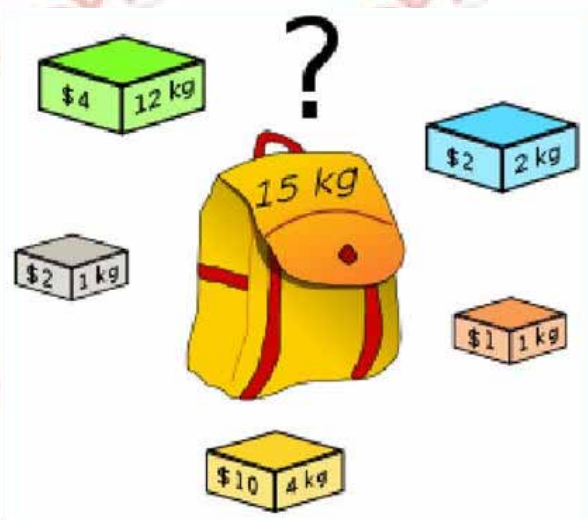
BÀI 4: BÀI TOÁN TỐI ƯU TỔ HỢP



Giới thiệu

Bài học này trình bày nội dung bài toán tối ưu tổ hợp là bài toán chỉ quan tâm đến một cấu hình "tốt nhất" theo một nghĩa nào đấy. Đây là bài toán có nhiều ứng dụng trong thực tiễn và lý thuyết tổ hợp đã đóng góp một phần đáng kể trong việc xây dựng những thuật toán hữu hiệu.

Nội dung

- Giới thiệu bài toán tối ưu tổ hợp
- Bài toàn người du lịch và bài toán cái túi
- Phương pháp duyệt toàn bộ
- Kỹ thuật đánh giá nhánh cận
- Phương pháp tham lam
- Bài toán tìm lịch gia công trên hai máy và thuật toán Johnson

Thời lượng học

- 6 tiết

Mục tiêu

Sau khi học bài này, các bạn có thể:

- Nắm được yêu cầu của bài toán tối ưu tổ hợp, một số bài toán điển hình
- Sử dụng được các phương pháp:
  - Duyệt toàn bộ
  - Đánh giá nhánh cận
  - Tham lam

  trong việc giải quyết bài toán tối ưu tổ hợp

TÌNH HUỐNG DẪN NHẬP

Tình huống: Bài toán người du lịch

Có n thành phố (đánh số từ 1 đến n). Một người du lịch, xuất phát từ thành thành phố s, muốn đi thăm tất cả các thành phố khác, mỗi thành phố đúng một lần, rồi lại quay về nơi xuất phát. Giả thiết biết chi phí đi từ thành phố i đến thành phố j là c(i, j), $1 \leq i, j \leq n$.

Câu hỏi

Hãy tìm một hành trình cho người du lịch sao cho chi phí của hành trình này là nhỏ nhất!

4.1. Giới thiệu bài toán

Bài toán tối ưu tổ hợp không quan tâm đến việc xây dựng tất cả các cấu hình như bài toán liệt kê mà chỉ nhằm xây dựng một cấu hình "tốt" nhất theo một nghĩa nào đấy. Vì thế nó là bài toán có nhiều ý nghĩa thực tiễn hơn cả. Lời giải của nó, cũng giống như bài toán liệt kê, phải được trình bày dưới dạng một thuật giải mà theo từng bước, ta xây dựng được cấu hình cần tìm. Việc thi hành được giao cho máy tính bằng một chương trình thực hiện thuật giải đã nêu.

Độ "tốt" của cấu hình phụ thuộc vào mục tiêu của bài toán và người ta phải lượng hóa chúng để có thể so sánh. Một cách thường làm là xây dựng một hàm f, ứng mỗi cấu hình X được xét với một con số, ký hiệu f(X) (gọi là *giá* của X). Khi đó, độ "tốt" của cấu hình được định nghĩa theo hai hướng: nếu mục tiêu của bài toán là chi phí thì cấu hình càng tốt nếu giá của nó càng nhỏ (như thế cấu hình tốt nhất là cấu hình có giá nhỏ nhất), nếu mục tiêu là hiệu quả thì cấu hình càng tốt nếu giá của nó càng lớn (như thế cấu hình tốt nhất là cấu hình có giá lớn nhất). Bài toán thứ nhất gọi là bài toán tìm min, bài toán thứ hai gọi là bài toán tìm max. Như vậy, bài toán tối ưu tổ hợp có thể phát biểu dưới hình thức toán học như sau:

Tìm $X \in D : f(X) \rightarrow \min (\max)$

trong đó D là tập hữu hạn, gồm các cấu hình thỏa mãn điều kiện của bài toán.

Hàm f được gọi là *hàm mục tiêu*. Tập hợp D được gọi là *miền xác định* hay *miền phương án*. Mỗi phần tử của D được gọi là một *phương án*. Phương án tốt nhất được gọi là *phương án tối ưu*. Giá của phương án tối ưu được gọi là *giá trị tối ưu*. Chú ý rằng do D hữu hạn nên phương án tối ưu bao giờ cũng tồn tại. Có thể có nhiều phương án tối ưu, nhưng giá trị tối ưu là duy nhất.

Trong mỗi bài toán cụ thể, ta phải chỉ rõ các điều kiện xác định D và cách tính hàm f (hàm f có thể tính bằng một công thức hoặc bằng một thủ tục).

Mục dưới đây giới thiệu hai bài toán điển hình của tối ưu tổ hợp là bài toán người du lịch và bài toán cái túi.

4.2. Bài toán người du lịch và bài toán cái túi

4.2.1. Bài toán người du lịch

Bài toán người du lịch được phát biểu như sau: "Có n thành phố (đánh số từ 1 đến n). Một người du lịch, xuất phát từ thành thành phố s, muốn đi thăm tất cả các thành phố khác, mỗi thành phố đúng một lần, rồi lại quay về nơi xuất phát. Giả thiết biết chi phí đi từ thành phố i đến thành phố j là c(i, j), $1 \le i, j \le n$. Hãy tìm một hành trình cho người du lịch sao cho chi phí của hành trình này là nhỏ nhất".

Mỗi hành trình của người du lịch được biểu diễn bằng một hoán vị $X = (x_1, x_2, ..., x_n)$ của {1, 2, ..., n} với $x_1 = s$ (hoán vị này biểu diễn hành trình $x_1 \rightarrow x_2 \rightarrow ... \rightarrow x_{n-1} \rightarrow x_n \rightarrow x_1$). Chi phí của hành trình X được tính bằng công thức $c(x_1, x_2) + c(x_2, x_3) + ... + c(x_n, x_1)$. Như thế, mô hình toán học của bài toán người du lịch là:

Tìm $X \in D : f(X) \rightarrow \min$

trong đó D là tập các hoán vị $X = (x_1, x_2, ..., x_n)$ của {1, 2, ..., n} có $x_1 = s$ (cho trước) và $f(X) = c(x_1, x_2) + c(x_2, x_3) + ... + c(x_n, x_1)$.

Tên gọi bài toán người du lịch mang tính chất tượng trưng, nó dùng để gọi chung cho các bài toán có mô hình toán học như trên mặc dù phát biểu có nội dung khác, chẳng hạn bài toán tìm chu trình sản xuất cho một nhà máy hóa chất sao cho chi phí xúc rửa các thiết bị (như bể chứa, ống dẫn, ...), mỗi khi chuyển từ loại hóa chất này sang loại hóa chất khác của chu trình, là ít nhất.

