জাতীয় হাইস্কুল প্রোগ্রামিং প্রতিযোগিতা

জুন ২৭, ২০২০

A. Butcher of Vowels

প্রতিটি টেস্টকেসের স্ট্রিং-টি ইনপুট নিয়ে তার প্রতিটি ক্যারাক্টারের উপর লুপ চালিয়ে দেখতে হবে কোন ক্যারাক্টার `a, e, i, o, u` এর কোন একটি কি না। যদি হয় তাহলে ঐ টেস্টকেসের জন্য `Yes` প্রিন্ট করতে হবে, অন্যথায় No ।

B. No More Child Labor

Key idea

সমস্যাটি সমাধানের জন্য প্রথমে বুঝতে হবে যে কখন একটা পাথে থাকা অক্ষরগুলো দিয়ে একটি প্যালিন্ড্রম বানানো সম্ভব। এর উত্তর হচ্ছে, পথে থাকা অক্ষরগুলোর মধ্যে যদি একাধিক অক্ষর পাওয়া না যায় যারা বিজোড় সংখ্যাকবার আছে এই পথে তাহলেই কেবল অক্ষরগুলো এমনভাবে বিন্যস্ত করা সম্ভব যেন একটি প্যালিন্ড্রম স্ট্রিং গঠিত হয়।

সাবটাস্ক ১

এখানে একটিই পথ সম্ভব $(1,1)$ থেকে $(1,M)$ পর্যন্ত। কাজেই শুধুমাত্র কোন অক্ষর কতবার আছে সেটা বের করে দেখতে হবে। যদি একাধিক অক্ষর বিজোড় সংখ্যাকবার না থাকে তাহলে এই পথের অক্ষরগুলো নিয়ে প্যালিন্ড্রম বানানো সম্ভব। অন্যথায় সম্ভব নয়। যেহেতু গ্রিডের সব বর্গ এই একটি পথেই অবস্থান করে। তাই প্যালিন্ড্রম বানানো সম্ভব হলে সব বর্গের জন্য উত্তর হবে 1 আর সম্ভব্য না হলে 0 ।

কতগুলো অক্ষর বিজোড় সংখ্যাক বার আছে তা জানার জন্য আমরা একটি $bitmask$ ব্যাবহার করতে পারি (সাবটাস্ক ৩ এর সমাধানে এ পদ্ধতি আলোচিত হবে) । এবং সেক্ষেত্রে প্রতি টেস্ট কেসে সমাধানের কমপ্লেক্সিটি হবেঃ $O(M)$

সাবটাস্ক ২

এক্ষেত্রে গ্রিডেটি $2 \times M$ আকারের হবে। একটু লক্ষ্য করলেই দেখা যাচ্ছে যে এই গ্রিডে $(1,1)$ থেকে $(2,M)$ পর্যন্ত যাওয়ার যেকোন পথ আসলে এরকম হবেঃ $(1,1) \rightarrow ... \rightarrow (1,Y) \rightarrow (2,Y) \rightarrow ... \rightarrow (2,M)$। এখান থেকে দেখা যায় যে $(1,Y)$ এবং $(2,Y)$ বর্গ দুটি থাকে এরকম পথ আসলে কেবল একটিই সম্ভব। কাজেই আমরা যদি $(1,1) \rightarrow ... \rightarrow (1,Y)$ আর (2,Y) $\rightarrow .... \rightarrow (2,M)$ পথে কোন অক্ষরগুলো কতবার আছে বের করে রাখি তাহলে আমরা বলতে পারব যে $(1,Y)$ এবং

$(2,Y)$ বর্গ দুটি থাকে এরকম পথটি থেকে প্যালিন্ড্রম বানানো যাবে কিনা। যদি বানানো সম্ভব হয় তাহলে $(1,1) \rightarrow ... \rightarrow (1,Y)$ এবং $(2,Y) \rightarrow .... \rightarrow (2,M)$ এর সবগুলো বর্গের উত্তরের সাথে ১ যোগ হবে। এক্ষেত্রে কমপ্লেক্সিটি কমানোর জন্য প্রথমে শুধুমাত্র $(1,Y)$ এবং $(2,Y)$ এর জন্য +১ করে পরবর্তীতে $Cumulative\ Sum$ করতে পারো।

কতগুলো অক্ষর বিজোড় সংখ্যাক বার আছে তা জানার জন্য আমরা একটি $bitmask$ ব্যাবহার করতে পারি (সাবটাস্ক ৩ এর সমাধানে এ পদ্ধতি আলোচিত হবে) । এবং সেক্ষেত্রে প্রতি টেস্ট কেসে সমাধানের কমপ্লেক্সিটি হবেঃ $O(M)$

সাবটাস্ক ৩

প্রথমেই আমরা দেখব কোন একটি পথে থাকা অক্ষরগুলো দিয়ে প্যালিন্ড্রম বানানো যায় কিনা তা বের করার জন্য বিটমাস্ক কীভাবে কাজ করে। আমরা সবাই XOR অপারেশনের বৈশিষ্ট্য জানি। XOR প্রক্রিয়ায় বিজোড় সংখ্যাক 1 অংশ নিলে রেজাল্ট হবে 1 , অন্যথায় সব ক্ষেত্রে রেজাল্ট হবে 0। কাজেই আমরা একটি ভ্যারিয়েবল $mask = 0$ কল্পনা করি। যেহেতু $mask = 0$ কাজেই আমরা বলতে পারি এর সব বিট পজিশনে 0 আছে। আমরা $'a' \rightarrow' j'$ অক্ষরগুলোর জন্য যথাক্রমে $0 - 9$ পর্যন্ত ম্যাপ করি। তার মানে হচ্ছে $'a'$ বলতে আমরা বুঝব 0 , $'b'$ বলতে 1, $'c'$ বলতে 2 । এখন আমরা পথে থাকা প্রতিটি ক্ ক্যারেক্টারের ম্যাপ ভ্যালু t এর জন্য যদি 2^t বা $(1 \ll t)$ দিয়ে mask এর সাথে XOR করব। তাহলে পথে থাকা প্রতিটি অক্ষরের জন্য এই অপারেশন করার পর $mask$ এর সেসব বিট পজিশনেই 1 থাকবে যেসব পজিশনের ক্যারেক্টারগুলো পথে বিজোড় সংখ্যাকবার এসেছে। $mask$ এ যদি একাধিক বিট পজিশনে 1 থাকে তাহলে আমরা বলতে পারব যে পথটি থেকে প্যালিন্ড্রম বানানো সম্ভব না।

এখন সাবটাস্ক ৩ তে গ্রিডটির সাইজ হবে $N \times M$ যেখানে $N \geq 3$ এবং $M \geq 3$ । কাজেই দেখা যাচ্ছে একটি বর্গ হয়ে অনেকগুলো পথ যেতে পারে অনেকভাবে। কাজেই প্রতিটি পথের জন্য আলাদা ভাবে প্যালিন্ড্রম বানান যাবে কিনা বের করা যথেষ্ঠ সময়সাপেক্ষ ব্যাপার হবে। এজন্য আমরা একটু উল্টোভাবে চিন্তা করব। ধরা যাক আমরা (i,j) বর্গে আছি। তাহলে আমরা এই বর্গের জন্য দুইটি ফাংশন $F(i,j,mask)$,$G(i,j,mask)$ এর মান বের করব যেখানে $mask = 0$ থেকে $2^{10} - 1$ পর্যন্ত যেকোন সংখ্যা।

$F(i,j,mask)$ = কতগুলো পথ আছে যেন $(1,1)$ থেকে (i,j) তে আসতে ঐ পথের অক্ষরগুলোর জন্য $mask$ ভ্যালু বানানো যায়।

$G(i,j,mask)$ = কতগুলো পথ আছে যেন (i,j) থেকে (N,M) তে আসতে ঐ পথের অক্ষরগুলোর জন্য $mask$ ভ্যালু বানানো যায়।

ধরা যাক $t = (i,j)$ বর্গে থাকা ক্যারেক্টারটির ম্যাপ ভ্যালু। So, $C = 2^t$

$\oplus = XOR\ operation.$

$$F(i,j,mask) = \begin{cases} 0 & \text{if } i = 1, j = 1, mask \neq C \\ 1 & \text{if } i = 1, j = 1, mask = C \\ F(i, j-1, mask \oplus C) & \text{if } i = 1 \\ F(i-1, j, mask \oplus C) & \text{if } j = 1 \\ F(i, j-1, mask \oplus C) + F(i-1, j, mask \oplus C) & \text{if } 1 < i \leq N, 1 < j \leq M \end{cases}$$

$$G(i,j,mask) = \begin{cases} 0 & \text{if } i = N, j = M, mask \neq C \\ 1 & \text{if } i = N, j = M, mask = C \\ G(i, j+1, mask \oplus C) & \text{if } i = N \\ G(i+1, j, mask \oplus C) & \text{if } j = M \\ G(i, j+1, mask \oplus C) + G(i+1, j, mask \oplus C) & \text{if } 1 \leq i < N, 1 \leq j < M \end{cases}$$

