

Diagramy i grafy za pomocą pakietu Graphviz

GRAFY DLA KAŻDEGO

Do przygotowania mamy prezentację, artykuł, dokumentację – tekst już gotowy, brakuje jedynie rysunków. Postanawiamy wykonać je ręcznie, ale... po co się męczyć, jeśli może to za nas zrobić automat? MACIEJ DOBOSZ

Czym jest, a czym nie jest Graphviz

Omawiane w niniejszym artykule narzędzie, Graphviz, pozwoli nam przygotować nawet najbardziej złożone grafy, począwszy od zwykłych drzew binarnych, poprzez diagramy UML, a skończywszy na grafach ilustrujących skomplikowane, wielokierunkowe zależności. Ponieważ Graphviz jest wyspecjalizowanym narzędziem wysokiego poziomu, nie nadaje się do tworzenia obrazów innego typu, np. wykresów, własnych kształtów itp. Z przykładów zamieszczonych w tym artykule będzie można łatwo wynioskować, czy Graphviz spełni się w zadanej mu roli.

Pakiet składa się zestawu programów i bibliotek. Najczęściej używanym narzędziem jest *dot*, niewielki program do tworzenia grafów skierowanych, który przetwarza plik utworzony w języku DOT na plik graficzny (wektorowy lub rastrowy). Oprócz tego do dyspozycji mamy kilka innych programów służących do tworzenia innych typów grafów – nieco rzadziej spotykanych na co dzień.

Wszystkie te programy mają podobną składnię. Wywołuje się je zazwyczaj w ten sposób:

```
program -Tformat plik_wejściowy.dot >
plik_wyjściowy
```

W przykładach poniżej *program* to *dot*, zaś *format* to jeden z wielu formatów graficznych obsługiwanych przez Graphviz: TIFF, BMP, JPEG, GIF, PS, PDF, EPS, SVG itd. Nie wszystkie formaty obsługiwane są w ten sam sposób i korzystają z różnych sterowników wyjścia. Jeśli chcemy wyświetlić listę obsługiwanych sterowników wyjścia dla danego formatu, wywołujemy program z dwukropkiem po nazwie formatu, np.:

```
dot -Tpdf:
```

Jeśli dany format obsługuje więcej niż jeden sterownik, możemy wymusić jego wykorzystanie, np.:

```
Tpng:cairo:cairo a1.dot > a1.png
```

Proste grafy

Przyjrzyjmy się najprostszemu przykładowi z Listing 1. Ppis grafu składa się ze słowa kluczowego *graph* (graf nieskierowany) bądź *digraph* (w przypadku grafów skierowanych, czyli digrafów), po którym

Listing 1: Kod grafu przedstawionego na Rysunku 1

```
graph a1 {
  "Witamy!"
}
```



Rysunek 1: Rezultat działania kodu z Listing 1.

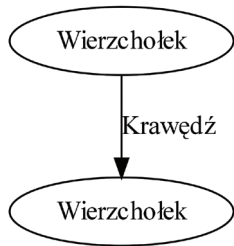
następuje nazwa grafu. Następnie mamy opisy poszczególnych elementów grafu, zwanych wierzchołkami i krawędziami. Jeśli ta terminologia brzmi nam obco, przyjrzyjmy się Rysunkowi 2.

Listing 2 znajduje się z kolei kod, który posłużył do wygenerowania Rysunku 2. Jak widzimy, tym razem użyliśmy etykiet (*label*). Domyślnie bowiem Graphviz używa jako etykiety nazwy danego wierzchołka. Gdybyśmy nie użyli etykiet, lecz spróbowali użyć nazw, otrzymalibyśmy

Listing 2: Kod grafu przedstawionego na Rysunku 2

```
digraph a2 {
  "Wierzchołek1" [label="Wierzchołek"];
  "Wierzchołek2" [label="Wierzchołek"];
  "Wierzchołek1" -> "Wierzchołek2" [label="Krawędź"];
}
```

jeden wierzchołek o nazwie *Wierzchołek*, wskazujący na samego siebie. Ponieważ chodziło nam o coś zupełnie innego, użyliśmy dwóch wierzchołków *Wierzchołek1* i *Wierzchołek2*, którym nadałmy tę samą etykietę *Wierzchołek*.



Rysunek 2: Rezultat działania kodu z Listingu 2.

