রিকার্সিভ প্রোগ্রাম ও রিকার্সিভ অ্যালগরিদম

By তানভীর | জুলাই 9, 2014 | Categories: কম্পিউটার, প্রোগ্রামিং | 19 Comments

একটা সেমিনার শুরু হতে যাচ্ছে। তুমি বসে আছো দর্শক সারিতে। পুরো হলে তিল ঠাই আর নাহিরে অবস্থা। কিন্তু তখনো বক্তৃতা শুরু হয় নি। হঠাৎ কৌতূহল হলো তোমার সিটটা প্রথম থেকে কত নাম্বার সারিতে সেটা জানতে। কিন্তু বসে বসে এতগুলো সারি গোনা খুবই বোরিং। তাহলে উপায়? খুবই সহজ, স্রেফ তোমার সামনের সারির একজনকে জিজ্ঞেস করো সে কত নাম্বার সারিতে বসে আছে। যদি সে বলে $latex n$ তাহলে তোমার নিজের সারি হবে $latex (n+1)$। তাহলে কি তোমার অনুরোধে সামনের ব্যক্তি বসে বসে আগের সারিগুলো গুণবে? নাহ! সেও স্রেফ তার সামনের জনকে জিজ্ঞেস করবে, কত নাম্বার সারিতে আছে। এবং উত্তরটা পেয়ে তার সাথে ১ যোগ করে তোমাকে জানাবে। এভাবে প্রশ্ন এগোতে এগোতে একেবারে প্রথম সারিতে বসা কারো কাছে পৌছাবে। সেই প্রথম সারির দর্শকের সামনে আর কেউ নেই। তাই সে আর কোনো গোণার ঝামেলা না করে পিছনের জনকে উত্তর দেবে, “আমি আছি ১ নাম্বার সারিতে”। তার পিছনের জন সেই উত্তরের সাথে ১ যোগ করে জানিয়ে দেবে তারও পিছনের জনকে। এভাবে উত্তরগুলো ১ যোগ হতে হতে এসে পৌছাবে তোমার কাছে। এবং তুমি জেনে যাবে তোমার নিজের সারি নাম্বার।

এই যে পুরোটা কাজটা নিজে না করে, আংশিক অন্যকে দিয়ে করানো।(যেখানে অপর জনও ঠিক তোমার মত করেই আরো অন্য

কাউকে দিয়ে কাজের বাকি অংশ করিয়ে নেবে)। এই পদ্ধতিকে বলে রিকার্সিভ পদ্ধতি।

কম্পিউটার বিজ্ঞানের সবচেয়ে মজার ধারণাগুলোর একটা হচ্ছে এই রিকার্সন। তবে আগেই বলে রাখি পৃথিবী আর দশটা মজাদার জিনিসের মত রিকার্সনও সহজসাধ্য কিছু নয়। এই ধারণাটা শিখতে বেগ পায়নি এমন কোনো কম্পিউটার বিজ্ঞানী বা প্রোগ্রামার নেই। তাই লেখাটির কোন পর্যায়ে যদি মনে হয় কিছু কিছু ব্যাপার ঘোলাটে মনে হচ্ছে, তখন হতাশ হবার কিছু নেই। ব্যাপারটাকে একটু সময় দিলেই দেখা যাবে হঠাৎ সব পরিষ্কার হয়ে গেছে। এটা অনেকটা সাইকেল চালানো, বা সাঁতার শেখার মত। লেগে থাকা ছাড়া যা শেখার আর কোনো সহজ উপায় নেই।

তবে শুরুতেই এত সব সতর্কবানী শুনে ভড়কে যেও না। কারণ, আমরা এগোবো খুব ছোটো ছোটো ধাপে। এবং চলার পথে গণিত থেকে শুরু করে প্রোগ্রামিংএর যেসব ধারণা আমাদের লাগবে সেগুলো আমরা অল্প করে এখানেই আলোচনা করবো। তাহলে শুরু করা যাক। শুরুতেই ফাংশন।

ফাংশন

ফাংশনের গাণিতিক সংজ্ঞা অথবা ফাংশন ধারনাটা সি, বা জাভার মত প্রোগ্রামিং ল্যাঙ্গুয়েজে কিভাবে কাজ করে তার খুটিনাটি নিয়ে আলোচনা করা আমাদের উদ্দেশ্য নয়। আমরা স্রেফ রিকার্সিভ অ্যালগরিদম জিনিসটা রিকার্সিভ ফাংশনের মাধ্যমে শিখতে চাই। আর এজন্য, ফাংশনকে আমরা একটু অন্যভাবে দেখার চেষ্টা করবো।

শুরুতেই একটা বাস্তব সংখ্যার গাণিতিক ফাংশন দেখি,

$latex f(x) = x\times x$ ... ... (1)

$latex x = 5$ এর জন্য এই ফাংশনটার মান বা আউটপুট হয়,

$latex f(5) = 5\times 5 = 25 $

একই ভাবে $latex x$ এর যে কোনো মানের জন্য $latex f(x)$ এর মানটা বের করা যায়। এখানে $latex x$ এর যে মানটা আমরা বসাচ্ছি, সেটাকে যদি বলি ইনপুট। তাহলে ফাংশন $latex f$ এর আউটপুট $latex f(x)$ যার মান ইনপুট এর বর্গ এর সমান।

ফাংশনের ইনপুট যে সংখ্যাই হতে হবে তেমন কোনো কথা নেই। আমরা এমন একটা ফাংশন ভাবতে পারি যার ইনপুট হচ্ছে একটা

› বাংলাদেশে বিজ্ঞানমনস্কতার প্রকৃত দ্বন্দ্ব: ধর্ম, নাকি রাষ্ট্র পরিকাঠামোর অবউন্নয়ন?
প্রকাশনায় CHANDRA DEB

› ইসলামের জন্ম, বিকাশ ও প্রাসাদ ষড়যন্ত্র (পর্ব ৪)
প্রকাশনায় খালেদ

› সোলাইমানের সাপ ও গোধূলি সন্ধ্যার নৃত্য
প্রকাশনায় সাব্বির

বিষয় অনুযায়ী লেখা

একটি বিভাগ পছন্দ

শহরের নাম। এবং আউটপুট হচ্ছে শহরটি যে দেশে সেটা। ফাংশনটার নাম যদি দেই $latex g$ তাহলে, $latex g$(ঢাকা) = বাংলাদেশ, g(প্যারিস) = ফ্রান্স, ইত্যাদি।

তার মানে, যে কোনো ফাংশনের জন্যই আমাদের প্রথমে বলে দিতে হবে, যে এর ইনপুট কী। যেমন f এর ইনপুট যে কোনো বাস্তব সংখ্যা, g এর ইনপুট যে কোনো শহরের নাম, ইত্যাদি। শুধু ইনপুট বললেই হবে না, ইনপুট দেওয়া থাকলে সেই অনুযায়ী ফাংশনটা কী আউটপুট দেবে তার একটা নিয়মও বলে দিতে হবে। এই নিয়মটা হতে পারে (1) নং সমীকরণের মত কোনো গাণিতিক সূত্র, আবার g এর মত কোনো বর্ণনা। এমনকি নিয়মটা এটা কোনো কম্পিউটার প্রোগ্রাম বা অ্যালগরিদম আকারেও লেখা হতে পারে।

ফাংশনের ইনপুট একাধিক হওয়া সম্ভব। যেমন, বাস্তব সংখ্যা x এবং y কে ইনপুট হিসাবে নিয়ে একটা ফাংশন h বানানো যায়,

$latex h(x,y) = \sqrt{x^2 + y^2} $ .....(2)

আমরা যারা পিথাগোরাসের উপপাদ্য জানি, তারা নিশ্চই চিনতে পেরেছি, $latex h $ ফাংশনটা স্রেফ $latex x$ দৈর্ঘ্যের ভূমি ও $latex y$ দৈর্ঘ্যের লম্ব বিশিষ্ট একটা সমকোনী ত্রিভূজের অতিভূজের দৈর্ঘ্য।

তার মানে, $latex h $ সমীকরণ (2) এর মত গাণিতিক ভাবে না লিখে। বর্ণনা আকারেও লেখা যায়, $latex h$ হচ্ছে এমন একটা ফাংশন যেটা সমকোনী ত্রিভূজের সমকোণ সংশ্লিষ্ট বাহু দুটির দৈর্ঘ্য ইনপুট নিয়ে অতিভুজের দৈর্ঘ্য আউটপুট দেয়। যে কোনো ফাংশন নিয়ে আলোচনার সময়, সেই ফাংশনটা 'আসলে কি করছে' তার নিজের ভাষায় বর্ণনা করতে পারা খুবই গুরুত্বপুর্ণ।

আমরা কিন্তু চাইলে, ফাংশন h কে f ব্যবহার করেও প্রকাশ করতে পারতাম। যেমন,

$latex h(x,y) = \sqrt{ f(x) + f(y)} $ .....(3)

খেয়াল করো (3) ও (2) নং সমীকরণ কিন্তু আসলে একই কাজ করছে। কারণ, (1) ব্যবহার করে আমরা পাই $latex f(x) = x\times x$ এবং, $latex f(y) = y\times y$, ফলে,

$latex h(x,y) = \sqrt{f(x) + f(y)} = \sqrt{x \times x + y\times y} = \sqrt{x^2 + y^2}$..... (4)

