennen sie die *Spurregel* (oder *Kapazitätsregel*). Die Bedingung, dass in jede Kreuzung genau so viele Autos hineinfahren wie herauskommen (sonst gibt es Stau) nennen wir die *Kreuzungsregel* (oder *Flusserhaltungsregel*, da sie dafür sorgt, dass der Verkehr an den Kreuzungen weiterfließen kann). Nur bei Start und Ziel gilt diese Regel nicht. Unter all den Verkehrsflüssen, die diese beiden Regeln erfüllen, suchen wir nun einen, bei dem möglichst viele Autos gleichzeitig am Start losfahren dürfen oder – was dann das Gleiche ist – am Ziel ankommen. Die Informatiker nennen so etwas dann einen *maximalen Fluss*.

Diese Art von Problemen gibt es nicht nur im Verkehrsbereich. Man kann sich so zum Beispiel auch überlegen, wie man möglichst schnell Gebäude evakuieren kann. Oder Daten durch ein Computernetz leiten. Fallen dir noch andere Beispiele ein?

Der Algorithmus

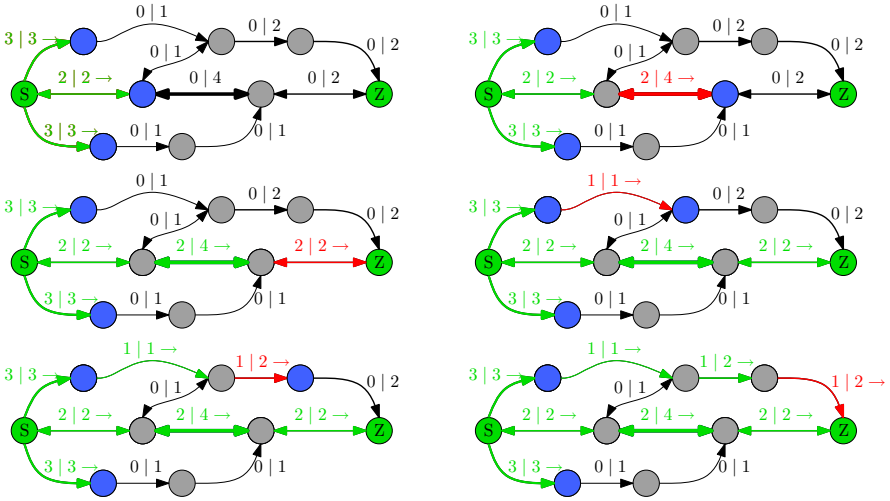
Wie finden wir nun einen maximalen Fluss? Versuchen wir es einfach mal: Am Anfang fahren noch gar keine Autos durch unser Straßennetz. Nun fangen wir

damit an, erst einmal vom Startpunkt aus so viele Autos losfahren zu lassen (rot), wie überhaupt möglich ist, ohne die Spurregel zu verletzen.



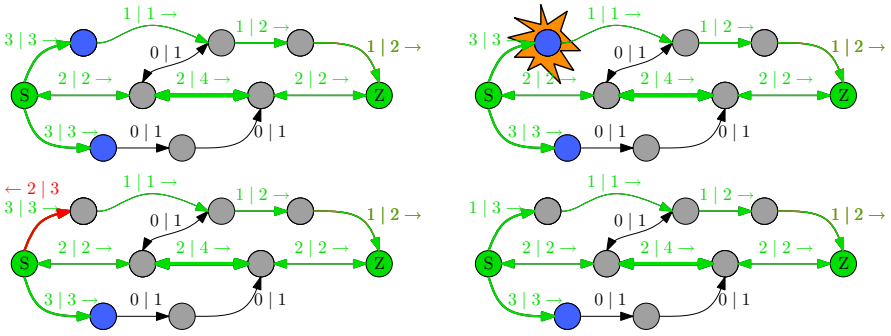
Also fahren (grün) nun auf jeder Straße, die vom Startpunkt los geht, so viele, wie freie Spuren vorhanden sind. Natürlich machen wir das alles erst einmal nicht mit echten Autos, sondern zum Beispiel mit Modellautos. Oder einfach mit Bleistift und Papier. Erst wenn wir die beste Lösung für unser Problem gefunden haben, können wir damit anfangen, den Verkehr auf den echten Straßen mit echten Autos zu regeln.

