99 little bugs in the code.
99 little bugs.
Take one down,
patch it around,
117 little bugs in the code.

no. 02

#devproblems

Czy komputery mogą się mylić?

wer. 19 z drobnymi modyfikacjami!

Wojciech Myszka

2025-01-01 11:14:14 +0000

HR EXCELLENCE IN RESEARCH

Politechnika Wrocławska

Część I

Poprawność algorytmów

Błędy

- Każdy program ma błędy.
- Tak długo, jak używany jest jedynie do zabawy — nie odgrywa to większej roli.
- Programy komercyjne…

Licencja

Gwarancja. Produkt jest zaprojektowany i oferowany jako produkt ogólnego zastosowania, a nie dla określonego celu jakiegokolwiek użytkownika. Licencjobiorca uznaje, ze Produkty mogą być wadliwe. W związku z powyższym zdecydowanie zaleca się Licencjobiorcy systematyczną archiwizację plików.[…]

*Produkt będzie działał **zasadniczo** zgodnie z załączonymi do Produktu materiałami drukowanymi, oraz (b) usługi pomocy technicznej świadczone przez firmę XXX będą zasadniczo zgodne z opisem zamieszczonym w odpowiednich materiałach drukowanych dostarczonych Licencjobiorcy przez firmę XXX, a pracownicy pomocy technicznej firmy XXX podejmą uzasadnione działania i wysiłki w celu rozwiązania ewentualnych problemów.[…]*

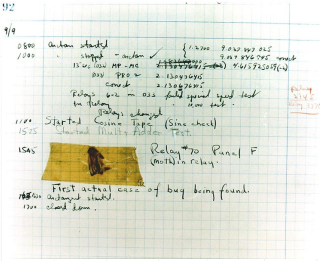
*Ograniczenie Odpowiedzialności . W maksymalnym zakresie dozwolonym przez prawo właściwe i z wyjątkiem postanowień gwarancji firmy XXX, firma XXX oraz jej dostawcy **nie będą ponosić odpowiedzialności za żadne szkody**[…]*

W każdym przypadku, całkowita odpowiedzialność firmy XXX na podstawie niniejszej Umowy jest ograniczona do kwoty rzeczywiście zapłaconej przez Licencjobiorcę za Produkt. […]

Bug

Błędy, w żargonie informatyków zwykło się nazywać „pluskwami” (czy może, raczej robalami?). Po angielsku — bug.

Anegdota mówi, że w 1947 roku Grace Murray Hopper (autorka Cobola) korzystała z komputera (działającego na przekaźnikach — Harvard Mark II). Pojawiły się problemy z jego pracą. Po badaniach, okazało się, że jakiś „robaczek latający” wpadł między styki przekaźnika. Sporządzono raport, który zachował się do dziś…



Section 3

Czy są programy wolne od błędów?

Czy są programy wolne od błędów?

- Dobre pytanie.

Czy są programy wolne od błędów?

- Dobre pytanie.
- Właściwie nie ma…

Czy są programy wolne od błędów?

- Dobre pytanie.
- Właściwie nie ma…
- …ale

Czy są programy wolne od błędów?

- Dobre pytanie.
- Właściwie nie ma...
- ...ale

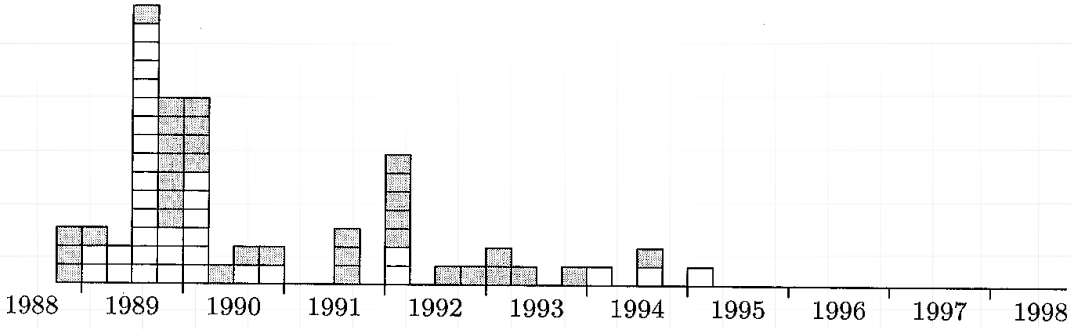
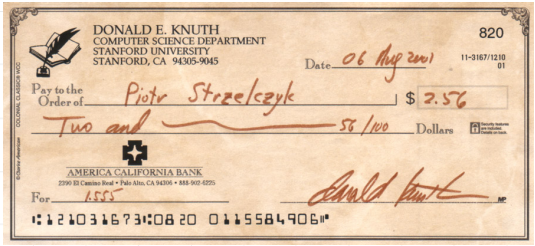


FIGURE 2. When the changes were made.

Donald Knuth i TEX

Więcej na temat tego w jaki sposób efekt ten osiągnął Donald Knuth poczytać można na stronach Wikipedii: http://en.wikipedia.org/wiki/Knuth_reward_check,
a przyklady czeków wystawionych przez Knutha polskim użytkownikom TEXa na stronach GUST: http://www.gust.org.pl/dek-checks/index_html


DONALD E. KNUTH
COMPUTER SCIENCE DEPARTMENT
STANFORD UNIVERSITY
STANFORD, CA 94305-9045

820

11-3167/1210
01

Date 06 Aug 2001

Pay to the Order of Piotr Strzelczyk $ 2.56

Two and 56/100 Dollars

AMERICA CALIFORNIA BANK
2390 El Camino Real • Palo Alto, CA 94306 • 888-902-8225

For 4555

⑆121031673⑆08 20 0115584906⑈

Uruchamianie programów i badanie ich poprawności

- Szacuje się, że ponad 70% nakładów na opracowanie złożonych systemów oprogramowania pochłania usuwanie błędów.
- Komputery — zasadniczo — są nieomylne. Psują się, ale przypadki, że generują błędne obliczenia są tak rzadkie, że spokojnie mówimy iż są nieomylne…
- Mylą się ludzie:
  - na etapie opracowania algorytmu,
  - na etapie jego programowania,
  - podczas przygotowania danych,
  - podczas obsługi programu,
  - podczas interpretacji wyników.