4.2.2. Bài toán cái túi

Bài toán cái túi được phát biểu như sau: "Có n đồ vật (đánh số từ 1 đến n). Với mỗi đồ vật i, ta biết p_i, v_i lần lượt là các trọng lượng và giá trị của vật đó (i = 1, 2, ..., n). Giả thiết có một cái túi, sức chứa không quá w đơn vị trọng lượng. Hãy tìm một phương án chọn đồ vật bỏ vào túi để có thể mang đi được sao cho tổng giá trị các vật được mang là lớn nhất".

Một phương án chọn đồ vật là một tập con của tập {1, 2, ..., n}, vì thế có thể biểu diễn mỗi phương án như thế như một dãy nhị phân $X = (x_1, x_2, ..., x_n)$, trong đó $x_i = 1$ khi và chỉ khi vật i được chọn (i = 1, 2, ..., n). Tổng trọng lượng của các vật được mang theo phương án này là $p_1x_1 + p_2x_2 + ... + p_nx_n$. Điều kiện các vật được chọn mang đi được là điều kiện tổng này không vượt quá w (sức chứa của cái túi). Tổng giá trị các vật được mang theo phương án X là $v_1x_1 + v_2x_2 + ... + v_nx_n$. Từ đó ta nhận được mô hình toán học của bài toán cái túi như sau:

Tìm $X \in D : f(X) \to \max$

trong đó D là tập hợp các dãy nhị phân $X = (x_1, x_2, ..., x_n)$ thỏa mãn bất đẳng thức $p_1x_1 + p_2x_2 + ... + p_nx_n \le w$ và $f(X) = v_1x_1 + v_2x_2 + ... + v_nx_n$.

Bài toán cái túi có nội dung giống như bài toán của người leo núi trước khi thám hiểm: chọn những vật đem theo sao cho sức anh ta mang được với tổng giá trị sử dụng trong chuyến leo núi là lớn nhất, vì thế bài toán này còn có tên gọi khác là bài toán của người leo núi.

Bài toán người du lịch là thí dụ cho những bài toán tối ưu với mục tiêu là chi phí, còn bài toán cái túi là thí dụ cho những bài toán tối ưu với mục tiêu là hiệu quả. Bạn đọc có thể lấy nhiều những thí dụ như vậy trong những bài toán thực tế. Về phần này, các bạn có thể xem thêm tài liệu tham khảo [2].

4.3. Phương pháp duyệt toàn bộ

Do đặc tính hữu hạn của miền phương án nên cách giải đơn giản nhất (cũng là tự nhiên nhất) một bài toán tối ưu tổ hợp là duyệt tất cả các phương án để so sánh. Sau khi duyệt xong, ta sẽ nhận được phương án tốt nhất (giống như chọn quả cam nặng nhất trong một sọt cam bằng cách so sánh từng quả một). Như thế bài toán tối ưu trên D được giải quyết trên cơ sở liệt kê miền D. Phương pháp giải bài toán tối ưu như vậy được gọi là *duyệt toàn bộ*.

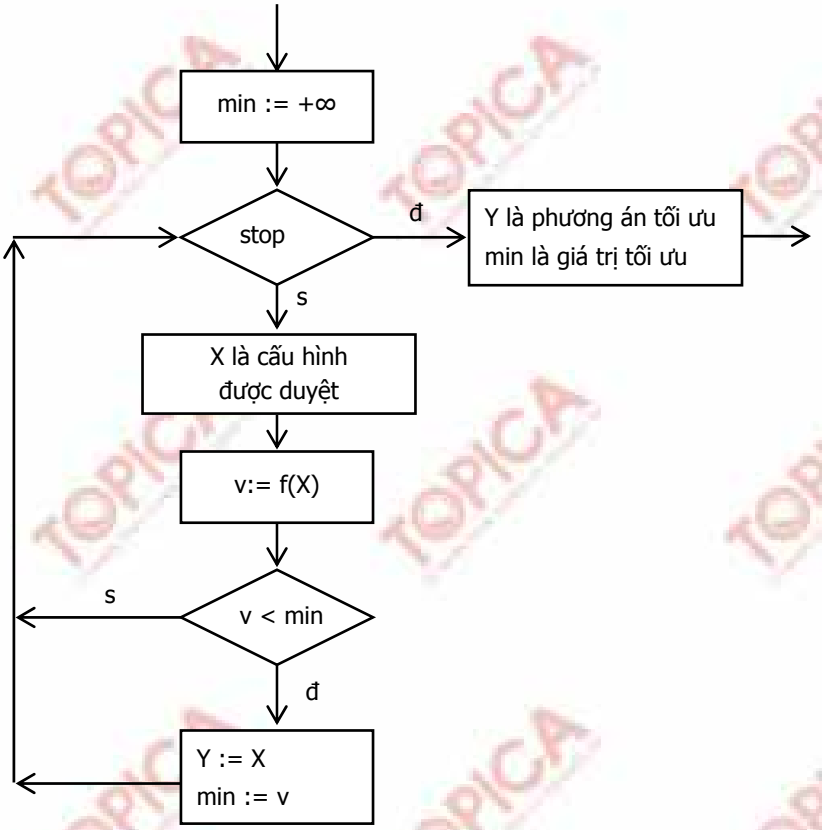
Để cụ thể, giả thiết bài toán có mô hình:

Tìm $X \in D : f(X) \to \min$

Gọi X là phương án duyệt, Y là phương án tốt nhất tại thời điểm được duyệt (gọi là phương án kỷ lục) và min là giá trị của phương án này (gọi là giá trị kỷ lục), stop là biến lôgic kiểm tra điều kiện kết thúc liệt kê. Khi đó phương pháp duyệt toàn bộ được mô tả bởi sơ đồ khối trên hình vẽ.

Vòng lặp trong sơ đồ là vòng lặp liệt kê, mỗi lần lặp, một phương án X được duyệt. Khi đó cần so sánh giá của phương án này với giá trị kỷ lục. Nếu X tốt hơn (phá kỷ lục) thì cần lưu lại X vào phương án kỷ lục Y và ghi nhận kỷ lục mới vào min. Khi liệt kê kết thúc, ta nhận được Y là phương án tối ưu và min là giá trị tối ưu. Việc khởi động giá trị min là cực lớn trước khi liệt kê nhằm đảm bảo kỷ lục được phá ít nhất một lần.

Với mô hình tìm max, ta khởi động giá trị kỷ lục cực nhỏ ($-\infty$) và đảo chiều bất đẳng thức khi so sánh.



Để cài đặt chương trình, ta dùng mô hình quay lui cho vòng lặp liệt kê như đã nêu trong thủ tục TRY(i) (xem bài 3). Trong thủ tục này, ta thay khối (*ghi nhận một cấu hình*) bằng khối (*ghi nhận kỷ kục*):

Thuật toán quay lui cho vòng lặp liệt kê

```
PROCEDURE TRY (i: INTEGER);
VAR j: INTEGER;
BEGIN
   FOR (j thuộc Si) DO
   IF (chấp nhận j) THEN
   BEGIN
         xi := j;
         (ghi nhận trạng thái mới);
         IF (i = n) THEN (ghi nhận kỷ kục) ELSE TRY(i+1);
         (trả về trạng thái cũ);
   END;
END;
```

Nội dung của khối (*ghi nhận kỷ kục*) là so sánh giá của cấu hình được duyệt với giá trị kỷ lục hiện thời, nếu giá này tốt hơn thì phải ghi nhận lại kỷ lục mới. Nên thiết kế một thủ tục riêng để thực hiện nhiệm vụ này.