P(i,j,mask) = $(1,1)$ থেকে (N,M) এ আসতে কতগুলো পথ আছে যারা (i,j) বর্গ হয়ে যায় এবং ঐ পথের অক্ষরগুলোর জন্য mask ভ্যালু বানানো যায়। ভলো পথের mask ভ্যালুতে একাধিক

বিট পজিশনে 1 থাকতে পারবে না।

তাহলে কোন একটি বর্গ (i,j) এর জন্য নির্ণেয় ভালো পথের সংখ্যা = $P(i,j,0)+\sum_{k=0}^{9} P(i,j,2^k)$

যেখানে, $p(i,j,mask) = \sum_{m=0}^{2^{10}-1}(F(i,j,m) \times G(i,j,mask \oplus m \oplus C))$

$DynamicProgramming$ করে আমরা বারবার একই সাবপ্রব্লেমের সমাধান করা আটকাতে পারব।

সলিউশনের কমপ্লেক্সিটিঃ প্রতি টেস্ট কেসে $O(N*M*2^{10})$

C. Video Game Pro

সাবটাস্ক ১ এর জন্য :

কোন একটি নির্দিষ্ট ইন্ডেক্স থেকে শুরু করে বাকি সব বিল্ডিং ভ্রমণ করার $(N-1)!$ টি ভিন্ন ভিন্ন পথ থাকা সম্ভব। তাহলে আমরা সম্ভাব্য সকল উপায়ে বিল্ডিং গুলো ভ্রমণ করার চেষ্টা করতে পারি প্রব্লেম এ দেয়া রেস্ট্রিকশন ফলো করে । এরমধ্যে যে উপায়ে সবচেয়ে বেশি কয়েন সংগ্রহ করা যাবে সেটিই আউটপুট । এ প্রক্রিয়ায় টাইম কমপ্লেক্সিটি দাঁড়াবে $O((N-1)! \times N \times N)$।

সাবটাস্ক ২ এর জন্য :

আমরা শুরুতেই সব ইন্ডেক্সের জন্য হিসাব করে রাখতে পারি এই ইন্ডেক্স থেকে শুরু করলে সর্বোচ্চ কতগুলো কয়েন সংগ্রহ করতে পারবে। এজন্য বিল্ডিং গুলোকে তাদের হাইট অনুযায়ী সর্ট করে ছোট হাইট থেকে হিসেব করা শুরু করব। কোন একটা বিল্ডিং নিয়ে কাজ করার সময় দেখব এই বিল্ডিং থেকে ডানে এবং বামে সর্বোচ্চ কোন ইন্ডেক্স পর্যন্ত লাফ দিয়ে যাওয়া যায় , ধরি আমি বর্তমানে i তম ইন্ডেক্সে আছি এবং এই ইন্ডেক্স থেকে বামে আমি সর্বোচ্চ l তম ইন্ডেক্সে যেতে পারি এবং ডানে সর্বোচ্চ r তম ইন্ডেক্সে যেতে পারি। তাহলে এখন আমি দেখব l থেকে r রেঞ্জের মধ্যে কোন কোন ইন্ডেক্সের বিল্ডিংগুলো i তম ইন্ডেক্সের বিল্ডিং এর সমান উচ্চতার। এদের সবার হিসাব আমরা একত্রে করতে পারি । তাহলে প্রথম এদের টোটাল কয়েন যোগ করি , ধরি সেই সংখ্যাটি x । এবার এই l,r রেঞ্জের মধ্যে অলরেডি হিসেব করা হয়েছে এমন ইন্ডেক্সগুলোর মধ্যে যে ইন্ডেক্সের থেকে সর্বোচ্চ কয়েন সংগ্রহ করা গিয়েছে সেই সংখ্যাটা x এর সাথে যোগ হবে। ধরি যোগ করার পর টোটাল y পাওয়া গেল । তাহলে l,r রেঞ্জের মধ্যে যেসব ইন্ডেক্সের বিল্ডিং এর উচ্চতা i এর সমান তাদের সকলের জন্য আউটপুট হবে y।

কমপ্লেক্সিটি : কোন একটা ইন্ডেক্স i এর জন্য আমরা সহজেই লুপ চালিয়ে l,r নির্ণয় করা এবং অন্যান্য প্রয়োজনিয় হিসাব করতে পারি। এজন্য কমপ্লেক্সিটি হবে $O(N)$ প্রত্যেক i এর জন্য । তাহলে সব ইন্ডেক্সের জন্য হিসাব করতে টোটাল টাইম লাগবে $O(N \times N)$ । এরপর প্রত্যেক কুয়েরিতে আমরা আগের হিসেব করা থেকে $O(1)$ কমপ্লেক্সিটিতে আউ্টপুট দিতে পারব।

সাবটাস্ক ৩ এর জন্য :

সাবটাস্ক ৩ এর বেসিক আইডিয়া সাবটাস্ক ২ এর মত ই , শুধু কিছু টেকনিক এবং ডেটা স্ট্রাকচার ব্যবহার করে কমপ্লেক্সিটি কমানো হয়েছে । আমরা শুরুতেই সব ইন্ডেক্সের জন্য হিসাব করে রাখতে পারি এই ইন্ডেক্স থেকে শুরু করলে সর্বোচ্চ কতগুলো কয়েন সংগ্রহ করতে পারবে। এজন্য বিল্ডিং গুলোকে তাদের হাইট অনুযায়ী সর্ট করে ছোট হাইট থেকে হিসেব করা শুরু করব। কোন একটা বিল্ডিং নিয়ে কাজ করার সময় দেখব এই বিল্ডিং থেকে ডানে এবং বামে সর্বোচ্চ কোন ইন্ডেক্স পর্যন্ত লাফ দিয়ে যাওয়া যায় , ধরি আমি বর্তমানে i তম ইন্ডেক্সে আছি এবং এই ইন্ডেক্স থেকে বামে আমি সর্বোচ্চ l তম ইন্ডেক্সে যেতে পারি এবং ডানে সর্বোচ্চ r তম ইন্ডেক্সে যেতে পারি। তাহলে এখন আমি দেখব l থেকে r রেঞ্জের মধ্যে কোন কোন ইন্ডেক্সের বিল্ডিংগুলো i তম ইন্ডেক্সের বিল্ডিং এর সমান উচ্চতার।

এদের সবার হিসাব আমরা একত্রে করতে পারি । তাহলে প্রথম এদের টোটাল কয়েন যোগ করি , ধরি সেই সংখ্যাটি x । এবার এই l,r রেঞ্জের মধ্যে অলরেডি হিসেব করা হয়েছে এমন ইন্ডেক্সগুলোর মধ্যে যে ইন্ডেক্সের থেকে সর্বোচ্চ কয়েন সংগ্রহ করা গিয়েছে সেই সংখ্যাটা x এর সাথে যোগ হবে। ধরি যোগ করার পর টোটাল y পাওয়া গেল । তাহলে l,r রেঞ্জের মধ্যে যেসব ইন্ডেক্সের বিল্ডিং এর উচ্চতা i এর সমান তাদের সকলের জন্য আউটপুট হবে y।

প্রত্যেক ইন্ডেক্স i এর জন্য আমরা l এবং r প্রিক্যালকুলেট করে রাখতে পারি স্লাইডিং উইন্ডো টেকনিক ব্যাবহার করে। এতে সব ইন্ডেক্সের জন্য হিসাব করতে টোটাল $O(N)$ টাইম লাগবে । এরপর প্রত্যেক ইন্ডেক্স i এর জন্য l,r রেঞ্জের মধ্যে অলরেডি হিসেব করা ইন্ডেক্সগুলোর মধ্যে কোনটার আউটপুট সবচেয়ে বড় এটি খুঁজার জন্য আমরা সেগমেন্ট ট্রী ব্যাবহার করতে পারি , এতে প্রত্যেক ইন্ডেক্সের জন্য খুঁজতে $O(logN)$ সময় লাগবে এবং এরপর এই ইন্ডেক্সের জন্য সেগমেন্ট ট্রী তে আপডেট করতে $O(logN)$ সময় লাগবে । তাহলে সব ইন্ডেক্সের জন্য করতে টোটাল টাইম কমপ্লেক্সিটি হবে $O(N \times logN)$।

D. Easy Peasy, Lemon Squeezy

সাবটাস্ক 1 এর জন্য :

সাবটাস্ক ১ এর জন্য একটা অ্যারে রেখে মিনিমাম এবং ম্যাক্সিমাম বের করার জন্য পুরা অ্যারে ট্রাভার্স করা যেতে পারে। ওর্স্ট কেস কমপ্লেক্সিটি $O(n^2)$।

সাবটাস্ক 2 এর জন্য :

সাবটাস্ক ২ এর জন্য একটা মাল্টিসেট রেখে লগ ফ্যাক্টরে অ্যাড অথবা ইরেজ করা যেতে পারে। যেহেতু সেট অলরেডি সর্ট অবস্থায় থাকে, তাই মিনিমাম এবং ম্যাক্সিমাম O(1) এ বের করা যায়। সুতরাং এটার জন্য ওর্স্ট কেস কমপ্লেক্সিটি $O(n \times log(n))$।

