**Kikiina**

**Státnicová**

**Směska**

**Ni-spol**

**1. Teorie grup: Grupoidy, pologrupy, monoidy a grupy. Podgrupy, cyklické grupy a jejich generátory.**

NI-MPI

* **Grupoid** – uspořádaná dvojice
  + = libovolná neprázdná množina
  + = binární operace nad (tzv. kule)
* **Pologrupa** – grupoid, pro který je asociativní
* **Monoid** – pologrupa, ve které existuje neutrální prvek
  + V monoidu existuje právě jeden neutrální prvek
* **Grupa** – monoid, ve kterém ke každému existuje inverzní prvek
  + V grupě má každý prvek právě 1 inverzní prvek
* **Abelovská grupa** – grupa, kde je komutativní

Uzavřená grupoid asociativní pologrupa monoid inverze grupa komutativní Abel. grupa

* **Podgrupa grupy** je je grupa
  + V každé grupě s alespoň 2 prvky existují alespoň 2 podgrupy
    - **Triviální podgrupy**
    - Ostatní podgrupy jsou **vlastní podgrupy**
  + **Průnik** podgrup je podgrupa
  + **Kritérium** podgrupovosti: je podgrupa , právě když
  + **Neutrální prvek** podgrupy je roven neutrálnímu prvku grupy
  + **Inverze** prvku v podgrupě je stejná, jako inverze stejného prvku v grupě
* **Řád grupy** – počet prvků množiny –
  + Podle řádu se dělí na **konečné a nekonečné grupy** (nekonečná )
  + **Lagrangeova věta** – buď podgrupa konečné grupy , potom řád dělí řád
  + **Sylowova věta** – buď grupa konečného řádu a číslo prvočíselný dělitel čísla . Pokud dělí (pro přirozené), pak grupa obsahuje podgrupu řádu
* je podg. obsah.
* **Grupa generovaná množinou** – podgrupa grupy
  + Množina je **generující množina** grupy
  + Pro jednoprvkovou množinu zavádíme značení , = generátor
  + je nejmenší podgrupa obsahující množinu
  + Všechny prvky lze získat pomocí „**grupového obalu**“
  + **Generátor** = prvek, jehož **mocněním** dostaneme všechny prvky grupy
* Grupa je rovna právě když a jsou nesoudělná čísla
* **Cyklická grupa** – existuje prvek = **generátor** cyklické grupy
* **Řád** prvku – buď prvek grupy . Pokud existuje , pak **nejmenší**  s touto vlastností je **řád prvku** . Pokud takové neexistuje, řád prvku je nekonečno
  + Řád prvku je roven řádu grupy
  + je cyklická, právě když , kde je liché prvočíslo a
* **Jak najít všechny generátory**
  + Je-li cyklická grupa řádu a nějaký její generátor, potom je také generátor tehdy, a jen tehdy, když a jsou nesoudělná
  + V cyklické grupě řádu je počet generátorů roven
    - = **Eulerova funkce** – každému přiřazuje počet přirozených čísel menších než , které jsou s ním nesoudělná
    - Takže pro prvočíslo je cyklická grupa řádu a má generátorů,
* Libovolná podgrupa cyklické grupy je opět cyklická grupa
* **Malá Fermatova věta** – pro libovolné a libovolné
  + Z důsledku Lagrangeovy věty: , kde je neutrální prvek

**2. Tělesa a okruhy: Základní definice a vlastnosti. Konečná tělesa. Okruhy polynomů, ireducibilní polynom.**

NI-MPI

* **Okruh** – , kde je neprázdná množina a binární operace na ní a platí:
  1. je **abelovská grupa** = **aditivní grupa** okruhu
     + Neutrální prvek = **nulový prvek** – značí se
     + Inverzní prvek vůči k  značíme
     + Lze definovat odčítání:
  2. je monoid = **multiplikativní monoid** okruhu
     + Je-li komutativní, je komutativní okruh
     + Neutrální prvek = **jednička**, značení 1
  3. Platí **distributivní zákon:**
* Základní vlastnosti okruhu
  + Násobení nulovým prvkem dává nulový prvek
  + Levý i pravý distribuční zákon pro odečítání:
* **Obor integrity** – okruh, ve kterém neexistují dělitelé nuly
  + **Dělitelé nuly** = nenulové prvky
* **Těleso** – okruh kde je abelovská grupa
  + Tuto grupu nazýváme **multiplikativní grupou** tělesa T
  + Pokud pro z tělesa platí , potom , nebo
    - Každé těleso je oborem integrity
* **Zobrazení** z okruhu/tělesa do okruhu/tělesa je **homomorfismus** těchto okruhů/těles, jestliže je homomorfismem příslušných aditivních a multiplikativních monoidů/grup a platí
  + Je-li navíc **bijekce** (prosté a na), jedná se o **izomorfismus** těchto okruhů/těles
  + Tělesa a nazýváme izomorfní, právě když existuje izomorfismus . V tomto případě je těleso izomorfní s tělesem
* **Konečné těleso** – těleso, které má konečný počet prvků
* **Řád tělesa** = počet prvků tělesa
* Základní příklad konečného tělesa – množina s operacemi **modulo prvočíslo** 
  + **-** aditivní grupa , multiplikativní grupa
    - - řád
      * Každý nenulový prvek je její **generátor**
      * Je grupou i pro neprvočíselné
    - - řád – není prvočíslo
      * Je **cyklická**
      * Počet generátorů závisí na řádu, je roven
* **Řád konečného tělesa** musí být mocnina prvočísla , kde je prvočíslo a je kladné celé číslo
  + Všechna tělesa řádu jsou **navzájem izomorfní**
* **Galois field** – těleso s prvky -
  + Každé konečné těleso je Galois Field
  + Prvočíslo = charakteristika tělesa
  + – aditivní grupa
    - Řád
    - Neutrální prvek
    - Pro není cyklická
  + – multiplikativní grupa
    - Řád – zas vynechávám 0, všechno by požrala
    - Neutrální prvek:
    - Inverzi lze nalézt pro každý prvek s REA v polynomiálním čase
    - Je vždy cyklická
* **Polynom nad okruhem**
  + Nad okruhem
  + … koeficienty polynomu – o ty nám jde
  + … **formální proměnná** polynomu
  + Pokud pro existuje , pak největší z  = **stupeň polynomu** , značeno
  + … nulový polynom – nedefinovaný stupeň
  + Abychom mohli dělat operace s polynomy, potřebujeme je umět s jejich koeficienty – lze vybudovat okruh polynomů nad libovolným okruhem (i tělesem)
* **Okruh polynomů** – množina všech polynomů nad okruhem spolu s operacemi **sčítání** a **násobení** definovanými předpisy

kde , tvoří okruh polynomů nad okruhem -

* + **Násobení** polynomů: buď těleso a nenulové polynomy. Platí
  + **Dělení** polynomů: buď těleso a nenulové polynomy. Pak existují jednoznačně určené polynomy takové, že

kde je buď nulový, nebo má stupeň ostře menší než stupeň

* + **Bézoutova rovnost** pro polynomy: Buďte a nenulové polynomy nad tělesem . Pak existují polynomy tak, že:
  + Buď těleso a polynom stupně . Prvek je **kořen polynomu** právě tehdy, když:

kde je stupně

* **Ireducibilní polynom** – buď stupně alespoň 1. Řekneme, že je ireducibilní nad okruhem , jestliže :
  + Mějme celé a prvočíslo . Označme počet monických polynomů stupně ireducibilních nad . Potom
    - **Monický polynom** – má za koeficient u nejvyšší mocniny jedničku
    - – **Möbiova funkce** definovaná pro celé :

**3. Funkce více proměnných: gradient, Hessián, definitnost matic, extrémy funkcí více proměnných bez omezení a s rovnostními omezeními.**

NI-MPI

**Parciální derivace**

* **Norma** na vektorovém prostoru je zobrazení splňující:

1. (trojúhelníková nerovnost)
   * Pro každé a všechny skaláry
   * Euklidovská norma na ():

* Reálná **funkce více proměnných** – zobrazení
  + … definiční obor, … obor hodnot
  + **Graf funkce** = množina:



* + **Okolí bodu** = buď a , -okolí bodu je množina
  + **Hromadný bod** = je hromadným bodem , pokud
    - Bod , který není hromadný, je izolovaný
* **Limita funkce více proměnných** – funkce , má limitu v hromadném bodě množiny , pokud:



* + Značení:



* + **Limita posloupnosti** – posl. má limitu , pokud
  + Mějme funkci . Funkce má v bodě limitu :



* + **Spojitost funkce** – funkce je spojitá v bodě , pokud:



* + - Funkce je spojitá, pokud je spojitá ve všech bodech definičního oboru
    - Formulace pomocí limity – je spojitá, pokud pro všechny neizolované body :



* **Parciální derivace** funkce ve směru (podle) v bodě , :



pokud tato limita existuje

* + Je to směrnice tečny ke grafu funkce ve směru osy
* **Gradient funkce** v bodě je vektor

Obsah obrázku Písmo, řada/pruh, text, rukopis

Popis byl vytvořen automaticky

* **Derivace ve směru** – , pak derivace ve směru v bodě , je:



* + Pokud existuje gradient v bodě (všechny parc. derivace jsou na nějakém okolí spojité), pak:



* **Parciální derivace 2. řádu** v bodě :



* + smíšená 2. parciální derivace
  + Opět zobrazení z podmnožiny
* **Hessova matice** – existují-li všechny 2. parciální derivace v bodě , zaznamenávají se do Hessovy matice (=Hessián):

Obsah obrázku text, Písmo, řada/pruh, snímek obrazovky

Popis byl vytvořen automaticky

* + Zaměnitelnost 2. parciálních derivací
    - Pokud jedna existuje a ta funkce je v  spojitá, potom druhá existuje, platí:



* **2. derivace ve směru** – lze derivovat ve směru v první derivaci ve směru – druhá parciální derivace ve směru v bodě :
  + - Existuje-li Hessova matice (existuje okolí takové, že má na něm spojité všechny druhé parciální derivace), potom:
* **Jacobiho matice** – matice prvních derivací
  + Máme
  + Jacobiho matice funkce (*psí*) je zobrazení :

Obsah obrázku Písmo, diagram, řada/pruh, snímek obrazovky

Popis byl vytvořen automaticky

pokud všechny derivace existují

* + Má na řádcích **složky gradientů** jednotlivých složek
* **Geometrický význam** gradientu
  + Gradient ukazuje směr (v definičním oboru) nejvyššího růstu funkce
  + V místech, kde je gradient nulový (stacionární body) hledáme extrémy
  + **Tečná nadrovina** – sjednocení tečen ve všech směrech (v bodě – tečná nadrovina v
    - Musí existovat gradient v
    - Rovnice nadroviny:



* + - **Normálový vektor**:



**Lokální extrémy – bez omezení**

* Reálná funkce má v bodě :
  + **Lokální minimum**, pokud:



* + **Ostré lokální minimum**, pokud:



* + **Globální minimum**, pokud:



* Je-li omezená a uzavřená, pak má spojitá funkce globální minimum a globální maximum
* **Nutná podmínka existence lokálního extrému** – nechť má v bodě parciální derivaci podle -té proměnné. Pokud má v bodě lokální extrém, pak:



* + Pokud existuje gradient v , pak exitence lokálního extrému implikuje, že
  + **Stacionární body** = body splňující
  + V úloze hledání extrémů jsou stacionární **body podezřelé z extrému = kritické body**
    - Počítají se mezi ně i body, kde neexistuje
* **Definitnost matic** – Řekneme, že je:
  + **Pozitivně semidefinitní**, pokud
  + **Negativně semidefinitní**, pokud
  + **Pozitivně definitní**, pokud ,
  + **Negativně definitní**, pokud,
  + **Indefinitní**, pokud není pozitivně ani negativně semidefinitní
    - Matice je indefinitní , a
* Pokud je **symetrická** matice, potom:
  + je **pozitivně semidefinitní** **nezáporná** všechna její vlastní čísla
  + je **pozitivně definitní** **kladná** všechna její vlastní čísla
  + je **negativně semidefinitní** **nekladná** všechna její vlastní čísla
  + je **negativně definitní** **záporná** všechna její vlastní čísla
  + je **indefinitní**  alespoň jedno **kladné** a alespoň jedno **záporné** vlastní číslo
* **Sylvestrovo kritérium**: je symetrická matice. Pro je čtvercová matice v levém horním rohu
  + je **pozitivně definitní** **determinant** všech matic je **kladný**
  + je **negativně definitní** **determinant** je **záporný** pro  **liché** a **kladný** pro **sudé**
* Jak poznat **indefinitnost** – pokud má na diagonále 2 prvky s **různým znaménkem**, pak je indefinitní
* Postačující podmínka existence extrému a sedlového bodu – nechť je stacionární bod funkce Nechť existuje okolí bodu takové, že má na okolí spojité všechny 2. parciální derivace, potom:
  + Je-li **pozitivně definitní**, pak je **ostré lokální minimum**
  + Je-li **negativně definitní**, pak je **ostré lokální maximum**
  + Je-li **indefinitní,** pak je **sedlový bod**
* **Nutná podmínka existence lokálního extrému** – nechť je stacionární bod funkce Nechť existuje okolí bodu takové, že má na okolí spojité všechny 2. parciální derivace, potom:
  + Je-li **lokální minimum**, pak je **pozitivně semidefinitní**
  + Je-li **lokální maximum**, pak je **negativně semidefinitní**
  + Tvrzení nelze obrátit
* Postup **analytického hledání extrémů**:

1. Najít **body podezřelé z extrému** = stacionární body a body, kde alespoň jedna parciální derivace neexistuje
2. Nalézt **Hessovu matici** v bodě podezřelém z extrému, pokud ta je
   1. **Pozitivně definitní**, pak je bod bodem **ostrého lokálního minima**
   2. **Negativně definitní**, pak je bod bodem **ostrého lokálního maxima**
   3. **Indefinitní,** pak je bod **sedlovým bodem** (takže není extrém)
   4. V ostatních případech je třeba rozhodovat jiným způsobem

**Lokální extrémy – rovnostní omezení**

* **Úloha vázaného extrému** (minima) s **rovnostní podmínkou** = minimalizuj za podmínky
  + … objektivní / účelová / minimalizovaná / optimalizovaná funkce
  + … **rovnostní podmínka** / vazba
  + Jsou-li všechny funkce lineární, je to úloha lineárního programování
* **Množina přípustných řešení**:



* + Máme úlohu hledání a bodů , pro které
  + Hledáme body lokálního minima vzhledem k množině : (pro nějaké okolí



* **Lagrangeova funkce** pro danou úlohu:

Obsah obrázku Písmo, text, bílé, typografie

Popis byl vytvořen automaticky

* + **Lagrangeovy multiplikátory** …
* **Postačující podmínka existence ostrého lokálního minima pro rovnostní vazby** – nechť mají spojité všechny 2. parciální derivace na nějaké otevřené nadmnožině .

Pokud dvojice splňuje podmínky:

1. (0. derivace)
2. (1. derivace) 
3. (2. derivace) pro každý (sloupcový) vektor splňují platí:



kde je Hessova matice funkce vzhledem k proměnným

potom je bodem **ostrého lokálního minima**

* + Body i. a ii. jsou ekvivalentní rovnosti

**4. Integrál funkcí více proměnných (Darbouxova konstrukce).**

NI-MPI

**Teorie 1D a 2D integrálu**

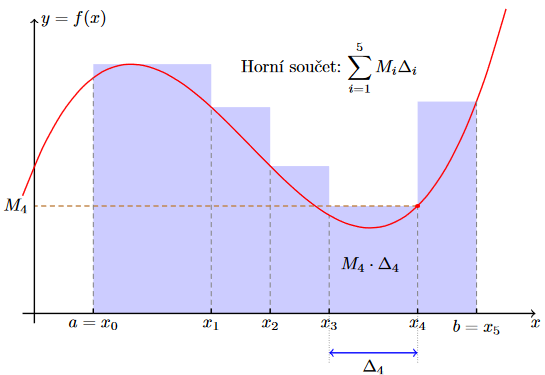
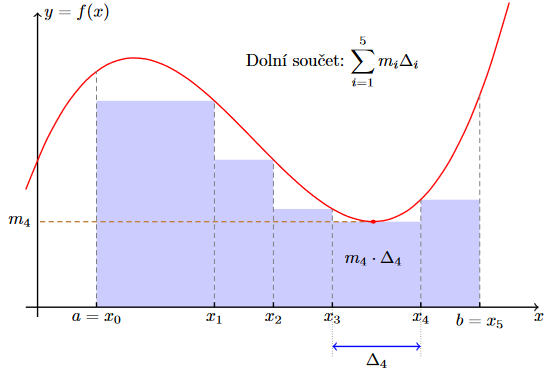
* **Integrál** = nástroj pro výpočet obsahu pod grafem nějaké funkce
* **Rozdělení intervalu** = konečná množina taková, že
  + … **dělící body** intervalu
  + … **norma rozdělení**
* **Darbouxův součet** (horní/dolní) – nechť je definovaná na a je jeho rozdělení. Označme:



Potom



je **horní a dolní (Darbouxův) součet** funkce při rozdělení



* + **Horní Darbouxův integrál** ( na ):



* + **Dolní darbouxův integrál** ( na ):



* + Pokud , nazveme tuto hodnotu **Darbouxovým integrálem** na :



* + - je **(Darbouxovsky) integrabilní** na
  + Posloupnost rozdělení je **normální**, pokud pro její normy platí:



* + Buď  **spojitá** na . Potom existuje . Je-li normální posloupnost rozdělení, potom



existují a jsou rovny

* + **Aditivita integrálu**:



* + **Multiplikativita integrálu**:



* **Primitivní funkce** – nechť funkce je definována v intervalu , kde . Funkci splňující podmínku

nazýváme primitivní funkcí k funkci v intervalu

* **2D integrál nad obdélníkovou oblastí**
  + Máme 2 rozdělení . je rozdělením
  + Označme:



* + **Horní Darbouxova suma** vzhledem k :



* + **Dolní Darbouxova suma** vzhledem k :



* + **Horní Darbouxův integrál** (funkce na ):



* + **Dolní Darbouxův integrál** (funkce na ):



* + (**dvojitý) Darbouxův integrál**:



* + - je Darbouxovsky integrabilní na
  + Posloupnost rozdělení je normální, pokud jsou obě původní rozdělení normální
* Buď integabilní funkce na . Pokud existuje jeden z integrálů



potom je roven dvojnému integrálu

* + **Výpočet dvojného integrálu** – funkci nejdřív zintegrujeme vzhledem k jedné proměnné a druhou považujeme za konstantu, výsledek (získaný pomocí Newtonovy formule) potom závisí už jen na jedné proměnné, vzhledem ke které se provede 2. integrace
* **Vlastnosti dvojného integrálu**
  + Množina **míry nula** – je pro hodnotu integrálu zanedbatelná
  + Pokud má průnik a míru nula, pak
  + Pokud , pak
  + Pro reálné a integrabilní :

Obsah obrázku Písmo, typografie, rukopis, kaligrafie

Popis byl vytvořen automaticky

**Metody pro výpočet 1D a 2D integrálu**

* **Newtonova formule**:



* **Per partes** pro určitý integrál:



* **Substituce** – funkce , a její derivace spojité na , spojitá na . Potom



* **Výpočet dvojného integrálu nad obecnou oblastí**

Obsah obrázku diagram, Vykreslený graf, řada/pruh, text

Popis byl vytvořen automaticky

* + 2 typy oblastí:
    - **Typ 1** – z intervalu omezené spojitými fcemi a splň.
    - **Typ 2** – z intervalu omezené spojitými fcemi a splň.
  + Pokud dané integrály existují, platí pro oblast :
    - Je-li typu 1, pak:

Obsah obrázku text, Písmo, rukopis, řada/pruh

Popis byl vytvořen automaticky

* + - Je-li typu 2, pak:

Obsah obrázku text, Písmo, rukopis, řada/pruh

Popis byl vytvořen automaticky

* **Substituce ve dvojném integrálu**
  + **Jacobiho matice** – matice prvních derivací
    - ****Máme
    - Jacobiho matice funkce (*psí*) je zobrazení :

Obsah obrázku Písmo, symbol, diagram, bílé

Popis byl vytvořen automatickyObsah obrázku Písmo, diagram, řada/pruh, snímek obrazovky

Popis byl vytvořen automaticky

pokud všechny derivace existují

* + - Má na řádcích **složky gradientů** jednotlivých složek :
      * Jsou to jednotlivé funkce
  + **Věta o substituci** – nechť je omezená uzavřená množina na . Nechť má spojité všechny parciální derivace (všech složek) na nějaké otevřené nadmnožině množiny a skoro všude na platí, že:
    - je **bijekce**
    - je **nenulový**

Potom pro každou spojitou funkci platí:

Obsah obrázku Písmo, text, typografie, kaligrafie

Popis byl vytvořen automaticky

kde

* + Transformace do jiného souřadnicového systému – například z kruhové výseče dělám obdélník
    - Pokud nemám ani oblast typu 1, ani oblast typu 2
  + Determinant je tam, protože transformace může změnit objem – nutno kompenzovat

**5. Numerická matematika: reprezentace čísel v počítači, chyby vznikající při výpočtech s pohyblivou řádovou čárkou, podmíněnost a stabilita numerických algoritmů.**

NI-MPI

**Strojová čísla**

* **Standard IEEE-754** – strojové číslo lze reprezentovat znaménkem a celými kladnými čísly a
* Pro celá čísla – vědecký zápis čísel v binární bázi: -
  + … **mantisa** – pevný počet cifer – platné cifry
  + … **exponent** – pevný počet cifer
* Omezení reprezentovatelných čísel podle přesnosti:

Obsah obrázku text, snímek obrazovky, Písmo, číslo

Popis byl vytvořen automaticky

* Určení reprezentované hodnoty :
  + Pokud a , pak
  + Pokud a , pak
  + Pokud a , pak … **normalizovaná čísla**
  + Pokud a , pak … **subnormální čísla**
  + Pokud a , pak
* **Skrytá jednička** – pro normalizované číslo se uloží o 1 platnou cifru více, než kolik je délka
  + Nemusí se reprezentovat, protože normalizovaná čísla vždycky začínají na 1
* Čísla, která lze takto reprezentovat = **strojová čísla**
* **Množina strojových čísel** – počet strojových čísel je závislý na a
  + = bias = ;offset
  + charakterizováno pomocí **strojové přesnosti** (machine epsilon) = vzdálenost čísla od nejbližšího většího čísla v :



* + - Pro jednoduchou přesnost platí
  + **Vzdálenost** libovolného normalizovaného čísla od jeho nejbližších sousedů z  je nejméně a nejvíce
* Pokud o reprezentaci čísel mimo rozsah **přetečení** / **podtečení**

**Chyby při výpočtech s pohyblivou řádovou čárkou**

* **Typy chyb**
  + Chyba **modelu** – mat. model úlohy je nějak zjednodušený, nebo se používají průměrné místo aktuálních hodnot
  + Chyba **dat** – data z měření bez absolutní přesnosti
  + Chyba **algoritmu** – algoritmus nemusí najít v konečném počtu kroků přesné řešení
  + **Zaokrouhlovací chyba** – při samotném výpočtu dochází např. k chybám v aritmetických operacích
* **Zaokrouhlovací chyby** - je přibližnou hodnotou čísla
  + **Absolutní chyba** reprezentace pomocí =
  + **Relativní chyba** pro reprezentace pomocí =
* **Zaokrouhlovací jednotka** – meze pro relativní chybu - (pro jednoduchou přesnost)
  + Zaokrouhlování směrem k 0 – usekneme bity přeskakující délku mantisy
* **Aritmetické operace** a chyba při jejích provádění
  + značí operaci sčítání, násobení, odčítání nebo dělení
  + Pokud nedojde k přetečení nebo podtečení, platí:



* + Operací nemusíme dostat strojové číslo – obecně reálné číslo, které je třeba zaokrouhlit
  + **Ztráta platných cifer**
    - Velké problémy – **krácení** – lze se mu vyhnout:
      * Přeformulováním problému tak, aby nedocházelo k odčítání
      * Použitím rozvojů funkcí do řad
      * Použitím jiných rovností
    - **Odečítání** – nechť a jsou normalizovaná strojová čísla a . Pokud pro nějaká kladná celá a , tak platí, že nejvíce a nejméně platných binárních bitů je ztraceno při provedení odčítání
  + **Původy zaokrouhlovacích chyb –** zaokrouhlovací chyby jednotlivých operací a jejich **kumulace, krácení**

**Numerické algoritmy**

* **Přímá metoda** – počítá řešení v konečném počtu kroků tak, že v teoretické absolutní přesnosti dává (přesné) řešení
  + Příklady:
    - GEM (Gaussova eliminační metoda)
    - Hledání inverze matice pomocí GEM
    - Hledání vlastních čísel
  + Není „samoopracující“ – když v jednom kroku vznikne chyba, už se jí nemusíme zbavit
* **Iterační metody** – hledají přibližná řešení matematických problémů tak, že konstruují posloupnost přibližných řešení
  + Každé další přibližné řešení je odvozeno z předchozího: pro a zobrazení
  + Zobrazení je voleno tak, aby posl. měla limitu, která je skutečným řešením dané úlohy
    - **Stacionární metoda** = pokud je neměnné pro všechny iterace
  + Příklady:
    - Richardsova metoda
    - Jacobiho metoda
    - Gauss-Seidelova metoda / SOR

**Podmíněnost numerických algoritmů**

* … numerický algoritmus
* … teoretický (přesný) výstup
* … vstupní data
* … výsledek v konečné (strojové) aritmetice
* **Dopředná / přímá chyba** - … **odchylka** spočítaného řešení od přesného řešení
  + Chyba výsledku
* **Zpětná chyba** - , je nejmenší číslo v normě, které to splňuje
  + Co musím zadat na vstup, abych dostala to, co jsem dostala
  + Promítnutí chyby algoritmu do jeho vstupu
* Pokud je pro všechny vstupy zpětná chyba relativně malá (podle kontextu), algoritmus je **zpětně stabilní**
* **Podmíněnost úlohy / algoritmu** – vyjadřuje závislost změny výstupu na změně vstupních dat
  + **Relativní číslo podmíněnosti**:

Obsah obrázku text, Písmo, řada/pruh, snímek obrazovky

Popis byl vytvořen automaticky

* + … zkoumaný definiční obor
  + úloha je dobře podmíněná – změna na vstupu málo změní výsledek
  + velké úloha je špatně podmíněná – při malé změně se výsledek změní hodně
* **Soustavy lineárních rovnic** - lineárních rovnic pro neznámých
  + Zápis
    - … regulární matice soustavy
    - … vektor pravých stran
    - … hledané řešení
* **Norma na vektorovém prostoru** = zobrazení splňující (pro každé skalár atd.):
  + **Přidružená maticová norma** matice k vektorové normě na :



* + - Definice suprema neprázdné omezené množiny :



* + - je norma a platí pro ni (pro všechny ):
      * ( … jednotková matice)
      * … konzistence normy
      * … submultiplikativita
* **Podmíněnost úlohy**:



* + „**relativní chyba v řešení** soustavy je **menší, než relativní chyba pravé strany** vynás.
  + … **číslo podmíněnosti** matice
    - Čím větší, tím horší podmíněnost a hrozí větší chyba
  + Takhle to dopadne, když je lehce změněna o perturbaci
    - Změna v řešení označena , potom platí:



* + - Z toho se dá odvodit ta nerovnost výše přes pravidla pro normu, ale vypisovat to nechcu

**Iterační numerické metody**

* Cílem je algoritmus, který konstruuje **posl. vektorů** , která se **„blíží“ přesnému řešení**
* Postup:

1. Startovací vektor zvolíme náhodně
2. Zvolíme **regulární matici**  (podle metody)
3. počítáme podle:

* Kdyby byla posloupnost konvergentní s limitou , potom je hledané řešení
* **Konvergence – volba Q**
  + Rovnost dosadíme:

Obsah obrázku Písmo, text, rukopis, kaligrafie

Popis byl vytvořen automaticky

* + - je vektor splňující a je jednotková matice
  + **Vektor chyby**:
  + Potom platí:



* + Potřebujeme , pro které
    - Chceme, aby se blížilo k 0¨
  + **Spektrální poloměr** matice : = absolutní hodnota největšího vlastního čísla (abs):



* + Nechť . Potom platí



* + - Máme zajištěnou konvergenci iterační metody, právě tehdy, když
    - Všechna vlastní čísla jsou v absolutní hodnotě
* **Ukončení iterace** – v kroku , dosáhne-li požadované přesnosti posloupnost je ostře klesající a iteraci lze zastavit, když nastane:



* + … uživatelem zadaný parametr
  + Nepraktické – nemáme
  + V kroce napočítámě reziduum , získáme **kritérium konvergence**:



* + Někdy místo rezidua méně náročné kritérium
  + V praxi – maximální počet iterací, po překročení metoda selže
  + Tady vzniká chyba algoritmu
* **Konkrétní metody – volby** 
  + prvky matice definujeme matice jako:

Obsah obrázku text, Písmo, řada/pruh, diagram

Popis byl vytvořen automaticky

* + , takže platí
  + = lower (pod diagonálou), D jakože diagonála, U = upper (nad diagonálou)
  + **Richardsonova metoda**:
    - Iterace:



* + - Konvergenci kontroluje matice
      * musí být blízko , aby byl třeba rozdíl pod
  + **Jacobiho metoda**:
    - Iterace:



* + - Konvergence kontrolována maticí
    - Postačující podmínka – je **diagonálně dominantní** Jacobiho metoda konverguje pro všechny volby
    - Matice je diagonálně dominantní, pokud pro každý řádek platí, že součet absolutních hodnot prvků vyjma diagonálního je menší než absolutní hodnota diagonálního prvku
  + **SOR / Gauss-Seidel metoda**:
    - **Gauss-Seidel**:
    - **SOR**:
    - … k urychlení konvergence, nenulové
    - Iterace:



* + - SOR konverguje, pokud a je symetrická a pozitivně definitní s kladnými prvky na diagonále

**6. Testování statistických hypotéz. T-testy, testy nezávislosti, testy dobré shody.**

NI-VSM

* **Náhodný výběr** z rozdělení = -tice stejně rozdělených nezávislých náhodných veličin (iid) s distribuční funkcí
* **Realizace náhodného výběru** = -tice konkrétních pozorovaných čísel
* **Kroky statistického uvažování**
  + **Odhad tvaru** rozdělení
  + **Odhad parametrů** rozdělení
    - **Bodový** odhad
    - **Intervalový** odhad
  + **Testování hypotéz** – ověření správnosti modelu
    - **Testy dobré shody** – ověřujeme hypotézy o tvaru pravděpodobnostního rozdělení
      * „má veličina normální rozdělení?“
    - **Parametrické testy** – tvoříme hypotézu o parametru a na základě dat se snažíme rozhodnout, zda je možné hypotézu zamítnout
      * „“

**Intervaly spolehlivosti**

* Zajímá nás interval, ve kterém leží skutečná hodnota parametru s danou pravděpodobností
* Interval určený statistikami a , splňující:

se nazývá **oboustranný interval spolehlivosti** (konfidenční interval)

* Interval , resp. určený statistikou , splňující:

se nazývá **horní/dolní (jednostranný) interval spolehlivosti**

* = **dolní/horní mez** intervalu spolehlivosti
* = **hladina** spolehlivosti
* Pro oboustranný interval spolehlivosti volíme tak, aby platilo
* Interval spolehlivosti pro **střední hodnotu při známém rozptylu**:
  + … **kritická hodnota** standardního normálního rozdělení
* Interval spolehlivosti pro **střední hodnotu při neznámém rozptylu** 
  + Neznáme odhadujeme ho pomocí **výběrového rozptylu**
  + Oboustranný interval spolehlivosti:
  + = kritická hodnota **studentova rozdělení** s  stupni volnosti
* Interval spolehlivosti pro **rozptyl**
  + Náhodný výběr z **normálního rozdělení**
  + Využijeme **výběrový rozptyl**
  + Oboustranný interval spolehlivosti
  + = **kritická hodnota rozdělení**  s  stupni volnosti na hladině
  + Na rozdíl od předchozích platí **POUZE pro normální rozdělení**

**Hypotézy a jejich testování**

* **Náhodný vektor** s nějakým rozdělením
  + Tvrzení o tomto rozdělení s neznámou platností = **hypotéza**
* **Testování hypotéz** – ověřování platnosti na základě pozorování hodnot
  + **Nulová hypotéza** = tvrzení, o kterém chceme rozhodovat
  + **Alternativní hypotéza** = opačné tvrzení, které v rozhodovacím procesu stavíme proti
  + Rozhodovací proces je založen na hod. , na jehož základě **zamítneme**/**nezamítneme** hypotézu
* Chyby při testování hypotéz
  + **Chyba 1. druhu** – zamítneme , i když platí
  + **Chyba 2. druhu** – nezamítneme , i když neplatí
* Výsledek – testujeme proti na hladině významnosti
  + **(ne)zamítáme** ve prospěch
* **Kritický obor** = množina realizací , pro které testování na hladině skončí zamítnutím
  + **zamítáme** na hladině
  + **nezamítáme** na hladině
* **P-hodnota** = minimální hladina významnosti , na které lze hypotézu zamítnout
  + Je-li p-hodnota menší než naše , zamítáme
  + P-hodnota je **horní mez** pro pravděpodobnost, s jakou bude při platnosti nulové hypotézy další realizace stejně příznivá zamítnutí, jako ta aktuální -
    - … možné rozdělení

**Parametrické testy**

* Určen náh. výběr z rozdělení určeno par. n. v. jsou iid s tímto rozd.
* Chceme testovat jednoduchou parametrickou hypotézu proti **oboustranné alternativě**:
  + **proti**  pro konkrétní hodnotu
* oboustranný interval spolehlivosti pro parametr sestavený na základě náhodné veličiny
  + Zamítneme , pokud
  + Nezamítneme , pokud
* Parametrické testy proti **jednostranné alternativě**:
  + **proti**
* **Jednostranný interval** spolehlivosti typu odpovídajícího alternativní hypotéze – horní interval spolehlivosti
  + Zamítneme , pokud
  + Nezamítneme , pokud
* Pro proti analogicky

**Testy o parametrech normálního rozdělení**

* náhodný výběr z
* Test proti alternativě na hladině významnosti
  + **Známý rozptyl** zamítneme, pokud neleží v intervalu
  + **Neznámý rozptyl** zam., pokud neleží v intervalu
* Test proti alternativě na hladině významnosti
  + zamítneme, pokud neleží v intervalu
* **Jednostranné** analogicky
  + **známý** rozptyl:
  + **neznámý** rozptyl:
  + :

**Testové statistiky**

* **Testová statistika** = sestrojíme statistiku = funkci náhodného vektoru , u které při platnosti nulové hypotézy známe její rozdělení
* V oblasti možných hodnot vybereme podmnožinu , pro kterou:

= při platnosti má hodnoty v s pravděpodobností nejvýše

* Zároveň obvykle chceme, aby
* Testování hypotéz: **Zamítneme** , jestliže **nezamítneme** , jestliže

**Jednovýběrové testy o střední hodnotě a rozptylu**

* **Testy o střední hodnotě normálního rozdělení**
  + náhodný výběr z + předpoklad, že známe

Pro testy o hodnotě porovnávané s uvažujeme testovou statistiku

* + - , kde
  + Pro test proti na hladině :

* + Pro test proti na hladině

podmínka na zamítnutí je stejná jako při testu na konfidenčním intervalu

* + - Je tady jen , ne , pozor
* **Testy o parametrech normálního rozdělení**
  + náhodný výběr z
  + **Test o střední hodnotě** – testová statistika a kritické obory při známém rozptylu :

**Obsah obrázku text, snímek obrazovky, Písmo, řada/pruh

Popis byl vytvořen automaticky**

* + **Test o střední hod**. – test. statistika a krit. obory při neznámém rozptylu (**jednovýběrový t-test**):

**Obsah obrázku text, Písmo, snímek obrazovky, řada/pruh

Popis byl vytvořen automaticky**

* + Testy **o rozptylu** na hladině významnosti :

**Obsah obrázku text, Písmo, snímek obrazovky, řada/pruh

Popis byl vytvořen automaticky**

**Párový t-test**

* Náhodný výběr z nějakého 2D rozdělení s neznámým vektorem středních h.
* Testujeme hypotézu proti
* jsou iid se střední hodnotou
  + Předpoklad, že kde neznáme

**Obsah obrázku text, Písmo, snímek obrazovky, číslo

Popis byl vytvořen automaticky**Lze převést na **párový t-test**

= Jednovýběrový t-test hypotézy proti

* + … výběrový rozptyl veličiny Z

**Dvouvýběrový t-test**

* **Náhodné výběry**  z a z (nezávislé)
* Testujeme hypotézu proti
* Test na základě stat., která má při plat. **studentovo rozdělení** s určitým počtem stupňů volnosti
  + Záleží, jestli (homoskedasticita)
  + **Obsah obrázku Písmo, text, řada/pruh, rukopis

    Popis byl vytvořen automatickyObsah obrázku text, Písmo, snímek obrazovky, řada/pruh

    Popis byl vytvořen automatickyStejné rozptyly :**
  + **Obsah obrázku text, Písmo, řada/pruh, snímek obrazovky

    Popis byl vytvořen automatickyObsah obrázku text, Písmo, řada/pruh, snímek obrazovky

    Popis byl vytvořen automatickyRůzné rozptyly :**

**F-test rovnosti rozptylů**

* Nezávislé náhodné výběry z a z
* **Obsah obrázku text, snímek obrazovky, Písmo, řada/pruh

  Popis byl vytvořen automaticky**Chceme testovat hypotézy porovnávající a
* … kritická hod. Fisher-Snedecorova F-rozdělení s  a stupni volnosti, které splňuje
* Citlivý na normalitu a – při nejistotě např. Levenův test

**Testy dobré shody**

* D.n.v. nabývající hodnot s pravděpodobnostmi – rozdělení :
* **Multinomické rozdělení**
  + Provedeme náhodný výběr z rozdělení – můžeme výsledek až na pořadí zaznamenat pomocí **četností**, s jakými jednotlivé hodnoty nastaly
    - Dostaneme náhodné veličiny
  + **Multinomické rozdělení** = rozdělení tohoto náhodného vektoru
    - Obsah obrázku text, Písmo, řada/pruh, snímek obrazovky

      Popis byl vytvořen automatickyZnačení , určeno pstmi:
  + … **binomické** rozdělení
  + Vlastnosti multinomického rozdělení
    - Buď
    - Podmíněná rozdělení podmnožin složek při fixovaných hodnotách zbylých složek jsou opět multinomická
    - Marginální rozdělení jsou binomická:
* **Obsah obrázku Písmo, text, řada/pruh, bílé

  Popis byl vytvořen automatickyPearsonova statistika** – buď . Pak Pearsonova statistika

má při asymptoticky rozdělené

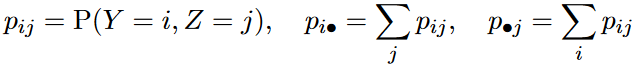
* + … **naměřené četnosti**
  + … **teoretické četnosti**
* **Test při známých parametrech**
  + Testování **shodnosti diskrétních rozdělení**
  + Náhodný výběr o velikosti z diskrétního rozdělení
    - Četnosti hodnot mají multinomické rozdělení
  + Testujeme hypotézu , že skutečné hodnoty pravděpodobností
  + Obsah obrázku text, Písmo, řada/pruh, snímek obrazovky

    Popis byl vytvořen automatickyProvedení testu:
    - … kritická hodnota  **rozdělení s  stupni volnosti**
  + Test je asymptotický, takže lze použít jen pro dostatečně velký rozsah výběru :
* **Test při neznámých parametrech**
  + Obecná situace, kdy:
    - : „náhodný výběr pochází z rozdělení , které může záviset na neznámé hodnotě nějakého parametru
    - : náhodný výběr pochází z jiného rozdělení (mimo parametrickou třídu )
  + **Převod na test hypotézy pro multinomiální rozdělení:**
    - Rozklad do kintervalů
    - **Obsah obrázku Písmo, řada/pruh, text, bílé

      Popis byl vytvořen automaticky**Četnosti naměřených hodnot v jednotlivých intervalech , mají multinomické rozdělení , kde
    - : skutečné hodnoty pravděpodobností jsou a mohou záviset na **neznámém m-rozměrném parametru** , jehož hodnotu při testování také **odhadujeme**
    - **=** hodnota minimalizující
    - Bodový odhad = **odhad metodou minimálního**
      * Obsah obrázku text, Písmo, řada/pruh, snímek obrazovky

        Popis byl vytvořen automatickyPro něj má statistika asymptoticky rozdělení
  + Provedení testu (při )
    - Počet stupňů volnosti = # chlívků = # odhadovaných parametrů – 1
    - Zas platí

**Test nezávislosti v kontingenčních tabulkách**

* Máme náhodný vektor s diskrétním rozdělením hodnoty , hodnoty
* ****Sdružené a marginální pravděpodobnosti:
* Ještě máme náhodný výběr z  o velikosti
  + počet výsledků, kdy nastala dvojice ,
  + Náhodné veličiny mají sdružené multinomiální rozdělení s parametrem a pravděpodobnostmi
* **Kontingenční tabulka** = náhodná matice rozměru se složkami
  + **Marginální četnosti**:

**Obsah obrázku text, snímek obrazovky, číslo, Písmo

Popis byl vytvořen automatickyObsah obrázku Písmo, text, řada/pruh, bílé

Popis byl vytvořen automaticky**

* + **Obsah obrázku Písmo, řada/pruh, text, bílé

    Popis byl vytvořen automaticky**Pro **celkový počet** platí:
  + Chceme testovat nezávislost veličin a
    - Pravděpodobnosti funkcemi marginálních pravděpodobností
    - **Obsah obrázku Písmo, číslo, bílé, Grafika

      Popis byl vytvořen automaticky**Počet nezávislých parametrů je , protože
* **Test nezávislosti test** 
  +  neznámými parametry
  + Odhady metodou min.

Obsah obrázku Písmo, číslo, řada/pruh, snímek obrazovky

Popis byl vytvořen automaticky

* + Statistika má asymptoticky rozd. s  stupni volnosti
  + Provedení testu:

Obsah obrázku text, řada/pruh, Písmo, snímek obrazovky

Popis byl vytvořen automaticky

* + - Počet **stupňů volnosti** (degrees of freedom dF) =
  + Alternativní výpočet statistiky :

Obsah obrázku Písmo, text, řada/pruh, číslo

Popis byl vytvořen automaticky

**7. Základy teorie informace a kódování, entropie.**

NI-VSM

**Entropie**

* **Entropie** = „míra neuspořádanosti“
* Uvažujeme **diskrétní náhodné veličiny**
  + Množina hodnot
  + Pravděpodobnostní funkce v  …
  + Pravděpodobnostní rozdělení …
  + Argument pravděpodobnostní funkce – o kterou veličinu se jedná
* **Entropie diskrétní náhodné veličiny**:
  + Log o základu 2,
  + Entropie závisí pouze na rozdělení veličiny
  + Je invariantní vůči transformacím:
* **Jednotky entropie** – báze logaritmu – , označuje jednotky entropie
  + … bit
  + … digit
  + … nat
  + Přechod mezi bázemi:
* Obsah obrázku text, kruh, Písmo, diagram

  Popis byl vytvořen automatickyEntropie **jako očekávaná míra neurčitosti**
  + lze chápat jako **střední hodnota**:
    - = **vlastní informace** = míra neurčitosti

entropie je očekávaná míra neurčitosti

* + Míra neurčitosti je vždy nezáporná a pro jisté jevy 0
  + Méně pravděpodobný jev vyšší míra neurčitosti
  + se při pozorování nezávislých jevů sčítá
* **Vlastnosti entropie**
  + **Nezápornost** entropie:
  + Entropie je konkávní funkcí rozdělení
  + V deterministických případech je entropie 0
  + Maximální rovnoměrné rozdělení – nejvyšší neurčitost
* **Sdružená entropie**  diskrétních náhodných veličin se sdruženým rozdělením :
  + Sdružená entropie diskrétního náhodného vektoru se sdruženým rozdělením :
  + Alternativně
* **Podmíněná entropie**  diskrétních náhodných veličin se sdruženým rozdělením :
  + Alternativně
    - Z

* **Řetězové pravidlo**:

… určuje, která část informace je ve veličině navíc oproti tomu, co je v

* **Relativní entropie** = Kullback-Leiblerova vzdálenost :
  + Pokud :
  + Je to „vzdálenost“ – nezáporná a 0, jen pokud
    - Ale ne opravdová, neplatí ani trojúhelníková nerovnost
    - Alternativně:
* **Vzájemná informace** :

= relativní entropie skutečného sdruženého rozdělení a rozdělení nezáv. veličin se stejnými marginálami:

* + Symetrie:
  + Z nezápornosti relativní entropie:
* **Vztah vzájemné informace a entropie**
  + Odvození přes věty o logaritmech
  + ; – vlastní informace
* **Informační nerovnost**: možná rozdělení :
  + Rovnost pouze pokud
  + Důsledky:
    - **Nezápornost vzájemné informace** – pro dvě d.n.v. :
      * Pokud **rovnost, pak jsou nezávislé**
      * je číselná charakteristika sdruženého rozdělení, která je schopná poznat nezávislost
    - **Maximalizace entropie** – pro d.n.v. s hodnotami z :
      * … počet prvků množiny – rovnost, pokud rovnoměrné rozdělení
      * Entropie je maximalizována **rovnoměrným rozdělením**
    - **Podmiňování redukuje entropii**:
      * **Rovnost, pokud jsou a nezávislé**
      * „informace neublíží“ – znalost n.v. může v průměru pouze redukovat neurč. v
      * Pouze v průměru, samotné může být pro nějaké větší než , ale:

**Teorie kódování**

* Jak zapsat zdrojovou zprávu, tak, aby následný přenos byl co nejefektivnější
* **D-ární abeceda** – abeceda obsahující přenositelných symbolů
* **Zpráva** je posloupnost znaků z
* Chceme co nejkratší zakódovanou zprávu
* Zobrazení z množiny do množiny konečných řetězců symbolů -ární abecedy nazýváme **kód** diskrétní náhodné veličiny
  + Obraz = **kódové slovo** příslušného prvku a jeho délku značíme
    - …řetězec symbolů z  délky
* **Střední délka** kódu náhodné veličiny s rozdělením :
  + …**délka kódového slova** příslušejícího k prvku :

