Chương 1. Tham lam

1.1. Giải thuật tham lam

Tham lam (*greedy*) là một phương pháp giải các bài toán tối ưu. Các thuật toán tham lam dựa vào sự đánh giá *tối ưu cục bộ* (*local optimum*) để đưa ra *quyết định tức thì* tại mỗi bước lựa chọn, dãy quyết định này cuối cùng sẽ tìm ra được phương án *tối ưu tổng thể* (*global optimum*). Với một bài toán có nhiều thuật toán để giải quyết, thông thường thuật toán tham lam có tốc độ tốt hơn hẳn so với các thuật toán tối ưu tổng thể.

Có những lầm tưởng cho rằng thuật toán tham lam có thể cho phương án tối ưu, có thể không. Ở đây ta cần phân biệt giữa hai khái niệm "thuật toán" và "chiến lược". Khi đề cập đến thuật toán tham lam, chắc chắn nó phải đúng, tức là phải đạt yêu cầu đề ra của bài toán (bởi tính đúng đắn là một trong những đặc trưng của thuật toán). Chiến lược tham lam là cách thức mà mọi thuật toán tham lam sử dụng để tìm ra phương án tối ưu, tuy nhiên chiến lược này đôi khi cũng được dùng để xây dựng phương án xấp xỉ (*appriximation*) với phương án tối ưu. Phương án xấp xỉ có thể được chấp nhận trong các vấn đề thực tế, hoặc được tích hợp vào một thuật toán khác (chẳng hạn dùng làm hàm cận của thuật toán nhánh cận) để tìm phương án tối ưu.

Trong bài này ta sẽ chỉ phân tích những bài toán mà trong đó thuật toán tham lam cho lời giải tối ưu. Khác với các kỹ thuật thiết kế thuật toán như chia để trị, liệt kê, quy hoạch động, rất khó để đưa ra một quy trình chung để tiếp cận bài toán và tìm thuật toán tham lam, chỉ có thể đưa ra hai kinh nghiệm:

- Thử tìm một thuật toán tối ưu tổng thể (ví dụ như quy hoạch động), sau đó đánh giá lại thuật toán, nhìn lại mỗi bước xử lý và đặt câu hỏi "Liệu tại bước này, việc đưa ra quyết định có thể đơn giản hóa đi không?".
- Hoặc nghĩ xem nếu như không có máy tính, không có khái niệm gì về các chiến lược tối ưu tổng thể thì ở góc độ con người, chúng ta sẽ đưa ra giải pháp như thế nào?

Xét một ví dụ: Trong máy rút tiền tự động (ATM) có n loại tiền giấy đánh số từ 0 tới $n-1$ với mệnh giá các tờ tiền là các số dương $a_0, a_1, \dots, a_{n-1}$ (đồng). Số lượng mỗi loại tiền coi như không hạn chế. Một người đến máy ATM và ra lệnh rút x (đồng). Hãy cho biết máy phải đưa ra ít nhất bao nhiêu tờ tiền để được số tiền đúng bằng x.

Bài toán này thể giải theo nhiều cách…

Nếu một máy ATM sử dụng thuật toán quay lui, có thể coi một cách trả tiền là một dãy $(k_0, k_1, \dots, k_{n-1})$ thỏa mãn:

$$a_0 \times k_0 + a_1 \times k_1 + \cdots + a_{n-1} \times k_{n-1} = x$$

(trong đó $k_i \geq 0$ là số tờ tiền loại i trong cách trả tiền tương ứng)

Máy sẽ duyệt mọi khả năng có thể của k_0 (từ 0 tới $\lfloor x/a_0 \rfloor$), với mỗi giá trị thử cho k_0 lại duyệt mọi khả năng có thể của k_1,… Mỗi khi ta có một cấu hình $(k_0, k_1, \dots, k_{n-1})$ tương ứng với một cách trả tiền, phương án tối ưu sẽ được cập nhật nhằm tìm ra cấu hình có $k_0 + k_1 + \cdots + k_{n-1}$ nhỏ nhất. Vấn đề với máy ATM loại này là khi số loại tiền n và số tiền cần đổi x là những số tương đối lớn, sự bùng nổ tổ hợp làm máy sẽ không thể duyệt hết mọi cấu hình trong thời gian cho phép và khách hàng sẽ thấy là máy … treo, ngay cả khi đã áp dụng các kỹ thuật nhánh cận.

Một máy ATM sử dụng chiến lược chia để trị sẽ nghĩ theo cách khác, để trả x đồng thì nếu $x = 0$ ta không cần trả tờ tiền nào cả. Nếu $x > 0$ thì máy sẽ phải trả ít nhất một tờ tiền, nếu phương án tối ưu có trả một tờ tiền mệnh giá a_i thì bài toán quy về đổi $x' = x - a_i$ (đồng). Thuật toán này dễ dàng cài đặt bằng một hàm đệ quy tính $f(x)$ là số tờ tiền ít nhất để trả x đồng:

$$f(x) = \begin{cases} 0, \text{nếu } x = 0 \\ 1 + \min\limits_{\forall i: a_i \le x} \{f(x - a_i)\} \end{cases} \quad (1.1)$$

Thuật toán này vẫn gặp phải vấn đề bùng nổ tổ hợp trong trường hợp n và x là những số tương đối lớn.

Nếu như x và các a_i là những số nguyên không âm, một máy ATM sử dụng chiến lược quy hoạch động có thể dựa vào công thức truy hồi (1.1) để triển khai tính toán trên một mảng $f[0 \dots x]$: trước hết gán $f[0] = 0$ làm cơ sở quy hoạch động, rồi dùng công thức truy hồi (1.1) tính lần lượt $f[1], f[2], \dots, f[x]$. Sau đó truy vết để xác định phương án trả tiền tối ưu. Máy ATM này có thời gian thực hiện giải thuật $O(n \times x)$ và chi phí bộ nhớ $O(x)$, nó có thể hoạt động tốt nếu đủ bộ nhớ và có một bộ xử lý nhanh.

Giả sử ta có các loại tiền tương tự hệ thống tiền tệ của Việt Nam: 1, 2, 5, 10, 20, 50, 100, 200 và 500 (nghìn đồng). Bây giờ để trả một số tiền 900 (nghìn đồng), hãy thử nghĩ theo một cách rất "con người" như các bà nội trợ: họ sẽ trả bằng một tờ 500 nghìn đồng và hai tờ 200 nghìn đồng dù không có khái niệm gì về chiến lược chia để trị, quy hoạch động, vét cạn,… Cách làm của họ rất đơn giản, mỗi khi rút một tờ tiền ra trả, họ *quyết định tức thời* bằng cách lấy ngay tờ tiền mệnh giá cao nhất không vượt quá giá trị cần trả, mà không cần để ý đến "hậu quả" của sự quyết định đó, các quyết định sẽ được đưa ra liên tiếp trong một vòng lặp cho tới khi số tiền phải trả bằng 0. Về bản chất, đây chính là chiến lược tham lam.

Trên thực tế tôi chưa biết hệ thống tiền tệ nào trên thế giới khiến cho cách làm này sai. Tuy nhiên có thể chỉ ra những hệ thống tiền tệ giả định mà cách này cho giải pháp không tối ưu. Chẳng hạn có 3 loại tiền: 1 đồng, 5 đồng và 8 đồng. Nếu cần đổi 10 đồng thì phương án tối ưu phải là dùng hai tờ 5 đồng, nhưng cách làm này sẽ cho phương án một tờ 8 đồng và hai tờ 1 đồng. Hậu quả của phép chọn tờ tiền 8 đồng đã làm cho lời giải không tối ưu.

Có hai kinh nghiệm khi tiếp cận bài toán và tìm thuật toán tham lam:

- *Khảo sát kỹ bài toán* để tìm ra các tính chất đặc biệt mà ở đó ta có thể đưa ra quyết định tức thời tại từng bước dựa vào sự đánh giá tối ưu cục bộ.
- Cần phải chứng minh được tính đúng đắn của thuật toán, tức là chứng minh được các quyết định dựa trên đánh giá tối ưu cục bộ sẽ đưa đến dãy quyết định tối ưu tổng thể. Một thuật toán sai sẽ dẫn đến cả một dãy các quyết định sai và dường như không có cơ may nào cho một dãy quyết định sai lại đưa ra được kết quả đúng. Đọc một lời giải tham lam sẽ có cảm giác nó rất dễ, nhưng kinh nghiệm cho thấy rằng với một bài toán có nhiều thuật giải thì công sức xây dựng một thuật toán tham lam lớn hơn nhiều so với các thuật toán khác.

1.2. Một số bài toán sử dụng giải thuật tham lam

Trong phần này ta sẽ khảo sát một số bài toán kinh điển mà lời giải bằng thuật toán tham lam được coi là mẫu mực, nhằm rút ra những kinh nghiệm trong thiết kế thuật toán tham lam.

Với những bài toán mang bản chất đệ quy, quy trình sau có thể dùng để tìm thuật toán tham lam (nếu có):

- Phân rã bài toán lớn ra thành các bài toán con đồng dạng mà nghiệm của các bài toán con có thể dùng để chỉ ra nghiệm của bài toán lớn. Bước này giống như thuật toán chia để trị và chúng ta có thể thiết kế sơ bộ một giải thuật chia để trị.
- Chỉ ra rằng không cần giải toàn bộ các bài toán con mà chỉ cần giải một bài toán con thôi là có thể chỉ ra nghiệm của bài toán lớn. Điều này cực kỳ quan trọng, nó chỉ ra rằng sự lựa chọn tham lam tức thời (tối ưu cục bộ) tại mỗi bước sẽ dẫn đến phương án tối ưu tổng thể.
- Phân tích dãy quyết định tham lam để sửa giải thuật chia để trị thành một giải thuật lặp.

Không phải bài toán nào mang bản chất đệ quy cũng có thể giải quyết bằng thuật toán tham lam, kể cả có thể giải quyết được bằng thuật toán tham lam thì cách tiếp cận tìm ra thuật toán đó cũng rất đa dạng. Trên đây chỉ là một quy trình phổ biến để tìm thuật toán tham lam mà thôi, ta xét một vài ví dụ cho dạng bài toán này…

1.2.1. Chọn hoạt động

Mô hình bài toán lý thuyết có thể phát biểu ngắn gọn: Trên trục số cho n khoảng, hãy chọn ra tối đa những khoảng hoàn toàn rời nhau, tức là hai khoảng bất kỳ được chọn giao nhau bằng $\emptyset$.

Một vấn đề thực tế áp dụng mô hình lý thuyết này là bài toán chọn hoạt động (*activity selection*) hay chọn tham gia sự kiện (*event selection*): Một người có dự định cho rất nhiều sự kiện và muốn tham gia vào một số tối đa những sự kiện đó. Giả sử có n sự kiện đánh số từ 0 tới $n-1$. Sự kiện i diễn ra trong suốt khoảng thời gian $(a_i, b_i]$ và nếu người này tham gia vào một sự kiện, thì người đó cần tham gia liên tục từ khi bắt đầu tới khi kết thúc sự kiện. Hãy giúp người đó chọn ra nhiều nhất các sự kiện để tham gia, sao cho tại mỗi thời điểm bất kỳ, người đó không tham gia quá một sự kiện, hay nói cách khác, khoảng thời gian tham gia hai sự kiện bất kỳ đã chọn là không giao nhau.

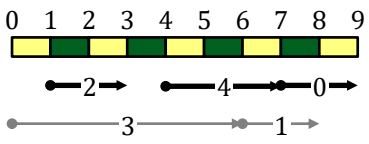
Input

- Dòng 1 chứa số nguyên dương $n \leq 10^5$
- n dòng tiếp theo, mỗi dòng chứa hai số nguyên a_i, b_i $(0 \leq a_i < b_i \leq 10^9)$ ứng với khoảng thời gian diễn ra sự kiện i.

