研究生算法课课堂笔记

上课日期： 2016-11-28 第(1)节课

组长学号及姓名：1601214525 许智敏

组员学号及姓名：1601214507 戴统宇

内容概要:

本节课主要讲授的是继续上节课所讲的贪心算法中的活动选择和任务调度问题，由上节课结束时的活动安排最少教室问题引入。具体如下:

1. 区间调度问题
2. 区间调度问题贪心法最优解
3. 任务调度最少延迟问题
4. 任务调度最少延迟问题贪心法最优解
5. 扩展着色问题
6. 扩展堆与二叉树

详细内容:

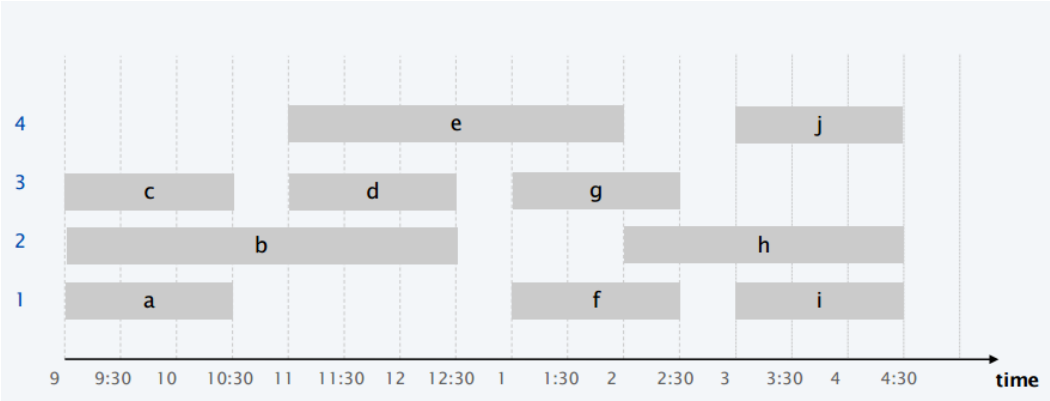
1. 区间调度（Interval Scheduling）问题

(1) 问题描述

假设课程活动 j 开始于 s_j 结束于 f_j，目标是寻找最小的教室数目，以保证安排下所有的课程活动，同时保证没有任何两个活动在同一个教室于同一时间段产生冲突。

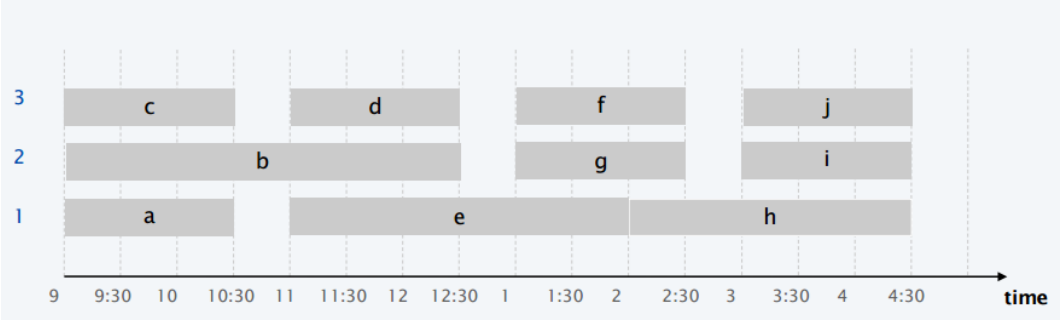
(2) 解决方案示例

用 4 间教室安排 10 个课程活动 a, b, c, d, e, f, g, h, i, j:



由上图调度可以发现，该调度策略满足安排下了所有的课程活动，同时没有任何两个活动在相同的教室于同一时间段产生冲突。

下面用 3 间教室给出如上 10 个课程活动的最优解:



对于之前的十个活动安排进行重新安排，发现了更优的方案。该调度策略满足安排下了所有的课程活动，同时没有任何两个活动在相同的教室于同一时间段产生冲突。

(3) 求解思路（贪心法）

贪心总体思路为，考虑将课程活动以某种顺序来进行排序，然后安排一个课程活动到一个可以安排的教室（具体哪一个教室需要考虑？），如果没有教室可安排，那么就申请一个新的教室。可以有以下四种方法:

A. 优先安排最早开始的课程活动（可行解——最优解）

按照课程活动的开始时间 s_j 来升序排序，通过最早开始时间 s_j 来用贪心

法进行安排：

最早开始的活动优先安排算法（earliest-start-time-first algorithm）：

EARLIEST-START-TIME-FIRST $(n, s_1, s_2, \ldots, s_n, f_1, f_2, \ldots, f_n)$

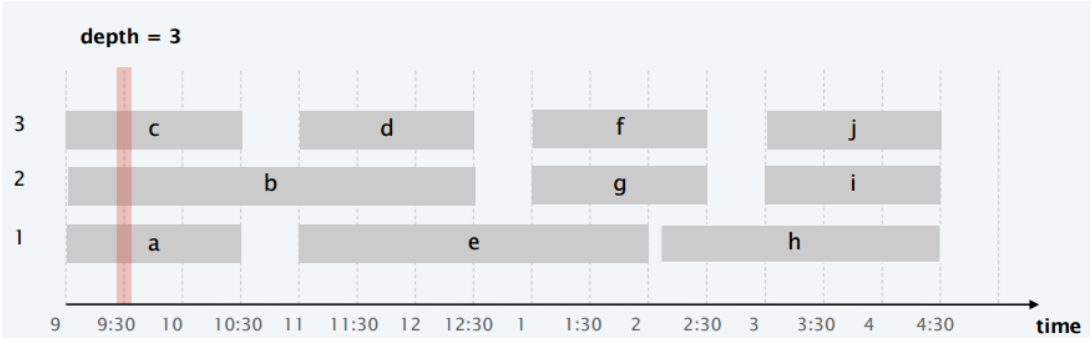
SORT lectures by start time so that $s_1 \leq s_2 \leq \ldots \leq s_n$.
$d \leftarrow 0$ ⟵ number of allocated classrooms
FOR $j = 1$ TO n
  IF lecture j is compatible with some classroom
    Schedule lecture j in any such classroom k.
  ELSE
    Allocate a new classroom $d + 1$.
    Schedule lecture j in classroom $d + 1$.
    $d \leftarrow d + 1$
RETURN schedule.

需要注意的是，在该算法中，选择的教室 k 是教室中的最后一个课程活动结束时间最早的一个。

该算法的时间复杂度为 O（n log n），**证明：**因为教室存储在一个优先队列里面（key = 该教室中最后一个活动的结束时间），为了决策课程活动 j 和哪些教室是相互兼容的，需要比较 s_j 和在优先队列里面具有最小 key 值的教室 k，如果兼容的话，那么把课程活动 j 加入到教室 k 中（把教室 k 的 key 值增加到 f_j）。**该算法保证不会将有冲突的两个课程活动安排在同一间教室，该算法的结果是最优的，证明（最优解的下界）：**

定义：一系列活动的**深度**定义为在任何时刻，可以同时进行的活动的最大数目。

注意：最少需要的教室数目一定大于等于**深度。**



1. 初始化d为算法申请的教室的数量

2. 申请教室d的前提是要安排的活动j与所有其他d-1个教室已存在的活动有冲突

3. 这样，这d个教室的已有课程的结束时间都晚于s_j

4. 由于我们按照活动开始时间排序，所有冲突产生的原因在于这些活动的开始时间不晚于s_j

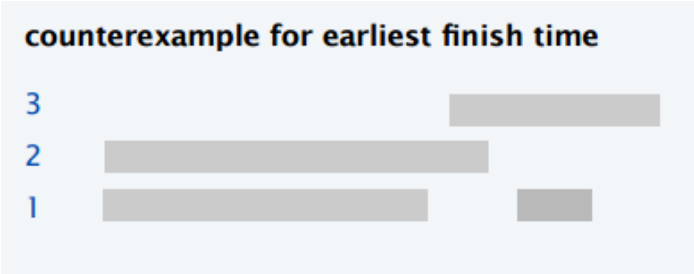
5. 所以在时间$s_j+\varepsilon$的时刻（ε很小），一共有d个活动产生冲突

