CHƯƠNG 7: QUY HOẠCH ĐỘNG

Mục tiêu của chương

Nắm vững:

- Các bước để giải một bài toán quy hoạch động: bài toán ba lô 1, bài toán ba lô 2, bài toán dãy con có tổng chia hết cho k và bài toán lập lịch thuê nhân công.

- Các bước cài đặt một chương trình sử dụng quy hoạch động.

- Vận dụng lý thuyết giải các bài tập quy hoạch động.

Nội dung của chương

Nghiên cứu các bước thực hiện các thuật toán trên bài toán ba lô 1, bài toán ba lô 2, bài toán dãy con có tổng chia hết cho k và bài toán lập lịch thuê nhân công.

7.1. Lý thuyết về quy hoạch động

Kỹ thuật quy hoạch động giải quyết vấn đề bằng cách chia vấn đề thành các vấn đề con. 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.

Thuật toán được thiết kế bằng kỹ thuật quy hoạch động sẽ là thuật toán lặp. Để thuận tiện cho việc sử dụng lại nghiệm của các bài toán con, chúng ta lưu lại các nghiệm đã tính vào một bảng (thông thường là mảng 1 chiều hoặc 2 chiều).

Tóm lại, 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.
- 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.

Kỹ thuật quy hoạch động thường được áp dụng để giải quyết các bài toán tối ưu (optimization problems). Các bài toán tối ưu thường là có một số lớn nghiệm, mỗi nghiệm được gắn với một giá, và mục tiêu của chúng ta là tìm ra nghiệm có giá nhỏ nhất: nghiệm tối ưu (optimization solution). Chẳng hạn, bài toán tìm đường đi từ thành phố A đến thành phố B trong bản đồ giao thông, có nhiều đường đi từ A đến B, giá của một đường đi đó là độ dài của nó, nghiệm tối ưu là đường đi ngắn nhất từ A đến B. Nếu nghiệm tối ưu của bài toán được tạo thành từ nghiệm tối ưu của các bài toán con thì ta có thể sử dụng kỹ thuật quy hoạch động.

Phương pháp quy hoạch

Phương pháp quy hoạch động dùng để giải bài toán tối ưu có bản chất đệ quy, tức

là việc tìm phương án tối ưu cho bài toán có thể đưa về tìm phương án tối ưu của số hữu hạn các bài toán con. Đối với nhiều thuật toán đệ quy chúng ta đã tìm hiểu, nguyên lý chia để trị thường đóng vai trò chủ đạo trong việc thiết kế thuật toán. Để giải quyết một bài toán lớn, ta chia nó làm nhiều bài toán con cùng dạng với nó để giải quyết độc lập. Trong phương pháp quy hoạch động, nguyên lý này càng được thể hiện rõ : Khi không biết cần phải giải quyết những bài toán con nào, ta sẽ đi giải quyết tất cả các bài toán con và lưu trữ những lời giải hay đáp số của chúng với mục đích sử dụng lại theo một sự phối hợp nào đó để giải quyết những bài toán tổng quát hơn. Đó chính là điểm khác nhau giữa quy hoạch động và phép phân giải đệ quy và cũng là nội dung của phương pháp quy hoạch động :

Phép phân giải đệ quy bắt đầu từ bài toán lớn phân rã thành nhiều bài toán con và đi giải từng bài toán con đó. Việc giải quyết từng bài toán con lại đưa về phép phân rã tiếp thành nhiều bài toán nhỏ hơn và lại đi giải tiếp bài toán nhỏ hơn đó bất kể nó đã được giải hay chưa.

Quy hoạch động bắt đầu từ việc giải tất cả các bài toán nhỏ nhất (bài toán cơ sở) để từ đó từng bước giải quyết những bài toán lớn hơn, cho tới khi giải được bài toán lớn nhất (bài toán ban đầu).

Ta xét một ví dụ đơn giản

Dãy Fibonacci là dãy vô hạn các số nguyên dương F[1], F[2],…. được định nghĩa như sau :

$$F[1]=\begin{cases} 1 & i \le 2 \\ F[i-2]+F[i-1] & i > 2 \end{cases}$$

Hãy tính F[6]

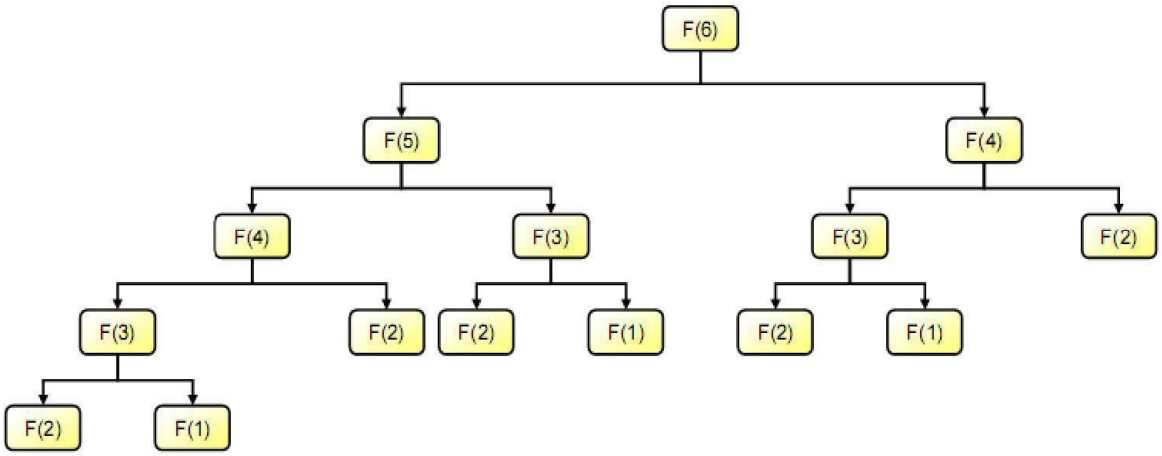
Xét hai cách cài đặt chương trình

```
int F(int i)
{
    if(i <=2) return 1 ;
    return F(i – 2) + F(i – 1)
}
void main()
{
   cout<<F(6)<< endl;
}
```

```
int F(100); // Bang phuong an
void main
{
    F[1] = F[2] = 1;
    for(int i =3; i<=6; i++)
        F[i] = F[i-2] + F[i -1]
   cout<<F(6)<< endl;
}
```

