IOI2010

中国国家集训队 作业

最大收益

解题报告

最大收益

How to Get the Maximum Income

解题报告

中山市第一中学 冯齐纬

目录

题目描述

给出 N 件单位时间任务，对于第 i 件任务，如果要完成该任务，需要占用[Si, Ti]间的某个时刻，且完成后会有 Vi 的收益。求最大收益。

$N \le 5000$，$1 \le Si \le Ti \le 10^8$，$1 \le Vi \le 10^8$。

澄清：一个时刻只能做一件任务，做一个任务也只需要一个时刻。

下面举一个样例说明。

样例输入	样例输出	样例说明
4 1 1 2 2 2 2 1 2 3 1 3 1	6	共有四个任务，其中第一个任务只能在时刻 1 完成，第二个任务只能在时刻 2 做，第三个任务只能在时刻 1 或时刻 2 做，第四个任务可以在[1,3]内任一时刻完成，四个任务的价值分别为 2、2、3 和 1。一种完成方案是：时刻 1 完成第一个任务，时刻 2 完成第三个任务，时刻 3 完成第四个任务，这样得到的总收益为 2+3+1=6，为最大值。

问题分析

算法一·直接搜索

看到题目，我们能立即得到一个正确的算法：

由于每件任务只能在指定区间的时刻完成，对每个任务 i，我们不妨枚举它做与否，如果做再枚举 Si 到 Ti 中的某个未占用时刻来完成它，同时标记所有的时刻是否被占用。记 max{Ti}-min{Si}+1=L，对于每一件事情我们都有最多 L+1 种选择，枚举出对待所有事情的方案后更新答案即可，这样不难得到一个 $O(L^N)$的算法。

这个算法的实现并没有什么难度。程序实现可以看附件。

但是这个复杂度是远远不能满足题目要求的，这只是给我们一个思路，我们还必须对算法进行大幅度改进。

算法二·搜索+二分图匹配

题目要求选若干个任务，使得选择的每个任务都可以在一个时刻完成，而时刻互不相同。假设我们已经确定取哪些任务，剩下的就是判断可行性了。任务与时刻的联系可以看作是不能重复的匹配，这很容易使人联想到二分图匹配。当我们确定一个集合 S 的任务，把任务当成 X 点，把时刻当成 Y 点，可建立如下二分图：

- 若任务 i∈S，则增添 Xi 到 X 点集合对应任务 i；

• 若对时刻 j 存在 i∈S 满足 Si≤j≤Ti，则增添 Yj 到 Y 点集合与时刻点 j 对应；

• 若任务 i 可以在时刻 j 被完成（Si≤j≤Ti），那么增添边(Xi, Yj)到边集 E。

这时，求出二分图的最大匹配，如果所有的 X 点都能有相应唯一匹配的 Y 点，就意味着任务集合 S 可以被全部完成，故可以用 S 集的所有任务的价值和更新答案。

于是我们得到了第二个算法：首先枚举每个任务是否选择，接着抽出选择的任务，试图寻找与所有时刻的匹配，若存在完备匹配则更新答案。因为枚举任务的复杂度为 $O(2^N)$，每一次判断的复杂度为 $O(N*|E|)=O(N^2*L)$，故算法总复杂度为 $O(2^N*N^2*L)$，题目中，N≤5000，$L\le10^9$，该复杂度仍然是个天文数字，我们须继续优化算法。

算法三：最大权二分图匹配

上面算法的瓶颈在于枚举，这限制了算法无论在其他方面怎么优化总复杂度都将是指数级别的。上面构造二分图的想法给了我们很大启示，能不能把先生成方案后构造二分图改成先构造二分图后求方案呢？不妨一开始就构造整个二分图，既然是求最大权，就给边加上权。构建二分图如下：

• 对任务 i∈[1,N]，增添点 Xi 到二分图的 X 集对应任务 i；

• 对时刻 j 若存在 i∈[1,N]满足 Si≤j≤Ti，则增添点 Yj 到 Y 集对应时刻点 j；

• 对任意任务 i 和时刻 j，若 Si≤j≤Ti，那么增添边(Xi, Yj)到边集 E，边权为 Vi。

这样子，相当于求 N*L（L 为涉及的时刻点个数）的带权二分图的一个合法匹配，使得所有匹配边的边权和最大。

求这个最大权匹配的算法不难实现。首先，若 N≠L，则增添额外的空白 X 点或 Y 点，使得最终 X 点和 Y 点数目相等。额外增加的点初始不连边。接着，对所有 XY 点对 u、v（即 u∈X 集，v∈Y 集），若不存在边(u, v)，则增添权为 0 的边(u, v)。这时得到了一个完全二分图。在这个二分图上，用经典的 KM 算法求出最大权完备匹配的总权值，该权值即为所求的最大总权和。

这样，先按上述方法构造 N*L 的带权二分图，然后求出最大权的一个匹配方案，得出了算法三。该算法的时间复杂度为 $O(\max\{N, L\}^3)$。

算法四：离散化+最大权二分图匹配

虽然上述算法已进了一大步，但由于 L 是无穷大的数（定义中 $L\le10^8$），算法仍未满足要求。我们所谓“相关时刻”定义为在某个

任务能取到的区间里的时刻，不难发现，这样会造成大量无用的 Y 点的存在。如果删去这些 Y 点，那么效率将大大提高。也就是说，我们可以去除其他无穷多的时刻点，只选择极少量的时刻点，使得只在这些时刻做任务得到的最大价值不变。

我们这样来找有用的时刻点。初始时，所有时刻点标记为黑色。接着，对于每一个任务 i(Si, Ti, Vi)，找到最小的 k，满足 k≥Si 且时刻点 k 为黑色，然后把时刻 k 标记为白色。最终，我们得到了恰好 N 个标记为白色的时刻点，这 N 个点就是要找的有用时刻点。不妨称这些白色的点为“活跃点”。

