研究生算法课课堂笔记

上课日期：2016 年 11 月 28 日　　第 2 节课

组员学号及姓名：

周昊宇 / 1601214498　张艺 / 1601214747　兰兆千/ 1601214438

本节课内容如下：

- 回顾了教室分配问题，并提出了另一种算法讨论其最优性
- 最小化任务延迟问题
- 总结了贪心法的证明思路
- 介绍了贪心法在优化缓存上的应用
- 图论中的贪心——最短路径算法

上节内容回顾：区间分割问题（教室分配问题）

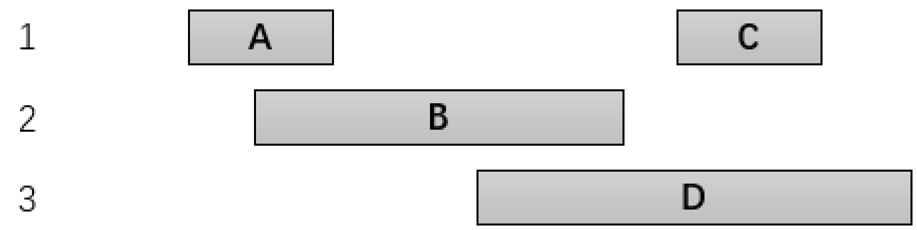
问题简述

已知课程 j 开始时间s_j和结束时间f_j，求安排下所有课程所需的最小教室数，使得没有两节课同一时间在同一间教室（即无课程冲突）。

按结束时间排序选择课程，优先安排空隙最小的教室

根据结束时间f_j排序（从小到大），然后依次安排课程。如果有很多个教室可以安排时，选择结束时间最晚的教室（空隙最小）安排。当所有教室都冲突时，新开一间教室安排。

如果不考虑空隙问题，该贪心法非最优性的反例



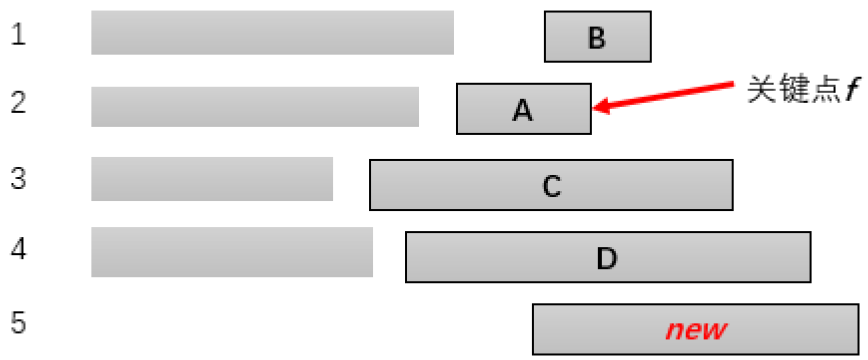
如果仅考虑按照结束时间排序，只要不冲突就随机安排教室，那么上图就是一个反例。因为课程 C 导致大量空闲时间被浪费，安排课程 D 时只能另开一间教室。

显然对于这个例子，只需要 AD 和 BC 两间教室即可。使用上面的算法不会导致这个问题，使用上节课讲的 earliest-start-time-first 算法也不会导致这个问题。

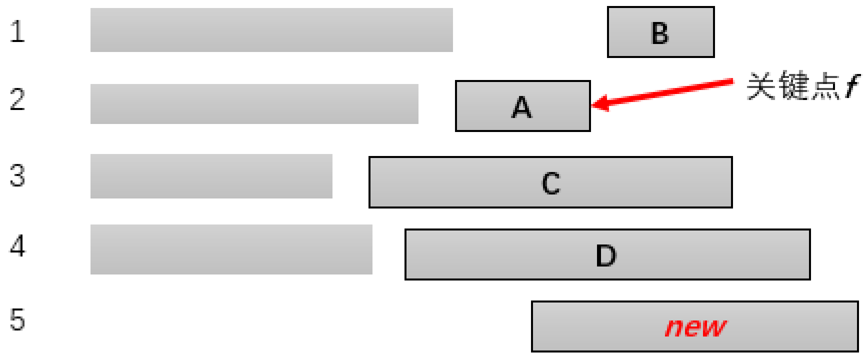
最优性证明

与上节课例子的证明方法类似，我们需要找到一个同一时刻的最大冲突数 depth，我们所需的教室数显然≥depth。我们只需证明贪心法可以达到这个下界 depth 即可。