6. 所以该算法产生的教室数目最少为d个，即该贪心法结果最优

B. 优先安排最早结束的活动（非可行解）

按照课程活动的结束时间f_j来升序排序，通过最早结束时间f_j来用贪心法进行安排：

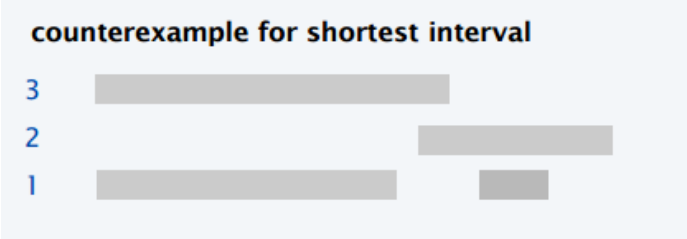
存在反例如下：



C. 优先安排最短持续时间的活动（非可行解）

按照课程活动的持续时间f_j-s_j来升序排，通过最短持续时间f_j-s_j用贪心法进行安排：

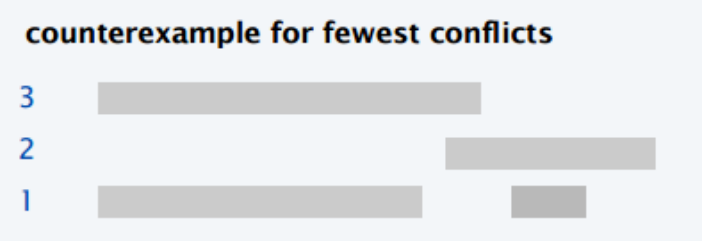
存在反例如下：



D. 优先安排冲突最少的活动（非可行解）

按照课程活动开始的时间s_j来排序，用贪心法进行安排：

存在反例如下：



(4) 思考

A. **对于安排活动的最小教室数量的问题，**可以转换成图的最小着色问题，每个顶点对应为一个活动，其边连接着不兼容的活动。为使任两个相邻结点的颜色均不相同，所需的最少颜色对应于找出调度给定的所有活动所需的最少教室数。

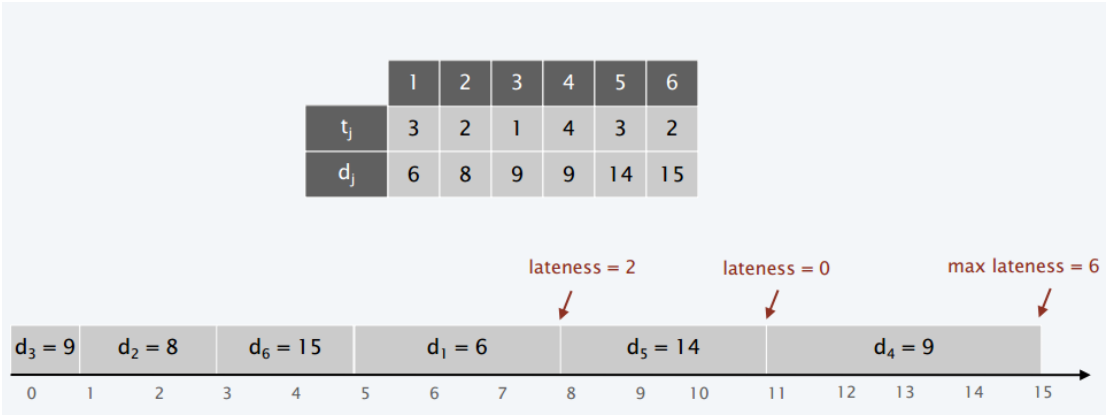
B. 在按照结束时间排序的问题中，是否能够保证只与 last 的课程活动冲突?