Thí dụ, giải bài toán người du lịch đã nêu trong 4.2.1, mô hình quay lui được dùng là liệt kê hoán vị (xem mục 3.3.2, bài 3):

Thuật toán quay lui giải bài toán người du lịch

```
PROCEDURE TRY (i: INTEGER);
VAR j: INTEGER;
BEGIN
   FOR j := 1 TO n DO
   IF (bj) THEN
   BEGIN
         xi := j;
         bj := FALSE;
         IF (i = n) THEN SCORE ELSE TRY(i+1);
         bj := TRUE;
   END;
END;
```

Trong đó thủ tục SCORE (ghi nhận kỷ lục) được thay cho thủ tục OUT (đưa cấu hình tìm được ra màn hình).

Trong thủ tục INIT, cần nhập n (số thành phố), s (thành phố xuất phát), c (bảng chi phí) và khởi tạo x_1 bằng s, khởi tạo các b_j bằng TRUE ngoại trừ b_s bằng FALSE, khởi tạo min bằng giá trị lớn nhất của kiểu dữ liệu của nó (chẳng hạn nếu là số thực thì có thể chọn giá trị này bằng 10^{37}, nếu là số nguyên 2 byte không dấu thì có thể chọn giá trị này bằng 65535, ...).

Nội dung chương trình chính giống như bài toán liệt kê, trong đó thay lời gọi TRY(1) bằng lời gọi TRY(2) (vì x_1 đã biết) và thêm vào thao tác đưa ra kết quả tìm được (gồm phương án và giá trị tối ưu) trước khi kết thúc.

Dưới đây là kết quả chạy từng bước của bài toán người du lịch với 4 thành phố {1, 2, 3, 4}, xuất phát từ thành phố 2 và bảng chi phí:

	1	2	3	4
1	0	3	3	5
2	5	0	1	3
3	2	2	0	3
4	4	3	2	0

Có 3! = 6 hành trình được duyệt, lần lượt như sau:

1) 2 1 3 4 2 chi phí 5 + 3 + 3 + 3 = 14 ghi nhận kỷ lục
2) 2 1 4 3 2 chi phí 5 + 5 + 2 + 2 = 14
3) 2 3 1 4 2 chi phí 1 + 2 + 5 + 3 = 11 ghi nhận kỷ lục
4) 2 3 4 1 2 chi phí 1 + 3 + 4 + 3 = 11
5) 2 4 1 3 2 chi phí 3 + 4 + 3 + 2 = 12
6) 2 4 3 1 2 chi phí 3 + 2 + 2 + 3 = 10 ghi nhận kỷ lục

Hành trình ứng với lần ghi kỷ lục cuối cùng là phương án tối ưu 2→4→3→1→2 với chi phí thấp nhất là 10.

Cũng dễ nhận thấy rằng vì hành trình là khép kín đi qua mọi thành phố nên vị trí xuất phát của hành trình là không quan trọng.

Bạn đọc có thể giải bài toán cái túi trong mục 4.2.2 bằng cách dùng mô hình quay lui liệt kê dãy nhị phân (xem mục 3.3.1, bài 3). Với mỗi dãy nhị phân (x_1, x_2, ..., x_n) được duyệt, cần thử lại bất đẳng thức $p_1x_1 + p_2x_2 + ... + p_nx_n \leq w$ cho phương án này, nếu thỏa mãn thì thủ tục SCORE (*ghi nhận kỷ lục*) được áp dụng.

Thuật toán quay lui giải bài toán cái túi

```
PROCEDURE TRY (i: INTEGER);
VAR j: INTEGER;
BEGIN
   FOR j := 0 TO 1 DO
   BEGIN
         x[i] := j;
         IF (i = n) THEN
         BEGIN
             IF (p1x1+p2x2+ ... +pnxn ≤ w) THEN SCORE;
          END ELSE TRY(i+1);
   END;
END;
```

Cũng có thể kiểm tra điều kiện mang đi được ngay tại mỗi lần xác định x_i để bớt được việc thử một số nhánh không cần thiết bằng cách tổ chức thêm một tham số r cho thủ tục TRY để truyền vào thủ tục này khả năng còn lại của túi tại bước thử thứ i. Khi đó điều kiện chấp nhận j cho x_i là $r \geq j$ và thủ tục TRY có nội dung như sau (không mất tính tổng quát, ta giả thiết trọng lượng các vật là các số nguyên):

```
PROCEDURE TRY (r, i: INTEGER);
VAR j: INTEGER;
BEGIN
   FOR j := 0 TO 1 DO
   IF (r >= j) THEN
   BEGIN
         x[i] := j;
         IF (i = n) THEN SCORE ELSE TRY(r >= j, i+1);
   END;
END;
```

Lời gọi khởi động thủ tục này trong chương trình chính sẽ là TRY(w, 1) (xác định vật chọn thứ nhất với khả năng của túi là w).

Thuật toán duyệt toàn bộ được áp dụng cho tất cả các bài toán tối ưu tổ hợp mà bài toán liệt kê các phương án đã được giải vì nó không phụ thuộc vào tính chất của hàm

mục tiêu. Hạn chế của phương pháp này là tính khả thi thấp vì số lượng các phương án phải duyệt thường quá lớn. Chẳng hạn với bài toán người du lịch 16 thành phố, số phương án phải duyệt là 15! = 1 307 674 368 000. Giả thiết máy tính mỗi giây duyệt được 10 triệu cấu hình, khi đó để duyệt hết, ta cần khoảng 130 764 giây, nghĩa là khoảng 36 giờ.

Tuy nhiên, việc có một thuật toán hiệu quả để giải một bài toán tối ưu không phải dễ dàng. Nhiều bài toán hiện nay vẫn chưa có cách gì giải quyết ngoài việc duyệt. Vì thế để nâng cao hiệu quả của cách giải này, người ta cố gắng tìm những giải pháp để hạn chế khối lượng duyệt. Một trong những giải pháp thường dùng là kỹ thuật đánh giá nhánh cận được trình bày trong mục dưới đây.

4.4. Kỹ thuật đánh giá nhánh cận

Trong mô hình duyệt toàn bộ đã trình bày trong mục trước, phương án phải được xây dựng xong rồi mới tính giá của phương án đó để so sánh. Điều này dẫn đến việc tính toán khá nhiều. Sở dĩ như vậy, vì ta chưa khai thác những đặc điểm của hàm mục tiêu, mà nếu để ý, rất có thể đã phát hiện được phương án đang xây dựng chắc chắn không tốt hơn kỷ lục hiện có, và nếu khẳng định được điều này, ta có thể chuyển sang xây dựng phương án khác, bỏ qua được một số nhánh tìm kiếm vô ích.

Giả sử phương án đã xây dựng xong i thành phần $x_1, x_2, ..., x_i$. Dù chưa tính được giá của toàn bộ phương án, nhưng bằng cách nào đó, ta đánh giá được một giới hạn (cận) chung cho những giá trị này (đối với bài toán tìm min là cận dưới, đối với bài toán tìm max là cận trên). Nếu cận tính được không tốt hơn kỷ lục hiện có (đối với bài toán tìm min là không nhỏ hơn, đối với bài toán tìm max là không lớn hơn) thì có nghĩa là hướng phát triển của nhánh tìm kiếm này là vô ích, có thể bỏ qua để xét giá trị khác cho x_i. Việc không xét những giá trị tiếp theo của x_i giúp cho loại bỏ được một loạt các nhánh trên cây tìm kiếm. Vì thế kỹ thuật đánh giá này có tên gọi là *đánh giá nhánh cận* (tìm cận tại mỗi nhánh tìm kiếm).