活跃点的求法并不困难，可以参照附录中的算法 A。

为了证明活跃点的作用，我们先引入一个命题。

命题一. 对于一个任务集合 S 中的所有任务，如果它们能在活跃点集合 T 中的时刻被全部完成，那么必然存在这样一种基于贪心的完成方式：从前到后依次扫描 T 中每一个时刻，在一个时刻，如果存在此时能完成的且尚未完成的任务，就完成 Ti 值最小的那个。

对命题一的证明：

设到目前为止剩下的活跃点和任务集合依然存在对应匹配，目前扫描剩余活跃点集合中最前的时刻。如果存在解是在当前时刻不做任务的，那么当前时刻做了任务一定不会差，即可以保证仍能完成匹配；

否则的话，由于该时刻点是剩余活跃点集合里最小的，即该时刻之前不能多完成任务，故如果存在方案是不选择 Ti 值最小的任务的，那么选择 Ti 值小的任务意味着给后面留下了一个 Ti 值相对大的“更容易”被完成的任务，它必然还是能被完成，所以后面一定还存在解。

故这种贪心方法一定能够构造出一种任务与时刻点的匹配方案。证毕。

然后，我们引出一个重要的命题。

命题二. 只在“活跃点”完成任务跟在所有时刻点完成任务所得的最优总价值相同。

对于这个命题的详细证明可以参见附录。

根据命题二，构建 N*N 的完全二分图：

- 对 i∈[1,N]，增添 Xi 到 X 集对应任务 i；
- 对 j∈[1,N]，记第 j 个活跃点为 Pj，增添 Yj 到 Y 集对应时刻点 Pj；
- 对 i, j∈[1,N]，增添边(Xi, Yj)到边集，若 Si≤j≤Ti 边权定为 Vi，否则边权定为 0。

再用 KM 算法即可得到最终的答案。

这个算法的时间复杂度为 $O(KM)=O(N^3)$。对于 $N\leq5000$，这个算法还是有所欠缺。

算法五·贪心+离散化+二分图匹配

我们需要求若干个价值和尽量大的任务，使得它们存在与时刻点的完备匹配。也就是说，我们要放弃一些任务。为什么要放弃呢？因为不能腾出一个时刻给它。为什么不能？因为它能取的所有时刻都被占用了。既然一个任务只会占用一个时刻点，如果两个任务去“争抢”一个时刻点，我们肯定选择价值高的任务。这不难得出命题三中的贪心策略。

命题三. 把任务按价值从大到小排序，依次考虑每一个任务。设之前选择的任务集合为 S，如果当前任务加到 S 中仍能保证每个任务都匹配到时刻点（可以只是活跃点），那么选择当前任务，否则不选择当前任务。这种贪心策略必然能够得到一种方案，其总价值为所求的最大价值。

对命题三的证明：

设当前任务为 i，前 i-1 个任务最优任务集合为 S。若当前任务加到 S 中仍能保证每个任务都匹配，那么必然要选择该任务，否则总价值就不是最优了；若加入 S 后会产生不能都匹配，那说明至少有一个

任务不能选，因为前面的任务的价值都比当前任务大，而只要不选当前任务，前面的不会发生冲突，当然不能牺牲之前一个价值更大的任务来选择当前的，所以必然不选择当前任务。又初始集合是最优，此后每一次选择都能保证得到的集合是最优的，故最终的结果为最优。

综上所述，贪心的正确性得证。

有了这个贪心，我们可以得出下面的算法。

先求出所有的活跃点以离散化时刻点。构建二分图：把所有任务看作 X 点，把所有活跃点看作 Y 点，若一个任务能在一个时刻完成，则在该任务与该时刻对应的点间连边。然后，把任务按价值从大到小排序，依次处理每一个任务对应的 X 点。若能够找到增广路使得当前 X 点匹配上某个 Y 点，且前面选择的任务集合中的任务保证仍能匹配活跃点，即表明当前任务能加到选择集合中，那么选择该任务，否则放弃该任务。

因为任务总数为 N，每一次找增广路的复杂度为 $O(N^2)$，所以这个算法的时间复杂度仍然是 $O(N^3)$。虽然复杂度没有变化，但至少提供了贪心这个策略。这为后面的优化做了基础。

算法六·贪心+堆

这个思路并不会直接引出后面的算法，但这里还是有必要说明一下这个算法，至少在复杂度方面，本算法起了承前启后的作用。

上面已经证明了贪心的正确性，其实将对任务按价值贪心和命题一里已证的对 Ti 值的贪心综合，不难得到这样的算法：

将任务按价值从大到小排序，保存已选任务集合 S（初始为空），然后依次扫描每一个任务。对于一个任务，如果把它加到任务集合中仍然能保证集合中所有任务都可以被完成，那么选择该任务。判断时，在每一个时刻完成能完成、尚未完成的且 Ti 值最小的任务，如果不存在则什么都不做。这个选择可以用堆实现。最后，看一下是否完成了所有的任务就可以完成判断了。

当然，并不需要枚举所有的时刻，可以选择把时刻离散化，也可以加一些判断来实现，这里不详细指出了。

算法时间复杂度为 $O(N^2*logN)$。

算法七·贪心+离散化+线性匹配

在计算过程当中，我们保证当前匹配方案与由命题 1 中的贪心策略得出的匹配方案一致，然后试图快速判断一个任务是否能加入，如果可以则更新匹配状态。

初始时，找出所有的活跃点，然后把任务按价值从大到小排序，依次考虑。

对第 i 个任务，试图寻找 Si 开始的某个时刻点来完成它。如果能找到的话，那么任务 i 就可以加到选择集合中。调用函数 find(I,Si)实现：

```
Algorithm find(i, x)                    //试图找到 x 开始的某时刻完成任务 i
    If x>Ti
        return false;
    if  时刻 x 尚未匹配任务
    {
        将时刻 x 与任务 i 匹配;
        return true;
    }
    j=时刻 x 当前匹配的任务;
    If Ti>Tj                            //按照命题一的贪心策略来选择
        return find(i,x+1);
    else
        if find(j, x+1)
        {
            将时刻 x 与任务 i 匹配;
            return true;
        }
        else return false;
```

