Wykład 9 - struktury rekurencyjne uporządkowane

1. **Zstępujące drzewa 2-3-4**
2. **Drzewa wyważone czerwono-czarne**
3. **B-drzewa - - wyszukiwanie zewnętrzne**

1. Zstępujące drzewa 2-3-4

(wg R.Sedgewick: Algorytmy w C++):

Definicja 1

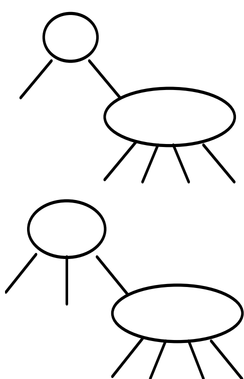
Drzewo poszukiwań *2-3-4* to drzewo, które albo jest puste, albo zawiera trzy rodzaje węzłów:

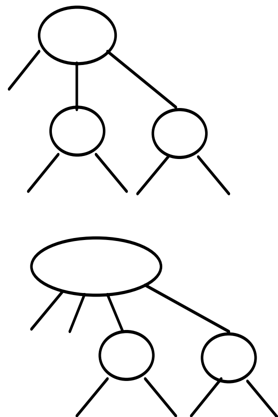
- *2- węzły* z jednym kluczem, lewym łączem do drzewa z mniejszymi kluczami i prawym łączem do drzewa z większymi kluczami
- *3-węzły* z dwoma kluczami, lewym łączem do drzewa z mniejszymi kluczami, środkowym łączem do drzewa z kluczami o wartościach kluczy zawartych pomiędzy wartościami kluczy węzła i prawym łączem do drzewa z większymi kluczami
- *4-węzły* z trzema kluczami i czterema łączami do drzew z wartościami kluczy zdefiniowanymi przez zakresy zawartych w węźle kluczy.

Definicja 2: Zrównoważone drzewo poszukiwań *2-3-4* to drzewo poszukiwań *2-3-4* z wszystkimi łączami do pustych drzew znajdującymi się w takiej samej odległości od korzenia.

Twierdzenie 1: Wyszukiwanie w *n-węzłowych* drzewach *2-3-4* odwiedza najwyżej *lgn +1* węzłów

Twierdzenie 2: Wstawianie do *n-węzłowego* drzewa *2-3-4* wymaga w najgorszym przypadku mniej niż *lgn+1* podziałów węzła i wydaje się, że wymaga średnio mniej niż jednego podziału węzła.

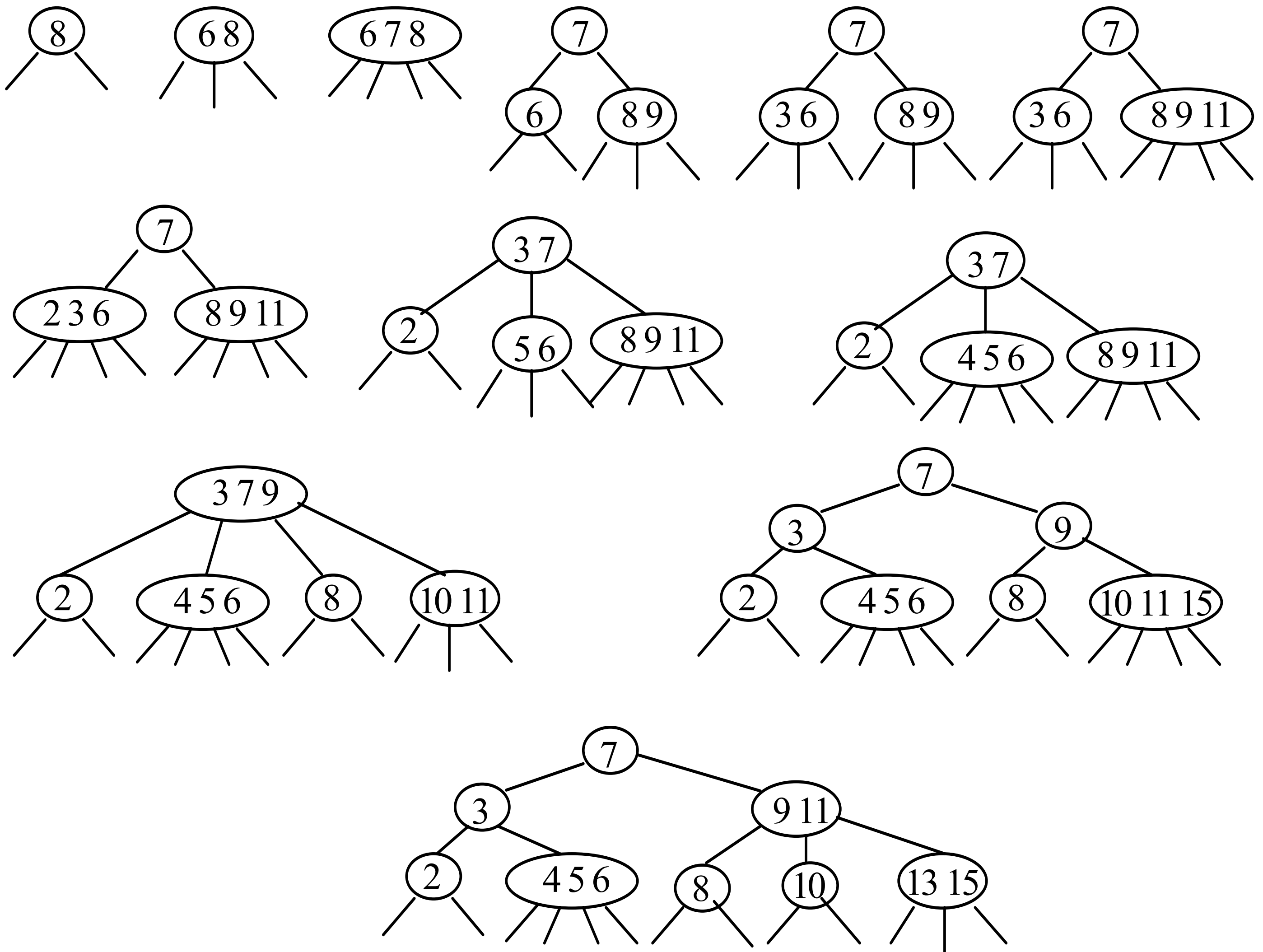




Podział *4-węzła* jest możliwy, jeśli nie jest on potomkiem *4-węzła*:

- przekazanie środkowego klucza do przodka i zamiana na dwa *2-węzły*
- zamiana przodka na węzeł o 1 rząd wyższy (*2-węzeł* na *3-węzeł* oraz *3-węzeł* na *4-węzeł*) z przyłączonymi dwoma *2-węzłami*

Przykład 1: Wstaw następujący ciąg: 8, 6, 7, 9, 3, 11, 2, 5, 4, 10, 15, 13 do drzewa 2-3-4



Algorytm wstawiania:

1. Wyszukaj miejsce wstawienia w węźle typu liść
2. Jeśli podczas wyszukiwania napotkasz na *4-węzeł*, podziel go na dwa *2-węzły* przekazując do ojca klucz środkowy oraz zamień łącza (bez zmiany dolnych węzłów oraz węzłów powyżej ojca)
3. Po osiągnięciu dołu drzewa wstaw nowy węzeł bezpośrednio przez przekształcenie albo *2-węzła* w *3-węzeł*, albo *3-węzła* w *4-węzeł*
4. Jeśli korzeń drzewa staje się *4-węzłem*, rozdziel go na trójkąt złożony z trzech *2-węzłów*. Jest to jedyny przypadek wzrostu drzewa w górę o jeden poziom

2. Drzewa wyważone czerwono-czarne
(wg R.Sedgewick: Algorytmy w C++):

Drzewo *czerwono-czarne* stanowi wygodną reprezentację abstrakcyjną *drzew 2-3-4*. Polega ona na przedstawieniu *drzew 2-3-4* jako standardowych drzew poszukiwań binarnych posiadających w każdym węźle dodatkową informację do zakodowania *3-węzłów* i *4-węzłow*.

Występują dwa typy łącz:

- ***czerwone***, które wiążą razem poddrzewa binarne zawierające *3-węzły* i *4-węzły*
- ***czarne***, które wiążą razem *drzewo 2-3-4*.

Twierdzenie 3: Wyszukiwanie w drzewie *czerwono-czarnym* z *n* węzłami wymaga mniej niż *2lgn +2* porównań.

Twierdzenie 4: Wyszukiwanie w drzewie *czerwono-czarnym* z *n* węzłami, zbudowanymi z losowych kluczy, używa średnio około *1,002 lgn* porównań.

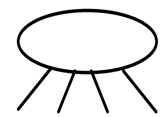
Definicja 3: *Czerwono-czarne* drzewo poszukiwań binarnych jest drzewem poszukiwań binarnych, w którym każdy węzeł jest oznaczony jako czerwony albo czarny z dodatkowym ograniczeniem, że na żadnej ścieżce od zewnętrznego łącza do korzenia *nie mogą wystąpić dwa kolejne czerwone węzły*.

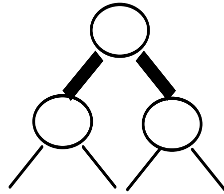
Definicja 4: Zrównoważone *czerwono-czarne* drzewo poszukiwań binarnych jest czerwono-czarnym drzewem poszukiwań binarnych, w którym wszystkie ścieżki od zewnętrznego łącza do korzenia *mają tę samą liczbę czarnych węzłów*.

Algorytm tworzenia drzewa czerwono-czarnego:

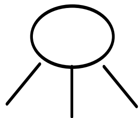
1. Wyszukaj w drzewie binarnym miejsce do wstawienia nowej danej jako liść
2. Podczas wyszukiwania każdy *4-węzeł* rozdziel na dwa *2-węzły*, przełączając bajt koloru we wszystkich trzech węzłach.
3. Po osiągnięciu dołu drzewa utwórz nowy czerwony węzeł dla wstawianego elementu.
4. Podczas powrotu w górę drzewa (po wywołaniu rekurencyjnym) sprawdź, czy należy wykonać rotację:
   4.1. Jeśli ścieżka wyszukiwania ma dwa czerwone węzły z tą samą orientacją (dwa przejścia w dół, albo na lewo, albo na prawo), wykonaj pojedynczą rotację od górnego węzła, następnie przełącz bit koloru, aby zrobić *4-węzeł*
   4.2. Jeśli ścieżka wyszukiwania ma dwa czerwone łącza z różną orientacją (dwa przejścia w dół, albo na lewo i na prawo, albo na prawo i na lewo), wykonaj pojedynczą rotację od dolnego węzła, przechodząc do przypadku 4.1 w następnym kroku w górę drzewa.

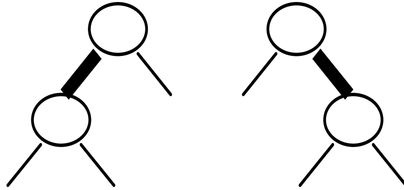
Przedstawianie i przekształcanie 3-węzłów i 4-węzłów jako potomków 3-węzła za pomocą zrównoważonych poddrzew binarnych

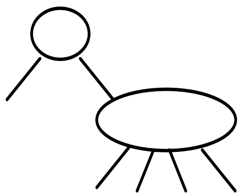


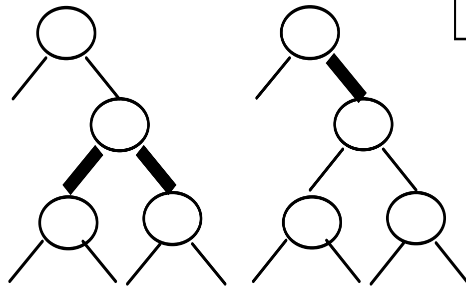


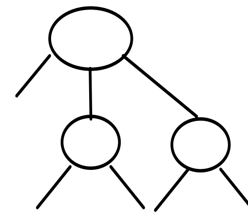
```
typedef ROsobaC*
POsobaC;
struct ROsobaC
 { OSOBA Dane;
   int Czerwony;
   PosobaC Lewy, Prawy;
 };
```

