সমস্যার বিশ্লেষণ:

এনএইচএসপিসির বাছাই পর্ব এবং মূল প্রতিযোগিতা অনলাইনে হওয়ার কারণে উভয় প্রতিযোগিতার সমস্যাগুলোর জটিলতার মাত্রাকে সামঞ্জস্যপূর্ণ রাখা হয়। বাছাই পর্বে প্রতিযোগীদেরকে পাঁচ ঘন্টায় মোট ১২টি সমস্যার সমাধান করতে দেয়া হয়। প্রতিযোগীদের উৎসাহের কারণে বাছাই পর্বের সময়সীমা ৩০ মিনিট বর্ধিত করা হয়। বাছাই পর্বের ১২টি সমস্যার প্রত্যেকটিতে আংশিক নম্বর তোলার সুবিধা রাখা হয়। এই সুবিধা প্রদানের জন্য আন্তর্জাতিক ইনফরমেটিক্স অলিম্পিয়াডের মান অবলম্বন করে প্রতিটি সমস্যাকে বেশ কয়েকটি ভাগে ভাগ করা হয়। এই প্রতিটি ভাগকে একটি সাবটাস্ক হিসেবে নির্ধারিত করা হয়। একটি সমস্যার জন্য প্রদত্ত ১০০ নম্বরকে ওই সমস্যাটির বিভিন্ন সাবটাস্কের ভিতরে বন্টন করা হয়। এর মাধ্যমে একটি কঠিন সমস্যাকে সকল পর্যায়ের প্রতিযোগীদের জন্য উপযুক্ত করা হয়। বাছাই পর্বের ১২ টি সমস্যার বিষয়বস্তু নির্ধারণের জন্য আন্তর্জাতিক ইনফরমেটিক্স অলিম্পিয়াডের সিলেবাসকে অবলম্বন করা হয়। সমস্যাগুলো তৈরী করতে গ্রাফ থিওরি, নাম্বার থিওরি, জ্যামিতি, ডাটা স্ট্রাকচার, রৈখিক বীজগণিত, ম্যাক্সিমাম ফ্লোসহ প্রয়োজনীয় আলগরিদম সমূহের সহায়তা গ্রহণ করা হয়। সমস্যাগুলো তৈরির সময় প্রতিযোগীদের দক্ষতা বৃদ্ধির জন্য তাদের পুঁথিগত বিদ্যাকে প্রাধান্য না দিয়ে চিন্তাশক্তিকে প্রয়োগ করার সুযোগ করে দেয়া হয়। আন্তর্জাতিক মানের সাথে সঙ্গতি রেখে বাছাই পর্বে দুটি ইন্টারএকটিভ সমস্যা অন্তর্ভুক্ত করা হয়. বাছাই পর্বের সাথে সঙ্গতি রেখে মূল প্রতিযোগিতাতেও ১২টি সমস্যা রাখা হয়। বাছাই পর্বের বিষয়সমূহের পাশাপাশি মূল পর্বে ডাইনামিক প্রোগ্রামিং এবং স্ট্রিং ম্যাচিংকেও অন্তর্ভুক্ত করা হয়। বাছাই পর্বের মত মূল পর্বেও প্রতিটি সমস্যার জন্য বেশ কয়েকটি সাবটাস্ক তৈরী করা হয়। মূল পর্বে একটি ইন্টারেকটিভ সমস্যা সমাধান করতে দেয়া হয়।

বাছাই এবং মূল পর্বের সমস্যাগুলোর সংক্ষিপ্ত বিশ্লেষণ দেওয়া হলো:

নাম : **Simple And**
সেটার: Hasibul Haque Himel (fire_tornado)
অল্টারনেট লেখক : Mahmud Sajjad Abeer （msabeer）
মূল ধারণা: DSU on Tree/ Bfs-Dfs-SparseTable

Subtask 1: $1 \le N, Q \le 10^4$

এই সাবটাস্কের জন্য, আমাদের কেবল ডিএফএস এর সহজ ধারণা প্রয়োজন। প্রথমে রুট থেকে, আমাদের একটি ডিএফএস চালানো উচিত যা প্রতিটি নোডের প্যারেন্ট নির্ধারণ করে। তারপরে প্রতিটি প্রশ্নের জন্য (u, k) আমাদের নোড u থেকে অন্য একটি ডিএফএস করা উচিত যা এর সাবট্রির k গভীরতায় যায় এবং বিটওয়াইস অ্যান্ড গণনা করে।

