CHƯƠNG 2. CHIA ĐỂ TRỊ

2.1. Chiến lược chia để trị (Divide-and-conquer)

Chia để trị là chiến lược thiết kế giải thuật nổi tiếng nhất. Các giải thuật chia để trị thường tiến hành theo một số bước sau:

- Chia vấn đề cần giải thành một số vấn đề con cùng dạng với vấn đề đã cho, chỉ khác là cỡ của chúng nhỏ hơn .
- Mỗi vấn đề con được giải quyết độc lập, sau đó ta kết hợp nghiệm của các vấn đề con để nhận được nghiệm của vấn đề đã cho.
- Nếu vấn đề con đủ nhỏ thì ta giải trực tiếp, nếu không được giải quyết bằng cách áp dụng đệ quy thủ tục trên (tiếp tục chia nhỏ bài toán), do đó các thuật toán được thiết kế bằng phương pháp chia để trị sẽ là các thuật toán đệ quy

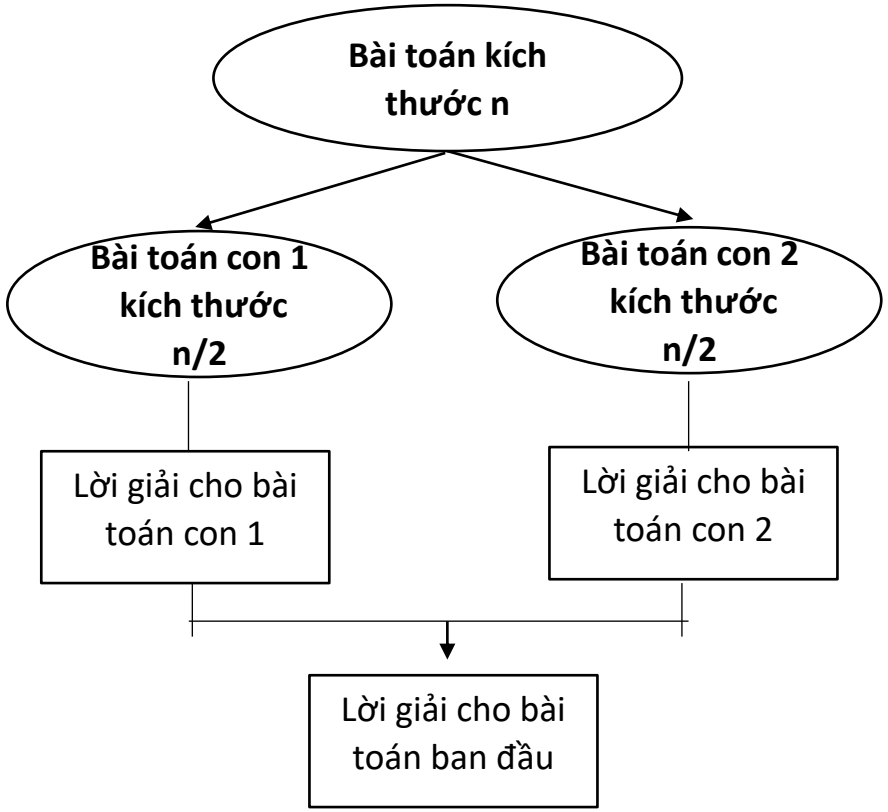
2.1. Chiến lược chia để trị (Divide-and-conquer)

Ý tưởng của phương pháp chia để trị (Divide & Conquer) là giải quyết bài toán thành 3 bước

- **Chia:** Chia bài toán thành các bài toán con có kích thước nhỏ hơn.
- **Trị:** Giải các bài toán con một cách độc lập.
- **Tổng hợp:** Tổng hợp các kết quả của các bài toán con để thu được lời giải của bài toán ban đầu.

2.2. Sơ đồ chung của kỹ thuật chia-để-trị

Trường hợp tiêu biểu của chiến lược chia để trị



2.2. Sơ đồ chung của kỹ thuật chia-để-trị)

```
DivideConquer (A,x)
// tìm nghiệm x của bài toán A.
{
if (A đủ nhỏ)
        Solve (A);
  else
{
Chia bài toán A thành các bài toán con A1, A2,…, Am;
            for (i = 1; i <= m ; i ++)

DivideConquer (Ai , xi);
Kết hợp các nghiệm xi của các bài toán con Ai (i=1, …, m) để nhận được nghiệm x của bài toán A;
}
}
```

2.3. Một số ví dụ minh họa chia để trị

Thuật toán tìm kiếm nhị phân: Cho mảng A cỡ n được sắp xếp theo thứ tự tăng dần: $A[0] \leq \ldots \leq A[n-1]$. Với x cho trước, ta cần tìm xem x có chứa trong mảng A hay không, tức là có hay không chỉ số $0 \leq i \leq n-1$ sao cho $A[i] = x$.

Kỹ thuật chia-để-trị gợi ý ta chia mảng A[0…n-1] thành 2 mảng con cỡ n/2 là A[0…k-1] và A[k+1…n-1], trong đó k là chỉ số đứng giữa mảng.

So sánh x với A[k]. Nếu $x = A[k]$ thì mảng A chứa x và $i = k$. Nếu không, do tính được sắp của mảng A, nếu $x < A[k]$ ta tìm x trong mảng A[0…k-1], còn nếu $x > A[k]$ ta tìm x trong mảng A[k+1…n-1].

