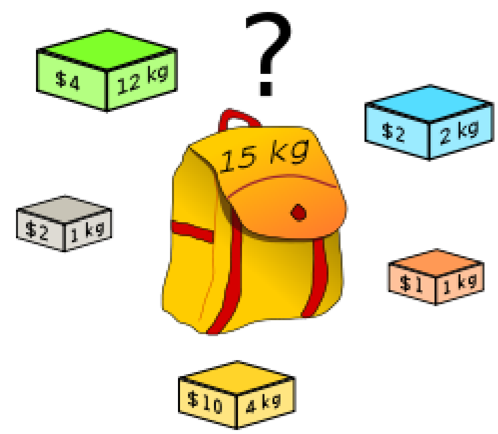
WIKIPEDIA

Bài toán xếp ba lô

Bách khoa toàn thư mở Wikipedia

Bài toán xếp ba lô (còn được biết đến với tên gọi **bài toán cái túi**) là một bài toán tối ưu hóa tổ hợp. Bài toán được đặt tên từ vấn đề chọn những gì quan trọng có thể nhét vừa vào trong một cái túi (với giới hạn khối lượng) để mang theo trong một chuyến đi. Các bài toán tương tự thường xuất hiện trong kinh doanh, toán tổ hợp, lý thuyết độ phức tạp tính toán, mật mã học và toán ứng dụng.



Ví dụ về một bài toán xếp ba lô giới hạn 1 chiều: chọn các hộp nào để làm cực đại lượng tiền trong khi giữ được tổng khối lượng dưới 15 kg? Bài toán đa chiều có thể xét đến khối lượng riêng và kích thước của các hộp, đó là bài toán xếp vali điển hình (*packing problem*).
(Lời giải là chọn tất cả các hộp trừ hộp xanh lục.)

Mục lục

Nội dung

Một kẻ trộm đột nhập vào một cửa hiệu tìm thấy có n mặt hàng có trọng lượng và giá trị khác nhau, nhưng hắn chỉ mang theo một cái túi có sức chứa về trọng lượng tối đa là M. Vậy kẻ trộm nên bỏ vào ba lô những món nào và số lượng bao nhiêu để đạt giá trị cao nhất trong khả năng mà hắn có thể mang đi được.

Dạng bài toán quyết định của bài toán xếp ba lô là câu hỏi "có thể đạt được một giá trị ít nhất là bao nhiêu theo phát biểu của bài toán".

Ta có n loại đồ vật, x_1 tới x_n. Mỗi đồ vật x_j có một giá trị p_j và một khối lượng w_j. Khối lượng tối đa mà ta có thể mang trong ba lô là C.

Bài xếp ba lô dạng 0-1

Hạn chế số đồ vật thuộc mỗi loại là 0 (không được chọn) và 1 (được chọn).

Bài xếp ba lô 0-1 có thể được phát biểu bằng toán học như sau:

Cực đại hóa $\sum_{j=1}^{n} p_j x_j .$

$$\text{sao cho} \sum_{j=1}^{n} w_j x_j \leq c, \qquad x_j = 0 \text{ or } 1, \quad j = 1, \ldots, n.$$

Bài xếp ba lô bị chặn hạn chế số đồ vật không được vượt quá một lượng nào đó.

> Bài xếp ba lô bị chặn có thể được phát biểu bằng toán học như sau:
>
> $$\text{Cực đại hóa} \sum_{j=1}^{n} p_j x_j.$$
>
> $$\text{sao cho} \sum_{j=1}^{n} w_j x_j \leq c, \qquad 0 \leq x_j \leq b_j, \quad j = 1, \ldots, n.$$

Bài xếp ba lô không bị chặn không có một hạn chế nào về số lượng đồ vật.

Trường hợp đặc biệt

Bài toán với các tính chất:

- là một bài toán quyết định
- là một bài toán 0/1
- với mỗi đồ vật, chi phí bằng giá trị: *C* = *V*

Lưu ý rằng trong trường hợp đặc biệt này, bài toán tương đương với:

> Cho một tập các số nguyên, tồn tại hay không một tập con có tổng đúng bằng *C*?

