Induktion in zwei Dimensionen und dynamische Programmierung

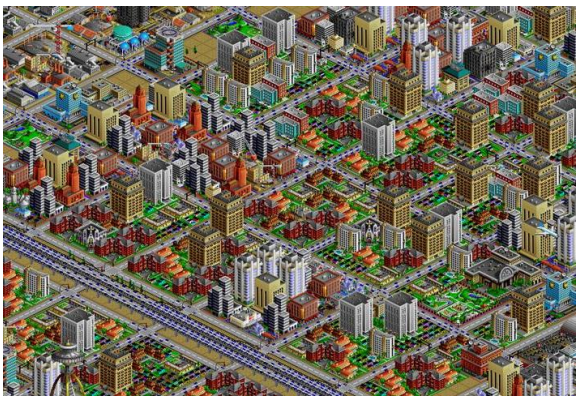
Michael Brand

22.02.2022

Photo by Vlado Paunovic from Pexels (creative commons)

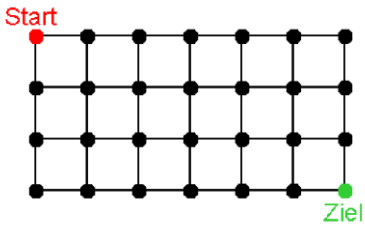
Induktion in zwei Dimensionen

Einige Städte wurden nach einem Raster aufgebaut. Strassen verlaufen entweder in Ost-West-Richtung oder in Nord-Süd-Richtung und trennen so Häuserblöcke ab. In New York beispielsweise gilt, dass *Avenues* von Norden nach Süden verlaufen und *Streets* von Ost nach West. Unten links siehst du einen Ausschnitt von New York von Google Earth mit dem Central Park am linken Bildrand. Auch in einem alten SimCity Spiel wurden Städte rasterförmig gebaut.



Alice wohnt im nordwestlichsten Häuserblock einer solchen Stadt und will ihren Freund Bob besuchen, der im südöstlichsten Block wohnt. Sie fährt mit dem Fahrrad entlang der Strassen und macht keine Umwege, d.h., sie fährt immer Richtung Osten oder Richtung Süden, aber nie west- oder nordwärts.

Im Bild rechts siehst du einen vereinfachten Stadtplan. Die schwarzen Punkte sind Kreuzungen. Es gibt sieben Kreuzungen in Ost-West-Richtung und vier Kreuzungen in Nord-Süd-Richtung. Wir wollen das konzis eine $(4,7)$-Stadt nennen.



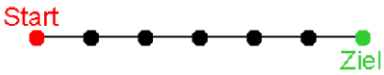
Konvention

In einer **(m, n)-Stadt** gibt es m Kreuzungen in Nord-Süd-Richtung und n Kreuzungen in Ost-West-Richtung. Ein **Weg** in einer solchen Stadt führt entlang der Strassen in östlicher oder südlicher Richtung (aber nie in westlicher oder nördlicher Richtung).

Aufgabe 1

Zeichne im obigen Stadtplan einige kürzeste Wege vom Start zum Ziel ein. Wie viele Häuserblöcke muss Alice entlangfahren, um ans Ziel zu gelangen? Welche Gemeinsamkeit haben alle Wege?

Eines Tages fragt sich Alice, wie viele Wege es durch die Stadt gibt, um von ihrer Wohnung zu Bob zu gelangen. Früher war das einfacher, denn es gab nur eine einzige lange Strasse in Ost-West-Richtung.



Aufgabe 2

Wie würdest du diese Stadt gemäss unserer Konvention bezeichnen? Wie viele Wege führen vom Start zum Ziel? Kannst du die Aussage auch verallgemeinern?

Fatalerweise gehört Bob zu den Leuten, die immer im neuesten Stadtteil wohnen wollen und jedes Mal, wenn ein neuer Wohnblock entsteht, zieht er um.

Aufgabe 3

Zeichne eine $(2,7)$-Stadt und zähle alle Wege vom Start oben links zum Ziel unten rechts. Bestimme die Anzahl Wege $T(n)$ in einer $(2,n)$-Stadt. Beweise deine Vermutung über die vollständige Induktion.

