V prezentaci by mělo být zřetelných
7 dále uvedených bodů.

[1.] ZADÁNÍ neboli pohádka, stručné neformální seznámení, zdroj úlohy.

[2.] PŘÍKLAD, UKÁZKA, OBRÁZEK Na příkladu ukážeme, „jak to funguje“ nebo alespoň, čeho chceme dosáhnout.

[3.] NÁPAD, ROZBOR pro snadné algoritmy nepovinný, jinak ale nejdůležitější část prezentace.

[4.] MATEMATICKÁ FORMULACE s využitím matematického jazyka (množina, úsečka, dělitelnost, graf, existuje, mohutnost, permutace, funkce atd.),

[5.] ALGORITMUS s využitím [4.], stačí slovně ale pořádně, paměťová a časová složitost, když ne přesně, tak alespoň odhad.

[6.] IMPLEMENTACE nemusí být celá, stačí ukázat nebo říci, co dělají nejdůležitější funkce, hlavní datové struktury, vtipné triky v kódu.

[7.] POZNÁMKY, ZKUŠENOSTI Co se vám dařilo a nedařilo, co vás překvapilo, na co si dát pozor, případně odkazy, další souvislosti apod.

**Jednotlivé body [1.] - [7.] se mohou všelijak prolínat,
jak to logika postupu řešení vyžaduje,
neměly by ale v prezentaci chybět.**

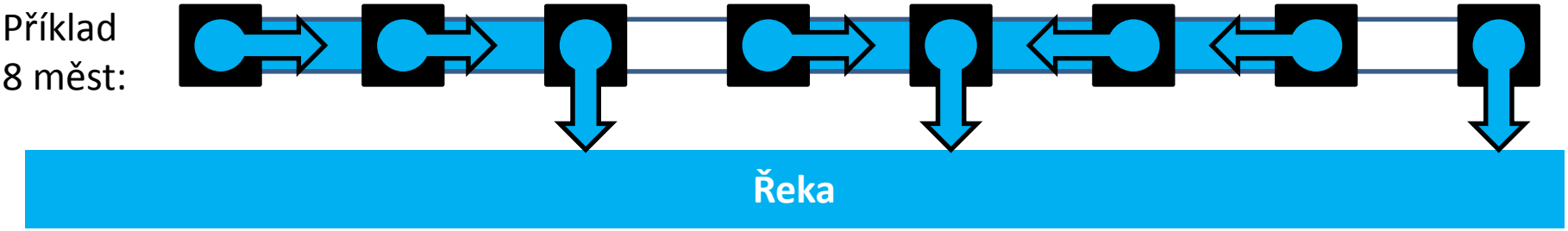
**Následující ukázka
je trochu více "ukecaná",
protože existuje jen v psané podobě.
Realistická prezentace bude spíš obsahovat
méně psaného textu a zbytek prezentující řekne.**

[1.] ZADÁNÍ

Na břehu řeky je n měst (n ≤ 100) propojených jednou rourou.
Každé město vypouští svou odpadní vodu buď přímo do řeky nebo rourou do sousedního města. Vypouštěcí strategie je pro každé město dlouhodobě fixní. Město nemůže vypouštět vodu do řeky i do roury zároveň.
Když do města dotéká voda z jednoho směru ze sousedního města nebo sousedních měst, může ji město buď spolu se svou vodou vypouštět do řeky nebo poslat spolu se svou vodou ve stejném směrem do sousedního města, pokud to jde.
Dvě sousední města nemohou vypouštět svou vodu do roury proti sobě. Kapacita roury je dostatečná pro libovolné množství vody.

Úloha: Zjistěte, kolik je možných konfigurací pro danou hodnotu n (max 100) .

