Algorytmika Internetu

Krzysztof Diks

Instytut Informatyki, Uniwersytet Warszawski

diks@mimuw.edu.pl

Streszczenie

W sieci Internet znajduje się blisko 50 000 000 000 stron, liczba adresów IP zbliża się do 3 000 000 000. Jak to jest możliwe, że pomimo olbrzymiego rozmiaru Internetu jesteśmy w stanie niezmiernie szybko odnaleźć interesujące nas informacje, dzielić się filmami i muzyką, zdobywać przyjaciół w miejscach, do których nigdy nie dotarlibyśmy osobiście?

Okazuje się, że okiełznanie sieci wymagało wynalezienia i zaimplementowania odpowiednich algorytmów. O niektórych z nich jest mowa na tym wykładzie.

Spis treści

1 WSTĘP

Algorytmika jest działem informatyki, który zajmuje się projektowaniem i analizowaniem algorytmów. Komputery bez algorytmów zapisanych w postaci programów okazują się bezużyteczne. Żeby jednak zaprząc komputery do realizowania pożądanych przez nas zadań musimy być przekonani, że wykorzystywane algorytmy są poprawne, tzn. że dla danych spełniających określone kryteria, po wykonaniu algorytmu otrzymamy oczekiwane wyniki, oraz że są one możliwe do wykonania na dostępnym sprzęcie – wykorzystywane komputery dysponują pamięcią o wystarczającej pojemności, a obliczenia zakończą się w akceptowanym przez nas czasie. Zatem główne aspekty analizy algorytmów to ich poprawność i wydajność. Analizy algorytmów dokonujemy abstrahując od sprzętu, na którym będą wykonywane. Powszechnie przyjmuje się, że algorytmy szybkie, to takie, które wykonują się w czasie wielomianowym ze względu na rozmiar danych. Okazuje się, że w dobie Internetu, nawet algorytmy liniowe mogą być nie do wykorzystania na współczesnych komputerach z powodu olbrzymich rozmiarów danych, które muszą być przetwarzane. A jednak potrafimy wyszukiwać w tak monstrualnej sieci jaką jest Internet oraz poznawać jej strukturę. Jak to jest możliwe? Ten wykład to próba naszkicowania odpowiedzi na te pytania.

2 AUTORSKA, BARDZO KRÓTKA HISTORIA ALGORYTMÓW

Historia algorytmów sięga starożytności. Około 350 lat przed naszą erą pojawił się powszechnie dziś wykładany w szkole algorytm Euklidesa, służący do obliczania największego wspólnego dzielnika dwóch liczb naturalnych. W roku 1844 Francuz Gabriel Lamé udowodnił, że liczba dzieleń wymagana w pesymistycznym przypadku do znalezienia największego wspólnego dzielnika dwóch liczba algorytmem Euklidesa jest nie większa niż pięć razy liczba cyfra w zapisie dziesiętnym mniejszej z tych liczb, co oznacza, że działa on w czasie wielomianowym ze względu na rozmiar danych. Wynik Lamé możemy traktować jako pierwszą analizę złożoności czasowej algorytmu Euklidesa i początek teorii złożoności obliczeniowej.

Pojęcie algorytmu, choć w obecnych czasach intuicyjnie oczywiste, wymagało formalizacji po to, żeby algorytmy mogły być badane precyzyjnymi, matematycznymi metodami. W roku 1936 Alan Turing zaproponował teoretyczny model maszyny liczącej znanej od tego czasu jako maszyna Turinga. Mimo swej prostoty moc obliczeniowa maszyny Turinga rozumiana jako zdolność rozwiązywania problemów algorytmicznych przy zadanych zasobach (pamięci i czasie) odpowiada współczesnym komputerom.

W roku 1962 Charles Hoare zaproponował najpopularniejszy dziś algorytm sortowania – QuickSort. Algorytm QuickSort działa pesymistycznie w czasie kwadratowym, ale znakomicie zachowuje się dla przeciętnych (losowych) danych. Mniej więcej w tym samym czasie Jack Edmonds, zajmujący się algorytmami grafowymi, zdefiniował klasę P tych problemów algorytmicznych, które można rozwiązać w czasie wielomianowym. Jednocześnie okazało się, że dla wielu ważnych problemów optymalizacyjnych nie można było znaleźć algorytmów działających w czasie wielomianowym. Wytłumaczenie tego zjawiska pojawiło się w roku 1971, kiedy to Stephen Cook podał pierwszy, tak zwany problem NP-zupełny, wykazując, że jeśli potrafilibyśmy sprawdzać w czasie wielomianowym, czy dana formuła logiczna (zdaniowa) ma wartościowanie, dla którego przyjmuje wartość prawda, to wiele innych problemów, dla których poszukiwano bezskutecznie wielomianowego rozwiązania, dałoby się także rozwiązać w czasie wielomianowym. W tym samym roku Richard Karp wskazał osiem innych problemów NP-zupełnych. Obecnie lista takich problemów liczy tysiące pozycji. To, czy problemy NP-zupełne mają wielomianowe rozwiązanie, jest jednym z siedmiu tzw. problemów milenijnych. Za rozwiązanie każdego z nich zaoferowano milion dolarów. Jeden z tych problemów – hipoteza Poincaré – został już rozwiązany. Jedyne znane algorytmy dla problemów NP-zupełnych działają w czasie wykładniczo zależnym od rozmiaru

danych. Oznacza to, że dla danych dużych rozmiarów, ale wciąż spotykanych w praktyce, nie zawsze jest możliwe znalezienie rozwiązania z pomocą współczesnych komputerów ze względu na nadmiernie długi czas obliczeń.

