摘要:

本文研究了组合数学中常见的球放盒子模型问题,并给出了一种使用回溯算法实现这类问题的通用代码模板。首先总结了这类问题的几种典型情况,如固定数量的球放入固定数量盒子的排列组合问题。然后介绍了回溯算法思想和实现过程。针对不同问题设置不同的球和盒子数组以及标记数组,通过递归函数调用实现全排列或全组合。最后给出了使用 C++实现的通用代码,并给出几个实例问题的测试结果。该通用代码实现了组合数学中球放盒子模型问题的自动求解,具有很好的扩展性和可复用性。

正文开头

组合数学中常见的一类问题是将一定数量的球放入一定数量的盒子中的排列组合问题。这类问题具有广泛的实际应用背景,如班级学生分组、员工分工等。由于问题类型相似且算法思路类似,可以设计一个通用的代码模板来实现这类问题的自动求解。本文将研究使用回溯算法来解决这类球放盒子模型问题,给出一个通用的 C++代码实现。

首先对这类问题进行分类总结。然后介绍回溯算法的基本思路。接着给出使用 C++设计的通用代码框架,包括球盒子数组以及标记数组的定义。通过设置不同的参数和递归函数,实现不同问题类型的求解。最后给出几个实例问题的测试结果,验证该通用代码的正确性和可扩展性。

本文旨在给出一个高效且通用的算法实现,解决组合数学中球放盒子模型问题的自动求解需求。该通用代码具有很好的复用性,可以方便地应用于相关其他问题。

我们从四个维度来考虑: 球是否相同,盒子是否相同,盒子是否可以放多个球,是否允许盒子为空.

1 全排列问题

限制: 球不同,盒子不同,球与盒子一样多

有 3 个不同的球,编号为 1,2,3.有 2 个不同盒子编号为 b1,b2,现在把球放到盒子里,每个盒子恰好放一个球,有多少不同的放法?

先手动列出所有的可能性,如下:

盒子	b_1	b_2	b_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

我们假设有 3 个小朋友,每个小朋友负责选取一个球放入对应 bi 盒子里。每个小朋友的任务如下.

- 第 i 个小朋友,从剩余的球里取出一个编号最小的球,放到编号为 i 个的盒子里
- 剩下的球让后面的小朋友取
- 回溯到第 i 个小朋友的时候,恢复现场:把编号 i 盒子里的球的取回来后,再从剩余的球里取出一个编号比 i 大的球,放到编号为 bi 个的盒子里

每个小朋友的任务是一样的,这是一种递归的算法,写出如下代码:

```
#include <iostream>
using namespace std;
const int maxn = 1e5+5;
int n;int rcd[maxn]; //  记录,rcd[1]=2,表示小朋友人选了球 2int vis[maxn]; // vis[i] 表示  球 i 被选走了,被使用了
void full_permutation(int dep) {
	if( dep > n ) {
		//输出选的球
		for(int i =1;i<=n;i++)
			cout << rcd[i] << " ";
		cout << endl;
		return;
	}
	for(int i = 1;i<=n;i++){
		if(vis[i]) continue; //球 i 被选了,就略过
		rcd[dep] = i;
		vis[i] = 1; //记录这球 i 被选
		full_permutation(dep+1); //下一个小朋友去选
		vis[i] = 0;//  放回这个球,  恢复现场
	}
}
int main(){
	n = 3;
	full_permutation(1);
	return 0;
```

}

这个问题在数学上叫做全排列问题,把 n 个不同的数进行全排列,记为 P(n,n),那么

$$P(n,n) = \underbrace{n \times (n-1) \times \cdots \times 1}_{n} = n!$$

所以这个代码的时间复杂度为 P(n,n)

2 排列问题

特点:球不同,盒子不同,盒子少于球,不可空

有 3 个不同的球,编号为 1,2,3 有 2 个不同盒子编号为 b1,b2,现在把球放到盒子里,每个盒子恰好放一个球,有多少不同的放法?