O nieomylności komputerów I

- Czerwiec 1994: Testerzy firmy Intel zauważyli, że procesor „źle liczy” (błąd w instrukcji dzielenia). Kierownictwo uznało, że problem nie dotknie zbyt wielu użytkowników.
- 19 października: Jeden z użytkowników zauważa błąd i nabiera pewności, że to procesor (Pentium).

```
4195835.0/3145727.0 = 1.333 820 449 136 241 002 (Correct value)
4195835.0/3145727.0 = 1.333 739 068 902 037 589 (Flawed Pentium)
```

- 24 października: zgłasza błąd do Intela, osoba odpowiedzialna za kontakt z klientem potwierdza błąd (odtwarza go), i mówi, że nikt jeszcze go nie zgłosił.
- 30 października „odkrywca” sprawę upublicznia (kilku swoim znajomym).
- 3 listopada informacja o błędzie pojawia się na grupie dyskusyjnej.
- 7 listopada Intel przyznaje się publicznie do błędu i ogłasza, że kolejne wydania procesora były już od niego wolne.
- 23 listopada pierwszy z producentów oprogramowania matematycznego (Mathworks) wypuszcza wersję Matlaba z modyfikacjami niwelującymi błąd.

O nieomylności komputerów II

- 24 listopada okazuje się, że Intel ciągle sprzedaje błędne procesory.
- 30 listopada Intel przedstawia dokładny raport opisujący problem (i potencjalne jego konsekwencje) [1]. Za późno!
- 12 grudnia IBM publikuje swój własny raport oceniający znacznie poważniej konsekwencje błędu i ogłasza, że wstrzymuje sprzedaż komputerów PC z procesorem Intela.
- 16 grudnia — akcje Intela spadają o 3,25$ (ok. 5%)
- 20 grudnia: Intel przeprasza i obiecuje bezpłatnie wymienić wadliwe procesory na żądanie.

(Na podstawie [2])

Przyczyny problemów Intela I

1. Niechęć Intela do poinformowania o problemie swoich najważniejszych klientów.
2. Niepoinformowanie swojej pomocy technicznej, co uniemożliwiło traktowanie zgłaszających błąd w sposób specjalny.
3. Pierwsze informacje Intela na temat problemu były odbierane przez wszystkich (poza firmą) jako niezadowalające i zdawkowe.
4. Nowy procesor Intela (Pentium) był przedmiotem bardzo intensywnej kampanii reklamowej.
5. Błąd dotyczył bardzo podstawowej operacji, którą łatwo było zademonstrować niefachowcom.
6. Błąd został wykryty na bardzo późnym etapie testowania procesora (po tym jak bardzo wiele procesorów zostało już sprzedanych).
7. Internet i sposób jego funkcjonowania.
8. Poważny konkurent firmy (IBM) zdecydował się na opublikowanie własnych analiz przedstawiających w innym świetle problem.

Znaczenie błędu

Failure category and system component	Hard or Soft	FIT rate (per 10^9 device hours)	MTBF (1 in x years)	Rate of significant failure seen by user
16 4-Mbit DRAM parts in a 60Mhz Pentium TM processor system without ECC	Soft	16	7 years	Depends upon where defect occurs and how propagated
Particle defects in PentiumTM processor	Hard	400-500	200-250 years	Depends upon where defect occurs and how propagated
16 4-Mbit DRAM parts in a 60Mhz Pentium TM processor system with ECC	Soft	160	700 years	Depends upon where defect occurs and how propagated
PC user on spreadsheet running 1,000 independent divides a day on the PentiumTM processor a	Hard	3.3	27,000 years	Less frequent than 1 in 27,000 years. Depends upon the way inaccurate result gets used

Na podstawie [1]

Znaczenie błędu

Class	Applications	MTBF	Impact of failure in div/rem/tran
Word processing	Microsoft Word, Wordperfect, etc.	Never	None
Spreadsheets (basic user)	123, Excel, QuattroPro (basic user runs fewer than 1000 div/day)	27,000 years	Unnoticeable
Publishing, Graphics	Print Shop, Adobe Acrobat viewers	270 years	Impact only on Viewing
Personal Money Management	Quicken, Money, Managing Your Money, Simply Money, TurboTax (fewer than 14,000 divides per day)	2,000 years	Unnoticeable
Games	X-Wing, Falcon (flight simulator), Strategy Games	270 years	Impact is benign, (since game)

Na podstawie [1]

Znaczenie błędu

Usage	Examples	Division intensive	Impact
Standard spreadsheet analysis	Corporate finance, budget or marketing analysis,	No	None
Basic financial calculations	Present value, yield to maturity	Some	Significant only in the extreme circumstance of > 10 million divisions per day
Complex mathematical models	Black-Scholes model, Binomial model	Some	Could be significant on continuous use
Path based models and simulations	Monte Carlo risk analysis, non recombining paths	Yes	Significant unless there is a low P2 factor.

Na podstawie [1]

Meltdown & Spectre

Meltdown

Meltdown breaks the most fundamental isolation between user applications and the operating system. This attack allows a program to access the memory, and thus also the secrets, of other programs and the operating system.

If your computer has a vulnerable processor and runs an unpatched operating system, it is not safe to work with sensitive information without the chance of leaking the information. This applies both to personal computers as well as cloud infrastructure. Luckily, there are software patches against Meltdown.

