P-III-1 Oříšková čokoláda

Přestože se hra může vyvíjet mnoha různými způsoby, celkový počet různých pozic je nejvýše $RS + 1$. Jedna pozice je snězená čokoláda. Jinak po každém kroku zbude z čokolády obdélník s levým horním rohem v políčku $(1, 1)$. Proto každé políčko reprezentuje pozici, ve kterém je ono nejpravějším a nejspodnějším políčkem dosud zbylého kusu čokolády.

Prázdná čokoláda a ty pozice, ze kterých neexistuje žádný povolený tah, jsou ze zadání prohrávajícími. Pro každou další pozici rozhodneme, zda je vyhrávající pro hráče na tahu, tak, že se podíváme, do kterých pozic se z ní lze jedním tahem dostat. Existuje-li z aktuální pozice tah do prohrávající pozice, je tato pozice vyhrávající. V opačném případě je prohrávající.

Zbývá pouze rozhodnout o tom, které tahy jsou povolené. Mějme pozici reprezentovanou políčkem (i, j). Řádek můžeme ulomit tehdy, je-li počet oříšků v i-tém řádku sudý. Analogická situace je pro sloupce. Přímočaré řešení tedy může postupně procházet políčka po řádcích a pro každé rozhodnout, v čase $\mathcal{O}(R + S)$, které tahy jsou z něj povolené. Následně vyhodnotí, zda je aktuální pozice vyhrávající, a pokračuje dále. Celková časová složitost uvedeného algoritmu je $\mathcal{O}(RS \cdot (R + S))$ a paměťová je optimální $\mathcal{O}(RS)$.

Pokusme se algoritmus vylepšit. Nebudeme v každém kroku zjišťovat, zda je součet čísel v řádku sudý, ale předpočítáme si tyto informace již na začátku. Podívejme se na čokoládu v pozici odpovídající políčku (i, j), kde j je alespoň 2, a pokusme se rozhodnout, zda je možné odlomit poslední řádek. Součet čísel v posledním řádku této čokolády je o $B_{i,j}$ větší než součet čísel v posledním řádku čokolády odpovídající pozici $(i, j - 1)$. Můžeme si tedy součty v posledních řádcích pamatovat a znovu je využívat. Pro všechny pozice tak můžeme spočítat součet v posledním řádku a analogicky také v posledním sloupci v čase $\mathcal{O}(RS)$. Společně s předchozím algoritmem dostáváme celkově optimální algoritmus s časovou i paměťovou složitostí $\mathcal{O}(RS)$.

```
#include <stdio.h>

#define MAX_R 1000
#define MAX_S 1000

int R, S;
int B[MAX_R][MAX_S];
int row_sum[MAX_R][MAX_S], col_sum[MAX_R][MAX_S];
int winning[MAX_R][MAX_S];

int main(void)
{
    scanf("%d%d", &R, &S);
```

```
    // Procházíme postupně všechny pozice (i, j) po řádcích.
    for (int i = 0; i < R; ++i) for (int j = 0; j < S; ++j) {
        scanf("%d", &B[i][j]);

        // Přepočítáme součty posledního řádku, resp. posledního sloupce.
        // Uchováváme pouze zbytek součtu po dělení dvěma.
        if (i == 0) col_sum[i][j] = B[i][j] % 2;
        else        col_sum[i][j] = (B[i][j] + col_sum[i - 1][j]) % 2;
        if (j == 0) row_sum[i][j] = B[i][j] % 2;
        else        row_sum[i][j] = (B[i][j] + row_sum[i][j - 1]) % 2;

        // Vyhodnotíme, zda je aktuální pozice vyhrávající pro hráče,
        // který je aktuálně na tahu. Stav je vyhrávající, pokud z něj
        // existuje povolený tah, který vede do prohrávajícího pozice.
        winning[i][j] = 0;
        if (row_sum[i][j] == 0 && (i == 0 || winning[i - 1][j] == 0)) {
          winning[i][j] = 1;
        }
        if (col_sum[i][j] == 0 && (j == 0 || winning[i][j - 1] == 0)) {
          winning[i][j] = 1;
        }
    }

    // Bobek s Bobkem vyhrají, pokud je (R, S) vyhrávající pozice
    if (winning[R - 1][S - 1]) printf("B\n");
    else                       printf("S\n");
    return 0;
}
```

