动态规划

对于动态规划，每个刚接触的人都需要一段时间来理解，特别是第一次接触的时候总是想不通为什么这种方法可行，这篇文章就是为了帮助大家理解动态规划，并通过讲解基本的 01 背包问题来引导读者如何去思考动态规划。本文力求通俗易懂，无异性，不让读者感到迷惑，引导读者去思考，所以如果你在阅读中发现有不通顺的地方，让你产生错误理解的地方，让你难得读懂的地方，请跟贴指出，谢谢！

----第一节----初识动态规划--------

经典的 01 背包问题是这样的：

有一个包和 n 个物品，包的容量为 m，每个物品都有各自的体积和价值，问当从这 n 个物品中选择多个物品放在包里而物品体积总数不超过包的容量 m 时，能够得到的最大价值是多少？[对于每个物品不可以取多次，最多只能取一次，之所以叫做 01 背包，0 表示不取，1 表示取]

为了用一种生动又更形象的方式来讲解此题，我把此题用另一种方式来描述，如下：

有一个国家，所有的国民都非常老实憨厚，某天他们在自己的国家发现了十座金矿，并且这十座金矿在地图上排成一条直线，国王知道这个消息后非常高兴，他希望能够把这些金子都挖出来造福国民，首先他把这些金矿按照在地图上的位置从西至东进行编号，依次为 0、1、2、3、4、5、6、7、8、9，然后他命令他的手下去对每一座金矿进行勘测，以便知道挖取每一座金矿需要多少人力以及每座金矿能够挖出多少金子，然后动员国民都来挖金子。

题目补充 1：挖每一座金矿需要的人数是固定的，多一个人少一个人都不行。国王知道每个金矿各需要多少人手，金矿 i 需要的人数为 peopleNeeded[i]。

题目补充 2：每一座金矿所挖出来的金子数是固定的，当第 i 座金矿有 peopleNeeded[i] 人去挖的话，就一定能恰好挖出 gold[i]个金子。否则一个金子都挖不出来。

题目补充 3：开采一座金矿的人完成开采工作后，他们不会再次去开采其它金矿，因此一个人最多只能使用一次。

题目补充 4：国王在全国范围内仅招募到了 10000 名愿意为了国家去挖金子的人，因此这些人可能不够把所有的金子都挖出来，但是国王希望挖到的金子越多越好。

题目补充 5：这个国家的每一个人都很老实（包括国王），不会私吞任何金子，也不会弄虚作假，不会说谎话。

题目补充 6：有很多人拿到这个题后的第一反应就是对每一个金矿求出平均每个人能挖出多少金子，然后从高到低进行选择，这里要强调这种方法是错的，如果你也是这样想的，请考虑背包模型，当有一个背包的容量为 10，共有 3 个物品，体积分别是 3、3、5，价值分别是 6、6、9，那么你的方法取到的是前两个物品，总价值是 12，但明显最大值是后两个物品组成的 15。

题目补充 7：我们只需要知道最多可以挖出多少金子即可，而不用关心哪些金矿挖哪些金矿不挖。

那么，国王究竟如何知道在只有 10000 个人的情况下最多能挖出多少金子呢？国王是如何思考这个问题的呢？

国王首先来到了第 9 个金矿的所在地（注意，第 9 个就是最后一个，因为是从 0 开始编号的，最西边的那个金矿是第 0 个），他的臣子告诉他，如果要挖取第 9 个金矿的话就需要 1500 个人，并且第 9 个金矿可以挖出 8888 个金子。听到这里国王哈哈大笑起来，因为原先他以为要知道十个金矿在仅有 10000 个人的情况下最多能挖出多少金子是一件很难思考的问题，但是，就在刚才听完他的臣子所说的那句话时，国王已经知道总共最多能挖出多少金子了，国王是如何在不了解其它金矿的情况下知道最多能挖出多少金子的呢？他的臣子们也不知道这个谜，因此他的臣子们就问他了：“最聪明的国王陛下，我们都没有告诉您其它金矿的情况，您是如何知道最终答案的呢？”

得意的国王笑了笑，然后把他最得意的“左、右手”叫到跟前，说到：“我并不需要

考虑最终要挖哪些金矿才能得到最多的金子，我只需要考虑我面前的这座金矿就可以了，对于我面前的这座金矿不外乎仅有两种选择，要么挖，要么不挖，对吧？”