Aufgabe 4

Untersuche nun systematisch, wie viele Wege es in einer (m,n)-Stadt gibt. Fülle dazu die Tabelle auf der nächsten Seite aus. Der Eintrag oben links in der Tabelle bezieht sich auf eine $(1,1)$-Stadt. Hier leben Alice und Bob im gleichen Gebäude und wir sagen, dass es einen Weg vom Start zum Ziel gibt, nämlich den "trivialen" Weg, bei dem Alice über keine Strasse fahren muss.

m \ n	1	2	3	4
1	1			
2				
3				
4				

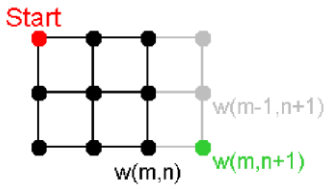
Stelle eine Vermutung an, wie die Tabelle weiter fortgesetzt werden könnte.

Wir möchten nun unsere Vermutung, dass die Diagonalen der Tabelle das Pascal'sche Dreieck bilden, mathematisch beweisen. Dazu bezeichnen wir mit $w(m,n)$ die Anzahl Wege, die es in einer (m,n)-Stadt gibt. Mit dieser Bezeichnung lässt sich unsere Tabelle folgendermassen schreiben:

m \ n	1	2	3	4
1	$w(1,1) = 1$	$w(1,2) = 1$	$w(1,3) = 1$	$w(1,4) = 1$
2	$w(2,1) = 1$	$w(2,2) = 2$	$w(2,3) = 3$	$w(2,4) = 4$
3	$w(3,1) = 1$	$w(3,2) = 3$	$w(3,3) = 6$	$w(3,4) = 10$
4	$w(4,1) = 1$	$w(4,2) = 4$		

Die erste Zeile und die erste Spalte konnten wir sofort ausfüllen, denn gemäss Aufgabe 2 gilt:

$$w(1,n) = w(m,1) = 1$$



Nehmen wir nun an, dass wir die Tabelle bereits bis zu einem gewissen $w(m,n)$ ausfüllen konnten (z.B. den grün markierten Teil). Wie können wir nun die nächste Spalte ausfüllen? Dazu müssen wir uns überlegen, wie wir zu dem grünen Punkt kommen, wir müssen also $w(m,n+1)$ berechnen. Wenn wir uns in dem Punkt $w(m,n)$ befinden, dann müssen wir die Strasse nach Osten nehmen. Das erhöht die Anzahl der Wege nicht. Es gibt also gleich viele Wege, um von Start aus über $w(m,n)$ nach

$w(m, n+1)$ zu gelangen, wie es Wege von Start aus zum Punkt $w(m, n)$ gibt. Allerdings können wir den Punkt $w(m, n+1)$ auch erreichen, indem wir von dem Punkt $w(m-1, n+1)$ aus die Strasse nach Süden nehmen. Es gilt also:

$$w(m, n+1) = w(m, n) + w(m-1, n+1)$$

In Worte gefasst bedeutet dies: Jede Zahl in der Tabelle ist die Summe der Zahl links davon und der Zahl darüber. Dies ist aber genau die Bildungsvorschrift des Pascal'schen Dreiecks. Wir können also eine neue Spalte von oben nach unten ausfüllen, indem wir in die erste Zeile eine 1 setzen und uns dann von oben nach unten gemäss dem Bildungsgesetz durcharbeiten. Dabei spielt es übrigens keine Rolle, ob wir eine neue Spalte oder eine neue Zeile einfügen, denn eine (n, m)-Stadt geht aus einer (m, n)-Stadt durch Spiegelung hervor.

Hintergrund: Induktion in zwei Dimensionen

Du kennst bereits das Induktionsprinzip, bei dem man die Korrektheit einer Aussage für alle $n \in \mathbb{N}$ beweist, indem man zeigt, dass die Aussage für $n = 1$ (oder einen anderen Basisfall) gültig ist und dann von der Gültigkeit für $n = k$ auf die Gültigkeit für $n = k + 1$ schliesst. Dieses Prinzip lässt sich auf zwei und mehr Dimensionen verallgemeinern, d.h. auf Aussagen, die mehr als einen Parameter enthalten.

Wir wollen uns die Funktionsweise eines Induktionsbeweises mit zwei Parametern anhand eines Beispiels anschauen.

Beispiel zur zweidimensionalen Induktion

Wir geben eine rekursive Bildungsvorschrift für das Ausfüllen einer Tabelle an, ähnlich wie in unserem Beispiel zu dem Pascal'schen Dreieck. Die Vorschrift lautet:

$$\begin{aligned} &f(1,1) = 2 \\ &f(m+1, n) = f(m, n) + 2(m+n) \\ &f(m, n+1) = f(m, n) + 2(m+n-1) \end{aligned}$$

Aufgabe 5

Fülle die untenstehende Tabelle gemäss dieser Vorschrift aus. Überzeuge dich davon, dass es keine Rolle spielt, ob du die Tabelle spaltenweise oder zeilenweise ausfüllst. Du kannst das entweder von Hand machen oder eine Software zur Tabellenkalkulation wie LibreOffice Calc oder Microsoft Excel verwenden.

m \ n	1	2	3	4
1	2			
2				
3				
4				

Wir wollen nun beweisen, dass man jeden Eintrag auch direkt mit der Formel

$$f(m,n) = (m+n)^2 - (m+n) - 2n + 2$$

berechnen kann.

