Εφαρμοσμένη Συνδυαστική

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΠΑΝ/ΜΙΟΥ ΠΕΙΡΑΙΩΣ 2021

1η Ομάδα Ασκήσεων (Παράδοση μέχρι την Δευτέρα 19/4/2021)

Άσκηση (1). *Να διατυπωθεί αλγόριθμος κατασκευής του συνόλου $\mathcal{F}_{n,k} \subseteq \mathcal{B}_{n,k}$ των δυαδικών λέξεων μήκους n με k μονάδες και χωρίς διαδοχικές μονάδες.*

Άσκηση (2). *Δίνονται ακέραιοι k, n, s, με $1 \leq k \leq n$. Να διατυπωθεί αναδρομικός αλγόριθμος κατασκευής των ακολουθιών του*

$$\mathcal{A}_{n,k} = \{a_1 a_2 \cdots a_k : 1 \leq a_1 < a_2 < \cdots < a_k \leq n\}$$

με $\sum_{i=1}^{k} a_i = s$.

Άσκηση (3). *Να διατυπωθεί αλγόριθμος κατασκευής των στοιχείων του συνόλου*

$$R_n = \{b \leq \mathcal{B}_n : b \leq_L \mathrm{rev}(b)\},$$

δηλαδή των δυαδικών λέξεων $b_1 b_2 \cdots b_{n-1} b_n$, $b_i \in \{0,1\}$, $i \in [n]$, που ικανοποιούν την ανισότητα

$$b_1 b_2 \cdots b_{n-1} b_n \leq_L b_n b_{n-1} \cdots b_2 b_1.$$

Λύση άσκησης 1

Απαρίθμηση: Κάθε λέξη $a \in \mathcal{F}_{n,k}$, $n \geq 2$, έχει $n-k$ μηδενικά και ανάμεσα σε κάθε ζεύγος από γειτονικά μηδενικά μπορεί να υπάρχει μία ή καμία μονάδα, οπότε έχει τη μορφή

$$1^{x_1}01^{x_2}0\cdots 1^{x_{n-k}}01^{x_{n-k+1}}, \qquad x_1+\cdots+x_{n-k+1}=k, \quad x_i \in \{0,1\}.$$

Επομένως υπάρχουν $\binom{n-k+1}{k}$ τέτοιες λέξεις.

Γεννήτρια συνάρτηση: Κάθε λέξη $a \in \mathcal{F}_{n,k}$, $n \geq 2$, διασπάται ως

$$a = 0b \qquad \text{ή} \qquad a = 10c, \qquad b \in \mathcal{F}_{n-1,k}, c \in \mathcal{F}_{n-2,k-1}, \tag{1}$$

με $\mathcal{F}_{0,0} = \{\varepsilon\}$, $\mathcal{F}_{1,0} = \{0\}$, $\mathcal{F}_{1,1} = \{1\}$. Επομένως, θέτοντας $f_{n,k} = |\mathcal{F}_{n,k}|$, προκύπτει η αναγωγική σχέση

$$f_{n,k} = f_{n-1,k} + f_{n-2,k-1}, \quad n \geq 2, \qquad f_{0,0} = f_{1,0} = f_{1,1} = 1.$$

Η αναγωγική αυτή σχέση λύνεται πιο εύκολα θεωρώντας τη ΓΣ

$$F = F(x,y) = \sum_{n\geq 0}\sum_{k\geq 0} f_{n,k}x^n y^k,$$

του συνόλου $\mathcal{F} = \bigcup_{n,k}\mathcal{F}_{n,k}$, η οποία βάσει της διάσπασης

$$\mathcal{F} = \{\varepsilon\} \cup \{1\} \cup 0\mathcal{F} \cup 10\mathcal{F}$$

ικανοποιεί τη σχέση $F = 1 + xy + xF + x^2yF$, από την οποία προκύπτει ότι

$$\begin{aligned} F &= \frac{1+xy}{1-x(1+xy)} = \sum_{m\geq 0} x^m(1+xy)^{m+1} = \sum_{m\geq 0}\sum_{k=0}^{m+1}\binom{m+1}{k} y^k x^{m+k} \\ &\overset{m+k=n}{=} \sum_{n\geq 0}\sum_{k=0}^{\lfloor (n+1)/2\rfloor}\binom{n-k+1}{k} y^k x^n \end{aligned}$$