把任务匹配时刻想象成将任务放到时刻点上，那么形象地说，该算法实现的是试图把任务 i 放到到某个时刻，然后把后面一些任务向后“挤”的过程。

因为一次扫描最多扫过全部的活跃点，易知算法七的时间复杂度为 $O(N^2)$。

算法七的一点优化

与任务 i 有关的计算中，我们在考虑当前任务(S0,T0,V0)与当前时刻点 x 时，每一次可以先找到 x 后面第一个活跃点 x'满足：x'为空活跃点或 x'此时匹配任务的 Ti 值大于 T0。若 x'为空活跃点则已经完成，否则用当前任务替换 x'已匹配任务，转化为 x'此时匹配的任务向后寻找匹配的过程。如果这个过程失败，可以增添一个标记，表示 x'匹配的这个任务不可以向后面找到匹配，这样以后如果再次遇到 x'就不需要再向后面扫一次了。

虽然这样子算法的效率依然是 $O(N^2)$，但因为常数还是很小，而且能去除一些不必要的计算，实际仍有一定的效果。

输入	输出
4 1 1 3 1 2 2 1 3 1 1 2 4	8

我们举左边所示的例子来描述最终的算法。

为方面描述，记一个限时[S,T]的价值为 V 的任务为(S,T,V)。

按照算法，首先将任务按价值排序，排序后依次为(1,2,4)，(1,1,3)， (1,2,2)，

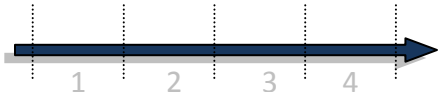

图 A1

(1,3,1)。离散化得到的“活跃点”为时刻 1、2、3 和 4。如图 A1，初始时四个活跃时刻点均为空（即未被任何任务占用）。

考虑第一个任务，按照算法，它将被放到时刻 1 上。把任务一(1,2,4)与时刻 1 匹配，结果如图 A2。

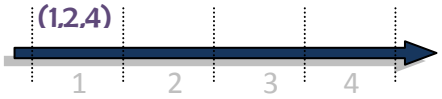

图 A2

接着考虑第二个任务。按照最终的算法，我们找时刻 1 与它匹配，并将之前的任务一替换，将任务一(1,2,4)挤到时刻 2 上。将任务一(1,2,4)与时刻 2 匹配、任务二(1,1,3)与时刻 1 匹配得到图 A3 的状态。

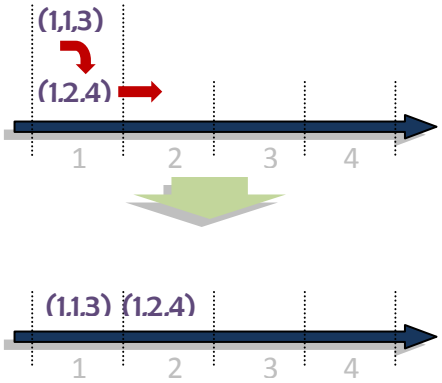

图 A3

对第三个任务(1,2,2)，首先考虑在第一个时刻。因为时刻 1 不是空时刻点，且时刻 1 已经匹配的任务是(1,1,3)，其 Ti 值比当前任务小，故当前任务向后滑动，继续寻找时刻点。

