Teoretyczne podstawy informatyki

Wykład 9:

Rozwiązywanie rekurencji
Algorytmy probabilistyczne

Rekurencja

Rekurencje były badane już w 1202 roku przez L. Fibonacciego, od którego nazwiska pochodzi nazwa liczb Fibonacciego.

A. De Moivre w 1730 roku wprowadził pojęcie funkcji tworzących do rozwiązywania rekurencji.

Czas działania programu

- Dla konkretnych danych wejściowych jest wyrażony liczba wykonanych prostych (elementarnych) operacji lub “kroków”. Jest dogodne zrobienie założenia że operacja elementarna jest maszynowo niezależna.
- Każde wykonanie **i**-tego wiersza programu jest równe c_i, przy czym c_i jest stałą.
- Kiedy algorytm zawiera **rekurencyjne wywołanie samego siebie**, jego czas działania można często opisać zależnością rekurencyjną wyrażającą czas dla problemu rozmiaru **n** za pomocą czasu dla podproblemów mniejszych rozmiarów.
- Możemy wiec użyć narzędzi matematycznych aby **„rozwiązać rekurencje”** i w ten sposób otrzymać oszacowania czasu działania algorytmu.

Rekurencja dla alg. typu “dziel i zwyciężaj”

- Rekurencja odpowiadającą czasowi działania algorytmu typu “dziel i zwyciężaj” opiera się na podziale jednego poziomu rekursji na trzy etapy.
  - Niech **T(n)** będzie czasem działania dla jednego problemu rozmiaru **n**.
  - Jeśli rozmiar problemu jest odpowiednio mały, powiedzmy **n ≤ c** dla pewnej stałej **c**, to przyjmujemy że jego rozwiązanie zajmuje stały czas, co zapiszemy jako **Θ(1)**.
  - Załóżmy że dzielimy problem na **a** podproblemów, każdy rozmiaru **n/b**. Jeśli **D(n)** jest czasem dzielenia problemu na podproblemy, a **C(n)** jest czasem scalania rozwiązań podproblemów w pełne rozwiązanie dla oryginalnego problemu, to otrzymujemy rekurencje

T(n) = Θ(1) **jeśli n ≤ c**

T(n) = a T(n/b) + D(n) + C(n) w przeciwnym przypadku

Rekurencja dla alg. “dziel i zwyciężaj”

- **Przykład:** algorytm sortowania przez scalanie
  - **dziel:** znajdujemy środek przedziału, zajmuje to czas stały **$D(n)=\Theta(1)$,**
  - **zwyciężaj:** rozwiązujemy rekurencyjnie dwa podproblemy, każdy rozmiaru **$n/2$**, co daje czas działania **$2\,T(n/2)$**,
  - **połącz:** działa w czasie **$\Theta(n)$**, a wiec **$C(n)=\Theta(n)$**.
- **Ostatecznie:**
  - **$T(n) = \Theta(1)$** **jeśli $n=1$**
  - **$T(n) = 2\,T(n/2) + \Theta(1) + \Theta(n)$** **jeśli $n>1$**
- Rozwiązaniem tej rekurencji jest **$T(n) = \Theta(n \log n)$**.

  Jak to sprawdzić?

Metody rozwiązywania rekurencji

Metoda podstawiania:

- zgadujemy oszacowanie, a następnie dowodzimy przez indukcję jego poprawność.

Metoda iteracyjna:

- przekształcamy rekurencję na sumę, korzystamy z technik ograniczania sum.

Metoda uniwersalna::

- stosujemy oszacowanie na rekurencję mające postać
  T(n) = a T(n/b) + f(n), gdzie **a≥1**, **b>1**, a **f(n)** jest daną funkcją. Następnie korzystamy z gotowego rozwiazania.

Metoda podstawiania

- Polega na **zgadnięciu postaci rozwiązania, a następnie wykazaniu przez indukcję, że jest ono poprawne**. Trzeba też znaleźć odpowiednie stałe. Bardzo skuteczna ale stosowana tylko w przypadkach kiedy łatwo jest przewidzieć postać rozwiązania.

Metoda podstawiania

Przykład:

- Postać rekurencji:
  $T(n) = 2T(n/2) + n$
- Zgadnięte rozwiązanie:
  $T(n) = \Theta(n \log n)$
- Podstawa:
  $n=2; T(1)=1; T(2)=4;$
- Indukcja:
  $T(n) \leq 2\,(c(n/2)\log(n/2)) + n \leq c\,n \log(n/2) + n$

  $T(n) \leq c\,n \log(n/2) + n = cn \log(n) - cn \log(2) + n$

  $T(n) \leq cn \log(n) - cn \log(2) + n = cn \log(n) - cn + n$

  $T(n) \leq cn \log(n) - cn + n \leq cn \log(n)$

  spełnione dla **$c>=1$**;

Metoda iteracyjna

- Polega na **rozwijaniu (iterowaniu) rekurencji** i wyrażanie jej jako sumy składników zależnych tylko od **n** warunków brzegowych. Następnie mogą być użyte techniki sumowania do oszacowania rozwiązania.

