Algorytmy probabilistyczne
czyli zdaj się na ślepy (?) los

osłabiamy żądanie, że rozwiązanie problemu algorytmicznego musi być poprawne dla wszystkich dopuszczalnych danych,

ale:
- nie rezygnujemy całkowicie z wymagania poprawności algorytmu
- nie chcemy mieć tylko nadzieję, że algorytm jest poprawny
- chcemy mieć matematycznie potwierdzoną możliwość pominięcia przypadków niepoprawności algorytmu (muszą być mało prawdopodobne)

wykorzystujemy rachunek prawdopodobieństwa do opisu działania algorytmów, które korzystają z „rzutu symetryczną monetą"

Przykład - problem chińskich filozofów

Wszystkie rozwiązania, które są w pełni symetryczne i nie korzystają ze zmiennych globalnych prowadzą do zastoju.

Losowe zachowanie się filozofa (rzucającego monetą) – protokół algorytmiczny:
powtarza bez końca:

1. filozofuje, aż zgłodnieje
2. rzuca monetą, aby wybrać stronę, z której weźmie pałeczkę
3. czeka, aż będzie ona wolna i bierze ją
4. jeśli druga jest zajęta, to
   4.1. odkłada trzymaną pałeczkę
   4.2. wykonuje czynność 2.
5. w przeciwnym przypadku bierze drugą pałeczkę
6. najada się
7. odkłada pałeczki i wraca do 1.

Z **prawdopodobieństwem równym 1** przedstawione rozwiązanie nie prowadzi do zastoju!

$P\{$*żaden filozof nie może jeść*$\} = 0$ w nieskończonym przedziale czasu

Niestety podane rozwiązanie dopuszcza zagłodzenie!

Algorytmy probabilistyczne dla problemów rozwiązywanych sekwencyjnie

Podstawy teoretyczne sprawdzania czy liczba jest pierwsza

$\pi(n)$ – funkcja gęstości liczb pierwszych; $\pi(n)$ = liczba liczb pierwszych mniejszych bądź równych n
np. $\pi(10) = 4$, bo mamy 2, 3, 5, 7

Twierdzenie (o liczbach pierwszych) $\lim_{n\to\infty} \frac{\pi(n)}{n/\ln n} = 1$

np.

$\pi(10^9) = 50\ 847\ 478$, a $n/\ln n = 48\ 254\ 942$, czyli względna różnica $\frac{50847478 - 48254942}{50847478} \cdot 100\% \approx 5{,}1\ \%$

- zatem prawdopodobieństwo tego, że losowo wybrana liczba spośród n pierwszych naturalnych jest pierwsza można oszacować przez $\frac{1}{\ln n}$
- czyli w celu znalezienia liczby pierwszej o takiej samej długości co podana liczba n trzeba zbadać około $\ln n$ losowo wybranych liczb w pobliżu n

np.
znalezienie 100-cyfrowej liczby pierwszej wymagałoby sprawdzenia około $\ln 10^{100} \approx 230$ losowo wybranych liczb 100-cyfrowych
(a nawet tylko 115, jeśli wybierano by tylko liczby nieparzyste)

Najprostsze podejście do sprawdzenia czy liczba n jest pierwsza:

- dzielimy do skutku przez 2, 3, 5, 7 aż do $\lfloor \sqrt{n} \rfloor$ (parzyste można pominąć)
- jeśli *n* nie dzieli się bez reszty przez żadną z nich, to jest pierwsza

złożoność czasowa tego algorytmu wynosi $\boldsymbol{O}(\sqrt{n})$

(jeżeli liczba *n* jest zakodowana binarnie, to do jej zapisania potrzeba $N = \lceil \lg(n+1) \rceil$ bitów)

rozmiarem danej wejściowej (liczby *n*) jest jej długość *N*, a zatem funkcja złożoności $F(N) = \boldsymbol{O}(2^{N/2})$

- algorytm ma złożoność **wykładniczą**

---

równość *modulo* : $k = 1 \pmod n$ ⇔ reszta z dzielenia $\frac{k}{n}$ równa się 1

Twierdzenie (Fermata)

Jeśli *n* jest liczbą pierwszą, to $k^{n-1} = 1 \pmod n$ dla każdego $k \in \{1, 2, ..., n-1\}$