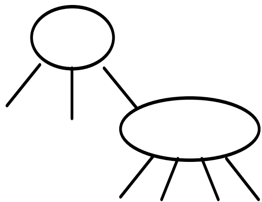


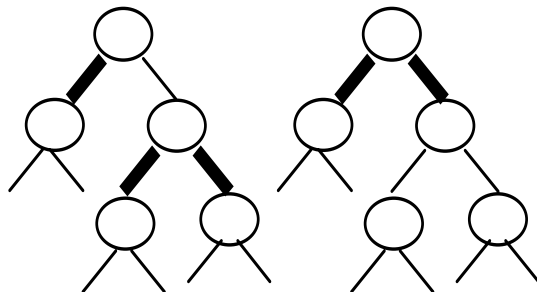


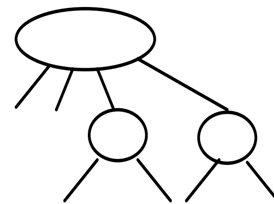


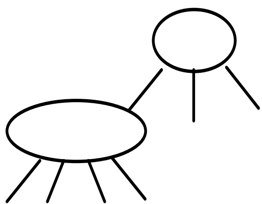


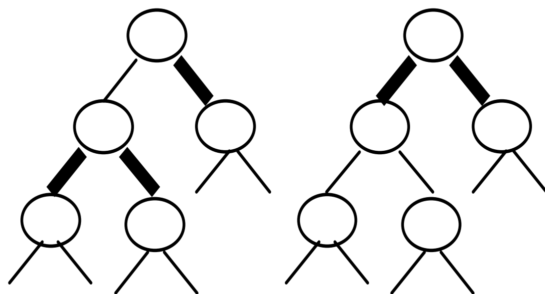


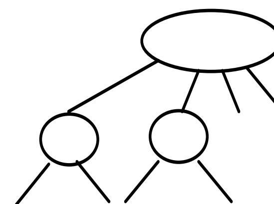


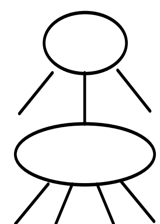


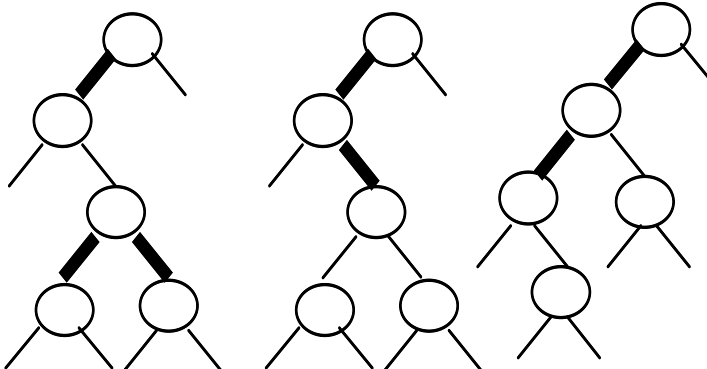


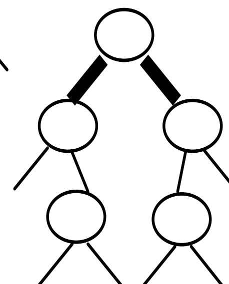