Επομένως, είναι $f_{n,k} = [x^n y^k]F = \binom{n-k+1}{k}$ και επιπλέον, αθροίζοντας για k, προκύπτει η σχέση

$$\begin{aligned} \sum_{k=0}^{\lfloor n/2\rfloor}\binom{n-k+1}{k} &= [x^n]F(x,1) = [x^n]\frac{1+x}{1-x-x^2} = [x^{n+1}]\frac{x}{1-x-x^2} + [x^n]\frac{x}{1-x-x^2} \\ &= f_{n+1} + f_n = f_{n+2}, \end{aligned}$$

όπου (f_n) η ακολουθία των αριθμών Fibonacci, η οποία ορίζεται από την αναγωγική σχέση

$$f_{n+2} = f_{n+1} + f_n, \quad n \geq 0, \qquad f_0 = 0, f_1 = 1.$$

Κατασκευή: Η αναδρομική κατασκευή βασίζεται στη διάσπαση (1).

```
1 def rec(pos, j):#pos = current position, j = remaining ones
2     global count, calls
3     if j == 0: count+=1; #print(b[1:])
4     elif pos == N and j == 1: b[pos] = 1; count += 1; #print(b[1:])
5     else:
6         if N - pos + 1 == 2*j:
7             for i in range(1, j+1): b[N-2*i+2] = 1; b[N-2*i+1] = 0
8             count += 1; #print(b[1:])
9         elif N - pos + 1 > 2*j: calls += 1; rec(pos+1, j) #if #remaining positions + 1 >= 2k, b[n]=0
10        if N - pos + 1 >= 2: b[pos] = 1; calls += 1; rec(pos+2, j-1) #If # remaining positions >=2, b[n]=1, b[n+1]=0
```

Η λέξη b περιέχει αρχικά μόνο μηδενικά. Η αρχική κλήση είναι η rec(1, K). Οι έλεγχοι στις γραμμές 3 και 6 απαλοίφουν τα μονοπάτια με κόμβους με 1 παιδί. Συγκεκριμένα, όταν δεν απομένουν μονάδες (γραμμή 3), τότε η λέξη θα περιέχει μόνο μηδενικά από την τρέχουσα θέση και μετά. Όταν οι θέσεις που απομένουν αρκούν οριακά για να φιλοξενήσουν τις j υπόλοιπες μονάδες (γραμμή 6), τότε η λέξη θα έχει τη μορφή $(01)^*$ ή $1(01)^*$ από την τρέχουσα θέση και μετά. Έτσι, ο αλγόριθμος είναι CAT. Η κατασκευή είναι προφανώς λεξικογραφική.

Παρακάτω φαίνεται ο επαναληπτικός αλγόριθμος λεξικογραφικής κατασκευής:

```
def lexNext(): #assumes b[0]=0
    j = N
    ones = 0 #number of 1 to the right of j
    for j in range(N, 0, -1): #find j>=1 s.t. b[j]=b[j-1]=0
        if b[j] == 1: ones += 1
        elif b[j]==0 and b[j-1]==0 and ones>0:
            b[j] = 1 #set 1, then minimize suffix containing (ones-1) 1's
            for i in range(j+1, N-2*ones + 3): b[i] = 0
            for i in range(1, ones): b[N-2*i+2] = 1; b[N-2*i+1] = 0
            #for i in range(j+1, N-j-2*ones-1): b[i]=0
            return True
    return False
```

Λύση άσκησης 2

Θέλουμε να κατασκευάσουμε τις ακολουθίες $a = (a_1, \ldots, a_k) \in A_{n,k}$, δηλαδή γνησίως αύξουσες ακολουθίες ακεραίων $a : [k] \to [n]$, με $\sum_{i=1}^{k} a_i = s$. Συμβολίζουμε με $A_{n,k,s}$ το σύνολο αυτών. Ισοδύναμα, αρκεί να κατασκευάσουμε τις δυαδικές λέξεις $b = b_1 \cdots b_n$ που αντιστοιχούν σε λύσεις της εξίσωσης

$$b_1 + \cdots + b_n = k, \qquad b_1 + 2b_2 + \cdots + nb_n = s, \quad b_i \in \{0, 1\}, \tag{*}$$

αφού κάθε λέξη b κωδικοποιεί μια ακολουθία a, βάσει της σχέσης

$$a_i = j \Leftrightarrow b_j = 1 \text{ και } \sum_{r=1}^{j} b_r = i, \qquad i \in [k], j \in [n].$$

Προφανώς, το πλήθος των λύσεων αυτών ισούται με $|A_{n,k,s}| = [x^k y^s] \prod_{i=1}^{n} (1 + xy^i)$.