Metoda iteracyjna

Przykład:

- Postać rekurencji:
  $T(n) = 3T(n/4) + n$
- Iterujemy:
  $T(n) = n + 3T(n/4) = n + 3((n/4) + 3T(n/16))$
  $= n + 3 (n/4) + 9T(n/16)$
  $= n + 3 n/4 + 9 n/16 + 27 T(n/64)$
- Iterujemy tak długo aż osiągniemy warunki brzegowe.
  Składnik **i**-ty w ciągu wynosi **$3^i\ n/4^i$**.
  Iterowanie kończymy, gdy **$n=1$** lub **$n/4^i = 1$** (czyli **$i > \log_4(n)$**).
  $T(n) \leq n + 3n/4 + 9n/16 + 27n/64 + \ldots + 3^{\log_4 n}\ \Theta(1)$
  $T(n) \leq 4n + 3^{\log_4 n}\ \Theta(1) = \Theta(n)$

Metoda iteracyjna

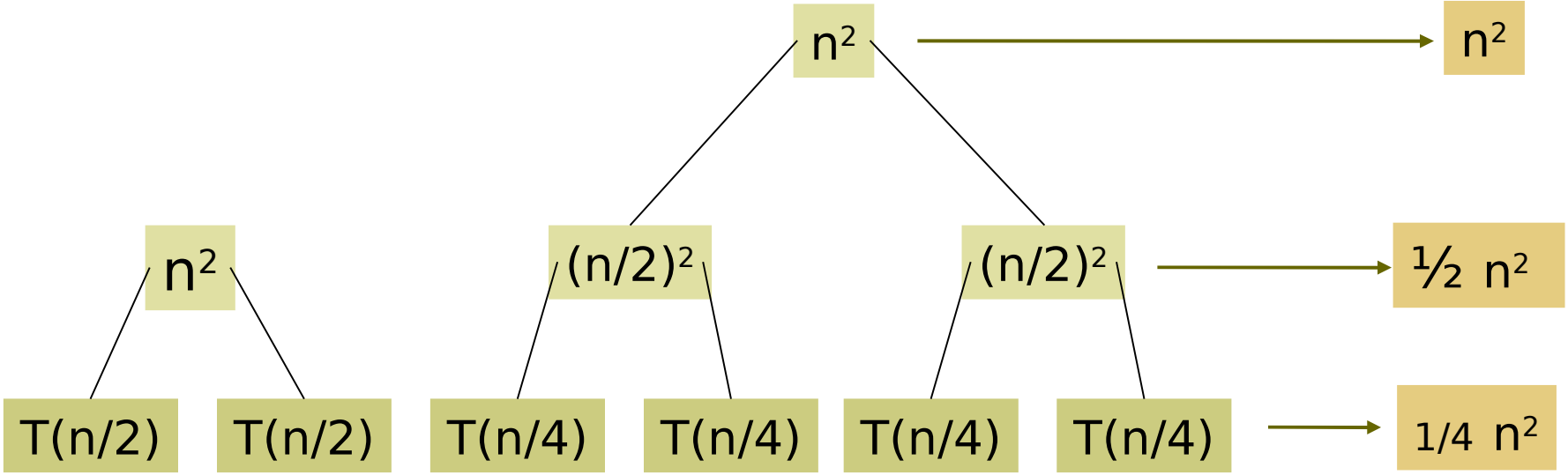
- Metoda iteracyjna jest zazwyczaj związana z dużą ilością przekształceń algebraicznych, więc zachowanie prostoty nie jest łatwe.
- Punkt kluczowy to skoncentrowanie się na dwóch parametrach:
  - liczbie iteracji koniecznych do osiągnięcia warunku brzegowego
  - oraz sumie składników pojawiających się w każdej iteracji.

Drzewa rekursji

- Pozwalają w dogodny sposób zilustrować rozwijanie rekurencji, jak również ułatwia stosowanie aparatu algebraicznego służącego do rozwiązywania tej rekurencji.
- Szczególnie użyteczne gdy rekurencja opisuje algorytm typu “dziel i zwyciężaj”.

