Ghép điểm - Match Points

Giả sử ta đã chọn được k cặp số để ghép với nhau. Giả sử số bé hơn của k cặp số đó lần lượt là $l_1, l_2, \dots, l_k$ và số lớn hơn tương ứng là $r_1, r_2, \dots, r_k$. Không mất tính tổng quá ta có thể giả sử rằng:

$$l_1 \le l_2 \le l_3 \le \cdots \le l_k$$

Có thể chứng minh được rằng sẽ tồn tại một hoán vị của r' của r sao cho r_i' ghép với l_i và $r_1' \le r_2' \le \cdots \le r_k'$ như sau:

- Xét hai số $l_i \le l_j$ mà có $r_i \ge r_j$, vì ta có thể ghép được các số này nên có $r_j - l_j \ge z$. Nhận thấy, $r_j - l_i \ge r_j - l_j \ge z$ và $r_i - l_j \ge r_j - l_j \ge z$ nên ta có thể ghép l_i với r_j và l_j với r_i mà kết quả không đổi.
- Vậy nếu tồn tại $l_i \le l_j$ và $r_i \ge r_j$ thì ta có thể đổi cặp này thành $l_i \le l_j$ và $r_i \le r_j$.
- Lặp lại quá trình trên nhiều lần ta sẽ nhận được $l_i \le l_{i+1}$ và $r_i \le r_{i+1}$.

Ta cũng có thể chứng minh được rằng có cách hoán vị các số trong l và r sao cho $l_k \le r_1$ như sau:

- Giả sử ngược lại là $r_1 < l_k$. Vậy thì ta có thể ghép l_1 với l_k và r_1 với r_k vì $l_k - l_1 > r_1 - l_1 \ge z$ và $r_k - r_1 > r_k - l_k \ge z$.
- Sau quá trình trên, ta thực hiện tính lại l và r như đã quy định. Lặp đi lặp lại thì cuối cùng sẽ có $l_k \le r_1$.

Trở lại với bài toán, ta có thể sort lại các số sao cho $x_1 \le x_2 \le \cdots \le x_n$.

Nhận thấy rằng nếu ta không ghép số x_1 thì có thể thay thế l_1 thành x_1 mà kết quả không đổi. Tương tự, ta có thể tiếp tục thay thế l_2 thành x_2, …, l_k thành x_k. Vì đảm bảo $l_k \le r_1$ nên sẽ không có chuyện một số bị dùng hai lần. Cũng tương tự như vậy, ta có thể thay r_k thành x_n, r_{k-1} thành x_{n-1}, …, r_1 thành x_{n-k+1}.

Vậy thì tồn tại kết quả tối ưu có dạng là dùng một số số bé nhất ghép với một số số lớn nhất.

Nhận thấy là nếu ghép được k cặp thì ta cũng ghép được $k-1$ cặp bằng cách bỏ đi một cặp và lại chuyển các cặp theo cách trên. Vì thế, ta có thể

tìm kiếm nhị phân số cặp ghép được rồi kiểm tra xem cách ghép có thỏa mãn không.

Làm như vậy thì độ phức tạp là $O(nlog(n))$ vì ta phải sort và tìm kiếm nhị phân.

Tuy nhiên thì cách trên không phải là cách duy nhất. Sau khi đã sort thì ta có cách $O(n)$ để tính được kết quả.

Đầu tiên phải nhận thấy là kết quả không vượt quá $\left\lfloor \frac{n}{2} \right\rfloor$. Vậy thì trong mọi cách ghép thỏa mãn có dạng đã trình bày ở trên, $\left\lfloor \frac{n}{2} \right\rfloor$ số đầu sẽ có vai trò là số bé hơn trong các cặp và $n - \left\lfloor \frac{n}{2} \right\rfloor$ số tiếp theo sẽ có vai trò là số lớn hơn.

Vậy thì bài toán bây giờ trở thành cho hai dãy a và b, ghép cặp giữa các số a_i và b_j sao cho $b_j - a_i \geq z$. Thì đây là một bài toán two pointers cổ điển, các bạn muốn thì có thể tự chứng minh. Gợi ý để chứng minh là dùng phương pháp “chọn số khác mà kết quả không đổi” như đã dùng ở trên.

https://codeforces.com/contest/1156/submission/55292490

Nhìn chung bài này không quá khó, điều kiện tối ưu thì có thể tự nhận ra được. Việc nhìn vào một nghiệm tối ưu bất kì và thay đổi một số giá trị sao cho kết quả không đổi nhưng nghiệm “đẹp” hơn vừa giúp ta giải được các bài toán, vừa phục vụ việc chứng minh các thuật toán này.

Tô màu cạnh - Coloring Edges

Đầu tiên nhận thấy chỉ trong trường hợp không có chu trình thì kết quả mới là 1.

Trong mọi trường hợp còn lại thì kết quả là 2. Thì có nhiều cách để đạt được kết quả là 2, ta có thể tìm một cách như sau:

- Giả sử tô được bằng 2 màu. Vậy thì đồ thị chỉ gồm toàn bộ cạnh của mỗi màu phải là một DAG (đồ thị có hướng không có chu trình).
- Vậy thì ta cần phải tìm được hai DAG sao cho gộp lại thì ra đồ thị ban đầu.
- Mặt khác, ta có các tính chất của DAG như sau:
    - Nếu một đồ thị là DAG thì nếu ta đảo ngược tất cả các cạnh của đồ thị này, ta cũng được một DAG.
    - Một DAG thì có thể toposort lại được, nghĩa là có thể đánh số các đỉnh sao cho chỉ có đường đi từ đỉnh bé hơn đến đỉnh lớn hơn. Vậy thì ta có thể định nghĩa một DAG đầy đủ là một DAG mà có cạnh có hướng giữa tất cả các cặp đỉnh. Mỗi DAG thì luôn là một thành phần con của một DAG đầy đủ.
    - Đảo ngược tất cả cạnh của DAG đầy đủ và ghép với DAG đầy đủ ban đầu thì ta sẽ thu được một đồ thị có hướng đầy đủ. Nếu ta tô hai cái DAG này khác màu nhau thì sẽ thu được kết quả cho hai DAG và mọi cặp hai đồ thị con của hai DAG này.
- Vậy thì để giải bài toán này, ta có thể tìm một DAG bất kì là đồ thị con của đồ thị ban đầu. Sau đó, ta toposort lại DAG này và đi tìm cái DAG đầy đủ của nó. Ta có thể tô tất cả các cạnh nằm trong đồ thị ban đầu màu 1 nếu nó nằm trong DAG đầy đủ và màu 2 nếu nó không nằm trong để luôn đạt được kết quả là 2.
- Vậy thì nên chọn DAG nào để giải quyết dễ ràng nhất?
    - Nên chọn cái đồ thị con mà chứa toàn các cạnh nối $u \to v$ sao cho $u < v$. Đồ thị này đã được toposort sẵn, DAG đầy đủ có thể dùng là đồ thị sao cho có cạnh $i \to j$ khi và chỉ khi $i < j$. Có thể chứng minh được theo kiểu khác là cách này sẽ đúng: khi đi bằng một màu thì đỉnh ta đi vào chỉ có tăng lên hoặc giảm đi, không vừa tăng vừa giảm nên không tạo thành chu trình được.

- Trên thực tế thì ta có thể sinh ra một hoán vị p bất kì, sau đó tô các cạnh $u \to v$ tùy theo u đứng trước hay sau v trong hoán vị.

https://codeforces.com/contest/1217/submission/60107780

Bài này có điều đặc biệt là đề bài cho $1 \leq n, m \leq 5000$ trong khi độ phức tạp thực ra chỉ có $O(n+m)$. Thì cái này là để mọi người nhìn vào, đoán vội là sẽ phải làm gì đó theo kiểu $O(n^2+m^2)$. Trong thi VOI, TST, APIO, IOI thì thường sẽ không có kiểu này. Ngoại lệ có thể là cách tối ưu hơn cách ban tổ chức $O(log)$ hoặc $O(log^2)$ hoặc là ban tổ chức cho phép chia căn mặc dù cách tối ưu dùng log …. Nói chung là thường sẽ không có chuyện để giới hạn sao cho $O(n^2)$ chạy được trong khi $O(n \times P)$ là thuật toán tối ưu, với $P \ll n$. Bài này thì thực ra không có thuật $O(n^2)$ nào dễ thấy cả, nên cho giới hạn như vậy chỉ để làm khó thí sinh mà thôi.

Túi thẻ - Card Bag

Mỗi trò chơi sẽ luôn có dạng là ta chọn lần lượt các số, mỗi số lớn hơn số trước đó cho đến một lúc nào đó thì dừng lại. Điều này xảy ra vì trò chơi sẽ luôn kết thúc khi ta chọn một số bé hơn hoặc bằng số đang có.

Giả sử số đang có là y, ta xét các trường hợp chọn ngẫu nhiên ra một số x:

- $x > y$: Trường hợp này có xác suất bằng số lượng số x chia cho số lượng số còn lại trong túi vì dĩ nhiên ta chưa chọn số $x > y$ nào.
- $x = y$: Trường hợp này có xác suất bằng (số lượng số x (y) trừ 1) chia cho số lượng số còn lại trong túi. Điều này xảy ra vì dĩ nhiên là ta cũng chưa chọn số $x = y$ nào.
- $x < y$: Trường hợp này có xác suất bằng số số bé hơn y trừ đi số số đã chọn mà bé hơn y rồi tổng cả chia cho số lượng số chưa chọn. Vì trường hợp này kết thúc trò chơi nên ta không cần quan tâm x cụ thể là số nào.

Ta có thể lưu lại thêm một thông tin ta đã chọn được bao nhiêu số để thực hiện quy hoạch động. Cụ thể, gọi $f[i][j]$ là xác suất mà game vẫn đang diễn ra, ta đã chọn được i số và j là giá trị của số hiện tại.

Gọi $cnt[i]$ là số lượng số i có trong túi ban đầu.

Có một biến win để tính xác suất thắng trong tất cả quá trình quy hoạch động.

Công thức cụ thể như sau:

- $f[0][0] = 1$ vì đây là vị trí bắt đầu trò chơi.
- $f[i+1][j] = sum\left(f[i][j'] \times \frac{cnt[j]}{n-i}\right)$ với $j' < j$. Điều kiện $j' < j$ là vì ta chỉ có thể chọn số lớn hơn số hiện tại mà game vẫn tiếp tục. $\frac{cnt[j]}{n-i}$ là xác suất để ra số j từ $n - i$ số chưa được chọn.
- Để tính được giá trị trên nhanh chóng thì ta có thể tính bằng tổng dồn, duyệt j tăng dần và tính là được.

Tính xong thì ta có thể tính kết quả:

$$win = sum\left(f[i][j] \times \frac{cnt[j]-1}{n-i}\right)$$

$\frac{cnt[j]-1}{n-i}$ là xác suất để chọn thêm một số j nữa từ các số còn lại và thắng game. Có thể vừa duyệt i vừa tính để tiết kiêm thời gian.

Trường hợp thua thì ta không cần quan tâm, vì game kết thúc và ta chỉ quan tâm đến xác suất thắng.

https://codeforces.com/contest/1156/submission/55293810

Nhìn chung bài này cần nhận ra ta cần những thông tin nào để có thể làm được. Từ đó chỉ cần quy hoạch động.