2.3. Một số ví dụ minh họa chia để trị

Thuật toán tìm kiếm nhị phân:

```
bool TKNP( int k)
{
 int bottom, top, mid;
 bottom = 1;
 top = Lenght();
   while (bottom <=top)
     {mid = (bottom+top)/2;
      if ( k==A[mid] )
          return true
      else if ( k< A[mid])
               top = mid-1;
      else bottom = mid+1;    }
return false;
}
```

- Tìm phép toán cơ bản
- Tính thời gian thực hiện (độ phức tạp tính toán) giải thuật theo phép toán cơ bản bạn đã xác định?

2.3. Một số ví dụ minh họa chia để trị

Sắp xếp nhanh(Quick sort): là phương pháp xếp thứ tự theo kiểu chia để trị. Thực hiện bằng cách phân hoạch 1 tập tin thành 2 phần và sắp thứ tự mỗi phần một cách độc lập với nhau.

Ý tưởng sắp xếp: dùng một phần tử trục ở giữa (phần tử chốt – pivot) và chia hai phần còn lại tương đối bằng nhau. Việc sắp xếp thực hiện bằng cách so sánh từng phần tử của cả hai bên với phần tử chốt , nếu chúng lớn hơn hay bằng phần tử chốt thì chuyển về bên phải; nhỏ hơn thì chuyển về bên trái. Lặp lại việc phân chia và sắp xếp như vậy trên mỗi dãy con cho đến khi chúng còn một phần tử.

2.3. Một số ví dụ minh họa chia để trị

Giải thuật sắp xếp nhanh Quicksort

Phần then chốt của giải thuật **Quicksort** là thủ tục phân hoạch(partition), chủ yếu là sắp xếp lại mảng sao cho thỏa mãn các điều kiện:

- Phần tử a[i] được đưa về đúng vị trí với 1 giá trị i nào đó
- Tất cả các phần tử trong nhóm a[left]…a[i-1] nhỏ hơn a[i]
- Tất cả các phần tử trong nhóm a[i+1]…a[right] lớn hơn hoặc bằng a[i]

Giả sử: cho một mảng chưa có thứ tự:

59 53 56 52 55 58 51 57 54

Sau khi phân hoạch, đc dãy số:

52 51 53 56 55 58 59 57 54

2.3. Một số ví dụ minh họa chia để trị

Giải thuật sắp xếp nhanh Quicksort

Giả sử ta chọn phần tử tận cùng trái là phần tử sẽ được đưa về vị trí đúng của nó (phần tử chốt – pivot). Quá trình phân hoạch đòi hỏi 2 biến chạy j và k: biến j chạy từ trái sang phải cho đến khi gặp một phần tử lớn hơn hay bằng phần tử chốt và biến k chạy từ phải sang trái cho đến khi găp một phần tử nhỏ hơn phần tử chốt, khi dừng tác vụ duyệt thì hai biến chạy sẽ hoán vị cho nhau. Quá trình tiếp diễn cho đến khi 2 biến chạy giao nhau(k<j)

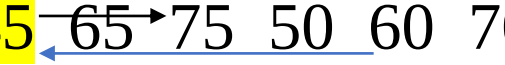
2.3. Một số ví dụ minh họa chia để trị

Giải thuật sắp xếp nhanh Quicksort

40 15 30 25 60 10 75 45 65 35 50 20 70 55

40 15 30 25 20 10 75 45 65 35 50 60 70 55

40 15 30 25 20 10 35 45 65 75 50 60 70 55



35 15 30 25 20 10 40 45 65 75 50 60 70 55

Thủ tục phân hoạch thường tốn N+1 lần so sánh khi thực hiện trên mảng N phần tử

2.3. Một số ví dụ minh họa chia để trị

Giải thuật sắp xếp nhanh Quicksort

```
void    QuickSort(Item   A[] , int a ,
int b)
 //Sắp xếp mảng A[a..b] với a ≤ b.
            {  if (a < b)
               {int   k;
               Partition(A, a, b, k);
                 if (a <= k – 1)
              QuickSort(A, a, k – 1);
                 if (k + 1 <= b)
              QuickSort(A, k + 1, b);
                 }
             }
```

2.3. Một số ví dụ minh họa chia để trị

Giải thuật sắp xếp nhanh Quicksort

```
void    QuickSort(Item   A[] , int a ,
int b)
 //Sắp xếp mảng A[a..b] với a ≤ b.
          {  if (a < b)
            {int   k;
            Partition(A, a, b, k);
              if (a <= k – 1)
           QuickSort(A, a, k – 1);
              if (k + 1 <= b)
           QuickSort(A, k + 1, b);
              }
           }
```

```
void  Partition( Item   A[] , int a , int b , int & k)
   {   keyType   pivot = A[a].key;
       int   left = a + 1;
       int   right =   b;
       do {
            while (( left <= right ) & (A[left].key <= pivot ))
                  left ++;
            while (( left <= right ) & (A[right].key > pivot ))
                  right --;
            if (left < right)
              {swap(A[left], A[right]);
                    left ++;
                    right --;
              }
            }
       while (left <= right);
       swap (A[a], A[right]) ;
       k = right ;
   }
```

2.3. Một số ví dụ minh họa chia để trị

Giải thuật sắp xếp nhanh Quicksort

Thực hiện sắp xếp nhanh trên dãy:

40 15 30 25 60 10 75 45 65 35 50 20 70 55

```
void  Partition( Item   A[] , int a , int b , int & k)
  {   keyType   pivot = A[a].key;
      int   left = a + 1;
      int   right =   b;
      do {
           while (( left <= right ) & (A[left].key <= pivot ))
                  left ++;
           while (( left <= right ) & (A[right].key > pivot ))
                  right --;
           if (left < right)
              {swap(A[left], A[right]);
                     left ++;
                     right --;
              }
           }
       while (left <= right);
       swap (A[a], A[right]) ;
       k = right ;
  }
```

