Wykład 4
Wyszukiwania w tablicach posortowanych

1. Wyszukiwanie sekwencyjne w tablicy posortowanej
2. Wyszukiwanie binarne bez powtórzeń
3. Wyszukiwanie binarne z powtórzeniami

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
-3	-1	1	2	3	4	5	7	8	23	30	36	41	55	61	66	71	77	80	82	90	99

W tablicach uporządkowanych dla **n** elementów mamy:

- najgorszy przypadek przeszukań - **n**
- średnia liczba przeszukiwań trafionych i chybionych - **n/2**

Algorytm wyszukiwania sekwencyjnego - poziom konceptualny

(1) Wskaż na pierwszy element N - elementowego ciągu.
(2) Dopóki wskazany element jest mniejszy od klucza i numer wskazanego elementu nie jest większy od N , wykonuj:
(3) porównaj wskazany element z kluczem
(4) wskaż na następny element.
(5) Jeśli wskazany element jest większy od klucza, wyszukiwanie zakończono bez znalezienia elementu, w przeciwnym razie wskazany element jest równy kluczowi

Złożoność algorytmu wyszukiwania sekwencyjnego

- prawdopodobieństwa, że poszukiwany element jest w tablicy: *p* oraz, ze znaleziono większy od niego: *q*
- prawdopodobieństwa, że poszukiwany element jest na *i-tym* miejscu: *p/N,* oraz, że na *i-tym* miejscu zaniechano poszukiwań: *q/N*
- czasy wyszukiwania trafionego oraz chybionego zależny od położenia *i* szukanego elementu *T1(N)=i* oraz elementu większego: *T2(N) = i*
- średni czas trafionego wyszukiwania z prawdopodobieństwem *p*:

$$T1(N) = \sum_{i=1}^{N} i * \frac{p}{N} = (N+1) * \frac{p}{2}$$

np. dla p=1 mamy T1(N) ≈ N/2

- średni czas chybionego wyszukiwania z prawdopodobieństwem *q*:

$$T2(N) = \sum_{i=1}^{N} i * \frac{q}{N} = (N+1) * \frac{q}{2}$$

np. dla q=1 mamy T2(N) ≈ N/2

Algorytm wyszukiwania sekwencyjnego - poziom implementacji

```
#include <conio.h>
#include <stdio.h>
 typedef int element;
 const long N=12;
 int SzukS(long L, long P, element klucz, long& ktory, element T[]);
 void wyswietl(element T[], long ile);

 void main()
 {  element T[N]={1,2,3,4,5,6,7,8,9,16,18,20};
    element liczba;
    long i, ktory, ile=12;;
    clrscr();
    wyswietl(T, ile);
    do
    { printf("Podaj liczbe: ");
       scanf("%d",&liczba);
       if (SzukS(0, N-1, liczba, ktory, T))
           printf("Szukana liczba ma numer %d w tablicy.\n", ktory+1);
       else printf("Nie ma tej liczby w tablicy.\n");
       printf("Jesli koniec, nacisnij ESC-/nie, nacisnij dowolny klawisz\n");
    } while(getch()!=27);
   }

 int SzukS(long L, long P, element klucz, long& ktory, element T[])
  {  ktory=L;
     while (L<=P)
       if (T[L] < klucz) L++;
       else
       { ktory=L;   break;}
     return (T[ktory]==klucz);
  }

void wyswietl(element T[], long ile)
 { for(long i=0; i<ile; i++)
    { printf("%d \n", T[i]);
         if (i%20==0)
          {char z=getch();
            if (z=='k') return; } } }
```

2. Wyszukiwanie binarne w tablicy posortowanej bez powtórzeń

W tablicach uporządkowanych dla **n** elementów mamy:

- najgorszy przypadek przeszukań - **lg n**
- średnia liczba przeszukań - **lg n**

- najlepszy przypadek - jedno przeszukanie

Algorytm wyszukiwania binarnego bez powtórzeń - poziom konceptualny

(1) *Wskaż na element środkowy ciągu N - elementowego*

(2) Dopóki przedział badanych elementów jest niemniejszy niż 1-elementowy i nie znaleziono elementu równego kluczowi, wykonuj:

(3) jeśli wskazany element jest mniejszy od klucza, to wskaż podciąg prawy z wyłączeniem wskazanego elementu, w przeciwnym razie

(4) jeśli wskazany element jest większy od klucza, to wskaż podciąg lewy z wyłączeniem wskazanego elementu, w przeciwnym razie

(5) wskazany element jest równy kluczowi i należy zakończyć wyszukiwanie;

(6) wskaż na środkowy element w podciągu z punktu (2.1) lub (2.2)

(3) *W wyniku wyszukiwania osiągnięto sukces w przypadku (2.3) lub po osiągnięciu zbioru jednoelementowego z (2.1) lub (2.2) nie znaleziono elementu równego kluczowi.*

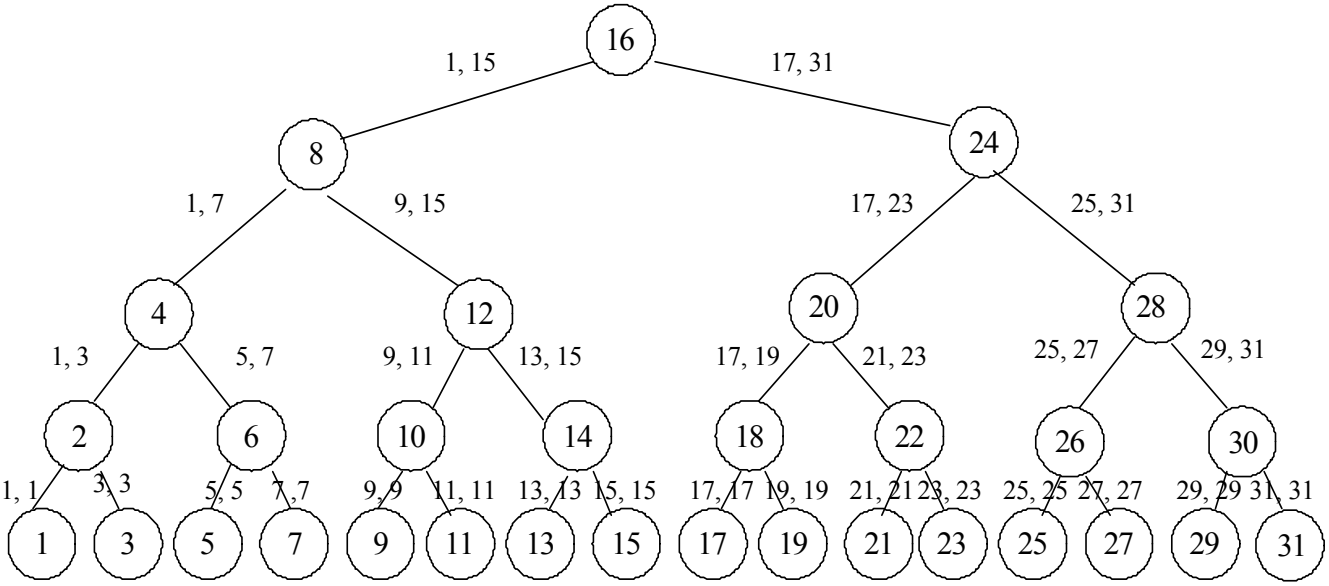
Algorytm wyszukiwania binarnego bez powtórzeń - poziom implementacji

```
#include <conio.h>
#include <stdio.h>

typedef int element;
const int N=12;
int SzukP(int L, int P, int klucz, int& ktory, element T[]);
void wyswietl(element T[], int ile);

void main()
{  element T[N]={1,2,3,4,5,6,7,8,9,16,18,20}, liczba;
   int i, ktory, ile=12;;
   clrscr();
   wyswietl(T,ile);
   do
   { printf("Podaj liczbe: ");
     scanf("%d",&liczba);
     if (SzukP(0, N-1, liczba, ktory, T))
      printf("Szukana liczba znajduje sie na miejscu %d w tablicy.", ktory);
     printf("Jesli koniec, nacisnij ESC-/nie, nacisnij dowolny klawisz");
   }while(getch()!=27);
  }
```

```
int SzukP(int L, int P, element klucz, int& ktory, element T[])
 {  int jest=0;
    while (L<=P && jest==0)
    {  ktory = (L + P) / 2;
       if (T[ktory] < klucz) L = ktory + 1;
       else
         if (T[ktory] > klucz) P = ktory - 1;
         else
           jest = 1;
    }
    return jest;
 }

void wyswietl(element T[], int ile)
 { for(long i=0; i<ile; i++)
   { printf("%d \n", T[i]);
         if (i%20==0)
          {char z=getch();
           if (z=='k') return;
          }
       }
   printf("%ld \n", ile);
 };
```

