Nội dung chính của chương trình bày một số lược đồ thuật toán kinh điển dùng để giải lớp các bài toán liệt kê, bài toán đếm, và bài toán tối ưu và bài toán tồn tại. Mỗi lược đồ thuật toán giải quyết một lớp các bài toán thỏa mãn một số tính chất nào đó. Đây là những lược đồ thuật toán quan trọng nhằm giúp người học vận dụng nó trong khi giải quyết các vấn đề trong tin học. Các lược đồ thuật toán được trình bày trong chương này bao gồm: thuật toán sinh, thuật toán đệ qui, thuật toán quay lui, thuật toán tham lam, thuật toán nhánh cận, thuật toán qui hoạch động.

1. Mô hình thuật toán sinh (Generative Algorithm)

Mô hình thuật toán sinh được dùng để giải lớp các bài toán liệt kê, bài toán đếm, bài toán tối ưu, bài toán tồn tại thỏa mãn hai điều kiện:

- **Điều kiện 1:** *Có thể xác định được một thứ tự trên tập các cấu hình cần liệt kê của bài toán. Biết cấu hình đầu tiên, biết cấu hình cuối cùng.*
- **Điều kiện 2:** *Từ một cấu hình chưa phải cuối cùng, ta xây dựng được thuật toán sinh ra cấu hình đứng ngay sau nó.*

Mô hình thuật toán sinh được biểu diễn thành hai bước: bước khởi tạo và bước lặp. Tại bước khởi tạo, cấu hình đầu tiên của bài toán sẽ được thiết lập. Điều này bao giờ cũng thực hiện được theo giả thiết của bài toán. Tại bước lặp, quá trình lặp được thực hiện khi gặp phải cấu hình cuối cùng. Điều kiện lặp của bài toán bao giờ cũng tồn tại theo giả thiết của bài toán. Hai chỉ thị cần thực hiện trong thân vòng lặp là đưa ra cấu hình hiện tại và sinh ra cấu hình kế tiếp. Mô hình sinh kế tiếp được thực hiện tùy thuộc vào mỗi bài toán cụ thể. Tổng quát, mô hình thuật toán sinh được thể hiện như dưới đây.

Thuật toán Generation;
begin
    Bước1 (Khởi tạo):
        <Thiết lập cấu hình đầu tiên>;
    Bước 2 (Bước lặp):
        while (*<Lặp khi cấu hình chưa phải cuối cùng>*) **do**
            <Đưa ra cấu hình hiện tại>;
            <Sinh ra cấu hình kế tiếp>;
    endwhile;

End.

Ví dụ 2.1. Vector $X = (x_1, x_2, .., x_n)$, trong đó $x_i = 0, 1$ được gọi là một xâu nhị phân có độ dài n. Hãy liệt kê các xâu nhị phân có độ dài n. Ví dụ với n=4, ta sẽ liệt kê được 24 xâu nhị phân độ dài 4 như trong Bảng 2.1.

Bảng 2.1. Các xâu nhị phân độ dài 4

STT	$X=(x_1, x_2, x_3, x_4)$	STT	$X=(x_1, x_2, x_3, x_4)$
0	0 0 0 0	**8**	1 0 0 0
1	0 0 0 1	**9**	1 0 0 1
2	0 0 1 0	**10**	1 0 1 0
3	0 0 1 1	**11**	1 0 1 1
4	0 1 0 0	**12**	1 1 0 0
5	0 1 0 1	**13**	1 1 0 1
6	0 1 1 0	**14**	1 1 1 0
7	0 1 1 1	**15**	1 1 1 1

Lời giải:

Điều kiện 1: Gọi thứ tự của xâu nhị phân $X=(x_1, x_2,.., x_n)$ là $f(X)$. Trong đó, $f(X)= k$ là số chuyển đổi xâu nhị X thành số ở hệ cơ số 10. Ví dụ, xâu $X = (1, 0, 1, 1)$ được chuyển thành số hệ cơ số 10 là 11 thì ta nói xâu X có thứ tự 11. Với cách quan niệm này, xâu đứng sau xâu có thứ tự 11 là 12 chính là xâu đứng ngay sau xâu $X = (1, 0, 1, 1)$. Xâu đầu tiên có thứ tự là *0* ứng với xâu có n số *0*. Xâu cuối cùng có thứ tự là 2^n-1 ứng với xâu có n số *1*. Như vậy, điều kiện 1 của thuật toán sinh đã được thỏa mãn.