Meltdown Paper

Spectre

Spectre breaks the isolation between different applications. It allows an attacker to trick error-free programs, which follow best practices, into leaking their secrets. In fact, the safety checks of said best practices actually increase the attack surface and may make applications more susceptible to Spectre

Spectre is harder to exploit than Meltdown, but it is also harder to mitigate. However, it is possible to prevent specific known exploits based on Spectre through software patches.

Spectre Paper

Section 5

Formalne badanie poprawności algorytmów

Jeszcze jedna definicja algorytmu

Zadanie algorytmiczne można scharakteryzować zwięźle jako złożone z:

1. zbioru I dopuszczalnych danych wejściowych
2. zależności R między danymi a żądanymi wynikami O

$$R : I \rightarrow O$$

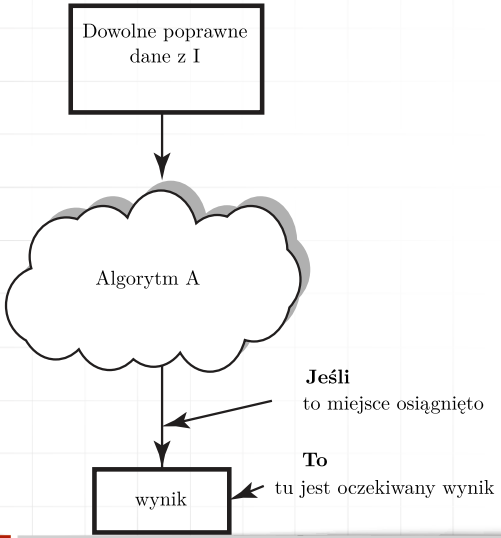
Poprawność częściowa i całkowita

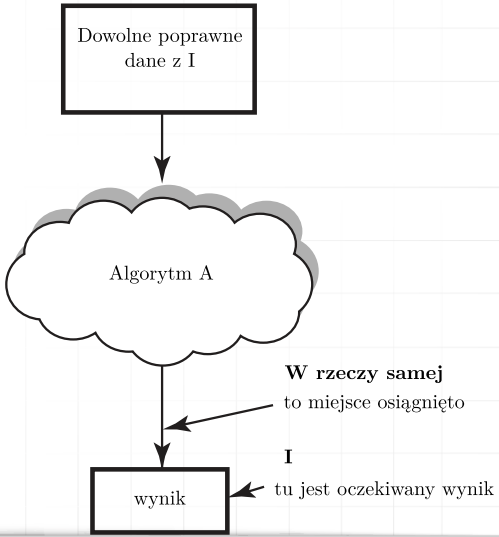
Algorytm *A* jest **częściowo poprawny** względem *I* i *R*, gdy dla każdego zestawu danych *X* z *I*, jeżeli *A* uruchomiony dla *X* zatrzyma się, to relacja *R* między *X* a otrzymanym zestawem wyników jest spełniona.

Częściowo poprawny algorytm sortowania mógłby zatrzymywać się nie dla wszystkich list dopuszczalnych, ale zatrzymując się, zawsze da w wyniku listę uporządkowaną poprawnie.

Algorytm **całkowicie poprawny** — poprawnie rozwiązuje zadanie dla każdego zestawu danych *X* z *I*: zawsze się zatrzymuje dając poprawne wyniki.

Poprawność częściowa i całkowita





Czy trzeba udowadniać poprawność algorytmów?

- Systemy bankowe…
- Systemy sterowania rakietami (balistycznymi)
- Pojazdy „kosmiczne”
- Systemy nadzoru chorych
- System ABS
- …

A praktyka jest taka (wedle powiedzonek): oprogramowanie udostępnia się użytkownikom nie wtedy, kiedy jego poprawność staje się pewna, ale wtedy, gdy szybkość odkrywania nowych błędów spada do poziomu, który może być zaakceptowany…

Jak to się robi?

- Ba!

Jak to się robi?

- Ba!
- Dla każdego poprawnego algorytmu można ściśle wykazać, że jest on poprawny.

Jak to się robi?

- Ba!
- Dla każdego poprawnego algorytmu można ściśle wykazać, że jest on poprawny.
- Inna sprawa, czy to łatwo zrobić…

Jak to się robi? cd

Obowiązują ogólne zasady:

- Z tego, że program działa poprawnie dla każdego zestawu danych, który „wypróbowaliśmy” nie wynika wcale, że działa dobrze dla innych zestawów.

Jak to się robi? cd

Obowiązują ogólne zasady:

- Z tego, że program działa poprawnie dla każdego zestawu danych, który „wypróbowaliśmy” nie wynika wcale, że działa dobrze dla innych zestawów.
- Jeżeli uda się nam znaleźć **jeden jedyny** dopuszczalny zestaw danych, dla którego program działa źle — to jest zły.

Jak to się robi? cd

Obowiązują ogólne zasady:

- Z tego, że program działa poprawnie dla każdego zestawu danych, który „wypróbowaliśmy” nie wynika wcale, że działa dobrze dla innych zestawów.
- Jeżeli uda się nam znaleźć **jeden jedyny** dopuszczalny zestaw danych, dla którego program działa źle — to jest zły.

Idealna sytuacja dla osób, które potrafią być tylko destrukcyjne!

Jak to się robi? cdcd

- Zazwyczaj zależy nam, żeby algorytm kiedyś się zatrzymał.

Jak to się robi? cdcd

- Zazwyczaj zależy nam, żeby algorytm kiedyś się zatrzymał.
- Aby wykazać, że algorytm kiedyś się zatrzyma można wybrać jakąś wielkość zależną od zmiennych i struktur danych algorytmu i wykazać, że wielkość ta jest **zbieżna**.