- 对于开辟了最后一个教室的活动 j，不能放在原有的 d-1 个教室里，只能放在新分配的第 d 个教室，那么活动 j 的开始时，其他教室一定正在进行课程，这些课程的结束时间一定比活动 j 的开始时间都晚，但结束时间一定比活动 j 早。
- 我们考虑前 d-1 个教室里的最后一个课程，找到结束的最早的那个教室，考虑其结束的时间点 f。那么在这个时间点 f，一定有 d-1 个活动同时进行。



为了证明上述结果一定成立，我们考虑下图这种反例是否存在。



假如第 1 个教室在 f 时刻没有在进行的课程，由于我们选取的 A 课程是所有 d-1 个教室中结束时间最早的，那么第 1 个教室里在 f 时刻之后一定安排有另一门 B 课程并且该课程的开始时间在 A 课程的结束时间之后。根据最小空隙原则，B 应该安排在 A 之后（2 号教室），而不是 1 号教室。因此这种反例是不存在的。

综上，可知在 f 时刻共有 d 个课程同时进行，因此至少需要 d 个教室。我们的贪心策略恰好需要 d 个教室，所以是最优的。

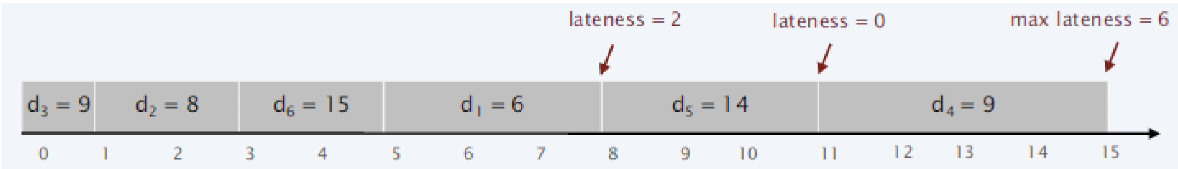
最小化任务延迟

问题简述

已知一些独占任务 j（如下图所示），处理 j 的所需时间为t_j，其截止时间为d_j。如果 j 的开始时间为s_j，则其结束时间$f_j = s_j + t_j$。规定延迟时间（Lateness）$l_j = \max\{0, f_j - d_j\}$。我们希望最大延迟时间 $\mathrm{L} = max_j l_j$ 最少。

	1	2	3	4	5	6
t_j	1	10	1	4	3	2
d_j	100	10	9	9	14	15

下图给出了一种安排方案，其 max lateness = 6：



几种自然的贪心策略：

- Shortest processing time first（最短处理时间优先）：按照处理时间t_j的升序安排任务
- Earliest deadline first（截止时间优先）：按照截止时间d_j的升序安排任务
- Smallest slack（最小的松弛）：按照松弛时间$d_j - t_j$的升序安排任务

最短处理时间优先（非最优）

我们可以举出一个反例说明这种情况并非最优。如下图，如果一个任务 2 的处理时间t_2很长，但是其截止时间d_2又很早，而另一个任务 1 的处理时间t_1很短，但截止时间d_1又很迟。按照这种算法的思路，会优先安排可以较快做完的任务 1，这样会导致任务 2 带来延误。

