Temat : Technika "dziel i zwyciężaj". Przykłady zastosowania. Analiza kosztu rozwiązań opartych na technice „dziel i zwyciężaj”.

Strategia typu „dziel i zwyciężaj’ polega na dekompozycji problemu na pewną skończoną liczbę podproblemów tego samego typu (faza „dziel”), a następnie połączeniu w pewien sposób otrzymanych częściowych rozwiązań w celu odnalezienia rozwiązania globalnego (faza „zwyciężaj”). Zastosowanie metody „dziel i zwyciężaj” często zmniejsza koszt czasowy algorytmu.

1. Problem wyszukiwania w ciągu uporządkowanym

WP: A: $a_0, a_1, ..., a_{n-1}$- ciąg liczb całkowitych ($n > 0$), uporządkowany rosnąco; x – szukana wartość; x jest liczbą całkowitą; liczby w ciągu się nie powtarzają.

WK: p=*true* gdy $x \in A$; p=*false* gdy $x \notin A$.

Operacja elementarna: porównania między elementami ciągu a liczbą x.

Rozmiar danych: n - długość ciągu

Algorytm liniowy

$i = 0$;

while ($i<n$ && $a_i<x$) i++;

$p= a_i =x$;

Złożoność czasowa pesymistyczna

Dane „najgorszego” przypadku to ciąg, w którym x nie występuje i wszystkie liczby w ciągu są mniejsze od x.

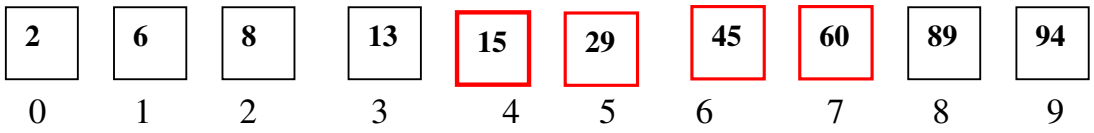
$$T_{max}(n) = \max\{t(d) : d \in D_n\} = n = \Theta(n)$$

Algorytm wyszukiwania binarnego (oparty na strategii „dziel i zwyciężaj”)

```
l=0; r=n-1; p=0;
while (l<=r && ! p)
    {
      m=(l+r)/2;
      if (a_m ==x)  p=1;
      else if (a_m <x)  l=m+1;
            else r=m-1;
    }
```

Przykład 1

A:



x=30
Porównywane z x wartości ciągu A: 15, 60, 29, 45

Złożoność czasowa pesymistyczna
Dane „najgorszego” przypadku to ciąg, w którym x nie występuje.

$$T_{max}(n) = \max\{t(d) : d \in D_n\} =$$

$$= \begin{cases} 2 & dla \quad n = 1 \\ T_{max}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 & dla \;\; n > 1 \end{cases}$$

Przyjmijmy, że n jest potęgą dwójki, czyli $n = 2^k$, dla pewnego k.
Wówczas:

$$T_{max}(n) = T_{max}\left(2^{k-1}\right) + 2 =$$
$$= T_{max}\left(2^{k-2}\right) + 2 + 2 = \ldots =$$
$$= \underbrace{2 + 2 + \ldots + 2}_{k+1} = 2(k+1) = 2(\log_2 n + 1) = \Theta(\log_2 n)$$

Wniosek: Strategia „dziel i zwyciężaj" obniża znacząco koszt algorytmu rozwiązującego problem wyszukiwania w ciągu uporządkowanym.

2. Problem min-max

Algorytm ustalania liczby największej i najmniejszej w ciągu danych.

WP: A: $a_0, a_1, \ldots, a_{n-1}$- ciąg liczb całkowitych ($n > 0$).
WK: Indeks $max \in \{0, 1, 2, \ldots, n-1\}$ taki, że $\forall_{i=0..n-1} a_{max} \geq a_i$ oraz
liczba $min \in \{0, 1, 2, \ldots, n\}$ taka, że $\forall_{i=0..n-1} a_{min} \leq a_i$.

Algorytm liniowy

```
max =0;
min =0;
for (i=1; i<n; i++)
   if (a_i > a_max ) max=i
   else if (a_i < a_min ) min= i;
```

Operacja elementarna: porównania między wartościami a_{max} i a_{min}, a elementami ciągu.

