Handbook for Sports Programmers(Beginner)

Special Group Interested in Programming Contest

Online Programming Camp 2020

Organized by

Tanmoy Tapos Datta

Special thanks to

Rezaul Hoque

Department of Computer Science and Engineering

Khulna University of Engineering & Technology

সূচীপত্র

প্রোগ্রামিং কন্টেস্টঃ প্রয়োজনীয়তা এবং SGIPC এর যাত্রা - 02

কেন আমি প্রোগ্রামিং শিখবো? - 05

কেন প্রোগ্রামিং কন্টেস্ট? - 10

মিহাই পাত্রাস্কুর চিঠি - 14

Advice from Nafis Sadique - 18

Advice from Anudeep Nekkanti - 20

সমস্যা সমাধানে মনস্তত্ত্ব - 22

Life of an IOI contestant - 28

Nick Wu’s experience - 31

কিভাবে ভাল প্রোগ্রামার হওয়া যায়? - 33

Teach Yourself Programming in Ten Years - 38

কনটেস্ট্যান্টদের জন্য বেসিক সিপিপি - 45

শর্টেস্ট পাথের অ্যালগরিদম - 64

শর্টেস্ট পাথ - প্রবলেম নিয়ে বকর বকর - 74

টাইম কমপ্লেক্সিটি - 85

অ্যালগরিদম কমপ্লেক্সিটি(বিগ “O” নোটেশন) - 88

Attacking Recursions and Practice Recursion - 95

রিকার্শন, সার্চিং এবং ডাইনামিক প্রোগ্রামিং - 107

FAQ of Competitive Programming - 115

**Most of the contents of this document are collected from various posts on the internet. For every article of this document, the source/author of the original articles are linked with hyperlink.

প্রোগ্রামিং কন্টেস্টঃ প্রয়োজনীয়তা এবং SGIPC এর যাত্রা

ABM Junaed, 1st president of SGIPC (Competitive Programming Club of KUET)
Software Engineer, Germany

যেভাবে শুরু

আমি সেকেন্ড ইয়ারে উঠার পরে IUT তে একটা প্রোগ্রামিং কন্টেস্টে অংশ নেই, সেটাই ছিল লাইফের ১ম কন্টেস্ট যেখানে বিভিন্ন ভার্সিটি থেকে টিমগুলো অংশগ্রহন করেছিল। কন্টেস্ট শুরুর আগে ভাবছিলাম যে এক্কেবারে ফাটায়া দিব। কত প্রোগ্রামিং করলাম জীবনে!

কন্টেস্ট শুরু, র‍্যাঙ্ক লিস্টে দেখি বিভিন্ন টিম একটার পরে একটা সল্ভ করেই যাচ্ছে। আর আমরা যাই সাবমিট করি না কেন তাও রং এন্সার, আর অনেকগুলা প্রবলেম তো বুঝতেই পারছি না কিভাবে সল্ভ করতে হবে! কি যে একটা অবস্থা তখন বলে বুঝাতে পারব না।
কন্টেস্ট শেষ, হতাশায় কারো সাথে কথাও বলতে পারতেছি না। মাত্র ১টা বা ২টা মনে হয় সল্ভ করেছিলাম। পরের কয়টা দিন যে কি পরিমাণ হতাশায় কেটেছে বলে বুঝাতে পারব না। এরপর শুরু হল ঘাটাঘাটি করা যে কেন ওরা এত ভাল করল কন্টেস্টে। এর পর দেখলাম যে কন্টেস্টে বিভিন্ন ক্যাটাগরির প্রবলেম থাকে। dynamic programming, greedy, graph এর বিভিন্ন প্রকার, bit masking সহ আরও বেশ কিছু ক্যাটাগরি। ভাল করতে হলে আসলে এই সব ক্যাটাগরি থেকে প্রচুর পরিমাণ প্রব্লেম সল্ভ করতে হবে, দরকার ভাল ট্রেনিং। আফসোস, এসব না জেনেই গিয়েছিলাম কন্টেস্টে, রেজাল্ট তো খারাপ হবেই! বিভিন্ন ভার্সিটিতে বাইরে থেকে টাকা দিয়ে ভাল ট্রেইনার আনে। তখন থেকেই চিন্তা করলাম একটা কালচার তৈরি করতে হবে, যাতে করে আমার জুনিয়ররা আমার মত অবস্থায় না পড়ে। একটা প্লাটফর্ম দরকার যেখানে আমরা সবাই একসাথে প্র্যাকটিস করতে পারি, নলেজ শেয়ার করতে পারি আর ভবিষ্যতের জন্য ভাল কন্টেস্ট প্রোগ্রামার গড়ে তুলতে পারি।

কার্যক্রম

এরপর আমরা শুরু করলাম একটা প্রোগ্রামিং ক্লাব, নাম দেয়া হল SGIPC. আর সেই ক্লাবের ১ম প্রেসিডেন্ট নির্বাচিত হলাম আমি। তো আমরা SGIPC শুরু করার আগে যখন ১ম বর্ষে ছিলাম এবং সম্ভবত ২য় বর্ষের শুরুর দিকেও আমাদের কিছু কন্টেস্ট নিয়েছিলেন আমাদের শ্রদ্ধেয় সিনিয়র ভাইরা। মিফতাহ ভাই, নাফি ভাই, আসাদ ভাই, রন্জু ভাই সহ আরও বেশ কয়েকজন সিনিয়র ভাইদের মাধ্যমেই আমাদের এই হাতেখড়ি।
এর পরে এই ক্লাব থেকে আমরা বিভিন্ন উদ্যোগ নিয়েছি কুয়েটে প্রোগ্রামিং কন্টেস্টের একটা ভাল কালচার গড়ে তুলতে। শুরু হল প্রতি সপ্তাহে কন্টেস্ট। ধীরে ধীরে শুরু হল ১ম বর্ষের জন্য সি এর উপরে ওয়ার্কশপ, এলগরিদমের উপরে ওয়ার্কশপ, প্রোগ্রামিং কন্টেস্টের উপরে বিভিন্ন ওয়ার্কশপ। আমরা প্রতিটা ব্যাচ থেকেই পেয়ে যাই বেশ প্রতিভাবান, hard working কয়েকজন প্রোগ্রামার।
আমরা রেগুলার উইক্লি কন্টেস্ট আয়োজন করতাম, সেখানে ৩টা করে প্রব্লেম থাকত সাধারণত। codeforces এ যেই কন্টেস্ট হতো তখনও আমরা ল্যাব খোলা রাখতাম যাতে করে সবাই একসাথে বসে কন্টেস্ট করা যায়। বড় কিছু কন্টেস্ট থাকত প্রতি সেমিস্টারে, একদম ICPC স্টাইলে, ১০টার মত প্রব্লেম থাকত, যাতে করে লম্বা সময় ধরে বসে থেকে কন্টেস্টের অভ্যাসটা গড়ে উঠে।
আমাদের কাছে একসময় ল্যাবের চাবি দেয়া হয়েছিল, মনে আছে আমরা প্রায় সারা রাত ল্যাবে থেকে প্র্যাকটিস করতাম। রাতে ল্যাবে বসে প্রবলেম সল্ভের সময়গুলো সত্যিই খুব অসাধারণ ছিল।

ডিপার্টমেন্টের সাপোর্ট

এবার বলি ডিপার্টমেন্ট থেকে আমরা যে অকুন্ঠ সাপোর্ট পেয়েছি তার কথা। SGIPC এর শুরুর দিকে ডিপার্টমেন্ট এর হেড ছিলেন শ্রদ্ধেয় হাশেম স্যার। ওনার কাছ থেকে আমরা বেশ সাপোর্ট পেয়েছি। আর আমাদের সাথে ছিলেন তনয় স্যার। উনি যে কি পরিমাণ হেল্প করেছেন সেটা বলে বুঝাতে পারব না, স্যার যখন হায়ার স্টাডিজের জন্য দেশের বাইরে চলে যাচ্ছিলেন তখন মনে হচ্ছিল যে আমাদের কার্যক্রম থমকে পড়বে।
এরপর আমাদের জন্য এগিয়ে এলেন আমজাদ স্যার। সেই সময় আমদের যিনি সবচেয়ে বেশি সাপোর্ট দিয়ে গেছেন তিনি আমজাদ স্যার। রন্জু স্যার, আসাদ স্যার, অভি স্যার ওনাদের কাছে যখনি কোন হেল্পের জন্য গিয়েছি ওনারা সর্বোচ্চ সাপোর্ট দিয়েছেন আমাদের। আমজাদ স্যারকে একবার গভীর রাতে ফোন দেয়ার ঘটনা আরেকদিন না হয় বলব। স্যার রাতে আমাদের ওয়ার্কশপের সময়, উইকেন্ডে কন্টেস্টের সময় আমাদের জন্য বসে থাকতেন। স্যারের কাছে আমরা খুবই কৃতজ্ঞ। কন্টেস্টে টিম পাঠানোর জন্য টাকার ব্যবস্থা করে দেয়া, managerial দিক গুলো দেখা, ল্যাবের ব্যবস্থা করে দেয়া, ল্যাবের চাবির ব্যবস্থা করে দেয়া সহ সব রকম সহযোগিতা স্যারের কাছ থেকে পেয়েছি।
একেকটা টিমের রেজিস্ট্রেশানের জন্য দরকার পড়ত ৩-৫ হাজার টাকা। আমাদের সিনিয়ররা বিভিন্ন সময় আমাদের রেজিস্ট্রেশান ফি দিয়ে হেল্প করেছেন।
লাস্ট যে বার আমি ICPC তে অংশ নেই ২০১১ বা ২০১২ তে,, সেবার মনে আছে কুয়েট থেকে ৫টা টিম অংশ নেয়। এক সময় একটা টিম করতেই আমাদের একটু কষ্ট হয়ে যত, সেখানে ৫টা টিম আসলেই বিরাট ব্যাপার! এবং প্রতিটা টিমই বেশ প্রতিভাবান ছিল, সব টিমেরই বেশ অনেকগুলো করে প্রবলেম সল্ভ করা ছিল।

কতটা জরুরী

প্রোগ্রামিং কন্টেস্ট করা কতটা জরুরী, বা বিভিন্ন অনলাইন জাজ থেকে প্রবলেম সলভঁ করা কতটা দরকারি তা আমি দুইটা ভাগে ভাগ করব।

১। নিজের স্কিল ডেভেলপ করাঃ

১ম ব্যাপার হল নিজের স্কিল ডেভেলপ করা। আমি প্রায় সব ক্যাটাগরি থেকে প্রব্লেম সল্ভ করেছিলাম, তাই প্রব্লেম দেখে মোটামুটি বলে দিতে পারতাম এই প্রবলেমটা কিভাবে সল্ভ করতে হবে। লজিক ডেভেলপমেন্টের জন্য প্রব্লেম সল্ভ করা খুব জরুরী। আমার মনে আছে, প্রফেশনাল লাইফে একটা প্রজেক্টে বেশ কঠিন একটা কাজ বিট মাস্কিং এর নলেজ থাকার কারণে আমি খুব সহজে সল্ভ করে ফেলেছিলাম।
জার্মানিতে আমি দেখেছি এরা স্কুল লেভেল থেকেই প্রোগ্রামিং করে। তাই এদের লজিক বেশ ভাল হয়। আমাদের যেহেতু ভার্সিটিতে ঢুকার আগে প্রোগ্রামিং করা হয় না, আর ভার্সিটিতেও ল্যাবে যে অল্প কিছু প্রবলেম সল্ভ করি তাতে নিজের প্রোগ্রামিং স্কিল শার্প করা খুব একটা সহজ না। তাই আমার সাজেশান হবে রিকার্শান, ডায়নামিক প্রোগ্রামিং, গ্রাফ সহ অন্যান্য টপিকগুলো থেকে কিছু প্রবলেম সল্ভ করা। তাহলে নিজের প্রোগ্রামিং স্কিলও শার্প হবে, কনফিডেন্সও বেশ বাড়বে।

২। জব সেক্টরেঃ

বাংলাদেশে দেখতাম প্রোগ্রামিং কন্টেস্টকে বেশ value দেয়া হত ইন্টারভিউ তে। ইন্টারভিউতে যে সব প্রবলেম সল্ভ করতে দিত সেগুলাও দেখতাম কন্টেস্ট এর প্রবলেম সল্ভ করা থাকলে খুব সহজ হয়ে যেত। তাই বাংলাদেশে জব সেক্টরে কন্টেস্টের ভ্যালু বেশ ভাল।

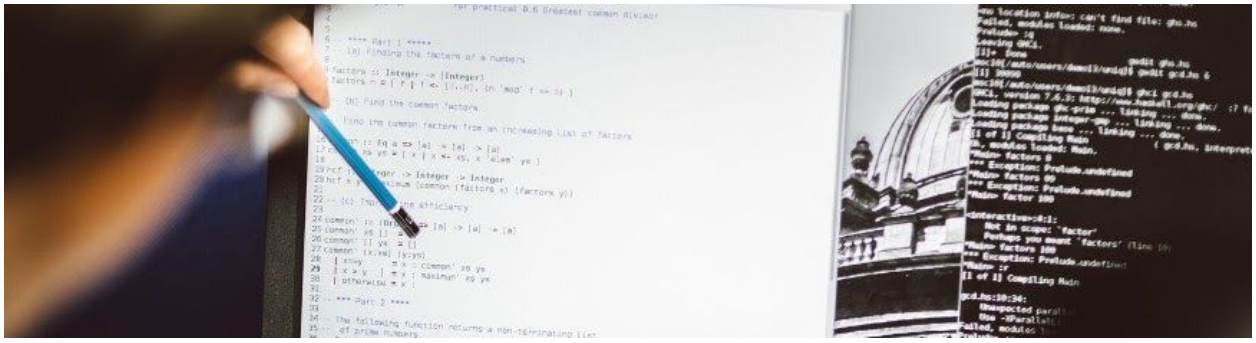
এবার আসি দেশের বাইরের কথায়। অন্য অনেকের মতই আমারও একটা ভুল ধারণা ছিল যে গুগল, ফেসবুকে প্রোগ্রামার হিসেবে কাজ করতে হলে একমাত্র মাধ্যম হল প্রোগ্রামিং কন্টেস্টে খুব ভাল প্রোফাইল থাকা। কিন্তু কন্টেস্টে একদম টপে থাকে হাতে গোনা কয়েকজন, হাজার খানেক প্রব্লেম সল্ভ করার মত সময় সবার থাকেও না, আর যারা করে তারা সবাই কিন্তু গুগলেও যায় না। আমি অনেককেই দেখেছি অনেক বড় বড় কোম্পানিতে জব করতে যারা প্রোগ্রামিং কন্টেস্ট করত না। আর শুধু গুগল, ফেসবুক ছাড়াও প্রোগ্রামারদের জন্য আরো অনেক ড্রিম কোম্পানি কিন্তু আছে।
আমি জার্মানিতে সফওয়ার ইঞ্জিনিয়ার হিসেবে কাজ করছি বেশ কিছু দিন হল। এরা কি দেখে রিক্রুট করে সেটা আমি দেখেছি। জার্মানিতে এরা ধরেই নেয় যে তোমার প্রোগ্রামিং এবিলিটি মোটামুটি ভাল, কারণ এরা স্কুল থেকেই প্রোগ্রামিং শেখায়। তাই এরা জবে রিক্রুটের ক্ষেত্রে প্রাধান্য দেয় জব রিলেটেড স্কিল সেট। এরা বিভিন্ন OOP related প্রশ্ন করে, ডাটাবেজের কিছু প্রশ্ন করে, যে প্ল্যাটফর্মে কাজ করতে হবে সে রিলেটেড প্রশ্ন করে। অন্যান্য দেশের কথা আমি বলতে পারব না, তবে এসব প্রশ্ন সব জায়গাতেই কমন হবার কথা। দেশের বাইরে থেকে ভাল একটা মাস্টার্স থাকলে দেখা যায় অনেক বড় কোম্পানিতে সহজে জয়েন করা যায়। Important হচ্ছে প্রব্লেম সলভ এর এবিলিটি তৈরি করা, লজিক ডেভেলপ করা, oop, design pattern সম্পর্কে ভাল ধারণা থাকা।
নিজের প্রোগ্রমিং স্কিল ডেভেলপ করা, চিন্তা করার এবিলিটি বাড়ানোর জন্য প্রোগ্রামিং কন্টেস্ট, uva বা অন্যান্য কন্টেস্টের প্রবলেম সল্ভ করা খুবই দরকারি, তবে মনে রাখতে হবে একজন ভাল সফটওয়ার ইঞ্জিনিয়ার হবার জন্য এটাই সব না। এর সাথে OOP ভালভাবে শিখতে হবে, design pattern, software architecture,software engineering, database এর বিভিন্ন জিনিস সম্পর্কে ভাল ধারণা থাকতে হবে।

২০০৯ এ শুরু হয়েছিল আমাদের SGIPC এর যাত্রা। ধীরে ধীরে আমরা অনেক ভাল ভাল প্রোগ্রামার পেয়েছি, এখন প্রোগ্রামিং কন্টেস্টে দেখি কুয়েটের অনেকেই বেশ ভাল করছে, অনলাইনে, অফলাইনে। এ যেন এক চারা গাছের মহীরুহ হয়ে উঠবার পথচলা।

কেন আমি প্রোগ্রামিং শিখবো?

শাফায়েত আশরাফ | জানুয়ারি ২১, ২০১৩

সকালে উঠেই টপকোডারে এ লেখা দেখলাম "একটি শিশুকে একই আইফোন দিলে সে দিনরাত অ্যাংগ্রি বার্ডস খেলবে, শিশুটিকে কোডিং শিখালে সে আইফোনটার জন্য সফট্ওয়্যার তৈরি করবে" দারুণ এই লেখাটা দেখে মনে হলো কেন আমরা প্রোগ্রামিং বা কোডিং শিখবো সেটা নিয়ে বাংলায় কিছু লিখি। এ লেখাটি প্রোগ্রামিং নিয়ে যাদের কোনো ধারণা নেই বা খুব সামান্য ধারণা আছে তাদের আগ্রহী করে তোলার একটি ছোট্ট প্রচেষ্টা।



কম্পিউটার একটি অসম্ভব ক্ষমতাবান কিন্তু নির্বোধ একটি যন্ত্র। একটি যন্ত্র ৫০জন সাধারণ মানুষের কাজ একাই করতে পারে কিন্তু ৫০টি যন্ত্র একটি অসাধারণ মানুষের কাজ করতে পারেনা(Hubbard, Elbert)। প্রোগ্রামিং শিখে আমরা একেকজন হয়ে উঠতে পারি সেই মানুষটি যে এই যন্ত্রকে ইচ্ছামত কথা শোনাতে পারে। তুমি যা বলবে যেভাবে কম্পিউটার তাই করবে, এটাই হলো সোজা কথায় প্রোগ্রামিং। হয়তো বলতে পারো এখনইতো কম্পিউটার সেটা করে, আমি গান শুনাতে বললে সে শুনিয়ে দেয়, আমি গেম খেলতে চাইলে সে আমার সাথে খেলতে শুরু করে। কিন্তু আসল ব্যাপারটা হলো একজন প্রোগ্রামার আগেই কম্পিউটারকে বলে রেখেছে যে তুমি গান শুনতে চাইলে সে যেন শুনিয়ে দেয়। সে যদি বলে রাখতো গেম খেলতে চাইলে পড়তে বসার উপদেশ দিতে তাহলে কম্পিউটার তাই করতো, তোমার কিছু করার থাকতোনা। প্রোগ্রামার হলো সে যার কথায় কম্পিউটার উঠা-বসা করে। দারুণ একটা ব্যাপার এটা, তাইনা?

কিন্তু তুমি কেন প্রোগ্রামিং শিখবে? বড় বড় কথা বলার আগে সবথেকে প্রথম কারণ আমি বলবো কারণ "প্রোগ্রামিং দারুণ মজার একটি জিনিস!"। কম্পিউটারের সাথে অন্য যন্ত্রের বড় পার্থক্য হলো এটা দিয়ে কতরকমের কাজ করানো যায় তার সীমা নেই বললে খুব একটা ভুল হবেনা। তাই প্রোগ্রামিং জানলে যে কতকিছু করা যায় তার তালিকা করতে বসলে শেষ করা কঠিন। তুমি দিনের পর দিন প্রোগ্রামিং করেও দেখবে জিনিসটা বোরিং হচ্ছেনা, প্রায় প্রতিদিনই নতুন মজার কিছু শিখছো, নতুন নতুন টেকনোলজী আবিষ্কারের সাথে সাথে তুমি আরো অনেক রকম কাজ করতে পারছো অথবা তুমিই করছো নতুন আবিষ্কার! আজ হয়তো জটিল কোনো সমীকরণ সমাধান করার জন্য ফাংশন লিখছো, কাল এসব ভালো লাগছেনা বলে লাল-নীল রঙ দিয়ে একটি অ্যানিমেশন বানাতে বসে গেলে, তোমার সৃষ্টিশীলতার সবটুকুই কাজে লাগাতে পারবে প্রোগ্রামিং এর জগতে।

ছবি: শাহরিয়ার মঞ্জুর, বিশ্বের সবচেয়ে সম্মানজনক প্রোগ্রামিং প্রতিযোগীতার বাংলাদেশি জাজ

A computer is a stupid machine with the ability to do incredibly smart things, while computer programmers are smart people with the ability to do incredibly stupid things. They are, in short, a perfect match. – Bill Bryson

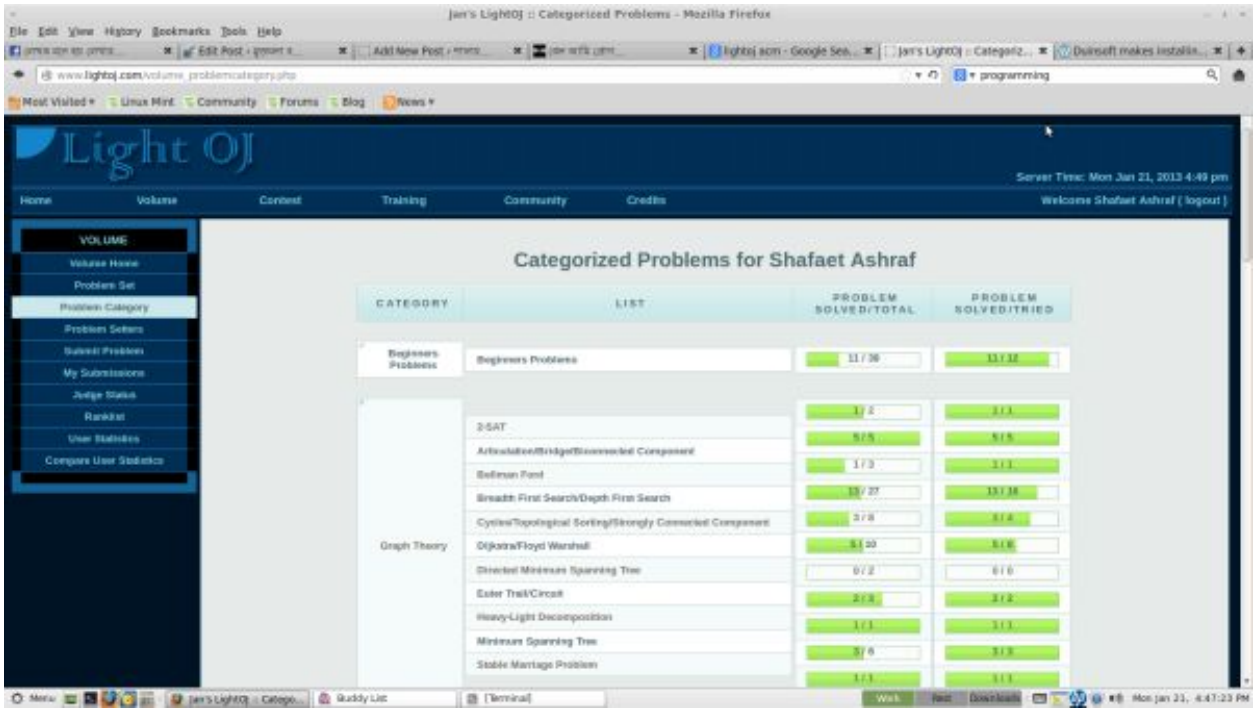
একটি স্কুল-কলেজ পড়ুয়া ছেলেমেয়ে কম্পিউটার বা মোবাইল দিয়ে কি করে? রাশিয়া-চীনের ছেলেমেয়েরা অনেকেই হয়তো অ্যাসেম্বলিতে কোড লিখে, কিন্তু জরিপ না করেও বলা যায় আমাদের দেশে বেশিভাগই মুভি দেখা, ফেসবুক , গেমস ছাড়া খুব বেশি কিছু করেনা। আসলে কম্পিউটার দিয়ে কি করা যায় তার ধারণাও অনেকের নাই। ছেলে বা মেয়েটিকে প্রোগ্রামিং শিখিয়ে দেয়া হলে তার জগৎটাই পাল্টে যাবে। সে তখন সারাদিন গেমস না খেলে হয়তো একটি গেমস বানিয়ে ফেলবে। আমি বাংলাদেশেরই কিছু স্কুল-কলেজ পড়ুয়া প্রোগ্রামারদের জানি যারা বাংলা কিবোর্ড নিয়ে কাজ করে, ওপেন সোর্স কমিউনিটিতে অবদান রাখে। প্রোগ্রামিং জানলে তুমি বুঝতে পারবে কম্পিউটার শুধু বিনোদনের যন্ত্র নয়, কম্পিউটার তৈরা করা হয়েছিল এর ক্ষমতাকে ব্যবহার করে বড় বড় গবেষণা,হিসাব করার জন্য, তুমি যদি গবেষণা নাও করো অন্তত এই ক্ষমতাটা ব্যবহার শিখবে, সৃষ্টিশীল অনেক কাজ করতে পারবে। কম্পিউটারের জগতে অসাধারণ কিছু অগ্রগতি হয়েছে খুব কম বয়েসী প্রোগ্রামারদের দিয়ে, বিল গেটস স্কুলে থাকতেই চমকে দেয়ার মত কিছু প্রোগ্রাম লিখেছিলেন, প্রোগ্রামিং কনটেস্টে হাইরেটেড কোডারদের অনেকেই স্কুল-কলেজ এখনও শেষ করেনি।

প্রোগ্রামিং করা মানে আনন্দের সাথে শেখা। এই শেখাটা খালি কম্পিউটারের মধ্য সীমাবদ্ধ না, অধিকাংশ ভালো প্রোগ্রামারদের খুবই ভালো গাণিতিক এবং লজিকাল জ্ঞান থাকে। দাবা খেলার মতোই প্রোগ্রামিং পুরোটাই লজিকের খেলা, কোন কাজের পর কোনটা করলে কি হবে, কিভাবে করলে আরো দ্রুত ফলাফল আসবে এইসব নিয়ে চিন্তা করতে করতে মস্তিষ্কের লজিকাল সেক্টরটা ডেভেলপ করে। আমার মতে চিন্তা করার মত আনন্দের এবং গুরুত্বপূর্ণ কাজ ২য়টি নেই। বিশেষ করে কম বয়সে প্রোগ্রামিং শিখালে সে চিন্তাশক্তি বৃদ্ধির যেই সুফলটা পাবে সেটা সারাজীবন কাজে লাগবে, সে যদি প্রোগ্রামিং পরে ছেড়েও দেয় তারপরেও চিন্তা করার ক্ষমতাটা থেকে যাবে।

প্রোগ্রামিং কি শুধু কম্পিউটার সাইন্স যারা পড়ে বা পড়তে চায় তারা শিখবে? সেটার কোনো যুক্তি নেই, তুমি যেই বিষয় নিয়েই পড়ছো বা পড়তে চাও, প্রোগ্রামিং তুমি আনন্দের জন্যই শিখতে পারো এবং চাইলে তোমার কাজেও লাগাতে পারো।

তুমি বিজ্ঞানের যেকোনো বিষয়ে লেখাপড়া করলেতো কথাই নেই, তোমার গবেষণায় প্রতি মূহুর্তে কম্পিউটার লাগবে, তুমি বিজনেস, আর্টস পড়লেও প্রোগ্রামিং কাজে লাগবে। তুমি কোম্পানির জন্য দারুণ একটি ওয়েবসাইট বানাতে পারো, একটি সফটওয়্যার বানাতে পারো যেটা যেসব কাজ বোরিং সেগুলো স্বয়ংক্রিয় ভাবে করে দিবে! আমি অনেক সময় ছোটো-খাটো কিন্তু বোরিং কাজ করার সময় চট করে একটা স্ক্রিপ্ট লিখে ফেলি, তারপর সেটাকে কাজ করতে দিয়ে ঘুম দেই!

প্রোগ্রামিং শেখা কি খুব কঠিন? উত্তর হলো হ্যা,যদি তোমার আগ্রহ না থাকে এবং কেও তোমাকে জোর করে শেখায়। যদি একবার মজা পেয়ে যান তাহলে এরপর কারো শেখানো লাগবেনা, নিজেই সব শিখে ফেলতে পারো। আমার উপদেশ হবে ২-৩ সপ্তাহ প্রোগ্রামিং করার পর যদি তোমার ভালো না লাগে তাহলে জোর করে করার দরকার নাই, এটা তোমার জন্য না, অন্য যেটা ভালো লাগে সেই কাজ করো। যদি একবার ভালো লাগে বাজী ধরে বলতে পারি কোড লিখতে লিখতে তুমি প্রায়ই খাবার কথাও ভুলে যাবে। যেকোন কাজের জন্যই সবথেকে গুরুত্বপূর্ণ ব্যাপার হলো ভালো লাগা, যেটা ভালো লাগেনা সেটা করার কোনো অর্থ আমি দেখিনা কারণ দুইদিন পর যা শিখসি সব ভুলে যাবো।



ছবি: lightoj, ঢাকা বিশ্ববিদ্যালয়ের জানে আলম জানের তৈরি করা অনলাইন জাজ যেখানে প্রবলেম সলভ করে সারা পৃথিবীর কোডাররা

শুরু কিভাবে করবে? তোমার যদি ইন্টারনেট কানেকশন থাকে তাহলে কথাই নেই, ইন্টারনেটে অসংখ্য টিউটোরিয়াল আছে। ইংরেজীর পাশাপাশী বাংলা কিছু ভালো রিসোর্সও তুমি পাবে। যেমন শ্রদ্ধেয় রাগিব হাসানের shikkok.com ওয়েবসাইট বা ফাহিম ভাইয়ের পাইথন সাইট। এছাড়া খান একাডেমিতেও প্রোগ্রামিং এর ভিডিও আছে, বরাবরের মতই খুবই সুন্দর করে বুঝিয়েছেন সালমান খান। ইন্টারনেট না থাকলে তোমাকে বই জোগাড় করতে হবে, ব্যাক্তিগত ভাবে বিগিনারদের জন্য

আমি ইন্টারনেটের থেকে বইকেই বেশি গুরুত্ব দিবো। প্রোগ্রামিং এর বইয়ের অভাব নেই দোকানে, তামিম শাহরিয়ার সুবিন ভাইয়ের একটি দারুণ বাংলা বই আছে। তবে একটা ব্যাপারে সতর্ক থাকবে "৭দিনে প্রোগ্রামিং শেখা" এই ধরণের চটকদার বইয়ের বা সাইটের ধারেকাছে যাবে, এগুলো সবকিছু ঝাপসা ভাবে শেখাবে, হার্ভার্ড শিল্ডের বইয়ের মত নামকরা এবং ভালো বই দেখে শিখো, বেসিক জিনিসগুলো পরিষ্কার হবে। এছাড়া লাগবে প্রোগ্রামিং এর জন্য কিছু সফটওয়্যার, এগুলোও সহজেই জোগাড় করতে পারবে। এরপর শুরু করে দাও কোড লেখা!! প্রথম ২ সপ্তাহ আপনার বেশ ঝামেলা লাগবে কারণ বিষয়টা নতুন, একটু পরপর আটকে যাবে, তারপর হঠাৎ দেখবেন সবকিছু সহজ হয়ে গিয়েছে, মূহুর্তের মধ্যেই ১০০ লাইনের কোড লিখে ফেলেছো। প্রোগ্রামিং শেখার প্রধান শর্ত হলো হাল ছাড়া যাবেনা। প্রথম দিকে কোনো কোড কপি পেস্ট করবেনা, নিজের হাতে লিখবে।

A good programmer is someone who looks both ways before crossing a one-way street. — Doug Linder, systems administrator

চাকরী-ক্যারিয়ার নিয়ে সবার মধ্যেই অনেক টেনশন থাকে। আনন্দের জন্য প্রোগ্রামিং শিখলেও এটা তোমার ক্যারিয়ারে খুবই গুরুত্বপূর্ণ। তুমি প্রোগ্রামিং জানলে নিশ্চিত থাকতে পারো কাজের কোনো অভাব জীবনে হবেনা। তুমি কোনো চাকরী না করেও ফ্রি-ল্যান্স কাজ করতে পারবে, এমনকি ছোটোখাট একটা কোম্পানিও খুলে বসতে পারবে। আমি আশেপাশে অনেককে দেখেছি কয়েক বন্ধু মিলে একটি ছোট কোম্পানি খুলে স্বাধীনভাবে কাজ করে, কি দারুণ একটা ব্যাপার! প্রোগ্রামিং জানার আরেকটি দারুণ ব্যাপার হলো তুমি ভালো কোনো কাজ করলে খুব সহজেই সারা বিশ্ব জেনে যাবে। পৃথিবীর আরেক প্রান্তের মানুষ তোমার বানানো সফটওয়্যার দিয়ে গান শুনবে, তোমার অপারেটিং সিস্টেম বুট করবে, আবার পিসি হ্যাং করলে হয়তো আপনাকেই গালি দিবে!! গুগলের মতো কোম্পানিতে কাজ করতে চাইলে তোমার কিছু করতে হবেনা, আপনার কাজের খ্যাতিতে তারাই তোমাকে এসে অফার দিবে। তবে প্রোগ্রামিং শেখার উদ্দেশ্য কখনোই গুগলে চাকরী বা খ্যাতি অর্জন হওয়া উচিত নয়, শিখবে আনন্দের জন্য, জানার জন্য।

সি বা জাভার মতো প্রোগ্রামিং ল্যাংগুয়েজ শেখা মানেই কিন্তু তুমি প্রোগ্রামিং শিখে ফেলোনি। ল্যাংগুয়েজ শেখা খুব সহজ কাজ, প্রথমে একটা কষ্ট করে শিখে ফেললে এরপর যেকোনো ল্যাংগুয়েজ শেখা যায়। তোমাকে খুবই ভালো লজিক ডেভেলপ করতে হবে, অ্যালগোরিদম আর ডাটা স্ট্রাকচার নিয়ে পড়ালেখা করতে হবে, গণিত জানতে হবে, তাহলেই তুমি একজন ভালো প্রোগ্রামার হয়ে উঠবে। তবে ভয়ের কিছু নেই, সবই তুমি ধীরে ধীরে শিখে ফেলতে পারবো, শুধু লাগবে চেষ্টা আর সময়। এটা আশা করবেনা যে ৬ মাসে তুমি অনেক ভালো প্রোগ্রামার হয়ে যাবে তবে লেগে থাকলে ২-৩ বছরে অবশ্যই মোটামুটি ভালো একটা লেভেলে তুমি পৌছাতে পারবে।

তুমি যদি কম্পিউটার সাইন্সের স্টুডেন্ট হও তাহলে এইসব কথাই তুমি হয়তো জানো, শুধু বলবো প্রোগ্রামিং কে আর ৫টা সাবজেক্টের মতো ভেবোনা, খালি সিজিপিএ বাড়াতে কোডিং শিখলে তোমার মতো অভাগা কেও নাই, প্রোগ্রামিং উপভোগ করার চেষ্টা করো, জানার আনন্দে শিখো।

আমার স্বপ্ন আমাদের দেশে একটা চিন্তা করার সংস্কৃতি তৈরি হবে। মানুষ একে অন্যের ব্যক্তিগত ব্যাপারে মাথা ঘামাবেনা, বরং মাথা ঘামাবে গাণিতিক সমস্যা নিয়ে, পাজল নিয়ে, অ্যালগোরিদম নিয়ে। ছেলেমেয়েরা তাদের মেধা গেমস খেলার কাজে না লাগিয়ে কাজে লাগাবে পৃথিবীর উন্নয়নে। বই পড়া, গণিত চর্চা করার পাশাপাশি প্রোগ্রামিং শিখা এই সংস্কৃতি শুরু করতে বিশাল একটি ভুমিকা রাখতে পারে। আমি মনে করি বর্তমান যুগে প্রোগ্রামিং শেখাটা অন্য যেকোন বিষয় শেখার মতই গুরুত্বপূর্ণ, কারণ আমাদের সব কাজে কম্পিউটার লাগে। তাই আপনার আশেপাশের ছেলেমেয়েদের গেমস খেলতে দেখলে তাদের প্রোগ্রামিং সম্পর্কে জানাও, উৎসাহিত করো, অবশ্যই জোর করে শেখানোর কোনো মানে হয়না, যার ভালো লাগবে সে শিখবে তবে সবাই অন্তত জানুক প্রোগ্রামিং কি, এছাড়া কিভাবে শেখার জন্য উৎসাহিত হবে? অনেকেই

ইউনিভার্সিটিতে আসার আগে জানেনা প্রোগ্রামিং বলে একটা বস্তু আছে! আর তোমরা প্রোগ্রামিং জানলে অন্যদেরও শিখতে সাহায্য করো, এভাবেই পরিবর্তন একসময় আসবেই, সবাই লজিক দিয়ে ভাবতে শিখবে, চিন্তা করার সংস্কৃতি তৈরি হবে।

শেষ করছি আমার খুব প্রিয় আরেকটি কোটেশন দিয়ে:

craftsman-ship has its quiet rewards, the satisfaction that comes from building a useful object and making it work. Excitement arrives with the flash of insight that cracks a previously intractable problem. The spiritual quest for elegance can turn the hacker into an artist.There are pleasures in parsimony, in squeezing the last drop of performance out of clever algorithms and tight coding. — steven skiena & miguel reville

হ্যাপি কোডিং!

কেন প্রোগ্রামিং কন্টেস্ট?