Příklad
8 měst:



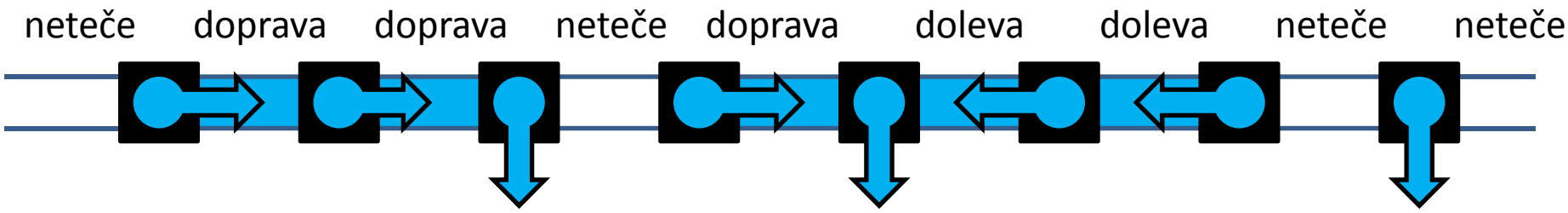
Úloha evropského jihozápadního regionálního kola ICPC 2002.

https://icpcarchive.ecs.baylor.edu/index.php?option=com_onlinejudge&Itemid=8&category=131&page=show_problem&problem=686

[2.] PŘÍKLAD

Když jsou města 4, máme celkem 21 možných konfigurací.

[3.] NÁPAD, ROZBOR



Situace v každém městě je dána směrem toku vpravo a vlevo bezprostředně od něj. U krajních měst si představujme, že se dále od nich táhne prázdná roura, v níž voda povinně neteče (nebo do města přitéká, to je jedno).

Když tedy postupujeme doleva doprava a koukáme jenom na směry toku mezi sousedními městy, jsme schopni celou konfiguraci jednoznačně rekonstruovat.

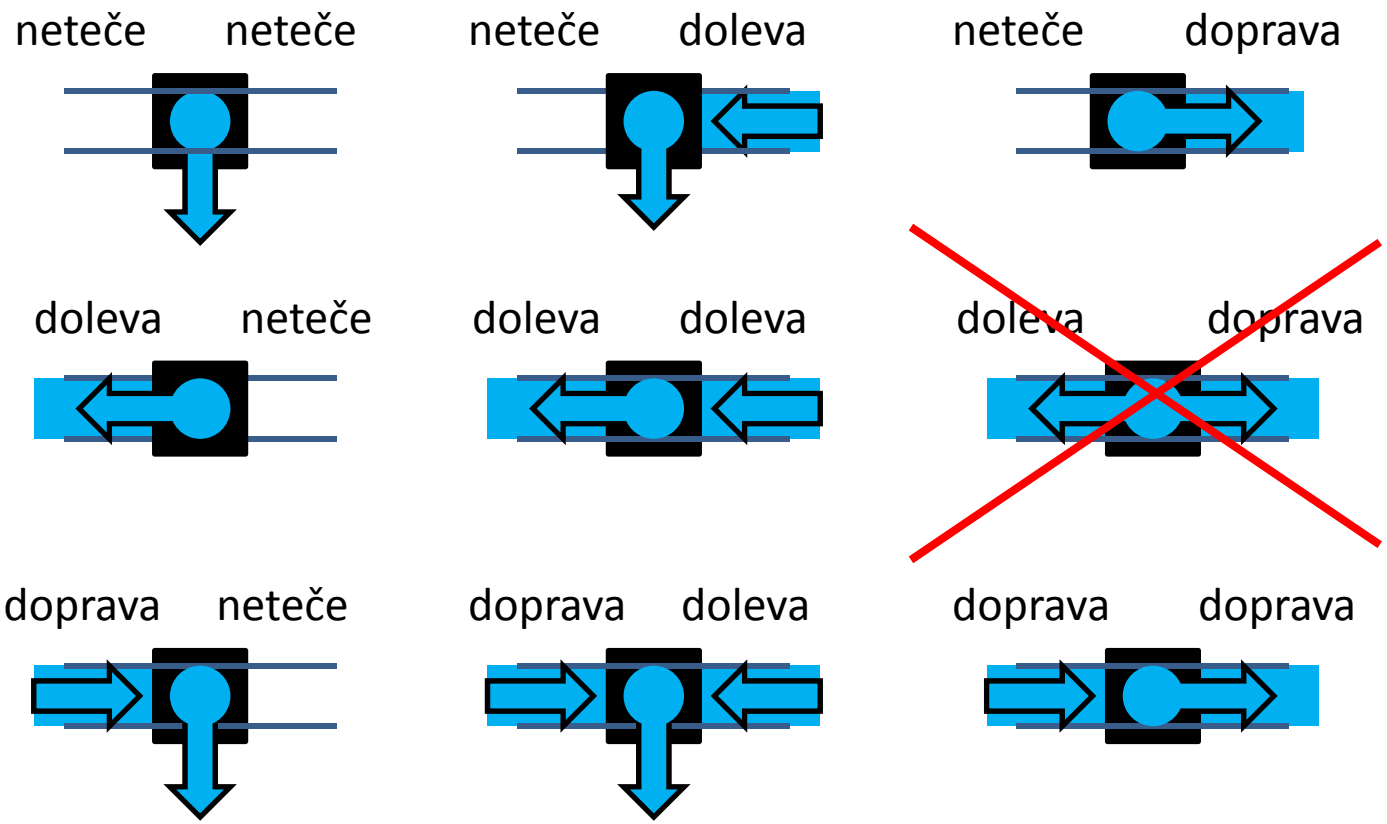
Tím můžeme vyloučit města z celé úlohy a soustředit se jen na jednotlivé toky.

