ALGORYTMY KOMBINATORYCZNE

----== WYKŁAD 1 ==----

Ogólne założenia:

Algorytmy kombinatoryczne zajmują się strukturami dyskretnymi. Takimi jak zbiory, permutacje, grafy czy ścieżki.

Będziemy się zajmowali:

- Wygenerowaniem wszystkich struktur – (oraz dowiedzeniem się ile takich struktur łącznie jest – bez generowania ich wszystkich)
- Znajdowaniem najlepszej struktury – optymalizacja

Na zajęciach poruszana będzie ogólna idea idąca za poszczególnymi algorytmami i następnie będzie omawiania ich dokładniejsza specyfika

Na Labach – będziemy realizowali zadania / projekty. Możemy pisać algorytmy w dowolnym języku i możemy korzystać z wszystkich pomocy. Należy wskazać czym się inspirowało oraz przed wszystkim należy rozumieć jak dany algorytm działa!

Projektów na ocenę będzie 2/3 w zależności od ilości przerobionego materiału z czego jedno zadanie będzie indywidualne (każdy dostanie inne). Na podstawie punktów z projektu będzie ocena z LAB'ów – i następnie jest ona przepisywana w ramach wykładu. Jak się nie zaliczy przedmiotu poprzez projekty to będzie poprawkowy test! Więc warto się skupić na projektach!

Algorytmy: Kombinatoryczne / Dyskretne – inaczej: przeliczalne / skończone.

Komputery też są dyskretne – mają skończoną liczbę stanów!

Wyróżniamy 3 rodzaje zadań:

- Generowanie struktur
- Przeliczanie możliwych do wygenerowania struktur
- Optymalizacja (wybór najlepszej struktury)

Dla nas nie ważne jest to że zbiór jest skończony … ale jak duży on jest! Z punktu widzenia algorytmu i działania komputera, wielkość zbioru ma znaczenie. Ponieważ ma to

wpływ na pojawianie się problemów z pamięcią i z czasem. Ważne z naszego punktu widzenia jest znalezienie takiego algorytmu który w jak najlepszym czasie znajdzie odpowiednią, interesującą nas wartość/strukturę.

TEMAT 1 : Generowanie struktur dyskretnych:

Problem -> jak daną strukturę przedstawić w komputerze za pomocą liczb?

- S = zbiór struktur (np. zbiór wszystkich grafów na N wierzchołkach – skończona struktura!)
- N = |S| (Liczba tych struktur)
- Funkcja; rank : S -> { 0 , 1 … N-1 } Każdemu elementowi naszego zbioru (strukturze) nadaje liczbę. (Liczbę rankingową – która jest w liniowym porządku)
- Tu musi zachodzić bijekcja! (Każdej liczbie – przyporządkowuje jedną strukturę). Bijekcja ma tą własność, że istnieje funkcja odwrotna.
- Funkcja (odwrotna do funkcji rank): unrank: { 0,1,…N-1 } -> S (Liczbom przyporządkowujemy struktury!)

Funkcje rank i unrank należy tak zaimplementować aby działały stosunkowo szybko.

Inne funkcje które tez będziemy starali się tworzyć:

s ∈ S

- Successor(s) – następnik
- Predeccessor(s) – poprzednik

Liczba rankingowa: rank(s) = K oznacza że K obiektów jest wcześniej niż obiekt s.

Tych bijekcji jest bardzo dużo. My będziemy wybierać tą, którą łatwo wyliczyć. Warto dobrać taką bijekcje – która spowoduje że rozwiązanie naszego zadania będzie łatwiejsze niż jak wybierzemy inną - aby wybrany algorytm był efektywny i pomógł nam rozwiązać nasze zadanie.

Liczba rankingowa w sposób automatyczny wygeneruje nam porządek. Ale możemy też zacząć z drugiej strony. Mając porządek można odnaleźć liczbę rankingową.

Generowanie podzbiorów:

S = { 1 , 2 , … , N } , liczba podzbiorów = 2^N

Np.: S = { 1 , 2 , 3 } , T ⊂ S , X(T) = (3 , 2 , 1) to wektor charakterystyczny, tutaj badamy czy 3 , 2 , 1 należą do zbioru.

T	X(T) = (3 , 2 , 1)	rank(T)
∅	000	0
{ 1 }	001	1
{ 2 }	010	2
{ 1 , 2 }	011	3
{ 3 }	100	4
{ 1 , 3 }	101	5
{ 2 , 3 }	110	6
{ 1 , 2 , 3 }	111	7

X(T) = (X_{n-1} , X_{n-2} , X_0)

$$x_i = \begin{cases} 1 & i \in T \\ 0 & i \notin T \end{cases}$$

Porządek Leksykograficzny (lex):

$$x_i = \begin{cases} 1 & n - i \in T \\ 0 & n - i \notin T \end{cases}$$

Żeby wygenerować wszystkie podzbiory danego zbioru nie musimy działać na zbiorach – możemy zapisać je jako liczbę.

