WYBRANE METODY DOSTĘPU DO DANYCH

1. WSTĘP

Coraz doskonalsze, szybsze i pojemniejsze pamięci komputerowe pozwalają gromadzić i przetwarzać coraz większe ilości informacji. Systemy baz danych stanowią więc jedno z podstawowych zastosowań informatyki. Efektywna organizacja danych wymaga specjalnego oprogramowania. W kolejnych podpunktach zostaną przedstawione pojęcia i problemy związane z gromadzeniem i przetwarzaniem danych.

Zacznijmy od zdefiniowania pojęcia bazy danych. Baza danych jest to zbiór danych o określonej strukturze, zapamiętany w pamięci pomocniczej komputera, mogący zaspokoić potrzeby wielu użytkowników korzystających z niego w sposób selektywny w dogodnym dla siebie czasie (definicja sformułowana w latach 1962-1963).

Oprogramowanie umożliwiające współpracę użytkownika z bazą danych jest nazywane systemem zarządzania bazą danych. Oprogramowanie to zarządza przede wszystkim organizacją danych w pamięci oraz umożliwia ich przetwarzanie.

Wyróżnia się trzy główne klasy modeli danych, różniące się rodzajem istniejących między danymi powiązań. Są to modele: hierarchiczne, sieciowe i relacyjne. Obecnie najbardziej rozpowszechniony jest model relacyjny, w którym zależności między danymi reprezentowane są przez rekordy danych zawierające odpowiednie klucze.

W ogólnie rozumianych zagadnieniach baz danych należy wyróżnić trzy podstawowe problemy:

1. *Problem rozmieszczenia*, polegający na podaniu takiego algorytmu A, który na podstawie klucza K_i zawartego w rekordzie R_i przydzieli miejsce (adres) dla tegoż rekordu w określonej strukturze S.
2. *Problem odszukania* rekordu R_i w strukturze S polega na ustaleniu adresu tego rekordu, na podstawie klucza K_i i algorytmu A.
3. *Problem wyszukania*, występuje wówczas, gdy na podstawie atrybutów innych niż klucz należy wyszukać odpowiedni rekord. Rozwiązanie tego problemu polega na przetesto-waniu pól wszystkich rekordów i porównaniu ich zawartości z wartością zadanego atrybutu.

2. CHARAKTERYSTYKA WYBRANYCH STRUKTUR DANYCH

Istniejące struktury danych można ogólnie podzielić na dwie grupy: struktury stałe, niezależne od zebranych w nich elementów oraz struktury dynamicznie zmienne, zależne od zgromadzonych w nich elementów.

Do struktur stałych należą: wektory, macierze, rekordy i tablice, do struktur dynamicznych: listy liniowe, nieliniowe i sieci.

Prezentację struktur zaczniemy od tablicy. Tablica jest strukturą jednorodną, składającą się ze składowych tego samego typu zwanego typem podstawowym, jest strukturą o dostępie swobodnym bowiem wszystkie elementy mogą być wybrane w dowolnej kolejności i są jednakowo dostępne. Tablica jednowymiarowa o rozmiarze n oznaczana jest $T[n]$. Tablica dwuwymiarowa jest analogiem macierzy prostokątnej. Przedstawić ją można jako m szeregowo ustawionych tablic o rozmiarze n. W odróżnieniu od tablicy rekord jest uporządkowanym zbiorem danych niekoniecznie tego samego typu.

Powyżej opisane struktury zwane są często podstawowymi, gdyż stanowią elementy, z których tworzy się bardziej złożone struktury. Zdefiniowanie typu danych i następnie określenie specyfikacji zmiennych tego typu oznacza, że zakres wartości przyjmowanych przez te zmienne i równocześnie ich wzorzec pamięciowy są ustalone raz na zawsze. Zmienne zadeklarowane w ten sposób nazywane są statycznymi. Jednakże, jest wiele zagadnień wymagających daleko bardziej skomplikowanych struktur danych. Charakterystyczną cechą tych struktur, wyróżniającą je spośród struktur podstawowych (tablic, rekordów czy zbiorów), jest możliwość zmieniania ich rozmiarów. Tak więc nie można przydzielić stałej ilości pamięci dla tego typu struktur, w konsekwencji czego kompilator nie może określić konkretnych adresów składowych takich zmiennych. Struktury takie nazywamy strukturami dynamicznymi.

Głównymi reprezentantami tych struktur są listy, drzewa i sieci. Wszystkie rekordy wchodzące wskład takich struktur mają jedno lub kilka dodatkowych pól, zwanych polami łącznikowymi lub polami wskaźniko-wymi. W praktyce pola te zawierają adresy elementów połączonych z danym rekordem.