“当然，当然”大臣们回答倒。

国王继续说道：“如果我挖取第 9 座金矿的话那么我现在就能获得 8888 个金子，而我将用去 1500 个人，那么我还剩下 8500 个人。我亲爱的左部下，如果你告诉我当我把所有剩下的 8500 个人和所有剩下的其它金矿都交给你去开采你最多能给我挖出多少金子的话，那么我不就知道了在第 9 个金矿一定开采的情况下所能得到的最大金币数吗？”

国王的左部下听后回答道：“国王陛下，您的意思是如果我能用 8500 个人在其它金矿最多开采出 x 个金币的话，那您一共就能够获得 x + 8888 个金子，对吗？”

“是啊，是啊……如果第 9 座金矿一定开采的话……”大臣们点头说到。

国王笑着继续对着他的右部下说到：“亲爱的右部下，也许我并不打算开采这第 9 座金矿，那么我依然拥有 10000 个人，如果我把这 10000 个人和剩下的金矿都给你的话，你最多能给我挖出多少个金子呢？”

国王的右部下聪明地说道：“尊敬的国王陛下，我明白您的意思了，如果我回答最多能购开采出 y 个金币的话，那您就可以在 y 和 x+8888 之间选择一个较大者，而这个较大者就是最终我们能获得的最大金币数，您看我这样理解对吗？”

国王笑得更灿烂了，问他的左部下：“那么亲爱的左部下，我给你 8500 个人和其余金矿的话你能告诉我最多能挖出多少金子吗？”

“请您放心，这个问题难不倒我”。左部下向国王打包票说到。

国王高兴地继续问他的右部下：“那右部下你呢，如果我给你 10000 个人和其余金矿的话你能告诉我最多能挖出多少金子吗？”

“当然能了！交给我吧！”右部下同左部下一样自信地回答道。

“那就拜托给你们两位了，现在我要回到我那舒适的王宫里去享受了，我期待着你们的答复。”国王说完就开始动身回去等消息了，他是多么地相信他的两个大臣能够给他一个准确的答复，因为国王其实知道他的两位大臣要比他聪明得多。

故事发展到这里，你是否在想国王的这两个大臣又是如何找到让国王满意的答案的呢？他们为什么能够如此自信呢？事实上他们的确比国王要聪明一些，因为他们从国王的身上学到了一点，就是这一点让他们充满了自信。

国王走后，国王的左、右部下来到了第 8 座金矿，早已在那里等待他们的金矿勘测兵向两位大臣报道：“聪明的两位大臣，您们好，第 8 座金矿需要 1000 个人才能开采，可以获得 7000 个金子”。

因为国王仅给他的左部下 8500 个人，所以国王的左部下叫来了两个人，对着其中一个人问到：“如果我给你 7500 个人和除了第 8、第 9 的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？”

然后国王的左部下继续问另一个人：“如果我给你 8500 个人和除了第 8、第 9 的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？”

国王的左部下在心里想着：“如果他们俩都能回答我的问题的话，那国王交给我的问

题不就解决了吗？哈哈哈！”

因为国王给了他的右部下 10000 个人，所以国王的右部下同样也叫来了两个人，对着其中一个人问：“如果我给你 9000 个人和除了第 8、第 9 的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？”

然后国王的右部下继续问他叫来的另一个人：“如果我给你 10000 个人和除了第 8、第 9 的其它所有金矿的话，你能告诉我你最多能挖出多少金子吗？”

此时，国王的右部下同左部下一样，他们都在为自己如此聪明而感到满足。

当然，这四个被叫来的人同样自信地回答没有问题，因为他们同样地从这两个大臣身上学到了相同的一点，而两位自认为自己一样很聪明的大臣得意地笑着回到了他们的府邸，等着别人回答他们提出来的问题，现在你知道了这两个大臣是如何解决国王交待给他们的问题了吗？

那么你认为被大臣叫去的那四个人又是怎么完成大臣交给他们的问题的呢？答案当然是他们找到了另外八个人！

没用多少功夫，这个问题已经在全国传开了，更多人的人找到了更更多的人来解决这个问题，而有些人却不需要去另外找两个人帮他，哪些人不需要别人的帮助就可以回答他们的问题呢？