Subtask 2: $k \le 1$

পূর্ববর্তী সাবটাস্ক মত, প্রথমে রুট থেকে, আমাদের একটি ডিএফএস করা উচিত যা প্রতিটি নোডের প্যারেন্ট নির্ধারণ করে। যদি $k = 0$ হয় তবে উত্তর হবে নোডে উপস্থিত নম্বর । যদি $k = 1$ হয় তবে আমাদের কেবল নোড u এর সংলগ্ন

অ্যাডজেসেন্সি লিস্ট দেখতে হবে এবং এটির প্যারেন্ট নোড ব্যতীত প্রতিটি নোডের মানের বিটওয়াইস অ্যান্ড গণনা করতে হবে। পূর্বে গণনা করা উত্তরগুলি পুনরায় গণনা এড়াতে আমাদের সংরক্ষণের প্রয়োজন হতে পারে। অন্যথায়, এটি TLE দিতে পারে।

Subtask 3:

মূল সাবটাস্কের জন্য, আমরা ডিএসইউ অন ট্রি নামে একটি ধারণা ব্যবহার করতে পারি।

প্রথমত প্রতিটি নোড v এর জন্য এই নোডের সাথে সম্পর্কিত সমস্ত প্রশ্নের সংরক্ষণ করি । তারপরে প্রতিটি নোডের জন্য, যা এর সাবট্রির রুট, একটি ডিএফএস চালাই । ডিএফএসের প্যারামিটার হল নোড v এবং map<int,int>mp। এই মাপে v এর সাবট্রির প্রত্যেক গভীরতা i এর জন্যে mp[i] = সকল নোডের bitwise অ্যান্ড ভালু যাদের গভীরতা i.

এই map সহজভাবে গণনা করা যেতে পারে। v এর সমস্ত চিলড্রেন বিবেচনা কর এবং তাদের জন্য এমন map গণনা কর । স্পষ্টতই, আমাদের মাপ mp এর চিলড্রেন এর মধ্যে যার আকার সর্বাধিক তার আকারের হবে। তারপরে প্রতিটি চিলড্রেনএর মাপ বিবেচনা করি এবং সেগুলিকে একত্রিত করি । অবশ্যই, আমরা একটি বৃহত্তর মাপের সাথে একটি ছোট মাপ একীভূত করব। এর পরে, তোমার কাছে বর্তমান নোডের সমস্ত প্রশ্নের উত্তর দেওয়ার জন্যে প্রস্তুত । উত্তর হবে -1 যদি v এর গভীরতা k এ কোন চিলড্রেন না থাকে।

নাম: **Himi's Negligence**
সেটার: Mahmud Sajjad Abeer (msabeer)
অল্টারনেট লেখক : Mirza Zamiur Rahman （Tashdid_trdb）
মূল ধারণা: Matrix Expo

সাবটাস্ক ১ঃ M = 1 থাকায় এই সমস্যাটি খুব সহজেই সমাধান করা সম্ভব কারণ এখানে প্রতি ঘড়েই ৪টি সুযোগ(a, b, c, #) ব্যবহার করা সম্ভব প্রথম সারির ঘড় বাদে। তাহলে উত্তর হবে 4^{N-1} যেটি O(logN) এ বিগমড দিয়ে করে ফেলা সম্ভব।

সাবটাস্ক ২ঃ N <= 5 দেয়া ছিল এখানে। এর সমাধান হচ্ছে ব্যাকট্র্যাক(রিকার্সিভলি সব রকম উপায় দেখার একটি মাধ্যম)। যদি ধীর হয়ে যায় তাহলে শর্ত না ভেঙ্গে ব্যাকট্র্যাক করলেই কাজ হয়ে যাবে। M <= 3 দেয়ায় এটি সময়সীমার মধ্যেই সব সঠিক উপায়গুলো দেখতে পারবে যদিও থিওরেটিকাল কমপ্লেক্সিটি আরো বেশি।