মীর ওয়াসি আহমেদ | মে ৩০, ২০১৩

আজ থেকে বছরখানেক আগে কোডফোর্সেস সাইটের কিছু পোস্ট এবং মন্তব্যে প্রোগ্রামিং কনটেস্টের সমালোচনা করা হয়, সেগুলোর জবাব দিতে এই পোস্টটি করেন কোডফোর্সেস টিম এর দলনেতা মাইক মির্জায়ানভ। মূল রাশান থেকে গুগল আর ইয়াহু ট্রান্সলেশন টুল ব্যবহার করে ইংরেজিতে অনুবাদ করে সেখান থেকে বাংলা অনুবাদ করা হয়েছে। মূল লেখাটা থেকে কিছুটা সংক্ষেপিত। অনুবাদ মীর ওয়াসি আহমেদের আর সম্পাদনায় ইকরাম মাহমুদ। মাইকের অনুমতিক্রমে এখানে লেখাটা প্রকাশ করা হল।

ওয়ার্ল্ড ফাইনালসে তার দলের সাথে মাইক (সবার বামে)। ফটো কার্টেসি – স্মার্কনিউজ।

ইদানিং কোডফোর্সেস এর বিভিন্ন পোস্টে এবং মন্তব্যে অনেকেই প্রোগ্রামিং কনটেস্টের সমালোচনা করছেন। সেগুলোর জবাবেই আমি এই নোট লিখছি।

এই বিষয়ে আমার দৃষ্টিভঙ্গি আসলে বেশ ইতিবাচক। আমি হয়তো বা পুরোপুরি নিরপেক্ষ হতে পারবো না - তারপরও চেষ্টা করছি।

আমি মাইক মির্জায়ানভ রাসিহোভিচ। আমাকে আপনি নাও চিনতে পারেন, আমি হচ্ছি সারাতভ স্টেট ইউনিভার্সিটির অলিম্পিয়াড ট্রেইনিং প্রোগ্রামের প্রধান। আমাকে বেশিরভাগ মানুষ চেনে কোডফোর্সেস প্রজেক্টের প্রতিষ্ঠাতা এবং দলনেতা হিসেবে। কোডফোর্সেস আমি বলবো আমার করা একটা দারুণ কাজ এবং খুবই আনন্দের সাথে জানাতে চাই, আমিই এর একমাত্র ডেভেলপার নই - আমার একটা অসাধারণ ডেভেলপার টিম আছে। আমি অনেকদিন ধরে প্রোগ্রামিং কনটেস্টের সাথে জড়িত, যতদূর মনে পড়ে বিশ্ববিদ্যালয়ের প্রথম বছর থেকেই। এসিএম আইসিপিসির ফাইনালে আমি ভালো করেছি, আমার কোচ করা দলগুলোও উল্লেখযোগ্য সাফল্য পেয়েছে। আমি বলতে পারি যে, আমি প্রোগ্রামিং কনটেস্টের একদম ভিতরের একজন লোক। 'কমার্শিয়াল প্রোগ্রামিং' কেও আমি বেশ কাছ থেকে দেখেছি। প্রায় ছ'মাসের মতো SaaS-প্রোডাক্ট কোম্পানি গ্রিড-ডায়নামিক্সের একটা গ্রুপের নেতৃত্বে ছিলাম। উল্লেখ্য, কোডফোর্সেসও আসলে বিশাল অংশে একটা ইন্ডাস্ট্রিয়াল এবং ইঞ্জিনিয়ারিং ডিজাইন।

যাই হোক, মূল প্রসংঙ্গে ফিরে আসি। দশ বছরেরও বেশি সময় প্রোগ্রামিং কনটেস্টের সাথে কমবেশি জড়িত থাকার পর আমার অভিমত হচ্ছে, প্রোগ্রামিং কনটেস্ট ছাত্রদের জন্যে খুবই গুরুত্বপূর্ণ এবং লাভজনক। আমি এর কারণগুলো কোন বিশেষ ক্রম না মেনে একে একে বলছি।

১. প্রোগ্রামিং কনটেস্ট অ্যালগোরিদম শেখায়। আমাদের দেশে অলিম্পিয়াড (বা কনটেস্ট) ছাড়া প্রায় কোথাওই ভালোভাবে অ্যালগোরিদম শেখানো হয় না। হয়তো দু-একটা ব্যতিক্রম আছে। প্রোগ্রামিং কনটেস্ট এমন একটা আন্দোলন, যেটা থেকে একটা ছেলে বা মেয়ে শিখতে পারে ডায়নামিক প্রোগ্রামিং, ডাটা স্ট্রাকচার, স্ট্রিং অ্যালগোরিদম - এরকম আরো অনেক কিছু। লক্ষণীয় ব্যাপার হল, কনটেস্টেন্টরা রেফারেন্স হিসেবে অনেকেই কোরমেনের বই অনুসরণ করে - যেটা আমাদের দেশের অ্যালগোরিদম কোর্সের পাঠ্যবই। কিন্তু এখানে (আমার বিশ্বাস বাইরের দেশেও) এই রকম একটা কোর্সে যথেষ্ট গভীরে গিয়ে পড়ানো হয় না এবং এই তত্ত্বীয় জ্ঞানের ব্যবহারিক প্রয়োগ কৌশলও খুব একটা শেখানো হয় না। অন্যদিকে এই অ্যালগরিদম বা ডাটা স্ট্রাকচারগুলোকে বাস্তব পৃথিবীর সমস্যা সমাধানের জন্য ব্যবহার করতে সক্ষম হওয়াই হচ্ছে প্রোগ্রামিং কন্টেস্টের মূল ব্যাপার।

২. কনটেস্ট দিয়েই অনেকে প্রোগ্রামিং শুরু করে। আমার ছাত্রদের মধ্যে এরকম অনেকেই ছিল যারা আগে প্রোগ্রামার ছিল না। কেউ কেউ ম্যাথ করত, কয়েকজন আবার ছিল গেমার। এইরকম ব্যাকগ্রাউন্ডের ছেলেমেয়েরা অনেকেই খুব আগ্রহ জাগানোর মতো ইন্টারেস্টিং প্রবলেম দেখে বা কন্টেস্টের খবর পেয়ে প্রোগ্রামিংয়ে আগ্রহী হয়। যেহেতু এখন আইটি ইন্ডাস্ট্রি অত্যন্ত দ্রুত এগিয়ে যাচ্ছে, কাজের ভালো সুযোগও তৈরি হচ্ছে। ফ্রেশ গ্র্যাজুয়েট প্রোগ্রামারদের ভালো বেতনের চাকুরী মোটামুটি নিশ্চিত। আমার দৃঢ় বিশ্বাস, যদি প্রোগ্রামিং কনটেস্ট না থাকতো, তাহলে ওরা হয়তো প্রোগ্রামিং শুরু করতো আরো কম ইন্টারেস্টিং বা কম সৃজনশীল পথে। অথবা তারা প্রোগ্রামিং এ আসার আগ্রহই হয়তো পেতো না।

৩. প্রোগ্রামিং কনটেস্ট প্রোগ্রামারদের বাগ-ফ্রি কোড লিখতে শেখায়, সব ধরণের কেস চিন্তা করতে শেখায়, শেখায় প্রোডাক্টিভ হতে। আমি এরকম অনেক দেখেছি, একজন নন কনটেস্টেন্ট ডেভেলপার কোড লিখছে, ব্রাউজার রিফ্রেশ করছে

(ক্লিক করছে, ফর্ম ফিল আপ করছে) এবং শেষে চেঁচিয়ে উঠছে "ধুর ছাই" বলে। তারপর সবকিছু আবার একদম গোড়া থেকে শুরু থেকে করছে। এই ব্যাপারটা তার জন্যে খুবই স্বাভাবিক। ঠিক একই জায়গায় একটা প্রোগ্রামিং কন্টেস্টেন্ট হয়তো হুবহু এই কাজগুলো করবে না। কারণ প্রোগ্রামিং কনটেস্টেন্টরা ডিবাগিং যাতে না লাগে (অথবা কম লাগে) সেজন্যে শুরুতেই বাগ-ফ্রি কোড লেখার চেষ্টা করে।

৪. প্রোগ্রামিং কনটেস্টের খুব গুরুত্বপূর্ণ দিক হচ্ছে এটা টিমওয়ার্ক শেখায়। একা কাজ করা আর একটা দলে অন্য মানুষদের সাথে কাজ করা - এই দুইয়ের মাঝে বিস্তর ফারাক। কোন কিছু নিয়ে অন্যদের সাথে মিলেমিশে গঠনমূলক আলোচনা করা এবং কোন সমস্যার সমাধান বের করা - এই ব্যাপারগুলো সবার মধ্যে সহজাতভাবে থাকে না। টিম কনটেস্ট এই জিনিসগুলো শেখায়। শেখায় অন্যদের কথা শুনতে, তাদের শক্তিমত্তা, দূর্বলতা এবং ব্যক্তিগত বৈশিষ্ট্যগুলোকে মাথায় রাখতে।

৫. প্রোগ্রামিং কনটেস্টেন্টদের বেসিকের ভিত মজবুত থাকে। তাছাড়া কনটেস্ট প্রাতিষ্ঠানিক শিক্ষা ছাড়া বেড়ে ওঠা প্রোগ্রামারদের বেসিক জিনিসগুলো শেখায়। আমি একবার একটা চাকরীর ইন্টারভিউ নিতে গিয়েছিলাম। অনেকেই বলতে পেরেছিল ডেটাবেইজ ইন্ডেক্স কি, কিন্তু খুব কম জনই বলতে পেরেছিল এটা কিভাবে ইমপ্লিমেন্ট করা যায় (অন্তত অ্যাপ্রক্সিমেটলি)। আপনি কি বলবেন, এটা জানার কোন প্রয়োজন নেই? আমি অনেককেই দেখেছি অনেক কনসেপ্ট, আর্কিটেকচার বা প্যাটার্ন জানে কিন্তু খুব সহজ কোন অ্যালগোরিদমের কম্পলেক্সিটি ক্যালকুলেট করে বের করতে পারে না।

৬. প্রোগ্রামিং কনটেস্ট থেকে পাওয়া মেডেল/সার্টিফিকেট ভালো চাকরি পেতে সাহায্য করে।

৭. প্রোগ্রামিং কনটেস্ট খুব ইন্টারেস্টিং মানুষজনের বন্ধুত্ব করার সুযোগ তৈরি করে। আমার সবচেয়ে কাছের বন্ধুরা - আমার প্রোগ্রামিং টিমমেটরা, যাদের সাথে থেকে আমি শিখেছি, যাদের কাছ থেকে শিখেছি। এরা সবাই খুবই ইন্টারেস্টিং, শিক্ষিত এবং চমৎকার কিছু মানুষ।

৮. শুধু বন্ধুত্ব নয়, কনটেস্ট করে অনেক ধরণের মানুষের সাথে পরিচিত হওয়া যায়। এই পরিচিতি যে কোন সময়ে কাজে লাগতে পারে।

৯. প্রোগ্রামিং কনটেস্ট ভ্রমণের সুযোগ করে দিতে পারে।

১০. প্রোগ্রামিং কনটেস্ট দ্রুত চিন্তা করতে শেখায়।

১১. খুব বেশি জায়গা নেই যেটাতে আমাদের দেশ (বা বিশ্ববিদ্যালয়) বিশ্বের আর সবার চাইতে এগিয়ে। দেশের (বা স্কুল বা বিশ্ববিদ্যালয়ের) এই রকম কোন অর্জনে আপনি যদি কোন ভূমিকা রাখতে পারেন, সেটা হবে ভীষণ মর্যাদাপূর্ণ এবং আপনি না চাইলেও সবার শ্রদ্ধা পাবেন। আমার ছাত্ররা যখন তাদের কোন পুরষ্কার নিতে মঞ্চে ওঠে, আমি আমার বুকের মাঝে তীব্র দেশপ্রেম অনুভব করি । যারা দেশের জন্যে গৌরব বয়ে আনে, তারা সত্যিকারের দেশপ্রেমিক। এই জয়গুলো খুব দরকারি - একটা ইতিবাচক ধারণা তৈরি করার জন্য। যেটা ছাড়া একটা অঞ্চলে বিনিয়োগ আকৃষ্ট করা বা ঐ অঞ্চলের সম্ভাবনা ফুটিয়ে তোলা কঠিন।

১২. প্রোগ্রামিং কনটেস্টেন্টদের মধ্যে একটা প্রথা চালু আছে। পুরনো কন্টেস্টেন্টরা বিশ্ববিদ্যালয়েই থেকে যান এবং শিক্ষকতা করেন। এটা খুবই প্রয়োজনীয় এবং খুবই ঠিক কাজ। মানতেই হবে, প্রোগ্রামিং খুবই পরিবর্তনশীল একটা ক্ষেত্র এবং প্রাচীনপন্থী বুড়ো প্রফেসররা নিত্যনতুন জ্ঞান দিতে পারেন না। তরুণ বিশ্ববিদ্যালয় শিক্ষকদের সম্পৃক্ততা – আইটি-শিক্ষা উন্নতির একমাত্র উপায়। দেখা গেছে সেরা গ্র্যাজুয়েটদের বিশ্ববিদ্যালয় ছেড়ে না যাবার পেছনে প্রোগ্রামিং কনটেস্টের একটা ভূমিকা আছে। এটা দারুণ একটা ব্যাপার।

১৩. প্রোগ্রামিং কনটেস্ট খুবই ইন্টারেস্টিং একটা জিনিস! প্রোগ্রামিং কন্টেস্টের সেই মুহূর্তগুলোকে আমি কখনো ভুলতে পারি না, যখন একটা প্রবলেম সলভ করার পর আমরা ওয়ার্ল্ড র‍্যাঙ্কিংয়ের শীর্ষে চলে গিয়েছিলাম। আমরা জানতাম চীনারা আমাদের ঘাড়ে নিঃশ্বাস ফেলছে...। কনটেস্ট থেকে আমি অনেক গুরুত্বপূর্ণ এবং ইন্টারেস্টিং জিনিস শিখেছি।

১৪. প্রোগ্রামিং কনটেস্টে ইন্ডাস্ট্রিয়াল ডেভেলপমেন্টের কিছু শেখানো হয় না ঠিক, কিন্তু দেখা গেছে যারা কনটেস্ট করেছে তারা খুব তাড়াতাড়িই সব কিছু শিখে নিতে পারছে। নিজ থেকে শেখা, দ্রুত চিন্তা করতে পারা এবং দ্রুত কোড করতে পারার কারণেই তাদের পক্ষে এটা সম্ভব হচ্ছে।

১৫. বিভিন্ন আইটি কোম্পানির ইন্টারভিউতে যেসব পাজল সলভ করতে দেওয়া হয়, সেগুলো প্রোগ্রামিং কনটেস্টেন্টদের পক্ষে সলভ করা সাধারণত খুব সহজ হয়। এধরণের কোম্পানিগুলো এমন লোকজন চায়, যারা কেবল কোন নির্দিষ্ট ল্যাঙ্গুয়েজ বা ফ্রেমওয়ার্কই জানে না, নিজে থেকে চিন্তাও করতে জানে।

এটা সত্যি যে, প্রোগ্রামিং কনটেস্টে উল্লেখযোগ্য সাফল্য পায় খুব কম ছাত্রই। এবং সেটা তারা পায় তাদের মেধা, পার্ফরমেন্স, ট্রেইনিং এবং কনটেস্ট করে যেসব অ্যালগরিদম আর টেকনিক শিখেছে, সেসবের জোরেই। হয়তো এই অল্প কিছু মানুষের সাফল্য পাওয়াটা আপনাদের অনেকের কাছেই প্রোগ্রামিং কনটেস্টের মূল্যকে কমিয়ে দিচ্ছে। কিন্তু ভুলে যাবেন না, প্রোগ্রামিং কনটেস্টের সাথে অনেকেই জড়িত, ছাত্রদের একটা বিশাল বাহিনী আছে, যাদের কাছে প্রোগ্রামিং কনটেস্ট ইন্টারেস্টিং, কাজের এবং প্রয়োজনীয়। আপনাদের যদি মনে হয় প্রোগ্রামিং কন্টেস্ট কোন কাজের কিছু না – বেশ তো, সেক্ষেত্রে আপনারা প্রোগ্রামিং কন্টেস্ট নাই করলেন! :) অনুগ্রহ করে বাকি সবার হয়ে সিদ্ধান্ত নেবেন না।

সবশেষে বলবো, সারাতভ স্টেট ইউনিভার্সিটির অলিম্পিয়াড ট্রেইনিং প্রোগ্রামে অনেক ছাত্রকে আমরা ট্রেইন করি। এদের মধ্যে অনেকেই কোনভাবেই আমাদের বিশ্ববিদ্যালয়ের সাথে জড়িত না। আমরা লেকচার দিই, কনটেস্ট নিই। আমরা প্রতিনিয়ত ছাত্রদের সময়ের সাথে সাথে আরো ভালো হয়ে উঠতে দেখি। আমাদের খুব বিরক্ত লাগে যখন কাউকে কাউকে বলতে শুনি যে, প্রোগ্রামিং কনটেস্টের কোন দরকার নেই। হ্যাঁ, আমাদের ছাত্রদের একটা বড় অংশ কোন পুরষ্কার পাবে না, কিন্তু সবাই একটা স্কিলসেট নিয়ে এখান থেকে বের হবে। এবং আমাদের এই প্রোগ্রামে তারা এটা পেয়ে যাচ্ছে বিনামূল্যে, খুব ইন্টারেস্টিং উপায়ে। এই যে 'অপ্রয়োজনীয়' শিক্ষাটা, এটা ছাত্রদের চাকরি পেতে এবং চাকরীক্ষেত্রে অন্যদের তুলনায় এগিয়ে রাখছে। ছাত্রদের শেখাবার এর চাইতে ভালো উপায় আর কীইবা হতে পারে?

মিহাই পাত্রাস্কুর চিঠি

ইকরাম মাহমুদ | জুন ২৫, ২০১২

মিহাই পাত্রাস্কু একজন রোমানিয়ান কম্পিউটার বিজ্ঞানী। আন্তর্জাতিক ইনফরমেটিক্স অলিম্পিয়াডে (IOI) ১৯৯৯ সালে রৌপ্য, ২০০০ ও ২০০১ সালে স্বর্ণপদক জয় করেন তিনি। ২০০৮ এ মাত্র দুই বছরে MIT থেকে PhD ডিগ্রী শেষ করেন মিহাই। তিনি ২০১০ থেকে আন্তর্জাতিক ইনফরমেটিক্স অলিম্পিয়াড এর সায়েন্টিফিক কমিটির সদস্য ছিলেন। মিহাই পাত্রাস্কু ডাটা স্ট্রাকচার নিয়ে তার মৌলিক গবেষণার জন্য ২০১২তে Presburger Award পান। ২০১২ সালের ৫ই জুন মাত্র ২৯ বছর বয়সে মিহাই পাত্রাস্কু মারা যান ব্রেইন ক্যান্সারে। মিহাই পাত্রাস্কু ভীষণ স্বহৃদয় এবং ইন্সপায়ারিং একজন মানুষ ছিলেন। ২০০৫ এ আতামুরাদ হেযরেতগুলিয়েভকে লেখা তার একটি চিঠি আমরা আতামুরাদের অনুমতিক্রমে প্রকাশ করছি।

মিহাই পাত্রাস্কু। ফটো কার্টেসি - MIT।

"Despite his very young age, Mihai Patrascu's work has broken through many old barriers on fundamental data structure problems, not only revitalizing but also revolutionizing a field that was almost silent for over a decade." -- প্রেসবার্গার অ্যাওয়ার্ড কমিটি, ২০১২

আতামুরাদের চিঠি

হ্যালো মিহাই,

আমার ইংরেজি খুব একটা ভালো না। প্লিজ কিছু মনে কোর না।

আমি আতামুরাদ হেযরেতগুলিয়েভ। আমার বয়স ১৬ আর আমি তুর্কিমিনিস্তানে থাকি। আমি ইনফরম্যাটিক্সে আগ্রহী এবং আমি এখন IOI এর জন্য প্রস্তুতি নিচ্ছি। শেষবারের IOI তে আমি ৬০০ তে ২৫৫ স্কোর করেছিলাম। কোন মেডেল পাইনি কারণ ২৬৫ তে ব্রোন্জের কাটআউট ছিলো। আমি ২০০৫ এবং ২০০৬ এ IOI তে অংশ নেবো এবং আমি স্বর্ণপদক জিততে চাই ।

আমি একটা কবিতা পড়েছিলাম - To follow the path: look to the master, follow the master, walk with the master, see through the master, become the master.

আমি সত্যি কাওকে খুঁজে বেড়াচ্ছি কিন্তু দু:খজনকভাবে আমার দেশ IOI তে মাত্র চারবছর ধরে অংশ নিচ্ছে আর কেউই কখনো কোন মেডেল জিততে পারেনি। সেজন্য আমার একটা "মাস্টার" দরকার দেশের বাইরে থেকে। তো সেইজন্য আমি গুগল করলাম আর তোমাকে খুঁজে পেলাম এবং আমার তোমার সাহায্য দরকার।

বন্ধু, আমি জানি কাওকে অ্যালগরিদম বা প্রোগ্রামিং শেখানো সহজ নয়। এবং আমি তোমাকে সেটা করতে বলবোও না। কিন্তু আমি তোমার সাহায্য চাই আমাকে পথ দেখানোর জন্য। প্লিজ আমাকে সাহায্য করো আমার প্রশ্নগুলোর উত্তর দিয়ে। কিভাবে প্রস্তুতি নিবো? কি পড়বো? কি সলভ করবো? তুমি সেটা কিভাবে করেছিলে? তুমি কিভাবে প্রস্তুতি নিয়েছিলে?

আমি কিছু অ্যালগরিদম আর ডাটা স্ট্রাকচার শিখেছি কিন্তু খুব একটা প্রবলেম সলভ করতে পারি না। যেমন ধরো, USACO এর গোল্ড লেভেলের প্রবলেম। এবং আমি জানি না কি পড়বো আমি। বা কি শিখবো। আমি প্রায় তিন বছর ধরে IOI এর জন্য ট্রেনিং করে যাচ্ছি।

আশা করি তুমি আমার সমস্যা বুঝতে পারছো। আমি তোমার উত্তরের জন্য অপেক্ষা করবো।

আতামুরাদ

মিহাই পাত্রাস্কুর উত্তর

হ্যালো!

অনেক ধন্যবাদ তোমার চিঠির জন্য :-)

আমি আসলে বহুদিন অলিম্পিয়াড থেকে দূরে আছি; আমি এখন রিসার্চ করছি থিওরি নিয়ে, আর সেটা আমাকে অনেক দূরে নিয়ে গেছে আমার IOI করার দিনগুলি থেকে। তবুও আমি তোমাকে কিছু উপদেশ দেবার চেষ্টা করবো আমার স্মৃতি থেকে, যা কিছু মনে পড়ে। কিন্তু মনে রেখো আমি যেহেতু বহুদিন ধরে অলিম্পিয়াডের সংস্পর্শে নেই হয়তো এই উপদেশগুলো সর্বোত্তম হবে না।

প্রথমত, ট্রেইন করার জন্য প্রবলেম সলভ করার চেয়ে ভালো কোন পথ নেই। আর প্রবলেমের জন্য সেরা উৎস হচ্ছে আগের IOI এর প্রবলেমগুলো অথবা অন্য প্রতিযোগিতাগুলোর প্রবলেমগুলো। তুমি CEOI (Central European OI), BOI (Balkan), BOI (Baltic) এর প্রবলেমগুলো দেখতে পারো। এই সব কন্টেস্টে অনেক ভালো প্রবলেম আছে, বিশেষ করে শেষ কিছু বছরের কন্টেস্টে।

আরেকটা চমৎকার জায়গা হচ্ছে USACO। ওদের বিশাল কালেকশন আছে ভালো প্রবলেমের। আমার যতদূর মনে পড়ে ওদের একটা ট্রেইনিং সিস্টেম ও আছে, যেখানে তুমি পর্যায়ক্রমে যতগুলো লেভেল পার করবে প্রবলেমগুলো তত কঠিন হতে থাকবে। এবং একইসাথে, তুমি USACO এর কন্টেস্টগুলোতেও অংশ নিতে পারো এবং নতুন প্রবলেমগুলো সলভ করতে পারো।

শুরুতে, তোমার কাছে ভীষণ কঠিন লাগতে পারে প্রবলেমগুলো সলভ করতে। কিন্তু আশা ছেড়ো না, অন্য কাওকে জিজ্ঞেস করো তোমাকে কিছু আইডিয়া দিতে। অথবা অন্য প্রবলেম ঘাটতে পারো, হয়তো খুব কাছাকাছি কোন একইরকম আইডিয়া তোমাকে এই প্রবলেমটা সলভ করতে সাহায্য করতে পারে। কিছু কিছু কন্টেস্টের অ্যানালাইসিসও থাকে যেখানে সবগুলো প্রবলেমের সমাধান আলোচনা করা হয়। সেগুলোও পড়তে পারো।

কিছু মেইলিং লিস্ট আর ফোরাম আছে যেগুলোতে মানুষজন আগ্রহী ইনফরম্যাটিক্স অলিম্পিয়াড নিয়ে। সেগুলো খুঁজে বের করতে পারো। তুমি সেখানে প্রশ্ন করতে পারো আর হয়তো তুমি জবাবও পাবে। অন্যদের করা প্রশ্নগুলোও তোমার ভাবনার খোরাক হতে পারে।

প্রবলেম সলভিং ছাড়া আরেকটা কাজ তুমি করতে পারো। সেটা হচ্ছে বই পড়া। দূর্ভাগ্যজনকভাবে আমি কোন ভালো বই এর নাম জানি না। Cormen, Leiserson এবং Rivest এর বইটা বেশ বিখ্যাত। কিন্তু ব্যাখ্যাগুলো খুব একটা ভালো না, আর তারা খুব বেশি ডিটেইলসের ভিতর ঢুকে যায়। কিন্তু তারপরও সবকিছু মিলিয়ে আমার মনে হয় এটা বেশ কাজের হবে অন্তত জানাটা যে বইটার ভেতর কি আছে।

আবার এটাও একটা প্রশ্ন যে তুমি কি বেশি করে কোডিং করবে নাকি বেশি করে থিওরি পড়বে। আমার মতামত হচ্ছে তোমার উচিত একটা সময় পর্যন্ত ধুমায়ে কোড করা, যতক্ষণ না পর্যন্ত তুমি খুব কম্ফোর্টেবল হচ্ছো কোডিং নিয়ে। এর মানে হচ্ছে, তুমি লিখে ফেলতে পারো যেটা তুমি লিখতে চাও এবং তুমি সেটা লিখতে পারো নির্ভূলভাবে। তারপর তোমার উচিত প্রচুর প্রবলেম সলভ করা থিওরিটিকালি এবং মাঝে মাঝে সেগুলো কোডে ইম্প্লিমেন্ট করে দেখা শুধু অভ্যাসটা রাখার জন্য। আমার মনে হয় আমি শেষ বছরগুলোতে বড়জোর একটা প্রবলেম ইম্প্লিমেন্ট করেছি প্রতি মাসে। যখন তুমি ভীষণ কম্ফোর্টেবল হয়ে যাবে প্রোগ্রামিং এ, তখন কোন প্রয়োজন নেই সেখানে সময় নষ্ট করার। সময়গুলো নষ্ট করো বরং অন্য জায়গাগুলোতে যেখানে তুমি দূর্বল।

আর সবকিছুর উপরে, আমার উপদেশ হচ্ছে রিল্যাক্স করো আর নিজের উপর ভরসা রাখো। অবশ্যই তোমার ভালো হতে হবে, কিন্তু একই সাথে তোমার ভেতর শক্ত বিশ্বাসও থাকতে হবে যে তুমি আসলেই ভালো। আর তুমি যদি খুব বেশি মাত্রাতিরিক্ত পরিশ্রম করতে থাকো, সেটা তোমার সৃষ্টিশীলতাকে ধ্বংস করে ফেলবে। There's really a very fine point where you balance work and ingenuity, and you need to be relaxed and confident in order to find that point.

শুভ কামনা।

মিহাই পাত্রাস্কু

উল্লেক্ষ্য, আতামুরাদ হেযরেতগুলিয়েভ ২০০৬ সালে ইনফরমেটিক্স অলিম্পিয়াডে স্বর্ণপদক জেতে।

What should be included in a detailed 4 year plan for a CS 1st year student in Bangladesh who wants to qualify for the ACM ICPC World Finals at the end of his/her 3rd or 4th year?

Answered by Nafis Sadique, Jahangirnagar University, ACM ICPC World Finalist - 2015 & 2016.

I think i may answer this question although i started programming when i was in college. I think you know how much hard work is needed if you want to qualify for world finals. There are many strong teams from different universities. And they all practice very hard for this. Qualifying for world finals would be a dream come true for each contestants.

So lets make a plan for qualifying to world finals. In the very first year of study one should learn the language in the first 2-3 months. You need to chose a language between C++ or Java. I will recommend C++ as it is better suited for programming contest. And make sure that your language skill is good by this time(i.e. you can code whatever you are thinking). After that you are gonna need to finish the following works by the end of first year.

1. Open account in UVa, Codeforces, LightOJ, Topcoder, SPOJ and USACO.
2. Solve 200+ ad-hock problems in UVa.
3. Learn basic DP and solve some classical problems.
4. Learn to use stack, queue, priority queue, lists and solve problems with them.
5. Learn basic graph algorithms like, DFS, BFS, Dijkstra, Floyd-Warshall, MST.
6. Solve some problems that require greedy solution.
7. Learn basic number theories and geometry.
8. Solve 500+ problems overall.
9. Participate in national contests, codeforces and topcoder contests.

So that pretty much finishes first year. There are also several topics i didn't mention(like sorting/searching), these are too basic topic and should be covered when solving ad-hock problem. After this your rating should be blue in codeforces and green in topcoder(make this the achievement).

So for second year you are going to learn some advanced algorithms. We can make another list for it.

a. Advanced data structures, segment tree and variations(solve at-least 50 problems on it), HLD(solve the QTREE's and you are good to go).
b. Solve some advanced DP problems. LightOJ has some great problems and they should be solved as much as possible,
c. Solve some hard problems on graph.
d. Learn game theory, combinatorics, probability and pretty much cover every topics that is written on LightOJ problem categories.
e. Solve another 500+ problems in this year. Complete USACO training system. Do not waste your time solving ad-hocks anymore. The problems should at-least be div2 C(on codeofrces scale) difficulty. And don't waste too much time solving problems that is way out of hand. But thinking about hard problems is actually great.
f. Learn some advanced number theory and geometry topics and solve a lot of problem on these.
g. Try to attend every contest possible.
h. Get better at your coding skills.

After this year your rating would be blue in topcoder and purple in codeforces. In national contests you will get around 15-20 place.

For third year there is only one practice plan. Solve 1000+ problems. They all should div2 D difficulty. By this time you will learn to solve d1 D. Your rating should be orange in CF and yellow on TC. But the most important thing is upsolving(solve the problems after the contest excluding the stopper one). Solve past dhaka regional and different onsite contests in Bangladesh. UVa has got a lot of it. You just need to search for them. Also take part in USACO monthly contests. You will probably end up finishing top 10 in national contest. If you are lucky you may even qualify for WF.

Dedicate the 4th year for more programming. I can't say much because i have just started 4th year. So stay tuned.

How did Anudeep Nekkanti become so good at competitive programming?

Anudeep Nekkanti
Software Engineer at Google, Zurich

I came to know about online judge for the first time in 2012 Jan. That was because of IOPC (programming contest by IIT Kanpur). It was held on CodeChef then. I tried for about 10 hours but could not solve a single problem.

2012 March, I started to do CodeChef long contest. I spent 7 days to solve 3 simple problems. March 7th 2012 was the day when I decided not to take any contest till I am confident about making into top 50.

I only knew SPOJ and Codechef . Then, I started solving problems on SPOJ sorted easy to hard. I solved about 300 problems by July 2012. Practice was the only thing I did. I used to try a problem for 2-3 hours. If I don't get it, I tried searching for solutions on forums. I read few tutorials on TopCoder, but I did not know that TopCoder also has algorithm problems.

I participated in following August's long contest, I was lot better this time, I could solve 7 problems. Ended 35th in Global ranking. From then I became lazy. I did not do any practice. I only did long contest every month. So, August to November, I did nothing.

With this limited exposure to programming I went to participate in ACM ICPC Regionals, Amrita. I could solve 4 problems there at onsite. I then understood that knowing how to solve is not enough, it is the ability to think and code fast that is more important.

Jan 2013, I started solving TopCoder problems. I did like about 300 Div 1 250 Pointers. 500 pointers were Aliens. Dynamic Programming was my enemy. But solving 250 pointers fast was enough to become yellow on TopCoder. I also came to know about CodeForces in Jan. I even started doing some contests there. Following Feb to July, I did very little programming. I did not see much improvement in myself.
I read a line on quora : "You solve about 20 Div1 500 problems in one week on TopCoder, and they will seem easy then" - Pradeep George Mathias
That line changed me a lot. I decided to give a try. Thanks to PGM.

August 2013, I decided to overcome fear of DP and div1 500 problems. I started to do Div 1 500 pointers, I did them very seriously, I solved like about 50 problems in about 12 days, believe me I could not do a single problem on my own. At times I was frustrated, disappointed, and hopeless. But no matter how down I felt, I decided to carry on. By August end I solved about 180 *500 pointers*. I slowly started to think Dynamically :) By then I was able to solve 4 out of 5 problems. DP became my friend, a very close friend. Now I am quite comfortable with 500 pointers. But still keep screwing them in the contests.

I did not have any Math or programming background, and this made it difficult for me. So, to conclude: all that matters is sheer practice.

Programming is fun, programming is easy. My failure at IOPC 2012 made me start it. I thought, I will do well in IOPC 2013 and stop programming. That is how I started it. Very soon I started to like it, then I got addicted to it. I enjoy the feel that I get when I see "Accepted". That awesome green color. My heart beat raises when ever I submit a solution. I get goosebumps. It was that fun that kept me going.

Don't do it, Play it. Enjoy it, It is a fun game.
After 21 months, I am still deeply in love with it :)

সমস্যা সমাধানে মনস্তত্ত্ব

- Taman Islam, Facebook London

শেয়ার বাজার হোক কিংবা প্রোগ্রামিং কনটেস্ট, ঈদের বাজার হোক বা বইমেলায় যাওয়া - প্রতি ধাপে আমরা "সমস্যা"র সমাধান করে থাকি। কোন বাসে গেলে বেশি ভালো হবে, কোন রংটায় আমাকে বেশি মানাবে, স্টকটা বিক্রি করে দেবো না কোনো নির্দিষ্ট মুহূর্তের জন্য অপেক্ষা করবো - এ সবই সমস্যা। এ ধরণের সমস্যা সমাধানের জন্য আমাদের নানাভাবে চিন্তা করতে হয়। কোনো কোনো সমস্যা আমরা শোনামাত্রই সমাধান করে দিতে পারি, যেমন, রাতে অন্ধকারে ঘরে বই পড়তে পারি না। সমস্যা শোনামাত্রই আমাদের মাথায় সমাধান চলে আসে, "বাতি জ্বালিয়ে চেষ্টা করেছো?" এতো দ্রুত এ সমস্যার সমাধান আমরা দিতে পারি কারণ এ ধরণের সমস্যার সাথে আমরা পরিচিত। যে ধরণের সমস্যা আমরা বারবার সমাধান করে অভ্যস্ত, আমাদের মস্তিষ্কে সে ধরণের সমস্যাদির সমাধান "অভ্যাস" হিসাবে জমা থাকে।

যেমন? অনেকে ব্যাপারটা ছায়াছবিতে দেখেছেন কিন্তু বাস্তব জীবনে হয়তো দেখেন নি। নায়িকা মাথায় আঘাত প্রাপ্ত হয়ে স্মৃতিভ্রষ্ট। সে নায়ককে চেনে না, বাবাকে চেনে না, সে কোথায় আছে তাও জানে না। কিন্তু সে বাংলায় (বা বলিউডে হলে হিন্দীতে) কথা বলে। সে তার ভাষা ভুলে যায় নি। কারণ ভাষাটা তার অভ্যাসের মতো। অনেকের জন্য বেশি দুঃখের ব্যাপার হলো নায়কেরা মাথায় আঘাতপ্রাপ্ত হয়ে স্মৃতিভ্রষ্ট হলে সাধারণত স্যান্ডো গেঞ্জি পড়ে বা খালি গায়ে থাকে কিন্তু নায়িকারা এটাও ভুলে না যে তাকে পোষাক পড়তে হবে। ঘটনাটি কেবল চলচ্চিত্র নয় বাস্তবেও সত্য।

সুতরাং সমস্যা সমাধানে ভালো হওয়ার সবচেয়ে ভালো উপায় কি?
জ্বী, অনুশীলন। যে বা যারা যে বিষয়ে ভালো, দেখা যায় তারা তাদের ঐ ব্যাপারে অনুশীলন করতে করতেই ভালো হয়েছে। "রোনালদো এতো ভালো খেলে তাও কত অনুশীলন করে!" এটা সত্য নাকি "রোনালদো এতো অনুশীলন করে বলেই এতো ভালো খেলে" এটা সত্য? দ্বিতীয়টি সত্য। অনেক অনেক মেধাবী খেলোয়াড় এবং কুশলী যখনই চিন্তা করেছে তাদের অনুশীলন লাগবে না - হারিয়ে গেছে। যারা দীর্ঘদিন ধরে অনুশীলন ধরে রাখতে পেরেছে কেবল তারাই টিকে আছে। সুতরাং অনুশীলনের কোনো বিকল্প নেই। প্রকৃতিতে টীকে থাকতে হলে যে অনুশীলনগুলো আমাদের প্রয়োজন, প্রাকৃতিকভাবে ওগুলো আমরা নিজেরা অজান্তেই করে থাকি। যেমন, শিশুদের হাঁটার অনুশীলন, শিশুদের হামাগুড়ি দিয়ে এগোতে এগোতে বসে চারদিকে তাকিয়ে দেখার অনুশীলন, জোরে কোথাও শব্দ হলে দৌড় দেয়ার অনুশীলন।

সুতরাং অনুশীলন করলেই কি সফল হওয়া সম্ভব? উঁহুম, প্রয়োজন "সঠিক" অনুশীলন। ভ্রান্ত অনুশীলনের ফল দিন দিন আরো খারাপ হয়ে যাওয়ার কথা। যেমন আপনি অনুশীলন করলেন প্রতিদিন দু'টা করে বার্গার খাবেন। এই অনুশীলন আপনি যত করবেন ফল ততই খারাপ হতে থাকবে (যদি এমন হয় আপনি এখন ৩টি করে খান, সংখ্যাটা কমাবেন তবে ফল ভালো হবে। কিন্তু আপনার কি আসলেই মাথা খারাপ?) আবার কোনো এক ব্যাটসম্যান অনুশীলন শুরু করল সে বাউন্সার বল দেখলে নুয়ে যাবে। এই অনুশীলনের ফলে তার ব্যাটিং দিন দিন

খারাপই হতে থাকবে। প্রোগ্রামিং প্রতিযোগীতার অনুশীলনেও এমন খারাপ জিনিস হতে পারে। আমি একজন খুব ভালো প্রতিযোগীকে খুব ভালভাবে চিনি যিনি জ্যামিতি ভয় পান। এই অনুশীলনে তার ক্ষতি কি ভয়াবহ হয়েছে তা বোঝানো তাকে না চিনলে সম্ভব না। এমনও হয়েছে জ্যামিতিক চিত্র সহ গ্রাফ সমস্যা, যেই সমস্যা তিনি খুব ভালভাবেই পারবেন কিন্তু তিনি জ্যামিতিক চিত্র দেখে ভয়ে সমস্যাই পড়েন নি। এটি একটি ভুল অনুশীলনের বাজে ফল। তিনি এতো শতো কঠিন সমস্যার সমাধান করেছেন যে জ্যামিতি শুরু করলে অবশ্যই তিনি ভালো পারতেন। তার ভুল কি? জ্যামিতি দেখে ভয় পাওয়ার অনুশীলন তাকে জ্যামিতির ব্যাপারেই বীতশ্রদ্ধ-ভীত করে ফেলেছে।

প্রোগ্রামিং অনুশীলনের ব্যাপারে এখন অনেক ভালো ভালো ওয়েবসাইট আছে। আমি চিন্তা করছিলাম আমার অল্প কিছু অভিজ্ঞতা এবং জ্ঞান থেকে মনস্তাত্ত্বিক কিছু দোষ গুণ নিয়ে আলোচনা করা যায় কিনা। নিচে আমি কয়েকটি চিন্তার দুর্বলতা বলার চেষ্টা করবো।

১। মেঘে মেঘে রবীন্দ্রনাথঃ
রৌদ্রজ্জ্বল বিকেলে আকাশের দিকে তাকিয়ে রবীন্দ্রনাথের চেহারার মেঘ দেখেছেন? আমরা প্রায় সবাই দেখেছি। ১৯৯৪ সালে ডায়ানা ডাইসের রুটিতে কামড় দেয়ার পর দেখতে পেলেন তার কামড়ানো রুটিতে মেরির (মরিয়ম আঃ) চেহারা ভেসে উঠেছে। আমরা মেঘে রবীন্দ্রনাথকে দেখে তেমন ফায়দা না পেলেও ডায়ানার রুটি ২০০৪ সালে ২৮ হাজার ডলারে নিলাম বিক্রি হয়েছে!
বাংগালী অখ্রিস্টানরা কখনো মেঘে মেরিকে দেখে না কেন? বিদেশীরা সাধারণতঃ রুটিতে রবীন্দ্রনাথকে দেখে না কেন?
কারণ অখ্রিস্ট বাংগালীরা সাধারণত মেরির চেহারা চিনে না, বিদেশীরা রবীন্দ্রনাথকে তেমন চেনে না। চিনলেও আমাদের মতো হৃদয়ে ধারণ করে না যে মাইলসের "মেঘে, রুটিতে - তুমি শুধু তুমি" ধরণের গান গাইবে। সুতরাং?