先手动列出所有的可能性,如下:

盒子	b_1	b_2
1	1	2
2	1	3
3	2	1
4	2	3
5	3	1
6	3	2

与上一个问题不同,这里的盒子的数量是少于球的数量的,那我们只要减少小朋友的数量到盒子的数量不就可以了吗?写出递归代码如下:

```
int n,m; //n 个球,m 个盒子
int rcd[maxn]; // 记录,rcd[1]=2,表示小朋友人选了球 2
int vis[maxn]; // vis[i] 表示 球 i 被选走了,被使用了
void permutation(int dep) {
    if( dep > m ) { //注意改了这里
        //输出选的球
        for(int i =1;i<=m;i++) //注意改了这里
            cout << rcd[i] << " ";
        cout << endl;
        return;
    }
    for(int i = 1;i<=n;i++){
        if(vis[i]) continue; //球 i 被选了,就略过
        rcd[dep] = i;
```

```
            vis[i] = 1; //记录这球 i 被选
            permutation(dep+1); //下一个小朋友去选
            vis[i] = 0;// 放回这个球, 恢复现场
        }
}
```

这个问题在数学上做排列问题,从 n 个不同的数里选 m 个数进行排列,记为 P(n,m),

$$P(n,m) = \underbrace{n \times (n-1) \times \cdots \times (n-m+1)}_{m} = \frac{n!}{(n-m)!}$$

那么

3 部分组合问题

特点:球不同,盒子相同

有 5 个不同的球,编号为 1,2,3,4,5 有 3 个相同盒子,现在把球放到盒子里,每个盒子恰好放一个球,有多少不同的放法?

因为三个盒子一样,我们注意到:

1	1	2	3
2	1	3	2
3	2	1	3
4	2	3	1
5	3	1	2
6	3	2	1

这些结果都是一样的.

同时我们列出所有的可能结果,得出一共有 10 种可能性,如下:

1	1	2	3
2	1	2	4
3	1	2	5
4	1	3	4
5	1	3	5
6	1	4	5
7	2	3	4
8	2	3	5
9	2	4	5
10	3	4	5

这里的重点问题在于如何避免重复.

这样思考:有三个小朋友,每个小朋友拿一个盒子,然后开始拿球,已经拿了 1,2,3 后,如何避免拿 1,3,2 或 3,2,1 等重复的情况呢?

答案就是: 保证第 i 个小朋友拿的球的编号大于前面的第 i-1 个小朋友拿的球的编号.

于是我们发现了一条规律:定序唯一性.在上式中这些重复的放法中,有序的只有一个.保证有序,就可以避免重复.

注意:这里不需要标记球是否已经被拿了,因为选的球的编号一定比前面的大.所以也不需要恢复现场,也不需要放回球.

```
#include <iostream>
using namespace std;
const int maxn = 1e5+5;
int n,m; // n 个球,m 个相同的盒子
int rcd[maxn]; // 记录,rcd[1]=2,表示小朋友人选了球 2
//int vis[maxn]; // vis[i] 表示 球 i 被选走了,被使用了
//不需要 vis,也不需要放回球,

// comb combination 的简写
// dep 深度,pre 前一个选的数
void comb(int dep,int pre) {
    if( dep > m ) {
        //输出选的球
```

```
        for(int i =1;i<=m;i++)
            cout << rcd[i] << " ";
        cout << endl;
        return;
    }
    for(int i = pre+1;i<=n;i++){
        rcd[dep] = i;
        comb(dep+1,i); //下一个小朋友去选
    }
}
int main(){
    n = 5;
    m = 3;
    comb(1,0);
    return 0;
}
```

这个问题在数学上叫做:一般组合问题.从 n 中选取 m 个数有多少种可能性,记为 C(n,m).根据集合的映射关系,得到如下:

$$C(n,m) = \frac{P(n,m)}{m!} = \frac{n!}{m! \cdot (n-m)!}$$

4 全组合问题

有 5 个不同的球,编号 1,2,3,4,5,有一个魔法盒子,它的容量可以变化,最少可以放 0 个球,最多可以放所有的球.问有多少不同的放法? 并输出所有不同的放法方案.

容易想到:当魔法盒子的容量固定时,问题就变成了上一个问题. 所以只要写一个 for 循环调用上面代码的 comb 函数就可以了,代码如下:

```
for(int i =1;i<=n;i++)
{
    m = i; //修改盒子的数量
    comb(1,0);
}
```

经过思考,如果让每个小朋友开始选球的时候,把前面的小朋友选的球输出,就得到了一个更简单的代码,如下:

```
#include <iostream>
using namespace std;
const int maxn = 1e5+5;
```

```
int n;int rcd[maxn];
// dep 深度,当前给哪个盒子选
void full_comb(int dep,int p){

    //每一次进入都输出
    for(int i =1 ;i<dep;i++)
    {
        cout << rcd[i] << " ";
    }
    cout << endl;
    for(int i = p+1;i <=n;i++)
    {
        rcd[dep] = i;
        comb(dep+1,i);
    }
}
int main() {
    n = 3;
    comb(1,0);
    return 0;
}
```

根据组合的相关公式$\sum_{i=0}^{n} C(n,i) = 2^n$. 得到这个代码的时间复杂为:$O(2^n)$

5 整数非零划分

特点:球相同,盒子不同,不允许为空

有 5 个相同的球,有 2 个不相同盒子,编号为 b1,b2,现在把球放到盒子里,每个盒子可以放多个球,但不能为空,有多少不同的放法?

