高级算法设计与分析作业4
动态规划

傅彦璋 23S003008

2024.5.1

1

排除题目歧义：“合并”两个数是指用两个数之和代替原本的两个数。

题目中的“代价”一词不符合直觉，以下用“收益”代之。

设$f(i,j)$表示将$a_i, a_{i+1}, ..., a_{j-1}, a_j$这$(j-i+1)$个数合并为一个数能产生的最大收益。设$s(i,j) = a_i + a_{i+1} + ... + a_{j-1} + a_j$。显然$f(i,i) = 0$。

我们有状态转移方程：

$$f(i,j) = \max_k \{f(i,k) + f(k+1,j) + s(i,j)\}$$

其中整数k满足$i \leq k < j$。

为了描述合并方案，我们用$dvd(i,j)$表示$a_i, ..., a_j$的最大收益合并方案如何得到：

$$dvd(i,j) = \text{argmax}_k \{f(i,k) + f(k+1,j) + s(i,j)\}$$

即

$$f(i,j) = f(i, dvd(i,j)) + f(dvd(i,j)+1, j) + s(i,j)$$

于是，我们有动态规划算法Algorithm1。

Algorithm 1: 求合并方案

Data: $a_1, ..., a_n$

Result: $dvd(i,j), f(i,j),\ 1 \le i < j \le n$

```
1  初始化f(i,i) = 0, 1 ≤ i ≤ n;
2  for len = 1 to n do
3      for (i,j) where j − i + 1 = len do
4          for k from i to (j − 1) do
5              cur ← f(i,k) + f(k+1,j) + s(i,j) ;
6              if cur bigger than f(i,j) then
7                  f(i,j) ← cur ;
8                  dvd(i,j) ← k ;
9              end
10         end
11     end
12 end
```

算法计算f值和dvd值。$f(1,n)$即为最大收益，最大收益的合并方案可由dvd描述。

算法第5行的$s(i,j)$可以通过预处理前缀和求得，预处理的时间是线性的。整个算法的时间复杂度属于$O(n^3)$，由于需要存储二维函数f和dvd，空间消耗是$O(n^2)$的。

2

(1)

定义函数$big : \{1,...,n\} \times \{1,...,n\} \to \mathcal{Z} \times \{1,...,n\} \times \{0,1\} \times \{0,1\} \times \{+,-,\times\}$. 对$1 \le i \le j \le n$，$big(i,j) = (val, k, type1, type2, op)$表示：将$a_i...,a_j$整合为一个表达式，表达式最大值能达到$val$，这个值是通过将$a_i,...,a_k$组合成的表达式(称左子表达式)和$a_{k+1},...,a_j$组合成的表达式（称右子表达式）进行$op$操作得到的。$type1 = 1$表示左子表达式是通过最大化得到的，$type1 = 0$则表示左子表达式是通过最小化得到的。$type2 = 1$表示右子表达式是通过最大化得到的，$type2 = 0$则表示右子表达式是通过最小化得到的。

定义函数$sml : \{1,...,n\} \times \{1,...,n\} \to \mathcal{Z} \times \{1,...,n\} \times \{0,1\} \times \{0,1\} \times \{+,-,\times\}$. 对$1 \le i \le j \le n$，$sml(i,j) = (val, k, type1, type2, op)$表示：将$a_i...,a_j$整合为一个表达式，表达式最小值能达到$val$，这个值是通过将$a_i,...,a_k$组合成的表达式(称左子表达式)和$a_{k+1},...,a_j$组合成的表达式（称右子表达式）进行$op$操作得到的。$type1 = 1$表示左子表达式是通过最大化得到的，$type1 = 0$则表示左子表达式是通过最小化得到的。$type2 = 1$表示右子表达式是通过最大化得到的，$type2 = 0$则表示右子表达式是通过最小化得到的。

