

## 4. úkol

úterý 15. prosince 2020 14:14

**Úloha 1.** Navrhněte diskrétní pravděpodobnostní prostor pro pravděpodobnostní experiment, při kterém házíme minci a čekáme, než padne první hlava. Vyjmenujte všechny elementární jevy, určete jejich pravděpodobnost a určete střední hodnotu náhodné veličiny popisující experiment.

Nechť naš dis. pr. prostor ( $\Omega, \mathcal{P}$ ) je tvořen našledovně:

$$\Omega = \{\text{padne hlava}, \text{padne oreł}\}$$

$$P(\text{padne hlava}) = P(\text{padne oreł}) = \frac{1}{2}$$

Podle věty z přednášky víme, že střední hodnota náhodné veličiny  $X$  popisující první výskyt el. jevu s pravděpodobností  $p$  je  $E[X] := \frac{1}{p}$

V našem případě tedy:

$$\underline{E[X]} = \underline{\frac{1}{\frac{1}{2}}} = \underline{2}$$

**Úloha 2.** Mějme funkci *RandomBit*, která vrací jeden rovnoměrně pravděpodobný, nezávisle náhodně vygenerovaný bit. Navrhněte, jak pomocí této funkce generovat rovnoměrně náhodně celá čísla z intervalu  $(0, N)$ . Všimněte si, že  $N$  nemusí být nutně nějaká mocnina dvojký.

### Algoritmus:

1. Najděme nejnižší mocninu  $2$  větší než naše číslo  $N$  ...  $x$
2. Vygenerujme náhodné číslo z intervalu  $<0; x$  ... pro každý  $z$  bitů čísla  $x$  až na nejvyšší (ten nastavíme na  $0$ ) nastavíme hodnotu náhodně pomocí funkce *RandomBit*
3. Pokud je toto číslo menší než  $N$ , konec, jinak opakuj krak 2

### Důkaz horečnosti a konečnosti:

- Krok 1 lze realizovat v čase  $O(\log N)$   
 $(x = 2^{\lfloor \log N \rfloor + 1}) \rightarrow$  konečný, korektní

- Krok 2: pravděpodobnost vygenerování každého čísla z  $<0; x)$  je stejná, jelikož má každé číslo  $\xrightarrow{\text{jednoznačný zápis}}$  v binární soustavě ( $O(\log N)$  bitová složitost)
- Krok 3: pravděpodobnost bude správně rozložena i v případě opakování – pokud číslo našlezní  $< N; x)$ , bude vygenerováno znova se stejnou pravděpodobností, ta se tedy rozloží na  $\frac{1}{N}$  (součet konvergující řady)

Konečnost: střední hodnota počtu pokusů je v "nejhorším případě" ( $N$  je mocnina  $2 + 1 \sim \frac{1}{\frac{1}{2}} = 2$ .  $O(1) \dots$  PRŮMĚRNÝ PŘÍPAD)

Algoritmus má v průměrném případě časovou složitost  $O(\log N)$  a konstantní paměťovou složitost.  $\square$

**Úloha 3.** Uvažujme posloupnost  $a_0, \dots, a_n$  celých čísel. Dvojici  $(i, j)$  nazveme inverzí právě tehdy, když platí  $i < j$  a zároveň  $a_i > a_j$ . Navrhněte algoritmus, který pro zadanou posloupnost zjistí, kolik obsahuje inverzí.



### Nainí algoritmus:

```
INVERZE = 0
FOR i IN 0...N
    FOR j IN i...N
        IF (i < j ∧ ai > aj)
            INVERZE ++

```

Důkaz hmotnosti: algoritmus skončí (používá pouze konečné cykly) s časovou složitostí  $O(n^2)$

Důkaz horečnosti: algoritmus vyzkouší každou kombinaci indexů právě jednou a počítá dle definice

**Úloha 4.** Docent Uzel ve svém zatím posledním článku publikoval myšlenku nové, na porovnání založené datové struktury podobné haldě, která umožňuje a) vložení nového prvku v čase  $\mathcal{O}(\log n)$ , b) vybudování struktury z  $n$  prvkového pole v čase  $\mathcal{O}(n)$ , c) nalezení maxima v čase  $\mathcal{O}(1)$  a d) odstranění maxima v čase  $\mathcal{O}(1)$ .

Bez znalosti detailů fungování této struktury dokážte, že taková datová struktura nemůže existovat.