Dzisiaj nie znamy algorytmu służącego do rozkładania liczby na czynniki będące liczbami pierwszymi w czasie wielomianowo zależnym od długości jej komputerowej reprezentacji. Fakt ten jest podstawą protokołu kryptograficznego RSA wynalezionego w roku 1977. Z drugiej strony, w roku 2002 wykazano, że to, czy liczba naturalna jest liczbą pierwszą, można sprawdzić w czasie wielomianowym. Czy powinniśmy się obawiać o protokół RSA?

3 TRZY PROSTE PROBLEMY ALGORYTMICZNE I ICH ROZWIĄZANIA

W tym rozdziale omówimy trzy proste problemy algorytmiczne, podamy ich rozwiązania oraz zanalizujemy złożoność obliczeniową zaproponowanych rozwiązań.

3.1 LIDER

Dane: dodatnia liczba całkowita n oraz ciąg liczb całkowitych $a[1]$, a[2], ..., $a[n]$.
Wynik: liczba całkowita x, która pojawia się w ciągu a więcej niż połowę razy (czyli taka liczba x, że $|\{i: a[i] = x\}| > n/2$), o ile takie x istnieje, w przeciwnym przypadku dowolny element z ciągu a.

Rozwiązanie, które proponujemy, wykorzystuje następujące spostrzeżenie. Niech u, v będą dwoma różnymi elementami ciągu a. Jeśli x jest liderem w a, to jest także liderem w ciągu a z usuniętymi elementami u i v. Innymi słowy, jeśli x jest liderem i każdy element w ciągu a o wartości różnej od x sparujemy z elementem o wartości różnej od w, to pozostaną tylko niesparowane elementy o wartościach równych x. Oto algorytm wykorzystujący powyższą ideę.

```
Algorytm Lider::
x := pierwszy element ciągu;
licz := 1;                                   //mamy licz niesparowanych elementów o wartości x
while nie koniec ciągu do
{
      y := kolejny element ciągu;
      if licz = 0 then
         { x := y; licz := 1}
      else if x = y then
            licz := licz +1
          else
            licz := licz - 1                 //parowanie
}
return x;
```

Proponujemy zastanowienie się, dlaczego ten algorytm jest poprawny. Skupmy się na jego złożoności. Za rozmiar zadania w tym przypadku przyjmujemy n – długość ciągu a. W każdym obrocie pętli pobieramy jeden element ciągu i wykonujemy na nim stałą liczbę operacji. Zatem złożoność czasowa jest liniowa, co w notacji asymptotycznej wyraża się formułą $O(n)$.

Zauważmy, że algorytm nie gwarantuje tego, że wynik x jest liderem. Żeby to stwierdzić, potrzebny jest jeszcze jeden przebieg przez dane i policzenie liczby wystąpień x w ciągu. W niektórych zastosowaniach do przetwarzania danych w Internecie drugi przebieg nie jest możliwy.

3.2 MNOŻENIE MACIERZY PRZEZ WEKTOR

Dane: liczba naturalna $n > 0$, kwadratowa macierz liczb rzeczywistych $A[1..n,1..n]$, wektor liczb rzeczywisty $x[1..n]$.
Wynik: wektor $y[1..n] = Ax$, gdzie $y[i] = A[i,1]*x[1] + A[i,2]*x[2] + ... + A[i,n]*x[n]$.

Algorytm mnożenia macierzy przez wektor wynika wprost z opisu wyniku.

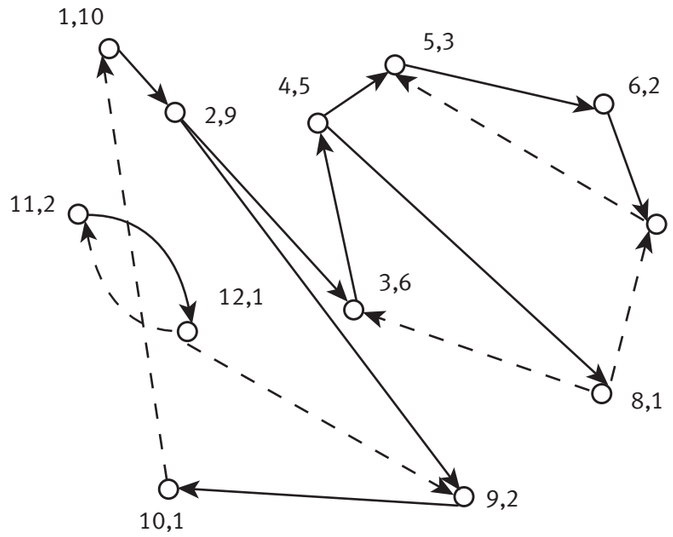
```
Algorytm Macierz_x_Wektor::
  for i := 1 to n do
  {
     y[i] := 0;
     for j := 1 to n do
        y[i] := y[i] + A[i,j]*x[j]
  }
  return y;
```