可以想到本题目其实就是求$x + y = 5$有多少种正解数解.列出所有解如下:

b_1	b_2
1	4
2	3
3	2
4	1

因为只有两个盒子,可以直接写 for 循环来进行枚举

```
for(int i =1 ;i<=4;i++)
{
    cout << i << " " << 5-i << endl;
}
```

但是如果盒子的数量不固定呢?那就不知道需要用几重 for 循环,所以这里使用递归算法,相当于动态的层数的 for

```
#include <iostream>
using namespace std;
const int maxn = 1e5+5;
int rcd[maxn];int n,m;
//left 表示剩下的数字是多少
void dfs(int dep,int left) {
    if( dep == m ) //最后一个人,全部拿
    {
        rcd[dep] = left;
        for(int i =1;i<=n;i++)
            cout << rcd[i] << " ";
        cout << endl;
        return;
    }

    //for(int i =1;i<=left;i++)
    //更好的写法,后面 m-dep 个人
    // 每个人都至少需要拿一个
    for(int i =1;i<=left-(m-dep);i++)
    {
        //当前不够
        if( left-i <=0 ) continue;
        rcd[dep] = i;
        dfs(dep+1,left-i);
    }
}
int main(){
    n = 5; //5 个球
    m = 2; //2 个盒子
    return 0;
}
```

这个题目本质上是求把 n 分成多份,每一份不可以为 0,有多少种分法. 也就是排列组合的经典问题:隔板法.

可以这样思考,5 个球排成一排,使用一个隔板把它们分成两份,分别放到两个盒子

里.想当于从 5 个间隔的位置中选 1 个,也就是 C(4,1)

所以更一般的公式如下:

$$C(n-1,m-1)$$

6 整数可零划分

有5个相同的球,有2个不相同盒子,编号为b_1,b_2,现在把球放到盒子里,每个盒子可以放多个球,可以为空,有多少不现的放法?

b_1	b_2
1	4
2	3
3	2
4	1
0	5
5	0

与上面的解法一样,把代码里改成从 0 开始就可以了.这里也是隔板法.具体的放法数对应的公式.通过构造映射来解.有 7 个球,先每个盒子放一个球,剩余 5 个球随便放,可以想到最后的结果已经 7 个球,盒子不空的放法.所以 5 个球可以为空放法就是

$$C(7-1,2-1) \ = \ C(5+2-1,2-1)=C(6,1)=6$$

更一般的对于 n 个相同的球,m 个不同的盒子,可以为空,可以多放的方法数

$C(m-1,n+m-1)$

有 5 个相同的球,有 2 个相同的盒子,现在把球放到盒子里,每个盒子可以放多个球,可以为空,有多少不现的放法?

b_1	b_2
0	5
1	4
2	3

本题目其实求的是$x+y=5 \ \wedge \ x \leqslant y$有多少种正解数解

根据定序唯一性.只要保证当前的盒子的数量大于等于前面的盒子里的球的数量.

这个题目可以认为是整数的有序划分.

代码 TODO

8 有重集排列

有 2 种相同的球,分别为 1,1,2,有三个不同的盒子,编号为 b1,b2,b3,现在把球放到盒子里,每个盒子可以放一个球,有多少不同的放法?

同样,先列出所有的放法.

b_1	b_2	b_3
1	1	2
1	2	1
2	1	1

使用缩小放大法,先思考简单的问题

如果只有一个盒子,那方案数只能是 2，因为只有两种球.

b_1
1
2

如果所有的球都一样,例如三个球都为:{1,1,1}.,这个时候每个盒子都只能选 1,也只有一种可能性

显然可得:对于盒子数只有一个或只有一种球的情况下,选球方案数为 1

从集合的角度来看,设有重集:A={1,1,2},表示每一个小球,再设 3 个人在小球的集合 A 按要求选择后的形成的选法方案集合为 f(3,A)

$$f(3,A) \ = \ \{(1,1,2),(1,2,1),(2,1,1)\}$$

由上式知,f(3,A)的元素数量为 3.且知当所有的元素都一样时,放球的方案数为 1.

