1) Napiště metodu *int zjistiCenuCesty(Node list)* pro binární strom, který nalezne cestu od listu směrem ke kořenu stromu (UML diagram – viz tabule)

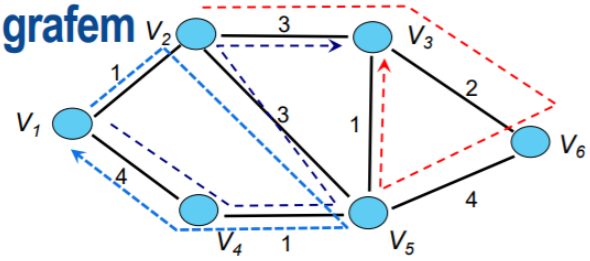
Cca musí se dodělat podle toho UML

```
public int zjistiCenuCesty(Node list) {
            Node tmp = list;
            int cena = 0;
            while(tmp != root) {
                  cena += tmp.getCena();
                  System.out.println(tmp.getNazev());
                  tmp = tmp.getRodic();
            }
            return cena;
      }
```

2) Definujte pojmy a) vážený graf, b) průchod grafem, c) cesta grafem, d) bipartitní graf

Vážený graf je graf, kde každá hrana *e* má přidělenou váhu.

Průchody a cesty grafem



Průchod je sekvence vrchlů $(v_1, v_2, ..., v_L)$ takových, že $\{(v_1, v_2), (v_2, v_3), ..., (v_{L-1}, v_L)\} \subseteq E$, např. $(V_2, V_3, V_6, V_5, V_3)$

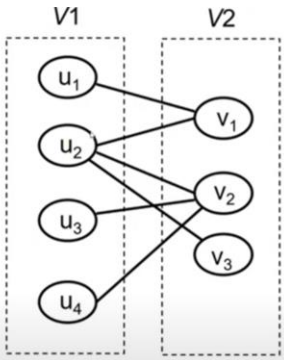
Jednoduchá cesta je průchod bez opakování vrcholů, např. $(V_1, V_4, V_5, V_2, V_3)$

Cyklus je průchod $(v_1, v_2, ..., v_L)$ kde $v_1 = v_L$ (první = poslední) bez opakování uzlů a $L>=3$, například $(V_1, V_2, V_5, V_4, V_1)$. Cyklus není cesta.

Graf je nazývaný *cyklický* jestliže obsahuje alespoň jeden cyklus; jinak se nazývá *acyklický*

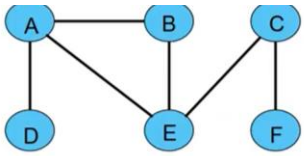
Bipartitní graf – nemůžou být spojeny dva prvky v rámci jedné množiny

bipartitní graf je neorientovaný graf $G = (V,E)$ kde V mohou být rozděleny do dvou vzájemně disjunktních množin V1 and V2, kde $(u,v) \in E$ znamená současně:
$u \in V1$ a $v \in V2$
anebo
$v \in V1$ a $u \in V2$.

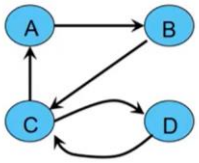


3) Spojitý a nespojitý graf

- Neorientovaný graf je spojitý, jestliže pro libovolné dva vrcholy u,v existuje cesta z u do v.



- Orientovaný graf je *pevně spojitý* jestliže pro libovolné dva vrcholy u,v existuje orientovaná cesta



  - Graf je *řídký* jestliže $|E| \approx |V|$
  - Graf je *hustý* jestliže $|E| \approx |V|^2$

4) Určete kostru grafu s použitím Kruskalova algoritmu (zadani tabule)

Funguje na principu shlukování dvou množin hran. Ze všech hran se vybere hrana s nejnižší hodnotou a množiny vrcholů jenž hrana propojuje se seskupí do jedné množiny. Tak se postupuje tak dlouho dokud všechny vrcholy nejsou v jedné množině. Pokud cesta propojuje vrcholy ze stejné množiny vrcholů tak se nepoužije a pokračuje se z následující hranou. Využívá se pokud známe celou topologii grafu.

Kruskalův a Distribuovaný algoritmus (obecně kostra grafu) **odstraňují cykly v grafu.**

5) Distribuovaný algoritmus

Zde se pracuje na principu, že se kostra tvoří na každé množině zároveň. Z každého vrcholu se vyrazí směrem po nejmenší hodnotě hrany. Pokud se vrcholy na cestě potkají tak vytvoří množinu pokud se nepotkají nic se neděje. Dále se pak vysílají znovu po nejmenší hodnotě hrany. Toto se děje dokud se celá kostra nenajde. Tento algoritmus se nejčastěji využívá v telekomunikačních sítích, kde neznáme celou topologii grafu.

6) Uveďte Dijkstrův algoritmus, b) srovnejte s algoritmem prohledávání do šířky a hloubky

Dijkstrův algoritmus pracuje jen pokud jsou všechny váhy hran kladné, poskytuje nejkratší cesty ze zdroje do všech vrcholů grafu. Směrovací tabulky IS-IS, OSPF. Může být ukončen okamžitě po nalezení cesty do hledaného vrcholu.

Algoritmy jsou identické, liší se ve způsobu výběru kandidátních cest. Do šířky (BFS) – fronta, do hloubky (DFS) – zásobník, Dijkstra – prioritní fronta. BFS se hodí pro nalezení nejkratší cesty v grafu, kde nejsou ohodnocené hrany.

https://www.algoritmy.net/article/5108/Dijkstruv-algoritmus

7) Srovnejte algoritmus prohledávání do šířky a do hloubky. Popište je.