Jak to się robi? cdcd

- Zazwyczaj zależy nam, żeby algorytm kiedyś się zatrzymał.
- Aby wykazać, że algorytm kiedyś się zatrzyma można wybrać jakąś wielkość zależną od zmiennych i struktur danych algorytmu i wykazać, że wielkość ta jest **zbieżna**.
- Jest mi bardzo przykro, że przypomina to Analizę Matematyczną…

Prosty przykład

Odwracanie napisu

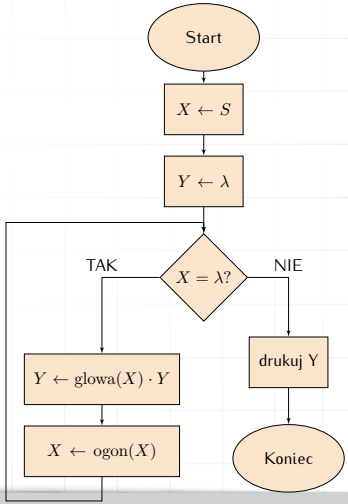
- Mamy pewien napis *S* zbudowany z ciągu symboli (na przykład zdanie języka naturalnego).
- Zadanie polega na stworzeniu procedury **odwrócone**(*S*) zwracającej symbole w kolejności odwrotnej.
- **odwrócone**("Ala ma kota") $\rightarrow$ "atok am alA"

Odwracanie napisu

Schemat blokowy

Odwracanie napisu

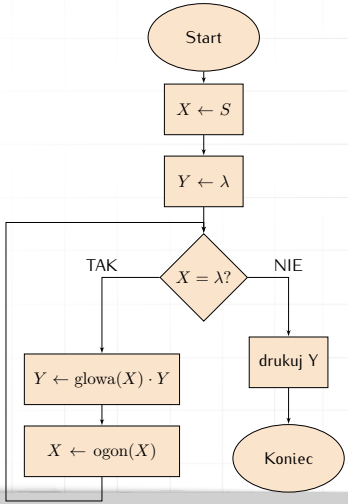
Schemat blokowy



Odwracanie napisu

Schemat blokowy

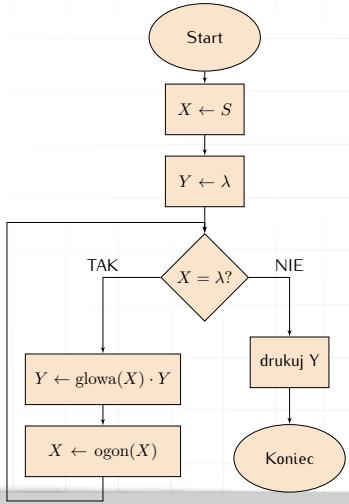
- λ to pusty napis



Odwracanie napisu

Schemat blokowy

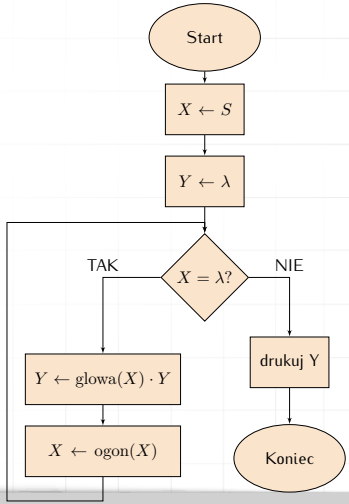
- λ to pusty napis
- **głowa**(*X*) to funkcja zwracająca pierwszy symbol z napisu *X*; **głowa**("Ala ma kota")="A"



Odwracanie napisu

Schemat blokowy

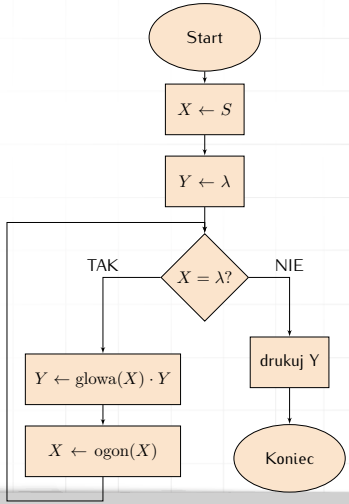
- λ to pusty napis
- **głowa**(*X*) to funkcja zwracająca pierwszy symbol z napisu *X*; **głowa**("Ala ma kota")="A"
- „·” to operator konkatenacji (łączenia napisów); "Ala "·"ma kota"="Ala ma kota"



Odwracanie napisu

Schemat blokowy

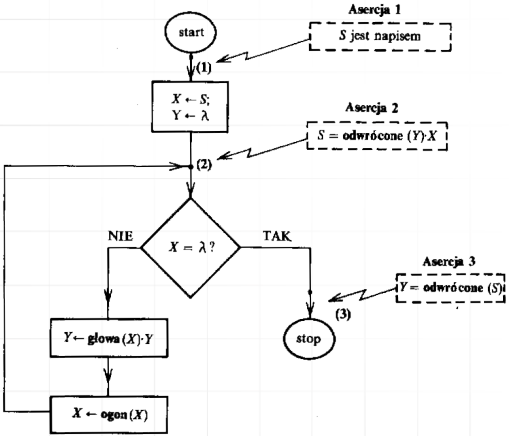
- λ to pusty napis
- **głowa**(X) to funkcja zwracająca pierwszy symbol z napisu X; **głowa**("Ala ma kota")="A"
- „$\cdot$” to operator konkatenacji (łączenia napisów); "Ala "$\cdot$"ma kota"="Ala ma kota"
- **ogon**(X) to funkcja zwracająca napis X „bez głowy”; **ogon**("Ala ma kota")="la ma kota"



Działanie algorytmu

Algorytm działa w kółko, kolejno „odrywając” symbole z końca *S* i dołączając je na początku nowo tworzonego napisu *Y*. Początkowo napis rozpoczyna jako pusty. Procedura kończy się, kiedy nic już nic nie zostanie do oderwania z *S*. Odrywania dokonuje się używając zmiennej *X*, której początkowo nadaje się wartość *S*, tak aby nie niszczyć oryginalnej wartości *S*.