2.3. Một số ví dụ minh họa chia để trị
Giải thuật Sắp xếp nhanh

Độ phức tạp của thuật toán:

- Phương pháp này khá hiệu quả, đặc biệt với dữ liệu lớn

$(n^2 + 3n)/2 \rightarrow O(n^2)$

$2C_{n/2} + n+1$

$O(n) = 2NlnN$

- Độ phức tạp của thuật toán với trường hợp trung bình và xấu nhất là O(nlogn) với n là tập các phần tử của dãy

2.3. Một số ví dụ minh họa chia để trị

Thuật toán sắp xếp hòa nhập (Merge sort)

Thuật toán sắp xếp hoà nhập (MergeSort) là một thuật toán được thiết kế bằng kỹ thuật chia - để - trị. Giả sử ta cần sắp xếp mảng A[a..b], trong đó a, b là các số nguyên không âm, a b, a là chỉ số đầu và b là chỉ số cuối của mảng. Ta chia mảng thành hai mảng con bởi chỉ số c nằm giữa a và b (c = (a + b) / 2). Các mảng con A[a..c] và A[c+1…b] được sắp xếp bằng cách gọi đệ quy thủ tục sắp xếp hoà nhập. Sau đó ta hoà nhập hai mảng con A[a…c] và A[c+1…b] đã được sắp thành mảng A[a…b] được sắp. Giả sử Merge(A,a,c,b) là hàm kết hợp hai mảng con đã được sắp A[a..c] và A[c+ 1..b] thành mảng A[a..b] được sắp.

2.3. Một số ví dụ minh họa chia để trị

Thuật toán sắp xếp hòa nhập (Merge sort)

Ý tưởng thuật toán: trước hết chia dãy cần sắp xếp thành hai phần. Sau đó lặp lại việc chia mỗi bên cho đến khi các phần chia ra chỉ còn một phần tử. Lúc đó, chúng ta sẽ có từng cặp phần tử (nếu chia chẵn) hoặc chỉ có một phần tử (nếu chia bị lẻ). Trường hợp chỉ có một phần tử thì coi như phần tử đó đã được sắp xếp. Tiếp theo ta sắp xếp các cặp phần tử từ tăng (hay giảm) bằng cách hoán vị chúng và trộn (merge) chúng lại với các phần tử được chia trước đó cho đến khi kết hợp thành một dãy theo thứ tự tăng (hay giảm).

Độ phức tạp giải thuật: Phương pháp này có độ phức tạp giải thuật trong trường hợp xấu nhất là O(nlogn). Do đó nó cũng được sử dụng phổ biến trong thực tế.

2.3. Một số ví dụ minh họa chia để trị
Thuật toán sắp xếp hòa nhập (Merge sort)

```
Void mergeSort(int a[],int i, int j)
{ int mid;
  if (i<j)
  { mid = (i+j)/2;
mergrSort(a,i,mid);// đệ quy vế trái
mergrSort(a,mid+1,j);// đệ quy vế phải}
Merge(a,i,mid,mid+1,j);
}
}
```

```
Void merge(int a[],int i1, int j1, int i2, int j2)
{int temp[50]// tạo mảng tạm để trộn
Int I,j,k;
i=i1;
J=i2;
K=0;
While(i<=j1 && j<=j2)
{ if a[i] < a{j} temp [k++] = a[i++];
Else
temp [k++] = a[j++];
}

}
```

2.3. Một số ví dụ minh họa chia để trị

Thuật toán sắp xếp hòa nhập (Merge sort)

```
void   MergeSort( Item  A[ ], int a, int b)
{
        if (a < b)
                {
                        int c = (a + b)/2;
                        MergeSort ( A, a, c );
                        MergeSort ( A, c+1, b);
                        Merge ( A, a, c, b);
                }
}
```

```
void   Merge( Item  A[] , int  a , int  c , int b)
// a, c, b là các số nguyên không âm, a ≤ c ≤ b.
// Các mảng con A[a…c] và A[c+1…b] đã được sắp.
{
        int i = a;
        int j = c + 1;
        int k = 0;
        int n = b – a + 1;
        Item * B = new Item[n];
 (1)    while (( i < c +1 )  &&  ( j < b +1 ))
          if ( A [i].key < A[j].key)
                        B[k ++]  =  A[i ++];
          else          B[k ++]  =  A[j ++];
(2)     while  ( i < c + 1)
                B[k ++] = A[i++];
(3)     while ( j < b +1)
                B[k ++] = A[ j ++];
        i  =  a;
(4)     for ( k = 0 ; k < n ; k ++)
                A[i ++] = B [k];
        delete  [ ]   B;
}
```

2.3. Một số ví dụ minh họa chia để trị

Phân tích sắp xếp hoà nhập.

Giả sử mảng cần sắp xếp A[a…b] có độ dài n, n = b – a +1, và T(n) là thời gian chạy của hàm MergeSort (A, a, b). Khi đó thời gian thực hiện mỗi lời gọi đệ quy MergeSort (A, a, c) và MergeSort (A, c + 1, b) là T(n/2).