SubsetLexRank(n, T) – oznacza z góry funkcję, obliczającą liczbę rankingową w porządku leksykograficznym. Dla zbioru T (n elementowego) chcemy znaleźć liczbę rankingową:

```
r = 0
For(int i = 1 ; i < n ; i++) {
    If (i in T) {
        r = r + 2^(n-1)
    }
}
return (r)
```

SubsetLexUnRank(n,r) – mamy dane podzbiory, zbioru n-elementowego, o rankingu r i chcemy znaleźć ten podzbiór który ma tą liczbę rankingową r.

```
T = ∅
For (i = n ; i > 1 ; i--) {
    If (r % 2 == 1) {
        T = T.append(i)
    }
    r = r  // 2 (dzielenie całkowite)
}
return (T)
```

Porządek minimalnych zmian:

- Chcemy tak uporządkować nasze podzbiory aby każdy następny, jak najmniej różnił się od poprzedniego. Miarą tego, jak bardzo się różnią dwa zbiory może być ilość elementów różnicy symetrycznej:
    a) Różnica symetryczna (podzbiór naszego zbioru, składający się z tych elementów, które należą tylko do jednego z nich)
    b) $T_1 \Delta T_2 = (T_1 - T_2) \cup (T_2 - T_1)$

- $dist(T_1 , T_2) = |T_1 \Delta T_2|$ (odległość jest liczbą całkowitą) jeśli dystans = 0 to zbiory T_1 i T_2 są identyczne… a jeśli $dist(T_1 , T_2) = 1$ to zbiory różnią się tylko jednym elementem.

- Chcemy ustawić nasze wszystkie możliwe podzbiory w takiej kolejności, aby każdy następny – powstał z poprzedniego poprzez dodanie lub usunięcie jednego elementu.

Odległość Heminga:

- Jest to miara która pozwala określić jak bardzo wektory binarne się od siebie różnią. Robimy to na bitach. Chcemy tak uporządkować wektory binarne, aby każda następna różniła się od poprzedniej jednym bitem.
  $V_1 = (0 , 1 , 0)$
  $V_2 = (0 , 1 , 1)$
  $dist_H(V1,V2)$ = liczba bitów na których się wektory różnią.

Kody Greya:

Jest to sposób zapisywania liczb gdzie każdy następny kod – różni się od poprzedniego – dokładnie jednym bitem. (Porządek minimalnych zmian) Zwróć uwagę, że to działa cyklicznie – działa w kółko, a więc ostatni od pierwszego również różni się jednym elementem. Możemy to w dowolnym momencie zacząć i będzie to działać w kółko, cyklicznie. (Ciekawostka: „Utworzenie kodu Greya jest jednoznaczne ze znalezieniem cyklu Hamiltona w kostce n-wymiarowej.")

∅ , {3} , {2,3} , {2} , {1,2} , {1,2,3} , {1,3} , {1}

000, 001, 011, 010, 110 , 111, 101, 100

Teraz będziemy starali się znaleźć jakąś ogólną regułę, która pozwala nam generować taki ciąg. (porządek minimalnych zmian).

G^N będziemy oznaczali ciąg na N bitach. Jest to lista której elementy będziemy oznaczali w ten sposób)

$G^n = [G_0^n , G^N, ... G^N_{2^{n-1}}]$

$G^1 = [0,1]$

$G^2 = [00,01,11,10]$

Itd… (zwróć uwagę, na kolejność!)

---== WYKŁAD 2 ==---

$G^3 = [000 , 001, 011, 010 , 110 , 111 , 101 , 100]$

Jak wygenerować Kod Greya? (chcemy mieć jakiś algorytm do tego):

Waga Heminga jakiegoś ciągu: $w(T) = \#_1$ (ilość jedynek w tym ciągu)

Algorytm znajdywania następnika (Successor):

- Jeżeli liczba jedynek jest parzysta to zamieniamy ostatni bit na przeciwny.
- Jeżeli liczba jedynek jest nieparzysta to szukamy pierwszego wystąpienia jedynki i poprzedzający bit (przed tą 1-szą jedynką) zamieniamy na bit przeciwny) – Uwaga! Czytamy zapis binarny od prawej strony. 😊 (Jeśli mamy nieparzystą liczbę jedynek – oraz ostatnią cyfrą jest jeden – to oznacza że jest to ostatnia liczba w naszym ciągu)

```
GreyCodeSuccessor(n,T){
    If (w(T) % 2 == 0) {
        U =  T Δ {n}
    }
    else {
        J = n
        while( j ∉ T) {
            J = J - 1
        }
        If ( J = 1) {
            Return „end"
        }
        Else {
            U = T Δ {j-1}
        }
    }
    return ( U )
```

r		G^3_r
0	000	000
1	001	001
2	010	011
3	011	010
4	100	110
5	101	111
6	110	101
7	111	100