যা কিনা, (2) এর সমার্থক। সমীকরণ (3) এ h ফাংশনটি, f ফাংশন ব্যবহার করে গাণিতিক ভাবে বর্ণিত হয়েছে।

তার মানে, একটা ফাংশনের মান গণনায় চাইলে অন্য ফাংশন ব্যবহার করা যায়। মজার ব্যাপার হচ্ছে, শুধু অন্য ফাংশনই না। কিছু কিছু ফাংশন বর্ণনাতে এমনকি নিজেই ব্যবহৃত হতে পারে। আর এ ধরণের ফাংশনকেই বলে রিকার্সিভ ফাংশন।

রিকার্সিভ ফাংশন

$latex 0, 1, 1, 2, 3, 5, 8, 13, 21, \ldots$

এই ধারাটাকে কি চেনা চেনা লাগে? হ্যাঁ এটা আমাদের অতি প্রিয় ফিবোনাচ্চি ধারা। প্রথম দুটো বাদে এই ধারার প্রতিটি সংখ্যা তার ঠিক আগের দুটি সংখ্যার যোগফল। কিন্তু তাহলে প্রথম দুটো সংখ্যা এলো কোথা থেকে? আসলে শুরুর দুটো সংখ্যা, এবং পরের দিকের সংখ্যাগুলোকে তার ঠিক আগের দুটি সংখ্যার থেকে বের করার উপায়টা গণিতবিদ ফিবোনাচ্চি ঠিক করে দিয়েছেন। আমরা যদি, 0, 1 এর বদলে 3, 4 দিয়ে সিরিজটা শুরু করতাম। তাহলে পেতাম,

$latex 3, 4, 7, 11, 18, 29, \ldots$

এই ধারাতেও, প্রথমদুটো বাদে বাকি সব সব সংখ্যা তার ঠিক আগের দুটি সংখ্যার যোগফল। কিন্তু দেখাই যাচ্ছে, এটা ফিবোনাচ্চি ধারা নয়।

ফিবোনাচ্চি ধারা ব্যবহার করে আমরা চাইলে একটা ফাংশন F কে সংজ্ঞায়িত করতে পারি যার ইনপুট হচ্ছে স্বাভাবিক সংখ্যা (অঋণাত্বক পূর্ণ সংখ্যা) n এবং আউটপুট হচ্ছে n তম ফিবোনাচ্চি সংখ্যা।

অর্থাৎ $latex F(0) = 0, F(4) = 3, F(7) = 13$, ইত্যাদি।

শুরুর দুটো সংখ্যা 0 ও 1 নেবার পর বাকিদের জন্য আমরা বলতে পারি,

$latex n$ তম ফিবোনাচ্চি সংখ্যা হচ্ছে, $latex (n-1)$ ও $latex (n-2)$ তম ফিবোনাচ্চি সংখ্যার যোগফল।

গাণিতিক ভাবে,

$latex F(0) = 0,$……(5)

$latex F(1) = 1,$……(6)

এবং

$latex F(n) = F(n-1) + F(n-2)$……(7)

(5), (6) এবং (7) এই তিনটা সমীকরণ মিলেই n তম ফিবোনাচ্চি সংখ্যা বের করার ফাংশনকে সংজ্ঞায়িত করে, যেখানে F(0) এবং F(1) হচ্ছে বেস ধাপ। এবং (7) হচ্ছে রিকার্সিভ ধাপ।

আমরা এদেরকে ব্যবহার করে কিভাবে 4 তম ফিবোনাচ্চি সংখ্যা বের করা যায় সেটা দেখি।

$latex F(4) = F(3) + F(2)$ ... .(8) [(7) ব্যবহার করে]

$latex =\{F(2)+F(1)\} + \{F(1) + F (0)\}$ ... ... (9) [(8) এর পদ দুটির উপর আবার (7) ব্যবহার করে]

$latex = \{F(1) + F(0)\} + F(1) + F(1) + F(0)$ ... ... (10)
[(9) এর প্রথম পদের উপর আবার (7) ব্যবহার করে]

$latex =1 + 0 + 1 + 1 + 0 $ [(5) ও (6) থেকে মান বসিয়ে]

$latex =3$

অর্থাৎ রিকার্সিভ ধাপে, কোনো ইনপুটের জন্য আমরা ফাংশনটির নিজেকে নিজের ভিন্ন ইনপুট দিয়ে সংজ্ঞায়িত করি। কিন্তু এভাবে যেন চিরকাল চলতেই না থাকে, তা নিশ্চিত করতে আমাদের এক বা একাধিক বেস ধাপ ঠিক করে দিতে হয়। যাদের মান, সরাসরি দেওয়া থাকে।

নিজেকে যাচাই-

1. কোনো ধনাত্বক পূর্ণ সংখ্যার ফ্যাক্টরিয়াল বলতে আমরা বুঝি 1 থেকে সেই সংখ্যা পর্যন্ত সবগুলো সংখ্যার গুণফল। অর্থাত 5 এর ফ্যাক্টরিয়াল হচ্ছে, $latex 1\times 2\times 3\times 4 \times 5$। আমরা এই ফ্যাক্টরিয়াল ফাংশনকে যদি নাম দেই $latex fact$ তাহলে এভাবে লিখতে পারি।
   $latex fact(n) = 1\times 2\times 3\times 4 \times \ldots\times n$ ... ... (11)

   এখন দেখাও যে রিকার্সিভ সমীকরণ

   $latex fact(1) = 1$ ... ... (12)

   $latex fact(n) = n\times fact(n-1)$ ... ... (13)

   মিলে আসলে (11) নং ফাংশনকেই প্রকাশ করছে। $latex fact(4),$ ও $latex fact(5)$ এর মান, প্রথমে সমীকরণ (11) এর

সাহায্যে এবং এরপর সমীকরণ (12) ও (13) এর সাহায্যের নির্ণয় করো।

2. দুইটি ইনপুট বিশিষ্ট একটা ফাংশন C কে রিকার্সিভ উপায়ে সংজ্ঞায়িত করা হয় নিম্নোক্ত উপায়ে।
   $latex C(n , 0)=1 $ ... ... (14)

   $latex C(n,n)=1 $ ... ... (15)

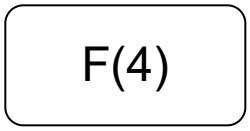
   $latex C(n,k) = C(n-1,k) + C (n-1,k-1) $ ... ... (16)

   এখানে বেস কেস (14), (15) বলছে যে, যদি C এর দ্বিতীয় ইনপুট ০ হয় অথবা C এর ইনপুট দুটো পরস্পরের সমান হয় তাহলে, ওইসব ইনপুটের জন্য C এর মান 1। বাকি অন্য সব ধরনের ইনপুট এর জন্য সমীকরন (16) ব্যবহার হবে। (14), (15) ও (16) ব্যবহার করে C(4, 2) এর মান নির্ণয় করো।

   [hint: $latex C(4,2) = C(3, 2) + C(3,1)= $ ...]

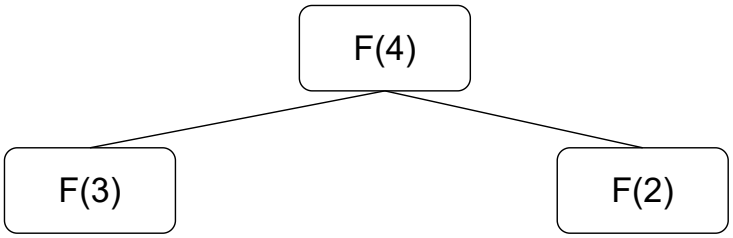
রিকার্সন ট্রি

রিকার্সন কিভাবে কাজ করে তা ছবি এঁকেও দেখানো যায়। সে জন্য সমীকরণ (5), (6) ও (7) এর মাধ্যমে F(4) এর মান বের করতে প্রথমে চিত্র – ১(ক) এর মত একটা আয়তক্ষেত্রের মধ্যে F(4) লিখি। এই আয়তক্ষেত্রকে আমরা বলবো নোড।



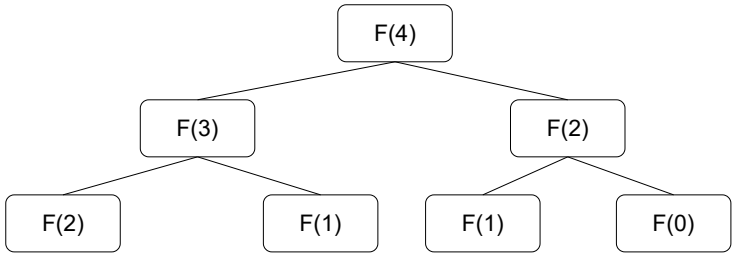
১(ক)

সমীকরণ (7) থেকে আমরা জানি F(4) হচ্ছে F(3) এবং F(2) এর যোগফল। সেটা বোঝাতে চিত্র-১(খ) তে আমরা F(4) এর নিচে দাগ দিয়ে আমরা আরো দুইটি নোড আঁকি।