Zatem
znalezienie takiego $k \in \{1, 2, ..., n-1\}$, dla którego $k^{n-1} \neq 1 \pmod n$ oznacza, że liczba *n* nie jest pierwsza (jest liczbą złożoną)
– takie *k* nazywamy **świadectwem złożoności** liczby *n*

(dla nieparzystej liczby złożonej *n* liczba świadectw złożoności wynosi przynajmniej $\frac{n-1}{2}$)

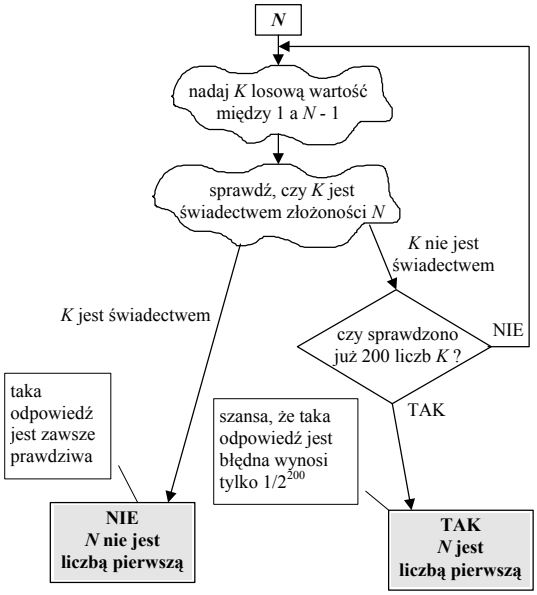
Przykład 1 - generowanie dużych liczb pierwszych

◊ liczb pierwszych jest nieskończenie wiele
◊ liczb pierwszych mniejszych od *N* jest rzędu $N / \log N$, np.
168 mniejszych od 1000, 78 500 mniejszych 1 000 000 itd.

Chcemy otrzymać nową liczbę pierwszą o 150 cyfrach:
losowo generujemy liczby i sprawdzamy czy są pierwsze - po 1000 próbach prawdopodobieństwo sukcesu wynosi ponad 90%,
ale sprawdzenie czy liczba jest pierwsza należy do klasy *NP*

Najlepszy algorytm deterministyczny ma złożoność $\boldsymbol{O(N^{O(\log \log N)})}$

Algorytmy probabilistyczne wykorzystują **świadectwa złożoności** liczby i:
- po znalezieniu takiego świadectwa dają odpowiedź „ liczba nie jest pierwsza”
- jeśli go nie znajdują, to po rozsądnym czasie przerywają szukanie z odpowiedzią „jest prawie pewne, że liczba jest pierwsza”



Przykład 2 - dopasowywanie wzorca

Wzorzec: boli występ $M = 11$

Tekst: Przypuśćmy, że chcemy stwierdzić, czy dany wzorzec symboli występuje w pewnym długim tekście

$N = 93$

JEST!

Prosty algorytm ma złożoność $\boldsymbol{O(N \times M)}$

Algorytm „odcisku palca”

- wyznaczamy dla każdego bloku M symboli liczbę zwaną „odciskiem palca”: reszta modulo K z dwójkowej (binarnej) postaci bloku, gdzie K jest losowo wybrana liczbą pierwszą o ok. N cyfrach dwójkowych (binarnych)
- porównujemy odciski kolejnych bloków z odciskiem wzorca
- algorytm przeszukuje cały tekst w czasie $\boldsymbol{O(N + M)}$
- prawdopodobieństwo, że jeden z bloków M symboli w tekście będzie miał tę sama resztę modulo K, co wzorzec złożony z M symboli, a ten blok będzie różny od wzorca, wynosi ok. $\mathbf{1 / N}$.