সাবটাস্ক 3 এর জন্য :

যেহেতু রিমুভ অপারেশন শুধু ব্যাক থেকে, আমাদের ম্যাক্সিমাম বের করার সময় ছোট এবং মিনিমাম বের করার জন্য বড় এলিমেন্ট গুলা কনসিডার না করলেও চলে। কীভাবে? ধরে নেয়া যাক, আমাদের অ্যারে A = [5]। তখন 1, 2 3, 4, 5 ইত্যাদি অ্যাড করলে এদের ম্যাক্সিমাম এর উপর কোনো ইফেক্ট থাকে না। এখনও ম্যাক্সিমাম 5, যতক্ষণ না আমাদের অ্যারে এর শেষ থেকে 5 রিমুভ করা লাগছে। সুতরাং আমরা নতুন আসা এলিমেন্ট শেষে অ্যাড করার পরিবর্তে বর্তমান ম্যাক্সিমাম অর্থাৎ 5 অ্যাড করতে পারি। এখন যদি আমাদের 5 এর থেকে বড় ভ্যালু আসে, আমরা এখন থেকে সেই বড় ভ্যালুটি অ্যাড করা শুরু করবো। উদাহরণস্বরূপঃ আমাদের [5, 1, 2, 3, 4, 5, 7] একটার পর আরেকটা অ্যাড করতে হবে। তখন আমাদের ম্যাক্সিমাম এর অ্যারে হবে [5, 5, 5, 5, 5, 7]। ম্যাক্সিমাম পেতে আমাদের শুধু অ্যারে এর লাস্ট এলিমেন্ট চেক করা লাগবে।

মিনিমাম এর জন্যেই আমাদের একই কাজ করতে হবে। ধরে নেয়া যাক আমাদের অ্যারে A = [2]। তখন থেকে 2, 3, 4, 5 অ্যাড করাতে মিনিমাম এর উপর কোনো ইফেক্ট নেই। এখনও মিনিমাম 2, যতক্ষণ না আমরা এটাকে ব্যাক থেকে রিমুভ করছি। সুতরাং আমরা নতুন এলিমেন্ট অ্যাড না করে বর্তমান মিনিমাম, অর্থাৎ 2 অ্যাড করতে থাকবো। এখন যদি 2 এর থেকে ছোট কোনো এলিমেন্ট আসে, আমরা তখন থেকে সেই ছোট ভ্যালুটি অ্যাড করতে থাকবো। উদাহরণস্বরূপঃ আমাদের [2, 3, 4, 5, 1] একটার পর আরেকটা অ্যাড করতে হবে। তাহলে তখন আমাদের মিনিমাম অ্যারে হবে [2, 2, 2, 2, 1]। মিনিমাম পেতেও অ্যারে এর শুধুমাত্র শেষ এলিমেন্টটি চেক করা লাগবে।

এটা করার জন্য আমাদের এমন একটা ডেটা স্ট্রাকচার লাগবে যেটা শেষ থেকে $O(1)$ এ অ্যাড অথবা রিমুভ করতে পারে। আমরা এটার জন্য স্ট্যাক ইউজ করতে পারি। মিনিমাম আর ম্যাক্সিমাম এর জন্য আমাদের দুইটা আলাদা স্ট্যাক লাগবে। যেহেতু মিনিমাম / ম্যাক্সিমামও $O(1)$ এ বের করা যায়, তাই প্রবলেমটি আমরা ইনপুট টাইমে সল্ভ করতে পারি।

E. Next Permutation

সাবটাস্ক 1 এর জন্য:

সাবটাস্ক 1 এর জন্য আমরা 1 থেকে 9 দৈর্ঘ্যের যত পারমুটেশন সম্ভব সব বের করে রাখব। এই কাজটা অনেকভাবে করা যায়। সবচেয়ে সহজ হল C++ এর $next\_permutation()$ ফাংশন ব্যবহার করা। এই ফাংশন ব্যবহার করে লেক্সিগ্রাফিকালি পরবর্তী পারমুটেশন পাওয়া যায়।

সব সাবটাস্কের জন্য কিছু কথা:

সবার আগে আমরা দেখি কোন ক্ষেত্রে সমাধান অসম্ভব। ধরি, প্রদত্ত পারমুটেশনটা হল $p[0 \dots n-1]$ এবং এতে d সংখ্যক descent রয়েছে। যদি $d = n-1$ হয়, এটা সবচেয়ে বড় পারমুটেশন, সুতরাং এক্ষেত্রে পরবর্তী পারমুটেশনের অস্তিত্ব নেই। যদি $d = 0$ হয়, তবে এটা সবচেয়ে ছোট পারমুটেশন এবং বাকি সকল পারমুটেশনের অন্তত একটা descent রয়েছে, তাই এক্ষেত্রেও পরবর্তী পারমুটেশনের অস্তিত্ব নেই। অন্য ক্ষেত্রে পরবর্তী পারমুটেশনের অস্তিত্ব থাকবে না যদি এটা d descent বিশিষ্ট পারমুটেশনগুলোর মধ্যে সবচেয়ে বড় হয়। d descent বিশিষ্ট পারমুটেশনগুলোর মধ্যে সবচেয়ে বড়টা হল: $N, N-1, \dots, N-d+1, 1, 2, \dots, N-d$। এটা এমন একটা পারমুটেশন যার প্রথমে সবচেয়ে বড় d টা উপাদান বড় থেকে ছোট ক্রমে রয়েছে এবং বাকিগুলা ছোট থেকে বড় ক্রমে রয়েছে।

মনে করি আমাদের কাছে একটা বুলিয়ান ফাংশন $f(n, x, d)$ আছে যা বলে দেয় n সংখ্যক উপাদান নিয়ে d সংখ্যক descent অর্জন করা যায় কি না যেখানে এদের পূর্বের উপাদানের order হবে x (অর্থাৎ, পূর্বের উপাদান ঐ n সংখ্যক উপাদানের $x-1$ টা থেকে ছোট; আর স্পষ্টভাবে বললে, প্রথম উপাদানটা 1 থেকে $x-1$ এর মধ্যে হলে একটা descent তৈরি হবে, নতুবা নয়)। এই ফাংশন ব্যবহার করে আমরা পরবর্তী পারমুটেশনটা বের করব।

সাফিক্স ধাপ: প্রদত্ত পারমুটেশনটার একটা সাফিক্স বিবেচনা কর, আমরা শুধু এই অংশে পরিবর্তনটা আনতে চাই। যেহেতু সবচেয়ে ছোট পারমুটেশন বের করতে চাই এই সাফিক্সের দৈর্ঘ্যে যথাসম্ভব ছোট রাখতে হবে। $i = n-2$ থেকে শুরু করে $i = 0$ পর্যন্ত একটা লুপ চালাব, কোন i এর জন্য $p[i]$ থেকে $p[n-1]$ পর্যন্ত উপাদানগুলোকে কোনভাবে সাজিয়ে আমরা উদ্দেশ্য সাধন করতে চাই। ধরি আমাদের সমস্যার সমাধান হল $q[0 \dots n-1]$। এখানে অবশ্যই $p[i] < q[i]$ এবং i থেকে $n-1$ ইনডেক্স পর্যন্ত p তে যে উপাদানগুলো আছে, q তেও সেই উপাদানগুলো আছে। এখন p এর এই সাফিক্সের উপাদানগুলোকে ছোট থেকে বড় একটা একটা করে বিবেচনা করি। ধরি এখন z বিবেচনা করছি, তাহলে $q[i] = z$ এবং $z > p[i]$। এখন আমরা ফাংশন f ব্যবহার করে সমাধানের অস্তিত্ব যাচাই করে দেখতে চাই, তবে তার জন্য কত descent অর্জন করতে হবে তা খুঁজে বের করতে হবে। ধরি, $p[i]$ থেকে $p[n-1]$ এর মধ্যে মোট descent সংখ্যা d। তাহলে কি আমরা বলতে পারি না $q[i]$ থেকে $q[n-1]$ এর মধ্যে d সংখ্যক descent থাকতে হবে? তবে এখানে একটা কেস আছে; যদি $p[i-1] > p[i]$ এবং $p[i-1] < z$ হয়, তাহলে আমাদের বাম দিকে একটা descent কমে যাচ্ছে। এই descent টা অবশ্যই $q[i]$ থেকে $q[n-1]$ এর মধ্যে চলে আসতে হবে। সুতরাং, যদি $p[i-1] > p[i]$ এবং $p[i-1] < z$ হয়, d এর মান 1 বাড়িয়ে দিবে। এখন তাহলে $f(n-i-1, x, d)$সত্য হলে আমরা বলতে পারি, $q[i] = z$ ব্যবহার করে একটা সমাধান আছে এবং

