

Elijah Meeks



D3.js

w akcji

HelionTV

Tytuł oryginału: D3.js in Action

Tłumaczenie: Tomasz Walczak

Projekt okładki: Studio Gravite / Olsztyn; Obarek, Pokoński, Pazdrijowski, Zaprucki
Materiały graficzne na okładce zostały wykorzystane za zgodą Shutterstock Images LLC.

ISBN: 978-83-283-1823-6

Original edition copyright © 2015 by Manning Publications Co.
All rights reserved.

Polish edition copyright © 2016 by HELION SA.
All rights reserved.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/d3jsak>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:

<ftp://ftp.helion.pl/przyklady/d3jsak.zip>

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

<i>Przedmowa</i>	9
<i>Podziękowania</i>	11
<i>O książce</i>	13

CZĘŚĆ I. PODSTAWY BIBLIOTEKI D3.JS 17

Rozdział 1. Wprowadzenie do biblioteki D3.js 19

1.1.	Czym jest D3.js?	20
1.2.	Jak działa biblioteka D3?	21
1.2.1.	W wizualizacji danych ważne są nie tylko aspekty wizualne	22
1.2.2.	W bibliotece D3 istotne są selekcja i wiązanie danych	26
1.2.3.	Biblioteka D3 umożliwia określanie wyglądu elementów stron internetowych na podstawie powiązanych danych	27
1.2.4.	Elementami strony mogą być elementy div, państwa lub diagramy przepływu	27
1.3.	Stosowanie standardu HTML5	28
1.3.1.	Model DOM	29
1.3.2.	Pisanie kodu w konsoli	34
1.3.3.	SVG	34
1.3.4.	Style CSS	39
1.3.5.	JavaScript	44
1.4.	Standardy dotyczące danych	49
1.4.1.	Dane tabelaryczne	50
1.4.2.	Dane zagnieżdżone	50
1.4.3.	Dane sieciowe	50
1.4.4.	Dane geograficzne	51
1.4.5.	Dane surowe	51
1.4.6.	Obiekty	53
1.5.	Standardy z obszaru wizualizowania informacji stosowane w bibliotece D3	54
1.6.	Twoja pierwsza aplikacja oparta na bibliotece D3	55
1.6.1.	„Witaj, świecie” z elementami <div>	56
1.6.2.	„Witaj, świecie” z kołami	56
1.6.3.	Komunikowanie się z biblioteką D3	58
1.7.	Podsumowanie	61

Rozdział 2. Przepływ danych podczas wizualizowania informacji 63

- 2.1. Praca z danymi 64
 - 2.1.1. Wczytywanie danych 64
 - 2.1.2. Formatowanie danych 67
 - 2.1.3. Przekształcanie danych 69
 - 2.1.4. Pomiar danych 73
- 2.2. Wiązanie danych 74
 - 2.2.1. Selekcje i wiązanie 74
 - 2.2.2. Dostęp do danych za pomocą funkcji wewnętrznych 76
 - 2.2.3. Uwzględnianie skal 79
- 2.3. Styl, atrybuty i treść w prezentacji danych 81
 - 2.3.1. Wizualizacja oparta na wczytanych danych 82
 - 2.3.2. Ustawianie kanałów 84
 - 2.3.3. Instrukcje enter, update i exit 87
- 2.4. Podsumowanie 93

Rozdział 3. Projektowanie sterowane danymi i interakcje 95

- 3.1. Architektura projektu 96
 - 3.1.1. Dane 96
 - 3.1.2. Zasoby 97
 - 3.1.3. Rysunki 97
 - 3.1.4. Arkusze stylów 97
 - 3.1.5. Biblioteki zewnętrzne 98
- 3.2. Interaktywne style i model DOM 100
 - 3.2.1. Zdarzenia 100
 - 3.2.2. Przejścia graficzne 103
 - 3.2.3. Manipulowanie modelem DOM 105
 - 3.2.4. Sensowne korzystanie z kolorów 107
- 3.3. Wstępnie generowane treści 113
 - 3.3.1. Rysunki 113
 - 3.3.2. Fragmenty kodu w HTML-u 115
 - 3.3.3. Wstępnie wygenerowana grafika SVG 116
- 3.4. Podsumowanie 122

CZĘŚĆ II. PODSTAWY WIZUALIZOWANIA INFORMACJI 123**Rozdział 4. Komponenty wykresów 125**

- 4.1. Ogólne zasady tworzenia wykresów 126
 - 4.1.1. Generatory 127
 - 4.1.2. Komponenty 127
 - 4.1.3. Układy 127
- 4.2. Tworzenie osi 128
 - 4.2.1. Wyświetlanie danych 128
 - 4.2.2. Określanie stylu osi 131
- 4.3. Złożone obiekty graficzne 135

- 4.4. Wykresy liniowe i interpolacja 143
 - 4.4.1. Rysowanie linii od określonych punktów 145
 - 4.4.2. Rysowanie wielu linii za pomocą kilku generatorów 147
 - 4.4.3. Omówienie interpolacji linii 148
- 4.5. Złożone akcesory 149
- 4.6. Podsumowanie 158

Rozdział 5. Układy 159

- 5.1. Histogramy 160
- 5.2. Wykresy kołowe 162
 - 5.2.1. Rysowanie wykresu kołowego 163
 - 5.2.2. Tworzenie wykresu pierścieniowego 165
 - 5.2.3. Przejścia 166
- 5.3. Układy dla grup kół 168
- 5.4. Drzewa 172
- 5.5. Układy skumulowane 177
- 5.6. Wtyczki służące do dodawania układów 183
 - 5.6.1. Diagram Sankeya 183
 - 5.6.2. Chmury słów 190
- 5.7. Podsumowanie 195

Rozdział 6. Wizualizowanie sieci 197

- 6.1. Statyczne diagramy sieci 198
 - 6.1.1. Dane o sieci 199
 - 6.1.2. Macierz sąsiedztwa 201
 - 6.1.3. Diagram łukowy 205
- 6.2. Układ oparty na siłach 208
 - 6.2.1. Tworzenie dla sieci diagramu opartego na siłach 209
 - 6.2.2. Znaczniki SVG 210
 - 6.2.3. Miary sieci 214
 - 6.2.4. Ustawienia układu opartego na siłach 216
 - 6.2.5. Aktualizowanie sieci 218
 - 6.2.6. Usuwanie i dodawanie węzłów oraz krawędzi 219
 - 6.2.7. Ręczne określanie pozycji węzłów 223
 - 6.2.8. Optymalizacja 226
- 6.3. Podsumowanie 226

Rozdział 7. Wizualizowanie informacji geoprzestrzennych 227

- 7.1. Podstawy tworzenia map 229
 - 7.1.1. Szukanie danych 230
 - 7.1.2. Rysowanie punktów na mapie 234
 - 7.1.3. Odwzorowania i obszary 236
 - 7.1.4. Interaktywność 238
- 7.2. Tworzenie lepszych map 239
 - 7.2.1. Siatka kartograficzna 240
 - 7.2.2. Operacja zoom 241
- 7.3. Zaawansowane aspekty map 244
 - 7.3.1. Tworzenie i obracanie globusa 244
 - 7.3.2. Odwzorowanie satelitarne 250

- 7.4. Dane i funkcje w bibliotece TopoJSON 251
 - 7.4.1. Format plików TopoJSON 251
 - 7.4.2. Wyświetlanie danych w formacie TopoJSON 252
 - 7.4.3. Scalanie 253
 - 7.4.4. Sąsiednie obiekty 255
- 7.5. Tworzenie map z kafelkami za pomocą instrukcji d3.geo.tile 256
- 7.6. Dalsza lektura związana z mapami 261
 - 7.6.1. Zoom dla transformacji 262
 - 7.6.2. Rysowanie na płótnie 262
 - 7.6.3. Zmiana odwzorowania dla danych rastrowych 262
 - 7.6.4. Technika hexbinning 262
 - 7.6.5. Diagramy Woronoja 262
 - 7.6.6. Mapy anamorficzne 262
- 7.7. Podsumowanie 263

Rozdział 8. Manipulowanie tradycyjnym modelem DOM za pomocą biblioteki D3 265

- 8.1. Przygotowania 267
 - 8.1.1. Style CSS 267
 - 8.1.2. Kod w HTML-u 268
- 8.2. Arkusz kalkulacyjny 268
 - 8.2.1. Tworzenie arkusza kalkulacyjnego za pomocą tabeli 268
 - 8.2.2. Tworzenie arkusza kalkulacyjnego za pomocą elementów `<div>` 270
 - 8.2.3. Dodawanie animacji do arkusza kalkulacyjnego 272
- 8.3. Płótno 273
 - 8.3.1. Rysowanie na płótnie 275
 - 8.3.2. Rysowanie i zapisywanie wielu obrazków 275
- 8.4. Galeria rysunków 277
 - 8.4.1. Interaktywne wyróżnianie elementów modelu DOM 279
 - 8.4.2. Selekcja elementów 281
- 8.5. Podsumowanie 283

CZĘŚĆ III. TECHNIKI ZAAWANSOWANE 285

Rozdział 9. Łączenie komponentów aplikacji interaktywnych 287

- 9.1. Jedno źródło danych, wiele perspektyw 289
 - 9.1.1. Podstawy tworzenia panelu kontrolnego dla danych 291
 - 9.1.2. Arkusz kalkulacyjny 292
 - 9.1.3. Wykres słupkowy 293
 - 9.1.4. Grupy kół 293
 - 9.1.5. Zmiana wielkości wykresów na podstawie rozmiaru ekranu 294
- 9.2. Interaktywność — zdarzenia związane z kursorem myszy 296
- 9.3. Kontrolka wyboru zakresu 299
 - 9.3.1. Tworzenie kontrolki wyboru zakresu 300
 - 9.3.2. Ułatwianie korzystania z kontrolki wyboru zakresu 303
 - 9.3.3. Zdarzenia kontrolki wyboru zakresu 306
 - 9.3.4. Ponowne rysowanie komponentów 307
- 9.4. Podsumowanie 308

Rozdział 10. Tworzenie układów i komponentów 311

- 10.1. Tworzenie układu 312
- 10.2. Pisanie własnych komponentów 319
 - 10.2.1. Wczytywanie przykładowych danych 320
 - 10.2.2. Wizualizacja komponentów ze skalami 322
 - 10.2.3. Dodawanie etykiet do komponentu 327
- 10.3. Podsumowanie 330

Rozdział 11. Wizualizowanie dużych zbiorów danych 331

- 11.1. Duże zbiory danych geograficznych 332
 - 11.1.1. Generowanie losowych danych geograficznych 334
 - 11.1.2. Rysowanie danych geograficznych na płótnie 336
 - 11.1.3. Techniki wyświetlania elementów w trybie mieszanym 338
- 11.2. Duże zbiory danych o sieciach 344
- 11.3. Optymalizowanie wybierania danych na podstawie współrzędnych x i y za pomocą drzew czwórkowych 349
 - 11.3.1. Generowanie losowych danych o współrzędnych x i y 349
 - 11.3.2. Wybór zakresu na podstawie współrzędnych x i y 349
- 11.4. Inne techniki optymalizacji 354
 - 11.4.1. Unikanie ustawiania ogólnego stylu opacyty 354
 - 11.4.2. Unikanie ogólnych selekcji 355
 - 11.4.3. Wstępne wyznaczanie pozycji 355
- 11.5. Podsumowanie 356

Rozdział 12. Biblioteka D3 i urządzenia przenośne 357

- 12.1. Zasady tworzenia projektów sterowanych danymi dla urządzeń przenośnych 359
- 12.2. Wizualizacja i dotyk 359
 - 12.2.1. Funkcja `d3.touches` 361
 - 12.2.2. Rejestrowanie na liście zdarzeń związanych z dotykiem 361
 - 12.2.3. Wizualizowanie zdarzeń związanych z dotykiem 362
 - 12.2.4. Przesuwanie za pomocą jednego przeciągnięcia palcem 363
 - 12.2.5. Wizualizowanie analizy dotknięć 365
 - 12.2.6. Zoom oparty na szczygnięciach 367
 - 12.2.7. Rotacja z użyciem trzech palców 370
 - 12.2.8. Łączenie wszystkich elementów 371
- 12.3. Dostosowujące się wizualizacje danych 374
 - 12.3.1. Tworzenie dostosowujących się wizualizacji danych 375
 - 12.3.2. Tworzenie podstawowej wersji aplikacji 376
 - 12.3.3. Rozwiązywanie na skalę tabletu 380
 - 12.3.4. Rozwiązanie dostosowane do telefonu 384
 - 12.3.5. Automatyczne wykrywanie ekranów o różnych rozmiarach 389
 - 12.3.6. Ogólne zasady tworzenia dostosowujących się wizualizacji danych 389
- 12.4. Geolokalizacja 389
- 12.5. Podsumowanie 390

Skorowidz 393

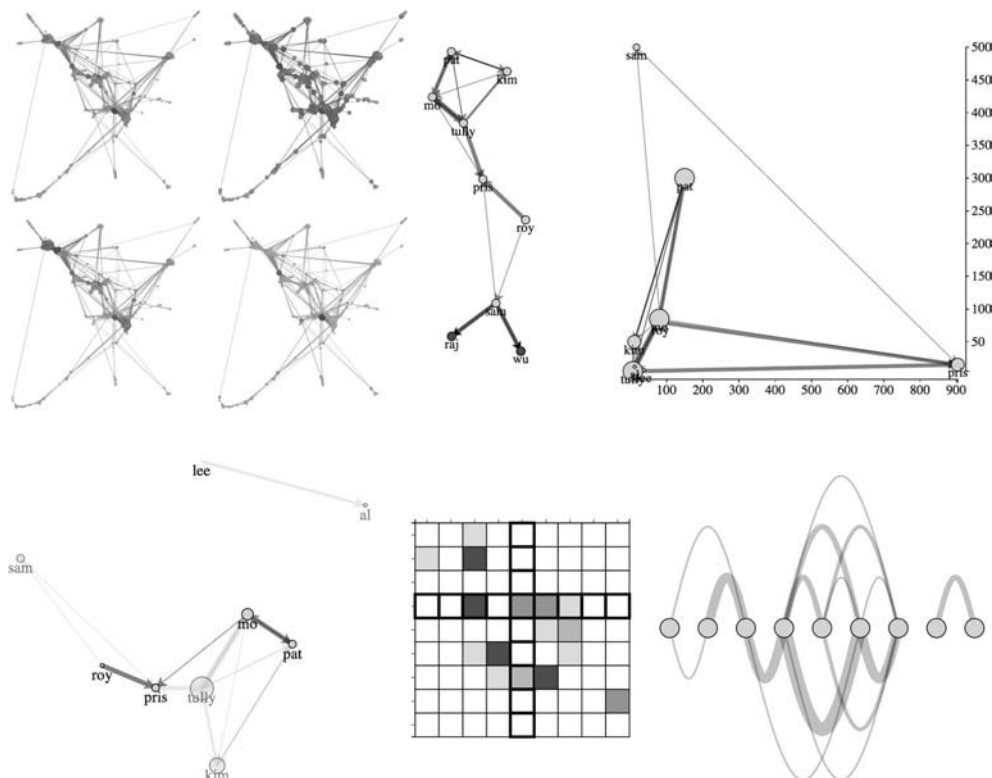
Wizualizowanie sieci

Zawartość rozdziału:

- Tworzenie macierzy sąsiedztwa i diagramów łukowych
- Używanie układu opartego na siłach
- Przedstawianie kierunków
- Dodawanie i usuwanie węzłów oraz krawędzi w sieciach

Analizowanie i wizualizowanie sieci to zadania wykonywane coraz częściej. Powodem jest rozrastanie się internetowych sieci społecznościowych, takich jak Twitter i Facebook, a także popularność mediów społecznościowych i powiązanych danych w tak zwanej sieci Web 2.0. Wizualizacje sieci, z którymi zetkniesz się w tym rozdziale (przykłady znajdziesz na rysunku 6.1), są interesujące przede wszystkim dlatego, że dotyczą głównie powiązań między elementami. Dlatego reprezentują systemy bardziej precyzyjnie niż tradycyjne „płaskie” dane przedstawiane w standardowych wizualizacjach danych.