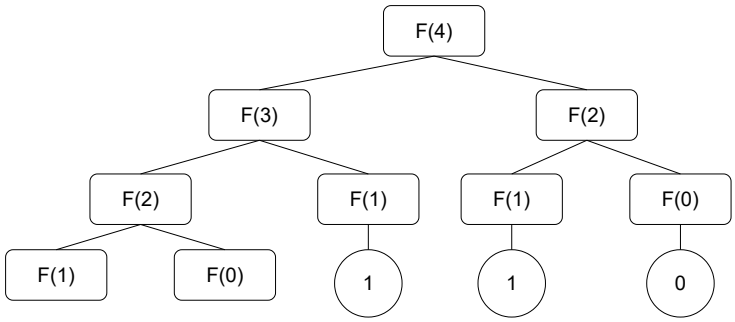
১(খ)

যাদের মধ্যে যথাক্রমে F(3) এবং F(2) লিখি। এদেরকে বলবো F(4) এর চাইল্ড নোড। কিন্তু এদের মানও আমরা সরাসরি জানি না।



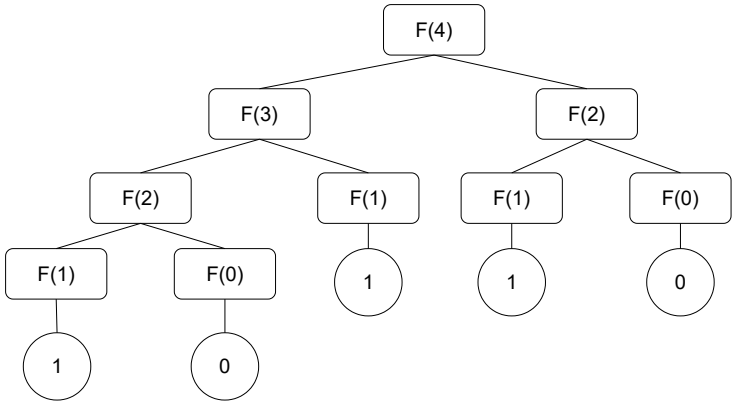
১(গ)

তাই চিত্র-১(গ) আবার সমীকরণ (7) প্রয়োগ করে F(3) ও F(2) এর নোডের নিচেও আরো দুটো করে চাইল্ড নোড আঁকি। দেখা যাচ্ছে এই চিত্রে এখন F(1), F(0) ওয়ালা নোড এসে গেছে। এদের মান আমরা সমীকরণ (5) ও (6) থেকে জানি।



১(ঘ)

যেহেতু এদের মান জানা সেহেতু চিত্র – ১(ঘ) তে F(1), F(0) লেখা নোডের নিচে একটা বৃত্যাকার চাইল্ড নোড এঁকে সেই মানগুলো বসিয়ে দেই। কিন্তু চিত্র -১(গ) তে একটা F(2) নোড রয়ে গেছে। তাই সেটার নিচে আগের নিয়মে F(1) এবং F(0) দুইটি আয়গক্ষেত্রাকার চাইল্ড নোড আঁকা হয়েছে। যাদের মান বসানো হয়েছে চিত্র ১(ঙ) তে।



১(ঙ)

এ ধরনের চিত্রকে বলে রুটেড ট্রি। এই ট্রির রুট নোড হচ্ছে F(4) যেখান থেকে ট্রিটা বড় হতে শুরু করে। ট্রির কোনো নোডের সঙ্গে

নিচের দিকে দাগ দিয়ে যুক্ত নোডকে বলে উপরের নোডটির চাইল্ড। আর চাইল্ডের প্যারেন্ট হচ্ছে উপরের নোডটা। কোনো প্যারেন্টের এক বা একাধিক চাইল্ড থাকতে পারে। ট্রির একদম নিচে যেখানে গিয়ে থামে ঐ নোডোগুলোর কোনো চাইল্ড নেই। এ ধরনের নোডকে বলে লিফ নোড। আমাদের চিত্র ১(ঙ) তে বৃত্তাকার নোডগুলো হচ্ছে লিফ নোড। বোঝাই যাচ্ছে, রুট নোডের কোনো প্যারেন্ট থাকে না।

চিত্র ১(ঙ) তে প্রতিটি প্যারেন্ট নোডের মান, তার চাইল্ড নোডের মানগুলোর যোগফল। চাইল্ড যদি লিফ হয়, তাহলে ঐ লিফের মান সরাসরি পাওয়া যায়। এভাবে ট্রির সবচেয়ে নিচের লেভেল থেকে ঠিক তার উপরের লেভেলের নোডগুলোর মান বের করতে হয় এবং এভাবে ধাপে ধাপে চলতে থাকলে এক সময় রুট নোডের মান বেরিয়ে পড়ে।

এভাবে যখন কোনো রিকার্সিভ ফাংশনের মান নির্ণয় করার প্রক্রিয়াকে ট্রি আকারে প্রকাশ করা হয়, সেই ট্রি কে বলে রিকার্সন ট্রি। চিত্র ১(ঙ) F(4) এর রিকার্সন ট্রি। এই ট্রির নানা অংশের নাম ধাম, আর সেটা বানানোর নিয়ম নিয়ে আমরা এত মাথা ঘামাচ্ছি কারণ পরবর্তীতে অনেক রিকার্সিভ অ্যালগরিদম বুঝতে আমরা এই ধরনের ট্রি ব্যবহার করব।

নিজেকে যাচাই-

3. সমীকরণ (14), (15) ও (16) তে বর্ণিত ফাংশন C(n,k) এর মান n=4 এবং k = 2 এর জন্য (অর্থাৎ C(4,2) এর মান) রিকার্সন ট্রি এঁকে নির্ণয় করো।
4. সমীকরণ (12), (13) এ বর্ণিত ফাংশন fact(n) এর মান n=5 এর জন্য রিকার্সন ট্রি এঁকে বের করো।

রিকার্সিভ প্রোগ্রাম

এতক্ষণে আমরা খাতায় সমীকরণ লিখে এবং রিকার্সন ট্রি এঁকে এই দুভাবে রিকার্সিভ ফাংশনের মান বের করতে শিখেছি। এখন দেখবো একটা কম্পিউটার প্রোগ্রাম লিখে কিভাবে একই কাজ করা যায়। নিচের C কোডটি লক্ষ্য করো।

[code]
```
#include <stdio.h>

int F(int n) {
if (n == 0 ) return 0; /* this is equation (5) */
else if (n ==1 ) return 1; /* this is equation (6) */
else return F(n-1) + F(n-2); /* this is equation (7) */
}
```

```
int main(){
int input = 4; /* we want to compute 4th fibonacci number */
int output = F(input); /* for input n, function F computes n'th fibonacci number */
printf ("the %dth Fibonacci number is %d\n",input, output); /*prints the output*/
return 0; /*terminate the program successfully*/
}
```

[/code]

ফাংশন F এর ইনপুট n যদি ০ বা 1 না হয় তাহলে (n-1) ও (n-2) এর জন্য ফাংশনটিকে কল করা হয়। এদের মান যখন ফিরে আসে তখন তাদের যোগফলকেই আউটপুট হিসাবের রিটার্ণ করা হয়। ফাংশনটি নিজেকেই পুনরায় কল করার সময় তার ইনপুটের মান কমিয়ে দেয়, ফলে এক সময় না এক সময় এই ইনপুটের মান 0 অথবা 1 হয়ে যায় এবং তখন নতুন করে নিজেকে কল না দিয়ে 5 বা 6 নাম্বার লাইন থেকে প্রোগ্রামটি ফিরে আসে।

তুমি চাইলে এই প্রোগ্রামটি main ফাংশনে ইনপুটের বিভিন্ন মান দিয়ে পরীক্ষা নিরীক্ষা করে দেখতে পারো।

তবে এ ধরনের রিকার্সিভ প্রোগ্রাম যদি এই প্রথম দেখে থাকো তাহলে হয়তো নিজেকে নিজে কল করার ব্যাপার টা এখনো পুরোপুরি পরিস্কার হয়নি। ভয় নেই, আমরা এখন খুব ধীরে ধীরে এই ধরনের প্রোগ্রাম কীভাবে কাজ করে তা বুঝবো। আর সেটা করবো মজার কিছু উদাহরণের সাহায্যে। প্রথম উদাহরণটি প্যালিনড্রম নিয়ে।

প্যালিন্ড্রম

বাংলায় মজার কিছু শব্দ বা বাক্যাংশ আছে যেমন 'রমাকান্তকামার', 'নয়ন', 'কীর্তন মঞ্চ পরে পঞ্চম নর্তকী' এদেরকে উল্টো দিক থেকে পড়লেও একই থাকে। ইংরেজীতেও এমন হয়, যেমন "Madam, I'm Adam"। এদেরকে বলে প্যালিন্ড্রম। তবে প্রোগ্রামিং এর সময় আমরা প্যালিণ্ড্রম সংজ্ঞায়িত করি এভাবে, যদি কোনো স্ট্রিং কে রিভার্স করলেও অপরিবর্তিত থাকে তাহলে সেটাই প্যালিন্ড্রম। যেমন, "dcabacd", "nayan", "dad" একটা প্যালিন্ড্রম।

তাহলে কোনো স্ট্রিং প্যালিন্ড্রম কি না, তা চেক করার কয়েক রকম অ্যালগরিদম আমরা ভাবতে পারি। একটা হচ্ছে, স্ট্রিং টার একটা কপি তৈরি করে, সেই কপিকে রিভার্স করে ফেলা। এরপর আদি স্ট্রিং এর সাথে তুলনা করে দেখা তারা একই কি না। অথবা কপি করতে না