这个问题我们可以假设存在这样的一个任务，虽然结束的很晚，但是开始的很早，这就会导致其与之前的很多课程活动都有冲突。

2. 任务调度最小延迟（scheduling to minimize lateness）

（1）问题描述

用一个进程资源在同一时刻处理一个任务。任务j需要t_j的处理时间，而且其截止日期是d_j，如果j开始于s_j，那么它结束于$f_j = s_j + t_j$。延迟定义为$\ell_j = \max\{0, f_j - d_j\}$，目标就是调度所有的任务，使得最大延迟$L = \max_j \ell_j$最小。



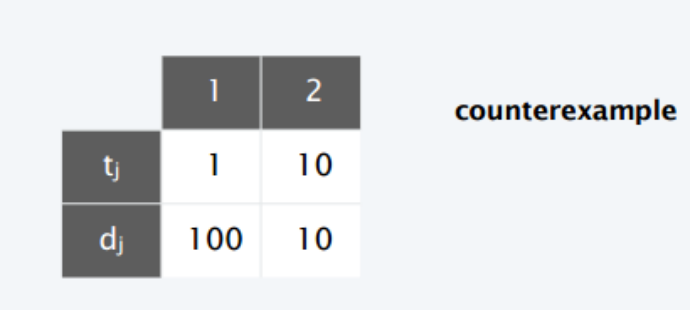
（2）求解思路（贪心法）

贪心总体思路为，考虑将所有任务以某种顺序来进行排序，然后安排一系列任务。有以下三种方法：

A. 优先安排最短持续时间的任务（非可行解）

按照任务的持续时间t_j来升序排序，通过最短持续时间t_j来用贪心法进行安排：

存在反例如下：



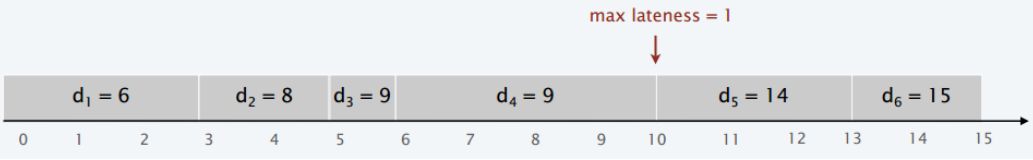
B. 优先安排最早 deadline 的任务（可行解——最优解）

按照任务的最早截止时间 d_j 来升序排序，通过最早截止时间 d_j 来用贪心法进行安排：

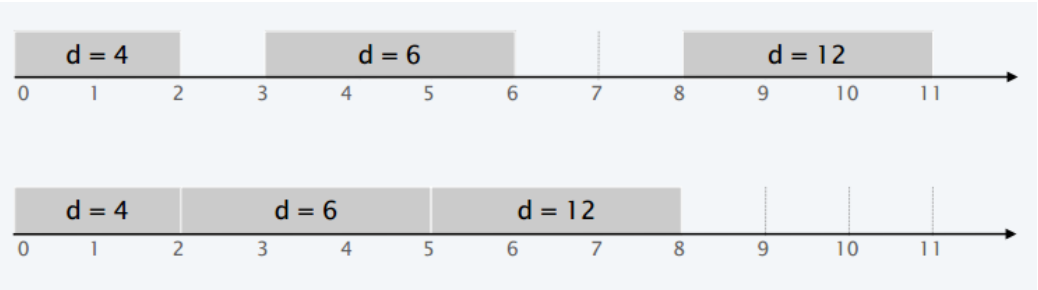
最早截止时间算法（earliest deadline first）：

EARLIEST-DEADLINE-FIRST $(n, t_1, t_2, \ldots, t_n, d_1, d_2, \ldots, d_n)$

SORT n jobs so that $d_1 \le d_2 \le \ldots \le d_n$.
$t \leftarrow 0$
FOR $j = 1$ TO n
  Assign job j to interval $[t, t + t_j]$.
  $s_j \leftarrow t$; $f_j \leftarrow t + t_j$
  $t \leftarrow t + t_j$
RETURN intervals $[s_1, f_1]$, $[s_2, f_2]$, ..., $[s_n, f_n]$.

