我们需要在组织索引之前确定某种组织方式。设想使用**计算某种值**来达到货位的筛选效果，这样做会比较高效。

目前设想到的值有：

1. **出入库点的裸时间花费**，也即根据提升机的三速、母车三速、子车三速，地图的三距完成计算，此步完全不考虑地图的布置情况，是在地图的property设置完成后的初始化计算选项。出入库时间可达蔓延，类似走迷宫算法。**分别计算所有的出/入库点。最后相加。**
   1. 此处注意，按照传统的出/入库点的时间计算并形成隔离类别，是考虑了将来的平均出入库时间花费。通常对于单个堆垛机货架而言，只有一个出入口，此时计算时间花费和是有意义的。但是如果存在多个的话，出入库可以选择就近或者并行，而并不需要依照某个出入库点对来进行计算。
   2. 综上考虑，假设每个出入库点的使用率为平均，那么就是对于当前货位而言，出入库裸时间花费为：

（1）

上述公式中代表了地图中，坐标为(x,y,z)的货位的**出入库值**，代表了在(x,y,z)处的第n个入库点到达该坐标的时间花费；TimeOutput亦然；代表入库点的总个数；代表出库点的总个数。

* 1. 考虑是否把出入库点的使用次数计入数据库，以方便对出入库值进行动态调整。如果考虑，则公式更变为：

（2）

（3）

上述公式中对每个出入库点的计算加上了权值，权值代表第n个入库点的权值；出库点亦然，代表为j。对于权值的计算见式3，其中代表了第i个出/入库点，代表了该出入库点的使用次数。此式对于出/入库点的权值计算通用，因此将公式合并为一种表达形式，但是出/入点的使用次数综合是分开的。**这样做可能会导致最终地图偏向于某个特定的出入库对。有没有什么可以有效阻止这种情况的方法？**暂时先使用公式1的做法，但是保留公式2、3的计算公式，以便扩展。

1. **倒货指数**，（**具体如何计算尚未确定**），唯一可以确定的是，该指数的计算只出现在**每次出入库计算之后**。可以理解为当前货道的混乱指数，（**对单行货道进行计算？**）设想每混放一种货物，倒货指数就上升，暂定为：

上述公式中，Complex代表货道的混乱程度，代表当前货道的货物种类数量。

1. **自定义优先级指数**，该指数暂且保留为一个默认值，方便后续的自定义权重扩展。可以理解成开设多个map对应的索引结构存储这些值，用户可以通过开关来决定是否把这些指数纳入计算。例如用户指定货物存储的货道编号；用户开启可混放等等。
2. **出/入库值**，最终的组织依据，（**如何计算**）根据上述的几个指数最终柔和成的一个依据指数，通常依照出/入库值的最大（或者最小）为优先选取度最高。理论上来讲这个值是动态计算出来的，也就是**每次执行出/入库指令时进行计算**。
   1. 假设最终的出/入库值按照越小越容易被选取达到，来计算（依照第一点裸时间花费）。那么在最终计算的时候，自定义优先级也是越小越高，倒货指数需要根据某种映射条件，再乘上一个放大阈值（可能跟其他数值不在一个数量级，因此需要放大），在乘上一个权重。具体做法是把所有上述的参与计算的指数都**映射到同一个值域内**，由于第一点裸时间花费是固定不变的，仅限于栅格地图的物理更改会改变，因此**采用裸时间花费的值域范围**。

(4)

上述公式放大偏移的公式，需要在计算完所有的指数之后进行统计计算。其中Index是目标指数，Original为原指数，Threshold为放大阈值，Offset为偏移量。其中又有：

(5)

(6)

上述公式(5)中，T代表第一点裸时间花费的所有数据集合，O代表目标参数的原数据集合。Mean函数代表均值，于是Offset就代表了罗时间花费的均值到0的偏移量。于是就有Index代表了从原值域映射到裸时间花费的值域。**上述公式4、5、6对任何指数的映射都有效**。

* 1. 那么假设最终的出/入库值的计算为上述以及各种自定义指数的和，则有：

(7)

上述公式公式中，代表公式4中的结果，为第i个指数；为第i个指数的权重；为第i个指数的开关。考虑出入库具有相同权重的计算，也就做到了类似货道先进先出的队列，货位先进后出的栈。**此外出库也考虑增加一项时间，也即入库时间越长，那么出库概率越大，暂时先不增加此项，因为假设所有的货物入库以后，都会出库**。

