Thuật toán ứng dụng
Ôn thi TTUD

TS. Đinh Viết Sang, TA. Đặng Xuân Vương

Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 15 tháng 1 năm 2021

Mục lục

Mục lục

PHÉP LẶP VÀ HÀM ĐỆ QUY

- Phép lặp và hàm đệ quy được sử dụng (có thể thay thế nhau) để tạo ra vòng lặp của các tính toán.
- Hàm đệ quy (Recursive function):
  - Là hàm (hoặc tập các hàm) chứa lệnh gọi lại chính nó.
  - Không yêu cầu biết trước số lần lặp.
  - Có thể gây ra lỗi tràn stack nếu gọi đệ quy quá sâu.
- Phép lặp (Iteration):
  - Sử dụng for hoặc while để lặp lại các thao tác bên trong khối lệnh.
  - Có thể cần biết trước số lần lặp và cần đánh chỉ số cho từng bước lặp.
  - Không cần lo lắng về bộ nhớ stack.
- Có thể khử đệ quy hàm đệ quy thành vòng lặp, nhưng trong nhiều trường hợp sẽ cần tự xây dựng và quản lý stack.

CHIẾN LƯỢC GIẢI (MÔ HÌNH GIẢI THUẬT)

Chiến lược giải là hướng tư duy để đi đến việc đưa ra thuật toán.
Chiến lược giải không phải thuật toán.

- Tìm kiếm vét cạn (Exhaustive Search)
  - Cố gắng liệt kê mọi cấu hình và chọn ra cấu hình thỏa mãn yêu cầu.
  - Có thể thực thi bằng hàm đệ quy hoặc phương pháp lặp (xem lại bài toán cái túi).
  - Ưu điểm: Có thể giải hầu hết các bài toán.
  - Nhược điểm: Thời gian chạy rất lớn.
  - Tối ưu:
    - Thu hẹp không gian tìm kiếm: Loại bỏ đi những cấu hình chắc chắn không thỏa mãn yêu cầu.
    - Dừng thuật toán sớm: Dừng thuật toán khi tìm được một lời giải mà chắc chắn không có lời giải nào tốt hơn.

CHIẾN LƯỢC GIẢI (MÔ HÌNH GIẢI THUẬT)

- Tìm kiếm vét cạn (Exhaustive Search)
  - Phương pháp quay lui (backtracking):
    - Là một thuật toán vét cạn, áp dụng cho các bài toán tìm kiếm.
    - Ý tưởng là thử từng lựa chọn cho mỗi thành phần trong cấu hình lời giải, nếu không tìm được lựa chọn phù hợp, quay về thành phần trước và thử lựa chọn khác cho nó (quay lui).
    - Thực thi bằng hàm đệ quy.
    - Tối ưu bằng nhánh và cận: Ước lượng lời giải tốt nhất có thể thu được trên nhánh hiện tại và so sánh với lời giải tốt nhất đã tìm được để quyết định có nên cắt nhánh hay không.

CHIẾN LƯỢC GIẢI (MÔ HÌNH GIẢI THUẬT)

- Chia để trị (Divide and Conquer)
  - Chia bài toán thành các bài toán con độc lập.
  - Giải các bài toán con độc lập.
  - Kết hợp kết quả
  - Thực thi bằng hàm đệ quy.
  - Có thể song song hóa (giải các bài toán con đồng thời).
- Quy hoạch động (Dynamic Programming)
  - Giải lần lượt các bài toán con gối nhau (kích thước nhỏ đến lớn).

