## P-III-4 Odpory

Pro zjednodušení popisu řešení uvažujme pouze případ, kdy $X$ je součtem čtyř čísel ze vstupu; případy, kdy je součtem nejvýše tří se řeší obdobně. Nechť čísla na vstupu jsou $a_1, \ldots, a_N$. Pro $i = 0, \ldots, N$ jako $S_i$ označme množinu všech součtů dvojic různých čísel z množiny $\{a_1, a_2, \ldots, a_i\}$. Pak úloha má řešení, jestliže existují nějaké indexy $i < j$ takové, že $X - a_i - a_j$ patří do množiny $S_{i-1}$.

Náš algoritmus tedy bude fungovat takto: na začátku máme $S_0 = \emptyset$. Pro $i = 1, \ldots, N-1$ provedeme následující:

- Pro $j = i+1, \ldots, N$ zkontrolujeme, zda $X - a_i - a_j$ patří do množiny $S_{i-1}$.
- Vytvoříme množinu $S_i$ tak, že do množiny $S_{i-1}$ přidáme všechny součty $a_i + a_k$ pro $k = 1, \ldots, i-1$.

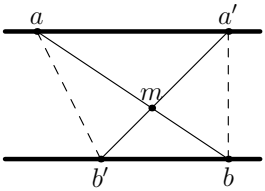
V každé iteraci tohoto cyklu provedeme $N$ operací s množinami (test existence prvku, přidání prvku), časová složitost tedy bude úměrná $N^2$ krát složitost množinové operace. Povšimněme si, že velikost každé množiny $S_i$ bude nejvýše $N^2$. Množinu si můžeme reprezentovat jako vyhledávací strom, v tom případě bude složitost každé operace $\mathcal{O}(\log N^2) = \mathcal{O}(\log N)$ a časová složitost celého algoritmu bude $\mathcal{O}(N^2 \log N)$. Nebo si množinu můžeme reprezentovat jako hashovací tabulku, v tom případě bude složitost každé operace konstantní a časová složitost celého algoritmu bude $\mathcal{O}(N^2)$, ale pouze v průměrném případě. Paměti spotřebujeme v obou případech $\mathcal{O}(N^2)$ buněk.

```
#include <cstdio>
#include <map>
#include <vector>

using namespace std;

int main(void)
{
  vector<unsigned> a;
  /* Je-li v aktuální množině součet x+y=z, pak do soucty[z] uložíme x nebo y.  */
  map<unsigned, unsigned> soucty;
  unsigned i, j, n, x, y;

  /* Načteme zadání.  */
  scanf("%u %u", &x, &n);
  for (i = 0; i < n; i++)
    {
      scanf("%u", &y);
      a.push_back(y);
    }

  for (i = 0; i < n; i++)
    {
      /* Je-li X rovno jednomu ze vstupních čísel, prostě ho vypíšeme.  */
```

```
      if (a[i] == x)
        {
          printf("%u\n", x);
          return 0;
        }

      if (a[i] > x)
        continue;

      /* Je-li X rovno součtu a[i] a dalších dvou čísel, vypíšeme tato tři čísla.  */
      if (soucty.count(x - a[i]) > 0)
        {
          unsigned b = soucty[x - a[i]];

          printf("%u %u %u\n", x - b - a[i], b, a[i]);
          return 0;
        }

      /* Ověříme, zda X je rovno součtu a[i], a[j] a dalších dvou čísel.  */
      for (j = i + 1; j < n; j++)
        {
          if (a[i] + a[j] > x)
            continue;

          if (soucty.count(x - a[i] - a[j]) > 0)
            {
              unsigned b = soucty[x - a[i] - a[j]];

              printf("%u %u %u %u\n", x - b - a[i] - a[j], b, a[i], a[j]);
              return 0;
            }
        }

      /* Upravíme množinu soucty přidáním součtů dvojic obsahujících a[i].  */
      for (j = 0; j < i; j++)
        {
          unsigned s = a[j] + a[i];

          if (s > x)
            continue;
          /* Ověříme, zda x je rovno součtu dvou čísel.  */
          if (s == x)
            {
              printf("%u %u\n", a[j], a[i]);
              return 0;
            }

          soucty[s] = a[i];
        }
    }

  printf("Konec sveta se odklada\n");
  return 0;
}
```