* 1. 对于例如倒货指数的逆向映射：

(8)

上述公式中，代表了整个地图的最长货道的长度。

**我们可以考虑把索引分成两个部分，一个是货道索引，一个是货位索引。根据需求，过道索引可以扩展-为货道编号。直到一个货物到达货道，再请求获取货位。这样可以保证货物出入库在先后顺序上保证不会出现死锁。**

关于地图的初始化信息扫描组织：为了简化计算模型，我们对栅格地图作出以下几点约束假设：

1. 单提升机垂直贯通，从底向上没有中断。
2. 货道方向为一个方向，并且默认为沿着列方向（视觉横向）。于是单条货道如果可达，则至少有一个、最多有两个点邻接轨道。
3. 用户设置货位轨道（Storage-Rail），并且只有该种货位可以邻接轨道（特指子车的贯通）。
4. 出入库点只存在于第一层，并且都至少连接一个电梯
5. 轨道+货位轨道不存在回环。
6. 一根连续的货道只允许使用一个区域设置。
7. 周围一圈空白栅格，用于设置抽象的特殊点，包括出入库点和提升机。

~~于是从所有出入库点和提升机开始，有：~~

1. **~~出入库只能邻接轨道和提升机；使用母车速度计算~~**
2. **~~轨道只能邻接出入库点、货位、货位轨道和提升机，并且纵向只能邻接轨道，横向可以邻接货位、轨道和货位轨道，提升机无限制；使用母车速度计算~~**
3. **~~货位只能邻接货位；使用子车速度计算~~**
4. **~~货位轨道只能邻接货位轨道和轨道；使用子车速度计算~~**
5. **~~提升机只能邻接轨道和出入库点；使用母车速度计算~~**
6. **~~垂直方向只有提升机能邻接提升机；使用提升机速度计算~~**

设计某种算法，能够正确识别货道、并且拥有出口的扫描算法。（？）随再次扫描，根据货道号组织index映射。

否决上述细粒度计算方式，重新考虑大粒度的计算方式，以货道为一个单位计算，也即：计算从一层出入库点经过轨道，二层向上从提升机位置经过轨道，各个轨道栅格的时间均值。**记录可以在货道方向上存在货位的轨道栅格，方便后续的扫描并组织货道。**于是邻接情况变成了：

1. **~~出入库点只能邻接轨道和提升机；使用母车速度计算~~**
2. **~~轨道只能邻接出入库点、货位轨道、轨道和提升机；对于入库点、轨道和提升机使用母车速度计算，对于货位轨道使用子车速度计算~~**
3. **~~货位轨道只能邻接轨道；使用子车速度计算~~**
4. **~~提升机只能邻接轨道和出入库点；使用母车速度计算~~**

**否决上述邻接情况，更改的地方有：把出入库点和提升机另外当作特殊栅格点来对待，并且增加了特殊栅格连接表，用来表示特殊栅格之间的时间花费。于是邻接情况变成了：**

1. **出入库点只能邻接一个轨道或者一个提升机，使用设置的特殊连接时间花费。**
2. **提升机只能邻接一个出入库、一个轨道或者提升机，使用设置的特殊连接时间花费；对于提升机使用提升机速度。**
3. **轨道邻接一个提升机、一个出入库点或者轨道，对于特殊栅格使用特殊连接时间花费；对于轨道使用母车速度。**
4. **\*轨道还能邻接货道，使用子车速度。**

**扫描算法**：记录一个Dictionary，记录出库点的一张三维表和一张出库点的三维表。**注意的是：每张表分别记录的是从出入库点开始的均值，但是从2层往上就没有出入库点了，只有提升机；于是2层往上就从提升机位置开始，对于2层往上重新开一张表，这张表也使用均值作为每个货位的index值。**组织为Dictionary<int, double[,]>，使用0和-1代表出、入库的均值index值，使用1以上代表对应层的index值，从提升机位置开始扫描，在这一层循环中，当前层全部栅格扫描完成后，需要进行合并，也即判断每一个提升机的位置的其下一层（z-1）是否也是提升机。于是对当前层的所有元素加上这些提升机代表时间的均值。**但是有一个问题**，也就是如果存在交错的不同坐标不同长的提升机，选择从那一层换提升机会出现一定量的时间差，影响到后续的Class分配。

