创新意识和创新思维的培养是打造“金课”的核心内容——教学案例两则

王婷 刘任任

湘潭大学计算机学院, 湖南湘潭, 411105

摘 要 新工科新理念，呼唤课程建设中创新型人才培养模式的改革与探索。课堂教学是高等学校人才培养过程中的主要环节，它对于培养学生的掌握和发现知识的创新意识以及分析和解决问题的创新思维能力将起到重要的作用。主要介绍了在计算机类专业的学科基础课程的课堂教学中，创新意识和创新思维培养的探索和实践。

关键字 人才培养，金课，创新意识，创新思维，教学方法

The Core of Building "Golden Courses" is to Cultivate Innovative Consciousness and Innovative Thinking — Based on Two Teaching Cases

Wang Ting Liu Renren*

School of Computer Science & Schoole of Cyberspace Science XiangTan University
Xiangtan 411105, China

Abstract—The new concept of new engineering requires the reform and exploration of the training model for innovative talents in curriculum construction. As the main link in talent training at universities, classroom teaching plays an important role in cultivating students' innovative consciousness of mastering and discovering knowledge and innovative thinking of analyzing and solving problems. This paper mainly introduces the exploration and practice of cultivating innovative consciousness and innovative thinking in the classroom teaching of basic courses for computer majors.

Key words—talent training, golden course, innovative consciousness, innovative thinking, teaching methods

1 引 言

2018 年 6 月 21 日，教育部部长陈宝生在改革开放以来第一次新时代中国高等学校本科教育工作会议上提出了消灭“水课”、建设“金课”的概念。2018 年 11 月举行的“2018 高等教育国际论坛年会”上，教育部高等教育司司长吴岩表示：“要消灭‘水课’，打造有创新性、挑战度的‘金课’。” 吴岩司长所提出的“金课”标准是“两性一度”即高阶性、创新性、挑战度。所谓“高阶性”，就是知识能力素质的有机融合，是要培养学生解决复杂问题的综合能力和高级思维。[1]

所谓“创新性”，是指课程内容要反映前沿性和时代性，教学形式呈现先进性和互动性，学习结果具有探究性和个性化。所谓“挑战度”，是指课程有一定难度，需要跳一跳才能够得着，老师备课和学生课下有较高要求[2]。

在第十一届“中国大学教学论坛”上，教育部高等教育司司长吴岩表示,课程是人才培养的核心要素，是教育的微观问题，解决的却是战略大问题。课程是“立德树人成效”这一人才培养根本标准的具体化、操作化和目标化，也是当前中国大学带有普遍意义的短板、瓶颈和关键所在[3]。因此课程改革将在各个学科和专业发展中起到举足轻重的作用。

我们理解，以上所说“金课”标准中的“创新性”，是针对课程的教学内容和教学形式而言的，是手段。那么，通过课程教学，我们到底要达到什么目的呢？在多年的教学实践中，我们深刻体会到，应该将培养学生的“创新意识和思维”作为课程教学的主要目的。

2 计算机专业学科基础或专业骨干课程教学中如何培养学生的创新意识和思维

众所周知，“离散数学”是计算机科学的数学基础，国外也有人称其为“计算机数学”，计算机科学巨匠、图灵奖得主 D. E. Knuth 也曾说过，计算机科学就是算法的科学。

我们在讲授《离散数学》、《算法设计与分析》等课程时，始终在思考如何培养学生的创新意识和思维。

在要求学生掌握课程中的基本概念和基本定理后，我们再进一步启发学生，教材中的定理的证明，是否还有更简单的证法？已知的成熟算法还能否再改进？教材中的某些结论是否正确？这样做，收到了很好的教学效果。一是“逼得”学生首先必须看懂所涉及的内容，二是使学生逐步养成“创新”思维，具有“创新”意识。