Output

Phương án chọn nhiều sự kiện nhất để tham gia

Sample Input	Sample Output
5 7 9 6 8 1 3 0 6 4 7	Event 2: (1, 3] Event 4: (4, 7] Event 0: (7, 9] Number of selected events: 3



❋ Chia để trị

Thuật toán chia để trị: Đặt hàm mục tiêu tìm $f(t)$ là tập sự kiện nhiều nhất có thể tham gia bắt đầu từ thời điểm t. Ta quan tâm tới $f(0)$ là phương án tối ưu của bài toán. Để tính $f(t)$, ta chỉ quan tâm tới các sự kiện bắt đầu từ thời điểm t trở đi (những sự kiện có $a[.] \geq t$).

Nếu không tồn tại sự kiện có $a[.] \geq t$, ta có $f(t) = \emptyset$, nếu không thì $f(t)$ có chứa ít nhất một sự kiện. Nếu phương án tối ưu ứng với $f(t)$ có chọn p là sự kiện đầu tiên để tham gia thì bài toán quy về tìm tập sự kiện nhiều nhất bắt đầu từ thời điểm sự kiện p kết thúc: $f(b_p)$. Vậy thì $f(t) = \{p\} \cup f(b_p)$. Thông qua việc thử tất cả các lựa chọn trên sự kiện đầu tiên p, phương án tối ưu $f(t)$ được xác định.

❋ **Phép chọn tham lam**

Thuật toán chia để trị phân rã bài toán lớn thành các bài toán con dựa trên phép chọn sự kiện đầu tiên để tham gia, ta có một quan sát làm cơ sở cho phép chọn tham lam:

Bổ đề 1-1

Xét bài toán chọn các sự kiện bắt đầu từ thời điểm 0, khi đó trong số các phương án tối ưu, chắc chắn có một phương án mà sự kiện được chọn tham gia đầu tiên là sự kiện kết thúc sớm nhất trong số n sự kiện.

Chứng minh

Gọi p_0 là sự kiện kết thúc sớm nhất. Với một phương án tối ưu bất kỳ, giả sử thứ tự tham gia các sự kiện trong phương án tối ưu đó là $(q_0, q_1, \dots, q_{k-1})$. Do p_0 là sự kiện kết thúc sớm nhất nên chắc chắn nó không thể kết thúc muộn hơn q_0. Vì vậy, việc thay q_0 bởi p_0 trong phương án này sẽ không gây ra sự xung đột nào về thời gian tham gia các sự kiện. Sự thay thế này cũng không làm giảm bớt số lượng sự kiện thực hiện được trong phương án tối ưu, nên $(\boxed{p_0}, q_1, q_2, \dots, q_k)$ cũng sẽ là một phương án tối ưu.

Yêu cầu của bài toán là chỉ cần đưa ra một phương án tối ưu, nên ta sẽ tìm phương án tối ưu có sự kiện tham gia đầu tiên là sự kiện kết thúc sớm nhất trong số n sự kiện. Điều này cho phép ta không cần thử n khả năng chọn sự kiện để tham gia đầu tiên, đi giải các bài toán con, rồi mới đánh giá chúng để đưa ra quyết định cuối cùng. Thuật toán sẽ đưa ra *quyết định tức thời*: chọn ngay sự kiện kết thúc sớm nhất p_0 để tham gia đầu tiên.

Sau khi chọn sự kiện p_0, bài toán lớn quy về bài toán con: Chọn nhiều sự kiện nhất trong số các sự kiện bắt đầu sau khi p_0 kết thúc. Phép chọn tham lam lại cho ta một quyết định tức thời: Sự kiện tiếp theo trong phương án tối ưu sẽ là sự kiện bắt đầu sau thời điểm $b[p_0]$ và có thời điểm kết thúc sớm nhất, gọi đó là sự kiện p_1. Và cứ như vậy chúng ta chọn tiếp các sự kiện $p_2, p_3, \dots$

❋ **Cài đặt giải thuật tham lam**

Tư tưởng của giải thuật tham lam có thể tóm tắt lại:

- Chọn p_0 là sự kiện kết thúc sớm nhất.
- Chọn p_1 là sự kiện kết thúc sớm nhất trong số các sự kiện bắt đầu sau khi p_0 kết thúc: $a[p_1] \geq b[p_0]$.
- Chọn p_2 là sự kiện kết thúc sớm nhất trong số các sự kiện bắt đầu sau khi p_1 kết thúc: $a[p_2] \geq b[p_1]$.
- …
- Cứ như vậy cho tới khi không còn sự kiện nào chọn được nữa.

Đến đây ta có thể thiết kế một giải thuật lặp:

- Sắp xếp các sự kiện theo thứ tự tăng dần của thời điểm kết thúc (`b[.]`)
- Khởi tạo thời điểm `FreeTime = 0`
- Duyệt các sự kiện theo danh sách đã sắp xếp (sự kiện kết thúc sớm hơn sẽ được xét trước sự kiện kết thúc muộn hơn), nếu xét đến sự kiện `p` có thời điểm bắt đầu `a[p] ≥ FreeTime` thì chọn ngay sự kiện `p` vào phương án tối ưu và cập nhật `FreeTime` thành thời điểm kết thúc sự kiện `p`: `FreeTime = b[p]`.

ACTIVITYSELECTION.cpp Chọn tham gia sự kiện

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int maxN = 1e5;
int n, a[maxN], b[maxN], id[maxN];

void Enter() //Nhập dữ liệu
{
    cin >> n;
    for (int i = 0; i < n; ++i)
    {
        cin >> a[i] >> b[i];
        id[i] = i;
    }
    //Sắp xếp bằng chỉ số theo thứ tự tăng dần của [.]:
    //b[id[0]] ≤ bid[1]] ≤ … ≤ b[id[n - 1]]
    sort(id, id + n, [](int i, int j)
    {
        return b[i] < b[j];
    });
}

void Greedy() //Thuật toán tham lam
{
    int m = 0;
    int FreeTime = 0;
    for (int i = 0; i < n; ++i)
    {
        int p = id[i]; //Xét sự kiện p = id[i]
        if (a[p] < FreeTime) continue; //p xung đột với sự kiện chọn trước đó, bỏ qua
        cout << "Event " << p << ": (" << a[p] << ", " << b[p] << "]\n"; //Chọn sự kiện p
        FreeTime = b[p];
        ++m;
    }
    cout << "Number of selected events: " << m;
}

int main()
{
    Enter();
    Greedy();
}
```

Dễ thấy rằng thuật toán tham lam ở hàm Greedy() được thực hiện trong thời gian $\Theta(n)$. Thời gian thực hiện giải thuật chủ yếu nằm ở thuật toán sắp xếp các công việc theo thời điểm kết thúc: $O(n \log n)$.

1.2.2. Phủ

Trên trục số cho n đoạn: $[a_0, b_0], [a_1, b_1], \dots, [a_{n-1}, b_{n-1}]$. Hãy chọn một số ít nhất các đoạn trong số n đoạn đã cho để phủ hết đoạn $[L, R]$.

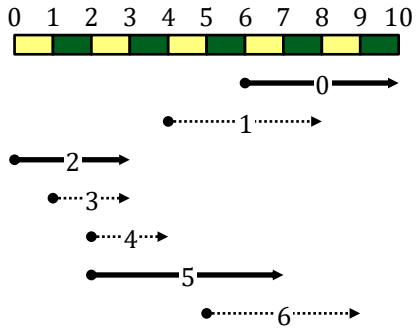
Input

- Dòng 1 chứa ba số nguyên n, L, R ($1 \le n \le 10^5, 0 \le L \le R \le 10^9$)
- n dòng tiếp theo, mỗi dòng chứa hai số nguyên a_i, b_i mô tả một đoạn ($0 \le a_i \le b_i \le 10^9$)

Output

Cách chọn ra ít nhất các đoạn để phủ đoạn $[L, R]$.

Sample Input	Sample Output
7 1 9 6 10 4 8 0 3 1 3 2 4 2 7 5 9	Selected Intervals: 2: [0, 3] 5: [2, 7] 0: [6, 10]



Để tiện cho việc trình bày, với một đoạn $[a_i, b_i]$, ta gọi a_i là cận dưới và b_i là cận trên của đoạn. Ngoài ra ta giả thiết rằng dữ liệu vào được cho để luôn tồn tại phương án phủ, trường hợp không tồn tại phương án phủ sẽ được lồng ghép vào trong quá trình cài đặt, sau khi đã phân tích thuật toán.

❋ Chia để trị

Nhận xét rằng phương án tối ưu để phủ $[L, R]$ chắc chắn phải chọn một đoạn nào đó để phủ điểm L. Giả sử đoạn $[a^*, b^*]$ chứa điểm L được chọn vào phương án tối ưu, nếu đoạn này phủ tới cả điểm R, ta có lời giải, còn nếu không, bài toán quy về phủ đoạn $[b^*, R]$ bằng ít đoạn nhất trong số các đoạn còn lại.

❋ Phép chọn tham lam

Thuật toán chia để trị quy việc giải bài toán với n đoạn về việc giải bài toán con với $n - 1$ đoạn dựa vào sự lựa chọn đoạn đầu tiên để phủ điểm L. Ta có một nhận xét làm cơ sở cho phép chọn tham lam:

Bổ đề 1-2

Trong các đoạn phủ được điểm L, đoạn có cận trên lớn nhất chắc chắn được chọn vào một phương án tối ưu nào đó.

Chứng minh

Trong các đoạn phủ được điểm L, gọi đoạn có cận trên lớn nhất là $[a^*, b^*]$. Với một phương án tối ưu bất kỳ mà không chọn đoạn $[a^*, b^*]$, thì trong phương án này chắc chắn phải có một đoạn phủ điểm L, gọi đoạn đó là $[a, b]$. Cả hai đoạn $[a, b]$ và $[a^*, b^*]$ đều phủ điểm L, nhưng đoạn $[a^*, b^*]$ sẽ phủ về bên phải điểm L nhiều hơn hoặc bằng so với đoạn $[a, b]$. Vậy nếu trong phương án tối ưu này ta thay thế đoạn $[a, b]$ bởi đoạn $[a^*, b^*]$ thì $[L, R]$ vẫn được phủ và số đoạn được chọn trong phương án vẫn không đổi. Suy ra ĐPCM.

Bổ đề 1-2 cho thấy: thay vì phải thử tất cả các khả năng chọn đoạn phủ điểm L rồi dẫn về bài toán con, ta có thể đưa ra quyết định tức thời: Trong số những đoạn phủ được điểm L, chọn ngay đoạn $[a^*, b^*]$ có cận trên b^* lớn nhất vào phương án tối ưu. Bài toán quy về việc phủ hết $[b^*, R]$ bằng các đoạn còn lại.

❋ Cài đặt giải thuật tham lam

Những phân tích trên cho phép ta thiết kế một giải thuật lặp: Sắp xếp các đoạn đã cho theo thứ tự tăng dần của cận dưới: $a_0 \le a_1 \le \cdots \le a_{n-1}$. Xét bài toán phủ $[L, R]$ bằng các đoạn tính từ chỉ số 0 trở đi...

Trước hết ta quét phần đầu dãy đoạn đã sắp xếp gồm những đoạn $[a_i, b_i]$ mà $a_i \le L$ để chọn ra đoạn $[a^*, b^*]$ có cận trên b^* lớn nhất. Nếu $[a^*, b^*]$ không tồn tại hoặc $[a^*, b^*]$ không phủ tới L thì thuật toán dừng và kết luận không thể phủ được $[L, R]$; nếu không, ta chọn luôn đoạn $[a^*, b^*]$ vào phương án tối ưu.

Nếu $b^* < R$, vấn đề lặp lại với việc phủ $[b^*, R]$. Để ý rằng những đoạn $[a_i, b_i]$ có $a_i \le L$ vừa quét qua ở bước trước có thể bỏ đi vì những đoạn này không phủ thêm về bên phải điểm b^* (điều này suy ra từ tính chất của đoạn $[a^*, b^*]$). Tức là thuật toán lặp lại với việc phủ $[b^*, R]$ bằng các đoạn tính từ chỉ số i chưa quét qua ở bước trước.

INTERVALCOVER.cpp Phủ

