Opakování

Vzorové písemné práce

- 2 vzorové písemné práce na webu předmětu
- **Struktura písemné práce**: 7 otázek
  1. 1 přehledová otázka
  2. otázky na propagaci a konzistenční algoritmy
  3. otázky na stromové prohledávání prohledávání
  4. otázky na lokální prohledávání a optimalizace
  5. 1 otázka na řešení problému v OPL

Vzorové písemné práce

- 2 vzorové písemné práce na webu předmětu
- **Struktura písemné práce**: 7 otázek
  1. 1 přehledová otázka
  2. otázky na propagaci a konzistenční algoritmy
  3. otázky na stromové prohledávání prohledávání
  4. otázky na lokální prohledávání a optimalizace
  5. 1 otázka na řešení problému v OPL
- **Hlavní typy otázek pro 2,3,4**
  - příklady k vypočítání (jako vzorové příklady – někdy kratší verze)
  - pojem (a příklad)
  - kód algoritmu (a příklad)
  - porovnání pojmů/algoritmů (a příklady)

Příklady s řešeními na webu I.

Příklady 1.

- binarizace
- AC-1 vs. AC-3
- konzistence mezí vs. hranová konzistence

Příklady 2.

- strom stavového prostoru: backtracking vs. kontrola dopředu vs. pohled dopředu

Příklady 3.

- omezení s váhami

Příklady 4.

- přehled konzistenčních algoritmů
- algoritmus AC-4
- algoritmus DAC

Příklady s řešeními na webu II.

Příklady 5.

- strom stavového prostoru: backtracking vs. kontrola dopředu
- pohled zpět: Gaschnigův skok zpět

Příklady 6.

- přehled prohledávacích algoritmů
- problém rozvrhování výuky v OPL

Příklady 7.

- binarizace
- podpora hodnoty
- konzistence mezí vs. hranová konzistence
- algoritmus DAC, nalezení řešení bez navracení

Příklady s řešeními na webu III.

Příklady 8.

- strom stavového prostoru: backtracking vs. kontrola dopředu vs. pohled dopředu
- problém plánování projektu v OPL

Příklady 9.

- stromový CSP, nalezení řešení bez navracení
- algoritmus PC-2
- problém rozvrhování výuky jedné místnosti v OPL
- strom stavového prostoru: kontrola dopředu vs. pohled dopředu bez iniciální konzistence
- rozvrhování úkolů pro zaměstnance na směny

Příklady s řešeními na webu IV.

Příklady 10

- nalezení podpory
- algoritmus DAC a nalezení řešení bez navracení
- problém přiřazení poboček k obchodům v OPL
- BBS-DBS, DDS
- rozvrhování na směny

Pojem: směrová konzistence a šířka grafu

Co to je šířka grafu a jaký je její význam? Použité pojmy objasněte.

- Šířka grafu je minimum z šířek všech jeho uspořádaných grafů.
- Šířka uspořádaného grafu je maximum z šířek jeho vrcholů.
- Šířka vrcholu v uspořádaném grafu je počet hran vedoucích z tohoto vrcholu do předchozích vrcholů.
- Uspořádaný graf je graf s lineárním uspořádáním vrcholů.
- Šířka grafu nám říká, jak silnou úroveň směrové i-konzistence potřebujeme, abychom nalezli řešení problému bez navracení.

Algoritmus: lokální prohledávání

- Napište (nebo alespoň slovně popište) algoritmus prohledávání s tabu seznamem. Je Váš algoritmus kombinován s nějakou další metodou lokálního prohledávání?

```
procedure TSHC(MaxZmen)
θ := náhodné ohodnocení proměnných    % iniciální přiřazení
PocetZmen := 0
while E(θ) > 0 ∧ PocetZmen<MaxZmen do
      vyber ⟨V,a⟩ s nejlepší evaluací tak, že
            není v tabu seznamu a nebo splňuje aspirační kriterium
      přidej ⟨V,c⟩ do tabu seznamu, kde c je současná hodnota V
      smaž nejstarší položku v tabu seznamu
      přiřad' a do V
      PocetZmen := PocetZmen+1
return θ
```

Algoritmus je kombinován s metodou stoupání/hill climbing.

Příklady stručněji: omezení s váhami

- Jaká je úroveň konzistence pro následující CSP problém s váhami s omezeními c1 a c2?

  P=({c1,c2},{X,Y})

  dom(X)=dom(Y)={r,s}

  con(c1)={X,Y}

  def(r,r)=1, def(r,s)=2, def(s,r)=4, def(s,s)=6

  con(c2)={Y}

  def(r)=0, def(s)=1

  Výsledek: blevel(P)=1. Nutno uvést stručný postup, např.

