CHƯƠNG 1: ĐỘ PHỨC TẠP

- Ta có:
  1. f(n) có tốc độ tăng $\leq$ g(n) <=> f(n) = O(g(n)) (hoặc f(n) ∈ O(g(n)))

     <=> ∃ c > 0 và N ≥ 0 sao cho f(n) ≤ c.g(n) , ∀n ≥ N

     <=> $\lim_{n\to\infty}\frac{f(n)}{g(n)} \leq$ c và 0
  2. f(n) có tốc độ tăng $\geq$ g(n) <=> f(n) = Ω(g(n)) (hoặc f(n) ∈ Ω(g(n)))

     <=> ∃ c > 0 và N ≥ 0 sao cho f(n) ≥ c.g(n) , ∀n ≥ N

     <=> $\lim_{n\to\infty}\frac{f(n)}{g(n)} \geq$ c và 0
  3. f(n) có tốc độ tăng = g(n) <=> f(n) = Θ(g(n)) (hoặc f(n) ∈ Θ(g(n)))

     <=> $\begin{cases} f(n) = O(g(n)) \\ f(n) = \Omega(g(n)) \end{cases}$

     <=> $\lim_{n\to\infty}\frac{f(n)}{g(n)} = \lim_{n\to\infty}\frac{g(n)}{f(n)} = c$

- Nếu T1(n) = O(f(n)) và T2(n) = O(g(n)) thì
  - $T_1(n)+T_2(n) = O(\max(f(n), g(n)))$
  - $T_1(n).T_2(n) = O(f(n).g(n))$
  - c.O(f(n)) = O(f(n))
  - O(c) ≡ O(1)
- $x - 1 \leq \lfloor x \rfloor \leq x \leq \lceil x \rceil \leq x + 1$ (x là số thực)
- Merge-sort:

  $$T(n) = \begin{cases} O(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n), & n > 1 \end{cases}$$

  Giải sử n=2k, coi O(1)=1 và O(n) = n, thì:

  $T(n) = 2T(n/2)+ n = \ldots = 2^kT(n/2^k) + kn = n + n\log_2 n = O(n\log_2 n)$

  - Với n ∈ N, a, b, c là số thực dương: $a^{\log_b n} = n^{\log_b a}$

CHƯƠNG 2: TRỰC TIẾP VÀ VÉT CẠN (BRUTE-FORCE)

• ***Bài toán tính tổng $S=1^2+2^2+\ldots+n^2$***

```
ALGORITHM Sum(n)
S ←0
for i ←1 to n do
    S ←S+i*i
return S
```

T(n) = O(n)

✦ *Có thể áp dụng chiến lược biến đổi để trị để giảm độ ptạp của giải thuật*

Ta có: $1+2^2+3^2+\ldots+n^2 = \dfrac{n(n+1)(2n+1)}{6}$

```
ALGORITHM Sum2(n)
return (n*(n+1)*(2*n+1))/6
```

T(n) = O(1)

• ***Tính a^n***

✦ *brute*-force

```
ALGORITHMS power_brute-force(a,n)
kq ← 1
if n = 0 return 1
for i ← 1 to n do
        kq ← kq * a
Return kq
```

T(n) = O(n)

✦ *chia để trị*

```
ALGORITHMS power_devide-and-conquer(a,n)
if n = 0 return 1
```

```
else t←power_devide-and-conquer(a, ⌊n/2⌋)
    if n mod 2 = 0
        then return t*t
    else return t*t*a
```

Hệ thức truy hồi

$$T(n) = \begin{cases} O(1), & n = 0 \\ T\left(\frac{n}{2}\right) + O(1), & n > 0 \end{cases}$$

=> $T(n) = O(\log_2 n)$

- ***Tính đa thức*** $p(x)=a_n x^n + a_{n-1}x^{n-1} + \ldots + a_1 x + a_0$

✦ *Brute-force*

ALGORITHM px(a[0..n-1],x)

```
p←1
m←1
for i ←0 to n – 1 do
    m←m*x
    p←p+a[i]*m
return p
```

$T(n) = O(n)$

✦ *Biến đổi để trị (dùng Horner)*

ALGORITHM px_horner(a[0..n-1],x)

```
p←a[n]
for i ←0 to n – 1 do
    p←p*x+a[i]
return p
```

