Rys. 6. Zaokrąglone serce albo dwupalczasta stopa

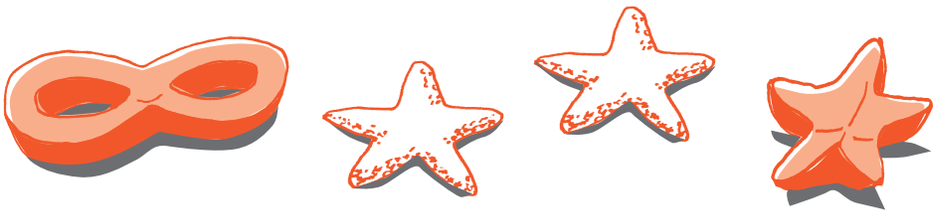
Rys. 7. Wykres funkcji $y = |x|^{3/2}$ (można sobie wyobrazić, że dopełniony łukiem od góry)

W odróżnieniu od „zewnętrznych kopczyków" te „wewnętrzne" zawsze mają jakąś niegładkość. Dobrym ćwiczeniem jest pokazanie, że funkcja $d_{\partial F}$ nigdy nie jest różniczkowalna w punkcie $\mathbf{x} \in F$ najdalszym od brzegu.

Co dalej? Dalsza zabawa piaskiem prowadzi do wniosku, że sytuacja jest jeszcze ciekawsza. Są kształty, jak na rysunku 6, gdzie foremka ma kant, ale kopczyk jest gładki mimo to. Kluczowe jest tutaj, że kant jest skierowany do wewnątrz; powoduje on, że funkcja odległości nie jest różniczkowalna *na zewnątrz* foremki, ale tego już nie widzimy. Podobny efekt uzyskamy, jeśli nasypiemy sobie piasku na stopę.

Są też foremki jak ta na rysunku 7, która wydaje się gładka, ale kopczyk ma krawędź aż do brzegu. Tu diabeł tkwi w szczegółach – okazuje się, że brzeg foremki nie jest dostatecznie gładki. Owszem, ma wszędzie określoną prostą styczną, ale jego krzywizna jest nieograniczona. W języku analizy: funkcja $|x|^{3/2}$ jest rożniczkowalna, ale tylko raz, a dwukrotnie już nie.

Wreszcie, ciekawy jest przypadek, gdy usypujemy kopczyk *na zewnątrz* foremki – w następnym numerze ukaże się artykuł Sławomira Dinewa poświęcony temu zagadnieniu. Trudno sobie wyobrazić, jak taki kopczyk wykonać w piaskownicy – na plaży być może byłaby szansa – ale sam problem okazuje się całkiem bogaty od strony czysto matematycznej. Foremki, dla których „zewnętrzne kopczyki" są gładkie, całkowicie charakteryzuje twierdzenie Motzkina… ale o tym za miesiąc!



O czym lepiej zapomnieć?

*Wojciech PRZYBYSZEWSKI**

* Wydział Matematyki, Informatyki i Mechaniki, Uniwersytet Warszawski

Mam wrażenie, że jeszcze kilka lat temu każda reklama laptopa składała się wyłącznie z wykrzykiwanych przez lektora kolejnych angielskich skrótów (np. RAM, SSD, HDMI), czasami wraz z wartościami liczbowymi je opisującymi. Wydaje mi się, że celem tych reklam nie było przekazanie widzowi informacji o parametrach sprzedawanego sprzętu, a jedynie zrobienie wrażenia na tych, którzy nie wiedzieli, co oznaczają wymienione skróty i wartości liczbowe, tak aby zyskali przekonanie, że prezentowany laptop korzysta z najnowszych technologii w ich najlepszym wydaniu. Z biegiem czasu ilość takich reklam się zmniejszała, pewnie przez fakt, że coraz więcej konsumentów ma świadomość, czym różni się np. pamięć RAM od dysku SSD.

W tym artykule skupimy się właśnie na parametrach związanych z pamięcią. Wiemy, że rodzajów pamięci w komputerze jest kilka – mamy m.in. rejestry, cache, RAM, dysk SSD, dysk twardy. To, czym się one od siebie różnią, świetnie opisał Tomasz Idziaszek w artykule *Pamięć w komputerze* w Δ_{16}^{5}. Nie będziemy tutaj powtarzać całego artykułu, ale wspomnimy tylko najważniejszy wniosek – główne różnice pomiędzy wymienionymi rodzajami pamięci to ich cena i szybkość dostępu do danych. Im szybszy jest jakiś rodzaj pamięci, tym jest droższy i mniej mamy go w komputerze. Dla przykładu laptop, na którym piszę ten artykuł, ma 477 GB pamięci na dysku SSD i 16 GB (czyli prawie 30 razy mniej) szybszej pamięci RAM.