```cpp
#include <iostream>
#include <vector>
#include <algorithm>
using namespace std;
const int maxN = 1e5;

int n, L, R, a[maxN], b[maxN], id[maxN];
vector<int> res; //vector chứa chỉ số những đoạn được chọn

void Enter() //Nhập dữ liệu
{
    cin >> n >> L >> R;
    for (int i = 0; i < n; ++i)
    {
        cin >> a[i] >> b[i];
        id[i] = i;
    }
    //Sắp xếp bằng chỉ số a[id[0]] ≤ a[id[1]] ≤ … ≤ a[id[n - 1]]
    sort(id, id + n, [](int i, int j)
    {
        return a[i] < a[j];
    });
}

void Greedy() //Thuật toán tham lam
{
    int i = 0;
    do //Phủ [L, R] bằng các đoạn chỉ số id[i … n)
    {
        //Tìm đoạn j có a[j] ≤ L và b[j] lớn nhất
        int j = -1;
        for (; i < n && a[id[i]] <= L; ++i)
            if (j == -1 || b[id[i]] > b[j])
                j = id[i];
        if (j == -1 || b[j] < L) //Không tồn tại đoạn phủ qua L
        {
            res.clear(); //Xóa vector kết quả
            break; //Ngừng thuật toán
        }
        res.push_back(j); //Chọn đoạn j vào phương án
        L = b[j]; //Quy về phủ [b[j], R] bằng các đoạn id[i], id[i + 1], …
    }
    while (L < R);
}
```

```
45
46  void Print()
47  {
48      if (res.empty()) cout << "No Solution!"; // không tồn tại phương án phủ
49      else //In ra các đoạn được chọn trong phương án phủ
50      {
51          cout << "Selected Intervals:\n";
52          for (int j: res)
53              cout << j << ": [" << a[j] << ", " << b[j] << "]\n";
54      }
55  }
56
57  int main()
58  {
59      Enter();
60      Greedy();
61      Print();
62  }
```

Thời gian thực hiện giải thuật chọn tham lam trong hàm $GreedySelection$ là $O(n)$ (đánh giá qua số lần thực hiện phép toán ++i), chi phí thời gian của thuật toán chủ yếu nằm ở thao tác sắp xếp: $O(n \log n)$.

1.2.3. Cưa gỗ

Cho một thanh gỗ độ dài L, người ta muốn cưa thanh gỗ này thành n thanh gỗ với độ dài $a_0, a_1, \dots, a_{n-1}$, $(L \geq \sum_{i=0}^{n-1} a_i)$, phần thừa (nếu có) sẽ được bỏ đi. Mỗi lần ta có thể lấy mỗi thanh gỗ cưa thành hai thanh với tỉ lệ độ dài tùy ý, để cưa một thanh gỗ độ dài k thành hai thanh mất chi phí đúng bằng k. Tìm cách cưa với chi phí ít nhất.

Input

- Dòng 1 chứa hai số nguyên dương $L \leq 10^9; n \leq 10^5$
- Dòng 2 chứa n số nguyên dương $a_0, a_1, \dots, a_{n-1}$ $(\sum_{i=0}^{n-1} a_i \leq L)$

Output

Cách cưa tìm được

Sample Input	Sample Output
27 5 3 4 5 6 7	27 = 11 + 16 16 = 7 + 9 11 = 5 + 6 9 = 4 + 5 5 = 2 + 3 Total Cost: 68

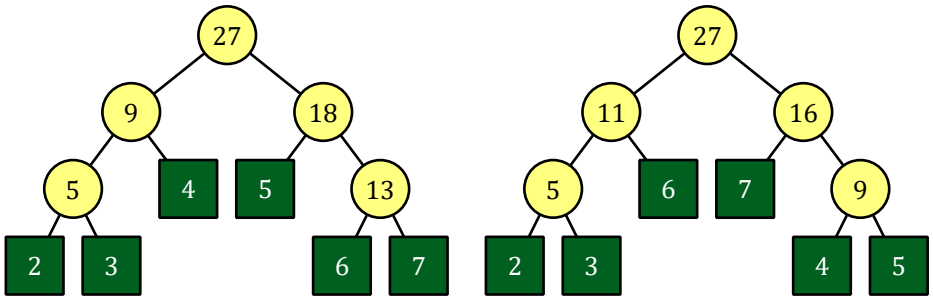
❋ Mô hình cây nhị phân đầy đủ

Có thể coi tổng độ dài n thanh gỗ cần cưa ra đúng bằng L, bởi nếu $\sum_{i=0}^{n-1} a_i < L$, sau khi có thành phẩm sẽ có thêm một thanh gỗ thừa, ta bổ sung luôn thanh gỗ thừa này vào trong tập hợp các thanh gỗ thành phẩm. Như ở ví dụ mẫu, ta cần cưa một thanh gỗ độ dài $L = 27$ thành 6 thanh gỗ độ dài 2, 3, 4, 5, 6, 7.

Vì tại mỗi bước ta cưa một thanh gỗ ra làm hai, rồi lại có thể tiếp tục cưa những thanh gỗ mới tạo ra… Phương án cưa gỗ có thể mô tả bằng một cây nhị phân đầy đủ với mỗi nút chứa độ dài một thanh gỗ. Nút gốc chứa giá trị L là độ dài thanh gỗ ban đầu. Nếu một nút không phải lá thì nó có đủ hai con chứa độ dài hai thanh gỗ cưa ra từ thanh gỗ có độ dài lưu trong nút cha. Như vậy tổng hai giá trị chứa trong hai nút con phải bằng giá trị chứa trong nút cha. Các nút lá chứa giá trị là

độ dài các thanh gỗ thành phẩm, tổng các giá trị trong các nút nhánh là chi phí của phương án cưa gỗ và ta cũng gọi là chi phí trên cây tương ứng.

Hình 1-1 là hai sơ đồ cưa gỗ ứng với input của ví dụ mẫu. Sơ đồ A mất tổng chi phí 72, nhưng sơ đồ B chỉ mất tổng chi phí 68, đây cũng là phương án tối ưu.



A: Không tối ưu, chi phí = 72 B: Tối ưu, chi phí = 68

Hình 1-1. Sơ đồ cưa gỗ

❋ **Chia để trị**

Trước hết ta xét bài toán đối ngẫu của bài toán cưa gỗ: Bài toán nối gỗ. Thay vì tìm cách cắt thanh gỗ độ dài L thành n thanh gỗ độ dài ngắn hơn theo yêu cầu, ta sẽ tìm quy trình ngược lại: Tại mỗi bước ta chọn hai thanh gỗ và nối lại thành một thanh với chi phí bằng độ dài thanh gỗ mới tạo thành. Quá trình kết thúc sau $n-1$ bước và ta còn duy nhất một thanh gỗ độ dài L. Bài toán cưa gỗ và bài toán nối gỗ là tương đương, tìm được phương án tối ưu cho bài toán này thì cũng tìm được phương án tối ưu cho bài toán kia và ngược lại.

Để tìm phương án tối ưu nối n thanh gỗ, ta xét mọi khả năng chọn hai thanh gỗ để nối lại trong bước đầu tiên, khi đó chi phí sẽ được tính bằng tổng độ dài hai thanh gỗ được chọn cộng thêm chi phí nối $n-1$ thanh gỗ còn lại tính cả thanh gỗ mới tạo ra (bài toán con). Chưa cần xét tới độ phức tạp tính toán, mô hình chia để trị này cũng đã khó để cài đặt khi mà tham số xác định bài toán con cũng có kích thước rất lớn (tương ứng với một tập hợp các thanh gỗ). Ta cần thêm các quan sát để có thể chọn ra ngay hai thanh gỗ để nối lại tại mỗi bước mà không cần thử tất cả các cách chọn.

❋ **Phép chọn tham lam**

Bất kỳ cấu trúc cây nhị phân đầy đủ nào thỏa mãn ba điều kiện:

- Nút gốc chứa giá trị L,
- Các nút lá chứa các độ dài thanh gỗ thành phẩm,
- Mỗi nút nhánh chứa giá trị bằng tổng hai giá trị trong hai nút con

đều là sơ đồ một phương án cưa gỗ hợp lệ và ngược lại. Bổ đề 1-3 cho ta nhận diện một cấu trúc cây nhị phân đầy đủ ứng với phương án cưa gỗ tối ưu.

Bổ đề 1-3

Tồn tại phương án cưa gỗ tối ưu mà trên cấu trúc cây nhị phân đầy đủ tương ứng với nó, hai lá chứa giá trị nhỏ nhất sẽ nằm sâu nhất và có cùng một nút cha

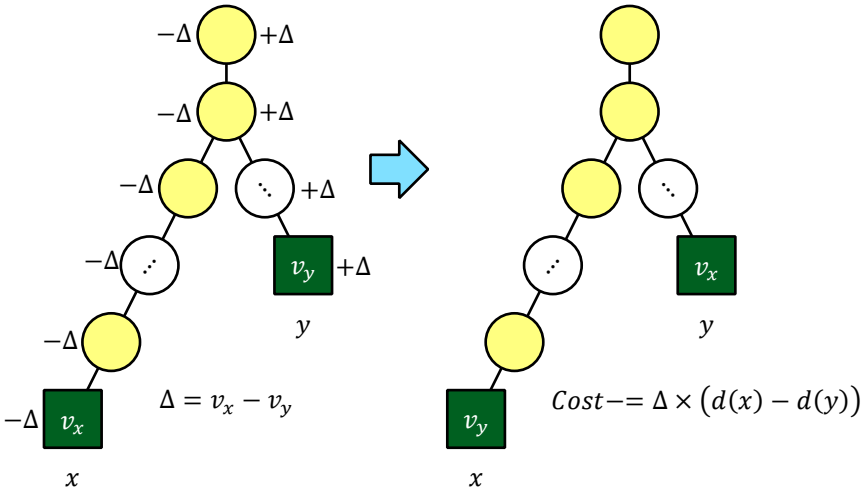
Chứng minh

Với mỗi nút x, ký hiệu $d(x)$ là độ sâu của nút x. Độ sâu của nút x chính là số tiền bối thực sự của nó.

Gọi x là một nút lá sâu nhất và y là một nút lá chứa giá trị nhỏ nhất. Gọi v_x là giá trị chứa trong nút x và v_y là giá trị chứa trong nút y. Ta có $d(x) \geq d(y)$ và $v_x \geq v_y$. Gọi $\Delta = v_x - v_y$, hiển nhiên $\Delta \geq 0$.

Ta sẽ đảo hai giá trị lưu trong lá x và lá y theo cách:

- Dọc trên đường đi từ gốc xuống x, giảm giá trị trong tất cả các nút đi Δ
- Dọc trên đường đi từ gốc xuống y, tăng giá trị trong tất cả các nút lên Δ



Hình 1-2. Đảo giá trị trong hai lá

Hình 1-2 mô tả quá trình thực hiện, rõ ràng trên cây mới, nút gốc vẫn chứa giá trị L, giá trị trong mỗi nút nhánh vẫn bằng tổng hai giá trị trong hai nút con, so với cây cũ thì giá trị trong lá x và lá y đã bị đảo cho nhau, cây mới vẫn là sơ đồ một phương án cưa gỗ hợp lệ.

Khi trừ Δ vào giá trị trong tất cả các nút dọc đường đi từ gốc xuống x, tổng giá trị trong các nút nhánh giảm đi $\Delta \times d(x)$, khi cộng Δ vào giá trị trong tất cả các nút dọc dường đi từ gốc xuống y, tổng giá trị trong các nút nhánh tăng lên $\Delta \times d(y)$. Vậy chi phí trên cây mới giảm đi so với chi phí trên cây cũ một lượng bằng:

$$\Delta \times \big(d(x) - d(y)\big)$$

Nếu $d(x) > d(y)$, cây ban đầu không tối ưu, nếu $d(x) = d(y)$, quy trình trên không làm tăng chi phí.

Ta thành công trong việc đưa giá trị nhỏ nhất về lá sâu nhất x để được một cây mới ít ra là không tệ hơn cây cũ, thực hiện công việc tương tự, ta có thể đưa giá trị nhỏ nhất trong các lá còn lại về nút anh em cùng cha của x. Bổ đề được chứng minh.

Bổ đề 1-3 cho thấy: Tồn tại phương án tối ưu mà hai thanh gỗ thành phẩm ngắn nhất được cưa ra trực tiếp từ cùng một thanh gỗ trung gian. Thuật toán tham lam có thể mô tả rất đơn giản trên bài toán nối gỗ: Bắt đầu với n thanh gỗ thành phẩm, tại mỗi bước, chọn hai thanh gỗ ngắn nhất nối lại thành một thanh với chi phí bằng tổng độ dài hai thanh gỗ được nối, và cứ lặp lại như vậy cho tới khi còn lại duy nhất một thanh gỗ độ dài L.

❋ Cài đặt

Vì tại mỗi bước ta cần chọn ra hai thanh gỗ ngắn nhất để nối lại, tập hợp các thanh gỗ sẽ được quản lý trong một hàng đợi ưu tiên để tiện cho việc lấy ra thanh gỗ ngắn nhất (thanh gỗ nào ngắn hơn được ưu tiên lấy ra hơn)

PARTITION.cpp Bài toán cưa gỗ

