**ANALIZA ALGORYTMÓW**

**DOKUMENTACJA WSTĘPNA**

**OPRACOWAŁ : Paweł Walczak**

**TUTOR: dr inż. Tomasz Trzciński**

***TREŚĆ:***

Zadanie 8. Mamy zestaw S niełamalnych patyków o długości o długości Zaproponuj algorytm wyliczający na ile sposobów można zbudować kwadrat przy użyciu 6 z tych patyków i wyznaczy, które patyki należy użyć.

***KONCEPCJA WYKONANIA:***

* Najpierw wyobraźmy sobie, że zbiór patyków przechowujemy w tablicy ( która jest strukturą o szybkim czasie dostępu ( O(1) ) .Kolejnym krokiem, jest posortowanie danej tablicy, od najkrótszych długości do najdłuższych. Przydatne również będzie zachowanie oryginalnej tablicy oraz stworzenie tablicy, która będzie przechowywała indeksy oryginalnych elementów po sortowaniu. Operację sortowania wykonam implementując własną klasę **QuickSort**, która oprócz posortowania tablicy, stworzy również wspomnianą wyżej tablicę indeksów. Zgodnie z teorią, ta operacja powinna być złożoności O(nlogn) lub w pesymistycznym przypadku maksymalnie O(, gdzie n to ilość patyków.
* Zastanówmy się , nad tym jak znaleźć kwadraty wykorzystując 6 patyków. Mamy dwa sposoby:

1. Zauważmy, że aby utworzyć taki kwadrat, możemy znaleźć przynajmniej dwa patyki o takiej samej długości length. W tym wypadku szukamy w tablicy, pary patyków , które po zsumowaniu będą miały długość length. Takich par szukamy w posortowanej tablicy dla indeksów mniejszych od pierwszego znalezionego zduplikowanego elementu. Jeżeli znajdziemy minimum dwie takie pary to możemy stworzyć kwadrato bokach {(length),(length),(para1),(para2)}. Zauważmy, że takich par i duplikatów może być więcej i należy znaleźć każdą opcję.
2. Kwadrat możemy stworzyć również, używając trzech patyków o tej samej długości length oraz trzech innych patyków które będą sumowały się do długości length. Postępujemy tutaj analogicznie do sytauacji z punktu 1.

**Rozważmy tutaj sposób w jaki zrealizować podane punkty. Należy postawić odpowiedzi na kilka pytań.**

1. **Jak znaleźć duplikaty ( patyki o tej samej długości ) ?**

Użyjemy tutaj klasy, którą nazwiemy Engine. Będzie posiadać ona metodę, w której będzie zaimplementowana pętla, przechodząca przez wszystkie elementy **tablicy posortowanej.** Będzie ona szukała długości, które są takie same. Po natknięciu się na taką sytuację, zlicza ona ile elementów się powtórzyło. Następnie kontynuuje w poszukiwaniu kolejnych duplikatów. Złożoność przeszukiwania O(n).

1. **Jakich struktur danych i klas użyć do przechowywania, duplikatów ( patyków o tej samej długości)?**

Do przechowywania duplikatów użyjemy klasy **Duplicate.** Będzie ona posiadała pola:

***int value // długość patyka***

***int numberOfSticks // liczba patyków o tej samej długości***

***std::vector<int> sticksIndices // indeksy duplikatów w oryginalnej tablicy***

***std::vector<DoubleSide> doubleSides// wektor klas DoubleSide ( omówione poźniej)***

***std::vector<TripleSide> tripleSides// wektor klas TripleSide ( omówione poźniej)***

Wektor klas Duplicate, będziemy z kolei przechowywać w klasie Engine. Do tego wektora będziemy „pushować” w sytuacji gdy znajdujemy w punkcie pierwszym Duplikat, i mamy informację o długości zduplikowanego patyka, ilości duplikatów, oraz ich indeksach w oryginalnej tablicy.

1. **Jak znaleźć pary/trójki patyków, które będą odpowiadały danym duplikatom?**

W momencie wykrycia duplikatu w punkcie 1. wywołujemy metodę, która będzie przeszukiwała posortowaną tablicę od indeksu 0 do indeksu pierwszego elementu zduplikowanegow celu znalezienia pary/trójki która zsumuje się do odpowiedniej długości. Inaczej wygląda procedura dla szukania par a trójke:

* **PARY**

W sytuacji gdy tablica jest posortowana, jest to bardzo proste. Pseudokod:

i=0; j= index od first duplicate element, sum is the value of duplicate element

If i+j = sum, then return (i,j). Some special function to check whether there are more elements of value sortedArray[i]/ sortedArray[j]

If i+j < sum, then move i to the right one position.  
If i+j > sum, then move j to the left one position.

Złożoność tej operacji to O(n).

* **TRÓJKI**

Tutaj jest już nieco trudniej.

i=0, n=index of first duplicate element ( in case there are at least 3 of them!) , sum is the value of duplicate element

for(i in 1…n)

j = i+1

k = n

while ( k >= j)

if(sortedArr[i] + sortedArr[j] + sortedArr[k] )

return (A[i], A[j], A[k]) // match!

else

(sortedArr[i] + sortedArr[j] + sortedArr[k] > 0 ) ? k--: j++

Done

Tutaj złożoność to już O(.

1. **Jak przechowywać te boki?**

Tutaj z pomocą idzie wspomniana wcześniej klasa **DoubleSide** oraz **TripleSide.** W chwili znalezenia pary lub trójki zostanie wywołana odpowiednio pewna metoda, która albo stworzy nowy obiekt DoubleSide lub TripleSide. Klasa DoubleSide będzie miała pola:

**int firstSide // pierwszy bok**

**int secondSide // drugi bok**

**int firstSideIndex // indeks w oryginalnej tablicy pierwszego boku**

**int secondSideIndex // analogicznie**

Klasa TripleSide będzie wyglądał analogicznie. Taki nowy obiekt będziemy wrzucać do wektora danej instancji Duplicate.

**UWAGA: KAŻDY NOWY INSERT DO WEKTORA TO KOSZT O(n), gdzie n długość wektora!**

**Po zakończeniu tych operacji,** mamy już pełną informację o możliwych długościach boków ( pełna informacja w klasie Duplicate), teraz należy stworzyć każdy możliwy kwadrat. Taki kwadrat będzie reprezentowała klasa Square, która będzie posiadała pola informujące, o długościach boków i ich indeksach. Tutaj należy będzie stworzyć kwadraty o różnych kombinacjach ( np. trzeba będzie uwzględnić możliwość wystąpenia boków stworzonych z patyków o tych samych długościach, ale o różnych indeksach w oryginalnej tablicy). Będziemy te kwadrat przechowywać w klasie Squares w wektorze std::vector <Square> squares.

**KONTEKST MENU ORAZ POBIERANIA DANYCH.**

Program będzie posiadał Menu konsolowe. Będzie możliwa opcja pobrania danych z pliku, lub manualnego wprowadzenia danych. Na koniec działania programu będzie wyświetlania pełna informacja o czasie wykonywania programu. Rozważam, również utworzenie statystyki na temat czasu wykonywania programu dla różnych zbiorów danych (tabela, wykresy).