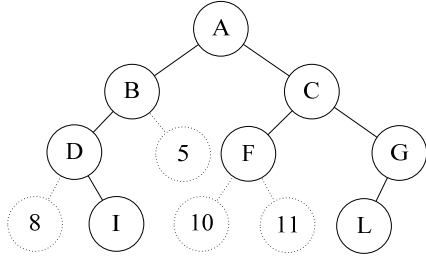
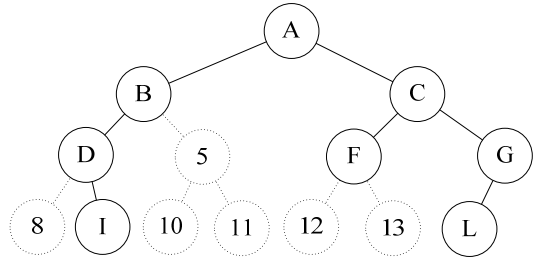


《数据结构与算法应用实践教程》

第一版第一次勘误表

修改位置	原 文	修 改
40 页的代码	缺少对 r 和 u 的声明	应在 exam()内加入声明的语句： LinkedList r,u;
50 页算法 4.1	第二行的 S = (SqStack *)malloc(MAXSIZE*sizeof(SqStack));	应该把该语句去掉或屏蔽掉： //S = (SqStack *)malloc(MAXSIZE*sizeof(SqStack));
55 页第 15 行	“尾打针增 1”；	“尾 指针 增 1”；
56 页第 10 行	释放栈 Q 占用的存储空间。	清空 栈 Q 占用的存储空间。
83 页第 24 行	int SubString(String Sub,String S, int pos, int len) {	int SubString(String &Sub,String S, int pos, int len) {
85 页第 28 行	若主串 S 中第 pos 个字符之后存在与 T 相等的子串，则返回第一个这样的子串在 S 中的位置，否则返回。	若主串 S 中第 pos 个字符之后存在与 T 相等的子串，则返回第一个这样的子串在 S 中的位置，否则返回 0 。
87 页第 35 行	return 0 ;	return 1 ;
97 页最后一行	(2) 计算第三位的时候，看第二位 b 的 next 值，为 1，则把 b 和 1 对应的 a 进行比较，不同，则第三位 a 的 next 的值为 1，因为一直比到最前一位，都没有发生比较相同的现象；（ 去掉红色逗号 ）	(2) 计算第三位的时候，看第二位 b 的 next 值为 1，则把 b 和 1 对应的 a 进行比较，不同，则第三位 a 的 next 的值为 1，因为一直比到最前一位，都没有发生比较相同的现象；
87 页第 19 行	$0 \leq i \leq m-1, 0 \leq j \leq n-1$	$1 \leq i \leq m, 1 \leq j \leq n$
87 页第 32 行	int col, totalN,k=0;	int col, totalN, k=1 ;
87 页第 37 行	for (col =1; col <a.nu; col ++)	for (col =1; col <= a.nu ; col ++)
87 页第 38 行	for (totalN=0; totalN <a.tu; totalN ++)	for (totalN= 1 ; totalN <= a.tu ; totalN ++)
109 页第 4 行	rpot[1]=1;	cpos [1]=1;
111 页第 9 和 10 行	if (!(M.rhead = (CrossLink *)malloc((m+1)*sizeof(OLink)))) return ERROR;	if (!(M.rhead = (CrossLink *)malloc((m+1)*sizeof(CrossLink)))) return ERROR;

	if (!(M.chead = (CrossLink *)malloc((n+1)*sizeof(OLink)))) return ERROR;	if (!(M.chead = (CrossLink *)malloc((n+1)*sizeof(CrossLink)))) return ERROR;
115 页代码	<pre> case 0: v = M.data[m].e + N.data[n].e; if(v!=0) { Q.data[q].i = M.data[m].i; Q.data[q].j = M.data[m].j; Q.data[q].e = v; ++q; } m++; n++; break; </pre> <p>注：此处代码为 case0 情况下的 case0:</p>	<pre> case 0: ++q; v = M.data[m].e + N.data[n].e; if(v!=0) { Q.data[q].i = M.data[m].i; Q.data[q].j = M.data[m].j; Q.data[q].e = v; } m++; n++; break; </pre>
115 页代码	<pre> case 1: ++q; Q.data[q].i = N.data[m].i; //将矩阵 N当前元素赋值给矩阵Q Q.data[q].j = N.data[m].j; Q.data[q].e = N.data[m].e; n++; break; } break; </pre> <p>注：此处代码为 case0 情况下的 case1:</p>	<pre> case 1: ++q; Q.data[q].i = N.data[n].i; //将 矩阵N当前元素赋值给矩阵Q Q.data[q].j = N.data[n].j; Q.data[q].e = N.data[n].e; n++; break; } break; </pre>
115 页代码	case 1:	case 1:

	<pre>++q; Q.data[q].i = N.data[m].i; Q.data[q].j = N.data[m].j; Q.data[q].e = N.data[m].e; n++; break; } }</pre>	<pre>++q; Q.data[q].i = N.data[n].i; Q.data[q].j = N.data[n].j; Q.data[q].e = N.data[n].e; n++; break; } }</pre>
111 页第 5 行	<p>当后者中的虚结点 5、8、10、11 补上，就构成了一棵完全二叉树。</p>  <p>图 6.5 非完全二叉树</p>	<p>当后者中的虚结点 5、8、10、11、12、13 补上，就构成了一棵完全二叉树。</p>  <p>图 6.5 非完全二叉树</p>
126 页倒数第五行	<pre>int i; for(i=0;i<MAXTREESIZE;i++) T[i]=0; //初值为空 printf("请按层序输入结点的值(整型), 0 表示空结点, 9999 表示结束。结点数\n");</pre>	<pre>int i=0; //for(i=0;i<MAXTREESIZE;i++) T[i]=0; //初值为空 printf("请按层序输入结点的值(整型), 0表示空结点, 9999 表示结束。结点数<=%d\n", MAXTREESIZE);</pre>
127 页算法 6.2	<pre>return T[2*i+1];</pre>	<pre>return T[2*i];</pre>
127 页算法 6.3	<pre>return T[(i+1)/2-1];</pre>	<pre>return T[i/2-1];</pre>
139 页第 4—5 行	示进行先根遍历所得结点的访问序列为：ABEHIJCD FG K， 进行	示进行先根遍历所得结点的访问序列为：ABEHIJC FD GK，

	后根遍历所得结点的访问序列为：HIJEBCFKGDA。	进行后根遍历所得结点的访问序列为：HIJEB F CKGDA。
140 页中间	第二个 ltag = $\begin{cases} 0 & \text{表示lchild指向左孩子的结点} \\ 1 & \text{表示lchild指向直接前驱的结点} \end{cases}$	rtag = $\begin{cases} 0 & \text{表示rchild指向右孩子的结点} \\ 1 & \text{表示rchild指向直接后继的结点} \end{cases}$
173 页图 7.10	<p>(c) 无向图 G3 的邻接表</p>	<p>(c) 无向图 G3 的邻接表</p>
198 页图 7.32	<p>(a) 有向图 G</p> <p>(b) 选顶点 3</p> <p>(c) 选顶点 1</p> <p>(d) 选顶点 4</p> <p>注：结点 5 到 6 方向有问题</p>	<p>(a) 有向图 G</p> <p>(b) 选顶点 3</p> <p>(c) 选顶点 1</p> <p>(d) 选顶点 4</p>
217 页倒数第 15 行	(数字 3 反而还被换到了最后一位)。也就是说，这个算法的效	(数字 5 反而还被换到了最后一位)。也就是说，这个算法

	率是非常低的。	的效率是非常低的。
228 页 6—7 行	<pre>while(i<=m) TR[k+l]=SR[i+l]; while (j<=n) TR[k+l]=SR[j+l];</pre>	<pre>while(i<=m) TR[k+l]=SR[i++]; while (j<=n) TR[k+l]=SR[j++];</pre>
244 页 4—5 行	<p>为$\lfloor \log_2 \rfloor + 1$，所以，折半查找在查找成功时和给定值进行比较的</p> <p>关键字个数至多为$\lfloor \log_2 \rfloor + 1$。</p>	<p>为$\lfloor \log_2 n \rfloor + 1$，所以，折半查找在查找成功时和给定值进行</p> <p>比较的关键字个数至多$\lfloor \log_2 n \rfloor + 1$。</p>
272 页倒数每 1 行	由上述定理可知：B-树的高度为 $O(\log n)$ 。	由上述定理可知：B-树的高度为 $O(\log n)$ 。
273 页倒数每 9 行	而 $m/\lg t > 1$ ，所以 m 较大时 $O(m \log n)$ 比平衡的二叉排序树上相应操作的时间 $O(\lg n)$ 大得多。	而 $m/\lg t > 1$ ，所以 m 较大时 $O(m \log n)$ 比平衡的二叉排序树上相应操作的时间 $O(\lg n)$ 大得多。