```cpp
#include <iostream>
#include <queue>
#include <algorithm>
#include <utility>
using namespace std;

int q, n;
priority_queue<int, deque<int>, greater<int>> PQ; //Ưu tiên số nhỏ

void ReadInput()
{
    cin >> q >> n;
    while(n-- > 0)
    {
        int a;
        cin >> a;
        PQ.push(a); //PQ ban đầu chứa các độ dài thành phẩm
        q -= a;
    }
    if (q > 0) PQ.push(q); //Nếu có đoạn thừa, tính luôn cả đoạn thừa
}

void Solve()
{
    deque<pair<int, int>> Result; //Danh sách các phép cưa trong phương án
    while (PQ.size() > 1)
    {
        int x = PQ.top(); PQ.pop(); //x: Độ dài đoạn ngắn nhất
        int y = PQ.top(); PQ.pop(); //y: Độ dài đoạn ngắn nhì
        Result.push_front(make_pair(x, y)); //Ghi nhận phép nối (cũng là phép cưa)
        PQ.push(x + y); //Đẩy độ dài đoạn sau khi nối vào PQ
    }
    PQ.pop(); //Xóa phần tử duy nhất trong PQ, tránh rác bộ nhớ
    long long Cost = 0LL;
    for (const pair<int, int>& p: Result) //In ra các thao tác đã ghi nhận
    {
        int JoinedLen = p.first + p.second; //Độ dài đoạn cần cưa
        Cost += JoinedLen; //Chi phí thêm độ dài đoạn cần cưa
        cout << JoinedLen << " = " <<
                p.first << " + " << p.second << '\n';
    }
    cout << "Total Cost: " << Cost << '\n';
}

int main()
{
    ReadInput();
    Solve();
}
```

Vì ta cần đưa ra thao tác cưa chứ không phải thao tác nối, danh sách các thao tác phải được in kết quả ngược lại với thứ tự nối gỗ được ghi nhận. Đó là lý do mỗi thao tác được ghi nhận bằng cách đẩy vào đầu danh sách `Result` (dòng 30).

Đối với chương trình này, lệnh `PQ.pop();` ở dòng 36 là thừa, đây chỉ là thói quen tự giác giải phóng hết các ô nhớ tạo ra nếu không còn dùng nữa thay vì ỷ lại vào cơ chế giải phóng tự động. Điều này có ích khi chương trình chạy một lần với nhiều bộ dữ liệu và hàng đợi ưu tiên `PQ` là biến toàn cục. Nếu không làm rỗng `PQ`, lần chạy với bộ dữ liệu sau sẽ có rác từ lần chạy bộ dữ liệu trước.

Dễ dàng chỉ ra được thuật toán thực hiện trong thời gian $O(n \times \log n)$: Vòng lặp `while (PQ.size() > 1)` thực hiện $n-1$ (hoặc n) lần. Trong mỗi lần lặp, lệnh `PQ.pop()` thực hiện hai lần trong thời gian $O(\log n)$ còn các lệnh khác thực hiện trong thời gian $O(1)$.

1.2.4. Mã hóa Huffman

Mã hóa Huffman [13] được sử dụng rộng rãi trong các kỹ thuật nén dữ liệu. Các thử nghiệm với dữ liệu thông thường cho thấy thuật toán mã Huffman có thể nén dữ liệu xuống còn từ 80% tới 10% tùy thuộc vào tình trạng dữ liệu gốc.

❋ Dãy bit biểu diễn dữ liệu

Xét bài toán lưu trữ một xâu ký tự. Thuật toán Huffman dựa vào tần suất xuất hiện của mỗi phần tử để xây dựng cơ chế biểu diễn mỗi ký tự bằng một dãy bit.

Giả sử văn bản là một xâu dài 28 ký tự $\in \{A, B, C, D, E, F\}$. Tần suất (số lần xuất hiện) của mỗi ký tự cho bởi:

Ký tự:	A	B	C	D	E	F
Tần suất:	11	5	5	4	2	1

Một cách thông thường là biểu diễn mỗi ký tự bởi một dãy bit chiều dài cố định (như bảng mã ASCII sử dụng 8 bit cho một ký tự hay bảng mã UCS-2 sử dụng 16 bit cho một ký tự). Chúng ta có 6 ký tự nên có thể biểu diễn mỗi ký tự bằng một dãy 3 bit, hai ký tự khác nhau ứng với hai dãy bit khác nhau:

Ký tự:tần suất	A:11	B:5	C:5	D:4	E:2	F:1
Mã bit	000	001	010	011	100	101

Vì mỗi ký tự chiếm 3 bit nên để biểu diễn xâu ký tự đã cho sẽ cần $28 \times 3 = 84$ bit. Cách biểu diễn này gọi là biểu diễn bằng *từ mã chiều dài cố định* (*fixed-length codewords*).

Một cách khác là biểu diễn mỗi ký tự bởi một dãy bit sao cho ký tự xuất hiện nhiều lần sẽ được biểu diễn bằng dãy bit ngắn trong khi ký tự xuất hiện ít lần hơn sẽ được biểu diễn bằng dãy bit dài:

Ký tự:tần suất	A:11	B:5	C:5	D:4	E:2	F:1
Mã bit	0	111	110	101	1001	1000

Với cách làm này, xâu ký tự đã cho có thể biểu diễn bằng:

$$1 \times 11 + 3 \times 5 + 3 \times 5 + 3 \times 4 + 4 \times 2 + 4 \times 1 = 65 \text{ bit}$$

Dữ liệu được nén xuống còn xấp xỉ 77% so với cách biểu diễn trước. Cách biểu diễn này được gọi là biểu diễn bằng từ mã chiều dài biến động (*variable-length codewords*).

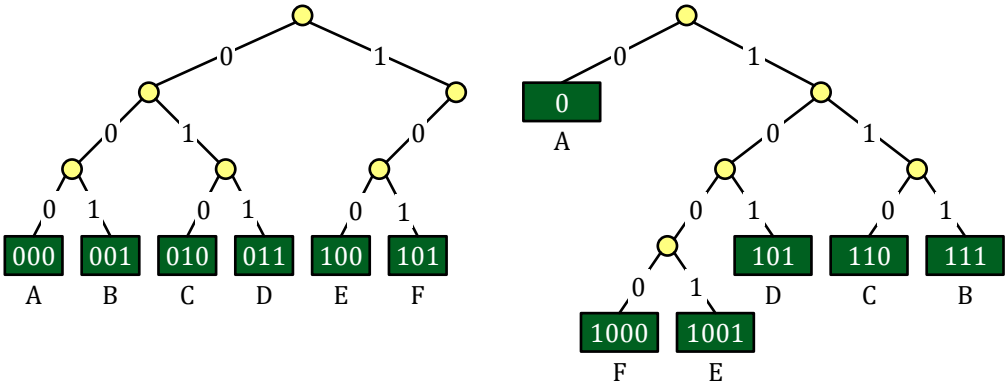
Vấn đề đặt ra với các từ mã chiều dài biến động là làm thế nào trong một dữ liệu biểu diễn bằng dãy bit, xác định mỗi ký tự tương ứng với đoạn trong dãy bit từ đâu tới đâu. Ví dụ nếu mã `A = 0`, `B = 1`, `C = 01`, `D = 11` thì với dữ liệu `0111` nó sẽ tương ứng với các văn bản `ABBB`, `ABD`, `ADB`, `CBB`, `CD`. Một khi dữ liệu không thể giải mã ra được văn bản một cách đơn định thì dữ liệu đó coi như hỏng.

Với cách biểu diễn bằng từ mã chiều dài biến động, không chỉ cần đảm bảo hai từ mã của hai ký tự khác nhau phải khác nhau, mà còn cần đảm bảo sẽ có cách giải mã văn bản một cách duy nhất từ dữ liệu. Không phải phương pháp biểu diễn từ mã chiều dài biến động nào cũng thỏa mãn điều đó.

❋ Mã phi tiền tố

Dãy bit biểu diễn một ký tự gọi là *từ mã* (*codeword*) của ký tự đó. Xét tập các từ mã của các ký tự, nếu không tồn tại một từ mã là tiền tố* của một từ mã khác thì tập từ mã này được gọi là *mã phi tiền tố* (*prefix-free codes* hay còn gọi là *prefix codes*). Hiển nhiên cách biểu diễn bằng từ mã chiều dài cố định là mã phi tiền tố.

Một mã phi tiền tố có thể biểu diễn bằng một cây nhị phân. Trong đó ta gọi nhánh con trái và nhánh con phải của một nút lần lượt là là nhánh 0 và nhánh 1. Các nút lá tương ứng với các ký tự và đường đi từ nút gốc tới nút lá sẽ tương ứng với một dãy nhị phân biểu diễn ký tự đó. Tên của cây nhị phân này là *Trie*†. Hình 1-3 là hai cây nhị phân tương ứng với hai mã tiền tố chiều dài cố định và chiều dài biến động ứng với ví dụ trên.



Hình 1-3. Cây biểu diễn mã phi tiền tố

Một xâu ký tự s (bản gốc) sẽ được mã hóa (nén) theo cách thức: Viết các từ mã của từng chữ cái nối tiếp nhau tạo thành một dãy bit. Ví dụ trong Hình 1-3:

Với cây biểu diễn mã chiều dài cố định thì xâu "AABE" sẽ được mã hóa thành:

$$\underbrace{000}_{A}\,\underbrace{000}_{A}\,\underbrace{001}_{B}\,\underbrace{100}_{E}$$

Với cây biểu diễn mã chiều dài thay đổi thì cũng xâu đó sẽ được mã hóa thành:

$$\underbrace{0}_{A}\,\underbrace{0}_{A}\,\underbrace{111}_{B}\,\underbrace{1001}_{E}$$

Tính chất phi tiền tố đảm bảo rằng nếu xuất phát từ gốc cây với một dữ liệu dạng dãy bit, ta chỉ cần đọc dãy bit từ trái qua phải, mỗi khi đọc một bit ta rẽ sang nhánh con tương ứng thì ta sẽ đứng ở một nút lá khi đọc xong từ mã của một ký tự. Chính vì vậy việc giải mã (giải nén) một dãy bit nhằm khôi phục xâu gốc có thể thực hiện theo thuật toán: Bắt đầu từ nút gốc của cây biểu diễn mã, đọc dãy bit từ trái qua phải, đọc được bit nào rẽ sang nhánh con đó. Mỗi khi đến được nút lá, ta xuất ra ký tự tương ứng và quay trở về nút gốc để đọc tiếp ký tự tiếp theo, cứ như vậy cho tới hết dãy bit.

Nếu ký hiệu Σ là tập các ký tự có mặt trong văn bản, $f(c)$ là số lượng ký tự c trong văn bản, $d(c)$ là độ dài dãy bit biểu diễn ký tự c thì số bit dùng để mã hóa văn bản là: $\sum_{c\in\Sigma} d[c] \times f[c]$

---

* Một xâu ký tự X được gọi là tiền tố của xâu ký tự Y nếu xâu X trùng với phần đầu xâu Y

† Tên gọi xuất phát từ "reTRIEval" dựa trên một công dụng phổ biến của cấu trúc dữ liệu. Từ *trie* này có thể phát âm là /'tri:/ (như "tree"), hoặc /'traɪ/ (như "try")

Kích thước dữ liệu mã hóa phụ thuộc vào cấu trúc cây nhị phân tương ứng với mã phi tiền tố được sử dụng. Bài toán đặt ra là tìm mã phi tiền tố để mã hóa văn bản bằng ít bit nhất.

Dữ liệu vào của bài toán được cho trong khuôn dạng sau:

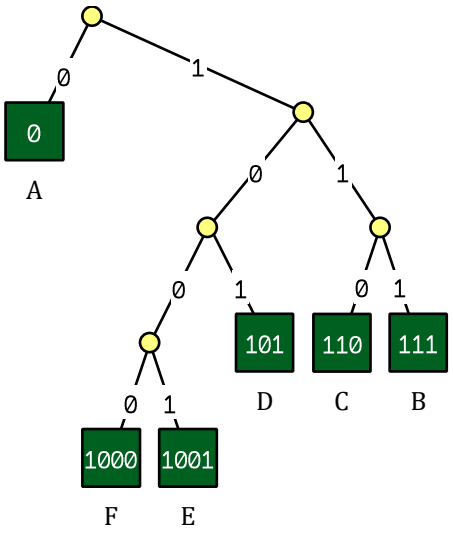
Input

- Dòng 1: Số nguyên dương n là số loại ký tự có trong văn bản ($n \leq 200$)
- n dòng tiếp theo, mỗi dòng ghi một ký tự c và tần suất của nó: $f(c)$ trong văn bản ($f(c) \leq 10^6$)

Output

Các từ mã tương ứng với các ký tự có mặt trong văn bản.

Sample Input	Sample Output
6 A 11 B 5 C 5 D 4 E 2 F 1	A: 0 B: 111 C: 110 D: 101 E: 1001 F: 1000 Encoded size: 65 bits



❋ Thuật toán Huffman

Thuật toán Huffman dùng để tìm trie ứng với mã phi tiền tố tối ưu khi biết tần suất của từng ký tự. Chứng minh tính đúng đắn của thuật toán khá dài dòng, tuy nhiên ta có thể dùng một vài quan sát đơn giản để quy về bài toán cưa gỗ trong mục 1.2.3.

Bổ đề 1-4

Cây biểu diễn mã phi tiền tố tối ưu phải là cây nhị phân đầy đủ. Tức là mọi nút nhánh của nó phải có đúng hai nút con.

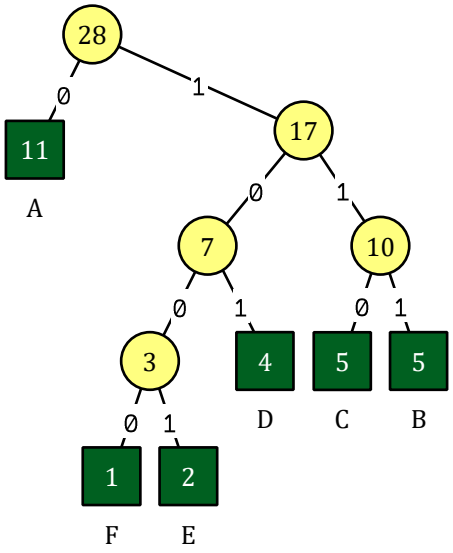
Chứng minh

Thật vậy, nếu một mã phi tiền tố tương ứng với cây nhị phân T không đầy đủ thì sẽ tồn tại một nút nhánh p chỉ có một nút con q. Xóa bỏ nút nhánh p và đưa nhánh con gốc q của nó vào thế chỗ, ta được một cây mới T' mà độ sâu của mọi lá trong nhánh cây gốc q giảm đi 1 còn độ sâu của các lá khác được giữ nguyên. Tức là T' sẽ biểu diễn một mã phi tiền tố với chi phí thấp hơn T. Vậy T không tối ưu.

Bổ đề 1-5

Xét một trie ứng với một mã phi tiền tố, nếu ta lưu tần suất của các ký tự trong các nút lá tương ứng và đặt tần suất trong các nút nhánh sao cho tần suất của một nút nhánh bằng tổng tần suất trong hai nút con. Khi đó độ dài dãy bit để mã hóa toàn văn bản bằng tổng tần suất trong các nút nhánh.

Chứng minh



Hình 1-4. Trie biểu diễn mã phi tiền tố với tần suất trong các nút

Ban đầu đặt tần suất trong tất cả các nút bằng 0. Xét quá trình mã hóa văn bản bằng dãy bit, cứ mỗi lần đọc một ký tự, ta phải viết ra từ mã của nó. Từ mã này đại diện cho một đường đi từ gốc xuống lá của trie. Dọc trên đường đi ta tăng tần suất trong tất cả các nút lên 1 đơn vị.

Tần suất trong mỗi nút x được tăng lên 1 mỗi khi ta thực hiện phép di chuyển từ gốc tới một lá nằm trong nhánh gốc x, vì vậy tần suất trong nút x chính là tổng tần suất các lá trong nhánh gốc x. Vậy thì tần suất trong mỗi nút lá chính là tần suất ký tự tương ứng trong văn bản, còn tần suất trong mỗi nút nhánh bằng tổng tần suất trong hai nút con.

Kết thúc quá trình mã hóa văn bản, tần suất trong mỗi nút nhánh là số lần ta đi ra khỏi nút nhánh đó để xuống nút con. Hơn nữa mỗi lần di chuyển từ nút nhánh xuống nút con, ta viết ra được 1 bit. Vậy thì độ dài dãy bit mã hóa toàn văn bản bằng tổng tần suất trong các nút nhánh.

Đến đây thì vấn đề trở nên giống hệt bài toán cưa gỗ trong mục 1.2.3. Thuật toán Huffman thực hiện theo cách tương tự như vậy, nhưng có lưu lại cả cấu trúc trie để khai thác trong quá trình mã hóa/giải mã:

Quản lý một tập PQ ban đầu gồm n nút, mỗi nút đại diện cho một ký tự và tần suất tương ứng. Lặp lại quá trình sau cho tới khi tập PQ chỉ còn 1 nút:

- Lấy ra từ PQ hai nút x, y có tần suất thấp nhất.
- Tạo ra nút mới z làm cha của cả x và y (thứ tự đặt con trái, con phải không quan trọng), đẩy z vào PQ

Cài đặt

Có thể chứng minh đơn giản bằng quy nạp rằng cây nhị phân đầy đủ có n lá thì có đúng $2n-1$ nút. Riêng đối với trie của thuật toán Huffman, kết quả này có thể suy ra trực tiếp từ thuật toán: Ban đầu PQ chứa n nút ứng với n lá của trie, sau mỗi lượt lặp, số nút trong PQ giảm đi 1. Vòng lặp kết thúc khi PQ còn 1 nút, vì vậy có $n-1$ lượt lặp. Mỗi lượt lặp bổ sung thêm 1 nút nhánh vào cây, vì vậy cây có $n-1$ nút nhánh và n nút lá ban đầu.

Tập PQ chứa các nút trong thuật toán Huffman được cài đặt bằng hàng đợi ưu tiên cho tiện với thao tác lấy ra hai nút có tần suất nhỏ nhất và bổ sung vào một nút mới là cha chung của hai nút vừa lấy ra. Ban đầu n lá được đánh số từ 0 tới $n-1$, các nút nhánh được thêm vào sẽ được đánh

số tiếp từ n tới $2n-2$ (gốc được đánh số $2n-2$). Mỗi nút x có tần suất $f[x]$, nút cha $Parent[x]$ và tên nhánh $Bit[x]$. Ở đây $Bit[x]$ bằng 0 hoặc 1 cho biết x là con trái hay con phải.

HUFFMAN.cpp Mã hóa Huffman