W jaki sposób ta zależność między ilością a szybkością różnych rodzajów pamięci wpływa na działanie procesora? Generalnie zasada jest prosta – jeśli procesor musi skorzystać z danych, które znajdują się w wolniejszej pamięci, to przenosi je do szybszej pamięci, aby mieć do nich łatwiejszy dostęp. W teorii brzmi świetnie, ale przecież szybszej pamięci mamy mniej. Co, jeśli procesor chce przenieść jakieś dane z dysku SSD do pamięci RAM, ale okazuje się, że jest ona już w całości wypełniona? Nie ma rady, trzeba wtedy coś z pamięci

Rozwiązanie zadania M 1772. Oznaczmy przez S zbiór punktów z zadania. Rozważmy jeden z punktów $A \in \mathcal{S}$, który jest końcem dokładnie a odcinków. Rozpatrzmy prostą ℓ przechodzącą przez A, która nie jest równoległa do żadnej z prostych łączących pary punktów z $\mathcal{S}$. Przesuwając równolegle prostą ℓ, możemy otrzymać proste ℓ_1 i ℓ_2 (równoległe do ℓ), które w pasie pomiędzy nimi nie zawierają ani jednego punktu z $\mathcal{S}$, za wyjątkiem punktu A.

Niech prosta ℓ_1 przecina dokładnie x odcinków wychodzących z punktu A. Wtedy ℓ_2 przecina pozostałe $a - x$ odcinki, gdyż każdy odcinek wychodzący z punktu A przecina dokładnie jedną z prostych ℓ_1 i ℓ_2. Ponadto każdy odcinek łączący dwa punkty z $\mathcal{S}$, różne od A, albo przecina obie proste ℓ_1 i ℓ_2, albo żadnej. Wobec tego liczby odcinków przeciętych prostymi ℓ_1 i ℓ_2 różnią się o $x - (a - x) = 2x - a$. Zgodnie z warunkami zadania liczba ta musi być parzysta. Oznacza to w szczególności, że a jest liczbą parzystą.

Rozwiązanie zadania F 1090.
Roztwór wody z mydłem tworzy powłokę bańki. Ma ona dwie powierzchnie rozdziału roztworu i powietrza: wewnętrzną i zewnętrzną. W związku z tym zwiększenie pola powierzchni $S = 4\pi r^2$ powłoki (rozmiaru bańki) o dS wymaga wykonania pracy $dW = 2\sigma dS$. Na powłokę bańki działa różnica ciśnień wewnątrz bańki p i ciśnienia zewnętrznego p_0. W stanie równowagi dla bańki o promieniu r praca tych sił potrzebna do powiększenia promienia o dr jest równa pracy potrzebnej do zwiększenia powierzchni powłoki o $dS = 4\pi((r+dr)^2 - r^2) \approx 8\pi r dr$:

$$(p - p_0)S dr = 4\pi r^2 (p - p_0) dr = 8\pi r \cdot 2\sigma dr,$$

a więc:

$$p = p_0 + \frac{4\sigma}{r}.$$

Zgodnie z prawem Archimedesa bańka będzie się unosiła, gdy gęstość ρ powietrza w jej wnętrzu będzie mniejsza od gęstości ρ_0 powietrza na zewnątrz (pomijamy ciężar powłoki bańki). Jest to możliwe, gdy temperatura T_b w bańce jest wyższa niż temperatura T na zewnątrz. W interesującym nas przedziale temperatur i ciśnień powietrze doskonale spełnia równanie gazu doskonałego. Gęstość gazu doskonałego o temperaturze T pod ciśnieniem p wynosi:

$$\rho = \frac{\mu p}{RT},$$

μ oznacza tu masę molową gazu (powietrza), R uniwersalną stałą gazową. Otrzymujemy warunek:

$$\rho < \rho_0 \quad \Leftrightarrow \quad T_b - T > \frac{4\sigma}{r p_0} T.$$

Dla danych z treści zadania $T_b - T > 0{,}018$ K.

