2 Tiefen- und Breitensuche

Übersicht

2.1 Spannende Bäume

Vor nicht allzu langer Zeit hat Karl auf dem Dachboden seiner Großmutter eine alte verstaubte Karte gefunden, auf der die Umrisse eines alten Palastes eingezeichnet sind. Nach einigen Wochen des Nachforschens in verschiedenen Bibliotheken meint er sein Ziel gefunden zu haben und reist mit einer Gruppe von Freunden zu dem vermeintlichen Ort. Dort stellen sie auch bald mit Hilfe neuer Radartechnik fest, dass tief unter der Erde ein Gebäude verschüttet liegt. Nach einigen Wochen Arbeit sind sie in den ersten Raum vorgedrungen und müssen nun etwas enttäuscht feststellen, dass alle Gänge mit einer starken Eisentür versperrt sind (Abb. 2.1).

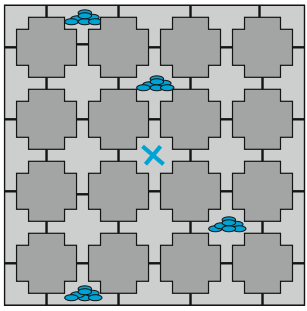

Abb. 2.1: Die Freunde sind im Zimmer mit dem blauen Kreuz gelandet. Wie können sie sicherstellen, dass ihnen der Zugang zu keinem Schatz verwehrt bleibt und sie dabei möglichst wenig Türen sprengen müssen? Die schwarzen Balken in den Gängen symbolisieren die dicken Türen.

In den Eisentüren befindet sich zwar ein Schlüsselloch, aber da es so dunkel ist, kann man den Inhalt der angrenzenden Räume nicht sehen. Zu allem Überfluss lassen sich die Schlüssel nicht finden; nun überlegen die Freunde Tag und Nacht, wie sie die Schätze wohl am besten bergen können. Immer von oben nach unten zu graben, würde viel zu lange dauern. Darauf haben sie sich schnell geeinigt. Karls nächste Idee, die Türen zu sprengen, wird jedoch mit großer Begeisterung angenommen. Jetzt brauchen sie nur noch einen Plan, wo die Sprengladungen nach und nach platziert werden. Der Sprengplan soll garantieren, dass nach seiner Durchführung alle Räume zu besichtigen sind, und dabei möglichst wenige Sprengungen enthalten. Wie sieht Ihr Vorschlag aus? Wie viele Sprengungen sind mindestens notwendig, um alle Kammern zu erreichen?

Das Problem lässt sich in die Sprache der Graphen übertragen: Die Knoten des Graphen stellen die unterschiedlichen Räume dar und die Gänge zwischen den Räumen die Kanten. Damit haben wir den Palastgrundriss modelliert. Fehlt noch der Sprengplan. In der Graphentheorie entspricht dieser einem Teilgraphen und jede Kante des Teilgraphen spiegelt einen Sprengvorgang wider. Da Max von dem mittleren Raum aus zu allen anderen Räumen gelangen will, suchen wir einen zusammenhängenden Teilgraphen, der alle Knoten enthält. Außerdem sollen möglichst wenig Sprengungen durchgeführt werden. Insbesondere Kreise kann man sich sparen. Also suchen wir nach einem Baum. Ein solcher Baum wird auch als spannender Baum bezeichnet, da er den ursprünglichen Graphen aufspannt.

Definition. Sei $G = (V, E)$ ein Graph. Ein Teilgraph $T = (V_0, E_0)$ von G, der ein Baum ist und für den $V_0 = V$ gilt, heißt spannender Baum (Abb. 2.2).

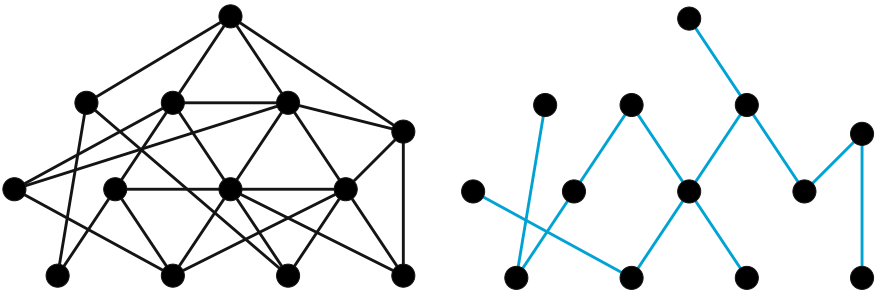

Abb. 2.2: Die blauen Kanten (rechts) bilden einen spannenden Baum in G (links).