Zbiór, w którym każdy element ma co najwyżej jeden poprzednik i jeden następnik, nazywamy *listą liniową*. Szczególnymi przypadkami listy liniowej są:

- *lista cykliczna*, w której nie można wyróżnić początkowego ani końcowego elementu oraz listy o specjalnych metodach dostępu takie jak:
- *stos* - w którym dopisywanie i usuwanie elementów odbywa się z jednego końca,
- *kolejka* - rekordy są dopisywane z jednego końca a usuwane z drugiego (tzn. usunięty może być tylko najwcześniej wpisany element).

Strukturę, w której element może posiadać wiele następników, lecz jeden poprzednik nazywamy *listą nieliniową* lub *drzewem*. Drzewo, w którym liczba następników wynosi zero, jeden lub dwa, *to drzewo binarne.* W przypadku istnienia elementów o wielu poprzednikach i wielu następnikach mówimy o *sieci.*

3. CHARAKTERYSTYKA WYBRANYCH METOD DOSTĘPU DO DANYCH

Po krótkim scharakteryzowaniu najczęściej występujących struktur danych przejdziemy do analizy efektywności różnych metod dostępu do zbiorów.

Organizacja danych w pamięci komputera jest ściśle związana z przyjętą metodą przeszukiwania, zaś wybór metody jest zależny od sposobu przetwarzania danego zbioru. Jeśli na przykład przetwarzanie polega na częstym przeglądaniu całego zbioru wystarcza sekwencyjna metoda jego przeszukiwania i organizacja liniowa, gdy wyszukuje się często wybrane, pojedyncze rekordy należy rozpatrzyć metodę indeksową, metodę binarnego drzewa poszukiwań lub metodę funkcji mieszającej. Pociąga to za sobą odpowiednią organizację zbiorów danych.

W celu porównania omawianych metod zostanie przyjęty pewien parametr porównawczy, będzie nim średnia liczba porównań jakich należy dokonać, by znaleźć dowolny rekord, lub inaczej średni czas przeszukiwania. Parametr ten wyraża się wzorem:

$$\mathbf{L}= \sum_{i=1}^{n} c_i p_i$$

gdzie :

L - średni czas przeszukiwania ;

$\mathbf{c_i}$ - liczba prób wykonanych w celu odnalezienia *i*-tego rekordu;

$\mathbf{p_i}$ - prawdopodobieństwo odwołania do *i*-tego rekordu.

Zakładając, że prawdopodobieństwo odwołania do każdego rekordu jest takie same ($\mathbf{p_i=p=1/n=const}$) powyższy wzór przyjmuje postać:

$$\mathbf{L}= p \sum_{i=1}^{n} c_i$$

3.1. Przeszukiwanie sekwencyjne

Jak już stwierdzono wcześniej w przypadku częstego przeglądania całego zbioru stosuje się sekwencyjną metodę przeszukiwania i liniową organizację danych. Algorytm przeszukiwania sekwencyjnego sprowadza się do kolejnego testowania rekordów i sprawdzania czy znaleziono rekord, którego klucz jest identyczny z zadanym. Średni czas przeszukiwania w tej metodzie wynosi:

$$\mathbf{L}= p \sum_{i=1}^{n} c_i$$

przy czym:

$\mathbf{c_i= i}$

$\mathbf{p=1/n}$

stąd otrzymujemy:

$$\mathbf{L}= \frac{1}{n} \sum_{i=1}^{n} i$$

a więc ostatecznie:

$$\mathbf{L}= \frac{n+1}{2}$$

3.2. Metoda binarnego drzewa poszukiwań

Jeżeli drzewo jest tak zorganizowane, że dla każdego węzła t_i wszystkie klucze z lewego poddrzewa węzła t_i są mniejsze od kluczy węzła t_i, a klucze z prawego poddrzewa są od niego większe, to takie drzewo jest nazwane *drzewem poszukiwań*.

W drzewie takim można znaleźć określony klucz posuwając się wzdłuż drogi poszukiwania, począwszy od korzenia i przechodząc do lewego lub prawego poddrzewa danego węzła w zależności tylko od wartości klucza w tym węźle. Postępując w sposób rekurencyjny (schodząc w dół drzewa), za każdym razem odrzucamy (mniej więcej) połowę elementów (z tej „niedobrej" połówki drzewa). Jest to niewątpliwa oszczędność czasu w stosunku do list liniowych i przeszukiwania sekwencyjnego.

Każde drzewo binarne może po odpowiednim rozmieszczeniu elementów, identyfikowanych za pomocą kluczy, stać się binarnym drzewem poszukiwań, dotyczy to również drzew wyważonych. Drzewo jest dokładnie wyważone jeżeli dla każdego węzła liczby węzłów w jego prawym i lewym poddrzewie różnią się co najwyżej o jeden. Średnia liczba porównań koniecznych do zlokalizowania klucza w drzewie dokładnie wyważonym o *n* węz-łach wynosi w przybliżeniu $h=log(n)-1$. Jednak nie każde drzewo poszukiwań jest wyważone. Obliczymy zatem średni czas poszukiwania dla dowolnie ukształtowanego drzewa. Zakładając, że każdy klucz może stać się korzeniem, prawdopodobieństwo tego, że *i*-ty klucz jest korzeniem wynosi $1/n$ (*n*- liczba rekordów w zbiorze). Lewe poddrzewo zawiera wówczas $i-1$ węzłów, prawe $n-i$.