Zdá se, že každá posloupnost toků je možná?

Není, bezprostředně vpravo od toku **doleva** se nemůže vyskytnout tok **doprava**, to by město mezi nimi vypouštělo vodu do roury oběma směry, a to se dle zadání nesmí. Jinak je zřejmě možná každá z osmi dalších kombinací sousedních směrů.

[3.] ROZBOR pokračování

Tabulka lokálních možností



[4.] MATEMATICKÁ FORMULACE

Označme jednotlivé směry:

neteče	X
doleva	L
doprava	R

Každá konfigurace je jednoznačně popsaná nějakým řetězcem těchto písmen. Podle tabulky na předchozí stránce může řetězec obsahovat libovolné dvojice sousedních znaků, kromě dvojice LR.

Okrajové přidané prázdné toky u obou krajních měst můžeme zanedbat a ekvivalentní formulace celé úlohy tedy bude :

Pro dané **n** najděte počet všech řetězců délky **n**-1 nad abecedou {X, L, R}, které neobsahují podřetězec "LR".

Na počty řetězců s různými omezeními se mnohdy hodí DP.

[3.] ROZBOR pokračování

Metoda DP typicky k nalezení počtu řetězců délky k+1 použije řetězce délky k.

Označme X(k) počet řetězců délky k končících písmenem X a analogicky
L(k) počet řetězců délky k končících písmenem L,
R(k) počet řetězců délky k končících písmenem R.

Končí-li řetězec libovolným z písmenem X, L, R,
lze jej prodloužit o jeden znak písmenem X, tudíž

$$X(k+1) = X(k) + R(k) + L(k).$$

Podobnou úvahou je zřejmé, že

$$L(k+1) = X(k) + R(k) + L(k).$$

Pouze když řetězec končí písmenem X nebo R,
lze jej prodloužit o jeden znak písmenem R, (dvojice LR je nepřípustná), tudíž

$$R(k+1) = X(k) + R(k).$$

Platí také X(1) = R(1) = L(1) = 1. Máme kompletní rekurentní vztahy a mohli bychom podle nich začít počítat . Řešením úlohy je pro n měst je tedy hodnota

$$X(n-1) + R(n-1) + L(n-1).$$

Tu spočteme v čase $\Theta(n)$, pokud dokážeme sčítat dvě čísla v konstantním čase. Dokážeme?

[3.] ROZBOR zlepšení

Odvozené rekurentní vztahy jsou si hodně podobné, levé strany jsou skoro stejné.

$$X(k+1) = X(k) + R(k) + L(k),$$
$$L(k+1) = X(k) + R(k) + L(k),$$
$$R(k+1) = X(k) + R(k).$$

Označme XR(k) počet řetězců délky k končících písmenem X nebo R.
Máme tedy $XR(k) = X(k) + R(k)$.