Chúng ta cần đánh giá thời gian chạy của hàm Merge(A, a, c, b). Xem xét hàm Merge ta thấy rằng, các lệnh lặp (1), (2), (3) cần thực hiện tất cả là n lần lặp, mỗi lần lặp chỉ cần thực hiện một số cố định các phép toán. Do đó tổng thời gian của ba lệnh lặp (1), (2), (3) là O(n). Lệnh lặp (4) cần thời gian O(n). Khi thực hiện hàm MergeSort(A, a, b) với a = b, chỉ một phép so sánh phải thực hiện, do đó T(1) = O(1). Từ hàm đệ quy MergeSort và các đánh giá trên, ta có quan hệ đệ quy sau

$$T(1) = O(1)$$

$$T(n) = 2T(n/2) + O(n) \text{ với } n>1$$

Vậy T(n) = O (n log n).

2.3. Một số ví dụ minh họa chia để trị

Thuật toán tìm Max, min

Cho mảng A cỡ n, ta cần tìm giá trị lớn nhất và nhỏ nhất của mảng A. Có thể giải bằng các cách khác nhau.

Cách 2: Áp dụng kỹ thuật chia để trị để giải bài toán tìm Max, min.

A[0..n-1] chia thành các mảng con A[0..k] và A[k+1..n-1] với 0<k<n-1

35 | min=15 Max= 30 | Max=25 min= 20| min=10 Max= 40 | 45 65 75 | 50 60| 70 55

Max = 35,min=15| Max=40 min =10

2.3. Một số ví dụ minh họa chia để trị
Thuật toán tìm Max, min

```
MaxMin (i, j, max, min)
//Biến max, min lưu lại giá trị lớn nhất, nhỏ nhất trong mảng A[i..j]
      {
            if (i = = j)
                  max = min = A[i];
            else if (i = = j-1)
                  if (A[i] < A[j])
                        {     max = A[j]; min = A[i];}
                  else  {     max = A[i]; min = A[j];}
            else {
                  mid = (i+j) / 2;
                  MaxMin (i, mid, max1, min1);
                  MaxMin (mid + 1, j, max2, min2);
                  if (max 1< max2)
                              max = max2;
                  else        max = max1;
                  if (min1 < min2)
                              min = min1;
                  else        min = min2;
                  }

      }
```

2.3. Một số ví dụ minh họa chia để trị

Thuật toán tìm Max, min

Gọi T(n) là số phép so sánh cần thực hiện. T(n) được xác định bởi quan hệ đệ quy sau:

$T(1) = 0$

$T(2) = 1$

$T(n) = 2T(n/2) + 2$ với $n > 2$

Áp dụng phương pháp thế lặp, ta tính được T(n) như sau:

$T(n) = 2\,T(n/2) + 2$

$= 2^2T(n/2^2) + 2^2 + 2$

$= 2^3T(n/2^3) + 2^3 + 2^2 + 2$

……

$= 2^kT(n/2^k) + 2^k + 2^{k-1} + \ldots + 2$

Với k là số nguyên dương sao cho $2^k \leq n < 2^{k+1}$, ta có

$T(n) = 2^kT(1) + 2^{k+1} - 2 = 2^{k+1} - 2 \leq 2(n-1)$

Như vậy, T(n) = O(n).

2.4. Phương pháp giảm để trị (decrease-and-conquer)

Đây là kỹ thuật dùng mối liên hệ giữa lời giải của một bài toán và lời giải cho một thể hiện nhỏ hơn của cùng một bài toán. Khi mối quan hệ đó đã được xác lập, ta có thể vận dụng nó theo kiểu từ trên xuống hoặc từ dưới lên. Cách thứ nhất dẫn tới một giải thuật đệ quy và cách thứ 2 dung giải thuật lặp.

Có 3 biến thể của giải thuật này:

- Giảm bớt một hằng số
- Giảm bớt một hệ số
- Giảm kích thước của biến

3.1. Phương pháp giảm để trị (decrease-and-conquer)

Ví dụ 1: Xét việc tính lũy thừa a^n với $a \neq 0$ và n là một số nguyên không âm. Mối liên hệ giữa một thể hiện kích thước n và một thể hiện kích thước n-1 được thể hiện bởi công thức $a^n = a^{n-1} \times a$. Do đó, hàm $f(n) = a^n$ có thể được tính bằng cách " từ trên xuống" bởi 1 định nghĩa đệ quy như sau:

$f(n) = f(n-1).a$ với $n>0$

Với cách giảm kích thước của biến, cùng một khuôn mẫu giảm kich thước sẽ diễn ra trong một lượt lặp của giải thuật.

3.2. Phương pháp giảm để trị (decrease-and-conquer)

Ví dụ 2: giải thuật tìm USCLN của 2 số nguyên theo công thức USCLN(a,b) = USCLN(b, a mod b) là một ví dụ về giảm kích thước của biến.

```
while b<>0 do
{r=a mod b;
   a=b;
   b=r;
}
return a
```

2.4. Phương pháp giảm để trị (decrease-and-conquer)

Ví dụ 3: Giải thuật sắp xếp chèn

Ý tưởng: Sắp xếp thứ tự mảng a[0..n-1]. Theo thuật toán giảm để trị, ta giả sử có bài toán con đã sắp thứ tự mảng a[0..n-2] đã được thực hiện. Việc cần làm là chèn phần tử a[n-1] đúng vị trí vào mảng con a[0..n-2] đã có thứ tự.

Có 2 cách thực hiện

+ Duyệt mảng con đã có thứ tự từ trái sang phải cho đến khi tìm thấy phần tử đầu tiên lớn hơn hay bằng với phần tử a[n-1] và chèn phần tử a[n-1] vào bên trái phần tử này.