সাবটাস্ক ৩ঃ এখানে N <= 10000 ছিল, তারমানে এখন আর ব্যাকট্রাক কাজ করবেনা। এখন ধরো কিছু জিনিষ আমি আগে বের করে রাখলাম। যেমনঃ শর্ত মেনে তৈরী করা যেকোন সারি থেকে শর্ত মেনে পরের সারিতে কি কি উপায়ে যাওয়া যায় বের করে রাখা। শর্ত মেনে প্রতি সারি তৈরী করার সবচেয়ে বেশি উপায় দিবে M=4 এর

জন্য কিন্তু সেটাও খুব বেশি না, মাত্র ২০০ মত। এটা বের করে রাখলে খুব সহজেই একটি ডাইনামিক প্রোগ্রামিং সমাধান লিখে ফেলা সম্ভব।

সাবটাস্ক ৪ঃ সাবটাস্ক ৩ এর মতই শর্ত মেনে আমি যতরকম সারি বানানো যায় তাদেরকে যদি একটি করে গ্রাফ নোড ধরি, তাহলে কোন নোড থেকে কোন নোডে যাওয়া যায় তার একটি
২০০ x ২০০ Adjacency Matrix তৈরী করে ফেলা সম্ভব। এই ম্যাট্রিক্সের N-1তম সূচক থেকে খুব সহজেই উপায় সংখ্যা বের করে ফেলা সম্ভব। ম্যাট্রিক্সের Nতম সূচকও বিগমডের মত করে O(logN) এ বের করে ফেলা যায় কিন্তু এতে মাট্রিক্স গুণ করার কমপ্লেক্সিটিও আমলে নিতে হবে, তাই প্রতি টেস্টে কমপ্লেক্সিটি দাঁড়াচ্ছে O(logN * |Node Count|^3)।

নাম: **Placing the Repeater**

সেটার: Raian Rahman (pranonraian)

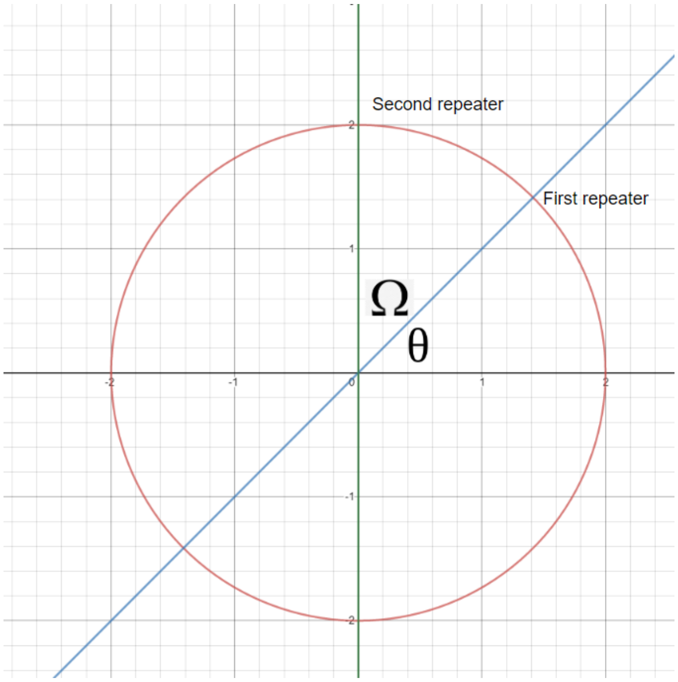
অল্টারনেট লেখক: Mirza Zamiur Rahman （Tashdid_trdb）, Hasibul Haque Himel （fire_tornado）, Asaduzzaman Herok （Arcturus）

মূল ধারণা: Basic Geometry

খুব সহজভাবে চিন্তা করলে এই সমস্যার জন্য একটি বৃত্তের কেন্দ্র এবং বৃত্তের পরিধির উপরের একটি বিন্দুর স্থানাঙ্ক দেওয়া আছে।

আমাদের কাজ হলো বৃত্তের পরিধির উপর আরো k−1 টি বিন্দুর স্থানাঙ্ক বের করতে হবে যাতে যেকোন বিন্দুর জন্য তার পাশ্ববর্তী বিন্দুর দুরত্ব সর্বোচ্চ হয় অর্থাৎ বিন্দুগুলোর দুরত্ব পাশ্ববর্তী বিন্দুর থেকে সমান দূরবর্তী হতে হবে।