2.1 案例 1

《图论》中的 Menger 定理是描述图的连通性与连接图中不同顶点的不相交通路数目之间关系的最基本的定理，该定理 1927 年由 Menger 提出。一般的图论著作都要将它作为重要的定理给出，并加以证明，其中大多数证明是引用 Dirac 的证明[4]-[5]，或者利用最大流最小割定理来证明[6]。这些证明虽然方法巧妙，富有启发性，但其证明都比较复杂，篇幅颇长，且引用的概念较多。我们就给学生留了一个课后作业，要求同学们在读懂该定理证明的基础上，给出一个相对比较简单的证明。以下是两个富有代表性的证明：

定理 (Menger, 1927) 分离图 G 中两个不邻接顶点 u 和 v 的顶点之最小数目等于互不相交的(u, v)-通路的最多数目。

证明之一：设分离G中u和v的顶点之最小数目为k，$S=\{w_1,w_2,\Lambda,w_k\}$是其分离顶点集合之一。显然，最多只存在$k$条互不相交的$(u,v)$-通路。下面证明一定存在$k$条互不相交的$(u,v)$-通路。

（反证法）假设G中最多只有$m(m<k)$条互不相交的(u,v)-通路。由于至少要k个顶点才能分离G中的顶点u和v，且$S=\{w_1,w_2,\Lambda,w_k\}$是其分离点集之一。于是，$S$中至少存在两个顶点$w_i$和$w_j(i\neq j)$，使得满足引理中所有仅含$w_i$而不含$S-\{w_i\}$中顶点的$(u,v)$-通路$\{P_{i1},P_{i2},\Lambda,P_{ir}\}$与所有仅含$w_j$而不含$S-\{w_j\}$中顶点的$(u,v)$-通路$\{P_{j1},P_{j2},\Lambda,P_{jh}\}$至少有一个公共顶点$t$，其中$t\in V-S$，且$t\neq u,v$。也即$t$是以下$(u,v)$-通路：$P_{i1},\mathrm{L},P_{ir},P_{j1},\mathrm{L},P_{jh}$的必经之顶点。

事实上，从引理可知，对每个$w_i\in S$，均有一条仅含w_i而不含$S-\{w_i\}$中顶点的(u,v)-通路$P_i,i=1,2,\Lambda,k$。若找不到上面所说的w_i和w_j，则这k条(u,v)-通路$\{P_1,P_2,\Lambda,P_k\}$中的任意两条P_i和P_j均无公共顶点$t\in V-S$，$t\neq u,v$，三条以上就更不存在这样的公共顶点t了。于是，P_1,P_2,Λ,P_k便是k条互不相交的(u,v)-通路了（注意：P_i仅含w_i而不含$S-\{w_i\}$中顶点）。这就与“只存在$m(m<k)$条互不相交的(u,v)-通路”的假设矛盾。

令$S'=S-\{w_i,w_j\}$。于是，子图$G-S'-\{t\}$中便不存在含w_i或w_j的(u,v)-通路了。从而$S''=S'\cup\{t\}$也将分离u和v。此时

$|S''|=|S'|+1=(|S|-2)+1=|S|-1<|S|$。这与$S$的假定矛盾。　证毕。

证明之二：设k是分离G中u和v的顶点之最小数目，$S=\{w_1,w_2,\Lambda,w_k\}$是其分离点集之一。显然，最多只存在$k$条互不相交的$(u,v)$-通路。下面证明一定存在$k$条互不相交的$(u,v)$-通路。

若不然，则由引理知，至少存在$w_i,w_j\in S,i\neq j$，使分别只含w_i和w_j的所有(u,v)-通路至少有一个公共顶点$t\in V-S$，$t\neq u,v$。显然，$G-S-\{t\}-\{w_i,w_j\}$中将不存在含w_i和w_j的(u,v)-通路了（注意t的特点）。令$S'=S\cup\{t\}-\{w_i,w_j\}$。于是，$S'$也将分离$u$和$v$而且$|S'|<|S|$。这就与$S$的假设矛盾。　证毕。

