区间调度——动态规划

原创 李然 LOA算法学习笔记 2021-01-17 23:47

01 问题描述

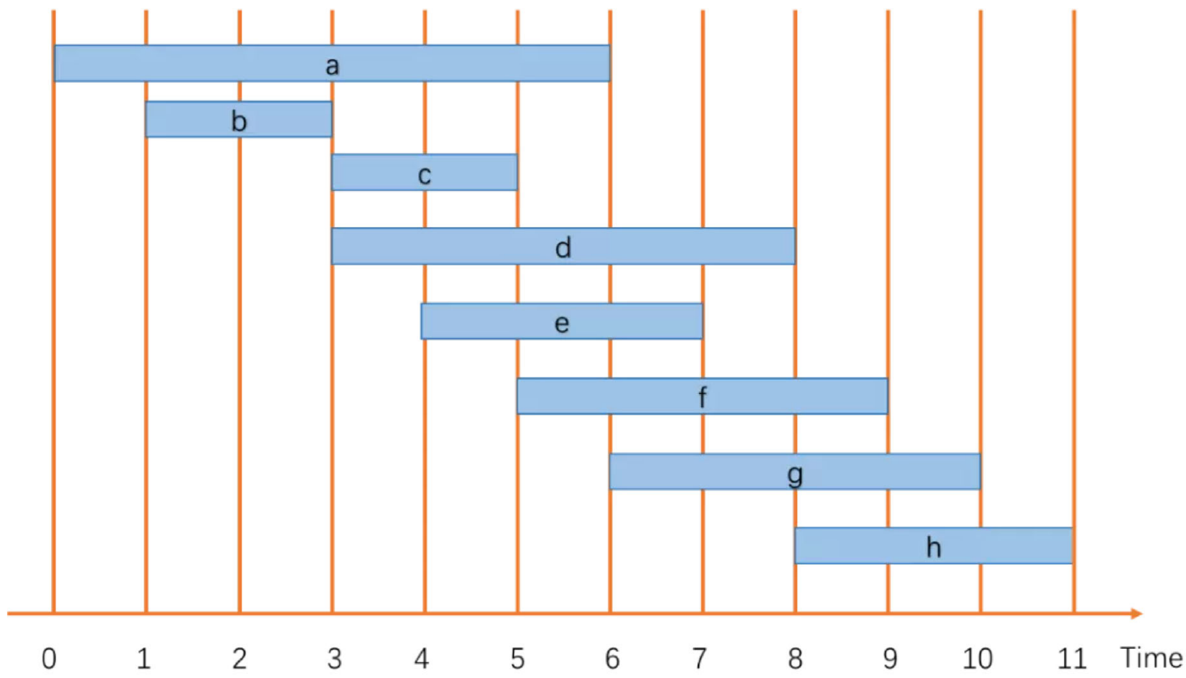
给定n个工作的开始时间、结束时间和产生的效益，求出能够获得最大效益的工作安排，前提是满足安排的工作调度是不冲突的。