Przyjmując, że:

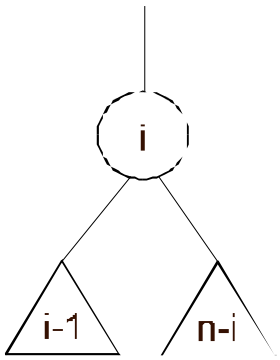
$\mathbf{a_{i-1}}$ - oznacza średnią długość drogi w lewym poddrzewie

$\mathbf{a_{n-i}}$ - średnią długość drogi w prawym poddrzewie

oraz, że wszystkie permutacje z pozostałych $n-1$ kluczy są jednakowo możliwe. Średnia długość drogi w drzewie o *n* węzłach jest sumą iloczynów poziomu każdego węzła i prawdopodobieństwa dostępu do niego:

$$\mathbf{a}_n = \frac{1}{n}\sum_{i=1}^{n} p_i$$

gdzie p_i jest długością drogi dla węzła *i*.



W przedstawionym powyżej drzewie węzły można podzielić na trzy klasy:

1. $i-1$ węzłów w lewym poddrzewie ma średnią długość drogi $a_{i-1}+1$;
2. korzeń ma długość drogi równą 1;
3. $n-i$ węzłów w prawym poddrzewie ma średnią długość drogi $a_{n-i}+1$.

Tak więc, poprzedni wzór można przedstawić jako sumę trzech składników:

$$a_n^{(i)} = (a_{i-1}+1)\frac{i-1}{n} + \frac{1}{n} + (a_{n-i}+1)\frac{n-i}{n}$$

Po odpowiednich przekształceniach (wykorzystując funkcję harmoniczną) otrzymujemy:

$$\mathbf{a_n} \approx 2(\ln(n)+\gamma) - 3 = 2\ln(n) - c$$

gdzie

n - liczba kluczy,

γ - stała Eulera $\cong$ 0.577

Ponieważ średnia długość drogi w drzewie dokładnie wyważonym wynosi w przybliżeniu $a'_n = \log(n)-1$ to otrzymujemy:

$$\lim_{n\to\infty} \frac{a_n}{a'_n} = \frac{2\ln(n)}{\log(n)} = 2\ln 2 \approx 1.386$$

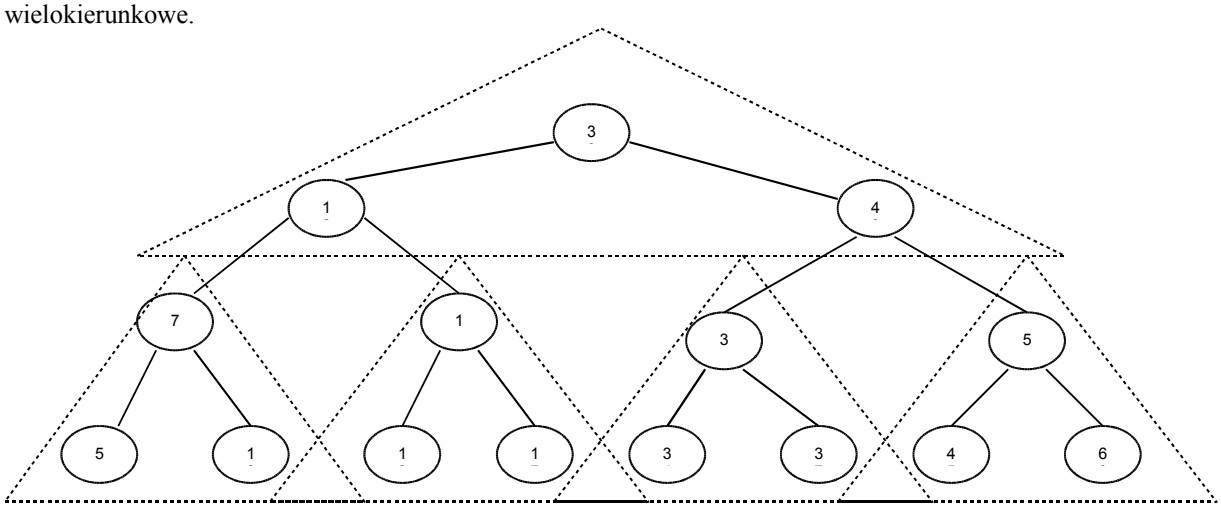
Powyższy wynik mówi o tym, że dzięki pracy włożonej w skonstruowanie drzewa dokładnie wyważonego można spodziewać się średniego skrócenia drogi poszukiwania o około 39%. Należy podkreślić słowo „średnio",

gdyż zysk ten może być niewspółmiernie większy, w niekorzystnym przypadku, w którym drzewo całkowicie degeneruje się do listy. Przypadek taki ma miejsce, gdy strukturę drzewiastą wykorzystamy dla uporządkowanych (w jakikolwiek sposób) danych. Przykładowo jeśli do drzewa będą wstawiane po kolei elementy: 1,2,3,4,5,6,7,8,9,10, wówczas struktura stanie się nie drzewem, lecz listą liniową, bo za każdym razem nowy element (jako największy ze wszystkich), będzie podczepiany do prawego następnika elementu wysuniętego najbardziej na prawo.