Ten rozdział dotyczy reprezentowania sieci. Dlatego ważne jest, by zrozumieć związaną z nimi terminologię. W kontekście sieci powiązane ze sobą obiekty (na przykład ludzie) są nazywane *węzłami*, a połączenia między nimi (na przykład między znajomymi na Facebooku) są określane jako *krawędzie*. Czasem węzły nazywa się też *wierzchołkami*, ponieważ to w nich łączą się krawędzie. Choć zrozumiałe jest, że rysunki z opisanymi węzłami i krawędziami są przydatne, to jedna z lekcji płynących z tego rozdziału jest taka, że nie istnieje jeden, najlepszy sposób przedstawiania sieci. Sieci czasem nazywane są też *grafami*, ponieważ to określenie stosowane jest w matematyce. Znaczenie węzła w sieci określa się na podstawie *centralności*. To oczywiście nie wszystkie pojęcia, ale podany zestaw wystarczy do rozpoczęcia pracy.



Rysunek 6.1. Oprócz omówienia podstaw analizy sieci (punkt 6.2.3) rozdział 6. obejmuje opisy tworzenia sieci na podstawie współrzędnych x i y (punkt 6.2.5), algorytmów opartych na siłach (podrozdział 6.2), macierzy sąsiedztwa (punkt 6.1.2) i diagramów łukowych (punkt 6.1.3)

Z sieciami związany jest nie tylko format danych, ale też sposób spojrzenia na nie. Gdy przetwarzasz dane o sieci, zwykle starasz się wykryć i wyświetlić wzorce występujące w danej sieci lub jej części. Poszczególne węzły są mniej istotne. Choć można tworzyć wizualizacje sieci, aby otrzymać atrakcyjny graficzny indeks (na przykład mapę myśli lub mapę witryny), to typowe techniki wizualizowania sieci służą do prezentowania jej struktury, a nie pojedynczych węzłów.

6.1. Statyczne diagramy sieci

Dane o sieciach różnią się od danych hierarchicznych. W sieciach mogą występować połączenia wiele do wielu, tak jak na diagramie Sankeya z rozdziału 5., natomiast w danych hierarchicznych węzeł może mieć wiele elementów podrzędnych, ale tylko jeden element nadrzędny (tak jak w układach tree i pack z rozdziału 5.). Sieć nie musi być siecią społecznościową. Ten format może reprezentować wiele różnych struktur — na przykład sieci transportowe lub powiązane dane bez narzuconej struktury. W tym rozdziale poznasz cztery popularne sposoby reprezentowania sieci — jako danych, za pomocą macierzy sąsiedztwa, przy użyciu diagramów łukowych i na diagramach opartych na siłach.

Każdy z tych sposobów związany jest z inną reprezentacją graficzną. Na przykład w układzie opartym na siłach węzły są przedstawiane jako koła, a krawędzie — jako linie. W macierzach sąsiedztwa węzły są rozmieszczane według współrzędnych x i y , a krawędzie to wypełnione kwadraty. Nie istnieje domyślna reprezentacja sieci; przykłady przedstawione w tym rozdziale ilustrują najczęściej stosowane techniki.

6.1.1. Dane o sieci

Choć dane o sieci można przechowywać w różnych formatach, najczęściej używa się *list krawędzi*. Lista krawędzi jest zwykle zapisana w pliku CSV, takim jak na listingu 6.1. Taki plik obejmuje kolumny węzłów źródłowych (ang. *source*) i docelowych (ang. *target*) oraz łańcuchy znaków lub liczby określające, które węzły są powiązane ze sobą. Każda krawędź może mieć też inne atrybuty, określające typ połączenia lub jego siłę, czas, przez jaki połączenie jest aktywne, kolor, a także inne niezbędne informacje. Ważne jest to, że potrzebne są tylko kolumny ze źródłowymi i docelowymi węzłami.

Listing 6.1. Plik edgelist.csv

```
source,target,weight
Maja,Piotr,1
Jan,Piotr,5
Jan,Maja,1
Leon,Piotr,5
Leon,Iga,3
Leon,Ada,1
Leon,Adam,3
Iga,Ada,2
Iga,Adam,1
Adam,Leon,7
Adam,Ada,1
Adam,Piotr,1
Ada,Leon,1
Ada,Iga,2
Ada,Adam,5
Igor,Alan,3
```

W sieciach skierowanych kolumny z węzłami źródłowymi i docelowymi określają kierunek połączenia między węzłami. W sieci skierowanej węzły mogą być połączone ze sobą w jednym kierunku, ale już niekoniecznie w drugim. Możliwe, że obserwujesz innego użytkownika na Twitterze, ale on nie musi obserwować Ciebie. W sieciach nieskierowanych zwykle też używane są kolumny z węzłami źródłowymi i docelowymi, ale połączenie działa tak samo w obu kierunkach. Pomyśl o sieci obejmującej połączenia między osobami, które uczestniczyły w tych samych zajęciach. Jeśli chodziłem z Tobą na te same zajęcia, Ty chodziłeś na nie ze mną. W tym rozdziale zetkniesz się z sieciami skierowanymi i sieciami z wagami.

W pliku znajduje się też kolumna *weight* określająca siłę połączeń. Tu lista krawędzi określa, ile razy osoba z kolumny *source* dodała do ulubionych tweety osoby z kolumny *target*. Maja polubiła jeden tweet Piotra, Jan dodał do ulubionych pięć tweetów Piotra itd. Jest to więc *sieć z wagami*, ponieważ krawędzie mają określone wagi. Jest

to też *sieć skierowana*, ponieważ krawędzie mają kierunek. To oznacza, że używana jest *sieć skierowana z wagami*, dlatego w wizualizacjach trzeba uwzględnić zarówno wagi, jak i kierunek.

Pod względem technicznym do utworzenia sieci potrzebna jest tylko lista krawędzi, ponieważ listę węzłów można uzyskać na podstawie unikatowych wartości z listy krawędzi. Takie zadanie wykonują standardowe pakiety do analiz sieci (na przykład narzędzie Gephi). Choć można uzyskać listę węzłów w JavaScriptcie, częściej tworzy się listę węzłów z dodatkowymi informacjami na ich temat. Taką listę przedstawia poniższy listing 6.2.

Listing 6.2. Plik `nodelist.csv`

```
id,followers,following
Maja,17,500
Jan,83,80
Piotr,904,15
Leon,7,5
Iga,11,50
Adam,80,85
Ada,150,300
Igor,38,7
Alan,12,12
```

Ponieważ dane dotyczą użytkowników Twittera, na podstawie ich statystyk dostępne są dodatkowe informacje. Tu używana jest liczba obserwujących (*followers*) i obserwowanych (*following*) osób. Taka lista nie musi zawierać nic poza identyfikatorem. Jednak dostęp do dodatkowych danych pozwala zmodyfikować wizualizację i uwzględnić atrybuty węzłów.

Reprezentacja sieci zależy od jej wielkości i charakteru. Jeśli sieć nie reprezentuje połączeń między podobnymi obiektami, ale przepływ towarów, informacji lub ruchu, można wykorzystać diagram Sankeya (tak jak w rozdziale 5.). Pamiętaj, że format danych dla diagramów Sankeya jest identyczny jak w tym przykładzie — potrzebne są tablica węzłów i tablica krawędzi. Diagram Sankeya nadaje się tylko do określonych rodzajów danych o sieci. Wykresy innych typów, na przykład macierze sąsiedztwa, są bardziej uniwersalnym narzędzie do prezentowania danych o sieciach.

Zanim zaczniesz pisać kod do tworzenia wizualizacji sieci, przygotuj arkusz stylów CSS, co pozwoli określać kolory na podstawie klas i ograniczyć ilość stylów wewnątrz-wierszowych. Listing 6.3 zawiera kod stylów CSS potrzebny we wszystkich przykładach z tego rozdziału. Pamiętaj, że style wewnątrz-wierszowe i tak będą potrzebne przy ustawianiu wartości liczbowych dla atrybutów elementów graficznych na podstawie danych — na przykład do ustawiania atrybutu `stroke-width` linii na podstawie siły połączenia.

Listing 6.3. Plik `networks.css`

```
.grid {
  stroke: black;
  stroke-width: 1px;
  fill: red;
```

```

}
.arc {
  stroke: black;
  fill: none;
}
.node {
  fill: lightgray;
  stroke: black;
  stroke-width: 1px;
}
circle.active {
  fill: red;
}
path.active {
  stroke: red;
}

```

6.1.2. Macierz sąsiedztwa

Gdy zetkniesz się z większą liczbą reprezentowanych graficznie sieci, może Ci się wydać, że jedynym sposobem ich wizualizowania jest używanie kół lub kwadratów dla węzłów oraz linii (prostych lub krzywych) dla krawędzi. Może być dla Ciebie zaskoczeniem to, że w jednej z najbardziej skutecznych wizualizacji sieci nie występują w ogóle linie. W *macierzy sąsiedztwa* połączenia między węzłami są obrazowane za pomocą siatki.

Zasady tworzenia macierzy sąsiedztwa są proste. Należy rozmieścić węzły na osiach x i y, a następnie wypełnić pola na przecięciach tych węzłów, które są ze sobą powiązane. Pola dla niepowiązanych węzłów pozostają puste. Ponieważ przykładowe dane dotyczą sieci skierowanej, węzły na osi y można uznać za źródłowe, a węzły na osi x — za docelowe. Takie rozwiązanie zastosowano kilka stron dalej. Ponadto ponieważ używana jest sieć z wagami, do ich określania posłuży jasność. Jaśniejsze kolory będą oznaczały słabsze połączenia, a ciemniejsze — silniejsze.

Jedyny problem z budowaniem macierzy sąsiedztwa w bibliotece D3 polega na tym, że nie istnieje gotowy układ. To oznacza, że wykres trzeba zbudować ręcznie — tak jak wcześniej wykres słupkowy, punktowy lub pudełkowy. Mike Bostock opracował bardzo efektowny przykład (<http://bost.ocks.org/mike/miserables/>). Możesz jednak przygotować działający wykres bez pisania dużej ilości kodu. Ilustruje to funkcja z listingu 6.4. Kod musi jednak przetwarzać dwie tablice obiektów JSON generowane na podstawie pliku CSV i sformatować dane w taki sposób, by możliwa była łatwa praca z nimi. Podobne zadania trzeba wykonać podczas pisania własnego układu, co zrobisz w rozdziale 10. Zwykle jest to dobre rozwiązanie.

Listing 6.4. Funkcja generująca macierz sąsiedztwa

```

function adjacency() {
  queue()
    .defer(d3.csv, "nodelist.csv")
    .defer(d3.csv, "edgelist.csv")
    .await(function(error, file1, file2) {
      createAdjacencyMatrix(file1, file2);
    });
}

```

← Najpierw trzeba wczytać dwa zbiory danych. Kolejka umożliwia tu wczytanie plików przed przejściem do dalszych zadań

```

});

function createAdjacencyMatrix(nodes, edges) {
  var edgeHash = {};
  for (x in edges) {
    var id = edges[x].source + "-" + edges[x].target;
    edgeHash[id] = edges[x];
  };
  matrix = [];
  for (a in nodes) {
    for (b in nodes) {
      var grid = {
        id: nodes[a].id + "-" + nodes[b].id,
        x: b, y: a, weight: 0;
        if (edgeHash[grid.id]) {
          grid.weight = edgeHash[grid.id].weight;
        };
        matrix.push(grid);
      };
    };
  };

  d3.select("svg")
    .append("g")
    .attr("transform", "translate(50,50)")
    .attr("id", "adjacencyG")
    .selectAll("rect")
    .data(matrix)
    .enter()
    .append("rect")
    .attr("class", "grid")
    .attr("width", 25)
    .attr("height", 25)
    .attr("x", function (d) {return d.x * 25})
    .attr("y", function (d) {return d.y * 25})
    .style("fill-opacity", function (d) {return d.weight * .2;});

  var scaleSize = nodes.length * 25;
  var nameScale = d3.scale.ordinal()
    .domain(nodes.map(function (e) {return e.id}))
    .rangePoints([0, scaleSize], 1);

  var xAxis = d3.svg.axis()
    .scale(nameScale).orient("top").tickSize(4);
  var yAxis = d3.svg.axis()
    .scale(nameScale).orient("left").tickSize(4);
  d3.select("#adjacencyG").append("g").call(yAxis);
  d3.select("#adjacencyG").append("g").call(xAxis)
    .selectAll("text")
    .style("text-anchor", "end")
    .attr("transform", "translate(-10,-10) rotate(90)");
};

```

← Skróót pozwala sprawdzać, czy istnieje krawędź między podanymi węzłami źródłowym i docelowym

← Tworzy wszystkie możliwe powiązania węzłów źródłowych z docelowymi

← Ustawia współrzędne x i y na podstawie pozycji węzłów źródłowego i docelowego w tablicy

← Jeśli dana krawędź istnieje na liście krawędzi, należy przypisać jej odpowiednią wagę

← Tworzy skalę porządkową na podstawie identyfikatorów węzłów

← Używane dla skali porządkowej

← Dla obu osi używana jest ta sama skala

← Rotuje tekst na osi y

Zastosowano tu kilka nowych technik. Użyto nowej skali, `d3.scale.ordinal`, przyjmującej tablicę różnych wartości i umożliwiającej rozmieszczenie ich na osi w taki sposób, jak zrobiono to z nazwami węzłów w tym przykładzie. Konieczne jest zastoso-