Drzewo rekursji dla alg. „dziel i zwyciężaj

$$T(n) = 2\,T(n/2) + n^2$$

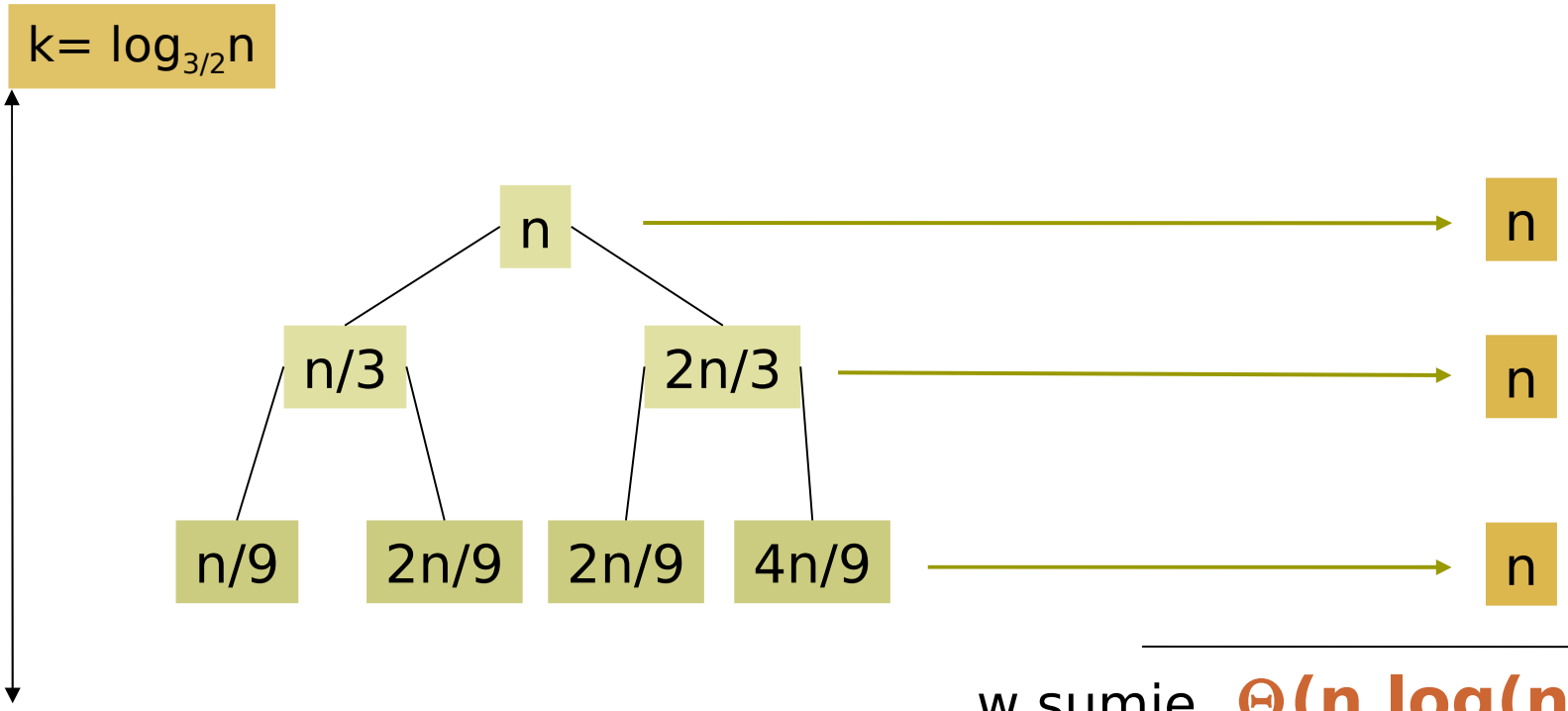


ostateczny wynik: $T(n) = \Theta(n^2)$

w sumie: $\Theta(n^2)$

Drzewa rekursji

T(n) = T(n/3) + T(2n/3) + n



Dzielimy tak długo aż **$n\,(2/3)^k = 1 \Rightarrow n = (3/2)^k$**

$\Rightarrow k = \log_{3/2}(n) = \log_{3/2}(2) * \log(n)$

ostateczny wynik: **T(n) = Θ(n log(n))**

Metoda rekurencji uniwersalnej

- Metoda rekurencji uniwersalnej podaje “uniwersalny przepis” rozwiązywania równania rekurencyjnego postaci:

  T(n) = a T(n/b) + f(n)

- gdzie **a≥1** i **b>1** są stałymi, a **f(n)** jest funkcja asymptotycznie dodatnia.
- Za wartość **(n/b)** przyjmujemy najbliższą liczbę całkowitą (mniejsza lub większą od wartości dokładnej).

Metoda rekurencji uniwersalnej

- Rekurencja opisuje czas działania algorytmu, który dzieli problem rozmiaru **n** na **a** problemów, każdy rozmiaru **n/b**, gdzie **a** i **b** są dodatnimi stałymi.
- Każdy z **a** problemów jest rozwiązywany rekurencyjnie w czasie **T(n/b)**.
- Koszt dzielenia problemu oraz łączenia rezultatów częściowych jest opisany funkcja **f(n)**.

Twierdzenie o rekurencji uniwersalnej

- Niech **$a \geq 1$** i **$b > 1$** będą stałymi, niech **$f(n)$** będzie pewną funkcją i niech **$T(n)$** będzie zdefiniowane dla nieujemnych liczb całkowitych przez rekurencje

$$T(n) = a\,T(n/b) + f(n)$$

gdzie **(n/b)** oznacza najbliższą liczbę całkowitą do wartości dokładnej **n/b**.