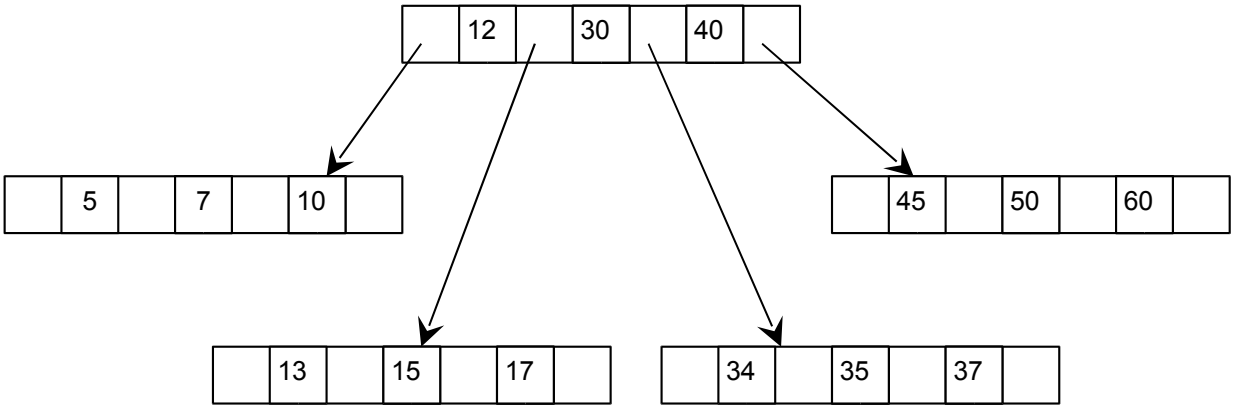
3.3. Organizacja dostępu za pomocą B-drzew

Metoda drzew binarnych stosowana jest przede wszystkim wtedy, gdy przeszukiwany zbiór znajduje się całkowicie w pamięci operacyjnej. Gdy zbiór rozmieszczony jest w pamięci zewnętrznej zasadnicze znaczenie ma liczba transmisji z pamięci zewnętrznej do pamięci operacyjnej. Operacja ta wymaga około 10^4 razy więcej czasu niż operacja porównania.

Przyjmijmy, że węzły drzewa są przechowywane w pamięci pomocniczej (np. dyskowej). Użycie drzewa binarnego dla, powiedzmy, miliona obiektów wymaga średnio około $\log_2 10^6 \approx 20$ kroków szukania. Ponieważ w każdym kroku wymaga się dostępu do dysku (z nieodłącznym opóźnieniem czasowym), to znacznie bardziej pożądana byłaby taka organizacja pamięci, która zmniejszyłaby liczbę odwołań. Idealne rozwiązanie tego problemu stanowi drzewo wielokierunkowe. Wraz z dostępem do pojedynczego obiektu w pamięci pomocniczej możemy pobrać bez dodatkowych kosztów całą grupę danych. Sugeruje to podzielenie drzewa na poddrzewa stanowiące jednostki o jednoczesnym dostępie. Te poddrzewa nazwiemy *stronami*. Na rysunkach pokazano drzewo binarne podzielone na strony zawierające po 3 węzły oraz odpowiadające mu drzewo wielokierunkowe.



Binarne drzewo podzielone na strony po 3 elementy



Drzewo wielodrogowe utworzone w wyniku pogrupowania wierzchołków drzewa z poprzedniego rysunku

Idea omówiona w powyższym przykładzie doprowadziła do zaproponowania koncepcji tzw. B-drzew. Są one wykorzystywane do wyszukiwania rekordów w bazach danych. Drzewo nazywamy B-drzewem klasy *t(h,m)* jeżeli jest ono drzewem pustym ($h = 0$) lub spełnione są następujące warunki:

1. Wszystkie drogi prowadzące z korzenia drzewa do liści są jednakowej długości równej h, liczbę h nazywamy wysokością drzewa
2. Każdy wierzchołek, z wyjątkiem korzenia i liści, ma co najmniej $m + 1$ potomków, korzeń jest albo liściem albo ma co najmniej 2 potomków
3. Każdy wierzchołek ma co najwyżej $2m + 1$ potomków.