Możemy jeszcze sprawdzić, czy pominięcie masy powłoki bańki nie jest zbyt grubym przybliżeniem. Grubość takiej powłoki $\delta \approx 1$ µm, a jej gęstość jest równa gęstości wody, 1 g/cm^3. Masa powłoki bańki o promieniu 2 cm wynosi więc około 5,1 mg, co stanowi około 12% masy powietrza wypartego przez bańkę.

Dane liczbowe w tym zadaniu zostały zaczerpnięte z pracy o kształcie wielkich baniek mydlanych: C. Cohen, B. D. Texier, E. Reyssat, J. H. Snoeijer, D. Quére, and C. Clanet, *PNAS* **114**, 2515 (2017).

RAM usunąć (np. zapisać część danych z powrotem na dysku SSD, zwalniając fragment RAM-u). No dobrze, ale jak wybrać, które dane odeślemy na SSD? Na pewno jeśli w RAM-ie mamy jakieś dane, z których nie będziemy korzystać w najbliższym czasie, to wydają się one dobrym kandydatem – przecież i tak nie są nam potrzebne w pamięci z szybkim dostępem. Ponadto, jeśli do jakiejś porcji danych odwołujemy się co chwila, to odesłanie ich na dysk SSD skończy się tym, że za chwilę znów będziemy musieli je ściągać do RAM-u, co spowolni działanie komputera. Widać więc, że podjęcie właściwej decyzji jest kluczowe dla zapewnienia efektywnego działania komputera.

Teoretyczny model pamięci w komputerze

Aby przeanalizować przedstawiony problem w szczegółach, posłużymy się uproszczonym modelem pamięci w komputerze. Założymy mianowicie, że mamy tylko dwa jej rodzaje – pamięć szybką i pamięć wolną. Komputer zwykle operuje na większych fragmentach pamięci, nazywanych stronami (w zależności od kontekstu mówimy też o blokach albo ramkach) – typowy rozmiar strony to coś rzędu 4 kB. Przyjmiemy więc, że w pamięci szybkiej możemy pomieścić maksymalnie k stron danych (inaczej można powiedzieć, że składa się ona z k ramek), natomiast pamięć wolna jest nieograniczona i może pomieścić nieskończenie wiele stron. W trakcie swojego działania procesor potrzebuje czasami odwołać się do danych znajdujących się na konkretnych stronach. Jeśli strona s, do której procesor chce się odwołać, znajduje się akurat w pamięci szybkiej, to nie ma żadnego problemu. Gorzej, jeśli strony s w pamięci szybkiej nie ma – wtedy mamy do czynienia z błędem braku strony (ang. *page fault*). Procesor musi sprowadzić stronę z pamięci wolnej do pamięci szybkiej, co zajmuje pewien czas. Ponadto, jeśli w pamięci szybkiej mamy już zajęte wszystkie k ramek, to procesor musi wybrać jedną stronę s', która aktualnie znajduje się w pamięci szybkiej, i usunąć ją (inaczej: zapomnieć) poprzez odesłanie jej do pamięci wolnej. Oczywiście chcielibyśmy w taki sposób wybierać stronę do zapomnienia, żeby zminimalizować liczbę błędów braku strony w czasie działania procesora. W ten sposób zapewnimy, że komputer będzie działać efektywnie.

Strategia FIFO

Jednym z możliwych algorytmów wyboru, którą stronę zapomnieć, jest strategia FIFO (ang. *first in, first out*). Zgodnie z jej założeniami, jeśli musimy zapomnieć którąś stronę, to wybieramy tę, która była najdawniej załadowana do pamięci szybkiej. Prześledźmy działanie tego algorytmu na konkretnym przykładzie. Załóżmy, że w pamięci szybkiej możemy zmieścić $k = 3$ strony, zaś procesor generuje kolejno odwołania do stron $D, E, C, A, D, E, B, D, E, C, A, B$. Oczywiście na samym początku nasza pamięć szybka jest pusta, więc pierwsze odwołanie do strony D spowoduje błąd braku strony i załadowanie D. Podobnie będzie przy dwóch kolejnych odwołaniach do stron E i C. Po tych trzech pierwszych odwołaniach mamy 3 błędy braku strony i trzymamy w pamięci szybkiej strony D, E, C. Kiedy procesor generuje odwołanie do strony A, znowu mamy błąd braku strony. Tym razem jednak musimy zdecydować, którą ze stron aktualnie trzymanych w pamięci szybkiej zapomnimy. Skoro trzymamy się strategii FIFO, to decydujemy się na usunięcie D, ponieważ ona była najwcześniej dodana. Mamy więc już 4 błędy braku strony, a w pamięci szybkiej trzymamy E, C, A. Dalszy przebieg algorytmu zaznaczamy w tabelce poniżej.

