Discrete Optimization

WEEK1

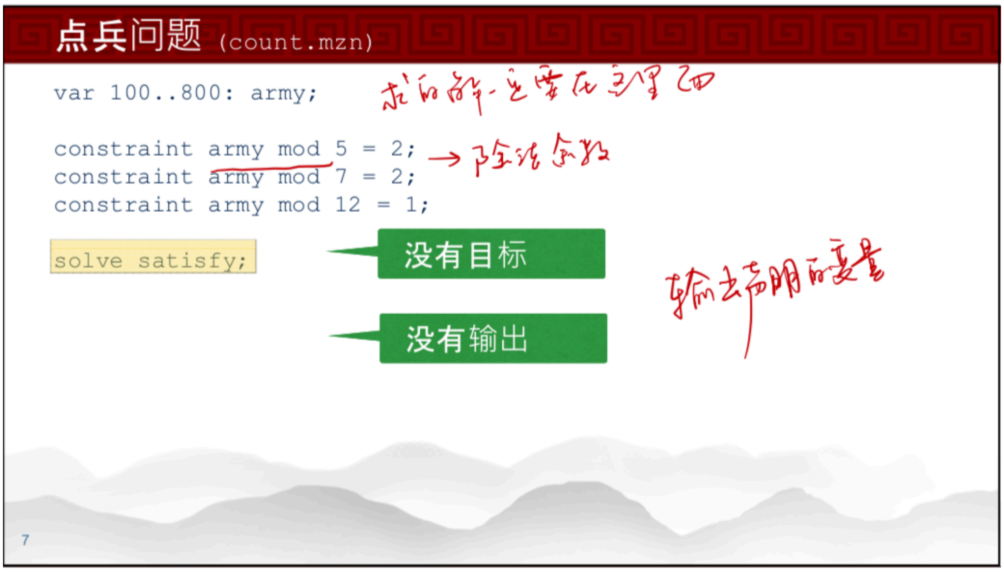
Mimizinc 基本介绍

```
%参数
int: budget;

%决策变量
var 0..1000: F;
var 0..400: L;
var 0..500: Z;
var 0..150: J;

%约束
constraint 13*F + 21*L + 17*Z + 100*J <= budget;

%目标
solve maximize 6*F + 10*L + 8*Z + 40*J;

%输出
output ["F = \(F), L = \(L), Z = \(Z), J = \(J)\n"]

```

1. 两种变量
    - 参数(与标准编程语言中的变量相似。它们必须被赋值)
      • int: i=3;
      • par int: i=3;
      • int: i; i=3;
    - 决策变量 (用var与一个类型 (或者一个范围/集合)来声明)
      •var int: i; constraint i >= 0; constraint i <= 4;
      •var 0..4: i;
      •var {0,1,2,3,4}: i;
2. 约束
    - constraint <约束表达式>
    - 基于标准的算术关系符来创建