- Wtedy funkcja **$T(n)$** może być ograniczona asymptotycznie w następujący sposób:
  - Jeśli **$f(n) = O(n^{\log_b a - \varepsilon})$** dla pewnej stałej **$\varepsilon > 0$**, to **$T(n) = \Theta(n^{\log_b a})$**.
  - Jeśli **$f(n) = \Theta(n^{\log_b a})$** to **$T(n) = \Theta(n^{\log_b a} \log n)$**.
  - Jeśli **$f(n) = n^{\log_b a + \varepsilon}$** dla pewnej stałej **$\varepsilon > 0$** i jeśli **$af(n/b) \leq cf(n)$** dla pewnej stałej **$c < 1$** i wszystkich dostatecznie dużych **n**, to **$T(n) = \Theta(f(n))$**

Twierdzenie o rekurencji uniwersalnej

“Intuicyjnie…”:

- W każdym z trzech przypadków porównujemy funkcje **f(n)** z funkcją $n^{\log_b a}$. Rozwiązanie rekurencji zależy od większej z dwóch funkcji.
- Jeśli funkcja $n^{\log_b a}$ jest większa, to rozwiązaniem rekurencji jest:

  $T(n) = \Theta(n^{\log_b a})$
- Jeśli funkcje są tego samego rzędu, to mnożymy przez **log n** i rozwiązaniem jest:

  $T(n) = \Theta(n^{\log_b a} \log n) = T(n) = \Theta(f(n) \log n)$.
- Jeśli **f(n)** jest większa, to rozwiązaniem jest:

  $T(n) = \Theta(f(n))$

Przykład

$T(n) = 9\,T(n/3) + n$

$a=9,\ b=3,$

$f(n)=n,$

a zatem $n^{\log_b a} = n^{\log_3 9} = \Theta(n^2)$.

Ponieważ $f(n)=O(n^{\log_3 9-\varepsilon})$, gdzie $\varepsilon=1$, możemy zastosować przypadek 1 z twierdzenia i wnioskować że rozwiązaniem jest $T(n) = \Theta(n^2)$.

Przykład

$T(n) = T(2n/3) + 1$

$a=1, b=3/2,$

$f(n)=1,$

a zatem **$n^{\log_b a} = n^{\log_{3/2} 1} = n^0 = 1$**.

Stosujemy przypadek 2, gdyż

$f(n) = \Theta(n^{\log_b a}) = \Theta(1)$,

a zatem rozwiązaniem rekurencji jest

$T(n) = \Theta(\log n)$.

Przykład

$T(n) = 3T(n/4) + n \log n$

$a=3$, $b=4$,
$f(n)=n \log n$,
a zatem **$n^{\log_b a} = n^{\log_4 3} = O(n^{0,793})$**.

Ponieważ **$f(n) = \Omega(n^{\log_4 3+\varepsilon})$** , gdzie **$\varepsilon \approx 0.2$**, wiec stosuje się tutaj przypadek 3, jeśli możemy pokazać ze dla **f(n)** zachodzi warunek regularności.
Dla dostatecznie dużych **n**:

$af(n/b) = 3(n/4)\log(n/4) \leq (3/4)n\log(n) = c\, f(n)$

dla **$c=3/4$**.

Warunek jest spełniony i możemy napisać że rozwiązaniem rekurencji jest **$T(n) = \Theta(n \log n)$**.

Przykład

$T(n) = 2T(n/2) + n \log n$

$a=2, b=2,$
$f(n)=n \log n,$

a zatem **$n^{\log_b a} = n$**.

Wydaje się że powinien to być przypadek 3, gdyż **$f(n)=n \log n$** jest asymptotycznie większe niż **$n^{\log_b a} = n$**, ale nie wielomianowo większy.

Stosunek **$f(n)/ n^{\log_b a} = (n \log n)/n = \log n$** jest asymptotycznie mniejszy niż **n^{ε}** dla każdej dodatniej stałej **ε**.

W konsekwencji rekurencja ta “wpada” w lukę miedzy przypadkiem 2 i 3.

Prawdopodobieństwo i algorytmy probabilistyczne

Teoria prawdopodobieństwa

- Teoria prawdopodobieństwa, szeroko stosowana we współczesnej nauce, ma również wiele zastosowań w informatyce, np.:
  - szacowanie czasu działania programów dla przypadków ze średnimi, czyli typowymi danymi wejściowymi,
  - wykorzystanie do projektowania algorytmów „podejmujących decyzje” w niepewnych sytuacjach, np. najlepsza możliwa diagnoza medyczna na podstawie dostępnej informacji,
  - algorytmy typu Monte Carlo,
  - różnego rodzaju symulatory procesów,
  - **prawie zawsze „prawdziwe” rozwiązania.**

Teoria prawdopodobieństwa

- **Przestrzeń probabilistyczna Ω:**

  Skończony zbiór punktów, z których każdy reprezentuje jeden z możliwych wyników doświadczenia. Każdy punkt **x** jest związany z taką nieujemną liczbą rzeczywistą zwaną prawdopodobieństwem **x**, że suma prawdopodobieństw wszystkich punktów wynosi **1**.
  Istnieje także pojęcie nieskończonych przestrzeni probabilistycznych ale nie mają one większego zastosowania w informatyce.

- **Zdarzenie E:**

  Podzbiór punktów w przestrzeni probabilistycznej. Prawdopodobieństwo zdarzenia, **P(E)**, jest sumą prawdopodobieństw punktów należących do tego zdarzenia.

