

RZECZYWISTE UWARUNKOWANIA PROJEKTOWANIA, TWORZENIA I WDRAŻANIA SYSTEMÓW INFORMATYCZNYCH NA PRZYKŁADZIE ALGORYTMÓW SORTOWANIA

Streszczenie. Przedstawiony referat jest omówieniem wybranych aspektów związanych z projektowaniem i tworzeniem systemów informatycznych, w szczególności tych elementów, które są związane z pracą systemu w obszarach krytycznych (o szczególnych wymaganiach). Omówienie obejmuje trzy aspekty związane z implementacją algorytmów sortowania: dobór algorytmu, metody zwiększenia wydajności oraz bezpieczeństwo wykonywania sortowania.

REAL REQUIREMENTS OF DESIGN SOFTWARE SYSTEMS FOR INSTANCE SORT-ALGORITHMS

Summary. The paper presents several problems with design and implementation of software systems, as regards these applications which are connected with working in critical conditions (with particular requirements). In the paper are described three problems connected with implementation of sort-algorithms: a selection of algorithms, methods of increasing productivity and security of running them.

1. Wstęp

Celem stosowania systemów informatycznych jest wspieranie organizacji w wykonywaniu jej poszczególnych zadań. Z uwagi na szeroki zakres stawianych zadań występuje mnogość rozwiązań ich realizujących. Ale nawet mimo istnienia dużego wyboru narzędzi i środków, w wielu przypadkach istnieje potrzeba stworzenia nowego systemu (przykładem może być większość systemów pisanych na zamówienie rządowe) lub głębokiej adaptacji systemu do potrzeb danej instytucji (np. wdrożenia systemów klasy ERP). Podczas wdrażania wielokrotnie można się natknąć na potrzebę wprowadzenia nowych funkcji i modułów. Celem niniejszego artykułu jest ukazanie kilku rzeczywistych uwarunkowań, z powodu których niemożliwe staje się proste zastosowanie standardowego narzędzia czy rozwiązania. Przypadki takie występują w obszarach, które nazywane bywają obszarami krytycznymi bądź zastosowaniami krytycznymi. Przymiotnik krytyczne ma sygnalizować istnienie szczególnych wymagań np. co do ilości obiektów, niezawodności, szybkości wykonywania. Z uwagi na obszerność tematyki omówiono jeden moduł mianowicie moduł **sortowania**. Wybrano go z kilku powodów. Po pierwsze:

algorytmy sortowania są bardzo dobrze zbadane, posiadają udowodnioną poprawność oraz wzory obliczające koszt obliczeń. Po drugie istnieje wiele implementacji poszczególnych algorytmów w różnych językach i dla różnych komputerów. Po trzecie są powszechnie stosowane: trudno jest znaleźć system informatyczny, w którym nie byłaby umieszczona procedura sortująca, znacznie łatwiej system w którym zawarty jest kilka różnych.

2. Definicje stosowanych pojęć

Na wstępie zostaną zdefiniowane podstawowe pojęcia stosowane przy omówieniu zagadnienia.

3. Baza danych

Bazą danych nazywamy zbiór danych o określonej strukturze, zapisany na zewnętrznym nośniku pamięciowym komputera, mogący zaspokoić potrzeby wielu użytkowników korzystających z niego w sposób selektywny w dogodnym dla siebie czasie [1].

Możliwe są różne poziomy opisu bazy danych:

- poziom wewnętrzny – określa sposoby organizacji danych w pamięci zewnętrznej komputera
- poziom pojęciowy – następuje w nim modelowanie obiektów świata rzeczywistego do przyjętego modelu danych
- poziom zewnętrzny – odnosi się do sposobu, w jaki dane są widziane przez poszczególne grupy użytkowników

Zbiory danych wraz z narzędziami do jej zarządzania nazywamy **systemem bazy danych**. Pojęcie to należy rozumieć bardzo szeroko, obejmuje ono zarówno komercyjne systemy zawierające szeroki zakres narzędzi, jak i rozwiązania autorskie, stworzone na potrzeby danego zastosowania. W poziomie wewnętrznym poszczególne elementy bazy danych są reprezentowane przez rekordy. Na tym poziomie bazą danych nazywa się zbiór tablic, z których każda jest zbiorem rekordów tego samego typu i każdy rekord posiada swój numer kolejny odpowiadający położeniu w tablicy. Założoną strukturę należy pojmować w sposób abstrakcyjny i umowny, jest to potrzebne do zdefiniowania relacji porządku i operacji porządkowania bazy danych. Zakładamy również, że każdy rekord ma pole będące kluczem o zdefiniowanym sposobie określenia kluczy mniejszych i większych.

Porządkowanie bazy danych ma na celu umożliwienie prezentacji danych na poziomie zewnętrznym zgodnie z założeniami tj. w określonym porządku.

Sortowanie tablicy polega na takim pozamienianiu kolejności położenia poszczególnych rekordów, by został spełniony warunek uporządkowania, czyli dla każdego rekordu w tabeli wartość klucza jest większa lub równa od wartości klucza rekordu poprzedniego.

Sortowanie bazy danych w poziomie wewnętrznym polega na posortowaniu poszczególnych tablic.