$T(n)=O(n)$

✦ *Nếu các hệ số a_0,… đều =1, ta có thể áp dụng chiến lược biến đổi để trị để giảm độ ptạp của giải thuật*

$$p(x) = x^n + x^{n-1} + \ldots + x + 1 = \frac{x^{n+1}-1}{x-1}$$

ALGORITHM px_transform(x)

if x=1 return n+1

else return $\frac{\text{power(x,n+1)}-1}{\text{x}-1}$

Nếu giải thuật tính power(x,n+1) được viết bằng chiến lược chia để trị thì độ ptạp của giải thuật trên là:

$T(n) = O(\log_2 n)$

• ***Giải thuật sắp xếp chọn trực tiếp (Selection Sort)***

ALGORITHM SelectionSort(A[0..n − 1])

//Input: An array A[0..n − 1] of orderable elements

//Output: Array A[0..n − 1] sorted in nondecreasing order

for i ←0 to n − 2 do

    min←i

    for j ←i + 1 to n − 1 do

        if A[j]<A[min] min←j

    swap A[i] and A[min]

$T(n) = (\sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} 1)c = (\sum_{i=0}^{n-2} n - i - 1)c = (n-1 + n-2 + \ldots + 1)c = n(n-1)/2\ c$

=> $T(n) = O(n^2)$

• ***Giải thuật tìm kiếm tuần tự (Sequential Search)***

ALGORITHM SequentialSearch(A[1..n], K)

for i ← 1 to n

do   if A[i] = K

    then return true

return false

$T(n)=O(n)$

- ***Bài toán so trùng mẫu của chuỗi ký tự (String Matching)***

```
ALGORITHM BruteForceStringMatch(T [0..n − 1], P[0..m − 1])
for i ←0 to n − m do
    j ←0
    while j <m and P[j ]= T [i + j ] do
        j ←j + 1
    if j = m return i
return −1
```

Tốt nhất T(n) = O(m)

Xấu nhất T(n)=O(nm)

- ***Đếm số chuỗi bắt đầu bằng 'A'và kết thúc là 'B' trong chuỗi cho trước***

```
ALGORITHMS substring_count_Linear (S)
countSub ← 0, count ← 0
for i ← 0 to n-1 do
    if S[ i ] == 'A'
        count ← count +1
    if S[ i ] == 'B'
        countSub ← countSub + count
return countSub
```

T(n) = O(n)

- ***Bài toán tìm cặp điểm gần nhất (Closest-Pair)(không gian 2 chiều)***

ALGORITHM BruteForceClosestPair(P)

///Input: A list P of n ($n \geq 2$) points $p_1(x_1, y_1), \ldots, p_n(x_n, y_n)$

//Output: The distance between the closest pair of points

```
d←∞
for i ←1 to n − 1 do
```

for j ←i + 1 to n do

d ←min(d, sqrt((x_i – x_j)^2 + (y_i – y_j)^2)) //sqrt is square root

return d

T(n) = $\left(\sum_{i=1}^{n-1}\sum_{j=i+1}^{n} 2\right)$c =2($\sum_{i=1}^{n-1} n - i$)c = n(n-1)c = O($n^2$)

✦ *Ta có thể giảm độ ptạp bằng giải thuật chia để trị*

//Giả sử P được sắp tăng theo hoành độ

//Gọi Q là tập điểm đã cho được sắp tăng theo tung độ y

ALGORITHM EfficientClosestPair(P, Q, n)

if n ≤ 3

return the minimal distance found by the brute-force algorithm

else

copy the first ⌈n/2⌉ points of P to array P_l

copy the same ⌈n/2⌉ points from Q to array Q_l

copy the remaining ⌊n/2⌋ points of P to array P_r

copy the same ⌊n/2⌋ points from Q to array Q_r

d_l←EfficientClosestPair(P_l, Q_l, ⌈n/2⌉)

d_r←EfficientClosestPair(P_r, Q_r, ⌊n/2⌋)

d ←min{d_l, d_r}

m←P[⌈n/2⌉ − 1].x

copy all the points of Q for which |x − m| < d into array S[0..num − 1]

dminsq ←d^2

for i ←0 to num − 2 do

k←i + 1

while k ≤ num − 1 and (S[k].y − S[i].y)^2 < dminsq

```
            dminsq ←min((S[k].x − S[i].x)^2+ (S[k].y − S[i].y)^2, dminsq)
            k←k + 1
return sqrt(dminsq)
```