আপনার মনে যা আছে, আপনি কেবল তাই দেখতে পাবেন। সহজ সমস্যার বেলায় বড় দলগুলোও পারে, ছোটোরাও পারে। কিন্তু তারপরও সাধারণত(সব সময় না কিন্তু বেশিরভাগ সময়) বড়রা আগে পারে কেন?
আমরা যখন সমস্যার সমাধানে মন দেই তখন শুরু থেকেই আমরা মনে মনে একটি ধারণা নিয়ে আগাই। কোনো সমস্যার শুরু হয়েছে এভাবে, "রহিম বাদশা ঢাকায় থাকে। ঢাকায় ক টি বাড়ি এবং খ টি রাস্তা আছে" এই দুই লাইন পড়ে অভিজ্ঞ মন কি চিন্তা করবে? সমস্যাটি খুব সম্ভবতঃ গ্রাফ সংক্রান্ত। "প্রতি দু'টি বাড়ি কেবল মাত্র একটি পথ দিয়ে সংযুক্ত"। মন বলবে সমস্যাটি খুব সম্ভবতঃ ট্রি সংক্রান্ত। এই পর্যায়ে যে যত বেশি সমস্যার সমাধান করেছে তার কাছে তত বেশি সুযোগ সৃষ্টি হবে। এবং ছাঁকতে ছাঁকতে অনেক সময় বড় প্রতিযোগীরা তাদের ধারণা নিয়ে উদাহরণ ইনপুট/আউটপুটে চলে যাবে এবং সেখান থেকে নিশ্চিত হবে যে তার ধারণা সঠিক/ভুল। যদি ধারণা সঠিক হয় সে সরাসরি প্রোগ্রাম লেখা শুরু করবে, ধারণা ভুল হলে আবার মাঝ থেকে সমস্যা পড়া শুরু করবে।
তাহলে সব সময় কেনো বড়রা তাড়াতাড়ি পারে না? -প্রায়শই সমস্যা এমনভাবে তৈরি হয়ে থাকে যাতে বিভ্রান্তির সৃষ্টি হয়। যারা হয়তো কেবল বিএফএস পারে তারা একটি ধারণা নিয়েই এগোতে থাকে এবং সমস্যা যদি আসলেও

বিএফএসের হয় তাহলে তারা সেটা সহজে পেরে ফেলে। কিন্তু যে বিএফএস / ডায়াক্সট্রা দু'টা পারে, তার ছাঁকতে কিছু বেশি সময় লাগতে পারে।

ভ্রান্তিঃ প্রশ্ন দেখেই ধারণা করা ঠিক না যে আপনি ফর লুপ পারেন বলে এটি ফর লুপের প্রশ্ন। আমরা যদিও সবাই এই ধরণের ভুল করে থাকি।

২। আমার কোড সব ইনপুটে কাজ করে, তবু কেনো "WA"?
সবচেয়ে সাধারণ প্রশ্ন। ফেসবুকে, ফোরামে সবচেয়ে বেশি সহজলভ্য প্রশ্ন এটি। আমি নিজেও খুবই ক্ষিপ্ত হতাম ছোটো বেলায়। অবশ্য দু'একটা সময়ে দেখে গিয়েছে যে আসলেও জাজদের ডাটায় ভুল বা এমন কিছু। আমার নিজের প্রশ্নেও একবার ডাটায় ভুল ছিল (২০১০ সালে, হ্যাংঝাউ দিয়ানজি বিশ্ববিদ্যালয়ের অনলাইন কনটেস্টে)। কিন্তু এ ধরণের ঘটনা তেমন সহজলভ্য না যেমন উপরোক্ত প্রশ্নটি।

এ ক্ষেত্রে আমাদের সমস্যা হচ্ছে "নিশ্চয়তা পক্ষপাত"। এটি স্বাভাবিক মানবিক ত্রুটি। আমি এ নিয়ে লিখতে লিখতেও হয়তো এ নিয়ে ভুল করছি, কে জানে! এ ত্রুটির বৈশিষ্ট্য হচ্ছে, নতুন যে জ্ঞান পাবেন তার মধ্যে কেবল যেটুকু আপনার পুরোনো জ্ঞানের সাথে সামঞ্জস্যপূর্ণ তাকে রাখবেন, বাকিগুলো ফেলে দিবেন। যেমন, আপনি ফুটবল জানেন এবং আপনাকে যদি কেউ বলে "ব্রাজিল পাকিস্তানকে ২০-০ গোলে হারিয়েছে" আপনি সেটি বিশ্বাস করবেন, কারণ এটি আপনার পূর্বতন জ্ঞানের সাথে সামঞ্জস্যপূর্ণ। কিন্তু আপনাকে যদি বলা হয়, "ভারত আর্জেন্টিনাকে ১-০ গোলে হারিয়েছে" - আপনি ভাববেন গুল মারছে, কারণ আপনার জ্ঞান একে সমর্থন করে না। প্রথমটি গাণিতিক এবং যৌক্তিকতার দিক থেকে বেশি অযৌক্তিক কিন্তু আমাদের জ্ঞানগত দিক থেকে সেটি স্বাভাবিক, দ্বিতীয়টি বেশি যৌক্তিক কিন্তু আমরা তা মানতে নারাজ। আবার একজন ব্রাজিল সমর্থক শোনামাত্রই দু'টি বাক্যই বিশ্বাস করবে কিন্তু আর্জেন্টিনা সমর্থক একটিও শোনামাত্রই বিশ্বাস করবে না। শুনতে অদ্ভুত শোনালেও আমরা প্রতিনিয়ত এ ভুলটা করে থাকি।

প্রোগ্রামিংয়ের ক্ষেত্রে যেটি হয়, আপনি একটি সমাধান দাঁড় করানোর পর, যতভাবেই চিন্তা করবেন আপনার কাছে মনে হতে থাকবে এ প্রোগ্রাম শতভাগ সঠিক। এই প্রোগ্রাম মিথ্যা হতে পারে না। যদি আপনি বুঝতেই পারতেন এ প্রোগ্রামে ভুল হতে পারে, তবে আপনি ফোরামে পোস্ট দিতেন না, নিজেই ভুল ধরে ফেলতে পারতেন। আমরা বেশিরভাগ ক্ষেত্রে জ্ঞাত-অজ্ঞাতসারে এমন সব ডাটা দিয়েই নিরীক্ষা করি যেগুলো আপনার প্রোগ্রামের সাথে মানানসই। অনেক সময় এটিও হয়ে থাকে যে আমরা সমস্যাই ভুল বুঝেছি। সে ক্ষেত্রে আমাদের ইনপুট গুলো আরো বেশি পক্ষপাতমূলক হয়ে থাকে।

এড়ানোর উপায়ঃ বেশি "WA" পেলে একটি নিরীক্ষক প্রোগ্রাম লেখা যেটি এলোমেলো ডাটা দিয়ে নিরীক্ষা চালাবে এবং একটি কম দক্ষ কিন্তু সঠিক সমাধান প্রোগ্রাম লেখা। তারপর এ দু'টোর আউটপুট তুলনা করা। তারপর ঐ কম দক্ষ প্রোগ্রামের আউটপুটের সাথে নিজের প্রোগ্রামের আউটপুট তুলনা করা।

৩। আমি মনে হয় একটি ধারা পেয়েছি!
অনেক সমস্যা সমাধানের ক্ষেত্রেই একটি ধারার বেশ প্রয়োজনীয়তা হয়ে থাকে। যেসকল ক্ষেত্রে ইনপুটের একটি ধারাবাহিকতা থাকে, সেসকল ক্ষেত্রে আউটপুটেরও একটি ধারাবাহিকতা থাকতেই পারে। যেমন ধরুন একটি সমস্যা দেয়া আছে, যেখানে ইনপুট কেবল একটি সংখ্যা ক এবং আপনাকে আউটপুট দিতে হবে। আরো ধরি, ১ < ক < ৫০০০০০।
এক্ষেত্রে কি করা যেতে পারে? শুরুতেই ক এর মান ১ - ১০ এর জন্য চালিয়ে নিয়ে দেখা যেতে পারে তাদের মধ্যে কোনো সামঞ্জস্য আছে কিনা। ধরুন ক এর মান ১ - ৫ এর ফল এলো নিম্নরূপঃ
১, ২, ৩ , ৫, ৮...
ইয়েই! ফিবোনাচি সিরিজ! সুতরাং প্রোগ্রাম জমা দিয়ে দিবো? উঁহুম! একটু দাঁড়াই।

Dispersion of (floor(n*e)), by antidiagonals সিরিজের প্রথম পাঁচটি সংখ্যাও একই! এবং সত্যি বলতে এমন আরো না হলেও ১০০টা ধারা আছে যাদের মধ্যে ১, ২, ৩, ৫, ৮ উপধারাটি আছে। সুতরাং ধারা পেয়েই আত্মহারা এবং তারপরে "ভুল উত্তর" রায় পেয়ে হতাশ হওয়ার কিছু নেই। ৫টি সংখ্যা দিয়ে ধারা হিসাব করে "ভুল উত্তর" পেলে, ১০টি সংখ্যার জন্য হিসাব করে দেখা যায়। তারপর হয়তো ধারণা পরিবর্তন হবে। যদি দেখা যায় এমন ১ - ১০০০ এর জন্য ধারণা ঠিক আছে, তাহলে এর পরে এলোমেলো অনেক বড় সংখ্যার জন্য নিজের ধারণাকে পরীক্ষা করা যেতে পারে। পরীক্ষা-নীরিক্ষার পরিমাণ বাড়াতে হবে, ধ্বসে যাওয়া ঠিক হবে না।

৪। কপালকুন্ডলেঃ
কপালকুন্ডলের সাথে সম্পর্ক নেই, কপালকুন্ডলের রচিয়তার একটি বাণী আছে। আমার বাণীটি হুবহু মনে নেই, তবে বংকিমচন্দ্র মনে হয় বলেছিলেন, কোনো কিছু রচনার পরে ১ বছর/ছয় মাস(আমার ঠিক সময় মনে নেই) ফেলে রাখো। তারপর আবার পড়। তখন যদি মনে হয় এটি প্রকাশনা যোগ্য তবে প্রকাশকের কাছে পাঠাও। রাতে ঘুমোতে যাওয়ার সময় গুণগুণ করে গান গাইতে গাইতে আমাদের প্রায়শই মনে হয় পৃথিবীর সেরা একটি গান আমি রচনা করে ফেলছি। সকালে ওঠার পরে দেখা যায় মনে থাকে না, থাকলেও অবাক লাগে, এমন ফালতু জিনিস আমি কি করে গাইলাম!

প্রোগ্রাম লিখেই জমা দেয়ার দরকার নেই। খুব বেশি সহজ হলে করা যেতে পারে। কিন্তু প্রোগ্রাম লিখে ৫ মিনিট সময় নিয়ে একটি পাশবিক(!) প্রোগ্রাম লিখে নিজে নিজে বিচার করা আমার মতে উত্তমতর ধারণা। এখন নির্ভর করে প্রতিযোগীতাটা কতখানি গুরুত্বপূর্ণ তার ওপর। যদি একান্তই ফালতু প্রতিযোগীতা হয় তবে তা না করলেও চলে, তারপরো নিজে নিজে বিচার করা সময়, পেনাল্টি, হতাশা সব কিছু থেকেই বাঁচায়। দশ মিনিটের একটি বিচারক প্রোগ্রাম যদি একটি পেনাল্টি থেকেও বাঁচায় তবুও বিশ মিনিট কেবল পেনাল্টি থেকেই বাঁচবে। এবং WA এর পরের হতাশাজনিত কারণে ডিবাগ করার সময় সাধারণত আরো বেশি যায়। আর যদি টপকোডার বা কোডফোর্সেস ঘরানার প্রতিযোগীতা হয় তবে এই বিচারক প্রোগ্রাম থেকে ইনপুট/কেস তৈরি করা যায় যেগুলো দিয়ে অন্যদের প্রোগ্রাম চ্যালেঞ্জ করা যায়।

৫। সহজ প্রশ্ন পারি না সহজেঃ
দুর্বল/নতুন প্রতিযোগীরা প্রায়শঃই অনেক কঠিন সমস্যা নির্ভিক চিত্তে আক্রমণ করে থাকে। এই ধরণের সাহসীকতা ভালো, আসলে এই ধরণের সাহসীকতার জন্যই মানব সমাজ আজ এতো দূর আসতে পেরেছে, গবাদী পশুরা পারে নি। কিন্তু প্রতিযোগীতার সময়ে ব্যাপারটা সবসময়(পড়ুন প্রায় সবসময়েই) ভালো নাও হতে পারে।

কেনো আমরা এ ধরণের সমস্যা আক্রমণ করে থাকি?
ধরুণ, আপনি কেবল for লুপ পারেন এবং আপনার কাছে অ্যালগরিদম বই আছে। আপনাকে একটি সমস্যা দেয়া হল, "একটি গ্রাফের দু'টি নোডের মধ্যে সর্বহ্রস্ব পথ বের করো" অথবা "১০০০০০০০০০ তম ফিবোনাচি সংখ্যা % ১০১ বের করো"
আপনি কোনটি চেষ্টা করবেন?

আমি জানি না আপনি কোনটি চেষ্টা করবেন কিন্তু আমি বা বেশিরভাগই লোকই এ সময় দ্বিতীয় সমস্যার সমাধানে চেষ্টা করবো। কারণ দ্বিতীয় সমস্যাটি আমাদের কাছে তুলনামূলক চেনা জানা। for লুপ দিয়ে হওয়ার কথা! কিন্তু যে কোনো অভিজ্ঞ প্রতিযোগীই জানে, প্রথম সমস্যাটি দ্বিতীয়টির তুলনায় সহজতর। এমনকি দ্বিতীয়টি ৪ ঘন্টা ধরে জমার পর জমা না দিয়ে, দু' ঘন্টা সময় দিয়ে প্রথমটি শিখে ফেলা তুলনামূলক ভালো পদ্ধতি।
এ ভ্রান্তিটিও মানবিক।
সমাধানঃ অন্যদের দেখা। আমি প্রশ্নটিকে সহজ মনে করছি তার পেছনে আসলেও কি কোনো ভাল যুক্তি আছে? - এ প্রশ্নটি করা।

বাড়তি টুলঃ
যে কোনো ধরণের প্রোগ্রামিংয়ে আমাদের কিছু উপকরণ লাগে। যেমন, আইডিই, ডিবাগার, কম্পিউটার, কীবোর্ড। কিন্তু এ ধরণের উপকরণ কে কেমন কাজে লাগাতে পারে তার উপরও নির্ভর করে একটি দল কেমন করতে পারে।
এই উপকরণগুলোর মধ্যে সবচেয়ে গুরুত্বপূর্ন হচ্ছে র‍্যাংকলিস্ট। র‍্যাংকলিস্ট ঠিকমতো অনুসরণ করার উপরে অনেক সময়েই ফলাফল পর্যন্ত নির্ভর করে!

২০০৯ সালের ঢাকা আইসিপিসির বেশ শক্তিশালী একটি টীম C সমস্যাটি নিয়ে বেশ চাপের মুখে পড়ে গিয়েছিল। যখন তারা দেখতে পেল আরেকটি টীম C সমাধান করেছে তখন তারা চিন্তা করল, ঐ টীম যেহেতু C সমাধান করেছে, সুতরাং C এতো কঠিন হবে না। তখন তারা নতুন আঙ্গিকে সমস্যা পড়ল এবং দেখা গেলো তারা একটি ইনপুটের সীমা ভুল বুঝেছিল।
এমন ঘটনা কম বেশি সব দলেরই আছে। র‍্যাঙ্কলিস্ট আমরা অনুসরণ করতে পারি বলেই এসিএমের সমাধানকৃত প্রশ্নের সংখ্যা এতো বেশি হয়। আমি একটি প্রতিযোগীতা র‍্যাঙ্কলিস্ট ছাড়া করেছিলাম, ব্যক্তিগত অভিজ্ঞতা থেকেই বলতে পারি, সে দিনই মনে হয় সবচেয়ে ভালো বুঝেছিলাম র‍্যাংক্লিস্ট কি অমূল্য সম্পদ! চীনের কোনো এক দলের

একটি কৌশল ছিল তারা শুরুতেই একটি কঠিন সমস্যায় একটি ফালতু প্রোগ্রাম জমা দেবে। এর ফলে অন্যরা, বিশেষ করে তাদের প্রতিদ্বন্দ্বীরা ভাববে ঐ সমস্যাটা তারা চেষ্টা করছে সুতরাং পারা যেতে পারে। এর ফলে ঐ দলের সময় নষ্ট হবে। র‍্যাঙ্কলিস্টের এতোই মূল্য!
ধরুন আমি জানি যে এ কনটেস্টে টমেক অংশ নিচ্ছে। কনটেস্টের ৩ ঘন্টা পরেও কেউ, এমনকি টমেকও কোনো একটি প্রবলেম ক চেষ্টা করে নি। সুতরাং ঐ সমস্যা আমার চেষ্টা করা মোটামুটি নিশ্চিন্তভাবেই অর্থহীন। ৫ নং সমস্যাটা বিশেষ করে এই উপায়ে খুব সহজে এড়ানো যায়।

দলীয় বোঝাপড়াটা এসিএম ঘরানার প্রতিযোগীতায় অনেক গুরুত্বপূর্ণ। সাধারণত মানুষ ৩ জনের দলভিত্তিক প্রতিযোগীতায় নিজের সামর্থ্যের ৮৫% এর মতো দিতে পারে। সুতরাং এই অংশগ্রহণের হারটাকে আরো উপরে নেয়ার জন্য দরকার দলভিত্তিক অনুশীলন এবং বোঝাপড়া। দলের মধ্যে চ্যালেঞ্জ ধরণের কিছু খেলা যেতে পারে। একজন প্রোগ্রাম করার পর অন্যজনের হাতে ছেড়ে দেবে। যার কাজ প্রথমজনের লেখা প্রোগ্রামকে চ্যালেঞ্জ করা। তবে এ ধরণের চ্যালেঞ্জ করার আগে নিশ্চিত হওয়া দরকার, যে চ্যালেঞ্জ করছে তার সাথে যে প্রোগ্রাম লিখেছে সে আলোচনা করে নি। নয়তো বাড়তি কোনো ফল আসবে না। সেই সাথে চ্যালেঞ্জকে খেলাধুলার মত করে নিতে হবে। চ্যালেঞ্জকে ইজ্জতের প্রশ্ন করে নিলে, চ্যালেঞ্জ করতে না যাওয়াই ভালো! পরে দেখা যাবে প্রতিযোগীতা বাদ দিয়ে হাতাহাতি লেগে গেছে!

পরিশেষঃ একটু লক্ষ্য করলেই দেখবেন, সবগুলো সমস্যার গোঁড়া একই জায়গায়। "আত্মগরিমা" বা "আমি যা জানি তাই"। এই একটি জিনিসই মূলতঃ পৃথিবীর সব সমস্যার কারণ। আপনি ইসলাম মানেন কারণ আল্লাহ সত্য বলে নয় বরং আপনি মনে করেন আল্লাহ সত্য বলে। অমুকে খ্রিস্টান কারণ যীশু ঈশ্বরপুত্র বলে নয় বরং অমুক বিশ্বাস করে যীশু ঈশ্বরপুত্র তাই।
নিশ্চয়তা ব্যাপারটা জ্ঞানীরা কখনোই দেয় না। কেবলমাত্র যাদের জ্ঞানের চেয়ে জ্ঞানের ছদ্মাবরণ (illusion) বড় তারা দিয়ে থাকে।
জ্ঞানী হওয়া ভালো, মূর্খ হওয়া একটু খারাপ, জ্ঞানের বিভ্রম কদাকার। জ্ঞানের বিভ্রম থেকে জ্ঞান আমাদের রক্ষা করুক!

What is a typical day of a competitive programmer preparing for ACM or IOI?

Answered by Pushkar Mishra, IOI 2014; Student Coach of the Indian team to IOI 2015

Let me answer this question from two perspectives, first one as a contestant myself and second one as a coach. I shall relate how my days were when I was involved in International Olympiad in Informatics (IOI).

Contestant:

As stated in one of my answers, I started taking competitive programming seriously towards the end of May 2013. For the two months of summer break from school, I familiarised myself with data structures, algorithms, and general programming techniques, and took part in Codechef long contests.

It was all going very well during the summer holidays. But when they ended, that was when I actually felt the pressure of having too many things on my plate.

July '13: The minimum attendance policy meant I couldn't skip school. The only alternative was to print the programming problems from sites like Codechef and SPOJ and solve them by hand during school hours. I sat in the last bench and didn't pay even an iota of attention to what was being taught by the teacher. I used to write the codes by hand so that when I got home, I could simply type them out on the computer, debug them and submit them. This was one of the many ways I tried saving time. I had to because the half-yearly examinations were just 2 months away, and I had to do well on them as well since I was applying abroad for admissions.

December '13-January '14: Things got even more stressful. During this period, I had to manage time between programming, school, cricket and foreign applications. I had U-19 Delhi Zonal Matches going on, application deadlines were right around the corner, and INOI (Indian National Olympiad in Informatics, the national level of the selection procedure to IOI training camp) was a month and a half away. To top it all, pre-board exams were due in a month.

This was my last chance to take INOI. If I didn't make it, IOI would just be a dream forever. Towards end of December, I decided to give up on pre-board preparation. I used to get up at around 4 in the morning, do two hours of problem solving, go for cricket practice at 6.30, return at 9.30, do my foreign applications for 3-4 hours, and spend the rest of the day programming, up till 12 in the night.

INOI happened. And pre-boards also went by. I qualified for IOITC but did terribly in the latter. But I couldn't care less.

February '14-March '14: I had to focus on my board examinations. Didn't program at all for a month and a half.

April '14: This was the time I started preparing for IOITC seriously. From 6 AM in the morning up till 1-2 AM in the night, just programming. Codechef, Codeforces, past IOITC problems, reading codes of other people, learning weird techniques from Chinese and Russian blogs, and what not. For one month, this was how my days were.

May '14 - July '14: Once I made it to the Indian team, the same schedule repeated itself up till IOI.

So my typical day was like: wake up at 4-5 AM in the morning, get ready, get to programming, do a set of 3 IOI (or similar contest) problems in 5 hours (6-11 AM), spend 2 hours analysing mistakes, learning what had to be learnt from the attempted problems, then break for lunch and GTA and some youtube. Then back to 5 hours slot from 3-8 PM. Then dinner, then 2 hours analysis and stuff again. Then some more reading of algorithms and techniques up till 1 AM in the night, and then finally -- the much needed -- sleep.

When IOI ended, I realised I had become so fat. All I used to do was sit in front of the computer the whole day. Here is a picture of me (second from left) at IOI '14. The one in the middle is none other than Michal Forišek. You may not agree that I look fat, but trust me that shirt is just good at hiding my pot belly. I had put on 12 kgs from what I was in December:

Coach:

This was much more fun. When I was in the IOI 2015 Pre-departure camp with the Indian team in Chennai Mathematical Institute, the schedule was quite similar to what it was when I was preparing. Just that my duties were different. I shared the room with the 3 male contestants, Tanay, Arjun and Kushagra. We used to wake up at about 9 AM, and start with programming at 10 AM. Mostly problem solving, technique discussions, teaching sessions, and chalk fights!!! This went up till 11 in the night. Then some rest. Then Tanay and Kushagra would start programming again 12 onwards. I used to spend that time hunting for good problems that could be used to teach new techniques, or new ways of approaching problems. This went up till 4 AM in the night. And then we just crashed into our beds.
Sometimes, we just crashed where we were. This used to be our state after 20 hours of programming:

But then all of it was worth it. There isn't a better feeling than when you see this:

3 Bronzes and 1 Silver. It just makes you feel so strong that you did your part in helping your country do well at one of the most prestigious programming competitions :)

What is it like to participate in ACM-ICPC contests, knowing that you are the weakest member of your team and that your teammates are doing all of the work for you? Do you feel embarrassed? How do you manage?

Nick Wu

Firstly, a more thorough description of my situation. Last year, I was a member of the Stanford ACM-ICPC World Finals team, along with Chenguang Zhu and John Pardon. Chenguang and John were both red coders on TopCoder - John was also a three-time IOI gold medalist. I was barely a yellow coder.

During our first few practices, it was glaringly obvious that I was the weakest algorithmically. The one redeeming factor I had was that I was a fast coder. My job was to secure a lead for our team, and to give Chenguang and John time to solve the harder problems of the set. I felt bad being useless past the second hour of any given contest - as a result, I started practicing more and more outside of our official practices. At one point, I was solving contest problems for 30 hours a week.

Competing with them was exciting nonetheless. Our practices usually went something like this:

1) I spend the first couple hours solving the easy problems of the set. Chenguang and John would go through the problems and categorize them, then work on solving the harder ones.
2) After I finish solving the easy problems, Chenguang and John take over for the remainder of the contest while I sit there and try to make progress on unsolved problems.

Being the fast coder on the team was convenient for us though - since I was the weakest algorithmically, it meant that John and Chenguang had more time to deal with the harder problems of the contest while I hacked away on a keyboard. At a moment's notice, too, I could just start working on a time-consuming implementation problem while debugging happens on paper.

One example of a contest that went terribly for me was the 2012 University of Chicago Invitational Programming Contest. Our team placed 3rd. The only problem I solved was problem I - Chenguang and John basically did that contest by themselves, having solved the other 7 problems that we ended up solving by themselves. That contest was

so demoralizing for me that I spent a full week mentally trying to recover. I didn't want to go to World Finals and solve a single problem and then be useless for the rest of the contest.

For Warsaw, the plan was for me to tackle the time-consuming implementation problems while Chenguang and John worked on the more algorithmic problems. This was never actually realized - instead of taking a backseat after solving B, I ended up solving C by myself, coding up E with more for loops than I ever imagined using in a single program, and solving L with John. If you had told me before the contest that I would be instrumental in solving more than half of the problems that we solved during the contest, I would have told you that you were completely crazy.

Being the weakest person is only as bad as you make it out to be. ACM-ICPC is a team contest. If you think of ACM-ICPC as a contest where you have three individuals working on a contest, you're not doing it right. As we've seen time and time again from World Finals scoreboards, it is rarely the team that looks the strongest on paper that wins. Sometimes it's luck, but a lot of the time it's good teamwork.

Just because you're the weakest on paper doesn't mean you won't make meaningful contributions during a contest. Focus not on your past, but on your present.

কিভাবে ভাল প্রোগ্রামার হওয়া যায়?

Hasan Abdullah

নীলক্ষেতে চক্কর মারলে দেখতে পাবে *Teach Yourself Java in 7 Days* টাইপের বেশ কিছু বই। লেখকরা আসলে তোমাকে আশ্বস্ত করার চেষ্টা করেন যে প্রোগ্রামিং খুব সহজ জিনিস। হুম, আসলেই সহজ *যদি না তা সত্যিই প্রোগ্রামিং হয়*। কোন একটা ফোরামে প্রশ্ন দেখেছিলাম "এক রাতের মধ্যে কিভাবে প্রোগ্রামার হওয়া যায়?" রসিক এক প্রোগ্রামার রিপ্লাই দিয়েছিলেন "তোমার ল্যাপটপ আর নেট কানেকশন সহ উত্তর মেরুতে চলে যাও। সেখানে ৬ মাস রাত থাকে। এক রাতেই শিখা যাবা!"
মানুষের সহজাত প্রকৃতিই হচ্ছে 'অল্প সময়ে, কম কষ্টে, ঘরে বসে' হাওয়া বাতাস খেতে খেতে সাফল্যের শীর্ষে চলে যাওয়ার ইচ্ছা। যে কোন বিষয়ের মত প্রোগ্রামিং এ ভাল করার জন্যেও প্রয়োজন কঠোর পরিশ্রম ও সঠিক অনুশীলন। প্রোগ্রামিং এ এক্সপার্ট হবার ব্যাপারে পিটার নরভিগ (Peter Norvig) Teach Yourself Programming in Ten Years নামের একটা আর্টিকেল লিখেছিলেন। আসলেই একজন এক্সপার্ট প্রোগ্রামার হিসেবে নিজেকে প্রতিষ্ঠিত করতে একটা দীর্ঘ পথ পারি দিতে হয়।

Lukáš Poláček তার একটা লেখায় প্রোগ্রামিং এ ভাল করার জন্য বিশেষ করে প্রোগ্রামিং কনটেস্ট এ ভাল করার জন্য ৭টা পয়েন্ট বিশিষ্ট ট্রেইনিং প্রোগ্রাম আর ৩টা পয়েন্টের সমন্বয়ে মোটিভেশন এর উপর আলোকপাত করেছেন। তার জবানিতে নিচে সেই লিখাটার ভাবানুবাদ তুলে ধরছি।

ট্রেইনিং প্রোগ্রাম

ট্রেইনিং ফিলোসফি

১৫ বছর বয়স পর্যন্ত আমি ওরিয়েন্টিয়ারিং (orienteering) এর সাথে জড়িত ছিলাম। এটা বনের মধ্যে ম্যাপ দেখে নির্দিষ্ট স্থানে সবার আগে পৌঁছানোর এক ধরণের প্রতিযোগিতা। আমি অনেক অনেক training করেছিলাম। বছরে ১৭০০ কিলোমিটার দৌঁড়েছিলাম পাহাড়ী এলাকায়। এরপর অবশ্য স্বাস্থ্যগত কিছু ঝামেলার কারণে এটা বাদ দিতে হয়েছিল। ওরিয়েন্টিয়ারিং বাদ দেয়ার পর আমার হাতে প্রচুর ফ্রি সময় আসলো। তখন এই সময়টাকে কাজে লাগালাম ম্যাথ আর প্রোগ্রামিং করার পিছনে। অ্যাথলেটদের ট্রেইনিং এর কিছু সিসটেম আমি কনটেস্ট প্রোগ্রামিং এর ট্রেইনিং এর উপর অ্যাপ্লাই করতাম।
আমার কাছে কিছু কিছু দিক দিয়ে প্রোগ্রামিং কনটেস্ট এর ট্রেইনিং আর ওরিয়েন্টেরিয়ারিং এর ট্রেইনিং প্রায় একই রকম মনে হত। এই যেমন, খেলোয়াড়দের ট্রেইনিংগুলো হয় various রকমের। যেখানে যেখানে তাদের দূর্বলতা পাওয়া যায় সেদিকেই ফোকাস করা হয়ে থাকে।

ধর, তুমি ফুটবল টিমের একজন স্ট্রাইকার। দৌঁড়ে ছুটে তুমি গোল করতে ভাল পারো। তুমি এটার উপরেই এক্সপার্ট। কিন্তু তুমি পেনাল্টি কিকের ব্যাপারে দূর্বল। কোচ কিন্তু তোমাকে নিয়মিত প্র্যাক্টিসের সাথে পেনাল্টি কিকের উপরও প্র্যাক্টিস করাবেন।
এই একই জিনিসটা ফলো করা উচিত কনটেস্ট প্রোগ্রামিং এর ট্রেইনিং সেশনগুলোতে। তুমি শুধু একটা সাইডের উপরই সলভ করে যাবে না। যদি তুমি জ্যামিতির প্রবলেমে দূর্বল হও কিন্তু ডিপিতে বেশ ভাল। তাহলে তুমি সারা কনটেস্ট লাইফে শুধু

ডিপিই সলভ করে যাবে এমন হওয়া ঠিক হবে না। জ্যামিতির টুকটাক প্রবলেমও সলভ করা উচিত কনটেস্ট শেষ হবার পরে হলেও। অর্থাৎ বিভিন্ন টপিক সম্পর্কে তোমার জ্ঞানের পরিধিকে একটু একটু করে বাড়াতে হবে।
মোদ্দা কথা হচ্ছে, তোমার ট্রেইনিং ফিলোসফি হবে "The training must be varied and you should focus on your weaknesses".

শিখতে হবে ডেটা স্ট্রাকচার ও অ্যালগরিদম

একজন শিক্ষার্থী যখন কনটেস্টে ভাল করতে চায় সে ডেটা স্ট্রাকচার আর এলগরিদম শেখা শুরু করে। Coremen এর লেখা অ্যালগরিদমের উপর অসাধারণ একটা বই আছে। এছাড়াও আরো অসংখ্য ব্লগ, টিউটোরিয়াল আছে এগুলো স্টাডি করার জন্য। ভার্সিটিতেও অ্যালগরিদমের কোর্স করানো হয় প্র্যাক্টিক্যাল সহ।
তবে কনটেস্ট প্রোগ্রামিং এর ক্ষেত্রে ডেটা স্ট্রাকচার-অ্যালগরিদমের নাম যতটা শোনা যায় আমার কাছে একটু বেশি-বেশিই মনে হয়। শুধুমাত্র এই দুইটা টপিক জানলেই অনেক ভাল কনটেস্ট্যান্ট হওয়া যাবে এই ধারণা খানিকটা ভুল। একজন ভাল প্রোগ্রামার হবার জন্য ডেটা স্ট্রাকচার-অ্যালগরিদম অন্যতম জরুরি একটা টপিক কিন্তু এটাই শেষ নয়। অনেক অনেক ডাটা স্ট্রাকচার -এলগরিদম জানা ছাড়াও তুমি ভাল প্রোগ্রামার হতে পারবে।
একটা উদাহরণ দেয়া যাক। তোমার শখ হল মাউন্টেইন বাইকিং করার। ভাঙ্গাচুরা রাস্তা, পাহাড়-পর্বতে সাইকেল নিয়ে লাফালাফির ভিডিও দেখে তুমি মাউন্টেইন বাইকিং এর উপর একটা বই কিনলা। আরো ভিডিও দেখে দেখে সব কারসাজি শিখা শুরু করলা। যদিও তুমি সাইকেল চালানোই জানো না। দুই মাস স্টাডির পর সাইকেল নিয়ে পাহাড়ে গিয়ে দেখলে "জীবন অত সোজা না!" যেই ব্যাসিক জ্ঞানগুলো তোমার আগেই জানার দরকার ছিল তা না জানার কারণে তুমি এখানে থিওরি জেনেও কিছু করতে পারছ না। সমান রাস্তায় সাইকেল চালানোতে এক্সপার্ট হলে পাহাড়ে গিয়ে কিছু দূর অন্তত চালাতে পারতে। প্রোগ্রামিং এর ব্যাপারটাও অনেকটা এরকম। তোমাকে ভাল করতে হবে তাই এক্ষনি এত্ত এত্ত অ্যালগো শিখতে বসে যাওয়ার চেয়ে আস্তে ধীরে বুঝা উচিত কোন অ্যালগো কিভাবে কাজ করছে, কেন এবং কখন কাজ করছে? এই প্রবলেম সলভ করার বিভিন্ন অ্যালগোর লাভ ক্ষতি কী? অনেক সময়ই খুব বেশি ইনফরমেশন কালেক্ট করার প্রবণতা ক্ষতির কারণ হয়ে দাঁড়ায়। তুমি তোমার রেঞ্জের প্রবলেম সলভ করতে থাকো, সাথে এগুলো শিখতে আর প্র্যাক্টিস করতে থাকো। ভুলটুল করতে করতেই শিখতে পারবা। কারণ মানুষ ভুল থেকেই শিখে।
যে কোন টপিক কারো থেকে শেখার আগে নিজে নিজে কিছুটা ঘাটাঘাটি করলে শেখানো আর শেখা দুইটা কাজই দারুণ হয়। আরেকটা উদাহরণ দেই। তোমার অ্যালগরিদম ক্লাসে স্যার ডেটা স্ট্রাকচার আর টাইম কমপ্লেক্সিটির ধারণা দেয়া ছাড়াই তোমাকে একটা unweighted graph এর shortest path বের করার প্রোগ্রাম করতে দিল। তোমার হালকা পাতলা ধারণা থাকার কারণে কোন রকমে একটা প্রোগ্রাম করে দিলে। সেটা ততটা ইফিসিয়েন্ট হল না। পরের লেকচারে স্যার দেখালেন কিভাবে queue implementation করতে হয় আর টাইম কমপ্লেক্সিটি কিভাবে হিসাব করতে হয়। এরকম শেখানোর পদ্ধতিটা কিন্তু 'আগে থিওরি, পরে প্র্যাক্টিক্যাল' এই সিসটেমের চেয়ে বেশি কার্যকর।

প্রবলেম সলভিং এর ক্ষমতা বাড়াও

এই কাজটা করা বেশ কঠিন। প্রবলেম সলভিং এর স্কিল বাড়ানো বলতে আসলে 'তোমার বুদ্ধিমত্তাকে আরো বাড়াও' এমনটাই বুঝায়।
আর এটা করার উপায় হচ্ছে নিয়মিত কঠিন কঠিন প্রবলেম সলভ করা। আবার এমন কঠিন না যেন হতাশা এসে ভর করে। তোমার ক্ষমতার চেয়ে একটু কঠিন প্রবলেম নিয়ে পড়ে থাকলে তোমার ব্রেইনকে কাজে লাগাতে পারবে। সব সময় সহজ প্রবলেম সলভ করলে দেখবা তুমি ভুল-টুল করতেছ না, কিন্তু তোমার ব্রেইন ঐসব সহজ প্রবলেমের জন্যেই ম্যাচ হয়ে থাকবে। ব্রেইনকে কাজ না দিয়ে অলস বসিয়ে রাখলে (সব সময় সহজ প্রবলেম সলভ করলে) বুদ্ধিমত্তা বাড়ানো কথা চিন্তা করা যায় না।
এজন্য অনলাইন জাজগুলোতে নিয়মিত প্রবলেমের লেভেল দেখে দেখে সলভ করা সবচেয়ে ভাল। পুরো প্রবলেমটাকে ছোট ছোট অংশে ভাগ করে একটা একটা পার্ট করে সলভ কর। এক্ষেত্রে ছোট দরজাওয়ালা একটা বাড়ির বিরাট বড় টেবিলের উদাহরণ দেয়া যেতে পারে। বিরাট বড় গরজিয়াস একটা টেবিল কিনে নিয়ে আসলা তোমার বাড়ির দোতলার ড্রয়িং রুমে

রাখবা বলে। দেখা গেল টেবিলের তুলনায় দরজাটা অনেক ছোট। সেক্ষেত্রে একটাই মাত্র কাজ করার থাকে তা হচ্ছে সাবধানে টেবিলটাকে কেটে টুকরাগুলোকে জায়গা মত নিয়ে আবার জোড়া দেয়া। প্রবলেমগুলোকেও এভাবে ভেঙ্গে ভেঙ্গে সলভ করে এরপর জোড়া দেয়ার কাজ করতে হবে।

প্রোগ্রামিং এর দক্ষতা বাড়াও

আগের পয়েন্টে বলা হয়েছে 'প্রবলেম সলভিং' এর ব্যাপারে। এখানে বলা হচ্ছে 'প্রোগ্রামিং' এর ব্যাপারে। দুইটা কিন্তু দুই জিনিস। তুমি যখন মনে মনে কোন একটা প্রবলেমের সমাধানের ব্যাপারে একটা সিদ্ধান্তে আসতে পারো সেটা হচ্ছে প্রবলেম সলভিং। আর তোমার মনে থাকা এই প্রসেস অনুযায়ী কোড করতে পারাটাই হচ্ছে প্রোগ্রামিং।
কনটেস্টের জন্য প্রবলেম সলভিং যেমন দরকার, প্রোগ্রামিংও দরকার। অর্থাৎ এমন হওয়া যাবে না যে, "কী কাজ করতে হবে সেটা বুঝতেছি কিন্তু কিভাবে কোড করতে হবে সেটা বুঝতেছি না!" এই আপদ এড়ানোর জন্য বেশি বেশি কোড করার বিকল্প নাই। প্রয়োজনে সহজ প্রবলেম সলভ করো। কিন্তু নিয়মিত কোডিং এর মধ্যে থাকা লাগবে।

খাতায় কোড কর আর ডিবাগ কর

যদিও এখনকার সময়ে মনে হয় না কেউ খাতায় কোড করে। এরপরেও যদি খাতায় কোড লিখে প্র্যাক্টিস করতে পার এটা তোমার জন্য বেশ ভাল কাজে দিবে। কারণ তুমি যখন IDE তে কোড করতে বসো, মাথায় এটা থাকেই যে ভুল হলে আবার সাথে সাথেই ঠিক করতে পারবো। কিন্তু কাগজে যখন লিখবা তখন লিখার আগে তোমার ব্রেইন অনেক দ্রুত কাজ করবে যথাসম্ভব নির্ভুল কোড লিখার জন্য। কারণ কাগজে কোড লিখা কঠিন।
আর অনসাইট টিম কনটেস্টের ক্ষেত্রে এই গুণের কারণে আরো সুবিধা পেতে পারো। কারণ সেখানে ৩ জনের জন্য একটাই পিসি থাকে। তাই অন্য টিমমেট পিসিতে কোড করতে থাকলে তোমাকে হয়ত খাতাতেই কোড লিখে রাখতে হতে পারে। কোড লিখে বিভিন্ন ইনপুটের জন্য কাগজেই ডিবাগ করার অভ্যাস গড়ে তুলো।

নিয়মিত কনটেস্টে অংশগ্রহণ

তুমি হয়ত বাসায় বসে ধীরেসুস্থে বেশ ভাল প্রবলেম সলভ বা কোড করতে পার। কিন্তু কনটেস্টের সময়ে? বাসায় সলভ করার সময় কিন্তু নির্দিষ্ট সময় শেষ হয়ে যাবার চিন্তা থাকে না। সেই প্রেশারটা থাকে না যেটা কনটেস্ট টাইমে থাকে। তাই স্ট্রেসের মধ্যে থেকে, প্রেশারের মধ্যে থেকে কিভাবে কার্য সমাধা করতে হয় সেটায় অভ্যস্ত হবার জন্যেও যত বেশি সম্ভব কনটেস্টে অংশ নেয়া উচিত।
আর বেশি বেশি কনটেস্ট করলেই সে সময়ে করণীয় বা কী ধরনের সিদ্ধান্ত নিতে হবে সে ব্যাপারে আইডিয়া হয়ে যাবে। যেমন ৫ ঘন্টার কনটেস্টের শেষ ঘন্টায় এসে একটা নতুন প্রবলেম শুরু করা সম্ভবত ভাল সিদ্ধান্ত না। যখন তোমার সাবমিট করা unsolved ২-৩ টা প্রবলেম রয়েছে। নতুন প্রবলেম শুরু করব কি করব না, কোন একটার পিছনে আর সময় দেয়া উচিত হবে কি হবে না? এই সিদ্ধান্তগুলো নেয়া সহজ হয় বেশি বেশি কনটেস্টে অংশ নিলে।

কোডের কোয়ালিটি বাড়াও

একটা প্রবলেম সলভ করেই শেষ দিয়ে দিও না। এই প্রবলেম অন্যেরা বা ভাল প্রোগ্রামাররা কিভাবে সলভ করেছেন সেগুলোও দেখ। এই প্রবলেমের জন্য হয়ত তাদের কোন সলুশন তোমার আর কাজে আসবে না কিন্তু পরে অন্য কোন এক সময় হয়ত ঠিকই কাজে আসবে।
কোডের ইফিসিয়েন্সি বাড়ানো যায় কিনা সেটা নিয়ে ঘাটাঘাটি করো। কিভাবে আরেকটু কম টাইম এ কাজ সারা যায় বা মেমরি কিভাবে আরেকটু বাঁচানো যায় এগুলো নিয়ে কাজ করো।
প্রোগ্রামিং একটা শিল্প। এর পরতে পরতে তাই থাকে সৌন্দর্য। কোডের সৌন্দর্য বা readability অনেক জরুরি একটা বিষয়।

শরীরের যত্ন নাও

ব্যাক পেইন প্রোগ্রামারদের একটা জাতীয় সমস্যা। মাঝে মধ্যে মনেই হয় যে "ব্যাক পেইন না থাকলে আর কিসের প্রোগ্রামার!" যদিও অনেকেরই ব্যাক পেইন নাই। তবে প্রোগ্রামাররা সাধারণত সারা দিন পিসি নিয়ে পড়ে থাকে তাই নানা রকম শারীরিক সমস্যা দেখা যেতে পারে। পিঠের বা মেরুদন্ডের হাঁড় ক্ষয়ে যাওয়া বা ব্যথা হওয়া প্রোগ্রামারের সংখ্যা চারিদিকে প্রচুর। এছাড়াও অনিদ্রা, চোখের সমস্যা এগুলো তো আছেই!
এজন্য কিছু সময় পিসিতে বসে কিছু সময় বিরতি দেয়া, মাঝে পানি-টানি খাওয়া, শূণ্যের দিকে তাকিয়ে থাকা, সবুজ গাছপালা বা নীল আকাশের দিকে তাকিয়ে থাকা ইত্যাদি করা যেতে পারে। এতে চোখ, ব্রেইন ইত্যাদির ক্লান্তি কমবে। কাজের প্রোডাক্টিভিটি বাড়বে।

মোটিভেশন (প্রেরণা)

প্রোগ্রামিং কনটেস্ট তোমার প্রোগ্রামিং এর স্কিলকে ব্যাপক ভাবে বাড়িয়ে দিবে। তুমি চিন্তা করতে পারবা না যে একেকটা কনটেস্টের পর তোমার স্কিলটা কতখানি পুশ হয়! হ্যাঁ তোমাকে অনেকেই বলবে যে রিয়েল লাইফে সফটওয়্যার ডেভেলপমেন্টের জন্য এসব প্রবলেম সলভিং এর দরকার হয় না। আসলেই তাই! তুমি খুব বেশি ভাগ্যবান হলে হয়ত তোমার কোম্পানীর কোন প্রোজেক্টে ৫-১০% কাজে তোমার শেখা ডেটা স্ট্রাকচার-অ্যালগরিদম ইমপ্লিমেন্ট করতে পারবা। বা তোমার কনটেস্টের স্কিল কাজে লাগাতে পারবা। কারণ কন্টেস্ট এরিয়াটা রিয়েল ওয়ার্ল্ডের চেয়ে ভিন্ন। কিন্তু কনটেস্ট করলে তোমার দক্ষতা যতটা বাড়বে তা অন্যান্য উপায়ে অর্জন করা কঠিন। একটা জিনিস চিন্তা করো, ছোট বেলায় আমাদেরকে গণিত শেখানো হয়। তার মানে কি আমাদের সবাইকে গণিতবিদ হতে হবে? আমাদেরকে কবিতা পড়ানো হয়, রচনা লিখা শেখানো হয়। তাই বলে আমরা সবাই কি লেখক হব? গণিত শেখানো হয় যেন ব্যক্তি জীবনে হিসাব নিকাশ করতে পারি না কোন একটা বিষয়ে ভাল অ্যানালাইসিস করতে পারি। লেখালেখি শেখানোর কারণ হচ্ছে যেন তুমি যে কোন বিষয়ে তোমার মনের ভাব সুন্দর ভাবে প্রকাশ করতে পারো। প্রোগ্রামিং কনটেস্টটাও ঠিক তেমন। তোমার ভবিষ্যতের ভীতটা গড়তে সাহায্য করবে।
কনটেস্ট তুমি তখনই করতে পারবা যখন এটাকে একটা খেলা হিসেবে নিবা। ছোট বেলায় বিকালে ক্রিকেট খেলতে বের হয়ে খেলার সময় বা খেলার পর যেই আনন্দ পেয়েছিলে, সেটা যদি এখানেও পাও তাহলে কনটেস্ট তোমার জন্যেই! কিন্তু তা না হয়ে যদি চিন্তা কর "অমুক ভাই কনটেস্ট করে এখন অমুক জায়গায় বড় জব করে। আমিও কনটেস্ট করবো" তাহলে মনে হয় না খুব বেশি দূর আগানো সম্ভব। কনটেস্ট ছাড়াও অনেক ভাল প্রোগ্রামার আছেন। অনেক বড় বড় প্রোগ্রামার আছেন যারা কনটেস্ট এর ব্যাপারে আগ্রহী না। যদি তোমার আগ্রহ না থাকে তাহলে জোর করার দরকার নাই। কারণ জানোই তো জোর করে ভালবাসা হয় না!
তুমি যদি কনটেস্ট করতে আনন্দ পাও তাহলে করো। যদি আনন্দ না পাও তাহলে অন্য কোন প্রোডাক্টিভ কাজ কর। যদি প্রোগ্রামিং না করে দিন ভর গেমস খেলে বা ফেসবুকিং করে সময় পার করো তাহলে আরেক বার ভাবো নিজের সম্পর্কে। আরেকবার ভাব আজ থেকে ৫ বছর পরের কথা। যে কোন একটা সেক্টরে এক্সপার্ট হও। হোক তা প্রোগ্রামিং বা নেটওয়ার্কিং! যে কোন কিছু! পাশ করে বের হয়ে যেন সিনিয়র কাউকে বলতে না হয় "ভাই কিছুই তো পারি না! কী করব এখন?"