Aufgabe 6

a) Zeige, dass die Formel für $f(1,1)$ den richtigen Wert berechnet. Das ist der Basisfall.

b) Nehme an, dass die Formel für $f(k,1)$ korrekt ist (für ein $k \in \mathbb{N}$), und zeige, dass die Formel dann auch für $f(k+1,1)$ korrekt ist. Das ist also eine gewöhnliche Induktion über m.

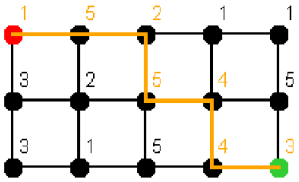
c) Nun müssen wir noch die Induktion über n führen. Dazu sei $k \in \mathbb{N}$ eine beliebige, aber fixierte Zahl. Nehme an, dass die Formel für $f(k,h)$ korrekt ist (für ein $h \in \mathbb{N}$), und zeige, dass die Formel dann auch für $f(k,h+1)$ korrekt ist.

Der geführte Induktionsbeweis entspricht also dem zeilenweisen Ausfüllen der Tabelle. In Aufgabe a) hast du gezeigt, dass die Formel den ersten Eintrag oben links richtig berechnet. In Aufgabe b) hast du induktiv gezeigt, dass die Formel alle Einträge der 1. Spalte richtig berechnet. In Aufgabe c) hast du schliesslich mit einer weiteren eindimensionalen Induktion gezeigt, dass die Formel jede Zeile (ausgehend vom Eintrag in der 1. Spalte) richtig berechnet. Dabei hätten wir die Schritte b) und c) auch entsprechend vertauschen können.

Kürzeste Wege mit dynamischer Programmierung finden

Entwicklung des Algorithmus

Durch zahlreiche Bautätigkeiten in den letzten Jahren ist das Geld in der Staatskasse knapp geworden. Aus diesem Grund beschliesst die Stadtregierung, dass jeder Verkehrsteilnehmer bei jeder Kreuzung, die er passiert, eine Abgabe zu entrichten hat. Die Höhe der Gebühr ist bei jeder Kreuzung unterschiedlich. Auf dem rechts abgebildeten Weg muss Alice also insgesamt $1 + 5 + 2 + 5 + 4 + 4 + 3 = 24$ CHF an Abgaben entrichten, um zu Bob zu gelangen:

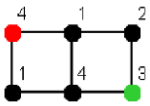


Natürlich fragt sie sich nun, ob es nicht einen günstigeren Weg gibt.

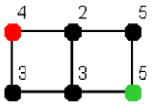
Aufgabe 7

Bestimme den preisgünstigsten Weg vom Start (roter Punkt) zum Ziel (grüner Punkt) in den folgenden Fällen

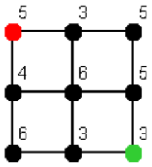
a)



b)



c)



Aufgabe 8

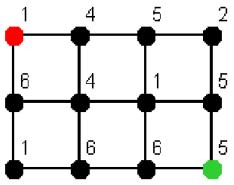
Ist der preisgünstigste Weg immer eindeutig?