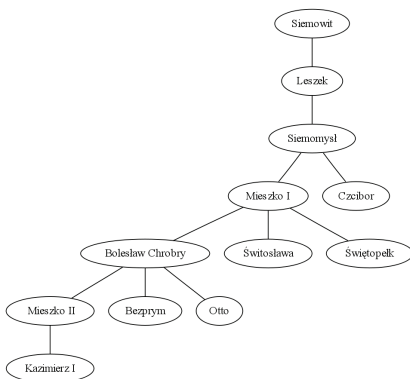
Zauważmy, że Listing 2 zamiast słowa kluczowego *graph* zawiera *digraph*: w ten sposób informujemy Graphviz, że ma wygenerować graf skierowany. Podobną rolę pełni sekwencja *->* zamiast użytej wcześniej *--*. Pomieszczenie tych dwu spo-

wodowałyby wyświetlenie komunikatu o błędzie.

Jedną z największych zalet Graphviz jest prostota określania zależności między wierzchołkami. Spójrzmy na przykład z Listingu 3: przedstawiamy na nim coś w rodzaju fragmentu drzewa genealogicznego dynastii Piastów. Wszystkie odgałęzienia określone są prostymi wyrażeniami, zawierającymi „potomne” elementy. Jak widzimy, DOT jest językiem wysokiego poziomu, pozwalającym skupić się na koncepcjach – szczegółami związanymi z układem grafu zajmie się odpowiedni program. Pamiętajmy, że każdy z nich wygeneruje inny typ grafu – by się o tym przekonać, wystarczy zastąpić polecenie *dot* w przykładach powyżej jednym z listy: *neato*, *twopi*, *circo*, *fdp*.

Modyfikacje wyglądu

Układ elementów grafu i wygląd wierzchołków można w dużej mierze modyfikować, łącznie z wykorzystaniem swoich własnych grafik. Listingu 4 widzimy kod korzystający z atrybutu *shapefile*, który pozwala załadować utworzony wcześniej obrazek i przypisać mu rolę kształtu danego wierzchołka. Możemy przy tym wykorzystywać dowolny obrazek w jednym z



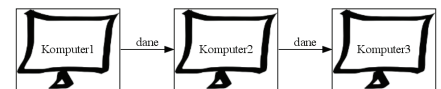
Rysunek 3: Rezultat działania kodu z Listingu 3.

Listing 3: Kod grafu przedstawionego na Rysunku 3

```
graph a3
{
  "Siemowit" -- "Leszek" -- "Siemomysł" -- "Mieszko I" -- "Bolesław Chrobry" -- "Mieszko II" --
  "Kazimierz I";
  "Siemomysł" -- "Czcibor";
  "Mieszko I" -- "Świętosława";
  "Mieszko I" -- "Świętopełk";
  "Bolesław Chrobry" -- "Bezprym";
  "Bolesław Chrobry" -- "Otto";
}
```

Listing 4: Kod grafu przedstawionego na Rysunku 4

```
digraph a4 {
  rankdir="LR";
  "Komputer1" [shapefile="komputer.png"];
  "Komputer2" [shapefile="komputer.png"];
  "Komputer3" [shapefile="komputer.png"];
  "Komputer1" -> "Komputer2" -> "Komputer3" [label="dane"];
}
```



Rysunek 4: Rezultat działania kodu z Listingu 4.

popularnych formatów obsługiwanych przez Graphviz, na przykład PNG.

Zauważmy, że w poprzednich przykładach pierwszy element znajduje się u góry wygenerowanego obrazku z grafem. O ile przy drzewie genealogicznym czy przedstawianiu algorytmu może być to pożądanym zachowaniem, to jednak w wielu wypadkach bardziej przydatny może być inny układ. Na Listingu 4 użyliśmy opcji *rankdir*, która określa kierunek: możemy użyć argumentów *TB* (od góry do dołu), *BT* (od dołu do góry), *LR* (od lewej do prawej) i *RL* (od prawej do lewej).

Zanim jednak zaczniemy korzystać z własnych kształtów, warto zapoznać się z tymi, które oferuje Graphviz. Domyślny kształt to *ellipse*, czyli elipsa. Do dyspozycji mamy również *box* (prostokąt), *polygon* (wielokąt), *circle* (koło), *triangle* (trójkąt), *diamond* (kwadrat skierowany jednym z wierzchołków do dołu), *folder* (ikonka katalogu), *note* (ikonka przedstawiająca notatkę) i wiele innych. Jeśli nie chcemy, by dany wierzchołek korzystał z jakiegokolwiek kształtu, wybieramy atrybut *plaintext* lub *none*.

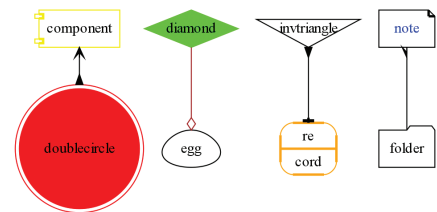
Modyfikacje kształtów dotyczą rzecz jasna nie tylko wierzchołków, ale i krawędzi, zwykle przedstawianych w formie strzałek. Jednak oprócz strzałek (wklęsłej – *vee* i trójkątnej – *normal*) możemy użyć rombu (*diamond*), wypełnionego prostokąta (*box*), koła (*circle*) itd. Kształty te można dodatkowo modyfikować, np. *diamond* oznacza niewypełniony romb, *obox* – niewypełniony prostokąt itd. Oprócz „główki” strzałki (*arrowhead*) mamy też jej zakończenie (*arrowtail*): atrybuty obu tych opcji możemy ustawiać oddzielnie.