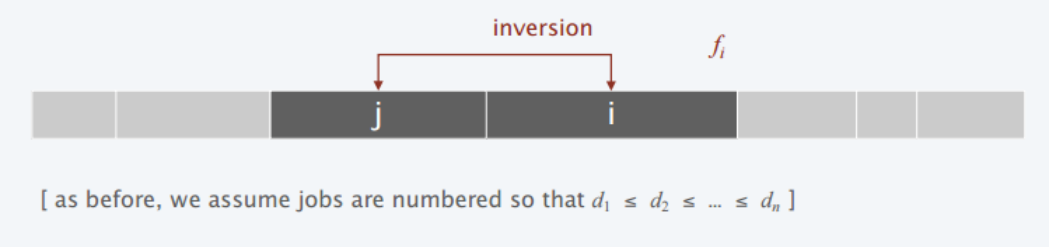


首先存在一个最优调度，这个最优调度是没有空闲时间的。最早 deadline 算法避免了空闲时间。

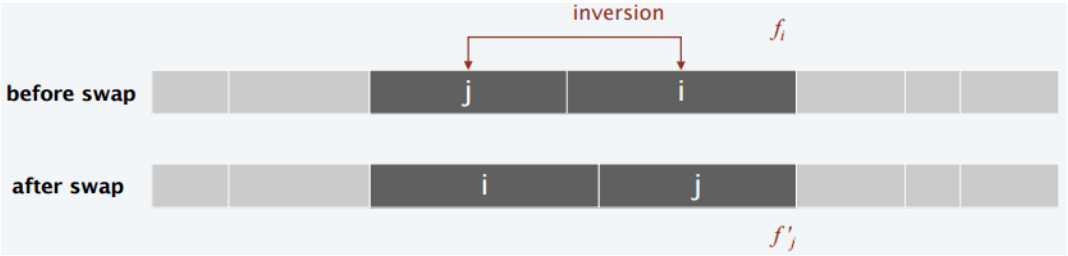


定义：假设一个调度 S，一个工作 i 和 j 的倒置如下图所示：

i < j，但是 j 在 i 前调度。



最早 deadline 算法不存在这样的逆序情况。如果一个没有闲置时间的调度存在倒置，那么存在一对倒置的连续执行的任务。



下面证明：交换两个邻接的倒置任务，不会增加最大延迟。

证明一：让ℓ作为交换前的延迟，ℓ'作为交换后的延迟，

对于所有的$k \neq i, j$，$\ell'_k = \ell_k$，$\ell'_i \le \ell_i$

由上图和定义可知，如果任务j截止比较晚，$\ell'_j = f'_j - d_j = f_i - d_j \le f_i - d_i \le \ell_i$

最早 deadline 算法产生的调度S是最优解。

证明二：定义S^*作为有最少倒置数的最优解，

我们可以假设S^*没有闲置时间，如果S^*没有倒置，则$S = S^*$，如果S^*有倒置，那么让$i-j$做一个邻接交换，由上一个证明可知，不会增大最大延迟，同时减少了倒置的数目，这与S^*的定义矛盾，所以，该算法产生的调度即为最优解。

