Thuật toán tham lam (7)

Nguyễn Thanh Bình
Khoa Công nghệ Thông tin
Trường đại học Bách khoa
Đại học Đà Nẵng

Thuật toán tham lam (greedy algorithms)

- Vấn đề tìm kiếm giải pháp tối ưu
  - Chia bài toán thành nhiều bài toán con
  - Giải quyết các bài toán con
  - Giải pháp của các bài toán con sẽ là giải pháp cho bài toán đặt ra
- Thuật toán chia để trị hoặc thuật toán đơn giản
  - Giải quyết **tất cả các bài toán con**
  - Độ phức tạp cao (thường hàm mũ)
  - Dễ thiết kế, dễ cài đặt
- Thuật toán quy hoạch động
  - **Ghi nhớ lại giải pháp của các bài toán con** (khi các bài toán con không hoàn toàn độc lập) để tránh các xử lý trùng lặp
  - Độ phức tạp thấp hơn (thường hàm đa thức)
  - Tuy nhiên, khó thiết kế và cài đặt giải pháp

Thuật toán tham lam

- Nguyên tắc thuật toán tham lam
  - Tối ưu từng bước -> tối ưu toàn cục
  - Chỉ giải quyết một bài toán con
    - Không xét tất cả các bài toán con
  - Xây dựng giải pháp từng bước một
    - Ở mỗi bước, thực hiện sự chọn lựa tốt nhất tại thời điểm đó (giải pháp cục bộ)
      - Không có giải pháp tổng thể
    - Không quay lại xem xét quyết định đã chọn lựa
    - Hy vọng kết quả thu được là giải pháp tối ưu

Thuật toán tham lam

- Ưu điểm
  - dễ thiết kế
  - dễ cài đặt
  - độ phức tạp thấp
- Nhược điểm
  - Không phải luôn cho giải pháp tối ưu
  - Khó để chứng minh thuật toán cho giải pháp tối ưu

Thuật toán tham lam

- Cấu trúc tổng quát

```
thamlam(C: tập hợp các ứng cử viên)
// hàm trả về giải pháp tối ưu, gồm các ứng cử viên
begin
    S = ∅          // S là giải pháp tối ưu
    while (C ≠ ∅ và S chưa là giải pháp) do
       x = chọn(C) // chọn x từ tập C theo tiêu chí của hàm chọn
       C = C - {x}
       if (S U {x} có triển vọng là giải pháp) then S:=S U {x}
       endif
    endwhile
       if (S là lời giải) then return S
       else return 0
       endif
end
```

Một số ứng dụng

- Thuê xe
- Đổi tiền
- Xếp ba lô
- Mã Huffman
- Tìm cây khung nhỏ nhất
  - Thuật toán Kruskal
  - Thuật toán Prim
- Tìm đường đi ngắn nhất
  - Thuật toán Dijsktra

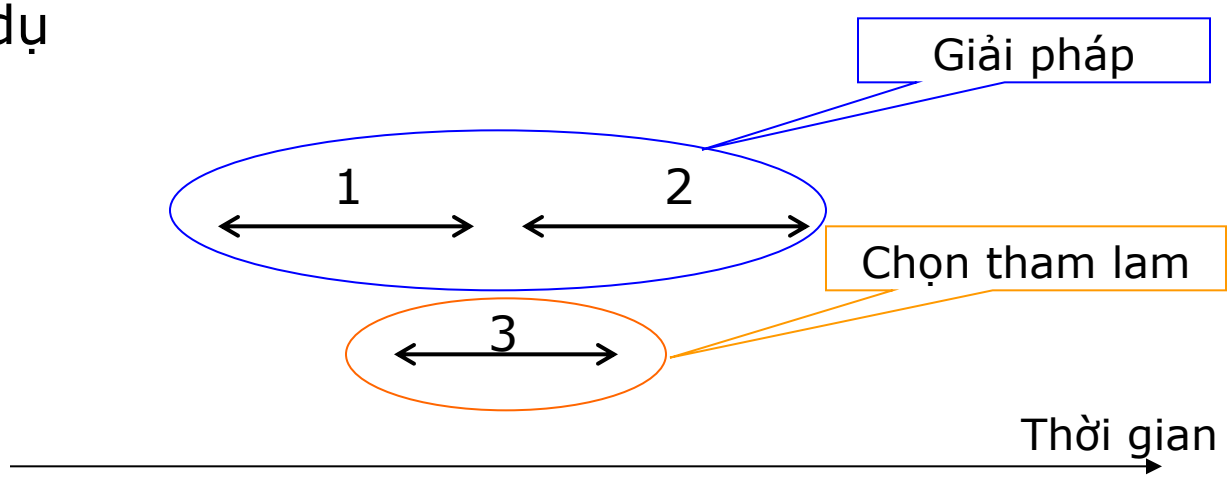
Thuê xe

- Bài toán
  - Có một chiếc xe ôtô duy nhất và có nhiều khách hàng yêu cầu được thuê xe. Mỗi yêu cầu thuê xe có một thời điểm bắt đầu thuê và một thời điểm kết thúc thuê.
  - Vấn đề: sắp xếp việc cho thuê làm sao để *số khách hàng* được thuê xe là nhiều nhất

- Hình thức hoá
  - Giả sử $S = \{a_1, a_2, \ldots, a_n\}$ tập hợp các yêu cầu thuê xe
  - Mỗi $a_i \in S$, $s(a_i)$ là thời điểm bắt đầu thuê và $f(a_i)$ là thời điểm kết thúc thuê
  - Gọi F là tập hợp lớn nhất các yêu cầu thoả mãn:
    - Hai yêu cầu bất kỳ phải lệnh nhau về thời gian, nghĩa là yêu cầu tiếp theo chỉ được bắt đầu khi yêu cầu trước đó kết thúc
    - Hay: $\forall a_1 \in F, a_2 \in F: s(a_1) \leq s(a_2) \Rightarrow f(a_1) \leq s(a_2)$

Thuê xe

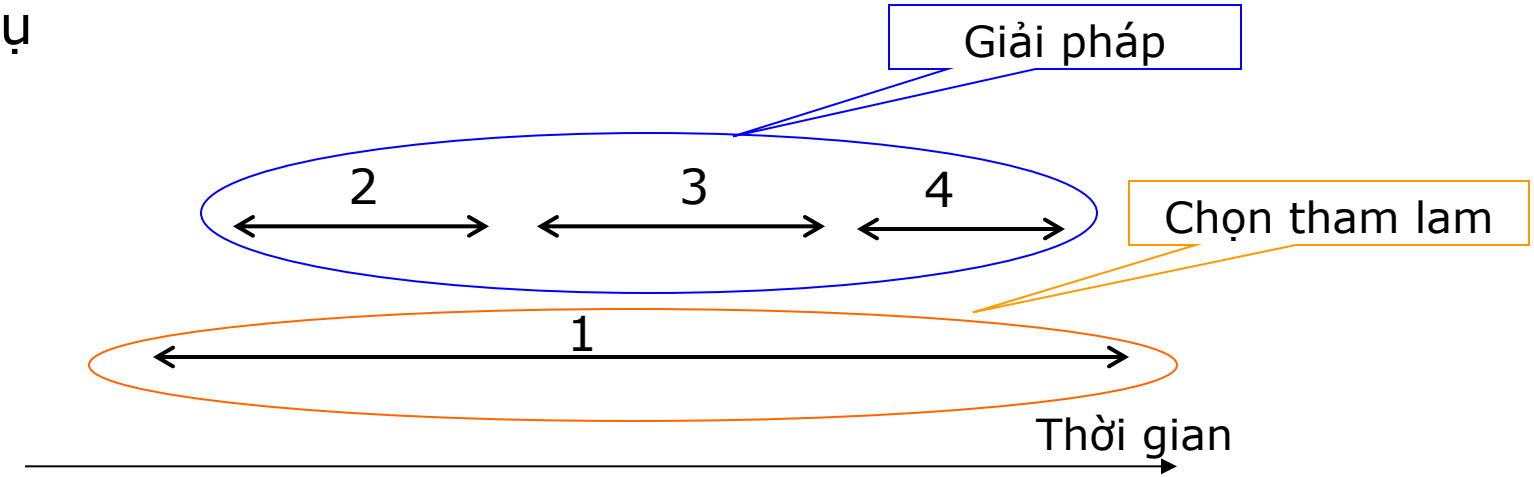
- Phép thử 1
  - Sắp xếp các yêu cầu thuê xe theo thứ tự tăng dần thời gian thuê
  - Chọn tham lam
    - chọn yêu cầu có thời gian thuê ngắn nhất
  - Ví dụ



Thuật toán không cho giải pháp tối ưu

Thuê xe

- Phép thử 2
  - Sắp xếp các yêu cầu thuê xe theo thứ tự thời điểm bắt đầu thuê
  - Chọn tham lam
    - chọn yêu cầu có thời điểm bắt đầu thuê sớm nhất
  - Ví dụ



Thuật toán cũng không cho giải pháp tối ưu

Thuê xe

- Phép thử 3
  - Sắp xếp các yêu cầu theo thứ tự thời điểm kết thúc thuê tăng dần
  - Chọn tham lam
    - chọn yêu cầu có thời điểm kết thúc thuê sớm nhất
  - Thuật toán cho giải pháp tối ưu
    - Tại sao ?
    - Chứng minh !