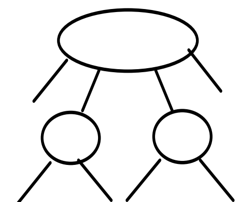




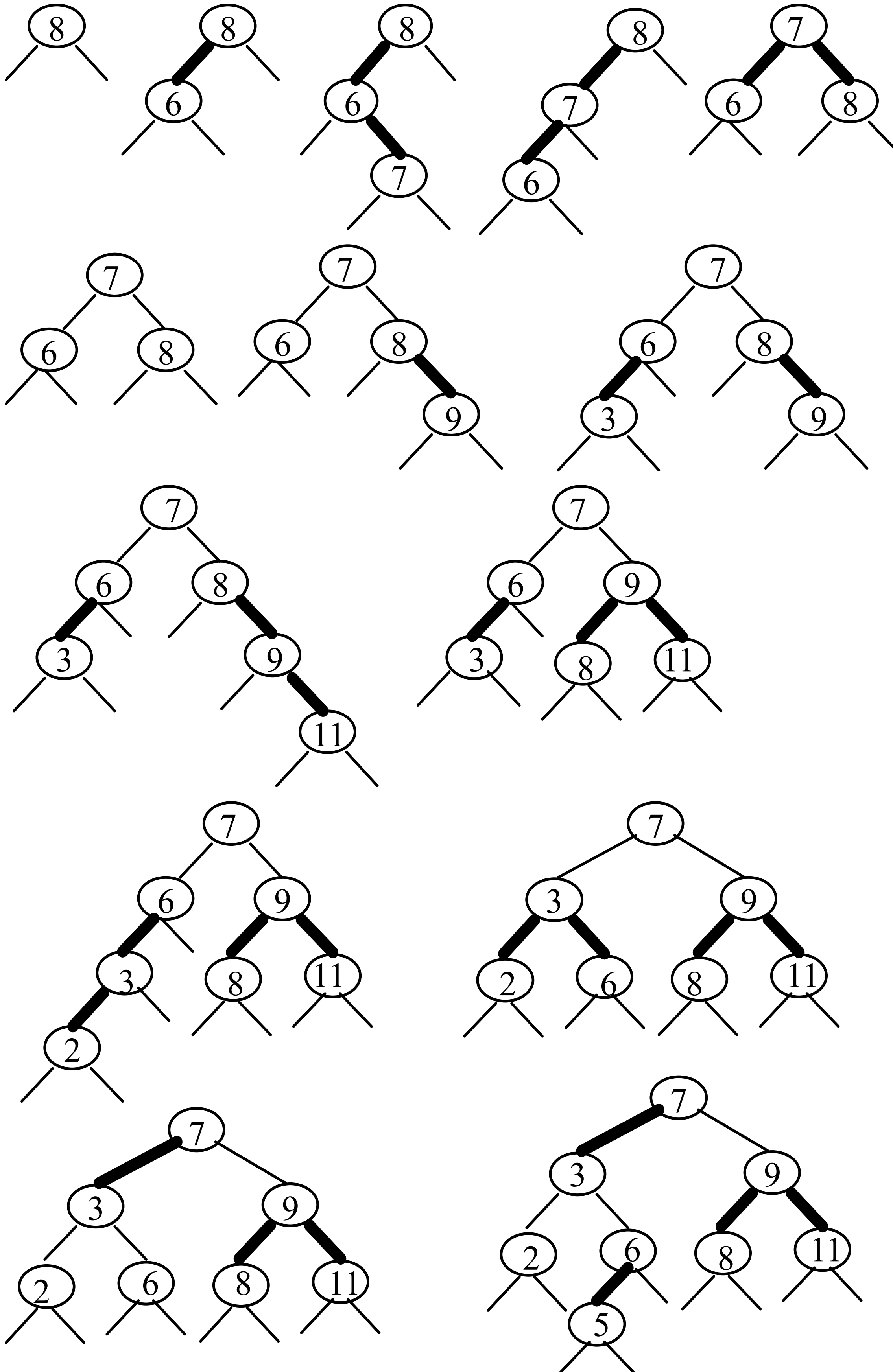


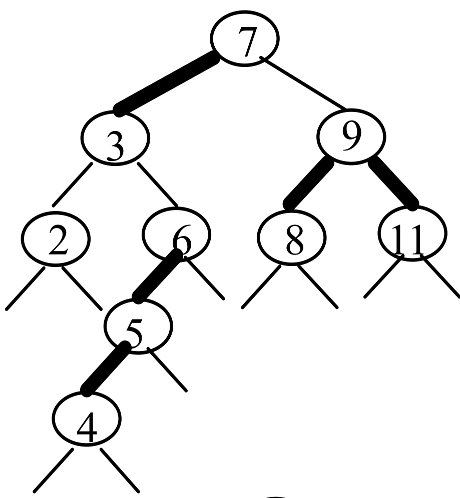


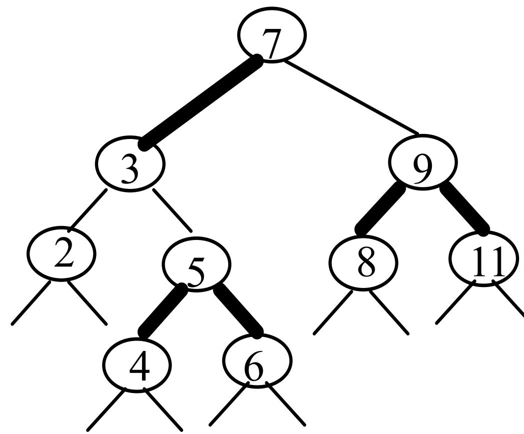


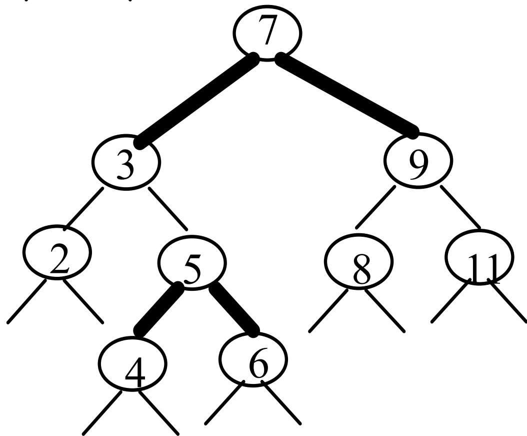


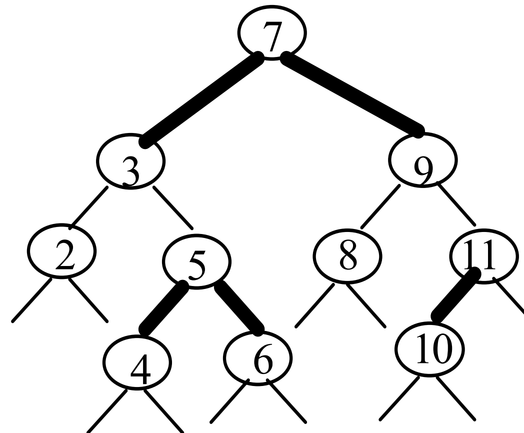
Przykład 2 - Dla ciągu z przykładu 1 (8, 6, 7, 9, 3, 11, 2, 5, 4, 10, 15, 13) wykonaj równoważne drzewo czerwono-czarne

