সমস্যা ও সমাধান

বই - ১

মোঃ মাহবুবুল হাসান

সেপ্টেম্বর, ২০১৭

সূচীপত্র

নকশা তালিকা

সারণী তালিকা

অধ্যায় ১

একটুস খানি কথা

প্রোগ্রামিং কন্টেস্ট এর ইতিহাস কিন্তু অনেক পুরোনো। তোমরা কি কেউ ACM ICPC World Finals এর prize giving ceremony দেখেছ? এখন তো আবার ইন্টারনেটের কল্যানে সেই অনুষ্ঠান দেখতে সশরীরে হাজির হতে হয় না। তো যা বলছিলাম, সেই অনুষ্ঠানে প্রতিবার বিল পাউচার প্রতি বছরের বিজয়ীদের নাম বলে আর সেই লিস্ট শুরু হয় 1977 সাল থেকে। তার মানে আজ থেকে প্রায় 40 বছরেরও আগে প্রোগ্রামিং কন্টেস্ট শুরু হয়েছে। আজ 2017 সালে এসে প্রোগ্রামিং কন্টেস্ট এর বিভিন্ন platform দাড়িয়ে গেছে। ACM ICPC World Finals, IOI, Codejam, Hackercup, Codefestival, Codechef Snackdown এবং আরও অনেক onsite event হয়। এছাড়াও online এ প্রোগ্রামিং প্রতিযোগিতা করার এবং প্র্যাক্টিস করার বহু সাইট আছে। এছাড়াও আছে প্রোগ্রামিং প্রতিযোগিতা নিয়ে আলোচনা করার অনেক ফোরাম। গত 4 বছরেরও কম সময়ে প্রোগ্রামিং প্রতিযোগিতার যে প্রসার হয়েছে তা আসলে কথায় প্রকাশ করা যাবে না। কিন্তু কষ্টের বিষয় হল আমাদের দেশে সে তুলনায় তেমন প্রসার নেই। আমাদের দেশে গত দশ পনেরো বছর ধরে গড়ে প্রতি বছর চার পাঁচটি করে কন্টেস্ট হয়। কিছু দিন আগে থেকে (2011 থেকে শুরু) প্রতি বছর প্রোগ্রামিং ক্যাম্প করার চেস্টা করা হয়। এছাড়াও স্কুল কলেজ লেভেলে NHSPC আয়জন করা হচ্ছে অল্প কিছু দিন আগে থেকে। এতেই সীমাবদ্ধ আমাদের প্রোগ্রামিং কন্টেস্ট। সেই তুলনায় রাশিয়া, চীন, পোল্যান্ড যারা কিনা প্রোগ্রামিং কন্টেস্টে সবসময় এগিয়ে থাকে তাদের আয়োজন শুনলে বা দেখলে আমার বেশ খারাপ বা আফসস লাগে। এজন্য না যে তারা অনেক কিছু করে। বরং তারা আমাদের থেকে অনেক কম onsite কন্টেস্ট করে। তারা যা করে তাহল quality পূর্ণ ক্যাম্প। সেখানে তাদের বুড়ো কন্টেস্টেন্টরা এসে quality পূর্ণ ক্লাস নেয় এবং quality পূর্ণ প্রবলেম নিয়ে আলোচনা করে। ফলে দেখা যায় আমরা আমাদের দেশের কন্টেস্টে 10 টি সমস্যার ভেতরে 7-8 টি করে সমাধান করতে পারলেও তাদের কন্টেস্টে যে 10 টি সমস্যা থাকে তার 2 বা 3 টি টেনেটুনে আমরা সমাধান করতে পারি আর তারা ঠিকই 8-9 টি করে সমাধান করে বসে থাকে। বলার অপেক্ষা রাখে না আমাদের আর তাদের ভেতরে আকাশ আর পাতাল তফাৎ।

আমার কাছে মনে হয়েছে সমস্যা সমাধানের দুটি হাতিয়ার আছে। এক জ্ঞান, দুই চেষ্টা। তুমি যদি না জান bfs কি বা dp কি তাহলে তো তা সম্পর্কিত সমস্যা সমাধান করা কঠিন। হয়তো bfs বা dp আসলে বেশ সহজ জিনিস, কেউ বই না পড়েই নিজে নিজে শিখতে পারবে বা নিজে থেকেই সেসব সমাধান বের করতে পারবে। কিন্তু fft বা kmp বা convex hull ইত্যাদি সম্পর্কিত algorithm বা theory এসব যদি তুমি না পড় না শিখ না জান তাহলে সেসব সম্পর্কিত সমস্যা সমাধান করা খুবই কঠিন হবে। সেজন্য প্রথমত জ্ঞান দরকার। আর দ্বিতীয় জিনিস হল চেষ্টা। তোমাকে সমস্যায় বলা থাকবে না যে এই সমস্যা bfs দিয়ে সমাধান হবে। তোমাকে চিন্তা করে, নানা modify করে বুঝতে হবে যে এটা এভাবে করে bfs করলে সমাধান হবে।

কিছু জ্ঞান যেমন bfs, dfs, mst, kmp, dp, gcd ইত্যাদি খুবই সহজ জিনিস। যারা এসব এখনও পার না তারা হয়তো আমাকে মনে মনে গালি দিচ্ছ যে কত কঠিন জিনিস, আর আমি কিনা এদের খুবই সহজ বলছি। কিন্তু সত্যি বলতে প্রোগ্রামিং কন্টেস্টে আরও অনেক কঠিন কঠিন জিনিস আছে। সেসবের

তুলনায় এসব খুবই সহজ। আমার মূল লক্ষ্য ছিল সেসব কঠিন জিনিস নিয়ে বই লিখার। যেমন- fft, nft, persistent data structure, link cut tree ইত্যাদি। কারণ আমার মনে হয়েছে সহজ বিষয়গুলি তো এক জন চাইলেই নেট থেকে পড়ে শিখতে পারবে কিন্তু এই কঠিন বিষয়গুলি ইংরেজিতেই ভাল করে নাই। ভাল করে কোথাও থাকলেও এক জায়গায় সব নেই। তাই বাংলায় লিখলে হয়তো যারা বাংলাদেশের advanced contestant তাদের জন্য ভাল হত। কিন্তু সেই বই লিখতে গিয়ে মনে হয়েছে যে আসলে সহজ বিষয়ের উপরেও বই দরকার। এজন্য না যে আমাদের দেশে advanced contestant এর সংখ্যা খুবই কম, বরং এজন্য যে আমি যখন advanced বিষয়ের উপর লিখব তখন যেন এটা মনে না হয়, আরে এই বিষয় কি ওরা জানে? অমুক টেকনিক কি ওরা জানে? এসব ভেবেই আমার "প্রোগ্রামিং কন্টেস্ট- ডেটা স্ট্রাকচার ও অ্যালগরিদম" বই লিখা। বইটা লিখা শুরু হয় 2011 সালে আর বেশির ভাগ জিনিস লিখা শেষ হয় 2013 সালে। এরপর নানা রকম আলসেমি পার করে 2015 তে তা pdf আকারে ছাড়া হয় এবং 2016 তে হার্ডকপি হিসাবে। কঠিন বিষয়ের উপর বই লিখার ইচ্ছা নিয়ে শুরু করে 5 বছর কাটিয়ে দিয়েছি সহজ বিষয়ের উপর লিখে। কিন্তু সেই বই এ শুধু theory ছিল। সেখানে তেমন কোনো সমস্যা নিয়ে আলোচনা ছিল না, প্রব্লেমের লিস্ট ও ছিল না। সত্যি বলতে, ঐ বইএ আরও একটি চ্যাপ্টার ছিল যেখানে প্রায় 300-400 সমস্যা নিয়ে আলোচনা করা ছিল (যত দূর সম্ভব 2013 সালের সব icpc regional এর problem এবং solution নিয়ে ছিল, এমনকি neerc, ceerc এবং chinese regional গুলিও ছিল)। কিন্তু বই এর আকার এতো বিশাল হচ্ছিল আর সেই সমস্যার আলোচনা এতো সংক্ষেপে ছিল যে আমি পরে ঐ অংশ মুছে ফেলি (এমনিতেই theory আলোচনা অংশ সংক্ষিপ্তও বলে যেই পরিমান গালাগালি হজম করতে হয়েছে সে তুলনায় ঐ চ্যাপ্টার মুছে দিয়ে ভালই হয়েছে বৈ কি!)। প্রথম বইএ প্রথম হাতিয়ার (জ্ঞান) নিয়ে কথা হলেও দ্বিতীয় হাতিয়ারের (চেস্টা) কোনো কিছুই ছিল না। সেই জন্যই আমার এই দ্বিতীয় বই। আমি নিজে প্রবলেম ক্যাটাগরাইজেসনে ওত অভ্যস্ত না। তাই এই বইয়ে যত সমস্যা আলোচনা করা হয়েছে তার লিস্টগুলি আমি বিভিন্ন সোর্স হতে নিয়েছি। মূল লিস্ট হল রুজিয়া লিউ এর এক নম্বর বই (সহজ বই) এর লিস্ট। এছাড়াও আমি timus এর beginner প্রব্লেমের একটি লিস্ট ব্যবহার করেছি। LightOJ এর টপিক এর লিস্ট হতেও বেশ কিছু সমস্যা আমি ব্যবহার করেছি। জ্যামিতির জন্য বাংলাদেশের জ্যামিতির গুরু নাফি এর একটা প্রবলেম লিস্ট ছিল আমাদের বুয়েটের গ্রুপে সেই লিস্ট ব্যবহার করেছি। Greedy সমস্যার জন্য codeforces এর কিছু সমস্যা ব্যবহার করেছি। এরকম নানা জায়গা হতে প্রবলেম নিয়ে সেসব নিয়ে আলোচনা করা হয়েছে।

এই বইয়ের মূল উদ্দেশ্য যদি তুমি মনে কর বিভিন্ন টপিকে প্র্যাক্টিস করা তাহলে ভুল ভাববে। মূল উদ্দেশ্য হল, চীনের বাচ্চা কন্টেস্টেন্টরা রুজিয়া লিউ এর যেই বই পড়ে তাতে আলোচিত প্রবলেমগুলো তোমাদের সামনে তুলে ধরা। আর বুঝানো কেন তারা এতো ভাল পারে আর কেন আমরা পারি না। তাদের সহজ বইয়েই যদি এতো কঠিন প্রবলেম থাকে তাহলে কঠিন বইয়ে না কি আছে! যদি তোমরা সত্যিকারের ভাল কন্টেস্টেন্ট হতে চাও, তাহলে এই বইয়ের প্রবলেম গুলিকে তোমাদের কাছে সোজা লাগতে হবে। আর যারা শখের বশে কন্টেস্ট করতে চায়, খাটা খাটনি করতে চায় না, তারা এই বইকে নিরাপদ দুরত্বে দূরে সরিয়ে রাখো।

সাধারনত বইয়ে প্রতিটি অধ্যায়ে প্রবলেম গুলি তাদের difficulty অনুযায়ী আছে। সুতরাং তুমি একটি করে প্রবলেম দেখ আর নিজেরা তা সমাধান করার চেষ্টা কর। যদি নিজেরা পার তাহলে তো ভাল কথা, আর না পারলে সমাধান একটু একটু করে পড়ে দেখ। এক লাইন দুই লাইন করে পড় আর নিজেরা আবার চিন্তা কর। যদি নিজে সমাধান করতে পার তাহলেও এই বইয়ের সমাধান পড়। কারণ দেখা যাবে, যদিও এই সমস্যা $O(n^3)$ এ সমাধান হয় কিন্তু এর $O(n^2)$ সমাধান হয়তো আমি বর্ণনা করেছি। বা অনেক সময় আমি নানা ছোট খাট ট্রিকের কথা বলেছি তা হয়তো অন্য প্রব্লেমে কাজে আসবে।

অনেক সময় এটাও হতে পারে যে তুমি কোনো একটি সমাধান বুঝছ না। এরকম না হওয়াটাই অস্বাভাবিক (সংক্ষিপ্তও সমাধানের জন্যই হোক বা কঠিন সমস্যার জন্যেই হোক)। এখানে অনেক সমস্যা আছে যেগুলি আমার নিজের সমাধান করতে বা অন্যের সমাধান পড়ে বুঝতে 4-5 দিন করে লেগেছে। সুতরাং তুমি যদি 1 দিন ব্যয় করেই হাল ছেড়ে দাও তাহলে হবে না। কাগজে কলমে বিভিন্ন কেস নিয়ে চিন্তা করতে হবে আমি কি বলছি, বা তা কীভাবে হচ্ছে বা হতে পারে।

অনেকেই বলে বই আরও সহজ করে লিখতে, কিন্তু সত্যি বলতে এর থেকেও সহজ করে লিখার আমার সময় নেই। আমি যদি আরও সহজ করে আরও অনেক সময় নিয়ে লিখি তাহলে আমার আসল

উদ্দেশ্য (কঠিন বিষয় গুলি শেখা এবং তা নিয়ে লেখা) আমার কখনই পুরন হবে না। বা যারা বর্তমানে advanced coder আছে তারা কখনও কঠিন জিনিসগুলি শিখতে পারবে না। সহজ বিষয়গুলির উপর লিখা বা শেখানোর অনেকেই আছে কিন্তু কঠিন জিনিসগুলি নিয়ে সংক্ষিপ্ত লিখাও নেই। সুতরাং আমি যদি কখনও আমার লক্ষ্য পুরন করতে পারি তাহলে হয়তো কোনোদিন আবার এই "কঠিন" করে লিখা বই রিভাইজ করে সহজ করে আরও ছবি দিয়ে লিখব। কিন্তু আশা করি তত দিনে আমাদের দেশে এতো 2400+ রেটিং এর কোডার থাকবে যে "এই কঠিন বই বুঝছিনা" এ কথা বলা মানুষের পরিমান কমে যাবে বা তাদের মধ্য হতে আর কেউ সহজ করে বই লিখবে।

আগেই বলেছি এই বইয়ে UVa, Timus, Codeforces, LightOJ থেকে সমস্যা আছে। আমি প্রব্লেমের নাম্বারের পাশাপাশি প্রব্লেমের নাম দিয়েছি যাতে তোমাদের খুঁজে পেতে সুবিধা হয়। বিশেষ করে codeforces এর ক্ষেত্রে এই সমস্যা বেশি দেখা যায় কারণ সেখানে একটি প্রব্লেমের নাম্বার দেওয়া বেশ কঠিন। আর মাঝে মাঝে আমি প্রবলেম statement একটু পরিবর্তন করে নিয়েছি। যেমন দেখা যাবে মূল প্রব্লেমে বলা হয়েছে যে ছাগলের কাহিনী আমি লিখেছি গাধার কাহিনী। অনেক সময় মূল প্রব্লেমে বলেছে যে অমুক অমুক ডিজিট ব্যাতিত অন্য সকল ডিজিট ব্যবহার করা যাবে। কিন্তু আমি প্রব্লেমে লিখেছি যে অমুক অমুক ডিজিট কেবল ব্যবহার করা যাবে। এসবের মধ্যে কিন্তু তেমন কোনো তফাৎ নেই। সুতরাং আশা করি তোমাদের বুঝতে সমস্যা হবে না।

আর আমরা এই বইয়ে আলোচিত সমস্যাগুলির কোড রাখার জন্য একটি git repository খুলেছি যার লিংক (https://bitbucket.org/shanto86/problem-book-1-solutions). তোমরা চাইলে তোমাদের ac কোড এখানে পাঠাতে পার যাতে অন্যরা তা দেখে শিখতে পারে। আমরা চেষ্টা করব কোনো একটি সমস্যার একটি মাত্র সমাধান এখানে রাখার। তবে যদি কেউ সম্পূর্ণ ভিন্ন ভাবে কোনো সমস্যা সমাধান করে তাহলে আমরা সেই নতুন সমাধানও রাখতে পারি। সেই সাথে ধন্যবাদ Nayeem Jahan Rafi, Evan Hossain, Rezwan Arefin, Md Sahedul Islam Sohel, Mohtasim Bellah, Moudud Khan Shahriar, Abdullah Al Raqibul Islam, Imran Ziad এবং Sumit Saha কে যারা বইয়ের টেকনিকাল রিভিউ করতে সাহায্য করেছে।

অধ্যায় ২

সহজ কিছু সমস্যা

এই অধ্যায়ে আমরা সহজ কিছু সমস্যা নিয়ে আলোচনা করব। আশা করি এই অধ্যায়ের সমস্যাগুলি সমাধান করতে কোনো algorithm লাগবে না। তোমাদের online judge (oj) আর c/c++ এর সাথে পরিচিত করাই এই অধ্যায়ের উদ্দেশ্য। তোমাদের যারা ইতোমধ্যেই online judge এ অন্তত ১০০ টি সমস্যা সমাধান করে ফেলেছ তাদের জন্য হয়তো এই অধ্যায় ওত কাজে আসবে না। তোমরা বরং ঘড়ি ধরে এই অধ্যায়ের সমস্যাগুলি সমাধান করার চেষ্টা করে দেখতে পার। কত কম সময়ে কত কম পেনাল্টিতে সব সমাধান করা যায় সে চেষ্টা কর। আর সব সমাধান শেষে একবার আমার বলা সমাধানগুলিতে চোখ বুলিয়ে নিতে পার। হয়তো কিছু নতুন টিপস পেয়ে যেতে পারো!

অনুশীলনী বেশ অনেকগুলি সমস্যা দেওয়া আছে। সবগুলিই মুলত c/c++ জানা থাকলেই হবে। সুতরাং তোমরা যারা প্রোগ্রামিং কন্টেস্ট জগতে নতুন তাদের জন্যই মুলত এই অধ্যায়। আরেকটা জিনিস, কিছু কিছু সমস্যা আছে যেগুলো হয়তো মনে হবে কোড করা পেইন। কিন্তু আমার মতে তাও সেসব সমস্যা করা উচিত। এরকম সমস্যা তোমাদের debug করার ক্ষমতা বৃদ্ধি করে এবং সুন্দর করে কোড করা শিখতে সাহায্য করে। সুতরাং কোনো সমস্যা দেখতে পেইনফুল হলে তাকে পাশ কাটিয়ে যেও না। আর কোনো একটি সমস্যা সমাধান করা হলে তোমরা চাইলে নেটে সার্চ করে অন্যদের কোড দেখতে পার। এতে করে নতুন টেকনিক শিখতে পারবে।

Timus 1000 A+B Problem

সমস্যা: তোমাকে a এবং b দুটি সংখ্যা দেওয়া আছে। তাদের যোগফল প্রিন্ট করতে হবে।

সমাধান: খুবই সহজ সমস্যা। তবে সরাসরি কোড করতে যাবার আগে দেখে নাও a আর b এর লিমিট কত। দুঃখের বিষয় হলো এই সমস্যায় এদের কোনো লিমিট দেওয়া নেই। আগের যুগে এরকম সমস্যা প্রায়ই হতো। অর্থাৎ দেখা যায় প্রব্লেমে ইনপুট ভ্যারিয়েবলের লিমিট গুলি দেয়া নেই। যদি এটা কন্টেস্টে হয় তাহলে judge কে clarification এর মাধ্যমে জিজ্ঞাসা করে নাও। আর যদি এটা offline হয় অর্থাৎ প্র্যাকটিসের জন্য online judge এ সমাধান করার সময় এরকম হয়ে থাকে তাহলে গ্রুপে বা ফোরামে জিজ্ঞাসা কর অথবা ফোরামের পুরনো পোস্ট একটু ঘেঁটে দেখো। এরপরও যদি না বুঝতে পারো কী কাহিনী তাহলে আমার মতে সেই সমস্যা না চেষ্টা করাই ভাল। OJ গুলিতে অনেক ভাল ভাল সমস্যা থাকে, শুধু শুধু একটা ambiguous সমস্যার পেছনে সময় ব্যয় করার কোনো মানে নেই।

যারা কেবল মাত্র C শিখছ তারা অনেকেই ইনপুট নেবার আগে prompt প্রিন্ট করে থাক। Prompt হলো "please enter your number", "please input a" ইত্যাদি। বলতে দ্বিধা নেই আমি যখন আমার জীবনের প্রথম সমস্যা সমাধান করেছিলাম (UVa 100) তখন এই কাজ করেছিলাম। এসব prompt দেবার আসলে কোনো দরকার নেই। বরং এসব prompt দিলে তুমি wrong answer (বা কখনও কখনও output limit exceed বা presentation error) পাবা। কেন? কারণটা বোঝার

জন্য তোমাকে আগে চিন্তা করতে হবে online judge কীভাবে কাজ করে। Online judge তোমার কোড নেয়, স্বাভাবিক ভাবেই তোমার কোড compile করে এবং সব শেষে তার কাছে থাকা এক বা একাধিক input ফাইল তোমার কোডকে দেয়। তোমার কোড সেই ইনপুটগুলি standard input হতে পড়ে (অর্থাৎ সাধারণ scanf, cin ইত্যাদি দিয়ে)। কোনো কোনো oj তে অবশ্য আমাদের ফাইল হতে পড়তে হয় (fscanf, fopen, freopen ইত্যাদি ব্যবহার করতে হয় সেক্ষেত্রে)। কোন জাজে কীভাবে ইনপুট আউটপুট করতে হবে তা সাধারনত তাদের frequently asked question বা এই জাতীয় সেকশনে লিখা থাকে। যাই হোক, কম্পাইল করা শেষে তোমার কোড সেই সব ইনপুট পড়ে output দেয়। এবার তোমার দেয়া এই output তারা তাদের কাছে থাকা সঠিক output এর সাথে মেলায়। এখন তুমি যদি অতিরিক্ত prompt দাও সেটা তো আর জাজদের output এ থাকবে না তাই না? আর তাছাড়া তুমি ভেবে দেখো যদি তুমি ইনপুট নেবার জন্য prompt দাও তাহলে তোমার আসল উত্তর আর prompt আলাদা করার কি কোনো উপায় আছে? নেই। কারণ জাজ তো সব আউটপুট নিয়ে এর পর চেক করে। সে তো জানে না কোনটা prompt আর কোনটা আউটপুট। সেজন্য prompt দেবার কোনো দরকার নেই। আর হ্যাঁ, আমি জানি যে আমি যত সহজে oj কাজ করে বললাম, আসলে তো অতো সহজ না, কিন্তু মোটামোটি ঐরকম ভাবেই কাজ করে।

অনেক কথা হলো, আমাদের এই সমস্যায় ফিরে আসা যাক। তুমি এই সমস্যার ফোরামে একটু ঘাঁটলেই বুঝবে যে এখানে a আর b কে int এ নিলেই চলবে। তাহলে আর দেরী কেন, ঝটপট কোড করে ফেল। ৬-৭ লাইনের বেশি কোড হওয়া উচিত না!

Timus 1001 Reverse Root

সমস্যা: সর্বোচ্চ 10^{18} হতে পারে এমন কিছু অঋণাত্মক পূর্ণ সংখ্যা দেয়া আছে। তোমাকে তাদের square root গুলি (সর্বোচ্চ 4 decimal digit পর্যন্ত) উলটো অর্ডারে প্রিন্ট করতে হবে। অর্থাৎ প্রথম সংখ্যার square root সবশেষে, দ্বিতীয় সংখ্যার square root শেষ হতে দ্বিতীয়তে এরকম করে ইনপুটের সব শেষের সংখ্যার square root আউটপুটে সবার প্রথমে থাকবে। ইনপুট এর সাইজ সর্বোচ্চ 256KB. উদাহরণস্বরূপ- তোমার ইনপুটের সংখ্যাগুলি যদি হয় 1, 4 তাহলে আউটপুট হবে 2, 1.

সমাধান: এটিও বেশ সহজ। তবে একটা সমস্যা হলো কতগুলি নাম্বার দেয়া থাকবে তা বলা নেই। শুধু বলা আছে ইনপুটের সাইজ 256KB. আমরা জানি এক character এ 1 byte[১]. তাহলে ইনপুটে থাকবে 256K টা ডিজিট। আমরা আমাদের হিসাবের সুবিধার জন্য ধরে নিতেই পারি যে একেকটা ডিজিট একেকটি সংখ্যা। যদিও আসলে তা না, এদের মাঝে অনেক space থাকবে। কিন্তু আমাদের তো exact সংখ্যা দরকার নেই। মোটামোটি একটা estimate হলেই হবে। তাহলে 256K সাইজের একটি অ্যারে নিলেই নিঃসন্দেহে যথেষ্ট হবে [২]। এই অ্যারেতে তুমি ইনপুট নিয়ে রাখো। ইনপুট শেষে তুমি এই অ্যারের পেছন থেকে এসে একে একে সবার square root প্রিন্ট কর। আর আশা করি ইতোমধ্যেই খেয়াল করেছ যে আমাদের int ব্যবহার করলে চলবে না, long long ব্যবহার করতে হবে বা চাইলে double এও ইনপুট নিতে পারো। কারণ আমাদের ইনপুটের সংখ্যার লিমিট 10^{18}.

আরও এগুনোর আগে সাইজের ব্যাপারটা আরেকটু পরিষ্কার করে নেওয়া যাক। এইযে এখানে 256K সাইজের একটি long long এর অ্যারে নিলে, তার মানে এই অ্যারেটি memory তে কত জায়গা দখল করে? তার আগে জানা দরকার একটি long long মেমরিতে কত জায়গা দখল করে। আশা করি তোমাদের মনে আছে int নেয় 32 বিট বা 4 বাইট আর long long নেয় 64 বিট বা 8 বাইট (1 বাইট = 8 বিট)। এর মানে আমাদের মোট $8 \times 256 = 2^3 \times 2^8 = 2^{11}$ কিলোবাইট বা 2 মেগাবাইট মেমরির প্রয়োজন হবে। float বা double কত সাইজ দখল করে? এটা মনে রাখা সহজ। float নেয় int এর সমান আর double নেয় long long এর সমান। যদিও compiler বা os ভেদে কোনো একটি ডেটা টাইপ একেক compiler বা os এ ভিন্ন ভিন্ন সাইজ দখল করতে পারে। কিন্তু সেটা নিয়ে আসলে চিন্তা না করলেও চলবে। তাহলে এই হিসাব থেকে তোমরা জানলে যে তোমাদের প্রায় 2 মেগাবাইট এর মত জায়গা লাগবে। এটি কেন গুরুত্বপূর্ণ? কারণ প্রতিটি সমস্যায়

[১] আগে না জানলে এখন জানলে। যেকোনো একটি character ইনপুটে বা text ফাইলে 1 বাইট জায়গা দখল করে।
[২] এখানে K(Kilo) এর মানে হলো 2^{10}, 1000 না। একই ভাবে M(Mega) হলো 2^{20} আর G(Giga) হলো 2^{30}.

time limit এর পাশাপাশি memory limit ও দেয়া থাকে। যেমন এই সমস্যার memory limit হলো 64 মেগাবাইট। খেয়াল করবে তোমার কোড যেন এর বেশি মেমোরি ব্যবহার না করে। না হলে memory limit exceed খেয়ে যেতে পারো! বেশির ভাগ সময়ই এই লিমিট লক্ষ্য করার দরকার হয় না। কিন্তু মাঝে মাঝে দেখা যায় তুমি DP করতে চাচ্ছ কিন্তু DP এর অ্যারে অনেক বেশি মেমরি দখল করে বসে আছে। সেসব সময় তোমাকে মেমরি অপ্টিমাইজ করতে হতে পারে।

এই সমস্যার ক্ষেত্রে তোমরা চাইলে অ্যারের সাইজ কত নিতে হবে সেই হিসাবের ঝামেলায় না গিয়ে stl এর vector ব্যবহার করতে পারো। একটি করে ইনপুট নাও আর vector এ push back কর। সবশেষে vector এর পেছন থেকে আসো। iterator ব্যবহার করার কোনই দরকার নেই (iterator কি যদি না জান তাহলে আপাতত জানার দরকার নেই, দরকারের সময় শিখে নিলেই হবে)। আমি মনে হয়না জীবনে vector এর জন্য iterator ব্যবহার করেছি। তোমাদের সুবিধার জন্য vector ব্যবহার করে এর কোড ২.১ এ দেওয়া হলো।

ফিরিস্তি ২.১: timus1001.cpp

```
১  #include <stdio.h>
২  #include <math.h>
৩  #include <vector>
৪  using namespace std;
৫
৬  int main() {
৭      vector<long long> V;
৮      long long a;
৯      while (scanf("%lld" &a) != EOF) {
১০         V.push_back(a);
১১     }
১২     for (int i = V.size() - 1; i >= 0; i--) {
১৩         printf("%.4lf\n", sqrt(V[i]));
১৪     }
১৫     return 0;
১৬ }
```

এখানে আরেকটা জিনিস। সেটা হল EOF. খেয়াল কর আমাদের ইনপুটে যতক্ষণ সংখ্যা আছে ততক্ষণ আমাদের ইনপুট নিতে হবে। এই কাজ আমরা এই EOF এর মাধ্যমে করতে পারি। যখন ইনপুট নেওয়া শেষ হয়ে যায় তখন তুমি মনে করতে পার যে scanf ফাংশন EOF (End Of File) রিটার্ন করে। সুতরাং আমরা ততক্ষণ ইনপুট প্রসেস করব যতক্ষণ না scanf ফাংশন EOF রিটার্ন করে।

UVa 12108 Extraordinarily Tired Students

সমস্যা: মনে কর একটি ক্লাসে সর্বোচ্চ দশ জন ছাত্রছাত্রী আছে। তারা প্রত্যেকেই ঘুমাচ্ছে। অবশ্য তাদের ঘুমানোর নিজস্ব একটি স্টাইল আছে। প্রত্যেকের স্টাইল আলাদা হতে পারে। তারা a সময় ঘুমায়, b সময় জাগে, a সময় ঘুমায়, b সময় জাগে- এভাবে চলতে থাকে (a ও b সর্বোচ্চ 5 হতে পারে)। তাদের প্রথম অবস্থা 1 হতে a + b এর মাঝে যেকোনোটি হতে পারে। যেমন- a = 2 আর b = 3 হলে এবং তাদের প্রথম অবস্থা 3 হলে তারা 2 সময় জেগে থাকবে (3 এ আছে মানে 2 টা ঘুম ইতোমধ্যেই শেষ এবং 1 টা জাগাও শেষ), পরের 2 সময় ঘুমাবে, 3 সময় জাগবে, 2 সময় ঘুমাবে, 3 সময় জাগবে- এভাবে চলবে। সবার জন্য a ও b দেওয়া থাকবে আর সেই সাথে দেয়া থাকবে শুরুর অবস্থা। বলতে হবে প্রথম কোন সময়ে তারা সবাই জেগে থাকবে। যদি এরকম সময় পাওয়া সম্ভব না হয় তাহলে -1 প্রিন্ট কর।

সমাধান: যারা গণিত পছন্দ কর তাদেরকে বলছি- তোমাদের trade-off শিখতে হবে। Trade-off মানে হলো কোনো একটি সমস্যা পাবার পর সেটা গণিত দিয়েই সমাধান করবা নাকি কিছু দুর গাণিতিক ভাবে করে বাকিটুকু কম্পিউটারের উপর ছেড়ে দিবা- সেটা। যেমন এই সমস্যার ক্ষেত্রে কেউ যদি মনে কর আরে এটা তো মজার গাণিতিক সমস্যা লাগছে, লসাগু লসাগু গন্ধ! হু কথা ঠিক, কিন্তু তাই বলে পুরোটা গাণিতিক ভাবে করা মনে হয় বেশ মুশকিল হবে। প্রতিটি ছাত্রছাত্রীর ঘুমের সাইকেল খুব জোড় 10 (a+b). মানে প্রতি $a + b$ সময় পর পর একই অবস্থা ফিরে আসবে। আমাদের সমস্যায় সর্বোচ্চ দশজন থাকতে পারে, সুতরাং তাদের লসাগু সর্বোচ্চ হতে পারে $lcm(1, 2, \ldots 10)$. এখানে লসাগু কীভাবে আসল বুঝেছ তো? দেখ, প্রথম জন তার $a + b$ সময় পর শুরুর অবস্থায় আসবে, দ্বিতীয় জন তার $a + b$ সময় পর তার শুরুর অবস্থায় আসবে এরকম করে সকলের জন্য আমরা তার সাইকেলের লেংথ জানি। এখন যদি প্রশ্ন করা হয় কত সময় পর সবাই একত্রে শুরুর অবস্থা আসবে তাহলে তা হবে এই সকল সাইকেল লেংথের lcm এর সমান। কারণ lcm কে সব সাইকেল লেংথ ভাগ করে। তার মানে lcm সময় পর সবাই শুরুর অবস্থায় আসবে।

তাহলে তুমি যা করতে পারো তাহলো 1, 2 এভাবে সেই লসাগু পর্যন্ত প্রতিটি সময়ে যেতে পারো এবং সবার ঘুমের state দেখতে পারো। এভাবে লসাগু পর্যন্ত সব সময় চেক করেও যদি সবাই জেগে আছে এরকম কোন সময় না পাও তার মানে এমন কোনো সময় পাওয়া সম্ভব নয়।

যারা আবার অলস তারা আবার কষ্ট করে lcm বের করতে যাবে না। সে জানে যে কোনো একটি বড় মান পর্যন্ত চেক করলেই হল। সুতরাং সে চোখ বন্ধ করে 10^6 বা 10^7 জাতীয় বড় একটি সংখ্যা পর্যন্ত চেক করবে। যদি এর ভিতরেও সমাধান না পাওয়া যায় তার মানে সমাধান নেই!

কোডকে সুন্দর করার জন্য চাইলে "অমুক সময়ে" "অমুক জনের" state কি তা বের করার জন্য একটি ফাংশন লিখতে পারো। অর্থাৎ ঐ ফাংশনকে বর্তমান লোকের a, b এর মান আর বর্তমান সময় দেবে, সে বলে দেবে যে ঐ লোক ঘুমিয়ে আছে না জেগে। এভাবে ফাংশন ফাংশন করে আলাদা আলাদা কোড করলে কোড বুঝতে বা ডিবাগ করতে সুবিধা হয়।

একটা উদাহরন দেই। কিছুদিন আগে এক কন্টেস্টে এক সমস্যায় আমাকে বলেছিল যে n ইনপুট আকারে দেওয়া থাকবে। তোমাকে সেই n এর উপর ভিত্তি করে একটা সমস্যা সমাধান করতে হবে এবং একটি লিস্ট এর সাইজ এবং সেই লিস্ট টি আউটপুট করতে হবে। কোড করার সময় আমি ভেবেছিলাম সমস্যাটা তো সহজ। সুতরাং কষ্ট করে সুন্দর করে কোড না করে সব কোড main এর ভেতরেই করি। কিন্তু যখন WA খেলাম তখন শুরু হল আসল ঝামেলা। ডিবাগ এর জন্য আমি চাইছিলাম n এর বিভিন্ন মানের জন্য লিস্টের সাইজ প্রিন্ট দেই। কিন্তু সমস্যা হল যেহেতু আমি সুন্দর করে কোড করি নাই সেহেতু দেখা গেছে আমি বিভিন্ন কেস বিভিন্ন ভাবে হ্যান্ডল করেছি। আমার আউটপুট ছড়িয়ে ছিটিয়ে বিভিন্ন জায়গায় আছে। সেগুল জায়গা আবার সহজে কমেন্ট করা যাচ্ছে না। এরকম নানান সমস্যা। শেষে আমি একটি ফাংশন লিখলাম এবং সেই ফাংশন হতে vector রিটার্ন করলাম। এবার আমার কাজ সহজ হয়ে গেল। আমার আউটপুট নানা জায়গায় আর ছড়িয়ে ছিটিয়ে নাই। আমি যদি শুরুতেই এই কাজ করতাম তাহলে আমার এতো সময় নষ্ট হত না। সুতরাং চেষ্টা করবা তোমার কোড শুরু থেকেই সুন্দর করে আলাদা আলাদা ফাংসনে ভেঙ্গে ভেঙ্গে লিখতে। আবার খেয়াল রেখ বেশি সুন্দর করতে গিয়ে যেন অনেক বেশি সময় চলে না যায়! সবই trade-off!

UVa 253 Cube painting

সমস্যা: দুটি ঘনকের প্রতিটি পৃষ্ঠের রং দেওয়া আছে। বলতে হবে এদের একটি ঘনককে ঘুরিয়ে ফিরিয়ে অন্য ঘনকের সাথে সদৃশ হয় এরকম orientation এ আনা যাবে কিনা। অর্থাৎ ঘুরানর পর দুটি ঘনকের উপরের পিঠের রং একই হবে, নিচের রং একই হবে, ডান বাম সামনে পিছে সব পৃষ্ঠে দুটি ঘনকের রং একই হবে- এরকম অবস্থানে আনা যাবে কিনা।

সমাধান: এটা আমার বেশ পছন্দের সমস্যা। কারণ যদি কেউ "সহজে" এটার কোড না করতে পারে তাহলে অন্তত সে অনেক খেটে খুটে অনেক বিশাল কোড করে সমাধান করতে পারবে। আবার কেউ একটু বুদ্ধি খাটিয়ে কোডটাকে একটু ছোট করে ফেলতে পারে। কেউ আবার একটু বেশি বুদ্ধি খাটিয়ে আরও ছোট করতে পারবে। যত বেশি বুদ্ধি খাটাবে তত ছোট হবে। কিন্তু সেই বুদ্ধিগুলি বের করতেও

কিন্তু সময় লাগবে। সুতরাং trade-off! তুমি কি সময় ব্যয় করে কোড ছোট করবে, নাকি dumb এর মত কম বুদ্ধিওয়ালা সমাধান লিখবে?

বড় কোড এবং সবচেয়ে কম বুদ্ধি ওয়ালা সমাধান কী তাতো বুঝতেই পারছ? নিজে থেকে কোনো ঘনককে যত ভাবে পারো ঘুরাও আর তাদের সবার জন্য একটি করে if-else লিখ। অর্থাৎ মনে মনে প্রথম ঘনককে যত ভাবে সম্ভব ঘুরাও। ঘুরানর পর তুমি জান এখন এর উপর/নিচ/ডান/বাম/সামনে-/পিছে কোন রঙ আছে। এখন তুমি সেই রঙগুলিকে দ্বিতীয় ঘনকের রঙের সাথে তুলনা করার জন্য if-else লিখ। যদি ভুল না করে থাকি তাহলে এজন্য তোমাকে 24টি if লিখতে হবে (আর তার ভেতরে 6টির মত == লজিক)। দ্বিতীয় ঘনককে তো আর ঘুরানর প্রয়োজন নেই। শুধু প্রথম ঘনককে মনে মনে ঘুরাবে। প্রথম ঘনকের ছয়টি পৃষ্ঠকে একে একে মনে মনে সামনের পৃষ্ঠ হিসাবে স্থির কল্পনা করবে। এখন এর বাকি পৃষ্ঠগুলিকে ঘুরিয়ে ঘুরিয়ে পরিবর্তন করে বিভিন্ন orientation পেতে পারো। যদি বুঝতে সমস্যা হয় তোমরা xyz axis কল্পনা কর। ঘনকের ছয়টি পৃষ্ঠ ছয় axis এ আছে (x+, x-, ...). এখন মনে কর x+ এর দিকের পৃষ্ঠ স্থির। তাহলে এই ঘনককে x axis এর সাপেক্ষে ঘুরালে মোট চারটি orientation পাওয়া যাবে (চারটি পৃষ্ঠ একে একে y+ এর দিকে আসবে- এভাবেও ভাবতে পারো)। নিঃসন্দেহে খুব একটা সহজ কাজ না। একে তো ঘনককে ঘুরিয়ে ঘুরিয়ে আঁকতে হবে কাগজে কলমে বা কল্পনায়, তারপর আবার সেটা এক গাদা if else করে লিখতেও হবে। তবে আর কোনো বুদ্ধি না পেলে এটাই করতে হবে। একটু সুন্দর বা সহজ করার জন্য যা করতে পারো তাহলো এতগুলো if-else না লিখে একটি 24×6 সাইজের অ্যারেতে সংখ্যা গুলি লিখে রাখো (অর্থাৎ দ্বিতীয় ঘনকের i তম পৃষ্ঠের সাথে কে মিলে এই তথ্য গুলি)। তাহলে একটি loop চালিয়ে এবং একটি if লিখেই কাজ সেরে ফেলতে পারো।

একটু কম ugly উপায় হলো খেয়াল করা যে ঘনকের পৃষ্ঠগুলি একটু বিশেষ ভাবে নাম্বারিং করা (statement এর চিত্র দেখো)। 1 আর 6 বিপরীত দিকে, 2 আর 5 বিপরীত দিকে এবং 3 আর 4 ও। এদের প্রত্যেক জোড়ার যোগফল কিন্তু 7. সুতরাং, আমরা যদি ঠিক করি উপরে, পিছনে আর ডানে কে তাহলে আমরা সেখান থেকে নিচ, সামনে আর বামে কে আছে তা বের করে ফেলতে পারি। সুতরাং এর আগের মত 24×6 এর দরকার নেই 24×3 ই যথেষ্ট, কারণ বাকি তিনটি তুমি এই ফর্মুলা ব্যবহার করে বের করতে পারবে।

আর কীভাবে সহজ করা যায়? খেয়াল কর, তুমি যদি উপরের পৃষ্ঠ ঠিক করে ফেল তাহলে চারিদিকে কি কি থাকবে তা কিন্তু fix হয়ে গেছে। যেমন উপরে যদি 1 থাকে তাহলে চারিদিকে থাকবে 2, 3, 5, 4. এখন এদেরকে যদি circular ভাবে একটি অ্যারে তে রাখো তাহলে কিন্তু পাশাপাশি দুটি সিলেক্ট করে তাদেরকে পিছন ও ডান পৃষ্ঠ কল্পনা করতে পারো। অর্থাৎ প্রতিটি পৃষ্ঠ উপরের দিকের জন্য তুমি একটি অ্যারে (4 দৈর্ঘ্যের) রাখো যে সেক্ষেত্রে চারিদকে কোন কোন পৃষ্ঠ থাকবে। যেমন উপরে 1 থাকলে চারিদিকে থাকবে 2, 3, 5 আর 4. উপরে 4 থাকলে চারিদিকে থাকবে 2, 1, 5 আর 6 এরকম করে সকল পৃষ্ঠ উপরে থাকার জন্য চারিদিকে কে থাকবে তার একটি লিস্ট বানিয়ে ফেলতে পারি। এরপর এই অ্যারের উপর লুপ চালিয়ে নির্বাচন করতে পারি পিছনে কে থাকবে, তাহলে তার পরের জন থাকবে ডানে। আর উপর, পিছন ও ডান নির্দিষ্ট হয়ে গেলে তো বাকিগুলো নির্দিষ্ট হয়ে যায় তাই না? তাহলে এই পদ্ধতিতে কত বড় অ্যারে লিখতে হচ্ছে? 6×4. বেশ কম! হয়তো এর থেকেও কমানো সম্ভব। চিন্তা করে দেখতে পারো, অথবা এটাই কোড করে ফেলতে পারো! সময় বাঁচবে।

Timus 1005 Stone Pile

সমস্যা: n টি সংখ্যা দেয়া আছে যেখানে $n \leq 20$ এবং সংখ্যাগুলি সর্বোচ্চ $100,000$ হতে পারে। তোমাকে এই সংখ্যাগুলিকে দুভাগে ভাগ করতে হবে। এবার প্রতিটি ভাগের সংখ্যাগুলিকে যোগ কর, ধরা যাক যোগফলগুলি হলো A ও B. তোমাদের লক্ষ্য হবে A ও B এর পার্থক্য যেন সবচেয়ে কম হয়।

সমাধান: অনেকভাবেই এই সমস্যা সমাধান করা যায়। খেয়াল কর সংখ্যাগুলি কিন্তু বেশ ছোট $(100,000)$। তাই আমরা একে coin change dp এর মত করে সমাধান করতে পারি। প্রদত্ত সংখ্যাগুলি খুব জোড় একবার ব্যবহার করে আমরা কোন কোন যোগফল বের করতে পারি - এই সমস্যাকে dp এর মাধ্যমে

করে ফেলা যায়।[১] এই DP করলে আমরা জানতে পারব A এবং B এর কি কি মান সম্ভব। আর এ জিনিস জানলে তো আমাদের সমাধান হয়েই যাবে!

এছাড়াও আমরা চাইলে একে backtrack করে সমাধান করতে পারি। আমরা একে একে 1 হতে n পর্যন্ত প্রতিটি স্থানে গিয়ে সিদ্ধান্ত নিব যে এই স্থানের সংখ্যা কোন ভাগে নেব। ধরা যাক আমাদের backtrack ফাংশনের parameter হবে দুটি সংখ্যা। প্রথমটি বলবে আমরা এখন কোন সংখ্যায় আছি (আমরা কিন্তু একে একে 1-তম হতে n-তম সংখ্যায় যাচ্ছি) আর দ্বিতীয় সংখ্যা বলবে আমরা যেই দুভাগে ভাগ করছি তার প্রথম ভাগে থাকা সংখ্যাগুলির যোগফল কত। চাইলে দ্বিতীয় ভাগের যোগফলের জন্য আরও একটি প্যারামিটার নিতে পার আবার চাইলে সবশেষে গিয়ে প্রথম ভাগের যোগফলকে মোট যোগফল হতে বাদ দিলেই তুমি দ্বিতীয় ভাগের যোগফল পেয়ে যাবে। যাই হোক, এভাবে করলে আমাদের time complexity $\mathrm{O}(2^n)$ হবে। কারণ আমরা n টি সংখ্যায় গিয়ে দুভাবে চেষ্টা করছি। প্রথমবার চেষ্টা করছি একে প্রথম ভাগে ফেলতে, পরের বার চেষ্টা করছি একে দ্বিতীয় ভাগে ফেলতে। সুতরাং n বার আমরা 2 ভাবে চেষ্টা করছি অর্থাৎ 2^n (যদি "কেন 2^n"- এটা বুঝতে সমস্যা হয় তাহলে তোমরা combinatorics বা discrete math নিয়ে একটু পড়াশুনা করতে পারো)।

Backtrack না করেও আমরা খুব সহজে এটা সমাধান করতে পারি। 0 হতে $2^n - 1$ এর মাঝে প্রতিটি সংখ্যার binary representation কল্পনা কর[২]। অর্থাৎ এই সংখ্যাগুলিকে n bit এর binary সংখ্যা হিসেবে কল্পনা কর। n টি 0 আর 1 দিয়ে তৈরি সব সংখ্যা তুমি পেয়ে যাবে। আমরা একে একে এই সকল সংখ্যা গুলিকে প্রসেস করব। মনে কর একটি সংখ্যা নিলাম। এবার এর 0 ওয়ালা স্থান গুলিকে প্রথম গ্রুপে আর 1 ওয়ালা স্থান গুলিকে দ্বিতীয় গ্রুপে ফেল। প্রতি ভাগের যোগফল বের কর। এরপর তাদের মাঝের পার্থক্য বের কর। উদাহরণ স্বরূপ মনে কর যে আমাদের n হলো 2. তাহলে 0 হতে $2^2 - 1 = 3$ পর্যন্ত সংখ্যা গুলি (এবং তাদের binary রূপ) হলো $0(00), 1(01), 2(10), 3(11)$. 00 এর মানে হলো আমাদের দুটি সংখ্যাকেই প্রথম গ্রুপে নিয়েছি, 10 এর মানে হলো আমরা প্রথম সংখ্যাকে (আশা করি একটু চিন্তা করলে বুঝতে পারবে যে i তম least significant bit, i তম সংখ্যা নির্দেশ করে যেখানে আমরা সংখ্যাগুলিকে 0 indexed ভাবে কল্পনা করছি, মানে প্রথম সংখ্যা আছে 0 তম বিটে) প্রথম গ্রুপে আর দ্বিতীয় সংখ্যাকে দ্বিতীয় গ্রুপে কল্পনা করছি। এভাবে বাকি 2 আর 3 কেউ একই ভাবে প্রসেস করতে পারি। এই সমাধানের time complexity হবে $\mathrm{O}(n2^n)$ কারণ তোমরা 0 হতে $2^n - 1$ এর লুপ চালানোর পর আরও একটি n দৈর্ঘ্যের লুপ চালিয়ে 0-1 দেখে দেখে দুভাগে ভাগ করছ। Backtrack এর ক্ষেত্রেও তুমি যদি ভাগ করার পর আবার লুপ চালিয়ে দেখতে কাকে কোন গ্রুপে ফেলেছ তাহলেও এই অতিরিক্ত n চলে আসতো তোমার complexity তে। কিন্তু উপরের backtrack সমাধানে আমরা তা না করে যখন কোনো সংখ্যাকে প্রথম ভাগে ফেলেছি তখনই প্রথম ভাগের যোগফলকে আপডেট করেছি। ফলে আমাদের আর অতিরিক্ত n এর লুপ লাগে নাই। কিন্তু আমাদের একটু আগে বলা bitmask এর সমাধানে আমরা অতিরিক্ত একটি লুপ চালাচ্ছি 0-1 এ ভাগ করার জন্য। এজন্য আমাদের সমাধানে অতিরিক্ত n চলে এসেছে।

এখন কথা হলো bitmask পদ্ধতির সমাধান হতে আমরা অতিরিক্ত n বাদ দিতে পারি কি না। মনে কর mask নামের ভ্যারিয়েবলকে (bitmask এর সংক্ষিপ্তরূপ) 0 হতে $2^n - 1$ পর্যন্ত লুপ চালাচ্ছি এবং প্রতিটি mask এর জন্য আমরা দুই ভাগের সংখ্যার যোগফল বের করি (আমরা mask এর বিট গুলি দেখে ঠিক করে কে কোন ভাগে যাবে)। এর আগেই বলেছি, আমাদের আসলে দুই ভাগের যোগফল রাখার দরকার নেই, শুধু এক ভাগের যোগফল জানলে আমরা মোট যোগফল হতে অপর ভাগের যোগফল সহজেই বের করতে পারব। সুতরাং মনে কর আমরা শুধু যেই ভাগে 1 বিট ওয়ালা সংখ্যা আছে তাদের যোগফল জানি। এখন ধর আমরা এখন $mask = x$ এ আছি। এর মানে $mask < x$ এর জন্য ইতোমধ্যেই আমরা 1 বিটের ভাগের সংখ্যার যোগফল জানি। এখন যখন $mask = x$ এ এলে তখন এর যেকোনো একটি on (1) বিট নাও। একে শূন্য করে দাও ($mask\hat{} = 1$). এই নতুন mask এর sum কিন্তু তুমি জানো কারণ on বিটকে শূন্য করে দেয়ার ফলে তোমার নতুন mask টি বর্তমান mask এর থেকে ছোট হয়ে গিয়েছে। তাহলে এই ছোট mask এর জন্য আমাদের আগে পাওয়া যোগফল এর সাথে আমাদের on বিটের সংখ্যা যোগ করে আমরা নতুন mask এর জন্য যোগফল বের করে ফেলতে পারি। কিন্তু কথা হল এই on বিট বের করতেই তো আমাদের n এর

[১] প্রোগ্রামিং কন্টেস্ট: ডেটা স্ট্রাকচার ও অ্যালগরিদম বইয়ের Dynamic Programming সেকশনে Coin change dp এর নানা variation পাবে।

[২] এই পদ্ধতিকে আমরা bitmask পদ্ধতি নাম দিতে পারি কারণ আমরা "all possible" bitmask এর উপর দিয়ে লুপ চালাচ্ছি।

একটি লুপ লাগবে। না তা লাগবে না যদি তুমি একটু বুদ্ধি করতে পারো। খেয়াল কর আমাদের কিন্তু যেকোনো on বিট নিলেই চলবে। আমরা কি খুব সহজে বলতে পারি অমুক সংখ্যার অমুক বিটটা on আছে? পারি, 2 হতে 3 এর 2nd bit টা সবসময় on, 4 হতে 7 এর 3rd bit টা, 8 হতে 15 এর 4th bit টা এরকম করে 2^{i-1} হতে $2^i - 1$ এর i তম বিটটা সব সময় on. তাহলে আমরা সরাসরি 0 হতে $2^n - 1$ এর লুপ না চালিয়ে 0 হতে $n - 1$ পর্যন্ত i এর একটা লুপ চালাবো। আর এর ভেতরে আরেকটা লুপ চালাবো সেই সেই mask দিয়ে যাদের i তম বিট আমাদের এই observation এ on অর্থাৎ 2^i হতে $2^{(i+1)} - 1$ পর্যন্ত। তাহলেই complexity $O(2^n)$ হয়ে যাবে। তোমাদের সুবিধার জন্য backtrack আর bitmask পদ্ধতির কোড ২.২ এ দেওয়া হলো। তোমরা চাইলে একে অন্য ভাবেও কোড করতে পার।

ফিরিস্তি ২.২: timus1005.cpp

```cpp
int num[22], n;
int onBit[1 << 20], sum[1 << 20];

void backtrack(int at, int group1Sum) {
    if (at == n) {
        // Find group2Sum
        // Keep a global variable to track answer.
        return;
    }
    // Don't take num[at].
    backtrack(at + 1, group1Sum);
    // Take num[at].
    backtrack(at + 1, group1Sum + num[at]);
}

int main() {
    scanf("%d", &n);
    for (int i = 0; i < n; i++) {
        scanf("%d", &num[i]);
    }

    bktk(0, 0); // Solution in backtrack.

    // Solution in bitmask.
    sum[0] = 0;
    for (int i = 0; i < n; i++) {
        for (int mask = (1 << i); mask < (1 << (i + 1))
            ; mask++) {
            // i is the on bit.
            sum[mask] = sum[mask ^ (1 << i)] + num[i];
            // Update answer, one part is: sum[mask]
                and another part is
            // total - sum[mask].
        }
    }
    return 0;
}
```

শেষ করার আগে একটা জিনিস। তোমরা চাইলে ধরে নিতে পারো যে n তম জিনিস সবসময় প্রথম গ্রুপে থাকবে, এরপর বাকি জিনিসগুলিকে সব ভাবে দুভাগে ভাগ করবে। তাহলে complexity $O(2^{n-1})$ হয়ে যাবে।

Timus 1014 Product of Digits

সমস্যা: 0 হতে 10^9 রেঞ্জের একটি সংখ্যা N দেয়া থাকবে। তোমাকে সবচেয়ে ছোট ধনাত্মক সংখ্যা Q আউটপুট দিতে হবে যেন Q এর অঙ্কগুলির গুণফল N হয়। যদি এমন কোন সংখ্যা না পাওয়া যায় তাহলে -1 প্রিন্ট করতে হবে। যেমন N = 10 এর জন্য Q হবে 25.

সমাধান: প্রথমত দেখ Q এর ডিজিটগুলি গুন করলে N পাওয়া যায়। অর্থাৎ আমাদের জানা দরকার কোন কোন সংখ্যা দিয়ে N কে ভাগ করা যায়। যদি N কে $2, 3, \ldots 9$ দিয়ে যতবার ভাগ যায় ততবার ভাগ দেবার পরও দেখো যে N তখনও 1 হয় নাই এর মানে এর জন্য আসলে কোনো Q পাওয়া সম্ভব নয়। যেমন N যদি 26 হয় তাহলে দেখবে 2 হতে 9 সব সংখ্যাদিয়ে যত বার ভাগ দেওয়া সম্ভব ভাগ দেওয়া হয়ে গেলে 13 বাকি থাকে। এর মানে এর জন্য আসলে কোনো Q খুঁজে পাওয়া যাবে না। কেন? কারণ Q এর ডিজিটগুলিতো আসলে 2 হতে 9 ই হবে তাই না? 0 আর 1 বিশেষ ক্ষেত্র ছাড়া ব্যবহার যে হবে না তা মনে হয় একটু চিন্তা করলেই বুঝা যায়। সুতরাং কোডের শুরুতেই তাদের "specially handle" করে ফেল। তাহলে আমাদের সংখ্যা শুধুমাত্র 2 হতে 9 দিয়ে ভাগ যাবে। বা আমরা আরেকটু চিন্তা করে বলে দিতে পারি যে N সংখ্যাটি 2, 3, 5 আর 7 prime গুলি দিয়ে ভাগ যাবে, আর অন্য কোনো prime দিয়ে নয়। সুতরাং আমরা বের করে ফেলি N কে 2, 3, 5 আর 7 দিয়ে কয়বার করে ভাগ করা যায়। এখন খেয়াল কর আমরা চাইলে 3টি 2 কে একত্র করে 8 বানাতে পারি, এতে করে আমাদের উত্তর এর ডিজিট সংখ্যা কিন্তু 2 ঘর কমে যায়। আবার আমরা চাইলে দুটি 2 কে একত্র করে 4 বানাতে পারি, 2 আর 3 কে একত্র করে 6 বানাতে পারি অথবা দুটি 3 কে একত্র করে 9 বানাতে পারি। সুতরাং তোমাকে চিন্তা করতে হবে তুমি কীভাবে এদেরকে combine করবে যাতে সবচেয়ে ছোট সংখ্যা পাওয়া যায়। আর আশা করি এটা বুঝছ যে ডিজিটগুলিকে ছোট হতে বড় অর্ডারে সাজিয়ে Q বানাতে হবে (758 বানানোর চেয়ে 578 বানানো লাভজনক)। এই পর্যায়ে এসে তুমি নিজে থেকে একটু চিন্তা কর কীভাবে তুমি 2, 3, 5 ও 7 দের combine করবে। আসলে 5 আর 7 কে তো combine করার কিছু নেই, combine করতে হবে 2 আর 3 কে। কারণ 5 আর 7 কে কারও সাথে combine করতে গেলে তা আর 1 ডিজিটে থাকে না।

এখন নানা case চিন্তা করা ছাড়া আসলে কোনো উপায় নেই। যদি শুধু একটা 2 বা একটা 3 থাকে তাহলে তো কিছু করার নেই। যদি শুধু এক গাদা 2 থাকে তাহলে তো তিনটা তিনটা করে নিয়ে যতক্ষণ সম্ভব 8 বানানো লাভজনক। যদি এরপর 4 বানানো সম্ভব হয় তাহলে 4 নাহলে আবার 2. কেন? তুমি $2^1, 2^2, 2^3 \ldots 2^10$ এরকম 10 টি N এর জন্য চিন্তা কর তাহলেই বুঝবে। কন্টেস্টে সবসময় প্রমাণ করার দরকার হয় না। যদিও এটা ঠিক যে প্রমাণ না করে কোনো কিছু করা risky কিন্তু তাও, প্রমাণ অনেক সময় সাপেক্ষ ব্যাপার। এর থেকে কিছু উদাহরন হাতে হাতে করে যদি তুমি নিজেকে বুঝ দিতে পার যে এটাই ঠিক- তাহলেই হবে। এর মানে এই না যে এরকম করলে ভুল হয় না! যাই হোক, একই ভাবে 3 এর ক্ষেত্রেও চিন্তা করতে পার তবে সেটা অনেক সহজ। এখন সমস্যা হবে যখন 2 ও থাকবে আবার 3 ও থাকবে। আবারো কিছুক্ষণ case analysis করলে বুঝবে প্রথমে 9 বানানোর চেষ্টা করতে হবে, এরপর 8, এরপর 6, 4 এরকম। আগেই বলেছি কেন এরকম হলো, প্রমাণ কি, এরকম ক্ষেত্রে কীভাবে চিন্তা করব এতসব প্রশ্ন করে সবসময় লাভ নেই। Case analysis করে intuition এর ভিত্তিতে সমাধান দাঁড় করাতে হবে।

যদি codejam বা hackercup এর মত contest হয় যেখানে একবারই submit করা যায় সেসব ক্ষেত্রে তোমরা চাইলে একটি dp বা backtrack কোড করে দেখবে তোমার claim টা কি ধরা যাক N = 1 হতে 1000 পর্যন্ত সঠিক কিনা। যদি হয় তাহলে তুমি মোটামোটি সাহস পাবে যে হয়তো তোমার সমাধান বড় N এর জন্যও সঠিক।

Timus 1020 Rope

সমস্যা: N (≤ 100) টি একই মাপের খুঁটিকে (সব খুঁটিই R ব্যাসার্ধের বৃত্ত) একটি convex polygon এর N টি বিন্দুতে রাখা আছে। একটি দড়িকে এই খুঁটিগুলির চারপাশ দিয়ে টানটান করে টানানো হলে সেই দড়ির দৈর্ঘ্য কত?

সমাধান: খুবই চমৎকার একটি সমস্যা। খেয়াল কর দড়ির কিছু অংশ টানটান আর কিছু অংশ গোল গোল। টানটান অংশ হল দুটি বৃত্তের মাঝে স্পর্শক। যেহেতু বৃত্তগুলি একই ব্যাসার্ধের সেহেতু স্পর্শকের দৈর্ঘ্য তাদের কেন্দ্রের দূরত্বের সমান হবে। এর মানে দড়ির সোজা অংশগুলির দৈর্ঘ্য হবে convex polygon এর পরিধির সমান। কিন্তু গোল অংশগুলি? এটাও খুব একটা কঠিন না। খেয়াল কর সবগুলো গোল অংশ জোড়া লাগালে একটি বৃত্ত হয়। এটাকে এভাবেও চিন্তা করতে পারো, মনে কর তুমি দড়ি ধরে হাঁটা শুরু করলে। গোল অংশগুলোতে এসে তুমি R ব্যাসার্ধের বৃত্ত ধরে একটু ঘুরছ এভাবে করতে করতে আবার শুরুর জায়গায় আসবে। এই পথে তুমি কিন্তু ঘুরে ঘুরে 360° ঘুরেছ বা R ব্যাসার্ধের বৃত্ত পুরোটা ঘুরেছ। বা তুমি চাইলে মনে মনে লম্বা অংশগুলি বাদ দিয়ে গোল অংশগুলিকে জোড়া লাগায়ে দেখতে পারো। সুতরাং আমরা যদি বৃত্তের পরিধি এবং পলিগনের পরিধি যোগ করি তাহলে মত দড়ির দৈর্ঘ্য পেয়ে যাব।

Timus 1044 Lucky Tickets. Easy!

সমস্যা: কতগুলি n (≤ 9 এবং n একটি জোড় সংখ্যা) ডিজিটের সংখ্যা আছে যাদের অঙ্কগুলির মাঝে প্রথম অর্ধেকের যোগফল শেষ অর্ধেকের সমান হয়। যেমন 1322 এখানে 4 টি অংক আছে যার প্রথম দুটি অংকের যোগফল আর শেষ দুটি অংকের যোগফল সমান।

সমাধান: এসব সমস্যার ক্ষেত্রে সবার আগে দেখে নিবে যে leading zero কি allowed কিনা। যেমন এই সমস্যার ক্ষেত্রে allowed. এখন n যদি জোড় হয় তাহলে সমস্যাটা বেশ সহজ। $n/2$ দৈর্ঘ্যের সবগুলি সংখ্যার জন্য তার অংকগুলির যোগফল আমাদের বের করতে হবে। এরপর আমরা বের করব কোন যোগফল কতভাবে হয়। ধরা যাক S যোগফল হয় k ভাবে। তাহলে k ভাবে প্রথম অর্ধেক আর আরও k ভাবে পরের অর্ধেকে আমরা S যোগফল ওয়ালা সংখ্যা বসাতে পারি, অর্থাৎ k^2 ভাবে আমাদের দুই অর্ধেকের অংকের যোগফল একই হতে পারে। এভাবে প্রতিটি S এর জন্য আমাদের k^2 বের করে তাদের যোগ করলেই হয়ে যাবে। আর কীভাবে $n/2$ দৈর্ঘ্যের সকল সংখ্যাগুলি প্রসেস করবে? তাতো বেশ সহজ, Timus 1005 এর সমাধানের মত তোমরা চাইলে backtrack করতে পারো, বা চাইলে লুপ চালিয়েও করতে পারো। যেহেতু n এর মান সর্বোচ্চ 9 সেহেতু $n/2$ এর সর্বোচ্চ মান মাত্র 4. অর্থাৎ চার ডিজিটের সকল সংখ্যার যোগফল আমাদের বের করতে হবে। এটা তো কোনো ব্যাপারই না তাই না? মনে কর তোমরা লুপের মাধ্যমে $n/2$ দৈর্ঘ্যের সকল সংখ্যার অংকগুলির যোগফল বের করছ। অর্থাৎ 0 হতে $10^{n/2}-1$ পর্যন্ত লুপ। আমরা কিন্তু চাইলে Timus 1005 এর মত এই লুপ চালানোর পর অংকগুলির উপর লুপ নাও চালাতে পারি। মনে কর আমরা চাইছি x এর অংক সমূহের যোগফল বের করতে। আমরা লিখতে পারি $sum[x] = sum[x/10] + (x\%10)$. তাহলেই কিন্তু x এর অংকগুলির যোগফল বের হয়ে যাবে। আলাদা করে অংকগুলির উপর লুপ চালানোর দরকার নেই। এই সমস্যায় হয়তো এই optimization এর দরকার নেই, কিন্তু অন্য কোনো সমস্যায় কাজে লেগে যেতে পারে।

Timus 1197 Lonesome Knight

সমস্যা: একটি দাবার বোর্ডে একটি ঘোড়ার স্থান দেওয়া আছে। বলতে হবে দাবার বোর্ডের কয়টি ঘরকে এই ঘোড়া আক্রমণ করে আছে। ইনপুট দেওয়া হবে a1, g6 এরকম করে।

সমাধান: প্রথমত আমাদের ইনপুটকে প্রসেস করে বের করতে হবে যে আমাদের ঘোড়া কোন row আর কোন column এ আছে। এই প্রসেস করা তো বেশ সহজ। তোমরা ঘোড়ার স্থানকে একটি string

এ ইনপুট নেবে আর এই string এর প্রথম character থেকে a এর ascii মান বাদ দেবে আর দ্বিতীয় character হতে 1 এর ascii মান বাদ দেবে (word[0] - 'a' এরকম)। মনে করা যাক আমাদের ঘোড়া আছে (r, c) তে। এখন এখান হতে ঘোড়াটি কোন কোন ঘর আক্রমণ করতে পারে? এসব ঘরকে আমরা এভাবে লিখতে পারি $(r+2, c+1)$, $(r+2, c-1)$, $(r-2, c+1)$ এরকম করে মোট 8 টি। $(r+2, c+1)$ এর মানে তো বুঝছ? এর মানে হল বর্তমান রো হতে দুই রো নিচে এবং বর্তমান কলাম হতে এক কলাম ডানের ঘর। এরকম করে একটি ঘোড়া 8 টি মুভ দিতে পারে আর আমরা এই প্রতিটি মুভ কেই এরকম $(r+2, c+1)$ এর মত করে প্রকাশ করতে পারি।

কিন্তু সমস্যা হল এই 8 টি ঘরের প্রতিটিই আমাদের দাবার বোর্ডে নাও থাকতে পারে। যেমন আমাদের বর্তমান সেল যদি হয় a1 তাহলে কিন্তু $(r-2, c-1)$ বলে কিছু নেই। সুতরাং আমাদের চেক করে দেখতে হবে কোন কোন ঘর দাবার বোর্ডে আছে ($0 \leq newR \leq 7$ এবং কলামের ক্ষেত্রেও একই ধরনের চেক)।

আমরা চাইলে এই নতুন অবস্থান গুলি হাতে হাতে লিখতে পারি। অথবা চাইলে সহজে কোড করার জন্য row বরাবর অবস্থান পরিবর্তন এবং column বরাবর অবস্থানের পরিবর্তনকে দুটি অ্যারেতে রেখে দিতে পারি: $dr[] = \{2, 2, -2, -2, 1, 1, -1, -1\}$ ও $dc[] = \{1, -1, 1, -1, 2, -2, 2, -2\}$. এরপর i এর একটি লুপ চালাব 0 হতে 7 পর্যন্ত এবং চেক করে দেখব $(r+dr[i], c+dc[i])$ আমাদের বোর্ডের ভেতরে আছে কিনা। এভাবে খুব চমৎকার ভাবে অনেক ছোট করে কোড হয়ে যাবে।

UVa 202 Repeating Decimals

সমস্যা: a ও b দুটি সংখ্যা দেয়া আছে ($0 \leq a \leq 3000, 1 \leq b \leq 3000$). তোমাকে a/b কে পৌনপনিক আকারে প্রকাশ করতে হবে। যেমন: $2/15 = 0.1(3)$.

সমাধান: এটি আমার অন্যতম পছন্দের একটি সমস্যা। কারণ আমাদের দেশের বেশির ভাগ কলেজ বা বিশ্ববিদ্যালয় পড়ুয়া ছাত্র ছাত্রীদের এটি সমাধান করতে দিলে বেশ confusion এ পরে যাবে কিন্তু এর সমাধান স্কুল পড়ুয়ারাও বুঝবে বা চেষ্টা করলে নিজে থেকে বের করতে পারবে বা আসলে স্কুল কলেজ থেকেই আমরা এই ব্যাপারে ধারনা নিয়ে এসেছি।

আমরা আসলে ভাগের সময় কী করি? ভাগ দেই, ভাগশেষের পিছে একটি শূন্য বসাই (হর লবের থেকে বড় ধরে নিচ্ছি), আবার ভাগ দেই, আবার ভাগশেষের পিছে শূন্য বসাই এভাবে চলতে থাকে। এখন খেয়াল কর যদি কখনও ভাগশেষ পুনরাবৃত্তি হয় তাহলে কিন্তু ভাগফলে আমরা যেই ডিজিটগুলি বসিয়েছি তারাও কিন্তু পুনরাবৃত্তি হবে, অর্থাৎ পৌনোপুনিক। এখন বলত যেকোনো a কে b দিয়ে ভাগ করলেই কী এই পৌনোপুনিক পাওয়া যাবে? হ্যাঁ। কারণ তোমার ভাগশেষ তো আসলে 0 হতে $b-1$ এর মাঝেই হবে তাই না? যেহেতু মাত্র b রকমের ভাগশেষ হতে পারে সেহেতু এক সময় না এক সময় পুন্রাব্রিত্তি হবেই। কি বিশ্বাস হচ্ছে না? মনে কর আমরা 2 কে 15 দিয়ে ভাগ করছি। প্রথমে তো ভাগশেষ 2 নিজেই। এর পর 5, এর পর 5 এবং 5 চলতেই থাকে। এ জন্য আসলে দ্বিতীয় ডিজিটের পর পৌনোপুনিক চলতে থাকবে $0.1(3)$. এবার একটু কঠিন উদাহরণ নেয়া যাক, 655/990. প্রথমে ভাগশেষ 655, এর পর 610, এর পর 160, এর পর 610. তাহলে এর পর থেকে 160, 610, 160, 610 এরকম চলতে থাকবে। অর্থাৎ আমাদের প্রথম ডিজিটের পর পরের দুটি ডিজিট বার বার পুনরাবৃত্তি হতে থাকবে। আমাদের উত্তর হবে $0.6(61)$. যেহেতু b খুব একটা বড় না সেহেতু নানা ভাবে এর সমাধান করা সম্ভব।

মনে কর আমরা একটি 0/1 এর অ্যারে রেখেছি। শুরুতে সব স্থানে 0 থাকবে। এখন আমরা ভাগ দেওয়া শুরু করব। ভাগের পরে যেই ভাগশেষ থাকবে আমরা অ্যারের সেই ভাগশেষের স্থানে গিয়ে 1 লিখে রাখব। কিন্তু যদি দেখি আগেই 1 আছে এর মানে আমাদের ভাগ সাইকেলে পরেছে। আর আমরা এখন জেনে গেছি এই সাইকেল কোন ভাগশেষ থেকে শুরু হয়েছে, ধরা যাক এই ভাগশেষ হল x। এখন আবার আমরা এই কাজটুকু করব। আবার শুরু থেকে ভাগ করা শুরু করি। যখন আমরা x কে ভাগশেষ হিসাবে পাব এর মানে আমরা এখন সাইকেলে ঢুকছি। এরপর আবার যখন x পাব, এর মানে আমাদের সাইকেল শেষ। এভাবে আমরা পুরো ভাগফলকে পৌনপনিক হিসাবে প্রিন্ট করতে পারি।

২.১ অনুশীলনী

২.১.১ সমস্যা

Simple

- Timus 1264 Workdays
- UVa 1583 Digit Generator
- Timus 2012 About Grisha N
- UVa 10082 WERTYU
- UVa 1584 Circular Sequence
- UVa 1586 Molar mass
- UVa 455 Periodic Strings
- UVa 133 The Dole Queue
- UVa 10340 All in All
- UVa 1588 Kickdown
- UVa 1339 Ancient Cipher
- UVa 1591 Data Mining
- UVa 272 TEX Quotes
- Timus 1082 Gaby Ivanushka
- Timus 1293 Eniya
- Timus 1409 Two Gangsters
- Timus 2023 Donald is a postman
- UVa 401 Palindromes
- UVa 1585 Score
- UVa 1225 Digit Counting
- UVa 227 Puzzle
- UVa 1368 DNA Consensus String
- UVa 1587 Box
- UVa 11809 Floating-Point Numbers
- UVa 489 Hangman Judge
- UVa 815 Flooded!
- Timus 1068 Sum

Easy

- Timus 1025 Democracy in Danger
- Timus 1079 Maximum
- Timus 1319 Hotel
- UVa 232 Crossword Answers
- UVa 512 Spreadsheet Tracking
- UVa 220 Othello
- UVa 508 Morse Mismatches
- Timus 1313 Some Words about Sport
- Timus 1149 Sinus Dances
- UVa 340 Master-Mind Hints
- UVa 213 Message Decoding
- UVa 201 Squares
- UVa 1590 IP Networks
- UVa 509 RAID!

Medium

- UVa 12412 A Typical Homework
- UVa 1589 Xiangqi

২.১.২ হিন্ট

UVa 272: সমস্যাটা কঠিন না। বিভিন্ন জন বিভিন্ন ভাবে ইনপুট নেবে। আমি যে ভাবে নেই তাহলো gets ব্যবহার করে। আমি আসলে scanf, printf আর gets এর বাইরে কিছু ব্যবহার করি না ইনপুট আউটপুটের জন্য। এসব ব্যবহারের জন্য যে সাবধানতা অবলম্বন করতে হয় তা নিয়ে আমি ইতোমধ্যেই "প্রোগ্রামিং কন্টেস্ট ডেটা স্ট্রাকচার ও অ্যালগরিদম" বইয়ে বলেছি। সুতরাং তা আবারো পুনরাবৃত্তি করছি না, শুধু জানিয়ে রাখলাম যে gets ব্যবহার করলে একটু সাবধান হতে হয়।

Timus 1068: ভুলে যেও না যে N কিন্তু 1 এর থেকে ছোট হতে পারে। পারত পক্ষে যেকোনো সমস্যা সমাধানের সময় ইনপুটে সবচেয়ে ছোট মান এবং বড় মান দিয়ে টেস্ট করে দেখা উচিত।

Timus 1025: সংখ্যাগুলিকে সর্ট করে নিয়ে চিন্তা করে দেখ।

Timus 1313: মনে কর তোমার row গুলি উপর হতে নিচে আর column গুলি বাম হতে ডানে 0 হতে $N-1$ পর্যন্ত নাম্বার করা। তাহলে নিচের বাম কোণা হতে উপরের ডান কোণা পর্যন্ত যে diagonal

যায় তাতে থাকা ঘরগুলির জন্য row এর id আর column এর id এর যোগফল কিন্তু একই হবে। আশা করি এই ট্রিকটা ইতোমধ্যেই তোমরা জানো। তাহলে একটু চিন্তা করে দেখো এই ট্রিক খাটিয়ে কীভাবে সহজে এই সমস্যা সমাধান করা যায়। দুটি নেস্টেড লুপ চালিয়েই কিন্তু সমস্যা সমাধান করে ফেলতে পারবে!

Timus 1079: n এর সবচেয়ে বড় মান দিয়ে টেস্ট করতে ভুলে যেও না। কে জানে overflow হয় কিনা!

Timus 1149: ফাংশন, লুপ ইত্যাদির ব্যবহার ঝালাই করে নেবার জন্য খুবই ভাল একটি সমস্যা। এখানে আসলে শিখানোর তেমন কিছু নেই। তুমি নিজে একটু ধীরে সুস্থে ঠাণ্ডা মাথায় চিন্তা করে কোড করে ফেল তাহলেই হবে।

অধ্যায় ৩

STL প্র্যাকটিস

STL এর পূর্ণরূপ হলো Standard Template Library. বহুল ব্যবহৃত কিছু অ্যালগরিদম (sort, binary search ইত্যাদি) এবং বহুল ব্যবহৃত কিছু data structure (queue, stack ইত্যাদি) এই STL এ আছে। STL না জানলেও তুমি এই অধ্যায়ের সমস্যাগুলি সমাধান করতে পারবে। কিন্তু যেই কাজ STL এ হয়তো এক দুই লাইনে হয়ে যাবে সে কাজ STL ছাড়া করতে হয়তো ১৫-২০ লাইন করে লাগবে। সুতরাং যেখানে STL ব্যবহার করা যায় সেখানে STL ব্যবহার করা উচিত। তবে STL অন্ধভাবে ব্যবহার করা উচিত না। এর মানে এই না যে তোমাকে জানতে হবে STL এর ভেতরে কীভাবে কোড করা আছে। যদি এমন কোনো সময় আসে যে তোমাকে STL ছাড়া সমস্যা সমাধান করতে হবে তাহলে যেন করতে পারো। এই জ্ঞান তোমাকে interview এর সময় বা কন্টেস্টের সময় দেখা যাবে বিভিন্ন ভাবে সাহায্য করছে। আর মাঝে মাঝে STL গুলো কীভাবে implement করা আছে তা জানলে ভাল হয়। তাহলে STL এর বিভিন্ন complexity সম্পর্কে ভাল ধারনা পাওয়া যায়। তোমাদের যাদের আগ্রহ আছে তারা STL সম্পর্কে আরও গভীরে পড়তে পারো নেটে খোঁজ করে। এই অধ্যায়ে মূলত আমরা কিছু সমস্যা দেখব যা STL দিয়ে সহজে সমাধান করা যায়। তোমরা চাইলে প্রথমে STL সম্পর্কে অনেক কিছু পড়তে পারো, সব STL শিখে ফেলতে পারো। কিন্তু তাতে করে সমস্যা হলো সেসব মনে থাকবে না। তবে তুমি যদি সল্ভ করতে করতে শেখ তাহলে সেই STL এর ব্যবহার মনে থাকে অনেক দিন এবং পরবর্তীতে যখন অন্য কোনো সমস্যায় দরকার পড়বে তুমি চট করে STL ব্যবহার করতে পারবে। আর এ অধ্যায়ের সমস্যাগুলি আসলে কীভাবে সমাধান করতে হবে সেটা বলা মুল উদ্দেশ্য না। মুল উদ্দেশ্য হলো কোন সমস্যায় কোন STL ব্যবহার করা যায় তা দেখানো এবং সেই STL নিয়ে কিছু আলোচনা করা। সুতরাং তোমরা চেষ্টা করবে এই অধ্যায়ের প্রতিটি সমস্যাকে সেখানে বলা STL ব্যবহার করে সমাধান করতে।

UVa 10474 Where is the Marble?

সমস্যা: তোমাকে N টি ($N \leq 10,000$) সংখ্যা দেয়া হবে। সংখ্যাগুলি সর্বোচ্চ $10,000$ হবে। প্রথমে তোমাকে এই সব সংখ্যা ছোট হতে বড় সর্ট করতে হবে। এরপর তোমাকে কিছু query দেওয়া হবে (প্রায় $10,000$ টি query)। প্রতিটি query তে একটি করে সংখ্যা q দেওয়া হবে। তোমাকে বলতে হবে তোমার সর্টেড সংখ্যার লিস্টে q আছে কি না, থাকলে প্রথম কোন স্থানে আছে। যেমন: 1, 3, 3, 5 এই সর্টেড নাম্বারগুলিতে 2 নেই। আবার প্রথম 3 আছে 2nd স্থানে।

সমাধান: ধরা যাক সংখ্যাগুলি A নামের একটি অ্যারেতে ইনপুট নেওয়া হয়েছে। প্রথমে আমরা এই সংখ্যাগুলিকে সর্ট করব। এজন্য আমরা STL এর sort ফাংশন ব্যবহার করব। আর এর জন্য আমাদের algorithm হেডার ফাইল include করতে হবে। সেই সাথে "using namespace std;" এই কথাটিও লিখতে হবে। সাধারণত সব হেডার ফাইল include করা শেষে এই লাইনটি আমি লিখে থাকি। কেন এই লাইনটি লিখতে হবে সেটা আরেকটা বিশাল আলোচনার জন্ম দিবে। তোমাদের আগ্রহ থাকলে namespace লিখে সার্চ দিও। আর stl এর যেকোনো হেডার ফাইলের জন্য তোমাদের এই

লাইন লিখতে হবে (একবার লিখলেই চলবে)। এখন কোডে তুমি যদি লিখ "sort(A, A + N)" তাহলে A[0] হতে A[N - 1] পর্যন্ত সর্ট হয়ে যাবে। যদি তোমাদের সংখ্যাগুলি A[1] হতে A[N] থাকে তাহলে তোমাদের লিখতে হবে "sort(A + 1, A + N + 1)"। মোট কথা তোমাদের যদি A[a] হতে A[b] পর্যন্ত sort করতে হয় তাহলে লিখতে হবে "sort(A + a, A + b + 1)". কেন এভাবে লিখবা তা জানতে হলে pointer নিয়ে একটু পড়াশুনা করতে হবে তোমাদের।

সর্ট তো করা হলো, কিন্তু কীভাবে বের করবে যে query এর সংখ্যা এই অ্যারেতে আছে কি না, থাকলে প্রথমে কোথায় আছে? এজন্য তোমরা চাইলে নিজে থেকে binary search এর কোড করতে পারো অথবা stl এর lower bound বা upper bound ব্যবহার করতে পারো। এদের ব্যবহার করতেও algorithm হেডার ফাইলের দরকার। Lower bound আর upper bound এর কোনটা যে কি দেয় তা আমার কখনই মনে থাকে না। যখন ব্যবহার করি তখন নেটে দেখে নেই। Lower bound ও upper bound দুই ক্ষেত্রেই আমাদের একটি সর্ট করা অ্যারে দিতে হয় আর একটি সংখ্যা দিতে হয়। Lower bound অ্যারে এর সেই স্থান দেয় যেখানের সংখ্যা query এর সংখ্যার সমান বা বড় এবং এদের মাঝে সবচেয়ে আগের স্থান দেয়। আর upper bound দেয় এমন স্থান যার সংখ্যা query এর সংখ্যা থেকে বড় এবং এদের মাঝে সবচেয়ে আগের স্থান দেয়। যেমন তোমার অ্যারে যদি হয় 1, 2, 2, 4, 5 তাহলে 2 এর lower bound যদি call করা হয় সেটি দিবে প্রথম 2 এর স্থান, আর যদি upper bound কে call করা হয় তাহলে দিবে 4 এর স্থান। এই lower bound বা upper bound এর ফাংশন কীভাবে call করতে হয়? "lower_bound(A, A + N, q)" যেখানে A অ্যারেতে 0 indexed ভাবে তুমি সঙ্খ্যাগুলি রেখেছ। একটা জিনিস, এইযে বললাম অমুকের স্থান দেবে - এই স্থানের মানে কি? Index? না, ওই স্থানের pointer. সুতরাং তোমরা যদি index চাও তাহলে এই lower bound এর return মান হতে A বিয়োগ করে নেবে। তাহলেই 0-indexed অ্যারে পজিশন পেয়ে যাবে। অর্থাৎ তোমাকে লিখতে হবে "lower_bound(A, A + N, q) - A". তাহলে তুমি index পেয়ে যাবে। কিন্তু যদি আমাদের অ্যারে 0 indexed না হয়ে 1 indexed হত তাহলে কীভাবে কল করতে? ঠিক sort ফাংশনে যেভাবে শিখেছি সেভাবে "lower_bound(A+1, A+N+1, q)".

এখন আশা যাক এদের time complexity তে। অনেকে মনে করে এটা তো binary search এর মত, সুতরাং এর time complexity হবে লগারিদম। কথাটা ঠিক আছে আবার ঠিক নেই। সত্যি কথা বলতে সব সময় এর complexity লগারিদম না। অ্যারে বা ভেক্টর বা এই জাতীয় random access করা সম্ভব এরকম data structure এর ক্ষেত্রে এরা লগারিদম (random access করা যায় মানে constant সময়ে যেকোনো index কে access করা যায় এরকম)। অন্য ক্ষেত্রে এরা লিনিয়ার ($O(n)$) যেমন list, set ইত্যাদি।[১] এসব ক্ষেত্রে linear সময় লাগে কারণ list বা set এসবের ক্ষেত্রে i তম index এ কে আছে এটা constant সময়ে বের করা সম্ভব না। তাই এসব ক্ষেত্রে সে একে একে সকল element দেখে এরপর lower bound বা upper bound এর মান নির্ণয় করে।

যাই হোক, তোমরা চাইলে আরও সহজে এই সমস্যা সমাধান করতে পারতে। সর্ট করার পর তুমি একটি অ্যারেতে লিখে রাখতে পারো যে, কোনো একটি সংখ্যা সর্টেড অ্যারেতে কোথায় প্রথম এসেছে। এই কাজ তো লিনিয়ার সময়েই করা যায়। মনে কর সর্ট করা অ্যারের শেষ হতে শুরুতে যাবে এবং লিখবে যে place[A[i]] = i তাহলেই হবে। কিন্তু যদি আমাদের ইনপুটের সংখ্যাগুলি 10,000 এর থেকে বড় হয়, অনেক বড়? চিন্তা কর- এক্ষেত্রে সমস্যা কি? সমস্যা হল আমরা এতো বড় সংখ্যাকে index হিসাবে ব্যবহার করতে পারি না বা করতে হলে যত বড় অ্যারের প্রয়োজন তত বড় অ্যারে আসলে নেওয়া সম্ভব না। এই অবস্থা থেকে মুক্তি দিতে পারে stl এর map. মনে কর তোমার কাছে কিছু নাম আছে আর তার বয়স আছে। তুমি M["hasan"] = 3 করলে hasan এর বিপক্ষে 3 থাকবে এই ম্যাপে। তুমি যদি পরে কখনও (যতবার) M["hasan"] কর 3 পাবে। ম্যাপ ব্যবহারের জন্য তোমাকে map নামের হেডার ব্যবহার করতে হবে। এটি declare করার syntax হলো "map<string, int> M;" অর্থাৎ যেই টাইপকে ম্যাপ করবা তা শুরুতে, আর যেই টাইপে ম্যাপ করবা তা শেষে। এখন তুমি একটি অ্যারের মত করে ব্যবহার করতে পারো একে। "M[?] = ?" করে তুমি map এ মান রাখতে পারো আবার "? = M[?]" করে ম্যাপ থেকে কোনো কিছুর মান

[১] Set এর ক্ষেত্রে এর নিজস্ব lower bound ও upper bound আছে। অর্থাৎ যদি "set<int> S" নামে একটি set তোমার কাছে থাকে তাহলে তোমরা "S.lower_bound(q);" এভাবে lower bound ব্যবহার করতে পারো এবং তা লগারিদমিক হবে। সাধারণ "lower_bound(S.begin(), S.end(), q);" করলে কিন্তু লিনিয়ার সময় লাগবে।

বের করতে পারো। ঠিক অ্যারের মত। সুতরাং ইনপুটের মান যদি অনেক বড় হয় সেক্ষেত্রে তোমাকে long long কে int এ ম্যাপ করতে হবে। Map এ কোনো কিছু রাখতে বা খুঁজে বের করতে সময় লাগে $\mathrm{O}(\log n)$ যেখানে n হল map এর সেই মুহূর্তের সাইজ। আর n টি জিনিস রাখতে $\mathrm{O}(n)$ memory লাগে।

UVa 101 The Blocks Problem

সমস্যা: 0 হতে n - 1 পর্যন্ত n টি ব্লক আছে যেখানে n সর্বোচ্চ 25 হতে পারে। এছাড়াও 0 হতে n - 1 পর্যন্ত n টি pile আছে। শুরুতে প্রতিটি ব্লক তার id এর সমান id এর pile এ থাকে। অর্থাৎ i তম ব্লক i তম pile এ থাকে। এরপর একে একে কিছু instruction দেওয়া হবে। Instruction গুলি 5 ধরনের হতে পারে। "move a onto b" বললে a ব্লককে b ব্লকের উপরে নিতে হবে। এজন্য তাদের উপরে যেসব ব্লক ছিল তাদেরকে তাদের শুরুর জায়গায় ফেরত দিতে হবে। শুরুর জায়গায় ফেরত দিতে হবে মানে তাদের শুরুর pile এ ফেরত দিতে হবে। একটা জিনিস খেয়াল কর এখানে কিন্তু বলে নায় যে কীভাবে ফেরত দিতে হবে, সেই জায়গায় কিছু থাকলে কি করতে হবে তা বলা নেই। চিন্তা করে দেখতে পার কেন বলা নাই, না বললে কোনো ক্ষতি আছে কিনা (আগে পুরো প্রবলেমটা তো পড়!)। আরেকটি instruction "move a over b" বললে a ব্লককে b ব্লক যেই pile এ আছে তার top এ রাখতে হবে। এসময় a ব্লকের উপরে যত ব্লক ছিল তাদেরকে তাদের শুরুর জায়গায় ফেরত দিতে হবে। "pile a onto b" বললে b এর উপরে থাকা সব ব্লককে তাদের শুরুর জায়গায় ফেরত দিতে হবে আর a ব্লক আর তার উপরে যত ব্লক আছে তাদেরকে b ব্লকের উপরে এনে রাখতে হবে। "pile a over b" বললে a ও তার উপরের সব ব্লককে b যেই pile এ আছে তার উপরে এনে রাখতে হবে। "quit" বললে আমাদের অপারেশন শেষ করে কোন পাইলে এখন কোন কোন ব্লক আছে তা নিচ হতে উপর এই ক্রমে প্রিন্ট করতে হবে।

সমাধান: Pile দেখেই চট করে stack ব্যবহার করতে বসে যেও না। কারণ আমাদের শুধু pile এর উপরের জিনিস না ভেতরের জিনিসও লাগবে। মনে কর a কোনো pile এর ভেতরে আছে। তাহলে তো আমাদের ভেতরের জিনিসগুলির উপর দিয়ে লুপ চালাতে হবে তাই না? হ্যাঁ, তোমরা চাইলে অবশ্য যতক্ষণ না a পাও ততক্ষণ ঐ stack থেকে তুলে আরেকটা কোনো জায়গায় রাখতে পারো। তবে আমি নিজে এই প্রবলেমের কোড করলে vector ব্যবহার করব। vector হলো এমন একটা অ্যারে যা প্রয়োজন মত লম্বা হয়। এটি ব্যবহারের জন্য তোমাকে আগে থেকে বলে দিতে হয় না যে তোমার কত সাইজের vector লাগবে (চাইলে বলতে পার)। তোমার যদি নতুন কোন সংখ্যা এর পেছনে ঢুকাতে হয় তাহলে বলবে "V.push_back(10);" তাহলেই এর পেছনে 10 ঢুকে যাবে। আবার চাইলে "pop _back" বলে পেছন থেকে সংখ্যা সরাতেও পারবে। এসব জিনিস constant সময়ে হবে বলে ধরে নিতে পারো অর্থাৎ কোন সময় লাগবে না এসব করতে। এছাড়াও vector এর আরেকটা সুবিধা হলো তুমি চাইলে "V[4]" বলে এর 4 index এর সংখ্যাকে access করতে পারবে। মোট কথা এটি একটি অ্যারে যা ব্যবহারের শুরুতে তোমাকে সাইজ বলে দিতে হয় না। vector এর push back, pop back ছাড়াও আরও অনেক ফাংশন আছে। এদের মাঝে বহুল ব্যবহৃতগুলি হলো size (vector এর বর্তমান সাইজ রিটার্ন করবে), clear (পুরো vector ধুয়ে মুছে পরিষ্কার করে দেবে) অন্যতম। সত্যি কথা বলতে এই চারটি ফাংশন ছাড়া আর কিছু আমার লাগে না। একটি কথা, vector এর জন্য erase, insert, find[১] ইত্যাদি কিছু ফাংশন আছে। তোমরা ভাবতে পারো যে "আহ কত সুন্দর একটি অ্যারের মাঝে আমরা insert, erase করতে পারি বা vector এর ভেতরে find করতে পারি।" না ঐ কাজ করতে যেও না। কারণ এতে insert বা erase বা find এর জন্য লিনিয়ার সময় লাগতে পারে worst case এ। কেবল মাত্র push back, pop back আর কোনো index এ $V[i]$ এর মত করে access করা constant সময়ের অপারেশন। আরও একটি কাজের ফাংশন আছে আর তাহলো resize. নাম শুনেই বুঝতে পারছ এ কি করে। 100 সাইজের একটি vector V কে যদি বল V.resize(50) তাহলে এর সাইজ 50 এ কমে যাবে। আবার উলটো ভাবে যদি 50 সাইজের অ্যারেকে বল V.resize(100) তাহলে আরও 50 টি স্থান যোগ হয়ে যাবে। নতুন জায়গাগুলিতে যতদূর সম্ভব garbage value থাকবে। যদি তুমি 0 দিয়ে তাদেরকে initialize করতে চাও তাহলে তোমাকে

[১] Find মনে হয় vector এর নিজস্ব ফাংশন না। এটি algorithm হেডার ফাইলের একটি ফাংশন যা sort/lower bound ইত্যাদির মত করে ব্যবহার করতে হয় অর্থাৎ find(V.begin(), V.end(), x) এরকম।

লিখতে হবে V.resize(100, 0). তোমাদের যদি n সাইজের একটি অ্যারে লাগে তাহলে n ইনপুট নেবার পর V.resize(n) করে n সাইজের অ্যারে বানিয়ে ফেলতে পারো V কে। এর পর সাধারণ ভাবে &V[i] বা V[i] দিয়ে ইনপুট আউটপুট বা হিসাব করতে পারো।

তাহলে তোমরা এই সমস্যা vector ব্যবহার করে সমাধান করে ফেলতে পার। যে কয়টি অপারেশন দেওয়া আছে সেগুলো implement করতে হলে আমাদের জানতে হবে কোন pile এ আমাদের সংখ্যা আছে। সেই কাজ তোমরা প্রতি pile এ গিয়ে তাতে লুপ চালিয়ে করে ফেলতে পারবে। আবার কোনো pile থেকে যদি উপরের সংখ্যা সরাতে চাও তাহলে pop back ফাংশন ব্যবহার করবে। কোনো pile এ নতুন সংখ্যা ঢুকাতে হলে push back ফাংশন ব্যবহার করতে পারবে। এরকম করে vector ব্যবহার করে খুব সহজেই এই সমস্যা সমাধান করা সম্ভব।

UVa 10815 Andy's First Dictionary

সমস্যা: তোমাকে একটি টেক্সট দেওয়া হবে। টেক্সটে দাড়ি কমা বা আরও অন্যান্য punctuation সহ capital বা small letter এ লিখা অনেক শব্দ থাকবে। তোমাকে এই টেক্সটে থাকা সকল ভিন্ন ভিন্ন শব্দ lexicographical অর্ডারে সাজিয়ে প্রিন্ট করতে হবে। সেই সাথে শব্দগুলিকে case insensitive হিসাবে কল্পনা করতে হবে। অর্থাৎ apple, AppLE, APPLE সবই এক। প্রিন্ট করার সময় তোমাকে শব্দগুলিকে lower case এ প্রিন্ট করতে হবে।

সমাধান: যদি তোমার কাছে একটি শব্দ থাকে তুমি তো খুব সহজেই তাকে lower case এ নিতে পারবে তাই না? যদি মনে না থাকে আবারো বলি, প্রতিটি character এর উপর লুপ চালিয়ে তুমি দেখো যে character টি upper case কি না ('A' <= word[i] && word[i] <= 'Z'). যদি হয় তাহলে তাকে lower case কর (word[i] = word[i] - 'A' + 'a'). এবার এই word কে stl এর set এ ঢুকিয়ে দাও। আমাদের গণিতের set এর মত stl এর set এও একই জিনিস একাধিকবার থাকে না। সেই সাথে তুমি যদি set এ লুপ চালাও তাহলে তুমি শব্দগুলিকে lexicographical এই সর্ট করা পাবা। আমাদের একটি সেট লাগবে যা string কে রাখবে। এর জন্য তোমার প্রয়োজন set নামের হেডার ফাইল। একে declare করতে হবে "set<string> S;" এই ভাবে। খেয়াল কর এখানে কিন্তু আমরা stl এর string ও ব্যবহার করছি (string হেডার ফাইল)। আমি ব্যক্তিগত ভাবে কখনও char* এর set ব্যবহার করি নাই তবে এটা বলতে পারি যে char* এর set ব্যবহার করলে আরও অনেক ঝামেলা করতে হতো (নিজে থেকে comparison function লিখে দিতে হতো)। মনে কর তোমার কাছে একটি শব্দ আছে "char word[20];" এ। তুমি একে আমাদের set এ ঢুকাতে চাও। এজন্য শুধু "S.insert(word);" লিখলেই চলবে। এখানে char এর অ্যারে নিজে থেকেই string এ convert হয়ে যাবে। সব শেষে set এর সকল শব্দ এর ভেতর দিয়ে লুপ চালানোর জন্য c++11 এর range based loop ব্যবহার করলেই চলে "for(string w : S)" এবার তুমি cout বা printf দিয়ে w কে print করে দিলেই হলো। cout দিয়ে প্রিন্ট করা সহজ তবে এতে সময় বেশি লাগতে পারে। আমি এজন্য printf ব্যবহার করি। তবে stl এর string কে printf দিয়ে প্রিন্ট করতে একটু কষ্ট করতে হয় printf("%s", w.c_str());. মূল সমস্যায় ফেরার আগে set এর কিছু দরকারি ফাংশন বলে নেই। size() - সেট এর সাইজ এবং এক্ষেত্রে unique count দেবে অর্থাৎ তুমি যদি একই জিনিস একাধিকবার insert করে থাকো তা একাধিকবার count হবে না, একবার count হবে। এছাড়াও empty() - সেট কি এখন ফাঁকা? count(x) - x কয়বার আছে, আসলে সর্বোচ্চ একবারই থাকা সম্ভব তাই এটি হয় 0 বা 1 রিটার্ন করবে। count এর complexity লগারিদমিক।

সবই তো বুঝলাম কিন্তু একটা টেক্সট থেকে কীভাবে শব্দ গুলি পাব? এক্ষেত্রে বিভিন্ন জনে বিভিন্ন উপায় অবলম্বন করে থাকে। আমি space ওয়ালা টেক্সট ইনপুটের ক্ষেত্রে সবসময় gets ব্যবহার করে থাকি। মনে কর একটি char line[100] অ্যারেতে পুরো লাইন ইনপুট নিয়েছি। এরপর আমি সকল non alphabet character কে space এ পরিবর্তন করে ফেলি (একটি লুপ চালালেই হয়)। সেই সাথে চাইলে upper case letter কেও lower case এ পরিবর্তন করে ফেলা যায় (তাহলে আর পরে upper case, lower case নিয়ে ঝামেলা করতে হয় না)। এবার stl এর istringstream ব্যবহার করি (sstream হেডার ফাইল)। একে declare করতে হয় এভাবে "istringstream iS(line);". এর মানে line অ্যারে হতে একটি istringstream (পূর্ণ রূপ input string stream) বানানো হলো, যার নাম iS. এবার আমরা এই লাইনের প্রতিটি শব্দকে আলাদা আলাদা করে চাই। খুবই সহজ-

while(iS >> w) যেখানে w হলো একটি stl এর string (string w). এবার এই শব্দকে stl এর set এ ঢুকিয়ে দিলেই হয়ে যাবে।

set ও istringstream এর যা ব্যবহার দেখালাম এর বাহিরে তেমন কোনো ব্যবহার নেই, অন্ততঃ আমার তেমন লাগে না (set এর size ফাংশন ছাড়া)। set এ মাঝে মাঝে erase ব্যবহার করতে হয়, কোনো একটি জিনিসকে set হতে মুছে ফেলার জন্য। এছাড়া lower bound বা upper bound এর কথা তো আগেই বলেছি। তুমি যদি নেটে কোনো একটি stl এর নাম লিখে সার্চ দাও তাহলে বহু উদাহরন পাবা।

Timus 1209 1, 10, 100, 1000...

সমস্যা: একটি sequence দেওয়া আছে: $110100100010000\ldots$. বলতে হবে k তম অংক কত যেখানে $1 \leq k \leq 2^{31} - 1$.

সমাধান: যারা গণিত পছন্দ কর তারা ফর্মুলা বের করে সমাধান করতে পারো। প্রথমে দেখ কোথায় কোথায় 1 আছে। 1, 2, 4, 7, 11 ইত্যাদিতে। চেষ্টা করে দেখ এদের কোনো ফর্মুলা আনতে পারো কিনা। অনেক উপায়েই ফর্মুলা বের করা যায়। যেমন একটা উপায় হতে পারে pattern খেয়াল করে: $1 = 1 + (0)$, $2 = 1 + (0 + 1)$, $4 = 1 + (0 + 1 + 2)$, $7 = 1 + (0 + 1 + 2 + 3)$. সুতরাং i তম সংখ্যা হবে $1 + (0 + 1 + \ldots i)$. ব্রাকেট এর ভেতরের অংশের ফর্মুলা তো জানোই! সুতরাং ইনপুট k এর জন্য আমাদের দেখতে হবে এমন কোনো i আছে কিনা। তা quadratic equation সমাধান করলেই সহজেই বের হয়ে যাবে।

ফর্মুলা বের না করেও করা যায়। এটা খেয়াল কর যে এই sequence এ খুব বেশি 1 নেই। পাশাপাশি দুটি 1 এর দূরত্ব linear হারে বাড়ছে। অর্থাৎ প্রথম দুটি 1 এর মাঝের দুরত্ব 0, দ্বিতীয় দুটি 1 এর মাঝের দুরত্ব 1, এর পরের দুরত্ব 2 এরকম করে 3, 4, 5... তাহলে 2^{31} দৈর্ঘ্যের মাঝে আসলে $\sqrt{2^{31}}$ এর বেশি 1 থাকবে না। কীভাবে বুঝলাম? সহজ, কিছুক্ষণ আগেই বলেছি আমাদের i তম 1 থাকবে i^2 এ (একদম ঠিক করে বললে $(i(i+1)/2+1)$ এ)। সুতরাং আমরা বলতে পারি 2^{31} এর ভেতরে মোট $\sqrt{2^{31}}$ টি 1 থাকবে। আমরা খুব সহজেই একটি লুপ চালিয়ে সব 1 এর স্থান বের করে ফেলতে পারব (মানে ফর্মুলায় না করে $0 + 1 + \ldots i$ এর কাজ লুপে করা) এবং তাদের একটি set এ রাখতে পারব। এরপর প্রতি ইনপুটে চেক করে দেখতে পারো যে set এ তোমাদের ইনপুটের সংখ্যা আছে কিনা। চাইলে তোমরা কষ্ট করে নিজে থেকে binary search ও করতে পারো। অথবা stl এর count, find, lower bound বা upper bound ও ব্যবহার করতে পারো।

UVa 156 Anagrams

সমস্যা: যদি দুটি শব্দে একই letter সমূহ একদম ভিন্ন অর্ডারে থাকে তাহলে তাদের anagram বলে। যেমন SPOT আর POTS. এই সমস্যার ক্ষেত্রে অক্ষর গুলি ভিন্ন অর্ডারে এবং ভিন্ন case এ থাকলেও আমরা তাদের anagram বলব। আবার যদি শব্দদুটিতে অক্ষরগুলি একই অর্ডারে থাকে কিন্তু তাদের কারো case যদি আলাদা হয় আমরা তাকেও anagram ধরব। যেমন spot আর spot কিন্তু anagram না, কিন্তু spot আর SpoT আবার anagram. একটি টেক্সটে যদি কোনো একটি শব্দের anagram না থাকে তাহলে তাকে ananagram বলে। আমাদেরকে একটি প্রদত্ত text এর সকল ananagram কে lexicographical অর্ডারে প্রিন্ট করতে হবে।

সমাধান: এই সমস্যার বেশির ভাগ আলোচনা UVa 10815 এর মত। শুধু একটি জিনিস নিয়ে কথা বলব। কীভাবে বুঝবে যে কোনো একটি শব্দের anagram এর আগে এসেছিল কিনা। প্রথমত চিন্তা কর দুটি শব্দ anagram কিনা তা কীভাবে বুঝবে। দুটি শব্দ anagram হয় যদি একই letter গুলি ভিন্ন অর্ডারে থেকে থাকে বা একই অর্ডারে থাকে কিন্তু কোনো একটি অক্ষরের case আলাদা হয়। এর মানে শব্দদুটিকে একই হওয়া যাবে না আর তাদের অক্ষরগুলিকে sort করলে একই শব্দ হবে (case insensitive comparison করতে হবে, অর্থাৎ দুটি string compare এর সময় case

এর ভিত্তিতে দুটি অক্ষরকে আলাদা মনে করা যাবে না)। অর্থাৎ S1 আর S2 একই হওয়া যাবে না (S1 != S2) আর এদেরেক সর্ট করার পর (sort(S1.begin(), S1.end()) ও S2 এর জন্যও একই রকম) ও যদি তাদের ভেতরে কোনো upper case letter থাকে তাদের small case এ পরিনত করার পর S1 ও S2 একই হবে (S1 == S2). এইযে একটি শব্দের সকল character কে সর্ট করেছ এবং তার ভেতরে যদি কোনো upper case letter থাকে তাকে lower case এ পরিনত করছ একে বলে একটি শব্দকে canonical form এ আনা। এতে লাভ কি? লাভ হলো- তোমরা তো মুল শব্দ দেখে সরাসরি বুঝতেছিলে না যে এই দুটি শব্দ anagram কিনা কিন্তু তাদের canonical form দেখে সহজেই বুঝতে পারবে তারা anagram কিনা। অবশ্য এই সমস্যার ক্ষেত্রে canonical form এর পাশাপাশি তোমাকে জানতে হবে মুল শব্দটি কি। কারণ দুটি শব্দ নিজেরাই যদি একই হয় তাহলে তো আর তারা anagram হবে না।

এটুকু যদি বুঝে থাকো তাহলে আমার মনে হয় set ব্যবহার করে সমস্যাটা সমাধান করে ফেলতে পারবে। তাও বলি। দুটি শব্দ anagram হওয়ার শর্ত দুটি। এক- তাদের একই হওয়া যাবে না, দুই- তাদের ভেতরে একই অক্ষরগুলি অন্য অর্ডারে বা অন্য case এ থাকতে পারে। আমরা একে একে এই দুটি শর্ত প্রয়োগ করব। প্রথমেই আমরা আমাদের ইনপুটে দেওয়া সব শব্দকে একটি সেটে ঢুকাব। তাহলে যদি একই রকম দুটি শব্দ থেকে থাকে আমরা তাদের পরিবর্তে একটি শব্দ পাব। আমাদের প্রথম শর্ত পুরন হয়ে গিয়েছে। এবার দ্বিতীয় শর্ত। আমাদেরকে প্রতিটি শব্দের জন্য তার canonical form বানাতে হবে। এখন এই একই canonical form ওয়ালা শব্দগুলিকে আমাদের বাদ দিতে হবে। কীভাবে? আমরা দুটি সেট নেব। একটি সেটে থাকবে সেসব canonical form যাদের anagram এখনও খুঁজে পাওয়া যায় নায়। অর্থাৎ ananagram শব্দগুলির canonical form এর সেট। আর আরেক সেটে থাকবে anagram শব্দগুলির canonical form. আমরা একে একে শব্দগুলি আর তাদের canonical form প্রসেস করব। প্রথমেই দেখ নতুন শব্দের canonical form কি ananagram এর সেটে আছে কিনা। যদি থাকে এর মানে এটি anagram. আমাদেরকে ananagram এর সেট হতে সেই canonical ফর্মকে মুছে ফেলতে হবে আর এই canonical form কে anagram এর সেটে insert করতে হবে। কিন্তু যদি এই শব্দ ananagram এর সেটে না থাকে তাহলে আমরা দেখব anagram এর সেটে আছে কিনা। যদি থাকে তার মানে এই শব্দও anagram. যেহেতু এই canonical form ইতিমধ্যেই anagram এর সেটে আছে সেহেতু আমাদের করার কিছু নেই। কিন্তু anagram এর সেটেও যদি না থাকে এর মানে এই শব্দটি আপাতত ananagram. সুতরাং এর canonical ফর্মকে আমরা ananagram এর সেটে insert করব। এই কাজ যদি আমরা সকল শব্দের জন্য করি তাহলে সব শব্দ প্রসেস শেষে আমরা সকল ananagram এর canonical form পেয়ে যাব। এবার শেষ কাজ- আমরা আবার আমাদের মূল শব্দ আর তার canonical form এর লিস্ট ধরে ধরে যাব। যদি দেখি তার canonical form টি ananagram এর সেটে আছে তার মানে এই শব্দটি ananagram. দাও প্রিন্ট করে। এরকম করে সকল ananagram প্রিন্ট করতে হবে। খেয়াল রেখ, আমাদের প্রিন্ট করা শব্দগুলি কিন্তু lexicographcal order এ প্রিন্ট করতে হবে। কীভাবে করবে? যেই শব্দগুলি প্রিন্ট করবে তাদের একটি set এ ঢুকিয়ে তাদের প্রিন্ট করতে পার, বা একটি vector এ ঢুকিয়ে তাদের সর্ট করে এরপর প্রিন্ট করতে পার। অথবা এইযে শেষ লুপে আমরা যখন মুল শব্দ আর তার anagram এর উপর লুপ চালাচ্ছিলাম তখনই যদি মূল শব্দের lexicographical অর্ডারে লুপ চালাই তাহলে আর কষ্ট করে আরেকটি set বা vector নিতে হবে না। আসলে আমরা তো একটি set এ আমাদের সকল শব্দকে ঢুকিয়েছিলাম (আমাদের সমাধানের প্রথম সেট) ঐ সেটের উপরেই সবসময় কাজ করলেই হবে, আমাদের আর অতিরিক্ত set বা vector নিতে হবে না।

UVa 12096 The SetStack Computer

সমস্যা: মনে কর একটি সেট এর stack আছে। একে একে তোমাকে সর্বোচ্চ 2000 টি অপারেশন বলবে। Push বললে তোমাকে একটি ফাকা সেট {} পুশ করতে হবে। Dup বললে তোমাকে stack এর উপরের যেই সেটটি আছে তা আবারো পুশ করতে হবে। Union বা Intersect বললে stack হতে উপরের দুটি element কে pop করে তাদের union বা intersection কে পুশ করতে হবে। Add বললে একই ভাবে stack এর উপরের দুটি element নিয়ে প্রথমটিকে (stack এ যে সবচেয়ে উপরে

ছিল) দ্বিতীয়টির ভেতরে ঢুকিয়ে দিতে হবে। যেমন মনে কর $A = \{\{\}, \{\{\}\}\}$ stack এর উপরে আছে আর $B = \{\{\}, \{\{\{\}\}\}\}$ stack এ উপর থেকে দ্বিতীয় তে আছে। তাহলে এদের union হবে $\{\{\}, \{\{\}\}, \{\{\{\}\}\}\}$, intersection হবে $\{\{\}\}$ আর add হবে $\{\{\}, \{\{\{\}\}\}, \{\{\}, \{\{\}\}\}\}$. প্রতিটি অপারেশনের পর তোমাকে stack এর উপরে থাকা সেটের সাইজ প্রিন্ট করতে হবে।

সমাধান: ও বাবা! সেটের সেট! তবে এই দেখে ভয় পেয়ে গেলে হবে না। আমাদের সমাধান করতে হবে। আমরা যা করতে পারি তাহলো প্রতিটি সেটকে একটি করে id দিতে পারি। ফলে একটি সেটের ভেতরে যদি কয়েক ধাপ nested সেটও থাকে আমরা আসলে তার id নিয়ে কাজ করব, সেই nested সেট নিয়ে না। অর্থাৎ প্রতিটি সেট এখন আসলে একটি id, আবার কোনো একটি সেট বা id হলো অন্য কিছু id এর সেট বা set<int>. stack এর উপরের সেটের সাইজই হলো আমাদের প্রতি অপারেশনের পর উত্তর। যখনই নতুন কোনো সেট তৈরি হবে আমরা map এ দেখে নেব যে এই সেট আগেই এসেছিল কিনা, আর না আসলে তাকে একটি নতুন id দিব। অর্থাৎ আমাদের map হবে map<set<int>, int> অর্থাৎ এটি একটি int এর সেট কে int এ map করে। এছাড়াও আমাদের আরেকটি reverse map এর দরকার হবে অর্থাৎ কোন id তে কোন সেট। তবে সেজন্য তোমরা চাইলে map না ব্যবহার করে একটি vector ব্যবহার করতে পারো (vector<set<int>>). আর সব শেষে আমাদের একটি stack লাগবে। চাইলে তোমরা সেটের stack নিতে পারো, আবার চাইলে id এর stack নিতে পারো। id এর সেট নেওয়া মনে হয় সুবিধা জনক।

মনে কর আমরা একটি id এর stack নিলাম। Push বললে আমরা একটি ফাকা সেট নিয়ে দেখব map এ ইতোমধ্যেই এরকম কোনো সেট আছে কিনা, না থাকলে একে একটি নতুন id দিয়ে map এ রাখব এবং সেই id আমাদের stack এ push করব। Dup বললে আমরা stack এর উপরের element নিয়ে তাকে আবার push করব। Union বা Intersect বললে আমরা stack হতে দুটি id নিয়ে আমাদের vector হতে দেখব এরা কোন কোন set. এর পর সেই দুটি সেটের মাঝে union বা intersect করে পাওয়া নতুন সেটকে map, stack ও vector এ রাখব (এমনও হতে পারে যে map এ ইতোমধ্যেই আছে, সেক্ষেত্রে map বা vector এ রাখার দরকার নেই)। সবশেষে Add এর ক্ষেত্রে আমরা একটি id কে ওপর সেটে ঢুকিয়ে আগের মত map, stack ও vector কে আপডেট করব। এখানে একটা জিনিস খেয়াল রেখ, vector এ থাকা আগের সেটকে যেন পরিবর্তন করে বস না। id কে সেটে insert করার আগে তোমাকে সেই সেট copy করে নিতে হবে। এসব ছোট খাট ভুল হতেই পারে। তোমাদেরকে debug করে করে বের করতে হবে এসব ভুল।

আমরা মোটামোটি map ও vector এর ব্যবহার ইতোমধ্যেই দেখে ফেলেছি। এবার দেখা যাক stack এর ব্যবহার। আমি ধরে নিচ্ছি তোমরা stack নামক ডেটা স্ট্রাকচার ইতোমধ্যেই জানো। এটি ব্যবহারের জন্য আমাদের দরকার stack নামক হেডার ফাইল। একটি int এর stack কে declare করতে হয় stack<int> S এরকম করে। এর দরকারি ফাংশন গুলি হলো push(x) - তাহলে x পুশ হয়ে যাবে। pop() - তাহলে উপরের element মুছে যাবে। খেয়াল কর pop কিন্তু কিছু return করে না, কেবল উপরের element কে ফেলে দেয়। top() - stack এর উপরের element কে return করবে তবে এটি মুছে যাবে না। empty() - এটি true দেবে যদি তোমার stack ফাঁকা হয়। size() - এটি stack এর সাইজ বলবে।

UVa 540 Team Queue

সমস্যা: অনেকগুলি মানুষ আছে এবং বলা আছে কে কোন দলে। এখন একে একে মানুষ আসছে। কে আসছে তা বলা আছে। এক জন এসে যদি দেখে তার দলের কেউ আগে থেকেই লাইনে আছে তাহলে সে তাদের পেছনে গিয়ে দাঁড়াবে। যদি না থাকে তাহলে সে সবার শেষে দাড়িয়ে যাবে। যখন লাইন থেকে কেউ বের হবে তখন লাইনের শুরু থেকে বের হবে। বলতে হবে লাইন থেকে কে কে কি অর্ডারে বের হবে। মোট 200,000 এর মত instruction থাকতে পারে।

সমাধান: Stl এর queue ব্যবহার করলে বেশ সহজেই সমাধান হয়ে যাবে। দুই ধরনের queue রাখতে হবে। একটি queue তে থাকবে লাইনে কোন কোন দলের মানুষ কি অর্ডারে আছে (ধরা যাক এটি বিশেষ queue), আর প্রতিটি দলের জন্য আলাদা আলাদা queue. যখন একজন মানুষ আসবে তখন তার দলের queue দেখো। যদি সেটি ফাকা নাহয় তাহলে সেই queue তে তাকে ঢুকিয়ে

দাও। যদি ফাকা হয় তাহলে তাকে সেই queue তে ঢুকিয়ে দাও এবং সেই সাথে তার দলকে (মানে তার দলের id কে) বিশেষ queue তে ঢুকিয়ে দাও। বের করার সময় দেখো বিশেষ queue এর শুরুতে কোন দল আছে, সেই দলের queue এর সামনের জনকে বের করে দাও। যদি এর ফলে সেই queue ফাঁকা হয়ে যায় তাহলে তার দলকে বিশেষ queue হতে বের করে ফেল। তাহলেই এই সমস্যা সমাধান হয়ে যাবে।

Stl এর queue ব্যবহারের জন্য আমাদের queue হেডার ফাইল লাগবে। int এর queue কে declare করতে হয় queue<int> Q এর মত করে। এর দরকারি ফাংশনগুলি হলো- push(x) - x পুশ হয়ে যাবে, pop() - পপ হবে কিন্তু এটি কোনো কিছু রিটার্ন করবে না, front() - queue এর সামনের জিনিস রিটার্ন করবে তবে পপ হবে না, size(), empty() ইত্যাদি অন্যান্য stl এর মতই কাজ করবে।

UVa 136 Ugly Numbers

সমস্যা: যেসকল সংখ্যার প্রাইম ফ্যাক্টরগুলি কেবলমাত্র 2, 3 বা 5 তাদের ugly সংখ্যা বলে। যেমন 1, 2, 3, 4, 5, 6, 8, 9 ইত্যাদি। তোমাকে 1500 তম ugly সংখ্যা বের করতে হবে।

সমাধান: এই সমস্যা সমাধানের নানা উপায় আছে। এর মাঝে একটি উপায় তোমাদের বলা যাক। মনে কর একটি সেটে প্রথম কিছু ugly সংখ্যা আছে। আমরা এদের মাঝে সবচেয়ে ছোটটি নিব (এবং ঐ জায়গা থেকে ঐ সংখ্যা মুছে দেব), এটি হবে আমাদের প্রথম ugly সংখ্যা। এর পর একে 2, 3 ও 5 দিয়ে গুন করে ঐ সেটে এই নতুন সংখ্যাগুলি রাখব। এরপর আবার সবচেয়ে ছোট সংখ্যাটি নিব, এটি হবে আমাদের দ্বিতীয় ugly সংখ্যা। আবার আমরা তাকে 2, 3 ও 5 দিয়ে গুন করে এই গুনফলগুলিকে সেটে রেখে দেব। এভাবে চলতে থাকবে 1500 বার। একটু চিন্তা করলে বুঝবে প্রথমে সেটে শুধু 1 থাকলেই চলবে। সুতরাং আমাদেরকে সেট হতে প্রতিবার minimum সংখ্যা বের করতে হবে। stl এর set এ এই কাজ সম্ভব বা তোমরা চাইলে priority queue ব্যবহার করতে পারো। Priority queue ব্যবহারের সমস্যা হল এখানে একই সংখ্যা একাধিক বার ঢুকালে একাধিক বার ঐ সংখ্যা থাকবে। তাহলে কিন্তু আমাদের এই সমস্যায় দুইবার 6 চলে আসবে (একবার 2×3 আরেকবার 3×2).[১] সুতরাং ভাল হয় যদি আমরা এখানে set ব্যবহার করি। set সম্পর্কে মনে হয় ইতোমধ্যেই অনেক কিছু বলেছি। কিন্তু একটা জিনিস বলি নাই কীভাবে এ থেকে সবচেয়ে ছোট সংখ্যা পাবে। সহজ "*S.begin()" (খেয়াল কর শুরুতে একটি star আছে)। সেটের সবচেয়ে ছোট সংখ্যা পেয়ে গেলে এর পরের কাজ হল সেট থেকে সেই সংখ্যা মুছে ফেলা। x কে মুছে ফেলতে চাইলে আমরা লিখতে পারি "S.erase(x)" এবং এজন্য লগারিদমিক সময়ের প্রয়োজন। বাকিটুকু আমার মনে হয় তোমরা করে ফেলতে পারবে।

UVa 1592 Database

সমস্যা: একটি টেবিল দেওয়া আছে। টেবিলের সর্বোচ্চ 10,000 টি row এবং 10 টি column থাকতে পারে। টেবিলের প্রতিটি সেলে কিছু text আছে। প্রতিটি row তে একটি column এর text তার পাশের column এর text হতে একটি কমা দ্বারা পৃথক থাকবে। বলতে হবে এমন দুটি row (r_1, r_2) ও দুটি column (c_1, c_2) আছে কিনা যেন $(r_1, c_1) = (r_1, c_2)$ ও $(r_2, c_1) = (r_2, c_2)$ হয়। অর্থাৎ ঐ সেলগুলির শব্দ একই হয় কিনা।

সমাধান: লিমিট অনুসারে তুমি চাইলেই r_1, r_2, c_1, c_2 এর চারটি লুপ চালাতে পারবে না। যেহেতু কলামের লিমিট বেশ কম তাহলে কলামের দুটি লুপ চালায়ে ফেলো। এতে করে আমাদের c_1, c_2 নির্দিষ্ট হয়ে যাবে। এখন কথা হলো এমন দুটি r_1, r_2 পাওয়া যাবে কিনা যা আমাদের condition পূর্ণ করে। আমরা যা করতে পারি তাহলো প্রতিটি row তে গিয়ে আমাদের নির্বাচিত দুটি কলামের string নিয়ে তাদেরকে জোড়া আকারে set বা map এ রাখতে পারি। যদি সব row দেখার পর দেখ একই

[১] যাদের এই কথা বুঝতে সমস্যা হচ্ছে- অর্থাৎ কেন দুইবার 6 আসবে তারা হাতে হাতে simulate করে দেখ।

জোড়া একাধিক বার এসেছে তার মানে এরকম r_1, r_2 পাওয়া যাবে। যেহেতু আমাদের শুধু হ্যাঁ না বললেই চলবে না, r_1, r_2 ও দিতে হবে, সেহেতু আসলে আমাদের set দিয়ে কাজ চলবে না। একটি map নিতে হবে আর লিখে রাখতে হবে যে এই string জোড়া অমুক row তে এসেছিল। যখন আমরা নতুন row তে এসে নতুন string জোড়া পাবো তখন আমরা map এ দেখবো এর আগে এই জোড়া এসেছিল কিনা বা আসলেও কোথায়। এভাবেই আমরা আমাদের দরকারি r_1, r_2 পেয়ে যাব।

এখন কোডে আসা যাক। প্রথম প্রশ্ন আমরা string জোড়া বানাব কীভাবে? তোমরা চাইলে একটি vector নিয়ে তাতে string গুলি রাখতে পারো। অথবা চাইলে একটি structure কে declare করে তাতে আমাদের দুটি string রাখতে পারো। এর পর সেই structure কে map করতে পারো। তবে সেক্ষেত্রে সমস্যা হলো তোমাদের সেই structure এর comparison function লিখতে হবে। এজন্য তোমাদের < operator টি overload করতে হবে। এ বিষয়ে তোমরা নেটে অথবা "প্রোগ্রামিং প্রতিযোগিতা" বইয়ে দেখে নিতে পারো। চাইলে অপারেটর ওভারলোড না করে functor ও declare করতে পারো। তুমি যদি vector ব্যবহার করতে তাহলে কোনো operator overload এর দরকার হত না, কারণ string এবং vector দুজনেরই < operator ভাল মতই define করা আছে।

তবে এই সমস্যার ক্ষেত্রে আমার কাছে সুবিধা জনক উপায় মনে হয় string এর pair নেওয়া। Pair হলো stl এর একটি জিনিস। একে আসলে দুই member variable ওয়ালা একটি structure কল্পনা করতে পারো। এটির জন্য দরকার utility নামক হেডার ফাইল। এটি declare এর উপায় হলো pair<string, string>. যদি একটি variable এ তুমি একটি pair রাখতে চাও তাহলে তুমি এরকম লিখতে পারো "pair<string, string> P = make_pair(A, B);" যেখানে A ও B দুটি string. কোনো একটি pair এর প্রথম element কে ব্যবহার করার জন্য লিখতে হয় "P.first" আর দ্বিতীয়টিকে ব্যবহারের জন্য "P.second". Pair এর সুবিধা হলো এই ক্ষেত্রে তুমি যদি pair এর vector বা array নিয়ে তাদের sort করতে চাও তাহলে কোনো অতিরিক্ত comparison ফাংশন লিখতে হবে না। তবে এর সমস্যাও আছে। প্রায়ই দেখা যায় যে কিছু দুর কোড করার পর মনে হয় আহা আরেকটা variable থাকলে মনে হয় ভাল হয়। সেক্ষেত্রে pair কে extend করা সম্ভব হয় না। তখন দেখা যায় আমাদের নতুন করে structure বানাতে হয়। এর ফলে অনেক কোড পরিবর্তন হয়। এখন অবশ্য tuple আছে। তোমরা চাইলে tuple ব্যবহার করতে পারো। আমি খুব একটা ব্যবহার করি নাই এটি।

তাহলে string জোড়া represent করার ঝামেলা তো গেল। কিন্তু আমাদের ইনপুট তো সুন্দর করে কলাম গুলো আলাদা করা থাকবে না। সেলের মানগুলি কমা দ্বারা পৃথক থাকবে। তাদের আলাদা করা যায় কীভাবে? বুদ্ধি হলো string এর push back ফাংশন ব্যবহার করা। gets দিয়ে ইনপুট নিয়ে (কারণ লাইনে space থাকতে পারে) এর পর কমা না পাওয়া পর্যন্ত push back ফাংশন ব্যবহার করে string বানাতে থাকো। কমা পেলে বা লাইন শেষ হয়ে গেলে এই string কে ধরে একটি vector এ ঢুকিয়ে দাও। তাহলে প্রসেস শেষে একটি row এর প্রতিটি column এর মান আলাদা আলাদা করে vector এ পেয়ে যাবে।

UVa 12100 Printer Queue

সমস্যা: একটি queue তে অনেকগুলি কাজ আছে। প্রতিটি কাজের একটি priority আছে। তুমি queue হতে প্রথম কাজ নিবে যদি দেখো এর থেকেও বেশি priority ওয়ালা কাজ queue তে আছে তাহলে একে সবার পেছনে রাখবে। আর যদি এরকম কাজ না থাকে তাহলে তুমি এই কাজ করবে। Queue তে একটি কাজ mark করা আছে। বলতে হবে সেই কাজটি কখন হবে। যেকোনো কাজ করতে 1 মিনিট সময় লাগে, আর queue হতে কোনো কাজ নিতে বা রাখতে কোনো সময় লাগে না। মোট কাজ মাত্র 100 টি হতে পারে।

সমাধান: আমাদের জানতে হবে যে বর্তমানে সবচেয়ে বেশি কত priority এর কাজ আছে। এ জন্য তোমরা একটি priority queue ব্যবহার করতে পারো। একটি কাজ নিয়ে দেখবে এর priority আর priority queue এর সবচেয়ে বড় সংখ্যা সমান কিনা। সমান হলে এই কাজটি করবে, এবং করা শেষে queue ও priority queue দু জায়গা হতেই এটি ফেলে দেবে। আর না হয়ে থাকলে queue এর শেষে এই কাজ রেখে দেবে। যেহেতু মাত্র 100 টি কাজ সেহেতু আমাদের এই $O(n^2 + n \log n)$

এর অ্যালগরিদমে কোনো সমস্যা হবে না। $O(n^2)$ কারণ- হয়তো প্রতিবার সব কাজ দেখার পর সব শেষেরটা আমরা নেব। তাহলে $O(n)$ বার $O(n)$ সাইজের অ্যারে বা queue আমাদের ঘেঁটে দেখতে হচ্ছে। আর প্রতিবার priority queue থেকে একটি সংখ্যা ফেলতে সময় লাগে $O(\log n)$. সুতরাং n টি জিনিস ফেলতে সময় লাগবে $O(n \log n)$. priority queue এর উপরের সংখ্যা কত এই প্রশ্নের উত্তর পেতে আসলে constant সময় লাগে।

একটি জিনিস, priority queue ব্যবহারের জন্য তোমাদের লাগবে queue নামের হেডার ফাইল। এছাড়া সব কাজ queue এর মত, শুধু মাত্র এর front() ফাংশন না থেকে থাকবে top() নামক ফাংশন। এটি সবসময় এর ভেতরে থাকা সর্বোচ্চ মানটি top এ রাখে। এই সমস্যার ক্ষেত্রে কিন্তু set ব্যবহার করতে যেয়ো না। কারণ set এ একটি সংখ্যা একাধিক বার insert করলেও সেটি মাত্র একবার থাকে। তোমরা চাইলে set এর পাশাপাশি একটি map নিয়ে তাতে সংখ্যাগুলির count রাখতে পারো কিন্তু তাতে মনে হয় ঝামেলা বেশি হয়ে যায়। এর থেকে priority queue ই ভাল।

৩.১ অনুশীলনী

৩.১.১ সমস্যা

Simple

- UVa 1593 Alignment of Code
- UVa 10935 Throwing cards away I
- UVa 1595 Symmetry
- UVa 1596 Bug Hunt
- UVa 12504 Updating a Dictionary
- UVa 400 Unix Is
- UVa 1594 Ducci Sequence
- UVa 10763 Foreign Exchange
- UVa 230 Borrowers
- UVa 822 Queue and A
- UVa 511 Do You Know the Way to San Jose?