Jetzt haben natürlich die Kreuzungen an den Endpunkten dieser Straßen einen „Überschuss“ (blau) an Autos, die wir irgendwie weiterleiten müssen,



damit die Kreuzungsregel nicht verletzt wird. Um diesen Stau abzubauen, schieben wir die Autos einfach auf einer der nächsten Straßen weiter (wieder rot). Wir müssen aber wie immer die Spurregel beachten, dürfen also nicht mehr Autos weiterschieben, als Spuren vorhanden sind. Durch das Weiterschieben entsteht natürlich an der nächsten Kreuzung ein neuer Stau (blau), aber wenn die Autos bei Z angekommen sind, müssen wir sie nicht mehr weiterschieben (sondern stellen sie dort einfach auf den Parkplatz).

Wir können also nicht immer so viele Autos weiterschieben, wie wir gerne möchten. Wichtig ist: wir schieben immer so viele Autos wie möglich weiter, jedoch niemals mehr, als durch die Straße passen und natürlich auch nicht mehr, als die Kreuzung Überschuss hat. Und wir müssen natürlich auch auf Einbahnstraßen achten. Falls wir irgendwo gar nicht mehr weiterkommen, weil mehr Autos an einer Kreuzung ankommen als über alle anderen Straßen weiterfahren können, müssen wir auch wieder Autos „zurückschieben“ können (wie an der Kreuzung oben links im nächsten Bild).



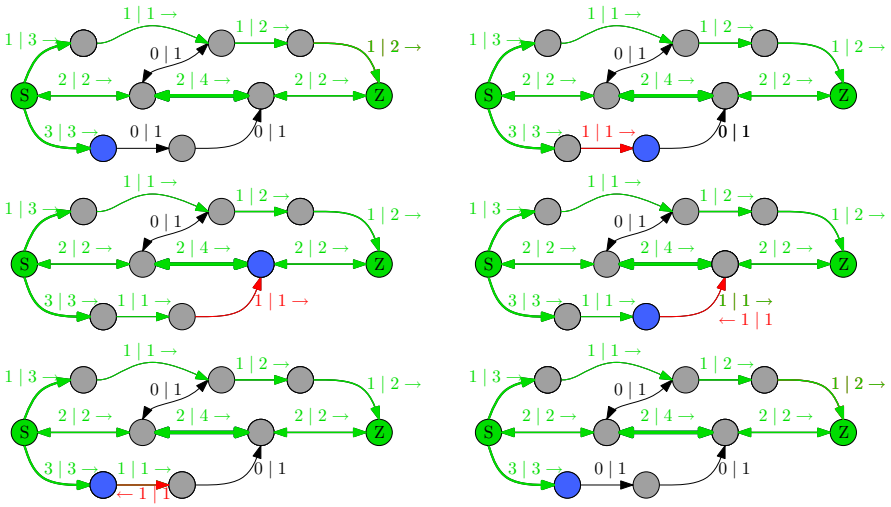
Dabei verringern wir also die Anzahl der Autos, die über eine Straße fahren. – Eigentlich darf man in Einbahnstraßen nicht rückwärts fahren, aber wir schieben hier ja keine echten Autos zurück. – Natürlich schieben wir höchstens so viele Autos zurück, wie nötig sind, damit der Überschuss an der Kreuzung verschwindet (danach ist die Kreuzung grau). Und selbstverständlich können wir nicht mehr Autos zurückschieben, als ankommen.