同样，在第二个时刻，因为时刻 2 不为空且匹配任务(1,2,4)的 Ti 值不比当前任务大，当前任务应继续向后滑动。

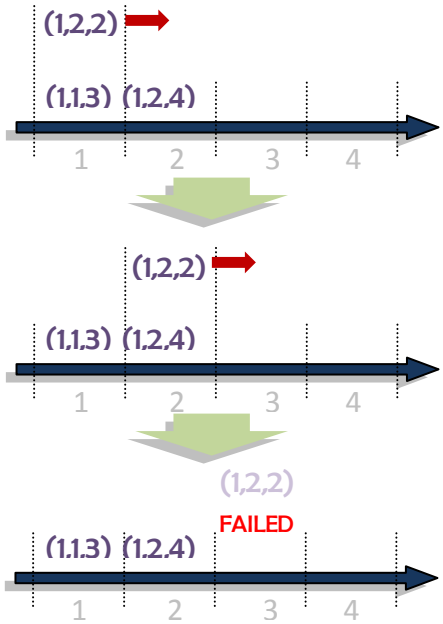

图 A4

因为当前任务的 Ti 值为 2<3，故到了时刻 3 已经意味寻找失败。因此第三个任务不被选择，匹配状态保持不变。如图 A4 所示。

到第四个任务(1,3,1)，首先考虑时刻 1。时刻 1 匹配的任务的 Ti 值比当前的 Ti 值小，故任务向后滑动。

到时刻 2，同样，匹配任务(1,2,4)的 Ti 值比 3 小，故继续向后寻找。

这时找到一个空活跃点时刻 3，于是将当前任务放到这里。将任务四(1,3,1)与时刻 3 匹配，得到新的匹配状态。如图 A5。

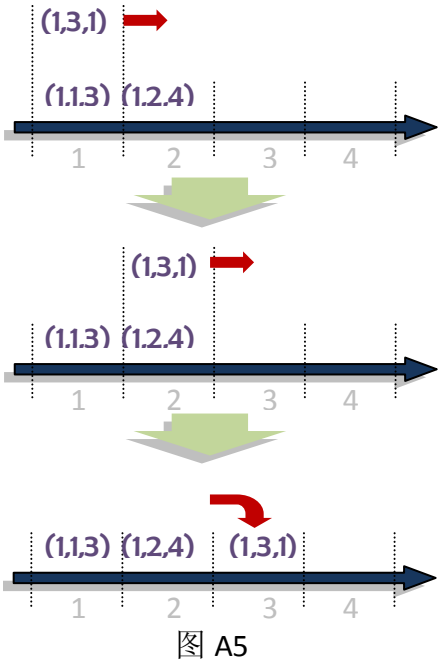

图 A5

最后，我们得到最大收益为 3+4+1=8。

数据生成与测试结果

本题共设置了 15 组测试数据，对于上面所述的 8 个算法，笔者均实现并进行了测试，测试结果如下：

测试点(分值)	算法一	算法二	算法三	算法四	算法五	算法六	算法七	算法七的优化
#1 (6)	0.63 s	0.20 s	0.02 s	0.02 s	0.02 s	0.02 s	0.02 s	0.02 s
#2 (6)	0.03 s	0.13 s	0.02 s	0.02 s	0.02 s	0.02 s	0.02 s	0.02 s
#3 (6)	-	-	0.03 s	0.02 s	0.02 s	0.02 s	0.02 s	0.02 s
#4 (6)	-	-	0.02 s	0.02 s	0.02 s	0.02 s	0.02 s	0.02 s
#5 (6)	-	-	0.47 s	0.53 s	0.02 s	0.03 s	0.02 s	0.02 s
#6 (7)	-	-	-	-	-	4.38 s	0.31 s	0.20 s
#7 (7)	-	-	-	-	0.08 s	0.55 s	0.03 s	0.02 s
#8 (7)	-	-	-	-	7.59 s	1.61 s	0.09 s	0.08 s
#9 (7)	-	-	-	-	-	0.77 s	0.09 s	0.09 s
#10 (7)	-	-	-	-	-	1.95 s	0.14 s	0.06 s
#11 (7)	-	-	-	-	-	5.01 s	0.58 s	0.70 s
#12 (7)	-	-	-	-	-	4.76 s	0.45 s	0.28 s
#13 (7)	-	-	-	-	0.14 s	0.55 s	0.03 s	0.03 s
#14 (7)	-	-	-	-	-	4.92 s	0.55 s	0.27 s
#15 (7)	-	-	-	-	-	3.05 s	0.23 s	0.13 s
总时间	-	-	-	-	-	*27.6 s*	*2.6 s*	*1.9 s*
得分	12	12	30	30	37	58	100	100

其中，绿色数字表示在规定时间（1s）内出解的测试的时间，红色数字表示超时的测试，'-'表示运行时间超过 10 秒。

（注：对于一些算法，某些极限数据可能导致运行错误，这里统一标记为'-'。）

现在详细给出每一组测试数据的生成方法。

数据 1：N=15、1≤Si≤Ti≤10 的随机小数据。

数据 2：N=15，(Si, Ti)取遍[1,5]间的所有整数对，即(1,1), (1,2), ..., (1,5), (2,2), ..., (2,5), ..., (4,5), (5,5)。Vi 为随机值。如图 B1，所有任务的可行时刻区间用蓝色线段表示出。

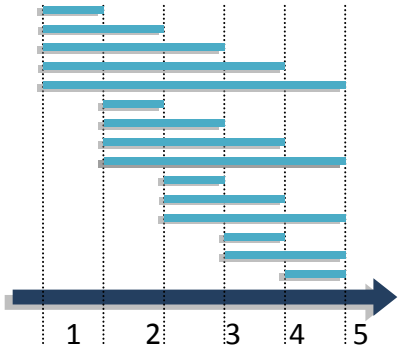

图 B1

数据 3：N=100、1≤Si≤Ti≤10 的随机数据，线段的密度较大。

数据 4：类似数据 2，N=91，(Si, Ti)取遍[1,13]间的所有整数对，即(1,1), ..., (1,13), (2,1), ..., (13, 13)。

数据 5：N=500、1≤Si≤Ti≤50 的随机数据。

数据 6：N=5000、1≤Si≤Ti≤5000 的随机大数据。

数据 7：N=5000、1≤Si≤Ti≤50 的随机数据，其中涉及的时刻点较少，也就是(Si, Ti)线段密度大。

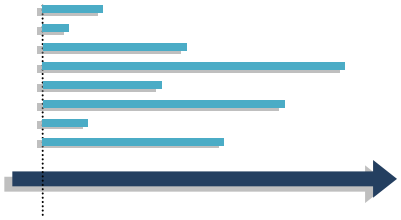

图 B2

数据 8：N=5000，Si=1，Ti 为[1,1000]的随机值，Vi 为[1,10^8]的随机值。如图 B2。

数据 9：N=5000，所有的任务只可以在某一个时刻完成，即 Si=Ti，Vi 为随机值。任务的 Si 和 Ti 会有重复，最多涉及 2000 个时刻点。

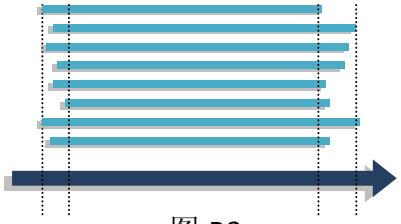

图 B3

数据 10：N=5000，1≤Si≤5，1000≤Ti≤1010，Vi 为随机。这时所有线段(Si, Vi)呈现基本一致的情形，只在两端略有差异。如图 B3。

数据 11：N=5000，Si=1，Ti 依次为 1 至 5000，Vi 值按 Ti 值递增。

数据 12：N=5000，Si=1，Ti 为 1 至 3500 间的随机数，且 Ti 值越小密度越大。Vi 值按 Ti 递增，相同 Ti 的任务 Vi 相同。如图 B4。

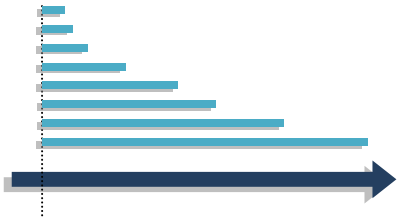

图 B4

数据 13：类似数据 2 和数据 4，N=4950，(Si, Ti)取遍[1,99]间的所有整数对，即(1,1), …, (1,99), (2,1), …, (99, 99)。可以参看图 B1。