Easy

- UVa 1598 Exchange
- UVa 212 Use of Hospital Facilities
- UVa 814 The Letter Carrier's Rounds
- UVa 1597 Searching the Web
- UVa 12333 Revenge of Fibonacci
- UVa 207 PGA Tour Prize Money
- UVa 221 Urban Elevations

৩.১.২ হিন্ট

UVa 1594: কোনো একটি sequence এর আগে এসেছিল কিনা এটা বুঝার জন্য তোমরা চাইলে vector এর map রাখতে পারো। অথবা তোমরা ধর 200 বা 500 বার চেষ্টা করবা যে শূন্য পেয়ে যাও কিনা, পেলে তো হয়েই গেল। আর না হলে বলে দাও যে হবে না। চাইলে বসে বসে প্রমাণ করতে পারো কেন 200 বা 500 যথেষ্ট!

UVa 10763: এই সমস্যা সমাধানে তোমরা pair এর দুটি vector নিতে পারো। একটিতে সোজা pair রাখবে আরেকটিতে উলটো pair (a,b এবং b,a). এখন দুটি vector কে sort করে তাদের == দিয়ে compare করে দেখো। তাহলেই হয়ে যাবে।

UVa 207: তোমরা তো %s দিয়ে ইনপুট আউটপুট করেছ, কিন্তু %20s বা %9.2lf বা %-20s এরকম ব্যবহার করেছ? না জানলে একটু নেটে দেখে জেনে নাও। এই সমস্যা stl এর অধ্যায়ে রাখার কারণ হলো sort এর জন্য comparison ফাংশন লিখতে শেখা।

UVa 814: এই সমস্যায় একটি জিনিস শিখলে মনে হয় ভাল হয়। তাহল string এর find ফাংশন। একে তুমি যদি কোনো একটি character দাও তাহলে তোমাকে বলে দেবে প্রথমে সেই character

কোথায় আছে। এছাড়াও এর আরেকটি দরকারি ফাংশন আছে, তাহলো substr(x, y)- এটি x হতে y - 1 পর্যন্ত substring দেয়। আশা করি এই দুটি ফাংশন ব্যবহার করে তোমরা email address কে দুই অংশে ভেঙ্গে ফেলতে পারবে। না পারলেও সমস্যা নাই, শেষ আশ্রয় for loop তো আছেই।

অধ্যায় ৪

Mathematics সম্পর্কিত সমস্যা

UVa 11582 Colossal Fibonacci Numbers!

সমস্যা: Fibonacci সংখ্যার সংজ্ঞা তো জানাই আছে তোমাদের? $f(0) = 0, f(1) = 1, f(n) = f(n-1) + f(n-2)$. তোমাদের a, b, n $(0 \leq a, b < 2^{64}, 1 \leq n \leq 1000)$ দেওয়া থাকবে। বলতে হবে $f(a^b) \bmod n$ এর মান কত।

সমাধান: তোমরা দুটি উপায়ে আগাতে পারো। একটি হলো বিভিন্ন n এর জন্য প্রথম ৫০ টি $f(i) \bmod n$ প্রিন্ট করে দেখা আর আশা করা যদি কোনো হিন্ট পেয়ে যাও। আর আরেকটা উপায় হলো theoretically চিন্তা করা। যারা অভিজ্ঞ কন্টেস্টেন্ট তারা এই সমস্যা দেখে প্রথমে যা ভাববে তাহল a^b অনেক অনেক বড় মান। সুতরাং হয় আমাদের matrix exponent এর মত কিছু করতে হবে নাহলে আমাদের sequence কে সাইকেলে ফেলতে হবে। যদি কেউ খুব recent সময়ে matrix exponent টেকনিক শিখে থাকো তাহলে তো কোনো কথাই নেই, সে সানন্দে এটাকে সেই ভাবে সমাধান করতে লেগে যাবে। কিন্তু একটা জিনিস, তোমাকে বের করতে হবে a^b তম fibonacci সংখ্যা আর a^b কিন্তু অনেক অনেক বড় সংখ্যা। Matrix exponent পদ্ধতিতে i তম fibonacci সংখ্যা বের করতে সময় লাগে মোটামোটি $\log(i)$, তাহলে a^b তম fibonacci সংখ্যা বের করতে লাগবে $\log(a^b) = b\log(a)$ সময়। যা অনেক অনেক বেশি হয়ে যায়। সুতরাং matrix exponent দিয়ে হবে না। তাহলে দেখা যাক এই sequence কে সাইকেলে ফেলা যায় কিনা। n = 6 এর জন্য $f(i) \bmod n$ এর মানগুলি দেখা যাক- 0, 1, 1, 2, 3, 5, 2, 1, 3, 4, 1, 5, 0, 5, 5, 4, 3, 1, 4, 5, 3, 2, 5, 1, 0, 1. আমরা কিন্তু আমাদের সাইকেল পেয়ে গেছি, কারণ আমরা পর পর 0 আর 1 পেয়েছি, অর্থাৎ আমাদের সাইকেলটা আবার শুরু থেকে শুরু হয়ে গেছে। এ অবস্থায় আমাদের মনে কয়েকটা প্রশ্নের উদয় হতে পারে। প্রথমত, সবসময় সাইকেলে পড়বে কি না। দ্বিতীয়ত, সাইকেলে পরলে কি এটা শুরু থেকে সাইকেলে পড়বে নাকি মাঝা মাঝি কোথাও থেকে। তৃতীয়ত, সাইকেলের length কিরকম হবে। আমারা একে একে এসব প্রশ্নের উত্তর পাবার চেষ্টা করি।

প্রথম প্রশ্ন- সবসময় সাইকেলে পড়বে কিনা। হ্যাঁ পড়বে। কেন? কারণ আমরা সবসময় $\bmod n$ নিচ্ছি, সুতরাং আমাদের সংখ্যাগুলি সবসময় 0 হতে $n-1$ পর্যন্ত হতে পারে। তুমি যদি পাশাপাশি দুটি সংখ্যা নাও তারা মোট n^2 রকম হতে পারে। অর্থাৎ n^2+1 length পর্যন্ত যাবার পর যেই pair পাবা সেটা অবশ্যই এর আগে এসেছে কারণ n^2 এর বেশি রকমের pair হওয়া সম্ভব না। সুতরাং আমরা সবসময় সাইকেল পাব। দ্বিতীয় প্রশ্ন- এই সাইকেলটি কি শুরুতে পড়বে কিনা অর্থাৎ $0, 1 \ldots, 0, 1, \ldots$ এরকম হবে কিনা। হ্যাঁ। কেন? ধরে নাও যে এটি 0, 1 এ পরে না, এটি মাঝের কোথাও a, b তে পরে অর্থাৎ $f(i) \bmod n$ হবে $0, 1 \ldots, a, b, \ldots, a, b, \ldots$. এখন দ্বিতীয় a, b এর কথা চিন্তা কর। এর আগের সংখ্যা কত? b - a. কিন্তু প্রথম a, b এর আগের সংখ্যাও b - a. অর্থাৎ আমাদের সাইকেলটা আসলে a, b তে গিয়ে পরে না, তার আগে গিয়ে পরে। কিন্তু এভাবে তো তুমি একঘর একঘর করে পিছাতে থাকতেই পারো তাই না? এর মানে আসলে সাইকেলটা 0, 1 থেকেই হয়। এখন তৃতীয় প্রশ্ন- সাইকেলর length কত। প্রথম প্রশ্নের উত্তর থেকে দেখেছ যে n^2 এর

মাঝে সাইকেলে পড়বেই। কিন্তু আসলে practically এর অনেক আগেই সাইকেলে পরে। এমনকি n এর মান 1000 এর মাঝে 3000 ধাপের মাঝেই সাইকেলে পরে যায়। কি ভাবে বের করলাম? সহজ, একটা কোড করেছি 7-8 লাইনের।

তাহলে আর কি। প্রতিটি n এর জন্য তোমরা পুরো সাইকেল বের করে আলাদা আলাদা vector এ রেখে দাও (vector এর অ্যারে)। এবার ইনপুট পেলে দেখো এই n এর জন্য সাইকেলের length কত। ধরা যাক c. তাহলে $a^b \bmod c$ বের করতে হবে। এটাতো bigmod ছাড়া কিছুই না। শুধু খেয়াল রাখবা overflow যেন না হয়। তাহলে আমরা $f(a^b) \bmod n$ এর মান পেয়ে যাব যা vector এর $a^b \bmod c$ তম সংখ্যা।

UVa 12169 Disgruntled Judge

সমস্যা: একটা দুষ্টু জাজ মনে মনে চারটি সংখ্যা x_1, a, b, T নিলো ($0 \leq x_1, a, b \leq 10000$). এরপর সে $x_i = (ax_{i-1}+b) \bmod 10001$ ব্যবহার করে $x_2, x_3, \ldots x_{2T}$ এর মান বের করল। এরপর সে তোমাকে $x_1, x_3, x_5, \ldots x_{2T-1}$ এর মান গুলি দিল আর তোমাকে বলল যে $x_2, x_4, x_6, \ldots x_{2T}$ এর মান প্রিন্ট করতে। যদি একাধিক উত্তর থাকে তাহলে যেকোনো একটি দিলেই চলবে। একাধিক উত্তর মানে এমনও তো হতে পারে যে একাধিক a, b এর জন্য একই $x_1, x_3, x_5, \ldots x_{2T-1}$ কিন্তু ভিন্ন $x_2, x_4, x_6, \ldots x_{2T}$ পাওয়া যায়। সেক্ষেত্রে যেকোনো একটি দিলেই চলবে। T এর মান সর্বোচ্চ 100 হতে পারে।

সমাধান: আমরা যদি a ও b এর উপর লুপ চালাই এবং $2T$ টি সংখ্যা বের করে দেখি যে তারা সমান কিনা তাহলে একটু বেশি সময় লেগে যায় $a \times b \times T = 10^{10}$. তাহলে কীভাবে আগানো যায়? তুমি দুই ভাবে চিন্তা করতে পারো। এক- a ও b দুইটি লুপ না চালিয়ে একটি লুপ চালানো যায় কিনা। দুই- আমাদের x_1 ও x_3 দেওয়া আছে। এখান থেকে কোনো ভাবে a ও b এর মানের ব্যাপারে কোনো সাহায্য পাওয়া যায় কিনা।

প্রথমে x_3 কে x_1 এর সাপেক্ষে লিখে দেখা যাক- $x_3 \equiv ax_2 + b \equiv a(ax_1 + b) + b \equiv a^2x_1+ab+b(\bmod 10001)$ বা $x_3-a^2x_1 = b(a+1)+10001c$ যেখানে c একটি integer[১]. যদি আমরা a এর উপর লুপ চালাই তাহলে এই equation এ দুইটি unknown থাকবে b আর c. তাহলে আমরা আমাদের equation কে লিখতে পারি- $bp+cq = r$ যেখানে p, q, r constant এবং b এর মান সর্বোচ্চ 10000 হতে পারে। কীভাবে আমরা এরকম ভাবে লিখলাম? সহজ, আমরা আগের সমীকরণে $p = a + 1$, $q = 10001$ আর $r = x_3 - a^2x_1$ বসাই তাহলেই আমরা $bp + cq = r$ পেয়ে যাব। এখন আমরা b এর এমন সমাধান চাই যেন তা অঋণাত্মক ও সর্বোচ্চ 10000 হয়। আমরা সাধারনত unknown কে x, y, z ইত্যাদি দ্বারা প্রকাশ করে থাকি। তাই আমরা সুবিধার জন্য b কে x ও c কে y দ্বারা পরিবর্তন করি। তাহলে আমাদের সমীকরণ দাঁড়ায় $px + qy = r$.

আমরা কিন্তু এই ধরনের সমীকরণ সমাধান করা জানি। আমাদের euclid এর অ্যালগরিদম ব্যবহার করতে হবে বা extended gcd খাটাতে হবে। সেক্ষেত্রে আমরা এমন x_0, y_0 পাবো যেন $px_0 + qy_0 = g$ যেখানে g হল p ও q এর gcd. এর general solution হবে $p(x_0+tq)+q(y_0-tp) = g$ যেখানে t হল integer constant. এটা আসলে extended gcd এরই একটি properties. এখন আমাদের ডান পাশকে r এর সমান করার জন্য দুই দিকে আরও একটি integer constant ধরা যাক k দিয়ে গুন করতে হবে ($k = r/g$). তাহলে দাঁড়াবে $kp(x_0 + tq) + kq(y_0 - tp) = r$. আমরা বলেছিলাম যে আমরা এমন অঋণাত্মক b (যা পরবর্তীতে x হয়ে গিয়েছিল) চাই যা এই equation কে satisfy করবে এবং 10001 এর থেকে ছোট হবে (বার বার 10000 এর সমান বা ছোট বলার থেকে 10001 এর থেকে ছোট বলা সহজ)। অর্থাৎ $0 \leq k(x_0 + tq) < 10001$ বা $-x_0/q \leq t < 10001/qk - x_0/q$. একটু উপরে যদি তাকাও তাহলে বুঝতে পারবে $q = 10001$. তার মানে $-x_0/10001 \leq t < 1/k - x_0/10001$. এখানে খেয়াল কর $1/k$ কিন্তু খুবই ছোট সংখ্যা, $[-x_0/10001, 1/k - x_0/10001)$ এই বাউন্ডের ভেতরে আসলে একটির বেশি integer থাকবে না। এর মানে এতক্ষণ এতো খেটেখুটে এতো এতো constant, এতো এতো equation

[১] $a \equiv b(\bmod c)$ হলে আমরা লিখতে পারি $a = b + kc$ যেখানে k একটি integer.

সমাধান করে আমরা জানতে পারলাম যে, প্রতিটি a এর জন্য আমরা আসলে সর্বোচ্চ 1টি b পাবো যেটা আমাদের x_1 হতে x_3 দিতে পারবে।

তাহলে সারমর্ম দাঁড়াল- আমরা a এর উপর লুপ চালাবো। Extended gcd ব্যবহার করে b এর মান বের করব। এরপর একটি T এর লুপ চালিয়ে $x_1 \dots x_{2T}$ বের করে মিলিয়ে দেখবো যে এরা আমাদের ইনপুট এর সাথে মেলে কিনা। আমাদের সময় লাগবে $a \times T = 10^6$. সাথে extended gcd এর একটা ফ্যাক্টর আছে, আমরা সেটাকে গণায় ধরলাম না। আর আসলে আমাদের সকল $x_1 \dots x_{2T}$ এর মান বের না করলেও চলে। যখন কোনো একটি মান ম্যাচ করবে না তখন আমরা পরের a এর মান পরীক্ষা করব। এর ফলে সময় আরও কম লাগবে।

UVa 10375 Choose and divide

সমস্যা: তোমাকে p, q, r, s দেওয়া আছে যেখানে এদের সবাই অঋণাত্মক ও সেই সাথে সর্বোচ্চ 10000. সেই সাথে $p \geq q$ ও $r \geq s$. তোমাকে $\binom{p}{q}/\binom{r}{s}$ বের করতে হবে। এটা বলা আছে যে উত্তর 10^8 এর বেশি হবে না এবং তোমাকে 5 digit precision পর্যন্ত প্রিন্ট করতে হবে।

সমাধান: আমি জানি না জাজের টেস্ট কেস কতটুকু শক্তিশালী। আমি এই প্রবলেম পেলে কি কি সাবধানতা অবলম্বন করে সমাধান করতাম তা দেখাই। প্রথমত এটা খেয়াল করতে হবে যে উত্তর ছোট হবে মানে এই না যে $\binom{p}{q}$ বা $\binom{r}{s}$ ছোট। এমন হতে পারে যে দুই জনই অনেক বড় কিন্তু ভাগ করলে ছোট হয়ে যায়। আবার সেই সাথে যেহেতু উত্তর আমাদের decimal এ দিতে হবে সুতরাং আমাদের খুব perfectly এদের মান বের করতে হবে সেরকম কথাও নেই। সুতরাং আমাদের মোটামোটি সাবধানতা অবলম্বন করলেই চলবে। সাবধানতা কেন? মনে কর তুমি প্রথমে $\binom{r}{s}$ বের করেছ এটা অনেক বড়, সুতরাং তোমার precision loss হয়ে গিয়েছে double এ রাখতে।[১] ফলে তুমি যখন আরেকটি বড় সংখ্যা $\binom{p}{q}$ কে ভাগ করবা তখন সেই আগের precision loss তোমাকে ভোগাবে। মানে 101 কে 100 দিয়ে ভাগ করা আর 100 কে 100 দিয়ে ভাগ করা তো আলাদা জিনিস তাই না (যদি কম্পিউটার মাত্র ২ ঘর precision রাখে)? তাহলে কি করা যায়? প্রথমত আমরা আমাদের term কে একটু ভেঙ্গে লিখি। $\frac{(p-q+1)\dots p}{1\dots q} \times \frac{1\dots s}{(r-s+1)\dots r}$. অর্থাৎ উপরে একগুচ্ছ সংখ্যা আছে, নিচে এক গুচ্ছ সংখ্যা আছে। উত্তর 10^8 এর বেশি হবে না, আর এদের কোনো সংখ্যাই 10^4 এর বেশি হবে না।

তুমি যা করবা তাহলো একটি variable নিবা ধরা যাক ans. আর L আর R দিয়ে ans এর জন্য lower এবং upper bound ঠিক করা হয়। এই L আর R কি তা একটু পরেই পরিস্কার হবে। যাই হোক, ans এর মান শুরুতে 1 কারণ আমরা এখন একে একে সব সংখ্যা গুন ভাগ করব। এখন দেখো যে আমাদের উত্তর L এর থেকে ছোট কিনা, ছোট হলে এবং উপরে যদি এখনো সংখ্যা বাকি থাকে তাহলে একে উপরের একটি সংখ্যা দিয়ে গুন করবা। আর যদি ans এর মান R এর থেকে বড় হয় এবং নিচে যদি এখনো সংখ্যা থেকে থাকে তাহলে নিচের কোনো একটি সংখ্যা দ্বারা ভাগ করব। আর আগের দুটি condition যদি পূর্ণ না হয় তাহলে আর কি, বাকি যেকোনো সংখ্যা দিয়ে গুন ভাগ করলেই হবে। যদি তোমরা L = 1 আর $R = 10^{12}$ নাও তাহলেই হয়তো হয়ে যাবে। এখানে কোনো ধরা বাধা নিয়ম নাই। মুল আইডিয়া হল আমি আমার ans কে কখনও খুব বড় হতে দেব না আবার খুব ছোট হতে দেব না। তুমি চাইলে এই প্রসেসের শুরুতেই উপরের সব সংখ্যাকে একটি vector আর নিচের সব সংখ্যাকে আলাদা আরেকটি vector এ রাখতে পারো। অথবা যেহেতু উপর নিচ দু জায়গাতেই সংখ্যাগুলি দুটি গ্রুপে আছে সেহেতু দুটি variable রেখেও তুমি রেঞ্জ বা সংখ্যা সামলাতে পারো।

[১] তুমি এভাবে চিন্তা করতে পারো যে, কম্পিউটার double এ কয়েক ঘর রাখতে পারে মাত্র। এখন যদি তুমি এখানে অনেক বড় মান রাখতে যাও তাহলে কম্পিউটার শেষের দিকের অংক আর রাখবে না, সেখানে সে 0 বসিয়ে দেবে।

UVa 10791 Minimum Sum LCM

সমস্যা: তোমাকে N ($< 2^{31}$) দেওয়া থাকবে। তোমাকে এমন কিছু সংখ্যা (একাধিক হতে হবে) বের করতে হবে যেন তাদের লসাগু N হয় এবং সেই সংখ্যাগুলির যোগফল সর্বনিম্ন হয়। যেমন N = 12 হলে আমাদের সংখ্যা গুলি হবে 3 আর 4. কারণ এদের লসাগু 12 এবং এদের যোগফল 7 যা সর্বনিম্ন। তোমাকে সেই সর্বনিম্ন যোগফলটি প্রিন্ট করতে হবে। মোট 100 টি test case দেওয়া থাকবে।

সমাধান: তোমাকে যদি কিছু সংখ্যা দেওয়া থাকে তাহলে তাদের লসাগু কীভাবে বের করবে? অনেক উপায়েই বের করা যায়। তার মাঝে একটি উপায় হলো সংখ্যাগুলিকে prime factorization করা। এরপর প্রতিটি prime এর জন্য দেখা যে তোমাকে দেওয়া সংখ্যাদের মাঝে কোনো সংখ্যায় তার সবচেয়ে বড় power কত আছে। এভাবে প্রতিটি prime এর জন্য সেই বড় power দিয়ে গুন করলেই তুমি লসাগু পেয়ে যাবে। উদাহরণ দেওয়া যাক। ধর আমরা 4, 6, 18 এর লসাগু বের করতে চাই। এখন $4 = 2^2$, $6 = 2 \times 3$ আর $18 = 2 \times 3^2$. তাহলে 2 এর সবচেয়ে বড় power আছে 2, আর 3 এর সবচেয়ে বড় power আছে 2। তাহলে আমাদের লসাগু হবে $2^2 \times 3^2 = 36$. এর মানে যদি আমাদের লসাগু দেওয়া থাকে তাহলে একে আগে prime factorize করে নেই। যেহেতু লসাগু খুব জোড় 2^{31} হতে পারে আমরা খুব সহজেই $\sqrt{N}$ এ তার prime factorize করতে পারি। বা তাতে যদি খুব বেশি সময় লেগে যায় তাহলে আমরা $\sqrt{N}$ পর্যন্ত prime generate করে রেখে N কে prime factorize করতে পারি। ধরা যাক এই prime factorization টি হল $p_1^{k_1} p_2^{k_2} p_3^{k_3} \ldots$. এর মানে দাঁড়াল আমরা যেই সংখ্যাগুলি নির্বাচন করব তাদের প্রতিটিতে $p_1, p_2 \ldots$ এর বাইরে কোনো prime থাকা যাবে না, আর এদের power আবার যথাক্রমে $k_1, k_2 \ldots$ এর বেশি হওয়া যাবে না। কীভাবে তুমি এই prime গুলিকে সংখ্যাগুলির মাঝে distribute করলে সংখ্যাগুলির যোগফল সর্বনিম্ন হয়? তোমরা চাইলে এখানে একটু বিরতি নিয়ে চিন্তা করতে পারো।

প্রথমত, কোনো একটি prime একাধিক সংখ্যায় থাকবে না। কারণ একাধিক সংখ্যার মাঝে কোনো একটিতে prime টির সর্বোচ্চ power থাকবে। সেটি বাদে বাকি গুলি মুছে দিলে আমাদের সংখ্যাগুলির যোগফল আরও কমে যাবে তাই না? একটা উদাহরণ দেখা যাক। ধর $12 = 2^2 \times 3$ তাহলে কোনো একটি সংখ্যায় 2^2 থাকবে। তাহলে আমরা চাইলে একটি 2^2 রেখে বাকি 2 সব মুছে ফেলতে পারি। আবার 2^2 আর 3 চাইলে একই সংখ্যাতেও থাকতে পারে বা ভিন্ন সংখ্যাতেও থাকতে পারে। যেমন $2^2, 3$ হতে পারে বা $2^2 3$ হতে পারে। কোনটা লাভজনক? অবশ্যই $2^2 + 3 < 2^2 3$. একটু কি বুঝতে পারছ কীভাবে prime গুলিকে রাখলে যোগফল সবচেয়ে কম হবে? হ্যাঁ তাদের আলাদা আলাদা করে রাখতে হবে। অর্থাৎ $p_1^{k_1}, p_2^{k^2}, \ldots$ এরকম। কেন? খেয়াল কর এখানে মুল ব্যাপার হল ab ছোট নাকি $a+b$? দেখা যাক, $ab ? a+b \Rightarrow ab-b ? a \Rightarrow b ? \frac{a}{a-1}$. এখানে ? মানে হল আমরা জানি না যে এখানে কি বসবে <, > নাকি =. আমাদের ডান পাশ প্রায় 1 এর কাছাকাছি একটা সংখ্যা। আর বাম পাশ তো কমপক্ষে 2. এর মানে সবসময় ab সংখ্যা $a+b$ এর থেকে বড় হবে যেখানে a, b প্রতিটি prime এর power (আমাদের এ সমস্যার ক্ষেত্রে এটি সত্য, তাই বলে সব ক্ষেত্রে $ab > a+b$ না যেমন $1 * 1 > 1 + 1$ কিন্তু সত্যি না)। অর্থাৎ আমাদের optimal সংখ্যাগুলি হবে $p_1^{k_1}, p_2^{k^2}, \ldots$.

সব শেষে একটা জিনিস, আশা করি খেয়াল করেছ যে তোমাদের সেটে একাধিক সংখ্যা থাকতে হবে। সুতরাং যদি প্রদত্ত N এ মাত্র একটি প্রাইম থাকে তাহলে কি করবা সেটা নিজেরা চিন্তা করে বের কর।

UVa 12716 GCD XOR

সমস্যা: মোট 10,000 টি টেস্ট কেস। প্রতি কেসে তোমাকে N ($\leq 3 \times 10^7$) দেওয়া থাকবে। তোমাকে বলতে হবে এমন কতগুলি (A, B) আছে যেন $1 \leq B \leq A \leq N$ হয় ও $gcd(A, B) = xor(A, B)$ হয়। যেমন $N = 7$ এর জন্য উত্তর 4 এবং (A, B) গুলি হল $(3, 2), (5, 4), (6, 4), (7, 5)$.

সমাধান: দুটি সংখ্যার gcd এর সাথে xor এর কোনই সম্পর্ক আপাত দৃষ্টি তে দেখা যাচ্ছে না। আমরা যেসব পদ্ধতিতে gcd বের করি তার সাথে xor তো দূরের কথা bitwise representation

এরই কোন সম্পর্ক দেখা যাচ্ছে না। এরকম অবস্থায় যেটা করা উচিত সেটা হলো pattern খোঁজা। একটা ছোট কোড করে N এর মান 100 পর্যন্ত যেসব pair পাওয়া যায় তাদের প্রিন্ট করে ফেলো। কিন্তু এরপরও কোনো pattern পাওয়া যায় না। যেমন কিছু বিদঘুটে pair হলো $(85, 68)$, $(90, 72)$ ইত্যাদি। যারা আসলেই pattern খোঁজায় পটু তারা হয়তো ইতোমধ্যেই pattern পেয়ে গেছে। যদি খেয়াল না করে থাকো তাহলে বলি এদের gcd বা xor হল তাদের মাঝের পার্থক্যের সমান। যদিও এটা ঠিক যে যেকোনো pair এর পার্থক্য সবসময় তার gcd হয় না বা xor ও হয়না। তবে যদি তাদের xor ও gcd সমান হয় তাহলে তাদের পার্থক্যও এদের সমান হয়। অন্তত N এর মান সর্বোচ্চ 100 এর জন্য যতগুলি pair পেয়েছ তাদের সবার ক্ষেত্রে এটা সত্য। এখন দুটি জিনিস। প্রথমত- এটা কি আসলেই সত্য? প্রমাণ কি? দ্বিতীয়ত- এটা কীভাবে efficiently কোড করা যাবে।

অনেকে প্রথম প্রশ্ন এর উত্তর না বের করেই দ্বিতীয় অংশে চলে যায়। অর্থাৎ প্রমাণ না করেই কোড করে বা কোড কীভাবে করবে তা চিন্তা করে। প্রোগ্রামিং কন্টেস্ট এর সময় প্রমাণ খুব একটা দরকারি না। তুমি আরও নিশ্চিত হতে হয়তো N এর মান 1000 বা 10,000 পর্যন্ত চেক করে দেখতে পারো। যদি এতো পর্যন্ত তোমার claim সত্যি হয় তাহলে ধরে নেয়া যায় যে আসলে সব N এর জন্য তোমার claim সত্য হবে। যদিও যতক্ষণ না প্রমাণ করছ ততক্ষণ মনে একটা খচখচ করতে থাকে। তাহলে চল এটা প্রমাণ করা যাক। খেয়াল কর, $A - B \leq xor(A, B)$. কেন? যদি B তে যেসব bit 1 আছে A তেও যদি তারা 1 থাকে তাহলে $A - B$ আর $xor(A, B)$ সমান হবে। অন্যথা xor এর মান বেশি চলে আসবে। আবার অন্য দিকে $gcd(A, B)$ কিন্তু A আর B দুজনকেই ভাগ করে। তাই $gcd(A, B)$ কিন্তু $A - B$ কে ভাগ করে। অর্থাৎ $gcd(A, B) \leq A - B$. তাহলে দেখতে পাচ্ছ $gcd(A, B) \leq A - B \leq xor(A, B)$. অর্থাৎ যদি xor আর gcd সমান হয় তাহলে তারা $A - B$ এর সমান হবে। কি সুন্দর সহজে প্রমাণ হয়ে গেল। কিন্তু মনে রেখো এই প্রমাণ সহজ হয়েছে কারণ তুমি আগে থেকেই জানো যে এখানে $A - B$ এর কোনো ভূমিকা আছে। নাহলে তুমি $A - B$ এর সাথে gcd বা xor এর সম্পর্ক খুঁজতে যেতে না। মোট কথা তোমার pattern খোঁজা বা explore করার মানসিকতা থাকতে হবে।

এখন তাহলে দ্বিতীয় প্রশ্নে ফিরে আসা যাক। কীভাবে এটি সমাধান করবা। প্রথমত নিশ্চয় তুমি A ও B দুটির উপরেই লুপ চালাতে পারবা না। আবার শুধু A বা শুধু B এর উপরে লুপ চালিয়ে কি লাভ হবে তাও বুঝা যাচ্ছে না। এরা দুজন ছাড়া লুপ চালানোর মত candidate আছে gcd. দেখা যাক gcd এর উপর লুপ চালিয়ে লাভ হয় কিনা। ধরা যাক আমাদের gcd এর লুপের variable হলো g. তাহলে আমরা A ও B সম্পর্কে কি বলতে পারি? প্রথমত A ও B উভয়েই g এর multiple (গুণিতক)। দ্বিতীয়ত A - B হবে g এর সমান (আমরা কিছুক্ষণ আগে তাই প্রমাণ করেছি)। এর মানে আমরা যদি A জানি তাহলে B হবে $A - g$. আমরা যদি A এর উপর লুপ চালাতে চাই তাহলে কীভাবে চালাব? একটি উপায় হলো 1 হতে N পর্যন্ত i এর লুপ চালিয়ে দেখা যে i কি g দ্বারা বিভাজ্য কিনা। কিন্তু এতে প্রতিবার $\mathrm{O}(N)$ এর লুপ চালাতে হয়। তা না করে তুমি সরাসরি g এর multiple এর উপর লুপ চালাতে পারো। g হতে N পর্যন্ত এবং প্রতিবার লুপের মান g পরিমাণ করে বাড়িয়ে [for(A = g; A <= N; A += g)] অথবা 1 হতে N/g পর্যন্ত i এর লুপ চালাও তাহলে A হবে $i \times g$. তাহলে এই nested loop চলতে সময় লাগবে $\mathrm{O}(N/g)$. যেহেতু g এর লুপ 1 হতে N পর্যন্ত চালাচ্ছ তাই মোট সময় লাগবে $N/1 + N/2 + \ldots N/N$ এবং এটি আসলে $N \log N$. সুতরাং এই দুটি nested লুপ চালিয়ে আমরা A, B আর g জেনে গেলাম। আমরা খুব সহজেই চেক করতে পারি যে $xor(A, B) = g$ কিনা। কিন্তু $gcd(A, B) = g$ কিনা তা চেক করতে তো আরও একটা $\mathrm{O}(\log N)$ লাগবে। কিন্তু একটু চিন্তা করে দেখো এই চেক এর কোনো দরকার নেই। কারণ A আর B পাশাপাশি g এর multiple. মানে $A = ig$ হলে $B = (i-1)g$ এবং এদের মাঝে g ছাড়া আর কোনো সাধারণ গুণনীয়ক নেই কারণ g ছাড়া এরা হলো i আর $i-1$. আর পাশাপাশি দুটি সংখ্যা সবসময় coprime. কেন? মনে কর তারা coprime না। এর মানে এরা p দ্বারা বিভাজ্য। এর মানে এদের বিয়োগফল $i-(i-1)$ ও p দ্বারা বিভাজ্য (দুটি সংখ্যা p দ্বারা বিভাজ্য হলে তাদের বিয়োগফলও p দ্বারা বিভাজ্য হবে)। মানে 1 সংখ্যাটি p দ্বারা বিভাজ্য। তাতো সম্ভব না তাই না?

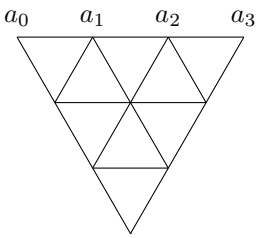
তাহলে আমরা আমাদের সমাধান পেয়ে গিয়েছি। আমরা প্রথমে gcd এর উপর লুপ চালাবো। এর পর A এর উপর লুপ চালাবো যেটা হবে gcd এর multiple. আর $B = A - g$. সবশেষে চেক করব $xor(A, B) = g$ কিনা। শেষ!

তাহলে এই সমস্যা থেকে কি শিখলাম? শিখলাম যে মাঝে মাঝে ধৈর্য ধরে pattern খুঁজতে হয়।

UVa 1635 Irrelevant Elements

সমস্যা: মনে কর তোমার বন্ধু n আর m দুটি সংখ্যা নির্বাচন করেছে ($1 \leq n \leq 10^5, 2 \leq m \leq 10^9$). এবারে সে n টি সংখ্যা $a_0, a_1, \ldots a_{n-1}$ নির্বাচন করেছে যেখানে প্রতিটি সংখ্যাই 0 হতে $m-1$ এর মাঝে। এবার সে পাশাপাশি দুটি করে সংখ্যা যোগ করল। তাহলে সে পেল $a_0+a_1, a_1+a_2 \ldots a_{n-2}+a_{n-1}$ এই $n-1$ টি সংখ্যা। এবার আবার সে একই কাজ করল এবং পেল $n-2$ টি সংখ্যা। এভাবে সে করতেই থাকল যতক্ষণ না সে একটি সংখ্যা পেল। এবার সে এই সংখ্যার $\mathrm{mod} m$ নিলো। এখন তোমাকে বলতে হবে শেষ সংখ্যার উপরে কোন কোন a_i এর কোনো প্রভাব নেই। যেমন $n=3$ ও $m=2$ হলে শেষ সংখ্যার উপরে a_1 এর কোনো প্রভাব নেই। কেন? কারণ $n=3$ এর জন্য শেষ সংখ্যাটি হলো $a_0+2a_1+a_2 \bmod m$ আর যেহেতু তুমি 2 দিয়ে mod নিচ্ছ সেহেতু শেষ সংখ্যাতে a_1 এর কোনো প্রভাব নেই।

সমাধান: যারা অভিজ্ঞ তারা এই সমস্যা পড়ার সাথে সাথেই বুঝতে পারবে যে শেষ সংখ্যা কি হবে। এখানেই আসলে অভিজ্ঞ আর অনভিজ্ঞদের পার্থক্য। অনভিজ্ঞরা বেশ কিছু সময় ব্যয় করবে এই common জিনিস বের করতে আর অভিজ্ঞরা এক দেখাতেই বুঝতে পারে যে শেষ মানটা কি হবে। যদি তোমরা না জানো যে শেষ মান কি হবে তাহলে n এর বিভিন্ন মানের জন্য শেষ মান বের করে দেখতে পারো। যেমন $n=2$ এর জন্য a_0+a_1. $n=3$ এর জন্য $a_0+2a_1+a_2$. $n=3$ এর জন্য $a_0+3a_1+3a_2+a_3$. বুঝতে পারছ কি হচ্ছে? n এর জন্য এই মান হবে $\binom{n-1}{0}a_0 + \binom{n-1}{1}a_1 + \binom{n-1}{2}a_2 + \ldots \binom{n-1}{n-1}a_{n-1}$. কেন? কারণটা বুঝার সবচেয়ে ভাল উপায় হল pascal এর triangle এর মত আঁকা এবং বের করা প্রতিটি সংখ্যার contribution কত করে। চিত্র ৪.১ এ জিনিসটা বুঝার চেষ্টা করে দেখতে পারো। আমরা কিন্তু ত্রিভুজই এঁকেছি তবে উলটো করে। এখানে আমাদের প্রথম সংখ্যাটি হলো উপরের বাম কোণা এবং এর পর বাকি সংখ্যাগুলি পরপর আছে। এখন কোনো একটি সংখ্যার contribution হলো সেটি যেই স্থানে আছে সেখান থেকে নিচের কোণায় যত ভাবে যাওয়া যায় তা। যেমন চিত্রের a_1 হতে নিচের কোণায় যেভাবেই যাও না কেন তোমাকে একধাপ বামে আর দুই ধাপ ডানে যেতে হবে। 1টা L আর 2টা R কে আমরা $\binom{2+1}{1} = \binom{4-1}{1}$ ভাবে সাজাতে পারি (ভুলে যেও না আমাদের n এখানে কিন্তু 4)। একই ভাবে তুমি প্রতিটি a_i এর জন্য যদি একটু হিসাব কর তাহলে দেখবে a_i এর coefficient হবে $\binom{n-1}{i}$.



নকশা ৪.১: Pascal Triangle

মুল সমস্যায় ফিরে যাওয়া যাক। আমাদের প্রতিটি i এর জন্য বের করতে হবে যে $\binom{n}{i}$ (n কে মনে কর 1 কমিয়ে নিয়েছি) m দ্বারা বিভাজ্য কিনা। এখানে m কিন্তু prime নাও হতে পারে। $\binom{n}{r}$ বের করার অনেক উপায় আছে। তোমাকে বিভিন্ন উপায় একে একে চিন্তা করে দেখতে হবে। আমি plan করতে পারি যে প্রথমে দেখবো $\binom{n}{0}$ কি m দ্বারা ভাগ যায় কিনা, এর পর দেখবো $\binom{n}{1}$ এভাবে $\binom{n}{n}$ পর্যন্ত সবগুলি। সুতরাং আমাদেরকে পরের টার্মকে আগের টার্ম এর সাপেক্ষে কীভাবে বের করা যায় তা চিন্তা করতে হবে। সহজ, $\binom{n}{i}$ কে লিখতে পারি $\binom{n}{i} = \frac{n-i+1}{i}\binom{n}{i-1}$. এরপর m কে prime factorize করে ফেলো এবং বের কর এর ভেতরে কোন কোন prime আছে আর তাদের power কত। এবার একে একে $\binom{n}{i}$ গুলি বের কর। এজন্য তোমাকে প্রতিবার $n-i+1$ গুন করতে হবে আর i ভাগ করতে হবে। একটু চিন্তা করলে বুঝবে গুন আর ভাগের সময় তোমার কেবল মাত্র m এর prime divisor গুলোতেই আগ্রহ আছে। আর অন্য prime গুলো গোল্লায় গেলেও কোনো সমস্যা নেই। এখন তোমাকে যা করতে হবে তাহলো এই prime গুলির power মনে রাখা। যদি দেখো সবগুলি power

ই m এ থাকা power এর সমান বা বড় তাহলে বিভাজ্য নাহলে বিভাজ্য না।

তাহলে আমাদের সমাধান দাঁড়াল- প্রথমে m কে prime factorize করে এর prime divisor আর তার power গুলি বের করব। এর পর একে একে একে $\binom{n}{i}$ বের করব (i এর মান 0 হতে n পর্যন্ত)। প্রতিবার একটি সংখ্যা গুন আর ভাগ করব। এই গুন ভাগের সময় আসলে আমরা দেখবো m এর ভেতরে থাকা prime গুলি কতবার করে আছে। ভাগের সময় এই count কমবে আর গুনের সময় বাড়বে। গুন ভাগ হয়ে গেলে প্রতিটি prime এর power চেক করে দেখবো যে m এ থাকা power অপেক্ষা কেউ কম আছে কিনা। যদি কম থাকে তাহলে বিভাজ্য না, আর যদি সবাই বেশি বা সমান হয় তাহলে বিভাজ্য। n কিন্তু খুব একটা বড় না তাই তাতে খুব একটা বেশি prime factor থাকবে না। তাই এই সমাধানের complexity ও অতো বেশি হবে না।

UVa 10820 Send a Table

সমস্যা: তোমাকে N ($\leq 50,000$) দেওয়া থাকবে। বলতে হবে কতগুলি (x, y) আছে যাতে করে x ও y এর মাঝে কোনো সাধারণ গুণনীয়ক না থাকে ($1 \leq x, y \leq N$). মোট 600 টি test case থাকতে পারে।

সমাধান: x ও y এর মাঝে কোনো সাধারণ গুণনীয়ক থাকা যাবে না- এটা আসলে phi function বা Euler's totient function এর কথা মনে করিয়ে দিচ্ছে। $phi(i)$ বলে যে i এর সমান বা ছোট কতগুলি সংখ্যা আছে যার সাথে i এর কোনো সাধারণ গুণনীয়ক থাকবে না। এর মানে আমাদের প্রবলেমে $x \leq y$ এর জন্য উত্তর হবে $\sum_{i=1}^{N} phi(i)$ কারণ আমরা প্রতিবা $i = y$ কে fix করছি আর দেখছি যে কতগুলি x পাওয়া যায়। আর $phi(i)$ কীভাবে বের করতে হয় তা প্রোগ্রামিং কন্টেস্ট বইয়ে বা অন্য কোথাও দেখে নিতে পারো। যদি আমরা phi বের করা শিখে যাই তাহলে আমরা phi এর একটা cumulative sum এর অ্যারে বানিয়ে রেখে দিতে পারি। Cumulative sum এর অ্যারে মানে হল $sum(i) = phi(1) + \ldots phi(i)$। এটা তো বের করা সহজ $sum(i) = sum(i-1) + phi(i)$. যদিও আমরা $phi(i)$ এ i এর থেকে বড় সংখ্যা যারা i এর সাথে coprime তাদের পাই নাই তবে এক্ষেত্রে আমরা সকল $x \leq y$ এর জন্য সমাধান পেয়ে গিয়েছি। আর বাকি থাকে $i > j$ যা আসলে $i \geq j$ এর সমান, মানে আমরা কিছুক্ষণ আগে যে মান বের করেছি তার প্রায় সমান। প্রায় কেন বললাম সেটা তোমরাই একটু কাগজ কলম ব্যবহার করলে বুঝতে পারবে।

UVa 1262 Password

সমস্যা: তোমাকে দুটি 6×5 (6 টি row, 5 টি column) সাইজের character এর গ্রিড দেওয়া থাকবে। 5 length এর একটি শব্দ valid password হবে যদি শব্দটির i তম character টি দুটি গ্রিডেরই i তম কলামে থাকে। তোমাকে K দেওয়া থাকবে। K তম lexicographically smallest valid password প্রিন্ট করতে হবে।

সমাধান: Counting সমস্যার ক্ষেত্রে lexicographically smallest উত্তর বের করার একটি common সমস্যা এটি। মুল আইডিয়া হলো তুমি বাম হতে ডানে একে একে উত্তরটির character গুলি পূরণ করবে। মনে কর তুমি ইতোমধ্যেই প্রথম $i-1$ ঘর পূরণ করে ফেলেছ আর এরকম অবস্থায় K তম শব্দ বের করতে চাচ্ছ। এখন তুমি i তম ঘর পূরণ করবে। তুমি প্রথমে এখানে A বসানোর চেষ্টা কর। যদি দেখো মোট X ভাবে A বসানো যায় তাহলে দেখো, $X \geq K$ কিনা। যদি হয় এর মানে এই জায়গাতে A ই বসবে। আর যদি নাহয় তাহলে K হতে X বিয়োগ কর আর i তম স্থানে B বসিয়ে চেষ্টা কর। একই ভাবে প্রয়োজনে C, D, E এই ভাবে একে একে Z পর্যন্ত চেষ্টা করে যাও।

তাহলে কাহিনী দাঁড়াল যে আমরা যদি password এর প্রথম কিছু ঘর জানা থাকা অবস্থায় পরের ঘরগুলিকে কতভাবে পূরণ করা যায় তা যদি বের করতে পারি তাহলে আমাদের সমস্যা সমাধান হয়ে যাবে। আর এই প্রশ্নের উত্তর মনে হয় সহজ তাই না? বাকি column গুলির প্রতিটিতে কয়টি করে character বসানো সম্ভব তা গুন করলেই পেয়ে যাবা।

একটা ছোট উদাহরণ দেয়া যাক। মনে কর তুমি 10ম শব্দ বের করতে চাও। তুমি করলে কি, প্রথম স্থানে A বসালে। এর পর তোমার বন্ধুকে জিজ্ঞাসা করলে, আমি প্রথম স্থানে A বসিয়েছি, তুমি কি বলবে এই অবস্থায় কতগুলি valid শব্দ আছে? ধর তোমার বন্ধু বলল 1 টি। তুমি বুঝলে 10ম শব্দটি A দিয়ে শুরু না। সেই সাথে তুমি জেনে গেলে প্রথম শব্দটি A দিয়ে শুরু। সুতরাং এখন তোমার মাথা ব্যাথা 9ম শব্দ নিয়ে (কারণ প্রথম শব্দ তো বাদ)। এখন মনে কর তুমি প্রথম স্থানে B বসালে আর তোমার বন্ধুকে জিজ্ঞাসা করলে এবার বল কতগুলি valid শব্দ আছে। সে বলল 50 টি। তাহলে তুমি বুঝে গিয়েছ যে প্রথম স্থানে B বসবে। এবার তোমার চিন্তা দ্বিতীয় স্থান নিয়ে। তুমি এখানে A বসালে। আবার বন্ধুকে জিজ্ঞাসা করলে যে BA দিয়ে শুরু শব্দ কয়টি? তোমার বন্ধু বলল 4 টি। তোমার দরকার 9ম শব্দ। তাহলে বুঝে গেলে দ্বিতীয় স্থানে A না। এবং সেই সাথে তুমি এখন থেকে 5ম শব্দের সন্ধান করবে। দ্বিতীয় স্থানে মনে কর B বসালে আর তোমার বন্ধু বলল BB দিয়ে শুরু শব্দের সংখ্যা 0টি। তুমি এবার C দিয়ে চেষ্টা কর। এভাবে যদি চালিয়ে যাও তুমি দেখবে পুরো শব্দ বানিয়ে ফেলেছ।

আর কয়টি শব্দ বানানো সম্ভব তা বের করা বেশ সহজ। প্রথমে বের কর প্রথম কলামে কতগুলি character বসতে পারে। এটা বের করা সহজ, তুমি A হতে Z প্রতিটি character এর জন্য দেখবে যে সেই character টি দুটি গ্রিডের প্রথম কলামে আছে কিনা। এরকম করে বের কর কত গুলি character দুটি গ্রিডের প্রথম কলামে আছে। একই ভাবে তুমি প্রতিটি কলামের জন্য এই সংখ্যা বের কর। এদের গুন করলেই তুমি পেয়ে যাবে যে কত গুলি শব্দ বানানো সম্ভব। যেহেতু আমরা প্রথম দিকের কিছু কলামে character বসিয়ে ফেলব সুতরাং তাদের বাদে বাকি কলাম গুলি নিয়ে হিসাব করলেই আমরা সেই বাকি কলামগুলি ব্যবহার করে কত গুলি শব্দ বানানো সম্ভব তা পেয়ে যাব।

UVa 1636 Headshot

সমস্যা: একটি পিস্তল আছে। পিস্তলটিতে n টি গুলির slot আছে। কোন slot এ গুলি আছে কোন slot এ নেই তা বলা আছে। এখন তোমার বন্ধু আর তুমি russian roulette খেলবে। প্রথমে তোমার বন্ধু পিস্তলটি নিলো এবং গুলির slot ঘুরাল। এর ফলে এখন পিস্তলটি random একটি slot এ আছে। এই অবস্থায় সে fire করল কিন্তু কোনো গুলি বেরুলো না। এবার তোমার পালা। তোমার হাতে দুটি option. এক- তুমি আবার slot গুলি ঘুরাবে ফলে পিস্তলটি random slot এ যাবে অথবা দুই- তুমি কোনো রকম না ঘুরায়েই গুলি করবে। তোমাকে বলতে হবে কোন উপায়ে গুলি বের না হবার সম্ভাবনা বেশি। যদি দুই উপায়েই সমান হয় তাহলে বল যে সমান। n এর মান 100.

সমাধান: আমরা দুই উপায়ের সম্ভাব্যতা বের করব এবং তুলনা করে দেখবো যে তারা সমান কিনা বা সমান না হলে কোন ভাবে গুলি না বের হবার সম্ভাবনা বেশি। প্রথমে সহজটা বের করা যাক- তুমি যদি slot কে ঘুরাও তাহলে গুলি না বেরুনোর সম্ভাব্যতা। এটা আসলে বেশিই সহজ। তোমার মোট n টি slot আছে। ধরা যাক তার a টিতে গুলি আছে আর $n-a$ টি তে নেই। তুমি যদি slot কে randomly ঘুরাও তাহলে গুলি নাই এরকম slot আসার সম্ভাবনা হলো $\frac{n-a}{n}$. কেন? কারণ যেহেতু n টি slot আছে তাই randomly ঘুরালে n ধরনের slot আসতে পারে। আর এদের মাঝে $n-a$ টিতে গুলি নেই। তাই গুলি নেই এরকম slot আসতে পারে $n-a$ ভাবে। সুতরাং $n-a$ কে n দিয়ে ভাগ করলে আমরা গুলি নেই এরকম slot আসার সম্ভাবনা বের করে ফেলতে পারব।

এখন চল বের করা যাক যদি না ঘুরাও তাহলে গুলি বের না হবার সম্ভাবনা কত। যেহেতু এর আগের বারে তোমার বন্ধু যখন triggure টিপেছে তখন কোনো গুলি বের হয় নাই সুতরাং তুমি জানো যে এখন পিস্তল এমন জায়গায় আছে যার আগের slot এ গুলি ছিল না। সুতরাং তুমি খুব সহজেই গুনতে পারো এরকম অবস্থান কয়টি আছে। এখন প্রশ্ন হল এদের মাঝের কয়টি অবস্থানের জন্য পরের অবস্থানেও গুলি নেই। উদাহরণ দেয়া যাক। ধরা যাক slot এর অবস্থা 0011 (0 মানে গুলি নেই, 1 মানে গুলি আছে)। এখন তোমার বন্ধুর ক্ষেত্রে slot প্রথম দুটির কোনো একটিতে ছিল। যদি তোমার বন্ধুর কাছে প্রথম অবস্থানে থাকে তাহলে তোমার ক্ষেত্রে গুলি হবে না। আর যদি তোমার বন্ধুর কাছে দ্বিতীয় অবস্থানে থাকে তাহলে তোমার কাছে গুলি হবে। অর্থাৎ তোমার কাছে গুলি হবার সম্ভাবনা 0.5. যদি slot এর অবস্থা 010101 হয় তাহলে কিন্তু গুলি হবার সম্ভাবনা 1. কারণ তোমার বন্ধুর ক্ষেত্রে যেহেতু গুলি হয় নাই সুতরাং তোমার ক্ষেত্রে গুলি হবেই কারণ সব 0 এর পরে 1 আছে। এটা সম্ভাব্যতার খুব basic সমস্যা।

UVa 10491 Cows and Cars

সমস্যা: একটি game show তে a টি দরজার পেছনে গাড়ি আর b টি দরজার পেছনে গরু আছে। তুমি প্রথমে একটি দরজা নির্বাচন করলে তোমার জন্য। এর পর উপস্থাপক তোমার দরজা ছাড়া অন্য দরজার মাঝে c টি দরজা খুলে দেখাল যে সেসবের পেছনে গরু আছে ($c < b$)। এবার তুমি তোমার দরজা বাদে অন্য যেকোনো দরজা নির্বাচন করলে। বলতে হবে তোমার গাড়ি পাবার সম্ভাবনা কত।

সমাধান: ধরা যাক $n = a + b$ অর্থাৎ মোট দরজার সংখ্যা n. এখন মনে কর তুমি একটি দরজা নির্বাচন করলে, তাহলে এতে গাড়ি থাকার সম্ভাবনা $\frac{a}{n}$. বাকি আছে $n-1$ টি দরজা। এর মাঝে c টি দরজা আমাদের উপস্থাপক খুলে দেখিয়ে দিয়েছে যে তাদের পেছনে গরু আছে। তাহলে বাকি থাকে মোট $n-1-c$ টি দরজা। যেহেতু আমি যেই দরজা নির্বাচন করেছি তার পেছনে গাড়ি আছে (কারণ আমরা ধরে নিয়েছি যে a/n probability তে আমাদের প্রথম দরজার পেছনে গাড়ি আছে) তাই বাকি $n-1-c$ টি দরজার মাঝে গাড়ি আছে $a-1$ টি। তাহলে আমি যদি বাকি $n-1-c$ টি দরজার একটি নির্বাচন করি তাতে গাড়ি থাকার সম্ভাবনা $\frac{a-1}{n-1-c}$. সুতরাং আমরা প্রথম যেই দরজা নির্বাচন করেছিলাম তাতে এবং দ্বিতীয় যেই দরজা নির্বাচন করলাম তাতে গাড়ি থাকার সম্ভাবনা $\frac{a}{n} \times \frac{a-1}{n-1-c}$.

আবার এমনও হতে পারে যে আমরা যেই দরজা প্রথমে নির্বাচন করেছিলাম তাতে গরু আছে। এটা হবার সম্ভাবনা $\frac{b}{n}$. এবার বাকি $n-1$ টি দরজার মাঝে উপস্থাপক c টি দরজা খুলে গরু দেখিয়েছিল। বাকি থাকে $n-1-c$ টি দরজা। এর মাঝে গাড়ি আছে a টি। আমরা যদি বাকি $n-1-c$ টি দরজা থেকে randomly নির্বাচন করি তাহলে গাড়ি পাবার সম্ভাবনা $\frac{a}{n-1-c}$. অর্থাৎ প্রথম দরজায় গরু আর দ্বিতীয় দরজায় গাড়ি পাবার সম্ভাবনা $\frac{b}{n} \times \frac{a}{n-1-c}$. তাহলে গাড়ি পাবার মোট সম্ভাবনা দাঁড়াবে $\frac{a}{n} \times \frac{a-1}{n-1-c} + \frac{b}{n} \times \frac{a}{n-1-c}$.

যারা probability তে একদম নতুন তাদের মনে প্রশ্ন আসতে পারে এরকম গুন করে যোগ করলাম কেন। অনেক ভাবেই এর ব্যাখ্যা দেওয়া যায়। মনে কর তোমার কাছে তিনটি coin আছে এবং এদের head পরার সম্ভাবনা p_1, p_2, p_3. প্রথমে তুমি প্রথম coin টস করবে। যদি head পরে তাহলে দ্বিতীয় coin, আর যদি প্রথম coin এ tail পরে তাহলে তৃতীয় coin টস করবে। বলতে হবে সবশেষে যেই কয়েনটি টস করবে তাতে head পরার সম্ভাবনা কত। খেয়াল কর, প্রথম coin টস করার পর কিন্তু দুই রকম ঘটনা ঘটতে পারে। p_1 সম্ভাবনায় head পরতে পারে, এবং সেই ক্ষেত্রে তুমি দ্বিতীয় coin টস করবে এবং তাতে head পরার সম্ভাবনা p_2. এই ঘটনা দুটি একত্রে ঘটার সম্ভাবনা $p_1 p_2$. একই ভাবে আমরা বলতে পারি প্রথম coin এ tail ও তৃতীয় coin এ হেড পরার সম্ভাবনা $(1-p_1)p_3$. যখন দুটি ঘটনা একত্রে ঘটানো দরকার তখন তাদের সম্ভাবনাকে গুন করতে হয়। এবার আমাদের বের করতে হবে শেষের coin এ head পরার সম্ভাবনা। শেষের coin দ্বিতীয় coin হতে পারে অথবা তৃতীয় coin হতে পারে। সুতরাং দ্বিতীয় coin এ head "অথবা" তৃতীয় coin এ head পরার সম্ভাবনা হলো $p_1 p_2 + (1-p_1)p_3$. এখানে যোগ করেছি, কারণ আমরা দুইটি ঘটনার কোনো একটি ঘটার সম্ভাবনা বের করতে চেয়েছি। অর্থাৎ "অথবা" হলে আমাদের সম্ভাব্যতাকে যোগ করতে হয়। কিন্তু কেন "এবং" হলে গুন আর "অথবা" হলে যোগ? এভাবে চিন্তা করতে পারো- মনে কর তোমার সামনে একটি dart board আছে। তুমি সেই dart board কে সমান তিনভাগে ভাগ করলে। এর পর একভাগকে সবুজ, এক ভাগ কে নীল আর আরেক ভাগকে লাল রং করলে। যেহেতু তিনটি রঙের জায়গার ক্ষেত্রফল সমান তাই তুমি যদি একটি dart ছুড়ে মার আর যদি সেটা board এর একটি random বিন্দুতে গিয়ে আঘাত করে সেই বিন্দুটির রং লাল হবার সম্ভাবনা $1/3$ (দুই রঙের মাঝে পড়লে কি হবে চিন্তা না করলেও চলবে)। একই ভাবে সবুজ হবার সম্ভাবনা $1/3$, নীল হবার সম্ভাবনা $1/3$. এখন যদি বলে সবুজ "অথবা" নীল হবার সম্ভাবনা কত। কি করবে? যোগ হবে তাই না? কারণ নীল হবার সম্ভাবনা $1/3$, সবুজ হবার সম্ভাবনা $1/3$ তাই নীল বা সবুজ হবার সম্ভাবনা $1/3 + 1/3 = 2/3$. বা অন্যভাবে যদি চিন্তা কর তাহলে দেখবে "নীল বা সবুজ" এর ক্ষেত্রফলকে যদি তুমি মোট ক্ষেত্রফল দিয়ে ভাগ কর তাহলে দেখবে $2/3$ পেয়েছ। সুতরাং অথবা হলে যোগ। এখন চিন্তা কর তোমার কাছে এরকম দুটি board আছে আর তুমি দুটি বোর্ডে দুটি dart মারলে। দুটিই নীল বিন্দুতে পরার সম্ভাবনা কত? $1/3 \times 1/3 = 1/9$ কেন? কারণ এই দুটি dart 9 ভাবে পরতে পারে- (লাল, লাল), (লাল, নীল), (লাল, সবুজ), এরকম আরও 6 রকম। এবং যেহেতু সবগুলি রঙে পরার সম্ভাবনা একই তাই এদের যেকোনো একটি combination পাবার সম্ভাবনা $1/9$. আর এদের মাঝে একটির ক্ষেত্রেই দুটিই

নীল। অর্থাৎ $1/9$. সুতরাং এবং হলে গুন। ঠিক counting এর মতই। Counting এর ক্ষেত্রেও কিন্তু "অথবা" এর ক্ষেত্রে তুমি যোগ কর আর "এবং" হলে গুন।

UVa 11181 Probability Given

সমস্যা: একটি দোকানে N টি জিনিস আছে ($N \leq 20$). এদের প্রতিটির কেনার সম্ভাব্যতা দেওয়া আছে $p_1, p_2 \ldots p_N$. তোমাকে বলা আছে যে মোট r টি জিনিস কেনা হয়েছে। তাহলে প্রতিটি জিনিসের জন্য বলতে হবে যে সেটি কেনার সম্ভাব্যতা কত।

সমাধান: মনে করে বস না যে i তমটি কেনার সম্ভাব্যতা তো দেয়াই আছে p_i. এখানে কিন্তু অতিরিক্ত আরেকটি কথা বলা আছে। "তুমি যদি জানো যে r টি জিনিস কেনা হয়েছে" তাহলে কোন জিনিস কেনার সম্ভাব্যতা কত তা বল। তোমাকে যখন $p_1, p_2 \ldots p_N$ ইত্যাদি বলা ছিল সেটা ছিল unconditional. কোনো শর্ত নেই, কোনো ধরা বাধা নেই কিছু নেই। এই অবস্থায় i তম জিনিস কেনার সম্ভাবনা হলো p_i. কিন্তু তোমাকে যা জিজ্ঞাসা করেছে তাতে কিন্তু একটা বাড়তি condition যোগ করে দিয়েছে- যদি তুমি জানো যে r টি জিনিস কেনা হয়েছে তাহলে বল ...। ব্যাপারটা একটা ছোট উদাহরণে বুঝানো যাক। মনে কর Real Madrid আর Barcelona এর মাঝে ফাইনাল খেলা হচ্ছে। যে জিতবে সে একটা cup পাবে। এখন খেলার আগে তোমার ফেসবুকের বিশেষজ্ঞ(!) বন্ধুদের status পড়ে তুমি বুঝতে পারছ যে যেহেতু আজ Messi খেলছে না তাই Barcelona জেতার সম্ভাবনা 30% আর Real Madrid এর 70%. কিন্তু মনে কর পরদিন তুমি ফেসবুক খুলে দেখলে যে Barcelona কাপ জিতেছে। এখন যদি তোমাকে কেউ জিজ্ঞাসা করে "Barcelona কাপ জিতেছে, এখন বল barcelona খেলায় জিতেছে তার সম্ভাবনা কত"। এর উত্তর সহজ তাই না? "100%". কারণ তোমাকে বলা হয়েছে "Barcelona জিতেছে" এই সাপেক্ষে বল যে Barcelona জেতার সম্ভাবনা কত। এই সমস্যাতেও একই ঘটনা ঘটছে। তোমাকে বলা হয়েছে যে "r টি জিনিস কেনা হয়েছে" তাহলে বল কোন জিনিস কেনার সম্ভাবনা কত।

দুইটা জিনিস নেয়া যাক। ধরা যাক $p_1 = 0.3$ আর $p_2 = 0.8$. তাহলে-

- 1 ও 2 দুটিই কেনা হয়েছে তার সম্ভাবনা $0.3 \times 0.8 = 0.24$
- 1 কেনা হয়েছে কিন্তু 2 কেনা হয় নাই তার সম্ভাবনা $0.3 \times 0.2 = 0.06$
- 1 কেনা হয় নাই কিন্তু 2 কেনা হয়েছে তার সম্ভাবনা $0.7 \times 0.8 = 0.56$
- দুটিই কেনা হয় নাই তার সম্ভাবনা $0.7 \times 0.2 = 0.14$

এখন মনে কর তোমাকে বলা হলো যে r = 1 টি জিনিস কেনা হয়েছে। এর মানে মাঝের দুইটি ঘটনার কোনো একটি ঘটেছে। আমাদের মোট ঘটনার space চারটি থেকে কমে দুটি তে নেমে গেছে। আগে চারটির যোগফল ছিল 1 এখন মাঝের দুটির যোগফল $0.06 + 0.56 = 0.62$ এটি হল আমাদের ঘটনার space. এখন এই space এ দ্বিতীয়টি (অর্থাৎ 1 কেনা হয়েছে কিন্তু 2 কেনা হয় নাই) ঘটার সম্ভাবনা $0.06/0.62$ আর তৃতীয়টি ঘটার সম্ভাবনা (অর্থাৎ 1 কেনা হয় নাই কিন্তু 2 কেনা হয়েছে) $0.56/0.62$. তাহলে 1 কেনার সম্ভাবনা $6/62$ আর 2 কেনার সম্ভাবনা $56/62$. বুঝতে পারছ তো কীভাবে হিসাব করলাম? প্রথমে বের করলাম আমাদের দেওয়া condition ঘটার সম্ভাবনা কত। সেটাই হলো আমাদের এখনকার space. এবার এই space এর প্রতিটি ঘটনা ঘটার সম্ভাবনা বের করেছি। চাইলে জিনিসটাকে কিছুটা weighted হিসাব নিকাশ মনে করতে পারো। আমাদের দুটি ঘটনা ঘটার সম্ভাবনা সমান না, এরা হলো 0.56 আর 0.06। সুতরাং তাদের প্রতিটি ঘটার সম্ভাবনা হলো $\frac{0.56}{0.56+0.06}$ আর $\frac{0.06}{0.56+0.06}$.

এখন তো বলবে- r এর মান 1 বলেই এতো সহজে হয়েছে। n = 4 আর r = 2 এর জন্য করে দেখাও না, মাথার ঘাম পায়ে পড়বে! হ্যাঁ হয়তো হাতে হাতে পুরোটা করতে অনেক কষ্ট হবে কিন্তু তুমি যদি মূল আইডিয়াটা বুঝে থাকো তাহলে আর পুরোটা হাতে হাতে করার দরকার নেই। মনে কর তুমি r টি জিনিস যত ভাবে কেনা যায় তার প্রতিটিকে লিস্ট করলে এবং তাদের সম্ভাব্যতা বের করলে। তাহলে আমাদের space হবে এই সব সম্ভাব্যতাগুলির যোগফল (S)। আর যদি জিজ্ঞাসা করা হয়

1 কেনার সম্ভাবনা কত? তাহলে এই লিস্টের দেখো কোন কোনটায় 1 আছে। তাদের সম্ভাব্যতার যোগফল বের করে তাকে S দিয়ে ভাগ করলেই হয়ে যাবে। তাই না? যেহেতু N এর মান মাত্র 20 হতে পারে সেহেতু $\binom{N}{r}$ আসলে খুব একটা বড় না। তোমরা চাইলে backtrack করে N টি জিনিস থেকে r জিনিস নির্বাচন করতে পারো। অথবা চাইলে bitmask এর লুপ চালিয়েও করতে পারো। এই ব্যাপারে আমরা প্রথম অধ্যায়েই আলোচনা করেছি। সুতরাং এখানে এই নিয়ে বলার দরকার নেই। সুতরাং তোমরা তোমাদের পছন্দের মেথড ব্যবহার করে এই সমস্যা সমাধান করে ফেল।

UVa 580 Critical Mass

সমস্যা: তোমাকে N দেওয়া থাকবে ($N \leq 30$). বলতে হবে কতগুলি N দৈর্ঘ্যের শব্দ আছে যা U আর L দিয়ে তৈরি এবং শব্দটির কোথাও না কোথাও পাশাপাশি 3 টি U আছে।

সমাধান: কোথাও পাশাপাশি তিনটি U আছে বের করার চেয়ে পাশাপাশি কোথাও তিনটি U নেই বের করা সহজ। কেন? কারণ পাশাপাশি তিনটি U "কোনো না কোনো জায়গায়" থাকতে হবে এটা হিসাবের থেকে "কোথাও" থাকবে না হিসাব করা সহজ। তুমি মনে কর একে একে character গুলি বসাচ্ছ। পাশাপাশি তিনটি U আছে কিনা এটা জানার জন্য তোমার আগের দুটি character তো জানতে হবেই সেই সাথে জানতে হবে ইতমধ্যে এর আগে কোথাও পাশাপাশি তিনটি U ছিল কিনা। কিন্তু যদি আমাদের সমস্যা হয় পাশাপাশি তিনটি U বসবে না, তাহলে শুধু আগের দুটি character জানলেই চলবে, কারণ আমরা কখনই পাশাপাশি তিনটি U বসাব না। সুতরাং আমরা যদি বের করতে পারি যে N দৈর্ঘ্যের কতগুলি শব্দে পাশাপাশি তিনটি U নেই তাকে আমরা যদি 2^N হতে বাদ দেই তাহলেই আমাদের উত্তর পেয়ে যাব।

এখন প্রশ্ন হলে কীভাবে বের করবে কতগুলি শব্দ আছে যাতে পাশাপাশি তিনটি U নেই। এটি একটি counting সমস্যা এবং counting সমস্যায় DP হর হামেশায় ব্যবহার হয়ে থাকে। মনে কর $dp[n]$ হল n দৈর্ঘ্যের valid শব্দের সংখ্যা। এখন একটি n দৈর্ঘ্যের valid শব্দ তিন উপায়ে তৈরি হতে পারে-

- শুরুতে L. সেক্ষেত্রে আমরা বাকি $n-1$ দৈর্ঘ্য $dp[n-1]$ ভাবে বানাতে পারি।
- শুরুতে UL. সেক্ষেত্রে আমরা বাকি $n-2$ দৈর্ঘ্য $dp[n-2]$ ভাবে বানাতে পারি।
- শুরুতে UUL. সেক্ষেত্রে আমরা বাকি $n-3$ দৈর্ঘ্য $dp[n-3]$ ভাবে বানাতে পারি।

আমরা কিন্তু শুরুতে UUU বসাতে পারব না। সুতরাং আমাদের recurrence হবে $dp[n] = dp[n-1] + dp[n-2] + dp[n-3]$. Base case আশা করি নিজেরা বের করে নিতে পারবে। Base case আসলে তেমন কিছু না। কোন কোন n এর জন্য আমরা আমাদের ফর্মুলা ব্যবহার করতে পারব না তাই base case. খেয়াল কর যে n এর মান যদি 1, 2 বা 3 হয় তাহলে আমরা আমাদের ফর্মুলায় negative সংখ্যা বা 0 এর জন্য dp জানতে চাচ্ছি। কিন্তু আমরা তাদের মান জানি না (হয়তো 0 এর জন্য চাইলে বের করা যায়)। সুতরাং 1, 2 এবং 3 কে base case মনে করতে পার, এবং তাদের মান কাগজে কলমে বের করে ফেল। এরপর এই ফর্মুলা ব্যবহার করে বাকি n এর জন্য $dp[n]$ বের করে ফেল।

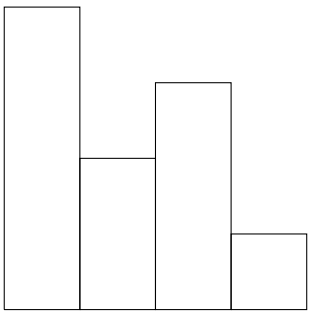
UVa 12034 Race

সমস্যা: একটি প্রতিযোগিতায় n টি ঘোড়া অংশ নিচ্ছে ($N \leq 1000$). বলতে হবে কতভাবে ঘোড়-দৌড়ের ফলাফল হবে। এই সমস্যার ক্ষেত্রে একাধিক ঘোড়া একই স্থান দখল করতে পারে। একটি উদাহরণ দিয়ে ব্যাপারটা বুঝা যাক। মনে কর দুটি ঘোড়া অংশ নিচ্ছে। মোট তিন রকমের ফলাফল হতে পারে। এক- দুটি ঘোড়াই প্রথম, দুই- প্রথম ঘোড়া প্রথম আর দ্বিতীয় ঘোড়া দ্বিতীয়, তিন- প্রথম ঘোড়া দ্বিতীয় আর দ্বিতীয় ঘোড়া প্রথম। যেহেতু উত্তর অনেক বড় হতে পারে তোমাকে উত্তর $\mod 10056$ এ দিতে হবে।

সমাধান: বুঝাই যাচ্ছে counting dp. ধরা যাক $dp[n]$ হলো n টি ঘোড়ার প্রতিযোগিতায় কত ভাবে ফলাফল হতে পারে। এখন একটি i এর লুপ চালাও যে এই প্রতিযোগিতায় কত জন প্রথম হয়েছে। এই i টি ঘোড়া আবার $\binom{n}{i}$ ভাবে নির্বাচন করা যায়। এখন বাকি $n-i$ টি ঘোড়ার প্রতিযোগিতার ফলাফল কত রকম হতে পারে? খুব সহজ $dp[n-i]$ রকম। তাহলে আমরা যা করব তাহলো n এর একটি লুপ চালাবো 1 হতে 1000 পর্যন্ত $dp[n]$ বের করার জন্য। এর ভেতরে i এর একটি লুপ চালাবো কত জন প্রথম হয়েছে তার জন্য। এর পর একে একে $\binom{n}{i}dp[n-i]$ যোগ করলেই আমরা $dp[n]$ পেয়ে যাব। এখানে আমাদের $\binom{n}{i}$ জানার দরকার। আসলে শুরুতেই সব $\binom{i}{j}$ বের করে রাখতে পারো। এজন্য $\binom{n}{i} = \binom{n-1}{i-1} + \binom{n-1}{i}$ এই recurrance ব্যবহার করতে পারো। তাহলে $\mathrm{O}(n^2)$ সময়ে সব কিছু generate হয়ে যাবে।

UVa 1638 Pole Arrangement

সমস্যা: n টি building আছে ($n \leq 20$) যাদের উচ্চতা যথাক্রমে $1, 2, \ldots n$. এই building গুলি যেকোনো order এ থাকতে পারে। যেমন- 4, 2, 3, 1 একটি উদাহরন (চিত্র ৪.২)। এখন তুমি যদি বাম দিক থেকে দেখো তাহলে কিন্তু শুধু 4 উচ্চতার building দেখতে পারবে। যদি ডান দিক থেকে দেখো তাহলে তিনটি building দেখতে পারবে যাদের উচ্চতা 1, 3, 4. তুমি কিন্তু 2 উচ্চতার building দেখতে পারবে না কারণ সেটি 3 উচ্চতার building দিয়ে ঢেকে গেছে। এখন তোমাকে বলা আছে বাম দিক হতে কয়টি building দেখা যায় (L) আর ডান দিক হতে কয়টি দেখা যায় (R). বলতে হবে n টি building কে কত ভাবে সাজান যায়।



নকশা ৪.২: UVa 1638

সমাধান: আরও একটি coutning dp সমস্যা। হয়তো তুমি চিন্তা করবে যে কোন order এ building গুলি বসাবে। তুমি প্রথমে প্রথম স্থানে building বসাবে, এর পর দ্বিতীয় স্থানে। কিন্তু তা করলে কোন কোন building বসিয়েছ এবং কোন order এ বসিয়েছ তা মনে রাখতে হয়। যেহেতু $n \leq 20$ তাই কোন order এ কোন কোন building বসিয়েছ তা মনে রাখা খুব বেশি হয়ে যায়।

আরেকটি আইডিয়া হতে পারে যে- building গুলির height এর order এ বসান। হয়তো ভাবতে পারো 1 হতে i building গুলি বসানোর পর আমরা যদি বাম থেকে কয়টি building দেখা যায় আর ডান থেকে কয়টি দেখা যায় এই দুইটি সংখ্যা state এ রাখি তাহলেই হবে। কিন্তু আমার মনে হয় তাতেও হবে না। কারণ যেসব building দেখা যায় না তাদের অবস্থানও একটি গুরুত্বপূর্ণ ব্যাপার। মনে কর 20, 10, 30 আর 20, 30. এখন মনে কর 20 আর 30 এর মাঝে 40 বসাবে। প্রথম ক্ষেত্রে সেই বসানোর জায়গা কিন্তু দুটি, কিন্তু দ্বিতীয় ক্ষেত্রে একটি। সুতরাং দেখা যায় না এরকম building এরও কিন্তু ভুমিকা আছে। তাহলে কি করা যায়?

একটু যদি চিন্তা কর তাহলে বুঝবে যে বাম বা ডান দিক হতে সবচেয়ে দূরে বা সবচেয়ে বড় যেই building দেখা যাবে সেটা হলো n. তাহলে আমরা আমাদের সমস্যাকে একটু সোজা করতে পারি আর তাহলো বাম হতে n পর্যন্ত L টি আর ডান হতে n পর্যন্ত R টি building দেখা যায় কত ভাবে। আরেকটা জিনিস খেয়াল কর n এর বাম দিকে কয়টি building আছে সেটিই আসল ব্যপার কোন কোন

height এর আছে তা কিন্তু ব্যাপার না। মনে কর n এর বামে x টি building আছে। আমরা চাইলে সেসব building এর height কে 1 হতে x এ map করতে পারি। অর্থাৎ সবচেয়ে ছোট building 1, এর পরের বড়টি 2 এরকম করে সবচেয়ে বড়টি x. তাহলে আমরা আসলে আমাদের সমস্যাকে দুইটি সহজ সমস্যায় ভাগ করে ফেলছি। মনে হয় এটুকু কথা এখনো অনেকের কাছে অপরিষ্কার লাগছে। পরিষ্কার করে বলা যাক।

আমরা একটি লুপ চালিয়ে ঠিক করব n এর বামে মোট কয়টি building আছে। ধরা যাক i টি। আমরা কিছুক্ষণ আগে বলেছি এই i টি building এর height কোনো ব্যাপার না, কারণ এদের পরই আছে n উচ্চতার building. তাই আমরা $\binom{n-1}{i}$ ভাবে এই i টি building নির্বাচন করতে পারি। আমরা চাইলে এই i টি building কে 1 হতে i height এ map করতে পারি। আর বাকি $n-1-i$ টি building কে ডানে রাখতে পারি আর তাদের একই ভাবে mapping করতে পারি (1 হতে $n-1-i$ পর্যন্ত)। তাহলে আমাদের সমস্যা দাঁড়ায় i টি building কে কতভাবে সাজান যায় যেন বাম হতে দেখলে $L-1$ টি building দেখা যায় (L তমটি হবে n height এর)। একই ভাবে ডান দিকে $n-1-i$ টি building কে কত ভাবে সাজান যায় যেন ডান হতে দেখলে $R-1$ টি building দেখা যায়। আমরা এই দুটি count কে গুন করে i এর জন্য উত্তর পাবো। এভাবে i এর লুপের প্রতি ক্ষেত্রের জন্য উত্তর যোগ করলে আমরা আমাদের আসল উত্তর পেয়ে যাব।

তাহলে আমাদের সমস্যা reduce হয়ে দাঁড়াল 1 হতে n পর্যন্ত n টি building কে (তাদের height 1 হতে n) কত ভাবে সাজালে বাম হতে দেখলে X টি building দেখা যাবে। এটার সমাধান কিন্তু দুই দিক হতে দেখার মত করেই হবে। প্রথমে n তম building কে বসাও। আর আরেকটি লুপ চালাও এর বামে কয়টি আর ডানে কয়টি building (ncr করতে হবে)। ধরা যাক বামে i টি building. তাহলে তোমাকে বের করতে হবে i টি building ব্যবহার করে X - 1 টি building দেখা যাবে এভাবে কত ভাবে building গুলি সাজান যাবে। এর মানে আমাদের dp এর state হবে (n, X). এর ভেতরে চলবে i এর একটি লুপ। এবং সেটি $(i, X-1)$ কে call করবে। আর বাকি $n-i$ টি building কে ডান দিকে তো ইচ্ছা খুশি মত বসান যাবে কারণ আমরা তো তাদের বাম দিক হতে দেখতেই পারব না।

UVa 1639 Candy

সমস্যা: দুটি বাক্সের প্রতিটিতে n টি করে চকলেট আছে ($n \leq 200,000$). তোমার বন্ধু p সম্ভাব্যতায় প্রথম বাক্স নির্বাচন করে, সুতরাং $1-p$ সম্ভাব্যতায় দ্বিতীয় বাক্স নির্বাচন করে। সে যে বাক্স নির্বাচন করে সেখান থেকে একটি চকলেট খায়। একদিন হলো কি সে তার নির্বাচিত বাক্স খুলে দেখে সেখানে কোনো চকলেট নেই। বলতে হবে ওপর বাক্সে চকলেটের expected পরিমাণ কত।

সমাধান: আমরা একটি লুপ চালাবো অন্য বাক্সে কয়টি চকলেট বাকি থাকবে তার উপর। ধরা যাক i পরিমাণ বাকি থাকে। এর মানে এমন একটি পর্যায় আসবে যখন একটি বাক্স (a) থেকে n টি চকলেট খাওয়া হয়ে যাবে, ওপর বাক্স (b) হতে $n-i$ টি খাওয়া হয়ে যাবে এবং এই সময় এসে আবার n টি চকলেট খাওয়া a বাক্স হতে খাইতে চাইবে। তার মানে এই পর্যায়ে এসে a বাক্স থেকে n টি খাওয়া হয়ে যাবে এবং b বাক্স থেকে $n-i$ টি খাওয়া হয়ে যাবে। এই খাওয়ার order মোট $\binom{n+n-i}{n}$ ভাবে হতে পারে। এখন শুধু খাওয়ার order কত রকম হবে সেটা বের করলেই তো হবে না, তাদের probability ও বের করতে হবে। a বাক্স হতে n টি খাওয়ার probability হলো p^n, b বাক্স হতে $n-i$ টি খাওয়ার probability হলো $(1-p)^{n-i}$. সেই সাথে সবশেষেও আবার a বাক্স হতে খেটে চাইবে, অর্থাৎ আরও একটি p. তাহলে মোট probability হবে $\binom{2n-i}{n} p^{n+1} \times (1-p)^{(n-i)}$. একই ভাবে b বাক্স হতে n টি খাওয়ার ঘটনা আলাদা ভাবে হিসাব করতে হবে। এখন কথা হলো এই সংখ্যা কীভাবে বের করবা? n এর বিশাল মানের জন্য $\binom{n}{i}$ এর মান কিন্তু অনেক বড় হবে, আবার p^n এর মানও কিন্তু অনেক ছোট হবে। সুতরাং সরাসরি হিসাব করলে মনে হয় উত্তর সঠিক আসবে না। এই ক্ষেত্রে logarithm খাটাতে হয়। আমরা আমাদের সূত্র সরাসরি না ব্যবহার করে যা করব তা হল তার logarithm বের করব- $\log(2n-i)! - \log n! - \log(n-i)! + (n+1)\log p + (n-i)\log(1-p)$. আর এই হিসাবের পর তার exponent নেব (e এর power). এভাবে আমরা বের করছি কত probability তে b বাক্সে (দ্বিতীয় বাক্সে) i থাকবে। এভাবে সব i এর জন্য probability বের করে তাকে i দিয়ে গুন

করে এদের সবাইকে যোগ করলে আমরা expectation পেয়ে যাব। একটা জিনিস, $\log n!$ এর মান কীভাবে বের করবা? সহজ, $\log n! = \log 1 \times 2 \times \ldots \times n = \log 1 + \log 2 + \ldots + \log n$. সুতরাং আমরা শুরুতেই সকল দরকারি n এর জন্য $\log n!$ এর মান বের করে রাখতে পারি।

UVa 10288 Coupons

সমস্যা: মোট n ধরনের কুপন আছে ($n \leq 33$) আর অনেক গুলি বাক্স আছে। তুমি যদি একটা বাক্স খুল তাহলে তার মাঝে একটি random কুপন পাবে। কুপনটি n ধরনের মাঝে যেকোনো ধরনের হতে পারে। তোমাকে বলতে হবে expected কতগুলি বাক্স খুললে তুমি সব ধরনের কুপন পাবে।

সমাধান: আমার অন্যতম পছন্দের expected value এর সমস্যা। কারণ সাধারণত আমরা expectation সমস্যাকে approach করি $\sum iP(i)$ এর মত করে। কিন্তু এই সমস্যাটি সমাধান করতে হয় একটু অন্য ভাবে। মনে কর আমাদের কাছে i টি ভিন্ন ধরনের কুপন আছে। তাহলে expected কয়টি বাক্স খুললে আমরা নতুন ধরনের কুপন পাবো? যেহেতু আমাদের কাছে ইতোমধ্যেই i ধরনের কুপন আছে এর মানে আমাদের কাছে $n - i$ ধরনের কুপন নেই। একটি বাক্স খুললে এই $n - i$ ধরনের কোনো একটি হবার সম্ভাবনা $p = \frac{n-i}{n}$. তাহলে আমাদের expected $1/p$ টি বাক্স খুলতে হবে নতুন কুপন পাবার জন্য। এভাবে আমরা $i+1$ টি ভিন্ন ধরনের কুপন পাবার জন্য expected বাক্স খুলার পরিমাণ পেয়ে যাব। আর একে একে আমরা $i + 2, i + 3 \ldots n$ পর্যন্ত উত্তর বের করে ফেলতে পারব।

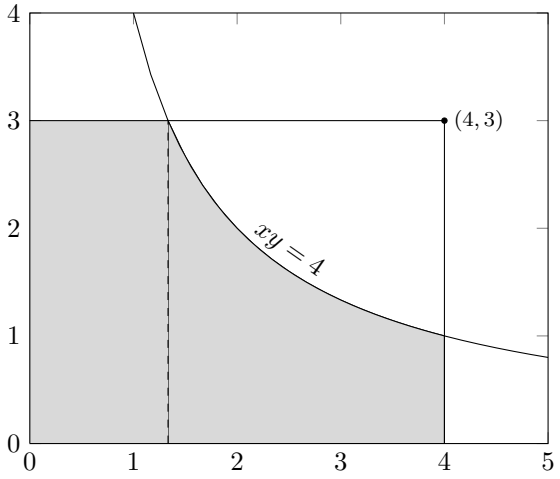
অনেকে হয়তো ভাবতে পারো নতুন কুপন পাবার probability p হলে expected বাক্স খুলার পরিমাণ $1/p$ কেন? এটা খুব একটা কঠিন না। নতুন কুপন পেতে j টি বাক্স খুলার দরকার মানে প্রথম $j-1$ বারে নতুন কুপন উঠে নাই, যার সম্ভাবনা $(1-p)^{(j-1)}$ আর j তম বারে নতুন কুপন পাওয়া গেছে, যার সম্ভাবনা p. সুতরাং j টি বাক্স খুলার পর নতুন কুপন পাবার সম্ভাবনা (এবং j টি বাক্স খুলার আগে কিন্তু নতুন কুপন পাবা না) $P(j) = p \times (1-p)^{(j-1)}$. সুতরাং আমাদের expected value হবে $\sum j \times P(j) = \sum j \times p \times (1-p)^{(j-1)}$. তোমরা চাইলে এই series sum বের করতে পারো অথবা একটু calculus খাটাতে পারো। আমরা জানি $1/(1-p) = \sum p^j$. আমাদের সমীকরণটা প্রায় একই ধরনের। প্রথমত আমাদের মূল সমীকরণে $(1-p)$ এর power নেওয়া হচ্ছিল। আমরা আমাদের series sum এ p এর বদলে $1-p$ বসাই। $1/p = \sum (1-p)^j$. এই সমীকরণকে তুমি যদি p এর সাপেক্ষে differentiate কর তাহলে পাবা $-1/p^2 = \sum -j \times (1-p)^{(j-1)}$. আর একটু এদিক অদিক করলেই $1/p = \sum j \times p \times (1-p)^{(i-1)}$ পেয়ে যাবা। তবে এটা মনে হয় একটু কঠিন হয়ে গেল। আরেকটা উপায়ে দেখানো যাক। মনে কর আমাদের expected value হবে E. এখন p সম্ভাব্যতায় প্রথম বারেই আমরা নতুন কুপন পেতে পারি $1 \times p$। অথবা $1-p$ সম্ভাব্যতায় প্রথম বার নতুন কুপন উঠে নায়। তাহলে এর পরে নতুন কুপন পেতে তোমাকে E টি বাক্স খুলতে হবে (এখন যে একটি খুললে সেটি বাদে) অর্থাৎ $(1-p)(1+E)$। সুতরাং আমাদের সূত্র দাঁড়াল $E = p + (1-p)(1+E)$ আর এটাকে সমাধান করলে দাঁড়াবে $E = 1/p$.

UVa 11346 Probability

সমস্যা: $(-a, -b)$ হতে (a, b) পর্যন্ত rectangle টি একটি special জায়গা। তোমাকে এই special জায়গার মাঝে একটি (x, y) বিন্দু randomly নির্বাচন করতে হবে (x আর y কে যে পূর্ণ সংখ্যা হতে হবে এমন কথা নেই) যেন $(0, 0)$ ও (x, y) বিন্দু দিয়ে তৈরি rectangle টির ক্ষেত্রফল S এর হতে বড় হয়। S, a, b এর মান তোমাকে ইনপুট হিসাবে দেওয়া থাকবে। বলতে হবে সেরকম একটি বিন্দু নির্বাচনের সম্ভাবনা কত।

সমাধান: যারা ইতোমধ্যেই কলেজে calculus পড়েছ তারা জানো যে এসব সমস্যার ক্ষেত্রে আমরা একটি quadrant এ হিসাব নিকাশ করি কারণ অন্য সব quadrant এ সব হিসাব নিকাশ symmetric হয়। এখানে যেমন আমাদের special জায়গা এবং (x, y) এর সাথে মূল বিন্দুর rectangle সবই symmetric. সুতরাং আমরা শুধু first quadrant এ হিসাব নিকাশ করব।

তাহলে আমরা শুধু $(0,0)$ হতে (a,b) এইটুকু special জায়গা নিয়ে মাথা ব্যাথা করব। কথা হল এর মাঝে কোন টুকু জায়গা থেকে (x,y) নির্বাচন করলে মূলবিন্দুর সাথে ক্ষেত্রফল S এর বড় হবে। একটু চিন্তা করে দেখো, $xy = S$ রেখার বিন্দু গুলির সাথে মূল বিন্দু ঠিক ঠিক S ক্ষেত্রফল তৈরি করে। এর উপরের (up) বিন্দুগুলির সাথে S এর থেকে বেশি এবং এর নিচের বিন্দুগুলির সাথে S এর থেকে কম ক্ষেত্রফলের আয়তক্ষেত্র তৈরি করে। তাহলে আমাদের বের করতে হবে $xy = S$ এই রেখার নিচে special জায়গার ক্ষেত্রফল। এই পরিমান যদি আমরা ab হতে বিয়োগ করি তাহলে আমরা পাব সেই অংশের ক্ষেত্রফল যা হতে (x,y) নির্বাচন করলে আয়তক্ষেত্রের ক্ষেত্রফল S হতে বড় হয়। এই ক্ষেত্রফলকে যদি আমরা ab দিয়ে ভাগ করি তাহলে আমরা আমাদের উত্তর পেয়ে যাব। এখন কথা হল এই $xy = S$ রেখার নিচের অংশের ক্ষেত্রফল কীভাবে বের করব।



নকশা ৪.৩: $(a,b) = (4,3)$ এবং $S = 4$

আমরা একটা উদাহরণ নিয়ে কাজ করি তাহলে মনে হয় সুবিধা হবে। ধরা যাক $(a,b) = (4,3)$ আর $S = 4$. তাহলে first quadrant এর যেই চিত্র হয় তা চিত্র ৪.৩ এ দেওয়া হলো। $xy = S = 4$ এটি একটি বাঁকা রেখা দেখতেই পাচ্ছ। আর সেই সাথে আমরা $(0,0)$ হতে $(4,3)$ এই আয়তক্ষেত্রও এঁকেছি। আমাদেরকে চিত্রের shaded জায়গার ক্ষেত্রফল বের করতে হবে। কারণ এই জায়গাতে যদি আমরা কোনো (x,y) নির্বাচন করি তাহলে মূল বিন্দুর সাপেক্ষে তার আয়তক্ষেত্রের ক্ষেত্রফল S অপেক্ষা কম হবে। দেখলেই বুঝতে পারছ shaded জায়গাকে দুই অংশে ভাগ করতে পারো। একটা হলো ছোট আয়তক্ষেত্র আরেকটা হলো $xy = 4$ এর নিচের অংশ। ছোট আয়তক্ষেত্রের উপরের ডান কোণা আছে $(S/b, b) = (4/3, 3)$ তে। যার ক্ষেত্রফল $S/b \times b = 4/3 \times 3 = 4 = S$. বাকি থাকে $S/b = 4/3$ হতে $a = 4$ পর্যন্ত $xy = 4$ এর নিচের অংশের ক্ষেত্রফল। এটি একটি সহজ integration $\int_{S/b}^{a} \frac{S}{x} dx = S \ln x \Big|_{S/b}^{a} = S(\ln(a) - \ln(S/b)) = S \ln(\frac{ab}{S})$[১]. তোমরা চাইলে a বা b এর মান বসিয়ে এই মান বের করতে পারো। তাহলে আমরা অন্য অংশের ক্ষেত্রফল বের করলাম যেখানের বিন্দুগুলির সাথে মূলবিন্দুর আয়তক্ষেত্রের ক্ষেত্রফল S এর কম। আমরা যদি ab হতে এই ক্ষেত্রফল বাদ দেই তাহলে $(0,0)$ হতে (a,b) পর্যন্ত সেই অংশের ক্ষেত্রফল পাবো যেখানের বিন্দুগুলির সাথে মূলবিন্দুর আয়তক্ষেত্রের ক্ষেত্রফল S এর বেশি। এবং সবশেষে একে যদি আমরা ab দিয়ে ভাগ করি তাহলে সেই বিন্দু নির্বাচনের probability পেয়ে যাব। আর একটু যদি সাবধান থাক তাহলে একটা special case হ্যান্ডেল যে করতে হবে তা ইতমধ্যেই বুঝে গেছ। Case টা হলো যদি $(0,b)$ হতে (a,b) এই লাইন $xy = S$ কে ছেদ না করে, অর্থাৎ $S/b > b$.

[১] $\ln(a) - \ln(b) = \ln(a/b)$

UVa 10900 So you want to be a 2^n-aire?

সমস্যা: একটি game show তে শুরুতে তোমাকে 1 টাকা দেওয়া হবে। এরপর একে একে তুমি n রাউন্ড খেলতে পারবে ($n \leq 30$)। প্রতিটি রাউন্ডের শুরুতেই তোমাকে একটি প্রশ্ন করা হবে। প্রশ্ন পারার সম্ভাবনা t হতে 1 এর মাঝে কোনো একটি random সংখ্যা ($0 \leq t \leq 1$). তুমি চাইলে প্রশ্নটির উত্তর না দিয়েই quit করতে পারো। quit করলে তোমার কাছে যত টাকা আছে তা তোমার হয়ে যাবে। আর যদি quit না কর তোমাকে প্রশ্নের উত্তর দিতে হবে। তুমি যদি প্রশ্নের উত্তর দিতে পারো তাহলে তোমার টাকার পরিমাণ দ্বিগুণ হয়ে যাবে এবং তুমি পরের রাউন্ডে যাবে। আর যদি ভুল হয় তাহলে তোমার টাকার পরিমাণ 0 হয়ে যাবে এবং খেলা শেষ হয়ে যাবে। তুমি তোমার expected টাকার পরিমাণ maximize করতে চাও। বলতে হবে এই expected টাকার পরিমাণ কত।

সমাধান: আমরা dp এর মত করব। আমরা বের করব যদি আমরা i তম রাউন্ডে থাকি তাহলে expected টাকার পরিমাণ কত। আমাদের base case কি হবে? মনে কর আমরা n তম রাউন্ডে আছি, তখন যদি আমরা ঠিক প্রশ্নের উত্তর দিতাম তাহলে $n+1$ রাউন্ডে উঠতাম। এখানে আর কোনো প্রশ্ন নেই। এখানে আমাদের খেলা শেষ করতেই হবে আর তাহলে আমরা 2^n টাকা পাবো। সুতরাং আমাদের base case হবে $f(n+1) = 2^n$. এখন recurrence এর পালা। প্রশ্ন হল i তম রাউন্ডে expected টাকার পরিমাণ কত। আমরা জানি আমাদের প্রশ্নের উত্তর পারার সম্ভাবনা t হতে 1 এর মাঝে যেকোনো একটি সংখ্যা। ধরা যাক p. তোমার হাতে দুইটি অপশন। এক- আমরা quit করব তাহলে আমরা 2^{i-1} টাকা পাবো। দুই- প্রশ্নের উত্তর দিব। যদি প্রশ্নের উত্তর দাও তাহলে কত টাকা পাবা? p সম্ভাব্যতায় $f(i+1)$ টাকা পাবা আর (1-p) সম্ভাব্যতায় 0 টাকা। তাহলে প্রশ্নের উত্তর দিলে expected টাকার পরিমাণ হবে $p \times f(i+1)$. এখন তুমি চাও তোমার টাকার পরিমাণ maximize করতে। তাহলে কি করবে? প্রশ্নের উত্তর দেবে নাকি দেবে না? সহজ উত্তর- যা করলে expected টাকার পরিমাণ বেশি হবে সেটা করবা। অর্থাৎ $max(2^{i-1}, p \times f(i+1))$. এটি হলো p এর জন্য expected টাকার পরিমাণ। কিন্তু p এর মান তো t হতে 1 পর্যন্ত হতে পারে। তাহলে? তাহলে যা করতে হবে তাহলো তুমি t হতে 1 পর্যন্ত সব মানের জন্য expected টাকার পরিমাণ যোগ কর এবং সেই সাথে t হতে 1 পর্যন্ত যতগুলি সংখ্যা আছে তা দিয়ে ভাগ কর। কিন্তু t হতে 1 পর্যন্ত তো অসংখ্য সংখ্যা আছে। কি করবা? উপায় হলো integration- $\frac{1}{1-t}\int_t^1 max(2^{i-1}, pf(i+1))dp$. কেন $(1-t)$ দিয়ে ভাগ হলো? কারণ তুমি t হতে 1 পর্যন্ত integrate করেছ। এটা কিছুটা গড় নেবার মত।

এখন কথা হলো এই integration করবা কীভাবে? $f(i+1)$ একটি constant কারণ এটি p এর উপর নির্ভর করে না। ধরে নেই $C = f(i+1)$. তাহলে আমরা আমাদের integration কে একটু সহজ করে লিখতে পারি $\frac{1}{1-t}\int_t^1 max(2^{i-1}, pC)dp$. যদি $pC < 2^{i-1} \Rightarrow p < 2^{i-1}/C$ হয় তাহলে আমাদের max এর মান হবে 2^{i-1} আর নাহলে হবে pC. তাহলে আমরা আমাদের integral কে দুই অংশে ভাগ করতে পারি t হতে $2^{i-1}/C$ আর $2^{i-1}/C$ হতে 1. শুধু একটা জিনিস খেয়াল রেখো যে $2^{i-1}/C$ যদি t এর থেকে ছোট হয় বা 1 এর থেকে বড় হয় তাহলে যেন তুমি উলটা পালটা কিছু না কর।

তাহলে এই সমস্যার মাধ্যমে তোমরা দেখলে কীভাবে expectation এর সমস্যাতে আমরা calculus ব্যবহার করতে পারি। আর Integration এর ফাংশন যদি max ওয়ালা হয় তাহলে আমরা কীভাবে তাকে ভেঙ্গে লিখে সমাধান করতে পারি।

UVa 11971 Polygon

সমস্যা: তোমাকে একটি N দৈর্ঘ্যের একটি সরলরেখা দেওয়া আছে। তুমি একে randomly K টি জায়গায় কাটলে ($1 \leq N \leq 10^6, 1 \leq K \leq 50$)। এর ফলে যে $K+1$ টি সেগমেন্ট তৈরি হলো তা দিয়ে একটি polygon গঠন করতে পারার probability কত?