```cpp
#include <iostream>
#include <queue>
#include <numeric>
using namespace std;
const int maxN = 200; //Số ký tự tối đa = số lá tối đa
const int maxTree = 2 * maxN - 1; //Số nút tối đa trên trie

int n, k; //n: Số lá, k: Số nút
char a[maxN]; //Danh sách các ký tự
int f[maxTree], Parent[maxTree], Bit[maxTree]; //Tần suất, nút cha, tên nhánh từng nút

struct TIdxCmp
{
    bool operator()(int i, int j) const //So sánh nút i và nút j
    {
        return f[i] > f[j]; //Ưu tiên nút có tần suất nhỏ hơn
    }
};

priority_queue<int, deque<int>, TIdxCmp> PQ; //Hàng đợi ưu tiên của thuật toán Huffman

void ReadInput() //Nhập dữ liệu
{
    cin >> n;
    for (int i = 0; i < n; ++i)
    {
        cin >> a[i] >> f[i];
        PQ.push(i); //Đẩy vào PQ các lá 0, 1, …, n - 1
    }
}

void BuildHuffmanTrie() //Dựng trie ứng với mã phi tiền tối tối ưu
{
    k = n; //Số nút hiện tại
    while (PQ.size() > 1) //Chừng nào PQ có từ 2 nút trở lên
    {
        int L = PQ.top(); PQ.pop(); //L: Nút có tần suất thấp nhất
        int R = PQ.top(); PQ.pop(); //R: Nút có tần suất thấp nhất trong các nút còn lại
        f[k] = f[L] + f[R]; //Tạo nút k có tần suất bằng tổng tần suất L, R
        Parent[L] = Parent[R] = k; //Cho k làm cha của L và R
        Bit[L] = 0; //L là nhánh 0
        Bit[R] = 1; //R là nhánh 1
        PQ.push(k); //Đẩy nút nhánh k vào PQ
        ++k; //Tăng số nút của trie
    }
}

void PrintCode(int x) //In mã nhị phân dọc trên đường đi từ gốc xuống x
{
    if (x == k - 1) //x là gốc cây
        return; //Không cần in gì cả
    PrintCode(Parent[x]); //Đệ quy in mã cha x
    cout << Bit[x]; //In ra tên nhánh đi xuống x
}

```