W górnym rzędzie $*$ oznacza, że wystąpił błąd braku strony. Z kolei w trzech dolnych wierszach trzymamy zawartość pamięci szybkiej po wczytaniu kolejnej strony, w kolejności, w jakiej były załadowane do pamięci.

Błąd	*	*	*	*	*	*	*			*	*	
Ciąg odwołań	D	E	C	A	D	E	B	D	E	C	A	B
Zawartość szybkiej pamięci	D	D E	D E C	E C A	C A D	A D E	D E B	D E B	D E B	E B C	B C A	B C A

Tabela 1. Strategia FIFO z trzema ramkami w pamięci szybkiej

Rozwiązanie zadania M 1773.
Rozważmy dowolny punkt P podstawy. Udowodnijmy, że jest on pokryty jednym z trójkątów z zadania. Rozważmy sferę ω leżącą wewnątrz ostrosłupa i styczną do podstawy w punkcie P. Zwiększamy jej promień, zachowując punkt styczności, do momentu aż po raz pierwszy ω będzie styczna do pewnej ściany ostrosłupa – bez straty ogólności przyjmijmy, że jest to ściana SA_1A_2, a przez Q oznaczmy punkt styczności tej ściany z ω.
Wtedy z równości odcinków stycznych do sfery $PA_1 = QA_1$ oraz $PA_2 = QA_2$, zatem trójkąty PA_1A_2 i QA_1A_2 są przystające. Oznacza to, że obracając ścianę SA_1A_2 wokół A_1A_2, tak aby pokryła się z trójkątem $X_1A_1A_2$ w płaszczyźnie podstawy, otrzymujemy, że punkt Q przechodzi na punkt P.
W szczególności P leży wewnątrz trójkąta $X_1A_1A_2$.

Okazało się, że dla tego ciągu strategia FIFO spowodowała aż 9 błędów braku strony. Jednak prawdopodobnie było to spowodowane tym, że mieliśmy tylko $k = 3$ ramki do dyspozycji. Oczywiście, gdyby ramek było więcej, to błędów braku strony byłoby mniej. Dla przykładu zobaczmy w tabelce, co dzieje się dla $k = 4$.

Błąd	*	*	*	*			*	*	*	*	*	*
Ciąg odwołań	D	E	C	A	D	E	B	D	E	C	A	B
Zawartość szybkiej pamięci	D	D E	D E C	D E C A	D E C A	D E C A	E C A B	C A B D	A B D E	B D E C	D E C A	E C A B

Tabela 2. Strategia FIFO z czterema ramkami w pamięci szybkiej

Czyli rzeczywiście dla większej liczby dostępnych ramek algorytm FIFO zrobił mniej błędów braku strony. Zaraz! Przecież teraz mieliśmy 10 takich błędów, a dla trzech ramek było ich tylko 9! Jak to możliwe, że gdy mamy więcej dostępnej pamięci, powstaje więcej błędów braku strony, które spowalniają procesor? Czy w takim razie, żeby przyspieszyć nieco swój komputer, powinienem zmniejszyć ilość dostępnego RAM-u? Przecież to brzmi absurdalnie.

Cechą niektórych algorytmów wymiany strony jest to, że przy odpowiednio spreparowanym ciągu odwołań zwiększenie liczby dostępnych ramek w pamięci szybkiej zwiększa liczbę błędów braku strony. Taką sytuację nazywamy anomalią Bélády'ego, od nazwiska informatyka, który w 1969 roku zademonstrował ją po raz pierwszy. Nie oznacza to jednak, że przy każdym ciągu odwołań zwiększenie liczby ramek będzie skutkowało większą liczbą błędów. Dla przykładu, jeśli dla rozważanego ciągu zwiększymy dostępną szybką pamięć do $k = 5$ ramek, to wtedy błędów braku strony będzie już tylko 5.

László Bélády (1928–2021) – węgierski informatyk pracujący zarówno na uniwersytetach, jak i w sektorze prywatnym.

Algorytm FIFO, mimo że jest bardzo prosty w implementacji (wystarczy trzymać jedną kolejkę ze stronami wczytywanymi do pamięci szybkiej), nie jest w praktyce używany we współczesnych systemach operacyjnych. Wynika to z tego, że nie bierze pod uwagę, czy do jakiejś strony były ostatnio odwołania, a tylko kiedy była wczytana, co intuicyjnie nie ma związku z częstością korzystania z niej.