সমাধান: নিঃসন্দেহে এটি একটি কঠিন সমস্যা। সত্যি কথা বলতে এটি পড়ার সাথে সাথে আমি কোনো idea পাই নাই। যদি pattern খোঁজার মাধ্যমে সমাধান করতে চাও তাহলে বেশ সহজেই

সমাধান হয়ে যাবে। তবে যদি নিজে থেকে গাণিতিক ভাবে সমাধান করতে চাও তাহলে প্রসেসটা খুব একটা সহজ হবে না। নেটে খুঁজে chinese site হতে আমি অন্তত 4 টি ভিন্ন ধরনের সমাধান দেখেছি। এ থেকে দুটি এখানে দেখানো যাক।

যেভাবেই সমাধান কর না কেন মূল আইডিয়া হলো $K+1$ টি সেগমেন্টের মাঝে সবচেয়ে বড় সেগমেন্টের দৈর্ঘ্য $1/2$ এর থেকে কম হবে। অর্থাৎ আমাদের বের করতে হবে প্রতিটি সেগমেন্টের দৈর্ঘ্য $1/2$ এর থেকে ছোট হবার probability কত? কেন $1/2$ এর থেকে ছোট হবে? মনে কর সবচেয়ে বড় সেগমেন্টটা নিলা। এরপর বাকি সেগমেন্টগুলি জোড়া লাগিয়ে লাগিয়ে তুমি বড় সেগমেন্টের অন্য মাথায় পউছাবা। যদি বড়টার দৈর্ঘ্য $1/2$ এর থেকে বড় হয় তাহলে তুমি কখনই অন্য প্রান্তে পোঁছাতে পারবে না। আর যদি $1/2$ এর সমান হয় তাহলে কেবল অপর প্রান্তে পোঁছাবে একটি লাইন তৈরি করে, polygon হবে না। আর যদি সবচেয়ে বড়টা $1/2$ এর থেকে ছোট হয় তাহলে আমরা polygon বানাতে অবশ্যই পারব। ও হ্যাঁ ভাল কথা, প্রবলেমে বলেছি সেগমেন্টের দৈর্ঘ্য N কিন্তু এখানে আমি এই দৈর্ঘ্য 1 ধরে কাজ করেছি। কেন? কারণ, আমাদের উত্তর আসলে N এর উপর নির্ভর করবে না তাই না? একটি 1 দৈর্ঘ্যের যেখানে যেখানে কাটা valid, 100 দৈর্ঘ্যের সেখানে সেখানে কাটা valid। মানে propotionally কাটতে হবে- এই আর কি। যদি তুমি 1 এর $1/3$ তে কাট তাহলে 100 এর $100/3$ তে কাটবা- এরকম। যদি কাটাগুলি 1 এর ক্ষেত্রে valid হয় তাহলে 100 এর ক্ষেত্রেও valid হবে। সুতরাং আমাদের N এর দরকার নেই। আমরা আমাদের হিসাবকে সহজ করার জন্য ধরে নিতে পারি সেগমেন্টের দৈর্ঘ্য 1. এখন আমাদের প্রথম সমাধান দেখা যাক।

আমরা "প্রতিটি segment এর দৈর্ঘ্য $1/2$ এর থেকে ছোট হবে" এটির probability না বের করে "কোনো একটি segment এর দৈর্ঘ্য $1/2$ এর সমান বা বড় হবে" এটার probability বের করব। খেয়াল কর কোনো একটি segment যদি $1/2$ এর সমান বা বড় হয় তাহলে আর কেউ কিন্তু $1/2$ এর সমান বা বড় হবে না কারণ বাকি যেই অংশ আছে তাতে অন্তত দুটি সেগমেন্ট বসাতে হবে (আমি ধরে নিচ্ছি $K > 1$ কারণ $K = 1$ হলে দুটি সেগমেন্ট হয় আর দুটি বাহু দিয়ে কখনই polygon হয় না)। মনে কর আমরা K বার randomly কেটেছি এবং $K+1$ টি সেগমেন্ট পেয়েছি। ধরা যাক এদের দৈর্ঘ্য হলো $x_0, x_1, \ldots x_k$ এবং x_0 হল সবচেয়ে বড়টা। আমরা একে সরলরেখার একদম বাম সাইডে রাখবো। তাহলে x_0 এর অন্য মাথা অবশ্যই $[1/2, 1]$ এর মাঝে থাকবে। শুধু তাই না, আসলে যেই K টি বিন্দুতে কাটা হবে সবগুলিই $[1/2, 1]$ এর রেঞ্জে পড়বে। এই probability কত? কোনো একটি বিন্দু $[0, 1]$ এই রেঞ্জের শেষ অর্ধেকে পড়বে তার probability $1/2$. তাহলে K টি বিন্দুই শেষ অর্ধেকে পড়বে তার probability হবে K টা $1/2$ গুন অর্থাৎ $\frac{1}{2^K}$. তাহলে এটাই কি উত্তর? না, এখানে আমরা ধরে নিয়েছি যে x_0 অর্থাৎ সবচেয়ে বড় segment টি বামে থাকবে। কিন্তু তুমি যখন random ভাবে K জায়গায় কাটবে তখন তো সবচেয়ে বড়টা বামে নাও থাকতে পারে। তাহলে উপায় কি? মনে কর তুমি random ভাবে K টি জায়গায় কাটার পর segment গুলির বাম থেকে ডানে দৈর্ঘ্য গুলি হলো $y_0, y_1, \ldots y_K$. তুমি cyclically ঘুরাতে থাকো যতক্ষণ না সবচেয়ে বড়টা বামে আসে অর্থাৎ তুমি দেখবা $y_1, y_2, \ldots y_K, y_0$ এরপর $y_2, \ldots, y_K, y_0, y_1$ এরকম করে ততক্ষণ cyclically ঘুরাতে থাকবা যতক্ষণ না সবচেয়ে বড়টা বামে আসে। তাহলে এভাবে তুমি যেকোনো ভাবে কাটার জন্য ঘুরিয়ে ঘুরিয়ে বড়টিকে সবচেয়ে বামে আনতে পারবে। যদি এটা সত্যি হয় তাহলে একবার ভাব যদি শুরুতেই সবচেয়ে বড়টা বামে থাকে তাহলে তাকে ঘুরিয়ে ঘুরিয়ে নিশ্চয় বিভিন্ন জায়গায় নেওয়া সম্ভব? যেহেতু মোট $K+1$ টি সেগমেন্ট আছে তাই একে K বার ঘুরান যাবে আর নিজে একটা, মোট $K+1$ অবস্থায় নিয়ে যাওয়া সম্ভব। এর মানে আসলে কোনো একটি সেগমেন্ট $1/2$ এর থেকে বড় পাবার সম্ভাবনা $\frac{K+1}{2^K}$ কারণ বামে সেই বড়টা পাবার সম্ভাবনা $\frac{1}{2^K}$ আর তাকে $K+1$ ভাবে ঘুরিয়ে নানা জায়গায় নিয়ে যাওয়া যায় আর এই সব নানা জায়গা তোমাকে সব অবস্থা দিতে সাহায্য করবে। তাহলে আমাদের polygon বানাতে পারার probability হবে $1 - \frac{K+1}{2^K}$.

এবার দ্বিতীয় সমাধান দেখা যাক। আমরা calculus ব্যবহার করব। ধরা যাক আমাদের সবচেয়ে বড় segment টির দৈর্ঘ্য x. আমরা x এর সাপেক্ষে integration করব। যেহেতু এটি $1/2$ এর থেকে বড় হবে তাই আমাদের integration এর রেঞ্জ হবে $[0.5, 1]$. এখন আমাদের বড় segment টি এক সাইডে হতে পারে বা ভেতরে হতে পারে। ধরা যাক এটি বাম সাইডে। তাহলে বাম সাইড হতে ঠিক x দূরত্বে একটি কাটা হবে। যদি আমরা dx কে খুব ছোট জায়গা ধরি তাহলে এই x দূরত্বে একটি কাটা হবার সম্ভাবনা $\frac{dx}{1} = dx$ যেখানে 1 হল পুরো সরলরেখার দৈর্ঘ্য। এখন এই কাটটি K টি কাটের যেকোনোটিই হতে পারে। কততম কাটটি এখানে? সেজন্য একটি K গুন হবে (কত ভাবে

ওখানে কাট হতে পারে? K ভাবে। কারণ K কাটের যেকোনো কাটই ওখানে হতে পারে)। আর এই কাট ছাড়া বাকি কাটগুলি ডান দিকের বাকি $1-x$ জায়গায় হবে। সুতরাং বাকি $K-1$ কাটের প্রতিটি ডানের $(1-x)$ এ পড়ার probability $(1-x)^{K-1}$. সুতরাং বামে x দৈর্ঘ্যের কাট হবার সম্ভাবনা $K(1-x)^{K-1}dx$. একই ভাবে যদি x অংশটি যদি ডান দিকে হয় তাহলে একই probability (সুতরাং 2 দিয়ে গুন হবে). এখন ধরা যাক আমাদের বড় সেগমেন্টটি সাইডে না হয়ে মাঝে। তাহলে এই ক্ষেত্রে এই সেগমেন্ট এর বামে একটি বিন্দু পড়বে আর ডানে আরেকটি। মনে কর আমরা ডান হতে x অংশ অন্য রঙে রং করেছি। তাহলে আমাদের x এর বাম মাথা আসলে রং না করা যেকোনো জায়গা থেকে শুরু হতে পারে। রং না করা অংশের দৈর্ঘ্য $(1-x)$ তাই এই অংশে একটি কাট হবার সম্ভাবনা $(1-x)$. এখান থেকে x দূরত্ব ডানে আমাদের আরেকটি কাট হতে হবে। এর সম্ভাবনা dx. এখন খেয়াল কর K টি কাটের মাঝে x এর বাম মাথায় একটি আর ডান মাথায় আরেকটি কাট আছে। বামেরটি K টি কাটের যেকোনোটি হতে পারে আর ডানেরটি হতে পারে $(K-1)$ ভাবে (উদাহরণ স্বরূপ- তৃতীয় কাট বামে আর প্রথম কাট ডানে)। তাহলে এই দুটি কাট হতে পারে $K(K-1)$ ভাবে। সুতরাং এদের গুন করলে x এর দুই দিকে দুটি কাট হবার সম্ভাবনা বের হয়ে যাবে। বাকি থাকে বাকি $K-2$ টি কাট। এরা আসলে x বাদে বাকি জায়গার যেকোনো কোথাও হতে পারে। x বাদে বাকি অংশের দৈর্ঘ্যের যোগফল $(1-x)$ তাই এটা হতে পারে $(1-x)^{K-2}$ ভাবে। সুতরাং x মাঝে হবার সম্ভাবনা $K(K-1)(1-x)(1-x)^{K-2}dx$. তাহলে x মাঝে হবার সম্ভাবনা আর সাইডে হবার সম্ভাবনাকে যোগ করে যদি integration কর তাহলে উত্তর পেয়ে যাবে।

শুরুতেই বলেছি নিঃসন্দেহে এটি একটি কঠিন সমস্যা। এবং একবার পড়েই তুমি এটা বুঝতে পারবা আমি আশা করি না। সত্যি কথা বলতে চাইনিজ লিখাকে translate করে এর পর এই সমাধান বুঝতে আমার বেশ অনেকক্ষণ লেগেছে।

UVa 1640 The Counting Problem

সমস্যা: তোমাকে দুটি সংখ্যা a ও b দেওয়া হবে ($1 \leq a \leq b \leq 10^8$). বলতে হবে এদের মাঝে যতগুলি সংখ্যা আছে (এই দুটি সহ) তাদের সব গুলিতে কোন digit কয়বার করে আছে। অর্থাৎ 0 কয়বার, 1 কয়বার এভাবে 9 পর্যন্ত সব digit কয়বার করে আছে বলতে হবে। যদিও মূল সমস্যাতে $a \leq b$ বলা নাই কিন্তু এখানে আলোচনার সুবিধার জন্য আমি এই condition আরোপ করেছি। যদি $b < a$ হয় তাহলে শুধু তাদের swap করে নিলেই হবে।

সমাধান: এটি একটি সুন্দর সহজ mathematical সমস্যা। যেসব সমস্যাতে বলে যে a ও b এর মাঝে কিছু একটা count করতে বা হিসাব করতে তখন সাধারণত যা করলে সহজ হয় তাহলো তুমি একটি ফাংশন $f(n)$ লিখবে যেটা 1 হতে n পর্যন্ত উত্তর বের করে। এর পর তুমি $f(b)-f(a-1)$ করবা, তাহলে তুমি a হতে b পর্যন্ত উত্তর পেয়ে যাবে। আমরা এই সমস্যাতেও এই ট্রিক খাটাতে পারি।

সুতরাং আমাদের প্রবলেম দাঁড়াল, একটি মান n দেওয়া হলে তুমি কীভাবে 1 হতে n পর্যন্ত সব সংখ্যার ডিজিট গুলির সংখ্যা বের করবে। বুঝার সুবিধার জন্য আমরা একটা উদাহরণ নেই- 315097. আমরা 1 হতে 315097 পর্যন্ত সংখ্যাগুলির ডিজিটের সংখ্যা বের করতে চাই। অর্থাৎ এই সংখ্যাগুলিতে কোন ডিজিট কয়টি করে আছে। এটি একটি 6 অংকের সংখ্যা। আমরা একে একে প্রতিটি অংকে যাব এবং সেখানে 0 হতে 9 পর্যন্ত লুপ চালাবো আর জিজ্ঞাসা করব এই স্থানে এই ডিজিট 1 হতে n পর্যন্ত কয়বার আসতে পারে। মনে কর আমরা জানতে চাচ্ছি i তম ঘরে (ডানের ঘর অর্থাৎ lsb এর জন্য $i=0$, এর বামের ঘরের জন্য $i=1$ এরকম) d ডিজিটটি কত বার আসে। এই ক্ষেত্রে d এর লুপ চলবে 0 হতে 9 পর্যন্ত, আর i এর লুপ চলবে 0 হতে 5 পর্যন্ত (কারণ সংখ্যাটি 6 ডিজিটের আর আমরা 0 indexing ধরেছি)। ধরা যাক 315097 এর জন্য i = 3 ও d = 1 এর জন্য এই count আমরা বের করতে চাই। তিনটি জিনিস দেখতে হবে $i=3$ এর বামে কি আছে (L = 31)? ডানে কি আছে (R = 097)? আর নিজে কি (S = 5)? প্রথমে দেখতে হবে d আর S এর সম্পর্ক কি। ছোট, বড় না সমান। তিন ক্ষেত্রে তিন রকমভাবে গণনা করতে হবে। $d=1$ এর জন্য এটি $S=5$ হতে ছোট। এই ক্ষেত্রে কল্পনা কর কখন কখন এখানে 1 আসবে? $i=3$ এ যখন 1 আসবে তখন আমাদের বামের সংখ্যা কিন্তু 0 হতে 31 পর্যন্ত মোট 32 $(L+1)$ টি সংখ্যার যেকোনোটি হতে পারে। আবার আমাদের

ডানের সংখ্যা 000 হতে 999 পর্যন্ত মোট 1000 (10^i) রকম হতে পারে। তাহলে 1 আসবে মোট 32×1000 বার বা ফর্মুলায় লিখলে $(L+1)10^i$ বার। যদি $d > S$ হয়? তাহলে কিন্তু আমাদের বামের সংখ্যা 31 হতে পারবে না। কেন? ধর $d = 7 > S = 5$ এখানে দেখো $(31)7(???)$ এরকম কোনো সংখ্যা কিন্তু 1 হতে 315097 এর মাঝে নেই। তার মানে আমাদের বামের অংশ হতে পারে 0 হতে 30 পর্যন্ত মোট 31 রকম। আর ডানের সংখ্যা আগের মতই 1000 রকম। এ হতে আমরা বের করতে পারি 7 কয় বার আসবে। বাকি থাকে $d = 5 = S = 5$ এর case. এই ক্ষেত্রে বামে 31 আসতে পারবে কিন্তু যখন 31 আসবে তখন আবার ডানে 000 হতে 999 পর্যন্ত সবগুলি সংখ্যাই হয়তো আসতে পারবে না। কেবল 000 হতে 097 (R) পর্যন্ত আসতে পারবে। বামে 0 হতে 30 হলে কি হবে তাতো $d > S$ এর case থেকেই বের করে ফেলেছ। আর যদি বামে L এর সমান হয় তাহলে ডানে $R+1$ রকম সংখ্যা আসতে পারে। সুতরাং এই ভাবে আমরা সকল i এর স্থানে সকল d এর সংখ্যা গুনে ফেলতে পারব।

তবে কিছু জিনিসে সাবধান হতে হবে। যেমন আমাদের এই উদাহরণে $i = 5$ এ $d = 0$ কখনই হবে না। আবার যখন $i = 0$ হবে তখন R এর মান কত হবে? আসলে আমি নিজেও জানি না যে R এর মান কত ধরলে মিলবে। আমি হলে এখানে কোনো risk নেব না। বরং এখানে R ছাড়াই হিসাব করব। ধর 315097 এ শেষ ঘরে (অর্থাৎ $i = 0$ এর ক্ষেত্রে) 1 কত বার আসে? সহজ 1 যতবার আসবে এর বামে 0 হতে 31509 পর্যন্ত মোট 31510 টি সংখ্যা আসবে। আর এর ডানে যেহেতু কোনো ঘর নাই তাই R এর হিসাব করার কোনো দরকার নেই। একই ভাবে $d = S$ বা $d > S$ এ প্রসেস করতে হবে।

আবারো বলি, এই সমস্যা বলে বুঝানো কঠিন। আমি তোমাকে খুব জোড় সমাধানের পথ বলে দিতে পারব। কিন্তু তোমাকে নিজে থেকে এটা বুঝতে হবে। কারণ এতে অনেক step আছে এবং প্রতিটি step এর জন্য ফর্মুলা বের করতে হবে। আমি নিশ্চিত এর থেকে হয়তো সহজ উপায়ে হিসাব করা সম্ভব যেটা আমার মাথায় এই মুহূর্তে আসছে না। তোমরা কেউ সেরকম সমাধান পেলে সবাইকে জানিয়ে দিও।

UVa 10213 How Many Pieces of Land?

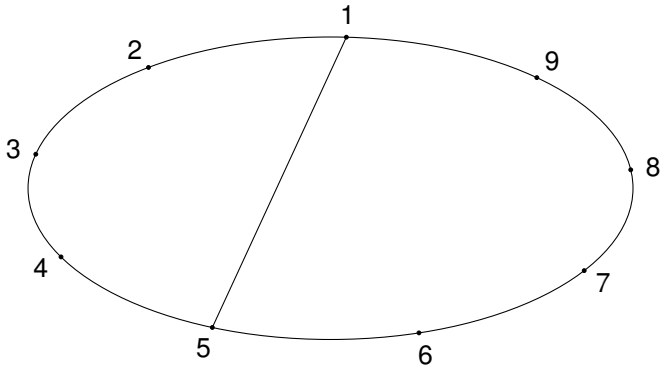
সমস্যা: একটি ellipse এর পরিধির উপর n টি বিন্দু নাও ($1 \leq n \leq 2^{31}$)। এবার প্রতি দুটি বিন্দু নিয়ে তাদের মাঝে রেখা টান। এতে করে ellipse টি অনেক ভাগে বিভক্ত হবে। বলতে হবে সর্বোচ্চ কত ভাগে ভাগ করা সম্ভব।

সমাধান: এই সমস্যা সমাধানের দুটি উপায় দেখা যাক।

প্রথম উপায় হলো এটা বিশ্বাস করা যে উত্তর একটি polynomial হবে। কত degree এর তা জানি না। তবে polynomial হবেই। এজন্য আমরা আসলে কয়েকটি n এর মানের জন্য উত্তর বের করব। ভাল হয় যদি তুমি $n = 1, 2, \ldots 6$ এরকম কতগুলি n এর জন্য যদি উত্তর বের করতে পারো। আমি জানি n এর এরকম বড় মানের জন্য উত্তর হাতে হাতে বের করা কষ্টকর কিন্তু করার কিছুই নেই। তুমি যদি একটু কষ্ট করে এই মান গুলি বের করতে পারো তাহলে সমাধানের অনেক কাছে চলে আসবে। n এর যতগুলি মানের জন্য উত্তর বের করেছ ঠিক তত ডিগ্রী এর polynomial নাও, এক্ষেত্রে যেমন 6 ডিগ্রী এর $p(n) = \sum a_i \times n^i$. যেহেতু আমরা $n = 1 \ldots 6$ এর জন্য $f(n)$ জানি সেহেতু আমরা যদি n এর মান আর $f(n)$ এর মান বসাই তাহলে 6 টি variable $(a_0, \ldots a_5)$ এর 6 টি equation পেয়ে যাব। এবার gaussian elimination ব্যবহার করে আমরা $a_0 \ldots a_5$ সবার মান বের করে ফেলতে পারব। তোমরা চাইলে Lagrange এর interpolation ফর্মুলাও ব্যবহার করতে পারো। ফর্মুলাটি আমি আর এখানে তুলে দিলাম না। কারণ আমি মনে করি এটি একটু advanced জিনিস। যদি কারো আগ্রহ থাকে তাহলে wiki দেখতে পারো। হয়তো পরে কোনো এক সময় এটা নিয়ে লিখব।

দ্বিতীয় উপায় হলো euler's formula- $V - E + F = 2$ যেখানে V হলো vertex এর সংখ্যা, E হলো edge এর সংখ্যা আর F হলো face এর সংখ্যা। Face মানে আমাদের সমস্যায় যেই "ভাগ" বলা হয়েছে সেটা। তবে খেয়াল রাখতে হবে face শুধু ভেতরের অংশ না বাহিরের অংশও নিতে পারে। দু একটা উদাহরণ দেয়া যাক তাহলেই পরিষ্কার হবে। একটি ত্রিভুজ নাও। ত্রিভুজের তিনটি vertex, তিনটি edge এবং দুইটি face একটি বাহিরে আর আরেকটি ভেতরে। তাহলে ফর্মুলা খাটে কিনা

দেখো $3-3+2=2$. সুন্দর মত খাটে। এবার একটি বর্গ আঁক আর তার দুটি কর্ণ আঁক। আমাদের vertex কিন্তু 5 টা। বর্গের চার কোণা আর কর্ণের ছেদ বিন্দু। Edge কিন্তু 8 টি। বর্গের চার বাহু আর কর্ণের ছেদ বিন্দু হতে বর্গের কোণাগুলিতে। আর face কিন্তু 5 টি। বর্গের ভেতরে 4 ভাগ আর বাহিরে এক ভাগ। তাহলে $5-8+5=2$. বুঝতেই পারছ এই ম্যাজিকাল ফর্মুলা আমাদের জীবনকে কত সহজ করে দিতে যাচ্ছে! আমাদের সমস্যায় আমরা যদি vertex আর edge এর সংখ্যা বের করতে পারি, তাহলে এই ফর্মুলা ব্যবহার করে আমরা face এর সংখ্যা খুব সহজেই বের করে ফেলতে পারব (ভুলে যেও না এই ফর্মুলা কিন্তু বাহিরের face কেও count করে)। প্রথমে তুমি চিন্তা কর কীভাবে বিন্দু গুলি বসাবা। আইডিয়া হলো এমন ভাবে বসাবা যেন এমন কোনো বিন্দু যেন না থাকে যেখানে দিয়ে 2 টির বেশি ellipse এর কর্ণ যায় (ellipse এর কর্ণ বলতে বুঝাচ্ছি আমাদের বসানো বিন্দু দের মাঝে আঁকা রেখা গুলি)। কারণ যদি কোনো বিন্দু দিয়ে তিনটি কর্ণ যায় আমরা চাইলে একটি বিন্দুকে সরিয়ে একটি ভাগ বাড়াতে পারি। তুমি একটু কাগজে কলমে একে দেখো। মনে কর একটা X এর মাঝে দিয়ে I এঁকেছ। আর আরেক জায়গায় ঠিক মাঝ দিয়ে না একে একটু সরিয়ে এঁকেছ। এই দুইভাবে আঁকা ছবির face এর সংখ্যা গুনে দেখো তাহলেই বুঝবে। অবশ্য আমাদের বসানো n টি বিন্দুতে যত খুশি কর্ণ মিলিত হোক আমাদের সমস্যা নেই। তাহলে প্রশ্ন হল vertex কয়টি? একটি ছবি আঁকা যাক।



নকশা ৪.৪: Ellipse এর উপর $n=9$ টি বিন্দু

চিত্র 8.8 এ আমরা ellipse এর উপর 9 টি বিন্দু এঁকেছি। মনে কর আমরা জানতে চাই $1-5$ এই লাইনের উপর 1 আর 5 ছাড়া আর কয়টি বিন্দু আছে? সহজ, খেয়াল কর এই লাইনের বামে আছে 3 টি বিন্দু আর ডানে আছে 4 টি বিন্দু। তাহলে একে মোট 3×4 টি জায়গায় ছেদ করবে। এর মানে কোনো একটি রেখার বামে যদি i টি বিন্দু থাকে তাহলে ডানে থাকবে $n-i-2$ টি বিন্দু। সুতরাং এটি মোট $i(n-i-2)$ টি ছেদ বিন্দু তৈরি করবে। কিন্তু 1 থেকে লাইন যদি আমরা 2 এ টানতাম তাহলে $i=0$. যদি 3 এ টানতাম তাহলে $i=1$ এভাবে যদি 9 এ টানতাম তাহলে $i=7$. সুতরাং একটি বিন্দু হতে যতগুলি কর্ণ তৈরি হয় তাতে মোট ছেদ বিন্দুর সংখ্যা $\sum_{i=0}^{n-2} i(n-i+2)$. যেহেতু একটি বিন্দু থেকে উৎপন্ন কর্ণগুলিতে এতোগুলি ছেদ বিন্দু পাওয়া যায় সেহেতু n টি বিন্দু হতে উৎপন্ন কর্ণগুলিতে ছেদ বিন্দুর সংখ্যা এর n গুন হবে। তবে একটা জিনিস, এখানে কিন্তু over count হয়েছে। কত বার over count হয়েছে? প্রতিটি ছেদ বিন্দু কিন্তু দুটি কর্ণের উপর থাকে। প্রতিটি কর্ণের প্রান্তবিন্দুর জন্য এই ছেদ বিন্দু count হবে। অর্থাৎ যদিও AB আর CD দুটি কর্ণের একটি ছেদ বিন্দু, কিন্তু সেটি A, B, C ও D এই চারটি বিন্দু সাপেক্ষেই count হবে। সুতরাং প্রতিটি ছেদ বিন্দু 4 বার count হয়েছে। তাই আমাদেরকে sum টিকে 4 দিয়ে ভাগ করতে হবে। সেই সাথে খেয়াল রেখো আমরা কিন্তু প্রান্ত বিন্দু গুনি নাই এখনো। সুতরাং ঐ sum কে 4 দিয়ে ভাগ করে n এর সাথে যোগ করলে আমরা vertex এর সংখ্যা পেয়ে যাব। যেহেতু n অনেক বড় আমরা লুপ চালিয়ে এই sum বের করতে পারব না। আমাদেরকে sum টির closed form বের করতে হবে। আমরা 8.১ এ এটার

closed form বের করে দেখিয়েছি।

$$
\begin{aligned}
V' &= \sum_{i=0}^{n-2} i(n-i+2) \\
&= \sum_{i=0}^{n-2} ni - i^2 + 2i \\
&= \sum_{i=0}^{n-2} ni - \sum_{i=0}^{n-2} i^2 + \sum_{i=0}^{n-2} 2i \\
&= n\sum_{i=0}^{n-2} i - \sum_{i=0}^{n-2} i^2 + 2\sum_{i=0}^{n-2} i \\
&= n\frac{(n-2)(n-1)}{2} - \frac{(n-2)(n-1)(2n-3)}{6} + 2\frac{(n-2)(n-1)}{2}
\end{aligned}
\tag{৪.১}
$$

এবার E বের করা যাক। আমরা E একটু সহজে বের করব। আমরা যদি আমাদের গ্রাফের সব vertex এর degree এর যোগফল বের করতে পারি তাকে 2 দিয়ে ভাগ করলেই E পেয়ে যাব। এই সমস্যায় degree এর যোগফল বের করা সহজ। কারণ প্রতিটি ছেদ বিন্দুর degree 4 আর শীর্ষবিন্দু গুলির degree $n+1$ (চিত্রে 1 এর সাথে 2 হতে শুরু করে 9 সবার একটি edge আছে। সেই সাথে 1 এর সাথে দুটি ellipse এর border আছে, এগুলিও edge, অর্থাৎ ellipse এর border বরাবর 1 যে 2 আর 9 এর সাথে যুক্ত তার কথা বলছি)। সুতরাং এদের সবাইকে যোগ করে 2 দিয়ে ভাগ করলেই আমরা E পেয়ে যাব। এখন আমরা euler এর ফর্মুলায় V ও E এর মান বসিয়ে F এর মান বের করলেই আমাদের সমস্যা সমাধান হয়ে যাবে।

একটা জিনিস। n এর মান কিন্তু 2^{31} এর মত হতে পারে। আবার আমাদের ফর্মুলাতে n^3 এর টার্ম আছে। আমরা যদি long long এও হিসাব করি তাহলেও overflow হবে। সুতরাং আমরা চাইলে c++ এ bigint এ কোড করতে পারি অথবা java তে কোড করলে আমরা java এর built in library এর BigInteger ব্যবহার করতে পারি। আমি বলব তোমাদের উচিত c++ ও java দুই ভাবেই সমাধান করা। দুই উপায়ের সাথে পরিচিত থাকা ভাল।

UVa 1363 Joseph's Problem

সমস্যা: n ও k দেওয়া থাকবে ($1 \le n, k \le 10^9$). $\sum_{i=1}^{n} k \bmod i$ বের করতে হবে।

সমাধান: সাধারণত তুমি যদি কোথাও i এর উপর বিশাল লুপ দেখো আর লুপের ভেতরে i যদি ভাগ অবস্থায় থাকে (integer division) তাহলে বেশির ভাগ সময় একটি সুন্দর $\sqrt{n}$ মেথড কাজে লাগে। আমরা i এর মান 1 হতে 30 এর জন্য $30/i$ এর মান টেবিল ৪.১ এ লক্ষ্য করি।

যখন তুমি ছোট ছোট সংখ্যা দিয়ে ভাগ করেছ তখন নিত্য নতুন মান পাওয়া যায়। যেমন 1 দিয়ে ভাগ করলে 30, 2 দিয়ে করলে 15 আবার 3 দিয়ে করলে 10. কিন্তু যদি বড় মান দিয়ে কর তখন দেখা যায় অনেকগুলির জন্য একই মান আসে। যেমন 16 হতে 30 যাই দিয়ে 30 কে ভাগ কর না কেন 1 হবে। 11 হতে 15 যাই দিয়ে ভাগ কর না কেন 2 হবে। সুতরাং মূল আইডিয়া হলো তুমি i এর মান কে দুই ভাগে ভাগ করবে ছোট আর বড়। ছোট মান গুলি দিয়ে ভাগ করে করে k/i এর মান বের করবে। আর বড় গুলি দিয়ে ভাগ করবে না। বরং বড়গুলি দিয়ে ভাগ করলে যেই উত্তর আসে সেই উত্তরের উপর লুপ চালাবে। এখন ছোট বড় ভাগ করবে কীভাবে? সাধারণত $\sqrt{k}$ দিয়ে এর সাপেক্ষে আমরা ছোট আর বড় ভাগ আলাদা করে থাকি। ছোট i গুলির জন্য আমরা 1 হতে ততক্ষণ লুপ চালাব যতক্ষণ k/i এর মান $\sqrt{k}$ এর বড় হয়। তাহলে আমরা সেই সব i প্রসেস করে ফেলেছি যাদের জন্য $k/i > \sqrt{k}$.

i	30/i	i	30/i	i	30/i
1	30	11	2	21	1
2	15	12	2	22	1
3	10	13	2	23	1
4	7	14	2	24	1
5	6	15	2	25	1
6	5	16	1	26	1
7	4	17	1	27	1
8	3	18	1	28	1
9	3	19	1	29	1
10	3	20	1	30	1

সারনী ৪.১: 30/i এর টেবিল

এবার বড় i গুলি প্রসেস করা যাক। আমাদের লক্ষ্য হল সেসব i প্রসেস করা যাদের জন্য $k/i \leq \sqrt{k}$ হয়। মনে কর আমাদের উত্তর হল j (অর্থাৎ আমরা j এর একটি লুপ চালাচ্ছি 1 হতে $\sqrt{k}$ পর্যন্ত)। তুমি কি বলতে পারবে কোন কোন মানের জন্য j উত্তর হবে? অর্থাৎ আমরা যদি decimal division এর কথা ভুলে যাই তাহলে আমরা সেসব a চাই যেন $k/a = j$ হয়। অর্থাৎ $a = k/j$. কিছু উদাহরণ দেখা যাক। $j = 1$ হলে $30/1 = 30$ যেটি হলো 1 উত্তর হবার শেষ মাথা (কারণ 31 দিয়ে ভাগ করলে 0 হয়ে যায়)। আমরা যদি $j = 2$ দিয়ে ভাগ করি তাহলে পাই $30/2 = 15$ (16 দিয়ে ভাগ করলে 1 হয়ে যায়) এবং এটি হলো 2 উত্তর হবার শেষ মাথা। $j = 3$ দিয়ে ভাগ করলে $30/3 = 10$ পাই এবং 10 হল 3 উত্তরের শেষ মাথা। সুতরাং আমরা উত্তর এর উপর যদি লুপ চালাই এবং তা দিয়ে যদি k কে integer ভাগ করি তাহলে আমরা সেই উত্তর এর শেষ মাথা পেয়ে যাব। তবে এই কাজ $\sqrt{k}$ এর থেকে বেশি উত্তরের জন্য করতে যেও না (যদি j এর লুপ ঠিক মত চালিয়ে থাক তাহলে এই সমস্যা হবে না)। বা করলেও একটু সাবধানে করিও। যেমন আমাদের উদাহরণে যদি আমরা উত্তর 9 ধরতে চাই এবং $30/9 = 3$ করি তাহলে কিন্তু হবে না। 3 এ কিন্তু উত্তর 10. আসলে 9 উত্তর আলা কোনো সংখ্যাই নেই। তুমি যদি একটু চিন্তা ভাবনা কর তাহলে বুঝবা যে $\sqrt{k}$ পর্যন্ত প্রতিটি মান উত্তর হবার যোগ্য। তাই এ নিয়ে আমরা ওত চিন্তা করব না। যাই হোক আমরা যদি জানি 1 উত্তর হবে শেষ কোন সংখ্যায় (30), 2 উত্তর হবে শেষ কোন সংখ্যায় (15), 3 উত্তর হবে শেষ কোন সংখ্যায় (10) তাহলে কিন্তু আমরা কোন উত্তর কয়বার বা কোন কোন সংখ্যার ক্ষেত্রে কোনটি উত্তর আসবে সব বের করে ফেলতে পারব। যেমন আমরা বলতে পারি 1 উত্তর হবে 16 হতে 30, 2 উত্তর হবে 11 হতে 15 ইত্যাদি। আমরা চাইলে এখন যেই সব সংখ্যার জন্য j উত্তর তাদের count নিতে পারি, তাদের যোগফল নিতে পারি ইত্যাদি।

এতক্ষণ আমরা একটা সমাধানের টেকনিক বললাম। এই সমাধানের টেকনিক ব্যবহার করে আমরা আমাদের সমস্যা কীভাবে সমাধান করা যায় তা দেখবো। প্রথমত খেয়াল কর, আমাদের সমস্যায় লুপের ভেতরে mod আছে। একে আমরা ভাগে পরিবর্তন করতে পারি। $k \bmod i = k - (k/i)i$ যেখানে ভাগটি হলো integer division. সুতরাং আমরা লিখতে পারি $\sum_{i=1}^{n} k - (k/i)i$ বা আসলে আমাদের $\sum_{i=1}^{n} (k/i)i$ বের করলেই চলে কারণ $\sum_{i=1}^{n} k = nk$। এবার আমাদের টেকনিক কাজে লাগবে। প্রথমেই দেখে নাও n কি k এর থেকে বড় কিনা। বড় হলে $i = k+1$ হতে n এর জন্য k/i যে 0 হবে তাতো আমরা আগে থেকেই জানি। সুতরাং সেসব বাদ। আমরা ধরে নিতে পারি $n \leq k$. এখন আমরা ছোট বড় দুই ভাগ করব $\sqrt{k}$ এর ভিত্তিতে। প্রথমে $i = 1$ হতে লুপ চালিয়ে $(k/i)i$ বের করতে থাকব যতক্ষণ $k/i > \sqrt{k}$ হয় (খেয়াল করতে হবে i যেন আবার n এর থেকে বড় না হয়ে যায়)। এবার ভাগফলের উপর লুপ চালানোর পালা। আমরা এবার j এর লুপ চালাবো 1 হতে $\sqrt{k}$ পর্যন্ত। প্রতিবার j উত্তর হবে এরকম শেষ সংখ্যা বের করব k/j. $j+1$ উত্তর হবে এরকম শেষ সংখ্যা $k/(j+1)$. সুতরাং $k/(j+1)+1$ হতে k/j পর্যন্ত সংখ্যাগুলির জন্য $k/x = j$ হবে যেখানে x হলো ঐ রেঞ্জে থাকা যেকোনো সংখ্যা। সুতরাং আমাদের sum যে ছিল $(k/i)i$ এর উপর সেটা এখন পালটে হয়ে যাবে jx কারণ আমরা ধরেছি j হল ভাগফল আর x হলো সেরকম সংখ্যা গুলি।

সুতরাং প্রতি j এর জন্য আমরা যদি x গুলি অর্থাৎ $k/(j+1)+1$ হতে k/j পর্যন্ত এর যোগফল বের করে ফেলতে পারি তাহলেই হবে। আর এই যোগফল চাইলেই ফর্মুলার সাহায্যে বের করে ফেলা যায়। একটা জিনিস আবারো খেয়াল রাখতে হবে যেন এই যে সংখ্যার রেঞ্জ এর মাঝে যেন n এর থেকে বড় সংখ্যা না থাকে। শুনতে কঠিন লাগলেও আসলে এটা ওত কঠিন না। তুমি একটু কাগজে কলমে নিজে থেকে চিন্তা করে দেখো আর কোড করে দেখো তাহলেই পরিষ্কার হয়ে যাবে।

UVa 11440 Help Mr. Tomisu

সমস্যা: M ও N দুটি সংখ্যা দেওয়া হবে ($1 \leq M \leq N \leq 10^7$ এবং $N - M \leq 10^5$). 2 এবং N! (N factorial) এর মাঝে কতগুলি সংখ্যা আছে যাদের সকল prime divisor ই M হতে বড়? 500 টি test case থাকতে পারে।

সমাধান: এধরনের mathematical সমস্যার ক্ষেত্রে সমাধানের কোনো নির্দিষ্ট টেকনিক নেই। মূলত observation এর দরকার। অর্থাৎ কাগজ কলম নিয়ে এদিক অদিক নানা দিক থেকে চিন্তা করে সমাধান করতে হবে। এই সমস্যার ক্ষেত্রে মূল observation হলো 2 থেকে N! এর মাঝে যেসকল সংখ্যা M! এর সাথে coprime তারাই উত্তর। কেন? কারণ M! এর ভেতরে M এর থেকে ছোট সব prime আছে। আবার M এর থেকে বড় কোনো prime এতে নেই। সুতরাং যেসকল সংখ্যা M! এর সাথে coprime তারাই উত্তর। $phi(M!)$ কিন্তু বের করা সহজ তাই না? এটি হলো $M!(1 - 1/p_1)(1 - 1/p_2)\dots$ যেখানে p_i হলো M এর সমান বা ছোট prime. সুতরাং আমরা যদি 10^7 পর্যন্ত সকল prime generate করে ফেলি তাহলে খুব সহজেই 10^7 পর্যন্ত সকল সংখ্যার factorial এর phi function আমরা বের করে ফেলতে পারব। কীভাবে?

খেয়াল কর $phi((i-1)!)$ এর সাথে $phi(i!)$ এর খুব close সম্পর্ক আছে। প্রথমত ফর্মুলার প্রথম term টি $(i-1)!$ হতে $i!$ হয়ে যাবে। সেই সাথে যদি i prime হয় তাহলে $(1 - 1/i)$ নামের একটা term চলে আসবে। অর্থাৎ যদি i prime না হয় তাহলে $phi((i-1)!)$ এর সাথে i গুন করলেই $phi(i!)$ পাওয়া যাবে। আর যদি prime হয় তাহলে $(i-1)$ গুন করলেই হবে। এভাবে আমরা $phi(i!)$ precalculate করে ফেলতে পারব। কিন্তু এটাতো বের হলো $M!$ এর সমান বা ছোট কয়টি coprime সংখ্যা আছে (M! এর সাথে)। আমাদের তো দরকার $N!$ এর সমান বা ছোট কয়টি coprime আছে (M! এর সাথে)। আসলে $phi(M!)$ কে তোমরা যদি $N!/M!$ দিয়ে গুন কর তাহলেই হবে। কেন? দেখো 1 হতে M! এর মাঝে x যদি M! এর সাথে coprime হয় তাহলে $(2M! + x)$ ও coprime হবে, $(3M! + x)$ ও হবে। অর্থাৎ 1 হতে M! এর মাঝের coprime এর লিস্ট M!+1 হতে 2M!, 2M! + 1 হতে 3M! ইত্যাদির মাঝেও থাকবে। সুতরাং আমাদের $phi(M!)$ এর সাথে $N!/M!$ বা $(M+1)(M+2)\dots N$ গুন করতে হবে। যেহেতু $N - M \leq 10^5$ তাই আমরা প্রতি case এ এই কাজ করতেই পারি।

UVa 10214 Trees in a Wood

সমস্যা: তোমাকে $[-a, a] \times [-b, b]$ সাইজের একটি গ্রিড দেওয়া থাকবে যেখানে $1 \leq a \leq 2000, 1 \leq b \leq 2 \times 10^6$. তুমি যদি $(0, 0)$ তে দাঁড়াও তাহলে তোমাকে বলতে হবে কয়টি গ্রিড বিন্দু দেখতে পারবে। যেমন, তুমি যদি $(0, 0)$ তে দাঁড়াও তাহলে তুমি $(1, 1), (-1, 1)$ এদের দেখতে পারবে কিন্তু $(2, 2), (10, 10), (5, 0)$ এদের দেখতে পারবে না। সমস্যার সুবিধার জন্যে ধরে নাও $(0, 0)$ কে আমরা গণার ভেতর ধরব না অর্থাৎ এই বিন্দুটি দেখা গেল কি গেল না তা নিয়ে না ভাবলেও হবে।

সমাধান: মূল সমস্যায় এই count কে মোট গ্রিড বিন্দুর সংখ্যা দিয়ে ভাগ করে output করতে হবে, কিন্তু আমার মনে হয় তোমরা সবাই মোট গ্রিড বিন্দুর সংখ্যা নিজেরাই গুনতে পারবে। সুতরাং এখানে আমরা কয়টি বিন্দু কে দেখতে পারব সেটার উপর নজর দেই। প্রথম কথা, আমরা চার axis এ চারটি বিন্দু দেখতে পারব $(1, 0), (0, 1), (-1, 0), (0, -1)$. আমরা যদি কোনোভাবে বের করতে পারি যে

প্রথম quadrant এ (axis বরাবর বিন্দু বাদে) কয়টি গ্রিড পয়েন্টকে আমরা দেখতে পারব, তাহলে তাকে চার দিয়ে গুন করলেই আমরা সকল quadrant এর জন্য উত্তর বের করে ফেলতে পারব। কারণ সব quadrant হলো symmetric. সুতরাং আমরা জানতে চাই $[1, a] \times [1, b]$ এর মাঝে কোন কোন বিন্দুকে আমরা দেখতে পারব। ধরা যাক আমরা (i, j) কে দেখতে পাই। তাহলে কিন্তু (ki, kj) কে আমরা দেখতে পারব না। আবার অন্যভাবে বললে বলা যায় যদি (i, j) একটি বিন্দু হয় এবং i ও j এর যদি একটি সাধারণ গুণনীয়ক g থাকে তাহলে $(i/g, j/g)$ দ্বারা (i, j) বাধা প্রাপ্ত হবে। অর্থাৎ আমরা সেসব (i, j) কে দেখতে পারব যাদের কোনো সাধারণ গুণনীয়ক নেই। তার মানে আমাদের বের করতে হবে $1 \leq i \leq a, 1 \leq j \leq b$ হলে কয়টি (i, j) এর জন্য $gcd(i, j) = 1$ বা i ও j পরস্পর coprime হবে। আমি এখানে এটি সমাধানের দুইটি উপায় বিস্তারিত বলব আর আরেকটি advanced উপায়ের জন্য সামান্য hints দেব। যাদের ইচ্ছা তারা যেন নেটে দেখে নিতে পারে।

প্রথম উপায় হলো euler এর phi function ব্যবহার করা। কিছুক্ষণ আগেই আমরা দেখেছি i এর সমান বা ছোট এবং তার সাথে coprime এরকম সংখ্যার count হলো $phi(i)$. আর কিছুক্ষণ আগেই (UVa 11440) এ দেখেছি এই coprime গুলি প্রতি i টি সংখ্যায় repeat হয়। অর্থাৎ 1 হতে i এর মাঝে যতগুলি সংখ্যা i এর সাথে coprime, $i+1$ হতে $2i$ এর মাঝে ঠিক ততটি সংখ্যাই coprime, $2i+1$ হতে $3i$ এর মাঝেও। এভাবে চলতে থাকবে। সুতরাং আমরা i এর একটি লুপ চালাবো 1 হতে a পর্যন্ত। এবং প্রতিটি i এর জন্য আমরা দেখবো 1 হতে b এর মাঝে কতগুলি সংখ্যা আছে যারা i এর সাথে coprime. এজন্য আমরা প্রথমে $phi(i)$ বের করব এবং b/i দিয়ে (integer division) গুন করব। বাকি থাকবে $i(b/i) + 1$ হতে b পর্যন্ত সংখ্যাগুলি। এখানে বাকি সংখ্যার পরিমাণ $b \bmod i$ তাই না? কারণ এগুলি হলো i এর সবথেকে শেষ multiple থেকে b পর্যন্ত সংখ্যাগুলি। যেহেতু i এর মান a থেকে বড় না সুতরাং এর মান 2000 থেকেও বড় না। অর্থাৎ এই যে অতিরিক্ত সংখ্যা তা কখনই 2000 থেকে বড় হবে না। আর এটি প্রতিটি i এর জন্যই। সুতরাং আমরা বলতে পারি 2000^2 টি সংখ্যার জন্য আমাদেরকে আলাদা করে gcd বের করতে হবে। এটা প্রতি case এ কোনো ব্যাপার না। যদিও কয়টি case থাকবে সেটা আমাদের বলা নেই। তাহলে আমাদের এই সমাধানের time complexity হবে $\mathrm{O}(a^2)$. সমাধানটা হলো- আমরা i এর একটি লুপ চালাবো 1 হতে a পর্যন্ত। তার phi(i) বের করব এবং b/i দিয়ে গুন করব। এরপর $i(b/i)+1$ থেকে b পর্যন্ত লুপ চালিয়ে i এর সাথে gcd বের করে দেখবো যে তাদের gcd 1 কিনা। এভাবে gcd 1 ওয়ালা pair এর সংখ্যা count করব।

এবার দ্বিতীয় উপায় বলা যাক। যখনই দেখবা coprime হতে হবে বা তাদের gcd g হতে হবে এরকম ধরনের কথাবার্তা বলা থাকবে তখন এই পদ্ধতি প্রায়ই কাজে লাগে। যদি বলত $gcd(i, j) = g$ কত গুলি (i, j) এর জন্য হবে? তাহলে জিনিসটা কঠিন। কিন্তু যদি বলত $gcd(i, j)$ g এর multiple হবে কত ভাবে? তাহলে কিন্তু সেই উত্তর বের করা সহজ। এটি কিন্তু $(a/g) \times (b/g)$ (integer ভাগ)। কেন? কারণ $[1, a/g] \times [1, b/g]$ এই গ্রিডের যেকোনো বিন্দু ধরা যাক (x, y) তাহলে (gx, gy) বিন্দুটি $[1, a] \times [1, b]$ গ্রিডে থাকবে এবং তাদের gcd হবে g এর multiple. এখন একটা জিনিস চিন্তা কর $[1, a] \times [1, b]$ এই গ্রিডে যেকোনো (x, y) এর gcd কিন্তু সর্বোচ্চ $M = min(a, b)$। তাহলে আমরা 1 হতে M পর্যন্ত প্রতিটি সংখ্যা i এর জন্য বের করতে পারি কতগুলি বিন্দু আছে যাদের gcd i এর multiple. তাহলে আমাদের সমস্যা নিচের সমস্যায় পরিণত হবে-

f ও g দুটি n সাইজের অ্যারে দেওয়া আছে। $f[i]$ এ থাকবে g এর অ্যারেতে i এর multiple এর পজিশনগুলিতে যেসব সংখ্যা থাকবে তাদের যোগফল অর্থাৎ $f[i] = \sum_j g[j]$ যেখানে j হলো i এর multiple. f এর মান সমূহ দেওয়া আছে, g এর মানগুলি বের করতে হবে।

এটি একটি common সমস্যা। অর্থাৎ বিভিন্ন সমস্যা সমাধান করতে গেলে এরকম সমস্যায় reduce হয়ে চলে আসে। এই সমস্যা $n \log n$ এ সমাধান করা যায়। আইডিয়া হল g এর মান পেছন থেকে বের করা, অর্থাৎ প্রথমে $g[n]$, এরপর $g[n-1]$, এরপর $g[n-2]$ এরকম করে একে একে g[1] পর্যন্ত মান বের করতে হবে। মনে কর $g[i]$ এর মান বের করতে চাও। যেহেতু আমরা পেছন থেকে মান বের করছি সুতরাং i এর থেকে বড় সব index এ g এর মান ইতোমধ্যেই বের করে ফেলেছি। অর্থাৎ আমরা i বাদে i এর সব multiple এর মান বের করে ফেলেছি। এখন আমরা যদি $f[i]$ হতে i বাদে i এর অন্যান্য multiple এর index এ থাকা g এর মান বিয়োগ করি তাহলে $g[i]$ বের হয়ে যাবে। অর্থাৎ তুমি যখন $g[4]$ বের করতে চাইবে তখন যা করতে হবে তাহলো $g[4] = f[4] - g[8] - g[12] - g[16] \ldots$. আর এটার complexity যে sieve এর মত হবে তাতো

বুঝতেই পারছ।

তৃতীয় উপায়ের কথা আমি বিস্তারিত বলব না। যাদের আগ্রহ আছে তারা mobious inversion পড়ে দেখতে পারো। এভাবে করলে complexity প্রতি কেসে $O(n)$ হবে (যদিও sieve এর মত precalculation লাগবে একটা)।

UVa 1393 Highways

সমস্যা: একটি $n \times m$ সাইজের গ্রিড দেওয়া আছে ($1 \leq n, m \leq 300$)। মনে কর গ্রিডের প্রতিটি বিন্দু একেকটি শহর। এখন তুমি প্রতিটি শহরের pair এর মাঝে রাস্তা বানাবে। এর ফলে কিছু রাস্তা overlap করতে পারে, যেমন $(1,1)$ হতে $(2,2)$ পর্যন্ত রাস্তা $(1,1)$ হতে $(3,3)$ পর্যন্ত রাস্তার সাথে overlap করে। এসব ক্ষেত্রে ছোট রাস্তাকে বিবেচনা করার দরকার নেই। তোমাকে বলতে হবে মোট কতগুলি রাস্তা আছে। এই গণনার সময় horizontal বা vertical রাস্তাগুলিও বিবেচনার বাহিরে রেখো।

সমাধান: আমরা ইতোমধ্যেই শিখে এসেছি যে যদি i ও j এর gcd 1 হয় তাহলে $(0,0)$ হতে (i,j) পর্যন্ত রেখা টানলে সেটি কোনো বিন্দুর উপর দিয়ে যায় না। অর্থাৎ এই রাস্তা অন্য কোনো ছোট রাস্তাকে overlap করে না। আমরা এরকম সব ছোট ছোট রাস্তা গুনব তাহলেই আমাদের মূল সমস্যা সমাধান হয়ে যাবে। আসলেই কি তাই? না। খেয়াল কর $(0,0)-(1,1)$ এবং $(1,1)-(2,2)$ এ রাস্তা দুটি কাউকেই cover করে না। কিন্তু এদুটিকে আলাদা করে count করলে overcount হয়ে যাবে। তাহলে কি করা যায়? আমরা কি সবচেয়ে বড় রাস্তা গুলি count করব? আমরা যদি তা করতে পারতাম তাহলে আসলেই এই সমস্যা সমাধান হয়ে যেত। কিন্তু আমি এভাবে ভেবে কোনো সমাধান আনতে পারি নাই।

যেভাবে সমাধান হবে তাহলো প্রতি direction এর জন্য বের করা যে কতগুলি রাস্তা আছে যা এই বরাবর চলে। যেমন একটি 3×3 গ্রিডে কতগুলি $(1,1)$ direction এর রাস্তা আছে? 3 টি। $(0,0),(1,0),(0,1)$ এই তিনটি বিন্দু হতে। অন্য বিন্দু গুলি হতে তুমি যদি $(1,1)$ বরাবর রাস্তা টান তাহলে হয় তা সরাসরি বাইরে যায় (যেমন $(2,0)$ হতে) অথবা আগের তিনটি রাস্তার উপর থাকে (যেমন $(1,1)$ হতে)। তাহলে আমরা i ও j এর উপর লুপ চালাবো। যদি দেখি তাদের gcd 1 না তাহলে তো বাদ। আর নাহলে আমাদের বের করতে হবে এই দিক বরাবর কতগুলি ভিন্ন ভিন্ন রাস্তা আছে। বা অন্য ভাবে কতগুলি বিন্দু থেকে এই রাস্তা শুরু হতে পারে? ধরা যাক আমরা জানতে চাইছি (x,y) হতে এই রাস্তা শুরু হতে পারে কিনা। এখান থেকে যদি কোনো রাস্তা শুরু হয় তাহলে এর পরের বিন্দু হবে $(x+i, y+j)$. আর আগেই বলেছি এই রাস্তা অন্য কোনো রাস্তাকে overlap করতে পারবে না। তাই আমরা $(x-i, y-j)$ ও দেখবো। কারণ যদি $(x-i, y-j)$ বিন্দুটি আমাদের গ্রিডে থাকে তাহলে ঐ বিন্দু থেকে শুরু (i,j) দিকের রাস্তা (x,y) হতে শুরু একই দিকের রাস্তাকে overlap করবে। তাহলে (x,y) হতে রাস্তা শুরু হতে পারবে যদি ($x+i \leq m$ এবং $y+j \leq n$) এবং ($x-i<0$ বা $y-j<0$) হয়। এরকম কতগুলি (x,y) পাওয়া যাবে? এরকম ক্ষেত্রে যেটা করা উচিত তাহলো একটা ছবি আঁকা এবং valid অংশ চিহ্নিত করে সেই অংশে কতগুলি বিন্দু আছে তা count করা। এই অংশ তোমাদের উপর ছেড়ে দেয়া যায়।

তাহলে এভাবে আমরা প্রতি case এ $O(nm)$ এ সমাধান করতে পারি। মনে হয় এই complexity তে accepted হয়ে যাবে। তবে তোমরা যারা সমাধানকে আরও ভাল করতে চাও তারা চিন্তা করে $O(300^2)$ এ precalculation করে রাখা যায় কিনা চেষ্টা করে দেখতে পারো। সমাধানটাতে dynamic programming ব্যবহার করতে হবে। এই সমাধানটা যত না math সম্পর্কিত তার থেকে অনেক বেশি dynamic programming সম্পর্কিত। তাই এখানে আর এ নিয়ে আলোচনা বাড়ালাম না।

UVa 1642 Magical GCD

সমস্যা: n টি ধনাত্মক সংখ্যা দেওয়া আছে যেখানে n সর্বোচ্চ 10^5 এবং প্রতিটি সংখ্যা সর্বোচ্চ 10^{12} হতে পারে। এমন একটি contiguous subsequence বের করতে হবে যার element গুলির gcd

এবং সেই subsequence এর length এর গুণফল যেন সকল contiguous subsequence এর মাঝে সবচেয়ে বেশি হয়। যেমন- যদি আমাদের মূল sequence হয় $30, 60, 20, 20, 20$ তাহলে আমাদের উত্তর হবে 80 যা আমরা শেষ চারটি element দিয়ে তৈরি contiguous subsequence থেকে পাবো।

সমাধান: এখানে সবচেয়ে বড় observation হলো-

মনে কর তুমি i এ শেষ হয় এরকম সকল contiguous subsequence নিলে এবং তাদের gcd গুলি বের করলে। এখানে আসলে $\log a$ এর বেশি ভিন্ন ভিন্ন gcd নেই যেখানে a হল আমাদের i তম সংখ্যা। অর্থাৎ তুমি যদি $(i), (i-1, i), (i-2, i-1, i), \ldots (1 \ldots i)$ এরকম index গুলির জন্য যদি যথাক্রমে $gcd(a[i]), gcd(a[i-1], a[i]) \ldots gcd(a[1], \ldots a[i])$ বের কর তাহলে এখানে $\log a[i]$ এর থেকে বেশি ভিন্ন gcd থাকবে না।

কেন? কারণ খুব সহজ। মনে কর তুমি i এ শেষ হওয়া সকল contiguous subsequence এর জন্য gcd বের করলে। অর্থাৎ $gcd(a[i])$, $gcd(a[i], a[i-1])$ এভাবে $gcd(a[i] \ldots a[1])$ পর্যন্ত। এরা একটি non increasing sequence. কারণ আগের সংখ্যার সাথে নতুন আরেকটি সংখ্যা লাগিয়ে gcd বের করলে নতুন সংখ্যা কখনই আগের সংখ্যা থেকে বড় হবে না। আর যদি ছোট হয় তাহলে কমপক্ষে দুই গুন ছোট হবে। কারণ পরের সংখ্যা তো আগের সংখ্যাকে ভাগ দেবেই। আর যদি সেই সাথে ছোট হতে হয় তাহলে কমপক্ষে দুভাগের সমান হবে। তার থেকে ছোটও হতে পারে, কিন্তু বড় হবার সম্ভাবনা নেই। অর্থাৎ যখন ছোট হচ্ছে তখন অন্তত 2 ভাগ হবেই। সুতরাং আমাদের এখানে আসলে $\log(a[i])$ এর থেকে বেশি ভিন্ন gcd থাকবে না। কারণ তুমি যদি $a[i]$ কে $\log a[i]$ বার 2 দিয়ে ভাগ কর তাহলে তো 1 হয়ে যাবে। আর একবার যদি 1 হয়ে যায় তাহলে তো আর ছোট হতে পারবে না।

তাহলে কীভাবে আমরা এই সমস্যা সমাধান করব? খুব সহজ। আমরা একটি অ্যারেতে ভিন্ন ভিন্ন gcd এর জন্য লিখে রাখবো যে এই gcd পেতে আমরা সর্বোচ্চ এতো দূর যেতে পারি। অর্থাৎ g gcd পেতে আমরা i হতে j পর্যন্ত যেতে পারি ($j < i$), g' পেতে হয়তো k পর্যন্ত যেতে পারি ($k < j$). এরকম। যখন আমরা নতুন সংখ্যা অর্থাৎ $i + 1$ তম সংখ্যা নেব তখন আমরা নতুন সংখ্যার সাথে পুরাতন gcd (g, g' ইত্যাদি) গুলির gcd বের করব। যদি পাশাপাশি দুটি gcd একই হয় তাহলে ঐ দুইটি range কে merge করে দেব। এভাবে চলতে থাকবে। উদাহরণ দেওয়া যাক, মনে কর আমাদের sequence এ দুটি সংখ্যা আছে $a[0] = 25, a[1] = 60$. তাহলে আমাদের অ্যারেতে লিখা থাকবে 60 পেতে 1 পর্যন্ত যেতে হয়, আর 5 পেতে 0 পর্যন্ত যেতে হয়। এখন মনে কর নতুন সংখ্যা আসল 35 ($a[2] = 35$). তখন আমরা লিখব 35 পেতে 2 পর্যন্ত যেতে হয়। 5 পেতে যেতে হয় 1 (gcd(35, 60) = 5), 5 পেতে যেতে হয় 0 (gcd(35, 5) = 5). খেয়াল কর শেষ দুটি gcd একই, সুতরাং এদের merge করে আমরা লিখতে পারি 5 পেতে 0 পর্যন্ত যেতে হয়। এই কাজ আসলে খুব সহজেই কোড করা যাবে, সে নিয়ে বিস্তারিত বলছি না। এই সমস্যা সমাধানের মূল জিনিস হলো $\log n$ সংক্রান্ত observation.

UVa 10868 Bungee Jumping

সমস্যা: w ভরের এক জন লোক একটি ব্রিজ এর উপর হতে bungee jumping করবে। ব্রিজটি s উচ্চতায় আছে। এখন লোকটি যদি লাফ মারে তাহলে সে একটি free fall body হিসাবে কাজ করবে, অর্থাৎ সে কোনো বাধা ছাড়া পৃথিবীর দিকে আসবে এবং তার accelaration হবে g (অভিকর্ষজ ত্বরণ). দড়িটির দৈর্ঘ্য l. যখন তার দূরত্ব ব্রিজ হতে l এর থেকে বেশি হয়ে যাবে তখন bungee এর দড়ি তাকে উপরে টানবে। ধরা যাক সে l দূরত্বের থেকেও Δl দূরত্ব বেশি অতিক্রম করেছে। সেসময় দড়িটি উপরের দিকে $k\Delta l$ force এ টানবে। লোকটি মাটিতে নিরাপদ ভাবে পৌঁছাবে যদি মাটি স্পর্শের সময় তার বেগ $10ms^{-1}$ এর বেশি নাহয়। যদি তার বেশি হয় তাহলে লোকটি মরে যাবে। তোমাকে w, l, s, k এর মান দেওয়া আছে। বলতে হবে লোকটি কি নিরাপদে মাটিতে নামতে পারবে নাকি মরে যাবে নাকি মাটি পর্যন্ত আসতেই পারবে না।

সমাধান: নিঃসন্দেহে আরও একটি বেশ কঠিন সমস্যা। সত্যি বলতে এই সমস্যা সমাধান করা আমার

জন্যও এক রকমের physics ঝালাই।

এই সমস্যা সমাধানের জন্য আমাদের মূল যেই theorem জানতে হবে তাহলো- শক্তির কোনো ধ্বংস নেই, এটি কেবল এক রূপ হতে অন্য রূপে রূপান্তরিত হয়। এখানে মানুষটি একটি ব্রিজের উপর দাড়িয়ে আছে। সুতরাং তার একটি potential energy (স্থিতি শক্তি) আছে। এখন সে যদি লাফ দেয় তাহলে যতক্ষণ না সে মাটিতে আসছে ততক্ষণ তার potential energy কমতে থাকবে। কিন্তু এই কমা শক্তি কই যাবে? এর এক অংশ পরিণত হবে kinetic energy (গতি শক্তি) তে আর আরেক অংশ পরিণত হবে bungee দড়ির potential energy তে, কারণ দড়ির দৈর্ঘ্য l এর থেকে বেশি হয়ে গেলে সেটি আবার পিছে (মানে উপরে) টানতে শুরু করবে। সুতরাং আমাদের কাজ এই সব energy বের করা।

তোমরা চাইলে potential energy এবং kinetic energy এর ফর্মুলা বা তার derivation নেটে দেখতে পারো। মূল কথা হলো m ভরের কোনো বস্তু যদি ভূপৃষ্ঠ হতে h উচ্চতায় থাকে তাহলে তার potential energy হবে mgh যেখানে g হলো অভিকর্ষজ ত্বরণ। আর v বেগের কোনো বস্তুর kinetic energy হলো $\frac{1}{2}mv^2$. আর সমস্যাতেই বলা আছে যে bungee দড়ি যদি সাম্য অবস্থা হতে Δl দূরত্ব সরানো হয় তাহলে সেখানে দড়িটি $k\Delta l$ শক্তি প্রয়োগ করবে। তাহলে দড়ির সাম্য অবস্থা থেকে z দূরত্বে এর potential energy এর পরিমাণ কত হবে? এভাবে চিন্তা কর x এ থাকতে এর উপর কাজ করা বলের পরিমাণ kx. এখন যদি তুমি তাকে খুব ছোট dx দূরত্ব সরাও তাহলে কাজ হবে $kxdx$ (কাজ = বল x সরণ). তাহলে 0 হতে z পর্যন্ত সড়াতে কাজের পরিমাণ $\int_0^z kxdx$ বা $\frac{1}{2}kz^2$.

তাহলে আমরা আমাদের প্রয়োজনীয় সব ফর্মুলা জানি। এখন আমাদের কাজ কিছু equation দাঁড় করানো। শুরুতেই বলেছি শক্তির ধ্বংস নেই। সুতরাং একদম শুরুতে যখন লোকটি ব্রিজের উপরে ছিল তখন তার potential energy এর পরিমাণ ছিল wgs. যেহেতু সে তখন স্থির অবস্থানে ছিল তাই তার kinetic energy 0. সুতরাং শুরুতে মোট energy wgs.

এখন মনে কর সে লাফ দিল এবং x দূরত্ব অতিক্রম করে ফেলল। এই অবস্থায় তার potential energy হবে $wg(s-x)$. কিন্তু kinetic energy? এটা তার বেগের উপর নির্ভর করে, ধরলাম তার বেগ v, তাহলে kinetic energy হবে $\frac{1}{2}wv^2$. আর bungee দড়ির potential energy? যদি $x \leq l$ হয় তাহলে কিন্তু দড়ির potential energy 0. কারণ এখনো দড়ি তার সাম্য অবস্থায় যায় নাই, সাম্য অবস্থা হতে দূরে যাওয়া তো দূরের কথা। সুতরাং এই ক্ষেত্রে আমাদের শক্তির নিত্যতা সূত্র বলে $wgs - wg(s-x) = \frac{1}{2}wv^2$ কারণ আমাদের potential energy তে যেই পরিমাণ পরিবর্তন হয়েছে তা সম্পূর্ণ kinetic energy তে রূপান্তরিত হয়েছে, দড়ির potential energy তে পরিণত হয় নাই। এখন যদি $s \leq l$ হয়, অর্থাৎ দড়ির দৈর্ঘ্য যদি ব্রিজের উচ্চতার চেয়ে বড় হয় তাহলে আমরা এই সূত্র হতে v এর মান বের করতে পারি ($x = s$ বসিয়ে)। এবং v এর মান 10 এর সাথে তুলনা করে দেখবো যে লোকটা মরে যায় নাকি ভালভাবে মাটিতে পৌঁছে।

এখন যদি $x > l$ হয় তাহলে? এবার তাহলে দড়ির potential energy আছে। সাম্য অবস্থা থেকে এটি $x - l$ দূরত্বে আছে। সুতরাং এর potential energy হবে $\frac{1}{2}k(x-l)^2$. অর্থাৎ আমাদের equation হবে $wgs - wg(s-x) = \frac{1}{2}wv^2 + \frac{1}{2}k(x-l)^2$. এখন কি করবে? এই সমীকরণে $x = s$ বসিয়ে v বের করবে? না। তার আগে দেখতে হবে দড়ি আদৌ মাটিতে পৌঁছায় কিনা। যদি দড়ি মাটিতে না পৌঁছায় তাহলে মাটিতে পৌঁছানর আগেই কোথাও থেকে সে ফেরত গিয়েছে। ঠিক সেই জায়গাতে সেই মুহূর্তে $v = 0$ হয়ে গেছে। তাই উপরের equation এ $v = 0$ বসিয়ে x বের কর। যদি দেখো যে $x < s$ এর জন্য সমাধান আছে তার মানে সে মাটি স্পর্শের আগেই ফিরে গেছে। আর নাহলে মাটি স্পর্শ করেছে। যদি মাটি স্পর্শ করে তাহলে $x = s$ বসিয়ে v এর মান বের করে ফেল।

LOJ 1278 Sum of Consecutive Integers

সমস্যা: একটি পূর্ণসংখ্যা N দেওয়া থাকবে ($1 \leq N \leq 10^{14}$). বলতে হবে N কে কত ভাবে একাধিক ধনাত্মক ক্রমিক পূর্ণসংখ্যার (consecutive positive integers) যোগফল হিসাবে প্রকাশ

করা যায়। যেমন $N = 15$ কে আমরা তিনভাবে প্রকাশ করতে পারি $1+2+3+4+5$, $4+5+6$ এবং $7+8$. Test case সংখ্যা 200 টি।

সমাধান: এধরনের সমস্যাগুলি সমাধান করা খুব একটা কঠিন না। যা করতে হয় তাহলো একে ফর্মুলার আকারে লিখতে হবে এরপর সেই ফর্মুলা কতভাবে সমাধান করা যায় তা দেখতে হবে। যেমন এই সমস্যাতে আমরা ধরতে পারি আমাদের সিরিজের শুরুর সংখ্যা a আর মোট n টি সংখ্যা আছে সেই সিরিজে। তাহলে আমাদের সিরিজ টা হবে $a, a+1, \ldots (a+n-1)$. এদের যোগফল হবে N. আর শর্ত মতে $n > 1$ এবং $a > 0$ হতে হবে। এই সিরিজের সংখ্যাগুলিকে যোগ করলে কত হবে? $\frac{n(2a+n-1)}{2}$. বিভিন্ন ভাবে এই যোগফল বের করা যায়। একটি উপায় হলো $\sum_{i=0}^{n-1} a+i$ তবে সহজ উপায় হলো গাউসের পদ্ধতি ব্যবহার করা। এ নিয়ে এর আগের বইয়ে আলোচনা হয়েছে তবে যদি কেউ না জেনে থাকো নেটে দেখে নিতে পারো। যাই হোক, তাহলে এই যোগফলের মান হলো N. এর মানে আমরা লিখতে পারি $\frac{n(2a+n-1)}{2} = N$ বা $n(2a+n-1) = 2N$. আমরা N জানি কিন্তু a বা n জানি না। আমাদের লক্ষ্য হলো কতগুলি valid a ও n এর মান পাওয়া যাবে তা বের করা। এই ফর্মুলা থেকে একটি জিনিস পরিষ্কার যে n হবে $2N$ এর একটি গুণনীয়ক। আর কি কি বুঝা যায়? যেহেতু আমাদের ডান পাশ একটি জোড় সংখ্যা সুতরাং আমাদের বাম দিকের দুটি factor এর একটিকে অবশ্যই জোড় হতে হবে। কিন্তু একটু খেয়াল করলে তুমি আরও কিছু আবিষ্কার করবে। যেমন মনে কর n জোড়, তাহলে $2a+n-1$ অবশই বিজোড় হবে। আবার যদি n বিজোড় হয় তাহলে $2a+n-1$ অবশ্যই জোড় হবে। উদাহরণ দেখা যাক, ধর $N = 6$. তাহলে $n(2a+n-1) = 12$. আমরা 12 কে নানা ভাবে factor করতে পারি $1 \times 12, 2 \times 6, 3 \times 4$. এর মানে আমাদের n এই 6 টি সংখ্যার কোনো একটি হবেই। কিন্তু কোন কোনটি হতে পারে? তুমি যদি n এর মান ধর 2 তাহলে $2a+n-1$ কে হতে হবে 6 (কারণ $n(2a+n-1) = 12$) আর তাহলে $a = 5/2$. এরকম করে সকল n এর জন্য আমরা a এর মান বের করতে পারি। এবং এই মানগুলির দিকে তাকালে আমরা বুঝতে পারব কোন কোন মান n এর জন্য valid আর কোন কোনটি না। valid কি না কীভাবে বুঝবে? যদি n ও a দুটিই পূর্ণসংখ্যা হয় এবং $n > 1, a \geq 1$ হয় তাহলেই সেটি valid. আরও একটি উদাহরণ দেখা যাক। ধর $N = 15$. তাহলে $n(2a+n-1) = 30$. 30 কে আমরা কত ভাবে factorize করতে পারি? $1 \times 30, 2 \times 15, 3 \times 10, 5 \times 6$. সুতরাং $n = 1, 2, 3, 5, 6, 10, 15, 30$ এই 8 রকম মান হতে পারে। এর মানে কি N = 15 তে উত্তর 8? না। আমরা Problem Description এ দেখেছি উত্তর 3. তাহলে কাহিনী কি? প্রথমত $n > 1$ হতে হবে। তাই 1 বাদ। এখনো 7 টি মান বাকি আছে। তুমি যদি $n = 10$ চেষ্টা কর তাহলে দেখবে $2a+n-1 = 3$ আর তাহলে $a = -3$. কিন্তু a কে ধনাত্মক হতে হবে। এর মানে $n = 10$ হতে পারবে না। একই ভাবে $n = 6, 15, 30$ হতে পারবে না। এর মানে সমাধান 3 টি। তাহলে এই দুইটি উদাহরণ থেকে কি শিখলে? শিখলাম যে, $2N$ কে factorize করলেই হবে না। $2N$ এর যত factor আছে তাদের মাঝে কিছু কিছু n হতে পারবে। কিন্তু কারা? প্রথমত 1 হতে পারবে না। কারণ $n > 1$. আর $a > 0$ হবার জন্য আমাদের কি শর্ত পূরণ করতে হবে? একটু চিন্তা করলে বুঝবে যে $2a+n-1 > n$ হতে হবে। কেন? কারণ $2N$ এর একটি factor হলো n আর ওপর factor $2a+n-1$. যেহেতু a ধনাত্মক তাহলে তো $2a+n-1 > n$ হবেই তাই না (কারণ $a > 0$ এবং integer হলে $2a-1 > 0$)? সুতরাং আমরা সব শেষে যা বুঝতে পারলাম তাহলো $2N$ এর কতগুলি divisor x আছে যা 1 এর থেকে বড়, x ও $2N/x$ একই সাথে জোড় না এবং $x < 2N/x$[১] হতে হবে। আমরা যা করতে পারি তাহলো $2N$ এর প্রতিটি divisor নিয়ে নিয়ে চেক করতে পারি। কিন্তু 200 টি case এর জন্য এই কাজ করা একটু বেশি costly হয়ে যেতে পারে। এর থেকে কি আরও ভাল কোনো উপায় আছে?

এই পর্যায়ে এসে তুমি নানা ভাবে চিন্তা করতে পারো। একটি উপায় হতে পারে- আমরা যদি restriction গুলিকে relax করি তাহলে কি হবে। অথবা উলটাও চিন্তা করতে পারো। আমরা একে একে restriction গুলিকে আরোপ করলে কি হবে। সবার প্রথমে আমাদের কোনো restriction নেই। তাই আমাদের বের করতে হবে কয়টি divisor আছে। এর একটি ফর্মুলা আছে $(q_1+1)(q_2+1)\ldots(q_k+1)$ যেখানে $2N = p_1^{q_1} p_2^{q_2} \ldots p_k^{q_k}$. এখন যেই তিনটি restriction আছে তার মাঝে সবচেয়ে কঠিন মনে হয় একটি জোড় আরেকটি বিজোড়। কিন্তু এই ফর্মুলা অনুসারে এটি খুব একটা কঠিন না। যদি $p_1 = 2$ হয় তাহলে (q_1+1) এর টার্মটা ফর্মুলা থেকে সরায়ে দিতে পারলেই হবে।

[১] সমান কি হতে পারবে? যেহেতু একটি জোড় আরেকটি বিজোড় হতে হবে সেহেতু আসলে দুটি সমান কখনই হতে পারবে না।

মানে আমরা 2 বাদে বাকি prime গুলিকে সব ভাবে distribute করছি। এবার 2 গুলিকে একবার বাম দিকে আরেকবার ডান দিকে দিলেই হবে। উদাহরণ দেয়া যাক, মনে কর $2N = 30$. তাহলে 2 গুলি সরিয়ে নিলে থাকে 15. 15 এর divisor হচ্ছে $1, 3, 5, 15$. এদের সাথে 2 গুন করলে কিছু divisor পাবো (ধরা যাক এদেরকে y দিয়ে প্রকাশ করা হয়) আর না করলে কিছু (ধরা যাক এদের x দিয়ে প্রকাশ করা হয়)। প্রতিটি y এর জন্য একটি x আছে (x = 2N/y) আর উলটোটাও সত্যি অর্থাৎ প্রতিটি x এর জন্য একটি y আছে। এই x, y pair এর একটি ছোট হবে অপরটির থেকে আর সেটিই হবে আমাদের n. কেবল $n = 1$ হতে পারবে না এটিই বাকি থাকল। এটা 1 বিয়োগ করে নিলেই হবে। তবে $N = 1$ এর ক্ষেত্রে একটু সাবধান হতে হবে এই আর কি।

যেমন $2N = 30 = 2 \times 3 \times 5$. তাহলে $(1+1)(1+1) = 4$. সুতরাং মোট 4 টি odd divisor. আর তাদের 2 দিয়ে গুন করলে আমরা আরও 4 টি পাবো। মোট 8 টি। এদেরকে xy আকারে লিখলে শুধু ছোট গুলি n হতে পারবে। সুতরাং $8/2 = 4$ টি। এই ছোট 4 টির মাঝে একটি হলো 1. তাহলে উত্তর $4 - 1 = 3$. সুতরাং আমাদের উত্তর হবে $(q_2 + 1) \ldots (q_k + 1) - 1$.

LOJ 1282 Leading and Trailing

সমস্যা: n এবং k দেওয়া আছে ($2 \le n < 2^{31}, 1 \le k \le 10^7$)। তোমাকে n^k এর শুরুর 3 ডিজিট এবং শেষের 3 ডিজিট প্রিন্ট করতে হবে।

সমাধান: শেষের তিন ডিজিট বের করা খুব সহজ। তোমাকে $n^k \bmod 1000$ বের করতে হবে তাহলেই হবে (bigmod)। কিন্তু প্রথম তিন ডিজিট? এখানেই আসে log এর সৃজনশীল ব্যবহার। একটা উদাহরণ দেখা যাক মনে কর $29^{8751919}$. একে log নিলে হবে $\log 29^{8751919} = 8751919 \log 29 = 1.27987888233738 \times 10^7$. যদিও যেকোনো base এর log নিলেই হয় কিন্তু আমরা এখানে 10 base log নিয়েছি। এই log এর মান দেখে কি বুঝা যায়? এভাবে চিন্তা করলে মনে হয় বুঝা কঠিন হয়ে যাবে। একটু উলটো ভাবে চিন্তা করা যাক। ধরা যাক শুরুর তিনটি অংক হলো abc. এর মানে আমরা আমাদের সংখ্যা n^k কে লিখতে পারি $0.abc \times 10^d$ যেখানে d হলো number of digit, আর আমরা শুধু abc এই তিনটি শুরুর সংখ্যাই ধরেছি। এর পরেও আরও অনেক ডিজিট আছে যেটা আপাতত আমরা consider করছি না, কিন্তু ব্যাপারটা মাথায় রাখো। এখন যদি এর log নাও তাহলে হবে $\log(0.abc \times 10^d) = \log 0.abc + d$. 0 আর 1 এর মাঝের যেকোনো সংখ্যাকে log করলে negative সংখ্যা পাওয়া যায়। Negative সংখ্যা নিয়ে চিন্তা করা একটু ঝামেলা। সেজন্য আমরা এই equation কে একটু ঘুরিয়ে লিখি- $\log a.bc \times 10^{d-1} = \log a.bc + (d-1)$. এখানে প্রথম অংশ $\log a.bc$ যা অবশ্যই ধনাত্মক কারণ $a.bc > 1$ (a হলো n^k এর প্রথম ডিজিট, আর প্রথম ডিজিট তো অবশ্যই শূন্য হবে না তাই না?) আর $a.bc < 10$. সুতরাং আমরা বলতে পারি $0 < \log a.bc < 1$. তার মানে $\log a.bc + (d-1)$ এর দশমিকের আগের সংখ্যা বলে $d-1$ এর মান আর তার পরের সংখ্যাগুলি হলো $\log a.bc$. কিছুক্ষণ আগের উদাহরণে আমরা দেখেছি $\log 29^{8751919} = 1.27987888233738 \times 10^7 = 12798788 + 0.8233738\ldots$. এ থেকে আমরা বুঝতে পারছি যে $29^{8751919}$ তে মোট 12798789 টি ডিজিট আছে। আর $\log a.bc\ldots = 0.8233738\ldots$. তাহলে আমরা $\log n^k$ হতে এর floor বাদ দিলেই আমরা $\log a.bc\ldots$ পেয়ে যাব। কিন্তু আমাদের দরকার প্রথম তিনটি ডিজিট। সেজন্য আমরা দুদিকে 10 এর power নিব, $a.bc\ldots = 10^{0.8233738\ldots} = 6.6584605\ldots$. এর মানে আমাদের প্রথম তিন ডিজিট 665. বিশ্বাস হচ্ছে না? তোমরা চাইলে wolframalpha.com বা python এ হিসাব করে দেখতে পারো $29^{8751919}$ এর আসল মান কত। বা চাইলে একটি bigint এর কোড c++ বা java তে লিখে ফেলতে পারো।

LOJ 1289 LCM from 1 to n

সমস্যা: মোট 10,000 টি case থাকতে পারে। তোমাকে n দেওয়া আছে $1 \le n \le 10^8$, $lcm(1, 2, \ldots n)$ বের করতে হবে। যেহেতু উত্তর অনেক বড় হবে তাই উত্তরটি 2^{32} এর mod এ দিতে হবে।

সমাধান: উত্তর 2^{32} এর mod এ দিতে হবে, তাই আমরা যদি সব হিসাব unsigned int এ করি তাহলে mod আপনা আপনিই হয়ে যাবে।

আশা করি তোমরা lcm বের করার নানা উপায় জান। চিন্তা কর এর মাঝে কোন পদ্ধতিটি এখানে কাজে দেবে। LCM বের করার একটি উপায় হলো প্রতিটি প্রাইম নিয়ে দেখা যে তার সর্বোচ্চ কোন power আমাদের প্রদত্ত সংখ্যার কোনো না কোনো একটি সংখ্যাকে ভাগ করে। আমাদের সমস্যায় আমাদেরকে 1 হতে n পর্যন্ত সবগুলি সংখ্যা দেওয়া আছে। আমরা যদি 1 হতে n পর্যন্ত প্রতিটি prime এর জন্য বের করতে পারি সেই প্রাইমের সর্বোচ্চ কোন power টি n এর সমান বা ছোট, তাহলেই আমরা lcm এ সেই prime এর power পেয়ে যাব। যেমন n যদি 15 হয় আর আমরা যদি 2 নিয়ে মাথা ব্যাথা করি তাহলে 2 এর সর্বোচ্চ power যা খুব জোড় n সেটা হবে $2^3 = 8$, কারণ $2^4 = 16$ কিন্তু 15 এর থেকে বড়। সুতরাং আমাদের lcm এ 2 এর power হবে 3. কিন্তু প্রতি case এ 10^8 পর্যন্ত সব prime নিয়ে নিয়ে এই চেক করা খুবই সময় সাপেক্ষ ব্যাপার। কারণ আমাদের case আছে 10,000 টি। তাহলে উপায় কি?

মাঝে মাঝে খুব ভাল উপায় বের করার থেকে বর্তমান পদ্ধতিকে optimize করতে চাওয়া টা বুদ্ধিমানের মত কাজ হয়। যেমন এখানে, একটু চিন্তা করলে বুঝতে পারবা যে আসলে $\sqrt{n}$ এর থেকে বড় প্রাইমগুলির খুব জোড় 1 power থাকবে। তাই আমরা যদি $\sqrt{n}$ এর থেকে বড় এবং সর্বোচ্চ n পর্যন্ত প্রাইমগুলির গুণফল সহজে এবং কম সময়ে বের করতে পারতাম তাহলে প্রতি case এ $\sqrt{n}$ সময় লাগত, যা ac পাবার জন্য যথেষ্ট। এই ক্ষেত্রে আমরা square root decomposition পদ্ধতি ব্যবহার করতে পারি। 1 হতে 10^8 পর্যন্ত সকল প্রাইম বের করে তাদেরকে square root segment এ ভাগ করে প্রতি segment এর গুণফল বের করে রাখো। তাহলেই তোমরা প্রতিটি n এর জন্য খুব সহজে $\sqrt{n}$ হতে n পর্যন্ত prime গুলির গুণফল বের করে ফেলতে পারবে। Square root decomposition না জানলে তো বই আর নেট আছেই।

LOJ 1236 Pairs Forming LCM

সমস্যা: মোট 200 টি case থাকতে পারে। তোমাকে n দেওয়া আছে $1 \leq n \leq 10^{14}$, বলতে হবে কতগুলি i এবং j আছে যেন $lcm(i, j) = n$ হয় যেখানে $1 \leq i \leq j \leq n$.

সমাধান: মনে কর n কে আমরা prime factorization করলাম $p_1^{q_1} p_2^{q_2} \ldots$. তাহলে প্রতি p_x এর জন্য i এবং j তে p_x এর সর্বোচ্চ power হতে পারবে q_x. শুধু তাই না, i বা j এর কোনো একটিতে q_x power থাকতেই হবে। যদি এই শেষ restriction টি না থাকত তাহলে i বা j তে power $0, 1, \ldots q_x$ এর কোনো একটি হতে পারত। অর্থাৎ i তে power হতে পারত $q_x + 1$ রকম, j তেও তাই। সুতরাং আমাদের মোট $(q_x + 1)(q_x + 1)$ রকমের power এর combination হতে পারত। কিন্তু শেষ restriction বলছে তাদের একটিকে q_x হতেই হবে। ধরা যাক i তে এই q_x power আছে আর j তে নাই, তাহলে j এর power হতে পারবে $0, 1, \ldots q_x - 1$ এই q_x রকম, একই ভাবে j তে q_x power কিন্তু i তে না, তা হতে পারে q_x ভাবে। i ও j দুটিতেই power q_x এটি হতে পারে 1 ভাবে। সুতরাং মোট $2q_x + 1$ ভাবে এই power থাকতে পারে। আমরা যদি প্রতিটি prime এর power এর জন্য এই সংখ্যা বের করে গুন করতে পারি তাহলে আমরা সকল i ও j এর সংখ্যা বের করে ফেলতে পারব। তবে এখানে একটা সমস্যা আছে। আমাদের সমস্যায় বলেছিল $i \leq j$ হতে হবে। কিন্তু আমাদের এই counting তা মানে না। সেটি মানতে হলে কি করতে হবে সেটা মনে হয় তোমরা নিজেরাই বের করে ফেলতে পারবে। যদি না পার, তাহলে কাগজে কলমে বের করে দেখ এই দুই count অর্থাৎ আমাদের আলোচিত count আর আসল উত্তর এর মাঝে পার্থক্য কেমন।

LOJ 1027 A Dangerous Maze

সমস্যা: একটি maze এ মোট n টি দরজা আছে (সর্বোচ্চ 100 টি)। প্রতিটি দরজার জন্য একটি সংখ্যা আছে a_i. যদি a_i ধনাত্মক হয় এর মানে হলো এই দরজা নির্বাচন করলে তুমি এই maze হতে a_i সময় পর বের হয়ে যাবে। আর যদি a_i ঋণাত্মক হয় তাহলে তোমাকে $abs(a_i)$ সময় পর আবারো এই maze এ ফেরত আসতে হবে কিন্তু তখন স্মৃতি শক্তি refresh করে দেওয়া হবে, অর্থাৎ

এর আগে কোন কোন দরজায় গিয়েছিলে তা তোমার আর মনে থাকবে না। প্রতিবার তুমি randomly দরজা নির্বাচন করবে। তোমাকে বলতে হবে এই maze হতে বেরুতে expected কত সময় লাগবে। যদি এই maze হতে বের হওয়া সম্ভব না হয় তাহলে impossible প্রিন্ট কর।

সমাধান: যদি কোনো দরজার জন্যই a_i ধনাত্মক না হয় তাহলে তো এই maze হতে বের হওয়া সম্ভব না।

এখন মনে কর আমরা দরজাগুলিকে দুই ভাগে ভাগ করলাম। এক ভাগে আছে a_i ধনাত্মকগুলি ধরা যাক এরা হলো x (অর্থাৎ দরজাগুলি হবে $x_1, x_2 \ldots$ এরকম)। আর অন্য ভাগে আছে a_i ঋণাত্মক দরজাগুলি যাদের আমরা y দিয়ে চিনব। ধরা যাক maze হতে বের হতে আমাদের expected সময় লাগে E. আমরা randomly একটি দরজা পছন্দ করলাম। কোনো একটি দরজা নির্বাচন করার probability হবে $1/n$. এখন যদি আমরা x_i দরজা পছন্দ করি তাহলে এই maze হতে বের হতে সময় লাগবে x_i. আর যদি y_i দরজা পছন্দ করি তাহলে সময় লাগবে $y_i + E$. কেন? কারণ y_i সময় পর আমরা maze এ ফিরে আসব, এর পর আমাদের expected সময় লাগবে E. তাহলে আমরা E সমান লিখতে পারি $\frac{1}{n}(x_1 + x_2 + \ldots) + \frac{1}{n}(y_1 + E + y_2 + E + \ldots)$. যদি এখনো না বুঝো তাহলে এভাবে বুঝার চেষ্টা কর- আমরা randomly একটি দরজা নির্বাচন করব। এটি x_i হলে সময় লাগবে x_i আর যদি y_i হয় তাহলে y_i সময় তো লাগবেই সেই সাথে E সময় লাগবে কারণ আমরা পুরো প্রসেস নতুন করে শুরু করছি। সুতরাং প্রতিটি দরজার জন্য কত সময় লাগবে তাকে যদি আমরা সেই দরজা নির্বাচন করার সম্ভাব্যতা দিয়ে গুন করে তাদের যোগ করি তাহলেই আমরা আমাদের expected value E পাবো। উপরের সমীকরণে E এর মান সমাধান করলেই আমরা আমাদের উত্তর পাবো।

এই সমস্যায় আমি ইচ্ছা করেই কিছু mathematically incorrect কথা বলেছি যেমন x_i দরজা তে সময় লাগবে x_i. কিন্তু আশা করি আমার মূল উদ্দেশ্য বুঝতে পারবে, কারণ আমি যদি দরজা বুঝাতে অন্য কোনো variable ব্যবহার করতাম বা mathematically correct ভাবে বলতে চাইতাম তাহলে মনে হয় অনেক অনেক variable এবং definition আমাকে বলতে হতো।

LOJ 1030 Discovering Gold

সমস্যা: একটি গুহাতে n টি ঘর (ঘরগুলি 1 হতে n দ্বারা প্রকাশ করা হয়) পাশাপাশি আছে। গুহার প্রতিটি ঘরে কিছু সোনা আছে আর সেই সোনার পরিমাণ তোমাকে input এ বলা আছে। তুমি শুরুতে 1 এ আছ। এই অবস্থায় তুমি একটি ছক্কা মারবে। ছক্কায় 1 হতে 6 এর মাঝে কোনো একটি সংখ্যা randomly পড়বে, ধরা যাক i. তখন তুমি যদি x তম ঘরে থাকো তাহলে i পড়ার পর তুমি $x+i$ তম ঘরে যাবে। যদি $x + i > n$ হয় তাহলে তুমি আবারো ছক্কা মারবে। যতক্ষণ না তুমি n এর ভেতরের একটি সংখ্যা পাও। যদি তুমি n এ পৌঁছায়ে যাও তাহলে তোমার খেলা শেষ। তুমি যতগুলি ঘরে যাবে সব ঘর হতে সোনা সংগ্রহ করবে। তোমাকে বলতে হবে এই খেলায় n এ পৌঁছালে expected সোনার পরিমাণ কত। n সর্বোচ্চ 100 হতে পারবে।

সমাধান: ধরা যাক E_i হলো i তম ঘর হতে শুরু করে n তম ঘরে গিয়ে শেষ করলে expected কত সোনা পাওয়া যায় সেই পরিমাণ (সেই ঘর সহ)। আমাদের বের করতে হবে E_1 এর মান। মনে কর তুমি 1 এ থাকতে একটি ছক্কা মারলে। তাহলে এতে 1 হতে 6 এর যেকোনো একটি পড়তে পারে। ধরা যাক x পরল। তাহলে পরের ঘর হবে $1 + x$. এখান থেকে তুমি expected E_{1+x} টি সোনা পাবে। x পড়ার সম্ভাবনা $1/6$. এরকম করে যদি তুমি x এর মান 1 হতে 6 সবগুলি চেষ্টা কর তাহলে পাবে $E_1 = gold[1] + \frac{1}{6}(E_2 + E_3 + \ldots E_7)$ এরকম করে তুমি সব E_i এর মান বের করে ফেলতে পারবে। সমস্যা হবে যখন কোনো একটি E_i থাকবে না। যেমন তুমি E_{n-1} এ থেকে যদি 2 ফেলো তাহলে তুমি E_{n+1} এ যেতে চাইবে। কিন্তু E_{n+1} বলে তো কিছু নেই। সেক্ষেত্রে কি করবে? সেক্ষেত্রে তো তুমি ঐ ঘরেই থাকো তাই না? তার মানে তুমি আবারো ছক্কা মারবে আর সেক্ষেত্রে তোমার expected মান হবে E_{n-1} ই। অর্থাৎ আমাদের equation হবে $E_{n-1} = gold[n-1] + \frac{1}{6}(E_n + E_{n-1} + E_{n-1} + \ldots)$ (মোট 5 টি E_{n-1} থাকবে)। সুতরাং আমরা যদি $i = n$ হতে 1 পর্যন্ত E_i এর মান বের করি তাহলেই আমাদের সমস্যা সমাধান হয়ে যাবে।

আর এখানে base case হবে $E_n = gold[n]$ কারণ n তম ঘরে আসলে তুমি শুধু ঐ ঘরের সোনা নিয়ে খেলা শেষ করে দেবে।

LOJ 1284 Lights inside 3D Grid

সমস্যা: একটি $X \times Y \times Z$ আকারের গ্রিড দেওয়া আছে। গ্রিডের প্রতিটি সেলে একটি লাইট আছে যা শুরুতে off. তুমি K বার একটি অপারেশন করবে। প্রতি অপারেশনে তুমি গ্রিডের দুটি ঘর randomly নির্বাচন করবে। ধরা যাক এরা হলো (x_1, y_1, z_1) এবং (x_2, y_2, z_2). এবার তুমি $min(x_1, x_2) \leq x \leq max(x_1, x_2)$, $min(y_1, y_2) \leq y \leq max(y_1, y_2)$, $min(z_1, z_2) \leq z \leq max(z_1, z_2)$ এর জন্য সকল (x, y, z) এর লাইটগুলি toggle করবে। বলতে হবে K বার এই অপারেশন করার পর expected কতগুলি বাল্ব on থাকবে। X, Y, Z এর মান সর্বোচ্চ 100 আর K সর্বোচ্চ 10,000 হতে পারে।

সমাধান: Linearity of Expectation বলে একটি কথা আছে। এটি বলে অনেক জিনিসের expectation হবে প্রতিটি জিনিসের expectation এর যোগফল। অর্থাৎ যেমন এই সমস্যাতে, expected কতগুলি লাইট on থাকবে সেই পরিমাণ হবে প্রতিটি লাইট expected কতবার on থাকবে তার যোগফল। অর্থাৎ আমরা গ্রিডের প্রতিটি লাইটের জন্য বের করব সেই লাইট K টি অপারেশনের পর expected কতবার on থাকবে। প্রতিটি লাইটের জন্য এই সংখ্যা যদি আমরা যোগ করি তাহলেই উত্তর পেয়ে যাব।

ধরা যাক আমরা জানতে চাইছি (x, y, z) K অপারেশন শেষে expected কত বার on থাকে। এটা বের করা আর K বার শেষে on থাকার probability বের করা একই কথা। কেন? উলটো ভাবে চিন্তা করতে পারো। মনে কর K বার শেষে on থাকার probability q. তাহলে Expected কত বার on থাকবে? $q \times 1 + (1-q) \times 0 = q$. সুতরাং আমরা K অপারেশন শেষে on থাকার probability বের করতে চাইছি।

ধরা যাক p হলো গ্রিডের দুটি সেল (x_1, y_1, z_1) এবং (x_2, y_2, z_2) নির্বাচনের probability যেন (x, y, z) বিন্দুটি $min(x_1, x_2) \leq x \leq max(x_1, x_2)$, $min(y_1, y_2) \leq y \leq max(y_1, y_2)$, $min(z_1, z_2) \leq z \leq max(z_1, z_2)$ মেনে চলে। আমরা p এর মান বের করব। একটা জিনিস খেয়াল কর, আমরা কিন্তু x, y এবং z যে যথাক্রমে $[min(x_1, x_2), max(x_1, x_2)]$, $[min(y_1, y_2), max(y_1, y_2)]$, $[min(z_1, z_2), max(z_1, z_2)]$ এর ভেতরে থাকবে তার probability আলাদা আলাদা করে বের করে গুন করতে পারি। তাহলেই আমাদের (x, y, z) বিন্দুটি আমাদের range গুলি মেনে চলবে। তাহলে আমাদের এখন বের করতে হবে কত probability তে আমরা x_1, x_2 নির্বাচন করতে পারি যেন $min(x_1, x_2) \leq x \leq max(x_1, x_2)$ হয় প্রদত্ত x এর জন্য। এখানে "হয় না" বের করাটা সহজ হবে। "হয় না" হবে যদি $x_1, x_2 < x$ হয় অথবা $x < x_1, x_2$ হয়। x এর থেকে ছোট মান আছে $x-1$ টি, আর বড় মান আছে $X-x$ টি। সুতরাং আমাদের নির্বাচিত x_1, x_2 দুটিই ছোট হবে তার probability $\frac{1}{x-1} \times \frac{1}{x-1}$ আর দুটিই বড় হবে তার probability $\frac{1}{X-x} \times \frac{1}{X-x}$. এদের যোগ করে 1 থেকে বিয়োগ করলেই আমরা x রেঞ্জের ভেতরে থাকবে তার probability পেয়ে যাব। একই ভাবে y এবং z ভেতরে থাকার probability বের করে সবাইকে গুন করলেই আমরা p পেয়ে যাব।

এখন আমাদের বের করতে হবে এই p probability এর ঘটনাটি K বারের মাঝে বিজোড় সংখ্যক বার ঘটার probability কত। কারণ জোড় সংখ্যক বার ঘটা মানে লাইট অফ থাকবে, আর বিজোড় সংখ্যক বার ঘটা মানে লাইট অন থাকবে। এবার লাগবে আমাদের binomial theorem এর জ্ঞান। আমরা জানি $(x+y)^n = \sum_{i=0}^{n} \binom{n}{i} x^i y^{n-i}$. আবার $(-x+y)^n = \sum_{i=0}^{n} \binom{n}{i} (-x)^i y^{n-i}$. আমরা এই দুটিকে যোগ করে পাই $(x+y)^n + (-x+y)^n = \sum_i 2\binom{n}{i} x^i y^{n-i}$ যেখানে i হলো 0 হতে n এর মাঝের সকল জোড় সংখ্যা। সুতরাং আমরা যদি x এর পরিবর্তে p আর y এর পরিবর্তে $(1-p)$ বসাই তাহলে আমরা খুব সহজে p probability এর ঘটনা জোড় সংখ্যক বার ঘটার সম্ভাবনা পেয়ে যাব। তাহলে 1 হতে এই সংখ্যা বাদ দিলে বিজোড় সঙ্খ্যকবার ঘটার probability ও পেয়ে যাব!

আমার মনে হয় পুরো সমাধানের মূল অংশ গুলো বলা হয়ে গিয়েছে। এখন পুরোটুকু বুঝে সমাধান করে ফেলা তোমার দায়িত্ব!

৪.১ অনুশীলনী

৪.১.১ সমস্যা

Simple

- UVa 1648 Business Center
- LOJ 1014 Ifter Party
- LOJ 1035 Intelligent Factorial Factorization
- LOJ 1067 Combinations
- LOJ 1090 Trailing Zeroes (II)
- LOJ 1109 False Ordering
- LOJ 1007 Mathematically Hard
- LOJ 1028 Trailing Zeroes (I)
- LOJ 1054 Efficient Pseudo Code
- LOJ 1077 How Many Points?
- LOJ 1098 A New Function
- LOJ 1163 Bank Robbery

Easy

- UVa 11040 Add bricks in the wall
- UVa 1210 Sum of Consecutive Prime Numbers
- UVa 10622 Perfect Pth Powers
- UVa 10886 Standard Deviation
- UVa 11246 K-Multiple Free Set
- UVa 11303 Permutation
- LOJ 1024 Eid
- LOJ 1197 Help Hanzo
- LOJ 1214 Large Division
- LOJ 1333 Grid Coloring
- LOJ 1340 Story of Tomisu Ghost
- LOJ 1038 Race to 1 Again
- LOJ 1151 Snakes and Ladders
- LOJ 1256 Word Puzzle
- UVa 1644 Prime Gap
- UVa 10539 Almost Prime Numbers
- UVa 1646 Edge Case
- UVa 1647 Computer Transformation
- UVa 11166 Power Signs
- UVa 1655 Exam
- LOJ 1138 Trailing Zeroes (III)
- LOJ 1213 Fantasy of a Summation
- LOJ 1259 Goldbach`s Conjecture
- LOJ 1336 Sigma Function
- LOJ 1370 Bi-shoe and Phi-shoe
- LOJ 1104 Birthday Paradox
- LOJ 1248 Dice (III)
- LOJ 1058 Parallelogram Counting

Medium

- UVa 808 Bee Breeding
- UVa 11526 H(n)
- UVa 1649 Binomial coefficients
- UVa 10479 The Hendrie Sequence
- UVa 12520 Square Garden
- UVa 1652 Fibonacci System
- UVa 10976 Fractions Again?!
- LOJ 1298 One Theorem, One Year
- LOJ 1161 Extreme GCD
- UVa 294 Divisors
- UVa 11105 Semi-prime H-numbers
- UVa 10640 Planes around the World
- UVa 766 Sum of powers
- UVa 12590 Guards II
- UVa 11895 Honorary Tickets
- LOJ 1234 Harmonic Number
- LOJ 1318 Strange Game

অধ্যায় ৫

Bruteforce এবং Backtrack সম্পর্কিত সমস্যা

Bruteforce এবং backtrack কোনো রকমের smart algorithm না। বরং এটি গাধার মত সোজা সাপটা ভাবে যা করতে বলেছে তা করা। কিন্তু তাই বলে বেশি গাধার মত করলে হবে না। হয়তো মাঝে মাঝে তোমাকে ছোট খাটো ট্রিক্স খাটিয়ে সমাধানটি সহজে কোড করতে হবে। বা কোনো সময় ছোট খাটো optimization করে run time কে কমিয়ে আনতে হবে।

UVa 725 Division

সমস্যা: N দেওয়া থাকবে ($2 \leq N \leq 79$). তোমাকে দুটি 5 ডিজিটের সংখ্যা abcde ও fghij বের করতে হবে যাতে করে $\frac{abcde}{fghij} = N$ হয় এবং এই 5 ডিজিটের দুটি সংখ্যার 10 টি ডিজিট আলাদা আলাদা হয়। যেমন $N = 62$ এর জন্য একটি উত্তর হতে পারে $\frac{79546}{01283}$.

সমাধান: যেহেতু fghij একটি 5 ডিজিটের সংখ্যা আমরা এর জন্য 0 হতে 99999 পর্যন্ত লুপ চালাতে পারি। এরপর একে input এর N এর সাথে গুন করলে abcde বের হয়ে যাবে। আমরা এরপর চেক করে দেখতে পারি যে এই দুটি সংখ্যার সকল ডিজিট আলাদা আলাদা কিনা। যদি হয় তাহলে এই জুটি আমাদের একটি সমাধান।

তোমরা চাইলে কিছু ছোট খাটো optimization করতে পারো। যেমন fghij এর লুপ 0 হতে না চালিয়ে 1000 হতে চালানো। কারণ এর থেকে ছোট দুটি সংখ্যার প্রথম দুটি ডিজিট শূন্য। আবার যদি দেখো যে N কে fghij দিয়ে গুন করলে সংখ্যাটি 99999 এর থেকে বড় বা 1000 থেকে ছোট হয় তাহলে আর কষ্ট করে চেক করার দরকার নেই। এই সমস্যাতে এই ধরনের সহজ optimization দরকার নেই, তবে হয়তো অন্য কোনো সমস্যায় দরকার হতে পারে।

UVa 11059 Maximum Product

সমস্যা: তোমাকে n টি সংখ্যার একটি অ্যারে $S_1, S_2, \ldots S_n$ দেওয়া থাকবে যেখানে $1 \leq n \leq 18$ এবং $\mid S_i \mid \leq 10$. তোমাকে এর একটি continuous subarray বের করতে হবে যেন তাদের গুণফল সর্বোচ্চ হয়।

সমাধান: n মাত্র 18, তাই আমরা চাইলে এর সকল sub array নির্বাচন করে তার element গুলিকে গুন করতে পারি। তবে একটা জিনিস, যেহেতু অনেকগুলি সংখ্যা গুন করছি আমাদের খেয়াল রাখতে হবে গুণফল যাতে overflow না করে। আমাদের সংখ্যাগুলি খুব জোড় 10 হয়। 18 টি সংখ্যার গুণফল

তাই সর্বোচ্চ 10^{18} হতে পারে যা long long এ ধরবে। সুতরাং আমাদের long long এর variable নিয়ে হিসাব নিকাশ করতে হবে। int নিয়ে কাজ করলে হবে না, overflow হবে।

UVa 524 Prime Ring Problem

সমস্যা: তোমাকে একটি জোড় সংখ্যা n দেওয়া থাকবে ($1 \leq n \leq 16$). একটি সাইকেলে 1 হতে n পর্যন্ত সংখ্যাগুলিকে কত ভাবে সাজানো যায় যেন পাশাপাশি যেকোনো দুটি সংখ্যার যোগফল একটি prime হয়। যেমন $n = 6$ এর জন্য 143256 একটি সমাধান।

সমাধান: আমরা সাইকেলের শুরুতে 1 বসাই। এরপর আমাদের কাজ হলো 2 হতে n পর্যন্ত $n-1$ টি সংখ্যা একে একে সাইকেলে বসানো। আমরা এটি $(n-1)!$ ভাবে করতে পারি। কিন্তু $n = 16$ এর জন্য এটি হবে $15!$ যা একটু বেশিই বড় হয়ে যায়। এখানে একটা জিনিস লক্ষ্যনিয়। পাশাপাশি দুটি সংখ্যা বসালে prime পাওয়া যাবে এরকম ঘটনা খুব একটা ঘটবে না। তাই আমরা যা করতে পারি তাহলো সাইকেলে আমরা যখন সংখ্যা বসাব তখনই নতুন সংখ্যা আর তার পাশের সংখ্যা যোগ করে দেখতে পারি যে তাদের যোগফল প্রাইম হয় কিনা। না হলে আর এগুনোর দরকার নেই। আর হলে পরের জায়গা নিয়ে আমরা মাথা ব্যাথা করব। অর্থাৎ backtrack এর মত করে সাইকেলে আমরা একে একে সংখ্যা বসাতে থাকব। মনে কর আমাদের backtrack এর ফাংশন হল $bktk(at)$. এর মানে আমরা এখন at তম স্থানে সংখ্যা বসানোর চেষ্টা করব। আমরা একটি অ্যারে রাখব যেখানে 0-1 ব্যবহার করে লিখে রাখব কোন সংখ্যা ব্যবহার করা হয়েছে আর কোন সংখ্যা ব্যবহার করা হয় নাই। সুতরাং আমরা at এ এসে সেই অ্যারের উপর দিয়ে লুপ চালিয়ে দেখব যে কোন কোন সংখ্যা ব্যবহার করা হয় নাই। তাদেরকে at এ বসানোর চেষ্টা করব। চেষ্টা করা মানে দেখব যে এই সংখ্যা এর আগের সংখ্যার সাথে যোগ করলে যোগফল প্রাইম হয় কিনা। না হলে তো আবার আমরা পরের সংখ্যা নিয়ে চেষ্টা করব। আর যদি প্রাইম হয় তাহলে অ্যারেতে লিখে রাখব যে এই সংখ্যা ব্যবহার করা হয়ে গিয়েছে এবং $bktk(at+1)$ কল করব। একটা জিনিস তা হল, $bktk(at+1)$ এর পরের লাইনে, অর্থাৎ $bktk(at+1)$ হতে ফিরে এসে তোমাকে যেই সংখ্যা ব্যবহার করেছ সেই সংখ্যাকে অ্যারেতে ব্যবহার করা হয় নাই বলতে হবে, অর্থাৎ $bktk(at+1)$ কল করার আগে যদি আমরা $used[i] = 1$ করে থাকি তাহলে ফাংশন হতে ফিরে আমাদেরকে $used[i] = 0$ করতে হবে। আর শেষ সংখ্যা বসানোর সময় তার সাথে শুরুর সংখ্যা যোগ করে দেখতে ভুলে যেও না।

UVa 12325 Zombie's Treasure Chest

সমস্যা: একটি সোনা দানার দোকানে গিয়ে দেখলে সেখানে দুই ধরনের রত্ন আছে। প্রথম ধরনের রত্নের প্রতিটির ওজন s_1 এবং প্রতিটির মূল্য v_1. দ্বিতীয় ধরনের রত্নের ওজন s_2 এবং মূল্য v_2. প্রতি ধরনের রত্ন অসংখ্য সংখ্যক আছে। তোমার কাছে মোট n ওজনের রত্ন নেওয়া যাবে এরকম একটি থলে আছে। তোমাকে বলতে হবে সবচেয়ে কত বেশি মূল্যের রত্ন তোমার থলে তে নিতে পারবে। n, s_1, s_2, v_1, v_2 প্রতিটি 32 bit এর integer.