- **Dopełnienie zdarzenia E czyli Ē :**

  Zbiór punktów przestrzeni probabilistycznej które nie należą do zdarzenia **E**. **P(E) + P(Ē) = 1**.

Prawdopodobieństwo warunkowe

- Prawdopodobieństwem warunkowym zajścia zdarzenia **F** pod warunkiem zajścia zdarzenia **E**, gdzie **P(E) > 0** nazywamy liczbę:
  $P(F|E) = P(E \cap F) / P(E)$.
- Jest to iloraz prawdopodobieństwa części wspólnej zdarzeń **E, F** i prawdopodobieństwa zdarzenia **E**.
- Zdarzenia **E, F** nazywamy niezależnymi jeśli zachodzi:
  $P(E \cap F) = P(E) \bullet P(F)$
- W przeciwnym wypadku zdarzenia są zależne.
- Dla zdarzeń niezależnych **E, F** zachodzi:
  $P(F|E) = P(F)$

Przykłady

- **Zdarzenia niezależne:**
  Rzucamy dwoma kostkami, wyrzucenie liczby „1” na pierwszej kostce (zdarzenie E) nie wpływa na możliwość pojawienia się liczby „1” na drugiej kostce (zdarzenie F). **$P(F|E) = P(F)$**

- **Zdarzenia zależne:**
  Ciągniemy dwa razy kartę z talii kart. Wyciągniecie jako pierwszej karty asa (zdarzenie E), wpływa na możliwość wyciągnięcia jako drugiej karty asa (zdarzenie F). **$P(F|E) \neq P(F)$.**

- W niektórych sytuacjach liczenie prawdopodobieństw jest łatwiejsze jeżeli podzielimy przestrzeń probabilistyczna na rozdzielne obszary **$R_1, R_2, \ldots, R_k$**.
  Wówczas **$P(E) = \Sigma_{i=0}^{k} P(E|R_i) \bullet P(R_i)$.**

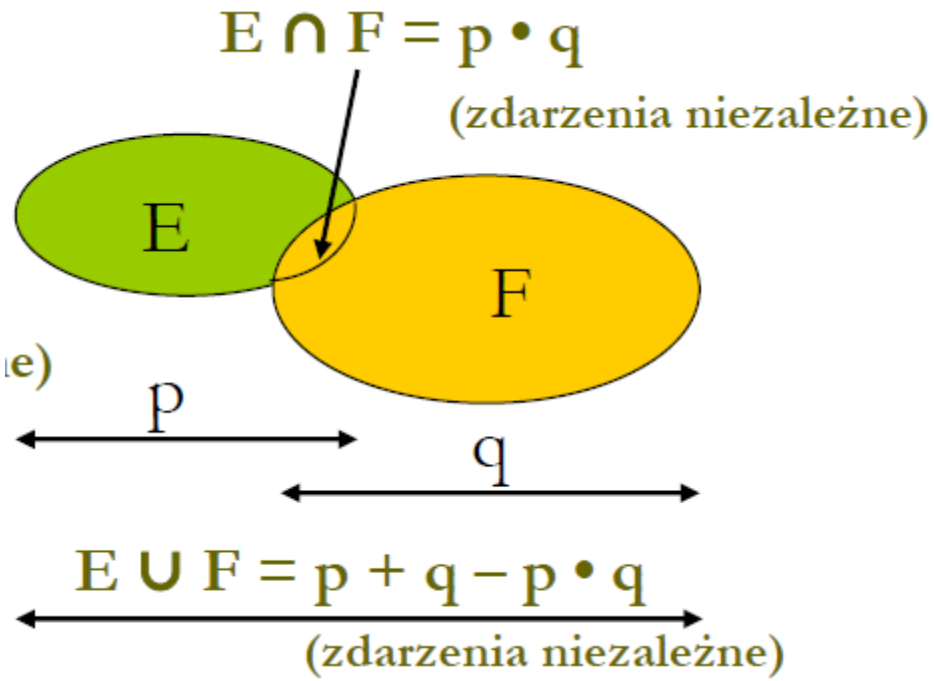
Przykład z kartami

- Ciągniemy dwie karty z talii **52** kart. Liczba możliwych wyników tego doświadczenia (czyli wariacji bez powtórzeń) wynosi $|\Omega| = 52 * 51 = 2652$.
- Oznaczmy poprzez **E** zdarzenie polegające na wyciągnieciu jako pierwszej karty As'a.
  $|E| = 4 * 51 = 204$.
  $P(E) = |E|/|\Omega| = 204/2652 = 1/13$.
- Prawdopodobieństwo wyciągnięcia As'a jako drugiej karty, (**zdarzenie F**), jeżeli pierwsza wyciągnięta karta to był As jest **$P(F|E) = P(E \cap F) / P(E) = 4 \bullet 3/204 = 1/17$**.
  $P(E) \bullet P(F|E) = 1/13 \bullet 1/17 = 1/221$.