Hoặc nếu đồ vật được phép có chi phí âm và *C* được chọn bằng 0, bài toán có dạng:

> Cho trước một tập các số nguyên, tồn tại hay không một tập con có tổng đúng bằng *0*?

Trường hợp đặc biệt này được gọi là bài toán tổng các tập con (*subset sum problem*). Với một số lý do, trong ngành mật mã học, người ta thường dùng cụm từ "bài toán xếp ba lô" khi thực ra đang có ý nói về "bài toán tổng con".

Bài toán xếp ba lô thường được giải bằng quy hoạch động, tuy chưa có một thuật toán thời gian đa thức cho bài toán tổng quát. Cả bài xếp ba lô tổng quát và bài toán tổng con đều là các bài NP-khó, và điều này dẫn đến các cố gắng sử dụng tổng con làm cơ sở cho các hệ thống mật mã hóa khóa công khai, chẳng hạn Merkle-Hellman. Các cố gắng này thường dùng nhóm thay vì các số nguyên. Merkle-Hellman và một số thuật toán tương tự khác đã bị phá, do các bài toán tổng con cụ thể mà họ tạo ra thực ra lại giải được bằng các thuật toán thời gian đa thức.

Phiên bản bài toán quyết định của bài xếp ba lô được mô tả ở trên là NP-đầy đủ và trong thực tế là một trong 21 bài toán NP-đầy đủ của Karp.

Bài xếp ba lô dạng phân số

Với mỗi loại, có thể chọn một phần của nó (ví dụ: 1Kg bơ có thể được cắt ra thành nhiều phần để bỏ vào ba lô)

Cách giải bằng quy hoạch động

Bài toán xếp ba lô có thể được giải trong thời gian giả-đa thức bằng quy hoạch động. Dưới đây là lời giải quy hoạch động cho *bài toán xếp ba lô không bị chặn*.

Gọi các chi phí là $c_1,..., c_n$ và các giá trị tương ứng là $v_1,..., v_n$. Ta cần cực đại hóa tổng giá trị với điều kiện tổng chi phí không vượt quá C. Khi đó, với mỗi $i \le C$, đặt $A(i)$ là giá trị lớn nhất có thể đạt được với tổng chi phí không vượt quá i. Rõ ràng, $A(C)$ là đáp số của bài toán.

Định nghĩa $A(i)$ một cách đệ quy như sau:

- $A(0) = 0$
- $A(i) = \max \{ v_j + A(i - c_j), A(i) \mid c_j \le i \}$

Ở đây, giá trị lớn nhất của tập rỗng được lấy bằng 0. Tính dần các kết quả từ $A(0)$ tới $A(C)$, ta sẽ được lời giải. Do việc tính mỗi $A(i)$ đòi hỏi xem xét n đồ vật (tất cả các giá trị này đã được tính từ trước), và có C giá trị của các $A(i)$ cần tính, nên thời gian chạy của lời giải quy hoạch động là $O(nC)$.

Điều này không mâu thuẫn với thực tế rằng bài toán xếp ba lô là NP-đầy đủ, do C, không như n, không thuộc mức đa thức theo độ dài của đầu vào cho bài toán. Độ dài đầu vào bài toán tỉ lệ thuận với số bit trong C, chứ không tỉ lệ với chính C.

Một giải pháp quy hoạch động tương tự cho *bài toán xếp ba lô 0-1* cũng chạy trong thời gian giả-đa thức. Cũng như trên, gọi các chi phí là $c_1,..., c_n$ và các giá trị tương ứng là $v_1,..., v_n$. Ta cần cực đại hóa tổng giá trị với điều kiện tổng chi phí không vượt quá C. Định nghĩa một hàm đệ quy $A(i, j)$ là giá trị lớn nhất có thể đạt được với chi phí không vượt quá j và sử dụng các đồ vật trong khoảng từ x_1 tới x_i.