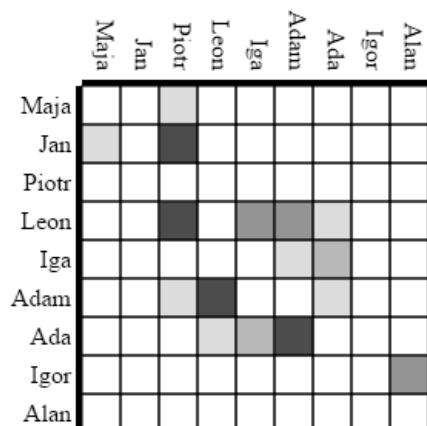
wanie nieomawianej wcześniej funkcji dla skal, `rangePoints`, która tworzy zestaw kubelków z wartościami wyświetlanych na osi lub w inny sposób. Funkcja wiąże każdą unikatową wartość z liczbą z podanego przedziału. Dla każdego punktu można też zadeklarować w drugiej, opcjonalnej zmiennej przesunięcie. Inny nowy fragment kodu wykorzystuje bibliotekę `queue.js`, potrzebną, ponieważ kod wczytuje dwa pliki CSV, a funkcję należy uruchomić dopiero po ich załadowaniu. Kod generuje reprezentującą macierz tablicę obiektów, która może wydawać się niezrozumiała. Jeśli jednak przyjrzyj się jej w konsoli, zobaczysz — co pokazuje rysunek 6.2 — że generowana jest lista wszystkich możliwych połączeń i ich sił.

```
< [▼Object 3, ▼Object 3, ▼Object 3,
  id: "Maja-Maja"    id: "Maja-Jan"    id: "Maja-Piotr"
  weight: 0          weight: 0          weight: "1"
  x: "0"             x: "1"             x: "2"
  y: "0"             y: "0"             y: "0"
  __proto__: Object  __proto__: Object  __proto__: Object
▼Object 3, ▼Object 3, ▼Object 3,
  id: "Maja-Leon"    id: "Maja-Iga"    id: "Maja-Adam"
  weight: 0          weight: 0          weight: 0
  x: "3"             x: "4"             x: "5"
  y: "0"             y: "0"             y: "0"
  __proto__: Object  __proto__: Object  __proto__: Object
▼Object 3, ▼Object 3, ▼Object 3,
  id: "Maja-Ada"     id: "Maja-Igor"   id: "Maja-Alan"
  weight: 0          weight: 0          weight: 0
  x: "6"             x: "7"            x: "8"
  y: "0"             y: "0"            y: "0"
  __proto__: Object  __proto__: Object  __proto__: Object
▼Object 3, ▼Object 3, ▼Object 3,
  id: "Jan-Maja"     id: "Jan-Jan"     id: "Jan-Piotr"
  weight: "1"        weight: 0          weight: "5"
  x: "0"             x: "1"            x: "2"
  y: "1"             y: "1"            y: "1"
  __proto__: Object  __proto__: Object  __proto__: Object
▼Object 3, ▼Object 3, ▼Object 3,
  id: "Jan-Leon"     id: "Jan-Iga"     id: "Jan-Adam"
  weight: 0          weight: 0          weight: 0
  x: "3"             x: "4"            x: "5"
  y: "1"             y: "1"            y: "1"
  __proto__: Object  __proto__: Object  __proto__: Object
  ▶Object, ▶Object, ▶Object, ▶Object, ▶Object, ▶Object, ▶Object,
  ▶Object, ▶Object, ▶Object, ▶Object, ▶Object, ▶Object, ▶Object,
```

Rysunek 6.2. Budowana tablica powiązań. Zauważ, że w tablicy zapisywane są wszystkie możliwe powiązania. Jednak tylko połączenia istniejące w zbiorze danych mają wagę większą niż 0. Zwróć ponadto uwagę, że w wyniku importu danych z pliku CSV wagi są zapisywane jako łańcuchy znaków

Na rysunku 6.3 pokazana została uzyskana macierz sąsiedztwa oparta na listach węzłów i krawędzi.

W wielu macierzach sąsiedztwa zauważysz, że kwadrat reprezentujący połączenie węzła z samym sobą jest zawsze wypełniony. W słownictwie związanym z sieciami oznacza to *pętlę własną*. Zjawisko to zachodzi, gdy węzeł jest powiązany sam ze sobą. W przykładzie taka pętla oznaczałaby, że ktoś dodał własny tweet do ulubionych. Na szczęście w używanym zbiorze danych nikt nie jest na tyle zdesperowany, by to zrobić.



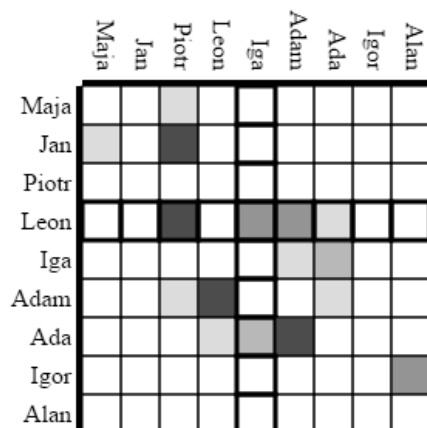
Rysunek 6.3. Macierz sąsiedztwa dla skierowanej sieci z wagami. Jaśniejszy kolor oznacza słabsze powiązanie, a ciemniejszy — silniejsze. Węzły źródłowe są zapisane na osi y, a docelowe — na osi x. Macierz pokazuje, że Jan dodał do ulubionych tweety Mai, natomiast Maja nie polubiła żadnych tweetów Jana

Jeśli chcesz, możesz dodać mechanizmy interaktywne, by zwiększyć czytelność macierzy. Siatki są czasem mało czytelne, gdy nie można wyróżnić wiersza i kolumny dla danego kwadratu. Skonfigurowanie wyróżniania w macierzy jest łatwe. Wystarczy dodać odbiornik zdarzenia `mouseover`, który uruchamia funkcję `gridOver` wyróżniającą wszystkie kwadraty o tej samej współrzędnej x lub y:

```
d3.selectAll("rect.grid").on("mouseover", gridOver);

function gridOver(d,i) {
  d3.selectAll("rect").style("stroke-width", function (p) {
    return p.x == d.x || p.y == d.y ? "3px" : "1px"});
};
```

Na rysunku 6.4 widać, że teraz przeniesienie kursora nad kwadrat siatki powoduje wyróżnienie wiersza i kolumny z tym kwadratem.



Rysunek 6.4. Macierz sąsiedztwa z wyróżnionymi kolumną i wierszem kwadratu z siatki. W tym przykładzie kursor znajduje się nad krawędzią łączącą Leona z Igą. Widać, że Leon polubił tweety czterech osób, spośród których jedną była Iga, natomiast tweety Igi zostały dodane do ulubionych tylko przez jeszcze jedną osobę, Adę (<http://bl.ocks.org/emeeeks/391e2cf83a0708c19f8c>)

6.1.3. Diagram łukowy

Do graficznego przedstawiania sieci można też wykorzystać diagramy łukowe. W takim diagramie węzły są rozmieszczone na linii i połączone łukami widocznymi nad linią lub pod nią. Nie istnieje układ dla diagramów łukowych, a przykładów dostępnych jest jeszcze mniej niż dla macierzy sąsiedztwa. Gdy już jednak zapoznasz się z kodem, zobaczysz, że zasady tworzenia takich diagramów są proste. Utwórz więc teraz jeszcze jeden pseudoukład (podobnie jak dla macierzy sąsiedztwa). Tym razem trzeba przetwarzać nie tylko krawędzie, ale też węzły. Potrzebny kod znajdziesz na listingu 6.5.

Listing 6.5. Kod generujący diagram łukowy

```
function arcDiagram() {
  queue()
    .defer(d3.csv, "odelist.csv")
    .defer(d3.csv, "edgelist.csv")
    .await(function(error, file1, file2) {
      createArcDiagram(file1, file2);
    });
}

function createArcDiagram(nodes, edges) {

  var nodeHash = {};
  for (x in nodes) {
    nodeHash[nodes[x].id] = nodes[x];
    nodes[x].x = parseInt(x) * 40;
  };

  for (x in edges) {
    edges[x].weight = parseInt(edges[x].weight);
    edges[x].source = nodeHash[edges[x].source];
    edges[x].target = nodeHash[edges[x].target];
  };

  linkScale = d3.scale.linear()
    .domain(d3.extent(edges, function (d) {return d.weight}))
    .range([5,10])

  var arcG = d3.select("svg").append("g").attr("id", "arcG")
    .attr("transform", "translate(50,250)");

  arcG.selectAll("path")
    .data(edges)
    .enter()
    .append("path")
    .attr("class", "arc")
    .style("stroke-width", function(d) {return d.weight * 2;})
    .style("opacity", .25)
    .attr("d", arc)

  arcG.selectAll("circle")
    .data(nodes)
    .enter()
    .append("circle")
```

Tworzy skrót łączący każdy obiekt JSON węzła z identyfikatorem

Ustawia dla każdego węzła współrzędną x na podstawie pozycji w tablicy

Zastępuje identyfikator tekstowy węzła wskaźnikiem do obiektu JSON

Rysuje krawędzie przy użyciu funkcji arc

```

.attr("class", "node")
.attr("r", 10)
.attr("cx", function (d) {return d.x;})

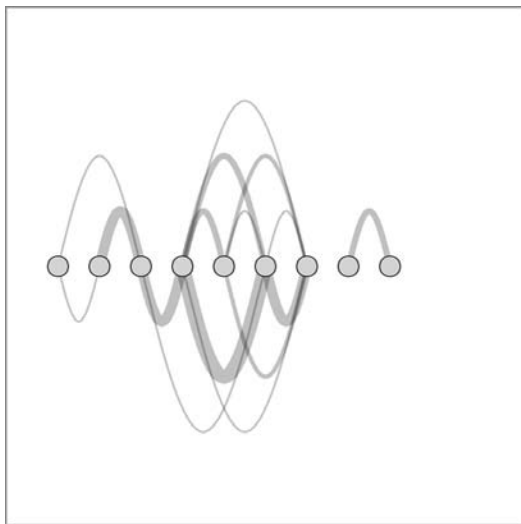
function arc(d,i) {
  var draw = d3.svg.line().interpolate("basis");
  var midX = (d.source.x + d.target.x) / 2;
  var midY = (d.source.x - d.target.x) * 2;
  return draw([[d.source.x,0],[midX,midY],[d.target.x,0]])
};
};
};

```

← Rysuje węzły jako koła według współrzędnych x węzłów

← Rysuje z wykorzystaniem interpolacji basis linię z węzła źródłowego do docelowego przez obliczony punkt środkowy

Zauważ, że w generowanej tablicy krawędzi używany jest skrót z identyfikatorem krawędzi w celu utworzenia referencji do obiektów. Dzięki przygotowaniu obiektów z referencjami do węzłów źródłowego i docelowego można łatwo wyznaczyć graficzne atrybuty elementów `<line>` i `<path>` używanych do przedstawiania krawędzi. Ta sama technika jest stosowana w opartych na siłach układach omówionych dalej w rozdziale. Wskutek uruchomienia tego kodu tworzony jest pierwszy diagram łukowy, pokazany na rysunku 6.5.



Rysunek 6.5. Diagram łukowy z połączeniami między węzłami reprezentowanymi jako łuki nad i pod węzłami. Łuki nad węzłami oznaczają krawędzie od lewej do prawej, natomiast łuki pod węzłami informują, że węzeł źródłowy znajduje się po prawej stronie, a docelowy — po lewej

W tego rodzaju abstrakcyjnych wykresach interaktywność nie jest już opcjonalna. Choć krawędzie są generowane według reguł, a liczba węzłów i krawędzi jest akceptowalna, trudno ustalić, kto jest z kim powiązany i w jaki sposób. Możesz dodać przydatny aspekt interaktywny i wyróżniać krawędzie węzła po umieszczeniu nad nim kursora oraz węzły powiązane z krawędzią — po przesunięciu nad nią myszy. W tym celu dodaj dwie funkcje przedstawione na listingu 6.6. Efekt zobrazowany został na rysunku 6.6.

Listing 6.6. Interaktywny kod diagramu łukowego

```

d3.selectAll("circle").on("mouseover", nodeOver);
d3.selectAll("path").on("mouseover", edgeOver);

function nodeOver(d,i) {
  d3.selectAll("circle").classed("active", function (p) {
    return p == d ? true : false;
  });

  d3.selectAll("path").classed("active", function (p) {
    return p.source == d || p.target == d ? true : false;
  });
};

function edgeOver(d) {
  d3.selectAll("path").classed("active", function(p) {
    return p == d ? true : false;
  });

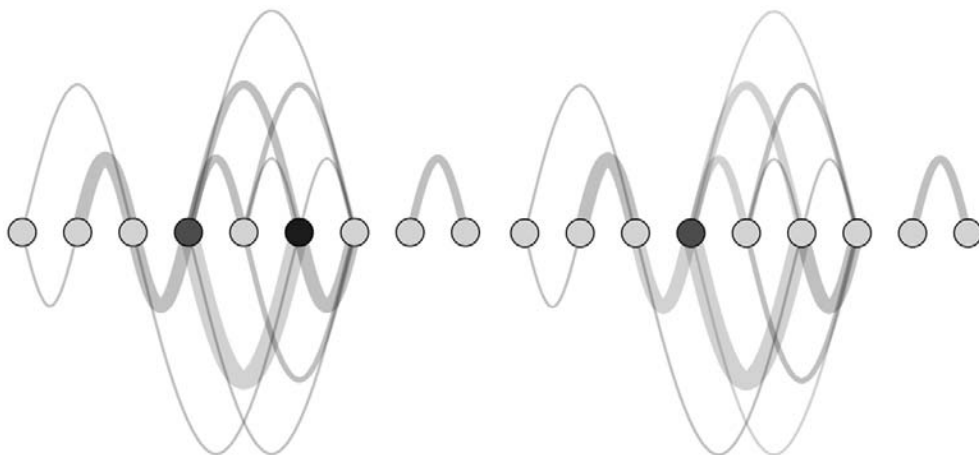
  d3.selectAll("circle").style("fill",function(p) {
    return p == d.source ? "blue" : p == d.target ? "green" : "lightgray";
  });
};

```

← Tworzy selekcję z wszystkimi węzłami, aby ustawić klasę węzła, nad którym znajduje się kursor, na "active"

← Krawędzie, w których wybrany węzeł jest wierzchołkiem źródłowym lub docelowym, są wyświetlane w kolorze czerwonym

← Ta zagnieżdżona instrukcja if sprawdza, czy dany węzeł jest wierzchołkiem źródłowym (kolor jest ustawiany na niebieski), czy docelowym (kolor zielony). Pozostałe węzły są wyświetlane w kolorze szarym



Rysunek 6.6. Reakcja na umieszczenie kursora nad krawędzią (po lewej) polega na wyróżnieniu krawędzi kolorem różowym, węzła źródłowego — kolorem niebieskim, a węzła docelowego — kolorem zielonym. Po przeniesieniu kursora nad węzeł (po prawej) węzeł jest wyróżniany kolorem czerwonym, a przyległe krawędzie — kolorem różowym (<http://bl.ocks.org/emeeks/dbbafc21d4c4fc879568>)

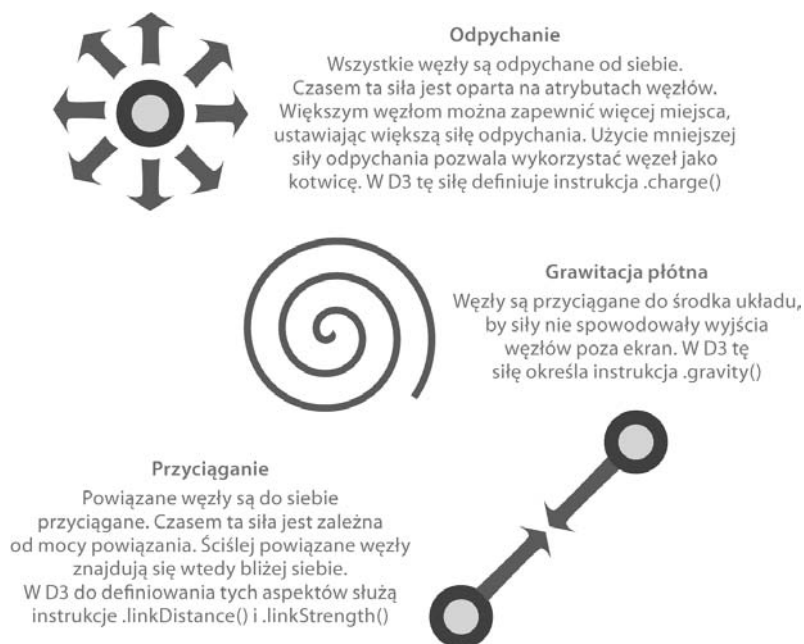


Jeśli chcesz dowiedzieć się więcej o diagramach łukowych i zastosować je do większych zbiorów danych, przyjrzyj się diagramom łukowym z wieloma osiami (ang. *hive plots*), czyli diagramom łukowym uporządkowanym na kilku osiach. W tej książce diagramy łukowe z wieloma osiami nie są opisywane, ale na stronie <https://github.com/d3/d3-plugins/tree/master/hive> znajdziesz wtyczkę z przeznaczonym dla nich układem. Zaletą

macierzy sąsiedztwa i diagramów łukowych jest kontrola nad porządkowaniem i rozmieszczaniem węzłów, a także ich liniowy układ. Następna technika wizualizowania sieci, której dotyczy pozostała część rozdziału, wykorzystuje zupełnie inne zasady określania, jak i gdzie należy rozmieszczać węzły i krawędzie.