BFS - využívá frontu, do které nejprve vloží vstupní vrchol. Z tohoto vrcholu se vezmou všichni sousedé a vloží se do fronty, z které se bere postupně a jejich sousedé se přidávají do fronty. Při tomto se každý už navštívený vrchol ukládá do nějakého pole/seznamu navštívených abychom ho nenavštěvovali znovu.

DFS - pracuje na principu, že prozkoumává prvně vrchol na jednu stranu a v ní pokračuje dokud nedojde na konec nebo nedorazí do už navštíveného vrcholu. Využívá zásobník. DFS je méně paměťově náročné, BFS je optimální.

Prohledávání do šířky – BFS

Algoritmus je **úplný** a **optimální** (pokud je cena hran stejná)**. Nalezne nejkratší cestu v neohodnoceném grafu.** Prostorová složitost **$O(b^m)$** – kde B je max. počet větvení a M je maximální hloubka grafu od počátečního uzlu

Časová složitost **O(b^m)**
Probíhá způsobem, kde jsou nejdříve prohledávány stavy blízké počátku a pokud algoritmus nedospěje k řešení, postupně prochází vzdálenější a vzdálenější uzly. Množina OPEN představuje posloupnost doposud nenavštívených stavů. Množina CLOSED představuje množinu těch stavů, které byly prohledány. Využívá frontu – přidává na konec, odebírá ze začátku. **DFS má výrazně nižší nároky na paměť**

BestFS – zobecňuje BFS, používá prioritní frontu, využívá heuristiku – pro odhad nejlepší cesty, není optimální – heuristika se může mýlit

A* - vychází z BestFS + bere v potaz cestu kterou urazil, úplný i optimální

Prohledávání do hloubky – DFS

Prostorová složitost **O(bm)** – kde B je max. počet větvení a M je maximální hloubka grafu od počátečního uzlu
Časová složitost **O(b^m)**
Probíhá v porovnání s BFS odlišně ‑ jsou naopak upřednostněny uzly, které jsou nejvzdálenější počátku. Pseudokód chování algoritmu je velice podobný předchozímu s drobným rozdílem ‑ prvky při generování prohledávaného stavu nejsou odebírány z počátku, ale z konce seznamu. Tato změna má významný rozdíl v chování a časových charakteristikách.
Nikdy nepoužívat pro nalazené nejkratší cesty!

8) Prohledávání s omezenou hloubkou (DLS)

Kompromis mezi BFS a DFS
Vychází z algoritmu DFS, **ale max. hloubka je omezena**, využívá zásobník (jako DFS)
Je výhodný pokud znám počet potřebných tahů – úspora paměti i průměrného času řešení
Ideální pokud známe hloubku, ve které se řešení nachází -> optimální
Nízká spotřeba paměti (z DFS)

9) Iterativní prohledávání do hloubky s omezenou hloubkou IDLS (IDDFS)

Je iterativní prohledávaní do hloubky. Vychází z DFS ale v každé iteraci se prohledává jen o hloubku níž.
Kompromis mezi BFS a DFS
Algoritmus je **úplný** a **optimální** (pokud je cena hran stejná), využívá zásobník
Funguje tak, že opakovaně spouští algoritmus **DLS** s postupně se zvyšující hloubkou zanoření.

10) Algoritmus a) k-nejbližších sousedů, b) křížovou validaci, c) matici záměn, d) jak spolu souvisí, e) specifičnost a senzitivita.

Jedná se o algoritmy strojového učení

k-nejbližších sousedů

- Projdou se všechna trénovací data a pro každý trénovací prvek se změří vzdálenost od klasifikovaného prvku, vybere se ta s nejmenší vzdáleností a použije se její třída

křížová validace

- pro ohodnocení vytvořeného modelu (algoritmu) na omezeném vzorku dat (trénování a testování)
- data určená pro trénování jsou současně testovací data
- rozdělení do n skupin (n-1 pro trénování, 1 pro testování)
- opakujeme n-krát

Kdy je křížová validace potřeba?

matice záměn

A: TP (true positive), B: FN (false negative)
C: FP (false positive), D: TN (true negative)

	Předpověděná třída		
Skutečná třída		Class=Ano	Class=Ne
	Class=Ano	A (TP)	B (FN)
	Class=Ne	C (FP)	D (TN)

Specifičnost a senzitivita

Jedná se o statistické údaje, hodnotí kvalitu jakéhokoliv testu.

Specifičnost vyjadřuje úspěšnost zachycení negativity. 100 % specifičnost znamená, že budou všichni negativní správně označeni za negativní (TN), ale také bude několik pozitivních označeno za negativní (FN)

Senzitivita vyjadřuje úspěšnost zachycení pozitivity. 100 % senzitivita znamená, že budou všichni pozitivní správně označeni (TP), ale také bude několik negativních označeno za pozitivní (FP)

Specifičnost (specificity, S) = $\frac{TN}{FP+TN}$

Senzitivita (recall, R) = $\frac{TP}{TP+FN}$

Přesnost (accuracy, A) $= \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$

11) Systém podpůrných vektorů (SVM), jak se liší oproti neuronovým sítím, k čemu slouží parametry C a epsilon

Algoritmus dokáže separovat i data, která původně nejsou lineárně separabilní (oddělitelná). Řeší explicitně to, co neuronové sítě řeší implicitně. Je poměrně nová metoda – obliba stoupá.
Zjednodušený postup

- Nelineární mapování – převod do prostoru s vyšší dimenzí
- Hledání optimální nadroviny (rozhodovací hranice)
- Ideální oddělení jednotlivých tříd
- Maximální vzdálenost mezi nadrovinou a jednotlivými prvky třídy

