

高三上信息选考算法综合

1. 某快递配送站负责本区域快递配送，配送站有用于配送快递的无人机 A 和 B，以及快递员 R。当快递到站后，重量超过 20kg 的快递放置在 HA 货架上，其他放置在 HB 货架上。HA 货架上的快递由无人机 A 配送，HB 货架上的快递由无人机 B 配送。若有新的快递到站，HA 或者 HB 货架上放满 c 件快递时，该快递放置在 HR 货架上，HR 货架上的快递由快递员 R 配送。无人机 A 或者 B 根据快递到站的时间顺序依次配送，快递员 R 根据快递重量由大到小进行配送。

一天上午 08:00 开始前 8 个快递的信息如图 a 所示，当 c 的值为 2 时，各快递的配送信息如图 b 所示。

快递编号	到站时间	快递重量	配送时间
1	8:17	25	50
2	8:18	26	32
3	8:19	28	38
4	8:20	8	40
5	8:21	9	30
6	8:22	11	40
7	8:23	10	50
8	8:24	29	40

图 a

快递编号	到站时间	出站时间	滞留时间
1	8:17	8:17	0
2	8:18	9:07	49
3	8:19	9:39	80
4	8:20	8:20	0
5	8:21	9:00	39
6	8:22	9:30	68
7	8:23	8:23	0
8	8:24	9:13	49

图 b

编程求解：给出某一天的快递编号、到站时间、快递重量、配送时间（分钟），以及货架 HA 和 HB 的容量 c，计算各快递的出站时间及每个快递在配送站的滞留时间。

(1) 由题意可知，为第 33 题图 a 中编号为 3 的快递配送的是_____（单选，填字母：A. 无人机 A /B. 无人机 B /C. 快递员 R）。

(2) 定义如下 insertxp(data,lst,k) 函数，参数 data 的每个元素由快递编号、到站时间、快递重量和配送时间四项构成，参数 lst 是由 data 中部分元素索引构成的列表，这部分元素已经按对应的快递重量降序排列。函数功能是将索引 k（k 为 data 列表元素的索引）插入列表 lst 中，并使 lst 中索引对应的 data 元素依然按快递重量降序排列。

```
def insertxp(data,lst,k):
    lst.append(k)
    i = len(lst) - 2
    while i >= 0 and data[lst[i]][2] < data[k][2]:
        lst[i+1] = lst[i]
        i -= 1
    lst[i+1] = k
```

若 data 为[[1,'08:25',39,20], [2,'09:37',7,1], [3,'10:05',45,33], [4,'10:17',23,16], [5,'10:33',24,18], [6,'10:37',36,21]]。请结合回答①和②两题。

①调用 insertxp(data, lst, k) 函数，若 lst 为 [2,0,3], k 为 4，则 while 语句中循环体的执行次数是_____。

②若将函数中 while 语句的条件 “i>= 0 and data[lst[i]][2] < data[k][2]” 误写为 “i > 0 and data[lst[i]][2] < data[k][2]”，则会导致某些情况下无法得到符合函数功能的结果。调用 insertxp(data, lst, k) 函数，下列 4 组数据中能测试出这一问题的是_____（单选，填字母）。

- A.lst 为 [2,0], k 为 3
- B.lst 为 [4,3], k 为 5
- C.lst 为 [2,3], k 为 4
- D.lst 为 [2,4], k 为 5

(3)实现计算各快递的出站时间及每个快递在配送站的滞留时间的部分 Python 程序如下，请在画线处填入合适的代码。

按重量降序插入快递索引的函数

```
def insertxp(data, lst, k):
```

```
    lst.append(k)          # 先把索引 k 加入列表末尾
    i = len(lst) - 2      # 从倒数第二个元素开始向前比较
    # 循环条件: 索引有效 且 当前元素对应重量 < 待插入元素重量
    while i >= 0 and data[lst[i]][2] < data[k][2]:
        lst[i+1] = lst[i] # 元素后移, 为插入 k 腾出位置
        i -= 1           # 向前移动索引
    lst[i+1] = k          # 将 k 插入正确位置
```

核心配送调度函数: 计算出站时间和滞留时间

```
def assign(exp):
```

```
    curr = [0]*3          # curr[0]=A 配送结束时间, curr[1]=B, curr[2]=R(初始 0 对应 08:00 分钟)
    q = []                 # q[0]=HA 货架, q[1]=HB 货架, q[2]=HR 货架 (存储快递索引)
    qout = [-1]*3         # 记录 A/B/R 正在配送的快递索引 (-1 表示无)
    w = [-1]*len(exp)     # 存储每个快递的滞留时间 (初始-1)
    for i in range(3):    # 初始化 3 个货架为空列表
        q.append([])
```