数据 14：类似数据 10，1≤Si≤5，3000≤Ti≤4000，Vi 为随机。

数据 15：价值 Vi 均为 1，1≤Si≤5，Si≤Ti≤4000。

更优的算法

笔者之前尝试了很久寻找比 $O(N^2)$在复杂度方面更优的算法，不过最后还是失败了。这里就提供一些想法，希望能够起到抛砖引玉的作用。如果读者有什么看法或意见的话可以与我联系，我的电子邮箱是 gdfqw93@163.com。

按照算法七优化算法中的模型，如果每一个匹配了任务的活跃点 x 加一个指针指向后面第一个 x′ 满足 x′为空活跃点或 x′此时匹配任务的 Ti 值大于 x 匹配任务的 Ti 值。把指针当作有向边吧，这样大致就构成了一些树（一个森林），然后每一次试图插入任务就是更新树的过程。但遗憾的是，由于“挤”的总次数已经是 $O(N^2)$级别的，直接模拟这个过程还是不能实现优化。

从另一个角度来实现判断。一个集合 S 的任务能被完全完成的充要条件是：不存在区间[S0,T0]，满足(满足 S0≤Si≤Ti≤T0 且 i 在 S 集中的任务 i 的数目) >T0-S0+1。但这里我也没有找到快速的判断方法。

会不会有更优的算法？会不会通过一些巧妙的构造与转化能够降低复杂度？还是 $O(N^2)$的算法已经是最优了？不好定论。

题目模型

不妨交代一下我出题的灵感吧。大概几个月前与邓原同学的交谈中知道这样一道题目：在 CS 游戏中，有 n 个敌人于时刻 0 同时出现，并且每个人有一个消失时间和一个价值，你可以在消失时间前任一时刻消灭他。现在你每一时刻可以射击一个人并获得相应的价值，求最后的最大总价值。这题用堆很容易解出来了。当时我就想能不能改成每个人都有出现时间和消失时间，然后求总价值，于是就形成了这道题。

其实这题的实际应用还是蛮普遍的。比如说，我们经常要选择做一些事情，有的事情逾期就不能做了，有的事情要等时机成熟。一个人活在世上最基本的就是去做事，去感受，去体验付出与收获。人不可能什么事情都做，什么事情都完成得很好，这便有了取舍；做事情要花时间，这便有了对时刻点的占用；一件事也不是任何时刻都可以做的，可能在某段时间才能够完成，这段时间可能很长，可能稍纵即逝，这便有了时限；完成一件事会有收获，可以是物质上的，也可以是精神上的，这便有了收益。这里我们抽象出一个最简单、最基本的模型，就成为了这一道题目。

实际当中解决这些问题并不是容易的。一般看来我们会先做比较“重要”的事情，可是何谓之“重要”？假如单看它的价值，那么可

能出现这样的情况：一件事情价值很大，而截止日期还有很久，一件事情的截止时间很短，机会稍纵即逝，价值却偏偏没有前一件大。如果先做价值大的，那很可能就做不了另一件事情，而实际上我们可以先做价值小的，然后再做价值大的。但同样，单看它的结束时间也是不行的，必须综合考虑。这时我便试图设计算法来解决这个问题，这个过程同样不容易。

读者可以试图考虑这样的更一般的问题：假如事件还有一定的完成时间，即要在[S,T]间的连续或不连续的 P 个时刻完成；假如一个事件的完成时间限制不是一个区间，而是很多个区间；假如完成了某些任务才能做另一些任务等等。

本题中，实际上仅有价值和时间两个约束，若是实际中可能还有更多更多的限制、困扰和其他影响因素。大千世界是难以用简单的数学模型来描述的。既然这样，人又何必要求完美，只要活过、做过，人生都是五彩斑斓的。

附录

对命题二的证明

命题二. 只在“活跃点”完成任务跟在所有时刻点完成任务所得的最优总价值相同。

证明：

试图用数学归纳法证明命题。

对于第一个任务(S1, T1, V1)，按上面方法确定的第一个活跃点为 S1，而因为 S1≤S1≤T1，故能够只在 S1 做任务，取得最大价值 V1。

对于前 m 个任务(1≤m<N)，已经激活了 m 个时刻点，假设存在只在这 m 个时刻点里完成的最优方案，记为任务集合 S。现在考虑第 m+1 个任务，无非有下面几种情况：

1）前 m+1 个任务的最优方案不包含任务 m+1。

这样，按照我们的方法，只会额外增加一个活跃点，而不会影响之前的活跃点。前 m+1 个任务的最优方案等同于前 m 个任务的最优方案，也就是只与前 m 各活跃点有关，所以前 m+1 个任务的最优方案可以只与前 m+1 个活跃点有关。

2）需要在 S 中删除若干个任务（事实上只会有一个），以腾出一个时刻点给当前任务 m+1。

与上面相同，当前任务只是替换了之前一个位置上的任务，所用的时刻点是不变的，而增加的活跃点不对之前活跃点产生影响，故前 m+1 个任务的最优方案只与活跃点有关。

3）能把第 m+1 个任务加到选择集合中，且不删除之前选择的任务。

由命题一知：按照命题一的贪心方法，必然能得到唯一的前 m 个任务与前 m 个活跃点的匹配方案 R。在当前的匹配 R 下，按上面的方法加入新的活跃点，然后试图把当前的第 m+1 个任务加入，找到一个时刻点与它匹配。我们从当前任务(S0,T0,V0)的限时起点 S0 开始枚举每一个时刻点，如果该时刻点尚未匹配任务，则意味着已经完成匹配，停止；如果该时刻点匹配了任务 k 满足 Tk≤T0，则继续；如果该时刻匹配了任务 k 满足 Tk>T0，则将任务 k 与当前任务交换，该时刻匹配当前任务，而继续扫描下去时将任务 k 作为新的“当前任务”（类似于算法六）。可以看出，这样子寻找匹配的方式实际上等价于命题一中所用的贪心策略。因为前 m+1 个任务一定有与所有时刻点的匹配方案的，所有该算法最终必然能到达找到空时刻点的步骤，即能完成匹配。由寻找策略易知，最终达到的那个空时刻点为 S0 及以后的第一个空时刻点，这个点要么之前已经激活、要么恰好在该步被激活，总之一定是空活跃点，故这样得到的匹配依然是任务与活跃点的。