এখন স্থানাংক দুটি থেকে আমরা বৃত্তের ব্যাসার্ধটি বের করতে পারি। (মনে করি ব্যাসার্ধ = ‘r’). শুধু তাই নয় আমরা কেন্দ্র ও প্রথম রিপিটারের সংযোগকারী সরলরেখার ঢালও বের করতে পারি।



ছবিতে প্রথম রিপিটার এবং দ্বিতীয় রিপিটারের অন্তর্ভুক্ত কৌনিক দুরত্ব omega এবং ব্যাসার্ধ হল r
ধরি প্রথম রিপিটারের স্থানাংক (x1,y1)। তাহলে আমরা দ্বিতীয় রিপিটারের স্থানাঙ্ক (x2,y2) নিম্নের সুত্র দিয়ে বের করতে পারি:

```
x2 = r *cos( theta + omega) +  x1
y1 = r *sin( theta + omega) +  y1
```

অতএব, আমাদের কাজ হল ঢাল theta এবং দুটি রিপিটারের কৌনিক দুরত্ব omega এবং বৃত্তের ব্যাসার্ধ r বের করতে হবে। তারপর খুব সহজেই উপরের সুত্রটি দিয়ে আমরা সমস্যাটি সমাধান করতে পারবো।.

নাম: **Fighting Over a Duster**

সেটার: Safayet Hossain Masum (curly_braces)

অল্টারনেট লেখক: Mirza Zamiur Rahman （Tashdid_trdb）

মূল ধারণা: Biparted Matching

Subtask 1: আমরা শিক্ষার্থীদের সকল সাব সেট এর জন্যে দেখব কোন সাব সেট এ সর্বোচ্চ সংখ্যক শিক্ষক খুশি।

Complexity $O(2^{(a+b)})$.

Subtask 2: আমরা শিক্ষকদের সকল সাবসেট এর জন্যে বের করবো সর্বোচ্চ সাবসেট, যেখানে কোনো শিক্ষক অপর কোনো শিক্ষকের বিরোধীতা করে না।

Complexity $O(2^n)$.

Subtask 3: একটু লক্ষ করলে দেখা যায় যে, কোনো শিক্ষক নিজের সেকশন এর অপর কোনো শিক্ষকের বিরোধীতা করে না। আমরা যেই সেকশন এ শিক্ষক কম ওই সেকশন এর শিক্ষকদের সবগুলো সাবসেট এর জন্যে দেখব অপর সেকশন এ কতজন শিক্ষক খুশি। এইভাবে সর্বোচ্চ সাবসেটটি বের করতে হবে।

Complexity $O(2^{(n/2)})$

Subtask 4: এই প্রবলেমটিকে একটি প্রচলিত "Edge cover" প্রবলেম এ রূপান্তরিত করা যায়। আমরা ধরে নেই যে প্রত্যেক শিক্ষক একটি নোড আর প্রত্যেক বিরোধিতা একটি Edge, তাহলে ওই গ্রাফ এর Edge cover ই হবে উত্তর।

Complexity $O(n^2\sqrt{n})$.

নাম: **Big LCM**

সেটার: Safayet Hossain Masum (curly_braces)

অল্টারনেট লেখক: Hasibul Haque Himel （fire_tornado）

মূল ধারণা: Pollard Rho

Subtask 1+2: আমরা লসাগু বের করবো এবং দেখব সর্বোচ্চ কোন n এর জন্যে BLCM কে c^n দিয়ে নিঃশেষে ভাগ করা যায়। এইটি প্রমাণ করা সম্ভব যে লসাগু 64 বিট এর বেশি হবে না।

Complexity $O(b * log(b))$

Subtask 3: এইখানে লসাগু 64 বিটে জায়গা নিবে না। তাই আমরা লসাগু কে prime factorization হিসেবে বের করতে পারি। সর্বশেষ c এর প্রতিটি মূলদ বিভাজক p এর জন্যে এর মধ্যে সর্বনিম্ন freqLCM(p)/freqC(p) ই হবে উত্তর। এখানে freqLCM(p) এর মানে হলো লসাগু তে p কতবার আছে, আর freqC(p) হলো c তে p কতবার আছে।

