Mua vé - Buy a Ticket

Vì đây là đường hai chiều nên dĩ nhiên ta sẽ đi cùng đường khi đi và về để được kết quả tốt nhất.

Vậy thì ta có thể bỏ qua việc đi ngược lại bằng cách là nhân đôi trọng số mỗi cạnh. Làm vậy thì kết quả bây giờ là tìm đường ngắn nhất để di chuyển đến một đỉnh rồi mua vé ở đó.

Thay vì với mỗi đỉnh ta phải đi đến một đỉnh khác rồi mua vé, ta có thể làm ngược lại là mua vé trước, sau đó đi về các đỉnh ban đầu. Làm vậy thì kết quả ở mỗi đỉnh chỉ là đường đi ngắn nhất từ các đỉnh khác sau khi mua vé.

Cụ thể, ta có thể mô phỏng lại quá trình:

- Có một đỉnh 0 nghĩa là chưa có vé.
- Ta có thêm cạnh từ $0 \to i$ trọng số a_i có nghĩa là ta mua vé ở i sẽ mất a_i.
- Kết quả ở đỉnh i chỉ là đường đi ngắn nhất từ 0 đến i, có nghĩa sẽ mua vé ở đâu đó rồi di chuyển đến i.

Độ phức tạp bằng độ phức tạp đường đi ngắn nhất, là $O\big((n+m)\log(n)\big)$ nếu dùng dijkstra.

https://codeforces.com/contest/938/submission/35363702

Bài này cho thấy là đảo ngược đường lại sẽ giúp giải quyết được nhiều bài toán liên quan đến đường đi ngắn nhất. Cụ thể, nếu phải tính cho nhiều đỉnh thì ta nên đi từ đầu chung đến đầu riêng để giải quyết nhanh được. Đầu chung ở đây là việc mua vé, vì dù có ở đâu thì việc mua vé là việc cuối cùng mà mỗi đỉnh cần làm.

Phép cộng trên đoạn thẳng - Addition on Segments

Để biết được số lớn nhất là số nào thì đầu tiên ta cần biết vị trí của nó ở đâu.

Trong bài này thì ta luôn có thể cho mỗi vị trí $1 \leq i \leq n$ là vị trí của số lớn nhất. Điều này có thể đạt được bằng cách là không làm mọi thao tác mà không chứa i. Nếu ta chỉ làm các thao tác mà có chứa i thì i chắc chắn là vị trí lớn nhất vì:

- Số ở vị trí thứ i lúc đầu là lớn nhất (tất cả bằng 0)
- $x_j \geq 1$ với mọi j nên ở mỗi thao tác ta làm thì số ở vị trí i luôn tăng.
- Những số mà không được tăng thì sẽ bé hơn số ở i.
- Những số khác cùng được tăng thì tăng không quá số lượng mà số ở vị trí i tăng, nên số ở vị trí thứ i vẫn luôn là lớn nhất.

Vậy thì ta có ý tưởng là duyệt vị trí i của số lớn nhất, rồi kiểm tra xem liệu có thể tạo được các số lớn nhất nào ở vị trí này.

Thì đây là một bài toán cổ điển: cho các số $y_1, y_2, \ldots, y_k$ là các số x_j tương ứng mà có $l_j \leq i \leq r_j$, tìm ra xem với mỗi số $1 \leq a \leq n$, có thể tạo được số này là tổng của một số số y_b hay không.

Bài toán này thì có thể giải được bằng quy hoạch động mất $O(k \times n)$, hay $O(q \times n)$ vì $k \leq q$. Vì ta cần giải cho n vị trí, độ phức tạp tổng cả là $O(qn^2)$, không đủ nhanh vì $n, q \leq 10^4$. Có thể sử dụng bitset để cải thiện thành $O\left(\frac{qn^2}{64}\right)$, tuy nhiên làm vậy thì vẫn không đủ nhanh.

Thì nhận thấy là các bài toán ta cần giải không phải hoàn toàn độc lập với nhau. Ví dụ, tập các số y_j khi $i = 1$ và $i = 2$ sẽ có rất nhiều điểm chung.

Vậy thì có ý tưởng là ta sẽ duy trì một mảng $f[x]$ là liệu có cách nào tạo được x ở vị trí i hiện tại không.

- Khi di chuyển từ $i - 1$ đến i thì ta cần thêm một số số và tập y_j và xóa một số số khỏi tập y_j.