```
56  void Print() //In kết quả
57  {
58      for (int i = 0; i < n; ++i) //Xét n lá từ 0 tới n - 1
59      {
60          cout << a[i] << ": "; //Ký tự a[i] được mã hóa bởi dãy bit…
61          PrintCode(i); //In mã nhị phân dọc trên đường đi từ gốc xuống lá i
62          cout << '\n';
63      }
64      //Chi phí của trie bằng tổng tần suất các nút nhánh
65      cout << "Encoded size: " << accumulate(f + n, f + k, 0) << " bits";
66  }
67
68  int main()
69  {
70      ReadInput();
71      BuildHuffmanTrie();
72      Print();
73  }
```

Thuật toán Huffman có thời gian thực hiện $O(n \log n)$ với n là số lượng chữ cái trong bảng chữ cái của văn bản. Nó hoàn toàn có thể làm việc với dữ liệu bất kỳ chứ không nhất thiết phải là văn bản. Mã hóa Huffman được sử dụng rộng rãi trong các định dạng nén phổ biến như ZIP, BZIP, PNG và ngay cả JPEG.

Chúng ta đã khảo sát vài bài toán mà ở đó, thuật toán tham lam được thiết kế dựa trên một mô hình chia để trị. Thuật toán tham lam cần đưa ra quyết định tức thời tại mỗi bước lựa chọn. Quyết định này sẽ ảnh hưởng ngay tới sự lựa chọn ở bước kế tiếp. Chính vì vậy, đôi khi phân tích kỹ hai quyết định liên tiếp có thể cho ta những tính chất của nghiệm tối ưu và từ đó có thể xây dựng một thuật toán hiệu quả. Ta sẽ khảo sát một số ví dụ như vậy.

1.2.5. Lịch xử lý lỗi

❋ Bài toán

Bạn là một người quản trị một hệ thống thông tin, và trong hệ thống cùng lúc đang xảy ra n lỗi đánh số từ 0 tới $n-1$. Lỗi i sẽ gây ra thiệt hại d_i sau mỗi ngày và để khắc phục lỗi đó cần t_i ngày. Tại một thời điểm, bạn chỉ có thể xử lý một lỗi. Hãy lên lịch trình xử lý lỗi sao cho tổng thiệt hại của n lỗi là nhỏ nhất có thể.

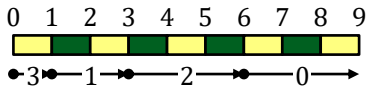
Input

- Dòng 1 chứa số nguyên dương $n \leq 10^5$
- n dòng tiếp theo, mỗi dòng chứa hai số nguyên dương $d_i, t_i \leq 10^4$ mô tả một lỗi

Output

Lịch xử lý lỗi

Sample Input	Sample Output
4 1 3 3 2 4 3 2 1	[0, 1]: Bug 3 (Damage = 2) [1, 3]: Bug 1 (Damage = 9) [3, 6]: Bug 2 (Damage = 24) [6, 9]: Bug 0 (Damage = 9) Minimum Damage = 44



❋ Thuật toán

Do mỗi lỗi chỉ ngưng gây thiệt hại khi nó được khắc phục nên để giảm thiểu chi phí thì khi bắt tay vào xử lý một lỗi sẽ phải làm liên tục cho tới khi khắc phục xong lỗi đó. Tức là ta cần tìm một hoán vị của dãy số $(0,1,\dots,n-1)$ tương ứng với thứ tự trong lịch trình khắc phục các lỗi.

Giả sử rằng lịch trình:

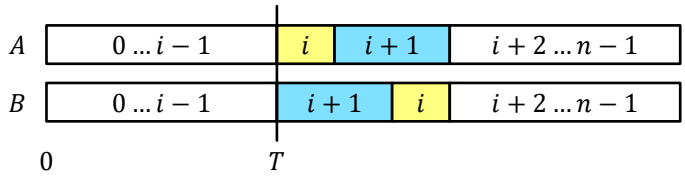
$$A = (0,1,\dots,\boxed{i,i+1},\dots,n-1)$$

xử lý lần lượt các lỗi từ 0 tới $n-1$ là lịch trình tối ưu. Ta lấy một lịch trình khác:

$$B = (0,1,\dots,\boxed{i+1,i},\dots,n-1)$$

có được bằng cách đảo thứ tự xử lý hai lỗi liên tiếp: i và $i+1$ trên lịch trình A

Nhận xét: ngoài hai lỗi i và $i+1$, thời điểm các lỗi khác được khắc phục là giống nhau trên hai lịch trình, có nghĩa là sự khác biệt về tổng thiệt hại của lịch trình A và lịch trình B chỉ nằm ở thiệt hại do hai lỗi i và $i+1$ gây ra.



Gọi T là thời điểm lỗi $i-1$ được khắc phục (trên cả hai lịch trình A và B). Khi đó:

Nếu theo lịch trình $A = (0,1,\dots i,i+1,\dots,n-1)$, thiệt hại của hai lỗi i và $i+1$ gây ra là:

$$\alpha = d_i \times (T + t_i) + d_{i+1} \times (T + t_i + t_{i+1})$$

Nếu theo lịch trình $B = (0,1,\dots,i+1,i,\dots,n-1)$, thiệt hại của hai lỗi i và $i+1$ gây ra là:

$$\beta = d_i \times (T + t_{i+1} + t_i) + d_{i+1} \times (T + t_{i+1})$$

Thiệt hại theo lịch trình A không thể lớn hơn thiệt hại theo lịch trình B (do A tối ưu), tức là thiệt hại α sẽ không lớn hơn thiệt hại β. Loại bỏ những hạng tử giống nhau ở công thức tính α và β, ta có:

$$\alpha \le \beta \Leftrightarrow d_{i+1} \times t_i \le d_i \times t_{i+1}$$
$$\Leftrightarrow \frac{t_i}{d_i} \le \frac{t_{i+1}}{d_{i+1}}$$

Vậy thì nếu lịch trình $A = (0,1,\dots,n-1)$ là tối ưu, nó sẽ phải thỏa mãn:

$$\frac{t_0}{d_0} \le \frac{t_1}{d_1} \le \dots \le \frac{t_{n-1}}{d_{n-1}}$$

Ngoài ra, nếu $\frac{t_i}{d_i} = \frac{t_{i+1}}{d_{i+1}}$ thì $\alpha = \beta$, tức là hai lịch trình sửa chữa A, B có cùng mức độ thiệt hại. Trong trường hợp này, việc sửa lỗi i trước hay $i+1$ trước đều cho các phương án cùng tối ưu.

Kết luận: phương án tối ưu cần tìm là phương án khắc phục các lỗi theo thứ tự tăng dần của tỉ số $\frac{t[.]}{d[.]}$. Lời giải đơn thuần chỉ là một thuật toán sắp xếp.

❋ Cài đặt

Ta biết rằng lỗi i bắt buộc phải sửa trước lỗi j nếu $\frac{t_i}{d_i} < \frac{t_j}{d_j}$. Việc dùng điều kiện này để làm phép so sánh trong thuật toán sắp xếp không có gì sai, tuy nhiên ta sẽ dùng điều kiện tương đương $d_j \times t_i < d_i \times t_j$ để việc tính toán có thể thực hiện chính xác tuyệt đối trên kiểu số nguyên.

BUGFIX.cpp Lịch xử lý lỗi

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int maxN = 1e5;
int n, d[maxN], t[maxN], id[maxN];
```

```cpp

void ReadInput() //Nhập dữ liệu
{
    cin >> n;
    for (int i = 0; i < n; ++i)
    {
        cin >> d[i] >> t[i];
        id[i] = i;
    }
}

void Greedy() //Thuật toán tham lam đơn giản là một phép sắp xếp
{
    sort(id, id + n, [] (int i, int j)
    {
        return d[j] * t[i] < d[i] * t[j]; //Lỗi i phải xếp trước lỗi j nếu…
    });
}

void Print() //In kết quả
{
    long long Time = 0, TotalDamage = 0;
    for (int i = 0; i < n; ++i) //Thứ tự chữa lỗi: id[0], id[1], …, id[n - 1]
    {
        int j = id[i];
        cout << "[" << Time << ", " << Time + t[j] << "]: ";
        Time += t[j];
        long long Damage = Time * d[j];
        cout << "Bug " << j << " (Damage = " << Damage << ")\n";
        TotalDamage += Damage;
    }
    cout << "Minimum Damage = " << TotalDamage;
}

int main()
{
    ReadInput();
    Greedy();
    Print();
}
```

Thuật toán thực hiện trong thời gian $O(n \log n)$, chi phí thời gian chủ yếu nằm ở thuật toán sắp xếp.

1.2.6. Lịch gia công trên hai máy

Bài toán

Bài toán lập lịch gia công trên hai máy (*Two-machine flow shop model*): Có n chi tiết đánh số từ 0 tới $n-1$ và hai máy A, B. Một chi tiết cần gia công trên máy A trước rồi chuyển sang gia công trên máy B. Thời gian gia công chi tiết i trên máy A và B lần lượt là a_i và b_i. Tại một thời điểm, mỗi máy chỉ có thể gia công một chi tiết. Hãy lập lịch gia công các chi tiết sao cho việc gia công toàn bộ n chi tiết được hoàn thành trong thời gian sớm nhất.

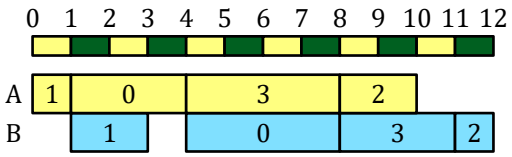
Input

- Dòng 1 chứa số nguyên dương $n \leq 10^5$
- n dòng tiếp, mỗi dòng chứa hai số nguyên dương a_i, b_i ứng với một chi tiết

Output

Lịch gia công tối ưu

Sample Input	Sample Output
4 3 4 1 2 2 1 4 3	1: A[0, 1]; B[1, 3] 0: A[1, 4]; B[4, 8] 3: A[4, 8]; B[8, 11] 2: A[8, 10]; B[11, 12] Execution makespan: 12



❋ **Phân tích lịch gia công**

Nhận xét rằng luôn tồn tại một phương án tối ưu mà các chi tiết gia công trên máy A theo thứ tự như thế nào thì sẽ được gia công trên máy B theo thứ tự đúng như vậy. Mỗi phương án có thể coi là một hoán vị của dãy n chi tiết.

Giả sử với giới hạn thời gian T, ta có cách thi hành phương án gia công P nào đó. Theo tư duy thông thường, cả hai máy sẽ khởi động cùng lúc tại thời điểm 0, khi đó máy A hoạt động liên tục còn máy B sẽ có những khoảng thời gian nghỉ để chờ chi tiết từ máy A. Tuy nhiên ta có thể lùi thời gian khởi động máy B để nó có thể hoạt động liên tục và hoàn thành đúng thời điểm T. Cụ thể là cho máy A hoạt động trong khoảng thời gian $[0, \sum_{i=0}^{n-1} a_i]$, máy B hoạt động trong khoảng thời gian $[T\text{-}\sum_{i=0}^{n-1} b_i\,, T]$, cả hai máy khi đã khởi động sẽ hoạt động không nghỉ và gia công tuần tự các chi tiết theo đúng thứ tự chỉ ra trong P. Trong mọi cách thi hành phương án P với giới hạn thời gian T, cách này cho phép mỗi chi tiết được gia công trên máy A sớm nhất và được gia công trên máy B muộn nhất có thể, đảm bảo rằng khi mỗi chi tiết bắt đầu gia công trên B thì nó đã hoàn tất việc gia công trên A.

Bổ đề 1-6 *(Quy tắc Johnson)*

Xét một phương án gia công $P = (p_0, p_1, \dots, p_{n-1})$ có thời gian hoàn thành là T. Xét hai chi tiết liên tiếp bất kỳ trong P: $y = p_i$ và $x = p_{i+1}$. Khi đó nếu:

$$\min(a_x, b_y) \le \min(a_y, b_x)$$

thì nếu đảo thứ tự hai chi tiết y và x cho nhau ta sẽ thu được phương án gia công Q có thời gian hoàn thành không vượt quá T.

Chứng minh

Phép chứng minh chỉ cần đưa ra một cách thi hành phương án Q để hoàn thành đúng thời điểm T là xong.

Với giới hạn thời gian T, ta thi hành phương án P cũng như phương án Q theo quy ước đã phân tích ở trên: Máy A hoạt động liên tục trong khoảng thời gian $[0, \sum_{i=0}^{n-1} a_i]$ còn máy B hoạt động liên tục trong khoảng thời gian $[T - \sum_{i=0}^{N-1} b_i\,, T]$.

Với mỗi máy, nó sẽ gia công tuần tự từng chi tiết theo phương án được giao, không có khoảng thời gian nào máy phải gia công cùng lúc nhiều chi tiết. Sự tương tranh không xảy ra trên lịch gia công của một máy.