Để đánh giá được nhánh cận, cần có sự xem xét kỹ tính chất của hàm mục tiêu và điều này không đơn giản. Thông thường, cận được đánh giá cố gắng đạt được hai tiêu chí:

- Càng sát với giá trị tối ưu của bài toán càng tốt
- Việc tính cận càng đơn giản càng tốt

Tiêu chí thứ nhất giúp cho việc lùi càng sớm trên cây tìm kiếm, nghĩa là càng cắt được nhiều nhánh trên cây này, tiêu chí thứ hai làm giảm bớt các phép tính trong một vòng lặp đệ quy (mà số lượng lồng nhau của chúng là rất lớn!). Trên thực tế hai tiêu chí này thường xung đột lẫn nhau: để đánh giá cận càng sát, việc tính nó càng phức tạp. Việc điều chỉnh hai tiêu chí này cho phù hợp là cả một nghệ thuật, nó đòi hỏi nhiều kinh nghiệm và kiến thức trong việc đánh giá các bất đẳng thức.

Bây giờ, giả sử $g(x_1, x_2, ..., x_i)$ là cận dưới tương ứng với bước thứ i của bài toán tìm min. Khi đó trong thủ tục TRY(i) của mô hình duyệt toàn bộ, trước khi gọi TRY(i +1), ta cần thử lại bất đẳng thức $g(x_1, x_2, ..., x_i) < \min$ (nghĩa là chỉ tiến sang bước sau nếu cận dưới nhỏ hơn kỷ lục hiện thời):

```
PROCEDURE TRY (i: INTEGER);
VAR j: INTEGER;
BEGIN
   FOR (j thuộc Si) DO
   IF (chấp nhận j) THEN
   BEGIN
         xi := j;
         (ghi nhận trạng thái mới);
         IF (i = n) THEN (ghi nhận kỷ kục)
            ELSE IF (g(x1, x2, ..., xi) < min) THEN TRY(i+1);
         (trả về trạng thái cũ);
   END;
END;
```

Dưới đây là một thí dụ minh họa đánh giá nhánh cận trong việc giải bài toán người du lịch.

Ví dụ:

Giả sử đang ở bước thứ i của hành trình người du lịch, khi đó chi phí của hành trình tính đến bước này là hoàn toàn xác định và được tính bằng:

$$t_i = c(x_1, x_2) + ... + c(x_{i-1}, x_i)$$

Từ bước này đến lúc kết thúc hành trình còn n − i + 1 bước nữa (n − i bước để đến thành phố cuối cùng và 1 bước để quay lại thành phố xuất phát). Chi phí của mỗi bước, trong mọi lựa chọn có thể, đều không thể ít hơn c_{min} là giá trị nhỏ nhất của toàn bộ bảng chi phí (trừ tại đường chéo c(i, i), i = 1, 2, ..., n là những giá trị không được dùng). Từ đó nhận được một cận dưới của bước đang xét là:

$$g(x_1, x_2, ..., x_i) = t_i + (n - i + 1).c_{min}$$

Để tính t_i trong từng bước, ta nên tổ chức thêm một tham số t nữa (ngoài tham số i) cho thủ tục đệ quy TRY để truyền giá trị này của bước trước vào. Khi đó, việc đánh giá nhánh cận vừa trình bày, được đưa vào thủ tục TRY của bài toán người du lịch như sau (giả thiết các chi phí được khai báo kiểu INTEGER):

```
PROCEDURE TRY (t, i: INTEGER);
VAR j: INTEGER;
BEGIN
   FOR j := 1 TO n DO
   IF (bj) THEN
   BEGIN
        xi := j;
        bj := FALSE;
         IF (i = n) THEN SCORE
        ELSE IF (t+c(xi-1, xi)+(n-i+1).cmin < min)
        THEN TRY(t+c(xi-1, xi), i+1);
          bj := TRUE;
   END;
END;
```

Lời gọi khởi động thủ tục này trong chương trình chính là TRY(0, 2) (xác định thành phố thứ 2 trong hành trình với chi phí ban đầu bằng 0). Đánh giá nhánh cận vừa trình bày cho bài toán người du lịch là đơn giản và dễ hiểu nhưng khá thô. Để thực sự nâng cao

hiệu quả người ta cần phải đánh giá tinh vi hơn nữa (và việc cài đặt cũng sẽ phức tạp hơn). Về phần này, các bạn nên xem thêm tài liệu tham khảo [2], trong đó có trình bày việc đánh giá nhánh cận bài toán người du lịch theo cách khác, phức tạp hơn nhưng hiệu quả hơn.

Cuối cùng cũng nên chú ý rằng, mặc dù trong trường hợp tồi nhất, việc đánh giá nhánh cận cũng không khác gì (hoặc khác không đáng kể) so với việc duyệt toàn bộ, nhưng trong những tình huống thông thường và với kích thước không lớn lắm, việc có đánh giá nhánh cận tỏ ra tốt hơn nhiều so với việc không đánh giá. Điều này có thể minh chứng bằng việc viết các chương trình chạy trên máy tính, trong đó có đếm số nhánh thực sự phải duyệt để so sánh.

Chẳng hạn xét bài toán người du lịch 6 thành phố, xuất phát từ thành phố 1, với bảng chi phí dưới đây:

```
0 9 6 3 8 8
6 0 7 3 4 5
3 9 0 9 8 9
3 8 8 0 5 4
3 4 4 7 0 9
9 4 3 3 9 0
```

Chương trình đánh giá nhánh cận như đã trình bày (có kèm việc đếm số lượng duyệt) cho ta kết quả số nhánh thực sự phải duyệt là 120 (so với toàn bộ là 325), trong đó số nhánh đi đến phương án đầy đủ là 19 (so với toàn bộ là 120) và nhận được hành trình tối ưu 1→4→6→2→5→3→1 với chi phí là 22. Điều đáng chú ý là, cùng bài toán này, nhưng xuất phát khác nhau, ta nhận được các khối lượng duyệt theo nhánh cận là khác nhau.

Bạn đọc thử đưa ra một cách đánh giá nhánh cận với bài toán cái túi và viết một chương trình theo mô hình vừa trình bày để chạy thử trên máy tính.

Có thể nói rằng những thuật toán hiệu quả giải các bài toán có nhiều ứng dụng trên thực tế là khá hiếm. Phần nhiều không tránh khỏi việc phải duyệt toàn bộ khi giải những bài toán này vì thế kỹ thuật đánh giá nhánh cận được quan tâm đặc biệt. Việc tìm được một đánh giá tốt cho những mô hình có nhiều ứng dụng giúp cho việc tìm kiếm lời giải được cải thiện lên rất nhiều và những kết quả này quả thật là những cống hiến đáng kể cho khoa học, nhất là trong thời kỳ máy tính được dùng rộng rãi như hiện nay.

4.5. Phương pháp tham lam

Như đã nói trong mục trên, hầu hết các bài toán thực tế có mô hình tối ưu tổ hợp, được giải bằng cách duyệt, dù có đưa đánh giá nhánh cận vào cũng chỉ giảm được thời gian tìm kiếm đến một giới hạn nhất định, trong khi trên thực tế, thời gian chờ đợi kết quả thường bị khống chế, nếu vượt khoảng thời gian này, kết quả dù có tốt cũng không còn ý nghĩa gì (chẳng hạn những bài toán về dự báo). Đấy là chưa kể việc tìm được một đánh giá nhánh cận thỏa mãn cả hai tiêu chí (vừa sát, vừa đơn giản) là một điều không dễ dàng. Vì thế, bên cạnh những phương pháp cho lời giải đúng, người ta còn quan tâm đến những cách giải gần đúng, trong đó nghiệm được tìm là tương đối tốt, có thể chấp nhận được, nhưng thời gian tìm kiếm phải nhanh.