Příklady stručněji: omezení s váhami

- Jaká je úroveň konzistence pro následující CSP problém s váhami s omezeními c1 a c2?

```
P=({c1,c2},{X,Y})
dom(X)=dom(Y)={r,s}
con(c1)={X,Y}
def(r,r)=1, def(r,s)=2, def(s,r)=4, def(s,s)=6
con(c2)={Y}
def(r)=0, def(s)=1
```

Výsledek: blevel(P)=1. Nutno uvést stručný postup, např.

Spočítáme $(c_1 \otimes c_2) \Downarrow_{\varnothing} () =$

$= \min(\mu_{c_1 \otimes c_2}(r,r), \mu_{c_1 \otimes c_2}(r,s), \mu_{c_1 \otimes c_2}(s,r), \mu_{c_1 \otimes c_2}(s,s)) =$

Příklady stručněji: omezení s váhami

- Jaká je úroveň konzistence pro následující CSP problém s váhami s omezeními c1 a c2?

  P=({c1,c2},{X,Y})

  dom(X)=dom(Y)={r,s}

  con(c1)={X,Y}

  def(r,r)=1, def(r,s)=2, def(s,r)=4, def(s,s)=6

  con(c2)={Y}

  def(r)=0, def(s)=1

  Výsledek: blevel(P)=1. Nutno uvést stručný postup, např.

  Spočítáme $(c_1 \otimes c_2) \Downarrow_{\varnothing} () =$

  $= \min(\mu_{c_1 \otimes c_2}(r,r), \mu_{c_1 \otimes c_2}(r,s), \mu_{c_1 \otimes c_2}(s,r), \mu_{c_1 \otimes c_2}(s,s)) =$

  $= \min(1+0, 2+1, 4+0, 6+1) = 1$

Příklady stručněji: omezení s váhami II.

- A jaká je úroveň splnění pro problém P1=({c1,c2},{Y})?
  - výsledek: úroveň splnění pro Y=r: 1, pro Y=s: 3, postup:

P=({c1,c2},{X,Y})

dom(X)=dom(Y)={r,s}

con(c1)={X,Y}

def(r,r)=1, def(r,s)=2, def(s,r)=4, def(s,s)=6

con(c2)={Y}

def(r)=0, def(s)=1

Příklady stručněji: omezení s váhami II.

- A jaká je úroveň splnění pro problém P1=({c1,c2},{Y})?
  - výsledek: úroveň splnění pro Y=r: 1, pro Y=s: 3, postup:

    Spočítáme

    $$(c_1 \otimes c_2) \Downarrow_Y (r) = \min(\mu_{c_1 \otimes c_2}(r,r), \mu_{c_1 \otimes c_2}(s,r)) = \min(1+0, 4+0) = 1$$

    $$(c_1 \otimes c_2) \Downarrow_Y (s) = \min(\mu_{c_1 \otimes c_2}(r,s), \mu_{c_1 \otimes c_2}(s,s)) = \min(2+1, 6+1) = 3$$

```
P=({c1,c2},{X,Y})
dom(X)=dom(Y)={r,s}
con(c1)={X,Y}
def(r,r)=1, def(r,s)=2, def(s,r)=4, def(s,s)=6
con(c2)={Y}
def(r)=0, def(s)=1
```

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

    total == sum (w in 1..nbWorkers) effectivity[w][W[w]];

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

    total == sum (w in 1..nbWorkers) effectivity[w][W[w]];
- Binární proměnné dvar boolean
  - problém bat'ohu: je předmět v bat'ohu?

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

    total == sum (w in 1..nbWorkers) effectivity[w][W[w]];
- Binární proměnné dvar boolean
  - problém bat'ohu: je předmět v bat'ohu? dvar boolean take[1..nbItems];

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

    total == sum (w in 1..nbWorkers) effectivity[w][W[w]];
- Binární proměnné dvar boolean
  - problém bat'ohu: je předmět v bat'ohu? dvar boolean take[1..nbItems];

    sum (i in Items) (take[i] * weights [i]) <= capacity;

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

    total == sum (w in 1..nbWorkers) effectivity[w][W[w]];
- Binární proměnné dvar boolean
  - problém bat'ohu: je předmět v bat'ohu? dvar boolean take[1..nbItems];

    sum (i in Items) (take[i] * weights [i]) <= capacity;
  - Sudoku: hodnota pole

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

    total == sum (w in 1..nbWorkers) effectivity[w][W[w]];
- Binární proměnné dvar boolean
  - problém bat'ohu: je předmět v bat'ohu? dvar boolean take[1..nbItems];

    sum (i in Items) (take[i] * weights [i]) <= capacity;
  - Sudoku: hodnota pole dvar boolean sudoku[1..Size][1..Size][1..Size];