Strategia LRU

Strategia, która wydaje się lepszym rozwiązaniem problemu zarządzania pamięcią, to LRU (ang. *least recently used*). Mówi ona, żeby usuwać tę stronę, która była używana najdawniej. Jest ona trudniejsza do implementacji od strategii FIFO, ale we współczesnych systemach operacyjnych używa się jej albo jakichś jej przybliżeń. Zobaczmy, jak strategia ta radzi sobie z rozważanym ciągiem odwołań, analizując jej działanie dla $k = 3$ ramek w tabelce poniżej.

Błąd	*	*	*	*	*	*	*			*	*	*
Ciąg odwołań	D	E	C	A	D	E	B	D	E	C	A	B
Zawartość szybkiej pamięci	D	D E	D E C	E C A	C A D	A D E	D E B	E B D	B D E	D E C	E C A	C A B

Tabela 3. Strategia LRU z trzema ramkami w pamięci szybkiej

Tym razem w trzech dolnych wierszach trzymamy zawartość pamięci szybkiej po wczytaniu kolejnej strony w kolejności ostatniego odwołania.

Wyszło nam 10 błędów braku strony. Czy jeśli dołożymy jeszcze jedną ramkę ($k = 4$), to liczba błędów się zwiększy? Zachęcamy Czytelnika do samodzielnego sprawdzenia, ile błędów wtedy wyjdzie. Okazuje się, że w ogólności zachodzi następujące twierdzenie.

Twierdzenie. *Strategia LRU jest wolna od anomalii Bélády'ego. To znaczy, dla dowolnego ciągu odwołań do pamięci S i liczb $k_1 \leqslant k_2$ liczba błędów braku strony*

Rozwiązanie zadania F 1089.
Powstawanie fal na powierzchni jeziora polega na propagacji okresowych zmian głębokości wody. Działającą tu siłą jest siła ciężkości proporcjonalna do masy wody. Miarą bezwładności jest ta sama masa wody, a więc prędkość fal na jeziorze nie zależy od gęstości cieczy wypełniającej jezioro. Można się o tym przekonać na podstawie analizy wymiarowej. Załóżmy, że prędkość propagacji fal c zależy od gęstości cieczy ρ, głębokości cieczy h i wartości przyspieszenia grawitacyjnego g, według wzoru:

$$c \propto g^{\alpha} \cdot \rho^{\beta} \cdot h^{\gamma}.$$

Przeanalizujmy wymiary: kilogram wystąpi tylko w gęstości i nigdzie więcej, a więc $\beta = 0$, czyli prędkość nie zależy od gęstości. Dalsza analiza prowadzi do wniosku, że $\alpha = \gamma = \frac{1}{2}$. Czterokrotny wzrost wartości przyspieszenia grawitacyjnego powoduje więc dwukrotny wzrost prędkości fal.

Czytelnik Dociekliwy, całkiem słusznie, zapyta, dlaczego w analizie wymiarowej uwzględniliśmy głębokość h, a nie uwzględniliśmy długości fali λ, także o wymiarze długości. „Nasz" wzór opisuje sytuację, gdy $h \ll \lambda$. Dla $h \gg \lambda$ otrzymujemy:

$$c \propto \sqrt{g \cdot \lambda}.$$

Dokładna analiza teoretyczna dla fal o małej amplitudzie prowadzi do ogólnego wzoru:

$$c = \sqrt{\frac{g\lambda}{2\pi} \operatorname{tgh} \frac{2\pi h}{\lambda}},$$

w którym „tgh" oznacza funkcję tangens hiperboliczny. Zadanie i przytoczona analiza dotyczy tzw. fal grawitacyjnych. Fale kapilarne opisują inne związki uwzględniające napięcie powierzchniowe.

dla ciągu S przy strategii LRU z k_2 dostępnymi ramkami w pamięci szybkiej nie będzie większa niż dla strategii LRU z k_1 ramkami.