Rozmiar danych: n - długość ciągu, w którym ustalane są wartości max i min

Złożoność czasowa pesymistyczna

Dane „najgorszego” przypadku to ciąg uporządkowany malejąco, na przykład:
n, *n*-1, ... , 1

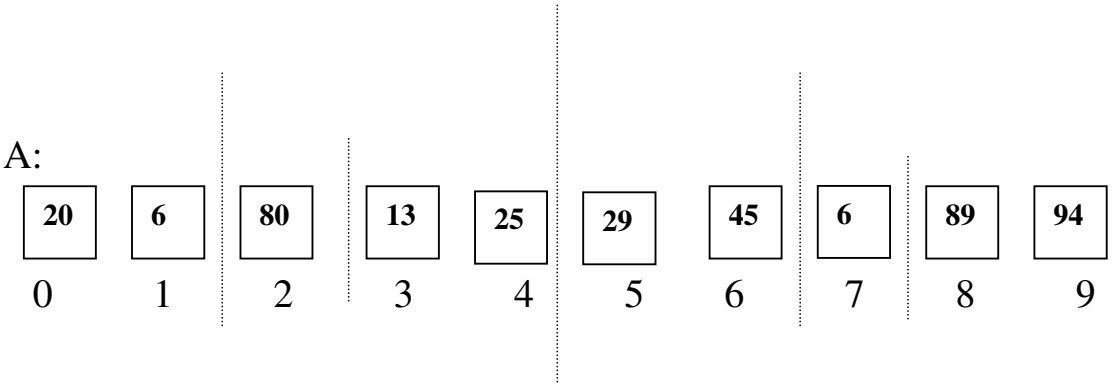
$$T_{max}(n) = \max\{t(d) : d \in D_n\} = 2(n-1) = \Theta(n)$$

Algorytm oparty o strategię „dziel i zwyciężaj”

```
min_max(i, j, min, max)
{
  x, min1, max1, min2, max2  //zmienne lokalne funkcji min_max
  if (i==j) { max=i; min=i; }
    else if  (i+1==j)
              if (a_j > a_i) {
                              max= j;
                              min= i;
                             }
              else {
                     max= i;
                     min= j;
                    }
          else {
                 x=(i+j-1)/2;
                 min_max(i, x, min1, max1);
                 min_max(x+1, j, min2, max2);
                 if (a_min2 > a_min1)  min=min1
                 else min=min2;
                 if (a_max2 > a_max1) max=max2
                 else max=max1;
                }
}
```

Przykład 3

A:



Złożoność czasowa pesymistyczna

Dane „najgorszego" przypadku to dowolne dane spełniające warunek **WP .**

$$T_{max}(n) = \max\{t(d) : d \in D_n\} =$$

$$= \begin{cases} 0 & dla\ \ n = 1 \\ 1 & dla\ \ n = 2 \\ 2 \cdot T_{max}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 & dla\ \ n > 2 \end{cases}$$

Przyjmijmy, że *n* jest potęgą dwójki, czyli $n = 2^k$, dla pewnego *k*. Wówczas:

$$T_{max}(n) = 2 \cdot T_{max}\left(2^{k-1}\right) + 2 =$$
$$= 2 \cdot \left(2 \cdot T_{max}\left(2^{k-2}\right) + 2\right) + 2 = \ldots =$$
$$= 2^{k-1} \cdot T_{max}(2) + 2^{k-1} + \ldots + 2 =$$
$$= \frac{1}{2}n + 2\frac{2^{k-1} - 1}{2 - 1} = \frac{1}{2}n + n - 2 = \frac{3}{2}n - 2 = \Theta(n)$$

Wniosek: Strategia „dziel i zwyciężaj" nie obniża znacząco kosztów czasowych w przypadku problemu min-max.

4. Problem sprawdzania, czy punkt należy do wielokąta wypukłego

WP: $w_0, w_1, ..., w_{n-1}$ – ciąg wierzchołków wielokąta wypukłego W,
x – punkt

WK: Zmienna logiczna p=1, gdy $x \in W$ i p=0 gdy $x \notin W$.

Algorytm naiwny

Sprawdzamy, czy punkt x leży co najmniej jednego z możliwych różnych trójkątów wyznaczonych przez dowolną trójkę różnych wierzchołków wielokąta W.

Takich trójkątów jest: $\binom{n}{3} = n \cdot (n-1) \cdot (n-2)$.

Jeżeli za operację elementarną przyjmiemy sprawdzenie, czy punkt leży wewnątrz trójkąta, to koszt pesymistyczny takiego rozwiązania jest równy:

$$T_{max}(n) = n \cdot (n-1) \cdot (n-2) = \Theta(n^3)$$

Można zmniejszyć liczbę rozważanych trójkątów do n-1, biorąc pod uwagę wszystkie różne trójkąty o tym samym, wspólnym wierzchołku, na przykład wierzchołku w_0. Wówczas złożoność algorytmu wynosi $\Theta(n)$.

