中山大学计算机学院本科生实验报告

（2021 学年第 1 学期）

课程名称：Data structures and algorithms　　任课教师：张子臻

年级	20 级	专业（方向）	软件工程
学号	20337270	姓名	钟海财
电话	13397996670	Email	2940599563@qq.com
开始日期	2021/10/21	完成日期	2021/10/26

1. 实验题目

1）heap

2）Subset Generation I

3）Permutation Generation

2.实验目的

1）实现小根堆的：插入操作 push 与根节点的删除操作 pop

2）给一个数字，使用递归，求出对应字母集合的全部子集

3）给一个数字，使用递归，求出对应数字集合的全排列

3.程序设计

1）heap

设计思路：

push 操作：先让 size()+1：n = n+1。用类似插入排序的方法，将 x 插入到 h(1)~…~h(n/2)~h(n)序列中。

Pop 操作：保存堆底元素 last = h[n--],再将 h(2)…~h(n/2)序列中比 last 小的节点（比其兄弟节点小的那个）都移动到其父亲节点，最后得到一个空节点，将 last 插入到该空节点中。

代码：

```
#include"heap.h"
void heap::push(int x) {
    int k = ++n;
    for(; k > 1 && h[k/2] > x;k/= 2)
       h[k] =  h[k/2];
    h[k] = x;//找到空位将 x 插入
}
void heap::pop(){
    int k, child;
    int last = h[n--];//将最后一个值保存下来
    //删除一个元素所以自减运算
    for (k = 1; 2*k <= n ; k = child){
        child = 2*k;
        //防止访问越界
        if (child != n && h[2*k] > h[2*k+1])
            ++child;
//较小的儿子与最后一个值的大小，如果儿子较小用儿子上滤，否则跳出循环
        if (h[child] < last) h[k] = h[child];
        else break;
    }
    h[k] = last;
}
```

2）Subset Generation I

设计思路：

设计了一个将数字转化为字母并输出的函数 void print(int t),转化规则：

if(num == 0) return ;

else cout<<(char)(num+64) ; //如 1->A,2->B… .

原来我是用递归，开始 sum=0。终止条件 sum==n，并输出子集中包含的元素。每次递归挑选原集合中第 sum 个位置的元素是否存在于子集中，不存在时：

调用递归挑选第 sum+1 个位置的元素；存在时：调用递归挑选第 sum+1 个位置的元素。但发现实际输出与期望输出不符，我的实际输出如果用 2 进制表示是：从 00000，00001，000010，000011, … ,111111(共 n 位)。但期望输出是先输出空集，遍历完含 A 的子集，再遍历不含 A 含 B 的子集，再遍历不含 A，B 含 C 的子集…最终只剩最后元素。

通过观察，我发现在我原来的代码上更改，新增一个参数 ad 记录是否有新增元素，递归时每次有新增元素(ad==1)就输出子集中的所有元素，并将递归中顺序（第 sum+1 个位置的元素存在的递归调用式放在前面）调换一下。且开始 main 主函数里的参数 ad=1（这样可以开始时输出集）。

代码：

```
#include<iostream>
using namespace std;
void print(int num){//将数字转化为字符：1->A
    if(num == 0 ) return ;
    else cout<<(char)(num+64) ;
}

void out(int ok[] ,int n ,int sum,int ad){//sum 表示子集元素的个数
    if(ad==1){//每次有新增元素时都输出
        for(int i = 0 ; i <sum;++i)
        print(ok[i]);
        cout<<endl;
    }
    if(sum == n ) return ;//全部元素都被遍历时
    ok[sum]= sum+1;
    out(ok,n,sum+1,1);//集合中有新增元素时输出
    ok[sum]= 0;
    out(ok,n,sum+1,0);//无新增元素时不输出
}
int main(){
    int num ;
    cin>>num;
    int ok[num];
    out(ok,num,0,1);//开始时输出空集
}
```

3）Permutation Generation

设计思路：

定义一个 bool 类型的全局的数组：bool exist[20]; 用于记录数字 i 是否排列过，若排列过 exist[i]==true。

读入数字存放在 num 中，定义一个 int 数组 ok[],用于按顺序存放已经排列过的数字，定义一个 int 变量 sum，记录排列过的数字的个数，初始时 sum=0。

递归设计：当 sum==num 时，按顺序输出 ok[]中的数字，再输出一个换行符，即输出了一种排列情况，再终止递归。

否则挑选排在第 sum 个位置的数字，使用循环对 1~num 的数字进行遍历，如果 exist[i]!=true,说明该数字 i 未被排列过，令 ok[sum]=i,exist[i]=true,再调用递归挑选第 sum+1 个位置的数字。

代码：

```
#include<iostream>
using namespace std;

bool exist[20];
void out(int ok[] ,int n ,int sum){ //选择排在第 sum 个位置的数字
    if(sum == n ){ //当所有数字都排列在 ok[]中时输出 ok[]
        for(int i = 0 ; i < n ; ++i) cout<< ok[i]<<" ";
        cout<<endl;
        return ;
    }
    for(int i = 1 ; i <= n ;++i){
        if(!exist[i]){//i 没在 ok[]中时将 i 放入 ok 中
            ok[sum]= i;
            exist[i] = 1 ;
            out(ok,n,sum+1);//选择排在下一个位置的数字
            exist[i] = 0;
        }
    }
}
```

```
int main(){
    int num ;
    cin>>num;
    int sum = 0;
    int ok[num];
    out(ok,num,sum);
}
```