Một trong những giải pháp để đạt được ý tưởng này là tìm cách tối ưu cục bộ thay cho tối ưu toàn cục. Các bài toán tối ưu cục bộ, do kích thước nhỏ, có thể giải bằng cách duyệt để được các phương án tối ưu cục bộ. Phương án thu được là sự kết hợp các phương án tối ưu cục bộ đã tìm. Nói chung phương án này không phải là tối ưu, nhưng trong một số tình huống nó có thể chấp nhận được. Các phương pháp này có tên gọi chung là các *phương pháp tham lam* (gredy).

Nội dung của "tham lam" rất đa dạng, nó phụ thuộc vào việc tổ chức các bài toán cục bộ, hàm mục tiêu của những bài toán này, và sự kết hợp các lời giải cục bộ để được lời giải cần tìm. Một giải pháp tham lam được đánh giá là tốt, nếu phần lớn các tình huống thực tế, nó cho một lời giải sát với phương án tối ưu mà thời gian tìm kiếm vẫn nằm trong phạm vi cho phép. Vì thế, bên cạnh việc nghiên cứu kỹ mục tiêu bài toán, người ta còn để ý khai thác những đặc điểm của những dữ liệu mà thực tế cung cấp, nhờ những đặc điểm này, nhiều khi tìm được những cách tham lam cho hiệu quả cao, phù hợp với những dữ liệu thực tế mà không nhất thiết phù hợp trong những trường hợp tổng quát.

Dưới đây trình bày một cách giải tham lam đối với bài toán người du lịch, nó chưa được hiệu quả lắm nhưng đơn giản và dễ hiểu.

Bắt đầu từ thành phố xuất phát s, gán x_1 bằng s, tìm thành phố tiếp theo có chi phí đi từ s đến đó là ít nhất (tìm giá trị nhỏ nhất của các c(s, j), $j \neq s$) và gán x_2 bằng số hiệu của thành phố này, ..., một cách tổng quát, từ thành phố x_{i-1}, tìm thành phố x_i là thành phố chưa đến, có chi phí đi từ x_{i-1} là ít nhất (i = 1, 2, ..., n). Hành trình $x_1, x_2, ..., x_n$ là kết quả cần tìm. Rõ ràng không thể khẳng định đây là hành trình tối ưu mặc dù nó được lựa chọn tốt nhất trong từng bước. Tuy nhiên việc tìm ra nó chỉ cần thực hiện $\frac{n(n-1)}{2}$ phép so sánh, hầu như cho kết quả ngay tức khắc với n lên đến hàng trăm.

Giải lại bài toán người du lịch trong thí dụ mục 4.4 bằng thuật toán tham lam vừa trình bày ta được hành trình (sau 10 phép so sánh) là 1→4→6→3→5→2→1 có chi phí 28 (so với hành trình tối ưu 1→4→6→2→5→3→1 có chi phí 22). Chú ý rằng khi thay đổi thành phố xuất phát, ta nhận được các hành trình này là khác nhau (dù tối ưu không thay đổi):

1→4→6→3→5→2→1	chi phí 28
2→4→1→3→5→6→2	chi phí 33
3→1→4→6→2→5→3	chi phí 22
4→1→3→5→2→6→4	chi phí 29
5→1→4→6→3→2→5	chi phí 26
6→3→1→4→5→2→6	chi phí 23

Do đó để nâng cao độ tốt của lời giải, ta có thể lặp lại thao tác trên và chọn kết quả tốt nhất trong chúng. Trong thí dụ trên ta được hành trình tốt nhất là 3→1→4→6→2→5→3 trùng với hành trình tối ưu (chú ý rằng, không phải trường hợp nào cũng may mắn như vậy!).

Từ thí dụ trên, ta cũng thấy rằng để tăng độ chính xác của lời giải, người ta thường chọn nhiều cách tham lam khác nhau (tương ứng với nhiều hướng ưu tiên khác nhau), rồi so sánh để chọn ra kết quả tốt nhất. Cách làm này nhiều khi cho lời giải khá tốt mà

thời gian tìm kiếm nhanh hơn nhiều so với việc phải duyệt toàn bộ (kể cả có đánh giá nhánh cận).

Ngoài ra, kết quả tìm kiếm tham lam cũng có thể dùng làm phương án xuất phát cho cách giải đánh giá nhánh cận, nó thường cho giá trị kỷ lục ban đầu khá tốt, điều này làm cho quá trình đánh giá nhánh cận phát hiện được những hướng tìm kiếm vô ích sớm hơn và vì thế tốc độ tìm kiếm cũng nhanh hơn.

Chẳng hạn để giải bài toán người du lịch trong mục 4.4 bằng đánh giá nhánh cận, có cài thêm thao tác tham lam như đã trình bày để tìm phương án ban đầu (trong thí dụ này, kỷ lục ban đầu là 22), số lượng nhánh phải tính là 18 (so với 120 nếu xuất phát từ kỷ lục $+\infty$) trong đó có 2 nhánh đi đến phương án đầy đủ (so với 19 nếu xuất phát từ kỷ lục $+\infty$) và kết quả nhận được hành trình tối ưu là phương án ban đầu.

Bạn đọc thử đưa ra một giải pháp tham lam cho bài toán cái túi và viết chương trình thực hiện chúng theo như cách đã làm với bài toán người du lịch.

4.6. Bài toán tìm lịch gia công trên hai máy và thuật toán Johnson

Có thể nói rằng, không có một thuật toán hữu hiệu nào áp dụng cho một lớp rộng rãi các bài toán tối ưu tổ hợp. Tuy nhiên trong một số bài toán, đặc biệt những bài toán có nhiều ý nghĩa ứng dụng thực tế, người ta cố gắng khai thác tối đa các đặc điểm của chúng để tìm ra những giải pháp đặc biệt, chỉ áp dụng riêng cho chúng, có lời giải đúng với thời gian tìm kiếm khả thi. Mặc dù những giải pháp này không nhiều và rất khó mở rộng, nhưng chúng có một ý nghĩa lý thuyết và ứng dụng rất lớn. Mục này giới thiệu một bài toán như vậy.

4.6.1. Phát biểu bài toán

"Có n chi tiết (đánh số từ 1 đến n) cần được lần lượt gia công trên hai máy, đầu tiên qua máy A, sau đó mới đến máy B. Giả thiết biết thời gian gia công chi tiết i tương ứng trên hai máy A, B là a_i và b_i (i = 1, 2, ..., n). Hãy tìm một lịch gia công (thứ tự gia công) để thời điểm hoàn thành tất cả các chi tiết là sớm nhất".

Nội dung của bài toán mô tả một yêu cầu thường thấy trong thi công: một công trình gồm nhiều hạng mục, mỗi hạng mục phải qua nhiều công đoạn với thứ tự thực hiện các công đoạn là xác định (chẳng hạn đào móng xong mới được xây tường, xây tường xong mới được lắp cửa, ...). Cần bố trí thứ tự thực hiện các hạng mục này để thời gian các nhóm thợ (mỗi nhóm thợ thực hiện một công đoạn) phải chờ đợi nhau là ít nhất.

Bài toán đã nêu xét trường hợp đơn giản nhất gồm hai công đoạn: máy A tượng trưng cho giai đoạn gia công thô, máy B tượng trưng cho giai đoạn gia công tinh. Mọi chi tiết phải qua giai đoạn gia công thô (máy A) rồi mới đến giai đoạn gia công tinh (máy B).

Chú ý:

Không có một giả thiết ràng buộc nào giữa các thời gian a_i và b_i, chúng có thể bằng, lớn hơn hay nhỏ hơn nhau tại bất cứ chi tiết i nào.