$T(n)=O(n\log_2 n)$

- ***Bài toán tìm cặp điểm gần nhất trong không gian k chiều***

```
ALGORITHMS Brute-forceKdimention (P[1...n, 1...k])
d ← ∞
for i ← 0 to n-2 do
        S ← 0
        For j ← i+1 to n - 1 do
            For m ← 1 to k do
                        S ← S + [ P[ j,m ] - P[ i,m ] ]^2
                d ← min (d, sqrt(S))
Return d
```

$$T_{(n,k)} = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1}\sum_{s=1}^{k} 1 = \sum_{i=0}^{n-2}\sum_{j=i+1}^{n-1} k = k\sum_{i=0}^{n-2}(n-i)$$

$$= k\,[\,(n-2) + (n-3) + \ldots + 1]$$

$$= (k.n(n-1))/2 = O(kn^2)$$

- ***Bài toán người đi du lịch (Traveling Salesman)***

```
ALGORITHM  Traveling Salesman(t[1…n])
d←∞, π ← Ø
    for i ←1 to (n-1)! do
            compute (p[2], …, p[n]) // một hoán vị của 2, 3, …, n
            min← d(t_1, t_p[2])+ d(t_p[2], t_p[3])+…+d(t_p[n], t_1)
            if min<d
                    min ←d
                    π ← t_1, t_p[2], …, t_p[n], t_1
```

8 return π

$T(n) = (n-1)!c=O((n-1)!)$

✦ *Có thể áp dụng giải thuật “tham ăn” để giảm độ ptạp*

```
ALGORITHM Traveler(T, w, s) //s là thành phố xuất phát
1. P ← (s); Q ← {s}; u ← s
3. while length[P]<|T|
4.        do      e ←  (u, v) with w(e) = min{w(u, x)| x ∈Adj[u] and x∉Q}
5.                P ← P⊗v    // Thêm v vào đường đi P
7.        Q ←Q ∪ {v}
6         u←v
7. P ← P⊗s      // Trở về đỉnh xuất phát
8. return P
```

$T(n)=O(n^2)$

CHƯƠNG 3: CHIA ĐỂ TRỊ

• ***Tìm kiếm nhị phân***

```
ALGORITHM BinarySearch(A[0..n − 1], K)
1         l←0; r ←n − 1
2         while l ≤ r do
3                 m←⌊(l + r)/2⌋
4                 if K = A[m] return m
5                 else     if K <A[m] r ←m − 1
6                                 else l←m + 1
7         return −1
```

Tốt nhất: $T(n)= O(1)$

Xấu nhất: $T(n)=O(\log_2 n)$

Trung bình: $T(n)=O(\log n)$

• ***Giải thuật MergeSort***

Merge-sort(A,p,r)

```
if p<r
then  q←⌊(p+r)/2⌋
      Merge-sort(A,p,q)
      Merge-sort(A,q+1,r)
      Merge(A,p,q,r)
```

Merge(A,p,q,r)

```
n1←q-p+1
n2←r-q
create array L[1.. n1+1] and R[l...n2+1]
for i←1 to n1 do
      L[i]←A[p+i-1]
for j←1 to n2 do
      R[j]←A[q+j]
      L[n1+1]←∞
      R[n2+1]←∞
i←1
j←1
for k←p to r do
      if L[i] ≤ R[j]
      then A[k]←L[i]
           i←i+1
      else A[k]←R[j]
           j←j+1
```

-Thời gian chạy:

$$T(n) = \begin{cases} O(1), & n = 1 \\ 2T\left(\frac{n}{2}\right) + O(n), & n > 1 \end{cases}$$

Giải sử n=2k, coi O(1)=1 và O(n) = n, thì:

$T(n) = 2T(n/2)+ n = \dots = 2^k T(n/2^k) + kn = n + n\log_2 n = O(n\log_2 n)$

- ***Giải thuật QuickSort (đọc Levitin)***
- ***Bài toán nhân các số nguyên lớn (n chữ số, n lớn và chẵn)***

ALGORITHM Product(a, b, n)

$a \leftarrow r_{n-1}r_{n-2}\ldots r_{n/2}r_{n/2-1}\ldots r_1r_0$; $b \leftarrow p_{n-1}p_{n-2}\ldots p_{n/2}p_{n/2-1}\ldots p_1p_0$

if n=1 return $r_0 * p_0$

else $a_1 \leftarrow r_{n-1}r_{n-2}\ldots r_{n/2}$; $a_0 \leftarrow r_{n/2-1}\ldots r_1r_0$