CHIẾN LƯỢC GIẢI (MÔ HÌNH GIẢI THUẬT)

- Các bước thực hiện thuật toán QHĐ:
  - Xác định bài toán con:
    - Các bài toán con thường là một mảng (mỗi bài toán con là một phần tử của mảng).
    - Số chiều và vai trò mỗi chiều phụ thuộc vào các items và ràng buộc của bài toán.
    - Thông thường, chiều thứ nhất là chiều của items, các chiều còn lại là các ràng buộc của bài toán.
    - Bài DRONE: mỗi nhà kho là một item, chiều items là số lượng nhà kho được xét kể từ nhà kho đầu tiên, chiều thứ 2 là thời gian đã sử dụng (liên quan đến ràng buộc về thời gian).
    - Nếu có nhiều nhóm items, có thể có nhiều chiều items:
    - Bài dãy con chung dài nhất: Chiều thứ nhất là chiều xâu con của dãy 1, chiều thứ 2 là chiều xâu con của dãy 2.
  - Xác định trường hợp cơ sở.
  - Tìm công thức truy hồi.
- Thực thi bằng hàm đệ quy hoặc vòng lặp.
- Không thể song song hóa (bắt buộc giải tuần tự các bài toán con).

CHIẾN LƯỢC GIẢI (MÔ HÌNH GIẢI THUẬT)

- Tham lam (Greedy)
  - Ý tưởng là tại mỗi bước, ta chỉ tìm lời giải tối ưu cục bộ (ví dụ như tối ưu trên một tập con các items, hoặc thỏa mãn một tập con các ràng buộc của đề bài).
  - Mỗi bài toán thường có rất nhiều thuật toán tham lam.
  - Bài toán cái túi: luôn chọn vật có giá trị lớn nhất, luôn chọn vật có giá trị nhỏ nhất hay luôn chọn vật có giá trị trung bình lớn nhất.
  - Nhược điểm: Các thuật toán tham lam thường không cho lời gian tối ưu.
  - Ưu điểm: Thuật toán tham lam thường đơn giản (cả về ý tưởng, cách thực thi lẫn độ phức tạp thời gian, bộ nhớ) và cũng cho lời giải với độ chính xác chấp nhận được. Thuật toán tham lam thường được dùng để giải các bài toán mà các mô hình thông thường không thể chạy trong thời gian chấp nhận được.
  - Fact: Thuật toán Dijkstra (tìm đường đi ngắn nhất) Prim và Kruskal (tìm cây khung nhỏ nhất) là 3 thuật toán xây dựng trên chiến lược tham lam.
  - Thuộc nhóm chiến lược Heuristic.

CHIẾN LƯỢC GIẢI (MÔ HÌNH GIẢI THUẬT)

- Một vài chiến lược khác (tham khảo):
  - Giảm để trị (decrease and conquer): Giảm bài toán lớn về một bài toán con, là trường hợp đặc biệt của chia để trị. VD: thuật toán tìm kiếm nhị phân (do mỗi lần chia đôi dãy, ta chỉ tiếp tục giải quyết bài toán trên một trong 2 dãy con).
  - Biến đổi để trị (transform and conquer): Biến đổi dữ liệu đầu vào để đưa bài toán về dạng dễ giải quyết hơn (ví dụ: sắp xếp lại mảng, sử dụng cấu trúc dữ liệu khác để lưu trữ hay như bài ICBUS, ta biến đổi đồ thị ban đầu G -> đồ thị G' và giải bài toán trên G') hoặc biến đổi yêu cầu đề bài thành một dạng quen thuộc (như bài BUS và bài TAXI, bài TAXI có thể quy dẫn về bài BUS hoặc TSP).
  - Nhóm Heuristic (dựa trên kinh nghiệm): chiến lược này thường đưa ra thuật toán dựa vào suy luận không đầy đủ kết hợp với phán đoán. Tham lam là một chiến lược thuộc nhóm này.

KINH NGHIỆM LÀM BÀI

Anh sẽ bỏ qua những bước như đọc đề hay debug nhé =))) Các bước còn lại như sau:

- Đầu tiên cần quan sát bài toán và đưa ra các nhận xét. Các bước sau đây không nhất thiết phải thực hiện theo thứ tự, mỗi bước có thể được lặp lại nhiều lần:
  - Có thể chia bài toán thành các bài toán con không?
  - Có thể biến đổi bài toán thành các bài quen thuộc không?
  - Có thể biến đổi dữ liệu để bài toán dễ giải hơn không?
  - Nhận xét các tính chất đặc biệt của bài toán.