* **Typy kódů**
  + **Nesingulární kód** d.n.v. – pokud je prosté zobrazení
    - Dostačující pro schopnost rekonstruovat z kódových slov jednotlivé hodnoty
    - Není dostačující pro dekódování posloupnosti hodnot (celých zpráv)
  + **Jednoznačně dekódovatelný kód** – pokud je nesingulární
    - = rozšíření kódu – zobrazení do :
      * Zápis jednotlivých kódových slov po sobě
    - Jsme schopni jednoznačně dekódovat libovolnou přijatou zprávu
  + Kód je **instantní** (prexifový), pokud žádné kódové slovo není prefixem jiného kódového slova
  + Obsah obrázku kruh, text, Písmo, diagram

    Popis byl vytvořen automatickyHierarchie kódů:
* **Kraftova nerovnost** – pro libovolný instantní kód nad -ární abecedou musí délky kódových slov splnit nerovnost:

Navíc, ke každé -tici délek, které splní tuto nerovnost, existuje instantní kód s kódovými slovy těchto délek

* + Pro jednoznačně dekódovatelné kódy analogicky (McMillanova věta)
    - Ke každému jednoznačně dekódovatelnému kódu lze sestrojit instantní kód, který má stejně dlouhá kódová slova
* **Optimální kódy**
  + **Střední délka** instantního -árního kódu d.n.v. je:
    - Rovnost, právě když
  + **Optimální kód** = kód o nejmenší střední délce
  + Uvažujme optimální instantní kód
    - **Optimálním kódem se od dolní meze dané entropií můžeme vzdálit maximálně o 1**
* **Huffmanovo kódování**
  + Algoritmus na sestrojení binárního Huffmanova kódu:

1. Spojíme 2 nejmíň pravděpodobné hodnoty nové rozdělení s o 1 menším počtem hodnot
2. Opakujeme, dokud nezůstane 1 hodnota prázdný řetěz jako kódové slovo
3. Zpětným chodem zkonstruujeme kódová slova všech původních hodnot
4. Pro hodnotu , která vznikla spojením a vytvoříme kódové slovo méně pravděpodobné hodnoty připojením 1 za kódové slovo a analogicky kódové slovo více pravděpodobné hodnoty připojením 0 za

* Tzn. Pokud a , tak a
  + Huffmanův kód je **optimální** – je-li Huffmanův kód a libovolný jednoznačně dekódovatelný kód, potom
  + Algoritmus sestrojení je hladový algoritmus, který **lokálně agreguje** 2 nejméně pravděpodobné hodnoty

**8. Markovské řetězce s diskrétním časem. Jejich limitní vlastnosti.**

NI-VSM

* **Náhodný proces** – buďte pravděpodobnostní prostor a indexová množina. Náhodná proces je systém náhodných veličin
* **Množina**  lze chápat jako **čas** časová souslednost dána uspořádáním
  + **Diskrétní čas** – je-li nejvýše spočetná
  + **Spojitý čas** – je-li nespočetná
* **Množina stavů**  = minimální podmnožina , pro kterou platí, že
  + = společná množina hodnot (diskrétní/kontinuum)
* **Trajektorie náhodného procesu**
  + **Náhodný proces** = zobrazení z  do prostoru funkcí
  + …hodnota funkce proměnných a ( je elementární jev)
  + **Trajektorie/realizace** náhodného procesu = funkce :
* Spočetná množina
* Diskrétní čas:
* Rozdělení v čase char. pravděpodobnostní funkcí:
* **Matice pravděpodobnostního přechodu** za čas mezi a :

**Markovský řetězec**

* Náhodný proces s nejvýše spočetnou množinou stavů nazýváme **markovský řetězec s diskrétním časem**, pokud pro každý stav splňuje **markovskou podmínku**:



* Následující podmínky jsou **ekvivalentní markovské podmínce**:





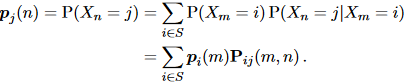
* **Ekvivalentní definice markovského řetězce** – náhodný proces s nejvýše spočetnou množinou stavů je markovský řetězec právě tehdy, když pro každé :



* **Chapman-Kolmogorova rovnice** – pro matice přechodu markovského řetězce platí :

**Homogenní markovský řetězec**

* Markovský řetězec je **homogenní**, pokud :
* Pro homogenní markovský řetězec platí :
* Pro homogenní m. ř. definujeme (jednokrokovou) **matici přechodu**:
  + Značení
  + **Pravděpodobnost přechodu** ze stavu  do během kroků:
  + Ch-K. rovnice pro homo. m. ř.:
    - Jiný zápis pro
* **Rozdělení náhodné veličiny**
  + Pro platí, že:



* + Maticový zápis:
  + Tím pádem pro h. m. ř. platí:
* **Stochastické matice** – matice přechodu je stochastická:
  + má **nezáporné prvky**:
  + **Součet řádků je roven 1**:
  + Součin stochastických matic je opět stochastická matice
  + K libovolné čtvercové stochastické matici existuje homogenní markovský řetězec s diskrétním časem takový, že je jeho maticí přechodu
* Příklad diagramu přechodu a matice přechodu:

Obsah obrázku text, Písmo, rukopis, bílé

Popis byl vytvořen automaticky

**Stacionární rozdělení**

* **Počáteční rozdělení** – může být libovolný vektor splňující:



* + Maticí přechodu definujeme třídu náhodných procesů definovaných na prostorech , pro které platí:



* **Stacionární rozdělení** – buď homogenní markovský řetězec s maticí přechodu . Pokud existuje vektor takový, že:

pro který platí , nazýváme jej stacionárním rozdělením řetězce

* + Existuje-li stacionární rozdělení, pak



* + Stacionární rozdělení má požadovanou vlastnost

**Klasifikace stavů markovského řetězce**

* Stav nazveme **trvalý (rekurentní)**, pokud:
  + **Trvalost** = každá trajektorie začínající v  se někdy vrátí do skoro jistě
* Stav nazveme **přechodný (transientní)**, pokud není trvalý, tj:
  + **Přechodnost** = existuje hodně trajektorií, které se do už nikdy nevrátí
* Čas první návštěvy stavu :

je-li množina neprázdná, a , je-li množina prázdná

* + …pravděpodobnost, že **řetězec někdy navštíví** , startoval-li v
  + …pst, že **1. návštěva** při startu z nastane v -tém kroku
  + Z toho vyplývá:
    - Stav je **trvalý**
    - Stav je **přechodný**
* **Střední doba návratu** do :

Obsah obrázku text, Písmo, snímek obrazovky, bílé

Popis byl vytvořen automaticky

* + **Trvalý stav**  = nenulový, pokud je střední doba návratu konečná:
    - Jinak je stav nulový
* **Perioda stavu** :

= **největší společný dělitel** časů, kdy se řetězec vrátí do stavu

* + **Periodický stav** =
  + **Aperiodický stav** =
  + , je-li stav periodický s , pak :



* + Pokud existují takové, že a = stavy jsou **vzájemně dosažitelné**, pak:
* **Klasifikace stavů** pomocí matice přechodu – stav m.ř. je:
  + **Přechodný** , potom
  + **Trvalý nulový** a , potom
  + **Trvalý nenulový aperiodický** , potom
  + **Trvalý nenulový periodický** má periodu a

**Rozklad množiny stavů**

* **Dosažitelný stav** ze stavu (, pokud se lze dostat z  do v konečném čase =
* Stavy a jsou **vzájemně dosažitelné** (), pokud a
  + Relace je ekvivalence
  + Pokud , jsou oba stejného typu
* **Uzavřená množina**
  + Uspořádejme stavy tak, aby byly na konci ():

Obsah obrázku Písmo, text, řada/pruh

Popis byl vytvořen automaticky

* + Z uzavřené množiny řetězec neuteče: a
  + Je-li uzavřená množina tvořena jediným stavem , pak tento stav = **pohlcující** (**absorbční**)
* **Rozložitelnost** – množina stavů **nerozložitelná** (**ireducibilní**), pokud platí
  + Markovský řetězec je nerozložitelný, pokud je nerozložitelná
* **Jednoznačný rozklad** – množina stavů lze jednoznačně rozložit do tvaru
  + … množina všech **přechodných stavů**
  + vzájemně disjunktní nerozložitelné uzavřené **množiny trvalých stavů**
  + **Matice přechodu** po uspořádání má tvar:

Obsah obrázku Písmo, diagram, číslo, snímek obrazovky

Popis byl vytvořen automaticky

* + - … čtvercová matice přechodů mezi **přechodnými stavy** v
    - … čtvercová matice přechodů mezi **trvalými stavy** v
    - … matice přechodů z množiny přechodných stavů do množiny trvalých stavů
* Je-li stav trvalý a , pak je také trvalý a
* V řetězci s konečně mnoha stavy
  + Nemohou být všechny stavy přechodné (takže **alespoň 1 trvalý**)
  + Neexistují stavy trvalé nulové (takže trvalé stavy **vždy nenulové**)
  + Tím pádem alespoň 1 **uzavřená nerozložitelná množina** trvalých nenulových stavů (bude jich konečně mnoho), zbylé stavy přechodné
* Mějme konečnou množinu stavů a . množina stavů dosažitelných z . Pak
  + Pokud , stav je **trvalý nenulový**
  + Pokud , stav je **přechodný**

**Existence stacionárního rozdělení**

* Pro rozložitelný markovský řetězec platí:
  + Jsou-li všechny stavy **přechodné nebo trvalé nulové**, stacionární rozdělení **neexistuje**
  + Jsou-li všechny stavy **trvalé nenulové**, stacionární rozdělení existuje a je **jediné**. Jsou-li navíc všechny stavy aperiodické, platí , tedy

pro libovolné

* Je-li množina stavů konečná, pak stacionární rozdělení existuje
* **Počet stacionárních rozdělení**
  + Pro každou množinu trvalých nenulových stavů, , existuje **stacionární rozdělení** splňující:



* + Pak platí, že vektor řeší rovnici



* + Z toho plyne, že máme celkem tolik lineárně nezávislých **stacionárních rozdělení** , kolik je (nenulových) množin
  + Pak libovolná konvexní kombinace



je stacionárním rozdělením procesu s maticí přechodu

**Pravděpodobnosti pohlcení**

* Množinu stavů lze jednoznačně rozložit do tvaru , přechodné, nerozložitelné trvalé
* konečná alespoň 1 neprázdná , všechny stavy jsou nenulové
* přechodný

Obsah obrázku Písmo, text, bílé, řada/pruh

Popis byl vytvořen automaticky

* Je-li konečná, bude řetězec v konečné době pohlcen jednou z množin
* **Čas absorbce** = náhodná veličina :



* + Je-li množina neprázdná, jinak
  + Je-li množina stavů konečná, pak
* Označme  **pravděpodobnost**, **že řetězec startující v opustí přechodem do stavu** :
  + Pravděpodobnost pohlcení v množině :
  + Obsah obrázku Písmo, číslo, kruh, snímek obrazovky

    Popis byl vytvořen automatickyPokud nás nezajímá, **kudy se řetězec** do uzavřené množiny  **dostal,** je výhodné sloučit stavy do jednoho „superstavu“:
    - Matice přechodu :
* **Matice pravděpodobností pohlcení**  je řešením rovnice
  + Řešení rovnice:
  + Je-li matice regulární, existuje jediné řešení:
  + Buď čtvercová matice taková, že při . Pak matice je regulární a platí
  + Pro matici pravděpodobnosti pohlcení platí
* **Počet návštěv stavu** = náhodná veličina :



* + Pro trvalý tedy platí skoro jistě:
  + Pro přechodný naopak
  + 
* = **střední počet návštěv stavu** , jestliže řetězec startuje v
  + Pro matici :
  + … **fundamentální matice** řetězce
* **Střední doba do pohlcení** při startu v :



* **Limita matice** 
  + Struktura matice odpovídající trvalým stavům

Obsah obrázku text, Písmo, číslo, diagram

Popis byl vytvořen automaticky

* + Předpoklad, že stavy trvalé jsou aperiodické. Pak



* + má v řádcích stacionární rozdělení

Obsah obrázku Písmo, diagram, text, řada/pruh

Popis byl vytvořen automaticky

* **Limita matice**  – buď konečná množina stavů, přechodné, trvalé aperiodické. Pak

Obsah obrázku text, Písmo, číslo, typografie

Popis byl vytvořen automaticky

**9. Markovské řetězce se spojitým časem. Souvislost s Markovskými řetezci s diskrétním časem a s Poissonovým procesem.**

NI-VSM

* **Čítací proces** = náhodný proces , jehož trajektorie jsou nezáporné, celočíselné a neklesající:

  + Pro udává přírůstek počet událostí, které nastaly během časového intervalu
  + **Binomický proces** = čítací proces takový, že časy mezi událostmi jsou nezávislé a geometricky rozdělené
  + Spojitý ekvivalent geometrického rozdělení je exponenciální
  + **Poissonův proces** = čítací proces s nezávislými exponenciálně rozdělenými časy mezi událostmi

**Poissonův proces**

* Buďte iid náhodné veličiny s rozdělením . Definujeme :

Obsah obrázku text, Písmo, bílé, typografie

Popis byl vytvořen automaticky

pak náhodný proces , kde:



nazveme **Poissonovým procesem**

* Modeluje **příchody událostí**, které jsou na sobě **nezávislé**. Mezi přicházejícími událostmi není žádná interakce
* Řekneme, že proces je **Poissonův proces**, pokud:

1. skoro jistě
2. má nezávislé přírůstky: a pro všechny



* **Binomický x Poissonův proces**
  + Liší se pouze rozdělením času mezi událostmi

Obsah obrázku text, Písmo, snímek obrazovky, řada/pruh

Popis byl vytvořen automaticky

* + Geometrické a exponenciální rozdělení mají společnou vlastnost – **bezpaměťovost**
  + Poisson lze chápat jako spojitou variantu binomického
  + Bezpaměťovost klíčová pro MŘ se spojitým časem
* **Exponenciální rozdělení** :

Obsah obrázku text, Písmo, rukopis, snímek obrazovky

Popis byl vytvořen automaticky Obsah obrázku text, Písmo, snímek obrazovky, řada/pruh

Popis byl vytvořen automaticky

* **Bezpaměťovost exponenciálního rozdělení** – buď exponenciálně rozdělená náhodná veličina. Pak :



* + Pokud jsme na bus čekali minut, pak pst, že budeme čekat dalších minut je stejná, jako kdybychom vůbec nečekali
* **Silná bezpaměťovost exponenciálního rozdělení** – buď a buď spojitá nezáporná náhodná veličina nezávislá na . Pak :



* + Pokud jsme na bus 143 čekali do příjezdu 180, pak pst, že budeme čekat dalších minut je stejná, jako kdybychom vůbec nečekali
* Buďte . Pak , tj.



* Náhodná veličina má Poissonovo rozdělení s parametrem
* **Bezpaměťovost** – buď pevné. Pak náhodný proces je Poissonův proces s intenzitou . Navíc je nezávislé na pro
  + Když přijdu k běžícímu Poiss. procesu v čase , je to stejné, jako by se proces restartoval

má **nezávislé přírůstky**

**Markovský řetězec se spojitým časem**

* **Markovský řetězec se spojitým časem** = náhodný proces s nejvýše spočetnou množinou stavů , který splňuje markovskou podmínku:



* **Rozdělení v čase** . Pro :



* **Matice pravděpodobností přechodu** za čas mezi a :



* Náhodný proces s nejvýše spočetnou množinou stavů je markovský právě tehdy, když

****



* **Chapman-Kolmogorova věta** – pro matice přechodu markovského řetězce platí :
* **Homogenní markovský řetězec** - platí:
  + **Chapman-Kolmogorova věta** pro homogenní MŘ:
  + **Rozdělení** v :
* **Matice skokových intenzit**
  + Pokud existují konečné limity



nazveme maticí (skokových) intenzit

* + - Předpokládáme, že limity existují a jsou konečné. Pak:



* + Vlastnosti matice skokových intenzit:
    - Neboť , platí pro :
    - Neboť , platí:

Obsah obrázku Písmo, řada/pruh, text, typografie

Popis byl vytvořen automaticky

* + - **Skok z do** :
    - **Skok z pryč**:
* **Simulace procesu** pomocí skokových intenzit
  + Buď čas do výskoku ze stavu , tj. pro . Pak:

1. Čas do výskoku z  je exponenciální s

=

1. Pravděpodobnost, že řetězec skočí z  do je dána poloměrem intenzit , tj.



* **Princip simulace**:

1. Začínám v
2. Generuji náhodný čas a „posunu“ hodiny o
3. Změním stav z  na s pravděpodobností
4. Opakuji od 2.

**Konstrukce řetězců se spojitým časem**

* **Homogenní MŘ se spojitým časem** (shrnutí)
  + **Markovská podmínka**



* + **Homogenita**



* + **Chapman-Kolmogorova rovnice**



* + **Matice skokových intenzit**



* + **Vlastnosti matice**



* **Diskrétní čas** časovaný Poissonovým procesem
  + Buď Poissonovský proces s intenzitou . Buď homogenní MŘ s diskrétním časem mající matici přechodu . Pak proces definovaný jako



je **homogenní markovský proces se spojitým časem**

* + **Dynamika procesu**
    - Bez ohledu na minulost má čas do další události rozdělení
    - Po uplynutí tohoto času přeskočí řetězec z  do s pravděpodobností
    - Čas do další události je opět exponenciální a nezávislý na minulosti
  + **Matice přechodu**
    - Pravděpodobnost, že za čas nastane právě události je
    - Za podmínky, že nastalo událostí, je pst přeskoku z  do dána -krokovou pravděpodobností přechodu řetězce :



* + - Navíc platí:



* + - **Matice přechodu** řetězce mají tvar:



* + - **Prvky matice přechodu** pro a :
    - Prvky pto :
    - Prvky pro :
* **Konstrukce diskrétního řetězce** pomocí :
  + Aby byla stochastická, musí být
    - Vždy splněno, je-li konečná
* **Simulace pomocí diskrétního řetězce** – mám a chci simulovat trajektorii

1. Matice přechodu:
2. Vygeneruji trajektorii Poissonova procesu
3. Vygeneruji trajektorii pomocí matice
4. Trajektorie :

**Kolmogorovy rovnice**

* Výpočet **matic přechodu** pomocí **matice intenzit**
  + MŘ se spojitým časem definovaná pomocí matice intenzit přeskoku
  + Chceme znát **vývoj rozdělení v čase** :
    - K tomu potřebujeme matice přechodu
  + je řešením **soustavy diferenciálních rovnic**:



* + - Maticově:



* **Kolmogorovy rovnice** – buď markovský řetězec s maticí intenzit . Pak pro matice přechodu platí:
  + **Kolmogorova dopředná rovnice**:
  + **Kolmogorova zpětná rovnice**:
* Rozdělení je řešením soustavy diferenciálních rovnic
  + Po složkách:

Obsah obrázku text, rukopis, Písmo, diagram

Popis byl vytvořen automaticky

* **Řešení Kolmogorových rovnic** – pro matice přechodu platí , kde



**Stacionární rozdělení**

* Buď MŘ s pravděpodobnostmi přechodu . Pak vektor nazvu **stacionárním rozdělením**, pokud :
* Vektor je **stacionárním rozdělením**, právě když
* **Limitní vlastnosti**
  + MŘ je **nerozložitelný (ireducibilní)**, pokud se z každého stavu mohu dostat do libovolného stavu pomocí konečně mnoha přeskoků

= pokud existuje a stavy takové, že pro

* + Buď **nerozložitelný MŘ se spojitým časem**:

1. Existuje-li stacionární rozdělení , pak je jednoznačné a :



1. Pokud stacionární rozdělení neexistuje, pak :



* Je-li množina stavů **konečná**, pak stacionární rozdělení markovského řetězce se spojitým časem existuje
* **Detailní rovnováha**
  + Stacionární rozdělení splňuje



* + Lze přepsat na:



* + Pokud rozdělení splňuje detailní rovnováhu:



pak je stacionárním rozdělením

**10. Systémy hromadné obsluhy a jejich limitní vlastnosti. Souvislost s Markovskými řetězci se spojitým časem.**

NI-VSM

**Exponenciální závody a Markovské řetězce**

* **Model hromadné obsluhy**

Obsah obrázku řada/pruh, snímek obrazovky, diagram, Písmo

Popis byl vytvořen automaticky

* + Princip:
    - Požadavky na server **přichází** náhodně s **intenzitou**
    - Je-li server zaneprázdněn vyřizováním požadavku, zařadí se nový do **fronty**
    - Server **vyřizuje** požadavky s **intenzitou**
* **Rozdělení minima** – buďte a nezávislé. Pak



buď nezávislé veličiny,. Pak



* + Z toho plyne, že:



* + Platí, že . Potom



* + Pro libovolné nezávislé náhodné veličiny platí



* **Vítěz závodů** – buďte a nezávislé. Pak



buď nezávislé veličiny,. Pak



* **Konstrukce matice intenzit**

1. Přeskok o :
2. Přeskok o :

1. Setrvání:

* Proces je MŘ se spojitým časem mezi jednotlivými stavy probíhají **exponenciální závody**

1. Začínám v
2. Generuji náhodný čas , kde a “posunu hodiny” o
3. Změním stav z  na s pravděpodobností
4. Opakuji od 2.
   * Pak se jedná o markovský řetězec s maticí intenzit takovou, že:

* Řetězec se spojitým časem lze popsat **diagramem** – váhy hran určeny intenzitou přeskoku mezi jednotlivými stavy

**Systémy hromadné obsluhy**

* **Model hromadné obsluhy**
  + [zákazníků za časovou jednotku] … **intenzita příchodů**
  + … náhodný čas mezi příchodem -ního a -tého zákazníka,

* + [zákazníků za časovou jednotku] … **intenzita obsluhy** 1 serveru
  + … čas obsluhy -tého zákazníka

* + Veličiny jsou nezávislé
  + Server obsahuje nezávislých obslužných míst
* **Proces hromadné obsluhy** = proces, který zaznamenává počet zákazníků v systému hromadné obsluhy (= **serveru a frontě**) v čase
  + Konečněrozměrná rozdělení procesu jsou jednoznačně určena rozděleními a
  + Intenzita příchodů je
  + Intenzita obsluhy zákazníků je nejvýše
    - počet zákazníků v systému poroste nad všechny meze
    - systém se ustálí na stabilním rovnovážném rozdělení
* **Kendallova notace** 
  + … rozdělení **časů příchodu**
  + … rozdělení **časů obsluhy**
  + … počet **obslužných míst**
  + … kapacita systému (neuvedeno = )
  + … velikost populace (neuvedeno = )
  + … typ obsluhy (neuvedeno = FIFO)
* Rozdělení jsou značena:
  + – **exponenciální** rozdělení (markovské)
  + – **degenerované rozdělení** soustředěné v hodnotě
  + – **obecné rozdělení**, neznámé nebo známé „neexponenciální“
* **Systém** 
  + Proces **zrodu a zániku** s parametry:
  + **Matice intenzit**:

Obsah obrázku Písmo, rukopis, typografie, text

Popis byl vytvořen automaticky

* + Pokud , pak existuje **stacionární rozdělení** ve tvaru:

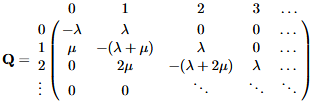




* + - Pokud , stacionární rozdělení neexistuje a
  + Stacionární vlastnosti
    - **Střední počet** zákazníků v **systému**:
    - **Střední počet** zákazníků na **serveru**:
    - **Střední počet** zákazníků ve **frontě**:
  + **Doba čekání** zákazníka ve **frontě**:
* **Systémy** 
  + Nekonečno obslužných míst – každý zákazník **okamžitě obsluhován**
  + Proces zrodu a zániku s parametry:



* + **Matice intenzit**:



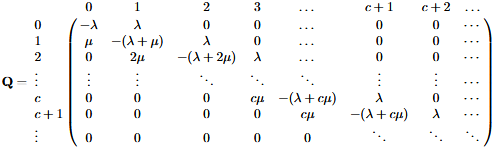
* + **Stacionární rozdělení** vždy existuje a je Poissonovo rozdělení s parametrem , takže pro :



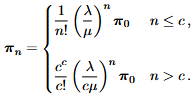
* **Systém :**
  + nezávislých obslužných míst existuje **fronta**
  + Proces zrodu a zániku s intenzitami:



* + **Matice intenzit**:



* + **Stacionární rozdělení** existuje pro :



* **Littleho věta** – buď striktně stacionární proces hromadné obsluhy. Buďte dále:

1. … střední **počet zákazníků** v systému
2. … střední **doba** strávená zákazníkem v systému
3. … intenzita procesu příchodů

Jsou-li všechny střední hodnoty konečné, pak

* **Systém** 
  + **Obecné rozdělení** příchodů i obsluhy (při zachování stacionarity)
  + … intenzita příchodů
  + … intenzita obsluhy
    - **Střední doba obsluhy** =
  + … **doba strávená -tým zákazníkem** v systému
    - … doba čekání ve frontě
    - … doba obsluhy
  + **Littleho věta** pro:
    - Celý **systém**:
    - Samotnou **frontu**:
  + Z toho vyplývá:

**Nehomogenní Poissonův proces**

* Buď funkce integrabilní na libovolném konečném intervalu, pak proces nazvu nehomogenním Poissonovým procese s intenzitou , pokud

1. skoro jistě
2. má nezávislé přírůstky
3. Pro :



* Modelujeme situaci, kdy se **intenzita příchodu** zákazníka **mění s časem**:
* **Systém** 
  + … **Poissonovské příchody** s intenzitou
  + … **obecné rozdělení** časů obsluhy se střední hodnotou a distribuční funkci
  + … neomezený počet serverů (**žádná fronta**)
  + Systém v  prázdný požadavek, který přišel v čase je v čase **stále v systému s pravděpodobností**:
    - Z pohledu tedy událost, která přišla v čase **přijmu s pravděpodobností**
    - Poissonovo rozdělení s parametrem
  + Počet zákazníků v systému má z dlouhodobého pohledu () Poissonovo rozdělení s parametrem
  + **Poissonovo rozdělení**:



* + - Pravděpodobnost, že v systému bude požadavků z dlouhodobého hlediska

**11. Význam tříd NP a NPH pro praktické výpočty.**

NI-KOP

**Kombinatorický problém**

* **Kombinatorický problém** – charakterizují ho:
  + Vstupní proměnné – seznam prvků
  + Výstupní proměnné – pořadí prvků (popis výstupu)
    - Hodnota, kterou chceme optimalizovat – například u TSP délka cesty, kterou minimalizujeme
  + Konfigurační proměnné – pořadí prvků (popis konfigurace)
    - Proměnná, kterou můžeme měnit v rámci řešení problému – např. u TSP pořadí měst
  + Omezení – každý prvek právě jednou
  + Optimalizační kritérium – nejmenší
* **Proměnné** – konečný počet, konečné domény
* **Instance a řešení problému**
  + **Instance**  – ohodnocení vstupních proměnných
  + **Konfigurace** – ohodnocení konfiguračních proměnných (během řešení instance)
  + **Řešení instance** – ohodnocení konfiguračních proměnných splňujících omezení
    - Omezení říká, jestli je řešením instance
  + **Optimální řešení** – řešení s nejlepší hodnotou optimal. Kritéria
  + Obsah obrázku kruh, diagram

    Popis byl vytvořen automaticky**Suboptimální řešení** – řešení s vyhovující hodnotou opt. Kritéria
* Typy problémů
  + **Rozhodovací problém**: Existuje řešení? Nebo platí omezení pro všechny konfigurace?
  + **Konstruktivní problém**: Sestrojte řešení (konfiguraci) takovou, aby platila omezení
  + **Enumerační problém**: Sestrojte všechna řešení, pro které platí omezení
  + Všechny typy problémů lze převést na optimalizační verze
  + Měření složitosti pomocí asymptotické složitosti
* **Turingův stroj** – původně pro rozhodnutelnost problému
  + Neomezená páska, čtecí hlavice, řídící jednotka (stav)
  + Obsah obrázku text, kruh, snímek obrazovky, Písmo

    Popis byl vytvořen automatickyZastavení = přechod do koncového stavu
  + – konečná neprázdná množina stavů
  + – konečná vstupní abeceda
  + – konečná neprázdná pracovná abeceda
  + – přechodová funkce
  + – počáteční stav
  + − prázdný symbol
  + − množina koncových stavů

**Třídy složitosti**

* **Třída P** – rozhodovací problém patří do třídy P, jestliže pro něj existuje program pro deterministický Turingův stroj, který jej řeší v čase , kde je velikost instance a konečné číslo
* **PSPACE** – existuje program pro deterministický Turingův stroj, který jej řeší v **paměti** , kde je velikost instance a konečné číslo
* **EXPTIME** – existuje program pro deterministický Turingův stroj, který jej řeší v **čase** , kde je polynom ve velikosti instance
* **NP** – problém patří do NP, pokud existuje program pro NTS, který každou ANO-instanci řeší v polynomiálním čase , kde je velikost instance a konečné číslo
  + NP
    - ANO – ověření probíhá jako existence odpovědi – certifikát v P
    - NE – musíme zkontrolovat všechny možnosti
  + **co-NP** – doplněk k NP problémům
    - NE – ověření jako existence 1 odpovědí (ne, není pravda, že nemůžeme najít řešení – certifikát v P)
    - ANO – ano, nemůžeme najít řešení – potřeba kontroly všech
* **NPH** – problém je NPH, pokud se efektivní (polynomiální) řešení všech problémů v NP dá zredukovat na řešení
  + Nejtěžší problém je takový, na který se dají převést všechny ostatní problémy
  + Je nejméně tak těžký, jako problémy, jež na něj byly převedeny
  + **NPC** – pokud je zároveň NPH a NP
    - **Cookova věta** – všechny NP problémy lze převést na SAT
* **NPO –** optimalizační problém patří do třídy NPO, jestliže splňuje podmínky:
  + Velikost výstupu instance omezena polynomem ve velikosti instance
  + Problém, zda daná konfigurace je řešením, patří do P
  + Existuje program pro TS, který vypočítá hodnotu optimalizačního kritéria pro každé řešení každé instance v polynomiálním čase
* Problémy různých složitostí:
  + P – součet dvou čísel
  + NP – faktorizace čísel
  + NPC – SAT
  + NPH – halting problem
* **Redukce problémů**
  + **Karpova redukce** – problém je karp-redukovatelný na , pokud existuje polynomiální algoritmus na DTS, který provede každou instanci problému na instanci problému tak, že jejich výstup je shodný
    - Platí tranzitivita redukovatelnosti problémů
  + **Turingova redukce** – problém je turing-redukovatelný na , pokud existuje algoritmus na DTS, který řeší každou instanci problému tak, že používá program pro problém jako podprogram
    - Karpova redukce je speciální případ

**Význam pro praktické výpočty**

* NP a NPH problémy nelze efektivně řešit – řešení je exponenciální na DTS (normálním počítači), které pro větší instance může trvat neakceptovatelně dlouho - – využití v kryptografii, hashování
* Neznámé problémy můžeme **redukcí převádět na známé problémy** a řešit je dle zvyku
  + Např. převod na SAT problém splnitelnosti
* V některých případech nepotřebujeme přesné řešení, stačí **přibližné/suboptimální**
* **Zrychlení řešení NPO úloh**
  + **Metoda redukce stavového prostoru** – např. branch and bound – nepoužitelné výsledky ignorovány
  + **Pseudopolynomiální algoritmy** – počet kroků algoritmu závisí na velikosti instance, ale závisí dále na parametru, který s velikostí nemá nic společného -
  + **Aproximace** – FPTAS, PTAS, APX – jednodušší heuristika pro přibližné řešení – spokojenost s určitou hladinou chybovosti algoritmu
  + **Randomizace** – algoritmy založené na náhodné volbě
* Obsah obrázku text, snímek obrazovky, Grafika, Písmo

  Popis byl vytvořen automaticky**Aproximativní algoritmy**
  + … hodnota optimalizačního kritéria řešení
  + … aproximované řešení instance
  + … optimální řešení instance
  + **Relativní chyba**:

Obsah obrázku text, Písmo, bílé, řada/pruh

Popis byl vytvořen automaticky

* + **Relativní kvalita**:



* + Chyba a kvalita jsou spolu spojené:
  + **APR** – algoritmus APR pro problém je -aproximativní (-aproximativní), jestliže každou instanci vyřešeí v polynomiálním čase s relativní kvalitou ()
    - () je aproximační prah problému
  + **PTAS** – algoritmus APR, který pro každé vyřeší každou instanci problému s relativní chybou nejvýše ε v čase polynomiálním vnazýváme polynomiální aproximační schéma problému
  + **FPTAS** – polynomiální aproximační schéma APR, jehož čas výpočtu závisí polynomiálně na **1/ε** nazýváme plně polynomiální aproximační schéma
* **Nejtypičtější NPH úlohy**
  + SAT, 3SAT – problém splnitelnosti booleovské formule
  + Problém batohu, obchodní cestující, hamiltonovská kružnice
  + Existence zadání
    - Offline – mám všechna data
    - Online – postupně dostávám údaje a upravuji řešení
  + Některé úlohy lze řešit ne lokálními, ale globálními metodami (dekompozice, dynam. prog., …)

**12. Experimentální vyhodnocení algoritmů, zejména randomizovaných.**

NI-KOP

**Experimentální vyhodnocení**

* **Analytické odpovědi o algoritmech**
  + Nejhorší případ, asymptotické meze
    - Významná oblast může být mimo zájem
    - Konstanty jsou nepraktické
    - Nejhorší případ je vzácný a nezajímavý, analýza je příliš složitá
  + Průměrný případ
    - Analýza je proveditelná pro jednoduché případy
    - Analýza závisí na statistických vlastnostech vstupu, které nemusí být známy
* Co nás zajímá
  + **Složitost** (teoretická x z hlediska nasazení)
  + **Kvalita řešení**
  + **Porozumění** – proč to blbne na určitých instancích
    - Závislost něčeho na něčem (výpočetní čas na velikosti instance, kvalita řešení na param.)
* **Experiment**:

co chci zjistit plán experimentu provedení experimentu sběr dat interpretace výsledku odpověď´

* **Metriky** – metrika vstupu závislost metrika výstupu interpretace
  + Abstrahujeme vlastnosti vstupu a výstupu do kvantitativních veličin = metrik
  + Zkoumáme závislost mezi metrikami vstupu a výstupu
  + Interpretace – zobecnění podle jednotlivých chodů
* **Metriky výstupu – složitost**
  + Metriky vypovídající o algoritmu (např. počet zkoumaných konfigurací)
  + Metriky vypovídající o implementaci (např. čas CPU)
  + Chceme metriku, která nezávisí na detailech implementace (v praxi náročné vyhodnocení optimalizačního kritéria)
* **Metriky vstupu**
  + Ostatní metriky vstupu – známe a udržíme konstantní při generování, nebo vůbec o nich nevíme
    - Tlak, teplota okolí, … - snažíme se udržet konstantní
* **Příklad metrik a závislostí** (3-SAT)
  + Závislost počtu kroků procedury DP (backtracking, výstupní metrika) na poměru počtu klauzulí k počtu proměnných (vstupní metrika)
  + Fázový přechod – pst splnitelnosti v závislosti na poměru počtu klauzulí k počtu proměnných
* **Neznámé metriky**
  + **Generování instancí** – pro každou hodnotu nastavované metriky vygenerujeme náhodné instance tak, aby každá instance se zadanou metrikou byla stejně pravděpodobná
  + Sběr instancí – seberu co nejvíce instancí, které má nasazený algoritmus zpracovávat
  + Více instancí je zdroj variance
  + Pro každou hodnotu zadané metriky provádím měření na více instancích
  + Statistické zpracování umožní potlačit varianci
* **Statistika pro jednu hodnotu zadané metriky**
  + Nejoblíbenější – průměr, medián
  + Poctivější – kontrola **statistického rozložení** – nemusí být uniformní ani normální
  + Parametry rozložení (střední hodnota, variance, …): komprese získaných dat
  + Příprava pro důležitou kvalitativní interpretaci – identifikace a charakterizace statistického rozložení (proč je to mix dvou rozdělení? Proč jsou některé instance o tolik těžší?)
* **Srovnání dvou algoritmů A, B**
  + Na základě parametrů rozložení:
    - A má lepší všechny parametry A je lepší
    - Jinak nevíme
  + **Na základě dominance**:
    - Pro každou instanci (zadané metriky) je A lepší nebo stejně dobrý A dominuje
  + Nevíme hlubší analýza, kde a proč je A nebo B lepší
* **Odvozené metriky**
  + **Primární metriky** – přímo měřené hodnoty
  + **Vizualizace, vzhled** – histogramy apod.
  + **Kvantitativní srovnání – sekundární metriky**:
    - Průměr, medián
    - Ad hoc kombinace primárních metrik (pokuty za nevyřešení)
    - Úplné charakteristiky statistického rozložení
    - Distribuční funkce, korigované distribuční funkce
    - Křížení distribučních funkcí
* **Randomizovaný algoritmus**
  + **Vyhodnocení na jedné instanci** – jedna instance algoritmus statistika, interpretace
    - **Srovnání na jedné instanci**
      * Počet kroků algoritmu jako náhodná proměnná
      * Histogramy úspěšných běhů
    - **Distribuční funkce** – Estimated Cumulative Distribution Function – ECDF
      * Pro každý krok pravděpodobnost, že algoritmus skončil nejvýše v tomto kroku
      * Počítá se z úspěšných běhů – jinak je na konci skok
      * Korekce na úspěšnost – jak do CDF promítnout, že algoritmus v nějakém počtu případů neuspěl
        + Pro každý krok pst, že algoritmus **úspěšně** skončil nejvýše v tomto kroku
        + Stačí přenásobit CDF úspěšností
  + **Vyhodnocení na sadě instancí**
    - Generátor instancí – všechny instance se zadanou metrikou jsou stejně pravděpodobné
    - Algoritmus – všechny hodnoty generátoru náhodných čísel jsou stejně pravděpodobné
    - Statistika 1 – potlačení variance z randomizace
    - Statistika 2 – potlačení variance v instancích
    - Interpretace
  + **Dva zdroje variance** – dva stupně zpracování
  + Protože statistické rozložení vůči char. vstupní instance nemusí být stejné jako rozložení vůči RNG
  + Vlastně statistika ze statistik
  + Přes hodnoty RNG – pro každou instanci tolik spuštění algoritmu, až dostaneme spolehlivá data
  + Přes instance – jako bez randomizace
* **Metriky 2. fáze**
  + Potlačení variance od instancí
  + Statistika ze statistik
  + Co je důležité pro odpověď z experimentu – co vypovídá nejvíc
    - Charakteristiky rozložení
    - Dominance obecně a dominance v zajímavých oblastech – např. malé instance nebo krátké časy nejsou důležité
    - Úspěšnost do určitého počtu kroků, atd.
* **Robustnost heuristiky**
  + Zpřeházení pořadí proměnných ve formuli (stejná booleova funkce) a SAT solver dá jiné řešení v jiném čase
  + Zpřeházením pořadí deklarací proměnných v programu a ten funguje rychleji
  + Závislost práce heuristiky na popisu instance (na rozdíl od instance samotné)
  + Např. lexikálním uspořádání dat, které nemění význam
  + Lexikální uspořádání může být důsledkem
    - Reprezentace množiny vektorem
    - Výběru první konfigurace z množiny stejně dobrých
  + Náhodný výběr místo postupného provádění randomizace
  + **Měření robustnosti**
    - Před algoritmem – náhodná perturbace (všechny jsou stejně pravděpodobné)
      * Ve statistice měření variance vyvolané perturbací
* **Interpretace experimentu**
  + **Data** – chování na konkrétních instancích, s konkrétními parametry
    - Kvantitativní, pohled zvenčí, mnoho jednotlivých dat
  + **Interpretace** – obecný (obecně užitečný) závěr
    - Nutná extrapolace – obecně málo spolehlivá – vyžaduje důvěru
    - Nejčastěji kvalitativní
    - Často vyžaduje porozumění
    - Pohled zevnitř
    - Jednoduchá formulace

**Obsah obrázku text, snímek obrazovky, Písmo, diagram

Popis byl vytvořen automatickyPrezentace výsledků**

* **Experiment**
* **IMRaD**
  + **Obsah obrázku text, snímek obrazovky, Písmo, řada/pruh

    Popis byl vytvořen automatickyIntroduction** – kontext výzkumu, proč byla studie provedena, jaká otázka, jaká hypotéza, proč experimentální přístup
    - Vybuduje výchozí bod pro další výklad
    - Důvod zahrnuje chybějící nebo neúplné znalosti, problémy
  + **Methods** – kdy, kde, jak provedeno, proč takové metody, co se podniklo k ověření korektnosti (RNG, …), jaký experimentální materiál, proč takový materiál
  + **Results** – jaké výsledky, jaká odpověď, hypotéza potvrzena?
    - Všechna relevantní data a pouze relevantní data
  + **Discussion** – co odpověď může znamenat, vztah k dosavadnímu výzkumu, perspektivy pro budoucí výzkum
    - Všechno, co podporuje tvrzení článku, všechno, co se nezdá podporovat tvrzení článku
* **Přesvědčení a data**
  + Tabulky bez grafů – nepřesvědčí, grafy bez tabulek – nejdou použít pro další výzkum
  + Chyby: Chybějící sloupce v tabulkách, grafy bez srozumitelného sdělení, nevýstižné, matoucí popisky, grafy/obrázky vyžadující dlouhý popis – složité sdělení, neúmyslné lži (lexikografické pořadí instancí, oslí můstky, posunuté počátky pokud to nedává smysl)

**13. Princip lokálních heuristik, pojem globálního a lokálního minima, obrana před uváznutím v lokálním minimu.**

NI-KOP

**Stavový prostor**

* **Stav** – jedno ohodnocení proměnných
* **Stavový prostor** – dvojice (S, Q), S = množina stavů, Q = množina operátorů
* **Akce** – aplikace operátoru na stav
* **Graf** stavového prostoru algoritmu – orientovaný graf, kde hrany odpovídají akcím
* **Okolí** stavu – množina stavů dosažitelných ze stavu aplikací některé operace
* K-okolí stavu – množina stavů dosažitelných ze stavu aplikací nejméně jedné a nejvýše k operací
* **Sousedn**í stavy – stavy z okolí stavu
* **Inverzní operátory** – libovolnou vloženou věc lze vyjmout, v grafu lze libovolně dlouho bloudit
* **Stavový prostor TSP/HC**:
  + Konfigurace – podgraf
  + Uzel stavového grafu = podgraf
  + Operátor = např. dvojzáměna na hranách
* Pokud je úkol nalézt cestu, je množina stavů posloupností akcí
* **Prohledávání stavového prostoru**:
  + **Úplná strategie** – navštíví všechny stavy kromě těch, o kterých víme, že nedávají (optimální) řešení
  + **Systematická strategie** – úplná strategie, která navštíví každý stav nejvýše jednou

**Heuristiky**

* **Heuristika** = přístup k řešení používající metody, které negarantují nalezení optimálního řešení
* Požadavky na heuristiku:
  + **Použitelnost v praxi** – metoda musí pracovat na praktických instancích, s přijatelnou přesností a přijatelným výpočetním časem
  + **Omezení a optimalizace** – nejlepší čas pro potřebnou přesnost, nejlepší přesnost v časovém limitu
  + **Přesnost** – přibližné a pro nás uspokojující řešení
* **Druhy heuristik**
  + **Konstruktivní heuristika** – začne z triviální konfigurace a postupnými kroky konstruuje řešení
  + **Iterativní heuristika** – začne z nějakého (i neplatného) řešení a to postupně vylepšuje
  + **Dvojfázové heuristiky** – první fáze slouží k získání řešení (konstruktivně, náhodné řešení) , ve druhé fázi iterativní vylepšování
* **Druhy heuristických metod**
  + **Lokální metody** – práce vždy jen s aktuálním stavem, může uvažovat i blízké okolí, např. sousedy
  + **Globální metody** (ne heuristiky) – dekompozice instance na menší instance téhož problému
    - **Rozděl a panuj –** možnost rekurze
* **Heuristická funkce**
  + Stavu s je přiřazena hodnota (odhad)
  + Preferovány jsou stavy s vyššími/nižšími hodnotami
  + Při shodné hodnotě u více stavů lze vybrat náhodně
* **Hladové heuristiky** – rozhodují ve prospěch lokálního optima
  + Doufáme, že nalezneme globální optimum

**Lokální heuristiky**

* U náročnějších problémů je prohledávání neúnosné
* Jednoduché heuristiky – jen jeden aktuální stav, koukají na sousedy (okolí)
* Cyklus heuristiky:
  + Kontrola ukončující podmínky
  + Pokud není ukončeno, výběr stavu z okolí
  + Pokud je lepší, označí se jako nejlepší
  + Pokud není, vrací se do prvního kroku bez nalezení nejlepšího
* Náhodná procházka
  + Posun na jakéhokoliv souseda, i horšího
  + Není úplná ani systematická
* **Best-only** – metoda nejrychlejšího sestupu/vzestupu
  + Pokud je nový stav lepší, tak se nahradí za momentální
    - Cyklus přes Q, takže se opravdu vybere nejlepší ze všech možností
  + Vrátí prázdný stav, pokud neexistuje lepší soused
  + Pořadí procházení množiny Q neovlivní výsledek, pokud nejlepších stavů není více
* **First improvement**
  + Znovu cyklus přes Q, ale jakmile se nalezne lepší řešení, tak return = první zlepšení
  + Vrátí prázdný stav, pokud neexistuje lepší soused
  + Pořadí procházení množiny Q ovlivní výsledek – randomizace
* **Návrh heuristiky** a jejího stavového prostoru
  + Chceme mnoho jednoduchých, ale rychlých akcí
  + Chceme mnoho akcí, které nemění konfiguraci drasticky (občas menší množství radikálních akcí)
* Okolí heuristik **Kernighan-Lin** – heuristiky pro TSP a bisekci grafu
  + Aplikace daného operátoru opakovaně až do stop podmínky bez ohledu na optimalizační kritérium nebo heuristickou funkci
  + Z takto získaného okolí výběr nejlepšího stavu jako příštího
* **Pohyb v prohledávacím prostoru**
  + Strategie mohou, ale nemusí být systematické
  + Typický krok prohledávání: výběr proměnné, výběr hodnoty proměnné
  + Prořezávání se vztahuje na oblast stavového prostoru
  + Možnost odvolat nastavení proměnné (backtracking, …)
* **Práce s heuristikou**
  + **White box fáze** – omezená sada instancí, detailní měření, vzhled, porozumění, modifikace heuristiky
  + **Black box fáze** – plná sada instancí, měření výsledků, ověření kvality a výkonu, žádné modifikace heuristiky