Cách 1 có hàm đệ quy F(i) để tính số Fibonacci thứ i. Quá trình tính toán có thể vẽ như cây dưới đây. Ta nhận thấy để tính F(6) nó phải tính một lần F(5), hai lần F(4), ba

lầm F(3), năm lần F(2), ba lần F(1).



Hình 7.1: Sơ đồ cây để tính số Fibonacci

Hàm đệ quy tính số Fibonacci

Cách 2: Trước hết nó tính sẵn F[1] và F[2], từ đó tính tiếp F[3], lại tính tiếp được F[4], F[5], F[6]. Đảm bảo mỗi giá trị Fibonacci chỉ phải tính một lần.

Trước khi áp dụng phương pháp quy hoạch động ta phải xem xét phương pháp đó có thỏa mãn những yêu cầu dưới đây hay không:

Bài toán lớn phải phân rã được thành nhiều bài toán con mà có sự phối hợp lời giải của các bài toán con đó cho ta lời giải bài toán lớn.

Vì quy hoạch động là đi giải tất cả bài toán con, nên nếu không đủ không gian vật lý lưu trữ lời giải (bộ nhớ, đĩa….) để phối hợp chúng thì phương pháp quy hoạch động cũng không thể thực hiện được.

Quá trình từ bài toán cơ sở tìm ra lời giải bài toán ban đầu phải qua hữu hạn bước.

Các khái niệm

Bài toán giải theo phương pháp quy hoạch động gọi là bài toán quy hoạch động

Công thức phối hợp nhiều nghiệm của các bài toán con để có nghiệm của bài toán lớn gọi là công thức truy hồi (hay phương trình truy toán) của quy hoạch động

Tập tất cả các 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 gian lưu trữ lời giải các bài toán con để tìm cách phối hợp chúng gọi là bảng phương án của quy hoạch động.

Các bước cài đặt một chương trình sử dụng quy hoạch động.

- Giải tất cả các bài toán cơ sở (thông thường rất dễ), lưu các lời giải vào bảng phương án.
- Dùng công thức truy hồi phối hợp 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 chúng vào bảng phương án. Cho tới khi bài toán ban đầu tìm được lời giải.
- Dựa vào bảng phương án, truy vết tìm ra nghiệm tối ưu.

7.2. Bài toán ba lô 1 (knapsack)

Cho n gói hàng. Gói hàng thứ i có khối lượng là A[i] và giá trị C[i]. Cần chọn những gói hàng nào để bỏ vào một ba lô sao tổng giá trị của các gói hàng đã chọn là lớn nhất nhưng tổng khối lượng của chúng không vượt quá khối lượng M cho trước. Mỗi gói chỉ chọn 1 hoặc không chọn.

Ví dụ: n = 5; M = 13

i	1	2	3	4	5
A[i]	3	4	5	2	1
C[i]	4	5	6	3	1

Tổng giá trị của các gói hàng bỏ vào ba lô: 16

Các gói được chọn: 1(3, 4), 2(4, 5), 3(5, 6), 5(1, 1)

Tham số thể hiện kích thước bài toán

- Kết quả bài toán là tổng giá trị lớn nhất của các món hàng được chọn trong n món sao cho tổng khối lượng không lớn hơn M cho trước, ký hiệu là F(n)
- Tham số thể hiện kích thước bài toán là số món hàng n
- Giá trị của F(n) có thể được tính từ giá trị của F(n-1) cộng thêm hoặc không cộng thêm giá trị của món hàng thứ n nhưng tổng khối lượng không lớn hơn M.
- Nếu chọn thêm món hàng thứ n thì tổng khối lượng được chọn trong (n-1) món hàng không lớn hơn (M-A[n])

Suy ra bài toán có 2 tham số: số món hàng và khối lượng giới hạn

Lập công thức đệ quy

Gọi F(i, v) là tổng giá trị lớn nhất của các gói hàng được chọn trong i gói hàng sao cho tổng khối lượng không lớn hơn v.