Điều kiện 2: Về nguyên tắc ta có thể lấy $k = f(X)$ là thứ tự của một xâu bất kỳ theo nguyên tắc ở trên, sau đó lấy thứ tự của xâu kế tiếp là $(k + 1)$ và chuyển đổi $(k+1)$ thành số ở hệ cơ số 10 ta sẽ được xâu nhị phân tiếp theo. Xâu cuối cùng sẽ là xâu có n số 1 ứng với thứ tự $k = 2^n-1$. Với cách làm này, ta có thể coi mỗi xâu nhị phân là một số, mỗi thành phần của xâu là một bít và chỉ cần cài đặt thuật toán chuyển đổi cơ số ở hệ 10 thành số ở hệ nhị phân. Ta có thể xây dựng thuật toán tổng quát hơn bằng cách xem mỗi xâu nhị phân là một mảng các phần tử có giá trị 0 hoặc 1. Sau đó, duyệt từ vị trí bên phải nhất của xâu nếu gặp số 1 ta chuyển thành 0 và gặp số 0 đầu tiên ta chuyển thành 1. Ví dụ với xâu X = (0, 1, 1, 1) được chuyển thành xâu X= (1, 0, 0, 0), xâu X = (1,0,0,0) được chuyển thành xâu X =(1, 0, 0, 1). Lời giải và thuật toán sinh xâu nhị phân kế tiếp được thể hiện trong chương

trình dưới đây. Trong đó, thuật toán sinh xâu nhị phân kế tiếp từ một xâu nhị phân bất kỳ là hàm *Next_Bits_String*().

```
#include <iostream>
#include <iomanip>
#define MAX 100
using namespace std;
int X[MAX], n, dem =
0; //sử dụng các biến
toàn cục X[], n, OK,
dem bool OK =true;
void Init(void){ //khởi
tạo xâu nhị phân đầu
tiên
        cout<<"Nhập n="; cin>>n;
        for(int i = 1; i<=n; i++) //thiết lập xâu với n số 0
                X[i]=0;
} void Result(void){ //đưa ra xâu nhị phân hiện
tại    cout<<"\n Xâu thứ "<<++dem<<":";
        for(int i=1; i<=n; i++)
           cout<<X[i]<<setw(3);
} void Next_Bits_String(void){ //thuật toán sinh xâu nhị phân kế
tiếp    int i=n;
        while(i>0 && X[i]){ //duyệt từ vị trí bên phải nhất
                X[i]=0; //nếu gặp X[i] = 1 ta chuyển thành 0
                i--; //lùi lại vị trí sau
        }
        if (i>0)  X[i]=1; //gặp X[i] =0 đầu tiên ta chuyển thành 1
else OK = false; //kết thúc khi gặp xâu có n số 1
}
int main(void){ //đây là thuật toán sinh    Init(); //thiết
lập cấu hình đầu tiên        while(OK){//lặp khi chưa
phải cấu hình cuối cùng              Result(); //đưa ra cấu
hình hiện tại
                Next_Bits_String(); //sinh ra cấu hình kế tiếp
```

```
        }
}
```

Ví dụ 2.2. Liệt kê các tập con k phần tử của 1, 2, .., n.

Lời giải. Mỗi tập con k phần tử của 1, 2, .., N là một tổ hợp chập K của 1, 2,.., N. Ví dụ với n=5, k= 3 ta sẽ có C(n,k) tập con trong Bảng 2.2.

Điều kiện 1. Ta gọi tập con X =(x_1,..x_K) là đứng trước tập con Y =(y_1, y_2,..y_K) nếu tìm được chỉ số t sao cho $x_1 = y_1, x_2 = y_2,.., x_{t-1} = y_{t-1}, x_t < y_t$. Ví dụ tập con $X = (1, 2, 3)$ đứng trước tập con $Y = (1, 2, 4)$ vì ta tìm được t=3 thỏa mãn $x_1 = y_1, x_2 = y_2, x_3 < y_3$. Tập con đầu tiên là $X = (1, 2,.., k)$, tập con cuối cùng là $(n–k+1,.., N)$. Như vậy điều kiện 1 của thuật toán sinh được thỏa mãn.

Điều kiện 2. Để ý rằng, tập con cuối cùng (n-k+1,..., n) luôn thỏa mãn đẳng thức X[i] = n – k + i. Ví dụ tập con cuối cùng X[] = (3, 4, 5) ta đều có: X[1] = 3 = 5 – 3 + 1; X[2] = 4 = 5 – 3 + 2; X[3] = 5 = 5 – 3 + 3. Để tìm tập con kế tiếp từ tập con bất kỳ ta chỉ cần duyệt từ phải qua trái tập con X[] = (x1, x2, .., xk) để xác định chỉ số i thỏa mãn điều kiện X[i] n – k + i. Ví dụ với X[] = (1, 4, 5), ta xác định được i=1 vì X[3] = 5 = 5-3+3, X[2] = 4 = 5-3+2, và X[1] = 1 5-3+1. Sau khi xác định được chỉ số i, tập con mới sẽ được sinh là Y[] = (y1,.., yi, ..., yk) ra thỏa mãn điều kiện: y1 = x1, y2 = x2,.., yi-1 = xi-1, yi = xi+1, và yj = xt + j – i với (j = j+1, ..., k).