Một lịch gia công các chi tiết được biểu diễn như một hoán vị $X = (x_1, x_2, ..., x_n)$ của tập {1, 2, ..., n}. Giá trị hàm mục tiêu T(X) của lịch X là thời điểm hoàn thành tất cả các chi tiết theo lịch này, nghĩa là thời điểm máy B hoàn thành chi tiết cuối cùng theo lịch X (không mất tính tổng quát, ta có thể xem thời điểm bắt đầu làm việc, nghĩa là Gọi $T_A(i)$ là thời điểm máy A hoàn thành chi tiết x_i của lịch X. Vì máy A không phụ

thuộc máy B nên thời điểm máy A bắt đầu chi tiết x_i là thời điểm nó hoàn thành chi tiết x_{i-1}, từ đó nhận được công thức truy hồi tính các $T_A(i)$ là:

(1) $T_A(i) = T_A(i-1) + a_i$, i = 1, 2, ..., n với giá trị ban đầu $T_A(0) = 0$

Gọi $T_B(i)$ là thời điểm máy B hoàn thành chi tiết x_i của lịch X. Vì máy B chỉ có thể bắt đầu chi tiết x_i nếu nó đã xong chi tiết x_{i-1}, đồng thời máy A phải xong chi tiết x_i, nghĩa là tại thời điểm muộn nhất của hai thời điểm này. Từ đó nhận được công thức truy hồi tính các $T_B(i)$ là:

(2) $T_B(i) = \max\{T_B(i-1), T_A(i)\} + b_i$, i = 1, 2, ..., n với giá trị ban đầu $T_B(0) = 0$

Thời điểm máy B hoàn thành chi tiết cuối cùng $T_B(n)$ của lịch X là thời điểm T(X) hoàn thành lịch này.

Giá trị T(X) được tính qua các giá trị trung gian $T_A(i)$, $T_B(i)$ bằng một vòng lặp đơn giản mà ta có thể dễ dàng xây dựng một hàm để tính nó.

Như vậy, dưới dạng toán học, bài toán tìm lịch gia công trên hai máy có thể phát biểu như sau:

Tìm $X \in D : T(X) \to \min$

trong đó D là tập các hoán vị của {1, 2, ..., n} và T(X) được xác định theo các công thức truy hồi vừa trình bày.

Thí dụ: Xét bài toán gia công 5 chi tiết trên hai máy A, B có các thời gian gia công cho bởi bảng sau:

chi tiết	1	2	3	4	5
máy A	3	4	6	5	6
máy B	3	3	2	7	3

Giả sử các chi tiết được gia công theo lịch X = (5, 3, 4, 1, 2). Các $T_A(i)$, $T_B(i)$ được tính theo các công thức truy hồi (1), (2) bằng cách lần lượt điền vào bảng sau:

lịch X	5	3	4	1	2
$T_A(i)$	6	12	17	20	24
$T_B(i)$	9	14	24	27	30

Từ đó nhận được T(X) = 30. Thời gian máy B phải chờ theo lịch này là 6 + 3 + 3 = 12.

Tương tự với lịch Y = (2, 3, 1, 4, 5), kết quả tính các $T_A(i)$, $T_B(i)$ được điền trong bảng sau:

lịch Y	2	3	1	4	5
$T_A(i)$	4	10	13	18	24
$T_B(i)$	7	12	16	25	28

và ta được T(Y) = 28. Với lịch này, thời gian B phải chờ là 4 + 3 + 1 + 2 = 10.

Thí dụ trên cho thấy, với những lịch khác nhau, thời điểm hoàn thành là khác nhau (vì thời gian máy B phải chờ là khác nhau). Kết quả trên cũng chứng tỏ rằng lịch Y tốt hơn lịch X. Vấn đề là cần tìm lịch X^* tốt nhất, nghĩa là có $T(X^*)$ nhỏ nhất.

Dĩ nhiên để giải bài toán này, ta có thể dùng giải pháp tầm thường là duyệt tất cả các lịch có thể có. Với mỗi lịch X được duyệt, cần tính T(X) để giữ lại lịch tốt nhất theo như mô hình duyệt toàn bộ đã trình bày. Chẳng hạn, với thí dụ trên ta cần duyệt

5! = 120 lịch X và tính ngần ấy các giá trị T(X) để so sánh. Rõ ràng là với cách giải này, ta chỉ có thể thực hiện được với kích thước n không đáng kể, ngay cả có đánh giá nhánh cận.

Nhà toán học Mỹ Johnson, đã tìm ra một thuật toán với độ đơn giản đáng ngạc nhiên, cho phép tìm lịch tối ưu với thời gian đa thức.

4.6.2. Thuật toán Johnson

Nội dung của thuật toán có thể trình bày đơn giản như sau: đầu tiên ghi dữ liệu của bài toán vào một bảng n cột, 2 dòng, trong đó mỗi cột ứng với một chi tiết, dòng đầu ghi các thời gian a_i (máy A), dòng thứ hai ghi các thời gian b_i (máy B) như thí dụ đã trình bày:

chi tiết	1	2	.....	n−1	n
máy A	a_1	a_2	.....	a_{n-1}	a_n
máy B	b_1	b_2	.....	b_{n-1}	b_n

sau đó chuẩn bị một băng gồm n ngăn để trống và lần lượt xếp mỗi chi tiết vào một ngăn theo các bước sau (bắt đầu từ bảng thời gian đã cho):

1) tìm giá trị bé nhất của bảng, nếu giá trị này thuộc dòng máy A (dòng 1) thì xếp chi tiết tương ứng vào ngăn bên trái nhất (còn rỗng) của băng, nếu giá trị này thuộc dòng máy B (dòng 2) thì xếp chi tiết tương ứng vào ngăn bên phải nhất (còn rỗng) của băng.

2) xóa cột ứng với chi tiết ra khỏi bảng và lặp lại bước 1) cho đến khi tất cả các chi tiết đều được xếp.

Khi kết thúc, lịch tối ưu được đọc từ trái sang phải của băng.

Nếu ký hiệu $x_1, x_2, ..., x_n$ là các chi tiết được xếp theo lịch thì thuật toán trên có thể biểu diễn bằng đoạn chương trình sau:

```
left := 1; {khởi động vị trí đầu trái}
right := n; {khởi động vị trí đầu phải}
WHILE (left ≤ right) DO
BEGIN
(tìm giá trị bé nhất của bảng và giả sử giá trị này đạt được tại dòng i và cột ứng với chi tiết j);
   IF (i = 1) THEN {thuộc dòng máy A}
   BEGIN
        x_left := j; left := left +1; {xếp chi tiết j vào ngăn bên trái nhất của băng}
    END ELSE {thuộc dòng máy B}
    BEGIN
        x_right := j; right := right -1; {xếp chi tiết j vào ngăn bên phải nhất của băng}
    END;
    (xóa cột ứng với chi tiết j ra khỏi bảng);
END;
```

Thuật toán gồm một vòng lặp *n* lần (mỗi lần xếp xong một chi tiết). Mỗi vòng lặp chủ yếu là thao tác tìm giá trị bé nhất của một tập hữu hạn (ban đầu gồm *2n* phần tử, sau mỗi lần lặp lại bớt đi 2 phần tử), vì thế không kể các phép gán và phép dịch chuyển

chỉ số, thuật toán gồm 2n + (2n − 2) + ... + 2 = n(n + 1) phép so sánh, một khối lượng tính toán rất ít so với việc duyệt toàn bộ.

Chú ý: Việc tính T(X) chỉ cần thực hiện một lần đối với lịch tối ưu X đã được tìm.