Thuê xe

- Chứng minh tính tối ưu của thuật toán
  - Giả sử F={x_1, x_2, …, x_p} là giải pháp đạt được bởi thuật toán tham lam và G={y_1, y_2, …, y_q} với q≥p là một giải pháp tối ưu (cho phép thực hiện nhiều yêu cầu nhất)
  - Cần chứng minh F là giải pháp tối ưu, nghĩa là p=q
  - Giả sử các phần tử của các tập hợp F và G được sắp xếp theo thứ tự thời điểm kết thúc thuê tăng dần
  - Nếu G không chứa F, thì phải tồn tại k sao cho: $\forall i<k$, $x_i=y_i$ và $x_k \neq y_k$.
  - Vì x_k được chọn bởi thuật toán tham lam, nên x_k có thời điểm kết thúc $f(x_k)$ nhỏ nhất. Vậy $f(x_k) \leq f(y_k)$. Mà $f(y_k) \leq s(y_{k+1})$, nên $f(x_k) \leq s(y_{k+1})$. Tức là, x_k và y_{k+1} lệch nhau về thời gian. Ngoài ra, do cùng thuộc giải pháp *F*, nên x_{k-1} và x_k lệch nhau về thời gian. Mà theo giả thiết thì $x_{k-1} = y_{k-1}$, vậy y_{k-1} và x_k và cũng lệch nhau về thời gian. Thay *G* bởi $G' = \{y_1, y_2, \ldots, y_{k-1}, x_k, y_{k+1}, \ldots, y_q\}$ thoả mãn ràng buộc sự lệnh nhau về thời gian của các yêu cầu.
  - Vậy G' cũng là một giải pháp tối ưu mà có số yêu cầu trùng với F nhiều hơn so với G.
  - Lặp lại bước trên, cuối cùng có được G'' chứa F mà |G''|=|G|
  - Nếu G'' có chứa yêu cầu không thuộc F (tức là các yêu cầu bắt đầu sau khi x_p kết thúc) thì yêu cầu đó đã phải được thêm vào F theo thuật toán tham lam
  - Vậy G'' = F, mà |G''| = |G|, nên F là giải pháp tối ưu

Thuê xe

- Thuật toán

```
thuexe(S)
begin
    n = length(S)
    // Sắp xếp các yêu cầu theo thời điểm kết thúc thuê tăng dần
    S = {a1, a2, …,an} với f(a1) ≤ f(a2) ≤ … ≤ f(an)
    F = {a1}
    i = 1
    for j from 2 to n do
        if (f(ai) ≤ s(aj)) then
            F = F ∪ {aj}
            i = j
        endif
    endfor
    return F
end
```

Thuê xe

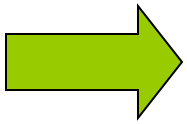
- Độ phức tạp của thuật toán
  - Sắp xếp các yêu cầu
    - O(nlogn)
  - Xây dựng giải pháp
    - O(n)

Tính chất của chiến lược tham lam

- Thuật toán tham lam
  - Xác định giải pháp sau một dãy liên tiếp các lựa chọn
  - Mỗi bước quyết định, lựa chọn dường như tốt nhất ở bước đó sẽ được chọn
  - Không luôn cho giải pháp tối ưu
- Một bài toán tối ưu bất kỳ có thể được giải quyết bởi thuật toán tham lam ?
  - Không luôn luôn đúng
- Tuy nhiên, nếu một bài toán có hai tính chất
  - Tính chất chọn lựa tham lam
  - Cấu trúc con tối ưu
- Thì bài toán có thể giải quyết được bởi thuật toán tham lam

Tính chất của chiến lược tham lam

- Tính chất chọn lựa tham lam (greedy choice property)
  - Luôn tồn tại một giải pháp tối ưu chứa một chọn lựa tham lam
    - Cần chỉ ra tồn tại một giải pháp tối ưu luôn bắt đầu bởi một chọn lựa tham lam
- Tính chất cấu trúc con tối ưu (optimal substructure)
  - Nếu *F* là một giải pháp tối ưu chứa một chọn lựa tham lam *c* thì *F-{c}* là giải pháp tối ưu cho bài toán con tương tự như bài toán đầu không chứa *c*
    - Giải pháp tối ưu của một bài toán *phải* chứa giải pháp tối ưu của bài toán con của nó



Chứng minh tính tối ưu của thuật toán tham lam

Tính chất của chiến lược tham lam

- Nếu một thuật toán tham lam thoả mãn hai tính chất
  - Tính chất chọn lựa tham lam
  - Tính chất cấu trúc con tối ưu
- Thì thuật toán tham lam cho giải pháp tối ưu

- Chứng minh
  - Theo tính chất chọn lựa tham lam, tồn tại giải pháp tối ưu F chứa một chọn lựa tham lam c_1. Theo tính chất cấu trúc con tối ưu, F-$\{c_1\}$ là giải pháp tối ưu của bài toán con không chứa c_1.
  - Áp dụng cho bài toán con không chứa c_1, theo tính chất chọn lựa tham lam, F-$\{c_1\}$ là giải pháp tối ưu chứa chọn lựa tham lam c_2. Theo tính chất cấu trúc con tối ưu, F-$\{c_1, c_2\}$ là giải pháp tối ưu cho bài toán con không chứa c_1 và c_2.
  - Tiếp tục lý giải như thế, cuối cùng chúng ta có
    $$F-\{c_1, c_2, \ldots, c_n\} = \varnothing$$
  - Hay: $F = \{c_1, c_2, \ldots, c_n\}$
  - Vậy giải pháp tối ưu F của bài toán ban đầu là một dãy các lựa chọn tham lam thực hiện bởi thuật toán tham lam

Chứng minh tính tối ưu thuật toán tham lam

- 2 cách
  - Chứng minh trực tiếp giải pháp của thuật toán là tối ưu
    - Nghĩa là không tồn tại giải pháp tối ưu khác tốt hơn
  - Chứng minh thuật toán thoả mãn hai tính chất
    - Tính chất chọn lựa tham lam
    - Tính chất cấu trúc con tối ưu

Đổi tiền

- Bài toán
  - Cho một hệ thống tiền tệ gồm các loại tờ giấy tiền có mệnh giá là 1, 5, 10, 20, 50. Cần đổi một số tiền S sao cho số tờ cần dùng ít nhất.
  - Ví dụ
    - 98 = 1 + 1 + 1 + 5 + 50 + 20 + 20
  - Thuật toán đơn giản
    - liệt kê tất cả các kết hợp có thể cho tổng số tiền là S
    - chọn kết hợp dùng ít số tờ nhất
    - Độ phức tạp hàm mũ !