Zanalizujmy teraz czas działania powyższego algorytmu. Najdroższą operację w tym algorytmie jest operacja mnożenia dwóch liczb rzeczywistych. Wykonujemy n^2 takich mnożeń. Jeśli za rozmiar zadań przyjmiemy właśnie n^2, to nasz algorytm działa w czasie liniowym ze względu na rozmiar danych – liczbę elementów macierzy A. Algorytmy liniowe są uważane za szybkie. Rozważmy czas działania naszego algorytmu dla $n =$ 100 000 000. Dalej się okaże, że nie jest to rozmiar „wzięty z sufitu". Przyjmijmy dodatkowo, że w ciągu jednej sekundy możemy wykonać 10^8 mnożeń. Wówczas czas potrzebny na wykonanie wszystkich mnożeń w naszym algorytmie wyniósłby 100 000 000 sekund, czyli około 1600 dni! Jest jeszcze jeden problem. Gdybyśmy na zapisanie jednej liczby rzeczywistej przeznaczyli 8 bajtów, to musielibyśmy mieć pamięć o rozmiarze $8*10^{16}$ bajtów, czyli więcej niż petabajt! Tak więc, używając konwencjonalnych komputerów, nie bylibyśmy w stanie realnie rozwiązać tego zadania.

W praktyce często pojawiają się macierze, w których wiele elementów to zera. Zauważmy, że jeżeli $A[i,j] = 0$, to wykonanie instrukcji $y[i] := y[i] + A[i,j]*x[j]$ nie zmienia wartości $y[i]$. Używając technik listowych można pominąć mnożenia przez 0. Wówczas liczba mnożeń (i dodawań) jest proporcjonalna do liczby niezerowych elementów macierzy A. Oznaczmy liczbę niezerowych elementów macierzy A przez $Nz(A)$. Czas działania algorytmu w tym przypadku wynosi $O(\max(n,Nz(A))$, co, gdy $Nz(A)$ jest liniowe ze względu na n, daje algorytm mnożenia macierzy w czasie liniowo zależnym od n. Wówczas nawet dla $n =$ 100 000 000 można się spodziewać wyniku w rozsądnym czasie. Macierz, której liczba niezerowych elementów jest liniowa ze względu na wymiar macierzy, nazywamy macierzą rzadką. Używając technik listowych macierze rzadkie można reprezentować w pamięci o rozmiarze proporcjonalnym do liczby niezerowych elementów macierzy, co dla $n =$ 100 000 000 jest znacząco mniej niż petabajt.

3.3 SILNIE SPÓJNE SKŁADOWE

Powiemy, że graf skierowany jest silnie spójny, jeśli dla każdej pary węzłów u i v istnieją skierowane ścieżki z u do v i v do u. Silnie spójną składową skierowanego grafu G nazywamy każdy jego maksymalny (w sensie zwierania) silnie spójny podgraf. Silnie spójnymi składowymi w grafie z rysunku 1 są podgrafy rozpięte na węzłach identyfikowanych przez pierwsze liczby w parach: {1,2,9,10}, {3,4,8}, {5,6,7}, {11,12}.



Rysunek 1.
Graf skierowany

Specyfikacja zadania Silnie spójne składowe jest następująca:
Dane: $G=(V,E)$ – graf skierowany.
Wynik: funkcja $s: V \quad \{1,\ldots,|V|\}$ taka, że dla każdej pary węzłów u, v, $s(u) = s(v)$ wtedy i tylko wtedy, gdy istnieją ścieżki w grafie G z u do v i z v do u.

Złożoność algorytmów znajdowania silnie spójnych składowych w grafie zależy od reprezentacji grafu. Są dwie zasadnicze reprezentacje: tablicowa, w której w tablicy kwadratowej A, indeksowanej węzłami, mamy $A[u,v] = 1$, gdy istnieje krawędź z u do v, natomiast $A[u,v] = 0$, gdy takiej krawędzi nie ma. Niezależnie od liczby krawędzi w grafie rozmiar takiej reprezentacji wynosi $|V|^2$. Ponieważ liczba krawędzi w grafie skierowanym waha się od 0 do $|V|\times(|V|-1)$, reprezentacją uwzględniającą ten fakt są listy sąsiedztwa. W tej reprezentacji dla każdego węzła przechowujemy listę sąsiadów, do których prowadzą krawędzie z tego węzła – tzw. listę „w przód” – oraz listę sąsiadów, od których prowadzą krawędzie do tego węzła – tzw. listę „w tył”. Rozmiar reprezentacji grafu w postaci list sąsiedztwa wynosi $O(|V| + |E|)$ i jest ona szczególnie wygodna i oszczędna w przypadku grafów rzadkich, to znaczy takich, w których liczba krawędzi liniowo zależy od liczby węzłów. W algorytmie wyznaczania silnie spójnych składowych opisanym poniżej przyjmiemy tę drugą reprezentację.

Przedstawimy teraz krótko algorytm, którego pełny opis wraz z dowodem poprawności można znaleźć w książce [1]. Na potrzeby opisu załóżmy, że zbiór węzłów to $V = \{1,2, \ldots, |V|\}$. W celu znalezienia silnie spójnych składowych przeszukujemy graf G dwukrotnie w głąb. Za pierwszym razem wykorzystujemy listy w przód, gdy w następnym przeszukiwaniu korzystamy z list w tył. Przeszukując graf w przód, numerujemy węzły w kolejności odwiedzania i dla każdego węzła liczymy liczbę węzłów w poddrzewie przeszukiwania o korzeniu w tym węźle. Na rysunku 1 pierwsze liczby w parach odpowiadają numerom w kolejności przeszukiwania (tutaj identyfikujemy je z węzłami), a drugie liczby w parach mówią o rozmiarach poddrzew przeszukiwania w przód. Krawędzie lasu przeszukiwania w przód są narysowane ciągłą, pogrubioną kreską. Zauważmy, że numeracja

i liczba węzłów w poddrzewach pozwala na łatwe stwierdzenie, czy węzeł v jest w poddrzewie przeszukiwania o korzeniu w u. Wystarczy sprawdzić, czy numer v jest nie mniejszy od numeru u, ale mniejszy od numeru u plus liczba węzłów w poddrzewie o korzeniu u. Dla przykładu węzeł o numerze 7 jest w poddrzewie węzła o numerze 3, ponieważ $3 \le 7 < 3 + 6$; węzeł o numerze 11 nie jest w poddrzewie węzła o numerze 1, ponieważ $11 \ge 1 + 10$.

