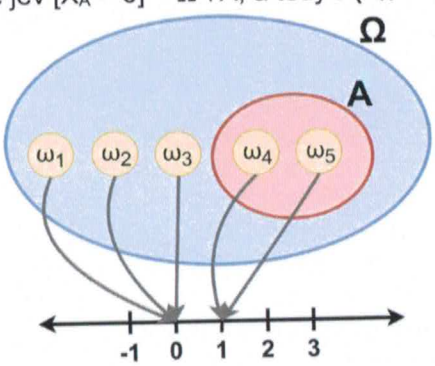
53. Randomizované algoritmy (Monte Carlo a Las Vegas algoritmy)

Indikátorová náhodná proměnná

Ať $(\Omega, \mathbb{S}, P)$ je pravděpodobnostní prostor, dále ať $A \in \mathbb{S}$ je jev (tudíž $A \subseteq \Omega$, A je tedy nějaká množina výsledků pokusu) a ať $X_A: \Omega \to \mathbb{R}$ je náhodná veličina. Řekněme, že X_A je **indikátorová náhodná proměnná jevu A**, pokud pro $\forall \omega \in \Omega$:

- $\omega \in A \Rightarrow X_A(\omega) = 1$
  - indikátorová náhodná veličina X_A **nabývá hodnoty 1** pro výsledky pokusu (elementární jevy) **ležící v jevu A**
  - z toho plyne, že jev $[X_A = 1] = A$, a tudíž $\mathbf{P(X_A = 1) = P(A)}$
- $\omega \in \Omega \setminus A \Rightarrow X_A(\omega) = 0$
  - indikátorová náhodná veličina X_A **nabývá hodnoty 0** pro výsledky pokusu (elementární jevy) **ležící mimo jev A**
  - z toho plyne, že jev $[X_A = 0] = \Omega \setminus A$, a tedy $\mathbf{P(X_A = 0) = P(\Omega \setminus A) = 1 - P(A)}$



Příklad. Házíme jedenkrát poctivou šestistěnnou kostkou. Základní prostor $\Omega = \{1, 2, 3, 4, 5, 6\}$. Máme jev $A = \{1, 2\}$, tedy jev A nastane tehdy, pokud na kostce padne hodnota 1, nebo 2. Pro odpovídající indikátorovou náhodnou proměnnou X_A platí:

- $[X_A = 0] = \{3, 4, 5, 6\}$
  - X_A nabývá hodnoty 0 tehdy, pokud na kostce padne jedno z čísel 3, 4, 5, 6
- $[X_A = 1] = \{1, 2\}$
  - X_A nabývá hodnoty 1 tehdy, pokud na kostce padne jedno z čísel 1, 2

Ať $(\Omega, \mathbb{S}, P)$ je pravděpodobnostní prostor. U indikátorových náhodných proměnných pozorujeme několik vlastností:

- **střední hodnota indikátorové náhodné proměnné**
  - ať X_A je indikátorová náhodná proměnná jevu $A \in \mathbb{S}$
  - **střední hodnotu** této náhodné veličiny spočítáme pomocí vzorce

$$EX_A = \sum_{a \in Im(X_A)} a \cdot P(X = a) = 0 \cdot P(X = 0) + 1 \cdot P(X = 1) = P(X = 1) = P(A)$$

  - *Im(X_A) je obor hodnot náhodné veličiny X_A, zde množina {0, 1}*
- **střední hodnota** indikátorové náhodné proměnné jevu A tedy přesně odpovídá **pravděpodobnosti jevu A**
- pro výše uvedený příklad házení kostkou platí **EX_A = P(A) = 1/3**, tedy očekávaná hodnota náhodné veličiny X_A je 1/3

