Diskretno modeliranje 2013/2014

2. in 3. vaje

k-ELEMENTNE PODMNOŽICE

Algoritem 1: kSubsetLexSucessor(n, k, T)

```
U ← T;
i ← k − 1;
while i ≥ 0 and T[i] == n + (i − (k − 1)) do
    i ← i − 1;
end
if i < 0 then
    return "Ne obstaja";
else
    for j ← i to k − 1 do
        U[j] = T[i] + j − (i − 1);
    end
end
```

RAZLAGA: Najprej pregledamo podmnožico z desne in najdemo element, ki ga lahko povečamo. Nato temu elementu povečamo vrednost za 1, vsak naslednji pa postane za 1 večji od prejšnjega.

Primer. Elementi naše množice naj bodo $1, 2, 3, 4, 5, 6, 7$ (n=7). Iščemo pa naslednika $k = 4$ elementne podmožice T = [1, 2, 6, 7]. Pomikamo se od zadaj. Prvi element, ki ga lahko povečamo je 2, drugače bi s povečanjem dobile element, ki je že v podmnožici. Tako je naslednik T množica [1, 3, 4, 5].

Algoritem 2: kSubsetLexRank(n, k, T)

```
r ← 0;
for j ← 1 to T[0] − 1 do
    r ← r + C(n−j, k−1);
end
for i ← 2 to k do
    for j ← T[i − 2] + 1 to T[i − 1] − 1 do
        r ← r + C(n−j, k−i);
    end
end
return r;
```

RAZLAGA: `Rank` k-elementne podmnožice $T = \{t_0, t_1, \ldots, t_n\}$ je določen kot število podmnožic pred njo v leksikografski ureditvi. Prva zanka `for` prešteje vse podmnožice, ki

imajo prvi element manjši od prvega elementa T (tj. če je prvi element T enak 3, preštejemo vse podmnožice, ki se začnejo z 1 ali 2). Druga zanka `for` na i-tem koraku prešteje vse podmnožice, ki so do $i-1$ elementa enaka T, i-ti element, pa imajo manjši od i-tega elementa T. Ko vse te podmnožice, ki so pred T seštejemo, je r ravno enak $\texttt{rank}(T)$.

Primer. Naj bo T = [3, 4, 7, 8] in $n = 8$ ter $k = 4$ Poiščimo njen rank. Koliko je množic, ki se začnejo s števko manjšo od 3.

- Vse, ki se začnejo z 1.
  Imamo $n-1$ izbir, ki jih razporedimo na $k-1$ preostalih mest, $\binom{n-1}{k-1}$.
- Vse, ki se začnejo z 2.
  Imamo $n-2$ izbir, ki jih razporedimo na $k-1$ preostalih mest. Poleg 2 ne smemo izbrati tudi 1, saj smo zahtevali naraščanje. $\binom{n-2}{k-1}$.

Za naš primer je to $\binom{7}{3} + \binom{6}{3} = 55$. Nadaljujemo.

- Koliko je takih, ki se začnejo s 3 in so manjše od naše množice. [2, ., ., .] Za drugi element nimamo možnost, mora biti enak 4. Množic, ki bi imele drugi element $2+1 = 3 <= t_2 <= 2 = 3-1$ ni.
- Koliko je takih množic, ki se začnejo s 3 in 4 ter so manjše od naše množice. [3, 4, ., .]. Za tretji element t_3 mora veljati $4+1 = 5 <= t_3 <= 6 = 7-1$. Takih, kjer je $t_3 = 5$ je ravno $\binom{8-5}{4-3}$. Takih, kjer je $t_3 = 6$ je ravno $\binom{8-6}{4-3}$. Torej vse skupaj $\binom{3}{1} + \binom{2}{1} = 5$.
- Koliko je takih množic, ki se začnejo z 3 in 4 ter 7 in so manjše od naše množice. [4, 3, 7, .]. Spet nimamo nobene, saj bi moralo veljati $7+1 = 8 <= t_4 <= 7 = 8-1$.

Rank množice [3, 4, 7, 8] je torej enak $55 + 5 = 60$.

Algoritem 3: kSubsetLexUnrank(n, k, r)

```
x ← 1;
T ← {};
for i ← 1 to k do
    while (n−x choose k−i) ≤ r do
        r ← r − (n−x choose k−i);
        x ← x + 1;
    end
    T[i − 1] ← x;
    x ← x + 1;
end
return T;
```

Primer. Naj bo $n = 8$ ter $k = 4$. Poiščimo podmnožico moči k, ki ima rank 60. S katero števko se začne?