Podczas przeszukiwania grafu (po listach) w tył wykrywamy silnie spójne składowe. Identyfikatorem silnie spójnej składowej będzie węzeł o najmniejszym numerze z tej składowej. Dlatego w drugiej fazie przeglądamy węzły w kolejności rosnących numerów. Gdy napotkamy węzeł, który nie był jeszcze widziany w drugiej fazie, to rozpoczynamy z niego przeszukiwanie w tył, ale tylko po węzłach, które są w poddrzewie przeszukiwania w przód o korzeniu w węźle, od którego rozpoczęliśmy przeszukiwanie właśnie wykrywanej składowej. Wszystkie napotkane w ten sposób węzły będą należały do silnie spójnej składowej, której identyfikatorem będzie węzeł, od którego rozpoczynaliśmy przeszukiwanie. Rozważmy jeszcze raz graf z rysunku 1. Rozpoczynamy od węzła 1 i idziemy po krawędziach w tył. W ten sposób dochodzimy do węzła 10 i ponieważ jest on w poddrzewie w przód węzła 1, to zaliczamy go do silnie spójnej składowej o numerze 1. Z węzła 10 próbujemy węzeł 9 i też zaliczamy go do silnie spójnej składowej o numerze 1. Z węzła 9 próbujemy przejść do węzła 12, ale nie jest on w poddrzewie w przód węzła 1, więc nie należy do silnie spójnej składowej o identyfikatorze 1. Kolejny węzeł odwiedzany z węzła 9 to 2 i zaliczamy go do silnie spójnej składowej węzła numer 1. W ten sposób wykryjemy całą silnie spójną składową zawierającą węzeł 1. Wszystkie jej węzły są zaznaczone jako odwiedzone. Przeglądając kolejno węzły, docieramy do węzła 3, jako pierwszego nieodwiedzonego w tył, i teraz z niego odkrywamy silnie spójną składową o identyfikatorze 3 (węzły 3, 4, 8). Następnie zostanie wykryta składowa o identyfikatorze 5 (węzły 5, 6, 7), a na koniec o identyfikatorze 11 (węzły 11 i 12).

Oto formalny zapis omówionego algorytmu.

```
Algorytm SilnieSpójneSkładowe::
Faza I::
W_przód(v: węzeł)
{
  ost_nr := ost_nr + 1; nr[v] := ost_nr;                    // ost_nr – ostatnio nadany numer
  wezly[ost_nr] := v;                                       // porządkowanie węzłów według numerów
  for each u – węzeł na liście w przód węzła v do
    if nr[u] = 0 then   W_przód(u)                          // nr[u] = 0 oznacza, że węzeł nie został odwiedzony;
  w_poddrzewie[v] := ost_nr – nr[v] + 1                     // rozmiar poddrzewa w przód
}
Przeszukiwanie w przód::
  ost_nr := 0;
  for each węzeł v do nr[v] := 0;
  for each węzeł v do
    if nr[v] = 0 then W_przód(v);

Faza II::
W_tył(v: węzeł, id_s: 1..|V|)                               // id_s – id aktualnie wykrywanej składowej
{
  s[v] := id_s;
  for each węzeł u na liście w tył węzła v do
    if ((s[u] = 0)                                          // u nie był jeszcze odwiedzony
        AND                                                 // oraz
```

```
      (nr[id_s] < nr[u] < nr[id_s]+w_poddrzewie[id_s]))          // jest w poddrzewie id_s
    then W_tył(u, id_s);
}

Przeszukiwanie w tył::
  for each węzeł v do s[v] := 0;                                 // s[v] = 0 oznacza, że nie v był odwiedzony
  for i := 1 to |V| do
    if s[wezly[v]] = 0 then W_tył(wezly[v], wezly[v]);
```

Uważny czytelnik dostrzeże, że złożoność czasowa powyższego algorytmu wynosi O(|*V*|+|*E*|) i jest on liniowy ze względu na rozmiar grafu. Ten algorytm bardzo dobrze zachowuje się dla grafów z bardzo dużą liczbą węzłów i liniową względem niej liczbą krawędzi.

W następnych rozdziałach przekonamy się, że trzy przedstawione problemy mają silny związek z Internetem.