$b_1 \leftarrow p_{n-1}p_{n-2}\ldots p_{n/2}$; $b_0 \leftarrow p_{n/2-1}\ldots p_1p_0$

$c_2 \leftarrow$Product(a_1, b_1, n/2); $c_0 \leftarrow$ Product(a_0, b_0, n/2)

$A \leftarrow a_1+a_0$; $B \leftarrow b_1+ b_0$;

$c_1 \leftarrow$ Product(A, B, n/2)- $(c_2 + c_0)$

return $c_2 10^n + c_1 10^{n/2} + c_0$

- ***Bài toán tìm cặp điểm gần nhau nhất (không gian 2 chiều) (xem chương 2)***
- ***Tìm vị trí phần tử lớn nhất trong mảng***

ALGORITHM MaxIndex(a[1...n])

```
if l=r return l
else    i←MaxIndex(a[l..⌊(l+r)/2⌋])
        j←MaxIndex(a[⌊(l+r)/2⌋+1..r])
        if a[i] > a[j]
            then return i
        else return j
```

$$T(n) = \begin{cases} O(1), n = 1 \\ 2T\left(\left\lceil \frac{n}{2} \right\rceil\right) + O(1), n > 1 \end{cases}$$

Áp dụng công thức $T(n)=O(n^{\log_2 2}) = O(n)$

- ***Tìm phần tử lớn nhất và nhỏ nhất trong mảng***

    ✦ *Chia để trị*

ALGORITHMS MinMax (A[1..n], min, max)

```
l ← 0, r ← n-1
if l = r then
        min ← a[l] , max ←  a[l]
else  if r - l =1 then
            if a[r] ≤ a[l] then
                        min ← a[r], max ← a[l]
            else        min ← a[l], max ← a[r]
else  MinMax (a[l,.., ⌊(l+r)/2⌋], min, max)
        MinMax (a[(⌊(l+r)/2⌋ +1),.., r], min2, max2)
        if  min2 < min        min ← min2
        if  max2 > max        max ← max2
```

$$T(n) = \begin{cases} O(1), n \leq 1 \\ 2T\left(\lceil \frac{n}{2} \rceil\right) + O(1), n > 1 \end{cases} = O(n)$$

✦ *Brute-Force*

ALGORITHMS MinMax_brute-force (A[1..n], min, max)

```
min ← ∞ , max ←  0
for i←1 to ndo
     if a[i] < min then
             min ← a[i]
     if a[i] > max then
             max ← a[i]
```

T(n)=O(n)

✦ *Biến đổi để trị*

ALGORITHMS ResortingMinMax (A[1..n],min,max)

```
Sắp xếp mảng bằng giải thuật MergeSort
min ← A[0]
max ← A[n-1]
```

$T(n)=O(n\log_2 n)$

• ***Sắp xếp các phần tử trong mảng (âm trước dương sau)***

```
ALGORITHMS  SortArray(A[0..n-1])
l ← 0, r ← n-1
while l ≤ r do
        if A[l] < 0 then
            l ← l +1
        else     swap (A[l], A[r])
                 r ←  r -1
return A
```

Thời gian thực hiện giải thuật: T(n) = O(n)

- ***Tìm 2 điểm gần nhau nhất trong không gian 1 chiều***

✦ *Chia để trị*

```
ALGORITHM ClosestPair(a[],l,r)//sắp xếp các ptử trong mảng tăng dần
if l=r return -1
else     if l – r = 1
                return l
else
         i←ClosestPair(a[],l,(l+r)/2)
         j←ClosestPair(a[],(l+r)/2 + 1,r)
         if a[i+1] – a[i] < a[(l+r)/2 + 1] – a[(l+r)/2] and a[i+1] – a[i] < a[j+1] – a[j]
                return i
         else if a[j+1] – a[j] < a[(l+r)/2 + 1] – a[(l+r)/2] and a[j+1] – a[j] < a[i+1] – a[i]
                return j
         else return (l+r)/2
```

$$T(n) = \begin{cases} O(1)\,, n < 3 \\ 2T(n/2) + O(1)\,, n \geq 3 \end{cases}$$

Ta có: $T(n) = O(n^{\log_2 2}) = O(n)$

✦ *Biến đổi để trị*

```
ALGORITMS PresortingClosestPair (A[1..n])
Sắp xếp mảng không giảm bằng MergeSort
min ← ∞
```

```
for i ← 0 to n-1 do
    If (A[i+1] - A[i] > min)
            min ← A[i+1] – A[i]
return min
```