Załóżmy sobie, że nasza reprezentacja liczby rankingowej $b_n\ b_{n-1} \dots b_0$ to:

$$b_0=1 \quad b_1=1 \quad b_2=0 \quad b_3=0$$

a kolejne bity kodu Greya będziemy oznaczali jako $a_n\ a_{n-1} \dots a_0$ i my chcemy znaleźć zależność pomiędzy a i b.

Własności:

$a_j = (b_j + b_{j+1}) \% 2$

$b_j = (\sum_{i=j}^{N-1} a_j\ ) \% 2$

Przykład: (liczba rankingowa) 0101101 <-> 111011 (Kod Greya)

```
GreyCodeRank(n,T){
  r = 0
  b = 0
  for ( i = n-1 ; i > 0 ; i--) {
        if ( n-i  ∉  T) {
        b = 1 – b
        }
        if ( b == 1 ) {
        r = r + 2
        }
  }
}
```

```
GreyCodeUnrank(n,r) {
  T = ∅
  b = 0
  for ( i = n-1 ; i > 0 ; i--) {
        b = r//2^i
        if (b != b_1) {
            T = T ∪ {n-i}
        }
   }
b_1 = b
r = r – b2^i
return(b)
```

K – elementowe podzbiory. Czyli jak mamy zbiór n-elementowy, to chcemy umieć wygenerować i ustawić w jakimś porządku, znaleźć liczby rankingowe dla wszystkich jego K elementowych podzbiorów których ilość podana jest wzorem $\binom{n}{k}$.

Przykład n=5 , k=3 ; $\binom{5}{3} = 10$, natomiast liczba wszystkich możliwych podzbiorów to $2^5 = 32$

Porządek leksykograficzny:

1,2,3	[1,2,3]	0
1,2,4	[1,2,4]	1
1,2,5	[1,2,5]	2
1,3,4	itd	3
1,3,5		4
1,4,5		5
2,3,4		6
2,3,5		7
2,4,5		8
3,4,5		9

K_SubsetLexSuccessor(T,n,k):

Patrzymy zawsze na ostatni element listy który możemy zwiększyć;

Przykład 12678 -> 12679 -> 12689 -> 12789 -> itd.

Skonstruowanie liczby rankingowej i unrank.

Dlaczego dla elementu (1,4,5) liczba rankingowa = 5?

Przypomnienie – liczba rankingowa jest to liczba elementów które poprzedzają naszą strukturę (w tym przypadku podzbiór) w naszym porządku. Liczba rankingowa = 5 ponieważ przed nią znajduje się dokładnie 5 innych podzbiorów.

Żeby to wyliczyć możemy te podzbiory podzielić na pewne klasy.

$T_1 = 1$, $T_2 = 4$, $T_3 = 5$

1) $X_1 < T_1$
2) $X_1 = T_2$ $X_2 < T_2$
3) $X_1 = T_1$ $X_2 = T_2$ $X_3 < T_3$
4) Itd. Itd. Liczymy ilości takich podzbiorów i ich suma = liczba rankingowa

Rank(T) = $rank(T) = \sum_{i=1}^{k} \sum_{j=t_{i-1}+1}^{t_i - 1} \binom{n-j}{k-i}$

K_SubsetLexRank(T,n,k) {

r = 0

$t_0 = 0$ (wartownik – pozwala pętli nie wyjść poza wskazane elemen)

for (i=0 ; i < n ; i++){

  if($t_{i-1} + 1 <= t_i - 1$){

    for($j=t_{i-1}+1$; $j < t_i - 1$; i++){

      r = r + $\binom{n - j}{k - i}$

    }

} }

Return(r)

Żeby ulepszyć powyższy algorytm warto wziąć inny porządek. Porządków jest wiele do wyboru)
(Porządek koleksograficzny: w tym przypadku od największego do najmniejszego (w ramach listy))

Wcześniejszy porządek	Inny/nowy porządek T	r
123	321	0
124	421	1
134	431	2
234	432	3
125	521	4
135	531	5
235	532	6
145	541	7
245	542	8
345	543	9

Algorytmy który działają na tym porządku są dużo prostsze np.: algorytm rank będzie składał się z jednej pętli.