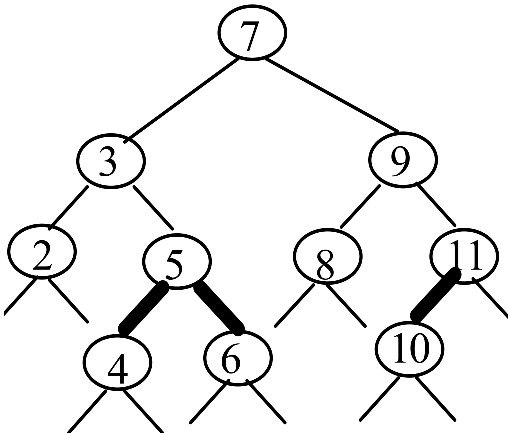


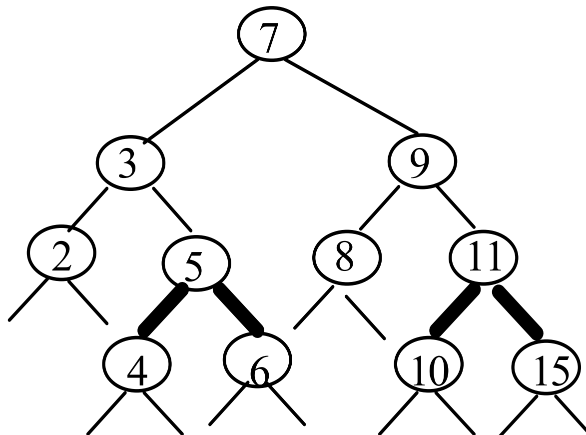

7
3
9
2
6
8
11
5
4

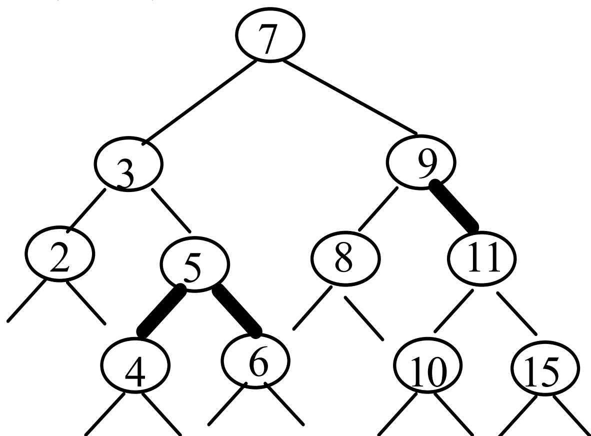

7
3
9
2
5
8
11
4
6

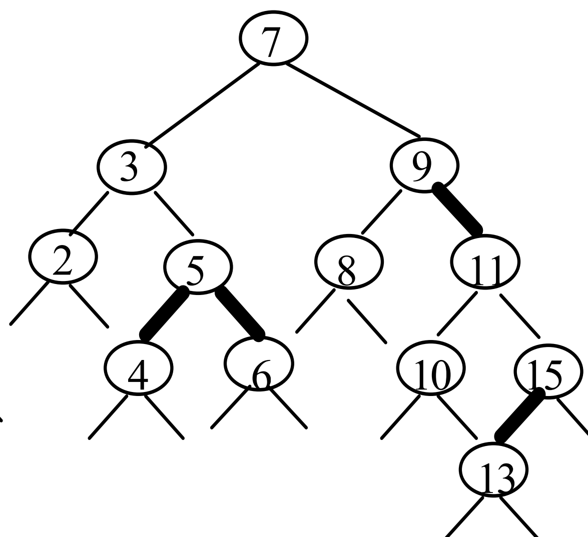

7
3
9
2
5
8
11
4
6


7
3
9
2
5
8
11
4
6
10


7
3
9
2
5
8
11
4
6
10


7
3
9
2
5
8
11
4
6
10
15


7
3
9
2
5
8
11
4
6
10
15


7
3
9
2
5
8
11
4
6
10
15
13

3. B-drzewa - wyszukiwanie zewnętrzne

(wg R.Sedgewick: Algorytmy w C++):

Wyszukiwanie zewnętrzne ma ogromne znaczenie praktyczne w celu uzyskania dostępu do elementów w bardzo dużych plikach.

Szczególnie ważne są takie algorytmy, które stanowią abstrakcyjny model efektywnego rozwiązania problemu wyszukiwania:

- dla urządzeń dyskowych
- lub w środowiskach z adresowaną ogromną pamięcią wirtualną.

Założenia algorytmów:

- dostęp sekwencyjny jest mniej kosztowny niż niesekwencyjny, stąd stosowanie abstrakcyjnego *modelu strony* zachowującego charakterystykę urządzeń zewnętrznych (blok w systemie plików; strona w systemie pamięci wirtualnej)
- na stronie nie trzymamy rzeczywistych danych, lecz *referencje* do nich (adresy stron lub interefejs do bazy danych) oraz kopie kluczy (aby uniknąć problemów związanych z integralnością danych). W rzeczywistych implementacjach należy referencje zastąpić właściwymi mechanizmami dostępu do danych (np. mechanizm dostępu w środowisku pamięci wirtualnej)
- unikanie uzależniania się od własności fizycznych określonych urządzeń fizycznych w celu uzyskania szczytowej wydajności, lecz dążenie do uzyskania dobrej wydajności na wielu różnych komputerach
- algorytmy *Szukaj, Wstaw* określają dobrą wydajność projektowanym systemom

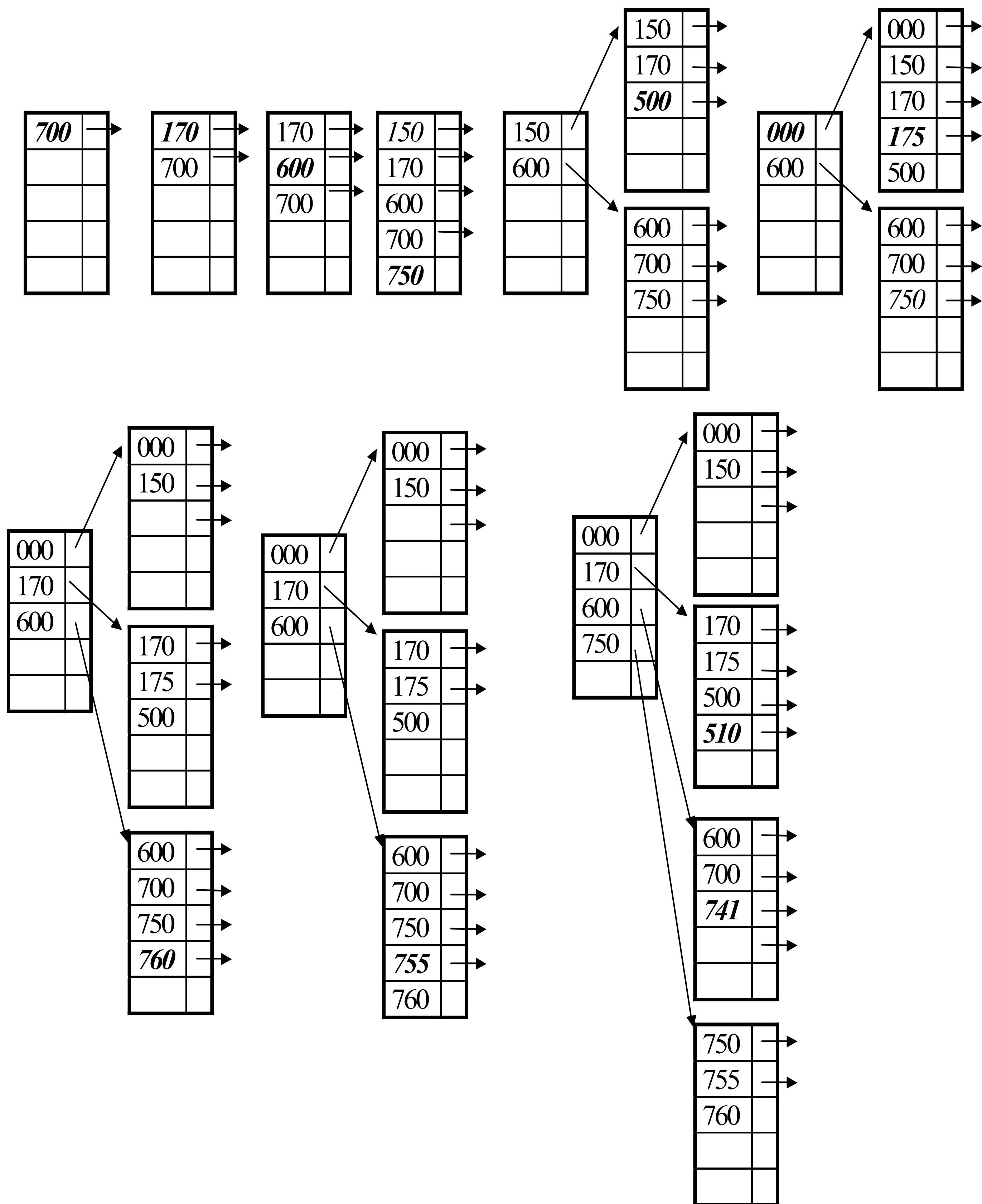
Definicja 5: Strona stanowi zwarty blok danych. Sondowanie jest pierwszym dostępem do strony.