আমরা সমাধান বের করার পর্বরতী বিল্ডিং ধাপে চলে যেতে পারব। কিন্তু x কত হবে? $p[i]$ থেকে $p[n-1]$ উপাদানগুলোর মধ্যে যতটা z হতে ছোট + 1।

বিল্ডিং ধাপ: এখন আমরা জানি $q[i] = z$ হবে। এরপরে $j = i + 1$ থেকে $j = n - 1$ পর্যন্ত একটা একটা করে উপাদান বসাতে থাকব। কোন একটা j এর জন্য অবশ্যই আমি এখানে সম্ভাব্য সবচেয়ে ছোট উপাদানটা বসাব। কোন একটা উপাদান স্থির করে, সমাধান সম্ভব কিনা যাচাই করার জন্য f ফাংশন ব্যবহার করতে পারব।

ফাংশন f আপাতত বিবেচনায় না আনলে, প্রতি টেস্টকেসে এই দুই ধাপের কমপ্লেক্সিটি দাঁড়ায় $O(n^2)$। এই দুই ধাপ সাবটাস্ক 2, 3 এবং 4 এর জন্য মোটামুটি একই। এখন আসি $f(n, x, d)$ কেমনে বের করবে তাতে।

সাবটাস্ক 2 এর জন্য:

$f(n, x, d)$ ফাংশনটা এই রিকারেন্স রিলেশন মেনে চলেঃ প্রত্যেক i $(1 \leq i \leq n)$ এর জন্য, সকল $f(n-1, i, d-less(i, x))$এর OR অর্থাৎ এদের মধ্যে অন্তত একটা সত্য হলেই কেবল $f(n, x, d)$ সত্য হবে। এখানে, যদি $i < x$ হয়, $less(i, x) = 1$ হবে অন্যথায় $less(i, x) = 0$ হবে।

DP দিয়ে এই কাজটা $O(n^4)$ কমপ্লেক্সিটিতে করা যায়। খেয়াল কর যে, এই DP প্রত্যেক টেস্টকেসের জন্য একই, সুতরাং কমপ্লেক্সিটি দাঁড়ায় $O(T \cdot n^2 + n^4)$।

সাবটাস্ক 3 এর জন্য:

উপরের DP এর স্টেট 3 টা, দেখা যাক ভিতরের লুপটা বাদ দেওয়া যায় কি না। খেয়াল কর যে, $i < x$ হলে, সবসময় আমরা $f(n-1, i, d-1)$ কল করি, অন্যথায় $f(n-1, i, d)$ কল করি। তাহলে, $g(n, x, d)$ হল, সকল $f(n-1, i, d-1)$ এর OR যেখানে $1 \leq ix$ । আর $h(n, x, d)$ হল, সকল $f(n-1, i, d)$ এর OR যেখানে $x \leq i \leq n$। অর্থাৎ g হল একটা প্রিফিক্স OR ফাংশন আর h হল একটা সাফিক্স OR ফাংশন। দুইটাই $O(n^3)$ কমপ্লেক্সিটিতে করা যাবে। তাহলে দাঁড়ায়, $f(n, x, d) = g(n, x, d)|h(n, x, d)$। সুতরাং f এর কমপ্লেক্সিটিতও $O(n^3)$ তে নেমে আসে। মোট কমপ্লেক্সিটি দাঁড়ায় $O(T \cdot n^2 + n^3)$।

সাবটাস্ক 4 এর জন্য:

এখন আমরা $f(n, x, d)$ এর মান $O(1)$ এ বের করা শিখব :p কেস এনালাইসিস করে আগানো লাগবে। n সংখ্যক উপাদানের মাঝে $n-1$ সংখ্যক descent থাকতে পারে, তবে x এর উপরে নির্ভর করে n টাও হতে পারে। প্রথমত, যদি $d > n$ অথবা $d < 0$ হয়, তবে সম্ভব না। যদি $d = 0$ হয়, তবে সকল উপাদান ছোট থেকে বড় ক্রমে থাকতে হবে এবং সবচেয়ে ছোট উপাদানটা x হতে বড় হতে হবে। অর্থাৎ, $d = 0$ হলে, উত্তর হবে $x == 1$। একইভাবে যদি $d = n$ হয়, তবে সকল উপাদান বড় থেকে ছোট ক্রমে থাকতে হবে এবং সবচেয়ে বড় উপাদানটা x থেকে ছোট হতে হবে। অর্থাৎ $d = n$ হলে, উত্তর হবে $x == n + 1$।

বাকি সব ক্ষেত্রে সমাধান সম্ভব! কিভাবে? সবগুলা সংখ্যাকে ছোট থেকে বড় ক্রমে বসিয়ে দিব। এখন যদি, $x > 1$ হয়, তবে ঐখানে একটা descent চলে আসবে এবং d এর মান 1 কমিয়ে দিব। এরপরে শেষের $d + 1$ সংখ্যক উপাদানকে আমরা উল্টিয়ে (reverse) দিব, ফলে d সংখ্যক descent চলে আসবে।

খেয়াল কর, এইখানে আমরা উপরে বর্ণিত বিল্ডিং ধাপের জন্য একটা এলগোরিদম পেলাম যেটা $O(n \log n)$ এ কাজ করে (চাইলে $O(n)$ এও করা যাবে)। এটা পরবর্তী সাবটাস্কের জন্য কাজে লাগবে।

সাবটাস্ক 5 এর জন্য:

এখন আমাদের $f(n, x, d)$ কাজ করে কন্সট্যান্ট টাইমে। আমাদের বিল্ডিং ধাপ কাজ করে লিনিয়ার টাইমে। বটলনেক হল সাফিক্স ধাপ, যেখানে এখনো $O(n^2)$ সময় লাগছে। ইনডেক্স i থেকে শুরু হওয়া একটা সাফিক্স ঠিক করার পরে, $p[i]$ থেকে বড় সকল উপাদান নিয়ে যাচাই করে দেখা লাগছে $q[i]$ তে এটা বসিয়ে সমাধান সম্ভব কি না। সকল উপাদান আসলে বিবেচনায় আনার দরকার নেই। বরং ব্যাপ্তির সীমানা (border case) গুলা নিয়ে যাচাই করলেই হবে :p

- $p[i]$ থেকে বড় উপাদানগুলোর মধ্যে সবচেয়ে ছোটটা
- $p[i]$ থেকে বড় উপাদানগুলোর মধ্যে সবচেয়ে বড়টা
- যদি $i > 0$ হয়, তবে $p[i-1]$ থেকে বড় উপাদানগুলোর মধ্যে সবচেয়ে ছোটটা (অবশ্যই এটা $p[i]$ থেকে বড় হওয়া লাগবে)

যেহেতু আমরা যাচাই কন্সট্যান্ট টাইমে করতে পারছি, আমাদের পুরো সমাধানটা $O(n \log n)$ এ চলে আসল। $f(n, x, d)$ তে x এর মান ইনপুট দেওয়ার জন্য আমাদের order জানিয়ে দেয় এমন একটা ডাটা স্ট্রাকচার লাগবে, সেটা হতে পারে Fenwick Tree বা তোমার পছন্দের কোন ডাটা স্ট্রাকচার।

F. Arrange the Bricks!

এই প্রবলেম টা মূলত একটা নাম্বার বাকি কয়টি নাম্বার দিয়ে বিভাজ্য, তা বের করার প্রবলেম। সেটা কিভাবে বের করতে হয় তা জানার আগে কেন সেটি কাজ করে তা আগে আমরা জানবো।

এখানে ইটের সাইজ/ডাইমেনশন হলো $1*2$। তাহলে, এই ডাইমেনশনের n টা ইট দিয়ে সাজানো আয়তক্ষেত্রের ক্ষেত্রফল কত হবে? অবশ্যই $n*2 = 2n$। এখানেই প্রব্লেমটির আসল মজা শুরু। একটা জিনিষ খেয়াল করে দেখো: আয়তক্ষেত্র টি তুমি যেভাবেই বানানোর চেষ্টা করো, এর ডাইমেনশন যদি $a*b$ হয়, তাহলে a ও b দুটি সংখ্যাই $2*n$ এর ডিভিজর হিসেবে থাকবে। কারণ, ডিভিজর হিসেবে না থাকলে অবশ্যই তাদের গুণ দিয়ে এই সংখ্যা বানানো সম্ভব না। না বুঝলে সম্পূর্ণ টা আবার পড়ার অনুরোধ রইলো।

এখন আসে প্রশ্ন, কিভাবে আমরা ডিভিজর বের করবো?