4. Uwarunkowania eksploatacyjne

Jak zauważono wcześniej struktura bazy danych musi zaspokoić potrzeby użytkowników systemu. Analizując różne systemy baz danych należy zwrócić uwagę na kilka problemów związanych z dopasowaniem bazy danych do rzeczywistych potrzeb użytkowników:

1. Nie ma idealnego systemu baz danych, który nadaje się do każdego zastosowania. Inne wymagania mają hurtownie danych, inne systemy finansowo-księgowo, inne też systemy sterowania produkcją.
2. Systemy baz danych jako takie wprowadzają pewne mechanizmy i logikę zarządzania bazami danych, co może być przeszkadzające w realizacji danego zadania.
3. Standardowe funkcje dostępne w systemie bazy danych (np. porządkowanie czy wyszukiwanie) mogą nie spełniać wymaganych kryteriów i należy je rozbudować bądź utworzyć nowe. Do takich kryteriów należą między innymi:
 - ograniczenie implementacyjne (np. w PARADOX'ie, jeżeli tablica posiada zdefiniowany klucz, wynik sortowania jest zapisywany do innej tablicy),
 - zbyt długi czas wykonywania,
 - niewystarczająca odporność na błędy i uszkodzenia związana z awariami,
 - niewystarczająca kontrola podczas współużytkowania bazy danych.

Przeanalizujmy teraz problem porządkowania bazy danych.

5. Metody porządkowania bazy danych

W bazie danych rekordy są ułożone w określonym porządku. Porządek ten wynika z założonego układu danych wymaganych przez użytkownika. W zależności od przeznaczenia niektóre dane powinny być posortowane alfabetycznie, niektóre chronologicznie. Jak powiedziano wcześniej **sortowanie** bazy danych polega na ułożeniu rekordów w założonym porządku. Samo sortowanie jest tylko jedną z wielu metod uzyskiwania założonego porządku.

Inną metodą jest także **wprowadzenie danych do bazy, aby w żadnym momencie porządek nie był zachwiany**. Okupione to jest jednak dodatkowym nakładem obliczeń związanych ze wstawieniem rekordu we właściwe miejsce.

Jeszcze inną metodą jest posługiwanie się **zbiorami indeksowymi** tj. zbiorami pomocniczymi zawierającymi odnośniki do poszczególnych rekordów i powiązania między nimi (w układzie listy, sterty czy kopca). Największą zaletą stosowania tego rozwiązania jest możliwość uzyskania tablicy w kilku układach, które mogą być dowolnie zmieniane z niemal natychmiastowym skutkiem. Może na przykład istnieć kilka zbiorów indeksowych, z których każdy przechowuje inną kolejność rekordów wyznaczoną inną relacją porządku lub innym kluczem. Wadami rozwiązań indeksowych jest konieczność posiadania specjalnych narzędzi (bibliotek funkcji) umożliwiających tworzenie indeksów, a następnie operowania na nich (takie procedury są zawarte we wszystkich systemach baz danych). Zbiory indeksowe również wymagają sortowania, ale z uwagi na zupełnie inną strukturę oraz inne uwarunkowania (są w pełni odtwarzalne na podstawie głównego zbioru) wymagają innych procedur. Dodatkowo stosowanie indeksów okupione jest dodatkowymi nakładami czasowymi (odwołanie do rekordu następuje za pośrednictwem wskaźnika zawartego w zbiorze indeksowym) oraz zwiększoną wrażliwość na błędy. Nie ma również przeszkód, by zastosować oba rozwiązania jednocześnie tzn. utrzymywania relacji porządku w kolejności ułożenia samych rekordów i jednocześnie stosowanie indeksów do uzyskania widoku tablicy w innym.

6. Zasady porządkowania

Zastanawiając się nad zastosowaniem właściwego algorytmu sortowania, oprócz jego złożoności, należy wziąć pod uwagę warunki uporządkowania końcowego. Należy do nich na przykład warunek niezmienniania kolejności zapisu operacji o tym samym kluczu. Ponieważ wiele rekordów może posiadać tę samą wartość klucza (nie jest zachowana zasada unikalności klucza), należy zbadać wpływ zmiany kolejności ułożenia rekordów o tym samym kluczu na dalszą pracę z bazą danych.

Algorytm sortowania dla dwóch rekordów o równych kluczach może zachowywać się w jeden z trzech sposobów:

1. Algorytm nie zmieni ich wzajemnego położenia.
2. Algorytm zawsze je zmienia.
3. Kolejność rekordów jest nieokreślona.

Istnieją takie przypadki, w których zmiana kolejności uporządkowania rekordów o tych samych kluczach jest niedopuszczalna, wtedy więc należy stosować algorytmy zachowujące tą kolejność.

7. Wybór algorytmu sortowania

Algorytmy sortowania są bardzo znaną dziedziną informatyki, w której pojawiło się bardzo dużo prac naukowych. Praktyczna implementacja tych algorytmów w rzeczywistych aplikacjach natrafia jednak na szereg uwarunkowań związanych przede wszystkim

z otoczeniem i środowiskiem w jakim dana aplikacja ma pracować. Poniżej zostaną omówione wybrane algorytmy wraz z cechami kwalifikującymi je do poszczególnych zastosowań.

Stosując notację przyjętą w [2] i [3] zakładamy, że rekord bazy danych ma postać:

```
type typ_rekordu= rekord
      klucz : typ_klucza;
end;
```

Każdy rekord ma pole będące kluczem o typie `typ_klucza`, w zbiorze wartości typu `typ_klucza` musi być zdefiniowana relacja liniowego porządku. Pole `klucz` nie musi być jawnie deklarowane w strukturze rekordu, może ono być wyliczane na podstawie jednego bądź wielu pól tego rekordu.