Z maksymalnym wypełnieniem B-drzewa klasy *t(h,m)* mamy do czynienia wtedy, gdy w każdym jego wierzchołku zapamiętanych jest $2m$ elementów, a z minimalnym – gdy w wierzchołkach pośrednich i w liściach zapamiętanych jest m elementów a w korzeniu zapamiętany jest 1 element. Przy tych 2 skrajnych możliwościach wypełnienia, przy ustalonej liczbie N elementów indeksu, otrzymujemy następujące ograniczenie liczby h poziomów w indeksie zorganizowanym według struktury B-drzewa klasy *t(h,m)*:

$$\log_{2m+1}(N+1) \le h \le 1 + \log_{m+1}\frac{N+1}{2}$$

3.3.1. Odszukiwanie w B-drzewie

Przy znanej wartości klucza x problem odszukania sprowadza się do znalezienia strony na której znajduje się element o kluczu równym x. Poszukiwana strona może być korzeniem, wierzchołkiem pośrednim bądź też liściem. Wyszukiwanie realizowane jest zgodnie z poniższym algorytmem.

1. Pod p podstaw identyfikator strony korzenia.
2. Czy p jest wartością niezerową ?
   - Tak, przejdź do kroku 3
   - Nie, koniec algorytmu, brak poszukiwanego elementu w indeksie
3. Sprowadź stronę wskazywaną przez p.
4. Czy x jest mniejsze od najmniejszego klucza na stronie p ?
   - Tak, pod p podstaw skrajnie lewy wskaźnik strony potomka, przejdź do kroku 2
   - Nie, przejdź do kroku 5
5. Czy na stronie p znajduje się klucz o wartości x ?
   - Tak, koniec algorytmu, znaleziono szukany element
   - Nie, przejdź do kroku 6
6. Czy na stronie p znajduje się klucz o wartości większej od x ?
   - Tak, pod p podstaw najbardziej prawą stronę potomka zawierającą wartości kluczy mniejsze bądź równe najmniejszemu kluczowi o wartości większej od x z aktualnej strony i przejdź do kroku 2
   - Nie, pod p podstaw skrajnie prawą stronę potomka, przejdź do kroku 2

Ponieważ w trakcie wyszukiwania liczba sprowadzonych stron jest co najwyżej równa wysokości B-drzewa, zatem złożoność czasowa powyższego algorytmu jest rzędu $O(\log_m N)$.

3.3.2. Dołączanie elementu indeksu

Dołączanie poprzedzone jest algorytmem wyszukiwania i , ma sens tylko wtedy, gdy tamten zakończył się niepomyślnie. Wówczas znana jest strona liścia, do której ma być dołączony element. Dołączenie to może być bezkolizyjne lub może spowodować przepełnienie strony (gdy na stronie zapamiętanych jest już $2m$ elementów). W pierwszym przypadku nowy element dołączany jest w ten sposób, by zachować rosnące uporządkowanie wartości klucza na stronie, natomiast w drugim przypadku stosuje się metodę kompensacji lub podziału.

Metodę kompensacji można stosować wtedy, gdy jedna ze stron sąsiadujących ze stroną z przepełnioną zawiera mniej niż $2m$ elementów. Wtedy porządkujemy elementy z obu stron z uwzględnieniem elementu dołączanego oraz elementu ojca (tj. tego elementu, dla którego wskaźniki znajdujące się po jego obu stronach wskazują strony uczestniczące w kompensacji). Element „środkowy" uporządkowanego ciągu wstawiamy w miejsce elementu ojca. Wszystkie elementy mniejsze od środkowego wstawiamy na jedną z kompensowanych stron, a wszystkie większe na drugą (zachowując uporządkowanie na stronie).

Gdy kompensacja nie jest możliwa należy zastosować metodę podziału przepełnionej strony na dwie. W tym celu porządkujemy rosnąco wszystkie elementy rozważanej strony (uwzględniamy także dołączany element) otrzymując ciąg $2m + 1$ elementów. Element środkowy tego ciągu dodajemy do strony ojca, elementy mniejsze od środkowego tworzą jedną stronę a elementy większe drugą stronę potomną. W trakcie dołączania elementu środkowego do strony ojca może także na tej stronie powstać przepełnienie (gdy strona ojca jest całkowicie zapełniona). Należy wtedy dokonać jej podziału, co może wywołać potrzebę podziału na jeszcze wyższym

poziomie i tak aż do korzenia. Konsekwencją podziału strony korzenia może być utworzenie 2 nowych stron i powiększenie wysokości drzewa.

W celu utrzymania wysokości drzewa na jak najniższym poziomie należy stosować przy dołączaniu w pierwszym rzędzie metodę kompensacji. Dopiero wtedy, gdy kompensacja jest niemożliwa dokonywać podziału.

3.3.2. Usuwanie elementu indeksu

Usuwanie poprzedzone jest algorytmem wyszukiwania i ma sens tylko wtedy, gdy poszukiwanie zakończyło się sukcesem. Wówczas zmienna s wskazuje stronę zawierającą element przeznaczony do usunięcia. Jeśli strona ta jest stroną liścia to element jest z niej usuwany. Może to jednak spowodować niedomiar, to znaczy liczba elementów na stronie może spaść poniżej wartości m. Jeśli natomiast strona nie jest liściem to w miejsce usuwanego elementu, wpisywany jest element $E_{\min}$ o najmniejszej wartości klucza z poddrzewa wskazywanego przez wskaźnik p, stojący bezpośrednio po prawej stronie usuwanego elementu. Element $E_{\min}$ jest następnie wstawiany w miejsce elementu usuwanego i usuwany ze strony liścia, co także może spowodować niedomiar.