Complexity $O(b * log(b))$

Subtask 4: এখানে যেহেতু লসাগু বের করা সম্ভব নয়, আমরা প্রথমে c কে prime factorization করবো। এখন c এর সকল prime বিভাজক p এর জন্যে দেখব আমাদের উল্লেখিত Range এর কোন নাম্বার এ p সর্বোচ্চ বার রয়েছে তৎপর SubTask 3 এর মত করে সমাধান করা সম্ভব।

Complexity $O(\sqrt{c} + log(b))$

Subtask 5: এইখানে c এর prime factorization করতে Pollard এর rho algorithm ব্যবহার করতে হবে।

Complexity $O(c^{(1/4)} + log(b))$

নাম: **Sum of Medians**

সেটার: Safayet Hossain Masum (curly_braces)

অল্টারনেট লেখক: Hasibul Haque Himel (fire_tornado)

মূল ধারণা: Math, Combinatorics

লক্ষ করলে দেখা যাবে যে, আমাদের 0 এর ছোট মধ্যমা বিশিষ্ট সাব সেট নিয়ে কোনো লাভ নেই, তাই হতে পারে আমাদের k এর চেয়ে কম Subset নেয়া লাগতে পারে।

Subtask 1: এখানে সবগুলো উত্তর হাতে বের করা সম্ভব।

Complexity $O(1)$.

Subtask 2: আমরা সকল সাবসেট বের করে এর জন্যে উত্তর বের করে দেখতে পারি।

Complexity $O(2^n)$.

Subtask 3: আমরা যদি সংখ্যাগুলো ছোট থেকে বড় ক্রম অনুসারে সাজাই তাহলে একটা সংখ্যার ছোট যদি L টা সংখ্যা থাকে আর বড় R টা তবে, দুই দিক থেকে এমন কত ভাবে সংখ্যা নেয়া যায় যেনো ছোট যতোগুলো তার থেকে বড় সর্বোচ্চ একটি বেশি হয়।

Complexity $O(n^2)$.

Subtask 4: এইখানে আমরা সকল nCr যা 10^{18} এর ছোট তা আগে থেকেই বের করে রাখতে পারি। এক্ষেত্রে Pascal এর ত্রিভুজ ব্যবহার করা যেতে পারে।

Complexity $O(n\sqrt{n})$.

Bonus: এটি প্রমাণ করা সম্ভব যে সাজানোর পর r তম সংখ্যাটি nCr টি সাবসেট এ মধ্যমা হিসেবে রয়েছে।

নাম: **Interactive GCD**

সেটার: Safayet Hossain Masum (curly_braces)

অল্টারনেট লেখক: Asaduzzaman Herok (Arcturus)

মূল ধারণা: Interactive

যদি গসাগু 1 বিশিষ্ট সাবসেট এর সংখ্যা জোড় হয় তবেই খেলা অমীমাংসিত থাকা সম্ভব।

Subtask 1 + 2: এক্ষেত্রে পুরো অ্যারে টা বের করে ফেলা সম্ভব, তারপর সকল সংখ্যা সম্ভব ইনপুট এর জন্য আউটপুট হাতে বের করা যেতে পারে।

Subtask 3: এটি inclusion-exclusion principle এর সাহায্যে করা সম্ভব। আমরা বের করবো কতগুলো সাবসেট এর গসাগু 1 এর বেশি। তৎপর মোট সাবসেট থেকে প্রাপ্ত উত্তর বাদ দিলেই কাঙ্খিত উত্তর পেয়ে যাবো।

Subtask 4: Subtask 3 এর পর এটি লক্ষ করা যাবে যে সকল সংখ্যা নিয়ে প্রশ্ন করে লাভ নেই। সর্বোচ্চ 60 টি প্রশ্নের আগেই উত্তর সম্পর্কে নিশ্চিত হওয়া সম্ভব। Mobius function নিয়ে জানাশোনার চেষ্টা করার উপদেশ রইলো।

নাম: The Hidden Island

সেটার: ফাহিম শাহরিয়ার

Subtask 3:

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)

Subtask 1:

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| * (2^10))

Subtask 2:

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|)