+ Duyệt mảng con đã có thứ tự từ phải sang trái cho đến khi tìm thấy phần tử đầu tiên nhỏ hơn hay bằng với phần tử a[n-1] và chèn phần tử a[n-1] vào bên phải phần tử này.

2.4. Phương pháp giảm để trị (decrease-and-conquer)

Chọn cách thứ 2:

a[0] ≤…a[j] ≤a[j+1] ≤… a[i-1]| a[i] …a[n-1] .

Ví dụ sắp xếp bằng phương pháp chèn được mô tả như sau:

390	**205**	**182**	**45**	**45**
205	**390**	**205**	**182**	**182**
182	182	**390**	**205**	**205**
45	45	45	**390**	**235**
235	235	235	235	**390**

2.4. Phương pháp giảm để trị (decrease-and-conquer)

Chọn cách thứ 2:

a[0] ≤…a[j] ≤a[j+1] ≤… a[i-1]| a[i] …a[n-1] .

Ví dụ sắp xếp bằng phương pháp chèn được mô tả như sau:

45 182 205 390 235 : 5 phần tử, [0..4]

Giả sử các số từ [0..3] đã sắp xếp đúng vị trí (sx tăng dần)

→ cần sắp xếp ptu a[4] = 235 vào đúng vị trí của nó.

309 309 235

309 309 45

205 309

K=235

A[j+1] = a[j] =309

3.1. Phương pháp giảm để trị (decrease-and-conquer)

Độ phức tạp của giải thuật sx bằng phương pháp chèn

1. TH xấu nhất: $O(n^2)$
2. TH trung bình: $O(n^2/4)$

Ngoài ra còn có các bài toán duyệt đồ thị

3.2. Phương pháp quy hoạch động (Dynamic Programming)

Quy hoạch động giải các bài toán tối ưu giá bằng cách kết hợp các lời giải của các bài toán con của bài toán đang xét

Kỹ thuật quy hoạch động giống kỹ thuật chia-để-trị ở chỗ cả hai đều giải quyết vấn đề bằng cách chia vấn đề thành các vấn đề con. Nhưng chia-để-trị là kỹ thuật top-down, nó tính nghiệm của các vấn đề con từ lớn tới nhỏ, nghiệm của các vấn đề con được tính độc lập bằng đệ quy. Đối lập, quy hoạch động là kỹ thuật bottom-up, tính nghiệm của các bài toán từ nhỏ đến lớn và ghi lại các kết quả đã tính được. Khi tính nghiệm của bài toán lớn thông qua nghiệm của các bài toán con, ta chỉ việc sử dụng các kết quả đã được ghi lại. Điều đó giúp ta tránh được phải tính nhiều lần nghiệm của cùng một bài toán con.

3.2. Phương pháp quy hoạch động

Để giải một bài toán bằng quy hoạch động, chúng ta cần thực hiện các bước sau:

- Đưa ra cách tính nghiệm của các bài toán con đơn giản nhất.
- Tìm ra các công thức (hoặc các quy tắc) xây dựng nghiệm của bài toán thông qua nghiệm của các bài toán con –Công thức truy hồi
- Thiết kế bảng để lưu nghiệm của các bài toán con.
- Tính nghiệm của các bài toán con từ nhỏ đến lớn và lưu vào bảng.
- Xây dựng nghiệm của bài toán từ bảng.

3.2. Phương pháp quy hoạch động

Ví dụ 1: Dãy Fibonaci là dãy số nguyên dương được định nghĩa như sau:

i<3: $F_i = 1$ ($F_1 = F_2 = 1$)

Với i>=3 : $F_i = F_{i-1} + F_{i-2.}$Tính F_6

Thuật toán lặp tính dãy số Fibonaci

Trong giải thuật lặp Fact ta đã tính tuần tự F(1), F(2),…, đến F(n). Và bởi vì để tính F(k) ta chỉ cần biết F(k-1) và F(k-2) nên ta chỉ cần lưu lại F(k-1) và F(k-2).

3.2. Phương pháp quy hoạch động

Giải thuật tính dãy fibonaci theo phương pháp đệ quy

```
int      Fibo(int n)
{
        if ((n = = 1) // (n = = 2))
                return   1;
        else
                return   Fibo (n-1) + Fibo(n-2);
}
```

```
int      Fibo1(int n)
{
        if ((n= = 1)//(n= = 2)
                return 1;
        else {
                int   previous = 1;
                int   current   = 1;
        for (int k = 3 ; k <=  n ; k ++)
                {
                        current + = previous;
                        previous = current – previous;
                }
        return   current;
        }
}
```

3.2. Phương pháp quy hoạch động

Ví dụ 2: Bài toán cái túi

Giả sử ta có chiếc túi có thể chứa được một khối lượng w, chúng ta có n loại đồ vật được đánh số i,…, n. Mỗi đồ vật loại i (i = 1,…, n) có khối lượng a_i và có giá trị c_i. Chúng ta muốn sắp xếp các đồ vật vào túi để nhận được túi có gía trị lớn nhất có thể được. Giả sử mỗi loại đồ vật có đủ nhiều đề xếp vào túi.