很明显，当被问到给你 z 个人和仅有第 0 座金矿时最多能挖出多少金子时，就不需要别人的帮助，因为你知道，如果 z 大于等于挖取第 0 座金矿所需要的人数的话，那么挖出来的最多金子数就是第 0 座金矿能够挖出来的金子数，如果这 z 个人不够开采第 0 座金矿，那么能挖出来的最多金子数就是 0，因为这唯一的金矿不够人力去开采。让我们为这些不需要别人的帮助就可以准确地得出答案的人们鼓掌吧，这就是传说中的底层劳动人民！

故事讲到这里先暂停一下，我们现在重新来分析一下这个故事，让我们对动态规划有个理性认识。

子问题：

国王需要根据两个大臣的答案以及第 9 座金矿的信息才能判断出最多能够开采出多少金子。为了解决自己面临的问题，他需要给别人制造另外两个问题，这两个问题就是子问题。

思考动态规划的第一点----最优子结构：

国王相信，只要他的两个大臣能够回答出正确的答案（对于考虑能够开采出的金子数，最多的也就是最优的同时也就是正确的），再加上他的聪明的判断就一定能得到最终的正确答案。我们把这种子问题最优时母问题通过优化选择后一定最优的情况叫做“最优子结构”。

思考动态规划的第二点----子问题重叠：

实际上国王也好，大臣也好，所有人面对的都是同样的问题，即给你一定数量的人，给你一定数量的金矿，让你求出能够开采出来的最多金子数。我们把这种母问题与子问题本质上是同一个问题的情况称为“子问题重叠”。然而问题中出现的不同点往往就是被子问题之间传递的参数，比如这里的人数和金矿数。

思考动态规划的第三点----边界：

想想如果不存在前面我们提到的那些底层劳动者的话这个问题能解决吗？永远都不可能！我们把这种子问题在一定时候就不再需要提出子子问题的情况叫做边界，没有边界就会出现死循环。

思考动态规划的第四点----子问题独立：

要知道，当国王的两个大臣在思考他们自己的问题时他们是不会关心对方是如何计

算怎样开采金矿的，因为他们知道，国王只会选择两个人中的一个作为最后方案，另一个人的方案并不会得到实施，因此一个人的决定对另一个人的决定是没有影响的。我们把这种一个母问题在对子问题选择时，当前被选择的子问题两两互不影响的情况叫做“子问题独立”。

这就是动态规划，具有“最优子结构”、“子问题重叠”、“边界”和“子问题独立”，当你发现你正在思考的问题具备这四个性质的话，那么恭喜你，你基本上已经找到了动态规划的方法。

有了上面的这几点，我们就可以写出动态规划的转移方程式，现在我们来写出对应这个问题的方程式，如果用 gold[mineNum]表示第 mineNum 个金矿能够挖出的金子数，用 peopleNeeded[mineNum]表示挖第 mineNum 个金矿需要的人数，用函数 f(people,mineNum)表示当有 people 个人和编号为 0、1、2、3、……、mineNum 的金矿时能够得到的最大金子数的话，f(people,mineNum)等于什么呢？或者说 f(people,mineNum)的转移方程是怎样的呢？

答案是：

当 mineNum = 0 且 people >= peopleNeeded[mineNum] 时 f(people,mineNum) = gold[mineNum]

当 mineNum = 0 且 people < peopleNeeded[mineNum]时 f(people,mineNum) = 0

当 mineNum != 0 时 f(people,mineNum) = f(people-peopleNeeded[mineNum], mineNum-1) + gold[mineNum]与 f(people, mineNum-1)中的较大者，前两个式子对应动态规划的“边界”，后一个式子对应动态规划的“最优子结构”请读者弄明白后再继续往下看。

----第二节----动态规划的优点--------

现在我假设读者你已经搞清楚了为什么动态规划是正确的方法，但是我们为什么需要使用动态规划呢？请先继续欣赏这个故事：

国王得知他的两个手下使用了和他相同的方法去解决交代给他们的问题后，不但没有认为他的两个大臣在偷懒，反而很高兴，因为他知道，他的大臣必然会找更多的人一起解决这个问题，而更多的人会找更更多的人，这样他这个聪明的方法就会在不经意间流传开来，而全国人民都会知道这个聪明的方法是他们伟大的国王想出来的，你说国王能不高兴吗？

但是国王也有一些担忧，因为他实在不知道这个“工程”要动用到多少人来完成，如果帮助他解决这个问题的人太多的话那么就太劳民伤财了。“会不会影响到今年的收成呢？”国王在心里想着这个问题，于是他请来了整个国家里唯一的两个数学天才，一个叫做小天，另一个叫做小才。