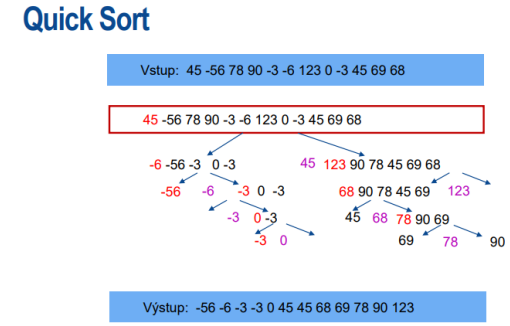
Parametr C – kompromis mezi složitostí modelu a mírou odchylek větších než které jsou tolerovány optimalizační rovnicí

Parametr ξ (ksí, asi myslel epsilon) – ovlivňuje šířku necitlivé oblasti, která se používá k nastavení vzorků trénovacích dat

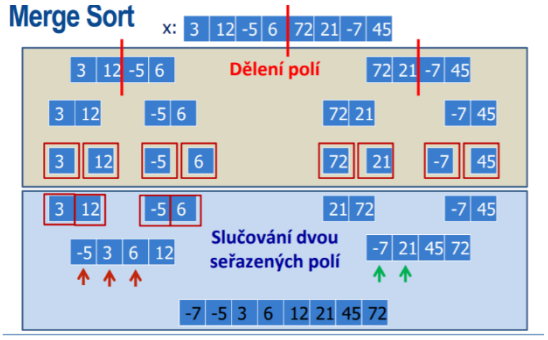
12) Algoritmy bubble sort, merge sort, quicksort

Bubble sort – porovnává 2 sousední prvky, pokud je vlevo větší hodnota, tak se prohodí, pokud je vlevo menší tak se pokračuje na další prvek, doba trvání je n-1. Složitost O(n^2) výsledek: 3, 4, 7, 10; nejpomalejší

Quicksort – náhodně zvolíme prvek z pole – pivot, přeházíme pole tak, aby na jedné straně pivota byly menší hodnoty, na druhé straně větší hodnoty, toto se opakuje pro obě části, dokud nenarazíme na pole velikosti 1, kdy je celé pole seřazené. Při vhodné volbě pivota je složitost O(n*log$n$), při špatné volbě je složitost O(n^2).



Merge sort – Nesetříděné pole je děleno na poloviny, dokud nová pole nemají velikost 1. Poté se skládají dohromady: porovná se první prvek z pole A s prvním prvkem pole B a seřadí se, porovná se druhý prvek z pole A s druhým prvkem pole B, seřadí se a umístí za první prvky. Tento proces je opakován dokud nejspou spojeny všechny prvky původního pole.



13) Objasněte pojmy proces a vlákno, b) upravde kód tak, aby hlavní vlákno skončilo až jako poslední (viz tabule)

Proces si lze představit jako program vykonávaný operačním systémem, jeden program může vykonávat více procesů. Jsou náročnější na vytvoření, komunikace probíhá předáváním zpráv (nesdílí data), dobrá škálovatelnost

Vlákna jsou stavební bloky procesů, které jsou jimi vykonávané. Proces může mít jedno nebo více vláken. Jsou méně náročná na systémové prostředky, sdílí paměť, snadnější programování, omezená škálovatelnost

Synchronizace vláken – bez zajištěn ísynchronizace může dojít k souběhu nebo uvíznutí (deadlock). Cílem je zabránit konfliktům mezi vlákny při přístupu ke sdíleným přostředkům

- Bariéra
- Zámek / semafor
- Synchronní komunikace

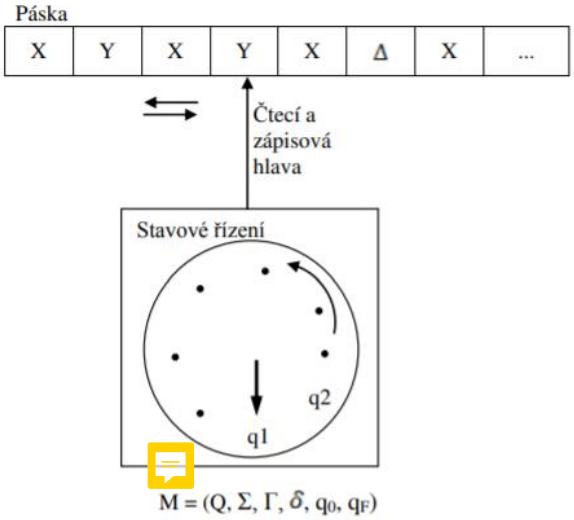
Souběh – dochází k němu v případě, kdy ke stejným datům přistupuje více vláken a jedno vlákon druhému přepíše data. Vzniká chyba za běhu programu.

Deadlock – nastává v případě, kdy dvě vlákna nebo procesy na sebe čekají až se uvolní jimi uzamčené sdílené zdroje

Aby hlavní vlákno skončilo jako poslední zařídíme pomocí *join()*. *Thread.join()*

14) Definujte Turingův stroj. Jak se liší přechodová funkce deterministického od nedeterministického

Turingův stroj je matematický model výpočetního stroje, vznikl za účelem zkoumání vyčíslitelnosti, je teoretický model s nejvyšší výpočetní silou



Q reprezentuje konečnou množinu vnitřních stavů

Σ je **vstupní abeceda**. Δ(prázdný symbol) není součástí Σ

Γ je konečná množina vnitřních symbolů. Je to tzv. pásková abeceda a platí pro ni $\Gamma \subset \Sigma$, $\Delta \in \Sigma$

δ je parciální funkce a popisuje chování Turringova stroje, neboli program. V tomto případě **nelze měnit program** za běhu počítače. Je nazývána přechodová funkce.

q_0 – počáteční stav, q_F – konečný stav

Varianty:

Univerzální turingův stroj – umožňuje změnu programu

Nedeterministický dosud nebyl sestrojen, sestrojení by vedlo ke konci veškeré asymetrické kryptografie, neví se, zda je to možné (problém P vs. NP), schopnost ve všech variantách nalézt jedno správné řešení, schopnost vytvářet nekonečné množství paralelních procesů

Paralelní – stejná vyčíslitelnost, vyšší výkon

Kvantový – založen na superpozice stavů

Deterministická přechodová funkce

$$\delta: (Q \setminus \{q_F\}) \times \Gamma \rightarrow Q \times (\Gamma \cup \{L, R\})$$

Nedeterministická přechodová funkce

$$\delta: (Q \setminus \{q_F\}) \times \Gamma \rightarrow Q \times 2^{(\Gamma \cup \{L,R\})}$$

L – posun hlavy o pozici doleva, R doprava, {L,R} ∉ Γ

15) Definujte konečný deterministický automat, rozhodněte, jaké vstupy přijímá a proč (viz tabule)

$\boldsymbol{M = (Q, \Sigma, \delta, q_0, F)}$ **F –** množina konečných stavů
$\delta: Q \times \Sigma \rightarrow Q$

Pro přijetí řetězce:

Veškeré znaky vstupní pásky byly přečteny
Automat skončil ve stavu, který je konečným stavem

Pro nepřijetí řetězce:

Veškeré znaky vstupní pásky byly přečteny
Automat skončil ve stavu, který **není** konečným stavem

Nedeterministický - Vše jako deterministický, jen přechodová funkce je definována jako $\delta: Q \times \Sigma \rightarrow 2^Q$
Je teoretický výpočetní model, který má schopnost ve všech variantách nalézt jedno správné řešení. Schopnost vytvářet nekonečné množství paralelních procesů

16) Definujte zásobníkový automat.

$\boldsymbol{P} = (Q, \Sigma, \Gamma, \boldsymbol{\delta}(q_i, a, \tau_m) \rightarrow \{(q_k, \tau_n), \dots\}, q_0, \tau_0, F)$

q_i – stav z množiny Q
a – symbol z množiny Σ nebo prázdný string

$\boldsymbol{\tau_m}$ je zásobníkový symbol, $\boldsymbol{\tau_n}$ – symbol, který nahrazuje $\boldsymbol{\tau_m}$ na vrcholu zásobníku, pokud je prázdný, poslední symbol zásobníku je odstraněn

$\boldsymbol{\tau_0}$ – počáteční stav zásobníku

Pokud automat odstraní poslední symbol ze zásobníku, potom se zastaví
Přijímá vstupní řetězec pokud vyprázdní zásobník, přečte veškeré vstupy vstupní pásky a skončí v konečném stavu

17) Definujte třídy složitosti P a NP. Jak se nazývá skupina výpočetně nejobtížnějších problémů. Jaké známe druhy automatů. Jaký je vztah mezi třídami P a NP?

Vyjadřují jak náročný výpočet je nezbytný, abychom problém vyřešili z pohledu výpočetního modelu TS

P – jsou schůdné algoritmy, běžící nejhůře v polynomiálním čase (PT). Jdou řešit TS.
NP – jsou neschůdné algoritmy. Jdou řešit v polynomiálním čase, pokud bychom měli k dispozici nedeterministický TS.
NP – úplný: nejtěžší úlohy z NP
NP – těžké: alespoň tak těžké jako nejtěžší z NP, nemusí být řešitelné TS

Problém ekvivalence P vs. NP – jeden z největších problémů současnosti v oblasti složitosti a matematiky vůbec
Deterministický a nedeterministický konečný automat, zásobníkový automat

Důkaz diagonalizace

Diagonalizace - mám množinu P, kde jsou všechny algoritmy, pomoci diagonalizace naleznu takový rozhodovací problém, který se s P alespoň v jednom řádku. P již ale obsahuje všechny případy – tj. nastává spor.

18) Existují problémy, které nelze vyčíslit? Co říká Church-Turingova teze?

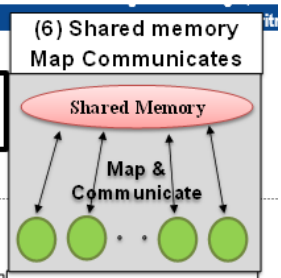
Ano – problém zastavení Turingova stroje
Church-Turingova teze říká, že každý algoritmus může být vykonán turingovým strojem

19) Distribuce výpočtů ve výpočetních gridech (např. map-reduce) !!