KINH NGHIỆM LÀM BÀI

- Xác định chiến lược giải (nếu bài toán được chia thành nhiều bài toán con, có thể xác định chiến lược cho từng bài toán con khác nhau):
  - Nếu là bài toán kinh điển, ta áp dụng một thuật toán kinh điển để giải quyết nó (ví dụ như bài toán sắp xếp dùng quick sort, merge sort, bài toán tìm đường đi ngắn nhất dùng dijkstra,...).
  - Dựa vào kích thước bài toán, xác định xem có thể sử dụng vét cạn không, nếu dùng vét cạn thì có cần thu hẹp không gian tìm kiếm (ví dụ: nhánh và cận) hay không.
  - Liệu có thể sử dụng chia để trị hay quy hoạch động hay không?
  - Nếu các mô hình trên không áp dụng được, ta thử nghĩ đến tham lam.
  - Đôi khi, ta có thể phán đoán được thuật toán trước rồi mới tìm cách chứng minh thuật toán đó đúng sau (ví dụ bài CHANGE, các thuật toán qhđ,...)
  - Có những thuật toán được đưa ra từ việc kết hợp nhiều chiến lược (ví dụ bài FIBWORDS: chia để trị + vét cạn).

KINH NGHIỆM LÀM BÀI

- Xác định từng module cần thực thi.
- Với mỗi module, ta làm các công việc sau:
  - Xác định cách cài đặt: các bước thực hiện, phép lặp hay hàm đệ quy,...
  - Tối ưu: Bước tối ưu module thường dựa trên việc thay đổi cấu trúc dữ liệu (ví dụ như thuật toán dijskstra sử dụng heap hay bài GOLD MINING sử dụng deque) để giảm thời gian thực hiện thuật toán. Đôi khi là tìm một thuật toán hoàn toàn khác (như bài WATERJUG sử dụng thuật toán Euclid). Đôi khi, việc tối ưu từng module không thể giúp các em giải quyết trọn vẹn bài toán, nguyên nhân có thể là do các em xác định sai chiến lược giải và thuật toán cha, lúc này các em cần xem lại chiến lược giải bài và thuật toán cha xem có thể thay đổi để tối ưu hay không. (Thuật toán cha là thuật toán giải quyết toàn bộ bài toán, thuật toán con là thuật toán của module).

CÁC DẠNG BÀI TẬP

- Ad hoc: Là dạng bài tập mà gần như các em chỉ cần làm theo đề bài mô tả. Ví dụ như bài BIRD (có N con chim, bắn lần lượt các con chim có mức năng lượng từ cao đến thấp, mỗi lần bắn tốn năng lượng bằng tổng sức mạnh của con chim bị bắn + con chim bên trái gần nó nhất + con chim bên phải gần nó nhất (còn sống)).
- Vét cạn: Là những bài có kích thước nhỏ. Các em cần xác định xem liệu có cần đặt thêm nhánh cận để được điểm tối đa hay không. Để giải các bài này, việc code được hàm đệ quy và biết cách đặt cận, kiểm tra điều kiện dừng là điều gần như bắt buộc.
- Chia để trị: Với các bài chia để trị, các em cần xác định được cách chia bài toán cũng như cách kết hợp lời giải, đây cũng chính là dấu hiệu của 1 bài chia để trị. Việc code được hàm đệ quy cũng là điều gần như bắt buộc. Thuật toán tìm kiếm nhị phân là một thuật toán mà các em cần nhớ.

CÁC DẠNG BÀI TẬP

- Quy hoạch động: Anh đã mô tả khá kỹ cách làm những bài này ở bước trên. Nếu áp dụng được các bước đó với bài toán nào, bài toán đó ắt hẳn là bài toán quy hoạch động =))) Với bài toán QHĐ, các em có thể code bằng hàm đệ quy hoặc vòng lặp đều ổn cả.
- Tham lam: Thường thì có khá ít bài tham lam, nhưng nếu có, hãy phán đoán thuật toán trước, rồi tìm cách chứng minh sau (hoặc cứ code rồi submit nếu cảm thấy tự tin là thuật toán của mình đúng).
- Đồ thị: Các em cần thành thạo các cách biểu diễn đồ thị (ma trận kề, ma trận trọng số, danh sách cạnh, danh sách kề) -> BFS, DFS -> Dijkstra (hoặc Floyd, Bellman-Ford) cho bài toán tìm đường đi ngắn nhất, Prim hoặc Kruskal cho bài tìm cây khung nhỏ nhất -> Thành phần liên thông mạnh (liên quan đến DAG, sắp xếp Topo hoặc dùng Tarjan), tìm khớp, cầu của đồ thị (Tarjan). Nhớ là các thuật toán trên đồ thị hầu hết vẫn áp dụng các chiến lược giải bài ở trên.