国王问小天：“小天啊，我发觉这个问题有点严重，我知道其实这可以简单的看成一个组合问题，也就是从十个金矿中选取若干个金矿进行开采，看看哪种组合得到的金子最多，也许用组合方法会更好一些。你能告诉我一共有多少种组合情况吗？”

“国王陛下，如果用组合方法的话一共要考虑 2 的 10 次方种情况，也就是 1024 种情况。”小天思考了一会回答到。

“嗯……，如果每一种情况我交给一个人去计算能得到的金子数的话，那我也要 1024 个人，其实还是挺多的。”国王好像再次感觉到了自己的方法是正确的。

国王心理期待着小才能够给它一个更好的答案，问到：“小才啊，那么你能告诉我用我的那个方法总共需要多少人吗？其实，我也计算过，好像需要的人数是 1+2+4+8+16+32+64+……，毕竟每一个人的确都需要找另外两个人来帮助他们……”

不辜负国王的期待，小才微笑着说到：“亲爱的国王陛下，其实我们并不需要那么多人，因为有很多问题其实是相同的，而我们只需要为每一个不同的问题使用一个人力便可。”

国王高兴的问到：“此话如何讲？”

“打个比方，如果有一个人需要知道 1000 个人和 3 个金矿可以开采出多少金子，同时另一个人也需要知道 1000 个人和 3 个金矿可以开采出多少金子的话，那么他们可以去询问相同的一个人，而不用各自找不同的人浪费人力了。”

国王思考着说到：“嗯，很有道理，如果问题是一样的话那么就不需要去询问两个不同的人了，也就是说一个不同的问题仅需要一个人力，那么一共有多少个不同的问题呢？”

“因为每个问题的人数可以从 0 取到 10000，而金矿数可以从 0 取到 10，所以最多大约有 10000 * 10 等于 100000 个不同的问题。” 小才一边算着一边回答。

“什么？十万个问题？十万个人力？”国王有点失望。

“请国王放心，事实上我们需要的人力远远小于这个数的，因为不是每一个问题都会遇到，也许我们仅需要一、两百个人力就可以解决这个问题了，这主要和各个金矿所需要的人数有关。” 小才立刻回答到。

故事的最后，自然是国王再一次向他的臣民们证明了他是这个国家里最聪明的人，现在我们通过故事的第二部分来考虑动态规划的另外两个思考点。

思考动态规划的第五点----做备忘录：

正如上面所说的一样，当我们遇到相同的问题时，我们可以问同一个人。讲的通俗一点就是，我们可以把问题的解放在一个变量中，如果再次遇到这个问题就直接从变量中获得答案，因此每一个问题仅会计算一遍，如果不做备忘的话，动态规划就没有任何优势可言了。

思考动态规划的第六点----时间分析：

正如上面所说，如果我们用穷举的方法，至少需要 2^n 个常数时间，因为总共有 2^n 种情况需要考虑，如果在背包问题中，包的容量为 1000，物品数为 100，那么需要考虑 2^100 种情况,这个数大约为 10 的 30 次方。

而如果用动态规划，最多大概只有 1000*100 = 100000 个不同的问题，这和 10 的 30 次方比起来优势是很明显的。而实际情况并不会出现那么多不同的问题，比如在金矿模型中，如果所有的金矿所需人口都是 1000 个人，那么问题总数大约只有 100 个。

非正式地，我们可以很容易得到动态规划所需时间，如果共有 questionCount 个相同的子问题，而每一个问题需要面对 chooseCount 种选择时，我们所需时间就为 questionCount * chooseCount 个常数。在金矿模型中，子问题最多有大概 people * n 个(其中 people 是用于开采金矿的总人数，n 是金矿的总数)，因此 questionCount = people * n，而就像国王需要考虑是采用左部下的结果还是采用右部下的结果一样，每个问题面对两个选择，因此 chooseCount = 2,所以程序运行时间为 T = O(questionCount * chooseCount) =O(people * n)，别忘了实际上需要的时间小于这个值，根据所遇到的具体情况有所不同。

这就是动态规划的魔力，它减少了大量的计算，因此我们需要动态规划！

----第三节----动态规划的思考角度----------

那么什么是动态规划呢？我个人觉得，如果一个解决问题的方法满足上面六个思考点中的前四个，那么这个方法就属于动态规划。而在思考动态规划方法时，后两点同样也是需要考虑的。