Thời gian chạy: $T(n) = O(n\log_2 n) + O(n) = O(n\log_2 n)$

✦ *Brute-force*

```
ALGORITMS BruteForceClosestPair (A[1..n])
min ← ∞
For i ← 1 to n-1 do
    For  j ← i+1 to n do
        If (A[j] - A[i] > min)
            min ← A[j] – A[i]
Return min
```

Thời gian chạy $T(n) = O(n^2)$

CHƯƠNG 4: BIẾN ĐỔI ĐỂ TRỊ

• ***Bài toán kiểm tra tính duy nhất của một phần tử***

✦ *Biến đổi để trị (sắp xếp tăng dần mảng trước)*

```
ALGORITHM PresortElementUniqueness(A[0..n − 1])
Sort the array A
for i ←0 to n − 2 do
if A[i]= A[i + 1] return false
return true
```

$T(n)=O(n) + O(n\log_2 n) = O(\log_2 n)$

Nếu dùng brute-force => $T(n)=O(n^2)$

• ***Giải thuật HeapSort***

```
MaxHeapify(A[1..n],  i)
l ← 2*i; r ← 2*i+1
```

```
2    if l ≤ n and A[l]>A[i]  largest ← l
3    else largest ← i
4    if r ≤ n and A[r]>A[largest]  largest ← r
5    if largest ≠ i
6           Exchange(A[i], A[largest])
7           MaxHeapify(A, largest)
```

```
BuildMaxHeap(A[1..n])
1   for i ← n/2 downto 1
2  do MaxHeapify(A, i)
```

```
HeapSort(A[1..n])
1   BuildMaxHeap(A)
2   for i ← n downto 2
3       do Exchange(A[1], A[i])
4             MaxHeapify(A[1..i-1], 1)
```

Thời gian chạy của MaxHeapify là $O(h)=O(\log_2 n)$

Thời gian chạy của BuilMaxHeap tối đa là $O(n\log_2 n)$

Thời gian chạy của vòng lặp 2-4 là $O(n\log_2 n)$

=> Vậy thời chạy của HeapSort là

$T(n)= O(n\log_2 n)+O(n \log_2 n)=O(n\log_2 n)$

- ***Qui tắc Horner (Xem chương 2)***

  Biểu diễn đa thức thành dạng thừa số

- ***Tính tổng*** $S=1^3+2^3+\ldots+n^3=n^2(n+1)^2/4 \Rightarrow T(n) = O(1)$

- ***Tính bội số chung nhỏ nhất (dựa vào ước chung lớn nhất)***

ALGORITHM lcm(m, n)

```
return n*m/gcd(m, n)
```

//tìm ước chung lớn nhất

```
ALGORITHM gcd (m, n)
//Input: Two nonnegative, not-both-zero integers m and n
//Output: Greatest common divisor of m and n
while n ≠ 0 do
      r ←m mod n
      m←n
      n←r
return m
```

$T(n)=O(\log_{3/2}(2.\max(m, n)/3) \approx O(\log_2 n)$

- ***Tìm phần tử giao nhau giữa 2 mảng***

  ✦ *Biến đổi để trị*

```
ALGORITHM IntersectionPresorting-based(A[1..n], B[1..m])
C←Ø
Merge-sort(A[1..n])
for i←1 to m do
j←BinarySearch(A[1..n ], B[i])
if (j > -1)
        C ← C ∪ A[j]
```

$T(n) = O(n\log n) + mO(\log n) = O((m+n) \log n)$

```
ALGORITHM IntersectionPresorting-based_2(A[1..n], B[1..m])
C←Ø
Merge-sort(A[1..n])
Merge-sort(B[1..m])
i ← 1
j ← 1
```

```
while (i< n+1 and j< m+1) do
    if (A[i] < B[j] )
        then    i ← i + 1
    else    if ( A[i] > B[j] )
            then    j ← j + 1
        else    if( A[i] = B[j] )
            then    C ← C ∪ A[i]
                    i ← i + 1
                    j ← j + 1
```

T(n) = O(n log n)+ O(mlogm)+ O(n +m)= O(s log s) với s =max{n,m}.