- **linearita střední hodnoty**
  - mějme *n* indikátorových náhodných proměnných X_{A1}, X_{A2}, …, X_{An} po řadě pro jevy A_1, A_2, …, $A_n \in \mathbb{S}$
  - definujme náhodnou proměnnou $X = \sum_{i=1}^{n} X_{A_i}$ jako sumu výše uvedených náhodných proměnných X_{A1}, X_{A2}, …, X_{An}
  - tato náhodná veličina nabývá hodnot odpovídajících přirozeným číslům v rozsahu **0 - n**
  - pro výpočet její střední hodnoty lze využít **linearity střední hodnoty**

$$EX = E\left(\sum_{i=1}^{n} X_{A_i}\right) = \sum_{i=1}^{n} EX_{A_i}$$

  - tedy střední hodnotu je možné vložit **dovnitř sumy** a jednoduše sečíst střední hodnoty jednotlivých indikátorových proměnných
  - v rámci příkladu uvažujme o *n* hodech šestistěnnou kostkou, jak bylo naznačeno výše
    - definujeme jevy A_1, A_2, A_3, …, A_n, kde jev A_i popisuje, že v *i.* hodu kostkou padla hodnota 1, nebo 2
    - zavedeme odpovídající indikátorové náhodné proměnné X_{A1}, X_{A2}, …, X_{An}, kde náhodná proměnná X_{Ai}
      - nabývá hodnoty 1, pokud v *i.* hodu padla na kostce hodnota z množiny {1, 2}
      - nabývá hodnoty 0, pokud v *i.* hodu padla na kostce hodnota z množiny {3, 4, 5, 6}
    - náhodná veličina $X = \sum_{i=1}^{n} X_{A_i}$ nabývá hodnoty od **0** do **n** dle toho, kolikrát na provedených *n* pokusech na kostce padla hodnota 1, nebo 2
    - v nejhorším případě nepadne žádná z těchto hodnot ani jednou, tedy X nabývá hodnoty **0**
    - v nejlepším případě padne některá z těchto hodnot pokaždé, tedy X nabývá hodnoty **n**
    - **zajímá nás průměrný případ**, tedy hledáme střední hodnotu náhodné veličiny X:

$$EX = E\left(\sum_{i=1}^{n} X_{A_i}\right) = \sum_{i=1}^{n} EX_{A_i} = \sum_{i=1}^{n} P(X_{A_i}) = \sum_{i=1}^{n} \frac{1}{3} = \frac{n}{3}$$

    - **hodnota 1, nebo 2 se na kostce průměrně objeví v n/3 případech z n hodů**

Indikátorové proměnné budeme používat pro pravděpodobnostní analýzu chování randomizovaných algoritmů v průměrném případě.

Problém náboru

Mějme *n* kandidátů na jednu pracovní pozici. Nad těmito kandidáty je možné sestrojit relaci totálního ostrého uspořádání dle toho, o jak kvalitní kandidáty jde. Jinak řečeno, pokud zvolíme dva libovolné kandidáty, po krátkém přijímacím pohovoru s každým z nich lze snadno a jednoznačně určit, který z nich je lepším kandidátem (žádní dva nejsou stejně dobří).

Zaměstnavatel postupně provádí přijímací pohovory se všemi kandidáty. Pokud je pracovní pozice dosud neobsazena, je zaměstnán jakýkoliv kandidát, který přijde na řadu (je lepší někoho zaměstnat než mít místo neobsazeno). Pokud je pracovní pozice obsazená a zaměstnavatel provede pracovní pohovor s kandidátem, který je lepší než právě zaměstnaná osoba, je tato osoba propuštěna a zaměstnán je nově objevený lepší kandidát.

Tímto postupem bude nakonec zaměstnán nejlepší ze všech kandidátů. Cena pracovního pohovoru je zanedbatelná, ovšem náklady na zaměstnání nového kandidáta jsou značné, označme je pomocí konstanty **c**. Pokaždé, když je zaměstnán nový kandidát, je třeba tyto náklady uhradit.

