Force Recursion

汉诺塔问题

打印n层汉诺塔从最左移到最右的步骤

```
/*
问题分析:
1）将前n-1层放到中间的位置
2）将n层从最左放到最右
3）将前n-1层从中间放到最右
*/
void process(int i,string left,string right,string mid)
{
    //base case
    if(i==1)
    {
        cout<<"move"<<" "<<i<<" "<<left<<"-->"<<right<<endl;
        return;
    }

   else{ // n-1 move to mid
    process(i-1,left,mid,right);
    //
    cout<<"move"<<" "<<i<<" "<<left<<"-->"<<right<<endl;
    //
    process(i-1,mid,right,left);}


}
void nHanoiTower(int N)
{
```

```
    process(N,"left","right","mid");
}
```

打印字符串的全部子序列

```
/*
 打印字符串的所有子串
 按照每个字符是否保留，得到所有的结果
 */

void processV2Char(string s,int index,string ss)
{
    //base case
    if(index==ss.length())
    {
        if(s.length()<1) cout<<"None"<<endl;
        else cout<<s<<endl;
        return;
    }

    //1. char is ommit
    processV2Char(s,++index,ss);
    --index;
    //2. char is keep
    processV2Char(s+ss[index],++index,ss);




}
void pprintChars(const string s)
{
    processV2Char("",0,s);
}

// test
pprintChars("abc");

//console ouput
None
c
b
bc
```

```
a
ac
ab
abc
```

打印一个字符串的排列

要求打印全部排列，不能重复打印

```
/*
 打印字符串所有的排列组合
 */

string  swapString(string ss,int i,int j)
{
    if(i==j) return ss;
    else
    {
        char tmp=ss[i];
        ss[i]=ss[j];
        ss[j]=tmp;
    }
    return ss;
}

void processPermutations(string s,int i){
    // base case
    if(i==s.length())
    {
        cout<<s<<endl;
        return;
    }

    //
    set<char> visited; //to avoid repeat.
    for(int j=i;j<s.length();j++) // i-th index.
    {
        //
        if(visited.find(s[j])==visited.end()) // no visit and swap.
        {
            visited.insert(s[j]);
            s=swapString(s,i,j); //swap i and j index char
            // process
```

```
            processPermutations(s,i+1);
            s=swapString(s, i, j);
        }
    }


}
void pprintPermutations(string s)
{
    processPermutations(s,0);
}
```