综上所述，可以证明出：对一个任务集合，如果存在它们与所有时刻点的匹配，那么必然存在它们与活跃时刻点的匹配。故原命题得证。

求出所有活跃点的算法 A

因为总活跃点与任务的顺序无关，不妨把所有任务按 S 值排序，然后从前到后依次寻找每一个活跃点。记第 i 个活跃点为 x[i]，记 x[0]=0，则有 x[i]=max{x[i-1]+1, Si}。算法如下：

Algorithm A. Find active points

```
将活跃点集合设为空;
把任务按照 Si 从小到大排序;
Int x=0;
For i=1 to N
    x=max(x+1, Si);
    新增活跃点 x;
Endfor;
```

算法 A 的时间复杂度为 O(N*logN)。

算法一的程序实现

```
// For IOI2010.
// Problem Income, Algorithm I.
// Offered by Long Haomin, No.1 Middle School of Zhongshan.

var
  n,i,j,k,l,w,v,max:longint;
  a:array[1..5000,1..3] of longint;
```

```
  g:array[1..5000]of longint;
  procedure dfs(x,now:longint);
    var
      i:longint;
    begin
      if x=n+1 then
        begin if now>max then max:=now; exit;end;
      for i:=a[x,1] to a[x,2] do
        begin
          if g[i]=0 then
            begin
              g[i]:=1;
              dfs(x+1,now+a[x,3]);
              g[i]:=0;
            end;
        end;
      dfs(x+1,now);
    end;
begin
  assign(input,'income.in');reset(input);
  assign(output,'income.out');rewrite(output);
  readln(n);
  for i:=1 to n do
    readln(a[i,1],a[i,2],a[i,3]);
  fillchar(g,sizeof(g),0);
  max:=0;
  dfs(1,0);
  writeln(max);
  close(input);close(Output);
end.
```

算法二的程序实现

```
// For IOI2010.
// Problem Income, Algorithm II.
// Written by Feng Qiwei in December, 2009.
```

```
#include <iostream>
#include <cstring>
using namespace std;

const int maxn=5010;

int s[maxn], t[maxn], v[maxn];
bool vy[maxn];
int lnk[maxn];
int n;

bool find(int x)
{
    for (int y=s[x]; y<=t[x]; y++)
        if (!vy[y])
        {
            vy[y]=true;
            if (lnk[y]==0 || find(lnk[y]))
            {
                lnk[y]=x;
                return true;
            }
        }
    return false;
}

int main()
{
    freopen("income.in", "r", stdin);
    freopen("income.out", "w", stdout);

    scanf("%d", &n);
    for (int i=1; i<=n; i++)
        scanf("%d%d%d", &s[i], &t[i], &v[i]);
    long long ans=0;
    for (int st=0; st<(1<<n); st++)
    {
        long long tmp=0;
        bool flag=true;
        memset(lnk, 0, sizeof(lnk));
        for (int i=1; flag && i<=n; i++)
```

```
            if (st&(1<<(i-1)))
            {
                memset(vy, false, sizeof(vy));
                if (!find(i)) flag=false;
                tmp+=v[i];
            }
        if (flag) ans>?=tmp;
    }
    printf("%I64d\n", ans);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

算法三的程序实现

```
// For IOI2010.
// Problem Income, Algorithm III.
// Written by Feng Qiwei in December, 2009.

#include <iostream>
#include <cstring>
using namespace std;

const int maxn=5010;

int s[maxn], t[maxn], v[maxn];
int lx[maxn], ly[maxn], lnk[maxn];
bool vx[maxn], vy[maxn];
int n, m;

int g(int i, int j)
{
    return (j>=s[i] && j<=t[i])?v[i]:0;
}

bool find(int x)
```

```
{
    vx[x]=true;
    for (int y=1; y<=n; y++)
        if (!vy[y] && lx[x]+ly[y]==g(x, y))
        {
            vy[y]=true;
            if (lnk[y]==0 || find(lnk[y]))
            {
                lnk[y]=x;
                return true;
            }
        }
    return false;
}

int main()
{
    freopen("income.in", "r", stdin);
    freopen("income.out", "w", stdout);
    scanf("%d", &n);
    m=0;
    for (int i=1; i<=n; i++)
    {
        scanf("%d%d%d", &s[i], &t[i], &v[i]);
        m>?=t[i];
    }

    n>?=m;
    for (int i=1, j; i<=n; i++)
        for (lx[i]=ly[i]=0, j=1; j<=n; j++)
            lx[i]>?=g(i, j);
    memset(lnk, 0, sizeof(lnk));
    for (int k=1; k<=n; k++)
        while (1)
        {
            memset(vx, 0, sizeof(vx));
            memset(vy, 0, sizeof(vy));
            if (find(k))
                break;
            int minv=1<<30;
            for (int i=1; i<=n; i++) if (vx[i])
```

```
                for (int j=1; j<=n; j++) if (!vy[j])
                    minv<?=lx[i]+ly[j]-g(i, j);
            for (int i=1; i<=n; i++)
            {
                if (vx[i]) lx[i]-=minv;
                if (vy[i]) ly[i]+=minv;
            }
        }
    long long ans=0;
    for (int i=1; i<=n; i++)
        ans+=g(lnk[i], i);
    printf("%lld\n", ans);