big和sml的函数值都是五元组，我们用$big.val$表示big五元组中的第一个元素。sml类

似。五元组中后四个元素是为了求得最优值后输出表达式二设置的辅助元素。

显然，我们有初始条件$big(i,i).val = sml(i,i).val = a_i$。

注意到这样的状态转移方程（$i \le k < j$）：

$$\begin{aligned} big(i,j).val = \max_k(\max(& \\ & big(i,k).val \times big(k+1,j).val, \\ & sml(i,k).val \times sml(k+1,j).val, \\ & big(i,k).val + big(k+1,j).val, \\ & big(i,k).val - sml(k+1,j).val \ \ )) \end{aligned}$$

$$\begin{aligned} sml(i,j).val = \min_k(\min(& \\ & sml(i,k).val - big(k+1,j).val, \\ & sml(i,k).val + sml(k+1,j).val, \\ & big(i,k).val \times sml(k+1,j).val, \\ & sml(i,k).val \times big(k+1,j).val \ \ )) \end{aligned}$$

五元组中后四个元素的取值，根据状态转移方程是可以自然地得到的。于是有求最大表达式的算法Algorithm2.

Algorithm 2: 求最大表达式

Data: $a_1, ..., a_n$

Result: $big(i,j), sml(i,j),\ 1 \le i < j \le n$

```
初始化big(i,i).val = sml(i,i).val = a_i, 1 ≤ i ≤ n;
for len = 1 to n do
    for (i,j) where j − i + 1 = len do
        for k from i to (j − 1) do
            由big(i,k), sml(i,k), big(k+1,j), sml(k+1,j)更新big(i,j)、sml(i,j) ;
        end
    end
end
```

整个表达式的最大值式算法给出的$big(1,n).val$.五元组$big(1,n)$描述了该最大值是如何得到的，故可根据五元组递归地写出带括号的表达式，递归步骤此处略。递归过程最坏的时间复杂度是$O(n)$。

算法的时间复杂度是$O(n^3)$的。由于要存储二维函数big和sml，算法的空间消耗是$O(n^2)$的。

(2)

与(1)类似，定义函数$big(i,j), sml(i,j), pos(i,j), neg(i,j)$分别表示$a_i$到$a_j$整合为一个表达式能得到的最大值、最小值、绝对值最小正值、绝对值最小负值。

有状态转移方程

$$
\begin{aligned}
big(i,j).val = \max_k(\max(&\\
&big(i,k).val \times big(k+1,j).val,\\
&sml(i,k).val \times sml(k+1,j).val,\\
&big(i,k).val + big(k+1,j).val,\\
&big(i,k).val - sml(k+1,j).val,\\
&big(i,k).val/pos(k+1,j).val,\\
&sml(i,k).val/neg(k+1,j).val \quad))
\end{aligned}
$$

$$
\begin{aligned}
sml(i,j).val = \min_k(\min(&\\
&sml(i,k).val - big(k+1,j).val,\\
&sml(i,k).val + sml(k+1,j).val,\\
&big(i,k).val \times sml(k+1,j).val,\\
&sml(i,k).val \times big(k+1,j).val,\\
&sml(i,k).val/pos(k+1,j).val,\\
&big(i,k).val/neg(k+1,j).val \quad))
\end{aligned}
$$

$$
\begin{aligned}
pos(i,j).val = \underset{k}{best}(best(&\\
&pos(i,k).val \pm pos(k+1,j).val,\\
&pos(i,k).val \pm neg(k+1,j).val,\\
&neg(i,k).val + pos(k+1,j).val,\\
&neg(i,k).val - neg(k+1,j).val,\\
&pos(i,k).val/big(k+1,j).val,\\
&neg(i,k).val/sml(k+1,j).val \quad))
\end{aligned}
$$

$$
\begin{aligned}
neg(i,j).val = \underset{k}{best}(best(\ & \\
& pos(i,k).val - pos(k+1,j).val, \\
& pos(i,k).val + neg(k+1,j).val, \\
& neg(i,k).val \pm pos(k+1,j).val, \\
& neg(i,k).val \pm neg(k+1,j).val, \\
& pos(i,k).val/sml(k+1,j).val, \\
& neg(i,k).val/big(k+1,j).val\ \ ))
\end{aligned}
$$