- Trường hợp A[i] > v:

  F(i, v) = F(i -1, v)

- Trường hợp A[i] <= v:

  - Nếu gói hàng thứ i không được chọn thì:

    F(i, v) = F(i -1, v)

  - Nếu gói hàng thứ i được chọn thì:

    F(i, v) = F(i -1, v – A[i]) + C[i]

    → F(i, v) = Max{ F(i -1, v); F(i -1, v – A[i]) + C[i]) }

Bài toán nhỏ nhất ứng với i = 0 ta có: F(0, v) = 0

Xây dựng bảng phương án:

- Cấu trúc bảng phương án:

  Dùng mảng F[0..n][0..M] chứa giá trị của các F(i, v)

- Cách tính giá trị trên bảng phương án:

  - Điền số 0 cho các ô trên dòng 0

  - Sử dụng công thức đệ quy và giá trị trên dòng (i -1) để tính dòng i

    • Trường hợp A[i] > v:

      F(i, v) = F(i -1, v)

    • Trường hợp A[i] <= v:

      F(i, v) = Max{ F(i -1, v); F(i -1, v – A[i]) + C[i]) }

Ví dụ bảng phương án:

• Trường hợp A[i] > v: F(i, v) = F(i -1, v)

• Trường hợp A[i] <= v: F(i, v) = Max{ F(i -1, v); F(i -1, v – A[i]) + C[i])}

C	A	v i	0	1	2	3	4	5	6	7	8	9	10	11	12	13
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	3	1	0	0	0	4	4	4	4	4	4	4	4	4	4	4
5	4	2	0	0	0	4	5	5	5	9	9	9	9	9	9	9
6	5	3	0	0	0	4	5	6	6	9	10	11	11	11	15	15
3	2	4	0	0	3	4	5	7	8	9	10	12	13	14	15	15
1	1	5	0	1	3	4	5	7	8	9	10	12	13	14	15	16

Thuật toán tạo bảng phương án

```
void TaoBangPhuongAn(F[0..n][0..M])
{
    for (v=0; v <= M; v++) F[0, v] = 0; // Điền số 0 cho dòng 0 của bảng
    for (i = 1; i <= n; i++)
        for (v=0; v <= M; v++)
        {
            F[i, v] = F[i-1, v];
            if (A[i] <= v && F[i, v] < F[i-1, v - A[i] ] + C[i])
                F[i, v] = F[ i-1, v - A[i] ] + C[i];
        }
}
```

Truy vết tìm lại các gói hàng đã chọn

Bắt đầu từ ô F[n, M] trên dòng n ta dò ngược về dòng 0 theo nguyên tắc:

- Nếu F[i, v] <> F[i–1, v]

  thì gói thứ i được chọn, ta truy tiếp ô F[i-1, v-A[i]].

- Nếu F[i, v] = F[i–1, v] thì gói thứ i không được chọn, ta truy tiếp ô F[i-1, v].

C	A	i v	0	1	2	3	4	5	6	7	8	9	10	11	12	13
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	3	1	0	0	0	4	4	4	4	4	4	4	4	4	4	4
5	4	2	0	0	0	4	5	5	5	9	9	9	9	9	9	9
6	5	3	0	0	0	4	5	6	6	9	10	11	11	11	15	15
3	2	4	0	0	3	4	5	7	8	9	10	12	13	14	15	15
1	1	5	0	1	3	4	5	7	8	9	10	12	13	14	15	16

Thuật toán truy vết tìm lại các gói hàng đã chọn

```
void TruyVet(F[0..n][0..M])
    { Bắt đầu từ ô F[n, M] trên dòng n: i = n; v = M;
      for (; i > 0; i --)
          if (F[i, v] != F[i-1, v])
          {
                <Món hàng thứ i được chọn >;
                v = v – A[i];
          }
    }
```

7.3. Bài toán ba lô 2 (knapsack)

Cho n loại hàng. Món hàng thuộc loại hàng i có khối lượng A[i] và giá trị C[i]. Số lượng các món hàng của mỗi loại không hạn chế. Cần chọn các món hàng trong từng loại để bỏ vào một ba lô sao cho tổng giá trị của các món hàng đã chọn là lớn nhất nhưng tổng khối lượng của chúng không vượt quá khối lượng M cho trước. Cho biết số lượng món hàng từng loại hàng được chọn

Ví dụ: n = 5; M = 13

i	1	2	3	4	5
A[i]	3	4	5	2	1
C[i]	4	5	6	3	1

Tổng giá trị của các món hàng bỏ vào ba lô: 19

Các món được chọn:

1 gói hàng loại 1 có khối lượng 3 và giá trị 4

5 gói hàng loại 4 có khối lượng 2 và giá trị 3

Xác định công thức đệ quy

Gọi F(i, v) là tổng giá trị lớn nhất của các món hàng được chọn sao cho tổng khối lượng <= v trong i loại hàng.

Trường hợp A[i] > v: F(i, v) = F(i-1, v)

Trường hợp A[i] <= v:

- Nếu loại hàng i không được chọn thì:

F(i, v) = F(i-1, v)

- Nếu có k món hàng loại i được chọn: (1 <= k <= v/A[i])

F(i, v) = F(i-1, v – A[i]*k) + C[i]*k

Do đó:

F(i, v) = Max{F(i-1, v – A[i]*k) + C[i]*k } k $\in$ [0, v/A[i]] Bài toán nhỏ nhất ứng với i = 0 hay v=0 ta có: F(0, v) = 0

Công thức đệ quy

Gọi F(i, v) là tổng giá trị lớn nhất của các món hàng được chọn có tổng khối lượng <= v trong i loại hàng đầu tiên

Với i = 0 : F(i, v) = 0

Với i > 0 :

- A[i] > v : F(i, v) = F(i - 1, v)

- A[i] <= v : F(i, v) = Max{ F(i-1, v – A[i]*k) + C[i]*k }

với $k \in$ [0, v/A[i]]

Xây dựng bảng phương án:

Cấu trúc bảng phương án: dùng 2 mảng

- Mảng F[0..n][0..M]: F[i, v] chứa giá trị của các F(i, v)

- Mảng S[1..n][1..M]: S[i,v] chứa số món hàng loại i được chọn

Nếu F(i, v) = F(i - 1, v): S[i, v] = 0

Ngược lại S[i, v] = k

Cách tính giá trị trên bảng phương án:

- Điền số 0 cho các ô trên dòng 0 và cột 0 của bảng F

- Sử dụng công thức đệ quy và giá trị trên dòng i -1 để tính dòng i của bảng F và bảng S

Ví dụ: Lập bảng phương án F

- Với i > 0 :

- A[i] > v : F(i, v) = F(i - 1, v)

- A[i] <= v : F(i, v) = Max{ F(i-1, v – A[i]*k) + C[i]*k }

với k $\in$ [0, v/A[i]]

Bảng F(i,v)

C[i]	A[i]	i v	0	1	2	3	4	5	6	7	8	9	10	11	12	13
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	3	1	0	0	0	4	4	4	8	8	8	12	12	12	16	16
5	4	2	0	0	0	4	4	5	8	9	9	12	13	14	16	17
6	5	3	0	0	0	4	4	5	8	9	10	12	13	14	16	17
3	2	4	0	0	0	4	4	7	8	10	11	13	14	16	17	19
1	1	5	0	0	1	4	5	7	8	10	11	13	14	16	17	19

Bảng S[i, v]

C[i]	A[i]	i v	1	2	3	4	5	6	7	8	9	10	11	12	13
4	3	1	0	0	1	1	1	2	2	2	3	3	3	4	4
5	4	2	0	0	0	0	1	0	1	1	0	1	2	0	1
6	5	3	0	0	0	0	0	0	0	1	0	0	0	0	0
3	2	4	0	0	0	0	1	0	2	1	3	2	4	3	5
1	1	5	0	1	0	1	0	0	0	0	0	0	0	0	0

Thuật toán tạo bảng phương án

```
void TaoBangPhuongAn(F[0..n][0..M], S[1..n][1..M])
{ <Điền số 0 cho dòng 0 và cột 0 của bảng F[0..n][0..M]>;
        for (i = 1; i <= n; i++)
                for (v=1; v <= M; v++)
                { F[i, v] = F[i-1, v];
                  S[i, v] = 0;
                  if (v >= A[i])
                        for(k = 1; k <= v/A[i]; k++)
                                if (F[i, v] < F[i-1, v - A[i]*k ] + C[i]*k)
                                { F[i, v] = F[i-1, v - A[i]*k ] + C[i]*k;
                                  S[i, v] = k;
                                }
                }
}
```

Truy vết tìm lại các gói hàng đã chọn

Bắt đầu từ ô S[n, M] trên dòng n ta dò ngược về dòng 1 theo nguyên tắc:

- Nếu S[i, v] <> 0 thì :

  - Loại hàng i được chọn với số lượng là S[i, v]

  - Truy tiếp ô S[i-1, v - S[i, v]*A[i]].

- Nếu S[i, v] = 0 thì :

  - Loại hàng i không được chọn,

  - Truy tiếp ô S[i-1, v].

C[i]	A[i]	i v	1	2	3	4	5	6	7	8	9	10	11	12	13
4	3	1	0	0	1	1	1	2	2	2	3	3	3	4	4
5	4	2	0	0	0	0	1	0	1	1	0	1	2	0	1
6	5	3	0	0	0	0	0	0	0	1	0	0	0	0	0
3	2	4	0	0	0	0	1	0	2	1	3	2	4	3	5
1	1	5	0	1	0	1	0	0	0	0	0	0	0	0	0

Thuật toán truy vết tìm lại các gói hàng đã chọn

```
void TruyVet(S[1..n][1..M])
{ i = n; v = M
        while (i > 0)
        {
                if (S[i, v] != 0)
                {
                        <in số lượng gói hàng i trong S[i, v] >;
                        v = v – S[i, v]*A[i];
                }
                i = i – 1;
        }
}
```

7.4. Bài toán dãy con có tổng chia hết cho k

Cho một dãy A gồm n số nguyên và một số nguyên dương k. Hãy tìm một dãy con (không nhất thiết phải liên tiếp nhau) dài nhất có tổng các số chia hết cho số k.

Ví dụ: n = 6 và k = 5

i	1	2	3	4	5	6
Dãy A[i]	11	6	7	12	20	8

Chiều dài dãy con: 4

Các phần tử được chọn là : 1 2 5 6

Có giá trị tương ứng là : : 11 6 20 8

Nhắc lại phép toán mod

Giả sử r = a mod k và z = b mod k

Ta có:

1. (a + b) mod k = (r + z) mod k
2. (-r) mod k = (- r + k) mod k
3. (r + z) mod k = v $\rightarrow$ z = (v – r) mod k = (v – r + k) mod k

i	1	2	3	4	5	6
A[i]	11	6	7	12	20	8
A[i] = A[i] % 5	1	1	2	2	0	3
	0	0	0	3	4	4

Xác định tham số

Gọi F(i) = chiều dài dãy con dài nhất trong miền [1..i] có tổng chia hết cho k.

1. Nếu dãy con dài nhất không có A[i] thì F(i) = F(i-1)

Với F(i-1) = chiều dài dãy con dài nhất trong miền [1..i-1] có tổng chia hết cho k

2. Nếu dãy con dài nhất có chứa A[i]: thì F(i) = F(i-1) + 1

Gọi r = A[i] mod k

- Nếu r = 0: thì F(i-1) = chiều dài dãy con dài nhất trong miền [1..i- 1] có tổng chia hết cho k

- Nếu r > 0 : do (r + k - r) mod k = 0 nên F(i-1) bằng chiều dài dãy con dài nhất trong miền [1..(i-1)] có tổng chia với k dư (k-r)

Tham số thể hiện kích thước của bài toán: kích thước miền và số dư của tổng chia với k

Lập công thức đệ quy

Gọi F(i, v) = chiều dài dãy con dài nhất trong miền [1..i] có tổng chia với k dư là v.