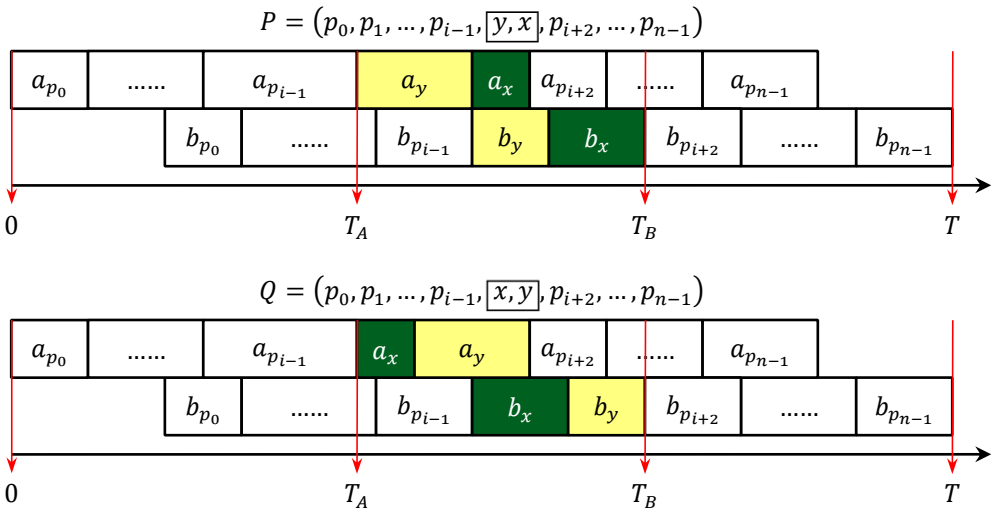
Tiếp theo, ta sẽ chỉ ra rằng sự tương tranh cũng không xảy ra trên lịch gia công của cả hai máy, tức là với mọi chi tiết, nó sẽ bắt đầu được gia công trên B khi và chỉ khi nó đã kết thúc việc gia công trên A, xét trên cả hai phương án P và Q.

Với mọi chi tiết $\notin \{x, y\}$ khoảng thời gian gia công nó trên máy A là như nhau trên hai phương án P và Q, khoảng thời gian gia công nó trên máy B cũng là như nhau trên hai phương án P và Q. Trong phương án P, chi tiết này được bắt đầu gia công trên máy B khi nó đã kết thúc việc gia công trên máy A, nên trên phương án Q cũng sẽ như vậy.

Riêng với hai chi tiết x, y, gọi:

- T_A là thời điểm máy A gia công xong chi tiết p_{i-1}
- T_B là thời điểm máy B bắt đầu gia công chi tiết p_{i+2}

Chú ý rằng T_A và T_B xác định, dù xét trên cách thi hành phương án P hay phương án Q (Hình 1-5)



Hình 1-5. Đảo hai chi tiết liên tiếp để tìm mỗi tương quan thứ tự

Vì lịch thi hành P là khả thi, mỗi chi tiết x, y sẽ được bắt đầu gia công trên B khi đã kết thúc gia công trên A. Tức là:

$$\begin{aligned} P \text{ khả thi} &\Longrightarrow \begin{cases} T_A + a_y + a_x \le T_B - b_x \\ T_A + a_y \le T_B - b_x - b_y \end{cases} \\ &\Leftrightarrow \begin{cases} a_y + b_x + a_x \le T_B - T_A \\ a_y + b_x + b_y \le T_B - T_A \end{cases} \\ &\Leftrightarrow a_y + b_x + \max(a_x, b_y) \le T_B - T_A \end{aligned} \tag{1.2}$$

Ta cần chứng minh trên phương án Q, mỗi chi tiết x, y cũng được bắt đầu gia công trên B khi đã kết thúc gia công trên A. Giả sử phản chứng rằng có ít nhất một trong hai chi tiết $\in \{x, y\}$ được gia công trên B khi nó chưa kết thúc gia công trên A. Khi đó:

$$\begin{aligned} Q \text{ không khả thi} &\Longrightarrow \begin{bmatrix} T_A + a_x > T_B - b_y - b_x \\ T_A + a_x + a_y > T_B - b_y \end{bmatrix} \\ &\Leftrightarrow \begin{bmatrix} a_x + b_y + b_x > T_B - T_A \\ a_y + b_y + a_y > T_B - T_A \end{bmatrix} \\ &\Leftrightarrow a_x + b_y + \max(a_y, b_x) > T_B - T_A \end{aligned} \tag{1.3}$$

Với hai số thực bất kỳ thì giá trị lớn nhất cộng với giá trị nhỏ nhất của hai số sẽ bằng tổng của chúng. Vì vậy hệ thức (1.2) và (1.3) có thể viết thành:

$$\begin{aligned} P \text{ khả thi} &\Longrightarrow a_x + a_y + b_x + b_y - \min(a_x, b_y) \le T_B - T_A \\ Q \text{ không khả thi} &\Longrightarrow a_x + a_y + b_x + b_y - \min(a_y, b_x) > T_B - T_A \end{aligned} \tag{1.4}$$

Từ giả thiết $\min(a_x, b_y) \le \min(a_y, b_x)$, ta suy ra

$$\begin{aligned} T_B - T_A &\ge a_x + a_y + b_x + b_y - \min(a_x, b_y) \\ &\ge a_x + a_y + b_x + b_y - \min(a_y, b_x) \\ &> T_B - T_A \text{ (Mâu thuẫn)} \end{aligned} \tag{1.5}$$

Mâu thuẫn trong hệ thức (1.5) cho thấy giả định "Có ít nhất một trong hai chi tiết $\in \{x, y\}$ được gia công trên B khi nó chưa kết thúc gia công trên A" là sai. Ta có ĐPCM.

❋ **Thuật toán Johnson**

Bổ đề 1-6 cho ta một kết quả quan trọng: Với một phương án tối ưu P bất kỳ, xét tập:

$$S = \{a_0, a_1, \dots, a_{n-1}, b_0, b_1, \dots, b_{n-1}\}$$

Nếu giá trị nhỏ nhất trong tập S (ký hiệu $\min(S)$) thuộc về a_x nào đó, nếu x không phải chi tiết đầu tiên trong P thì sẽ có chi tiết y đứng liền trước nó. Do:

$$\min(a_x, b_y) = a_x = \min(S) \leq \min(a_y, b_x)$$

Bổ đề 1-6 cho phép đảo thứ tự hai chi tiết x, y mà vẫn bảo tồn tính tối ưu của phương án P. Bằng một số lượt đảo thứ tự hai chi tiết liên tiếp như vậy, ta có thể đưa được chi tiết x về đầu dãy P. Suy ra chắc chắn tồn tại phương án tối ưu mà trong đó chi tiết x được gia công đầu tiên.

Nếu giá trị nhỏ nhất trong tập thuộc về b_y nào đó, nếu y không phải chi tiết cuối cùng trong P thì sẽ có chi tiết x đứng liền sau nó. Do:

$$\min(a_x, b_y) = b_y = \min(S) \leq \min(a_y, b_x)$$

Bổ đề 1-6 cho phép đảo thứ tự hai chi tiết x, y mà vẫn bảo tồn tính tối ưu của phương án. Tương tự như trên, ta có thể đưa chi tiết y về cuối dãy phương án bằng một số lượt đảo thứ tự hai chi tiết liên tiếp. Suy ra chắc chắn tồn tại phương án tối ưu mà chi tiết y được gia công cuối cùng.

❋ **Thuật toán Johnson**

Những nhận xét trên cho ta thuật toán: Khởi tạo phương án P gồm n ô trống. Khởi tạo tập S gồm n chi tiết. Thực hiện n bước lặp, mỗi bước ta lấy ra từ S chi tiết i có $\min(a_i, b_i)$ nhỏ nhất trên tất cả các phương án thuộc S:

- Nếu $a_i < b_i$: Đưa chi tiết i vào ô trống đầu tiên (do $\min(S) = a_i$)
- Nếu $a_i > b_i$: Đưa chi tiết i vào ô trống cuối cùng (do $\min(S) = b_i$)
- Nếu $a_i = b_i$: Đưa chi tiết i vào ô trống đầu tiên hay cuối cùng đều được.

(Khi một chi tiết được đưa vào ô trống, ô đó sẽ không còn trống nữa)

Thuật toán Johnson có thể cài đặt để thực hiện trong thời gian $O(n \log n)$ nếu như:

- Cài đặt tập S bằng những cấu trúc dữ liệu cho phép dễ dàng xác định phần tử nhỏ nhất và loại bỏ phần tử, chẳng hạn hàng đợi ưu tiên hoặc tập sắp thứ tự.
- Sắp xếp trước các chi tiết trong tập S theo thứ tự tăng dần của $\min(a_i, b_i)$ $(\forall i = 0,1, \dots, n)$, sau đó chỉ cần duyệt tập S và tiến hành đưa các chi tiết vào ô trống.
- Cách cài đặt hiệu quả nhất là:
  - Khởi tạo phương án P là một dãy các chi tiết, dãy này được chia làm hai đoạn: đoạn đầu gồm các chi tiết i có $a_i < b_i$, đoạn cuối gồm các chi tiết j có $a_j > b_j$, những chi tiết k có $a_k = b_k$ đưa vào đoạn đầu hay đoạn cuối đều được.
  - Sắp xếp các chi tiết trong đoạn đầu theo thứ tự tăng dần của $a[.]$, sắp xếp các chi tiết trong đoạn cuối theo thứ tự giảm dần của $b[.]$. Có thể áp dụng các thuật toán sắp xếp dựa vào giá trị khóa để đạt hiệu quả cao hơn so với các thuật toán sắp xếp chỉ dựa trên so sánh.

❋ **Cài đặt**

Thực tế, Johnson [14] chỉ đề xuất một luật chọn cho bài toán lập lịch gia công trên hai máy. Nhiều cách cài đặt khác nhau được đưa ra nhưng đều dựa trên luật chọn đó.

JOHNSON.cpp Lập lịch gia công trên 2 máy

```cpp
#include <iostream>
#include <algorithm>
using namespace std;
const int maxN = 1e5;

int n, k; //n: Số chi tiết, k: Số lượng chi tiết có a[.] < b[.]
int a[maxN], b[maxN], p[maxN];

void ReadInput()
{
    cin >> n;
    k = 0;
    int j = n;
    //Ban đầu coi như p gồm toàn "ô trống"
    for (int i = 0; i < n; ++i)
    {
        cin >> a[i] >> b[i];
        if (a[i] < b[i]) p[k++] = i; //Đưa i vào ô trống đầu tiên trong p
        else p[--j] = i; //Đưa i vào ô trống cuối cùng trong p
    }
}

void Greedy() //Thuật toán tham lam
{
    //Sắp xếp đoạn đầu p[0...k) theo thứ tự tăng dần của a[.]
    sort(p, p + k, [](int x, int y)
    {
        return a[x] < a[y];
    });
    //Sắp xếp đoạn sau p[k...n) theo thứ tự giảm dần của b[.]
    sort(p + k, p + n, [](int x, int y)
    {
        return b[x] > b[y];
    });
}

void Print() //In kết quả
{
    int aTime = 0, bTime = 0;
    for (int i = 0; i < n; ++i)
    {
        int x = p[i]; //Gia công chi tiết x
        //ATime = Thời điểm A bắt đầu làm x
        cout << x << ": ";
        cout << "A[" << aTime << ", " << aTime + a[x] << "]; ";
        aTime += a[x]; //ATime = Thời điểm A kết thúc làm x
        bTime = max(bTime, aTime); //BTime = Thời điểm B bắt đầu làm x
        cout << "B[" << bTime << ", " << bTime + b[x] << "]\n";
        bTime += b[x]; //BTime = Thời điểm B kết thúc làm x
    }
    cout << "Execution makespan: " << bTime;
}

int main()
{
    ReadInput();
    Greedy();
    Print();
}
```

❋ Tránh mắc sai lầm

Định nghĩa một quan hệ hai ngôi, ký hiệu "≼", giữa các chi tiết:

$$x \preccurlyeq y \Leftrightarrow \min(a_x, b_y) \leq \min(a_y, b_x)$$

Bổ đề 1-6 là cơ sở quan trọng của thuật toán, nội dung của nó là với hai chi tiết liên tiếp (y, x) trong một phương án thỏa mãn $x \preccurlyeq y$, ta có thể đảo thứ tự gia công thành (x, y) mà vẫn thi hành được phương án trong thời gian không đổi. Tuy nhiên cần nhấn mạnh rằng điều này không đúng nếu y và x không phải hai chi tiết liên tiếp.

Trong bài toán Lịch xử lý lỗi, quan hệ tìm ra giữa hai phần tử liên tiếp là một quan hệ thứ tự yếu, nó suy ra hiển nhiên từ quan hệ thứ tự yếu của tập các phân số. Chính vì vậy một thuật toán sắp xếp theo quan hệ thứ tự yếu là phù hợp và chính xác để tìm ra phương án tối ưu.