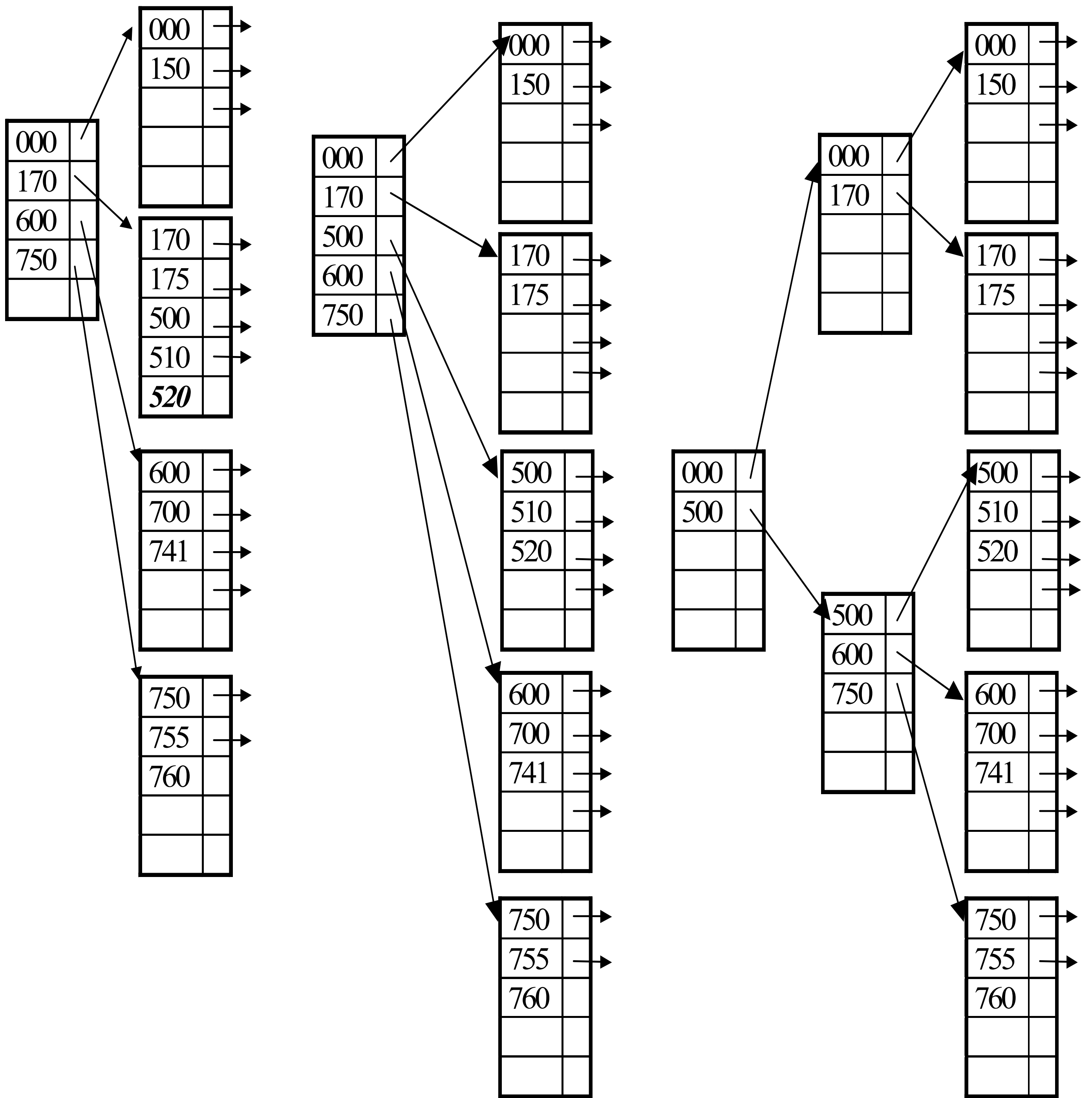
Definicja 6: B-drzewo *rzędu m* to drzewo, które jest puste albo zawiera *k-węzły z k-1 kluczami* i *k łączami* do drzew reprezentujących każdy z *k* przedziałów ograniczanych przez klucze i posiada następujące własności strukturalne: dla korzenia *k* musi zawierać się pomiędzy *2* i *m*, zaś dla każdego innego węzła pomiędzy *m/2* i *m* (tak aby zmieścił się na stronie); zaś wszystkie łącza do pustych drzew muszą się znajdować się w tej samej odległości od korzenia.

Rozpatrywany model B-drzewa posiada następujące własności:

- uogólnia *drzewa 2-3-4* do drzew zawierających od *m/2* do *m* węzłów
- reprezentuje wielokierunkowe węzły jako tablice elementów i łączy
- implementuje indeks zamiast struktury wyszukiwania zawierającej elementy (węzły zewnętrzne zwierają klucze oraz referencje do elementów oraz węzły wewnętrzne kopie kluczy z referencjami do stron)
- rozdziela strukturę od dołu ku górze
- oddziela indeks od elementów.

Przykład 3 - budowa B-drzewa dla przykładowych danych




000
170
600
750
000
150
170
175
500
510
520
600
700
741
750
755
760
000
170
500
600
750
000
150
170
175
500
510
520
600
700
741
750
755
760
000
500
000
170
500
600
750
000
150
170
175
500
510
520
600
700
741
750
755
760

Algorytm tworzenia B-drzewa

1. Wyszukaj rekurencyjnie w węzłach wewnętrznych (wysokość *dodatnia B-drzewa*) pierwszy klucz większy niż klucz wyszukiwania i wykonaj wyszukiwanie w poddrzewie wskazanym przez poprzedzające ten klucz łącze.
2. Po dojściu do węzła zewnętrznego (wysokość zerowa *B-drzewa*) sprawdź, czy jest tam element równy kluczowi lub większy od klucza i wstaw nowy element przed nim, rozsuwając elementy tablicy strony zewnętrznej
3. Jeśli po wstawieniu nastąpi wypełnienie tablicy, podziel węzeł na dwie „połówki", tworząc nowy węzeł zewnętrzny z danymi z drugiej połówki starego węzła (po wstawieniu nowego elementu). Następnie wstaw do węzła wewnętrznego (ojca) nowy element zawierający kopię pierwszego klucza i wskazanie na nowy węzeł zewnętrzny.
4. Po wstawieniu nowego elementu do węzła wewnętrznego po podziale w kroku 3, jeśli nastąpiło przepełnienie tablicy węzła, podziel w taki sam sposób węzeł wewnętrzny przemieszczając połowę większych kopii kluczy oraz wskazania na odpowiadające im węzły potomne do nowego węzła wewnętrznego. Jeśli to konieczne, rozszerz podziały na całą drogę wyszukiwania aż do korzenia.