6.2. Układ oparty na siłach

Nazwa układu opartego na siłach związana jest z metodą używaną do określania optymalnej graficznej reprezentacji sieci. Układ `force()`, podobnie jak opisane w rozdziale 5. układy dla chmur słów i diagramów Sankeya, dynamicznie aktualizuje pozycje elementów, by znaleźć dla nich najlepsze lokalizacje. Jednak w odróżnieniu od innych układów robi to w czasie rzeczywistym, a nie w ramach wstępnego przetwarzania przed wyświetleniem wykresu. Układ oparty na siłach działa na podstawie trzech sił, co pokazuje rysunek 6.7. Te siły odpychają węzły od siebie, przyciągają powiązane węzły i zapobiegają wyjściu węzłów poza ekran.



Rysunek 6.7. Siły w algorytmach opartych na siłach — odpychanie, grawitacja i przyciąganie. W takich algorytmach można uwzględniać też inne czynniki (na przykład hierarchie i wykrywanie społeczności), ale wymienione cechy są najczęściej spotykane. W dużych sieciach siły oblicza się z przybliżeniem, by poprawić wydajność algorytmów

Z tego podrozdziału dowiesz się, jak działają układy oparte na siłach i jak je tworzyć. Poznasz też ogólne zasady analizowania sieci, które to zasady pomogą Ci lepiej zrozumieć owe układy. Ponadto zobaczysz, jak dodawać i usuwać węzły oraz krawędzie, a także jak dostosowywać ustawienia układu w locie.

6.2.1. Tworzenie dla sieci diagramu opartego na siłach

Układ `force()` inicjowany na listingu 6.7 ma ustawienia, z którymi już się zetknąłeś. Najbardziej oczywiste jest ustawienie `size()`, określające tablicę z szerokością i wysokością układu, na podstawie której obliczane są siły. Ustawienia `nodes()` i `links()` są takie same jak dla układów Sankeya z rozdziału 5. Przyjmują tablice danych reprezentujące węzły i krawędzie. W tablicy z krawędziami tworzone są referencje do węzłów źródłowych i docelowych (tak jak dla diagramów łukowych). Takiego formatu oczekuje układ `force()`. Używane są tu liczby całkowite określające pozycje węzłów w ich tablicy. W rozdziale 5. podobnie sformatowano dane w tablicy krawędzi na potrzeby diagramu Sankeya. Nowym ustawieniem na listingu 6.7 jest `charge()` — określa ono, jak bardzo każdy węzeł odpycha inne węzły. Stosowany jest też nowy odbiornik zdarzeń, "tick", który należy powiązać z funkcją aktualizującą pozycje węzłów i krawędzi.

Listing 6.7. Funkcja dla układu opartego na siłach

```
function forceDirected() {
  queue()
    .defer(d3.csv, "nodelist.csv")
    .defer(d3.csv, "edgelist.csv")
    .await(function(error, file1, file2) {
      createForceLayout(file1, file2);
    });

  function createForceLayout(nodes, edges) {
    var nodeHash = {};
    for (x in nodes) {
      nodeHash[nodes[x].id] = nodes[x];
    };
    for (x in edges) {
      edges[x].weight = parseInt(edges[x].weight);
      edges[x].source = nodeHash[edges[x].source];
      edges[x].target = nodeHash[edges[x].target];
    };

    var weightScale = d3.scale.linear()
      .domain(d3.extent(edges, function(d) {return d.weight;}))
      .range([.1,1]);

    var force = d3.layout.force().charge(-1000)
      .size([500,500])
      .nodes(nodes)
      .links(edges)
      .on("tick", forceTick);

    d3.select("svg").selectAll("line.link")
      .data(edges, function (d) {return d.source.id + "-" + d.target.id;})
      .enter()
      .append("line")
      .attr("class", "link")
      .style("stroke", "black")
      .style("opacity", .5)
      .style("stroke-width", function(d) {return d.weight;});
  }
}
```

Określa, jak bardzo węzły się odpychają.
Wartość dodatnia oznacza przyciąganie

Zdarzenia "tick" są zgłaszane cały czas i powodują
wywoływanie powiązanej funkcji

Klucze węzłów i krawędzi
pomagają w późniejszym
aktualizowaniu sieci

```

var nodeEnter = d3.select("svg").selectAll("g.node")
  .data(nodes, function (d) {return d.id})
  .enter()
  .append("g")
  .attr("class", "node");

nodeEnter.append("circle")
  .attr("r", 5)
  .style("fill", "lightgray")
  .style("stroke", "black")
  .style("stroke-width", "1px");

nodeEnter.append("text")
  .style("text-anchor", "middle")
  .attr("y", 15)
  .text(function(d) {return d.id;});

force.start();

function forceTick() {
  d3.selectAll("line.link")
    .attr("x1", function (d) {return d.source.x;})
    .attr("x2", function (d) {return d.target.x;})
    .attr("y1", function (d) {return d.source.y;})
    .attr("y2", function (d) {return d.target.y;});

  d3.selectAll("g.node")
    .attr("transform", function (d) {
      return "translate("+d.x+","+d.y+")";
    })
  };
};

```

← **Zainicjowanie sieci prowadzi do zgłaszania zdarzeń "tick" i obliczania centralności węzłów**

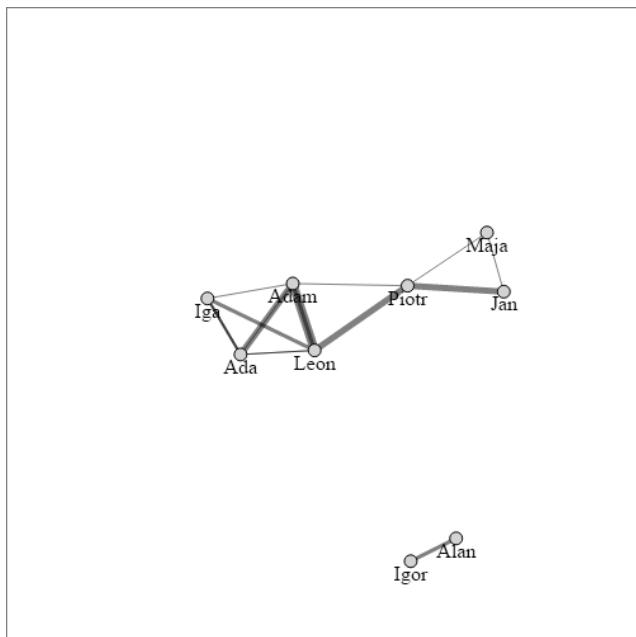
← **Funkcja forceTick aktualizuje rysujący krawędzie i węzły kod na podstawie nowo obliczonych pozycji węzłów**

Animowanego charakteru układu opartego na siłach nie da się pokazać w książce. Jednak na rysunku 6.8 widoczna jest struktura sieci, trudniejsza do przedstawienia w macierzach sąsiedztwa i diagramach łukowych. Od razu widać tu, że cztery węzły (Adama, Leona, Igi i Ady) są powiązane ze sobą i tworzą tak zwaną *klikę*. Trzy inne węzły (reprezentujące Jana, Piotra i Maję) są bardziej peryferyjne. Dwa węzły widoczne po prawej stronie (Igora i Alana) są połączone tylko ze sobą. Jedynym powodem, dla którego te dwa węzły są widoczne na ekranie, jest siła grawitacji z układu przyciągająca niepowiązane elementy do środka wykresu.

Grubość linii odpowiada sile powiązań. Choć jednak pokazana jest siła krawędzi, w układzie nie są widoczne kierunki. Można stwierdzić, że sieć jest skierowana, ponieważ rysowane krawędzie są półprzezroczyste. Dzięki temu widać, że dwie krawędzie o różnych wagach się pokrywają. Potrzebna jest metoda pokazująca kierunek krawędzi. Można to zrobić dzięki przekształceniu linii w strzałki za pomocą znaczników SVG.

6.2.2. Znaczniki SVG

Czasem potrzebny jest symbol (na przykład grot strzałki) na rysowanej linii lub ścieżce. Wymaga to zdefiniowania znacznika w elemencie `svg:defs` i powiązania go z elementem, na którym symbol ma zostać narysowany. Znaczniki można definiować statycz-



Rysunek 6.8. Układ oparty na siłach dla używanego zbioru danych. W grafice zastosowano ustawienia domyślne z układu opartego na siłach (<http://bl.ocks.org/emeeks/040c4eb87d36de3c87d3>)

nie w HTML-u lub generować dynamicznie podobnie jak inne elementy SVG, tak jak w kodzie na listingu 6.8. Znacznikiem może być dowolny kształt SVG. Tu używana jest ścieżka, ponieważ pozwala narysować grot strzałki. Znacznik można umieścić na początku, na końcu lub pośrodku linii. Dostępne ustawienia pozwalają określić kierunek symbolu względem nadrzędnego elementu.

Listing 6.8. Definicja i zastosowanie znacznika

```
var marker = d3.select("svg").append('defs')
  .append('marker')
  .attr("id", "Triangle")
  .attr("refX", 12)
  .attr("refY", 6)
  .attr("markerUnits", 'userSpaceOnUse')
  .attr("markerWidth", 12)
  .attr("markerHeight", 18)
  .attr("orient", 'auto')
  .append('path')
  .attr("d", 'M 0 0 12 6 0 12 3 6');
```

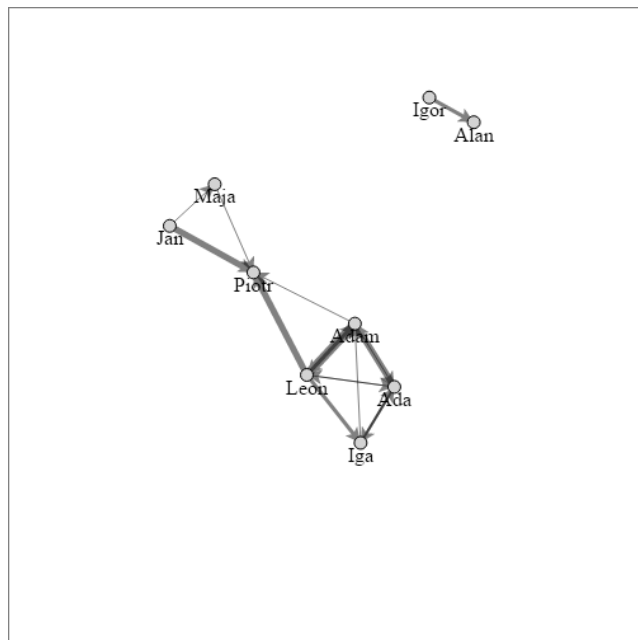
Domyślne ustawienia znacznika powodują określenie jego wielkości na podstawie atrybutu stroke-width elementu nadrzędnego. W tej sytuacji doprowadziłoby to do powstania nieczytelnych znaczników

```
d3.selectAll("line").attr("marker-end", "url(#Triangle)");
```

Znacznik jest przypisywany do linii w wyniku powiązania go z atrybutem marker-end, marker-start lub marker-mid

Dzięki zdefiniowanym na listingu 6.8 znacznikom można lepiej zrozumieć sieć (zobacz rysunek 6.9). Widać, w jaki sposób węzły są ze sobą powiązane. Można też zobaczyć, między którymi węzłami występują wzajemne powiązania (krawędzie w obu kierunkach). Wzajemność jest tu istotna, ponieważ występuje duża różnica między osobami lubiącymi tweety Kate Perry a użytkownikami, których wpisy polubiła Kate Perry (obecnie jest to użytkowniczka Twittera obserwowana przez największą liczbę ludzi).

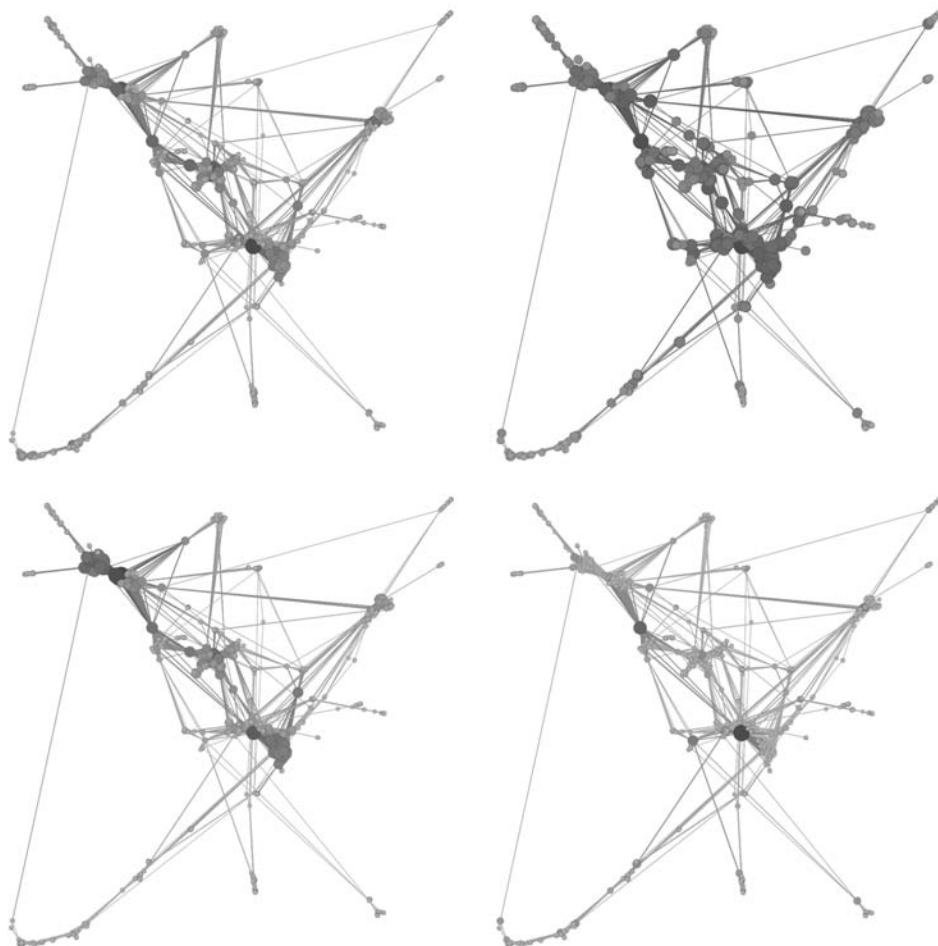
Kierunek krawędzi ma znaczenie, przy czym można przedstawiać go na różne sposoby, na przykład za pomocą zakrzywionych linii lub krawędzi rozszerzających się na jednym końcu. Aby uzyskać taki efekt, należy tworzyć krawędzie za pomocą elementu `<path>` zamiast `<line>`, tak jak w diagramach Sankeya lub diagramach łukowych.