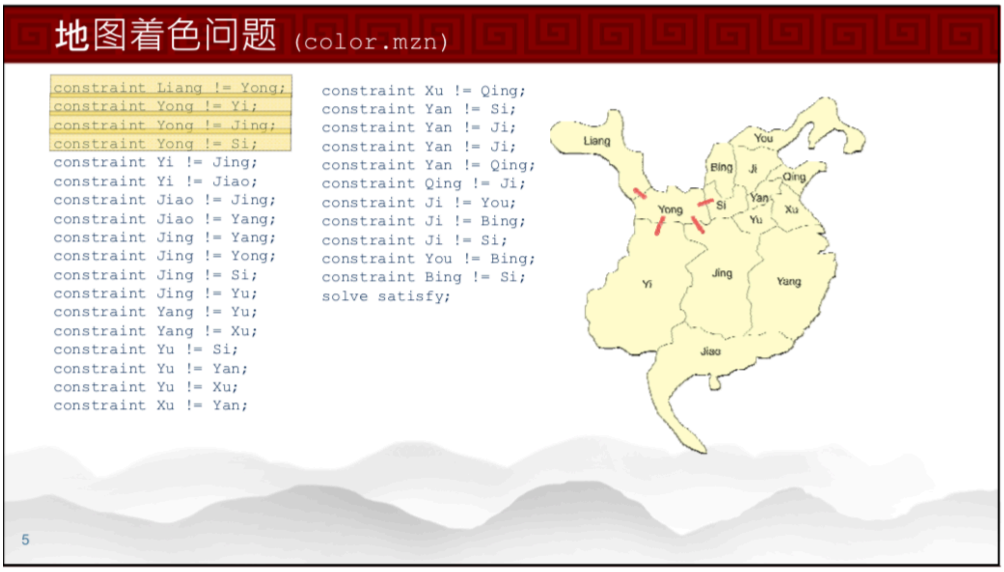
      > = != > < >= <=
3. 输出与字符串

- output <字符串列表>
- show(v) 以字符串形式输出v的值 (v) 在字符串常量中显示v
- "house" ++ "boat"用于连接字符串



4. 默认情况下，MiniZinc输出
   - 所有声明的变量
   - 且没有被表达书复制

> 适合用于不需要求得最优解的满足问题



5. 枚举类型定义一个具有有限对象的集合
    - 决策变量和参数可以是枚举类型
    - 数组下标可以是枚举类型
    - 集合可以基于枚举类型

对象建模

```
enum DISH;
int: capacity;
array[DISH] of int: satisf;
array[DISH] of int: size;
array[DISH] of var int: amt; % how many of each dish
constraint forall(i in DISH)(amt[i] >= 0);
constraint sum(i in DISH)(size[i] * amt[i])
   <= capacity;
solve maximize sum(i in DISH)(satisf[i] * amt[i]);
output ["Amount = ", show(amt), "\n"];
```

1. 下标范围表达式
    - l..u(l,u为整数)
    - 枚举类型
2. 参数和变量数组
    - array[范围] of 变量声明
3. 数组查找
    - 数组名[下标表达式]
4. 生成器表达式
    - forall(i in 范围)(bool 型表达式)
      对于范围内所有的i，对应布尔表达式都为真
    - sum(i in 范围)(表达式)
      对范围内所有i 对应的表达式累加

数组和推倒式

```
enum PRODUCT;
array[PRODUCT] of float: profit;
enum RESOURCE;
array[RESOURCE] of float: capacity;
array[PRODUCT,RESOURCE] of float: consumption;
array[PRODUCT] of var int: produce;


constraint forall(p in PRODUCT)(produce[p] >= 0);
constraint forall(r in RESOURCE)(sum (p in PRODUCT)(consumption[p, r] * produce[p]) <= capacity[r]);

solve maximize sum(p in PRODUCT)
   (profit[p]*produce[p]);

output ["\(p): \(produce[p])\n" | p in PRODUCT];


```

1. 一个数组可以是多维的，可如下声明为

   > array[下标集合1,下标集合2, ...]of类型

2. 数组的下标集合必须是
   - 一个整型范围或者枚举类型
   - 或者是固定值的集合表达式，而它的值则是一个范围
3. 数组的元素可以是任何类型，但不可以是另外 一个数组，例如，

   > array[PRODUCT,RESOURCE] of int: consumption;

4. 内建函数length返回一维数组的长度
5. 数组推导式有以下形式
   - [表达式 | 生成器1, 生成器2, ...]
   - [表达式 | 生成器1, 生成器2, ... where 测试]
6. example

   ```
     [i + j | i, j in 1..4 where i < j]
   = [1+2, 1+3, 1+4, 2+3, 2+4, 3+4]
   = [3, 4, 5, 5, 6, 7]

   forall(i,j in 1..10 where i < j)
     (a[i] != a[j])

   forall([a[i] != a[j]| i,j in 1..10 where i < j])
   ```

全局约束

```
include "alldifferent.mzn"
alldifferent(variables)
```

WEEK2

集合的选择

1. 0-1基本模型

```
enum MOVES;
int: timeBound;
array[MOVES] of int: power;
array[MOVES] of int: duration;

array[MOVES] of var int: occur;

constraint forall(i in MOVES)(occur[i] >= 0);
constraint forall(i in MOVES)(occur[i] <= 1);
constraint (sum(i in MOVES)(duration[i] *occur[i])) <= timeBound;

solve maximize sum(i in MOVES)(power[i] *occur[i]);
```