1. Nếu dãy con dài nhất không có A[i] thì F(i, v) = F(i-1, v)
2. Nếu dãy con dài nhất có chứa A[i]:

Gọi r = A[i] mod k ta có F(i, v) = F(i-1, v - r) + 1

- Nếu v – r = 0 : F(i, v) = F(i-1, 0) + 1
- Nếu v – r > 0 và F(i-1, v - r)>0: F(i, v) = F(i-1, v - r) + 1
- Nếu v – r < 0 và F(i-1, v-r+k)>0: F(i, v) = F(i-1, v-r+k) + 1

thay (v - r) mod k = (v – r + k) mod k

Bài toán nhỏ nhất ứng với i = 1:

• F(1,v)= 0 nếu r <> v

• F(1,v)= 1 nếu r = v

Công thức đệ quy

Gọi r = A[i] mod k

- Với i = 1:

  - F(1, v) = 0 nếu r <> v

  - F(1, v) = 1 nếu r = v

- Với i > 1 :

  - Nếu v = r thì : F(i, v) = Max {F(i-1, v), F(i-1, 0) +1 }

  - Nếu v > r và F(i-1, v- r) >0 thì

    F(i, v) = Max {F(i-1, v), F(i-1, v – r) +1 }

  - Nếu v < r và F(i-1, v – r + k)>0 thì

    F(i, v) = Max{F(i-1, v), F(i-1, v – r + k) + 1}

Tinh chế công thức đệ quy

Gọi r = A[i] mod k

• Với i = 1:

  - F(1, v) = 0 nếu r <> v

  - F(1, v) = 1 nếu r = v

• Với i > 1 :

  - Nếu v = r thì : F(i, v) = Max {F(i-1, v), F(i-1, 0) +1 }

  - Nếu v<>r và F(i-1, (v- r +k) mod k) > 0 thì

    F(i, v) = Max {F(i-1, v), F(i-1, (v – r + k) mod k) +1 }

Xây dựng bảng phương án:

- Cấu trúc bảng phương án:

  Dùng mảng F[1..n][0..K-1] chứa giá trị của các F(i, v)

- Cách tính giá trị trên bảng phương án:

  - Điền giá trị dòng 1: Nếu A[1] mod k = v thì F[1, v] = 1 ngược lại F[1, v] = 0

  - Sử dụng công thức đệ quy và giá trị trên dòng i -1 để tính dòng i

Ví dụ bảng phương án:

• Với i > 1 :

  - Nếu v = r thì : F(i, v) = Max {F(i-1, v), F(i-1, 0) +1 }

  - Nếu v<>r và F(i-1, (v-r+k) mod k) > 0 thì

    F(i, v) = Max {F(i-1, v), F(i-1, (v – r + k) mod k) +1}

i	r = A[i] | v	0	1	2	3	4
1	1	0	1	0	0	0
2	1	4 | 0	0 | 1	1 | 2	0 | 2	3 | 0
3	2	3 | 0	4 | 1	0 | 2	1 | 2	2 | 3
4	2	3 | 3	4 | 3	0 | 2	1 | 2	2 | 3
5	0	0 | 4	1 | 4	2 | 3	3 | 3	4 | 4
6	3	2 | 4	3 | 4	4 | 5	1 | 5	2 | 6

Thuật toán tạo bảng phương án

```
voidTaoBangPhuongAn(F[1..n][0..k-1])
{
    for (v=0; v <= k-1; v++) F[1, v] = (A[i]%k == v) ? 1 : 0;
    for (i = 2; i <= n; i++)
        for (v=0; v <= k-1; v++)
        { r = A[i] % k;
          F[i, v] = F[i-1, v];
          if (v == r && F[i, v] <= F[i-1, 0 ])
              F[i, v] = F[i-1, 0] + 1;
          else if (F[i-1, (v - r + k)%k] >0 && F[i, v] <= F[i-1, (v - r + k)%k])
        F[i, v] = F[ i -1, (v - r + k)%k] + 1;
        }
}
```

Truy vết

Bắt đầu từ ô F[n, 0] trên dòng n ta dò ngược về dòng 0 theo nguyên tắc:

• Nếu F[i–1, (v-r+k)%k] > 0 và F[i, v] > F[i–1, (v-r+k)%k]:

- A[i] được chọn
- Truy tiếp ô F[i -1, (v-r+k)%k].

• Ngược lại thì A[i] không được chọn, ta truy tiếp ô F[i -1, v].

i	r = A[i] | v	0	1	2	3	4
1	1	0	1	0	0	0
2	1	4 | 0	0 | 1	1 | 2	0 | 2	3 | 0
3	2	3 | 0	4 | 1	0 | 2	1 | 2	2 | 3
4	2	3 | 3	4 | 3	0 | 2	1 | 2	2 | 3
5	0	0 | 4	1 | 4	2 | 3	3 | 3	4 | 4
6	3	2 | 4	3 | 4	4 | 5	1 | 5	2 | 6

7.5. Lập lịch thuê nhân công

Có một dự án kéo dài trong T tháng, người quản lý cần phải lập lịch sử dụng công nhân mỗi tháng cho dự án. Biết rằng, số công nhân tối thiểu cần trong tháng thứ i là Scn[i]; tiền dịch vụ khi thuê 1 công nhân mới là DV; tiền đền bù khi sa thải một công nhân là ST; lương tháng mỗi công nhân phải trả là LT.