Rysunek 6.9. Teraz przy krawędziach widoczne są znaczniki (groty strzałek), określające kierunek powiązań. Zwróć uwagę, że wszystkie groty mają tę samą wielkość

Jeśli uruchomiłeś przedstawiony kod, sieć prawdopodobnie wyglądała nieco inaczej niż na rysunku 6.9. Wynika to z tego, że wizualizacje sieci utworzone za pomocą układu opartego na siłach powstają na podstawie wzajemnych oddziaływań sił. Nawet w tak małej sieci oddziaływania mogą prowadzić do różnych lokalizacji węzłów. Może to spowodować, że użytkownicy mylnie uznają, iż różne wykresy oznaczają różne sieci. Jednym z rozwiązań jest wygenerowanie sieci za pomocą układu opartego na siłach i utrwalenie go w celu utworzenia bazowej mapy sieci. Potem można wprowadzać zmiany graficzne w utrwalonej sieci. Pojęcie mapy bazowej pochodzi z geografii. W dziedzinie wizualizowania sieci dotyczy stosowania tego samego układu z węzłami i krawędziami w różnych rozmiarach lub kolorach. Dzięki temu użytkownicy mogą zidentyfikować obszary sieci znacząco różniące się od innych według różnych miar. Wykorzystanie mapy bazowej pokazano na rysunku 6.10, ilustrującym, że jedną sieć można interpretować na wiele sposobów.

Układ oparty na siłach ma dodatkową zaletę, ponieważ pokazuje ogólną strukturę sieci. W zależności od wielkości i złożoności sieci może to wystarczać. Niemniej jednak dla niektórych danych o sieciach trzeba wyświetlić także inne miary.



Rysunek 6.10. Ta sama sieć mierzona na podstawie centralności stopniowej (ang. degree centrality; lewy górny rysunek), centralności bliskości (ang. closeness centrality; prawy górny rysunek), centralności wektora własnego (ang. eigenvector centrality; lewy dolny rysunek) i centralności pośredniej (ang. betweenness centrality; prawy dolny rysunek). Bardziej centralne węzły są większe i jasnoczerwone. Mniej centralne węzły są mniejsze i szare. Zauważ, że choć według wszystkich miar centralne są te same węzły, ich względny poziom centralności się zmienia, podobnie jak centralność innych węzłów

Pojęcie z obszaru wizualizowania informacji — kula włosów (ang. hairball)

Wizualizacje sieci są efektowne, bywają jednak tak skomplikowane, że stają się nieczytelne. Dlatego zetkniesz się z krytyką tak gęstych wizualizacji sieci i z zarzutem, że diagram jest niezrozumiały. Takie wizualizacje czasem można nazwać *kulą włosów*, ponieważ wiele krawędzi nachodzi na siebie, przez co wykres przypomina gęszcz nieuporządkowanych włosów.

Jeśli uważasz, że układ oparty na siłach jest nieczytelny, możesz połączyć go z inną wizualizacją, na przykład z macierzą sąsiedztwa, i wyróżniać elementy na obu wykresach, gdy użytkownik z nich korzysta. Techniki łączenia wizualizacji w ten sposób przedstawia rozdział 11.

6.2.3. Miary sieci

Badania nad sieciami trwają od dawna — przynajmniej od kilkudziesięciu lat, a jeśli uwzględnić matematyczną teorię grafów, od wieków. Dlatego w trakcie pracy z sieciami dostępne są różne pojęcia i miary. Tu przedstawiony jest tylko krótki ich przegląd. Jeśli chcesz dowiedzieć się więcej o sieciach, zapoznaj się ze znakomitym wprowadzeniem do sieci i ich analizy, opracowanym przez S. Weingarta, I. Milligana i S. Grahama (http://www.themacroscope.org/?page_id=337).

WAGA KRAWĘDZI

Zauważ, że w zbiorze danych dla każdej krawędzi dostępna jest wartość w kolumnie `weight`. Reprezentuje ona siłę połączenia dwóch węzłów. W przykładzie przyjęto, że im więcej polubień, tym silniejsze powiązanie między użytkownikami Twittera. Wyższe wagi przekładają się na szersze linie. Ponadto siły mogą wpływać na działanie układu opartego na siłach, co opisano dalej.

CENTRALNOŚĆ

Sieci reprezentują systemy. Jedną z istotnych rzeczy, jakie warto wiedzieć o węzłach systemu, jest to, które z nich są ważniejsze od innych. Określa to *centralność*. Węzły centralne są uważane za ważniejsze lub mające większy wpływ. Istnieje wiele różnych miar centralności. Kilka z nich pokazano na rysunku 6.10. Poszczególne miary bardziej precyzyjnie określają centralność w sieciach różnego typu. Jedną z miar centralności, centralność stopniowa, jest używana przez układ `force()` biblioteki D3.

CENTRALNOŚĆ STOPNIOWA

Centralność stopniowa (inna nazwa — centralność stopnia relacji) to łączna liczba krawędzi powiązanych z węzłem. W przykładowych danych centralność stopniowa dla Adama jest równa 6, ponieważ węzeł tej osoby jest wierzchołkiem źródłowym lub docelowym dla sześciu krawędzi. Centralność stopniowa to ogólna miara znaczenia węzła w sieci, gdyż przyjmuje się, że osoby lub obiekty o większej liczbie powiązań są ważniejsze lub mają większy wpływ. Centralność stopniowa z wagami służy do określania łącznej wartości krawędzi węzła. Dla Adama wartość tej miary to 18. Ponadto można uwzględniać *stopnie wejściowe* (ang. *in degree*) i *stopnie wyjściowe* (ang. *out degree*), służące do odróżniania krawędzi przychodzących od krawędzi wychodzących. Dla Adama wartości te wynoszą, odpowiednio, 4 i 2.

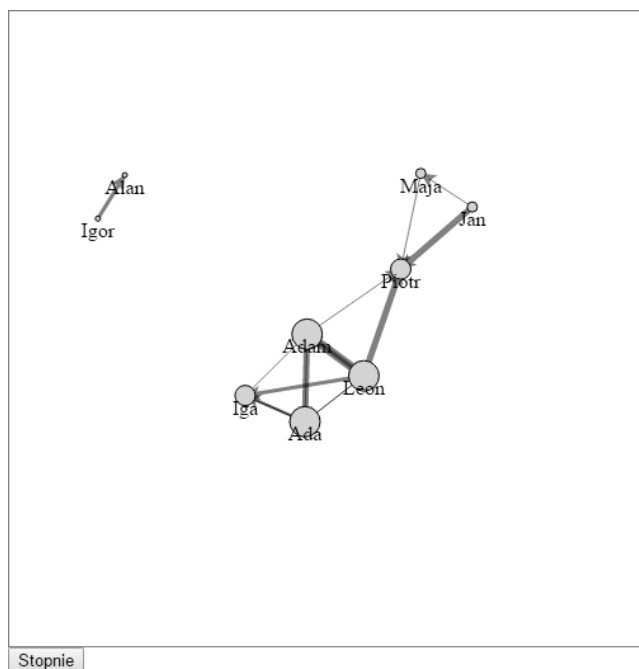
Przy każdym uruchomieniu układu `force()` biblioteka D3 określa łączną liczbę krawędzi węzła i na bazie tego aktualizuje jego atrybut `weight`. Dalej pokazano, jak na tej podstawie można wpływać na działanie układu opartego na siłach. Na razie dodaj przycisk, który pozwala zmienić wielkość węzłów z wykorzystaniem wartości atrybutu `weight`:

```
d3.select("#controls").append("button")
  .on("click", sizeByDegree).html("Stopnie");
```

```
function sizeByDegree() {
  force.stop();
```

```
d3.selectAll("circle")
  .attr("r", function(d) {return d.weight * 2;});
};
```

Rysunek 6.11 ilustruje wartość centralności stopniowej. Choć w małej sieci można zobaczyć i łatwo policzyć krawędzie oraz węzły, możliwość szybkiego znalezienia węzłów o największej i najmniejszej liczbie powiązań jest bardzo przydatna. Zauważ, że liczone są tu krawędzie biegnące w obu kierunkach. Dlatego, choć Leon jest powiązany z większą liczbą osób, jego węzeł jest tej samej wielkości co węzły Adama i Ady, mających tyle samo połączeń, ale z mniejszą liczbą użytkowników.



Rysunek 6.11. Wielkość węzłów jest określana na podstawie wag (łącznej liczby krawędzi). Promień koła jest ustawiany na wagę razy dwa

SKUPIENIA I MODULARNOŚĆ

Jednym z najważniejszych aspektów dotyczących sieci jest to, czy występują w niej społeczności i jak wyglądają. Aby to ustalić, należy zbadać, czy grupy węzłów są powiązane ze sobą w większym stopniu niż z resztą sieci (określa to *modularność*). Można też sprawdzić, czy węzły są wzajemnie powiązane (dotyczy tego *analiza skupień*). W pomiarach z tego obszaru są też uwzględniane wspomniane wcześniej kliki. *Klika* to grupa w pełni powiązanych ze sobą węzłów.

Zauważ, że wzajemne powiązania i struktura społeczności powinny być widoczne w układzie opartym na siłach. Wykres pokazuje, że skupienie obejmuje czterech mocno powiązanych użytkowników, a pozostałe osoby są oddalone. Jeśli wolisz dokonać innych pomiarów sieci i wykryć występujące w niej struktury, zapoznaj się z algorytmem wykrywania społeczności opracowanym przez Davida Mimno za pomocą biblioteki

D3 (<http://mimno.infosci.cornell.edu/community/>). Ten algorytm działa w przeglądarce i można go stosunkowo łatwo zintegrować z daną siecią, aby pokolorować ją na podstawie przynależności węzłów do społeczności.

6.2.4. Ustawienia układu opartego na siłach

Po zainicjowaniu układu opartego na siłach początkowo ustawienie `charge` miało wartość `-1000`. To ustawienie (a także kilka innych) zapewnia większą kontrolę nad działaniem układu.

USTAWIENIE CHARGE

Ustawienie `charge` określa szybkość, z jaką węzły są odpychane od siebie. Jeśli nie ustawisz tej wartości, użyte zostanie ustawienie domyślne `-30`. Powód użycia wartości `-1000` jest taki, że przy ustawieniu domyślnym sieć na ekranie będzie bardzo mała (rysunek 6.12).

Oprócz podania stałej wartości ustawienia `charge` można użyć funkcji akcesora i zmieniać tę wartość na podstawie atrybutów węzła. Możesz na przykład uzależnić ustawienie `charge` od wagi (centralności stopniowej), tak by węzły o wielu krawędziach bardziej odpychały od siebie inne węzły i miały dzięki temu więcej miejsca na wykresie.



Rysunek 6.12. Układ sieci przy domyślnym ustawieniu `charge`. Węzły są skupione zbyt blisko siebie, by były czytelne

Ujemne wartości ustawienia `charge` w układzie opartym na siłach oznaczają odpychanie. Możesz też podać wartość dodatnią, jeśli chcesz, by węzły były przyciągane do siebie. W wizualizacjach typowych sieci prawdopodobnie spowoduje to problemy, może jednak okazać się przydatne w bardziej skomplikowanych diagramach.

GRAWITACJA

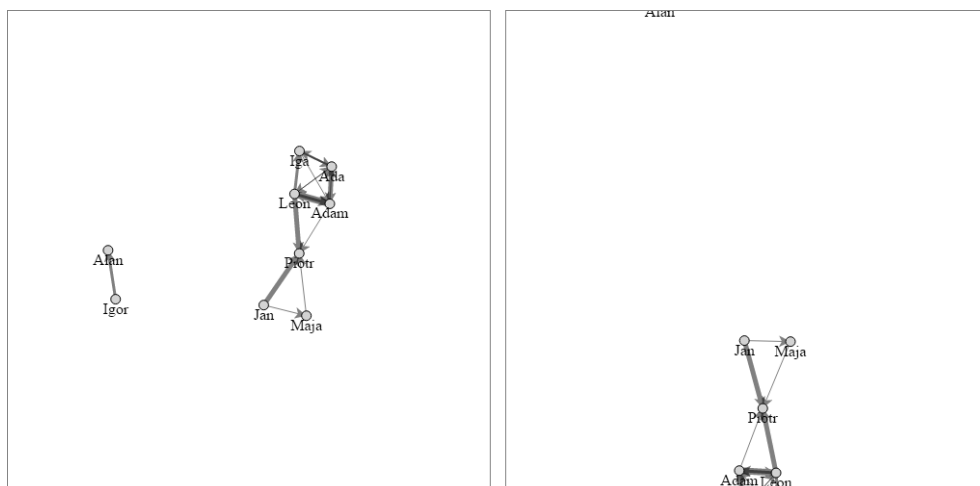
Gdy węzły się odpychają, jedyną siłą zapobiegającą ich wyjściu poza wykres jest *grawitacja płót*. Przyciąga ona wszystkie węzły do środka układu. Gdy grawitacja nie jest ustawiona, domyślnie ma wartość `0`, `1`. Rysunek 6.13 pokazuje efekt zwiększenia lub zmniejszenia grawitacji (przy ustawieniu `charge(-1000)`).

Grawitacja, w odróżnieniu od ustawienia `charge`, nie pozwala stosować funkcji akcesora, dlatego trzeba podać stałą wartość.

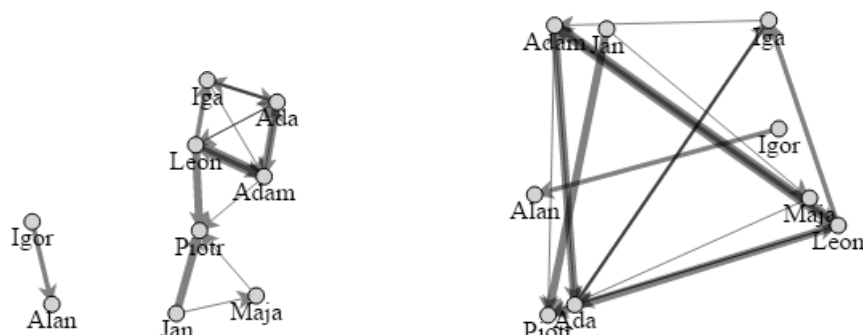
WŁAŚCIWOŚĆ LINKDISTANCE

Przyciąganie się między węzłami zależy od wartości właściwości `linkDistance`. Określa ona optymalną odległość między połączonymi węzłami. Jedną z przyczyn, dla których trzeba było ustalić tak wysoką wartość ustawienia `charge`, jest to, że właściwość `linkDistance` domyślnie ma wartość `20`. Po jej ustawieniu na `50` można zmniejszyć wartość ustawienia `charge` na `-100`. Efekt widoczny jest na rysunku 6.14.

Zbyt wysoka wartość parametru `linkDistance` powoduje, że sieć staje się za bardzo ściśnięta. Wskazują na to wyraźnie widoczne trójkąty w wizualizacji sieci. Rysunek 6.15 pokazuje sieć ściśniętą z powodu ustawienia dla właściwości `linkDistance` wartości `200`.



