

This is PDF version of Shafaet's Planet Blog

(Education purpose only)

Compiled & Edited by: *Ratul Mahjabin*

Table of contents

Motivation

- Why should I learn programming?
- Why should I study computer science?
- Programming contest and Online Judge
- Confused programmer

Algorithm Basic

- Big O notation
- complexity class
- Halting problem
- Binary search - 1
- Binary search - 2
- Floyd cycle finding algorithm

Data structure

- Data Structure : LinkedList
- Data Structure : Stack
- Data Structure : Queue and Circular Queue
- Data Structure : Sliding Range Minimum Query
- Data Structure : Disjoint set(Union Find)
- Data Structure : Trie (Prefix/Radix Tree)
- Data Structure : Segment Tree - 1
- Data Structure : Segment Tree - 2 (Lazy Propagation)
- Array Compression
- Lowest Common Ancestor
- Binary Indexed Tree
- Square Root Decomposition
- Algorithm : L-R-U Cash
- Probabilistic Data Structure: Bloom Filter

Graph Theory

- Graph Theory Intro - 1
- Graph Theory Intro -2(Variable Graph store -1)
- Graph Theory Intro -3 (Variable Graph store -2)
- Breadth First search
- Minimum Spanning Tree : Prim
- Minimum Spanning Tree : Kruskal
- Topological sort
- Depth First search and again topological sort
- Dijkstra
- Floyd Warshall
- Bellman Ford
- Strongly Connected component
- Maximum Flow 1
- Maximum Flow 2
- Minimum vertex problem
- Stable Marriage problem
- Tree Diameter
- Longest Path Problem

MISC

- Direction Array
- Meet in the middle
- Tail Recursion optimization

Game Theory

- Game theory 1
- Game theory 2(Nim Game)
- Game Theory 3(Sprague Grundy numbers)

Dynamic Programming

- Dynamic Programming Intro - 1 (Fibonacci)
- Shortest path
- Longest Increasing subsequence, Path printing
- Longest Common subsequence
- Coin change, 0-1 Knapsack
- Subset sum, Combinatorics, Decision Problem
- Bitmask
- Minimum Vertex Cover problem

Backtracking

- Permutation Generator

Math

- Modular Arithmetic
- Prime Generator: Sieve of Eratosthenes
- Bitwise Sieve
- Arrangement and De-arrangement
- Probability : Expected Value

String

- Rabin Karp String Matching
- KMP

কেন আমি প্রোগ্রামিং শিখবো?

জানুয়ারি ২১, ২০১৩ by [শাফায়েত](#)



সকালে উঠেই [টপকোডারে](#) এ লেখা দেখলাম “একটি শিশুকে একই আইফোন দিলে সে দিনবাত অ্যাংগু বার্ডস খেলবে, শিশুটিকে কোডিং শিখালে সে আইফোনটার জন্য সফটওয়্যার 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 ছোট 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। যদতো 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 বেখেছে 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 কতৃকমের 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

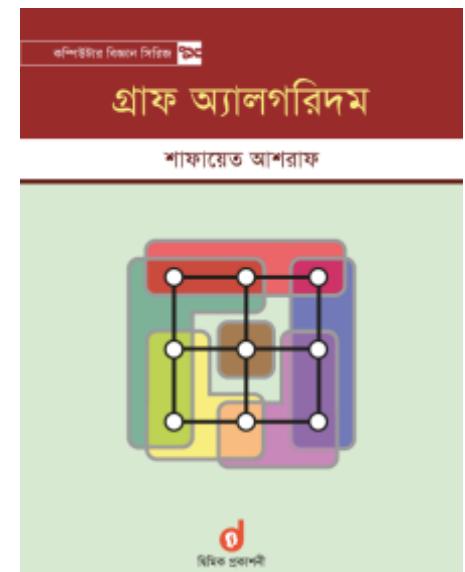
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



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 সবচুকুই 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? রাষ্যিয়া-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 পাণ্টে যাবে। সে তখন সারাদিন গেমস না খেলে হয়তো একটি গেমস বানিয়ে ফেলবে। আমি বাংলাদেশেরই কিছু স্কুল-কলেজ পড়ুয়া প্রোগ্রামারদের জানি যাবা বাংলা কিবোর্ড নিয়ে কাজ করে, ওপেন সোর্স কমিউনিটিতে অবদান রাখে। প্রোগ্রামিং জানলে তুমি বুঝতে পারবে কম্পিউটার শুধু বিনোদনের যন্ত্র নয়, কম্পিউটার তৈরী করা হয়েছিল এর ক্ষমতাকে ব্যবহার করে বড় বড় গবেষণা, হিসাব করার জন্য, তুমি যদি গবেষণা নাও করো অন্তত এই ক্ষমতাটা ব্যবহার শিখবে,


Shafaetsplanet
2.4K likes
শাফায়েত অ্যালগরিদম
শাফায়েত আশরাফ

[Like Page](#)

প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

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 কনটেন্ট 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 মন্তিষ্ঠের লজিকাল সেটৱটা ডেভেলপ করে। আমার মতে চিন্তা করার মত আনন্দের এবং গুরুত্বপূর্ণ কাজ ২য়টি নেই। বিশেষ করে কম বয়সে প্রোগ্রামিং শিখালে সে চিন্তাপক্ষি বৃদ্ধির যেই সুফলটা পাবে সেটা সারাজীবন কাজে লাগবে, সে যদি প্রোগ্রামিং পরে ছেড়েও দেয় তারপরেও চিন্তা করার ক্ষমতাটা থেকে যাবে।

প্রোগ্রামিং কি শুধু কম্পিউটার সাইন 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 মূহর্তে 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 হ্যাঁ, 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 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।

স্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল ট্রি/বেডিল ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

The screenshot shows the LightOJ categorized problems page for user Shafaeet Ashraf. The left sidebar has a 'VOLUME' section with links to Volume Home, Problem Set, and Problem Category. The main content area displays 'Categorized Problems for Shafaeet Ashraf'. A table lists problems under the 'Beginners' category, with sub-categories like 2-SAT, Articulation/Bridge/Biconnected Component, Bellman Ford, Breadth First Search/Depth First Search, Cycles/Topological Sorting/Strongly Connected Component, Dijkstra/Floyd Warshall, Directed Minimum Spanning Tree, Euler Trail/Circuit, Heavy-Light Decomposition, Minimum Spanning Tree, and Stable Marriage Problem. Each row in the table shows the problem name, a list icon, solved/total counts (e.g., 11/39), and solved/tried counts (e.g., 11/12).

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

গুরু কিভাবে করবে? তোমার যদি ইন্টারনেট কানেকশন থাকে তাহলে কথাই নেই, ইন্টারনেটে অসংখ্য টিউটোরিয়াল আছে। ইংরেজীর পশাপাশী বাংলা কিছু ভালো রিসোর্সও তুমি পাবে। যেমন শুঙ্খেয় 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, ব্যক্তিগত 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, হার্ডার্ড শিল্ডের বইয়ের মত নামকরা এবং ভালো বই দেখে শিখে, 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 তোমার বেশ ঝামেলা লাগবে কারণ বিষয়টা নতুন, একটু পরপর আটকে যাবে, তারপর হঠাৎ

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

দেখবেন সবকিছু সহজ হয়ে গিয়েছে, মূল্যের 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 বছু 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। প্রথিবীর আরেক প্রান্তের মানুষ তোমার বানানো সফটওয়্যার দিয়ে গান শুনবে, তোমার অপারেটিং সিস্টেম বুট করবে, আবার পিসি হ্যাং করলে হ্যতো তোমাকেই 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 সাইনের স্টুডেন্ট হও তাহলে এইসব কথাই তুমি হ্যতো জানো, শুধু বলবো প্রোগ্রামিং কে আর তো সাবজেক্টের মতো ভেবোনা, খালি সিজিপিএ বাড়তে কোডিং শিখলে তোমার মতো অভাগা কেও নাই, প্রোগ্রামিং উপভোগ করার চেষ্টা করো, জানার আনন্দে শিখো।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

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, অ্যালগরিদম 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 ভূমিকা 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 হ্যনা, যার ভালো লাগবে সে শিখবে তবে সবাই অঙ্গত জানুক প্রোগ্রামিং কি, এছাড়া কিভাবে শেখার জন্য উৎসাহিত হবে? অনেকেই ইউনিভার্সিটিতে আসার আগে জানেনা প্রোগ্রামিং বলে একটা বষ্ট 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 আবেকষ্টি 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!

ফেসবুকে মন্তব্য

12 comments

12 Comments

Sort by Oldest

Create PDF in your applications with the Pdfcrowd [HTML to PDF API](#)

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

 **Shafaetsplanet**
2.4K likes
শাফায়েত অ্যালগরিদম
শাফায়েত আশরাফ
[Like Page](#)

Be the first of your friends to like this



Shafaetsplanet
52 minutes ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিঙের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

*i, W) = 1 if W =
i + 1, W - C[i]) -*

SHAFAEETPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / ম্যাপ

PDFCROWD

কম্পিউটার বিজ্ঞান

আগস্ট ৮, ২০১৩ by শাফায়েত



“আলকেমিস্ট” বইয়ের রুটিওয়ালা স্বপ্ন দেখতো পর্টেক হবার। কিন্তু সেই স্বপ্নকে সত্যি করার জন্য রাখালবালক হয়ে দেশবিদেশ ঘুরে না বেড়িয়ে সে স্বপ্নকে চাপা দিয়েছিলো কারণ মানুষের চোখে রুটিওয়ালার জীবন রাখালবালকের থেকে বেশি সম্ভানের। তাই সমাজকে খুশি করতে গিয়ে তার স্বপ্ন পূরণ করা হ্যনি। আমরা বেশিভাগই একেকজন রুটিওয়ালা, নিজের স্বপ্ন পূরণ না করে আগে চিন্তা করি কি নিয়ে পড়ালেখা করলে বেশি টাকার চাকরী পাওয়া যাবে। একটা বিষয়ের “ব” না জেনে হট করে বিশ্ববিদ্যালয়ে ভর্তি হয়ে যাই চাকরির আশায়, তারপর একসময় হতাশা যিরে ধরে।

কম্পিউটার সায়েন্স খুবই জনপ্রিয় একটা বিষয়। এই বিষয়টা নিয়ে মানুষের মধ্যে অনেক ভুল ধারণা প্রচলিত আছে, এই সাবজেক্টটাতে কি পড়ানো হয় এ বিষয়ে সবার ভালো ধারণা নেই। সবথেকে দুঃখজনক ব্যাপার হলো এই সাবজেক্টে কি পড়ানো হয় সেটা না জেনেই বহু ছাত্রছাত্রী ভর্তি হয়ে যায় চাকরীর বাজার ভালো এই কারণে, এটা শুধু কম্পিউটার সায়েন্স না বরং বাংলাদেশে যেকোনো সাবজেক্টের জন্যই সত্য। এরপর যেটা হয় সেটা হলো খাপ খাওয়াতে না পেরে অনেকেই হতাশ হয়ে পড়ে।

আমার সৌভাগ্য যে কম্পিউটার সায়েন্স সম্পর্কে ভালো না জানলেও প্রোগ্রামিং আমার আগে থেকেই ভালো লাগতো এবং কম্পিউটার নিয়ে অনেক আগ্রহ ছিলো ছোটোবেলা থেকে তাই এখন আনন্দময় সময় কাটাতে পারছি, কিন্তু সবার এই সৌভাগ্য হ্যনা, এবং এটা শুধু সিস্টেমের দোষ না সাথে নিজেদের অসচেতনতাও বিশাল অংশে দায়ী। আমি অনেক বন্ধুবান্ধবকে দেখেছি ভর্তি পরীক্ষায় ভালো রেজাল্ট করে চাকরীর বাজার দেখে সাবজেক্ট বেছে নিয়ে এখন হতাশ হয়ে পড়েছে।

“We are told from childhood onward that everything we want to do is impossible.

We grow up with this idea, and as the years accumulate, so too do the layers of

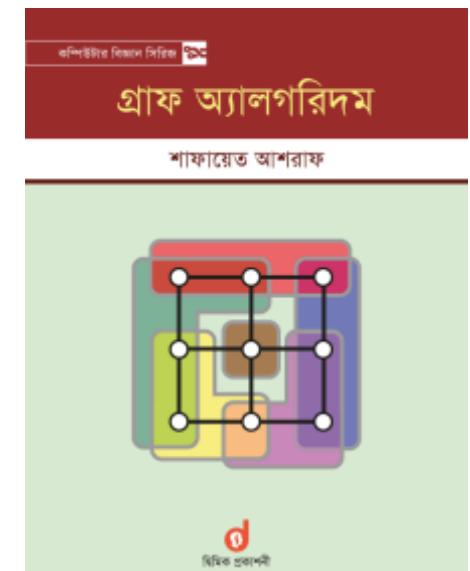
সাবক্ষাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



“prejudice, fear and guilt. There comes a time when our personal calling is so deeply buried in our soul as to be invisible. But it's still there” – The alchemist,Paulo Coelho”

আমার এই লেখার উদ্দেশ্য কম্পিউটার সায়েন্সে কি কি পড়ানো হয় সেটা সবাইকে জানানো, বিশেষ করে কম বয়সীদের যারা এখনো কোনো বিষয় বেছে নেয়নি এবং তাদের অভিভাবকদের। তর্তি পরীক্ষা নিয়ে অনেক গাইডলাইন থাকলেও সাবজেক্ট নিয়ে গাইডলাইনের অভাব আছে। আর যেগুলো আছে সেগুলোতেও কম্পিউটার সায়েন্সে ক্রিয়েটিভিটি লাগে, মুখস্থ কম করতে হয় এসব কথাবার্তা শুধু লিখে রাখসে, আসলে কি কি পড়ানো হয় সেটা লিখেনি।

বাংলাদেশের মানুষের এক অংশের ধারণা কম্পিউটার সায়েন্সে ওয়ার্ড, পাওয়ারপয়েন্ট এসব শেখানো হয়, এবং এগুলো যেহেতু পাড়ার দোকানদারও ভালো পারে তাই কম্পিউটার সায়েন্স পড়ার কোনো মানে নাই। আবার কম্পিউটার সম্পর্কে ভালো জানে, টুকটাক প্রোগ্রামিংও কিছুটা জানে এমন মানুষের ধারণা এখনে শুধু এইচটিমিএল, পিএইচপিতে এ ওয়েবসাইট বানানো শেখায়, যেগুলো কম্পিউটার সায়েন্স না পড়লেও শেখা যায়, ফিল্যাঞ্চারদের মধ্যে এই ধারণাটা বেশি কাজ করে।

এখন আমরা তাহলে এক এক করে দেখি কম্পিউটার সায়েন্সে কি কি টপিক পড়তে হবে। টপিকগুলো সম্পর্কে সংক্ষেপে আমরা জানবো। এরপরে নিজেই সিদ্ধান্ত নেয়া যাবে উপরের ধারণা গুলো কতটা সত্যি। টপিকগুলো সম্পর্কে জেনে একজন কাজ হবে তোমার ভেবে দেখা এই সাবজেক্টটা কি সত্যিই তোমার পছন্দ নাকি তুমি ইলেকট্রনিক্স, মেকানিক্স, ফিজিক্স বা অন্য কোনো সাবজেক্ট পড়বে।

প্রোগ্রামিং: কম্পিউটার সায়েন্স বললে অবধারিত ভাবে প্রথমে চলে আসে প্রোগ্রামিং। প্রোগ্রামিং হলো কম্পিউটারকে কথা শুনানোর উপায়, বোকা কম্পিউটারকে দিয়ে ইচ্ছামত কাজ করিয়ে নেয়া। কম্পিউটার যেহেতু মানুষের ভাষা বুঝেনা তাকে বোঝাতে হয় বিশেষ ভাষায় যাকে বলে প্রোগ্রামিং ল্যাংগুয়েজ। প্রোগ্রামিং ল্যাংগুয়েজ শিখে সফটওয়্যার, ওয়েবসাইট যেমন বানানো সম্ভব তেমনি গাণিতিক সমস্যা সমাধান করা সম্ভব, রকেটের গতিপথ নির্ণয় করা সম্ভব, কোয়ান্টাম মেকানিক্স নিয়ে গবেষণা সম্ভব, ডিএনএ অ্যানালাইসিস করা সম্ভব। এককথায় বলতে গেলে প্রোগ্রামিং এর জ্ঞান আধুনিক যুগের সুপারপাওয়ার যেটা দিয়ে পৃথিবীকে নিয়ন্ত্রণ করা সম্ভব।

বাংলাদেশে বেশিভাগ বিশ্ববিদ্যালয়ে প্রথম সেমিস্টারে শিখানো হয় সি ল্যাংগুয়েজ যেটাকে বলা যেতে পারে প্রোগ্রামারদের মাত্রভাষা। এরপরে অবজেক্ট ওরিয়েন্টেড প্রোগ্রামিং কোর্সে শিখানো হয় জাভা। সফটওয়্যার প্রোগ্রামিং এবং বড় বড় প্রজেক্টের জন্য জাভা জনপ্রিয়। কোনো কোনো



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে
হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইনেক্সেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগোরিদম

ডাটা স্টোরেজ(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

বিশ্ববিদ্যালয়ে জাভা শেখানোর সময় অ্যান্ড্রয়েডে প্রোগ্রামিং শেখানো হয়।

প্রোগ্রামিং ল্যাংগুয়েজ কোর্স দুই ভাগে করানো হয়। ক্লাসরুমে কিছু থিওরিটিকাল কথাবার্তা থাকে তবে এই কোর্সের মূল অংশ হ্য ল্যাবে। সেখানে প্রোগ্রামিং ল্যাংগুয়েজ দিয়ে বিভিন্ন সমস্যা সমাধান করতে দেয়া হয়। সবশেষে সাধারণত একটা প্রজেক্ট করতে দেয়া হয়, সেখানে স্টুডেন্টরা ক্রিয়েটিভিটি দেখানোর সুযোগ পায়।

কম্পিউটার সায়েন্সের আরো অনেক অংশ থাকলেও ভালো প্রোগ্রামিং জানা এই সাবজেক্টে ভালো করার পূর্বশর্ত।

পরবর্তীতে অ্যাসেম্বলী ল্যাংগুয়েজের জন্য আরেকটি কোর্স থাকে। এখানে একদম লো লেভেলে সরাসরি মেমরির বিভিন্ন অংশ নিয়ে কাজ করা যায়। অ্যাসেম্বলী ল্যাংগুয়েজ জানতে সাহায্য করে কম্পিউটার কিভাবে মেমরিতে বিট/বাইট হিসাব করে ডাটা রাখে, একদম হার্ডওয়্যার লেভেলে কিভাবে ডাটা রাখা হ্য সেটা তুমি জানতে পারবে। অ্যাসেম্বলী ভালোভাবে শিখলে ভাইরাস বানানোর মতো মজার কাজ করা সম্ভব, প্রোগ্রামিং করে মেমরির বিভিন্ন অংশ কট্টোল করা সম্ভব।

