SIROCCO CUP 2026 / CT2

Chuyên - Đề 2

Thứ Tư, 04/11/2026 · Online

180 phút · 4 bài · 400 điểm

Đồ thị, dữ liệu trên cây và tối ưu DP

Bài	Tên bài	Tag
A	Chia domino	Đồ thị hai phía, Chu trình, Mô hình hóa
B	Cây khung OR nhỏ nhất	Bitmask, DSU, Tham lam
C	Màu nổi trội	Cây, DSU on tree, Đếm tần suất
D	Chia đoạn tối ưu	Quy hoạch động, Chia để trị, Cửa sổ tần suất

Problemset trực tuyến
https://codeforces.com/gym/729110/problems

CT2-A / KHÁM PHÁ LỜI GIẢI

A. Chia domino

Đồ thị hai phía | Chu trình | Mô hình hóa

ĐIỂM MẤU CHỐT

Các số là đỉnh, domino là cạnh; trong mỗi nhóm, không có hai cạnh chung đầu mút.

Từ quan sát đến thuật toán

Nếu a[i]=b[i], trả NO vì một nhóm sẽ lặp số ngay trên quân đó. Nếu một số xuất hiện hơn hai lần, cũng không thể chia vào hai nhóm. Có 2n lần xuất hiện và chỉ n giá trị, nên trong trường hợp không bị loại, mỗi giá trị xuất hiện đúng hai lần.

Đồ thị trên các giá trị vì thế có bậc 2 ở mọi đỉnh, có thể chứa hai cạnh song song. Mỗi thành phần là một chu trình. Để hai cạnh kề khác nhóm, phải tô cạnh xen kẽ quanh chu trình; điều này làm được khi và chỉ khi chu trình chẵn. Kiểm tra đồ thị hai phía bằng BFS là đủ trong đồ thị bậc 2 này.

Kiểm tra bằng ví dụ

Test đầu: nhóm một gồm quân 1,3; nhóm hai gồm quân 2,4. Test sau, mỗi quân đã lặp số ngay trong bản thân nó.

CT2-A / CHỨNG MINH VÀ CÀI ĐẶT

Chia domino

Vì sao thuật toán đúng?

Mỗi số phải xuất hiện nhiều nhất một lần ở mỗi nhóm, nên điều kiện không khuyên và bậc ≤2 là cần. Vì tổng bậc bằng 2n, tất cả bậc buộc bằng 2. Trong một chu trình lẻ, tô cạnh xen kẽ sẽ làm cạnh cuối và đầu cùng nhóm tại một đỉnh, nên không thể. Chu trình chẵn tô xen kẽ được; thực hiện độc lập cho các thành phần sẽ phân hết domino mà không lặp số. Trong đồ thị bậc 2, không có chu trình lẻ tương đương hai phía.

Mã giả

```
ok=true; degree[1..n]=0; adj empty
for each (u,v):
    if u=v: ok=false
    degree[u]++; degree[v]++
    adj[u].push(v); adj[v].push(u)
if any degree[x]>2: ok=false
color[1..n]=-1
for each uncolored vertex s:
    color[s]=0; queue=[s]
    while queue not empty:
        u=pop_front(queue)
        for v in adj[u]:
            if color[v]=-1:
                color[v]=color[u] xor 1; push(v)
            else if color[v]=color[u]: ok=false
print YES if ok else NO
```

Độ phức tạp

O(n) thời gian và O(n) bộ nhớ mỗi test.

Các lỗi dễ gặp

- Kiểm tra đồ thị hai phía đơn thuần là chưa đủ nếu bỏ qua điều kiện bậc.
- Hai quân giống nhau có thể hợp lệ: hai cạnh song song tạo chu trình chẵn dài 2.
- Không cần in cách chia; đầu ra chỉ YES hoặc NO.

CT2-A / LỜI GIẢI TỪNG SUBTASK

Chia domino

Subtask	Điểm	Giới hạn bổ sung
1	30	t ≤ 5 và n ≤ 18.
2	70	Không có giới hạn bổ sung.

Subtask 1 - 30 điểm

Duyệt 2^n cách gán quân vào hai nhóm. Với mỗi cách, đánh dấu các giá trị trong từng nhóm và loại ngay nếu lặp.

Độ phức tạp: O(Σn×2^n) thời gian, O(n) bộ nhớ.

Subtask 2 - 70 điểm

Loại khuyên, loại số xuất hiện trên hai lần; BFS tô hai màu đồ thị trên các giá trị để loại chu trình lẻ.

Độ phức tạp: O(Σn) thời gian, O(n) bộ nhớ.