Szkic dowodu. Wystarczy porównać zawartość pamięci szybkiej w obu algorytmach przy danym ciągu odwołań. W każdym momencie algorytm z k_2 ramkami w pamięci szybkiej trzyma k_2 stron, do których ostatnio było jakieś odwołanie. Skoro $k_1 \leqslant k_2$, to algorytm z k_1 ramkami przechowuje podzbiór tych stron. Jeśli więc na jakiejś pozycji w algorytmie z k_2 ramkami występuje błąd braku strony, to tym bardziej wystąpi on w algorytmie z k_1 ramkami. Nie dość więc, że w algorytmie z k_1 ramkami występuje co najmniej tyle samo błędów co w algorytmie z k_2 ramkami, to jeszcze w algorytmie z mniejszą liczbą ramek błąd braku strony występuje zawsze, gdy występuje on w algorytmie z większą liczbą ramek. □

Strategia optymalna

Rozpatrzyliśmy już dwa algorytmy wymiany stron. Dla rozważanego ciągu przy strategii FIFO z trzema ramkami wystąpiło 9 błędów braku strony, a dla strategii LRU z taką samą liczbą ramek było ich 10. Można zadać sobie pytanie, ile najmniej błędów braku strony może się pojawić przy odpowiednio dobranej strategii korzystającej z takiej liczby ramek. No i oczywiście jak taka najlepsza możliwa strategia wygląda.

Okazuje się, że strategię optymalną można bardzo łatwo opisać – z pamięci szybkiej należy zapominać tę stronę, do której odwołanie nastąpi najpóźniej w przyszłości. Dla rozważanego ciągu jeśli zastosujemy strategię optymalną z $k = 3$ ramkami, to po pierwszych trzech błędach braku strony, kiedy to wczytamy do pamięci D, E, C, widząc następne odwołanie do strony A, zapomnimy z pamięci stronę C. Rzeczywiście, do strony D będziemy się odwoływać już w kolejnym kroku, do strony E jeszcze w kolejnym, podczas gdy następne odwołanie do strony C występuje dopiero za sześć kroków. Jak dokładnie zadziała algorytm optymalny dla $k = 3$, przeanalizujmy w poniższej tabelce.

Błąd	*	*	*	*			*			*	*	
Ciąg odwołań	D	E	C	A	D	E	B	D	E	C	A	B
Zawartość szybkiej pamięci	D	D E	D E C	D E A	E D A	D E A	D E B	E B D	B E D	B C E	B A B	B A C

Tabela 4. Strategia optymalna z trzema ramkami w pamięci szybkiej

Tym razem w trzech dolnych wierszach trzymamy zawartość pamięci szybkiej po wczytaniu kolejnej strony w kolejności, w jakiej odwołamy się do tych stron w przyszłości.

Tym razem było tylko 7 błędów braku strony, czyli rzeczywiście mniej niż dla strategii FIFO i LRU. Pozostawiamy Czytelnikowi dowód, że tak zdefiniowana strategia jest rzeczywiście optymalna, czyli dla każdego możliwego ciągu odwołań popełni nie więcej błędów braku strony niż dowolna inna strategia z taką samą liczbą ramek. Oczywiście własność ta oznacza, że strategia optymalna jest wolna od anomalii Bélády'ego.

Skoro znamy strategię optymalną, to czemu w ogóle rozważaliśmy FIFO i LRU? Jako że jest optymalna, to przecież najbardziej opłaca się ją zaimplementować w systemie operacyjnym! Otóż strategia optymalna ma jedną podstawową wadę – nie da się jej zaimplementować w praktyce. Decyzja o tym, którą stronę usunąć, wymagała, żeby przeanalizować, jakie odwołania do stron pojawią się w przyszłości, a przecież żaden komputer nie potrafi przewidzieć, co zaraz zrobi użytkownik. Ma on wiedzę tylko o tym, jakie odwołania pojawiały się w przeszłości. Tak to niestety się czasami zdarza, że teoretycznie najlepsze rozwiązania nie są możliwe do zrealizowania w praktyce. Okazuje się jednak, że strategia LRU z $2k$ stronami popełni z grubsza dwa razy więcej błędów braku strony niż strategia optymalna z k stronami. To jednak temat na zupełnie inny artykuł.

Konkretnie Sleator i Tarjan pokazali w 1985 roku, że jeśli algorytm optymalny z k_{OPT} ramkami popełni t błędów braku strony, to algorytm LRU z $k_{\mathrm{LRU}} \geqslant k_{\mathrm{OPT}}$ ramkami popełni takich błędów co najwyżej

$$\frac{k_{\mathrm{LRU}}}{k_{\mathrm{LRU}} - k_{\mathrm{OPT}} + 1}(t + k_{\mathrm{OPT}}).$$