- Việc thêm số vào thì rất dễ, nếu thêm số x mà đang tạo được số a thì số $a+x$ cũng tạo được. Để cập nhật ta có thể duyệt ngược $a+x$ từ n về x rồi kiểm tra.
- Việc xóa số x thì phức tạp hơn vì nếu ta đang tạo được một số $a = b+x$, chưa chắc là xóa x đi đã không tạo được a.
- Để thực hiện cả thêm + xóa thì thay vì chỉ lưu là có tạo được các số không, ta có thể lưu là có bao nhiêu cách để tạo được một số nào đó.
- Vì thứ tự các số y_j không quan trọng nên ta có thể thực hiện thêm + xóa tùy ý.

Cụ thể, gọi $f[i]$ là số cách tạo được số i bằng một tập số nào đó.

Ban đầu tập số trống nên $f[i] \neq 0$ khi $i > 0$ và $f[0] = 1$.

Khi thêm một số x vào tập, ta sẽ có:

```
for(int i=n; i>=x; i--) f[i]+=f[i-x];
```

Điều này chỉ là với mỗi cách tạo được số $i-x$ thì có thêm nấy đó cách tạo được số i. Duyệt ngược lại để chỉ sử dụng một mảng.
Khi xóa một số khỏi tập x thì ta chỉ cần làm ngược lại:

```
for(int i=x; i<=n; i++) f[i]-=f[i-x];
```

Số cách tạo dĩ nhiên có thể rất lớn, vì vậy khi tính toán thì ta phải tính modulo. Chọn modulo thì nên tránh những cái hay dùng như là 10^9+7, $10^9+9, 998244353$. Mỗi người nên chọn một số nguyên tố lớn của riêng mình để dùng làm mod. Nên chọn số có tính chất đặt biệt cho dễ nhớ, vì dụ như là đối xứng, chứa ngày sinh. Ví dụ mình hay dùng 999727999. Số này dễ nhớ vì có 9 chữ số, tất cả đều là 9, 3 số ở giữa là 727, meme khá nổi tiếng ngày xưa.
https://codeforces.com/contest/981/submission/64111010
Bài này sử dụng quy hoạch động nhưng lại làm ngược lại. Với những loại thao tác mà thứ tự không quan trọng, nếu làm ngược lại được sẽ giúp giải quyết nhiều vấn đề.

ZS và nghịch lí ngày sinh - ZS and The Birthday Paradox

Đầu tiên ta phải giải được bài toán là có n người mà chọn ra k người thì xác suất để có hai người có cùng ngày sinh là bao nhiêu?

- Nếu $k > n$ thì xác suất là 1 vì theo định lí Dirichlet, luôn có hai người có cùng ngày sinh.
- Nếu $k \leq n$ thì ta có thể tìm ra xác suất mà không có hai người có cùng ngày sinh như sau:
    - Số cách chọn ngày sinh của k người sao cho không có ai có cùng ngày sinh là: $n \times (n-1) \times (n-2) \times ... \times (n-k+1)$. Điều này là vì người thứ nhất có thể sinh bất kì ngày nào, người thứ hai có $n-1$ vì phải tránh ngày sinh của người thứ nhất, …
    - Tổng số cách chọn ngày sinh của k người đơn giản là n^k vì mỗi người có n ngày sinh có thể chọn.
    - Vậy thì kết quả là $\frac{n \times (n-1) \times (n-2) \times ... \times (n-k+1)}{n^k}$ (1)
- Xác suất hai người có cùng ngày sinh sẽ là:

$$1 - \frac{n \times (n-1) \times (n-2) \times ... \times (n-k+1)}{n^k}$$

Bài toán dùng 2^n thay vì n, nên công thức cho bài toán thực tế là:

$$1 - \frac{2^n \times (2^n-1) \times (2^n-2) \times ... \times (2^n-k+1)}{2^{nk}}$$

Nếu cho một phần số giản $\frac{A}{B}$ mà $0 < \frac{A}{B} < 1$ thì $1 - \frac{A}{B} = \frac{B-A}{B}$ cũng là một phân số tối giản. Điều này là vì $1 = \gcd(A, B) = \gcd(B-A, B)$ theo thuật toán Euclid.

Vậy ta sẽ đi tìm phân số tối giản của $\frac{2^n \times (2^n-1) \times (2^n-2) \times ... \times (2^n-k+1)}{2^{nk}}$.

Nhận thấy là mẫu số chỉ có toàn 2 nên để rút gọn mẫu số ta chỉ cần đếm số lượng 2 trên tử số.