সাবটাস্ক ১ এর জন্যঃ

সাবটাস্ক ১ এর জন্য আমরা n^2 এর একটি লুপ চালিয়েই বের করে নিতে পারবো উত্তর।

সাবটাস্ক ২ এর জন্যঃ

এখানে n এর মান নিহাতই বড়। তাই ট্রিভিয়াল n^2 এখানে কাজ করবে না। এরজন্য প্রয়োজন 'Sieve of Eratosthenes'. পারো না? তাহলে এখনই গুগল করে শিখে নাও।

সাবটাস্ক ৩ এর জন্যঃ

এবার আসি সবচেয়ে মজার সলিউশন পার্টে। আশা করি, এই সাবটাস্ক সলিউশন পড়ার আগে তোমার কিছু সিভ(প্রাইম জেনারেটর) শেখা শেষ।

এখন দেখ, আমরা জানি (না জানলেও জেনে নাও এখন):

একটা নাম্বার n এর প্রাইম ডিভিজর $p_1, p_2, p_3, p_4......p_x$. এখানে p একেকটা প্রাইম সংখ্যা নির্দেশ করছে. তাহলে, n কে আমরা লিখতে পারি এভাবেঃ

$n = {p_1}^{a_1} * {p_2}^{a_2} * {p_3}^{a_3} * {p_4}^{a_4}.........{p_n}^{a_n}$

এখানে, ${p_x}^{a_x}$ দিয়ে বোঝানো হচ্ছে একটি প্রাইম p_x গুণ হিসেবে আছে a_x বার।

তাহলে আমাদের নাম্বার অফ ডিভিজরের সূত্র টা হবে এরকমঃ

$NOD = (a_1 + 1) * (a_2 + 1) * (a_3 + 1) * (a_4 + 1)......(a_n + 1)$

এখন কথা হচ্ছে, প্রাইম ডিভিজর বের করবো কিভাবে? উত্তরঃ সিভ দিয়ে।

কমপ্লেক্সিটি এনালাইসিসঃ এখানে n হতে পারে সর্বোচ্চ 10^14 পর্যন্ত। তাই আমাদের প্রাইম বের করা লাগবে 10^7 পর্যন্ত। 1 থেকে 10^7 পর্যন্ত প্রাইম আছে 664579 টি. তাই লুপ ঘুরবো প্রত্যেক টেস্টকেসের জন্য, সর্বোচ্চ বার 664579. তাই আমরা বলতে পারি, এর সাথে ১০০ টা টেস্ট কেস থাকলে প্রবলেম 0.1 সেকেন্ডই পাস করে যাবে।

ধন্যবাদ।

G. ACT 1: Ash Counting Trees

ধরো তুমি এখনো কোন নোড লেবেল করো নাই। তাহলে প্রথম নোডটা লেবেল করার জন্য কয়টা অপশন আছে? একটাই। সেই একটা নোড হলো রুট নোডটা।

সেই নোডটা লেবেল করার পর তার K-টা চাইল্ড নোডও অপশনে চলে আসে, কিন্তু সে নিজে আর অপশন হিসেবে থাকে না। তাই, অপশনের সংখ্যা $K - 1$ পরিমাণ বেড়ে যায়।

তার মানে, তুমি প্রথম নোডটা 1 উপায়ে লেবেল করতে পারো।

দ্বিতীয় নোডটা $1 + K - 1$ উপায়ে লেবেল করতে পারো।

তৃতীয় নোডটা $1 + K - 1 + K - 1 = 1 + 2 \times (K - 1)$উপায়ে লেবেল করতে পারো।

অর্থাৎ, তুমি i-তম নোডটা $1 + (i - 1)(K - 1)$ উপায়ে লেবেল করতে পারো। অতএব, $F(N, K) = \prod_{i=1}^{N} (1 + (i - 1) \times (K - 1))$

সাবটাস্ক ১

যেহেতু $N \leq 10^6$, আমরা $F(N, K)$ এর প্রত্যেক টার্ম বের করার জন্য জাস্ট একটা লুপ চালিয়ে দিতে পারি এবং তাকে যতবার সম্ভব P দ্বারা ভাগ করে তার মধ্যে থেকে P এর সকল ফ্যাক্টর সরিয়ে আমরা $\frac{F(N,K)}{P^Z}$ বের করতে পারি (যেখানে Z ম্যাক্সিমাম)।

এখানে একটা সম্ভাব্য ভুল হচ্ছে modulo অপারেশনের পরে P দিয়ে ভাগ দেওয়া, তবে তা সহজেই খেয়াল করা উচিত যেহেতু এই ভুলে স্যাম্পল ইনপুট/আউটপুটও মিলবে না।

টাইম কমপ্লেক্সিটিঃ $O(N \log_P N)$

সাবটাস্ক ২

এখানে $N \leq 10^{12}$, কিন্তু $K \leq 2$ মাত্র। তবে যখন $K = 1$, তখন কিন্তু উত্তর সবসময়ই 1

যখন $K = 2$? $F(N, K)$ আসলে $N!$ (N-এর ফ্যাক্টোরিয়াল)

এত বড় N এর জন্য ফ্যাক্টোরিয়াল বের করার ট্রিকটা আছে P^Q এর সাইজের মধ্যে। এই সাবটাস্কে $Q = 1$ অবশ্য, তাই modulo অপারেশনটা আল্টিমেটলি P দিয়েই হবে, অর্থাৎ প্রবলেমটা দাঁড়াচ্ছে $N! \pmod P$ বের করতে হবে (আসলে $\frac{N!}{P^Z} \pmod P$)।

যদি $N = 10$ হয়, তখন $N! = 1 \times 2 \times 3 \times 4 \times 5 \times 6 \times 7 \times 8 \times 9 \times 10$

ধরো $P = 3$, তাহলে প্রত্যেক P টার্ম পরে প্রত্যেক টার্মের mod ভ্যালু রিপিট করবে।

অর্থাৎ, $N! = 1 \times 2 \times 0 \times 1 \times 2 \times 0 \times 1 \times 2 \times 0 \times 1 \pmod{P}$

তার মানে, জাস্ট সাবটাস্ক ১-এর মত করে লুপ চালিয়েই $F(P-1,K)$ বের করে আমরা উত্তরের সাথে $F(P-1,K)^{\frac{N}{P}}$ গুণ করে দিতে পারি (যেহেতু $F(P-1,K)$ $\frac{N}{P}$ বার রিপিট হবে)। কিন্তু তাতে শেষের $(N \pmod{P})) - 1$ সংখ্যক টার্ম মিসিং থাকবে, সেগুলোও লুপ চালিয়ে সহজে উত্তরের সাথে গুণ করে ফেলা সম্ভব।

কিন্তু আরো কিছু টার্ম তো আসলে মিসিং আছে, যেমন 6। একে P দ্বারা ভাগ করলে তো 2 পাওয়া যায় যা আমাদের উত্তরের সাথে গুণ করা হয়নি। আসলে P এর সকল গুণিতকেরই একই অবস্থা। এই গুণিতকগুলো কারা আসলে?

P, $2P$, $3P$, $4P$, $5P$, $6P$.....

এদের সবার মধ্যে কিন্তু P সাধারণ পদ (অর্থাৎ কমোন নেওয়া যায়)। তাহলে এদের সবাইকে P দ্বারা ভাগ করলে আমরা কী পাই?

1, 2, 3, 4, 5, 6.... যাদেরকে গুণ করলে আবার সেই ফ্যাক্টোরিয়ালই আসতেছে!

এরকম $\frac{N}{P}$ সংখ্যক টার্ম থাকবে। অর্থাৎ, $F(\frac{N}{P}, K)$ উত্তরের সাথে গুণ করতে হবে। তার মানে প্রবলেমটা একটা ক্ষুদ্রাকার সাব-প্রবলেমে কনভার্ট হচ্ছে। এই পার্টটা রিকার্সিভলি এবং ইটারেটিভলি দুইভাবেই সল্ভ করা সম্ভব। এখানে বেস কেস হবে $N < P$, যা লুপ চালিয়েই বের করা সম্ভব। রিকার্সনের লেভেল সংখ্যা হবে $\log_P N$

টাইম কমপ্লেক্সিটিঃ $O(P \log_P N)$

সাবটাস্ক ৩

এখন যেহেতু $K \leq 10^6$, $F(N,K)$ ফ্যাক্টোরিয়াল নাও হতে পারে।

প্রথমত, $(K-1)$ যদি P দ্বারা নিঃশেষে বিভাজ্য হয় তখন কী হবে?

এক্ষেত্রে $1 + (i-1) \times (K-1) \equiv 1 \pmod{P}$, তাই উত্তর সব সময়ই 1 হবে। তার মানে $(K-1)$ যদি P দ্বারা বিভাজ্য না হয়, সেটাই বেশি চিন্তার বিষয়।

ধরো $K = 6$ এবং $P = 3$

তাহলে $F(N,5) = 1 \times 6 \times 11 \times 16 \times 21 \times 26 \times 31 \times 36 \times 41 \times 46 \times 51 \times 56 \times 61 \times 66.....$

$P = 3$ দ্বারা mod করার পর?

$F(N,5) = 1 \times 0 \times 2 \times 1 \times 0 \times 2 \times 1 \times 0 \times 2 \times 1 \times 0 \times 2 \times 1 \times 0..... \pmod{3}$

এক্ষেত্রেও P ঘর পর পর কিছু টার্ম রিপিট হচ্ছে! কেন? এটাও বেসিক মডুলার অ্যারিথমেটিক দ্বারা প্রমাণ করা সম্ভব।

আমাদের i-তম টার্মটা কী? $1 + (i-1) \times (K-1)$

এটাকে P দ্বারা mod করলে কী পাই?