4 AUTORSKA, BARDZO KRÓTKA HISTORIA INTERNETU

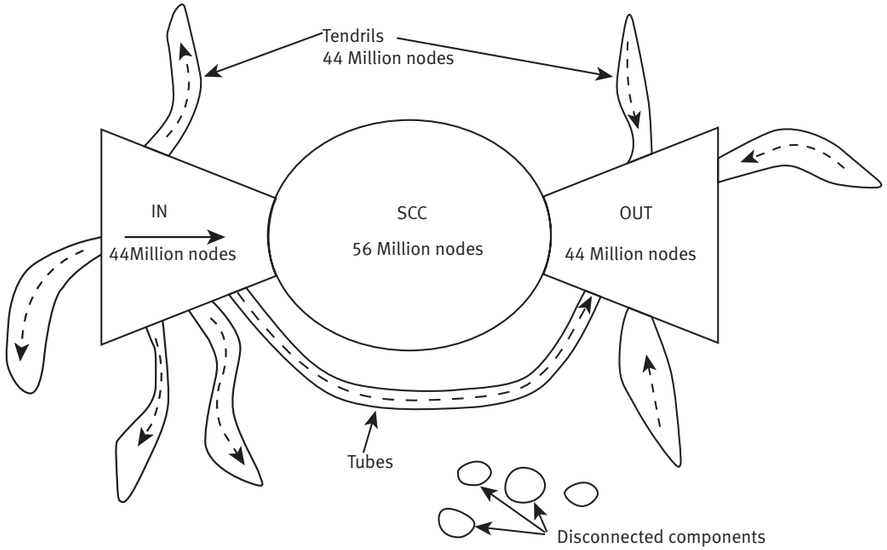
Internet to ogólnoświatowa sieć komputerowa, która z punktu widzenia przeciętnego użytkownika jest jednorodną siecią logiczną, w której węzły (komputery) są jednoznacznie identyfikowane przez swój adres. Początki Internetu sięgają końca lat sześćdziesiątych XX wieku, kiedy to powstaje sieć ARPANET łącząca cztery jednostki naukowe w Kalifornii i w stanie Utah. Sieć ARPANET była poletkiem doświadczalnym dla powstania i rozwoju powszechnie dziś używanej rodziny protokołów komunikacyjnych w sieci Internet – TCP/IP. W roku 1975 powstaje firma Microsoft, której rozwiązania przyczyniły się i nadal przyczyniają do upowszechniania świata komputerów wśród zwykłych ludzi. W roku 1976 pojawił się system operacyjny Unix – dziadek systemów operacyjnych z rodziny Linux. W tym też roku królowa Elżbieta wysłała swój pierwszy e-mail. W roku 1979 powstaje USENET – protoplasta powszechnych dziś sieci społecznościowych. W roku 1981 pojawia się komputer osobisty IBM PC. Bez komputerów osobistych, które możemy postawić na biurku i korzystać z ich dobrodziejstw w domu, globalizacja Internetu nie byłaby możliwa. W roku 1982, wraz z ustaleniem protokołów TCP/IP, nazwa Internet (z wielkiej litery) zaczyna się rozpowszechniać. W roku 1987 liczba hostów w Internecie przekracza 10 000, w roku 1989 przekroczona zostaje granica 100 000 hostów, w roku 1992 liczba hostów sięga miliona. Według Internet System Consortium, w lipcu 2010 roku liczba hostów wynosiła 768 913 036.

W roku 1990 sieć ARPANET przechodzi do historii. Rok 1991 to narodziny sieci WWW – ogólnoświatowej sieci powiązanych stron zawierających różnorodne treści multimedialne. Sieć jest rozwijana z wykorzystaniem Internetu, a strony w sieci WWW są identyfikowane poprzez tzw. adresy URL *(Uniform Resource Locator)*. W roku 1993 pojawia się MOSAIC – pierwsza graficzna przeglądarka stron WWW, a niedługo po niej Netscape. Rok 1994 to narodziny Yahoo. Największy obecnie gracz w Internecie to firma Google, która powstała w roku 1998. W roku 2004 startuje Facebook – najpopularniejsza obecnie w świecie sieć społecznościowa.
Z polskiej historii odnotujmy udostępnienie w roku 2000 komunikatora Gadu-Gadu, powstanie w roku 2001 serwisu gier Kurnik, a w roku 2006 – portalu społecznościowego Nasza-Klasa.

Czym charakteryzuje się Internet, a z punktu widzenia zwykłego użytkownika – sieć WWW? Jakie problemy algorytmiczne należy rozwiązać, żeby zasoby sieci stały się łatwo i szybko dostępne? Próbujemy odpowiedzieć na te pytania w następnym punkcie.

5 CHARAKTERYSTYKA INTERNETU

Zanim zajmiemy się problemami algorytmicznymi związanymi z Internetem, zastanówmy się, jak można opisać tę sieć. Skupimy się tutaj bardziej na sieci WWW, z której na co dzień korzystamy. Sieć WWW możemy przedstawić jako graf skierowany, którego węzłami są strony internetowe, natomiast dwie strony *V*, *W* są połączone krawędzią, jeśli na stronie *V* istnieje dowiązanie (link) do strony *W*. W roku 2000 grupa naukowców z AltaVista Company, IBM Almaden Research Center i Compaq Systems Research Center, opisała strukturę sieci WWW na podstawie danych zebranych przez **szperacze** (ang. *crawlers*) firmy Alta Vista [2]. Szperacze odwiedziły ponad 200 milionów stron, przechodząc po więcej niż półtora miliarda łączach. W wyniku analizy ustalono, że graf sieci WWW ma strukturę przedstawioną na rysunku 2.