- Bài toán cái túi được mô tả chính xác như sau. Cho trước các số nguyên dương w, a_i, và c_i (i = 1,…,n). Chúng ta cần tìm các số nguyên không âm x_i (i = 1,…, n) sao cho

$$\sum_{n-1}^{n} x_i a_i \leq w \text{ và}$$

$$\sum_{n-1}^{n} x_i c_i \text{ đạt giá trị lớn nhất.}$$

3.2. Phương pháp quy hoạch động

Bài toán cái túi(tiếp)

Xét trường hợp đơn giản nhất: chỉ có một loại đồ vật (n = 1). Trong trường hợp này ta tìm được ngay lời giải: xếp đồ vật vào ba lô cho tới khi nào không xếp được nữa thì thôi, tức là ta tìm được ngay nghiệm $x_i = w/a_i$.

Bây giờ ta tìm cách tính nghiệm của bài toán " xếp n loại đồ vật vào túi khối lượng w" thông qua nghiệm của bài toán con " xếp k loại đồ vật ($1 \le k \le n$) vào túi khối lượng v ($1 \le v \le w$).

Ta gọi tắt là bài toán con (k,w), gọi cost(k,v) là giá trị lớn nhất của túi khối lượng v ($1 \le v \le w$) và chỉ chứa các đồ vật 1, 2, …, k. Ta tìm công thức cost(k,v). Với k = 1 và ($1 \le v \le w$), ta có: $x_i v/a$ và cost (1,v) = $x_i c_i$ (1)

3.2. Phương pháp quy hoạch động

Bài toán cái túi(tiếp)

Giả sử ta đã tính được cost (s,u) với $1 \leq s < k$ và $1 \leq u \leq v$, ta cần tính cost (k,v) theo các cost (s,u) đã biết đó. Gọi $y_k = v / a_k$, ta có

$$cost(k,v) = \max[cost(k-1,u) + x_k c_k] \quad (2)$$

Trong đó, max được lấy với tất cả $x_k = 0, 1, \ldots, y_k$ và $u = v - x_k a_k$ (tức là được lấy với tất cả các khả năng xếp đồ vật thứ k). Như vậy, tính cost (k,v) được quy về tính cost (k-1,u) với $u \leq v$. Giá trị của x_k trong (2) mà cost (k-1,u) + $x_k c_k$ đạt max chính là số đồ vật loại k cần xếp. Giá trị lớn nhất của túi sẽ là cost(n, w).

Chúng ta sẽ tính nghiệm của bài toán từ cỡ nhỏ đến cỡ lớn theo các công thức (1) và (2).

3.2. Phương pháp quy hoạch động

Bài toán cái túi(tiếp)

Nghiệm của các bài toán con sẽ được lưu trong mảng 2 chiều A[0..n-1][0..w-1], cần lưu ý là nghiệm của bài toán con (k,v) được lưu giữ trong A[k-1][v-1], vì các chỉ số của mảng được đánh số từ 0 Mỗi thành phần A[k-1][v-1] sẽ chứa cost(k,v) và số đồ vật loại k cần xếp. Từ các công thức (1) và (2) ta có thể tính được các thành phần của mảng A lần lượt theo dòng 0, 1,…n-1.

Từ bảng A đã làm đầy, làm thế nào xác định được nghiệm của bài toán, tức là xác định được số đồ vật loại i (i = 1,2,…,n) cần xếp vào ba lô? Ô A[n-1][w-1] chứa giá trị lớn nhất của ba lô cost (n,w) và số đồ vật loại n cần xếp x_n. Tính v = $w – x_n a_n$. Tìm đến ô A[n-2][v-1] ta biết được cost(n-1,v) và số đồ vật loại n-1 cần xếp x_{n-1}. Tiếp tục quá trình trên, ta tìm được x_{n-2},..,x_2 và cuối cùng là x_1.

https://www.youtube.com/watch?v=Oyc80BkADlU

https://www.youtube.com/watch?v=75pne6MTALk

3.2. Phương pháp quy hoạch động

Bài toán cái túi(tiếp)

Gọi F[i,j] là giá trị lớn nhất có thể có bằng cách chọn các món hàng i{1,2,…,i-1,i} với giới hạn trọng lượng j. Thì giá trị lớn nhất khi được chọn trong số n gói với giới hạn trọng lượng M chính lá F[n,M]. Công thức truy hồi tính F[i,j]

Với giới hạn trọng lượng j, việc chọn tối ưu trong số các gói {1,2,…,i-1,i} để có giá trị lớn nhất có 2 khả năng:

- Nếu không chọn gói thứ i thì F[i,j] là giá trị lớn nhất có thể bằng cách chọn trong số các gói {1,2,…,i-1}với giới hạn trọng lượng là j. Tức là:

F[i,j] = F[i-1,j]

- Nếu chọn gói thứ i thì F[i,j] bằng giá trị gói thứ i là a_i cộng với giá trị lớn nhất có thể có được bằng cách chọn trong số các gọi {1,2,…,i-1,i}với giới hạn trọng lượng là j-a[i]. Tức là:

F[i,j] = V[i] + F[i-1,j-a[i]]

3.2. Phương pháp quy hoạch động

Cơ sở quy hoạch động:

Ta thấy F[0,j] = giá trị lớn nhất có thể có bằng cách chọn trong số 0 goi và j = 0

Tính bảng phương án:

Bảng phương án gồm n+1 dòng, w +1 cột, trước tiên được điền cơ sở quuy hoạch động:

Dòng 0 toàn số 0. Sử dụng công thức truy hồi, dùng dòng 0 tính dòng 1, dung dòng 1 tính dòng 2… đến khi hết dòng n

3.2. Phương pháp quy hoạch động

Bài tập về nhà:

1. Bài toán cái túi
2. Bài toán tìm chuỗi con chung dài nhất

Yêu cầu: Hiểu đúng bài toán, lập bảng tính kết quá; tìm được bài toán đơn giản nhất.Bằng phương pháp quy hoạch động.