Zakładamy, że tablica podlegająca sortowaniu ma rozmiar n , gdzie n jest liczbą naturalną

```
var array[1..n] of typ_rekordu
```

Omawiając poszczególne algorytmy posłużymy się pojęciem złożoności obliczeniowej algorytmu rozumianej jako liczbę kroków wykonywaną przez algorytm. Z założeń wynika, że złożoność ta będzie funkcją n .

Najbardziej znanymi algorytmami sortowania są:

1. Sortowanie przez proste wstawianie, proste wybieranie, bąbelkowe.

Zaletą ich jest bardzo prosty zapis algorytmu, wadą duża złożoność obliczeniowa rzędu $n*n$. Najczęściej stosowane do sortowania małych tablic, szczególnie tam gdzie nie jest opłacalne stosowanie zewnętrznych bibliotek funkcji.

Dla przykładu poniżej zamieszczono algorytm sortowania przez proste wybieranie [2]

```
for i:=1 to n-1 do
  „przypisz zmiennej k indeks rekordu o najmniejszym
  kluczu spośród rekordów A[i..n]”
  „Wymień A[i] z A[k]”
end for;
```

2. **Algorytm Shella** - czyli sortowanie za pomocą malejących przyrostów. Jego podstawową wadą jest silna zależność złożoności od doboru przyrostów i gorsza efektywność od algorytmu Quicksort
3. Sortowanie szybkie – algorytm **quicksort**. Często stosowany algorytm o bardzo dużej wydajności, w wielu bibliotekach znajdują się jego implementacje (co gwarantuje ich prawidłowość oraz prawidłową optymalizację ze względu na komputer i system operacyjny dla którego są one napisane). Ma dobrą wydajność pod warunkiem prawidłowo wybieranego klucza osiowego (dla założeń pesymistycznych złożoność wynosi $n*n$ a dodatkowo jest utrudniona niezawodna implementacja tak, aby wystarczyło zasobów do

wykonania sortowania w szczególności stosu, gdyż część implementacji jest rekurencyjna). Jednak dla wielu zastosowań, i to zarówno dla tablic o przypadkowym układzie jak i częściowo uporządkowanych złożoność jest bliska $n \cdot \log n$. Wadą algorytmu jest niemożność określenia wzajemnego ułożenia rekordów o równych wartościach klucza tzn. algorytm w zależności od ułożenia innych rekordów może ale nie musi zmienić kolejności ułożenia takich rekordów.

4. Algorytmy sortowania, w których korzysta się ze szczególnych własności kluczy, czyli sortowanie **kubelkowe i pozycyjne**. Zaletą stosowania tych algorytmów jest bardzo duża wydajność dla niektórych rodzajów zastosowań, gdy można właściwie wykorzystać właściwości klucza (tzn. jest znany rozkład występujących wartości kluczy). Złożoność obliczeniowa tych algorytmów jest bardzo niska, nawet rzędu n , wadą jest utrudniona implementacja, która musi uwzględniać szczególne właściwości danego zastosowania. Dodatkowo, z uwagi na wykorzystywane typy listowe, algorytmy te są trudne do implementacji dla bardzo dużych tablic z uwagi na ograniczenia zasobów pamięci.
5. Sortowanie przez **łączenie naturalne** służy do sortowania plików sekwencyjnych. Jego podstawową cechą jest to, że nie wymaga swobodnego dostępu do czytanych danych tylko pobiera je sekwencyjnie. Ponieważ tablice w bazach danych zapisywane są w plikach ten sposób sortowania znakomicie się do nich nadaje. Po pierwsze nie ma ograniczeń co do wielkości bazy danych ze względu na posiadaną pamięć operacyjną, przy czym okupione jest to znacznymi potrzebami pamięci dyskowej (tyle samo co zajmuje oryginalna tablica). Po drugie mimo, że obecnie swobodny dostęp do pliku tablicy (odczyt i zapis rekordów w dowolnej kolejności) jest prawie zawsze możliwy, to dostęp sekwencyjny jest znacznie szybszy w szczególności wspierane przez buforowanie (zarówno na poziomie aplikacji jak i systemu operacyjnego czy samego sprzętu). Złożoność obliczeniowa jest rzędu $n \cdot \log n$, przy czym znamienne jest, że dla danych częściowo uporządkowanych jest znacząco mniejsza, a dla tablicy uporządkowanej wynosi n .

8. Metody zwiększania wydajności procedur sortowania

Uwarunkowania eksploatacyjne określają na jakich danych będzie pracować aplikacja, jakie operacje będą na nich wykonywane, jakie zasoby sprzętowe są do tego wykorzystywane itp. Korzystając z tej wiedzy można tak zmodyfikować zastosowane algorytmy by zoptymalizować ich złożoność obliczeniową oraz, co się z tym wiąże, czas ich wykonywania.

