工程框架：采用工厂模式，各种全排列生成算法均实现了接口PermutationGeneration并且继承了Permutation基类。

其中接口PermutationGeneration里面有三类方法，分别为void genPermutation()，用于根据给定的初始序列按某种顺序生成之后的所有排列；void genPermutation(int pos)则是用于根据给定的初始序列按某种顺序生成之后的第pos个排列；void genAllPermutation(int per\_size)用于按照某种顺序生成per\_size大小的所有排列。Permutation基类包含了排列列表和排列长度属性，以及相关的初始化和输出操作。工厂类为PermutationFactory用于根据用户的选择生成不同的方法实例。Util是工具类，里面静态实现了计算十进制数的阶乘，以及转化为递增进位制数和递减进位制数等操作。PermutationTest是测试类，用户首先输入选择的全排列生成算法的序号，接着输入生成排列的长度，控制台即会一次输出所有的全排列。各全排列算法思路如下：

1. 字典序法：

首先根据输入的排列长度，生成初始的数字排列， 接着计算排列的递增进位制表示的中介数(记录当前数字右边比当前数字小的数字个数)，之后分别计算出之后的中介数表示，并将其转化为对应的排列(从左往右依次填入，注意左边已经出现的比前数字小的数字)。

1. 递增进位制数法：

首先根据输入的排列长度，生成初始的数字排列， 接着计算排列的递增进位制表示的中介数(anan-1…a2)↑, *ai*：*i*的右边比*i*小的数字的个数)，之后分别计算出之后的中介数表示，并将其转化为对应的排列(从大到小求出n,n-1,…,2,1的位置,依次填入)。

1. 递减进位制数法：

首先根据输入的排列长度，生成初始的数字排列， 接着计算排列的递减进位制表示的中介数((a2a3…an-1an)↓，*ai*：*i*的右边比*i*小的数字的个数)，之后分别计算出之后的中介数表示，并将其转化为对应的排列(从大到小求出n,n-1,…,2,1的位置,依次填入)。

1. 邻位对换法法：

首先根据输入的排列长度，生成初始的数字排列， 接着计算排列的递减进位制表示的中介数以及各位数的方向(2的方向一定是向左, 对于每一个大于2的数字*i*，如果i为奇数，其方向性决定于bi-1的奇偶性，奇向右、偶向左。如果i为偶数，其方向性决定于bi-1+ bi-2的奇偶性，同样是奇向右、偶向左。当得到方向性后，bi的值就是背向i的方向直到排列边界这个区间里比i小的数字的个数), 之后分别计算出之后的中介数表示，并将其转化为对应的排列(从大到小求出n,n-1,…,2,1的位置以及方向,依次填入)。

公共函数伪代码：

procedure init\_permutation(size)

for i = 1:size do

P[i] = i

end for

return P

end procedure

字典序法伪代码：

procedure lexorder\_permutation(size)

P = init\_permutation(size)

while lex\_end(P) == false do

for i=size-1:1 do

if P[i]<P[i+1] then

break

end if

end for

for j=size:i+1 do

if P[j]>P[i] then

break

end if

end for

swap(P[i],P[j])

mid = (size+i)/2

for k=i+1:mid do

swap(P[k], P[size+i+1-k])

end for

end while

end procedure

procedure lex\_end(P)

for i = 1: P.size-1 do

if P[i] < P[i+1] then

return false

end if

end for

return true

end procedure

递增进位制数法伪代码：

procedure incorder\_permutation(size)

P = init\_permutation(size)

M = get\_incmedia(P)

while(inc\_end(M, size) == false) do

addone\_incmedia(M)

inc\_to\_permutation(M,P)

end while

end procedure

procedure get\_incmedia(P)

for i =1:P.size-1

M [i] = cal\_media(P.size+1-i, P)

end for

return M

end procedure

procedure cal\_media(i, P)

m = 0

for j=1:size do

if P[j]=i then

break

endif

end for

for k=j:P.size do

if P[k]<i then

m = m+1

end if

end for

return m

end procedure

procedure inc\_end(M, size)

for i = 1:M.size do

if M[i] !=size-i then

return false

end if

end for

return true

end procedure

procedure addone\_incmedia(M)