W przypadku wystąpienia niedomiaru stosujemy jedną z metod jego likwidacji, a mianowicie metodę łączenia lub kompensacji.

Łączenie

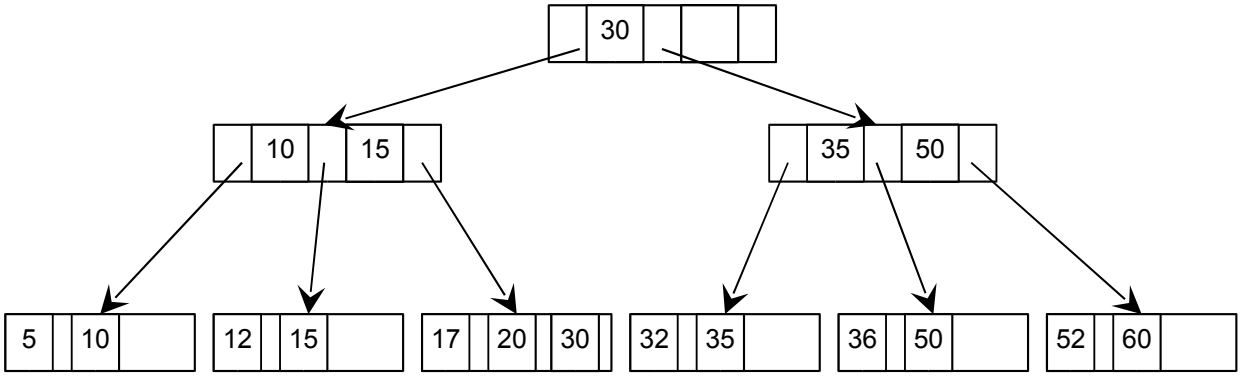
Przypuśćmy, że na stronie s wystąpił niedomiar, to znaczy zawiera ona obecnie $j < m$ elementów, a jednocześnie jedna z jej stron sąsiednich (oznaczmy ją symbolem s_1) zawiera k elementów i $j + k < 2m$. Możemy wtedy stosować metodę łączenia stron s i s_1. Wszystkie elementy ze strony s_1 są „przenoszone" na stronę s (z zachowaniem uporządkowania). Ze strony rodzica usuwany jest element rozdzielający strony s i s_1 (i także przenoszony do łączonej strony). Wyłączenie elementu ze strony rodzica może tam spowodować niedomiar. Likwidacja tego niedomiaru może z kolei spowodować niedomiar na stronie jej ojca, itd. – w skrajnym przypadku aż do korzenia B-drzewa. Stosowanie operacji łączenia jest jedyną metodą zmniejszenia wysokości drzewa.

Kompensacja

Jeśli wśród sąsiadów strony s, na której wystąpił niedomiar nie ma kandydata do połączenia ze stroną s (gdyż nie są spełnione wymagane do tego warunki), stosuje się metodą kompensacji. W metodzie tej postępujemy analogicznie jak w przypadku kompensacji przy dołączaniu z tym, że obecnie nie ma oczywiście potrzeby uwzględniania dodatkowego (dołączanego) elementu.

3.4. B*-drzewa

Na podstawie tej samej idei co B-drzewa, tworzone są tzw. B*-drzewa. Różnica między B-drzewami a B*-drzewami polega na tym, że B-drzewa są drzewami o orientacji wierzchołkowej natomiast B*-drzewa mają orientację liściową (w wierzchołkach nie będących liśćmi zawarte są jedynie wartości klucza, natomiast związane z nimi dane umieszczane są jedynie w liściach).



Przykład B*-drzewa klasy t*(3,2,2) z łącznikami między liśćmi

Każde B*-drzewo charakteryzują trzy następujące parametry:

- *h** – wysokość B*-drzewa, tj. długość dróg z korzenia do liści.
- *m** – liczba wartości klucza w korzeniu B*-drzewa i wierzchołkach nie będących liśćmi (korzeń zawiera od 1 do 2*m** wartości klucza, natomiast każdy wierzchołek pośredni zawiera od *m** do 2*m** wartości klucza)
- *m* – liczba elementów w liściach (liść zawiera od *m* do 2*m* elementów indeksu)

Rysunek na poprzedniej stronie pokazuje przykładową organizację B*-drzewa. Zauważmy, że podstawową różnicę w stosunku do B-drzew stanowi umieszczenie wszystkich danych w liściach drzewa (klucze znajdujące się w korzeniu i wierzchołkach pośrednich są powielone w liściach).