$1 + (i-1) \times (K-1) \equiv 1 + ((i-1) \pmod{P}) \times (K-1) \pmod{P}$

তাই, যেকোন $i > P$ এর জন্য i-তম টার্মটা কোন একটা j-তম টার্মের সমান হবে যেখানে $j = i \pmod{P}$

এখন, $F(N,5) \pmod{P}$ বের করতে হলে আমরা সাবটাস্ক ২ এর মত একটা লুপ চালিয়ে রিপিটিং পার্টটা বের করব এবং তার এক্সপোনেনশিয়েশন (সূচক বা পাওয়ার) উত্তরের সাথে গুণ করব। আগেরবারের মত বাকি অংশও আমরা লুপ চালায়ে বের করতে পারব। কিন্তু যেসব টার্ম P দ্বারা বিভাজ্য?

আমাদের উদাহরণে এই টার্মগুলো হচ্ছে 6, 21, 36, 51, 66...

কিন্তু আগেরবারের সাথে এবার একটা পার্থক্য আছে। এবার আমরা যদি এই টার্মগুলোকে P দ্বারা ভাগ করি তাহলে প্রথম টার্মটা 1 হয় না। বরং টার্মগুলো এমন হয়ঃ 2, 7, 12, 17, 22...

এবারও টার্মগুলোর মধ্যে ডিফারেন্স একই থাকতেছে, তা হলো $K - 1 = 5$.

তার মানে আমাদের নতুন সাব-প্রবলেমও আগেরটার মতই, শুধু প্রথম টার্মটা আলাদা। কিন্তু তা কী আসলেই সমস্যা? ইমপ্লিমেন্টেশন একটু কঠিন আগের থেকে, কিন্তু অতটা কঠিনও না।

রিকার্শনের যেকোন স্টেটে আমরা লুপের মাধ্যমে সহজেই প্রথম কোন টার্ম P দ্বারা বিভাজ্য তা বের করতে পারি। তারপর তা P দিয়ে ভাগ করলেই আমরা নতুন সাব-প্রবলেমের প্রথম টার্মটা পেয়ে যাচ্ছি। রিকার্সিভলি এটার ইমপ্লিমেন্টেশন অপেক্ষাকৃত সহজ হওয়ার কথা, সেক্ষেত্রে এই প্রথম টার্ম এবং টার্মের সংখ্যা রিকার্শনের প্যারামিটার হিসেবে রাখা উচিত।

টাইম কমপ্লেক্সিটিঃ $O(P \log_P N)$

সাবটাস্ক ৪

এবার $Q \geq 1$, তাই আরো কয়েকটা জিনিস খেয়াল করতে হবে।

প্রথমত, $(K-1)$ P দ্বারা বিভাজ্য হলেই উত্তর আর 1 না। তবে এটা নিশ্চিত যে এক্ষেত্রেও আগের মত প্রত্যেক P^Q টার্ম পর পর একটা রিপিটেশন থাকবে যা আগের মত করে প্রমাণ করা সম্ভব। তাছাড়া $(K-1)$ P দ্বারা বিভাজ্য হলে এটাও নিশ্চিত যে $F(N, K)$ এর কোন টার্মই P দ্বারা বিভাজ্য হবে না (সহজেই প্রমাণ করা সম্ভব)। তাই, রিপিটিটিভ পার্টটা লুপ দিয়ে বের করে সহজেই আগের মত করে উত্তর বের করা সম্ভব।

যখন $(K-1)$ P দ্বারা বিভাজ্য না, তখন কিছু টার্ম P দ্বারা বিভাজ্য হতে পারে এবং সেগুলোকে আগের মত করেই সামলাতে হবে। তবে এবার একটা জিনিস খেয়াল করতে হবে যে রিপিটিশন আর P টার্ম পর পর হচ্ছে না, বরং P^Q টার্ম পর পর হচ্ছে। তাই আমরা এই পুরো রিপিটেড পার্টটা এমনভাবে নিব যেন তার মধ্যে কোন P দ্বারা বিভাজ্য টার্ম না থাকে এবং আগের মতই তার এক্সপোনেনশিয়েন ও বাদবাকি টার্মগুলো উত্তরের সাথে গুণ করব।

আর যেসব টার্ম P দ্বারা বিভাজ্য সেগুলো? আমরা যদি ওগুলোকে P দ্বারা ভাগ করি, তাহলে আবার সাবটাস্ক ৩ এর মতই সাবপ্রবলেম পাব।

টাইম কমপ্লেক্সিটিঃ $O(P^Q \log_P N)$

H. Karim Meets Gollum

1. আমরা শুরুতেই $\lceil\sqrt{\text{MAXN}}\rceil$ পর্যন্ত সিভ করে 1 থেকে $\sqrt{\text{MAXN}}$ এর মধ্যে সকল প্রাইম নম্বর বের করে একটি অ্যারেতে স্টোর করে রাখতে পারি। এই কাজ টির কমপ্লেক্সিটি হল $O(\sqrt{\text{MAXN}} \ln\ln\sqrt{\text{MAXN}})$ ।

2. তাহলে প্রতি কুয়েরির N কে শুধু ঐ প্রাইম নম্বরের লিস্টের উপর ইটারেট করেই তার প্রাইম ফ্যাক্টর গুলো বের করা যাবে। এই কাজটির কমপ্লেক্সিটি হল $O(\pi(\sqrt{\text{MAXN}}))$ যেখানে $\pi(x)$ হল প্রাইম কাউন্টিং ফাংশন ($\pi(x) = x$ এর সমান বা ছোট প্রাইম নম্বরের সংখ্যা)। এবং $\pi(10^8) = 5761455$ ।

3. মনে করি, N এর কোন একটি প্রাইম ফ্যাক্টর p এবং তা সর্বোচ্চ α বার N কে ভাগ করে।

4. এখন প্রতি প্রাইম p এবং তার মাল্টিপ্লিসিটি α এর জন্য আমরা $s(p, \alpha)$ ফাংশনের মান বের করব, যেখানে $s(p, \alpha) = \sum_{i=0}^{\alpha} p^{Ki} \bmod (10^9+7)$ । এই কাজটি করতে প্রতি প্রাইমের জন্য আমাদের কমপ্লেক্সিটি হবে $O(\lg K + \lg \alpha)$ ।

5. এখন N এর সবগুলো প্রাইম ফ্যাক্টরের জন্য s এর মান গুণ করলেই ($\bmod(10^9+7)$ এ) আমরা আমাদের কুয়েরির উত্তর পেয়ে যাব। এইটি করতে আমাদের সময় লাগবে, $O(\lg K \times \lg N + \lg N)$ ।

6. সুতরাং সবগুলো কুয়েরি এর উত্তর করতে আমাদের সময় লাগবে, $O(\sqrt{\text{MAXN}} \ln\ln\sqrt{\text{MAXN}} + Q \times ((1 + \lg \text{MAXK}) \times \lg \text{MAXN} + \pi(\sqrt{\text{MAXN}})))$ ।

I. Array Partition

একটি গুরুত্বপূর্ন অ্যারের পার্টিশন এ একটি unique non-decreasing sequence বের করা যায়। সেটা হল প্রতি সাবঅ্যারে থেকে আমরা greedily next smallest একটি করে উপাদান নিতে হবে। যেমন $[2,2,3],[2,3,4],[1,3,4]$ থেকে greedily $2,2,3$ অনুক্রম টি নেওয়া হবে। unique sequence গুলো থেকে গুরুত্বপূর্ণ পার্টিশনগুলো গণনা করা হবে।

কাউন্ট করার জন্য ডাইনামিক প্রোগ্রামিং পদ্ধতি অবলম্বন করা হবে। ধরি $dp[pos][num] =$ প্রথম pos টি উপাদান ব্যবহার করে কতগুলো গুরুত্বপূর্ণ পার্টিশন করা যায় যেখানে sequence টির শেষ উপাদান হল num। যেমন $[3,1,2,4]$ এ $dp[4][4] = 3$, কারণ $[3,1,2],[4],[3,1],[2],[4],[3],[1,2,4]$ এই 3টি।

সাবটাস্ক ১ঃ

একটি n^4 সলিউশন লেখা যায় যেখানে $dp[num][pos]$ বের করতে pos থেকে কিছুটা পেছানো হবে, pos' এর আগ পর্যন্ত। তারপর এমন সকল $dp[num'][pos']$ এর যোগফল নিতে হবে যেন $dp[num'][pos']$ এর একটি sequence এর শেষ উপাদান num এর পর $[pos'+1.....pos]$ সাবঅ্যারে থেকে next smallest উপাদান num হয়। এক্ষেত্রে খেয়াল রাখতে হবে যেন, $[pos'+1...pos]$ এর ভিতরে num' এর চেয়ে সমান বা বড় উপাদান num ই হয়।