    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

算法四的程序实现

```
// For IOI2010.
// Problem Income, Algorithm IV.
// Written by Feng Qiwei in December, 2009.

#include <iostream>
using namespace std;

const int maxn=5010;

struct item
{
    int s, t, v;
} d[maxn];
int lx[maxn], ly[maxn], lnk[maxn];
bool vx[maxn], vy[maxn];
int pos[maxn];
int n;
```

```
int g(int i, int j)
{
    return (pos[j]>=d[i].s && pos[j]<=d[i].t)?d[i].v:0;
}

void sort(int fi, int la)
{
    if (fi>=la) return;
    int i=fi, j=la, g=d[(i+j)>>1].s;
    while (i<=j)
    {
        while (d[i].s<g) i++;
        while (g<d[j].s) j--;
        if (i<=j) swap(d[i++], d[j--]);
    }
    sort(fi, j); sort(i, la);
}

bool find(int x)
{
    vx[x]=true;
    for (int y=1; y<=n; y++)
        if (!vy[y] && lx[x]+ly[y]==g(x, y))
        {
            vy[y]=true;
            if (lnk[y]==0 || find(lnk[y]))
            {
                lnk[y]=x;
                return true;
            }
        }
    return false;
}

int main()
{
    freopen("income.in", "r", stdin);
    freopen("income.out", "w", stdout);

    scanf("%d", &n);
```

```
    for (int i=1; i<=n; i++)
        scanf("%d%d%d", &d[i].s, &d[i].t, &d[i].v);
    sort(1, n);
    for (int i=1, cur=0; i<=n; i++, cur++)
        pos[i]=cur>?=d[i].s;

    for (int i=1, j; i<=n; i++)
        for (lx[i]=ly[i]=0, j=1; j<=n; j++)
            lx[i]>?=g(i, j);
    memset(lnk, 0, sizeof(lnk));
    for (int k=1; k<=n; k++)
        while (1)
        {
            memset(vx, 0, sizeof(vx));
            memset(vy, 0, sizeof(vy));
            if (find(k))
                break;
            int minv=1<<30;
            for (int i=1; i<=n; i++) if (vx[i])
                for (int j=1; j<=n; j++) if (!vy[j])
                    minv<?=lx[i]+ly[j]-g(i, j);
            for (int i=1; i<=n; i++)
            {
                if (vx[i]) lx[i]-=minv;
                if (vy[i]) ly[i]+=minv;
            }
        }
    long long ans=0;
    for (int i=1; i<=n; i++)
        ans+=g(lnk[i], i);
    printf("%lld\n", ans);

    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

算法五的程序实现

```
// For IOI2010.
// Problem Income, Algorithm V.
// Written by Feng Qiwei in December, 2009.

#include <iostream>
#include <cstring>
using namespace std;

const int maxn=5010;

int s[maxn], t[maxn], v[maxn];
bool vy[maxn];
int lnk[maxn];
int n;

void sort(int fi, int la)
{
    if (fi>=la) return;
    int i=fi, j=la, g=v[(i+j)>>1];
    while (i<=j)
    {
        while (v[i]>g) i++;
        while (g>v[j]) j--;
        if (i<=j)
        {
            swap(s[i], s[j]);
            swap(t[i], t[j]);
            swap(v[i], v[j]);
            i++; j--;
        }
    }
    sort(fi, j); sort(i, la);
}

bool find(int x)
{
    for (int y=s[x]; y<=t[x]; y++)
        if (!vy[y])
```

```
        {
            vy[y]=true;
            if (lnk[y]==0 || find(lnk[y]))
            {
                lnk[y]=x;
                return true;
            }
        }
    return false;
}

int main()
{
    freopen("income.in", "r", stdin);
    freopen("income.out", "w", stdout);

    scanf("%d", &n);
    for (int i=1; i<=n; i++)
        scanf("%d%d%d", &s[i], &t[i], &v[i]);
    sort(1, n);
    long long ans=0;
    memset(lnk, 0, sizeof(lnk));
    for (int i=1; i<=n; i++)
    {
        memset(vy, false, sizeof(vy));
        if (!find(i)) continue;
        ans+=v[i];
    }
    printf("%I64d\n", ans);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

算法六的程序实现

```
// For IOI2010.
```

```
// Problem Income, Algorithm VI.
// Written by Feng Qiwei in December, 2009.

#include <iostream>
#include <cstring>
using namespace std;

const int maxn=5010;

struct item
{
    int s, t, v, tmp, ps;
} d[maxn];
int pos[maxn];
int list[maxn];
bool ch[maxn];
int n;

void sort(int fi, int la)
{
    if (fi>=la) return;
    int i=fi, j=la, g=d[(i+j)>>1].tmp;
    while (i<=j)
    {
        while (d[i].tmp<g) i++;
        while (g<d[j].tmp) j--;
        if (i<=j)
            swap(d[i], d[j]),
            i++, j--;
    }
    sort(fi, j); sort(i, la);
}

int h[maxn], num;

void add(int val)
{
    h[++num]=val;
    for (int i=num, j=num/2; j>0; i=j, j=j/2)
        if (h[i]<h[j])
            swap(h[i], h[j]);
```

```
        else
            break;
}

void del()
{
    h[1]=h[num--];
    for (int i=1, j=2; j<=num; i=j, j=j*2)
    {
        if (j<num && h[j+1]<h[j]) j++;
        if (h[i]<h[j]) break;
        swap(h[i], h[j]);
    }
}

bool check()
{
    num=0;
    for (int time=1, cur=1; time<=n; time++)
    {
        while (cur<=n && d[list[cur]].s==pos[time])
        {
            if (ch[list[cur]]) add(d[list[cur]].t);
            cur++;
        }
        if (num>0)
        {
            if (h[1]<pos[time])
                return false;
            del();
        }
    }
    return num==0;
}

int main()
{
    freopen("income.in", "r", stdin);
    freopen("income.out", "w", stdout);

    scanf("%d", &n);
```

```
    for (int i=1; i<=n; i++)
        scanf("%d%d%d", &d[i].s, &d[i].t, &d[i].v);
    for (int i=1; i<=n; i++) d[i].tmp=d[i].s;
    sort(1, n);
    for (int i=1, cur=0; i<=n; i++, cur++)
        pos[i]=cur>?=d[i].s;
    for (int i=1; i<=n; i++) d[d[i].ps=i].tmp=-d[i].v;
    sort(1, n);
    for (int i=1; i<=n; i++) list[d[i].ps]=i;

    long long ans=0;
    memset(ch, 0, sizeof(ch));
    for (int i=1; i<=n; i++)
    {
        ch[i]=true;
        if (!check())
            ch[i]=false;
        else
            ans+=d[i].v;
    }
    printf("%I64d\n", ans);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

算法七的程序实现

```
// For IOI2010.
// Problem Income, Algorithm VII.
// Written by Feng Qiwei in December, 2009.