Algorytmy postępowania przy wyszukiwaniu, dołączaniu i usuwaniu z B*-drzewa niewiele różnią się od tych stosowanych do B-drzew dlatego też nie będziemy ich tu omawiać szczegółowo. Interesujące jest zastosowanie idei B*-drzew do organizacji sekwencyjnego przeglądania danych uporządkowanych według wartości klucza (a także filtrowania danych wg klucza). By to umożliwić wystarczy uzupełnić strukturę B*-drzewa o listę dwukierunkową łączącą ze sobą sąsiednie liście. Przykład takiej organizacji struktury danych przedstawiono na rysunku powyżej.

3.5. Metoda funkcji mieszającej

Kolejną dość często wykorzystywaną strukturą jest tablica. Dla tablicy nieuporządkowanej średnia liczba testów potrzebnych do odszukania elementu wynosi *n/2*, natomiast dla tablicy uporządkowanej przy stosowaniu metody dychotomicznych podziałów (dzielenie przedziałów na połowy) liczba testów redukuje się do *1/2 log₂n*. Wspomniana metoda polega na testowaniu elementu znajdującego się w połowie aktualnego przedziału. Jeśli element ten nie jest elementem szukanym, to ustala się kolejny przedział, który jest jedną z połówek ostatnio testowanego przedziału i przeprowadza się kolejny test.

Inną metodą stosowaną w odniesieniu do tablic jest metoda funkcji mieszającej. Zadaniem funkcji mieszającej jest transformacja kluczy *K* w indeksy tablicy. Podstawową trudność w użyciu transformacji kluczy stanowi fakt, że zbiór możliwych wartości kluczy jest znacznie większy od zbioru adresów (indeksów tablicy).

Mając zadany klucz *K* w pierwszym kroku operacji należy obliczyć związany z nim indeks, w drugim zaś sprawdzić, czy obiekt o kluczu *K* jest rzeczywiście pod adresem wyznaczonym przez funkcję *h*. Przypadek znalezienia pod danym adresem innego klucza niż żądany nazywamy kolizją. Natomiast algorytm wyznaczający alternatywne indeksy nazywamy algorytmem usuwającym kolizje.

Jeśli liczbę możliwych adresów oznaczymy przez *N* (pojemność bazy), to wystąpienie każdego adresu ADR_i z przedziału [*1,N*] powinno być jednakowo prawdopodobne, tzn. że dla każdego ADR_i zachodzi:

$$\mathbf{P\{h(K_i) = ADR_i\} = 1/N}$$

Obliczmy liczbę prób jaką należy wykonać aby umieścić *l+1* rekord w tablicy *n*- elementowej

$$\mathbf{L}_{l+1} = \sum_{j=1}^{l+1} c_j p_j$$

gdzie:

$\mathbf{L_{l+1}}$ - liczba prób jaką należy wykonać aby rozmieścić (odnaleźć) l+1 rekord

$\mathbf{c_j}$ - kolejno wykonywane próby

$\mathbf{p_j}$ - prawdopodobieństwo odszukania (rozmieszczenia) rekordu

$$p_1 = \frac{n-l}{n}$$

$$p_2 = \frac{l}{n}\frac{n-l}{n-1}$$

$$p_3 = \frac{l}{n}\frac{l-1}{n-1}\frac{n-l}{n-2}$$

.
.
.

$$p_j = \frac{l}{n}\frac{l-1}{n-1}\cdots\frac{l-j+2}{n-j+2}\frac{n-l}{n-j+1}$$

stąd:

$$\mathbf{L_{l+1}} = \frac{n+1}{n-l+1}$$

Ponieważ liczba prób przy rozmieszczaniu jest taka sama jak przy odszukiwaniu, to wartość powyższą można wykorzystać do obliczenia średniej liczby prób potrzebnych do znalezienia losowego klucza w tablicy:

$$L = \frac{1}{l}\sum_{k=1}^{l} l_k = \frac{n+1}{l}\sum_{k=1}^{l}\frac{1}{n-k+2} = \frac{n+1}{l}\ (H_{n+1} - H_{n-l+1})$$

gdzie: $H_n = 1+1/2+...+1/n$ jest funkcją harmoniczną, w przybliżeniu równą $\ln(n) +\gamma$,
a γ jest stałą Eulera $\cong 0.577$.
Podstawiając $\alpha = l/(n+1)$ otrzymamy

$L = -1/\alpha \ln(1-\alpha)$

α - jest w przybliżeniu iloczynem liczby zajętych i dostępnych adresów, nazwanym współczynnikiem wypełnienia: $\alpha = 0$ oznacza, że tablica jest pusta, $\alpha = n/(n+1)$ oznacza, że tablica jest pełna.