Badanie poprawności



Idea postępowania:

- Tworzymy „punkty kontrolne” (asercje).
- Pierwsza z nich „kontroluje” dane wejściowe (czy zgodne z założeniami).
- Ostatnia — „sprawdza” wynik.
- Najistotniejsza jest jednak druga. Nadzoruje ona sytuację przed podjęciem decyzji czy potrzebne jest jest jeszcze jedno wykonanie pętli czy należy już kończyć. Stwierdza ona, że w punkcie kontrolnym (2) połączone wartości *X* i *Y* tworzą początkowy napis (przy czym *Y* jest odwrócony!)

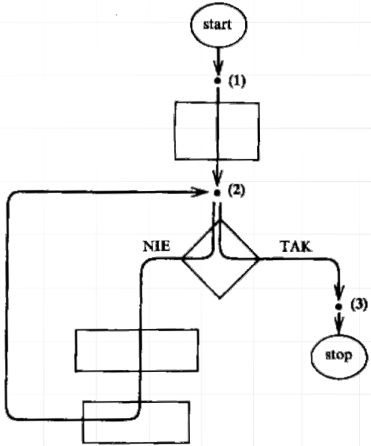
Pokazać powinniśmy, że wszystkie asercje są niezmiennikami, to znaczy, że podczas każdego wykonania algorytmu są prawdziwe (**dla danych dopuszczalnych**!)

Badanie poprawności, cd

- Sztuczka polega na tym, że rozpatruje się wszystkie możliwe drogi podążania procesora z jednego punktu kontrolnego do następnego.

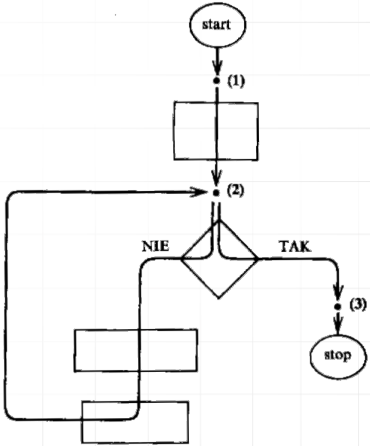
Badanie poprawności, cd

- Sztuczka polega na tym, że rozpatruje się wszystkie możliwe drogi podążania procesora z jednego punktu kontrolnego do następnego.



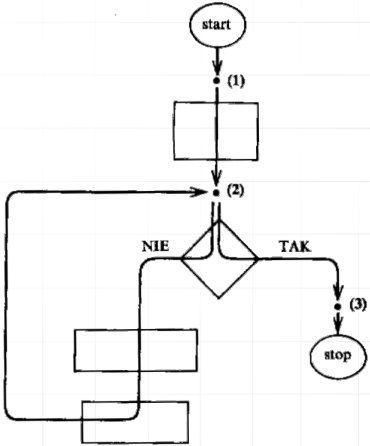
Badanie poprawności, cd

- Sztuczka polega na tym, że rozpatruje się wszystkie możliwe drogi podążania procesora z jednego punktu kontrolnego do następnego.
- W przypadku tego algorytmu możliwe są trzy drogi: z (1) do (2), z (2) do (3) i z (2) do (2)



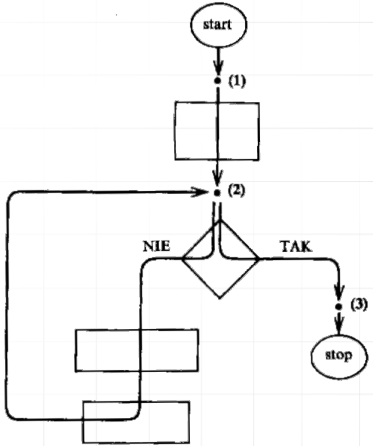
Badanie poprawności, cd

- Sztuczka polega na tym, że rozpatruje się wszystkie możliwe drogi podążania procesora z jednego punktu kontrolnego do następnego.
- W przypadku tego algorytmu możliwe są trzy drogi: z (1) do (2), z (2) do (3) i z (2) do (2)
- Pierwsza z nich przechodzona jest tylko jeden raz — na samym początku



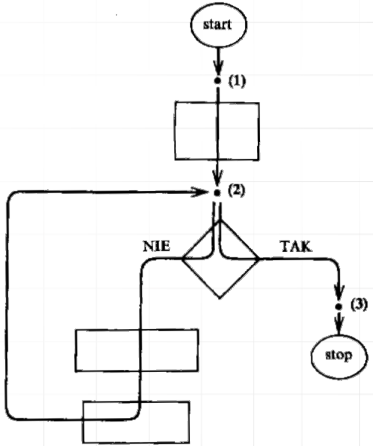
Badanie poprawności, cd

- Sztuczka polega na tym, że rozpatruje się wszystkie możliwe drogi podążania procesora z jednego punktu kontrolnego do następnego.
- W przypadku tego algorytmu możliwe są trzy drogi: z (1) do (2), z (2) do (3) i z (2) do (2)
- Pierwsza z nich przechodzona jest tylko jeden raz — na samym początku
- Druga z nich **co najwyżej** jeden raz gdy algorytm się zatrzymuje.