于是可以用类似的算法完成(1)的任务。

3

设$f(i,j)$表示走到位置(i,j)的最小代价。显然$f(0,0)=0$。

我们约定$f(-1,j)=f(i,-1)=a_{i(-1)}=a_{(-1)j}=+\infty$，则有状态转移方程：

$$f(i,j)=min\{f(i,j-1)+a_{i(j-1)},\ f(i-1,j)+b_{(i-1)j}\}$$

为了描述旅行路线，我们用$last(i,j)$表示到达(i,j)位置的最优路径中上一步的位置.$last(i,j)$只可能取$(i-1,j)$或$(i,j-1)$.

有如下动态规划算法：

Algorithm 3: 求旅行路线

Data: m,n,a,b

Result: $f(i,j),last(i,j),\ 1\le i\le m,1\le j\le n$

```
初始化f(i,j) ← +∞, 1 ≤ i ≤ m, 1 ≤ j ≤ n;
for i = 0 to m do
    for j = 0 to n do
        if f(i,j) + a_ij < f(i,j+1) then
            f(i,j+1) ← f(i,j) + a_ij ;
            last(i,j+1) ← (i,j) ;
        end
        if f(i,j) + b_ij < f(i+1,j) then
            f(i+1,j) ← f(i,j) + b_ij ;
            last(i+1,j) ← (i,j) ;
        end
    end
end
```

算法输出$f(m,n)$即是最优旅行路线的代价。最优旅行路线由$last$递归地表示。算法的时间复杂度是$O(mn)$的，空间消耗是$O(mn)$的。

4

首先考虑这样一个问题，一个带边权的有向无环图$G=(V,E)$，边(u,v)的权值记作$c(u,v)$。图中有源点s满足$\forall v \in V,(s,v) \in E$，有汇点$t$满足$\forall u \in V,(u,t) \in E$。有如下时间代价$O(|E|)$的动态规划算法Algorithm4可求从$s$到$t$的最长路径。

Algorithm 4: 求有源汇点DAG的最长路径

```
Data: G, s, t
Result: s到t的最长路径长度f(t)
初始化f(i) ← 0, i ∈ V; A ← {s} ;
for u ∈ A do
    for v where (u, v) ∈ E do
        if f(u) + c(u, v) > f(v) then
            f(v) = f(u) + c(u, v) ;
        end
        if ∀(w, v) ∈ E, w ∈ A then
            A = A ∪ {v}
        end
    end
end
输出f(t)
```

现在考虑将叠箱子的问题转化为在DAG中求最长路径的问题。我们用如下方法构造一个DAG：

将这个DAG表示为$G=(V,E)$。其中$|V|=n+2$。我们将节点编号为$0,1,...,n+1$.$c(u,v)$表示编号为u的节点到编号为v的节点之间的边权。

E初始为空，我们对边进行如下构造。如果第u个箱子可以放在第v个箱子上面，则在编号为u的节点和编号为v的节点之间构造一条边(u,v)，其权值$c(u,v)=h_v$，即第v个箱子的高度。编号为0的节点为源点，其向所有其他点连边，$c(0,v)=h_v,\ v=1,2,...,n$。编号为$n+1$的节点为汇点，其他所有点向其连边，$c(u,n+1)=0,u=0,1,...,n$。

从节点0（源点）到节点$n+1$（汇点）的最长路径即为箱子堆叠的最大高度，可使用Algorithm4求解。构造图G的复杂度是$O(n^2)$的。根据边的构造，我们知道，$|E|$是$O(n^2)$的。所以，总的求解时间代价是$O(n^2)$的。