	Start	End	Yield
Work1	0	3	4
Work2	5	6	5
Work3	2	8	10

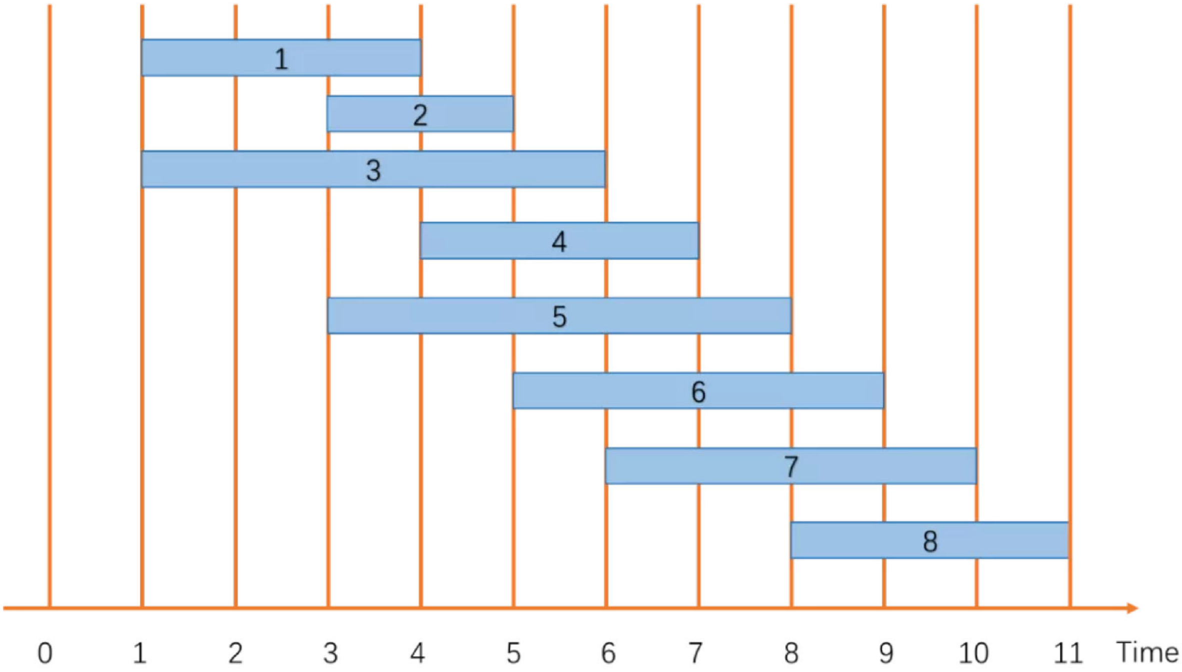
例如：

按照不考虑效益权重的工作调度，正如算法课讲过的贪心思想，依次结束时间最早且相容的工作，这里选出{1,2}，其实现的最大效益为9。但显而易见，10才是最大效益，因此最早结束时间的贪心策略在带权重（或者效益）的区间调度问题里已不适用。

02 问题求解思路



上图为若干工作安排，其求解思路是按照每个工作的结束时间来进行升序排序。该问题核心：找到一个彼此兼容且权重最大的任务子集。



设dp[i]表示对前i个工作结束后所能达到的最大效益；f[i]表示序列中前一个与之不冲突的最大工作编号，如下图示，f[h]=5，f[6]=2。

dp[i]不选择工作i，那么它一定是前i-1个工作的最优解，即；dp[i-1]；

若选择了工作i，那么它一定是前f[i]个工作的最优解加v[i]。

考虑状态转移方程，对于每个工作i，比较dp[i-1]和dp[f[i]]+v[i]的大小，务必牢记f[i]表示前一个与之不冲突的最大工作编号。计算出来所有的dp[i]之后，自顶而下，求出所有使权值之和变为最大的需求。

$$dp[i] = \begin{cases} 0 & i=0 \\ \max(dp[i-1], dp[f[i]] + v[i]) & i>0 \end{cases}$$

03 时间复杂度分析

自顶而下的递归算法是指数时间的，因为会重复计算子问题，不带记忆性，最坏计算dp[i]复杂度为1.618^n（介于$2^{(n/2)}$ ~ 2^n)

1）最慢规模减少策略考虑，f[i]每次减少2，即f[i]=i-2，如果每次f[i]每次只减少1，则f[i]=i-1，进而d[i-1]=dp[f[i]]，那么一定有dp[i-1]≤dp[f[i]]+v[i]，这样一半的分支就不用继续递归下去了，会减掉一半的计算量，两个分支将变成1个分支，反而规模减少更快。

2) 因此最坏情况是按规模1步长减少(不变形式分支)和按规模2步长减少(可变形式分支)的混合结果，因此是两种满二叉树的中间值，即介于$2^{(n/2)}$ ~ 2^n两个满二叉树的层数不同(一个层数为n/2,一个层数n)，规模减少速度的不同。

自底向上带记忆性的算法时间复杂度为O(nlogn)。

1) 根据完成时间排序所需时间：归并排序O(nlogn)

2) 对于每个i，计算f[i]：二分查找O(nlogn)

3) 计算dp[i]：每次计算得到一个值，一共n个，O(n)

04 算法设计

```cpp
#include <iostream>
#include <algorithm>
#include <cstring>

using namespace std;
const int maxn = 205;
struct JOB{
    int s, e, v, index;
    JOB(int s=0, int e=0, int v=0, int index=0):s(s), e(e), v(v), index(index){}
}job[maxn];
int frt[maxn], dp[maxn];

bool cmpe(JOB a, JOB b){ //定义按结束时间升序排序的排序规则
    return a.e<b.e;
}
/*bool cmps(JOB a, JOB b){ //定义按开始时间升序排序的排序规则
    return a.s<b.s;
}*/

int main()
{
    int n; cin >> n;
    //JOB a=(1,2,3);
    memset(frt, 0, sizeof(frt));
    memset(dp, 0, sizeof(dp));
    for(int i=1;i<=n;i++) cin >> job[i].s >> job[i].e >> job[i].v;
    //按结束时间升序排序，并为元素编号
    sort(job, job+n, cmpe);
    for(int i=1; i<=n; i++) job[i].index=i;
    //求frt[]数组
    //sort(job, job+n, cmps); //按开始时间升序排列
    for(int i=n;i>0;i--){
        for(int j=n-1;j>0;j--){
            if(job[j].e<=job[i].s) {frt[i]=job[j].index; break;}
        }
    }
    //sort(job, job+n, cmpe);
    for(int i=1;i<=n;i++){
        dp[i] = max(dp[i-1], dp[frt[i]]+job[i].v);
    }
    cout << dp[n];
    return 0;
}
```

05 作者感悟

动态规划的实质就是把大问题拆成许多个小问题，通过寻找大问题和小问题之间的递推关系，解决每个小问题，最终达到解决原问题的结果。为了进一步减少时间复杂度，动态规划可以把所有已经解决的子问题答案记忆下来，在新问题里需要用到的子问题可以直接提取，避免了重复计算，从而节约了时间。

LOA公众号关闭通知

LOA算法学习笔记

以习近平同志为核心的党中央关心学校思想政治工作纪实

中国文明网

三重跨界，真实为核，《那时的你》如何跨时空书写青春与价值对话？

电视评论