利用子问题分解(集合分类)思想,显然第一个人在集合 A 上只有两种选球的方案:

1. 选 1,剩余的小球集合为$B_1 = \{1, 2\}$,产生一个新问题:$f(2, B_1)$
2. 选 2,剩余的小球集合为$B_2 = \{1, 1\}$,产生一个新问题:$f(2, B_2)$

那么$f(3, A) = f(2, B_1) + f(2, B_2)$,易知只有一个盒子的情况下,只有一种放法.

那么我们可以根据上面的公式,写一个递归算法来求答案

```
// 有重复集合排列问题
#include <iostream>
using namespace std;
const int maxn= 1e5+5;
int n,m; //n 个原始元素,m 个盒子
int cnt;//cnt 个不同的元素
int a[maxn]; // 把相同元素放到箱子里
int rcd[maxn]; //记录选的值

void dfs(int dep) {
	if( dep > m) {//边界,输出
		for(int i =1;i<=m;i++)
			cout << rcd[i] << " ";
		cout << endl;
		return;
	}
	for(int i =1;i<=cnt;i++) {
		if( a[i] > 0) {
			a[i] --;
			rcd[dep] = i;
			dfs(dep+1);
			a[i] ++; //恢复现场
		}
	}
}

int main()
{
	cin >> n >> m;
	for(int i =1;i<=n;i++) {
		int t;
		cin >>t;
```

```
            if(a[t] == 0 ) cnt++;
            a[t]++;
        }
        dfs(1);
        return 0;
}
```

根据上面的代码,得知,把相同的球放到同一个箱子里,假如最后共有m个箱子,如果每个人每次从某个箱子里取一个球,不停的这样操作,最后可以得到所有的选球方案.

重要的思想是**集合分类**:相同的球算一类,对每个人来来说,**他选球的可能性就是球分类数 m**

数学解:

有 m 组球,每组球有$a[i]$个,有 n 个箱子,问有多少个排列方式

本问题的本质,本题是数学上的**有重集合排列问题**

如果盒子的数量与球的数量一样,且每个盒子放一个球,最后的方案数:

$$\frac{n!}{\prod_{i=1}^{m} a_i!}$$

9 有重集组合

与上题一样,只不过盒相同.

显然只有一种方法,如下:

1	1	2

具体代码与组合类似,使用**定序唯一性**,改写上题的代码,保证每个盒子里的球都比前面球的编号大.

10 第二类 Stirling 数

特点:球不同,盒子相同,可以多放,不可以为空

在这个问题中，我们考虑了以下情况：有 n 个不同的球，编号为 1,2,3,...,n，还有

m(m<=n)个相同的盒子，这些盒子可以放多个球，但不能是空的。我们需要回答以下问题：

1. 有多少种放法?
2. 列出所有的方案?

在这里，“盒子相同”意味着我们不需要考虑它们的顺序。
首先了为理解题目,手动枚举(暴力)一下,当 n=3，m=2 时，有 3 种方案

1 2	3
1 3	2
1	2 3

当有 n=4,m=2 时,有 7 种方案.

1 2 3	4
1 2 4	3
1 2	3 4
1 3 4	2
1 3	2 4
1 4	2 3
1	2 3 4

我们定义问题为 S(n,m),表示为有 n 个不同的球,m 个相同的盒子时的放法数量.

使用递推法,先考虑最简单的情况:

- 当 n < m 时,没有可能的方案,S(n,m) = 0
- 当 n = m 时,只有一种方案,S(n,m) = 1
- 当 m = 1 时,也只有一种方案,S(n,m)=1

当 n > m 时,我们考虑编号最大的那个球 n,显然

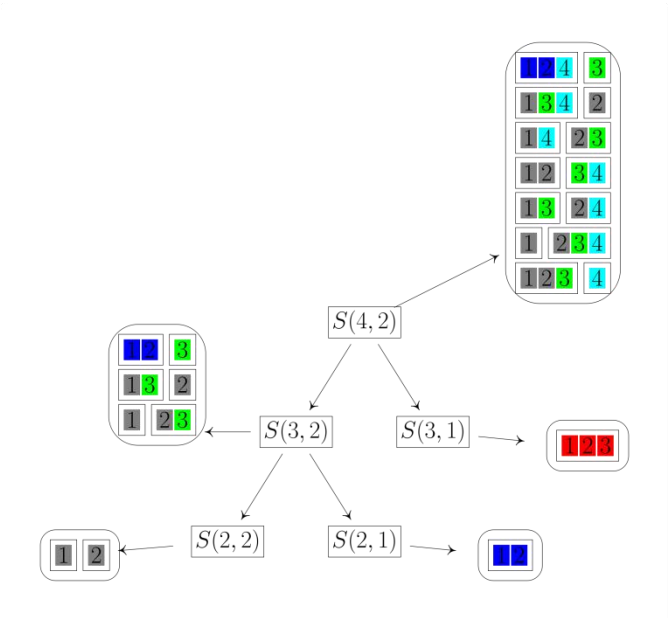
- 要么编号为 n 的球单独放一个盒子,此时,取走 1 个盒子,问题变成 S(n-1,m-1)
- 要么将球 n 单独放入一个盒子，这时问题转化为 S(n-1,m-1)
- 要么将球 n 和其他球放在同一个盒子里。我们可以将其他球先放入 m 个盒子中，因为球是不同的，这 m 个盒子中的每个盒子单独看，里面的球即使数量相同，球也不同，**我们可以认为此时有 m 个不同的盒子**。因此，编号为n的球可以放在任何一个盒子中，有 m 种放法。总的放法数为$m \times S(n-1,m)$。