Nyní druhou rovnici ponecháme, sečteme první a třetí a dostaneme

$$L(k+1) = X(k) + R(k) + L(k),$$
$$X(k+1) + R(k+1) = 2(X(k) + R(k)) + L(k).$$

Podle definice XR(k) můžeme tyto dvě rovnice přepsat na

$$L(k+1) = XR(k) + L(k),$$
$$XR(k+1) = 2 \cdot XR(k) + L(k).$$

Platí také XR(1) = 2, L(1) = 1. Máme kompletní rekurentní vztahy a mohli bychom začít počítat . Řešením úlohy je pro n měst je tedy hodnota

$$XR(n-1) + L(n-1).$$

Tu spočteme v čase $\Theta(n)$, pokud dokážeme sčítat dvě čísla v konstantním čase. Dokážeme?

[3.] ROZBOR zlepšení

Máme

$$L(k+1) = XR(k) + L(k),$$
$$XR(k+1) = 2 \cdot XR(k) + L(k),$$
$$L(1) = 1,$$
$$XR(1) = 2.$$

Vyplňme si pro začátek ručně tabulku:

	k = 1	k = 2	k = 3	k = 4	k = 5	k = 6	k = 7
L(k)	1	3	8	21	55	144	377
XR(k)	2	5	13	34	89	233	610
L(k)+XR(k)	3	8	21	55	144	377	987

Čísla L(1), XR(1), L(2), XR(2), L(3), XR(3), L(4), XR(4), ... představují Fibonacciho posloupnost
1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... (důkaz možný indukcí apod.),
součet L(k)+XR(k) je součet dvou po sobě jdoucích Fibonacciho čísel, čili je to také Fibonacciho číslo.

[3.] ROZBOR zlepšení

Pro výpočet si rovnice

$$L(k+1) = XR(k) + L(k),$$
$$XR(k+1) = 2 \cdot XR(k) + L(k).$$

upravíme na:

$$L(k+1) = XR(k) + L(k),$$
$$XR(k+1) = XR(k) + XR(k) + L(k).$$

tj:

$$L(k+1) = XR(k) + L(k),$$
$$XR(k+1) = XR(k) + L(k+1).$$

Když budeme uchovávat L(k) v proměnné L a XR(k) v proměnné XR,
můžeme psát celý výpočet s rostoucím k:

```
int L = 1, XR = 2;
for(int k = 1; k < n; k++) {  // n = no of cities
    L += XR;
    XR += L;
}
print("%d\n", L);  // final result
```

Bude funkční?

[3.] ROZBOR zlepšení

Dosadíme za n maximální vstupní hodnotu uvedenou v zadání úlohy, tj 100. a program vytiskne -552082539, evidentně 32 bitový typ int přetéká.
Bude přetékat 64 bitový typ?

Pohledem do tabulky vidíme, že pro dané n na vstupu počítáme jako výsledek 2n-té Fibonacciho číslo.

Kolik bitů má F(200)? Cca tolik, kolik je $\log_2(F(200))$. Provedeme tedy odhad.
Z taháku se dozvíme, že

$$F(n) \approx \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n$$

Logaritmujeme a upravíme (kalkulačka pomůže):

$$log_2\big(F(n)\big) \approx log_2\left(\frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n\right) = log_2\left(\frac{1}{\sqrt{5}}\right) + n \cdot log_2\left(\frac{1+\sqrt{5}}{2}\right)$$

$$\doteq log_2(0.447) \dot{+} n \cdot log_2(1.618) \doteq 0.694 \cdot n - 1.162 \approx 0.7n$$

Tedy $\log_2(F(200))$ je cca $0.7 \cdot 200 = 140$.
Datový typ s tímto počtem bitů nemáme a nestačilo by ani 128 bitů.