9. Rozkład uporządkowania danych

Brak uporządkowania może być spowodowany wieloma czynnikami, które można podzielić na dwie grupy. Pierwszą jest niezgodność kolejności wprowadzania danych i ich późniejszego układu w bazie. Część z danych jest z założenia wprowadzana przypadkowo (ze względu na wymagany porządek ułożenia), szczególnie jeżeli jest to uporządkowanie alfabetyczne lub podobne. Natomiast bazy o porządku chronologicznym są zapełniane w sposób wysoce skorelowany z ich późniejszym porządkiem. Z tego też względu można się spodziewać różnego uporządkowania przed sortowaniem w zależności od specyfiki danej bazy. Tablice zawierające dane chronologiczne, czyli te przechowujące zapis historii transakcji, z racji swojej objętości mają znaczący wpływ na globalny czas sortowania (tablice takie stopniowo przyrastają, i biorąc pod uwagę ich wieloletni czas życia, objętość tablic z danymi typu chronologicznego jest znacząco większa od tablic przechowujących dane parametryczne, obraz, stany obiektu czy nawet katalog obiektów). Dla uzyskania oszczędności należy więc w pierwszej kolejności zoptymalizować procedury dotyczące sortowania najdłuższych tablic, czyli tych zawierających dane chronologiczne. Najprostszą rzeczą jest wykorzystanie właściwości częściowego uporządkowania. Większość rekordów, tj. prawie wszystkie wprowadzone przed poprzednio wykonanym sortowaniem, ma właściwe położenie, nieuporządkowane są jedynie te później dodane, a i wśród nich niewiele jest takich które spowodują zmianę położenie rekordów wcześniejszych. Wykorzystanie tej właściwości polega na tym, aby wstępnie posortować ostatni fragment tablicy, następnie należy sprawdzić czy tablica nie jest już właściwie uporządkowana. Testowanie ma na celu uniknięcie sytuacji, w której sortowaniu poddana jest uporządkowana tablica. Jest to istotna czynność, gdyż znakomicie skraca czas potrzebny na sortowanie tablicy w przypadkach gdy nieuporządkowana jest tylko jej końcowa część. W wypadku gdy test ma wynik negatywny poddajemy sortowaniu całość. Dużą trudność sprawia optymalne określenie wielkości fragmentu wstępnie sortowanego, można go przyjąć jako pewien procent całości (np. ostatnie 10% tablicy) lub oszacować przeciętną ilość rekordów dopisanych od ostatniego sortowania (np. wyniesie to 3000 rekordów). Dla przyjętych założeń można obliczyć oszczędności czasowe. Przy założeniu optymistycznym gdy tablica nie wymaga porządkowania całości i przyjęciu złożoności algorytmu sortowania $n \cdot \log n$, poszczególne czasy wynoszą:

- Sortowanie całości tablicy $T_c = n \cdot \log n$
- Czas sortowania końcowego fragmentu T_k (zakładamy jego wielkość na 10 %):

$$T_k = 0.1 \cdot n \cdot \log(0.1 \cdot n) = 0.1 \cdot n \cdot (\log n - 1) < 0.1 T_c$$
- Czas sprawdzenia uporządkowania $T_s = n = T_c / \log n$;
- Czas całkowity przy założeniu optymistycznym T_{op}

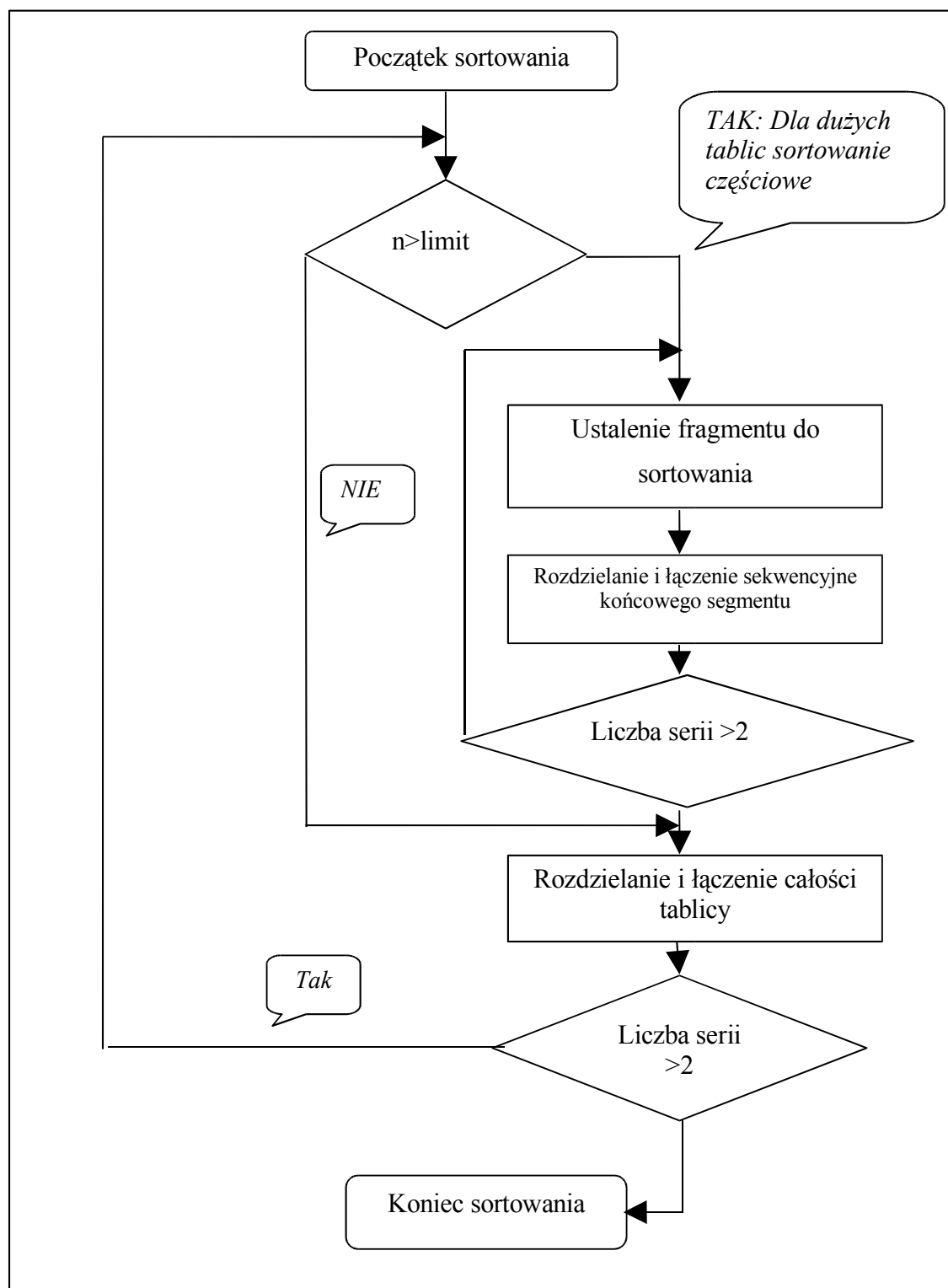
$$T_{op} = T_k + T_s = 0.1 \cdot T_c + (1 / \log n) \cdot T_c = T_c \cdot (0.1 + 1 / \log n)$$
- Czas całkowity przy założeniu pesymistycznym T_{ps}