**Lokální minima**

* **Lokální minimum** – všechny sousední stavy mají horší hodnotu optimalizačního kritéria, ale není to ta nejnižší hodnota v celém SP
* **Globální optimum** – stav, ve kterém je hodnota optimalizačního kritéria nejlepší mezi všemi možnými stavy
* **Uváznutí v lokálních minimech** – kvalita výsledného řešení silně závisí na kvalitě počátečního řešení
* **Řízení úniku z lokálních minim**
  + **Diverzifikace** – rovnoměrný průzkum stavového prostoru
    - Velká ochota připustit akci, která vede k horšímu řešení
  + **Intenzifikace** – konvergence k finálnímu řešení
    - Malá ochota připustit akci, která vede k horšímu řešení
* **Řešení diverzifikace**
  + Zvětšení okolí – pravidelné k-okolí – exponenciální velikost s k
    - Okolí Kernigham-Lin
    - Dynamické řízení okolí
  + Pokročilejší heuristiky:
    - Připuštění akce, která **zhorší řešení** (simulované ochlazování)
      * Nutné řízení – intenzifikace vs. Diverzifikace
    - Práce s **více stavy/konfiguracemi** (genetické algoritmy)
      * Různé způsoby interakce mezi současně zpracovávanými stavy
      * Spuštění z více různých náhodných počátečních stavů (randomizované algoritmy)
    - **Mapování stavového prostoru** (částečné), modelování stavového prostoru, učení
    - Backtracking – Algoritmus vycouvá z minima návratem – může být časově náročné
    - Restriktivní opatření, kterým stavům se vyhnout (tabu search)

**14. Princip genetických algoritmů, význam selekčního tlaku pro jejich funkci.**

NI-KOP

* Simulovaná evoluce
  + **Více stavů najednou** – snížení možnosti uváznutí v jediném minimu, zpravidla konstantní počet
  + Operátory – unární x binární
  + Definice operátorů – nad reprezentacemi, problémově nezávislé
  + Konfigurace = **jedinec** (fenotyp)
  + Kódování konfigurace – genetická reprezentace jedince **(genotyp**, chromozom)
  + Proměnná kódování – **gen**
  + Hodnota proměnné – alela
  + Aktuální množina reprezentací konfigurací – **generace**, populace
  + Unární operátor – **mutace**
  + Binární operátor – **křížení**
  + Optimalizační kritérium – **zdatnost (fitness)**
  + Rozšíření kvalitní konfigurace – **konvergence**
  + Rozšíření konfigurace uvázlé v lokálním minimu – **degenerace**
  + Biodiverzita – diverzita populace
  + Obsah obrázku text, diagram, snímek obrazovky, Písmo

    Popis byl vytvořen automaticky
  + Explorace = diverzifikace, exploitace = intenzifikace
  + **Genetické algoritmy**
    - Kódování – **binární řetězec**
    - Binární operátory – křížení – 1-bodové, 2-bodové, uniformní
    - Unární operátory – mutace, inverze
    - Řízení populace – nová generace nahradí původní
  + **Evoluční strategie**
    - Kódování – **vektor reálných čísel**
    - Strategické parametry – mutace, interakce mezi složkami
    - Operátory – mutace (dominuje) – přičtení Gaussova rozložení
    - Křížení – diskriminující, průměrující
    - Řízení populace – z x rodičů a y potomků se vybere x členů nové generace, nebo náhrada vcelku
  + **Genetické programování**
    - Kódování – **rozkladový strom** výrazu
    - Operátory – křížení, mutace, definice stavebního bloku, editace
    - Řízení populace – nová populace nahradí starou
  + **Evoluční programování**
    - Kódování **– automat**
    - Operátory (pouze unární)
      * Změna výstupního symbolu
      * Změna přechodu
      * Přidání/vypuštění stavu
      * Změna počátečního stavu
    - Řízení populace – z x rodičů a y potomků se vybere x členů nové generace
  + Společné rysy
    - více individuí
    - prostředky diverzifikace – mutace, …
    - prostředky intenzifikace – selekce pro rekombinaci, selekce pro další generaci
  + charakteristické rysy
    - jaká reprezentace
    - jaké unární, binární operátory
    - jaká selekce pro rekombinaci a konstrukci následující generace
* **Genetické algoritmy**
  + **Binární řetězec**
  + Vektor proměnných obecně různých domén
  + Permutace řetězce z dané abecedy
  + Genetické operátory – nastavení pravděpodobnost mutace, pravděpodobnost křížení
  + **Křížení** (rekombinace) – jednobodové (náhodný bod), dvoubodové (dva náhodné body), uniformní (vektor náhodných 0/1)
  + **Inverze** – zpřeházet pořadí bitů, ale ponechat význam proměnných
  + **Mutace** – náhodně vybraný bod z celé populace
    - Někteří jedinci mohou být mutování vícekrát, někteří vůbec
  + **Selekce** – účel je způsobit, aby početní zastoupení jedince v populaci odpovídalo jeho zdatnosti
    - Řízení selekčního tlaku – převod informace ze zdatnosti na početnost
    - Selekční tlak = pravděpodobnost výběru nejlepšího jedince
      * Velký – nebezpečí degenerace
      * Malý – pomalá konvergence
      * Šum vnesený mutací může převážit nad pomalou konvergencí – divergence
      * Regulace selekčního tlaku se liší podle výběrového mechanismu
  + **Ruletový výběr** – rozevření políčka rulety odpovídá žádané pravděpodobnosti výběru
    - Provedeme m náhodných voleb úhlu, odtud m prvků
    - Jeden prvek může být vybrán vícekrát
  + **Řízení selekčního tlaku pro ruletový výběr** – přímá úměra mezi zdatností a poměrnou pravděpodobností výskytu často zdegeneruje populaci – je třeba nadržovat slabším
    - Přepočítání zdatnosti lineární funkcí – scaling – Lineární škálování
    - Použití pořadí ve zdatnosti místo zdatnosti (ranking)
  + **Turnajový výběr** – náhodně vybrat r jedinců (turnaj) a z něj nejlepšího
    - Opakovat až do naplnění populace
    - Selekční tlak se řídí velikostí turnaje
    - „turnaj“ s jedním jedincem – žádný selekční tlak
    - Turnaj přes celou populaci – jistota výběru nejlepšího jedince
  + **Zkrácený výběr** – z populace M jedinců se vybere pM nejzdatnějších, každý vstupuje do rekombinace 1/p krát
    - Informace obsažená ve zdatnosti je redukována na jediný bit
    - Variace – několik stejně velkých skupin podle zdatnosti, pro každou skupinu četnost výběru, redukce informace na typicky 2 bity
  + **Řízení populace**
    - Náhrada en bloc – nová generace vzniklá křížením a mutací nahradí starou
    - Částečná náhrada – z x rodičů a y potomků se vybere x členů nové generace
    - Ustálená populace – po každém křížení potomek nahradí nejslabšího jedince
    - Přechodové formy mezi en bloc a ustálenou populací
    - **Elitismus** – několik málo nejlepších jedinců přejde ze staré populace do nové (pozor na degeneraci)
    - Velikost populace
      * Malá (10) – nebezpečí ztráty diverzity
      * Méně obtížné problémy – od 30 jedinců, lineární růst
      * Obtížné problémy – 100 jedinců
      * Velké instance – sublineární růst s velikostí populace, technické důvody
  + podmínky ukončení – pevný počet generací, příznaky konvergence
    - změna průměrné zdatnosti mezi generacemi
    - Rozložení zdatnosti v generaci
  + **Omezující podmínky** – co když výsledek genetického operátoru není řešení
    - **Relaxace** – převod omezujících podmínek na penalizaci
      * Velikost penalizace – každé řešení lepší než ne-řešení, minimální penalizace, která zabrání aby bylo ne-řešení vybráno jako optimální
      * Způsob penalizace – vzdálenost od řešení (odhad ceny opravy, počet porušených podmínek)
    - **Trest smrti** – zahodit výsledek
      * Zmarněná práce – výpočet optimalizačního kritéria, genetický operátor
      * Snížená dostupnost stavového prostoru
    - **Doménové reprezentace a operátory** – viz permutační operátory
    - **Dekodéry –** volba reprezentace tak, aby každý genotyp odpovídal řešení
      * Každé řešení musí být poskytnuto
      * Každý výstup dekodéru je řešením
      * Každé řešení reprezentováno týmž počtem genotypů
      * Lokalita – malá změna genotypu působí malou změnu řešení
      * Rychlý výpočet
* **Schémata a teorie stavebních bloků**
  + Reprezentace stavebních bloků – schémata – co mají chromozomy společného?
  + **Implicitní paralelismus** – každý genotyp o n genech obsahuje 2^n schémat
    - Každý výpočet zdatnosti získává údaj o průměrné zdatnosti všech schémat
    - Zpracování populace o velikosti M znamená zpracování O(M^3) schémat
    - Jiný pohled na GA:
      * Populace jako množina schémat
      * Vývoj populace jako vývoj zastoupení schémat
    - GA jako zpětnovazební dynamický systém – ve smyčce obíhají schémata, převod zdatnosti schémat na jejich zastoupení
      * Věta o schématech – selekce s pravděpodobností úměrnou zdatnosti, jednobodové křížení
      * Krátká schémata malého řádu přežívají ve smyčce snáze
      * Větší schémata potřebují lepší zdatnost
      * Mutace zkracuje životnost
  + **Linkage problém**
    - Životnost a vývoj schémat závisí na jejich délce, nikoliv pouze na řádu
      * Uniformní křížení zpracovává správně pouze schémata řádu 1
    - Řešení – dynamické přeuspořádání, explicitní práce se stavebními bloky
  + Zavádějící, klamné funkce – kombinace podobných funkcí – optimalizační kritérium testovacích úloh
    - Reálné úlohy mohou skrytě mít podobný charakter
  + **Kompetentní GA**
    - Klasický GA nezachází optimálně se schématy vyšších řádů, protože o nich neví
    - Řešení:
      * Práce se stavebními bloky explicitně jako s oddělenými fragmenty genetické informace – messy GA
      * Práce s pravděpodobnostním modelem vazeb mezi hodnotami proměnných, které vytvářejí stavební bloky – bayesovská optimalizace
* **Fast messy GA**
  + Fragment genetické informace – někde nedospecifikovaný, někde přespecifikovaný
  + Hodnocení schémat – referenční jedinec (kompetetivní šablona) poskytne informaci pro nedospecifikovaná místa
    - Bereme v úvahu první výskyt hodnoty pro dané místo genotypu
  + **Generování schémat**
    - Deterministicky – vygenerovat všechny a vyhodnotit (messy GA)
    - Stochasticky – vygenerovat schémata většího řádu s rovnoměrnou pravděpodobností
      * Při filtraci využít selekce a implicitního paralelismu (fast messy GA)
  + **Filtrace schémat** – generovaná schémata mají řád k < l
    - Selekce turnajovým výběrem
      * Velikost turnaje 2, prvý účastník zvolen náhodně
      * Druhý účastník musí mít společnou určitou míru informace
      * Soutěží nápady na podobné téma
      * Zůstává zdatnější
    - Zkrácení schématu vyjmutím náhodného genu
    - Zkrácená schémata musí být znovu vyhodnocena
  + **Rekombinace schémat**
    - Rozdělení (cut) obou rodičů v náhodných bodech
    - Spojení (splice) rozdělených fragmentů
    - Obě operace mají nezávisle nastavenou pravděpodobnost
* Algoritmy založené na statistických modelech závislostí
  + Obsah obrázku text, diagram, snímek obrazovky, Písmo

    Popis byl vytvořen automaticky
  + **Bayesovská optimalizace** – po odhadu modelování bayesovskou sítí – optimalizační algoritmus a metrika
  + **Bayesovská síť** – každá proměnná reprezentace modelována uzlem orientovaného acyklického grafu
  + Užití bayesovských sítí
    - Najít nejpravděpodobnější vysvětlení pozorovaných pravděpodobností
    - Zjistit pravděpodobnosti hodnot proměnných v závislosti na vstupních pravděpodobných
  + Optimalizace bayesovských sítí
    - Metrika – jak dobře vystihuje měřená síť současnou populaci
    - Algoritmus – pouze nejlepší nebo jiná lokální heuristika
      * Operace – přidání hrany, obrácení hrany, odebrání hrany
* **Paralelizace evolučních algoritmů**
  + **Granularita**
    - Jedna aktuální generace v paralelním systému
      * Sdílená paměť
      * Výpočet zdatnosti, křížení, mutace se snadno škálují s počtem procesorů
      * Minimální a maximální zdatnost, ranking – nutná komunikace, logaritmická složitost
    - Jedna aktuální generace na procesor – nutná občasná výměna genetického materiálu (např. se sousedními procesory)
      * Napomáhá zachovat diverzitu genetického materiálu
* Island parallel GA

**15. Princip simulovaného ochlazování, význam parametrů a způsoby jejich řízení.**

NI-KOP

**Simulované ochlazování**

**Obsah obrázku text, Písmo, snímek obrazovky, řada/pruh

Popis byl vytvořen automaticky** Obsah obrázku text, Písmo, snímek obrazovky, algebra

Popis byl vytvořen automaticky

* **Algoritmus iterativní optimalizace**
  + Frozen, equilibrium a cool určují průběh teploty – rozvrh ochlazování
  + Princip algoritmu:
    - Náhodně se zvolí počáteční řešení, volba počáteční teploty
    - Náhodný výběr souseda
      * Pokud je lepší, přechod
      * Pokud ne, zjistí se, jak moc je horší
        + Pokud je dostatečně vysoká teplota a zhoršení dostatečně malé, přejdeme
    - Po x krocích snižování teploty (ne pokaždé, aby to nebylo moc rychle)
      * x = **délka ekvilibria**
      * Ovlivňuje pravděpodobnost přechodu do horšího stavu
    - Po dostatečném snížení teploty konec
* Vlastnosti simulovaného ochlazování
  + **Diverzifikace** – rovnoměrný průzkum stavového prostoru
    - Velká ochota připustit akci, která vede k horšímu řešení
  + **Intenzifikace** – konvergence k finálnímu řešení
    - Malá ochota připustit akci, která vede k horšímu řešení
  + **Počáteční stav** – řešení z jiné heuristiky nebo náhodná řešení
  + **Vysoké teploty** – převaha diverzifikace
    - Vysoká pravděpodobnost přijetí horšího řešení
  + Nízké teploty – převaha intenzifikace, konvergence k minimu
* **Ochlazovací funkce**:
  + … číslo kroku
  + … hloubka lokálního minima
  + Proces po nekonečném počtu kroků skončí v globálním minimu – asymptotická konvergence
* **Parametry**
  + **Equilibrium a koeficient ochlazování**
    - Teplota se vynásobí parametrem mezi 0 a 1. většinou 0.8 – 0.999
    - Správná míra ochlazování lze za běhu řídit délkou ekvilibria (konstantní nebo proměnná délka)
    - Příliš velký koeficient – výpočet trvá dlouho bez zlepšení
    - Příliš nízký – malé prohledávání, riziko uváznutí v lokálním minimu
    - **Délka ekvilibria** – ovlivňuje počet okolních stavů, které prohledáváme v každém teplotním skoku
      * Správná hodnota závisí na míře ochlazování
      * Příliš vysoká – dlouhý výpočet bez zlepšení
      * Příliš nízká – uváznutí v lokálním minimu
  + **Počáteční teplota**
    - Známe hloubku lokálních minim nastavíme teplotu tak, aby pravděpodobnost úniku z minima byla např. 0.5
    - **Zpětnovazební řízení**
      * Rychle zvyšujeme teplotu
      * Sledujeme četnost přijatých změn k horšímu
      * Zaznamenáme teplotu pro pravděpodobnost např. 0.5
      * Vrátíme původní stav a nastavíme teplotu
    - Příliš vysoká – přijmutí i velkých zhoršení
    - Příliš nízká – riziko uváznutí v lokálním minimu v důsledku nedostatečného prohledávání stavového prostoru
  + **Frozen** – ukončovací podmínka
    - Četnost změn (jakýchkoliv) klesla pod nastavenou mez
    - Četnost změn k lepšímu klesla pod nastavenou mez
    - Pevná mez teploty
    - Koncová teplota
      * Příliš vysoká – riziko uváznutí v lokálním minimu
      * Příliš nízká – výpočet trvá dlouho bez dalšího zlepšení výsledku
* **Cenová funkce**
  + Lze spočítat, i když konfigurace není řešením (výpočet optimalizačního kritéria)
  + Navádí optimalizaci stále blíže k řešení (hodnocení „špatnosti“ konfigurace)
  + Možnosti implementace
    - Konfigurace, která není řešení, má **mnohem horší cost** než jakékoliv řešení
      * Konvergence vždy k řešení
      * 2 úlohy, 2 fáze – nalezení řešení, vylepšení řešení
        + Nebudou se střídat
        + Mohou potřebovat jiná řešení
    - Konfigurace, která není řešením, může mít **stejné cost** jako nějaké řešení
      * Lepší vlastnosti stavového prostoru
      * Konvergence k „ne-řešení“ je možná
* **Počáteční řešení**
  + **Náhodná počáteční řešení**
    - Vícenásobné spuštění
    - Měření iterativní síly
    - Dobře aplikované simulované ochlazování není závislé na počátečním řešení – těžiště práce v iteracích
  + **Konstruktivní počáteční řešení**
    - Chytrá konstruktivní fáze vede k hlubokému lokálnímu minimu
    - Je to aspoň nějaké minimum
* **Omezující podmínky**
  + Např. vyhození nevalidních sousedů, relaxace je vhodnější
  + **Relaxace** – náhrada omezujících podmínek přirážkou
  + **Omezující podmínky** – co když nemohu zabránit, aby akce převedla řešení na konfiguraci, která řešením není?
    - Relaxace, zahození kandidátní konfigurace, oprava kandidátní konfigurace
* Vývoj heuristiky
  + **White box evaluation** – omezená sada instancí, detailní měření, vzhled, porozumění, modifikace heuristika a adaptačních mechanismů
  + **Black box evaluation** – plná sada instancí, měření výsledků, ověření kvality a výkonu

**16. Výkonnostní měřítka paralelních algoritmů, PRAM model, APRAM model, škálovatelnost.**

NI-PDP

**Výkonnostní měřítka paralelních algoritmů**

* **Paralelní časová složitost** závisí nejen na n, ale i na počtu procesorů/jader p
  + p = # procesorů = # jader = # vláken
  + **Paralelní čas**  = čas, který uplynul od začátku paralelního výpočtu do okamžiku, kdy poslední (nejpomalejší) procesor skončil výpočet
  + závisí na architektuře paralelního výpočtu hodnocení par. Algoritmu musí vždy brát v úvahu architekturu počítače
  + je meřen čítáním:
    - Výpočetních kroků – aritmetické, logické, paměťové operace
    - Komunikačních kroků – přenos a výměna dat mezi procesory
* **Paralelní cena –** 
  + Většinou statická alokace výpočetních jader
  + Výpočet začíná vytvořením vláken a ty jsou použita k výpočtu až do konce, i když některá mohou být nějakou dobu neaktivní (idle)
  + Měřením kvality je součin procesory – čas = paralelní cena
  + Sekvenční složitost =
  + **Cenová optimalita** – paralelní algoritmus má optimální cenu, pokud
    - Cena je optimální právě tehdy když
* **Paralelní zrychlení** , nebo asymptoticky
  + Paralelní zrychlení je lineární, právě když (
  + **Lineární zrychlení** = nejvyšší cíl paralelního programu – jestliže stoupne -krát, chceme, aby klesnul -krát – obtížně splnitelné
  + **Superlineární zrychlení** – výjimečně dosažitelné

1. Sekvenční algoritmus je paměťově náročnější, než kapacita paměti a souhrnná kapacita pamětí paralelního systému je dostatečná a při paralelním výpočtu ušetříme swapování mezi hlavní pamětí a diskem

* Sekvenční algoritmus znevýhodněn tím, že běží za jiných HW podmínek než paralelní

1. Anomálie při prohledávání kombinatorického stavového prostoru

* **Paralelní efektivnost** relativní vytížení jader dedikovaných paralelnímu výpočtu během výpočtu
  + Vždy (komunikační a synchronizační režie)
  + zrychlení na jádro
  + Konstanta
  + Paralelní algoritmus má konstantní efektivnost, jestliže
* **Paralelní optimalita výkonnosti** – z definic plyne, že paralelní algoritmus je cenově optimální má lineární zrychlení má konstantní efektivnost

**PRAM model**

* **PRAM** = paralelní RAM = výpočetní model
  + Množina p procesorů
    - 1 procesor vlastní lokální (soukromá) paměť + index i
  + M sdílených paměťových buněk (pole)
  + Každý p může přistoupit do jakékoliv buňky sdílené paměti v O(1) čase
  + Řešení konfliktů – explicitní ošetření
* **PRAM algoritmus**
  + Vstup = n položek v (obvykle prvních) n buňkách sdílené paměti
  + Výstup = n’ položek v n’ buňkách sdílené paměti
* Procesy provádí synchronně 3 typy instrukcí:
  + **READ** – čtení sdílené buňky
  + **LOCAL** – lokální operace
  + **WRITE** – zápis do buňky sdílené paměti
* Komunikace procesorů = READ/WRITE sdílené buňky
* PRAM algoritmus lze zapsat regulárními výrazy
* **Jednotkový** model R/L/W trvá čas 1
* **Globální** model L trvá 1 a R/W trvají konstantní čas
* Ošetřování konfliktů při přístupech do sdílené paměti
  + **EREW** – Exclusive Read Exclusive Write
    - Žádné 2 procesory nesmějí číst nebo zapisovat tutéž sdílenou pam. Buňku současně
  + **CREW** – Concurrent Read Exclusive Write
    - Současná čtení 1 b. povolena, ale v 1 okamžiku může jen 1 proces zkoušet zapsat do dané buňky
  + **CRCW** – Concurrent Read Concurrent Write
    - Povoleny současné čtení a zápisy téže buňky
    - **Priority**-CRCW-PRAM
      * Procesy mají pevné priority – dokončení zápisu povoleno procesu s nejvyšší prioritou
    - **Arbitrary** CRCW-PRAM
      * Dokončení zápisu povoleno náhodně vybranému procesu (algoritmus nesmí činit žádné předpoklady, který proces byl vybrán)
    - **Common**-CRCW-PRAM
      * Všechny procesy smí dokončit zápis, pokud jsou všechny zapisované hodnoty stejné
        + Každý a. musí zajistit splnění podmínky
        + Jinak a. není správný a stav PRAM není definován

**APRAM model**

* **APRAM** = asynchronní PRAM
  + Procesy pracují asynchronně, neexistují centrální hodiny
  + READ, WRITE, LOCAL jako PRAM
  + Nutná explicitní synchronizace – **bariérová synchronizace**
  + Doba přístupu do sdílené paměti není jednotková
* **APRAM výpočet** = posloupnost globálních fází, ve kterých procesory pracují asynchronně, oddělených bariérovou synchronizací
* Dva + procesory nemohou přistupovat do téže buňky v téže globální fázi, pokud jeden z nich do ní zapisuje
* Výkonnostní parametry:
  + Lokální operace – 1
  + Globální READ/WRITE: d
  + K po sobě jdoucích globálních R/W: d+k-1
  + Bariérová synchronizace: B(p)
* 2 možné implementace bariéry
  + **Centrální čítač**
    - Inicializovaný na 0 a na příchozí fázi, procesy přistupují ve vzájemném vyloučení