চাইলে, একটা লুপের সাহায্যেও চেক করাযায়। স্ট্রিংটা s এবং স্ট্রিং এর দৈর্ঘ্য n হলে।

```
[code language="cpp" ]
int pal(char* s){
int b, e, n;
n = strlen(s);
b = 0;
e = n-1;
while(e>b){
if (s[b] != s[e]) return 0;
b++;
e−;
}
return 1;
}
[/code]
```

ফাংশনটার b এবং e যথাক্রমে শুরু ও শেষ থেকে মাঝের দিকে এগোতে থাকে, এবং মাঝের কোনো ধাপে অমিল পেলেই স্ট্রিংটাকে প্যালিন্ড্রম নয় বলে ঘোষণা করে (০ রিটার্ন করে)। এই লুপ থেকে যদি কিছু রিটার্ণ না হয়, তার মানে স্ট্রিংটি প্যালিন্ড্রম। তখন 1 রিটার্ন হয়।

বোঝাই যাচ্ছে সমস্যাটা কঠিন কিছু নয়। এ কারণেই রিকার্সিভ প্রোগ্রাম কিভাবে কাজ করে তার খুটিনাটি দেখতে প্রথমে আমরা এই pal এর অ্যালগরিদমটাকে রিকার্সিভলি লিখবো। তার মানে, আরো মজার মজার সমস্যাগুলোকে একটু অপেক্ষা করতেই হচ্ছে।

তার জন্য আমরা একটা ফাংশন ভাবতে পারি যাকে কোনো স্ট্রিং আর সেই স্ট্রিং এর মাঝের দুইটি ইন্ডেক্স দিয়ে দিলে, সেই ইন্ডেক্সের মধ্যবর্তী অংশটা প্যালিন্ড্রম কি না সেটা রিটার্ণ করে। ফাংশন টিকে নাম দেই rec_pal

C তে লিখলে তার সিগনেচারটা দেখতে হবে।

```
[code language="cpp"]
int rec_pal( char* s, int b, int e);
[/code]
```

যেখানে, s হচ্ছে স্ট্রিং এর পয়েন্টার। b হচ্ছে, যেখান থেকে চেক করতে চাই, তার শুরুর ইন্ডেক্স। আর e হচ্ছে, সেই শেষের ইন্ডেক্স। ফাংশনটাকে যদি, এভাবে কল করি rec_pal(s,0, n-1) যেখানে n হচ্ছে স্ট্রিং s এর দৈর্ঘ্য তাহলে পুরো স্ট্রিংটাই প্যালিন্ড্রম কি না তা জানা যাবে।

এখন মনে করি আমরা স্ট্রিং s এর মাঝে b থেকে e ইনডেক্স পর্যন্ত প্যালিন্ড্রম কি না তা চেক করতে চাই। তার জন্য প্রথমে দেখতে হবে s[b] আর s[e] সমান কি না। যদি সমান না হয়, তাহলে সাথে সাথে আমরা বলতে পারি যে স্ট্রিংটা প্যালিন্ড্রম নয়। আর যদি সমান হয়, তাহলে মাঝের অংশটুকু চেক করতে হবে যে তারা প্যালিন্ড্রম কি না। এ কাজে আমরা rec_pal ফাংশনটি ব্যবহার করতে পারি। তার মানে,

[code language="cpp"]
if (s[b] != s[e]) return 0;
return rec_pal(s, b+1, e-1);
[/code]

এই দুই লাইনে সেটাই করা হচ্ছে। প্রথমে দেখছি শুরুর আর শেষের অর্থাৎ, b আর e অবস্থানের ক্যারেকটার দুটো সমান কি না। অসমান হলে 0 রিটার্ন করবো সরাসরি। যদি তারা সমান হয় তাহলে, b এর একঘর ডানে থেকে শুরু করে e এর একঘর বামের অংশ পর্যন্ত স্ট্রিং টাকে চেক করতে হবে। তার জন্য rec_pal কল করলেই হবে। খেয়াল করো, rec_pal নিজে কিভাবে কাজ করে তা আমরা এখনো জানি না। কিন্তু সেটাও চাইলে এই আগের দুই স্টেপের মতই কাজ করতে পারে। মানে, মাঝের অংশটারও শুরুর আর শেষের ক্যারেকটার মিলিয়ে দেখবে, এবং তারপর, আবার rec_pal ব্যবহার করবে মাঝের অংশটার জন্য। এভাবে b বাড়তে বাড়তে আর e কমতে কমতে একটা আরেকটাকে পার করে যায়, তখন বুঝতে হবে স্ট্রিং টি প্যালিন্ড্রম। তার মানে ফাংশনটাকে পূর্ণাঙ্গ ফাংশনটাকে আমরা নিচের মত করে লিখতে পারি।

[code language="cpp"]
int rec_pal(char* s, int b, int e){
if (b>e) return 1;
if (s[b] != s[e]) return 0;
return rec_pal(s, b+1, e-1);
}
[/code]

কোনো ইনপুট স্ট্রিং s এর জন্য মেইন প্রোগ্রাম থেকে ফাংশনটিকে কল করা হবে এভাবে।

[code language="cpp"]
int main(){
int n, result;
char s[] = "nayan";

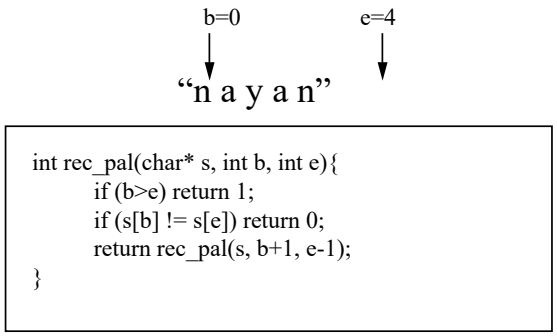
n = strlen(s);

```
result = rec_pal(s,0,n-1);
printf("%d\n",result);
}
```

এখন আমরা s = "nayan" এই ইনপুটের জন্য rec_pal ফাংশনটা কিভাবে কাজ করছে তা ধাপে ধাপে বুঝতে চেষ্টা করবো।

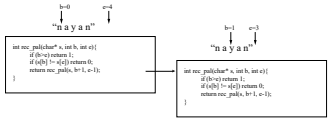
এখানে স্ট্রিং এর দৈর্ঘ্য n = 4 তাই মেইন ফাংশন থেকে কল করা হবে rec_pal(s, 0, 3);

এই ইনপুট নিয়ে প্রোগ্রামটি তখন rec_pal ফাংশনের মধ্যে প্রবেশ করবে।



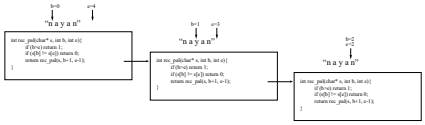
২(ক)

চিত্র ২(ক) তে একটা বক্সের মধ্যে rec_pal এবং উপরে তার ইনপুটগুলো দেখানো হলো। এক্ষেত্রে, ইনপুট স্ট্রিং টা হচ্ছে, "nayan" এবং b ও e যথাক্রমে শুরুর আর শেষের ক্যারেকটার দুটোকে ইন্ডেক্স করছে। বক্সের কোডের দ্বিতীয় লাইনের if কন্ডিশন মিথ্যা তাই সেখানে কিছু হবে না। এখন দেখতে হবে b থেকে e তম ইন্ডেক্স পর্যন্ত স্ট্রিং এর প্রথম আর শেষ ক্যারেকটার একই কি না। সেটা দেখছি তৃতীয় লাইনে। তৃতীয় লাইনের if কন্ডিশনও মিথ্যা কারণ এখানে s[b] আর s[e] সমান। এখন আমাদের দেখতে হবে, b+1 থেকে e-1 অংশটা প্যালিন্ড্রম কি না। সে জন্য, চতুর্থ লাইনে এসে, s, b+1 আর e-1 ইনপুট নিয়ে rec_pal ফাংশনটি আবার কল হচ্ছে। এখানে ফাংশনটা নিজেকেই নিজে কল করছে তাই একে বলে রিকার্সিভ কল।



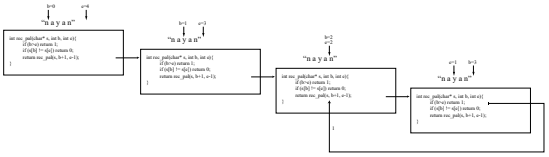
২(খ)

কম্পিউটার তার কল স্ট্যাকের সাহায্যে এই ধরনের কল ইমপ্লিমেন্ট করে। কিন্তু বোঝার সুবিধার্থে আমরা ভেবে নেব, রিকার্সিভ কল হলে প্রথমে ওই ফাংশনটার একটা কপি তৈরি হয়। এবং পরিবর্তিত ইনপুটগুলো নিয়ে, নতুন সেই কপিটা চলতে শুরু করে। এই পুরো ব্যাপারটা বোঝানো হয়েছে চিত্র- ২(খ) তে। যেখানে ডানদিকের বক্স টা হচ্ছে ফাংশনের নতুন কপি। বাম বাক্সের লাইন থেকে নতুন বাক্সটা কল হয়েছে সেখান থেকে একটা তীর এসেছে নতুন বক্সে। এবং নতুন বক্সের ইনপুটগুলো বাক্সর উপরে দেখানো হয়েছে। নতুন বক্সটার ভেরিয়েবল গুলোর নাম আগের মত হলেও, এরা যেহেতু নতুন মেমরি স্পেসে আছে, এদের মান তাই নতুন ইনপুটগুলোর সমান হবে। একারণেই বামের বাক্সে b এর মান ০ আর ডানের বাক্সে b এর মান 1 উভয়ের নাম b হলেও, এরা যেহেতু ভিন্ন বাক্সের ভেরিয়েবল সেহেতু নিজ বাক্সের মধ্যে এদের আলাদা নিজস্ব মান থাকতে পারে। এখন ডানের বাক্সের তৃতীয় লাইনে আবার চেক করা হবে, নতুন s[b] আর s[e] সমান কি না। ডানের বাক্সে এরা সমান। তাই আবার মাঝের অংশটা নিয়ে একই ভাবে আরেকবার রিকার্সিভ কল হবে। চিত্র – ২(গ) তে সেটাই দেখানো হয়েছে।