$$T_{ps} = T_k + T_s + T_c = T_c \cdot (1 + 0.1 + 1/\log n);$$

Wynika z tego, że przy założeniach optymistycznych oszczędności czasowe są znaczne, i to tym większe, im większa jest baza danych (nawet kilkukrotne, w rzeczywistych implementacjach obserwuje się prawie liniową zależność czasu sortowania od wielkości tablicy (czyli n zamiast $n \cdot \log n$ lub co gorsza n^2)).

Na Rysunku 1 przedstawiono omawianą procedurę sortowania częściowego, opartą o algorytm sortowania przez łączenie naturalne.

Czas sortowania fragmentu tablicy jest mały (proporcjonalny do jego wielkości, w przyjętych założeniach wynosi mniej niż 10%). Przy wykorzystaniu algorytmów, których złożoność obliczeniowa maleje wraz ze wzrostem uporządkowania danych, dodatkowy nakład czasowy związany z sortowaniem ostatniej części tablicy jest rekompensowany zmniejszeniem czasu potrzebnego na posortowanie całości (gdyż jej część jest już uporządkowana). Takim algorytmem jest sortowanie przez łączenie naturalne, szczególnie że ilość nawrotów jest silnie zależna od stopnia uporządkowania. W najlepszym przypadku, dla tablicy uporządkowanej wynosi 1, dzięki czemu nie jest wymagana dodatkowa procedura testująca uporządkowanie (wykorzystuje się licznik utworzonych serii). Algorytm znakomicie się nadaje do sortowania długich zbiorów danych, o niewielkich zaburzeniach porządku, np. takich jakie występują w tablicach z danymi chronologicznymi. W połączeniu z zastosowaniem sortowania częściowego można oczekiwać złożoności obliczeniowej rzędu n . Dodatkowym atutem jest możliwość stosowania buforowania jako techniki optymalizującej dostęp do danych.



Rysunek 1. Procedura sortowania częściowego oparta o algorytm sortowania przez łączenie naturalne.

Fig.1. Procedure for partial sorting based on natural connection sorting algorithm.

10. Wpływ szybkości wykonywania poszczególnych operacji na szybkość wykonywania sortowania

Podczas sortowania operacjami krytycznymi pod względem czasu wykonywania są operacje obliczenia kluczy (jeżeli klucz nie jest rzeczywistym polem rekordu) i związane z tym porównanie kluczy oraz operacje pobrania i zapisu rekordów (przy przestawianiu na inną pozycję). Czasy obliczeń zależą od rzeczywistych konstrukcji rekordu oraz od jego wielkości. Obliczenie klucza jest z reguły krótką operacją, choć w niektórych tablicach, gdy porządek jest np. wynikiem odwzorowania uporządkowania z innej tablicy, czas obliczenia klucza może się znacznie wydłużać, natomiast czas porównania jest tak mały że można go pominąć. Dla dużych tablic, które są przechowywane w pamięciach masowych czas dostępu, operacji zapisu i odczytu, jest znacznie dłuższy (kilka rzędów) od tablic umieszczonych w pamięci operacyjnej. Dla uzyskania krótszego czasu dostępu można stosować technikę buforowania danych, czyli tymczasowego przechowywania części tablicy w pamięci operacyjnej. Ponieważ jednak czas wymiany danych pomiędzy buforem i pamięcią masową jest długi, należy tak zoptymalizować proces sortowania aby uniknąć częstej potrzeby odczytu lub zapisu rekordu z poza obszaru obejmowanego przez bufor, w niektórych bowiem przypadkach wprowadzenie tej techniki zmniejsza szybkość zamiast ją zwiększyć.

11. Bezpieczeństwo wykonywania

12. Wrażliwość bazy danych na błędy sortowania

Komputer, jak każda maszyna, ma pewien poziom niezawodności który jest mniejszy od 100%, czyli istnieje prawdopodobieństwo że nie wykona danej operacji bądź wykona ją źle. Oczywiście istnieje wiele technik zwiększających niezawodność, ale nigdy nie można założyć że nie nastąpi błąd wykonania. Dodatkowym utrudnieniem jest czynnik ludzki, od którego np. zależy niedopuszczenie do przepełnienia pamięci masowej, a który to stan występuje w rzeczywistym świecie.

Operacja sortowania jest bardzo wrażliwa na błędy, jest to spowodowane wykonywaniem bardzo wielu operacji modyfikacji bazy danych w krótkim czasie. Po pierwsze powoduje to znaczne obciążenia komputera (w tym szczególnie pamięci masowych, sieci) i część uszkodzeń ujawnia się właśnie w takich momentach. Dodatkowo modyfikacji podlegają duże fragmenty tablic i to wielokrotnie, każdy błąd może spowodować znaczne uszkodzenia informacji w bazie danych.