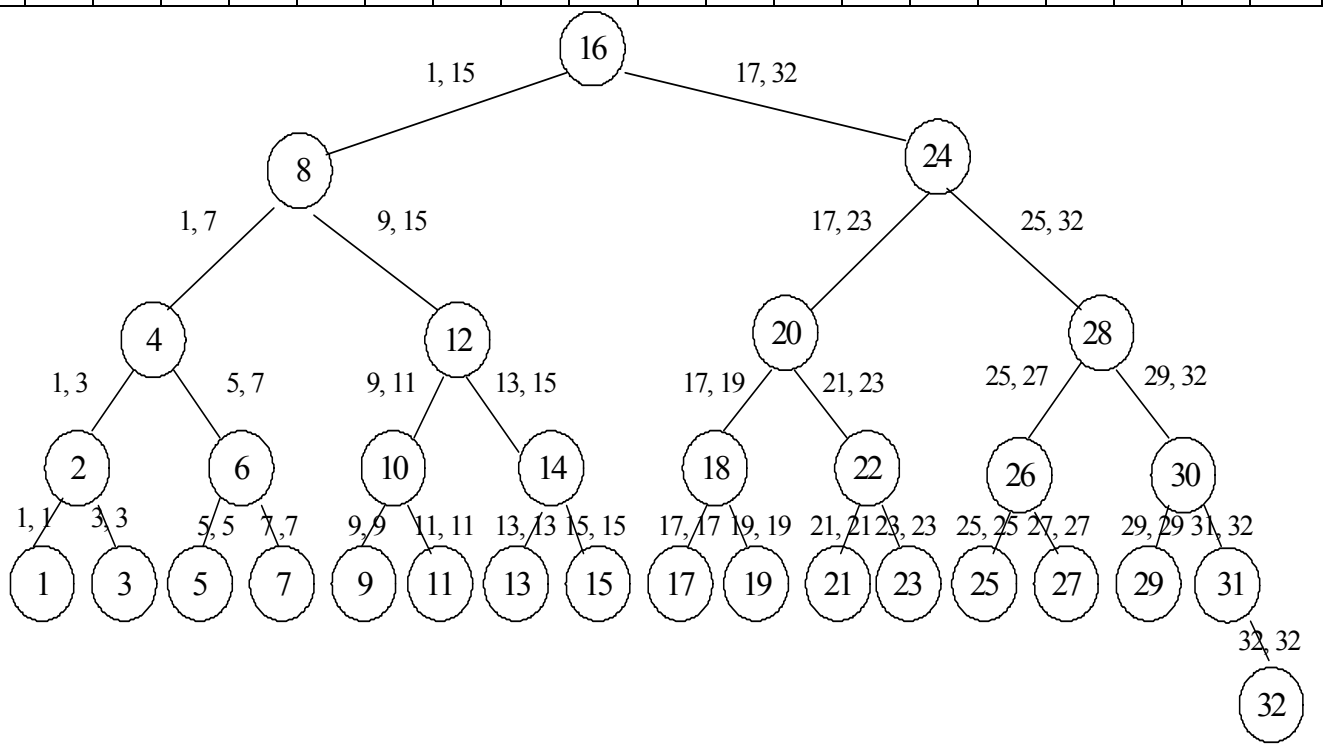
Przykład przeszukiwania binarnego bez powtórzeń - nieparzysta liczba elementów

l.p.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	-3	-1	1	2	3	4	5	7	8	23	25	28	31	34	36	39	41	42	45
l.p.	20	21	22	23	24	25	26	27	28	29	30	31							
	46	48	49	52	55	58	61	63	66	67	71	74							