[3.] ROZBOR zlepšení

Výpočet Fibonacciho čísla vyžaduje jen sčítání.
Reprezentujeme přirozená čísla pomocí pole, do každého prvku uložíme jednu číslici.

Přičtení jednoho čísla ke druhému proběhne v cyklu od nejnižšího řádu, stejně jako sčítáme dvě čísla pod sebou na papíře. Nesmíme zapomenout na případný přenos 1 přes desítku.

```
void add(int num1[], int num2[]) {
  for(int i = 0; i < ARRSIZE-1; i++) { // proceed digit by digit
    num2[i] += num1[i];
    num2[i+1] += num2[i]/10;    // add carry 1 to the next digit
    num2[i] %= 10;              // manage "overflow" of num2[i]
  }
}
```

	před přičtením										po přičtení									
num1	0	0	0	0	0	7	5	0	2	5	0	0	0	0	0	7	5	0	2	5
num2	0	0	0	0	0	4	6	3	6	8	0	0	0	0	1	2	1	3	9	3

Protože data úlohy nejsou zvlášť velká, dovolíme si komfort a budeme při každém sčítání procházet celé pole. Registrace počtu platných číslic by byla snadnou úpravou.

5. ALGORITMUS, složitost

Vstup: n, max_n.
Výstup: Fib(2*n)

Inicializace:
Pole L a XR, obě délky 1+ceil ($\log_{10}$(F(2*max_n))) = 1+ceil(max_n * 0.416)[1].
Vynuluj pole L a XR; L[0] = 1, XR[0] = 2.

Opakuj n-1 krát *// výroba dvojic Fibonacciho čísel*
přičti XR k L *// sčítej cifru po cifře, hodnota se nevejde do typu long*
přičti L k XR *// dtto*

Tisk L bez úvodních nul od nejvyššího řádu.

Složitost časová:
n krát dva průchody polem XR a L, jeden průchod je lineárně úměrný délce pole,
n ≤ max_n, tedy

$$\Theta(n * max\_n) = O(max\_n^2).$$

Složitost paměťová:
zřejmě $\Theta(max\_n)$.

[1]. Přičtená 1 způsobí nejvyšší řád v poli nulový, pro jistotu :-).

[6.] IMPLEMENTACE

```
#define ARRSIZE 43
int L [ARRSIZE], XR [ARRSIZE];

int main() {
    int n;
    while (scanf("%d", &n) != EOF) {

        // initialize L and XR
        for(int i = 1; i < ARRSIZE; i++)
            L[i] = XR[i] = 0;
        L[0] = 1; XR[0] = 2;

        // compute the (2*n)-th fibonacci number
        for(int i = 1; i < n; i++) {
            add(XR,L);
            add(L,XR);
        }

        // print the fibonaccci number
        int k = ARRSIZE-1;
        while ((k > 0) && (L[k] == 0)) k--; // find first non-0 digit
        while (k >= 0)                      // print digits one by one
            printf("%d", L[k--]);
        printf("\n");
    }
    return 0;
}
```

```
void add(int num1[], int num2[]) {
  for(int i = 0; i < ARRSIZE-1; i++){
    num2[i] += num1[i];
    num2[i+1] += num2[i]/10;
    num2[i] %= 10;
  }
}
```

[7.] POZNÁMKY

Po provedení rozboru bylo snadné kód ladit na svém stroji bez odevzdávání,
takže nakonec bylo řešení akceptováno na první pokus.

Vyplatí se nahlížet do taháku se základními poučkami a vzorečky.

Odhad velikosti (prostocviky s logaritmy) byl také užitečný.

Autoři úlohy byli laskaví -- nedali na vstup ostrých dat matoucí 0.

Při počítání možností metodou DP postupného přidávání prvků na konec posloupnosti
může hrozit v soutěžích právě Fibonacciho posloupnost, je proto vhodné rychle toto
podezření ověřit, např. vygenerovat Fibonacciho čísla
a ověřit vůči příkladu v zadání apod.
(Podobná poznámka se může stejně dobře týkat i např. trojúhelníkových čísel,
mocnin prvočísel, apod.)