	1	2
tj	1	10
dj	100	10

最小的松弛（非最优）

我们可以举出一个反例说明这种情况并非最优。如下图，如果一个任务 2 的松弛时间$d_2 - t_2$很短，处理时间却很长，而另一个任务 1 的松弛时间比任务 2 长，但是处理时间很短，截止时间t_1又很早。按照这种算法的思路，会优先安排松弛时间较早的任务 2，由于任务 2 的完成时间很长，会对截止时间较早的任务 1 造成很大延误。

	1	2
tj	1	10
dj	2	10

截止时间优先（最优）

根据截止时间排序（从小到大），然后无缝连续安排任务。该算法的伪代码描述如下：

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

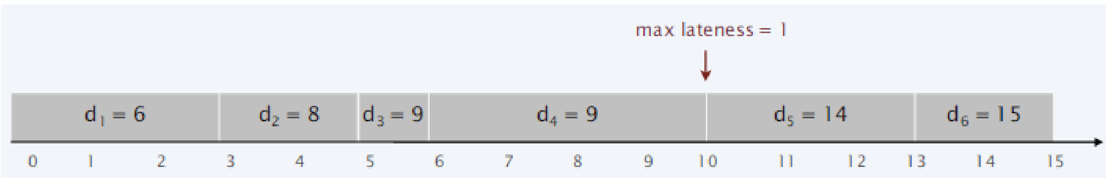
  Assign job j to interval $[t, t + t_j]$.

  $s_j \leftarrow t$; $f_j \leftarrow t + t_j$

  $t \leftarrow t + t_j$

RETURN intervals $[s_1, f_1], [s_2, f_2], \ldots, [s_n, f_n]$.

使用该算法对给出的例子进行任务安排，其 max lateness = 1：



EARLIEST-DEADLINE-FIRST 算法的最优性证明

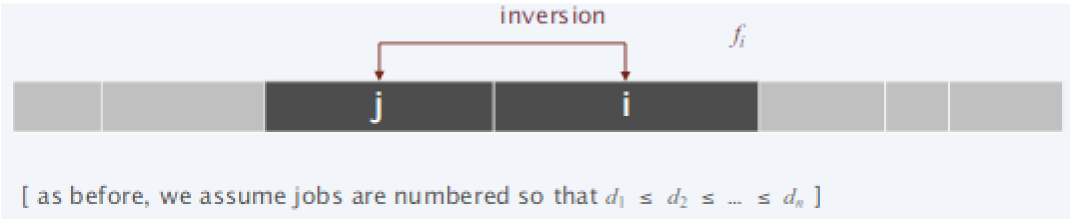
为了证明 Earliest-deadline-first 算法是一种最优的贪心策略，我们先引入一些概念和一些观察。

空闲

- 观察 1：一定有一种最佳的安排方式可以让任务之间没有空闲。
- 观察 2：Earliest-deadline-first 算法就没有空闲。

逆序

✧ 定义：给定一个安排 S，如果任务 i 比 j 截止时间早，但是 j 却安排在 i 前面，那么 i 和 j 就称为一对逆序。如下图所示：

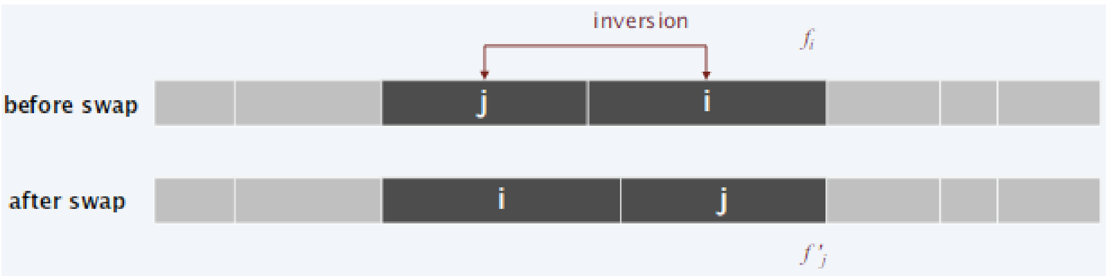


- 观察 3：Earliest-deadline-first 算法没有逆序。
- 观察 4：如果一种安排方式（没有空闲时间）里有逆序，那么一定存在一对相邻的逆序。