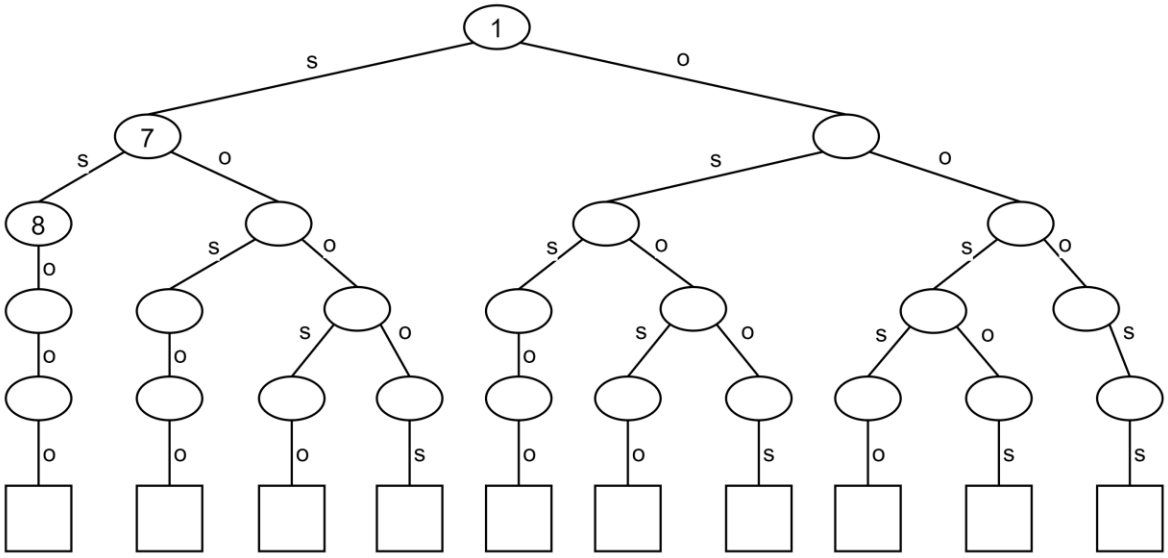
Aufgabe 9

Eine Strategie für das Finden eines preisgünstigsten Weges könnte ein Greedy-Algorithmus sein. Dabei wählt man in jedem Schritt die preisgünstigere von den zwei Alternativen (oder einen beliebigen Weg, wenn beide Nachbarkreuzungen gleich teuer sind). Diese Strategie versagt allerdings bei diesem Problem. Konstruiere eine $(3,3)$-Stadt, bei der der Greedy-Algorithmus nicht den preisgünstigsten Weg findet.

Aufgabe 10

Eine Möglichkeit, die sicher das richtige Ergebnis liefern wird, ist die Brute-Force-Methode, d.h. alle möglichen Wege durchzuprobieren. Führe diese Methode an der abgebildeten $(3,4)$-Stadt aus. Erinnere dich, dass ein Weg durch ein Wort mit 2 S und 3 O beschrieben wird. Diese Worte sind alle im Baumdiagramm abgebildet. Fülle in die Ovale jeweils die aktuellen Gesamtkosten entlang eines Weges ein.





Aufgabe 11

Warum ist die Brute-Force-Methode aus der vorigen Aufgabe nicht praktikabel, obwohl sie immer die richtige Lösung liefert?

Tatsächlich haben wir uns mit der Brute-Force-Methode zu viel Aufwand gemacht. Wir haben z.B. berechnet, dass man vom Start aus auf dem Weg SO Gesamtkosten von 11 CHF erhält, aber auf dem Weg OS lediglich Kosten von 9 CHF hat. Wenn wir aber einmal bei der Kreuzung in der 2. Zeile und 2. Spalte angekommen sind, dann spielt es für die Folgekosten keine Rolle mehr, wie wir zu dieser Kreuzung gelangt sind. Wir hätten uns im Baum also alle Wege nach SO (Kosten 11 CHF) sparen können, denn die analogen Wege mit dem Anfang OS (Kosten 9 CHF) sind sicher preisgünstiger.

Dies ist die Grundidee der dynamischen Programmierung. Wir fassen die Kosten im Stadtplan zunächst in einer Kostentabelle zusammen, deren Einträge $g(m,n)$ die Preise an den Kreuzungen darstellen (g wie "Geld"; diese Zahlen werden häufig auch als "Gewichte" bezeichnet) . Danach berechnen wir eine Gesamtkostentabelle. Die Einträge $k(m,n)$ geben an, wie viel der preisgünstigste Weg zur Kreuzung in der m-ten Zeile und n-ten Spalte kostet. Die Tabelleneinträge $k(1,n)$ und $k(m,1)$ können wir direkt notieren, denn diese Kreuzungen sind nur auf einem Weg erreichbar:

Kostentabelle

m \ n	1	2	3	4
1	1	4	5	2
2	6	4	1	5
3	1	6	6	5

Minimale Gesamtkostentabelle

m \ n	1	2	3	4
1	1	5	10	12
2	7			
3	8			

In der linken Tabelle bedeutet also $g(1,4) = 2$ die Kosten, die beim Passieren der Kreuzung in der 1. Zeile und 4. Spalte anfallen, und in der rechten Tabelle sind $k(1,4) = g(1,1) + g(1,2) + g(1,3) + g(1,4) = 12$ die Kosten, die Alice insgesamt hat, um vom Start bis zu dieser Kreuzung zu gelangen.

Aufgabe 12

Wie gross ist nun $k(2,2)$?