13. Kontrola pamięci (sterta i stos)

Jedną z metod zwiększenia odporności na błędy i uszkodzenia jest ich wykrywanie zanim spowodują powstanie uszkodzeń. Część z komputerów posiada wbudowane procedury diagnostyczne, które są regularnie uruchamiane. Dotyczą one w szczególności pamięci operacyjnej i masowej, procesora. Nie są one w stanie skontrolować i wykryć błędów przepełnienia sterty, stosu, powstałych w wyniku przypadkowych modyfikacji kodu programu (lub innych ważnych miejsc) w pamięci operacyjnej. Błędy takie mogą być wynikiem uszkodzenia komputera ale częściej przez wadliwie działający program, i to niekoniecznie bieżący. Powszechne stosowanie środowisk wielozadaniowych wprowadziło nową kategorię błędów spowodowanych wadliwym działaniem innego zadania czy procesu, system operacyjny nie zawsze gwarantuje odpowiednią niezależność ich wykonywania. Lekarstwem na to jest wprowadzenie procedur diagnostycznych kontrolujących na bieżąco zasoby systemu, przed wejściem do obszarów krytycznych. Można też wprowadzić metodę sum kontrolnych dla kontroli i diagnostyki pewnych fragmentów pamięci operacyjnej (np. tę zawierającą kod programu).

14. Kontrola dostępu (wpływ sortowania na innych użytkowników)

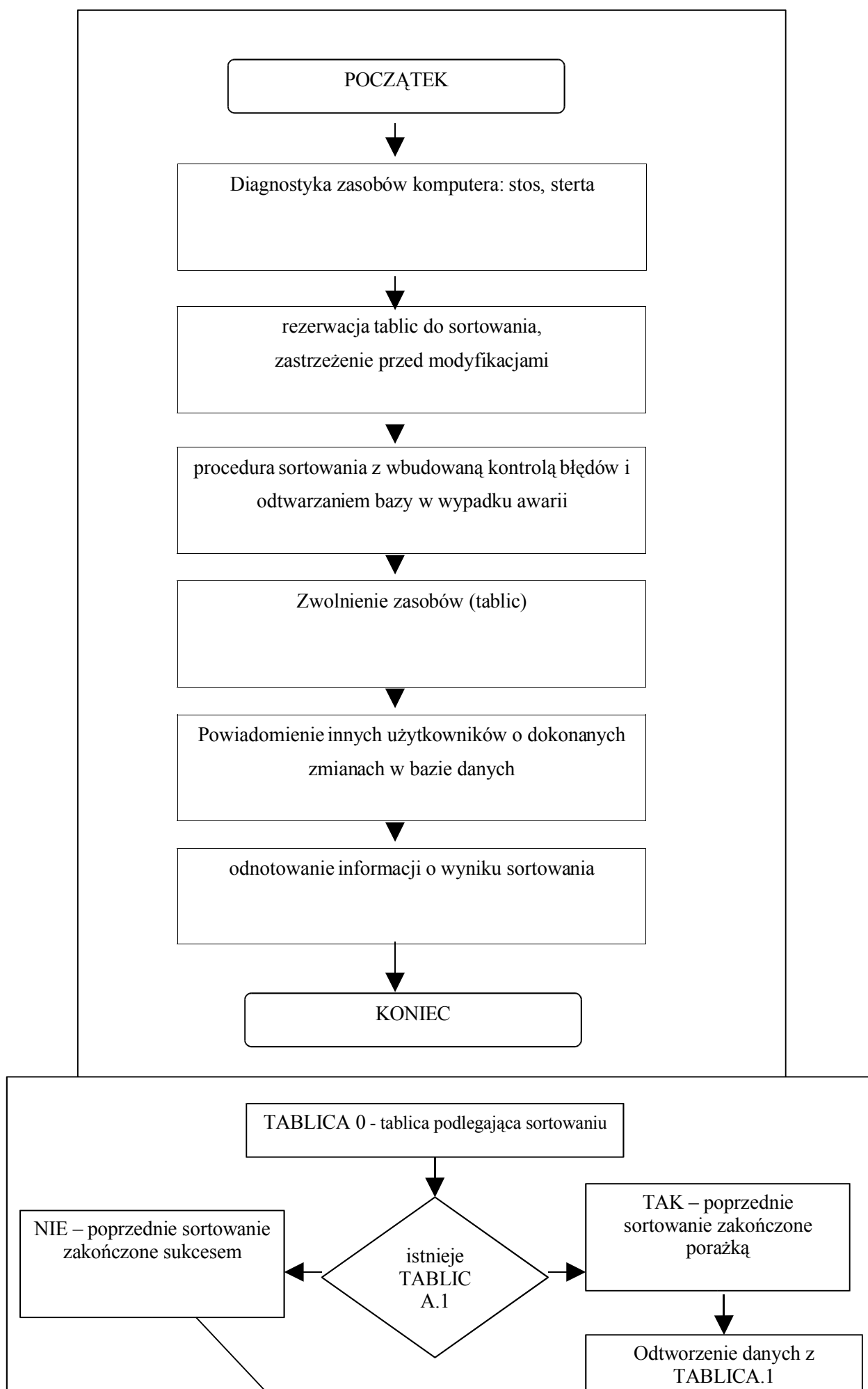
Ponieważ sortowanie bazy danych polega na zmianie położenia rekordów podczas pracy sieciowej, gdy z bazy korzysta wielu użytkowników, zmiany takie mogą być powodem powstania sytuacji błędnych. Wiele aplikacji korzysta z różnego rodzaju indeksów i wskaźników obrazujących położenie rekordów aktualnie poddawanych przetwarzaniu. Dlatego też jeżeli położenie danego rekordu się zmieni i w jego miejsce zostanie umieszczony inny rekord to aplikacja wykonując swoje zadania może w najgorszym przypadku dokonać błędnych zapisów. Dla uniknięcia takich sytuacji należy wprowadzić mechanizmy obronne i to z obu stron : aplikacji sortującej i aplikacji przetwarzającej bazę danych. Po stronie aplikacji sortującej należy wprowadzić kontrolę używania poszczególnych tablic przez inne procesy (obecnie sieciowe systemy operacyjne dają wiele możliwości kontrolowania dostępu do zasobów). Najprostszym rozwiązaniem jest oczekiwanie chwili, gdy żadna tablica nie jest używana, czyli zająć całą bazę danych. W dużych bazach danych używanych przez wielu użytkowników stan taki jest trudny do uzyskania więc proces sortowania musi być tak skonstruowany, aby zabezpieczał przed wykonywaniem operacji przez procesy mające nieaktualne informacje o niej. Można to uzyskać wprowadzając metodę powiadamiania o zmianie bazy danych (np. system Windows NT i Windows 95/98 posiada wbudowane mechanizmy buforowania plików z automatycznym powiadamianiem o zmianie ich oryginału), każdy proces przetwarzający bazę danych przed przystąpieniem do wykonania operacji krytycznych (tzn. takich dla których zmiana ułożenia rekordów może spowodować sytuacje błędne) sprawdzają czy nie otrzymały sygnału o sortowaniu. Na czas samego sortowania proces sortujący musi mieć wyłączność na dostęp do tablicy (przynajmniej wyłączność na zapis),