Bạn đọc có thể dễ dàng cài đặt chương trình thực hiện thuật toán Johnson và chạy thử với kích thước hàng trăm chi tiết.

Với thí dụ đã nêu, kết quả từng bước tính theo thuật toán Johnson được trình bày trên bảng thời gian và băng xếp các chi tiết như sau (giá trị bé nhất tìm được của bảng, được khoanh trong cặp ngoặc tròn):

Bước 1: Bảng thời gian

chi tiết	1	2	3	4	5
máy A	3	4	6	5	6
máy B	3	3	(2)	7	3

Băng xếp các chi tiết

				3

Bước 2: Bảng thời gian

chi tiết	1	2	4	5
máy A	(3)	4	5	6
máy B	3	3	7	3

Băng xếp các chi tiết

1				3

Bước 3: Bảng thời gian

chi tiết	2	4	5
máy A	4	5	6
máy B	(3)	7	3

Băng xếp các chi tiết

1			2	3

Bước 4: Bảng thời gian

chi tiết	4	5
máy A	5	6
máy B	7	(3)

Băng xếp các chi tiết

1		5	2	3

Bước 5: Bảng thời gian

chi tiết	4
máy A	(5)
máy B	7

Băng xếp các chi tiết

1	4	5	2	3

Băng xếp các chi tiết cuối cùng cho lịch tối ưu X^* = (1, 4, 5, 2, 3) và thời điểm hoàn thành của nó được tính từ bảng sau:

lịch X^*	1	4	5	2	3
$T_A(i)$	3	8	14	18	24
$T_B(i)$	6	15	18	21	26

Nghĩa là $T(X^*) = 26$. Thời gian máy B phải chờ theo lịch này là 3 + 2 + 3 = 8.

Với thí dụ này bạn đọc cũng thấy khối lượng tính toán là rất ít so với việc phải duyệt 120 lịch và phải tính tất cả các thời điểm hoàn thành của chúng.

Chú ý:

- Thời điểm máy A hoàn thành công việc bằng tổng thời gian làm việc của nó $a_1 + a_2 ... + a_n$, không phụ thuộc vào lịch.
- Thời điểm máy B hoàn thành công việc bằng tổng thời gian làm việc của nó $b_1 + b_2 ... + b_n$ cộng với thời gian mà nó phải chờ, vì thế lịch tốt nhất cũng là lịch mà máy B phải chờ ít nhất.
- Trong việc tìm giá trị bé nhất của bảng, vị trí của chúng là không duy nhất, ta có thể chọn ô nào cũng được, điều này dẫn đến có thể có nhiều lịch tối ưu khác nhau (nhưng giá trị tối ưu là duy nhất).
- Trên thực tế, để tránh máy B phải khởi động nhiều lần (do quá trình nghỉ và làm việc xen kẽ), người ta có thể dồn tất cả khoảng thời gian mà máy B phải chờ lên đầu để máy B hoạt động liên tục theo lịch đã tìm. Chẳng hạn trong thí dụ trên, theo lịch tối ưu X^*, máy B có thể bắt đầu từ thời điểm 8 (thay vì thời điểm 3), làm việc liên tục và kết thúc vào thời điểm 26:

lịch X^*	bắt đầu	1	4	5	2	3
$T_B(i)$	8	11	18	21	24	26

Để chứng minh thuật toán này, Johnson đã khảo sát quy luật thay đổi của hàm mục tiêu T(X) khi thay X bởi lịch nhận được từ X bằng cách hoán vị hai thành phần kề nhau, từ đó nhận được một định lý là cơ sở cho thuật toán đã nêu. Chi tiết các chứng minh có thể tham khảo tài liệu [2]. Kỹ thuật chứng minh của Johnson là một kỹ thuật cơ bản trong lý thuyết lập lịch, có tên gọi là *thủ thuật hoán vị*.

4.6.3. Mở rộng bài toán cho 3 máy

Dễ dàng mở rộng phát biểu của bài toán tìm lịch gia công trên 2 máy cho nhiều máy. Đơn giản nhất là cho 3 máy như sau:

"Có *n* chi tiết (đánh số từ 1 đến n) cần được lần lượt gia công trên 3 máy, đầu tiên qua máy A, sau đó đến máy B, cuối cùng mới đến máy C. Giả thiết biết thời gian gia công chi tiết i tương ứng trên các máy A, B, C là a_i, b_i, c_i (i = 1, 2, ..., n). Hãy tìm một lịch gia công (thứ tự gia công) để thời điểm hoàn thành tất cả các chi tiết là sớm nhất".

Điều đáng chú ý là, mặc dù việc mở rộng bài toán là đơn giản và tự nhiên, nhưng hiện nay chưa có một thuật toán hữu hiệu nào kiểu như thuật toán Johnson cho trường hợp 3 máy hoặc nhiều hơn. Trong trường hợp tổng quát, người ta vẫn phải dùng giải pháp duyệt toàn bộ có đánh giá nhánh cận.

Tuy nhiên, trong trường hợp 3 máy, một số trường hợp riêng có thể dẫn về bài toán 2 máy như định lý dưới đây (không chứng minh):

Định lý. Nếu bài toán 3 máy A, B, C thỏa mãn điều kiện

$$\max \{b_i\} \le \min \{a_i\} \text{ hoặc } \max \{b_i\} \le \min \{c_i\}$$

(nghĩa là thời gian gia công của máy B khá nhỏ so với thời gian gia công của máy A hoặc máy C) thì lịch tối ưu của nó trùng với lịch tối ưu của bài toán 2 máy A', B' với các thời gian tương ứng $a'_i = a_i + b_i$ và $b'_i = b_i + c_i$ (chú ý rằng chỉ có lịch tối ưu của chúng là trùng nhau còn thời điểm hoàn thành của chúng là khác nhau).

Ví dụ: Xét bài toán 3 máy A, B, C:

chi tiết	1	2	3	4	5
máy A	7	11	8	7	6
máy B	6	5	3	5	3
máy C	4	12	7	8	3

ta có $\max \{b_i\} = 6$, $\min \{a_i\} = 6$. Bài toán đã cho thỏa mãn điều kiện đưa về 2 máy A', B' với bảng thời gian:

chi tiết	1	2	3	4	5
máy A'	13	16	11	12	9
máy B'	10	17	10	13	6

Giải bài toán 2 máy này theo thuật toán Johnson ta được lịch tối ưu $X^* = (4, 2, 3, 1, 5)$. Thời điểm hoàn thành của lịch này được tính theo bảng thời gian 3 máy bằng các công thức truy hồi:

(1) $T_A(i) = T_A(i-1) + a_i$, i = 1, 2, ..., n với giá trị ban đầu $T_A(0) = 0$

(2) $T_B(i) = \max\{T_B(i-1), T_A(i)\} + b_i$, i = 1, 2, ..., n với giá trị ban đầu $T_B(0) = 0$

(3) $T_C(i) = \max\{T_C(i-1), T_B(i)\} + c_i$, i = 1, 2, ..., n với giá trị ban đầu $T_C(0) = 0$

lịch X^*	4	2	3	1	5
$T_A(i)$	7	18	26	33	39
$T_B(i)$	12	23	29	39	42
$T_C(i)$	20	35	42	46	49

và nhận được kết quả $T(X^*) = T_C(5) = 49$.

Qua việc mở rộng vừa trình bày, ta cũng nhận thấy một điều là, trong việc giải các bài toán tối ưu tổ hợp, thuật toán càng hiệu quả thì tính riêng biệt của nó càng lớn.