CT2-B / KHÁM PHÁ LỜI GIẢI

B. Cây khung OR nhỏ nhất

Bitmask | DSU | Tham lam

ĐIỂM MẤU CHỐT

Muốn một bit của OR bằng 0 thì tất cả cạnh được chọn đều phải có bit đó bằng 0.

Từ quan sát đến thuật toán

Bắt đầu bằng mask có cả 30 bit thấp bằng 1. Một cạnh w được phép dùng nếu (w & mask)=w. Khi các cạnh được phép còn liên thông, chúng luôn chứa một cây khung.

Xét bit từ 29 xuống 0. Thử xóa bit đó khỏi mask và dùng DSU kiểm tra liên thông chỉ với cạnh còn được phép. Nếu liên thông, giữ quyết định xóa; nếu không, khôi phục bit. Tránh chỉ kiểm tra riêng bit đang xét: phải giữ mọi bit 0 đã quyết định trước đó.

Kiểm tra bằng ví dụ

Chọn hai cạnh trọng số 1 và 2 được OR bằng 3. Hai cách chọn còn lại có OR lần lượt là 5 và 6.

CT2-B / CHỨNG MINH VÀ CÀI ĐẶT

Cây khung OR nhỏ nhất

Vì sao thuật toán đúng?

Tại mọi bước, tồn tại cây khung chỉ dùng các cạnh phù hợp mask. Nếu xóa bit b mà vẫn liên thông thì tồn tại cây có bit b bằng 0 và tuân thủ các bit cao đã chọn; mọi nghiệm có cùng tiền tố bit cao nhưng bit b bằng 1 đều lớn hơn. Nếu mất liên thông, không cây khung nào có thể đặt bit b bằng 0 trong các ràng buộc trước đó. Quyết định lần lượt từ bit cao tới thấp do đó cho giá trị nhỏ nhất theo thứ tự số nguyên.

Mã giả

```
mask=(1<<30)-1
for bit=29..0:
    candidate=mask xor (1<<bit)
    init DSU with n components
    for each edge (u,v,w):
        if (w & candidate)=w: unite(u,v)
    if component_count=1: mask=candidate
print mask
```

Độ phức tạp

O(30(m+n)α(n)) thời gian mỗi test, O(m+n) bộ nhớ; α là hàm nghịch đảo Ackermann của DSU.

Các lỗi dễ gặp

- Cây khung có tổng trọng số nhỏ nhất chưa chắc có OR nhỏ nhất.
- Phải khởi tạo lại DSU cho từng mask thử.
- w ≤ 10^9 nên xét bit 29..0; không dùng nhầm phép AND thay cho OR trong mục tiêu.

CT2-B / LỜI GIẢI TỪNG SUBTASK

Cây khung OR nhỏ nhất

Subtask	Điểm	Giới hạn bổ sung
1	20	t ≤ 10, n ≤ 8 và m ≤ 15.
2	30	Mọi trọng số w ≤ 31.
3	50	Không có giới hạn bổ sung.

Subtask 1 - 20 điểm

Thử mọi tập n-1 cạnh. Dùng DSU kiểm tra liên thông rồi tính OR; lấy giá trị nhỏ nhất.

Độ phức tạp: O(t×2^m×m×α(n)) thời gian; O(n+m) bộ nhớ.

Subtask 2 - 30 điểm

Duyệt mask từ 0 đến 31. Với mỗi mask, kiểm tra liên thông bằng các cạnh có (w & mask)=w; mask khả thi đầu tiên là đáp án.

Độ phức tạp: O(32(m+n)α(n)) thời gian, O(m+n) bộ nhớ.

Subtask 3 - 50 điểm

Tham lam xóa từng bit từ cao xuống thấp, kiểm tra liên thông dưới toàn bộ mask hiện tại.

Độ phức tạp: O(30(m+n)α(n)) thời gian, O(m+n) bộ nhớ.

CT2-C / KHÁM PHÁ LỜI GIẢI

C. Màu nổi trội

Cây | DSU on tree | Đếm tần suất

ĐIỂM MẤU CHỐT

Các cây con chồng lặp rất nhiều; giữ dữ liệu của con lớn để tránh đếm lại toàn bộ.

Từ quan sát đến thuật toán

Tiền xử lý size[u], con nặng heavy[u], thứ tự Euler tin[u]..tout[u]. Khi thêm một đỉnh màu x, tăng cnt[x]. Nếu cnt[x]>best, gán best=cnt[x], sum=x; nếu bằng best, cộng x vào sum.

