P-III-4 Sbírka čísel

Pojďme nejdříve zjistit, kolik čísel páně Pterodaktylova sbírka obnáší. Zvolme si pevný základ soustavy B a označme $n(d, s)$ počet všech čísel, která mají v naší soustavě zápis délky d (včetně případných nevýznamných nul na začátku) s ciferným součtem s. Jelikož ciferný součet 0 má pouze číslo ze samých nul, je $n(d, 0) = 1$ pro každé d. Naopak prázdné číslo má nutně nulový ciferný součet, takže $n(0, 0) = 1$ a $n(0, s) = 0$ pro $s > 0$. Pro $d > 0$, $s > 0$ pak platí následující vztah:

$$n(d, s) = \sum_{i=0}^{\min(B-1,s)} n(d-1, s-i), \qquad (*)$$

v němž sčítáme přes všechny možnosti, jak zvolit číslici i na prvním místě zápisu. Takto můžeme pohodlně spočítat libovolné $n(d, s)$: nejdříve spočítáme $n(0, 0)$ až $n(0, s)$, z nich pak $n(1, 0)$ až $n(1, s)$ atd.

Jelikož čísla délky D mají ciferný součet nejvýše $S = D(B-1)$, všech čísel ve sbírce je $n(D, 0)+n(D, 1)+\ldots+n(D, S)$. Takto jsme je dokonce spočítali v pořadí podle jejich ciferného součtu, takže pokud chceme k-té číslo sbírky, snadno nalezneme jeho ciferný součet s a zjistíme, kolikáté je s tímto součtem. Následně zjistíme podle formulky $(*)$ pro $n(D, s)$ první číslici i a kolikáté číslo délky $D-1$ se součtem $s-i$ hledáme. Stejným způsobem pak získáme zbývající cifry.

Všechna $n(d, s)$ pro $d \leq D$ a $s \leq S$ získáme pomocí naší formulky v čase $\mathcal{O}(DSB) = \mathcal{O}(D^2B^2)$. Samotné hledání pak obnáší sečíst S hodnot $n(d, s)$ a následně provést D iterací po B krocích. Vše tedy stihneme v čase $\mathcal{O}(D^2B^2)$, což je pro meze uvedené v zadání „na odmáčknutí." Paměťová složitost navrženého řešení je pak $O(DS) = O(D^2B)$.

Program se drží tohoto postupu, jedinou potenciální zradou by mohlo být, že čísla, se kterými pracujeme, se vejdou až do 64-bitového číselného typu. Také samozřejmě nesmíme vypisovat nuly na začátku čísla.

```
#include <stdio.h>

#define MAXB 10
#define MAXD 30
#define MAXS ((MAXB-1)*MAXD)
typedef long long int big;

int main(void)
{
  int B, D;
  big k;
  big n[MAXD+1][MAXS+1] = { 0 };
```

```
  FILE *fi = fopen("sbirka.in", "r");
  fscanf(fi, "%d%d%lld", &B, &D, &k);
  fclose(fi);

  // Spočítáme všechna n[d][s]
  n[0][0] = 1;
  int S = (B-1)*D;
  for (int d=1; d<=D; d++)
    for (int s=0; s<=S; s++)
        for (int x=0; x<B && x<=s; x++)
          n[d][s] += n[d-1][s-x];
  k--;

  // Najdeme správný součet
  int s = 0;
  while (s <= S && k >= n[D][s])
    k -= n[D][s], s++;

  // Dopočítáme číslice a rovnou je vypíšeme
  FILE *fo = fopen("sbirka.out", "w");
  int z = 1;                                  // Zatím samé nuly
  for (int d=D; d>1; d--)
    {
      int x = 0;                              // Hledáme číslici
      while (x < B-1 && n[d-1][s-x] <= k)
        {
          k -= n[d-1][s-x];
          x++;
        }
      if (x || !z)                            // Vypíšeme ji
        {
          fputc('0' + x, fo);
          z = 0;
        }
      s -= x;
    }
  fprintf(fo, "%d\n", (int)s);                // Poslední číslice
  fclose(fo);

  return 0;
}
```

P-III-5 Strouhání mrkve

Když vás Hopsálek najímal do továrny, vysvětlil vám, jak si představuje postup na hledání nejlepšího začátku první směny: „V poli si pro každý začátek první směny budete pamatovat, kolik směn je možné v tomto případě zrušit. Nakonec podle příchodu posledního zákazníka spočtete počet směn, který by byl třeba, kdyby nebyla žádná směna zrušená, odečtete (v poli uložený) počet zrušených směn a podle toho si vyberete takový začátek první směny, aby směn bylo co nejméně.“

Označme začátek první směny i_0. Víme, že i_0 může nabývat hodnot 1 až $M-1$. Pro všechny tyto hodnoty si v poli uložíme, kolik směn bychom mohli zrušit, kdyby

první směna začínala v i_0-té minutě. Na začátku jsou v poli samé nuly. Poté budeme zpracovávat postupně jednotlivé zákazníky od prvního do posledního. V každém kroku budeme zkoumat, kolik směn můžeme zrušit v pauze mezi příchodem aktuálního zákazníka a zákazníka před ním (tj. nejdříve mezi prvním a druhým, pak mezi druhým a třetím atd.).