P-III-2 Ztracené pakety

K řešení použijeme metodu dělení intervalu. Řekněme, že chceme vypsat chybějící pakety s čísly v intervalu $\langle a, b\rangle$ a v souboru S máme uložena čísla všech došlých paketů v tomto intervalu. Rozdělme si interval $\langle a, b\rangle$ na několik podintervalů $I_1 = \langle a, m_1\rangle$, $I_2 = \langle m_1 + 1, m_2\rangle$, …, $I_t = \langle m_{t-1} + 1, b\rangle$ zhruba stejné velikosti. Pro tyto podintervaly si pořídíme pomocné soubory $S_1, \ldots, S_t$ a každé číslo ze souboru S patřící do intervalu I_i (pro nějaké $i \in \{1, \ldots, t\}$) si překopírujeme do souboru S_i.

Zároveň si počítáme počet čísel n_i v tomto souboru. Zjevně pokud n_i je rovno velikosti intervalu I_i, pak v intervalu I_i nechybí žádný paket, a tyto intervaly proto můžeme ignorovat. Na ostatní intervaly se zavoláme rekurzivně. Pokud v rekurzi dospějeme k intervalu, který neobsahuje žádná čísla došlých paketů, pak všechna čísla v tomto intervalu chybí a vypíšeme je na výstup.

Na kolik podintervalů bychom měli interval dělit? Kdybychom zvolili $t = 2$ a interval půlili, může se nám v nejhorším případě stát, že nejprve budeme zpracovávat 1 interval velikosti n, na další úrovni rekurze 2 intervaly velikosti $n/2$, …, až na $(\log_2 k)$-té úrovni rekurze budeme zpracovávat k intervalů velikosti n/k. Na každé další úrovni již budeme zpracovávat k intervalů velikosti vždy poloviční než na předchozí úrovni, a tedy celkový počet zpracovaných čísel na každé další úrovni se vždy zmenší alespoň na polovinu. Celková délka zpracovávaných intervalů, která odpovídá počtu čísel čtených ze souborů, tedy bude

$$1 \cdot n + 2 \cdot \frac{n}{2} + 4 \cdot \frac{n}{4} + \ldots + k \cdot \frac{n}{k} + n/2 + n/4 + \ldots \leq n \log_2 k + n.$$

Vzhledem k tomu, že každé čtené číslo kromě těch ve vstupním souboru také zapíšeme, může počet čísel zapsaných a čtených ze souborů být dohromady až $2n \log_2 k$, což pro zadaná omezení na n a k může přesáhnout povolené omezení na počet operací se soubory.

Nicméně, s omezeními ze zadání můžeme interval dělit na $t = \max(2, k)$ dílů. V tomto případě budeme pro každé chybějící číslo uvažovat postupně intervaly délky n, n/k, n/k^2, ..., které ho obsahují. První interval $\langle 1, n \rangle$ je navíc pro všechna chybějící čísla stejný, a proto ho počítáme pouze jednou (stejné pozorování platí i pro kratší intervaly, které obsahují více než jedno chybějící číslo; žádné takové intervaly ale nemusí existovat, proto tuto redundanci v odhadu počtu operací se soubory ignorujeme). Celkový počet čtení je tedy nejvýše $n + k(n/k + n/k^2 + \ldots) \leq 3n$, a počet zápisů je nejvýše $2n$. Počet operací se soubory tedy nejvýše $5n$, což odpovídá omezení ze zadání.