Przykład przeszukiwania binarnego bez powtórzeń - parzysta liczba elementów

l.p.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	-3	-1	1	2	3	4	5	7	8	23	25	28	31	34	36	39	41	42	45
l.p.	20	21	22	23	24	25	26	27	28	29	30	31	32						
	46	48	49	52	55	58	61	63	66	67	71	74	76						



Wyszukiwanie binarne w tablicy posortowanej z powtórzeniami
Analiza wydajności algorytmów sortowania

1. Wyszukiwanie binarne w tablicy posortowanej z powtórzeniami elementów

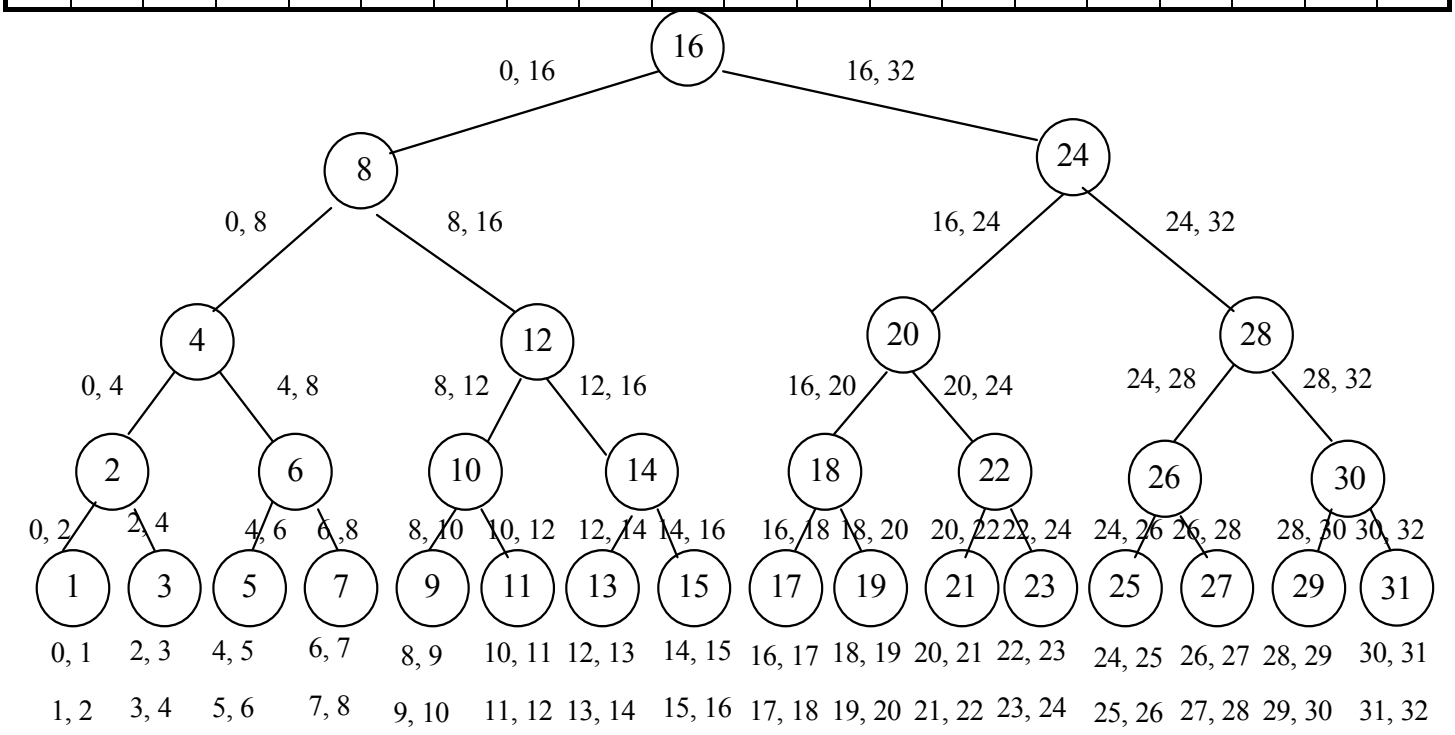
Algorytm wyszukiwania binarnego z powtórzeniami - poziom konceptualny