Bảng 2.2. Tập con 3 phần tử của 1, 2, 3, 4, 5

STT	Tập con
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

Chương trình cài đặt thuật toán sinh tập con k phần tử được thể hiện như dưới đây.

Trong đó, thuật toán sinh tổ hợp kế tiếp có tên là Next_Combination().

```
#include <iostream>
#include <iomanip>
#define MAX 100
int X[MAX], n, k, dem=0;
bool OK = true;
using namespace std;

void Init(void){ //thiết lập tập con đầu tiên
        cout<<"\n Nhập n, k:"; cin>>n>>k;
        for(int i=1; i<=k; i++)  //tập con đầu tiên là 1, 2, .., k
                X[i] = i;
}

void Result(void){ //đưa ra tập con hiện tại  cout<<"\n
        Kết quả "<<++dem<<":";
                for(int i=1; i<=k; i++) //đưa ra X[] =( x1, x2, .., xk)
                cout<<X[i]<<setw(3);
}

void Next_Combination(void){ //sinh tập con k phần tử từ tập con bất kỳ
        int i = k; //duyệt từ vị trí bên phải nhất của tập con
        while(i>0 && X[i]== n-k+i) //tìm i sao cho xi  n-k+i
                i--;
        if (i>0){//nếu chưa phải là tập con cuối cùng
                X[i]= X[i]+1; //thay đổi giá trị tại vị  trí i: xi = xi +1;
        for(int j=i+1; j<=k; j++) //các vị trí j từ i+1,.., k
        X[j] = X[i] + j - i; // được thay đổi là xj = xi +j - i;
        }
        else  //nếu là tập con cuối cùng
                OK = false; //ta kết thúc duyệt
}
int main(void){
        Init(); //khởi tạo cấu hình đầu tiên
```

```
        while(OK){ //lặp trong khi cấu hình chưa phải cuối cùng
        Result(); //đưa ra cấu hình hiện tại
                Next_Combination(); //sinh ra cấu hình kế tiếp
        }
}
```

Ví dụ 2.3. Liệt kê các hoán vị của 1, 2, .., n.

Lời giải. Mỗi hoán vị của 1, 2, .., N là một cách xếp có tính đến thứ tự của 1, 2,..,N. Số các hoán vị là N!. Ví dụ với N =3 ta có 6 hoán vị dưới đây.

Bảng 2.3. Hoán vị của 1, 2, 3.

STT	Hoán vị
1	1 2 3
2	1 3 2
3	2 1 3
4	2 3 1
5	3 1 2
6	3 2 1

Điều kiện 1. Có thể xác định được nhiều trật tự khác nhau trên các hoán vị. Tuy nhiên, thứ tự đơn giản nhất có thể được xác định như sau. Hoán vị $X =(x_1, x_2,.., x_n)$ được gọi là đứng sau hoán vị $Y = (y_1, y_2,..,y_n)$ nếu tồn tại chỉ số k sao cho $x_1 = y_1$, $x_2 = y_2$,…, $x_{k-1} =y_{k-1}$, $x_k<y_k$. Ví dụ hoán vị X = (1, 2, 3) được gọi là đứng sau hoán vị Y =(1, 3, 2) vì tồn tại k =2 để $x_1 = y_1$, và $x_2<y_2$. Hoán vị đầu tiên là X[] = (1, 2, …, n), hoán vị cuối cùng là X[] = (n, n-1, …, 1).

Điều kiện 2. Được thể hiện thông qua hàm Next_Permutation() như chương trình dưới đây.

```
#include <iostream>
#include <iomanip> #define MAX 100 int
X[MAX], n,  dem=0; bool OK = true;
using namespace std; void Init(void){
//thiết lập hoán vị đầu tiên  cout<<"\n
Nhap n:"; cin>>n;
        for(int i=1; i<=n; i++) //thiết lập X[] = (1, 2, ..,n)
                X[i] = i;
```

```
} void Result(void){ //đưa ra hoán vị hiện
tại      cout<<"\n Kết quả "<<++dem<<":";
         for(int i=1; i<=n; i++)
            cout<<X[i]<<setw(3);
}
void Next_Permutation(void){ //sinh ra hoán vị kế tiếp
int j = n-1; //xuất phát từ vị trí j = n-1
         while(j>0 && X[j]>X[j+1]) //tìm chỉ số j sao cho X[j] < X[j+1]
         j--;
         if ( j > 0){ // nếu chưa phải hoán vị cuối cùng
int k = n; //xuất phát từ vị trí k = n
                  while(X[j]>X[k]) //tìm chỉ số k sao cho X[j] < X[k]
                           k--;
                  int t = X[j]; X[j] = X[k]; X[k]=t; //đổi chỗ X[j] cho
X[k]              int r = j+1, s = n;              while (r<=s){ //lật ngược
lại đoạn từ j+1,..,n                    t=X[r]; X[r]=X[s]; X[s]=t;
                           r++; s--;
                  }
         }
         else  //nếu là cấu hình cuối cùng
                  OK = false; //ta kết thúc duyệt
}
int main(void){ //đây là thuật toán sinh
Init(); //thiết lập cấu hình đầu tiên
         while(OK){ //lặp trong khi cấu hình chưa phải cuối cùng
Result(); //đưa ra cấu hình hiện tại
                  Next_Permutation(); //sinh ra cấu hình kế tiếp
         }
}
```