Časová složitost algoritmu je tak zjevně $\mathcal{O}(n)$. Naivní rekurzivní implementace popsaného algoritmu by měla paměťovou složitost $\mathcal{O}(k \log n)$; povšimněme si ale, že si místo rekurze můžeme jednoduše udržovat seznam nejvýše k ještě nezpracovaných intervalů (z nichž každý obsahuje alespoň jedno číslo chybějícího paketu) a postupně ho procházet. Tím se paměťová složitost sníží na $\mathcal{O}(k)$.

```
#include <stdio.h>
#include <stdlib.h>

#define MAXK 100

/*
 *  Správa pomocných souborů, aby bylo zjevné, že jich řešení
 *  nepoužívá více než 2k+2. Praktičtější by bylo použít tmpfile().
 */
static unsigned num_tmp_files;
static FILE *file_pool[2 * MAXK];
static unsigned file_pool_size;

static FILE *
get_temporary_file(void)
{
  FILE *ret;

  if (file_pool_size > 0)
    {
      ret = file_pool[--file_pool_size];
      rewind(ret);
    }
  else
    {
      char tmpname[10];
      sprintf(tmpname, "tmp%d", num_tmp_files++);
      ret = fopen(tmpname, "w+");
    }
  return ret;
}

static void
release_tmp_file(FILE *f)
```

```
{
  file_pool[file_pool_size++] = f;
}

static void
remove_temporary_files(void)
{
  for (unsigned i = 0; i < num_tmp_files; i++)
    {
      char tmpname[10];
      sprintf(tmpname, "tmp%d", i);
      remove(tmpname);
    }
}

/*
 *  Seznam intervalů <f,t> ke zpracování. Interval obsahuje c čísel
 *  paketů a tato čísla jsou uložena v souboru s.
 */
typedef struct
{
  unsigned f, t, c;
  FILE *s;
} interval;

static interval ke_zpracovani[MAXK];
static unsigned int_ke_zprac;

/*
 *  Zpracuj interval <f,t>, v němž je c čísel, uložených v souboru s.
 *  Interval budeme dělit na d podintervalů.
 */
static void
zpracuj_interval(unsigned f, unsigned t, unsigned c, FILE *s, unsigned d)
{
  FILE *tmpfs[MAXK];
  unsigned pocet[MAXK];
  unsigned m = t - f + 1;
  unsigned i, plen, a, ai;

  for (i = 0; i < d; i++)
    pocet[i] = 0;

  /*
   *  Jestliže je délka intervalu <f,t> větší než d, přípravme pomocné
   *  soubory. Jinak na této úrovni rekurze končíme s intervaly délky 1.
   */
  if (m > d)
    {
      for (i = 0; i < d; i++)
        tmpfs[i] = get_temporary_file();
      plen = (m + d - 1) / d;
    }
  else
    plen = 1;

  /* Rozdělme čísla paketů do podintervalů. */
  for (i = 0; i < c; i++)
```

```
    {
      fscanf(s, "%u", &a);
      ai = (a - f) / plen;

      pocet[ai]++;
      if (m > d)
        fprintf(tmpfs[ai], "%d\n", a);
    }

  /*
   *  Intervaly, které obsahují alespoň jedno chybějící číslo,
   *  zařadíme do sezamu ke zpracování (nebo toto chybějící číslo
   *  vypíšeme, pokud je celý interval prázdný).
   */
  for (i = 0; i < d; i++)
    {
      unsigned l = f + plen * i, u = l + plen - 1;
      interval nint;

      if (u > t)
        u = t;

      if (pocet[i] == u - l + 1)
        {
          if (m > d)
            release_tmp_file(tmpfs[i]);
          continue;
        }

      if (pocet[i] == 0)
        {
          unsigned j;

          for (j = l; j <= u; j++)
            printf("%d ", j);
          if (m > d)
            release_tmp_file(tmpfs[i]);
          continue;
        }

      nint.f = l;
      nint.t = u;
      nint.c = pocet[i];
      rewind(tmpfs[i]);
      nint.s = tmpfs[i];
      ke_zpracovani[int_ke_zprac++] = nint;
    }
}

int main(void)
{
  FILE *fin = fopen("pakety.txt", "r");
  unsigned n, k, d;

  fscanf(fin, "%u%u", &n, &k);
  d = k < 2 ? 2 : k;

  int_ke_zprac = 0;
  zpracuj_interval(1, n, n - k, fin, d);
  while (int_ke_zprac > 0)
```

```
    {
      interval a = ke_zpracovani[--int_ke_zprac];
      zpracuj_interval(a.f, a.t, a.c, a.s, d);
      release_tmp_file(a.s);
    }
  remove_temporary_files();
  return 0;
}
```