遍历每个快递 (i 是当前处理的快递索引)

```
for i in range(len(exp)):
```

```
    # 先处理出站: 依次检查 A、B、R 的配送状态 (j=0=A, j=1=B, j=2=R)
```

```
    for j in range(3):
```

```
        ① k = qout[j] # 取出配送方 j 正在配送的快递索引 k
```

```
        # 循环条件: 货架 j 有快递 且 (无正在配送的快递 或 正在配送的快递已完成)
```

```
        while len(q[j]) > 0 and (k == -1 or curr[j]+exp[k][3] <= exp[i][1]):
```

```
            qout[j] = q[j].pop(0) # 取出货架 j 第一个快递, 标记为“正在配送”
```

```
            if k == -1:           # 首次配送 (无正在配送的快递)
```

```
                ② curr[j] = exp[qout[j]][1] # 配送结束时间=该快递到站时间
```

```
                w[qout[j]] = 0           # 无滞留, 滞留时间为 0
```

```
            else:                 # 非首次配送 (有正在配送的快递)
```

```
                # 若当前配送完成时间 ≤ 新快递到站时间 → 新快递到站即配送
```

```
                if curr[j]+exp[k][3] <= exp[qout[j]][1]:
```

```
                    w[qout[j]] = 0
```

```
                    curr[j] = exp[qout[j]][1]
```

```
                else:             # 新快递需要等待, 计算滞留时间
```

```
                    curr[j] = curr[j]+exp[k][3] # 更新配送结束时间
```

```
                    ③ w[qout[j]] = curr[j] - exp[qout[j]][1] # 滞留=结束-到站
```

```
# 进站操作: 按重量和货架容量分配快递
```

```
if exp[i][2] > 20 and len(q[0]) < c: # 重量>20kg 且 HA 货架未滿
```

```
    q[0].append(i)
```

```
elif exp[i][2] <= 20 and len(q[1]) < c: # 重量≤20kg 且 HB 货架未滿
```

```
    q[1].append(i)
```

```

else: # 货架满, 放入 HR 货架 (按重量降序插入)
    insertxp(exp, q[2], i)

# 无人机 A、B, 快递员 R 分别配送货架上的快递, 代码略
# 输出每个快递的配送信息, 如图 b 所示, 代码略
'''
读取当天快递的数据, 按到站时间依次存入列表 data 中。列表的每个元素包含 4 个数据项,
分别对应快递编号、到站时间、快递重量和配送时间, 如 data[0]为[1,'08:25',39,20], 代码略
将 data 中各元素的到站时间格式转换为分钟表示形式, 代码略
读取货架 HA 和 HB 的容量, 存入 c, 代码略
'''
assign(data)

```

2.在人工智能与高性能计算融合的场景中,需将若干计算任务分配到 n 个计算节点上。每个计算节点的算力上限为 m (单位: TFLOPS, 万亿次浮点运算每秒), 每个计算任务的算力需求不超过 m , 且所有任务的总算力需求恰好为 $n \times m$, 任务不可拆分, 为最大化资源利用率, 需要让每个节点满负荷运行, 需确定各节点的任务分配方案。

例如, 有 8 个计算任务, 各任务的应用场景及算力需求如图 a 所示, 现需将任务分配到 4 个计算节点, 每个节点的算力上限为 1200TFLOPS。计算节点的任务分配方案可能有多种, 其中一种可行方案如图 b 所示:

任务编号	任务类型	算力需求 (TFLOPS)
1	AI 大模型训练	1200
2	全球气候模拟	800
3	自动驾驶城市路况仿真	700
4	基因组批量测序分析	400
5	金融高频交易策略回测	200
6	电影特效复杂场景渲染	600
7	大语言模型预训练	500
8	卫星遥感图像 AI 解译	400

图 a

计算节点	分配的任务编号
节点 1	1
节点 2	2、4
节点 3	3、7
节点 4	5、6、8

图 b

请编写 python 程序, 实现计算节点任务分配方案。

(1) 将上述 8 个计算任务分配到 3 个计算节点, 每个计算节点的算力上限为 1600 TFLOPS。计算节点分配的任务编号情况为 `[[1,4], _____]`。

(2) 定义如下 `sorttasks(data)` 函数, 函数 `data` 的每个元素由“任务编号”、“任务类型”、“算力需求” 3 个数据项组成。函数的功能是根据各任务算力需求降序排序。请在划线处填入合适代码。

```

def sorttasks(data):
    task = [[] for i in range(len(data))]
    x = 0

```

```

for i in range(len(data)):
    t = data[i][2]
    j = x - 1
    while _____:
        task[j + 1] = task[j]
        j -= 1
    task[j + 1] = [data[i][0], t]
    x += 1
return task

```

(3) 实现节点算力分配功能的部分 python 程序如下, 程序中用到的列表方法函数如下表所示, 请在程序中划线处填入合适的代码。

```

def findflops():
    global num    # num 为全局变量
    a = [0] * (m + 1)
    lnk = [-1] * (m + 1)
    for i in range(cnt):
        if not flag[i]:
            t = tasks[i][1]
            for j in range(m, t - 1, -1):
                if a[j] < a[j - t] + t:
                    a[j] = a[j - t] + t
                    lnk[j] = i
    if a[m] == m:
        p = m
        while _____①_____:
            j = lnk[p]
            lst[num].append(tasks[j][0])
            q.append([num, j])
            flag[j] = True
            p -= tasks[j][1]
            num = (num + 1) % n
        return True
    return False

```

'''

读取节点总数存入变量 n, 单个计算节点的算力上限 (TFLOPS) 存入变量 m, 读取计算任务存入列表 data 中, 代码略

'''

```

tasks = sorttasks(data)
cnt = len(tasks)
num = 0
flag = [False] * cnt
lst=[]
for i in range(n):
    lst.append([])

```

```
q=[] # 记录节点分配的任务编号情况
while len(q) != cnt:
    ②
    if not found and q != []:
        tmp = q.pop(0)
        lst[tmp[0]] = []
    ③
# 输出各计算节点分配的任务编号，代码略
```