Το ελάχιστο και μέγιστο άθροισμα k αριθμών του συνόλου $\{\ell, \ell+1, \cdots, r\}$, όπου $r - \ell + 1 \geq k$, είναι αντίστοιχα ίσα με

$$m(\ell, k) = \ell + \ell + 1 + \cdots + \ell + k - 1 = k\ell + \frac{k(k-1)}{2} = \frac{k(2\ell + k - 1)}{2}$$

και

$$M(r, k) = r + r - 1 + \cdots + r - (k-1) = kr - \frac{k(k-1)}{2} = \frac{k(2r - k + 1)}{2},$$

οπότε θα πρέπει προφανώς να ισχύει $m(1, k) \leq s \leq M(n, k)$ ώστε το $A_{n,k,s}$ να είναι μη κενό.

```
#decide A[r], ps = partial sum of inserted elements
def rec(r, ps): #r = #elements remaining to be chosen
    global count
    if r == 1: #val = s - ps
        #if val > 0 and val < A[r+1]: #redundant? previous checks ensure that a solution exists
        A[r] = s-ps; count+=1; #print(A[1:])
    else:
        m = (r - 1)*r//2; #min sum of first r-1 elements
        for val in range(r, A[r+1]): #valid values for A[r]
            if m + val + ps > s: break #cannot reach goal s
            elif m + val + ps == s:
                for i in range(1, r): A[i] = i;
                A[r] = val; count+=1; break
            M = (2*val - r + 1)*r//2 #sum for last r elements
            if ps + M > s: A[r] = val; rec(r-1, ps+val)
            elif ps + M == s:
                for i in range(r, 0, -1): A[i] = val; val-=1;
                count+=1;
```

Λύση άσκησης 3

Απαρίθμηση: Οι κανόνες παραγωγής των λέξεων αυτών είναι οι

$$S \to \varepsilon|0|1|0S0|1S1|0B1, \qquad B \to \varepsilon|0B|1B$$

Ισοδύναμα, κάθε λέξη $w \in \mathcal{R}_n$, διασπάται ως

$$w = 0u0 \quad \acute{\eta} \quad w = 1u1 \quad \acute{\eta} \quad w = 0v1, \qquad u \in \mathcal{R}_{n-2}, v \in \mathcal{B}_{n-2}$$

για $n \geq 2$, όπου $\mathcal{R}_0 = \mathcal{B}_0 = \{\varepsilon\}$ και $\mathcal{R}_1 = \mathcal{B}_1 = \{0, 1\}$.

Επομένως, η ΓΣ $F(x) = \sum_{n\geq 0} |\mathcal{R}_n| x^n$ ικανοποιεί τη σχέση

$$F(x) = 1 + 2x + 2x^2 F + \frac{x^2}{1-2x},$$

οπότε προκύπτει ότι

$$F(x) = \frac{1-3x^2}{(1-2x)(1-2x^2)} = \frac{1/2}{1-2x} + \frac{(1+\sqrt{2})/4}{1-x\sqrt{2}} + \frac{(1-\sqrt{2})/4}{1+x\sqrt{2}}$$

και

$$[x^n]F(x) = \frac{1}{2}2^n + \frac{1+\sqrt{2}}{4}(\sqrt{2})^n + \frac{1-\sqrt{2}}{4}(-\sqrt{2})^n = 2^{n-1} + 2^{\lfloor (n+1)/2 \rfloor - 1}$$

Επαλήθευση: Με τον ακόλουθο κώδικα

```python
import sympy as sm
x = sm.symbols('x')
f = (1-3*x**2)/(1-2*x)/(1-2*x**2)
print("f(x) =", f, "\n=", f.series(x,0,11))
```

Output:

```
f(x) = (1 - 3*x**2)/((1 - 2*x)*(1 - 2*x**2))
= 1 + 2*x + 3*x**2 + 6*x**3 + 10*x**4 + 20*x**5 + 36*x**6 + 72*x**7 + 136*
   x**8 + 272*x**9 + 528*x**10 + O(x**11)
```