P-III-3 Magická síť

Úkol 1: Řešením je například síť

$$\begin{aligned}
&\text{ONE-IN-THREE}(y, z, p)\\
&\text{ONE-IN-THREE}(n, n, j)\\
&\text{ONE-IN-THREE}(x, x', n)\\
&\text{ONE-IN-THREE}(y, y', n)\\
&\text{ONE-IN-THREE}(x', z', p').
\end{aligned}$$

Omezení ONE-IN-THREE(y, z, p) lze volbou hodnoty pro p splnit, právě když alespoň jedna z proměnných y a z má hodnotu 0. Všimněme si, že ONE-IN-THREE(n, n, j) vynucuje, že proměnná n má hodnotu 0. Proto kombinace ONE-IN-THREE(x, x', n) a ONE-IN-THREE(y, y', n) vynucuje, že $x' = 1 - x$ a $y' = 1 - y$. Konečně omezení ONE-IN-THREE(x', y', p') lze volbou hodnoty pro p' splnit, právě když alespoň jedna z proměnných x' a y' má hodnotu 0, což je ekvivalentní tomu, že alespoň jedna z proměnných x a y má hodnotu 1.

Úkol 2: Tuto úlohu vyřešíme obdobně jako příklad 3 ze studijního textu. Připomeňme, že *maj* je funkce se třemi vstupy, která vrací ten vstup, který se vyskytuje nejčastěji. Nechť (a_1, b_1, c_1), (a_2, b_2, c_2) a (a_3, b_3, c_3) jsou tři trojice vstupů, které splňují omezení CH. Pak alespoň jedna z hodnot a_1 a b_1 je rovna 1, a tedy $a_1 + b_1 \geq 1$. Obdobně $a_2 + b_2 \geq 1$ a $a_3 + b_3 \geq 1$.

Dostáváme tedy $3 \leq (a_1+b_1)+(a_2+b_2)+(a_3+b_3) = (a_1+a_2+a_3)+(b_1+b_2+b_3)$. Proto buď alespoň dvě z hodnot a_1, a_2 a a_3 jsou rovny 1, nebo alespoň dvě z hodnot b_1, b_2 a b_3 jsou rovny 1. Ekvivalentně, alespoň jedna z hodnot $maj(a_1, a_2, a_3)$ a $maj(b_1, b_2, b_3)$ je rovna 1. Obdobně, alespoň jedna z hodnot $maj(b_1, b_2, b_3)$ a $maj(c_1, c_2, c_3)$ je rovna 0. Proto hodnoty

$$(maj(a_1, a_2, a_3), maj(b_1, b_2, b_3), maj(c_1, c_2, c_3))$$

splňují omezení CH.

Nechť $S(x, y, z)$ je libovolná síť sestavená pouze z omezení CH. Dle předchozího odstavce pro libovolná tři přiřazení P_1, P_2 a P_3 hodnot proměnným, která splňují všechna omezení v síti S, přiřazení vzniklé z P_1, P_2 a P_3 zkombinováním funkcí *maj* také splňuje všechna omezení v síti S. Speciálně, jestliže síť S přijímá trojice vstupů (x_1, y_1, z_1), (x_2, y_2, z_2) a (x_3, y_3, z_3), pak také přijímá trojici

$$(maj(x_1, x_2, x_3), maj(y_1, y_2, y_3), maj(z_1, z_2, z_3)).$$

Ovšem omezení ONE-IN-THREE je splněno trojicemi hodnot $(1,0,0)$, $(0,1,0)$, $(0,0,1)$, ale není splněno jejich kombinací pomocí funkce *maj*, která je rovná $(0,0,0)$. Proto žádná síť sestavená pouze z omezení CH nesimuluje ONE-IN-THREE.