Besitzt denn jetzt jeder Graph einen spannenden Baum, bzw. können wir Karl garantieren, dass er auf jeden Fall einen günstigen Sprengplan finden wird? In dem folgenden Graphen (Abb. 2.3) erkennt man recht schnell, dass ein spannender Baum nicht existieren kann. Wie soll der Baum denn auch die Knoten a und b miteinander verbinden, wenn zwischen den beiden Knoten schon im ursprüngli-

chen Graphen kein Weg existiert? Wie würde ein entsprechender Palastgrundriss aussehen, für den es keinen geeigneten Sprengplan gibt?

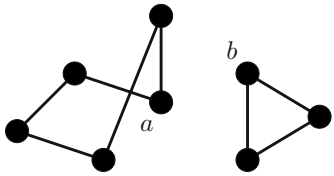

Abb. 2.3: Der Graph ist nicht zusammenhängend. Also kann auch kein spannender Baum in ihm existieren.

Wir müssen also fordern, dass der Graph zusammenhängend ist. Diese Bedingung reicht dann schon aus.

Satz 2.1. *Jeder zusammenhängende Graph enthält einen spannenden Baum.*

Beweis: Die Idee des Beweises ist es, nach und nach Kanten aus einem gegebenen Graphen G zu löschen, sodass der neue Graph zusammenhängend bleibt, aber keine Kreise mehr enthält. Sei G also ein zusammenhängender Graph. Entweder ist G bereits ein Baum, dann ist der Beweis erbracht, oder G besitzt einen Kreis C. Entfernen wir aus G eine Kante e des Kreises C, so ist $G' = G - e$ nach wie vor zusammenhängend (Abb. 2.4).

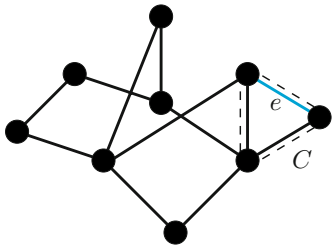

Abb. 2.4: Löscht man die blaue Kante aus dem Graphen, so ergibt sich ein neuer, zusammenhängender Teilgraph des ursprünglichen Graphen.

Entweder ist G' nun kreisfrei oder G' besitzt wieder einen Kreis C'. Im zweiten Fall entfernen wir wieder eine Kante e' aus C'. Diesen Löschvorgang wiederholen wir so lange, bis der Graph keine Kreise mehr enthält. (Wie viele Kanten müssen in einem gegebenen Graphen mit m Kanten gelöscht werden? Wie viele Kanten können es maximal sein?)[6] Da der neue Graph dieselben Knoten hat, wie der ursprüngliche, zusammenhängend und kreisfrei ist, ist er ein spannender Baum (Theorem 1.8). □

Der Beweis enthält schon eine Idee, wie man mit Hilfe eines Algorithmus einen spannenden Baum konstruieren kann. Allerdings ist es schwierig in einem gegebenen Graphen einen Kreis zu finden, ohne vorher einen spannenden Baum berech-

net zu haben. Im nächsten Abschnitt werden wir zwei einfache Methoden kennen lernen, einen spannenden Baum zu berechnen.