Hàm solve(u,keep): xử lý các con nhẹ với keep=false; xử lý con nặng với keep=true. Sau đó dữ liệu đang chứa đúng cây con nặng. Thêm mọi đỉnh trong các cây con nhẹ bằng đoạn Euler và thêm u. Lưu ans[u]=sum.

Nếu keep=false, xóa toàn bộ tần suất của cây con u rồi đặt best=sum=0. Tại thời điểm này bộ đếm chỉ chứa cây con u, nên có thể reset thẳng, không phải tìm lại màu nhiều nhất sau từng lần xóa. Triển khai bằng ngăn xếp sự kiện nếu muốn tránh tràn stack ở cây dây.

Kiểm tra bằng ví dụ

Mọi màu ở gốc cùng xuất hiện một lần nên tổng là 10. Cây con đỉnh 2 có các màu 2,3,4 nên tổng là 9.

CT2-C / CHỨNG MINH VÀ CÀI ĐẶT

Màu nổi trội

Vì sao thuật toán đúng?

Theo quy nạp, mỗi lần lưu ans[u], dữ liệu con nặng đã đúng, dữ liệu con nhẹ cũ đã được xóa, rồi toàn bộ cây con nhẹ và u được thêm đúng một lần. Do đó cnt mô tả chính xác cây con u. Quy tắc cập nhật best,sum giữ tổng của đúng các màu có tần suất lớn nhất. Một đỉnh chỉ bị thêm lại khi nó thuộc nhánh nhẹ; đi lên qua một cạnh nhẹ làm kích thước cây con tăng ít nhất gấp đôi. Mỗi đỉnh vì thế chịu O(log n) lần thêm/xóa.

Mã giả

```
solve(u,keep):
    for light child v: solve(v,false)
    if heavy[u] exists: solve(heavy[u],true)
    for light child v:
        for x in Euler subtree(v): add(c[x])
    add(c[u]); ans[u]=sum
    if not keep:
        for x in Euler subtree(u): cnt[c[x]]-=1
        best=0; sum=0

add(x):
    cnt[x]+=1
    if cnt[x]>best: best=cnt[x]; sum=x
    else if cnt[x]=best: sum+=x

solve(1,true); print ans[1..n]
```

Độ phức tạp

O(n log n) thời gian, O(n) bộ nhớ với DSU on tree và mảng tần suất. Tổng màu dùng 64 bit.

Các lỗi dễ gặp

- Cộng mã màu, không cộng số đỉnh hay tổng tần suất.
- Chỉ reset best,sum khi toàn bộ dữ liệu hiện tại đã bị xóa.
- Cây dây 10^5 đỉnh cần lưu ý giới hạn stack; DFS lặp xử lý được.

CT2-C / LỜI GIẢI TỪNG SUBTASK

Màu nổi trội

Subtask	Điểm	Giới hạn bổ sung
1	30	$n \le 2000$.
2	20	Các cạnh là $(i,i+1)$, $1 \le i < n$.
3	50	Không có giới hạn bổ sung.

Subtask 1 - 30 điểm

Với từng v, duyệt riêng cây con v, đếm màu và tính tổng các màu có tần suất lớn nhất.

Độ phức tạp: $O(n^2)$ thời gian, $O(n)$ bộ nhớ.

Subtask 2 - 20 điểm

Cây con v chính là hậu tố v..n. Quét v từ n về 1, thêm c[v] vào bộ đếm và cập nhật best,sum; lưu kết quả ngay.

Độ phức tạp: $O(n)$ thời gian, $O(n)$ bộ nhớ.

Subtask 3 - 50 điểm

DSU on tree giữ cây con nặng, thêm lại các cây con nhẹ và chỉ xóa khi keep=false.

Độ phức tạp: $O(n \log n)$ thời gian, $O(n)$ bộ nhớ.

CT2-D / KHÁM PHÁ LỜI GIẢI

D. Chia đoạn tối ưu

Quy hoạch động | Chia để trị | Cửa sổ tần suất

ĐIỂM MẤU CHỐT

DP chọn vị trí cắt; chi phí đếm cặp cho phép dịch cửa sổ nhanh và vị trí cắt tối ưu có tính đơn điệu.

Từ quan sát đến thuật toán

Đặt C(l,r)=Σ_x cnt[x](cnt[x]-1)/2. D[j][i] là chi phí nhỏ nhất chia i phần tử đầu thành j đoạn. D[j][i]=min(D[j-1][p]+C(p+1,i)) với j-1 ≤ p < i. Khởi tạo D[0][0]=0, các trạng thái khác là INF.