$A(i,j)$ được định nghĩa đệ quy như sau:

- f[i][j]:= giá trị lớn nhất có thể lấy được;
- v[i]:= là giá trị của đồ vật thứ i;
- w[i]:=trọng lượng của đồ vật thứ i
- NOTES: Đây là bài toán không giới hạn đồ vật
- ***f[i][j]=max(f[i-1][j],v[i]+f[i][j-w[i]])*** *// khi đã lấy vật thứ* ***i*** *thì có thể lấy thêm nữa nên ta có* ***f[i][j-w[i]]:=*** *đã trừ đi trọng lượng của đồ vật thứ i và vẫn có thể lấy thêm được nữa // i là đồ vật không giới hạn*
- $A(0, j) = 0$
- $A(i, 0) = 0$
- $A(i, j) = A(i - 1, j)$ if $c_i > j$
- $A(i, j) = \max(A(i - 1, j), v_i + A(i - 1, j - c_i))$ if $c_i \le j$

Để có lời giải, ta tính $A(n, C)$. Để làm điều này, ta có thể dùng 1 bảng để lưu các tính toán trước đó. Cách giải này do đó sẽ chạy trong thời gian $O(nC)$ và không gian $O(nC)$, tuy ta có thể giảm độ phức tạp không gian xuống $O(C)$ bằng một số sửa đổi nhỏ.

Thuật toán tham lam

Martello và Toth (1990) đã đưa ra một thuật toán gần đúng kiểu tham lam (*greedy approximation algorithm*) để giải bài toán xếp ba lô. Giải thuật này sắp xếp các đồ vật theo thứ tự giảm dần về giá trị, sau đó theo thứ tự đó xếp các đồ vật vào ba lô cho đến khi không cho thêm được đồ vật nào vào nữa.

Tham khảo

Sách tham khảo

- Garey, Michael R.; David S. Johnson (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman. ISBN 0-7167-1045-5. A6: MP9, pg.247.
- Kellerer, Hans; Pferschy, Ulrich; Pisinger, David (2004). *Knapsack Problems*. Springer. ISBN 3-540-40286-1. MR 2161720. doi:10.1007/978-3-540-24777-7.
- Martello, Silvano; Toth, Paolo (1990). *Knapsack problems: Algorithms and computer implementations*. Wiley-Interscience. ISBN 0-471-92420-2. MR 1086874.

Liên kết ngoài

- Free download of the book "Knapsack problems: Algorithms and computer implementations", by Silvano Martello and Paolo Toth (http://www.or.deis.unibo.it/knapsack.html)
- Lecture slides on the knapsack problem (http://www.cse.unl.edu/~goddard/Courses/CSCE310J/Lectures/Lecture8-DynamicProgramming.pdf)
- PYAsUKP: Yet Another solver for the Unbounded Knapsack Problem (http://download.gna.org/pyasukp), with code taking advantage of the dominance relations in an hybrid algorithm, benchmarks and downloadable copies of some papers.
- Home page of David Pisinger (http://www.diku.dk/~pisinger/) with downloadable copies of some papers on the publication list (including "Where are the hard knapsack problems?")
- Knapsack Problem solutions in many languages (http://rosettacode.org/wiki/Knapsack_Problem) at Rosetta Code
- Dynamic Programming algorithm to 0/1 Knapsack problem (http://www.personal.kent.edu/~rmuhamma/Algorithms/MyAlgorithms/Dynamic/knapsackdyn.htm)
- Knapsack Problem solver (online) (http://karaffeltut.com/NEWKaraffeltutCom/Knapsack/knapsack.html)
- Solving 0-1-KNAPSACK with Genetic Algorithms in Ruby (http://www.nils-haldenwang.de/computer-science/computational-intelligence/genetic-algorithm-vs-0-1-knapsack)
- Codes for Quadratic Knapsack Problem (http://www.adaptivebox.net/CILib/code/qkpcodes_link.html)