Listing 5: Kod grafu przedstawionego na Rysunku 5

```

digraph a5 {
a1 [shape="component",label="component",color="yellow"];
a2 [shape="doublecircle",label="doublecircle",style="filled",color="red"];
a3 [shape="diamond",label="diamond",style="filled",color="green"];
a4 [shape="egg",label="egg"];
a5 [shape="invtriangle",label="invtriangle"];
a6 [shape="record",style="rounded",color="orange",label="{ re | cord }"];
a7 [shape="note",label="note",fontcolor="blue"];
a8 [shape="folder",label="folder"];
a1 -> a2 [arrowtail=vee,arrowhead=inv];
a3 -> a4 [arrowhead=odiamond,color="brown"];
a5 -> a6 [arrowtail=inv,arrowhead=tee];
a7 -> a8 [arrowtail=lcrow,arrowhead=none];
}

```



Rysunek 5: Rezultat działania kodu z Listingu 5.

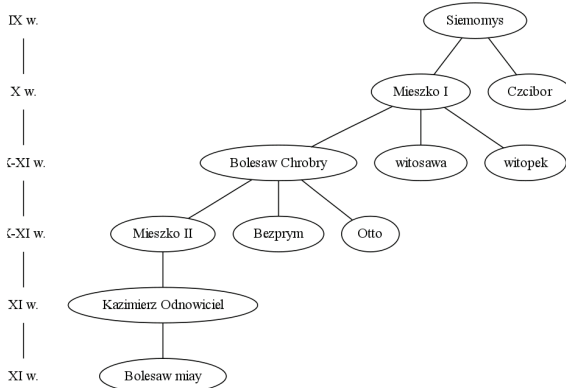
modyfikujących standardowy wygląd grafu. Zwróćmy uwagę na wierzchołek *a6*: jest to specjalny typ zwany rekordem. Rekordy służą do dzielenia wierzchołka na logiczne części, z których każda może mieć własne krawędzie.

Z kolei na Rysunku 6 widzimy dwa grafy. Pierwszy z nich ma ustawiony kształt na *none* (czyli „brak”) i reprezentuje linię czasu. Z kolei drugi jest zmodyfikowaną wersją grafu z Listingu 3. Jeśli chcemy wymusić ustawienie dwóch wierzchołków na tym samym poziomie, używamy wyrażenia *rank=same*, które nadaje poszczególnym elementom ten sam „status”.

Graphviz a Python

Graphviz ma wiele dowiązań do różnych języków. Dowiązania te ułatwiają automatyczne generowanie grafów, bez konieczności dodatkowego przetwarzania przez skrypty pośredniczące. Możemy np. bezpośrednio pobierać dane z systemu, generować graf zależności i wyświetlać go na WWW – wszystko to za pomocą Pythona.

Na Listingu 7 widzimy przykład możliwości pakietu *graphviz-python*, generującego drzewo zależności między modułami jądra. Wykorzystuje on plik */proc/modules*, zawierający informacje o wszystkich modułach. Pierwsza kolumna tego pliku zawiera nazwę modułu, druga określa, ile pamięci zajmuje dany moduł, trzecia podaje liczbę instancji załadowanych w danej chwili, zaś w czwartej znajdziemy listę modułów, od których zależy dany moduł lub znak -, jeśli lista jest pusta, przy czym poszczególne elementy tej listy oddzielone są przecinkami. Piąta



Rysunek 6: Rezultat działania kodu z Listingu 6.

Listing 6: Kod grafu przedstawionego na Rysunku 6

```

graph a6
{
{
node [shape=none]
a1[label="IX w."];
a2[label="X w."];
a3[label="X-XI w."];
a4[label="X-XI w."];
a5[label="XI w."];
a6[label="XI w."];
a1 -- a2 -- a3 -- a4 -- a5 -- a6;
}
"Siemomysł" -- "Mieszko I" -- "Bolesław Chrobry" -- "Mieszko II" -- "Kazimierz Odnowiciel" -- "Bolesław Śmiały";
"Siemomysł" -- "Czcibor";
"Mieszko I" -- "Świętosława";
"Mieszko I" -- "Świętopełk";
"Bolesław Chrobry" -- "Bezprym";
"Bolesław Chrobry" -- "Otto";
}