Úkol 3: Připomeňme, že omezení $S_{h_1h_2\ldots h_n}$ ze studijního textu zakazuje právě jednu kombinaci hodnot svých n vstupů, a to konkrétně kombinaci $(h_1, h_2, \ldots, h_n)$. Nechť Q je síť s proměnnými $x_1, \ldots, x_n$ skládající se právě z omezení $S_{h_1h_2\ldots h_n}(x_1, \ldots, x_n)$ takových, že omezení O není splněno pro vstup $(h_1, \ldots, h_n)$. Tato síť Q simuluje omezení O. Proto stačí ukázat, že s použitím omezení ONE-IN-THREE lze simulovat libovolné omezení $S_{h_1h_2\ldots h_n}$.

Nechť $i_1, i_2, \ldots, i_k$ jsou indexy takové, že $h_{i_1} = h_{i_2} = \ldots = h_{i_k} = 0$, a všechny ostatní hodnoty z $h_1, \ldots, h_n$ jsou rovny 1. Uvažme síť

$$\begin{aligned}
&\text{ONE-IN-THREE}(p, p, j)\\
&\text{ONE-IN-THREE}(x_{i_1}, x'_{i_1}, p)\\
&\text{ONE-IN-THREE}(x_{i_2}, x'_{i_2}, p)\\
&\ldots\\
&\text{ONE-IN-THREE}(x_{i_k}, x'_{i_k}, p)\\
&S_{11\ldots1}(x'_1, \ldots, x'_n)
\end{aligned}$$

kde x'_i je rovno x_i jestliže $i \notin \{i_1, i_2, \ldots, i_k\}$. Jako v první úloze si rozmyslíme, že tato sít vynucuje $x'_{i_1} = 1 - x'_{i_1}$, $\ldots$, $x'_{i_k} = 1 - x'_{i_k}$, a proto simuluje omezení $S_{h_1h_2\ldots h_n}(x_1, \ldots, x_n)$.

Stačí tedy ukázat, že s použitím omezení ONE-IN-THREE lze simulovat omezení $S_{11\ldots1}(x_1, \ldots, x_n)$ pro každé přirozené číslo n. Pro $n = 1$ je omezení $S_1(x_1)$ simulováno sítí ONE-IN-THREE(x_1, x_1, j), vynucující že $x_1 = 0$. Pro $n = 2$ je $S_{11}(x_1, x_2)$ simulováno sítí ONE-IN-THREE(x_1, x_2, z). Pro $n = 3$ je $S_{111}(x_1, x_2, x_3)$ simulováno sítí $S_{11}(x_1, x'_1)$, $S_{11}(x_2, x'_2)$, $S_{11}(x_3, x'_3)$, ONE-IN-THREE(x'_1, x'_2, x'_3): díky omezení ONE-IN-THREE(x'_1, x'_2, x'_3) je $x'_i = 1$ pro nějaké $i \in \{1, 2, 3\}$, a proto $x_i = 0$.

Nechť $n \geq 4$, a předpokládejme, že už jsme zkonstruovali síť simulující omezení $S_{11\ldots1}(x_1, \ldots, x_{n-1})$ s $n - 1$ vstupy. Uvažme síť

$$\begin{aligned}
&S_{11\ldots1}(x_1, \ldots, x_{n-2}, z)\\
&S_{111}(x_{n-1}, x_n, z')\\
&\text{ONE-IN-THREE}(p, p, j)\\
&\text{ONE-IN-THREE}(z, z', p).
\end{aligned}$$

Tato síť vynucuje, že $z' = 1 - z$. Je-li alespoň jedna z proměnných x_{n-1} a x_n rovna 0, pak můžeme zvolit $z' = 1$ a $z = 0$ a $S_{11\ldots1}(x_1, \ldots, x_{n-2}, z)$ je splněno. Jestliže $x_{n-1} = x_n = 1$, pak omezení $S_{111}(x_{n-1}, x_n, z')$ vynucuje $z' = 0$, a tedy $z = 1$. Omezení $S_{11\ldots1}(x_1, \ldots, x_{n-2}, z)$ pak vynutí, že alespoň jedna z proměnných $x_1, \ldots, x_{n-2}$ je rovna 0.

Tato síť tedy vynucuje, že alespoň jedna z proměnných $x_1, \ldots, x_n$ je rovna 0, a proto simuluje omezení $S_{11\ldots1}(x_1, \ldots, x_n)$. Takto můžeme postupně sestrojit všechna požadovaná omezení.