Đổi tiền

- Thuật toán tham lam
  - Chọn lựa tham lam
    - ở mỗi bước, chọn tờ giấy tiền có mệnh giá cao nhất có thể mà không vượt quá tổng số tiền cần đổi
  - Ví dụ: S = 98

S=98	*S*=48	*S*=28	*S*=8	*S*=3	*S*=2	*S*=1
50	20	20	5	1	1	1

Đổi tiền

- Chứng minh thuật toán tham lam cho giải pháp tối ưu (1)
  - Giả sử F là giải pháp tối ưu
  - F phải thoả mãn các ràng buộc
    - F chỉ chứa nhiều nhất 4 tờ tiền mệnh giá 1
      - 5 tờ mệnh giá 1 = 1 tờ mệnh giá 5
    - F chỉ chứa nhiều nhất 1 tờ tiền mệnh giá 5
      - 2 tờ mệnh giá 5 = 1 tờ mệnh giá 10
    - F chỉ chứa nhiều nhất 1 tờ tiền mệnh giá 10
      - 2 tờ mệnh giá 10 = 1 tờ mệnh giá 20
    - Nếu F không chứa tờ tiền nào mệnh giá 10, thì chỉ chứa nhiều nhất 2 tờ mệnh giá 20
      - 3 tờ mệnh giá 20 = 1 tờ mệnh giá 50 + 1 tờ mệnh giá 10
    - Nếu F có chứa tờ tiền mệnh giá 10, thì chỉ chứa nhiều nhất 1 tờ mệnh giá 20
      - 2 tờ mệnh giá 20 + 1 tờ mệnh giá 10 = 1 tờ mệnh giá 50

Đổi tiền

- Chứng minh thuật toán tham lam cho giải pháp tối ưu (2)
  - Chỉ cần chỉ ra bài toán thoả mãn hai tính chất
  - **Tính chất chọn lựa tham lam**: cần chỉ ra luôn tồn tại giải pháp tối ưu bắt đầu bởi một chọn lựa tham lam
    - Nếu $S \geq 50$, thuật toán tham lam chọn lựa tờ tiền mệnh giá 50. Cần chứng minh rằng F phải bắt đầu bởi chọn lựa tờ tiền mệnh giá 50. Bằng phản chứng, giả sử F không chọn lựa tờ tiền mệnh giá 50. Vì F phải thoả mãn các ràng buộc trên, nên có thể chỉ có 2 khả năng: $4x1+5+10+20 < 50$ và $4x1+5+2x20 < 50$

      Vậy nếu $S \geq 50$, F phải chứa ít nhất một tờ tiền mệnh giá 50
    - Nếu $50 > S \geq 20$, lý giải tương tự, F phải chứa ít nhất một tờ tiền mệnh giá 20
    - Tiếp tục …

Đổi tiền

- Chứng minh thuật toán tham lam cho giải pháp tối ưu (3)
  - **Tính chất cấu trúc con tối ưu**
    - Giả sử F là giải pháp tối ưu cho tổng số tiền S, p là tờ tiền được chọn lựa cuối cùng bởi thuật toán tham lam. Cần chỉ ra rằng F-{p} là giải pháp tối ưu cho bài toán con S-p.
    - Chứng minh bằng phản chứng: giả sử tồn tại một giải pháp tối ưu tốt hơn F' cho bài toán con S-p. Khi đó, F' $\cup${p} là giải pháp tối ưu tốt hơn F cho bài toán S. Điều này mâu thuẩn giả thiết.

      Vậy F' $\cup${p}= F hay F' =F-{p}.

Đổi tiền

- Thuật toán tham lam

```
doitien(S)
begin
   F = ∅
   if (S ≥ 50) then
       F = F ∪ {(S div 50) tờ mệnh giá 50}
       S = S mod 50
   endif
   if (S ≥ 20) then
       F = F ∪ {(S div 20) tờ mệnh giá 20}
       S = S mod 20
   endif
   if (S ≥ 10) then
       F = F ∪ {(S div 10) tờ mệnh giá 10}
       S = S mod 10
   endif
   // tương tự cho các tờ tiền mệnh giá 5, 2, 1
   …
end
```

Đổi tiền

- Lưu ý
  - Thuật toán tham lam này không cho giải pháp tối ưu đối với mọi hệ thống tiền tệ
    - Chẳng hạn, thuật toán sẽ không cho giải pháp tối ưu đối với hệ thống tiền tệ {6, 4, 1}
    - Ví dụ
      - S = 8
      - Giải pháp cho bởi thuật toán tham lam: 6 + 1 + 1
      - Giải pháp tối ưu: 4 + 4

Xếp ba lô

- Bài toán
  - cho n đồ vật và một ba lô có trọng lượng tối đa W
  - mỗi đồ vật i có trọng lượng w_i
  - mỗi đồ vật i có giá trị v_i
  - gọi x_i là một phần của đồ vật i, $0 \leq x_i \leq 1$, x_i có trọng lượng $x_i w_i$ và giá trị $x_i v_i$
  - Yêu cầu: xếp các đồ vật vào ba lô để tổng giá trị ba lô lớn nhất
- Bài toán xếp ba lô này được gọi là *xếp ba lô « từng phần »*
  - có thể chỉ cần xếp vào ba lô một phần của đồ vật
- Bài toán xếp ba lô đã gặp được gọi là *xếp ba lô « 0-1 »*
  - một đồ vật hoặc được xếp vào ba lô (1) hoặc không được xếp vào ba lô (0)

Xếp ba lô

- Ý tưởng
  - Tập ứng cử viên là các đồ vật
  - Ở mỗi bước, chọn đồ vật *triển vọng nhất* và xếp vào ba lô một phần lớn nhất có thể của đồ vật này
    - đối với các đồ vật được chọn đầu tiên, xếp toàn bộ đồ vật vào ba lô
    - đối với đồ vật được chọn cuối cùng, có thể chỉ xếp một phần đồ vật vào ba lô
  - Thuật toán dừng khi ba lô đầy
  - Chọn đồ vật theo tiêu chí nào?
    - Giá trị giảm dần
    - Trọng lượng tăng dần
    - Tỷ lệ giá trị trên trọng lượng (v_i/w_i) giảm dần

Xếp ba lô: Ví dụ

Giá trị	*Trọng lượng*	*Giá trị / Trọng lượng*
8	2	4
10	5	2
15	3	5
36	4	9
42	6	7

Ba lô có thể chứa trọng lượng lớn nhất là 8.

Phép thử 1: Chọn đồ vật theo thứ tự giá trị giảm dần. Tổng giá trị các đồ vật được chọn là:

$$42 + 36 \times 2 / 4 = 60.$$

Phép thử 2: Chọn đồ vật theo thứ tự trọng lượng tăng dần. Tổng giá trị các đồ vật được chọn là:

$$8 + 15 + 36 \times 3 / 4 = 50.$$

Phép thử 3: Chọn đồ vật theo thứ tự tỷ lệ giá trị trên trọng lượng giảm dần. Tổng giá trị các đồ vật được chọn là:

$$36 + 42 \times 4 / 6 = 64.$$

Xếp ba lô

- Chọn lựa tham lam: Chọn đồ vật có tỷ lệ giá trị trên trọng lượng (vi/wi) giảm dần
- Thuật toán

```
xepbalotungphan()
begin
    sắp xếp các đồ vật theo tỷ lệ v_i/w_i giảm dần
    w = W
    i = 1
    while (w ≥ w_i) do // xếp toàn bộ đồ vật vào ba lô
        x_i = 1
        w = w − w_i
        i = i +1
    endwhile
    x_i = w/w_i // đồ vật cuối cùng được chọn để xếp vào ba lô
    for k from i + 1 to n do
        x_i = 0 // các đồ vật không được xếp vào bao lô
    endfor
    return (x_1, x_2, …,x_n) // giải pháp
end
```

Xếp ba lô

- Thuật toán
  - Phân tích độ phức tạp
    - Sắp xếp: O(nlogn)
    - Lặp: duyệt n đồ vật, vậy O(n)

Xếp ba lô

- Chứng minh tính tối ưu (1)
  - Cách 1: chứng minh trực tiếp (1)
    - Giả sử $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$
    - X = $\{x_1, x_2, \ldots x_n\}$ là giải pháp được xác định bởi thuật toán tham lam, V(X) là tổng giá trị
    - Giả sử j là chỉ số nhỏ nhất mà $x_j<1$, vậy $X=\{1,\ldots,1,x_j,0,\ldots,0\}$ (chỉ có đồ vật cuối cùng được chọn một số phần)
    - Bằng phản chứng, giả sử Y = $\{y_1, y_2, \ldots y_n\}$ giải pháp khác và V(Y) là tổng giá trị
    - Ta có

$$V(X)-V(Y)=\sum_{i=1}^{n} x_i v_i - \sum_{i=1}^{n} y_i v_i = \sum_{i=1}^{n} v_i (x_i - y_i)$$

$$= \sum_{i=1}^{n} (x_i - y_i) w_i \frac{v_i}{w_i}$$

Xếp ba lô

- Chứng minh tính tối ưu (2)
  - Cách 1: chứng minh trực tiếp (2)
    - Có ba trường hợp xảy ra