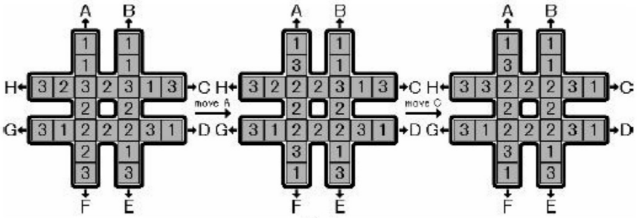
সমাধান: এর সমাধানটি আমার কাছে বেশ interesting লেগেছে। ধরা যাক $s_1 > s_2$. এখন দুটি case কল্পনা কর। প্রথম case, $s_1 > \sqrt{n}$. এই ক্ষেত্রে প্রথম জিনিস তুমি n/s_1 এর বেশি বার নিতে পারবে না আর যেহেতু $s_1 > \sqrt{n}$ তাই তুমি আসলে $\sqrt{n}$ এর বেশি বার প্রথম রত্ন নিতে পারবে না। কারণ s_1 ওজনের জিনিস তুমি যদি $\sqrt{n}$ এর থেকে বেশি বার যদি নাও তাহলে এই জিনিস গুলির ওজন হবে $s_1\sqrt{n}$ এর বেশি। যেহেতু $s_1 > \sqrt{n}$ সেহেতু মোট জিনিসের ওজন n এর থেকে বেশি হয়ে যায়। কিন্তু তা তো সম্ভব না। সুতরাং তুমি খুব সহজেই প্রথম রত্ন কয়টি নেবে তার উপর লুপ চালাতে পারবে (সর্বোচ্চ $\sqrt{n}$ টি নিতে পারবে)। ধরা যাক i টি নেবে। তাহলে বাকি থাকবে $x = n - is_1$ পরিমাণ জায়গা। এই জায়গায় তুমি দ্বিতীয় রত্ন সর্বোচ্চ x/s_2 টি নিতে পারবে।

সমস্যা হলো যখন $s_1 \leq \sqrt{n}$ হবে। এর মানে s_2 ও কিন্তু ছোট। এই ক্ষেত্রে একটি সুবিধা আছে। s_2 সংখ্যক প্রথম রত্নের ওজন আর s_1 সংখ্যক দ্বিতীয় রত্নের ওজন সমান (উভয়ই s_1s_2)। তুমি কোনটি নেবে? যার প্রতি কেজি মুল্য বেশি অর্থাৎ যার v_i/s_i বেশি তাই না? ধরা যাক $v_1/s_1 > v_2/s_2$. এর

মানে তুমি চেষ্টা করবে প্রথম ধরনের রত্ন বেশি বেশি করে নিতে। অর্থাৎ তুমি দ্বিতীয় রত্নটি কখনই s_1 এর সমান বা এর থেকে বেশি সংখ্যক নেবে না। নিলে সেই s_1 টি দ্বিতীয় রত্নের পরিবর্তে s_2 টি প্রথম রত্ন নিতে পারবে। যেহেতু $s_1 \leq \sqrt{n}$ সুতরাং তুমি দ্বিতীয় রত্ন 0 হতে $s_1 - 1$ প্রত্যেক ভাবে নেবার চেষ্টা করবে। আর বাকি অংশটুকু প্রথম রত্ন দিয়ে ভরানোর চেষ্টা করবে। এভাবে সকল প্রকার ভাবে চেষ্টা করলে তুমি optimal উপায় পেয়ে যাবে।

UVa 1343 The Rotation Game

সমস্যা: এই সমস্যায় চিত্র ৫.১ এর মত একটি বোর্ড দেওয়া থাকবে। বোর্ডটিতে 1, 2 এবং 3 সংখ্যা-গুলি মোট 8 বার করে থাকবে (সুতরাং মোট 24 টি সংখ্যা)। তুমি প্রদত্ত বোর্ডে 8 রকমের অপারেশন করতে পারবে। এগুলি চিত্রে $A, B \ldots H$ দিয়ে প্রকাশ করা হয়েছে। অর্থাৎ প্রতিটি অপারেশন হলো বড় row বা column কে cyclically উপরে-নিচে বা ডানে-বামে ঘুরান। যেমন চিত্রের প্রথম বোর্ডকে A বরাবর ঘুরালে দ্বিতীয় বোর্ড পাওয়া যায়, আবার দ্বিতীয় বোর্ডকে C বরাবর ঘুরালে তৃতীয় বোর্ড পাওয়া যায়। তোমাকে ঘুরিয়ে ঘুরিয়ে এমন অবস্থানে আনতে হবে যেন মাঝের 8 টি ঘরে একই সংখ্যা থাকে। যেমন তিন নাম্বার বোর্ডে মাঝের 8 টি ঘরে 2 আছে। তোমাকে এই কাজ সবচেয়ে কম সংখ্যক অপারেশন ব্যবহার করে করতে হবে। এরকম সমাধান একাধিক থাকলে lexicographically smallest উত্তরটি দিতে হবে।



নকশা ৫.১: UVa 1343 এর বোর্ড

সমাধান: নিশ্চয় এসমস্যা নানা ভাবে সমাধান করা যায়। আমরা এখানে IDA* ব্যবহার করব। IDA* এর পূর্ণরূপ হলো iterative deepening A*. দুর্ভাগ্য বশত আমি আমার "প্রোগ্রামিং কন্টেস্ট" বইয়ে IDA* সম্পর্কে কিছুই লিখি নাই। তাই এখানে এ নিয়ে কিছু লিখছি।

IDA* বুঝার আগে আমাদের বুঝতে হবে searching কি। মনে কর আমাদের এই সমস্যায় শুরুর স্থান (state) একটি বোর্ড। এর সাথে কিছু অপারেশন দেওয়া থাকবে। কোনো এক স্থানে যদি কোনো অপারেশন কর তাহলে তুমি আরেক স্থানে যাবে। আমাদের লক্ষ্য হলো সবচেয়ে কম অপারেশন ব্যবহার করে আমাদের কাঙ্ক্ষিত লক্ষ্যে পৌঁছান। এটিই হলো searching. অর্থাৎ প্রদত্ত অপারেশন সমূহ ব্যবহার করে শুরুর স্থান হতে লক্ষ্যে সবচেয়ে কম খরচে পৌঁছানই হলো searching (আসলে সবচেয়ে কম খরচে পৌঁছান সবসময় লক্ষ্য থাকে না। কিন্তু আমাদের আলোচনার সুবিধার জন্য আমরা ধরে নেই সবচেয়ে কম খরচে পৌঁছাতে চাই)।

যদি উপরের প্যারা ভাল করে পরে থাকো তাহলে দেখবে এখানে তিনটি জিনিস আছে। এক- শুরুর স্থান, দুই- লক্ষ্য, তিন- অপারেশন। একটা জিনিস, লক্ষ্য কিন্তু একটা নির্দিষ্ট স্থান নাও হতে পারে। এটা হচ্ছে একটি rule যা দিয়ে তুমি বুঝতে পারবে যে তুমি তোমার লক্ষ্যে পৌঁছে গিয়েছ। যেমন এই সমস্যাতে লক্ষ্য কিন্তু নানা রকম হতে পারে। তবে লক্ষ্য বুঝার rule হলো মাঝের 8 টি ঘরে একই সংখ্যা থাকা। অন্য ঘর গুলিতে যা খুশি থাকুক তা নিয়ে মাথা ব্যাথা নেই।

তাহলে এই searching সমস্যা সমাধান করবা কীভাবে। একটি উপায় হতে পারে bfs বা dfs করে সমাধান করা। কিন্তু আমরা dfs করে সমাধান করতে পারব না। কেন? মনে কর A হতে B তে যাওয়া যায় (এখানে A, B, C ইত্যাদিকে state হিসাবে চিন্তা কর, এই সমস্যার সাথে এদের কোনো সম্পর্ক নেই)। আবার A হতে C হয়ে B তে যাওয়া যায়। এখন dfs তো চাইলে A-C-B এই রাস্তাতে

B তে যেতে পারে ফলে A-B যে ছোট রাস্তা আছে সেটা আবিষ্কার হবে না। সুতরাং dfs দিয়ে আসলে সবচেয়ে ছোট রাস্তা বের করা সম্ভব হয় না। তাহলে কি আমরা bfs দিয়ে সমাধান করতে পারি? হয়তো হবে। কিন্তু সমস্যা হলো আমাদের state সমূহ save করে রাখতে হবে তাদের visited মার্ক করতে হবে। এসব করতে হয়তো অনেক সময় লাগবে (গ্রাফে সার্চ করতে সময় লাগবে না, সময় লাগবে state মার্ক ও save করতে) এবং অনেক মেমরিও লাগবে।

এই সমস্যাকে দূর করার জন্য আছে IDA* search. IDA* search এ তেমন কোনো মেমরি লাগে না, state ও মার্ক করতে হয় না। বিনিময়ে search করতে একটু বেশি সময় লাগে এটাই সমস্যা। তোমরা যদি এখনো backtracking নিয়ে একদমই না পড়ে থাকো তাহলে তোমরা একটু পড়ে আসো। চাইলে নেট বা আমার আগের বইয়ে দেখতে পারো।

IDA* বলার আগে আমি তোমাদের ID-DFS নিয়ে বলব। এর পূর্ণ রূপ হলো iterative deepening depth first search. মনে কর আমাদের গ্রাফটি unweighted. আমাদের dfs এর বাহিরে একটি depthLimit এর লুপ থাকবে। প্রথমে এই depthLimit এর মান হবে 1. আমরা যেটা করব তাহল একটা dfs চালাব যার depth এর limit হবে depthLimit. অর্থাৎ dfs করার সময় আমরা এর বেশি depth এ যেতে পারব না। আরও বিস্তারিত বললে বলা যায়, সাধারণত আমাদের dfs ফাংশনের parameter এ থাকে x অর্থাৎ বর্তমান নোড। এর সাথে সাথে এখন আরও দুটি parameter থাকবে, current_depth এবং max_depth. প্রতিবার dfs কল করার সময় আমরা current_depth এর মান এক করে বাড়িয়ে দেব। আর যদি কখনো দেখি সেই মান max_depth এর থেকেও বড় হয়ে গিয়েছে তাহলে আর না এগিয়ে ফেরত আসব। এর ফলে লাভ কি হচ্ছে? লাভ হচ্ছে যে আমরা source নোড হতে target এ shortest path এ যেতে পারব। কেন? কারণ প্রথমে আমরা 1 max_depth দিয়ে চেষ্টা করব, এই ক্ষেত্রে যদি dfs আমাদের target এ পৌঁছায় তাহলে আমরা জানব 1 উত্তর। যদি তা না হয় তাহলে max_depth কে বাড়িয়ে 2 করব এবং আবারো dfs কল করব, এবার target এ পৌঁছাতে পারলে 2 উত্তর হবে। না হলে আবার বাড়িয়ে 3 করে dfs. এভাবে চলতে থাকবে যতক্ষণ না আমরা target পাচ্ছি। খেয়াল কর এই dfs এ আমরা visited বলে কিছু রাখছি না বা রাখার দরকার নেই। তাহলে এভাবে ID-DFS এর মাধ্যমে আমরা source হতে target এর shortest path বের করে ফেলতে পারব। সমস্যা কি তা তো বুঝতেই পারছ, একই state বার বার ভিজিট করছি, সময় বেশি লাগছে।

এখন চল আমরা আমাদের সমস্যায় একটু সময়ের জন্য ফেরত যাই। আমরা চাইলে আমাদের সমস্যাও ID-DFS এর মাধ্যমে সমাধান করতে পারি। যদিও তাতে অনেক বেশি সময় লাগবে। কিন্তু সমাধান তো হবে। প্রশ্ন হলো আমরা আমাদের সমাধানকে কীভাবে improve করতে পারি। উত্তর-heuristic. এটা আবার কি? এটি এমন একটি ফাংশন যাকে যদি তুমি একটি state দাও তাহলে এটি তোমাকে বলবে এই state হতে সমাধানে পৌঁছাতে কমপক্ষে আরও কত move লাগবে। এই "আরও কত move লাগবে" এই জিনিসের যেই heuristic যত ভাল approximation দিতে পারবে সেই heuristic তত ভাল। যেমন আমাদের সমস্যার একটি heuristic বের করা যাক। আমাদের সমস্যায় মাঝের 8 টি ঘরের মাঝে ধরা যাক x টিতে 1 নেই। তাহলে আমরা বলতে পারি অন্তত x move লাগবে সবগুলিতে 1 হতে। এরকম করে আমরা বের করতে পারি সবগুলি 2 এবং সবগুলি 3 হতে অন্তত কত cost লাগবে। এই তিনটি সংখ্যার মাঝে সবচেয়ে কমটিই আমাদের heuristic এর মান হতে পারে। কেন? কারণ এক move দিলে মাঝের একটি মাত্র ঘরেই তোমার কাঙ্ক্ষিত সংখ্যা আসতে পারে তাই না? (ব্যাপারটা ওত সহজ না, তাই একটু নিজে নিজে চিন্তা করে দেখো)

এখন প্রশ্ন হলো আমরা কীভাবে এই heuristic ব্যবহার করতে পারি। আমরা আমাদের ID-DFS এ ফিরে যাই। যদি আমাদের কাছে এরকম কোনো heuristic থাকে তাহলে আমরা কিছু optimization করতে পারি। মনে কর আমরা আছি current_depth এ আর আমাদের depth এর সর্বোচ্চ লিমিট হলো max_depth. এখন আমরা আমাদের state এর heuristic মান বের করব ধরা যাক h. যদি দেখি current_depth আর h এর যোগফল max_depth কে ছাড়িয়ে গিয়েছে তাহলে তো আর এই পথে গিয়ে লাভ নেই। তাই dfs আর না এগিয়ে ফেরত যাব। এখানে কি হয়েছে খেয়াল করেছ? তুমি জানো বর্তমান state এ আসতে তোমার কত ধাপ লেগেছে। আর তুমি heuristic থেকে বের করেছ যে আর কত ধাপ অবশ্যই লাগবেই (heuristic সবসময় তোমাকে "at least" মান দেয়)। এ থেকে তুমি খুব সহজেই বের করতে পারো যে এই রাস্তা ধরে গেলে তোমার target এ পৌঁছাতে কত ধাপ লাগবেই। এখন যদি দেখো এই ধাপ সংখ্যা তোমার max_depth কেও ছাড়িয়ে

যায় তাহলে তো আর এই পথে গিয়ে লাভ নেই তাই না? এটাই IDA*[১]। তুমি চাইলে কিছুক্ষণ আগে বলা heuristic কে কাজে লাগিয়ে আমাদের এই সমস্যাটি সমাধান করে ফেলতে পারো ঠিক আমি যেভাবে বললাম সেভাবে। খুব সহজ কোড।

কিছুক্ষণ আগে বললাম ঠিক এটাই IDA* না আরও কিছু আছে। এই আরও কিছু কি? তুমি চাইলে একটু চিন্তা করতে পারো আমাদের সমাধানে আর কোথায় উন্নতি করা যায়। খুব সহজ উন্নতি কিন্তু। মনে কর আমাদের দেওয়া max_depth এর মাঝে সমাধান নেই। তাহলে আমরা কি করছিলাম? max_depth কে এক বাড়াচ্ছিলাম। কিন্তু আমরা চাইলে এক না বাড়িয়ে একটু বেশি বাড়াতে পারি। আমরা তো অনেক অনেক নোড হতে current_depth আর h এর যোগফল দেখে ফেরত এসেছি তাই না? আমরা কিন্তু চাইলে এদের মাঝে সবচেয়ে ছোটটিকে আমাদের পরবর্তী max_depth হিসাবে set করতে পারি। অর্থাৎ, আমরা max_depth এ লিমিট থাকা অবস্থায় যেই সব নোড হতে ফেরত এসেছি সেইসব নোড হতে আমরা target এ পৌঁছানর একটা estimate পেয়েছিলাম। সেই সব estimate এর মাঝে সবচেয়ে ছোটটি আমরা পরের max_depth হিসাবে নিতে পারি। কারণ অন্য কোনো উপায়ে তো এর থেকে কমে target এ যাওয়া সম্ভব না তাই না? তাই আমরা চোখ বন্ধ করে max_depth না বাড়িয়ে, এরকম বুদ্ধি করে max_depth কে যদি বাড়াই তাহলেই আমাদের algorithm টি IDA* হয়ে যাবে। তোমরা চাইলে wiki তে IDA* এর pseudocode দেখে নিতে পারো বুঝার সুবিধার জন্য।

UVa 1374 Power Calculus

সমস্যা: মনে কর তোমাকে x দেওয়া আছে, তোমাকে x^n বের করতে হবে। এই কাজ তুমি নানা ভাবে করতে পারো। যেমন $n = 7$ এর ক্ষেত্রে, তুমি প্রথমে x^2 বের করতে পারো x কে x এর সাথে গুন করে। x^2 কে x^2 এর সাথেই গুন করে x^4 বের করতে পারো। এর পর একে x^2 এবং সবশেষে x এর সাথে গুন করে x^7 বের করতে পারো। এর ফলে মোট 4 টি গুন করা লাগলো। তা না করে তুমি চাইলে, x^4 বের করার পর আবারো একে নিজের সাথে গুন করে x^8 বের করতে পারো আর একে x দিয়ে ভাগ করে x^7 বের করতে পারো। এক্ষেত্রেও তোমার 4 টি গুন-ভাগ করতে হচ্ছে। তোমাকে n দেওয়া থাকবে ($1 \leq n \leq 1000$). বলতে হবে সবচেয়ে কম কতটি গুন ভাগ ব্যবহার করে তুমি x^n বের করতে পারবে।

সমাধান: প্রথম কথা এটি dp না। কারণ তুমি এখন x^a তে আছ এটাই শুধু গুরুত্বপূর্ণ নয়। এটি তুমি কীভাবে পেয়েছ, আর কি কি power তুমি জানো সেটাও গুরুত্বপূর্ণ। যেমন মনে কর তুমি x^{100} বের করার পথে যদি x^{25} বের করে থাকো তাহলে তুমি একবার ভাগ করেই x^{75} বের করতে পারবে। কিন্তু তুমি যদি x^{25} না বের করে থাকো, তাহলে হয়তো একটু কঠিন হয়ে যাবে x^{75} বের করার জন্য।

এই সমস্যা সমাধানের জন্য তোমাকে backtrack করতে হবে। এজন্য আমাদের কিছু optimization এবং heuristic বের করতে হবে। মনে কর এখন পর্যন্ত তুমি x^a বের করেছ সবচেয়ে বড়। আর তুমি x^n চাও। যদি কেউ জিজ্ঞাসা করে আর কত move লাগবেই। তুমি কি করবে? একটা উপায় হলো এক move পর খুব জোড় x^{2a} পাবে, দুই move পর x^{4a} এরকম করে b move পর খুব জোড় $x^{2^b a}$ পাবাই। এখন যদি হিসাব করে দেখো x^n পেতে হলে কমপক্ষে আর কত move লাগবে অর্থাৎ সবচেয়ে ছোট কোন b এর জন্য $2^b a \geq n$ হবে তাহলে সেই মানকেই তোমরা heuristic হিসাবে ধরতে পারো।

আরেকটি বেশ গুরুত্বপূর্ণ optimization আছে। আর তাহলো সবসময় আগের move এ যেই x^a পাওয়া গিয়েছে তাকে ব্যবহার করে সমাধান পাওয়া যাবে। অর্থাৎ আগের move এ তুমি x^{10} পেয়েছ, তুমি এই move এ সেটি ব্যবহার করবে। এটা কেমন জানি। এর প্রমাণ কি? এমনও তো হতে পারে যে অন্য দুটি power কে গুন ভাগ করে যা পাওয়া যায় তার সাথে পরে হয়তো x^{10} কে গুন ভাগ করতে হবে। হতে পারে। তাহলে কি করা যায়? একটি উপায় হল বসে বসে প্রমাণ করা যে আসলেই এই optimization সঠিক। অথবা তুমি 1 হতে 1000 পর্যন্ত এই optimization সহ এবং বাদে দুই ভাবেই উত্তর বের করে দেখো। যদি দেখো যে সব ক্ষেত্রেই উত্তর একই আসে, তার মানে ধরে নিতে পারো যে তোমার optimization ঠিক আছে।

[১] উম... ঠিক এটাই না, আরও একটু আছে, পরে বলছি

মনে হয় এই দুই optimization / heuristic ব্যবহার করে IDA* বা backtrack করলে accepted হয়ে যাবে।

UVa 1602 Lattice Animals

সমস্যা: তোমরা কি tetris খেলেছ? যদি tetris খেলে না থাকো তাহলে এই সমস্যাটি ছবি ছাড়া কম কথায় বুঝানো একটু কঠিন হয়ে যাবে। তাই তোমরা সাইটে পড়ে নিতে পারো। n সাইজের polymino হল n টি বর্গ পাশাপাশি বসিয়ে একটি connected জিনিস বানান। আমরা দুটি polymino কে একই বলব যদি একটি থেকে আরেকটি ঘুরিয়ে বা flip (mirror) করে বা translate করে পাওয়া যায়। যেমন $n = 3$ এর জন্য দুই রকম polymino পাওয়া যায়। তিনটি বর্গ পাশাপাশি, অথবা L এর মত করে তিনটি বর্গ লাগান। আর অন্যগুলি হয় rotate করে বা flip করে বানানো সম্ভব। তোমাকে n, w এবং h দেওয়া আছে বলতে হবে $w \times h$ সাইজের একটি grid এ কতগুলি n সাইজের ভিন্ন polymino পাওয়া সম্ভব। $(1 \leq n \leq 10, 1 \leq w, h \leq n)$

সমাধান: আগের দুটি সমস্যা হতে এই সমস্যার মুল পার্থক্য হলো আগের দুটি সমস্যায় একটি সমাধান বের করতে বলা হয়েছে বা shortest path এ সমাধান বের করতে বলা হয়েছে। কিন্তু এখানে মোট কতগুলি সমাধান আছে তা বের করতে বলেছে। সুতরাং এক্ষেত্রে IDA* বা BFS/DFS কাজে আসবে না। হয় আমাদের dp বা mathematically counting করে বের করতে হবে, নাহলে backtrack করে বের করতে হবে। এক্ষেত্রে আসলে mathematical ভাবে counting করা মনে হয় না খুব একটা সহজ হবে। আর তাছাড়া n এর সাইজও কম আছে। তাই এখানে backtrack বা simulation জাতীয় পদ্ধতিই উপায়।

আমাদের সমাধানে n টি লিস্ট থাকবে। i তম লিস্টে থাকবে i সাইজের সকল ভিন্ন ভিন্ন polymino. ভিন্ন মানে এদের কাউকে flip, rotate বা translate করলে এই লিস্টের অন্য কোনো polymino পাওয়া যাবে না। এইসব polymino এর সাথে আরও একটি cell সকল ভাবে জুড়ে দিয়ে i+1 সাইজের polymino গুলি পাওয়া যাবে। কিন্তু দেখা যাবে এদের একটিকে translate, rotate বা flip করে অন্য আরেকটা পাওয়া যায়। তাই আমাদেরকে i+1 হতে i+2 সাইজের polymino গুলি বের করার আগেই i+1 এর লিস্ট হতে এসব similar polymino দের দূর করতে হবে। উদাহরন স্বরুপ মনে কর আমাদের কাছে একটি polymino আছে $(2, 2), (2, 3)$. এখন আমরা একটি করে নতুন cell লাগিয়ে একে $(2, 2), (2, 3), (2, 4)$ বানাতে পারি, আবার $(2, 1), (2, 2), (2, 3)$ ও বানাতে পারি। কিন্তু এই দুটি কিন্তু আবার একই (translation সাপেক্ষে)। এদের দুজনকেই রেখে আমাদের লাভ নেই। সুতরাং আমরা যদি একটি polymino এর লিস্ট থেকে শুধু ভিন্ন ভিন্ন polymino দের রাখতে পারি তাহলে আমরা 1 সাইজ দিয়ে শুরু করে একে একে 2, 3, 4 এরকম সকল সাইজের জন্য list পেয়ে যাব। প্রশ্ন হলো কীভাবে আমরা এই ভিন্ন ভিন্ন polymino গুলি পেতে পারি বা দুটি polymino যে একই তা কীভাবে বুঝতে পারি।

এজন্য আমাদের যা করতে হবে তাহল polymino দেরকে canonical ফর্মে প্রকাশ করতে হবে। Canonical ফর্ম মানে হল প্রতিটি polymino কে এমন ভাবে প্রকাশ করা যেন একই রকম (এক্ষেত্রে flip, translate আর rotate এর সাপেক্ষে) polymino দেরকে canonical ফর্মে প্রকাশ করলে একই canonical ফর্ম নেয়। এতে লাভ কি? লাভ হল কোনো একটি polymino এর লিস্টে সকল polymino এর জন্য যদি আমরা canonical ফর্ম বের করি তাহলে ভিন্ন ভিন্ন canonical ফর্ম ওয়ালা polymino নিলেই আমাদের ভিন্ন polymino এর লিস্ট হয়ে যাবে। এখন প্রশ্ন হল একটি polymino কে কীভাবে canonical ফর্মে represent করা যায়। একটি polymino কে আমরা square গুলির coordinate সম্বলিত একটি set হিসাবে represent করতে পারি। কিন্তু এটি কিন্তু canonical form না। কারণ চিন্তা কর 1 সাইজের একটি polymino $(0, 0)$ হতে পারে অথবা $(1, 1)$ হতে পারে বা অন্য কোনো কিছু। কিন্তু এই সবগুলিই একই (translation সাপেক্ষে)। তাহলে এদেরকে canonical ফর্মে কীভাবে নেয়া যায়?

যাতে translation এর ফলে একাধিক polymino কে ভিন্ন বিবেচনা না করি সেজন্য একটি সহজ উপায় হলো polymino এর bounding rectangle এর একটি নির্দিষ্ট কোণা (ধরা যাক নিচের বাম কোণা) কে নিয়ে $(0, 0)$ তে বসাতে হবে। এজন্য তোমাকে square গুলির সবচেয়ে কম x নিতে হবে এবং সবচেয়ে কম y. এরপর সব square হতে তুমি যদি এই (x, y) বিয়োগ কর তাহলেই

canonical form পেয়ে যাবা যা তোমাকে translation এর হাত হতে রক্ষা করবে। উদাহরন- $(0,0)$ আর $(1,2)$ এই দুজনের ক্ষেত্রে তুমি যদি উপরের কাজ কর তাহলে কিন্তু দুই ক্ষেত্রেই $(0,0)$ পাবে, এর মানে আমরা দুজনকে রাখব না, এদের এক জনকে রাখলেই চলে। Rotation বা flip এর জন্য কি করা যায়? এক্ষেত্রেও নানা উপায় আছে। একটি উপায় হলো polymino কে rotate কর (4 ভাবে rotate করা যায়, 0 বার, 1 বার, 2 বার আর 3 বার 90 ডিগ্রী rotate করা), flip করা (শুধু যেকোনো একটি flip দিলেই চলবে horizontal বার vertical), flip করার পর rotate করা (আবারো 4 ভাবে rotate)- এভাবে যতগুলি (মোট 8 রকম) polymino পাবে সবগুলিই তো আমাদের সমস্যা মতে একই তাই না? এদের প্রত্যেকের translation সাপেক্ষে canonical form বের কর আর তাদের একটি সেটে রাখ। খেয়াল কর একটি polymino বা তার translation সাপেক্ষে canonical ফর্ম কিন্তু নিজেই একটি cell এর সেট। আমরা এই polymino গুলির জন্য যদি একটি সেট নেই তাহলে এটি হবে cell এর সেটের সেট।

আমরা তাহলে কি করছি? আমরা একটি লিস্টের প্রত্যেক polymino নিচ্ছি, নিয়ে তাদের বিভিন্ন ভাবে rotate আর flip করে তাদের translation সাপেক্ষে canonical ফর্ম বের করছি। এর পর সেই canonical ফর্মকে একটি সেটে পুরে দিচ্ছি। এই কাজ সব polymino এর জন্য করে ফেললে আমরা আমাদের সেটে কিন্তু সকল ভিন্ন ভিন্ন polymino পেয়ে যাব।

তাহলে আমাদের সমাধান প্রায় শেষ, শুধু একটি জিনিস বাকি আছে- কীভাবে নতুন নতুন polymino পাওয়া যাবে। আমরা ইতোমধ্যেই 1 সাইজের সকল polymino পেয়ে গিয়েছি। এখন আমরা চেষ্টা করব i সাইজের polymino গুলি হতে $i+1$ সাইজের polymino বের করার। যদি তা সম্ভব হয় তাহলে আমরা $1, 2, \ldots n$ সাইজের সকল polymino বের করে ফেলতে পারব। i সাইজের polymino হতে $i+1$ সাইজের polymino বের করার উপায় হলো i সাইজের polymino এর প্রতিটি square এর চারিদিকে নতুন square ($i+1$ তম, কারণ ইতোমধ্যেই i টি square আছে) বসানোর চেষ্টা করা। তাহলে আমরা একটি একটি করে নতুন polymino পাবো $i+1$ সাইজের। চেষ্টা করার কথা বললাম কারণ এমনও তো হতে পারে যে যেখানে বসাতে চাচ্ছ সেখানে আগে থেকেই একটি square আছে। সুতরাং সেই ব্যাপারে একটু সাবধান হতে হবে। তাহলে এভাবে আমরা $i+1$ সাইজের জন্য একটি লিস্ট পেয়ে যাচ্ছি। এরপর এদের থেকে আমাদের একই রকমের polymino গুলি বাদ দিতে হবে (উপরে canonical ফর্ম নিয়ে অনেক বকবক করেছি)। এভাবে আমরা একে একে n সাইজের লিস্ট পেয়ে যাব।

আর $w \times h$ এটা কোনো ব্যাপার না। একবার সব polymino পেয়ে গেলে তুমি সব $w \times h$ এর জন্য উত্তর বের করে রাখতে পারবে। অর্থাৎ তোমার কোড একবার মাত্র n সাইজের সব polymino বের করবে এরপর তুমি একটি $n \times w \times h$ সাইজের একটি matrix কে precompute করে রাখবা যেখানে লিখা থাকবে n সাইজের কয়টি polymino আছে যারা $w \times h$ সাইজে আঁটে আর তারা rotation, flip বা translation সাপেক্ষে একই জিনিস দেয় না।

UVa 225 Golygons

সমস্যা: একটি গ্রিডে কিছু কিছু বিন্দুতে যাওয়া নিষেধ। সেসব বিন্দু ইনপুটে দেওয়া থাকবে। এখন তোমাকে মোট n ধাপ দিতে হবে ($n \leq 20$)। প্রথম ধাপের সাইজ 1, দ্বিতীয় ধাপের সাইজ 2 এরকম করে n ধাপ। তুমি শুরু করবে $(0,0)$ হতে এবং প্রথম ধাপ তুমি যেকোনো দিকে দিতে পারো। এই ধাপ দেবার পর তোমাকে অবশ্যই ডানে বা বামে ঘুরতে হবে। যেমন তুমি 1 ধাপ দিলে positive y axis বরাবর। এরপর তোমাকে হয় positive x অথবা negative x axis এর দিকে ঘুরে তাকাতে হবে এবং সেই দিকে তুমি তোমার দ্বিতীয় ধাপ দেবে (2 সাইজের)। আবারো তুমি ডানে বা বামে ঘুরবে এবং সেই দিকে 3 সাইজের ধাপ দিবে। এভাবে n বার। খেয়াল করতে হবে এই ধাপ দেবার সময় যেন তুমি নিষিদ্ধ বিন্দুতে বা নিষিদ্ধ বিন্দুর উপর দিয়ে না যাও। বলতে হবে এমনভাবে n ধাপ দেওয়া সম্ভব কিনা যেন কোনো নিষিদ্ধ বিন্দুকে স্পর্শ না করে আবার $(0,0)$ তে ফিরে আসা যায়।

সমাধান: প্রথম ধাপ যদি তুমি উপরে বা নিচের দিকে দাও, তাহলে সকল বিজোড়তম ধাপ তুমি হয় উপরে নাহয় নিচে দিবে আর জোড়তম ধাপ হয় বামে না হয় ডানে দেবে। একই ভাবে যদি প্রথম ধাপ তুমি বামে বা ডানে দাও তাহলে কি হবে তা বুঝতেই পারছ। আপাতত ধরে নাও আমরা প্রথম ধাপ উপরের দিকে দিয়েছি। এখন মনে কর তুমি কিছু ধাপ দেবার পর (x,y) এ আছ। এখন উপর

দিকে d ধাপ দেওয়া মানে $(x, y + d)$ তে যাওয়া, এভাবে ডান বাম নিচ সব কিছুর জন্য আসলে তুমি coordinate এর হিসাব নিকাশ করবা। এখন খেয়াল কর, তোমার কাছে যত বিজোড় ধাপ বাকি আছে সেই সকল বিজোড় ধাপের সাইজ গুলি যোগ করে যদি দেখো তা $abs(y)$ এর থেকে ছোট, এর মানে যতই চেষ্টা কর তুমি $y = 0$ করতে পারবে না। একই ভাবে তুমি সকল জোড় ধাপের সাইজ যোগ করে দেখবে তার মান $abs(x)$ এর থেকে ছোট হয় কিনা। হলে $x = 0$ করতে পারবে না। এটা একটা optimization. এছাড়াও খেয়াল কর backtrack এর সময় যখন তুমি ধাপ দেবে তখন জানার প্রয়োজন হবে যে (x, y_1) হতে (x, y_2) (বা একই ভাবে x axis বরাবর) এর মাঝে কোনো নিষিদ্ধ বিন্দু আছে কিনা। তুমি চাইলে প্রতি row এবং column এর জন্য একটি consecutive sum এর অ্যারে রেখে এই তথ্য $O(1)$ এ বের করে ফেলতে পারো। এই কয়টি optimization যদি ব্যবহার কর তাহলেই আমার মনে হয় এই সমস্যা সমাধান হয়ে যাবে।

৫.১ অনুশীলনী

৫.১.১ সমস্যা

Easy

- UVa 140 Bandwidth
- UVa 818 Cutting Chains
- UVa 12113 Overlapping Squares
- UVa 11214 Guaring the Chessboard
- UVa 817 According to Bartjens
- Uva 211 The Domino Effect
- UVa 690 Pipeline Scheduling
- UVa 1575 Factors
- UVa 1533 Moving Pegs
- UVa 11882 Biggest Number
- UVa 12569 Planning mobile robot on Tree (Easy Version)

Medium

- UVa 1354 Mobile Computing
- UVa 12107 Digit Puzzle
- UVa 11694 Gokigen Naname
- UVa 10837 A Research Problem
- LOJ 1143 Knights in FEN
- UVa 1603 Square Destroyer
- UVa 11846 Finding Seats Again
- UVa 10384 The Wall Pushers
- LOJ 1121 15 Puzzle
- LOJ 1397 Sudoku Solver

৫.১.২ হিন্ট

UVa 140: মাত্র 8 টি নোড। তাই তোমরা all possible permutation বের করে চেক করতে পারো। এজন্য backtrack করতে পারো আবার চাইলে algorithm হেডার ফাইলের next permutation ও ব্যবহার করতে পারো।

UVa 211: তুমি একটি নির্দিষ্ট order এ domino বসাবে তাহলেই হবে। যেমন $(1, 1)$, $(1, 2)$, $(1, 3)$, $\ldots$ $(1, c)$, $(2, 1)$ $\ldots$ $(2, c)$ $\ldots$ $(r, 1)$, $\ldots$ (r, c) এই অর্ডারে তুমি cell গুলিতে যাবে এবং একটি domino বসানোর চেষ্টা করবে। তাহলে সুবিধা হলো প্রতিবার তোমার হাতে মাত্র দুটি option. হয় এই ঘরের সাথে তার পাশের ঘরের, অথবা এই ঘরের সাথে নিচের ঘরের। আর যদি এই ঘর আগে থেকেই পূরণ হয়ে থাকে তাহলে তো এটা চেষ্টা করার কোনো মানে নেই।

UVa 818: সমস্যাটি বেশ সুন্দর, কিন্তু এর statement টা ambiguous। কিন্তু এর পরও দিলাম। কোন কোন link খুলতে হবে এর উপরে bruteforce করার পর একটি graph theory বা adhoc বা greedy (কীভাবে একটি সমস্যা এতো ক্যাটাগরির হতে পারে?- আসলে এজন্যই আমার এই সমস্যাটি ভাল লেগেছে এবং তোমাদের জন্য এটি দিলাম) সমস্যা সমাধান করতে হয়। এই পরিবর্তিত সমস্যাটি বেশ সুন্দর। আমার মনে হয় তোমাদের এটি নিজে থেকেই সমাধান করা উচিত।

UVa 690: খুব একটা কঠিন না। কিন্তু কোড একটু ঝামেলার মনে হতে পারে। Reservation table ইনপুট নেবার পর প্রথম কাজ হবে একে কয় ঘর shift করে বসানো সম্ভব তা বের করা। n ঘর shift করে তো বসানো সম্ভব কারণ n ঘর সরালে তাদের মাঝে আর কোনো overlap ই থাকে না। আর কোন কোন shift করে বসানো সম্ভব সেটি বের করে রাখতে হবে শুরুতেই। এরপর এর উপর ভিত্তি করে backtrack. কিন্তু এই যে বসানো সম্ভব কিনা বের করা, এই অংশটুকু আমরা কীভাবে খুব দ্রুত করতে পারি? আমরা চাইলে bitmask ব্যবহার করতে পারি। মনে কর আমাদের কাছে একটি bitmask আছে। এখন আমরা জানতে চাচ্ছি reservation table কে c ঘর সরিয়ে বসালে overlap করে কিনা। এটা আসলে c ঘর shift করে bitwise and করে দেখার মত।

UVa 12113: সুন্দর সমস্যা, যদিও কোড একটু কঠিন হতে পারে। প্রথমত খেয়াল কর তুমি মাত্র 9 ভাবে 2×2 square কে বসাতে পারবে। আর বলাই আছে যে 6 বারের বেশি বসাবে না। অর্থাৎ মোট $9^6 < 10^6$ ভাবে বসানো সম্ভব। সুতরাং তুমি চাইলে শুরুতেই সব ভাবে বসিয়ে বসিয়ে shape গুলি hash করে রাখতে পারো। অথবা শুরুতেই hash এর ঝামেলায় না গিয়ে সকল shape বের করে একটি অ্যারেতে রেখে মিলিয়ে দেখতে পারো ac হয় কিনা।

UVa 817: 4^8 কিন্তু খুব একটা বেশি না। অর্থাৎ তোমরা প্রথম 8 টি character এর পর কী বসাবা (বা হয়তো বসাবা না) তার উপর bruteforce করলেও হবে।

UVa 11882: একটি heuristic হতে পারে তুমি কোনো জায়গায় পৌঁছানর পর দেখা যে আর কয়টি জায়গায় খুব জোড় যেতে পারবে (ধর bfs করে)। IDA* মনে হয় না করাই ভাল কারণ তোমাকে সবচেয়ে বড় সংখ্যা বের করতে হবে, এখানে goal কিন্তু নির্দিষ্ট না। অর্থাৎ তুমি কোনো একটা সংখ্যা দেখেই বুঝতে পারবা না যে এটাই উত্তর কিনা। তুমি সব সংখ্যা দেখার পর বুঝবে কোনটা উত্তর। এখন মনে কর তুমি ইতোমধ্যেই 2345 পেয়ে গেছ। আর এখন 12 নেবার পর দেখছ আর দুটি অংকের বেশি পাওয়া যাবে না। এর মানে আমাদের আর এগিয়ে লাভ নেই। এভাবে pruning (pruning মানে হলো backtrack এর সময় আর না এগিয়ে ফিরে যাওয়া) করলেই হয়ে যাবার কথা।

UVa 1354: মাত্র 6 টি জিনিস থাকতে পারে। তাই তুমি আসলে সকল ভাবে tree বানানোর চেষ্টা করতে পারো। তুমি dynamic programming করার মত করে backtrack করলে কিন্তু হবে না। অর্থাৎ, যদি তুমি এরকম কর- root এ তুমি সব জিনিস দিলে, এখানে তুমি সিদ্ধান্ত নিলে ডানে কে যাবে বামে কে যাবে, এরপর তুমি বামে আর ডানে recursive call করলে ঐ সব জিনিস দিয়ে- না তা হবে না। কারণ বামের জিনিস দিয়ে বিভিন্ন ভাবে tree বানানো যায় আবার ডানের জিনিসগুলি দিয়েও অনেক ভাবে tree বানানো যায়। সুতরাং তুমি বামের কোন tree এর সাথে ডানের কোন tree কে মেলাবে সেটা একটা কঠিন ব্যাপার। তবে হ্যাঁ, তুমি যদি বাম বা ডান যেকোনো দিকের জন্য যদি যত ভাবে tree বানানো যায় তাদের প্রতিটিকেই (বা তাদের যেই information গুলি তোমার আসলে লাগবে) return করতে পারো তাহলেই সমস্যা সমাধান হয়ে যাবে। কোনো নোডের জন্য আসলে শুধু এটুকু জানলেই হয় যে তার subtree তে থাকা নোডগুলি বাম দিকে এবং ডান দিকে কতদূর পর্যন্ত যায়। তাহলে দুদিকের মোট ভর আর দুদিকের subtree দুটি দুদিকে কতদূর যায় এই দুটি তথ্য জানা থাকলে বর্তমান নোডের জন্যও এই তথ্য তৈরি করে return করতে পারবে।

UVa 1603: প্রতিটি ম্যাচ কাঠির জন্য লিখে রাখবে কতগুলি square এর border এ সেই কাঠি আছে। Backtrack এ কাঠিগুলিকে তার উপর দিয়ে যাওয়া square এর সংখ্যার ভিত্তিতে বড় হতে ছোট তে সর্ট করবে এবং এই order এ তাদেরকে remove করার চেষ্টা করবে। কেন? স্বাভাবিক ভাবে চিন্তা কর, যেই ম্যাচ কাঠি সরালে বেশির ভাগ square নষ্ট হবে তাকে সরানোই তো সাধারণত লাভ জনক তাই না? আর কি optimize করা যায়? যদি দেখো এখন যেসব কাঠি বাকি আছে তাদের প্রতিটি দিয়ে একের বেশি square যায় না তাহলে যেকয়টি square বাকি আছে ঠিক তত সংখ্যক কাঠি তোমাকে সড়াতে হবে। সুতরাং আর backtrack না করে তুমি কয়টি square বাকি আছে সেই count দেখতে পারো। তবে যেহেতু n এর মান কম তাই এসব optimization নাও করা লাগতে পারে। তাও এসব বিভিন্ন রকম optimization জানা থাকা ভাল, কাজে লেগে যেতে পারে অন্য কোনো সমস্যায়।

অধ্যায় ৬

Data Structure সম্পর্কিত সমস্যা

UVa 210 Concurrency Simulator

সমস্যা: একটি প্রোগ্রামে 5 ধরনের instruction থাকতে পারে। "variable = constant", "print variable", "lock", "unlock", "end". এখানে variable a হতে z এর মাঝের যেকোনো character হতে পারবে এবং constant 0 হতে 99 এর মাঝের যেকোনো সংখ্যা হতে পারবে। আমাদের ইনপুটে সর্বোচ্চ 10 টি প্রোগ্রাম থাকবে এবং প্রতিটিতে সর্বোচ্চ 25 টি instruction থাকবে। প্রোগ্রামগুলি এক core ওয়ালা machine এ চলবে। তবে প্রোগ্রামগুলি একটির পর একটি এভাবে চলবে না। বরং একটি নির্দিষ্ট সময়ের লিমিট আছে ধরা যাক T. প্রথম T সময় প্রথম প্রোগ্রাম, দ্বিতীয় T সময় দ্বিতীয় প্রোগ্রাম, এভাবে চলতে থাকবে এবং সকল প্রোগ্রাম হতে T সময় চলার পর আবার প্রথম প্রোগ্রামে ফিরে আসবে। এভাবে cyclically চলতে থাকবে। প্রতিটি instruction চলতে কত সময় করে লাগে তা বলা আছে। যদি কোনো instruction চলতে চলতে T সময় পার হয়ে যায় তাহলে সেই instruction শেষ না হওয়া পর্যন্ত পরের প্রোগ্রামে যাবে না।

তবে একটা ব্যাপার আছে। একাধিক প্রোগ্রাম একই সাথে lock করতে পারবে না। যদি অন্য কোনো প্রোগ্রাম এখন lock না করে থাকে তাহলে সে lock করতে পারবে। যদি কেউ lock করতে চায় এবং অন্য কেউ ইতোমধ্যেই lock করে থাকে তাহলে সে block queue তে ঢুকবে। শুরুতে সবাই ready queue তে থাকে এবং cpu এই ready queue এর শুরুর প্রোগ্রাম T সময় প্রসেস করার পর ready queue এর শেষে রাখে। তবে এই lock এর ক্ষেত্রে পরের যে প্রোগ্রাম lock করতে চায় তাকে ready queue এর শেষে না রেখে block queue এর শেষে রাখে। এভাবে চলতে থাকে। যেই প্রোগ্রামটি শুরুতে lock করেছিল সে যখন unlock করবে তখন block queue এর শুরুর প্রোগ্রাম ready queue এর শেষে রাখতে হবে। এভাবে সকল প্রোগ্রাম শেষ না হওয়া পর্যন্ত চলতে থাকবে। তোমাকে এই প্রোগ্রাম গুলি দিয়ে যা প্রিন্ট হবে তা আউটপুট দিতে হবে।

একটি জিনিস আগে বলা হয় নাই তাহলো, এখানে variable গুলি shared. অর্থাৎ যদি একটি প্রোগ্রাম কোনো একটি ভ্যারিয়াবেল এর মান পরিবর্তন করে তাহলে অন্য প্রোগ্রামেও তা পরিবর্তন হয়ে যাবে। আর শুরুতে সকল ভ্যারিয়েবলের মান 0. নিচে sample input আর output দেওয়া হলো। এখানে প্রথমের সংখ্যাটি বলছে কয়টি প্রোগ্রাম, পরের 5 টি সংখ্যা হলো উপরের 5 টি instruction এর জন্য কত করে সময় লাগে (একই অর্ডারে), আর শেষের সংখ্যাটি হচ্ছে T এর মান। আর প্রতিটি প্রোগ্রাম end দ্বারা পৃথক করা আছে তাতো বুঝতেই পারছ (মনে রেখো end কিন্তু একটি instruction)।

Sample input:
```
3 1 1 1 1 1 1
a = 4
print a
lock
b = 9
print b
unlock
print b
end
a = 3
print a
lock
b = 8
print b
unlock
print b
end
b = 5
a = 17
print a
print b
lock
b = 21
print b
unlock
print b
end
```

Output of the sample input:
```
1: 3
2: 3
3: 17
3: 9
1: 9
1: 9
2: 8
2: 8
3: 21
3: 21
```

সমাধান: সহজ একটি সমস্যা, statement ই যা কঠিন। তুমি দুই ধরনের queue রাখবে- running queue আর blocked queue। চাইলে stl এর queue ব্যবহার করতে পারো। এই দুই ধরনের queue তে তোমার প্রোগ্রামের index রাখবে। শুরুতে running queue তে 1 হতে n পুশ করে নেবে যেখানে n হলো মোট প্রোগ্রামের সংখ্যা। সেই সাথে একটি global array রাখবে যে কোন প্রোগ্রামের কততম instruction পর্যন্ত ইতোমধ্যেই run করা হয়েছে। প্রতিবার তোমরা running queue এর সামনে থেকে প্রোগ্রামের index নেবে। এরপরের কাজ হলো সেই প্রোগ্রাম কে run করানো। তুমি সেই global array থেকে দেখবে কোন instruction তোমাকে run করাতে হবে। যদি assignment বা print instruction হয় তাহলে তো সহজ। যদি end হয় তাও সহজ, এই প্রোগ্রামকে আর তুমি running queue তে ঢুকাবে না। এখন মুল ঝামেলা হলো lock আর unlock নিয়ে। যখন তুমি unlock করবে তখন সমস্যায় যা বলেছে তা করবে, অর্থাৎ blocked queue এর শুরুর প্রোগ্রামকে running queue এর শেষে ঢুকাবে। আর যখন lock এ আসবে, তখন দেখবে আগে থেকে যদি কেউ lock থাকে তাহলে এই lock execute না করে একে blocked queue তে ঢুকিয়ে চলে যাবে। একে কিন্তু আর running queue তে ঢুকাবে না। আর যদি দেখো কেউ lock নেই তাহলে তো সহজ, একেই execute করতে থাকবে। যেহেতু তোমাকে জানতে হবে আর কেউ lock আছে কিনা, এ জন্য একটি বাড়তি flag রাখতে হবে। আর unlock এ এই flag কে clear করতে ভুলে যেও না।

আর সময়ের ব্যাপারটাতো সহজ। তুমি প্রতিটি instruction run করবা আর T হতে সেই সময় বাদ দিবে। যদি নতুন কোনো instruction run করতে গিয়ে দেখো T এর মান 0 বা negative হয়ে গেছে এবং আরও instruction বাকি আছে তাহলে এই প্রোগ্রামকে running queue এর পেছনে ঢুকায়ে চলে গেলেই হবে।

মোট কথা এই সমস্যা বেশ সহজ। তোমাকে ধৈর্য ধরে কোডটি করতে হবে, তাহলেই হবে।

UVa 514 Rails

সমস্যা: একটি ট্রেন স্টেশনে একটি মাত্র ট্রেন লাইন আছে। এই লাইনে A লাইন থেকে বগি আসে, স্টেশনে ঢুকে এরপর বেরুনোর সময় B লাইনের দিকে যায়। জিনিসটা কিছুটা Y এর মত দেখতে। মনে কর Y এর বামে A আর ডানে B। A হতে একে একে বগি আসবে এসে Y এর নিচে একের পর একে জড় হবে। মাঝে মাঝে তুমি চাইলে নিচ হতে B এর দিকে বগি দিতে পারবে। যেমন মনে কর A এর দিকে 1, 2, 3 এই তিনটি বগি আছে। প্রথমে 1 কে এনে নিচে রাখলে, এর পর 2 কে এনে নিচে রাখলে, এর পর মনে কর 2 কে B এর দিকে দিলে, এর পর 3 কে A হতে নিচে এনে B এর দিকে দিলে। সবশেষে 1 কে B এর দিকে দিলে। তাহলে শুরুতে 1, 2, 3 ছিল এখন 2, 3, 1 হয়ে গেছে। তোমাকে বলা আছে A এর দিকে কয়টি বগি আছে। ধরা যাক n টি বগি ($1, 2, \ldots n$ এরকম অর্ডারে) আছে। তোমাকে বলতে হবে B এর দিকে বগিগুলি প্রদত্ত order এ নেওয়া সম্ভব কিনা। n এর মান খুব জোড় 1000.

সমাধান: এটিও বেশ সহজ একটি সমস্যা। যদি stack জানা থাকে তাহলে আরও সহজ হবে। মনে কর B এর শুরুতে i আছে। একে নেবার একটিই মাত্র উপায় আছে। আর তাহলো A এর দিক থেকে 1 হতে i পর্যন্ত সকল বগিকে এনে Y এর নিচে রাখা এবং i কে B এর দিকে নেওয়া। এবার B এর পরের সংখ্যা দেখো কত। যদি দেখো সেটি নিচে রাখা সংখ্যাগুলির মাঝে সবচেয়ে উপরে আছে তাহলে তো হয়েই গেল। আর যদি উপরে না থেকে ভেতরে থাকে তাহলে তো আর হবে না। যেমন $i = 3$ হবার পর এখন যদি তুমি 1 চাও তাহলে তো হবে না। কারণ 1 এর উপর 2 আছে। 2 কে না নিয়ে 1 নেওয়া সম্ভব না। সুতরাং এই ক্ষেত্রে B এর sequence পাওয়া সম্ভব না। আর যদি B এর পরবর্তী সংখ্যাটি Y এর নিচে না থেকে A এর দিকে থেকে থাকে তাহলে সেই সংখ্যা পর্যন্ত Y এর নিচে নিতে হবে। এভাবে যদি তুমি B এর সব সংখ্যা নিতে পারো তাহলেই হয়ে যাবে। এখন একটা জিনিস, Y এর উপরের সংখ্যা দেখা সহজ, কিন্তু Y এর ভেতরে কোনো একটি নির্দিষ্ট সংখ্যা আছে কিনা তা দেখতে হলে তো লুপ চালাতে হবে। তাহলে উপায়? যেহেতু n এর মান কম, সেহেতু আমরা প্রতিবার লুপ চালাতেই পারি। কিন্তু এটা কি লুপ না চালিয়ে করা সম্ভব? হ্যাঁ, তুমি চাইলে একটি অ্যারে নিয়ে লিখে রাখতে পারো কোন সংখ্যা কখন কই আছে। যখন কোনো সংখ্যাকে Y এর নিচে রাখবে তখন লিখে রাখবে যে সেটি নিচে আছে। তাহলে এই অ্যারে দেখে তুমি খুব সহজেই বুঝে যাবে যে সেটি Y এর নিচে আছে। এর থেকেও একটি সহজ উপায় আছে। তাহলো আমাদের সমাধানের জন্য কিন্তু এমনিতেই লিখে রাখতে হবে A এর দিকে কোন পর্যন্ত সংখ্যা নেওয়া হয়ে গেছে। ধরা যাক j পর্যন্ত নেওয়া হয়ে গেছে। এর মানে A এর দিকের বাকি সংখ্যা হলো $j + 1$ হতে n পর্যন্ত। আমরা কোনো একটি সংখ্যা Y এর নিচে আছে কিনা তা না দেখে A তে নেই কিনা দেখলেও হয়। সুতরাং এভাবে আমরা আমাদের সমস্যা $\mathrm{O}(n)$ এ সমাধান করতে পারি।

UVa 442 Matrix Chain Multiplication

সমস্যা: কিছু matrix এর নাম এবং তাদের dimension দেওয়া আছে। সেই সাথে কিছু matrix গুনের expression দেওয়া আছে। যেমন- (A(BC)), ((AB)(CD) ইত্যাদি। তোমাকে বলতে হবে প্রদত্ত expression এর মত করে গুন করতে কত cost. A(p, q) এবং B(q, r) দুটি matrix গুন করার cost হলো $p \times q \times r$. সর্বোচ্চ 26 টি ম্যাট্রিক্স দেওয়া থাকতে পারে।

সমাধান: অনেক ভাবেই এই সমস্যা সমাধান করা যায়। এখানে আমরা দেখবো কীভাবে stack ব্যবহার করে এই সমস্যা আমরা সমাধান করতে পারি।

সমস্যাকে একটু সহজ করে নেই। মনে কর তোমার কিছু bracket sequence দেওয়া আছে, তোমাকে বলতে হবে এই bracket sequence টি balanced কিনা। যেমন (()()) একটি balanced sequence কিন্তু (()(কিন্তু balanced sequence না। এটা কীভাবে stack দিয়ে সমাধান করা যায় তা কি জানো? সহজ, তুমি (পেলে stack এ ঢুকাবে আর) পেলে দেখবে যে stack এর উপরে

(আছে কিনা। থাকলে তা তুলে ফেলবে, আর না থাকলে বুঝবে যে sequence টি balanced না। একটু চিন্তা করলে বুঝবে এখানে stack এর প্রয়োজন নেই, শুধু একটি counter রাখলেই হয়। অর্থাৎ stack এ কয়টি জিনিস আছে তা জানলেই হয়, কি কি আছে তা না জানলেও চলে, কারণ তুমি stack এ সবসময় (প্রবেশ করাও।

যাই হোক, stack এর মাধ্যমে balanced bracket sequence সমস্যা কীভাবে সমাধান করা হয় এটার সাপেক্ষে যদি আমাদের সমস্যা চিন্তা কর তাহলে আশা করি সমাধান পেয়ে যাবে। এটা পরিষ্কার যে আমাদেরকে stack ব্যবহার করতে হবে। কিন্তু stack এ কি রাখবা, কখন ঢুকাবা, কখন বের করবা ইত্যাদি হলো চিন্তা করার জিনিস। তোমরা চাইলে নিজেরা একটু চিন্তা করে দেখতে পারো নিচের সমাধান পড়ার আগে।

একটি সমাধান হতে পারে, তোমরা বাম হতে ডানে আসবে। যদি (পাও তাহলে তা ignore কর। যদি একটি matrix পাও তাহলে তার dimension কে stack এ প্রবেশ করাও। আর যদি) পাও তাহলে stack হতে উপরের দুটি matrix এর dimension তুল। ধরা যাক এরা হলো (a, b) [ধরলাম এটি উপরে ছিল] আর (c, d). তাহলে আমাদের দেখতে হবে $d = a$ কিনা। নাহলে আমাদের matrix দুটি গুন করা সম্ভব না। আর যদি তারা সমান হয় তাহলে এদের গুন করতে cost হবে $c \times a \times b$ আর নতুন matrix এর dimension হবে (c, b). এই matrix কে আমরা আবার stack এ পুশ করব। এভাবে সব গুন হয়ে গেলে সকল cost এর যোগফলই হবে আমাদের উত্তর।

UVa 11988 Broken Keyboard (aka Beiju Text)

সমস্যা: একটি লোক একটি string কীভাবে type করেছে তা দেওয়া আছে। String টিতে কিছু letter এবং underscore আছে। এর সাথে আছে [আর]. [এর মানে হল ঐ মুহূর্তে লোকটি home টিপেছে। এর ফলে তার typing এর cursor লাইনের শুরুতে এসে গিয়েছিল। আর] এর মানে হল end. এর ফলে তার cursor লাইনের শেষে এসে গিয়েছিল। তোমাকে বলতে হবে প্রদত্ত string এর মত করে টাইপ করলে আসলে string টি দেখতে কেমন হবে। যেমন ইনপুট abc[de]f হলে আউটপুট হবে deabcf.

সমাধান: খুব একটা কঠিন না। তুমি যদি একটু চিন্তা কর তাহলে stack দিয়ে সমাধান দাঁড় করাতে পারবে (যদিও খুব সহজ হবে না সেই সমাধান)। তবে সমাধানটা খুব সহজ হবে যদি তুমি stl এর list ব্যবহার কর। খেয়াল কর তোমার আসলে কোনো এক মুহূর্তে কি কি করতে হতে পারে। হয় তুমি string এর একদম শেষে প্রবেশ করাবে, অথবা একদম শুরুতে অথবা এর আগে যেখানে প্রবেশ করিয়েছিলে ঠিক তার পরে। এই কাজগুলি list দিয়ে $\mathrm{O}(1)$ সময়ে করতে পারবে। খেয়াল কর vector এর ক্ষেত্রে শুরুতে বা মাঝে insert করার cost কিন্তু constant না। কিন্তু list এর ক্ষেত্রে তুমি যদি মনে রাখো এর আগে কই ছিলে তাহলে এই তিনটি অপারেশন constant সময়ে করতে পারবে।

List এর insert ফাংশন আছে, তাকে তুমি যদি iterator দাও আর একটি মান দাও তাহলে তা ঐ iterator এর আগে সেই মানকে insert করবে। মনে কর আমরা list<char>::iterator it নামে একটি variable এ iterator রাখব যা নির্দেশ করবে তুমি কোথায় পরের character কে insert করবে, অর্থাৎ ঐ iterator যেখানে নির্দেশ করে তার ঠিক আগের জায়গায় নতুন character প্রবেশ করবে। শুরুতে it এর মান হবে L.end() এর সমান, যেখানে L হল আমাদের লিস্ট। আমরা যেসকল character প্রবেশ করাব সব লিস্ট এর শেষে প্রবেশ হবে। এখন যদি home টিপে তাহলে এই iterator এর মান পরিবর্তন হয়ে হয়ে যাবে L.begin(), এবং এর পরে যত character তুমি প্রবেশ করাবে সব "বর্তমান" লিস্ট এর শুরুর আগে প্রবেশ হবে। খেয়াল কর, যদিও আমাদের নতুন character লিস্ট এর শুরুতে প্রবেশ করাবে এবং এর ফলে L.begin() এর মান পরিবর্তন হবে, কিন্তু it এর মান কিন্তু পরিবর্তন হবে না। এটি কিন্তু আগের শুরুর মাথাকে পয়েন্ট করবে এবং তুমি যত নতুন character প্রবেশ করাবে সব ঠিক জায়গায় প্রবেশ হবে। একই ভাবে তুমি যদি end পাও তাহলে it এর মান L.end() করে দাও। এরকম করে করলেই এই সমস্যা সমাধান হয়ে যাবে।

UVa 12657 Boxes in a Line

সমস্যা: একটি স্থানে n টি বাক্স আছে, বামে হতে ডানে $1, 2 \ldots n$ এভাবে। n এর মান সর্বোচ্চ 100,000 হতে পারে। এখন তোমাকে চার ধরনের অপারেশন দেওয়া হবে। এক- X ও Y কে swap কর, দুই- X কে Y এর বামে রাখো, তিন- X কে Y এর ডানে রাখো, চার- পুরো বাক্সের sequence উলটিয়ে ফেলো। এরকম সর্বোচ্চ 100,000 টি অপারেশন দেওয়া থাকবে। তোমাকে সকল অপারেশন শেষে বিজোড়তম বাক্সগুলির (বাম হতে প্রথম, তৃতীয় এরকম) id এর যোগফল প্রিন্ট করতে হবে।

সমাধান: বোঝাই যাচ্ছে এখানে তোমাকে doubly linked list কোড করতে হবে। যা যা অপারেশন করতে বলা হয়েছে সেসবের জন্য লিংক লিস্টের লিংক গুলি কীভাবে ভাঙতে হবে বা গড়তে হবে সেটা চিন্তা করলেই হবে। আর অপারেশনের সময় X কোথায় আছে তা জানার জন্য একটি লিংক এর অ্যারে রাখতে পারি যা প্রতিটি বাক্স কোথায় আছে তা বলবে। খুব একটা কঠিন হবে না, তুমি হাতে হাতে লিংক লিস্টের কোড করার চেষ্টা করতে পারো। চাইলে stl এর list ব্যবহার করতে পারো তবে সেটি আরও ঝামেলাদায়ক হতে পারে এই সমস্যার ক্ষেত্রে।

UVa 679 Dropping Balls

সমস্যা: একটি tree দেওয়া আছে। Tree টি complete binary tree অর্থাৎ প্রতিটি নোডের দুটি করে child আছে। আর সকল leaf নোডগুলি D তম লেভেলে আছে। বাম হতে ডানে প্রথম লেভেলের নোডের নাম্বার হলো 1, দ্বিতীয় লেভেলের 2, 3. তৃতীয় লেভেলের 4, 5, 6, 7. ঠিক heap কে অ্যারেতে represent করলে নোডগুলির index যেরকম হয় সেরকম। D এর মান সর্বোচ্চ 20 হতে পারে। শুরুতে প্রতিটি নোডে একটি করে flag থাকে এবং এর মান false. এক এক করে মোট I টি বল ঢুকবে 1 দিয়ে। বলটি যেই নোডে থাকবে সেই নোডের flag এর মান flip হয়ে যাবে (true থাকলে false, false থাকলে true). সেই সাথে false হলে (flip এর আগে false হলে) সে বামে যাবে, আর যদি true হয় তাহলে সে যাবে ডানে। এভাবে ডানে বামে যেতে থাকবে যতক্ষণ না সে leaf এ পৌঁছে। তোমাকে বলতে হবে I তম বল কোন leaf এ যায়।

সমাধান: এটা বুঝাই যাচ্ছে যে i তম নোডের বামের child $2i$ আর ডানের child $2i + 1$. D depth এর ট্রি এর মোট নোড সংখ্যা $2^D - 1$. সুতরাং আমরা এই সাইজের একটি অ্যারে নিয়ে পুরো ব্যাপারটা simulate করতে পারি। এতে খুব একটা সময় লাগবে না। প্রতি বলের জন্য $\mathrm{O}(D)$ সময় লাগবে।

তবে এর থেকেও সহজ কোড করে এই সমস্যা সমাধান করা সম্ভব। তবে তার জন্য একটু চিন্তা করতে হবে। মনে কর তোমার কাছে 50 টি বল আছে। আর তুমি জানতে চাচ্ছ শেষটা কই যাবে? তুমি বলগুলিকে একে একে tree এর ভেতর দিয়ে না নিয়ে গিয়ে এদেরকে একত্রে নিয়ে যাবে আর জানতে চাইবে শেষটি কই যাবে। যদি 1 নাম্বার নোডের দিকে তাকাও তাহলে বুঝবে প্রথম বল বামে, দ্বিতীয় বল ডানে, তৃতীয় বল বামে এভাবে সকল বিজোড়তম বল বামে আর জোড়তম বল ডানে যায়। এর মানে 50 তম বল বা শেষ বল ডানে যাবে। শুধু সে না, মোট 25 টি বল ডানে যাবে। অর্থাৎ এখন আমাদের প্রশ্ন হলো আমরা 3 ($= 2 \times 1 + 1$) এ এসেছি। আমার কাছে 25 টি বল আছে এখন বলতে হবে শেষ বল কোথায় যাবে। একইভাবে বিজোড় বলগুলি বামে যাবে আর জোড় বলগুলি ডানে। তাই 25 তম বল বামে যায়। শুধু সে না মোট $\lceil 25/2 \rceil = 13$ টি বল বামে যাবে। সুতরাং আমাদের পরবর্তী প্রশ্ন হবে 13 টি বল নিয়ে আমরা 6 ($= 2 \times 3$) এ আছি, শেষ বল কই যাবে। এভাবে তুমি একটি while loop বা একটি recursive ফাংশনের সাহায্যে সমাধান করতে পারো।

UVa 122 Trees on the level

সমস্যা: একটি binary tree দেওয়া আছে (অর্থাৎ প্রতিটি নোডের সর্বোচ্চ দুটি child থাকবে)। তাদের লেভেল অনুসারে প্রিন্ট করতে হবে। অর্থাৎ প্রথমে সবচেয়ে উপরের level এর নোডগুলি বাম হতে ডানে, এরপর দ্বিতীয় লেভেলের নোডগুলি বাম হতে ডানে এভাবে প্রতিটি লেভেলের নোড বাম

হতে ডানে প্রিন্ট করতে হবে। তবে tree টির input একটু অন্যভাবে দেওয়া। মনে কর 4 হলো root তার বামে 2 আর 2 এর ডানে 9. তাহলে input এ 9 এর জন্য বলা থাকবে LR. অর্থাৎ root হতে কোনো নোডে Left-Right ভাবে কীভাবে যাওয়া লাগে সেটা দেয়া। এরকম করে প্রতিটি নোডের জন্য ইনপুট দেওয়া আছে, তোমাকে level অনুসারে বাম হতে ডানে নোডগুলি প্রিন্ট করতে হবে। যদি ইনপুট গুলি পূর্ণ tree নির্দেশ না করে (যেমন L বলে কিছু নেই কিন্তু LL আছে) তাহলে not complete কথাটি প্রিন্ট করতে হবে।

সমাধান: এই সমস্যা সমাধানের জন্য তোমাদের ইনপুট নেবার পর parsing করতে হবে। যা আমরা ইতোমধ্যেই আলোচনা করে এসেছি অন্য চ্যাপ্টারে। সুতরাং আমরা এখানে parsing নিয়ে আর আলোচনা করব না।

তোমরা চাইলেই LR এর স্ট্রিংগুলি দেখে দেখে একটি tree বানিয়ে ফেলতে পারো। প্রথমে প্রতিটি string দেখে tree তে সেই নোড বানাও, এসময় এমন হতে পারে যে কিছু নোড আগে থেকেই বানানো আছে। তোমাকে মূলত নতুন নোডগুলি বানাতে হবে। নোড বানানোর সময়, শেষ নোডে তুমি তার নাম লিখে রাখবে। যদি tree বানানো শেষে দেখো প্রতিটি নোডের নাম আছে তাহলে tree টি ঠিক আছে। নাহলে ইনপুট invalid. এখন তোমরা root হতে একটি bfs করলেই level অনুসারে আউটপুট পেয়ে যাবে। কারণ bfs এ সবচেয়ে কম দূরত্বের নোড আগে visit হয়। তবে যেহেতু আমরা বাম হতে ডানে প্রিন্ট করতে চাই সেজন্য কোনো নোডের child দের queue তে ঢুকানোর সময় তার left child কে right child এর আগে ঢুকাতে হবে। তাহলেই হয়ে যাবে।

কিন্তু এই সমস্যা আরও সহজে সমাধান করা যায়। খেয়াল কর উপরের লেভেলে থাকা মানে string এর দৈর্ঘ্য কম। সুতরাং আমরা যেহেতু লেভেল অনুসারে print করতে চাই সেহেতু আমাদের নোডগুলিকে তাদের string এর length অনুসারে sort করতে হবে। যদি length একই হয় তখন তাদেরকে lexicographically sort করলেই হবে। কারণ L তো R এর থেকে ছোট। আর আমরা চাই বামের নোডকে আগে প্রিন্ট করতে হবে। আর ইনপুট valid কিনা সেটা চেক করাও খুব একটা কঠিন না। মনে কর সকল LR string এর সেট হল S. তাহলে প্রতিটি ইনপুট স্ট্রিঙের জন্য দেখতে হবে যে তার শেষ character বাদ দিলে যেই স্ট্রিং হয় সেটি S এ আছে কিনা। এর মানে দাঁড়াবে প্রতিটি নোডের parent দেওয়া আছে।

UVa 548 Tree

সমস্যা: একটি binary tree এর root হতে কোনো একটি leaf নোড পর্যন্ত path এর cost হলো এই path এর প্রতিটি নোডের সংখ্যার (তার id এর) যোগফল। আমাদের সবচেয়ে কম cost টি প্রিন্ট করতে হবে। তবে tree টি ইনপুটে সরাসরি দেওয়া নেই। এর inorder এবং post order traversal দেওয়া আছে। Tree তে সর্বোচ্চ 10,000 টি নোড থাকতে পারে।

সমাধান: যদি আমাদেরকে সরাসরি tree টি দেওয়া থাকত তাহলে আমার মনে হয় বাকি সমস্যাটুকু খুব সহজেই সমাধান করে ফেলতা। এর জন্য একটি dfs লিখা লাগত। Dfs এ আমরা cost নানা ভাবে handle করতে পারব। মনে হয় সহজ উপায় হবে ফাংশনের parameter হিসাবে cost নেওয়া। অর্থাৎ dfs ফাংশনটি শুধু কোন নোডে আছি শুধু তাই নিবে না বরং সেই নোডে আসতে কত cost সেটাও নেবে। যদি কোনো নোডে এসে দেখি এটি leaf নোড, এর কোনো child নেই তাহলে আমরা global একটি variable কে প্রয়োজনে update করব। এই global variable টি হলো "সবচেয়ে কম cost এর leaf নোড যাবার path এর cost". আর আমরা যখন বর্তমান নোড হতে তার child এ যেতে চাইব তখন তার বর্তমান cost এর সাথে নতুন নোডের cost যোগ করে এই দুটি সংখ্যা দিয়ে dfs কে call করতে হবে।

তাহলে দাঁড়াল- যদি আমরা inorder আর post order traversal হতে আমাদের মূল tree বের করতে পারি তাহলে আমাদের সমস্যা সমাধান হয়ে যাবে।

এর আগে দেখে নেই inorder আর post order traversal কী। এদুটি ছাড়াও আরেকটি আছে তাহলো preorder traversal. আমরা এই তিনটি একই সাথে দেখি। মনে রাখার সুবিধার জন্য সবসময় এদের নাম খেয়াল করবা in, post, pre. এরা বলে root কই যায়। যেমন in এ root থাকে মাঝে, post এ root থাকে শেষে, আর pre তে root থাকে শুরুতে। এর মানে কি? যেমন

pre এর ক্ষেত্রে এর মানে হলো প্রথমে তুমি root নোড লিখবে এর পর left subtree এর preorder traversal লিখবে, এর পর right subtree এর। একই ভাবে, post এর ক্ষেত্রে সবশেষে root লিখবে, শুরুতে বামের, এর পর ডানের। inorder এ মাঝে থাকবে root, শুরুতে বামের আর শেষে ডানের। অর্থাৎ বাম ডানের order fixed. কথা হল root কোথায় বসবে। সেটা in, pre, post দেখে বুঝা যাবে।

আমাদের এই সমস্যায় inorder আর post order traversal দেওয়া আছে। এখান থেকে কি কি তথ্য পাওয়া যায়? প্রথমত Postorder traversal এর শেষের নোডটি হবে root. এখন inorder traversal এ root এর অবস্থান বের করলে, এর বামের অংশটুকু হবে left subtree আর ডানের অংশ right subtree. একটি inverse অ্যারে (অর্থাৎ কোনো একটি নোড inorder traversal এ কোথায় আছে) রাখলে inorder traversal এ root এর অবস্থান $\mathrm{O}(1)$ এ বের করা যাবে। ধরা যাক left subtree তে আছে L টি আর right subtree তে আছে R টি নোড। তাহলে post order traversal এর প্রথম L টি নোড হবে left subtree এর post order traversal, আর তার পরের R টি নোড হবে right subtree এর post order traversal. এর মানে তোমাকে যদি inorder আর post order traversal দেওয়া থাকে তুমি $\mathrm{O}(1)$ এ left subtree এর আর right subtree এর inorder এবং post order traversal বের করতে পারবে। তার মানে আমরা আবার মুল সমস্যায় ফিরে গিয়েছি- আমাদের ছোট একটি ট্রি এর inorder ও post order traversal দেওয়া আছে, ট্রিটি বের করতে হবে। এই ভাবে তুমি পুরো tree বানিয়ে ফেলতে পারবে।

মজার জিনিস হলো তুমি আসলে পুরো tree না বানিয়ে tree বানানোর পরে dfs এর মাধ্যমে যেই কাজ করতে সেটা এবং এইযে কিছুক্ষণ আগে inorder এবং post order নিয়ে কাজ করার যে ব্যাপারটি বললাম সেটা একত্র করে খুব ছোট কোড করে পুরো সমস্যা সমাধান করে ফেলতে পারবে।

LOJ 1101 A Secret Mission

সমস্যা: তোমাকে 50,000 নোড এবং 100,000 টি edge বিশিষ্ট একটি weighted গ্রাফ দেওয়া আছে। এরপর তোমাকে 50,000 টি query করা হবে। প্রতিটি query তে দুটি নোড দেওয়া থাকবে, ধরা যাক তারা হলো s এবং t. প্রতি query তে তোমাকে s এবং t এর মাঝের সবচেয়ে ভাল path বের করতে হবে। s এবং t এর মাঝে তো অনেক path থাকতে পারে। তাহলে কোনটি সবচেয়ে ভাল? এই ক্ষেত্রে সেই path টি সবচেয়ে ভাল যেই path এ থাকা edge সমূহের মাঝে সবচেয়ে বেশি cost টি সবচেয়ে কম হবে। মনে কর s আর t এর মাঝে দুটি path আছে। প্রথম path এ থাকা edge সমূহের মাঝে সবচেয়ে বেশি cost হলো a, আর দ্বিতীয় path এ থাকা edge সমূহের মাঝে সবচেয়ে বেশি cost হলো b. তাহলে তুমি a আর b এর মাঝে সবচেয়ে কম যেটা পাবে সেটা হবে তোমার উত্তর।

সমাধান: মনে কর মাত্র একটি query আছে। তাহলে কীভাবে সমাধান করতে? Dijkstra এর মত করে তাই না? Dijkstra তে আমরা cost রাখি s হতে কোনো নোডে যাবার cost. কিন্তু আমাদের রাখতে হবে কোনো নোডে আসার জন্য সবচেয়ে কম কত বড় cost এর edge ব্যবহার করতে হয়। অর্থাৎ dijkstra এর শুরুতে তুমি dist এর অ্যারেকে infinity দিয়ে initialize করবে এবং dist[s] কে 0 দিয়ে। এবার relax করার পালা। মনে কর তুমি u হতে v তে যাবার চেষ্টা করছ। u-v এর মাঝের edge এর cost w. তাহলে u এর মধ্য দিয়ে v তে যাবার জন্য cost হবে dist[u] আর w এর মাঝে যে বড় সে। এখন এই মান যদি dist[v] এর থেকে ছোট হয় তাহলে dist[v] কে আমরা update করব, ঠিক dijkstra এর মত। dijkstra তে আমরা লিখতাম $dist[v] = min(dist[v], dist[u] + w)$, কিন্তু এক্ষেত্রে আমাদের লিখতে হবে $dist[v] = min(dist[v], max(dist[u], w))$. এখন এই সমাধানে আমাদের সময় লাগবে $\mathrm{O}(E \log V)$ এর মত। অবশ্যই প্রতি case এ এটি আমাদের করা সম্ভব না। তাহলে কীভাবে আমরা এই সমস্যা সমাধান করতে পারি?

একটি case এর যেই সমাধান তার সাথে কি কোনো কিছুর মিল পাও? যদি মিল না পাও তাহলে চিন্তা করতে থাকো। প্রথমত আমরা চেষ্টা করব বেশি cost এর কোনো edge ব্যবহার না করতে। যত কম cost এর edge ব্যবহার করা সম্ভব। Sum কোনো ব্যাপার না। আমাদের কম cost এর edge দের ব্যবহার করতে হবে। এখন বুঝতে পারছ এর সাথে কিসের মিল আছে? Minimum spanning

tree এর। যদি একটু ভাল করে চিন্তা কর তাহলে বুঝবে s আর t এর মাঝের যেকোনো optimal path হবে এই minimum spanning tree এর ভেতর দিয়ে path. কেন? এর প্রমাণ খুব একটা কঠিন না। মনে কর আমরা s হতে t তে এমন একটি optimal path পাই যার cost হবে MST এর ভেতর দিয়ে পাথের থেকেও কম cost এর। এর মানে optimal path এর জন্য সবচেয়ে বেশি cost এর যেই edge পাওয়া তার cost হবে MST এর ভেতর দিয়ে path এর উপর থাকা সবচেয়ে বড় cost এর edge এর থেকে কম cost এর। এখন তুমি যদি একটু চিন্তা কর তাহলে দেখবে MST এর ভেতরের পাথে সবচেয়ে বড় cost এর যেই edge ছিল তাকে বাহিরের পাথের কোনো একটি edge দিয়ে replace করে MST এর cost আরও কমানো যাবে। কিন্তু MST এর cost তো কখনও কমানো সম্ভব না তাই না? সুতরাং এরকম হবে না, অর্থাৎ s হতে t এর পাথ সবসময় MST এর ভেতর দিয়ে হবে।

সুতরাং আমরা প্রদত্ত গ্রাফের MST বের করে ফেলব। এর পর প্রতি query তে s আর t এর মাঝে এই MST এর ভেতর দিয়ে path বের করব আর এই path এর সবচেয়ে বেশি cost এর edge বের করব। তাহলেই হয়ে যাবে। যেহেতু একটি tree তে $V-1$ টি edge থাকে তাই আমাদের প্রতি query তে সময় লাগবে $\mathrm{O}(V)$.

কিন্তু এই complexity ও যথেষ্ট না। এই ক্ষেত্রে বুদ্ধি হলো LCA (Least Common Ancestor). যখনই একটি tree তে দুটি নোডের মাঝে কিছু একটা করতে বলবে তখন LCA প্রায় সময়ই কাজে লাগে। s হতে root এ যাও আর t হতে root এ যাও। এই দুই পাথে তো অনেকগুলি নোড থাকতে পারে। আবার এই দুই পাথেই আছে এরকম বহু নোডও থাকতে পারে। এই উভয় পথেই থাকা নোড-গুলির মাঝে যেটি root হতে সবচেয়ে দূরে থাকে সেটিই s ও t এর LCA. LCA কীভাবে বের করতে হবে তা না জানলে পড়ে নাও। মূল আইডিয়া হলো ট্রি এর প্রতিটি নোডের $2^0, 2^1, 2^2, 2^3 \ldots 2^{\log V}$ তম parent গুলি বের করে রাখতে হয়। এখন আমরা যদি শুধু ওত তম parent না বরং ওত তম parent এ যেতে যেই path তার মাঝের সবচেয়ে বড় edge এর cost ও লিখে রাখি তাহলে কিন্তু আমরা s হতে c এবং t হতে c path এর সবচেয়ে বড় cost এর edge ও বের করে ফেলতে পারব, যেখানে c হল LCA. তাহলেই আমাদের সমাধান lca বের করার মত $\log V$ তে হয়ে যাবে।

LOJ 1128 Greatest Parent

সমস্যা: একটি 100,000 নোডের rooted tree আছে। প্রতিটি নোডের একটি weight আছে। Tree এর weight গুলি এমন হবে যেন যেকোনো নোডের weight তার parent এর weight অপেক্ষা বেশি হয়। এখন তোমাকে প্রায় 50,000 query করা হবে। প্রতি query তে তোমাকে v নোড এবং w weight দেওয়া থাকবে। তোমাকে বলতে হবে v নোডের সবচেয়ে উপরের (অর্থাৎ root এর কাছাকাছি) কোন ancestor এর weight, w এর সমান বা এর থেকে বেশি।

সমাধান: তুমি যদি LOJ 1101 করে থাকো তাহলে মনে হয় না এটা খুব একটা কঠিন লাগবে। প্রথম কথা, তুমি যদি একে tree না মনে করে একটি লাইন কল্পনা কর তাহলে তো এটা binary search ছাড়া আর কিছুই না। তাহলে কি আমরা tree এর প্রতিটি নোডের জন্য তার সকল ancestor দের নিয়ে একটি লিস্ট বানিয়ে রাখবো আর query আসলে সেই list এ binary search করব? না তা হবে না। কারণ একটি লাইন গ্রাফের কথাই চিন্তা কর (লাইন গ্রাফ মানে সেই গ্রাফে কোনো branching নেই)। এই গ্রাফে i তম নোডের list এ i টি ancestor কে রাখতে হবে। সুতরাং তোমার V^2 জায়গা লেগে যাবে। তাহলে কি উপায়? উপায় হলো LCA এর মত করে প্রতিটি নোডের জন্য $2^0, 2^1, 2^2, 2^3 \ldots$ তম ancestor রাখা। এরপর তুমি v থেকে শুরু করবে। দেখবে v এর 2^k তম ancestor এর weight w থেকে ছোট কিনা, ছোট হলে 2^{k-1} তম ancestor এর সাথে একই রকম চেক করবে। আর যদি বড় বা সমান হয় তাহলে v কে সেই 2^k তম parent বানাবে এবং একই ভাবে তার এবার 2^{k-1} তম parent চেক করবে। খেয়াল কর প্রতি চেক শেষে কিন্তু আমাদের k এর মান এক করে কমছে। সুতরাং আমরা $k = \log V$ থেকে শুরু করলে $\mathrm{O}(\log V)$ এর বেশি সময় লাগবে না। আর $\log V$ এর বেশি k নিয়ে শুরু করার দরকার নেই তাই না (না বুঝলে LCA পড়ে দেখো)?

LOJ 1080 Binary Simulation

সমস্যা: একটি 100,000 সাইজের 0-1 এর অ্যারে আছে। দুই ধরনের অপারেশন থাকতে পারে। এক- i হতে j পর্যন্ত সকল element কে flip কর অর্থাৎ 0 থাকলে 1, আর 1 থাকলে 0. দুই- i তম নাম্বার কত তা প্রিন্ট করতে হবে। মোট operation এর সংখ্যা 50,000.

সমাধান: নানা ভাবে এই সমস্যা সমাধান করা যায়। এখানে আমি তিন রকম সমাধানের কথা বলব। প্রতিটি সমাধানেই আমাদের প্রতি অপারেশনের জন্য সময় লাগবে $\mathrm{O}(\log n)$.

প্রথম উপায় হলো lazy propagation. যারা জানো না তারা আগে lazy propagation পড়ে নাও। এখন এই সমস্যায় যখন বলবে কোনো রেঞ্জে flip করার কথা তখন সেই রেঞ্জ update করতে গিয়ে যেই যেই বড় বড় রেঞ্জ পাবে যাদের তোমার update এর রেঞ্জ পুরোপুরি cover করে সেখানে lazy রেখে আসবে। এরপর query এর সময় নিচে নামার সময় যদি lazy রেখে আসা কোনো রেঞ্জের নিচে নামতে চাই তখন আমাদের সেই lazy propagate করতে হবে। এই সমাধানে segment tree এর প্রতিটি নোডে যেই যেই information রাখতে হবে সেগুলি হলো- এই রেঞ্জে কতগুলি lazy জমে আছে। তুমি চাইলে এখানে যতগুলি lazy জমে আছে তার parity রাখতে পারো (parity মানে সেই সংখ্যা জোড় নাকি বিজোড় তা)। কারণ 3 টা lazy জমা হওয়া আর 1 টা জমা হওয়া কিন্তু একই কথা।

এখন দ্বিতীয় সমাধান দেখা যাক। প্রথম সমাধান নিয়ে একটু চিন্তা কর। আদৌ কি lazy কে propagate করার দরকার আছে? আমরা lazy রেখে চলে যাব। এরপর যখন বলবে যে i তম নোডে কি আছে 0 না 1, তখন আমরা segment tree এর উপর থেকে i এর leaf পর্যন্ত যাব। আর রাস্তায় দেখবো যে এই i তম স্থানটি কতবার flip হয়েছে। এই flip এর সংখ্যা আর কিছুই না যতগুলি নোড দিয়ে আমাকে নামতে হবে তাদের lazy এর যোগফল। তাহলে আমরা এই সমাধানে lazy কে propagate না করেই সমাধান করে ফেলছি।

এখন তৃতীয় সমাধান। একটা জিনিস খেয়াল কর, আমরা যদি flip না করতে বলে বলতাম যে i হতে j পর্যন্ত 1 যোগ কর আর query এর সময় যদি জিজ্ঞাসা করতাম যে অমুক স্থানে সংখ্যার parity কত (বা অমুক স্থানের সংখ্যা কত) তাহলে এটি আমাদের সমস্যার equivalent. অর্থাৎ আমাদের মূল সমস্যা সমাধান করা আর পরিবর্তিত এই সমস্যা সমাধান করা একই কথা। এই কাজ তো segment tree এর সাহায্যে করাই যাবে। কিন্তু binary indexed tree দিয়ে কীভাবে সহজ ভাবে করবে? যদি BIT সম্পর্কে জেনে থাকো তাহলে বুঝতেই পারছ মূল সমস্যা কোথায়। BIT এ আমরা একটি segment কে update করতে পারি না। আমাদের সবসময় একটি index কে update করতে হয়। আর query করে পাওয়া যায় 1 হতে i পর্যন্ত সংখ্যার যোগফল। এই সমস্যায় আমাদের দরকার i তম সংখ্যা আর update করা দরকার i হতে j. সুতরাং সরাসরি সমাধান হবে না। একটু চিন্তা ভাবনা করতে হবে।

ধরা যাক আমাদের অ্যারেটি হলো $x_1, x_2, x_3 \ldots$. আমরা এই অ্যারেকে transform করে লিখব $x_1, x_2 - x_1, x_3 - x_2, \ldots$. তাহলে লাভ কি? প্রথম লাভ, 1 হতে i তম সংখ্যা পর্যন্ত যোগ করলে তুমি পাবে $x_1 + (x_2 - x_1) + (x_3 - x_2) + \ldots (x_i - x_{i-1}) = x_i$. বাহ আমরা x_i পেয়ে গেলাম প্রথম থেকে i তম সংখ্যা পর্যন্ত যোগ করে। এখন মূল সমস্যা, i হতে j পর্যন্ত 1 যোগ করতে হবে, অর্থাৎ আমাদেরকে $x_i, x_{i+1} \ldots x_j$ পর্যন্ত সংখ্যাগুলির সাথে 1 যোগ করতে হবে। কিন্তু আমরা তো আর $x_1, x_2, x_3 \ldots$ রাখছি না BIT এ, আমরা রাখছি $x_1, x_2 - x_1, x_3 - x_2, \ldots$. তাহলে এই পরিবর্তিত sequence এ কী কী পরিবর্তন করতে হবে একটু চিন্তা করে দেখো তো! প্রথমত i তম সংখ্যায় 1 যোগ করতে হবে কারণ i তম সংখ্যা হচ্ছে $x_i - x_{i-1}$, আর আমরা x_i কে 1 বাড়াচ্ছি। আর আমাদেরকে $j + 1$ তম সংখ্যা থেকে 1 বিয়োগ করতে হবে কারণ সেটি হলো $x_{j+1} - x_j$ (i বা j যদি মাথায় হয় তাহলে একটু সাবধান হতে হবে হয়তো)। আর কিন্তু কোনো সংখ্যায় কোনো পরিবর্তন করতে হবে না। কেন? কাগজ কলম নিয়ে চিন্তা করে দেখো তাহলেই বুঝবে। সুতরাং আমরা পরিবর্তিত sequence এ দুইটি update করব আর query করব শুরু হতে কোনো index পর্যন্ত সকল সংখ্যার যোগফল। অর্থাৎ আমরা BIT দিয়ে আমাদের সমস্যা সমাধান করে ফেললাম।

LOJ 1087 Diablo

সমস্যা: তোমাকে n টি সংখ্যা দেওয়া আছে (সর্বোচ্চ 100,000 টি)। সেই সাথে সর্বোচ্চ 50,000 টি (q) অপারেশন। দুই ধরনের অপারেশন হতে পারে। এক- আমাদের sequence এর শেষে একটি নতুন সংখ্যা প্রবেশ করাতে হবে। দুই- প্রথম থেকে k তম সংখ্যা কত? এই প্রশ্নের উত্তর দেবার পর সেই সংখ্যাকে ফেলে দাও।

সমাধান: অবশ্যই binary search tree দিয়ে সমাধান করার চেষ্টা করব না। সেটা হবে মশা মারতে কামান দাগার মত। সহজে কীভাবে সমাধান করা যায়? একটি উপায় হলো segment tree ব্যবহার করা। একটি n+q সাইজের অ্যারে নিয়ে তার segment tree বানিয়ে ফেলো। এই সময় প্রথম n টি স্থানে 1 আর শেষের q টি স্থানে 0 রাখো। যখনই k তম সংখ্যা চাইবে তুমি segment tree হতে k তম 1 কোথায় তা বের কর। এজন্য একটি উপায় হতে পারে পজিশনের উপর binary search করা যে "আমি এখন guess করছি i তম স্থানে k তম 1 আছে"। এবার query করে দেখ যে আসলেই তা সত্যি কিনা। অর্থাৎ তোমাকে দেখতে হবে 1 হতে i পর্যন্ত 1 এর যোগফল কত আর i এ 1 আছে কিনা। তুমি যদি দেখ যে এই যোগফল k এর থেকে বেশি বা i এ 1 নেই তার মানে তুমি বুঝে যাবে যে k তম 1 আছে i এর বামে। এরকম করে binary search করতে থাকলে তুমি k তম 1 পেয়ে যাবে। কিন্তু এর জন্য প্রতি query এর complexity হবে $\mathrm{O}(\log^2 n)$.

এই কাজ তুমি খুব সহজেই $\mathrm{O}(\log n)$ এ করতে পারবা। তুমি tree এর কোনো নোডে এসে দেখবে এর left tree তে কতগুলি 1 আছে আর right tree তে কতগুলি। এর উপর ভিত্তি করে তুমি বুঝে যাবে কোন দিকে k তম 1 আছে। এভাবে k তম 1 কোন index এ আছে তা বের করে ফেলতে পারবে। এটাতো গেল প্রথম অপারেশন। অপর অপারেশনের (নতুন সংখ্যা প্রবেশ করানো) সময় তুমি n এর পরের ঘর কে 0 হতে 1 করবে, আবার দ্বিতীয় বার যখন নতুন সংখ্যা প্রবেশ করতে বলবে তখন তার পরের 0 কে 1 করবে এরকম করে চলতে থাকবে। অর্থাৎ আমাদের segment tree তে কোনো একটি স্থানে 1 যোগ করতে হবে। আর কোন id তে কোন সংখ্যা আছে তা তো আমরা একটা আলাদা অ্যারেতে লিখে রাখতেই পারি। তুমি যখন segment tree হতে জেনে যাবে কোন index এ k তম সংখ্যা আছে তখন সেই id তে রাখা সংখ্যা দেখে নিলেই হলো।

LOJ 1089 Points in Segments (II)

সমস্যা: তোমাকে কিছু রেঞ্জ দেওয়া আছে (সর্বোচ্চ 50,000 টি)। সেই সাথে কিছু সংখ্যা দেওয়া আছে (সর্বোচ্চ 50,000 টি)। তোমাকে প্রতিটি সংখ্যার জন্য বলতে হবে সেটি কতগুলি রেঞ্জের ভেতরে আছে। রেঞ্জের দুই মাথা বা প্রদত্ত সংখ্যাগুলি 10^8 এর মত হতে পারে।

সমাধান: খুব একটা কঠিন সমস্যা না। নানা ভাবে এই সমস্যার সমাধান করা যায়।

একটি উপায় হলো segment tree ব্যবহার করা। মনে কর প্রদত্ত প্রতিটি $[i, j]$ সেগমেন্টের জন্য segment tree তে সেই রেঞ্জের পুরোটুকুতে 1 যোগ করবে। এর পর প্রতি query তে দেখতে হবে query এর স্থানে কত আছে। কিন্তু এই সমাধানে সমস্যা হলো আমাদের সংখ্যাগুলি 10^8 পর্যন্ত হতে পারে। সুতরাং আমাদের 10^8 সাইজের অ্যারের segment tree করা লাগবে। যেটা একটু বেশি memory খেয়ে ফেলবে। এখানে একটা জিনিস লক্ষ্য করার আছে, সেটা হলো রেঞ্জের দুই মাথা বা query এর নাম্বার গুলি কত সেটা কোনো ব্যাপার না, তাদের relative order ই ব্যাপার। অর্থাৎ তোমাকে যদি একটি রেঞ্জ $[1, 10]$ দেয় আর query যদি হয় 5 তাহলে এর সমাধান করা যা কথা, রেঞ্জ $[1, 3]$ নিয়ে 2 সংখ্যা নিয়ে তার সমাধান করা একই কথা। সুতরাং আমরা সকল সংখ্যা নিয়ে sort করে তাদেরকে $\mathrm{O}(n)$ রেঞ্জের নাম্বারে assign করে দিতে পারি অর্থাৎ sort করার পর (এবং duplicate[১] মুছে ফেলার পর) প্রথম সংখ্যা 1, এরপরেরটা 2, তার পরেরটা 3 এরকম। এখানে সব সংখ্যা মানে হল query এর সংখ্যা আর range এর দুই প্রান্তের সংখ্যা। তোমরা চাইলে একটু চিন্তা

[১]Duplicate মুছে ফেলার একটি সহজ উপায় হল sort(A.begin(), A.end()) আর এর পর A.erase(unique(A.begin(), A.end()), A.end()).

করতে পারো এখানে segment tree ব্যবহার না করে LOJ 1080 এর মত করে BIT ব্যবহার করে সমাধান করা যায় কিনা।

চাইলে segment tree বা BIT ছাড়াই এই সমস্যা সমাধান করা যায়। একটি বিন্দু নিয়ে চিন্তা করে দেখো, যদি তুমি কোনো ভাবে এর বামে কয়টি রেঞ্জের শুরুর মাথা আছে, এর ডানে কয়টি আছে, বামে রেঞ্জের শেষের মাথা কয়টি আছে আর ডানেই বা কয়টি আছে এই চারটি সংখ্যা জানো তাহলে কিন্তু তুমি বলে ফেলতে পারবে এই বিন্দুটি কোনো রেঞ্জের ভেতরে আছে কিনা বা থাকলে কয়টি রেঞ্জের ভেতরে আছে। আসলে চারটি সংখ্যার দরকার নেই, দুটি হলেই চলে। এই বিন্দুর আগে বা ঐ বিন্দুর স্থানে কতগুলি রেঞ্জের শুরুর মাথা আছে (A) আর ঐ বিন্দুর আগে কয়টি রেঞ্জের শেষ মাথা আছে (B). তাহলে $A - B$ হবে এই বিন্দু কতগুলি রেঞ্জের ভেতরে আছে তা। এটা খুব একটা কঠিন না। তুমি শুরুর মাথা নিয়ে একটা sorted array রাখো আর তাতে প্রতিটি বিন্দুর জন্য binary search কর তাহলেই হয়ে যাবে। একই ভাবে তুমি শেষের মাথা sorted array তে নিয়ে binary search করে B বের করতে পারো। তাহলেই প্রতিটি বিন্দুর জন্য সমাধান বের হয়ে যাবে।

চাইলে line sweep এর মত করেও সমাধান করতে পারো। মনে কর প্রতিটি query এর বিন্দু, রেঞ্জের শুরুর মাথা শেষের মাথা এসবকে একেকটি event হিসাবে নিলে। এর পর সব event কে sort করে ফেলো আর বাম দিক থেকে line sweep কর। যদি একটি রেঞ্জের শুরুর মাথা পাও তাহলে একটি counter কে বাড়াও। যদি রেঞ্জের শেষের মাথা পাও তাহলে সেই counter কে কমাও। আর যদি কোনো query এর বিন্দু পাও তাহলে তার জন্য উত্তর হবে বর্তমান counter এর মান। সহজ না? তবে যদি একই বিন্দুতে যদি query বিন্দু আর range এর শুরু বা শেষের মাথা থাকে তাহলে একটু সমস্যা। কারণ শুরুর মাথা যদি আগে প্রসেস কর তাহলে counter বাড়াতে হবে। আবার query বিন্দু প্রসেস করলে counter এর মান দেখতে হবে। সুতরাং একই বিন্দুতে ইভেন্টগুলি কি অর্ডারে প্রসেস করছ তা অনেক গুরুত্বপূর্ণ। তুমি নিজে একটু চিন্তা করে দেখতে পার কি অর্ডারে প্রসেস করা উচিত। উত্তর হল- প্রথমে শুরুর মাথা, এরপর query বিন্দুগুলি এবং সবশেষে শেষের মাথা। এটা করা খুবই সহজ, সর্ট এর comparison function এ কে আগে আসবে কে পরে আসবে সেখানে এসব লজিক লিখে দিলেই হবে। আরও সহজ হবে তুমি যদি শুরুর মাথাকে -1 দিয়ে, query বিন্দুকে 0 দিয়ে এবং শেষের মাথাকে 1 দিয়ে প্রকাশ কর, আর এই মানের ভিত্তিতে যদি তুমি সর্ট কর (যখন বিন্দুর মান একই হয় তখন)।

LOJ 1093 Ghajini

সমস্যা: তোমাকে 100,000 টি সংখ্যার একটি অ্যারে a দেওয়া আছে। সেই সাথে তোমাকে আরও একটি সংখ্যা d দেওয়া আছে। তোমাকে v এর সবচেয়ে বড় মান বের করতে হবে যেখানে $v = a[i] - a[j]$ এবং $abs(i-j)+1 \leq d$. অর্থাৎ d এর বেশি দূরে (index এর সাপেক্ষে) নেই এরকম দুটি সংখ্যার সর্বোচ্চ difference বের করতে হবে।

সমাধান: আমরা যদি সকল d দৈর্ঘ্যের subarray এর জন্য minimum এবং maximum বের করতে পারি তাহলে আমাদের সমস্যা খুব সহজেই সমাধান হয়ে যাবে। তুমি চাইলে এজন্য segment tree ব্যবহার করতে পারো। বা চাইলে "static ডেটায় rmq" এর টেকনিক খাটাতে পারো। অথবা চাইলে prioirty queue ব্যবহার করতে পারো। আমরা এখানে priority queue এর সমাধান বলব। আমরা যদি যেকোনো d দৈর্ঘ্যের subarray এর maximum বের করতে পারি তাহলে minimum ও বের করে ফেলতে পারব। তাহলে কীভাবে maximum বের করব? আমরা বাম হতে ডান দিকে যাব এবং একে একে সংখ্যা এবং তার index কে একটি structure এর ভেতর ভরে priority queue তে পুশ করব। সেই সাথে প্রতিটি index এ এসে দেখবো সেই মুহূর্তে priority queue এর top এ কে আছে। যদি তার index আমাদের বর্তমান স্থানের থেকে d এর বেশি দূরত্বে থাকে তাহলে তাকে ধরে ফেলে দেব (pop)। এরকম করে ধরে ফেলে দিতে থাকব যতক্ষণ priority queue এর top এর index বর্তমান স্থান থেকে d এর বেশি দূরত্বে থাকবে। যখন d এর বেশি দূরে নয় এরকম index পাব তখন সেটিই হবে আমাদের maximum (এখান থেকে বামের দিকে d দূরত্ব পর্যন্ত)। এভাবে

আমরা প্রতিটি স্থানে গিয়ে এর আগের d ঘরের মাঝে সবচেয়ে বড় এবং সবচেয়ে ছোট সংখ্যা বের করে ফেলতে পারব। তাহলেই তো আমাদের সমস্যা সমাধান হয়ে যায় তাই না?

LOJ 1267 Points in Rectangle (II)

সমস্যা: তোমাকে 2d গ্রিডে 50,000 টি বিন্দু দেওয়া থাকবে। সেই সাথে তোমাকে 50,000 টি আয়তক্ষেত্র query হিসাবে দেওয়া হবে। তোমাকে প্রতিটি আয়তক্ষেত্রের জন্য বলতে হবে প্রদত্ত বিন্দুগুলির মাঝে কতগুলি বিন্দু তার ভেতরে আছে। এখানে বিন্দু এবং আয়তক্ষেত্রের coordinate 0 হতে 10^9 পর্যন্ত হতে পারে।

সমাধান: এই সমস্যা সমাধানের আগে চাইলে LOJ 1266 সমাধান করে নিতে পারো (এটি exercise এ আছে)। এর সাথে 1266 এর পার্থক্য কোথায়? মূল পার্থক্য দুটি জায়গায়। প্রথমত এখানে বিন্দুগুলি আগেই দেওয়া আছে। আর এখানে coordinate অনেক বড় হতে পারে। আমরা "বিন্দুগুলি আগে আগে দেয়ার" সুবিধা নিয়ে সমাধান করব।

আমরা y axis বরাবর একটি segment tree রাখবো। এখন আমরা বাম হতে ডানে line sweep করব। যখন কোনো একটি বিন্দু (x, y) পাবো তখন segment tree এর y এ আমরা 1 যোগ করব। আর যখন একটি আয়তক্ষেত্রের বাম প্রান্ত পাবো তখন আমরা segment tree তে query করব যে $[y_1, y_2]$ এই range এর যোগফল কত। ধরা যাক এটি A. একই ভাবে আমরা যখন ডান প্রান্ত পাবো তখন আবার query করব, ধরা যাক এটি হবে B. তাহলে এই আয়তক্ষেত্রের ভেতরে $B - A$ টি বিন্দু থাকবে। একটু সাবধান হতে হবে, খেয়াল করতে হবে বিন্দু কি আয়তক্ষেত্রের border এ থাকলেও আমাদের count করতে হবে কিনা। সেই অনুসারে আমাদের A আর B গণনা করতে হবে। তবে সেটি বেশি details. আমরা যদি মূল আইডিয়া বুঝি তাহলে এটুকু এদিক ওদিক কোনো ব্যাপার না। চিন্তা করে দেখতে পারো এই সমস্যায় BIT ব্যবহার করা যায় কিনা।

LOJ 1188 Fast Queries

সমস্যা: একটি 100,000 সংখ্যার অ্যারে দেওয়া আছে। তোমাকে 50,000 query করা হবে। প্রতি query তে তোমাকে i এবং j দেওয়া হবে। বলতে হবে i হতে j এর মাঝে কতগুলি ভিন্ন ভিন্ন সংখ্যা আছে। অ্যারের সংখ্যা গুলি 0 হতে 10^5 পর্যন্ত হতে পারে।

সমাধান: সমস্যাটা শুরুতে একটু কঠিন লাগবে। কিন্তু তোমরা যদি একটা trick শিখে যাও তাহলে এধরনের সমস্যা সহজ হয়ে যাবে।

প্রথমত খেয়াল কর, কোন কোন range এ query হবে তা আগেই বলা আছে। অর্থাৎ আমরা offline এ তাদের প্রসেস করতে পারি চাইলে। সুতরাং আমরা যা করব তাহলো অ্যারের বাম হতে ডানে যাব কিছুটা line sweep এর মত। আমরা যখনই কোনো একটি query রেঞ্জের ডান প্রান্তে আসব তখন আমরা সেই রেঞ্জে query করব। কিন্তু কি query করব? কীভাবে আমরা বের করব কতগুলি unique সংখ্যা আছে? খুব একটা কঠিন না। মনে কর আমরা line sweep করতে করতে এখন x তম index এ এসে গেছি। এখানের সংখ্যাটি মনে কর $A[x]$. আমাদের যা করতে হবে তাহলো দেখতে হবে এর আগে কি $A[x]$ কোথাও ছিল কিনা। এটি একটি map রেখে খুব সহজেই করতে পারো। বা যেহেতু সংখ্যাগুলি ছোট সেহেতু একটি অ্যারে রেখেও করতে পারো। যদি $A[x]$ এর আগে থেকে থাকে তাহলে সেই জায়গা শূন্য করে দেব। আর x এর জায়গা 1 করে দেব। তাহলে কি হচ্ছে খেয়াল কর- এখন যদি তুমি x এ থেকে থাকো তাহলে x এর বামে সকল unique সংখ্যায় এখন 1 আছে আর বাকি গুলিতে 0. শুধু তাই না, মনে কর x এর বামে একাধিক 10 আছে। তাহলে শেষ 10 এ 1 থাকবে, বাকি 10 গুলিতে 0. এর ফলে যেকোনো মুহূর্তে query করলে সঠিক উত্তর আসবে। একটা উদাহরণ দেখো- মনে কর আমাদের সংখ্যাগুলি হলো 10 1 10. এখন মনে কর তুমি শেষে আছ। এখানে ডান প্রান্ত আছে এরকম রেঞ্জ সমূহতে তুমি query করবে। তাহলে তুমিই বল কোন 10 এ 1 রাখবে? নিশ্চয় ডানেরটায়? তা নাহলে তোমার query যদি তৃতীয় index হতে তৃতীয় index

এই হয় তাহলে আর উত্তর 1 পাবে না। আবার দুটি 10 এই 1 থাকলে প্রথম index হতে তৃতীয় index এ query করলে দুটি 10 ই count হয়ে যাবে। সুতরাং কোনো একটি মুহূর্তে আমাদের একটি 10 এই 1 থাকবে, বাকি গুলিতে 0. শুধু তাই না এই 1 ওয়ালা 10 হবে সবচেয়ে ডানেরটা (আবার এতো ডানে যেন না হয় যে সেটা আবার x কে অতিক্রম করে)। সুতরাং আমরা segment tree বা BIT ব্যবহার করে এই সমস্যা সমাধান করে ফেলতে পারব।

যদি এই সমস্যাতে query সমূহ online হত তাহলে এই সমস্যা প্রতি query $\mathrm{O}(\log^2 n)$ time complexity তে সমাধান করা যেত। এজন্য অবশ্য তোমাদের merge sort tree ধরনের কিছু ব্যবহার করতে হত। তবে মূল আইডিয়া আমার বর্ণিত সমাধানেই বলা আছে। তোমাদের আগ্রহ থাকলে এই সমস্যাকে online মনে করে প্র্যাক্টিস করতে পার। আশা করি পরে কোনো সময় আমরা merge sort tree নিয়ে আলোচনা করব।

LOJ 1348 Aladdin and the Return Journey

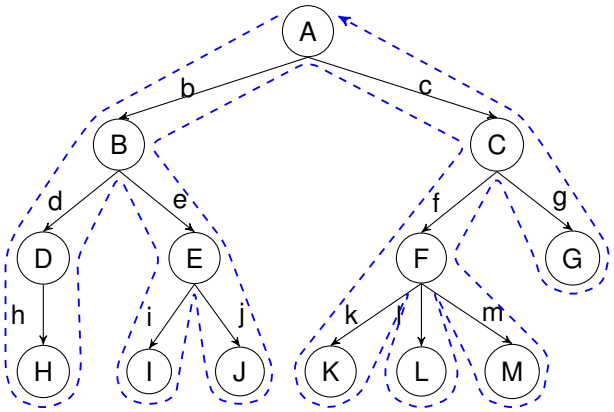
সমস্যা: একটি 30,000 নোডের tree আছে। প্রতিটি নোডের একটি weight আছে। তোমাকে দুই ধরনের operation দেওয়া হবে। এক- i এবং j দেওয়া হবে, এদের মাঝের যেই path তাতে থাকা নোডগুলির weight এর যোগফল প্রিন্ট করতে হবে। দুই- i এবং v দেওয়া থাকবে তোমাকে i নোডের weight কে v তে পরিবর্তন করতে হবে। মোট operation এর সংখ্যা 100,000. Weight সর্বোচ্চ 1000 হতে পারে।

সমাধান: যদি LOJ 1128 বা LOJ 1101 এর মত করে সমাধান করতে চাও তাহলে হবে না। কারণ এখানে weight পরিবর্তন হচ্ছে। LCA এর মত করে max, min বা sum তখনই বের করা যাবে যখন weight পরিবর্তন হবে না। তোমরা চাইলে Heavy Light Decomposition এর মত কঠিন টেকনিক ব্যবহার করেও এই সমস্যা সমাধান করতে পারো। তবে এর বেশ সুন্দর একটি সমাধান আছে। সমাধানটি এক কথায় বলা যায়- "Euler path এর উপর Segment Tree ব্যবহার করে সমাধান"।

একটি tree এর euler path চিত্র ৬.১ এ দেখানো হলো। Tree এর জন্য euler path এমন কোনো কঠিন জিনিস না। একটা dfs tree আর কি- Child এর dfs কে call দেবার সময় নিচে নামা আর ঐ নোডের পুরো subtree প্রসেস শেষে উপরে উঠে আসা। আমরা এই path এর edge গুলিকে যদি পর পর লিখি তাহলে দাঁড়াবে b d h h d e i i j j e b c f k k l l m m f g g c. বুঝতেই পারছ এখানে বড় হাতের B নোডের সাথে parent এর যেই edge সেটি হলো ছোট হাতের b আর এভাবেই অন্য সব edge এর নাম দেওয়া হয়েছে। এখন একটা কাজ কর, নিচে নামার সময় positive আর উপরে উঠার সময় negative দাও, অর্থাৎ +b +d +h -h -d +e +i -i +j -j -e -b +c +f +k -k +l -l +m -m -f +g -g -c. মনে কর b হলো B নোডের weight. তাহলে শুরু থেকে যদি তুমি +j পর্যন্ত সব সংখ্যা যোগ কর তাহলে পাবা b+e+j বা root হতে J পর্যন্ত সকল নোডের cost এর যোগফল। এখানে root এর cost a বাদ গিয়েছে। কিন্তু আমার মনে হয় না সেটা কোনো ব্যাপার। তোমরা চাইলে a সবসময় যোগ করে নিতে পারো। বা চাইলে শুরুতে +a রাখতে পারো অথবা, কল্পনা করতে পারো যে root এর parent হতে root এর যেই edge তার cost a. অনেক ভাবেই এই ব্যাপারটা handle করা যায়। তাই সেটা নিয়ে মাথা ব্যাথার কিছু নেই। সুতরাং এই পদ্ধতিতে আমরা root হতে যেকোনো node এর path এর সকল নোডের cost এর যোগফল পেয়ে যাব। কিন্তু আমাদের দরকার s হতে t পর্যন্ত path এর নোডগুলির cost এর যোগফল। এই ক্ষেত্রে একটা বুদ্ধি আছে। মনে কর s আর t এর lca হলো c. তাহলে cost(s, t) = cost(root, s) + cost(root, t) - 2cost(root, c) যেখানে cost(a, b) মানে হলো a হতে b এর path এর নোডগুলির cost এর যোগফল।

সুতরাং আমাদের কাছে যদি প্রদত্ত tree এর euler path বের করা থাকে তাহলে এই sequence এর শুরু থেকে কোনো নোড পর্যন্ত যোগফল বের করে করে আমরা s হতে t এর দূরত্ব বের করে ফেলতে পারি যেকোনো s এবং t এর জন্য। এখন কথা হলো আমাদের কাছে কিছু update অপারেশন দেওয়া থাকবে যেখানে নোডের weight পরিবর্তন হতে পারে। অর্থাৎ আমাদের sequence এর 2 টি element পরিবর্তন হতে পারে প্রতি update এ। আমরা যদি একটি segment tree বা binary indexed tree রাখি তাহলে এই update এবং segment sum এর কাজ $\mathrm{O}(\log V)$ সময়ে করে

ফেলতে পারব কারণ এই sequence এ 2V টি element থাকবে। আর এই দৈর্ঘ্যের একটি অ্যারের segment tree এর update এবং query করতে $\mathrm{O}(\log V)$ সময় লাগবে।



নকশা ৬.১: একটি tree এর Euler path

৬.১ অনুশীলনী

৬.১.১ সমস্যা

Simple

- UVa 839 Not so Mobile
- UVa 673 Parantheses Balance
- UVa 699 The Falling Leaves
- LOJ 1212 Double Ended Queue

Easy

- UVa 297 Quadtrees
- UVa 536 Tree Recovery
- UVa 246 10-20-30
- LOJ 1113 Discover the Web
- LOJ 1082 Array Queries
- LOJ 1083 Histogram
- LOJ 1103 Castle Walls
- LOJ 1135 Count the Multiples of 3
- LOJ 1207 Poster for Election
- LOJ 1293 Document Analyzer
- UVa 712 S-Trees
- UVa 127 "Accordian" Patience
- UVa 1322 Minimizing Maximizer
- LOJ 1303 Ferris Wheel
- LOJ 1162 Min Max Roads
- LOJ 1097 Lucky Number
- LOJ 1112 Curious Robin Hood
- LOJ 1187 Lining up Students
- LOJ 1266 Points in Rectangle

Medium

- UVa 10562 Undraw the Trees
- UVa 806 Spatial Structures
- UVa 1662 Brackets Removal
- LOJ 1120 Rectangle Union
- LOJ 1343 Aladdin and the Black Stones
- Codeforces Round 121 Div 1 C Fools and Roads
- UVa 12166 Equilibrium Mobile
- UVa 10410 Tree Reconstruction
- LOJ 1081 Square Queries
- LOJ 1424 New Land
- LOJ 1204 Weird Advertisement

৬.১.২ হিন্ট

LOJ 1212: Stl এর deque সম্পর্কে না জানলে দেখে নিতে পারো।

LOJ 1113: আমি আমার আগের সমাধান দেখে নিজেই মুগ্ধ! আমি দুইটি stack রেখেছিলাম। একটি back এর জন্য আরেকটা forward এর জন্য। যখন আমরা back কমান্ড পাবো তখন back এর stack হতে উপরের সাইটটি নিয়ে forward এর stack এ পুশ করেছি। যখন forward কমান্ড পেয়েছি তখন forward থেকে সাইট নিয়ে back এর stack এ পুশ করেছি। আর নতুন সাইট visit করলে forward এর stack ফাঁকা করে দিয়েছি আর তাকে back এর stack এ পুশ করেছি। সুন্দর সমাধান না?

LOJ 1082: এই সমস্যা segment tree ব্যবহার করে খুব সহজেই $\mathrm{O}(n)$ এ preprocessing এবং প্রতি query $\mathrm{O}(\log n)$ complexity তে সমাধান করতে পারবে। তবে তোমরা চাইলে $\mathrm{O}(n \log n)$ এ preprocessing এবং প্রতি query $\mathrm{O}(1)$ এও সমাধান করতে পারো। প্রোগ্রামিং কন্টেস্ট বইয়ের Data structure চ্যাপ্টারে "Static ডেটায় Query" টপিকে এই বিষয়ে আলোচনা করা হয়েছে।

LOJ 1083: এর সমাধান ইতোমধ্যেই প্রোগ্রামিং কন্টেস্ট বইয়ের stack এর অংশে আলোচনা করা হয়েছে (0-1 matrix এ সব 1 ওয়ালা সবচেয়ে বড় আয়তক্ষেত্র বের করার সমাধান দেখো)।

LOJ 1135: Segment tree এর একটি segment এ সংখ্যাগুলির $\mathrm{mod}3$ এর মান (0, 1 আর 2) কয়টি করে আছে তার count রাখতে হবে। LOJ 1080 এর মত আমরা চাইলে lazy propagate না করলেও পারি।

LOJ 1187: তুমি একটি array নাও। প্রথমে array এর সকল স্থানে 1 থাকবে। এখন ধর আমাদের দেওয়া সংখ্যাগুলির মাঝে শেষ সংখ্যা হলো x. এর মানে কি? এর মানে এর বামে x টি ছোট সংখ্যা আছে। এর মানে আমাদের array এর $x+1$ তম 1 যেখানে আছে সেটি হলো আমাদের শেষ সংখ্যা। আমরা শেষ সংখ্যা পেয়ে গিয়েছি, সুতরাং এই সংখ্যা array তে 0 করে দাও। এখন আমাদেরকে দেওয়া সংখ্যাদের মাঝের পরের সংখ্যা নাও আর একই ভাবে array থেকে বের কর কোনটি তোমার সংখ্যা। এখন কথা হলো খুব দ্রুত array এর উপর এই operation গুলি (k তম 1 বের করা এবং কোনো একটি স্থানকে 1 থেকে 0 করা) করবা কীভাবে? অবশ্যই segment tree!

LOJ 1207: এরকম সমস্যার ক্ষেত্রে শুরুতেই যেটা চিন্তা করতে হবে তাহলো কোনো একটি নির্দিষ্ট order এ তোমাকে poster গুলি প্রসেস করতে হবে। হয় বাম থেকে ডানে, অথবা উপর থেকে নিচে, বা নিচ থেকে উপরে ইত্যাদি। নিচ মানে হলো যেই poster আগে লাগানো হয়েছে, আর উপর মানে হলো যেই poster পরে লাগানো হয়েছে। যেমন এই সমস্যায় আমরা উপর থেকে নিচে প্রসেস করব। তাহলে কীভাবে হবে? আমরা একটি করে poster নিব আর একটি array তে লিখে রাখবো যে এই এই জায়গায় ইতোমধ্যে কোনো একটি poster এর অংশ আছে। ফলে যখন আমাদের জানার দরকার হবে যে- কোনো একটি poster দেখা যাবে কিনা, আমাদের যা করতে হবে তাহলো array এর ঐ অংশে দেখতে হবে কোনো স্থান আছে কিনা যেখানে এখনো কোনো poster নেই। আর এই কাজ তো segment tree এর সাহায্যে করতে পারবে।

LOJ 1266: 2 dimensional BIT ব্যবহার করে খুব সহজেই এই সমস্যা সমাধান করা যায়।

LOJ 1293: প্রবলেমটি দাঁড়ায়- একটি অ্যারে দেওয়া আছে যাতে নানা সংখ্যা দেওয়া আছে। এমন একটি range বের কর যাতে এই অ্যারেতে থাকা সকল ভিন্ন ভিন্ন সংখ্যা ঐ range এ থাকে। যদি এরকম একাধিক range থাকে তাহলে সবচেয়ে ছোট range, আর তাও যদি একাধিক থাকে তাহলে সবচেয়ে বামের range আউটপুট করতে হবে। নানা ভাবে এই সমস্যা সমাধান করা যায়। একটি উপায় হলো দুই হাত পদ্ধতি। মনে কর আমাদের সংখ্যাগুলি হলো 1 হতে n. তাহলে n সাইজের একটি অ্যারে নাও আর তা শুরুতে 0 করে দাও। আরও দুটি variable L এবং R নাও যাদের শুরুর মান হবে 0. আরও একটি variable লাগবে যার কাজ হলো L হতে R এর মাঝে কয়টি ভিন্ন সংখ্যা আছে তার count রাখা। এখন দেখো সেই count কি n কিনা। n হলে L হতে R এই range একটি candidate. এবার L এর মান এক বাড়াও। তাহলে একটি সংখ্যা বের হয়ে যাবে। তুমি count এর অ্যারেতে এই সংখ্যার count এক কমিয়ে দাও। যদি দেখো এর ফলে এই সংখ্যার count 0 হয়ে

গেছে এর মানে L হতে R এর ভেতরে ভিন্ন সংখ্যা একটি কমে গেছে। সুতরাং আমরা ভিন্ন সংখ্যার count ও এক কমিয়ে দেব। এবার মনে কর ভিন্ন সংখ্যার count এখনো n হয়নি। সেক্ষেত্রে আমরা R এর মান এক বাড়াব অর্থাৎ আমাদের L হতে R এর মাঝে একটি সংখ্যা বাড়বে। আমরা অ্যারেতে সেই সংখ্যার count এক বাড়াব। যদি দেখি count বাড়ানোর আগে সেই count টি 0 ছিল, এর মানে ভিন্ন সংখ্যা একটি বাড়ছে। সুতরাং আমরা ভিন্ন সংখ্যার count ও এক বাড়িয়ে দেব। এভাবে যত candidate রেঞ্জ পাওয়া যাবে তাদের মাঝের best range ই আমাদের উত্তর হবে।

UVa 12166: প্রতিটি bar কে horizontal থাকতে হবে। এখন তুমি যদি সবচেয়ে নিচের bar কে কল্পনা কর তাহলে বুঝবে এই bar এর দুই মাথার ভর সমান হতে হবে (ধরা যাক এর এক মাথায় w ভর আছে)। এই bar আবার যেই bar থেকে ঝুলে সেই bar এ 2w ভর প্রয়োগ করবে। এর মানে এর অন্য মাথাতেও 2w ভর থাকতে হবে। আরও যদি চিন্তা কর তাহলে বুঝবে d depth এ যদি w ভর থাকে সেটি root এ $w2^d$ ভর contribute করে। তাহলে আমাদের সমাধান কি? একটু যদি চিন্তা কর তাহলে বুঝবে যে সকল ভরের জন্য $w2^d$ বের করার পর যেই সংখ্যাটি সবচেয়ে বেশি বার আসবে সেই সংখ্যাগুলির ভর গুলি থাকবে, বাকি গুলি তোমাকে পরিবর্তন করতে হবে।

LOJ 1081: এই সমস্যা linear preprocessing এবং $\mathrm{O}(1)$ এ কীভাবে সমাধান করতে হবে তা প্রোগ্রামিং কন্টেস্ট বইয়ের Adhoc টেকনিক চ্যাপ্টারে বলা আছে।

LOJ 1120: প্রথমত আমাদের coordinate compress করতে হবে (না করলেও চলবে)। এরপর আমরা বাম হতে ডানে line sweep করব। এজন্য আমাদের এমন একটি segment tree লাগবে যেখানে দুই ধরনের অপারেশন support করবে। এক- i হতে j পর্যন্ত এক করে বাড়াও। দুই- পুরো অ্যারেতে কতগুলি 0 আছে। এরকম একটি segment tree যদি বানাতে পারো তাহলে বাকি টুকু সমাধান হয়ে যাবে। তোমাকে rectangle গুলিকে বামের মাথা আর ডানের মাথা অনুসারে sort করতে হবে (এগুলি একেকটি event)। ধরা যাক একটি rectangle হলো $(x_1, y_1) - (x_2, y_2)$ যেখানে $x_1 \leq x_2, y_1 \leq y_2$. যদি শুরুর মাথা পাও ($x = x_1$) তাহলে rectangle এর উপর আর নিচের মাথার মাঝে তুমি 1 যোগ করবে (segment tree টি হবে y axis বরাবর) অর্থাৎ segment tree তে $[y_1, y_2]$ রেঞ্জে 1 যোগ করবে। আর শেষ মাথা পেলে ($x = x_2$) ঐ একই রেঞ্জে এক বিয়োগ করবে। আর প্রতিবার দেখবে segment tree তে কতগুলি 0 আছে তাকে যদি তুমি x axis বরাবর jump এর পরিমাণ দিয়ে গুন কর তাহলে এই jump এর মাঝে কত খানি অংশ ফাঁকা আছে তা বের হয়ে যাবে। এখানে Jump বলতে বুঝাচ্ছি যে x অক্ষের এক event হতে পরের event এর মাঝের jump.