নাম: ACT 1: Ash Counting Trees

সেটার: Tahsin Masrur

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 KKK-টা চাইল্ড নোডও অপশনে চলে আসে, কিন্তু সে নিজে আর অপশন হিসেবে থাকে না। তাই, অপশনের সংখ্যা K−1K-1K−1 পরিমাণ বেড়ে যায়।

তার মানে, তুমি প্রথম নোডটা 111 উপায়ে লেবেল করতে পারো।

দ্বিতীয় নোডটা 1+K−11+K-11+K−1 উপায়ে লেবেল করতে পারো।

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

অর্থাৎ, তুমি iii-তম নোডটা 1+(i−1)(K−1)1+(i-1)(K-1)1+(i−1)(K−1) উপায়ে লেবেল করতে পারো।

অতএব, F(N,K)=∏i=1N(1+(i−1)×(K−1))F(N,K) = \prod\limits_{i=1}^{N} (1 + (i-1)\times(K-1))F(N,K)=i=1∏N(1+(i−1)×(K−1))

সাবটাস্ক ১

যেহেতু N≤106N \leq 10^6N≤106, আমরা F(N,K)F(N,K)F(N,K) এর প্রত্যেক টার্ম বের করার জন্য জাস্ট একটা লুপ চালিয়ে দিতে পারি এবং তাকে যতবার সম্ভব PPP দ্বারা ভাগ করে তার মধ্যে থেকে

PPP এর সকল ফ্যাক্টর সরিয়ে আমরা F(N,K)PZ\frac{F(N,K)}{P^Z}PZF(N,K) বের করতে পারি (যেখানে ZZZ ম্যাক্সিমাম)।

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

টাইম কমপ্লেক্সিটি O(NlogPN)O(N\log_P{N})O(NlogPN)

সাবটাস্ক ২

এখানে N≤1012N \leq 10^{12}N≤1012, কিন্তু K≤2K \leq 2K≤2 মাত্র। তবে যখন K=1K=1K=1, তখন কিন্তু উত্তর সবসময়ই 111

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

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

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

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

অর্থাৎ, N!=1×2×0×1×2×0×1×2×0×1(modP)N! = 1 \times 2 \times 0 \times 1 \times 2 \times 0 \times 1 \times 2 \times 0 \times 1 \pmod PN!=1×2×0×1×2×0×1×2×0×1(modP)

তার মানে, জাস্ট সাবটাস্ক ১-এর মত করে লুপ চালিয়েই F(P−1,K)F(P-1, K)F(P−1,K) বের করে আমরা উত্তরের সাথে F(P−1,K)NPF(P-1,K)^{\frac{N}{P}}F(P−1,K)PN গুণ করে দিতে পারি

(যেহেতু F(P−1,K)F(P-1, K)F(P−1,K) NP\frac{N}{P}PN বার রিপিট হবে)। কিন্তু তাতে শেষের (N(modP))−1(N \pmod P) - 1(N(modP))−1 সংখ্যক টার্ম মিসিং থাকবে, সেগুলোও লুপ চালিয়ে সহজে উত্তরের সাথে গুণ করে ফেলা সম্ভব।

কিন্তু আরো কিছু টার্ম তো আসলে মিসিং আছে, যেমন 666। একে PPP দ্বারা ভাগ করলে তো 222 পাওয়া যায় যা আমাদের উত্তরের সাথে গুণ করা হয়নি। আসলে PPP এর সকল গুণিতকেরই একই অবস্থা। এই গুণিতকগুলো কারা আসলে?

PPP, 2P2P2P, 3P3P3P, 4P4P4P, 5P5P5P, 6P6P6P.....

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

111, 222, 333, 444, 555, 666.... যাদেরকে গুণ করলে আবার সেই ফ্যাক্টোরিয়ালই আসতেছে!

এরকম NP\frac{N}{P}PN সংখ্যক টার্ম থাকবে। অর্থাৎ, F(NP,K)F(\frac{N}{P},K)F(PN,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<PN<PN<P, যা লুপ চালিয়েই বের করা সম্ভব। রিকার্সনের লেভেল সংখ্যা হবে log⁡PN\log_P{N}logPN

টাইম কমপ্লেক্সিটি O(Plog⁡PN)O(P\log_P{N})O(PlogPN)

সাবটাস্ক ৩

এখন যেহেতু K≤106K \leq 10^6K≤106, F(N,K)F(N,K)F(N,K) ফ্যাক্টোরিয়াল নাও হতে পারে।

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

এক্ষেত্রে 1+(i−1)×(K−1)≡1(modP)1 + (i-1)\times(K-1) \equiv 1 \pmod{P}1+(i−1)×(K−1)≡1(modP), তাই উত্তর সব সময়ই 111 হবে। তার মানে (K−1)(K-1)(K−1) যদি PPP দ্বারা বিভাজ্য না হয়, সেটাই বেশি চিন্তার বিষয়।

ধরো K=6K=6K=6 এবং `$P=3$

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

P=3P=3P=3 দ্বারা mod করার পর?