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

    total == sum (w in 1..nbWorkers) effectivity[w][W[w]];
- Binární proměnné dvar boolean
  - problém bat'ohu: je předmět v bat'ohu? dvar boolean take[1..nbItems];

    sum (i in Items) (take[i] * weights [i]) <= capacity;
  - Sudoku: hodnota pole dvar boolean sudoku[1..Size][1..Size][1..Size];

    forall (i,k in 1..Size) sum (j in 1..Size) sudoku[i][j][k] == 1;

Doménové proměnné

- Celočíselné proměnné dvar int+
  - Sudoku: hodnota pole dvar int+ sudoku[1..Size][1..Size] in 1..Size;

    forall (i in 1..Size) allDifferent(all (j in 1..Size) sudoku[i][j]); //ruzne pro kazdy radek
  - přiřazení pracovníků:pracovník vyrábí produkt

    dvar int+ W[1..nbWorkers] in 1..nbProducts;

    total == sum (w in 1..nbWorkers) effectivity[w][W[w]];
- Binární proměnné dvar boolean
  - problém bat'ohu: je předmět v bat'ohu? dvar boolean take[1..nbItems];

    sum (i in Items) (take[i] * weights [i]) <= capacity;
  - Sudoku: hodnota pole dvar boolean sudoku[1..Size][1..Size][1..Size];

    forall (i,k in 1..Size) sum (j in 1..Size) sudoku[i][j][k] == 1;

    navíc: forall (i, j in 1..Size) sum(k in 1..Size) sudoku[i][j][k] == 1;

Rozvrhování: zdroje

- Jeden zdroj
  - jednotková doba trvání úlohy: dvar int, allDifferent(casy)
  - odlišné doby trvání úloh: dvar interval, dvar sequence, noOverlap(casy)

Rozvrhování: zdroje

- Jeden zdroj
  - jednotková doba trvání úlohy: dvar int, allDifferent(casy)
  - odlišné doby trvání úloh: dvar interval, dvar sequence, noOverlap(casy)
- Více zdrojů
  - nepožadujeme určení zdroje pro úlohu
    - bez doménových proměnných pro určení zdroje úlohy

      cumulFunction, pulse(casy[uloha], zdroju[uloha])

Rozvrhování: zdroje

- Jeden zdroj
  - jednotková doba trvání úlohy: dvar int, allDifferent(casy)
  - odlišné doby trvání úloh: dvar interval, dvar sequence, noOverlap(casy)
- Více zdrojů
  - nepožadujeme určení zdroje pro úlohu
    - bez doménových proměnných pro určení zdroje úlohy

      cumulFunction, pulse(casy[uloha], zdroju[uloha])
  - požadujeme určení zdroje pro úlohu

    včetně doménových proměnných pro určení zdroje úlohy

    dvar interval prirazeni ... optional

    alternative(casy[uloha], ... prirazeni[uloha][zdroje], zdroju[uloha])

Rozvrhování II.

- Vztahy mezi úlohami a precedence endBeforeStart, endAtStart, ...

```
tuple Zavislost { int Pred; int Po; }
{Zavislost} zavislosti = ...;
forall (zav in zavislosti) endBeforeStart(casy[zav.Pred], casy[zav.Po]);
```

Rozvrhování II.

- Vztahy mezi úlohami a precedence endBeforeStart, endAtStart, ...

```
tuple Zavislost { int Pred; int Po; }
{Zavislost} zavislosti = ...;
forall (zav in zavislosti) endBeforeStart(casy[zav.Pred], casy[zav.Po]);
```

- Volitelné úlohy presenceOf(dvar interval)

```
dvar interval prirazeni[1..Uloh][1..Stroju] optional;
```

Rozvrhování II.

- Vztahy mezi úlohami a precedence endBeforeStart, endAtStart, ...

```
tuple Zavislost { int Pred; int Po; }
{Zavislost} zavislosti = ...;
forall (zav in zavislosti) endBeforeStart(casy[zav.Pred], casy[zav.Po]);
```

- Volitelné úlohy presenceOf(dvar interval)

```
dvar interval prirazeni[1..Uloh][1..Stroju] optional;
forall (u in 1..Uloh, s not in mozne[u] ) presenceOf( prirazeni[u][s] ) == 0;
forall (u in 1..Uloh, s in prikazane[u]) presenceOf( prirazeni[u][s] ) == 1;
```

Rozvrhování II.