Po naší struktuře vyzadujeme nalezení i odstranění maxima v konstantním čase.

Musí být tedy schopna kompletně se upravit, přičemž mezi maximy <sup>resp.</sup> přehází v konstantním čase. odstraněním maxim

Zároveň po této struktuře ale vyzadujeme vybudování z neseriázeného pole v lineárním čase. Potřebovali bychom si tedy parsováním všech prvků předprípravit všechna maxima. (SERIADIT POLE)  
Na to ale potřebujeme složitost  $\mathcal{O}(\log n \cdot n)$   
 $\Rightarrow$  SPOR.

**Úloha 5.** Uvažujte následující tahovou hru pro dva hráče. Na stole leží řada  $N$  mincí různé hodnoty. V každém tahu si hráč, který je zrovna na řadě, může vzít nějakou minci z jednoho ze dvou konců řady. Cílem hráčů je získat co největší počet peněz. První tah patří Vám. Navrhnete algoritmus, který pro zadanou řadu řekne, jaký bude Váš bodový zisk při použití optimální strategie.

### Naivní algoritmus:

$\text{TAM} = \{ \text{size}, \text{index}, \text{skore1}, \text{skore2} \}$

↓  
 PRCHODSKORE  
 přichodí  
 skore1 ↔ 2

↓  
 umožní jednoznačně  
 reprezentovat "subarray"  
 aktuálního stavu hry

↓  
 skóre hráče  
 na tahu

↑ skóre 2. hráče

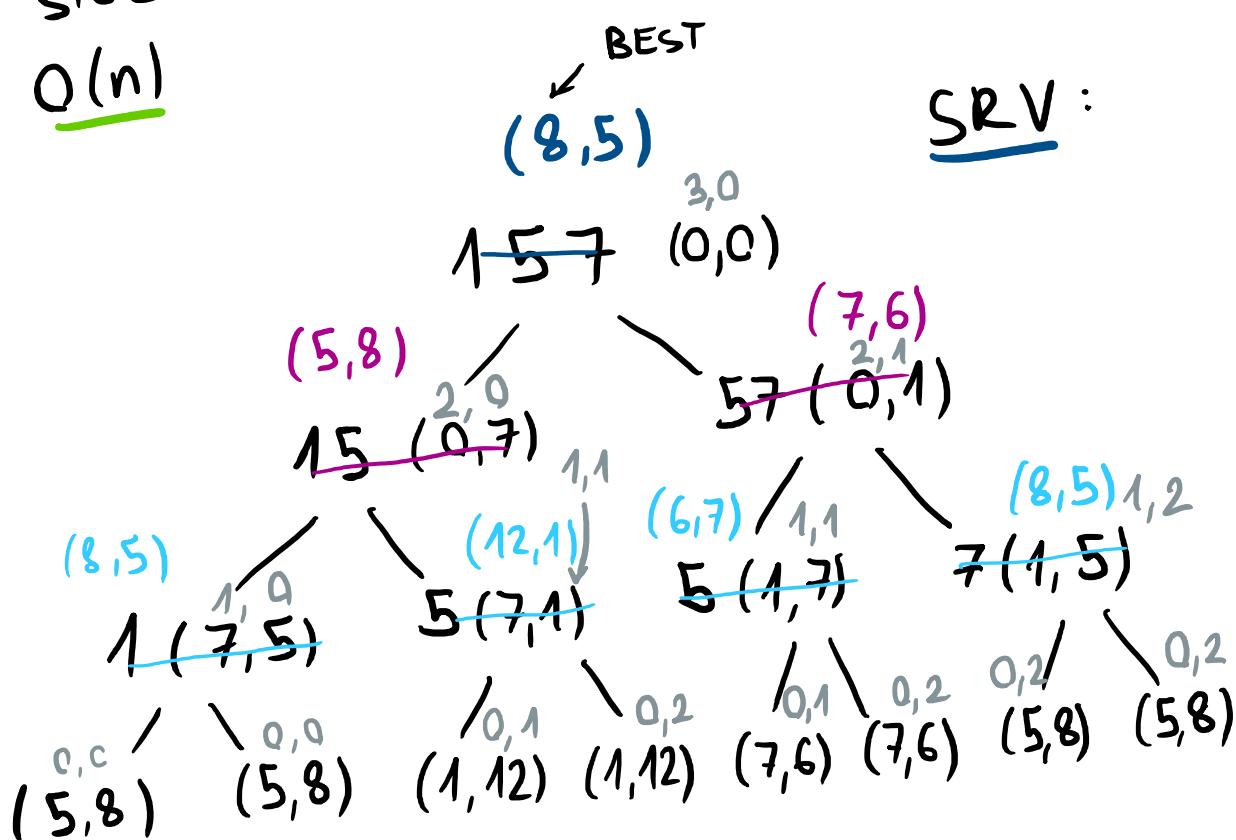
$\text{TAM nejlepsiTah}(\text{TAM})$

- POKUD size == 0  
↳ VRAJ TAM
- start = nejlepsiTah( $\{ \text{size}-1, \text{index}+1, \text{skore2}, \text{skore1} + \text{DATA}[\text{index}] \}$ )
- end = nejlepsiTah( $\{ \text{size}-1, \text{index}, \text{skore2}, \text{skore1} + \text{DATA}[\text{index}+\text{size}-1] \}$ )

RETURN start → skore2 ≥ end → skore2  
 ? start → prchodSkore(): end → prchod Skore()

Důlha konečnosti: tento algoritmus v každé úrovni stromu rekurzivního volání sníží velikost  $\text{SIZE}$  o 1, přičemž na velikosti  $O$  se zastaví

Důlha horečnosti: tento algoritmus prozkouší všechny možnosti, vybere tu nejlepší možnost a buď vráť Skutečně zkouší vzít žeton ze začátku i z konce řady. Celkem bude mít tedy složitost časovou  $O(2^n)$  a paměťovou  $O(n)$