➢ 性质：交换两个相邻的逆序任务，一定不会增大最大延迟时间。

➢ 证明如下：

令l为交换前的延迟时间，l'为交换后的延迟时间



- 对于$k \neq i, j$ 的任务，$l_k = l'_k$（交换两个相邻的任务对其它任务的延迟时间不产生影响）
- $l'_i \le l_i$ （将任务 i 提前，延迟时间必然不会增大）
- 如果任务 j 是延误的，$l'_j = f'_j - d_j$ （定义）

$= f_i - d_j$ （交换后 j 的结束时间变为交换前 i 的结束时间）

$\le f_i - d_i$ （因为 i 和 j 是逆序的，所以$d_i \le d_j$）

$\le l_i$ （定义）

这说明第 j 个任务交换之后的新 late 不会比第 i 个原有的 late 更多。

综上，交换两个相邻的逆序任务，一定不会增大最大延迟时间。

最优性证明

- 欲证：Earliest-deadline-first 算法得出的 S 是最优安排方式。
- 反证法：记 S*为一种逆序最少的最优安排方式，不妨设 S*没有空闲。
  - 如果 S*没有逆序，那么 S=S*。
  - 如果 S*有逆序对，记相邻的逆序对为 i 和 j。由上述证明可知，交换 i 和 j 一定不会增大延迟时间，而且还会减少逆序的数量。这就和 S*的定义矛盾。

思考题

在本题中我们的优化目标是最小化 max lateness，如果将目标变为最小化所有的 lateness 之和，那么 Earliest-deadline-first 算法还能得到最优解吗？

贪心分析的策略总结

- 贪心法总是领先：贪心法得到的解始终领先于其他算法的解。
- 结构边界：找出每个可行解的一种结构性边界，再证明贪心法总能达到这个边界。（例子：安排下所有活动的情况下，最小化教室问题）
- 交换（变换）：将某个最优解变换为贪心法得到的解，而不使之变坏。（例子：树形结构中的最大独立集问题）

最优缓存替换策略（略讲部分）

理想策略是所谓“最优法”：把未来最晚用到的缓存牺牲掉。由于做不到，因此只能用 LRU 来近似，这样有时候效果不好，比如在循环遍历数组中的元素超过了缓存大小的时候。

图论中的贪心 – 最短路径算法

正权图的 DIJKSTRA 算法

维护一个顶点集 S，起点 s 到其中每个顶点 u 的最短路长度 d(u)已经算好。在集合外的顶点中找到一个顶点 v，要求$\mathrm{\pi(v)} = \min_{u\in S} d(u) + l(u,v)$最小，并记 d(v)为上述$\mathrm{\pi(v)}$，然后加入集合。

正确性证明

- 欲证：维护的顶点集 S 里的每个顶点 u 通过上述算法得到的 d(u)都是 s 到 u 的最短路长度。
- 归纳法：
    - 在 S 里只有 s 的时候易证。
    - 在 S 里有 k 个顶点时，假设成立，那么令 v 是加入 S 的新顶点，(u, v)是算法选中的边，那么$\mathrm{\pi(v)}$就是 s 到 u 的最短长度加上(u, v)的边长。
    - 对于每个 s 到 v 的路径 P，它一定不比$\mathrm{\pi(v)}$短：令(x, y)是 P 里离开 S 的第一条边，基于已有的假设，有$\mathrm{l(P) \geq d(x) + l(x,y) \geq \pi(y) \geq \pi(v)}$。

综上，命题得证。

优化

- 可以不用非要从公式算出$\mathrm{\pi(v)}$，而是可以每次迭代更新$\mathrm{\pi(v)}$为更小的值。
- 使用优先队列来选择使$\mathrm{\pi(v)}$最小的 S 外面的顶点（老师提醒注意一下 STL 里面，PriorityQueue 接口与 Queue 接口的区别）。