Chi phí thỏa bất đẳng thức Monge: C(a,c)+C(b,d) ≤ C(a,d)+C(b,c), với a ≤ b ≤ c ≤ d. Hiệu vế phải trừ vế trái chính là số cặp bằng nhau có một đầu trong [a,b-1], đầu kia trong [c+1,d], nên không âm. Do đó vị trí cắt tối ưu nhỏ nhất opt[j][i] không giảm theo i.

Tính một hàng DP bằng chia để trị. Tại mid, thử p trong phạm vi optL..min(mid-1,optR), rồi dùng vị trí tốt nhất để thu hẹp hai nửa. Duy trì C bằng cửa sổ [L,R]: thêm x làm cost tăng cnt[x] trước khi tăng cnt; xóa x thì giảm cnt trước, sau đó trừ cnt[x] khỏi cost. Không khởi tạo lại cửa sổ cho từng p.

Khởi tạo cửa sổ rỗng một lần cho mỗi hàng DP. Với cách duyệt đệ quy và quét ứng viên tăng dần, tổng dịch hai đầu trên mỗi mức chia để trị là O(n), nên mỗi hàng tốn O(n log n). Chỉ cần giữ hàng trước và hàng hiện tại.

Kiểm tra bằng ví dụ

Chia [1] | [1,3] | [3,3,2,1]. Chỉ đoạn cuối có một cặp bằng nhau.

CT2-D / CHỨNG MINH VÀ CÀI ĐẶT

Chia đoạn tối ưu

Vì sao thuật toán đúng?

Công thức DP xét đoạn cuối của mọi cách chia và không bỏ sót cách nào. Tính Monge dẫn tới tính đơn điệu của nghiệm tối ưu: nếu hai vị trí tối ưu đảo thứ tự, áp dụng bất đẳng thức cho hai đoạn cuối sẽ đổi chéo chúng mà không tăng tổng chi phí; khi hòa chọn vị trí nhỏ nhất, không thể giữ một nghịch thế. Vì vậy phạm vi chia để trị luôn chứa ít nhất một nghiệm tối ưu. Cửa sổ tần suất tính đúng C vì mỗi cặp được thêm khi đầu mút sau cùng vào cửa sổ và bị xóa khi một đầu mút ra ngoài.

Mã giả

```
prev[0]=0; prev[1..n]=INF
for j=1..k:
    cur[0..n]=INF; reset_window()
    calc(j,n,j-1,n-1)
    prev=cur
print prev[n]

calc(lo,hi,optL,optR):
    if lo>hi: return
    mid=(lo+hi)//2; bestP=-1
    for p=optL..min(mid-1,optR):
        move_window(p+1,mid)
        if prev[p]+cost < cur[mid]:
            cur[mid]=prev[p]+cost; bestP=p
    calc(lo,mid-1,optL,bestP)
    calc(mid+1,hi,bestP,optR)
```

Độ phức tạp

O(kn log n) thời gian, O(n) bộ nhớ; chi phí, INF và DP dùng số nguyên 64 bit.

Các lỗi dễ gặp

- Phải chia đúng k đoạn không rỗng, nên p<mid và p≥j-1.
- Khi xóa x: giảm cnt[x] trước rồi mới trừ vào cost.
- Không dùng n^2 ô chi phí ở full test; cửa sổ thay thế bảng chi phí.
- Giữ lựa chọn nhỏ nhất khi hòa để diễn giải opt đơn điệu nhất quán.

CT2-D / LỜI GIẢI TỪNG SUBTASK

Chia đoạn tối ưu

Subtask	Điểm	Giới hạn bổ sung
1	20	n ≤ 18.
2	30	n ≤ 2000.
3	50	Không có giới hạn bổ sung.

Subtask 1 - 20 điểm

Duyệt các tập k-1 điểm cắt trong n-1 khe. Tính chi phí từng đoạn bằng bộ đếm rồi lấy tổng nhỏ nhất.

Độ phức tạp: O(n×2^(n-1)) thời gian là cận trên; O(n) bộ nhớ.

Subtask 2 - 30 điểm

DP O(kn²): với mỗi hàng j và đầu phải i, quét p từ i-1 giảm về j-1, thêm a[p+1] vào bộ đếm để tính C(p+1,i) trong O(1) mỗi bước. Không cần lưu bảng chi phí.

Độ phức tạp: O(kn²) thời gian, O(n) bộ nhớ.

Subtask 3 - 50 điểm

Dùng tính Monge, chia để trị một hàng DP và cửa sổ tần suất để tính chi phí động như lời giải đầy đủ.

Độ phức tạp: O(kn log n) thời gian, O(n) bộ nhớ.