Řekněme tedy, že aktuální zákazník přišel v čase b a zákazník před ním v čase a. Pro všechny možná i_0 bychom si chtěli napočítat, kolik můžeme v pauze mezi a a b zrušit směn, pokud první začínala v čase i_0. Na tom ale není nic těžkého – směna, ve které se zpracuje zákazník s příchodem a, musí začínat v čase $a-((a-i_0) \bmod M)$ a směna, ve které se zpracuje zákazník s příchodem b, musí začínat v čase $b-((b-i_0) \bmod M)$, takže můžeme zrušit

$$\left[b-((b-i_0) \bmod M)-(a-((a-i_0) \bmod M))\right]/M$$

směn. Takto upravíme počty zrušených směn v jednom kroku pro všechna možná i_0.

Když takto zpracujeme pauzy mezi všemi zákazníky, stačí už jenom pro každé i_0 spočítat počet směn, které bychom potřebovali pro zpracování všech zakázek, kdybychom směny nerušili, odečteme spočtený počet zrušených směn a máme skutečný počet směn v případě, že první začínala v čase i_0.

Takové řešení by jistě fungovalo, nicméně jeho časová složitost je $\mathcal{O}(NM)$, protože na zpracování jednoho zákazníka musíme upravit počty zrušených směn pro M možných začátků.

Takové řešení vás (a ani nás :-)) zajisté nemůže uspokojit, tak se pojďme zamyslet, jak bychom ho mohli vylepšit. Začněme s důležitým pozorováním. Řekněme, že zkoumáme pauzu mezi příchody v a a b minut. Nejvíc směn můžeme zrušit tehdy, když další směna začne hned v čase $a+1$ (to je v případě $i_0 = (a \bmod M)+1$). Když budeme i_0 zvyšovat, budeme moci chvíli rušit stejný počet směn, dokud nedojde k tomu, že poslední rušená směna poprvé nezasáhne do času b – pak budeme moci směn zrušit o jednu méně. Stejný počet směn (tj. o jedna méně než pro začátek i_0) budeme moci zrušit pro všechna zbylá i_0 až do i_0-1.

Uvědomili jsme si tedy, že v poli počtu zrušených směn chceme přidat jednomu souvislému úseku hodnot i_0 číslo $s+1$ a zbylému úseku číslo s. Zkusíme tedy najít jinou reprezentaci pole, která by nám umožnila provádět takovéhle úpravy v konstantním čase. Začneme tím, že si mimo pole budeme pamatovat číslo, které je třeba přičíst ke všem číslům v poli. Pak můžeme k tomuto číslu přičíst s a jediné, co bude třeba udělat, bude souvislému úseku v původním poli přičíst číslo jedna. Tento problém už vyřešíme lehce – v poli si pro různá i_0 nebudeme pamatovat skutečný počet zrušených směn, ale jenom rozdíl počtu zrušených směn oproti předchozímu prvku. Pak můžeme přičtení jedničky k souvislému úseku provést tak, že přičteme jedničku k počátečnímu prvku úseku a odečteme jedničku od prvku, který následuje za posledním prvkem úseku. Nakonec stačí naše pole rozdílů projít a spočítat skutečné hodnoty zrušených směn.

Naše vylepšené řešení má časovou složitost $\mathcal{O}(N+M)$ a paměťovou $\mathcal{O}(M)$, což je jistě pro vás i pana Hopsálka uspokojivé.

```
#include <stdio.h>
#define MAX 1000042

int M, N, zrusene_diff[MAX];
FILE *fr, *fw;

int main(void) {
  fr=fopen("mrkev.in", "r");
  fw=fopen("mrkev.out","w");

  fscanf(fr, "%d%d", &M, &N);
  int last=0, zrusene_celkem=0;
  for (int i=0; i<N; i++) {
    int next; fscanf(fr, "%d", &next);
    int length = next - last - 1;
    if (length >= M) {
      zrusene_celkem += (length/M)-1;
      length %= M;
      int s = last % M;
      if (s+length >= M) {
        zrusene_diff[s]++; zrusene_diff[M]--;  /* tohle je zbytečné */
        zrusene_diff[0]++; zrusene_diff[s+length-M+1]--;
      } else {
        zrusene_diff[s]++; zrusene_diff[s+length+1]--;
      }
    }
    last = next;
  }
  fclose(fr);

  int zrusene = 0, best = (1U<<31)-1;
  for (int i=0; i<M; i++) {
    zrusene += zrusene_diff[i];
    int curr = (last-i+M-1)/M - zrusene - zrusene_celkem;
    if (curr < best) best=curr;
  }
  fprintf(fw, "%d\n", best);
  zrusene = 0;
  for (int i=0, uz=0; i<M; i++) {
    zrusene += zrusene_diff[i];
    int curr = (last-i+M-1)/M - zrusene - zrusene_celkem;
    if (curr == best) uz++ && fputc(' ', fw), fprintf(fw, "%d", i+1);
  }
  fputc('\n', fw);
  fclose(fw);

  return 0;
}
```