βρίσκουμε τους συντελεστές της ΓΣ $f(x)$ για $0 \leq n \leq 10$. Εκτελούμε έναν αλγόριθμο κατασκευής των δυαδικών λέξεων, απορρίπτοντας όσες λέξεις ανήκουν στο $\mathcal{B}_n \setminus \mathcal{R}_n$ με κατάληλη συνάρτηση ελέγχου:

```python
def inR(b, l, r): #assumes 1 <= l, r <= n
    while l <= r :
        if b[l] < b[r]: return True #b < rev(b)
        elif b[l] > b[r]: return False #b > rev(b)
        l, r = l+1, r-1
    return True #b = rev(b)
```

Τα αποτελέσματα του αλγορίθμου ταυτίζονται με τους παραπάνω συντελεστές, άρα η απαρίθμηση είναι σωστή.

Βιβλιογραφία: Αναζητούμε την ακολουθία $1, 2, 3, 6, 10, 20, 36, 72, 136, 272, 528$ στην Online Encyclopedia of Integer Sequences OEIS και βλέπουμε ότι αντιστοιχεί στην ακολουθία A005418. Εκεί μπορούμε να πάρουμε πληροφορίες για ισοδύναμα αντικείμενα που απαριθμούνται από αυτή την ακολουθία και αποτελέσματα σχετικά με αυτήν.

Κατασκευή: Η κατασκευή όλων των δυαδικών λέξεων με απόρριψη μπορεί να δώσει αλγόριθμο CAT, διότι $|\mathcal{B}_n|/|\mathcal{R}_n| \leq 2 = O(1)$ και η συνάρτηση απόρριψης μπορεί να δειχθεί ότι εκτελεί κατά μέσο όρο $O(1)$ ελέγχους. Ωστόσο, είναι προτιμότερος ένας αλγόριθμος κατασκευής μόνο των στοιχείων του $\mathcal{R}_n$.

Για την επαναληπτική λεξικογραφική κατασκευή, αρκεί να προσδιορίσουμε την επόμενη της τυχαίας λέξης $b = b_1 b_2 \cdots b_n \in \mathcal{R}_n$. Έστω $c = c_1 c_2 \cdots c_n \in \mathcal{R}_n$ η επόμενη αυτή λέξη. Η c προκύπτει εντοπίζοντας το δεξιότερο 0 στην b, έστω ότι αυτό βρίσκεται στη θέση $k \in [n]$, θέτοντας $c_k = 1$, $c_1 \cdots c_{k-1} = b_1 \cdots b_{k-1}$ και μετατρέποντας το επίθεμα $c_{k+1} \cdots c_n$ στο ελάχιστο δυνατό, ώστε $c_1 \cdots c_k \leq \operatorname{rev}(c_{n+1-k} \cdots c_n)$. Για την τελευταία μετατροπή, βρίσκουμε τη θέση j του πρώτου 0 και τη θέση k του τελευταίου 0 στην b. Θέτουμε $c_1 \cdots c_{k-1} = b_1 \cdots b_{k-1}$, $c_k = 1$. Επιπλέον, οι τελευταίες $j-1$ θέσεις παίρνουν τιμή 1, δηλαδή $c_{n-j+2} \cdots c_n = b_{n-j+2} \cdots b_n$, ενώ οι θέσεις $c_{k+1} \cdots c_{n-j+1}$ γίνονται ίσες με 0. Τέλος, αν $c_j \cdots c_{n+1-j} \notin \mathcal{R}$, θέτουμε $c_{n+1-j} = 1$.

Ο αλγόριθμος αυτός έχει λίγο καλύτερη απόδοση από τον αναδρομικό αλγόριθμο με απόρριψη.

```python
def lexNext():
    j, m = 1, (n+1)//2   #j will become the position of the first 0
    while j <= m and b[j]==1 : j = j+1 #find first 0
    if j > m: return False #no next
    k = n+1-j #only 1's to the right of k
    while b[k]==1:
        b[k]=0
        k=k-1 #find rightmost 0
    b[k]=1 #set this to 1
    if not inR(b, j, n+1-j): b[n+1-j] = 1
    return True
```