只要队列不为空，每次取出队列中的一个，向4个方向延伸，分别是前后左右。第二层往上开始，从提升机位置开始扫描，方法如上，只是要在最开始计算该提升机点到第一层的时间。根据上述红粗字的约束来跳转，并且记录第一个遇到的方向上的的状态，并向这个方向上一直延伸到出现第二种状态为止（例如轨道-货道-轨道）。其中要根据传入的**货道方向**来决定“轨道-货道”的关系跳转。如果已经计算过的位置则跳过不进行重复计算，如果是新计算的值则把当前位置压入队列。重复以上过程。

**TODO::此处应插入一个状态转移图**。

然后会出现多个出入库点对应的裸时间值。其中可能会出现“轨道不能达，但是通过货道可达”，以及“通过出入库点都不可达”的情况。此时对于但凡出现一种出-入库对值存在0（也即其中一个不可达）的情况，那么这个位置就是不可达。对于其他情况，我们取同出入库点的均值，或者权重值（例如同样都是入库点的多个裸时间花费，对应多个入库点）。这样做的原因在于：

1. 如果出入库点只有一个（一个入库点或者一个出库点），那么对应的栅格肯定是距离出入库点越近，时间花费就越少。此时取均值就意味着单个出入库点的时间花费。
2. 如果存在多个出库点或者入库点，那么取均值就代表了多个点的等概率访问。例如，两个贴角的、沿列方向对称的入库点对应的值的均值，就都是相等的。
3. 对于可达货位，但是无法通过轨道直接到达的货道，需要根据上述均值来决定选取一条货道作为空闲的“可穿越轨道”。优先选取裸时间花费均值最小的，如若出现多个相同最小值，那么选**均值乘以差值**最小的。

**TODO::此处应插入公式语言**

关于“可穿越轨道”是否要选取，选取哪一条的算法：（？？）

初步设计总映射表映射货道，为一个数组或者链表形式；每个货道再映射货位。货道和货位的映射可以是相同的结构，此时货道组织的上述各个指数为拥有货位的指数的均值。

**另外数据库有必要记录所有的出/入库情况，约束所有的同品类货物具有相同的品规和批次。记录数据有助于帮助得知仓库在某个时间段内，某个货物的周转率。**

**🡪周转率如何计算？**

# 可用确定的时间花费计算公式

**关于地图四周增加一圈障碍物，可以只考虑在计算时自动加上，而非当作实际地图的一个展示、或者隐藏部分存入数据库。**

关于已至加速度、减速度，总路程S，最高速度，求行驶时间:

(1)

(2)

上式1、2分别是未到达最高速度时的总时间花费(1)，和到达最高速度的总时间花费(2)，临界值S为：

(3)

关于迭代过程中的均值计算：

# 货道双端筛选——返回入库点Id策略

该算法目前存在于LogicsService类中TakeCertainMapItemsId函数，参数有三个：入库点MapItem，货道CargoWay以及货道出口1或2。其中1和2分别表示使用的是CargoWay的Bottom/ Top/ InputDict/ OutputDict/ AvailabeCount/ PotentialEnergy这几个外部表现属性的使用是左端还是右端，1表示左端，2表示右端。

策略流程：

1. 货道为空
   1. 入库点MapItem为空
      1. 双头通
         1. 双头势能相等 🡪 根据交叉法则（单双选择双头中的一头），选取其中一头中简单花费最小的入库点MapItem的Id并返回；若此时传入的货道出口不是应当选择的那头，返回0.
         2. 双头势能不等 🡪 选取高势的一头中简单花费最小的入库点MapItem的Id并返回；若此时传入的货道出口不是高势的一头，返回0.
      2. 单头通 🡪 选取可通的那头中简单花费最小的入库点MapItem的Id并返回；若当前传入的货道出口不是通的一头，返回0。
   2. 入库点MapItem不为空
      1. 双头都对该入库点MapItem可通
         1. 双头对应入库点MapItem的简单花费相等 🡪 根据交叉法则选择其中一头，返回该入库点MapItem的Id；若此时传入的货道出口不是选择的一头，返回0.
         2. 双头对应入库点MapItem的简单花费不相等 🡪 选择简单花费小的一头，返回入库点MapItem的Id；若此时传入的货道出口不是简单花费小的一头，返回0.
      2. 非双头可通 🡪 若当前货道出口对应的入库点MapItem可达，则返回该入库点MapItem的Id，否则返回0。
2. 货道不为空
   1. 入库点MapItem为空 🡪 根据传入的货道出口，选择对应入库点简单花费最小的入库点MapItem并返回其Id，不通返回0。
   2. 入库点MapItem不为空 🡪 根据传入的货道出口，若入库点MapItem可达，则返回其Id；不可达返回0。