✦ *Brute-force*

```
ALGORITHM IntersectionBrute-force(A[1..n], B[1..m])
C←Ø
for i←1 to n do
    for j←1 to m do
        if (A[i] = B[j])
        then    C ← C ∪ A[i]
        B ← B \ { B[j] }
```

T(n) = O(nm)

• ***Tìm 2 phần tử có tổng bằng s trong mảng***

```
ALGORITHM FindingPairSum(a[1..n],s)

sort(a)
i←0
j←n-1
while(i<j) do
    if a[i]  + a[j]=s then return true
    else if a[i] + a[j] < s then j←j-1
    else a[i] + a[j] > s then i←i+1
return false
```

$T(n) = O(n\log_2 n)$

- ***Tìm 2 số lớn nhất có tổng = n***

Nếu n=2k thì x=y=k và $xy=k^2$

Ngược lại

$x=\lfloor n/2 \rfloor$ và $y = n - \lfloor n/2 \rfloor$

T(n) = O(1)

- ***Các công thức dùng tính tổng***

$1+2+3+\ldots+n = \frac{n(n+1)}{2}$

$1+2^2+3^2+\ldots+n^2 = \frac{n(n+1)(2n+1)}{6}$

$1+2^3+3^3+\ldots+n^3 = \left[\frac{n(n+1)}{2}\right]^2$

$1+3+5+7+...+(2n-1) = n^2$

$1^2+3^2+5^2\ldots+(2n+1)^2 = \frac{(2n+1)(2n+2)(2n+3)}{6}$

2+4+6+8+...+2n-1 = n(n+1)

$2^2+4^2+6^2\ldots+(2n)^2 = \frac{2n(2n+1)(2n+2)}{6}$

$-1+3-5+7-9+(-1)^n(2n-1) = (-1)^n n$

$1+x+x^2+...+x^n = \frac{x^{n+1}-1}{x-1}$ đk:x<>0,1

$1.2+2.3+3.4+...+n(n+1) = \frac{n(n+1)(n+2)}{3}$

$1n+2(n-1)+3(n-2)+\ldots..+n= \frac{n(n+1)(n+2)}{6}$

$1.2.3+2.3.4+3.5.6+...+n(n+1)(n+2) = \frac{n(n+1)(n+2)(n+3)}{4}$

$1.2^0+2.2^1+3.2^2+\ldots+n.2^{(n-1)} = (n-1).2^n+1$

$$\frac{1}{1.2}+\frac{1}{2.3}+\frac{1}{3.4}+\cdots+\frac{1}{n(n+1)}=\frac{n}{n+1}$$

$$\frac{1}{\sqrt{1}+\sqrt{2}}+\frac{1}{\sqrt{2}+\sqrt{3}}+\frac{1}{\sqrt{3}+\sqrt{4}}+\cdots+\frac{1}{\sqrt{n}+\sqrt{(n+1)}}=\sqrt{n+1}-\sqrt{n}$$

CHƯƠNG 5: QUY HOẠCH ĐỘNG

• ***Bài toán tính số Fibonacci thứ n (trực tiếp + biến đổi để trị)***

✦ *brute-force*

ALGORITHMS Fibonacci(n)

1. if n ≤ 1
2. return n
3. return Fibonacci (n-1) + Fibonacci(n-2)

✦ *Quy hoạch động*

ALGORITHM Fibonacci(n)

1 f[0] ← 0

2 f[1] ← 1

3 for i ←2 to n

4 do f[i] ← f[i-1] +f[i-2]

5 return f[n]

✦ *Biến đổi để trị*

ALGORITHMS climbstair(F[n])

1. φ ← ((1 + sqrt(5))/2
2. Fn ← (φ^n + (1 – φ)^n)/sqrt (5)
3. Return Fn

• ***Bài toán dãy các đồng xu***

Hệ thức truy hồi:

$$F(n)=\begin{cases} 0 \text{ nếu } n=0 \\ c_1 \text{ nếu } n=1 \\ \max\{F(n-2)+c_n, F(n-1)\} \text{ nếu } n>1 \end{cases}$$

```
ALGORITHM CoinRow(C[1..n])
F[0]←0; F[1]←C[1]
for i ←2 to n do
    F[i]←max(C[i]+ F[i − 2], F[i − 1])
return F[n]
```