LOJ 1204: যদি খেয়াল করে থাকো যে K মাত্র 10 হতে পারে তাহলে আমার মনে হয় সহজেই সমাধান হয়ে যাবে এ সমস্যা। যদি তাও সমাধান না পেয়ে থাকো তাহলে চিন্তা কর, K = 1 হলে কি করবে, K = 2 হলে কি করবে এরকম। তেমন কিছুই না, আমাদের segment tree তে K সাইজের একটি অ্যারে রাখতে হবে যেটা বলবে i বার cover হয়েছে এরকম কয়টি বিন্দু আছে $[y_1, y_2]$ রেঞ্জে। সেই সাথে যেহেতু আমাদের segment ধরে update করতে হবে সেহেতু আমাদের lazy ব্যবহার করতে হবে। তবে lazy propagate করা যাবে না। কেন? তা তোমরা K সাইজের অ্যারে update করার সময় বুঝবে! একই সাথে যেহেতু coordinate অনেক বড় হতে পারে সেহেতু আমাদের coordinate compress করতে হবে। আর line sweep যে করতে হবে তা আশা করি ইতোমধ্যেই বুঝে গেছ! যদিও সমস্যাটা কঠিন না, কিন্তু একই সাথে অনেক গুলি টেকনিক দরকার এই সমাধানে তাই এটি একটু কঠিন।

অধ্যায় ৭

Greedy

Codeforces Round 26 B: Regular Bracket Sequence (26B)

সমস্যা: একটি 10^6 দৈর্ঘ্যের bracket sequence দেওয়া আছে। অর্থাৎ "(" আর ")" সম্বলিত একটি string. তুমি চাইলে এই string হতে কিছু bracket মুছে ফেলতে পারো। তোমার লক্ষ্য হলো সবচেয়ে বড় balanced bracket sequence প্রিন্ট করা।

সমাধান: আশা করি ইতিমধ্যেই শিখে গেছ কীভাবে stack ব্যবহার করে বা একটি counter রেখে একটি bracket sequence কী balanced কিনা তা বের করতে হয়। এখন এখানে বলেছে কিছু ব্র্যাকেট মুছে এটাকে সবচেয়ে বড় balanced bracket sequence বানাতে। এর মানে আমাদেরকে যত কম সম্ভব character মুছতে হবে। এখন আমাদের counter বা stack এর সমাধানের কথা চিন্তা কর। আমরা (পেলে এক বাড়াই বা stack এ সেই character (আমাদের এই সমাধানের ক্ষেত্রে index রাখতে হবে) আর) পেলে এক কমাই বা stack হতে উপরের element তুলে ফেলে দেই। যদি এই কমানোর সময় দেখো negative হয়ে গিয়েছে (বা stack হতে তুলতে গিয়ে যদি দেখ যে stack ফাঁকা) তাহলে তো invalid sequence হয়ে যাবে, এর মানে এই closing bracket নেওয়া যাবে না, তাই না? কিন্তু opening bracket এর কি হবে? সব গুলো নিবা? যেমন (() এরকম হলে তো প্রথম (বা দ্বিতীয়) opening bracket নিবা না। কিন্তু এটা কীভাবে বুঝা যাবে? আবার counter বা stack এর সমাধানের কথা মনে কর। একটা condition তো ছিল negative হওয়া যাবে না, আরেকটা condition কি ছিল? সবশেষে stack বা counter ফাঁকা কিনা। তাহলে এটা কীভাবে আমাদের সমাধানে ব্যবহার করব? সহজ, প্রসেস শেষে আমাদের stack এ যেই যেই opening bracket বাকি থাকবে তাদের নেব না- তাহলেই হবে। সুতরাং এভাবে বুঝব আমরা কাদের নেব কাদের নেব না। আমরা একটা boolean এর অ্যারে নিতে পারি যে কাদের নেব আর কাদের নেব না তা মার্ক করার জন্য। মার্ক করা হয়ে গেলে আমরা মার্ক না করা দের বাম হতে ডান পর্যন্ত প্রিন্ট করলেই উত্তর পেয়ে যাব।

Codeforces 44 J: Triminoes (44J)

সমস্যা: একটি 1000×1000 সাইজের গ্রিড দেওয়া আছে। গ্রিডটি দাবার বোর্ড এর মত সাদা কাল রঙ করা, অর্থাৎ দুটি সাদা ঘর পাশাপাশি না, আবার দুটি কাল ঘরও পাশাপাশি না। এই বোর্ড এর কিছু ঘর আবার কেটে ফেলা হয়েছে। কাল ঘর b, সাদা ঘর w আর কাটা ঘর . দিয়ে মার্ক করা আছে। তোমাকে এই বোর্ডে triminoes বসাতে হবে। Triminoes এর সাইজ 1×3 নাহয় 3×1 হয়। শর্ত হল triminoes গুলির দুই মাথায় সাদা ঘর থাকতে হবে আর মাঝে কালো ঘর থাকতে হবে। সেই সাথে বোর্ডের সকল ঘর (যেসব কাটা হয় নাই) তাদেরকে triminoes দ্বারা কাভার হতে

হবে, আর triminoes কে বোর্ডের বাহিরে থাকা যাবে না। তোমাকে বোর্ডটি trimino দ্বারা কাভার করে প্রিন্ট করতে হবে তবে কোনো একটি ঘর যেন আবার একাধিক trimino দ্বারা কাভার না থাকে। প্রিন্ট করতেও একটু কায়দা করতে হবে। প্রতিটি trimino কে a, b, c বা d দিয়ে মার্ক করতে হবে। অর্থাৎ পাশাপাশি একই অক্ষরের তিনটি ঘর থাকলে আমরা বুঝব যে তারা একটি trimino. আর খেয়াল রাখতে হবে যে পাশাপাশি দুটি trimino এর অক্ষর যেন একই না হয়। যদি এই শর্তগুলি মেনে trimino বসান সম্ভব না হয় তাহলে বল যে সমাধান সম্ভব না।

সমাধান: বেশ সুন্দর একটি সমস্যা। একটু চিন্তা করে দেখ কোন কোন স্থানে আমাদের trimino এক ভাবেই বসাতেই হবে। মনে কর বোর্ডের উপরের সারির কোনো একটি ঘর কালো তাহলে আমাদের অবশ্যই horizontal ভাবে একটি trimino বসাতে হবে। একই ভাবে যদি বাম দিকের কলাম বরাবর একটি কালো ঘর থাকে তাহলে তাকে কাভার করে আমাদেরকে vertically একটি trimino বসাতে হবে। এছাড়া কিন্তু আমাদের কোনো গতি নাই তাই না? এগুলি বসানোর পর আর কোন গুলোতে বসাতেই হবে? একই ভাবে ডানের কলাম আর নিচের সারি তাই না? কিন্তু এর পর? ভেতরের দিকের ঘর নিয়ে চিন্তা করার আগে একবার ভেবে দেখ তো বাইরের যেসব সাদা ঘর এখনও বাকি আছে তাদের ক্ষেত্রে কি হবে? উপরের সারির সাদা ঘরগুলির ক্ষেত্রে vertically আর বামের কলামের বাকি সাদা ঘরের ক্ষেত্রে horizontally করে trimino বসাতে হবে। এভাবে নিচের সারি আর ডানের কলামেও। তাহলে এভাবে আমাদের চতুর্দিকের ঘর গুলি কাভার হয়ে যাবে। একই ভাবে আমরা ভেতরের দিকের বর্ডারের ঘরগুলি পুরন করব আর এভাবে চলতে থাকবে। একটা জিনিস, যদি দেখ যে উপরে বাম দিকের কোণার ঘর কালো বা বর্ডারের কালো ঘর কাভার করে যখন সাদা ঘরগুলি কাভার করে trimino বসাচ্ছিলে তখন যদি দেখ যে পাশের ঘরে সাদা নেই তাহলে কিন্তু কোনো সমাধান নেই তাই না? কারণ প্রথম ক্ষেত্রে তার উপরে বা বামে সাদা ঘর নেই, পরের ক্ষেত্রেও তাই উপরে বা বামে সাদা ঘর নেই। এই কথা অন্যান্য কোণার ঘরের জন্যেও প্রযোজ্য।

তাহলে আমাদের সমস্যার প্রায় সমাধান শেষ, শুধু বাকি আছে এই trimino দেরকে a, b, c আর d দিয়ে মার্ক করা। এটা মনে হচ্ছে বেশ কঠিন কাজ। কীভাবে বুঝা যাবে যে এই চারটা দিয়ে রঙ করা যাবে বা যাবে না? আচ্ছা, আমরা কি সবসময় এই চারটা দিয়ে রঙ করতে পারব? একটু কাগজে কলমে চেষ্টা করে দেখতে পার যে চারটা দিয়ে রঙ করা যায় না অথচ trimino বসান যায় এরকম সম্ভব কিনা। কিছুক্ষণ চেষ্টা করলে দেখবা যে- না সম্ভব না এরকম বোর্ড বানানো। যদিও এটা বুঝা বা বিশ্বাস করা কঠিন কিন্তু তাও তুমি মনে করতে পার যে আসলে যদি trimino বসান যায় তাহলে রঙও করা যাবে। আসলে তোমরা যদি four color theorem জেনে থাক তাহলে বুঝবে যে চারটি রঙ দ্বারা সকল trimino রঙ করা সম্ভব। এখন কীভাবে রঙ করবে সেটা চিন্তা কর। প্রথমত আমাদের একটু systematic ভাবে বসাতে হবে। যেখানে সেখানে trimino বসান যাবে না, কারণ তা করলে কোনো একটি trimino এর রঙ করা কঠিন হয়ে যাবে। আমরা কিছুক্ষণ আগে চতুর্দিকে বসিয়েছিলাম trimino, আমরা চাইলে কি আরেকটু কম ছড়িয়ে বসাতে পারি? আমরা চাইলে মনে হয় চতুর্দিকে না বসিয়ে শুধু উপরে আর বামে বসাতে পারি। এরপর বোর্ড এক সারি আর এক কলাম ছোট হয়ে যাবে। নতুন বোর্ডে আমরা আবার trimino বসাব উপরের সারি আর বামের কলামে। আমরা কিন্তু চাইলে শুধু উপরের সারিতেও বসাতে পারি। তাহলে বোর্ডের সাইজ এক সারি করে কমবে। কোডও সহজ হবে। মানে, প্রথমে দেখবে উপরের রো তে কোনো কালো ঘর আছে কিনা, থাকলে তাকে cover করে একটি horizontal trimino বসাতে হবে। এরপর দেখ কোনো সাদা বাকি আছে কিনা, থাকলে তাদের কাভার করে vertical trimino বসাব। এভাবে যদি বসাই তাহলে কিন্তু আমরা যখন একটি trimino বসাব তখন এর সাথে লেগে থাকা trimino এর পরিমাণও কিন্তু কম। যখন আমরা trimino কে vertically বসিয়েছি তখন এর উপরে একটি আর বামে ডানে দুইটি trimino থাকতে পারে (পরে একে লাগিয়ে অন্য কোনো trimino বসাতে হতে পারে, তবে আমরা সেটি নিয়ে ভাবছি না, আমরা ভাবছি একে বসানোর সময় ইতোমধ্যেই কয়টি trimino একে স্পর্শ করে আছে)। যখন একটি trimino কে horizontal ভাবে বসিয়েছি তখন অবস্থা একটু ট্রিকি। উপরে খুব জোড় একটা trimino থাকবে, কারণ horizontally একটি trimino বসালে এর উপর একটি সাদা ঘর থাকতে পারে, আর ডানে বামে খুব জোড় একটা করে বসান যায়। সুতরাং দুই ক্ষেত্রেই তোমার কাছে চারটি রঙের মাঝে একটি রঙ অবশ্যই বাকি থাকবে। ব্যাস হয়ে গেল! তবে আমার মনে হয় কি যে মাত্র তিনটি রঙ দিয়েই এদের রঙ করা সম্ভব। তোমরা একটু চিন্তা করে দেখতে পার!

Codeforces 45 D: Event Dates (45D)

সমস্যা: N টি ভ্যারিয়েবল আছে। প্রতিটি ভ্যারিয়েবলের জন্য একটি করে রেঞ্জ আছে, ধরা যাক i তম ভ্যারিয়েবলের জন্য এই রেঞ্জ হচ্ছে $[l_i, r_i]$. তোমাকে প্রতিটি ভ্যারিয়েবলের জন্য একটি মান নির্বাচন করতে হবে যেন একাধিক ভ্যারিয়েবলের মান একই না হয় এবং প্রতিটি ভ্যারিয়েবলের মান তার রেঞ্জের মাঝে হয়। এখানে $N \leq 100$ এবং $1 \leq l_i \leq r_i \leq 10^7$. আর এটা বলা আছে যে প্রদত্ত সকল test case এর জন্য সমাধান আছে।

সমাধান: Greedy এর খুব কমন একটি সমস্যা। আমি এখানে এমন একটি সমাধান বলব যেটা N এর বড় মানের জন্যও কাজ করবে। প্রথমে একটি তুলনামুলক খারাপ সমাধান বলি এরপর তাকে উন্নতি করব।

প্রথমে সকল ভ্যারিয়েবল unassigned. এরপর আমরা i এর একটি লুপ চালাব যেটি 1 হতে 10^7 এ যাবে। প্রতিটি i এ গিয়ে দেখব যে এখান থেকে কোনো ভ্যারিয়েবলের রেঞ্জ শুরু হয় কিনা। হলে আমরা সেই ভ্যারিয়েবলকে active হিসেবে মার্ক করব। এরপর এখন যেসকল ভ্যারিয়েবল active, তাদের মাঝে যেটির r এর মান সবচেয়ে ছোট, তার মান i assign করব এবং এই ভ্যারিয়েবলকে আর active রাখব না। এভাবে সব ভ্যারিয়েবলের মান assign করলেই হবে। কিন্তু কোনো এক সময়ে এসে যদি এমন হয় যে, আমরা i এ আছি, কোনো একটি ভ্যারিয়েবল active আছে কিন্তু i সেই active variable এর r এর থেকেও বেশি, এর মানে আমাদের এই সমস্যা তখন সমাধান করা যাবে না। যদিও এই সমস্যায় এরকম কখনও হবে না, কারণ বলা আছে যে সবসময় সমাধান থাকবে। কিন্তু হয়তো অন্য সমস্যায় সবসময় সমাধান নাও থাকতে পারে। যাই হোক, এই পদ্ধতিতে আমাদের সময় লাগছে $N \times 10^7$ কারণ আমরা প্রতিটি স্থানে গিয়ে সকল ভ্যারিয়েবলের উপর লুপ চালিয়ে নির্বাচন করছি কোন ভ্যারিয়েবলে মান assign করব।

আমরা i এর লুপ না চালিয়ে একটি heap বা priority queue ব্যবহার করতে পারি। তাহলে কিন্তু আমাদের সময় কমে $10^7 \log N$ হয়ে যাবে। এই heap এ থাকবে বর্তমানে active সকল ভ্যারিয়েবল এবং তাদের r এর মান। এদের মাঝে r যার সবথেকে কম সে এই heap এর সবচেয়ে উপরে থাকবে। অর্থাৎ আমাদের সমাধান হল, আমরা আগের মত i এর লুপ চালাব। কোনো একটি i এ এসে প্রথমে দেখব এখানে কোন কোন ভ্যারিয়েবলের রেঞ্জ শুরু হয়, তাদের আমরা heap এ ঢুকিয়ে দেব। এরপর দেখব heap এ কেউ আছে কিনা, থাকলে সবচেয়ে উপরের ভ্যারিয়েবল নিব এবং তাকে i মান দেব।

এই সমাধানকে আরেকটু চিন্তা করে $\mathrm{O}(N \log N)$ এ কমাতে পার। খেয়াল কর কোন রেঞ্জ কোথায় শুরু হয়েছে এটা ওতটা গুরুত্বপূর্ণ না। গুরুত্ব পূর্ণ হল তাদের order. আমরা শুরুতেই ভ্যারিয়েবল গুলিকে সর্ট করব। সর্ট করার criteria হবে- তাদের রেঞ্জের শুরুর মাথা। অর্থাৎ যাদের রেঞ্জ আগে শুরু হয় তারা আগে থাকবে। এখন তুমি একটি variable নাও ধরা যাক i যা নির্দেশ করবে তুমি এখন কোন মানে আছে। ঠিক উপরের i এর মত। কিন্তু আমাদেরকে এই i এর লুপ চালাতে হবে না। আরেকটা variable রাখো, ধরা যাক at, যা আমাদের variable এর অ্যারেতে একটি স্থান নির্দেশ করবে। শুরুতে at অ্যারের শুরুর স্থান নির্দেশ করবে। আর i এর শুরুর মান হবে 1. সেই সাথে আমাদের একটি priority queue থাকবে যা শুরুতে ফাঁকা থাকবে। আগের মতই এই priority queue তে থাকবে ভ্যারিয়েবলগুলি আর যেই ভ্যারিয়েবল এর রেঞ্জের শেষ মাথা ছোট তা থাকবে priority queue এর উপরে। এখন একটি while লুপ চলবে যতক্ষণ না সব ভ্যারিয়েবল প্রসেস হয়ে যায়। এখন কথা হল লুপের ভেতরে কি করবা।

- প্রথমে দেখ যে at যাকে পয়েন্ট করে আছে তার শুরুর মাথা কি i এর সমান কিনা। তাহলে at যাকে পয়েন্ট করে আছে সেই ভ্যারিয়েবলকে priority queue তে প্রবেশ করাও, at কে অ্যারের পরের ভ্যারিয়েবলে পয়েন্ট কর আর while loop কে continue কর।
- যদি উপরের condition সত্যি না হয় তাহলে চেক কর যে তোমার priority queue ফাঁকা কিনা। যদি ফাঁকা হয় তাহলে at যাকে পয়েন্ট করে আছে তার রেঞ্জের শুরুর মাথার মান i এর মান হিসাবে সেট কর আর while loop কে continue কর।
- যদি উপরের কোনো condition ই সত্যি না হয় তার মানে আমাদের priority queue তে কেউ না কেউ আছে। এখন priority queue থেকে উপরের ভ্যারিয়েবল তুল আর দেখ তার

রেঞ্জের শেষ মাথার ভেতরে কি i আছে কিনা। যদি না থাকে তার মানে এই সমস্যার সমাধান নেই (যদিও এই সমস্যার ক্ষেত্রে সবসময় সমাধান থাকবে বলেছে)। যদি i ভেতরে থাকে তাহলে এই ভ্যারিয়েবল কে i মান দাও। i এর মান এক বাড়াও আর while loop কে continue কর।

তাহলেই $\mathrm{O}(N \log N)$ এ এই সমস্যা সমাধান হয়ে যাবে।

Codeforces Round 57 D: Eternal Victory (61D)

সমস্যা: তোমাকে একটি n নোডের ট্রি দেওয়া আছে। এর মান সর্বোচ্চ 100,000 হতে পারে। ট্রির প্রতিটি edge এর cost হবে non-negative এবং সর্বোচ্চ 20000. তোমাকে সবচেয়ে কম খরচের path বের করতে হবে যা 1 হতে আরম্ভ করে সকল নোড ঘুরে যেকোনো নোডে শেষ হয়।

সমাধান: এটিও বেশ কমন সমস্যা। মনে কর তোমাকে বলল যে 1 হতে আরম্ভ করে 1 এ এসেই শেষ করতে হবে। তাহলে উত্তর কি হত? সহজ, সকল edge এর cost এর যোগফলের দ্বিগুন। কেন? কিছুটা dfs এর মত করে traverse করতে পার। নিচে যেতে থাকবা যতক্ষণ যাওয়া যায়, এর পর ফিরে এসে আবার নিচে যাওয়ার চেষ্টা করবে এরকম ভাবে করলে আসলে প্রতিটি edge দিয়ে একবার মাত্র নামবা আর একবার মাত্র উঠবা।

এখন এখানে বলেছে তুমি যেকোনো স্থানে শেষ করতে পারবা। এর মানে তুমি যেখানে শেষ করেছ সেখান থেকে 1 এ ফেরত আসার দরকার নেই। সুতরাং বুঝতেই পারছ যে 1 হতে যে নোডে যেতে সবচেয়ে বেশি cost লাগে, সেই সবচেয়ে দুরের নোডে যদি তুমি শেষ কর তাহলেই তোমার লাভ। এর মানে এর আগের উত্তর হতে সবচেয়ে দুরের নোডের cost বিয়োগ হবে।

Codeforces Round 66 B: Need For Brake (73B)

সমস্যা: একটি রেসিং খেলায় N জন খেলোয়াড় আছে এবং তাদের বর্তমান পয়েন্ট $a_1, a_2, \ldots a_N$. প্রত্যেকের নামও দেওয়া আছে। এখন একটি রেস হবে যেখানে প্রথম m স্থান অর্জনকারীদের যথাক্রমে $b_1, b_2, \ldots b_m$ পয়েন্ট দেওয়া হবে। তুমি প্রথম খেলোয়াড় অর্থাৎ তোমার বর্তমান পয়েন্ট a_1. তোমাকে বলতে হবে তুমি সবচেয়ে ভাল কততম স্থান দখল করতে পার আর সবচেয়ে খারাপ কততম স্থান দখল করতে পার। যদি রেস শেষে একাধিক খেলোয়াড়ের পয়েন্ট সমান হয় তাহলে তাদের নামের lexicographical order এ তাদের rank করা হবে। $N \leq 10^5$.

সমাধান: একটু চিন্তা করার সমস্যা। প্রথমত আমরা মনে করতে পারি যে $m = n$ কারণ তা নাহলে আমরা ইচ্ছা মত অতিরিক্ত $b_i = 0$ নিতে পারি। প্রথমে বের করা যাক আমি কত খারাপ স্থান পেতে পারি। এজন্য অবশ্যই আমাকে সবচেয়ে ছোট b_i পেতে হবে। এখন কথা হল অন্যদেরকে আমি কীভাবে b_i গুলি দিলে সর্বোচ্চ জন আমার চেয়ে ভাল rank পাবে। আমরা প্রথমেই বাকি পয়েন্ট গুলি সর্ট করে নেই এবং সেই সাথে বাকি খেলোয়াড় দেরও সর্ট করে নেই। এবার তুমি সবচেয়ে খারাপ খেলোয়াড়কে নাও, এবং দেখ একে যদি তুমি সবচেয়ে বড় score দাও তাহলে তার rank কি তোমার থেকে ভাল হয় কিনা, হলে তাকে সেই পয়েন্ট দাও এবং একই ভাবে এর পরের খারাপ খেলোয়াড় আর পরের বড় পয়েন্ট নিয়ে একই চেষ্টা কর। আর না হলে, এই খেলোয়াড়কে বাদ দিয়ে পরের খারাপ খেলোয়াড় নাও এবং তাকে এই সবচেয়ে বড় পয়েন্ট দেবার চেষ্টা কর। এরকম করে তুমি যদি চেষ্টা করতে থাক তাহলে সবচেয়ে বেশি খেলোয়াড় তোমার থেকে বেশি rank পাবে। এই কাজ কিন্তু তুমি $\mathrm{O}(n)$ এ খুব সহজেই করতে পারবে, তোমাকে শুধু দুটি অ্যারেতে দুটি index কে পয়েন্ট করে রাখার জন্য দুটি ভ্যারিয়েবল লাগবে।

ঠিক উলটো করে তুমি কীভাবে বের করবে তুমি সবচেয়ে ভাল কত rank পাবে? অবশ্যই এই ক্ষেত্রে তুমি সবচেয়ে বড় score নেবে। এখন কথা হল বাকি score গুলি বাকি দের মাঝে কীভাবে বিতরন করলে সবচেয়ে কম জন তোমার উপরে থাকবে বা সবচেয়ে বেশি জন তোমার নিচে থাকবে। আগের মতই বাকি score এবং বাকি খেলোয়াড় দের সর্ট করে ফেল। আবারো সবথেকে খারাপ খেলোয়াড়কে নাও কিন্তু এবার দেখ সবচেয়ে বড় কোন score দিলে এ তোমার নিচে থাকবে, তাকে

সেই score দিয়ে দাও। এরপর পরের খারাপ খেলোয়াড় নাও এবং একই ভাবে দেখ একে সবচেয়ে বড় কোন score দিলে সে তোমার নিচে থাকবে। এভাবে তুমি যদি score গুলি assign কর তাহলে দেখবে যে সবচেয়ে বেশি জন তোমার নিচে থাকবে। এই কাজও তুমি $O(n)$ এ খুব সহজেই করতে হবে। অর্থাৎ তুমি দুটি লিস্ট সর্ট করার পর লিনিয়ার সময়ে বাকি সমস্যাটুকু সমাধান করতে পারবে।

এধরনের সমস্যার ক্ষেত্রে binary search ও অনেক সময় কাজে লাগে। যেমন তোমরা চাইলে এই সমস্যা binary search দিয়েও চিন্তা করতে পার। মনে কর তোমরা binary search করে ঠিক করবে যে "কমপক্ষে g সংখ্যক লোককে আমার উপরে রাখা সম্ভব কিনা"। এবার এই প্রশ্নের জবাব তুমি দেবার চেষ্টা কর তাহলেই সমাধান হয়ে যাবে।

Codeforces Round 79 Div 1 D: Castle (101D)

সমস্যা: একটি n নোডের rooted weighted ট্রি আছে ($n \leq 10^5$)। অর্থাৎ প্রতিটি edge এর একটি করে weight আছে ($weight \leq 1000$), মনে করতে পার কোনো একটি edge এর এক মাথা হতে অপর মাথায় যেতে যত সময় লাগে সেটিই weight. তুমি ট্রিটির root হতে আরম্ভ করে একে একে সব নোড ঘুরবা। শর্ত হলো এই ঘুরার সময় তুমি এক edge দুই বারের বেশি ব্যবহার করতে পারবে না। Root বাদে বাকি কোনো একটি নোডে treasure আছে। কোন নোডে আছে সেটা তুমি আগে থেকে জানো না। ট্রি এর কোন edge এর weight কত তা তুমি আগে থেকে জান। তোমাকে treasure খুঁজে পাবার expected সময় কমাতে হবে।

সমাধান: কোনো edge দুই বারের বেশি ব্যবহার করব না আবার সকল নোড ভিজিট করব এর মানে হলো আমরা একটি ট্রি এর dfs traversal বা euler tour এর মত করে ভিজিট করব। অন্য ভাবে বলা যায়, মনে কর তুমি যদি কোনো নোডে আস তাহলে এর নিচের পুরো subtree ভিজিট না করে ফিরে যাবা না। কারণ ফিরে গেলে তো তোমার নোড হতে তোমার parent এর যেই edge তা দুই বার ব্যবহার হয়ে যাবে। আর আমাদের বলা আছে কোনো edge দুবারের বেশি ব্যবহার করা যাবে না। এখন কথা হল, কোনো নোডের নিচের পুরো subtree কে ভিজিট করার সময় যদি treasure পেয়ে যাও তাহলে সেটা আলাদা কথা, কিন্তু যদি না পাও তাহলে পুরো subtree ভিজিট করে ফেরত যাবা এবং সেই ক্ষেত্রে খরচ লাগবে পুরো subtree এর edge সমুহের weight যোগ করলে যত হয় তার দ্বিগুন। কেন? কারণ প্রতিটি edge এর ক্ষেত্রে তুমি নিচে নামার সময় একবার আর ফেরত যাবার সময় একবার, মোট দুইবার করে প্রতিটি edge ব্যবহার করছ।

তাহলে প্রশ্ন হলো আমরা কি অর্ডারে একটি নোডের নিচে থাকা subtree গুলি ভিজিট করব? খেয়াল কর, কি অর্ডারে ভিজিট করব তা কিন্তু fixed. এমন না যে কিছু দূর ঘুরলাম এর পর ঠিক করব বাকিটুকু কীভাবে ঘুরব। বরং কার পরে কই যাব তা আগে থেকেই নির্ধারিত। যদি এর মাঝে কোনো এক সময়ে treasure পেয়ে যাই তাহলে সেখানেই শেষ, আর না পেলে আগে নির্ধারিত রাস্তায় যেতে থাকব। সুতরাং যখন একটি নোডে আছ তখন এটা fixed যে কোন subtree এর পর কোন subtree যাব। আমরা যদি এই অর্ডারটা বের করতে পারি তাহলেই সমাধান হয়ে যাবে।

মনে কর আমরা এখন যেই নোডে আছি তার child সমুহের জন্য উত্তর জানি। অর্থাৎ সেসব child এ গেলে এবং সেসব child এর subtree তে যদি treasure থাকে তাহলে expected কত সময় লাগে তা খুঁজে বের করতে আমরা জানি। ধরা যাক i তম child এর জন্য এই উত্তর হলো a_i (অর্থাৎ আমরা ধরে নিচ্ছি যে এই subtree তে আমাদের treasure আছে, এখন বলতে হবে optimal ভাবে আমরা যদি ভিজিট করি তাহলে expected কত সময় লাগবে সেই treasure বের করতে) আর এই a_i এ parent হতে i তম child এর edge এর weight ও আছে। আরও ধরা যাক i তম child এর subtree এর সাইজ s_i. হিসাবের জন্য আমাদের আরও কয়েকটি সংখ্যা জানার দরকার। একটি হলো আমরা বর্তমানে যেই নোডে আছি তার subtree সমুহে মোট কতগুলি নোড আছে, ধরা যাক m টি ($= \sum s_i$). আর i তম subtree এর সকল edge এর weight এর যোগফলের দ্বিগুন ধরা যাক T_i (parent হতে i তম child এর edge এর weight সহ)। মনে কর আমরা যেই নোডের জন্য উত্তর বের করছি তার k টি child আছে। আমার মনে হয় এই সংখ্যাগুলি জানলেই হবে। এখন মনে কর আমরা প্রথমে 1 নম্বর child এর subtree ঘুরব, এরপর 2 নম্বর এরকম করে k নম্বর। তাহলে আমাদের expected কত সময় লাগবে? এখানে আমরা চাইলে probability খাটাতে পারি। প্রথম subtree

তে treasure থাকার সম্ভাবনা s_1/k, সেই ক্ষেত্রে সময় লাগে a_1. দ্বিতীয় subtree তে treasure থাকার সম্ভাবনা s_2/k, সেই ক্ষেত্রে সময় লাগে $T_1 + a_2$. কেন? কারণ আমরা প্রথমে প্রথম subtree খুঁজেছি এবং পাই নাই (সুতরাং T_1 সময় লেগে গিয়েছে) এরপর এই subtree তে এসেছি এবং treasure খুঁজে পেতে a_2 সময় লেগে গিয়েছে। এরকম করে k টি child. সুতরাং এদের সবাইকে যোগ করলে দাঁড়ায় $s_1/k*(a_1)+s_2/k*(T_1+a_2)+\ldots+s_k/k*(T_1+T_2+\ldots T_{k-1}+a_k)$. এবার মূল প্রশ্ন, এই k টি child কে কোন অর্ডারে ভিজিট করলে আমাদের cost সবচেয়ে কম হবে। এক্ষেত্রে সাধারনত যেটা দেখতে হয় তাহল কার আগে থাকা উচিত i নাকি $i+1$ সেটা দেখা। i যদি আগে থাকে তাহলে তো আমরা জানি আমাদের মোট cost কত। তাও আমরা একটু অন্য ভাবে sum টাকে লিখি- $s_1*(a_1)+\ldots+s_i*(T_1+T_2+\ldots T_{i-1}+a_i)+s_{i+1}*(T_1+\ldots T_i+a_{i+1})+\ldots$ যেহেতু $1/k$ একটি constant তাই একে বাদ দিয়ে দিয়েছি। এখন একটু কষ্ট করে বের কর যদি $i+1$ আগে থাকে তাহলে cost কত- $s_1 * (a_1) + \ldots + s_{i+1} * (T_1 + T_2 + \ldots T_{i-1} + a_{i+1}) + s_i * (T_1 + \ldots T_{i-1} + T_{i+1} + a_i) + \ldots$. যদি আমরা ধরি i আগে থাকার cost A আর i পরে থাকার cost B তাহলে এখন বলতে হবে কে বড় A নাকি B? চেষ্টা করে দেখা যাক। আমরা $A-B$ বের করে দেখি $A - B = s_{i+1} * T_i - s_i * T_{i+1}$. সুতরাং i তম subtree আগে থাকা উচিত যদি $A - B < 0$ হয় বা $\frac{T_i}{s_i} < \frac{T_{i+1}}{s_{i+1}}$ হয়। এর মানে আসলে আমাদেরকে subtree সমুহকে T_i/s_i অনুসারে সর্ট করতে হবে। তাহলেই হবে। খেয়াল কর আমরা এখানে শুধু পাশাপাশি দুটি জিনিস compare করেছি। আমাদেরকে কি পাশাপাশি নয় এরকম জিনিসও হিসাব করে দেখতে হবে? সাধারণত তার দরকার হয় না। তুমি চাইলে খেটে খুটে সেই হিসাব করতে পার। কিন্তু আমার দেখা মতে বেশির (হয়তো সবসময়) ভাগ সময়ই শুধু পাশাপাশি দেখলেই চলে।

Codeforces Round 99 Div 1 D Digits Permutation (138B)

সমস্যা: একটি n দৈর্ঘ্যের সংখ্যা দেওয়া আছে ($n \leq 10^5$)। তোমাকে এই সংখ্যার অংকগুলিকে permute করে আলাদা আলাদা দুটি সংখ্যা বানাতে হবে। চাইলে দুটি সংখ্যা একই হতে পারে। শর্ত হল এই দুটি সংখ্যা যোগ করলে সবচেয়ে বেশি সংখ্যক 0 শেষে থাকে। যেমন তোমাকে দেওয়া সংখ্যা যদি হয় 198 তাহলে তুমি চাইলে 981 আর 819 বানাতে পার এতে করে শেষে দুটি শূন্য থাকবে- $981 + 819 = 1800$. তুমি চাইলে তোমার সংখ্যার শুরুতে শূন্য রাখতে পার।

সমাধান: খুব একটা কঠিন সমস্যা না। একদম ডানের দুটি যোগ করলে 10 হবে। এরপর পরের গুল যোগ করলে 9 হবে আর তাহলে 10 এর হাতের 1 এই 9 এর সাথে যোগ হয়ে 10 হবে। যেমন উপরের উদাহরনেও এই একই ঘটনা ঘটেছে। সুতরাং আমরা 10 বানাতে পারি এভাবে- $(1,9),(2,8)\ldots$ আর 9 বানাতে পারি এভাবে- $(0,9),(1,8),(2,7),\ldots$. তাহলে আমরা যেটা করব আমরা একটা লুপ চালিয়ে ঠিক করব কোন ভাবে আমরা এই 10 বানাব। এরপর সর্বোচ্চ 9 কীভাবে বানানো যায় সেটা কিন্তু greedy, কারণ তুমি জান উপরের কার সাথে নিচের কাকে মিলাতে হবে। সুতরাং এদের দুজনের মাঝে সবচেয়ে কম সংখ্যক তুমি নিতে পারবে। যেমন মনে কর উপরে 1 আছে x টি আর নিচে 8 আছে y টি, তাহলে তুমি এদের নিয়ে $min(x,y)$ টি 9 বানাতে পারবে।

কিন্তু উপরের সমাধান সম্পূর্ণ ঠিক নেই। তুমি যদি কিছু case নিয়ে চিন্তা করে দেখ তাহলেই বুঝবে। যেমন 10 এর ক্ষেত্রে উপরের সমাধান বলবে কোনো শূন্য থাকবে না শেষে কিন্তু আমরা চাইলে $10 + 10 = 20$ করতে পারি। অর্থাৎ $0 + 0 = 0$ এর case আমরা ঠিক ভাবে হ্যান্ডেল করি নাই। আগের সব কিছুই ঠিক আছে শুধু আরেকটা জিনিস চেষ্টা করতে হবে তাহল শেষে দুটি সংখ্যারই কিছু 0 থাকতে পারবে। কতগুলি? তাতো আমাদের জানা নেই! উপায় কি? একটি লুপ চালিয়ে দাও কতগুলি 0 শেষে থাকবে তার উপর, এর পর আগের মত করে লুপ চালিয়ে বের কর শেষের 10 কীভাবে বানাবা এবং এরপর কতগুলি 9 বানানো যায়। তাহলেই হবে।

Codeforces Round 100 C New Year Snowmen (140C)

সমস্যা: n টি সংখ্যা আছে ($n \leq 10^5$). তোমাকে এদের হতে তিনটি করে ভিন্ন ভিন্ন সংখ্যা নিয়ে একটি সেট বানাতে হবে, অর্থাৎ এদের কোনো দুটিই যেন সমান না হয়। বলতে হবে এরকম সর্বোচ্চ কতটি সেট বানানো যাবে।

সমাধান: প্রথমে আমরা বের করে নেই কোন সংখ্যা কয়বার করে আছে। এখন খেয়াল কর, আমাদের optimal solution এ অবশ্যই এরকম একটি সেট থাকবে যাতে সবচেয়ে বেশি বার থাকা সংখ্যাটি আছে। কেন? কারণ মনে কর যে এরকম কোনো সেট নেই। তাহলে আমরা কি চাইলেই যেকোনো সেট হতে যেকোনো সংখ্যা সরিয়ে আমাদের সেই সবচেয়ে বেশি বার থাকা সংখ্যাটি নিতে পারি না? অবশ্যই পারি। এখন চিন্তা কর, আমরা কি এরকম সেট নিতে পারি যাতে সবচেয়ে বেশি বার থাকা দুটি সংখ্যা আছে (অর্থাৎ যে দুটি সংখ্যা সবচেয়ে বেশি বার আছে)? দেখা যাক। মনে কর সবচেয়ে বেশি বার আছে A এর পর B. আমাদের কাছে যদি $\{A, B, ?\}$ এর মত সেট থাকে তাহলে তো হয়েই গেল। মনে কর নেই। কিন্তু কিছুক্ষণ আগেই আমরা দেখেছি A ওয়ালা একটি সেট অবশ্যই আছে। ধরা যাক সেই সেটটি হল $\{A, X, Y\}$. এখন খুঁজে দেখ এমন কোনো সেট আছে কিনা যেখানে B আর X একই সাথে নেই। এরকম সেট খুঁজে পাওয়া সম্ভব কারণ B এর পরিমান X এর থেকে বেশি। যদি এরকম কোনো সেট না থাকে তার মানে অতিরিক্ত B আছে এবং সেটি দিয়েও কাজ চালান যাবে। তাহলে এই দুটির যেটিই ঘটুক না কেন আমরা X এর পরিবর্তে B বসাতে পারব। একই ভাবে আমরা এর পরের বড় সংখ্যার জন্যও একই রকম জিনিস প্রমাণ করতে পারি। এর মানে দাঁড়াল আমরা যদি সবচেয়ে বেশি আছে এরকম তিনটি সংখ্যা নিয়ে সেট গঠন করি তাহলেই optimal solution পাওয়া যাবে। এরকম করে প্রতিবার আমরা সবচেয়ে বেশি আছে এরকম তিনটি সংখ্যা নিয়ে নিয়ে সেট গঠন করব।

Codeforces Round 109 Div 1 A Hometask (154A)

সমস্যা: তোমাকে character এর কিছু forbidden pair দেওয়া আছে (pair এর character দুটি আলাদা হবে)। অর্থাৎ প্রতি pair এর character গুলি পাশাপাশি থাকতে পারবে না। এরকম সর্বোচ্চ 13 টি pair দেওয়া আছে। সেই সাথে একই character একাধিক forbidden pair এ থাকবে না। তোমাকে একটি স্ট্রিং দেওয়া আছে যার দৈর্ঘ্য সর্বোচ্চ 10^5 হতে পারে। তোমাকে এই স্ট্রিং হতে সবচেয়ে কম সংখ্যক character মুছে ফেলতে হবে যেন কোনো forbidden pair না থাকে। যেমন তোমার স্ট্রিং যদি হয় ababa এবং একটি forbidden pair থাকে- ab তাহলে তুমি দুইটি b মুছে ফেলে aaa বানিয়ে ফেলবে। এর থেকে কমে সম্ভব না।

সমাধান: মনে কর স্ট্রিংটির তুমি বাম হতে ডানে আসছ। পরপর দুটি character নিয়ে যদি দেখ এরা forbidden pair তাহলে তো এদের এক জনকে মুছতে হবেই। কোনটিকে মুছবা? মনে হয় পরের জনকে। কারণ প্রথম জনকে মুছলে এই pair টা ভেঙ্গে গেল আর কোনো লাভ নেই। কিন্তু পরের জনকে মুছলে হয়তো এই পরের জন আর তার পরের জন মিলে যদি আরেকটি pair তৈরি করে রাখে তাহলে তারাও মরল! যেমন aba হলে আর আমাদের forbidden pair যদি ab হয় তাহলে আমরা যদি প্রথম a মুছি তাহলে কোনো লাভ হচ্ছে না, আরও একটি character মুছতে হবে। কিন্তু যদি আমরা b মুছি তাহলে পরের forbidden pair ও ভেঙ্গে যাচ্ছে। সুতরাং আমাদের পরের character মুছা লাভ জনক।

তবে তুমি যদি আরও একটু চিন্তা কর তাহলে দেখবে abbbb মার্কা স্ট্রিং এর ক্ষেত্রে এই সমাধান খাটে না। কারণ এখানে a মুছা লাভ জনক। এজন্য greedy সমস্যার ক্ষেত্রে খুবই সাবধান হতে হয়। কোন দিক দিয়ে কোন anti case উদয় হয় ঠিক নাই। প্রথমে যেই সমাধান মনে হবে একদম সঠিক পরক্ষনেই মনে হতে পারে এই সমাধান তো একদমই ভুল!

যাই হোক, তাহলে আমরা আরও চিন্তা করতে থাকি, কীভাবে এই সমস্যা সমাধান করা যায়। এই abbbb এর উদাহরন ভাল করে দেখ। এই গ্রুপ থেকে আমরা যেই character কম আছে তাকে মুছে ফেলব। এখন এই আইডিয়া কাজে লাগিয়ে কীভাবে পুরো সমস্যা সমাধান করা যায় চিন্তা করা যাক। আমরা প্রত্যেকটি forbidden pair নেব। ধরা যাক বর্তমান forbidden pair হল ab. আমরা জানি

যে a বা b আর কোনো pair এ নেই। সুতরাং আমাদেরকে a বা/এবং b মুছতে হবে যেন পাশাপাশি কোনো ab বা ba না থাকে। খেয়াল কর আমাদেরকে কিছু a আর কিছু b মুছতে হতে পারে, এরকম হবে না যে সব a বা সব b মুছবা। যেমন abbbbcaaaab. এক্ষেত্রে প্রথম a আর শেষ b মুছা লাভ জনক। তাহলে সমাধান কি বুঝতে পারছ? আমরা যখন ab কে নিয়ে কাজ করব তখন দেখব আমাদের ইনপুটে কোথায় কোথায় a বা b পাশাপাশি আছে। তাদেরকে গ্রুপ গ্রুপ করে নেব। এবার একই গ্রুপে দেখব কতগুলি a আছে আর কতগুলি b আছে। যেটি কম থাকবে আমরা সেই গ্রুপ থেকে তাকে মুছব। যেমন এই কিছুক্ষণ আগের উদাহরনে দুটি গ্রপ আছে যা c দ্বারা আলাদা করা। প্রথম গ্রপে a কম আছে তাই সেখান থেকে a কে মুছব আর দ্বিতীয় গ্রুপে b কম আছে তাই তাকে মুছব।

Codeforces Round 103 Div 2 E Competition (144E)

সমস্যা: মনে কর একটি গ্রিডে কিছু খেলোয়াড় আছে যাদের coordinate (i, j) যেখানে $i \geq j$. যেসব ঘরের জন্য $i = j$ তাদের আমরা target cell বলব। যখনই খেলা শুরু হবে তখন খেলোয়াড়রা target cell এর দিকে দৌড়াবে। প্রতিটি খেলোয়াড় তার স্থান থেকে সবচেয়ে কাছের target cell এ যাবে। যদি এরকম একাধিক থাকে তাহলে তাদের যেকোনো একটিতে গেলেই হবে। এই যাওয়াটা অবশ্যই সবচেয়ে কম সময়ে হবে। অর্থাৎ মনে কর একজন খেলোয়াড় তিনটি target cell এ যেতে সময় নেয় যথাক্রমে 10, 12, 10. তাহলে সে যে দুটিতে যেতে 10 সময় লাগবে তাদের কোনো একটিতেই যাবে এবং 10 সময়েই যাবে। এই যাওয়ার পথ তুমি নির্ধারণ করে দিতে পার। অর্থাৎ কখন নিচে যাবে আর কখন বামে যাবে বা কখন কোন দিকে যাবে তা তুমি ঠিক করে দেবে। প্রতিটি খেলোয়াড় একক সময়ে এক ঘর ডানে, বামে, উপরে বা নিচে যেতে পারে। কিন্তু খেয়াল রাখতে হবে একই মুহূর্তে যেন দুজন খেলোয়াড় একই ঘরে না থাকে (সেটা target cell হোক আর যাই হোক)। তোমাকে বলতে হবে সবচেয়ে বেশি কত জন খেলোয়াড়কে তুমি একই সাথে খেলাতে পারবা যেন তারা তাদের সবচেয়ে কাছের target cell এ দৌড়ে যাবে (কীভাবে দৌড়ে যাবে তা তুমি ঠিক করে দেবে), কিন্তু কোনো দুজন একই সময়ে একই ঘরে না থাকে। গ্রিডের সাইজ 10^5 এর মত হতে পারবে। আর কোন খেলোয়াড় কোন ঘর থেকে খেলা শুরু করে তাও বলা আছে।

সমাধান: প্রথমত খেয়াল কর যেকোনো এক জন খেলোয়াড় এর জন্য তার candidate target cell কিন্তু পাশাপাশি। অর্থাৎ প্রতিটি খেলোয়াড় এর জন্য তার target cell সমুহ একটি রেঞ্জ। যেমন মনে কর একজন খেলোয়াড় $(10, 5)$ এ আছে। তার জন্য target cell সমুহ হবে $(10, 10), (9, 9) \ldots (5, 5)$ কারণ এদের প্রতিটিতে যেতে তোমার 5 একক করে সময় লাগবে। এভাবে আমরা প্রতিটি খেলোয়াড়ের জন্য একটি করে রেঞ্জ পাব। আমাকে সর্বোচ্চ সংখ্যক রেঞ্জ নির্বাচন করতে হবে যেন সেসব রেঞ্জের প্রতিটি হতে আমরা একটি করে সংখ্যা নির্বাচন করতে পারি যাতে করে কোনো দুটি সংখ্যা একই না হয়।

খুব একটা কঠিন না। Codeforces 45 D Event Dates তেই আমরা এরকম সমস্যা সমাধান করেছি। আমরা একটি heap বা priority queue নেব। এবার আমরা বাম হতে ডানে যাব। দেখব বর্তমান স্থান হতে কোনো রেঞ্জ শুরু হয় কিনা। হলে সেই রেঞ্জকে আমরা heap এ push করব। এরপর heap এ দেখব heap এ কোনো রেঞ্জ আছে কিনা। থাকলে তাদের মাঝে কোন রেঞ্জের শেষ মাথা সবচেয়ে ছোট। তাকে বর্তমান সংখ্যা দিয়ে দেব। এভাবে করলেই হবে।

কেন এই কাজ করলে খেলোয়াড়দের রাস্তা ছেদ করবে না? উত্তর দেয়াটা কঠিন। প্রমাণ করা আরও কঠিন। এইসব ক্ষেত্রে কাগজে কলমে কিছু case analysis করে দেখে নিজেকে বিশ্বাস করাতে হয়!

Codeforces Round 116 Div 2 D Name (180D)

সমস্যা: তোমাকে দুইটি স্ট্রিং s এবং t দেওয়া আছে। তোমাকে s এর character গুলিকে permute করে এমন একটি স্ট্রিং বানাতে হবে যেটি lexicographically সবচেয়ে ছোট এবং t অপেক্ষা lexicographically বড়। স্ট্রিং সমুহের দৈর্ঘ্য সর্বোচ্চ 5000 হতে পারে।

সমাধান: সমস্যাটা বেশ সহজ। আমাদেরকে t অপেক্ষা বড় কিন্তু সবচেয়ে ছোট একটি স্ট্রিং বানাতে হবে। এরকম একটি স্ট্রিং এর বৈশিষ্ট্য কি? দুটি স্ট্রিং এর প্রথমে কিছু ঘর একই থাকবে এর পর s এর অক্ষরটি t এর অক্ষর হতে বড় হবে। এরপর s এর বাকিটুকু ছোট হতে বড়তে সর্টেড হবে। যেমন s = abac, t = abob এর ক্ষেত্রে আমরা শুরুর একই ঘর a রাখতে পারি। এর পর s এ c বসাব যেটি t এর b অপেক্ষা বড়। এরপর s এ b আর a বাকি থাকবে। এদেরকে সর্ট করলে ab দাঁড়ায়। সুতরাং আমাদের s হবে acab. এখন প্রশ্ন হলো কতদুর আমরা শুরুতে একই রাখব? যেহেতু স্ট্রিঙের দৈর্ঘ্য মাত্র 5000 সেহেতু আমরা প্রত্যেক স্থানে গিয়ে গিয়ে চেষ্টা করতে পারি। একটু চিন্তা করলে বুঝবে যে এই কমন অংশ যত বড় হবে তত ভাল। তাই আমরা চাইলে এই কমন অংশের লেংথ এর উপর বাইনারি সার্চ চালাতে পারি, এতে করে যদি আমাদের স্ট্রিং গুলির লেংথ আরও বেশি হত তাহলেও সমস্যা হত না। আসলে আরও চিন্তা করলে আমার বিশ্বাস বাইনারি সার্চ ছাড়াও সমাধান করা যাবে। যাই হোক শুরুর কমন অংশের লেংথ জেনে গেলে তুমি বাকি যেসব স্ট্রিং আছে তাদের থেকে এমন একটি character নেবে যেটি ঐ স্থানের t এর character হতে যেন বড় হয়। এরপর বাকিগুলিকে সর্টেড ভাবে বসিয়ে দেবে।

Codeforces ABBY Cup 2.0 A3 Educational Game (178A3)

সমস্যা: তোমাকে n সাইজের একটি অ্যারে দেওয়া আছে ($n \leq 10^5$). ধরা যাক অ্যারেটি হলো $a_1, a_2, \ldots a_n$. তুমি যতবার ইচ্ছা একটি অপারেশন চালাতে পার। একে (i, t) দ্বারা চিহ্নিত করা যাক। এদের লিমিট হলো $1 \leq i \leq n$ এবং $i + 2^t \leq n$. এই অপারেশনের ফলে a_i এর মান 1 কমবে এবং a_{i+2^t} এর মান 1 বাড়বে। যেমন $(1, 0, 1, 2)$ তে $(1, 1)$ অপারেশন চালালে $(0, 0, 2, 2)$ হবে। তোমার লক্ষ্য হবে প্রথম k ঘরকে শূন্য করে ফেলা। বলতে হবে প্রথম k ঘরকে শূন্য করতে সর্বনিম্ন কয়টি অপারেশন লাগবে। k এর মান 1 হতে $n - 1$ পর্যন্ত হতে পারে। অর্থাৎ এই $n - 1$ টি k এর মানের জন্য তোমাকে সমাধান করতে হবে। n এর মান সর্বোচ্চ 10^5 হতে পারে।

সমাধান: যেহেতু আমাদের সংখ্যা সবসময় বাম হতে ডানে যায় তাই আমরা প্রথমে a_1 কে শূন্য করব এর পর a_2 কে- এরকম করে করতে থাকব। যখন আমরা a_1 কে শূন্য করব তখন আমার a_1 এর সংখ্যাগুলিকে কোথায় পাঠাবো? খুব সহজ উত্তর- সবচেয়ে দূরে। কিন্তু সবচেয়ে দূরে পাঠানোই কি সবচেয়ে লাভজনক? অবশ্যই! মনে কর a_1 কে তুমি 2^{t_1} ঘর সরাও এরপর 2^{t_2} ঘর সরাও। যদি $t_1 = t_2$ হয় তাহলে কিন্তু তুমি চাইলে এই দুটি move না দিয়ে একবারে 2^{t_1+1} move দিতে পারতে। আবার যদি $t_1 < t_2$ হলে তোমার জন্য t_2 এর move আগে দেওয়া ভাল হত, কারণ এতে করে সকল k এর জন্য সমাধানে এটি কাজে লাগবে। মানে k এর মান যাই হোক না কেন, তুমি যদি a_1 কে সবচেয়ে দূরে পাঠাও তাহলেই তোমার লাভ।

৭.১ অনুশীলনী

৭.১.১ সমস্যা

Simple

- Codeforces 67 A Partial Teacher
- Codeforces Round 66 D FreeDiv
- Codeforces Round 89 C Fancy Number
- Codeforces Round 87 Div 1 B Lawnmower
- Codeforces Round 98 B Permutation
- Codeforces 158 B Taxi
- Codeforces Round 111 Div 2 A Twins
- Codeforces Round 111 Div 2 B Unlucky Ticket
- Codeforces 159 B Matchmaker
- Codeforces 169 B Replacing Digits
- Codeforces Round 113 Div 2 C Median
- Codeforces 174 B File List
- Codeforces 176 A Trading Business
- Codeforces 119 Div 1 A Permutations
- Codeforces 142 Div 2 A Dragons

Easy

- Codeforces 76 B Mice
- Codeforces Round 67 D Big Maximum Sum
- Codeforces Round 71 C Beaver
- Codeforces 81 D Polycarp's Picture Gallery
- Codeforces 120 F Spiders
- Codeforces Round 93 Div 1 E-reader Display
- Codeforces 159 E Zebra Tower
- Codeforces 161 A Dress'em in Vests!
- Codeforces 161 B Discounts
- Codeforces Round 117 Div 2 C Optimal Sum
- Codeforces Round 129 Div 2 B Little Elephant and Sorting
- Codeforces Round 131 Div 2 B Hometask
- Codeforces Round 148 Div 1 C World Eater Brothers

Medium

- Codeforces 86 B Tetris Revisited
- Codeforces Round 92 Div 1 C Brackets
- Codeforces Round 96 Div 1 D Constants in the language of Shakespeare
- Codeforces Round 97 Div 1 C Zero One
- Codeforces Round 140 Div 1 B Naughty Stone Piles
- Codeforces Round 140 Div 1 D The table

Hard

- Codeforces 128 Div 2 E Transportation

অধ্যায় ৮

Dynamic Programming

UVa 437 The tower of babylon

সমস্যা: তোমার কাছে n ধরনের ব্লক আছে ($n \leq 30$)। প্রত্যেক ধরনের ব্লক অসংখ্য পরিমানে আছে। প্রতি ধরনের ব্লকের দৈর্ঘ্য, প্রস্থ ও উচ্চতা দেওয়া আছে। তোমাকে একটি ব্লকের উপর আরেকটি ব্লক এভাবে একটি ব্লকের টাওয়ার বানাতে হবে। শর্ত একটিই- তুমি কোনো একটি ব্লককে যখন আরেকটি ব্লকের উপরে রাখবে তখন যেন নিচের ব্লকের dimension উপরের ব্লকের dimension হতে বড় হয়। অর্থাৎ নিচের ব্লকের দৈর্ঘ্য ও প্রস্থ উপরের ব্লকের যথাক্রমে দৈর্ঘ্য ও প্রস্থ অপেক্ষা অবশ্যই বড় হতে হবে। তুমি চাইলে ব্লকগুলিকে এদিক ওদিক ঘুরিয়ে নিতে পার। বলতে হবে তুমি সবচেয়ে কত উঁচু টাওয়ার বানাতে পারবে।

সমাধান: সমস্যাটি আমার বেশ পছন্দ হয়েছে। একটু একটু coin change dp এর গন্ধ বের হয় আবার একটু একটু longest increasing subsequence এরও গন্ধ বের হয়। এখানে প্রথম সমস্যা মনে হয় "প্রত্যেক ধরনের (অসংখ্য) ব্লক দেওয়া আছে"। এতো সংখ্যক ব্লক নিয়ে কাজ করা তো মুশকিল। আমরা কি এই ব্লকের সংখ্যা কমিয়ে আনতে পারি কোনো ভাবে? খেয়াল কর যেহেতু নিচের ব্লককে উপরের ব্লক হতে বড় হতে হবে সেহেতু আমরা আসলে এক ধরনের ব্লক একবারই ব্যবহার করতে পারব। তবে হ্যাঁ হয়তো ঐ ব্লককে ঘুরিয়ে আবার বসাতে পারি। এই ঘুরানর ঝামেলা দূর করার জন্য আমরা যা করতে পারি তাহল প্রতিটি ব্লককে সব ভাবে ঘুরিয়ে আলাদা আলাদা ব্লক তৈরি করতে পারি। এরপর বলতে পারি এই হলো সব ব্লক, তুমি আর ব্লকগুলিকে ঘুরাতে পারবে না। তাহলেই তো হল তাই না? তাহলে আমাদের সমস্যা দাঁড়াল- কিছু ব্লক দেওয়া আছে। আগের মত শর্ত মেনে আমরা কত উচু টাওয়ার বানাতে পারব। আমরা একটি ব্লক এক বারই ব্যবহার করতে পারব আর কোনো ব্লককে ঘুরাতে পারব না।

এখন কীভাবে আগানো যায়? সমস্যায় দৈর্ঘ্য প্রস্থ কিরকম বড় হবে তাও বলা নাই! কিন্তু সেটা জানা কি জরুরি? এটা জানলেই তো হয় যে বর্তমান টাওয়ারের সবচেয়ে উপরের ব্লক কে? অমুক ব্লক সবার উপরে থাকলে এর উপরে আর কত বড় টাওয়ার বানানো যায়- এটাই হবে আমাদের DP. অর্থাৎ আমরা একটি ফাংশন লিখব DP নামে যেটি একটি প্যারামিটার নেয়, ধরা যাক i. আমাকে বলতে হবে i ব্লকের উপর সবচেয়ে বড় কত লম্বা টাওয়ার বানানো সম্ভব। সেটা কীভাবে সমাধান করব? খুব সহজ, আমরা সকল ব্লকের উপর লুপ চালিয়ে দেখব যে সেই ব্লককে i এর উপর বসান সম্ভব কিনা (ধরা যাক নতুন ব্লকটি হল k) যদি বসান যায় তাহলে আমরা $DP(k) + height[k]$ উচ্চতার একটি টাওয়ার বানাতে পারি। এরকম করে সকল k এর জন্য চেষ্টা করে সবচেয়ে বেশি মানটি নিলেই হবে। যদি আর কোনো ব্লক বসান সম্ভব না হয় তাহলে 0 রিটার্ন করে দিবা। আর শুরুতে কীভাবে DP কল করবা? চাইলে প্রতিটি ব্লককে নিচে বসিয়ে DP কল করতে পার। অথবা চাইলে বিশাল বড় দৈর্ঘ্য প্রস্থের কিন্তু উচ্চতা শূন্য এরকম একটি ব্লক আমদানি করে তাকে দিয়ে DP কল করতে পার। এরকম নতুন ব্লক আনলে সুবিধা হল এর উপর অন্য যেকোনো ব্লককে রাখতে পারব, অন্য কোনো ব্লকের উপর একে রাখতে পারবা না আর এর উচ্চতা শূন্য, সুতরাং উত্তরে এটি কোনো প্রভাব রাখবে না।

এই সমাধানের time complexity কত? $\mathrm{O}(n^2)$ যেখানে n হল ব্লকের সংখ্যা। কারণ তোমার state হল $\mathrm{O}(n)$ আর প্রত্যেক state হতে তুমি $\mathrm{O}(n)$ এর একটি লুপ চালিয়ে নতুন ব্লক বসানোর চেষ্টা করেছ।

UVa 1347 Tour

সমস্যা: একটি 2d plane এ n টি বিন্দু দেওয়া আছে (যদিও বলা নেই কিন্তু তোমরা ধরে নিতে পার $n \leq 1000$). কোনো দুটি বিন্দু একই x এ নেই। তোমাকে একদম বামের বিন্দু হতে শুরু করতে হবে (অর্থাৎ যার x সবচেয়ে কম), সেখান থেকে তুমি ডানের কোনো বিন্দুতে যাও, সেখান থেকে তার ডানের এরকম করে করে একদম সবচেয়ে ডানের বিন্দুতে যাও। এরপর একই ভাবে সেখান থেকে তার বামের কোনো বিন্দুতে যাও, এর পর তার বামের এরকম করে সবচেয়ে বামের বিন্দুতে ফেরত আসো। এই যে বাম হতে ডানে গিয়ে আবার বামে ফিরে আসলে এই সময়ে তোমাকে n টি বিন্দুই ঘুরে আসতে হবে। বলতে হবে সবচেয়ে কম কত দুরত্ব অতিক্রম করে তুমি এই কাজ করতে পারবে।

সমাধান: এটিও আমার বেশ পছন্দের সমস্যা। এই সমস্যার একটি নাম আছে- bitonic tour. বলতে দ্বিধা নেই এই সমস্যা প্রথমে দেখলে মনে হতে পারে এটা কীভাবে dynamic programming! কারণ এখানে state কি হতে পারে এটা একটা বিশাল চিন্তার ব্যাপার। এখানে পথটা কি রকম একটু দেখ- একদম বাম মাথা হতে শুরু করে কিছু কিছু করে ডানে যাবা, এর পর একদম ডান মাথায় পউছালে আবার একটু একটু করে বামে এসে একদম বাম প্রান্তে চলে আসতে হবে। এবং শুধু তাই না, তোমাকে আবার সকল বিন্দুতেই যেতে হবে। কিন্তু state কি?

State বের করতে গেলে আমাদেরকে একটু চিন্তা পাল্টাতে হবে। DP করতে গেলে একটু systematic ভাবে আগাতে হয়। এই যে তুমি এক বিন্দু থেকে মাঝের কিছু বিন্দু বাদ দিয়ে ডানে আরেক বিন্দুতে লাফ দিচ্ছ এটিই হলো এখানে মূল সমস্যা। আমরা কি কোনো ভাবে কোনো বিন্দুকে skip না করে একে একে বিন্দু গুলিকে নিতে পারি? মনে হতে পারে- তা কীভাবে সম্ভব! কিন্তু আসলে সম্ভব, একটু চিন্তা কর।

এখানে সমাধানের মূল আইডিয়া হল- বাম থেকে ডানে গিয়ে ফিরে আবার বামে আসা- এভাবে চিন্তা না করে চিন্তা করতে হবে- বাম থেকে দুই পথে ডানে যাওয়া যেন সকল বিন্দুই কাভার হয়- তাহলেই কিন্তু হয়ে যায়। একটু চিন্তা করে দেখ। এখন কিন্তু সমস্যা অনেক অনেক সহজ হয়ে গিয়েছে!

তাহলে প্রশ্ন হলো state কি? কোনো এক মুহূর্তে state হবে বাম হতে ডানে যাওয়ার দুটি পথ এখন কই কই আছে। ধরা যাক আমরা বিন্দুগুলিকে বাম হতে ডান 1 হতে n এ নাম্বারিং করেছি। তাহলে state কে আমরা (i, j) দিয়ে প্রকাশ করতে পারি যার মানে হবে- প্রথম পথটি এখন i এ আছে আর দ্বিতীয় পথটি এখন j তে আছে। আর $DP(i, j)$ ফাংশনকে রিটার্ন করতে হবে বাকি পথের optimal দুরত্ব, অর্থাৎ i আর j হতে ডানে n এ যেতে হবে আর বাকি সব বিন্দুকে ব্যবহার করতে হবে এরকম পাথ দুটির সবচেয়ে কম দুরত্ব। তাহলে প্রশ্ন হলো $DP(i, j)$ থেকে তুমি কই যাবে? যদি $i > j$ হয় তাহলে তুমি এর পরে $i + 1$ এ যাবার চেষ্টা করবে আর যদি $j > i$ হয় তাহলে $j + 1$ এ। অর্থাৎ তুমি বাম হতে ডানে একে একে বিন্দু গুলিকে consider করবে। আমরা আমাদের সুবিধার জন্য ধরে নিতে পারি $k = max(i, j) + 1$ অর্থাৎ আমরা এর পরে k তে যাব। কিন্তু কই থেকে? i থেকে নাকি j থেকে? দুই ভাবেই চেষ্টা করতে হবে। অর্থাৎ যদি আমরা i থেকে k তে যাই তাহলে হবে $DP(k, j) + dist(i, k)$ আর যদি j থেকে যাই তাহলে হবে $DP(i, k) + dist(k, j)$ এদের মাঝে যেটি সবচেয়ে কম সেটিই $DP(i, j)$ এর মান তাই না? আর DP এর শুরু করবে কীভাবে? খুব সহজ $DP(1, 1)$ অর্থাৎ দুটি পথের প্রান্ত তখন 1 এ থাকবে। আর শেষ হবে কখন? যখন $k = n + 1$ হয়ে যাবে। তখন কিন্তু i আর j এর ভেতর যেই প্রান্ত এখনও n এ পৌছায় নাই তাকে n এ পউছায়ে দিতে হবে (নাহলে তো সাইকেল হবে না) এবং সেই মানকে রিটার্ন করতে হবে DP থেকে। অথবা যদি দেখ যে k এর মান n তাহলে i এবং j উভয় স্থান থেকেই n এর দুরত্ব যোগ করে রিটার্ন করবে।

এই সমাধানের time complexity হল $\mathrm{O}(n^2)$ কারণ তোমার state হল $\mathrm{O}(n^2)$ আর প্রত্যেক state হতে তুমি দুই ধরনের transition করছ।

UVa 116 Unidirectional TSP

সমস্যা: একটি গ্রিড আছে। গ্রিডের row সংখ্যা খুব জোড় 10 টি আর কলাম সংখ্যা খুব জোড় 100 টি। গ্রিডের প্রতিটি ঘরে একটি করে সংখ্যা আছে এবং এই সংখ্যাগুলি ধনাত্মক বা ঋণাত্মক হতে পারে (সংখ্যাগুলি 30 বিটে প্রকাশ করা সম্ভব)। তুমি গ্রিডের উপরের বাম কোনা হতে নিচের ডান কোনায় যাবে। যেসব ঘর দিয়ে যাবে তাদের সংখ্যার যোগফল তোমাকে সর্বনিম্ন করতে হবে। তবে এক ঘর থেকে আরেক ঘরে যাবার কিছু নিয়ম আছে। তুমি যদি (r, c) তে থেকে থাক তাহলে তুমি এই তিনটি ঘরের কোনো একটিতে যেতে পারবে- $(r-1, c+1), (r, c+1), (r+1, c+1)$. যদি $(r-1, c+1)$ গ্রিডের উপর দিয়ে বেরিয়ে যায় তাহলে সে নিচ দিয়ে আসবে অর্থাৎ যেহেতু উপরে $(r-1, c+1)$ বলে কিছু নেই সেহেতু সে নিচে চলে যাবে $(R, c+1)$ যেখানে R হল শেষ row. একই ভাবে যদি তুমি নিচে যাবার সময় দেখ যে $(r+1, c+1)$ বলে কিছু নেই তাহলে তুমি উপরে চলে যাবে অর্থাৎ $(1, c+1)$ এ চলে যাবে। এক কথায় পুরো গ্রিডটি একটি সিলিন্ডারের মত (সিলিন্ডারের উচ্চতা x axis বরাবর কল্পনা কর)।

সমাধান: মনে কর একটি ফাংশন আছে DP যা একটি গ্রিডের সেল (r, c) কে প্যারামিটার হিসাবে নেয় এবং রিটার্ন করে সেখান থেকে নিচের ডানের কোনায় যাবার সবচেয়ে কম খরচের রাস্তা। এখানে একটি জিনিস, কোনো কোনো সেল হতে কিন্তু নিচের ডান কোনায় পউছান সম্ভব না। সেক্ষেত্রে মনে কর আমাদের ফাংশন অনেক অনেক বড় মান রিটার্ন করে। এই মানকে এতো বড় হতে হবে যেন কোনো পাথের খরচ যেন এতো বড় হতে না পারে। এই সমস্যার ক্ষেত্রে যেহেতু 100 টি কলাম থাকতে পারে তার মানে আমাদেরকে 100 টি 30 বিটের সংখ্যা পার হয়ে যেতে হতে পারে। সুতরাং আমাদের বড় সংখ্যাটি এর থেকে বড় হলেই হবে। অনেক সময় দেখা যায় যে এই বড় সংখ্যা বের করা বেশ কষ্টকর। সেসব ক্ষেত্রে খরচের পাশাপাশি আরও একটি সংখ্যা রাখতে পার যা বলবে সমাধান আছে কিনা। যদি সমাধান থাকে তাহলে অপর সংখ্যা দেখে বুঝতে পারবে যে খরচ কত।

যাই হোক, মূল সমস্যায় ফেরত আসি। তাহলে তুমি DP ফাংশনের ভেতর থেকে $(r-1, c+1), (r, c+1), (r+1, c+1)$ এই তিন স্থানের জন্য DP কল করবে (বা দরকারে গ্রিড wrap করবে আর কি)। এদের ভেতর থেকে যেটি সবচেয়ে কম তার সাথে তোমার স্থানের সংখ্যা যোগ করে রিটার্ন করবে তাহলেই হয়ে যাবে।

এই সমাধানের time complexity হবে $O(RC)$ যেখানে R হল row সংখ্যা আর C হল কলাম সংখ্যা। কারণ আমাদের $O(RC)$ সংখ্যক state আছে আর প্রতিটি state হতে 3টি transition করতে হয়। সুতরাং আমাদের complexity $O(RC)$.

মূল সমস্যায় অবশ্য আমাদের কোন পথে যেতে হবে সেটাও প্রিন্ট করতে বলেছে। যদি এরকম একাধিক পথ থাকে তাহলে lexicographically ছোটটি প্রিন্ট করতে হবে বলেছে। কিন্তু সেটা তো খুব একটা কঠিন না। প্রতিটি DP এর সময় লিখে রাখবে যে সেই স্থানের optimal মান তুমি তিনটি transition এর মাঝে কোনটি থেকে পেয়েছ। তাহলেই তো হল তাই না? কারণ তাহলে তুমি শুরুর সেলে লিখে রেখেছ এর পরে কই যেতে হবে, সেই সেলে লিখা আছে এর পরে কই যেতে হবে এভাবে তুমি উপরের বাম কোনা হতে নিচের ডান কোনায় চলে আসতে পারবে। আর যেহেতু আমাদের lexicographically ছোট রাস্তা প্রিন্ট করতে হবে সেহেতু যখন আমরা দেখব তিনটি রাস্তার মাঝে একাধিক রাস্তা সবচেয়ে কম খরচ দেয় তখন আমরা lexicographically ছোট সেল বেছে নেব। অর্থাৎ সাধারন path printing এর মত করেই আমরা পাথ প্রিন্ট করব।

UVa 12563 Jin Ge Jin Qu hao

সমস্যা: তোমার কাছে n টি ($n \leq 50$) গান আছে এবং এই গানগুলি বাজতে কত সময় নেয় তাও বলা আছে। প্রতিটি গানের দৈর্ঘ্য সর্বোচ্চ 3 মিনিট বা 180 সেকেন্ড। এই n টির বাইরে একটি বিশেষ গান আছে যেটি ঠিক 678 সেকেন্ড নেয়। তোমার কাছে t সেকেন্ড সময় আছে ($t \leq 10^9$). প্রশ্ন হল এই সময়ের মাঝে তুমি সর্বোচ্চ কতটি গান বাজাতে পারবে? যদি একাধিক ভাবে তুমি এই কাজ করতে পার তাহলে তোমাকে গান শুনবার সময় কে সবচেয়ে বেশি করার চেষ্টা করতে হবে। এখন এখানে একটা ব্যাপার আছে আর তাহল তোমার সব গান যে t সময়ের ভেতরেই শেষ হতে হবে সেরকম

কোনো কথা নেই। শেষ গানকে অবশ্যই t সময়ের আগে শুরু হতে হবে আর যখন খুশি তখন শেষ হতে পারবে। অর্থাৎ মনে কর তুমি ঠিক $t-1$ সময়ে 100 সেকেন্ডের গান শুরু করলে তাহলে তুমি $t+99$ সময় পর্যন্ত গান শুনতে পারবে। এখন আর কিন্তু কোনো নতুন গান শুনতে পারবে না কারণ t সময় পার হয়ে গিয়েছে।

সমাধান: এখানে যদি শুধু বলত যে সর্বোচ্চ কতটি গান শুনতে পারবে তাহলে কিন্তু খুব সহজ হত। Greedy ভাবে সবচেয়ে ছোট গান নিতে থাকতে হত যতক্ষণ তোমার নেয়া গান গুলির মোট সময় t এর থেকে কম হত। কিন্তু এখানে গান সর্বোচ্চ করার পাশাপাশি মোট সময় কেও সর্বোচ্চ করতে বলা হয়েছে। এই condition এর কারনে কিন্তু তোমার greedy সমাধান কাজ করবে না। উদাহরন স্বরূপ মনে করতে পার t হল 6 আর তোমার কাছে তিনটি গান আছে 2, 5, 678 সেকেন্ডের। যদি greedy ভাবে কর তাহলে 2 আর 678 নেবে। কিন্তু optimal সমাধান হল 5 আর 678 নেয়া। তাহলে উপায় কি?

প্রথমে কিছু observation. আমাদের বাজান শেষ গান অবশ্যই বিশেষ গান হতে হবে। কেন? দুই ধরনের কেস হতে পারে- একটি হল আমরা আমাদের বিশেষ গানটা বাজাই নাই অথবা বাজিয়েছি কিন্তু শেষে না। প্রথম ক্ষেত্রে মনে কর আমরা সবার শেষে যেই গান বাজিয়েছি সেটি বাজতে সময় নিয়েছে x সময়। আমরা জানি $x \leq 180$. আমরা যদি এই গানের পরিবর্তে বিশেষ গান বাজাতাম তাহলে অবশ্যই সেই গান অনেক বেশি সময় ধরে বাজত। এখন দ্বিতীয় কেস দেখা যাক। মনে কর এই বিশেষ গানটা আমরা বাজিয়েছি কিন্তু সবার শেষে না। এই ক্ষেত্রে আমরা কিন্তু যেই গান সবার শেষে বাজিয়েছি সেটা বিশেষ গান যেই মুহূর্ত হতে বাজান শুরু করেছিলাম সেই মুহূর্ত হতে এই গানটা বাজাব, এরপর বাকি গানগুলি বাজান শেষ করে আমাদের বিশেষ গানটা বাজাব। এতে করে কিন্তু বিশেষ গান বাজাতে কোনো সমস্যা হবে না আবার বিশেষ গানটাও সবার শেষে বাজল। অর্থাৎ আমরা আমাদের বিশেষ গানকে সবসময় শেষে বাজাব।

এর মানে হল আমরা বাকি n টি গান হতে সবচেয়ে বেশি কতগুলি গান বাজাতে পারি যেন তাদের মোট সময় t অপেক্ষা কম হয়। যদি একাধিক ভাবে এই সর্বোচ্চ সংখ্যক গান আমরা নির্বাচন করতে পারি তাহলে আমরা চেষ্টা করব সবচেয়ে বেশি সময় যাতে গান বাজে। এরপর আমরা আমাদের বিশেষ গান বাজাব।

এখন এই সমস্যা সমাধান করা কিন্তু খুব একটা কঠিন না। নানা ভাবে তুমি এখন DP করতে পার। DP সমাধান আলোচনা করা আগে একটা জিনিস, তাহল যদিও t এর মান অনেক হতে পারে কিন্তু $50 \times 180 + 678$ কিন্তু খুব একটা বড় না। যদি ইনপুটের t এর মান এর থেকেও বড় হয় তার মানে হবে আমরা সব গান বাজাতে পারব। সুতরাং আমরা ধরে নিতে পারি আমাদের t এর মান এর থেকে ছোট।

এখন আমরা DP কীভাবে করব সেটা নিয়ে ভাববো। প্রথমেই আমাদেরকে state বের করতে হবে। আমরা আসলে কি চাই? আমরা জানতে চাই যে i টি গান ব্যবহার করে j সময় ধরে গান বাজাতে পারি কিনা। কিন্তু কোন i টি গান? একটি উপায় হতে পারে বিটমাস্ক DP, অর্থাৎ n টি গানের কোন কোনটি ব্যবহার হয়েছে তা বিটমাস্কের মাধ্যমে প্রকাশ করা। কিন্তু সেক্ষেত্রে সমস্যা হল n এর মান অনেক বড়, তাই 2^n ও অনেক বড় হবে। সুতরাং এভাবে কোন i টি গান আমরা নিয়েছি তা প্রকাশ করা যাবে না। এখানে একটা জিনিস খেয়াল কর, আমাদের কিন্তু এটা জানার দরকার নেই কোন কোন i টি গান আমরা ব্যবহার করেছি। আমরা এভাবেও বলতে পারি যে, "প্রথম k টার ভেতরে i টা" আমরা ব্যবহার করেছি। সুতরাং আমাদের state হবে (k, i, j) যার মানে হবে- প্রথম k টি গানের মাঝে i টি গান ব্যবহার করে আমরা j সময় গান বাজাতে পারি কি না। তুমি চাইলে একে BFS বা DFS এর মতও চিন্তা করতে পার। যেহেতু এটা DP এর অধ্যায় আমরা DP এর মত করে চিন্তা করি। আমরা মুলত প্রতিটা $j < t, i \leq n$ এর জন্য জানতে চাচ্ছি (n, i, j) কি সম্ভব কিনা। মনে কর আমরা কোনো একটি (k, i, j) তে আছি। এখান থেকে আমরা কি করতে পারি? আমরা k তম গান নিতে পারি, সেক্ষেত্রে আমাদের জানা দরকার যে $(k-1, i-1, j-song[k])$ কি সম্ভব কিনা। অথবা আমরা k তম গান নাও নিতে পারি, সেক্ষেত্রে আমাদের জানা দরকার $(k-1, i, j)$ কি সম্ভব কিনা। সবশেষে আমাদের base case হবে যখন $k = 0$. যদি এই ক্ষেত্রে $i = j = 0$ হয় তার মানে সম্ভব, আর যদি তা না হয়, তার মানে সম্ভব না। সুতরাং আমাদের এই সমাধানে আমাদের state হল n^2t আর আমাদের DP এর ভেতরে মাত্র দুটি transition. তাই আমাদের time complexity হবে $O(n^2t)$.

তোমরা চাইলে এরকম state করতে পারতে- (k, j) যার মানে হবে, প্রথম k টি গান হতে

সর্বোচ্চ কতগুলি বাজিয়ে আমরা j সময় কাভার করতে পারি। তাহলে আমাদের time complexity হতে একটি n কমে যাবে।

UVa 11400 Lighting System Design

সমস্যা: মনে কর তোমার কাছে n ধরনের বাল্ব আছে ($n \leq 1000$). i তম ধরনের বাল্বের voltage rating V_i, সেই রেটিং এর একটি voltage source এর দাম K_i, সেই রেটিং এর একটি বাল্বের দাম C_i আর আমাদের এই রেটিং এর L_i টি বাল্ব কিনতে হবে। আমরা চাইলে কোনো একটি বাল্বের পরিবর্তে তার থেকে বেশি voltage rating ওয়ালা বাল্ব কিনতে পারি। যদি আমরা বেশি voltage rating ওয়ালা বাল্ব ব্যবহার করি তাহলে যেই voltage rating এর বাল্ব আমরা ব্যবহার করছি সেই rating এর voltage source আমাদের ব্যবহার করতে হবে। একটি voltage source একাধিক বাল্বকে জ্বালাতে পারে। বলতে হবে সবচেয়ে কম কত খরচে সব বাল্বকে জ্বালান সম্ভব।

সমাধান: এখানে মূল সমস্যা হল "কোনো একটি বাল্ব এর পরিবর্তে তার চাইতে বেশি voltage rating ওয়ালা বাল্ব কিনতে পারি"। এর মানে আমরা যদি বাল্বগুলিকে তাদের voltage rating অনুসারে সর্ট করি তাহলে এই কথাটা একটু সহজে বলা সম্ভব। ধরা যাক আমরা বাল্বগুলিকে ছোট হতে বড় voltage rating এ সর্ট করেছি। এর মানে দাঁড়ায় i হতে j তম ($i \leq j$) voltage rating এর বাল্বগুলিকে আমরা j তম বাল্ব দ্বারা replace করতে পারি। এখন বলতে হবে সবচেয়ে কম কত খরচে এই কাজ করতে পারব। এখানে কিন্তু খরচের নানা component আছে। যেমন আমরা যদি j তম বাল্ব দিয়ে i হতে j পর্যন্ত সকল বাল্ব replace করি তাহলে আমাদেরকে j তম voltage source কিনতে হবে। এরপর i হতে j তম পর্যন্ত মোট যতগুলি বাল্ব আছে ততগুলি j তম বাল্ব কিনতে হবে। এখন একটা বিষয়- এমন কি হতে পারে যে i হতে j এর মাঝের কিছু বাল্বের জন্য আমরা j তম বাল্ব দিয়ে replace না করে k তম বাল্ব ($k > j$) দিয়ে replace করা optimal হবে (আর বাকি গুলি j তম দিয়ে)? না এরকম হবে না। হয় তোমাকে i হতে j সকলকে j দিয়ে replace করতে হবে অথবা সকলকে k দিয়ে। মাঝের কিছু k দিয়ে করতে হবে এরকম হবে না। কেন? এভাবে চিন্তা কর k দিয়ে replace করা কখন লাভ জনক? যদি দেখ k তম বাল্বের দাম খুব কম হয়। তাহলে সেক্ষেত্রে শুধু মাত্র i হতে j এর মাঝের "কিছু" কেন? সব বাল্বকেই তো আমরা k তম দিয়ে replace করতে পারি তাই না? আর যদি k তম বাল্বের দাম বেশি হয় তাহলে কেন শুধু শুধু আমরা ঐ ধরনের বাল্ব কিনতে যাব বল?

তাহলে এখন চিন্তা কর এই সমস্যার state কি হবে? State হবে- আমরা i তম পর্যন্ত সকল বাল্বকে replace করে ফেলেছি, এখন বল বাকিটুকু replace করতে আমার কত খরচ হবে। এখানে i তম পর্যন্ত replace করে ফেলেছি মানে হলো আমাদের কাছে i তম voltage source আছে আর আমি হয়তো i এর আগে কিছু কিছু বাল্বকে i তম টাইপের বাল্ব দিয়ে replace করেছি। এটাই হল DP(i). এখন এর পর কোন DP তে যাব? ধরা যাক j তে যাব যেখানে $j > i$. এর মানে কি? এর মানে হলো আমরা $i + 1$ তম হতে j পর্যন্ত সকল বাল্বকে j তম বাল্ব দিয়ে replace করব, সেই সাথে আমাদের j তম voltage source কিনতে হবে। এটা গেল মূল খরচ। সেই সাথে j এর পরের অংশের জন্য আমাদের খরচ হবে DP(j). i হতে বড় সকল j এর জন্য এই cost সমুহ বের করে এদের মাঝে সবচেয়ে ছোটটিই হবে DP(i) এর মান। আর base case তো সহজ। যদি আমরা DP(n) এ চলে আসি এর মানে হবে আমাদের নেওয়া শেষ, সুতরাং আমাদের বাকিটুকু নিতে cost হবে 0.

যেহেতু আমাদের state সংখ্যা n টি আর প্রতি state এ n এর লুপ চালাতে হয় তাই আমাদের time complexity $\mathrm{O}(n^2)$.

UVa 11584 Partitioning by Palindromes

সমস্যা: সর্বোচ্চ 1000 দৈর্ঘ্যের একটি স্ট্রিং দেওয়া থাকবে। তোমাকে বলতে হবে এই স্ট্রিংটি সর্বনিম্ন কয়টি ভাগে ভাগ করা যায় যেন প্রতিটি ভাগ একটি palindrome হয়। যেমন aaadbccb এখানে একে তিন ভাগে ভাগ করা যায় যার প্রতিটিই palindrome (aaa)(d)(bccb). একে এর কম palindrome এ ভাগ করা যায় না।

সমাধান: আগের মত করেই চিন্তা কর। আমাদের DP(i) এর মানে মনে কর- "আমরা ইতিমধ্যেই শুরু হতে i এর আগ পর্যন্ত সব ভাগ করে ফেলেছি, এখন বল i হতে শেষ পর্যন্ত সর্বনিম্ন কত ভাগে ভাগ করা যায়"? এই মান কীভাবে বের করব? সহজ! j এর একটি লুপ চালাও আর দেখ i হতে j পর্যন্ত স্ট্রিংটি palindrome কিনা যদি palindrome হয় তাহলে 1+DP(j+1) হবে একটি candidate. এরকম সকল j এর মাঝে যেটি সবচেয়ে কম সেটিই উত্তর। i এ এসে প্রতিটি j এর জন্য i থেকে j পর্যন্ত স্ট্রিঙটি palindrome কি না এটা বের করা মনে হতে পারে অনেক costly. কারণ প্রতিটি i এ এসে তুমি j এর একটি লুপ চালাবে i হতে n পর্যন্ত। এই লুপের ভেতরে তুমি চেক করবে i হতে j একটি palindrome কিনা। কোনো একটি স্ট্রিং palindrome কিনা তাতো আশা করি $\mathrm{O}(n)$ এই করতে পারবা। তাহলে এখানে DP(i) এর ভেতর সময় লাগছে $\mathrm{O}(n^2)$ (j এর লুপের ভেতর $\mathrm{O}(n)$ এ palindrome বের করা)। আর যেহেতু মোট n টি state সেহেতু এই সমাধানে মোট সময় লাগবে $\mathrm{O}(n^3)$.

এখন উপায় কি? এভাবে ভাব- আমাদের সমাধানে costly অংশ কোনটা? অবশ্যই palindrome বের করার অংশ। আমরা কি কোনো ভাবে একটি সাবস্ট্রিং palindrome কিনা তা আরও দ্রুত বের করতে পারি? একটু যদি খেয়াল কর তাহলে বুঝবে এটিও একটি dp আসলে। তুমি জানতে চাচ্ছ i হতে j সাবস্ট্রিংটি palindrome কিনা। সেটা কীভাবে বের হবে? যদি i আর j একই হয় তাহলে তো হয়েই গেল। আর যদি নাহয় দেখবা i আর j তম character একই কিনা। যদি না হয় তাহলে তো palindrome না আর যদি দেখ একই তাহলে দেখ $i+1$ হতে $j-1$ কি palindrome কিনা। এর মানে আমাদের এই palindrome এর dp এর state $\mathrm{O}(n^2)$ আর ভেতরে মাত্র একটা transition. অর্থাৎ মোট time complexity হবে $\mathrm{O}(n^2)$.

খেয়াল কর $n^2 \times n^2 = n^4$ না কিন্তু। এখানে মোট time complexity হচ্ছে $n^2 + n^2 = n^2$ (constant গুন হিসাবে থাকলে তার কোনো দাম নেই)। যদিও একটি dp এর ভেতর হতে আরেকটি dp কে কল করা হচ্ছে কিন্তু দেখ পরের dp এর state কিন্তু আগের dp এর উপর নির্ভর করে না। পরের dp এর state কিন্তু $\mathrm{O}(n^2)$ ই থাকবে যেখানে n হল স্ট্রিং এর দৈর্ঘ্য। তুমি এভাবে চিন্তা করতে পার- দ্বিতীয় dp এর সকল state এর মান বের করতে সময় লাগে $\mathrm{O}(n^2)$. এর পর তো প্রথম dp তে palindrome বের করতে $\mathrm{O}(1)$ সময় লাগবে তাই না? তাহলে প্রথম dp তে মোট সময় লাগবে $\mathrm{O}(n^2)$. এভাবে মোট $\mathrm{O}(n^2)$ সময়ে সব কিছুর মান বের হয়ে যাবে।

সব শেষে একটি কথা, তোমরা কি বুঝতে পারছ কেন এটা greedy ভাবে হবে না? অর্থাৎ আমরা প্রথমে সবচেয়ে বড় সাবস্ট্রিং নেব যেটি palindrome, এর পর আবার শুরু থেকে সবচেয়ে বড় সাবস্ট্রিং নেব। এরকম করে করলে কেন সবচেয়ে কম ভাবে ভাগ হবে না? চিন্তা করে দেখ তো anticase বের করতে পার কিনা। যেমন একটি anticase হতে পারে pppbbp. তুমি যদি greedy ভাবে ভাগ কর তাহলে হবে (ppp)(bb)(p) কিন্তু optimal হবে (pp)(pbbp).

UVa 1625 Color Length

সমস্যা: দুটি স্ট্রিং দেওয়া আছে যাদের দৈর্ঘ্য সর্বোচ্চ 5000 হতে পারে। প্রতিটি স্ট্রিংএ A হতে Z এই 26 ধরনের character থাকতে পারে। এখন তোমাকে এই দুইটি স্ট্রিং merge করতে হবে। Merge মানে দুটি স্ট্রিং কে চিরুনির মত করে merge করা। Formal ভাবে বলা খুব কঠিন এর থেকে কয়েকটা উদাহরন দেয়া যাক। যেমন ABC আর DE কে merge করে ABDCE, ABDEC, DEABC এরকম নানা স্ট্রিং বানানো যায়। কিন্তু BDEAC, BCDE ইত্যাদি বানানো যাবে না কিন্তু। এখন আমাদের লক্ষ্য কি? আমাদের লক্ষ্য হলো এমন ভাবে merge করা যাতে আমাদের score সবচেয়ে কম হয়। একটি স্ট্রিং এর score কত? এটি হল এতে থাকা সকল ভিন্ন ভিন্ন character এর score এর যোগফল। একটি character এর score কত? সেটি হল- এই character প্রথমে কই আছে, আর শেষে কই আছে এই দুটি index এর বিয়গফল। যেমন যদি আমাদের স্ট্রিঙটি হয় AABAB তাহলে প্রথম A আছে 1 এ (1-indexing ধরে) আর শেষ A আছে 4 এ। তাহলে A এর score হবে 3. একই ভাবে B এর score হবে 2. তাহলে মোট score হবে 5. আমাদের লক্ষ্য হবে এমন ভাবে merge করা যাতে score সবচেয়ে কম হয়।

সমাধান: আমি যখন এই সমস্যা প্রথম দেখলাম তখন আমার মাথা কীভাবে কাজ করেছিল তা বলি।

প্রথমে আমার মনে হয়েছে হয় এটি greedy নাহলে এটি dp. কিন্তু যেহেতু string এর দৈর্ঘ্য মাত্র 5000 করে তাহলে মনে হয় greedy হবে না। বা অন্য ভাবে বললে যেহেতু মান কম সেহেতু আমরা চাইলে dp চালাতেই পারি। দুইটি স্ট্রিং দেওয়া আছে, আমাদের dp এর state কি হবে? দুটি স্ট্রিং দেওয়া থাকলে সাধারনত state হয় (i, j) যেখানে i হল প্রথম স্ট্রিং এ একটি index আর j হল দ্বিতীয় স্ট্রিং এ। শুরু হতে i আর j পর্যন্ত প্রসেস করা শেষ। এখন বাকিটুকু প্রসেস করতে হবে। এখন এই state এর সাথে আমাদের score এর সম্পর্ক কি? এখন পর্যন্ত আমরা score নিয়ে একটুও চিন্তা করি নাই, সুতরাং এখন একটু চিন্তা ভাবনা করা যাক। আমরা যদি প্রতিটি character এর জন্য তার প্রথম index এর বিয়োগ করতে পারি আর শেষ index টা যোগ করতে পারি তাহলেই কিন্তু হয়ে যাবে। এখন চিন্তা কর আমরা যদি (i, j) থেকে $(i + 1, j)$ বা $(i, j + 1)$ এ যাই তাহলে এই যোগ বিয়োগের কাজ করতে পারবা কিনা? দেখা যাক! মনে কর আমরা (i, j) থেকে $(i + 1, j)$ তে যাব। তার মানে আমরা $S[i + 1]$ নেব (ধরি S হল প্রথম স্ট্রিং)। এটি আসলে merged স্ট্রিং এ কত তম index? i+j+1 তম তাই না? কারণ এর আগে আমরা $i+j$ টি character নিয়েছি, সুতরাং এটি হবে $i + j + 1$ তম। আর কি জানার দরকার? জানা দরকার যে, $S[i+1]$ এই character টি $S[1 \dots i]$ বা $T[1 \dots j]$ তে আছে কিনা। যদি এদের কোনোটিতেই না থেকে থাকে তার মানে এটি হবে এই character এর প্রথম occurance এবং সেক্ষেত্রে আমরা এই স্থান অর্থাৎ $i + j + 1$ বিয়োগ করব। আর যদি দেখি এটি শেষ character অর্থাৎ $S[i + 2 \dots]$ আর $T[j + 1 \dots]$ এ এই character নেই তাহলে এই স্থান যোগ করব। আর সেই সাথে যোগ করব $(i+1, j)$ state এর dp এর মানকে। একই ভাবে আমরা $(i, j + 1)$ এর জন্যও একই জিনিস হিসাব করব। যেহেতু আমাদেরকে score সবচেয়ে কম করতে হবে সুতরাং এই দুটি option এর মাঝে যেটি আমাদের কম score দেবে সেটিই হবে আমাদের $DP(i, j)$ এর মান।

একটি জিনিস বাকি থেকে গেছে আর তাহল অমুক character $S[1 \dots i]$ এর মাঝে আছে কিনা বা $S[i \dots]$ এর মাঝে আছে কিনা- এ জিনিস কীভাবে বের করবে? এটা নানা ভাবেই খুব দ্রুত উত্তর করা যায়। এর মাঝে সবচেয়ে বুদ্ধিমান উপায় মনে হয় প্রতি character এর জন্য বের করে রাখা যে সেই character টি S এ সবার প্রথম কোথায় আছে আর সবশেষে কোথায় আছে। তাহলে এই সবার শুরু আর শেষ স্থানের সাথে compare করে বলে ফেলতে পারবে সেই character টি $S[1 \dots i]$ বা $S[i \dots]$ এর মাঝে আছে কিনা। আর এই সমাধানের complexity তো বুঝতেই পারছ- $\mathrm{O}(n^2)$ যেখানে n হলো স্ট্রিং এর length. কারণ আমাদের state $\mathrm{O}(n^2)$ আর state এর ভেতরে মাত্র দুটি transition.

UVa 10003 Cutting Sticks

সমস্যা: তোমার কাছে একটি লাঠি আছে। লাঠির সর্বোচ্চ দৈর্ঘ্য 1000. এই লাঠিতে n টি মার্ক করা আছে ($n \leq 50$). এই মার্ক করা স্থান গুলি তোমাকে কাটতে হবে। একবার কাটার cost হল ঐ মুহূর্তে যেই লাঠি কাটতেছ সেই লাঠির দৈর্ঘ্য। তোমাকে সবচেয়ে কম খরচে সকল দাগ দেওয়া স্থান কাটতে হবে। যেমন মনে কর লাঠির দৈর্ঘ্য হল 10 আর তোমার মোট তিনটি জায়গায় দাগ আছে- 2, 4 আর 7. তুমি যদি প্রথমে 2, এরপর 4, এরপর 7 এ কাট তাহলে খরচ হবে $10 + 8 + 6 = 24$. কিন্তু তুমি যদি প্রথমে 4, এরপর 2 তারপর 7 এ কাট তাহলে খরচ হবে $10 + 4 + 6 = 20$.

সমাধান: বেশ শিক্ষণীয় সমস্যা। এই সমস্যাটিও দেখলে প্রথমে greedy এর গন্ধ পাওয়া যায়। কিন্তু আমার মনে হয় না এটা greedy ভাবে সমাধান হবে। আর যদি greedy নাহয় তাহলে এধরনের সমস্যা সাধারণত dp দিয়ে সমাধান হয়। কীভাবে বুঝবা? Experience, তোমার যত experience হবে তত intuition বাড়বে। এখন প্রশ্ন হল যদি dp দিয়েই সমাধান হয় তাহলে state কি হবে? একটু কাটার প্রসেসটা কল্পনা কর। লাঠির এক জায়গায় কাটবা, দুই টুকরা হয়ে যাবে। আবার আরেক জায়গায় কাটবা, যাকে কাটলা সেটি দুই টুকরা হয়ে যাবে। এভাবে। এখানে একটি জিনিস, এক টুকরা কে কেটে কুচি কুচি করার খরচের সাথে অন্য টুকরা কুচি কুচি করার খরচের কোনো সম্পর্ক নেই। তাহলে আমরা "টুকরা" কে state হিসাবে বিবেচনা করতে পারি। একটি টুকরাকে represent করা কিন্তু খুবই সহজ। শুধু দুটি মাথা রাখলেই হয়। যেমন আমি যদি 10 দৈর্ঘ্যের একটি লাঠির 2 আর 6 এ কাটি তাহলে এটি তিন টুকরা হয়ে যাবে। এদের state হল $(0, 2), (2, 6), (6, 10)$. যেহেতু আমাদের

লাঠির দৈর্ঘ্য 1000 সেহেতু আসলে 1000^2 এর বেশি state হবে না। কিন্তু তুমি যদি একটু চিন্তা কর তাহলে বুঝবে আসলে এতো state হবে না। মনে কর প্রশ্নের উদাহরনে 2, 4 আর 7 এ দাগ ছিল। সুতরাং তুমি কিন্তু কখনও 8 বা 5 ইত্যাদি স্থানে কাটবে না। অর্থাৎ আসলে n^2 এর বেশি state হবে না। একে represent করাও কিন্তু বেশ সহজ। তুমি শুধু মার্কগুলির index রাখবে তাহলেই হবে। মানে যেমন ধর এর আগের উদাহরনে যদি $mark[1] = 2$, $mark[2] = 4$, $mark[3] = 7$ হয় তাহলে $(2, 7)$ না বলে আমরা বলতে পারি $(1, 3)$ কারণ $mark[1] = 2$ আর $mark[3] = 7$. আমাদের কিন্তু 0 আর 10 (অর্থাৎ 0 আর I) ও লাগবে। এদেরকে আমরা দুটি অতিরিক্ত mark হিসাবে রাখতেই পারি। যাই হোক মূল কথা- আমরা $O(n^2)$ state এ একটি লাঠির টুকরা রাখতে পারি। আমাদের DP কে রিটার্ন করতে হবে এই টুকরা এর ভেতরে যত মার্ক আছে সেসবের সবগুলিকে কাটতে হলে সর্বনিম্ন কত খরচ হবে। এটা কিন্তু খুব একটা কঠিন না। তোমার এই টুকরা এর ভেতরে যতগুলি মার্ক আছে, তাদের একে একে কাটার চেষ্টা করতে হবে। অর্থাৎ তুমি যদি $DP(i, j)$ তে থাক তাহলে সকল $i < k < j$ এর জন্য তোমাকে $DP(i, k) + DP(k, j) + length(i, j)$ এর সর্বনিম্ন মান বের করতে হবে। একে রিটার্ন করলেই হবে। এর মানে আমরা DP ফাংশনের ভেতরে একটি লুপ চালাচ্ছি। সুতরাং আমাদের time complexity হবে $O(n^3)$.

UVa 1626 Brackets sequence

সমস্যা: তোমাকে একটি bracket sequence দেওয়া আছে। এতে (,), [, আর] থাকতে পারে। এই sequence এর দৈর্ঘ্য সর্বোচ্চ 100 হতে পারে। তোমাকে এতে সবচেয়ে কম সংখ্যক character প্রবেশ করিয়ে balance করতে হবে। যদি একাধিক ভাবে করা যায় তাহলে যেকোনো একটি প্রিন্ট করলেই চলবে।

সমাধান: একটি স্ট্রিং এর উপর dp করার জন্য সবচেয়ে বহুল ব্যবহৃত state হল- এখন আমি কোন index এ আছি, অর্থাৎ state যদি i হয় এর মানে হবে শুরু হতে i পর্যন্ত প্রসেস করে ফেলেছি বাকিটুকুর উত্তর কত। কিন্তু এখানে এই state খাটবে না। কারণ ধর ([]) এই ক্ষেত্রে তো প্রথম opening bracket শেষ closing bracket এর সাথে match হয়। মনে কর এদের দু জনকে আমরা ম্যাচ করিয়েছি। এখন আমি যদি বলি 1 নম্বর পজিশন প্রসেস করেছি আমি বাকিটুকু প্রসেস করব তাহলে তো হবে না। কারণ তোমার তো 4 নম্বর পজিশনও প্রসেস হয়ে গিয়েছে তাই না? তাহলে কি বুঝতে পারছ এখানে state কি হবে? খুব একটা কঠিন না, একটি substring হবে এই সমস্যার state. অর্থাৎ $DP(i, j)$ হবে আমাদের ফাংশন যার কাজ হবে i হতে j index এর মাঝের string টুকুকে balance করতে সর্বনিম্ন extra কয়টি character লাগে তা প্রবেশ করিয়ে সেই string কে return করা। যেহেতু string এর দৈর্ঘ্য মাত্র 100 সুতরাং আমরা চাইলে আরও একটি লুপ চালাতে পারব এই ফাংশনের ভেতরে। সেই লুপ দিয়ে আমরা আমাদের substring এর প্রথম character এর জন্য matching bracket খুঁজব। যদি প্রথম character টি opening হয় তাহলে এই substring এর ভেতরে কোথাও তার matching closing bracket থাকতে পারে। ধরা যাক k তে আছে, তাহলে আমাদের একটি candidate হবে $S[i] + DP(i + 1, k - 1) + S[k] + DP(k + 1, j)$ তাই না (এখানে S হল ইনপুটের bracket sequence, আর আমরা মনে করছি DP আমাদেরকে ? কারণ আমি i কে k এর সাথে match করছি আর $[i + 1, k - 1]$ এই substring টুকুতে যত কম extra character প্রয়োজন তা প্রবেশ করাচ্ছি। একই ভাবে k এর পরের অংশতেও আমরা যত কম সম্ভব extra character প্রবেশ করাচ্ছি। এভাবে k এর লুপ চালিয়ে আমরা কিছু candidate পেলাম। আর কি হতে পারে? আমরা চাইলে $S[i]$ এর matching closing bracket কোথাও insert করতে পারি, ধরা যাক $S[k]$ এর পর insert করছি। সেক্ষেত্রে candidate হবে $S[i] + DP(i, k)+? + DP(k + 1, j)$ এখানে ? হল matching closing bracket. যদি $S[i]$ closing bracket হয় তাহলে তো সহজ, এর matching opening bracket টা আমরা শুরুতে বসাব আর এর পর বাকিটুকুকে দলাই মলাই করতে একটা DP কল করব। এখানে time complexity হবে $O(n^3)$.

তুমি চাইলে একটু চিন্তা ভাবনা করে এই সমাধানটা একটু সহজ করতে পার। যেমন দ্বিতীয় ক্ষেত্রে (যখন S[i] এর closing bracker আমরা প্রবেশ করিয়েছি) আমি বলেছি যে আমরা $S[i]$ এর matching closing bracket কোথায় বসাব তার উপর লুপ চালাব, কিন্তু একটু চিন্তা করলেই বুঝবে

যে আসলে লুপ চালানর কোনো দরকার নেই। আমরা যদি $S[i]$ এর পরেই সেই closing bracket বসিয়ে দেই তাহলে কিন্তু কোনো ক্ষতি নেই।

যাই হোক, এই সমস্যা কিন্তু চাইলে greedy ভাবেও সমাধান করতে পার। একটু চিন্তা করে দেখ- কোনো একটি bracket sequence এ কয়টি অতিরিক্ত character দিলে সেটি balance হবে সেটি তুমি greedy ভাবে বের করতে পার কিনা। খুব একটা কঠিন না কিন্তু! Stack দিয়েই এই কাজ করা যায়।

UVa 1331 Minimax Triangulation

সমস্যা: একটি m নোডের পলিগন আছে ($m \leq 50$)। তোমাকে কিছু chord বা কর্ণ টানতে হবে যাতে পুরো পলিগনটি অনেকগুলি triangle এ বিভক্ত হয়। একটু কাগজ কলম নিয়ে যদি চিন্তা ভাবনা কর তাহলে বুঝতে পারবে যে m নোডের যেকোনো পলিগণকে এরকম ত্রিভুজে বিভক্ত করতে ঠিক $m-3$ টি কর্ণের দরকার। কর্ণ মানে তো বুঝছ? একটি নোড হতে অন্য যেকোনো নোডে লাইন টানা (তার পাশের দুজন বাদে)। তোমার লক্ষ্য হলো এমন ভাবে বিভক্ত করা যাতে ত্রিভুজগুলির ক্ষেত্রফলগুলির মাঝে যেটি সবচেয়ে বড় সেটি যেন সবচেয়ে কম হয়।

সমাধান: এতক্ষণ আমরা খুব সাধারন dp দেখেছি। এই সমস্যাটি একটু "অসাধারন" কারণ এখানে একটু cyclic ব্যাপার স্যাপার আছে। অর্থাৎ এখানে state 1D বা 2D ম্যাট্রিক্স এর মত করে চিন্তা করলে হয় না। Cyclic ভাবে চিন্তা করতে হয়, তাই একটু ঝামেলা। যখনই দেখবা cycle তখন তুমি 0 indexing করবা। কারণ এতে একটি সুবিধা হচ্ছে i এর পরের নোডকে তুমি $(i+1)\%m$ আর আগের নোডকে $(i+m-1)\%m$ আকারে লিখতে পার। তুমি যদি এই ফর্মুলা না বুঝে থাক তাহলে একটু হাতে হাতে হিসাব করে দেখ, দেখবে 0 এর পর 1, এরপর 2 এভাবে $m-1$ এর পর আবার 0 পাবে। উল্টোদিকের ফর্মুলাও একই ভাবে কাজ করে। এই ফর্মুলা না ব্যবহার করলে বা 0 indexing ব্যবহার না করলে তোমার কোড আরও জটিল হবে অথবা প্রতিবার আগের/পরের নোড বের করতে গেলে তোমাকে if-else লিখতে হয় যা ঝামেলার।

এখন প্রশ্ন হল এই সমস্যার জন্য state কি বা কীভাবে সমাধান করতে হবে। যদি dp তে সামান্য পরিমাণও অভ্যস্ত হয়ে থাক তাহলে তুমি এটুকু বুঝবে যে state হল একটা polygon. সেখানে তুমি একটা কর্ণ টানবে, পলিগণকে দুই ভাগে ভাগ করবে, তাদের জন্য dp কল করবে, এরপর তাদের উত্তর merge করে তোমার উত্তর রিটার্ন করবে। উত্তর merge করা তো ব্যাপার না। দুই dp হতে দুই দিকের উত্তর আসবে (তাদের ত্রিভুজগুলির মাঝে সবচেয়ে বড়টির ক্ষেত্রফল), এবার এদের মাঝে যেটি বড় সেটি রিটার্ন করলেই হয়ে যাবে। এই dp এর base case ও সহজ। যখন দেখবে তোমার পলিগনটি একটি ত্রিভুজ তখন আর ভাগ করার দরকার নেই। সবই তো সহজ গোলাম হোসেইন কিন্তু state টা কীভাবে লিখব!

তুমি নিশ্চয় এতো বোকা না যে ভাববে "আরে পুরো পলিগন এর সব নোডের index পাঠিয়ে দিলেই তো হল"। কারণ সেক্ষেত্রে state অনেকগুলি (exponential) হয়। আমাদেরকে এই পলিগনগুলি একটু systematic ভাবে বানাতে হবে যেন তাদেরকে সহজে represent করা সম্ভব হয়। এতে কিছু লাভ আছে, এক কোড করা সহজ, দুই এবং সবচেয়ে গুরুত্বপূর্ণ তাদের state সংখ্যা exponential হবে না। এখন কথা হলে কীভাবে systematic ভাবে আমরা আগাতে পারি। তোমরা চাইলে vertex এর সাপেক্ষে চিন্তা করতে পার। আমার মনে হয়েছে vertex ধরে চিন্তা করলে $\mathrm{O}(n^4)$ এ সমাধান করা সম্ভব। কিন্তু তুমি যদি edge ধরে চিন্তা কর তাহলে বেশ সহজেই $\mathrm{O}(n^3)$ এ সমাধান করা সম্ভব। এই vertex ধরা বা edge ধরা কে আমি metaphorical অর্থে বলেছি। আশা করি একটু পরেই ব্যাপারটা পরিস্কার হবে। চল দেখি edge ধরে চিন্তা করা মানে কি।

মনে কর আমাদের কাছে একটি পলিগন এর চেইন আছে যার vertex গুলি হবে 0 হতে $n-1$ পর্যন্ত। আমরা $0-(n-1)$ এই edge নিয়ে কাজ শুরু করব। এখন এই edge টি অবশ্যই কোনো না কোনো ত্রিভুজের বাহু হবে। কিন্তু কোন ত্রিভুজের? বেশি চিন্তা না করে একটি লুপ চালিয়ে দাও তৃতীয় বিন্দুর জন্য 0 আর $n-1$ ছাড়া যেকেউ হতে পারে। ধরা যাক আমরা লুপ চালিয়ে i কে সেই তৃতীয় বিন্দু হিসাবে নির্বাচন করছি। তাহলে আমাদের ত্রিভুজটি হবে $0-i-(n-1)$. আর এর ফলে পলিগনটি একটি ত্রিভুজ এবং সর্বোচ্চ দুইটি চেইনে বিভক্ত হয়ে যাবে। চেইন দুটি হল $0-i$ আর

$i-(n-1)$. আশা করি এটা বুঝতে পারছ যে যদি $i=1$ হয় তাহলে $0-i$ বলে কোনো চেইন থাকবে না আর যদি $i=n-2$ হয় তাহলে $i-(n-1)$ বলেও কোনো চেইন থাকবে না। এখন এই চেইন দুটির জন্য recursively DP কল দিলেই হল। আর merge কীভাবে করবে তাতো আগেই আলোচনা করেছি। সুতরাং আমাদের state হবে (a,b) যার মানে হল a হতে b এর চেইন টুকু আর $a-b$ ইতমধ্যেই একটি বাহু দ্বারা জোড়া লাগান আছে। এই DP এর ভিতরে আমাদের কাজ হবে একটি i এর লুপ চালিয়ে $a-i-b$ ত্রিভুজ তৈরি করা আর $a-i$, $i-b$ এই চেইন দুটির জন্য DP কল করা।

UVa 1220 Party at Hali-Bula

সমস্যা: একটি n নোডের rooted tree দেওয়া আছে ($n \leq 200$). বলতে হবে তুমি সর্বোচ্চ কতগুলি নোড সিলেক্ট করতে পারবে যেন এদের মাঝে এমন দুটি নোড না থাকে যাদের একটি ওপরটির parent. আরও বলতে হবে সর্বোচ্চ নোডের সেট কি মাত্র এক রকমই হতে পারে নাকি একাধিক রকম হতে পারে।

সমাধান: ট্রিতে DP করতে হবে। আমরা root থেকে DP করা শুরু করব। DP এর state হবে আমরা এখন কোন নোডে আছি, ধরা যাক আমরা বর্তমানে u নোডে আছি। DP ফাংশন দুইটি মান return করবে। একটি হল এই নোডকে যদি নেই তাহলে এই নোড সহ পুরো subtree (মানে এই নোড এবং এর নিচের নোডগুলি) হতে সর্বোচ্চ কতগুলি নোড নির্বাচন করা যাবে (parent এর restriction মেনে)। আর আরেকটি মান হলো এই নোডকে না নিয়ে সর্বোচ্চ কতগুলি নোড নির্বাচন করা সম্ভব। তোমরা চাইলে দুইটি মান রিটার্ন না করে state এর প্যারামিটার আরও একটি বাড়ায়ে দিতে পার। 0 দিয়ে বুঝাবা যে u কে নিবা না আর 1 দিয়ে বুঝাবা যে u নিবা। যাই হোক। এখন কথা হল এই মান দুইটি কীভাবে বের করবা। ধরা যাক তুমি এখন $(u,0)$ state এ আছ অর্থাৎ u তে আছ আর একে ছাড়াই তুমি সর্বোচ্চ সংখ্যক নোড নিতে চাও। তাহলে তুমি যেটা করবে তাহল u এর প্রতিটি child c এর জন্য $(c,0)$ আর $(c,1)$ এর মাঝে বড় মানটি নেবে। কারণ যেহেতু u নিচ্ছ না সেহেতু c নিবে কিনা কিছুই যায় আসে না। তুমি c এর যে ক্ষেত্রে বেশি সংখ্যক নোড পাবে সেটিই নিবে। আর $(u,1)$ এর ক্ষেত্রে তোমার c নেবার কোনো সুযোগ নেই। সুতরাং সকল child c এর জন্য তোমাকে $(c,0)$ নিতে হবে। এভাবে আমরা সর্বোচ্চ নোডের সেটে কয়টা নোড আছে তা বের করতে পারব।

এখন প্রশ্ন হল এই সেট কতভাবে গঠন করা যাবে। এটাও খুব একটা কঠিন না। যদিও এই সমস্যায় জানতে চেয়েছে যে এক ভাবে না একাধিক ভাবে, কিন্তু যদি বলত মোট কতভাবে তৈরি করা যাবে তাহলেও করা সম্ভব। তুমি যখন $(u,0)$ বা $(u,1)$ এর মান বের করবে তখন সেটি কত ভাবে হয় তাও বের করবে। প্রথমে দেখা যাক $(u,1)$ কতভাবে হবে তা কীভাবে বের করবা। $(u,1)$ তো সকল $(c,0)$ এর যোগফল। সুতরাং $(c,0)$ গুলি যত ভাবে তৈরি করা সম্ভব তাদের গুনফল এর সমান ভাবে $(u,1)$ তৈরি করা যাবে। কারণ তুমি যদি এক সাবট্রি হতে 10 ভাবে $(c,0)$ পাও আর আরেক সাবট্রি হতে যদি 20 ভাবে পাও তাহলে তো মোট 200 ভাবে তুমি $(u,1)$ বানাতে পারবে তাই না? আর আশা করি $(u,0)$ এর ক্ষেত্রে কি হবে তা নিজেরাই করে নিতে পারবে। সূত্রটা যদিও $(u,1)$ এর মত সহজ না কিন্তু ওত কঠিনও না। আর আরেকটা কথা, এই সমস্যায় কিন্তু কত ভাবে বানাতে হবে তা চায় নাই, চেয়েছে এক না একাধিক। সুতরাং তুমি আসলে কয়েকটা if else লাগিয়ে হিসাব করতে পার। তুমি যদি পুরো উত্তর বের করতে চাও যে কতভাবে সেই সেট পাওয়া যায় তাহলে সেই সংখ্যা overflow করতে পারে।

আর আশা করি বুঝতে পারছ যে এখানে time complexity হবে $\mathrm{O}(n)$. অনেকে মনে করতে পার state তো $2*n$ আর এর ভেতরে এর child গুলির উপর দিয়ে লুপ চালান হচ্ছে। সর্বোচ্চ child সংখ্যা হতে পারে n ($n-1$ আর কি) সুতরাং মোট তো n^2 হওয়া উচিত। না এখানে লক্ষ্য কর যে যদি কোনো একটি নোডের child সংখ্যা বেশি হয় এর মানে অন্য নোডের child সংখ্যা কম হবে। তুমি এভাবেও চিন্তা করতে পার যে মোট কতবার DP কল হবে? DP কল হবে প্রতিটি নোড হতে তার child এর জন্য। এর মানে প্রতি parent-child সম্পর্কের জন্য। অর্থাৎ যতগুলি edge ততগুলি সম্পর্ক। মোট edge n টি (আসলে $n-1$). সুতরাং আমাদের DP কল হবে $2n$ বার। এটিই আমাদের time complexity. মোট কথা এই DP এর complexity হবে $\mathrm{O}(n)$.