问题一

给定一个整型数组arr，代表数值不同的纸牌排成一条线。玩家A和玩家B依次拿走每张纸牌，规定玩家A先拿，玩家B后拿，但是每个玩家每次只能拿走最左或最右的纸牌，玩家A和玩家B都绝顶聪明。请返回最后获胜者的分数。
【举例】
arr=[1, 2, 100, 4]。
开始时，玩家A只能拿走1或4。如果开始时玩家A拿走1，则排列变为[2, 100, 4]，接下来玩家 B可以拿走2或4，然后继续轮到玩家A. . .
如果开始时玩家A拿走4，则排列变为[1, 2, 100]，接下来玩家B可以拿走1或100，然后继续轮到玩家A. . .
玩家A作为绝顶聪明的人不会先拿4，因为拿4之后，玩家B将拿走100。所以玩家A会先拿1，让排列变为[2, 100, 4]，接下来玩家B不管怎么选，100都会被玩家 A拿走。玩家A会获胜，分数为101。所以返回101。
arr=[1, 100, 2]。
开始时，玩家A不管拿1还是2，玩家B作为绝顶聪明的人，都会把100拿走。玩家B会获胜，分数为100。所以返回100。

```
/*
 给定一个整型数组ar，代表数值不同的纸牌排成一条线。玩家A和玩家B依次拿走每张纸
 牌，规定玩家A先拿，玩家B后拿，但是每个玩家每次只能拿走最左或最右的纸牌，玩家A
 和玩家B都绝顶聪明。请返回最后获胜者的分数。
 【举例】
 arr=[1,2,100.4]。
 开始时，玩家A只能拿走1或4。如果开始时玩家A拿走1，则排列变为【2，100，4】，接下来
 玩家B可以拿走2或4，然后继续轮到玩家A..
 如果开始时玩家A拿走4，则排列变为【1，2，100】，接下来玩家B可以拿走1或100，然后继
 续轮到玩家A..
 玩家A作为绝顶聪明的人不会先拿4，因为拿4之后，玩家B将拿走100。所以玩家A会先拿1
 让排列变为【2，100，4】，接下来玩家B不管怎么选，100都会被玩家A拿走。玩家A会获胜，
 分数为101。所以返回101。arr=[1.100.2]。
 开始时，玩家A不管拿1还是2，玩家B作为绝顶聪明的人，都会把100拿走。玩家B会获胜，
```

```
 分数为100。所以返回100。
 */

/*
思路：
将问题分解：可以将A，B按照前后次序的过程提升为两种操作即：前后操作
1.前操作（F）：按照规则进行一次选择
2.后操作（S）：按照规则在前操作之后进行一次选择之后，再进行选择的操作
所以总的操作过程为：F->选择->S
在选择的过程中蕴含着不能让对方拿到大分值的贪心策略。
*/
int S(vector<int> arr,int L,int R);
int F(vector<int> arr,int L,int R);

int F(vector<int> arr,int L,int R)
{
    if(L==R) return arr[L];   //前操作

    // 前操作的选择策略：前操作选择+后操作选择 最大化
    return max(arr[L]+S(arr,L+1,R),arr[R]+S(arr,L,R-1));
}
int S(vector<int> arr,int L,int R){
    if(L==R) return 0; //因为后操作之前有一次选择的机会，所以，这里后操作没有选择了，为0

    // 后操作选择的策略：因为后操作之前的选择已经选择了一个分值，这要想前操作+后操作（前一个对象，A或者B）分值整体偏低，也就是自己的分值高；eg：[1,100,2],因为1或者2，已经被选择了，所以这里返回100和剩下的一个，两个其中小的那个，而对于自己来讲，100就在自己的前操作中找到，最大化了自己，最小化了对方。
    return min(F(arr,L+1,R),F(arr,L,R-1));
}


int question1(vector<int> arr)
{
    // 前面的F，代表着A，后面的S，代表着B，返回两者获胜者分数
    return max(F(arr,0,arr.size()-1),S(arr,0,arr.size()-1));
}
```

逆序栈

逆序栈，不能申请额外的数据结构，只能使用递归函数

```
/*
 逆序栈，不能申请额外的数据结构，只能使用递归函数
 */

```

```cpp
// 移除栈底元素，返回

int stackPopBottom(stack<int>& s)
{
    int result = s.top();
    s.pop();
    if(s.empty()) return result; //base case

    else
    {
//        result=s.top();
//        s.pop();
        int last = stackPopBottom(s);
        s.push(result);
        return last;
    }
}

void stackReverse(stack<int>& s)
{
    // base case
    if(s.empty()) return;

    else
    {
        int last=stackPopBottom(s);
        stackReverse(s);
        s.push(last);

    }
}
```