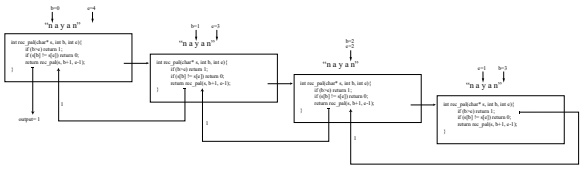
২(গ) **[বড় করে দেখতে ছবিতে ক্লিক করুন]**

এভাবে এই তৃতীয় বাক্স থেকে চতুর্থ বারের মত আবারো ফাংশনটা কল হবে (চিত্র- ২(ঘ)) ।



২(ঘ)**[বড় করে দেখতে ছবিতে ক্লিক করুন]**

কিন্তু এতক্ষণে পুরো স্ট্রিংটাই প্যালিন্ড্রম কি না তা চেক হয়ে গেছে। ফলে b, e কে অতিক্রম করে যাবে এবং এই বাক্সের প্রথম if কন্ডিশন সত্যি হবে এবং 1 রিটার্ন করবে। একটা তীরের মাধ্যমে আমরা সেটা বুঝিয়েছি। খেয়াল করো, স্ট্রিং টা প্যালিন্ড্রম না হলে আগের কোনো ধাপে দ্বিতীয় if কন্ডিশন টা সত্যি হতো ফলে 0 রিটার্ন করতো।



২(ঙ)**[বড় করে দেখতে ছবিতে ক্লিক করুন]**

চিত্র- ২(ঙ) তে এই রিটার্ন ভ্যালু একে একে আগের বাক্সগুলোতে পৌছাচ্ছে, এবং সব শেষে আউটপুট ১ দিয়ে নির্দেশ করছে যে স্ট্রিং s একটি প্যালিন্ড্রম।

নিজেকে যাচাই-

5. “nabin” স্ট্রিং টার জন্য rec_pal ফাংশনটা কিভাবে কাজ রে তা, চিত্র ২(ক) থেকে ২(ঙ) এর মত করে দেখাও।

রিকার্সন নিয়ে খেলাধুলা

স্ট্রিং প্রিন্ট

মনে করো আমরা একটা স্ট্রিংএর প্রতিটা ক্যারেকটার প্রিন্ট করতে চাই। তা করার হাজারটা উপায় আছে। যেমন যদি স্ট্রিংটা হয় s তাহএল স্রেফ

[code language=”cpp”]
printf(“%s”, s);
[/code]

লিখলেই পুর স্ট্রিংটা প্রিন্ট হয়ে যাবে।

চাইলে একটা লুপের মধ্যে স্ট্রিং এর প্রতিটা ক্যারেকটার একটা একটা করে প্রিন্ট করা যায়।

[code language=”cpp”]
int i;
char s[] = “abcde”
for (i= 0; s[i]!= ‘\0’; i++){

```
printf("%c",s[i]);
}
```

কিন্তু লুপ ঘোরাতে বোরিং লাগলে আমরা আমাদের নতুন শেখা রিকার্সনের সাহায্যেও পুরো স্ট্রিংটা প্রিন্ট করতে পারি। নিচের কোডের সাহায্যে,

```
#include <stdio.h>
void rec_print(char* s){
if (s[0] == '\0') return; /*স্ট্রিং শেষ হয়ে গেলে আর কিছু না করে ফিরে যাও */
printf("%c", s[0]); /*স্ট্রিং এর প্রথম ক্যারেক্টার প্রিন্ট করো */
rec_print(s+1); /*এর পর বাকিটা প্রিণ্ট করতে বাকিটুকু ইনিপুট হিসাবে দিয়ে নিজেকে রিকার্সিভ কল করো */
}

int main(){
int i;
char s[] = "abcde";
rec_print(s);
printf("\n");
return 0;
}
```

এখানে ফাংশন rec_print(char *s) কোনো স্ট্রিংকে রিকার্সিভলি প্রিন্ট করে। ফাংশনটির শেষ লাইনে এটা নিজের অন্য এক কপিকে কল করছে s+1 ইনপুট দিয়ে। আমরা জানি, s যদি কোনো স্ট্রিং এর প্রথম ক্যারেক্টারের পয়েন্টার হয়। তাহলে s+1 হচ্ছে ঐ স্ট্রিংএর দ্বিতীয় ক্যারেক্টারের পয়েন্টার। স্ট্রিং এর শেষে সব সময় শেষ নির্দেশী NULL ক্যারেক্টার তথা '\০' থাকে। তাই ফাংশনের প্রথম লাইনেই আমরা দেখে নিচ্ছি স্ট্রিং এর শেষে পৌছে গেলাম কি না।

নিজেকে যাচাই-

6. rec_print ফাংশনটাকে একটু বদলে নিয়ে rec_print2 নামের একটা ফাংশন লেখা যায় যেখানে শেষ দু লাইন আগুপিছু করা হয়েছে। অর্থাৎ প্রথম ক্যারেক্টারটা প্রিণ্ট করা হচ্ছে রিকার্সিভ কল হবার পরে। ঠিক এমন,

   ```
   [code language="cpp"]
   ```

```
void rec_print2(char* s){
if (s[0] == '\0') return;
rec_print2(s+1);
printf("%c", s[0]);
}
```

[/code]
এই আগের প্রোগ্রামে rec_print2 ব্যবহার করে দেখ কী প্রিন্ট হয়। আউটপুটে কি কি পরিবর্তন হচ্ছে? কেন?

7. n উপাদান বিশিষ্ট একটা integer অ্যারের উপাদানগুলোকে রিকার্সিভ ফাংশনের সাহায্যে প্রিন্ট করো।

গসাগু

দুইটি ধনাত্বক পূর্ণ সংখ্যার গসাগু বের করার উপায় আমরা সবাই জানি। একটাকে ভাজক আর আরেকটাকে ভাজ্য ধরে ভাগ দিতে হয়। যদি কোনো অবশিষ্ট না থাকে তাহলে ভাজকই গসাগু। আর যদি অবশিষ্ট থাকে তাহলে, আগের ভাজককে নতুন ভাজ্য ধরে এবং আগের ভাগশেষ কে নতুন ভাজক ধরে ভাগ করেত হয়। যতক্ষণ ভাগশেষ 0 না হয় ততক্ষণ এই প্রক্রিয়া চলতেই থাকে। এভাবে এক সময় অবশ্যই ভাগশেষ 0 হয়। আর তখনই সেই ভাজককে বলে একেবারে শুরুর সংখ্যাদুটোর গসাগু। এটা হলো গসাগু বের করার ইউক্লিডিয়ান অ্যালগরিদম।

অ্যালগরিদমটাকে প্রোগ্রাম আকারে এভাবে লেখা যায়,

```
[code language="cpp"]
#include <stdio.h>
void gcd(int a, int b){
int r;
while(1){
r = b%a;
if (r == 0) return a;
else{
b = a;
a = r;
}
}
}

int main(){
int a = 12;
int b = 18;
```

```
int result;
result = gcd(a,b);
printf (“the gcd of %d and %d is: %d”, result);
}
```

কিন্তু gcd ফাংশনটির লুপের মধ্যে আমরা প্রতিবার ভাগশেষ অনুযায়ী, ভাজক ভাজ্যকে বদলে নিয়ে একই কাজ করছি। তাই চাইলে গসাগু রিকার্সনের যাহায্যেও বের করা যায় যায় এভাবে,

```
#include <stdio.h>
int rec_gcd(int a, int b){
int r;
r = b%a;
if (r == 0) return a;
else return rec_gcd(r,a);
}

int main(){
int a = 12;
int b = 18;
int result;
result = rec_gcd(a,b);
printf (“the gcd of %d and %d is: %d\n”, a,b,result);
}
}
```

C এর টারনারি অপারেটর ব্যবহার করে rec_gcd ফাংশনটিকে এক লাইনে লেখা যায় এভাবে।

```

int rec_gcd2(int a, int b){
return (b%a==0?a:rec_gcd2(b%a,a);
}

```

অর্থাৎ, b%a == 0 হলে গসাগু a নাইলে, রিকার্সিভলি আবার rec_gcd2 কল হবে ভাজক ভাজ্য বদলে নিয়ে।