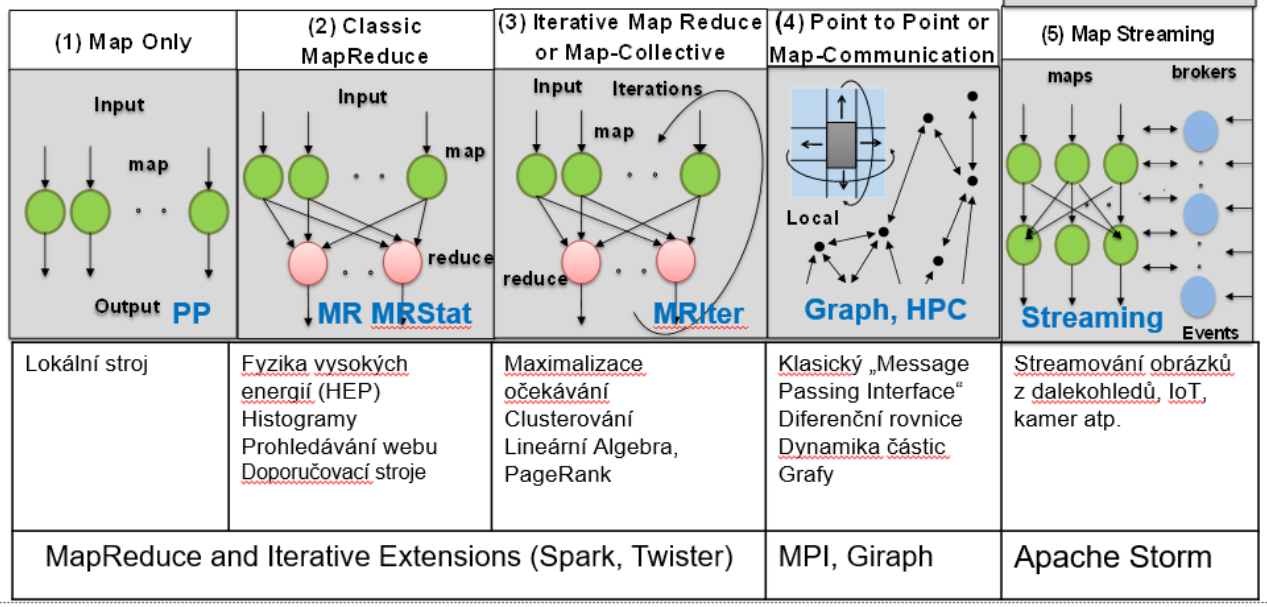
Jedná se o síť distribuovaných zdrojů zahrnující počítače, přepínače, směrovača, data a další.
Zdroje mohou být vlastněny různými organizacemi
Jsou řízeny speciálním SW, který umožňuje sdílení a správu zdrojů
Jsou formou „Super virtuálního počítače“

Architektury distribuce výpočtů

Obtížně paralelizovatelné algoritmy nad grafy



- Kliknutím vložíte text.



Lokální stroj	Fyzika vysokých energií (HEP) Histogramy Prohledávání webu Doporučovací stroje	Maximalizace očekávání Clusterování Lineární Algebra, PageRank	Klasický „Message Passing Interface“ Diferenční rovnice Dynamika částic Grafy	Streamování obrázků z dalekohledů, IoT, kamer atp.
MapReduce and Iterative Extensions (Spark, Twister)			MPI, Giraph	Apache Storm

20) Mějme problém 8 dam, rozmístění dam na šachovnici, aby se vzájemně nenapadaly. Pro s algoritmus a) navrhněte chromozom, b) navrhněte hodnotící (firness) funkci

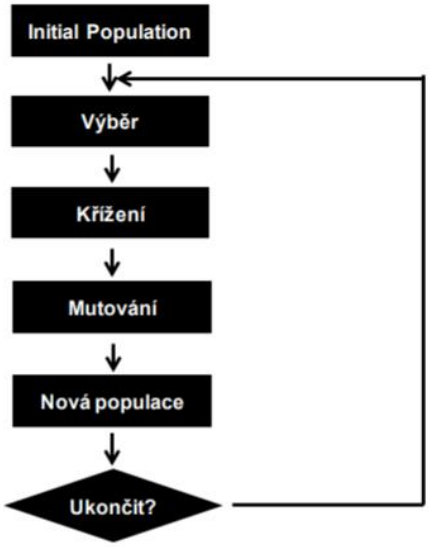
a) (4,6,8,3,1,7,5,2)

b) Chromozom reprezentující řešení N dam bude ohodnocen N – tedy 8, znamená že na sebe neútočí žádné dámy, pokud na sebe budou útočit 2 dámy, ohodnocení bude N-2, tedy 6

c) Standardizovaná fitness - přepočet hrubé fitness na určitou referenční hodnotu, optimální řešení je rovno nejvyšší hodnotě; $s(i,t) = r_{max} - r(i,t)$; $r(i,t)$ je hodnota hrubé fitness; r_{max} je referenční hodnota fitness

21) Napište obecný evoluční algoritmus, popište možnosti paralelizace genetických algoritmů.

```
begin
        t := 0;
        inicializujPopulaci P(t);
        ohodnoť P(t);
        while neníKonec do
                t := t + 1;
                P' := vyberJedince P(t);
                křižJedince P'(t);
                mutujJedince P'(t);
                ohodnoť P'(t);
                P := vytvořNovouPopulaci P,P'(t);
        od
end
```



Možnosti paralelizace:

- Operací nad jednou populací (výpočet fitness, selekce)
- Sub-populací
- Kombinace obou

Migrační model – populace rozdělena do více sub-populací vyvíjejících se nezávisle, po určitém počtu generací dochází k migraci jedinců

Globální model – nerozděluje se do sub-populací, průběh výpočtu genetického algoritmu je rozdělen do paralelních procesů, řídící proces provádí selekci a přiřazuje hodnocení jedincům, procesy křížení, mutace a výpočet fitness funkce řeší další procesy

Difúzní model – každý jedinec je řízen zvlášť a vstupuje do křížení jen s jedinci v okolí

Hybridní paralelizace

22) Uveďte princip evolučních algoritmů, objasněte křížení a uveďte jeho varianty

Evoluční algoritmy – jedná se o způsob optimalizace. Využívají modely:

- přirozený výběr (výběr silnějšího jedince, podle fitness funkce)
- náhodný genetický drift (mutace, decimuje jedince s vysokou fitness)
- reprodukční proces (křížení jedinců).

Spadají sem genetické algoritmy, genetické programování, evoluční strategie a evoluční programování.

Křížení – operátor kombinuje genetický materiál dvou rodičů. Výsledkem jsou dva potomci.
Mutace – operátor mutace pracuje s jedním jedincem a je běžně aplikován jako další operátor po křížení.
Úkolem křížení je posunout řešení blíže směrem k předpokládanému řešení a kombinuje výhody dvou potomků. Naopak mutace vytváří drobné odchylky a zanáší do řešení náhodnost neboli novou informaci.