CÁC DẠNG BÀI TẬP

Mấy dạng bài tập sau chắc sẽ không có trong đề thi, nhưng anh cứ viết cho chắc:

- Toán học: Liên quan đến các tính chất toán học như bài WATERJUG (sử dụng thuật toán Euclid).
- Xử lý xâu: Thuật toán tìm xâu chung dài nhất (quy hoạch động), KMP để tìm kiếm xâu,...
- Hình học: Bao lồi. Có thể phải xử lý sai số số thực, cẩn thận :v
- Bài toán lạ: Cố gắng đưa về bài toán quen xem :v Nếu không làm thế được, hãy cố quan sát và nhận xét những điểm đặc biệt của nó.

Mục lục

CONTAINER

- Cách 1: Xét từng hoán vị, với mỗi hoán vị, xếp lần lượt từ trong ra ngoài, từ trái sang phải (khi xếp đầy hàng trên cùng thì chuyển xuống hàng dưới).
- Cách 2: Với mỗi item, thử từng vị trí (tương ứng với ô trái trên). Có thể sắp xếp các item theo kích thước (có thể ưu tiên diện tích, chiều dài hoặc chiều rộng) trước rồi mới xếp vào container.
- Nhìn chung, cả 2 thuật toán trên đều không tốt lắm, nhưng anh nghĩ là test bài này cũng không quá chặt, vì vậy mọi người vẫn có thể được full điểm với 1 trong 2 cách làm trên.

Mục lục

Kruskal vs Prim

Xin lỗi mọi người, có vẻ anh nhớ nhầm về 1 bài toán có thể giải bằng Kruskal mà không thể giải bằng Prim, bài đó không có trong chương trình của mình nhé.

Tuy nhiên, vẫn có 1 số ưu nhược điểm của 2 thuật toán này:

- Thuật toán Prim chỉ chạy được trên đồ thị liên thông, thuật toán Kruskal có thể chạy trên đồ thị không liên thông (sẽ sinh ra rừng có trọng số nhỏ nhất).
- Thuật toán Prim nếu cài với Heap mới có thể chạy nhanh bằng thuật toán Kruskal.
- Thuật toán Prim chạy nhanh hơn trên đồ thị dày (tỉ lệ cạnh / bình phương số đỉnh lớn), thuật toán Kruskal chạy nhanh hơn trên đồ thị thưa.

Bài toán mà anh bị nhầm là bài "COUNT SPANNING TREES". Bài này không liên quan gì đến bài cây khung nhỏ nhất, nhưng để giải được thì mình cần dùng cấu trúc Disjoint Set (cấu trúc này cũng được sử dụng trong thuật toán Kruskal - đây là lý do làm anh nhớ nhầm :v). Mọi người có thể tham khảo code của anh ở đây.