UVa 1218 Perfect Service

সমস্যা: একটি n নোডের ট্রি দেওয়া থাকবে ($n \leq 10000$)। তোমাকে ট্রি এর নোড গুলিকে সাদা ও কালো রঙ করতে হবে যেন সকল সাদা নোডের ঠিক একটি কালো নোড neighbor হিসাবে থাকে। দুটি সাদা রঙের নোড neighbor হলে সমস্যা নেই। এখানে neighbor মানে হল adjacent. বলতে হবে সর্বনিম্ন কতগুলি নোডকে কালো রঙ করতে হবে।

সমাধান: আবারো ট্রিতে DP করতে হবে। তবে এবার একটু বেশি চিন্তা ভাবনা করতে হবে। যদিও এই সমস্যার ট্রিটি rooted না কিন্তু আমরা চাইলে যেকোনো একটি নোডকে root হিসাবে কল্পনা করতে পারি। এতে করে আমাদের dp চালান সহজ হবে।

মনে কর আমরা এখন একটি নোড u তে আছি। একে আমরা চাইলে সাদা রঙ করতে পারি আবার কালো করতে পারি। আমাদের বের করতে হবে এই দুই ক্ষেত্রে সর্বনিম্ন কতগুলি নোড কালো রঙ করলে পুরো সাবট্রি আমাদের শর্ত মানবে। এজন্য ভাবতে হবে u এর child সমুহ হতে আমাদের কি কি তথ্য জানা দরকার। এর উপর ভিত্তি করেই আমরা আমাদের state দাঁড় করাতে পারব। প্রথমেই ধরা যাক আমাদের নোড যদি কালো রঙ করা হয় তাহলে কি হবে। সেক্ষেত্রে এই নোডের যেকোনো neighbor কালো হতে পারবে। আবার সাদাও হতে পারবে তবে সেই ক্ষেত্রে সেই সাদা নোডের আর কোনো neighbor কালো নোড হিসাবে থাকতে পারবে না। এখানে একটা জিনিস লক্ষ্য করার বিষয়- সেটা হল কোনো একটি নোড সাদা হলেও তার দুই রকম অবস্থা থাকতে পারে। এক- তার নিচে কোনো কালো neighbor থাকা যাবে না, দুই- কালো neighbor থাকতে হবে। সুতরাং আমাদের প্রতিটি নোডের এই তিনটি state থাকবে- u নোড কালো, u নোড সাদা কিন্তু এর কোনো child কালো হতে পারবে না, u নোড সাদা এবং এর নিচে কালো একটি নোড থাকতেই হবে।

এখন একে একে দেখা যাক প্রতিটি ক্ষেত্রে আমাদের কি করতে হবে। মনে কর আমরা এখন যেই নোডে আছি তাকে কালো রঙ করতে চাই। সেক্ষেত্রে child নোড গুলিকে সাদা রঙ করতে হবে এবং তাদের নিচে কোনো কালো neighbor থাকা যাবে না।

যদি নিজেকে সাদা আর এর নিচে কোনো কালো না রাখতে চাই তাহলে কি করতে হবে? সেক্ষেত্রে নিচের সবগুলি সাদা হতে হবে। কি ধরনের সাদা? দ্বিতীয় ধরনের সাদা, অর্থাৎ তাদের অবশ্যই একটি কালো নোড neighbor হিসাবে থাকতে হবে। কেন? কারণ যদি তাদের নিচেও যদি কালো না থাকে তাহলে তো তাদের আর কোনো কালো neighbor থাকা সম্ভব না।

বাকি থাকল যদি আমরা নিজেকে সাদা করতে চাই আর তার নিচে যদি একটি কালো থাকতে হয়। সেই ক্ষেত্রে আমাদের নিচে যারা আছে তাদের একটিকে কালো করতে হবে আর বাকি গুলিকে সাদা করতে হবে। কোন ধরনের সাদা? যাদের অন্তত একটি নিচে কালো আছে, অর্থাৎ দ্বিতীয় ধরনের। তাহলে এই গেল আমাদের state আর state আপডেট করার উপায়।

এখন খেয়াল কর, প্রথম দুই আপডেটের ক্ষেত্রে কোনো লুপ নেই (child এর DP কল করার লুপ বাদে)। কিন্তু শেষ আপডেটের ক্ষেত্রে কিন্তু একটা লুপ লাগছে। সেই লুপ চালিয়ে আমরা কালো নোড fix করছি এর পর আরেকটি লুপ চালিয়ে আমরা বাকি নোডের জন্য DP কল করছি। সুতরাং আমাদের time complexity হয়ে যাচ্ছে $\mathrm{O}(n^2)$. যদি আমাদের এই কালো নোড fix করার লুপ (অর্থাৎ আমার কোন child কালো হবে) child এর DP কল করার লুপের সাথে nested ভাবে না থাকত তাহলে কিন্তু $\mathrm{O}(n)$ হত। আমরা কি কোনো ভাবে এই লুপ বাদ দিতে পারি বা nested না করে করতে পারি? হ্যাঁ পারি। এসব ক্ষেত্রে একটি কমন টেকনিক আছে। খেয়াল কর, আমরা এখানে কালো নোড fix করে বাকি নোড গুলির দ্বিতীয় ধরনের সাদার cost এর যোগফল বের করছি তাই না? আমরা এই যোগফল কালো নোডের লুপের ভেতরে বের না করে বাইরে বের করি। অর্থাৎ আমরা আমাদের নোডের নিচে যতগুলি child আছে সবার দ্বিতীয় ধরনের সাদার যোগফল বের করি, ধরা যাক S. এর পর কালো নোডের জন্য লুপ চালাই। ধরা যাক আমরা একটা নোড নির্বাচন করেছি যে একে আমরা কালো করব। তাহলে S হতে আমরা এই নোড দ্বিতীয় ধরনের সাদা করতে যেই cost লাগে তা বাদ দেই আর একে কালো করতে যেই cost তা যোগ করি। তাহলেই তো হয়ে গেল তাই না? আমাদের এখন আর লুপ দুটি আর nested নেই। সুতরাং এভাবে আমরা $\mathrm{O}(n)$ এ আমরা এই সমস্যা সমাধান করতে পারি। তোমরা চাইলে এই কাজ দুটি লুপের পরিবর্তে একটি লুপ দিয়েও করতে পার।

UVa 1252 Twenty Questions

সমস্যা: n টি ভিন্ন ভিন্ন সংখ্যা আছে ($n \leq 128$). তোমার বন্ধু এই n টি সংখ্যার মাঝে কোনো একটি মনে মনে ধরেছে আর তোমাকে বলেছে যে সে কোন সংখ্যাটি মনে মনে ধরেছে তা বের করতে। তুমি জান যে এই n টি সংখ্যা কি কি। প্রতিটি সংখ্যা m বিটের ($m \leq 11$). তুমি তোমার বন্ধুকে কিছু প্রশ্ন করতে পার যেমন- i তম বিট কি 0 নাকি 1. এরপর সেই উত্তরের উপর ভিত্তি করে তুমি আবার নতুন প্রশ্ন করতে পার। বলতে হবে সবচেয়ে কম কত গুলি প্রশ্ন করে তুমি বলতে পারবে তোমার বন্ধুর সংখ্যাটি কত। যেমন মনে কর যদি $n = 1$ হয় তাহলে কিন্তু তুমি কোনো প্রশ্ন ছাড়াই বলে দিতে পার তোমার বন্ধুর সংখ্যা কত। আবার $n = 2$ হয় আর সংখ্যা দুটি হয় 100 আর 110 তাহলে তুমি একটি প্রশ্ন করেই বুঝতে পারবে তোমার বন্ধু কোন সংখ্যাটি মনে মনে ধরেছে (মাঝের বিট অর্থাৎ 1 তম বিট কি 0 না 1).

সমাধান: অনেকেই হয়তো শুরুতে ভাবতে পারে আমাকে কিছু বিট এর পজিশন নির্বাচন করতে হবে, ধরা যাক $[i_0, i_1, \ldots]$ যেন সেসব বিট সম্পর্কে জিজ্ঞাসা করলে তুমি বুঝে ফেলতে পার যে কোন সংখ্যা তোমার বন্ধু মনে মনে ধরেছে। কীভাবে বুঝতে পারবে? মনে কর তুমি এই সব বিট সম্পর্কে জিজ্ঞাসা করলে আর তোমার জিজ্ঞাসার উত্তর হল $[b_0, b_1, \ldots]$. যদি দেখ যে n টি সংখ্যার প্রতিটির জন্য যদি এই উত্তরের লিস্ট আলাদা আলাদা হয় এর মানে এই উত্তর দেখেই তুমি বলে ফেলতে পারবে যে কোন সংখ্যাটি তোমার বন্ধু মনে মনে ধরেছে। যেমন উপরের উদাহরন 100, 110 এর ক্ষেত্রে তুমি যদি শুধু মাঝের বিট সম্পর্কে জিজ্ঞাসা কর তাহলে উত্তরের লিস্ট প্রথম সংখ্যার জন্য $[0]$ আর দ্বিতীয় সংখ্যার জন্য $[1]$. অর্থাৎ আলাদা। সুতরাং আমরা শুধু এই বিট নিয়ে প্রশ্ন করলেই বুঝে যাব কোন সংখ্যা তোমার বন্ধু মনে মনে ধরেছে। কিন্তু একটু চিন্তা করলে বুঝবে এখানে তুমি একটা বিশাল বড় জিনিস ignore করে গিয়েছ। আর তাহল তুমি কিন্তু চাইলে "প্রথম প্রশ্নের" উত্তর এর উপর ভিত্তি করে "দ্বিতীয় প্রশ্ন" করতে পারবে। অর্থাৎ আমি কিছুক্ষণ আগে যা বললাম তাতে কিন্তু তুমি কোন কোন বিট নিয়ে প্রশ্ন করবে তা কিন্তু আগেই নির্ধারণ করে ফেলেছ। Dynamically তুমি প্রশ্ন করছ না। হয়তো dynamically প্রশ্ন করলে তুমি কম প্রশ্নেই বের করে ফেলতে পারবে তোমার বন্ধু কোন সংখ্যা ধরেছে! একটু যদি চিন্তা কর তাহলেই তুমি এরকম একটি case বের করে ফেলতে পারবে। যেমন সংখ্যাগুলি যদি এমন হয়- 000, 001, 110, 100. একটা বিট জিজ্ঞাসা করে যে তুমি 4 টা সংখ্যা আলাদা করতে পারবা না এটা একটা বোকাও বুঝে। সুতরাং দেখা যাক আমরা এমন দুটা বিট পাই কিনা যেই দুটি বিট জিজ্ঞাসা করলে আমরা এই চারটি সংখ্যা আলাদা করতে পারব। ধরা যাক আমরা প্রথম (msb) আর মাঝের বিট নিয়ে জিজ্ঞাসা করব। তাহলে কিন্তু প্রথম দুটি সংখ্যার ক্ষেত্রেই উত্তরের লিস্ট হবে $[0, 0]$. যদি আমরা প্রথম আর শেষ বিট নিয়ে জিজ্ঞাসা করি তাহলে শেষ দুটা সংখ্যার ক্ষেত্রে উত্তরের লিস্ট পাব $[1, 0]$. আর যদি আমরা মাঝের আর শেষের (lsb) বিট নিয়ে জিজ্ঞাসা করি তাহলে প্রথম আর শেষ দুই সংখ্যার ক্ষেত্রেই পাব $[0, 0]$. এর মানে কোনোই লাভ হচ্ছে না। আমাদের তিনটি প্রশ্ন করতেই হচ্ছে। কিন্তু আমরা যদি প্রথম প্রশ্নের উত্তরের উপর ভিত্তি করে যদি দ্বিতীয় প্রশ্ন করি তাহলে কিন্তু আমরা মাত্র দুটি প্রশ্ন করেই বের করে ফেলতে পারব। ধরা যাক আমরা প্রথম বিট (msb) নিয়ে প্রশ্ন করলাম উত্তর 0 হতে পারে আবার 1 ও হতে পারে। যদি উত্তর 0 হয় এর মানে তোমার বন্ধু হয় 000 নাহয় 001 ধরেছে। আমরা যদি এই ক্ষেত্রে শেষের বিট নিয়ে জিজ্ঞাসা করি তাহলেই বুঝে যাব যে 000 নাকি 001 উত্তর। কিন্তু যদি আমাদের প্রথম প্রশ্নের উত্তর 1 হয় এর মানে তোমার বন্ধু হয় 110 নাহয় 100 ধরেছে। এই ক্ষেত্রে আমরা মাঝের বিট নিয়ে জিজ্ঞাসা করব। তাহলেই আমরা বুঝে যাব তোমার বন্ধু কি ধরেছে। এর মানে আসলে আমাদের এই সমস্যা এতো সহজে সমাধান হবে না। আমাদের কে একটি একটি করে প্রশ্ন করতে হবে আর সেই প্রশ্নের উত্তর কি তার উপর ভিত্তি করে পরের প্রশ্ন করতে হবে।

তাহলে এটি কীভাবে সমাধান করবা? একটা উপায় হল তুমি একটি bitmask এ লিখে রাখবা এখনও কোন কোন সংখ্যা valid, অর্থাৎ তোমার বন্ধু কোন কোন সংখ্যা মনে মনে ধরতে পারে। তুমি একটি করে প্রশ্ন করবা। আর সেই প্রশ্নের উপর ভিত্তি করে তোমার এই valid সংখ্যার সেট দুই ভাগে ভাগ হয়ে যাবে। এই দুই ভাগে আবার DP চালিয়ে দেখবা যে কোন ভাগে বেশি প্রশ্ন করতে হবে। এরকম প্রতিটি বিট নিয়ে নিয়ে try করলেই হবে। কিন্তু সমস্যা হল এই ক্ষেত্রে mask এর সাইজ 2^n যা অনেক বেশি। কিন্তু আমাদের m কিন্তু বেশ ছোট মাত্র 11. আমরা কি 2^m জাতীয় কিছু করতে পারি? আমরা 2^m এ চাইলে mark করতে পারি যে কোন কোন বিট সম্পর্কে আমরা জিজ্ঞাসা করব।

কিন্তু তাহলেই তো যথেষ্ট হবে না। কারণ এই সব বিটের প্রশ্নের উত্তর বিভিন্ন হতে পারে আর তার উপর ভিত্তি করে আমাদের eligible valid number এর সেট আলাদা আলাদা হতে পারে। তাহলে আমরা এক কাজ করতে পারি আর তাহল সেসব প্রশ্নের উত্তর কি কি তারও একটা mask রাখতে পারি (ধরা যাক কোন কোন প্রশ্ন করেছি সেটা হল mask1 আর উত্তর গুলি mask2, যেসব প্রশ্ন করা হয় নাই সেসবে 0 বা 1 যেকোনোটিই থাকতে পারে, আমরা আমাদের সুবিধার জন্য 0 রাখব)। এর মানে আমাদের state হবে $2^m \times 2^m$ অর্থাৎ আমরা কোন কোন বিট নিয়ে প্রশ্ন করেছি এবং সেসব প্রশ্নের উত্তর কি ছিল। প্রতিবার আমরা নতুন একটি বিট নির্বাচন করব আর সেই বিট নিয়ে জিজ্ঞাসা করব। উত্তর 0 বা 1 হতে পারে। প্রতিটির জন্য আমরা DP কল করব। যদি দেখি যে এই পর্যায়ে এসে mask1 প্রশ্ন গুলি করে উত্তর mask2 হয়েছে এরকম সংখ্যার পরিমান কেবল একটি এর মানে আর আমাদের প্রশ্ন করতে হবে না। এভাবে DP করলেই হবে। অর্থাৎ মনে কর আমরা এখন DP(mask1, mask2) তে এসেছি। প্রথমেই দেখতে হবে এখন valid উত্তরের সেটে কতগুলি সংখ্যা আছে (n এর একটি লুপ চালিয়ে চেক করব যে প্রতিটি সংখ্যাকে mask1 দিয়ে and করলে mask2 আসে কিনা)। যদি মাত্র একটি সংখ্যা থাকে এর মানে আমাদের আর প্রশ্ন করার দরকার নেই (0 রিটার্ন করে দাও)। আর যদি দেখ একাধিক আছে এর মানে আমাদের প্রশ্ন করতে হবে। কোন বিট সম্পর্কে প্রশ্ন করব তার একটি লুপ চালাব (i এর একটি লুপ 0 হতে m-1 পর্যন্ত)। এর উত্তর 0 হতে পারে বা 1 ও হতে পারে। 0 এর ক্ষেত্রে আমাদের কল করতে হবে DP(mask1 | (1<<i), mask2) আর 1 এর ক্ষেত্রে আমরা কল করব DP(mask1 | (1<<i), mask2 | (1<<i)). এর পর এদের ভেতরে যেটি বেশি তার সাথে এক যোগ করলে (এখন যে i তম বিট নিয়ে প্রশ্ন করলে তার জন্য 1) তুমি পাবে i তম বিট নিয়ে প্রশ্ন করলে এটি সহ আর সর্বোচ্চ কতটি প্রশ্ন করতে হয় তার পরিমান। এভাবে প্রতিটি i এর জন্য আমরা যেসব উত্তর পাব তাদের ভেতর হতে যার জন্য তুমি সবচেয়ে কম পাবে সেটিই হবে তোমার DP(mask1, mask2) এর উত্তর।

আপাত দৃষ্টিতে তোমার complexity $\mathrm{O}\big(2^m \times 2^m \times (n+m)\big)$ কারণ প্রতিটি state এর ভেতরে তোমাকে একটি n আর আরেকটি m এর লুপ চালাতে হচ্ছে। তোমরা চাইলে শুরুতেই একটি precompute করে রেখে এখান থেকে n সরিয়ে ফেলতে পার। আমরা n এর লুপ চালিয়ে বের করছি যে mask1 প্রশ্নের জন্য কতগুলির ক্ষেত্রে উত্তর mask2. আমরা যা করতে পারি তাহল DP চালানর আগেই $2^m \times n$ একটি লুপ চালিয়ে $2^m \times 2^m$ সাইজের একটি টেবিল precompute করে রাখতে পারি যে mask1 প্রশ্নের জন্য mask2 উত্তর কতগুলি সংখ্যার জন্য। এতে করে আমাদের time complexity কমে হবে $\mathrm{O}\big(2^m \times 2^m \times m\big)$.

এখানে একটা জিনিস, যদিও আমি বললাম যে আমাদের time complexity $\mathrm{O}\big(2^m \times 2^m \times m\big)$ কিন্তু আসলে আমাদের complexity $\mathrm{O}\big(3^m \times m\big)$. কেন? কারণ খেয়াল কর, প্রতিটি প্রশ্নের জন্য আসলে আমরা 3 টি স্টেট রাখছি। একটি হল "প্রশ্ন করি নি", আরেকটি হল "প্রশ্ন করেছি এবং 0 উত্তর" আর সর্বশেষটি হল "প্রশ্ন করেছি এবং 1 উত্তর"। যেহেতু আমাদের কাছে মোট m টি প্রশ্ন আছে সেহেতু আমাদের state হল 3^m টি, $2^m \times 2^m$ না। যদিও আমরা $2^m \times 2^m$ মেমরি ব্যবহার করছি। তোমরা চাইলে 3^m মেমরিও ব্যবহার করতে পার। এক্ষেত্রে দুটি bitmask না রেখে একটি bitmask রাখতে হবে কিন্তু সেটি হবে 3 base এর। আগে তো 0 আর 1 দিয়ে বুঝাচ্ছিলাম যে প্রশ্ন করেছি কি করে নাই বা প্রশ্নের উত্তর কি 0 না 1. 3 base এর ক্ষেত্রে আমরা যেটা করব তাহল 0 দিয়ে বুঝাব প্রশ্ন করি নাই, 1 দিয়ে বুঝাব প্রশ্ন করেছি কিন্তু প্রশ্নের উত্তর 0 আর 2 দিয়ে বুঝাব প্রশ্ন করেছি কিন্তু প্রশ্নের উত্তর হল 1. বা চাইলে তোমরা তোমাদের ইচ্ছা মত state define করে নিতে পার।

UVa 1412 Fund Management

সমস্যা: তোমার কাছে কিছু টাকা আছে, টাকার পরিমান 0.01 হতে 10^9 হতে পারে। তুমি m দিন ধরে কিছু শেয়ার কেনা বেচা করতে পার ($1 \leq m \leq 100$). তুমি n ধরনের শেয়ার কেনা বেচা করতে পার ($n \leq 8$). শেয়ারগুলির প্রতিদিনের দাম দেওয়া আছে আর এও বলা আছে যে তুমি কোন শেয়ার সর্বোচ্চ কত সংখ্যক একবারে কিনে রাখতে পারবে। অর্থাৎ কোনো একটা শেয়ারের লিমিট যদি হয় 2 এর মানে হল এমন কোনো সময় থাকা যাবে না যখন তোমার কাছে সেই শেয়ার 3 টি আছে। তুমি চাইলে বহুবার 1 টি কিনে 1 টি বেচতে পার, কিন্তু কোনো সময়ে 2 টির বেশি না থাকলেই হলো। এরকম সকল শেয়ারের লিমিট আছে এবং এই লিমিট সর্বোচ্চ 8 হতে পারে। এছাড়াও মোট শেয়ারের

উপরও আরেকটা লিমিট আছে অর্থাৎ তুমি কোনো এক সময়ে এর বেশি সংখ্যক শেয়ার রাখতে পারবা না এবং এই লিমিট সর্বোচ্চ 8 হতে পারে। তোমাকে প্রতিটি শেয়ার এর প্রতিদিন এর দাম দেওয়া আছে। তুমি এক দিনে খুব জোড় একটি শেয়ার কিনতে পারবে বা একটি শেয়ার বেচতে পারবে, একই সাথে কিনতে ও বেচতে পারবে না বা একাধিক কেনা বেচাও করতে পারবে না। বলতে হবে তুমি সকল শর্ত মেনে কেনা বেচা করলে কত লাভ হবে। এছাড়াও বলতে হবে তুমি কীভাবে শেয়ার কেনা বেচা করবে।

সমাধান: DP টা বেশ সহজ। তুমি একদিন একদিন করে আগাবে আর সেই দিন তুমি কিছু কিনবে নাহয় বেচবে নাহয় কিছুই করবে না। সুতরাং তোমাকে দুটি জিনিস মনে রাখতে হবে (state). এক- তুমি এখন কত তম দিনে আছ, দুই- তোমার কাছে এখন কোন শেয়ার কয়টি করে আছে। তুমি যদি এই দুটি জিনিস মনে রাখতে পার তাহলে তো DP করা কোনো ব্যাপারই না। কথা হল কোন জিনিস কয়টা করে আছে সেটা state করে রাখলে কি খুউউউব বড় state হয়ে যায় না? এখানে আমাদের 8 রকমের শেয়ার থাকতে পারে আর 8 রকমের শেয়ার এর প্রতিটি 9 রকম ভাবে থাকতে পারে (যেমন কোনো একটি শেয়ার 0 টি, 1 টি এরকম করে সর্বোচ্চ 8 টি থাকতে পারে)। সুতরাং আমাদের state এর সাইজ হয় $9^8 = 43 \times 10^6$ আর এর সাথে গুন হবে দিনের সংখ্যা 100 অর্থাৎ 4×10^9 এর মত। এর ভিতরে আবার আমরা কোন শেয়ার বেচব বা কোন শেয়ার কিনব তার একটি লুপ চালাব। সুতরাং আমাদের time complexity হয়ে যাচ্ছে $8 \times 4 \times 10^9$. অনেক বড়! একে তো সময়ে আটবে না, দুই এ এতো বড় state মেমরিতে ধরবে না।

কি করা যায়? একটা জিনিস খেয়াল কর আমরা যে ধরেছি যে আমাদের কাছে এখন কোন কোন শেয়ার আছে তা 9^8 রকম হতে পারে এটা কিন্তু একটা overestimate. কারণ মনে কর প্রথম শেয়ার তোমার কাছে 8 টা আছে, তাহলে কিন্তু পরের আর কোনো শেয়ার তুমি কিনতে পারবে না। এখন প্রশ্ন হল এটা কত বড় overestimate? তোমরা চাইলে একটি backtrack কোড লিখে দেখতে পার যে আসলে কত গুলি valid state আছে, অর্থাৎ তুমি খুব জোড় 8 ধরনের শেয়ার কিনতে পারবা আর মোট 8 টার বেশি শেয়ার কিনতে পারবে না। তাহলে কতগুলি ভিন্ন ভিন্ন valid state হতে পারে। অথবা চাইলে তুমি mathematically ও বের করতে পার। এখানে আমাদের প্রশ্ন হল 8 ধরনের জিনিসকে 8 টি বাক্সে মোট কত ভাবে আমরা distribute করতে পারি। এটা combinatorics এ খুবই common সমস্যা। 8 টি জিনিসকে 8 টি 1 মনে কর, আর 8 টি বাক্সের জন্য 7 টি 0 কল্পনা কর। এবার এই $8 + 7 = 15$ টি জিনিসকে permute কর। শুরু হতে প্রথম 0 পর্যন্ত যত গুলি 1 আছে সেগুলি থাকবে প্রথম বাক্সে, প্রথম 0 হতে দ্বিতীয় 0 এর মাঝের 1 থাকবে দ্বিতীয় বাক্সে, এরকম করে শেষ (সপ্তম) 0 এর শেষে যা থাকবে তা থাকবে শেষ (অষ্টম) বাক্সে। তার মানে আমরা যদি বের করতে পারি এই 15 টি ডিজিট কত ভাবে permute করা যায় তাহলেই আমরা পেয়ে যাব কত ধরনের state থাকবে। 15টি ডিজিট যার 8 টি 1 আর 7 টি 0 এদেরকে মোট $\frac{15!}{7!8!} = 6435$ ভাবে permute করা যায়। এটা ঠিক যে আমরা এখানে কিছু জিনিস ধরি নাই যেমন- 7 টি জিনিসকে 8 টি বাক্সে বা 7 টি জিনিসকে 7 টি বাক্সে বা ইত্যাদি। কিন্তু এটুকু বুঝা যাচ্ছে যে আমাদের মোট state এর সংখ্যা আসলে 9^8 এর মত ভয়ঙ্কর বড় হবে না। কারণ দেখ যদি আমাদের 8 টির কম ধরনের শেয়ার থাকে তাহলে আমরা চাইলেই নতুন শেয়ার সৃষ্টি করতে পারি যার লিমিট হবে 0 বা যার দাম প্রতিদিন 0 টাকা হবে। তাহলেই হবে। বাকি থাকল তুমি তো কোনো এক মুহূর্তে 8 টি শেয়ার না রেখে 7 টি শেয়ারও রাখতে পার বা 6 টি বা এরকম আর কি। সেক্ষেত্রে মনে কর তোমার কাছে 8 টি না, মোট 9 টি বাক্স আছে। আর তুমি শেষ বাক্স ধরে ফেলে দিবে। তাহলেই তো $0 \ldots 8$ যেকোনো পরিমান বলকে 8 টি বাক্সে কত ভাবে রাখা যায় তা বের হয়ে যাবে তাই না? এর পরিমান কত? $\frac{16!}{8!8!} = 12870$. সুতরাং আমাদের valid state আসলে বেশি না।

তাহলে আমরা কোড করব কীভাবে? যদি memory সমস্যা না হত তাহলে আমরা চাইলে 9^8 রকম করে state represent করতে পারতাম। এতে TLE খাবা না কিন্তু MLE খেয়ে যেতে পার। তাহলে কি করা যায়?

একটি উপায় হল state কে save করে রাখার জন্য আমরা যে array ব্যবহার করে থাকি তা না করে একটি map ব্যবহার করতে পার। অর্থাৎ তুমি DP এর ভিতরে ঢুকে map চেক করে দেখবা যে তুমি এই state এ আগে এসেছ কিনা আসলে উত্তর রিটার্ন করে দাও। আর না আসলে উত্তর বের হয়ে গেলে তা map এ রেখে দাও পরে কাজে লাগবে এই ভেবে। কোন শেয়ার কয়টি করে আছে সেটি প্রকাশের জন্য তোমরা চাইলে 9 base এর সংখ্যা ব্যবহার করতে পার। আবার চাইলে একটি

vector ব্যবহার করতে পার। কারণ তুমি তো map ই ব্যবহার করছ, সুতরাং vector কে map করে রাখতে কোডের দিক থেকে কোনো সমস্যা হবে না। যদিও হয়তো একটি 9 base সংখ্যা রাখা cost efficient. আবার cost efficient নাও হতে পারে, কারণ তোমাকে 9 base এর নাম্বার থেকে গুন ভাগ mod করে বুঝতে হবে কোন শেয়ার কয়টা আছে ইত্যাদি, যা হয়তো costly হতে পারে। তোমাকে একটু experiment করে দেখতে হবে আর কি। 9 base এর নাম্বার মানে তো বুঝছ? এর আগে যেরকম 3 base এর নাম্বার ব্যবহার করেছিলাম ঠিক সেরকম।

UVa 10934 Dropping Water balloons

সমস্যা: তোমার কাছে মোট k টি পানি ভরা বেলুন আছে ($k \leq 100$). তোমাকে বলতে হবে n তালা একটি বিল্ডিঙের সবচেয়ে নিচের কোন তালা থেকে বেলুনটি ফেললে তা ফেটে যায় (n এর মান 64 বিট এর একটি সংখ্যা)। তোমাকে এই কাজ সবচেয়ে কম বার চেষ্টা করে বের করতে হবে। একটা চেষ্টা মানে কোনো এক তালা হতে একটি বেলুন ফেলা। মনে কর তুমি কোনো এক তালা হতে বেলুন ফেললে আর তা ফাটল না, তাহলে কিন্তু চাইলে সেই বেলুন তুমি আবারো ব্যবহার করতে পারবে। যদি তোমার 63 এর বেশি বার বেলুন ফেলতে হয় তাহলে বল যে উত্তর 63 এর বেশি।

সমাধান: সমস্যাটা একটু জটিল। বেলুনের সংখ্যা বেশ কম মাত্র 100, কিন্তু বিল্ডিঙের উচ্চতা বেশ বেশি। তবে একটা জিনিস, তাহল উত্তর কিন্তু খুব কম, মাত্র 63 হতে পারবে খুব জোড়। এর বেশি হলে তা নিয়ে না ভাবলেও হবে। একটা টেকনিক হল তোমার যেটা উত্তর সেটাকে DP এর state বানানো, আর ইনপুটে দেওয়া কোনো বড় সঙ্খ্যাকে DP এর উত্তর বানানো। যেমন এই ক্ষেত্রে আমরা বলতে পারি, তোমার কাছে i টা বেলুন আছে আর তুমি j বার চেষ্টা করতে পারবে- তাহলে সর্বোচ্চ কত উচ্চতার বিল্ডিং এর ক্ষেত্রে তুমি উত্তর বের করতে পারবে? যদি আমরা সকল i, j এর জন্য এই মান জেনে যাই তাহলে n এর জন্য আমাদের উত্তর বের করা বেশ সহজ হবে।

তাহলে আমাদের প্রশ্ন হল i টি বেলুন ব্যবহার করে j চেষ্টায় আমরা কত উচু বিল্ডিঙের উত্তর বের করতে পারব। সমস্যাটা খুব একটা কঠিন না। ধরা যাক কোনো এক তালা হতে আমরা একটি বেলুন ফেললাম। যদি এটি না ফাটে, তাহলে আমরা i টি বেলুন ব্যবহার করে $j-1$ চেষ্টায় ধরা যাক A উচ্চতার বিল্ডিঙের উত্তর বের করতে পারি। আর যদি বেলুনটি ফেটে যায় তাহলে আমাদেরকে দেখতে হবে $j-1$ চেষ্টায় $i-1$ টি বেলুন ব্যবহার করে কত উচ্চতার বিল্ডিঙের উত্তর বের করতে পারি, ধরা যাক B. তাহলে আমাদের i, j এর জন্য উত্তর হবে $A+B+1$. এভাবে i, j এর একটি DP করলেই হয়ে যাবে। আমাদেরকে base case নিয়েও একটু চিন্তা ভাবনা করতে হবে বা এই যে কিছুক্ষণ আগের ফর্মুলা $A+B+1$ এর 1 নিয়েও হয়তো একটু চিন্তা ভাবনা করতে হবে। কিন্তু মূল আইডিয়া হল এটা।

UVa 1336 Fixing the great wall

সমস্যা: একটি দেয়ালে একটি রোবট আছে। রোবটটি শুরুতে কোথায় আছে তা বলা আছে। এই দেয়ালের কিছু কিছু স্থান নষ্ট হয়ে গিয়েছে। রোবট যদি সেসব স্থানের উপর দেয়া যায় তাহলে সেসব স্থান ভাল হয়ে যাবে। প্রতিটি নষ্ট স্থান সারাতে কত খরচ হবে তা দেওয়া আছে। তবে যদি কোনো নষ্ট স্থান সারান না হয় তাহলে তার খরচ প্রতি সেকেন্ডে কিছু পরিমান করে বৃদ্ধি পাবে (প্রতিটি নষ্ট স্থানের জন্য এই খরচ ভিন্ন ভিন্ন হতে পারে)। এই রোবটটির speed দেওয়া আছে এবং এই speed টি constant. এই speed এ রোবট বামে বা ডানে যেতে পারে। এখন বলতে হবে সবচেয়ে কম কত খরচে সবগুলি নষ্ট জায়গা সারান যাবে। মোট নষ্ট স্থানের সংখ্যা সর্বোচ্চ 1000.

সমাধান: মনে কর রোবট তার শুরুর জায়গা থেকে কিছু দূর বামে গেল। তাহলে সে যেটুকু স্থান কাভার করল সেসব স্থানে যেসব নষ্ট জায়গা ছিল তা ঠিক হয়ে যাবে। এখন কথা হল কীভাবে এই রোবট ডানে বামে যাবে যাতে করে সব নষ্ট স্থান কাভার হয়। হতে পারে যে রোবট প্রথমে একদম বামের নষ্ট স্থানে গেল এর পর সব থেকে ডানের। বা প্রথমে একদম ডানে গেল, এর পর সব থেকে বামে। কিন্তু এই দুই অপশনে তো optimal উত্তর নাও পাওয়া যেতে পারে। মনে কর তোমার ঠিক বামে এমন একটি

নষ্ট স্থান আসে যেটির খরচ প্রতি সেকেন্ডে অনেক করে বাড়ে। আবার মনে কর এরকম একটি জায়গা ডানেও আছে। তাহলে তোমার উচিত হবে তুমি বামে গিয়ে ঐ বেশি খরচের জায়গা কাভার করে ঘুরে ডানে এসে আবার বেশি খরচের জায়গা কাভার করা, এরপর বাকিটুকু কাভার করা যাবে। এখানে খেয়াল কর আমরা কিন্তু বামের বেশি খরচের জায়গা পার করে পরের নষ্ট জায়গা গুলিতে যাই নাই। এবং একবারে বাম/ডান গিয়ে পরে একবারে ডান/বাম যাওয়ার থেকে এভাবে যাওয়া অবশ্যই ভাল। তাহলে একটু চিন্তা করে দেখ তো আমাদের optimal উত্তর এর রাস্তা দেখতে কেমন হবে? Zig-Zag তাই না? অর্থাৎ প্রথমে কিছু দূর বামে, এরপর কিছু দূর ডানে, এর পর কিছু দূর বামে, আবার কিছু দূর ডানে এরকম করে। কিন্তু এটা কীভাবে DP এর state আকারে লিখবে? বা DP design করবে কীভাবে? আমাদের মনে রাখতে হবে যে আমরা বাম দিকে কত দূর ইতমধ্যেই কাভার করেছি আর ডান দিকে কত দূর। এই দুই মাথা রাখতে হবে। কিন্তু আমরা কোথায় আছি? খেয়াল কর আমরা কিন্তু বলতেই পারি আমরা হয় বাম মাথায় আছি নাহয় ডান মাথায়। মাঝে থেকে কিন্তু লাভ নেই। আমরা যখন বাম মাথায় আছে তখন চাইলে আরও বামে যেতে পারি। অথবা ডান দিকে যতদূর কাভার হয়েছে তার থেকেও বেশি দূর যেতে পারি। অর্থাৎ আমাদের state হবে i, j, k যেখানে i হল বামের মাথা, j হল ডানের মাথা আর k এর মান 0 মানে আমরা বামে আছি আর 1 মানে ডানে আছি। এই state হতে আমরা $i-1, j, 0$ তে যেতে পারি আবার চাইলে $i, j+1, 1$ এও যেতে পারি। এভাবে আমরা পুরো সমস্যা সমাধান করতে পারি।

অর্থাৎ আমাদের DP ফাংশনের parameter হল i, j, k. এর মানে- আমরা i হতে j কাভার করেছি এবং k এর মান অনুসারে আমরা হয় i এ আছি অথবা j তে আছি। এখন প্রশ্ন হল আমরা বাকি কাভার করা হয় নায় এরকম জায়গা গুলি কত কম খরচে কাভার করতে পারব। আগেই বলেছি আমরা এই state হতে $i, j+1, 1$ আর $i-1, j, 0$ এই দুই state এ যাবার চেষ্টা করব। ধরা যাক আমরা $i, j+1, 1$ এর জন্য DP কল করলাম এবং তা হতে DP রিটার্ন হল। এখানে খেয়াল কর কোনো এক নষ্ট স্থানের খরচ কিন্তু কোন সময়ে সেখানে গিয়েছ তার উপর নির্ভর করে। অর্থাৎ আমাদের state এর আরেকটা parameter লাগবে- সময়। কিন্তু তা করলে তো state অনেক বেশি হয়ে যাবে। আর কি উপায় আছে? উপায় হল, খেয়াল কর, মনে কর আমরা i হতে a তে যাচ্ছি। আর ইতমধ্যেই i হতে j কাভার হয়ে গিয়েছে। বাকি যারা আছে তাদের খরচ প্রতি সেকেন্ডে কত করে বাড়ে তা আমাদের জানা আছে। আমরা যা করব তাহল i হতে a তে যেতে যেই সময় লাগে সেই সময়ে এসব নষ্ট স্থানের খরচ কত বাড়বে সেটা হিসাব করে ফেলব। তাহলে খেয়াল কর কোনো নষ্ট স্থানে গেলে তার খরচ আর variable না, বরং fixed. এর খরচ যত বেড়েছে তা ইতমধ্যেই হিসাব করে ফেলা হয়েছে। সুতরাং আমাদের আর কোনো ঝামেলা বাকি নেই। আমরা প্রতি DP তে DP কল করব আর যেই স্থান গুলি বাকি আছে তাদের খরচ কত বাড়ল তা হিসাব করে ফেলব। একটা ছোট উদাহরন দেই। মনে কর আমরা $i = 1$ হতে $a = 3$ এ যাচ্ছি, আর আমাদের $j = 2$. 1 হতে 3 এ যেতে মনে কর সময় লাগে x. এখন আমাদের দেখতে হবে x সময়ে বাকি নষ্ট স্থান গুলির খরচ কত বাড়ে। সেই পরিমাণটাই আমরা যোগ করে ফেলব। ফলে সময়ের সাথে সাথে কোনো নষ্ট স্থানের খরচ যা বাড়ে তা প্রতি transition এই হ্যান্ডেল হয়ে যায়।

UVa 12105 Bigger is Better

সমস্যা: LED display এর মত 0 বানাতে 6 টা, 1 বানাতে 2 টা, 2 বানাতে 5 টা, 3 বানাতে 5 টা, 4 বানাতে 4 টা, 5 বানাতে 5 টা, 6 বানাতে 6 টা, 7 বানাতে 3 টা, 8 বানাতে 7 টা, 9 বানাতে 6 টা ম্যাচকাঠি লাগে। তোমাকে n টা ম্যাচকাঠি দেওয়া আছে। বলতে হবে এই ম্যাচকাঠি গুলি ব্যবহার করে m দ্বারা বিভাজ্য সবচেয়ে বড় কোন সংখ্যা তৈরি করতে পারবা। n এর মান সর্বোচ্চ 100 আর m এর মান সর্বোচ্চ 3000 হতে পারবে। যদি এরকম কোনো সংখ্যা তৈরি করা সম্ভব না হয় তাহলে -1 প্রিন্ট করতে হবে।

সমাধান: প্রথম দেখায় এটা খুব একটা কঠিন সমস্যা মনে হয় না। আমাদের প্রথম চিন্তা এর state কী হবে। সবার প্রথমে $n \times m$ এর মত state মাথায় আসে অর্থাৎ আমাদের এখন আর কয়টা ম্যাচকাঠি বাকি আছে আর বাকি যেই সংখ্যা বানাতে হবে তাকে m দিয়ে mod করলে কত হবে বা আমরা যেটুকু সংখ্যা বানিয়েছি তাকে m দিয়ে mod করলে কত হবে। আর DP এর value হিসাবে থাকবে যে কত

বড় সংখ্যা বানানো যায়। কিন্তু এখানে কিছু details বাকি আছে। যেমন আমরা বাম হতে ডানে অংক বসিয়ে বসিয়ে সংখ্যা তৈরি করব নাকি ডান হতে বামে বসিয়ে বসিয়ে, আমরা যেই সংখ্যা তৈরি করব তা সবচেয়ে বড় বানাব কীভাবে ইত্যাদি। ধীরে ধীরে আগানো যাক।

এখানে clash of interest আছে। মানে, মনে কর আমরা যদি বাম হতে ডানে সংখ্যা তৈরি করতাম তাহলে কিন্তু সংখ্যার mod কত হবে তা বের করা সহজ । আগের mod যদি a হয় তাহলে নতুন mod হবে $10a + d \pmod{m}$ যেখানে d হল তুমি এখন যেই ডিজিট বসালে তা। কিন্তু তুমি যদি ডান হতে বামে যাও তাহলে এটি একটু জটিল। মনে কর আমাদের একটি সংখ্যা আছে এবং আমরা এর বামে একটি ডিজিট বসাচ্ছি। ধরা যাক আমাদের ডানের সংখ্যার mod a এবং তার length b আর তুমি বামে যেই নতুন ডিজিট বসাচ্ছ তাহল d. তাহলে আমাদের নতুন mod হবে $d10^b + a \pmod{m}$. একটু জটিল। শুধু জটিল না আসলে এভাবে করলে dp এর state বেড়ে যাবে। খেয়াল কর, আমরা কোন ডিজিট d বসাব তার একটা লুপ চালাচ্ছি। আবার a আর m কিন্তু dp এর state এ থাকবে বলেছিলাম। কিন্তু আমাদের তো b ও দরকার। তাহলে b কেও state এ রাখতে হবে। যেহেতু n এর মান সর্বোচ্চ 100 আর 1 বানাতে আমাদের সবচেয়ে কম 2 টি কাঠির দরকার হয় তাই আসলে b এর সর্বোচ্চ মান $100/2 = 50$ (অর্থাৎ 50 টি 1). সুতরাং আমরা যদি ডান হতে বামে যাই তাহলে আমাদের একটি অতিরিক্ত প্যারামিটারের প্রয়োজন হবে। যদিও সেই প্যারামিটারের সর্বোচ্চ মান খুব একটা বড় না।

আমরা কিন্তু এটা preliminary চিন্তা ভাবনা করলাম। আমরা এখনও সবকিছু ফাইনাল করি নাই। শুধু মাত্র এটা দেখলাম যে আমরা বাম থেকে ডানে গেলে কি হয় আবার ডান থেকে বামে গেলে কি হয়। এছাড়াও আমাদের আরেকটা বিষয় মাথায় রাখতে হবে যে আমরা আমাদের শেষ উত্তরটা বড় করতে চাই।

আমরা প্রথমে দেখি বাম হতে ডানে গেলে কি হয়। অর্থাৎ আমরা বাম হতে একে একে ডিজিট বসাচ্ছি। ধরা যাক বর্তমান mod a, আর আমাদের n টি কাঠি বাকি আছে- এই দুটি হল state. আর আমাদের DP এর value হবে বাকি n টি কাঠি ব্যবহার করে (সব যে ব্যবহার করতে হবে এমন কথা নেই) এর ডানে সবচেয়ে কত বড় সংখ্যা তৈরি করা যায় যেন পুরো সঙ্খ্যাটি m দ্বারা ভাগ যায়। আমরা প্রতিটি state এ d এর একটি লুপ চালাব। d বসালে আমাদের নতুন mod হবে $10a+d \pmod{m}$. এবার এটি ব্যবহার করে আর n হতে d তে ব্যবহৃত কাঠিগুলি বাদ দিয়ে আমরা recursive ভাবে DP কল করব। আর আমাদের base case হবে যখন আমাদের state এর mod এর মান 0 বা আমাদের কাছে থাকা n দিয়ে যখন আর কাঠি বানানো সম্ভব না। এরকম করে আমরা প্রতিটি d এর জন্য চেষ্টা করব এবং দেখব সবচেয়ে কত বড় সংখ্যা আমরা ডানে বসাতে পারি।

এখন এই DP ব্যবহার করে আমরা path printing এর মত করে সবচেয়ে বড় সংখ্যা প্রিন্ট করতে পারি। প্রথমত খেয়াল কর আমাদের বামের সংখ্যা শূন্য হওয়া যাবে না। তাই আমরা 1 হতে 9 এর লুপ চালাব এবং DP কে বলব যে আমরা এই সংখ্যা বসিয়েছি, আমাদের বর্তমান mod এতো, এখন বল আমাদের ডানে সবচেয়ে বড় কত লেংথের সংখ্যা বসান যায়? 1 হতে 9 এর যেটি বসালে সবচেয়ে বেশি সংখ্যক অংক বসান যায় আমরা সেই অংকটি বসাব, যদি একাধিকের জন্য সবচেয়ে বেশি সংখ্যক অংক বসান যায় তাহলে আমরা তাদের মধ্য হতে সবচেয়ে বড় অংক বসাব। এরপর হতে আমরা 0 হতে 9 এর জন্য চেষ্টা করব। অর্থাৎ এদের প্রতিটি বসিয়ে বের করব আমাদের বর্তমান mod কত, আর কয়টি কাঠি বাকি আছে। তার উপর ভিত্তি করে আমরা DP কল করব।

যদি আমরা ডান হতে বামে DP করতে চাই তাহলে আমাদের state হবে আমাদের ডান দিকের লেংথ কত, ডান দিকের সংখ্যার mod কত আর কয়টি কাঠি বাকি আছে- এখন বল বাম দিকে সর্বোচ্চ আর কতগুলি অংক বসান যাবে। কিন্তু এতে একটা সমস্যা আছে। এরকম DP এর ক্ষেত্রে path printing একটু সমস্যা। যদি যেকোনো path চাইত তাহলে হয়তো সমস্যা ছিল না, কিন্তু সবচেয়ে বড় সংখ্যা চেয়েই একটু কঠিন করে ফেলেছে। এভাবে চিন্তা কর, আমরা ডান হতে বামে সংখ্যা বসাচ্ছি। মনে কর ডানে 7 বসালাম এরপর DP তে দেখলাম যে বামে আর 3 টি সংখ্যা বসান যাবে। আবার মনে কর 5 বসালেও DP বলছে যে আর 3 টি বসাতে পারব। কিন্তু আমরা তো জানি না কোনটা বসালে ভাল হবে। আমরা যদি ডান হতে বামে আসতাম তাহলে ব্যাপারটা সহজ হত, তাই না? এই জন্য এই সমস্যার ক্ষেত্রে ডান হতে বামে DP করা সমস্যা। অসম্ভব কিন্তু না। আমরা যদি DP এর return মান হিসাবে বামে কত বড় লেংথের সংখ্যা বসান যায় তা না রেখে, বামে কত বড় সংখ্যা অর্থাৎ পুরো সংখ্যাটাই রাখি তাহলেই হবে। কিন্তু এক্ষেত্রে আমাদের bigint ব্যবহার করতে হবে কারণ আমাদের উত্তর তো 50 লেংথের হতে পারে।

সুতরাং আমাদের যদি সবচেয়ে বড় বা ছোট path (লেংথে না, lexicographically বা numerically) প্রিন্ট করতে বলে তাহলে একটু সাবধান হয়ে DP ডিজাইন করতে হবে।

UVa 1204 Fun Game

সমস্যা: একটি স্থানে কিছু ছেলে মেয়ে গোল হয়ে বসে আছে। তুমি randomly একজনকে একটি কাগজ দেবে। সে সেখানে B লিখবে যদি সে ছেলে হয়, আর G লিখবে যদি সে মেয়ে হয়। এরপর সে তার পছন্দ মত এক দিকে (clockwise বা anti-clockwise) সেই কাগজটি তার পাশের জনকে দেবে। তার পাশের জনও একই ভাবে B বা G লিখবে। এরপর প্রথম জন যেদিকে কাগজ দিয়েছিল দ্বিতীয় জনও একই দিকে দিবে। এভাবে চলতে থাকবে। কোনো একজন এর কাছে গিয়ে এই কাগজ শেষ হবে। সেই কোনো একজন কে তার ঠিক নেই। এরকম করে তোমাকে সর্বোচ্চ মোট 16 টি কাগজ দেওয়া আছে। সেই কাগজ গুলিতে কি লিখা আছে তা তোমাদের ইনপুট। বলতে হবে সবচেয়ে কম কতজন ছেলে মেয়ে থাকলে সবগুলি কাগজ valid. ইনপুট string গুলির লেংথ সর্বোচ্চ 100 হতে পারে।

সমাধান: সমস্যাটা বেশ কমন। আমরা সমস্যাটা একটু সহজ করি। মনে কর আমরা জানি যে সবসময় clockwise যাবে। তাহলে আমাদের সমাধান কি হবে? এটার সমাধান আমরা শিখব। তাহলে তুমি মূল সমস্যার সমাধান নিজেরাই করতে পারবে।

এই সমাধানের জন্য তোমাকে প্রথমেই দেখতে হবে যে প্রদত্ত N টি স্ট্রিং এর মাঝে এমন কেউ আছে কিনা যে অন্য কারও ভেতরে substring আকারে আছে। থাকলে ছোটটাকে ধরে ফেলে দিতে হবে। কারণ আমাদের উত্তরে যদি বড়টা থাকে তাহলে তো ছোটটা এমনি এমনিই থাকবে তাই না? আমরা যদি এই ছোটদেরকে সরিয়ে না ফেলি তাহলে আমাদের সমাধানে সমস্যা হবে। এখন আরও মনে কর এমন কোনো স্ট্রিং নেই যেটা পুরা circle ঘুরে আসে। আমরা পরে দেখব এই assumption না থাকলেও আমাদের সমাধান কীভাবে কাজ করে। তার মানে আমাদের optimal উত্তর হবে এরকম- প্রথমে একটি স্ট্রিং থাকবে S_1, তাকে যতদূর সম্ভব overlap করে এরপর থাকবে S_2, একে যতদূর সম্ভব overlap করে থাকবে S_3 এরকম করে সবশেষে থাকবে S_N. এই S_N আবার ঘুরে গিয়ে S_1 এর সাথে সবচেয়ে বেশি দূর overlap করে থাকবে। খেয়াল কর যখন আমরা S_3 বসাচ্ছি তখন আমরা কিন্তু শুধু S_2 নিয়েই চিন্তা করছি, S_1 কে পাত্তাই দিচ্ছি না। কেন? কারণ S_3 যদি S_1 এর সাথে overlap করে আর এদের মাঝে যদি S_2 থাকে, এর মানে S_2 পুরোপুরি S_3 এর ভেতরে আছে। কিন্তু এরকম স্ট্রিং তো আমরা শুরুতেই সরিয়ে ফেলেছি তাই না? সুতরাং আমাদের state এ যা রাখতে হবে তাহল এখন পর্যন্ত কোন কোন স্ট্রিং ব্যবহার করে ফেলেছি আর সর্বশেষে কোন স্ট্রিং ব্যবহার করা হয়েছে। ঠিক travelling salesman problem এর মত। তাহলে আমরা কিভাবে সমাধান করব? প্রথমে আমরা যেকোনো একটি স্ট্রিং নিয়ে শুরু করব (TSP তে যেরকম যেকোনো একটি নোড থেকে শুরু করি)। যেকোনো স্ট্রিং থেকে শুরু করলেই হবে কারণ প্রতিটি স্ট্রিংই তো আমাদের circle এ থাকবে তাই না? সুতরাং আমরা একটি স্ট্রিং নিয়ে শুরু করলাম (ধরা যাক S_1). এর পর আমাদের state হবে সর্বশেষ কোন স্ট্রিং ব্যবহার করেছি আর কোন কোন স্ট্রিং এই পর্যন্ত ব্যবহার করেছি। এরপর আমরা কোন স্ট্রিং নিব তার উপর লুপ চালাব। দেখব যে এই স্ট্রিং ইতোমধ্যেই নেওয়া হয়েছে কিনা। যদি নেওয়া হয়ে থাকে তাহলে তো আর এটা নেবার প্রশ্নই আসে না। আর যদি না নেয়া হয় তাহলে এটা নেয়ার চেষ্টা করব। প্রশ্ন হল এটা নিলে cost কত হবে? Cost হবে এর আগের স্ট্রিং এ এই স্ট্রিং যতদূর সম্ভব overlap করার পর যেটুকু স্ট্রিং বাকি থাকে যোগ বাকি অংশের DP. এভাবে যেসকল স্ট্রিং নেওয়া বাকি আছে তাদের মাঝে যেটি নিলে আমাদের cost সবচেয়ে কম হয় সেটিই নিব। এভাবে আমরা DP শেষ করব। তাহলে এই DP তে আমরা স্ট্রিং গুলি একের পর এক সবচেয়ে কম খরচে জোড়া লাগিয়ে একটা চেইন বানিয়েছি, এখনও একটি circle বানাই নাই। Circle বানানোর জন্য তেমন কিছু না আমাদের base case নিয়ে একটু চিন্তা করতে হবে। আমাদের এই DP এর base case কি? যখন সব স্ট্রিং নেওয়া হয়ে গিয়েছে। আমরা যদি একটু চিন্তা করি তাহলে বুঝব যে সব শেষ স্ট্রিং একদম শুরুর স্ট্রিং এর সাথে সবচেয়ে বেশি পরিমান overlap করিয়ে দিতে পারলেই আসলে আমরা আমাদের circle কেস হ্যান্ডল করে ফেলতে পারব (হয়তো এতো সহজ না, একটু চিন্তা ভাবনা করে সঠিক ফর্মুলা বের করতে হবে, কিন্তু মূল আইডিয়া এটাই)। অর্থাৎ আমাদের

এতক্ষণ যেই মোট খরচ হয়েছে তা হতে শুরুর আর শেষের স্ট্রিং যত খানি overlap করে সেটুকুর cost আমাদের বিয়োগ করতে হবে। এই DP এর time complexity $O(n^2 2^n)$ যদি আমরা দুটি স্ট্রিং সবচেয়ে বেশি কত খানি করে overlap করে থাকে সেটুকু precompute করে রাখি। ঐ জিনিস precompute করতে তো বেশি cost নেই তাই না? খুব জোড় $O(n^2 L^2)$ যেখানে সর্বোচ্চ লেংথ হল L.

তাহলে আমরা সবগুলি স্ট্রিং ব্যবহার করে সবচেয়ে কম লেংথের একটি circle বানিয়ে ফেললাম। এবার আসা যাক আমরা যে কিছু জিনিস বাদ দিয়েছিলাম সেখানে। আমরা ধরে নিয়েছিলাম যে আমাদের এমন কোনো স্ট্রিং নেই যেটি পুরো circle কাভার করে না। আমরা এই restriction উঠিয়ে নেই। একটু চিন্তা করলে বুঝবে এই কেস এমনি এমনিই হ্যান্ডল হয়ে যায়। এটা কিন্তু আমি অনেক বড় কথা বলে ফেলেছি। আশা করি এই কথা তোমরা চোখ বুজে বিশ্বাস করবে না। একটু কাগজে কলমে করে দেখ কেন এই কথা সত্য। ফরমালি এই কথা প্রমাণ করা খুব একটা সহজ কাজ না। আবার বিভিন্ন কেস দিয়েও যে তোমাদের কে convince করার চেষ্টা করব সেটাও খুব একটা সহজ কাজ না। সহজ কাজ হবে তোমাদের উপর ছেড়ে দেওয়া :) তোমরা যেভাবে চিন্তা করবে তাহল anti case বানানোর চেষ্টা করবে। মনে কর তুমি মনে করতেছ যে এরকম case এ তোমার এই সমাধান ভুল উত্তর দেবে। কিন্তু যখন দেখবে যে দেয় না তখন চিন্তা করবে কেন দেয় না, কীভাবে এই সমাধান তোমার case কে হ্যান্ডল করছে। এভাবে তুমি আসলে বুঝতে পারবে কেন এই সমাধান সঠিক। অথবা আরেক কাজ করতে পার, প্রতিটি স্ট্রিংকে ছোট করে ফেলতে পার।

বাকি থাকল আমরা যে শুধু clockwise ধরেছি সেটা। এটা হ্যান্ডল করা খুব একটা কঠিন না। যেটা করবে সেটা হল প্রতিটা স্ট্রিং এর দুই রকম ভাবে নিতে পার এক সোজা ভাবে আরেকটা উলটো করে। সুতরাং আমাদের bitmask এ N টিই বিট থাকবে কোন স্ট্রিং নেওয়া হয়েছে কোনটা নেওয়া হয় নাই তা বুঝানর জন্য। ওদিকে আরেকটা থাকবে যা বলবে সর্বশেষ কোন স্ট্রিং নিয়েছিলে সেটা বুঝানর জন্য। আরেকটা নতুন সংখ্যা নিতে হবে যেটা বলবে যে শেষ স্ট্রিংটা কি সোজা নেওয়া হয়েছিল নাকি উলটো। ব্যাস!

UVa 12099 The Bookcase

সমস্যা: N টি বই আছে ($N \leq 70$). প্রতিটি বই এর height এবং thickness দেওয়া আছে। Height সর্বোচ্চ 300 এবং thickness সর্বোচ্চ 30 হতে পারে। তোমাকে এই বইগুলি রাখার জন্য একটি বুকসেলফ বানাতে হবে। যেহেতু শেলফের depth বইগুলির মাঝে যার width সবচেয়ে বেশি, তার সমান হবে সেহেতু আমরা বই এর width আর শেলফের depth নিয়ে ভাবব না। আমাদের শেলফে তিনটি থাক থাকা লাগবে। প্রতিটি থাকের উচ্চতা হবে সেই থাকে থাকা বই গুলির মাঝে যার উচ্চতা সবচেয়ে বেশি তার সমান। আর বুকশেলফের প্রসস্থ হবে প্রতিটি থাকে থাকা বইগুলির thickness এর যোগফল যার সবচেয়ে বেশি তার সমান। অর্থাৎ আমরা বইগুলিকে tightly pack করে একটি বুকশেলফ বানাব। আমাদের লক্ষ্য এই শেলফের ক্ষেত্রফল (অর্থাৎ উচ্চতা গুন প্রস্থ) যেন সবচেয়ে কম হয়। এই সবচেয়ে কম ক্ষেত্রফল তোমাকে প্রিন্ট করতে হবে।

সমাধান: এই সমস্যাটা বেশ সুন্দর। কারণ এখানে অনেক parameter আছে। সুতরাং প্রব্লেম পড়ার সাথে সাথে বুঝা যায় না আমাদের expected time complexity কত হবে বা তোমার state এ কি কি রাখবা। খুব naive ভাবে state দাঁড় করালে state এ কি কি থাকে? তিন শেলফের উচ্চতা, তিন শেলফের বর্তমান প্রসস্থ, আর বর্তমানে বই এর অ্যারের প্রথম হতে কয়টা বই নেওয়া হয়েছে। অবশ্যই এতগুলি প্যারামিটার নেওয়া সম্ভব না, তোমরা চাইলে এর স্টেটের complexity বের করে দেখতে পার। প্রশ্ন হল এখন কি কি ভাবে এই অবস্থা ভাল করা যায়। প্রথমে খেয়াল কর, আমরা চাইলে শুরুতেই বলতে পারি যে আমারা সবচেয়ে উচু বই প্রথম থাকে রাখব। এর ফলে আমাদেরকে আর তিনটি শেলফের উচ্চতা রাখা লাগছে না, দুটি রাখলেই হয়ে যাচ্ছে। আমরা আমাদের সুবিধার জন্য বইগুলিকে তাদের উচ্চতা অনুসারে সর্ট করি আর প্রথম বইকে প্রথম থাকে দিয়ে দেই। তাহলে আমাদের 7 টা প্যারামিটার থেকে একটা কমলো। আরেকটা জিনিস খেয়াল কর, আমাদের কিন্তু সব শেলফের প্রসস্থ রাখার দরকার নেই। আমরা জানি কোন কোন বই নিয়েছি, আর যদি আমরা জানি দুই শেলফের বই এর প্রসস্থ তাহলেই আমরা তৃতীয় শেলফের প্রসস্থ বের করে ফেলতে পারব। বাকি

থাকল 5 টা। সুতরাং আমাদের কাছে আছে দুইটা শেলফের উচ্চতা, দুইটা শেলফের প্রসস্থ আর বই এর অ্যারের কোন পর্যন্ত বই নেওয়া হয়েছে। কিন্তু এই 5 টা কিন্তু independent, সুতরাং এদেরকে আর এভাবে কমান সম্ভব না। একটা কমন টেকনিকের কথা বলেছিলাম। তাহল প্যারামিটারকে ভ্যালুতে পাঠানো। খেয়াল কর আমাদের এই DP এর ভ্যালু হল 0 বা 1, সম্ভব কি সম্ভব না। আমরা কি কোনো ভাবে উত্তর বা কোনো একটা প্যারামিটারকে ভ্যালুতে আনতে পারি না? সম্ভব কি সম্ভব না সেটা নাহয় -1 দিয়ে বুঝাব? এরকম হতে পারে- আমরা দুইটা শেলফের প্রসস্থ জানি (তৃতীয় শেলফের প্রসস্থ হল এই পর্যন্ত নেওয়া বই এর প্রসস্থ বিয়োগ এই দুইটা শেলফের প্রসস্থ), আর একটা শেলফের উচ্চতা জানি, আর কোন কোন বই নেওয়া হয়েছে তাও জানি। ভ্যালু তে থাকবে তৃতীয় শেলফের উচ্চতা সবচেয়ে কম কত হলে আমাদের state এর 4 টা প্যারামিটারই কাজ করবে। শুধু শুধু তো তৃতীয় শেলফের উচ্চতা বাড়িয়ে লাভ নাই তাই না? আমাদের যত কমানো যায় ততই লাভ। সুতরাং এভাবে আমরা 4 টা প্যারামিটারে নামাতে পারি। তাহলে আমাদের complexity দাঁড়ায় $2100^2 \times 70 \times 300$. এখনও বেশ বেশি। অন্তত আরও একটি কমাতে হবে। এখন খেয়াল কর আমাদের কি চেয়েছে। আমাদের চেয়েছে মোট ক্ষেত্রফল। আমরা তো ভ্যালু বা প্যারামিটার হিসাবে একটি উচ্চতা বা একটি প্রস্থ এরকম করে রাখছি। আমরা কি আমাদের উত্তর কি চেয়েছে তার ভিত্তিতে কোনো optimize করতে পারি? এই ধাপটাই মনে হয় সবচেয়ে কঠিন। উত্তর বলে দিচ্ছি- আমরা চাইলে ভ্যালু হিসাবে একটা শেলফের উচ্চতা না বরং দুইটি শেলফের উচ্চতার যোগফল রাখতে পারি। আর একটি প্যারামিটার ব্যবহার করতে পারি এটা বলতে যে এখানে কয়টা শেলফের উচ্চতা আছে, একটা না দুইটা। সুতরাং আমাদের state হবে $2100^2 \times 70 \times 2$ আর ভ্যালুতে থাকবে দুইটি (বা একটি) শেলফের উচ্চতার যোগফল। নিঃসন্দেহে এটা খুব একটা সহজ observation না। কিন্তু এটা একটা বিশাল শিক্ষা!

UVa 12170 Easy Climb

সমস্যা: n টি পাহাড় পাশাপাশি আছে ($n \leq 100$). প্রতিটি পাহাড়ের উচ্চতা h_i দেওয়া আছে ($h_i \leq 10^9$). তুমি চাইলে যেকোনো পাহাড়ের উচ্চতা কমাতে বা বাড়াতে পার। এর জন্য cost হবে তুমি মোট যেই কয় unit কমালে বা বাড়ালে (সবসময় integer unit করে কমবে বা বাড়বে)। পাহাড়ের উচ্চতাগুলি এমন হতে হবে যেন পাশাপাশি দুটি পাহাড়ের উচ্চতার পার্থক্য d এর বেশি না হয় ($d \leq 10^9$). বলতে হবে সবচেয়ে কম কত cost এ এই কাজ করা সম্ভব। যদি সম্ভব না হয় তাহলে impossible প্রিন্ট করতে হবে।

সমাধান: বলার অপেক্ষা রাখে না যে সমস্যাটা খুব একটা সহজ নয়। খুব naive ভাবে চিন্তা করলে state হবে আমরা এখন কোন পাহাড়ে আছি, আর বর্তমান পাহাড়ের modified height কত। তাহলে আমরা কোনো একটি state এ এসে এর পরের পাহাড়ের height কত হবে তার উপর লুপ চালাব তাহলেই হবে। যেহেতু কোনো পাহাড়ের height সর্বোচ্চ 10^9 হতে পারে সেহেতু এটা একটা ব্যাপক costly সমাধান। কী করা যায়?

একটু চিন্তা করলেই বুঝা যায় যে- সব height আমাদের জন্য important না। কিন্তু কোন কোন height আমাদের জন্য important তাও বুঝতে পারা খুব একটা সহজ না। মনে কর $i < j$. যদি $h_j > h_i + d * (j - i)$ হয় বা $h_j < h_i - d * (j - i)$ হয় এর মানে i আর j এর মাঝের পাহাড়ের উচ্চতা যেভাবেই রাখো না কেন সেটা valid arrangement হবে না। কারণ i এর পরের পাহাড় তার থেকে খুব জোড় d উচ্চতা বেশি হতে পারে, তার পরের পাহাড় $2d$, তার পরের পাহাড় $3d$, এরকম করে j তম পাহাড় খুব জোড় $d * (j - i)$ উচু হতে পারবে। একই ভাবে কত কম উচু হতে পারবে সেটাও বের করা যাবে। মূল কথা আমরা যদি কোনো একটি পাহাড় ফিক্স করি তাহলে এর আগের বা পরের পাহাড় গুলির জন্য bound পাওয়া যাবে। সবগুলি পাহাড়কে তার bound গুলি মানতে হবে। এখানে মূল observation হল কোনো একটি কলামে (পাহাড়ে) যতগুলি boundary আছে সেগুলিই হলো important height. অর্থাৎ তুমি প্রতিটি পাহাড়কে ফিক্স মনে করে বের কর যে i তম পাহাড়টির জন্য bound কি কি। এই সব bound এর মান গুলিই হলো important height. এখানে একটা জিনিস খেয়াল কর, সেটা হল- optimal pattern যে এই সব height ব্যবহার না করে হতে পারবে না- তা না। কিন্তু এই সব height ব্যবহার করে optimal মান পাওয়া যাবেই। চাইলে এটা আমরা

ফর্মাল ভাবে প্রমাণ করতে পারি। আমি নিজেও ঠিক নিশ্চিত না কীভাবে এটা প্রমাণ করা সম্ভব। কিন্তু আসলে প্রগ্রামিং কন্টেস্টে আমরা অনেক জিনিস ফর্মাল ভাবে প্রমাণ করি না। বরং কিছু উদাহরন, কিছু intuition এর উপর নির্ভর করে আমরা সমাধান করে থাকি। তাহলে প্রতিটি পাহাড়ে আমাদের $O(n)$ টি সম্ভব স্থান থাকবে (আসলে $2n-1$, কারণ প্রতিটি j তে অন্য সকল i এর জন্য উপরে আর নিচে দুটি করে বাউন্ড পাওয়া যায়, আর তাছাড়া ঐ কলামের initial height তো আছেই)। সুতরাং আমরা i তম পাহাড়ের প্রতিটি interesting height থেকে পরের পাহাড়ের প্রতিটি interesting height গুলিতে যাবার চেষ্টা করব (d এর লিমিট মেনে)। অর্থাৎ $O(n^3)$ সমাধান।

চাইলে অবশ্য একে $O(n^2)$ এ পরিনত করতে পার। মূল আইডিয়া হল, মনে কর তুমি i তম পাহাড়ের j তম interesting height এর জন্য optimal মান বের করেছ। এখানে মনে কর $i-1$ তম পাহাড়ের a হতে b পর্যন্ত interesting height গুলি হতে আসা যায়। এখন তুমি যদি i তম পাহাড়ের $j+1$ তম interesting height এর জন্য optimal মান বের করতে চাও তাহলে একবার চিন্তা করে দেখ তো $i-1$ পাহাড়ের কোন কোন interesting height হতে আসতে পারবে? প্রায় a হতে b পর্যন্ত interesting height হতে তাই না ($a < b$)? মনে কর $j+1$ তম interesting height, j তম interesting height অপেক্ষা বড়। এর মানে হয়তো দেখা যাবে a স্থানটি $j+1$ তম interesting height হতে d এরও বেশি দূরে চলে গিয়েছে। সুতরাং আমাদেরকে সেটা খেয়াল রাখতে হবে এবং প্রয়োজনে a এর মান বাড়াতে হবে (বাড়ানো নাও লাগতে পারে, বা একাধিকবারও বাড়াতে হতে পারে)। একই ভাবে দেখা যাবে $b+1$ তম interesting height টি আমাদের থেকে d দুরত্বে চলে এসেছে। সুতরাং b এর মানও হয়তো বাড়ানোর প্রয়োজন হতে পারে। এভাবে আমরা $j+1$ এ কই কই থেকে আসা যায় সেই $[a,b]$ রেঞ্জের মান আমরা খুব সহজেই বের করে ফেলতে পারি। এভাবে এক বাড়ায়ে কমায়ে $[a,b]$ বের করার কারণ হল এতে করে i তম পাহাড়ের সকল interesting height এর জন্য এই রেঞ্জ linear সময়ে বের করে ফেলা যাবে (প্রতিটি linear না, সবগুলি মিলে linear. একে amortized time বলে)। আমরা যখন রেঞ্জ জেনে ফেলেছি বাকি টুকু করা বেশ সহজ। এখন আমাদের সমস্যা হল- আমরা i তম পাহাড়ের j তম interesting height এর জন্য উত্তর বের করছি, আর $i-1$ তম পাহাড়ের $[a,b]$ রেঞ্জের interesting height হতে আমরা এখানে আসতে পারি। ধরা যাক আমরা k থেকে আসছি ($a \le k \le b$), সেক্ষেত্রে আমাদের cost হবে- $dp[i-1][k] + ABS(height[i-1][k] - height[i][j])$. এরকম সকল k এর জন্য আমাদের যেসকল cost পাওয়া যাবে, তাদের মাঝ হতে সবচেয়ে কমটি আমাদের নিতে হবে। এটি খুব সহজেই RMQ ব্যবহার করে করা যায়। মনে করা যাক আমরা $[a,b]$ কে দুটি ভাগে ভাগ করছি $[a,c],[c+1,b]$ যেন $height[i][j] \ge height[i-1][c]$ এবং $height[i][j] \le height[i-1][c+1]$ হয়। তাহলে $[a,c]$ অংশের জন্য cost এর ফর্মুলা আমরা লিখতে পারি $dp[i-1][k] - height[i-1][k] + height[i][j]$ আর অন্য অংশের জন্য আমরা লিখতে পারি $dp[i-1][k] + height[i-1][k] - height[i][j]$. খেয়াল কর তুমি যখন i তম পাহাড়ের j তম interesting height এ দাড়িয়ে আছ তখন তোমাদের জন্য শেষ term একটি constant. বাকি থাকছে শুরুর অংশটুকুকে minimize করা। আমরা শুরুর দুটি টার্মকে একত্র করে একটি RMQ বানাব এবং আমাদের দরকারি রেঞ্জে ($[a,c]$ বা $[c+1,b]$) query করব।

UVa 1380 A scheduling problem

সমস্যা: একটি n নোডের ট্রি দেওয়া আছে ($n \le 200$). ট্রিটির edge গুলি directed ও হতে পারে আবার undirected ও হতে পারে। তোমাদেরকে undirected edge গুলিতে direction দিতে হবে যেন ট্রির সবচেয়ে লম্বা চেইন ছোট হয়। চেইন মানে হল একই direction এ পর পর কিছু edge.

সমাধান: এটিও বেশ কঠিন সমস্যা। মনে কর তুমি সকল undirected edge কে মুছে ফেললে, এর ফলে সবচেয়ে বড় যেই chain হয় তার দৈর্ঘ্য মনে কর d. তাহলে তোমার উত্তর কমপক্ষে d হবেই। মনে কর আমাদের কাছে একটি ফাংশন আছে (এটি একটি সংখ্যা x নেয়) যাকে আমরা যদি জিজ্ঞাসা করি যে undirected edge গুলির direction এমন ভাবে সেট করা সম্ভব কিনা যেন সবচেয়ে বড় চেইনের দৈর্ঘ্য x এর চেয়ে বেশি না হয়। এরকম একটি ফাংশন থাকলে আমরা একে একে একে $d, d+1, d+2 \ldots$ নিয়ে প্রশ্ন করব। তাহলেই আমরা আমাদের উত্তর পেয়ে যাব।

এখন আমাদের কাজ হল এই ফাংশন লিখা। আমরা আসলে এই ট্রি এর উপরে DP করব। প্রতিটি নোডে দুই ধরনের মান থাকবে, f আর g. ধরা যাক i একটি নোড। আমরা তো i নোডের নিচের undirected edge সমুহকে নানা ভাবে direction দিতে পারি যেন x এর থেকে বেশি বড় chain তৈরি না হয়। এরকম যত ভাবে orient করা সম্ভব তার মাঝে কোনো একটির ক্ষেত্রে তো i হতে নিচে গিয়েছে এরকম চেইনের দৈর্ঘ্য সবচেয়ে কম হবে তাই না? মনে কর সেই দৈর্ঘ্য হল $f[i]$. একই ভাবে i এ উঠে এসেছে এরকম চেইনের দৈর্ঘ্য সবচেয়ে কম $g[i]$. আমরা দেখব এই f আর g কীভাবে বের করব। আমাদের নোড i এর কিন্তু তিন ধরনের child আছে। ধরা যাক প্রথম ধরনের child হল- a. এদের ক্ষেত্রে i হতে তাদের প্রতি ইতোমধ্যেই একটি directed edge আছে। দ্বিতীয় ধরনের child হল-b. এদের ক্ষেত্রে এদের হতে i এর দিকে ইতোমধ্যেই একটি directed edge আছে। আর তৃতীয় ধরনের child হল c. এদের সাথে i এর undirected edge আছে। মনে কর a সমুহের মাঝে সবচেয়ে বড় f এর মান হল $f[a]$. একই ভাবে b সমুহের মাঝে সবচেয়ে বড় g এর মান হল $g[b]$. এর মানে আমাদের ইতোমধ্যেই $f[a] + g[b]$ দৈর্ঘ্যের চেইন তৈরি হয়ে গিয়েছে। সুতরাং আমাদের দেখতে হবে এটি যেন x এর থেকে বেশি না হয়ে যায়। এখন বাকি থাকে c নোড গুলি। এদের মাঝে কিছু নোডের ক্ষেত্রে i হতে নিচে direction এ edge হবে, আর কিছুর ক্ষেত্রে উপরের direction এ। মনে কর আমরা c কে দুই ভাগে ভাগ করলাম cf আর cg. i থেকে cf এ edge থাকবে আর cg থেকে i এ থাকবে। এখন মনে কর cf সমুহের মাঝে সবচেয়ে f এর মান বেশি ধরা যাক $f[cf]$. এখন যদি cg সমুহের মাঝে যদি এমন কোনো নোড থাকে যার f এর মান $f[cf]$ এর থেকে কম, তাহলে কিন্তু আমরা তাকে cg হতে সরিয়ে cf এ আনতে পারি। এতে করে কিন্তু আমাদের কোনো চেইনের দৈর্ঘ্য কিন্তু একটুও বাড়বে না। কারণ এই নতুন আসা নোডের থেকেও বড় f আলা একটি নোড ইতিমধ্যেই আমাদের cf এর মাঝে আছে। সুতরাং আমরা যেটা করতে পারি তাহল, c এর নোড সমুহকে f এর মান অনুযায়ী সর্ট করতে পারি। এর পর একটি লুপ চালিয়ে ঠিক করতে পারি যে কোন টুকু cf, আর কোন টুকু cg. এটুকু যদি ঠিক হয়ে যায় তাহলে আমরা $f[cf]$ আর $g[cg]$ বের করে ফেলতে পারব। $f[cf]$ তো আমরা লুপের ভেতরেই বের করে ফেলতে পারব, আর $g[cg]$ এর মান আমরা নাহয় আগেই উল্টোদিক হতে লুপ চালিয়ে precompute করে রাখব। যাই হোক, সুতরাং আমাদের কাছে এখন 4 টি মান আছে- $f[a], g[b], f[cf], g[cg]$. এদের হতে আমরা প্রথমে দেখব যে এরা x এর শর্ত ভঙ্গ করে কিনা। যদি না করে তাহলে আমরা এখান থেকে optimal $f[i]$ বের করার চেষ্টা করব। একই ভাবে i নোডের g এর মান বের করব। এভাবে আমরা সকল নোডের f, g বের করব। এই প্রসেসের সময় যদি কখনই x এর শর্ত ভঙ্গ নাহয়, তার মানে এই x ই আমাদের উত্তর।

মজার জিনিস হল, এই সমস্যার উত্তর হয় d হবে নাহয় $d + 1$. সেজন্য আমরা শুধু $x = d$ চেষ্টা করলেই বুঝব যে উত্তর কি d নাকি $d + 1$. কেন $d + 1$ এর বেশি লাগবে না তার প্রমাণ আমি জানি না। হয়তো তোমরা কাগজে কলমে করলে নিজেকে convince করতে পারবে। এই সমাধানের time complexity $\mathrm{O}(n^2)$.

UVa 10559 Blocks

সমস্যা: একটি বাক্সের সারি দেওয়া আছে। এখানে সর্বোচ্চ 200 টি বাক্স থাকতে পারে। বাক্স গুলি বিভিন্ন রঙের হতে পারে। তুমি যদি কোনো বাক্সে ক্লিক কর তাহলে তার সাথে লাগানো (পাশাপাশি লাগানো থাকতে হবে) আগে পরে যত একই রঙের বাক্স আছে সব গায়েব হয়ে যাবে। যদি মোট k টি বাক্স গায়েব হয় তাহলে তোমার score হবে k^2. এই গায়েব হয়ে যাওয়া বাক্সের দুই পাশের বাক্সগুলি আবার কিন্তু পাশাপাশি চলে আসবে। তোমাকে বাক্সগুলিকে ক্লিক করে করে সকল বাক্স গায়েব করতে হবে। তোমার লক্ষ্য হল তোমার মোট score সর্বোচ্চ করা।

সমাধান: এটি খুব একটা কঠিন না। এখানে যেই জিনিসটা সমস্যাটাকে কঠিন করে ফেলে সেটা হল তুমি যদি কোথাও ক্লিক কর তাহলে কিছু বাক্স গায়েব হয়ে যায় আর বামের আর ডানের বাক্স গুলি জোড়া লেগে যায়। এর ফলে এই বাক্সের সারিকে state আকারে প্রকাশ করাটা খুব একটা সহজ কাজ না। সুতরাং এভাবে চিন্তা করলে হবে না।

মূল সমাধানে যাবার আগে আমরা একটা কাজ করি। তাহল আমাদের ইনপুট গুলি আলাদা আলাদা

বাক্স হিসাবে চিন্তা না করে, একই রঙের বাক্স গুলিকে আমরা গ্রুপ করে ফেলি। ফলে আমাদের ইনপুট বা আমাদের অ্যারে হবে বাক্সের গ্রুপের। প্রতিটি গ্রুপের একটি করে রঙ থাকবে আর বাক্সের পরিমান থাকবে।

আমাদের state মনে কর (i, j) অর্থাৎ আমাদেরকে i হতে j তম গ্রুপকে প্রসেস করতে হবে। এখন চিন্তা কর তুমি বাম থেকে ডানে আসছ। তোমার হাতে দুইটা অপশন। তুমি বামের গ্রুপকে ক্লিক করবে। তাহলে i তম গ্রুপকে ক্লিক করলে যত cost হবে তা পাবে আর $(i+1, j)$ কে সমাধান করার জন্য যেই cost হবে সেটা। এটাই হল প্রথম অপশনের cost. দ্বিতীয় অপশন হল ক্লিক না করা। সেই ক্ষেত্রে পরবর্তীতে এই গ্রুপ পরের একই রঙের আরেকটি গ্রুপের সাথে merge হবে, ধরা যাক সেটি হল k. এখন খেয়াল কর i আর k কে merge করতে হলে কিন্তু $(i+1, k-1)$ এই রেঞ্জকে গায়েব করতে হবে। এটুকু গায়েব করার cost কিন্তু আমরা dp কল করলেই পাব। এখন আমাদের i আর k গ্রুপ merge হয়ে যাবে। এই জিনিসটা আমরা dp তে কীভাবে করব? খুব একটা কঠিন না, আমরা (i, j) এর সাথে সাথে আরও একটি parameter রাখব, prev. যেটা বলবে আমাদের বর্তমান গ্রুপ i এর সাথে i এর বামদিকে থেকে আসা কত গুলি বাক্স merge হবে (যেমন উপরে আমরা (k, j) কল করব সেই সাথে তৃতীয় parameter এ বলব যে k গ্রুপের সাথে i এ থাকা বাক্সগুলি যোগ হবে)। তাহলে আমাদের dp এর parameter হল $(i, j, prev)$, আর এর ভেতরে আমরা k এর একটি লুপ চালাব। সুতরাং আমাদের dp টি হবে $O(n^4)$ যদিও সব স্টেট visit হবে না।

UVa 1228 Integer Transmission

সমস্যা: তোমার কাছে n বিটের একটি সংখ্যা আছে। তুমি একে একে বিট গুলি একটি নেটওয়ার্ক এর ভেতর দিয়ে পাঠাবে। i তম বিট তুমি পাঠাবে i তম সেকেন্ডে। এই বিটটি receiver পাবে $i+1$ হতে $i+d+1$ তম সেকেন্ডের যেকোনো সময়ে। যদি একাধিক বিট একই সময়ে পৌছায় তাহলে তাদের অর্ডার যেকোনো হতে পারে। যদি receiver তার প্রাপ্ত বিটগুলিকে পাওয়ার অর্ডারে সাজায় তাহলে বলেত হবে কত গুলি ভিন্ন ভিন্ন বিট sequence সে পেতে পারে। n আর d এর মান সর্বোচ্চ 64 হতে পারে।

সমাধান: বেশ সুন্দর সমস্যা। যদি d এর মান খুব বেশি নাহত তাহলে হয়তো শেষ d বিটের কি অবস্থা সেটা বিটমাস্কে রাখতে পারতে। কিন্তু d এর মান বেশ বড়। কি করা যায়? এভাবে চিন্তা করতে পার- মনে কর তুমি আউটপুট জান। তুমি কি বলতে পারবে কোন বিট কোথায় থেকে এসেছে? যদি তুমি এই কাজ করতে পার তাহলে তুমি যেটা করবে তাহল তুমি বাম হতে ডানে যাবে আর 0 বা 1 বসানোর চেষ্টা করবে। আর প্রতিবার জিজ্ঞাসা করবে যে এই বিট কোথায় থেকে এসেছে। যদি এই জিজ্ঞাসার উত্তর হয় না এটা কোথাও থেকে আসতে পারে না, তাহলে তুমি ঐ স্থানে ঐ বিট বসাতে পারবে না। এভাবে তুমি count করতে পার যে কত ভাবে তোমার আউটপুট গঠিত হতে পারে। এখন কথা হল তুমি কীভাবে বলবে যে কোন বিট কোথায় থেকে এসেছে। এটা কিন্তু খুব একটা কঠিন না। আউটপুটের i তম 0 আসবে ইনপুটের i তম 0 থেকে (একই ভাবে 1 বিট ও)। খেয়াল কর, আউটপুটের i তম 0 যে ইনপুটের i তম 0 হতেই আসবে তার কিন্তু কোনো নিশ্চয়তা নেই। কিন্তু যেকোনো valid আউটপুটের জন্য আমরা যদি মনে করি যে ইনপুটের i তম 0 আউটপুটের 0 থেকে এসেছে তাতে কোনো সমস্যা নেই। আমরা মনে করতেই পারি যে এর মানে কোনো স্থানে 0 বা 1 বসাতে গেলে তোমাকে জানতে হবে যে ইতোমধ্যেই তুমি কতগুলি 0 বা 1 বসিয়েছ। এই দুটি information জানলেই তুমি বলতে পারবে যে তুমি কি 0 বসাতে পারবে কিনা বা 1 বসাতে পারবে কিনা। মনে কর আমরা ইতোমধ্যেই $i-1$ টি 0 আর $j-1$ টি 1 ব্যবহার করে ফেলেছি। আমরা কি এখন 0 বসাতে পারব? এর উত্তর কীভাবে দেবে? মনে কর $i-1$ তম 0 আর $j-1$ তম 1 এর মাঝে যেটি সবচেয়ে পরে ছিল তা ছিল t তম বিট। এখন দেখ i তম 0 কত তম বিট, ধরা যাক s তম। এর মানে প্রশ্ন হল t তম বিট আসার পর কি s তম বিট আসতে পারে কিনা, অর্থাৎ $s-t$ এর সাথে d কে তুলনা করে দেখতে হবে যে এই কাজ সম্ভব কিনা।

UVa 1628 Pizza Delivery

সমস্যা: একটি 1d coordinate system এ তোমার n টি বন্ধুর বাসা আছে। n এর মান সর্বোচ্চ 100. সেই সাথে তুমি কোথায় আছ সেটাও বলা আছে। প্রতি unit distance যেতে এক unit করে সময় লাগে। তোমাকে সবার বাসায় যেতে হবে। তোমাকে তোমার বন্ধুদের মোট waiting time কমাতে হবে। মোট waiting time হল তোমার সকল বন্ধুর waiting time এর যোগফল। কোনো একটি বন্ধুর waiting time হল কত সময় পর তুমি তার সাথে প্রথম দেখা করতে গিয়েছ। যেমন মনে কর তোমার দুটি বন্ধু আছে এক জন আছে 8 এ, আরেকজন আছে 11 তে। তুমি আছ 10 এ। তুমি যদি প্রথমে প্রথম বন্ধুর বাসায় যাও তাহলে সময় লাগবে 2, এরপরে প্রথম বন্ধুর বাসা হতে দ্বিতীয় বন্ধুর বাসায় যেতে সময় লাগবে 3, অর্থাৎ তুমি 5 সময় পর দ্বিতীয় বন্ধুকে দেখা করতে গিয়েছ। সুতরাং তোমার বন্ধু দের waiting time হল 2 এবং 5. মোট waiting time 7. তুমি যদি প্রথমে দ্বিতীয় বন্ধুর সাথে দেখা করতে এরপর প্রথম বন্ধুর সাথে তাহলে তোমার মোট waiting time হত 4. তোমাকে বলতে হবে মোট waiting time সবচেয়ে কম কত করা সম্ভব।

সমাধান: এটুকু তো বুঝা যাচ্ছে যে তুমি zig zag করে যাবে। অর্থাৎ প্রথমে কিছুদুর বামে, এরপর কিছুদুর ডানে এরকম করে তুমি তোমার বন্ধুদের বাসায় যাবে। সুতরাং তোমার state হবে বামে কোন বন্ধুর বাসায় ইতোমধ্যেই গিয়েছ, ডানে ইতোমধ্যেই কোন বন্ধুর বাসায় গিয়েছ আর এখন তুমি কোথায় আছ ডানে না বামে। অর্থাৎ আমাদের স্টেট $2n^2$. খেয়াল কর, আমরা যদিও কোনো সময়ে আমাদের রেঞ্জের ভেতরে থাকতে পাইর, কিন্তু আসলে তা state হিসাবে রাখলে costly হয়। এর থেকে তুমি বাম প্রান্তে বা ডান প্রান্তে থাকবে এবং এর পর ঠিক করবে তুমি কই যাবে, বামে পরের বন্ধুর বাসায় যাবে নাকি ডানে পরের বন্ধুর বাসায় যাবে।

মনে কর তুমি $(left, right, side)$ এ আছ। এরপরে তুমি হয় $left-1$ এ যাবে অথবা $right+1$ এ যাবে। এখানে left মানে হল left তম বন্ধুর বাসা (একই ভাবে right). side এর উপর ভিত্তি করে সেখানে যেতে কত সময় লাগবে তা বের করা সম্ভব। কিন্তু সমস্যা হল এতে করে তো আমরা যেই বন্ধুর বাসায় যাচ্ছি তার waiting সময় বের হচ্ছে না। আসলে এই সমস্যায় যার কাছে যাচ্ছ তার waiting সময় বের করলে হবে না। বরং এইযে এখন যেই সময় লাগছে সেটা কিন্তু এখন পর্যন্ত যার যার বাসায় যাওয়া বাকি তাদের সকলের waiting সময়ে contribute করছে। যদি এর পরের বন্ধুর বাসায় যেতে সময় লাগে t (বর্তমান বন্ধুর বাসা হতে পরের বন্ধুর বাসা দুরত্ব), আর এখনও যদি k জন বন্ধুর বাসায় যাওয়া বাকি থাকে তাহলে মোট kt waiting সময় contribute হবে। এরপর আমরা dp কল করব। এভাবে সকল contribution যোগ করলে আমরা মোট waiting time পেয়ে যাব।

৮.১ অনুশীলনী

৮.১.১ সমস্যা

Simple

- UVa 10285 Longest Run on a Snowboard
- UVa 1213 Sum of Different Primes
- UVa 1650 Number String
- UVa 1543 Telescope
- UVa 1645 Count
- UVa 12063 Zeros and Ones

Easy

- UVa 10618 Tango Tango Insurrection
- UVa 10118 Free Candies
- UVa 1629 Cake slicing
- UVa 1630 Folding
- UVa 242 Stamp and Envelope Size
- UVa 10723 Cyborg Genes
- UVa 1631 Locker
- UVa 1633 Dyslexic Gollum
- UVa 12093 Protecting Zonk
- UVa 12589 Learning Vector
- UVa 1379 Pitcher Rotation
- UVa 1371 Period
- UVa 1637 Double Patience
- UVa 1393 Highways
- UVa 1390 Interconnect
- UVa 1414 Hanoi Towers
- LOJ 1085 All Possible Increasing Subsequences

Medium

- UVa 1439 Exclusive Access 2
- UVa 1375 The best name for your baby
- UVa 1632 Alibaba
- UVa 10641 Barisal Stadium
- UVa 1289 Stacking Plates
- UVa 1443 Garlands
- UVa 1579 Matryoshka
- UVa 11982 Fantasy Cricket
- UVa 12371 Guards
- UVa 12212 Password Remembering
- UVa 1654 Pairs of integers

Hard

- UVa 1634 The Picnic
- UVa 10271 Chopsticks

৮.১.২ হিন্ট

UVa 10118: বেশ tricky সমস্যা। প্রতিটি stack আর কয়টি করে ক্যান্ডি আছে এবং এখন ঝুরিতে কোন কোন টাইপের ক্যান্ডি আছে তা মিলে স্টেট। কিন্তু এই স্টেট তো বিশাল! প্রায় $40^4 \times 20^5$. মজার জিনিস হল কোন কোন ক্যান্ডি এখন ঝুরিতে আছে তা স্টেট এ রাখার প্রয়োজন নেই। কারণ তুমি যদি জান যে এখন কোন কোন ক্যান্ডি stack এ আছে এর মানে তুমি জান কোন কোন ক্যান্ডি ইতোমধ্যেই তুমি প্রসেস করেছ। আর প্রসেস করার সময় তো জোড়া হলেই তুমি পকেটে ঢুকিয়েছ তাই না? এর মানে তুমি শুধু দেখবে কোন কোন ক্যান্ডি এই মুহূর্তে odd সঙ্খ্যকবার প্রসেস হয়েছে। সেগুলিই এখন ঝুরিতে পরে আছে। অর্থাৎ তুমি চাইলে DP ফাংশনের প্যারামিটার হিসাবে ঝুরিতে কোন কোন ক্যান্ডি আছে তা পাঠাতে পার, কিন্তু তা dp এর state হিসাবে রাখার কোনো প্রয়োজন নেই।

UVa 1439: এটা বেশ মজার সমস্যা। প্রথমত তুমি একে গ্রাফে represent করে ফেল। প্রতিটি character এর জন্য একটি করে নোড থাকবে আর প্রতিটি প্রসেসে যেই দুটি character আছে তাদের মাঝে একটি edge দিয়ে দাও। তোমার সমস্যা তাহলে দাঁড়াল যে- প্রতিটি edge কে একটি direction দিতে হবে যেন এখানে কোনো cycle তৈরি না হয় এবং সবচেয়ে বড় চেইন এর দৈর্ঘ্য যেন সবচেয়ে কম হয়। এখানে একটা জিনিস খেয়াল রাখতে হবে যে আমাদের নোডের সংখ্যা কিন্তু মাত্র 15. আরও একটা জিনিস খেয়াল করতে পার তাহল এখানে কিন্তু বলে নাই যে "যদি সবসময় deadlock এ পরে তাহলে impossible প্রিন্ট কর"। এর মানে উত্তর সবসময় সম্ভব। কেন? উত্তর কিন্তু বেশ সহজ আমরা চাইলে নোডগুলিকে একটা অর্ডারে রাখতে পারি আর প্রতিটি edge কে আগের হতে পরের দিকে direction দেব, অর্থাৎ topological sort এর মত। মানে আমরা প্রথমে নোডগুলিকে একটা অর্ডার দেব, ধর $L - M - N - \ldots$ এরকম, এর পর যদি L আর N এর মাঝে কোনো edge থাকে তাহলে দেখবে এদের কে আগে কে পরে আছে আমাদের অর্ডারে। এক্ষেত্রে L আগে আর N পরে। তাহলে আমরা L হতে N এর direction এ edge দেব। তাহলেই আমাদের গ্রাফে আর সাইকেল

থাকবে না। কিন্তু আমাদের লক্ষ্য হল chain এর দৈর্ঘ্য সবচেয়ে কম করা। এই কাজ কীভাবে করবা? মনে কর আমাদের সবচেয়ে বড় চেইনের দৈর্ঘ্য d. তাহলে আমরা কিন্তু এই নোডগুলিকে d গ্রুপে ভাগ করতে পারি যেখানে প্রতিটি গ্রুপের মাঝে কোনো edge নেই, আর এই গ্রুপের অর্ডার অনুসারে edge গুলির direction হবে। এর মানে আমাদের সমস্যা দাঁড়াল আমরা আমাদের নোডগুলিকে সবচেয়ে কম কত গুলি গ্রুপে ভাগ করতে পারি যেন এই গ্রুপগুলির নিজেদের মাঝে কোনো edge না থাকে। কীভাবে করবা? খুব একটা কঠিন না। যেহেতু আমাদের নোড সংখ্যা কম সেহেতু আমরা নোড সমুহের একটি bitmask রাখতে পারি যে কোন কোন নোডগুলি ইতোমধ্যেই নেওয়া হয়ে গেছে। এরপর এখন ঠিক করবে কোন গ্রুপ নেবে। এইযে কোন গ্রুপ নেবে তাকে অবশ্যই এখন পর্যন্ত নেওয়া হয় নাই তার submask হতে হবে। এই submask এর ভেতরে যদি কোনো edge না থাকে তাহলেই কেবল আমরা এই submask নিতে পারব। তাহলে কোন কোন submask আমরা নিতে পারব তা আগে থেকেই precompute করে ফেলি। প্রতিটি mask এর জন্য আমরা দুইটি for লুপ চালিয়ে দেখব এই মাস্কের ভেতরে আছে এরকম নোডদের ভেতরে কোনো edge আছে কিনা। যদি থাকে তার মানে এই মাস্ক invalid আর না থাকলে valid. এই precompute করতে $\mathrm{O}(2^n n^2)$ এর বেশি সময় লাগবে না। আর ওদিকে আমাদের dp তে সময় লাগবে 3^n কারণ আমরা প্রতিটি মাস্কে গিয়ে তার submask নিয়ে নিয়ে চেষ্টা করছি। কেন এটা $2^n \times 2^n$ নাহয়ে 3^n হল তা তো আমরা আল্গরিদম ও ডেটা স্ট্রাকচার বই হতে জেনে ফেলেছিই।

UVa 1375: এই ধরনের সমস্যার ক্ষেত্রে একটি ট্রিক হল কোনো রুল যদি হয় $A = XYZ$ তাহলে একে পরিবর্তন করে লিখা $A = XT_1, T_1 = YZ$ অর্থাৎ কোনো রুলের ডান দিকে দুইটির বেশি character থাকবে না। এই কাজ করলে dp করতে সহজ হবে। আমাদের dp এর state হবে আমরা এখন কোন character এ আছি (হতে পারে এটি ইনপুটের কোনো character বা কোনো temporary character, যেমন উপরের T_1) আর আমাদের কত লেংথের শব্দ বানাতে হবে। রিটার্ন ভ্যালু হবে সেই লেংথের lexicographically সবচেয়ে ছোট স্ট্রিং অথবা empty স্ট্রিং যদি সেই লেংথের কোনো শব্দ বানানো সম্ভব না হয়। তাহলে তুমি dp এর ভেতরে কি করবে? প্রথমত দেখবে বর্তমান character কি ছোট হাতের কিনা (terminal symbol) তাহলে তো একটাই অপশন, 1 লেংথের শব্দ (যদি স্টেট এর লেংথ 1 নাহয় তার মানে কোনো শব্দ বানানো সম্ভব না)। আর যদি terminal symbol না হয় এর মানে আমরা এই symbol কে নানা ভাবে expand করতে পারি। আমরা সকল ভাবেই expand করার চেষ্টা করব এবং এই সকল চেষ্টার মাঝে lexicographically সবচেয়ে ছোটটি তুমি রিটার্ন করবে। এখন এই যে expand করার চেষ্টা- সেটা কি ভাবে করবে? যেহেতু আমাদের সর্বোচ্চ মাত্র দুটি character থাকবে expansion এ, সুতরাং এই কাজ বেশ সহজ। আমরা একটি লুপ চালিয়ে ঠিক করব কত লেংথ বাম থেকে আসবে, কত লেংথ ডান থেকে।

অধ্যায় ৯

Graph Theory

UVa 1599 Ideal Path

সমস্যা: একটি n নোডের গ্রাফ দেওয়া আছে ($n \leq 100000$). গ্রাফের edge গুলি (সর্বোচ্চ 200000 টি) undirected এবং weighted. তোমাকে গ্রাফে 1 হতে n এ সবচেয়ে optimal path বের করতে হবে। কোনো একটি পাথ optimal হবে যদি তার থেকে ছোট দৈর্ঘ্যের আর কোনো পাথ না থাকে এবং একই দৈর্ঘ্যের পাথের মাঝে তা যদি lexicographically সবচেয়ে ছোট হয়।

সমাধান: বুঝাই যাচ্ছে এর সমাধান BFS. আমাদের optimal path এর দৈর্ঘ্য কত হবে সেটা BFS করে খুব সহজেই বের করা যায়। ধরা যাক আমরা 1 হতে একটি আর n হতে আরেকটি BFS চালিয়ে 1 এবং n হতে প্রতিটি নোডের দুরত্ব বের করেছি। মনে কর 1 হতে v নোডের দুরত্ব f[v] আর n হতে v নোডের দুরত্ব g[v]. যদি কোনো নোড v এর জন্য যদি $f[v] + g[v]$ এর মান $f[n]$ এর সমান না হয় তাহলে সেই v নোডকে মুছে ফেলা যায়। এভাবে আমরা সকল অপ্রয়োজনীয় নোডকে মুছে ফেলব।

এবার আমরা 1 নোডকে ধরে পুরো গ্রাফকে ঝুলিয়ে দেই। প্রথম লেয়ারে থাকবে 1, দ্বিতীয় লেয়ারে থাকবে $f[v] = 1$ ওয়ালা নোড সমুহ, এরকম করে সব শেষ লেয়ারে থাকবে n. অর্থাৎ আমরা $f[v]$ অনুসারে নোডদেরকে লেয়ারে লেয়ারে সাজিয়েছি। এখন আমাদের কাজ হল lexicographically ছোট পাথ বের করা।

মনে কর শুরুতে কেবল মাত্র 1 active, আর বাকি সকল নোড deactive. প্রথমে আমরা প্রথম লেয়ার দেখব (এই লেয়ারে কেবল 1 আছে)। এই লেয়ারে যতগুলি active নোড আছে তাদের সেই সব neighbor দের দেখব যারা পরের লেয়ারে আছে। এদের মাঝে যেসব edge আছে তাদের মাঝে সবচেয়ে ছোট লেবেল (ছোট weight) বের করতে হবে। এরপর যেসকল edge এর লেবেল এই ছোটটি, তার অপর প্রান্তের নোডগুলিকে আমরা active করে দেব। এরপর পরের লেভেলের active নোডগুলিকে একই ভাবে প্রসেস করব। এরকম করে সকল লেভেল প্রসেস করলে আমরা 1 হতে n পর্যন্ত পাথের edge সমুহের লেবেলগুলি পেয়ে যাব।

UVa 12171 Sculpture

সমস্যা: একটি sculpture কিছু rectangular 3d বাক্স দ্বারা গঠিত (সর্বোচ্চ 50 টি)। এমন হতে পারে যে কোনো একটি বাক্সের ভেতরে অন্য একটি বাক্স, বা দুটি বাক্স overlap করছে। এই পুরো sculpture কে তুমি একটি তরলে ডুবালে। বলতে হবে sculpture এর volume কত আর তরলে ডুবানো অবস্থায় এর পৃষ্ঠ তলের ক্ষেত্রফল কত। Coordinate গুলি 1 হতে সর্বোচ্চ 500 পর্যন্ত হতে পারে। মোট 100 টি টেস্টকেস থাকতে পারে।

সমাধান: সমস্যাটা কিন্তু বেশ সহজ। যেহেতু সর্বোচ্চ coordinate 500 হতে পারে সেহেতু আমরা $500 \times 500 \times 500$ সাইজের একটি গ্রিড নেব এরপর আমাদের যেসব বাক্স দেওয়া আছে তারা যেসব

গ্রিড পজিশন জুড়ে থাকে তাদের মার্ক করে দেব। সকল বাক্সের জন্য মার্ক করা শেষ হয়ে গেলে volume বের করা খুবই সহজ। শুধু গুনতে হবে মোট কয়টি তরলের ঘর আছে (কীভাবে বের করবা একটু পরেই বলছি)। বাইরের পৃষ্ঠের ক্ষেত্রফল বের করাও কিন্তু বেশ সহজ। আমরা নিশ্চিত যে $0, 0, 0$ বাইরের তরলের একটি ঘর। আমরা এই ঘর হতে শুরু করে একটি BFS করব যেটি সকল unmark ঘর দিয়ে যায়। এর ফলে আমরা সকল তরল এর সেল বের করে ফেলতে পারব (মোট আয়তন থেকে এটি বিয়োগ করলেই আমরা structure এর volume পাব)। এর ফলে কি হবে? এখন আমরা প্রতিটি ঘরে যেতে পারি, যদি এটা তরলের ঘর হয় তাহলে আমরা এর ডান বাম উপর নিচে সামনে পিছে দেখব যে কোনো বাক্সের সেল আছে কিনা। থাকলে আমরা পৃষ্ঠ তলের খেত্রফল এক বাড়ায়ে দেব। কিন্তু সমস্যা হল এটা বেশ costly. কারণ আমাদের test case এর পরিমাণ 100 টি। তাহলে 100 বার $500 \times 500 \times 500$ করা একটু সমস্যা। এছাড়াও আমরা শুরুতে 50 টি বাক্স নিয়ে তা দখল করে এরকম প্রতিটি ঘর মার্ক করেছি এটা করতেও কিন্তু প্রতি কেসে 50 বার 500^3 কাজ করতে হবে।

কীভাবে আমরা এই কাজকে একটু দ্রুত করতে পারি? coordinate compression টেকনিক। মনে কর তোমার কাছে একটি বাক্স আছে যার এক কোনা $(2, 2, 2)$ এ আর আরেক কোনা $(5, 5, 5)$ এ। আমাদের কিন্তু $500 \times 500 \times 500$ গ্রিডের দরকার নেই। খেয়াল কর $x = 100$ তে যা ঘটে $x = 200$ তেও একই জিনিস ঘটে। এখানে interesting coordinate হল $x = 2, 5$ একই ভাবে y, z এও। অর্থাৎ যদিও বাক্সগুলি বিশাল জায়গা জুড়ে আছে কিন্তু আমরা শুধু মাত্র গুটি কয়েক coordinate নিয়ে interested. যেহেতু আমাদের মাত্র 50 টি বাক্স আছে সেহেতু আমরা প্রতিটি বাক্সের বাম আর ডান মিলায়ে মোট 100 টি x coordinate নিয়ে interested (একই ভাবে y আর z coordinate ও)। সুতরাং আমাদের আসলে 500^3 সাইজের গ্রিড না নিয়ে 100^3 সাইজের গ্রিড নিয়ে কাজ করলেই হবে। ঠিক কীভাবে এটা কোড করতে হয় সেটা তোমরা নিজেরাই একটু চিন্তা করলে বের করতে পারবে। জিনিসটা কঠিন না। তোমরা ইনপুটে দেওয়া সকল x, y আর z coordinate গুলি লিস্ট করবে। এরপর তাদেরকে সর্ট করে unique করে নেবে। মনে কর x এর লিস্ট দেখতে এরকমঃ 10, 30, 40. তাহলে তোমরা বাক্সে গিয়ে coordinate গুলিকে rename করবে। 10 মানে 1, 30 মানে 2, 40 মানে 3 এরকম। আর ওদিকে লিখে রাখবে 1 মানে আসলে 10, 2 মানে আসলে 30 ইত্যাদি। এরপর ছোট গ্রিডে তুমি BFS করে সকল তরল বের করবে। আর যখন তুমি volume বা area এর হিসাব করবা তখন তুমি যেই সেলের জন্য হিসাব করছ তাকে real coordinate এ পরিবর্তন করে নেবে।

UVa 11853 Paintball

সমস্যা: একটি 1000×1000 সাইজের মাঠ আছে। তুমি এর বাম বাহুর এক বিন্দু হতে ডান বাহুর এক বিন্দুতে যাবে। মাঠের বিভিন্ন জায়গায় তোমার বন্ধু জলকামান নিয়ে প্রস্তুত আছে। এসব বন্ধুর (x, y) coordinate দেওয়া আছে আর বলা আছে তারা কতদূর পর্যন্ত পানি ছুড়তে পারে (radius). বলতে হবে তুমি নিজেকে তোমার সকল বন্ধুর কাছ থেকে রক্ষা করে মাঠের বাম হতে ডানে যেতে পারবে কিনা। সর্বোচ্চ 1000 জন বন্ধু থাকবে।

সমাধান: বেশ কিছুক্ষণ চিন্তা ভাবনা করলে বুঝতে পারবে যে বাম দিক হতে ডান দিকে যাওয়া সম্ভব হবে যদি "নিচ হতে উপরে connected circle এর গুচ্ছ" না থাকে। তুমি circle গুলিকে একটি কলাম (থাম্বা) হিসাবে চিন্তা কর। যদি মাঠের নিচ হতে উপরে এই কলাম গুলি থাকে তার মানে তুমি কিছুতেই বাম থেকে ডানে যেতে পারবে না তাই না? কিন্তু যদি এরকম কলামের দেওয়াল না থাকে? তাহলে যাওয়া যাবে। কেন? তুমি মনে কর বাম দিকে থেকে কিছু তরল ঢাললে। তাহলে কি তারা ফাঁক ফোঁকর দিয়ে ডান দিকে চলে যাবে না? যাবে। সুতরাং আমাদেরকে বের করতে হবে নিচ হতে উপরে connected circle আছে কিনা। এটা তো করা খুব সহজ তাই না? তুমি BFS করবে। শুরুতে queue তে সেসব circle ঢুকাবে যারা আয়তের নিচের প্রান্তকে ছেদ বা স্পর্শ করে। এরপর তুমি queue হতে একটি বৃত্ত উঠাবে আর তার adjacent যেসব বৃত্ত এখনও visit হয় নি তাদেরকে queue তে প্রবেশ করাবে। দুটি বৃত্ত adjacent হবে যদি তারা ছেদ করে বা স্পর্শ করে। এরকম করতে করতে যদি এমন একটি বৃত্ত পাও যেটা আয়তের উপরের বাহুকে স্পর্শ বা ছেদ করে তার মানে আমাদের পক্ষে বাম হতে ডানে যাওয়া সম্ভব না।

UVa 10129 Play on Words

সমস্যা: তোমাকে সর্বোচ্চ 100,000 টি ইংরেজি শব্দ দেওয়া থাকবে। তোমাকে বলতে হবে তুমি এই সকল শব্দ পাশাপাশি এক লাইনে রাখতে পারবা কিনা যাতে পাশাপাশি থাকা দুটি শব্দের ক্ষেত্রে আগেরটার শেষ অক্ষর আর পরেরটার শুরুর অক্ষর একই হয়।

সমাধান: প্রথমত খেয়াল কর কোনো একটি শব্দের প্রথম আর শেষ অক্ষর বাদে বাকি অক্ষরের কোনো কাজ নেই। সুতরাং আমরা প্রতিটি শব্দকে দুটি অক্ষরের কল্পনা করতে পারি (এক অক্ষরের হলে কি করবা তা আশা করি নিজেরা বের করতে পারবা)। এখন আমরা এখান থেকে একটা directed গ্রাফ বানাব। গ্রাফের নোড হবে ইংরেজি ভাষার অক্ষর গুলি। আর edge হবে প্রতিটি শব্দের প্রথম অক্ষর হতে শেষ অক্ষর এর দিকে directed edge. এবার চিন্তা কর তো আমাদের উত্তর আছে মানে কি? মানে হল- একটি নোড হতে আরম্ভ করে প্রতিটি edge একবার করে traverse করে কোনো একটি নোডে শেষ হওয়া। অর্থাৎ directed গ্রাফে euler path আছে কিনা সেটা জিজ্ঞাসা করছে। আর এই কাজ কীভাবে করতে হয় তাতো আমরা প্রোগ্রামিং কন্টেস্ট বই এ দেখেছি।

UVa 1572 Self Assembly

সমস্যা: তোমাকে বিভিন্ন ধরনের বর্গ দেওয়া আছে। প্রতি ধরনের বর্গ অসংখ্য সংখ্যক আছে। প্রতি ধরনের বর্গের প্রতিটি বাহুতে লেবেল আছে। লেবেলগুলি দেখতে এরকম হয়- B+, A+, A- ইত্যাদি। অর্থাৎ দুই লেংথের লেবেল, প্রথম character টি হয় একটি ইংরেজি অক্ষর আর পরেরটি হয় + বা -. আরেকটি বিশেষ ধরনের লেবেল আছে 00. এখন তুমি দুটি বর্গকে পাশাপাশি রাখতে পারবা যদি adjacent বাহু দুটিতে একই ইংরেজি অক্ষর থাকে কিন্তু ভিন্ন sign (+ আর -) থাকে। যেমন তুমি A+ আর A- কে পাশাপাশি রাখতে পারবা। 00 এর পাশে কাউকেই রাখতে পারবা না। তুমি চাইলে বর্গগুলিকে ঘুরাতে পার বা reflect করতে পার। বলতে হবে তুমি এই সকল বর্গ ব্যবহার করে unbounded structure বানাতে পারবা কিনা। unbounded structure মানে তুমি বর্গ জোড়া লাগাতেই থাকবা অর্থাৎ জোড়া লাগানো চাইলে কখনই থামাতে পারবা না।

সমাধান: শুরুতে unbounded structure জিনিসটা দেখে একটু ভয় লাগতে পারে। কিন্তু ওতটা কঠিন না কিন্তু। তোমার যেই infinite structure সেটা এরকম হবে- প্রথম বর্গের কোনো একটি বাহু এর পাশে দ্বিতীয় বর্গ, এই দ্বিতীয় বর্গের কোনো একটি বাহুর পাশে তৃতীয় বর্গ এরকম করে চলতে থাকবে। আর অন্য কোনো বর্গ কিন্তু ব্যাপার না। এই আইডিয়া ব্যবহার করে এখন তুমি নানা উপায়ে এই সমস্যা সমাধান করতে পার। একটি হতে পারে- আমাদের গ্রাফের state (বা নোড) হবে কোন বর্গে আমরা আছি আর এর কোন বাহুতে। এখান থেকে আমরা এই বর্গের যেকোনো বাহু হয়ে তার সাথে খাপ খায় এরকম অন্য বর্গের বাহুর state এ যেতে পারি (হয়তো নিজের মত একটি বর্গের কোনো বাহু দিয়েও ঢুকতে পারি)। সুতরাং আমরা এই দুই state এর ভেতরে একটি directed edge দিতে পারি। এরকম সকল state এর ভেতরে আমরা directed edge দেব। যদি এই directed গ্রাফে cycle থাকে তার মানে আমরা unbounded structure বানাতে পারব।

UVa 1395 Slim Span

সমস্যা: একটি weighted graph দেওয়া আছে। গ্রাফে সর্বোচ্চ 100 টি নোড থাকবে। তোমাকে এমন একটি spanning tree বের করতে হবে যার সবচেয়ে বেশি cost এর edge এবং সবচেয়ে কম cost এর edge এর cost এর পার্থক্য যেন সবচেয়ে কম হয়।

সমাধান: খুব একটা কঠিন সমস্যা না। তুমি যদি ঠিক কর যে তোমার সবচেয়ে কম cost এর edge এর cost কত (ধর b) তাহলে বাকিটুকু তো সহজ তাই না? তুমি কেবল মাত্র সেসব edge ই নিবে যাদের cost কমপক্ষে b এর সমান। এবং এদের মাঝে তুমি Minimum Spanning Tree (MST) করবে। এরপর MST এর সবচেয়ে বড় cost এর edge আর b এর পার্থক্য পরীক্ষা করে দেখবে। এই

সমাধানে আমাদেরকে প্রতিটি edge এর উপর লুপ চালিয়ে b ঠিক করতে হতে পারে। সুতরাং সেই লুপ হবে 100^2 এর, কারণ 100 টি নোডের গ্রাফে প্রায় 100^2 টি edge থাকতে পারে। এরপর আমাদের বাকি edge গুলির উপর MST করতে হবে। আর MST এর complexity হল $\mathrm{O}(m \log m)$ যদি kruskal কর। সুতরাং আমাদের complexity দাঁড়াচ্ছে প্রায় $\mathrm{O}(100^4 \log 100^2)$. এই সমস্যার ক্ষেত্রে এই সমাধানই accepted হয়।

তবে তোমরা চাইলে এর থেকে একটু ভাল সমাধানও দাঁড় করাতে পার। মনে কর তুমি edge গুলিকে ছোট হতে বড়তে সর্ট করলে। এরপর তুমি একে একে edge গুলিকে (ছোট হতে বড় cost এর ক্রমে) আমাদের গ্রাফে insert করার চেষ্টা করবে। কোনো একটি edge কে insert করার আগে দেখবে তার দুই মাথা ইতোমধ্যেই connected কিনা। যদি না হয় তাহলে চোখ বন্ধ করে এই edge কে insert কর। আর যদি দুই মাথা ইতোমধ্যেই connected হয়ে থাকে তাহলে মনে কর এই দুই মাথা হল a আর b. তুমি দেখ a আর b এর মাঝে যেই path সেই path এ সবচেয়ে কম cost এর edge কোনটি। তুমি তাকে graph থেকে মুছে ফেল আর তোমার নতুন edge কে insert কর। বর্তমান edge যদি insert করা শেষ হয় তাহলে দেখ এখন গ্রাফে সবচেয়ে বড় cost আর সবচেয়ে কম cost এর edge কোন কোনটি। তাদের উপর নির্ভর করে তুমি তোমার উত্তর আপডেট করে ফেল। তবে খেয়াল রেখ তোমাদের গ্রাফে যেন $n-1$ টি edge থাকে, তা না হলে গ্রাফটি connected হবে না। আমাদের এই সমাধানে কোনো এক মুহূর্তে আমাদের গ্রাফে $n-1$ টির বেশি edge থাকবে না। সুতরাং n^2 টি edge কে insert করতে গেলে তোমাকে একটি BFS করতে হবে এবং তার cost আসলে $\mathrm{O}(n)$. তাহলে আমাদের মোট complexity হবে $\mathrm{O}(n^3)$.