Rysunek 6.13. Zwiększenie grawitacji do 0,2 (po lewej) powoduje zbliżenie dwóch komponentów do środka układu. Zmniejszenie grawitacji do 0,05 (po prawej) sprawia, że mały komponent wychodzi poza ekran



Rysunek 6.14. Dzięki dostosowaniu właściwości `linkDistance` sieć staje się dużo bardziej czytelna

Rysunek 6.15. Zniekształcenie wynikające z wysokiej wartości właściwości `linkDistance` powoduje, że wykres wygląda tak, jakby Piotr był połączony z Adą. Ponadto niepowiązane ze sobą węzły znajdują się teraz w jednym skupieniu

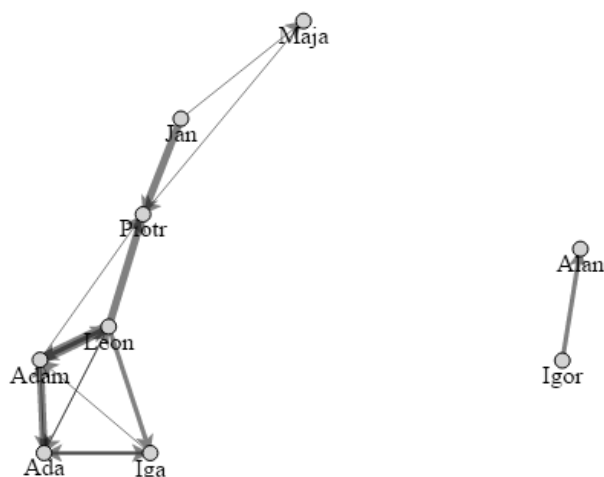
Właściwości `linkDistance` można przypisać funkcję zwracającą wartość na podstawie wag krawędzi. Pozwala to zmniejszać i zwiększać odległości dla wyższych lub niższych wag. Jednak aby uzyskać taki efekt, lepiej jest posłużyć się ustawieniem `linkStrength`.

USTAWIENIE LINKSTRENGTH

Układ oparty na siłach to symulacja fizyczna, prowadząca do ułożenia sieci w optymalnej graficznej postaci. Jeśli w sieci występują silniejsze i słabsze powiązania, dobrym pomysłem jest wywieranie przez nie większego i mniejszego wpływu na węzły. Aby uzyskać taki efekt, zastosuj ustawienie `linkStrength`. Przyjmuje ono stałą wartość lub funkcję akcesora określającą siłę krawędzi na podstawie atrybutów:

```
force.linkStrength(function (d) {return weightScale(d.weight);});
```

Rysunek 6.16 wyraźnie uwidacznia skutki tego ustawienia. Widać tu, że niektóre połączenia są słabe.



Rysunek 6.16. Uzależnienie siły przyciągania od siły powiązań między węzłami radykalnie zmienia strukturę sieci. Słabsze powiązania między Janem i Mają sprawiają, że obejmująca ich część sieci jest oddalona od reszty węzłów

6.2.5. Aktualizowanie sieci

Gdy tworzysz sieć, chcesz umożliwić użytkownikom dodawanie i usuwanie jej węzłów oraz ich przenoszenie. Możliwe też, że chcesz mieć możliwość dynamicznego dostosowywania ustawień zamiast zmieniania ich tylko podczas tworzenia układu.

ZATRZYMYWANIE I PONOWNE URUCHAMIANIE UKŁADU

Układ oparty na siłach w pewnym momencie zwalnia, a ostatecznie zatrzymuje się, gdy sieć ma odpowiedni wygląd i węzły nie są już przenoszone w nowe miejsca. Jeśli chcesz ponownie zobaczyć animację po zatrzymaniu układu, musisz jeszcze raz go uruchomić. Ponadto w celu wprowadzenia zmian w ustawieniach albo dodania lub usunięcia fragmentów sieci trzeba zatrzymać układ, a następnie ponownie go uruchomić.

INSTRUKCJA `FORCE.STOP()`

Za pomocą instrukcji `force.stop()` możesz wyłączyć interaktywne działanie układu. Powoduje to zatrzymanie symulacji. Układ warto zatrzymać, gdy w innym miejscu strony zachodzą interakcje z komponentami lub gdy trzeba zmienić wygląd sieci.

INSTRUKCJA `FORCE.START()`

Aby rozpocząć lub ponownie uruchomić animację dla układu, wywołaj instrukcję `force.start()`. Zetknąłeś się już z tym poleceniem, ponieważ we wcześniejszym przykładzie posłużyło do uruchomienia układu opartego na siłach.

INSTRUKCJA `FORCE.RESUME()`

Jeśli nie wprowadziłeś żadnych zmian w węzłach ani krawędziach sieci, a chcesz ponownie zacząć animację, wywołaj instrukcję `force.resume()`. Zeruje ona parametr powodujący zatrzymanie animacji, dzięki czemu elementy układu znów zaczynają się poruszać.

INSTRUKCJA FORCE.TICK()

Jeżeli chcesz przesunąć układ o jeden krok do przodu, wywołaj polecenie `force.tick()`. Układ oparty na siłach zużywa czasem dużo zasobów. Możliwe, że chcesz uruchomić go tylko na kilka sekund, zamiast wykonywać cały czas.

OPERACJA FORCE.DRAG()

W tradycyjnych programach do analizowania sieci użytkownik może przenosić węzły na nowe pozycje. Służy do tego operacja `force.drag()`. Operacje przypominają komponenty, ponieważ też są wywoływane dla elementu za pomocą instrukcji `.call()`, ale zamiast tworzyć elementy SVG, generują zestaw odbiorników zdarzeń.

Operacja `force.drag()` tworzy odbiorniki zdarzeń związane z przeciąganiem. Umożliwia to użytkownikom kliknięcie i przeciągnięcie węzłów w trakcie działania układu opartego na siłach. Aby włączyć obsługę przeciągania dla wszystkich węzłów, wybierz je i wywołaj dla selekcji operację `force.drag()`:

```
d3.selectAll("g.node").call(force.drag());
```

ATRYBUT FIXED

Gdy układ oparty na siłach jest powiązany z węzłami, każdy węzeł ma atrybut logiczny `fixed`, określający, czy węzeł podlega siłom w każdym kroku generowania wykresu. Skuteczna interaktywna technika polega na ustawieniu węzłów jako stałych, gdy użytkownik wchodzi z nimi w interakcje. To pozwala użytkownikom przeciągać węzły na nowe pozycje na płótnie i w ten sposób wizualnie uporządkować ważne węzły. W celu odróżnienia stałych węzłów od pozostałych można za pomocą funkcji ustawić dla tych pierwszych większą wartość właściwości `"stroke-width"`. Efekt przeciągnięcia kilku węzłów pokazuje rysunek 6.17.

```
d3.selectAll("g.site").on("click", fixNode);
```

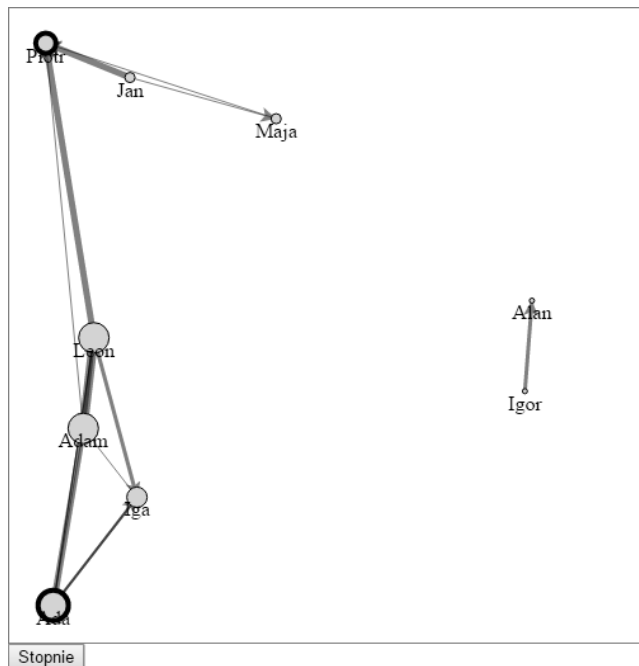
```
function fixNode(d) {
  d3.select(this).select("circle").style("stroke-width", 4);
  d.fixed = true;
};
```

6.2.6. Usuwanie i dodawanie węzłów oraz krawędzi

Możliwe, że chcesz przefiltrować sieć albo pozwolić użytkownikom na dodawanie lub usuwanie węzłów. Aby przefiltrować sieć, musisz zatrzymać układ (instrukcja `stop()`), usunąć niepotrzebne węzły i krawędzie, ponownie powiązać tablice z układem opartym na siłach i uruchomić układ (instrukcja `start()`).

Można w tym celu zastosować filtr do tablicy węzłów. Załóżmy, że chcesz wyświetlić sieć z osobami obserwowanymi przez ponad 20 użytkowników, aby zobaczyć powiązania między najbardziej wpływowymi ludźmi.

To jednak nie wystarczy, ponieważ w układzie pozostaną krawędzie prowadzące do nieistniejących w nim węzłów. Potrzebny jest też bardziej skomplikowany filtr dla tablicy krawędzi. Zastosowanie funkcji `.indexOf` do tablicy pozwala łatwo przefiltrować krawędzie. Wystarczy sprawdzić, czy przefiltrowana tablica węzłów zawiera zarówno



Rysunek 6.17. Węzeł reprezentujący Adę został przeciągnięty w lewy dolny róg i utrwalony na tej pozycji. Węzeł reprezentujący Piotra przeniesiono w lewy górny róg i też utrwalono. Pozostałe, nieutrwalone węzły przyjęły pozycje na podstawie układu opartego na siłach

źródłowy, jak i docelowy wierzchołek. Ponieważ przy wiązaniu tablic z selekcją na listingu 6.8 zastosowano klucze, można wywołać operację `selection.exit()` i łatwo zaktualizować sieć. Potrzebny kod jest przedstawiony na listingu 6.9, a efekty jego działania ilustruje rysunek 6.18.

Listing 6.9. Filtrowanie sieci

```
function filterNetwork() {
  force.stop();
  var originalNodes = force.nodes();
  var originalLinks = force.links();
  var influentialNodes = originalNodes.filter(function (d) {
    return d.followers > 20;
  });

  var influentialLinks = originalLinks.filter(function (d) {
    return influentialNodes.indexOf(d.source) > -1 &&
      influentialNodes.indexOf(d.target) > -1;
  });

  d3.selectAll("g.node")
    .data(influentialNodes, function (d) {return d.id;})
    .exit()
    .transition()
    .duration(4000)
    .style("opacity", 0)
    .remove();
```

← Pobiera obecne wersje tablic węzłów i krawędzi powiązanych z układem opartym na siłach

← Tworzy tablicę krawędzi powiązanych z istniejącymi węzłami

```

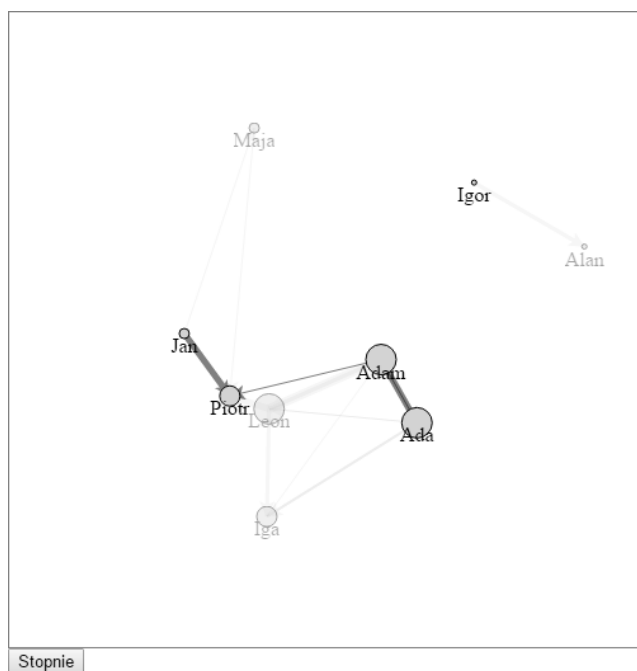
d3.selectAll("line.link")
  .data(influentialLinks, function(d) {
    return d.source.id + "-" + d.target.id;
  })
  .exit()
  .transition()
  .duration(3000)
  .style("opacity", 0)
  .remove();

force
  .nodes(influentialNodes)
  .links(influentialLinks);

force.start();
};

```

← Dodanie przejścia do operacji `.exit()` powoduje zastosowanie go tylko do usuwanych węzłów. Usuwanie odbywa się dopiero po zakończeniu przejścia



Rysunek 6.18. Przefiltrowana sieć obejmuje tylko węzły osób obserwowanych przez ponad 20 użytkowników. Filtrowanie jest uruchamiane przez przycisk Stopnie. Zauważ, że Igor nie ma żadnych powiązań (stopień 0), dlatego odpowiadające mu koło ma promień 0 i jest niewidoczne. Na zrzucie uchwycono dwa procesy — przechodzenie węzłów od zerowej do pełnej przezroczystości i usuwanie krawędzi

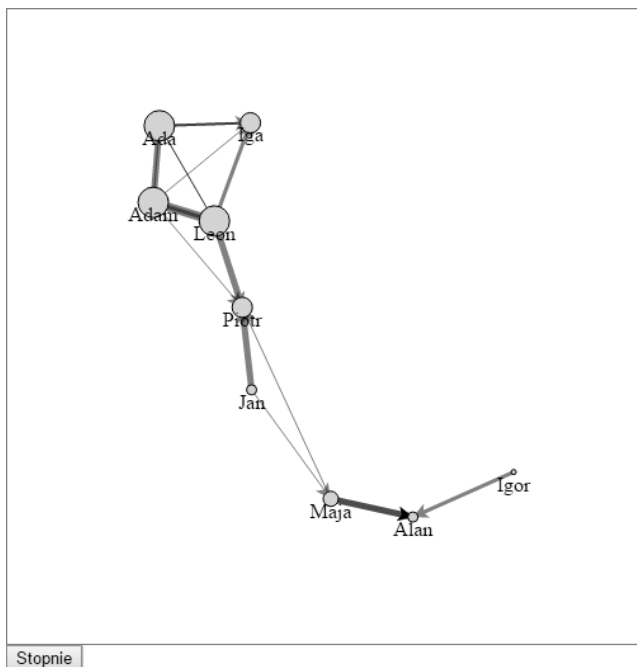
Ponieważ algorytm oparty na siłach jest po filtrowaniu ponownie uruchamiany, możesz zobaczyć, jak usunięcie tak wielu węzłów wpływa na kształt sieci. Animacja jest tu ważna, ponieważ ujawnia zmiany w strukturze sieci.

Dodawanie węzłów i krawędzi do sieci jest łatwe — o ile dane są właściwie sformatowane. Należy zatrzymać oparty na siłach układ, dodać odpowiednio sformatowane węzły lub krawędzie do tablic i ponownie powiązać dane. Jeśli na przykład chcesz dodać krawędź między Mają i Alanem (pokazaną na rysunku 6.19), musisz zatrzymać układ w opisany wcześniej sposób, utworzyć nowy punkt danych reprezentujący krawędź i dodać go do tablicy krawędzi. Następnie należy ponownie powiązać dane, dodać nową linię reprezentującą krawędź i jeszcze raz uruchomić układ. Przedstawia to listing 6.10.

Listing 6.10. Funkcja służąca do dodawania krawędzi

```
function addEdge() {
  force.stop();
  var oldEdges = force.links();
  var nodes = force.nodes();
  newEdge = {source: nodes[0], target: nodes[8], weight: 5};
  oldEdges.push(newEdge);
  force.links(oldEdges);
  d3.select("svg").selectAll("line.link")
    .data(oldEdges, function(d) {
      return d.source.id + "-" + d.target.id;
    })
    .enter()
    .insert("line", "g.node")
    .attr("class", "link")
    .style("stroke", "red")
    .style("stroke-width", 5)
    .attr("marker-end", "url(#Triangle)");

  force.start();
};
```



Rysunek 6.19. Sieć z nową krawędzią. Zauważ, że ponieważ ponownie zainicjowano układ, wagi dla węzła reprezentującego Alana zostały poprawnie przeliczone

Jeśli chcesz dodać nowe węzły pokazane na rysunku 6.20, warto, żebyś od razu utworzył nowe krawędzie. Nie jest to konieczne, ale w przeciwnym razie węzły będą krążyć po ekranie i nie zostaną połączone z obecną siecią. Potrzebny kod pokazano na listingu 6.11.