Insgesamt suchen wir uns in jedem Schritt eine Kreuzung mit Überschuss aus (also eine Kreuzung, an der im Moment mehr Autos ankommen, als weiterfahren) und schieben einen möglichst großen Teil davon weiter. Damit ergibt sich Folgendes:

Die Prozedur WEITERSCHIEBEN schiebt von einer Kreuzung aus, die Überschuss hat, entweder Autos weiter, oder zurück.

```
1  procedure WEITERSCHIEBEN (K)
2  Voraussetzung K ist eine Kreuzung mit Überschuss
3  begin
4      Wähle eine der beiden folgenden Möglichkeiten:
5          Suche unter den Straßen, die von K weg führen, eine Straße aus,
           auf die noch Autos passen und schiebe so viele überschüssige Au-
           tos wie möglich darauf weiter.
       oder
6          Suche unter den Straßen, die zu K hin führen, eine Straße aus,
           auf der Autos zu dieser Kreuzung fahren und schiebe so viele
           überschüssige Autos wie möglich über sie zurück.
7      Ende der Wahl
8  end
```

Leider führt dieses Vorgehen noch nicht zum Erfolg. Es kann dabei nämlich leicht passieren, dass zwei Kreuzungen (oder auch mehr als zwei) ihren Überschuss immer hin und her schieben und wir nie zu einem Ende kommen, wie bei den drei Kreuzungen im Bild. Die Informatiker sagen: „Der Algorithmus *terminiert* (endet) nicht.“



Wir brauchen also eine gute Idee, um die Suche nach dem besten Fluss etwas zielgerichteter zu machen. Führen wir doch zusätzlich die folgende Regel ein: Jede Kreuzung bekommt eine *Höhe*. Am Anfang haben alle Kreuzungen die Höhe 0. Später heben wir die Kreuzungen nach und nach an und zwar so: Wir vereinbaren, dass man Autos immer nur *von oben nach unten* schieben darf. Um also den Überschuss an einer Kreuzung weiterschieben zu dürfen, müssen wir sie erst mal hochheben, sagen wir auf die Höhe 1. Dann dürfen wir zu allen Nachbarkreuzungen, die tiefer liegen, Überschuss weiterschieben (oder zurückschieben). Am Anfang heben wir also S auf die Höhe 1 und schieben wie bisher so viele Autos wie möglich von S weiter. Danach heben wir die nächste Kreuzung, die einen Überschuss hat, auf 1 und schieben weiter, dann die nächste und so weiter. Die Zielkreuzung Z brauchen wir niemals anzuheben, denn wenn die Autos dort angekommen sind, müssen sie ja nicht mehr weiterfahren. Normalerweise können so schon ziemlich viele Autos bei Z ankommen. Trotzdem kann es passieren, dass es nun noch Kreuzungen mit Überschuss gibt, die aber keine Nachbarkreuzungen mit Höhe 0 mehr haben, um ihren Überschuss loszuwerden. Dann dürfen wir sie weiter anheben. Wie weit heben wir sie an? Nun, mindestens um 1. Es kann sein, dass sich dadurch keine neue Möglichkeit zum Schieben ergibt. Dann können wir die Kreuzung noch um 1 weiter anheben. Aber nur so lange, bis sich gerade so wieder eine Möglichkeit zum Weiterschieben (oder zum Rückwärtsschieben!) ergibt. Auch das Anheben der Kreuzungen machen wir natürlich nur im Modell oder auf Papier. Wenn wir die Lösung unseres Problems gefunden haben, ist es nicht

nötig, in der realen Welt die Bagger anrollen zu lassen, um irgendwelche realen Kreuzungen höherzulegen.

Die Prozedur ERHÖHEN hebt eine Kreuzung K an, wenn sie einen Überschuss hat, diesen jedoch nicht weiterschieben kann.

```
1  procedure ERHÖHEN (K)
2  Voraussetzung Kein Weiterschieben des Überschusses von K möglich.
3  begin
4      Erhöhe K so lange, bis sich eine Möglichkeit zum Weiter- oder
       Zurückschieben zu einer niedrigeren Kreuzung ergibt.
5  end
```

In die Prozedur *Weiterschieben* fügen wir die Bedingung hinzu, dass Überschuss nur nach unten geschoben werden kann:

Diese Prozedur WEITERSCHIEBEN hat im Vergleich zur obigen Prozedur WEITERSCHIEBEN die Einschränkung, dass nur „bergab“ geschoben werden kann.

```
1  procedure WEITERSCHIEBEN (K)
2  begin
3      Führe einen der beiden folgenden Schritte durch:
4          Suche eine Straße, auf die noch Autos passen und die von K
           nach beispielsweise L führt. Falls K höher als L ist, schiebe
           überschüssige Autos auf dieser Straße weiter. Jedoch niemals mehr
           als auf die Straße passen und auch niemals mehr, als die Kreuzung
           noch Überschuss hat.
         oder
5          Suche eine Straße aus, auf der schon Autos von z.B. L zu K fahren.
           Falls K höher als L liegt, schiebe Autos zurück. Schiebe nie mehr
           Autos zurück, als die Kreuzung Überschuss hat.
6      Ende der Wahl     (Es ist nicht immer beides möglich!)
7  end
```

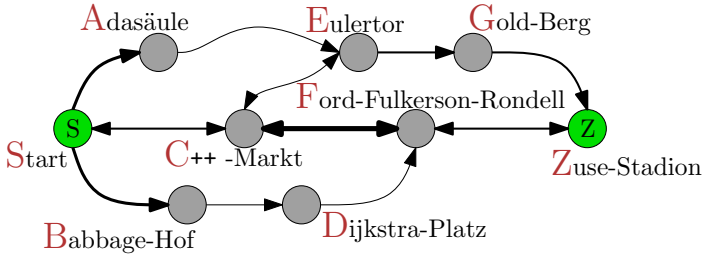
Wir können nun diese beiden Prozeduren (Erhöhen und Weiterschieben) so lange wiederholen, wie es eine Kreuzung mit Überschuss gibt, bzw. so lange, wie es von S aus noch eine freie Straße gibt, auf der wir Autos losschicken können. Wir hören auf, S zu erhöhen, wenn S die Höhe n hat, wobei n die Anzahl der Kreuzungen in unserem Straßennetzwerk ist. Danach beseitigen wir nur noch den Überfluss an den restlichen Kreuzungen und sind fertig. Warum wir dann aufhören können, werden wir erst weiter unten, im Abschnitt „Warum funktioniert das?“ begründen. Ebensogut können wir S auch gleich von Anfang an auf Höhe n heben. In unserem Beispiel hat S also die Höhe 9.

Der eigentlich Algorithmus sieht also so aus:

Der Algorithmus MAXIMALER FLUSS findet einen maximalen Fluss vom Startpunkt S zum Ziel, indem wiederholt die Prozeduren ERHÖHEN und WEITERSCHIEBEN aufgerufen werden.

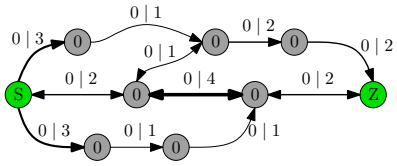
```
1  procedure MAXIMALER FLUSS (G, S, Z)
2  begin
3      Setze S auf Höhe n (n ist die Gesamtzahl der Kreuzungen)
4      Schiebe so viele Autos von S weg, wie jeweils auf die Straßen passen,
       die von S wegführen.
5      Setze alle anderen Kreuzungen (außer S) auf Höhe 0.
6      while Es exisitiert eine Kreuzung K mit Überschuss do
7          if Weiterschieben bei K möglich
8              Führe Prozedur WEITERSCHIEBEN (bei Kreuzung K) aus.
9          else
10             Führe Prozedur ERHÖHEN (bei Kreuzung K) aus.
11     endwhile
12 end
```

Um ein Beispiel für einen kompletten Durchlauf des Algorithmus beschreiben zu können, müssen wir nun erstmal allen Kreuzungen Namen geben:

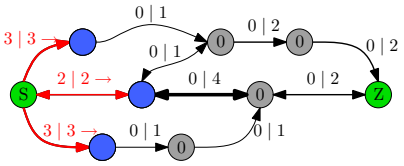


Die folgenden Bilder zeigen den Fortgang des Algorithmus. Wir haben in jede Kreuzung jeweils seine momentane Höhe hineingeschrieben.