Trong thực tế, lịch gia công còn phải thỏa mãn thêm nhiều điều kiện khác. Vì những ứng dụng quan trọng của chúng mà trong lý thuyết tối ưu đã hình thành một lĩnh vực riêng, chuyên nghiên cứu về các bài toán lập lịch, được gọi là *lý thuyết lập lịch* hay *quy hoạch lịch*.

TÓM LƯỢC CUỐI BÀI

Qua bài học, các bạn đã nắm được những nét chính của bài toán tối ưu tổ hợp, hiểu và ứng dụng được ý nghĩa của bài toán tối ưu trong mô hình thực tế.

Các bạn cần ghi nhớ các vấn đề sau :

- Các yêu cầu của bài toán tối ưu tổ hợp
- Bài toàn người du lịch và bài toán cái túi
- Phương pháp duyệt toàn bộ
- Kỹ thuật đánh giá nhánh cận
- Phương pháp tham lam
- Bài toán tìm lịch gia công trên hai máy và thuật toán Johnson

BÀI TẬP

Các bài tập trong bài này đều là các bài tập lập trình. Bạn đọc có thể viết trên bất cứ ngôn ngữ nào mà bạn thích. Mặc dù trong bài giảng các đoạn chương trình đều viết phỏng theo ngôn ngữ Pascal (là ngôn ngữ gần gũi với việc trình bày thuật toán), bạn đọc có thể dễ dàng chuyển nó sang ngôn ngữ được dùng miễn là hiểu được nội dung cách giải.

Các bài 1, 2, 3 là những bài tập thực hiện lời giải đã trình bày trong bài giảng.

1. Viết chương trình giải bài toán người du lịch. Dữ liệu nhập từ bàn phím, kết quả đưa ra màn hình. Yêu cầu giải bằng các phương pháp khác nhau:
   - Phương pháp tham lam
   - Duyệt toàn bộ không đánh giá nhánh cận
   - Duyệt toàn bộ có đánh giá nhánh cận

   và so sánh kết quả của các phương pháp này.
2. Viết chương trình giải bài toán cái túi với những yêu cầu giống bài toán người du lịch.
3. Viết chương trình giải bài toán gia công chi tiết trên 2 máy bằng hai cách:
   - Thuật toán Johnson
   - Duyệt toàn bộ có đánh giá nhánh cận

   và so sánh kết quả của các phương pháp này.

Các bài tập tiếp theo lấy từ các mô hình trên thực tế. Tất cả đều được giải bằng cách duyệt toàn bộ có đánh giá nhánh cận. Bạn đọc phải xây dựng mô hình toán học, đưa ra giải pháp liệt kê và đánh giá nhánh cận sao cho càng hiệu quả càng tốt. Để dễ dàng kiểm chứng chương trình, dữ liệu vào cũng như kết quả ra đều được ghi trên những file văn bản với khuôn dạng như yêu cầu trong đề bài, trong đó các dữ liệu số ghi trên cùng dòng phải cách nhau ít nhất một dấu trắng để phân biệt. Đây thực sự là những bài tập lớn, đòi hỏi phải có thời gian làm trên máy và kiểm tra tính đúng đắn của chương trình.

4. **Bài toán phân công.** Có n người và n công việc (được đánh số từ 1 đến n). Biết chi phí để thợ i hoàn thành công việc j là c(i, j). Tìm phương án phân công mỗi người mỗi việc để tất cả công việc đều được hoàn thành với tổng chi phí là ít nhất.

   File dữ liệu vào cho trong file có tên là B4.INP với khuôn dạng:

   n

   c(1, 1) c(1, 2) ... c(1, n)

   c(2, 1) c(2, 2) ... c(2, n)

   .....

   c(n, 1) c(n, 2) ... c(n, n)

   File kết quả ra ghi lên file có tên là B4.OUT với khuôn dạng:

   x(1) x(2) ... x(n)

   trong đó x(i) là công việc phân cho người i. Giả thiết các giá trị c(i, j) là nguyên dương không quá 100 và n không quá 20.
5. **Bài toán cho thuê máy.** Một ông chủ có một cái máy ủi đất để cho thuê, thời gian thuê tính theo đơn vị ngày trong tháng. Đầu tháng, ông ta nhận được yêu cầu thuê máy của m khách hàng (các khách được đánh số từ 1 đến m). Mỗi khách hàng sẽ cho biết danh sách các ngày cần thuê trong tháng (các ngày trong tháng được đánh số từ 1 đến 30). Giả thiết rằng ông chủ hoặc từ chối khách hoặc thỏa mãn đúng các yêu cầu của khách. Hãy tìm một phương án chọn khách cho thuê của ông chủ để tổng số ngày sử dụng máy là nhiều nhất.

   File dữ liệu vào cho trong file có tên là B5.INP với khuôn dạng:

m
k(1, 1) k(1, 2) ... k(1, 30)
k(2, 1) k(2, 2) ... k(2, 30)
.....
k(m, 1) k(m, 2) ... k(m, 30)
trong đó k(i, j) = 1 nếu khách i thuê ngày j và k(i, j) = 0 nếu trái lại.
File kết quả ra ghi lên file có tên là B5.OUT với khuôn dạng:
n
x(1) x(2) ... x(n)
trong đó n là số khách được chấp nhận và các x(i), i = 1, 2, ..., n là các số hiệu của những khách này.
Giới hạn m (số khách) không quá 20.

6. **Bài toán đóng thùng.** Có n đồ vật (đánh số từ 1 đến n) có trọng lượng là p(1), p(2), ..., p(n). Có một số cái thùng có cùng dung lượng là w, w ≥ p(i), i = 1, 2, ..., n. Các thùng lần lượt được ghi số hiệu 1, 2, ... Tìm phương án đóng thùng xếp tất cả các đồ vật đã cho với số lượng thùng ít nhất.
File dữ liệu vào cho trong file có tên là B6.INP với khuôn dạng:
n
p(1) p(2) ... p(n)
w
File kết quả ra ghi lên file có tên là B6.OUT với khuôn dạng:
k
x(1) x(2) ... x(n)
trong đó k là số thùng cần dùng, x(i) là số hiệu thùng mà đồ vật i được xếp vào. Giả thiết các giá trị p(i) là nguyên dương không quá 100 và n không quá 20.

7. **Bài toán phân phối.** Có hai máy tính A và B đồng thời cùng hoạt động. Có n chương trình (đánh số từ 1 đến n) có thể chạy ở máy nào cũng được. Do các đặc thù riêng của từng máy cũng như của từng chương trình nên thời gian chạy các chương trình trên các máy là khác nhau. Giả sử biết thời gian chạy chương trình i trên các máy A, B tương ứng là a(i) và b(i), i = 1, 2, ..., n. Hãy tìm cách phân phối các chương trình này vào các máy để thời điểm hoàn thành tất cả các chương trình là sớm nhất (xem thời điểm các máy bắt đầu làm việc là bằng 0)...
File dữ liệu vào cho trong file có tên là B7.INP với khuôn dạng:
n
a(1) a(2) ... a(n)
b(1) b(2) ... b(n)
File kết quả ra ghi lên file có tên là B7.OUT với khuôn dạng:
x(1) x(2) ... x (n)
trong đó x(i) là một trong hai ký tự A hoặc B, với quy ước nếu nó là A thì chương trình i phân cho máy A còn nếu nó là B thì chương trình i phân cho máy B. Các ký tự này được viết liền nhau trên một dòng. Giả thiết các a(i), b(i) là các số nguyên dương không quá 100 và n không quá 20.

CÂU HỎI THƯỜNG GẶP

1. Nhiệm vụ chính của bài toán tối ưu?
2. Ý nghĩa của bài toán tối ưu tổ hợp là gì?