字符转换

规定1和A对应、2和B对应、3和C对应..，那么一个数字字符串比如"111”，就可以转化为"AAA”、"KA”和"AK”。给定一个只有数字字符组成的字符串str，返回有多少种转化结果。

```cpp
/*
 规定1和A对应、2和B对应、3和C对应..
 那么一个数字字符串比如"111”，就可以转化为"AAA”、"KA”和"AK”。
 给定一个只有数字字符组成的字符串str，返回有多少种转化结果。
 */

// 来到了i位置，前i-1位置已经确定了多少种了，总体的结果数
// 由i到结束来决定
```

```cpp
int processToString(vector<char> chs,int i)
{
    //base case
    if (i==chs.size()) return 1; // 到了结尾了，返回一种组成方案

    if(chs[i]=='0') return 0;// i位置为0，也就说明前i-1的组成方式，不能进行下去

    if(chs[i]=='1') //  i位置为1，会有两种组合方案：a.‘1’当作‘A’，b.i与i+1位置（不论i+1位置为多少）作为’K‘
    {
        //a
        int res =processToString(chs,i+1);
        if(i+1<chs.size()) // i+1不越界
        {
            res +=processToString(chs,i+2);
        }

        return res;
    }
    //
    if(chs[i]=='2') //i位置为2，也会有两种组合方案：a.‘2’当作‘B’，b.i与i+1位置（i+1位置可以为：1，2，3，4，5，6）作为’K‘
    {
        // a
        int res=processToString(chs,i+1);
        if((i+1<chs.size())&&(chs[i+1]>='0' && chs[i+1]<='6')) //i+1没有越界 && i+2
           {
            res +=processToString(chs,i+2);
        }
        return res;
    }

    // chs[i] ='3' ~ '9'
    // i位置为其他值，只能自己一位作为一个作为一个转换方案
    return processToString(chs,i+1);

}
int numStrToString(vector<char> chs)
{
    return processToString(chs, 0);

}
```

```cpp
// 返回所有的组合方案


void processToStringV2(vector<char> chs,int i,string r,vector<string>& res)
{
    //base case
    if (i==chs.size())  // 到了结尾了，返回一种组成方案
    {
        res.push_back(r);
        return;
    }

    if(chs[i]=='0') return;// i位置为0，也就说明前i-1的组成方式，不能进行下去

    if(chs[i]=='1') //  i位置为1，会有两种组合方案：a.‘1’当作‘A’，b.i与i+1位置（不论i+1位置为多少）作为’K‘
    {
        //a
        processToStringV2(chs,i+1,r+'A',res);
        if(i+1<chs.size()) // i+1不越界
        {
            processToStringV2(chs,i+2,r+'K',res);
        }
        return;  //避免重复出现
    }
    //
    if(chs[i]=='2') //i位置为2，也会有两种组合方案：a.‘2’当作‘B’，b.i与i+1位置（i+1位置可以为：1，2，3，4，5，6）作为’K‘
    {
        // a
        processToStringV2(chs,i+1,r+'B',res);
        if((i+1<chs.size())&&(chs[i+1]>='0' && chs[i+1]<='6')) //i+1没有越界 && i+2
           {
               if(chs[i+1]=='0') processToStringV2(chs,i+1,r+'T',res);
               else if (chs[i+1]=='1') processToStringV2(chs,i+1,r+'U',res);
               else if (chs[i+1]=='2') processToStringV2(chs,i+1,r+'V',res);
               else if (chs[i+1]=='3') processToStringV2(chs,i+1,r+'W',res);
               else if (chs[i+1]=='4') processToStringV2(chs,i+1,r+'X',res);
               else if (chs[i+1]=='5') processToStringV2(chs,i+1,r+'Y',res);
               else processToStringV2(chs,i+1,r+'Z',res);

            }

        return; //避免重复出现

    }
```

```
45
46      // chs[i] ='3' ~ '9'
47      // i位置为其他值，只能自己一位作为一个作为一个转换方案
48      int tmp=chs[i]-'0';
49      char ss='A'+ (tmp-1);
50      processToStringV2(chs,i+1,r+ss,res);
51
52  }
53  vector<string> numStrToStringV2(vector<char> chs)
54  {
55  //    return processToString(chs, 0);
56      vector<string> res;
57      processToStringV2(chs,0,"",res);
58      return res;
59
60  }
```

背包问题

```
1   /*
2    给定两个长度都为N的数组 we i ghtsi和 values， weights【i】和 values【i】
3    分别代表号物品的重量和价值。给定一个正数bag，表示一个载重bag的袋子，你装的物品
4    不能超过这个重量。返回你能装下最多的价值是多少?
5    */
6
7   // 考虑i位置选择与否的返回价值
8   int processBagMaxValue(vector<int> weights,vector<int> values,int i,int maxWeight,int
    sumWeight)
9   {
10      // 如果超重了，则i位置没有返回的价值
11      if(sumWeight>maxWeight) return 0;
12      // 如果物品已经被选择完了，i位置就没有选择了，也就没有返回价值
13      if(i>weights.size()-1) return 0;
14
15      // 选择i位置要/不要，返回价值最大的那个选择
16      return
    max(values[i]+processBagMaxValue(weights,values,i+1,maxWeight,sumWeight+weights[i]), //要
17                  processBagMaxValue(weights,values,i+1,maxWeight,sumWeight));  //不要
18  }
19
20  int bagMaxValue(const vector<int> weights, const vector<int> values, int maxWeight)
21  {
22      // return max value.
23      return processBagMaxValue(weights,values,0,maxWeight,0);
24  }
```

n皇后问题