2.2 Wie findet man spannende Bäume?

Auf dem Schatzplan einen spannenden Baum zu finden, ist nicht so schwierig. Ihnen sind sicher schon einige Möglichkeiten eingefallen. Aber wie erhalten wir einen spannenden Baum, wenn uns ein viel größerer Graph gegeben ist? Der Graph basierend auf dem Stadtplan von Berlin hat zum Beispiel 10.000 Knoten und 30.000 Kanten. Wir brauchen also Regeln, mit denen wir das Problem lösen können.
Schauen wir noch einmal auf den Schatzplan und überlegen uns, wie wir hier sinnvoll einen Sprengplan konstruieren können. Eine erste Idee dazu ist die folgende: Wir starten in dem ersten Zimmer mit dem Kreuz und zünden dort eine helle Fackel an. Von dort aus betrachten wir alle Gänge nacheinander, die von diesem Zimmer aus losgehen. Als Erstes schauen wir durch das Schlüsselloch der Tür in dem Gang. Wenn wir ein Licht durch das Schlüsselloch sehen, wissen wir, dass wir in dem angrenzenden Zimmer schon einmal waren. Wir markieren den Gang mit einem dicken Kreuz, damit wir die Tür auf keinen Fall sprengen (Abb. 2.5). Ist es hingegen in dem anderen Zimmer dunkel, so waren wir noch nicht dort und sprengen die Tür. Als Nächstes kann dann ein Suchtrupp das Zimmer untersuchen und eine Fackel anzünden, während der Sprengmeister die restlichen Türen und Gänge begutachtet. Die Schatzsucher könnten auf diese Weise zu dem folgenden Netz gekommen sein (Abb. 2.5). Welche Türen würden sie danach sprengen? Hat die Wahl des nächsten Zimmers eine Auswirkung darauf, welche Türen zerstört werden?

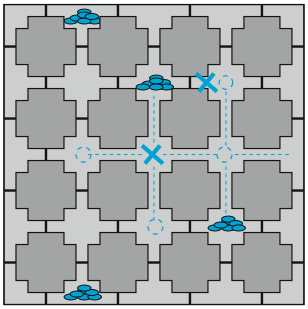

Abb. 2.5: Bisher haben die Schatzsucher erfolgreich die ersten Zimmer begutachtet. Wie sollten sie weitermachen?

Damit wir die obige Idee auch allgemein umsetzen können, sodass zum Beispiel auch ein Computer sie versteht, sollten wir sie noch einmal etwas genauer aufschreiben:

Algorithmus 2.1 Breitensuche (engl. *breadth first search*, BFS)

Input: Zusammenhängender Graph $G = (V, E)$.

Output: Aufspannender Baum $T = (V, E_0)$ von G.

Schritt 1:
- Sei E_0 zunächst leer.
- Bereiten Sie zwei ebenfalls leere Knotenlisten L_1und L_2 vor.
- Wählen Sie einen beliebigen Startknoten $v_0 \in V$ und schreibe v_0 in L_1 und L_2.

Schritt 2: Falls L_1 leer ist, dann STOPP und return $T = (V, E_0)$. Ansonsten nehmen Sie den ersten Knoten v aus L_1.

Schritt 3: Falls alle Nachbarknoten von v in L_2 sind, so löschen Sie v aus L_1. Ansonsten wählen Sie einen der Nachbarknoten w aus, der nicht in L_2 ist, schreiben Sie diesen ans Ende der Listen L_1 und L_2 und fügen Sie die Kante (v, w) zu E_0 hinzu.

Schritt 4: Gehen Sie zu Schritt 2.

Übertragen wir den Algorithmus wieder zurück in das Schatzkartenbeispiel, so gilt: Die Kantenmenge E_0 speichert die Türen, die gesprengt werden; in der Liste L_2 merken wir uns alle Zimmer, die wir schon besucht haben, und die Liste L_1 enthält all diejenigen erreichten Zimmer, von denen aus noch nicht alle Türen überprüft bzw. gegebenenfalls auch gesprengt wurden.
Wenden wir den Algorithmus doch einmal auf den folgenden Graphen an.