**P-III-5 Kocourkov**

Řešení začneme pozorováním, že v optimálním řešeni se trasy žádných dvou přívozů nekříží. Pokud by se trasy přívozů mezi dvojicí míst $a$ a $b$ a dvojicí $a'$ a $b'$ křížily, pak by záměnou těchto přívozů na přívozy spojující dvojici $a$ a $b'$ a dvojici $a'$ a $b$ vzniklo řešeni, kde součet délek tras všech přívozů je menší.




$$\begin{aligned}|ab| + |a'b'| &= (|am| + |mb|) + (|a'm| + |mb'|) \\ &= (|am| + |mb'|) + (|a'm| + |mb'|) > |ab'| + |a'b|\end{aligned}$$

Toto pozorování nás dovede k následujícímu. Označme $\mathrm{OPT}(i, j)$ nejmenší možný součet délek tras přívozů, které spojují prvních $i$ míst na levé straně kanálu a prvních $j$ míst na jeho pravé straně. Pokud $i \geq 2$ a $j \geq 2$, pak v libovolném optimálním řešení jsou $i$-té a $j$-té místo spojeny přívozem a přívozem je spojena právě jedna z následujících tří dvojic míst: $(i-1)$-ní místo a $j$-té místo, $i$-té místo a $(j-1)$-ní místo nebo $(i-1)$-ní místo a $(j-1)$-ní místo. Tedy platí následující:

$$\mathrm{OPT}(i, j) = d(i, j) + \min\left(\mathrm{OPT}(i-1, j), \mathrm{OPT}(i, j-1), \mathrm{OPT}(i-1, j-1)\right),$$

kde $d(i, j)$ je vzdálenost $i$-tého místa na levé straně a $j$-tého místa na pravé straně. Společně s $\mathrm{OPT}(1, 1) = d(1, 1)$, $\mathrm{OPT}(i, 1) = d(i, 1) + \mathrm{OPT}(i-1, 1)$ a $\mathrm{OPT}(1, j) = d(1, j) + \mathrm{OPT}(1, j-1)$ tak máme rekurzivní předpis pro výpočet všech hodnot $\mathrm{OPT}(i, j)$, $1 \leq i \leq a$ a $1 \leq j \leq b$.

Na chvíli předpokládejme, že paměťový limit umožňuje uložení pole čísel o velikosti $A \times B$. Kvůli nutnosti vypsat optimální řešeni bychom kromě pole OPT také potřebovali pole REK, které by obsahovalo indikaci, která ze tří výše uvedených možností nastala. Takto navržené řešení má časovou i paměťovou složitost $\mathcal{O}(AB)$ a získá zhruba 10 bodů z 15 možných.