পারমুটেশন বা বিন্যাস

$latex n$ টি ভিন্ন ভিন্ন জিনিসকে কত উপায়ে একটা সারিতে সাজানো যায়? উত্তরটা আমরা জানি। ফ্যাক্টরিয়াল $latex n$ উপায়ে। ফ্যাক্টরিয়াল n এর মান বের করার ফাংশনটাও আমরা শিখেছি এই লেখার যাচাই প্রশ্ন (১) এ। এখন যদি তোমাকে n=৫ টি ইংরেজী ক্যারেক্টার দিয়ে বলা হয় সম্ভাব্য সবগুলো বিন্যাসের তালিকা করতে তাহলে মোট fact(5) = 120 টা স্ট্রিং লিখতে হবে। এই কাজ হাতে না করে, একটা প্রোগ্রাম লিখে ফেলাই শ্রেয়। প্রোগ্রামটা কেমন হবে তা বুঝতে আমরা ধরি ক্যারেক্টার দেওয়া আছে মাত্র 4 টি a, b, c, d।
এখন প্রথম a লিখি। এর পর b নিয়ে a এর আগে ও পরে বসিয়ে দুই রকম স্ট্রিং বানাতে পারি,
ক) ba
খ) ab

এখন ক) নং উপায়ের জন্য আবার c কে বসানো যায় তিনটি স্থানে (আগে, মাঝে, পরে)। এভাবে পাই ৩ দৈর্ঘ্যের তিনটি স্ট্রিং

১) cba
২) bca
৩) bac

সেই সঙ্গে, খ) নং স্ট্রিং ab এর ও তিনটি স্থানে c বসিয়ে পাই,
৪) cab
৫) acb
৬) abc

এখন যদি কেউ আরেকটা ক্যারেক্টার d দেয়, তাহলে সেই ডি কে এই 6 রকম স্ট্রিং এর প্রতিটির মধ্যে সম্ভাব্য 4 রকম করে বসানো যাবে। এবং এভাবে আমরা $latex 6 \times 4 = 24 $ রকম স্ট্রিং পাবো।
ব্যাপারটাকে নিম্নের চিত্রের মত করে দেখা যেতে পারে,

চিত্র-৩(ক): একটা একটা করে নিয়ে 4 টি ভিন্ন ক্যারেক্টারের সম্ভাব্য সবগুলো পারমুটেশনের তালিকা তৈরি।

খেয়াল করুণ ৩(ক) নং চিত্রে প্রতিটি ধাপে আমরা নতুন একটা করে ক্যারেক্টার নেই, তারপর সেগুলোকে বর্তমান স্ট্রিং এর সবগুলো সম্ভাব্য স্থানে বসিয়ে বসিয়ে নতুন স্ট্রিং তৈরি করি। এই কাজটা প্রতি ধাপেই একই রকম। তাই একটা রিকার্সিভ ফাংশনের সাহায্যে এটাকে লিখে ফেলা যায়। সেজন্য আমরা এবারে ব্যবহার করবো C++ ।

[code language="cpp"]

```
#include <iostream>
#include <string>

using namespace std;

string s = “abcd”;

void perm(int take, string per){
if (take==s.size()) {
cout<<per<<endl;
return;
}

for (int i = 0; i<=per.size(); i++){
string str = per;
str.insert(i,1,s[take]);
perm(take+1,str);
}
}

int main() {
perm(0,"");
return 0;
}
```

[/code]

perm ফাংশনটার হাতে ইনপুট হিসাবে আসে একটা স্ট্রিং per এবং একটা নতুন ক্যারেক্টার। এখানে take ভেরিয়েবল দিয়ে বোঝানো হচ্ছে নতুন ক্যারেকটারটা হচ্ছে s[take]।

এটা হাতে পাবার সাথে সাথে, perm ফাংশনটা, প্রথমে দেখে per স্ট্রিং টাতে ইতিমধ্যেই সবগুলো ক্যারেক্টার নেওয়া হয়ে গেছে কি না, তাহলে স্ট্রিংটা প্রিণ্ট করে return করে। এটা হচ্ছে ফাংশনের বেস কেস।

আর যদি তা না হয়, তাহলে নতুন ক্যারেক্টারটাকে প্রাপ্ত স্ট্রিং সামনে, মাঝের প্রতিটি স্থানে ও শেষে যুক্ত করে নতুন নতুন স্ট্রিং তৈরি করে। এবং সেই সদ্যপ্রাপ্ত স্ট্রিং আবার perm ফাংশনের ইনপুট হিসাবে দেয়, সেই সঙ্গে দেয় নতুন যে ক্যারেক্টারটা নিতে হবে তার ইন্ডেক্স take+1। এভাবে ফাংশনটি তার পরবর্তী লেভেলে প্রবেশ করে।

যেসব ক্যারেক্টারের পারমুটেশন নিতে হবে তারা উপরের গ্লোবাল স্ট্রিং s এ থাকে। এবং শুরুতে main() থেকে perm(0,

“”) কল হয়। কারণ একেবারে শুরুতে আমাদের হাতে আছে একটা ফাকা স্ট্রিং, এবং আমাদেরকে নিতে হবে s এর 0 তম ক্যারেক্টারটা।

পুনরাবৃত্ত বস্তুর বিন্যাস

আগের সেকশনে আমরা দেখেছি, n সংখ্যক ভিন্ন ভিন্ন বস্তুর সবগুলো পারমুটেশন বা বিন্যাস বের করার উপায়। এখন দেখবো যদি সবগুলো বস্তু ভিন্ন না হয়ে কিছু কিছু বস্তু একই থাকে তাহলে কী হয়। যেমন, যদি aab এই তিনোটি ক্যারেক্টার এর সব রকম পারমুটেশন বের করতে বলা হয়। তাহলে হবে
baa
aba
aab

এই তিনটি। যেখানে, abc এর পারমুটেশন সম্ভব 6 টি। তার মানে এ ধরনের ইনপুটের ক্ষেত্রে আমাদের ফাংশনটিকে বদলাতে হবে। সেই পরিবর্তিত ফাংশন rep_perm সহ প্রোগ্রামটি দেখতে হবে এমন।

[code language=“cpp”]

```
#include <iostream>
#include <string>

using namespace std;

string s = “aabbcc”;
int c = 0;

void rep_perm(int take, string per){
if (take==s.size()) {
cout<<per<<endl; c++;
return;
}
for (int i = 0; i<=per.size(); i++){
string str = per;
str.insert(i,1,s[take]);
rep_perm(take+1,str);
if(str[i+1]==s[take]) break;
}
}

int main() {
rep_perm(0,"");
```

```
cout<<"number of permutations: "<<c<<endl;
return 0;
}
```

[/code]

এই rep_perm কাজ করছে কিভাবে?

মনে করো ইনপুট হিসাবে ক্যারেক্টার দেওয়া আছে a, b, b, c।

তাহলে শুরুতে আমি a নিয়ে স্ট্রিং পাবো
a
এর পর b নিয়ে আগে ও পরে বসিয়ে পাবো
ক) ba
খ) ab

এর পর, আবার আরেকটা b নেব, নিয়ে বসাবো ক) স্ট্রিং এর শুরুতে। ফলে পাবো,
১) bba
কিন্তু এর পরই rec_perm এর
if(str[i+1]==s[take]) break;
এই কন্ডিশন সত্যি হয়ে যাবে, অর্থাৎ আমরা দেখবো যে নতুন বসানো ক্যারেক্টারটা ঠিক তার আগের ক্যারেক্টারটার সঙ্গে মিলে গেছে। তাই আবার না বসিয়ে থেমে যাবো। কেননা, আমরা যদি ba এর সামনে b বসাই, অথবা মাঝে b বসাই উভয় ক্ষেত্রেই পাই bba। তাই একবার বসিয়ে থেমে যাওয়াই যথেষ্ট।

এর পর, খ) স্ট্রিং টাকে নিয়ে আবার b বসাতে শুরু করবো, তখন পাবো,
২) bab
৩) abb
এর পরেই, আবারো সেই break; স্টেটমেন্ট টা এক্সিকিউট হবে, ফলে প্রথম তিনটি ক্যারেকটার নিয়ে আমরা পাবো ৩ টি স্ট্রিং। এবার ১ থেকে ৩ নং স্ট্রিং এর প্রত্যেককে নিয়ে তার মধ্যে চিত্র-৩(ক) এর মত পদ্ধতি অবলম্বন করে আমরা পাই
cbba
bcba
bbca
bbac
cbab
bcab
bacb
...
...

