ANALIZA ALGORYTMÓW

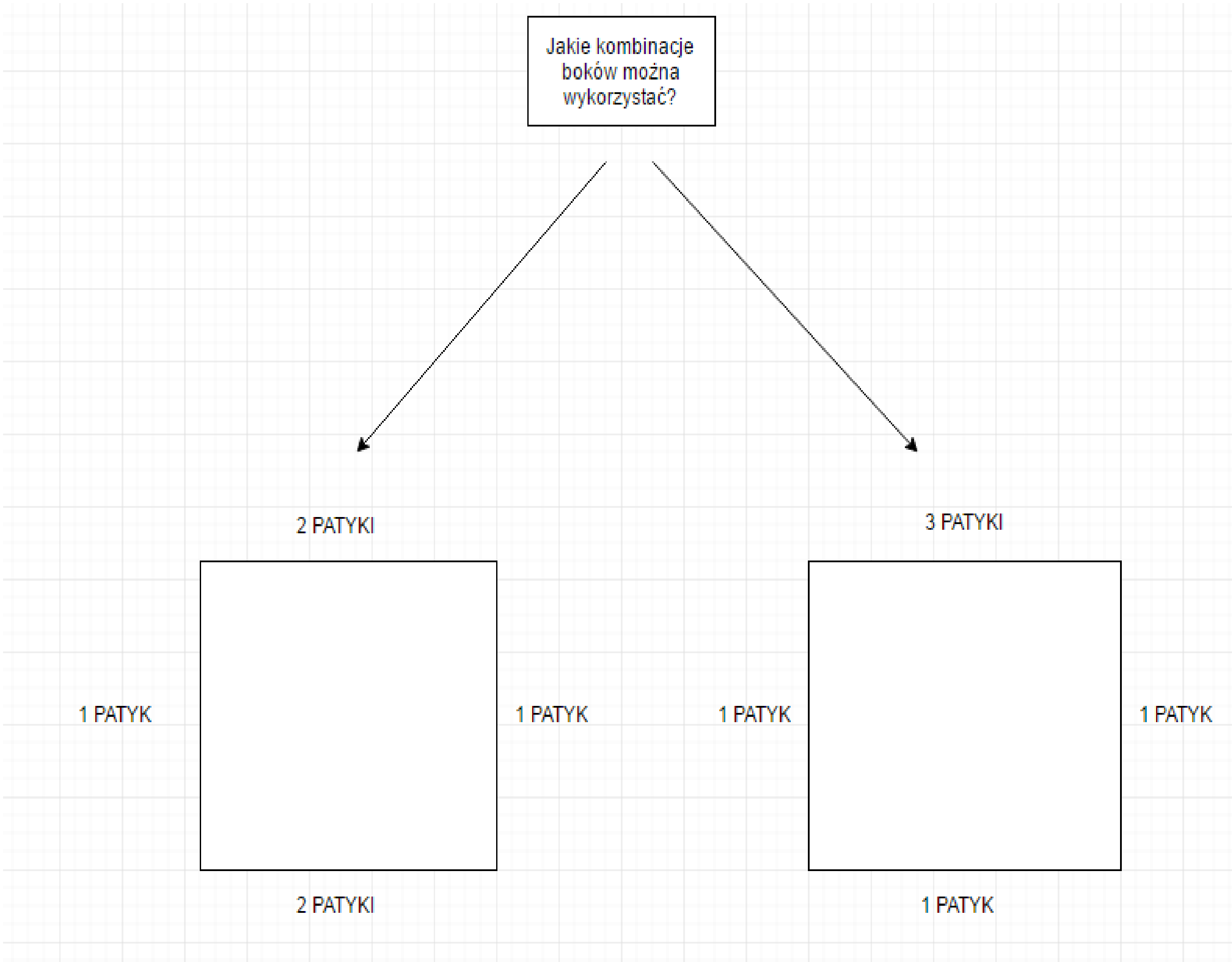
DOKUMENTACJA WSTĘPNA

OPRACOWAŁ : Paweł Walczak

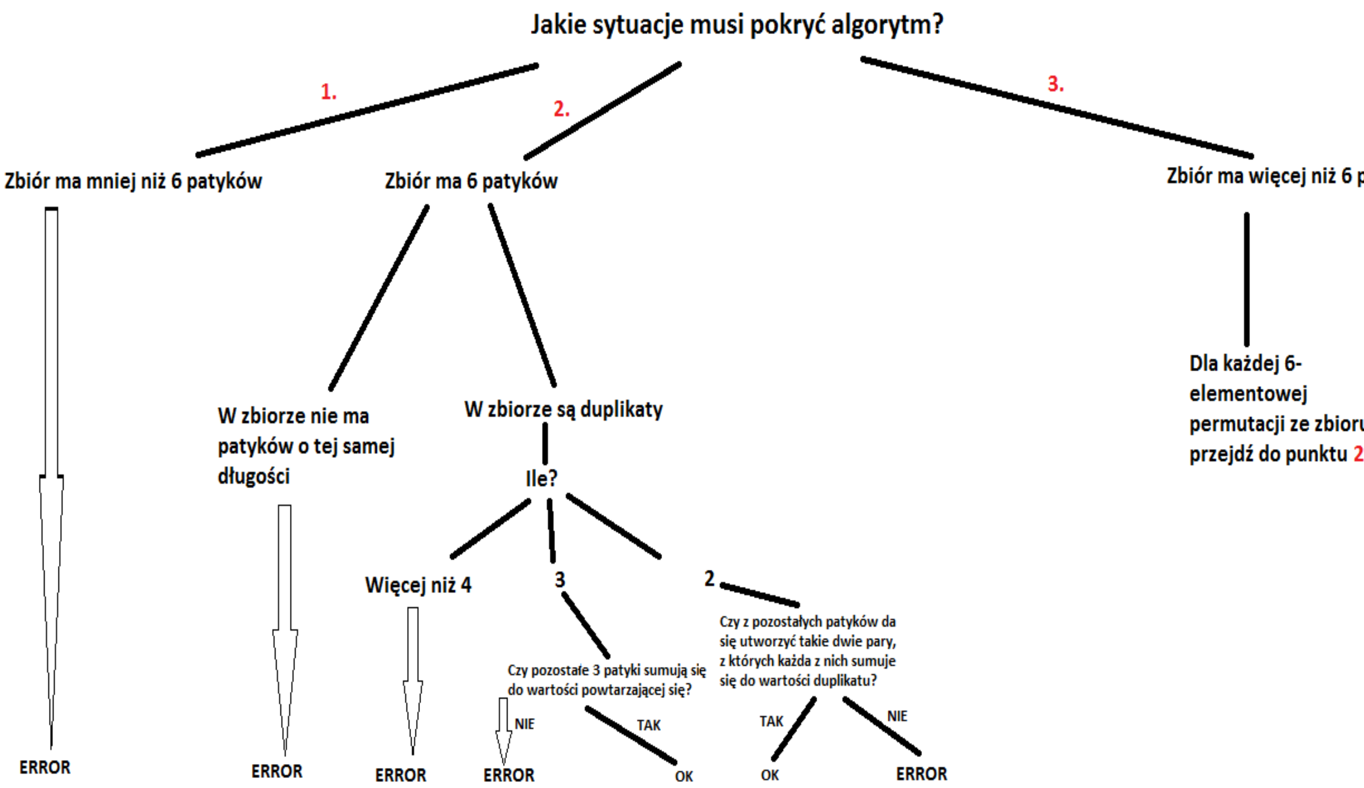
TUTOR: dr inż. Tomasz Trzciński

TREŚĆ:

Zadanie 8. Mamy zestaw S niełamalnych patyków o długości s_io długości s_i, $i \in (1,2,3,\ldots,S)$. Zaproponuj algorytm wyliczający na ile sposobów można zbudować kwadrat przy użyciu 6 z tych patyków i wyznaczy, które patyki należy użyć.



DRZEWO DECYZJI DLA BRUTE-FORCE



PODEJŚCIE BRUTE-FORCE

N = ilość patyków

Double tab[N] = zbiórpatyków

If N < 6

return ERROR

if N==6

Jeśli w zbiorze nie ma duplikatów

return ERROR;

Jeśli w zbiorze wiecej niż 4 patyki o tej samej dlugosc

return ERROR;

Jeśli w zbiorze sa dwa takie same patyki

Sprawdz czy z pozostałych elementow zbioru da się stworzyc dwie pary, z których kazda z nich będzie sumowac się dlugosci patyka powtarzajacego się

Jeśli_tak _to_wypisz_ktore_patyki_uzyc

Jeśli w zbiorze sa 3 takie same patyki

Sprawdz czy pozostale elementy zbioru sumuja się do dlugoscpatykowpowtarzajacych się

Jeśli_tak _to_wypisz_ktore_patyki_uzyc

```
        Else

                return ERROR;

else

        for (inti = 0; i< N-6; i++)

         for(int j = i + 1; j < N-5; j++)

          for( int k = j + 1; k < N – 4; k++)

           for( int l = k + 1; l < N – 3; l++ )

            for( int m = l + 1; m < N –2; m++ )

              for( int n = m + 1; n < N – 1; n++ )

                {

                  // mamy tu kazda permutacje 6-elementowa ze zbioru

                Jeżeli_w_permutacji_nie_ma_duplikatów

                continue;

        Jeżeli_w_permutacji_wiecej_niz_3_takie_same_wartosci

                continue;

                  Jeżeli_w_permutacji_3_takie_same_wartosci

                Sprawdz_czy_reszta_sumuje_się_do_duplikatu

                        Jeśli_tak _to_wypisz_ktore_patyki_uzyc

        Jeżeli_w_permutacji_2_takie_same_wartosci

                    Sprawdz_czy_istnieją_dwie_inne_pary_ktore_sumuja_sie_do_duplikatu
```