Cần phải thuê hay sa thải bao nhiêu công nhân mỗi tháng để tổng chi phí nhân công của dự án là nhỏ nhất.

Giả thiết	Kết luận
T=3 DV=4; ST=5; LT=6 Scn={11; 9; 10}	265 11 10 10

Xác định tham số thể hiện kích thước bài toán

- Tham số thể hiện kích thước bài toán là số tháng T
- Tổng chi phí nhân công trong T tháng được tính từ tổng chi phí nhân công của T-1 tháng cộng thêm chi phí trả nhân công của tháng thứ T.
- Chi phí trả nhân công của tháng thứ T bao gồm :

  - Tiền lương trả cho số nhân công của tháng T và

  - Tiền dịch vụ nếu số nhân công của tháng T lớn hơn số nhân công tháng T-1 hay tiền sa thải nếu số nhân công trong tháng T nhỏ hơn số nhân công của tháng T-1.
- Kích thước bài toán phụ thuộc vào 2 tham số: số tháng và số nhân công của tháng

Lập công thức đệ quy

- Scn[i] lưu số công nhân cần thuê cho tháng thứ i
- Smax là số công nhân của tháng cần nhiều người nhất
- Bài toán con nhỏ nhất ứng với i = 1 (tháng đầu tiên):

  C(1, j) = j * (DV + LT) với j = Scn[1]..Smax

- C(i, j) là chi phí tối thiểu của i tháng đầu tiên nếu tại tháng thứ i có j công nhân được thuê.

  C(i, j) = Min{ C(i-1, k) + chi phí để từ k người thành j người }

  (i=2..T; j =Scn[i]..Smax; k = Scn[i-1]..Smax)

- Kết quả bài toán là: Kq = Min{C(T, j) + chi phí sa thải j người}

  j=Scn[T]..Smax

Xây dựng bảng chứa C(i, j)

- Mảng C[1..T+1, 1..Smax]: C[i, j] ghi nhận giá trị C(i, j)

  C(1, j) = j * (DV + LT) với j = Scn[1]..Smax

  C(i, j) = Min{ C(i-1, k) + chi phí để từ k người thành j người }

  (i=1..T; j =Scn[i]..Smax; k = Scn[i-1]..Smax)

  C(T+1, j) = C(T, j) + (j * ST) j=Scn[T]..Smax

Scn	i \ j	9	10	11
11	1			99
9	2	99+45+12=156	99+50+6=155	99+55=164
10	3		155+50=205	155+55+4=214
	4		205+60=265	214+66=280

Xây dựng bảng truy vết số công nhân

Mảng Truoc[1..T, 1..Smax]: Truoc[i, j] := k là số người thuê ở tháng thứ i-1 để có C[i, j]

Scn	i \ j	9	10	11
11	1			99
9	2	99+45+12=156	99+50+6=155	99+55=164
10	3		155+50=205	155+55+4=214
	4		205+60=265	214+66=280

Scn	i \ j	9	10	11
11	1			
9	2	11	11	11
10	3		10	10

Thuật toán tạo bảng phương án C và Truoc

```
{       for (j=Scn[1]; j <= Smax; j++) C[1, j] = j * (DV + LT);
        for (i = 2; i <= T; i++)
                for (j=Scn[i]; j <= Smax; j++)
                {       C[i, j] = MAXINT;
                        for (k=Scn[i-1]; k <= Smax; k++)
                        {       X = C[i-1, k] ;
                                if (k > j) X = X + (k – j)*DV; else X = X + (j – k)*ST
                                if (X< C[i, j]) { C[i, j] = X; Truoc[i, j] = k; }
                        }
                }
        for (j=Scn[T]; j <= Smax; j++) C[T+1, j] = C(T, j) + (j * ST) ;
}
```

Thuật toán truy vết số công nhân mỗi tháng

```
{ //Tìm số nhân công của tháng thứ T
        S = C[T+1, Scn[T] ]; k = Scn[T];
        for (j=Scn[T]+1; j <= Smax; j++)
                if (S > C[T+1, j ]) { k = j; S =C[T+1, j ]; }
        <Tháng T cần k công nhân>
        for (i=T; i > 1; i--)
        {       k = Truoc[i, k];
                <Tháng i-1 cần k công nhân>
        }
}
```

Bài tập vận dụng

1. Có n đồ vật, vật thứ i có trọng lượng Ai và giá trị Bi. Hãy chọn ra một số các đồ vật, mỗi vật một cái để xếp vào 1 vali có trọng lượng tối đa W sao cho tổng giá trị của vali là lớn nhất.
2. Cho dãy A_1, A_2,..., A_N. Tìm một dãy con của dãy đó có tổng bằng S.
3. Trong siêu thị có n đồ vật ($n \leq 1000$), đồ vật thứ i có trọng lượng là $W[i] \leq 1000$ và giá trị $V[i] \leq 1000$. Một khách hàng vào siêu thị, khách hàng mang theo một cái túi có thể mang được tối đa trọng lượng M ($M \leq 1000$). Hỏi khách hàng sẽ lấy đi những đồ vật nào để được tổng giá trị lớn nhất.

   Giải quyết bài toán trong các trường hợp sau:

a) Mỗi vật chỉ được chọn một lần.

b) Mỗi vật được chọn nhiều lần (không hạn chế số lần).