#include <iostream>
#include <cstring>
using namespace std;

const int maxn=5010;
```

```
struct item
{
    int s, t, v, tmp;
} d[maxn];
int pos[maxn], lnk[maxn];
int stx[maxn], sty[maxn];
int n;

void sort(int fi, int la)
{
    if (fi>=la) return;
    int i=fi, j=la, g=d[(i+j)>>1].tmp;
    while (i<=j)
    {
        while (d[i].tmp<g) i++;
        while (g<d[j].tmp) j--;
        if (i<=j)
            swap(d[i], d[j]),
            i++, j--;
    }
    sort(fi, j); sort(i, la);
}

bool find(int x, int cur)
{
    if (pos[cur]>d[x].t) return false;
    if (lnk[cur]==0)
    {
        lnk[cur]=x;
        return true;
    }
    if (find(d[x].t<d[lnk[cur]].t?lnk[cur]:x, cur+1))
    {
        lnk[cur]=(d[x].t<d[lnk[cur]].t?x:lnk[cur]);
        return true;
    }
    return false;
}

int main()
```

```
{
    freopen("income.in", "r", stdin);
    freopen("income.out", "w", stdout);

    scanf("%d", &n);
    for (int i=1; i<=n; i++)
        scanf("%d%d%d", &d[i].s, &d[i].t, &d[i].v);
    for (int i=1; i<=n; i++) d[i].tmp=d[i].s;
    sort(1, n);
    for (int i=1, cur=0; i<=n; i++, cur++)
        pos[i]=cur>?=d[i].s;
    for (int i=1; i<=n; i++) d[i].tmp=-d[i].v;
    sort(1, n);

    memset(lnk, 0, sizeof(lnk));
    long long ans=0;
    for (int i=1; i<=n; i++)
    {
        int cur=1;
        while (pos[cur]<d[i].s) cur++;
        if (find(i, cur))
            ans+=d[i].v;
    }
    printf("%I64d\n", ans);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

算法七优化后的程序实现

```
// For IOI2010.
// Problem Income, Optimization of Algorithm VII.
// Written by Feng Qiwei in December, 2009.

#include <iostream>
#include <cstring>
```

```
using namespace std;

const int maxn=5010;

struct item
{
    int s, t, v, tmp;
} d[maxn];
int pos[maxn], lnk[maxn];
int stx[maxn], sty[maxn];
bool lab[maxn];
int n;

void sort(int fi, int la)
{
    if (fi>=la) return;
    int i=fi, j=la, g=d[(i+j)>>1].tmp;
    while (i<=j)
    {
        while (d[i].tmp<g) i++;
        while (g<d[j].tmp) j--;
        if (i<=j) swap(d[i++], d[j--]);
    }
    sort(fi, j); sort(i, la);
}

int main()
{
    freopen("income.in", "r", stdin);
    freopen("income.out", "w", stdout);

    scanf("%d", &n);
    for (int i=1; i<=n; i++)
        scanf("%d%d%d", &d[i].s, &d[i].t, &d[i].v);
    for (int i=1; i<=n; i++) d[i].tmp=d[i].s;
    sort(1, n);
    for (int i=1, cur=0; i<=n; i++, cur++)
        pos[i]=cur>?=d[i].s;
    for (int i=1; i<=n; i++) d[i].tmp=-d[i].v;
    sort(1, n);
```

```
    memset(lnk, 0, sizeof(lnk));
    memset(lab, true, sizeof(lab));
    long long ans=0;
    for (int k=1; k<=n; k++)
    {
        int cur=1;
        while (pos[cur]<d[k].s) cur++;
        int num=0, x=k;
        bool flag=true;
        while (1)
        {
            while (cur<=n && pos[cur]<=d[x].t && lnk[cur]>0 &&
d[lnk[cur]].t<=d[x].t)
                cur++;
            if (cur>n || pos[cur]>d[x].t || !lab[cur])
            {
                flag=false;
                break;
            }
            num++; stx[num]=x; sty[num]=cur;
            if (lnk[cur]==0)
                break;
            x=lnk[cur];
        }
        if (flag)
        {
            for (int i=1; i<=num; i++)
                lnk[sty[i]]=stx[i];
            ans+=d[k].v;
        }
        else
            for (int i=1; i<=num; i++)
                lab[sty[i]]=false;
    }
    printf("%I64d\n", ans);
    fclose(stdin);
    fclose(stdout);
    return 0;
}
```

感谢

感谢老师多年的指导，感谢中山一中提供的训练环境。

感谢唐文斌和胡伟栋教练的指导。

感谢中山一中的师兄学长们的指导与帮助。

感谢广州二中的邓原同学提供的出题灵感以及对我解题的帮助。

感谢朝阳实验学校的郭晓旭同学和汕头金山中学的李新野同学在讨论中给我的帮助与启示。

感谢中山市第一中学初中部的龙浩民同学为我提供了算法一的程序。

感谢Internet提供的资源。

感谢父母，感谢多年来所有关心、帮助过我的人。

最后感谢伟大的社会主义国家为我们提供了这样一个和平、宁静的生活学习环境。