এরকম মোট (4!)/(2!) = 12 টি স্ট্রিং।

নিজেকে যাচাই-

1. চিত্র ৩(ক) টি সম্পূর্ণ করো। এবং “abbcc” এই ইনপুটের জন্য rep_perm ফাংশনটি ব্যবহার করে ৩(ক) এর মত একটি চিত্র আঁকো।

About the Author: তানভীর

মুক্তমনা ব্লগ সদস্য।

Related Posts

'অর্থনৈতিক বক্রতা', কতটা দায়ী কৃত্রিম বুদ্ধিমত্তা ?
ডিসেম্বর 22, 2016 | 2 Comments

বিগডেটা -ধর্ম, গণতন্ত্র এবং রাজনীতির ভবিষ্যত
আগস্ট 17, 2014 | 20 Comments

প্রোগ্রামি
ডিসেম্বর
Comme

19 Comments

zitul_mahmud জুন 11, 2016 at 1:28 পূর্বাহ্ন - Reply

খুব ভালো লেগেছে লেখাটা।

তাহমিদ-উল-ইসলাম নভেম্বর 1, 2015 at 5:32 অপরাহ্ন - Reply

আপনার লেখাটি অনেকটাই পড়েছি। সাবলীল হলেও একটু বুঝে বুঝে আগাচ্ছি। একটু বুঝে বুঝে আগাচ্ছি। আচ্ছা ভাইয়া, আমি কলেজে পড়ি। এবার ইনফরমেটিক্স অলিম্পিয়াডে অংশ নিবো। ভালো করতে গেলে, রিকার্শনের পরে ডায়নামিক প্রোগ্রামিংয়ের আর কী কী জিনিস শেখা লাগবে? মানে কতটুকু জানলে ওখানে ভালো একটা ফলাফল করতে পারব? 🙂

জীবন জুলাই 22, 2014 at 1:50 পূর্বাহ্ন - Reply

আমি অভিভুত :rotfl: অনেক কিছু শিখলাম । এত সহজ, সাবলিল উপস্থাপনা আমার জীবনে এই প্রথম দেখলাম । অনেক অনেক ধন্যবাদ (Y)

তানভীরুল ইসলাম জুলাই 22, 2014 at 1:02 অপরাহ্ন - Reply

@জীবন,
লেখাটা আপনার কাজে লেগেছে জেনে ভালো লাগলো। অনেক ধন্যবাদ। 🙂

সৌরভ জুলাই 21, 2014 at 9:56 অপরাহ্ন - Reply

অমানবিক হয়েছে লেখাটা, রিকার্সিভ ফাংশন নিয়ে এমন বিস্তৃত বর্ণনা আগে কোথাও দেখেছি বলে মনে পরছে না। তবে লেখাটা যেহেতু বড় হয়েই গেছে তাই আর একটা ছোট প্যারা যোগ করে দেয়া যেতে পারে! রিকার্সন ব্যবহারের সুবিধা-অসুবিধা। সুবিধার মধ্যে যেমন আছে: সমাধানের সত্যতা প্রমাণ করা সহজ, কোডের আকার ছোট হয় – তাই ভুল করার সম্ভাবনা কমে, এক ধরণের সমস্যা আছে যা রিকার্সন ছাড়া করা বাস্তব সম্মত না যেমন: কিছু জিনিষের সব ধরণের বিন্যাস বা সমাবেশ বের করা যেখানে আলোচ্য বস্তু গুলোর সংখ্যা সমাধাণ লেখার সময় অজানা থাকে, এবং এক ধরণের সমস্যা

রিকার্সন ছাড়া সম্ভব না যেমন: ইনপুট যদি একটি ট্রি হয় তাহলে রিকার্সন ছাড়া ওই ট্রিয়ের ডালে-ডালে পাতায়-পাতায় ঘোরাঘুরি করা সম্ভব না। রিকার্সনের অসুবিধা বা খারাপ দিকের মধ্যে আছে: রিকার্সিভ প্রোগ্রাম কম্পিউটারে বেশি মেমরি নেয়, কিছুটা স্লথ গতীতে চলে... ইত্যাদি।
এরকম একটা প্যারা যোগ করে দিলে আমার মত কেউ যদি থাকে যে কোন কিছু শেখার আগে ওটা আদৌ তার কাজে লাগবে নাকি জানার চেষ্টা করে, তার জন্য সুবিধা হবে!

শুভকামনা

তানভীরুল ইসলাম জুলাই 22, 2014 at 1:01 অপরাহ্ন - Reply

@সৌরভ,

হ্যাঁ, ভালো কথা মনে করেছো! এরকম একটা সেকশন যোগ করতে হবে।
চমৎকার মন্তব্যের জন্য (F)

হাসনাত মিলন জুলাই 16, 2014 at 12:30 পূর্বাহ্ন - Reply

এককথায় অসাধারণ। এমন সাবলীল লেখা, তাও আবার এমন একটা বিষয়ে... অসাধারণ। আপনার আরো লেখা আশা করতেছি।

তানভীরুল ইসলাম জুলাই 22, 2014 at 12:56 অপরাহ্ন - Reply

@হাসনাত মিলন,

ধন্যবাদ আপনাকে। 🙂

মাহমুদুল ফয়সল আল আমীন জুলাই 11, 2014 at 2:04 পূর্বাহ্ন - Reply

দুখিত, ইংরেজিতে লিখছি।

Very nice initiative indeed.

This comment is only for the author and who understand my poor writing skill. And it is just my humble opinion. The main weakness of our skill in recursion is due to the tendency to understand the

recursion tree. We instead should concentrate only on the validity of the definition. When we think of the recursion tree, its concept is similar to using loop… i mean, thinking of all the steps from the beginning to the end of a particular example and then generalizing it. It often leads incorrect definition.

Instead, we may think recursive as an definition by induction and we just need to verify the definition logically straightforward. Therefore, we can verify our definition just by a proof by induction. This is more mathematical and rigorous.

Steps:
1. Think of a parameter of the function that can vary in size (e.g. n).
2. Define the function for the smallest (one or more) instances of that parameter (e.g. f(0)).
3. Now we need to define the function for an arbitrary instance of that parameter (e.g. f(n)). Now we have the induction hypothesis – function for the smaller instance (e.g. f(n-1)). Now we need some computation (e.g. h(f(n-1))) on it so that it become equal to the original function (e.g. f(n)).

Example 1. Palindrome:

step 1. a string or list of characters is the input and it can vary in size.
step 2. P([]) = true
P([a])=true
step 3. Lets think that we need to check the string – a[1]a[2]…a[n-1]a[n] (here each a[i] is a character)
We have the induction hypothesis – a[2]…a[n-1] is already palindrome.
Now, we only need to see whether a[1] is same as a[n].
Therefore, together, our definition can be-
P(a[1]a[2]…a[n-1]a[n]) = (a[1]==a[n]) && P(a[2]…a[n-1]).

Example 2. Factorial:
step 1. A number is the input and it can vary in size.
step 2. F(0) = 1 (by definition)
step 3. Lets think that we need to compute n*(n-1)*…*1 by F(n).

We have the induction hypothesis – F(n-1) computes (n-1)*...*1.
Now, we only need to see what we need more to equalize the function. And it is just a multiplication of n with (n-1)*...*1.
F(n) = n*(n-1)*...*1 = n*F(n-1)

Here we don't need to concentrate on the recursion tree. Just the proof by induction what we already learnt at our high school is sufficient to define a function. The next step is how to convert the function into a program. I always prefer functional programming language (like Haskell, Erlang) to convert the mathematical expression with less effort.
If someone like C, the approach can be kept similar.

Thank you.

মুক্তমনা এডমিন জুলাই 16, 2014 at 10:06 পূর্বাহ্ন - Reply

@মাহমুদুল ফয়সল আল আমীন,
মুক্তমনায় ইংরেজী মন্তব্য গ্রহণ করা হয় না। তবে গুরুত্ব এবং মান বিবেচনায় এই মন্তব্যটা ছাপানো হলো। পরবর্তীতে বাংলায় লিখতে অনুরোধ করা হলো।

তানভীরুল ইসলাম জুলাই 16, 2014 at 12:42 অপরাহ্ন - Reply

@মাহমুদুল ফয়সল আল আমীন,

চমৎকার মন্তব্যের জন্য ধন্যবাদ।

আপনি যেভাবে লিখেছেন সেভবে লিখলে রিকার্সিভ ফাংশনের ব্যাখ্যাটা আরো রিগোরাস হত নিঃসন্দেহে। ইনফ্যাক্ট এই লেখাটা লিখতে যতটা সময় লেগেছে, তার চেয়ে অনেক বেশি সময় নিয়েছি কী লিখবো আর কী লিখবো না সেই সিদ্ধান্ত নিতে। এবং নানান চিন্তাভাবনার পর এই অংশ এভাবে লেখার সিদ্ধান্ত নেই। কেন সেটা ব্যাখ্যা করি,

১) আমি চেয়েছি খুব ইলিমেন্টারি ট্যুলস ব্যবহার করতে। যেন ক্লাস সিক্স সেভেনের ছেলেমেয়েরাও ফলো করতে পারে। তাই আপনি যেভাবে বললেন, তেমন সেকেন্ড অর্ডার লজিক ব্যবহার করার উপায়

ছিলো না। (এমনকি উচ্চতর ক্লাসেও সবাই এসব ধারণায় ফ্লুয়েন্ট নয়)।

২) আর ডিস্ক্রিট ম্যাথের বই এর মত করে রিকার্সিভ ফাংশনের রিগোরাস বর্ণনা দিতে গেলে শুরুতে, প্রপোজিশনাল লজিক থেকে শুরু করে ফার্স্ট অর্ডার সেকেন্ড অর্ডার লজিক শিখিয়ে নিতে হয়। তারপর, ইনডাকশন আর রিকারেন্স রিলেশনের সম্পর্ক তৈরির অবকাশ থাকে। ইন ফ্যাক্ট, এ কারণেই ডিস্ক্রিট ম্যাথের বইগুলো লজিক চ্যাপ্টার দিয়ে শুরু করে।