অ্যালগোরিদম: তুমি কি জানো কিভাবে বড় একটা ফাইলকে কমপ্রেস করে সাইজ কমিয়ে ফেলা সম্ভব হ্য? গুগল কিভাবে লাখ লাখ ওয়েবসাইট থেকে চোখের পলকে দরকারি ডাটা খুজে আনে? কিভাবে লাখ লাখ টেরাবাইটের ডিএনএ সিকোয়েন্স থেকে খুজে বের করা হ্য জীবনের রহস্য? এধরণের প্রবলেম সলভিং এব হাতেখড়ি হ্য অ্যালগোরিদম কোর্স। অ্যালগোরিদম হলো একটি সমস্যাকে সমাধান করার বিভিন্ন ধাপ। অ্যালগোরিদম কোর্সে মূলত শেখানো হ্য প্রবলেম সলভিং টেকনিক। সেখানে এধরণের রিয়েল লাইফ প্রবলেম সরাসরি সলভ করা শেখাবেনা, তবে এই কোর্সটা একটা বেস টেকনিক করবে, কিছু কমন প্রবলেম সলভিং টেকনিক শিখিয়ে দিবে। কম্পিউটারে মেমরি এবং সময় কম খরচ করে কিভাবে সমস্যা সমাধান করা যায় সেগুলো শেখানো হ্য এখানে। এই কোর্সও ল্যাব এবং থিওরী দুটোই থাকে। এই কোর্সটা করার সময় তুমি বুঝতে পারবে যে শুধু প্রোগ্রামিং ল্যাংগুয়েজ জেনে কোনো লাভ নেই, সেটা ব্যবহার সমস্যা সমাধান করতে জানতে হবে।

ডাটা স্ট্রাকচার: ফেসবুক কিভাবে এত মানুষের তথ্য সংরক্ষণ করে? এলোমেলো ভাবে সংরক্ষণ করলে তাড়াতাড়ি খুজে পাওয়া সমস্যা, তাই তথ্য সংরক্ষণ করার নির্দিষ্ট কিছু টেকনিক আছে। ড্রয়ারে লেভেল করে কাগজপত্র রাখলে যেমন সহজে খুজে পাওয়া যায় তেমনি কিছু নির্দিষ্ট স্ট্রাকচার ফলো করে ডাটা সেভ করলে সহজে সেটা কাজের সময় পাওয়া যায়। ডাটা স্ট্রাকচার কোর্স এগুলো সম্পর্কে ধারণা দেয়া হ্য।

অ্যালগোরিদম এবং ডাটা স্ট্রাকচার হলো কম্পিউটার সায়েন্সের মূল ভিত্তি, প্রতিটা টপিকে এগুলো কাজে লাগে।

শ্লাইডিং রেজ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

গণিত: কম্পিউটার সায়েন্সের স্টুডেন্টদের ভালো গণিত জানার দরকার হয়। এটার কারণটা সবার কাছে পরিষ্কার না। নতুন অ্যালগরিদম বা ডাটা স্ট্রাকচার ডিজাইন করার সময় এগুলো কতটা ভালো কাজ করবে সেটা নির্ধারণ করতে গণিত দরকার হয়। একটা সমস্যা অনেক ভাবে সমাধান করা যায়, কোন পদ্ধতিটা সবথেকে ভালো, কোনটা কম মেমরিতে কম সময়ে কাজ করবে এসব হিসাবের জন্য গণিতের জ্ঞান খুব দরকার। তুমি যত ভালো গণিত জানবে তোমার প্রবলেম সলভিং স্কিল তত ভালো হবে। কম্পিউটার সায়েন্সকে এজন্য “অ্যাপ্লাইড ম্যাথ” বলতেও শুনেছি অনেককে।

গণিতের অনেক কঠিন কঠিন সমস্যাও আজকাল কম্পিউটার দিয়ে সলভ করা হয়, এমনকি থিওরেম প্রমাণও করা হয়। গণিতের মধ্যে জানা লাগবে মূলত কম্পিউটেরিঙ্গ, প্রোগ্রামিং, নাম্বার থিওরি, জিওমেট্রি এবং লিনিয়ার অ্যালজেব্রা ইত্যাদি। এছাড়া কিছু ক্যালকুলাস শেখানো হয়। প্রোগ্রামিটির জন্য আলাদা কোর্স করানো হয় এবং খুবই ইটারেন্সিং কিছু প্রবলেম সলভ করানো হয় সেখানে। কারো যদি গণিত ভালো লাগে তার জন্য কম্পিউটার সায়েন্সে ভালো করা সহজ হয় যায়। আবার কেও গণিতের রিয়েল লাইফ অ্যাপ্লিকেশন শিখতে চাইলেও এটা তার জন্য একটা ভালো সাবজেক্ট হতে পারে।

অপারেটিং সিস্টেম প্রোগ্রামিং: এখানে শিখানো হয় কিভাবে অপারেটিং সিস্টেম কাজ করে। তবে তারমানে এই না যে উইন্ডোজ চালানো শেখানো হয়! এখানে শিখায় অপারেটিং সিস্টেমের ইন্টারনাল স্ট্রাকচার কিরকম। কম্পিউটারে একই সাথে ১০টা কাজ করলে অপারেটিং সিস্টেমকে ১০টা কাজের জন্য বিশেষ শিডিউল তৈরি করতে হয়, সেগুলো এখানে শিখানো হয়। তারপর ধরো ২টা প্রোগ্রাম একই সাথে প্রিন্টার ব্যবহার করতে চাইলো, কাকে অপেক্ষায় রেখে কাকে দিবে এ ধরণের রিসোর্স ম্যানেজমেন্টের কাজও শিখানো হয়।

সিস্টেম প্রোগ্রামিং এ মুক্তসোর্স অপারেটিং সিস্টেম ইউনিল্ক/লিনাক্সের সোর্সকোড নিয়ে ঘাটাঘাটি করা হয়। এই কোর্সটা ভালোমত করলে শেখা যাবে কিভাবে মডেম বা বিডিএল ডিভাইসের জন্য ড্রাইভার তৈরি করা যায়, কিভাবে অপারেটিং সিস্টেমের ডিতরে কোড মডিফাই করা যায়। ডিবিয়েটে কেও অপারেটিং সিস্টেম তৈরি করতে চাইলে বা এই সম্পর্কিত বিসার্চ করতে চাইলে সেটার ব্যাকগ্রাউন্ড তৈরি হয় এখানে।

ডাটাবেস: ওয়েবসাইট বা বড়বড় সফটওয়্যার বিশাল ডাটা রাখে “ডাটাবেস” এর ডিতর। ডাটাবেসে কিভাবে ডাটা রাখতে হয়, কিভাবে সেখান থেকে ডাটা নিয়ে আসতে হয় ইত্যাদি নিয়ে এই কোর্স।

আর্টিফিশিয়াল ইন্টেলিজেন্স(এ.আই): নাম শুনেই বোঝা যাচ্ছে কি নিয়ে এই কোর্স, যিসিসের জন্য খুবই পপুলার একটা টপিক এটা। কিভাবে রোবটকে দিয়ে কাজ করানো যায়, গেমের ক্যারেক্টার নিজে নিজে বিডিএল সিন্ড্রেন্স নেয়, কম্পিউটার কিভাবে দাবা খেলে এই ধরণের দারুণ ইটারেন্সিং

আর্টিফিশিয়াল ইন্টেলিজেন্স পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্রিমাম ফ্লো-১

ম্যাক্রিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এবং ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

সব টপিক এ.আই এর অগ্রভূত। ল্যাবে প্রোগ্রামিং করে এসব টেকনিক ইমপ্লিমেন্টও করতে হয়। গেম প্রোগ্রামিং বা রোবোটিক্স এ আগ্রহ থাকলে এই কোস্টি তোমার খুব পছন্দ হবে। নিউরাল নেটওয়ার্ক, জেনেটিক অ্যালগোরিদমের মতো দারুণ সব জিনিস শিখতে পারবে।

কম্পাইলার: সাদামাটা ভাষায় কম্পাইলার জিনিসটার কাজ প্রোগ্রামিং ল্যাংগুয়েজকে মেশিন ভাষায় পরিণত করা যাতে হার্ডওয়্যার সেটা বুঝতে পারে। প্রোগ্রামিং ল্যাংগুয়েজ কিভাবে কাজ করে, কিভাবে নিজের প্রোগ্রামিং ল্যাংগুয়েজ তৈরি করা যায় এগুলো নিয়েই কম্পাইলার কোর্স। প্রোগ্রামিং ল্যাংগুয়েজের ইনস্ট্রাকশন গুলো কম্পিউটার কিভাবে মেশিন কোড বানিয়ে কাজ করে সেগুলো শেখানো হয় এখানে। মানুষের ভাষার মতো প্রোগ্রামিং ভাষারও গ্রামার থাকে, সেগুলো নিয়ে জানতে পারবে।

গ্রাফিক্স: আমরা কম্পিউটার গেমস বা অ্যানিমেটেড মুভিতে এত সুন্দর গ্রাফিক্স দেখি তার পিছনে আছে প্রচুর ম্যাথমেটিকাল থিওরী। যেমন গেমসে লাইটিং, শেডিং কিভাবে বাস্তবসম্বত করা যায় তার পিছনে আছে অনেক থিওরী। গেমারদের কাছে প্রচলিত শব্দ "এন্টি এলিয়াসিং", "শেডিং" ইত্যাদি সম্পর্কে এই টপিকে বিস্তারিত পড়ানো হয়। প্রচুর জ্যামিতি দরকার হয় এখানে। যেমন একটা 3-ডি বাল্কে ৪৫ ডিগ্রী ঘূরালে কো-অর্ডিনেট কত পরিবর্তন হবে এসব হিসাব এখানে করতে হয় এবং ল্যাবে সেই অনুযায়ী গ্রাফিকাল প্রোগ্রাম লিখতে হয়, তাই শেখাটা শুধু মুখস্থে সীমাবদ্ধ থাকেনা। তুমি কলেজে থাকতে হয়তো বৃত্ত, সরলরেখার ইকুয়েশন শিখেছো, এখানে শিখবে ওগুলো ব্যবহার করে কিভাবে গ্রাফিক্স তৈরি হয়।

নেটওয়ার্কিং: কিভাবে একটা কম্পিউটারকে আরো ১০ কম্পিউটারের সাথে কানেক্ট করতে হয়, কিভাবে ইন্টারনেট কাজ করে, কিভাবে নেটওয়ার্কে ডাটা প্যাকেট পাঠানো হয় এবং অন্য প্রাণে রিসিভ করা হয় ইত্যাদি শেখানো হয়। নেটওয়ার্ক সিকিউরিটি এই টপিকের অগ্রভূত। জানতে পারবে ল্যান, ডিএনএস সার্ভার, আইপিভি-৬, ক্রিপ্টোগ্রাফি ইত্যাদি সম্পর্কে।

ডিস্ট্রিবিউটেড সিস্টেম: তুমি হয়তো জানো বড় বড় যেসব কাজে অনেক মেমরি, শক্তি দরকার হয় সেসব কাজে অনেকগুলো কম্পিউটারকে একসাথে ব্যবহার করে কাজ করা হয়। বড় বড় রিসার্চের কাজ এভাবে করা হয়, ওয়েবসাইটগুলোতেও অনেকগুলো সার্ভার একসাথে কাজ করে। কিভাবে ডিস্ট্রিবিউটেড সিস্টেম ডিজাইন করা যায় সেটা নিয়েই এই কোর্স।

সফটওয়্যার ইঞ্জিনিয়ারিং: আধুনিক সফটওয়্যার ডিজাইনের টেকনিক পড়ানো হয় এই টপিকে। সব বিশ্ববিদ্যালয়ের কারিকুলামে সম্ভবত এটা নেই।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

এখন আসি হার্ডওয়্যার রিলেটেড কিছু টপিকে। এ ব্যাপারে আগ্রহ এবং জ্ঞান কম বলে কম কথা শেষ করছি। কম্পিউটার সায়েন্সে হার্ডওয়্যার নিয়ে অনেক কিছু পড়ানো হয়। কম্পিউটার আর্কিটেকচারে কম্পিউটারে মূল গঠন পড়ানো হয়। কিভাবে কম্পিউটারের তথ্য যাবার পথ বা “ডাটা বাস” কাজ করে, এজিপির সাথে পিসিআই এবং পার্থক্য, ক্যাশ মেমরি ইত্যাদি শেখানো হয়। ইলেক্ট্রিকাল সার্কিটের মতো কম্পিউটারে থাকে বিশেষ ডিজিটাল সার্কিট, সেগুলো নিয়ে পড়ানো হয় ডিজিটাল সিস্টেমস টপিকে। এছাড়া বিভিন্ন ডিডাইস যোগ করলে কিভাবে সেটা কম্পিউটারের সাথে যোগাযোগ করে এসব পড়ানো হয় হার্ডওয়্যার অংশে। ল্যাবে বিভিন্ন ধরণের সার্কিট তৈরি করতে হয়। প্রজেক্ট অংশে অনেক ক্রিয়েটিভিটি দেখানোর সুযোগ থাকে, কেও কেও বোর্ট তৈরি করে, আমাদের ফ্লাসের একজন মোবাইল দিয়ে নিয়ন্ত্রণ করা যায় এমন খেলনা গাড়ি বানিয়েছিলো।

এই হলো মোটামুটি কম্পিউটার সায়েন্সে যা যা পড়ানো হয় তার সামানী। অনেক কিছু যতো বাদ পড়ে নিয়েছে এই মুহূর্তে মাথায় না থাকার কারণে। কম্পিউটার সায়েন্সের আকর্ষণীয় একটা দিক হলো “কনটেস্ট”। সারাবছরই বিভিন্ন বিশ্ববিদ্যালয় বা কোম্পানি প্রোগ্রামিং কনটেস্ট এবং সফটওয়্যার কনটেস্ট আয়োজন করে। প্রোগ্রামিং কনটেস্টে মূলত অলিম্পিয়াড স্টাইলে অ্যালগোরিদমের সাহায্যে প্রবলেম সল্ভ করতে হয়। এখানে সুযোগ আছে সারা বিশ্বের বড় বড় প্রোগ্রামারদের সাথে কনটেস্ট করার। বাংলাদেশের মানুষের গর্ব করার মতো জিনিস খুব বেশি নেই, তবে প্রোগ্রামিং কনটেস্ট অবশ্যই সেই অল্প জিনিসগুলোর একটা, অনেকবছর ধরেই বাংলাদেশিরা এসব কনটেস্টে ভালো ফল করছে। সফটওয়্যার কনটেস্টে মূলত বিভিন্ন সফটওয়্যার, ওয়েব সাইট ডিজাইন করতে হয়, মোবাইল বিশেষ করে অ্যান্ড্রয়েড ডিতিক মোবাইলের সফটওয়্যার কনটেস্ট বর্তমানে খুব জনপ্রিয়। মাইক্রোসফট ইমেজিন কাপের মতো বড় বড় আন্তর্জাতিক সফটওয়্যার কনটেস্টেও বাংলাদেশিরা অংশ নেয়। আবার তুমি চাইলে হার্ডওয়্যার কনটেস্টও করতে পারো, দারুণ একটা বোর্ট বানিয়ে চমকে দিতো পারো সবাইকে।

আবেকষ্টি আকর্ষণীয় দিক হলো গবেষণার সুযোগ। কম্পিউটার সায়েন্স শেষ বর্ষে কিছু ক্রেডিট থাকে গবেষণা বা প্রজেক্টের জন্য। প্রতি বছরই বাংলাদেশের অনার্সের ছাত্ররা ভালো ভালো জার্নালে পেপার পাবলিশ করে থাকে। কম্পিউটার সাইনের গবেষণার একটা সুবিধা হলো নিজের ডেস্কটপ কম্পিউটার ব্যবহার করেই বড় বড় গবেষণা করে ফেলা যায়, কোটি টাকা যন্ত্রপাতির দরকার হয় না(অবশ্যই দরকার হয়, তবে সেগুলো ছাড়াও অনেক কাজ করা যায়)।

কম্পিউটার সায়েন্স অনার্স করে তুমি চাইলে অন্যান্য বিষয় নিয়েও পড়ালেখা করতে পারো কারণ এখন সবকাজে কম্পিউটার দরকার হয়। যেমন রিসেন্টলি অ্যাস্ট্রোইনফরমেটিক্স নামের একটা সাবজেক্ট নিয়ে গবেষণা হচ্ছে, এদের কাজ মহাকাশ গবেষণায় কম্পিউটারকে আরো ভালোভাবে কিভাবে ব্যবহার করা যায় সেটা নিয়ে কাজ করা। তুমি চাইলে ফিজিক্সের লাইনে গিয়ে কোয়ান্টাম কম্পিউটার নিয়ে কাজ করতে পারো।

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Be the first of your friends to like this

Shafaetsplanet 52 minutes ago
ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিঙের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হয়।

$i, W) = 1 \text{ if } W =$
 $i + 1, W - C[i]) -$

SHAFAEtsPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / ম্যার

আবার বায়োলজী ডালো লাগলে বায়োইনফরমেটিক্স নিয়ে পড়ালেখা করতে পারো, কাজ করতে পারবো ন্যানোটেকনোলজী, ডিএনএ/প্রোটিন অ্যানালাইসিস নিয়ে। এরকম হাজারটা সম্ভাবনা তোমার সামনে থাকবে।

আশা করি লেখাটা পড়ার পর কম্পিউটার সায়েন্সে কি পড়ানো হয় সে সম্পর্কে একটা ধারণা তৈরি হয়েছে এবং ভুল ধারণাগুলো ভেঙে গিয়েছে। চাকরির বাজার নিয়ে কিছু বলবোনা, যে আগেই চাকরির চিন্তা করে সাবজেক্ট চয়েস করে তার জন্য এ লেখা নয়, তবে তারপরেও শুধু বলে রাখি বর্তমানে কম্পিউটার সায়েন্স পরে আমি কাওকে বেকার বসে থাকতে দেখিনি।