      - i<j, thì $x_i=1$, vậy $x_i-y_i\geq 0$ và $v_i/w_i\geq v_j/w_j$, suy ra

        $(x_i-y_i)v_i/w_i\geq(x_i-y_i)v_j/w_j$

      - i>j, thì $x_i=0$, vậy $x_i-y_i\leq 0$ và $v_i/w_i\leq v_j/w_j$, cũng suy ra

        $(x_i-y_i)v_i/w_i\geq(x_i-y_i)v_j/w_j$

      - i=j, thì $(x_i-y_i)v_i/w_i=(x_i-y_i)v_j/w_j$

      Vậy ta có:

      $$V(X)-V(Y)\geq \frac{v_j}{w_j}\sum_{i=1}^{n}(x_i-y_i)w_i=\frac{v_j}{w_j}\left(W-\sum_{i=1}^{n}y_iw_i\right)\geq 0 \ do\ \sum_{i=1}^{n}y_iw_i\leq W$$

      Như thế: $V(X) \geq V(Y)$

      Hay V(X) là giải pháp tối ưu

Xếp ba lô

- Chứng minh tính tối ưu (3)
  - Cách 2: chứng minh hai tính chất
    - **Tính chất chọn lựa tham lam**: cần chỉ ra rằng tồn tại giải pháp tối ưu chứa một chọn lựa tham lam
      - Giả sử k là đồ vật có tỉ lệ v_k/w_k lớn nhất, S là một giải pháp tối ưu, giá trị của S là V(S)
      - Bằng phản chứng, giả sử S không chứa k
      - Tổng giá trị của S là $V(S) = \sum_{i=1}^{n} x_i v_i$
      - Nếu một số phần của đồ vật k còn lại không được chọn, thì khi đó, với $j \in S$, $x_j \neq 0$, $j \neq k$, thay thế j trong S bởi k, với cùng trọng lượng $x_j w_j = x_k w_k$ (để không vượt trọng lượng ba lô), ta sẽ nhận được giải pháp S' tốt hơn S. Mâu thuẫn giả thiết.

$$x_j v_j \leq x_k v_k \quad chia\ cho \quad x_j w_j = x_k w_k \Rightarrow \frac{v_j}{w_j} \leq \frac{v_k}{w_k}$$

⇒ S phải chứa đồ vật k

Xếp ba lô

- Chứng minh tính tối ưu (4)
  - Cách 2: chứng minh hai tính chất
    - **Tính chất cấu trúc con tối ưu**: nếu S là giải pháp tối ưu chứa chọn lựa tham lam c thì tồn tại giải pháp S' tối ưu cho bài toán con không chứa c
      - Giả sử S là giải pháp tối ưu chứa đồ vật k có tỷ lệ v_k/w_k lớn nhất với trọng lượng lớn nhất có thể (p=max(W, w_k còn lại))
      - Khi đó S' =S-{k} là giải pháp cho bài toán con không chứa k với ba lô trọng lượng tối đa giảm p
      - Bằng phản chứng, giả sử S' không tối ưu. Khi đó tồn tại S'' tốt hơn S' là giải pháp tối ưu cho bài toán con không chứa k. Vậy, S''$\cup${k}, với k có trọng lượng p sẽ là giải pháp tốt hơn giải pháp S cho bài toán ban đầu. Mâu thuẫn giả thiết.
      - S' phải là giải pháp tối ưu cho bài toán con.

Xếp ba lô

- Lưu ý
  - Bài toán xếp ba lô từng phần được giải quyết bởi thuật toán tham lam
  - Bài toán xếp ba lô 0-1 không thể được giải quyết bởi thuật toán tham lam
    - Ngược lại, được giải quyết bởi thuật toán quy hoạch động

Mã Huffman

- Kỹ thuật hiệu quả trong nén dữ liệu
  - tiết kiệm từ 20 đến 90% không gian lưu trữ
- Ý tưởng
  - Xem dữ liệu là một dãy kí tự
  - Sử dụng tần suất xuất hiện của mỗi kí tự để xây dựng giải pháp tối ưu bằng cách biểu diễn mỗi kí tự bởi một xâu nhị phân
  - Thuật toán tham lam

Mã Huffman

- Ví dụ
  - Cần nén một tệp chứa 100.000 kí tự, tệp chỉ gồm các kí tự a, b, c, d, e, f. Tần suất xuất hiện của các kí tự trong tệp được xác định
  - Biểu diễn mỗi kí tự bằng xâu nhị phân có độ dài hằng số: *mã có độ dài hằng số* (fixed-length codes)
    - Cần 3 bít để biểu diễn 6 kí tự

Kí tự	a	b	c	d	e	f
Tần suất	45000	13000	12000	16000	9000	5000
Dãy bít	000	001	010	011	100	101

  - Số bít cần thiết để lưu trữ tệp: 300000
  - Tồn tại giải pháp tốt hơn ?

Mã Huffman

- Mỗi kí tự được biểu diễn bằng độ dài xâu nhị phân thay đổi được: *mã có độ dài thay đổi* (variable-length codes)
  - Kí tự có tần suất xuất hiện lớn được biểu diễn bởi xâu nhị phân có độ dài ngắn hơn

- Ví dụ

Kí tự	a	b	c	d	e	f
Tần suất	45000	13000	12000	16000	9000	5000
Dãy bít	0	101	100	111	1101	1100

- Số bít cần để lưu trữ tệp

  (45x1+13x3+12x3+16x4+9x4+5x4)x1000=224000 bít

- Chính là giải pháp tối ưu cho tệp dữ liệu này

Mã Huffman

- Mã tiền tố (prefix codes)
  - Không một mã nào là tiền tố của một mã khác
  - Quá trình giải mã trở nên đơn giản
    - Xác định mã của mỗi kí tự không nhập nhằng

- Ví dụ
  - Với bảng mã

Kí tự	a	b	c	d	e	f
Tần suất	45000	13000	12000	16000	9000	5000
Dãy bít	0	101	100	111	1101	1100

  - Dãy kí tự *abc* được mã hoá: *0101100*
  - Chuỗi nhị phân *001011101* được giải mã một cách duy nhất: *aabe*

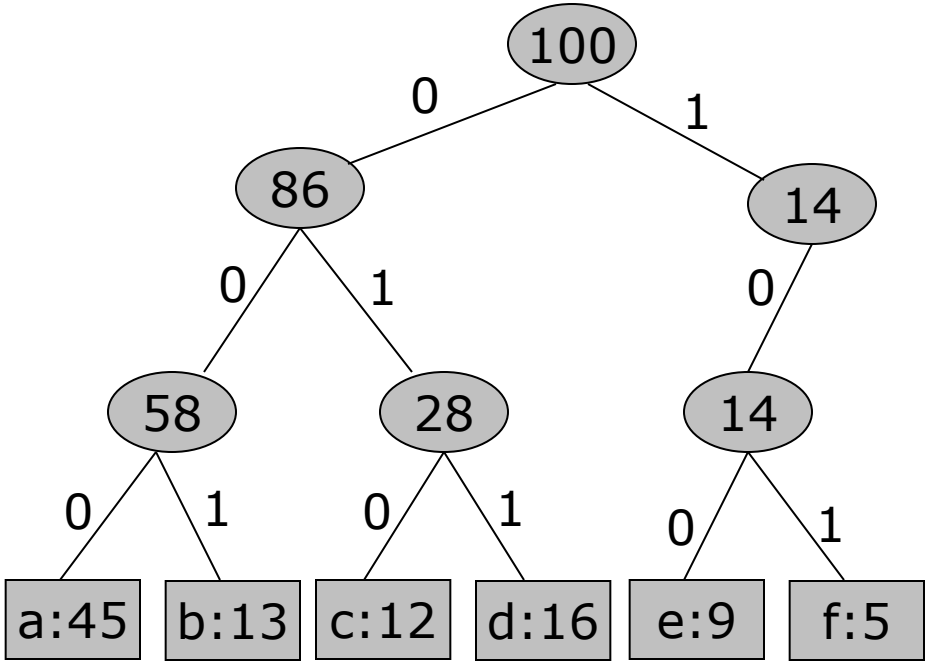
Mã Huffman

- Quá trình giải mã cần một **sự biểu diễn các mã tiền tố** để xác định mỗi mã có thể được xác định dễ dàng
  - **Cây nhị phân** với mỗi nút lá biểu diễn một kí tự
  - Mã nhị phân cho mỗi kí tự là đường đi từ nút gốc đến nút lá chứa kí tự đó
    - đọc 1 nghĩa là đi đến nút phải, đọc 0 đi đến nút trái
  - Cây nhị phân gồm
    - Mỗi nút lá được gán nhãn là một kí tự và tần suất xuất hiện của kí tự đó
    - Mỗi nút trong được gán nhãn là tổng tần suất xuất hiện của các nút lá của các cây con của nút trong đó