Mit der gleichen Methode füllen wir nun den Rest der Tabelle aus. Die Bildungsvorschrift lautet:

$$k(m+1, n+1) = \min\big(k(m+1, n), k(m, n+1)\big) + g(m+1, n+1)$$

Dabei sind die $k(1, n)$ und $k(m, 1)$ bekannt.

	…	n	$n+1$
…		…	…
m	…	$k(m,n)$	$k(m,n+1)$
$m+1$	…	$k(m+1,n)$	$k(m+1,n+1)$

Hintergrund

Diese Methode ist als dynamische Programmierung bekannt. Der Begriff wurde von dem Amerikaner Richard Bellman in den 1950er Jahren geprägt. In seiner Autobiographie schrieb er, dass er diesen Begriff erfunden habe, als er bei der RAND Corporation als Berater für das US-Militär zuständig war und fürchtete, dass seine mathematische Forschung nicht mehr finanziert würde. Das Wort *Programming* wurde in Militärkreisen als die Suche nach einem optimalen Programm, z.B. für die Logistik oder das Training der Truppen, verstanden. *Linear Programming* war ein Synonym für Optimierung im mathematischen Sinn. Im Gegensatz dazu war seine Methode aber dynamisch, denn sie liess sich auf Probleme mit mehreren Parametern anwenden. "*Thus, I thought dynamic programming was a good name. It was something that not even a Congressman could object to*". [Zitiert in https://en.wikipedia.org/wiki/Dynamic_programming]

Aufgabe 13

Fülle die minimale Gesamtkostentabelle komplett aus.

Aufgabe 14

Wie können wir die Methode anpassen, so dass wir zusätzlich zu den minimalen Kosten auch noch den kürzesten Weg selbst bestimmen können?

Überlegen wir uns zum Schluss noch, wie viele Operationen für diesen Algorithmus nötig sind, also bestimmen wir seine Laufzeit. Für jedes Feld in der Tabelle (ausser für das erste) fällt eine Addition an, das sind für eine (m, n)-Stadt also $m \cdot n - 1$ Additionen. Ausserdem muss für alle Felder, ausgenommen die erste Zeile und die erste Spalte, auch noch ein Vergleich vorgenommen werden, um jeweils das Minimum zu bestimmen. Das sind $(m-1) \cdot (n-1) = m \cdot n - \cdots$ Vergleiche. Der Algorithmus hat also eine quadratische Laufzeit, was viel besser als die faktorielle Laufzeit der Brute-Force-Methode ist.

Aufgabe 15 (optional)

a) Schreibe ein Programm, welches zu einer gegebenen Kostentabelle die minimale Gesamtkostentabelle berechnet.

b) Ergänze dein Programm so, dass auch der kürzeste Weg als ein Wort, konstruiert mit den Buchstaben S und O, ausgegeben wird.

Was du gelernt hast

Wir haben gesehen, wie wir das Prinzip der Induktion auf Aussagen mit zwei Parametern erweitern können. Wir haben einen Wert, der uns interessiert, wie z.B. die Gesamtzahl der Wege oder den niedrigsten Preis, dadurch erhalten, indem wir eine Tabelle schrittweise nach einem bestimmten Bildungsgesetz, d.h. unter Einbezug bereits berechneter Grössen aus einem Fall mit kleineren Parameterwerten, ausgefüllt haben.

In der Informatik ist dies auch als dynamische Programmierung bekannt und ist eine Strategie, Programme zu entwerfen, die den unnötigen Rechenaufwand einer Brute-Force-Methode umgehen. Mit dieser Methode ist es uns gelungen, einen Algorithmus zu entwerfen, der einen kürzesten (preisgünstigsten) Weg im Gitter in quadratischer Laufzeit berechnen kann.

Ausblick

Unser vereinfachter Stadtplan einer (m, n)-Stadt ist ein Beispiel eines Graphen, nämlich ein so genannter Gittergraph. Ein Graph besteht allgemein aus einer Menge von Knoten (Kreuzungen), die durch Kanten (Strassen) verbunden sind. Diese Kanten können mit Gewichten (Kosten) versehen sein, die z.B. Distanzen zwischen zwei Knoten repräsentieren. Auch in einem solch allgemeinen Graphen gibt es einen Algorithmus, der die kürzeste Distanz zwischen zwei Knoten berechnet, den Algorithmus von Dijkstra. Er kann ebenfalls mit dynamischer Programmierung gefunden werden. Der hier vorgestellte Algorithmus ist ein Spezialfall des Algorithmus von Dijkstra.