- Podzielmy przestrzeń na dwa obszary:
  - **R_1** - pierwszą karta jest As, **$|R_1| = 4*51 = 204$**.
  - **R_2** - pierwszą karta nie jest As, **$|R_2| = 2652-204 = 2448$**.
  - **$P(E \cap F)=P((E \cap F)|R1) \bullet P(R1) + P((E \cap F)|R2) \bullet P(R2)$**
  - **$P(E \cap F| R_2) = 0$**
  - **$P(E \cap F| R_1) = 4 \bullet 3/ 4 \bullet 51 = 1/17$**
  - **$P(R_1) = 204/ 2652 = 1/13$**
  - **$P(E \cap F) = P((E \cap F)|R_1) \bullet P(R_1) + P((E \cap F)|R_2) \bullet P(R_2) = 1/17 \bullet 1/13 + 0 \bullet P(R_2) = 1/221$.**

- Gdybyśmy po wyciągnięciu pierwszej karty zwracali ją z powrotem do talii, to mielibyśmy
  $P(F|E) = P(F) = 1/13$ (zdarzenia niezależne).
- Wówczas **$P(E) \bullet P(F|E)=1/13 \bullet 1/13=1/169$**.

Reguły związane z wieloma zdarzeniami



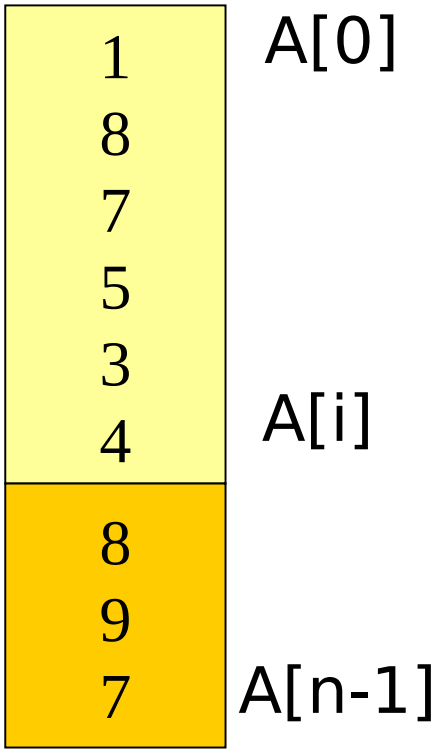
- W zastosowaniach czasem akceptujemy że nie możemy wyznaczyć dokładnie prawdopodobieństw oraz zależności między zdarzeniami. Potrafimy tylko wskazać sytuacje najmniej lub najbardziej prawdopodobne.
- **Zastosowanie: różnego typu diagnostyka**

Oczekiwane wartości obliczeń

- Przypuśćmy, że mamy pewną funkcję określoną na przestrzeni probabilistycznej **f(x)**. Wartość oczekiwana tej funkcji po wszystkich punktach przestrzeni
  E(f) = Σ f(x) P(x).

- Mamy tablicę **n** liczb całkowitych, sprawdzamy czy jakaś liczba całkowita „**x**” jest elementem tej tablicy. Algorytm przegląda całą tablice, po napotkaniu **A[i] = x** kończy działanie.
  - Jeżeli **A[0] = x** to algorytm ***O*(1)**
  - Jeżeli **A[n-1] = x** to algorytm ***O*(n)**

- $E(f) = \sum_{i=0}^{n-1} (c\ i + d) \bullet (1/n) = c \bullet (n-1) / 2 + d$
- $E(f) \sim c \bullet n/2$ dla dużego n



Algorytmy wykorzystujące prawdopodobieństwo

- Jest bardzo wiele różnych typów algorytmów wykorzystujących prawdopodobieństwo.
- Jeden z nich to tzw. **algorytmy Monte-Carlo** które wykorzystują liczby losowe do zwracania albo wyniku pożądanego („prawda”), albo żadnego („nie wiem”). Wykonując algorytm **stałą** liczbę razy, możemy rozwiązać problem, dochodząc do wniosku, że jeśli żadne z tych powtórzeń nie doprowadziło nas do odpowiedzi „prawda”, to odpowiedzią jest „fałsz”.
  Odpowiednio dobierając liczbę powtórzeń, możemy dostosować prawdopodobieństwo niepoprawnego wniosku „fałsz” do tak niskiego poziomu, jak w danym przypadku uznamy za konieczne.

- **Nigdy** jednak nie osiągniemy prawdopodobieństwa popełnienia błędu na poziomie **zero**.

Co to są liczby losowe?

- Mówimy, że wyniki pewnych doświadczeń są **losowe,** co oznacza ze wszystkie możliwe wyniki są równie prawdopodobne.
- Przykładowo, jeżeli rzucamy normalną (prawidłową) kostką do gry to zakładamy że nie ma możliwości fizycznego kontrolowania wyniku tego rzutu w taki sposób aby jeden wynik był bardziej prawdopodobny od drugiego.
- Podobnie zakładamy że mając uczciwie potasowana talie kart, nie możemy wpłynąć na wynik rozdania- prawdopodobieństwo otrzymania w rozdaniu każdej karty jest identyczne.