- *Rozszerz N - elementowy ciąg o dowolny element skrajny lewy i dowolny element skrajny prawy, gdzie dodane elementy nie należą do badanego ciągu*
- *Dopóki badany podciąg jest niemniejszy niż 3 - elementowy, wykonuj:*
  - *Wskaż na środkowy element wyznaczonego ciągu*
  - *Jeśli wskazany element jest mniejszy od klucza, wyznacz podciąg prawy z włączeniem wskazanego elementu, w przeciwnym przypadku*
  - *jeśli wskazany element jest równy lub większy od klucza, wyznacz podciąg lewy z włączeniem wskazanego elementu.*

(3) *Jeśli w wyznaczonym 2 - elementowym ciągu element prawy jest elementem dodanym lub element prawy jest różny od klucza, nie znaleziono elementu równego kluczowi-zwróć 0, w przeciwnym razie element prawy jest równy kluczowi – zwróć 1.*

Przykład przeszukiwania binarnego z powtórzeniami - nieparzysta liczba elementów

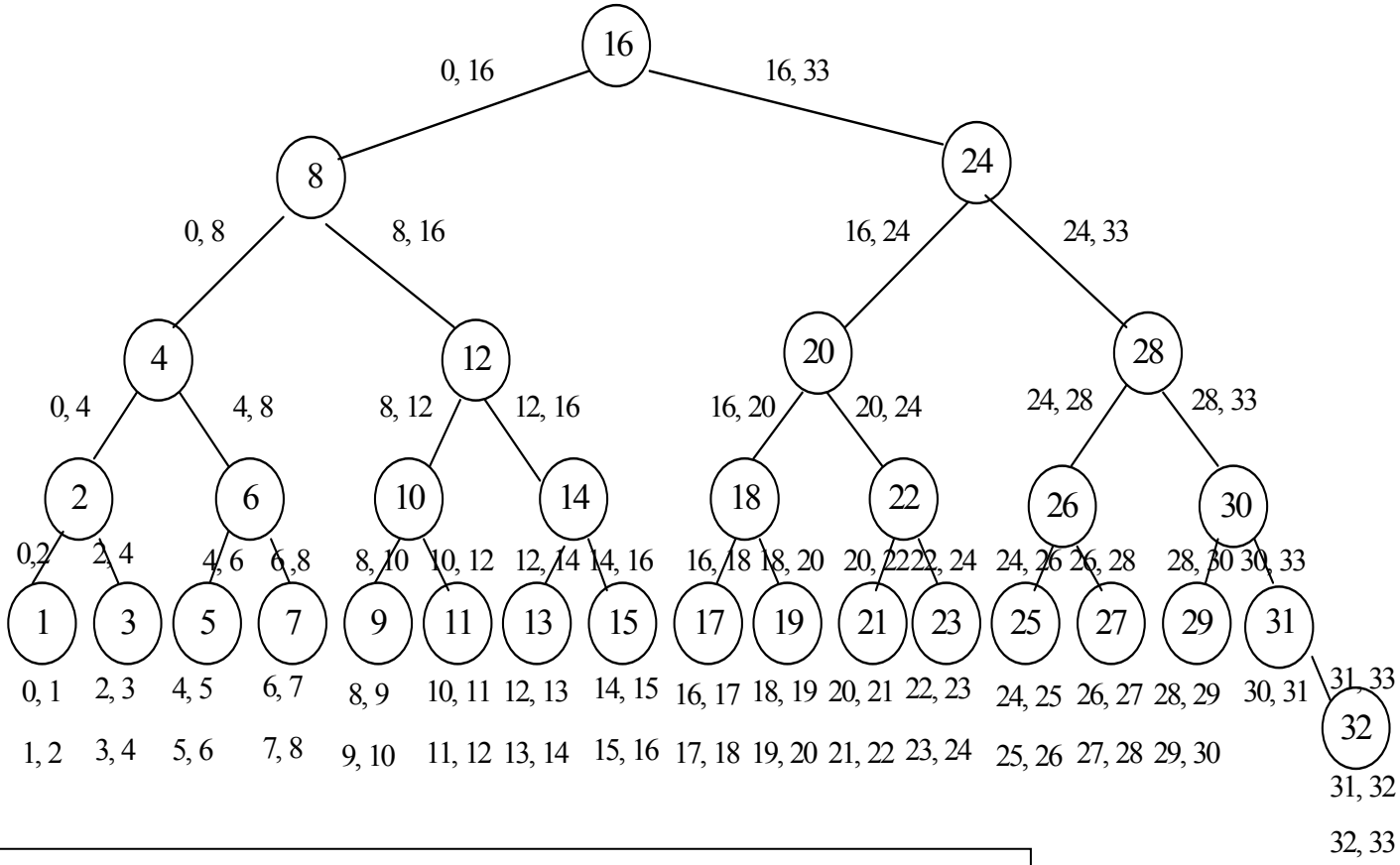
l.p.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
	-3	-1	1	1	3	4	5	7	7	7	25	25	31	34	36	39	42	42	42
l.p.	20	21	22	23	24	25	26	27	28	29	30	31							
	46	48	48	55	55	55	61	63	66	67	74	74							