Jeśli_tak _to_wypisz_ktore_patyki_uzyc

}

end

Złożoność BRUTFEORCE – $\boldsymbol{O(n^6)}$

Proponowany przez mnie algorytm, będzie zamiast w zachłanny sposób badać każdą permutację w zbiorze, będzie rozwiązywał problem w sposób optymalniejszy.

PROPONOWANY ALGORYTM – PSEUDOKOD

int N = ilość patyków

doubletab[N] = zbiór patyków

intlliczba = liczba patykow o tej samej dlugosci

liczba = 1

pary = struktura danych do przechowywania par i indeksow

triplety = struktura danych do przechowywania tripletow i indeksow

posortowanaTab[] = tab.sort(); // prawdopodobnie quicksort

```
indexTab[] = tab.indexSort(); // tablica ktora przechowuje indeksy elementow z oryginalnej tablicy po przesortowaniu

if N < 6
    return ERROR

else if N == 6
    ifposortowanaTab[6] != posortowanaTab[5] // nie ma nawet dwochpatykow o tej samej dlugosci
        return ERROR
    elseifposortowanaTab[5] != posortowanaTab[4] // jest para patykow o tej samej dlugosci
        sprawdz czy istnieja w 4 pierwszych elementach posortowanej tablicy pary, z ktorychkazda sumuje sie do duplikatu
    else
        sprawdz czy pierwsze 3 elementy tablicy sumujasie do wartosci tripletu
else
    for( int I = 1; i <n; i++ ) ones[posortowanaTab[i]]++; // tutaj sa przechowywane ilosciwystepien w kodzie patykow o danej dlugosci

for( int I = 1; i<n; i++ )
        for( int j = i + 1; j <n ; j++)
            if ( posortowanaTab[j] != posortowanaTab[j-1])
                    sum+=ones[posortowanaTab[j]]*(ones[posortowanaTab[j]]-1)*(ones[posortowanaTab[j]]-2)/6 * twos[posortowanaTab[j]-posortowanaTab[i]]; // układ bokow ( 3,1,1,1) , wzory wynikają z dwumianu newtona
            for( int k = 1; k <i ; k++)
                    twos[posortowanaTab[i]+posortowanaTab[k]]++; // tutaj dodajemy ilość wystapien par sumujących się do wartości indeksu
    for ( int i = 1; i <n; i++) // tutaj zalatwimysprawe z ukladembokow (1,1,2,2)
        if(posortowanaTab[i]!=posortowanaTab[i-1] &&ones[posortowanaTab[i]]>1)
        {
```

```
                mozliwosciDlaPar = 
ones[posortowanaTab[i]]*(ones[posortowanaTab[i]] – 1)/2; //  tu tez dwumian newtona 
zastosowany, dla szukania par bez powtorzen

                temp = 0;

                for ( int j = 1; posortowanaTab[j]*2 <posortowanaTab[i]; j++)

                    if (posortowanaTab[j]!=posortowanaTab[j-1]) {

                        sum += mozliwosciDlaPar * ones[posortowanaTab[j]]* 
(ones[posortowanaTab[j]]-1)/2 *ones[posortowanaTab[i]-posortowanaTab[j]]* 
(ones[posortowanaTab[i]-posortowanaTab[j]]-1) / 2;

                        sum += mozliwosciDlaPar * temp * 
ones[posortowanaTab[j]]*ones[posortowanaTab[i]-posortowanaTab[j]];

                        temp += 
ones[posortowanaTab[j]]*ones[posortowanaTab[i]-posortowanaTab[j]];

                    }

                    if(posortowanaTab[i]%2==0) { // wszystkie takie same to znaczy 
kwadrat ( X/2+X/2,X/2+X/2,X,X) i (A+B, X/2+X/2, X,X) gdzie A+B=X


        sum+=mozliwosciDlaPar*ones[posortowanaTab[i]/2]*(ones[posortowanaTab[i]/2]-
1)* (ones[posortowanaTab[i]/2]-2)*(ones[posortowanaTab[i]/2]-3)/24;

                        sum += 
mozliwosciDlaPar*temp*ones[posortowanaTab[i]/2]*(ones[posortowanaTab[i]/2]-1)/2;

                    }

            }

        return sum;
```

Złożoność algorytmu:

- quicksort – O(nlogn)

- reszta kodu – O(n^2)

Ogólna złożoność algorytmu – O(n^2)