Co to są liczby losowe?

- **Wszystkie** generowane przez komputer **losowe** sekwencje są wynikiem działania specjalnego rodzaju algorytmu zwanego **generatorem liczb losowych** (ang. random number generator). Zaprojektowanie takiego algorytmu wymaga specjalistycznej wiedzy matematycznej.
  Przykład prostego generatora który całkiem dobrze sprawdza się w praktyce to tzw. "**liniowy generator kongurencyjny**".
  Wyznaczamy **stałe a ≥ 2, b ≥ 1, x_0 ≥ 0** oraz **współczynnik m > max(a, b, x_0)**.
  Możemy teraz wygenerować sekwencje liczb x_1, x_2, ... za pomocą wzoru:

  $$x_{n+1} = (a\, x_n + b) \bmod (m)$$

- Dla właściwych wartości stałych a, b, m oraz x_0, sekwencja wynikowa będzie wyglądała na losową, mimo ze została ona wygenerowana przy użyciu konkretnego algorytmu i na podstawie "jądra" x_0.
- **Dla szeregu zastosowań istotna jest odtwarzalność sekwencji liczb losowych.**

Algorytmy wykorzystujące prawdopodobieństwo

- Mamy pudełko w którym jest **n**-procesorów, nie mamy pewności czy zostały przetestowane przez producenta. Zakładamy że prawdopodobieństwo że procesor jest wadliwy (w nieprzetestowanym pudelku) jest 0.10.
- **Co możemy zrobić aby potwierdzić czy pudełko dobre?**
  - przejrzeć wszystkie procesory -> algorytm O(n)
  - losowo wybrać k procesorów do sprawdzenia -> algorytm O(1)

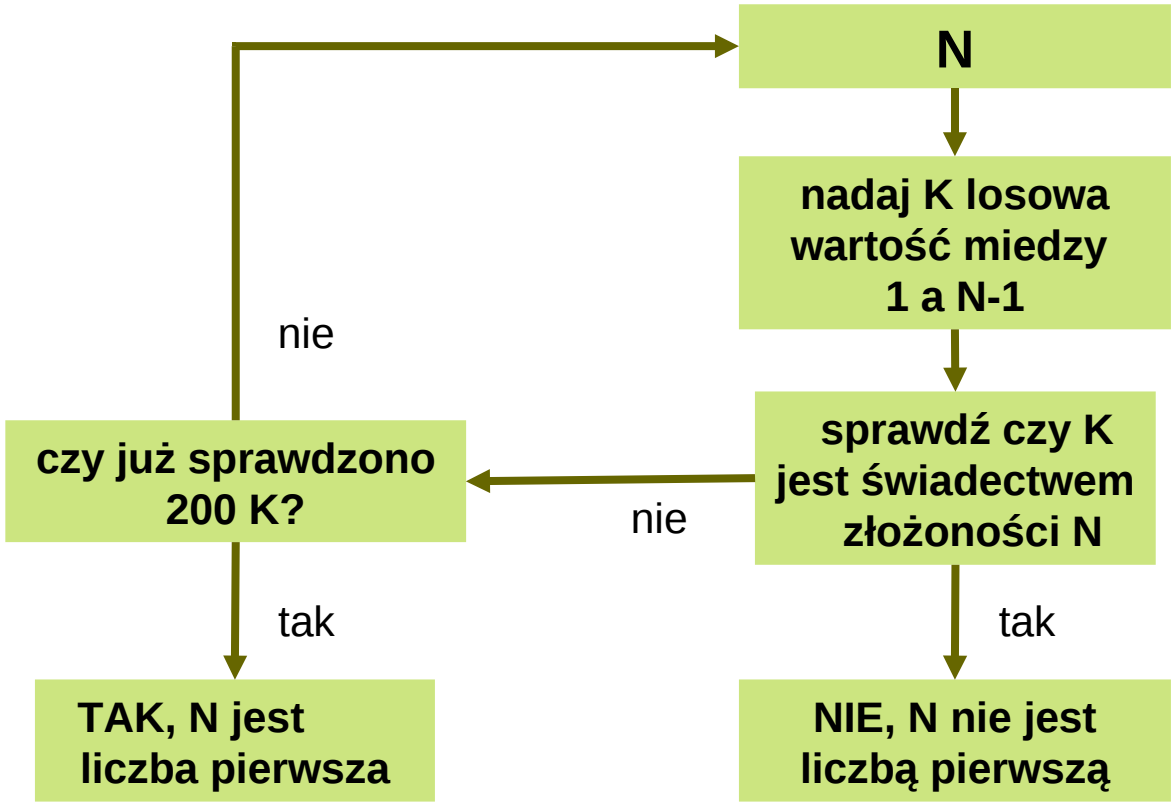
  błąd polegałby na uznaniu ze pudełko dobre (przetestowane) jeżeli nie było takie.
- Przykład: Losujemy **k=131** procesorów.
  Jeżeli procesor jest dobry odpowiadamy „nie wiem”.
  Prawdopodobieństwo ze „nie wiem” dla każdego z **k**-procesorów $(0.9)^k = (0.9)^{131} = 10^{-6}$.
  10^{-6} to jest prawdopodobieństwo że pudełko uznamy za dobre choć nie było testowane przez producenta.
  Za cenę błędu = 10^{-6}, zamieniliśmy algorytm z ***O*(n)** na ***O*(1)**. Możemy regulować wielkość błędu/czas działania algorytmu zmieniając **k**.