Badanie poprawności, cd

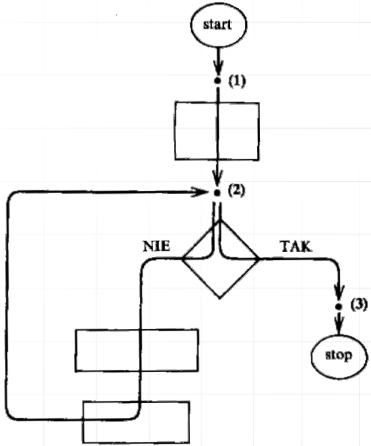
- Sztuczka polega na tym, że rozpatruje się wszystkie możliwe drogi podążania procesora z jednego punktu kontrolnego do następnego.
- W przypadku tego algorytmu możliwe są trzy drogi: z (1) do (2), z (2) do (3) i z (2) do (2)
- Pierwsza z nich przechodzona jest tylko jeden raz — na samym początku
- Druga z nich **co najwyżej** jeden raz gdy algorytm się zatrzymuje.
- Trzecia przechodzona będzie wiele razy (ile?)



Badanie poprawności, cd

- Sztuczka polega na tym, że rozpatruje się wszystkie możliwe drogi podążania procesora z jednego punktu kontrolnego do następnego.
- W przypadku tego algorytmu możliwe są trzy drogi: z (1) do (2), z (2) do (3) i z (2) do (2)
- Pierwsza z nich przechodzona jest tylko jeden raz — na samym początku
- Druga z nich **co najwyżej** jeden raz gdy algorytm się zatrzymuje.
- Trzecia przechodzona będzie wiele razy (ile?)

Zauważmy, że w żadnym odcinku nie występują już pętle. Instrukcje są dosyć proste…
Idea postępowania polega na pokazaniu, że jeśli asercja na początku odcinka jest prawdziwa i rzeczywiście przejdziemy ten odcinek to asercja na końcu odcinka również będzie prawdziwa kiedy do niej dotrzemy.



Asercje...

- Przebieg algorytmu składa się z odcinków oddzielonych punktami kontrolnymi.

Asercje…

- Przebieg algorytmu składa się z odcinków oddzielonych punktami kontrolnymi.
- Jeżeli dla każdego odcinka z prawdziwości asercji początkowej wynika prawdziwość asercji końcowej…

Asercje…

- Przebieg algorytmu składa się z odcinków oddzielonych punktami kontrolnymi.
- Jeżeli dla każdego odcinka z prawdziwości asercji początkowej wynika prawdziwość asercji końcowej…
- I jeśli pierwsza asercja w całym ciągu odpowiada poprawności danych wejściowych (z góry jest przyjęta za prawdziwą)…

Asercje…

- Przebieg algorytmu składa się z odcinków oddzielonych punktami kontrolnymi.
- Jeżeli dla każdego odcinka z prawdziwości asercji początkowej wynika prawdziwość asercji końcowej…
- I jeśli pierwsza asercja w całym ciągu odpowiada poprawności danych wejściowych (z góry jest przyjęta za prawdziwą)…
- To prawdziwość asercji propaguje się wzdłuż całego ciągu…

Asercje…

- Przebieg algorytmu składa się z odcinków oddzielonych punktami kontrolnymi.
- Jeżeli dla każdego odcinka z prawdziwości asercji początkowej wynika prawdziwość asercji końcowej…
- I jeśli pierwsza asercja w całym ciągu odpowiada poprawności danych wejściowych (z góry jest przyjęta za prawdziwą)…
- To prawdziwość asercji propaguje się wzdłuż całego ciągu…
- Zatem prawdziwa będzie i asercja końcowa…

Asercje…

- Przebieg algorytmu składa się z odcinków oddzielonych punktami kontrolnymi.
- Jeżeli dla każdego odcinka z prawdziwości asercji początkowej wynika prawdziwość asercji końcowej…
- I jeśli pierwsza asercja w całym ciągu odpowiada poprawności danych wejściowych (z góry jest przyjęta za prawdziwą)…
- To prawdziwość asercji propaguje się wzdłuż całego ciągu…
- Zatem prawdziwa będzie i asercja końcowa…
- Co gwarantuje **częściową poprawność algorytmu**.

Co teraz trzeba zrobić?

Udowodnić należy trzy twierdzenia:

$(1 \to 2)$ Dla każdego napisu S po wykonaniu dwóch instrukcji $X \leftarrow S;\ Y \leftarrow \lambda$ będzie spełniona równość $S = \textbf{odwrócone}(Y) \cdot X$

Co teraz trzeba zrobić?

Udowodnić należy trzy twierdzenia:

- $(1 \to 2)$ Dla każdego napisu S po wykonaniu dwóch instrukcji $X \leftarrow S;\ Y \leftarrow \lambda$ będzie spełniona równość $S = \textbf{odwrócone}(Y) \cdot X$
- $(2 \to 3)$ Jeśli $S = \textbf{odwrócone}(Y) \cdot X$ i $X = \lambda$ to $Y = \textbf{odwrócone}(S)$

Co teraz trzeba zrobić?

Udowodnić należy trzy twierdzenia:

- $(1 \to 2)$ Dla każdego napisu S po wykonaniu dwóch instrukcji $X \leftarrow S;\ Y \leftarrow \lambda$ będzie spełniona równość $S = \textbf{odwrócone}(Y) \cdot X$
- $(2 \to 3)$ Jeśli $S = \textbf{odwrócone}(Y) \cdot X$ i $X = \lambda$ to $Y = \textbf{odwrócone}(S)$
- $(2 \to 2)$ Jeśli $S = \textbf{odwrócone}(Y) \cdot X$ i $X \neq \lambda$ to po wykonaniu instrukcji $Y \leftarrow \textbf{głowa}(X) \cdot Y;\ X \leftarrow \textbf{ogon}(X)$ zachodzi równość $S = \textbf{odwrócone}(Y) \cdot X$ dla nowych wartości X i Y.

Dowód

Zacznijmy od $(1 \to 2)$. Po wykonaniu $X \leftarrow S$ zmienna X ma wartość S, a po wykonaniu jeszcze $Y \leftarrow \lambda$ zmienna Y zawiera napis pusty.