综上所述

$$S(n,m)=\begin{cases}0 & n<m\\ 1 & n=m\\ 1 & m=1\\ S(n-1,m-1)+m\times S(n-1,m) & n>m\end{cases}$$

这种类型的问题被称为第二类 Stirling 数。

验证/演示:



所以对于第一问: 得到如下的求方案数的代码

```
int s[100][100];
int stirling(n,m) {
	if( n < m) return 0;
	if( n == m || m == 1) return 1;

	//记忆化
	if( !s[n][m]) return s[n][m];
	s[n][m] = stirling(n-1,m-1) + m*stirling(n-1,m);
	return s[n][m];
}
```

第二问: 根据上面的验证的过程,对于 n 号球,

1. 它要么单独放一个盒子,这个盒子此时就是就后一个盒子
2. 要么每个盒子都放一下

对于球 i 来说,记录他在哪个盒子里

所以最每个球只要记录它的所在盒子的编号就可以了.

```
#include <iostream>
#include <iomanip>
using namespace std;
const int maxn = 105;
int n = 4;
int m = 2;
int cnt = 0;
int ball[maxn] ;//记录 ball i  在哪个盒子里
void print_stirling() {
	cout <<    setw(4) << ++cnt << ":      ";
	for(int i = 1;i <= m ;++i ) // i: 1->m
	{
		cout << "[ ";
		for(int j = 1;j <= n ;++j ) // j: 1->n
		{
			if( ball[j] == i )
				cout << j << " ";
		}
		cout << "] ";
	}
	cout << endl;
}

// n 个不同球，m 个相同盒子
void stirling(int n,int m) {
	if( n < m ) return ;
	if( n == m) {
		for(int i =1;i<=m;i++)
			ball[i] = i;
		print_stirling();
		return;
	}
	if( m == 1) {
		for(int i =1;i<=n;i++)
			ball[i] = 1;
		print_stirling();
		return;
```

```
        }

        //情况 1: 最后一个球放最后一个盒子里
        ball[n] = m;
        stirling(n-1,m-1);

        //情况 2: 最后一个球放每个盒子里
        for(int i =1;i<=m;i++) {
                ball[n] = i;
                stirling(n-1,m);
        }
}

int main (int argc, char *argv[]) {
        stirling(n,m);
        return 0;
}
```

其它情况

1. 球不同,盒子不同,可以多放,不可以为空

与上一个题目,不同就在于,**盒子不同**,也就是需要考虑顺序.

$$\begin{aligned} G_1(n,r) &= P(m,m) \times S(n,m) \\ &= m! \times S(n,m) \end{aligned}$$

1. 球不同,盒子相同,可以为空

$$\begin{aligned} G_2(n,r) &= S(n,1) + S(n,2) + \ ... + S(n,m) \\ &= \sum_{i=1}^{m} S(n,i) \end{aligned}$$

球不同,盒子不同,可以多放,不可以为空

与上一个题目,不同就在于,盒子不同,也就是需要考虑顺序.

$$G(n,r) = P(m,m) \times S(n,m)$$
$$= m! \times S(n,m)$$

总结

	球是否相同	盒是否相同	盒是否可以为空	是否只能放一个	特点
排列	不同	不同	不空	放一个	不同球排队
组合	不同	相同	不空	放一个	不同球选取
整数非零划分	相同	不同	不空	放多个	相同球分堆
整数可零划分	相同	不同	可空	放多个	相同球分堆,有空堆
有重集排列	重复	不同	不空	放一个	多胞胎排队
有重集组合	重复	相同	不空	放一个	多胞胎选取
第二类stirling数	不同	相同	不空	放多个	不同球分堆