N – bodové křížení
Operace křížení se provádí v jednom nebo více bodech chromozomu. U jednobodového křížení se navzájem vymění od určité pozice části rodičů a vzniknou tak 2 potomci, z nichž každý obsahuje část genů z obou rodičů. U vícebodových křížení dochází k výměně více úseků chromozomů.

Uniformní křížení
Další variantou rekombinačního operátoru. Tento operátor prochází dvojici nulového a jedničkového chromozomu a provádí výměnu jednotlivých genů. Vnáší do kódu žádanou diverzitu. Používá křížící masku, která udává, které geny se kříží, maska pro druhého rodiče je inverzní k masce prvního rodiče

23) Co je to rozhodovací strom? Pomocí jaké metriky se dělí uzly? Jaké znáte algoritmy dělení uzlů?

Binární rozhodovací strom s rozhodovacím procesem, vnitřní uzly představují např. otázku s odpovědí ano/ne a listy představují rozhodnutí

Metrika: Entropie, Informační zisk, GINI index

Algoritmy: ID3, C4.5, CART

24) Binární vyhledávací strom. Napiště pseudo kód: a) vyhledání hodnoty, b) vkládání hodnoty

a) vyhledání hodnoty

```
search(uzel, hodnota)
   if (uzel neexistuje)
         vrať prázdnou hodnotu //hodnota nebyla nalezena
   else if (hodnota je stejná jako uzel)
         vrať aktuální uzel
   else if (hodnota je menší než uzel)
         volej rekurzivně search na levý podstrom
   else //(hodnota je větší než uzel)
         volej rekurzivně search na pravý podstrom
```

b) vkládání

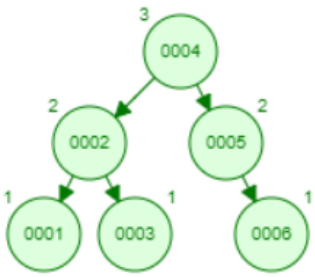
```
insert(strom, novy uzel)
   if (strom je prázdný)
         vlož na kořen
   else if(shoduje se s aktuálním uzlem)
         nedělej nic (duplicita)
   else if(nový uzel je menší než aktuální uzel)
         volej rekurzivně insert na levý podstrom
   else
         volej rekurzivně insert na pravý podstrom
```

25) Napiště kód pro průchod všech položek binárního vyhledávacího stromu.

Pre-order – vrací od kořene dolů: 4 2 1 3 5 6

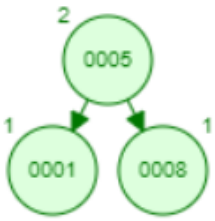
In-order – vrací od nejmenší hodnoty po největší (u BVS): 1 2 3 4 5 6

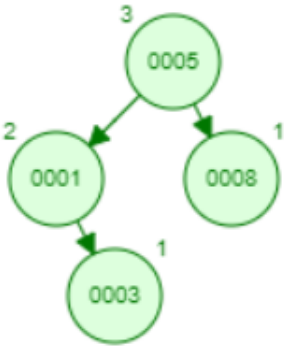
Post-prder – vrací od nejhlubšího z levého, pravého a kořen: 1 3 2 6 5 4

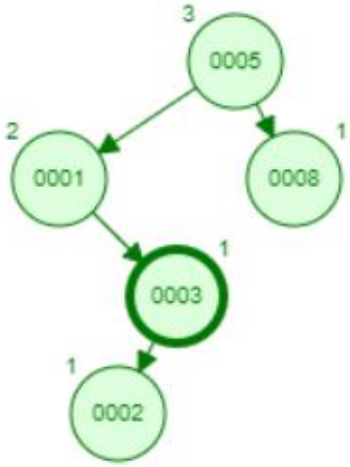


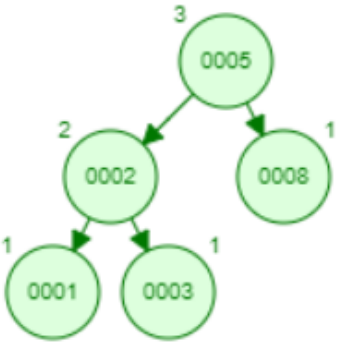

```
public void vypis(){
      vypis(koren);
}
private void vypis(Uzel u){
      if(u == null){
            return;
      }
      //in-order
      vypis(u.getLevy());
      System.out.println(u.getData());
      vypis(u.getPravy());

      //pre-order
      System.out.println(u.getData());
      vypis(u.getLevy());
      vypis(u.getPravy());

      //post-order
      vypis(u.getLevy());
      vypis(u.getPravy());
      System.out.println(u.getData());
}
```

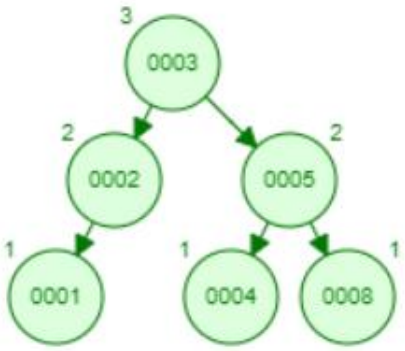
26) Do prázdného binární vyhledávácího stromu vyvažovaného AVL vložte: 8,5,1,3,2,4