ভর্তি পরীক্ষার্থীদের বলবো, যেসব সাবজেক্ট পড়ার কথা ভাবছো প্রতিটা সম্পর্কে ভালোভাবে জানার চেষ্টা করো, সেখানে কি কি পড়ানো হয় সেটা খোজ নাও, এবং যদি সেগুলো তোমার সত্ত্বেই পছন্দ হয় তাহলেই ভর্তি হয়ে যাও, কোনো সাবজেক্টেই ১ নম্বর-২ নম্বর না, ইলেক্ট্রিকাল মোটরও মেকানিকালের থেকে “ভালো সাবজেক্ট” না, সেটাই তোমার কাছে “ভালো সাবজেক্ট” যেটা তোমার পছন্দ, সেটা যদি পুষ্টিবিজ্ঞানের মতো মানুষের চেখে পিছের দিকের সাবজেক্ট হয় তাহলেও সেটা ভালো, মানুষই তৈরি বিভাজন অনুসরণ করলে নিজের সাথে প্রতারণা করা হয়।

সকল ভর্তি পরীক্ষার্থীদের প্রতি শুভকামনা, আশা করি ভয়াবহ এই সময় সহজে পার করে ফেলতে পারবে।

“What’s the world’s greatest lie?” the boy asked, completely surprised. “It’s this: that at a certain point in our lives, we lose control of what’s happening to us, and our lives become controlled by fate. That’s the world’s greatest lie.”- The alchemist,Paulo Coelho

ফেসবুকে মন্তব্য

18 comments

18 Comments

Sort by Oldest

প্রোগ্রামিং কনটেস্ট এবং অনলাইন জাজে হাতেখড়ি

জানুয়ারি ৭, ২০১৩ by [শাফায়েত](#)

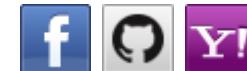


(আমাকে অনেকে মেইল করে জিজ্ঞাসা করে কিভাবে সে প্রোগ্রামিং কনটেস্টের সাথে যুক্ত হতে পারে? তাদের হ্যাতো কয়েক বাকে কিছুটা বুঝিয়ে বলার চেষ্টা করি কিন্তু সব কথা বা গাইডলাইন এভাবে দেয়া সম্ভব হ্যনা। **এই লেখাটা তাদের জন্য যারা সামান্য প্রোগ্রামিং শিখেছে বা শেখার কথা ভাবছে এবং প্রোগ্রামিং কনটেস্টে অংশ নেবার কথা ভাবছে।** আমি যখন প্রথম শুরু করেছিলাম তখন যেসব প্রশ্ন মাথায় এসেছিলো বা অনেকেই যেধরণের প্রশ্ন করে সেগুলোর উত্তর দিতে চেষ্টা করবো এখানে এবং কিভাবে অনলাইন জাজে প্রবলেম সলভ করতে হয় সেটা দেখাবো।)

তোমার বয়স যতই হোক বা যেখানেই পড়ালেখা করো তুমি প্রোগ্রামিং কনটেস্টে অংশ নিতে পারবে, এমনকি তুমি যদি স্কুলে পড়ে তাহলেও। ইনফরমেটিক্স অলিম্পিয়াড হলো স্কুল-কলেজের স্টুডেন্টদের জন্য প্রোগ্রামিং কনটেস্ট, বাংলাদেশ জাতীয় পর্যায়ের অলিম্পিয়াডে ভালো করে তুমি অংশ নিতে পারবে অন্তর্জাতিক ইনফরমেটিক্স অলিম্পিয়াডে। তুমি যদি স্কুল-কলেজ থেকেই সিরিয়াসলি কনটেস্ট করা শুরু করো তাহলে খুবই ভালো সম্ভাবনা আছে বিশ্ববিদ্যালয় পর্যায় গিয়ে তুমি দেশের সেৱা একজন কনটেস্টেন্ট হতে পারবে, বাষিয়া-চীনের মতো যেসব দেশ কনটেস্টে সবথেকে ভালো তাৰা খুব কম বয়সে এটা শুরু করে। তুমি যদি এই মধ্যে কোনো বিশ্ববিদ্যালয়ে ১ম বা ২য় বর্ষে পড়ালেখা করো তাহলেও মোটেও দেৱী হয়ে যায়নি, তুমি বিভিন্ন বিশ্ববিদ্যালয়ের আয়োজিত ন্যশনাল লেভেলের কনটেস্টগুলোতে অংশ নিতে পারবে। তবে তোমার লক্ষ্য হবে আকশ ছোয়াৰ, তুমি হ্যাতো একসময় অংশ নিবে acm icpc ওয়ার্ল্ড ফাইনালে, প্রোগ্রামিং কনটেস্টের সবথেকে সম্মানজনক প্রতিযোগীতায়। এটার জন্য তোমাকে খুবই ভালো ফলাফল করতে হবে রিজিয়নাল কনটেস্ট। প্রতি বছৰই বাংলাদেশ থেকে ১টি বা ২টি টিম ওয়ার্ল্ড ফাইনালে অংশ নেয়।



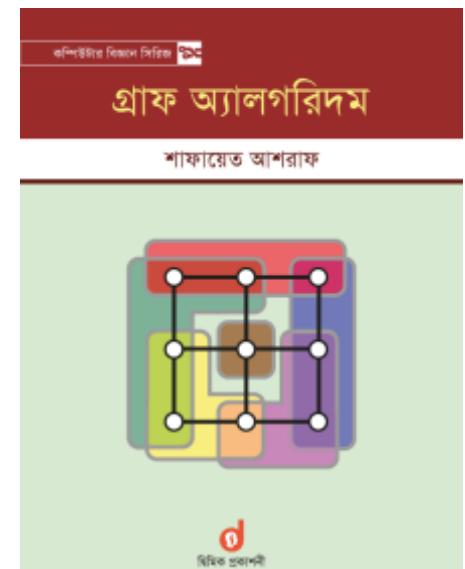
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



এগুলো ছাড়াও বিভিন্ন ওয়েবসাইটে নিয়মিত অনলাইন কনটেক্ট হয়, সেগুলোতে অংশ নিতে পারে যে কেও, শুধু তোমাকে ইমেইল আইডি দিয়ে সাইটে রেজিস্টার করতে হবে। সেখানে তুমি সাবা বিশ্বের প্রোগ্রামারদের সাথে প্রতিযোগীতা করবে, তোমাকে পারফরমেন্স অনুযায়ী গ্র্যান্ডমাস্টার/এক্সপার্ট ইত্যাদি বেটিং দেয়া হবে, সেখানে দেখতে পারবে বাংলাদেশি প্রোগ্রামাররা ইন্টারন্যুশনাল পর্যায়েও খুবই ভালো ফলাফল করে। প্রোগ্রামিং কনটেক্ট টপ-বেটেড কোডারদের অনেকের বয়স ১৫-১৬ বা তারও কম তাই তোমার ইউনিভার্সিটিতে যাওয়া পর্যন্ত অপেক্ষা করা দরকার নেই।

এখন প্রথম প্রশ্ন হলো তুমি শুরু করবে কিভাবে? প্রথম কাজ অবশ্যই প্রোগ্রামিং শেখ। তুমি যদি এখনো প্রোগ্রামিং না শিখে থাকো তাহলে আগে সেটা শুরু করে দাও। সি++ এর একটা ভালো বই যোগাড় করো আর বেসিক জিনিসগুলো শিখে ফেলো। প্রথমেই খুব বেশি শেখা দরকার নেই, বেসিক লজিক-লুপ চালানো-ইনপুট/আউটপুট নেয়া শিখেই তুমি প্রবলেম সলভিং প্র্যাকটিস করতে পারবে, একই সাথে শিখতে থাকবে আরো অ্যাডভান্সড টপিকগুলো। ল্যাংগুয়েজ শেখা আসলে কনটেক্টের প্রস্তুতির বড় অংশ না, এটা শিখতে বেশি সময় লাগবেনা, এরপরে অনেক বেশি সময় দিতে হবে প্রবলেম সলভিং টেকনিকগুলো শিখতে তবে সেই সময়টা তোমার একদমই বোরিং লাগবেনা সেটার নিশ্চয়তা দিতে পারি, এই শেখা চলতে থাকবে তুমি যতদিন কনটেক্ট করবে ততদিন, শেখার কোনো শেষ নেই।

প্রোগ্রামিং ল্যাংগুয়েজ ছাড়াও কি কি বিষয় শেখা লাগবে? তুমি প্রোগ্রামিং কনটেক্টে অংশ নেয়ার প্রস্তুতি নিছিঃ, তুমি কোনো ওয়েব ডেভেলপার না শুধু ল্যাংগুয়েজ শিখে গুগলে সার্চ করে ফাঁশন কপি-পেস্ট করে কাজ শুরু করবে, বা তুমি পরীক্ষার প্রস্তুতি নিছনা যে কোনো একটা বই মুখ্য করে ফেলবে। তুমি প্রোগ্রামিং কনটেক্ট করবে তাই তোমার জানার কোনো লিমিট থাকবেনা। তোমাকে খুবই ভালো লজিক ডেভেলপ করতে হবে, তোমাকে গণিতে বিশেষ করে কম্পিউটেরিক্স, প্রোগ্রামিং আর নাম্বার থিওরিতে অবশ্যই ভালো হতে হবে, অনেক অ্যালগরিদম আর ডাটা স্ট্রাকচার কিভাবে কাজ করে তোমাকে জানতে হবে এবং সেগুলো কোডে ইমপ্রিমেন্ট করতে জানতে হবে। কর্তৃপক্ষ কর্তৃপক্ষ শব্দগুলো শুনে ভয়ের কিছু নেই, কনটেক্ট করতে করতেই তুমি এগুলো শিখে যাবে, আগেই সবকিছু শিখে কেও শুরু করেনা। তোমাকে এই লেখার শেষ অংশ কিছু রিসোর্স দিবো শুরু করার জন্য।

এখন আসি যেটা শিখতে মূলত লেখাটা শুরু করেছিলাম সেই অংশে, অনলাইন জাজে হাতেখড়ি। তুমি যখন একটা প্রবলেম সলভ করবে সেটা সঠিক নাকি সেটা বলে দিবে অনলাইন জাজ। অনলাইন জাজ আসলে এক ধরণের অনলাইন সফটওয়্যার যে তোমার কোড পরীক্ষা করে কয়েক সেকেন্ডের মাঝে বলে দেয় সেটা সঠিক নাকি। প্র্যাকটিস করার জন্য অসংখ্য অনলাইন জাজ আছে। তুমি যত্নে জানো এইসব জাজের মধ্যে একটি আছে একজন বাংলাদেশির তৈরি করা, জাজটির নাম [lightoj](#), তৈরি করেছেন ঢাকা বিশ্ববিদ্যালয়ের জানে আলম জান যিনি বর্তমানে



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রাম

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইড্রি অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

গুগলে কাজ করছেন। তবে আমরা শুরু করবো বাংলাদেশিদের মধ্যে সবথেকে পপুলার UVa online judge দিয়ে, এটা তৈরি করেছে স্পেনের ভ্যালাডেলিড ইউনিভার্সিটি, লিংকটা হলো <http://uva.onlinejudge.org/>। তুমি শুরুতেই এখানে একটা অ্যাকাউন্ট খুলে ফেলো।

The screenshot shows a Mozilla Firefox browser window with the UVa Online Judge website loaded. The URL in the address bar is http://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=8&category=12. The page displays a list of contest volumes under the heading "Contest Volumes". The list includes Volume C, Volume CI, Volume CII, Volume CIII, Volume CIV, Volume CV, Volume CVI, Volume CVII, Volume CVIII, Volume CIX, Volume CX, Volume CXI, Volume CXII, Volume CXIII, Volume CXIV, Volume CXV, and Volume CXVI. The "Volume C" link is highlighted in red, indicating it is the current selection. On the left side of the page, there is a sidebar with links for "Main menu", "Home", "My Account", "Contact Us", "TOOLS on the Old UVa OJ Site", "ACM-ICPC Live Archive", "Logout", "Online Judge", "Quick Submit", "Migrate submissions", "My Submissions", "My Statistics", "My uHunt with Virtual Contest Service", "Browse Problems", "Problemsetters' Credits", and "Live Rankings". The status bar at the bottom of the browser shows the URL again and the date and time as "Mon Jan 7, 1:42:38 PM".

অ্যাকাউন্ট খুলে তুমি বামে browse problems থেকে “Contest Volumes” লিংকটাতে যাও। সেখানে দেখবে উপরের মতো volume C, volume CI ইত্যাদি লেখা আছে। তুমি volume C তে ক্লিক করো। সেখানে দেখবে প্রবলেমের একটা লিস্ট এসে গিয়েছে।

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

Title	Total Submissions / Solving %	Total Users / Solving %
✓ 10000 - Longest Paths	23702 98.17%	5626 58.28%
✓ 10001 - Garden of Eden	4092 93.88%	1275 67.60%
✓ 10002 - Center of Masses	8570 98.48%	1815 64.85%
✓ 10003 - Cutting Sticks	20999 42.88%	6143 87.11%
✓ 10004 - Bicoloring	34279 42.94%	9604 87.29%
✓ 10005 - Packing polygons	6992 15.99%	1679 45.44%
✓ 10006 - Carmichael Numbers	25252 34.70%	6503 82.79%
✓ 10007 - Count the Trees	5977 35.40%	2302 69.68%
✓ 10008 - What's Cryptanalysis?	21150 56.61%	9799 93.33%
✓ 10009 - All Roads Lead Where?	9192 93.86%	2507 73.75%
✓ 10010 - Where's Waldorf?	20528 38.37%	5674 75.29%
✓ 10011 - Where Can You Hide?	3870 37.66%	519 35.36%
✓ 10012 - How Big Is It?	6703 37.12%	1609 64.57%
✓ 10013 - Super long sums	39999 98.13%	7498 62.00%
✓ 10014 - Simple calculations	7715 26.07%	2686 65.38%
✓ 10015 - Joseph's Cousin	6211 52.89%	2618 41.89%
✓ 10016 - Flip-Flop the Squarelotron	2221 45.79%	899 73.75%
✓ 10017 - The Never Ending Towers of Hanoi	4297 18.91%	1334 45.70%
✓ 10018 - Reverse and Add	60970 41.07%	14410 85.18%

ডানের কলামগুলোতে দেয়া আছে ক্ষেত্রে সমস্যাটা সলভ করেছে আর ক্ষেত্রে সেটা ক্ষেত্রে সমস্যাটা সলভ করেছে। যে প্রবলেম যত বেশি মানুষ সলভ করেছে সেটা তত সহজ হ্বার সম্ভাবনা বেশি। তুমি নিচে স্ক্রল করে 10055 – Hashmat the Brave Warrior প্রবলেমটায় ক্লিক করো। প্রবলেমটি সেটা করেছেন বাংলাদেশের Shahriar Manzoor, তিনি খুবই বিখ্যাত প্রবলেমসেটার এবং পুরো পৃথিবীতে হাতেগোণা অন্ন যে ক্ষেত্রে মানুষ তিনি ওয়ার্ল্ড ফাইনালের জাজ হতে পেরেছেন তাদের একজন। প্রবলেমটা একবার তুমি রিডিং পোর্টে তারপর আমি ব্যাখ্যা করছি।

যেকোনো প্রবলেমকে ঢটা অংশ ভাগ করা যায়। প্রথম ভাগে থাকে প্রবলেমের বর্ণনা। এরপরে থাকে “Input” অংশ। এই অংশ বলা থাকে তোমাকে কি ধরণের ডায়ালুর জন্য প্রবলেমটা সলভ করতে হবে। “Output” অংশে থাকে কিভাবে আউটপুট প্রিণ্ট করতে থাকে। কয়েকটি sample input/output দেয়া হয় বোঝার সুবিধার জন্য। কিন্তু এটা কিন্তু জাজ এর Input না। জাজের একটি ইনপুট ফাইল থাকে, hashmat প্রবলেমটার জন্য সেটা হতে পারে এরকম:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্রিমাম ফ্রা-১

ম্যাক্রিমাম ফ্রা-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এবং ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

The screenshot shows a text editor window with the title bar "input.in". The content of the file is a list of 26 pairs of integers, each pair on a new line. The pairs are: 1 86 83, 2 15 77, 3 35 93, 4 92 86, 5 21 49, 6 27 62, 7 59 90, 8 26 63, 9 26 40, 10 36 72, 11 68 11, 12 29 67, 13 30 82, 14 23 62, 15 35 67, 16 2 29, 17 58 22, 18 67 69, 19 56 93, 20 42 11, 21 73 29, 22 19 21, 23 37 84, 24 24 98, 25 70 15, 26 26 13.