```

Listing 7: Demonstracja dowiązań Graphviz do Pythona

```
#!/usr/bin/python

# graf zależności między modułami jądra

# autor: Michael Hohn <mhohn@lbl.gov>
# bazując na modgraph.tcl Johna Ellsona <ellson@research.att.com>

import sys
import gv

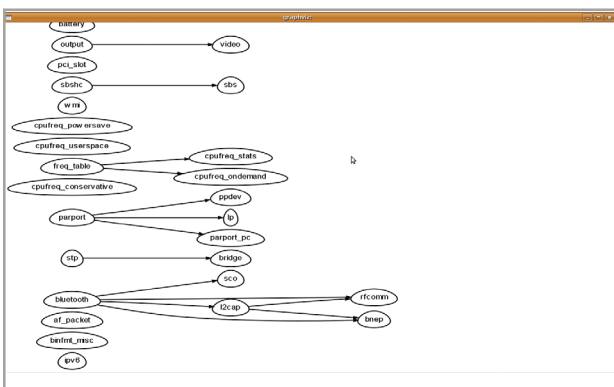
modules = open("/proc/modules", 'r').readlines()

G = gv.digraph("G")
N = gv.protonode(G)
E = gv.protoedge(G)

gv.setv(G, 'rankdir', 'LR')
gv.setv(G, 'nodesep', '0.05')
gv.setv(N, 'shape', 'egg')
gv.setv(N, 'width', '0')
gv.setv(N, 'height', '0')
gv.setv(N, 'margin', '0.03')
gv.setv(N, 'fontsize', '8')
gv.setv(N, 'fontname', 'helvetica')
gv.setv(E, 'arrowsize', '.4')

for rec in modules:
    fields = rec.split(' ')
    n = gv.node(G, fields[0])
    for usedby in fields[3].split(','):
        if (usedby != '-' & (usedby != '')):
            gv.edge(n, gv.node(G, usedby))

gv.layout(G, 'dot')
gv.render(G, 'xlib')
```



Rysunek 7: Rezultat działania kodu z Listingu 7 – wykres zależności między modułami jądra.

kolumna zawiera status modułu, zaś szósta – bieżący offset pamięci jądra danego modułu.

Jak nietrudno się zorientować, przygotowanie tak przygotowanych danych sprowadza się do wykorzystania informacji zawartych w pierwszej i czwartej kolumnie (na listingu odpowiednio: *fields[0]* i *fields[3]*).

Przed wszystkim jednak importujemy moduły *sys* i *gv* i wczytujemy plik */proc/modules* za pomocą standardowej funkcji *readlines()*. Uzyskujemy w ten sposób listę wierszy, z których każdy zawiera sześć pól oddzielonych spacjami i zakończonych znakiem nowego wiersza.

Następnie ustawiamy właściwości grafu (*G*), wierzchołków (*N*) i krawędzi (*E*) za pomocą funkcji *gv.setv()*. Niektóre funkcje wyglądają znajomo, np. wiersz:

```
gv.setv(G, 'rankdir', 'LR')
```

Jest on oczywiście odpowiednikiem *rankdir="LR"*; z Listingu 4.

Ustawivszy właściwości poszczególnych elementów grafu, przystępujemy do części właściwej: przetwarzamy każdy wiersz (*rec*) dzieląc go na pola, przy czym rolę separatora pełni znak spacji (*split(' ')*). Następnie tworzymy wierzchołek o nazwie modułu, czyli pierwszego pola (*n = gv.node(G, fields[0])*). Podobnie postępujemy z polem czwartym (*fields[3]*), dzieląc je funkcją *split()*, tym razem jednak rolę separatora przyjmuje przecinek. Oczywiście musimy odrzucić polazawierające znak - bądź puste (*if (usedby != '-') & (usedby != '')*), po czym rysujemy krawędź od bieżącego węzła do wszystkich, których nazwy znajdują się w *fields[3]*.

Na koniec pozostaje nam określić, który rodzaj grafu wybierzemy (w tym wypadku będzie to *dot*) i renderujemy obrazek. Jeśli zamiast wyświetlać go na ekranie postanowimy zapisać go do pliku, możemy zastąpić ostatnią linijkę wierszem:

```
gv.render(G, 'png', 'test.png')
```

Milego eksperymentowania! ■

INFO

- [1] Strona domowa Graphviz: <http://www.graphviz.org/>
- [2] Dokumentacja Graphviz: <http://www.graphviz.org/Documentation.php>
- [3] Visitors, skrypt do tworzenia grafów od wiedz in stron WWW: <http://www.hpimg.org/visitors/>
- [4] VizierFX, biblioteka do renderowania grafów Graphviz napisana w Adobe Flex: <http://markandrewgoetz.com/vizierfx/sitors/>