1. Proces dorazí k bariéře, zkontroluje, zda je v příchozí fázi a inkrementuje čítač
2. Je-li čítač , proces se deaktivuje
3. Jinak nastaví bariéru do odchozí fáze a aktivuje ostatní procesy
4. Poslední aktivovaný proces nastaví bariéru do příchozí fáze
   * **Binární redukční strom**
5. Každý proces dorazí k bariéře a zkontroluje, zda je v příchozí fázi
6. Čeká, až skončí redukce v jeho podstromu
7. Po jejím skončení pošle signál rodiči
8. Kořen stromu počká na redukci z obou podstromů a přepne do odchozí fáze

**Škálovatelnost**

* **Amdahlův zákon saturace paralelizace**
  + Každý sekvenční algoritmus A s časem nad daty o velikosti n se proporčně skládá z

1. Inherentně sekvenčního podílu , který může provést pouze 1 vlákno
2. Paralelizovatelného podílu
   * Nechť A je paralelizován pro pevné n pomocí vláken
   * Pak pro zrychlení A platí při vláknech ideálně:

Obsah obrázku Písmo, číslo, řada/pruh, text

Popis byl vytvořen automaticky

* + Nezávisle na tom, kolik vláken bylo použito, nemůže zrychlení přesáhnout
  + Po jisté hranici nemá přidávání procesů už smysl, bo pro něj není dost paralelní práce
  + Problém fixní velikosti poskytuje omezené množství paralelismu a tudíž při provedení i omezuje použitelný počet paralelních vláken/jader
* **Gustafsonův zákon**
  + S rostoucím p máme úměrně navyšovat i velikost problému n
  + Pak sekvenční část trvá konstantní čas nezávisle na (V/V operace, inicializace), kdežto inherentně paralelní část bude lineárně škálovat s  v čase
  + pro monotónně rostoucí
  + **Paralelní škálovatelnost** – schopnost par. Počítače se zvětšit, pokud narůstá velikost řešeného problému
    - **Silná** – schopnost p. a. pro fixní dosáhnout lineárního zrychlení s rostoucím
      * Měří pokles efektivnosti, pokud roste a se nemění
    - **Slabá** – definuje, jak se mění par. Čas s  pro fixní
      * Měří růst takový, že při rostoucím zůstává efektivnost stejná
    - **Škálovatelnost** = schopnost p. a. držet paralelní optimalitu, pokud oba a rostou/klesají
  + **Izoefektivní funkce**
    - = asymptoticky minimální funkce taková, že



* + - = asymptoticky maximální funkce taková, že



* + Z Amdahlova zákonu vyplývá:



* + - Abychom si udrželi konstantní efektivnost, musí být procesorů alespoň
  + Z Gustafsonova zákonu vyplývá, že když velikost problému roste s  vztahem



efektivnost nebude klesat

**17. Programování nad sdílenou pamětí, programový model OpenMP, datový a funkční paralelismus, synchronizace vláken, vícevláknové algoritmy (násobení polynomů, násobení matic, řazení).**

NI-PDP

**OpenMP**

* **OpenMP** – explicitní model paralelního výpočtu, kdy má programátor plnou kontrolu a zodpovědnost za paralelní výpočet
* **Paralelní regiony** = části původně sekvenčního kódu
  + V nich pomocí fork-join mechanismu vytvářena, prováděna a ukončována paralelní vlákna

Obsah obrázku diagram, řada/pruh, Plán

Popis byl vytvořen automaticky

* Mimo par. regiony pouze 1 hlavní (master) vlákno
* Podpora pro iterační i funkční model paralelismu
* Pomocí OpenMP direktiv v kódu paralelní regiony, ve kterých bude výpočet prováděn více paralelně běžícími vlákny nebo paralelně běžícími úlohami, kdy je každá úloha prováděna 1 vláknem
* Zákaz skákat z paralelního regionu ven či dovnitř
* Kompilace s -fopenmp
* Tvorba paralelních regionů – **direktiva parallel**
  + Možné klauzule direktivy:
    - **If(podmínka)**: podmínka paralelizace regionu
    - **Num\_threads(výraz)**: počet vláken v paralelním regionu
    - **Vlastnosti(seznam proměnných)**: OpenMP vlastnosti proměnných v paralelním regionu
  + Na konci p. r. je implicitní bariéra
  + Po jejím provedení jsou nová vlákna ukončena a dál pokračuje jen hlavní vlákno 0
  + Pokud je 1 vlákno předčasně ukončeno, jsou ukončena všechna vlákna i celý program
* Vlastnosti proměnných v paralelním regionu
  + **Shared** – daná skalární proměnná (ne pole, ne struktura) je sdílená všemi vlákny
  + **Private** – daná proměnná je lokální ve vláknech – každé vlákno má nezávislou minimalizovanou instanci této proměnné
  + **Firstprivate** – proměnná je lokální ve vlákně, každé vlákno ji má inicializovanou na hodnotu, kterou měla před vstupem do p. r.
  + **Lastprivate** (pouze v paralelních cyklech) – p. je lokální ve vláknech, ale hodnota ze sekvenčně poslední iterace se po skončení p. cyklu překopíruje do proměnné hlavního vlákna procesu
  + **Default** – určuje, jakou z předchozích vlastností budou mít implicitně všechny proměnné použité v paralelním regionu
  + **Reduction** – určuje, že daná sdílená proměnná je lokálně nakopírovaná do každého vlákna a že po skončení par. Regionu se všechny lokální instance této proměnné zredukují pomocí zadaného redukčního operátoru a výsledek bude zapsán do původní sdílené proměnné
    - Musí to být skalární proměnná
    - Redukční operátory: +,\*,-,&,^,|,&&,||
    - Nelze kombinovat s direktivou task
  + **Threadprivate** – def. Globální platnost hodnot lokálních proměnných vláken v rámci celého programu napříč všemi paralelními regiony
  + Počet vláken ve všech regionech musí být stejný, proměnné si „drží“ hodnoty při přestupech mezi p. r.

**Datový a funkční paralelismus**

* **Direktiva for** – přidělení jednotlivých iterací for cyklu uvnitř par. Regionu jednotlivým vláknům
* Na konci par. Cyklu implicitně bariéra
* Možné klauzule:
  + **Schedule**(): upřesňuje způsob přidělení iterací cyklu vláknům
  + **Collapse**(): upřesňuje paralelizaci vnořených cyklů (implicitně for jen na nejvyšší úrovni)
  + **Ordered**(): pořadí provádění iterací je stejné jako při sekvenčním provádění
  + **Nowait**(): vlákna po dokončení svých iterací nečekají na bariéře
* **Klauzule schedule –** schedule(typ) schedule(typ, chunk-size)
  + Typy klauzulí:
    - **Static** – buď jsou vláknům přiděleny staticky cyklicky bloky (=chunks) o velikosti chunk-size, nebo se přidělí rovnoměrně (n/p)
    - **Dynamic** – dynamicky přiřazuje chunky po sobě jdoucích iterací velikosti chunk-size nebo 1
    - **Guided** – vláknům dynamicky přiděleny bloky x iterací, kde
    - **Runtime** – způsob přiřazení zvolen v okamžiku spuštění dle systémové proměnné OMP\_SCHEDULE
    - Auto – přidělení it. Necháno kompilátoru/běhovému prostředí
  + **Efektivita**:
    - Schedule(**static**[,k])
      * Nejmenší režie
      * Rovnoměrné rozdělení iterací
      * Ideální, pokud mají všechny iterace stejnou výpočetní dobu
      * K ovlivňuje promíchání iterací
    - Schedule(**dynamic**[,k])
      * Vyšší režie kvůli synchronizaci
      * Vyšší k snižuje režii
      * Výhodné při kolísavé době iterací
    - Schedule(**guided**[,k])
      * Vyšší režie (synchr.)
      * Vyšší k režii snižuje
      * Výhodné při postupně rostoucí době iterací
* **Paralelizace 2-úrovňového for cyklu**
  + **Statické přidělení** vláken (pro jednoduchost)

# parallel for nebo #parallel for collapse(2)

For(…)

For(…)

Funkce()

* + Paralelizace **pouze vnitřního cyklu**
    - For(…)

#...parallel for

For(…)

Funkce()

* + - # … parallel

For(…)

#...for

For(…)

Funkce()

Nutné, je-li vnitřní smyčka datově závislá na vnější smyčce

* **Task** = úloha
* Podporuje složitější funkční paralelismus s větší režií – vhodné i pro rekurzivní algoritmy (zapouzdření kódu i dat)
* **Přidělování úloh** – typ producent-konzument
  + Vlákna jsou producenti i konzumenti
* **Úloha** = jednotka par. Výpočtu, obsahuje:
  + Ukazatel na začátek svého kódu (k provedení)
  + Vstupní data
  + Dat. Strukturu, do které vloží svůj identifikátor vlákno, jakmile danou úlohu začne provádět jeho konzument (=vlastnické vlákno)
* **#pragma omp task** způsobí:
  + Vlákno – producent vygeneruje novou úlohu a vloží ji do zásobárny úloh (=task pool)
  + Úloha čeká, než ji volné vlákno – konzument vyzvedne a provede
* **Podmíněné spouštění par. Úloh**
  + If(…) – efektivní řízení task par. Rekurzivních kódů, kdy rekurze závisí na splnění podmínky
    - Splněno – synovská úloha do task poolu
    - Nesplněno – pozastavení rodičovské úlohy a odložení do zásobárny úloh, ihned provedení nové synovské úlohy, po dokončení vyzvednutí rodiče a dokončení
  + #pragma omp **taskwait**
    - Rodičovská úloha čeká na dokončení všech synovských úloh
    - Stromová rekurze
* Volání task direktivy musí být uvnitř paralelního regionu
* Rekurzi startuje jediné vlákno
  + #pragma omp parallel num\_threads(…){

# pragma omp single

Funkce()

}

**Synchronizace vláken**

* **Synchronizační direktivy**
  + **Barrier** – místo, kam par. Vlákna daného p. r. musí dorazit a čekat na ostatní
  + **Master** – daný blok kódu smí provést pouze hlavní vlákno
  + **Single** – daný blok kódu smí provést pouze 1 libovolné vlákno
  + **Critical** – vytvoření kritické sekce
  + **Atomic** – operace nad paměťovou buňkou bude provedena jednovláknově a nepřerušitelně
  + **Flush** – propsání aktuálních hodnot daných sdílených proměnných do sdílené paměti
  + **Taskwait** – synchronizace synovských úloh s rodičovskou v task paralelismu
* **Bariéra a serializace** v par. Regionu
  + # pragma omp **single** – následující blok se smí provést pouze jednou – ostatní vlákna čekají na implicitní bariéře za single blokem
  + # pragma omp **master** – následující blok smí provést pouze hlavní vlákno – ostatní pokračují hned kódem, který je za tím
  + # pragma omp **barrier** – synchronizační bod, vlákna uspávána a probouzena, až dorazí všechna vlákna
    - Implicitně na konci par. Regionu a single
* **Kritická sekce**
  + Jedna/více částí kódu par. Regionu, které lze v 1 okamžiku provádět pouze 1 vláknem
  + Direktiva # pragma omp critical – anonymní kritická sekce
  + Několik kritických sekcí – vzájemné vyloučení vstupu vláken platí globálně pro všechny její výskyty
  + Direktiva #pragma omp critical name – pojmenovaná krit. sekce taky může být víckrát, platí to samé
* **Direktiva atomic** a její použití
  + Přístup do pam. Místa se skalárním datovým typem (integer, floating-point, … ) bude atomická operace
    - Nepřerušitelná R/W/RMW
    - Bude deterministický výsledek
  + Read, write, update, capture
  + Inkrementace - #... atomic update
  + Capture – rozšiřuje update o získání hodnoty dané pr. Před/po modifikaci
* Uživatelsky řízené předčasné ukončení par. Regionu – **direktiva cancel**
  + Provedením vydá vlákno ostatním signál k ukončení – přejde na bariéru
  + Další vlákna, která později narazí na cancel provedou totéž
  + Vlákna, která už poslední volání cancel minuly, standardně dokončují
  + # pragma omp cancel construct[if(expr)]
    - Construct [parallel, for, taskgroup, sections]

**Vícevláknové algoritmy**

* **Prohledávání kombinatorického SP**
  + NPH úloha najít určitý 1 stav ve velkém SP
  + Vstupní proměnné, stavové proměnné, výstupní proměnné, omezení, optimalizační kritérium
  + Rozhodovací vs konstruktivní vs enumerační
  + **SB-DFS** – přípustný koncový stav bez optimalizace
  + **BB-DFS** – diskrétní optimalizační problém
    - Přípustný koncový stav s max./min. cenou
  + **PP-DFS** – prohledávání v iteracích se zvyšující se hloubkou SP (např. lineární prohlubování)
    - BB-DFS do hloubky L, pokud nenašlo řešení, prohloubí se
* **Paralelní algoritmy pro PKPS**
  + Čas. Složitost PKPS je superpolynomiální
  + Paralelní prohledávání může mít anomální chování
  + Základní podmínka úspěšného par. PKPS:
    - Jádra by měla být pokud možno stále vytížena prohl. pokud možno disjunktních částí SP
* **Statické rozdělení SP**
  + Nerozlišujeme mezi procesem a vláknem
  + P CPU jader, každé p\_i provádí v 1 okamžiku jedno vlákno
  + Základní postup statického rozdělení výpočtu:
    - Master vlákno – sekvenční BFS – vygeneruje odlišných stavových prostorů s cca stejným počtem nastavených stavových proměnných
    - Prohledávání stavových podprostorů přiděleno vláknům
    - Každé vlákno (včetně hlavního) provede sekvenční DFS přiděleného SPp pomocí lokálního zásobníku
    - Výsledky lokálních PKSP předají hlavnímu vláknu – globální řešení
* **Problémy efektivnosti** statického rozdělení SP
  + P jader by mělo mít podobný výkon a parametry paměti
    - Stejně rozsáhlé podprostory se stejnou sekvenční časovou náročností
  + ALE: navracení (=backtracking) je silně datově závislé
    - Výpočet vláken se může dost lišit
    - Některá pak budou neúčinná – neefektivní
* Obsah obrázku kresba, diagram, Dětské kresby, skica

  Popis byl vytvořen automaticky**Statické rozdělení SP a anomální chování**
  + V případě prohledávání celého SP rozděleného pouze staticky může vzniknout anomální chování:
    - 4-vláknové řešení vpravo pomalejší než 2-vláknové vlevo
  + 2. příklad:Obsah obrázku řada/pruh, trojúhelník, diagram

    Popis byl vytvořen automaticky
    - V případě FSB-DFS:
      * Paralelní DFS s 2 vlákny trvá stejně jako se 4 vlákny
      * Anomálie – přidáním jader může DFS
        + Superlineárně zrychlit
        + Zpomalit
  + Pro efektivní PKSP:
    - Jemnější statická dekompozice v modelu dynamického Master-Slave
    - Doplnění o dynamické vyvažování zátěže
* **Dynamické vyvažování zátěže**
  + generuje podprostory a přiřadí je vláknům
  + – DFS pomocí lokálního zásobníku (= je aktivní)
  + Aktivní vyprázdní zásobník, ale nenajde řešení
    - Stává se nečinným, ale žádá jiná vlákna o přidělení neprozkoumaných částí jejich SP
  + se stane dárcem - = příjemce
  + Půlení zásobníku (rozdělování na k částí – požadavků) - částí
    - Neexp. stavy blízko dna/vrcholu zásobníku skrývají pravděpodobně větší/menší části SP
    - Položky nad řeznou výškou H se nepředávají
* Paralelní algoritmus dyn. **Master-Slave DFS**
  + Hlavní vlákno = Master, dalších vláken = Slaves
  + M – sekvenční BFS podprostory pro vlákna S
  + M pošle každému S 1 podprostor z množiny
  + S po přijetí podprostoru jede sekvenční DFS, nikdy se nevrací za počáteční stav svého zásobníku
  + S neúspěšně ukončí lok. PKSP požádá M o další podprostor (definován lok. Zásobníkem)
    - M má nepřidělené podprostory přidělí S další lokální PKSP
    - M odpoví negativně požádá S o ukončení aktivity
  + FSB-DFS – S nalezne řešení informuje M M oznámí všem S konec
* **Klasifikace efektivně paralelizovatelných algoritmů**
  + **Výpočetně intenzivní algoritmy –** čas procesoru strávený výpočtem nad daty je větší než čas nutný na přesun dat z paměti do CPU (PKSP pro NPH úlohy, …)
  + **Paměťově intenzivní algoritmy –** čas procesoru strávený nad výpočtem je menší, než čas nutný na přesun dat z paměti do CPU
    - Počet výpočetních operací na přenesený bajt/prvek je příliš malý
    - Typicky a. s lineární výpočetní složitostí, kde data přen. Z paměti do CPU použita jen k-krát, kde je malá konstanta
    - Skalární součin, dynamické programování, Fourierovy transformace
  + O smysluplnosti paralelizace rozhoduje typ úlohy
    - Nutná podmínka = teoretické zrychlení
    - Rozhodující je řád výpočetní složitosti nebo multiplikativní konstanta (u lin. Složitosti)
* **Zdroje neefektivity** OpenMP kódů
  + **Nevyvázěná výpočetní zátěž** pro jednotlivá vlákna – bariéra čekající vlákna nevyužitá jádra
  + Příliš **těsná synchronizace –** velký počet bariér/krit. sekcí
  + **Omezený paralelismus –** # iterací for < # vláken
  + **Vysoká režie** správy vláken – častá tvorba/zánik, schedule(dynamic)
  + **Významná sekvenční část –** z Amdahlova zákonu
  + **Neefektivní využívání keš** paměti – falešné sdílení, častý zápis
* **Falešné sdílení –** různá vlákna zapisují na různé adresy, které jsou ale natolik blízké, že jsou namapovány do stejného bloku keš paměti
  + U datového paralelismu typické
  + Zabránění vede na protichůdný požadavek, než je požadavek přístupu se třídou 1 u jednovláknových aplikací
* **Snížení dopadu falešného sdílení**
  + Vhodnější rozdělení iterací cyklu nad dostatečně velkým polem mezi vlákna je blokově rovnoměrné – schedule(static)
  + Vhodná **chunk-size** při statickém/dynamickém přidělování bloků iterací vláknům
    - Pole A začíná na adrese dělitelné cache\_line\_size, čili pole A je v paměti zarovnáno stejně jako bloky keše
  + Umělé navýšení velikosti zapisované datové struktury připojením jalové výplně (dummy data)
    - Např. každý prvek pole navýšen na velikost bloku keše + podm. Zarovnání
    - Dobré pro malá sdílená pole, kde má každé vlákno vyhrazené místo pro zápis svého lok. Výsledku velikosti blok keše

**Paralelizace násobení polynomů, násobení matic, řazení**

* **Násobení polynomů –** vstup = 2 polynomy A, B, výstup = C = A B, sekvenční složitost
  + Sekvenční algoritmus – for cyklus pro init C, potom dvojitý for cyklus pro
  + **Paralelizace vnějšího cyklu –** každé vlákno počítá násobení 1 členu 1. polynomu – kritické zápisy
    - Všechna vlákna čtou postupně všechny prvky polynomu B
    - Oblasti v poli C (zápis) nejsou disjunktní – potřeba vzájemného vyloučení ()
      * uvnitř dvojitého for cyklu
  + **Paralelizace vnitřního j-cyklu –** vlákna sdílejí index i – najednou čtou A[i] (paralelní vnitřní for cyklus)
    - Paralelní zápis je disjunktní, ale + režie opak. Rozdělení m iterací vnitřního cyklu nově vytvořeným p vláknům + režie synchronizace n\*T\_bar
    - Pro 1 člen jsou spočítány násobky
    - 2. možnost: + použití před i-cyklem – vlákna se vytvoří jen jednou v iteracích se rozděluje B na konci každé iterace bariéra
    - Ale vždy vzájemné zneplatnění keše, neodstranitelné falešné sdílení
  + **Paralelizace polynomu C –** nerozdělujeme násobení, ale výsledné členy – každý se počítá zvlášť
    - Pouze paralelní čtení, žádné kolize zápisu, potřeba vyvažování výpočetní zátěže
    - Minimum falešného sdílení – vlákna provedou maximum výpočtů lokálně – zápis pouze výsledku do sdílené paměti
* **Násobení hustých matic –** 2 čtvercové nn matice A, B, výstup = C = A B
  + Sekvenčně – středoškolský algoritmus – n^2 skalárních součinů ř. A a sloupců B – n^3 násobení
    - 2 for cykly pro iteraci přes A a B a třetí for cyklus () pro skalární součin
  + **Paralelizace i-cyklu –** každý řádek matice C = 1 vlákno, zapisovatelné oblasti disjunktní
    - Pouze 1 implicitní bariéra na konci par. regionu
  + **Paralelizace j-cyklu** **s blokovaným přidělením** oblastí – větší synchronizace – n\*T\_barr, bez kolizí
    - Pro velké n/p není falešné sdílení
    - Každé vlákno zapisuje do souvislé části daného řádku C o velikosti n/p prvků
  + **Paralelizace j-cyklu s cyklickým přidělením** iterací – jako předtím synchron. n\*T\_barr + bez kolizí
    - Místo je
    - Falešné sdílení – různá vlákna zapisují současně do sousedních prvků stejné řádky matice C – do stejného keš bloku
  + **Přidělení iterací j-cyklu v paralelním regionu –** stejné jako 2 kromě synchronizace
    - Před for(i<n) – vytvoření p vláken pouze jednou
  + **Paralelizace k-cyklu –** 
    - Skalární součin paralelní redukcí (1 řádek A a 1 sloupec B) – obrovská režie synchronizace
    - Možnost přidat před zapsání C – 1x tvorba vláken, ale rychlejší
* **Násobení řídké matice vektorem –** y = Ax (A n x n, počet nenulových prvků N), algoritmus SpMVM
  + **Formáty pro uložení řídkých matic**
    - **Souřadnicový (COO) –** A reprezentována 3 poli: indexy řádků nenulových prvků, indexy sloupců nenulových p., hodnoty nenulových p. – pořadí typicky po řádcích
      * **Paralelizace**: 2x parallel for, výpočet jako tmp – přičtení za tím jako atomic update
      * Nepřímá indexace – **falešné sdílení** – neefektivní
        + Možno, aby víc vláken přičítalo ke stejnému prvku vektoru y současně
    - **Komprimované řádky (CSR) –** A repr. 3 poli: (položky podle indexů ř. a sl.) indexy do pole A.ColInd, od kterých jsou v něm uloženy indexy sloupců nenulových prvků jednotlivých řádků A; indexy sloupců nenulových prvků; hodnoty nenulových prvků A
      * **Paralelizace:**  před 1. for
        + – řádky distribuovány blokově

Nevyvážená zátěž, falešné sdílení sousedních vláken

* + - * + – řádky přidělovány cyklicky

Taky nevyvážená zátěž, horší falešné sdílení (všechna vlákna)