- Vztahy mezi úlohami a precedence endBeforeStart, endAtStart, ...

```
tuple Zavislost { int Pred; int Po; }
{Zavislost} zavislosti = ...;
forall (zav in zavislosti) endBeforeStart(casy[zav.Pred], casy[zav.Po]);
```

- Volitelné úlohy presenceOf(dvar interval)

```
dvar interval prirazeni[1..Uloh][1..Stroju] optional;
forall (u in 1..Uloh, s not in mozne[u] ) presenceOf( prirazeni[u][s] ) == 0;
forall (u in 1..Uloh, s in prikazane[u]) presenceOf( prirazeni[u][s] ) == 1;
```

- Dovolená: práce lidí se nepřekrývá s jejich dovolenou

Rozvrhování II.

- Vztahy mezi úlohami a precedence endBeforeStart, endAtStart, ...

```
tuple Zavislost { int Pred; int Po; }
{Zavislost} zavislosti = ...;
forall (zav in zavislosti) endBeforeStart(casy[zav.Pred], casy[zav.Po]);
```

- Volitelné úlohy presenceOf(dvar interval)

```
dvar interval prirazeni[1..Uloh][1..Stroju] optional;
forall (u in 1..Uloh, s not in mozne[u] ) presenceOf( prirazeni[u][s] ) == 0;
forall (u in 1..Uloh, s in prikazane[u]) presenceOf( prirazeni[u][s] ) == 1;
```

- Dovolená: práce lidí se nepřekrývá s jejich dovolenou

  řešení: přidáme dodatečné intervalové proměnné

```
dvar interval casy[u in 1..Prace+Dovolene] size trvani[u];
forall (u in 1..Prace+Dovolene)  noOverlap(casy);
```

Rozvrhování II.

- Vztahy mezi úlohami a precedence endBeforeStart, endAtStart, ...

```
tuple Zavislost { int Pred; int Po; }
{Zavislost} zavislosti = ...;
forall (zav in zavislosti) endBeforeStart(casy[zav.Pred], casy[zav.Po]);
```

- Volitelné úlohy presenceOf(dvar interval)

```
dvar interval prirazeni[1..Uloh][1..Stroju] optional;
forall (u in 1..Uloh, s not in mozne[u] ) presenceOf( prirazeni[u][s] ) == 0;
forall (u in 1..Uloh, s in prikazane[u]) presenceOf( prirazeni[u][s] ) == 1;
```

- Dovolená: práce lidí se nepřekrývá s jejich dovolenou

  řešení: přidáme dodatečné intervalové proměnné

```
dvar interval casy[u in 1..Prace+Dovolene] size trvani[u];
forall (u in 1..Prace+Dovolene)  noOverlap(casy);
cumulFunction nastroje = sum (p in 1..Prace)  pulse( casy[p],nastroju[p] );
```

Optimalizace

- Problém batʼohu: maximalizace součtu cen předmětů v batʼohu

Optimalizace

- Problém batʼohu: maximalizace součtu cen předmětů v batʼohu

  maximize sum (i in nbItems) (take[i] * prices[i]);

Optimalizace

- Problém batʼohu: maximalizace součtu cen předmětů v batʼohu

  maximize sum (i in nbItems) (take[i] * prices[i]);

- Minimalizace času dokončení poslední úlohy

Optimalizace

- Problém batʼohu: maximalizace součtu cen předmětů v batʼohu

  maximize sum (i in nbItems) (take[i] * prices[i]);

- Minimalizace času dokončení poslední úlohy

  minimize sum (u in 1..Pocet) endOf(casy[u]);

  zadány poslední operace úlohy:

Optimalizace

- Problém bať ohu: maximalizace součtu cen předmětů v bať ohu

  maximize sum (i in nbItems) (take[i] * prices[i]);

- Minimalizace času dokončení poslední úlohy

  minimize sum (u in 1..Pocet) endOf(casy[u]);

  zadány poslední operace úlohy:

  minimize sum (u in 1..Pocet) endOf(casy[u][Posledni]);

Optimalizace

- Problém bat'ohu: maximalizace součtu cen předmětů v bat'ohu

  maximize sum (i in nbItems) (take[i] * prices[i]);

- Minimalizace času dokončení poslední úlohy

  minimize sum (u in 1..Pocet) endOf(casy[u]);

  zadány poslední operace úlohy:

  minimize sum (u in 1..Pocet) endOf(casy[u][Posledni]);