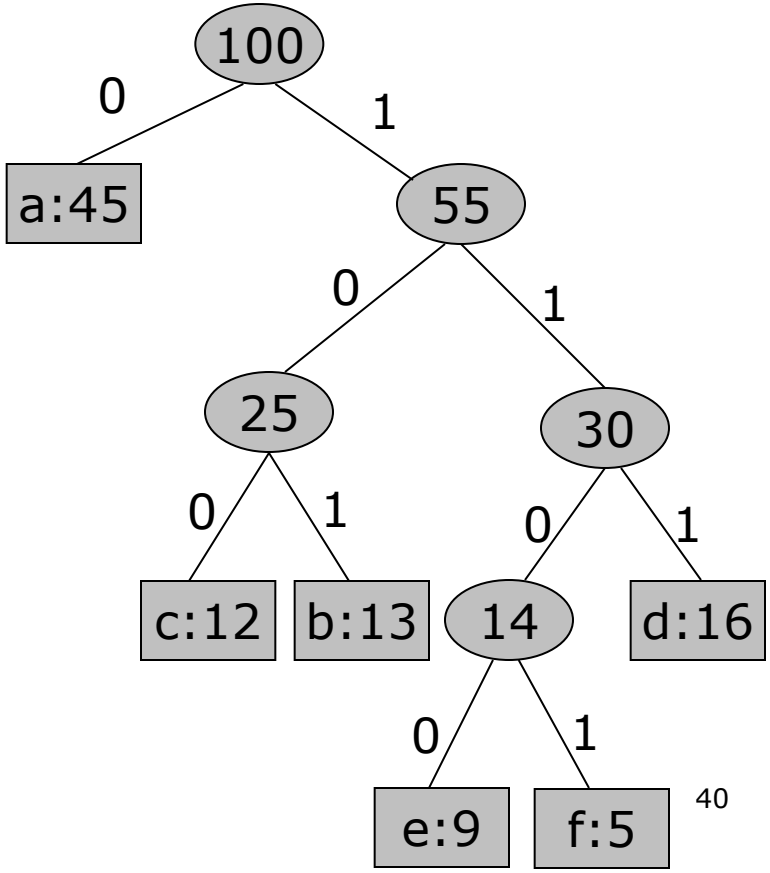
Mã Huffman

- Ví dụ cây nhị phân

Mã có độ dài hằng số



Mã có độ dài thay đổi



Mã Huffman

- Nhận xét
  - Mã có độ dài thay đổi, cũng là mã tối ưu, được biểu diễn bởi *cây nhị phân đầy đủ*
    - Cây nhị phân đầy đủ là cây mà mỗi nút trong có đúng hai nút con
  - Mã có độ dài cố định, là mã không tối ưu, được biểu diễn bởi cây nhị phân không đầy đủ

- Chỉ xét mã độ dài thay đổi, nghĩa là chỉ làm việc với cây nhị phân đầy đủ
  - Gọi C là bảng kí tự được sử dụng bởi các tệp dữ liệu
  - Số nút lá của cây nhị phân là |C|
    - Mỗi nút lá biểu diễn một kí tự
  - Số nút trong của cây nhị phân |C| - 1
    - Chứng minh

Mã Huffman

- Tính số bít cần để mã hoá tệp
  - Giả sử T là cây nhị phân biểu diễn các mã tiền tố
  - Mỗi kí tự c $\in$ C
    - f(c) là tần suất xuất hiện của kí tự c
    - d(c) là độ sâu của nút lá biểu diễn c, chính là số bít biểu diễn c
  - Số bít cần để mã hoá tệp là

$$B(T) = \sum_{c \in C} f(c)d(c)$$

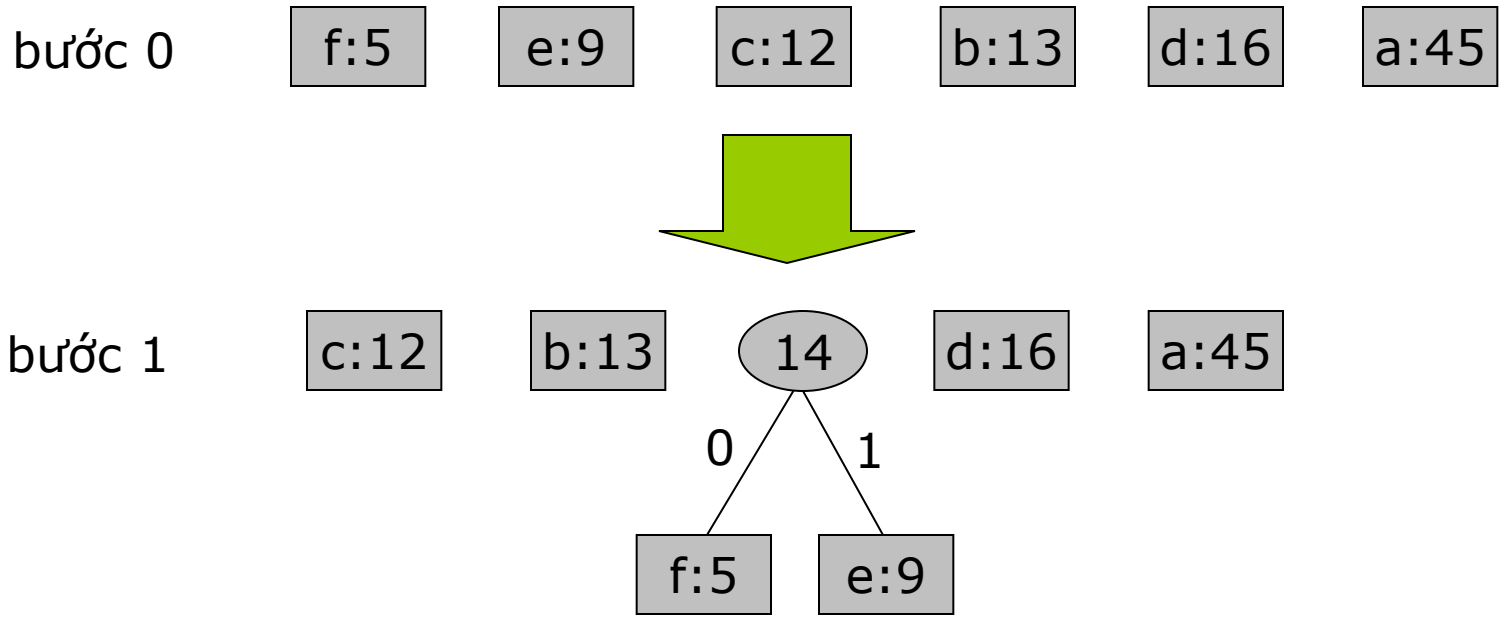
  - B(T) được gọi là chi phí của cây T

Mã Huffman

- Thuật toán tham lam phát minh bởi Huffman
  - Xây dựng cây nhị phân mã tiền tố tối ưu, hay còn được gọi là mã Huffman
  - Cây nhị phân sẽ được xây dựng từ dưới lên
    - Bắt đầu bởi |C| nút lá và thực hiện |C|-1 phép hoà nhập các cây con để tạo ra cây nhị phân cuối cùng
      - Tại sao |C| - 1 phép hoà nhập ?
    - Chọn lựa tham lam: ở mỗi bước, hai cây con có tần suất xuất hiện thấp nhất được chọn để hoà nhập
    - Khi hoà nhập hai cây con, một cây con mới được tạo ra với tần suất xuất hiện bằng tổng tần suất xuất hiện của hai cây con được hoà nhập