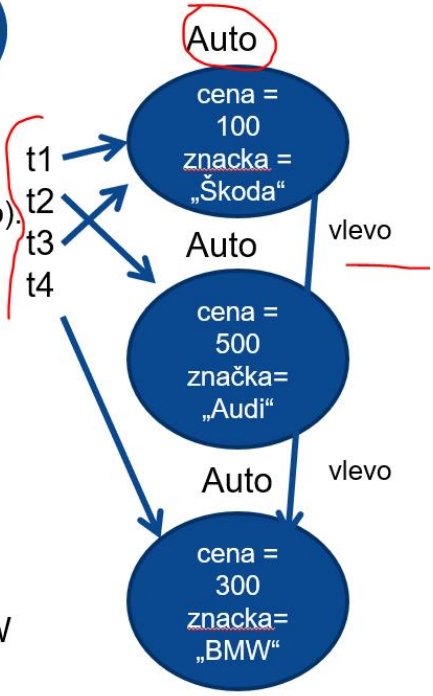




Informace v paměti

Návrh třídy a vytvoření objektu (3)

- **Zadání**: Rozšiřte předchozí příklad tak, aby obsahoval informaci o autě, se kterým sousedí vlevo
  (výsledek by měl odpovídat schéma vpravo).



1. S pomocí proměnné „t1“ zjistětě značku souseda
2. S pomocí proměnné „t1“ zjistěte značku sousedovic souseda
3. Předchozí program ponechte a prohoďte Audi a BMW

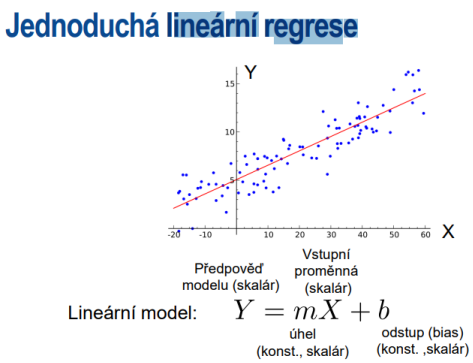
Nakreslete diagram tříd UML, který modeluje vztahy mezi následujícími třídami: obchod, pracovník prodeje, oddělení, manažer, zboží, katalog obchodu, web obchodu a zákazník.

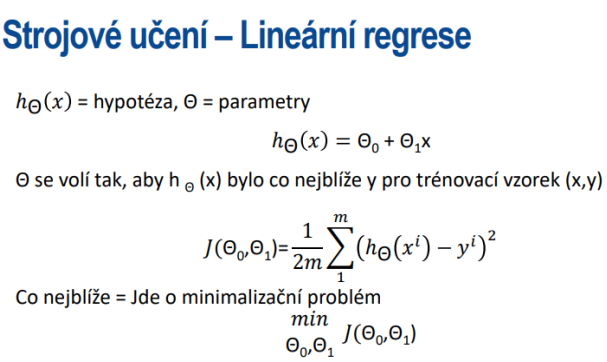
Napište algoritmus pro sečtení všech hodnot v lineárním seznamu.

```
public int spocitejHodnoty() {
        int sum = 0;
        Element tmp = first;
        while(tmp != null){
            //System.out.println(tmp.getData());
            sum += tmp.getData();
            tmp = tmp.getNext();
        }
        return sum;
    }
```

Uveďte definici konečného automatu včetně přechodové funkce. Jak se liší deterministický od nedeterministického?

Popište algoritmus lineární regrese. V čem se liší oproti definici neuronu neuronové sítě?
- algoritmus strojového učení





Existují nevyčíslitelné problémy? Uveďte důkaz.
Rozhodovací problém – je vstupní číslo liché?
Důkaz nevyčíslitelného problému – diagonalizace
Označme P jako množinu všech rozhodovacích problémů – patří sem jeLiche1()
Chceme program, který má na výstupu opačnou hodnotu algoritmu - jestliže množina P obsahuje všechny algoritmy, s D se liší minimálně v jednom řádku -> Což je spor (Důkaz pomocí diagonalizace)

Ano, problém zastavení Turingova stroje.

Napište metodu reverse(), která prohodí pořadí prvků v lineárním seznamu. POZN: pokud si nevíte rady, uveďte alespoň metodu „void vložNaZačátek()“ a „int odstraňPoslední“.

```
public Uzel reverse(Uzel node)
	{
		Uzel prev = null;
		Uzel current = node;
		Uzel next = null;
		while (current != null) {
			next = current.getDalsi();
			current.setDalsi(prev);
			prev = current;
			current = next;
		}
		node = prev;
		return node;
	}
```

```
public void print(Uzel node){….};
```

v Main: `Uzel head = ls.reverse(ls.prvni);  ls.print(head);`

Mějme abstraktní datovou strukturu strom. Napište kód, který vypíše všechny hodnoty ve stromu.

Definujte složitosti NP. Jaký je vztah mezi třídami P a NP?

Mějme směrovače a, b, c, d, e, f. V obrázku je vyznačeno zpoždění. Algoritmem vyberte linky, které mají nejnižší celkovou cenu.

Napište algoritmus pro vložení hodnoty na konec lineárního seznamu.

```
public void vlozNaKonec(int i) {
		Element tmp = first;
		while(tmp.getNext() != null) {
			tmp = tmp.getNext();
			if(tmp.getNext() == null) {
				Element a = new Element(i);
				tmp.setNext(a);
				return;
			}
		}
	}
```

Uvažujme genetické algoritmy, co je to a) ruletový výběr a jak se liší oproti turnajovému výběru? B) co je to standardizovaná fitness?