- Maximalice (váženého) počtu realizovaných úloh

Optimalizace

- Problém batʼohu: maximalizace součtu cen předmětů v batʼohu

  maximize sum (i in nbItems) (take[i] * prices[i]);

- Minimalizace času dokončení poslední úlohy

  minimize sum (u in 1..Pocet) endOf(casy[u]);

  zadány poslední operace úlohy:

  minimize sum (u in 1..Pocet) endOf(casy[u][Posledni]);

- Maximalice (váženého) počtu realizovaných úloh

  maximize sum (u in 1..Pocet) (vahy[u] * presenceOf(casy[u]));

Určete čas a místnost pro výuku množiny předmětů. Datová instance:

1. rozvrh tvoříte pro 3 vyučovací dny a každý den má 10 hodin
2. máte zadáno 14 předmětů
3. délka výuky předmětu je 4 nebo 5 hodin (určeno pro každý předmět předem)
4. máte k dispozici 3 místnosti
5. máte zadány 3 třídy (skupiny žáků)
6. každá třída má v zadání určeno 4 až 5 předmětů, které bude navštěvovat

Omezující podmínky:

7. výuka předmětu musí probíhat souvisle bez přerušení (tj. nesmí např. začínat jeden den večer a končit druhý den ráno)
8. v každé místnosti je nejvýše jeden předmět v danou dobu
9. pro každou třídu je zadána množina jejích předmětů; každá třída může mít vždy nejvýše jeden z těchto předmětů v danou dobu

Účelová funkce:

10. Jednotlivé předměty by měly být do rozvrhu umístěny tak, aby výuka všech tříd skončila co nejdříve (např. třetí den dopoledne). Tj. minimalizujte součet koncových časů výuky posledních předmětů všech tříd.

Školní rozvrh: vstupní proměnné

```
int Dny = 3;
int Hodin = 10;
int Predmetu = 14;
int Mistnosti = 3;
int Tridy = 3;
```

Školní rozvrh: vstupní proměnné

```
int Dny = 3;
int Hodin = 10;
int Predmetu = 14;
int Mistnosti = 3;
int Tridy = 3;

range rTridy = 1..Tridy;
range rMistnosti = 1..Mistnosti;
```

Školní rozvrh: vstupní proměnné

```
int Dny = 3;
int Hodin = 10;
int Predmetu = 14;
int Mistnosti = 3;
int Tridy = 3;

range rTridy = 1..Tridy;
range rMistnosti = 1..Mistnosti;

tuple predmetyTrid{
 key int predmet;
 int trida;
 int trvani;}

// např. trida 1 ma predmety 1,2,3; trida 2 ma predmety 4,5, ...
//       predmety 1,2,3,4,5 maji trvani 4,4,5,5,4
// predmet je klic, tj. napr. <2> odkazuje na cely tuple <2,1,4>
{predmetyTrid} PredmetyTrid = {<1,1,4>,<2,1,4>,<3,1,5>,<4,2,5>,<5,2,4>,...};
```

Školní rozvrh: model

Doménové proměnné

Školní rozvrh: model

Doménové proměnné

```
// přiřazení času předmětům v maximálním rozmezí Dny*Hodin, zajištění trvání
dvar interval casy[p in PredmetyTrid] in 0 .. Dny*Hodin size p.trvani;
```

Školní rozvrh: model

Doménové proměnné

```
// přiřazení času předmětům v maximálním rozmezí Dny*Hodin, zajištění trvání
dvar interval casy[p in PredmetyTrid] in 0 .. Dny*Hodin size p.trvani;

// volitelný interval pro přiřazení předmětů a tříd do místnosti
dvar interval prirazeni[PredmetyTrid][rMistnosti] optional;
```

Školní rozvrh: model

Doménové proměnné

```
// přiřazení času předmětům v maximálním rozmezí Dny*Hodin, zajištění trvání
dvar interval casy[p in PredmetyTrid] in 0 .. Dny*Hodin size p.trvani;

// volitelný interval pro přiřazení předmětů a tříd do místnosti
dvar interval prirazeni[PredmetyTrid][rMistnosti] optional;
```

Každý předmět je vyučován právě v jedné místnosti

Školní rozvrh: model

Doménové proměnné

```
// přiřazení času předmětům v maximálním rozmezí Dny*Hodin, zajištění trvání
dvar interval casy[p in PredmetyTrid] in 0 .. Dny*Hodin size p.trvani;

// volitelný interval pro přiřazení předmětů a tříd do místnosti
dvar interval prirazeni[PredmetyTrid][rMistnosti] optional;
```