4. Một công ty máy tính nhận được N hợp đồng ($N \leq 50$) lắp đặt hệ thống máy tính tại N công ty. Hợp đồng thứ i có giá trị là Ci (số nguyên) và cần Ai nhân sự để thực hiện. Do số lượng nhân sự của công ty có hạn, nên Công ty muốn ưu tiên chọn một số hợp đồng để thực hiện trước sao cho tổng giá trị của các hợp đồng đã chọn là lớn nhất nhưng tổng số nhân sự thực hiện các hợp đồng đó không vượt quá số lượng M nhân sự ($M \leq 100$) hiện có của công ty.

Hãy trình bày thuật giải để chọn lựa các hợp đồng theo yêu cầu trên, cho biết tổng giá trị của các hợp đồng đã chọn; giá trị hợp đồng và số lượng nhân sự của các hợp đồng đã chọn.

5. Cho số tự nhiên n, hãy cho biết tất cả các dãy số tự nhiên tăng, có tổng bằng n. Chẳng hạn, với n = 6, ta có các dãy sau:

1 , 2 , 3

1, 5

2, 4

6

6. Cho một tập A các loại tiền A = {a1, a2, …, an}, trong đó mỗi ai là mệnh giá của một loại tiền, ai là số nguyên dương. Vấn đề đổi tiền được xác định như sau. Cho một số nguyên dương c (số tiền cần đổi), hãy tìm số ít nhất các tờ tiền với các mệnh giá trong A sao cho tổng của chứng bằng c. Giả thiết rằng, mỗi loại tiền có đủ nhiều, và có một loại tiền có mệnh giá là 1.

PHỤ LỤC

1. Code Thuật toán Selection-Sort

```
#include <iostream.h>
#include <conio.h>
#define max 100
//Nhap day
void NhapDay(int a[],int n) {
  for(int i=0; i<n; i++) {
  cout<<"\n a["<<i<<"] =";
  cin>>a[i];
  }
}
//Xuat day
void XuatDay(int a[],int n) {
  cout<<"\n IN DAY: ";
  for(int i=0; i<n; i++)
  cout<<a[i]<<"\t";
}
//Hoan vi 2 phan tu
void Swap(int &a,int &b) {
   int t = a;
   a = b;
   b = t;
}
//Thuat toan Selection Sort
void SelectionSort(int a[],int n) {
   int min; // chi so phan tu nho nhat trong day hien hanh
   for(int i=0; i<n-1; i++) {
      min = i;
      for(int j=i+1; j<n; j++)
        if(a[min]>a[j])
           min = j; //ghi nhan vi tri phan tu nho nhat
```

```
    if(min!= i)
        Swap(a[i],a[min]); // doi chu 2 phan tu
   }
}
//Chuong trinh chinh
void main() {
  int a[max],n;
  clrscr();
  cout<<"Nhap so phan tu:";
  cin>>n;
  NhapDay(a,n);
  cout<<"\n Day vua nhap la:";
  XuatDay(a,n);
  cout<<endl;  SelectionSort (a,n);
  cout<<"\n Day vua sap xep la:";
  XuatDay(a,n);
  getch();
}
```

2. Thuật toán Bubble-Sort

```
#include <iostream.h>
#include <conio.h>
#define max 100
//Nhap day
void NhapDay(int a[],int n) {
  for(int i=0; i<n; i++) {
  cout<<"\n a["<<i<<"] =";
  cin>>a[i];
  }
}
//Xuat day
void XuatDay(int a[],int n) {
  cout<<"\n IN DAY: ";
```

```
  for(int i=0; i<n; i++)
cout<<a[i]<<"\t";
}
//Hoan vi 2 phan tu
void Swap(int &a,int &b) {
  int t = a;
  a = b;
  b = t;
}
//Sap xep cac phan tu
void BubbleSort(int a[],int n) {
  for(int i = 0; i < n-1; i++)
  for(int j = n-1; j > i; j--)
      if(a[j]<a[j-1])
        Swap(a[j-1],a[j]);
}
//Chuong trinh chinh
void main() {
 int a[max],n;
 clrscr();
 cout<<"\n Nhap so phan tu:";
 cin>>n;
 NhapDay(a,n);
 XuatDay(a,n);
 cout<<"\n Sap xep Bubble-Sort: \n";
 BubbleSort(a,n);
 cout<<"\n Day vua sap xep la:";
  XuatDay(a,n);
  getch();
}
```

3. Thuật toán Insert-Sort

```
#include <iostream.h>
#include <conio.h>
#define max 100
//Nhap day
void NhapDay(int a[],int n) {
  for(int i=0; i<n; i++) {
  cout<<"\n a["<<i<<"] =";
  cin>>a[i];
  }
}
//Xuat day
void XuatDay(int a[],int n) {
  cout<<"\n IN DAY: ";
  for(int i=0; i<n; i++)
  cout<<a[i]<<"\t";
}
//Hoan vi 2 phan tu
void Swap(int &a,int &b) {
  int t = a;
  a = b;
  b = t;
}
//Thu tuc Insertion Sort
void InsertionSort(int a[],int n) {
  for(int i=1; i<n; i++)
    for(int j=i; j>0; j--)
      if(a[j]<a[j-1])
       Swap(a[j],a[j-1]);
}
//Chuong trinh chinh
void main() {
```

```
 int a[max],n;
 clrscr();
 cout<<"Nhap so phan tu:";
 cin>>n;
 NhapDay(a,n);
 cout<<"\n Day vua nhap la:";
 XuatDay(a,n);
 cout<<endl;
 InsertionSort (a,n);
 cout<<"\n Day vua sap xep la:";
 XuatDay(a,n);
 getch();
}
```