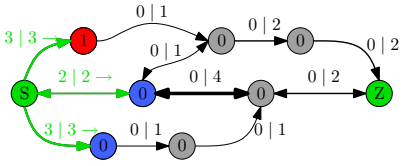
Anfangszustand:



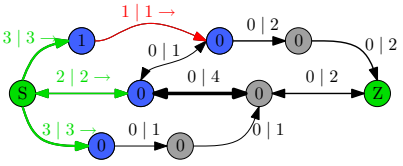
0. Schiebe alle Autos vom Start weg:



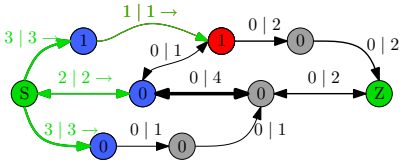
1. ERHÖHEN (Adasäule) auf 1:



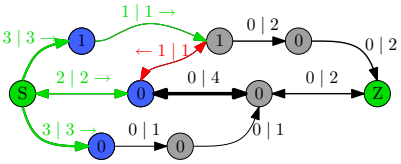
2. WEITERSCHIEBEN (Adasäule):
Ein Auto zum Eulertor:



3. Erhöhen (Eulertor) auf 1



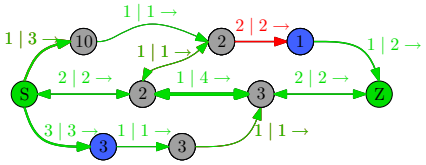
4. Weiterschieben (E): 1 nach C



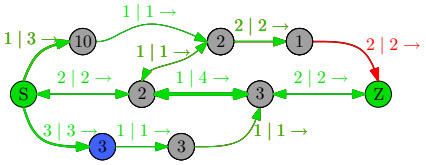
Die restlichen Schritte geben wir nur noch in Kurzform an:

5. Erhöhen (C) auf 1
6. Weiterschieben (C): 3 nach F
7. Erhöhen (F) auf 1
8. Weiterschieben (F): 2 nach Z
9. Erhöhen (F) auf 2
10. Weiterschieben (F): 1 nach C
11. Erhöhen (C) auf 2
12. Weiterschieben (C): 1 nach E
13. Weiterschieben (E): 1 nach G
14. Erhöhen (G) auf 1
15. Weiterschieben (G): 1 nach Z
16. Erhöhen (B) auf 1
17. Weiterschieben (B): 1 nach D
18. Erhöhen (D) auf 2
19. Weiterschieben (D): 1 nach B
20. Erhöhen (B) auf 3
21. Weiterschieben (B): 1 nach D
22. Erhöhen (D) auf 3
23. Weiterschieben (D): 1 nach F
24. Erhöhen (A) auf 10
25. Weiterschieben (A): 2 nach S
26. Erhöhen (F) auf 3
27. Weiterschieben (F): 1 nach C
28. Weiterschieben (C): 1 nach E
29. Erhöhen (E) auf 2

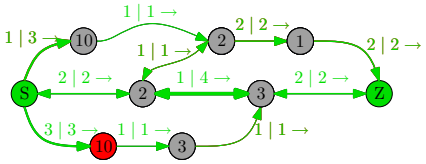
30. Weiterschieben (E): 1 nach G



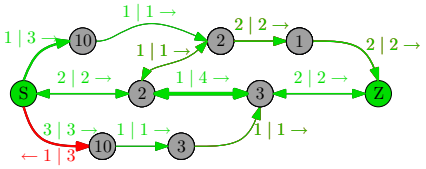
31. Weiterschieben (G): 1 nach Z



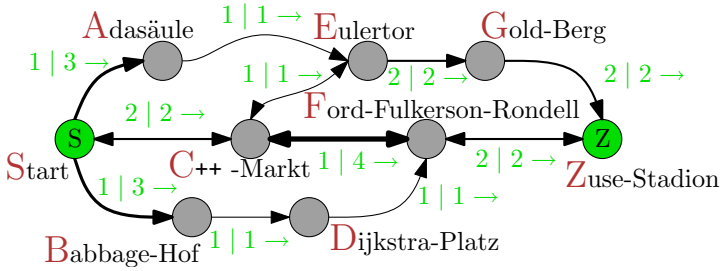
32. Erhöhen (B) auf 10



33. Weiterschieben (B): 2 nach S



Hier ist die Lösung unseres Problems:



Diese Lösung könnten wir nun verwenden, um in der realen Welt den Verkehr zu regeln, ohne dass es Stau gibt.