Každý předmět je vyučován právě v jedné místnosti

```
forall(p in PredmetyTrid)
     alternative(casy[p.predmet], all(m in rMistnosti) prirazeni[p][m]);
```

Školní rozvrh: model

Doménové proměnné

```
// přiřazení času předmětům v maximálním rozmezí Dny*Hodin, zajištění trvání
dvar interval casy[p in PredmetyTrid] in 0 .. Dny*Hodin size p.trvani;

// volitelný interval pro přiřazení předmětů a tříd do místnosti
dvar interval prirazeni[PredmetyTrid][rMistnosti] optional;
```

Každý předmět je vyučován právě v jedné místnosti

```
forall(p in PredmetyTrid)
     alternative(casy[p.predmet], all(m in rMistnosti) prirazeni[p][m]);
```

V každé místnosti je nejvýše jeden předmět v danou dobu

Školní rozvrh: model

Doménové proměnné

```
// přiřazení času předmětům v maximálním rozmezí Dny*Hodin, zajištění trvání
dvar interval casy[p in PredmetyTrid] in 0 .. Dny*Hodin size p.trvani;

// volitelný interval pro přiřazení předmětů a tříd do místnosti
dvar interval prirazeni[PredmetyTrid][rMistnosti] optional;
```

Každý předmět je vyučován právě v jedné místnosti

```
forall(p in PredmetyTrid)
     alternative(casy[p.predmet], all(m in rMistnosti) prirazeni[p][m]);
```

V každé místnosti je nejvýše jeden předmět v danou dobu

```
// sekvence rozvrhu místností z důvodu nepřekrývání
dvar sequence rozvrhMistnosti[m in rMistnosti] in
     all (p in PredmetyTrid) prirazeni[p][m];

forall(m in rMistnosti) noOverlap(rozvrhMistnosti[m]);
```

Školní rozvrh: model II.

Každý předmět je vyučován pro konkrétní třídu (skupinu žáků),

tj. pro každou třídu je zadána množina jejích předmětů

Školní rozvrh: model II.

Každý předmět je vyučován pro konkrétní třídu (skupinu žáků),

tj. pro každou třídu je zadána množina jejích předmětů

```
// sekvence rozvrhu tříd z důvodu nepřekrývání
dvar sequence rozvrhTridy[t in rTridy] in
  all (m in rMistnosti, p in PredmetyTrid : p.trida == t) prirazeni[p][m];
```

a tedy každá třída může mít vždy nejvýše jeden předmět v danou dobu

Školní rozvrh: model II.

Každý předmět je vyučován pro konkrétní třídu (skupinu žáků),

tj. pro každou třídu je zadána množina jejích předmětů

```
// sekvence rozvrhu tříd z důvodu nepřekrývání
dvar sequence rozvrhTridy[t in rTridy] in
  all (m in rMistnosti, p in PredmetyTrid : p.trida == t) prirazeni[p][m];
```

a tedy každá třída může mít vždy nejvýše jeden předmět v danou dobu

```
forall (t in rTridy)
       noOverlap(rozvrhTridy[t]);
```

Školní rozvrh: model II.

Každý předmět je vyučován pro konkrétní třídu (skupinu žáků),

tj. pro každou třídu je zadána množina jejích předmětů

```
// sekvence rozvrhu tříd z důvodu nepřekrývání
dvar sequence rozvrhTridy[t in rTridy] in
  all (m in rMistnosti, p in PredmetyTrid : p.trida == t) prirazeni[p][m];
```

a tedy každá třída může mít vždy nejvýše jeden předmět v danou dobu

```
forall (t in rTridy)
       noOverlap(rozvrhTridy[t]);
```

Výuka předmětu musí probíhat bez přerušení

Školní rozvrh: model II.

Každý předmět je vyučován pro konkrétní třídu (skupinu žáků),

tj. pro každou třídu je zadána množina jejích předmětů

```
// sekvence rozvrhu tříd z důvodu nepřekrývání
dvar sequence rozvrhTridy[t in rTridy] in
  all (m in rMistnosti, p in PredmetyTrid : p.trida == t) prirazeni[p][m];
```

a tedy každá třída může mít vždy nejvýše jeden předmět v danou dobu

```
forall (t in rTridy)
       noOverlap(rozvrhTridy[t]);
```

Výuka předmětu musí probíhat bez přerušení

```
forall (p in PredmetyTrid)
       startOf(casy[p]) % Hodin <= (Hodin-p.trvani);
```

Školní rozvrh: optimalizace

Jednotlivé předměty by měly být do rozvrhu umístěny tak, aby výuka všech tříd skončila co nejdříve (např. třetí den dopoledne). Tj. **minimalizujeme součet koncových časů výuky posledních předmětů všech tříd**.