1.
   - Vseh podmnožic, ki se začnejo z 1 je: Imamo $n-1$ izbir, ki jih razporedimo na $k-1$ preostalih mest, $\binom{n-1}{k-1}$. Torej takih je $\binom{7}{3} = 35$. $r = r - 35 = 25$.
   - Vseh podmnožic, ki se začnejo z 2 je: Imamo $n-2$ izbir, ki jih razporedimo na $k-1$ preostalih mest, $\binom{n-2}{k-1}$. Torej takih je $\binom{6}{3} = 20$. $r = 25 - 20 = 5$.
   - Vseh podmnožic, ki se začnejo z 3 je $\binom{5}{3} = 10$. To pomeni, da je prva števka 3.
2. Trenutna situacija: [3, ., ., .]
   - Vseh podmnožic, ki se začnejo s 3 in imajo drugi element enak 4 je: $\binom{4}{2} = 6$. Torej je druga števka 4, saj $6 \not\leq r = 5$.
3. Trenutna situacija: [3, 4, ., .]
   - Vseh podmnožic, ki se začnejo s 3 in imajo drugi element enak 4 ter tretji 5 $\binom{3}{1} = 3$. Torej $r = r - 3 = 2$.
   - Vseh podmnožic, ki se začnejo s 3 in imajo drugi element enak 4 ter tretji 6 $\binom{2}{1} = 2$. Torej $r = r - 2 = 0$.
   - Vseh podmnožic, ki se začnejo s 3 in imajo drugi element enak 4 ter tretji 7 $\binom{1}{1} = 1$. Torej $1 \not\leq r = 0$. Torej je tretji element enak 7.

   Trenutna situacija: [3, 4, 7, .]

   - Vseh podmnožic, ki se začnejo s 3 in imajo drugi element enak 4 in tretjega 7 ter četrtega 8 je $\binom{0}{0} = 1$. Torej $1 \not\leq r = 0$. Torej je četrti element enak 8.

[3, 4, 7, 8]

Razlaga: Tu rešujemo obraten problem, tj. iz ranka bi radi ugotovili podmnožico. Zanka `for` pove, da gledamo i-ti element podmnožice, ki jo iščemo. Pri tem povečujemo število x in ugotavljamo, če so podmnožice, ki imajo na i-tem elementu število x pred iskano podmnožico (tj. imajo manjši rank). Ko za nek x niso več, je to število ravno i-ti element množice, ki jo iščemo.

Algoritem 4: kSubsetLexNext(n, k, T)

```
r ← kSubsetLexRank(n, k, T);
if r == (n k) − 1 then
    return "Ne obstaja";
end
return kSubsetLexUnrank(n, k, r + 1);
```

Algoritem 5: kSubsetLexAll(n, k)

```
T ← [];
for r ← 0 to (n k) − 1 do
    T[r] ← kSubsetLexUnrank(n, k, r);
    // Pozor, to je Append v pythonu.
end
return T;
```

Algoritem 6: kSubsetLexRandom(n, k)

```
r ←randint(0, (n k) − 1);
return kSubsetLexUnrank(n, k, r);
```

Permutacije

Algoritem 7: PermLexSucessor(n, π)

```
i ← n − 2;
while i ≥ 0 and π[i] > π[i + 1] do
    i ← i − 1;
end
if i < 0 then
    return "Ne obstaja";
else
    // odzadaj poiščemo prvega, ki je manjši od π[i]
    j ← n − 1 ;
    while π[j] < π[i] do
        j ← j − 1;
    end
    // zamenjamo π[i] in π[j]
    t ← π[j] ;
    π[j] ← π[i] ;
    π[i] ← t;
    // obrnemo zadnji del
    ρ ← π ;
    for k ← i + 1 to n − 1 do
        π[k] ← ρ[n + i − k];
    end
end
return π;
```

Razlaga: S prvo zanko `while` odkrijemo element permutacije $\pi = (\pi_0, \pi_1, \ldots, \pi_{n-1})$, ki ga lahko zamenjamo (povečamo), da dobimo naslednika. Nato izmed elementov, ki so za njim poiščemo prvega, ki je večji, ter ju v novi permutaciji zamenjamo. Nato še v novi permutacijo popravimo vse elemente za njim, tako da jih razvrstimo naraščajoče.

Primer. Poglejmo si, kaj je naslednik permutacije [3, 6, 2, 7, 5, 4, 1]. Prvi element, ki ga lahko povečamo je 2, saj velja $2 < 7 > 5 > 4 > 1$. Torej moramo zamenjati 2 in 4, saj je 4 prvi element v seznamu, gledano odzadaj, ki je večji od 2. Tako dobimo pomožno permutacijo [**3, 6, 4**, 7, 5, 2, 1], zadnji del seznama je potrebno še obrniti, tako dobimo [3, 6, 4, 1, 2, 5, 7].