T(n)=O(n)

- ***Bài toán robot nhặt các đồng xu***

```
ALGORITHM RobotCoinCollection(C[1..n, 1..m])
F[1, 1]←C[1, 1]
for j ←2 to m do
F[1, j]←F[1, j − 1]+ C[1, j]
for i ←2 to n do
    F[i, 1]←F[i − 1, 1]+ C[i, 1]
for j ←2 to m do
        F[i, j ]←max(F[i − 1, j], F[i, j − 1]) + C[i, j ]
return F[n, m]
```

T(n) =(n-1)(m-1)c=Θ(nm)

Thời gian tìm đường đi khi đã có bảng kết quả là Θ(n+m)

- ***Bài toán cái túi***

```
ALGORITHM  knapsack(v[1..n], w[1..n], W)
for i ←0 to n do F(i, 0) ←0
for j ←1 to W do F[0, j]←0
for i ←1 to n do
        for j ←1 to W do
```

```
        if j-wi ≥ 0
        then  F[i, j ]←max(F[i−1, j], vi+ F[i-1, j-wi])
        else F[i, j ] ← F(i-1, j)
return F[n, W]
```

T(n)=O(nW)

- ***Bài toán xâu con chung dài nhất (tham khảo)***
- ***Tìm dãy con liên tiếp có tổng lớn nhất***

```
ALGORITHM summax(a[1..n])
    max_ending_here = max_so_far = 0
    for i←1 to n do
        max_ending_here ← max(a[i], max_ending_here + a[i])
        max_so_far ← max(max_so_far, max_ending_here)
    return max_so_far
```

⇨ T(n) = O(n)

- ***Tính hệ số nhị thức C(k,n)***

```
ALGORITHM Combination(n, k)
    for j ← 0 to n do
        C[0][j]←1
        C[j][j]←1
    for i ←1 to k do
        do for j ← i+1 to n do
            C[i][j]←C[i][j-1] + C[i-1][j-1]
    return C[k][n]
```

```
ALGORITHM Combination2(n, k)
    a ← Ø
    for i ← to n do
        if i ≤ k then
            a [i] ← 1
            u ← i – 1
```

```
        else u ← k
        for j ← u downto do
                a[j] ← a[j - 1] − a[j]
    return a[k]
```

CHƯƠNG 6: THAM ĂN

• ***Giải thuật Kruskal***

```
ALGORITHM Kruskal(G, w)
F ← Ø; Q ←E[G]; N ← V[G]
while |F|<|N|-1 and Q ≠ Ø
    do  e ← Extractmin(Q)         // e có trọng số bé nhất
        if F∪{e} not contain cycle then F ← F∪{e}
        if |F|<|N|-1
            then G is not connected
        else return T  // T= (V, F)
```

$T(n)=O(V\log_2 E)$

• ***Giải thuật Dijkstra***

```
RELAX(u, v, w)
if d[v] > d[u] + w(u, v)
    then d[v] ← d[u] + w(u, v)
         π[v] ← u
```

```
INITIALIZE-SINGLE-SOURCE(G, s)
for each vertex v ∈ V[G]
    do d[v] ← ∞
       π[v] ← NIL
d[s] ← 0
```

```
DIJKSTRA(G, w, s)
1  INITIALIZE-SINGLE-SOURCE(G, s)
2  S ← Ø
3  Q ← V[G]
4  while Q ≠ Ø
5      do u ← EXTRACT-MIN(Q)
6         S ← S ∪ {u}
7         for each vertex v ∈ Adj[u]
8             do RELAX(u, v, w)
```

Vậy tổng chi phí là O(VlgV) +O(E) (nếu |E|>VlgV thì coi lệnh 8 là cơ bản, ngược lại là lệnh 5)

• ***Bài toán người du lịch (xem chương 2)***

• ***Bài toán tô màu***

```
Greedy(G)
Newclr ← Ø
for each uncolored vertex v of G do
    if v is not adjacent to any vertex in Newclr
    then  Newclr ← Newclr ∪ {v}  //greedy
return Newclr
```

```
ColoringGraph(G)
C ← Ø; N ← Ø
while V[G] ≠ Ø  do
    C ← Greedy(G)
    Coloring every v in C the same color k□N
    V[G] ← V[G]-C
    N ← N ∪ {k}
return N   // tập màu ít nhất có thể tô
```