Mục lục

ADDEDEGE

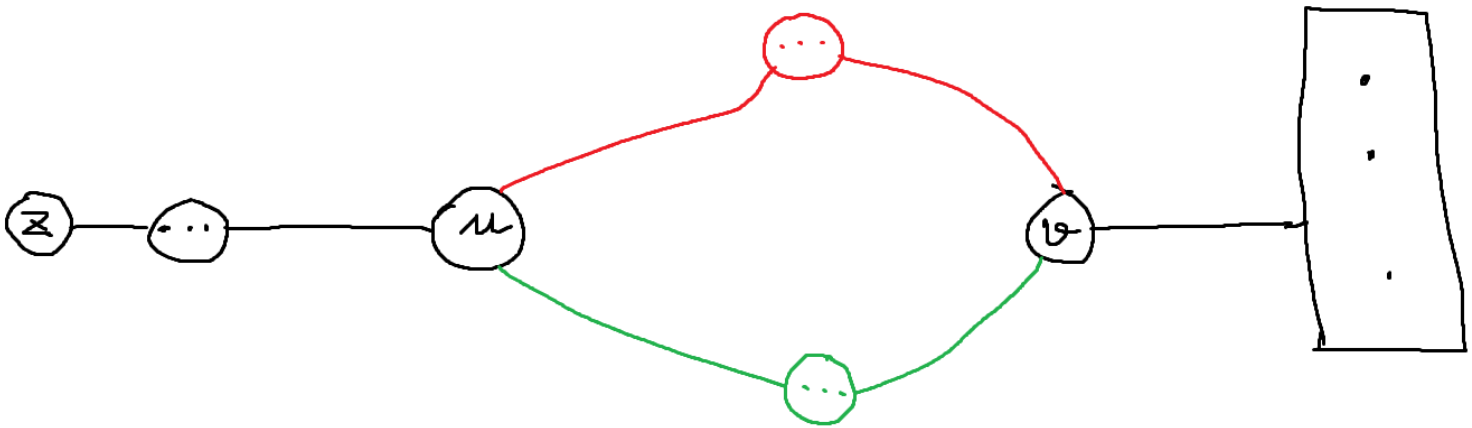
- Cho một đồ thị.
- Chu trình đơn là chu trình mà mỗi cạnh và mỗi đỉnh chỉ xuất hiện tối đa 1 lần (trừ đỉnh bắt đầu và đỉnh kết thúc).
- Đếm số cặp đỉnh mà khi thêm cạnh nối giữa 2 đỉnh đó, đồ thị có thêm đúng 1 chu trình đơn.

Nhận xét 1

- Nếu thêm 1 cạnh nối giữa 2 thành phần liên thông, ta không thu được bất kỳ chu trình đơn nào.
- Vậy từ giờ, ta sẽ chỉ xét từng thành phần liên thông của đồ thị.

Nhận xét 2

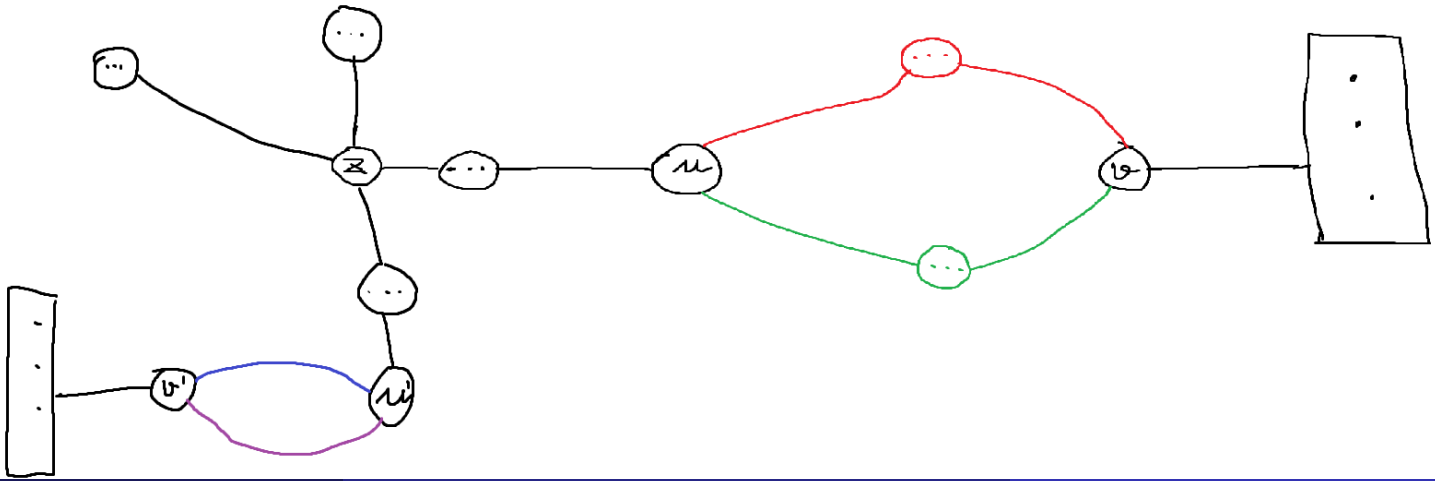
- Gọi u và v là 2 đỉnh thuộc cùng một chu trình.
- Gọi z là một điểm bất kỳ mà để đi từ z đến v, buộc phải đi qua u.
- Nhận xét: Với đỉnh w bất kỳ mà để đi được từ z đến w, buộc phải đi qua v, khi thêm cạnh (z, w), ta nhận được ít nhất 2 chu trình đơn:
  - $z- > w- > v- > REDPATH- > u- > z$
  - $z- > w- > v- > GREENPATH- > u- > z$