Rysunek 2.
Sieć WWW [źródło: 2]

Co możemy wyczytać z tego rysunku? Gdybyśmy zapomnieli o orientacjach krawędzi, to prawie wszystkie strony należałyby do jednej spójnej składowej. Sieć jednak jest grafem skierowanym. Składają się na nią cztery główne części. Największa jest część środkowa SCC – silnie spójna składowa. W składowej SCC, z każdej strony możemy dojść do każdej innej, wykonując być może wiele kliknięć. Z dowolnej strony części IN można przejść przez SCC do dowolnej strony części OUT. Z części IN wychodzą tzw. **wici** (ang. *tendrils*), czyli strony, które są osiągalne z pewnych stron w IN. Podobnie wici prowadzą do OUT – strony, z których można dostać się do stron w OUT. Część stron z OUT jest bezpośrednio osiągalna ze stron IN za pomocą tzw. **rur** (ang. *tubes*). Po zanalizowaniu próbki sieci, Broder i inni doszli do ważnych wniosków [2]:

- sieć WWW jest skierowanym grafem rzadkim, tzn. liczba jej krawędzi liniowo zależy od liczby węzłów (badana sieć zawierała $2{,}7\times10^8$ stron (adresów URL) i $2{,}13\times10^9$ dowiązań,
- odległości w sieci są małe; w sieci zbadanej przez Brodera średnia długość ścieżki skierowanej wynosiła 16 – tyle kliknięć wystarczy, żeby dotrzeć do pożądanej, osiągalnej strony; gdybyśmy zapomnieli o orientacji krawędzi, to średnia długość ścieżki wyniosłaby 6.

Obecnie rozmiar Internetu jest dużo większy niż w roku 2000. Jak już wspomnieliśmy, w lipcu 2010 roku zarejestrowano 768 913 036 hostów. 1 grudnia 2010 w użyciu było 3 300 466 944 adresów IP obejmujących

240 krajów. Szacuje się, że na dzień 1 stycznia 2011 liczba stron poindeksowanych przez firmę Google wynosiła ponad 21 miliardów. Jaki stąd wniosek? Projektując algorytmy dla sieci WWW musimy wiedzieć, że działamy na sieci o miliardach węzłów, która szczęśliwie jest rzadka. Nie jest to zaskakujące. W sieci są miliardy stron, ale na każdej stronie może być od kilku do kilkudziesięciu dowiązań.

Sieć WWW niesie z sobą wiele wyzwań algorytmicznych, zarówno ze względu na ogrom Internetu, ale także dlatego, że Internet jest w pełni rozproszony i bez centralnej administracji. Do podstawowych problemów algorytmicznych związanych z siecią WWW można zaliczyć:

- wyszukiwanie i składowanie stron (zawartości);
- indeksowanie stron i przetwarzanie zapytań;
- odpowiadanie na zapytania w sposób zadowalający użytkownika;
- zgłębianie i analiza sieci WWW.

W następnym punkcie przedstawimy algorytm PageRank, który służy do nadawania rang stronom w taki sposób, żeby po znalezieniu stron w odpowiedzi na pytanie użytkownika wymienić je w kolejności od najważniejszych do najmniej ważnych. Uważny czytelnik natychmiast spostrzeże trudności w tak sformułowanym problemie. Strona, która jest ważna dla jednego użytkownika, może być mniej ważna dla innego. Czy istnieje jakiś w miarę obiektywny ranking stron, który byłby akceptowalny przez większość użytkowników Internetu? Takie pytanie zadali sobie twórcy firmy Google, Sergey Brin i Larry Page. Odpowiedzią na nie był algorytm PageRank, który przyczynił się do powstania i rozwoju Google.

6 ALGORYTM PAGERANK

Przedstawiony poniżej opis algorytmu PageRank został przygotowany już wcześniej i ukazał się w czasopiśmie „Delta” nr 8/2008 pod tytułem *Miara ważności* [3].

Każdy z nas choć raz w życiu użył wyszukiwarki Google. Czy zastanawialiśmy się, dlaczego wyszukiwarka Google podaje adresy stron będących odpowiedzią na zapytanie właśnie w takiej, a nie innej kolejności? Autor przeprowadził eksperyment i zanotował, co Google.pl dała w odpowiedzi na zapytanie matematyka. Oto 5 pierwszych adresów do stron, które autor otrzymał (6.01.2011, godz. 16.45):

1. www.matematyka.pl;
2. www.matematyka.pisz.pl;
3. pl.wikipedia.org/wiki/Matematyka;
4. matematyka.org;
5. www.math.edu.pl.

Przeglądarka Google podała, że wybrała je spośród 4 860 000 kandydatów. Dlaczego właśnie te strony uznano za najważniejsze? Jaka jest miara ważności (ranga) strony? Okazuje się, że podstawą analizy ważności stron w Google jest analiza połączeń w sieci WWW. Przypomnijmy, że sieć WWW jest grafem skierowanym, w którym węzłami są strony, a krawędzie odpowiadają dowiązaniom pomiędzy stronami – strona zawierająca adres internetowy innej strony jest początkiem krawędzi, a strona o danym adresie jest jej końcem. Czy można na podstawie samego grafu powiązań stron powiedzieć, które strony są ważniejsze, a które mniej?

Strona istotna, to strona interesująca. Strona interesująca, to taka, do której łatwo dotrzeć, ponieważ wiele innych stron na nią wskazuje. Na dodatek wiele spośród stron wskazujących na ważną stronę też jest ważnych i pasjonujących itd. Sergey Brin i Larry Page wyrazili to matematycznie w następujący sposób.