Tuy nhiên trong bài toán Lịch gia công trên hai máy, quan hệ "$\preccurlyeq$" không phải quan hệ thứ tự yếu và vì thế nó cũng không dẫn xuất ra được bất kỳ quan hệ thứ tự yếu ngặt "$\prec$" nào cả. Việc máy móc áp dụng thuật toán sắp xếp so sánh với quan hệ $\preccurlyeq$ là sai lầm nghiêm trọng, sẽ cho kết quả sai, hoặc chương trình sinh lỗi, tùy theo cách cài đặt thuật toán sắp xếp.

Ví dụ: Có 3 chi tiết với thời gian gia công trên hai máy là:

$$x: (a_x = 3, b_x = 2)$$
$$y: (a_y = 1, b_y = 1)$$
$$z: (a_z = 2, b_z = 3)$$

Ta có:

$x \preccurlyeq y$ vì $\min(a_x, b_y) = 1 = \min(a_y, b_x)$

$y \preccurlyeq z$ vì $\min(a_y, b_z) = 1 = \min(a_z, b_y)$

Tuy nhiên $x \not\preccurlyeq z$ vì $\min(a_x, b_z) = 3 > 2 = \min(a_z, b_x)$. Quan hệ "$\preccurlyeq$" không có tính bắc cầu.

Với phương án có thứ tự ban đầu (x, y, z) (thời gian thi hành bằng 9), thuật toán sắp xếp theo quan hệ $\preccurlyeq$ rất có thể cho rằng dãy đã sắp xếp rồi (do $x \preccurlyeq y$ và $y \preccurlyeq z$) và không đưa ra sự điều chỉnh nào cả, mặc dù phương án (x, y, z) không tối ưu. Trong ví dụ này ba phương án (y, z, x), (z, y, x) hoặc (z, x, y) mới là tối ưu với thời gian thi hành bằng 8.

Các phân tích, nhận xét nhằm tìm ra đặc điểm thứ tự của hai phần tử liên tiếp đôi khi không tìm ra được thứ tự của hai phần tử bất kỳ trong phương án tối ưu. Việc máy móc áp dụng thuật toán sắp xếp trong trường hợp này (hay nói chung là áp dụng các giải pháp không phù hợp với ngữ cảnh) sẽ dẫn tới những lỗi sai rất khó phát hiện, đôi khi còn khiến cho lập trình viên nghi ngờ về tính đúng đắn của thuật toán. Đây cũng là kinh nghiệm khi thiết kế thuật toán: ta cần phải chứng minh chặt chẽ với đầy đủ cơ sở toán học mới có thể hiểu đúng bản chất và tránh được những sai sót trong cài đặt.

Riêng đối với bài toán Lịch gia công trên hai máy, quan hệ "$\preccurlyeq$" sẽ trở thành quan hệ thứ tự yếu nếu $a_i \neq b_i$ với mọi chi tiết i. Khi đó phương án tối ưu sẽ đơn giản là sắp xếp các chi tiết theo nguyên tắc: Chi tiết x phải xếp trước chi tiết y nếu:

$$\min(a_x, b_y) < \min(a_y, b_x)$$

Việc chứng minh trong trường hợp này "$\preccurlyeq$" là quan hệ thứ tự yếu và thuật toán trên là đúng đắn, chúng ta coi như bài tập.

❋ **Trường hợp gia công trên 3 máy**

Bài toán lập lịch gia công với ≥ 3 máy là bài toán NP- đầy đủ. Hiện không có thuật toán với độ phức tạp đa thức để giải. Riêng trường hợp 3 máy theo thứ tự A, B, C, nếu thời gian gia công chi tiết i trên ba máy lần lượt là $a[i], b[i], c[i]$, thì nếu dữ liệu thỏa mãn:

$$\max_{\forall i}\{b_i\} \le \min_{\forall i}\{a_i\}$$

hoặc:

$$\max_{\forall i}\{b_i\} \le \min_{\forall i}\{c_i\}$$

(Tức là thời gian gia công của máy B khá nhỏ so với máy A hoặc máy C)

Trong trường hợp này, thứ tự gia công tối ưu trên ba máy A, B, C sẽ trùng với thứ tự gia công tối ưu trên 2 máy AB, BC. Trong đó thời gian gia công chi tiết i trên máy AB là $a_i + b_i$ và thời gian gia công chi tiết i trên máy BC là $b_i + c_i$.

Bài tập 1-1

Bạn là người lập lịch giảng dạy cho n chuyên đề, chuyên đề thứ i cần bắt đầu ngay sau thời điểm s_i và kết thúc tại thời điểm f_i: $(s_i, f_i]$. Mỗi chuyên đề trong thời gian diễn gia hoạt động giảng dạy sẽ cần một phòng học riêng. Hãy xếp lịch giảng dạy sao cho số phòng học phải sử dụng là ít nhất. Tìm thuật toán $O(n \log n)$.

Bài tập 1-2

Trên trục số cho n điểm đen và n điểm trắng hoàn toàn phân biệt. Hãy tìm thuật toán $O(n \log n)$ để xác định đoạn, mỗi đoạn nối một điểm đen với một điểm trắng, sao cho không có hai đoạn thẳng nào có chung đầu mút và tổng độ dài n đoạn là nhỏ nhất có thể.

Bài tập 1-3

Chứng minh rằng để mã hóa một xâu bằng thuật toán Huffman, số bit phải sử dụng trong xâu mã bằng tổng tần suất của các nút trong cây Huffman trừ đi tần suất của nút gốc.

Bài tập 1-4

Xét mã phi tiền tố của một tập n ký tự. Nếu các ký tự được sắp xếp sẵn theo thứ tự không giảm của tuần suất thì chúng ta có thể xây dựng cây Huffman trong thời gian $O(n)$ bằng cách sử dụng hai hàng đợi: Tạo ra các nút lá chứa ký tự và đẩy chúng vào hàng đợi 1 theo thứ tự từ nút tần suất thấp nhất tới nút tần suất cao nhất. Khởi tạo hàng đợi 2 rỗng, lặp lại các thao tác sau $n - 1$ lần:

- Lấy hai nút x, y có tần suất thấp nhất ra khỏi các hàng đợi bằng cách đánh giá các phần tử ở đầu của cả hai hàng đợi
- Tạo ra một nút z làm nút cha của hai nút x, y với tần suất bằng tổng tần suất của x và y và đẩy z vào cuối hàng đợi 2

Chứng minh tính đúng đắn và cài đặt thuật toán trên.

Bài tập 1-5

Bài toán xếp ba lô (Knapsack) chúng ta đã khảo sát trong chuyên đề kỹ thuật nhánh cận và quy hoạch động là bài toán 0/1 Knapsack: Với một sản phẩm chỉ có hai trạng thái: chọn hay không chọn. Chúng ta cũng đã khảo sát bài toán Fractional Knapsack: Cho phép chọn một phần sản phẩm: Với một sản phẩm trọng lượng w và giá trị v, nếu ta lấy một phần trọng lượng $w' \le w$ của

sản phẩm đó thì sẽ được giá trị là $v * \frac{w'}{v}$. Chỉ ra rằng hàm $Estimate$ trong mục **Error! Reference source not found.** cho kết quả tối ưu đối với bài toán Fractional Knapsack.

Bài tập 1-6

Giáo sư X lái xe trong một chuyến hành trình "Xuyên Việt" từ Cao Bằng tới Cà Mau dài l km. Dọc trên đường đi có n trạm xăng và trạm xăng thứ i cách Cao Bằng d_i km. Bình xăng của xe khi đổ đầy có thể đi được m km. Hãy tìm thuật toán $O(n)$ xác định các điểm dừng để đổ xăng tại các trạm xăng sao cho số lần phải dừng đổ xăng là ít nhất trong cả hành trình.

Bài tập 1-7

Cho n điểm thực trên trục số: $x_1, x_2, \dots, x_n \in \mathbb{R}$. Hãy chỉ ra ít nhất các đoạn dạng $[i, i+1]$ với i là số nguyên để phủ hết n điểm đã cho.

Bài tập 1-8

Bạn có n nhiệm vụ, mỗi nhiệm vụ cần làm trong một đơn vị thời gian. Nhiệm vụ thứ i có thời điểm phải hoàn thành (deadline) là d_i và nếu bạn hoàn thành nhiệm vụ đó sau thời hạn d_i thì sẽ phải mất một khoản phạt p_i. Bạn bắt đầu từ thời điểm 0 và tại mỗi thời điểm chỉ có thể thực hiện một nhiệm vụ. Hãy tìm thuật toán $O(n \log n)$ để lên lịch thực hiện các nhiệm vụ sao cho tổng số tiền bị phạt là ít nhất.

Bài tập 1-9

Một lần Tôn Tẫn đua ngựa với vua Tề. Tôn Tẫn và vua Tề mỗi người có n con ngựa đánh số từ 1 tới n, con ngựa thứ i của Tôn Tẫn có tốc độ là a_i, con ngựa thứ i của vua Tề có tốc độ là b_i. Luật chơi như sau:

- Có tất cả n cặp đua, mỗi cặp đua có một ngựa của Tôn Tẫn và một ngựa của vua Tề.
- Con ngựa nào cũng phải tham gia đúng một cặp đua
- Trong một cặp đua, con ngựa nào tốc độ cao hơn sẽ thắng, nếu hai con ngựa có cùng tốc độ thì kết quả của cặp đua đó sẽ hoà.
- Trong một cặp đua, con ngựa của bên nào thắng thì bên đó sẽ được 1 điểm, hoà và thua không có điểm.

Hãy giúp Tôn Tẫn chọn ngựa ra đấu n cặp đua với vua Tề sao cho hiệu số: Điểm của Tôn Tẫn - Điểm của vua Tề là lớn nhất có thể.

Bài tập 1-10

Bạn có n dự án phải thực hiện, mỗi dự án i cần một số tiền đầu tư a_i để mua thiết bị, thuê nhân công trước khi thực hiện và sau khi hoàn thành bạn thu lại được số tiền bằng b_i. Các dự án có thể thực hiện theo thứ tự bất kỳ. Bạn cần tính số vốn ít nhất cần có ban đầu để có thể hoàn thành hết n dự án (tiền thu lại từ mỗi dự án sẽ được cộng vào vốn sau khi hoàn thành dự án). Tìm thuật toán $O(n \log n)$.

Bài tập 1-11

Trong bài toán Lịch gia công trên hai máy, giả sử $a_i \neq b_i$ với mọi chi tiết i. Chứng minh rằng quan hệ "$\preccurlyeq$" giữa hai chi tiết định nghĩa bởi:

$$x \preccurlyeq y \Leftrightarrow \min(a_x, b_y) \leq \min(a_y, b_x)$$

là một quan hệ thứ tự yếu. Từ đó suy ra quan hệ "$\prec$":

$$x \prec y \Leftrightarrow \min(a_x, b_y) < \min(a_y, b_x)$$

là quan hệ thứ tự yếu ngặt.

Bài tập 1-12

Xét bài toán Lịch gia công trên hai máy trong đó thời gian gia công mỗi chi tiết trên mỗi máy đều là số nguyên không âm, giả sử có chi tiết i thỏa mãn $a_i = b_i$. Chứng minh rằng với mọi phương án P, ta có thể rút ra chi tiết i rồi chèn vào vị trí bất kỳ trong dãy còn lại mà không làm thay đổi thời gian thi hành của P.

Kết hợp với Bài tập 1-11, chứng minh rằng quan hệ "$\preccurlyeq$" trên các chi tiết định nghĩa bởi:

$$x \preccurlyeq y \Leftrightarrow \min(2 \times a_x, 2 \times b_y + 1) \leq \min(2 \times a_y, 2 \times b_x + 1)$$

là một quan hệ thứ tự yếu. Từ đó suy ra thuật toán sắp xếp các chi tiết theo nguyên tắc: Chi tiết x phải xếp trước chi tiết y nếu:

$$\min(2 \times a_x, 2 \times b_y + 1) < \min(2 \times a_y, 2 \times b_x + 1)$$

sẽ cho phương án tối ưu.