C. 优先安排最小 slack 的任务（非可行解）

按照任务的最小 slack $d_j - t_j$ 来升序排序，通过最小 slack $d_j - t_j$ 来用贪心法

进行安排：

存在反例如下：

	1	2
t_j	1	10
d_j	2	10

counterexample

3. 拓展内容

（1）堆和二叉树的区别

内存方面：

二叉树的指针要占用内存，浪费空间，堆就不会。

二叉树的内存不连续，堆可以用数组实现，内存连续。

结构方面：

☆二叉树是严格有序的，左右孩子与父节点存在严格的大小关系。

堆只保证顶点比子女大（或小），不能保证左右孩子之间的大小关系。

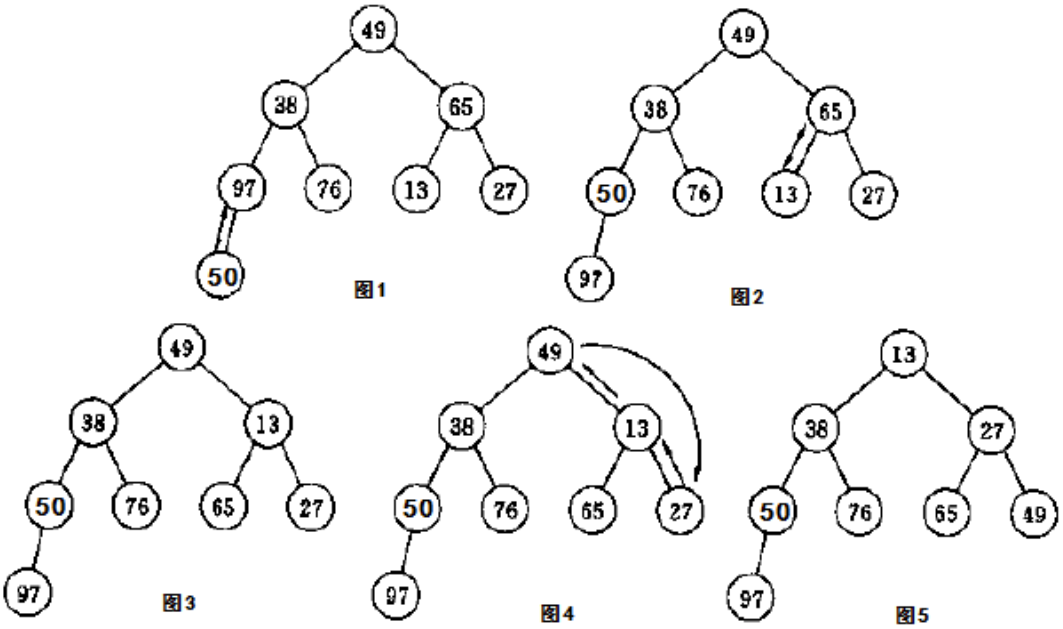
因此，查找某个 key 值的数据时，堆需要O（n），二叉树O（log$n$）

（2）堆的定义

堆分为大顶堆和小顶堆，满足 Key[i]>=Key[2i+1]&&key>=key[2i+2]称为大顶堆，满足 Key[i]<=key[2i+1]&&Key[i]<=key[2i+2]称为小顶堆（数组下标从 0 开始）。也就是说任何一非叶节点的关键字不大于或者不小于其左右孩子节点的关键字。

（3）堆的一些操作分析

A. 堆的初始化（小顶堆）：



从一个无序序列初始化为一个堆的过程就是一个反复“筛选”的过程。由完全二叉树的性质可以知，一个有 n 个节点的完全二叉树的最后一个非叶节点是节点[n/2]，堆的初始化过程就从这个[n/2]节点开始。上图为如下无序数组的初始化：

{49,38,65,97,76,13,27,50}

首先，未处理的数组对应的堆为图 1 模样。从第四个节点开始（[8/2]=4），因为 50 < 97，故要交换两节点，**交换后还要继续对其新的左子树进行比较**。易见其左子树只有节点 97，已经为最佳情况，故可以继续堆的初始化，如图 2。再考虑第三个节点，因为 13 < 27 < 65，即节点 13 为当前的最小节点，故与节点 65 交换，并**对新的左子树进行比较**，其也为最佳情况，故可继续堆的初始化，结果如图 3。然后考虑第二个节点，因为 38 < 50 < 76，故已经为最优情况，不用调整。最后再考虑第一个节点，根节点。因为 13 < 38 < 49，故需要将根节点 49 与其右孩子节点 13 交换，**交换后还要继续对其新的右子树进行比较**，可见右子树还需要调整，因为 27 < 49 < 65,故将节点 49 与节点 27 交换。此时已经处理完了根节点，初始化结束。最终结果如图 5.

(参考：http://www.i3geek.com/archives/682)

B. 堆的插入：

堆的插入的思想就是先在最后的结点添加一个元素，然后沿着树上升，跟堆的初始化大致相同。

C. 堆的排序：

利用大顶堆(小顶堆)堆顶记录的是最大关键字(最小关键字)这一特性，使得每次从无序中选择最大记录(最小记录)变得简单。

其**基本思想**为(小顶堆)：

1)将待排序关键字序列(R1,R2....Rn)初始化；

2)将堆顶元素 R[1]与最后一个元素 R[n]交换，此时得到新的无序区(R1,R2,.....Rn-1)和新的有序区(Rn),且满足 R[1,2...n-1]>=R[n];

3)由于交换后新的堆顶 R[1]可能违反堆的性质，因此需要对当前无序区(R1,R2,.....Rn-1)调整为新堆，然后再次将 R[1]与无序区最后一个元素交换，得到新的无序区(R1,R2....Rn-2)和新的有序区(Rn-1,Rn)。不断重复此过程直到有序区的元素个数为 n-1，则整个排序过程完成。序列是降序排序。

(参考：http://blog.csdn.net/xiaoxiaoxuewen/article/details/7570621/)

由此可见：堆的各种操作均基于堆的初始化进行。