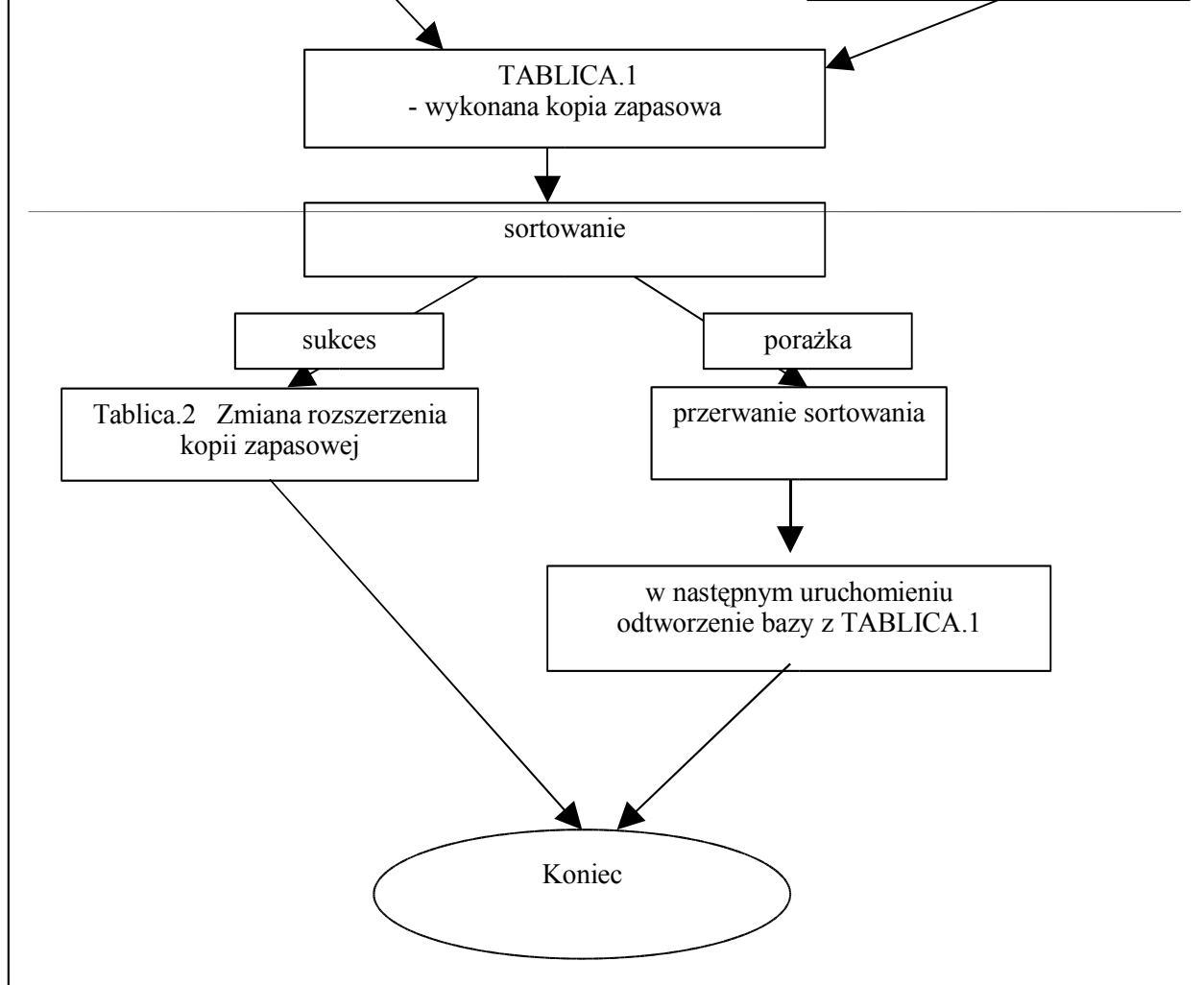
jeżeli jednak takie blokowanie wprowadza zbyt długą przerwę w dostępie do niej można tak zmodyfikować algorytm sortujący, aby podzielić go na etapy, pomiędzy którymi zostaną wprowadzone przerwy na obsłużenie innych procesów (warunkiem jest aby po każdym etapie tablica była spójna fizycznie i logicznie).