2. 0-1 布尔模型

```
enum MOVES;
int: timeBound;
array[MOVES] of int: power;
array[MOVES] of int: duration;

array[MOVES] of var bool: occur;

constraint (sum(i in MOVES)(duration[i] * bool2int(occur[i]))) <= timeBound;

solve maximize sum(i in MOVES)(power[i] * bool2int(occur[i]));

```

3. 0-1集合模型

```
enum MOVES;
int: timeBound;
array[MOVES] of int: power;
array[MOVES] of int: duration;

var set of MOVES: occur;

constraint (sum(i in occur)(duration[i]))<= timeBound;
solve maximize sum(i in occur)(power[i]);
```

- in(集合中的元素 例如: x in s)
- subset, superset(子集，超集)
- intersect(交集)
- union(并集)
- card(集合势)
- diff(差运算，例如:x diff y = x \ y)
- symdiff(对称差)

> 例如: {1, 2, 5, 6} symdiff {2, 3, 4, 5} = {1, 3, 4, 6}

固定势集合的选择

1. 有两种方式去表达固定势集合
    - var set of OBJ + 势约束
      适用情况:求解器本身支持集合,**OBJ**不是太大
    - array[1..u] of var OBJ
      适用情况:当u比较小

```
var set of SPOT: attacks;
card(attacks) = size


array[1..size] of var SPOT: attacks;
%some constraint
forall(i in 1..u-1)(x[i] < x[i+1]);

```

有界势集合的选择

```
int: nSpots;
set of int: SPOT = 1..nSpots;
array[SPOT] of int: damage;
enum SYMB;
array[SYMB] of set of SPOT: group;
int: size;

set of int: SPOTx = {0} union SPOT;
array[1..size] of var SPOTx: attacks;
constraint forall(i in 1..size-1)(attacks[i] >= (attacks[i] != 0) + attacks[i+1]);
constraint forall(s in SYMB)(sum(i in 1..size)(attacks[i] in group[s]) <= 1);

var int: totalDamages =sum(p in attacks)(damage[p]);
solve maximize (totalDamages);