* + - * + – dyn. přidělování – lepší vyváženost, ale vyšší režie
      * Matice s nepravidelným rozdělením nenulových prvků
        + **Vyvažující metoda** – matice v CSR formátu na p řádkových pásů (=bands), které mají přibližně stejný počet nenulových prvků, každé vlákno 1 pás
        + Pomocí omp parallel a omp barrier (iterace po pásech)
* **Paralelizace QuickSortu**
  + **QuickSort** – varianta Lomuto: pivot třízení prvků L a R
    - Funkční paralelismus – rekurzivní volání na L a R
    - Velké množství malých tasků velká režie, zlepšení:
      * Tail call optimization – odstranění koncové rekurze (tail recursion), resp. Nahrazení 1 ze 2 rekurzivních volání iterací
      * Prahování – zavedení prahu počtu prvků pro vytváření nových OpenMP úloh
      * Paralelizace rozdělování řetězce – varianta Hoare
  + **QuickSort – varianta Hoare**
    - Pole se souběžně prochází zleva i zprava a přitom se porovn. navštívené prvky s pivotem
    - L menší než pivot a R větší rovno pivot oba na správných pozicích; L i R menší než pivot L je správně; L i R větší než pivot R je správně; L větší a R menší prohození
    - Iterace, až se průchody zleva a zprava potkají = neutralizace – v každé i. neutr. min. 1 prvek
    - **Rozdělení řetězce pro rekurzi lze paralelizovat** – indexy průchodů = sdílené proměnné
      * Každé vlákno chce získat unikátní hodnoty těchto proměnných tak, aby mohlo k rozdělení přispět svou prací na disjunktních dvojicích prvků pole
      * Na konci zbyde max. 1 špatně zařazený prvek – levný a rychlý úklid
      * Vyžaduje aktivovaný vnořený OpenMP paralelismus -
        + Vede na strom rekurzivních volání par. vlákna se rozvětvují do par. vláken
      * Operace s indexy pomocí
* **Paralelizace MergeSortu**: **MergeSort** standartní binární rozdělování, problém je s Merge fází
  + **Klasický task** paralelismus je **moc pomalý –** slučování moc krátkých částí, falešné sdílení
    - Zlepšení: Prah počtu prvků; místo 2 nových úloh pouze 1 pro levou část pravá zpracována sekvenčně; paralelizace Merge
  + **Paralelizace Merge –** u sekvenčního průchod 2 seřazenými poli zleva doprava a porovn. počátečních prvků menší do výsledního pole = 2-cestné slučování
    - Binární matice, kde seř. pole A jsou řádky a B sloupce, pokud A[i]>B[j], pak M[i,j]=0, jinak 1
    - Tlustá lomená čára dělí oblast 0 a 1
    - M proložíme p-1 vedlejšími diagonálami v diagonální matici vzdálenosti n/2p od sebe označíme průsečíky s hranicí (x0…xp) oblasti jsou ohraničené pomocí průsečíků
    - Dělení na pásy podle # CPU, průsečíky = oblast čísel rozdělených podle 1 ekviv. počtu
      * Budou se všemi vlákny zapisovat do setříděného pole
    - Každé vlákno pak sekv. sloučí bloky Ai, Bi, na konci zřetězení slouč. pos. od každého vlákna
    - vláken počítá průsečíků za možné falešné sdílení, rychlejší p-cestné
* **Paralelizace p-cestného MergeSortu** – **PMWMS = parallel multi-way MergeSort**
  + P vláken rozdělí vstupní neseř. pole o velikosti n na p částí velikostí n/p, vlákna seřadí sekv. svoje s
  + Po bariéře každé vlákno i > 0 vypočte vektor svých p tzv. **rozdělovačů**

= p indexů do p seřaz. částí pole S[p][n/p] takových, že součet délek disjunktních úseků určených v S[p][n/p] dvěma po sobě jdoucími vektory rozdělovačů split\_vec[i] a split\_vec[i+1] má vel. n/p

* + Po bariéře vlákno i vyřízne podle svých p rozdělovačů svých p disjunktních seřaz. úseků délky n/p
  + Úseky sloučí pomocí sekvenčního p-cestného slučování do seřazeného pole velikosti n/p
  + Výsledný seřazený úsek o velikosti n/p vloží na připravené disjunktní místo B[i] výstupního pole
  + **Splitters\_by\_Rank –** vlákno vezme sdílené pole p seřazených polí s[0]…s[p-1] a pořadí rank svého 1. čísla na výstupu a vygeneruje pole p rozdělovačů těchto polí takových, že počet prvků vlevo od nich sečtený přes všechna pole S[i] se rovná přesně rank
  + Po seřazení jednotlivých částí a výpočtu oddělovačů musí být bariéry
  + **Části:** Sekvenční řazení čísel, provedení p hledání v polích velikosti , sekvenční p-cestné slučování p polí celkové délky
  + N je velké, p malé – bude dominovat první člen

**18. Programování nad distribuovanou pamětí, programový model MPI (vícevláknové procesy, komunikátory, 2-bodové blokující a neblokující komunikační operace, kolektivní operace), paralelní násobení hustých matic, paralelní mocninná metoda.**

NI-PDP

* **MPI** = systém zasílání zpráv mezi procesy
* Komunikace procesů/vláken
  + **OpenMP –** pomocí čtení/zápisů z/do sdílené paměti, podpora pro redukci
  + **MPI –** procesy nesdílí paměť – komunikace zasíláním zpráv, všechny proměnné privátní
    - Redukce pro všechny procesy najednou:
* **Využití sdílené paměti**:
  + **Jen MPI** – na každém jádru 1 či několik MPI procesů – nedělí se o vlákna
  + **MPI + OpenMP** – výpočetní uzel MPI proces(y) pomocí OpenMP dělení na několik vláken, běžících na jádrech
  + **Hybrid** – 1 OpenMP vlákno na jádro
* **Kombinace MPI + OpenMP**
  + Inicializace Vrací v proměnné zaručenou míru spolupráce MPI s vlákny
  + Požadovaná míra spolupráce MPI s vlákny:
    - – pouze MPI, procesy se nedělí na vlákna
    - – vícevláknové procesy s omezením, že pouze hlavní vlákno může zavolat funkce MPI = jednoportový model
    - – vícevláknové procesy s om., že v daném okamžiku smí funkce MPI volat pouze 1 vlákno (volání MPI funkcí je kritická sekce) = jednoportový model
    - – vícevláknové procesy, kde všechna vlákna mohou volat funkce MPI bez omezení = všeportový model
* **Komunikátor** – určuje množinu procesů, v rámci níž probíhá komunikace
* **Intra-komunikátor** – asociovaný s konkrétní skupinou procesů
* – předdef. Intra-kom. Pro všechny MPI procesy
* **Inter-komunikátor** – 2 různé skupiny procesů
* – číslo procesu, – počet procesů
* **Komunikační MPI operace**: 2-bodové (komunikace mezi 2 procesy), kolektivní – (komunik. mezi všemi p.)
* **Základní 2-bodová kom.** – zdrojový p. - – určí cílový p., cílový p. - – určí zdrojový p.
* **Blokující komunikační operace** – příslušná MPI funkce je ukončena teprve po dosažení určitého stavu dané komunikační operace
  + Buf – ukazatel na posílaná data
  + Count – počet posílaných položek
  + Datatype – dat. Typ posílaných dat
  + Dest – číslo cíl. Procesu
  + Tag – značka zprávy
  + Comm – MPI komunikátor
  + Source – číslo zdrojového procesu
  + Status – ukazatel na stavový objekt
  + Zbytek stejně jako u Send
* **Typ přenášených dat**
  + Parametr datatype – typ
  + Základní datové typy - , , …
  + Složitější -
* **Množství** přenášených dat – Lze najednou posílat víc prvků, ale stejného dat. Typu a uložené za sebou v paměti, Parametr count
* **Zdrojový a cílový proces** – Cíl – parametr dest, Zdroj – parametr source
  + Přijetí od 1 konkrétního zdroje od libovolného zdroje -
* **Značky přenášených dat – Tag** – rozeznání sémantického významu zpráv
  + Přijetí konkrétní značky libovolné značky -
* **Stavový objekt** – Proměnná typu
  + Můžeme ignorovat –
  + Struktura s položkami:
    - – číslo zdroj. Procesu zprávy
    - – značka přijaté zprávy
  + Pomocí velikost zprávy
* je blokující – ukončena až když lze modifikovat vstupní buffer
  + Realizuje standardní mód – návrat z funkce nastane, když jsou data:
    - Odeslána cílovému procesu
    - Překopírována do dočasného systémového bufferu pro pozdější odeslání
  + Kvůli odesílání je to nelokální operace
* – realizuje Buffered mode, návrat zaručeně nezávisí na připravenosti příjemce přijímat data, lokální operace
  + Pokud příjem nebyl iniciován, MPI musí odesílaná data uložit do bufferu, který si musí uživatel předtím připravit pomocí
* – Synchronous mode – není návrat, dokud není inicializace přijetí dat, nelokální operace
* – Ready mode – pokud při volání není init příjmu, vrátí chybu, nelokální operace
* **Standardní mód –** MPI rozhodne, jestli použít Buffered/Synchronous – Uživatel to neovládá
  + MPI – stanovisko, že korektní MPI program není na systémovém bufrování závislý
* Tyhle sendy jsou blokující ve smyslu, že po návratu z nich můžeme buffer odesílaných dat přepsat
* Recv je blokující ve smyslu, že po jejich ukončení jsou přijatá data uložená v bufferu a lze je číst
* Neblokující funkce **, , ,**  iniciují odeslání dat a skončí
  + může začít, až když příjemce iniciuje příjem, ostatní libovolně
* Buffer vstupních dat nelze modifikovat, dokud není dokončení komunikační operace explicitně otestováno
* Neblokující funkce iniciuje příjem dat
* Buffer není možné použít, dokud není dokončení operace příjmu dat explicitně otestováno
* Všechny neblokující funkce mají dodatečný parametr – ukazatel na proměnnou typu
  + Vstupní arg. Funkcí, které slouží pro testování/čekání na dokončení těchto komunikačních operací
  + **Testování dokončení** -
    - Neblokující, vrátí okamžitě nebo chybu
  + **Čekání na dokončení** -
    - Blokující, vrátí až tehdy, když jsou data skutečně obdržena
* U neblokujícího příjmu **stavový objekt** až z funkcí Test/Wait, NE z Irecv
  + Parametry a stavový objekt
* Neblokující operace důležité, bo umožňují překrývání volání komunikačních párů – není nutná serializace
* **/** – dokončení libovolné operace
* **/** – dokončení všech operací z množiny
* **Komunikační módy** neblokujících operací
  + Vrací okamžitě nezávisle na splnění dané podmínky
  + Na splnění podm. závisí návrat z funkcí čekání na dokončení neblokujících operací , …
* **,**  – testují příchod zprávy, aniž by zpráva byla přijata
* – nebolokující lokální funkce
  + , pokud existuje zpráva, kterou lze přijmout a která odpovídá parametrům
  + Pak vrátí argumentu stejnou hodnotu, jakou by vrátila operace
  + Jinak vrátí a je nedefinován
* může být a může být
* Sondovaná zpráva nemusí být přijata po té, co byla sondována a danou zprávu lze tedy sond. opakovaně
* – blokující nelokální funkce
  + Vrátí až poté, co existuje zpráva, kterou lze přijmout a která odpovídá parametrům a ve výstupním argumentu vrátí stejnou hodnotu, jakou by vrátila operace
* **Požadavky na implementaci** a – měly by garantovan následující:
  + Je-li zavolán jedním procesem a jiný proces zavolá s kompatibilními parametry, pak se volání úspěšně vrátí KROMĚ případů, kdy zprávu přijme konkurenční funkce
  + Pokud proces aktivně čeká pomocí a odpovídající zpráva byla vyslána, pak volání v konečném čase vrátí , pokud
    - Kompat. zprávu nepřijme konkurenční provedená jiným vl. téhož procesu
    - Taková zpráva nebyla sond. konkurenční operací provedenou jiným vl. téhož proc.
* Volání Iprobe/Probe det. zprávu, kterou by byla přij. ve stejném místě vol. f. se stejnými arg.
* Ve vícevláknových procesech je seznam příchozích zpráv sdílen a může docházet k soupeření vláken o přijetí zpráv kompatibilních s těmi, které vysondovaly předchozími voláními Probe/Iprobe
* **Sondování s rezervací pro budoucí –** pro zajištění větší korektnosti a efektivnosti soupeření
  + Neblokující
    - Oproti Iprobe vrátí v případě, že zpráva existuje, v hodnotě argumentu message hangle na vysondovanou zprávu
  + Message je vstupem volání funkce
    - Před návr. z volání se message handle reset. Na
    - Volání s takovouto hodnotou argumentu message nic nepřijme
* Využití funkcí pro **testování příchodu zpráv**
  + Příchod „volitelných“ zpráv – předčasné uk. výpočtu při nalezení optim. řešení jiným procesem
  + Příjem zprávy neznámé velikosti – zjištění velikosti zprávy pomocí a , alokace bufferu, příjem dat pomocí
* MPI neposkytuje mechanismy pro řešení chyb komunikačního systému
* Chyby způsobené voláním MPI funkce s chybným argumentem, nedostatek zdrojů, …
* **Návratová hodnota MPI funkce** – úspěch/neúspěch
  + Úspěch **,** neúspěch chybový kód
* **Chybový kód** = základ pro obsluhu chyby (=error handler) dané MPI funkce, která se při výskytu chyby zavolá před návratem
* 3 předdefinované obsluhy chyb:
  + **–** chyba násilně ukončen celý pr., k návratu chyb. funkce nedojde
    - Procesy interně zavolají
    - Implicitně asociovaná s
  + neukončí program, vrátí chybový kód funkce
    - Stav MPI výpočtu není po chybě MPI standardem definován
    - Pro diagnostiku stavu a výpis chybového hlášení
  + **–** násilné ukončení procesů spoj. s daným komunikátorem, ale ne všech
* Funkce pro **vytvoření kódu obsluhy chyby**, její navázání na komunikátory, testování vazeb a jejich zrušení:

uživatelsky naprogramované volání obsluhy

**Násobení hustých matic**

* **Násobení hustých matic**: Předpokládám klasický školní algoritmus na násobení matic a blokově-šachovnicové mapování matic
* **Naivní algoritmus**: Každý procesor potřebuje odpovídající submatice pomocí AAG
  + Na závěr se provede lokální vynásobení, časová náročnost: , paměťově neefektivní (nevleze se to do paměti jednoho procesoru)
* Obsah obrázku text, snímek obrazovky, design, typografie

  Popis byl vytvořen automaticky**Cannonův systolický algoritmus**:
  + Rozděluji výslednou matici C do které zapisuju mezivýsledky na P pod-matic, podle toho P procesů – algoritmus zabere kroků (matice A a B mají každá celkem P buněk)
  + Krok 1:
    - Cyklický posuv řádků A a sloupců B ()
      * Matice A: posouvám řádky doleva – 1. řádek o 0, druhý řádek o 1, 3. řádek o 2 atd.
      * Matice B: posouvám sloupce nahoru – 1. sloupec o 0, 2. sloupec o 1 atd.
    - Proces spočítá součin – jedna buňka matice
  + Krok 2:
    - Cyklický posuv A a B (tentokrát vždy o jedno políčko)
      * Každý řádek A o 1 doleva, každý sloupec B o 1 nahoru
    - Proces updatuje příslušnou buňku (takže se zase násobí buňky A a B a potom se to k  přičte)
  + Krok 2 pak opakuju, dokud se celkově neposunu o políček (udělá se -krát podle velikosti matice )
  + Výsledná složitost
  + Kvůli těm točkám jsou na to optimální 2D toroidy
* Obsah obrázku text, snímek obrazovky, Písmo, design

  Popis byl vytvořen automaticky**Foxův algoritmus** – Broadcast-Multiply-Roll:
  + Zase procesů a kroků, každý proces zpracovává jedno políčko – 1 součin
  + Krok 1:
    - Nastavíme (to je jako nějaké číslo posuvu)
    - Broadcast (jeden všem) od políčka (na diagonále) na každém řádku (celému řádku) -
    - Podle toho se inicializuje matice – různých hodnot se násobí hodnotami v  políčkách (procesech):
  + Krok 2:
    - Vybírám nová políčka k boadcastu – „posun doprava“ – cyklický posuv o 1 (na obrázcích modře mezi iteracemi), nastavuju
      * Broadcastovat se budou políčka
    - Sloupce se posunou o 1 nahoru
  + Krok 2 se zase opakuje do kroku
  + Časová složitost, škálovatelnost podobná jako u Cannonova algoritmu
* **Násobení matice vektorem**
  + **Řádkové mapování** – každý proces má jeden řádek a příslušný prvek vektoru
    - Broadcastuje ho všem
    - Všichni mají nově celý vektor a počítají 1 prvek – rychlost závisí na broadcastu
  + **Sloupcové mapování**
    - Každý p. má sloupce a příslušný prvek z  – může spočítat vlastní příspěvek do vektoru , který je zatím ve formě matice bez operace sčítání
    - Máme matici sčítanců v řádcích, co jsou po sloupcích v procesech
    - Řádková redukce přes několik procesů s operací sčítání – složitost jako u řídké matice
  + **Šachovnicové mapování** – každá buňka je proces
    - Každý sloupec potřebuje prvek se stejným indexem, jako je sloupec
    - Jako při řádkovém – distribuce dle sloupce
    - Jako při sloupcovém – tvorba příspěvků a pak redukce s operací sčítání

**Paralelní mocninná metoda**

* **Mocninná metoda**: hledá iterativně největší vlastní číslo, vhodné pro velmi řídkou matici, využití: Google PageRank
  + Opakovaně se násobí matice vektorem:
* **Algoritmus**:
  1. Inicializace – vytvořím nenulový počáteční vektor (typicky x = (1,1,1,1,...))
  2. Násobení matice-vektor (MVM) – vynásobím A vektorem x, vznikne vektor y = Ax (nějaký algoritmus pro násobení řídké matice vektorem)
  3. Normalizace – spočtu normu 𝛼 vektoru y
  4. Aktualizace vektoru – nahradím x normalizovaným y = x/𝛼 (paralelní redukce)
  5. Vyhodnotíme kritérium konvergence, pokud není splněno, opakuju od kroku 2
     + Pokud je splněno, je aproximací dominantního (největšího) vlastního čísla
* **Implementace v MPI**:
  + Předpokládáme řídkou matici, předem neurčená struktura
  + Procesory provádí lokální násobení, dílčí výsledky redukují (MPI\_Allreduce)
  + Máme komunikační skupiny pro řádky, sloupce a diagonálu
    - – provede sloupcový broadcast
    - – zpětně namapuje hodnoty na diagonálu
    - – zjistí normalizaci přes diagonálu
* **Náhodné mapování matice**:
  + Každý proces potřebuje celý vektor a pro výpočet své části
  + Na konci každé iterace je pro synchronizaci vektoru a mezi procesy
  + Po provedení algoritmu má každý proces celý vektor a 𝛼
  + Složitost: paměť, kde
* **Řádkové mapování matice**:
  + Každý procesor potřebuje celý vektor , ale vektor si již můžou rovnoměrně rozdělit
  + Matice rozdělena do horizontálních pásů velikosti
  + Získání vektoru pomocí , potom každý proces lokálně vypočte svou část
  + Normalizace se provede lokálně a vektor se vyřeší normalizovaným výsledkem
  + Rychlejší (nepotřebujeme kopírovat y do x),
  + Složitost:
* **Šachovnicové mapování matice**:
  + Procesy tvoří virtuální 2D mřížku
  + Každý procesor potřebuje jen část vektoru (menší paměťové nároky)
  + Vektor je namapován na hlavní diagonálu
  + Procesy na diagonále posílají hodnoty vektoru po sloupcích (sloupcový broadcast)
  + Paralelně všechny procesory spočítají svou část submatice (lokální výpočty)
  + Provede se nejdřív redukce zpět na diagonálu, a potom redukce diagonály
  + Složitost: Obsah obrázku snímek obrazovky, řada/pruh, Písmo, Vykreslený graf

    Popis byl vytvořen automaticky
  + Rozdělení komunikátorů:
    - Potřebujeme provést paralelní redukci jen ve virtuálních řádcích matice procesů
    - – rozdělí komunikátor podle pole „barev“ (řádková souřadnice)
    - Redukci tak můžeme provádět ve všech řádcích nezávisle na sobě
    - Potřebujeme ale taky komunikátor pro diagonální procesy, nejefektivnější časově i paměťově

**Kolektivní operace**

* Operace prováděné nad všemi MPI procesy – blokující i neblokující verze
  + – OAB (one to all broadcast) – root proces vysílá tutéž zprávu všem ostatním
  + – AOG (all to one gather) – root proces posbírá data ode všech, včetně sebe
    - Stejná velikost informací u všech, jinak s variabilitou funkce
  + – AAG/AAB (all to all gather) – všichni ode všech, včetně sebe, sbírají data
  + – OAS (one to all scatter) – distribuuje části včetně sebe samého
    - Scatter na rozdíl od broadcastu neposílá všem to samé, ale každému různý chunk informace
  + – AAS (all to all scatter) – symetrická distribuce všichni všem

**19. Přímé ortogonální a hyperkubické propojovací sítě paralelních počítačů (definice, vlastnosti, vnořování).**

NI-PDP

**Základní definice a vlastnosti**

* **Přímé propojovací sítě** – lze je popsat souvislými grafy
  + Vrchol = výpočetní uzel = procesory, lokální paměť a směrovač
  + Hrana = komunikační linka
  + Nepřímé sítě – tvořeny přepínači
  + Přímé sítě vlastně propojují jednotlivé směrovače mezi sebou
* **Vlastnosti**
  + **Stupeň** uzlu = počet sousedů uzlu
  + **Obsah obrázku řada/pruh, diagram

    Popis byl vytvořen automatickyBisekční šířka** = velikost nejmenšího hranového řezu grafu na 2 poloviny
  + **Uzlová symetrie** = pro libovolnou dvojici uzlů a existuje automorfismus, který zobrazí na (obrázek)
  + **Souvislost** = minimální počet uzlů/hran, jejichž odebrání způsobí rozpojení grafu
  + **Bipartita** = existuje obarvení vrcholu dvěma barvami tak, že koncové vrcholy každé hrany mají odlišnou barvu
  + **Kartézský součin** – k. s. množin vrcholů, a hrana vede tehdy, pokud vedla původně v jednom nebo druhém grafu
  + **Regularita** – stupeň každého uzlu je roven konstantě
* **Požadavky na propojovací sítě**
  + **Malý a konstantní stupeň uzlu** – technologický požadavek
    - **Řídké grafy** = stupeň je omezený konstantou, počet hran
      * **Hustá topologie** = stupně uzlů jsou rostoucí funkcí
    - Konstantní stupeň nutnou podmínkou pro uzlovou symetrii
  + **Malý průměr a malá průměrná vzdálenost** – pro snížení režie komunikačních operací
    - V protikladu s požadavkem na nízký stupeň uzlů
    - Průměr -uzlové sítě s konstantním stupněm je
  + **Symetrie** – zjednodušený návrh algoritmů
  + **Škálovatelnost**
    - **Inkrementálně škálovatelná topologie** = definovaná pro jakékoli
    - **Částečně škálovatelná topologie** = není definovaná pro jakékoliv
    - **Efektivně škálovatelná topologie** = pro vytvoření -uzlové instance z -uzlové instance je třeba odebrat pouze hran a pak hran přidat
      * Obsah obrázku řada/pruh, skica

