要求实现的基本问题：

1.机器人类型：人形、轮形和机器狗形

(1)对于每个机器人来说，表示在边上的移动速度，表示在楼梯上的移动速度

(2)对于每个机器人来说，有代表该机器人能完成的工作类型

2.Tasks

(1)包含当前坐标={ord,stair},stair代表在第几层，ord代表在stair层数上标号为ord节点的位置

(2)包含skill，只考虑有一个skill类型，即不需要多个机器人组合完成该项目（后期优化内容）

(3)包含内容，即从初始坐标前往目标坐标{ord2,stair2}

(4) 包含时间time，考虑最原始的情况，即任务一个一个处理。

3. 电梯、楼梯坐标{ord,stair}

(1)电梯和楼梯：电梯的移动速度非匀速，楼梯的移动速度是匀速的，且不同类型的机器人的移动速度恒定并且不变。

(2)对于电梯，不同类型机器人的移动速度是固定的，但是电梯不是匀速的，意味着从1到3楼的边长小于1到2楼边长的两倍，将3维图转化为二维图时候需要注意。

4.难点：如何处理冲突问题：

(1)边是一个双向车道，每个车道仅能有一个机器人意味着不同时间段内边会有占用的情况

(2)电梯：电梯内容量有限,仅能容下一个机器人,会有电梯运行冲突。

5.实现方式：

(1)目前解决方法：贪心+最短路

一、系统总体框架（模块结构）

1. 输入模块

机器人信息输入：type, skill, ord, floor, v1, v2, available  
任务信息输入：skill\_req, start\_ord, start\_floor, target\_ord, target\_floor, time\_limit  
环境信息输入：lift, stair, edges

2. 建图模块（Graph Construction）

将楼层和节点映射为二维图节点 (ord, floor)  
建立边集：同层边、楼梯边、电梯边  
电梯非匀速，容量有限，需设定 lift\_time[f1][f2] 表

3. 冲突检测模块（Conflict Manager）

EdgeTag：记录每条边的占用状态  
LiftTag：记录电梯占用状态  
函数：is\_available(edge, current\_time, travel\_time)

4. 路径规划模块（Routing Algorithm）

核心算法：带时间占用约束的最短路径  
基础算法：Dijkstra + 时间过滤

5. 任务调度模块（Task Assignment）

任务按时间顺序贪心处理  
机器人匹配条件：技能匹配 + 距离最短  
更新机器人状态与可用时间

6. 状态更新模块

更新 Tag[edge], LiftTag, Robot 状态  
维护当前时间戳和路径占用信息

初始化地图信息

Initialize Graph G(ord, floor)  
Initialize EdgeTag[], LiftTag[]  
Initialize Current\_time = 0  
  
for task in Tasks:  
 feasible\_robots = [r for r in Robots if r.skill == task.skill and r.available]  
 初始化路径  
 best\_robot = None  
 best\_cost = INF  
 best\_route = None  
 寻找合适路径  
 for r in feasible\_robots:  
 route, cost = FindShortestPath(G, r.position, task.start, current\_time, EdgeTag, LiftTag)  
 if cost < best\_cost:  
 best\_cost = cost  
 best\_robot = r  
 best\_route = route  
 找到路径后更新信息  
 if best\_robot:  
 route\_to\_target, cost\_target = FindShortestPath(G, task.start, task.target, current\_time, EdgeTag, LiftTag)  
 total\_time = best\_cost + cost\_target  
 UpdateEdgeTag(best\_route + route\_to\_target, total\_time)  
 UpdateRobot(best\_robot, task.target, total\_time)  
 LogSolution(task, best\_robot, route\_to\_target)  
 else:  
 LogFailure(task)

暂时的最短路径计算方法：

FindShortestPath （temp）  
def FindShortestPath(G, start, end, current\_time, EdgeTag, LiftTag):  
 dist = defaultdict(lambda: INF)  
 pq = PriorityQueue()  
 dist[start] = 0  
 pq.push((0, start))  
 prev = {}  
  
 while pq not empty:  
 t, u = pq.pop()  
 if u == end:  
 break  
 for edge in G[u]:  
 v = edge.target  
 travel\_time = edge.length / robot\_speed(edge.type)  
 if edge.type == 'lift':  
 if not is\_lift\_available(LiftTag, current\_time + t, travel\_time):  
 continue  
 else:  
 if not is\_edge\_available(EdgeTag, edge, current\_time + t, travel\_time):  
 continue  
 if dist[v] > dist[u] + travel\_time:  
 dist[v] = dist[u] + travel\_time  
 prev[v] = u  
 pq.push((dist[v], v))  
  
 return reconstruct\_path(prev, start, end), dist[end]