2.2 案例 2

排序操作在几乎所有的计算机应用中大量频繁使用。因此，自计算机诞生以来，人们一直在研究和改进排序算法。最简单又直观的排序方法要数选择排序、冒泡排序和插入排序，但它们的运行时间都是$O(n^2)$。为了提高排序的效率，人们提出了各种改进算法，其

中，1959 年由 D. L. Shell 提出的缩小增量法[7]（又称 Shell 排序算法）就是对插入排序的有效改进。Shell 排序算法的基本思想是，在排序过程中，每一遍对距离为 h 的那些元素实行直接插入排序，简称为 h -步排序，h 值逐步递减。最后，对所有元素进行 1-排序。在 h -步排序中，从第 $h+1$ 个元素起，每个元素 x 与其左边相距 h 的那些元素依次比较：若 x 小于某个元素 a_i，则将 a_i 往右移 h 个位置，……，最后将 x 插入适当的位置。我们启发学生：对每个 x，其左边相距 h 的那些元素已经是排好序的，简称 h -有序。因此，我们不需要将与 x 左边依次相距 h 的元素逐个比较大小，而可以用折半查找法将 x 插入适当的位置，使得在 h -步排序后，元素都是 h -有序的。这样，平均情况下，比较次数可由 $O(n^2)$ 降到 $O(n\log_2 n)$，而且，排序的元素越多，优势越明显。

(1) Shell 排序算法

Shell 排序中，每一遍 h -步排序可用以下算法描述：

```
Procedure h-sort(n,h:integer);
    var x,I,j:integer;
     b:booleam;
       Begin
          for i:=h+1 to n do
             begin
                x:=a[i]; j:=i-h; b:=true;
                   while (b=true) and (a[j]>x) do
                      begin
                         a[j+h]:=a[j]; j:=j-h;
                     if j<=0 then b:=false
                end
           a[j+h]:=x;
     end
End; {h-sort}
```

上述过程执行一次就使相距为 h 的那些元素是 h -有序的。取 h 的一个递减序列 $h_1, h_2, \mathrm{L}\ , h_t$ 满足：$h_t = 1, h_{i+1} < h_i, i = 1, \mathrm{L}\ , t-1$。调用 t 次 h-sort(n, h)，就可以使 n 个元素的序列排好序。尽管目前尚不知道如何选择步长才能产生最好的结果，但大量的研究已经得出一些局部结论。例如，Knuth[8]指出，若取 $h_{k-1} = 3h_k + 1, h_t = 1, t = \lfloor \log_3 n \rfloor - 1, k = t, \mathrm{L}\ , 2$，则产生的步长序列

$$\mathrm{L}\ , 121, 40, 13, 4, 1 \qquad (1)$$

是合理的选择。以下算法不妨就取序列（1）式。

```
procedure Shellsort (a:array[1..n] of integer);

  var I,h:integer;
  begin
     h := (3^⌊log3 n⌋−1 − 1) div 2;
    while h>0 do
        begin
           h-sort(n, h)
           h:=(h-1) div 3
        end;
end; {Shellsort}
```

(2) 改进的 Shell 排序算法

如前所述，用折半插入法确定 h -步长排序中 x 的位置，首先要计算出折半元素所处的范围的左端位置 l 和右端位置 r。显然，$r = i - h$，其中，i 为 x 的位置。而 l 必满足 $1 \le l \le h$，并且，$l = h$ 当且仅当 h 整除 r。于是，l 可由（2）式确定：

$$l = \begin{cases} h & h \mid r \\ r - h \times (r \ div \ h) & else \end{cases} \qquad (2)$$

其次，要计算 l 到 r 之间相距 h 的元素个数。显然，不管 l 为（2）式中的何值，$r-l$ 总是 h 的倍数，且 $(r-l)/h$ 即表示区间中长度为 h 的子区间的个数。因此，l 到 r 间相距 h 的元素个数 s 为：$s = (r-l)\ div\ h + 1$。