ব্যর্থতা

পদে পদে বাধা আসবে, হতাশা আসবে। অমুক জুনিয়র যেই প্রবলেম আধা ঘন্টায় সলভ করেছে আমার সেটা লাগলো পাক্কা সাড়ে চার দিন!!! আমাকে দিয়ে আর প্রোগ্রামিং হবে না!!! এমন কথা মাথায় এনো না।
সবার জীবনেই না পাওয়া থাকে। ব্যর্থতা থাকে। এমন কি যত যত সফল মানুষ দেখি। তাদেরও প্রত্যেকের জীবনেই ব্যর্থতার গল্প রয়েছে। সবাই জানে ব্যর্থতা থেকে বা ভুল থেকে শিক্ষা নিয়ে নিজেকে সামনের দিকে ঠেলে দিলেই সফলতা আসবে। কিন্তু সবাই এটা জানে না এই কাজটা ঠিক কিভাবে করতে হয়! আত্মসমালোচনা করো, নিজের সাথে কথা কথা বলো। নিজের ভুলগুলোর ব্যাপারে নফসকে (মন বা অন্তরকে) বলো সতর্ক হতে, ভাল দিকগুলোর জন্য নিজেকে নিজেই অ্যাপ্রিশিয়েট করো। সেরেফ লেগে থাকো যে কাজই করো না কেনো, যদি তাতে আনন্দ পাও। সফলতা আসবেই!

Teach Yourself Programming in Ten Years

পিটার নরভিগ | অক্টোবর ১৮, ২০১৩

Peter Norvig (born December 14, 1956) is an Americancomputer scientist. He is a director of research at Google Inc., and used to be its director of search quality.

২১ দিনে প্রোগ্রামিং শেখা আসলে সম্ভব না। চমৎকার প্রোগ্রামার হবার জন্য প্রয়োজন প্রচুর সময় নিয়ে স্বেচ্ছাকৃত অনুশীলন। কিভাবে নিজেকে দশ বছর সময়ের মধ্যে একজন চমৎকার প্রোগ্রামারে পরিণত করা যায় সেটা নিয়ে লিখেছেন গুগলের ডিরেক্টর অফ রিসার্চ - পিটার নরভিগ। পিটার নরভিগের অনুমতিক্রমে লেখাটা বাংলায় অনুবাদ করেছেন ইকরাম মাহমুদ।

১. সবার এত তাড়া কেন?

যদি তুমি হেঁটে কোন একটা বই এর দোকানে গিয়ে ঢোকো বইয়ের তাকে তুমি Teach Yourself Java in 7 Days টাইপের একগাদা বই দেখতে পাবে। যেগুলোর প্রত্যেকটা দাবি করবে যে সেই বইটা তোমাকে খুব ভালোভাবে ভিজুয়াল বেসিক, উইন্ডোজ, ইন্টারনেট, ইত্যাদি সব শিখিয়ে দিতে পারবে। এবং শুধু সেখানেই শেষ না, তাদের দাবি সেটা শেখাতে তাদের বড়জোর কয়েক ঘন্টা লাগবে - কিংবা খুব বেশি হলে কয়েকটা দিন। আমি **অ্যামাজনে** একটা পাওয়ার সার্চ মারলাম নিচের কুয়েরিটা লিখে

```
pubdate: after 1992 and title: days and
      (title: learn or title: teach yourself)
```

এবং আমার কুয়েরিটার জন্য অ্যামাজন ২৪৮টা বই খুঁজে বের করলো। তার মধ্যে প্রথম ৭৮ টা বই হচ্ছে কম্পিউটারের উপর (৭৯ নাম্বার টা হচ্ছে **Learn Bengali in 30 days**)। আমি কুয়েরিতে "days" এর বদলে "hours" লিখেও প্রায় একই সমান রেজাল্ট পেলাম - ২৫৩ টা বই, যাদের মধ্যে ৭৭টা হচ্ছে কম্পিউটারের উপর, আর তারপর ৭৮ নাম্বারটা হচ্ছে **Teach Yourself Grammar and Style in 24 Hours**। মজার ব্যাপার হচ্ছে, প্রথম ২০০ টা বইয়ের ৯৬% হচ্ছে কম্পিউটারের ওপর।

আমরা এটা থেকে এই উপসংহারে পৌছাতে পারি যে, হয় মানুষজনের ভীষণ তাড়া কম্পিউটার শেখার জন্য অথবা বাকি সবকিছুর সাথে তুলনা করতে গেলে কম্পিউটার হচ্ছে খুবই সহজ একটা জিনিস শেখার জন্য। কোথাও কোন বই এর অস্তিত্ব নেই যেটা বিটোভেন বা কোয়ান্টাম ফিজিক্স শেখাতে পারে কয়েকদিনের মধ্যে। এমনকি কয়েকদিনের মধ্যে কিভাবে কুকুরের চুল কাটা শেখা যায় সেটা নিয়েও কোন বই নেই। ফেলিসন এবং

তার বন্ধুরা এই ব্যাপারে মন্তব্য করেছিলো তাদের বইতে। **How to Design Programs** বইটাতে ওরা লিখেছিলো - "Bad programming is easy. *Idiots* can learn it in 21 days, even if they are *dummies*."

আমরা একটু বিশ্লেষণ করি, **Learn C++ in Three Days** এরকম একটা বইয়ের নাম থেকে আমরা কি কি বৃত্তান্ত পেতে পারি -

- Learn: মাত্র তিন দিনে হয়তো তোমার সময়ও হবে না ভালোমতো কয়েকটা প্রোগ্রাম লেখার। তোমার ব্যর্থতা আর সফলতা থেকে শিক্ষা লাভ করার সময় হবার তো প্রশ্নই আসে না। তোমার সময় হবে না একজন অভিজ্ঞ প্রোগ্রামারের সাথে কাজ করার এবং বোঝার যে C++ এর সত্যিকারের পৃথিবীটুকু কেমন বা C++ নিয়ে ইন্ডাস্ট্রিয়াল কাজ করার অভিজ্ঞতাও কেমন। সংক্ষেপে, তোমার সময়ই হবে না কোন কিছু শেখার। তো বইটা শুধু তোমাকে একটা কৃত্তিম আবছা আবছা পরিচয় দিতে পারবে, কিন্তু কোন গভীর জ্ঞান দিতে পারবে না। এবং আমি জানি, তুমি অবশ্যই ছোট বেলা থেকে শুনে এসেছো, "অল্পবিদ্যা ভয়ংকরী"। কথাটা মিথ্যে না।
- C++: তুমি যদি আগে অন্য কোন ল্যাঙ্গুয়েজে প্রোগ্রামিং করে থাকো তাহলে তিন দিনে তুমি হয়তো কিছু সিনট্যাক্স শিখতে পারবে C++ এর। কিন্তু তুমি কিছুতেই এর চে' বেশি কিছু শিখতে পারবে না এই ল্যাঙ্গুয়েজটা ব্যবহার করা নিয়ে। ধরো, তুমি যদি আগে **Basic** এ প্রোগ্রাম লিখে থাকো, তুমি হয়তো শিখবে কিভাবে তোমার Basic এর প্রোগ্রামটাকে C++ এ লেখা যায়। কিন্তু তুমি শিখবে না, কেন C++ ভালো (কিংবা খারাপ)। যদি ব্যাপারটা তাই হয়, তাহলে কি লাভটা হলো বলো? অ্যালান পারলিস বলেছিলো, "A language that doesn't affect the way you think about programming, is not worth knowing"। একটা লাভ হয়তো, তুমি এক চিমটি C++ শিখবে (যেটা হয়তো খুবই সামান্য জাভাস্ক্রিপ্ট কিংবা ফ্ল্যাশের ফ্লেক্স শেখার মতো হবে) যেটুকু দিয়ে তুমি একটা ছোট্ট কিছু একটা কাজ করতে পারবে (ত্রিভুজের ক্ষেত্রফল বের করা টাইপের) একটা সাদামাটা ইন্টারফেস লিখে। কিন্তু তাহলে তো তুমি প্রোগ্রামিং শিখছো না, তুমি শুধু শিখছো ঠিক ওই নির্দিষ্ট কাজটা কিভাবে করতে হয়।
- in Three Days: দূর্ভাগ্যজনকভাবে, তিন দিন খুবই কম সময় এবং কোনভাবেই যথেষ্ট নয়। আমি এর পরের সেকশনে এটা নিয়ে আরো বিস্তারিত লিখছি।

২. দশ বছরে প্রোগ্রামিং শেখা

গবেষকরা (**Bloom (1985)**, **Bryan & Harter (1899)**, **Hayes (1989)**, **Simmon & Chase (1973)**) দেখিয়েছে, কোন কিছুতে এক্সপার্ট হতে প্রায় দশ বছর সময় লাগে। এই কথাটা একটা বিশাল রেঞ্জের ফিল্ডের জন্যই সত্যি - দাবা খেলা, সঙ্গীতচর্চা, টেলিগ্রাফ অপারেশন, ছবি আঁকা, পিয়ানো বাজানো, সাঁতার, টেনিস কিংবা ধরো নিওরোসাইকোলজি এবং টপোলজি নিয়ে রিসার্চ করা। এখানে সাফল্যের চাবি হচ্ছে *স্বেচ্ছাকৃত* প্র্যাকটিস: তাই বলে সেটা শুধু একই জিনিস বারবার করা না, নিজেকে ক্রমাগত চ্যালেঞ্জের মুখোমুখি করা কঠিন থেকে কঠিনতর কাজ দিয়ে যেগুলো তোমার বর্তমান সামর্থ্যের বাইরে। তারপর সেই ধরণের কাজে অ্যাটেম্পট নেয়া উচিত, নিজের পারফরমেন্স বিশ্লেষণ করে, ভুল শুধরে, কিভাবে আরো ভালোভাবে সেটা করা যায় সেটা ভাবা উচিত। এবং তারপর উচিত পুনরায় এবং পুনরায় এই প্রসেস রিপিট করা। সত্যি কথা, কোথাও কোন

সহজ শর্টকাট নেই: এমনকি মোজার্ট, যাকে মাত্র ৪ বছর বয়সে মিউজিকাল প্রডিজি ভাবা হতো, তারও আরো ১৩ বছর লেগেছে তার প্রথম বিশ্বমানের সঙ্গীত রচনা করতে। আবার ধরো, তোমার মনে হতে পারে বিটলস ১৯৬৪ সালে রাতারাতি অনেকগুলো এক নাম্বার হিট নিয়ে এড সালিভানের শোতে উপস্থিত হয়েছিলো । কিন্তু সত্যি কথা কি, তারা আসলে লিভারপুলের ছোট ছোট ক্লাবে পারফর্ম করতো ১৯৫৭ সাল থেকে। যদিও তাদের কিছুটা জনপ্রিয়তা ছিলো তারপরও তাদের প্রথম সফল অ্যালবাম Sgt. Peppers বের হয়েছিলো ঠিক দশ বছর পর ১৯৬৭ সালে। **ম্যালকম গ্ল্যাডওয়েল** বার্লিন একাডেমি অফ মিউজিক এ ছাত্র-ছাত্রীদের উপর একটা গবেষণার কথা লিখেছে। পুরো ক্লাসটাকে কোয়ালিটির দিক দিয়ে তিন গ্রুপে ভাগ করে সবাইকে আলাদা আলাদা ভাবে জিজ্ঞেস করেছিলো তারা কতটুকু প্র্যাকটিস করেছে। এই হচ্ছে তার মন্তব্য -

> এই তিনটা গ্রুপের সবাই প্রায় এক সময় সঙ্গীতচর্চা শুরু করে - প্রায় ৫ বছর বয়সে। প্রথম কয়েক বছরে তাদের সবাই মোটামুটি একই ধরণের প্র্যাকটিস করেছিলো - সপ্তাহে ২ বা ৩ ঘন্টা করে। তারপর ৮ বছর বয়স থেকে সত্যিকারের পার্থক্য তৈরী হতে শুরু করে। যেসব ছাত্র-ছাত্রী তাদের ক্লাসের সেরা গ্রুপে পড়ে তাদের সবাই বাকি সবার চেয়ে অনেক বেশি প্র্যাকটিস করতে শুরু করে এই বয়সে এসে - ৯ বছর বয়সে সপ্তাহে ৬ ঘন্টা করে, ১২ বছর বয়সে সপ্তাহে ৮ ঘন্টা করে, ১৪ বছর বয়সে সপ্তাহে ১৬ ঘন্টা এবং এরপর সংখ্যাটা শুধু বাড়তে থাকে বয়সের সাথে সাথে। আসলে এরপর থেকে ২০ বছর বয়স পর্যন্ত ওদের প্রত্যেকে সপ্তাহে অন্তত ৩০ ঘন্টা করে প্র্যাকটিস করছিলো। তো ২০ বছর বয়সের মধ্যে সবচে' এলিট পারফর্মারদের প্র্যাকটিসের সময়ের পরিমাণ ছিলো ১০,০০০ ঘন্টার বেশি। সেখানে মোটামুটি ভালো ছাত্রদের প্র্যাকটিস ছিলো ৮,০০০ ঘন্টা আর ভবিষ্যৎ মিউজিক টিচারদের ছিলো মাত্র ৪,০০০ ঘন্টার মতো।

তো হয়তো ১০ বছর নয়, এই ১০,০০০ ঘন্টাই হচ্ছে ম্যাজিক নাম্বার। হেনরি কারটিয়ের ব্রেসন (১৯০৮-২০০৪) বলেছিলো, "Your first 10,000 photographs are your worst," (কিন্তু সে প্রতি ঘন্টায় একাধিক ছবি তুলতো)। স্যামুয়েন জনসন (১৭০৯-১৭৮৪) এই চিন্তাটাকে আরেকটু দূর নিয়ে গিয়েছিলো এটুকু বলে যে "Excellence in any department can be attained only by the labor of a lifetime; it is not to be purchased at a lesser price." চৌচার (১৩৪০-১৪০০) অভিযোগ করেছিলো এই বলে যে, "the lyf so short, the craft so long to lerne."। হিপোক্রেটাস বিখ্যাত তার "ars longa, vita brevis" বাণীর জন্য, যেটা আসলে "Ars longa, vita brevis, occasio praeceps, experimentum periculosum, iudicium difficile" এর একটা ছোট্ট অংশ। যেটা বাংলায় অনুবাদ করলে দাঁড়াবে, "জীবন ছোট্ট, শিল্প বড় দীর্ঘ, সুযোগ শুধুই পলায়মান, এক্সপেরিমেন্ট বিশ্বাসঘাতক, মীমাংসা বড়ই কঠিন"।

৩. তুমি তাহলে প্রোগ্রামার হতে চাও

এটা হচ্ছে সফল প্রোগ্রামার হবার জন্য আমার রেসিপি

- প্রোগ্রামিং এ প্রকৃতভাবে **আগ্রহী** হও এবং প্রোগ্রামিং কর শুধুমাত্র মজা পাবার জন্য। নিশ্চিত করো যে তুমি ঠিক ততটাই মজা পাচ্ছো ঠিক যতটা মজা পেলে প্রোগ্রামিং এর পেছনে তোমার জীবনের ১০ বছর বা ১০,০০০ ঘন্টা অতিবাহিত করতে পারবে।
- **প্রোগ্রাম লেখো**। কোন কিছু শেখার জন্য সবচে' ভালো উপায় হচ্ছে সেটা করতে করতে শেখা। একটু টেকনিকালি বলতে গেলে, "কোন নির্দিষ্ট এলাকায় পারফর্মেন্সের চূড়ান্তে পৌছানো খুব স্বাভাবিকভাবে অভিজ্ঞতার সাথে সাথেই আসে না। বরং, এমনকি সবচে' অভিজ্ঞদের জন্যও পারফর্মেন্সের লেভেলের উন্নতি ঘটানো যায় উন্নতি করার ক্রমাগত চেষ্টার মধ্য দিয়ে।" (পৃষ্ঠা ৩৬৬) এবং "কাওকে কিছু শেখানোর সবচে' ভালো উপায় হচ্ছে তার জন্য তার ক্ষমতা অনুযায়ী কঠিন কোন সুনির্দিষ্ট কাজ দেয়া। তারপর তাকে ফিডব্যাক দেয়া সে কি ঠিকঠাক করছে আর কি ভুল করছে। তাকে পুনরায় একই কাজ সম্পন্ন করে এবং ভুল শুধরে সেটা ভালোভাবে করতে শেখার সুযোগ দেয়া।" (পৃষ্ঠা ২০-২১) এই বইটা হচ্ছে একটা ইন্টারেস্টিং রেফারেন্স এই দৃষ্টিভঙ্গীর - **Cognition in Practice: Mind, Mathematics, and Culture in Everyday Life**।
- **কথা বলো** অন্য প্রোগ্রামারদের সাথে; অন্যদের প্রোগ্রাম পড়ো। এটা আরো অনেক বেশি গুরুত্বপূর্ণ শয়ে শয়ে বই পড়া বা কোর্স করার চেয়ে।
- তুমি যদি চাও চার বছর খরচ করতে পারো **বিশ্ববিদ্যালয়ে** গিয়ে (অথবা আরো বেশি যদি তুমি মাস্টার্স বা ডক্টরেট করতে চাও)। এটা যেসব চাকরি তোমার ডিগ্রী দেখতে চায় সেরকম জায়গায় তোমাকে চাকরির সুযোগ করে দিবে এবং একই সাথে তোমাকে এই ফিল্ডটাকে আরো গভীরভাবে বোঝার ক্ষমতা দেবে। কিন্তু তুমি যদি প্রথাগত পড়াশুনায় আনন্দ না পাও, তুমি হয়তো একই অভিজ্ঞতাগুলো অর্জন করতে পারো নিজে নিজে আরেকটু বেশি খেটে কিংবা কোথাও কাজ করে। কিন্তু যেভাবেই করো না কেন, শুধু পুঁথিগত বিদ্যা কখনোই খুব যথেষ্ট কিছু হবে না। The New Hacker's Dictionary এর লেখক এরিক রেমন্ড বলেছেন, "Computer science education cannot make anybody an expert programmer any more than studying brushes and pigment can make somebody an expert painter"। আমার নিয়োগ করা সেরা প্রোগ্রামারদের একজনের শুধু মাত্র একটা হাইস্কুল ডিগ্রি ছিলো; কিন্তু তারপরও সে প্রচুর **চমৎকার সফ্টওয়্যার** বানিয়েছে, তার ভক্তদের নিজস্ব **নিউজ গ্রুপ** আছে, আর যথেষ্ট স্টক অপশনও উপার্জন করেছে নিজের জন্য একটা আস্ত **নাইটক্লাব** কেনার জন্য।
- **প্রজেক্টে কাজ করো অন্য প্রোগ্রামারদের সাথে**। হয়তো কখনো তুমি নিজেকে খুঁজে পাবে টিমের সেরা প্রোগ্রামার হিসেবে, আর কখনো দেখবে তুমি টিমের সবচে' বাজে প্রোগ্রামার। যখন তুমি দেখবে তুমি তোমার টিমের সেরা প্রোগ্রামার, নিজের সামর্থ্যকে চ্যালেঞ্জ করো প্রজেক্টটাকে লিড করে, অন্যদেরকে তোমার দূরদর্শিতা দিয়ে অনুপ্রেরণা যুগিয়ে। যখন তুমি জানো তুমি তোমার টিমের সবচে' বাজে প্রোগ্রামার, সেটা একটা চমৎকার সুযোগ কিভাবে দক্ষ প্রোগ্রামাররা কাজ করে সেটা শেখার। অবশ্য তুমি এটাও শিখবে যে তারা কোন জিনিসগুলো করতে অপছন্দ করে (কারণ সেই কাজগুলো হয়তো তারা তোমাকে দিয়ে করাবে)।
- **অন্যদের করে যাওয়া প্রজেক্ট নিয়ে কাজ করো**। বোঝার চেষ্টা করো অন্য কারো লেখা একটা প্রোগ্রাম কিভাবে কাজ করে। বোঝার চেষ্টা করো যে প্রোগ্রামটা লিখেছিলো সে যদি আশেপাশে না থাকে তখন কেমন লাগবে নিজে নিজে ওর প্রোগ্রামটা বুঝতে আর যদি কিছু কাজ না করে সেটা ফিক্স করতে। চিন্তা করো, কিভাবে তুমি তোমার প্রোগ্রামটাকে লিখতে পারো যাতে পরে যারা তোমার প্রোগ্রাম মেইনটেইন করবে তাদের জন্য কাজটা যাতে সহজ হয়।

- **অন্তত আধা ডজন প্রোগ্রামিং ল্যাঙ্গুয়েজ শেখো**। অন্তত একটা শেখো যেটা ক্লাস অ্যাবস্ট্রাকশন সাপোর্ট করে (যেমন C++ বা Java)। একইভাবে অন্তত একটা করে ল্যাঙ্গুয়েজ শেখো যেগুলো এই জিনিসগুলো সাপোর্ট করে - ফাংশনাল অ্যাবস্ট্রাকশন(যেমন Lisp বা ML), সিন্থেটিক অ্যাবস্ট্রাশন (যেমন Lisp), ডিক্লেয়ারেটিভ স্পেসিফিকেশন (যেমন Proglog বা C++ এর টেম্প্লেট), কোরুটিন (যেমন Icon বা Scheme) আর প্যারালিলজম (যেমন Sisal)।
- ভুলে যেও না কম্পিউটার সায়েন্সের মধ্যে আস্ত একটা **কম্পিউটার** অন্তর্ভুক্ত আছে। জেনে নাও কতক্ষণ লাগে তোমার কম্পিউটারের একটা ইন্সট্রাকশন চালাতে, একটা ওয়ার্ড মেমরি থেকে ফেচ করতে (ক্যাশ সহ বা ছাড়া), ধারাবাহিকভাবে পাশাপাশি ওয়ার্ডগুলো ডিস্ক থেকে রিড করতে কিংবা নতুন একটা ডিস্ক লোকেশনে ডাটা খুঁজতে। (এখানে **উত্তর** পাবে)
- **ল্যাঙ্গুয়েজ স্ট্যান্ডার্ডাইজেশন** এর সাথে সংপৃক্ত হও। সেটা যেকোন কিছু হতে পারে, ANSI C++ এর কমিটি হতে পারে, অথবা তোমার ব্যক্তিগত সিদ্ধান্ত হতে পারে যে তুমি ইন্ডেন্টেশন লেভেলে দুইটা স্পেস ব্যবহার করবে নাকি চারটা স্পেস ব্যবহার করবে। যেভাবেই হোক না কেন তুমি শিখতে শুরু করবে কোন জিনিসগুলো মানুষ পছন্দ করে একটা ল্যাঙ্গুয়েজে, কত গভীরভাবে তারা সেগুলোর গুরুত্ব অনুভব করে আর এটাও বুঝতে পারবে যে কেন তারা সেভাবে অনুভব করে।
- একটা ভালো চেতনা তৈরী করো ল্যাঙ্গুয়েজ স্ট্যান্ডার্ডাইজেশন এর ব্যাপারে যেন যত তাড়াতাড়ি সম্ভব এসব স্ট্যান্ডার্ডাইজেশন থেকে **মুক্তি লাভ** করতে পারো।

এই সবকিছু মাথায় রাখার পর, এটা নিয়ে প্রশ্ন করা যায় যে তুমি কতদূর যেতে পারবে শুধুমাত্র পুঁথিগত বিদ্যা দিয়ে। আমার প্রথম সন্তান জন্মানোর আগে আমি মোটামুটি সব "How To" দিয়ে শুরু হওয়া বইগুলো পড়ে ফেলেছিলাম এবং তারপরও আমার নিজেকে খুব নিতান্তই অনভিজ্ঞ মনে হচ্ছিল, যার কোনই ধারণা নেই কোন কিছু সম্বন্ধে। ৩০ মাস পর, যখন আমার দ্বিতীয় সন্তানের জন্ম হলো, আমি কি আগের বইগুলোতে ফিরে গিয়েছিলাম সবকিছু আবার নেড়েচেড়ে দেখার জন্য? নাহ। বরং, আমি নির্ভর করেছি আমার ব্যক্তিগত অভিজ্ঞতার ওপরে, যেগুলো আমার কাছে আরো অনেক বেশি কাজের আর স্বস্তিদায়ক মনে হয়েছে এক্সপার্টদের লেখা হাজার হাজার পাতার চেয়ে।

ফ্রেড ব্রুকস তার রচনা No Silver Bullet এ অসাধারণ সফটওয়্যার ডিজাইনার খুঁজে বের করার তিন-খন্ড পরিকল্পনার কথা লিখেছে। সে বলেছে -

- সিস্টেমিকালি সেরা ডিজাইনারদের খুঁজে বের করো যত জলদি সম্ভব।
- একজন প্রশিক্ষককে দ্বায়িত্ব দাও তাকে গড়ে তোলার জন্য আর স্বযত্নে একটা ক্যারিয়ার ফাইল রাখো তার জন্য।
- সমান অভিজ্ঞতা সম্পন্ন ডিজাইনারদেরকে নিজেদের মধ্যে যোগাযোগ করার সুযোগ করে দাও, যাতে তারা একে অপরকে উদ্দীপিত করতে পারে।

এই চিন্তাধারা ধরে নেয় প্রতিটি মানুষের মধ্যে ক্ষমতা আছে মহান সফ্টওয়্যার ডিজাইনার হবার; আসল কাজ হচ্ছে তাদের গড়ে তোলা। অ্যালান পারলিস আরো সংক্ষেপে এই চিন্তাটা প্রকাশ করেছে, "Everyone can be taught to sculpt: Michelangelo would have had to be taught how not to. So it is with the great programmers"। পারলিস বলছে যে, মহানদের মধ্যে একটা নিজস্ব সহজাত ক্ষমতা আছে প্রশিক্ষণের সীমা অতিক্রম করে যাবার। কিন্তু সেই গুণটা কোত্থেকে আসে? সেটা কি আসলেই শুধু তাদের সহজাত প্রকৃতিদত্ত

গুণ? নাকি তারা সেটা তাদের অধ্যাবসায় আর অনলস পরিশ্রম দিয়ে গড়ে তোলে? এর উত্তর আমি দিতে পারি অগাস্ট গুস্তো (*Ratatouille* সিনেমার কাল্পনিক শেফ) এর কথা থেকে, "anyone can cook, but only the fearless can be great."। আমার মনে হয়, এটা আসে জীবনের একটা বড় অংশ স্বেচ্ছাকৃত প্রতিনিয়ত প্র্যাকটিস করার জন্য সঁপে দেয়ার মনোভাব থেকে। কিংবা হয়তো শুধু *fearless* শব্দটাই এই পুরো চিন্তাটুকুর সারমর্ম। অথবা গুস্তোর সমালোচক, অান্তন ইগোর কথামতো, "Not everyone can become a great artist, but a great artist can come from anywhere."।

তো হাত বাড়াও বইয়ের তাকে, আর কিনে ফেলো ওই জাভা/রুবি/জাভাস্ক্রিপ্ট/PHP বইটা; তুমি হয়তো কিছু একটা শিখবে সেখান থেকে। কিন্তু সেটা তোমার জীবন বদলে দেবে না, তোমাকে মহান প্রোগ্রামার বানাবে না ২৪ ঘন্টায়, ২৪ দিনে বা এমনকি ২৪ সপ্তাহে। কিন্তু কেমন হয়, যদি তুমি পরিশ্রম করতে শুরু করো অল্প অল্প করে নিজেকে ইম্প্রুভ করার সামনের ২৪ মাস সময় ধরে? আমি নিশ্চিত যে, সেটা তোমাকে অনেক দূরে নিয়ে যেতে শুরু করবে …

৪ পরিশিষ্ট

৪.১ প্রথম ল্যাঙ্গুয়েজ

আমাকে অনেকে জিজ্ঞেস করেছে কোন প্রোগ্রামিং ল্যাঙ্গুয়েজ তাদের শুরুতে শেখা উচিত। আসলে এই প্রশ্নের কোন সহজ উত্তর নেই, কিন্তু এই পয়েন্টগুলো বিবেচনা করো।

- *তোমার বন্ধুদের ব্যবহার করো।* যখন কেউ জানতে চায়, "আমি কোন অপারেটিং সিস্টেম ব্যবহার করবো? উইন্ডোজ, ম্যাক নাকি ইউনিক্স?", আমি সাধারণত বলি "সেটা ব্যবহার করো যেটা তোমার বন্ধুরা ব্যবহার করে"। সেটার সুবিধা হচ্ছে সেটা ল্যাঙ্গুয়েজ হোক বা অপারেটিং সিস্টেমই হোক তুমি তোমার বন্ধুদের কাছ থেকে শেখার সুযোগ পাবে। আবার তুমি এভাবেও চিন্তা করতে পারো, ভবিষ্যতে কারা তোমার বন্ধু হবে? তুমি যদি এই ল্যাঙ্গুয়েজ বা অপারেটিং সিস্টেম ব্যবহার করতে শুরু করো সেটা তোমাকে কোন প্রোগ্রামারদের কমিউনিটির অংশ বানাবে? তোমার পছন্দের ল্যাঙ্গুয়েজের কি একটা বিশাল ক্রমবর্ধমান কমিউনিটি আছে, নাকি একটু ছোট্ট মৃতপ্রায় কমিউনিটি আছে? প্রশ্নের উত্তর খুঁজে বের করার জন্য কি বই, ওয়েব সাইট বা অনলাইন ফোরাম আছে? এই সব ফোরামের লোকজনকে কি তোমার পছন্দ হয়?
- *Keep it simple.* C++ বা জাভার মতো ল্যাঙ্গুয়েজগুলো আসলে ডিজাইন করা হয়েছে প্রফেশনাল ডেভেলপমেন্টের জন্য যেখানে সাধারণত প্রোগ্রামারদের একটা বড় টিম থাকে, যারা খুব মাথা ঘামায় প্রোগ্রামের রানটাইম এফিশিয়েন্সি নিয়ে। সে কারণে, এই ল্যাঙ্গুয়েজগুলো এইসব জিনিস চিন্তা করে বেশ জটিল করে ডিজাইন করা। তুমি যদি শুধু প্রোগ্রামিং শিখতে চাও, তোমার এই জটিলতার মুখোমুখি হবার প্রয়োজন নেই। তোমার প্রয়োজন এমন একটা ল্যাঙ্গুয়েজ যেটা ডিজাইন করা হয়েছে সহজে শেখার জন্য আর এমনভাবে ডিজাইন করা হয়েছে যাতে নবীনতম প্রোগ্রামারও সেটা মনে রাখতে পারে।

- *খেলো*। তোমাকে যদি আমি পিয়ানো শিখতে দেই, কিভাবে শেখাটা মজার হবে? একটা উপায় হতে পারে তুমি প্রতিটা কী চাপার পর শব্দ হবে, যেটা হচ্ছে নরমাল ইন্টারেক্টিভ উপায়। অথবা তুমি "ব্যাচ" মোডে শিখতে পারো, যেখানে তুমি পুরো গানটার সবগুলো নোট বসানোর পর একসাথে পুরো গানটা বাজবে। নিঃসন্দেহে নরমাল ইন্টারেক্টিভভাবে পিয়ানো শেখাটা সহজ, এবং একই কথা প্রোগ্রামিং এর জন্যও প্রযোজ্য। সেজন্য আমি তোমাকে বলব, একটা ল্যাঙ্গুয়েজ খুঁজে বের করো যেটা ইন্টারেক্টিভ মোডে চালানো যায় এবং সেটা ব্যবহার করো।

এই জিনিসগুলো বিবেচনা করলে প্রথম প্রোগ্রামিং ল্যাঙ্গুয়েজের জন্য আমার রিকোমেন্ডেশন হবে **পাইথন** অথবা **স্কিম**। কিন্তু তোমার ক্ষেত্রে সবকিছু অন্যরকম হতেও পারে, হয়তো তোমার জন্য এর চে' ভালো অল্টারনেটিভ আছে। যেমন ধরো, তোমার বয়স যদি এক সংখ্যার হয়, তুমি হয়তো **অ্যালিস** বা **স্কুইক** (বড়দের জন্যও স্কুইক বেশ উপভোগ্য হতে পারে)। আসল ব্যাপার হচ্ছে মনস্থির করা আর সেই ল্যাঙ্গুয়েজ শেখা শুরু করা।

৪.২ বই এবং অন্যান্য রিসোর্স

আমাকে অনেকে জিজ্ঞেস করেছে কোন বই আর ওয়েবপেইজ থেকে তারা শিখতে পারে। আমি বরাবরের মতোই বলছি, "পুঁথিগত বিদ্যা যথেষ্ট নহে" কিন্তু তারপরও আমি এগুলো সুপারিশ করবো -

- স্কিম: **Structure and Interpretation of Computer Programs** (অ্যাবেলসন এবং সুসম্যান) হয়তো কম্পিউটার বিজ্ঞান শেখার জন্য সবচে' সেরা বই। আর একই সাথে এটা প্রোগ্রামিংও শেখায় কম্পিউটার সায়েন্স শেখার একটা উপায় হিসেবে। তুমি **অনলাইন ভিডিও লেকচার** দেখতে পারো এই বইটার আর একই সাথে **অনলাইনে পুরো বইটা** পড়তেও পারো। এটা আসলে খুব সহজ বই না পড়ার জন্য এবং হয়তো অনেক মানুষ স্বাচ্ছন্দ্যবোধ করবে না এটার সাথে কারণ এটা বেশ চ্যালেঞ্জিং।
- স্কিম: **How to Design Programs** (ফেলেইসন এবং অন্যন্য) হচ্ছে প্রোগ্রাম এলিগ্যান্ট আর ফাংশানালভাবে ডিজাইন করার উপর আমার পড়া সেরা বই গুলোর একটা।
- পাইথন: **Python Programming: An Intro to CS** (জেলে) খুব ভালো একটা ইন্ট্রো হতে পারে পাইথন শেখার জন্য।
- পাইথন: **Python.org** এ বেশ কিছু **অনলাইন টিউটোরিয়াল** আছে।
- Oz: **Concepts, Techniques, and Models of Computer Programming** (ভ্যান রয় এবং হ্যারিডি) কে অনেকে আধুনিক সময়ের অ্যাবেলসন এবং সুসম্যান হিসেবে বিবেচনা করে। এটা অনেকটা প্রোগ্রামিং এর সব বড় বড় আইডিয়ার উপর প্রমোদভ্রমণের মতো এবং এটা অনেক বড় রেঞ্জের কনসেপ্ট কাভার করে। হয়তো অ্যাবেলস এবং সুসম্যান পড়ার চেয়ে সহজ হবে এটা পড়া। এটা Oz বলে একটা ভাষা ব্যবহার করে, যেটা হয়তো খুব বিখ্যাত না কিন্তু সেটা অন্য প্রোগ্রামিং ল্যাঙ্গুয়েজ শেখার জন্য চমৎকার ভিত হতে পারে।

কনটেস্ট্যান্টদের জন্য বেসিক সিপিপি

ইকরাম মাহমুদ, ঢাকা বিশ্ববিদ্যালয়, গুগল

প্রথম অংশ - বেসিক সি++

আমরা যারা কন্টেস্ট প্রোগ্রামার, আমরা যতটা না একজন প্রোগ্রামার তার চে' অনেক বেশি হচ্ছি একজন প্রবলেম সলভার। আমরা কিন্তু ঠিক প্রথাগত কম্পিউটার সায়েন্টিস্ট ও নই, যারা মোটামুটি একটা কঠিন প্রবলেম পৃথিবীর কোণাকানচি থেকে খুঁজে বের করে, সেটার পেছনে একটা বছর ধাই করে উড়িয়ে দেবে, তারপর গালটা অনেকক্ষণ চুলকায়ে টুলকায়ে মনে করতে বসবে শেষ কবে দাঁড়ি কাটার সৌভাগ্য হয়েছিল। আবার আমরা ইনডাস্ট্রি প্রোগ্রামারদের মতও নই – যারা মোটামুটি একমাস ধরে একই চুইংগাম চাবাতে চাবাতে একটা বোরিং কোড অলস হাতে বসে বসে লিখতে থাকে। কোড লিখতে লিখতে ঘুমাই পড়ে, আবার ঘুম থেকে উঠে আবার একই কোড লিখতে থাকে।

আমরা কন্টেস্টে করি অনেক বেশি চাপের মধ্যে, আমাদের প্রবলেম পড়ে বুঝতে হয় কি চাওয়া হচ্ছে, আমাদের অ্যালগরিদম ডিজাইন করতে হয়, আমাদের একটা কোড ডিজাইন করতে হয়, এবং আমরা সেটা করি মোটামুটি কোন ধরণের ফ্লোচার্ট হাবিজাবি না এঁকেই। এবং তারপর আমাদের কোডগুলো বসে বসে আমাদের ডিবাগও করতে হয়, টেস্টিং ও করতে হয়। তো ব্যাপারটা দাড়াচ্ছে, আমরা প্রবলেম সলভার, আমরা কম্পিউটার সায়েন্টিস্ট, আমরা কোডার, আমরা সফটওয়্যার আর্কিটেক্ট, আমরা ডিবাগার – কিন্তু আমরা পুরোপুরি এগুলোর কিছুই নই। কিন্তু একই সাথে আমরা এগুলোর সবকিছু।