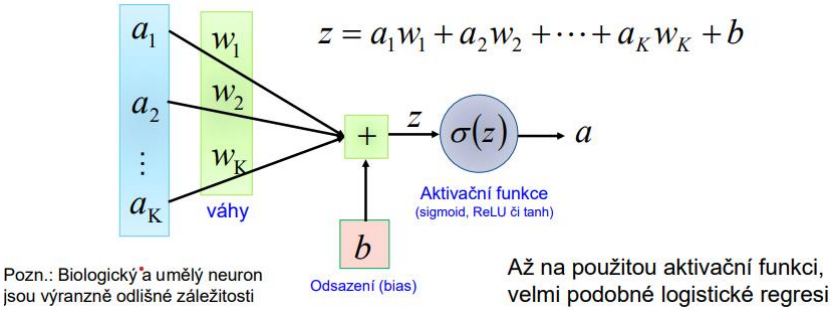
Ruletový výběr: pravděpodobnost výběru závisí na kvalitě jedince (kolik místa na ruletě zabírá). Pokud jedinec ostatní převyšuje výraznou měrou, nová populace bude tvořena téměř výhradně geny. K tomu se využívají techniky potlačení/- podpory.
Ruletový výběr, varianta 2: jedinci jsou seřazeni vzestupně podle hodnoty fitness a velikost místa je určena rovnicí. Tímto se potlačují nadprůměrní jedinci, kteří by negativně ovlivňovali další generace.

Turnajový výběr - náhodně se vybere n jedinců a postupným porovnáváním je vybrán nejlepší. Analogie turnaje mezi rivaly, 2 soupeři. V převážné většině případů nejvhodnější.
Standardizovaná fitness - přepočet hrubé fitness na určitou referenční hodnotu, optimální řešení je rovno nejvyšší hodnotě; $s(i,t) = r_{max} - r(i,t)$; $r(i,t)$ je hodnota hrubé fitness; r_{max} je referenční hodnota fitness

Popište umělý neuron. Popište maticový zápis a objasněte optimalizaci výpočtu odsazení.

Část neuronové sítě

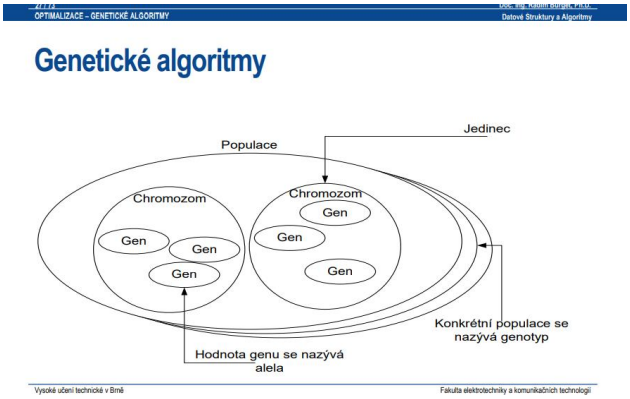
Umělý neuron $f: R^K \rightarrow R$



Pozn.: Biologický a umělý neuron jsou výranzně odlišné záležitosti

Až na použitou aktivační funkci, velmi podobné logistické regresi

Popište princip genetických algoritmů. Naznačte řešení problému 8 dam.

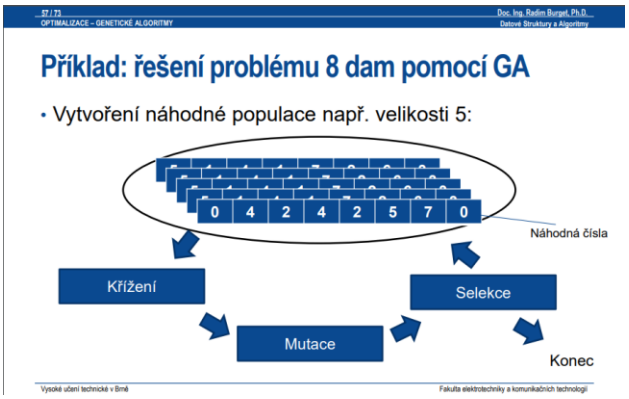


Rozdíl mezi individuem (fenotyp) a jeho reprezentací (genotyp).
Chromozom se dále dělí na geny
Gen na i-té pozici reprezentuje stejnou charakteristiku v každém jedinci.
Alela je hodnota, které může nabývat gen

Etapy návrhu algoritmu:
1) Reprezentace problému
2) Vytvoření počáteční populace
3) Vytvoření fitness funkce
4) Operátory selekce
5) Genetické operátory
6) Obnova populace
7) Ukončení algoritmu
8) Kontrola běhu algoritmu

Problém 8mi dam - na šachovnici 8x8 rozmístěte 8 dam tak, aby žádná neohrožovala jinou

- Konfigurace dam na šachovnici – Fenotyp
- Možná konfigurace chromozomu (čísel 1-8) – Genotyp
- Genotyp = Chromozom
- Chromozom obsahuje celkem 8 genů (8 dam).
- Gen může nabývat hodnot (alel) 1-8.



Mějme za úkol uspořádat hru čísla (patnáctka) s pomocí genetických algoritmů (ne BFS). a) Navrhněte chromozom, zvolte oprerace kombinace b) popičtě algoritmus evoluce c) uveďte způsob výpočtu fitness funkce

Mějme třídu A. Zakreslete výslednou podobu stavu paměti po provedení tohoto kodu.

Genetické programování. Neuronové sítě. Jsou schopny řešit stejný typ problému?

Mějme k dispozici 50 trojúhelníků, jejichž tvar, barvu a stupeň průhlednosti můžete měnit. Úkolem je rozmístit trojúhelníky tak, aby výsledný obrázek co nejvíce odpovídal vzoru. Řešte pomocí genetických algoritmů: a) navrhněte chromozom, b) navrhněte fitness funkci

Umět algoritmy Dijkstruv atd. může být zadán graf, nalést cestu. UML diagram

Umět z vyvažovaného stromu odstranit prvek.