Einige offene Fragen

- Welche Kreuzung jeweils ausgewählt wird, ist im Algorithmus noch nicht genau festgelegt. Er funktioniert mit jeder beliebigen Reihenfolge, solange wir nur die Regeln für das Weiterschieben („immer nach unten“) und das Hochheben („nur, wenn kein Weiterschieben mehr möglich ist und dann nur so weit, bis sich eine neue Möglichkeit ergibt“) beachten! Unser Beispiel braucht 33 Schritte: 15 mal Erhöhen und 18 mal Weiterschieben (das Erhöhen des Startpunktes ganz am Anfang und das Weiterschieben von dort aus haben wir nicht mitgezählt). Versuche doch mal, das obige Beispiel mit einer anderen Reihenfolge durchzuspielen. Schaffst du es, in weniger Schritten zum Ziel zu kommen?
- Hast du gemerkt, dass Überschuss erst dann Richtung S zurückgeschoben werden kann, wenn die Höhe einer Kreuzung mehr als n ist?

Warum funktioniert das?

Der Algorithmus scheint auf magische Weise zu funktionieren. Wenn du Lust hast, kannst du einfach noch ein bisschen mit einem anderen Straßennetz herumexperimentieren. Falls dich aber interessiert, warum er funktioniert, dann lies einfach weiter:

Als erstes sollten wir uns klar machen, dass am Ende wirklich ein gültiger Verkehrsfluss herauskommt. Dazu müssen wir nur sehen, dass

- wir niemals mehr Autos über eine Straße geschickt haben, als sie Spuren hat (Spurregel) und
- am Ende keine Kreuzung mehr einen Überschuss hat (Kreuzungsregel).

Also kann der Verkehr ungehindert fließen.

Wenn man es an ein paar Beispielen ausprobiert hat, merkt man, dass es einen großen Unterschied gibt zwischen Kreuzungen, die höher als n liegen und Kreuzungen, die niedriger sind. Von hohen Kreuzungen aus wird der Überschuss nämlich immer nur in Richtung S zurückgeschoben. Dies sind die Kreuzungen, die keine Chance mehr haben, ihren Überschuss in Richtung Z loszuwerden. Was aber passiert vorher?

Am Anfang wird immer nur Überschuss von Kreuzungen der Höhe 1 zu Kreuzungen der Höhe 0 geschoben. Dies sind sozusagen die einfachen Fälle. Erst wenn alle einfachen Möglichkeiten ausgeschöpft sind, werden die Kreuzungen nach und nach höher gehoben. Eine wichtige Beobachtung ist, dass Überschuss immer nur um 1 nach unten geschoben wird, also von einer Kreuzung K der Höhe h zu einer Nachbarkreuzung L der Höhe $h-1$. Es kann nie

passieren, dass die Nachbarkreuzung L beispielsweise Höhe $h-2$ hat. Schließlich hätten wir K dann gar nicht so weit hoch heben dürfen. Falls die Autos irgendwann bei Z ankommen sollen, müssen sie also erst von h auf $h-1$, dann von $h-1$ auf $h-2$ und so weiter, langsam hinab bis zu Z, das immer Höhe 0 hat. Also geht es über mindestens h verschiedene Stationen. Dadurch ist nun gewährleistet, dass Überschuss eben nicht ewig zwischen zwei Kreuzungen hin und her geschoben werden kann, denn insgesamt kann es ja höchstens $n-1$ verschiedene Stationen geben ($n-1$, nicht n, da wir nie bei S vorbeikommen werden). Außerdem ist so gewährleistet, dass wirklich alle Möglichkeiten versucht werden, einen Überschuss noch weiterzuschieben, bevor wieder zu S zurückgeschoben wird. Wenn es wirklich keine Möglichkeit mehr gibt, den Überschuss nach Z zu leiten, funktioniert das Zurückschieben zu S nach dem gleichen Prinzip. Am Schluss ist der ganze Überschuss entweder bei Z oder bei S angekommen und wir haben den besten Verkehrsfluss gefunden.