পেতর মিত্রিচেভ বলতো, কন্টেস্ট প্রোগ্রামাররা হচ্ছে ফরমুলা ওয়ান রেসের ড্রাইভারদের মত। যে লোকটা একটা ট্রাক চালায়, সেও একটা ড্রাইভার, আর যে ফরমুলা ওয়ানে গাড়ি চালায় সেও ড্রাইভার, কিন্তু দুটার মধ্যে আকাশ পাতাল পার্থক্য। আমরা ভাবতেই পারি ফরমুলা ওয়ানে একটা গাড়ি নিয়ে ফালাফালি করার মধ্যে এমন কি আছে, বরং ট্রাক চালানোই অনেক কাজের। কিন্তু দুইটার মধ্যে ঠিক এভাবে তুলনা করা যায় না। যারা একবার ফরমুলা ওয়ানে গাড়ি ছোটানো শুরু করে, শুধু তারাই বুঝতে পারে এক্সাইটমেন্টটা কেমন, এবং সেটা কতটা আলাদা একটা বোকাসোকা হাইওয়েতে অন্ধকার রাতে তারা দেখতে দেখতে আর ঘুমাতে ঘুমাতে একটা স্টুপিড ট্রাক চালানোর সাথে।

এই লেখাটাতে আমি সি++ নিয়ে লিখবো। আর লিখবো কন্টেস্ট প্রোগ্রামারদের ঠিক কতটুকু সি++ জানা দরকার সেটা নিয়ে, আর ঠিক সেইটুকুই। কন্টেস্ট প্রোগ্রামাররা আর সবার মতো না, যে তাদেরকে একটা মোটাসোটা স্বাস্থ্যবান সি++ এর বই ধরিয়ে দিয়ে "নাচো" টাইপের কিছু একটা বলে আমি উধাও হয়ে যাবো।

আর STL নিয়ে যা কিছু লিখবো তার বেশিরভাগ জুড়ে থাকবে ডাটা স্ট্রাকচার। আসলে কি, কম্পিউটারে আমরা প্রচুর ডাটা জমা করি। এখন ডাটাগুলোকে কম্পিউটারে কিভাবে রাখলে আমাদের কাজ করতে সুবিধা হবে, আমরা যদি সেটা ভেবে সেভাবে ডাটাগুলোকে সাজিয়ে রাখি, সেই বুদ্ধিমানের মত করে ডাটা সাজানোটাকেই আসলে ডাটা স্ট্রাকচার বলে।

একজন নতুন কন্টেস্ট্যান্টের কাছে ডাটা স্ট্রাকচার নিয়ে লাফালাফি করাটা খুব অদ্ভূত দেখাবে। এবং আমি যদি পুরোপুরি সবকিছুই ব্যাক্ষা করতে যাই, যে কোনটা কেন দরকার, কিভাবে দরকার, তাহলে এই লেখাটা বিশাল হয়ে যাবে আর আমার আঙ্গুল ব্যাথা করবে এটা লিখতে। তো আমি সব লিখে যাচ্ছি, যাতে তুমি শুধু জিনিসগুলো জেনে রাখতে পারো, কোন না কোন দিন হঠাৎ দেখবে একটা প্রবলেম সলভ করতে তোমার কিছু জানা ডাটা স্ট্রাকচার ব্যবহার করতে হচ্ছে! :)

প্রথম অংশ - সি++ নিয়ে বকরবকর

স্ট্রাকচার

আমাদের মাঝে মাঝেই একসাথে অনেকগুলো ডাটা রাখতে হয়। ধরো, তোমার ফোনবুকে প্রতিটা নামের সাথে একটা করে নাম্বার আছে। খুব সহজ। সমস্যাটা হয়ে গেলো, পৃথিবী অনেকটুকু এগিয়ে গেছে, এখন আর কেউ বাটন গুতোগুতি করে ফোন করে না, সবাই ধাই ধাই করে মেইল করে। তো তোমার ফোন নাম্বারের পাশাপাশি মেইল অ্যাডরেসও রাখার দরকার হচ্ছে। তো ব্যাপারটা কি দাঁড়াচ্ছে, তুমি একই সাথে দুইটা ডাটা রাখছো।

আমরা এই জিনিসটাকে বলি স্ট্রাকচার। এটা লিখি অনেকটা এভাবে।

```
struct data {
  char name[20];
  int number, email_num;
};
```

অবশ্যই ইমেইল আইডি ইনটেজার হয় না - আমি শুধু বোঝানোর লিখছি।
তো আগে আমাদের কিছু ডাটা টাইপ ছিল int, char, double এই ধরনের। আমরা এইমাত্র আমাদের নিজেদের জন্য একটা ডাটা টাইপ তৈরী করলাম, যার নাম হচ্ছে data । ডাটা হচ্ছে একটা মিষ্টির প্যাকেট যার মধ্যে বিভিন্ন ধরণের অনেকগুলো মিষ্টি আছে।এভাবে চিন্তা করলে যদি দেখো তোমার খুব মিষ্টি খেতে ইচ্ছে করছে, তাহলে চিন্তা করো, data হচ্ছে একটা বাক্স যেটার ভিতর আরো কিছু বাক্স(char, int) আছে।

মাঝে মাঝে আমাদের একই জিনিস বারবার ব্যবহার করতে হয়। ধরো, তোমার প্রতিদিন কিছু নতুন মানুষের সাথে দেখা হচ্ছে, ওরা হচ্ছে তোমার নতুন ডাটা - কিন্তু নতুন মানুষগুলোই তোমার সবকিছু নয়, তোমার অনেক কাজ আছে করার মতো আর অনেক অনেক কিছু। তো তোমার দিনটা অনেকটা এরকম কিছু একটা হবে।

```
struct day {
  data new_people[100];
  int total_new_people;
  worklist todo[100];
};
```

যেখানে worklist হয়তো আরেক টাইপের ডাটা। day হলো একটা বাক্স যেটার ভিতরে data টাইপের কিছু বাক্সও আছে।

ব্যাপারটা কঠিন লাগছে? হুমম, এটার নামই হলো অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং। যারা প্রথম প্রথম এটা বুঝতে পারে, তারা তার পরের মোটামুটি কয়েক সপ্তাহ যেখানে সেখানে এটা নিয়ে আঁতলামি করতে খুব ভালোবাসে। তো জিনিসটা কঠিন লাগাটাই স্বাভাবিক।

কিন্তু আমার মনে হয় না, জিনিসটা একটুও কঠিন। জিনিসটা ফালতু রকমের সোজা। আমরা একটা বাক্সের ভিতর আরেকটা বাক্স ভরতেই পারি, আমাদের যত ইচ্ছে খুশি, ততগুলো বাক্সের ভিতর ততগুলো করে বাক্স ভরে, তারপর ওই বাক্সগুলোর ভিতর আরো গাদা গাদা বাক্স ভরতেই পারি। তারপর যখন মনে হবে "উরি বাবা, পাগল হয়ে যাচ্ছি", তখন উইনঅ্যাম্প খুলে অর্ণবের গান শুনতেই পারি, (যে অলরেডি অবজেক্ট অরিয়েন্টেড প্রোগ্রামিং পড়তে গিয়ে পাগল হয়ে গেছে)

বাক্সে বাক্সে বন্দি বাক্স
বাক্সে বাক্সে বন্দি বাসা,
বাক্স নিয়ে টাক show করা,
বাক্সতে সব স্বপ্ন আশা।

অর্ণবের মত একটা ননকোডার মানুষ যদি এই জিনিস বুঝতে পারে, আমরা কেন পারবো না? আরে, আমরা হলাম গিয়ে কন্টেস্ট প্রোগ্রামার, এলিট ক্লাসের মানুষজন!

বেশতো, মূল ব্যাপারটা কিন্তু এরকমই। কিন্তু আরেকটু ভেবে দেখো, প্রতিদিন কি তুমি বসে বসে মাথার ভিতর খালি ডাটাই ঢুকাও, নাকি কিছু করোও? যদি কিছু করো, তবে সেটা অবশ্যই কিছু কাজ, তাই না? হয়তো তুমি ক্লাসে বসে বসে ক্লাসওয়ার্কও করো, হয়তো প্রতিদিন তোমার মিস তোমাকে এক গাদা অংক দেয়, তুমি বসে বসে যোগ করো। তো তখন ব্যাপারটা কি হচ্ছে

```
struct day {
  data new_people[100];
  int total_new_people;
  worklist todo[100];

  void charm_your_miss() {
        while( true ) do_math();
        while( true ) smile();
  }

  void do_math() {
        int a, b;
        scanf("%d %d",&a,&b);
        printf("%d\n", a+b );
  }

  void smile() {
        printf(" :) ");
  }
};
```

এটা হচ্ছে তোমার দিনেরই একটা অংশ, তাইনা? ডাটার পাশাপাশি তোমাকে কিছু কাজও রাখতে হচ্ছে। নাইলে আসলে মিসকে চার্ম করা যাচ্ছে না, এমনকি হাসাও যাচ্ছে না।

এখন যদি আমার শুধু একটা স্ট্রাকচারের কোন একটা ডাটা অ্যাকসেস করা লাগে। আমরা সেটা করি ডট অপারেটর দিয়ে। ধরো আমার শুধু দরকার total_new_people কে। সেটাকে ধরে আমি শুণ্য করে দিবো। তাহলে আমার মেইন এর ভিতর কোডটা হবে এমন।

```
int main() {
  day a_bright_new_day, mora_day;
  mora_day.total_new_people = 0;
  return 0; // already more gesi! :(
}
```

আচ্ছা, এখন ধরো, আমার দিনটা খুব সুন্দর, আমার মিসকে চার্ম করা লাগতেসে না, কারণ স্কুল ছুটি। তো আমি কি করবো, আমি সারাদিন বসে বসে বাবল ফুলাবো আর হাসবো। তো আমি একটা কাজ করবো, হাসবো। (বাবল ফুলানো অবশ্যই কোন কাজের কাজ না, তাই না? ;)) তখন আমি কাজটাও করবো একই ভাবে। একটা ডট অপারেটর ধরে হাসি মেরে দিবো।

```
int main() {
  day a_bright_new_day, mora_day;
  while( true ) a_bright_new_day.smile();
  return 0; // are baba, din to shesh hobei!
}
```

তো এই হলো ঘটনা।

এখন সমস্যা হলো, প্রতিদিন কতকিছুই ঘটতে পারে। তার উপর ছেলেপুলে বড় হচ্ছে। এখন সবকিছু তো আর সবাইকে বলা যায় না, তো আমরা আসলে সব ডাটা এমন ভাবে রাখতে পারবো না, যে আজাইরা লোকজন এসে বলবে, "চিচিং ফাঁক! এই নাও ডট অপারেটর", তারপর ডট অপারেটর দিয়ে আমাদের সব ডাটা মেরে দিবে।

এই জিনিসটা এড়ানোর জন্য আমরা একটা কি-ওয়ার্ড ব্যবহার করি। চিচিং ফাঁক এর ঠিক উল্টা। এটা হলো - private। তোমার প্রাইভেট জিনিসপাতিতে কারো সাধ্য নাই হাতাহাতি করার। ;) তো জিনিসটা তখন লিখে এভাবে।

```
struct day {
  data new_people[100];
  int total_new_people;
  worklist todo[100];

  private:
        sweetthings bolbo_keno[100];
```

```
};
```

জিনিসপাতি পাবলিক করতে চাইলেও কোন সমস্যা নাই। পাবলিক করে দিলেই হবে। সেটা লিখতে হবে এভাবে।

```
struct day {
  data new_people[100];
  int total_new_people;
  worklist todo[100];

  public:
        sweetthings bolboi_to[100];

};
```

এবার একটু জ্ঞান কপচাই। সি++ এ, একটা ফাউ জিনিস আছে, সেটার নাম হলো ক্লাস। ক্লাস একটু মুরুব্বি টাইপের জিনিস। একইভাবেই লিখেও যদিও তারপরও।

```
class day {
  data new_people[100];
  int total_new_people;
  worklist todo[100];

  public:
        sweetthings bolbo_na_vabsilam[100];

};
```

ক্লাসের ব্যাপারটা হলো, তুমি যদি কিছু না বলে দাও ক্লাস সম্বন্ধে তাইলে ক্লাসের সব ডাটা প্রাইভেট থাকবে। আর স্ট্রাকচারের ক্ষেত্রে ঠিক উল্টা, ওর সব কিছুই পাবলিক, বলে না দিলে। এইজন্যই বললাম, ক্লাস একটু মুরুব্বি টাইপের জিনিস, বেশি ভাবের উপর থাকে।

ফাংশন ওভারলোড

মাঝে মাঝেই আমাদের একই কাজ করতে হয়, অনেক আলাদা আলাদা ভাবে। ধরো নরমাল হাসির ব্যাপারটাই, আমরা একেকজনের সামনে একেকভাবে হাসি। হেডমাস্টারের সামনে গিয়ে খ্যাঁক খ্যাঁক করে হাসতে হাসতে গড়াই পড়ে যাই না আমরা। আবার বন্ধুদের সাথে মুখ টিপে টিপে লাজুক লাজুক হাসিও আমরা দেই না। তো ব্যাপারটা একই কাজ কিন্তু ভিন্ন ভিন্ন জায়গায় ভিন্ন ভিন্ন রকমের কাজ করতে হচ্ছে।

একটা সহজ উদাহরণ হচ্ছে অ্যাবসলুট ভ্যালু নেয়া। কোন সংখ্যার শুধু মান নেয়াটাকে বলে অ্যাবসলুট ভ্যালু নেয়া। যেমন -২০ এর অ্যাবসলুট ভ্যালু হলো ২০। মাইনাস উড়ে গেছে।

```
int absolute_int( int x ) {
  if( x < 0 ) return -x;
  else return x;
}
```

তো ধরো আমার long long এও একই কাজ করা লাগবে। তাইলে আমি লিখবো

```
long long absolute_ll( long long x ) {
  if( x < 0 ) return -x;
  else return x;
}
```

সি++ এ দুইটা আলাদা নাম লেখা লাগে না। এরকম দুইটা একই নামের ফাংশন লিখে দিলেই হয়। সে নিজে নিজে বুঝে নিবে তোমার ডাটা টাইপ দেখে যে তুমি মুচকি হাসি দিতে চাচ্ছো, না খ্যাঁক খ্যাঁক করে হাসতে চাচ্ছো।

```
long long absolute( long long x ) {
  if( x < 0 ) return -x;
  else return x;
}

int absolute( int x ) {
  if( x < 0 ) return -x;
  else return x;
}
```

হেডার ফাইল

সি++ এ নেমস্পেস ব্যবহার করলে হেডার লিখার জন্য .h মারা লাগে না। সেটা যদি সি++ এর হয়, যেমন algorithm, তাহলে এটা লেখা হবে এভাবে।

```
#include <algorithm>
using namespace std;
```

কিন্তু সেটা যদি সি এর কিছু একটা হয়, আমরা শুরুতে c লিখে দিবো। যেমন stdio.h লেখা হয় এভাবে।

```
#include <cstdio>
using namespace std;
```

সত্যি কথা আমি এসবের কিছুই করি না। আমার একটা জায়গায় যা কিছু মনে পড়ে, যার যার কথা মনে পড়ে, সবাইকে একসাথে লিখে রেখেছি। জাস্ট ধরে কথা বার্তা ছাড়াই একটা কপি মারি।

```
#include<cstdio>
#include<sstream>
#include<cstdlib>
#include<cctype>
#include<cmath>
#include<algorithm>
#include<set>
#include<queue>
#include<stack>
#include<list>
#include<iostream>
#include<fstream>
#include<numeric>
#include<string>
#include<vector>
#include<cstring>
#include<map>
#include<iterator>
using namespace std;
```

এই হলো আমার হেডারদের কাহিনী।

কিন কাউট!

আমি অবশ্য উচ্চারণ করি সি-ইন, সি-আউট।
তুমি যদি একজন সি কোডার হও, তুমি অবশ্যই scanf দিয়ে ইনপুট নাও, আর printf দিয়ে আউটপুট মারো। খুব স্বাভাবিক ব্যাপার, আরে এতে লজ্জা পাওয়ার কিছু নেই, সবাই তাই করে! কিন্তু ব্যাপারটা হলো, সি তে তোমাকে বলে দিতে হয় তুমি কি ধরণের ডাটা নিয়ে খেলছো, মনে রাখতে হয় %lld, %lu, %c, %s, %d, %lf এইরকম বিচ্ছিরি দেখতে অনেক অনেক কিছু।

সিপিপিতে তুমি শুধু একটা সি-ইন মেরে দিবা, কিচ্ছু মনে রাখা লাগবে না।
যেমন এটা হলো একটা সি কোড

```
int main() {
  int murgi;
```

```
 double water;
 long long bank_balance;

 scanf("%d %lf %lld",&murgi, &water, &bank_balance);
 printf("%d %lf %lld",murgi, water, bank_balance);

return 0;

}
```

একই জিনিস সি-ইন সি-আউট দিয়ে লিখলে এরকম হবে।

```
int main() {
 int murgi;
 double water;
 long long bank_balance;

 cin  >> murgi >> water >> bank_balance;
 cout << murgi << water << bank_balance;

 return 0; // ki ase jibone?
}
```

খালি একটাই সমস্যা। এরা একটু স্লো। কাম কাজ একটু বেশি ধীরে সুস্থে করে।

টেম্প্লেট

টেম্প্লেট কি জিনিস সেটা যদি তুমি বুঝতেই চাও, এই কোডটায় একটা উঁকি মারো। ভয় পেয়েছো? এটা হলো পৃথিবীর এখনকার এক নাম্বার কোডারটার কোড।

```
template<class T> inline T gcd(T a,T b)
```

আচ্ছা, এটা মোটামুটি নির্দোষই লাগছে, তাই না?
আমরা একটু আগে ফাংশন ওভারলোডিং দেখলাম না? ওই জিনিসটাই আরেকভাবে করা যায়, টেম্প্লেট ব্যবহার করে। যেমন ওই অ্যাবসলুট ভ্যালুর কোডটা টেম্প্লেট দিয়ে লিখলে এরকম হতো

```
template<class T> T absolute( T x ) {
 if( x < 0 ) return -x;
 else return x;
}
```

মানে আমার দুই তিনবার একই ফাংশন আলাদা আলাদা করে ডাটাটাইপ পাল্টে লিখতে হতো না। আমি টেম্প্লেটটা দেখিয়ে নিলাম, এই জন্য যে, আমরা একটু পরে STL নিয়ে আঁতলামি শুরু করবো, STL এর সবকিছুতে টেম্প্লেট ব্যবহার করে লেখা, যাতে তুমি ইচ্ছামতো ক্লাস/স্ট্রাকচার ব্যবহার করে নিজের ডাটাটাইপ ব্যবহার করতে পারো, আর সেটা নিয়ে ইচ্ছামতো নাচানাচিও যেন করতে পারো।

দ্বিতীয় অংশ - স্ট্যান্ডার্ড টেম্প্লেট লাইব্রেরী

আগের অংশে শেষে আমরা টেম্প্লেট দেখলাম। টেম্প্লেট করে কি, যদি আমি ডিফাইন করে দেই আমার কাজটা কি, সে যেকোন ধরণের ডাটা টাইপ নিয়ে ওই কাজটা করতে পারবে। সি++ এ একটা বিশাল লাইব্রেরী আছে, যার কোডগুলো যেকোন ধরণের ডাটার জন্য কাজ করতে পারে। এই টেম্প্লেট লাইব্রেরীর সবচে' স্ট্যান্ডার্ড ভারশনটার নামই স্ট্যান্ডার্ড টেম্প্লেট লাইব্রেরী, ওরফে STL।

STL হল একটা বেশ বড়সড় একটা লাইব্রেরী। মোটামুটি বেশিরভাগ ডাটা স্ট্রাকচার আর হাবিজাবি এটার মধ্যে লিখে রাখা আছে, তোমাকে শুধু জানতে হবে সেটা তুমি কিভাবে ব্যবহার করবে।

ভেক্টর

মাঝে মাঝে এমন হয় - আমাদের একটা 2D অ্যারে দরকার, যেটায় মোটামুটি প্রতিটায় সর্বোচ্চ ১০০০০ টা ডাটা রাখতে হবে, আর প্রতিটা ডাটায় সর্বোচ্চ ১০০০০টা করে ডাটা রাখা লাগবে। কিন্তু আমাকে এটাও বলা আছে যে সর্বোচ্চ ১০০০০০ টা ডাটা থাকতে পারে।

খুব সাধারণভাবে যেটা মাথায় আসে, সেটা হচ্ছে এরকম কিছু একটা

```
int array[10000][10000];
```

তাই না? এটা কিন্তু বেশ বড়সড় একটা অ্যারে। আমার কম্পিউটার মাথা ঘুরে পড়ে যাবে তাকে এই পরিমান মেমরি অ্যালোকেট করতে বললে, কিন্তু আমার আসলে এত বেশি জায়গা লাগছে না, কারণ আমাকে বলেই দেয়া হয়েয়ে ডাটা সবমিলে সর্বোচ্চ ১০০০০০ টা থাকে পারে।

এধরণের সময়, আমরা ডাইনামিক মেমরি অ্যালোকেট করি - ঠিক যতটুকু মেমরি দরকার ঠিক ততটুকুই নেই। যেটা ম্যানুয়ালি করা বেশ ঝক্কি, আর সেটায় মেমরি পরিষ্কারও করে দিতে হয় কাজ শেষে, নইলে সব ডাটা জমতে জমতে কম্পিউটারের গলা চিপে ধরে।

ভেক্টর হলো একটা অ্যারে, যেটায় ডাইনামিকালি জিনিসপাতি ঢুকিয়ে রাখা যায়। মানে, এটাও একটা অ্যারে, কিন্তু সেটা ঠিক ততটুকু মেমরি খায়, যতটুকু খাওয়া লাগে।

ভেক্টর ডিক্লেয়ার করে এভাবে

```
vector< int > array;
```

তুমি যদি অন্য কোন টাইপের ডাটা নিতে চাও তাইলে int এর জায়গায় সেই ডাটার নাম লিখতে হবে। যেমন এটা আরো কিছু অ্যারে।

```
vector< double > water;
vector< long long > balance;
vector< char > characters;
vector< day > diary;
```

ভেক্টরে কোন ডাটা রাখতে হলে, সেই ভেক্টরের শেষে ডাটাটাকে পুশ করতে হয়।

```
array.push_back( 100 );
```

আর ভেক্টরে কটা ডাটা আছে সেটা আমরা জানতে পারি .size() ফাংশনকে কল করে। যেমন ধরো, আমি একটা ভেক্টরে কিছু ইন্টেজার ঢুকাবো, তারপর সবাইকে প্রিন্ট করবো, সেটার কোড হবে এরকম।

```
int main() {
  vector< int > v;
  v.push_back( 1 );
  v.push_back( 2 );
  v.push_back( 3 );
  v.push_back( 4 );

  for(int i=0; i<v.size(); i++) cout << v[i] << endl;

  return 0;
}
```

বাকি সব কিছুতে ভেক্টরকে সাধারণ অ্যারের মত ব্যবহার করা যায়। যেমন আমি 0th এলিমেন্টটা পাল্টে দিতে পারি v[0] = 10000 লিখে। আরেকটা মজা হচ্ছে আমরা অ্যারেতে ধাম করে সরাসরি কপি করতে পারি না । কিন্তু ভেক্টরে সেটা করা যায়।

```
int main() {
  vector< int > v, t;
  v.push_back( 1 );
  v.push_back( 2 );
  v.push_back( 3 );
  v.push_back( 4 );

  t = v; // copying
  for(int i=0; i<t.size(); i++) cout << t[i] << endl;

  return 0;
```

```
}
```

ভেক্টরে যদি আমি 2D ডাটা রাখতে চাই তাহলে সেটা দুভাবে করা যায়। আমি প্রথমটা প্রেফার করি, পরেরটা দেখতে আমার ভয় ভয় লাগে। কিন্তু মাঝে মাঝে কোন পথ থাকে না সেটা লেখা ছাড়া।

```
vector< int > v[100];
vector< vector< int > > v;
vector< vector< vector< int > > > v; // 3 dimensional
```

একটা জিনিসে একটা সাবধান থেকো, vector<vector<int>> v; এভাবে লিখলে >> এর জন্য কিছু কম্পাইলর কিন্তু এরর মারে।

স্ট্রিং

স্ট্রিং হচ্ছে মজার একটা ডাটা স্ট্রাকচার। মোটামুটি এর কাজ অনেকটা ক্যারেক্টার অ্যারের মতই। কিন্তু এটা ব্যবহার করা বেশ সহজ। যেমন নিচে কিছু স্ট্রিং টাইপের জিনিসপাতির কাজ দেখিয়ে দিলাম।

```
int main() {
  string a, b, c;
  a = "this is a string"; // easy assigning
  b = a; // copy hoye gelo! :O
  c = a + b // c te rakhlam a ar b er concatation
  cout << c << endl; // print korlam
  printf("%s\n", c.c_str() ); // printf diyei korlam na hoy

  cout << c.size() << endl; // length print korlam
  for(int i=0; i<c.size(); i++) cout << c[i] ;
// ekta ekta kore character print korlam

  return 0;
}
```

তুমি যদি এখন স্ট্রিং এর ভেক্টর রাখতে চাও তাহলে সেটাকে ডিক্লেয়ার করতে হবে এভাবে।

```
vector< string > vs;
```

সহজ না?

স্ট্যাক

ধরো, তোমার মা একগাদা প্লেট ধুতে নিয়ে যাচ্ছে খাওয়ার টেবিল থেকে। সবার পরে যেটা রাখা হবে, সেই প্লেটটাকে কিন্তু সবার উপরে রাখা হবে, আর সেটাই কিন্তু সবার আগে ধোয়া হবে।

এই জিনিসটাকে বলে স্ট্যাক। মানে আমরা সবার পরে যাকে প্রসেসিং করতে ঢুকাচ্ছি তাকে যদি আগে প্রসেসিং করি তাহলে সেটাই স্ট্যাক। STL এ স্ট্যাক ব্যবহার করতে হয় এভাবে।

```
stack< int > st;
    st.push( 100 ); // inserting 100
    st.push( 101 ); // inserting 101
    st.push( 102 ); // inserting 102

    while( !st.empty() ) {
    cout << st.top() << endl; // printing the top
    st.pop(); // removing that one
    }
```

কিউ

ধরো তুমি বাসের টিকেট কিনে লাইনে দাঁড়িয়ে আছো। এখন বাসে ওঠাটা হচ্ছে আমার কাজ(প্রসেসিং)। কাকে আগে বাসে উঠতে দিবে? যে সবার আগে এসেছে, তাকে। এটাকে বলে কিউ – যে সবার আগে এসেছে তাকে আগে প্রসেস করা।

```
    queue< int > q;
    q.push( 100 ); // inserting 100
    q.push( 101 ); // inserting 101
    q.push( 102 ); // inserting 102

    while( !q.empty() ) {
    cout << q.front() << endl; // printing the front
    q.pop(); // removing that one
    }
```

প্রায়োরিটি কিউ

আমাদের পাড়ার মুচির প্রতিদিন একগাদা কাজ আসে। সে করে কি, সবচে' বেশি পয়সা পাওয়া যাবে যেই কাজে সেই কাজগুলো সবার আগে করে ফেলে। সে প্রায়োরিটি তাদেরকেই বেশি দেয় যাদের কাজে বেশি পয়সা পাওয়া যাবে।

এটাও এক ধরণের কিউ শুধু পার্থক্য হচ্ছে যার দাম যত বেশি তাকে তত আগে প্রসেস করা হচ্ছে।

```
    priority_queue< int > q;
    q.push( 10230 ); // inserting 10230
    q.push( 1021 ); // inserting 1021
```

```
q.push( 102322 ); // inserting 102322

while( !q.empty() ) {
cout << q.top() << endl; // printing the top
q.pop(); // removing that one
}
```

ইটারেটর

ইটারেটার হলো অনেকটা সি এর পয়েন্টারের মত একটা জিনিস। ইটারেটর আসলে পরে কাজে লাগবে, কারণ অনেক জায়গায়ই STL এর ফাংশনগুলো একটা অ্যাডরেস পাঠায়, যে আমি যেই ডাটাটাকে খুঁজছি, সেটা ঠিক কোথায় আছে।

ইটারেটর ডিক্লেয়ার করে এইভাবে

```
vector< int > :: iterator i;
vector< double > :: iterator j;
```

আর ফর লুপ দিয়ে একটা ভেক্টরের প্রথম থেকে শেষ পর্যন্ত সব এলিমেন্টের গলা কাটতে চাই তাহলে সেটা লিখতে হবে এভাবে।

```
vector< int > v; v.pb( 1 ); v.pb( 2 ); v.pb( 3 );
vector< int > :: iterator i;
for( i = v.begin(); i < v.end(); i++ ) {
printf("%d\n", *i);
// ei khane gola kato!
}
```

সর্ট

ধরো আমার কাছে কিছু নাম্বার আছে, আমি সেগুলোকে ছোট থেকে বড়তে সাজাবো, বা উল্টো কাজটা করবো, বড় থেকে ছোটতে সাজাবো। এই কাজটাকে বলে সর্ট করা। যদি তুমি সর্ট করার নিয়ে পড়াশুনা করে ফাটাই ফেলতে চাও তাইলে এইখানে একটু ঢু মারো।

STL এ সর্ট করা খুব সহজ। ধরো আমার একটা ভেক্টর v আছে, সেটা আমি সর্ট করবো। তাহলো আমার শুধু লিখতে হবে -

```
sort( v.begin(), v.end() );
```

তাহলে সে ছোট থেকে বড় তে ভেক্টরটাকে সর্ট করে ফেলবে। এখন ধরো আমাকে যদি আরেকটু ঝামেলার কিছু করতে বলে। যেমন ধরো চাচা চৌধুরী তার মেয়ের বিয়ে দিবে, তো সে গেলো ঘটক পাখি ভাইয়ের কাছে। ঘটক পাখি ভাইয়ের কাছে একটা ছেলে মানে, তার নাম-ধাম, তার বংশ, সে কত টাকা কামায়, তার উচ্চতা কতো, আর তার ওজন কত। ছেলেটা সি++ এ কোড করে না জাভাতে কোড করে, সেটা নিয়ে ঘটক পাখি ভাইয়ের কোনই মাথা ব্যাথা নাই। তো সে করলো কি

চাচা চৌধুরীকে শুধু এই কয়টা ডাটাই সাপ্লাই দিলো কয়েকটা বস্তা ভরে। এখন চাচা চৌধুরী পাড়ার প্যান্ট ঢিলা মাস্তানের কাছ থেকে শুনলো তুমি একটা বস প্রোগ্রামার, তো সে এসে তোমাকে বলল, "বাবাজি! আমাকে একটা সফটওয়্যার বানিয়ে দাও, যেটা আমার ডাটাগুলোকে সাজাবে"।

বেশ তো, এখন আমার ডাটাটা হচ্ছে এরকম - (চাচা চৌধুরী আবার বংশ নিয়ে মাথা ঘামায় না)

```
struct data {
  char name[100];
  int height, weight;
  long long income;
};
```

চাচা চৌধুরী যেটা নিয়ে মাথা ঘামায় সেটা হলো পোলার কত টাকা কামাই। যদি দুইটা পোলার সমান কামাই হয়, তাইলে যেই পোলার হাইট ভালো, সেই পোলা লিস্টে আগে থাকবে। আর যদি দুই পোলার হাইট সমান হয় তাইলে যেই পোলার ওজন কম, সেই পোলা আগে থাকবে। আর যদি দুই পোলার ওজন সমান হয়, তাইলে যেই পোলার নাম ছোট সেই পোলা আগে থাকবে।

এখন তোমাকে এই অনুযায়ী সর্ট করে দিতে হবে। আর তুমি যদি বেশি হাংকি পাংকি করো, তাইলে প্যান্ট ঢিলা মাস্তান এসে তোমাকে সাইজ করে দিবে।

এই কাজটা দুই ভাবে করা যায়। সবচে সহজটা হলো একটা কম্পেয়ার ফাংশন লিখে।

```
bool compare( data a, data b ) {
  if( a.income == b.income ) {
        if( a.height == b.height ) {
        if( a.weight == b.weight )
        return strlen( a.name ) < strlen( b.name );
        else return a.weight < b.weight;
        }else return a.height > b.height;
  }else return a.income > b.income;
}
```

এই ফাংশনটা গ্লোবালি ডিক্লেয়ার করে যেখানে তুমি সর্ট করতে চাও সেখানে লিখতে হবে।

```
sort( v.begin(), v.end(), compare );
```

কম্পেয়ার ফাংশনটা রিটার্ন করবে a কি b এর আগে বসবে কি না। আর কিছু না।

সর্ট করার অন্য পথটা হচ্ছে অপারেটর ওভারলোড করে। ধরো, আমরা যখন বলি ২ < ৩ আমরা বুঝে নেই যে ২ হচ্ছে ৩ এর ছোট - মানের দিক দিয়ে। এখন একটা স্ট্রাকচার কখন অন্য আরেকটা স্ট্রাকচারের চেয়ে ছোট হবে? এই জিনিসটা তোমার প্রোগ্রামে ডিফাইন করে দিতে হবে। এখানে খেয়াল করো, ছোট হবার মানে বোঝাচ্ছে সে লিস্টে আগে থাকবে।

আমি যদি একই কাজটা অপারেটর ওভারলোড দিয়ে করতে চাই, সেটা এরকম হবে।

```
struct data {
  char name[100];
  int height, weight;
  long long income;

  bool operator < ( const data& b ) const {
        if( income == b.income ) {
        if( height == b.height ) {
        if( weight == b.weight )
        return strlen( name ) < strlen( b.name );
        else return weight < b.weight;
        }else return height > b.height;
        }else return income > b.income;
  }

};
```

এখানে কিন্তু আমি এই ডাটাটাকেই অন্য আরেকটা ডাটা b এর সাথে তুলনা করছি, সেজন্য আমার আগেরটার মতো a কে লাগছে না।

আর আমার সর্ট এর কমান্ড লিখতে হচ্ছে এইভাবে।

```
sort( v.begin(), v.end() );
```

তোমার যদি ভেক্টর ব্যবহার করতে আপত্তি থাকে, ধরো ভেক্টর দেখলেই হাঁচি আসা শুরু করে, নাক চুলকায় কিংবা এধরণের কিছু, তুমি সাধারণ অ্যারেই ব্যবহার করতে পারো।
ধরো সেক্ষেত্রে অ্যারেটা হবে এরকম -

```
data array[100];
sort( array, array + n );
```

যেখানে n হচ্ছে অ্যারেতে কতগুলো ডাটাকে তুমি সর্ট করতে চাও।

তুমি যদি 3 নাম্বার (0 based)থেকে 10 নাম্বার পর্যন্ত সর্ট করতে চাও লিখো

```
sort( array+3, array+11 );
```

সেট

কোন কিছুর সেট বলতে আসলে বুঝায় শুধু জিনিসগুলোর নাম একবার করে থাকাকে।
যেমন A = { রহিম, করিম, গরু, বিড়াল, করিম, বালিশ, রহিম, করিম } একটা সেট না, কিন্তু
A = { রহিম, করিম, গরু, বিড়াল, বালিশ } একটা সেট।

STL এর সেট করে কি, সেট এ সব ডাটা গুলো একবার করে রাখে, আর ডাটাগুলোকে সর্ট ও করে রাখে। এটা হলো সেট এর কাজ কারবার –

```
set< int > s;
s.insert( 10 ); s.insert( 5 ); s.insert( 9 );

set< int > :: iterator it;
for(it = s.begin(); it != s.end(); it++) {
cout << *it << endl;
}
```

যদি তুমি স্ট্রাকচার টাইপের ডাটা রাখতে চাও সেট এ, শুধু < অপারেটরটা ওভারলোড করে ওকে বলে নিও, যে তুমি ছোট বলতে কি বুঝাচ্ছো। বাকি কাজ ওই করবে।

সেট সাধারণত এধরণের প্রবলেমগুলোতে কাজে লাগে। আমাকে অনেকগুলো সংখ্যা দিয়ে বলল, এখানে ইউনিক কয়টা সংখ্যা আছে। সেক্ষেত্রে আমি খালি একটার পর একটা সংখ্যা ইনপুট নিতে থাকবো তো নিতেই থাকবো, আর সেটে ঢুকাবো তো ঢুকাতেই থাকবো, তারপর খালি সেটের সাইজ প্রিন্ট করে দিবো। কেল্লা ফতেহ!

ম্যাপ

ম্যাপও সেটের মতো একটা জিনিস। কিন্তু ম্যাপ সেটের মত কোন জিনিস একটা রেখে ওই ধরণের বাকি সবাইকে বাইরে ফেলে দেয় না।

তবে এভাবে ভাবার চেয়ে ম্যাপকে আরেকটু সহজভাবে ভাবা যায়। একটা অ্যারের কথা চিন্তা করো, আমরা করি কি অ্যারের একটা ইনডেক্সে ডাটা জমাই না? কেমন হতো, যদি ইনডেক্সটা শুধু সংখ্যা না হয়ে যেকোন কিছু হতে পারতো? ধরো, ১ নম্বর ইনডেক্সে না রেখে, "বাংলাদেশ" নামের ইনডেক্সে ডাটা যদি রাখতে পারতাম? তখন ব্যাপারটা দাঁড়াতো আমাদের একটা ম্যাজিক অ্যারে আছে যেটাই আমরা যেকোন ধরণের ডাটা জমিয়ে রাখতে পারি আমাদের ইচ্ছা মতো যে কোন ধরণের ইনডেক্স দিয়ে।

সহজভাবে ম্যাপকে তুমি এভাবে চিন্তা করতে পারো, ম্যাপ হচ্ছে একটা অ্যারে, যেটার ইনডেক্স যেকোন কিছুই হতে পারে, আর সেটাতে যেটা ইচ্ছে সেটাই রাখা যেতে পারে!

```
map< string, int  > m;
string goru;

while( cin >> goru ) {
if( goru == "moro" ) break;
m[ goru ] ++;
cout << goru <<" ase " << m[ goru ] << " ta :D " << endl;
}
```

এই প্রোগ্রামটা করবে কি, গরুর নাম ইনপুট নিতে থাকবে, আর প্রতিবার বলবে যে ওই জাতের কয়টা গরু আছে। ম্যাপকে অ্যারের মত ধরেই ইনক্রিমেন্ট করা যায়।

অবশ্য তুমি যদি তোমার বানানো কোন স্ট্রাকচার/ক্লাস রাখতে চাও ইনডেক্স হিসেবে, তোমাকে সেটার জন্য < অপারেটরটা ওভারলোড করে দিতে হবে।

স্ট্রিংস্ট্রিম

ধরো কোন শয়তান খুব শখ করে প্রবলেম সেট তৈরী করলো, আমাকে বলল, "তোমাকে একলাইনে যত্তগুলা ইচ্ছা তত্তগুলা করে সংখ্যা দিমু, তুমি আমারে সর্ট কইরা দিবা! মুহাহাহাহা!" তখন কষে একটা চড় মারতে ইচ্ছে করলেও কিছু করার নেই। তোমাকে তাই করতে হবে।

আমরা লাইনের ইনপুট নেই হচ্ছে গেটস দিয়ে।
তো ব্যাপারটা এরকম হবে।

```
char line[1000];
while( gets( line ) ) {
stringstream ss( line ); // initialize kortesi
int num; vector< int > v;
while( ss >> num ) v.push_back( num ); // :P
sort( v.begin(), v.end() );
// print routine
}
```

ss এর পরের হোয়াইল লুপ অংশটা তুমি cin এর মতো করেই ভাবতে পারো! ;) আমি সেভাবেই চিন্তা করি।

পেয়ার

STL এর একটা স্ট্রাকচার বানানো আছে, যার অবস্থা মোটামুটি এইরকম।

```
struct pair {
  int first, second;
};
```

তবে জিনিসটা এমন না, তুমি যেকোন টাইপে কাজ করতে পারো। যেমন এটা যদি আমি STL এর পেয়ার দিয়ে লিখি, জিনিসটা হবে এরকম

```
pair< int, int > p;
```

এই চেহারাটা কি মনে পড়ে? একে কি আগে দেখেছো? হুমম, ম্যাপের স্ট্রাকচারে আরেকবার চোখ বুলাও। ;)

আমরা ইচ্ছে মতো পেয়ার ডিফাইন করতে পারি, যেভাবে ইচ্ছে। ম্যাপের ডাটা টাইপের মতনই!

```
pair< int, int > p;
pair< int, double > x;
pair< double, string > moru;
pair< goru, goru > fau;
```

যা ইচ্ছে!