তবে সত্যিকারের জাজ ইনপুট ফাইলে কি আছে সেটা খালি প্রবলেম সেটারো জানেন, তোমাকে বা আমাকে সেটা দেয়া হবেনা, তোমাকে খালি বলা হবে ইনপুট ফাইলে কি ধরণের ইনপুট আছে, ইনপুট ফাইল কত বড়, যে সংখ্যা বা স্ট্রিং ইনপুট দেয়া হবে সেগুলো কত বড় ইত্যাদি। প্রতিটি ইনপুটের জন্য তোমাকে আউটপুট প্রিন্ট করতে হবে প্রবলেমে বলা ইনস্ট্রাকশন অনুযায়ী। জাজের কাছে একটা answer ফাইল আছে। তুমি যেসব আউটপুট প্রিন্ট করবে জাজ সেটাকে আরেকটি আউটপুট ফাইল নিয়ে যাবে। তারপরে answer ফাইলের এর সাথে তোমার কোডথেকে পাওয়া আউটপুট মিলিয়ে দেখবে। এই কারণে তুমি আউটপুটে অতিরিক্ত কোনো কিছু প্রিন্ট করতে পারবেনা, যেমন hashmat প্রবলেমে তুমি প্রতি লাইনে সুন্দর করে “the answer is 10” এভাবে প্রিন্ট করলে হবেনা, যেভাবে বলেছে সেভাবে শুধুমাত্র উত্তরটা প্রিন্ট করতে হবে। তোমাকে ফাইল নিয়ে চিন্তা করতে হবেনা, তুমি সাধারণভাবেই printf,scanf দিয়ে কাজ করবে, ফাইলের ব্যাপার জাজ হ্যান্ডেল করবে।

এখন তুমি একটা সলিউশন লিখে ফেলবে। সলিউশনটা হতে পারে এরকম:

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্স(১):

ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

The screenshot shows a C++ IDE interface with a menu bar (File, Edit, Search, View, Document, Project, Build, Tools, Help) and a toolbar with various icons. A project tree on the left shows a folder named 'solution'. The main editor window contains the following C++ code:

```
1 #include<cstdio>
2 using namespace std;
3 int main()
4 {
5     long long a,b;
6     while(scanf("%lld%lld",&a,&b)==2)
7     {
8         long long ans=a-b;
9         if(ans<0) ans=ans*-1;
10        printf("%lld\n",ans);
11    }
12    return 0;
13 }
14 }
```

আমি সরাসরি কোড না দিয়ে ইমেজ দিলাম যাতে কেও কপি-পেস্ট করে প্রথম সলিউশন লেখার আনন্দ মিস না করে। while এর ভিতরে scanf এর একক ব্যবহার হয়তো তুমি আগে দেখেনি, আপাতত জেনে রাখো এভাবে লিখলে যতক্ষণ ২টি সংখ্যা ইনপুট দেয়া হবে ততক্ষণ তোমার কোড কাজ করবে, ফাইল যখন শেষ হবে(EOF=end of file) তখন কোড টার্মিনেট করবে। এই কোডটা লিখে স্যাম্পল ইনপুট গুলো কিবোর্ড দিয়ে ইনপুট দিয়ে দেখো যে আউটপুটের সাথে মিলছে নাকি। মনে রাখবে শুধুমাত্র আউটপুট অংশটা ভিন্ন একটা ফাইলে নিয়ে জাজের সাথে মিলিয়ে দেখা হবে।

এবার বামে quick submit এ লিঙ্ক করো। প্রবলেম আইডি হলো 10055 যেটা প্রবলেমের উপরেই লেখা আছে। ল্যাংগুয়েজ সিলেক্ট করো C++ 4.5.3।

কোয়ান্টাম কম্পিউটার কী?
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet
2.4K likes
ফাফ অ্যালগরিদম
শাফায়েত আশরাফ

Like Page

Be the first of your friends to like this

Shafaetsplanet 55 minutes ago
ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্বিনেটরিক্যাল প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

*i, W) = 1 if W =
i + 1, W - C[i]) -*

SHAFAEETPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / মার

Problem ID	10055
Language	<input type="checkbox"/> ANSI C 4.5.3 - GNU C Compiler with options: -lm -lcrypt -O2 -pipe -ansi <input type="checkbox"/> DONLINE_JUDGE <input type="checkbox"/> JAVA 1.6.0 - Java Sun JDK <input checked="" type="checkbox"/> C++ 4.5.3 - GNU C++ Compiler with options: -lm -lcrypt -O2 -pipe -DONLINE_JUDGE <input type="checkbox"/> PASCAL 2.4.0 - Free Pascal Compiler
Paste your code...	<pre>#include<iostream> using namespace std; int main() { long long a,b; while(scanf("%lld%lld",&a,&b)==2) { long long ans=a-b; if(ans<0) ans=ans*-1; printf("%lld\n",ans); } return 0; }</pre>
test	

এরপর সাবমিট বাটনে ক্লিক করো। জাজের সার্ভার ঠিকঠাক থাকলে কয়েক সেকেন্ডে জাজ হয়ে যাবে। বামে my submissions এ ক্লিক করো এখন। জাজ হয়ে গেলে দেখাবে “Accepted”। তারমানে তোমার কোড সঠিক, এইমাত্র তুমি অনলাইন জাজে তোমার প্রথম প্রবলেম সলভ করেছো!! UVa'তে সার্ভারের অনেক সময় সমস্যা করলে verdict আসতে দেরি করে, তখন একটু অপেক্ষা করতে হবে, তবে সেটা খুব কমই হয়। তুমি কোড সাবমিট করার পর যেসব verdict পেতে পাবো সেগুলো হলো:

Wrong answer: তোমার সলিউশনে ভুল আছে। sample input এর জন্য তোমার কোড সঠিক উত্তর দিলেও জাজের মূল ইনপুট ফাইলের জন্য দিতে পারছেন। যেমন ধরো hashmat প্রবলেমে বলা আছে $2^{32}-1$ পর্যন্ত ইনপুট দেয়া হবে, সাধারণ integer এর লিমিট হলো $2^{31}-1$, তাই long long ব্যবহার না করলে তুমি wrong answer পাবে। wrong answer আসলে চিন্তা করো কোন ইনপুটের জন্য তোমার প্রোগ্রাম কাজ করছেন।

Time limit exceeded: প্রবলেমের উপর হ্যাতো লেখা দেখেছো time limit: 3 seconds, তারমানে সবগুলো ইনপুটের জন্য মোট 3 সেকেন্ডের মাঝে তোমাকে আউটপুট দিতে হবে, যদি টাইম পার হয়ে যায় তাহলে এই verdict পাবে। তখন চিন্তা করো কিভাবে কোড অপটিমাইজ

করা যায়। আমরা সাধারণত ধরে নেই ১ সেকেন্ডে মোটামুটি ১০৮৮ পর্যন্ত লুপ চালানো যাবে।

Run time error: তোমার কোড এক্সিকিউশন করার সময় কোনো একটা কারণে বন্ধ হয়ে গিয়েছে। যেমন হ্যাতো কোথাও তুমি শুণ্য দিয়ে কাওকে ভাগ করার চেষ্টা করছো বা তোমার অ্যারের সাইজ বেশি ছোটো।

Presentation error: তুমি যদি কোডে অতিরিক্ত space প্রিন্ট করো তাহলে এই verdict পাবে। তারমানে তোমার সলিউশন ঠিকই আছে তবে প্রিন্টিং ফরমেট ঠিক নাই। সাধারণত খুব সহজেই এই ইরোৱ দূর করা সম্ভব।

Accepted: তোমার সলিউশন সঠিক, অভিনন্দন, এখন অন্য আরেকটি প্রবলেম সলভ করো।

এখন তুমি বামে My uHunt with Virtual Contest Service লিংকে ক্লিক করো। সিঙ্গাপুর ন্যাশনাল ইউনিভার্সিটির ফেলিক্স হালিম এই সাইটটা তৈরি করেছেন। এখানে খুব সহজেই তোমার সব statistics দেখতে পাবে, সাবমিট করার পর চাইলে এখানেই তুমি verdict দেখতে পারো। নিচের দিকে “Next Problems to Solve” অংশ থেকে তুমি সলভ করার মতো প্রবলেম খুজে নিতে পারবে, প্রথম দিকে সহজ যত প্রবলেম আছে সব সলভ করে ফেলবে, এরপর দক্ষতার বাড়াতে কঠিন প্রবলেম সলভ করা শুরু করবে। “World Ranklist” অংশে দেখতে পারবে তোমার অবস্থান কোথায়। উপরের দিকে search problems ঘরে প্রবলেম id বসিয়ে খুব সহজে যেকোনো প্রবলেমের লিংক খুজে বের করতে পারবে। এছাড়া এই সাইটে আরো কিছু কাজের টুলস আছে, একটু গুতাগুতি করে বুঝে নিবে।

প্রথম দিকে তোমার input/output ফরমেট নিয়ে একটু সমস্যা হতে পারে। eof পর্যন্ত ইনপুট নেয়া, blank line পেলে ব্রেক করা এসব ব্যাপার নতুন লাগতে পারে। এজন্য তুমি এই ফাইলটা ডাউনলোড করে নাও:

“ ইনপুট/আউটপুট টেকনিক+অ্যারে+স্ট্রিং+ডাটাটাইপ+পয়েন্টার

এখানে কয়েকটা ছোটোছোটো pdf ফাইল আছে, প্রবলেম সলভিং শুরু করার জন্য খুবই কাজের এগুলো।

যে বইগুলো তোমার পড়া খুবই দরকার সেগুলো হলো knuth এর concrete math, rosen এর discrete math, কোরম্যান বা শাহনীর algorithm, স্টিভেন-ফেলিক্স হালিমের competitive programming।

প্রবলেম সলভ করতে গিয়ে অনেক সময়ই তুমি আটকে যাবে, বা রবার wrong answer থাবে। তখন তুমি কি করবে? প্রথম কাজ হলো আরো ভালোভাবে চিন্তা করো। এবপরেও না পারলে সাহায্য নাও। হতে পারে তুমি এমন জায়গায় থাকো যেখানে সাহায্য করার মতো শিক্ষক বা বড় ভাই নাই। ইন্টারনেটের যুগে এটা কোনো ব্যপারই না, তুমি বিডিই ফোরামের সাহায্য নাও। তুমি জান ভাইয়ের [lightoj](#) তে বেজিস্টার করলে একটি চমৎকার ফোরাম পাবে যেখানে অনেকেই সাহায্য করবে। uva সহ প্রায় সব অনলাইন জাজেরই ফোরাম আছে, তুমি গুগলে সার্চ করলেই লিংক পাবে সেগুলোর। সেখানে তুমি বড় বড় প্রোগ্রামারদের সাথে কথা বলো, দেখো তারা কিভাবে প্র্যাকটিস করে। বাংলাদেশের প্রোগ্রামারদের সাথে যোগাযোগ করার জন্য চমৎকার জায়গা হতে পারে ফেসবুক, তুমি নিচের ছফ্প গুলোতে জয়েন করো:

[প্রোগ্রাম্যা-বাংলা ইনফরমেটিক্স ব্লগ](#)

[BD programmers](#)

[Bangladesh Informatics Olympiad](#)

এসব জায়গায় তুমি সাহায্য করার মতো অনেক উদার মানুষকে পাবে। [ফাহিম ভাইয়ের সাইট](#) তুমি দারুণ সব রিসোর্স পাবে, বিডিই অনলাইন জাজের কোনটার কি ফিচার আছে সেগুলো জানতে পারবে। তুমি uva'র সহজ প্রবলেমের একটা তালিকা পাবে [এখানে](#), এগুলো দিয়ে সলভ শুরু করতে পারো।

বুঝতেই পারছো তুমি যেখানেই থাকোনা কেনো প্রোগ্রামিং কনটেস্ট ভালো করার মতো সব রিসোর্স ইন্টারনেটেই আছে, তোমার কাজ হলো সেগুলো ব্যবহার করে খুব ভালো করে প্র্যাকটিস করা। অনেকবার তোমার মনে হবে “আমি পারছিনা”, এরকম ভেবে অনেকেই হাল ছেড়ে দেয়, তুমি ছাড়বেনা, একসময় তুমি ভালো করতে বাধ্য। নেটে তুমি অনেক কোড পাবে যেগুলো সাবমিট করে তুমি accepted পেতে পারো কিন্তু ভুলেও সেই কাজ করবেনা, তাহলে বিশাল ক্ষতি হবে, তোমাকে এ ব্যাপারে নিজের কাছে সৎ থাকতে হবে। কিছুটা দক্ষতা আসার পর বেশি প্রবলেম সলভ করার থেকে বেশি গুরুত্ব দিবে ভালো প্রবলেম সলভ করাকে, অনলাইন জাজে তুমি কয়টা প্রবলেম সলভ করেছো সেটা কেও দেখবেনা, দেখবে তুমি বিডিই কনটেস্ট কেমন করো সেটা।

সবশেষে বলতে চাই তোমার চলার পথ হয়তো খুব সহজ হবেনা, অনেকেই নিরুৎসাহিত করবে, অনেক শিক্ষক পছন্দ করবেনা তোমার কনটেস্ট করা, অনেক সময় অন্য কাজের সাথে তাল মিলিয়ে কনটেস্ট করা কঠিন হবে, এগুলোই বাস্তবতা বাংলাদেশ। এসবের মধ্যে কেও কেও হাল ছেড়ে দিবে, বাকিরা কোনো কিছুতেই হাল ছাড়বেনা, জয় একমাত্র তাদেরই প্রাপ্তি, তোমাকে বেছে নিতে হবে তুমি কোন দলে থাকবে। আমি কনটেস্ট শুরু করি ২০১০ এ, বেশ কিছু ন্যাশনাল আর রিজিয়নাল কনটেস্ট করেছি, কোনোভাবেই আমাকে সিনিয়র কনটেস্টেন্ট বা খুব ভালো কনটেস্টেন্ট বলা

অ্যালগরিদম কমপ্লেক্সিটি(বিগ “O” নোটেশন)

অক্টোবর ১২, ২০১২ by শাফায়েত



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

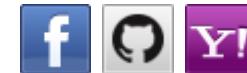
- “ ১. আমার অ্যালগরিদম কি নির্ধারিত সময়ের মধ্যে ফলাফল এনে দিবে?
- ২. সর্বোচ্চ কত বড় ইনপুটের জন্য আমার অ্যালগরিদম কাজ করবে?
- ৩. আমার অ্যালগরিদম কতখানি মেমরি ব্যবহার করছে?

আমরা অ্যালগরিদমের কমপ্লেক্সিটি বের করে প্রথম দুটি প্রশ্নের উত্তর দিতে পারি। একটি অ্যালগরিদম যতগুলো “ইনস্ট্রুকশন” ব্যবহার করে কাজ করে সেটাই সোজা কথাই সেই অ্যালগরিদমের কমপ্লেক্সিটি। দুটি নম্বর গুণ করা একটি ইনস্ট্রুকশন, আবার একটি লুপ ১০০ বার চললে সেখানে আছে ১০০টি ইনস্ট্রুকশন। 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 অ্যালগরিদমের 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:

```
1 int myAlgorithm1(int n)
2 {
```

সাবস্ক্রাইব

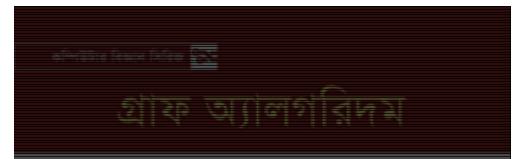
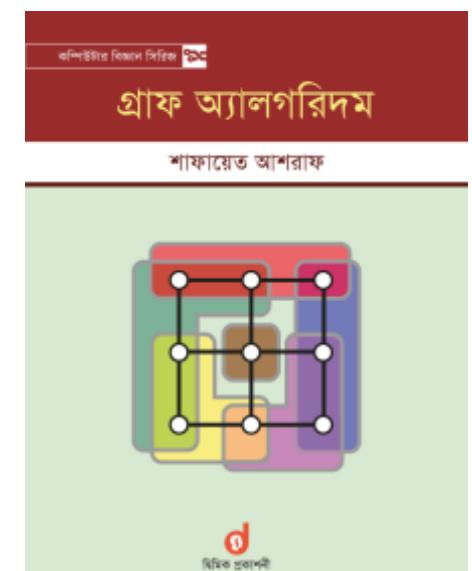


Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

Traveloka Singapore (বিস্তারিত...)



```

3 int x=n+10;
4 x=x/2;
5 return x;
6 }

```

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 হ্যানা।

“ 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:

```

1 int myAlgorithm2(int n)
2 {
3     int sum=0;
4     for(int i=1;i<=n;i++)
5     {
6         sum+=i;
7         if(sum>=1000) break;
8     }
9     return sum;
10 }

```

এই কোডে একটি লুপ চলছে এবং সেটা n এর উপর নির্ভরশীল। $n = 100$ হলে লুপ 100 বার চলবে। লুপের ভিতরে বা বাইরে ক্যাটি ইনস্ট্রাকশন আছে সেটা নিয়ে চিন্তা করবোনা, কারণ সেটার সংখ্যা খুবই কম। উপরের অ্যালগোরিদমের কমপ্লেক্সিটি $O(n)$ কারণ এখনে লুপটি n বার চলবে। তুমি বলতে পারো sum যদি 1000 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 পর্যন্ত চলবে।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে
হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগোরিদম

ডাটা স্টোরেজ(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

worst case এ যতগুলো ইনস্ট্রাকশন থাকবে সেটাই আমাদের কমপ্লেক্সিটি।

“

```
1 int myAlgorithm3(int n)
2 {
3     int sum=0;
4     for(int i=1;i<=n;i++)
5     {
6         for(int j=i;j<=n;j++)
7         {
8             sum+=(i+j);
9         }
10    }
11    return sum;
12 }
```

উপরের কোডে ডিতরের লুপটা প্রথমবার n বার চলছে, পরেরবার $n - 1$ বার। তাহলে মোট লুপ চলছে

$$n + (n - 1) + (n - 3) + \dots + 1 = n \times (n + 1)/2 = (n^2 + n)/2 \text{ বার।}$$

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 ওটি অ্যালগোরিদমের মধ্যে সবথেকে সময় কম লাগবে কোনটি? 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:

```
1 int myAlgorithm4(int n,int *val,int key)
2 {
3     int low=1,high=n;
4     while(low<=high)
5     {
6         int mid=(low+high)/2;
7         if(key<val[mid]) low=mid-1;
8         if(key>val[mid]) high=mid+1;
9         if(key==val[mid]) return 1;
10    }
}
```

শ্লাইডিং রেজ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্লেক্সিটি/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```
11 return 0;  
12 }
```

এটা একটা বাইনারি সার্চের কোড। প্রতিবার $low + high = n + 1$ বা n এর মান দুই ভাগে ভাগ হয়ে যাচ্ছে। একটি সংখ্যাকে সর্বোচ্চ কর্তব্য দিয়ে ভাগ করা যায়? একটি হিসাব করলেই বের করতে পারবে সর্বোচ্চ ভাগ করা যাবে $\log_2 n$ বার। তারমানে $\log_2 n$ ধাপের পর লুপটি বের করবে। তাহলে কমপ্লেক্সিটি হবে $O(\log_2 n)$ ।

এখন ধরো একটি অ্যালগরিদমে কয়েকটি লুপ আছে, একটি n^4 লুপ আছে, একটি n^2 লুপ আছে আর একটি $\log n$ লুপ আছে। তাহলে মোট ইনস্ট্রুকশন: $n^4 + n^3 + \log n$ 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:

```
1 int myAlgorithm5(int n)  
2 {  
3     if(n==1) return 1;  
4     return n*myAlgorithm5(n-1);  
5 }
```

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

$$f(n) = \text{ইনস্ট্রুকশন সংখ্যা}$$

$$f(n) = n^2 + 3n + 112 \text{ হলে কমপ্লেক্সিটি } O(n^2) /$$

$$f(n) = n^3 + 999n + 112 \text{ হলে কমপ্লেক্সিটি } O(n^3) /$$

$$f(n) = 6 \times \log(n) + n \times \log n \text{ হলে কমপ্লেক্সিটি } O(n \times \log n) /$$

$$f(n) = 2^n + n2 + 100 \text{ হলে কমপ্লেক্সিটি } O(2^n) / (\text{এটাকে exponential কমপ্লেক্সিটি বলে})$$

বিগনারদের আরেকটি কমন ডুল হলো এভাবে কোড লেখা:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্রিমাম ফ্রো-১

ম্যাক্রিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