        Popis byl vytvořen automatickyChceme-li zmenšit nebo zvětšit danou síť o uzlů, musíme v původní síti přepojit pouze uzlů
      * 2D mřížka: částečně škálovatelná, ne efektivně:
        + Obrázek – zvětšení uzlu o 1, ale přepojení 4 hran
  + **Hierarchická rekurzivita** – graf má h. r. topologii, jestliže obsahuje menší instance sebe sama
    - Množina stejně definovaných grafů různých dimenzí, kde nižší dimenze jsou podgrafy vyšších dimenzí
    - Obvykle částečně škálovatelné
    - Dobrá pro realizaci obvodů a induktivní návrhy a mapování paralelních rekurzivních algo.
  + **Vysoká souvislost a odolnost vůči poruchám** – kvůli spolehlivosti a robustnosti sítí
    - V případě výpadku uzlů nebo spojů by měla síť nabídnout náhradní krátké spoje
    - Chceme malý chybový průměr a malou chybovou průměrnou vzdálenost
  + **Velká bisekční šířka** – pro algoritmy typu binární rozděl a panuj
    - Rozdělení problému na dva stejně velké podproblémy
    - Rekurzivní řešení v obou polovinách paralelně, možná výměna mezivýsledků
    - Sloučení výsledků z obou polovin do konečného výsledku
  + **Podpora pro směrování a kolektivní komunikační operace** – chceme minimální směrování
  + **Vnořitelnost jiných a do jiných topologií**
    - **Vnoření** = má-li komunikační graf par. algoritmu jinou strukturu, odlišnou od topologie propojovací sítě, je třeba efektivně zobrazit graf procesů do topologie sítě
  + **Existence hamiltonovských kružnic či cest a existence 2-barvení**
    - Hamiltonovská kružnice = vnoření -uzlové kružnice – speciální případ vnořitelnosti
    - Kritická vlastnost – zjednodušení návrhu algoritmů, kde jsou procesory označeny čísly a komunikace je posun dat ve směru indexování, např. třídící algoritmy

**Striktně ortogonální (mřížkové) topologie**

* **-rozměrná binární hyperkrychle**

Obsah obrázku řada/pruh, diagram, kruh

Popis byl vytvořen automatickyObsah obrázku text, Písmo, řada/pruh, číslo

Popis byl vytvořen automaticky

* + Uzly = **binární řetězce**
    - Sousední uzly = liší se právě v 1 bitu každý uzel má sousedů a hyperkrychle je **regulární**
  + Stupeň není konstantní, roste logaritmicky – **hustá topologie**
  + Vzdálenost dvou uzlů je Hammingova vzdálenost řetězců (počet odlišných stejnolehlých bitů)
  + Počet uzlů vzdálenosti od libovolného uzlu vždy průměrná vzdálenost
  + Uzlově i hranově **symetrická**
  + **Částečná škálovatelnost** – velikost pouze mocniny 2
  + **Efektivně škálovatelná** – 2 podkrychle apod. – dobré pro řešení problému přidělování procesorů v hyperkubickém počítači
  + Optimální souvislost (rovna stupni uzlu – ), největší možná bisekční šířka
  + **Vyvážený bipartitní** graf, hamiltonovský graf
  + Základní minimální směrovací algoritmus – **e-cube**
    - Cesta : porovnají se bitové řetězce a dimenze hran, ze kterých se cesta skládá, odpovídají zprava doleva souřadnicím, ve kterých se a liší
  + Simuluje efektivně téměř jakoukoli jinou známou topologií – většina topologií je do ní **optimálně vnořitelná**
  + Její hamiltonovské kružnice = **Grayovy kódy**
  + Testovací architektura pro paralelizaci problémů
* **-rozměrná mřížka o velikosti stran ,**

Obsah obrázku řada/pruh, diagram, snímek obrazovky, Barevnost

Popis byl vytvořen automatickyObsah obrázku text, Písmo, číslo, černobílá

Popis byl vytvořen automaticky

* + Sestavena **kartézským součinem** mřížek nižších dimenzí:
  + 1-D mřížka je vlastně řada procesorů
  + Uzly mřížky jsou značeny -znakovými -árními řetězci
  + Dva uzly jsou sousední, právě když se liší v jedné souřadnici o jedničku
  + **Není regulární ani uzlově symetrická** – stupeň uzlu závisí na jeho poloze
  + **Nejsou hranově symetrické** (kvůli tomu tak debilní vzorce)
  + 2-D a 3-D mřížky – topologie s **velkým průměrem**
  + **Částečně škálovatelná** (líp než hyperkrychle) – součin čísel větších než
  + Je **hierarchicky rekurzivní** – obsahuje podmřížky stejné dimenze, ale menších délek stran
  + Má **optimální souvislost** – počet disjunktních cest mezi 2 uzly je roven minimu ze stupňů koncových uzlů
  + **Bipartitní,** ne nutně vyvážená, vždy má hamiltonovské cesty a jestliže má aspoň 1 strana sudou délku, má i hamiltonovskou kružnici
  + Algoritmus pro minimální směrování – **dimenzionálně uspořádané směrování**
    - Při konstrukci cesty mezi lib. 2 uzly se hrany smějí používat pouze v jediném pořadí dimenzí
* **-rozměrný toroid o velikosti stran ,**

**Obsah obrázku diagram, řada/pruh, kostka

Popis byl vytvořen automatickyObsah obrázku text, Písmo, číslo, řada/pruh

Popis byl vytvořen automaticky**

* + **„zabalená mřížka“** – od mřížky se liší jen tak, že každá **lineární řada je** **uzavřena do kružnice** přidáním hrany (=**obalující hrana**), která spojí první a poslední uzel
    - Jednorozměrný toroid je **kružnice** = prstenec
  + Dva uzly jsou sousední, právě když se liší v jedné souřadnici o jedničku **modulo velikost strany** v dané dimenzi
    - Díky tomu je toroid **regulární** (stupeň 2n) a **uzlově symetrický**
    - Automorfismus = přeložení
  + Průměr poloviční oproti stejně velké mřížce
  + Částečně škálovatelné a hierarchicky rekurzivní, dekompozice na kartézský součin stejně jako u mřížek:
  + Algoritmus pro minimální směrování – dimenzionálně uspořádané směrování
    - Kvůli kružnicím může dojít k zablokování
  + Bipartitní, pokud jsou všechny délky stran sudé (kvůli kružnicím), hamiltonovský
  + Komerčně populární pro masové paralelní počítače

**Hyperkubické topologie**

* Nedostatek hyperkrychle – logaritmicky rostoucí stupeň uzlu
  + Proto topologie odvozené z hyperkrychle, které mají její dobré vlastnosti, ale konstantní stupeň
* O motýlcích a spol. obecně platí:
  + Vzniknou **rozvinutím** každého uzlu hyperkrychle **na více uzlů**
  + Kvůli tomu zhoršená částečná škálovatelnost hodnoty apod.
  + Optimální z hlediska průměru – dosahují logaritmický průměr při konstantním stupni uzlů
  + Dobrá bisekční šířka -
* **Zabalený motýlek dimenze**

Obsah obrázku diagram, klipart

Popis byl vytvořen automatickyObsah obrázku text, Písmo, číslo, řada/pruh

Popis byl vytvořen automaticky

* + Vznikne rozvinutím do – kružnice nových uzlů
  + Vrcholy kružnic jsou indexovány
  + Dva druhy hran – **hyperkubické a kružnicové** (proto není hranově symetrický)
    - Kružnicová (motýlková) hrana – v rámci jedné kružnice
  + Vrchol v cyklu je spojen hyperkubickou hranou s vrcholem v cyklu
    - **Hyperkubické hrany** spojují sousední uzly vlevo a vpravo – každý uzel má dva sousedy ve své kružnici a dva sousedy v sousedních kružnicích
      * Takhle vznikají křížové hrany, které spolu s kružnicovými hranami tvoří základní motýlky
  + **Regulární, uzlově symetrický**
  + Není hierarchicky rekurzivní - neobsahuje podkružnice
  + Kvůli kružnicím je **vyvážený bipartitní**, pokud je sudé
  + Vždy existují hamiltonovské kružnice
  + Optimální průměr, řídký graf
* **Obyčejný motýlek dimenze**

Obsah obrázku diagram, snímek obrazovky, řada/pruh, text

Popis byl vytvořen automaticky Obsah obrázku text, Písmo, řada/pruh, číslo

Popis byl vytvořen automaticky

* + Obrázek: rozřezání na
  + Vznikne **ze zabaleného motýlka**, tak, že **rozřízneme každou kružnici v**  **v uzlu na pozici 0** tak, že tento uzel se rozdvojí a vzniklé poloviny se podělí o hyperkubické hrany
  + Místo -uzlových kružnic vzniknou -uzlové **lineární řady**
  + se má k podobně jako mřížka k toroidu
  + Není symetrický, není regulární
  + Dva druhy hran – **přímé a křížové** (hyperkubické)
  + Uzly organizovány do **řad** a do **sloupců** (stupňů)
    - Hrany spojující sloupec se sloupcem = hrany úrovně
  + **Hierarchicky rekurzivní** - obsahuje dva podgrafy
    - Odebrání uzlů ve sloupci nebo ve sloupci
  + **Bipartitní** (střídání barvy po sloupcích), není hamiltonovský
  + Směrování – pouze jedna nejkratší cesta, e-cube směrování
  + Využití – minimální permutační síť – levná náhrada křížového přepínače
* **Vnořovací problém**
  + **Statické vnořování** – máme **graf procesů** a **graf fyzického propojení** výpočetních uzlů, chceme namapovat procesy a uzly tak, aby mohly efektivně komunikovat a nevytěžovat síť
    - Simulační mechanismus, který dovolí, aby se počítač s topologií 1choval jako počítač s topologií 2 a nebyly potřeba algoritmické změny
    - Uvažujeme **zdrojový graf** s **vrcholy** a **hranami**  a **cílovou síť** s **uzly** a **linkami** . Pak **(statické) vnoření** do je dvojice zobrazení , kde:
    - … množina všech **cest** v síti
    - Uspořádaná dvojice zobrazení – procesní uzly se mapují na výpočetní uzly, hrany mezi procesy se však musí namapovat na cesty ve výpočetních uzlech
  + **Měřítka kvality vnoření**
    - **Maximální zatížení cílového uzlu** – maximální počet zdrojových vrcholů namapovaných na jeden cílový uzel
      * Maximální počet procesů, který bude přidělen 1 procesoru
      * Chceme stejnoměrné zatížení (liší se max o 1)
    - **Expanze vnoření** – poměr velikosti cílové sítě (= počet výpočetních uzlů) a velikosti zdrojového grafu (=počet procesů)
      * Větší expanze implikuje dražší simulace
      * Chceme blízkou 1 při jedničkovém zatížení a snižujeme úměrným rovnoměrným zvětšením zatížení uzlů
    - **Maximální dilatace zdrojových hran v cílové síti** – maximální délka obrazů zdrojových hran v cílové síti
      * Po jak dlouhých cestách budou muset v cílovém počítači putovat zprávy posílané mezi procesy, které jsou v zdrojovém grafu sousední
      * Sledujeme, pokud máme přepínání citlivé na vzdálenosti, jinak průměrná dilatace
    - **Maximální zahlcení cílové hrany**
      * **Linkové zahlcení** – maximální počet obrazů zdrojových hran procházejících skrz cílové linky
      * **Uzlové zahlcení** – maximální počet obrazů zdrojových hran procházejících skrz cílové uzly
      * Spíš sledujeme průměrné zahlcení
  + **Kvaziizometrická topologie** – sítě a jsou kvaziizometrické, pokud může být vnořen do a naopak s konstantními měřítky vnoření
    - a jsou výpočetně ekvivalentní, pokud jedna může může simulovat druhou s konstantním zpomalením (implikováno kvaziizometrií)
  + **Vnoření hyperkrychle do nízkorozměrných mřížek**

Obsah obrázku diagram, řada/pruh, skica, Plán

Popis byl vytvořen automaticky

* + - Jde vlastně o mapování logické funkce
      * **Svobodovy a Karnaughovy mapy** – Svoboda lexikograficky, Karnaugh Grayův kód
    - Máme hyperkrychlický algoritmus a chceme, aby běžel na 2D mřížce
    - Hyperkrychle i mřížka jsou **rekurzivní**
    - Využití **Mortonovy křivky** – jednotlivé hyperkubické souřadnice se mapují rekurzivně střídavě ve směrech a :



* + - * Alternativně **lexikografické mapování po řádcích/sloupcích**
        + Po řádcích – horní půlka bitů do 1 dimente, dolní do druhé, sloupce analogicky jako transpozice
    - Tímhle dostaneme **4 podkrychle**, které se mapují do 2D mřížky rekurzivně v tzv. **Z-fraktálu**
    - Hyperkrychli rozdělíme na 4 podkrychle, mřížku na 4 kvadranty
    - Uděláme **vnoření podkrychlí do kvadrantů**
      * Děláme rekurzivně do doby, než se dojde na mřížky 2x2
      * Poté lexikograficky uděláme cestu – Z tvar, který se skládá z malých Z – fraktální křivka

**20. Paralelní algoritmy pro redukci, prefixový součet a segmentový prefixový součet na PRAM, v ortogonálních, hyperkubických a obecných topologiích, aplikace.**

NI-PDP

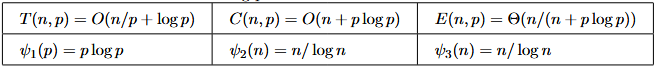
**Paralelní redukce**

* Dáno **vstupní pole** prvků množiny a **binární operace** na
* Cíl = vypočítat hodnotu
* Postačující podmínka pro paralelizovatelnost – **asociativnost** operace
* Optimální triviální algoritmus:

Obsah obrázku text, snímek obrazovky, Písmo, řada/pruh

Popis byl vytvořen automaticky

* **Paralelní výpočet** – operace se musí aplikovat na co nejvíc nezávislých párů vstupních hodnot
* Pokud je na asociativní bin. Operace, potom je paralelní redukce pole o velikosti hodnot z  na procesorech proveditelné na **, přímých a nepřímých stromech, hyperkrychlích a hyperkubických topologiích, mřížkách a toroidech** s následujícími vlastnostmi:



* Paralelní redukce na (a) EREW PRAM, (b) hyperkrychli, (c)1-D WH mřížce, (d) nepřímém stromu

Obsah obrázku nachový, diagram, snímek obrazovky, řada/pruh

Popis byl vytvořen automaticky

* + Bruh je to vlastně součet vrcholů no já se poseru
* Redukce v MPI -
* **Hyperkrychle** – předávání po binomiální kostře
* **Binární strom** nebo motýlek – z listů redukce
* Wormhole (WH) 1D **mřížka** – simulace hyperkrychle
* **PRAM** – redukce každého -tého prvku

**Prefixový součet**

* **Prefixový součet** – zobecnění redukce – taky pole z množiny a asociativní binární operace nad (říkáme jí součet, ale nemusí to být součet, může to být libovolná asociativní operace)
* **Výstup** není jedna hodnota (redukce ), ale **pole stejně velké jako původní pole**
  + je rovno **redukci počátečních hodnot** z
  + Cíl = vypočítat **pole prefixů** pole ,
* Obsah obrázku řada/pruh, Písmo, snímek obrazovky, text

  Popis byl vytvořen automatickyParalelní prefixový součet (PPS) na **EREW PRAM**

Obsah obrázku text, snímek obrazovky, Písmo, číslo

Popis byl vytvořen automaticky

* + jako paralelní redukce
  + V každém kroku seřazeno prvků
  + Pro tvorbu -tého stavu se přičítají všechny prvky s posunem o prvků
* PPS na **nepřímém stomu/motýlku**
  + Pořadí indexace vstupu dáno pořadím listů při průchodu do hloubky
  + **Vnitřní uzly** nevlastní vstupní hodnoty, ale **realizují výpočet**
  + PPS vstupních hodnot v listech binárního stromu výšky lze vypočítat v  krocích. Je-li úplný, PPS potřebuje kroků
  + Výpočet má podobu vzestupné vlny, kdy každý vnitřní uzel čeká na hodnoty z obou podstromů, které sečte a pošle svému rodiči, současně ale **předá hodnotu z levého podstromu do pravého**
  + Vzestupná vlna iniciuje menší sestupné vlny, až dorazí do kořene, tam iniciuje sestupnou vlnu v pravém podstromu
  + Sestupující hodnoty se pouze kopírují do všech listů podstromu, kde jsou přidány k mezivýsledku

Obsah obrázku skica, stativ

Popis byl vytvořen automaticky

* + Celkový počet kroků = nejvýše dvojnásobek výšky stromu

Obsah obrázku diagram

Popis byl vytvořen automaticky

* PPS na **přímém stromu** – modifikace předchozího algoritmu pro p. strom s omezeným větvením
  + Každý uzel stromu **vlastní počáteční hodnotu**
  + Strom není pole, takže musí proběhnout **lineární indexace**
    - Číslování **postorder** – průchod stromu od listů v podstromu zleva
  + Vnitřní uzel při vzestupné vlně čeká na hodnoty ze všech podstromů
  + K nim přidá svoji hodnotu a výsledek pošle rodiči
  + Hodnotu z daného podstromu pošle do všech podstromů vpravo
  + Při sestupné vlně si hodnotu shora započte pro sebe a předá podstromům
  + PPS v  krocích

Obsah obrázku Dětské kresby

Popis byl vytvořen automaticky

* PPS na **hyperkrychli**
  + Rozšíření vysílání všichni-všem v SF modelu, trvá paralelních kroků pro
  + Indexace – zase potřeba **linearizace** – buď **lexikografická**, nebo podle **Grayova kódu**
  + Každý procesor má **2 pomocné registry** a
    - Ve procesy akumulují pouze to, co je zajímá z hlediska prefixového součtu (dáno indexem)
    - V  akumulují vše včetně hodnot, které je nezajímají, ale které mají jako prostředníci předat procesorům, které je potřebují
  + PPS na hyperkrychli je **normální hyperkubický algoritmus** – lze efektivně implementovat na jakékoliv hyperkubické či posuvné síti

Obsah obrázku diagram, Plán

Popis byl vytvořen automaticky

* PPS na ortogonální mřížce
  + SF
    - Mapování vstupního pole na mřížku – podle řádku, sloupce, zig-zag, diagonálně, náhodně
    - Po řádku – 3 fáze:

Obsah obrázku snímek obrazovky, Písmo, nachový, Barevnost

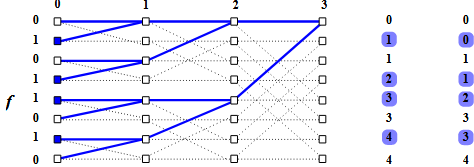
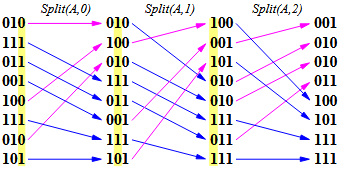
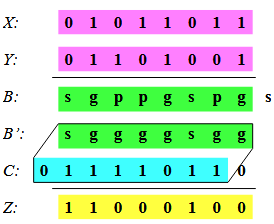
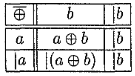
Popis byl vytvořen automaticky

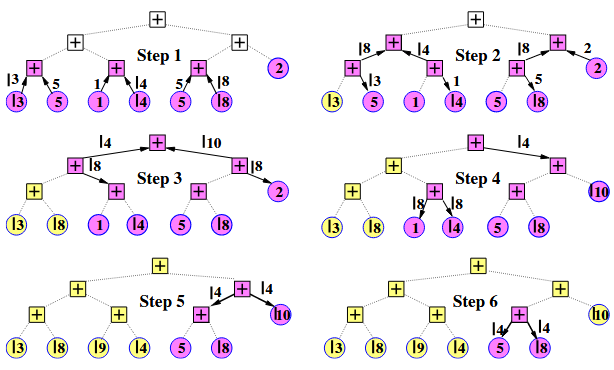
* + - * Fáze 1: horizontální PPS s jednotlivými řádky – uzly nejvíc vpravo drží součet řádku
      * Fáze 2: vertikální PPS se sloupcem nejvíc vpravo – takže uzly nejvíc vpravo budou mít správné globální hodnoty prefixového součtu
      * Fáze 3: uzly nejvíc vpraco pošlou horizontálně globální hodnoty prefixového součtu do jejich řádků tak, aby všechny uzly mohly globalizovat svůj výsledek
  + WH
    - Simulace na 1-D WH mřížce je v podstatě PPS na nepřímém úplném binárním stromu, který se zploští do lineárního pole
    - Oblouky zleva doprava odpovídají vzestupným vlnám, oblouky zprava doleva sestupným vlnám

Obsah obrázku diagram, Šeřík, vzor

Popis byl vytvořen automaticky Obsah obrázku design

Popis byl vytvořen automaticky s nízkou mírou spolehlivosti

* **Aplikace prefixového součtu**
  + **Zhušťovací problém** (packing problem)
    - Podmnožina z  procesorů připojených ke vstupům nepřímé vícestupňové sítě (nD motýlek) má paket, který je potřeba dopravit na 2. výstupní stranu sítě tak, aby i-tý paket odshora skončil na i-tém výstupním vodiči shora
    - Máme množinu procesů, každý má příznak 0 nebo 1 (1 = patří do distribuované množiny)
    - Chci určit pořadí procesů označených 1 (např. dle pořadí se rozhoduje místo pro zápis)
  + **Paralelní RadixSort** – řazení v lexikografickém pořadí
    - Nejdřív řadí podle jednotek, desítek apod.
    - Zhušťování dle každého lexikografického symbolu a přehazování se zachováním pořadí
  + **Paralelní sčítačka s predikcí přenosu**
    - ****Aby operace netrvala
    - 2-bitová čísla – paralelně sčítáním – sečteme jako
    - = určitě carry, = ne, = možná carry – řetěžec doplníme zprava a vyměníme za – z  se stávají 1, z  0, posuneme o jedno místo – máme
    - Sčítámě
* **Segmentový prefixový součet – SPPS**
  + Zobecnění PPS pro případ, kdy je vstupní pole **rozděleno do různě velkých segmentů**
  + Cíl = vypočítat všechny **prefixové součty uvnitř segmentů izolovaně**, rovnoměné zatížení procesorů
  + SPPS se provádí jako globální PPS nad celým polem, ale s **modifikovanou operací,** jejíž tabulka se odvodí z tabulky
    - Vertikální čáry = dané číslo je na levé hranici nějakého segmentu, nebo vlevo od něj už jsou pouze levé hraniční prvky
    - Operace je stejná jako ta původní, ale navíc si všímá, zda její operandy jsou levými hraničními prvky segmentů
    - Pokud ano, žádná hodnota zleva se nesmí za tuto hranici dostat
    - Aplikace současně značku levé hranice posouvá doprava
    - Je-li asociativní, pak je taky asociativní



* Aplikace segmentového prefixového součtu – **paralelní segmentový QuickSort**
  + Rozdělení pole podle procesů (rovnoměrně)
  + Vstupní posloupnost A je rovnoměrně rozdělena mezi procesorů
  + V 1 iteraci hlavní smyčky je každý segment rozdělen na 3 podsegmenty
  + Pivot pro daný segment je jeho první prvek zleva