Školní rozvrh: optimalizace

Jednotlivé předměty by měly být do rozvrhu umístěny tak, aby výuka všech tříd skončila co nejdříve (např. třetí den dopoledne). Tj. **minimalizujeme součet koncových časů výuky posledních předmětů všech tříd**.

```
// čas konce posledního předmětu každé třídy
dexpr int maxTridy[t in rTridy] =
      max(p in PredmetyTrid : p.trida == t) endOf(casy[p]);

// součet koncových časů posledních hodin všech tříd
minimize sum(t in rTridy) maxTridy[t];
```

Rozvrhování vyšetření pacientů v nemocnici

Každý pacient musí absolvovat vysetření zadaného typu, která probíhají na různých pracovištích. Cílem řešení je pro každé vyšetření pacienta určit, ve *kterém čase* bude probíhat a *které pracoviště* bude vyšetření realizovat. Dále:

- Vyšetření jednoho pacienta se nesmí překrývat.
- Vyšetření na jednom pracovišti se nesmí překrývat.
- Každý typ vyšetření má stanovenu délku provádění a pracoviště, na kterých může být prováděno.
- Některý typ vyšetření musí proběhnout před realizací jiného typu vyšetření (pokud je pacient absolvuje oba). Jsou proto zadány dvojice typů vyšetření určující, které vyšetření musí být realizováno před zahájením dalšího vyšetření.

Vyšetření pacientů se závažnějším typem onemocnění by měla být dokončena dříve, aby mohli dříve nastoupit na operaci. Cílem je tedy minimalizovat vážený součet dokončení nejpozdějších vyšetření pacientů.

Jednoduché zadání

Příklad jednoduchého zadání s řešením:

- počet pacientů 2, počet typů vyšetření 2, počet pracovišť 2;
- doby trvání vyšetření daného typu: 1, 2;
- pracoviště pro vyšetření daného typu: 1:1,2, 2:1;
- typy vyšetření pro pacienty: 1:1,2, 2:1;
- priority pacientů: 1,10;
- precedence pro dané typy vyšetření: 1 před 2;
- dvě optimální řešení s kvalitou 13:
  - pacient, vyšetření, čas, pracoviště
  - 1,1,0,1; 1,2,1,2; 2,1,0,2
  - 1,1,0,2; 1,2,1,2; 2,1,0,1.

Vstupní data a typy

Konstanty a intervaly

```
int NbPacients = ...;
int NbMedicals = ...;
int NbRooms = ...;

range RPacients = 1..NbPacients;
range RMedicals = 1..NbMedicals;
range RRooms = 1..NbRooms;
```

Vyšetření: typ, místnosti

Vstupní data a typy

Konstanty a intervaly

```
int NbPacients = ...;
int NbMedicals = ...;
int NbRooms = ...;

range RPacients = 1..NbPacients;
range RMedicals = 1..NbMedicals;
range RRooms = 1..NbRooms;
```

Vyšetření: typ, místnosti

```
tuple Tmedical{
 int duration; {int} rooms; }

Tmedical Medicals[RMedicals] = ...;      // [<2,{1,3}>, <2,{2,4}>, <3,{1,2,3}>]
```

Pacient: typy vyšetření

Vstupní data a typy

Konstanty a intervaly

```
int NbPacients = ...;
int NbMedicals = ...;
int NbRooms = ...;

range RPacients = 1..NbPacients;
range RMedicals = 1..NbMedicals;
range RRooms = 1..NbRooms;
```

Vyšetření: typ, místnosti

```
tuple Tmedical{
 int duration; {int} rooms; }

Tmedical Medicals[RMedicals] = ...;      // [<2,{1,3}>, <2,{2,4}>, <3,{1,2,3}>]
```

Pacient: typy vyšetření

```
tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;              // {<1,1>,<1,2>,<2,2>,<2,3>,<3,1>, ...}
```

Vyšetření pacienta

```
tuple Tmedical{
 int duration; {int} rooms; }

Tmedical Medicals[RMedicals] = ...;

tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;
```

Časy vyšetření pacienta

Vyšetření pacienta

```
tuple Tmedical{
 int duration; {int} rooms; }

Tmedical Medicals[RMedicals] = ...;

tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;
```

Časy vyšetření pacienta

```
dvar interval times[pm in PMedicals] size Medicals[pm.medical].duration;
```

Nepřekrytí vyšetření pacienta

Vyšetření pacienta

```
tuple Tmedical{
 int duration; {int} rooms; }

Tmedical Medicals[RMedicals] = ...;

tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;
```