再次，要确定 s 个元素的中间元素位置 m 为：$m = l + (s\ div\ 2) \times h$。

在用折半插入法找 x 的插入位置的循环中，l 与 r 的值交替改变，直到 $l > r$。这时，x 的插入位置就是 l。插入 x 之前，要把从 l 开始到 $i-h$ 中相距 h 的元素依次右移 h 个位置。

综上所述，改进的 Shell 排序如下：

```
program Shellsort (input, output);
   const nsort=100;   {*待排序的元素个数*}
   var a: array[-121..nsort] of integer;
       m,n,I,j,s,h: integer;
   procedure binsert(n,h: integer);
    {*步长为h的折半插入排序*}
    var x,I,j,r,s,m: integer;
    begin
       for i:=h+1 to n do
        begin
           x:=a[i];   j:=i-h;   r:=j;   l:=r-h*(r div h);
           if   l=0 then l:=h;
           while l<=r do
              begin
                 s:=(r-l) div h+1;
                 m:=l+(s div 2)*h;
                 if x<a[m] then r:=m-h else l:=m+h;
              end;
           while j>=l do
              begin
                 a[j+h]:=a[j];
                 j:=j-h;
```

```
                end;
              a[l]:=x
            end;
        end; {binsert}
    begin
      n:=nsort;
      writeln('n=', n);
      Randomize;   {*随机产生n个1000以内的待排序的整数元素*}
      for i:=1 to n do a[i]:=trunc(random*1000);
      m:=trunc(ln(n)/ln(3))-1;   {*根据n的大小计算第一个步长 h1 *}
      h:=1;
      for i:=1 to m do h:=h*3;
      h:=(h-1) div 2;
      for i:=l-h to 0 do a[i]:=0;   {*对数组a的 h1 个扩展元素初始化*}
      while h>0 do
        begin
          binsert(n, h);
          h:=(h-1) div 3     {*选择下一个步长*}
        end;
    end; {Shellsort}
```

(3) 分析与比较

众所周知，Shell 排序的算法分析是一个至今尚未解决的数学难题，这是因为它的时间复杂度是所取步长的函数。但我们可以将改进的 h -步排序算法与原来的 h -步排序算法在比较次数 C 上作一个比较，主要分析元素 x 分别与 $a[j]$ 和 $a[m]$ 的比较次数。

① 算法 h-sort 的比较次数

a. 最小比较次数 $C_{\min}^{h}$ 。

当 x 比它左边相距 h 的元素大时，比较不需再进行下去，故只比较一次。因此，$C_{\min}^{h}=\sum_{i=h+1}^{n}1=n-h$ 。

b. 最大比较次数 $C_{\max}^{h}$ 。

设第 i 遍的 h -步排序共有 S_i 个元素，则由（2）式，有

$$\begin{aligned}S_i&=(r-l)\ div\ h+1\\&=\begin{cases}(r-h)\ div\ h+1 & h|r\\(r-(r-h\times(r\ div\ h)))\ div\ h+1 & else\end{cases}\\&=\begin{cases}(r-h)\ div\ h+1 & h|r\\r\ div\ h+1 & else\end{cases}\end{aligned}\qquad(3)$$

(i) 当 $h|r$ 时，有 $r=i-h\geq h$，即 $i\geq 2h$，从而，

$$S_i=(r-h)\ div\ h+1=(i-2h)\ div\ h+1 \qquad (4)$$

于是，

$$\begin{aligned}C_{\max}^{h}&=\frac{1}{h}\sum_{i=2h}^{n}(i-2h)+\sum_{i=2h}^{n}1\\&=\frac{(n-2h)(n-2h+1)}{2h}+(n-2h+1)\\&=\frac{n(n-2h+1)}{2h}\end{aligned}\qquad(5)$$