15. Metody wykrywania uszkodzeń

Zabezpieczeniem bazy danych przed uszkodzeniem podczas sortowania jest wykonanie kopii zapasowej tablic. Można niejako przy okazji wprowadzić mechanizm odzyskiwania danych po awarii. Jak już wspomniano powyżej konwencjonalne mechanizmy typu TTS (Transaction Tracking System) wbudowane w systemy baz danych nie są najodpowiedniejsze do chronienia baz podczas sortowania. Wystarczy jednak wykonać kopię zapasową tablicy i w wypadku uszkodzenia, przywrócić dane z kopii. Do bazy należy jeszcze wprowadzić znaczniki tak, aby aplikacja mogła automatycznie rozpoznać czy poprzednie sortowanie zakończyło się sukcesem czy porażką. Przykładowo można wprowadzić znaczące rozszerzenia do nazwy tablicy. Dodatkową zaletą tego mechanizmu jest niejako automatyczne tworzenie kopii zapasowej. Poważną wadą jest dwukrotne powiększenie ilości zajmowanego miejsca na przechowanie tymczasowych kopii zapasowych, co jednak przy obecnych pojemnościach dysków nie wymaga dużych nakładów.







Rysunek 3. Pełna procedura sortowania
Fig.3. The full sorting procedure

16. Wnioski

Podczas projektowania i analizy systemów informatycznych stosuje się wiele technik wspomagających określone czynności w poszczególnych zakresach:

- w zakresie potrzeb informacyjnych użytkownika
- w zakresie funkcji systemu
- w zakresie modelowania danych

Do programowania używa się wielu dostępnych narzędzi, bibliotek i funkcji. Stosowanie ich znacząco podnosi wydajność i jakość tworzonego oprogramowania. Istnieje jednak wiele zastosowań, dla których narzędzia te, gdzie indziej dobrze pracujące, zawodzą. Należy wtedy dokładnie zidentyfikować przyczynę i na tej podstawie rozbudować, przebudować lub stworzyć całkowicie nowy moduł czy funkcję. Niniejszy artykuł miał na celu ukazanie, że implementacja nawet tak dobrze znanej i udokumentowanej funkcji jaką jest sortowanie, w niektórych zastosowaniach wymaga szczegółowych badań związanych z istnieniem rzeczywistych uwarunkowań.

Niewłaściwe zaprojektowanie systemu jest przyczyną wielu błędów, a co za tym idzie, ponoszenia dodatkowych kosztów. Można przypuszczać, że gro takich błędów występuje w obszarach krytycznych, z uwagi na trudność w ich zidentyfikowaniu i oszacowaniu

rzeczywistych wymagań. Część błędów pojawia się dopiero na etapie ostatecznego wdrożenia lub po określonym czasie pracy, gdy baza danych przekroczy krytyczną objętość. Odpowiednio wczesne wykrycie takich miejsc i zastosowanie odpowiednich środków zaradczych spowoduje uniknięcie wielu przykrych konsekwencji. Lekarstwem na to jest stosowanie doświadczonego zespołu projektantów i analityków, którzy potrafią zidentyfikować obszary krytyczne mogące wystąpić w danym zastosowaniu (gdyż wymagana jest zarówno wiedza informatyczna jak i wiedza o funkcjonowaniu informatyzowanej organizacji) oraz przeznaczenie odpowiednich środków na etap pielęgnacji i konserwacji systemu.

LITERATURA:

1. Claude Delobel, Mochel Adiba: Relacyjne bazy danych. Wydawnictwo Naukowo Techniczne. Warszawa 1989.
2. Zbigniew Czech: Analiza Algorytmów. Politechnika Śląska Instytut Informatyki. Gliwice 1990.
3. Lech Banachowski, Antoni Kreczmar, Wojciech Rytter: Analiza algorytmów i struktur danych. Wydawnictwo Naukowo Techniczne. Warszawa 1987.
4. Piotr Kowalski, Rola czynnika czasu w konwersji systemów informatycznych. Publikacje z konferencji Inteligentne Systemy wspomaganie decyzji w zarządzaniu Transformacje systemów pod redakcją H.Sroki i S. Stanka Katowice 1997
5. G.Bilewicz E.Ziemia Wybrane problemy wykorzystania semiformalnych metod specyfikacji oprogramowania. Katowice 1997
6. Dariusz Mazur. Skalowalność w systemach informatycznych zarządzania (niepublikowane).

Abstract

The database, which is chosen during the design of software systems, must be adapted to real requirements of users. There are special critical conditions, when designers must implement new functions, for example: sort-algorithm, which is described in the paper. There are presented three problems:

1. The important signs of fundamental sort algorithms
2. The propositions the methods of increase the productivity, for example to take into consideration the fact, that in some databases most information are already sorted.
3. The methods of increasing the security, in particular to control memory and access of users during the running the sort procedures.