Epilog

Jogis Schwester hat einige Zeit später gelernt, wie man wirklich beweisen kann, dass der Algorithmus den besten Verkehrsfluss findet. Das ist nämlich ziemlich schwierig. Es stellte sich aber heraus, dass es tatsächlich egal war, in welcher Reihenfolge man die Kreuzungen aussucht, von denen man den Überschuss weiterschiebt oder wann man sie anhebt. Sie hat auch erfahren, dass Andrew Goldberg und Robert Tarjan ihn im Jahre 1988 gefunden haben. Jogi steht auf dem Weg zum Stadion trotzdem noch oft im Stau.

Lösung

Es gibt tatsächlich eine Möglichkeit, bei der nur 19 Schritte benötigt werden, um den besten Verkehrsfluss zu finden:

1. Erhöhen (A) auf 1
2. Weiterschieben (A): 1 nach E
3. Erhöhen (C) auf 1
4. Weiterschieben (C): 1 nach E
5. Weiterschieben (C): 1 nach F
8. Erhöhen (B) auf 1
7. Weiterschieben (B): 1 nach D
8. Erhöhen (D) auf 1
9. Weiterschieben (D): 1 nach F
10. Erhöhen (F) auf 1
11. Weiterschieben (F): 2 nach Z
12. Erhöhen (E) auf 1
13. Weiterschieben (E): 2 nach G
14. Erhöhen (G) auf 1
15. Weiterschieben (G): 2 nach Z
16. Erhöhen (A) auf 10
17. Weiterschieben (A): 2 nach S
18. Erhöhen (B) auf 10
19. Weiterschieben (B): 2 nach S

Zum Weiterlesen

1. Kapitel **??**.

   Viele Flussalgorithmen sind angelehnt an eine so genannten Tiefensuche oder Breitensuche im Graphen, so auch der Algorithmus von Ford-Fulkerson. Wie eine Tiefensuche in einem Graphen grundsätzlich funktioniert und wie man sie nutzt, ist in diesem Kapitel nachzulesen.

2. Kapitel **??**.

   In seltenen Fällen kann der Goldberg-Tarjan Algorithmus irgendwo im Graphen Fluss in einem Kreis herumschicken, ohne dass dies zum effektiven Fluss vom Start zum Ziel beiträgt. Mit einer Zyklensuche kann man diese Kreise nachträglich entfernen, nachdem der maximale Fluss gefunden wurde. In diesem Kapitel steht, wie man so etwas macht.

3. Kapitel **??**.

   Ein verwandtes Problem zu den maximalen Flüssen ist die Suche nach einem kürzesten Weg. Hier will man nicht etwa einen ganzen Fluss vom Start zum Ziel senden, sondern für ein einzelnes Auto den schnellsten Weg vom Start zum Ziel finden. Wie man diese kürzesten Wege findet, kann man in diesem Kapitel nachlesen.

4. Die 3D-Animation *Flow Commander* `http://i11www.iti.uni-karlsruhe.de/adw/jaws/GTVisualizer3D.jnlp` (benötigt Java WebStart).

   Warum die Höhe nicht dreidimensional darstellen? Flieg' durch den Graph und schau dir das Schieben und das Anheben am Graphen in 3D an! Wenn auf deinem Rechner Java installiert ist (auf den meisten Rechnern ist das der Fall) kannst bei dieser URL *Flow Commander* installieren und starten.

5. Einer der ersten schnellen Flussalgorithmen: Lestor R. Ford, Jr. und D. R. Fulkerson (1962). *Flows in Networks*. Princeton University Press.

6. Der Originalveröffentlichung des hier gezeigten Algorithmus: Andrew V. Goldberg and Robert E. Tarjan (1988). *A new approach to the maximum-flow problem*. Journal of the ACM 35: 921-940. `http://dx.doi.org/10.1145/48014.61051`

7. Der Artikel in der englischen Wikipedia zum Algorithmus von Goldberg und Tarjan: `http://en.wikipedia.org/wiki/Push-relabel_algorithm`.

8. Der Wikipedia-Artikel zu Flüssen: `http://de.wikipedia.org/wiki/Fluss_(Graphentheorie)`.

   In diesem Artikel kann man unter anderem nachlesen, wie Flüsse und so genannte *Schnitte* zusammenhängen.