Listing 6.11. Funkcja do dodawania węzłów i krawędzi

```
function addNodesAndEdges() {
  force.stop();
  var oldEdges = force.links();
  var oldNodes = force.nodes();
  var newNode1 = {id: "raj", followers: 100, following: 67};
  var newNode2 = {id: "wu", followers: 50, following: 33};
  var newEdge1 = {source: oldNodes[0], target: newNode1, weight: 5};
  var newEdge2 = {source: oldNodes[0], target: newNode2, weight: 5};
  oldEdges.push(newEdge1, newEdge2);
  oldNodes.push(newNode1, newNode2);
  force.links(oldEdges).nodes(oldNodes);
  d3.select("svg").selectAll("line.link")
    .data(oldEdges, function(d) {
      return d.source.id + "-" + d.target.id
    })
    .enter()
    .insert("line", "g.node")
    .attr("class", "link")
    .style("stroke", "red")
    .style("stroke-width", 5)
    .attr("marker-end", "url(#Triangle)");

  var nodeEnter = d3.select("svg").selectAll("g.node")
    .data(oldNodes, function(d) {
      return d.id
    })
    .enter()
    .append("g")
    .attr("class", "node")
    .call(force.drag());

  nodeEnter.append("circle")
    .attr("r", 5)
    .style("fill", "red")
    .style("stroke", "darkred")
    .style("stroke-width", "2px");

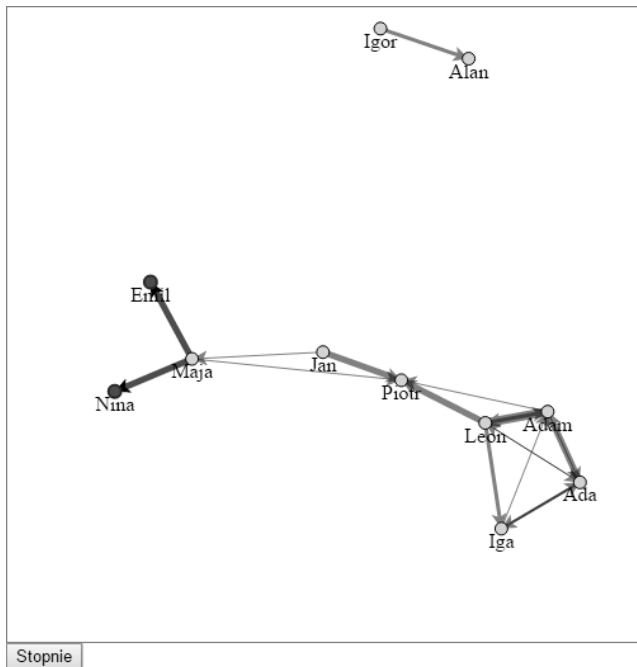
  nodeEnter.append("text")
    .style("text-anchor", "middle")
    .attr("y", 15)
    .text(function(d) {return d.id;});

  force.start();
};
```

6.2.7. Ręczne określanie pozycji węzłów

Układ oparty na siłach nie przenosi elementów. Zamiast tego oblicza ich pozycje na podstawie atrybutów x i y oraz relacji między elementami. W każdym kroku te atrybuty są aktualizowane. Funkcja generująca kroki wybiera elementy `<line>` i `<g>` oraz przenosi je zgodnie z nowymi wartościami x i y.

Gdy chcesz ręcznie przenosić elementy, możesz to zrobić w standardowy sposób. Najpierw jednak zatrzymaj pracę układu, by funkcja generująca kroki nie zmieniła ustawionych przez Ciebie pozycji elementów. Rozmieść węzły tak jak na wykresie



Rysunek 6.20. Sieć z dwoma nowymi węzłami (reprezentującymi Emila i Ninę) powiązanymi z węzłem reprezentującym Maję

punktowym. Uwzględnij liczbę obserwujących i obserwowanych użytkowników. Warto też dodać osie, by zwiększyć czytelność wykresu. Potrzebny kod pokazano na listingu 6.12, a efekty jego uruchomienia — na rysunku 6.21.

Listing 6.12. Ręczne przenoszenie węzłów

```
function manuallyPositionNodes() {
  var xExtent = d3.extent(force.nodes(), function(d) {
    return parseInt(d.followers)
  });
  var yExtent = d3.extent(force.nodes(), function(d) {
    return parseInt(d.following)
  });
  var xScale = d3.scale.linear().domain(xExtent).range([50,450]);
  var yScale = d3.scale.linear().domain(yExtent).range([450,50]);

  force.stop();
  d3.selectAll("g.node")
    .transition()
    .duration(1000)
    .attr("transform", function(d) {
      return "translate("+ xScale(d.followers)
        +","+yScale(d.following) +")";
    });

  d3.selectAll("line.link")
    .transition()
    .duration(1000)
    .attr("x1", function(d) {return xScale(d.source.followers);})
```

```

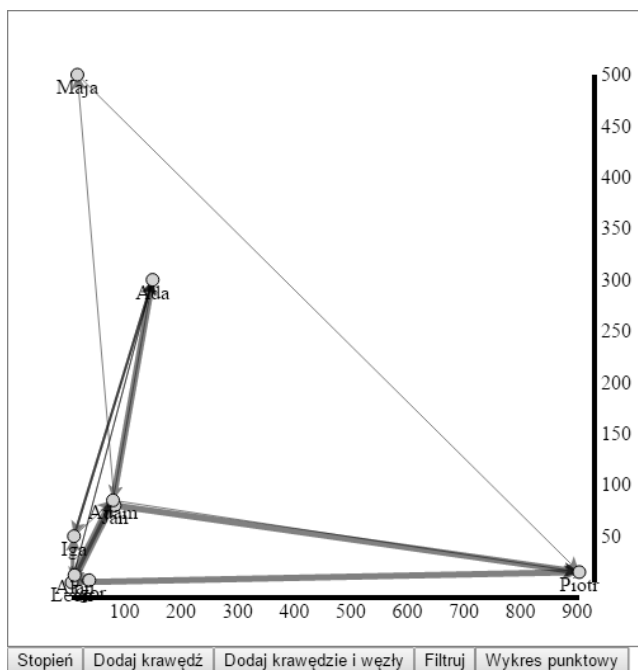
.attr("y1", function(d) {return yScale(d.source.following);})
.attr("x2", function(d) {return xScale(d.target.followers);})
.attr("y2", function(d) {return yScale(d.target.following);});

var xAxis = d3.svg.axis().scale(xScale).orient("bottom").tickSize(4);
var yAxis = d3.svg.axis().scale(yScale).orient("right").tickSize(4);

d3.select("svg").append("g").attr("transform",
  "translate(0,460)").call(xAxis);
d3.select("svg").append("g").attr("transform",
  "translate(460,0)").call(yAxis);

d3.selectAll("g.node").each(function(d){
  d.x = xScale(d.followers);
  d.px = xScale(d.followers);
  d.y = yScale(d.following);
  d.py = yScale(d.following);
});
};

```



Rysunek 6.21. Gdy sieć jest przedstawiana za pomocą wykresu punktowego, krawędzie powodują wizualny chaos. Ten wykres jest mało czytelny, warto go jednak porównać z układem opartym na siłach (<http://bl.ocks.org/emeeks/91e3737e8e6c02cd31f2>)

Zauważ, że dla każdego węzła trzeba zaktualizować atrybuty x i y , a ponadto zmodyfikować atrybuty px i py . Atrybuty px i py to współrzędne x i y węzła przed ostatnim krokiem. Jeśli ich nie zaktualizujesz, układ po ponownym uruchomieniu przyjmie, że węzły są przesuwane z bardzo dużą szybkością, dlatego gwałtownie przeniesie je na nową pozycję.

Jeśli nie zaktualizujesz atrybutów x , y , px i py , przy następnym uruchomieniu układu węzły przed ich przesunięciem najpierw wrócą na swoje pierwotne pozycje.

W zastosowanym podejściu podczas ponownego uruchomienia układu za pomocą instrukcji `force.start()` węzły i krawędzie w ramach animacji są przenoszone z bieżących pozycji.

6.2.8. Optymalizacja

Układ oparty na siłach wymaga bardzo dużo zasobów. To dlatego spowalnia pracę, a w końcu się zatrzymuje. Jeśli taki układ jest używany dla dużej sieci, może obciążyć komputer użytkownika i sprawić, że sprzęt przestanie reagować. Dlatego pierwszą wskazówką z obszaru optymalizacji jest ograniczanie liczby węzłów w sieci, a także liczby krawędzi. Zgodnie z ogólną regułą nie należy dodawać więcej niż 100 węzłów, chyba że wiesz, iż użytkownicy korzystają z przeglądarek najlepiej radzących sobie z formatem SVG (takich jak Safari i Chrome).

Jeśli jednak musisz uwzględnić większą liczbę węzłów i chcesz ograniczyć obciążenie, możesz wykorzystać instrukcję `force.chargeDistance()` do ustawienia maksymalnej odległości, dla której uwzględniana jest siła odpychania. Im niższe jest to ustawienie, tym mniej ustrukturyzowany będzie układ, ale tym szybciej będzie działał. Ponieważ sieci bardzo różnią się między sobą, musisz poeksperymentować z różnymi wartościami ustawienia `chargeDistance`, by znaleźć optymalny poziom dla przetwarzanej sieci.

6.3. Podsumowanie

W tym rozdziale nauczyłeś się kilku technik wyświetlania danych o sieci i zapoznałeś się szczegółowo z układem opartym na siłach udostępnianym przez bibliotekę D3 na potrzeby takich danych. Nie istnieje jeden sposób wizualnego reprezentowania sieci. Teraz znasz szereg metod oraz ich statyczne, dynamiczne i interaktywne odmiany, którymi możesz się posługiwać. Oto omówione zagadnienia:

- Podstawy terminologii i miar związanych z sieciami (na przykład pojęcia takie jak: krawędź, węzeł, stopień i centralność).
- Formatowanie list węzłów i krawędzi w sposób typowy dla biblioteki D3.
- Budowanie macierzy sąsiedztwa dla skierowanej sieci z wagami i dodawanie interaktywności w celu umożliwienia eksploracji macierzy.
- Tworzenie interaktywnego diagramu łukowego dla skierowanej sieci z wagami.
- Stosowanie prostych technik wyszukiwania krawędzi dla danego węzła.
- Budowanie i modyfikowanie układów opartych na siłach.
- Używanie akcesorów do tworzenia dynamicznych sił.
- Dodawanie interaktywnych mechanizmów zmiany wielkości węzłów na podstawie centralności stopniowej.

Skoncentrowano się tu na wizualizowaniu informacji o sieci, ponieważ współczesny świat jest pełen takich danych. W następnym rozdziale omówiono inną powszechnie spotykaną, ale specyficzną dziedzinę — wizualizowanie danych geograficznych. Tu nauczyłeś się różnych sposobów reprezentowania sieci. W rozdziale 7. poznasz różne techniki generowania map — w postaci kafelków, globusa i tradycyjnych wielokątów opartych na danych.

Skorowidz

A

- akcesory, 49, 73, 351
 - dla obszarów, 151
 - złożone, 149
- aktualizowanie
 - siatki, 314
 - sieci, 218
 - stron, 90
- algorytmy generujące wizualizacje, 189
- analizowanie modelu DOM, 31
- anamorficzne mapy, 262
- animacje, 272
- aplikacja
 - QGIS, 231
 - Bivariate Hexbin Map, 345
- aplikacje interaktywne, 25, 287
- architektura projektu, 96
- arkusz kalkulacyjny, 266, 268, 292
 - animacje, 272
 - element `<div>`, 270
 - tabele, 268
- arkusze stylów, 41, 97, 267
- atrybut
 - class, 42
 - logiczny fixed, 219
- automatyczna kwantyzacja, 113
- automatyczne wykrywanie ekranów, 389

B

- baza danych PostGIS, 231
- biblioteka
 - colorbrewer.js, 113
 - D3.js, 19, 21
 - GDAL, 231
 - queue.js, 203
 - TopoJSON, 251, 253
- biblioteki zewnętrzne, 98
- bitmapa, 276

C

- centralność, 214
 - stopniowa, degree centrality, 213
- chmura tagów, 190
- CSS, Cascading Style Sheets, 29
- CSV, Comma-Separated Values, 50

D

- D3.js, 19, 21
- dane, 96
 - czasowe, 69
 - generator, 345, 349
 - geograficzne, 51, 52, 230
 - geometryczne, 68
 - geoprzestrzenne, 227
 - formatowanie, 67
 - funkcje wewnątrzwerszowe, 76
 - kategorialne, 68
 - liczbowe, 67, 70, 71
 - o sieci, 197, 199, 226
 - pomiar, 73
 - przekształcanie, 69
 - rastrowe, 262
 - rzutowanie, 69
 - sieciowe, 50, 52
 - sortowanie, 272
 - surowe, 51, 69
 - tabelaryczne, 50
 - topologiczne, 68
 - w formacie TopoJSON, 252
 - wczytywanie, 64
 - wektorowe, 259
 - wiązanie, 74
 - wizualizacja, 82
 - zagnieżdżone, 50, 72
- data-driven documents, 19
- definicja znacznika, 211
- dendrogram, 175, 177

diagram
 łukowy, 205, 206
 z wieloma osiami, 207
 oparty na siłach, 209
 Sankeya, 183, 186, 189
 strumieniowy, 150, 156, 178
 Woronoja, 262
 statyczny sieci, 198
 dodawanie
 etykiet do komponentu, 327
 kontrolki zooma, 336
 krawędzi, 219, 222
 osi, 142
 prostokątów, 319
 siatki kartograficznej, 240
 układów, 183
 węzłów, 219, 223
 zawartości strony, 56
 dokumenty sterowane danymi, 19
 DOM, Document Object Model, 20, 29, 100, 265
 dostęp do danych, 66, 76
 dotyk, 361
 rejestrowanie zdarzeń, 361
 wizualizowanie zdarzeń, 362
 drzewa, 172
 czwórkowe, quadtrees, 331, 349, 352
 duże zbiory danych, 331
 geograficznych, 332
 o sieciach, 344
 działanie biblioteki D3, 21