Mã Huffman

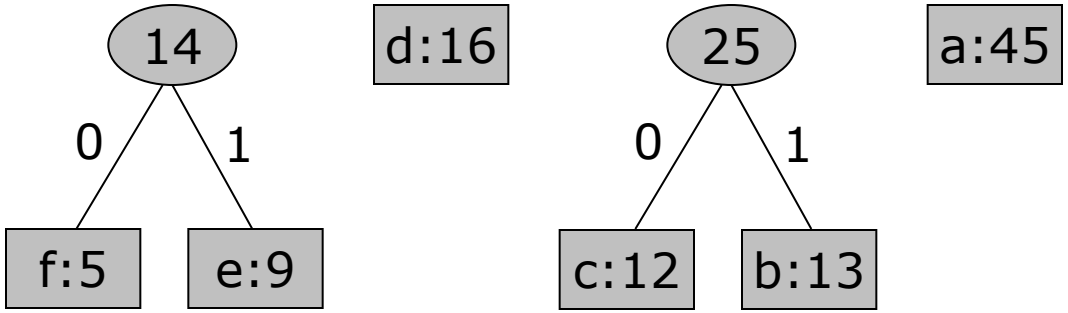
- Ví dụ (1)



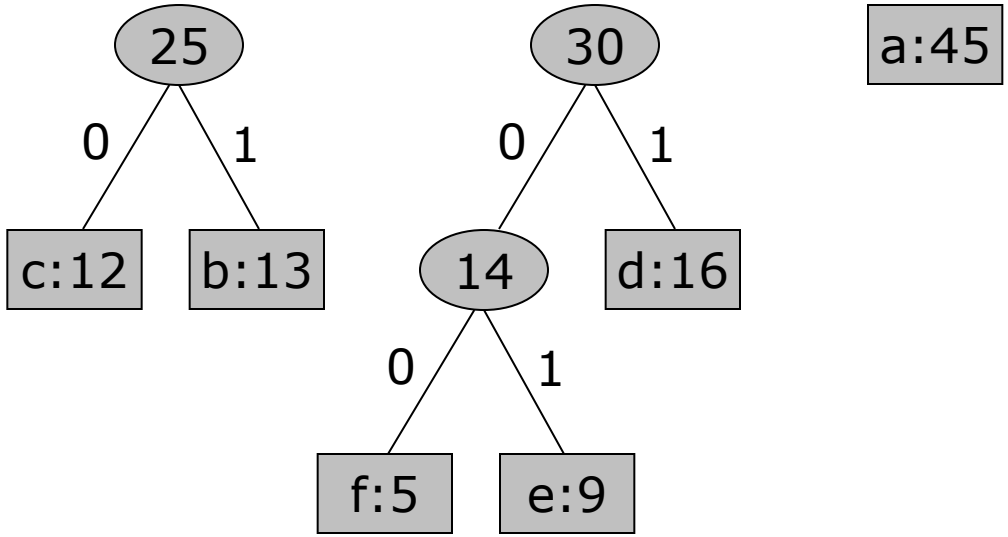
Mã Huffman

- Ví dụ (2)

bước 3

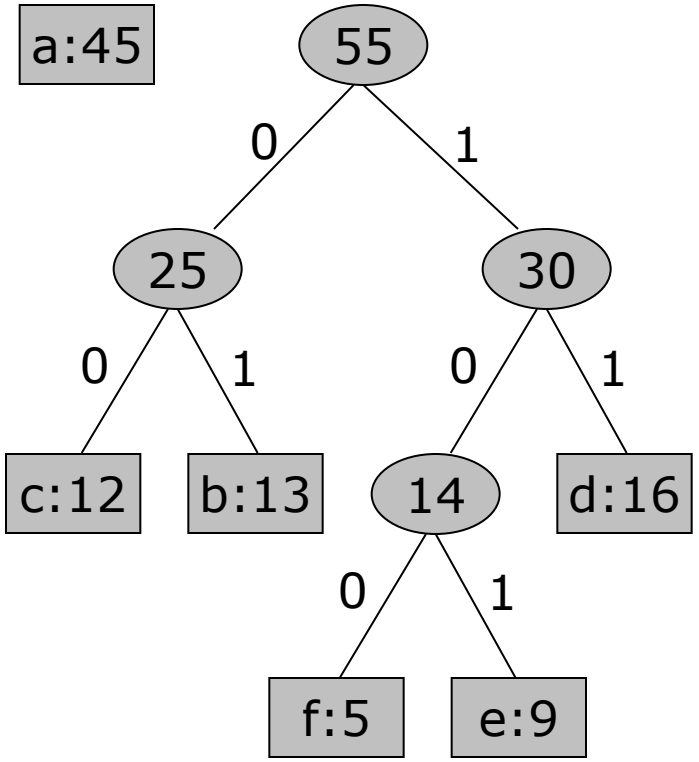


bước 4

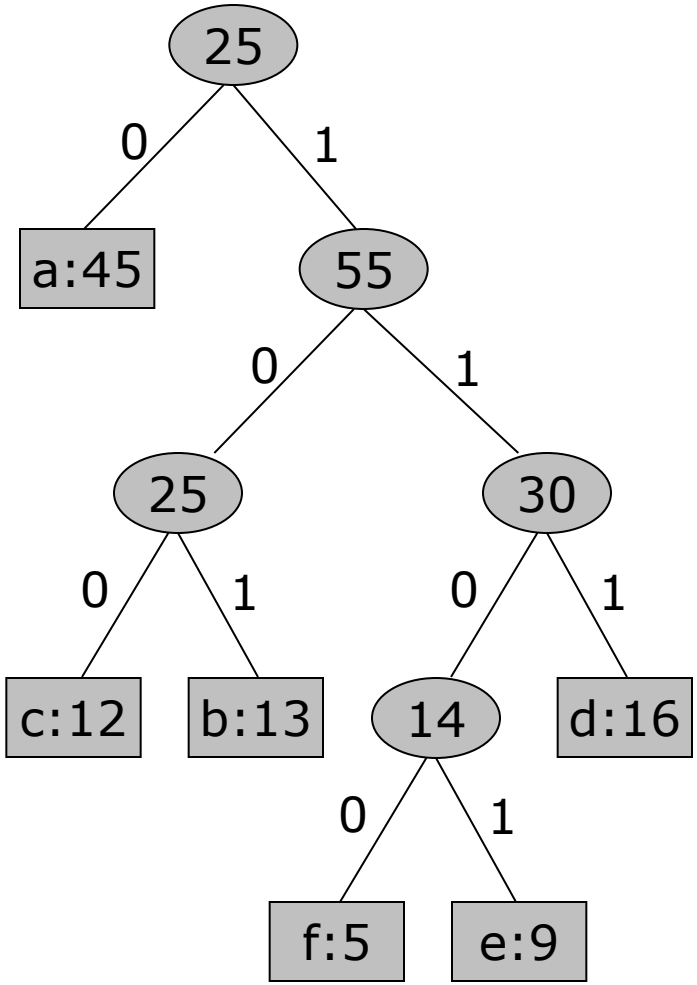


Mã Huffman

- Ví dụ (3)



bước 5



bước 6

Mã Huffman

- Thuật toán

```
huffman (C)
begin
   n = |C|
   L = C
   for i from 1 to n-1 do // thực hiện n-1 lần hoà nhập
       x = cây có tần xuất thấp nhất trong L
       L = L - {x}
       y = cây có tần xuất thấp nhất trong L
       L = L - {y}
       left(z) = x          // z có cây con trái là x
       right(z) = y         // z có cây con phải là y
       f(z) = f(x) + f(y)
       L = L ∪ {z}
   endfor
end
```

Từ cây nhị phân mã Huffman, làm thế nào để giải mã một chuỗi nhị phân ?