Beispiel. Wir beginnen im Graphen aus der Abbildung 2.6 mit dem Knoten a. Dann gilt also $v_0 = a$, $L_1 = \{a\}$, $L_2 = \{a\}$ und E_0 enthält noch keine Kante. In Schritt 2 wählen wir jetzt den ersten Knoten aus der Liste L_1, d.h. den Knoten a. Bisher ist noch keiner seiner Nachbarn in der Liste L_2 eingetragen. Wir fügen nun einen seiner Nachbarn, z.B. b, in die Liste L_1 und L_2 und die Kante (a, b) in E_0 ein. Damit gilt

$$L_1 = \{a, b\}, L_2 = \{a, b\} \text{ und } E_0 = \{(a, b)\}.$$

Da noch nicht alle Nachbarn von a in L_2 sind, betrachten wir nun den Knoten c und somit gilt nach Schritt 3

$$L_1 = \{a, b, c\}, L_2 = \{a, b, c\} \text{ und } E_0 = \{(a, b), (a, c)\}.$$

Jetzt sind alle Nachbarn von a in L_2 und wir löschen a aus L_1, d.h. $L_1 = \{b, c\}$. Als Nächstes gehen wir zum Knoten b. Auch hier haben wir noch zwei Nachbarn, die nicht in L_2 enthalten sind, nämlich d und g. Nachdem wir diese beiden betrachtet haben, können wir auch b aus L_1 löschen und erhalten

$$L_1 = \{c, d, g\}, L_2 = \{a, b, c, d, g\} \text{ und } E_0 = \{(a, b), (a, c), (b, d), (b, g)\}$$

(Abb. 2.6). Wie sieht die Fortführung des Algorithmus aus? ■

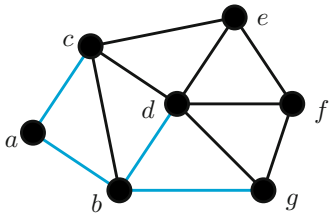

Abb. 2.6: Wir starten mit dem Knoten a und führen nach und nach die Schritte des Algorithmus aus.

Auf diesem Graphen hat der Algorithmus funktioniert und einen spannenden Baum berechnet. Bisher haben wir aber noch nicht bewiesen, dass der Algorithmus immer einen solchen Baum berechnet. Wir könnten auch einfach Glück gehabt haben. Damit wir uns nicht noch länger mit dieser Unsicherheit abfinden müssen, wollen wir seine Korrektheit beweisen.

Satz 2.2. *Wenn ein Graph zusammenhängend ist, so lässt sich ein spannender Baum mit der Breitensuche (Algorithmus 2.1) berechnen.*

Beweis: Den Beweis führen wir in drei Teilen: Zunächst zeigen wir, dass am Ende des Algorithmus alle Knoten in L_2 sind, d.h. $L_2 = V$. Danach beweisen wir, dass der berechnete Graph (L_2, E_0) zusammenhängend ist, und dann, dass er kreisfrei ist. Aus den beiden letzten Eigenschaften können wir sofort folgern, dass der Graph ein Baum ist (Satz 1.8).
Teil 1: Am Ende des Algorithmus enthält L_2 sämtliche Knoten.

Beweis: Beweis durch Widerspruch. Angenommen, am Ende des Algorithmus gibt es einen Knoten u_0, der nicht in L_2 enthalten ist, d.h. $u_0 \notin L_2$.
Da G zusammenhängend ist, gibt es einen Weg vom Startknoten v_0 zu u_0. Sei u_1 der erste Knoten auf diesem Weg, der nicht in L_2 ist, und v_1 dessen Vorgängerknoten auf dem Weg, also $u_1 \notin L_2$ und $v_1 \in L_2$. Da niemals Knoten aus L_2 gelöscht werden, war u_1 niemals in L_2 (Abb. 2.7).

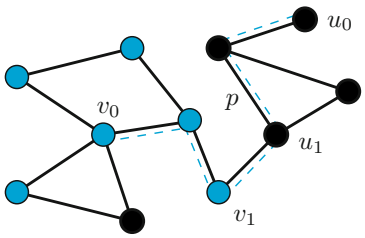

Abb. 2.7: Der Knoten u_0 ist mit v_0 über einen Weg verbunden. Die blauen Knoten sind in der Menge L_2, die anderen nicht.