面对问题要寻找动态规划的方法，首先要清楚一点，动态规划不是算法，它是一种方法，它是在一件事情发生的过程中寻找最优值的方法，因此，我们需要对这件事情所发生的过程进行考虑。而通常我们从过程的最后一步开始考虑，而不是先考虑过程的开始。

打个比方，上面的挖金矿问题，我们可以认为整个开采过程是从西至东进行开采的（也就是从第 0 座开始），那么总有面对最后一座金矿的时候（第 9 座），对这座金矿不外乎两个选择，开采与不开采，在最后一步确定时再去确定倒数第二步，直到考虑第 0 座金矿（过程的开始）。

而过程的开始，也就是考虑的最后一步，就是边界。

因此在遇到一个问题想用动态规划的方法去解决时，不妨先思考一下这个过程是怎样的，然后考虑过程的最后一步是如何选择的，通常我们需要自己去构造一个过程，比如后面的练习。

----第四节----总结-------

那么遇到问题如何用动态规划去解决呢？根据上面的分析我们可以按照下面的步骤去考虑：

1、构造问题所对应的过程。

2、思考过程的最后一个步骤，看看有哪些选择情况。

3、找到最后一步的子问题，确保符合“子问题重叠”，把子问题中不相同的地方设置为参数。

4、使得子问题符合“最优子结构”。

5、找到边界，考虑边界的各种处理方式。

6、确保满足“子问题独立”，一般而言，如果我们是在多个子问题中选择一个作为实施方案，而不会同时实施多个方案，那么子问题就是独立的。

7、考虑如何做备忘录。

8、分析所需时间是否满足要求。

9、写出转移方程式。

----第五节----练习-------

题目一：买书

有一书店引进了一套书，共有 3 卷，每卷书定价是 60 元，书店为了搞促销，推出一个活动，活动如下：

如果单独购买其中一卷，那么可以打 9.5 折。

如果同时购买两卷不同的，那么可以打 9 折。

如果同时购买三卷不同的，那么可以打 8.5 折。

如果小明希望购买第 1 卷 x 本，第 2 卷 y 本，第 3 卷 z 本，那么至少需要多少钱呢？（x、y、z 为三个已知整数）。

当然，这道题完全可以不用动态规划来解，但是现在我们是要学习动态规划，因此请想想如何用动态规划来做？

答案：

1、过程为一次一次的购买，每一次购买也许只买一本（这有三种方案），或者买两本（这也有三种方案），或者三本一起买（这有一种方案），最后直到买完所有需要的书。

2、最后一步我必然会在 7 种购买方案中选择一种，因此我要在 7 种购买方案中选择一个最佳情况。

3、子问题是，我选择了某个方案后，如何使得购买剩余的书能用最少的钱？并且这个选择不会使得剩余的书为负数。母问题和子问题都是给定三卷书的购买量，求最少需要用的钱，所以有“子问题重叠”，问题中三个购买量设置为参数，分别为 i、j、k。

4、的确符合。

5、边界是一次购买就可以买完所有的书，处理方式请读者自己考虑。

6、每次选择最多有 7 种方案，并且不会同时实施其中多种，因此方案的选择互不影响，所以有“子问题独立”。

7、我可以用 minMoney[i][j][k]来保存购买第 1 卷 i 本，第 2 卷 j 本，第 3 卷 k 本时所需的最少金钱。

8、共有 x * y * z 个问题，每个问题面对 7 种选择，时间为：O(x * y * z * 7) = O(x * y * z)。

9、用函数 MinMoney(i,j,k)来表示购买第 1 卷 i 本，第 2 卷 j 本，第 3 卷 k 本时所需的最少金钱，那么有：

MinMoney(i,j,k)=min(s1,s2,s3,s4,s5,s6,s7),其中 s1,s2,s3,s4,s5,s6,s7 分别为对应的 7 种方案使用的最少金钱：

s1 = 60 * 0.95 + MinMoney(i-1,j,k)

s2 = 60 * 0.95 + MinMoney(i,j-1,k)

s3 = 60 * 0.95 + MinMoney(i,j,k-1)

s4 = (60 + 60) * 0.9 + MinMoney(i-1,j-1,k)

s5 = (60 + 60) * 0.9 + MinMoney(i-1,j,k-1)

s6 = (60 + 60) * 0.9 + MinMoney(i-1,j,k-1)

s7 = (60 + 60 + 60) * 0.85 + MinMoney(i-1,j-1,k-1)