নেক্সট পারমুটেশন, প্রিভ পারমুটেশন

ধরো হঠাৎ একদিন ঘুম থেকে উঠে দেখলা যে তোমার এগারোটা বাচ্চা এবং কালকে ঈদ আর আজকে তোমার ওদের জন্য ঈদের জামা কিনতে হবে। সমস্যা হচ্ছে, তোমার বউ এরই মধ্যে এগারোটা জামা কিনে ফেলেছে আর আরো সমস্যা হচ্ছে সেটা সে লটারি করে দিয়ে দিয়েছে এবং সেজন্য যাদের যাদের জামা পছন্দ হয়নি তারা কান্নাকাটি করছে। তো তোমার খুব মন খারাপ, তুমি চাও ঈদের দিনের সুখ যাতে সবচে' বেশি হয়। আর তুমি এটাও জানো কোন জামা পড়লে কোন বাচ্চা কতটুকু সুখি হবে। এখন আমাদের সবার সুখের যোগফল ম্যাক্সিমাইজ করতে হবে।

এধরণের প্রবলেমকে বলা হয় কম্প্লিট সার্চ। আমাদের সবগুলো অপশন ট্রাই করতে হবে। ধরো তিনটা বাচ্চার জন্য অল পসিবল ট্রাই করা হচ্ছে এরকম - (জামার নাম্বার দিয়ে)

আবু	গাবু	ডাবু
১	২	৩
১	৩	২
২	১	৩
২	৩	১
৩	১	২
৩	২	১

এখন এভাবে যদি আমি এগারোটা বাচ্চার জন্য ঈদের জামা পড়িয়ে দেখতে চাই আমার খবরই আছে - 11! ভাবে ট্রাই করতে হবে। তো সেই জন্যই আছে STL এর নেক্সট পারমুটেশন

```
vector< int > v;
for(int i=0; i<11; i++) v.push_back( i );

do {
// protitat jama prottekke porai dekho shukh maximize hochche kina
}while( next_permutation( v.begin(), v.end() ) );
```

আমরা ৩ এর জন্য যেভাবে সবগুলো পারমুটেশন জেনারেট করেছি, সেটাই এই নেক্সট পারমুটেশন করবে। খেয়াল কোর যে, নেক্সট পারমুটেশন কিন্তু ঠিক অ্যালফাবেটিকালি পরের পারমুটেশনটাকে নেয়। তুমি যদি সব পারমুটেশন চাও, প্রথমে অবশ্যই অ্যারেটাকে সর্টেড রেখো।

রিভার্স

রিভার্স হচ্ছে একটা কিছুকে ধরে উল্টাই দেয়া।
ধরো আমার একটা ভেক্টর আছে।

```
vector< int > nacho;
reverse( nacho.begin(), nacho.end() );
```

পরের স্টেটমেন্টটা লিখলে, সে নাচোকে উল্টাই দিবে।

তো এই ছিলো ব্যাসিক সি++ আর STL।
কোন প্রশ্ন থাকলে আমাকে জানাও, আমি সেটা এখানে আপডেট করবো। আর একটা জিনিস মাথায় রেখো, তুমি যদি কোন কিছুতে ভালো হতে চাও, তোমার প্র্যাকটিস লাগবে, আর সেটা কেউ শিখিয়ে দিতে পারবে না। তুমি যদি বেশি কোন কিছু ব্যবহার করবে, তুমি তত ভালো হবে সেটাতে।

শুভ সি-প্লাস-প্লাসিং! ;)

শর্টেস্ট পাথের অ্যালগরিদম

ইকরাম মাহমুদ, ঢাকা বিশ্ববিদ্যালয়, গুগল


Smartest path
ছোটরাই ভালো!

আমরা যখন প্রবলেম সলভ করি কম্পিউটার ব্যবহার করে, বেশিরভাগ প্রবলেম হয় অপটিমাইজেশন প্রবলেম। (অপটিমাইজেশন মানে হচ্ছে যদি অনেকগুলো পথ থাকে কিছু একটা করার, আমরা সবচে' সেরা পথটাতে সেটা করবো) ধরা যাক কেউ ঢাকা থেকে সান ফ্রান্সিসকো যাবে, সে করবে কি একটা ট্র্যাভেল এজেন্ট এর কাছে যাবে সবচে' কম খরচের রুটটা বের করে দিতে। ট্র্যাভেল এজেন্ট করবে কি, একটা কম্পিউটার প্রোগ্রামকে জিজ্ঞেস করবে পসিবল রুটগুলো দেখানোর জন্য, আর কম খরচের রুটগুলো দেখানোর জন্য। তারপর সে সেই অনুযায়ী টিকেট কেটে দেবে। এখানে সবচে' সস্তা রুটটা হচ্ছে আমাদের ''অপটিমাল'' পথ।

আমরা বুঝলাম যে ঢাকা থেকে সান ফ্রান্সিসকো যাবে সে কি কি করবে। কিন্তু কম্পিউটার প্রোগ্রামটা করবে টা কি?

শর্টেস্ট পাথ

শর্টেস্ট পাথ এর সংজ্ঞাটা হবে অনেকটা এরকম। আমার যদি অনেকগুলো শহর থাকে, আর শহরগুলোর মধ্যে যদি অনেকগুলো পথ থাকে, (যেই পথগুলোর একেকটার একেকরকমের দৈর্ঘ্য থাকতে পারে - এবং কোন পথে কোন জ্যাম থাকবে না) তাহলে কোন একটা শহর থেকে অন্য আরেকটা শহরে যেই পথ দিয়ে গেলে সবচে' কম দূরত্ব যাওয়া লাগবে সেটা হচ্ছে আমার প্রথম শহরটা থেকে দ্বিতীয় শহরটার শর্টেস্ট পাথ।

যখন আমরা গ্রাফ থিওরীর টার্মিনোলজিতে কথা বলি, তখন আমরা শহরগুলোকে শহর না বলে বলি নোড(node), আর পথগুলো পথ না বলে বলি এজ(edge), আর আমরা এই পুরো ম্যাপটাকে বলে গ্রাফ(graph)। টার্মিনোলজি শেখাটা কাজের, কারণ তাহলে তুমি যেই ছেলেটা চাইনিজ ছাড়া আর কিছুতে কথা বলতে পারে না, কিন্তু গ্রাফ থিওরী জানে, তার সাথেও কিছুক্ষণ গ্রাফ থিওরী নিয়ে পটপট করতে পারবা। আমার প্রথম প্রথম ম্যাপটাকে "গ্রাফ" বলে চিন্তা করতে খুব সমস্যা হতো, কারণ আমরা যেটাকে "গ্রাফ" বলে চিন্তা স্কুল কলেজে, সেটা পুরোপুরি আলাদা জিনিস। কিন্তু একটা সময় তুমি এমনিতেই অভ্যস্ত হয়ে যাবা।

ক্যামনে করবো?

বেশ তো, কিন্তু প্রথমে চিন্তা করো তুমি কিভাবে, কম্পিউটারে ডাটা হিসেবে শহরগুলোর পথগুলোকে রাখবা। সবচে' সহজভাবে জিনিসটা এভাবে করা যায় - আমি একটা 2D অ্যারে রাখি, distance নামের যেখানে distance[i][j] হবে i তম শহর থেকে j তম শহরে যাওয়ার দূরত্ব। তো এই ম্যাট্রিক্সটাকে সাধারণত বলা হয় অ্যাডজাসেন্সি ম্যাট্রিক্স(adjacency matrix)।

```
#define M 100
int distance[M][M];
```

অ্যাডজাসেন্সি ম্যাট্রিক্স এর প্রথম সমস্যা হচ্ছে এটা জায়গা বেশি খায়। ধরো তোমার ২০ ০০০ টা শহর আছে আর তাদের মধ্য ৫০ ০০০ টা পথ আছে। আমার এখানে সবমিলে ডাটা আসলে ৫০ ০০০। কিন্তু তুমি যখন অ্যারে ডিক্লেয়ার করতে যাচ্ছো, তোমার ২০০০০ x ২০০০০ সাইজের অ্যারে ডিক্লেয়ার করতে হচ্ছে, যার বেশিরভাগই তুমি ব্যবহার করছো না। আর তোমার RAM এ হয়তো এত জায়গাও নেই। দ্বিতীয় সমস্যা হচ্ছে, তুমি ঠিক জানো না i এর সাথে কোন কোন j তে পথ আছে। তো তোমার সবগুলো j খুঁজে খুঁজে দেখতে হবে সেখান থেকে পথ আছে কি না, যেটা আরো সমস্যা। কারণ এমন হতে পারে i নাম্বার শহরটার শুধু একটা শহরের সাথে পথ আছে, আর সেটা আছে সবার শেষ ইন্ডেক্সটাতে। তো এই ক্ষেত্রে আমরা বোকার মত সময় নষ্ট করবো পথ খুঁজতে।

সহজ উপায় হচ্ছে আমরা দুটো ভেক্টরের অ্যারে রাখতে পারি। ধরো প্রথমটার নাম হচ্ছে edge, দ্বিতীয়টার নাম হচ্ছে cost।

```
#define M 100
vector< int > edge[M], cost[M];
```

edge[i] তে থাকবে সবগুলো নোড যাদের i এর সাথে পথ আছে, আর cost[i] তে থাকবে, যথাক্রমে সেই পথগুলোর দূরত্ব। মানে ধরো edge[i] এর প্রথম এলিমেন্টটা যদি 23 হয় তাহলে cost[i] এর প্রথম এলিমেন্টটা হবে i থেকে 23 এর দূরত্ব।

প্রথম অ্যালগরিদম – ব্রেডথ ফার্স্ট সার্চ (Breadth First Search - BFS)

তোমার যদি অল্পিকটু রিকার্শন জানা থাকে তুমি DFS চালাই দিতে পারো গ্রাফ এর উপর শর্টেস্ট পাথ বের করার জন্য। কিন্তু রিকার্শন একটা ঝামেলার হয়ে যায় আসলে, প্রচুর ফাংশন কল আর ডিপেন্ডেন্সির কারণে জিনিসটা স্লো হয়ে যায়।

শর্টেস্ট পাথ বের করার একটা সহজ আর কাজের পদ্ধতি হচ্ছে BFS। BFS লেখা বেশ সোজা, মানে কন্টেস্টের সময়ে খুব দ্রুত লিখে ফেলা যায়। আর তুমি যদি হাল্কা পাতলা অপটিমাইজ করতে পারো তাহলে BFS ঝড়ের বেগে উড়বে।

BFS এ আমরা যেটা করি তা হচ্ছে। আমরা একটা কিউ(queue) রাখি, আর একটা দূরত্ব রাখার জন্য অ্যারে রাখি। প্রথমে আমরা ধরে নেই সবার দূরত্ব অসীম, আর শুধু শুরু শহরটার দূরত্ব হচ্ছে শূণ্য। তারপর আমরা কিউতে শুরুর শহরটাকে ঢুকাই।

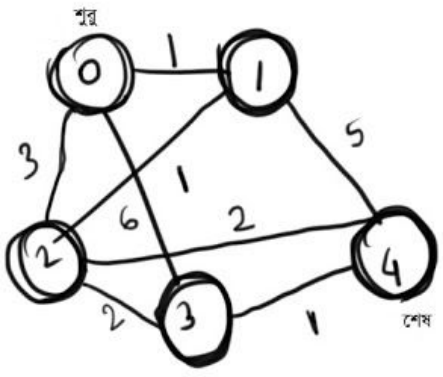
এরপর যতক্ষণ না কিউ খালি হচ্ছে ততক্ষণ আমরা করি কি কিউর প্রথমে যেই শহরটা আছে, সেটাকে বের করে আনি প্রসেসিং এর জন্য। তারপর ওটা থেকে যেই শহরগুলোতে যাবার পথ আছে সেই শহরগুলোতে যদি আমরা এই শহরটা থেকে যাই তাহলে যদি আগের চেয়ে কম সময় লাগে যেতে

আমরা

1. সেই শহরটার নতুন দূরত্ব আপডেট করি
2. নতুন দূরত্ব পাওয়া শহরটাকে প্রসেস করার জন্য কিউতে পুশ করি।

তারপর যতক্ষণ না পর্যন্ত সব শহরগুলোকে প্রসেস হচ্ছে ততক্ষণ ধরে আমরা আপডেট করতে থাকবো। যদি কোন একটা সময় আপডেট আর করা না যায় (মানে, আমরা কোন শহরের জন্যই এমন কোন নতুন পথ পাচ্ছি না যেটা দিয়ে গেলে ওই শহরটাতে বর্তমান পথটার চেয়ে তাড়াতাড়ি যাওয়া যাবে)

ধরো এই গ্রাফটার জন্য



প্রথমে আমরা 0 কে কিউতে পুশ করবো। এখন 0 এর দূরত্ব হচ্ছে 0 আর বাকি সবার দূরত্ব হচ্ছে অসীম।

শহর	দূরত্ব
0	0
1	অসীম
2	অসীম
3	অসীম
4	অসীম

0 থেকে যদি আমি 1 এ যাই, তাহলে 1 এর নতুন দূরত্ব হবে 1। 1 কে প্রসেসিং এর জন্য কিউতে ঢুকাই।
0 থেকে যদি আমি 2 এ যাই, তাহলে 2 এর নতুন দূরত্ব হবে 3। 2 কে প্রসেসিং এর জন্য কিউতে ঢুকাই।
0 থেকে যদি আমি 3 এ যাই, তাহলে 3 এর নতুন দূরত্ব হবে 6। 3 কে প্রসেসিং এর জন্য কিউতে ঢুকাই।

এখন আমাদের কিউ এর অবস্থা এই রকম 1,2,3

শহর	দূরত্ব
0	0
1	1
2	3
3	6
4	অসীম

আমরা কিউ এর ডগা থেকে 1 কে বের করে আনবো প্রসেসিং এর জন্য। 1 এর দূরত্ব হচ্ছে 1।

1 থেকে যদি আমি 0 এ যাই, তাহলে 0 এর নতুন দূরত্ব হবে 1+1 = 2। কোন দরকার নাই আপডেটের, 0 এর দূরত্ব 0।
1 থেকে যদি আমি 2 এ যাই, তাহলে 2 এর নতুন দূরত্ব হবে 1+1 = 2। 2 কে প্রসেসিং এর জন্য আবার কিউতে ঢুকাই।
1 থেকে যদি আমি 4 এ যাই, তাহলে 4 এর নতুন দূরত্ব হবে 1+5 = 6। 4 কে প্রসেসিং এর জন্য আবার কিউতে ঢুকাই।

এখন আমাদের কিউ এর অবস্থা এই রকম 2,3,2,4

শহর	দূরত্ব
0	0

1	1
2	2
3	6
4	6

আমরা কিউ এর ডগা থেকে 2 কে বের করে আনবো প্রসেসিং এর জন্য। 2 এর দূরত্ব হচ্ছে 2।

2 থেকে যদি আমি 0 এ যাই, তাহলে 0 এর নতুন দূরত্ব হবে 2+3= 5 । কোন দরকার নাই আপডেটের, 0 এর দূরত্ব 0।
2 থেকে যদি আমি 1 এ যাই, তাহলে 1 এর নতুন দূরত্ব হবে 2+1= 3 । কোন দরকার নাই আপডেটের, 1 এর দূরত্ব 1।
2 থেকে যদি আমি 3 এ যাই, তাহলে 3 এর নতুন দূরত্ব হবে 2+2= 4 । 3 কে প্রসেসিং এর জন্য আবার কিউতে ঢুকাই।
2 থেকে যদি আমি 4 এ যাই, তাহলে 4 এর নতুন দূরত্ব হবে 2+2 = 4। 4 কে প্রসেসিং এর জন্য আবার কিউতে ঢুকাই।

এখন আমাদের কিউ এর অবস্থা এই রকম 3,2,4,3,4

শহর	দূরত্ব
0	0
1	1
2	2
3	4
4	4

আমরা কিউ এর ডগা থেকে 3 কে বের করে আনবো প্রসেসিং এর জন্য। 3 এর দূরত্ব হচ্ছে 4।

3 থেকে যদি আমি 3 এ যাই, তাহলে 0 এর নতুন দূরত্ব হবে 4+6= 10 । কোন দরকার নাই আপডেটের, 0 এর দূরত্ব 0।
3 থেকে যদি আমি 2 এ যাই, তাহলে 2 এর নতুন দূরত্ব হবে 4+2= 6। কোন দরকার নাই আপডেটের, 2 এর দূরত্ব 2।
3 থেকে যদি আমি 4 এ যাই, তাহলে 4 এর নতুন দূরত্ব হবে 4+1= 5। কোন দরকার নাই আপডেটের, 4 এর দূরত্ব 4।

এখন আমাদের কিউ এর অবস্থা এই রকম 2,4,3,4

আর দূরত্বগুলো হচ্ছে

শহর	দূরত্ব
0	0
1	1
2	2
3	4
4	4

এরপর কিউ খালি না হওয়া পর্যন্ত লুপ চলতে থাকবে। কিন্তু এরপর আর কোন আপডেট হবে না। কারণ এটাই আসলে 0 থেকে বাকি সব শহরগুলোতে যাবার সবচে' জন্য কম দূরত্ব।

সিপিপিতে এই অ্যালগরিদমটার ইম্প্লিমেন্টেশন হবে এরকম

```
vector<int> edge[100], cost[100];
const int infinity = 1000000000;

edge[i][j] = jth node connected with i
cost[i][j] = cost of that edge

int bfs(int source, int destination) {

        int d[100];
        for(int i=0; i<100; i++) d[i] = infinity;

        queue<int> q;
        q.push( source );
        d[ source ] = 0;

        while( !q.empty() ) {
                int u = q.front(); q.pop();
                int ucost = d[ u ];

                for(int i=0; i<edge[u].size(); i++) {
                        int v = edge[u][i], vcost = cost[ u ][i] + ucost;

                        // updating - this part is also called relaxing
                        if( d[v] > vcost ) {
                                d[v] = vcost;
                                q.push( v );
                        }
                }
        }

        return d[ destination ];
}
```

দ্বিতীয় অ্যালগরিদম - ডায়াক্সট্রা (Dijkstra)

BFS খুবই ভালো জিনিস। সত্যি কথা আমার ভয়াবহ পছন্দের একটা জিনিস BFS কারণ ধাই ধাই করে BFS কোড লিখে ফেলা যায়, যদি তুমি প্রবলেমটাকে গ্রাফের প্রবলেমে পাল্টাই ফেলতে পারো। অল্প কিছু চালাকি দিয়ে BFS কে ফাস্ট করে ফেলা যায়। একটা হয়তো তুমি এখনই দেখতে পাচ্ছো, একটা শহরকে যদি এরিমধ্যে কিউতে থাকে, ওটাকে আবার কিউতে ঢোকানোর কোন মানে নেই। এ ধরণের চালাকিকে আমরা প্রোগ্রামাররা বলি "প্রুনিঙ" (pruning)। প্রুনিং এর খাস বাংলা হচ্ছে ছাটাই করা - মানে ধরে ছাটাই করে ছোট করে দেয়া।

BFS এর একটা পিছুটান আছে। (পিছুটান বলতে আমি আসলে Draw-back বুঝাচ্ছি, ফেলে আসা স্মৃতির কথা বলছি না)
সেটা হচ্ছে, যদি এমন হয়, যে আমি শহর 5 কে যখন প্রসেস করলাম তার দূরত্ব পেলাম 100। তো আমি ওর আশে পাশের সব শহরকে আপডেট করলাম। তারপর আরেকটু পর আবার 5 কে পেলাম কিউতে তখন তার দূরত্ব হচ্ছে 50। আমি আবার

আশেপাশের সবাইকে আপডেট করলাম। তারপর আবার কিউতে 5 কে পেলাম। 5 মুখটা পাঁচ করে ভ্যাবাচ্যাকা খেয়ে দাঁড়িয়ে আছে 40 দূরত্ব নিয়ে।

তো তুমি দেখতে পাচ্ছো, আমাদের আপডেটে একটা সমস্যা আছে - আমরা কিউতে যাকে আগে পাচ্ছি তাকে প্রসেস করছি। তো এভাবে প্রসেস করলে এরকম একটা পরিস্থিতি সম্ভব যেখানে আমরা একই শহরকে বহুবার ভুল দূরত্ব পেয়েও আপডেট করতে থাকবো। ওই শহরের উপর যেসব শহর নির্ভরশীল তারাও বহুবার আপডেট হবে। তাদের উপর যারা নির্ভরশীল তারাও বহুবার আপডেট হবে। সব মিলে একটা মহা ক্যাচাল লেগে যাচ্ছে আপডেটের।

Edsger W. Dijkstra ১৯৫৯ সালে এই ক্যাচাল বন্ধ করার একটা পথ খুঁজে পেলো। সে বলল কি, আমরা আরেকটা চালাকি করে দেখতে পারি, কিউ থেকে যখন শহর বের করে আনবো প্রসেস করার জন্য, আমরা সবচে' কাছের শহরটাকে বের করে আনতে পারি। তাহলে সেই শহরটাকে আর কখনো আরেকবার প্রসেস করা লাগবে না।

একটা মজার জিনিস হচ্ছে, ডায়াক্সট্রা যখন বিয়ে করতে গেলো, বিয়ে রেজিস্টার করার জন্য ওর পেশাতে লিখলো - "কম্পিউটার প্রোগ্রামার", তারপর কাজি অফিসের লোকগুলা বলল মিথ্যা কথা কম বলতে, "কম্পিউটার প্রোগ্রামার" কোন পেশার নামই হতে পারে না। তো সে আরেকটা পেশার নাম লিখলো, তারপর লোকগুলো ওকে বিয়ে করতে দিলো। কে জানতো পন্চাশ বছর পর লাখ লাখ মানুষ "কম্পিউটার প্রোগ্রামিং" করে পয়সা কামাবে?

তো আমাদের কাজ হচ্ছে শুধু কিউটা তুলে দিয়ে সেখানে একটা প্রায়োরিটি কিউ বসানোর, সে নোডটা সবচে' কম দূরত্বে আছে তাকে আমরা সবার আগে প্রসেস করবো - এটা হচ্ছে আমাদের প্রায়েরিটি।

সিপিপি তে একটা সহজ ইম্প্লিমেন্টেশন হবে এরকম -

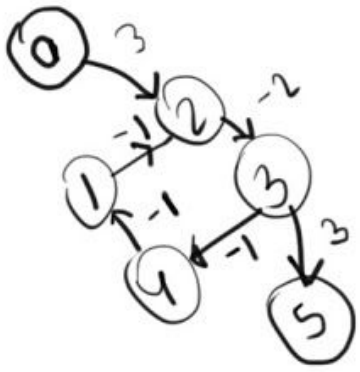
```
vector<int> edge[100], cost[100];
const int infinity = 1000000000;

edge[i][j] = jth node connected with i
cost[i][j] = cost of that edge

struct data {
        int city, dist;
        bool operator < ( const data& p ) const {
                return dist > p.dist;
        }
};

int dijkstra(int source, int destination) {

        int d[100];
        for(int i=0; i<100; i++) d[i] = infinity;

        priority_queue<data> q;
        data u, v;
        u.city = source, u.dist = 0;
        q.push( u );
        d[ source ] = 0;

        while( !q.empty() ) {
```

```
            u = q.top(); q.pop();
            int ucost = d[ u.city ];

            for(int i=0; i<edge[u.city].size(); i++) {
                v.city = edge[u.city][i], v.dist = cost[u.city][i] + ucost;
                // relaxing :)
                if( d[v.city] > v.dist ) {
                    d[v.city] = v.dist;
                    q.push( v );
                }
            }
        }

        return d[ destination ];
    }
```

আমি প্রথম দিকে কেন জানি ডায়াক্সট্রা লিখতে খুব ভয় পেতাম। আমি এখন ঠিক বুঝতে পারি না কেন। আমি কেন ডায়াক্সট্রা লিখতে ভয় পেতাম, সেটা আমার কাছে বিশাল রহস্য। বিশ্বাস করো আর নাই করো - তুমি যদি তোমার ভয়ের সুইচটা কোনভাবে বন্ধ করে দিতে পারো, তোমার ভাঙা গাড়ি রকেটের মতো চলতে শুরু করবে।

তৃতীয় অ্যালগরিদম - বেলম্যান ফোর্ড (Bellman Ford)

গ্রাফটা যদি এরকম হয় তাহলে কি হবে?



আমরা যতবার ইচ্ছা 2-3-4-1 এ পাক খেতে পারি। যতবার পাক খাবো তত আমাদের 0 থেকে 5 এর দূরত্ব কমবে, তাই না? তো এই লুপ অনন্তবার চলতে থাকবে তো চলতে থাকবে। আমাদের BFS তো ফেইল খাবেই খাবে, ডায়াক্সট্রাও ফেইল খাবে।

গ্রাফে যদি নেগেটিভ সাইকেল থাকে - শুধু সেক্ষেত্রেই এই গ্যানজামটা লাগতেসে।
তো এই ক্ষেত্রে আমরা বেলম্যান ফোর্ড ব্যবহার করি। খেয়াল করো, যে একটা কানেক্টেড গ্রাফে যদি একটা নেগেটিভ সাইকেল আমরা পাই, তাহলে গ্রাফটাতে শর্টেস্ট পাথ খোঁজার আর কোন মানে হয় না। বেলম্যান ফোর্ড করে কি যতক্ষণ রিল্যাক্স করা যায়, ততক্ষণ রিল্যাক্স করে। (রিলাক্স মানে আপডেট করা, আরাম করা না কিন্তু - আর সে n-1 এর বেশি আপডেট করে না) তারপর রিলাক্স করা শেষে সে দেখে এখনো কোন রিলাক্স করার মতো edge আছে কিনা। n-1 বার প্রতিটা নোড নিয়ে আপডেট করলে, ততক্ষণে সবগুলো আপডেট করার মতো নোড আপডেট হয়ে যাবার কথা। যদি তা না হয়, তার মানে অবশ্যই নেগেটিভ একটা সাইকেল আছে কোথাও।

এখানে রিল্যাক্স করার মানে হচ্ছে যদি এজটা হয় u থেকে v তে আর দৈর্ঘ্য হয় cost[u][v]

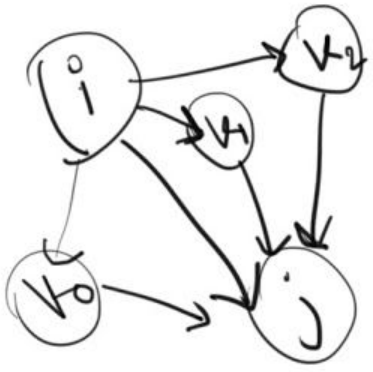
```
if(  d[v] > d[u] + cost[u][v] )
    d[v] = d[u] + cost[u][v];
```

তুমি যাতে এখানে এসে একটা ধাক্কা না খাও, সেজন্য আগের প্রত্যেকবার রিল্যাক্স করার সময় কমেন্টে আমি আলাদা করে বলে নিয়েছি যে আমি রিল্যাক্স করছি।

বেলম্যান ফোর্ডের একটা সহজ সুডোকোড উইকিপিডিয়াতে আছে। যেহেতু তুমি এখন শর্টেস্ট পাথ নিয়ে অনেক কিছু জানো, তোমার একটুও কষ্ট হবে না এটা বুঝতে।

চতুর্থ অ্যালগরিদম - ফ্লয়েড ওয়ার্শাল (Floyd-Warshall)

ফ্লয়েড ওয়ার্শাল এর মূল কন্সেপ্টটা খুব সহজ। তুমি যদি কন্সেপ্টটা বোঝো, তুমি যেকোন ফ্লয়েড ওয়ার্শাল প্রবলেম সলভ করতে পারবে। কিন্তু অ্যালগরিদমটা কেন কাজ করে সেটা বোঝার জন্য তুমি এখানে একটা ঢু মারতে পারো। আমার এই সাধাসিধা টিউটোরিয়ালটা দিয়ে আমি আপাতত কাওকে ভয় পাওয়াতে চাই না কঠিন কিছু লিখে।



আমি যদি i থেকে j তে যেতে চাই তাহলে , হয় আমি সোজা i থেকে j তে যাবো, অথবা কোন একটা k হয়ে j তে যাবো। যখন আমি k হয়ে যাবো তখন আমার i থেকে j এর সবচে' ছোট দূরত্ব dist[i][j] হবে dist[i][k] + dist[k][j] । এখানে dist[a][b] মানে

a থেকে b যাওয়ার সবচে' ছোট পথটার দূরত্ব। আমি যদি সবগুলো k এর জন্য ট্রাই করি একবার করে, তাহলে আপডেট করতে থাকলে আপডেট করার শেষে অবশ্যই dist[a][b] হবে a থেকে b তে যাওয়ার সবচে' ছোট পথ।

তাহলে অ্যালগরিদমটা দাড়াচ্ছে এরকম

```
int d[100][100]; // d[i][j] = distance from i to j

for(int k=0; k<n; k++)
for(int i=0; i<n; i++)
for(int j=0; j<n; j++) {
        if( d[i][j] > d[i][k] + d[k][j] )
                d[i][j] = d[i][k] + d[k][j];
}
```

ফ্লয়েড ওয়ার্শালের মজা হচ্ছে এটা লেখা খুব সহজ। সত্যি কথা আমি যখন লিখি কন্টেস্টের সময় ম্যাকরো লাগিয়ে তখন সেটা দেখতে এরকম হয়

```
REP(k,n) REP(i,n) REP(j,n) d[i][j] = min( d[i][j], d[i][k] + d[k][j] );
```

আর আরেকটা সুবিধা হচ্ছে, বাকি সব অ্যালগরিদম একটা শহরকে ধরে নেয় শুরুর শহর (source), তারপর বাকি সবগুলো শহরের দূরত্ব বের করে ওই শহরটা থেকে। এটাকে আমরা বলি Single Source Shortest Path (SSSP)। ফ্লয়েড ওয়ার্শাল করে কি সবগুলো শহর থেকে সবগুলো শহরের শর্টেস্ট পাথ বের করে ফেলে। তোমার অ্যলগরিদম শেষে তুমি যদি জানতে চাও a থেকে b এর দূরত্ব কত সেটা তুমি dist[a][b] থেকেই পেয়ে যাবে।

শর্টেস্ট পাথ - প্রবলেম নিয়ে বকর বকর

ইকরাম মাহমুদ, ঢাকা বিশ্ববিদ্যালয়, গুগল

প্যাট-প্যাটানি

> *Prime numbers are what is left when you have taken all the patterns away. I think prime numbers are like life. --- Mark Haddon (The Curious Incident of the Dog at Night Time)*

উপরের কোটেশনটার সাথে শর্টেস্ট পাথের তেমন কোন সম্পর্ক নেই। কিন্তু কোটেশনটা দেখার পর আমার লিখতে ইচ্ছা করলো, সেজন্য আমি কোটেশনটা তুলে দিলাম।

সত্যিকারের পৃথিবীতে তুমি প্রচুর প্যাটার্ন খুঁজে পাবে। ধরো তুমি যদি একটা গ্যাস বেলুন কিনে সেটাকে ছেড়ে দাও, সেটা সবসময়ই আকাশের দিকে উঠে যাবে। তুমি যদি আরেকটা গ্যাস বেলুন কিনে একই কাজ করো, একই জিনিস ঘটবে। তো এটা হচ্ছে একটা প্যাটার্ন। প্যাটার্ন বলতে আমি বুঝাচ্ছি একই ধাঁচের ঘটনা। ধরো একটা জায়গায় আগুন লাগলো, তুমি যদি সেখানে পানি ঠেলে দাও, আগুনটা নিভে যাবে। আরেকটা জায়গায় যদি আগুন লাগে তাহলে একই সলুশন কাজ করবে। কারণ দুইটা একই ধাঁচের ঘটনা। এখন তোমার শুধু একটা বালতি খুঁজে বের করতে হবে আর পানি এনে আগুনটায় ঢালতে হবে। এরপর যখনই তুমি একই প্যাটার্নটা দেখতে পাবে, তুমি অলরেডি এটার সলুশন জানো। বেশ তো, এখন ধরো একটা ছোট্ট আগুন না লেগে একটা আস্ত তিনতালা বিল্ডিং এ আগুন লেগে গেলো । এখন আর আমাদের বালতি সলুশন কাজ করবে না। আমরা পানিই ঢালবো কিন্তু সলুশনটা একটু পাল্টাতে হবে। বালতির বদলে ফায়ার সার্ভিসের গাড়ি লাগবে। কিন্তু মূল সলুশন একই থাকলো, আমরা পানিই ঢালবো।

বেশির ভাগ বাস্তব জগতের প্রবলেমে এই ধরণের প্যাটার্ন পাওয়া যায়। তুমি যদি একটু ভালোভাবে প্রবলেমটা বোঝার চেষ্টা করো, তুমি প্রবলেমের ধাঁচগুলো বুঝতে পারবে। তারপর তুমি এটাও বুঝতে পারবে যে এই প্রবলেমটার জন্য একটা সম্পূর্ণ নতুন সলুশন বের করার কোন দরকার নাই, তোমার শুধু এই প্রবলেমটাকে অন্য একটা প্রবলেমের প্যাটার্নে ফেলতে হবে যেই প্যাটার্নের সলুশন তুমি অলরেডি জানো । তারপর সলুশনটাকে পাল্টাতে হবে যাতে সলুশনটা এই প্রবলেমটার জন্যও কাজ করে।

কি রকম?

ধরো, একটা ট্রাক ড্রাইভার কুমিল্লা থেকে চুয়াডাঙা যাবে। যদি সে কিছু তেল বাঁচাতে পারে, তাইলে সেই তেল বেচে সে নারিকেল কিনতে পারবে। তো সে করবে কি, এমন ভাবে কুমিল্লা থেকে চুয়াডাঙা যাবে যাতে তার পথের দৈর্ঘ্য সবচে কম হয়। তোমার যদি আগের টিউটোরিয়ালটা পড়া থাকে, তুমি নিশ্চই বুঝতে পারছো, এখানে শহরগুলো হচ্ছে আমার node, আর শহরের মাঝের পথগুলো হচ্ছে আমার edge, আর পথের দৈর্ঘ্য হচ্ছে আমার cost। আমরা cost মিনিমাইজ করে দুইটা node এর মধ্যে একটা পথ খুঁজছি। তো এভাবে চিন্তা করলে এই প্রবলেমটা আপনাআপনিই শর্টেস্ট পাথ প্রবলেম হয়ে যাচ্ছে, যেটার সলুশন তুমি এর মধ্যেই জানো।

তারপর ধরো, একটা ট্রাভেল এজেন্ট তোমার জন্য দুবাই এর টিকেট কাটবে। তুমি পয়সা বাঁচাতে চাও, কারণ তুমি খুব গরিব। ট্রাভেল এজেন্ট এখন তোমার জন্য সবচে' সস্তা রুটটা বের করবে। এখন দেখো, যদি দুইটা শহরের মধ্যে একটা ফ্লাইট থাকে, তাইলে সেটা হচ্ছে আমাদের edge(অবশ্যই শহরগুলো হচ্ছে node), আর এই edge এর cost হচ্ছে টিকেট এর দাম। আমরা এখন দুইটা node এর মধ্যে একটা পথ খুঁজে বের করার চেষ্টা করছি যেখানে টিকেটের দাম (cost) সবচে' কম হবে।

ট্রাক ড্রাইভারের প্রবলেম আর দুবাইওয়ালার প্রবলেম, দুইটা শেষ পর্যন্ত ঘুরে ফিরে একই প্রবলেম হয়ে যাচ্ছে যখন তুমি প্যাটার্নটা দেখতে পাচ্ছো।

আরিকটু কঠিন উদাহরণ

(৪ লিটার মানে কিন্তু 4 liters)

তোমাকে তোমার বউ দুইটা বালতি ধরায় দিলো। একটা বালতি লাল আরেকটা বালতি নীল। লাল বালতির সাইজ হচ্ছে ৫ লিটার আর নীল বালতির সাইজ হচ্ছে ৩ লিটার। ওয়াশিং মেশিনে ঠিক ৪ লিটার পানি ঢালতে হবে একবারে একটা বালতি থেকে, কারণ ওয়াশিং মেশিনটা এত মোটকা যে ওটা কিছুতেই বাথরুমে ঢুকানো যাচ্ছা না। তুমি ট্যাপ থেকে ইচ্ছা মতো পানি নিতে পারো বালতিতে। এখন একটা বালতিতে ৪ লিটার পানি ক্যামনে বানানো যায়?

তুমি তিন ধরণের কাজ করতে পারো
১. একটা বালতিতে পুরোপুরি পানি ভরতে পারো
২. বালতি পুরাটা খালি করতে পারো
৩. একটা বালতি থেকে আরেকটা বালতিতে পানি ঢালতে পারো, তবে শর্ত হচ্ছে কোন পানি উপচাতে পারবে না, তাহলে তোমার সাধের কার্পেট নষ্ট হয়ে যাবে।

তোমাকে যদি আমি বলি এখথুনি এই কাজটা করে দাও সবচে' কম সময় নিয়ে, তাহলে প্রবলেমটা একটা শর্টেস্ট পাথ প্রবলেম হয়ে যায়!

এটুকু শোনার পর তোমার একটু ধাক্কা খাওয়া কথা। কারণ সোজাসুজি এখানে শহর দেখা যাচ্ছে না, পথও দেখা যাচ্ছে না, শর্টেস্ট পাথ কোথথেকে আসলো সেটা তো আরো পরের কথা। ঠিক এই জন্যই গ্রাফ থিওরির টার্মগুলো কাজের। সেগুলো তোমাকে সাহায্য করবে প্রবলেমটার একটা অ্যাবস্ট্রাক্ট শেপ তৈরী করতে।

এবার চোখ বন্ধ করে দুইটা বালতি কল্পনা করো। লাল বালতিটা নীলটার চে একটু বড়। দুইটায় পানি ভর্তি। এখন তুমি যদি তিন ঘন্টা ঢালাঢালির পর ক্লান্ত হয়ে আমাকে ফোন করো, তুমি ঠিক কি কি তথ্য দিবা আমাকে? অবশ্যই তুমি আমাকে বালতি দুইটার সাইজ বলবা। তারপর? এবার একটু ভাবো আর কি কি বললে আমি ঠিকঠাক মতো বুঝবো তুমি কোন অবস্থায় আছো।

অবশ্যই সেটা হচ্ছে তিন ঘন্টা ঢালাঢালির পর কোন বালতিতে কতটুকু পানি আছে। এটা বললে আমি পরিষ্কার বুঝতে পারবো তোমার অবস্থা কি। আমার আর তোমার বাসায় আসা লাগবে না। আমরা এটাকে state, সোজা বাংলায় অবস্থা।

ধরো কোন ভাবে আমার লাল বালতিতে ৩ লিটার পানি আছে আর নীল বালতিতে ২ লিটার। এই অবস্থা থেকে আর কোন কোন অবস্থায় যাওয়া যায় ?