Czyli $\textbf{odwrócone}(Y) = \textbf{odwrócone}(\lambda)$, zatem $\textbf{odwrócone}(Y) \cdot X = \lambda \cdot X = X$.

Równością, która miała zachodzić po wykonaniu tych dwu instrukcji jest więc po prostu $S = X$. Co kończy dowód.

W analogiczny sposób można pokazać $(2 \to 3)$.

Dowód cd

Pokażemy teraz, że $(2 \to 2)$, czyli jeśli spełniona jest asercja 2 i pętla będzie wykonana jeden raz to asercja zostanie spełniona i po tym wykonaniu.
Załóżmy, że $X \neq \lambda$ (czyli w szczególności, że X ma i głową i ogon — puste napisy ich nie mają) i że napis S jest dokładnie taki sam jak $\textbf{odwrócone}(Y) \cdot X$.

Wykonujemy teraz działania:

$$Y \leftarrow \textbf{głowa}(X) \cdot Y \quad \text{i} \quad X \leftarrow \textbf{ogon}(X)$$

zatem

$$\begin{gathered}\textbf{odwrócone}(Y) \cdot X = \\ \textbf{odwrócone}(\textbf{głowa}(X) \cdot Y) \cdot \textbf{ogon}(X) = \\ \textbf{odwrócone}(Y) \cdot \textbf{głowa}(X) \cdot \textbf{ogon}(X) = \\ \textbf{odwrócone}(Y) \cdot X\end{gathered}$$

Wnioski

Pokazaliśmy, że wszystkie asercje są spełnione. Zatem algorytm jest częściowo poprawny (czyli — jeżeli się zatrzyma to daje poprawne wyniki.

Jak pokazać, że zatrzyma się dla każdych poprawnych danych?

Jedynie miejsce gdzie algorytm może się „zapętlić” to ścieżka $(2 \to 2)$. Ale na tej ścieżce za każdym razem dokonywana jest operacja „odcinania głowy” od *X* co powoduje, że za każdym razem *X* jest krótszy.

Gdy *X* będzie napisem jednoliterowym — odcięcie głowy stworzy pusty ciąg znaków.

Zatem algorytm jest poprawny!

I po co to wszystko?

1. Algorytm był prosty, a dowód zagmatwany (i mocno matematyczny).
2. Jego istota polega na odpowiednim doborze punktów kontrolnych (początek i koniec algorytmu, oraz kilka takich innych miejsc między innymi tak dobranych aby likwidować pętle.
3. Ważną sprawą jest dobór „niezmienników” (asercji).
4. Na koniec należy dobrać „warunek stopu” czyli taki „parametr” o którym można powiedzieć że w trakcie wykonywania algorytmu będzie zmierzał do jakiejś wartości (**będzie zbieżny**).

Niestety, nie da się skonstruować automatu, który będzie służył do dowodzenia poprawności algorytmów…

Lista błędów oprogramowania na Wikipedii

https://en.wikipedia.org/wiki/List_of_software_bugs

Kilka przykładów I

- Na początku lat 60. jeden z amerykańskich statków kosmicznych z serii Mariner wysłany na Wenus został utracony na zawsze, co kosztowało miliony dolarów, z powodu błędu w programie komputerowym sterującym lotem.
- W 1981 r. jedna ze stacji telewizyjnych relacjonujących wybory prowincjonalne w Quebecu w Kanadzie podał nieprawdziwe informacje sugerujące, że mała partyjka (w sumie bez szans) prowadzi.
- W serii incydentów między 1985 a 1987 rokiem kilku pacjentów otrzymało ogromne dawki promieniowania z systemów radioterapii Therac-25; trzech z nich zmarło z powodu powikłań. Blokady bezpieczeństwa sprzętu z poprzednich modeli zostały zastąpione kontrolami bezpieczeństwa oprogramowania, ale wszystkie te incydenty wiązały się z błędami programistycznymi.
- Kilka lat temu pewna Duńska kobieta otrzymała, około swoich 107. urodzin, skomputeryzowany list od lokalnych władz szkolnych z instrukcjami dotyczącymi procedury rejestracji do pierwszej klasy w szkole podstawowej. Okazało się, że w bazie danych w polu „wiek” przydzielono tylko dwie cyfry.

Kilka przykładów II

- Na przełomie tysiącleci problemy z oprogramowaniem stały się głównymi wiadomościami wraz z tak zwanym Problemem Roku 2000, czyli błędem Y2K. Obawiano się, że 1 stycznia 2000 r. rozpęta się piekło, ponieważ komputery, które używały dwóch cyfr do przechowywania lat, błędnie przyjmą, że rok podany jako 00 to 1900, podczas gdy w rzeczywistości był to 2000. Firmy programistyczne na całym świecie musiały podjąć niezwykle kosztowną (i, patrząc wstecz, całkiem udaną) próbę skorygowania tych programów.
- Błąd oprogramowania Patriot MIM-104 spowodował, że zegar systemowy przesunął się o jedną trzecią sekundy w ciągu stu godzin – co spowodowało, że nie udało się zlokalizować i przechwycić nadlatującego irackiego pocisku rakietowego Al Hussein, który następnie uderzył w koszary Dharan w Arabii Saudyjskiej (25 lutego 1991 r.), zabijając 28 Amerykanów.