Probabilistyczne algorytmy sprawdzania

„ Czy liczba N jest liczbą pierwszą ?”

- W połowie lat 70-tych odkryto **dwa bardzo eleganckie probabilistyczne algorytmy** sprawdzające, czy liczba jest pierwsza. Były one jednymi z pierwszych rozwiązań probabilistycznych dla trudnych problemów algorytmicznych. Wywołały fale badań które doprowadziły do probabilistycznych rozwiązań wielu innych problemów.
- Oba algorytmy wykonują się w czasie wielomianowym (niskiego stopnia), zależnym od liczby cyfr w danej liczbie **N** (czyli ***O* (log N)**).
- Oba algorytmy są oparte na losowym szukaniu pewnych rodzajów potwierdzeń lub **świadectw złożoności** liczby N.
- Po znalezieniu takiego świadectwa algorytm może się bezpiecznie zatrzymać z odpowiedzią **„nie, N nie jest liczbą pierwszą”,** ponieważ istnieje bezdyskusyjny dowód że **N** jest liczbą złożoną.
- Poszukiwanie musi być przeprowadzone w taki sposób aby w pewnym rozsądnym czasie algorytm mógł przerwać szukanie odpowiadając, że **N** jest liczbą pierwszą z bardzo małą szansą omyłki.
- Trzeba zatem znaleźć dająca się **szybko sprawdzać** definicje świadectwa złożoności.

„Czy liczba N jest liczbą pierwszą ?



jeśli to jest odpowiedź to jest to prawda z błędem $\mathbf{1/2^{200}}$

jeśli to jest odpowiedź to jest to prawda

Świadectwa złożoności (zarys)

- Każda liczba parzysta poza 2 to jest liczba złożona
- Jeżeli suma cyfr liczby jest podzielna przez 3 to liczba jest złożona (iteracyjny prosty algorytm liniowo zależny od liczby cyfr)
- **Test pierwszości Fermata:**
  - jeśli **n** jest liczbą pierwszą oraz **k** jest dowolna liczba całkowita **(1, n-1)**,
    to $\mathbf{k^{n-1} \equiv 1 \pmod n}$.
  - natomiast jeśli **n** jest liczbą złożoną (z wyjątkiem kilku złych liczb złożonych – liczb Carmichael'a) oraz jeśli **k** wybierzemy losowo z przedziału **(1, n-1)** to prawdopodobieństwo tego że $\mathbf{k^{n-1} \not\equiv 1 \pmod n}$ jest mniejsze niż **½**.
  - Zatem liczby złożone (poza liczbami Carmichael'a) spełniają warunek testu dla danego **k** z prawdopodobieństwem nie mniejszym niż **½**.
- **Test pierwszości Solovay-Strassena:**
  - jeśli **k** i **n** nie maja wspólnych dzielników (co by było świadectwem złożoności) policz:
    $\mathbf{X = k^{(n-1)/2} \pmod n}$, **Y = Js(n,k) (symbol Jacobiego),**
    jeśli $\mathbf{X \neq Y}$ to **k** jest świadectwem złożoności liczby **n**.
  - dla tego testu nie ma 'złych' liczb złożonych.

Podsumowanie

- Przestrzeń probabilistyczna składa się z punktów z których każdy reprezentuje wynik jakiegoś doświadczenia. Każdy punkt **x** związany jest z nieujemna liczbą zwaną prawdopodobieństwem punktu **x**. Suma prawdopodobieństw wszystkich punktów składających się na przestrzeń probabilistyczna wynosi **1**.
- Zdarzenie jest podzbiorem punktów z przestrzeni probabilistycznej. Prawdopodobieństwo zdarzenia jest sumą prawdopodobieństw należących do niego punktów. Prawdopodobieństwo każdego zdarzenia mieści się w przedziale od **0** do **1**.

Podsumowanie

- Reguła sum określa, że prawdopodobieństwo tego, że zajdzie jedno z dwóch zdarzeń **E** lub **F** jest większe lub równe większemu z prawdopodobieństw obu zdarzeń, ale nie większa niż suma tych prawdopodobieństw.
- Reguła iloczynów określa, ze prawdopodobieństwo tego, że wynikiem pewnego doświadczenia będą dwa zdarzenia **E** i **F**, jest nie większe niż mniejsze z prawdopodobieństw obu zdarzeń.
- Wykonując algorytm **Monte Carlo** stałą liczbę razy, możemy rozwiązać problem, dochodząc do wniosku, że jeśli żadne z tych powtórzeń nie doprowadziło nas do odpowiedzi „prawda”, to odpowiedzią jest „fałsz”.