Nach der Anweisung in Schritt 2 werden Knoten nur gleichzeitig in L_1 und L_2 eingetragen. Da $v_1 \in L_2$ ist, war v_1 auch in L_1. Der Algorithmus stoppt erst,

wenn L_1 leer ist, d.h. v_1 muss irgendwann gelöscht worden sein. Allerdings erfolgt dies erst, wenn alle Nachbarknoten in L_2 sind. Das ist ein Widerspruch dazu, dass v_1 den Nachbarknoten u_1 besitzt, der nach unserer Wahl noch nicht in L_2 enthalten ist. Die Annahme $u_0 \notin L_2$ ist also falsch. Somit enthält L_2 am Ende sämtliche Knoten. □

Teil 2: Der berechnete Graph (V, E_0) ist zusammenhängend.

Beweis: Den Zusammenhang erhalten wir durch die Art und Weise, wie Knoten und auch Kanten in die Liste L_2 bzw. in die Kantenmenge E_0 in Schritt 3 eingefügt werden. Wir beweisen die Aussage mit vollständiger Induktion, indem wir zeigen, dass zu jedem Zeitpunkt im Algorithmus die Knoten aus L_2 und E_0 einen zusammenhängenden Graphen bilden. Die Induktion geht über die Anzahl der Knoten in L_2. Zum Start des Algorithmus stimmt die Aussage, da der erste Knoten v_0 zusammenhängend ist. Damit haben wir den *Induktionsanfang*.
Die *Induktionsannahme* lautet: Die bisherigen k Knoten aus der Liste L_2 zusammen mit den bisherigen Kanten in E_0 bilden einen zusammenhängenden Graphen. Im *Induktionsschluss* müssen wir zeigen, dass der Knoten, der als nächstes in die Liste L_2 eingefügt wird, ebenfalls über einen Weg mit den anderen k Knoten aus der Liste verbunden ist. Betrachten wir also den Schritt des „Einfügens“ eines neuen Knotens w etwas genauer (Abb. 2.8). Der Knoten w wird in die Liste L_2 eingefügt, wenn ein Knoten v aus L_1 (und somit auch aus L_2) zu ihm benachbart ist. Zu diesem Zeitpunkt wird dann die Kante (v, w) in die Menge E_0 einbezogen und w sowohl in der Liste L_1 als auch in der Liste L_2 gespeichert. Da v mit jedem Knoten in der Liste L_2 über einen Weg verbunden ist (Induktionsannahme), ist auch w mit allen Knoten verbunden (wir gelangen über die Kante (w, v) vom Knoten w zum Knoten v und von dort aus zu jedem anderen Knoten). Damit ist L_2 und E_0 auch nach dem Hinzufügen des nächsten Knotens zusammenhängend. So haben wir auch diese Behauptung bewiesen. □

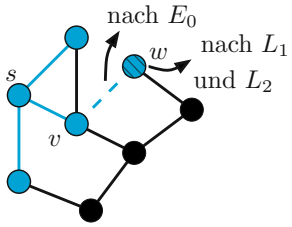

Abb. 2.8: Die blauen Knoten sind schon in L_1 und L_2 und die blauen Kanten in E_0 enthalten. Wird w über v erreicht, so wird die Kante (v, w) in E_0 und der Knoten w in die Listen L_1 und L_2 eingefügt.

Teil 3: Der Teilgraph (V, E_0) ist kreisfrei.

Beweis: Gäbe es einen Kreis, so müsste dieser irgendwann in Schritt 3 geschlossen werden. Die schließende Kante verbindet dann zwei Knoten, die schon in L_2 sind; in Schritt 3 werden aber nur Kanten zu E_0 hinzugefügt, bei denen ein Knoten zu diesem Zeitpunkt noch nicht in L_2 ist. □

Aus den drei bewiesenen Behauptungen folgt sofort die Aussage des Satzes. □

Eine einfache Variante des Algorithmus fügt die Knoten in Schritt 3 nicht ans Ende, sondern an den Anfang der Listen L_1 und L_2. Dieser Algorithmus wird auch als Tiefensuche bezeichnet.

Algorithmus 2.2 Tiefensuche (engl. *depth first search*, DFS)

Input: Zusammenhängender Graph $G = (V, E)$.

Output: Spannender Baum $T = (V, E_0)$ von G.