因此，平均比较次数为：

$$\begin{aligned}C_{ave}^{h}&=\frac{1}{2}(C_{\min}^{h}+C_{\max}^{h})\\&=\frac{1}{2}[(n-h)+\frac{n(n-2h+1)}{2h}]\\&=\frac{1}{4h}(n^2+n-2h^2)\\&=O(n^2)\end{aligned}\qquad(6)$$

(ii) 当 $h|r$ 不成立时，有：

$$\begin{aligned}C_{\max}^{h}&=\sum_{i=h+1}^{n}S_i=\frac{1}{h}\sum_{i=h+1}^{n}(i-h)+\sum_{i=h+1}^{n}1\\&=[1+2+\Lambda\ +(n-h)]+(n-h)\\&=\frac{(n-h)(n-h+1)}{h}+(n-h)\end{aligned}\qquad(7)$$

因此，平均比较次数为：

$$\begin{aligned}C_{ave}^{h}&=\frac{1}{2}(C_{\min}^{h}+C_{\max}^{h})\\&=\frac{1}{2}[\frac{(n-h)(n-h+1)}{h}]+(n-h)\\&=\frac{(n-h)(n-h+3)}{2h}=O(n^2)\end{aligned}\qquad(8)$$

② 改进算法 binsert 的比较次数

显然，比较一次，就去掉当前元素的一半。这样，对 S_i 个元素最多要对分 $\lfloor\log_2 S_i\rfloor+1$ 次，最少要对分 $\lfloor\log_2 S_i\rfloor$ 次，S_i 由（3）式定义。因此，

$$C_{\min}^{h}=\sum_{i=h+1}^{n}\lfloor\log_2 S_i\rfloor \qquad (9)$$

$$C_{\max}^{h}=\sum_{i=h+1}^{n}\lfloor\log_2 S_i\rfloor+1 \qquad (10)$$

$$C_{ave}^{h}=\frac{1}{2}(C_{\min}^{h}+C_{\max}^{h}) \qquad (11)$$

我们可以用积分

$$\int_{h+1}^{n} \log x\, dx = x(\log x - c)\Big|_{h+1}^{n} = n(\log n - c) - (h+1)(\log(h+1) - c)$$

，其中 $c = \log e = 1/\ln 2 = 1.44269...$ 来近似计算和式（9）和（10）可得（11）式结果为 $O(n\log_2 n)$。

③ 结论

通过分析比较得知，对一遍 h -步排序而言，改进前后的元素比较次数分别为 $O(n^2)$ 和 $O(n\log_2 n)$。平均情况下，当 n 越大时，改进后的算法效率越高。

3 结束语

创新是一个民族进步的灵魂，是一个国家兴旺发达的不竭动力。著名的教育家陶行知说过，“处处是创造之地，天天是创造之时，人人是创造之才。” 我们应该将培养学生的“创新意识和思维”作为“金课”建设的核心，作为课程教学的主要目的。只有这样持之以恒坚持下去，才能培养出建设社会主义的栋梁之才，到那时，也许可以回答“钱学森之问”。

参 考 文 献

[1] 崔佳，宋耀武. “金课”的教学设计原则探究[J]. 中国高等教育, 2019(5): 46-48.

[2] 陈宝生. 在新时代全国高等学校本科教育工作会议上的讲话[J]. 中国高等教育, 2018(增刊 3): 4-10.

[3] 吴岩. 建设中国“金课”[J]. 中国大学教学, 2018(12): 4-9.

[4] Harary, F, 图论，上海科学技术出版社，1980.

[5] Dirac, G. A, Short proof of Menger’s graph theorem. Mathmatika 13(1966), 42-44.

[6] 李慰萱，图论，湖南科学技术出版社，1980.

[7] Shell D. L., A Highspeed Sorting Procedure, Comm ACM, 1959:2(7):30-32.

[8] Knuth D. E., The Art of Computer Programming, Vol. 3, Reading Mass: Addison-Wesley, 1973.