Twierdzenie 5: Wyszukiwanie i wstawianie w *B-drzewie* rzędu m z n elementami wymagają liczby sondowań z przedziału pomiędzy $log_m n$ i $log_{m/2} n$ - dla zastosowań praktycznych jest to liczba stała.

Twierdzenie 6: Oczekiwana liczba stron *B-drzewa* rzędu m zbudowanego z n losowych elementów wynosi około $1.44\ n/m$.

Definicja węzła B-drzewa

```
#ifndef STRUKT
#define STRUKT
const int DL=10;
 struct OSOBA
   {   int Numer;
       char Nazwisko[DL];
   };
#endif
```

```
//B_drzewo – plikB_DRZEW.H

#ifndef BDRZEWO
#define BDRZEWO
#include "strukt.h"

 const int N= 4;
 struct Strona;
 typedef Strona* PStrona;

 struct  Element
  {  int  Klucz;
     OSOBA* Dane;
     PStrona Nastepna;
  };

 struct Strona
  { int  Ile;
    Element B_tab[N];
  };

 typedef  void(* zrob)(OSOBA&);

 void Inicjalizacja (PStrona& Wezel,int& Wysokosc);
 void Wstaw(PStrona& Wezel, OSOBA* N_dane, int& Wysokosc);
 PStrona WstawB(PStrona Wezel,int Wysokosc,OSOBA* N_dane);
 int Szukaj(PStrona Wezel, int Klucz, int Wysokosc, OSOBA& Dane);
 int Nowa_Strona(PStrona& Nowa);
 void Dla_kazdego(PStrona Wezel, zrob funkcja, int Wysokosc);
 void Dla_jednego(PStrona& Wezel, zrob funkcja, int Klucz,int Wysokosc);
 #endif
```

```
#include "B_MDRZEW.H"
#include <string.h>
#include <conio.h>
#include <stdio.h>

int Nowa_Strona(PStrona& Nowa)
  { Nowa= new Strona;       //nowy węzeł albo liść
    if (Nowa == NULL) return 0;
    Nowa->Ile=0;                    // z zerową liczbą danych
    return 1;}

void Inicjalizacja (PStrona &Wezel,int& Wysokosc)
  { Wysokosc= 0;                 //inicjalizacja danych Bdrzewa- wysokość równa 0
    Nowa_Strona(Wezel); }

 int Szukaj(PStrona Wezel, int Klucz, int Wysokosc, OSOBA& Dane)
  { int i;        //poszukiwanie danej wg klucza z przekazanie kopii danej Dane z liścia
    if (Wysokosc == 0)
     { for (i=0; i<Wezel->Ile;i++)
         if (Klucz==Wezel->B_tab[i].Klucz)
          { Dane=*Wezel->B_tab[i].Dane; //pobranie kopii znalezionej danej z liścia
            return 1; }
     }
    else                         //poszukiwanie liścia z danymi przechodząc przez węzły z kluczami
     for ( i= 0; i< Wezel->Ile; i++)
       if (i+1== Wezel->Ile || Klucz < Wezel->B_tab[i+1].Klucz)
        return Szukaj(Wezel->B_tab[i].Nastepna, Klucz, Wysokosc-1, Dane);
   return 0;
 }

 PStrona Podzial (PStrona Wezel)
 { int i;
   PStrona Nowa=NULL;
   if  (Nowa_Strona(Nowa))                        //podział liścia lub węzła z kluczami
     { for (i=0; i< N/2; i++)
         Nowa->B_tab[i]= Wezel->B_tab[N/2+i]; //do nowego węzła połowa danych
         Wezel->Ile= N/2; Nowa->Ile= N/2; }
   return Nowa;}                     //zwrot nowego węzła z połową danych starego węzła
```

```
PStrona WstawB(PStrona Wezel,int Wysokosc,OSOBA* N_dane)
 { int i=0, j=0; Element Pom; PStrona Nowa;
   int Klucz= N_dane->Numer;
   Pom.Klucz= Klucz;
   Pom.Dane=N_dane;
   Pom.Nastepna=NULL;
   if (Wysokosc== 0)                    //wysokość =0 oznacza poziom liści w B-drzewie
    while (j < Wezel->Ile)               //poszukiwanie miejsca w liściu do wstawienia
     { if (Klucz < Wezel->B_tab[j].Klucz) break; //i przejście do wstawiania elementu z danymi
       else j++; }                          // na właściwe miejsce (również po podziale liścia)
   else
    { if (Klucz < Wezel->B_tab[0].Klucz)       //zmiana pierwszego klucza w pierwszym węźle,
      Wezel->B_tab[0].Klucz= Klucz;          // jeśli pojawił się klucz mniejszy
      while (j < Wezel->Ile)  //wstawianie klucza na właściwe miejsce, gdy wysokość B-drzewa jest
       { if (j+1== Wezel->Ile || Klucz < Wezel->B_tab[j+1].Klucz)  // większa od 0
          { Nowa=WstawB(Wezel->B_tab[j++].Nastepna, Wysokosc-1,N_dane);
            if (Nowa== NULL) return NULL;     //nie podzielono węzła
            Pom.Klucz= Nowa->B_tab[0].Klucz;
            Pom.Nastepna= Nowa;
            break; }   // przejście do wstawiania elementu z kluczem na właściwe miejsce po podziale następcy
         j++; }
       }
   for (i= Wezel->Ile; i>j;i--)
      Wezel->B_tab[i]= Wezel->B_tab[i-1];  //rozsuń elementy tablicy w miejscu j i wstaw
   Wezel->B_tab[j]= Pom;                  //nowy  element z kluczem i danymi, gdy węzeł jest liściem
   Wezel->Ile++;                          //lub tylko z kluczem, gdy węzeł  zawiera tylko klucze
   if (Wezel->Ile < N)  return NULL; //jeśli tablica nie jest zapełniona, zakończ wstawianie
   else return Podzial(Wezel); }//jeśli tablica jest pełna, podziel węzeł: liść albo węzeł z kluczami