Załóżmy, że mamy n stron S_1, S_2, ..., S_n. Niech $w(S_i)$ będzie liczbą rzeczywistą dodatnią mierzącą ważność strony S_i. Wówczas określamy:

$$w(S_i)=\sum_{S_j \in We(S_i)} \frac{w(S_j)}{|Wy(S_j)|},$$

gdzie $We(S_i)$ to zbiór stron zawierających adres strony S_i (czyli zbiór początków krawędzi prowadzących do S_i), zaś $|Wy(S_j)|$ jest liczbą odnośników wychodzących ze strony S_j, czyli krawędzi prowadzących do stron ze zbioru $Wy(S_j)$. Wzór ten oznacza, że ważność strony mierzy się ważnością stron na nią wskazujących. Jeżeli przyjmiemy na początek, że wszystkie strony są jednakowo ważne i dla każdej z nich $w(S_i)=1/n$, gdzie n to liczba wszystkich stron, to ważność stron można obliczyć za pomocą następującej metody iteracyjnej (proces 1):

$$w_{k+1}(S_i)=\sum_{S_j \in We(S_i)} \frac{w_k(S_j)}{|Wy(S_j)|}.$$

Na powyższy proces można spojrzeć jak na błądzenie losowe po sieci WWW. Rozpoczynamy z dowolnej strony, a następnie z każdej oglądanej strony ruszamy losowo do jednej ze stron, do których adresy umieszczono na tej stronie. Jeżeli nasz proces będziemy powtarzali bardzo długo, to pewne strony będziemy oglądali częściej niż inne. Strony oglądane częściej są ważniejsze.

Niestety, może się zdarzyć, że przemieszczając się po stronach natrafimy na takie, z których nie ma wyjścia, np. strony zawierające zdjęcia. W takim przypadku zakładamy, że w następnym kroku wybierzemy losowo dowolną ze stron w Internecie. Teraz nasz proces iteracyjny ma następującą postać (proces 2):

$$w_{k+1}(S_i) = \sum_{S_j \in We(S_i)} \frac{w_k(S_j)}{|Wy(S_j)|} + \sum_{S_j \in K} \frac{w_k(S_j)}{n},$$

gdzie K oznacza zbiór wszystkich stron końcowych, czyli takich, które nie zawierają dowiązań do żadnych innych stron. Jesteśmy już blisko algorytmu (a raczej jego idei) firmy Google służącego do ustalania ważności stron.

Obserwując zachowanie użytkowników Internetu, Brin i Page zauważyli, że czasami porzucają oni bieżące przeszukiwanie i rozpoczynają nowe od (losowo) wybranej strony. Dlatego zmodyfikowali swój proces iteracyjny wprowadzając parametr α o wartości z przedziału (0,1). W każdej iteracji z prawdopodobieństwem α aktualne przeszukiwanie jest kontynuowane, natomiast z prawdopodobieństwem $(1 - \alpha)$ rozpoczynane jest nowe przeszukiwanie. Ostatecznie postać każdej iteracji jest następująca (proces 3):

$$w_{k+1}(S_i) = \alpha\left(\sum_{S_j \in We(S_i)} \frac{w_k(S_j)}{|S_j|} + \sum_{S_j \in K} \frac{w_k(S_j)}{n}\right) + (1-\alpha)\sum_{j=1}^{n} \frac{w_k(S_j)}{n}.$$

Bardziej doświadczeni czytelnicy pewnie już spostrzegli, że nasz proces iteracyjny jest procesem stochastycznym zbieżnym do stacjonarnego rozkładu prawdopodobieństwa. Celem kolejnych modyfikacji procesu było zapewnienie właśnie takiej zbieżności. Końcowy rozkład prawdopodobieństwa odpowiada ważności stron.

Zastanówmy się teraz, jaki jest związek opisanego procesu iteracyjnego z mnożeniem macierzy przez wektor. Sieć WWW na potrzeby tego procesu możemy opisać za pomocą macierzy H o rozmiarach n x n i zdefiniowanej następująco:

$$H[j,i] = \begin{cases} 1/|W(S_i)| & \text{jeśli istnieje dowiązanie z } S_i \text{ do } S_j \\ 0 & \text{w przeciwnym razie.} \end{cases}$$

Oznaczmy przez $w_k[1..n]$ wektor ważności stron po k-tej iteracji, tzn. $w_k[i]$ jest ważnością strony S_i po k-tej iteracji. Zgodnie z opisaną konwencją, na początku wszystkie strony są jednakowo ważne. Zatem dla każdej strony S_i mamy $w_0[i] = 1/n$. Przy takiej notacji proces 1 możemy zapisać następująco:

$$w_{k+1}=Hw_k.$$

Jest tak dlatego, ponieważ *i*-ty wiersz macierzy *H* zawiera odwrotności liczby dowiązań na stronach, z których prowadzą łącza do strony S_i.

Proces 2 uwzględnia fakt, że są strony niezawierające żadnych dowiązań. W macierzy *H* kolumny odpowiadające takim stronom zawierają same 0. Z takich stron do dalszego przeszukiwania wybieramy dowolną stronę z prawdopodobieństwem $1/n$. W notacji macierzowej wystarczy zatem zastąpić w macierzy *H* wszystkie kolumny zawierające same 0 przez kolumny, w których na każdej pozycji jest wartość $1/n$. Oznaczmy tak powstałą macierz przez *S*. Wówczas proces 2 ma postać:

$$w_{k+1}=Sw_k.$$

Proces 3 to zmodyfikowany proces 2, w którym z prawdopodobieństwem α kontynuujemy przeszukiwanie sieci z danej strony, a z prawdopodobieństwem $(1 - \alpha)$ przechodzimy do losowej strony. Oznacza to następującą modyfikację macierzy *S*: każdą pozycję macierzy *S* mnożymy przez α i dodajemy do tego $(1 - \alpha)/n$. Tak otrzymaną macierz oznaczmy przez *G*. Tak więc iteracyjny proces obliczania rang stron w macierzowej reprezentacji zapisuje się następująco:

$$w_{k+1}=Gw_k.$$

To jest zwykłe mnożenie macierzy przez wektor. Wiemy już, że macierz *G* reprezentuje sieć składającą się z ponad 21 miliardów węzłów. Niestety, chociaż sieć jest rzadka (zawiera kilkaset miliardów krawędzi), to macierz *G* nie jest rzadka – do każdego elementu macierzy *S* dodaliśmy bowiem $(1 - \alpha)/n$. Szczęśliwie, używając przekształceń algebraicznych możemy każdą iterację zapisać jako mnożenie rzadkiej macierzy *H* przez wektor plus dwie modyfikacje otrzymanego wektora:

1. $w_{k+1}=Hw_k$,
2. pomnóż każdy element wektora w_{k+1} przez α: $w_{k+1}=\alpha w_{k+1}$,
3. oblicz wartość β_k równą $1/n$ sumy: $(1 -\alpha)$ oraz sumy tych elementów wektora w_k, które odpowiadają stronom niezawierającym dowiązań do innych stron, pomnożonej przez α:

   $$\beta k= \frac{1}{n}\left((1 - \alpha) + (\Sigma_{S_i \in k}\, \alpha\, w_k[i])\right),$$

4. do każdego elementu wektora w_{k+1} dodaj β_k.

Jest jeszcze wiele pytań, które pozostają bez odpowiedzi. Jak szybko powyższy proces zbiega do końcowego rozkładu? Jak dobierać parametr α? Parametr α dobrany w Google wynosi 0.85. Macierz *G* z takim parametrem nosi nazwę macierzy Google. Okazuje się, że w tym przypadku wystarczy kilkanaście iteracji, aby policzyć wektor rang. Ale każda iteracja to setki miliardów operacji arytmetycznych i przetwarzanie olbrzymiego grafu. W jaki sposób możemy przyśpieszyć proces obliczeń? Zauważmy, że mnożąc macierz przez wektor można niezależnie pomnożyć każdy wiersz macierzy przez ten wektor. A gdyby zrobić to równolegle? To już jednak inna historia.

Zainteresowanych dokładną analizą algorytmu PageRank odsyłam do wspaniałej książki autorstwa Amy N. Langville i Carla D. Meyera, *Google's PageRank and Beyond: The Science of Search Engine Rankings* [4].

PODSUMOWANIE

Przedstawiliśmy trzy problemy algorytmiczne, które znajdują bezpośrednie zastosowanie w przetwarzaniu danych w Internecie. Algorytm obliczania silnie spójnych składowych może posłużyć do analizy sieci WWW. Mnożenie macierzy przez wektor wykorzystuje się do obliczania rang stron w sieci WWW tylko na podstawie struktury sieci. Co zadziwiające, również Lider znajduje zastosowanie. Wyobraźmy sobie ruter, który steruje ruchem pakietów w bardzo ruchliwej sieci. Administratorów sieci często interesuje, jakie pakiety generują największy ruch, a dokładniej, pod które adresy jest dostarczana największa liczba komunikatów. Jeśli pakietów są miliardy, to nie jesteśmy w stanie zapamiętać danych o wszystkich z nich, a następnie przetworzyć je. Algorytm przedstawiony w tym opracowaniu jest tak zwanym algorytmem strumieniowym – na bieżąco przetwarzany jest strumień danych i nie ma możliwości powrotu do historii. Jak się okazało można w ten sposób wskazać adres, który najbardziej obciąża sieć, jeśli tylko generuje on ponad połowę ruchu. Niewielka modyfikacja przedstawionego algorytmu umożliwia wskazywanie pakietów, z których każdy generuje np. co najmniej jedną dziesiątą ruchu.

Jaki morał płynie z naszych rozważań? Warto się uczyć algorytmiki. Nawet wydawałoby się dziwne problemy pojawiają się w tak gorących zastosowaniach, jak przetwarzanie Internetu. Niestety nie każdy algorytm, który wydaje się dobry w konwencjonalnych zastosowaniach (np. działa w czasie wielomianowym) nadaje się do przetwarzania takiego ogromu danych, jakie niesie ze sobą Internet. Nawet algorytmy liniowe mogą wymagać wielodniowych obliczeń. W przetwarzaniu danych w Internecie może pomóc równoległość, algorytmy strumieniowe i statystyczne. Algorytm PageRank pokazuje też, że bardzo dobre efekty z punktu widzenia użytkowników dają algorytmy, które naśladują ich naturalne zachowanie. Zachęcam czytelników do poszukiwania naturalnych problemów związanych z Internetem i rozwiązywania ich w stylu algorytmu PageRank.

LITERATURA

1. Banachowski L., Diks K., Rytter W., *Algorytmy i struktury danych*, WNT, Warszawa 2006
2. Broder A., Kumar R., Maghoul F., Raghavan P., Rajagopalan S., Stata R., Tomkins A., Wiener J.L., *Graph structure in the Web*, „Computer Networks" 2000, Vol. 33 (1-6), s. 309-320
3. Diks K., *Miara ważności*, „Delta" 8/2008
4. Langville A.N., Meyer C.D., *Google's PageRank and Beyond: The Science of Search Engine Rankings*, Princeton University Press, Princeton 2006