- Đầu tiên thấy là có một cái 2^n trên tử nên có ít nhất n thừa số 2 trên tử.

- Vì trường hợp có nhiều người hơn ngày thì kết quả đã sử lí riêng, nên $k \leq 2^n$.
- Vậy thì tất cả các số $2^n - 1, 2^n - 2, \dots, 2^n - k$ sẽ không có số nào chia hết cho 2^n cả.
- Vì thế thì số lượng thừa số nguyên tố 2 trong $2^n - i$ chính là số lượng thừa số nguyên tố 2 trong $\gcd(2^n - i, 2^n)$.
- $\gcd(2^n - i, 2^n) = \gcd(i, 2^n)$. Vì $i < k \leq 2^n$ nên số lượng thừa số nguyên tố 2 trong $2^n - i$ thực tế chính là số lượng thừa số nguyên số 2 trong i.
- Vậy thì ta cần tính tổng số lượng thừa số nguyên tố 2 trong các số $1, 2, 3, \dots, k - 1$.
- Đây là bài toán tính số lượng thừa số nguyên tố p của một số trong một giai thừa $f!$. Bài toán này cổ điển và có thể giải được bằng cách tính:

$$\left\lfloor \frac{f}{p} \right\rfloor + \left\lfloor \frac{f}{p^2} \right\rfloor + \left\lfloor \frac{f}{p^3} \right\rfloor + \cdots$$

  - Số thứ nhất là số lượng số chia hết cho p. Ta chia tất cả các số chia hết cho p và loại bỏ các số còn lại, sẽ được $\left\lfloor \frac{f}{p} \right\rfloor$ số. Sau đó lại lặp lại để đếm, sẽ được $\left\lfloor \frac{\left\lfloor \frac{f}{p} \right\rfloor}{p} \right\rfloor = \left\lfloor \frac{f}{p^2} \right\rfloor$. Lặp lại quá trình này đến khi kết quả bằng 0 sẽ được tổng số lượng. $\left\lfloor \frac{\left\lfloor \frac{f}{p^i} \right\rfloor}{p} \right\rfloor = \left\lfloor \frac{f}{p^{i+1}} \right\rfloor$ có thể thấy được bằng việc viết f dưới dạng một số trong hệ cơ số p, mỗi lần chia tương ứng với việc loại bỏ một số số cuối.

Vậy đến đây ta tính được số lượng thừa số nguyên tố 2 ở mẫu số, xem như là tính được mẫu số. Vậy tính tử số rút gọn như thế nào?

- Ta phải tính tích của k số liên tục, chia kết của cho 2^a với a số lượng thừa số nguyên tố 2 trên tử, rồi mod cho $10^6 + 3$ là một số nguyên tố.
- Nhận thấy là $\gcd(10^6 + 3, 2^a) = 1$ nên nếu tích k số liên tiếp mà chia hết cho $10^6 + 3$ thì cả tử số cũng chia hết cho $10^6 + 3$.
- Nếu $k \geq 10^6 + 3$ thì dĩ nhiên là tích k số liên tiếp sẽ chia hết cho $10^6 + 3$, ta biết được luôn tử số là 0.

- Nếu $k < 10^6 + 3$ thì ta có thể duyệt để tính tích rồi thực hiện chia 2^a bằng cách dùng nghịch đảo modulo.

Tính được tử số và mẫu số, ta có kết quả của bài toán.

Một số lưu ý: Trong khi tính toán sẽ có chỗ ta cần tính $2^{O(n \times k)}$. Thì $O(n \times k) \approx 10^{36}$ trong trường hợp xấu nhất nên C++ long long không giải quyết được. Thì cách để làm là sử dụng hàm phi Euler hoặc trong trường hợp này là đinh lí Fermat nhỏ.

Cụ thể, nếu $\gcd(x, p) = 1$ thì $x^{\varphi(p)} \equiv 1 \ (mod\ p)$. Trong trường hợp p là số nguyên tố, $\varphi(p) = p - 1$.
https://codeforces.com/contest/711/submission/27338056

Nhìn chung bài này chỉ là toán học. Những kiến thức về hàm phi Euler hay cách tính số thừa số, tổ hợp, chỉnh hợp sẽ rất tiện lợi trong các bài toán đếm, đôi khi để giải quyết quy hoạch động nhanh hơn hoặc thay thế hoàn toàn quy hoạch động.