Mã Huffman

- Chứng minh sự đúng đắn của thuật toán
  - Chứng minh hai tính chất
    - Tính chất chọn lựa tham lam
    - Tính chất cấu trúc con tối ưu

Ứng dụng trong đồ thị

- Tìm cây khung nhỏ nhất
  - Thuật toán Kruskal
    - Luôn chọn cạnh ngắn nhất
  - Thuật toán Prim
    - Luôn chọn cạnh ngắn nhất giữa các đỉnh thuộc cây bao phủ tối thiểu và các đỉnh không thuộc cây bao phủ tối thiểu
- Tìm đường đi ngắn nhất
  - Thuật toán Dijkstra
    - Luôn chọn cạnh ngắn nhất nối một đỉnh đã đi qua đến một đỉnh chưa đi qua

Cây khung nhỏ nhất (minimum spanning tree)

- Định nghĩa
  - G=(V,E) là một *đồ thị* trong đó V là tập các *đỉnh* và E là tập các *cạnh* nối hai đỉnh
  - Một đồ thị gọi là *vô hướng* nếu các cạnh không định hướng (cạnh (u,v) ≡ cạnh (v,u))
  - Một đồ thị được gọi là *liên thông* nếu $\forall u,v \in N$ thì tồn tại đường đi nối u và v
  - Một *chu trình* là một đường đi từ đỉnh v đến đỉnh v có độ dài lớn hơn không và không có cạnh nào xuất hiện hai lần
  - Một *cây* là một đồ thị vô hướng, liên thông và không chứa chu trình
  - Tập hợp các cây rời nhau được gọi là *rừng*
  - *Cây khung* của đồ thị vô hướng G=(V,E) là một cây nối tất cả các đỉnh trong V
  - Cho một đồ thị vô hướng G=(V,E), mỗi cạnh $e=(u,v)\in E$ có một trọng số w(e). *Cây khung nhỏ nhất* T của đồ thị G là một cây khung với tổng trọng số các cạnh thuộc T là nhỏ nhất

Cây khung nhỏ nhất

- Ý tưởng
  - Xây dựng thuật toán tham lam xác định cây khung nhỏ nhất
    - Ở mỗi bước, cây khung nhỏ nhất được làm lớn lên bằng cách thêm vào một cạnh
    - Thuật toán thao tác trên một tập cạnh E, mà phải bảo đảm bất biến của vòng lặp như sau:
      - Trước mỗi lần lặp, A là tập con của cây khung nhỏ nhất
    - Ở mỗi bước, cạnh (u,v) được thêm vào A phải bảo đảm $A \cup \{(u,v)\}$ là tập con của cây khung nhỏ nhất
      - Cạnh (u,v) được gọi là ***cạnh hợp lệ***

Cây khung nhỏ nhất

- Thuật toán tham lam

```
caykhungnhonhat(G, w)
begin
    A = ∅
    while (A chưa tạo cây khung) do
        xác định cạnh (u,v) hợp lệ đối với A
        A = A ∪ {(u,v)}
    endwhile
    return (A)
end
```

- Dựa trên ý tưởng này, có hai thuật toán tìm cây khung nhỏ nhất: Kruskal và Prim
- Mỗi thuật toán sử dụng một luật riêng để xác định cạnh hợp lệ

Thuật toán Kruskal

- Tập A là **một rừng** (gồm nhiều cây)
- Ý tưởng
  - Từ một rừng hoà nhập các cây sao cho cuối cùng thu được cây khung nhỏ nhất
- Chọn lựa tham lam
  - Ở mỗi bước, xác định cạnh hợp lệ là cạnh (u,v) có trọng số nhỏ nhất nối hai cây trong A
- Thuật toán
  - Sắp xếp các cạnh trong E theo thứ tự giảm dần trong lượng
  - Ban đầu, A gồm |V| cây, mỗi cây chỉ chứa 1 đỉnh
  - Đối với mỗi cạnh trong E, chọn cạnh (u,v) có trọng số nhỏ nhất
  - Nếu u và v không thuộc cùng một cây trong A (để tránh tạo nên chu trình) thì (u,v) được thêm vào A và hoà nhập hai cây chứa u và v
  - Sau khi hoà nhập tất cả các cây thu được tập A chỉ còn chứa đúng 1 cây, chính la cây khung nhỏ nhất

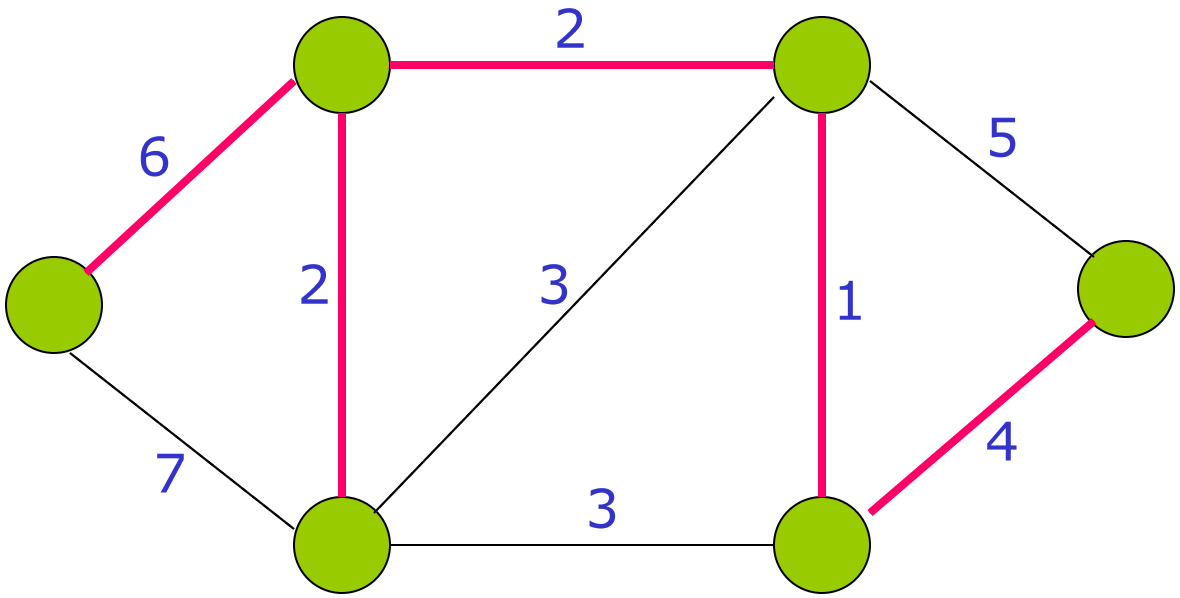
Thuật toán Kruskal

- Thuật toán

```
kruskal(V, E, w)
begin
    A = ∅
    sắp xếp các cạnh trong E theo thứ tự trọng số tăng dần
    for (mỗi cạnh trong E) do
        chọn cạnh (u,v) có trọng số nhỏ nhất
        if (u và v thuộc hai cây khác nhau) then
            // nối u và v không tạo chu trình
            A = A ∪ {(u,v)}
            hoà nhập hai cây chứa u và v
        endif
    endfor
    return (A)
end
```

Thuật toán Kruskal

- Ví dụ



Thuật toán Prim

- A là tập các cạnh tạo nên đúng **một cây**
- Ý tưởng
  - Từ một cây ban đầu chỉ là một đỉnh bất kỳ, thêm vào cây các cạnh sao cho cuối cùng thu được cây khung nhỏ nhất
- Chọn lựa tham lam
  - Ở mỗi bước, cạnh hợp lệ được thêm vào A là cạnh có trọng số thấp nhất nối cây và đỉnh không thuộc cây
- Thuật toán
  - Ban đầu, cây chỉ chứa một đỉnh r bất kỳ
  - Ở mỗi bước, chọn cạnh (u,v) - với u thuộc cây và v không thuộc cây - có trọng số nhỏ nhất
  - Thêm v vào cây và nối u và v
  - Cuối cùng, thu được cây khung nhỏ nhất