সাবটাস্ক ২ঃ

আগের সলিউশন টিকে n^3 বানানো যায় যদি খেয়াল করা হয় যে একটা নির্দিষ্ট pos' এ আসলে num' এর একটি রেঞ্জ পাওয়া যায়, তা হল, $[x+1, num]$, যেখানে x হল $[pos'+1...pos]$ এর ভিতরে num থেকে ছোট সবচেয়ে বড় সংখ্যা । অর্থাৎ ওই রেঞ্জে থাকা সব num' এর জন্যই $dp[num'][pos']$, $dp[num][pos]$ এর সাথে যোগ হবে। যেহেতু এই রেঞ্জ এর সংখ্যা গুলো আগেই কম্পিউট করা হয়ে গেছে, তাই একটি cumulative sum এর অ্যারের সাহায্যে $O(1)$ টাইম এ রেঞ্জ সামটি পাওয়া যাবে। আবার একটি জিনিস খেয়াল রাখতে হবে, তা হল pos' যাতে এমন হয় যেন $[pos'+1...pos]$ এ কমপক্ষে একটি num থাকে।

সাবটাস্ক ৩ঃ

সলিউশন টিকে n^2 বানাতে হলে আরো অবজার্ভ করতে হবে যে $dp[pos][num]$ বের করতে গিয়ে আসলে যে সকল $dp[pos'][num']$ যোগ করা হয় তারা আসলে শুধু একটি রেঞ্জই না বরং কিছুটা rectangular range (dp table এ) এর মত। হাতে কলমে একটি 2d dp table বানিয়ে এটি ভালো মত বোঝা যায় যে $dp[pos][num]$ এ কারা কন্ট্রিবিউট করে। যেটা করতে হবে তা হল আগে এমন p বের করতে হবে যার জন্য $[p+1...pos]$ এ শুধু মাত্র $p+1$ অবস্থানেই num আছে (p এর চেয়ে বড় pos' গুলো কোন কন্ট্রিবিউশন রাখতে পারবে না) । এরপর প্রথমেই সব $dp[pos'][num]$ যোগ করা হবে যাদের জন্য $pos' \le p$ এবং $num' \le num$, এটি 2d cumulative sum এর সাহায্যে $O(1)$ এ করা যাবে। এতে কিছু ওভারকাউন্ট হয়েছে, যে যে pos', num' এর জন্য $pos' \le pos'' \le pos$ এবং $num' \le num'' \le num$ আছে যেন $A[pos''] = num''$,তাদের $dp[pos'][num']$ ওভারকাউন্ট হয়েছে। তাই একটি pos নির্দিষ্ট করে যখন $num = 1...n$ এর জন্য $dp[pos][num]$ নির্ণয় করা হবে তখন একটি stack মেইন্টেইন করতে করতে হবে যার ভেতরে num'', pos'' গুলো থাকবে, decreasing order এ, এই stack টি আপডেট করার সময়ই ওভারকাউন্ট করা অংশ গুলি বিয়োগ করে দিতে হবে। (সেটাও 2d cumulative sum এর মাধ্যমে)। এতে সলিউশনটি n^2 হবে।

J. Hououin Kyouma !

সাবটাস্ক ১ঃ

সবসময় $Q >= T$ এখানে । তাই প্রথমে T টা প্রশ্ন করে কোন কোন ঘণ্টায় ক্লাস এবং কোন কোন ঘণ্টায় ক্লাস নেই, তা বের করে ফেলা যাবে । এরপর একটি লিনিয়ার লুপ চালিয়েই রেজাল্ট বের করা

যাবে ।

সাবটাস্ক ২ঃ

ক্লাস টাইমের মত ফ্রী টাইমেরও $[t_1, t_2]$ রেঞ্জ আছে বলে ধরা যায়, যেখানে ফ্রী টাইম t_1-তম ঘণ্টা থেকে শুরু হয়ে t_2-তম ঘণ্টায় শেষ হয় । আমরা যদি এরকম সব ফ্রী টাইমের রেঞ্জ বের করতে পারি, তাহলে তাদের মাঝে যে রেঞ্জের দৈর্ঘ্য সর্বাধিক, সেই দৈর্ঘ্যই আমাদের রেজাল্ট ।

প্রোপার্টি ১ঃ যদি $[l, r]$ রেঞ্জের সম্পূর্ণ সময়ে ক্লাস না থাকে, তাহলে এই রেঞ্জের ফ্রী টাইম হবে $= (r - l + 1)$ ঘণ্টা ।

প্রোপার্টি ২ঃ $[l, r]$ রেঞ্জের সম্পূর্ণ সময়ে যদি ক্লাস থাকে, তাহলে এই রেঞ্জের ফ্রী টাইম হবে $= 0$ ঘণ্টা ।

এখন আমরা আমরা যদি কোনো সময় t_i-তম ঘণ্টায় থাকি এবং সেটি ক্লাস টাইম হয়, তাহলে প্রোপার্টি ২ ব্যবহার করে বাইনারী সার্চ চালিয়ে এই ক্লাস কত তম ঘণ্টায় শেষ হয়, সেটা খুব সহজেই বের করে ফেলা যায় । ক্লাস টাইম যততম ঘণ্টায় শেষ, ঠিক টার পরের ঘণ্টাই ফ্রি টাইম হবে ।

t_i-তম ঘণ্টা ফ্রী টাইম হলে কী করতে হত, সেটা হোমটাস্ক ;) ।

এভাবে আমরা খুব সহজেই বাইনারী সার্চের মাধ্যমে 1^{st} ঘণ্টা থেকে T পর্যন্ত ক্লাস টাইম, ফ্রী টাইমের সব রেঞ্জ বের করে ফেলতে পারব । যেহেতু, প্রতি ক্লাসের শুরু এবং শেষ ঘণ্টা বের করতে বাইনারী সার্চ লাগছে, তাই সর্বমোট প্রশ্ন দরকার হবে $\leq 2 \times N \times log_2(T)$, যা 5×10^4 এর প্রশ্নসীমায় হয়ে যায় ।

সাবটাস্ক ৩ঃ

সাবটাস্ক ৩ এর সলিউশন মূলত সাবটাস্ক ২ এর সলিউশনের অপটিমাইজেশন । সাবটাস্ক ২ এ আমরা যখন বাইনারী সার্চ চালাচ্ছিলাম, তখন এক বাইনারী সার্চ থেকে পাওয়া ইনফর্মেশন আমরা কিন্তু অন্য কোনো বাইনারী সার্চে কাজে লাগাইনি । এখন আমরা এটি কাজে লাগাব । এই সাবটাস্কে আমরা সব প্রশ্ন $[1, t]$ এই ফরম্যাটে করব অর্থাৎ, 1^{st} ঘণ্টা থেকে t-তম ঘণ্টা পর্যন্ত টোটাল ফ্রী টাইম কত - সেটা জিজ্ঞাসা করব । এর কারণ একটু পরেই আমারা বুঝতে পারব ।

প্রোপার্টি ৩ঃ $[1, t_1 - 1]$ রেঞ্জে টোটাল ফ্রী টাইম T_1 ঘণ্টা এবং `$[1, t_2]$` রেঞ্জে টোটাল ফ্রী টাইম T_2 ঘণ্টা হলে $[t_1, t_2]$ রেঞ্জে টোটাল ফ্রী টাইম $(T_2 - T_1)$ ঘণ্টা, যেখানে $1 < t_1 \leq t_2 \leq T$ ।

ধরা যাক, 1^{st} ঘণ্টা থেকে ক্লাস টাইম, ফ্রী টাইম বের করতে করতে আমরা এখন t_i-তম ঘণ্টায় আছি । তার মানে $[1, t_i - 1]$পর্যন্ত টোটাল ফ্রী টাইম কত, সেটাও আমাদের জানাই আছে ।

ধরি, t_i-তম ঘন্টাটি ফ্রী টাইম এবং আগের কোনো এক বাইনারী সার্চ থেকে $[1, t_j]$ রেঞ্জের টোটাল ফ্রী টাইম কত সেটা বের করা আছে, যেখানে $t_j > t_i$ । তাহলে প্রোপার্টি ৩ ব্যবহার করে কোনো এক্সট্রা প্রশ্ন জিজ্ঞাসা না করেই $[t_i, t_j]$ রেঞ্জের ফ্রী টাইম কত, সেটা আমরা বের করে ফেলতে পারব ।

এই ফ্রী টাইম যদি $(t_j - t_i + 1)$ হয়, তার মানে $[t_i, t_j]$ রেঞ্জের পুরোটাই ফ্রী টাইম এবং এতে করে আমরা t_j-তম ঘণ্টার পর কোথায় ফ্রী টাইম শেষ হয়, সেটা খুঁজতে পারি । আর যদি ফ্রী টাইম $(t_j - t_i + 1)$ না হয়, তার মানে $[t_i, t_j]$ রেঞ্জের মাঝেই কোথাও ক্লাস টাইম আছে । অর্থাৎ, t_i-তম ঘন্টায় থাকা ফ্রী টাইম $[t_i, t_j]$ রেঞ্জের কোনো এক ঘণ্টাতেই শেষ হবে । তাই আমরা এই রেঞ্জের ভিতরেই সেই শেষ ঘণ্টা খুঁজব ।

t_i-তম ঘন্টাটি ক্লাস টাইম হলে কী করতে হত, সেটা হোমটাস্ক :D ।

এভাবে আগের বাইনারী সার্চ থেকে পাওয়া ইনফর্মেশন কাজে লাগিয়ে আমরা পরবর্তী বাইনারী সার্চ চালানোর রেঞ্জগুলো ছোট করে ফেলতে পারি । এখন প্রশ্ন হচ্ছে, টোটাল প্রশ্নের সংখ্যা এভাবে কত কমবে?