Časy vyšetření pacienta

```
dvar interval times[pm in PMedicals] size Medicals[pm.medical].duration;
```

Nepřekrytí vyšetření pacienta

```
dvar sequence seqPacient[p in RPacients] in
                  all(pm in PMedicals: pm.pacient == p) times[pm];
```

Vyšetření pacienta

```
tuple Tmedical{
 int duration; {int} rooms; }

Tmedical Medicals[RMedicals] = ...;

tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;
```

Časy vyšetření pacienta

```
dvar interval times[pm in PMedicals] size Medicals[pm.medical].duration;
```

Nepřekrytí vyšetření pacienta

```
dvar sequence seqPacient[p in RPacients] in
                  all(pm in PMedicals: pm.pacient == p) times[pm];

forall (p in RPacients) noOverlap(seqPacient[p]);
```

Vyšetření na jednom pracovišti

```
dvar interval times[pm in PMedicals] size
Medicals[pm.medical].duration;
```

Proměnné pro výběr místnosti pro vyšetření pacienta

Vyšetření na jednom pracovišti

```
dvar interval times[pm in PMedicals] size
Medicals[pm.medical].duration;
```

Proměnné pro výběr místnosti pro vyšetření pacienta

```
dvar interval assigned[PMedicals][RRooms] optional;
```

Na každém pracovišti maximálně jedno vyšetření

Vyšetření na jednom pracovišti

```
dvar interval times[pm in PMedicals] size
Medicals[pm.medical].duration;
```

Proměnné pro výběr místnosti pro vyšetření pacienta

```
dvar interval assigned[PMedicals][RRooms] optional;
```

Na každém pracovišti maximálně jedno vyšetření

```
forall (pm in PMedicals)
        alternative (times[pm],
                     all (r in RRooms: r in Medicals[pm.medical].rooms)
                          assigned[pm][r]);
```

Vyšetření na jednom pracovišti

```
dvar interval times[pm in PMedicals] size
Medicals[pm.medical].duration;
```

Proměnné pro výběr místnosti pro vyšetření pacienta

```
dvar interval assigned[PMedicals][RRooms] optional;
```

Na každém pracovišti maximálně jedno vyšetření

```
forall (pm in PMedicals)
        alternative (times[pm],
                     all (r in RRooms: r in Medicals[pm.medical].rooms)
                          assigned[pm][r]);

forall (r in RRooms)
        noOverlap(all (pm in PMedicals: r in Medicals[pm.medical].rooms)
                  assigned[pm][r]);
```

Precedence

```
tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;

dvar interval times[pm in PMedicals] size Medicals[pm.medical].duration;
```

Typy a vstupní data

Precedence

```
tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;

dvar interval times[pm in PMedicals] size Medicals[pm.medical].duration;
```

Typy a vstupní data

```
tuple Tprec{
 int before;
 int after; }
```

Precedence

```
tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;

dvar interval times[pm in PMedicals] size Medicals[pm.medical].duration;
```

Typy a vstupní data

```
tuple Tprec{
 int before;
 int after; }

{Tprec} Prec = ...;
```

Omezení na precedence

Precedence

```
tuple pMedical{
 int pacient; int medical; }

{pMedical} PMedicals = ...;

dvar interval times[pm in PMedicals] size Medicals[pm.medical].duration;
```

Typy a vstupní data

```
tuple Tprec{
 int before;
 int after; }

{Tprec} Prec = ...;
```

Omezení na precedence

```
forall (pr in Prec, pmB in PMedicals, pmA in PMedicals:
            pmB.pacient == pmA.pacient &&
            pmB.medical == pr.before &&
            pmA.medical == pr.after)
          endBeforeStart(times[pmB],times[pmA]);
```

Minimalizace vážených časů posledních vyšetření

Typy a vstupní data

Minimalizace vážených časů posledních vyšetření

Typy a vstupní data

```
int Priorities[RPacients] = ...;
```

Minimalizace

Minimalizace vážených časů posledních vyšetření

Typy a vstupní data

```
int Priorities[RPacients] = ...;
```

Minimalizace

```
dexpr int pacientMax[p in RPacients]
              = max (pm in PMedicals: pm.pacient == p) endOf(times[pm]);
```

Minimalizace vážených časů posledních vyšetření

Typy a vstupní data

```
int Priorities[RPacients] = ...;
```

Minimalizace

```
dexpr int pacientMax[p in RPacients]
              = max (pm in PMedicals: pm.pacient == p) endOf(times[pm]);

minimize sum (p in RPacients) Priorities[p]*pacientMax[p];
```