```

```
var set of SPOT: attacks;
card(attacks) <= size
```

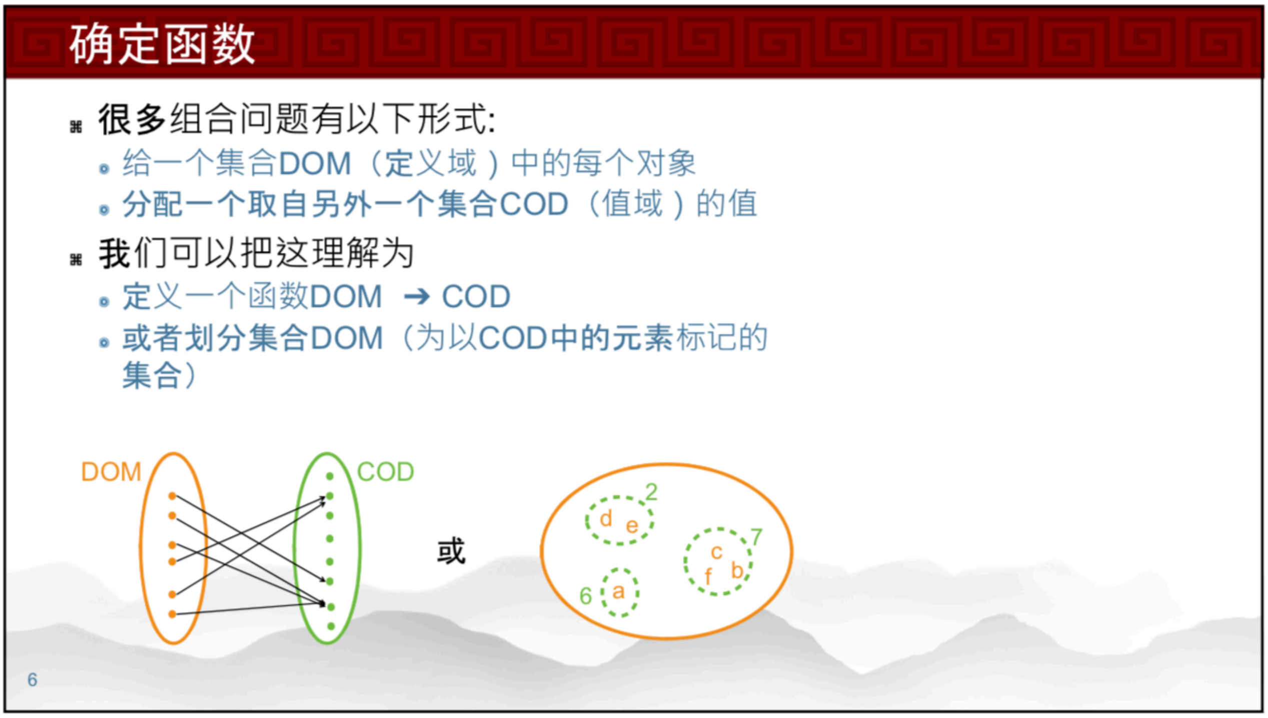
有多种方式去表示集合

1. var set of OBJ
    - 适用情况:求解器本身支持集合 • 适用情况:OBJ不是太大
2. array[OBJ] of var bool / 0..1
    - 适用情况:OBJ不是太大
3. array[1..u] of var OBJ
    - 只用于固定势u
    - 适用情况:当u比较小
4. array[1..u] of var OBJx
    - 需要表示“无”这个元素

WEEK3

函数建模

1. 确定函数



2. 这个函数可以为:
    - 单射:分配问题
    - 双射(|DOM|=|COD|):匹配问题

全局势约束

1. 我们有特殊的约束来限定划分类的大小

```
global_cardinality(x, v, c)
```

    - 约束 $c_i = \Sigma_{j \in 1..n}(x_j = v_i)$
    - global_cardinality(x,[1,2],[2,1]); x = [1,1,2,3] ✅, [1,2,3,4] ❌
2. 收集出现次数，要求每个值都出现

```
global_cardinality_closed(x, v, c)
```

3. 限定出现次数的上限和下限

```
global_cardinality_low_up_closed(x,v,l,u)
```

纯划分

1. MiniZinc包含了一个用于去值对称的全局约束

```
value_precede_chain(array[int] of int: c,array[int] of var int: x)
```

   - 强制c[i]在x中的第一次出现先于c[i+1]在x中 的第一次出现
   - value_precede_chain([1,2,3], x) x = [1,1,2,3] ✅, [1,3,1,2] ❌,[1,2,1,2] ✅

WEEK4

多重建模

1. 视角
   在以下情况下，函数 f: DOM ➔ COD 是特殊的
   - |DOM| = |COD|
   - 函数 f 是双射的
2. 一个双射函数有两个视角

```
array[DOM] of var COD: f;
array[COD] of var DOM: finv;
```

3. 连通约束

> 利用include "globals.mzn"; inverse(x1, x2); 如果做得合适，基于CP的求解器可以从模型 结合中获益，提高求解效率

```
include "globals.mzn";
enum FOOD;
enum WINE;
array[FOOD, WINE] of int: joy;
array[FOOD] of var WINE: drink;
array[WINE] of var FOOD: eat;
constraint inverse(eat, drink);
solve maximize sum(f in FOOD)(joy[f, drink[f]]);
% solve maximize sum(w in WINE)(joy[eat[w], w]);
```

4. 在我们的例子中，一些需求无法在某个特定的视角下表示，这时就只能利用结合模型来阐 述整个问题

```
enum PIVOT;
PIVOT: first;
set of int: POS = 1..card(PIVOT);
array[PIVOT] of int: coord; % coord of pivot
int: m; % number of precedences
set of int: PREC = 1..m;
array[PREC] of PIVOT: left;
array[PREC] of PIVOT: right;

array[PIVOT] of var POS: order;
array[POS] of var PIVOT: route;

route[1] = first;
inverse(order,route);
forall(i in PREC)
```

```
   (order[left[i]] < order[right[i]]);

solve minimize sum(i in 1..card(PIVOT)-1)
                (abs(coord[route[i]] - coord[route[i+1]]));
```