void Wstaw(PStrona& Wezel, OSOBA* N_dane, int& Wysokosc)
 { PStrona Nowa, Pom;
   Nowa= WstawB(Wezel, Wysokosc, N_dane); //wstawia się nową daną
   if (Nowa !=NULL)                  //nastąpił podział węzła: liścia albo węzła z kluczami
    { if (! Nowa_Strona(Pom)) return;
      Pom->Ile= 2;     //tworzenie nowego węzła z kluczami jako przodka podzielonego węzła
      Pom->B_tab[0].Klucz= Wezel->B_tab[0].Klucz;
      Pom->B_tab[1].Klucz= Nowa->B_tab[0].Klucz;
      Pom->B_tab[0].Nastepna= Wezel; //referencja do starego węzła z połową danych
      Pom->B_tab[1].Nastepna=Nowa;  //referencja do nowego węzła z połową danych
      Wezel= Pom;                      //nowy węzeł
      Wysokosc++;  }
 }
```

```
void Dla_kazdego(PStrona Wezel, zrob funkcja, int Wysokosc)
 { int i, j;                //wyświetlanie zawartości drzewa
  if (Wezel !=NULL)
   { printf("\nWys: %d\n",Wysokosc);
     if (Wysokosc==0)         //wyświetlanie zawartości liści Bdrzewa wraz z danymi
       for (i= 0; i< Wezel->Ile ;i++)
           funkcja(*Wezel->B_tab[i].Dane);
    else
     { for (i= 0; i<Wezel->Ile;i++) //wyświetlanie zawartości węzłów Bdrzewa z kluczami
         printf(" Klucz: %d", Wezel->B_tab[i].Klucz);
       printf("\n");
       for (j= 0; j<Wezel->Ile;j++)
                 //przejście do następców danego węzła: albo liści albo węzłów z kluczami
          Dla_kazdego(Wezel->B_tab[j].Nastepna,funkcja, Wysokosc-1);  }
     }
  }
//wykonanie operacji na danej określonej wartością danej klucz
 void Dla_jednego(PStrona& Wezel,zrob funkcja, int klucz, int Wysokosc)
 {OSOBA Dane;
  if (Szukaj(Wezel, klucz, Wysokosc, Dane) )
   funkcja(Dane);  }
```

```
#ifndef DODATKI              //plik DODATKI.H
#define DODATKI
 const int POZ=4;
     //4. funkcje ogolnego przeznaczenia
  void Komunikat(char*);
  char Menu(const int ile, char *Polecenia[]);
#endif

#include <conio.h>
#include <stdio.h>
#include "dodatki.h"

 char Menu(const int ile, char *Polecenia[])
  { clrscr();
    for (int i=0; i<ile;i++)
      printf("\n%s",Polecenia[i]);
    return getch(); }

  void Komunikat(char* s)
   {  printf(s); getch(); }
```

```
#ifndef WE_WY        //plik WE_WY.H
#define WE_WY
#include "strukt.h"

//3. funkcje we/wy dla struktur
 void Pokaz_dane (OSOBA &Dana);
 OSOBA Dane();
 int Podaj_klucz();
#endif
```

```
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "we_wy.h"
#include "strukt.h"

  OSOBA Dane()
   {
    char bufor[DL+2];
    OSOBA Nowy;
    bufor[0]=DL;
    Nowy.Numer=Podaj_klucz();
    printf("\nnazwisko: ");
    strcpy(Nowy.Nazwisko,cgets(bufor));
    return Nowy;
   }

 int Podaj_klucz()
    { int Klucz;
      do
       { printf("\nPodaj klucz: ");
       } while (scanf("%d",&Klucz)!=1);
      return Klucz;
    }

  void Pokaz_dane(OSOBA &Dana)
   {clrscr();
    printf("\nNumer: %d\n", Dana.Numer);
    printf("Nazwisko:        %s\n", Dana.Nazwisko);
    printf("Nacisnij dowolny klawisz...\n"); getch();
   }
```

```
//B-drzewo
#include <conio.h>
#include <string.h>
#include <stdio.h>
#include "B_MDRZEW.H"
#include " DODATKI.H"
#include "WE_WY.H"
 char *Polecenia[]={ "1 : Wstawianie Bdrzewa ",   "2 : Wydruk drzewa",
                              "3 : Wyswietlenie elementu drzewa",
                              "Esc - Koniec programu"};
 void Wstaw_do_drzewa_B(PStrona& Poczatek, int& Wysokosc);
 void Dla_jednego_(PStrona &Poczatek, zrob funkcja, int Wysokosc);
 void main(void)
  { PStrona Poczatek_B; int  Wysokosc;     //dane Bdrzewa
   char Co;
   Inicjalizacja(Poczatek_B, Wysokosc);
   do
    { Co = Menu(POZ,Polecenia);
     switch(Co)
      {case '1'  : Wstaw_do_drzewa_B(Poczatek_B, Wysokosc);        break;
       case '2'  : Dla_kazdego(Poczatek_B, Pokaz_dane, Wysokosc);  break;
       case '3'  : Dla_jednego_(Poczatek_B, Pokaz_dane, Wysokosc);   break;
       case 27   :  Komunikat("\nKoniec programu");                break;
       default   :  Komunikat("\nZla opcja"); }
   } while (Co!=27);
  }

void Wstaw_do_drzewa_B(PStrona& Poczatek, int& Wysokosc)
  {OSOBA* Dana = new OSOBA;
   if (Dana==NULL)   Komunikat("\nBrak pamieci");
   else
    {*Dana= Dane();
     Wstaw(Poczatek, Dana, Wysokosc);}
   }

 void Dla_jednego_(PStrona &Poczatek, zrob funkcja, int Wysokosc)
  {  int Klucz;
     if (Poczatek==NULL)   Komunikat("\nDrzewo puste");
     else
      { Klucz=Podaj_klucz();
        Dla_jednego(Poczatek,Pokaz_dane, Klucz, Wysokosc);} }
```