৩) রিকার্সন ট্রির বর্ণনাটা এই লেখা বোঝার জন্য অপরিহার্য কিছু নয়। আমি রিকার্সনকে কল্পনায় চিত্রিত করার একটা বিকল্প উপায় হিসাবে ট্রিকে উপস্থাপন করেছি। কারণ গণিতের বিষয়গুলো কোনো কোনো শিক্ষার্থী অ্যালজেব্রাইক ফর্ম হিসাবে ভালো বোঝে আবার কোনো কোনো শিক্ষার্থী মনের মধ্যে একটা ছবি আঁকতে না পারলে কোনো কিছু শিখেই সন্তুষ্ট হয় না। আর রিকার্সন ট্রি সঠিক ভাবে ব্যবহার করলে লজিক্যাল ফ্যালাসি হবার কোনো সুযোগ নেই।

৪) প্রোগ্রামের এক্সিকিউশন ট্রি, শুধু প্রোগ্রাম বোঝার জন্যই নয়, তাত্ত্বিক কম্পিউটার বিজ্ঞানেও অনেক কাজে লাগে। ইনডিটার্মিনিস্টিক পলিনমিয়াল টাইম অ্যালগরিদম, QIP, র‍্যান্ডমাইজশ কমপ্লেক্সিটি, সহ নানান কমপ্লেক্সিটি ক্লাসের আলোচনায় এক্সিকিউশন ট্রি একটা অপরিহার্য উপাদান।

৫) পরে আমরা যখন ডাইনামিক প্রোগ্রামিং নিয়ে লিখবো তখন, প্রবলেমের অপটিমাল সাবস্ট্রাকচার, আর ওভারল্যাপিং সাবপ্রবলেম এইসব ধারণা বোঝাতেও ট্রি ব্যবহার করতে হবে। বর্তমান বর্ণনা সেই ভবিষ্যতের লেখারও একটা পূর্বপ্রস্তুতি।

সবশেষে, আবারো অসংখ্য ধন্যবাদ এত যত্ন নিয়ে এতবড় একটা কমেন্ট করার জন্য। আসলে কোনো কিছু শেখারই ‘একমাত্র সঠিক নিয়ম’ বলে কিছু নেই। তাই উপরের লেখা থেকে যারা পুরোপুরি বুঝবে না তাদের অনেকে নিশ্চয়ই আপনার মন্তব্যটা থেকে উপকৃত হবে। 🙂

গুবরে ফড়িং জুলাই 10, 2014 at 11:32 অপরাহ্ন - Reply

কঠিন, এতই কঠিন যে এত সহজ করে বোঝানোর পরও মাথার উপ্রে দিয়ে গেল।
তবু অনেক বিজ্ঞান ও গণিতমনস্ক তরুণ ঠিকই বুঝবে এবং তাদের জন্য বাংলা ভাষায় লেখাটা জরুরিও।
আপনার লেখাগুলি নিয়মিত আসবে, খুবই স্বল্পবিরতি নিয়ে, সেই প্রত্যাশা...

তানভীরুল ইসলাম জুলাই 12, 2014 at 9:33 অপরাহ্ন - Reply

@গুবরে ফড়িং,

আরে না না! কঠিন কিছু না। আমিই বোধ হয় পারিনি বুঝিয়ে লিখতে। লেখাটা অবশ্য এক বসায় পড়ার জন্য লেখা না। রিতীমত হোমওয়ার্কটোমওয়ার্কও দেওয়া আছে। পড়তে গিয়ে ঘুম আসা খুব স্বাভাবিক। 😛

রামগড়ড়ের ছানা জুলাই 16, 2014 at 9:08 পূর্বাহ্ন - Reply

@গুবরে ফড়িং,
আসলে এটা একটু "স্পেশালাইজড" লেখা, যারা টুকটাক প্রোগ্রামিং জানে তাদের জন্য। আমি সহ আরো কয়েকজন মুক্তমনায় প্রোগ্রামিং নিয়ে লিখলেও সেগুলো ছিল 'পপুলার সাইন্স' ধরনের। সেই হিসাবে মুক্তমনায় প্রথমবারের মত সত্যিকারের প্রোগ্রামিং নিয়ে লেখার জন্য তানভীর ভাইকে অভিনন্দন দেয়া যেতে পারে 🙂 ।

অভিজিৎ জুলাই 10, 2014 at 10:45 পূর্বাহ্ন - Reply

অর্ধেক পড়লাম। আয়েশ করে পড়তে হবে।
যতটুকু পড়লাম দুর্দান্ত লেগেছে।

পড়তে পড়তে মনে হল, আপনার বাংলায় প্রোগ্রামিং এর উপর একটা বই লেখা উচিৎ।
আর ২য় বইটা হওয়া উচিৎ কোয়ান্টাম পদার্থবিদ্যার উপরে। এত সুন্দর আর সহজ করে কোয়ান্টাম পদার্থবিদ্যার প্রান্তিক বিষয়গুলোর উপর আলোচনা বাংলা ভাষায় আমি আর কোথাও পাইনি (হ্যা এমনকি অধ্যাপক জাফর ইকবালের কথা মাথায় রেখেও বলছি)। আপনার আরো লেখা উচিৎ।

আমাদের দুর্ভাগ্য, প্রকাশকের গাফিলতির কারণে আপনার করা স্টিফেন হকিং এর গ্র্যান্ড ডিজাইনের অনুবাদটা আলোর মুখ দেখল না। কিন্তু আমি যেখানে পারি সেটা রেফার করতে চেষ্টা করি। মীজান ভাইয়ের (ড. মীজান রহমান) এর সাথে 'শূন্য থেকে মহাবিশ্ব' নামের প্রকাশিতব্য বইটাতেও বহু জায়গায় আপনার কাজের রেফারন্স দিয়েছি।

ভাল থাকবেন।

তানভীরুল ইসলাম জুলাই 10, 2014 at 12:09 অপরাহ্ন - Reply

@অভিজিৎ দা,

আপনি হঠাৎ করে আমাকে "আপনি আপনি" শুরু করলেন কেন? বেশি বস হয়ে গেলাম নাকি 😛
অবশ্য খেয়াল করে দেখলাম এক বছর পর পর একেক্টা লেখা পোস্ট করছি। ভুলে যেতেও পারেন 😕

> *পড়তে পড়তে মনে হল, আপনার বাংলায় প্রোগ্রামিং এর উপর একটা বই লেখা উচিৎ।*

আসলে শাফায়েতের সাথে একটা অ্যালগরিদমিক প্রোগ্রামিং এর বই লেখার কাজ চলছে। এইটা তার একটা চ্যাপ্টারের শুরুর অংশ। প্রোগ্রামিং কন্টেস্ট সম্পর্কিত অ্যাডভান্সড কিছু ধারনা নিয়ে শাফায়েত আরো কিছু যোগ করবে।

> *আর ২য় বইটা হওয়া উচিৎ কোয়ান্টাম পদার্থবিদ্যার উপরে।*

এ রকম একটা পরিকল্পনা আছে আসলে। কম্পিউটার বিজ্ঞানে প্রাথমিক ধারনা আছে এমন অডিয়েন্সের জন্য একটা ইন্ট্রোডাক্টরি টিউটোরিয়াল লিখেছি। ইংরেজীতে। সেটাকে বাংলা করলেই একটা বই হয়ে যেতে পারে। এ ছাড়াও বেশ কিছু প্রেজেন্টেশন ম্যাটেরিয়াল তৈরি করেছি। সেগুলো নিয়ে একটা ভিডিও লেকচার সিরিজও করা যায়। আগামী কয়েক মাসের মধ্যেই এইসব শেষ করতে

চেষ্টা করবো। কিন্তু, জাফর স্যারের কথা মাথায় রাখার কথা বলে লজ্জা দিয়ে দিলেন।

গ্র্যান্ড ডিজাইন প্রকাশিত না হওয়াটা আসলেই দুর্ভাগ্য। আপনি অনেক সাহায্য করেছিলেন ওই সময়। র্যান্ডম-হাউসের সাথে কথাও বলেছিলেন। কিন্তু হায়, আমাদের দেশে যদি আরো প্রফেশনাল কিছু প্রকাশক থাকতো! অবশ্য অনুবাদটা মুক্তমনাতে তো আছেই কেউ চাইলে পড়তে পারবে। 🙂

> *অর্ধেক পড়লাম। আয়েশ করে পড়তে হবে।*

কোথাও কোনো ভুল-ভ্রান্তি থাকলো কি না, বা কোনো অংশ অন্যভাবে লিখলে ভালো হত কি না, বা কিছু বাদ পড়লো কি না, এসব কিছু চোখে পড়লে জানাবেন। 🙂

ডিম ডিসেম্বর 22, 2015 at 10:33 পূর্বাহ্ন - Reply

রিকার্সিভ প্রোগ্রামের উপর আপনার এই লেখাটা পড়ে অনেক ভাল লেগেছে।
আপনাদের algorithm এর উপর বইটা কি প্রকাশিত হয়েছে ??
যদি প্রকাশিত হয়ে থাকে তবে নামটা জানান আর
যদি না হয়ে থাকে তবে কবে হবে তার একটা সম্ভাব্য সময় বলবেন।
আমি আপনাদের বই পড়তে অনেক আগ্রহী।
🙂 🙂

তামান্না ঝুমু জুলাই 9, 2014 at 9:34 অপরাহ্ন - Reply

অনেকদিন পর আপনার লেখা পেলাম। চিরদিন যেন লেখা পাই।

তানভীরুল ইসলাম জুলাই 10, 2014 at 12:12 অপরাহ্ন - Reply

@তামান্না ঝুমু,
আমিন 🙂

Leave A Comment

Comment...

Name (required) Email (required) Website

POST COMMENT