*** Xây dựng đc giả mã hoặc giải thuật (chắc chắn hiểu): TH này thì đc lấy điểm B2.

Các khái niệm

- Bài toán quy hoạch động là bài toán được giải theo phương pháp quy hoạch động
- Công thức phối hợp nghiệm của bài toán con để có nghiệm của bải toán lớn gọi là **công thức truy hồi** của quy hoạch động.
- Tập bài toán nhỏ nhất có ngay lời giải để từ đó giải quyết các bài toán lớn hơn gọi là Cơ sở quy hoạch động
- Không giai lưu trữ các lời giải bài toán con để tìm cách phối hợp chúng gọi **là bảng phương án các quy hoạch động**

Các bước cài đặt một chương trình quy hoạch động

- Giải tất cả các bài toán cơ sở (thường rất dễ), lưu các lời giải vào bảng phương án
- Dùng công thức truy hồi phối hợp với những lời giải của những bài toán nhỏ đã lưu trong bảng phương án để tìm lời giải của những bài toán lớn hơn và lưu vào bảng phương án cho đến khi bài toán ban đầu tìm được lời giải
- Dựa vào bảng phương án truy tìm ra nghiệm tối ưu

3.3. THUẬT TOÁN QUAY LUI

Trong thực tế chúng ta thường gặp các câu hỏi chẳng hạn như "có bao nhiêu khả năng...?", "hãy cho biết tất cả các khả năng...?", hoặc "có tồn tại hay không một khả năng...?".

Ví dụ, có hay không một cách đặt 8 con hậu vào bàn cờ sao cho chúng không tấn công nhau. Các vấn đề như thế thông thường đòi hỏi ta phải xem xét tất cả các khả năng có thể có

Thuật toán quay lui dùng để giải các bài toán liệt kê các cấu hình, mỗi cấu hình được xây dựng bằng cách xây dựng từng phần tử, mỗi phần tử được chọn bằng cách thử tất cả các khả năng.

3.3.1. Tìm kiếm vét cạn

Tìm **kiếm vét cạn** (exhaustive search) là xem xét tất cả các ứng cử viên nhằm phát hiện ra đối tượng mong muốn.

Ý tưởng của các thuật toán này là sinh-kiểm, tức là sinh ra tất cả các khả năng có thể có và kiểm tra mỗi khả năng xem nó có thoả mãn các điều kiện của bài toán không. Trong nhiều vấn đề, tất cả các khả năng mà ta cần xem xét có thể quy về các đối tượng tổ hợp (các tập con của một tập), hoặc các hoán vị của n đối tượng, hoặc các tổ hợp k đối tượng từ n đối tượng. Trong các trường hợp như thế, ta cần phải sinh ra, chẳng hạn, tất cả các hoán vị, rồi kiểm tra xem mỗi hoán vị có là nghiệm của bài toán không.

Ví dụ 1(Bài toán 8 con hậu)

- Vì các con hậu phải nằm trên các hàng khác nhau, ta có thể đánh số các con hậu từ 1 đến 8, con hậu i là con hậu đứng ở hàng thứ i (i=1,...,8). Gọi x_i là cột mà con hậu thứ i đứng. Vì các con hậu phải đứng ở các cột khác nhau, nên (x_1, x_2, ...,x_8) là một hoán vị của 8 số 1, 2,..., 8. Như vậy tất cả các ứng cử viên cho nghiệm của bài toán 8 con hậu là tất cả các hoán vị của 8 số 1, 2,..., 8. Đến đây ta có thể đưa ra thuật toán như sau: sinh ra tất cả các hoán vị của (x_1, x_2, ...,x_8), với mỗi hoán vị ta kiểm tra xem hai ô bất kì (i,x_i) và (j,x_j) có cùng đường chéo hay không.
- Đối với bài toán tổng quát: đặt n con hậu vào bàn cờ nxn, số các hoán vị cần xem xét là n!, và do dó thuật toán đặt **n con hậu bằng tìm kiếm vét cạn đòi hỏi thời gian O(n!).**

3.3.2. Quay lui

- Ưu điểm của quay lui so với tìm kiếm vét cạn là ở chỗ có thể cho phép ta hạn chế các khả năng cần xem xét.
- Trong nhiều vấn đề, việc tìm nghiệm của vấn đề được quy về tìm một dãy các trạng thái (a_1, a_2,…, a_k,…), trong đó mỗi a_i (i = 1,2,…) là một trạng thái được chọn ra từ một tập hữu hạn A_i các trạng thái, thoả mãn các điều kiện nào đó. Tìm kiếm vét cạn đòi hỏi ta phải xem xét tất cả các dãy trạng thái đó để tìm ra dãy trạng thái thoả mãn các yêu cầu của bài toán.

3.3.2. Quay lui

- Gọi dãy các trạng thái (a_1, a_2,…, a_n) thoả mãn các yêu cầu của bài toán là vectơ nghiệm.

Ý tưởng của kỹ thuật quay lui là ta xây dựng vectơ nghiệm xuất phát từ vectơ rỗng, mỗi bước ta bổ sung thêm một thành phần của vectơ nghiệm, lần lượt a_1,a_2,…

- Đầu tiên, tập S_1 các ứng cử viên có thể là thành phần đầu tiên của vectơ nghiệm chính là A_1.
- Chọn $a_1 \in S_1$, ta có vectơ (a_1). Giả sử sau bước thứ i-1, ta đã tìm được vectơ (a_1,a_2,…,a_{i-1}). Ta sẽ gọi các vectơ như thế là nghiệm một phần (nó thoả mãn các đòi hỏi của bài toán, những chưa "đầy đủ"). Bây giờ ta mở rộng nghiệm một phần (a_1,a_2,…,a_{i-1}) bằng cách bổ sung thêm thành phần thứ i. Muốn vậy, ta cần xác định tập S_i các ứng cử viên cho thành phần thứ i của vectơ nghiệm.