Algoritmus: Problém náboru
Vstup: n kandidátů
Výstup: index *hired* nejlepšího kandidáta

1. *hired = 0*
2. *for k ∈ [1 .. n]:*
    a. *interview(k)*
    b. *if **k** is better than **hired**:*
        i. *hire(k)*
        ii. *hired = k*

- **nejlepší případ**
    - nejschopnější kandidát nalezen jako první
    - náklady za zaměstnání se uhradí pouze jednou
    - složitost činí **Ω(c)**
    - při náhodném pořadí příchozích kandidátů je pravděpodobnost nejlepšího případu **1/n**
        - všech možných permutací množiny kandidátů je **n!**
        - příznivých permutací, kdy první kandidát je napevno zvolen, je **(n-1)!**
        - $\frac{(n-1)!}{n!} = \frac{1}{n}$
- **nejhorší případ**
    - kandidáti přicházejí přesně v pořadí daném uspořádáním kvality kandidátů

- postupně zaměstnáme všech *n* kandidátů
- to odpovídá složitosti **O(c·n)**
- při náhodném pořadí příchozích kandidátů je pravděpodobnost nejhoršího případu **1/n!**
  - všech možných permutací množiny kandidátů je **n!**
  - příznivá permutace, kdy jsou kandidáti seřazeni dle kvality, je pouze jedna

- **průměrný případ**
  - uvažujme o náhodném pořadí příchozích kandidátů
  - zavedeme *n* indikátorových náhodných proměnných X_1, X_2, X_3, …, X_n, kde náhodná proměnná X_i
    - nabývá hodnoty 1, pokud *i.* kandidát byl zaměstnán
      - byl lepší než všichni předchozí kandidáti
    - nabývá hodnoty 0, pokud *i.* kandidát nebyl zaměstnán
      - některý z předchozích kandidátů byl lepší
  - najdeme střední hodnotu indikátorové proměnné X_i
    - $EX_i = P(X_i)$
    - hledáme tedy pravděpodobnost, že *i.* kandidát je zaměstnán
    - k tomu dojde pouze tehdy, pokud je nejlepší z prvních *i* příchozích kandidátů
    - tato pravděpodobnost tedy činí **1/i**, což přesně odpovídá i střední hodnotě proměnné X_i
    - **$EX_i = 1/i$**
  - dále definujeme náhodnou proměnnou $X = \sum_{i=1}^{n} X_i$, která popisuje počet postupně zaměstnaných kandidátů, tedy v nejlepším případě nabývá hodnoty **1**, v nejhorším případě nabývá hodnoty **n**
  - zkoumejme průměrný případ
    - $EX = E(\sum_{i=1}^{n} X_i) = \sum_{i=1}^{n} EX_i = \sum_{i=1}^{n} \frac{1}{i} = ln\, n + O(1)$
  - tedy v průměrném případě provedeme O(ln n) zaměstnání kandidáta, a tedy náklady na zaměstnávání budou O(c ln n), což je výrazně příznivější než nejhorší případ
  - **toto ovšem platí pouze v případě, že se můžeme spolehnout na náhodné pořadí kandidátů na vstupu**

Randomizované algoritmy

V předchozí části jsme viděli, že problém náboru se v průměrném případě chová výrazně lépe než v nejhorším případě, ovšem to jen za předpokladu, že pořadí kandidátů je skutečně náhodné (můžeme se spolehnout, že není podvržené, neřídí se podle nějakého pravidla, s nímž jsme při analýze nepočítali).

Budeme tedy používat **randomizaci** pro zajištění odvozené složitosti v průměrném případě. Běžné metody randomizace jsou:

- **randomizace vstupu**

    - náhodně zpřeházíme vektor vstupních hodnot
        - problému náboru
- **náhodné volby v těle algoritmu**
    - za běhu algoritmu náhodně volíme hodnoty, které nemusí mít souvislost se vstupem
        - quick sort

Rozlišujeme dvě základní třídy randomizovaných algoritmů:

- **Las Vegas algoritmy**
    - Las Vegas algoritmy **vždy vrátí správný výsledek**
    - typicky je u nich garantovaná **lepší průměrná časová složitost** než je složitost odpovídajícího deterministického algoritmu
    - ovšem pro některé instance (nikoliv však v očekávaném případě) může být jeho složitost horší než složitost odpovídajícího deterministického algoritmu
    - pravděpodobnost, že algoritmus typu Las Vegas poběží výrazně déle než jeho deterministická varianta, je typicky poměrně malá
- **Monte Carlo algoritmy**
    - Monte Carlo algoritmy **mohou vrátit nesprávný výsledek**
    - je zde garantovaná **lepší časová složitost v nejhorším případě** než časová složitost odpovídajícího deterministického algoritmu
    - pravděpodobnost, že algoritmus typu Monte Carlo vrátí nesprávný výsledek, je typicky poměrně malá

Předpokládejme, že máme pole sudé délky o velikosti n obsahující prvních n kladných přirozených čísel ve zcela náhodném pořadí. Chceme sestrojit algoritmus, který najde libovolný index i do tohoto pole, na němž se nachází číslo vyšší než $n/2$.

Deterministický algoritmus

Algoritmus: Deterministické nalezení čísla z horní poloviny rozsahu
Vstup: Pole V s náhodnou permutací n přirozených čísel z rozsahu 1 až n
Výstup: Index i, kde $V[i] > n/2$

1. *for $i \in [1 .. n]$:*
    a. *if($V[i] > n/2$):*
        i. *return i*

    - v nejlepším případě je vhodný index nalezen hned jako první, tedy se potýkáme se složitostí **Ω(1)**
    - v nejhorším případě musíme prohledat n/2 prvků pole, než najdeme vhodný index, tedy složitost odpovídá **O(n)**

Las Vegas algoritmus

Algoritmus: Randomizované nalezení čísla z horní poloviny rozsahu, styl Las Vegas
Vstup: Pole *V* s náhodnou permutací *n* přirozených čísel z rozsahu *1* až *n*
Výstup: Index *i*, kde *V[i] > n/2*

1. *while True:*
    a. *i = randInt(0, n-1)*
    b. *if(V[i] > n/2):*
        i. *return i*

- **v nejlepším případě** je vhodný index nalezen hned jako první, tedy se potýkáme se složitostí **Ω(1)**
- **v nejhorším případě cyklíme**, což je horší než čas O(n), ovšem pravděpodobnost něčeho takového se blíží nule
- analyzujme **průměrný případ**
    - zavedeme náhodné jevy A_1, A_2, A_3, … (je jich tentokrát nekonečně mnoho), přičemž jev A_i popisuje situaci, že bylo nezbytné pustit *i.* iteraci while cyklu (hlavní smyčky), neboť algoritmus dosud neskončil
    - zavedeme indikátorové náhodné proměnné X_{A1}, X_{A2}, X_{A3}, …, přičemž náhodná proměnná X_{Ai} popisuje jev, že bylo spuštěno *i.* provedení hlavní smyčky
        - X_{Ai} nabývá hodnoty 1 tehdy, pokud bylo nutné spustit *i.* iteraci (algoritmus neskončil dříve)
        - X_{Ai} nabývá hodnoty 0, pokud nebylo nutné spustit *i.* iteraci (algoritmus skončil již dříve)
    - zřejmě $EX_1 = P(A_1) = 1$, neboť první iterace je spuštěna vždy, tedy pravděpodobnost jevu A_1 je 1
    - pravděpodobnost, že jsme v první iteraci našli hodnotu vyšší než *n/2*, je 1/2, tedy pravděpodobnost, že potřebujeme spustit druhou iteraci, je rovněž 1/2
    - tedy $EX_2 = P(A_2) = 1/2$
    - obecně pravděpodobnost, že je nutné spustit *i.* iteraci, odpovídá $1/2^{i-1}$, neboť se současně muselo stát, že všechny předchozí (a **nezávislé**) náhodné volby indexu byly nevhodné
    - zavedeme náhodnou proměnnou $X = \sum_{i=1}^{\infty} X_i$, která popisuje počet spuštění hlavní smyčky algoritmu
    - zřejmě $EX = E(\sum_{i=1}^{\infty} X_i) = \sum_{i=1}^{\infty} EX_i = \sum_{i=1}^{\infty} 2^{-(i-1)}$
    - jde o geometrickou řadu s kvocientem **q = 1/2** a s prvním prvkem $a_1 = 2^{-(1-1)} = 1$, tedy lze psát $EX = \frac{a_1}{1-q} = \frac{1}{1-\frac{1}{2}} = 2$
    - **očekávaný počet provedení hlavní smyčky je 2**, tedy algoritmus pracuje s průměrnou časovou složitostí **O(1)**