Schritt 1:
- Sei E_0 zunächst leer.
- Bereiten Sie zwei ebenfalls leere Knotenlisten L_1und L_2 vor.
- Wählen Sie einen beliebigen Startknoten $v_0 \in V$ und schreiben Sie v_0 in L_1 und L_2.

Schritt 2: Falls L_1 leer ist, dann STOPP und return $T = (V, E_0)$. Ansonsten nehmen Sie den ersten Knoten v aus L_1.

Schritt 3: Falls alle Nachbarknoten von v in L_2 sind, so löschen Sie v aus L_1. Ansonsten wählen Sie einen der Nachbarknoten w aus, der nicht in L_2 ist, schreiben Sie diesen an den Anfang der Listen L_1 und L_2 und fügen Sie die Kante (v, w) zu E_0 hinzu.

Schritt 4: Gehen Sie zu Schritt 2.

Können Sie sich vorstellen, wie man auf die beiden Namen, Breiten- und Tiefensuche, gekommen ist? Probieren Sie die Algorithmen an dem folgenden Graphen aus (Abb. 2.9). Starten Sie dabei mit dem Knoten v_0 und vergleichen Sie anschließend die berechneten spannenden Bäume miteinander.

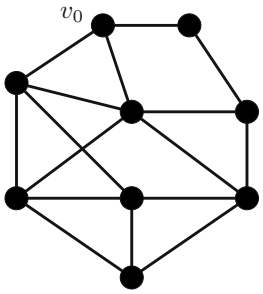

Abb. 2.9: Welche Bäume berechnen die beiden Algorithmen BFS und DFS?

2.3 Anwendungen von BFS und DFS

In Abschnitt 1.2 hatten wir schon gesagt, dass der Zusammenhang eines Graphen einen Schlüsselbegriff darstellt. Bisher hatten wir jedoch noch keine Methode kennen gelernt, um einen Graphen auf seinen Zusammenhang zu überprüfen bzw. die einzelnen Zusammenhangskomponenten zu bestimmen. Mit Hilfe der beiden Algorithmen BFS und DFS ist das leicht möglich. Wie könnte ein solcher Algorithmus aussehen? Versuchen Sie auch zu begründen, warum der von Ihnen vorgeschlagene Algorithmus die unterschiedlichen Zusammenhangskomponenten bestimmt?[7]

Aber nicht nur die einzelnen Zusammenhangskomponenten eines Graphen können durch spannende Bäume berechnet werden. Durch einen spannenden Baum, den wir aus der Breitensuche oder der Tiefensuche erhalten, kann auch überprüft werden, ob ein Graph bipartit ist oder nicht (Abschnitt 2.3.1). Außerdem haben die Bäume der Breitensuche eine ganz spezielle Eigenschaft: Wenn wir mit dem Knoten v_0 starten, so ist jeder Weg in dem Baum vom Knoten v_0 aus ein kürzester Weg bezüglich der Anzahl der Kanten (Abschnitt 2.3.2). Diese letzten beiden Anwendungen betrachten wir in den nächsten zwei Abschnitten genauer.

2.3.1 Erkennen von bipartiten Graphen

In Beispiel 1.1 hatten wir bipartite Graphen schon vorgestellt. Ein Graph $G = (V, E)$ heißt bipartit, wenn seine Knoten in zwei Teilmengen A und B zerlegt werden können, sodass nur Kanten zwischen den beiden Teilmengen und nicht innerhalb dieser verlaufen (Abb. 2.10).

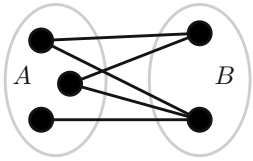

Abb. 2.10: Die Knotenmenge lässt sich in zwei Mengen zerlegen, wobei innerhalb der Mengen keine Kante verläuft.

Bipartite Graphen treten häufig bei der Darstellung von Zuordnungsproblemen auf. Ein typisches Zuordnungsproblem hat jeder Personaler zu bewältigen: Den freien Arbeitsplätzen in seinem Unternehmen muss er geeignete Kandidaten bzw. Bewerber zuordnen. Oder auch die Zuweisung von Räumen an der Universität zu den unterschiedlichen Veranstaltungen kann über einen bipartiten Graphen modelliert werden. Mit solchen Problemen werden wir uns später genauer beschäftigen (Kapitel 12.2). In diesem Abschnitt soll es darum gehen, wie man bei einem gegebenen Graphen feststellen kann, ob er bipartit ist oder nicht. Eine einfache Charakterisierung über die in einem bipartiten Graphen enthaltenden Kreise macht dies möglich.