```

1 int myAlgorithm6(char *s)
2 {
3     int c=0;
4     for(int i=0;i<strlen(s);i++)
5     {
6         if(s[i]=='a') c++;
7     }
8     return c;
9 }
```

s স্ট্রিং এর দৈর্ঘ্য $|s|$ হলে এখানে কমপ্লেক্সিটি হলো $O(|s|^2)$ । কেন স্থায়ার হলো? কারণ $\text{strlen}(s)$ ফাংশনের নিজের কমপ্লেক্সিটি হলো $O(|s|)$, একে লুপের মধ্যে আরো $O(|s|)$ বার কল করা হয়েছে। তাই $\text{strlen}(s)$ এর মান আগে অন্য একটি ড্যারিয়েবলের 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 ইনস্ট্রুকশন রান করতে পারবে। এটা জাজ-পিসি অনুসারে কমবেশি হতে পারে, যেমন টপকোডার আরো অনেক বেশি ইনস্ট্রুকশন ১ সেকেন্ডে রান করতে পারে কিন্তু spoj বা কোডশেফ তাদের পেন্টিয়াম ৩ পিসি দিয়ে 10^7 টাও সহজে রান করতে পারেনা। অনসাইট ন্যশনাল 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:

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

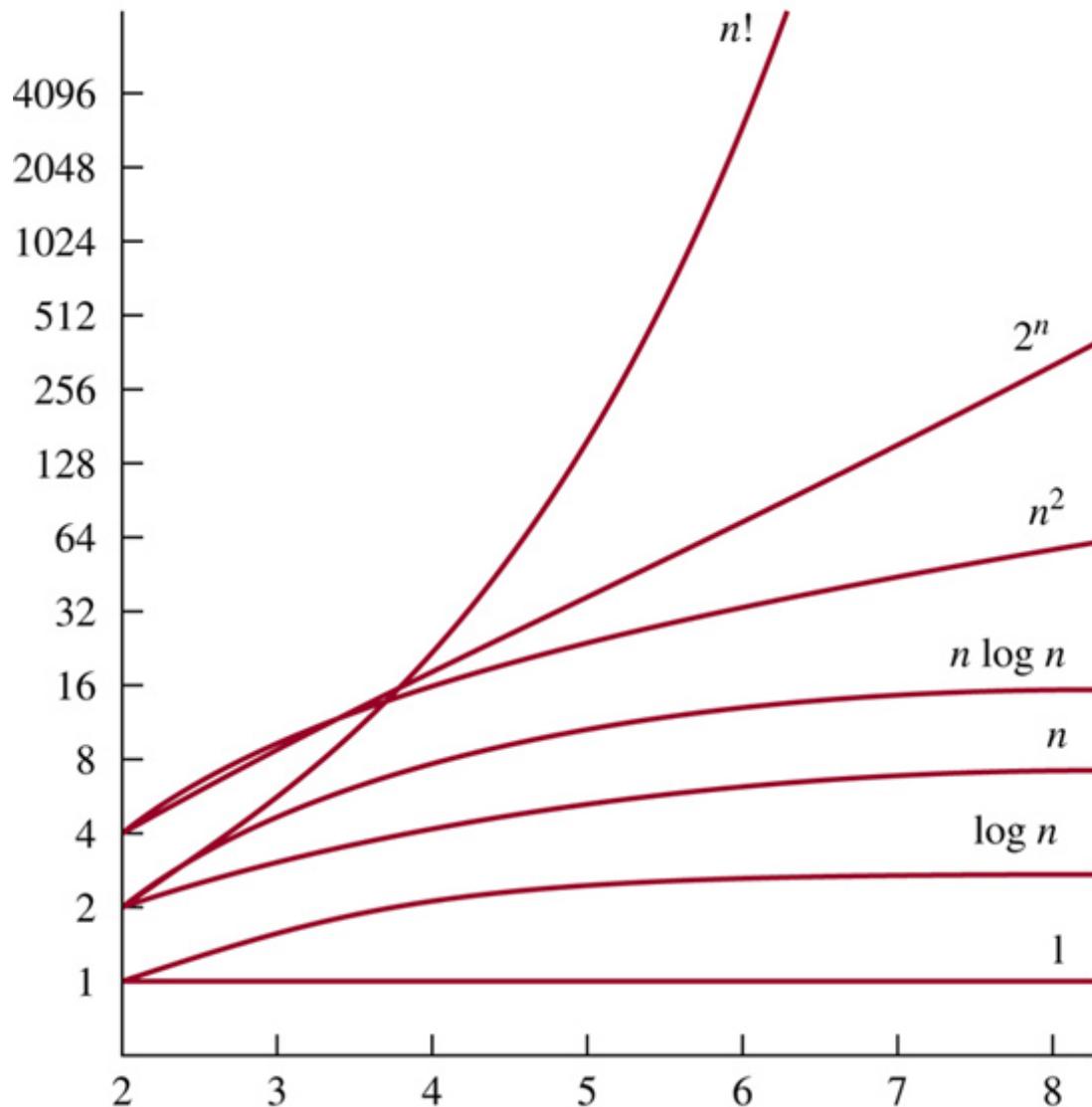
অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



(x axis=input size, y axis=number of instructions)

কনটেন্টে 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 ম্যাক্সিমাম-ম্যাচিং প্রবলেম। $n=10^5$ হলে সাধারণ $n \log n$ কমপ্লেক্সিটিতে প্রবলেম সলভ করতে হয় তাই সম্ভাবনা আছে এটা একটা বাইনারি সার্চ বা সেগেমন্ট ট্রি এর প্রবলেম।

আজ এই প্রশ্নটি নিয়ে আরো অনেক কিছু জানার আছে, বিস্তারিত পড়েও:

[A Gentle Introduction to Algorithm Complexity Analysis](#)

[Complexity and Big-O Notation](#)

কমপ্লেক্সিটি ক্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

আগস্ট ১৪, ২০১৩ by শাফায়েত

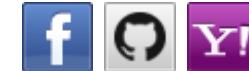


আমরা যখন প্রবলেম সলভ করতে করতে মাথার চুল ছিড়ে ফেলি তখন প্রায়ই এমন কিছু প্রবলেম সামনে আসে যেগুলো সলভ করতে গেলে সবথেকে শক্তিশালী কম্পিউটারও হাজার হাজার বছর লাগিয়ে দিবে। বড় বড় কম্পিউটার বিজ্ঞানীরা যখন দিন-রাত চিন্তা করেও এগুলো সলভ করতে পারলেননা তখন তারা এই প্রবলেমগুলোকে কিছু ক্যাটাগরিতে ফেলে দিয়ে বললেন “এই ক্যাটাগরির প্রবলেমগুলো সলভ করার সাধ্য এখনো আমদের হ্যানি, তোমরা কেও পারলে সলভ করে দিয়ে ১০ লক্ষ ডলার পুরস্কার নিয়ে যাও”। সেগুলোকেই আমরা NP-complete, NP-hard ইত্যাদি নামে চিনি।

NP ক্যাটাগরির প্রবলেমগুলো কম্পিউটার সায়েজে খুব বিখ্যাত। যেমন একটা গ্রাফে সবথেকে লম্বা পথ খুজে বের করা, ঢটা রঙ দিয়ে গ্রাফ কালারিং করা ইত্যাদি। এগুলোকে এখন পর্যন্ত কেও পলিনমিয়াল টাইমে সলভ করতে পারেনি, যেসব সলিউশন বের হয়েছে সেগুলো সামান্য বড় ইনপুটের জন্যই কাজ করেন। মজার ব্যাপার হলো এই প্রবলেমগুলোর যেকোন একটা যদি তুমি সলভ করতে পারো তাহলে সবগুলো প্রবলেমই সলভ করা যাবে তোমার সলিউশনের সাহায্যে, অনেক বছর ধরে মানুষকে ভাবানো কঠিন সব প্রবলেম চোখের নিমিষে সলভ হয়ে যাবে একটা মাত্র সলিউশন দিয়ে, কম্পিউটার বিজ্ঞানের চেহারাটাই পাণ্টে দিবে এটা। সেজন্যই এত বড় বড় পুরস্কার ঘোষণা করা হয়েছে NP প্রবলেম সলভ করা জন্য।

এই আর্টিকেলটা পড়ার আগে তোমাকে **অ্যালগোরিদম কমপ্লেক্সিটি** নিয়ে জানতে হবে। এছাড়া NP নিয়ে জানতে হলে আমদের প্রথমে কয়েকটা টপিক নিয়ে কিছুটা জানতে হবে, আগে সেটা নিয়ে আলোচনা করবো। এই লেখাটা পড়ার পর তুমি P-NP, টুরিং মেশিন নিয়ে একটা ভালো ধারণা পাবে।

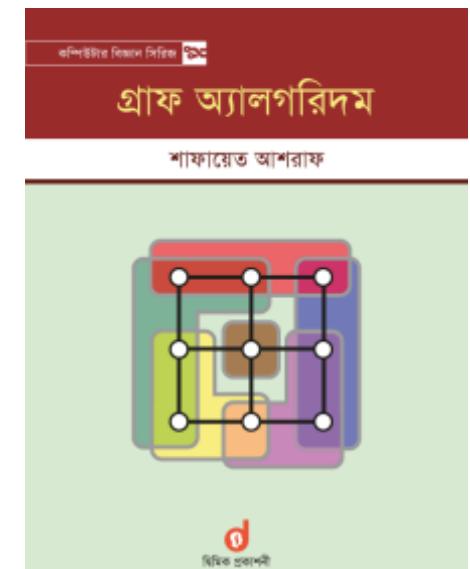
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

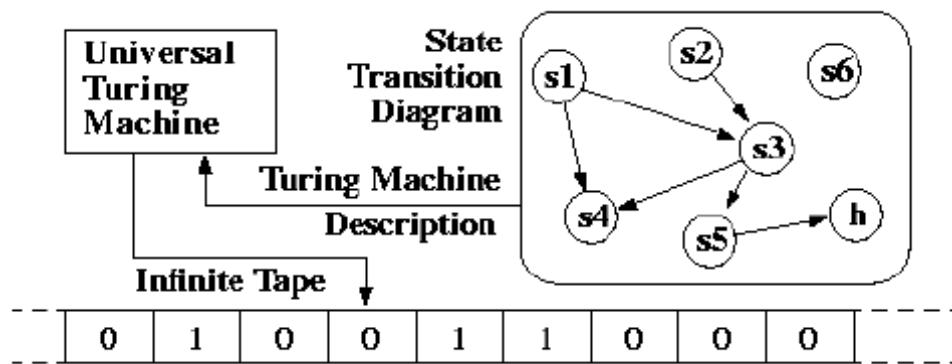
শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



টুরিং মেশিন

১৯৩৬ সালে এলান টুরিং একটা “হাইপোথেটিকাল” মেশিন ডিজাইন করেন, বাস্তবে তৈরি করেননি, শুধুই থিওরি দিয়েছেন। মেশিনটা অ্যালগোরিদম সিমুলেট করতে পারে। এই মেশিন দিয়ে ব্যাখ্যা করা যায় কিভাবে কম্পিউটার একটা প্রবলেম সলভ করতে পারে। টুরিং চিন্তা করতেন “একটা প্রবলেম কম্পিউটেবল” বলতে কি বোঝায়? একটা প্রবলেম “কম্পিউটেবল” মানে হলো “কিছু ইনস্ট্রাকশন থাকবে যেগুলো একটা কম্পিউটার ফলো করলে একটা কাজ শেষ হবে”, এই ইনস্ট্রাকশনগুলোই হলো অ্যালগোরিদম। ইনস্ট্রাকশনগুলো কম্পিউটার ফলো করতে পারবে নাকি সেটা নির্ভর করে মেশিনের সামর্থ্যের উপর। যেসব অ্যালগোরিদম টুরিং মেশিন দিয়ে সলভ করা যায় সেগুলো “টুরিং-কম্পুটেবল”।

টুরিং মেশিনে একটা অসীম লম্বা টেপ থাকে(নিচের ছবি)। একটা হেড টেপের কোনো জায়গায় পয়েন্ট করে প্রবলেমের একটা “স্টেট” নির্দেশ করে। প্রতিটা স্টেট থেকে কোন স্টেট যাওয়া যায় সেটা নির্ভর করে ওই প্রবলেমের “ট্রানজিশন ডায়াগ্রাম” এর উপর। সহজ কথায় ডায়াগ্রামে বলা থাকে কোন ইনপুটের জন্য কোন স্টেট থেকে কোন স্টেটে যেতে হবে সেটা।



আধুনিক কম্পিউটারগুলো টুরিং মেশিন কম্পিউটেবল। টুরিং মেশিন দিয়ে সলভ করা না গেলে কম্পিউটার দিয়ে সলভ করা যায়না, তাই এটা নিয়ে আমাদের এত আগ্রহ।

নন-ডিটারমিনিস্টিক টুরিং মেশিন:



প্রোগ্রামিং কনটেন্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগোরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

ধরো তোমার ইনস্ট্রাকশন সেটে বলা আছে “তুমি যদি একটা ১০ নম্বর স্টেটে থাকো এবং ইনপুটে একটা “A” পাও তাহলে বামে যাও” এবং একই সাথে বলা থাকে “তুমি যদি একটা ১০ নম্বর স্টেটে থাকো এবং ইনপুটে একটা “A” পাও তাহলে ডানে যাও” তাহলে তুমি কোনটা করবে? একই স্টেটে একই ইনপুটের জন্য ডিএ ইনস্ট্রাকশন! **নন-ডিটারমিনিস্টিক টুরিং মেশিন** এই ধরণের সিচুয়েশনে অনুমানের উপর নির্ভর করবে। সে বেন্ডমলি অনুমান করে কোনো একটা পাথে চলে যাবে। হয়তো সেই অনুমান ভুল এবং কম্পিউটেশনটা ভুল আউটকাম দেবে। কিন্তু এখানে বিবেচ্য হলো এমন কোনো সঠিক সিদ্ধান্তের সিকুয়েন্স আছে কি না। **নন-ডিটারমিনিস্টিক মেশিন** সবসময় ঠিক রেজাল্ট দিবে এমন কোনো কথা নেই, যদি রেজাল্ট পজিটিভ হয় তাহলে অবশ্যই সলিউশন আছে, কিন্তু নেগেটিভ হলেও নিশ্চিত করে বলা যায়না যে আসলে পজিটিভ রেজাল্ট আছে নাকি নেই। এজন্যই এধরণের মেশিনের নাম নন-ডিটারমিনিস্টিক।

ডিটারমিনিস্টিক টুরিং মেশিন: এই মেশিনের ইনস্ট্রাকশন সেটে এভাবে একই সাথে দুই রকম ইনস্ট্রাশন থাকবেন। প্রতি সিচুয়েশনে নির্দিষ্ট কোনদিকে যেতে হবে সেটা আগে থেকে জানা থাকবে।

আধুনিক কম্পিউটার ডিটারমিনিস্টিক টুরিং মেশিন এর সমতুল্য। নন-ডিটারমিনিস্টিক ইনস্ট্রাকশন সেট নিয়ে কম্পিউটার কাজ করেন।

পলিনমিয়াল কমপ্লেক্সিটি: যেসব প্রবলেমের কমপ্লেক্সিটি ইনপুট সাইজের পাওয়ার সেগুলো পলিনমিয়াল কমপ্লেক্সিটি, যেমন $O(n), O(n\log n), O(n^3)$ ইত্যাদি।

P(Polynomial) প্রবলেম

P বা পলিনমিয়াল ক্যাটাগরি প্রবলেমকে ডিটারমিনিস্টিক টুরিং মেশিন দিয়ে পলিনমিয়াল টাইমে সলভ করা যায়। যেমন শর্টেস্ট পাথ প্রবলেম, ন্যাপস্যাক প্রবলেম। শুধু যে সেগুলো সলভ করা যায় সেটা না, একটা “সার্টিফিকেট” দিলে সেটা পলিনমিয়াল টাইমে ডেরিফাই ও করা যায়। সার্টিফিকেট কি? ধরা যাক ইউলার পাথ প্রবলেমের কথা, একটা গ্রাফের সবগুলো এজ ভিজিট করতে হবে এবং কোন এজ দুইবার ভিজিট করা যাবেনা। এখানে সার্টিফিকেট হলো কিছু এজ এর সিকোয়েন্স (e_1, e_2, \dots, e_K)। আমাদেরকে এই সিকোয়েন্স দিলে সহজেই বলতে পারবো এটা একটা ভ্যালিড পাথ নাকি। প্রতিটা নোড ধরে আগবো, দেখবো এক নোড দুইবার ভিজিট হয়েছে নাকি এবং সবগুলো নোড ভিজিট হয়েছে নাকি। $O(E)$ তেই আমরা এটা ডেরিফাই করতে পারবো যেখানে E হলো এজ সংখ্যা। P প্রবলেম পলিনমিয়াল টাইমে সলভ করা যায়, এবং একটা সলিউশন(সার্টিফিকেট) ভ্যালিড নাকি সেটা পলিনমিয়াল টাইমে ডেরিফাই করা যায়।

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)
ডিসজয়েট সেট(ইউনিয়ন ফাইল)
ট্রাই(প্রিফিল ট্রি/বেডিল ট্রি)
সেগমেন্ট ট্রি-১
সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)
অ্যারে কমপ্রেশন/ম্যাপিং
লোয়েস্ট কমন অ্যানসেক্টর
বাইনারি ইনডেক্সড ট্রি
স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:
এল-আর-ইউ ক্যাশ(**নতুন**)
ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):
গ্রাফ থিওরিতে হাতেখড়ি
অ্যাডজেসেন্সি ম্যাট্রিক্স
অ্যাডজেসেন্সি লিস্ট
ব্রেথড ফার্স্ট সার্চ (বিএফএস)
মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)
মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)
টপোলজিকাল সেট
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল
সেট
ডায়াক্রস্টা
ফ্লয়েড ওয়ার্শল
বেলম্যান ফোর্ড

NP(Non-deterministic Polynomial) প্রবলেম

NP প্রবলেমকে ডিটারমিনিস্টিক টুরিং মেশিন দিয়ে পলিনমিয়াল টাইমে ভেরিফাই করা যায়। কিন্তু পলিনমিয়াল টাইমে সলভ করা যায় কি যায় না সেটা প্রমাণ করা সম্ভব হ্যানি। যেমন হ্যামিল্টন পাথ প্রবলেম, একটা গ্রাফের সব নোড ডিজিট করতে হবে, একটা নোড দুইবার ডিজিট করা যাবেনা। তোমাকে যদি কিছু নোডের সিকুয়েন্স দেয় তুমি সহজেই ইটলার পাথ এবং মতো করে ভেরিফাই করতে পারবে। কিন্তু একটা গ্রাফে হ্যামিল্টন পাথ আছে নাকি নাই কিভাবে বলবে? এটা বের করার জন্য পলিনমিয়াল অ্যালগোরিদম এখন পর্যন্ত আবিষ্কার হ্যানি। পলিনমিয়াল অ্যালগোরিদম না থাকলে আমরা সলভ করি এক্সপোনেন্টসিয়াল অ্যালগোরিদম দিয়ে যেগুলোর কমপ্লেক্সিটি $O(2^n), (k^n)$ এরকম। এই ধরণের কমপ্লেক্সিটি সার্জেন্ট করে তোমাকে ক্রট ফোর্সের সাহায্যে সবকরম পসিবিলিটি চেক করতে হবে, সাধারণত ব্যাকট্যাকিং দিয়ে এসব প্রবলেম সলভ করা হয়।

P=NP??

প্রতিটা P প্রবলেমেই NP প্রবলেম ক্যাটাগরিতে পরে। কারণ সব P প্রবলেমকে পলিনমিয়াল টাইমে ভেরিফাই করা যায় যেটা NP ক্যাটাগরিতে একমাত্র শর্ত। তারমানে P হলো NP এর একটা সাবসেট। কিন্তু সব NP প্রবলেমই কি P প্রবলেম? যেসব প্রবলেমের সার্টিফিকেট পলিনমিয়াল টাইমে ভেরিফাই করা যায় সেগুলোর সবগুলোকেই কি পলিনমিয়াল টাইমে সলভ করা যায়? হ্যামিল্টন পাথ, স্যাট প্রবলেম ইত্যাদির ক্ষেত্রে আমরা জানিনা প্রবলেমগুলো P তে ফেলা যায় নাকি। আমরা জানিনা $P=NP$ নাকি $P \neq NP$ । এটা কম্পিউটার সায়েন্সের সবথেকে বিখ্যাত একটি ওপেন প্রবলেম। ক্ষেত্রে ম্যাথমেটিকাল ইন্সটিউট এটাকে ৭টি মিলেনিয়াম প্রাইজ প্রবলেম এর একটি ধরে ১০ লক্ষ ডলারের প্রাইজমানি ঘোষণা করেছে! তবে কম্পিউটার বিজ্ঞানী এবং গণিতবিদরা ধারণা করে $P \neq NP$, কিন্তু শুধুমাত্র ধারণা কখনোই বিজ্ঞান নয়, আইনস্টাইন মুখে কি ধারণার কথা বললেন সেটা বিজ্ঞান নয়, তিনি পিয়ার রিভিউড জার্নালে যেটা প্রকাশ করেছেন সেটা বিজ্ঞান!

আর্টিকুলেশন পয়েন্ট এবং বিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্রিমাম ফ্লো-১

ম্যাক্রিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

টি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

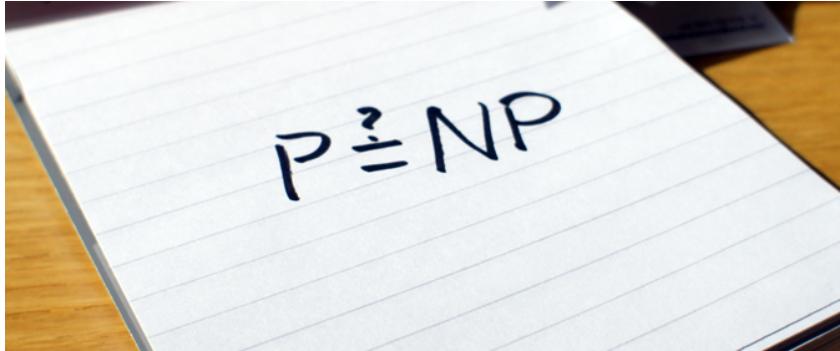
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সাবসিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)



[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

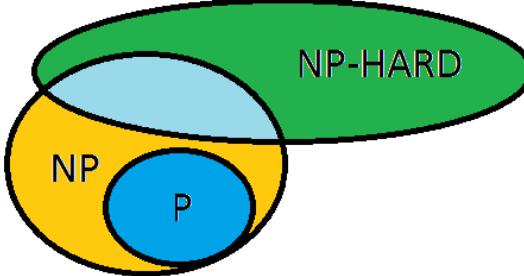
অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

চেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

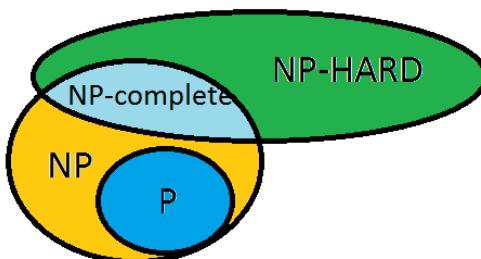
কোয়ান্টাম কম্পিউটার(২)



P!=NP ধরে নিয়ে ভ্যান ডায়াগ্রাম

যদি কখনো প্রমাণ হয় যে $P=NP$ তাহলে ছোট নীল সার্কেলটা আর দরকার হবে, হলুদ সার্কেলে $P=NP$ লিখে দেয়া যাবে। NP-hard ক্যাটাগরির যেসব প্রবলেম পলিনমিয়াল টাইমে ডেরিফাই করা যায় সেগুলো NP ক্যাটাগরিতে, বাকিগুলো বাইরে। নীল অংশের প্রবলেমগুলো দুটো ক্যাটাগরিতে কমন, সেগুলোর নাম কি?

NP complete: একটা প্রবলেম NP-complete হবে কেবল যদি প্রবলেমটা NP-hard হয় এবং সেটা NP ক্যাটাগরীতেও পড়ে। **NP complete** প্রবলেমকে পলিনমিয়াল টাইমে অন্য যেকোনো NP প্রবলেমে কনভার্ট করা যায় এবং এই প্রবলেমের সার্টিফিকেটকে পলিনমিয়াল টাইমে ডেরিফাইও করা যায়।



P!=NP ধরে নিয়ে ভ্যান ডায়াগ্রাম

কোয়ান্টাম কম্পিউটার কী?
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet
 2.4K likes
 শাফায়েত আশরাফ

Like Page
Be the first of your friends to like this

Shafaetsplanet
 58 minutes ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্বিনেটরিক্সের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

i, W) = 1 if W =
i + 1, W - C[i]) -

SHAFAEETPLANET.COM
দাইনামিক প্রোগ্রামিং ৬ / চ্যাপ

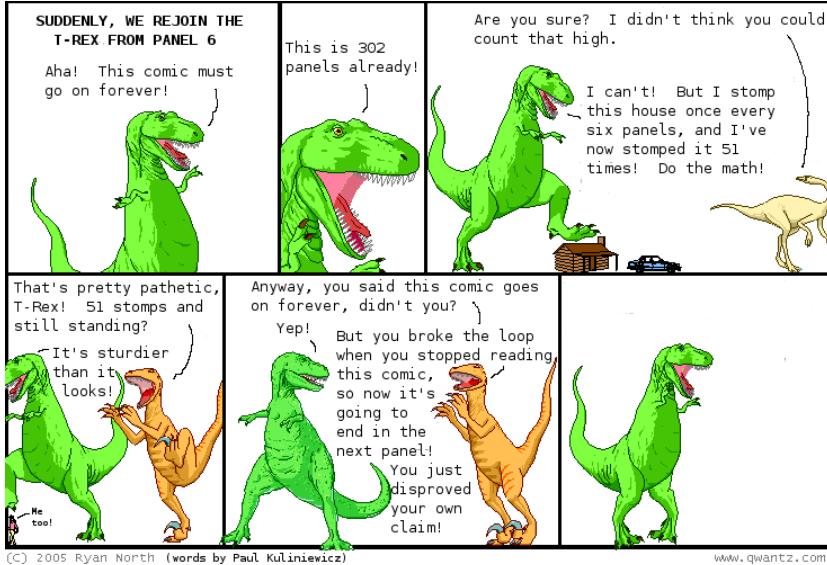
একটা প্রবলেম NP-hard প্রমাণ করতে হলে আমাদের সেটাকে পরিচিত কোনো NP-complete প্রবলেমকে সেই প্রবলেমে কনভার্ট করতে হবে। যেমন লংগেস্ট পাথ প্রবলেমে বলা হয়েছে গ্রাফে সবথেকে লম্বা পাথ বের করতে হবে যেন কোনো নোড সেই পাথে দুইবার না থাকে। হ্যামিল্টনিয়ান পাথ একটা পরিচিত NP-complete প্রবলেম, গ্রাফে যদি হ্যামিল্টনিয়ান পাথ থাকে তাহলে সবথেকে লম্বা পাথে $n - 1$ টা এজ থাকবে কারণ n টা নোড একবার করে ডিজিট করতে হবে। হ্যামিল্টন প্রবলেমকে আমরা একটু ভিন্ন ভাবে বলতে পারি “একটা গ্রাফে কি এমন কোনো পাথ থাকা সম্ভব যেটায় k টা এজ আছে?”, $k=n-1$ এর জন্য প্রবলেমটা সলভ করতে পারলে হ্যামিল্টন পাথের প্রবলেম সলভ হয়ে যাবে। এবং এই প্রবলেমটা সলভ করতে পারলে k এর বিভিন্ন মানের জন্য সলভ করে সবথেকে লম্বা পাথও বের করতে পারবো! তাহলে দেখা যাচ্ছে হ্যামিল্টন পাথ প্রবলেমকে একটু ভিন্ন ভাবে বর্ণনা করে আমরা লংগেস্ট পাথ প্রবলেমে কনভার্ট করলাম। যেহেতু হ্যামিল্টন পাথের পলিনমিয়াল সলিউশন আমরা জানিনা, লংগেস্ট পাথেরও জানিনা।

এবার হ্যতো তুমি বুঝতে পারছো একটা মাত্র প্রবলেম পলিনমিয়াল টাইমে সলভ করতে পারলে কেন সবগুলো করা যাবে।

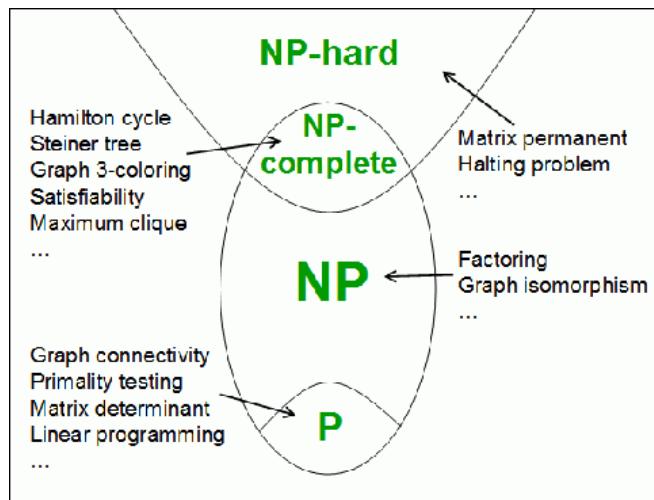
ডিসিশান প্রবলেম হলো সেগুলোকে yes/no দিয়ে উত্তর দেয়া যায়। যেমন A থেকে B তে কি 10 সেকেন্ডে পৌছানো সম্ভব? অপটিমাইজেশন প্রবলেম অ্যান্ডারকে মিনিমাইজ বা ম্যাক্সিমাইজ করে। যেমন A থেকে B তে সব থেকে কম কত সেকেন্ডে পৌছানো সম্ভব। উপরের ক্যাটাগরিগুলো প্রবলেমের ডিসিশান ভার্সন নিয়ে কাজ করে। যেকোনো অপটিমাইজেশন প্রবলেমকেই একটু ভিন্নভাবে বর্ণনা করে ডিসিশান প্রবলেম বানিয়ে ফেলা যায় যেটা আমরা উপরে হ্যামিল্টন পাথের ক্ষেত্রে করেছি।

এখন প্রশ্ন হলো প্রথম np-complete প্রবলেমটা কি? প্রথম একটা np-complete প্রবলেম না থাকলে আমরা অন্য প্রবলেমকে np-complete বা hard ক্যাটাগরিতে ফেলতে পারছিনা যেহেতু আমরা এটা করছি প্রবলেমকে কনভার্ট করে। ধরা যাক তোমার কাছে কিছু বুলিয়ান ভ্যারিয়েবল আছে x,y,z ইত্যাদি। তুমি কিছু শর্ত দিলে এরকম: “ X সত্যি হলে Y মিথ্যা”, “ Z এবং X দুইটাই মিথ্যা” ইত্যাদি। এখন তোমাকে x,y,z এ true/false ভ্যালু অ্যাসাইন করতে হবে যেন সব রিলেশন সত্যি হয়। এটাকে বলা হয় বুলিয়ান স্যাটিসফাইবিলিটি প্রবলেম বা সংক্ষেপে স্যাট প্রবলেম। রিলেশনে ২টি ভ্যারিয়েবল থাকলে ২-স্যাট প্রবলেম, ৩টি থাকলে ৩-স্যাট, যেমন “ X এবং Y মিথ্যা হলে Z সত্য” এরকম রিলেশন থাকলে সেটা ৩-স্যাট, জেনারেলাইজ করে বলা হয় k-sat। ২-স্যাটকে পলিনমিয়াল টাইমে সলভ করা যায়, $k > 2$ হলে পলিনমিয়াল সলিউশন পাওয়া যায়না। NP ক্যাটাগরির ফর্মাল ডেফিনিশন ব্যবহার করে **Cook-Levin** প্রমাণ করে k-sat একটি NP complete প্রবলেম। এটাই ছিল প্রথম প্রমাণ। Cook পরবর্তিতে কম্পিউটার সায়েন্সের সর্বোচ্চ টুরিং পুরস্কার পান তার কাজের জন্য। k-sat প্রবলেম যদি তুমি পলিনমিয়াল টাইমে সলভ করতে পারো ঢাকের নিমিষে তুমি আইনস্টাইন-নিউটন টাইপের মানুষের সমতুল্য হয়ে যাবে।

চুরিঙ এর একটা বিখ্যাত NP-hard প্রবলেম হলো **হান্টিং প্রবলেম**। “একটা কম্পিউটার প্রোগ্রাম দেয়া হলো, বলতে হবে এটা কখনো শেষ হবে নাকি অসীম সময় ধরে চলতে থাকবে”। এটাকে পলিনমিয়াল টাইমে সলভ বা ডেরিফাই কিছুই করা যায় না। বিস্তারিত জানতে আমার [এই লেখাটা](#) পড় কিন্তু তার আগে একটা কমিকস পড়ি:



নিচের ছবিতে প্রতিটা ক্যাটগরির কিছু প্রবলেম দেখানো হয়েছে:



প্রোগ্রামিং কনটেক্টে প্রায়ই NP প্রবলেম দেখা যায় তবে সেখানে ইনপুটের সাইজ অনেক কমিয়ে দেয়া হয় বা বিশেষ কোনো কন্ডিশন যোগ করে দেয়া হয় যাতে ব্যাকট্র্যাকিং করে ৩-১০ সেকেন্ডের মাঝে সলভ করা সম্ভব হয়। অনেক রিয়েল লাইফ সমস্যার পলিনমিয়াল সলিউশন না থাকায় বিজ্ঞানীরা চেষ্টা করছেন এক্সপোনেন্টসিয়াল সলিউশনকেই যতটা সম্ভব উন্নত করার। ব্রাওও এন্ড বাউল্ড টেকনিকের সাহায্যে ব্যাকট্র্যাকিং এ সার্চ স্পেস কমিয়ে মোটামুটি দ্রুত সলিউশন বের করা যায়। অনেক সময় হিউরিস্টিক এর সাহায্য নেয়া হয়। ট্রাডেলিং সেলসম্যান, গ্রাফ কালারিং ইত্যাদি সলভ করার জন্য অনেক আধুনিক অ্যালগোরিদম আছে, এগুলো নিয়ে এখনো রিসার্চ চলছে, তোমার আগ্রহ থাকলে কিছু রিসার্চ পেপার নামিয়ে নিজেই পড়লেখা করতে পারো।

বেফাৰেন্স:

<http://www.quora.com/What-are-P-NP-NP-complete-and-NP-hard>

<http://stackoverflow.com/questions/1857244/np-vs-np-complete-vs-np-hard-what-does-it-all-mean>

http://en.wikipedia.org/wiki/P_versus_NP_problem

Introduction to Algorithms by Cormen

http://en.wikipedia.org/wiki/Hamiltonian_path_problem

http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Non-deterministic_Turing_machine.html

<http://plato.stanford.edu/entries/turing-machine/#Computability>

তানভীরুল ইসলামকে ধন্যবাদ নন-ডিটারমিনিস্টিক এর ব্যাপারটা আমাকে বুঝিয়ে বলার জন্য।

ফেসবুকে মন্তব্য

5 comments

হাল্টিং প্রবলেম

সেপ্টেম্বর ২৭, ২০১৬ by শাফায়েত



গণিত বা কম্পিউটার বিজ্ঞানের সব সমস্যাই কি সমাধানযোগ্য? আমরা জানি NP ক্যাটাগরির সমস্যাগুলোকে পলিনোমিয়াল সময়ে সমাধান করার সম্ভব নাকি সেটা জানা এখন পর্যন্ত সম্ভব হ্য নি, কিন্তু ইনপুটের আকার যথেষ্ট ছোট হলে অথবা তোমার হাতে অসীম সময় এবং মেমরি থাকলে NP সমস্যাও এক্সপোনেনশিয়াল সময়ে সমাধান করা সম্ভব। কিন্তু এমন কিছু সমস্যা আছে যেটা তোমার হাতে যত বড় সুপার কম্পিউটারই থাকুক সমাধান করা সম্ভব না। এখনে আমি ধরে নিচ্ছি আমাদের কম্পিউটারগুলো **টুরিং মেশিন কম্পিউটেবল**। (টুরিং মেশিন কি মনে না থাকলে আগে আমার [এই লেখাটা পড়ো](#))

হাল্টিং প্রবলেম (Halting Problem) এমনই একটা সমস্যা যার কোনো সমাধান নেই। তোমাকে একটা কম্পিউটার প্রোগ্রাম এবং একটা ইনপুট দেয়া হলো। তোমাকে বলতে হবে সেই ইনপুটের জন্য প্রোগ্রামটা কি সীমিত সময়ে থামবে নাকি অসীম বা ইনফিনিটি লুপে আটকে যাবে?