UVa 1653 Yet Another Multiple Problem

সমস্যা: একটি সংখ্যা n ($n \leq 10^4$) এবং কিছু ডিজিট দেওয়া আছে। কেবল মাত্র এই সকল ডিজিট ব্যবহার করে তোমাকে n এর সবচেয়ে ছোট multiple বানাতে হবে।

সমাধান: এই সমস্যা বিভিন্ন কারণে আমার বেশ পছন্দের। প্রথমত তোমরা দেখবে যে- যদিও আপাত দৃষ্টিতে এখানে কোনো নোড নেই, কোনো edge নেই, কিন্তু এর সমাধান হবে BFS দিয়ে। দ্বিতীয়ত তোমরা দেখবে কখন DP ব্যবহার না করে BFS ব্যবহার করতে হয়।

প্রথমে একটু DP এর মত করে চিন্তা করে দেখি। DP করতে গেলে আমাদের স্টেট এর প্রয়োজন হয়। এই সমস্যার ক্ষেত্রে স্টেট হল, এখন mod কত (অর্থাৎ n দ্বারা ভাগ করলে ভাগশেষ কত)। ধরা যাক আমরা এখন m mod এ আছি। এখন আমরা যদি d ডিজিট ব্যবহার করতে চাই তাহলে পরের স্টেট হবে $(10m+d) \pmod n$. এভাবে আমরা দেখব যে 0 তে পৌঁছান যায় কিনা। প্রশ্ন হল এই পদ্ধতি কেন কাজ করবে না?

কাজ না করার মূল কারণ হল এক্ষেত্রে DP তে সাইকেল তৈরি হয়। মনে কর আমাদের $n = 2$. এখন আমরা শুরুতে 0 তে আছি। এখন আমরা 1 ব্যবহার করলাম করে $(10 * 0 + 1) \pmod 2$ এ গেলাম অর্থাৎ আমরা recursively 1 এর জন্য উত্তর জানতে চাচ্ছি। এখানে আমরা আবার 1 ব্যবহার করতে পারি এবং সেক্ষেত্রে আমরা আবার 1 এ যাব। খেয়াল কর, আমরা 1 এ ছিলাম আবার 1 এ এসেছি। অর্থাৎ DP করলে আমাদের একটা সাইকেলে পরে যাবে। সাইকেলের দৈর্ঘ্য যে সবসময় 1 হবে তা নয়। চাইলে বেশিও হতে পারে। সুতরাং DP করা যাবে না। এই ক্ষেত্রে BFS করতে হয়। আগের মতই আমাদের state (বা নোড) হল mod n এর মান। আর edge গুলি হল ডিজিট। অর্থাৎ একটা edge ব্যবহার করা মানে একটি ডিজিট বাড়তি লাগানো। আমরা 0 state হতে শুরু করব। আমাদের লক্ষ্য হল সবচেয়ে কম edge (সবচেয়ে কম সংখ্যক ডিজিট) ব্যবহার করে 0 তে পৌঁছান (যদি একাধিক থাকে তাহলে lexicographically সবচেয়ে ছোট উপায় বের করা)। এখানে আমরা 0 থেকে শুরু করে আবার 0 তে পৌঁছানোর রাস্তা যে খুঁজতেছি সেটা নিয়ে আবার দ্বন্দে পরে যেও না। চাইলে তোমরা শুরুর 0 কে আলাদা একটা বিশেষ state ধরে নিতে পার। আমাদের লক্ষ্য হল সেই বিশেষ state হতে mod 0 state এ পৌঁছান। যাই হোক। এখন কীভাবে আমরা BFS করব? খুব একটা কঠিন না। আমরা প্রতিটি state হতে সকল digit ব্যবহার করব এবং বের করব সেই ডিজিট ব্যবহার করে পরে কোন state এ যাওয়া যায় (দুইটি নোডের মাঝে edge). আমরা যদি BFS করি তাহলেই সবচেয়ে কম edge ব্যবহার করে আমরা mod 0 তে পৌঁছাতে পারব অর্থাৎ সবচেয়ে কম

ডিজিট ব্যবহার করে 0 তে পৌঁছাতে পারব। আর যদি আমরা কোনো নোড থেকে যেসকল edge বের হয় সেসকল edge কে ছোট হতে বড় order এ চেক করি তাহলেই আমরা lexicographically ছোট পাথ পেয়ে যাব।

DP দিয়ে যে করা যাবেই না তা না, তবে সেক্ষেত্রে DP তে mod এর পাশাপাশি length এর আরেকটি প্যারামিটার রাখতে হয়, ফলে এর state এর পরিমাণ বেড়ে যায়।

UVa 10603 Fill

সমস্যা: তিনটি পানির পাত্র আছে যাদের আয়তন a, b এবং c. এরা প্রত্যেকে খুব জোড় 200 হতে পারে। শুরুতে প্রথম দুটি পাত্র ফাঁকা আর তৃতীয় পাত্রটি ভরা। তুমি চাইলে এক পাত্র থেকে আরেক পাত্রে পানি ঢালতে পার যতক্ষণ না যেই পাত্র থেকে ঢালছ সেটি ফাঁকা হয়ে যায় বা যেই পাত্রে ঢালছ সেটি ভরে না যায়। তোমাকে বলতে হবে যে এরকম ঢালাঢালি করে তুমি d আয়তনের পানি আলাদা করতে পারবা কিনা। যদি না পার তাহলে বলতে হবে d এর কাছাকাছি কিন্তু তার থেকে ছোট কোন পরিমাণ তুমি আলাদা করতে পারবে।

সমাধান: এই সমস্যাটাও DP দিয়ে হবে না। তোমাকে BFS করতে হবে। মনে কর তুমি তৃতীয় পাত্র থেকে প্রথম পাত্রে পানি ঢাললে এবং DP কে recursively জিজ্ঞাসা করলে এই state হতে d পাওয়া যায় কিনা। সেখানে গিয়ে কিন্তু তুমি আবার প্রথম পাত্র থেকে তৃতীয় পাত্রে পানি ঢেলে জিজ্ঞাসা করবে যে এই স্থান হতে d পাওয়া যায় কিনা। দেখ সাইকেলে পরে গেছে তোমার DP. সুতরাং এটা DP দিয়ে হবে না। তোমাকে BFS করতে হবে বা চাইলে DFS ও করতে পার। আমাদের state হল কোন পাত্রে কি পরিমাণ পানি আছে। প্রতিটি state হতে তুমি ছয় রকমের কাজ করতে পার। প্রথম/দ্বিতীয়/তৃতীয় পাত্র থেকে অপর দুই পাত্রে পানি ঢালা। এই ঢালার পর তুমি নতুন state এ চলে আসবে। এভাবে তুমি BFS বা DFS করে যত state এ যাওয়া সম্ভব সব state ভিজিট করবে। সবশেষে যেসকল state ভিজিট হবে সেগুলতে গিয়ে গিয়ে দেখতে হবে যে d পরিমাণ আলাদা করা যায় কিনা, আর না গেলে d এর কাছাকাছি কোন পরিমাণ আলাদা করা যায়।

এখানে একটা জিনিস লক্ষ্য করার। সেটা হল- এখানে আমরা $\mathrm{O}(200^3)$ সাইজের state এর উপর BFS করছি। আমাদের state হবে (i, j, k) যেখানে i, j আর k হল তিনটি পাত্রে পানির পরিমান। কিন্তু খেয়াল কর, আমরা যদি জানি প্রথম দুই পাত্রে কি পরিমাণ পানি আছে তাহলে কিন্তু আমরা খুব সহজেই বের করতে পারব তৃতীয় পাত্রে কি পরিমাণ পানি আছে, কারণ তিনটি পাত্রে সবসময় মোট c পরিমাণ পানি থাকবে। অর্থাৎ $k = c - i - j$. সুতরাং এই observation খাটিয়ে আমরা আমাদের state $\mathrm{O}(200^2)$ এ নামাতে পারি।

UVa 10048 Audiophobia

সমস্যা: একটি 100 নোডের weighted গ্রাফ দেওয়া আছে। সেই সাথে সর্বোচ্চ 10,000 টি query দেওয়া থাকবে। প্রতিটি query তে দুটি নোড দেওয়া হবে। এই দুটি নোডের মধ্যে এমন একটি পাথ বের করতে হবে যাতে থাকা edge গুলির মাঝে সবচেয়ে বেশি cost যেন সবচেয়ে কম হয়।

সমাধান: প্রথমে আমরা দেখি যে একটি query কীভাবে সমাধান করা যায়। মনে কর আমাদের query হল s থেকে t তে যেতে হবে। খুব একটা কঠিন না। আমরা dijkstra এর মত করে s হতে t তে shortest path বের করব। তবে shortest path এর ক্ষেত্রে dijkstra তে আমরা কীভাবে augment করি তা একটু চিন্তা কর। মনে কর আমরা এখন u তে আছি আর $u - v$ edge বরাবর augment করার চেষ্টা করছি। ধরা যাক edge এর cost w. তাহলে আমরা দেখি যে $d[v]$ কি $d[u] + w$ এর থেকে বড় না ছোট। যদি বড় হয় তাহলে আমরা $d[v]$ এর মানকে এই নতুন মান দিয়ে update করি। এরকম করে u এর সকল adjacent edge এর জন্য update শেষ হয়ে গেলে আমরা আমাদের priority queue হতে সবচেয়ে কম $d[u]$ ওয়ালা u কে তুলি। ঠিক একই ভাবে আমরা dijkstra এর মত করে এই সমস্যা সমাধান করতে পারি। পার্থক্য হল $d[v]$ কে $d[u] + w$ এর

সাথে তুলনা বা আপডেট না করে আমরা $max(d[u], w)$ দিয়ে তুলনা বা আপডেট করব। তাহলেই হয়ে যাবে। এই সমাধানের complexity হবে dijkstra এর মত অর্থাৎ $\mathrm{O}(m \log n)$ এর মত। কেউ কেউ একে prim এর মতও মনে করতে পার।

আমরা চাইলে একটু সহজ ভাবেও সমাধান করতে পারতাম। কিছুক্ষণ আগে বললাম উপরের সমাধানটা কিছুটা prim এর মত। আমরা চাইলে kruskal এর মতও সমাধান দাঁড় করাতে পারি। প্রথমেই edge গুলিকে তার cost অনুযায়ী সর্ট করে ফেল। এর পর ছোট হতে বড় ক্রমে একে একে edge গুলিকে নাও। প্রতিটা edge নিবা আর তার দুই প্রান্তকে merge করবা। এরপর দেখবা যে s আর t কি connected হয়ে গেল কিনা। যদি হয়ে যায় তাহলে শেষ যেই edge ব্যবহার করলা তার cost ই হল উত্তর। আমার মনে হয় priority queue ব্যবহার করে prim এর মত কোড করার থেকে union find ব্যবহার করে kruskal এর মত কোড করা তুলনামুলক সহজ।

কিন্তু এটা তো বুঝাই যাচ্ছে যে উপরের দুটি সমাধান 10,000 query এর জন্য বার বার করা সম্ভব না। TLE হয়ে যাবে। উপায় কি?

এবার আমরা তিনটি সমাধান দেখব। প্রথমে এভাবে চিন্তা করতে পার- আমাদের যদি বহুবার query করে দুটি নোড এর মাঝে shortest পাথ বের করতে বলে তাহলে কি করব? একটি উপায় হল আমরা শুরুতেই প্রতিটি নোড হতে dijkstra চালায়ে প্রতিটি নোড হতে অপর সকল নোডের shortest path বের করে রাখব। একবার dijkstra চালাতে খরচ হয় $\mathrm{O}(m \log n)$. তাহলে n বার চালাতে খরচ হবে $\mathrm{O}(nm \log n)$. একই ভাবে আমরা আমাদের উপরের modified dijkstra সমাধানটা শুরুতেই n বার চালালে প্রতিটি নোড জোড়ার মাঝের জন্য উত্তর বের হয়ে যাবে।

দ্বিতীয় উপায় হল- আমাদের all pair shortest path বের করার জন্য একটি বিশেষ algorithm ছিল- floyd warshall. সেটাকে এই সমস্যার জন্য আমরা modify করতে পারি। এটা একটু চিন্তা করলেই পারবে। Floyd warshall এর ক্ষেত্রে আমরা $d[u][v]$ এর সাথে $d[u][k] + d[k][v]$ এর তুলনা করতাম। এই সমস্যার ক্ষেত্রে $d[u][v]$ এর সাথে $max(d[u][k], d[k][v])$ তুলনা করব। খুবই সহজ। Complexity হবে $\mathrm{O}(n^3)$.

তৃতীয় উপায় তুলনামুলক কঠিন। কিন্তু আশা করি এই সমাধানের টেকনিক তোমাদের অন্য কোনো সমস্যায় কাজে লাগবে। প্রথম দুটি সমাধানে আমরা উপরের dijkstra বা prim এর মত সমাধানকে আপগ্রেড করেছি। আমরা কি কোনো ভাবে kruskal এর মত সমাধানকে আপগ্রেড করতে পারি? এই চিন্তা থেকেই এই সমাধান। আগের মতই আমরা edge এর cost গুলিকে সর্ট করব। এরপর একে একে ছোট হতে বড় edge গুলি একে একে প্রসেস করব। প্রথমে দেখব যে বর্তমান edge এর দুই প্রান্ত ইতোমধ্যেই connected কিনা। যদি হয় তাহলে তো কিছু করার নেই। আর যদি না হয় তার মানে এই edge এর দুই প্রান্ত দুটি connected component এ আছে। ধরলাম এই দুটি component হল A আর B. তাহলে A তে থাকা নোড এবং B তে থাকা নোড এই দুটি নোডের জন্য উত্তর হবে বর্তমান edge এর cost. এই সমাধানের complexity এবার হিসাব করা যাক। আমাদের এরকম merge অপারেশন কিন্তু $n - 1$ বার এর বেশি হবে না (MST এর প্রতিটি edge এর জন্য)। Worst case এ এই merge অপারেশন সময় নেবে $\mathrm{O}(n^2)$ (যদি দুটি component এ $n/2$ এর মত করে নোড থাকে)। যেহেতু আমরা এই কাজ n বার এর মত করব সেহেতু আমাদের মোট complexity হবে $\mathrm{O}(n^3)$.

তোমরা চাইলে $\mathrm{O}(n \log n)$ এ preprocess করে $\mathrm{O}(\log n)$ এ প্রতি query উত্তর করতে পার। এজন্য তোমাদেরকে kruskal এর মত কাজ করে প্রাপ্ত tree এর উপর Least Common Ancestor (LCA) টেকনিক খাটাতে হবে। সংক্ষেপে বলি। Kruskal এর algorithm খাটিয়ে আমরা প্রথমেই tree বানিয়ে ফেলব। এরপর প্রতিটি নোড v এর জন্য আমরা 2^i তম parent বের করে রাখব। ঠিক LCA যেভাবে করা হয়। পার্থক্য হল, আমরা এখন কেবল মাত্র 2^i তম parent বের করব না, বরং ঠিক একই ভাবে 2^i তম parent পর্যন্ত পাথের উপরের সবচেয়ে বড় cost এর edge এর cost ও বের করব। এরফলে আমরা যেভাবে $\mathrm{O}(\log n)$ এ LCA বের করতে পারি, ঠিক একই ভাবে আমরা ট্রিতে দুটি নোডের মাঝে সবচেয়ে বড় cost এর edge এর cost ও বের করতে পারি।

UVa 11671 Sign of a Matrix

সমস্যা: শুরুতে একটি $n \times n$ সাইজের সকল element 0 ওয়ালা একটি ম্যাট্রিক্স থাকবে ($n \leq 100$)। তুমি এক অপারেশনে কোনো একটি কলাম বা কোনো একটি রো এর সকল element কে এক বাড়াতে বা কমাতে পার। তুমি এক বা একাধিক অপারেশন চালাতে পার। এখন ইনপুট হিসাবে তোমাকে একটি $n \times n$ সাইজের ম্যাট্রিক্স দেওয়া থাকবে যার প্রতিটি element হবে 0, + বা -. 0 মানে হল তুমি সকল অপারেশন চালানর পর ঐ জায়গার সংখ্যা 0, + মানে হল সকল অপারেশনের পর ঐ স্থানের সংখ্যা ধনাত্মক আর - মানে ঐ জায়গার সংখ্যা ঋণাত্মক। বলতে হবে সবচেয়ে কম কয়টি অপারেশন করে তুমি ইনপুটে দেওয়া ম্যাট্রিক্সের মত একটি ম্যাট্রিক্স বানাতে পারবে।

সমাধান: ক্ল্যাসিকাল একটি সমস্যা। প্রথমে চিন্তা করে দেখ কোনো একটি রো তে কি একত্রে +1 আর -1 অপারেশনের কোনো দরকার আছে? নাই। মনে কর তুমি কোনো একটি রোতে 10 বার +1 করেছ আর 3 বার -1. এর থেকে কি 7 বার +1 করা ভাল না? এরমানে প্রতিটি রো/কলামে আমরা হয় শুধু যোগ করব নাহয় শুধু বিয়োগ করব। এখন প্রতিটি রো আর কলামে একটি করে সংখ্যা লিখি ধর সংখ্যাটি হতে পারে +10, -5, +4, 0 ইত্যাদি। এর মানে তো বুঝছ? ঐ রো বা কলামে +1 বা -1 অপারেশনগুলি কতবার করে চালানো হয়েছে তা। যেহেতু ঐ সংখ্যা আমরা জানি না, আমরা এদেরকে একটি ভ্যারিয়েবল দ্বারা নির্দেশ করতে পারি। যেমন i তম কলামের সংখ্যা C_i আর i তম রো এর সংখ্যা R_i. এই ভ্যারিয়েবল গুলির সাপেক্ষে আমরা ইনপুট কে কীভাবে লিখতে পারি? মনে কর (i, j) একটি সেল। এখানে 0 আছে। এর মানে $R_i + C_j = 0$. যদি সেখানে + থাকে এর মানে $R_i + C_j > 0$ আর যদি - থাকে তার মানে $R_i + C_j < 0$. তাহলে আমাদের সমস্যা দাঁড়াচ্ছে যে- কিছু equation দেওয়া আছে, বলতে হবে এর সমাধান আছে কিনা। থাকলে $\sum abs(R_i) + abs(C_i)$ এর সর্বনিম্ন মান কত।

এখন আমরা একটা নতুন টেকনিক শিখব। মনে কর আমাদের কিছু equation দেওয়া আছে $d_i + w[i][j] \geq d_j$ যেখানে d_i গুলি হল ভ্যারিয়েবল আর $w[i][j]$ হল constant. এরকম মনে কর $d_1 \ldots d_n$ ভ্যারিয়েবল ব্যবহার করে কিছু inequality দেওয়া আছে। আমাদেরকে বলতে হবে এর সমাধান আছে কিনা, থাকলে এমন একটি সমাধান বের করতে হবে যেখানে $\sum abs(d_i)$ এর মান সর্বনিম্ন হয়।

খেয়াল কর, আমরা যদি এই নতুন সমস্যা সমাধান করতে পারি তাহলে আমাদের মূল সমস্যাটা সমাধান হয়ে যাবে। কীভাবে? খুব সহজ, আমরা ধরব $d_1 = R_1, \ldots d_n = R_n$ আর $d_{n+1} = -C_1 \ldots d_{2n} = -C_n$. তাহলে আমরা একে একে আমাদের equation গুলি দেখি $R_i + C_j = 0$ কে আমরা লিখতে পারি $d_i - d_{n+j} \geq 0$ এবং $d_i - d_{n+j} \leq 0$ যা $d_i \geq d_j + w[i][j]$ এর মত তাই না? অন্যগুলি দেখি, $R_i + C_j > 0$ কে আমরা লিখতে পারি $d_i - d_{j+n} \geq 1$. একই ভাবে তৃতীয় equation কেও আমরা আমাদের কাঙ্খিত ফর্মে লিখতে পারি। এর অর্থ দাঁড়াল আমরা যদি আমাদের এই নতুন সমস্যা সমাধান করতে পারি তাহলেই মূল সমস্যা সমাধান হয়ে যাবে। কথা হল এই নতুন সমস্যা কীভাবে সমাধান করব।

এক কথায় বলা যায় আমাদের এই নতুন সমস্যাকে আমরা shortest path সমস্যায় কনভার্ট করতে পারি। কীভাবে? এই কীভাবেটা আমার মনে থাকে না। আমি কীভাবে চিন্তা করি খেয়াল করতে পার-

মনে কর u হতে v তে w cost এর একটি edge আছে। ধরা যাক কোনো একটি সোর্স থেকে এই গ্রাফে তুমি প্রতিটি নোডে shortest path বের করেছ। ঐ সোর্স হতে u পর্যন্ত shortest path হল $d[u]$, একই ভাবে v পর্যন্ত shortest path $d[v]$. তাহলে কিন্তু $d[u] + w[u][v] \geq d[v]$ হবে। কেন $d[u] + w_{uv} < d[v]$ হতে পারবে না? কারণ এর মানে হল তুমি যদি সোর্স হতে u তে $d[u]$ cost এ আসো এর পর u হতে v এর $w[u][v]$ cost এর edge দিয়ে v তে যাও তাহলে তোমার cost হবে $d[u] + w[u][v]$ যা equation মোতাবেক $d[v]$ হতে কম। তাতো সম্ভব না কারণ $d[v]$ হল সোর্স হতে v পর্যন্ত shortest path. সুতরাং $d[u] + w[u][v] \geq d[v]$ হবে।

সুতরাং আমরা যেটা করব তাহল, প্রত্যেক $d[u] + w[u][v] \geq d[v]$ equation এর জন্য u হতে v তে $w[u][v]$ cost এর একটি directed edge দেব। এরপর আমরা একটি সোর্স নেব এবং তা হতে সকল নোডে 0 cost এর edge দেব। এরপর আমরা এই সোর্স হতে সকল নোডের shortest path বের করব। যেহেতু $w[u][v]$ negative হতে পারে সেহেতু আমাদেরকে bellman

ford অ্যালগরিদম চালাতে হবে। যদি দেখি আমাদের গ্রাফে negative cycle আছে তার মানে এই সমস্যার কোনো সমাধান নেই। আর যদি negative cycle না থাকে তাহলে এই shortest path অ্যালগরিদম তোমাকে optimal $d[u]$ এর মান দেবে। অর্থাৎ এমন সব $d[u]$ এর মান দেবে যেন $\sum abs(d[u])$ এর মান যেন সবচেয়ে ছোট হয়। কেন? প্রমাণ জানতে চাইলে তোমরা Coreman এর বই দেখতে পার।

LOJ 1099 Not the Best

সমস্যা: N নোডের একটি গ্রাফ আছে ($N \leq 5000$) যাতে সর্বোচ্চ 10^5 টি undirected weighted edge থাকতে পারে। তোমাকে 1 হতে N এ যাবার second shortest path বের করতে হবে। Second shortest path হল সেই shortest path যার cost একদম shortest path অপেক্ষা বেশি হয়। তুমি চাইলে একটি edge একাধিক বার ব্যবহার করতে পার।

সমাধান: খুব একটা কঠিন সমস্যা না। তুমি প্রতিটি নোডের জন্য দুটি cost রাখবে। একটু হল shortest path এর cost ধরা যাক $d_1[i]$ আর আরেকটি হবে second shortest path এর cost ধরা যাক $d_2[i]$. এরপর তুমি স্বাভাবিক ভাবে dijkstra বা যেকোনো shortest path অ্যালগরিদম চালিয়ে $d_2[N]$ বের করবে। শুরুতে $d_1[1] = 0$ হবে। এরপর একে তুমি priority queue তে ঢুকাবে, অর্থাৎ কোন নোড আর d_1 নাকি d_2. এরপর আমরা priority queue হতে একে একে state তুলব। এরপর এই নোডের adjacent edge গুলি দেখব। ধরা যাক আমরা এখন যেই state এ আছি তার cost d আর আমরা এখন যেই edge দেখছি তার cost w, আর এই edge এর ওপর মাথা v. তাহলে আমরা $d + w$ কে $d_1[v]$ আর $d_2[v]$ এর সাথে তুলনা করব। যদি $d + w$ এর মান $d_1[v]$ এবং $d_2[v]$ এর থেকে কম হয় তাহলে $d_2[v] = d_1[v]$ করব আর $d_1[v] = d + w$ করব। যদি তা না হয়, তাহলে দেখব $d_1[v] < d + w < d_2[v]$ কিনা। তাহলে $d_2[v] = d + w$ করব। মানে আমরা v এর second shortest path কে আপডেট করব। এভাবে যখন priority queue যখন ফাঁকা হয়ে যাবে তখন আমরা $d_2[N]$ আউটপুট করব। এখানে কিছু details আমি বাদ দিয়েছি, যেমন শুরুতে d_1 ও d_2 কে infinity দিয়ে initialize করে দিতে হবে, কোনো একটি $d_1[v]$ বা $d_2[v]$ আপডেট হলে তাকে priority queue তে ঢুকাতে হবে ইত্যাদি।

LOJ 1034 Hit the Light Switches

সমস্যা: একটি N নোডের ($N \leq 10000$) directed গ্রাফ দেওয়া আছে যাতে সর্বোচ্চ 10^5 টি edge থাকতে পারে। গ্রাফের প্রতিটি নোড শুরুতে সাদা রঙের। তুমি যদি কোনো নোডকে কালো রঙ কর তাহলে তা থেকে যত গুলি directed edge বের হয় তার ওপর প্রান্তও কালো হয়ে যাবে, এরপর তাদের থেকে যেসব directed edge বের হয় তারাও কালো হবে এভাবে চলতে থাকবে। তোমাকে বলতে হবে তুমি সবচেয়ে কম কতগুলি নোড কালো করলে পুরো গ্রাফকে কালো করে ফেলতে পারবে।

সমাধান: এভাবে চিন্তা কর- তুমি যেকোনো একটি নোড নিলে। যদি দেখ এতে প্রবেশ করে এরকম যদি একটি edge থাকে তাহলে ভাব যে- থাক একে আর রঙ না করি, এতে তো একটি directed edge প্রবেশ করেছে, সুতরাং তার ওপর প্রান্ত যখন কালো হবে তখন এও কালো হয়ে যাবে। তাহলে তুমি কালো রঙ করবে কাকে? যাতে কোনো directed edge প্রবেশ করে না অর্থাৎ যার indegree 0. শুধু তাদের কালো রঙ করলেই কি হবে? না। একটা খুব সহজ উদাহরন নাও। মনে কর 2 নোডের একটি সাইকেল। A থেকে B তে edge আর B থেকে A তে edge. এক্ষেত্রে কিন্তু তোমাকে একটি নোডকে রঙ করতেই হবে। তাহলে উপায় কি? উপায় হল, তোমাকে Strong Connected Component বের করতে হবে। এরপর একই SCC তে থাকা নোডগুলিকে collapse করে ফেলতে হবে। তাহলে আমরা এমন একটি directed গ্রাফ পাব যাতে কোনো সাইকেল থাকবে না। এবার আমরা আগের পদ্ধতিতে ফিরে যাব। দেখব এই গ্রাফে কতগুলি নোডের indegree 0. ততগুলি নোডকে কালো করলেই হবে। অর্থাৎ সেসব নোড যেসব component কে represent করে সেসব component থেকে যেকোনো একটি নোডকে কালো রঙ করলেই হবে।

LOJ 1221 Travel Company

সমস্যা: একটি N নোডের ($N \leq 100$) directed গ্রাফ দেওয়া আছে যাতে সর্বোচ্চ 9900 টি edge থাকতে পারে। প্রতিটি edge এর দুটি করে weight আছে। একটি হল income আর আরেকটি হল expense. তুমি যদি একটি নোড হতে আরম্ভ করে কিছু edge ঘুরে আবার শুরুর নোডে ফেরত আসো এবং তোমার যদি মোট expense যদি হয় E আর মোট income যদি হয় I তাহলে profit ratio হবে I/E. তোমাকে বলতে হবে এমন একটি সাইকেল পাওয়া সম্ভব কিনা যার profit ratio এর মানে P অপেক্ষা বড়।

সমাধান: মনে কর এরকম একটি সাইকেল খুঁজে পাওয়া সম্ভব। এর মানে সেই সাইকেলের জন্য আমরা লিখতে পারি $\sum I / \sum E > P$ বা $\sum I - P * E > 0$ বা $\sum P * E - I < 0$. অর্থাৎ আমরা যদি প্রতিটি edge এর cost কে $P * E - I$ আকারে লিখি এবং এই গ্রাফে যদি আমরা একটা negative cycle খুঁজে পাই তার মানে আমরা এমন একটি সাইকেল খুঁজে পেতে পারি যার profit ratio এর মান P অপেক্ষা বড়। এখন আমরা যেকোনো negative cycle finding algorithm ব্যবহার করে এই সমস্যা সমাধান করতে পারি।

UVa 753 A Plug for UNIX

সমস্যা: তোমার কাছে 100 টির মত device আছে এবং একটি ঘরে 100 টির মত প্লাগ আছে। একটি দোকানে কিছু converter আছে যা এক প্লাগের ধরনকে আরেক প্লাগে কনভার্ট করে। সব ধরনের কনভার্টার নেই, তবে যেগুলো আছে তা অনেক পরিমাণে আছে। কোন ডিভাইস এর কিরকম প্লাগ দরকার তা বলা আছে। তুমি চাইলে সেই ডিভাইসকে সেই ধরনের প্লাগে লাগাতে পার অথবা এক বা একাধিক কনভার্টার ব্যবহার করে অন্য প্লাগে লাগাতে পার। বলতে হবে সর্বোচ্চ কতগুলি ডিভাইস তুমি একত্রে প্লাগে লাগাতে পারবে।

সমাধান: একটি প্লাগে একাধিক ডিভাইস লাগাতে পারবা না, সর্বোচ্চ কতগুলি ডিভাইস লাগানো সম্ভব- ইত্যাদি হল max flow এর হিন্ট। Maxflow সমস্যার ক্ষেত্রে কিছু কমন টার্ম হল- "এতোটির বেশি ব্যবহার করতে পারবা না" (capacity), "সর্বোচ্চ কতটি" (maximum flow), আর জিনিসগুলির মাঝে একটা general সম্পর্ক বজায় থাকে, মানে এই সমস্যার ক্ষেত্রে খেয়াল কর এক ধরনের কনভার্টার তুমি একাধিক বার একাধিক ডিভাইসের জন্য ব্যবহার করতে পার- এটা একটা general সম্পর্ক। DP সমস্যার ক্ষেত্রে সাধারনত এরকম many to many সম্পর্ক থাকে না। যাই হোক এগুলো অবশ্য সেন্স এর ব্যাপার। যত বেশি সমস্যা সমাধান করবা, তোমার এই সকল সেন্স তত পরিস্কার হবে।

এই সমস্যার ক্ষেত্রে ফ্লো এর ডিজাইন কীভাবে করবা? কয়েকটা জিনিস পরিস্কার। আমাদের সোর্স এর সাথে থাকবে ডিভাইস আর সিঙ্ক এর সাথে থাকবে প্লাগ। এই সকল edge এর ক্যাপাসিটি হবে 1. এখন কথা হল ডিভাইস, প্লাগ আর কনভার্টার এর মাঝে edge গুলি কীভাবে দিবা। প্রথমে খেয়াল কর, ডিভাইস আর প্লাগের মাঝের edge গুলি কিন্তু obvious. যেই ডিভাইস যেই প্লাগের সাথে compatible তাদের মাঝে একটি edge দিতে হবে। edge এর capacity আসলে ব্যাপার না। তুমি চাইলে 1 দিতে পার আবার infinity ও দিতে পার। কারণ source থেকে তো একটি ডিভাইসে একটির বেশি ফ্লো আসবে না। এখন কথা হল যদি ডিভাইসটি সরাসরি প্লাগে না গিয়ে কনভার্টারে যায়? একটি উপায় হল, যেই প্লাগ থেকে অন্য যে প্লাগে কনভার্টার আছে তাদের মাঝে edge দেওয়া। এর মানে হবে, কোনো একটি ডিভাইস এই টাইপের প্লাগে লাগিয়েছি, এখন এই প্লাগ চাইলে আরেক প্লাগে ঢুকান যাবে অথবা চাইলে কারেন্টে লাগানো যাবে (তাহলে আমাদের প্রশ্ন হল- সর্বোচ্চ কতগুলি ডিভাইস কারেন্টে লাগানো যাবে)। এই edge এর ক্যাপাসিটি কত হবে? Infinity. কেন? কারণ আমরা তো একই ধরনের কনভার্টার একাধিক বার ব্যবহার করতে পারি তাই না? এই সমস্যায় Infinity হিসাবে কত মান আমরা ব্যবহার করতে পারি? যেহেতু আমাদের ডিভাইস সংখ্যা 100 টি, সেহেতু কোনো কনভার্টার 100 এর বেশি বার ব্যবহার হবে না। সুতরাং infinity এর মান 100 এর বেশি বা সমান হলেই হবে। এই গ্রাফের maxflow ই হবে উত্তর।

UVa 11082 Matrix Decompressing

সমস্যা: একটি ম্যাট্রিক্স দেওয়া আছে। ম্যাট্রিক্স এর সর্বোচ্চ ডাইমেনশন কোনো একদিকে 20 হতে পারে (অর্থাৎ 20×20 হল সবচেয়ে বড় ম্যাট্রিক্স)। তোমাকে এই ম্যাট্রিক্সের প্রতিটি রো এবং কলামের সংখ্যার যোগফল দেওয়া আছে। তোমাকে ম্যাট্রিক্সটি প্রিন্ট করতে হবে। ম্যাট্রিক্স এর প্রতি ঘরে 1 হতে 20 পর্যন্ত সংখ্যা বসতে পারে।

সমাধান: এটা যে ফ্লো এর সমস্যা সেটা বুঝা ওতটা সহজ না। কিন্তু একবার দেখলে আশা করি এর পর থেকে এরকম সমস্যা সমাধান করতে পারবা। যেহেতু প্রতিটি ঘরে কমপক্ষে 1 দিতেই হবে সেহেতু আমরা প্রতিটি ঘরে 1 দিয়ে দেই। সেই অনুসারে আমরা প্রতিটি রো এবং প্রতিটি কলামের যোগফলকে adjust করে নেই। এখন আসা যাক কীভাবে গ্রাফ বানাব সে বিষয়ে। এই সমস্যার ক্ষেত্রে আমাদের সোর্স এর সাথে রোগুলির edge থাকবে আর কলাম গুলির সাথে সিঙ্ক এর edge থাকবে। edge গুলির capacity হবে ঐ রো বা কলামের সংখ্যাগুলির যোগফল। আর প্রতিটি রো থেকে প্রতিটি কলামে edge থাকবে। সেই edge এর capacity হবে 19 (কারণ প্রতিটি সেলে সর্বোচ্চ 20 থাকতে পারে আর 1 তো ইতোমধ্যেই দিয়ে ফেলেছি)। এই গ্রাফে ফ্লো চালালেই আমাদের উত্তর পাওয়া যাবে। এখানে কি হচ্ছে খেয়াল কর, প্রতিটি রো এর যেই যোগফল তা সোর্স থেকে বের হয়ে সেই রো এর বিভিন্ন সেলের ভেতর দিয়ে বিভিন্ন কলামে যাচ্ছে। সোর্স আর রো এর মধ্যকার capacity এটা নিশ্চিত করছে যে সেই রো এর সংখ্যা গুলির যোগফল যেন আমাদের প্রদত্ত যোগফলকে অতিক্রম না করে। একই ভাবে কলামের যোগফলও আমরা আমাদের capacity দিয়ে বাউন্ড করছি। প্রতিটি রো আর কলামের মাঝের 19 capacity এর edge দিয়ে আমরা নিশ্চিত করছি যে কোনো ঘরে যেন 19 এর বেশি সংখ্যা না যায়। আর আমরা maxflow চালিয়ে নিশ্চিত করছি যেন প্রতিটি সোর্স-রো edge গুলি saturated হয়। খেয়াল কর, যদি এরকম হয় যে এমন একটি সোর্স-রো edge আছে যা saturated হয় নায়, তার মানে আমাদের এরকম কোনো ম্যাট্রিক্স পাওয়া সম্ভব না। তবে এই সমস্যার ক্ষেত্রে বলা আছে যে উত্তর সবসময় সম্ভব। সুতরাং এই সমস্যায় আমাদের এটা চেক করার দরকার নেই।

UVa 1349 Optimal Bus Route Design

সমস্যা: একটি গ্রাফ দেওয়া আছে যেখানে সর্বোচ্চ 99 টি vertex থাকতে পারে। গ্রাফটি directed এবং weighted. তোমাকে কিছু সাইকেল বের করতে হবে যেন প্রতিটি নোড দিয়ে ঠিক একটি সাইকেল যায়। তোমার লক্ষ্য হল এই সাইকেলগুলির cost এর যোগফল যেন সবচেয়ে কম হয়। সাইকেলগুলিকে কমপক্ষে 2 length এর হতে হবে।

সমাধান: বেশ ট্রিকি সমস্যা। আগে থেকে এধরনের সমস্যা সমাধান না করলে পারা কঠিন। প্রতিটি নোডের দুটি কপি কর। প্রথম কপিকে সোর্স এর সাথে edge দিতে হবে, আর দ্বিতীয় কপিকে সিঙ্ক এর সাথে। এদের প্রতিটির capacity 1 আর cost 0. এখন মনে কর মূল গ্রাফে একটি edge আছে যেটা u আর v এর মাঝে আর তার cost হল w. এর জন্য আমাদেরকে যা করতে হবে তাহল u এর প্রথম কপি হতে v এর দ্বিতীয় কপিতে w cost এর এবং 1 capacity এর একটি edge দিতে হবে। এখন যদি তুমি mincost maxflow চালাও তাহলেই উত্তর পেয়ে যাবে (যদি flow এর পরিমাণ n এর থেকে কম হয়, তাহলে কোনো সমাধান নেই)। প্রশ্ন হল কেন? খেয়াল কর maxflow তে কখনই এমন অবস্থা হবে না যে- সোর্স হতে u এর প্রথম কপি, এর পর u এর দ্বিতীয় কপি এবং সবশেষে সিঙ্ক। কারণ আমাদের দেওয়া edge গুলির u এবং v সবসময় আলাদা হবে (যদি একই হয় তাহলে সেই edge কে ফেলে দিলেই হল, কারণ আমাদের উত্তরে তার কোনো প্রভাব নেই)। তাহলে একটু যদি খেয়াল কর দেখবে, আমাদের ফ্লো দেখতে এরকম হবে- সোর্স হতে a এর প্রথম কপি, এর পর b এর দ্বিতীয় কপি এরপর সিঙ্ক (এখানে a আর b আলাদা)। আমরা একে (a, b) লিখতে পারি। এভাবে আমরা n টি pair লিখতে পারি যেখানে এদের প্রথম element সবসময় আলাদা, আর দ্বিতীয় element ও সবসময় আলাদা। কারণ আমাদের সোর্স হতে প্রথম কপিতে capacity 1, একই ভাবে দ্বিতীয় কপি হতে সিঙ্ক এ capacity 1. এখন আমরা যদি এই pair গুলিকে ভাল ভাবে দেখি তাহলে কিছু সাইকেল দেখতে পাব। কেন? মনে কর আমাদের একটি pair (a, b), এরপর আমরা b দিয়ে শুরু এরকম একটি pair নিব, ধরা যাক (b, c), এরপর (c, d) এরকম করে সবশেষে হবে (d, a).

অর্থাৎ আমরা a থেকে শুরু করেছিলাম a তেই শেষ করেছি। মানে সাইকেল। এরকম এক বা একাধিক সাইকেল গঠন হবে n টি pair দিয়ে। যেহেতু কোনো (u, u) মার্কা pair নাই সেহেতু সাইকেল গুলি কমপক্ষে 2 দৈর্ঘ্যের হবেই। আর যেহেতু আমরা "mincost" maxflow করছি সেহেতু এই সাইকেল গুলির cost এর যোগফল সবচেয়ে কম হবে।

UVa 1515 Pool Construction

সমস্যা: একটি 2d গ্রিড দেওয়া আছে। গ্রিডের প্রতিটি সেলে হয় # থাকবে নাহয় . থাকবে। # মানে হল সেখানে ঘাস আছে আর . থাকার মানে হল সেখানে পানি আছে। তুমি চাইলে d টাকা খরচ করে কোনো একটা ঘাসের সেল খুঁড়ে সেখানে গর্ত করে পানি ঢালতে পার। আবার চাইলে f টাকা খরচ করে কোনো একটা পানির সেলকে মাটি দিয়ে ভরাট করে সেখানে ঘাস লাগাতে পার। সবশেষে যদি দুটি পানি আর ঘাসের দুটি সেল পাশাপাশি থাকে তাহলে তাদের মাঝে ব্যারা দিতে হবে এবং এর জন্য খরচ হবে b টাকা। গ্রিডের বাউন্ডারি এর কোনো সেল পানির সেল হতে পারবে না। বলতে হবে কত কম খরচে তুমি এই কাজ করতে পারবে। গ্রিডের সাইজ সর্বোচ্চ 50×50 আর d, b, f এর সর্বোচ্চ মান 10,000 এর মত।

সমাধান: এটিও বেশ ট্রিকি সমস্যা। আগে থেকে এ ধরনের সমস্যা সমাধান না করলে পারা কঠিন। এই ধরনের সমস্যা সমাধান হয় mincut ব্যবহার করে। Mincut সমস্যায় একটি weighted undirected গ্রাফ দেওয়া থাকে। গ্রাফে দুটি নোড s আর t ও দেওয়া থাকবে। সমস্যা হল তুমি এই গ্রাফ থেকে কিছু কিছু edge কে মুছে ফেলবে যেন s আর t এই দুটি নোড disconnected হয়। তোমার লক্ষ্য হল মুছে ফেলা edge গুলির cost এর যোগফল যেন সবচেয়ে কম হয়। এই সমস্যার সমাধান আর s হতে t তে flow (এখানে গ্রাফের weight গুলি edge এর capacity হিসাবে ব্যবহার হবে) এর পরিমাণ একই হবে। যদি কারও আগ্রহ থাকে তাহলে maxflow mincut এর theory গুলি পড়ে দেখতে পার।

এখন কথা হল এই সমস্যা কীভাবে সমাধান করব। এইসব ক্ষেত্রে যেটা হয় আমরা দুইটা নোড s আর t নেই। মনে কর s হল পানি এর নোড আর t হল ঘাসের নোড। তুমি mincut করার পর s এর সাথে যুক্ত নোড গুলি পানি দ্বারা ভরাট থাকবে আর t এর সাথে যুক্ত নোডগুলি ঘাস দ্বারা ভরাট থাকবে। তাহলে এখন একটু চিন্তা করে দেখত যে s আর t বাদে আর কিসের কিসের নোড দরকার? সহজ, প্রতিটি সেলের জন্য একটি করে নোড দরকার। তাহলে এই গেল আমাদের নোড। এখন চিন্তা করতে হবে কীভাবে edge দিবা বা edge গুলির capacity বা weight কত হবে। প্রথমত s আর t বাদে সেলের যেসকল নোড আছে তাদের মাঝে যদি দুটি সেল পাশাপাশি হয় তাহলে তাদের মাঝে edge দেওয়া দরকার। কারণ যদি এই দুটি সেলের একটি যদি পানি হয় আরেকটা যদি ঘাস হয় তাহলে তো s-t কাটে তারা আলাদা হবে এবং এই আলাদা করার জন্য আমাদের b cost হবে তাই না (বাউন্ডারি এর জন্য)? সুতরাং যেসকল সেল পাশাপাশি আছে তাদের নোড এর মাঝে b cost এর edge হবে। আর যারা পাশাপাশি নাই? তাদের মাঝে আসলে কোনো edge দেওয়া লাগবে না। কারণ তারা কি পানি নাকি ঘাস এটা নিয়ে আসলে কোনো মাথা ব্যথা নাই। শুধু পাশাপাশি পানি আর ঘাস থাকলে তাদের মাঝে বাউন্ডারি দিতে হবে এজন্য এই b cost এর edge দেওয়া। অর্থাৎ মনে কর i আর j নোড দুটি পাশাপাশি। তাহলে আমরা i হতে j তে এবং j হতে i তে b cost এর একটি edge দেব। এখন চিন্তা কর s এর সাথে সেলের নোডগুলির edge এর কথা। মনে কর x একটি সেল। আমরা চিন্তা করব s এর সাথে x এর edge থাকা না থাকার মানে কি বা থাকলে তার cost এর মানে কি। আমাদের মূল লক্ষ্য হল s-t কাট বের করা। যদি এই কাটে s আর x দুই পক্ষে থাকে এর মানে হল x হবে ঘাস (আগেই বলেছি s হল পানি বা cut এর পর s এর সাথে যুক্ত সকল নোড হবে পানি)। এর মানে s-x এখানে একটি edge দিতে হবে এবং সেই edge এর cost হবে x কে ঘাস বানানোর cost. যদি x আগে থেকেই ঘাস থাকে তাহলে তো এই cost 0. আর যদি আগে থেকে পানি থাকে তাহলে এই cost হবে f. কেন এরকম হল? কারণ s-t কাটের সময় যদি s-x আলাদা হয় তাহলে তো s-x এর মাঝের edge তো কাটা পরবে, এর মানে x হয়ে যাবে ঘাস (যেহেতু আগেই বলেছি t এর সাথে যুক্ত সব কিছু ঘাস) । এর জন্য কত cost হবে? x কে ঘাস বানাতে যত cost, তাই না? সেটাই হল s-x এর cost. একই ভাবে আমরা x-t এর মাঝে edge দিতে পারি এবং তার cost হবে

x কে পানি বানানোর cost. আর বাকি থাকল "বাউন্ডারি এর কোনো সেল পানি হতে পারবে না"। এর জন্য আমাদের যা করতে হবে তাহল শুরুতেই বাউন্ডারি এর সকল পানির সেলকে f খরচ দিয়ে ঘাসে পরিনত করতে হবে। এরপর আমরা এই সকল সেলকে t এর সাথে infinity cost এর একটি edge দিয়ে দেব। আগেই বলেছি t হল ঘাসের নোড। আর আমরা চাইনা এসকল নোড t হতে বিচ্ছিন্ন হোক। তাই আমরা বললাম যদি বিচ্ছিন্ন করতেই হয় তাহলে infinity cost দাও। এর মানেই হল mincut এ এরা কখনও আলাদা হবে না, তাই না? কারণ এদের আলাদা না করে s হতে বাকি সব নোডকে আলাদা করা লাভ জনক (অর্থাৎ সকল নোডকে f খরচে ঘাস বানিয়ে ফেলা)।

তাহলে আমাদের সকল নোড আর সকল edge বানানো শেষ। এখন আমাদের কাজ s হতে t তে ফ্লো চালানো। Maxflow এর মানই হবে আমাদের mincut এর মান।

UVa 10735 Euler Circuit

সমস্যা: একটি 100 নোডের গ্রাফ দেওয়া আছে। গ্রাফে edge গুলির মাঝে কিছু directed আর কিছু undirected. তোমাকে undirected edge গুলিতে direction দিতে হবে যেন গ্রাফটিতে euler circuit থাকে। বলতে হবে এরকম করা সম্ভব কিনা হলে তোমাকে euler circuit প্রিন্ট করতে হবে।

সমাধান: একটি directed গ্রাফে euler circuit থাকার শর্ত হল গ্রাফটিকে connected হতে হবে (মানে সকল directed edge কে undirected মনে করে) আর প্রতিটি নোডের indegree আর outdegree সমান হতে হবে। প্রথম condition চেক করা তো খুব সহজ। প্রশ্ন হল দ্বিতীয়টি করা সম্ভব কিনা।

Euler circuit থাকার শর্ত হল তার indegree আর outdegree এর সংখ্যা সমান হবে। আমরা জানি প্রতিটি নোডে কয়টি edge লেগে আছে। সুতরাং সেই নোডে কয়টি directed edge প্রবেশ করতে হবে আর কয়টি edge বের হতে হবে তা আমরা হিসাব করে বের করে ফেলতে পারি। প্রশ্ন হল প্রতিটি নোডের এই demand পূরণ করা সম্ভব কিনা। মনে কর আমরা গ্রাফের নোড গুলিকে বাম পাশে রাখব আর গ্রাফের edge গুলিকে flow network এর ডান পাশে। প্রদত্ত গ্রাফের প্রতিটি undirected edge এর জন্য যেসব নোড out-degree হিসাবে কাজ করতে পারে সেসব নোড আর সেই undirected edge এর মাঝে আমাদের flow network এ edge দেই। এর মানে কোনো একটি undirected edge এর জন্য তার edge থাকবে তার দুই প্রান্তের নোডের সাথেই। ডান দিকের প্রতিটি edge কেই তার adjacent কোনো একটি নোডের সাথে match করাতে হবে। ঐ দিকে বামদিকের প্রতিটি নোডের জন্য আমরা জানি কয়টি অতিরিক্ত edge কে out-degree হিসাবে পেতে হবে। অতিরিক্ত বললাম কারণ ইতিমধ্যেই যেসব directed edge আছে তাদের আর indegree বা outdegree কত হতে হবে সেই সংখ্যা গুলি নিয়ে হিসাব করলেই বের হয়ে আসবে যে প্রতিটি নোডের আর কয়টি অতিরিক্ত outdegree দরকার। এটা হল প্রতিটি নোডের চাহিদা। সুতরাং আমরা এখানে একটা flow network বানায়ে ফেলতে পারি। সোর্স হতে বাম দিকের প্রতিটি নোডে তাদের চাহিদা মোতাবেক capacity সহ edge দেব আর ওদিকে ডান দিকের edge এর নোড গুলির জন্য সিঙ্ক এর সাথে 1 capacity ওয়ালা edge দেব (কারণ প্রতিটি edge কেবল একটি out-degree দিতে পারে)। এই ফ্লো গ্রাফ সমাধান করলেই আমরা পেয়ে যাব কোন edge এর direction কিরকম হবে। এটুকু জেনে গেলে আমরা euler circuit বের করার অ্যালগরিদম ব্যবহার করে বাকি টুকু করে ফেলতে পারব।

UVa 1659 Help Little Laura

সমস্যা: 2D তে কিছু বিন্দু আছে। এদের সংখ্যা সর্বোচ্চ 100 হতে পারে। এদের কারও কারও মাঝে directed edge আছে। সেই সাথে দুটি সংখ্যা X এবং Y দেওয়া আছে। তুমি একটি কলম নিয়ে তোমার ইচ্ছা মত কোনো বিন্দু হতে শুরু করে কিছু directed edge ঘুরে আবার শুরুর বিন্দুতে ফিরে আসতে পার। এভাবে তুমি যতবার খুশি করতে পার। চাইলে একবারও না করতে পার। তবে কোনো edge তুমি একাধিক বার ব্যবহার করতে পারবে না। এখন এর ফলে মনে কর তুমি d দুরত্ব অতিক্রম

করলে আর c টি edge ভ্রমন করেছ। তাহলে তোমার cost হবে $d \times X - c \times Y$. তোমার লক্ষ্য হল cost কে সর্বোচ্চ করা।

সমাধান: Maxflow এর সমস্যাগুল যে ত্যাঁদড় মার্কা হয় তা আশা করি ইতমধ্যে বুঝে গিয়েছ। কই থেকে কীভাবে এটা যে flow হয়ে যায় তা ধরতে পারা বেশ কঠিন। অনেক experience লাগে।

আগের সমস্যার মত এই সমস্যাতে একটা বড় observation আছে, আর তাহল প্রতিটি নোডের indegree আর outdegree সমান হতে হবে। এই property ব্যবহার করে কীভাবে সমাধান করা যায়?

প্রথমত প্রতিটি edge এর জন্য তার cost বের কর। কোনো একটি edge এর যদি দৈর্ঘ্য হয় d তাহলে তার cost হবে $dX - Y$. আমাদের cost কিন্তু সবচেয়ে বেশি করতে বলেছে। সুতরাং আমরা প্রথমেই যেসকল edge এর cost ধনাত্মক তাদের নিয়ে ফেলব। মনে কর এসব edge নেবার ফলে কোনো একটি নোডের indegree হয়েছে 5 আর outdegree হয়েছে 7. এর মানে কি? এর মানে হল আমাদেরকে হয় দুটি outdegree বাদ দিতে হবে অথবা দুটি indegree বাড়াতে হবে। আসলে ঠিক তা না, কারণ আমরা যদি তিনটি outdegree কমাই আর একটি indegree কমাই তাহলেও হবে। মূল কথা হল আমাদেরকে indegree আর outdegree সমান করতে হবে। আগেই তো দেখেছ এধরনের সমস্যাকে কীভাবে approach করতে হয়। প্রথমে প্রতিটি নোডের জন্য দেখ এতে indegree বেশি নাকি outdegree. ধর indegree বেশি, এর মানে এতে কিছু অতিরিক্ত indegree আছে যা আমাদের কমাতে হবে অথবা কিছু outdegree বাড়াতে হবে। আপাতত indegree কমানোর কথা ভুলে যাও। মনে কর আমরা শুধু outdegree বাড়াবো। সেক্ষেত্রে যত বাড়াতে চাই তত সোর্স হতে ঐ নোডে capacity হিসাবে দিতে হবে। কারণ আমরা যখন ফ্লো চালাব তখন ঐ পরিমাণ ফ্লো কোনো না কোনো directed edge দিয়ে বের হবে ফলে outdegree বাড়বে। একইভাবে যদি কোনো নোডের outdegree যদি বেশি হয় এর মানে হয় outdegree কমাতে হবে অথবা indegree বাড়াতে হবে। আগের মতই আপাতত ভুলে যাই যে outdegree কমালেও হয়। যদি আমরা indegree বাড়াতে চাই তাহলে আমরা সেই নোড থেকে সিঙ্কে ঐ পরিমাণ capacity এর edge দেব তাহলেই হয়ে যাবে। আর শুরুতেই আমরা যেসকল edge নেই নাই (কারণ তাদের cost ছিল negative) তাদেরকে আমরা গ্রাফে নেব। তাদের capacity হবে 1 কারণ এসব edge কে তো আমরা একাধিক বার নিতে পারব না। আর cost? $dX - Y$ ছিল negative. আর আমাদের চেষ্টা করতে হবে যেন আমাদের শেষ score সর্বোচ্চ হয়। অর্থাৎ আমাদের কে এই flow তে নেওয়া edge গুলির cost কে maximize করতে হবে। বা অন্যভাবে এদের negative cost কে minimize করতে হবে। আমরা তো "mincost" maxflow চালাব, সুতরাং এসব edge এর cost দিতে হবে $-(dX - Y)$.

এখন বাকি থাকল ভুলে যাওয়া অংশ। অর্থাৎ কোনো নোডের indegree বেশি হলে তা কমানো যায় কীভাবে তা বের করা (একই ভাবে outdegree বেশি হলে তা কমানো)। খেয়াল কর, indegree কমানো মানে কিন্তু আগের নেওয়া একটি edge কে বাদ দেওয়া। ফ্লো এর মত কর চিন্তা কর- এর মানে হল আগে একটি ফ্লো ছিল সেটাকে cancel করা। তাহলে কি করবা? মনে কর u হতে v একটি edge ছিল যার cost ছিল c. মনে কর এই edge টি আমরা বাদ দিয়ে আমরা u এর outdegree কমাতে চাই। তুমি যেটা করবা তাহল v হতে u একটি edge দিবা যার cost হবে $-c$. কেন? আমরা যদি v হতে u edge এ ফ্লো দেই তাহলে এই u হতে v এর edge নেওয়াটা cancel হয়ে যায়। আর শুরুতেই আমরা কিন্তু c cost নিয়েছিলাম (কারণ $c > 0$) এখন আমরা যদি এই edge নেই তাহলে আমাদের যোগ হবে $-c$, এর মানে আগের cost টা কাটা যাচ্ছে। সুতরাং ব্যাপার খানা এমন, আমি outdegree কমাতে চাই- এই সমস্যাটা indegree বাড়াতে চাই এ convert হয়ে গিয়েছে। এভাবে তুমি mincost maxflow দিয়ে এই সমস্যা সমাধান করে ফেলতে পারবে।

এই সমস্যা সমাধানের আরেকটি উপায় হল negative cycle cancellation algorithm. তোমাদের ইচ্ছা থাকলে এটা নিয়ে একটু পড়া শুনা করতে পার। তবে মূল আইডিয়া হল, তুমি চাইছ তোমার score যত বেশি হবে তত ভাল। এই সমস্যাকে একটু পাল্টাতে হবে। অর্থাৎ আমাদের score যত কম হয় তত ভালতে পরিবর্তন করতে হবে। খুব সহজ, প্রতিটি edge এর cost কে -1 দিয়ে গুন করে দিলেই হবে। এরপর তুমি এই গ্রাফে negative cycle খুঁজে বের কর। এতে augment path কর, অর্থাৎ প্রতিটি edge কে উলটিয়ে দাও আর তাদের cost কে negate করে দাও। এরকম করে যতক্ষণ negative cycle খুঁজে পাবা এই প্রসেস চালাতে থাক। প্রতিবার তুমি তোমার negative cycle এর cost যদি যোগ করতে থাক তাহলে এই শেষ যোগফলটিই তোমার

উত্তর। এটিই negative cycle cancellation algorithm.

UVa 1664 Conquer a New Region

সমস্যা: একটি 200,000 নোডের ট্রি দেওয়া আছে। ট্রি এর প্রতিটি edge এর weight আছে। দুটি নোড u আর v এর মাঝের path এর cost হল- এই path এ যতগুলি edge আছে তাদের মাঝে সবচেয়ে কম weight যত তত। তোমাকে এই ট্রি এর সেন্টার বের করতে হবে। আমরা সেই নোডকে সেন্টার বলব যেই নোড থেকে অন্য সকল নোডের path এর cost এর যোগফল সর্বোচ্চ হয়। এই সর্বোচ্চ cost এর যোগফল আমাদের প্রিন্ট করতে হবে।

সমাধান: বেশ জটিল সমস্যা। বুঝা যায় কীভাবে সমাধান করতে হবে কিন্তু কোনো কারনে সমাধানটা সহজে ধরা দেয় না। বিভিন্ন ভাবে চিন্তা করতে পার। একটা উপায় হতে পারে DP এর মত করে চিন্তা কর। আরেকটা উপায় হতে পারে edge গুলোকে তাদের weight অনুসারে সর্ট করে (কিছুটা kruskal এর মত)। সর্ট করার পর আমরা edge গুলিকে একে একে add করতে হবে। এই ক্ষেত্রে সাধারনত union find ও ব্যবহার করা হয়ে থাকে এটা বুঝাতে যে যেসকল edge নেওয়া হয়েছে তারা কি কি connected component গঠন করে। কিন্তু এতো সবের পরও কোনো কারনে সমাধানটা ধরা দেয় না। তোমরা চাইলে পরের অংশটুকু পড়ার আগে নিজেরা চিন্তা করে দেখতে পার।

মনে কর আমাদের সবচেয়ে কম weight এর edge হল $u-v$ যার weight w. ধরা যাক u এর দিকে আছে $c(u)$ সংখ্যক নোড আর v এর দিকে আছে $c(v)$ সংখ্যক নোড। আমরা যদি আমাদের center u এর দিকে বানাই তাহলে এই edge এর মারফতে আমাদের cost হবে $c(v) * w$. আর যদি সেন্টারটা হয় v এর দিকে তাহলে এই edge এর জন্য আমাদের cost হবে $c(u) * w$. কেন বুঝতে পারছ তো? কারণ মনে কর তোমরা v এর দিকে center বানালে। আর w হল সবথেকে কম cost এর edge. তাহলে u এর দিকের যত নোড তাদের সবাইকে এই w cost এর edge পার হয়ে আসতে হবে। আর যেহেতু এর cost সবচেয়ে কম সুতরাং অন্য কোনো edge এই সকল নোডের cost কে প্রভাবিত করতে পারবে না। সবই তো বুঝলাম, কিন্তু এর পর? এরপরের টুকু চিন্তা করতে গেলেই কোনো কারনে সব গুবলেট পাকায়ে যায়। একটু চিন্তা কর। আমরা এখন চাইলে u এর দিক আর v এর দিক কিন্তু independently চিন্তা করতে পারি। u এর দিকে যেই center হোক না কেন তার cost এর সাথে $c(v) * w$ যোগ হবে আর ওদিকে v এর দিকে যেই center হোক না কেন তার cost এর সাথে $c(u) * w$ যোগ হবে। সুতরাং এভাবে চিন্তা করতে পার যে আমাদেরকে এখন এই দুটি ছোট ট্রি এর জন্য সমাধান করতে হবে। যদি আমাদের নোডের সংখ্যা কম হত তাহলে নাহয় আমরা $\mathrm{O}(n^2)$ এ সমাধান করে ফেলতাম। কিন্তু আমাদের n তো অনেক বড়। আমাদেরকে $\mathrm{O}(n^2)$ এ সমাধান করলে হবে না। তাহলে কি করা যায়?

এই পর্যায়ে এসে যেই জিনিসটা খেয়াল করতে হবে তাহল, যদি আমাদের ট্রি এর নোড সংখ্যা 1 হয় তাহলে আমরা তার উত্তর জানি। মনে কর আমাদের দুটা ট্রি আছে আর এদের দুজনের উত্তরও আমরা জানি। এখন এদের মাঝে একটি edge জোড়া লাগাচ্ছি। তাহলে এই বড় ট্রি এর উত্তর কি আমরা জানি? জানি। ঠিক আগের মত করে চিন্তা কর। মনে কর প্রথম ট্রি এর u এর সাথে দ্বিতীয় ট্রি এর v কে w weight এর একটি edge দিয়ে জোড়া লাগাচ্ছি, যেখানে w এর weight মনে কর সবচেয়ে কম। u এর দিকের ট্রি এর উত্তর মনে কর $ans[u]$ আর v এর দিকের ট্রি এর উত্তর মনে কর $ans[v]$. তাহলে এই বড় ট্রি এর উত্তর কত হবে? দুই রকম হতে পারে। আমাদের সেন্টারটা u এর দিকে, সেইখেত্রে আমাদেরে উত্তর হবে $ans[u] + c(v) * w$ তাই না ($c(u)$ এর মানে হল u এর tree এর সাইজ)? কারণ u এর দিকে কোথায় সেন্টার কিছুই যায় আসে না, এই নতুন w weight এর edge লাগানোর ফলে v এর দিকের $c(v)$ সংখ্যক নোড গুলিকে w পার হয়ে u এর দিকে যেতে হবে। u এর দিকে কোন সেন্টারে তাদের যেতে হবে সেটা কিন্তু কোনো ব্যাপার না। একই ভাবে ওদিকে যদি u এর দিকের নোড গুলি v এর দিকে আসত তাহলে আমাদের উত্তর হত $ans[v] + c(u) * w$. তাহলে তুমিই বল আমাদের বড় ট্রি এর জন্য cost কত হবে? এই দুটির মাঝে যেটি বড় সেটি তাই না (কারণ আমাদের লক্ষ্য cost এর যোগফল বেশি করা)? এর মানে আমাদেরকে আসলে edge গুলো বড় হতে ছোট অনুসারে সর্ট করে প্রসেস করতে হবে। এই edge গুলি একে একে নিবে। এদের জোড়া লাগাবে। জোড়া লাগানর ফলে যেই বড় ট্রি হল তার উত্তর বের করবে। এরকম করে চলতে থাকবে।

আর এই জিনিস union find দিয়ে খুব সহজেই করা সম্ভব।

৯.১ অনুশীলনী

৯.১.১ সমস্যা

Simple

- UVa 1151 Buy or Build
- UVa 1658 Admiral
- UVa 820 Internet Bandwidth
- UVa 208 Firetruck
- UVa 506 System Dependencies
- UVa 247 Calling Circles
- UVa 821 Page Hopping
- UVa 10305 Ordering Tasks
- UVa 439 Knight Moves
- UVa 572 Oil Deposits

Easy

- UVa 1600 Patrol Robot
- UVa 10651 Pebble Solitaire
- UVa 1001 Say Cheese
- UVa 10801 Lift Hopping
- UVa 12264 Risk
- Uva 1025 A spy in the metro
- UVa 1103 Ancient Message
- UVa 810 A Dicey Problem
- UVa 1665 Islands
- UVa 658 Its not a Bug, its a feature!
- UVa 12661 Funny Car Racing
- UVa 1660 Cable TV Network
- UVa 1663 Purifying Machine
- UVa 12219 Common Subexpression Elimination
- UVa 1601 The Morning after Halloween
- UVa 804 Petri Net Simulation
- UVa 816 Abbott's Revenge
- UVa 10653 Bombs! No they are mines!!

Medium

- UVa 1279 Asteroid Rangers
- UVa 1667 Network Mess
- UVa 1669 Holiday's Accommodation
- UVa 1518 Train Delays
- LOJ 1208 Dangerous Bull! Who wants to Pull?
- UVa 12549 Sentry Robots
- UVa 1668 Let's Go Green
- UVa 1670 Kingdom Roadmap

অধ্যায় ১০

Adhoc

UVa 120 Stacks of Flapjacks

সমস্যা: একটি প্যানকেক এর স্ট্যাক দেওয়া আছে। স্ট্যাকের প্রতিটি প্যানকেকের সাইজ দেওয়া আছে। দুটি কেকের সাইজ এক হবে না। তুমি এক অপারেশনে স্ট্যাকের উপরের কিছু কেক নিয়ে তাদের উলটিয়ে আবার স্ট্যাকের উপরে রাখতে পারবে। তোমার লক্ষ্য হল সবার নিচে যেন সবচেয়ে বড় কেক, তার উপর তার থেকে ছোট এভাবে সবার উপরে যেন সবচেয়ে ছোট কেক থাকে। তোমাকে যে সবচেয়ে কম অপারেশনে এই কাজ করতে হবে তা না। যেকোনো valid sequence প্রিন্ট করলেই হবে। এই সমস্যায় সর্বোচ্চ 30 টি কেক থাকতে পারে।

সমাধান: খুব সহজ সমস্যা তাই না? প্রথমে দেখ সবচেয়ে বড় কেক কোথায় আছে? এটা কি সবার নিচে আছে? তাহলে তো এর জন্য কিছুই করতে হবে না। যদি সবার উপরে থাকে তাহলে তাকে আমরা এক অপারেশনে নিচে আনতে পারি তাই না? আর যদি মাঝা মাঝি কোথাও থাকে? তাহলেও সহজ তাকে এক অপারেশনে আমরা উপরে আনতে পারি আর আরেক অপারেশনে একদম নিচে আনতে পারি। এভাবে আমরা সবচেয়ে বড় কেক কে নিচে আনতে পারি। এটা হয়ে গেলে আমরা এই কেকের কথা ভুলে যেতে পারি। যেহেতু কেকের সংখ্যা খুব কম সেহেতু আমাদের এই সমাধানের complexity নিয়ে ভাব্বার দরকার নেই।

UVa 1605 Building for UN

সমস্যা: n দল আছে ($n \leq 50$). তোমাকে একটি যেকোনো সাইজের একটি 3d grid নিতে হবে। এই গ্রিডের প্রতিটি সেল তোমাকে কোনো না কোনো দলকে দিতে হবে। শর্ত দুইটা। এক- যেকোনো দুটি দল অবশ্যই কোথাও না কোথাও পাশাপাশি দুটি সেলে থাকবে। দুই- প্রতিটি দলের সেলগুলি নিজেদের ভেতরে connected হতে হবে। গ্রিডের সাইজ কত হবে তা তোমার ইচ্ছা। গ্রিডে সর্বোচ্চ 10^6 টি সেল থাকতে পারবে।

সমাধান: এধরনের সমস্যার নির্দিষ্ট কোনো টেকনিক নাই। Puzzle এর মত। একটি সমাধান হতে পারে দুই লেয়ারের গ্রিড বানানো। নিচের লেয়ারে প্রত্যেক রোতে একটি করে দল থাকবে। আর উপরের লেয়ারে প্রতিটি কলামে একটি করে দল থাকবে। নিজেদের লেয়ারে তো প্রতিটি দল connected. আবার প্রতিটি দলের দুই লেয়ারে যেই connected জায়গা আছে তাদের মাঝে connection থাকবেই। আবার প্রতি দুটি দলের মাঝেও connection আছে। সুতরাং এই সমাধান একটি সঠিক সমাধান।

UVa 1152 4 Values whose Sum is 0

সমস্যা: আমাদের কাছে 4 টি লিস্ট আছে। প্রতিটি লিস্টে n টি করে সংখ্যা আছে। n এর মান সর্বোচ্চ 4000 হতে পারবে। বলতে হবে তুমি কত ভাবে এই 4 লিস্ট হতে 4 টি সংখ্যা নির্বাচন করতে পারবে (প্রতিটি লিস্ট হতে একটি) যেন তাদের যোগফল 0 হয়।

সমাধান: খুব একটা কঠিন না। তুমি চিন্তা কর যদি 4 টা লিস্ট না থেকে 2 টি লিস্ট থাকত তাহলে কীভাবে সমাধান হত? তুমি প্রথম লিস্ট scan করতে। মনে কর x প্রথম লিস্টের একটি সংখ্যা। তাহলে তুমি দেখতে যে দ্বিতীয় লিস্টে $-x$ আছে কিনা। তুমি চাইলে map, set, hash ইত্যাদি ব্যবহার করতে পার। আবার চাইলে তুমি দুটি লিস্টকে sort করে two pointer মেথড খাটাতে পার। যাই হোক, এদের যেকোনো টির সমাধান $\mathrm{O}(n \log n)$ এর মত (hash করলে অবশ্য $\mathrm{O}(n)$).

আমাদের সমস্যায় চারটি লিস্ট আছে কিন্তু তারা খুব ছোট সাইজের, মাত্র 4000. তাহলে আমরা প্রথম দুটি লিস্ট হতে একটি লিস্ট বানায়ে ফেলতে পারি $\mathrm{O}(n^2)$ সময়ে। এই লিস্টে থাকবে ঐ দুটি লিস্টের যেকোনো দুটি element এর যোগফল। অর্থাৎ এই নতুন লিস্টের সাইজ হবে $\mathrm{O}(n^2)$. একই ভাবে শেষ দুটি লিস্ট নিয়ে আমরা আরেকটা লিস্ট বানাতে পারি। এরপর তো সহজ। কিছুক্ষণ আগে বলা দুই লিস্টের সমাধান আমরা এই দুই লিস্টে খাটাতে পারি।

UVa 11134 Fabled Rooks

সমস্যা: একটি $n \times n$ সাইজের দাবার বোর্ড আছে যেখানে n এর সর্বোচ্চ মান হতে পারে 5000. তুমি এই বোর্ডে n টি নৌকা বসাতে পারবে। তবে তুমি যেখানে সেখানে বসাতে পারবে না। প্রতিটি নৌকার জন্য তোমাকে একটি rectangle দেওয়া আছে। তোমাকে ঐ নৌকা ঐ rectangle এর ভেতরেই বসাতে হবে। শর্ত হল, কোনো দুটি নৌকা যেন একই রো বা একই কলামে না থাকে। বলতে হবে এভাবে নৌকা গুলি বসান সম্ভব কিনা, সম্ভব হলে বসায়ে দেখাতে হবে।

সমাধান: আমার অন্যতম পছন্দের একটি সমস্যা। আমি এখানে দুটি সমাধান বলব। তবে এর দ্বিতীয় সমাধানের জন্যই এই সমস্যা আমার বেশি প্রিয়।

প্রথম সমাধান হল- ফ্লো। তোমরা যারা বেশ কিছু ফ্লো এর সমস্যা সমাধান করেছ তারা হয়তো দেখেছ যে এরকম সমস্যা প্রায়ই flow বা bipartite matching দিয়ে সমাধান হয়। সেই experience থেকেই এই সমস্যা ফ্লো এর মত করে চিন্তা করা। আমাদের experience অনুসারে আমাদের বামে থাকবে রো আর ডানে থাকবে কলাম। সোর্স হতে বামের নোডগুলিতে 1 capacity এর edge থাকবে। একই ভাবে ডানের নোডগুলি হতে সিঙ্ক এ 1 capacity এর edge থাকবে। এখন কথা হল এই ডান আর বামের ভেতরে কীভাবে edge দেবে। এখানে যদি কোনো rectangle এর বাউন্ড না থাকত তাহলে তো খুব সহজ, বামের প্রতিটি হতে ডানের প্রতিটি নোডে আমরা edge দিতাম, মানে যেকোনো রো এর যেকোনো কলামে নৌকা বসতে পারবে। কিন্তু এই ক্ষেত্রে তো তা হবে না। আমাদের বাউন্ড আছে। আমার সমাধান হল এই বাম আর ডানের মাঝে প্রতিটি নৌকার জন্য 2 টি করে নোড দেব এবং তাদের মাঝে একটি edge থাকবে যার capacity হবে 1. এই দুটি নোড এবং তাদের মাঝের capacity এটা নিশ্চিত করবে যে- কোনো একটি আয়তের জন্য যেন একটিই নৌকা থাকে। এবার এই আয়ত যেই যেই রো আর যেই যেই কলাম জুড়ে থাকে তাদের মাঝে 1 capacity এর edge দিয়ে দাও তাহলেই হবে। চিন্তা করে দেখ আমাদের সমস্যার সকল constraint এই গ্রাফে আছে কিনা? প্রতিটি রো তে একটি করে নৌকা থাকতে হবে, প্রতিটি কলামে একটি করে নৌকা থাকতে হবে, প্রতিটি আয়তের মাঝে অন্তত একটি নৌকা থাকতে হবে ইত্যাদি। সুতরাং এই flow চালিয়ে যদি দেখ maxflow এর মান n তার মানে সমাধান আছে। নাহলে সমাধান নাই।

এখন দ্বিতীয় সমাধান দেখা যাক। এই সমস্যাতে খুব বড় powerful একটা observation হল আমরা দুটি axis কে independently সমাধান করতে পারি। অর্থাৎ আমরা আয়তের জন্য সমাধান করব না। বরং মনে করব প্রতিটি নৌকার জন্য সে কোন রো তে বসতে পারবে তার রেঞ্জ দেওয়া আছে। একই ভাবে সে কোন কলামে বসতে পারবে তারও রেঞ্জ দেওয়া আছে। তুমি দেখ যে প্রতিটি রোতে নৌকা বসাতে পারবা কিনা। একই ভাবে দেখ প্রতিটি নৌকার জন্য কলাম বের করতে পার

কিনা। যদি এদুটি কাজই পার তাহলে তো সমাধান হয়ে গেল। খেয়াল করে দেখ আমাদের সমস্যা কত সহজ হয়ে গেল! 2d একটি সমস্যা 1d তে পরিবর্তন হয়ে গিয়েছে। আমরা যদি রো এর জন্য সমাধান করতে পারি তাহলে একই সমাধান আমরা কলামের জন্য খাটাবো। আর আমার মনে হয় রো এর জন্য সমাধান কেমন হবে তা তোমরা জান, তাই না? Greedy. কিছুটা line sweep এর মতও বলতে পার। তুমি একে একে রো গুলিতে যাবে। প্রথমে প্রথম রো, এরপর দ্বিতীয় রো এরকম করে সকল রো তে যাবা। মনে কর তুমি এখন একটি রোতে আছ। এখন দেখ এই রো হতে কোন কোন নৌকার রেঞ্জ শুরু হয়। সেসকল রেঞ্জের শেষ মাথা একটা data structure এ রাখো। সুতরাং বর্তমানে data structure এ সেই সকল নৌকা আছে (মানে নৌকার রেঞ্জের শেষ মাথা) যাদেরকে তুমি বর্তমান রো তে বসাতে পার। কিন্তু কোন নৌকাকে বসাবা? যার শেষ মাথা সবচেয়ে ছোট, অর্থাৎ যার ভবিষ্যতে বসাতে সমস্যা হতে পারে, এরকম নৌকাকে তুমি বসাবা। অর্থাৎ Data structure কে বলবা তোমার কাছে সবচেয়ে ছোট সংখ্যা কত আছে তাকে দাও। এই নৌকাকে এই স্থানে বসাবা। তবে বসানোর আগে একবার দেখে নাও এর রেঞ্জের শেষ মাথা ইতোমধ্যেই পার করে এসেছ কিনা। তাহলে কিন্তু আর তুমি এই নৌকাকে বসাতে পারবা না, তার মানে সমাধানও নেই। তুমি এখানে data strcture হিসাবে map বা multiset ব্যবহার করতে পার।

UVa 11054 Wine trading in Gergovia

সমস্যা: এক লাইনে অনেক গুলি সেল। প্রতিটি সেলে তোমার বন্ধু আছে। প্রতিটি বন্ধু একটি জিনিসের কিছু ইউনিট করে চায় বা তার কাছে সেই জিনিসের কিছু ইউনিট অতিরিক্ত আছে। যেসব বন্ধু দের কাছে অতিরিক্ত জিনিস আছে তোমাকে তাদের কাছ থেকে জিনিস নিয়ে যারা চায় তাদেরকে দিতে হবে। এক ইউনিট জিনিস তুমি যদি i তম স্থানে থাকা বন্ধুর কাছ থেকে নিয়ে j তম স্থানের বন্ধুর কে দাও তাহলে এর জন্য cost হবে $abs(j-i)$. তোমাকে সর্বনিম্ন cost এ সবার চাহিদা মিটাতে হবে। তোমার মোট বন্ধুর সংখ্যা সর্বোচ্চ 100,000 হতে পারে। আর বলা আছে মোট চাহিদার পরিমান মোট অতিরিক্ত থাকার পরিমানের সমান।

সমাধান: যেহেতু চাহিদার পরিমান অতিরিক্ত থাকার পরিমানের সমান সেজন্য এই কাজ greedy ভাবে করলেই তুমি optimal সমাধান পাবে। Greedy মানে হল- তুমি মনে কর বাম থেকে ডান দিকে যাচ্ছ। যদি দেখ প্রথম জন এর কিছু চাহিদা আছে এর মানে এই পরিমান ডান দিক থেকে আসবে। আর যদি দেখ এর কিছু অতিরিক্ত আছে এর মানে এই পরিমান ডান দিকে যাবে। তাহলে কি পরিমান ডানে বা বামে যাবে তার উপর ভিত্তি করে তুমি তোমার উত্তরকে আপডেট করবা। এরপর দ্বিতীয় জনের কাছে যাবা। এবার একটু হিসাব নিকাশ করে দেখ প্রথম দুই জন আর বাদ বাকি সবার মাঝে কি পরিমান আদান প্রদান হবে। সেই মোতাবেক উত্তর আপডেট কর। এভাবে চলতে থাকবে। অর্থাৎ, এখানে কে কাকে দিল সেটা চিন্তা না করে আমাদেরকে চিন্তা করতে হবে প্রথম i জন আর বাদ বাকি জনের মাঝে কি পরিমান আদান প্রদান হবে। অর্থাৎ প্রতিটি edge এ কি পরিমান আদান প্রদান হবে।

যেমন মনে কর আমাদের চাহিদা (ঋণাত্মক) আর অতিরিক্ত (ধনাত্মক) এর পরিমান গুলি হল: $3, 1, -5, 1$. এখানে প্রথম জনের আছে 3. এর মানে বাম দিকে থেকে 3 টি জিনিস ডানে যাবে। এরপর দেখ প্রথম দুই জনের আছে 4 টি। এর মানে প্রথম দুই জন হতে পরের জনের কাছে যাবে 4 টি জিনিস। প্রথম তিন জনের কাছে আছে -1 টি জিনিস, অর্থাৎ প্রথম তিন জনের 1 টি জিনিসের চাহিদা আছে যা আসবে ডান দিক থেকে। আর প্রথম চারজনের চাহিদা/অতিরিক্ত আছে 0 টি। সুতরাং আর কিছু না। তাই এখানে মোট $3 + 4 + 1 = 8$ cost হয়েছে। এটিই সর্বনিম্ন।

UVa 11572 Unique Snowflakes

সমস্যা: একটি সংখ্যার অ্যারে দেওয়া আছে যার লেংথ সর্বোচ্চ 100,000. তোমাকে এই অ্যারের সবচেয়ে বড় সাবঅ্যারে খুঁজে বের করতে হবে যেন তাতে কোনো সংখ্যা দুইবার না থাকে।

সমাধান: বেশ ট্রিকি সমস্যা। প্রথমেই আমরা $last$ নামের একটি অ্যারে বের করব। $last[i]$ এর মান হবে অ্যারের i তম স্থানে যেই সংখ্যা আছে সেটি i index এর আগে কোন index এ আছে। যদি এর আগে কোথাও না থাকে তাহলে আমরা এর মান 0 দেব। $last$ এর অ্যারে বের করা কিন্তু খুব একটা কঠিন না। আমরা একটি map রেখে আমাদের ইনপুটের অ্যারেকে বাম হতে ডানে scan করলেই last অ্যারের মান গুলি পাব। মনে কর আমরা অ্যারের বাম হতে ডানে যাচ্ছি। প্রতিটি স্থানে এসে এই স্থানে থাকা সংখ্যাকে আমরা map এ খুঁজব। যদি না পাই তার মানে এর last এর মান 0. আর যদি পাই তাহলে map এ সেই সংখ্যার জন্য যেই মান আছে তাই হবে last এর মান। অতঃপর আমরা map এ এই সংখ্যার মান হিসাবে বর্তমান index লিখে রাখব। অর্থাৎ যেকোনো মুহূর্তে map এ থাকবে এর আগে অমুক সংখ্যা কোন index এ আছে। সুতরাং এভাবে আমরা last এর মান বের করতে পারি।

এখন চিন্তা করা যাক last এর অ্যারেকে ব্যবহার করে আমাদের সমস্যা কীভাবে সমাধান করতে পারি। দুটি variable নেয়া যাক l আর r যাদের initial মান 1. l মানে হল সাবঅ্যারের বাম প্রান্ত, আর r হল ডান প্রান্ত। এখন আমরা r এর মান এক করে বাড়াবো। বাড়ানোর পর বর্তমান r এর জন্য দেখব $last[r]$ এর মান কি বর্তমান l এর থেকে বড়? যদি বড় হয় তাহলে $l = last[r]$ করে দেব। মানে আমরা আমাদের সাবঅ্যারেকে ছোট করে ফেলছি। এরকম করে r কে আমরা একে একে 1 হতে n পর্যন্ত (আমরা ধরে নিচ্ছি অ্যারেটির index 1 হতে n) নেব এবং l এর মান প্রয়োজনে আপডেট করব। এই আপডেট করা শেষে আমরা দেখব সেই l আর r এর মান কি optimal কিনা (অর্থাৎ আমাদের দরকার সবচেয়ে বড় $r - l + 1$ এর মান)।

UVa 1606 Amphiphilic Carbon Molecules

সমস্যা: 2d তে সর্বোচ্চ 1000 টি বিন্দু আছে। প্রতিটি বিন্দু হয় সাদা নাহয় কালো। তোমাকে এই 2d plane এর উপর দিয়ে একটি লাইন টানতে হবে। তুমি চাইলে তোমার লাইনকে প্রদত্ত বিন্দুর উপর দিয়েও টানতে পার। এরপর তুমি ঠিক করবে এই লাইনের কোন পাশ সাদা আর কোন পাশ কালো। তোমার পয়েন্ট হবে সাদা পাশে যেকয়টি সাদা বিন্দু আছে যোগ কালো পাশে কয়টি কালো বিন্দু আছে যোগ তোমার রেখার উপর কয়টি বিন্দু আছে। তোমার লক্ষ্য তোমার পয়েন্ট সর্বোচ্চ করা।

সমাধান: প্রথম কথা, এই ধরনের সমস্যা angle sweep করে করতে হয়। কীভাবে চিন্তা করবা? মনে কর তোমার optimal line কোথাও না কোথাও আছে। এখন তোমাকে চেষ্টা করতে হবে এই লাইনকে একটু সিস্টেমে কীভাবে আনা যায়। প্রথমে এই লাইনকে একটু ডানে বা বামে সরালে কিন্তু তোমার উত্তর পরিবর্তন হবে না। তুমি চাইলে ততক্ষণ পর্যন্ত কোনো এক দিকে সরাতে পার যতক্ষণ না সে একটা বিন্দুকে স্পর্শ করে। সুতরাং আমরা বলতে পারি আমরা এমন একটি অপ্টিমাল লাইন অবশ্যই পাব যা প্রদত্ত বিন্দুগুলির কোনো একটির উপর দিয়ে যাবে। এবার এই লাইন যেই বিন্দুর উপর দিয়ে যায়, তাতে স্থির রেখে তুমি লাইনকে চাইলে ঘুরাতে পার। আগের মতই একটু ডান বা বামে ঘুরালে কিন্তু উত্তর পরিবর্তন হবে না। সুতরাং তুমি চাইলে এই লাইনকে ঘুরাতে থাকতে পার যতক্ষণ না সে আরও একটি বিন্দুকে স্পর্শ করে। অর্থাৎ এমন একটি অপ্টিমাল লাইন আমরা পাবই যা প্রদত্ত বিন্দুগুলির কোনো দুটির উপর দিয়ে যাবেই। এই ব্যাপারটা আসলে আমাদের সমাধানকে অনেক সহজ করে দেয়। আমাদের এখন আর অসীম সংখ্যক লাইন candidate হিসাবে নেই। মাত্র n^2 টা candidate লাইন। এই লাইনের একদিকে ভাববা কালো আরেকদিকে সাদা, এরপর গুনে দেখবা তোমার পয়েন্ট কত। সুতরাং প্রতি candidate এর জন্য আমাদের পয়েন্ট হিসাব করতে আরও $\mathrm{O}(n)$ সময় লাগবে। সুতরাং এভাবে করলে আমাদের time complexity হবে $\mathrm{O}(n^3)$. আমাদেরকে time complexity কমাতে হবে।

এখানেই আসবে angle sweep. খেয়াল কর আমরা যদি আমাদের লাইনকে একটা বিন্দুতে fix করি, তাহলে একে যদি আমরা ডানে বা বামে ঘুরাই তাহলে কিন্তু আমাদের উত্তর ওত বেশি পরিবর্তন হয় না। এই আইডিয়াটাই আমাদের কাজে লাগাতে হবে। আমরা প্রথমে একটি একটি করে বিন্দু fix করব। এতে করে আমাদের সময় লাগবে $\mathrm{O}(n)$. এবার বাদ বাকি বিন্দুগুলিকে আমরা এই fixed বিন্দু সাপেক্ষে সর্ট করে ফেলি। মনে কর আমাদের fixed বিন্দু হচ্ছে 0 আর বাদ বাকি বিন্দুগুলি হল

$1 \ldots n-1$ এবং মনে করি আমাদের এই বিন্দুগুলি angle অনুসারে counter clockwise সর্ট করা। শুরুতে মনে কর তোমার লাইন হচ্ছে 0 আর 1 দিয়ে যে লাইন যায়- সেই লাইন। তুমি সবসময় মনে করবা তোমার ডান পাশ সাদা, আর বাম পাশ কালো। এখন তুমি গুনে দেখ তোমার ডান পাশে কয়টা সাদা আর বাম পাশে কয়টা কালো বিন্দু আছে। এই কাজ প্রথমবার তুমি লুপ চালিয়ে করবা। কিন্তু এর পর থেকে আর লুপ চালাবা না। কারণ প্রতিবার লুপ চালালে তো আবার $\mathrm{O}(n^3)$ হয়ে যাবে তাই না (fixed point এর একটি লুপ, $0-i$ রেখা নির্বাচনের একটি লুপ, আর দুই পাশে সাদা কালো দেখার জন্য আরেকটি লুপ)? তাহলে আমরা কি করতে পারি? মনে কর আমরা $0-1$ এর জন্য উত্তর জানি। এখন আমরা $0-2$ এর জন্য উত্তর বের করব। এতে করে কিন্তু আমাদের উত্তর খুব একটা পরিবর্তন হবে না তাই না? আমরা $0-1$ থেকে $0-2$ যাবার ফলে 1 বিন্দুটি এখন আমাদের ডান দিকে এসে গেছে, আর 2 বিন্দুটি আমাদের রেখার উপর এসে গেছে। সুতরাং আমরা আমাদের দুদিকের সাদা কালো বিন্দুর সংখ্যা adjust করে নিতে পারি। ওদিকে 0 এর পেছন দিকেও কিছু বিন্দু এদিক ওদিক হয়ে গিয়েছে। মানে, তুমি যদি 1 হতে 0 হয়ে পেছন দিকে লাইন টান, আর 2 হতে 0 হয়ে পেছন দিকে লাইন টান তাহলে এদের মাঝে কিছু বিন্দু আছে এবং তারা তাদের সাইড পরিবর্তন করেছে। সুতরাং সেটাও আমাদের হিসাব রাখতে হবে। এটা আসলে খুব কঠিন না, তুমি একটি index রাখবে যে 1 হতে 0 হয়ে পেছনে লাইন টানলে সবার শেষ ডান পাশের বিন্দু কোনটা। এবার যখন তুমি 2 তে আসবে, তখন দেখবে এবার সেই শেষ বিন্দু কে, ঠিক line sweep এর মত। Line sweep এ আমরা হয়তো আমাদের ডান মাথা এক বাড়াতাম, আর তার উপর ভিত্তি করে বাম মাথা হয়তো অনেক কয়ঘর বাড়াতে হত। এখানেও ঠিক তাই। এভাবে আমরা i এর মান $1 \ldots n-1$ নিব এবং দুপাশের সাদা কালো বিন্দুর সংখ্যা গুনতে থাকব। তাহলেই আমরা $\mathrm{O}(n^2)$ সময়ে উত্তর বের করে ফেলতে পারব। আসলে এই time complexity হবে $\mathrm{O}(n^2 \log n)$ কারণ আমাদের প্রথম লুপের ভেতরে একটি সর্ট আছে। এছাড়া বাকি angle sweep এর অংশটুকু linear.

যারা বেশ অভিজ্ঞ কোডার তারা এই সমাধানকে আরও simplify করে! খুবই সুক্ষ বিষয়। খেয়াল কর- আমরা চাইলে কালো বিন্দু গুলিকে fixed point এর সাপেক্ষে reflect করতে পারি। এর ফলে আমাদের সমস্যা আর সাদা কালো থাকবে না। আমরা বলতে পারি- এই fixed point সাপেক্ষে লাইন টানলে এর এক পাশে সবচেয়ে বেশি সংখ্যক বিন্দু থাকতে হবে। এতে করে আমাদের সমস্যা অনেক simplify হয়ে যায়। আমাদের এখন আর সাদা কালর সংখ্যা মনে রাখতে হবে না। এখন আমাদের কোডকে আরও সহজ করা যাবে।

এই সমাধানকে আরও আরও সহজ করা যায়! এটা হয়তো লিখে বুঝান একটু কঠিন হবে। তাও বলি, তোমাদের যাদের আগ্রহ আছে তারা চিন্তা করে দেখতে পার। আমরা চাইলে fixed পয়েন্ট এর সাপক্ষে প্রতিটি বিন্দুর জন্য বের করতে পারি যে আমাদের লাইন কত হতে কত angle এর ভেতরে থাকলে আমাদের বিন্দুটি ডান দিকে থাকে। অর্থাৎ প্রতিটি বিন্দুর একটি entry angle থাকবে আর exit angle থাকবে। এবার আমরা এই entry আর exit কে event হিসাবে কল্পনা করে সর্ট করতে পারি এবং line sweep এর মত করে বের করতে পারি যে কোনো এক মুহূর্তে সর্বোচ্চ কতগুলি বিন্দু active ছিল।

UVa 1471 Defense Lines

সমস্যা: তোমাকে সর্বোচ্চ 200,000 দৈর্ঘ্যের একটি অ্যারে দেওয়া আছে। তুমি চাইলে এর একটি সাবঅ্যারে মুছে ফেলতে পার। তোমার লক্ষ্য হল এই মুছে ফেলার পর সবচেয়ে বড় সাব অ্যারে বের করা- যা increasing হয়।

সমাধান: মনে কর আমরা এখন i index এ দাড়িয়ে। মনে কর আমরা জানি এখান থেকে শুরু করে (অর্থাৎ i তম স্থান সহ) এর ডান দিকে কতদুর পর্যন্ত strictly increasing sequence পাওয়া যায়। তাহলে আমরা আর কি জানতে চাই? জানতে চাই যে, i এর বাম দিকে কোথাও শেষ হওয়া সবচেয়ে বড় strictly increasing sequence এর লেংথ কত এবং এই sequence এর শেষ সঙ্খ্যাটা $a[i]$ এর থেকে ছোট হতে হবে যেখানে a হচ্ছে আমাদের ইনপুটের অ্যারে। এটুকু যদি আমরা করতে পারি তাহলে প্রতিটা i এর জন্য আমরা এই কাজ করলেই আমাদের উত্তর বেরিয়ে আসবে।

প্রথমে আমরা দেখি i index এর জন্য i তম স্থান সহ এর ডান দিকে strictly increasing sequence এর লেংথ কীভাবে বের করা যায়। এটা সহজ এবং কমন। আমরা ডান দিক থেকে আসব। শুরুতে $length1[n] = 1$ অর্থাৎ n তম স্থানের জন্য উত্তর 1. এর পর i তম স্থানের জন্য আমরা দেখব এই স্থানের সংখ্যা $i + 1$ তম স্থানের সংখ্যার থেকে ছোট কিনা। যদি ছোট হয়, তাহলে আমাদের i তম স্থানের উত্তর $i + 1$ তম স্থানের উত্তরের থেকে এক বেশি হবে। আর যদি ছোট না হয়? তাহলে তো সহজ, উত্তর 1. এভাবে আমরা i তম স্থানের জন্য i তম স্থান সহ এর ডানে strictly increasing sequence এর মান বের করতে পারব।

এবার কঠিন অংশ। কীভাবে i তম স্থানে দাড়িয়ে আমরা বের করতে পারব i এর বাম দিকে কোথাও $a[i]$ হতে ছোট কোনো সংখ্যায় শেষ হওয়া সবচেয়ে বড় strictly increasing sequence এর লেংথ কত। প্রথমত কিছুক্ষণ আগেই আমরা প্রতিটি স্থানের জন্য সেই স্থান সহ তার ডান দিকে strictly increasing sequence এর সর্বোচ্চ লেংথ বের করেছি। একই ভাবে আমরা খুব সহজেই বের করতে পারব i এ শেষ হওয়া (i তম স্থান সহ) strictly increasing sequence এর লেংথ কত, ধরা যাক এটা আমরা $length2$ নামের এক অ্যারেতে রেখেছি। যদি এটুকু হয়ে যায় তাহলে আমাদের কাজ হবে হবে- প্রতিটি i এ গিয়ে সবগুলি $j < i$ দেখা যাতে করে $a[j] < a[i]$ হয় এবং $length2[j]$ সর্বোচ্চ হয়। এই কাজ নানা উপায়ে করা যায়।

একটি উপায় হতে পারে আমরা i এর লুপ বাম থেকে ডানে যাব। আর একটি segment tree রাখব। আমরা segment tree তে জিজ্ঞাসা করব- "আচ্ছা a[i] থেকে ছোট index এ সবচেয়ে বড় কত সংখ্যা আছে?"। আর আমাদের i এ কাজ শেষ হয়ে গেলে একে আমরা segment tree তে রেখে দেব। আমাদের আপডেট অপারেশন হবে- "a[i] এ length2[i] রেখে দেবার চেষ্টা কর তো!"। চেষ্টা বললাম এ কারনে যে, হয়তো $a[i]$ এ ইতোমধ্যেই $length2[i]$ এর থেকেও বড় সংখ্যা আছে। তাহলে এভাবে segment tree রেখে আমরা খুব সহজেই $\mathrm{O}(n \log n)$ এ এই সমস্যার সমাধান করতে পারি।

আমরা চাইলে binary search বা set ব্যবহার করেও করতে পারি। মনে কর আমাদের set এ $(a[j], length2[j])$ সমুহ আছে। আমরা i তম index এ আসব। এসে সেটকে জিজ্ঞাসা করব যে $a[i]$ এর থেকে immediate ছোট কোন $a[j]$ আছে? এই কাজ আমরা lower bound ব্যবহার করে সহজেই করতে পারি। সেই a[j] বের করে তার length2[j] নেব। কিন্তু এমনও তো হতে পারে যে কোনো immediate ছোট a[j] না নিয়ে তার থেকেও ছোট কোনো a[j] এর জন্য length2[j] ভাল। হতে পারে, তবে এখানেই মজা। আমরা যখন $(a[j], length2[j])$ ঢুকাব তখনই দেখব যে এর থেকে ছোট a[k] কত? তার length2[k] কত? সেটা কি length2[j] এর থেকে বড়? তাহলে তো $(a[j], length2[j])$ ঢুকানর কোনো দরকার নেই। অর্থাৎ আমাদের সেট টাতে সবসময় a গুলি increasing আকারে থাকবে (সেটের property অনুসারে) আর length2 ও থাকবে increasing অর্ডারে (আমরা এটা নিশ্চিত করব)। একই ভাবে সেটে $(a[i], length2[i])$ ঢুকানোর পর যদি দেখি এর পরের $length2[k]$ এর মান আমাদের $length2[i]$ এর থেকে ছোট, তাহলে তো সেই $(a[k], length2[k])$ রাখার আর দরকার নেই। এভাবে আমরা সেট কে সবসময় ঠিক ভাবে maintain করে রাখব।

UVa 10954 Add All

সমস্যা: তোমাকে n টি সংখ্যা দেওয়া আছে ($n \leq 5000$). তোমাকে এসকল সংখ্যা যোগ করতে হবে। দুটি সংখ্যা a আর b কে যোগ করার cost হল $a + b$. সর্বনিম্ন কত cost এ সকল সংখ্যাকে তুমি যোগ করতে পারবে।

সমাধান: এটা একটা ক্লাসিক্যাল সমস্যা। এর সমাধান huffman coding. Huffman coding এর সমস্যা মনে হয় আমরা greedy section এ দেখে এসেছি। এই বই এ না থাকলেও প্রোগ্রামিং কন্টেস্ট বই এ অবশ্যই আছে। প্রব্লেমটা ছিল এরকম- আমাদেরকে বলা আছে কোন অক্ষর কতবার ব্যবহৃত হয়। এখন আমাদেরকে প্রতিটি অক্ষরের জন্য একটি বাইনারি কোড বের করতে হবে যেন কোডগুলির কোনটি অন্য কোনটির prefix না হয়। তোমাকে সবচেয়ে কম খরচে এই কাজ করতে হবে। এখানে খরচ মানে হল- ধর কোনো একটি অক্ষর 5 বার ব্যবহৃত হয়। আর তার কোডের দৈর্ঘ্য 3. এর মানে এক্ষেত্রে খরচ হল 15. এই সমস্যার সমাধান হল huffman coding. এখানে আমরা এই কোডিং এর বিস্তারিত বলব না। এই সমাধানের একটা মুখ্য অংশ ছিল একটা বাইনারি ট্রি বানানো।

খেয়াল কর, এই বাইনারি ট্রি তে প্রথমেই weight গুলি দুই ভাগ হত, এর পর প্রতিটি নোডে আরও দুই ভাগ হত। এরকম করে যতক্ষণ না একটি সংখ্যা থাকে ততক্ষণ ভাগ হতেই থাকে তাই না? বা অন্যভাবে চিন্তা কর, প্রতিটি নোডে দুইগ্রুপ সংখ্যা যোগ হয়। এরকম করতে করতে root এ গিয়ে সব সংখ্যা একত্রে যোগ হয়। এটাই তো আমাদের বর্তমান সমস্যাতেও হচ্ছে তাই না? Huffman coding এর সমস্যাতে আমাদেরকে মোট কোডের লেংথকে সর্বনিম্ন করতে হত। মোট কোডের লেংথ মানে হল প্রতিটি কোডের লেংথ আর তার weight (ঐ অক্ষর কত বার আছে) এর গুনফল। আমাদের সংখ্যাতেও weight দেওয়া আছে (প্রতিটি সংখ্যা কত)। আমাদেরকে বের করতে হবে প্রতিটি সংখ্যা কতবার করে যোগ হয়, মানে huffman coding এ প্রতিটি অক্ষরের length যত, ততবারই তো ঐ সংখ্যা যোগ অপারেশনে অংশ নেয় তাই না? আর কোনো একটি সংখ্যা যোগ অপারেশনে অংশ নিলে আমাদের cost দিতে হয়, আর আমাদের সমস্যার মূল লক্ষ্যই হল এই cost কমানো। এটাই তো huffman coding এর কাজ তাই না?

UVa 12627 Erratic Expansion

সমস্যা: শুরুতে একটি লাল বেলুন থাকবে। এরপর প্রতি stage এ এরকম পরিবর্তন হবে- নীল বেলুন পরিবর্তন হয়ে 2×2 সাইজের চারটা নীল বেলুনে পরিবর্তন হবে আর লাল বেলুন পরিবর্তন হয়ে 2×2 সাইজের চারটি বেলুনে পরিবর্তন হবে যার নিচের ডান কোণার নতুন বেলুনের রঙ হবে নীল আর বাকি তিনটির রঙ হবে লাল। এরকম করে K stage এই অপারেশন করলে ($K \leq 30$) বলতে হবে এই শেষ বোর্ডে A হতে B রো পর্যন্ত মোট কতগুলি লাল বেলুন আছে।

সমাধান: তোমাদেরকে এর আগেই বলেছি যদি কখনও A হতে B পর্যন্ত কিছু গুনতে বলে বা সমাধান করতে বলে তাহলে চেষ্টা করবে 1 হতে B পর্যন্ত উত্তর বের করে তা হতে 1 হতে $A-1$ এর উত্তর বিয়োগ করে সমাধান করতে। এই ক্ষেত্রে এটা সম্ভব। অর্থাৎ আমরা যদি সমাধান করতে পারি যে- 1 হতে A পর্যন্ত রো গুলিতে কতগুলি লাল বেলুন আছে- তাহলেই মূল সমস্যা সমাধান হয়ে যাবে।

এখন কিছু observation এর বেলা। প্রথমে খেয়াল কর K stage পরে বোর্ডের সাইজ কত? 1 stage পরে এর সাইজ হয় 2×2, 2 stage পর 4×4, 3 stage পর 8×8 এরকম। সুতরাং K stage পর সাইজ হবে $2^K \times 2^K$. আরও খেয়াল কর, একটি বেলুন এক stage পর চারটি বেলুনে পরিবরতিত হয়। এই চারটি বেলুনের উপরের দুটি বেলুন, মূল বেলুনের K stage পর যেই বোর্ড হবে তার উপরের অর্ধেক বানাবে, আর চারটি বেলুনের নিচের দুটি বেলুন K stage পরের বোর্ডের নিচের অর্ধেক দেবে। এখন আমরা A এর সাথে compare করব। A কোথায় আছে? নিচের অর্ধেকে নাকি উপরের অর্ধেকে? যদি নিচের অর্ধেকে হয় তাহলে উপরের দুটি বেলুন $K-1$ stage পর যে কয়টি লাল বেলুন দেবে সবগুলি আমাদের যোগ করতে হবে। সেই সাথে আমরা $A = A - 2^K/2$ করব আর নিচের দুটি বেলুনের জন্য recursively জিজ্ঞাসা করব যে $K-1$ stage পর এই নতুন A এর জন্য কয়টি লাল বেলুন পাওয়া যাবে। আর যদি A উপরের অর্ধেকে পরত তাহলে উপরের বেলুনগুলির জন্য পুরাতন A নিয়ে ফাংশন কল করলেই হয়।

তার মানে আমাদের একটা ফাংশন রাখতে হবে যেটি বর্তমান বেলুনের রঙ, stage এর সংখ্যা আর কয়টা রো এর জন্য হিসাব করতে হবে তা parameter হিসাবে নেবে। এই ফাংশন recursively নিজেকে কল করবে এক stage কমের জন্য, এবং দরকারে কয়টা রো (A) তার মান পরিবর্তন হবে। বেলুনের রঙও পরিবর্তন হতে পারে। কিছুক্ষণ আগেই আমরা দুইটা case দেখেছি। যদি A উপরের অর্ধেকে পরে তাহলে এই ফাংশন দুইবার কল হবে, আর যদি নিচে পরে তাহলেও দুইবার কল হবে নিচের দুটি বেলুনের সাথে। আর আমরা এই দ্বিতীয় case এ উপরের দুটি বেলুনের পুরোপুরি নেব (অর্থাৎ আরও $K-1$ stage পর যেই অবস্থা হবে তার সব বেলুন) তাই না? আমরা কিন্তু আগে থেকে বের করতে পারি লাল বেলুনকে i stage পর্যন্ত expand করলে তাতে কয়টা লাল বেলুন থাকবে। আর নীল বেলুনকে expand করলে তো কোনো লাল বেলুন পাবে না। সুতরাং এখানে আমরা আমাদের ফাংশনকে দুইবার কল করব।

এবার দেখ, আমাদের কিন্তু দুইবার কল করার দরকার নেই। এই দুটি কলের একটি যদি নীল বেলুন নিয়ে হয় তাহলে তো expand করে লাভ নেই। আর যদি দুটিই লাল হয় তাহলে তো আসলে দুইবার কল করার দরকার নেই। একবার কল করে তাকে দুই দিয়ে গুন করে দিলেই হল।

UVa 1451 Average

সমস্যা: তোমাকে সর্বোচ্চ 100,000 লেংথের একটি অ্যারে দেওয়া আছে যার প্রতিটি element হয় 0 নাহয় 1. তোমাকে একটি সাব অ্যারে নির্বাচন করতে হবে যার লেংথ কমপক্ষে L হয় এবং এতে এক এর সংখ্যা গড়ে সবচেয়ে বেশি হয়। মানে যদি এই সাব অ্যারে এর লেংথ হয় s আর এতে 1 এর সংখ্যা হয় a তাহলে আমাদের লক্ষ্য হল a/s কে সর্বোচ্চ করা যেখানে $s \geq L$. L এর মান সর্বোচ্চ 1000 হতে পারবে।

সমাধান: অর্থাৎ আমাদেরকে i আর j দুটি index বের করতে হবে যেন $j - i + 1 \geq L$ হয় আর i হতে j পর্যন্ত 1 এর সংখ্যা যদি S হয় তাহলে আমাদের লক্ষ্য $\frac{S}{j-i+1}$ এর মান যেন সবচেয়ে বেশি হয়। কথা হল S এর মান কীভাবে বের করবে? বেশ কমন একটা টেকনিক ব্যবহার করতে হবে। Cumulative sum. আমরা যদি 1 এর cumulative sum যদি p নামের একটি অ্যারে তে রাখি তাহলে $S = p[j] - p[i-1]$. তাহলে আমাদের average 1 এর ফর্মুলা হবে $\frac{p[j]-p[i-1]}{j-i+1}$. এই ফর্মুলা দেখতে বেশ জটিল লাগছে। আমরা যদি $i = i - 1$ বসাই তাহলে বেশ সুন্দর ফর্মুলা হবে $\frac{p[j]-p[i]}{j-i}$. একটু চিন্তা করে দেখ তো এই ফর্মুলা পরিচিত লাগে কিনা? এটা slope এর ফর্মুলার মত। তুমি যদি i, j এগুলকে x হিসাবে ভাব আর p গুলিকে y হিসাবে ভাব তাহলে এটা হল slope. তাহলে আমাদের প্রব্লেম modify করলে দাঁড়ায়, তোমার কাছে n টি বিন্দু দেওয়া আছে। তোমাকে $i < j$ choose করতে হবে যেন $j - i \geq L - 1$ হয় এবং slope সবচেয়ে বড় হয়।

মনে কর তোমার কাছে অনেক গুলো বিন্দু আছে। এখন তোমাকে বলল এদের ভেতর থেকে এমন একটি বিন্দু নির্বাচন করতে যেটা আরেকটা বিন্দু q এর সাথে সবচেয়ে বড় slope তৈরি করবে, যেখানে q এই বিন্দুগুলি থেকে বামে অবস্থিত। একটু চিন্তা করে দেখ তো q কীভাবে বের করবা? সহজ। তুমি যদি এই বিন্দুগুলির convex hull তৈরি কর তাহলে q হতে tangent টানলেই তুমি সবচেয়ে বড় slope তৈরি করতে পারবা। আসলে convex hull এর দরকার নাই। শুধু upper left hull তৈরি করলেই হবে।

তাহলে কি দাঁড়াল? আমরা ডান থেকে বামে আসব। যখন আমরা i এ আছি তখন $i + L$ কে upper left hull এ insert করব। এরপর আমরা i তম বিন্দু হতে tangent টানব। এখন দুইটা প্রশ্ন- কীভাবে upper left hull বানাবা আর কীভাবে tangent টানবা।

Upper left hull বানানো কিন্তু খুব একটা কঠিন না। মনে কর এই hull এর বিন্দু গুলি হল- $u[1], u[2] \dots u[m]$ ডান থেকে বামে এই অর্ডারে। এখন যখন নতুন একটি বিন্দু p আসবে, তুমি দেখবে $p, u[m], u[m-1]$ এরা কি counter clockwise এ আছে কিনা, থাকলে $u[m]$ কে ধরে ফেলে দিতে পার এবং আবার ঐ turn চেক করতে হবে যতক্ষণ না turn টা clockwise এ থাকে (অথবা 2 এর কম element থাকে)। সবশেষে p কে u অ্যারেতে insert করতে হবে। এভাবে আমরা upper left hull কে maintain করতে পারি।

এখন দ্বিতীয় প্রশ্ন, কীভাবে tangent টানবে? বেশ সহজ, আমরা আমাদের u এর অ্যারে তে ternary search করতে পারি সবচেয়ে বড় slope কোন বিন্দুতে হয় তা বের করার জন্য। আমার বিশ্বাস এটা সমাধানের জন্য যথেষ্ট। তবে ইচ্ছা করলে অতিরিক্ত $\log n$ factor না এনেও সমাধান করা যায়। ছোট্ট একটা হিন্ট দেই। যদি q হতে tanget টানলে হালের বিন্দুটি যদি হয় $u[k]$ তাহলে কিন্তু $u[1] \dots u[k-1]$ এই বিন্দুগুলি কখনও optimal slope এর অংশ হবে না। এই property ব্যবহার করে তুমি ternary search ব্যবহার না করে amortized linear সময়ে সমাধান করতে পার।

UVa 714 Copying Books

সমস্যা: সর্বোচ্চ 500 লেংথের একটি অ্যারে আছে। তোমাকে এই আরেকে k ভাগে ভাগ করতে হবে (অর্থাৎ k টি সাবঅ্যারে তে ভাগ করতে হবে)। প্রতিটি ভাগের সংখ্যাগুলি যোগ কর। লক্ষ্য হল সবচেয়ে বড় যোগফল যেন সবচেয়ে ছোট হয়। আমাদের প্রদত্ত অ্যারের সকল সংখ্যা ধনাত্মক।

সমাধান: মাঝে মাঝে চিন্তা করতে হয়- যদি আমার উত্তর জানা থাকত তাহলে কি আমরা বের করতে পারতাম যে আমরা আমাদের ইনপুটের অ্যারেকে k বা তার কম ভাগে ভাগ করতে পারব কিনা? এক্ষেত্রে ব্যাপারটা সহজ তাই না? মনে কর আমরা জানি আমাদের সবচেয়ে বড় যোগফল হল S. তাহলে আমরা অ্যারের প্রথম থেকে সংখ্যাকে প্রথম ভাগে রাখতে থাকব যতক্ষণ যোগফল S অতিক্রম না করে। এর পর দ্বিতীয় ভাগ নিতে থাকব। এভাবে চলতে থাকবে। এভাবে সব সংখ্যা নেয়া হয়ে গেলে দেখব কয়টা ভাগ হয়েছে।

কিন্তু সমস্যা হল আমরা তো আমাদের উত্তর জানি না তাই না? কি উপায়? Binary search. খেয়াল কর আমরা মনে কর দুই রকম guess করলাম, একটা হল S_1 আরেকটা S_2 যেখানে $S_1 < S_2$. S_1 এর জন্য মনে কর ভাগ হয় K_1 আর S_2 এর জন্য K_2. আমরা বলতে পারি $K_1 \geq K_2$. কারণ যদি আমাদের S কম হয় তাহলে তো ভাগের সংখ্যা বেড়ে যাবে তাই না? সুতরাং আমরা binary search করে সঠিক S এর মান বের করে ফেলতে পারব। যদি কোনো একটি guess এর জন্য দেখি K এর মান আমাদের কাঙ্ক্ষিত মানের থেকে বেশি হয়েছে এর মানে হবে আমাদের S বাড়াতে হবে। কারণ এই S এর ফলে আমাদের ভাগ সংখ্যা বেশি হয়ে গিয়েছে (K এর মান বেশি), সুতরাং আমাদের S এর মান বাড়িয়ে K এর মান কমাতে হবে। একই ভাবে যদি কোনো একটি guess এর জন্য যদি দেখি K এর মান আমাদের কাঙ্ক্ষিত মানের থেকে হয়েছে তাহলে S কমাতে হবে। এভাবে আমরা optimal S এর মান বের করতে পারব।

UVa 11093 Just Finish it up

সমস্যা: একটি বৃত্তাকার রাস্তায় N টি গ্যাস স্টেশন আছে। প্রতিটি গ্যাস স্টেশনে কি পরিমান গ্যাস আছে তা বলা আছে। আবার প্রতিটি গ্যাস স্টেশন হতে তার পরের স্টেশনের দুরত্বও বলা আছে। বলতে হবে কোন স্টেশন থেকে তুমি কোনো গ্যাস ছাড়া গাড়ি নিয়ে যাত্রা শুরু করলে কোনো সমস্যা ছাড়া আবার সেই স্টেশনে ফেরত আসতে পারবে (পুরো রাস্তা গোল করে ঘুরে)। N এর মান 100000 হতে পারে। এক ইউনিট দুরত্ব যেতে এক ইউনিট গ্যাস লাগে।

সমাধান: এটা একটা পাজেল জাতীয় সমস্যা। আমরা কোনো একটি রাস্তায় যেতে যে পরিমান গ্যাস খরচ হয় সেই পরিমান গ্যাস আমরা তার আগের গ্যাস স্টেসন থেকে বিয়োগ করব। তাহলে আমাদের প্রবলেম পরিবর্তন হয়ে দাঁড়ায়- একটি N দৈর্ঘ্যের circular অ্যারে আছে। আমাদেরকে বলতে হবে অ্যারের কোন জায়গা থেকে শুরু করলে পুরো অ্যারের সংখ্যাগুলি যোগ করতে করতে ঘুরে আসলেও কোথাও আমাদের যোগফল ঋণাত্মক হবে না।

আমরা আমাদের কাজ সহজ করার জন্য এই অ্যারেকে তার নিজের মত এক অ্যারের (আগেরটার কপি) সাথে জোড়া লাগায়ে $2N$ লেংথের বানাব। অর্থাৎ i তম index এ যে সংখ্যা আছে $i + N$ তম index এও একই সংখ্যা আছে ($1 \leq i \leq N$ আর ধরে নিচ্ছি অ্যারেটি 1 indexed). এবার এই অ্যারের প্রথম স্থান থেকে প্রতিটি স্থান পর্যন্ত cumulative sum বের করব। এই cumulative sum এর অ্যারে মনে কর S. মনে কর আমরা যাত্রা শুরু করব i এর ঠিক পরে থেকে তাহলে j তে পৌঁছাতে গ্যাসের পরিমান হবে $S[j] - S[i]$. আমাদের লক্ষ্য এমন একটি i নির্বাচন করা যেন সকল $j > i$ এর জন্য $S[j] - S[i] \geq 0$ হয়। এখানে তোমরা ভাবতে পার আমি কেন হঠাত $2N$ দৈর্ঘ্যের অ্যারে নিলাম। এটা আসলে circular প্রব্লেমকে লিনিয়ারে কনভার্ট করার একটি সুন্দর টেকনিক। তুমি যদি প্রত্যেক $[i, i + N - 1]$ রেঞ্জের জন্য সমাধান করতে পার ($1 \leq i \leq N$) তাহলে পুরো সাইকেল এর জন্য সমাধান হয়ে যাবে (সব সমস্যার ক্ষেত্রেই যে সম্ভব তা না, কিন্তু বেশ অনেক সমস্যার জন্য সম্ভব)। যেমন এই সমস্যার ক্ষেত্রে আমাদের কাজ হল $1 \leq i \leq N$ মেনে এমন একটি i নির্বাচন করতে হবে যেন $[i + 1, i + N]$ এই রেঞ্জের ভেতরে যেকোনো j এর জন্য $S[j] - S[i] \geq 0$ হয় (আসলে $j \leq i + N$ এর জন্য সত্য হলে আমাদের সমস্যার ক্ষেত্রে সকল $j \leq 2N$ এর জন্যও সত্য হবে)। এখানে $S[j] - S[i]$ এর মানে হল $i + 1$ থেকে j পর্যন্ত সংখ্যার যোগফল। আর আমরা চাই এই যোগফল যেন ঋণাত্মক না হয়। এখন তোমরাই চিন্তা করে দেখ কোন i নির্বাচন করবা? আসলে যেকোনো j এর জন্য $S[j]$ তো fixed. আমরা যদি $S[i]$ কে কমাতে পারি তাহলে $S[j] - S[i]$ সবচেয়ে বড় হবে। আর আমাদেরকে কিন্তু $i \leq N$ নির্বাচন করতে হবে তা না হলে তো $i + N$ তো 2N এর থেকে বেশি হয়ে যাবে। তাহলে আমরা $1 \leq i \leq N$ এর জন্য সবচেয়ে ছোট $S[i]$ নির্বাচন

করব। এরপর আমরা প্রত্যেক j এর জন্য চেক করে দেখব যে $S[j] - S[i] \geq 0$ হয় কিনা।

UVa 1607 Gates

সমস্যা: অনেকগুলি NAND গেট আছে। গেট গুলি একটি আরেকটির সাথে বিভিন্ন ভাবে জোড়া লাগানো আছে। তবে তাদের মাঝে সাইকেল নেই অর্থাৎ কোনো একটি গেটের আউটপুট গিয়ে আবার তার নিজের ইনপুটে (সরাসরি হোক বা অন্য আরও গেটের মাঝ দিয়ে হোক) লাগবে না। এই সিস্টেমে মূল ইনপুট একটাই x. এই একই x ই বিভিন্ন গেটের ইনপুট হিসাবে কাজ করবে বা অন্য কোনো গেটের আউটপুট আরেকটা গেটের ইনপুট হিসাবে কাজ করবে। তোমাকে এই base ইনপুট গুলর সবচেয়ে কম সংখ্যক x এ রাখতে হবে আর বাকিগুলি 0 বা 1 করে দিতে হবে যেন আমাদের circuit ফাংশন ঠিক থাকে। অর্থাৎ x এর 0 বা 1 দিলে আগে যা আসত এখনও যেন তাই আসে। ইনপুটে গেটের সংখ্যা 200,000 হতে পারে। আর base ইনপুটের সংখ্যা সর্বোচ্চ 100,000 হতে পারে।

সমাধান: প্রচণ্ড ট্রিকি একটা সমস্যা। প্রথমে দেখ যে x এর মান যদি 0 হয় তাহলে আউটপুট কত আর 1 হলেই বা আউটপুট কত। যদি তারা সমান হয় তার মানে আসলে কোনো x ইনপুট না নিয়ে সব কয়টা 0 বা 1 দিলেই হয়। সমস্যা হল যখন x এর মান 0 এর জন্য শেষ আউটপুট আর x এর মান 1 এর জন্য শেষ আউটপুটের মান সমান হয় না। প্রথম কথা আমাদের ইনপুটে কমপক্ষে একটি x লাগবেই তাই না? কারণ যদি ইনপুটে কোনো x না থাকে তাহলে তো ইনপুট vary করবে না। ইনপুট vary না করলে তো x এর মান যাই হোক না কেন আউটপুট একই থাকবে। কিন্তু আমরা তো জানি x এর মান vary করলে আমাদের আউটপুটও vary করবে। উপায় কি?