Przykład przeszukiwania binarnego z powtórzeniami - parzysta liczba elementów

l.p.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19

	-3	-1	1	1	3	4	5	7	7	7	25	25	31	34	36	39	42	42	42
l.p.	20	21	22	23	24	25	26	27	28	29	30	31	32						
	46	48	48	55	55	55	61	63	66	67	74	74	75						



W tablicach uporządkowanych dla **n** elementów mamy:

- średnia liczba przeszukań - **lg n**
- najgorszy przypadek przeszukań - **lg n**

Wyszukiwanie binarne z powtórzeniami, implementacja w C/C++

```
#include <conio.h>
#include <stdio.h>
 typedef int element;
 const long N=12;
 int SzukW(long L, long P, element klucz, long& ktory, element T[]);
 void wyswietl(element T[], long ile);

 void main()
 {  element T[N]={1,2,3,4,5,6,7,8,9,16,18,20};
    element liczba;
    long i, ktory, ile=12;;
    clrscr();
    wyswietl(T,ile);
    do
    { printf("Podaj liczbe: ");
       scanf("%d",&liczba);
       if (SzukW(0, N-1, liczba, ktory, T))
          printf("Szukana liczba ma numer %d w tablicy.\n", ktory+1);
       else printf("Nie ma tej liczby w tablicy.\n");
       printf("Jesli koniec, nacisnij ESC-/nie, nacisnij dowolny klawisz\n");
    } while(getch()!=27);
   }

 int SzukW(long L, long P, element klucz, long& ktory, element T[])
  { long S;
    ktory=P+1;
    L--;
    while ((L+1) != ktory)
     { S = (L + ktory ) / 2;
       if (T[S] < klucz)   L = S;
       else       ktory=S; }
    return !(ktory > P || T[ktory] != klucz);
  }
void wyswietl(element T[], long ile)
 { for(long i=0; i<ile; i++)
    { printf("%d \n", T[i]);
         if (i%20==0)
          {char z=getch();
        if (z=='k') return; }}
      printf("%ld \n", ile);   }
```