Algoritem 8: AllPermSuccessor(n)

```
T ← [];
i ← 0;
T[0] ← [1, 2 ..., n];
while π! = "Ne obstaja" do
    i = i + 1;
    π ← PermLexSucessor(n, π);
    T[i] ← π;
    // Naredi kopijo π v pythonu, ko prirejaš!
end
```

Algoritem 9: PermLexRank(n, π)

```
r ← 0;
ρ ← π;
for j ← 1 to n do
    r ← r + (ρ[j − 1] − 1)(n − j)!;
    for i ← j + 1 to n do
        if ρ[i − 1] > ρ[j − 1] then
            ρ[i − 1] ← ρ[i − 1] − 1;
        end
    end
end
return r;
```

Razlaga: Najprej za j-ti element permutacije π izračunamo koliko je permutacij z manjšimi vrednostmi na j-tem mestu. Nato pogledamo elemente desno od j-tega. Vsem, ki imajo vrednost večjo od j-tega, jo lahko zmanjšamo za 1, saj gledamo podproblem, ki ne vsebuje vrednosti, ki jo ima j-ti element. Nato rešujemo rekurzivno naprej.

Primer. Izračunamo rank permutacije $[3, 5, 1, 2, 4]$.

- Na prvem mestu imamo lahko 1 ali 2 na ostalih mestih je izbira poljubna.

  $$\text{rank}([3, 5, 1, 2, 4], 5) = 2 \times 4! + \text{rank}([4, 1, 2, 3], 4).$$

  Vsem številom večjim od 3 smo zmanjšali vrednost za 1, tako smo dobili 4 elementno podmnožico.

- $$\text{rank}([4, 1, 2, 3], 4) = 3 \times 3! + \text{rank}([1, 2, 3], 3).$$

  Na prvem mestu imamo lahko 1 ali 2 ali 3 na ostalih mestih je izbira poljubna. Elementov ni potrebno popraviti, saj so vsi manjši od 4.

- $$\text{rank}([1, 2, 3], 3) = 0 + \text{rank}([1, 2], 2).$$

  Na prvem mestu ne moremo izbrati nobenega elementa, saj je 1 najmanjši.

- $$\mathrm{rank}([1,2],2) = 0 + \mathrm{rank}([1],1) = 0.$$

  Vse skupaj dobimo $\mathrm{rank}([3,5,1,2,4],5) = 48 + 18 + 0 = 66$.

Algoritem 10: PermLexUnrank(n, r)

```
π[n − 1] ← 1;
for j ← 1 to n − 1 do
    d ← (r mod (j + 1)!)/j!;
    r ← r − d · j!;
    π[n − 1 − j] ← d + 1;
    for i ← n − j to n − 1 do
        if π[i] > d then
            π[i] ← π[i] + 1;
        end
    end
end
return π;
```

Razlaga: Glede na to, kako je izračunan rank permutacije r, ga lahko zapišemo kot $\sum_{i=1}^{n-1} d_i i!$, kjer je $0 \leq d_i < i$ (enolično). Permutacijo dešifriramo v obratni smeri (od desne proti levi). Z deljenjem z $(j+1)!$ dobimo kot ostanek, doprinos $(n-j)$-tega elementa k ranku, tj. $(\pi[n-j-1]-1)j!$. Vrednost $(n-j)$-tega elementa dobimo tako, da izraz delimo z $j!$ ter prištejemo 1. Ker pa smo pri računanju ranka za 1 zmanjševali elemente, ki so bili večji od izbranega, jih tu za 1 povečamo ter nadaljujemo proti levi.

Primer. Poizkusimo rekonstruirati kar prejšnji primer. PermLexUnrank(5, 66)

- P = [0, 0, 0, 0, 1]; d = (66 mod 2!)/1! = 0; r = r - 0*1!; P = [0, 0, 0, 1, 1];
  P = [0, 0, 0, 1, 2];
- P = [0, 0, 0, 1, 2]; d = (66 mod 3!)/2! = 0; r = r - 0*2!; P = [0, 0, 1, 1, 2];
  P = [0, 0, 1, 2, 3];
- P = [0, 0, 1, 2, 3]; d = (66 mod 4!)/3! = 3; r = r - 3*3! = 48; P = [0, 4, 1, 2, 3];
  P = [0, 4, 1, 2, 3];
- P = [0, 4, 1, 2, 3]; d = (48 mod 5!)/4! = 2 : r = r - 2*4! = 0; P = [3, 4, 1, 2, 3];
  P = [3, 5, 1, 2, 4];

Algoritem 11: randomPerm(n)

```
r ←randint(0, n! − 1);
return PermLexUnrank(n, r);
```