Algorytm wykorzystujący metodę „dziel i zwyciężaj” (opis)

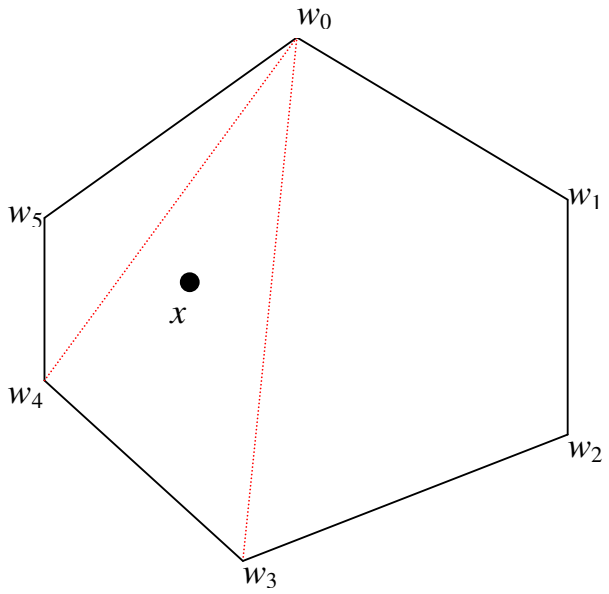
Załóżmy, że W ma więcej niż trzy wierzchołki.

Prowadzimy przekątną łączącą w_0 z $w_{\lceil (n-1)/2 \rceil}$. Są trzy możliwości.

1) Punkt x leży na prostej w_0 - $w_{\lceil (n-1)/2 \rceil}$. W tym przypadku łatwo stwierdzić, czy x należy do odcinka w_0 - $w_{\lceil (n-1)/2 \rceil}$, czy leży poza nim.
2) Punkt x leży po lewej stronie prostej w_0 - $w_{\lceil (n-1)/2 \rceil}$. Należy sprawdzić rekurencyjnie, czy x leży wewnątrz wielokąta o wierzchołkach $w_0, w_2, ..., w_{\lceil (n-1)/2 \rceil}$.

3) Punkt x leży po prawej stronie prostej w_0 - $w_{\lceil (n-1)/2 \rceil}$. Należy sprawdzić rekurencyjnie, czy x leży wewnątrz wielokąta o wierzchołkach $w_0, w_{\lceil (n-1)/2 \rceil}, w_{\lceil (n-1)/2 \rceil+1}, \ldots, w_{n-1}$.

Przykład 4



Rozmiar zadania: n – liczba wierzchołków wielokąta W
Operacja elementarna:

- operacja sprawdzania, po której stronie prostej leży punkt,
- operacja sprawdzania, czy punkt leży wewnątrz trójkąta,
- operacja sprawdzania, czy punkt należy do odcinka

$$T_{max}(n) = \begin{cases} 1 \quad dla \quad n \le 3 \\ T_{max}\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + 2 \quad dla \;\; n > 3 \end{cases}$$

Przyjmijmy, że n jest potęgą dwójki, czyli $n = 2^k$, dla pewnego k.
Wówczas:

$$T_{max}(n) = T_{max}\left(2^{k-1}\right) + 2 =$$
$$= T_{max}\left(2^{k-2}\right) + 2 + 2 = \ldots =$$
$$= 1 + \underbrace{2 + 2 + \ldots + 2}_{k-1} = 2(k-1) + 1 = 2(\log_2 n - 1) + 1 =$$
$$= \Theta\left(\log_2 n\right)$$

Wniosek: Strategia „dziel i zwyciężaj” obniża radykalnie koszt pesymistyczny algorytmu sprawdzania, czy punkt leży wewnątrz wielokąta wypukłego.

Gdy algorytm oparty o strategię „dziel i zwyciężaj” dzieli realizację problemu o rozmiarze *n* na podległe (mniejsze) realizacje o rozmiarze n/c, równanie rekurencyjne opisujące pesymistyczną złożoność tego algorytmu ma postać następującą:

$$T_{max}(n) = \begin{cases} d & gdy \quad n = 1 \\ aT_{\max}\left(\frac{n}{c}\right) + g(n) & gdy \quad n > 1 \end{cases}$$

Gdzie $g(n)$ jest kosztem procesów podziału i łączenia zaś d stałą.