carry = 0

M[M.size]=M[M.size]+1

if M[M.size]==2 then

M[M.size]=0

carry=1

end if

for i=M.size-1:1 do

if carry=1 then

M[i]=M[i]+carry

if M[i]==M.size+2-i then

M[i]=0

carry=1

end if

else then

break

end if

end for

end procedure

procedure inc\_to\_permutation(M,P)

for i=1:P.size do

P[i]=0

end for

for i=1:M.size do

cnt = M[i]+1

for j=P.size:1 do

if P[j]==0 then

cnt= cnt-1

if cnt==0 then

break

end if

end if

end for

P[j]=P.size+1-i

end for

for i=1:P.size do

if P[i]==0 then

P[i]=1

end if

end for

end procedure

递减进位制数法伪代码：

procedure decorder\_permutation(size)

P = init\_permutation(size)

M = get\_decmedia(P)

while(dec\_end(M, size) == false) do

addone\_decmedia(M)

dec\_to\_permutation(M,P)

end while

end procedure

procedure get\_decmedia(P)

for i = 1: M.size do

M[i] = cal\_media(i+1, P)

end for

return M

end procedure

procedure dec\_end(M, size)

for i = 1:M.size do

if M[i] !=i then

return false

end if

end for

return true

end procedure

procedure addone\_decmedia(M)

carry = 0

M[M.size]=M[M.size]+1

if M[M.size]==M.size+1 then

M[M.size]=0

carry=1

end if

for i=M.size-1:1 do

if carry=1 then

M[i]=M[i]+carry

if M[i]==i+1 then

M[i]=0

carry=1

end if

else then

break

end if

end for

end procedure

procedure dec\_to\_permutation(M,P)

for i=1:P.size do

P[i]=0

end for

for i=M.size-1:1 do

cnt = M[i]+1

for j=P.size:1 do

if P[j]==0 then

cnt= cnt-1

if cnt==0 then

break

end if

end if

end for

P[j]=i+1

end for

for i=1:P.size do

if P[i]==0 then

P[i]=1

end if

end if

end procedure

邻位对换法伪代码：

procedure sjtorder\_permutation(size)

P = init\_permutation(size)

get\_sjtmedia\_direct(P, M, D)

while(sjt\_end(D) == false) do

addone\_sjtmedia(M,D)

sjt\_to\_permutation(M,P)

end while

end procedure

procedure get\_sjtmedia\_direct(P, M, D)

D [1] = true

M [1] = cal\_sjtmedia(2, P, D)

for i = 2:P.size do

if( i+1)%2 == 1 then

if M[i-1]%2==1 then

D[i] = false

else then

D [i] = true

end if

else

if (M [i-1]+M [i-2])%2==1 then

D [i]=false

else then

D [i]=true

end if

end if

M [i] = cal\_sjtmedia(i+1,P,D)

end for

end procedure

procedure cal\_sjtmedia(i, P, D)

for j=1:P.size do

if P[j]==i then

break;

end if

end for

m=0

if D[i-1] then

for k=j+1:P.size do

if P[k]<i then

m = m+1

end if

end for

else then

for k=j-1:1 do

if P[k]<i then

m = m+1

end if

end for

end if

return m

end procedure

procedure sjt\_end(P, D)

for i = 1:P.size do

if P[i] != 1 then

if D[i-1] then

if i-1 < 1 or P[i-1] < P[i] then

return false

end if

else then

if i+1 >P.size or P[i+1] < P[i] then

return false

end if

end if

end for

return true

procedure addone\_sjtmedia(M, D)

addone\_decmedia(M)

D[1] = true

for i = 2 to M.size do

if (i+1)%2 == 1 then

if M[i-1]%2==1 then

D [i] = false

else then

D[i] = true

end if

else then

if (M [i-1]+M [i-2])%2==1 then

D [i]=false

else then

D [i]=true

end if

end if

end for

end procedure

procedure sjt\_to\_permutation(M,P)

for i=1:P.size do

P[i]=0

end for

for i=M.size-1:1 do

cnt = M[i]+1

if D[i] then

for j=P.size:1 do

if P[j]==0 then

cnt= cnt-1

if cnt==0 then

break

end if

end if

end for

else then

for j=1:P.size do

if P[j]==0 then

cnt= cnt-1

if cnt==0 then

break

end if

end if

end for

end if

P[j]=i+1

end for

for i=1:P.size do

if P[i]==0 then

P[i]=1

end if

end if

end procedure