Kilka przykładów III

- Grupa sześciu samolotów F-22 Raptor lecących z bazy Hickam na Hawajach do nowej bazy w Japonii doświadczyła wielu awarii komputerów w momencie przekroczenia 180. południka długości geograficznej (międzynarodowej linii zmiany daty). Awarie komputerów obejmowały co najmniej nawigację (całkowicie utraconą) i komunikację. Myśliwce były w stanie powrócić na Hawaje, podążając za swoimi tankowcami, co mogłoby być problematyczne, gdyby pogoda nie była dobra. Błąd został naprawiony w ciągu 48 godzin, co pozwoliło na opóźnione przemieszczenie.
- W maju 2015 r. użytkownicy iPhone'a odkryli błąd, w którym wysłanie określonej sekwencji znaków i symboli Unicode jako tekstu do innego użytkownika iPhone'a powodowało awarię interfejsu SpringBoard odbierającego iPhone'a, a także mogło spowodować awarię całego telefonu, wywołać przywrócenie ustawień fabrycznych lub zakłócić łączność urządzenia, w znacznym stopniu, uniemożliwiając jego normalne działanie. Błąd utrzymywał się przez tygodnie, zyskał znaczną sławę i spowodował, że wiele osób wykorzystało go do robienia kawałów innym

Kilka przykładów IV

użytkownikom iOS, zanim Apple ostatecznie załatało go 30 czerwca 2015 r. za pomocą iOS 8.4.

- Pierwsza próba Izraela wylądowania bezzałogowego statku kosmicznego na Księżycu zakończyła się niepowodzeniem 11 kwietnia 2019 r. z powodu błędu oprogramowania w systemie silnika, który uniemożliwił zwolnienie podczas ostatecznego zejścia na powierzchnię Księżyca. Inżynierowie próbowali naprawić ten błąd, zdalnie restartując silnik, ale kiedy odzyskali nad nim kontrolę, *Beresheet* nie był w stanie zwolnić na czas, aby uniknąć twardego, awaryjnego lądowania, które go rozbiło.
- 19 lipca 2024 r. amerykańska firma zajmująca się cyberbezpieczeństwem CrowdStrike rozpowszechniła wadliwą aktualizację swojego oprogramowania zabezpieczającego Falcon Sensor, która spowodowała powszechne problemy z komputerami z systemem Microsoft Windows, na których działało oprogramowanie. W rezultacie **około 8,5 miliona systemów uległo awarii i nie można ich było prawidłowo ponownie uruchomić** w tym, co zostało nazwane największą awarią w historii technologii informatycznych i „historyczną pod względem skali”.

Kilka przykładów V

W połowie lipca 2024 r. Delta Air Lines, główny amerykański przewoźnik i największa linia lotnicza na świecie pod względem przychodów, aktywów i kapitalizacji rynkowej, doświadczyła zakłóceń operacyjnych po incydencie CrowdStrike z 2024 r., w tym **odwołania ponad 1200 lotów**. Kryzys rozpoczął się rano w piątek 19 lipca, kiedy główni przewoźnicy wydali nakaz zatrzymania naziemnego, ale podczas gdy inni przewoźnicy szybko się otrząsnęli, kryzys trwał nadal w przypadku Delta, a linia lotnicza w końcu wznowiła normalne operacje lotnicze 25 lipca. Delta potwierdziła, że kryzys doprowadził do **odwołania ponad 7000 lotów** w ciągu pięciu dni **zakłócenia, które dotknęły ponad 1,3 miliona pasażerów**.

Geneza I

1. Jest taka stara, angielska piosenka *99 Bottles of Beer on the Wall* śpiewana (w pubach) dla „zabicia czasu”.
   Jej treść idzie jakoś tak:
   99 bottles of beer on the wall, 99 bottles of beer. Take one down and pass it around - 98 bottles of beer on the wall.
   98 bottles of beer on the wall, 98 bottles of beer. Take one down and pass it around - 97 bottles of beer on the wall.
   . . .
   2 bottles of beer on the wall, 2 bottles of beer. Take one down and pass it around - 1 bottle of beer on the wall.
   1 bottle of beer on the wall, 1 bottle of beer. Take it down and pass it around - no more bottles of beer on the wall.
2. Dawno temu, (gdy Internet bł jeszcze „słaby”) na jednej z grup dyskusyjnych poświęconych programowaniu ktoś zamieścił cały jej teks, co spowodowało złość uczestników:

Geneza II

- bo off-topic,
- bo zużyło strasznie dużo przepustowości.

3. Ktoś zamieścił komentarz typu: *Gdybyś umiał programować, to tekst tej piosenki można zapisać krócej tak:* i dalej pojawił się fragment w jakimś języku programowania.
4. Kolejni uczestnicy dyskusji dodawali przykłady w innych językach programowania…
5. Zebrano je na wielu stronach. Jedną z nich może być `https://rosettacode.org/wiki/99_bottles_of_beer`

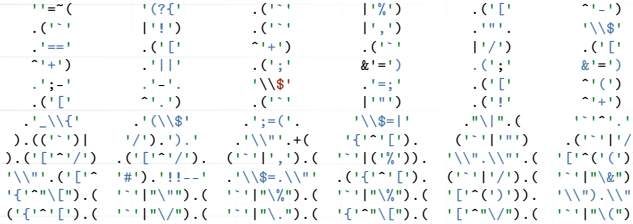
Przykład w LaTeXu I

```
\documentclass{article}

\newcounter{beer}

\newcommand{\verses}[1]{
  \setcounter{beer}{#1}
  \par\noindent
  \arabic{beer} bottles of beer on the wall,\\
  \arabic{beer} bottles of beer!\\
  Take one down, pass it around---\\
  \addtocounter{beer}{-1}
  \arabic{beer} bottles of beer on the wall!\\
  \ifnum#1>0
    \verses{\value{beer}}
  \fi
```

Przykład w LaTeXu II

```
}

\begin{document}
\verses{99}
\end{document}
```

Perl



Geneza ilustracji…

…jest już chyba jasna…

Bibliografia

Intel.
White paper: Statistical analysis of floating point flaw.
https://web.archive.org/web/20050123064847/http://www.intel.com/support/processors/pentium/fdiv/wp/, 1994.

Thomas R. Nicely.
Pentium FDIV flaw FAQ.
https://web.archive.org/web/20191126144803/http://www.trnicely.net:80/pentbug/pentbug.html, 2008.