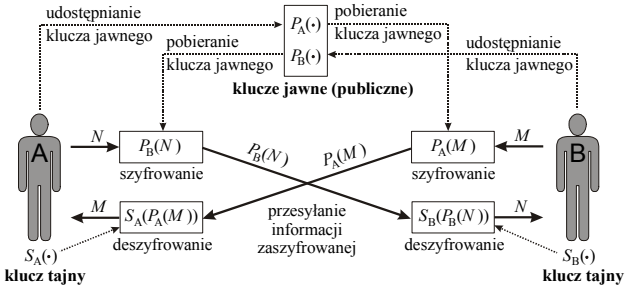
Jest to tzw. algorytm **Monte Carlo** - szybki (liniowy) i poprawny z dużym prawdopodobieństwem.
Tzw. algorytmy **Las Vegas** są zawsze poprawne i szybkie z dużym prawdopodobieństwem

Badania nad algorytmami probabilistycznymi i ich zastosowaniami

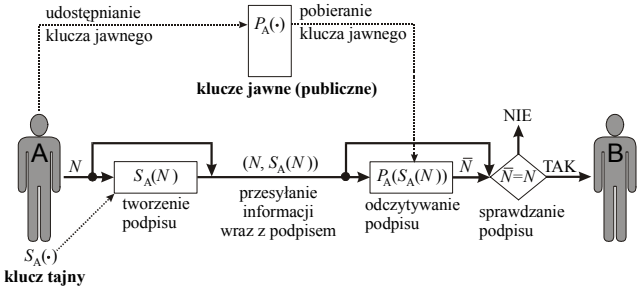
- na trudnościach w rozwiązywaniu pewnych problemów algorytmicznych, nawet w oparciu o podejście probabilistyczne, bazuje **kryptografia**
- nowatorskie podejście do szyfrowania - systemy kryptograficzne z kluczami publicznymi (pierwsza realizacja metody - kryptosystem RSA)
- szukanie efektywnych algorytmów probabilistycznych dla konwencjonalnych problemów algorytmicznych
- łączenie probabilizmu i równoległości
- powiązania pomiędzy algorytmami Monte Carlo i Las Vegas
- definiowanie i badanie probabilistycznych klas złożoności
- realizacja „losowości” wyboru na deterministycznym komputerze
- odwoływanie się do zjawisk fizycznych
- generatory liczb pseudolosowych (ciągu pseudolosowego nie można odróżnić od ciągu liczb prawdziwie losowych w czasie wielomianowym)

System kryptograficzny z kluczem jawnym

- każdy użytkownik tworzy dwa własne klucze: **jawny** i **tajny**
- klucz tajny jest chroniony, a jawny dostępny powszechnie
- klucz określa postać funkcji, którą można stosować do dowolnej wiadomości, przekształcając ją w postać zaszyfrowaną
- $\boldsymbol{P_A(\cdot)}$ – funkcja odpowiadająca kluczowi **jawnemu** użytkownika A
- $\boldsymbol{S_A(\cdot)}$ – funkcja odpowiadająca kluczowi **tajnemu** użytkownika A
- $P_A(\cdot)$ i $S_A(\cdot)$ powinny tworzyć parę funkcji wzajemnie odwrotnych: $M = S_A(P_A(M))$ i $M = P_A(S_A(M))$ dla dowolnej wiadomości M
- nikt poza użytkownikiem A nie potrafi wyznaczyć postaci funkcji $S_A(\cdot)$ w rozsądnym czasie (powinien to być problem trudno rozwiązywalny – o złożoności ponadwielomianowej)



Schemat działania systemu kryptograficznego



Schemat tworzenia podpisu cyfrowego

System kryptograficzny RSA (Rivest, Shamir, Adleman)

- klucz jawny i tajny oparty jest na liczbie $\boldsymbol{n = p \cdot q}$, gdzie p i q są dużymi liczbami pierwszymi (np. po 100 cyfr w zapisie dziesiętnym); liczba n jest podawana jako element klucza jawnego
- stworzyć klucz jest łatwo, bo dużą liczbę pierwszą można znaleźć z dużym prawdopodobieństwem w rozsądnym czasie
  (podany wcześniej algorytm Millera-Rabina ma złożoność czasową $\boldsymbol{O(s \cdot N)}$ dla prawdopodobieństwa niepoprawnego rozpoznania $1/2^s$,
  gdzie N jest liczbą bitów potrzebnych do binarnego zakodowania sprawdzanej liczby pierwszej)
- ale złamać szyfr jest bardzo trudno, bo, aby odtworzyć postać funkcji $S_A(\cdot)$, trzeba dokonać rozkładu liczby $\boldsymbol{n}$ na iloczyn dwóch liczb pierwszych, a problem znalezienia czynnika $\boldsymbol{p}$ ma złożoność $\boldsymbol{O(2^N)}$