3.3.2. Quay lui

Cần lưu ý rằng, tập S_i được xác định theo các yêu cầu của bài toán và các thành phần $a_1,a_2,\ldots,a_{i-1}$ đã chọn trước, và do đó S_i là tập con của tập A_i các trạng thái.

Có hai khả năng:

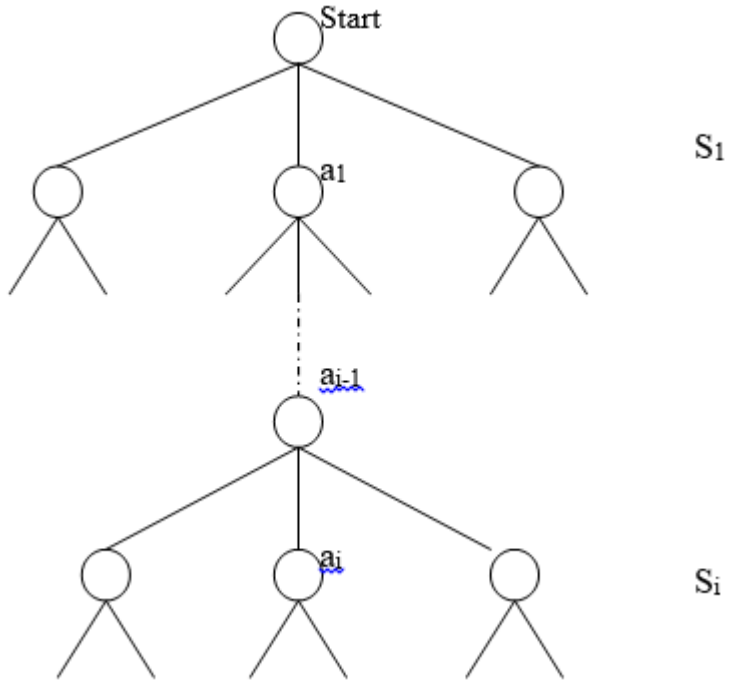
- Nếu S_i không rỗng, ta chọn $a_i \in S_i$ và thu được nghiệm một phần $(a_1,a_2,\ldots,a_{i-1},a_i)$, đồng thời loại a_i đã chọn khỏi S_i. Sau đó ta lại tiếp tục mở rộng nghiệm một phần $(a_1,a_2,\ldots,a_i)$ bằng cách áp dụng đệ quy thủ tục mở rộng nghiệm.
- Nếu S_i rỗng, điều này có nghĩa là ta không thể mở rộng nghiệm một phần $(a_1,a_2,\ldots,a_{i-2},a_{i-1})$, thì ta quay lại chọn phần tử mới a'_{i-1} trong S_{j-1} làm thành phần thứ i-1 của vectơ nghiệm. Nếu thành công (khi S_{i-1} không rỗng) ta nhận được vectơ $(a_1,a_2,\ldots,a_{i-2},a'_{i-1})$ rồi tiếp tục mở rộng nghiệm một phần này. Nếu không chọn được a'_{i-1} thì ta quay lui tiếp để chọn a'_{i-2}… Khi quay lui để chọn a'_1 mà S_1 đã trở thành rỗng thì thuật toán dừng.

3.3.2. Quay lui

•Trong quá trình mở rộng nghiệm một phần, ta cần kiểm tra xem nó có là nghiệm không. Nếu là nghiệm, ta ghi lại hoặc in ra nghiệm này. Kỹ thuật quay lui cho phép ta tìm ra tất cả các nghiệm của bài toán.

•Kỹ thuật quay lui thực chất là kỹ thuật đi qua cây tìm kiếm theo độ sâu, được xây dựng như sau

- Các đỉnh con của gốc là các trạng thái của S_1
- Giả sử a_{i-1} là một đỉnh ở mức thứ i-1 của cây. Khi đó các đỉnh con của a_{i-1} sẽ là các trạng thái thuộc tập ứng cử viên S_i. Cây tìm kiếm được thể hiện trong hình.


Start
a1
S1
ai-1
ai
Si

Lược đồ thuật toán quay lui đệ quy. Giả sử vector là nghiệm một phần $(a_1,a_2,\ldots,a_{i-1})$. Hàm đệ quy chọn thành phần thứ i của vector nghiệm là như sau:

```
Backtrack(vector , i)
// Chọn thành phần thứ i của vector.
{
        if (vector là nghiệm)
                        viết ra nghiệm;
            Tính Si;
        for (mỗi ai∈Si)
                        Backtrack(vector + (ai) , i+1);
}
```

Bài tập về nhà ngày 9/3

- 1. Làm lại phần sắp xếp nhanh theo giải thuật đã cho đối với 2 nửa dãy số như yêu cầu trên lớp
- 2. Thực hiện sắp xếp trộn (Merge sort) với dãy số
- 40 15 30 25 60 10 75 45 65 35 50 20 70 55

Theo đúng giải thuật đã được học trên lớp.

Bài tập làm ra giấy hoặc trên file text và dùng để điểm danh bằng chatbox cho buổi học sau