4. Thuật toán sắp xếp nhanh - Quick Sort

```
#include <iostream.h>
#include <conio.h>
#define max 100
// Nhap mang
void NhapMang(int A[],int n) {
 for(int i=0; i<n; i++) {
  cout<<"nhap Phan tu thu A["<<i<<"] =";
  cin>>A[i];
 }
}
// In mang
void XuatMang(int A[],int n) {
 cout<<endl;
 for(int i=0; i<n; i++)
  cout<<A[i]<<"\t";
}
// Đoi cho 2 so
void Swap(int &a,int &b) {
```

```
 int temp = a;
 a = b;
 b = temp;
}
// Thuat toan Quick-sort
void QuickSort(int A[], int Left, int Right) {  int
 i = Left, j = Right;
 int pivot = A[(Left + Right) / 2];
 /* partition */
 while (i <= j) {
  while (A[i] < pivot)
   i++;
   while (A[j] > pivot)
    j--;
    if (i <= j) {
    Swap(A[i],A[j]);
    i++;
   j--;
  }
 }
 /* recursion */
 if (Left < j)
  QuickSort(A, Left, j);  if
 (i < Right)
  QuickSort(A, i, Right);
}
// ham main
void main() {
int A[max], n;
 clrscr();
 cout<<"Nhap so phan tu:";
 cin>>n;
```

```
 NhapMang(A,n);
 cout<<"\nMang vua nhap la:";
 XuatMang(A,n);
 cout<<"\nSap xep theo Quick Sort:";
 QuickSort(A,0,n-1);  XuatMang(A,n);
 getch();
}
```

5. Thuật toán sắp xếp vun đống - Heap sort

```
#include <iostream.h>
#include <conio.h>
#define max 100
// Nhap day
void NhapMang(int A[],int n) {
 for(int i=0; i<n; i++) {
  cout<<"nhap Phan tu thu A["<<i<<"] =";
  cin>>A[i];
 }
}
// In day
void XuatMang(int A[],int n) {
 cout<<endl;
 for(int i=0; i<n; i++)
  cout<<A[i]<<"\t";
}
// Đoi cho 2 so
void Swap(int &a,int &b) {
 int temp = a;
 a = b;
 b = temp;
}
//Hoan vi nut cha thu i phai lon hon nut con (vun dong)
```

```
void Heapify(int A[],int n, int i) {
 int Left = 2*(i+1)-1;
 int Right = 2*(i+1);
 int Largest;
 if(Left<n && A[Left]>A[i])
  Largest = Left;
 else
  Largest = i;
 if(Right<n && A[Right]>A[Largest])
  Largest = Right;
 if(i!=Largest) {
  Swap(A[i],A[Largest]);
  Heapify(A,n,Largest);
 }
}
//Xay dung Heap sao cho moi nut cha luon lon hon nut con tren cay (tao cay)
void BuildHeap(int A[], int n) {
 for(int i = n/2-1; i>=0; i--)
  Heapify(A,n,i);
}
// Heap-sort
void HeapSort(int A[], int n) {
 BuildHeap(A,n);
 for(int i = n-1; i>0; i--){
  Swap(A[0],A[i]);
  Heapify(A,i,0);
 }
}
// Ham main
void main() {
int A[max], n;
 clrscr();
```

```
 cout<<"Nhap so phan tu:";
 cin>>n;
 NhapMang(A,n);
 cout<<"\nMang vua nhap la:";
 XuatMang(A,n);
 cout<<"\nSap xep theo Heap Sort:";
 HeapSort(A,n);
 XuatMang(A,n);
 getch();
}
```

6. Thuật toán tìm kiếm tuần tự - Sequential search

+Thực hiện trên dãy khoá chưa sắp xếp

```
int SequentialSearch(int x, int a[],int  n){
int i =1;
  while (i <n &&a[i]!=x)
    i = i+1;
  return (i);  // nếu giá trị trả về là i<=n-1 (tìm thấy), i=n (không tìm thấy)
}
```

+ Thực hiện trên dãy khoá đã được sắp xếp

```
int SequentialSearch (int x, int a[], int n){
int i =1;
  while (i <=n && a[i]<x) i = i+1;

  if (a[i]==x) return 1;
  else return 0; // giá trị trả về là 1 (tìm thấy), 0 (không tìm thấy)
}
```

7. Thuật toán tìm kiếm nhị phân - Binary sort

/*Thuật toán tìm kiếm nhị phân không đệ quy*/

```
int BinarySearch(int x, int a[],int n) // Tìm khoá x trong dãy a1, a2…,an
{ int  left =0; //Left trỏ về chỉ số đầu dãy
 int right =n-1; //right trỏ về vị trí cuối dãy
```

```
 int mid; // mid trỏ vào giữa dãy
 int found = 0; // (0: không tìm thấy, 1: tìm thấy)
 while (left <= right && found==0){
   mid = (left + right) / 2;//mid ở giữa dãy
   if (a[mid]== x) //trường hợp tìm thấy dừng thuật toán
     found = 1;
    else
        if (a[mid]<x)
           left = mid +1; //xác định lại đoạn tìm tiếp theo là bên phải
        else
           right = mid -1; //xác định lại đoạn tìm tiếp theo là bên trái
  }
return found;
}
```

/*Thuật toán tìm kiếm nhị phân đệ quy*/

```
int BinSearch(int[] a, int gt, int dau, int cuoi)
{
   int i, j;
   i = dau; j = cuoi;
   if (i > j)
  {
      return -1; // không tìm thấy
   }
   int giua = (i + j) / 2;
   if (gt == a[giua])
     {
        return giua; // Tìm thấy
      }
   else if (gt < a[giua])
     {
        j = giua - 1;
        return BinSearch(a, gt, dau, j);
```