১. লাল ৫ লিটার, নীল ০ লিটার (নীল বালতির সব পানি লাল বালতিতে)
২. লাল ২ লিটার, নীল ৩ লিটার (নীল বালতি ভর্তি করলাম লাল বালতি থেকে ঢেলে)
৩. লাল ০ লিটার, নীল ২ লিটার (লাল বালতি খালি করলাম)
৪. লাল ৩ লিটার, নীল ০ লিটার (নীল বালতি খালি করলাম)
৫. লাল ০ লিটার, নীল ০ লিটার (দুইটাই খালি করলাম)

তো এবার দেখো, তোমার একটা অবস্থা থেকে আরেকটা অবস্থায় যাওয়ার জন্য একটা move লাগতেসে। আমরা যদি অবস্থা গুলোকে শহর চিন্তা করি, তাহলে শহর (৩,২) থেকে পাঁচটা শহরে যাওয়া যায় (৫,০), (২,৩), (০,২), (৩,০), (০,০), এবং যেতে একটা move লাগে।

আমার প্রবলেম হচ্ছে আমাকে (০,০) শহর থেকে (x,৪) অথবা (৪,x) শহরে যেতে হবে সবচে' কম সংখ্যক move দিয়ে, যখন x যেকোন সংখ্যা হতে পারে। (একবার ৪ লিটার পানি পাইলে আর বাকি বালতি নিয়ে চিন্তা করার কোন দরকার নাই, তাই না?) হুমম, প্রবলেমটা কি চেনা চেনা লাগতেসে না? আমরা ঘুরে ফিরে আবার শর্টেস্ট পাথ প্রবলেমে ফিরে আসলাম।

এই প্রবলেমটা হচ্ছে হুবহু UVa 571। অবশ্য একই সাথে পথও বের করতে হয়। এই প্রবলেমটার আমার সলুশন হচ্ছে এটা। তুমি যদি কোড দেখে ভয় পাও, তাহলে ভূতরা কষ্ট পাবে।

```
#include<cstdio>
#include<cstring>
#include<cstdlib>
#include<cctype>

#include<cmath>
#include<iostream>
#include<fstream>
#include<numeric>

#include<string>
#include<vector>
#include<queue>
#include<map>
#include<algorithm>
#include<set>
#include<sstream>
#include<stack>
#include<list>
#include<iterator>
using namespace std;

#define REP(i,n) for(__typeof(n) i=0; i<(n); i++)
#define FOR(i,a,b) for(__typeof(b) i=(a); i<=(b); i++)
#define CLEAR(t) memset((t), 0, sizeof(t))

#define sz size()
#define pb push_back
#define pf push_front

#define VI vector<int>
#define VS vector<string>
#define LL long long

#define INF (1<<30)
#define eps 1e-11

#define fillA 0
#define fillB 1
#define emptyA 2
#define emptyB 3
#define pourAB 4
#define pourBA 5
```

```
string s[] = { "fill A", "fill B", "empty A", "empty B", "pour A B", "pour B A" };
struct state { int a, b; };
int dist[1005][1005];
state parent[1005][1005];
int work[1005][1005];

void print( int xa, int xb ) {
        if( dist[xa][xb] != 0 ) {
        print( parent[xa][xb].a, parent[xa][xb].b );
        printf("%s\n",s[ work[xa][xb] ].c_str());
        }
}

int main() {
        state ux,u,v;
        int x;
        int n;
        while( scanf("%d%d%d", &ux.a, &ux.b, &n) == 3 ) {


        REP(i,1002) REP(j,1002) dist[i][j] = INF;
        u.a = 0; u.b = 0;
        dist[u.a][u.b] = 0;

        queue<state> q;
        q.push( u );
        while( !q.empty() ) {
        u = q.front();
        q.pop();
        //cout <<"processing "<< u.a <<" "<<u.b << endl;
        if( u.b == n ) {
                //cout <<u.a<<" "<<u.b<<endl;
                break;
        }


        // now the operations
        x = dist[u.a][u.b] + 1;

        //filling A
        v.a = ux.a;
        v.b = u.b;

        if( dist[v.a][v.b] > x ) {
                dist[v.a][v.b] = x;
                parent[v.a][v.b] = u;
                q.push( v );
                work[v.a][v.b] = fillA;
        }

        // filling B
        v.a = u.a;
```

```
v.b = ux.b;

if( dist[v.a][v.b] > x ) {
        dist[v.a][v.b] = x;
        parent[v.a][v.b] = u;
        q.push( v );
        work[v.a][v.b] = fillB;
}

// empty A
v.a = 0;
v.b = u.b;

if( dist[v.a][v.b] > x ) {
        dist[v.a][v.b] = x;
        parent[v.a][v.b] = u;
        q.push( v );
        work[v.a][v.b] = emptyA;
}

// empty B
v.a = u.a;
v.b = 0;

if( dist[v.a][v.b] > x ) {
        dist[v.a][v.b] = x;
        parent[v.a][v.b] = u;
        q.push( v );
        work[v.a][v.b] = emptyB;
}


//pour A -> B
if( u.a+u.b >= ux.b ) {
        v.a = u.a - ( ux.b - u.b );
        v.b = ux.b;
}else {
        v.b = u.b + u.a;
        v.a = 0;
}



if( dist[v.a][v.b] > x ) {
        dist[v.a][v.b] = x;
        parent[v.a][v.b] = u;
        q.push( v );
        work[v.a][v.b] = pourAB;
}

// pour B -> A
```

```
        if( u.a+u.b >= ux.a ) {
                v.b = u.b - ( ux.a - u.a );
                v.a = ux.a;
        }else {
                v.a = u.a + u.b;
                v.b = 0;
        }

        if( dist[v.a][v.b] > x ) {
                dist[v.a][v.b] = x;
                parent[v.a][v.b] = u;
                q.push( v );
                work[v.a][v.b] = pourBA;
        }

        }
        print( u.a, u.b );
        printf("success\n");

        }


        return 0;
}
```

আরেকটা মজার প্রবলেম

সুপার মারিও একটা মনিটরের ভিতর থাকে। সেখানে অনেকগুলো গ্রাম আর অনেকগুলো দূর্গ। সুপার মারিও পুরো জায়গাটা গুগল ম্যাপ খুলে প্রচুর মাপামাপি করসে, তো সে খুব ভালো করে জানে কোন কোন গ্রামগুলো বা দূর্গগুলোর মধ্যে পথ আছে, আর পথ থাকলে সেই পথের দূরত্ব কত। তো মারিও বহুদিন একা একা পথ মাপামাপি করার পর বোরড হয়ে করলো কি, একটা দূর্গে গিয়ে একটা রাজকণ্যাকে উদ্ধার করলো।কিভাবে কিভাবে সেই দূর্গে সে একটা জাদুর বুট খুঁজে পেলো, যেটা পড়লে সে নিমিষে একটা জায়গা থেকে আরেকটা জায়গায় চলে যেতে পারে। শুধু দুইটা সমস্য। বুটটা বেশ পুরনো, তো কিছুতেই ৫০ কিলোমিটারের বেশি টানা বুট পরে দৌড়ানো যায় না, আর বুটটাকে ৩ বারের বেশি ব্যবহার করা যায় না। এর পরের বার ব্যবহার করলে বুটটা সোজা গর্তে গিয়ে হাজির হবে।

আরেকটা বাজে জিনিস হচ্ছে ধাম করে একটা দেয়ালে বাড়ি না খেলে সে থামতে পারে না, তো দৌড় থামানোর জন্য ওর একটা গ্রামে থামতে হবে অথবা একটা দূর্গে থামতে হবে। আর দৌড় শুরু করার জন্য তার একটা জায়গায় বসে বুট পরতে হবে, পথের মাঝখানে বসে সে বুট পরতে পারবে না, সেজন্য তার দৌড় শুরু করতে হবে কোন একটা গ্রাম থেকে অথবা দূর্গ থেকে।

মারিও এখন কোন এক অজানা(!) কারণে খুব দ্রুত রাজকণ্যাকে নিয়ে বাড়ি ফিরতে চায়! খালি ওর মাথায় একটু বুদ্ধি কম, সেজন্য সে বুঝতে পারছে না কিভাবে কিভাবে গেলে বা কিভাবে কিভাবে বুট পড়লে সবচে' দ্রুত বাড়ি ফেরা যাবে। তো মারিওকে বলতে হবে কিভাবে কিভাবে দৌড়ালে সে সবচে' কম সময়ে বাড়ি ফিরতে পারবে।

ঢিশিয়া

আমাদের আগের প্রবলেমটার পুরো সিচুয়েশন বর্ণনা করা যেতো (লাল বালতিতে পানি, নীল বালতিতে পানি) এই দুইটা জিনিস দিয়ে। আমরা তারপর পুরো প্রবলেমটাকে এমনভাবে মডেল করসিলাম, যাতে জিনিসটাকে শর্টেস্ট পাথ প্রবলেম বানিয়ে ফেলা যায়।

ঠিক একইভাবে চিন্তা করো, যদি সুপার মারিও অনেক দৌড়াদৌড়ির পর (কিংবা দৌড়াতে দৌড়াতে) তোমাকে ফোন করে বুদ্ধি চায়, সে তোমাকে কি কি বললে তুমি পুরো সিচুয়েশনটা বুঝতে পারবা? অবশ্যই সবার আগে তাকে বলতে হবে সে কোন জায়গায় আছে। তারপর? তারপর তাকে বলতে হবে সে আর কতবার বুটটা ব্যবহার করতে পারবে। এবং যদি সে বুট পরে দৌড়াতে থাকে, তাহলে বলতে হবে সে কতটুকু দৌড়ালো বুট পড়ে। এই তিনটা বললে আমরা পুরো অবস্থাটা হাড়ে হাড়ে বুঝতে পারি, তাই না?

তো এটা হচ্ছে আমাদের state (কোথায়, বুট কয়বার পরতে পারবে, এই বুটে আর কত মাইল দৌড়ানো যাবে)। এবার এটাকে [a][b][c] হিসেবে চিন্তা করো, আর যদি [a][b][c] থেকে [x][y][z] এ যাওয়া যায়, তাহলে তার মানে হচ্ছে এই দুইটা অবস্থার মধ্যে একটা edge আছে, যেটার cost হবে a থেকে x এর দূরত্ব, যেটা মারিও ইতিমধ্যে জানে। আর বুট পরা থাকলে তো কোন কথাই নাই, মারিওর কোন সময়ই লাগবে না যাইতে। তো আমাদের বর্তমান দূর্গ থেকে বাড়ি তে যাবার সবচে' কম সময়ে যাবার পথ খুঁজে বের করতে হবে।

তো এটা আবারও শর্টেস্ট পাথে ফিরে আসলো ঘুরে ফিরে।
এই প্রবলেমটা হচ্ছে UVa 10269 আর এটা হচ্ছে আমার সলুশন।

```
#include<cstdio>
#include<sstream>
#include<cstdlib>
#include<cctype>
#include<cmath>
#include<algorithm>
#include<set>
#include<queue>
#include<stack>
#include<list>
#include<iostream>
#include<fstream>
#include<numeric>
#include<string>
#include<vector>
#include<cstring>
#include<map>
#include<iterator>
using namespace std;

#define FORIT(i, m) for (__typeof((m).begin()) i=(m).begin(); i!=(m).end(); ++i)
#define REP(i,n) for(int i=0; i<(n); i++)
#define FOR(i,a,b) for(__typeof(b) i=(a); i<=(b); i++)

#define sz size()
#define pb push_back
#define ALL(x) x.begin(), x.end()

#define i64 long long
#define SET(t,v) memset((t), (v), sizeof(t))
```

```
#define REV(x) reverse( ALL( x ) )

#define IO freopen("","r",stdin); freopen("","w",stdout);
#define debug(x) cerr << __LINE__ <<" "<< #x " = " << x << endl

typedef vector<int> vi;

int A,B,M,L,K;
int memo[105][505][15];
vi edge[105], cost[105];

int main() {
	int t;
	scanf("%d",&t);

	while( t-- ) {
		scanf("%d %d %d %d %d",&A,&B,&M,&L,&K);
		int x,y,z;

		REP(i,105) { edge[i].clear(); cost[i].clear(); }
		SET( memo, 63 );
		int inf = memo[0][0][0];

		while( M-- ) {
			scanf("%d %d %d",&x,&y,&z);
			edge[x].pb( y ); edge[y].pb( x );
			cost[x].pb( z ); cost[y].pb( z );
		}

		memo[A+B][0][K] = 0;
		queue< int > q;
		q.push( A+B ); q.push( 0 ); q.push( K );

		while( !q.empty() ) {
			int a = q.front(); q.pop();
			int l = q.front(); q.pop();
			int k = q.front(); q.pop();
			int bcost = memo[a][l][k];

			REP(i,edge[a].sz) {
				int d = cost[a][i];
				if( a <= A ) {
					// opt 1
					if( l >= d ) {
						int aa = edge[a][i];
						int ll = l - d;
						int kk = k;
						if( memo[aa][ll][kk] > bcost ) {
							memo[aa][ll][kk] = bcost;
							q.push( aa ); q.push( ll ); q.push( kk );
						}
					}
```

```
                                        // opt 2
                                        if( k > 0 && L >= d ) {
                                                int aa = edge[a][i];
                                                int ll = L - d;
                                                int kk = k-1;
                                                if( memo[aa][ll][kk] > bcost ) {
                                                        memo[aa][ll][kk] = bcost;
                                                        q.push( aa ); q.push( ll ); q.push( kk );
                                                }
                                        }

                                        // opt 3
                                        int aa = edge[a][i];
                                        int ll = 0;
                                        int kk = k;
                                        if( memo[aa][ll][kk] > bcost + d ) {
                                                memo[aa][ll][kk] = bcost + d;
                                                q.push( aa ); q.push( ll ); q.push( kk );
                                        }
                                }else {
                                        // opt 2
                                        if( k > 0 && L >= d ) {
                                                int aa = edge[a][i];
                                                int ll = L - d;
                                                int kk = k-1;
                                                if( memo[aa][ll][kk] > bcost ) {
                                                        memo[aa][ll][kk] = bcost;
                                                        q.push( aa ); q.push( ll ); q.push( kk );
                                                }
                                        }

                                        // opt 3
                                        int aa = edge[a][i];
                                        int ll = 0;
                                        int kk = k;
                                        if( memo[aa][ll][kk] > bcost + d ) {
                                                memo[aa][ll][kk] = bcost + d;
                                                q.push( aa ); q.push( ll ); q.push( kk );
                                        }
                                }
                        }

                }

                int ans = inf;
                REP(l,L+1) REP(k,K+1) ans = min( ans, memo[1][l][k] );
                printf("%d\n", ans );


        }
```

```
    return 0;
}
```

শেষ কথা মোটেও শেষ কথা নয়

শর্টেস্ট পাথ কিভাবে মডেল করতে হয়, সেটা নিয়ে আমি আরো দুই দিস্তা গল্প লিখতে পারি। কিন্তু আমার মনে হয় তুমি বুঝতে শুরু করেছো, আসলে কি কারিশমাটা করতে হয় শর্টেস্ট পাথ প্রবলেমগুলোকে নিয়ে। তোমাকে প্রবলেমটাকে একটা গ্রাফে কনভার্ট করতে হবে, আর বুঝতে হবে ঠিক ঠিক কি জানলে তুমি একটা অবস্থা পুরোপুরি বর্ণনা করতে পারো। তুমি যদি বড় হয়ে কখনো সিরিয়াস প্রবলেম সলভার হও, কিংবা রিসার্চার হও, কিংবা সত্যিকারের কোন চরম এক্সাইটিং প্রবলেম নিয়ে কাজ করতে করতে বুড়ো হয়ে যাও, ঘুরে ফিরে তুমি একই কাজ করবা। তোমার প্রথম কাজ হবে গুতাই গুতাই দেখা যে প্রবলেমটাকে একটা known solutionওয়ালা প্রবলেম বানানো যায় কিনা। যদি একেবারেই না যায়, তাহলে তোমার দ্বিতীয় কাজ হবে একটা ঝাকানাকা সলুশন বের করে বিখ্যাত হয়ে যাওয়া। তারপর মানুষ জন এখন যেভাবে Dijkstra লিখে ফাংশন লিখে অনলাইন জাজে সাবমিট করে, সেরকমভাবে তোমার নামে ফাংশন লিখে অনলাইন জাজে সাবমিট করবে। তারপর ২৩ বার WA খেয়ে ফেইসবুকে তোমার নাম নিয়ে স্ট্যাটাস দিবে। কি মজা!

> *Hope, it is the quintessential human delusion, simultaneously the source of your greatest strength, and your greatest weakness.*
> *--- The Architect (Matrix Reloaded)*

খাটা মাথানোর জন্য

BFS

www.spoj.pl/problems/ANARC05I
www.spoj.pl/problems/BYTESE1
www.spoj.pl/problems/CERCO7K
www.spoj.pl/problems/CHMAZE
www.spoj.pl/problems/CLEANRBT
www.spoj.pl/problems/CLOCKS
www.spoj.pl/problems/CURSE
www.spoj.pl/problems/DP
www.spoj.pl/problems/ESCJAILA
www.spoj.pl/problems/HIKE
www.spoj.pl/problems/INUMBER
www.spoj.pl/problems/MAWORK
www.spoj.pl/problems/MLASERP
www.spoj.pl/problems/MTWALK
www.spoj.pl/problems/ONEZERO
www.spoj.pl/problems/PPATH

www.spoj.pl/problems/ROADS
www.spoj.pl/problems/SHOP
www.spoj.pl/problems/TOE1
www.spoj.pl/problems/TOE2

Dijkstra

www.spoj.pl/problems/GONDOR
www.spoj.pl/problems/HIGHWAYS
www.spoj.pl/problems/INCARDS
www.spoj.pl/problems/MELE3
www.spoj.pl/problems/NAJKRACI
www.spoj.pl/problems/SHPATH
www.spoj.pl/problems/TRAFFICN

এগুলো শেষ করে পেট না ভরলে http://uvatoolkit.com/ এখানে গিয়ে BFS বা Dijikstra লিখে একটা সার্চ মারো!

শুভ কামনা!

টাইম কমপ্লেক্সিটি

- ঝংকার মাহবুব

টাইম কমপ্লেক্সিটি নামটা একটু কঠিন। এইটার নাম আমি দিলে এইটার নাম হতো- "কতক্ষণ লাগবে?" বা আরেকটু গুছিয়ে বললে বলবো- "কোড রান করতে কতক্ষণ লাগবে?", বা "কোড কতবার চলবে?

যেমন ধর তোর কাছে একটা friends নামে একটা array আছে। সেটার মধ্যে তোর সব ফ্রেন্ডের নাম লেখা আছে। এখন তুই একটা for লুপ চালিয়ে তোর friends নামক array এর সব উপাদানকে আউটপুট হিসেবে দেখতে চাস, নিচের মতো

```
var friends = ["abul", "babul", "cabul", "dabul"];


for(var i = 0; i< friends.length; i++){
      var friend = friends[i];
      console.log(friend);
}
```

এই কোড রান করলে কয়টা আউটপুট দেখবি?

চারটা

কারণ তোর friends নামক array তে চারটা উপাদান আছে।

বিশ্বাস না হলে, www.habluderadda.com/consoleএ গিয়ে টাইপ করে চেক করে দেখ

যদি তোর friends নামক array তে চারটা উপাদান না থেকে পাঁচটা উপাদান থাকতো তাহলে কয়টা আউটপুট দেখতি? নিশ্চয়ই পাঁচটা দেখতি। আর তোর friends নামক array তে চৌদ্দটা উপাদান থাকলে তুই চৌদ্দটা আউটপুট দেখতি।

তারমানে তোর array এর মধ্যে যতগুলা উপাদান আছে, যতগুলা আউটপুট দেখাবে।

উপরের for লুপের দিকে আরেকবার তাকা। দেখবি লুপের ভিতরে একবার করে আউটপুট দেখানো আছে। আর লুপ একবার চললে একবার আউটপুট দেয়। তাই যদি চৌদ্দবার চলে, তাহলে চৌদ্দটা আউটপুট দেখায়।

তারমানে কতবার চলবে সেটা নির্ভর করতেছে কতটা উপাদান আছে। অর্থাৎ চারটা থাকলে চারবার। পাঁচটা থাকলে পাঁচবার। চৌদ্দটা থাকলে চৌদ্দবার। আর উপাদান সংখ্যা n হইলে, n বার চলবে।

যদি কোন একটা কোড বা n বার চলে তাকে প্রোগ্রামিংয়ের ভাষায় বলে এইটা O(n) বার চলবে। বা সেই কোডের টাইম কমপ্লেক্সিটি (time complexity) হচ্ছে O(n)

এইখানে O আছে Order শব্দ এর প্রথম অক্ষর থেকে। অর্ডার (order) শব্দের বাংলা অর্থ হচ্ছে মাত্রা। সেটা একটু পরে একটু ভালো করে বুঝা যাবে। আপাতত সামনে আগাইতে থাক।

ডাইরেক্ট একশন:

ধর তুই তোর friends নামক array থেকে যেকোন একটা উপাদান বের করতে চাস। ধরে তুই ২পজিশনে থাকা উপাদানকে বের করতে চাস। তাহলে তুই নিচের মতো করে কোড লিখবি।

```
var friends = ["abul", "babul", "cabul", "dabul", "ebul"];
var friend = friends[2];
```

এই যে একটা নিদৃষ্ট পজিশন থেকে একটা উপাদান বের করছস। সেটা করার জন্য তোকে পুরা array টা ঘুরা লাগে নাই। জাস্ট ওই পজিশনে গিয়ে, অর্থাৎ জায়গামতো গিয়ে পেয়ে গেছস। তাই এইটার জন্য ১টা জায়গায় যাওয়া লাগবে। তাই কোন একটা array থেকে কোন একটা নিদৃস্ট পজিশনের উপাদান বের করতে হলে একবার কোড চালানো লাগবে তাই এইটার time complexity হবে O(1)

সার্চ এলগোরিদম

তুই যদি কোন একটা array এর মধ্যে নিদৃষ্ট পজিশনের কাউকে বের না করে, কোন একটা উপাদান আছে কিনা জানতে চাস। তখন তোকে প্রথম উপাদান থেকে শুরু করতে হবে। তারপর একটা একটা করে সামনে আগাইতে আগাইতে দেখতে হবে সেটা আছে কিনা। তারমানে কোন কিছু খোঁজার জন্য তোকে সবগুলা উপাদানের একটা একটা করে দেখতে হয়। সেটার জন্য নিচের মতো করে কোড লিখস

```
var choukirTola = ["tShirt", "jeans", "juta", "boi", "gamcha", "charger",
"panjabi"];
for(var i = 0; i < choukirTola.length; i++){
      var jinish = choukirTola[i];
      if( jinish == "charger"){
            console.log("charger paisi");
            break;
      }
}
```

উপরের কোড যেহেতু তুই সবগুলা উপাদান খুজস। আর আর array এর মধ্যে কয়টা উপাদান আছে সেটাকে সাধারণভাবে বললে বলা যায় n সংখ্যক উপাদান আছে। তাই

Sort এলগোরিদম:

Selection Sort এর কোড দেখছিলি? না দেখলে এইখানে থেকে দেখে আয়।

নিচে Selection Sort এর কোড দেখানো আছে। এই কোডের আগা-মাথা কিচ্ছু না বুঝতে পারলেও এক নজর দেখ। দেখবি একটা for লুপের ভিতরে আরেকটা for লুপ আছে।

```
function selectionSort(arr){
  var minIdx, temp,
      len = arr.length;
  for(var i = 0; i < len; i++){
    minIdx = i;
    for(var  j = i+1; j<len; j++){
       if(arr[j]<arr[minIdx]){
          minIdx = j;
       }
    }
    temp = arr[i];
    arr[i] = arr[minIdx];
    arr[minIdx] = temp;
  }
  return arr;
}
```

তুই জানস, একটা for লুপের জন্য সবগুলা উপাদান একবার একবার করে চলে। তারমানে চারটা উপাদান থাকলে চারবার চলবে। তারমানে array এর মধ্যে যতগুলা উপাদান আছে, ততবার ভিতরের কোডগুলা চলবে। এখন ধর, একটা উপাদানের জন্য কোড for লুপের ভিতরে গেল। এখন ভিতরে গিয়ে দেখলো আরেকটা লুপ। এখন ভিতরের যেহেতু একটা লুপ, তাই ভিতরের লুপের জন্য সেই লুপ array এর যতগুলা উপাদান আছে, ততবার চলবে।

তারমানে বাইরের লুপের একটা উপাদানের জন্য ভিতরের লুপে যতগুলা উপাদান আছে ততবার চলবে। ধর, ভিতরের array এর মধ্যে n সংখ্যক উপাদান আছে। তাই ভিতরে লুপ n বার চলবে। আর যদি বাইরের লুপে দুইটা উপাদান থাকে, তাহলে বাইরের লুপের প্রথম উপাদানের জন্য ভিতরে গিয়ে, ভিতরের লুপ n বার চলবে। আবার বাইরের লুপের দ্বিতীয় উপাদানের জন্য ভিতরের লুপ n বার চলবে। তারমানে ভিতরের লুপ, দুইবার n সংখ্যকবার চলবে। এই দুইবার n সংখ্যকবার চলাকে লিখতে পারস ২n

আর যদি বাইরের array এর মধ্যে 3টা উপাদান থাকে, তাহলে ভিতরে গিয়ে, বাইরের লুপের প্রত্যেকটা উপাদানের জন্য n সংখ্যকবার চলে। মোট 3n বার চলবে।

তারমানে বাইরের লুপে একটা উপাদান থাকলে ভিতরের লুপ n বার। বাইরের লুপে 2টা উপাদান থাকলে ভিতরের লুপ 2n বার। আর বাইরের লুপে 3টা উপাদান থাকলে ভিতরের লুপ 3n বার চলবে। আর যদি বাইরের লুপে n সংখ্যক উপাদান থাকে তাহলে ভিতরের লুপ nXn বা n^2 বার চলবে।

তাই একটা লুপের ভিতরের আরেকটা লুপ থাকলে সেই কোডের টাইম কমপ্লেক্সিটি হয় $O(n^2)$

আপাতত এইটুক মনে রাখতে পারলেই হবে।

অ্যালগরিদম কমপ্লেক্সিটি(বিগ “O” নোটেশন)

শাফায়েত আশরাফ | অক্টোবর ১২, ২০১২

তুমি যখন একটা অ্যালগরিদমকে কোডে ইমপ্লিমেন্ট করবে তার আগে তোমার জানা দরকার অ্যালগরিদমটি কতটা কার্যকর। অ্যালগোরিদম লেখার আগে নিজে নিজে কিছু প্রশ্নের উত্তর দিতে হবে,যেমন:

১. আমার অ্যালগোরিদম কি নির্ধারিত সময়ের মধ্যে ফলাফল এনে দিবে?

২. সর্বোচ্চ কত বড় ইনপুটের জন্য আমার অ্যালগোরিদম কাজ করবে?

৩. আমার অ্যালগোরিদম কতখানি মেমরি ব্যবহার করছে?

আমরা অ্যালগোরিদমের কমপ্লেক্সিটি বের করে প্রথম দুটি প্রশ্নের উত্তর দিতে পারি। একটি অ্যালগরিদম যতগুলো “ইনস্ট্রাকশন” ব্যবহার করে কাজ করে সেটাই সোজা কথাই সেই অ্যালগোরিদমের কমপ্লেক্সিটি। দুটি নম্বর গুণ করা একটি ইনস্ট্রাকশন, আবার একটি লুপ ১০০ বার চললে সেখানে আছে ১০০টি ইনস্ট্রাকশন। ফলাফল আসতে কতক্ষণ লাগবে সেটা সিপিউর প্রসেসরের উপর নির্ভর করবে, কমপ্লেক্সিটি আমাদের cputime বলে দিবেনা, কমপ্লেক্সিটি আমাদের বলে দিবে আমাদের অ্যালগরিদমটি তুলনামূলকভাবে কতটা ভালো। অর্থাৎ এটা হলো অ্যালগরদিমের কার্যকারিতা নির্ধারণের একটা স্কেল। আর BIG O

নোটেশন হলো কমপ্লেক্সিটি লিখে প্রকাশ করার নোটেশন।

আমরা একটা উদাহরণ দিয়ে শুরু করি। আমাদের একটি ফাংশন আছে যার নাম myAlgorithm, আমরা সেই ফাংশনের কমপ্লেক্সিটি বের করবো। মনে করো ফাংশনটি এরকম:

```
int myAlgorithm1(int n)
{
  int x=n+10;
  x=x/2;
  return x;
}
```

এই অ্যালগোরিদমটি n এর ভ্যালু যাই হোকনা কেন সবসময় একটি constant সংখ্যক ইনস্ট্রাকশন নিয়ে কাজ করবে। কোডটিকে মেশিন কোডে পরিণত করলে যোগ-ভাগ মিলিয়ে ৩-৪ ইনস্ট্রাকশন পাওয়া যাবে,আমাদের সেটা নিয়ে ম্যাথাব্যাথার দরকার নাই। প্রসেসর এত দ্রুত কাজ করে যে এত কম ইনস্ট্রাকশন নিয়ে কাজ করতে যে সময় লাগে সেটা নিয়ে আমরা চিন্তাই করিনা,ইনস্ট্রাকশন অনেক বেশি হলে আমরা চিন্তা করি,আর লুপ না থাকলে সাধারণত চিন্তা করার মত বেশি হয়না।

অ্যালগোরিদমের কমপ্লেক্সিটি হলো O(1),এর মানে হলো ইনপুটের আকার যেমনই হোকনা কেন একটি constant টাইমে অ্যালগোরিদমটি কাজ করা শেষ করবে।

এবার পরের কোডটি দেখ:

```
int myAlgorithm2(int n)
{
  int sum=0;
  for(int i=1;i<=n;i++)
  {
    sum+=i;
    if(sum>=1000) break;
  }
  return sum;
}
```

এই কোডে একটি লুপ চলছে এবং সেটা n এর উপর নির্ভরশীল। n = 100 হলে লুপ ১০০ বার চলবে। লুপের ভিতরে বা বাইরে কয়টি ইনস্ট্রাকশন আছে সেটা নিয়ে চিন্তা করবোনা,কারণ সেটার সংখ্যা খুবই কম। উপরের অ্যালগোরিদমের কমপ্লেক্সিটি O(n) কারণ এখানে লুপটি n বার চলবে। তুমি বলতে পারো sum যদি ১০০০ এর থেকে বড় হয় তাহলে break করে দিচ্ছি, n পর্যন্ত চলার আগেই লুপটি break হয়ে যেতে পারে। কিন্তু প্রোগ্রামাররা সবসময় worst case বা সবথেকে খারাপ কেস নিয়ে কাজ করে! এটা ঠিক যে লুপটি আগে break করতেই পারে,কিন্তু worst case এ সেটা n পর্যন্তইতো চলবে।

worst case এ যতগুলো ইনস্ট্রাকশন থাকবে সেটাই আমাদের কমপ্লেক্সিটি।

```
int myAlgorithm3(int n)
{
        int sum=0;
        for(int i=1;i<=n;i++)
        {
                for(int j=i;j<=n;j++)
                {
                        sum+=(i+j);
```

```
            }
        }
        return sum;
}
```

উপরের কোডে ভিতরের লুপটা প্রথমবার n বার চলছে,পরেরবার n−1 বার। তাহলে মোট লুপ চলছে $n+(n-1)+(n-3)+\ldots\ldots+1=n\times(n+1)/2=(n^2+n)/2$ বার। n^2 এর সাথে n^2+n এর তেমন কোনো পার্থক্য নেই। আবার $n^2/2$ এর সাথে n^2 এর পার্থক্যও খুব সামান্য। তাই কমপ্লেক্সিটি হবে $O(n^2)$।

কমপ্লেক্সিটি হিসাবের সময় constant factor গুলোকে বাদ দিয়ে দিতে হয়। তবে কোড লেখার সময় constant factor এর কথা অবশ্যই মাথায় রাখতে হবে।

উপরের ৩টি অ্যালগোরিদমের মধ্যে সবথেকে সময় কম লাগবে কোনটির? অবশ্যই O(1) এর সময় কম লাগবে এবং $O(n^2)$

এর বেশি লাগবে। এভাবেই কমপ্লেক্সিটি হিসাব করে অ্যালগোরিদমের কার্যকারিতা তুলনা করা যায়। পরের কোডটি দেখো:

```
int myAlgorithm4(int n,int *val,int key)
{
        int low=1,high=n;
        while(low<=high)
        {
                int mid=(low+high)/2;
                if(key<val[mid]) low=mid-1;
                if(key>val[mid]) high=mid+1;
                if(key==val[mid]) return 1;
        }
        return 0;
}
```

এটা একটা বাইনারি সার্চের কোড। প্রতিবার low+high=n+1 বা n এর মান দুই ভাগে ভাগ হয়ে যাচ্ছে। একটি সংখ্যাকে সর্বোচ্চ কতবার ২দিয়ে ভাগ করা যায়? একটি হিসাব করলেই বের করতে পারবে সর্বোচ্চ ভাগ করা যাবে $\log_2 n$ বার। তারমানে $\log_2 n$

ধাপের পর লুপটি ব্রেক করবে। তাহলে কমপ্লেক্সিটি হবে $O(\log_2 n)$।

এখন ধরো একটি অ্যালগোরিদমে কয়েকটি লুপ আছে,একটি n^4 লুপ আছে,একটি n^2 লুপ আছে আর একটি log n লুপ আছে। তাহলে মোট ইনস্ট্রাকশন: $n^4+n^3+\log n$ টি। কিন্তু n^4 এর তুলনায় বাকি টার্মগুলো এতো ছোটো যে সেগুলোকে বাদ দিয়েই আমরা কমপ্লেক্সিটি হিসাব করবো, $O(n^4)$।

রিকার্সিভ ফাংশনে depth এর উপর কমপ্লেক্সিটি নির্ভর করে,যেমন:

```
int myAlgorithm5(int n)
{
	if(n==1) return 1;
	return n*myAlgorithm5(n-1);
}
```

এই অ্যালগোরিদমে সর্বোচ্চ depth হলো n, তাই কমপ্লেক্সিটি হলো O(n)। নিচে ছোট করে আরো কিছু উদাহরণ দিলাম:

f(n) = ইনস্ট্রাকশন সংখ্যা

$f(n) = n^2+3n+112$ হলে কমপ্লেক্সিটি $O(n^2)$।

$f(n) = n^3+999n+112$ হলে কমপ্লেক্সিটি $O(n^3)$।

$f(n) = 6\times\log(n)+n\times\log n$ হলে কমপ্লেক্সিটি $O(n\times\log n)$ ।

$f(n) = 2^n+n^2+100$ হলে কমপ্লেক্সিটি $O(2^n)$। (এটাকে exponential কমপ্লেক্সিটি বলে)

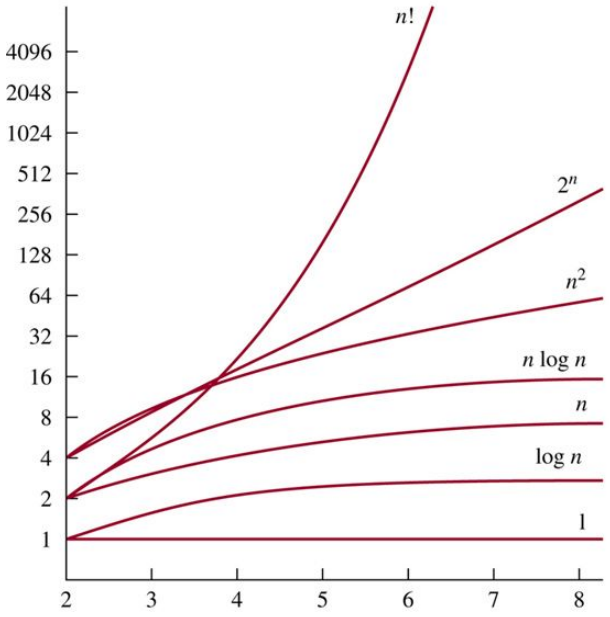
বিগিনারদের আরেকটি কমন ভুল হলো এভাবে কোড লেখা:

```
int myAlgorithm6(char *s)
{
	int c=0;
	for(int i=0;i<strlen(s);i++)
	{
		if(s[i]=='a') c++;
	}
	return c;
}
```

s স্ট্রিং এর দৈর্ঘ্য |s| হলে এখানে কমপ্লেক্সিটি হলো $O(|s|^2)$। কেন স্কয়ার হলো? কারণ strlen(s) ফাংশনের নিজের কমপ্লেক্সিটি হলো O(|s|), একে লুপের মধ্যে আরো O (|s|) বার কল করা হয়েছে। তাই strlen(s) এর মান আগে অন্য একটি ভ্যারিয়েবলের রেখে তারপর সেটা দিয়ে লুপ চালাতে হবে,তাহলে O(|s|) এ লুপ চলবে।

প্রোগ্রামিং কনটেস্টে আমরা ধরে নেই জাজ এর পিসি ১ সেকেন্ডে মোটামুটি 10^8 টা ইনস্ট্রাকশন রান করতে পারবে। এটা জাজ-পিসি অনুসারে কমবেশি হতে পারে,যেমন টপকোডার আরো অনেক বেশি ইনস্ট্রাকশন ১ সেকেন্ডে রান করতে পারে কিন্তু spoj বা কোডশেফ তাদের পেন্টিয়াম ৩ পিসি দিয়ে 10^7 টাও সহজে রান করতে পারেনা। অনসাইট ন্যাশনাল কনটেস্টে আমরা ১ সেকেন্ডে 10^8 ধরেই কোড লিখি। কোড লেখার আগে প্রথমে দেখবে তোমার অ্যালগোরিদমের worst case কমপ্লেক্সিটি কত এবং টেস্ট কেস কয়টা এবং দেখবে টাইম লিমিট কত। **অনেক নতুন প্রোগ্রামার অ্যালগোরিদমের কমপ্লেক্সিটি সঠিক ভাবে হিসাব করলেও টেস্ট কেস সংখ্যাকে গুরুত্ব দেয়না,এ ব্যাপারে সতর্ক থাকতে হবে।**

নিচের গ্রাফে বিভিন্ন কমপ্লেক্সিটির অ্যালগোরিদমের তুলনা দেখানো হয়েছে:



কনটেস্টে প্রবলেমের ইনপুট সাইজ দেখে অনেক সময় expected algorithm অনুমান করা যায়। যেমন n=100 হলে সম্ভাবনা আছে এটা একটা n^3 কমপ্লেক্সিটির ডিপি প্রবলেম,বা ম্যাক্সফ্লো-ম্যাচিং প্রবলেম। n=10^5 হলে সাধারণ nlogn কমপ্লেক্সিটিতে প্রবলেম সলভ করতে হয় তাই সম্ভাবনা আছে এটা একটা বাইনারি সার্চ বা সেগমেন্ট ট্রি এর প্রবলেম।

আজ এই পর্যন্তই। কমপ্লেক্সিটি নিয়ে আরো অনেক কিছু জানার আছে,বিস্তারিত পড়তে:

A Gentle Introduction to Algorithm Complexity Analysis

Complexity and Big-O Notation

Attacking Recursions and Practice Recursion

Zobayer Hasan, Solution Architect at TigerIT

Some general approach for solving recursive problems:

If anyone want to read an awesome super cool tutorial on recursion in BANGLA... read from Fahim Vai's page

#1: Forget about what you need to do, just think about any input, for which you know what your function should output, as you know this step, you can build up a solution for your problem. Suppose you have a function which solves a task, and you know, this task is somehow related to another similar task. So you just keep calling that function again and again thinking that, the function I'm calling will solve the problem anyhow, and the function will also say, I'll solve this problem, if you give me the result for another sub-problem first!!! Well, then you'll say, "So, why don't you use your twin-brother to solve that part?" and so on... The following example will show you how to start writing a recursive function.

Example: Think about computing n! recursively. I still don't know what and how my function will do this. And I also don't know, like what could be 5! or 7!... But nothing to worry about, I know that 0! = 1! = 1. And I also know that, n! = n(n-1)!. So I will think like that, "I will get n! from a function F, if some one gives me (n-1)!, then I'll multiply it with n and produce results". And, thus, F is the function for computing n!, so why don't I use again to get (n-1)! ? And when F tries to find out what could be the value of (n-1)!, it also stumbles at the same point, it wonders what would be the value of (n-2)!... then we tell him to use F again to get this... and this thing keeps

going as long as we don't know what F actually should return for any k. In case of k = 1, as we know now F can return 1 for k = 1, and it don't need to call another F to solve anything first...

```
int factorial(int n) {
  // I know this, so I don't want my function to go any further...
  if(n==0) return 1;
  // don't bother what to do, just reuse the function...
  else return n*factorial(n-1);
}
```

#2: What ever you can do with loops, can be done with recursions as well. A simple technique for converting recursions and loops is shown below:

Forward:

for loop:

```
for(int i = 0; i < n; i++) {
  // do whatever needed
}
```

Equivalent recursion:

```
void FOR(int i, int n) {
  if(i==n) return; // terminates
  // do whatever needed
  FOR(i+1, n); // go to next step
}
```

Backward:

for loop:

```
for(int i = n-1; i >= 0; i -= 1) {
  // do whatever needed
```

```
}
```

Equivalent recursion:

```
void ROF(int i, int n) {
  if(i==n) return; // terminates
  ROF(i+1, n); // keep going to the last
  // do whatever needed when returning from prev steps
}
```

Well, you may wonder how this is backward loop? But just think of its execution cycle, just after entering the function, it is calling itself again incrementing the value of *i*, and the execution routine that you have written under the function call, was paused there. From the new function it enters, it works the same way, call itself again before executing anything...Thus when you have reached the limiting condition, or the base condition, then the function stops recursion and starts returning, and the whole process can be shown as below...let n=5, and we want to print 5 4 3 2 1...code for this might be:

```
void function(int i, int n) {
  if(i<=n) {
     function(i+1, n);
     printf("%d ", i);
  }
}
```

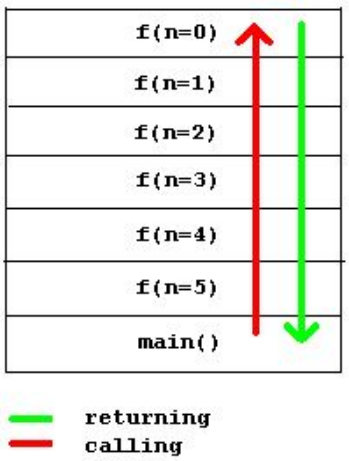
Explanation might look like this:

```
01|call function1 with i=1
02|    call function2 with i=2
03|        call function3 with i=3
04|            call function4 with i=4
05|                call function5 with i=5
06|                    call function6 with i=6
07|                        i breaks condition, no more calls
08|                    return to function5
09|                    print 5
```

```
10|                return to function4
11|                print 4
12|            return to function3
13|            print 3
14|        return to function2
15|        print 2
16|    return to function1
17|    print 1
18|return to main, done!
```

Left side number shows the execution steps, so from the above program, we get, 5 4 3 2 1. So indeed it ran on reverse direction...

#3: Use the advantage of call stack. When you call functions recursively, it stays in the memory as the following picture demonstrates.




```
int f(int n) {
  if(n==0) return 1;
  return n*f(n-1);
```

}

You know about stack, in a stack, you cannot remove any item unless it is the topmost. So you can consider the calling of a recursive function as a stack, where, you can't remove the memory used by f(n=3) before removing f(n=2) and so so... So, you can easily see that, the functions will hold all their variables and values until it is released. This actually serves the purpose of using an array.

#4: Be careful while using recursions. From a programming contest aspects, recursions are always best to avoid. As you've seen above, most recursions can be done using loops somehow. Recursions have a great deal of drawbacks and it most of the time extends the execution time of your program. Though recursions are very very easy to understand and they are like the first idea in many problems that pops out in mind first... they still bear the risks of exceeding memory and time limits.

Generally, in loops, all the variables are loaded at the same time which causes it a very low memory consumption and faster access to the instructions. But whenever we use recursions, each function is allotted a space at the moment it is called which requires much more time and all the internal piece of codes stored again which keeps the memory consumption rising up. As your compiler allows you a specific amount of memory (generally 32 MB) to use, you may overrun the stack limit by excessive recursive calls which causes a Stack Overflow error (a Runtime Error).

So, please think a while before writing recursions whether your program is capable of running under the imposed time and memory constraints. Generally recursions in O(lg n) are safe to use, also we may go to a O(n) recursion if n is pretty small.

#5: If the recursion tree has some overlapping branches, most of the times, what we do, is to store already computed values, so, when we meet any function which was called before, we may stop branching again and use previously computed values, which is a common technique knows as Dynamic Programming (DP), we will talk about that later as that is pretty advanced.

These techniques will be used in almost all problems when writing recursive solution. Just don't forget the definition of recursion:

Definition of recursion = See the Definition of recursion

Now try this

Practice Recursion

You may like to try out some simple problems to practice recursions. Try to solve all of them without using any global variables. And try on your own before looking at the solutions. Also please notify any error to me (zobayer1[at]gmail[dot]com).

Before looking at the problems, you may like to read this post about how to attack recursive problems.

Problem 1:

You will be given an array of integers, write a recursive solution to print it in reverse order.

Input:

5

69 87 45 21 47

Output:

47 21 45 87 69

see answer

Problem 2:

Write a recursive function to print an array in the following order.

[0] [n-1]

[1] [n-2]

.....

.....

[(n-1)/2] [n/2]

Input:

5

1 5 7 8 9

Output:

1 9

5 8

7 7

see answer

Problem 3:

Write a recursive program to remove all odd integers from an array. **You must not use any extra array or print anything in the function.** Just read input, call the recursive function, then print the array in main().