Paměťové omezení úlohy ale neumožňuje vytvoření pole čísel o velikosti $A \times B$. Nejprve si uvědomme, že k výpočtu hodnot v poli OPT postačí paměť velikosti $\mathcal{O}(B)$: hodnoty $\mathrm{OPT}(i, j)$ pro pevné $i$ a $j$ mezi 1 a $B$ lze spočítat z hodnot $\mathrm{OPT}(i-1, j)$ a tedy potřebujeme uchovávat v paměti pouze dva řádky tohoto pole. K uložení pole REK pak použijeme následující trik. Do jedné 8-bitové (bytové) proměnné uložíme 4 hodnoty pole REK, každou hodnotu do 2 bitů. Tím potřebujeme k uložení pole REK nejvýše $(20\,000)^2/4 = 100 \cdot 10^6$ bytů paměti. Takové množství paměti však k dispozici máme.

```
#include <stdio.h>
#include <math.h>
#define MAX 20100
```

```
double L;   // délka kanálu
double S;   // šířka kanálu
int A,B;    // počet vybraných míst na levé a pravé straně kanálu
double a[MAX],b[MAX];
            // umístění vybraných míst na levé a pravé straně kanálu
unsigned char optimalni_reseni[MAX][MAX/4];
            // bitové pole pro uložení optimálního řešení
            // následují makra pro práci s tímto polem
#define prirad(pole,pozice,hodnota) pole[(pozice)/4] |= (hodnota) << (2*((pozice)%4))
#define vrat(pole,pozice) ((pole[(pozice)/4] >> (2*((pozice)%4))) & 3)

double spocitej_delku(double x, double y)
{
    // vypočte délku přívozu mezi místy ve vzdálenostech x a y na stranách kanálu
    return sqrt((x-y)*(x-y) + S*S);
}

void nacti_zadani(void)
{
    scanf("%lf %lf %d %d", &L, &S, &A, &B);
    for (int i=0; i<A; i++) scanf("%lf", &a[i]);
    for (int j=0; j<B; j++) scanf("%lf", &b[j]);
}

void najdi_reseni(void)o
{
    double soucet_delek[MAX];
            // pomocné pole pro dynamické programování
            // j-tá hodnota obsahuje součet délek v optimálním řešení pro i a j
    double stara_predchozi_hodnota;
            // pomocná proměnná pro dynamické programování
            // hodnota pro i-1 a j-1 - tímto se vyhneme manipulaci s dvěma poli
    double varianta1,varianta2,varianta3;
            // pomocné proměnné pro porovnání tří variant napojení
            // čísla variant odpovídají jejich bitovým maskám

    // inicializace polí soucet_delek a optimalni_reseni[0][*]
    soucet_delek[0] = spocitej_delku(a[0], b[0]);
    prirad(optimalni_reseni[0], 0, 3);
    for (int j=1; j<B; j++) {
        prirad(optimalni_reseni[0], j, 2);
        soucet_delek[j] = soucet_delek[j-1] + spocitej_delku(a[0], b[j]);
    }

    // výpočet hodnot polí soucet_delek a optimalni_reseni
    for (int i=1; i<A; i++) {
        stara_predchozi_hodnota = soucet_delek[0];
        soucet_delek[0] = soucet_delek[0] + spocitej_delku(a[i], b[0]);
        prirad(optimalni_reseni[i], 0, 1);
        for (int j=1; j<B; j++) {
            varianta1 = soucet_delek[j];         // místa i-1 a j spojena přívozem
            varianta2 = soucet_delek[j-1];       // místa i a j-1 spojena přívozem
            varianta3 = stara_predchozi_hodnota; // místa i-1 a j-1 spojena přívozem
            stara_predchozi_hodnota = soucet_delek[j];

            // porovnání trí možných variant řešení
            if (varianta1 < varianta2 && varianta1 < varianta3) {
                soucet_delek[j] = varianta1 + spocitej_delku(a[i], b[j]);
```

```
                prirad(optimalni_reseni[i], j, 1);
                continue;
            }
            if (varianta2 < varianta3) {
                soucet_delek[j] = varianta2 + spocitej_delku(a[i], b[j]);
                prirad(optimalni_reseni[i], j, 2);
                continue;
            }
            soucet_delek[j] = varianta3 + spocitej_delku(a[i], b[j]);
            prirad(optimalni_reseni[i], j, 3);
        }
    }
}

void vytiskni_reseni(int dvojic, int leva, int prava)
{
    // procedura pro vytištení řešení
    if (leva == -1) {
        printf("%d\n", dvojic);
        return;
    }

    switch (vrat(optimalni_reseni[leva], prava)) {
      case 1:
        vytiskni_reseni(dvojic+1, leva-1, prava);
        break;
      case 2:
        vytiskni_reseni(dvojic+1, leva, prava-1);
        break;
      case 3:
        vytiskni_reseni(dvojic+1, leva-1, prava-1);
        break;
    }

    printf("%d %d\n", leva+1, prava+1);
}

int main(void)
{
    nacti_zadani();
    najdi_reseni();
    vytiskni_reseni(0, A-1, B-1);
    return 0;
}
```