F(N,5)=1×0×2×1×0×2×1×0×2×1×0×2×1×0.....(mod3)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}F(N,5)=1×0×2×1×0×2×1×0×2×1×0×2×1×0.....(mod3)

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

আমাদের iii-তম টার্মটা কী? 1+(i−1)×(K−1)1+(i-1)\times(K-1)1+(i−1)×(K−1)

এটাকে PPP দ্বারা mod করলে কী পাই?

1+(i−1)×(K−1)≡1+((i−1)(modP))×(K−1)(modP)1 + (i-1)\times(K-1) \equiv 1 + ((i-1)\pmod{P})\times(K-1) \pmod P1+(i−1)×(K−1)≡1+((i−1)(modP))×(K−1)(modP)

তাই, যেকোন i>Pi>Pi>P এর জন্য iii-তম টার্মটা কোন একটা jjj-তম টার্মের সমান হবে যেখানে j=i(modP)j=i \pmod Pj=i(modP)

এখন, F(N,5)(modP)F(N,5) \pmod PF(N,5)(modP) 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 PPP দ্বারা বিভাজ্য?

আমাদের উদাহরণে এই টার্মগুলো হচ্ছে 666, 212121, 363636, 515151, 666666...

কিন্তু আগেরবারের সাথে এবার একটা পার্থক্য আছে। এবার আমরা যদি এই টার্মগুলোকে PPP দ্বারা ভাগ করি তাহলে প্রথম টার্মটা 111 হয় না। বরং টার্মগুলো এমন হয়ঃ 222, 777, 121212, 171717, 222222...

এবারও টার্মগুলোর মধ্যে ডিফারেন্স একই থাকতেছে, তা হলো K−1=5K-1=5K−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 PPP দ্বারা বিভাজ্য তা বের করতে পারি। তারপর তা PPP 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 কমপ্লেক্সিটি O(Plog PN)O(P\log_P{N})O(PlogPN)

সাবটাস্ক ৪

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

প্রথমত, (K−1)(K-1)(K−1) PPP দ্বারা বিভাজ্য হলেই উত্তর আর 111 না। তবে এটা নিশ্চিত যে এক্ষেত্রেও আগের মত প্রত্যেক PQP^QPQ টার্ম পর পর একটা রিপিটেশন থাকবে যা আগের মত করে প্রমাণ করা সম্ভব। তাছাড়া (K−1)(K-1)(K−1) PPP দ্বারা বিভাজ্য হলে এটাও নিশ্চিত যে F(N,K)F(N,K)F(N,K) এর কোন টার্মই PPP 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)(K-1)(K−1) PPP দ্বারা বিভাজ্য না, তখন কিছু টার্ম PPP দ্বারা বিভাজ্য হতে পারে এবং সেগুলোকে আগের মত করেই সামলাতে হবে। তবে এবার একটা জিনিস খেয়াল করতে হবে যে রিপিটিশন আর PPP টার্ম পর পর হচ্ছে না, বরং PQP^QPQ টার্ম পর পর হচ্ছে। তাই আমরা এই পুরো রিপিটেড পার্টটা এমনভাবে নিব যেন তার মধ্যে কোন PPP দ্বারা বিভাজ্য টার্ম না থাকে এবং আগের মতই তার এক্সপোনেনশিয়েন ও বাদবাকি টার্মগুলো উত্তরের সাথে গুণ করব।

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

টাইম কমপ্লেক্সিটি O(PQlog⁡PN)O(P^Q\log_P{N})O(PQlogPN)

নাম:

সাবটাস্ক ১ঃ

সবসময় Q>=TQ >= TQ>=T এখানে। তাই প্রথমে TTT 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 [t1,t2][t_1, t_2][t1,t2] রেঞ্জ আছে বলে ধরা যায়, যেখানে ফ্রী টাইম t1t_1t1-তম ঘণ্টা থেকে শুরু হয়ে t2t_2t2-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][l, r][l,r] রেঞ্জের সম্পূর্ণ সময়ে ক্লাস না থাকে, তাহলে এই রেঞ্জের ফ্রী টাইম হবে =(r−l+1)= (r-l+1)=(r−l+1) ঘণ্টা।

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

এখন আমরা আমরা যদি কোনো সময় tit_iti-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।

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

এভাবে আমরা খুব সহজেই বাইনারী সার্চের মাধ্যমে 1st1^{st}1st ঘণ্টা থেকে TTT 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 ≤2×N×log2(T) \leq 2\times N\times log_2(T)≤2×N×log2(T), যা 5×1045\times 10^45×104 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][1, t][1,t] এই ফরম্যাটে করব অর্থাৎ, 1st1^{st}1st ঘণ্টা থেকে ttt-তম ঘণ্টা পর্যন্ত টোটাল ফ্রী টাইম কত - সেটা জিজ্ঞাসা করব । এর কারণ একটু পরেই আমারা বুঝতে পারব ।

প্রোপার্টি ৩ঃ [1,t1−1][1, t_1-1][1,t1−1] রেঞ্জে টোটাল ফ্রী টাইম T1T_1T1 ঘণ্টা এবং [1,t2][1, t_2][1,t2] রেঞ্জে টোটাল ফ্রী টাইম T2T_2T2 ঘণ্টা হলে [t1,t2][t_1, t_2][t1,t2] রেঞ্জে টোটাল ফ্রী টাইম (T2−T1)(T_2-T_1)(T2−T1) ঘণ্টা, যেখানে 1<t1≤t2≤T1 < t_1 \leq t_2 \leq T1<t1≤t2≤T ।

ধরা যাক, 1st1^{st}1st ঘণ্টা থেকে ক্লাস টাইম, ফ্রী টাইম বের করতে করতে আমরা এখন tit_iti-তম ঘণ্টায় আছি । তার মানে [1,ti−1][1, t_i-1][1,ti−1]পর্যন্ত টোটাল ফ্রী টাইম কত, সেটাও আমাদের জানাই আছে ।

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

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

tit_iti-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 1st1_{st}1st ঘণ্টা থেকে প্রথম ক্লাস কততম ঘণ্টায় শুরু হয়, সেটা বের করতে log⁡2(T)\log_2(T)log2(T) সংখ্যক প্রশ্ন লাগবে। এরপর উপরের নিয়মে রেঞ্জ ছোট করার পর বাইনারী সার্চ চালাতে হবে এমন সবচেয়ে বড় রেঞ্জের দৈর্ঘ্য হবে T2\frac{T}{2}2T । এমন দৈর্ঘ্যের রেঞ্জ ম্যাক্সিমাম 222টা হওয়া সম্ভব (কারণ 2×T2=T2 \times \frac{T}{2} = T2×2T=T) । তাহলে, আরও 2×log⁡2(T2)2\times \log_2(\frac{T}{2})2×log2(2T) সংখ্যক প্রশ্ন লাগবে। এভাবে এগোতে থাকলে দেখা যায়, টোটাল প্রশ্নের সংখ্যা = log⁡2(T)+2×log⁡2(T2)+4×log⁡2(T4)+.....+k×log⁡2(Tk)\log_2(T)+2\times \log_2(\frac{T}{2})+4\times \log_2(\frac{T}{4})+.....+k\times \log_2(\frac{T}{k})log2(T)+2×log2(2T)+4×log2(4T)+.....+k×log2(kT), যেখানে k=ceil(log⁡2(N))k = ceil(\log_2(N))k=ceil(log2(N)) ।

এই প্রয়োজনীয় প্রশ্নের সংখ্যা 4.3×1044.3\times 10^44.3×104 এর চেয়ে কম। আরও একুরেট ক্যালকুলেশন করলে দেখা যায়, প্রশ্নসংখ্যা কখনোই 418434184341843 এর বেশি দরকার হবে না, বরং কেয়ারফুল থাকলে আরও কম লাগবে।