Satz 2.3. *Ein Graph ist genau dann bipartit, wenn er keinen Kreis ungerader Länge enthält.*

Der Beweis ist konstruktiv, d.h. er zeigt nicht nur, dass die Aussage stimmt, sondern enthält auch die Methode, einen Kreis ungerader Länge zu finden, falls er in dem Graphen vorhanden ist. Diese Methode basiert auf der Berechnung eines spannenden Baums.

Beweis: Im Beweis gehen wir davon aus, dass der gegebene Graph G zusammenhängend ist. Wieso ist der Satz trotz dieser Einschränkung bewiesen?[8]

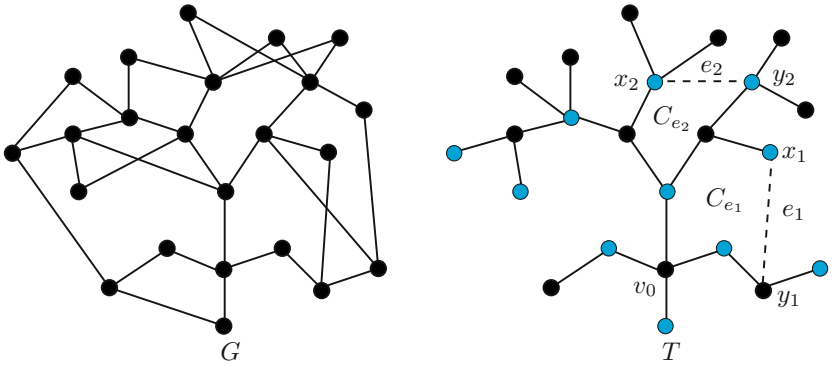

Abb. 2.11: Der Abstand zu r in T definiert eine bipartite Partition von G (blaue und schwarze Knoten). Jede Kante $e = (x, y) \in E(G) \backslash G(T)$ bildet einen Kreis C_e von gerader Länge. Also müssen sowohl x_1 und y_1 als auch x_2 und y_2 in unterschiedlichen Komponenten liegen.

„⇐“: Wir zeigen als Erstes, dass ein zusammenhängender Graph G, der keinen Kreis ungerader Länge enthält, bipartit ist. Dazu teilen wir die Knoten V im Folgenden in zwei Mengen A und B auf, sodass zwischen den beiden Mengen keine Kante verläuft. Für die Aufteilung benötigen wir einen spannenden Baum. Sei $T \subseteq G$ ein spannender Baum und v_0 ein beliebiger Knoten aus G. Dann definieren wir die Menge A als die Menge aller Knoten, die mit v_0 durch einen Weg ungerader Länge verbunden sind, und mit B die Menge aller Knoten, die mit v_0 durch einen Weg gerader Länge verbunden sind (Abb. 2.11). Diese Definition ist sinnvoll, da in T ein eindeutig bestimmter Weg von v_0 zu jedem Knoten existiert. Wir müssen nun zeigen, dass diese Aufteilung bipartit ist, d.h. dass keine Kante von G zwei Knoten aus A oder zwei Knoten aus B verbindet.
Sei $e = (x, y)$ eine beliebige Kante von G. Wir betrachten zwei Fälle: *Fall 1*: Die Kante e ist eine Baumkante des spannenden Baums. Dann liegen x und y nach der Definition von A und B in unterschiedlichen Mengen. *Fall 2*: Die Kante e ist eine Nichtbaumkante. Nehmen wir an, sie verbindet zwei Knoten aus derselben Menge, wie z.B. die Kante e_2 aus der Abbildung 2.11. Wieso kann diese Kante e_2 nicht im ursprünglichen Graphen G (der nur Kreise gerader Länge enthält) existiert haben?[9] Damit haben wir bewiesen, dass G bipartit ist.