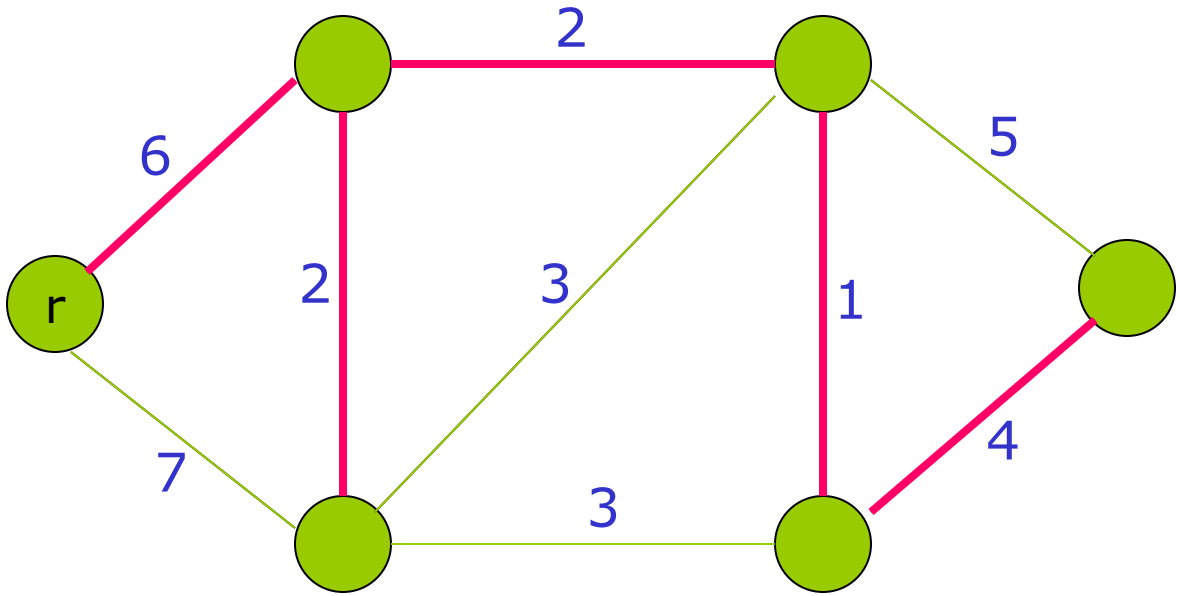
Thuật toán Prim

- Thuật toán

```
prim(V, E, w, r) // r là đỉnh bất kỳ
begin
    A = ∅
    V_A = {r}
    for (mỗi đỉnh w∈V mà (r,w)∈E) do
        L = L ∪ {r,w} // danh sách các đỉnh kề r
    endfor
    while (|V_A| < n) do // A chưa là cây khung
        chọn cạnh (u,v) có trọng số nhỏ nhất trong L, với u∈V_A
        if (v ∉ V_A) then  // thêm đỉnh v vào cây khung
            A = A ∪ {(u,v)}
            V_A = V_A ∪ {v}
            for (mỗi đỉnh w∈V mà (v,w)∈E) do
                L = L ∪ {v,w} // thêm vào L các đỉnh kề v
            endfor
        endif
    endwhile
    return (A)
end
```

Thuật toán Prim

- Ví dụ



Cây khung nhỏ nhất

- Độ phức tạp
  - Thuật toán Kruskal $O(|E|log(|E|))$
  - Thuật toán Prim $O(|V|^2)$
  - Tại sao ?

Đường đi ngắn nhất

- Cho đồ thị có hướng G=(V,E), E tập các cung
- Các cung có trọng số không âm
- Gọi s là đỉnh xuất phát
- Xác định đường đi ngắn nhất từ đỉnh s đến mỗi đỉnh còn lại

- Thuật toán Dijkstra

Thuật toán Dijkstra

- Ý tưởng
  - Gọi S là tập các đỉnh đã xác định được đường đi ngắn nhất từ s đến
    - mỗi $u \in S$ ta có $d[u]$ là độ dài đường đi ngắn nhất từ s đến u
    - mỗi $u \notin S$ ta có $d[u]$ là độ dài đường đi ngắn nhất từ s đến u qua các đỉnh thuộc S: $d[u] = min(d[x]+w[x,u], \forall x \in S)$
    - xuất phát, $S = \{s\}$
  - Chọn lựa tham lam
    - ở mỗi bước, thêm vào S đỉnh v sao cho đường đi từ s đến v qua các đỉnh thuộc S là ngắn nhất
    - mỗi khi thêm vào S đỉnh v thì phải tính lại độ dài đường đi ngắn nhất từ s đến các đỉnh còn lại trong V qua các đỉnh thuộc S
  - Cuối cùng, khi $S = V$ thì dừng lại

Thuật toán Dijkstra

- Thuật toán (1)

```
dijkstra(V, E, w[1..n,1..n])
begin
    S = {s}
    for (mỗi u ∈ V) do // khởi tạo đường đi ngắn nhất từ s
        d[u] = w[s,u]
    endfor
    for i from 1 to n-1 do // lặp n-1 lần, để thêm n-1 đỉnh vào S
        chọn v ∈ V-S mà có d[v] nhỏ nhất
        S = S ∪ {v}
        for (mỗi u ∈ V-S) do // tính lại đường đi ngắn nhất từ s đến u
            d[u] = min(d[u], d[v] + w[v,u])
        endfor
    endfor
    return (d)
end
```

Thuật toán Dijkstra

- Thuật toán (2)
  - Thuật toán mới chỉ tính độ dài của đường đi ngắn nhất
  - Cần lưu lại đường đi ngắn nhất
    - Dùng mảng *p*, với *p[v]* chứa đỉnh đứng trước *v* trong đường đi ngắn nhất từ *s* đến *v* qua các đỉnh trong *S*

Bài tập (1)

- Bài 1
  - Cho tập hợp A gồm n số nguyên, tìm tập hợp con S của A thoả mãn:
    (i) có đúng m phần tử ($m \leq n$)
    (ii) tổng các phần tử của S là lớn nhất
  1. Xây dựng thuật toán tham lam xác định S
  2. Chứng minh thuật toán cho giải pháp tối ưu

- Bài 2
  - Một người cắt tóc phục vụ n khách hàng. Mỗi khách hàng cần một thời gian phục vụ t_i khác nhau. Mỗi thời điểm người cắt tóc chỉ có thể phục vụ một khách hàng.
  1. Đề xuất thuật toán vét cạn
  2. Xây dựng thuật toán tham lam lập lịch phục vụ các khách hàng sao cho tổng thời gian chờ và được phục vụ của các khách hàng là nhỏ nhất
  3. Chứng minh thuật toán cho giải pháp tối ưu
  4. So sánh độ phức tạp của thuật toán tham lam và thuật toán vét cạn

Bài tập (2)

- Bài 3
  - Có n công việc, mỗi công việc cần 1 đơn vị thời gian để hoàn thành. Nếu mỗi công việc i bắt đầu trước hoặc tại thời điểm d_i thì sẽ mang lại lợi ích g_i.

    Xây dựng thuật toán tham lam lập lịch các công việc sao cho tổng lợi ích thu được là lớn nhất (lưu ý, phụ thuộc vào các thời điểm d_i không nhất thiết tất cả các công việc đều được lập lịch)

Giải thích 1

- Chứng minh
  - Định lý: một cây nhị phân đầy đủ có n nút lá thì có n-1 nút trong
  - Quy nạp
    - **Bước cơ sở**: đúng khi n = 1, có 0 nút trong
    - **Giả thiết**: định lý đúng với cây nhị phân đầy đủ có số nút lá nhỏ hơn n
    - **Suy dẫn**: Giả sử cây nhị phân T gồm nút gốc r và các cây con T_1, T_2 có n nút lá. Giả sử T_1 (tương ứng T_2) có n_1 (n_2) nút lá. Vậy $n = n_1+n_2$.

      Số nút trong của cây T bằng số nút trong của cây T_1, cây T_2 và thêm nút gốc r.

      Áp dụng giả thiết quy nạp cho cây T_1 và cây T_2, số nút trong của T_1 (tương ứng T_2) là n_1-1 (n_2-1).

      Vậy số nút trong của cây T là:

      (n1-1)+(n2-1) + 1 = n1+n2-1 = n-1