4.程序运行与测试

1）heap 主函数不知道，故无测试，已通过了 matrix 的评测

2）Subset Generation I

序号	测试输入	输出	是否通过
1	**3**	1. 2. A 3. AB 4. ABC 5. AC 6. B 7. BC 8. C	是
2	**4**	1. 2. A 3. AB 4. ABC 5. ABCD 6. ABD 7. AC 8. ACD 9. AD 10. B 11. BC 12. BCD 13. BD 14. C 15. CD 16. D	是
3	**5**	1. 1 2. A 3. AB 4. ABC 5. ABCD 6. ABCDE 7. ABCE 8. ABD 9. ABDE 10. ABE 11. AC	是

		12. ACD 13. ACDE 14. ACE 15. AD 16. ADE 17. AE 18. B 19. BC 20. BCD 21. BCDE 22. BCE 23. BD 24. BDE 25. BE 26. C 27. CD 28. CDE 29. CE 30. D 31. DE 32. E	

3）Permutation Generation

序号	输入	输出	是否通过
1	**3**	1. 1 2 3 2. 1 3 2 3. 2 1 3 4. 2 3 1 5. 3 1 2 6. 3 2 1	是
2	**4**	1. 1 2 3 4 2. 1 2 4 3 3. 1 3 2 4 4. 1 3 4 2 5. 1 4 2 3 6. 1 4 3 2 7. 2 1 3 4 8. 2 1 4 3 9. 2 3 1 4 10. 2 3 4 1 11. 2 4 1 3 12. 2 4 3 1 13. 3 1 2 4 14. 3 1 4 2 15. 3 2 1 4 16. 3 2 4 1 17. 3 4 1 2 18. 3 4 2 1 19. 4 1 2 3 20. 4 1 3 2 21. 4 2 1 3 22. 4 2 3 1 23. 4 3 1 2 24. 4 3 2 1	是
3	**5**	1. 1 2 3 4 5 2. 1 2 3 5 4 3. 1 2 4 3 5 4. 1 2 4 5 3 5. 1 2 5 3 4	是

			6.1 2 5 4 3 7.1 3 2 4 5 8.1 3 2 5 4 9.1 3 4 2 5 10. 1 3 4 5 2 11. 1 3 5 2 4 12. 1 3 5 4 2 13. 1 4 2 3 5 14. 1 4 2 5 3 15. 1 4 3 2 5 16. 1 4 3 5 2 17. 1 4 5 2 3 18. 1 4 5 3 2 19. 1 5 2 3 4 20. 1 5 2 4 3 21. 1 5 3 2 4 22. 1 5 3 4 2 23. 1 5 4 2 3 24. 1 5 4 3 2 25. 2 1 3 4 5 26. 2 1 3 5 4 27. 2 1 4 3 5 28. 2 1 4 5 3 29. 2 1 5 3 4 30. 2 1 5 4 3 31. 2 3 1 4 5 32. 2 3 1 5 4 33. 2 3 4 1 5 34. 2 3 4 5 1 35. 2 3 5 1 4 36. 2 3 5 4 1 37. 2 4 1 3 5 38. 2 4 1 5 3 39. 2 4 3 1 5 40. 2 4 3 5 1 41. 2 4 5 1 3 42. 2 4 5 3 1 43. 2 5 1 3 4 44. 2 5 1 4 3 45. 2 5 3 1 4 46. 2 5 3 4 1 47. 2 5 4 1 3 48. 2 5 4 3 1 49. 3 1 2 4 5 50. 3 1 2 5 4 51. 3 1 4 2 5	

```
52. 3 1 4 5 2
53. 3 1 5 2 4
54. 3 1 5 4 2
55. 3 2 1 4 5
56. 3 2 1 5 4
57. 3 2 4 1 5
58. 3 2 4 5 1
59. 3 2 5 1 4
60. 3 2 5 4 1
61. 3 4 1 2 5
62. 3 4 1 5 2
63. 3 4 2 1 5
64. 3 4 2 5 1
65. 3 4 5 1 2
66. 3 4 5 2 1
67. 3 5 1 2 4
68. 3 5 1 4 2
69. 3 5 2 1 4
70. 3 5 2 4 1
71. 3 5 4 1 2
72. 3 5 4 2 1
73. 4 1 2 3 5
74. 4 1 2 5 3
75. 4 1 3 2 5
76. 4 1 3 5 2
77. 4 1 5 2 3
78. 4 1 5 3 2
79. 4 2 1 3 5
80. 4 2 1 5 3
81. 4 2 3 1 5
82. 4 2 3 5 1
83. 4 2 5 1 3
84. 4 2 5 3 1
85. 4 3 1 2 5
86. 4 3 1 5 2
87. 4 3 2 1 5
88. 4 3 2 5 1
89. 4 3 5 1 2
90. 4 3 5 2 1
91. 4 5 1 2 3
92. 4 5 1 3 2
93. 4 5 2 1 3
94. 4 5 2 3 1
95. 4 5 3 1 2
96. 4 5 3 2 1
97. 5 1 2 3 4
```

		98. 5 1 2 4 3 99. 5 1 3 2 4 100. 5 1 3 4 2 101. 5 1 4 2 3 102. 5 1 4 3 2 103. 5 2 1 3 4 104. 5 2 1 4 3 105. 5 2 3 1 4 106. 5 2 3 4 1 107. 5 2 4 1 3 108. 5 2 4 3 1 109. 5 3 1 2 4 110. 5 3 1 4 2 111. 5 3 2 1 4 112. 5 3 2 4 1 113. 5 3 4 1 2 114. 5 3 4 2 1 115. 5 4 1 2 3 116. 5 4 1 3 2 117. 5 4 2 1 3 118. 5 4 2 3 1 119. 5 4 3 1 2 120. 5 4 3 2 1	

5.实验总结与心得

在使用递归的时候，类似这次的 2）3）题都是使用了 2 个递归式，我们可以画出对应的二叉树帮助我们理解递归调用的顺序是什么，是先执行了什么，再执行了什么。

附录、提交文件清单