Hình 1: ADDEDGE1

Nhận xét 3

- Xét các cặp u' và v' tương tự đối với z, ta đều thấy tính chất tương tự. Việc thêm cạnh nối từ z đến các đỉnh nằm phía sau u hay u' là không thỏa mãn đề bài. Ta nói z bị chặn bởi chu trình đi qua u, chu trình đi qua u',... Có một tập các đỉnh bị chặn bởi cùng một tập các chu trình như vậy, ta gọi tập này là một vùng.
- Nói cách khác, ta thấy các chu trình chia cắt đồ thị ra thành nhiều vùng, ta không thể tạo cạnh nối giữa các vùng này, hoặc cạnh nối từ một đỉnh trong vùng này với một đỉnh thuộc chu trình. Vậy, ta chỉ còn có thể xem xét việc thêm cạnh giữa các đỉnh thuộc cùng chu trình.



Nhận xét 4

- Mỗi vùng là một cây. Dễ dàng chứng minh bằng phản chứng: nếu 1 vùng có chứa chu trình, vậy chu trình đó sẽ chia vùng đó ra làm 2.
- Nếu 2 đỉnh z_1 và z_2 thuộc cùng 1 vùng, việc thêm cạnh (z_1, z_2) sẽ tạo ra thêm đúng một chu trình đơn.
- Vậy, với một vùng có K đỉnh, $deg(z_i)$ là bậc của z_i, số cặp đỉnh z_1 và z_2 thỏa mãn điều kiện trên là:

$$\frac{K \times (K-1) - \sum_{i=1}^{K} deg(z_i)}{2} \tag{1}$$

- Do vùng là cây, nên tổng bậc bằng $2 \times (K-1)$, vậy, với mỗi vùng, số cạnh có thể thêm là:

$$\frac{K \times (K-1) - 2 \times (K-1)}{2} = \frac{(K-1) \times (K-2)}{2} \tag{2}$$

Thuật toán

- Xác định thành phần liên thông của đồ thị.
- Với mỗi thành phần liên thông, tìm các chu trình và chia đồ thị thành các vùng.
- Với mỗi vùng, tính số cạnh có thể thêm theo công thức trên.
- Kết quả của bài toán là tổng số cạnh thêm của tất cả các vùng.

Nhận xét 5

- Việc xác định các chu trình là tương đối khó khăn
- Mục đích chính là xác định các vùng
- Mỗi vùng được tạo thành từ các cầu của đồ thị, các vùng được nối với nhau bởi các cạnh của chu trình
- Vậy chỉ cần xóa bỏ đi các cạnh thuộc chu trình thì ta sẽ thu được các vùng
- Để tìm các cạnh của chu trình, ta tìm tất cả cầu của đồ thị, những cạnh không phải cầu là những cạnh thuộc chu trình
- Thuật toán Tarjan để tìm cầu

Thuật toán ứng dụng
Ôn thi TTUD

TS. Đinh Viết Sang, TA. Đặng Xuân Vương

Trường Đại học Bách khoa Hà Nội
Viện Công nghệ thông tin và Truyền thông

Ngày 15 tháng 1 năm 2021