প্রথমে 1_{st} ঘণ্টা থেকে প্রথম ক্লাস কততম ঘণ্টায় শুরু হয়, সেটা বের করতে $\log_2(T)$ সংখ্যক প্রশ্ন লাগবে । এরপর উপরের নিয়মে রেঞ্জ ছোট করার পর বাইনারী সার্চ চালাতে হবে এমন সবচেয়ে বড় রেঞ্জের দৈর্ঘ্য হবে $\frac{T}{2}$ । এমন দৈর্ঘ্যের রেঞ্জ ম্যাক্সিমাম 2 টা হওয়া সম্ভব (কারণ $2 \times \frac{T}{2} = T$) । তাহলে, আরও $2 \times \log_2(\frac{T}{2})$ সংখ্যক প্রশ্ন লাগবে । এভাবে এগোতে থাকলে দেখা যায়, টোটাল প্রশ্নের সংখ্যা = $\log_2(T) + 2 \times \log_2(\frac{T}{2}) + 4 \times \log_2(\frac{T}{4}) + ... + k \times \log_2(\frac{T}{k})$, যেখানে $k = ceil(\log_2(2 \times N))$ ।

এই প্রয়োজনীয় প্রশ্নের সংখ্যা 4.3×10^4 এর চেয়ে কম । আরও একুরেট ক্যালকুলেশন করলে দেখা যায়, প্রশ্নসংখ্যা কখনোই 41843 এর বেশি দরকার হবে না, বরং কেয়ারফুল থাকলে আরও কম লাগবে ।

K. The Hidden Island

সাবটাস্ক ১ঃ

হারানো অক্ষরগুলোতে সর্ট করে নেই। এখন আমরা ? পজিশন গুলোতে বাম থেকে ডানে এক বা একাধিক হারানো অক্ষর বসিয়ে সব ধরনের স্ট্রিং বানানর চেষ্টা করি। স্ট্রিং গুলোর মধ্যে সবচেয়ে ছোটটিই বেছে নিলেই হবে।

Time Complexity: $O(|S| * (2^K))$

সাবটাস্ক ২ঃ

আমরা Dynamic Programming দিয়ে এটি সমাধান করতে পারি। স্টেট হিসেবে আমরা রাখব স্ট্রিং S এর পজিশন এবং K থেকে কতগুলো অক্ষর ব্যবহার করেছি। এতে আমরা প্রতিটি |S| * K টি স্টেট এর জন্য স্ট্রিং বানিয়ে ফেলতে পারি।

Time Complexity:$O(|S| * K * |S|)$

সাবটাস্ক ৩ঃ

ধরি, আমাদের "?" আছে X টা। আমরা একটা লিস্ট L রাখব যেখানে শুরুতে (K - X) টা হারানো অক্ষর রাখব। হারানো অক্ষর লিস্টে বর্ণানুক্রমে রাখব।

আমরা একটা ফাকা স্ট্রিং R নিই। S স্ট্রিং টির আমরা বাম থেকে ডানে যেতে থাকি। যখনই S স্ট্রিং টিতে আমরা একটা ইংরেজি অক্ষর পাব, আমরা সেটা R এ যোগ করে দিব। যদি "?" পাই, তাহলে হারানো অক্ষর থেকে L লিস্টে একটি অক্ষর যোগ করে দিব।

এখন "?" এ অবশ্যই একটি অক্ষর বসবে, তাই আমরা লিস্ট সবচেয়ে ছোট অক্ষরটি R এ বসাব। এখন আমরা লিস্ট থেকে আরও কিছু অক্ষর বসাতে পারি R এ। তবে সেটি নির্ভর করে "?" এর পরের অক্ষরটি কি তার উপর ।

ধরি, "?" এর পরের অক্ষরটি 'D'. তাহলে বর্ণানুক্রমে ছোট স্ট্রিং বানানর জন্য আমাদের উচিত লিস্টে থাকা 'D' এর চেয়ে ছোট সবগুলো অক্ষর R এ যোগ করা। 'D' এর চেয়ে বড় কিছু আমাদের স্ট্রিং এর এ পর্যায়ে যোগ করা ঠিক না। কারণ এর চেয়ে পরের 'D' টি নিয়েই আমরা বর্ণানুক্রমে ছোট পেতে পারি।

আমাদের লিস্টে যদি কিছু 'D' থাকে, সেটা যোগ করব কিনা, সেটা দেখা যাক।

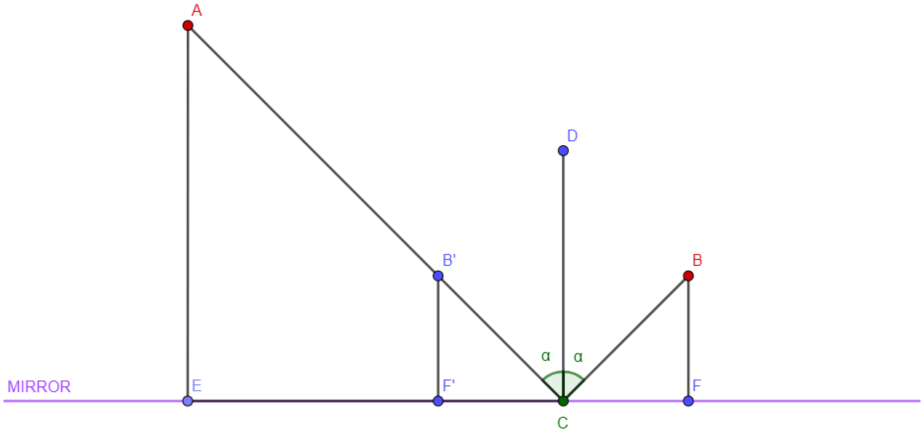
ধরি, স্ট্রিংটি হল ?DDA... । আমরা অবশ্যই একটি অক্ষর বসানোর নিয়মটি এখন ভুলে যাই। যদি প্রথম ? এ আমরা 'D' বসাই তাহলে আমরা পাব, DDDA.... । যদি প্রথমে না বসাই, পাব DDA.... । দ্বিতীয়টি বর্ণানুক্রমে ছোট। আবার ধরি, স্ট্রিংটি হল ?DDE... । যদি প্রথম ? এ আমরা 'D' বসাই তাহলে আমরা পাব, DDDE... । যদি প্রথমে না বসাই, পাব DDE.... । প্রথমটি বর্ণানুক্রমে ছোট। তাই

আমাদের ? এর পরে এমন 'D' ব্যতীত প্রথম অক্ষর জানতে হবে। অক্ষর ছোট হলে, এখন না বসানো শ্রেয়। বড় হলে, এখনই 'D' বসানো শ্রেয়।

এটি আমরা একটি লুপ দিয়েই করতে পারি।

Time Complexity: $O(|S| + K)$

L. Magic Mirror on the Wall



ধরো, ক্যাবলাকান্ত A বিন্দুতে এবং চালাকান্তী B বিন্দুতে আছে। আলোর প্রতিফলনের সূত্রানুযায়ী, C বিন্দুতে ক্যাবলাকান্ত চালাকান্তীকে দেখতে পেলে $\angle ACD = \angle BCD$ হতে হবে। আবার যেহেতু CD আয়নার তলের উপর লম্ব, $\angle ACE = \angle BCF$ হবে। B', B এর এবং F', F এর সদৃশ বিন্দু হলে, সদৃশ ত্রিভুজের গুণাবলী অনুযায়ী, $\frac{B'F'}{AE} = \frac{CF'}{CE}$। উল্লেখ্য, $AE, BF, B'F'$ আয়নার তলের উপর লম্ব।

এখানে, $\frac{B'F'}{AE} = \frac{CF'}{CE}$

বা, $\frac{B'F'+AE}{AE} = \frac{CF'+CE}{CE}$

বা, $\frac{B'F'+AE}{AE} = \frac{EF}{CE}$

বা, $CE = EF(\frac{AE}{B'F'+AE})$

বা, $CE = (x_2 - x_1)(\frac{y_1}{y_1+y_2})$

সুতরাং $C(x,\ y) \equiv (x_1 + (x_2 - x_1)(\frac{y_1}{y_1+y_2}),\ 0)$