# 背包问题映射以及动态规划算法的魔改

关于0-1背包问题、多重背包问题和完全背包问题的说明和解法可以参照：

<https://www.cnblogs.com/yuxiaoba/p/8460105.html>

简述：

1. 0-1背包问题：物品向量X，物品价值向量V，物品重量（体积）向量W，并且这三项的总大小为d；背包总容量Y。求一个向量集R={0, 1}d ，s.t. , 。
2. 多重背包问题：物品向量X，物品价值向量V，物品重量（体积）向量W，物品数量向量K，并且这四项的总大小为d；背包总容量Y。求一个向量集R=[0, ]d ，s.t. , 。该问题可以通过拆解数量()，拆分成多个()，J表示拆解后的表示个数，并且满足对于任意的组成的向量Q={0, 1}J，。拆分规则可以是：拆成1，2，4……，例如=5，则可以拆分成1，2，4，那么就可以组合成：1，2，1+2，4，1+4。并把多重背包问题映射成0-1背包问题。
3. 完全背包问题：物品向量X，物品价值向量V，物品重量（体积）向量W，并且这三项的总大小为d，每件物品可以无限取；背包总容量Y。求一个向量集R={N}d ，s.t. , ，N为自然数。

若使用DFS来解决背包问题，时间复杂度会随着搜索深度的增加而成指数增加，假设最大深度为l，共n件物品，则时间复杂度为：。

使用动态规划的解法，对于0-1背包问题的通项公式为：

(1)

上式中，i代表考虑放入第i件物品，j表示背包容量为j，因此, ；f[i][j]表示第i件物品放入容量为j的最优情况。通常j > 。

对于完全背包问题，通项公式为：

(2)

对于上式来说，i表示考虑放入k件第i件物品，j表示背包容量为j，因此, , ，N表示自然数。实际上公式(1)也可以也可以和使用公式(2)代替，只不过k的取值仅为0和1，扩展开即为公式(1)。

对于多重背包问题，可以使用分解法归约到0-1背包问题，使用公式(1)。也可以考虑使用公式(2)更为泛化，但是约束有变：

(3)

上式中物品不再是无限，而是有限个数个。

**关于本课题中的问题：**已知X个候选货道，也记为向量X，需要放入Y托货物，货道放入k托货物时的得分为：，其中status表示计算时的各种状态，通项公式中不作介绍。要求在能够放入Y托货物的情况下，求出得分最小，也即最优解。

该问题可以规约为背包问题：

1. X个候选货道可以映射为X个货物
2. 要放入Y托货物可以映射为背包容量为Y
3. 每个货道的货位体积为1，容积为，可以映射为每个货物的体积和最多可选数量
4. X个候选货道的解为向量Q，s.t. ，可以映射为背包最大容量约束

于是该问题可以映射为多重背包问题，并且满足约束4。由于该问题的“背包容量”不再是“不超过”，而是“必须等于”。因此需要对通项公式(3)进行更改：

(4)

(5)

上式中i代表第i根货道，j代表扫描到第i根货道时，全局放入j托货物时的最优解。k代表选择k托货物入库到第i根货道，t[i][j]代表扫描到第i根货道时，全局放入j托货物时，共放入了多少托货物。使用这一限制是因为：货道可用空位数量可能不足以容纳j托货物，而这一状态需要延续到下一层迭代中计算，那么当下一次迭代到全局需放入j托货物时，已经放入了t[i][j]托，从而计算出还需要放入s托才能全部放入需要的品托数，那么在下一层迭代中，当迭代到j时，直接从s托开始计算，原因在于，不论分数高低，考虑优先放入所有j托货物时，扫描到第i根货道时的最优解。如果选择k托大于了第i根货道的可容纳数量，那么就延续第i根货道的前一状态j-1。

上述针对本问题的动态规划通项公式的基础上，增加几个用于记录状态的数组即可，那么当计算完毕、且t[X][Y]=Y时，返回所有状态数组的最后一项就是最优解。该算法的复杂度为：。算法流程如下：