----第六节----代码参考------

下面提供金矿问题的程序源代码帮助读者理解，并提供测试数据给大家练习。

输入文件名为“beibao.in”，因为这个问题实际上就是背包问题，所以测试数据文件名就保留原名吧。

输入文件第一行有两个数，第一个是国王可用用来开采金矿的总人数，第二个是总共发现的金矿数。

输入文件的第 2 至 n+1 行每行有两个数，第 i 行的两个数分别表示第 i-1 个金矿需要的人数和可以得到的金子数。

输出文件仅一个整数，表示能够得到的最大金子数。

输入样例：

100 5

77 92

22 22

29 87

50 46

99 90

输出样例：

133

参考代码如下：

```
/*
=========程序信息========
对应题目：01 背包之金矿模型
使用语言：c++
使用编译器：Visual Studio 2005.NET
使用算法：动态规划
算法运行时间：O(people * n) [people 是人数，n 是金矿数]
作者：贵州大学 05 级  刘永辉
昵称：SDJL
编写时间：2008 年 8 月
联系 QQ：44561907
E-Mail：44561907@qq.com
获得更多文章请访问我的博客：www.cnblogs.com/sdjl
如果发现 BUG 或有写得不好的地方请发邮件告诉我:)
转载请保留此部分信息:)
*/

#include "stdafx.h"
#include <iostream>
#include <fstream>
using namespace std;

const int max_n = 100;//程序支持的最多金矿数
const int max_people = 10000;//程序支持的最多人数

int n;//金矿数
int peopleTotal;//可以用于挖金子的人数
int peopleNeed[max_n];//每座金矿需要的人数
int gold[max_n];//每座金矿能够挖出来的金子数
int maxGold[max_people][max_n];//maxGold[i][j]保存了 i 个人挖前 j 个金矿能够得到的最大金
子数，等于-1 时表示未知

//初始化数据
void init()
```

```
{
    ifstream inputFile("beibao.in");
    inputFile>>peopleTotal>>n;
    for(int i=0; i<n; i++)
        inputFile>>peopleNeed[i]>>gold[i];
    inputFile.close();

    for(int i=0; i<=peopleTotal; i++)
        for(int j=0; j<n; j++)
            maxGold[i][j] = -1;//等于-1 时表示未知 [对应动态规划中的“做备忘录”]

}

//获得在仅有 people 个人和前 mineNum 个金矿时能够得到的最大金子数，注意“前多少个"
也是从 0 开始编号的
int GetMaxGold(int people, int mineNum)
{
    //申明返回的最大金子数
    int retMaxGold;

    //如果这个问题曾经计算过  [对应动态规划中的“做备忘录”]
    if(maxGold[people][mineNum] != -1)
    {
        //获得保存起来的值
        retMaxGold = maxGold[people][mineNum];
    }
    else if(mineNum == 0)//如果仅有一个金矿时 [对应动态规划中的“边界”]
    {
        //当给出的人数足够开采这座金矿
        if(people >= peopleNeed[mineNum])
        {
            //得到的最大值就是这座金矿的金子数
            retMaxGold = gold[mineNum];
        }
        else//否则这唯一的一座金矿也不能开采
        {
            //得到的最大值为 0 个金子
            retMaxGold = 0;
        }
    }
    else if(people >= peopleNeed[mineNum])//如果给出的人够开采这座金矿 [对应动态规划
中的“最优子结构”]
    {
```

```
            //考虑开采与不开采两种情况，取最大值
            retMaxGold = max(GetMaxGold(people - peopleNeed[mineNum],mineNum -1) +
gold[mineNum],
                                                GetMaxGold(people,mineNum - 1));
        }
        else//否则给出的人不够开采这座金矿 [对应动态规划中的“最优子结构”]
        {
            //仅考虑不开采的情况
            retMaxGold  = GetMaxGold(people,mineNum - 1);
        }

        //做备忘录
        maxGold[people][mineNum] = retMaxGold;
        return retMaxGold;
    }

    int _tmain(int argc, _TCHAR* argv[])
    {
        //初始化数据
        init();
        //输出给定 peopleTotal 个人和 n 个金矿能够获得的最大金子数，再次提醒编号从 0 开始，
    所以最后一个金矿编号为 n-1
        cout<<GetMaxGold(peopleTotal,n-1);
        system("pause");
        return 0;
    }
```