2. Mô hình thuật toán đệ qui (Recursion Algorithm)

Một đối tượng được định nghĩa trực tiếp hoặc gián tiếp thông qua chính nó được gọi là phép định nghĩa bằng đệ qui. Thuật toán giải bài toán *P* một cách trực tiếp hoặc gián tiếp

thông qua bài toán *P*' giống như *P* được gọi là thuật toán đệ qui giải bài toán *P*. Một hàm được gọi là đệ qui nếu nó được gọi trực tiếp hoặc gián tiếp đến chính nó.

Tổng quát, một bài toán có thể giải được bằng đệ qui nếu nó thỏa mãn hai điều kiện:

- **Phân tích được**: *Có thể giải được bài toán P bằng bài toán P' giống như P. Bài tóa P' và chỉ khác P ở dữ liệu đầu vào. Việc giải bài toán P' cũng được thực hiện theo cách phân tích giống như P.*
- **Điều kiện dừng**: *Dãy các bài toán P' giống như P là hữu hạn và sẽ dừng tại một bài toán xác định nào đó.*

Thuật toán đệ qui tổng quát có thể được mô tả như sau:

Thuật toán Recursion (P) {

1. *Nếu P thỏa mãn điều kiện dừng:*
   <Giải P với điều kiện dừng>;
2. Nếu *P* không thỏa mãn điều kiện dừng:
   <Giải P' giống như P:Recursion(P')>;

}

Ví dụ 2.4. Tìm tổng của n số tự nhiên đầu tiên bằng phương pháp đệ qui.

Lời giải. Gọi S_n là tổng của n số tự nhiên. Khi đó:

- **Bước phân tích**: dễ dàng phận thấy tổng n số tự nhiên $S_n = n + S_{n-1}$, với n□1. • **Điều kiện dừng**: $S_0 = 0$ nếu n = 0;

Từ đó ta có lời giải của bài toán như sau:

```
int     Tong (int  n ) {
        if (n ==0 ) return(0); //Điều kiện dừng
        else return(n + Tong(n-1)); //Điều kiện phân tích được
}
```

Chẳng hạn ta cần tìm tổng của 5 số tự nhiên đầu tiên, khi đó:

S = Tong(5)
= 5 + Tong(4)
= 5 + 4 + Tong(3)
= 5 + 4 + 3 + Tong(2)
= 5 + 4 + 3 + 2+ Tong(1)
= 5 + 4 + 3 + 2 + 1 + Tong(0)

= 5 + 4 + 3 + 2 + 1 + 0
= 15

Ví dụ 2.5. Tìm n!.

Lời giải. Gọi Sn là n!. Khi đó:

- **Bước phân tích**: Sn = n*(n-1)! nếu n>0;
- **Điều kiện dừng**: s0=1 nếu n=0.

Từ đó ta có lời giải của bài toán như sau:

```
long   Giaithua (int  n ) {
        if (n ==0 ) return(1); //Điều kiện dừng
        else return(n *Giaithua(n-1)); //Điều kiện phân tích được
        } Ví dụ 2.6. Tìm ước số chung lớn nhất của a và b bằng
```

phương pháp đệ qui.

Lời giải. Gọi d =USCLN(a,b). Khi đó:

- **Bước phân tích**: nếu b□0 thì d = USCLN(a, b) = USCLN(b, r), trong đó a =b, b = r = a mod b .
- **Điều kiện dừng**: nếu b = 0 thì a là ước số chung lớn nhất của a và b. Từ đó ta có lời giải của bài toán như sau:

```
int          USCLN (int a, int b ) {      if (a ==b )
return(a); //Điều kiện dừng          else { //Điều kiện phân tích được
        int r = a % b; a = b; b = r;          return(USCLN(a, b)); //giải bài toán
USCLN(a, b)
        }
}
```