Задача НАЙ-МАЛЪК ПОДНИЗ

Пояснение към решенията

Означаваме с N дължината на s и с M – дължината на p

Решение за $N < 40$ **и** $M < 10$ – програма `small_23p.cpp`

Това решението е с кубична сложност спрямо дължината на низа s. Чрез двоен цикъл генерираме всички поднизове на низа `s`. За всеки такъв подниз `ss` чрез функцията `check` проверяваме. дали `ss` съдържа всичките елементи на низа `p`. Когато това е изпълнено, в променливата `m` записваме текущата най-малка дължина на намерения подниз и в променливата `res` записваме самия низ. Накрая отпечатваме низа `res`.

Решение за $N < 2\,000$ **и** $M < 100$ – програма `small_38p.cpp`

Вместо да генерираме всички поднизове на s, както е в предното решение, можем да генерираме само поднизовете с дължина L, която да е по-голяма или равна на дължината на низа p и генерирането да става по нарастване на L. Тогава първият намерен подниз е решението на задачата.

Решение за $N < 31\,000$ **и** $M < 100$ – програма `small_54p.cpp`

Използваме двоично търсене, за да намерим дължината `m` на най-късия подниз на `s`, който съдържа всички елементи на `p`. Търсената дължина `m` не може да е по-голяма от дължината на `s` и не може да е по-малка от дължината на `p`. Затова започваме двоичното търсене в интервал с долна граница `L=p.size()` и горна граница `U=s.size()`. В цикъла `while(L<=U)` пресмятаме средната дължина `mid=(L+U)/2` и проверяваме чрез функцията `found(mid,b)` дали има подниз на `s` с дължина `mid`, който съдържа всичките елементи на `p`. Ако това е така, няма смисъл да търсим минималната дължина за стойности по-големи от `mid`, променяме горния край на интервала за търсене `U=mid-1` и записваме в `m` текущата минимална дължина, а в `i` записваме началния индекс на текущия минимален подниз.

Функцията `bool found(int len,int &b)` работи по следния начин: В цикъл по `i` разглеждаме всички поднизове на `s`, които са с дължина `len`. Във вектора `c` записваме кратностите на буквите от `p` и проверяваме, дали низът `ss=s.substr(i,len)` съдържа всичките елементи на `p` чрез цикъла

```
for(char ch : ss) if(c[ch]>0) c[ch]--;
```

Когато след този цикъл, векторът `c` съдържа само нули, това означава че сме намерили подниз, съдържащ всичките елементи на `p`. Тогава излизаме от тялото на функцията `found` с връшане на `true` и `b=i`.

Решение за $N < 900\,000$ **и** $M < 1000$ – програма `small_77p.cpp`

Използваме двоично търсене по същия начин, както в предното решение, но с по-бързо работеща функция `bool found(int len,int &b)`, която сега е реализирана както следва:

Използва се векторът `t`, в който в началото се пресмятат кратностите на първите `len` елемента на `s`. След това чрез функцията `sub(t)` се проверява, дали в началния подниз на `s` с дължина `len` се съдържат всички елементи на `p`. Ако е така, функцията `found` връща `true` и зарежда `b` с нула. Ако не е така, започва цикъл с индекс `i`, в който поднизът с дължина `len` се движи надясно, коригират се стойностите на `t`, а именно:

```
t[s[i-1]]--; t[s[i+len-1]]++;
```

и се извиква `if(sub(t)){b=i; return true;}`.

Пълно решение – програма `small_100p.cpp`

Това решението е с линейна сложност спрямо дължината на низа s. В програмата използваме вектор `cP`, за който пресмятаме в `cP[p[k]]` колко пъти буквата `p[i]` се съдържа в низа `p`. Аналогично използваме вектор `cS`, за който пресмятаме в `cS[s[j]]` колко пъти буквата `s[j]` се съдържа в подниза на `s`, образуван във всеки момент от движещ се прозорец с ляв индекс `i` и десен индекс `j`. В цикъл с индекс `j` движим десния индекс `j` на прозореца.

В променливата `c` броим колко елемента от низа `p` се съдържат във всеки момент в прозореца. Когато `c` стане равно на дължината `n2` на `p`, движим левия индекс `i` на прозореца, докато прозорецът стане възможно най-малък, но съдържащ всичките елементи на `p`, при което коригираме стойностите в масива `cS`.

Така намираме текущ най-малък подниз на `s`, който съдържа всичките елементи на `p` и в променливите `min_len` и `min_beg` записваме съответно неговата дължина и началният му индекс. Продължаваме процеса до достигане края на низа `s` и отпечатваме `s.substr(min_beg, min_len)`, когато `min_beg` не е равно на началната си стойност `-1`.

Пример за действието на алгоритъма с движещия се прозорец при следния вход:

```
xezbxccxaaczbx
abc
```

1. Започваме от началото на низа `s` и търсим прозорец, съдържащ всички елементи на низа `p`. Когато намерим най-малкия такъв прозорец, спираме:
`xezbxccxa`aczbx

2. Намаляваме отляво прозореца възможно най-много, но да остане такъв, че да съдържа всички елементи на низа `p`. Когато намерим най-малкия такъв прозорец, спираме:
`xez`**`bxccxa`**`aczbx` Така намираме подниз `bxccxa`, съдържащ всички елементи на `p` и той е с дължина 6.

3. Намаляваме отляво прозореца с един елемент и той вече не съдържа всички елементи на низа `p`.
`xezb`xccxa`aczbx`

4. Разширяваме отдясно текущия прозорец, докато той отново съдържа точно всички елементи на низа `p`:
`xezb`xccxaaczb`x`

5. Намаляваме отляво текущия прозорец, докато той съдържа всички елементи на низа `p`, но е най-малък такъв:
`xezbxccxa`**`aczb`**`x` Така намираме подниз `aczb`, съдържащ всички елементи на `p` и той е с дължина 4.

6. Намаляваме отляво прозореца с един елемент, така че той да съдържа с един по-малко от всичките елементи на низа `p`.
`xezbxccxaa`czb`x`

7. Разширяваме прозореца надясно и спираме целия процес, защото достигаме края на низа `s`:
`xezbxccxaa`czbx Полученият прозорец не съдържа всички елементи на `p`.

В процеса на движението на прозореца намерихме два *минимални* подниза, съдържащи всички елементи на `p`. Първият е `bxccxa` с дължина 6, а вторият е `aczb` с дължина 4. Отговорът на теста е вторият подниз, защото е с по-малка дължина.

Зорница Дженкова