Kodowanie mieszające jest efektywniejsze pod względem liczby porównań, potrzebnych do wyszukiwania lub wstawiania, od binarnego drzewa poszukiwań, należy jednak wziąć pod uwagę fakt, że operacja porównania dla każdej z tych metod może wymagać zupełnie innego czasu. Jeśli przykładowo metoda funkcji mieszającej wymaga średnio dwóch porównań a drzewo sześciu, to nie jest to równoznaczne z tym, że haszowanie w tym przypadku jest trzykrotnie szybsze – co więcej, może się nawet okazać, że wykorzystanie metody drzewa poszukiwań jest szybsze, bo np. funkcja mieszająca ma bardzo dużą złożoność obliczeniową. Wadą kodowania mieszającego w stosunku do metod dynamicznego przydzielania pamięci jest stała wielkość tablicy, co uniemożliwia dopasowanie jej do aktualnych potrzeb. Dlatego jest tu niezbędne możliwie dokładne określenie a priori liczby obiektów w celu uniknięcia złego wykorzystania pamięci lub słabej efektywności (a czasami - przepełnienia tablicy). Nawet wtedy, gdy znana jest dokładnie liczba obiektów (przypadek bardzo rzadki) troska o dużą efektywność metody dyktuje niewielkie zwiększenie tablicy (o około 10%). Drugą ważną niedogodnością są trudności występujące przy usuwaniu elementu z tablicy. Podsumowując należy przyznać, że organizacja drzewiasta jest zalecana wtedy, gdy ilość danych jest trudna do określenia lub często się zmienia.

<u>Przykładowe funkcje mieszające i algorytmy rozwiązywania kolizji:</u>

1. - wycięcie 3 cyfr z klucza i normalizacja do zakresu pamięci
   h(k) = (cyfra klucza: 4,5 i 6-ta) *M/1000)
2. - podział, składanie i dzielenie przez największą liczbę pierwszą <=M
   h(k) = (lewe 16 bitów k + prawe 16 bitów k) mod M
3. - dzielenie przez największą liczbę pierwszą <=M
   h(k) = k mod M
4. - mieszanie multiplikatywne Fibonacciego
   h(k) = M *(0.618034*k) mod M

<u>Algorytmy rozwiązywania kolizji</u>:

1. - sondowanie liniowe z krokiem 1
   $a_i = (a_{i-1} + 1) \bmod M$
2. - sondowanie liniowe z krokiem 7
   $a_i = (a_{i-1} + 7) \bmod M$
3. - podwójne mieszanie zależne (sondowanie z krokiem zależnym od wartości h(k))
   $a_i = [a_{i-1} + h_2(a_0)] \bmod M$
   gdzie krok sondowania:
   $$h_2(a_0) = \begin{cases} 1 \quad gdy \ \ a_0 = 0 \\ M - a_0 \ gdy \ \ a_0 \neq 0 \end{cases}$$
4. - podwójne mieszanie niezależne (sondowanie z krokiem zależnym od k)
   $a_i = [a_{i-1} + h_2(k)] \bmod M$
   gdzie krok sondowania:
   $h_2(k) = 1 + [k \bmod (M-2)]$

Gdzie:

k – klucz
M – rozmiar tablicy
a_i – i-ta próba wyznaczenia adresu

4. ZADANIA DO WYKONANIA

1. Przygotować zbiór z danymi testowymi.

2. Przeprowadzić eksperymenty z metodą przeszukiwania sekwencyjnego dla danych przypadkowych, dla danych posortowanych. Porównać wyniki. W wyniku powinny się znaleźć tablica z danymi, liczba przeprowadzonych prób wyszukania, średnia liczba testów (eksperymentalna), minimalna i maksymalna liczba testów.
3. Te same testy przeprowadzić dla wyszukiwania metodą podziałów dychotomicznych. Porównać uzyskane wyniki.
4. Przeprowadzić eksperymenty z drzewami binarnymi. Porównać uzyskane wyniki dla „zwykłych" drzew (bez wyważenia) i drzew dokładnie wyważonych. Czy średnia liczba porównań w obu przypadkach różni się znacząco ?
5. Określić przykładową kolejność wprowadzania do drzewa 31 liczb tak by uzyskać dokładnie wyważone drzewo binarne.
6. Przeprowadzić eksperymenty dotyczące wyszukiwania w B-drzewach, dla różnych wielkości strony. Określić optymalny ze względu na czas wyszukiwania rozmiar strony.
7. Obliczyć minimalną i maksymalną liczbę elementów jakie można umieścić w B-drzewie klasy $t(h,m)$.
8. Przeprowadzić eksperymenty dotyczące wyszukiwania w B*-drzewach, dla różnych wielkości strony. Określić optymalny ze względu na czas wyszukiwania rozmiar strony.
9. Porównać wyniki uzyskane dla B-drzew i B*-drzew.
10. Przeprowadzić eksperymenty dotyczące wstawiania danych do tablicy mieszającej. Sprawdzić wpływ rozmiaru tablicy, wybranej funkcji mieszającej i funkcji rozwiązywania kolizji na liczbę kolizji występujących podczas wstawiania. Porównać uzyskane wyniki z przewidywaniami teoretycznymi.