E

ekrany, 389
 element
 <canvas>, 37, 337, 339, 343
 <circle>, 37, 91, 139, 177
 <div>, 56, 270
 <g>, 39, 89, 139
 <line>, 37
 <path.domain>, 134
 <path>, 35, 39, 118
 <polygon>, 37
 <rect>, 37, 139
 <select>, 281
 <svg>, 35, 339, 343
 <text>, 38

elementy
 strony, 27
 SVG, 59

F

filtrowanie sieci, 220
 format
 CSV, 50
 GeoJSON, 231
 JSON, 230
 shapefile, 230, 231
 SVG, 21
 TopoJSON, 251
 formatowanie danych, 67
 formaty plików, 65
 funkcja
 .append(), 57, 75, 88
 .attr(), 76
 .brighter(), 108
 .darker(), 108, 109
 .data(), 75, 91
 .delay(), 59, 104
 .domain(), 71
 .duration(), 60
 .each(), 119
 .empty(), 119
 .enter(), 47, 48, 76, 87
 .exit(), 46, 87, 91
 .filter(), 48
 .html(), 34, 49, 76, 116
 .insert(), 76, 121
 .node(), 105
 .on(), 56
 .range(), 71
 .remove(), 89
 .rotate(), 193
 .style(), 34
 .transition(), 59
 brushed(), 303
 changeView(), 386
 d3.dsv(), 50
 d3.geo.tile, 256
 d3.scale(), 70
 d3.selectall(), 75
 d3.touches(), 361
 d3.tsv(), 50
 force.drag(), 219

force.resume(), 218
 force.start(), 218, 226
 force.stop(), 218
 force.tick(), 219
 layout(), 188
 legend(), 323
 processGrid(), 313
 redraw(), 294
 resizeGrid1(), 317
 select, 34, 57
 selectAll.data(), 249
 toDataURL(), 276
 visit(), 354
 zoomFinished(), 341
 funkcje
 anonimowe, 49
 dla układu opartego na siłach, 209
 do generowania mapy, 232
 do obsługi dotknięć, 369, 373
 generujące diagram łukowy, 205
 generujące macierz sąsiedztwa, 201
 rysowania elementów, 25
 sortujące dane, 272
 tablic, 47
 wewnętrzzwierszowe, 76
 wyróżniające elementy, 299
 z rodziny zoom, 341
 G
 galeria rysunków, 277
 generator
 d3.svg.area, 152
 linii, 146, 147
 obszarów, 154
 siatki kartograficznej, 239
 generowanie
 danych, 345, 349
 histogramu, 79
 losowych danych geograficznych, 334
 treści, 113
 wizualizacji, 64, 93
 wykresu, 139
 GeoJSON, 230
 geokodowanie, 230
 geolokalizacja, 389
 urządzenia, 389
 GIS, Geographic Information System, 228, 230

globus, 244
 grafika wektorowa, 38
 grawitacja płótina, 208, 216
 grupowanie, 71
 elementów SVG, 40
 grupy kół, 169, 293

H

HCL, 110
 histogramy, 79, 160
 HTML, 115
 HTML5, 28

I

informacje geoprzestrzenne, 227
 inspekcja modelu DOM, 36, 48
 inspektor elementów, 32
 instrukcja, *Patrz* funkcja
 interaktywne
 style, 100
 wyróżnianie elementów, 279
 interaktywność, 238, 296
 legendy, 325
 interaktywny kod diagramu łukowego, 207
 interpolacja, 143, 148, 152
 kolorów, 110
 linii, 148

J

JavaScript, 44
 jednostki legendy, 327

K

kanały, 85
 kartogram, 238
 kategorie danych, 71
 klasa
 active, 42, 44
 inactive, 42
 tentative, 42–44
 kod
 arkusza kalkulacyjnego, 292
 diagramu łukowego, 205
 do rysowania legendy, 327
 do rysowania na płótnie, 274

kod
 do wyróżniania wierszy, 296
 histogramu, 161
 macierzy sąsiedztwa, 201
 mapy z kafelkami, 258
 panelu kontrolnego, 291
 rysujący diagram Sankeya, 186
 układu opartego na siłach, 209
 wiązania danych, 168
 wykresu słupkowego, 293
 związany z analizą dotknięć, 366
 kodowanie danych geograficznych, 230
 kolejność
 rysowania elementów, 58
 wyświetlania elementów, 30
 kolor, 107
 kolory rozróżnialne, 111
 komponenty, 322
 własne, 319
 wykresów, 125
 konsola JavaScriptu, 31
 analizowanie modelu DOM, 31
 pisanie kodu, 34
 testowanie kod, 49
 kontrolka
 wyboru zakresu, 299, 349, 383
 zooma, 336
 kula włosów, hairball, 213

L

LAB, 110
 layout, *Patrz* układ
 liczba węzłów, 226
 linie, 145
 lista krawędzi, 199
 literały, 53
 łańcuchowe, 53

Ł

łańcuchy wywołań metod, 45
 łączenie komponentów aplikacji, 287

M

macierz sąsiedztwa, 201, 204
 manipulowanie
 kolejnością, 58
 modelem DOM, 105

mapy, 23
 anamorficzne, 262
 dane geograficzne, 230
 element `<canvas>`, 337
 format TopoJSON, 252
 interaktywność, 238
 obszary, 236
 odwzorowania, 233, 236
 odwzorowanie satelitarne, 250
 przesuwanie, 242
 przybliżanie, 242, 243
 punkty, 235
 rysowanie punktów, 234
 sąsiednie obiekty, 255
 scalanie, 253
 siatka kartograficzna, 240
 skala, 234
 tworzenie, 229
 w bibliotece D3, 228
 wielokąty, 235
 z kafelkami, 54, 256, 258
 z przykładowymi danymi, 335
 zoom, 241
 mechanizm ustawiania wymiarów, 315
 mediana wieku, 138
 metadane, 230
 metoda, *Patrz* funkcja
 miary sieci, 214
 mieszanie kolorów, 109
 model DOM, 20, 29, 100, 265
 modularność, 215
 modyfikowanie kolorów, 321

N

nagłówek, 327
 narzędzie Gephi, 51

O

obiekty, 53
 graficzne, 135
 nadrzędne, 51
 obliczanie wysokości komórek siatki, 318
 obracanie globusa, 244
 obsługa
 dotknięć, 369, 373, 384
 rotacji, 370
 obszary, 236

- odbiornik zdarzeń, 56, 209
- odpychanie, 208
- odwzorowania, 236
- odwzorowania, 233
 - Mollweidego, 237
 - satelitarne, 250
- ograniczanie liczby węzłów, 226
- okna wyskakujące, 378
- określanie
 - pozycji węzłów, 223
 - wyglądu elementów, 27
- operacja zoom, 241
- optymalizacja, 354

P

- panel kontrolny, 288
- pętla while, 119
- pierwsza aplikacja, 55
- pisanie kodu, 34
- plik
 - bigdata.css, 333
 - bigdata.html, 333
 - boxplots.csv, 136
 - ch4stylesheet.css, 134
 - ch7.css, 229
 - cities.csv, 67, 73, 230
 - colorbrewer.js, 112
 - d3ia.css, 98, 115
 - d3ia_2.html, 98
 - edgelist.csv, 199
 - modal.html, 115
 - movies.csv, 150
 - networks.css, 200
 - nodelist.csv, 200
 - soccerviz.js, 99
 - streamdata.csv, 179
 - tile.js, 257
 - tweetdata.csv, 143, 144
 - tweets.json, 66, 85, 313
 - worddata.csv, 191
 - world.geojson, 252
 - worldcup.csv, 96
- plótno, 273
- SVG, 57
- pole ograniczające, 231
- pomiar danych, 73
- ponowne rysowanie komponentów, 307

- powiększanie rysunku, 279
- pozycja węzłów, 223
- proces wizualizowania danych, 64
- projektowanie sterowane danymi, 95
- przejścia, 166
 - graficzne, 103
- przekształcanie danych, 69
- przenoszenie węzłów, 224
- przepływ danych, 63
- przesuwanie map, 242
- przezroczystość, 43
- przybliżanie mapy, 243
- przyciąganie, 208
- punkty na mapie, 234

R

- ręczne przenoszenie węzłów, 224
- RGB, 107
- rotacja, 370
- rozmiar ekranu, 294, 389
- rysowanie
 - danych geograficznych na płótnie, 336
 - dendrogramu, 172
 - diagramu Sankeya, 186
 - komponentów, 307
 - legendy, 327
 - linii, 145, 147
 - mapy z przykładowymi danymi, 335
 - na płótnie, 262, 274–276
 - punktów na mapie, 234
 - skumulowanych obszarów, 154
 - warstw danych, 152
 - wykresu kołowego, 163
 - wykresu pudełkowego, 138
- rysunek rastrowy, 276
- rysunki, 97, 113
- rzutowanie danych, 69

S

- scalanie, 253
- selekcje, 26, 74
 - elementów, 281
- selektory CSS, 40
- serwis
 - bl.ocks.org, 345
 - GitHub, 192

siatka, 313
 kartograficzna, 240
 sieci, 197
 nieskierowane, 199
 nowa krawędź, 222
 przefiltrowane, 221
 skierowane, 199
 siły, 208
 skale, 70, 234
 HCL, 110
 kolorów, 71
 bazujące na regionach, 122
 kwantylowe, 71
 LAB, 110
 quantize, 323
 skalowanie, 70
 skupienia, 215
 słowo kluczowe this, 105
 sortowanie, 272
 kolumn, 272
 stopnie
 wejściowe, 214
 wyjściowe, 214
 stosowanie standardu HTML5, 28
 styl opacity, 354
 style
 CSS, 39, 267
 CSS panelu kontrolnego, 290
 osi, 131
 SVG, Scalable Vector Graphics, 21, 29, 34, 116
 systemy
 GIS, 230
 informacji geograficznej, 228
 szkic panelu kontrolnego, 289

Ś

ścieżka, 153

T

tablety, 380
 tablice, 47
 powiązań, 203
 współrzędnych, 230
 technika hexbinning, 262
 techniki optymalizacji, 354

telefon, 384
 teoria kolorów, 108
 treść w prezentacji danych, 81
 tworzenie
 arkusza kalkulacyjnego, 268
 chmury słów, 192
 diagramu dla sieci, 209
 dostosowujących się wizualizacji danych, 375
 drzew czwórkowych, 352
 etykiet, 88
 globusa, 244
 interaktywnych wizualizacji sieci, 24
 komponentów, 319
 kontrolki wyboru zakresu, 300
 lepszyc map, 239
 linii i kół, 57
 macierzy sąsiedztwa, 201
 map, 23, 229
 osi wykresu, 128
 panelu kontrolnego, 291
 projektów sterowanych danymi, 359
 przykładowych danych, 334
 selekcji, 75
 sieci, 198
 układów, 312
 wykresu, 126
 pierścieniowego, 165
 punktowego, 85

U

układ, layout, 159
 d3.layout.grid.js, 312, 315
 dla chmury słów, 194
 dla grup kół, 168
 drzewiasty, 173
 oparty na siłach, 208, 346
 aktualizowanie sieci, 218
 diagramy, 209
 miary sieci, 214
 optymalizacja, 226
 pozycja węzłów, 223
 ustawienia, 216
 węzły i krawędzie, 219
 znaczniki SVG, 210
 skumulowany, 177, 179
 ukrywanie miast, 248

- uruchamianie układu, 218
- urządzenia mobilne, 357
 - analiza dotknięć, 365
 - dotyk, 359
 - przesuwanie elementów, 363, 365
 - rotacja, 370
 - wizualizacja, 359
- usługa geokodowania, 230
- ustawianie
 - atrybutów, 56
 - kanałów, 84
 - klucza, 91
 - stylu, 56
 - wymiarów, 315
- ustawienia odwzorowania satelitarnego, 250
- ustawienie
 - charge, 216
 - linkStrength, 217
- usuwanie
 - krawędzi, 219
 - węzłów, 219
- UTF-8, 30
- używanie układu dla siatki, 313

W

- waga krawędzi, 214
- wczytywanie danych, 64, 320
 - geograficznych, 235
- węzły, 197
 - docelowe, 199
 - określanie pozycji, 223
 - źródłowe, 199
- wiązanie
 - danych, binding data, 26, 64, 74, 91
 - komponentów, 322
- wierzchołki, 197
- wizualizacja, 189
 - analizy dotknięć, 365
 - danych, 20, 64, 374, 381
 - danych na urządzenia przenośne, 391
 - dużych zbiorów danych, 331, 336
 - informacji, 38, 54, 63, 85, 108, 114
 - sieci, 24, 197, 213, 347
- właściwość, 30, 230
 - clipAngle, 247
 - linkDistance, 216
- współrzędne x i y, 349

- wtyczka
 - d3.hexbin, 262
 - Firebug, 32
- wtyczki biblioteki D3, 185
- wybieranie zakresu, 349, 352
- wygląd elementów, 27
- wykres, 125
 - kołowy, 162
 - liniowy, 143, 151
 - pierścieniowy, 165
 - pudełkowy, 138, 139
 - punktowy, 85, 128, 137, 144
 - skumulowany, 155
 - śłupkowy, 83, 293
- wykresy
 - dane, 128
 - generatory, 127
 - komponenty, 127
 - osie, 128
 - style osi, 131
 - układy, 127
 - wielokrotnego użytku, 329
 - zasady tworzenia, 126
- wypełnienia ścieżek, 41
- wypełnienie, 44
- wyróżnianie
 - wierszy, 296
 - zagnieżdżonych elementów danych, 298
- wyświetlanie
 - danych, 128, 340
 - elementów, 338
- wyznaczanie sąsiadujących obiektów, 255

Z

- zamykanie ścieżek, 41
- zapisywanie obrazków, 275
- zasoby, 97
- zastosowanie znacznika, 211
- zbędne ozdobniki, chartjunk, 114
- zbiory wielokątów, multipolygons, 241
- zdarzenia, 100
 - kontrolki wyboru zakresu, 306
 - myszy, 296
 - związane z dotykiem, 361
- zdarzenie mouseover, 204, 238

zmienianie

wielkości wykresu, 294

typu danych, 69

znaczniki SVG, 210

znaki UTF-8, 30

zoom, 241, 336

odwzorowania, 262

oparty na szczypnięciach, 367

semantyczny, 243

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA



Helion SA

D3.js

jest biblioteką przeznaczoną do tworzenia zaawansowanych wizualizacji danych w rozmaitych aplikacjach internetowych. Ta solidnie zaprojektowana biblioteka oczywiście umożliwia generowanie wykresów, ale pozwala również na tworzenie map, interaktywnych diagramów, paneli kontrolnych dla danych, raportów i wielu innych animowanych elementów. Zapewnia co najmniej tę samą wydajność co Flash czy aplety Javy, ale jest zintegrowana ze standardami internetowymi i z modelem DOM dla HTML. To nie koniec zalet D3.js — to także świetne narzędzie do dynamicznego aktualizowania bardziej standardowych witryn internetowych.

Niniejsza książka pozwoli Ci na płynne rozpoczęcie pracy z tą biblioteką, dającą fantastyczne możliwości. Nauczysz się, jak tworzyć interaktywną grafikę i aplikacje sterowane danymi. Zaczyniesz od zestawu praktycznych przykładów, dostosowanych do różnego rodzaju wykresów, sieci i map. Wykorzystasz przy tym gotowe układy z biblioteki D3.js. Zapoznasz się z praktycznymi technikami projektowania zawartości stron, tworzenia animacji i prezentowania zmieniających się danych. Zobaczysz między innymi, jak tworzyć interaktywną grafikę i wykorzystywać dane przesyłane strumieniowo.

Dzięki tej książce zrozumiesz:

- zasady wizualizacji danych
- techniki wiązania i wczytywania danych oraz tworzenia elementów graficznych na ich podstawie
- metody pracy z grafiką wektorową
- tworzenie elementów służących do wizualizacji danych
- rozbudowane aplikacje wykorzystujące mapy
- sposoby tworzenia kompletnych aplikacji opartych na bibliotece D3.js również dla urządzeń przenośnych

Elijah Meeks — starszy inżynier wizualizacji danych w firmie Netflix. Z biblioteki D3.js korzystał w pracy dla Uniwersytetu Stanforda i znanych firm z całego świata. Był opiekunem technicznym wielu projektów naukowych, takich jak ORBIS (<http://orbis.stanford.edu>) czy Kindred Britain (<http://kindred.stanford.edu>).

Jeden obraz mówi więcej niż tysiąc słów. Czy już masz pomysł na wykorzystanie D3.js?

Helion

41045

numer katalogowy

księgarnia internetowa



<http://helion.pl>

zamówienia telefoniczne



0 801 339900



0 601 339900

Sprawdź najnowsze promocje:

● <http://helion.pl/promocje>

Książki najchętniej czytane:

● <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

● <http://helion.pl/nowosci>

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-283-1823-6



9 788328 318236

Informatyka w najlepszym wydaniu

cena: 69,00 zł