Input:

6

1 54 88 6 55 7

Output:

54 88 6

see answer

Problem 4:

Write a recursive solution to print the polynomial series for any input n:

$1 + x + x^2 + \ldots\ldots\ldots\ldots + x^{n-1}$

Input:

5

Output:

1 + x + x^2 + x^3 + x^4

see answer

Problem 5:

Write a recursive solution to evaluate the previous polynomial for any given x and n.

Like, when x=2 and n=5, we have $1 + x + x^2 + ... + x^{n-1} = 31$

Input:

2 5

Output:

31

see answer

Problem 6:

Write a recursive program to compute n!

Input:

5

Output:

120

see answer

Problem 7:

Write a recursive program to compute n^{th} fibonacci number. 1^{st} and 2^{nd} fibonacci numbers are 1, 1.

Input:

6

Output:

8

see answer

Problem 8:

Write a recursive program to determine whether a given integer is prime or not.

Input:

49

999983

1

Output:

not prime

prime

not prime

see answer

Problem 9:

Write a recursive function that finds the gcd of two non-negative integers.

Input:

25 8895

Output:

5

see answer

Problem 10:

Write a recursive solution to compute lcm of two integers. You must not use the formula lcm(a,b) = (a x b) / gcd(a,b); find lcm from scratch...

Input:

23 488

Output:

11224

see answer

Problem 11:

Suppose you are given an array of integers in an arbitrary order. Write a recursive solution to find the maximum element from the array.

Input:

5

7 4 9 6 2

Output:

9

see answer

Problem 12:

Write a recursive solution to find the **second maximum** number from a given set of integers.

Input:

5

5 8 7 9 3

Output:

8

see answer

Problem 13:

Implement linear search recursively, i.e. given an array of integers, find a specific value from it.

Input format: first n, the number of elements. Then n integers. Then, q, number of query, then q integers. Output format: for each of the q integers, print its index (within 0 to n-1) in the array or print 'not found', whichever is appropriate.

Input:

5

2 9 4 7 6

2

5 9

Output:

not found

1

see answer

Problem 14:

Implement binary search recursively, i.e. given an array of **sorted** integers, find a query integer from it.

Input format: first n, the number of elements. Then n integers. Then, q, number of query, then q integers. Output format: for each of the q integers, print its index (within 0 to n-1) in the array or print 'not found', whichever is appropriate.

Input:

5

1 2 3 4 5

2

3 -5

Output:

2

not found

see answer

Problem 15:

Write a recursive solution to get the reverse of a given integer. **Function must return an int**

Input:

123405

Output:

504321

see answer

Problem 16:

Read a string from keyboard and print it in reversed order. **You must not use any array to store the characters**. Write a recursive solutions to solve this problem.

Input:

helloo

Output:

oolleh

see answer

Problem 17:

Write a recursive program that determines whether a given sentence is palindromic or not just considering the alpha-numeric characters ('a'-'z'), ('A'-'Z'), ('0'-'9').

Input:

madam, i'm adam

hulala

Output:

palindromic

not palindromic

see answer

Problem 18:

Implement strcat(), stracpy(), strcmp() and strlen() recursively.

Input:

test on your own

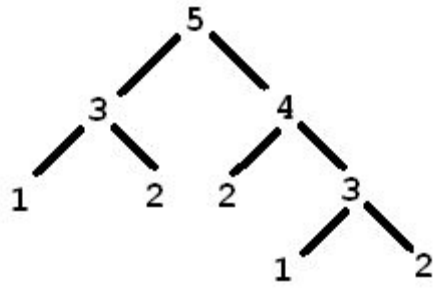
Output:

test on your own

see answer

Problem 19:

If you already solved the problem for finding the n^{th} fibonacci number, then you must have a clear vision on how the program flow works. So now, in this problem, print the values of your fibonacci function in pre-order, in-order and post-order traversal. For example, when n = 5, your program calls 3 and 4 from it, from the call of 3, your program calls 1 and 2 again..... here is the picture:



Input:

5

Output:

Inorder: 1 3 2 5 2 4 1 3 2

Preorder: 5 3 1 2 4 2 3 1 2

Postorder: 1 2 3 2 1 2 3 4 5

see answer

Problem 20:

All of you have seen the tower of Hanoi. You have 3 pillars 'a', 'b' and 'c', and you need transfer all disks from one pillar to another. Conditions are, only one disk at a time is movable, and you can never place a larger disk over a smaller one. Write a recursive solution to print the moves that simulates the task, a -> b means move the topmost of tower a to tower b.

Input:

3

Output:

a -> c

a -> b

c -> b

a -> c

b -> a

b -> c

a -> c

see answer

Good Luck!!!

রিকার্শন, সার্চিং এবং ডাইনামিক প্রোগ্রামিং

ইকরাম মাহমুদ, ঢাকা বিশ্ববিদ্যালয়, গুগল

সূচনা

ধরো, হঠাৎ একদিন তোমার সাথে আমার দেখা হলো ভূতের গলির সামনে। আর তুমি দেখলে যে আমি গলা ফাটিয়ে "ফাহিম! ফাহিম!" করে চিল্লাচ্ছি। তোমার নির্ঘাত মনে হবে আমাকে ভূতে ধরেছে। তুমি হয়তো কিছুক্ষণ বোকা বোকা হয়ে আমার দিকে তাকিয়ে থাকবে, আর ভাবতে শুরু করবে আমাকে কোন কবিরাজের কাছে নেয়া যায় ঝাড়ফুঁক করার জন্য। তারপর হয়তো তোমার বন্ধুটা পেট চেপে হাসতে শুরু করবে, আর বলবে - "আহা! ছেলেটা পুরাই রিকার্সিভ হয়ে গেছে!"

তখন হয়তো তুমি ভ্রু কুঁচকাবে, আর জিজ্ঞেস করবে, "রিকার্সিভ? সেটা কি খায় না মাথায় দেয়?"

রিকার্সিভ ফাংশন

একটা ফাংশন যখন নিজেই নিজেকে ডাকাডাকি করতে থাকে। তখন আমরা বলি ফাংশনটা রিকার্সিভ। কিন্তু সমস্যা হলো, ফাংশন জিনিসটা কি? আর সেটা শুধু ডাকাডাকি করবেই বা কেন? সে নিশ্চই কুরবানির ছাগল না, নাকি? :|

বেশ তো, আমি তোমাকে ফাংশন বুঝাবো। খালি একটা শর্ত! তোমাকে সুকুমার রায়ের একটা গল্প পড়তে হবে।

বেজায় গরম। গাছতলায় দিব্যি ছায়ার মধ্যে চুপচাপ শুয়ে আছি, তবু ঘেমে অস্থির। ঘাসের উপর রুমালটা ছিল, ঘাম মুছবার জন্য যেই সেটা তুলতে গিয়েছি অমনি রুমালটা বলল ``ম্যাও!" কি আপদ! রুমালটা ম্যাও করে কেন?

চেয়ে দেখি রুমাল তো আর রুমাল নেই, দিব্যি মোটা-সোটা লাল টক্*টকে একটা বেড়াল গোঁফ ফুলিয়ে প্যাট্ প্যাট্ করে আমার দিকে তাকিয়ে আছে!
আমি বললাম, ``কি মুশকিল! ছিল রুমাল, হয়ে গেল একটা বেড়াল।"

অমনি বেড়ালটা বলে উঠল, ``মুশকিল অবার কি? ছিল একটা ডিম, হয়ে গেল দিব্যি একটা প্যাঁক্*পেঁকে হাঁস। এ তো হামেশাই হচ্ছে।"

আমি খানিক ভেবে বললাম, ``তা হলে তোমায় এখন কি বলে ডাকব? তুমি তো সত্যিকারের বেড়াল নও, আসলে তুমি হচ্ছ রুমাল।"

বেড়াল বলল, ``বেড়ালও বলতে পার, রুমালও বলতে পার, চন্দ্রবিন্দুও বলতে পার।" আমি বললাম, ``চন্দ্রবিন্দু কেন?"

শুনে বেড়ালটা ``তাও জানো না?" বলে এক চোখ বুজে ফ্যাচ্*ফ্যাচ্ করে বিশ্রীরকম হাসতে লাগল। আমি ভারি অপ্রস্তুত হয়ে গেলাম। মনে হল, ঐ চন্দ্রবিন্দুর কথাটা নিশ্চয় আমার বোঝা উচিত ছিল। তাই থতমত খেয়ে তাড়াতাড়ি বলে ফেললাম, ``ও হ্যাঁ-হ্যাঁ, বুঝতে পেরেছি।"

তো ধরো, তোমার একটা রুমাল ছিল, তুমি দিলে একটা ফুঁ, ওমনি রুমালটা বিড়াল হয়ে গেলো। এখানে আমার ইনপুট হচ্ছে রুমাল, ফুঁ দেয়াটা হচ্ছে আমার প্রসেসিং, আর বিড়ালটা হচ্ছে আমার আউটপুট। তো আমরা খুব ভান করি আমরা খুব বুদ্ধিমান, সেজন্য আমরা খুব কম কথা বলি, আর এই কথাটা এভাবে লিখি -

বিড়াল = ফুঁ(রুমাল)

বেশিরভাগ মানুষ এত কষ্ট করে ফুঁ ও লিখতে যায় না, তারা লিখে

বিড়াল = f(রুমাল)

এটার মানে হচ্ছে আমি যদি একটা রুমালকে ধরে f মেরে দেই তাহলে আমি একটা বিড়াল পেয়ে যাবো। f মারা মানে কি, সেটা আমরা ধরে নিচ্ছি আমরা জানি। :)

মাঝে মাঝে এমন হয়, ধরো তুমি কাচ্চি বিরিয়ানি রাঁধতে চাচ্ছো, তো এখন রুমাল কে ধরে f মেরে দিলে তো কোন লাভ হবে না! কারণ রুমালকে f মেরে দিলে সেটা তো বিড়াল হয়ে যাবে, কাচ্চি বিরিয়ানি আসবে কোত্থেকে?

তো আমাদের করতে হবে কি, একটা আলু নিতে হবে, এক বাটি চাল নিতে হবে, তেল নিতে হবে, একটু লবণ নিতে হবে, আর একটা ছাগল চুরি করে আনতে হবে। তো ব্যাপারটা একই রকমই দাঁড়াবে শুধু f না মেরে একটা 'রান্না' মেরে দিতে হবে।

তো তখন ব্যাপারটা দাঁড়াবে এরকম -

কাচ্চি বিরিয়ানি = রান্না(আলু, চাল, তেল, লবণ, চুরি করা ছাগল)

ফাংশন, যেটা রিকার্সিভ

আচ্ছা তাহলে, আমরা এখন জানি ফাংশন কি। তো আসো আমরা একটা ফাংশন তৈরী করি।

sum(n) = 1 + 2 + 3 + ... + n

সুকুমার রায় যখন প্রথম এই ফাংশনটার চেহারাটা দেখেছিল, সে কি লিখেছিল জানো?

ভয় পেয়ো না, ভয় পেয়ো না, তোমায় আমি মারব না-
সত্যি বলছি কুস্তি ক'রে তোমার সঙ্গে পারব না।
মন্‌টা আমার বড্ড নরম, হাড়ে আমার রাগ্‌টি নেই,
তোমায় আমি চিবিয়ে খাব এমন আমার সাধ্যি নেই!
মাথায় আমার শিং দেখে ভাই ভয় পেয়েছ কতই না-
জানো না মোর মাথার ব্যারাম, কাউকে আমি গুঁতোই না?

তো সত্যি বলছি, এই ফাংশনটা দেখে ডরানোর কিচ্ছু নেই। এখানে আমরা ফুঁ মারার বদলে sum মারবো, আর ইনপুট হচ্ছে n , যেখানে n মানে যেকোন একটা সংখ্যা(১, ২, ৪২০, ৬৯, ০, ৩১৪) বা যেকোন কিছু - আর sum মারলে বিড়াল হওয়ার বদলে n টা ১ থেকে n পর্যন্ত সব সংখ্যার যোগফল হয়ে যাবে।

যেমন ধরো

sum(5) = 1 + 2 + 3 + 4 + 5 = 15

sum(6) = 1 + 2 + 3 + 4 + 5 + 6 = 21

sum(7) = 1 + 2 + 3 + 4 + 5 + 6 + 7 = 28

বুঝছো? হুমম?
না বুঝলে কিন্তু কামড়াবো!

একটা মজা দেখো, আমরা যদি 6 কে একটা sum মেরে তারপর সেটাকে sum(6) বানিয়ে তার সাথে 7 যোগ করে দেই, তাহলে আমরা sum(7) পেয়ে যাচ্ছি! ওয়াও! কি অদ্ভূত ব্যাপার! তাই না? কি আশ্চর্য ব্যাপার! :O একদমই ছিল রুমাল, হয়ে গেলো বিড়াল!

তাহলে দেখো, আমরা লিখতে পারি,
sum(7) = sum(6) + 7

আমরা এই জিনিসগুলোও তাহলে লিখতে পারি

sum(6) = sum(5) + 6
sum(5) = sum(4) + 5
sum(4) = sum(3) + 4
sum(3) = sum(2) + 3
...
sum(0) = 0 // :(বেচারা!

সমস্যা হলো, এভাবে ঘন্টার পর ঘন্টা সব সংখ্যা লেখার চেয়ে আমরা যেহেতু কম কথা বলি, আমরা লিখতে পারি
sum(n) = sum(n-1) + n
তবে যদি n = 0 হয় তবে সেটা ০

সি প্রোগ্রামিং ল্যাঙ্গুয়েজে আমরা লিখতে পারি

```
int sum( int n ) {
   if( n == 0 ) return 0;
   else return n + sum( n-1 );
}
```

এখন দেখো, এই ফাংশনটা করছে কি, নিজের নামের আরেকটা ফাংশনকে ডাকাডাকি করছে ওর কাজের একটা অংশ করে দেয়ার জন্য। তাই না? এটাকেই আমরা বলি রিকার্শন।

তো সহজ কথায়, যখন একটা ভালো(!) মানুষ ফাংশনের ঘাড়ে একটা ভূত চেপে বসে আর সেটা নিজের নাম ধরে ডাকাডাকি শুরু করে দেয়, তখন সেটাকে বলে রিকার্শন।

কিছু উদার উদাহরণ
ফিবোনাকি সংখ্যা

অনেক অনেক দিন আগে ইতালিতে একটা বিশাল মাথাওয়ালা গণিতবিদ থাকতো যার নাম ছিল ফিবোনাকি। ফিবোনাকির ছিল অনেক ঝাকড়া চুল, আর সেখানটায় নাকি অনেক গাবদা গাবদা উকুন ছিল। তো সে করতো কি খুব মাথা চুলকাতো, আর সারাদিন বসে বসে ফ্যাঁচ ফ্যাঁচ করে কাঁদতো। তো একদিন সে গেলো গঙ্গা নাপিতের দোকানে। সেখানে গিয়ে সে বগল

কামালো আর মাথার সব চুল ফেলে দিলো আর তারপর মুহাহাহাহাহা করে একটা বিশাল বিটকেলে হাসি দিয়ে লুঙ্গি হাতে ধরে নাঁচতে নাঁচতে বাসায় ফিরে আসলো।

কিন্তু উকুন ফেলে দিলে কি হবে, আকাশে একটা শকুন উড়ছিল মিগ২৯ নিয়ে। তো সেই শকুনটা করলো কি ওর মিগ২৯ নিয়ে পুঁউউউউ করে উড়ে আসলো আর একটা ছোট্ট অণূবিক্ষণিক মিসাইলের ডগায় এক জোড়া উকুন নিয়ে সেটা শিউউউ করে ছুঁড়ে মারলো ফিবোনাকির মাথায়।

তো প্রথম মাসে ফিবোনাকির মাথায় ছিল ১ জোড়া উকুন। (সবমিলে ১)
প্রথম মাসের শেষে ১ জোড়াই থাকলো। (সবমিলে ১)
দ্বিতীয় মাসের শুরুতে সেই জোড়া করলো কি, আরো ১ জোড়া উকুনের জন্ম দিলো।
তো সেই মাসের শেষে ২ জোড়া উকুন থাকলো। (সবমিলে ২)
তৃতীয় মাসে এই ২ জোড়া করলো কি আরো ২ জোড়া জন্ম দিলো, আর মাসের শেষে প্রথম জোড়া মরে গেলো কারণ কোন উকুন ২ মাসের বেশি বাঁচে না।
তো সেই মাসের শেষে ৩ জোড়া উকুন থাকলো। (সবমিলে ৩)

তুমি যদি এভাবে হিসেব করতেই থাকো, তাহলে দেখবে পরের মাসে ৫ জোড়া উকুন হয়, তার পরের মাসে ৮ জোড়া।

তো জিনিসটা দাঁড়ায় এরকম ১, ১, ২, ৩, ৫, ৮, ১৩, ২১, ৩৪, ...

তো ফিবোনাকি খুব মন খারাপ করে বসে বসে উকুনের কামড় খেতেই থাকলো তো খেতেই থাকলো। ন্যাড়া হবার পরদিন বেলতলায় যাওয়ার পর ওর মাথায় একটা বেল পড়েছিল বলে সে আবার চুল সব ফেলে দেয়ার সাহস পেলো না। তো সে বসে বসে কামড় খেয়ে খেয়ে উকুন গুনতে গুনতে হঠাৎ দেখলো, আরি আজিব! আগের দুই মাসের উকুন সংখ্যা যোগ করলেই এই মাসে মাথায় কয়টা উকুন থাকবে সেটা বের হয়ে যায়। তো সে ইয়ে ইয়ে ইয়ে বলে নাচানাচি শুরু করলো, আর ওর নাচ দেখে সবাই ভাবলো সে কি না কি আবিষ্কার করে ফেলেছে - তো সেজন্য সে খুব খুব বিখ্যাত হয়ে গেলো।

আসলে ব্যাপারটা তেমন কঠিন কিছু না, এই মাসে যারা বেঁচে থাকবে, তাদের বয়স যেহেতু ২ এর কম হতে হবে, তাহলে অবশ্যই তাদের জন্ম ঠিক আগের মাসে, অথবা তার আগের মাসের আগের মাসে হতেই হবে তাই না?

তাহলে দেখো, এখানেও একটা রিকার্শন আছে।

fibonacci(5) = fibonacci(4) + fibonacci(3)

তাই না? এটার জেনারেল ফর্ম হচ্ছে

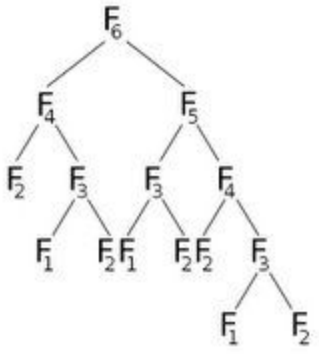
fibonacci(n) = fibonacci(n-1) + fibonacci(n-2)

তবে, যদি n = 0 অথবা n = 1 হয় তাহলে fibonacci(n) = 1

এটা আমরা সি তে লিখলে সেটা হবে -

```
int fibonacci( int n ) {
   if( n == 0 || n == 1 ) return 1;
   else return fibonacci( n-1 ) + fibonacci( n-2 );
}
```

এখন সমস্যা হলো কি - যখনই আমার 6 কে ফিবোনাকি মারতে হচ্ছে, তখন সে 5 আর 4 এর ফিবোনাকিকে ডাকছে, আবার 5 ডাকছে 4 আর 3 এর ফিবোনাকিকে। ব্যাপারটা অনেকটা এরকম হচ্ছে। (এখানে ০ মাস বলে কিছু নেই - ১ থেকে ওর গোনা শুরু করেছে)



From Wikipedia, the free encyclopedia

একবার তাকাও, দেখো f(2) কে সেই রকম ভাবে ডাকাডাকি করা হচ্ছে। পাঁচ পাঁচ বার! আবার f(3) কে ডাকা হচ্ছে তিন তিন বার! তো এখানে অনেক অনেক বার ডাকাডাকি করা হচ্ছে, আর কাওকে ডাকাডাকি করলে সে আরো অন্য অনেককে ডাকাডাকি করছে। তো প্রচুর বার ডাকাডাকি করে সবাই খুব ক্লান্ত হয়ে যাচ্ছে, আর আমাদের প্রচুর ভালো ভালো সময় নষ্ট হচ্ছে, যেগুলো দিয়ে আমরা প্রচুর কার্টুন দেখতে পারতাম আর অনেক অনেক গল্পের বই পড়তে পারতাম। তুমি যদি f(10) কে ডাকাডাকি করো তাহলে দেখবে সে f(2) কে ডাকতে ডাকতে মাথাই খারাপ করে ফেলছে।

তো আমরা একটা কাজ করতে পারি, আমরা যদি মনে রাখি f(3) এর মান কতো, আর সেটা লিখে রাখি, তাহলে যখনই কেউ f(3) কে ডাকবে, সে পাগলের মতো ওর মান জানার জন্য f(2) আর f(1) কে বারবার ডাকাডাকি করবে না। তাই না?

তাহলে আমরা প্রোগ্রামটা এভাবে লিখতে পারি -

```
int array[20];
int fibonacci( int n ) {
   if( n == 0 || n == 1 ) return 1;
   else if( array[n] == 0 ) // ami oke ekbaro process kori nai - or khub mon kharap! :(
      array[n] = fibonacci( n-1 ) + fibonacci( n-2 ); // ye! o khushi! :D

   return array[n];
```

}
এটা আর আগেরটার একটাই পার্থক্য - আমি যেহেতু একবার প্রসেসিং এর পরই array[n] এর মধ্যে লিখে রাখছি fibonacci(n) এর মান - আমার বারবার সেটা প্রসেস করা লাগছে না। আমার সময় বেঁচে যাচ্ছে কার্টুন দেখার জন্য।

এই যে কান্ডটা আমরা এই মাত্র করলাম, এটার একটা গাল ভরা নাম আছে - ডাইনামিক প্রোগ্রামিং। যখন কেউ নতুন ডাইনামিক প্রোগ্রামিং শেখে, সে টানা কয়েকদিন ধেই ধেই করে নাচানাচি করে, আর সবার সাথে ডাইনামিক প্রোগ্রামিং নিয়ে কথা বলতে চায়। তারপর ঘুমাতে গেলে রিকার্শন স্বপ্ন দেখে, আর ঘুমের ঘোরে বিড়বিড় করে, ডাই ডাই-নামি-ক প্রো-প্রো-গ্রামিং! ইয়ে! গুড্ডুড গুড্ডুড!

ডেপ্থ ফার্স্ট সার্চ

সুপার রোমিও মার্চের শুরুতে টুম্ব রেইডার লারা ক্রফটের প্রেমে পড়লো। কিন্তু লারা ক্রফট ছেলেদেরকে যত না ভালোবাসতো তারচে' বেশি ভালোবাসতে কবরের মধ্যে গিয়ে দৌড়াদৌড়ি করতে। সুপার রোমিও তবুও অনেক আশা নিয়ে চুইঙ্গামের মত লেগে থাকলো, কিন্তু এর কিছুদিন পর লারা ক্রফটের সাথে সুপারম্যানের ইয়ে হয়ে গেলো। সুপার রোমিও মুখ বাঁকা করে বসে থাকলো, কারণ এবার সে বুঝতে পারলো ওর আর কোন চান্সই নেই! :'(

এদিকে হলো কি, ওর চুপচাপ গুটিসুটি মেরে উদাস উদাস হয়ে বসে থাকা দেখে বিলকিস বাণু ওর প্রেমে পড়ে গেলো। কারণ বিলকিস বাণু কবিতা পড়তে খুব ভালোবাসে আর ওর ধারণা একটু দেবদাস দেবদাস টাইপের ছেলে না হলে নাকি প্রেম জমে না। কিন্তু সুপার রোমিও তো বিলকিস বাণুর সাথে প্রেম করবেই না। তো সে করলো কি, শহরে পালিয়ে গেলো - মতিঝিলে গিয়ে এফিডেফিট করে নিজের নাম পাল্টে রাখলো, সুপার মারিও। আর সেই ফাঁকে সে দেখলো একটা বিশাল বিলবোর্ডে বিপাশা বসু একটা সাবান নিয়ে 'কিছু একটা' করছে একদম সব্বার সামনে(চিচিচি!), ঠিক রাস্তার উপরে!(চিচিচিচি!) তো সেই 'কিছু একটা' দেখে তার মনে প্রেম প্রেম ভাব হলো, সে ঠিক করলো বিপাশা বসুকে গিয়ে বলবে ওর সাথে প্রেম করতে।

কিন্তু সমস্যা হলো, বিপাশা বসু থাকে হচ্ছে প্যান্ট ঢিলা মাস্তানের পাড়ায়। এবং কেউ একজন বিপাশা বসুর সাথে প্রেম নিবেদন করতে গেলেই সে ধরে কষে একটা মাইর দিয়ে দেয়। এইজন্যই নাকি সুপারম্যান কখনো বিপাশা বসুর কাছাকাছি যেতে পারেনি। তার ছিঁড়া মজনু এসে বলল ওর কাছে নাকি একটা ম্যাপ আছে, সেখানে পরিষ্কার করে লেখা আছে কোথায় কোথায় দেয়াল আর কোথায় কোথায় প্যান্ট ঢিলা মাস্তানের চ্যালারা অপেক্ষা করছে, আর কোথায় বিপাশা বসু।

ম্যাপটা দেখে বলতে হবে, সুপার মারিও বিপাশা বসুর কাছাকাছি যেতে পারবে কিনা। আর যদি পারে, তাহলে সে যত তাড়াতাড়ি সম্ভব তত তাড়াতাড়ি প্রেম নিবেদন করতে চায়। কারণ সে শুনেছে স্পাইডারম্যানও নাকি কাছে ধারেই আছে। সে যদি আগে এসে চামে চামে প্রেম নিবেদন করে দেয়, তাহলে বিপাশা বসুর কাছে সুপার মারিও কোন বেইল ই পাবে না!

সুপার মারিও চিৎকার করে বলল, দেখাও তোমার ম্যাপ! তার ছিঁড়া মজনু ওর পিছনের পকেট থেকে একটা ময়লা গ্রাফ পেপার বের করলো, সেটাতে লেখা ছিল

MX.....XB.
..X..X.X..
...X.X.X..
.....X....

তারপর তার ছিঁড়া মজনু করলো কি, এদিক ওদিক তাকিয়ে তারপর ফিসফিস করে তোমাকে ওর ম্যাপটা বুঝিয়ে দিলো - M টা হলো মজনুর বাড়ি, যেখানে ওরা এখন বসে বসে ফন্দি করছে, আর X গুলো হলো প্যান্ট ঢিলা মাস্তানের চ্যালাদের আড্ডা মারার চায়ের দোকান, ভুলেও ওদের কাছাকাছি গেলে ওরা ধরে স্ক্রু টাইট করে দিবে। আর . গুলোতে কোন সমস্যা

নাই - ওখান দিয়ে হাঁটা যাবে। কিন্তু এই পাড়ার নিয়ম হচ্ছে সব কিছুতে সোজাসুজি করতে হবে, কোণাকুনি হাঁটা যাবে না। ও কোণাকুনি হাঁটে এটা শুনলে কিছুতেই বিপাশা বসু ওকে বিয়ে করবে না।
তার ছিঁড়া মজনুকে সুপার মারিও জিজ্ঞেস করলো, কতক্ষণ হাঁটতে হবে। সে বলল, প্রতিটা . পার হতে এক মিনিট লাগবে। তারপর সে জানতে চাইলো, আসলে কখনো পৌছাতে পারবো তো। তখন তার ছিঁড়া মজনু ভাবতে শুরু করলো, কিন্তু অনেকক্ষণ মাথা চুলকিয়েও সে কিছুতেই বের করতে পারলো না কিছু। সুপার মারিও খুব বিরক্ত হয়ে ঘন্টাখানেক ওর মাথা চুলকানো দেখলো, তারপর তোমার কাছে চলে আসলো, কারণ তুমি নাকি কম্পিউটার দিয়ে খুটখাট করে কঠিন কঠিন সব কাজ করে ফেলতে পারো।

হুমম, এখন দেখো আমাদের কি করতে হচ্ছে। মজনু তার ঘর থেকে উপরে, বামে, ডানে আর নিচে যেতে পারছে। আমরা যদি সাধারণ গ্রাফ পেপারের মতো নাম্বারিং করি তাহলে ঘরগুলোর কো অর্ডিনেট হচ্ছে এরকম

(0,0) (0,1) (0,2) ....
(1,0) (1,1) (1,2) ....
(2,0) (2,1) (2,2) ....
.
.
.

মানে যদি কোন ঘর (i,j) হয় তাহলে ওর উপর নিচের ঘরগুলোর কো অর্ডিনেট এরকম হবে।

(i-1 , j-1) (i-1 , j) (i-1 , j+1)
(i , j-1) (i , j) (i , j+1)
(i+1, j-1) (i+1 , j) (i+1, j+1)

খেয়াল করো, মারিও কিন্তু প্রতিটা ঘরে গিয়ে একই কাজ করবে, উপরে যাবার চেষ্টা করবে, বামে যাবার চেষ্টা করবে, ডানে যাবার চেষ্টা করবে, তারপর নিচে যাবার চেষ্টা করবে - যদি কখনো দেখে সে চায়ের দোকানের দিকে এগোচ্ছে সে ফিরে আসবে, তারপর আবার বাকি দিকগুলো দেখবে।

যেহেতু সে একই কাজ বারবার করছে সেহেতু আমরা একটা ফাংশনকে রিকার্সিভলি ডেকেই পুরো কাজটা করতে পারি তাই না? তাহলে জিনিসটা দাড়াবে এরকম -

```
char map[100][100];
int found = 0;
void check( int i, int j ) {
   if( map[i][j] == 'B' ) found  = 1; // mario can reach bipasa
   else if( map[i][j] == 'X' ) return;
   else {
      check(i-1,j);
      check(i,j-1);
      check(i,j+1);
      check(i+1,j);
   }
}
```

মূল ব্যাপারটা অনেকটা এরকমই - তবে তোমার চেক করতে হবে তুমি কখনো বাউন্ডারির বাইরে চলে যাচ্ছো কিনা। সেজন্য ফাংশনটায় এরকম একটা লাইন জুড়ে দিতে হবে।

```
if( i < 0 || j < 0 || i >= totalrow || j >= totalcol ) return;
```

এই খোঁজাখুঁজির ব্যাপারগুলোকে বলা হয় সার্চিং আর এই মাত্র আমরা রিকার্সিভ ভাবে যে সার্চটা করলাম এটার নাম হচ্ছে Depth First Search কারণ সে যখনই দেখছে সে ঘরটাতে ঢুকতে পারছে, সে সেই ঘরটা থেকে আরো গভীরে গিয়ে খোঁজাখুঁজি করছে।

রিকার্শন হচ্ছে প্রোগ্রামিং এর সবচে' গুরুত্বপূর্ণ টপিকগুলোর একটা। অনেককিছুই সহজে করার জন্য আমাদের রিকার্শন ব্যবহার করতে হয়। রিকার্শনের পরিষ্কার ধারণা না থাকলে আসলে অনেককিছুই খুব কঠিন হয়ে যায়।

এই জন্যই আমার এত বেশি বকবক করা! :D

রিকার্শন সম্বন্ধে আরো পড়তে পারো জোবায়ের হাসানের ব্লগে, Attacking Recursion এবং তারপর Practice Recursion এ একগাদা প্রবলেম দেয়া আছে সলুশন সহ।

আর টপকোডারের টিউটোরিয়ালও পড়ে দেখতে পারো।

শুভ কামনা! ;)

FAQ of Competitive Programming/Programming Contest (কম্পিটিটিভ প্রোগ্রামিং/প্রোগ্রামিং কন্টেস্ট এর সচরাচর জিজ্ঞাস্য)

Hasnain Heickal (Jami), Assistant Professor, CSE, DU.

অনলাইনে বিভিন্ন ফোরামে প্রায়শই বেশ কিছু প্রশ্ন ঘুরে ফিরে অনেকেই জিজ্ঞেস করেন। কম্পিটিটিভ প্রোগ্রামিং এর ব্যাপারে এসব কিছু প্রশ্ন (এবং কিছু সম্পূরক প্রশ্ন) এর উত্তর দেওয়ার জন্য এই ব্লগ। এই ব্লগের পুরো অংশেই ধারণা করে নেওয়া হচ্ছে যে প্রশ্নকর্তা প্রোগ্রামিং শিখতে চান কম্পিটিটিভ প্রোগ্রামিং বা কন্টেস্ট করার জন্য। অন্যান্য ক্ষেত্রে এই পরামর্শগুলো যথাযথ নাও হতে পারে।

(আমার সংক্ষিপ্ত পরিচয়ঃ জাজিং ডিরেক্টর, আইসিপিসি ঢাকা রিজিওনাল ২০১৯। টীম লিডার, বাংলাদেশ দল, আই ও আই, ২০১৪। কো-অর্ডিনেটর, একাডেমিক টীম, জাতীয় হাই স্কুল প্রোগ্রামিং প্রতিযোগিতা, ২০১৬।)

প্রশ্ন ১ঃ *আমি প্রোগ্রামিং খুব একটা পারি না, কিভাবে শুরু করা উচিৎ? কোন ল্যাঙ্গুয়েজ শেখা শুরু করা উচিৎ?*

উত্তরঃ প্রোগ্রামিং এর বিভিন্ন ল্যাঙ্গুয়েজগুলো মধ্যে আসলে অনেক সাদৃশ্য আছে। যার কারণে একটি ল্যাঙ্গুয়েজ শিখলে অন্যটি শিখতে খুব বেশি সময় দরকার হয় না। তবে অবশ্যই কিছু স্পেশাল ল্যাঙ্গুয়েজ আছে, যেগুলো আমাদের মূলধারার প্রোগ্রামিং ল্যাঙ্গুয়েজগুলো থেকে অনেকখানিই আলাদা। যদি আপনার খুব বেশি কঠিন না মনে হয়ে থাকে, তাহলে আপনার সি/সি++ শেখা উচিৎ। কম্পিটিটিভ প্রোগ্রামিং এর জন্য এটি সবচেয়ে বহুল ব্যবহৃত এবং বেশিরভাগ বিচারক এই ল্যাঙ্গুয়েজেই কোড করে থাকেন। যদিও অন্যান্য ল্যাঙ্গুয়েজ যেমন, জাভা বা পাইথনও বিভিন্ন প্রোগ্রামিং কন্টেস্টে ব্যবহার করা হয়ে থাকে, কিন্তু সি/সি++ সার্বজনীন ভাবে প্রায় সব মূল ধারার প্রোগ্রামিং কন্টেস্টে ব্যবহৃত হয়। কাজেই আপনি যদি এর মধ্যেই অন্য কোন ল্যাঙ্গুয়েজ শেখার জন্য সময় না দিয়ে থাকেন, তাহলে সি/সি++ শেখাটাই হবে বুদ্ধিমানের কাজ।

প্রশ্ন ২ঃ *আমি সি/জাভা/পাইথন পারি। আমার কি সি++ শেখা উচিৎ?*

উত্তরঃ সি/জাভা/পাইথন ইত্যাদি ল্যাঙ্গুয়েজ দিয়ে কম্পিটিটিভ প্রোগ্রামিং করা খুবই সম্ভব। তবে কন্টেস্টে এর ক্ষেত্রে সি++ সবচেয়ে জনপ্রিয় ভাষা হওয়ার পিছনে কিছু কারণ রয়েছে। সি++ আসলে একমাত্র ল্যাঙ্গুয়েজ যেটি একই সাথে অনেক দ্রুত (প্রায় সি এর মত) এবং যার STL এর মত সমৃদ্ধ লাইব্রেরী আছে (জাভা/পাইথনের মত)। আসলে দুই ক্ষেত্রেই এটি একটি মধ্যম অবস্থা যার কারণে কন্টেস্ট্যান্টরা দ্রুত উন্নতি করতে পারে। সি বা জাভার সাথে সি++ এর সিনট্যাক্সের প্রচুর মিল থাকায়, শিখতে অতটা সময় লাগে না। পাইথনের সাথে যদিও মিল কম, কিন্তু পাইথনের ধীরগতি আসলে অনেক সময়ই কন্টেস্টের জন্য বেশ অসুবিধাজনক।

তবে আমি সি++ শিখতে জোর দিয়ে চাই মূলত একটি কারণে। অনলাইনে কম্পিটিটিভ প্রোগ্রামিং এর যত রিসোর্স পাওয়া যায় তার ৯৫ ভাগের বেশি আসলে সি++ এর কথা মাথায় রেখে লেখা। কাজেই সি++ কোডাররা এইসব ক্ষেত্রে একটু অতিরিক্ত সুবিধাভোগী হন বলে আমার ধারনা।

প্রশ্ন ৩ঃ *সি++ এ আমার কি কি শেখা উচিৎ?*

উত্তরঃ প্রথম ধাপে, ভ্যারিয়েবল, ইফ-এলসইফ, লুপ, অ্যারে এবং স্ট্রিং পর্যন্ত শেখাই যথেষ্ঠ। এগুলো জানলে এবং ভালমত ব্যবহার করতে পারলে কন্টেস্টের প্রায় সব প্রব্লেমই সমাধান করা সম্ভব। দ্বিতীয় ধাপে, গুছিয়ে প্রোগ্রাম লেখার জন্য ফাংশন, স্ট্রাকচার ও STL (Standard Template Library) শেখা দরকার। তৃতীয় ধাপে, কিছু ডাটা স্ট্রাকচারের জন্য পয়েন্টার এবং ডায়নামিক মেমোরি অ্যালোকেশন শেখা দরকার হতে পারে।

প্রোগ্রামিং এর অনেক উচ্চতর ধারণা আসলে কম্পিটিটিভ প্রোগ্রামিং এ ব্যবহার করে খুব একটা সুবিধা পাওয়া যায় না। যেমনঃ অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং, ফাংশনাল প্রোগ্রামিং ইত্যাদি। যেহেতু কন্টেস্টের বেশিরভাগ প্রোগ্রাম আসলে তুলনামূলক ছোট হয় কাজেই সেগুলোতে এইসব কৌশল ব্যবহার করার উপযোগিতা কম। এজন্যই সি++ এর ক্লাস বা অবজেক্ট ওরিয়েন্টেড বিষয়গুলো না জানলেও চলবে। সি++ এর সি সাবসেটটুকু এবং STL জানাটাই যথেষ্ঠ হবে।

প্রশ্ন ৪ঃ *সি++ কোন বই থেকে শিখলে সব থেকে ভাল হবে?*

উত্তরঃ বাংলা ইংরেজী যে কোন বই থেকেই শিখতে পারেন। বা অনলাইনেও সার্চ করে শিখতে পারেন। তবে একটু শেখার পর পরই নিজে নিজে সমস্যা সমাধানে গুরুত্ব দিতে হবে। না হলে শুধু পড়ে গেলে কোন লাভ হবে না। বাংলায়, তামিম শাহরিয়ার সুবিনের বইটি শুরু করার জন্য বেশ কাজের। এছাড়া eshikkha.net এও বাংলায় টিউটোরিয়ালের পাশাপাশি সমস্যা সমাধানের সুযোগ রয়েছে। ইংরেজী বই এর ক্ষেত্রে অনেকেই Herbert Schildt এর বইটি পছন্দ করেন।

প্রশ্ন ৫ঃ *কন্টেস্ট কখন এবং কিভাবে শুরু করা উচিৎ? আমার কি আগে প্রোগ্রামিং ভাষা পুরোপুরি আয়ত্ব করে তারপর কন্টেস্ট শুরু করা উচিৎ?*

উত্তরঃ কন্টেস্ট শুরু করার জন্য সবথেকে ভাল উপায় হল কোন একটা অনলাইন জাজ ব্যবহার শুরু করা। নিয়মিত প্রোগ্রামিং কন্টেস্ট হয় এরকম কিছু অনলাইন জাজ হলঃ

- codeforces.com
- codechef.com
- atcoder.com

তবে শুরুতে এই জাজগুলোতে কন্টেস্ট করতে গেলে কঠিন মনে হওয়াটা খুব স্বাভাবিক। কাজেই যারা নতুন শুরু করছেন, তারা নিচের কোন একটি “সহজ” অনলাইন জাজ দিয়ে শুরু করতে পারেনঃ

- eshikkha.net
- urionlinejudge.com.br
- dimikoj.com
- algo.codemarshal.org/problems

এর বাইরেও বেশ কিছু অনলাইন জাজ আছে, যেগুলোতে প্রচুর প্র্যাকটিস প্রব্লেম রয়েছে। ICPC বা সমমানের প্রতিযোগিতাগুলোতে ভাল করতে চাইলে এই জাজগুলোতে নিয়মিত প্র্যাকটিস করা বেশ জরুরী। এই জাজগুলো হলঃ

- lightoj.com
- onlinejudge.org
- spoj.com

- acm.timus.ru

প্রোগ্রামিং এর সামান্য ধারণা হওয়া মাত্রই কোন একটি অনলাইন জাজ থেকে সমস্যা সমাধান করা শুরু করা দরকার। সে ক্ষেত্রে শুরুতে আপনি “সহজ” অনলাইন জাজগুলো দিয়ে শুরু করতে পারেন। আস্তে আস্তে স্কিল এবং কনফিডেন্স বৃদ্ধি পেলে তখন অন্যান্য অনলাইন জাজ থেকে সমস্যা সমাধান ও কন্টেস্ট করা শুরু করতে পারেন।

প্রোগ্রামিং ভাষা ভালভাবে আয়ত্ব করার জন্যই আসলে বিভিন্ন জাজে সমস্যা সমাধান করা খুব জরুরী। কাজেই দেরি না করে, যত দ্রুত পারা যায় শুরু করে দেওয়াই উত্তম।