এখানেই প্রব্লেম সল্ভের মজা। খেয়াল কর তোমরা জান যে $00\ldots0$ এর জন্য এক আউটপুট আর $11\ldots1$ এর জন্য আরেক ইনপুট। আমাদেরকে এমন একটি ইনপুট বের করতে হবে যাতে কমপক্ষে একটি x থাকে এবং $x = 0$ বসালে $00\ldots0$ এর জন্য যেই আউটপুট (ধরলাম A) আসে সেটি দেবে আর $x = 1$ বসালে $11\ldots1$ এর জন্য যেই আউটপুট (ধরলাম B) আসে সেটি দেবে। কীভাবে সমাধান করবা?

আমরা আপাতত মনে করি আমাদের ইনপুট 5 টা। তাহলে আমরা এই 6 টা ইনপুট চেষ্টা করব $00000, 00001, 00011, 00111, 01111, 11111$. আমাদের একদম বামে আছে A, একদম ডানে আছে B আর মাঝের গুলি A বা B যেকোনোটিই হতে পারে। আমরা এই ছয়টি ইনপুট sequence এর মাঝে অবশ্যই পাশাপাশি দুইটা পাবই যার বামেরটা A আর ডানেরটা B. ধরা যাক 00011 এ A আর 00111 এ B. তাহলে আমরা যদি $00x11$ ইনপুট হিসাবে দেই তাহলেই কিন্তু আমাদের সমস্যা সমাধান হয়ে যাবে। কারণ আমরা জানি আমাদের কমপক্ষে একটা x লাগবেই আর এখানে একটি x ব্যবহার করা হয়েছে। x এর মান 0 হলে আমাদের আউটপুট A আসে আর x এর মান 1 হলে B আসে, আর সেটাই তো আমাদের দরকার তাই না? সুতরাং আমরা যদি N ইনপুটের জন্য এই রকম $N+1$ টি ইনপুট sequence চেষ্টা করি তাহলেই তো হয়ে যাবে। সমস্যা হল এতে TLE হবে কারণ আমাদেরকে প্রায় $200,000$ বার এই কাজ করতে হচ্ছে। আর প্রতিবার আমাদেরকে $100,000$ টি গেট প্রসেস করতে হচ্ছে। উপায় কি?

Binary search. আমরা বাইনারি সার্চ করে এমন জায়গা বের করব যেখানে পাশাপাশি A আর B আছে। একটা জিনিস, এখানে আমাদের ইনপুট sequence যে সবসময় $AAA\ldots AABBB\ldots B$ এর মত হবে তা কিন্তু না। এমনও হতে পারে $AABBABBABBABB$. এটা নিশ্চিত যে আমাদের সবার বামে A আছে আর সবার ডানে B আছে। আমাদের কাজ এমন পাশাপাশি দুটা বের করা যেখানে A আর B পর পর থাকে। কীভাবে করব? আমরা যেটা নিশ্চিত করব সেটা হল- সবসময় আমাদের sequence এর একদম বামে A আর একদম ডানে B আছে। মনে কর বাইনারি সার্চের সময় আমাদের মধ্যখান test করছি। সেখানে আমরা A পেলাম। তাহলে আমরা কোন ভাগ নেব? বাম না ডান? সহজ, ডান। কারণ আমরা চাই আমাদের sequence এর বামে যেন A আর ডানে B থাকে। আমরা যদি বাম ভাগ নিতাম তাহলে তো ডান প্রান্তে A থাকত তাই না? সুতরাং আমাদের ডান ভাগ নিতে হবে। একইভাবে আমাদের মধ্যখানে যদি B পাওয়া যায় তাহলে আমাদের কে বাম ভাগ নিতে হবে। এভাবে binary search করতে থাকলে একসময় আমাদের sequence মাত্র দুই লেংথের হয়ে যাবে যার বামে A আর ডানে B. সুতরাং আমাদের সমস্যা সমাধান হয়ে যাবে।

UVa 12174 Shuffle

সমস্যা: তোমার গানের লিস্টে s (সর্বোচ্চ 100,000) টি গান আছে। গান গুলি 1 হতে s দ্বারা প্রকাশ করা হবে। তুমি shuffle ফিচার ব্যবহার কর। অর্থাৎ একদম শুরুতে তোমার সফটওয়্যার গান গুলির একটি random permutation করবে এরপর সেসব শেষ হয়ে গেলে আবার নতুন একটি random permutation করবে এরকম করে চলতে থাকবে। তোমাকে গানের partial history দেওয়া আছে (সর্বশেষ বাজা n টি গান, $n \leq 100,000$)। অর্থাৎ শেষ এতটা গান কি কি ছিল। বলতে হবে কয়টা গান পর তোমার সফটওয়্যার আবার গানের লিস্ট shuffle করবে। তোমাকে আসলে কয়টা গান পর বলতে হবে না, বলতে হবে "কয়টা গান পর" এর কয়টা উত্তর আছে। অর্থাৎ যদি এরকম হয় যে, 2 টা গান পরও শাফল হতে পারে, 4 টা গান পরও হতে পারে, আর কোনো অপশন নাই। এর মানে এরকম 2 টা উত্তর আছে। এমনও হতে পারে যে প্রদত্ত history টা ভুল, সেক্ষেত্রে 0 আউটপুট করতে হবে।

সমাধান: মনে কর গানের অ্যারেটি হল 1-indexed. এর মানে হল গানের একটি অ্যারেতে 1 হতে n index এ সর্বশেষ n টি গান দেওয়া আছে। আমরা যেটা করব তাহল প্রতিটি i এ যাব আর মনে করব i এর আগে দিয়ে শাফল হয়েছিল, ফলে i হতে $i+s-1$ পর্যন্ত গানগুলির index আলাদা আলাদা হবে। বা অন্যভাবে বললে দাঁড়ায় তুমি যদি দেখ যে i হতে $i+s-1$ এর মাঝে যদি কোনো duplicate সংখ্যা থাকে তার মানে হবে i থেকে কোনো শাফল শুরু হতে পারবে না। এরকম করে আমরা সকল i এ বের করব যে সেখান থেকে কোনো শাফল শুরু হতে পারে কিনা। কোন কোন i আমাদের চেক করা উচিত? যেসকল i এর জন্য $[i, i+s-1]$ এই রেঞ্জটি আমাদের ইনপুট এর সাথে intersect করে। তার মানে $[2-s, n]$ এই রেঞ্জের মাঝের সকল মানকে i নিয়ে নিয়ে চেষ্টা করতে হবে। কোনো একটি i এর জন্য এই s লেংথের অ্যারে sequence চেক করা তো সহজ। কিন্তু প্রতিটি i এর জন্য তো আর আমরা $\mathrm{O}(s)$ এ এই কাজ করতে পারি না তাই না? এখানে আমরা sliding window টেকনিক এর সাহায্য নেব। আমরা প্রথমে দেখব যে $[2-s, 1]$ এর মাঝে কোনো duplicate আছে কিনা, এরপর দেখব $[1-s, 2]$ এর মাঝে আছে কিনা এরকম করে আমরা window কে এক ঘর করে সরাতে থাকব যতক্ষণ না আমরা সব window এর হিসাব করে ফেলছি। কীভাবে হিসাব করবা তাতো সহজ। আমরা একটি অ্যারে নিব যাতে লিখা থাকবে বর্তমান segment এ কোন সংখ্যা কয়বার করে আছে। আর আরেকটা variable রাখব যে এই অ্যারেতে এখন কয়টা 2 বা 2 এর বেশি আছে। যেহেতু আমরা প্রতি window আপডেটে মাত্র একটি সংখ্যা মুছে যায় বা নতুন যোগ হয় সেহেতু আমরা এই অ্যারে আপডেট আর এই অতিরিক্ত ভ্যারিয়েবল আপডেট এর কাজ খুব সহজেই করতে পারব।

এই কাজ শেষ হয়ে গেলে এখন দেখতে হবে কোন কোন জায়গা থেকে shuffle শুরু হতে পারে। মনে কর i তম স্থান থেকে shuffle হতে পারে। এর মানে হল $i \pm s, i \pm 2s \ldots$ এর প্রতিটা হতেই shuffle শুরু হয়। এর মানে আমাদেরকে 0 হতে $s-1$ পর্যন্ত প্রতিটি mod এর জন্য চেক করতে হবে যে সেই mod এর সকল index থেকেই shuffle শুরু হতে পারে কিনা। যা খুবই সহজ কাজ।

UVa 1608 Non boring sequence

সমস্যা: একটি n দৈর্ঘ্যের সংখ্যার অ্যারে দেওয়া আছে ($n \leq 200,000$). অ্যারেটি non-boring হবে যদি এর প্রতিটি সাবঅ্যারেতে একটি unique সংখ্যা থাকে। বলতে হবে প্রদত্ত অ্যারেটি boring না non-boring.

সমাধান: যদি তুমি এমন একটি সাবঅ্যারে খুঁজে বের করতে পার যাতে কোনো ইউনিক element নেই তাহলেই তোমার কাজ শেষ। এখন প্রশ্ন হল কীভাবে খুব দ্রুত এরকম একটি সাবঅ্যারে খুঁজে বের করতে পারবে। মনে কর তুমি প্রথমে পুরো অ্যারে নিলে $[1, n]$. এতে একটি ইউনিক element i এ খুঁজে পেলে। এর মানে যেকোনো $[a, b]$ যেখানে $a < i < b$ এর জন্য একটি ইউনিক element আছেই। এরপর আমরা দেখব $[1, i-1]$ অংশটুকু কি boring কিনা, আর অন্যদিকের $[i+1, n]$ অংশটুকু boring কিনা। একই ভাবে এই ছোট সাবঅ্যারেতে আমরা দেখব কোনো unique element আছে কিনা। থাকলে তার সাপেক্ষে আমাদেরকে এই সাবঅ্যারেকে ভাগ করতে হবে এবং এভাবে চলতে থাকলে আমাদের সমস্যা সমাধান হয়ে যাবে। কিন্তু কোনো একটি সাবঅ্যারেতে ইউনিক element আছে কিনা বা থাকলে কোনটা এটা কীভাবে বের করবা? প্রতিবার $\mathrm{O}(n)$ এ এই ভাগের কাজ করলে

কিন্তু আবার হবে না, তাহলে আবার $O(n^2)$ হয়ে যাবে কারণ আমরা যে সবসময় মাঝামাঝি ভাগ করতে পারব তার কি নিশ্চয়তা? এমনও তো হতে পারে প্রতিবার আমাদের ভাগ হচ্ছে এক মাথায়। সুতরাং আমাদের প্রথমবার n, এরপর $n-1$ এরকম করে মোট $O(n^2)$ বার কাজ করতে হবে। উপায়?

মনে কর আমরা $[i, j]$ এই সাব অ্যারে তে আছি। আমরা জিজ্ঞাসা করব i এর সংখ্যা কি unique? নাহলে j এর সংখ্যা কি unique? না হলে $i+1$ এর সঙ্খ্যাকি unique না হলে $j-1$ এর সংখ্যাকি ইউনিক? এরকম করে zigzag ভাবে জিজ্ঞাসা করব। তাহলে লাভটা কি? লাভ হল মনে কর এই প্রসেস শেষে আমরা আমাদের সাবঅ্যারেকে a আর b এই দুই সাইজে ভাগ করেছি। আর এই ভাগের জন্য আমাদের খরচ হয় $min(a, b)$ তাই না? কারণ দেখ $a = n-1$ আর $b = 1$ হলে আমরা দ্বিতীয় ধাপেই বের করতে পারব যে আমাদের সবশেষে ইউনিক সংখ্যা আছে। যেহেতু $a + b = j - i + 1$ সেহেতু আসলে a আর b এর সর্বনিম্ন মান $(j - i + 1)/2$ এর বেশি হবে না। তুমি একটু যদি কাগজে কলমে হিসাব করে দেখ দেখবা এ কাজের জন্য তোমার খরচ হবে $O(n)$. তবে এখনও একটা জিনিস বাকি আছে সেটা হল কোনো একটি সাবঅ্যারেতে কোনো একটি element ইউনিক কিনা তা কীভাবে বের করবে? সেটি সহজ, আমরা দেখব যে এর সমান বাম দিকে সর্বশেষ কোথায় আছে, আর ডানদিকে সবার আগে কোথায় আছে। যদি এ দুটির কোনটিই $[i, j]$ সীমানার মাঝে না থাকে তার মানে এটি ইউনিক। কোনো একটি সংখ্যার সমান সংখ্যা এর আগে কোথায় আছে বা এর পরে কোথায় আছে তা আমরা খুব সহজেই map ব্যবহার করে precalculate করে রাখতে পারি।

UVa 1609 Foul Play

সমস্যা: n টি ফুটবল টিম আছে (n এর মান 2 এর পাওয়ার হবে এবং সর্বোচ্চ 1024 হবে)। প্রতিটি টিম অন্য যেকোনো টিমের সাথে খেললে খেলার ফলাফল কি হবে তা তুমি জান। এই টিম গুলির মাঝে তোমার একটি টিম আছে। তুমি জান যে তোমার টিম অন্তত বাকি টিম গুলির অর্ধেককে হারাতে পারে। আর যাদের হারাতে পার না, তাদের হারাতে পারে এরকম কোনো টিমকে তোমার টিম আবার হারাতে পারে। টুর্নামেন্টের প্রথম রাউন্ডে $n/2$ টি খেলা হবে এবং খেলায় যারা জিতবে তারা দ্বিতীয় রাউন্ডে যাবে। একই ভাবে দ্বিতীয় রাউন্ডে $n/4$ টি খেলা হবে। এরকম করে একজন চ্যাম্পিয়ন হবে। তোমার কাজ হল- এমন ভাবে টুর্নামেন্ট ডিজাইন করা যাতে তোমার দল চ্যাম্পিয়ন হয়।

সমাধান: তুমি যদি প্রথমেই কে কার সাথে খেলবে সেটা ঠিক কর অর্থাৎ ট্রি এর leaf এ কে কোথায় আছে সেটা যদি প্রথমেই ঠিক করতে যাও তাহলে নানা রকমের সমস্যায় পরবা। যেমন মনে কর তোমার টিম যাদের হারাতে পারে না তারা হল B আর যাদেরকে তোমার টিম হারাতে পারে তারা হল A. তাহলে এরকম কিছু scenario চিন্তা করতে পার- A এর একটা টিম B এর সব টিম কে হারাতে পারে, কিন্তু সে আবার A এর বাকি টিম এর কাছে হারে। A এর অন্যরা B এর কোনো টিমকে হারাতে পারে না। এর মানে সেই special দল যেন অন্য কোনো A এর দলের কাছে যেন না হারে সেটা নিশ্চিত করে planning করতে হবে। এছাড়াও আরও অনেক কিছু খেয়াল রাখতে হবে। মনে কর B তে একটা টিম আছে যে কারও কাছে হারে না শুধু মাত্র A এর একটা টিম এর কাছে হারে। সুতরাং আমাদের খেয়াল রাখতে হবে A এর সেই টিম যেন আবার অন্য কারও কাছে হেরে না বসে। এরকম নানা জিনিস খেয়াল রেখে একবারে যদি তুমি দলের schedule ঠিক করতে চাও তাহলে সমাধান করা বেশ কঠিন হয়ে যায়।

এই ক্ষেত্রে অনেক সময় যেটা কাজে লাগে তাহল তোমার প্রব্লেম কে প্রতি রাউন্ডে ছোট করে ফেলা। যেমন মনে কর- এই সমস্যার ক্ষেত্রে বলা আছে যে, মোট দল n টা (যেটা একটা 2 এর power, 2^k). এর মাঝে তোমার একটা দল, কিছু A দল যাদের সংখ্যা মনে করি a, আর কিছু B দল যাদের সংখ্যা মনে করি b. তুমি A এর সবাইকে হারাতে পার। B এর সবাই তোমাকে হারাতে পারে। B এর প্রতিটি দল A তে আছে এমন কোনো একটি দলের কাছে হারে। $a \geq n/2 = 2^{k-1}$ আর $b < n/2 = 2^{k-1}$. আমাদের বলা আছে যে এরকম ক্ষেত্রে কোনো না কোনো ভাবে schedule করা সম্ভব যে তোমার দল শেষে জিতবে।

আমরা যদি এক রাউন্ড খেলা করাই তাহলে বাকি থাকে $n/2 = 2^{k-1}$ দল। আমরা কি প্রথম রাউন্ড এমন ভাবে schedule করতে পারি যেন, এই দ্বিতীয় রাউন্ডে তোমার দল থাকে, কিছু A দল

থাকে যেন তাদের সংখ্যা $a' \geq n/4 = 2^{k-2}$ হয় আর B দলের সংখ্যা $b' < n/4 = 2^{k-2}$ হয় আর যেসকল B দল এখনও বাকি আছে তাদেরকে হারাতে পারে এরকম কোনো এক দল যেন এখনও A তে বাকি থাকে। যদি আমরা এভাবে প্রথম রাউন্ড schedule করতে পারি তাহলে তো এটা আবার আমাদের কাছে নতুন কিন্তু ছোট একই রকম প্রব্লেম দাঁড়াবে তাই না? সুতরাং আমরা যদি এরকম করে একটি করে রাউন্ড schedule করতে পারি তাহলে সব রাউন্ড শেষে আমাদের দল জিতবে।

এখন প্রশ্ন হলে কীভাবে এই schedule করবা? একটু চিন্তা করলেই মনে হয় তা বের করে ফেলতে পারবা। আমরা A আর B নিয়ে কাজ শুরু করব। B থেকে একটা টিম নিব আর দেখব A এর কেউ তাকে হারাতে পারে কিনা। পারলে এদের দুজনকে একত্রে schedule করে দেব আর তাদের দুজনকে A আর B থেকে বাদ দেব। কিন্তু এমন তো হতে পারে যে B এর যেই দলকে আমি এখন schedule করতে চাচ্ছি তাকে A এর যারা হারাতে পারত তাদের ইতোমধ্যেই B এর অন্য দলের সাথে schedule করে ফেলেছি। কোনো সমস্যা নাই। সেই ক্ষেত্রে আমরা একে অন্য যেকোনো B দলের সাথে schedule করায়ে দেব। আর যদি সে একাই বাকি থাকে B দলের ভেতর থেকে তাহলে তাকে যেকোনো একটা A দলের সাথে schedule করে দেব। যেহেতু B দলের সংখ্যা A দলের সংখ্যার থেকে কম সেহেতু আমাদের এই প্রসেসে B এর সেট আগে ফাঁকা হয়ে যাবে। তখন আমরা আমাদের দলকে A এর যেকোনো দলের সাথে schedule করব আর বাকি A দলকে নিজেদের মাঝে যেকোনো ভাবে schedule করব। প্রশ্ন হল এভাবে schedule করলে আমাদের পরের রাউন্ড এ সব শর্ত মেনে দল গুলি থাকে কিনা।

আমাদের কাছে কি কি শর্ত ছিল?

- তোমার দলকে পরের রাউন্ডে উঠতে হবে।
- $b < n/2$ ছিল $b' < n/4$ হতে হবে।
- $a \geq n/2$ ছিল $a \geq n/4$ হতে হবে।
- প্রতিটি B' দলের জন্য A' সেটে এমন এক দলকে থাকতে হবে যে তাকে হারাতে পারে।

আমরা আগেই দেখেছি যে যেহেতু B দলের সংখ্যা A দলের সংখ্যার থেকে কম সেহেতু B দলের সকলের scheduling এর পর A দলে কেউ না কেউ বাকি থাকবে। সুতরাং আমরা আমাদের দলকে A দলের কারও না কারও সাথে schedule করছি। সুতরাং আমাদের দল পরের রাউন্ডে যাবেই।

মনে কর আমাদের প্রসেসে b দলের মাঝে t দল A এর সাথে schedule হয়। এর মানে বাকি $b - t$ দল নিজেদের মাঝে অথবা শেষ জন A দলের কারও সাথে schedule হয়। এর মানে ঐ t দলের কেউ পরের রাউন্ডের মুখ দেখবে না। বাকি $b - t$ দল নিজেদের মাঝে খেলবে আর তার ফলে এদের অর্ধেক জন পরের রাউন্ডে যাবে। যদি $b - t$ বিজোড় হয় তাহলে শেষ বিজোড় দল A এর কাউকে হারিয়ে পরের রাউন্ডে যাবে। ওদিকে $a - t$ টি A এর দল বাকি থাকবে (বা $a - t - 1$ যদি b বিজোড় হয়)। তাহলে কি $a' > b'$ হবে? খেয়াল কর যদি আমরা $a' > b'$ প্রমাণ করতে পারি তাহলে দুই আর তিন নাম্বার শর্ত কিন্তু একবারেই পুরন হয়ে যাবে। কারণ $a' + b' = n/2 - 1$ আর $a' > b'$ হলে $2b' < n/2 - 1$ বা $b' < n/4$. একই ভাবে আমরা অন্য শর্ত প্রমাণ করতে পারি। তাহলে আমাদের এখন দেখাতে হবে যে $a' > b'$. যেহেতু শুরুতেই $a > b$ ছিল, সুতরাং আমরা যদি ঐ t গুলিকে যদি schedule করে ফেলি তাহলেও $a - t > b - t$ থাকবে। এবং এই t দলগুলি খেলার পর সেখানে A দল গুলিই জিতবে, B দল গুলি সেসব খেলায় আর পরের রাউন্ডে যাবে না। খেয়াল কর এখানে t এর সংখ্যা কিন্তু কমপক্ষে 1 হবেই (যদি না হয় এর মানে B তে কেউ নাই, সেক্ষেত্রে তুমি কার সাথে কাকে schedule করলা তা কিন্তু ব্যাপার না, তোমার দল জিতবেই কারণ এখন আর কোনো B নেই)। এখন b বিজোড় হলে একটা B কে A এর সাথে খেলাতে হবে এবং সে ক্ষেত্রে B এর দল পরের রাউন্ডে যাবে। যেহেতু t থেকে একটা না একটা A এর দল পাবই আর এখানে একটা B এর দল পাচ্ছি সুতরাং এখন পর্যন্ত B এর দল সংখ্যা কিন্তু A এর দল সংখ্যা থেকে বেশি হয় নায়।

এখন এই বিজোড় দল খেলার পরও $a - t - 1 > b - t - 1$. তার মানে আমরা যদি এখন A আর B কে নিজেদের মাঝে খেলাই তাহলে কিন্তু A এর দলের সংখ্যা B এর দলের সংখ্যার থেকে কম হবে না বরং বেশি হতে পারে। আমরা এখানে প্রমাণ করলাম $a' \geq b'$. কিন্তু তুমি এখন যদি চিন্তা কর কখন $a' = b'$ হতে পারে তখন একটু যদি case analysis কর তাহলে বুঝবে যে $a' = b'$ হতে পারে না। কেন? দেখ, $a' = b'$ তখনই হবে যদি $t = 1$ হত আর $b - t$ তে বিজোড় সংখ্যক দল থাকত। শেষ

বিজোড় দলকেও মনে কর schedule করলা। বাকি থাকল $b - t - 1$ সংখ্যক জোড় দল। ওদিকে তোমার দল আর $a - t - 1$ সংখ্যক বিজোড় দল বাকি থাকবে। কারণ B এর গ্রুপ আর তুমি মিলে বিজোড়। যেহেতু জোড়ায় জোড়ায় খেলা হবে সেহেতু A দলের বাকি দের সংখ্যা তো বিজোড় হবেই। মনে কর তুমি A এর কারও সাথে খেললে। তাহলে A এর বাকি দের সংখ্যা B দের বাকি দের সংখ্যার খুব জোড় সমান হবে। কিন্তু আসলেই কি সমান হতে পারবে? সমান হলে তো পরের রাউন্ডে তোমার দলের সাথে খেলার আর কাউকে পাওয়া যাবে না। এর মানে আসলে A তে B এর থেকে আরও দুই জন বেশি আছে। এর মানে $(a - t - 1)/2 > (b - t - 1)$. প্রমাণটা একটু জটিল হয়ে গেল। হয়তো আরও সহজ প্রমাণ আছে। কিন্তু আসলে এতো details এ প্রমাণ করার দরকার নাই। তুমি যদি convince হউ যে এটা কাজ করছে তাহলেই হল। আর তুমি যদি চাও তাহলে কষ্ট করে প্রমাণ করতেই পার। সত্যি কথা বলতে এই প্রমাণ আমি করার জন্য করেছি। এবং এই প্রমাণ আমি নিজে পরে বুঝতে বেশ কষ্ট হচ্ছে। করার কিছু নেই। এধরনের প্রমাণ এরকম জটিলই হয় সাধারণত। এজন্য সবচেয়ে ভাল হয় নিজেরাই এরকম case analysis করে করে প্রমাণ করা। আর যদি কেউ একান্তই প্রমাণ দেখতে চাও তাহলে ঠাণ্ডা মাথায় এখানে লিখা প্রমাণ কয়েক দিন ধরে কয়েক বার করে পড়, আশা করি বুঝবা।

UVa 1442 Cave

সমস্যা: একটি গুহা আছে। গুহাটাকে n unit লম্বা হিসাবে কল্পনা করতে পার ($n \leq 1,000,000$)। প্রতিটি unit এ কিছু পরিমান ছাদ আর কিছু পরিমান মাটি আছে। প্রতিটি স্থানে সমুদ্র পৃষ্ঠ হতে কত উপরে মাটি আর কত উপরে ছাদ আছে তা বলা আছে। এখন তুমি এই গুহাতে কিছু পরিমান তরল সংরক্ষন করতে পার। খেয়াল রাখতে হবে তোমার তরল যেন ছাদ অতিক্রম না করে। আর যেহেতু তুমি তরল রাখবা- তরল কিন্তু চতুর্দিকে গড়িয়ে যায়। সুতরাং খেয়াল রাখতে হবে যে তুমি এক জায়গায় তরল রাখলে আর তা গড়িয়ে পাশের কোনো ছাদ অতিক্রম করল- এটা যেন না হয়। বলতে হবে সর্বোচ্চ কি পরিমান তরল তুমি এই গুহাতে সংরক্ষন করতে পারবে। আর এটা বলা আছে যে তরল গড়িয়ে গুহার সীমানা অতিক্রম করতে পারবে না। অর্থাৎ দুদিকে অনেক উচু মাটি আছে বলে মনে করতে পার। (ভাল হয় তোমরা যদি original problem টা পড়)।

সমাধান: নানা ভাবেই এই সমস্যা সমাধান করা যায়। এর মাঝে একটি উপায় হল প্রতিটি স্থানের জন্য এটা বের করা যে এই স্থানে কত উচু পর্যন্ত তরল রাখা যাবে যেন এটি আমাদের বামের বা ডানের কোনো ছাদকে অতিক্রম না করে। আমরা দুইভাগে এটা করব। প্রথম ভাগে আমরা বের করব আমরা যদি বামের কোনো ছাদকে অতিক্রম করতে না চাই তাহলে আমাদের তরলের উচ্চতা কত হবে। এর পরের ভাগে আমরা বের করব আমরা যদি ডানের কোনো ছাদকে অতিক্রম করতে না চাই তাহলে আমাদের তরলের উচ্চতা কত হবে। এই দুইটি সংখ্যা বের হয়ে গেলে তো আমাদের সমস্যা সমাধান হয়ে যাবে।

তাহলে আমরা এখন বের করতে চাই আমাদের তরলের উচ্চতা কত হলেও বামের কোনো ছাদকে অতিক্রম করবে না। এজন্য আমরা বাম হতে ডানে আসব, কিছুটা linear scan এর মত। মনে কর আমরা এখন i তম স্থানে আছি আর জানি যে বামে কোনো ছাদকে অতিক্রম না করতে হলে সর্বোচ্চ তরলের height কত হতে পারে। এখন আমরা i তম স্থানে আসলাম। এই তরলের height limit কে আমাদেরকে আপডেট করতে হবে। কয়েকটা কেস হতে পারে। মনে কর বর্তমান স্থানের ছাদের উচ্চতা h, মেঝের উচ্চতা f আর আমাদের বর্তমান limit L. তিন ধরনের কেস হতে পারে।

- $L \geq h > f$. এখন এই i তম স্থানের তরলের উচ্চতা L অতিক্রম তো করতে পারবেই না, উপরন্তু h ও অতিক্রম করতে পারবে না। সেজন্য L এখন পরিবর্তন হয়ে h হয়ে যাবে।
- $h > L > f$. এটা সবচেয়ে সহজ কেস। এক্ষেত্রে আমাদের তরলের উচ্চতার লিমিট L ই থাকবে।
- $h > f \geq L$. এটা বেশ ট্রিকি কেস। এক্ষেত্রে আমাদের মেঝে আগের লিমিট L এর থেকেও উচুতে আছে। এর মানে হল এই স্থানে কোনো তরল থাকতে পারবে না। কারণ থাকলেই তো বাম দিকে চলে যাবে, আর বাম দিকে যাওয়ার মানেই L কে violate করা। কিন্তু একটা জিনিস

খেয়াল কর, এই স্থানের ডান দিকে যারা আছে তারা কিন্তু এখন f উচ্চতা পর্যন্ত তরল রাখতে পারবে। কারণ এই স্থান হতে ডান দিকের তরলের উচ্চতা যতক্ষণ না f অতিক্রম করবে ততক্ষণ তারা i এর বামে যেতে পারবে না আর L কেও violate করতে পারবে না। তাহলে এই কেসে যা করতে হবে তাহল $L = f$.

সুতরাং আমরা যদি উপরের এই প্রসেস বাম আর ডান দুই দিক থেকে করি তাহলে প্রত্যেক স্থানের জন্য আমরা L পাব। আর L পেলে তো আমরা তা থেকে মেঝের উচ্চতা বাদ দিলেই ঐ স্থানে কি পরিমান তরল থাকতে পারবে তা পেয়ে যাব।

UVa 12265 Selling Land

সমস্যা: একটি $R \times C$ গ্রিড আছে $(R, C \leq 1,000)$। গ্রিডের বিভিন্ন সেলে পানি আছে। তুমি এক বা একাধিক সাবগ্রিড মার্ক করতে পার যেন তার ভেতরে কোনো সেলে পানি না থাকে। তবে খেয়াল রাখতে হবে যে একাধিক সাবগ্রিডের নিচের-ডান কোণা যেন একই না হয়। তোমার লক্ষ্য এসব সাবগ্রিডের পরিধির যোগফল যেন সর্বোচ্চ হয়। বলতে হবে সাবগ্রিড সমুহের পরিধি কত কত।

সমাধান: এরকম সমস্যার ক্ষেত্রে আমাদেরকে maximum 0 area subrectangle এর $\mathrm{O}(n^2)$ সমাধানের টেকনিক খাটাতে হয়। এই টেকনিক না জানলে আগে তা পড়ে আসো। ওখানে আমরা কি করেছিলাম? আমরা প্রথমে subrectangle এর base ঠিক করেছিলাম। এরপর প্রতিটি কলাম একের পর গিয়েছিলাম এবং তা stack এ রেখে প্রসেস করেছিলাম। মনে কর আমরা একটি base ঠিক করলাম, অর্থাৎ একটি রো ঠিক করলাম যে যেসকল rectangle এর base এই রো তে আছে তাদের উত্তর আমরা এখন বের করব। এবার আমরা বাম হতে ডানে যাই। প্রতিটি কলামে গিয়ে আমাদের জানতে হবে এই কলামে আমাদের ঠিক করা base এর থেকে কত উচু পর্যন্ত পানি ছাড়া সেল আছে। এই সংখ্যা আমরা খুব সহজেই $\mathrm{O}(n^2)$ এ precompute করে রাখতে পারি। এখন আমরা stack এ এই height গুলি রেখে প্রসেস করব। যদি stack ফাঁকা হয় তাহলে আমরা stack এ ঢুকিয়ে রাখব বর্তমান কলাম আর তার height. যদি ইতোমধ্যেই stack এ কেউ থাকে তাহলে দেখব stack এর উপরে যে আছে তার height কি আমার বর্তমান কলামের height থেকে বড় কিনা। যদি বড় হয় তাহলে তো লাভ নাই তাই না? কারণ ওত বড় height এর rectangle বর্তমান কলাম থেকে বানানো সম্ভব না। সেজন্য তাকে ধরে ফেলে দেব। এরকম করে stack এর উপরে যত জন আমার বর্তমান কলামের থেকে বেশি height এর আছে তাদের ধরে ফেলে দিতে হবে। এই কাজ করার পর আমাদেরকে stack এ বর্তমান height প্রবেশ করাতে হবে। তবে এই height শুরু হবে কই থেকে? সর্বশেষ যাকে তুলে ফেলেছ সে যেখান থেকে শুরু হয়েছিল সেখান থেকে। এভাবে আমাদেরকে stack প্রসেস করতে হবে। আর বর্তমান কলামের প্রসেস করা হয়ে গেলে তুমি আসলে বর্তমান কলামের জন্য সবচেয়ে বড় perimeter এর rectangle কত সেটাও বের করে ফেলতে পারবে। দেখ বর্তমান base আর কলাম কিন্তু fixed. সেহেতু তোমাকে চেষ্টা করতে হবে তোমার stack এ থাকা কোন কলামের $r - c$ সবচেয়ে বেশি (c মানে stack এর কোনো একটি element কোন কলাম থেকে শুরু হয়েছিল তা, আর r মানে তার উচ্চতা)। কারণ আমাদের perimeter এর ফর্মুলা তো $c' - c + r$ এর মত (মত বললাম কারণ এর সাথে হয়তো 1-2 যোগ হিসাবে আছে বা কিছু সংখ্যা গুন হিসাবে থাকতে পারে)। এই কাজ stack এ যখন কোনো একটি কলাম প্রবেশ করাবে তার সাথে সাথে বর্তমান সর্বোচ্চ $r - c$ ও রাখলেই হয়ে যাবে। অর্থাৎ stack এর প্রতি element এ লিখা থাকবে যে stack এ তার নিচে যারা আছে (সে সহ) তাদের সবার মাঝে সর্বোচ্চ $r - c$ এর মান কত।

১০.১ অনুশীলনী

১০.১.১ সমস্যা

Simple

- UVa 1149 Bin Packing
- UVa 12545 Bits Equalizer
- UVa 1611 Crane
- UVa 1617 Laptop
- UVa 11536 Smallest Sub-Array
- UVa 1610 Party Games
- UVa 11491 Erasing and Winning
- UVa 1616 Caravan Robbers
- UVa 1618 Weak Key

Easy

- UVa 177 Paper Folding
- UVa 1612 Guess
- UVa 1615 Highway
- UVa 1619 Feel Good
- UVa 11925 Generating Permutations
- UVa 1153 Keep the Customer Satisfied
- UVa 10570 Meeting with Aliens
- UVa 1312 Cricket FieldCircles

Medium

- UVa 1613 K-Graph Oddity
- UVa 1620 Lazy Susan
- UVa 1622 Robot
- UVa 10366 Faucet Flow
- UVa 1580 Pirate Chest
- UVa 1614 Hell on the Markets
- UVa 1621 Jumping Around
- UVa 1623 Enter The Dragon
- UVa 11175 From D to E and back
- UVa 1624 Knots

অধ্যায় ১১

Geometry

UVa 190 Circle Through Three Points

সমস্যা: 2d তে তিনটি বিন্দু দেওয়া আছে যারা একই সরল রেখার উপর অবস্থিত না। তোমাকে এই তিনটি বিন্দু দিয়ে যায় এরকম একটি বৃত্তের সমীকরণ প্রিন্ট করতে হবে। এই সমস্যার ক্ষেত্রে বৃত্তকে দুইভাবে প্রিন্ট করতে হবে। এক- $(x-h)^2+(y-k)^2=r^2$ এর ফর্মে, আরেকটা হল- $x^2+y^2+cx+dy-e=0$ এই ফর্মে।

সমাধান: নানা উপায়ে এই সমস্যা সমাধান করা যায়। একটা জিনিস খেয়াল কর- তুমি যদি কোনো একটি ফর্মে বৃত্তের সমীকরণ বের করতে পার তাহলে তাকে অন্য ফর্মে পরিনত করা আসলে কোনো ব্যাপারই না। তাই আমরা চেষ্টা করব যেকোনো একটি ফর্মে সমীকরণ বের করতে।

প্রথম সমাধান- আমরা $x^2+y^2+cx+dy-e=0$ এই সমীকরণে c,d,e এর মান বের করতে চেষ্টা করব। আমাদের কাছে তিনটি বিন্দু আছে। সেই তিনটি বিন্দুর x আর y এর মান গুলি যদি আমরা এই সমীকরণে বসাই তাহলে আমরা তিনটি equation পাব যেখানে তিনটি unknown মান থাকবে (c,d,e). আমরা খুব সহজেই এই তিনটি equation সমাধান করে তিনটি variable এর মান বের করে ফেলতে পারি। এজন্য তোমরা determinate এর পদ্ধতি ব্যবহার করতে পার, gaussian elemination ব্যবহার করতে পার, হাতে হাতে সমাধান বের করতে পার।

দ্বিতীয় সমাধান- আমরা $(x-h)^2+(y-k)^2=r^2$ এই সমীকরণে h,k,r এর মান সমাধানের চেষ্টা করব। প্রথমেই আমরা তিনটি বিন্দুর x আর y এর মান বসিয়ে তিনটি আলাদা সমীকরণ বের করে ফেলি। এই তিনটি সমীকরণের format হবে $h^2+ah+k^2+bk+c-r^2=0$ যেখানে a,b,c হল তিনটি constant (মানে আমরা যদি x আর y এর মান বসাই তাহলে a, b, c এর জায়গায় কিছু constant মান ওয়ালা সমীকরণ পাব)। আমরা যদি আমাদের একটি সমীকরণকে বাকি দুইটি সমীকরণ হতে বিয়োগ করি তাহলে আমরা দুইটি সমীকরণ পাব যাদের ফরম্যাট হবে $ah+bk+c=0$ টাইপের। অর্থাৎ দুটি সমীকরণ আর দুটি variable (h,k). খুব সহজেই আমরা h আর k এর মান বের করে ফেলতে পারব। এরপর এদের মান কোনো একটি সমীকরণে বসিয়ে আমরা r এর মান বের করে ফেলতে পারব। তাহলেই হবে।

তৃতীয় সমাধান- আমরা ত্রিকোণমিতির $R=\frac{abc}{4A}$ এই সুত্র খাটিয়ে খুব সহজেই প্রদত্ত তিনটি বিন্দুর ত্রিভুজের পরিবৃত্তের ব্যাসার্ধ R বের করে ফেলতে পারব (এখানে a, b, c হল ত্রিভুজের তিনটি বাহুর দৈর্ঘ্য আর A হল ত্রিভুজের ক্ষেত্রফল)। বৃত্তের ব্যাসার্ধ যখন জেনে গেছি তখন আমরা প্রদত্ত বিন্দু তিনটির যেকোনো দুটিতে এই R ব্যাসার্ধের বৃত্ত আঁকতে পারি যারা সর্বোচ্চ দুইটি বিন্দুতে পরস্পরকে ছেদ করবে। এই দুটি বিন্দুর কোনো একটিই হবে আমাদের সমাধান বৃত্তের কেন্দ্র। সুতরাং আমরা যদি দুটি বৃত্তের ছেদ বিন্দু বের করতে পারি তাহলেই এই সমস্যা সমাধান হয়ে যায়। দুটি বৃত্তের ছেদ বিন্দু কীভাবে বের করতে হয় তা আমরা প্রোগ্রামিং কন্টেস্ট বইএ দেখে এসেছি।

UVa 378 Intersecting Lines

সমস্যা: দুটি সরলরেখা দেওয়া আছে। ইনপুটে প্রতিটি সরলরেখার জন্য তাদের উপর অবস্থিত দুটি আলাদা বিন্দু দেওয়া থাকবে। খেয়াল কর, এখানে ইনপুট কিন্তু সরল রেখা, সরল রেখাংশ না। তোমাকে বলতে হবে এই দুটি রেখা কীভাবে পরস্পরকে ছেদ করে। তিন রকম কাহিনী হতে পারে- তারা পরস্পরকে ছেদ করে না (যখন তারা parallel হয়), তারা দুজন একই সরল রেখা নির্দেশ করে, অথবা তারা পরস্পরকে একটি বিন্দুতে ছেদ করে। বলতে হবে এই তিন ঘটনার কোনটা ঘটে প্রদত্ত ইনপুটের ক্ষেত্রে। যদি তৃতীয় ঘটনা হয়ে থাকে, তাহলে ছেদ বিন্দুর coordinate ও দিতে হবে।

সমাধান: মনে কর প্রথম রেখার বিন্দু দুটি A ও B. আর দ্বিতীয় রেখার বিন্দু দুটি C ও D. প্রথমে আমরা প্রথম দুটি কেস হ্যান্ডেল করি। আমরা ABC আর ABD এই দুটি ত্রিভুজের signed ক্ষেত্রফল বের করি। যদি দেখি এই দুটি ক্ষেত্রফলই শূন্য, এর মানে এটি দ্বিতীয় কেসে পড়ে। কারণ ABC ত্রিভুজের ক্ষেত্রফল শূন্য মানে A, B, C একই সরল রেখায়। আবার A, B আর D ও যদি একই রেখায় হয় তার মানে A, B, C আর D চারটি বিন্দুই আসলে এক রেখায়।

যদি এই দুটি ত্রিভুজের signed ক্ষেত্রফল শূন্য না হয়, কিন্তু এদের signed মান যদি সমান হয় তার মানে AB আর CD সমান্তরাল। এখানে signed ক্ষেত্রফল তুলনা করা জরুরি। কারণ এমন তো হতেই পারে যে AB এর দুই পাশে C আর D কিন্তু তারা AB এর সাথে সমান ক্ষেত্রফল তৈরি করে। কিন্তু সেক্ষেত্রেতো AB আর CD পরস্পরকে ছেদ করে। সুতরাং আমাদেরকে signed ক্ষেত্রফল তুলনা করতে হবে। আর ক্ষেত্রফল সমান হলে যে তারা সমান্তরাল হয় এই উপপাদ্য তো স্কুলের বইএ আছে তাই না?

আর যদি উপরের কোনো কেস না হয় তাহলে তারা পরস্পরকে ছেদ করে। দুটি রেখা দেওয়া আছে, তাদের ছেদ বিন্দু কীভাবে বের করতে হবে তা প্রোগ্রামিং কন্টেস্ট বইয়ে আমরা দেখেছি। তোমরা চেষ্টা কর vector বা প্যারামেট্রিক ফর্মের মাধ্যমে এর সমাধান করার।

UVa 460 Overlapping Rectangles

সমস্যা: দুটি axis parallel আয়তক্ষেত্র দেওয়া আছে (ইনপুটে প্রতিটি আয়তক্ষেত্রের দুই কোণার coordinate দেওয়া থাকবে)। বলতে হবে তাদের intersection (intersection এর ক্ষেত্রফল শূন্য এর থেকে বেশি হতে হবে) আছে কিনা। থাকলে সেই intersection আয়তক্ষেত্রটির দুই কোণার coordinate প্রিন্ট করতে হবে।

সমাধান: এই সমস্যাটা খুব কঠিন মনে হতে পারে। তুমি যদি case by case চিন্তা কর তাহলে নিঃসন্দেহে এটা খুব কঠিন সমস্যা। কিন্তু এর খুব সহজ একটা সমাধান আছে। মনে কর প্রথম আয়তক্ষেত্রের নিচের বাম কোণার coordinate (Ax_1, Ay_1) আর উপরের ডান কোণার coordinate (Ax_2, Ay_2) ($Ax_1 < Ax_2$ আর $Ay_1 < Ay_2$). একই ভাবে দ্বিতীয় আয়তক্ষেত্রকে B দ্বারা প্রকাশ করি। তাহলে intersection আয়তক্ষেত্রকে আমরা যদি C দ্বারা প্রকাশ করি তাহলে আমরা লিখতে পারি $Cx_1 = max(Ax_1, Bx_1)$, $Cy_1 = max(Ay_1, By_1)$, $Cx_2 = min(Ax_2, Bx_2)$, $Cy_2 = min(Ay_2, By_2)$. কেন? মনে কর A আর B এর শুধু বাম বাহু চিন্তা কর। ধর A এর বাম বাহু B এর বাম বাহুর বামে আছে। তাহলে তুমি একটু চিন্তা করে দেখ তো যে যদি A আর B এর intersection থাকে তাহলে সেই intersection rectangle এর বাম বাহু কোথায় থাকবে? সহজ, A আর B এর বাম বাহুর মাঝে যেটা ডানে আছে সেখানে, মানে B এর বাম বাহুতে। কেন? কারণ A এর বাম বাহু হতে B এর বাম বাহু পর্যন্ততে তো কোনো intersection থাকতে পারে না, তাই না? একই ভাবে আমরা C এর নিচ, উপর, আর ডান বাহু বের করতে পারি। কিন্তু আমরা কীভাবে বুঝব যে আমাদের overlap হয়েছে কিনা? সহজ, তুমি C এর বাম ডান উপর নিচ সব বাহু বের করে ফেলেছ। এখন চেক করে দেখ যে, C এর নিচের বাহু কি আসলেই C এর উপরের বাহু থেকে নিচে কিনা। যদি নাহয় এর মানে C বলে আসলে কিছু নাই। একইভাবে তোমাকে দেখতে হবে C এর বাম বাহু কি আসলেই C এর ডান বাহু থেকে বামে কিনা। খেয়াল কর, কিছুক্ষণ আগে আমরা বের করেছি যে C এর যদি বাম বাহু থাকে তাহলে সেটা কোথায়, C এর ডান বাহু কোথায় এরকম উপর নিচও বের

করেছি। এরপর আমাদেরকে চেক করে দেখতে হবে যে, আমরা যে বের করেছি C এর ডান বাম উপর নিচ কোথায়, তা কি আসলেই হতে পারে? যদি হতে না পারে তার মানে C বলতে কেউ নাই। আর যদি হতে পারে তাহলে তো আমরা তার দুই কোণার coordinate পেয়ে গেছি। এখন তোমাদের কাজ হবে কাগজ কলমে কিছু ছবি এঁকে এই প্রসেস যে কাজ করে তা নিজেকে convince করা। বিশেষ করে, যখন কোনো intersection থাকে না তখন এই পদ্ধতি কেন কাজ করে তা বুঝাটাই গুরুত্বপূর্ণ।

UVa 10175 Sphere

সমস্যা: দুটি গোলক এর ব্যাসার্ধ r_1, r_2 এবং তাদের কেন্দ্রদের মধ্যবর্তী দুরত্ব d দেওয়া আছে। বলতে হবে তাদের মিলিত জিনিসের আয়তন আর পৃষ্ঠতলের ক্ষেত্রফল কত। বলা আছে যে $d > r_1, r_2$ এবং $d < r_1 + r_2$ হবে। সেই সাথে এই বস্তুটির ভর বলা আছে। বলতে হবে এই বস্তুটি একটি তরল পদার্থে ডুববে না ভাসবে। বস্তুটির এবং তরলের মধ্যের specific gravity দেওয়া আছে।

সমাধান: আমরা যদি বস্তুটির আয়তন বের করতে পারি তাহলে তাকে যদি আমরা specific gravity এর সাথে গুন করি তাহলে আমরা একই আয়তনের তরলের ভর পাব। আর আমরা এই তরলের ভর আর আমাদের বস্তুর ভর তুলনা করলেই বুঝতে পারব যে আমাদের বস্তুটি তরলে ভাসবে না ডুববে।

d এর লিমিটের দিকে তাকাও। কি বুঝলা? দুটি গোলকের কেন্দ্র একটি আরেকটির ভেতরে থাকবে না। এখন দুটি গোলক পরস্পরকে ছেদ করে যেই বৃত্ত তৈরি করে তার ভেতর দিয়ে একটি তল আঁকি। এর ফলে এই তলের দুই দিকে দুটি গোলকের অংশ থাকবে। মনে কর তলের বাম দিকে আছে A গোলক আর ডান দিকে B গোলক। আসলে তলের বাম দিকেও B গোলকের কিছু অংশ থাকবে, তবে তা সম্পূর্ণ ভাবে A গোলকের ভেতরে থাকবে। এই যে তলের বাম দিকে B গোলকের কিছু অংশ যে A গোলকের ভেতরে আছে তাকে B গোলকের spherical cap বলে। তোমরা চাইলে নেটে সার্চ দিলেই এর আয়তন আর পৃষ্ঠতলের ক্ষেত্রফল বের করতে পারবা। অথবা চাইলে আমরা হাতে হাতে calculus ব্যবহার করে এই মানগুলির ফর্মুলা বের করতে পারি। আমরা যদি এই spherical cap এর আয়তন বের করতে পারি তাহলে পুরো গোলকের আয়তন হতে তা বিয়োগ করে কোনো গোলকের যেই অংশ বাইরে আছে তার আয়তন বের করতে পারব।

আমরা ধরে নিতে পারি আমাদের দেওয়া দুটি গোলকের একটি আছে মূল বিন্দুতে আরেকটি আছে $(d, 0, 0)$ তে। এবার তুমি XY তল কল্পনা কর। এখানে P, Q কেন্দ্র বিশিষ্ট দুটি বৃত্ত দেখতে পাবে যাদের ব্যাসার্ধ r_1, r_2 আর তাদের মধ্যবর্তী দুরত্ব d. মনে কর তারা A আর B বিন্দুতে ছেদ করে। AB লাইন টান। এটা তো জানই যে AB লাইন বৃত্তের কেন্দ্র দুটির সংযোগকারি লাইন PQ এর উপর লম্ব। মনে কর PQ আর AB এর ছেদবিন্দু D. আমরা PD আর QD বের করব। আমরা যদি PD জানি তাহলে P গোলকের যেই অংশ Q গোলকের ভেতরে আছে তার আয়তন এবং পৃষ্ঠতলের ক্ষেত্রফল বের করে ফেলতে পারব। একটা জিনিস, তোমরা হয়তো খেয়াল করেছ যে আমি P আর Q কে একবার বৃত্ত, একবার গোলক আর একবার তাদের কেন্দ্র বলছি। কেন্দ্র বলাতে আশা করি আমি কোনো অপরাধ করে ফেলি নাই। তবে অনেকে গোলক আর বৃত্ত এই দুটি ব্যবহার করায় নাখোশ হতে পার। আমাদেরকে আসলে P আর Q হতে তাদের ছেদ তলের দুরত্ব (বা AB লাইনের দুরত্ব) বের করতে হবে।

যাই হোক। তাহলে আমাদের কাছে $PA = r_1$ আছে, $QA = r_2$ আছে আর $PQ = d$ আছে। আমাদের বের করতে হবে PD এর মান। আমরা যদি APQ ত্রিভুজের উচ্চতা AD (PQ কে ভুমি ধরে) বের করতে পারি তাহলেই কিন্তু হয়ে যায় কারণ $PD^2 = PA^2 - AD^2$. যেহেতু APQ ত্রিভুজের তিনটি বাহু জানা আছে সেহেতু তোমরা হেরনের ফর্মুলা ব্যবহার করে সহজেই ত্রিভুজের ক্ষেত্রফল বের করতে পারবা। আবার ত্রিভুজের ভুমি উচ্চতা এর ফর্মুলা ব্যবহার করে উচ্চতা AD এর মান বের করতে পারবা যেহেতু আমাদের ভুমি PQ জানা। অথবা চাইলে PAD আর QAD ত্রিভুজ দুটির পিথাগোরাসের ফর্মুলা যদি লিখ সেখান থেকেও PD এর মান বের করতে পারবা। মোট কথা আমরা PD এর মান বের করতে পারব। খেয়াল কর PD হল কেন্দ্র হতে cap এর তলের দুরত্ব।

তাহলে আমরা আমাদের সমস্যাকে দুটি গোলক হতে একটি গোলকে পরিবর্তন করতে পারি। আমাদের নতুন সমস্যা হবে- আমাদের একটি গোলক দেওয়া আছে যার ব্যাসার্ধ r, এবং কেন্দ্র হতে

এর cap এর তলের দুরত্ব h দেওয়া আছে। বলতে হবে এই cap এর আয়তন কত আর পৃষ্ঠতলের ক্ষেত্রফল কত।

এবার ক্যালকুলাস এর পালা। আমরা আমাদের কাজ সহজ করার জন্য ধরে নেব গোলকটি মূল বিন্দুতে আছে এবং আমরা $x = h$ হতে $x = r$ এই অংশটুকুর আয়তন এবং পৃষ্ঠতলের ক্ষেত্রফল বের করতে চাই। এখানে পৃষ্ঠতল বলতে আমরা শুধু গোলকের বাইরের অংশটুকুর কথা বলছি।

যেহেতু আমরা $x = h$ হতে $x = r$ পর্যন্ত অংশের আয়তন জানতে চাই সেহেতু আমাদের integration এর লিমিট হবে h থেকে r. আমরা এই h থেকে r অংশকে ছোট ছোট dx অংশে ভাগ করি। এই ছোট ছোট অংশকে আমরা cylinder হিসাবে কল্পনা করতে পারি। কেন? আমাদের গোলকের কেন্দ্র মূল বিন্দুতে আছে। এখন তুমি $x = 0$ আর $x = 1$ দুইটি plane আঁক। এরপর এই দুই plane কে খুব কাছে আনো। Plane এর ভেতরের এই অংশটুকু দেখতে cylinder এর মত তাই না? কারণ এর বাম পাশের অংশ একটি বৃত্ত, ডান পাশের অংশ একটি বৃত্ত। যদি এই দুই plane খুব কাছে থাকে তাহলে আমরা বলতে পারি এই বৃত্ত দুটির ব্যাসার্ধ প্রায় সমান, আর সমান হলেই তো এটি একটি cylinder হয় তাই না? যাই হোক তাহলে একটি সিলিন্ডারের উচ্চতা হবে dx আর ব্যাসার্ধ হবে y. কোনো একটি x এ y এর মান হবে $\sqrt{r^2 - x^2}$. তাহলে আমাদের integration এর ফর্মুলা হবে $\int_h^r f(x)$ যেখানে $f(x) = \pi(r^2 - x^2)\,dx$ হবে যদি আমরা আয়তন বের করতে চাই আর $f(x) = 2\pi\sqrt{r^2 - x^2}\,dx$ হবে যদি আমরা পৃষ্ঠতলের ক্ষেত্রফল বের করতে চাই। আশা করি এরকম integration কীভাবে করতে হয় তা তোমাদের জানা আছে? ভুলে গিয়ে থাকলে বা না জানা থাকলে calculus বই পড়তে হবে!

UVa 10522 Height to Area

সমস্যা: একটি ত্রিভুজের তিন কোণা হতে তাদের বিপরীত বাহুর উপর আঁকা লম্বের উচ্চতা দেওয়া আছে। তোমাকে ত্রিভুজের ক্ষেত্রফল বলতে হবে। যদি এমন কোনো ত্রিভুজ সম্ভব না হয় যার এই তিনটি উচ্চতা আছে, তাহলে বল যে ইনপুটে ভুল আছে।

সমাধান: এরকম জ্যামিতি সমস্যার ক্ষেত্রে কোনো বাধা ধরা নিয়ম থাকে না। যে যেভাবে পারে সমাধান করতে পারে। হয়তো ত্রিভুজের উচ্চতা দেওয়া আছে, ত্রিভুজের ক্ষেত্রফল বের করতে হবে- এরকম ফর্মুলা আছে কিন্তু কন্টেস্ট সময়ে তুমি তো আর ইন্টারনেট পাবা না। সুতরাং নিজেকে সমাধান করতে হবে।

মনে কর H_a, H_b আর H_c হল যথাক্রমে A, B, আর C হতে বিপরীত বাহুর উপর আঁকা লম্বের উচ্চতা। আরও মনে করা যাক a, b, c হল যথাক্রমে A, B আর C বিন্দুর বিপরীত বাহুর দৈর্ঘ্য। আমরা বলতে পারি $a \times H_a$, $b \times H_b$ আর $c \times H_c$ সমান, কারণ এরা প্রত্যেকে ত্রিভুজের খেত্রফলের দ্বিগুন। এখান থেকে আমরা বলতে পারি $a : b : c$ এর মান $1/H_a : 1/H_b : 1/H_c$ এর সমান বা অন্য ভাবে এমন একটি k আছে যেন $a = k/H_a, b = k/H_b, c = k/H_c$. এখন আমরা হেরনের ফর্মুলা ব্যবহার করে ত্রিভুজের ক্ষেত্রফল বের করতে পারি। এর পর $\frac{a \times H_a}{2}$ এর সাথে সেই ক্ষেত্রফলকে একত্রে equation এ বসালে আমরা k এর মান বের করতে পারব। আর k এর মান বের করলে হেরনের ফর্মুলায় সেই মান বসিয়ে আমরা ত্রিভুজের ক্ষেত্রফল বের করতে পারব! আর কখন ইনপুট ভুল? হেরনের ফর্মুলায় তো একটা square root আছে। তুমি যদি দেখ এই square root এর ভেতরে negative সংখ্যা চলে আসছে তার মানে হবে ইনপুট ভুল। অনেক সময় দেখা যায় square root এর ভেতরে একাধিক টার্ম negative চলে আসে আর তারা গুন করে positive হয়ে যায়। সেসব ক্ষেত্রে তোমরা উত্তর বের করে দেখবে যে এই উত্তরের ত্রিভুজের জন্য কি আসলেই ইনপুটের height গুলি পাওয়া সম্ভব কিনা।

UVa 10709 Intersection is not that Easy

সমস্যা: দুটি সরলরেখা বা দুটি সরল রেখাংশ বা একটি সরল রেখা আর আরেকটি সরল রেখাংশ দেওয়া আছে। বলতে হবে তাদের মধ্যে সবচেয়ে কম দুরত্ব কত।

সমাধান: Vector বা প্যারামেট্রিক ফর্মের মাধ্যমে দুটি রেখা ছেদ করে কিনা বা ছেদ করলে সেই ছেদ বিন্দু কি তা জানার যেই পদ্ধতি সেই পদ্ধতি খাটিয়ে আমরা সহজেই বের করতে পারি প্রদত্ত রেখা বা রেখাংশ দের মাঝে intersection হয় কিনা বা হলে intersection বিন্দুটি কোনটি।

এখন আরও কিছু কেস বাকি থাকে। একটি কেস হল প্রদত্ত চারটি বিন্দুই একই সরল রেখার উপর অবস্থিত। এই ক্ষেত্রে যদি ইনপুটের কেউ সরল রেখা হয় তার মানে তাদের মাঝে দুরত্ব 0. আর যদি তা না হয়, তার মানে আমাদের ইনপুট দুটি সরল রেখাংশ। সেক্ষেত্রে দেখতে হবে কোনো রেখাংশের কোনো একটি প্রান্ত বিন্দু অন্যটির ভেতরে আছে কিনা, থাকলে 0. না থাকলে আমরা জানি এক রেখাংশের এক প্রান্ত আর আরেক রেখাংশের আরেক প্রান্তের মাঝে সবচেয়ে কম দুরত্ব পাওয়া যাবে। কিন্তু কোন প্রান্ত দুটি? এটা কষ্ট করে খুঁজে বের করার থেকে আমরা প্রতিটি pair (মোট চার ভাবে) চেষ্টা করে তাদের মাঝে সবচেয়ে কমটা নিলেই পারি।

আরেকটি কেস হল যখন প্রদত্ত রেখা/রেখাংশ দুটি সমান্তরাল হয়। একই ভাবে আমরা দেখি দুজনের মাঝে কেউ রেখা আছে কিনা। থাকলে আমরা অপর রেখা/রেখাংশের যেকোনো প্রান্ত হতে অপর রেখার উপর লম্ব টানব। সেটাই আমাদের optimal দুরত্ব হবে। আর যদি দুটিই রেখাংশ হয়? তাহলে একটু ঝামেলার। প্রথমত আমরা প্রান্ত বিন্দুর বিভিন্ন ভাবে pair নিয়ে চেষ্টা করব। এরপর আমরা প্রতিটি প্রান্ত বিন্দু হতে অপর রেখাংশের উপর লম্ব টেনে দেখব যে লম্বের প্রান্ত বিন্দু অপর রেখাংশের উপর আছে কিনা। থাকলে সেই লম্ব দুরত্ব একটি candidate হবে। এরকম সকল candidate এর মাঝে থেকে আমাদের সচবেয়ে কম মান নিতে হবে।

আসলে আমাদের বাকি যেই কেস আছে- প্রদত্ত ইনপুট এর রেখা/রেখাংশ পরস্পরকে ছেদ করে না আবার parallel ও না- সেক্ষেত্রেও উপরের দ্বিতীয় কেস খাটবে। অর্থাৎ আমরা প্রান্ত বিন্দুদের pair নিয়ে তাদের দুরত্ব দেখব আর প্রতিটি প্রান্ত হতে ওপর রেখাংশে লম্ব টানব এবং দেখব সেই লম্বের পাদ বিন্দু ওপর রেখাংশ/রেখার উপর আছে কিনা।

এতো গেল case analysis. বাকি থাকল কোনো একটি বিন্দু হতে কোনো একটি রেখার উপর লম্ব টানা। কীভাবে করবা? খুব একটা কঠিন না যদি তুমি parametric equation বা vector ব্যবহার কর। মনে কর তুমি C হতে AB রেখার উপর লম্ব টানবা। মনে কর AB এর উপর যেকোনো একটি বিন্দু P. আমরা parametric form ব্যবহার করে P কে লিখতে পারি $A+(B-A)t$. এখন যদি CP আর AB লম্ব হয় তাহলে আমরা লিখতে পারি $CP.AB = 0$. এখান থেকে আমরা t এর মান সমাধান করতে পারি। একটা উদাহরন করে দেখান যাক। মনে কর আমরা $C(0,2)$ হতে AB এর উপর লম্ব টানতে চাই যেখানে $A(-5,0)$ আর $B(1,0)$. তাহলে আমরা AB এর উপর যেকোনো বিন্দু P এর parametric form লিখতে পারি $A+(B-A)t$ বা $(-5,0)+[(1,0)-(-5,0)]t$ বা $(-5,0)+(6,0)t$ বা $(1,0)t$ বা $(t,0)$. এখন যদি CP আর AB লম্ব হয় তাহলে আমরা লিখতে পারি $CP.AB = 0$ বা $[P-C].[B-A] = 0$ বা $[(t,0)-(0,2)].[(1,0)-(-5,0)] = 0$ বা $(t,-2).(6,0) = 0$ বা $6t = 0$ বা $t = 0$. তাহলে P এর মান পেতে আমরা P এর parametric form $(t,0)$ তে t এর মান বসাই। তাহলে P হবে $(0,0)$. এখানে অবশ্যই অনেক বেশি শূন্য নেওয়াতে আমাদের হিসাব নিকাশ অনেক সহজ হয়ে গিয়েছে কিন্তু আশা করি এতে উদাহরন বুঝতে সুবিধা হবে। তুমি এই হিসাব গুলি যেকোনো A, B, C এর জন্য করতে পার। তাহলে C হতে AB এর উপর লম্বের পাদ বিন্দু পেয়ে যাবে।

UVa 10979 How many triangles?

সমস্যা: সর্বোচ্চ দশটি সরল রেখাংশ থাকবে। বলতে হবে চিত্রে কয়টি ত্রিভুজ আছে। তিনটি বিন্দু সমরৈখিক হলে তারা ত্রিভুজ তৈরি করে না।

সমাধান: জ্যামিতির সমস্যা যে সবসময় জ্যামিতিক ফর্মুলা দিয়ে সমাধান করতে হবে তা না। মাঝে মাঝে গ্রাফ এর মত করেও সমাধান হতে পারে। আমরা প্রথমেই প্রতিটি রেখাংশকে অন্য সকল রেখাংশদের সাথে ছেদ করাব। এরকম করে সকল ছেদ বিন্দু আর রেখাংশদের প্রান্ত বিন্দু মিলে তৈরি হবে আমাদের vertex এর সেট। এরপর আমরা দেখব প্রতিটি রেখাংশের উপর আমাদের কোন কোন vertex আছে। তাদের মাঝে আমরা edge দেব। আসলে আমাদের ছেদ বিন্দু কিন্তু বেশি হবে না।

তাই আমরা তিনটি করে vertex নিয়ে চেক করে দেখতে পারি তারা সমরৈখিক কিনা। যদি না হয় তাহলে তাদের প্রতি দুটির মাঝে edge আছে কিনা। শুধু এখানে একটা জিনিস খেয়াল করবা যে একই বিন্দুকে তুমি যেন একাধিক বার vertex হিসাবে কল্পনা করে না বস। কারণ দেখা যাবে দুটি রেখার ছেদ বিন্দু আর অন্য দুটি রেখার ছেদ বিন্দু একই হতে পারে। সুতরাং তুমি যখন vertex এর সেট বের করেছ তখন সকল duplicate ধরে ফেলে দেবে।

UVa 10674 Tangents

সমস্যা: দুটি বৃত্ত দেওয়া আছে। তোমাকে এই দুটি বৃত্তের মাঝে যত গুলি স্পর্শক টানা সম্ভব টানতে হবে। প্রতিটি স্পর্শক এর জন্য বলতে হবে সেই স্পর্শক এর দৈর্ঘ্য কত আর সেই স্পর্শক বৃত্ত দুটিকে যেই দুটি বিন্দুতে স্পর্শ করে তাদের coordinate প্রিন্ট করতে হবে। যদি বৃত্ত দুটির মাঝে অসংখ্য স্পর্শক থাকে তাহলে -1 প্রিন্ট করতে হবে।

সমাধান: অনেক কেস চিন্তা করতে হবে। একে একে আসা যাক।

প্রথমে কখন অসংখ্য স্পর্শক থাকবে তা চিন্তা কর। যখন দুটি বৃত্ত একই তখন। সুতরাং প্রথমেই এই কেস কে বিদায় করে দিতে হবে।

এরপরের কেস, যখন একটি বৃত্ত সম্পূর্ণ ভাবে আরেকটি বৃত্তের ভেতরে থাকবে (এমনকি স্পর্শও করবে না) তখন তাদের মাঝে কোনো স্পর্শক থাকবে না।

এরকম আরেকটা কেস আছে। যখন বৃত্ত দুটি ভেতর থেকে স্পর্শ করবে তখন তাদের মাত্র একটি স্পর্শক থাকবে। স্পর্শক এর দৈর্ঘ্য হবে 0 আর সেই coordinate বের করাও খুব একটা কঠিন না। কারণ স্পর্শ বিন্দু বৃত্তের কেন্দ্রদ্বয়ের সংযোগ রেখার উপরে থাকে।

এছাড়া বাদ বাকি সব কেসে উপরে নিচে দুটি স্পর্শক থাকবেই। কীভাবে এই স্পর্শকের সব তথ্য বের করা যায় তা পরে দেখব। আপাতত বাদ বাকি কেস গুলি বের করে ফেলি।

এর পরের কেসটি হবে যখন বৃত্ত দুটি বাইরে থেকে স্পর্শ করে আছে। তখন তাদের আরও একটি স্পর্শক আছে। আর এর তথ্য বের করা খুব একটা কঠিন না।

আর একটি মাত্র কেস, আর তাহল একটি বৃত্ত আরেকটি বৃত্তের সম্পূর্ণ বাইরে। সেক্ষেত্রে তাদের মাঝে আরও দুইটি স্পর্শক থাকবে যেটা দেখতে কিছুটা X এর মত মনে করতে পার।

এখন স্পর্শকগুলি বের করা যাক। আমি শুধু উপরের আর নিচের স্পর্শকের তথ্য বের করব। X এর মত স্পর্শকের তথ্য আশা করি তোমরা একই ভাবে নিজেরা বের করতে পারবে।

মনে কর আমাদের বৃত্ত দুটির কেন্দ্র হল O_1, O_2, তাদের ব্যাসার্ধ r_1, r_2. আমরা আমাদের সুবিধার জন্য মনে করব $r_1 \leq r_2$. আরও মনে কর উপরের যে স্পর্শক তা এই বৃত্ত দুটিকে P_1, P_2 তে ছেদ করে। আমাদের কাজ P_1P_2 এই দৈর্ঘ্য বের করা আর P_1, P_2 এই বিন্দু দুটির coordinate বের করা। এই কাজ করতে পারলে তুমি নিচের স্পর্শকের জন্যও একই কাজ করতে পারবে। এখন O_1 থেকে O_2P_2 এর উপর লম্ব টান, মনে কর এদের ছেদ বিন্দু M. এখন O_1O_2M এই ত্রিভুজের দিকে তাকাও। এটি একটি সমকোণী ত্রিভুজ, যার দুটি বাহু তুমি জান, O_1O_2 এই দুরত্ব তুমি জান কারণ তুমি এই দুটি বিন্দুর coordinate জান। আবার তুমি $O_2M = r_2 - r_1$ জান। সুতরাং তুমি খুব সহজেই $\angle O_1O_2M$ কোণের মান বের করতে পারবা। একটু খেয়াল রেখ যেন তোমার ফর্মুলা $O_2M = 0$ হলেও যেন কাজ করে। তুমি যদি $\angle O_1O_2M = \angle O_1O_2P_2$ এর মান জান তাহলে $\angle O_2O_1P_1$ এর মানও জান কারণ O_1P_1 আর O_2P_2 সমান্তরাল। এর মানে P_1 বিন্দুটি O_1 সাপেক্ষে কত angle তৈরি করে আছে তা তুমি জান, আর সেই তথ্য থেকে তুমি $(r\cos\theta + x_{center}, r\sin\theta + y_{center})$ ফর্মুলা(polar coordinate এর ফর্মুলা) ব্যবহার করে P_1 এর coordinate খুব সহজেই বের করে ফেলতে পারবে। একই ভাবে P_2 ও বের করতে পারবে। আর দুই প্রান্তের coordinate বের করতে পারলে তো তাদের দুরত্বও বের করতে পারবা।

Timus 1697 Sniper Shot

সমস্যা: একটি 3d coordinate সিস্টেমে একজন sniper বসে আছে। Sniper এর পজিশন coordinate দেওয়া আছে, ধরা যাক S. একটা লোক A বিন্দু হতে B বিন্দুতে যাবে। A এবং B বিন্দুর coordinate দেওয়া আছে এবং এই দুটি বিন্দুই XY প্লেন এর উপর অবস্থিত। এছাড়াও এখানে n টি বিল্ডিং ($n \leq 100$) আছে। প্রতিটি বিল্ডিং আয়তাকার ঘনক (rectangular parallelepiped). এদের নিচের তল XY প্লেনের সাথে আছে। ইনপুট হিসাবে XY প্লেনে তার বিপরীত দুই প্রান্ত এবং এই বিল্ডিং এর উচ্চতা দেওয়া আছে। Sniper এর গুলির বেগ infinity. সুতরাং সে লোককে দেখতে পাওয়া মাত্রই গুলি করতে পারবে। লোকটি A হতে B তে যায়। বলতে হবে কোন বিন্দুতে সবার আগে তাকে মারা যাবে। খেয়াল রাখতে হবে তোমার গুলি যেন কোনো বিল্ডীং এর ভেতর দিয়ে না যায়, চাইলে স্পর্শ করতে পারবে। আরও বলা আছে AB রেখাংশটা কোনো বিল্ডীং এর ভেতর দিয়ে যায় না এবং sniper এর পজিশনও কোনো বিল্ডিং এর ভেতর হবে না।

সমাধান: বেশ কঠিন সমস্যা। এধরনের সমস্যা সহজে কেউ কোড করতে চাইবে না। বিশেষ করে কন্টেস্ট সময়ে। কারণ এসব সমস্যায় দেখা যায় অনেক ধরনের কেস হয় এবং কোডও অনেক বড় হয়। সুতরাং দক্ষ না হলে এধরনের সমস্যায় AC পাওয়া প্রায় অসম্ভব। সমাধানের মূল আইডিয়া কিন্তু কঠিন না। কিন্তু খেয়াল করলে দেখবা এই সমাধানে অনেক গুলি component আছে, প্রতিটি component নিজেরা জটিল হওয়ায় দেখা যায়, বহু ধরনের corner case আমরা overlook করছি। এরপর আবার দেখা যাবে precision error নিয়েও মাথা ঘামাতে হচ্ছে। এজন্যই এই সমস্যাকে বেশ কঠিন বলেছি। তবে একটা জিনিস ভাল যে, এধরনের সমস্যার ক্ষেত্রে আমাদের time complexity নিয়ে খুব একটা ভাবতে হয় না।

প্রথমেই দেখ যে S কি AB রেখাংশে আছে কিনা। থাকলে A তেই মারা সম্ভব। কারণ আমাদের বলা আছে যে AB রেখার উপর কোনো বিল্ডিং থাকবে না। আরেকটা কেস হতে পারে S বিন্দু AB রেখার উপরে (কিন্তু AB এর উপর না)। সেক্ষেত্রে দেখতে হবে যে S আর A বা B এর মাঝে কি কোনো বিল্ডিং আছে কিনা।

তা না হলে আমরা SAB দিয়ে একটি plane কল্পনা করব। আমরা যেদিকে শুট করব তা কিন্তু এই প্লেনে আছে। সমস্যা হল আমাদেরকে বের করতে হবে কোন কোন দিকে শুট করলে AB লাইনকে টাচ করার আগেই কোনো একটি বিল্ডিংকে হিট করবে। সেজন্য আমরা প্রতিটি বিল্ডিং এর জন্য বের করব সেই বিল্ডিং আর SAB প্লেনের ছেদ অংশ। যেহেতু এই ছেদ অংশ AB এর উপরে থাকবে না সেহেতু হয় তা SAB প্লেনে AB এর যেদিকে S আছে সেদিকে আছে নাহয় বিপরীত দিকে আছে। আবার যেদিকে S আছে সেদিকে থাকলেও হয়তো দেখা যাবে S এর পেছন দিকে আছে। আমরা এইসকল কেস হ্যান্ডেল করে বের করব প্রতিটি বিল্ডিং এর জন্য AB রেখার কোন কোন অংশ S দেখতে পায় না। এর পর এই দেখতে পায়না অংশগুলি আর AB নিয়ে একটু হিসাব করলেই আমরা প্রথম কোন বিন্দু দেখতে পায়, বা আদউ কোনো বিন্দু দেখতে পায় না তা বের করা যাবে।

এখন চল একেকটি অংশ কীভাবে সমাধান করা যায় তা দেখা যাক।

প্রথম অংশ- বিল্ডিং এর সাথে SAB এর ছেদ অংশ কীভাবে বের করবা। যেহেতু প্রতিটি বিল্ডিং convex সেহেতু যেকোনো একটি প্লেনের সাথে তার ছেদ অংশ একটি convex polygon হবে। যদি বিল্ডিং concave হত তাহলে কিন্তু এই কথা সত্য হত না। সুতরাং এই আইডিয়া কাজে লাগায়ে আমরা আমাদের সমাধানকে একটু সহজ করতে পারি। আমরা প্রতিটি বিল্ডিংকে এখন থেকে 12 টি 3d লাইন সেগমেন্ট হিসাবে কল্পনা করব (অর্থাৎ ঘনকের 12টি বাহু)। আমরা এই প্রতিটি সেগমেন্টকে SAB প্লেনের সাথে intersect করাব। এরপর যেই intersection বিন্দুগুলি পেলাম তাদের নিয়ে একটি convex polygon বানাব। তাহলেই আমরা আমাদের 3d প্রব্লেমকে 2d তে কনভার্ট করতে পারব।

এই প্রথম অংশের আবার দুটি অংশ। প্রথম অংশ- একটি 3d লাইনের সাথে প্লেনের intersection করা। এখানে নানা কেস থাকতে পারে, যেমন হয়তো 3d লাইন প্লেনের parallel, সুতরাং তাদের কোনো intersection নেই। বা দেখা যাবে, পুরো লাইন সেগমেন্টই এই 3d প্লেনের উপর আছে। এই কেস দুটি তোমাকে খেয়াল রাখতে হবে। আর যদি এই দুটি কেসের কোনোটিই নাহয় এর মানে তুমি parametric equation ব্যবহার করে বেশ সহজেই লাইনের সাথে প্লেনের intersection বের করতে পারবে। এরপর তোমাকে দেখতে হবে এই intersection বিন্দু কি আদৌ লাইন সেগমেন্টের উপর অবস্থিত কিনা। দ্বিতীয় অংশ হল- যখন তুমি সব intersection বিন্দু পেয়ে যাবে তখন তাদের

convex hull বানাতে হবে।

এবার দ্বিতীয় অংশ- প্রতিটি বিল্ডিং এর জন্য আমরা কীভাবে বের করব AB সেগমেন্টের কোন কোন অংশ সেই বিল্ডিং ঢেকে রেখেছে। খেয়াল কর, এখানে বিল্ডিং মানে 3d না, আমরা এর আগের অংশে একে 2d বানিয়ে ফেলেছি। এখন তুমি S এর সাথে বিল্ডিং এর প্রতিটি শীর্ষবিন্দুর সংযোগকারী লাইন টান। ধরা যাক আমরা এখন বিল্ডিং এর T বিন্দু নিয়ে কাজ করছি। তাহলে তুমি দেখ ST এর বর্ধিত অংশ (T এর দিকের) AB সেগমেন্টকে ছেদ করে কিনা। যদি কোনো শীর্ষের জন্যই ছেদ না করে তার মানে আমাদের এই বিল্ডিং নিয়ে কোনো কাজ নেই। এই যে ST এর বর্ধিত অংশ, AB সেগমেন্ট এসব যে special condition, তুমি যদি parametric ফর্মে বা vector ফর্মে এই লাইন লাইন intersection বা line plane intersection এসব সমাধান কর তাহলে খুব সহজেই তুমি parameter এর মান দেখে বুঝতে পারবে যে এটা কি লাইন সেগমেন্ট এর উপর আছে কিনা, বা T এর দিকের বর্ধিত অংশে আছে কিনা ইত্যাদি। এখানেও একই রকম কিছু সমস্যা আছে, যেমন হয়তো ST আর AB সমান্তরাল ইত্যাদি। কিন্তু এসব ঝামেলা শেষে আমরা পাব যে ST এর বর্ধিত অংশ এর সাথে AB এর ছেদ বিন্দু। এখানে একটা জিনিস, তাহল ST আর AB যদি সমান্তরাল হয় তাহলে আসলে সেটা কোন দিকের infinity তা বের করতে হবে। A এর দিকের না B এর দিকের। এর জন্য যা করবা তাহল ST এর সাপেক্ষে SA এর কোন আর SB এর কোণ দেখবা, যার জন্য কোণের মান কম তার দিকের infinity কে ছেদ বিন্দু হিসাবে নিবা। এবার বিল্ডিং এর সকল শীর্ষের জন্য ছেদ বিন্দু পেলে তুমি একদম বামের ছেদ বিন্দু আর একদম ডানের ছেদ বিন্দু নিয়ে একটি সেগমেন্ট গঠন করবে যা AB লাইন এর উপর অবস্থিত। বা অন্যভাবে বলা যায়- এই লাইন সেগমেন্ট এর অংশটুকু এই বিল্ডিং দিয়ে ঢেকে থাকবে।

এবার তৃতীয় অংশ- দ্বিতীয় অংশ শেষে আমাদের প্রব্লেম এখন 1d তে চলে এসেছে। প্রতিটি বিল্ডিং এর জন্য আমরা এখন AB লাইনের উপর একটি করে সেগমেন্ট পেয়েছি, যা S থেকে দেখা যায় না। প্রশ্ন হল AB এর কোন বিন্দু প্রথম S থেকে দেখা যায়। খেয়াল কর, এমন হতে পারে যে, একটি বিন্দু $[A, C]$ অংশ ঢেকে রেখেছে আরেকটি বিল্ডিং $[C, B]$ অংশ ঢেকে রেখেছে ($A < C < B$). সেক্ষেত্রে কিন্তু তুমি আসলে C বিন্দু দেখতে পাও। কারণ বলেছে যে তোমার গুলি বিল্ডিংকে স্পর্শ করতে পারে। কিন্তু যদি এখানে আরেকটা বিল্ডিং থাকত যা $[A, B]$ পুরটা ঢেকে রাখত, তাহলে কিন্তু C দেখতে পারবে না। কিন্তু এই দুটি ক্ষেত্রেই তুমি কিন্তু A আর B দুটি স্থানই দেখতে পাবে। সুতরাং এইসকল কেস মনে রেখে আমাদের সমাধান করতে হবে। আমরা এই সেগমেন্টগুলির প্রান্তগুলিকে event হিসাবে কল্পনা করি। A এর দিকের event হল starting আর B এর দিকের event হবে ending. A এর দিকের event গুলি আগে থাকবে এইভাবে আমরা সকল event সর্ট করব। যদি একই বিন্দুতে একাধিক event থাকে তাহলে আমরা ending কে আগে রাখব আর starting কে পরে। এভাবে সকল event কে সর্ট করি। এরপর এই সর্ট করা event গুলিকে একে একে প্রসেস করি। আমরা একটা counter রাখব যার মান শুরুতে শূন্য। যখন আমরা starting event পাব তখন তাকে এক বাড়াবো, আর যখন ending event পাব তখন এক কমাব। এই এক বাড়ানোর আগে বা এক কমানর পরে আমরা দেখব যে counter এর মান শূন্য কিনা। যদি শূন্য হয় এর মানে এটি একটি candidate বিন্দু। আমরা দেখব এই বিন্দু AB সেগমেন্ট এর উপরে আছে কিনা। থাকলে এই বিন্দুই সমাধান। কিন্তু এরকম যদি কোনো বিন্দু না পাও তার মানে কোনো সমাধান নেই। এখানে একটাই জিনিস খেয়াল রাখতে হবে, আর তাহল আমাদের এর আগের অংশের infinity যেন এখানে ঠিক মত হ্যান্ডল হয়।

UVa 10283 The Kissing Circles

সমস্যা: একটি বড় বৃত্ত আছে R ব্যাসার্ধের। তাতে n টি ($n \leq 100$) ছোট ছোট বৃত্ত আছে যা বড় বৃত্তকে স্পর্শ করে আছে এবং ছোট বৃত্ত গুলও পাশাপাশি স্পর্শ করে আছে, ঠিক পাশাপাশি গোল করে দাঁড়ানর মত। বলতে হবে এই ছোট বৃত্তের ব্যাসার্ধ কত। ভাল হয় যদি তোমরা মূল statement এর ছবি দেখ।

সমাধান: মুলসমস্যায় আরও দুটি আউটপুট চেয়েছে কিন্তু তুমি যদি ছোট বৃত্তের ব্যাসার্ধ বের করতে

পার তাহলে বাকিটুকু ডাল ভাত।

এই ধরনের সমস্যাকে বলা হয় packing সমস্যা। এখানে তোমাকে ছোট বৃত্তের ব্যাসার্ধ বের করতে হবে যেন সব বৃত্ত একত্রে pack হয়ে থাকে। এধরনের সমস্যার ক্ষেত্রে স্পর্শ বিন্দু আর symmetry কে খুব ভাল করে খেয়াল করতে হয়। আর এধরনের সমস্যায় প্রায়ই দেখা যায় binary search ব্যবহার করলে সমস্যা অনেক সহজ হয়ে যায়।

যেমন এই সমস্যার ক্ষেত্রে, আমরা ছোট ছোট বৃত্তের কেন্দ্রগুলিকে যোগ করে একটি সম n ভুজ (n sided regular polygon) বানাতে পারি যার কেন্দ্র হবে বৃত্তের কেন্দ্র। যদি $n < 3$ হয় তাহলে কিন্তু ঠিক মত সমভুজ তৈরি হবে না। সুতরাং তুমি ওদেরকে আলাদা করে হ্যান্ডল করে ফেল। তাহলে আমরা এখন মনে করতে পারি $n \geq 3$. এখন আমাদের যা করতে হবে তা হল এই সমভুজের সাথে বড় বৃত্তের সম্পর্ক বের করতে হবে। একটা খুব সহজ সম্পর্ক হল এদের কেন্দ্র দুটি একই। এখান থেকে আমরা বলতে পারি, কেন্দ্র থেকে সমভুজের কোনো বিন্দুর দুরত্বের সাথে ছোট বৃত্তের ব্যাসার্ধ যোগ করলে আমরা বড় বৃত্তের ব্যাসার্ধ পাব। এই সমীকরণ সমাধান করলেই আমরা ছোট বৃত্তের ব্যাসার্ধ পেয়ে যাব।

মনে কর আমাদের ছোট বৃত্তের ব্যাসার্ধ r. তাহলে সমভুজের এক বাহুর দৈর্ঘ্য হবে $2r$. এখন এই বাহুর দৈর্ঘ্যের সাথে কেন্দ্র হতে সমভুজের শীর্ষের দুরত্বের সম্পর্ক কি? আমরা সমভুজের একটি বাহুর দুটি প্রান্তকে কেন্দ্রের সাথে যোগ করি। যেহেতু সমভুজে n টি বাহু মিলে কেন্দ্রে 2π কোণ তৈরি করে সেহেতু একটি বাহু $\frac{2\pi}{n}$ কোণ তৈরি করবে। আর এই যে বাহুটি কেন্দ্রের সাথে যে ত্রিভুজ তৈরি করে তা একটি সমদ্বিবাহু ত্রিভুজ। আমাদের লক্ষ্য এই সমদ্বিবাহু ত্রিভুজের অন্য দুটি বাহুর দৈর্ঘ্য বের করা যেখানে সমভুজের বাহুর দৈর্ঘ্য $2r$. কীভাবে বের করবে? আমরা কেন্দ্র হতে ঐ বাহুর উপর একটি লম্ব টানি। তাহলে এই ছোট ত্রিভুজ একটি সমকোণী ত্রিভুজ হবে যার অতিভুজ আমাদের বের করতে হবে। আমাদের ভুমির দৈর্ঘ্য r, কোণ $\frac{\pi}{n}$. সুতরাং আমাদের অতিভুজ হবে $r/\sin\frac{\pi}{n}$. এটিই হল কেন্দ্র হতে সমভুজের শীর্ষ বিন্দুর দুরত্ব। এর সাথে r যোগ করলে এটি হবে বড় বৃত্তের ব্যাসার্ধ R. অর্থাৎ $r + r/\sin\frac{\pi}{n} = R$ যেখানে r unknown. আমরা এই সমীকরণকে খুব সহজেই সমাধান করতে পারি।

UVa 10286 Trouble with a Pentagon

সমস্যা: একটি সম পঞ্চভুজ দেওয়া আছে। তোমাকে এর ভেতরে সবচেয়ে বড় বর্গ প্যাক করতে হবে যেন বর্গের একটি শীর্ষ পঞ্চভুজের একটি শীর্ষে থাকে।

সমাধান: সমস্যাতে দেওয়া চিত্রটি ভাল করে খেয়াল কর। চিত্রের বর্গটি বাম ডান ব্যালান্স করে আছে। এছাড়াও খেয়াল কর বর্গের ডান আর বামের শীর্ষ দুটি পঞ্চভুজের বাহুর উপর আছে। এই টুকু তথ্য ব্যবহার করে আমরা বর্গের সাইজ বের করে ফেলতে পারি।

বর্গ আর পঞ্চভুজের যেই শীর্ষ একই বিন্দুতে তা থেকে বিপরীত বাহুর উপর লম্ব টান। তোমার বর্গের কেন্দ্র অবশ্যই এই লম্বের উপর থাকবে। তুমি এই কেন্দ্র এর উপর binary search করতে পার। কেন্দ্র বেশি নিচে ধরলে বর্গ অনেক বড় হবে ফলে বর্গের শীর্ষ পঞ্চভুজের বাইরে চলে যাবে। আবার যদি কেন্দ্র বেশি উপরে হয়ে যায় তাহলে বর্গ বেশি ছোট হবে। এবং এর ফলে বর্গের শীর্ষ পঞ্চভুজের ভেতরে থাকবে। এরকম করে binary search করলে দেখবে একসময় বর্গের শীর্ষ পঞ্চভুজের উপরে থাকবে। এই বর্গের সাইজই হবে আমাদের উত্তর। আসলে বর্গের শীর্ষ পঞ্চভুজের ভেতরে না বাইরে তা চেক করা ওত কঠিন না। তুমি পঞ্চভুজের বাহু সাপেক্ষে turn দেখবে তাহলেই হবে (point in a polygon technique). আরও একটি জিনিস খেয়াল কর, তোমাকে এই binary search কিন্তু প্রতি ইনপুটে করতে হবে না। তুমি যদি 1 দৈর্ঘ্যের পঞ্চভুজের জন্য উত্তর বের কর a, তাহলে f দৈর্ঘ্যের পঞ্চভুজের জন্য উত্তর হবে fa.

এছাড়াও তুমি চাইলে বর্গের বাহুর দৈর্ঘ্যের জন্য ফর্মুলা বের করতে পার। বাইনারি সার্চ না করেও এই সমস্যা সমাধান করা সম্ভব।

UVa 10695 Find the Point

সমস্যা: একটি ত্রিভুজ ABC এর তিনটি শীর্ষ বিন্দুর coordinate দেওয়া আছে। সেই সাথে এর ভেতরে একটি O এর জন্য $\angle AOB, \angle BOC, \angle AOC$ দেওয়া আছে। তোমাকে O এর coordinate প্রিন্ট করতে হবে অথবা বলতে হবে এমন কোনো O নেই।

সমাধান: মনে কর একটি বৃত্ত আছে যার একটি জ্যা হল AB. এখন তুমি যদি AB এর যেকোনো এক দিকের বৃত্তচাপ নাও তাহলে সেই চাপের যেকোনো বিন্দুতে AB একই পরিমান কোণ তৈরি করবে। উলটা ভাবে বললে বলা যায়- মনে কর AB দেওয়া আছে আর বলা আছে O বিন্দুতে AB $\angle AOB$ তৈরি করে। তাহলে আমরা এমন একটি বৃত্ত আঁকতে পারি যার একটি জ্যা AB এবং সেই বৃত্ত চাপের উপরে প্রত্যেক বিন্দুতে AB জ্যা $\angle AOB$ তৈরি করবে।

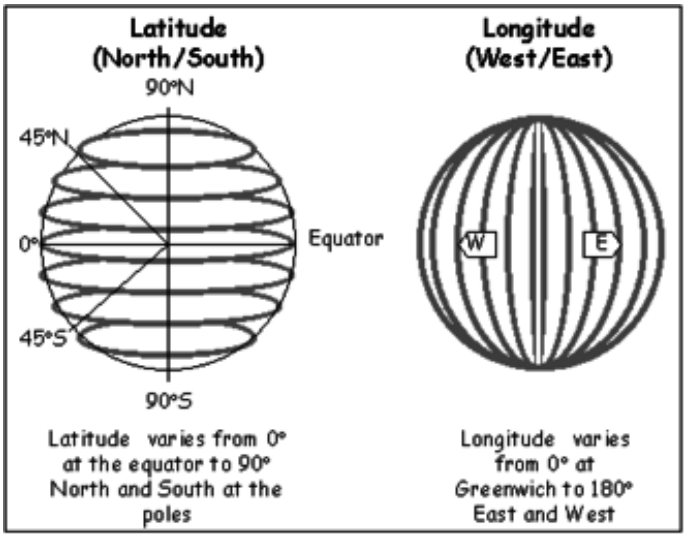
অর্থাৎ মনে কর A আর B fixed আর $\angle AOB$ ও fixed. তাহলে O বিন্দুর লোকাস (locas) হবে একটি বৃত্ত চাপ। আসলে একটি বৃত্ত চাপ বলা ঠিক হচ্ছে না। AB এর এক পাশে O এর লোকাস একটি বৃত্তের চাপ আর আরেক পাশে আরেক বৃত্তের চাপ। আমাদের এই সমস্যায় আমরা এমন একটি বৃত্ত চাপ নেব যেন $\angle AOB$ এর মান প্রদত্ত মানের সমান হয় আর এই চাপটি AB এর যেদিকে C আছে সেদিকে থাকে। এই কাজ করা কিন্তু কঠিন না। আমরা কিন্তু চাইলে মনে করতে পারি $OA = OB$. এটা মনে করলে OAB হবে সমদ্বিবাহু ত্রিভুজ। সুতরাং O এর coordinate বের করা বেশ সহজ হবে। আসলে এরকম দুটি O এর coordinate পাওয়া যাবে। যেটি AB এর যে পাশে C আছে তাকে নেব। এরপর A, O আর B দিয়ে যেই বৃত্ত যায় তার সমীকরণ বের করব। একই ভাবে আমরা A, O আর C দিয়ে যাওয়া বৃত্তের সমীকরণ বের করব। এরপর এদেরকে intersect করলেই আমরা O পেয়ে যাব। যদি এই O ত্রিভুজের ভেতরে থাকে তার মানে আমাদের সমাধান আছে, আর না হলে সমাধান নেই।

UVa 10809 Great Circle

সমস্যা: একটি গোলাকার গ্রহে দুটি বিন্দুর longitude আর latitude দেওয়া আছে। তোমাকে এদের এক বিন্দু হতে আরেক বিন্দুতে গোলকের পৃষ্ঠ দিয়ে shortest distance এ যেতে হবে। তোমাকে shortest path এ থাকা সবচেয়ে উত্তরের পয়েন্ট এর latitude ডিগ্রী এবং মিনিটে প্রিন্ট করতে হবে। যদি একাধিক উত্তর থাকে তাহলে undefined প্রিন্ট করতে হবে।

সমাধান: এই সমস্যা সমাধানের আগে দেখে নেওয়া যাক latitude আর longitude আসলে কি। ১১.১ দেখ। মনে কর গোলকের উপর কোনো একটি বিন্দুর সাথে তুমি গোলকের কেন্দ্র যোগ করলে। এই রেখা XY প্লেনের সাথে যেই কোণ তৈরি করে সেটাই latitude. যদি বিন্দুটি XY প্লেনের উপরে থাকে অর্থাৎ $z > 0$ হয় তাহলে আমরা কোণের শেষে N লাগাই বা একে north latitude বলা হয় আর নিচে হলে অর্থাৎ $z < 0$ হলে আমরা কোণের সাথে S লাগাই বা একে south latitude বলে। এই সমস্যাতে বিষুব রেখা অর্থাৎ $z = 0$ হলে 0N লিখা হবে। আমরা latitude কে θ দিয়ে প্রকাশ করব। আর আমরা আমাদের সমস্যায় N হলে কোণকে ধনাত্মক আর S হলে ঋণাত্মক করে নেব। যদি গোলকের ব্যাসার্ধ 1 হয় তাহলে আমাদের বিন্দুটির z coordinate হবে $sin\theta$.

এবার longitude দেখা যাক। আমরা north pole থেকে south pole একটি বৃত্তাকার লাইন টানি, যে বৃত্তের কেন্দ্র হবে গোলকের কেন্দ্র। এরকম তো আসলে অনেক বৃত্ত পাওয়া সম্ভব। আমরা যেকোনো একটি আঁকবো আর তাকে বলব 0 longitude. আমাদের যেই বিন্দুর longitude জানা দরকার সেই বিন্দু, north pole, south pole দিয়ে বৃত্ত চাপ আঁকি। যদি এই বৃত্ত চাপ আমাদের 0 এর বামে হয় তাহলে এটি হবে west আর ডানে হলে east. এখন এই বৃত্ত চাপ তো বিষুব রেখাকে একটি বিন্দুতে ছেদ করেছে। বৃত্তচাপটি বিষুব রেখায় যেখানে ছেদ করেছে সেই বিন্দু, 0 এর বৃত্তচাপ বিষুব রেখাকে যেই বিন্দুতে ছেদ করেছে সেই বিন্দু- এই দুটি বিন্দু গোলকের কেন্দ্রে যেই কোণ তৈরি করে তাই longitude. আমরা আগের মত west কে ধরব ঋণাত্মক আর east কে ধনাত্মক, কারণ



নকশা ১১.১: Latitude এবং Longitude, wikipedia হতে

এতে করে আমরা longitude কে খুব সহজেই x, y এ পরিনত করতে পারব। মনে কর ϕ হল longitude. এখন আমাদের বিন্দুথেকে সরাসরি নিচে XY প্লেনের উপর লম্ব টান। আমরা এই বিন্দুর x, y নিয়ে আগ্রহি, কারণ এর x, y ই হল আমাদের বিন্দুর x, y. গোলকের কেন্দ্র থেকে এর দুরত্ব $\cos\theta$. এবার $x = \cos\theta \sin\phi$ আর $y = \cos\theta \cos\phi$.

সুতরাং এতো সাধনার পর আমরা latitude longitude হতে আমাদের প্রদত্ত বিন্দু দুটির x, y, z বের করে ফেলব।

এখন আসো এর যত special case আছে সব গুলোকে আমরা হ্যান্ডল করব। আমাদেরকে বলেছে যে এক বিন্দু হতে অপর বিন্দুতে shortest path এ যেতে হবে এবং এই shortest path এর উপরের সবচেয়ে north most বিন্দু প্রিন্ট করতে হবে। যদি এই উত্তর unique নাহয় তাহলে undefined বলতে হবে।

যদি প্রদত্ত দুটি বিন্দু একই হয় এর মানে আমাদের shortest path এর দৈর্ঘ্য 0. সুতরাং আমাদের উত্তর হবে আমাদের ইনপুট এর বিন্দুটিই।

যদি প্রদত্ত বিন্দুদুটির একটি north pole এ থাকে তার মানে তাদের মাঝের যত shortest path সব তো north pole দিয়ে যাবে। সুতরাং north pole হবে উত্তর। এছাড়াও যদি ইনপুটের কোনো বিন্দু south pole এ হয় তাহলেও ঐ একই কথা, আমাদের যেকোনো shortest path ই north pole দিয়ে যাবে।

এসব বাদে যদি আমাদের ইনপুটের বিন্দু গোলকের কেন্দ্রের সাপেক্ষে opposite এ থাকে (অর্থাৎ তারা একটি ব্যাস গঠন করে) তাহলে তাদের দিয়ে অসংখ্য বৃত্ত তৈরি হয় যার কেন্দ্র গোলকের কেন্দ্র। সুতরাং এক্ষেত্রে unique কোনো উত্তর নেই।

এবার হল আমাদের মূল সমস্যা। আমাদেরকে গোলকের কেন্দ্রকে কেন্দ্র করে ইনপুটের দুটি বিন্দু দিয়ে যায় এরকম একটি বৃত্ত আঁকতে হবে। সেটি গোলকের সবচেয়ে উপরের যে বিন্দু দিয়ে যায় তাকে প্রিন্ট করতে হবে। বা একে অন্য ভাবে দেখতে পার। মনে কর প্রদত্ত দুটি বিন্দু আর গোলকের কেন্দ্র দিয়ে একটি প্লেন আঁকি। এটি গোলকের সবচেয়ে উপরের যে বিন্দু দিয়ে যাবে সেটিই উত্তর, অবশ্য চেক করতে হবে যে সেই বিন্দুটি আদৌ আমাদের ইনপুটের দুটি বিন্দুর মাঝের shortest path এর উপর আছে কিনা।

মনে কর আমরা জানি আমাদের উত্তরের বিন্দু X, কেন্দ্র O আর ইনপুটের দুটি বিন্দু A আর B. তাহলে $\angle AOB$ বের করব। সেই সাথে দেখব $\angle AOX + \angle BOX$ এর মান $\angle AOB$ এর থেকে বড় না ছোট না সমান। ছোট তো কখনই হবে না। যদি বড় হয় এর মানে X বিন্দু A আর B এর মাঝের

shortest path এ থাকবে না। সেক্ষেত্রে উত্তর হবে A আর B এর মাঝে যে বিন্দুটি সবচেয়ে উত্তরে থাকে। অন্যথা X বিন্দুটিই আমাদের উত্তর।

সুতরাং প্রশ্ন হল X কীভাবে বের করব। আমরা এখানে একটু vector আর calculus ব্যবহার করব। আমরা OA আর OB vector জানি, কারণ A আর B এর coordinate জানি। এই ভেক্টর দুটি হল (x_A, y_A, z_A) আর (x_B, y_B, z_B). এদেরকে যদি আমরা cross product করি তাহলে আমি যেই plane এর কথা বলছিলাম তার normal vector পাব। আর normal vector জানা মানেই ঐ প্লেন জানা। কারণ কোনো একটি নরমাল ভেক্টর তো একটি মাত্র প্লেনেরই normal হতে পারে তাই না? ধরা যাক এই নরমাল ভেক্টর হল $N = (x_N, y_N, z_N)$ যার সব গুলি component এর মান আমাদের জানা। এখন মনে কর আমাদের optimal বিন্দু হল (x, y, z). এই বিন্দুটি যেমন গোলকের উপর আছে ঠিক তেমনি প্লেনের উপরেও আছে। এখন একটু চিন্তা করে দেখ তো এই বিন্দুটি আমাদের নরমাল ভেক্টর এর সাথে কি সম্পর্ক তৈরি করবে? সহজ, এদের মাঝের কোণ সমকোণ বা 90 ডিগ্রী। দুটি ভেক্টর 90 ডিগ্রী এর equation কীভাবে লিখবা? এটাও সহজ, এদের মধ্যকার dot product শূন্য হবে। সুতরাং আমরা লিখতে পারি $xx_N + yy_N + zz_N = 0$. আবার যেহেতু (x, y, z) এই বিন্দুটি গোলকের উপরে আছে। সুতরাং আমরা লিখতে পারি $x^2 + y^2 + z^2 = 1$. আমাদের কাজ হল এই দুটি equation কে সিদ্ধ করে এরকম সবচেয়ে বড় z এর মান বের করা। আমরা প্রথম equation থেকে x এর মান দ্বিতীয় সমীকরণে বসিয়ে z এর একটি equation দাঁড় করাতে পারি এবং এরপর চেষ্টা করতে পারি বের করতে যে z এর সর্বোচ্চ মান কোথায় হবে। অথবা চাইলে lagrange multiplier মেথড খাটাতে পার। এই দুটি ক্ষেত্রেই বেশ বড় সর ক্যাল্কুলেশন করতে হবে। একটু ধৈর্য ধরতে করতে হবে আর কি।

UVa 10089 Repackaging

সমস্যা: তোমার একটি ফ্যাক্টরিতে তিন ধরনের গ্লাস উতপন্ন হয়। তিন ধরনের গ্লাসের সাইজ কিন্তু আলাদা আলাদা। এখন এদেরকে বিভিন্ন ভাবে প্যাক করা হয়। প্রতি প্যাকে এই তিন ধরনের গ্লাস বিভিন্ন পরিমানে আছে। যেমন একটি প্যাক হতে পারে $(1, 11, 5)$ এর মানে এখানে প্রথম ধরনের গ্লাস আছে একটি, দ্বিতীয় ধরনের গ্লাস আছে এগারোটি আর তৃতীয় ধরনের গ্লাস আছে পাঁচটি। প্রতি ধরনের প্যাক অসংখ্য পরিমানে আছে। তোমার কাজ হল এসকল প্যাক থেকে যেকোনো সংখ্যক প্যাক নেবে, তুমি চাইলে কোনো এক ধরনের প্যাক একাধিক নিতে পার আবার সেই প্যাক নাও নিতে পার। এরপর এসকল প্যাককে তুমি খুলবে এবং পুনরায় প্যাক করবে। শর্ত হল এই নতুন প্যাক গুলিতে তিন ধরনের গ্লাস একই সংখ্যায় থাকবে আর তুমি তোমার নির্বাচিত প্যাকগুলি থেকে যত গুলি গ্লাস পেয়েছিলে সব কটিকেই ব্যবহার করতে হবে। বলতে হবে তুমি এভাবে গ্লাসগুলিকে প্যাক করতে পারবে কিনা। তোমার ফ্যাক্টরিতে সর্বোচ্চ 1000 ধরনের প্যাক থাকতে পারে।

সমাধান: মনে কর প্যাকগুলি হল একেকটি 3d coordinate বিন্দু। এদেরকে আমরা p_i দ্বারা নির্দেশ করব। এখন মনে কর আমরা i তম প্যাক নেব a_i টি। শর্ত হল $\sum a_i p_i$ এর coordinate এর x, y এবং z তিনটিই একই হবে, $a_i \geq 0$ এবং a_i গুলি পূর্ণ সংখ্যা হবে। $\sum a_i p_i$ এর coordinate এর x, y এবং z তিনটিই একই হবে এই শর্তকে আমরা লিখতে পারি $\sum a_i p_{xi} = \sum a_i p_{yi} = \sum a_i p_{zi}$ বা $\sum a_i (p_{xi} - p_{zi}) = \sum a_i (p_{yi} - p_{zi}) = 0$. অর্থাৎ আমরা আমাদের 3d প্রব্লেমকে 2d তে কনভার্ট করে ফেললাম। আমরা এখন আর $p(x, y, z)$ নিয়ে চিন্তা করবনা বরং $q(x, y)$ নিয়ে চিন্তা করব যেখানে $q_x = p_x - p_z, q_y = p_y - p_z$ এবং আমাদের শর্ত হল $\sum a_i q_{xi} = \sum a_i q_{yi} = 0$. বলতে হবে এর সমাধান আছে কিনা।

এখনই হল মজার অংশ। এর সমাধান কি? যদি চিন্তা কর তাহলে দেখবে আমাদের সকল q বিন্দু গুলি যদি প্লট কর এবং তার convex hull নাও, এরপর যদি দেখ মূল বিন্দু সেই convex hull এর ভেতরে তার মানে সমাধান আছে। আর না থাকলে বুঝতে হবে সমাধান নেই। মূল বিন্দু convex hull এর ভেতরে না থাকলে যে সমাধান নেই তা তো বুঝাই যায়। কারণ মূল বিন্দু সাপেক্ষে সকল বিন্দু একই দিকে, সুতরাং কেউ কাউকে cancel করে না। কিন্তু মূল বিন্দু ভেতরে থাকলে যে সমাধান আছে তা বুঝা একটু কঠিন। খেয়াল কর আমাদের কিন্তু কিছু কিছু $a_i = 0$ হতে পারে, সবাই শূন্য না হলেই

হল। সুতরাং আমরা এই convex hull এর উপর থেকে আসলে তিনটি বিন্দু নির্বাচন করতে পারি যার ভেতরে মূলবিন্দু আছে (যেহেতু কনভেক্স হালের ভেতরে মূল বিন্দু ছিল)। বাকি গুলর a_i আমরা শূন্য করে দিতে পারি। এখন আমরা দেখি যে এই বাকি তিনটি বিন্দুর a_i কি বের করা যায় কিনা যেন আমাদের সমীকরণ সিদ্ধ্য করে। এখন এখানে দুটি equation, একটি x এর আরেকটি y এর, আর ভ্যারিয়েবল তিনটি। এর মানে এর সমাধান আছেই। তবে $a_i \geq 0$ হবে কিনা তা একটু চিন্তার বিষয়। আমি নিজেও এর প্রমাণ এই মুহূর্তে করতে পারছি না। কিন্তু ঐ যে, প্রোগ্রামিং কন্টেস্টে সব কিছু যে প্রমাণ করতে হয় তা কথা নেই। তোমার যদি আগ্রহ থাকে তাহলে প্রমাণ করতে পার, আর তা নাহলে intuition এর উপর ভিত্তি করে করে ফেলতে পার।

১১.১ অনুশীলনী

১১.১.১ সমস্যা

Simple

- Timus 1710 Boris, You Are Wrong!
- UVa 476 Points in Figures: Rectangles
- UVa 477 Points in Figures: Rectangles and Circles
- UVa 478 Points in Figures: Rectangles, Circles and Triangles
- UVa 191 Intersection
- UVa 438 The Circumference of the Circle
- UVa 453 Intersecting Circles
- UVa 10573 Geometry Paradox
- UVa 10725 Triangular Square
- UVa 10242 Fourth Point!!
- UVa 10341 Solve It
- UVa 218 Moth Eradication
- UVa 361 Cops and Robbers
- UVa 109 SCUD Busters
- UVa 596 The Incredible Hull
- Uva 10065 Useless Tile Packers
- UVa 10245 UVa The Closest Pair Problem

Easy

- Timus 1703 Robotic Arm
- Timus 3114 Spherical Mirrors
- UVa 10287 Gifts in a Hexagonal Box
- UVa 10289 A Square and Equilateral Triangles
- UVa 10353 Circles in Hexagon
- UVa 10345 Cricket/Football Goes Down
- UVa 10372 Leaps Tall Buildings (in a single bound)
- UVa 10398 The Golden Pentagon
- UVa 10566 Crossed Ladders
- UVa 10668 Expanding Rods
- UVa 10598 Find the Latitude
- UVa 811 The Fortified Forest
- UVa 10078 The Art Gallery
- UVa 10209 Is This Integration?
- UVa 10215 The Largest/Smallest Box
- UVa 11186 Circum Triangle

Medium

- UVa 11580 Finding the Transmitter
- UVa 10402 Triangle Covering
- UVa 11123 Counting Trapezoid
- UVa 11009 Geometry Once Again
- UVa 10322 The Four in One Stadium
- UVa 10386 Circles in Triangle
- UVa 10631 Normals
- UVa 819 GIfts Large and Small
- UVa 10135 Herding Frosh
- UVa 10256 The Great Divide
- UVa 10210 Romeo & Juliet!
- UVa 10251 Min-Max Cake
- UVa 10481 The Gift Wrappers of Hollywood