6/8

- algoritmus tedy vždy vrátí správný výsledek, může mít v nejhorším případě značně horší časovou složitost než deterministická varianta, ale v průměrném případě se chová výrazně lépe než deterministická varianta v nejhorším případě

Monte Carlo algoritmus

Algoritmus: Randomizované nalezení čísla z horní poloviny rozsahu, styl Monte Carlo
Vstup: Pole *V* s náhodnou permutací *n* přirozených čísel z rozsahu *1* až *n*, konstanta *c* značící pevný počet opakování hlavní smyčky
Výstup: Index *i*, kde *V[i] > n/2*

1. *for i ∈ [1 .. c]:*
    a. *i = randInt(0, n-1)*
    b. *if(V[i] >= n/2):*
        i. *return i*
2. *return randInt(0, n)*

- **v nejlepším případě** je vhodný index nalezen hned jako první, tedy se potýkáme se složitostí **Ω(1)**
- **v nejhorším případě** je smyčka provedena *c*krát a ani poté není nalezen korektní výsledek, algoritmus vrací náhodný index, potýkáme se však opět se složitostí **O(1)**
- tento nejhorší případ nastane s pravděpodobností $\mathbf{1/2^c}$, což může být velice nízká pravděpodobnost, pokud zvolíme dostatečně vysoké číslo **c**
- provedeme-li hlavní smyčku pouze jednou, algoritmus vrátí špatný výsledek s pravděpodobností **1/2**, což je poměrně nepříznivé
- metodu navýšení opakování hlavní smyčky pro snížení pravděpodobnosti chyby nazýváme **amplifikace**
    - je však žádoucí hodnotu *c* skutečně stanovit jako konstantu, abychom si udrželi v nejhorším případě čas **O(1)**
    - pokud bychom hlavní smyčku prováděli například *n/2*krát, tedy v nekonstatním počtu, dostaneme se v nejhorším případě na složitost **O(n)**
- při analýze průměrného případu bychom zjistili, že průměrný počet spuštění hlavní smyčky odpovídá hodnotě $\mathbf{2 - 2^{-c}}$, což rovněž odpovídá složitosti **O(1)**

Příklady dalších randomizovaných algoritmů

- **Quick-sort**
    - sekvenční řadicí algoritmus, který provádí náhodný výběr **pivotního prvku** za běhu algoritmu
    - jde o algoritmus typu **Las Vegas**
        - v každém spuštění vrátí korektní výsledek
        - v nejhorším případě složitost $\mathbf{O(n^2)}$
        - v průměrném případě složitost $\mathbf{O(n \cdot \log n)}$

- **Miller-Rabinův algoritmus**
  - pravděpodobnostní algoritmus pro testování prvočíselnosti vstupního čísla
  - vychází z ověřování jisté algebraické vlastnosti, která musí platit pro každé prvočíslo
  - tato vlastnost u složených čísel platit nemusí, ale může
  - jde o algoritmus typu **Monte Carlo**
    - má garantovanou složitost **$O(\log^3 n)$**, kde n je vstupní číslo
    - pokud má na vstupu prvočíslo, vždy jej označí za prvočíslo
    - pokud má na vstupu složené číslo, je šance nejvýše 1/4, že jej chybně prohlásí za prvočíslo, jinak jej prohlásí za složené číslo
    - pokud tedy algoritmus nějaké číslo prohlásí za složené, je zcela jistě složené

Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele kocotom.