## Dynamické programování:

- Všimněme si, že se v řešení vykyla duplicita při délce 1 a indexu 1, jednou se stavem (1,7) a jednou (7,1).
- Náš algoritmus zbytečně volá stejnou posloupností volání rekurze  
na obou, přičemž je zřejmé,  
že vždy bude výsledné skóre  $(1+a, 7+b)$   
a  $(7+a, 1+b)$ , kde algoritmus vybere  
první variantu  $(7+b > 1+b)$ .

Pro dané posloupnosti se nám tedy vyplatí si pamatovat, při které kombinaci délky a indexu dosáhneme nejlepšího skóre.

## Upravený algoritmus:

TAM nejlepsi Tah (TAM)

- POKUD size == 0  
↳ VRAŤ TAM
- POKUD je TABLE [size][index] definováno a  $TAH \rightarrow skore2 < TABLE [size][index]$   
↳ VRAŤ TABLE [size][index]
- start = nejlepsi Tah({size-1, index+1, skore2, skore1 + DATA [index]})
- end = nejlepsi Tah({size-1, index, skore2, skore1 + DATA [index+size-1]})
- TABLE [size][index] =  
↳ start  $\rightarrow$  skore2  $\geq$  end  $\rightarrow$  skore2  
? start  $\rightarrow$  prchod Skore(): end  $\rightarrow$  prohod Skore()  
• RETURN TABLE [size][index]

Ještě si napišme tuto vylepšení  
iterativně:

TABLE[N][0] = (0,0);  
FOR SIZE IN N - 1 ... 1  
FOR INDEX IN 0 ... (N-SIZE-1)

PREV = TABLE [SIZE][INDEX]

start = (PREV → skore<sub>2</sub>, PREV → skore<sub>1</sub> + DATA  
[index])

POKUD TABLE [SIZE-1][INDEX+1] není def.

nebo je skore<sub>2</sub> ≥ než start → skore<sub>2</sub>

  L TABLE [SIZE-1][INDEX+1] = start → <sup>PROHOD</sup><sub>skore<sub>2</sub>)</sub>

konec - (PREV → skore<sub>2</sub>, PREV → skore<sub>1</sub>  
+ DATA [index+size-1])

POKUD TABLE [SIZE-1][INDEX] není def.

nebo je skore<sub>2</sub> ≥ než konec → skore<sub>2</sub>

  L TABLE [SIZE-1][INDEX] = konec → <sup>PROHOD</sup><sub>skore<sub>2</sub>)</sub>

RETURN max ( TABLE[0][.] → skore<sub>1</sub>)

Důraz konečnosti: algoritmus obsahuje pouze konečné cykly, tedy skončí a to v čase  $O(n^2)$

Důraz horečnosti: algoritmus stále vyzkouší a wygeneruje všechny možnosti a vrátí tedy tu nejlepší.