• ***Phân công công việc***

```
ALGORITHMS Assignment ( C[1..n][1..n])

For  i ← 1 to n
        do for j ← 1 to n
                do  job[ i ][ j ] ← 0
For i ← 1 to n
         do  min ← ∞ , temp ← 0
          for j ← 1 to n
              do if (job[ i ][ j ] == 0 and C[ i ][ j ] < min)
                      then  min ← C[ i ][ j ]
                               temp ← j
          do job[ i ][temp] ← 1
```

• ***Cây bao trùm lớn nhất***

```
ALGORITHMS MaxSpanningTree (G, w)

F = ∅, Q = E[G], N = V[G]
While | F | < | N | -1 and Q ≠ ∅
        Do
                 e ← extramax(Q) // chọn cạnh lớn nhất trong Q và loại ra khỏi đồ thị
                 If (not_contain_cycle (F v {e})
                      Then F ← F v {e}
                 If (| F | < | N | -1)
                        then return null // ko liên thông
                    Else return F
```

CHƯƠNG 7: QUAY LUI VÀ NHÁNH CẬN

Đặc trưng

```
BackTracking(x[1..k], n) // xác định x[k], k nguyên

for j ←1 to n_k // xét khả năng j, trong n_k khả năng
    do if accepting j
               then  <computing x[k] in A_k that subjects to j>
                          if k = n
```

```
then < recording 1 solution >
else BackTracking(x[1..k+1], n)
```

• ***Tìm tất cả các xâu nhị phân n bits***

```
BanaryBackTracking(b[1..k], n)
for j ← 0 to 1  // đồng nhất các khả năng j với các giá trị
    do // bk có thể nhận trong Ak={0,1}
        b[k] ← j
        if k = n
            then Print(b[1..k])
            else return BanaryBackTracking(b[1..k+1], n)
```

T(n)=2T(n-1)+O(1) = O(2n)

• ***Tìm tất cả các xâu nhị phân n bits không chứa 2 bit 0 liên tiếp***

```
ALGORITHM no00BinaryBackTracking(b[1..k],n)
    for j←0 to 1
    do
        if ( b[k-1] + j ≠ 0) then b[k]=j;
        if k=n
            then Print(b[1..k])
            else no00BinaryBackTracking(b[1...k+1],n)
```

$T(n)=2T(n-1)+ O(1)+O(1)=O(2^n)$

• ***Tìm tất cả các xâu nhị phân n bits không chứa 2 bit 1 liên tiếp***

```
Binary2 (B[1..k],n)
    For j ← 0 to 1
        do      if  b[ k-1 ] + j ≠ 2
```

```
            then  b[ k ] ← j
        if k = n print ( b[ k..n ])
        else Binary2( b[ 1, .., k+1 ] , n )
```

• ***Tìm tất cả các chỉnh hợp chập k của n số {1, 2, …, n}***

```
K_PermutationBackTracking(p[1..i], k, n)
for j←1 to n  // j ∈ A_i={1, 2, …, n} that p[i] can accept
  do if b[j] = true      // accepting possibility j
     then p[i] ← j
        b[j] ← false // record new status
        if i =k
            then Print_result(p[1..k])
            else K_PermutationBackTracking (p[1..i+1], k, n)
        b[j] ← true  // return old status
```

$T(k, n)=nT(k-1, n)+O(n) = O(n^k)$

• ***Bài toán sắp đặt n quân hậu***

```
NQeen(x[1..i], n)
for j←1 to n
  do if a[j]=true and b[i+j]=true and c[i-j]=true
     then x[i] ← j
          a[j] ← false; b[i+j] ← false; c[i-j] ← fasle
          if i=n
             then Print_result(x[1..i])
             else NQeen(x[1..i+1], n)
          a[j] ← true; b[i+j] ← true; c[i-j] ← true
```

T(n) = nT(n-1)+O(n) = O(n!)

- ***Tìm dãy con có tổng = d***

```
ALGORITHM SubsetSum(sum,b[1..k], A[i..n],d)
    if(sum = d) print(b[1...k])
    else for j←i to n do
        if(sum+a[j] <= d)
            then b[k]=a[j]
                SubsetSum(sum+a[j],b[1..k+1],A[j+1…n],d)
```