```

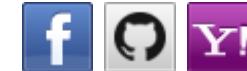
1 def my_program(x):
2     count = 0
3     while True:
4         if count == x:
5             break
6         count = count + 1

```

উপরের কোডে তুমি যদি $x = -10$ ইনপুট দাও তাহলে প্রোগ্রামটা কখনোই থামবে না (কোডটা পাইথনে লেখা, ইন্টিজার ও ড্বারফো এর ভয় নেই)

তোমার কাজ হলো এমন একটা প্রোগ্রাম লেখা যেটা যেকোনো আরেকটা প্রোগ্রাম এবং ইনপুট দেখে বলে দিতে পারবে প্রোগ্রামটা ইনফিনিটি লুপে আটকে যাবে নাকি। এটাই হলো হাল্টিং প্রবলেম (Halting Problem)।

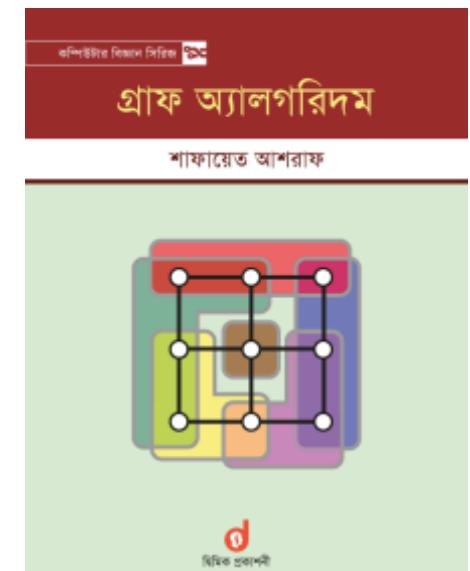
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



এখন আমরা “প্রফুল্ল বাই কনট্রাডিকশন” মেথডের সাহায্যে প্রমাণ করবো যে হাল্টিং প্রবলেম সমস্যার সমাধান করা সম্ভব না। আমরা প্রথমে ধরে নিবো যে সমস্যাটির একটা সমাধান আছে এবং তারপর প্রমাণ করবো যে সমাধানটি অস্তিত্ব থাকা অসম্ভব।

ধরো আমাদের কাছে একটা প্রোগ্রাম আছে যেটা হাল্টিং প্রবলেম সমাধান করতে পারে। প্রোগ্রামটা হতে পারে এরকম:

```
1 will_it_halt(P, I)
2   if P(I) halts in finite time
3     return TRUE
4   else
5     return FALSE
6
```

এটা একটা কান্নানিক কোড। কোডটার প্যারামিটার হলো একটা প্রোগ্রাম P এবং P কে প্যারামিটার হিসাবে পাঠানো হবে এমন একটা ইনপুট।। কোডটা কোন উপায়ে বলে দিতে পারে P প্রোগ্রামে। ইনপুট পাঠানো হলে সেটা সীমিত সময়ে থামবে নাকি থামবে না।

এ ধরণের কোডকে বলা হয় **Oracle Machine**। Oracle এর আক্ষরিক অর্থ হলো একজন ওরা যে আধ্যাত্মিক ক্ষমতা দিয়ে সমস্যা সমাধান করতে পারে। আমাদের will_it_halt কোডও জাদুকরি ক্ষমতা দিয়ে হাল্টিং প্রবলেম সমাধান করতে পারে। এখন আমরা যদি প্রমাণ করতে পারি এমন কোনো Oracle Machine থাকা সম্ভব না যেটা দিয়ে হাল্টিং প্রবলেম সমাধান করা যায় তাহলেই আমাদের কাজ শেষ।

এখন আমরা আবেকটা প্রোগ্রাম লিখবো, মনে করো প্রোগ্রামটার নাম প্যারাডক্স (Paradox)।

```
1 paradox(program)
2   will_it_halt(program, program)
3     Run Forever
4   else
5     return TRUE
6
```

প্যারাডক্স ইনপুট হিসাবে একটা প্রোগ্রামকে গ্রহণ করে। একটা প্রোগ্রাম হলো কিছু স্ট্রিং দিয়ে লেখা কিছু ইনস্ট্রাকশন। তাই আমরা চাইলে প্যারামিটার হিসাবে প্যারাডক্স প্রোগ্রামটাকেই পাঠাতে পারি!

```
1 paradox(paradox)
```



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে
হাতেখড়ি

কনফিউজড প্রোগ্রাম

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)
হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্টোরেজ(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

এখন দ্বিতীয় লাইনে আমরা ওবাকল মেশিনকে জিজ্ঞেস করছি প্যারাডক্স প্রোগ্রামে ইনপুট হিসাবে প্যারাডক্স প্রোগ্রামটাকেই পাঠালে সেটা ইনফিনিটি লুপে আটকে যাবে নাকি যাবে না। যদি ওবাকল বলে যে ইনফিনিটি লুপে আটকে যাবে না, তাহলে আমরা ওয়ালে সেটাকে ইনফিনিটি লুপে আটকে দিবো! আর ওবাকল যদি বলে যে ইনফিনিটি লুপে আটকে যাবে তাহলে আমরা প্রোগ্রাম থেকে বের হয়ে যাবো। তারমানে ওবাকল যা বলছে সেটা সত্য না, বরং তার উল্টাটা ঘটছে। এটাই হলো কন্ট্রাডিকশন, এ থেকে আমরা বলতে পারি will_it_halt নামক ওবাকল মেশিনটার অস্তিত্ব থাকা সম্ভব না!

এটাই হলো হাল্টিং প্রবলেম যা সমাধানযোগ্য না সেটার প্রমাণ।

মোবাইলে কিছু কিছু অ্যাপ চালু করলে কোনো একটা বাগের জন্য স্ক্রীন ফ্রিজ হয়ে যায়, তখন ফোনের ব্যাটারি খুলে রিস্টার্ট করা ছাড়া উপায় থাকে না। এমন কোনো অ্যাপ কি তুমি লিখতে পারবে যেটা বলে দিতে পারবে কোন অ্যাপ চালালে স্ক্রীণ ফ্রিজ হয়ে যেতে পারে? এটা হাল্টিং প্রবলেমেরই একটা ভ্যারিয়েশন যা **এই লেখাটায়** সুন্দর করে ব্যাখ্যা করা আছে, আগ্রহ থাকলে পড়তে পারো।

হাল্টিং প্রবলেম যদি সমাধানযোগ হতো তাহলে কি হত? তাহলে কোড ডিবাগিং করা অনেক সহজ হয়ে যেতে, কোড চালু না করেই আমরা বলতে পারতাম কোডটা ইনফিনিটি লুপে পড়বে নাকি। এর থেকেও গুরুত্বপূর্ণ ব্যাপার হলো, অনেক গাণিতিক সমস্যা সমাধান করা যেত এই will_it_halt প্রোগ্রামটা দিয়ে। একটা উদাহরণ দেই। গোল্ডবার্থ নামক একজন গণিতবিদ বহু বছর আগে বলে গেছেন “২ এর থেকে বড় যে কোন জোড় সংখ্যাকে দুটি প্রাইম সংখ্যার যোগফল হিসাবে লেখা সম্ভব”। কেও এখনো প্রমাণ করতে পারে নি যে কনজেকচারটা সত্য নাকি মিথ্যা। এখন আমি একটা প্রোগ্রাম লিখলাম প্রমাণ করার জন্য:

```
1 def goldbach():
2     k = 4
3     while True:
4         ok = False
5         for p1 in range(k):
6             for p2 in range(k):
7                 if(prime(p1) && prime(p2) && p1+p2==k):
8                     ok = True
9                 if not ok:
10                     exit()
11                 k += 2
```

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

এই প্রোগ্রামটা $k=4$ থেকে শুরু করে প্রতিটি জোড় সংখ্যাকে ক্রুট ফোর্সের মাধ্যমে দুটি প্রাইম সংখ্যার যোগফল হিসাবে লেখার চেষ্টা করবো। যদি কোনো k এর জন্য $ok = \text{false}$ হয় তাহলে প্রমাণ হবে যে কনজেকচারটা মিথ্যা, তখন প্রোগ্রামটা বন্ধ হয়ে যাবে। আব যদি কনজেকচারটা সত্য হয় তাহলে অসীম সময় ধরে প্রোগ্রামটা চলতে থাকবে। `will_it_halt` প্রোগ্রামটার অস্তিত্ব থাকলে এখন `will_it_halt(goldback, null)` এর আউটপুট দেখেই বলে দিতে পারতাম কনজেকচারটি সত্য নাকি মিথ্যা!

তুমি যদি এমন একটা গাণিতিক মডেল বের করতে পারো যা হার্টিং প্রবলেমকে সমাধান করতে পারে তাহলে আমরা আমরা একটা **হাইপার কম্পিউটেশন** বা **super-Turing computation** মডেল পাবো, তখন আমরা এমন সব সমস্যা সমাধান করতে পারবো যা টুরিং মেশিন দিয়ে সমাধান করা সম্ভব না।

হার্টিং প্রবলেম অনেক দার্শনিক প্রশ্নেরও জন্ম দিয়েছে। যেমন আমরা কি কখনো নিজের মন্তিষ্ঠ সম্পর্কে ১০০% জানতে পারবো? নাকি কোনো খিওরিটিকাল সীমাবদ্ধতার কারণে মন্তিষ্ঠের অনেক বহস্য আমরা কোনো জানতে পারবো না?

আজ এখানেই শেষ, হ্যাপি কোডিং!

রেফারেন্স

<http://www.cgl.uwaterloo.ca/csk/halt/>

<https://www.quora.com/If-the-halting-problem-was-solvable-what-would-be-the-implications>

ফেসবুকে মন্তব্য

2 comments

আর্টিকুলেশন পয়েন্ট এবং বিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম খিওরি(৩):

গেম খিওরি-১

গেম খিওরি-২

গেম খিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সারসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

বাইনারি সার্চ - ১

জুন ৭, ২০১৫ by শাফায়েত



তুমি নিশ্চয়ই লক্ষ্য করেছো ডিকশনারিতে লাখ লাখ শব্দ থাকলেও প্রযোজনীয় শব্দটা খুজে পেতে কখনো খুব বেশি সময় লাগে না। এটার কারণ হলো শব্দগুলো অক্ষর অনুযায়ী সাজানো থাকে। তাই তুমি যদি dynamite শব্দটা ডিকশনারিতে খোজার চেষ্টা করো এবং এলোমেলোভাবে কোনো একটা পাতা খুলে kite শব্দটা খুজে পাও তাহলে তুমি নিশ্চিত হয়ে যেতে পারো যে তুমি যে শব্দটা খোজার চেষ্টা করেছো সেটা বাম দিকে কোথাও আছে। আবার যদি তুমি dear শব্দটা খুজে পাও তখন তুমি আর বাম দিকের পাতাগুলোয় খোজার চেষ্টা করবে না। এভাবে অন্ন সময়ের মধ্যে ডিকশনারিতে যেকোনো শব্দ খুজে পাওয়া যায়।

বাইনারি সার্চ হলো অনেকটা এরকম একটা পদ্ধতি যেটা ব্যবহার করে একটা অ্যারে থেকে কোনো একটা তথ্য খুজে বের করা যায়।

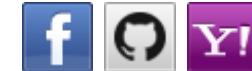
ধৰো তোমার কাছে একটা অ্যারেতে অনেকগুলো সংখ্যা আছে এরকম:

“ 100, 2, 10, 50, 20, 500, 100, 150, 200, 1000, 100

সংখ্যাগুলো এলোমেলোভাবে সাজানো থাকায় এখন থেকে একটা নির্দিষ্ট সংখ্যা খুজে পাওয়া সহজ না। তুমি যদি অ্যারে থেকে ৫০০ সংখ্যাটা খুজতে চাও তাহলে সবগুলো ইনডেক্সই পরীক্ষা করে দেখতে হবে, এটাকে বলা হয় লিনিয়ার সার্চ। অ্যারের আকার যদি n হয় তাহলে লিনিয়ার সার্চের টাইম কমপ্লেক্সিটি হলো $O(n)$ ।

কিন্তু যদি সংখ্যাগুলো নিচের মত সাজানো থাকে তাহলে খুজে পাওয়া অনেক সহজ হয়ে যায়:

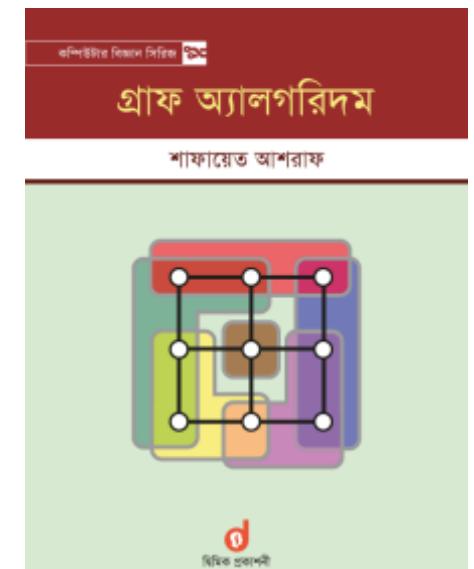
সাবক্সাইব



Secured by OneAll Social Login

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



“ 2, 10, 20, 50, 100, 100, 100, 150, 200, 500, 1000

এখন যদি ১৫০ সংখ্যাটা খুজে বের করতে হয় তাহলে আমরা প্রথমে ঠিক মাঝের ইনডেক্সটা পরীক্ষা করবো। প্রথম ইনডেক্স ০ এবং শেষের ইনডেক্স ১০ হলে মাঝের ইনডেক্সটা হলো $(0+10)/2$ বা ৫ তম ইনডেক্স।

“ \$2, 10, 20, 50, 100, **100**, 100, 150, 200, 500, 1000

মাঝের ইনডেক্সের সংখ্যাটা ১০০ যা ১৫০ এর থেকে ছোট, আমরা জেনে গেলাম যে সংখ্যাটা ডান পাশে কোথাও আছে, বামের অংশ আমাদের আর দরকার নাই।

“ 100, 150, **200**, 500, 1000

এখন বাকি অংশটা নিয়ে আবারো একই কাজ করবো। এবার মাঝের ইনডেক্সের সংখ্যাটা হলো ২০০ যেটা ১৫০ এর থেকে বড়। তারমানে ডানের অংশটা আমরা ফেলে দিতে পারি।

“ **100**, 150

এবার মাঝের সংখ্যাটা হলো ১০০ যা ১৫০ এর থেকে ছোট। আবারো বামের অংশ ফেলে দিবো, থাকবে শুধু:

“ **150**

এবার মাঝের ইনডেক্সের সংখ্যাটা হলো ১৫০। তারমানে কাঞ্চিত সংখ্যাটাকে খুজে পাওয়া গেছে।

বাইনারি সার্চে প্রতিবার অ্যাবের ঠিক অর্ধেক অংশ আমরা বাতিল করে দিচ্ছি এবং বাকি অর্ধেক অংশে খুজছি। একটা সংখ্যা n কে সর্বোচ্চ কয়বার ২ দিয়ে ভাগ করা যায় যতক্ষণ না সংখ্যাটা ১ হয়ে যাচ্ছে? উত্তর হলো $\log_2 n$ । তাই বাইনারি সার্চে সর্বোচ্চ $\log_2(n)$ সংখ্যক ধাপের পর আমরা দরকারি সংখ্যাটা খুজে পাবো, **কমপ্লেক্সিটি** $O(\log_2 n)$ ।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

কিন্তু একটা সমস্যা হলো বাইনারি সার্চ করার আগে অবশ্যই সংখ্যাগুলোকে ছোট থেকে বড় বা বড় থেকে ছোট সাজিয়ে নিতে হবে, এই প্রক্রিয়াটাকে বলা হয় সট্টিং। তুমি যতই চেষ্টা কর না কেন একটা অ্যারে $O(n \times \log_2 n)$ এর কম কমপ্লেক্সিটি সর্ট করতে পারবে না।* তাহলে কেও বলতে পারে যে সর্ট করতে যে সময় লাগছে তার থেকে অনেক কম সময়ে লিনিয়ার সার্চ করেই আমরা সংখ্যাটা খুজে পেতে পারি, বাইনারি সার্চ কেন করবো? তোমার যদি একটা অ্যারেতে মাত্র ১ বার সার্চ করা দরকার হয় তাহলে কষ্ট করে সর্ট করে বাইনারি সার্চ করার থেকে লিনিয়ার সার্চ করা অনেক ভালো। কিন্তু যদি এমন হয় একটা অ্যারেতে অনেকবার সার্চ করা দরকার হবে? যেমন একটা ডিকশনারিতে বিডিম সময় বিডিম শব্দ খোজা দরকার হয়, সেক্ষেত্রে সর্ট করে রাখাই বুদ্ধিমানের কাজ।

নিচের পাইথন কোডে বাইনারি সার্চের ইমপ্লিমেন্টেশন দেখানো হচ্ছে:

```
1 def search(array, key):
2     begin=0
3     end=len(array)-1
4     index=None
5     while begin<=end:
6         mid=(begin+end)/2
7         if key == array[mid]:
8             index=mid #The value is found, save the index.
9             break
10        elif key > array[mid]: begin=mid+1 #Search the right portion
11        elif key < array[mid]: end=mid-1 #Search the left portion
12    return index #If the number is not found, index will contain None.
13
14
15 info=[100,2,10,50,20,500,100,150,200,1000,100]
16 info=sorted(info)
17 while True:
18     key = int(raw_input())
19     print search(info,key)
```

উপরের কোডে যদি আমরা ১০০ ইনপুট দেই তাহলে ৫ রিটার্ন করবে, কারণ অ্যারেটা সর্ট করার পর ৫ নম্বর ইনডেক্সে ১০০ আছে। কিন্তু সর্টেড অ্যারের দিকে তাকালে আমরা দেখছি যে ৪,৫,৬ এই সবগুলো ইনডেক্সে ১০০ আছে। আমরা যদি চাই কোনো সংখ্যা একাধিকবার থাকলে সবথেকে বামের ইনডেক্সটা রিটার্ন করতে, তাহলে কোডটা কিভাবে পরিবর্তন করতে হবে? সেক্ষেত্রে বাইনারি সার্চ ফাংশনটা হবে এরকম:

স্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্লেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

1 def search(array, key):
2     begin=0
3     end=len(array)-1
4     index=None
5     while begin<=end:
6         mid=(begin+end)/2
7         if key == array[mid]:
8             index=mid #One occurrence of the value is found, save the index
9             end=mid-1 #Continue searching the left portion after one occurrence is found
10        elif key > array[mid]: begin=mid+1
11        elif key < array[mid]: end=mid-1
12    return index #Index will contain None if the value is not found

```

শুধুমাত্র ১টা মাত্র লাইন পরিবর্তন করা হয়েছে, এবার কোনো সংখ্যা খুজে পাবার পর খোজা বন্ধ না করে বামের বাকি অংশটুকুতে খুজতে থাকবো।

লোয়ার বাউন্ড: তোমাকে একটা সেটেড অ্যারে দেয়া আছে। তুমি নতুন একটা সংখ্যা X সেই অ্যারেতে চুকাতে চাও। লোয়ার বাউন্ড হলো সবথেকে বামের ইনডেক্স যেখানে তুমি সংখ্যাটা ঢুকিয়ে বাকি সংখ্যাগুলোকে একঘর ডানে সরালে অ্যারেটা তখনো সেটেড থাকবে। মনে করো অ্যারেটা এরকম:

“ 10 20 20 30 30 40 50

তাহলে 20 এর লোয়ার বাউন্ড হলো ইনডেক্স 1 (0 বেসড ইনডেক্স), কারণ 1 নম্বর ইনডেক্সে তুমি 20 সংখ্যাটাকে বসিয়ে বাকি সংখ্যাগুলোকে ডানে সরিয়ে দেয়ার পরেও অ্যারেটা সেটেড থেকে যাচ্ছে। ঠিক সেরকম 25 এর জন্য লোয়ার বাউন্ড হলো ইনডেক্স 3। সহজ কথায় সবথেকে বামের যে ইনডেক্সে X এর সমান বা বড় কোনো সংখ্যা আছে সেই ইনডেক্সটাই হলো লোয়ার বাউন্ড। X যদি অ্যারের প্রতিটা সংখ্যার থেকে বড় হয় তাহলে সর্বশেষ ইনডেক্সের পরবর্তী ইনডেক্সটা অর্থাৎ n তম ইনডেক্সটা হলো লোয়ার বাউন্ড যেখানে n হলো অ্যারের আকার।

নিচের কোডে বাইনারি সার্চ করে লোয়ার ইনডেক্স খুজে বের করা হয়েছে, তারপর সেই ইনডেক্সে নতুন সংখ্যাটা বসিয়ে দেয়া হয়েছে।

```

1 def searchLowerBound(array, key):
2     begin=0
3     end=len(array)-1
4     index=-1
5     while begin<=end:

```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেস্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন

সার্ভিসিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্বসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

```

6     mid=(begin+end)/2
7     if key == array[mid]:
8         index=mid
9         end=mid-1
10    elif key > array[mid]: begin=mid+1
11    elif key < array[mid]: end=mid-1
12    return begin
13
14
15
16 info=[100,2,10,50,20,500,100,150,200,1000,100]
17 info=sorted(info)
18 print info
19 while True:
20     X = int(raw_input())
21     lowerbound=searchLowerBound(info,X)
22     info.insert(lowerbound,X)
23     print "New array: ",info

```

(অ্যারেতে এভাবে সংখ্যা ইনসার্ট করতে $O(n)$ সময় লাগে। অ্যারের জায়গায় লিংক লিস্ট ব্যবহার করে $O(1)$ সময়ে ইনসার্ট করা যায় কিন্তু সেক্ষেত্রে বাইনারি সার্চ করা সম্ভব না কারণ লিংক লিস্টে যেকোনো ইনডেক্স রেডম এক্সেস করা যায় না, পয়েন্টার ধরে আগতে হয়। একটা উপায় হতে পারে বাইনারি সার্চ ট্রি আকারে তথ্যগুলো সাজিয়ে রাখা, সেটা আমরা এই লেখায় দেখবো না, তুমি চাইলে নিজে শিখে নিতে পারো।)

আপার বাটন্ড: আপার বাটন্ড হলো সব থেকে ডানের ইনডেক্স যেখানে তুমি নতুন সংখ্যাটা ঢুকালে অ্যারেটা সর্টেড থাকবে। তারমানে সবথেকে বামের যে ইনডেক্সে **X** এর বড় কোনো সংখ্যা আছে সেই ইনডেক্সটাই হলো আপার বাটন্ড। উপরের উদাহরণটায় ২০ এর আপার বাটন্ড হলো ইনডেক্স ৩। আপার বাটন্ড বের করার কোডটা তোমার বাড়ির কাজ হিসাবে থাকলো!

প্রথম পর্ব এখানেই শেষ। পরের পর্বে বাইনেক্ষন মেথড নিয়ে আলোচনা করবো।

প্র্যাকটিসের জন্য প্রবলেম:

<https://uva.onlinejudge.org/index.php?>

option=onlinejudge&Itemid=8&page=show_problem&problem=1552http://www.lightoj.com/volume_showproblem.php?
problem=1088

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

হ্যাপি কোডিং!

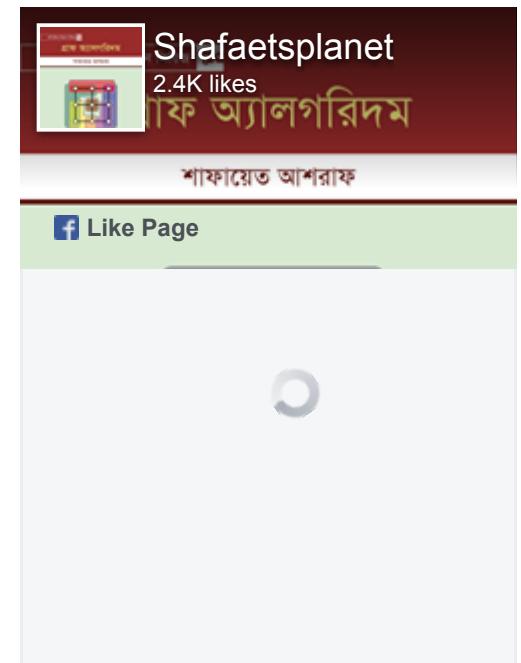
* কাউটিং সর্ট বা রেডিভ্র সর্ট ব্যবহার করে $O(n)$ এ সর্টিং করা যায় তবে সেগুলো কাজ করে বিশেষ ধরণের কিছু ইনপুটের ক্ষেত্রে। ইনপুট কি ধরণের হবে সেটার উপর কোনো শর্ত না থাকলে $O(\log_2 n)$ এর কমে সর্টিং করা সম্ভব না সেটা গাণিতিকভাবে প্রমাণ করা যায়।

ফেসবুকে মন্তব্য

3 comments

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



বাইনারি সার্চ - ১

জুন ৭, ২০১৫ by শাফায়েত



তুমি নিশ্চয়ই লক্ষ্য করেছো ডিকশনারিতে লাখ লাখ শব্দ থাকলেও প্রযোজনীয় শব্দটা খুজে পেতে কখনো খুব বেশি সময় লাগে না। এটার কারণ হলো শব্দগুলো অক্ষর অনুযায়ী সাজানো থাকে। তাই তুমি যদি dynamite শব্দটা ডিকশনারিতে খোজার চেষ্টা করো এবং এলোমেলোভাবে কোনো একটা পাতা খুলে kite শব্দটা খুজে পাও তাহলে তুমি নিশ্চিত হয়ে যেতে পারো যে তুমি যে শব্দটা খোজার চেষ্টা করেছো সেটা বাম দিকে কোথাও আছে। আবার যদি তুমি dear শব্দটা খুজে পাও তখন তুমি আর বাম দিকের পাতাগুলোয় খোজার চেষ্টা করবে না। এভাবে অন্ন সময়ের মধ্যে ডিকশনারিতে যেকোনো শব্দ খুজে পাওয়া যায়।

বাইনারি সার্চ হলো অনেকটা এরকম একটা পদ্ধতি যেটা ব্যবহার করে একটা অ্যাবে থেকে কোনো একটা তথ্য খুজে বের করা যায়।

ধৰো তোমার কাছে একটা অ্যাবেতে অনেকগুলো সংখ্যা আছে এরকম:

“ 100, 2, 10, 50, 20, 500, 100, 150, 200, 1000, 100

সংখ্যাগুলো এলোমেলোভাবে সাজানো থাকায় এখন থেকে একটা নির্দিষ্ট সংখ্যা খুজে পাওয়া সহজ না। তুমি যদি অ্যাবে থেকে ৫০০ সংখ্যাটা খুজতে চাও তাহলে সবগুলো ইনডেক্সই পরীক্ষা করে দেখতে হবে, এটাকে বলা হয় লিনিয়ার সার্চ। অ্যাবের আকার যদি n হয় তাহলে লিনিয়ার সার্চের টাইম কমপ্লেক্সিটি হলো $O(n)$ ।

কিন্তু যদি সংখ্যাগুলো নিচের মত সাজানো থাকে তাহলে খুজে পাওয়া অনেক সহজ হয়ে যায়:

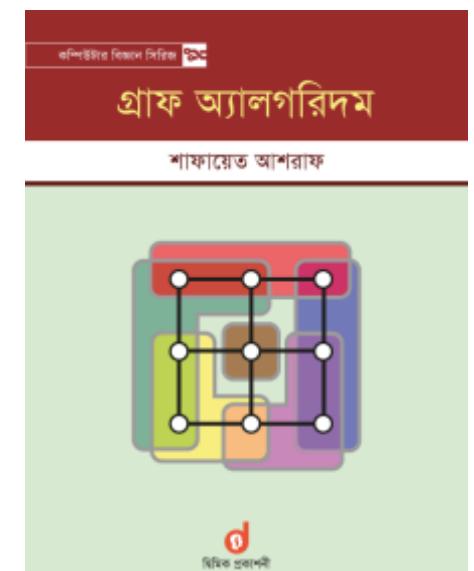
সাবক্সাইব



Secured by OneAll Social Login

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



“ 2, 10, 20, 50, 100, 100, 100, 150, 200, 500, 1000

এখন যদি ১৫০ সংখ্যাটা খুজে বের করতে হয় তাহলে আমরা প্রথমে ঠিক মাঝের ইনডেক্সটা পরীক্ষা করবো। প্রথম ইনডেক্স ০ এবং শেষের ইনডেক্স ১০ হলে মাঝের ইনডেক্সটা হলো $(0+10)/2$ বা ৫ তম ইনডেক্স।

“ \$2, 10, 20, 50, 100, **100**, 100, 150, 200, 500, 1000

মাঝের ইনডেক্সের সংখ্যাটা ১০০ যা ১৫০ এর থেকে ছোট, আমরা জেনে গেলাম যে সংখ্যাটা ডান পাশে কোথাও আছে, বামের অংশ আমাদের আর দরকার নাই।

“ 100, 150, **200**, 500, 1000

এখন বাকি অংশটা নিয়ে আবারো একই কাজ করবো। এবার মাঝের ইনডেক্সের সংখ্যাটা হলো ২০০ যেটা ১৫০ এর থেকে বড়। তারমানে ডানের অংশটা আমরা ফেলে দিতে পারি।

“ **100**, 150

এবার মাঝের সংখ্যাটা হলো ১০০ যা ১৫০ এর থেকে ছোট। আবারো বামের অংশ ফেলে দিবো, থাকবে শুধু:

“ **150**

এবার মাঝের ইনডেক্সের সংখ্যাটা হলো ১৫০। তারমানে কাঞ্চিত সংখ্যাটাকে খুজে পাওয়া গেছে।

বাইনারি সার্চে প্রতিবার অ্যাবের ঠিক অর্ধেক অংশ আমরা বাতিল করে দিচ্ছি এবং বাকি অর্ধেক অংশে খুজছি। একটা সংখ্যা n কে সর্বোচ্চ কয়বার ২ দিয়ে ভাগ করা যায় যতক্ষণ না সংখ্যাটা ১ হয়ে যাচ্ছে? উত্তর হলো $\log_2 n$ । তাই বাইনারি সার্চে সর্বোচ্চ $\log_2(n)$ সংখ্যক ধাপের পর আমরা দরকারি সংখ্যাটা খুজে পাবো, **কমপ্লেক্সিটি** $O(\log_2 n)$ ।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

কিন্তু একটা সমস্যা হলো বাইনারি সার্চ করার আগে অবশ্যই সংখ্যাগুলোকে ছোট থেকে বড় বা বড় থেকে ছোট সাজিয়ে নিতে হবে, এই প্রক্রিয়াটাকে বলা হয় সট্টিং। তুমি যতই চেষ্টা কর না কেন একটা অ্যারে $O(n \times \log_2 n)$ এর কম কমপ্লেক্সিটি সর্ট করতে পারবে না।* তাহলে কেও বলতে পারে যে সর্ট করতে যে সময় লাগছে তার থেকে অনেক কম সময়ে লিনিয়ার সার্চ করেই আমরা সংখ্যাটা খুজে পেতে পারি, বাইনারি সার্চ কেন করবো? তোমার যদি একটা অ্যারেতে মাত্র ১ বার সার্চ করা দরকার হয় তাহলে কষ্ট করে সর্ট করে বাইনারি সার্চ করার থেকে লিনিয়ার সার্চ করা অনেক ভালো। কিন্তু যদি এমন হয় একটা অ্যারেতে অনেকবার সার্চ করা দরকার হবে? যেমন একটা ডিকশনারিতে বিডিম সময় বিডিম শব্দ খোজা দরকার হয়, সেক্ষেত্রে সর্ট করে রাখাই বুদ্ধিমানের কাজ।

নিচের পাইথন কোডে বাইনারি সার্চের ইমপ্লিমেন্টেশন দেখানো হচ্ছে:

```
1 def search(array, key):
2     begin=0
3     end=len(array)-1
4     index=None
5     while begin<=end:
6         mid=(begin+end)/2
7         if key == array[mid]:
8             index=mid #The value is found, save the index.
9             break
10        elif key > array[mid]: begin=mid+1 #Search the right portion
11        elif key < array[mid]: end=mid-1 #Search the left portion
12    return index #If the number is not found, index will contain None.
13
14
15 info=[100,2,10,50,20,500,100,150,200,1000,100]
16 info=sorted(info)
17 while True:
18     key = int(raw_input())
19     print search(info,key)
```

উপরের কোডে যদি আমরা ১০০ ইনপুট দেই তাহলে ৫ রিটার্ন করবে, কারণ অ্যারেটা সর্ট করার পর ৫ নম্বর ইনডেক্সে ১০০ আছে। কিন্তু সর্টেড অ্যারের দিকে তাকালে আমরা দেখছি যে ৪,৫,৬ এই সবগুলো ইনডেক্সেই ১০০ আছে। আমরা যদি চাই কোনো সংখ্যা একাধিকবার থাকলে সবথেকে বামের ইনডেক্সটা রিটার্ন করতে, তাহলে কোডটা কিভাবে পরিবর্তন করতে হবে? সেক্ষেত্রে বাইনারি সার্চ ফাংশনটা হবে এরকম:

স্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্লেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

1 def search(array, key):
2     begin=0
3     end=len(array)-1
4     index=None
5     while begin<=end:
6         mid=(begin+end)/2
7         if key == array[mid]:
8             index=mid #One occurrence of the value is found, save the index
9             end=mid-1 #Continue searching the left portion after one occurrence is found
10        elif key > array[mid]: begin=mid+1
11        elif key < array[mid]: end=mid-1
12    return index #Index will contain None if the value is not found

```

শুধুমাত্র ১টা মাত্র লাইন পরিবর্তন করা হয়েছে, এবার কোনো সংখ্যা খুজে পাবার পর খোজা বন্ধ না করে বামের বাকি অংশটুকুতে খুজতে থাকবো।

লোয়ার বাউন্ড: তোমাকে একটা সেটেড অ্যারে দেয়া আছে। তুমি নতুন একটা সংখ্যা X সেই অ্যারেতে চুকাতে চাও। লোয়ার বাউন্ড হলো সবথেকে বামের ইনডেক্স যেখানে তুমি সংখ্যাটা ঢুকিয়ে বাকি সংখ্যাগুলোকে একঘর ডানে সরালে অ্যারেটা তখনো সেটেড থাকবে। মনে করো অ্যারেটা এরকম:

“ 10 20 20 30 30 40 50

তাহলে 20 এর লোয়ার বাউন্ড হলো ইনডেক্স 1 (0 বেসড ইনডেক্স), কারণ 1 নম্বর ইনডেক্সে তুমি 20 সংখ্যাটাকে বসিয়ে বাকি সংখ্যাগুলোকে ডানে সরিয়ে দেয়ার পরেও অ্যারেটা সেটেড থেকে যাচ্ছে। ঠিক সেরকম 25 এর জন্য লোয়ার বাউন্ড হলো ইনডেক্স 3। সহজ কথায় সবথেকে বামের যে ইনডেক্সে X এর সমান বা বড় কোনো সংখ্যা আছে সেই ইনডেক্সটাই হলো লোয়ার বাউন্ড। X যদি অ্যারের প্রতিটা সংখ্যার থেকে বড় হয় তাহলে সর্বশেষ ইনডেক্সের পরবর্তী ইনডেক্সটা অর্থাৎ n তম ইনডেক্সটা হলো লোয়ার বাউন্ড যেখানে n হলো অ্যারের আকার।

নিচের কোডে বাইনারি সার্চ করে লোয়ার ইনডেক্স খুজে বের করা হয়েছে, তারপর সেই ইনডেক্সে নতুন সংখ্যাটা বসিয়ে দেয়া হয়েছে।

```

1 def searchLowerBound(array, key):
2     begin=0
3     end=len(array)-1
4     index=-1
5     while begin<=end:

```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেস্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন

সার্ভিসিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্বসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

```

6     mid=(begin+end)/2
7     if key == array[mid]:
8         index=mid
9         end=mid-1
10    elif key > array[mid]: begin=mid+1
11    elif key < array[mid]: end=mid-1
12    return begin
13
14
15
16 info=[100,2,10,50,20,500,100,150,200,1000,100]
17 info=sorted(info)
18 print info
19 while True:
20     X = int(raw_input())
21     lowerbound=searchLowerBound(info,X)
22     info.insert(lowerbound,X)
23     print "New array: ",info

```

(অ্যারেতে এভাবে সংখ্যা ইনসার্ট করতে $O(n)$ সময় লাগে। অ্যারের জায়গায় লিংক লিস্ট ব্যবহার করে $O(1)$ সময়ে ইনসার্ট করা যায় কিন্তু সেক্ষেত্রে বাইনারি সার্চ করা সম্ভব না কারণ লিংক লিস্টে যেকোনো ইনডেক্স রেডম এক্সেস করা যায় না, পয়েন্টার ধরে আগতে হয়। একটা উপায় হতে পারে বাইনারি সার্চ ট্রি আকারে তথ্যগুলো সাজিয়ে রাখা, সেটা আমরা এই লেখায় দেখবো না, তুমি চাইলে নিজে শিখে নিতে পারো।)

আপার বাটন্ড: আপার বাটন্ড হলো সব থেকে ডানের ইনডেক্স যেখানে তুমি নতুন সংখ্যাটা ঢুকালে অ্যারেটা সর্টেড থাকবে। তারমানে সবথেকে বামের যে ইনডেক্সে **X** এর বড় কোনো সংখ্যা আছে সেই ইনডেক্সটাই হলো আপার বাটন্ড। উপরের উদাহরণটায় ২০ এর আপার বাটন্ড হলো ইনডেক্স ৩। আপার বাটন্ড বের করার কোডটা তোমার বাড়ির কাজ হিসাবে থাকলো!

প্রথম পর্ব এখানেই শেষ। পরের পর্বে বাইনেক্ষন মেথড নিয়ে আলোচনা করবো।

প্র্যাকটিসের জন্য প্রবলেম:

<https://uva.onlinejudge.org/index.php?>

option=onlinejudge&Itemid=8&page=show_problem&problem=1552http://www.lightoj.com/volume_showproblem.php?
problem=1088

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

হ্যাপি কোডিং!

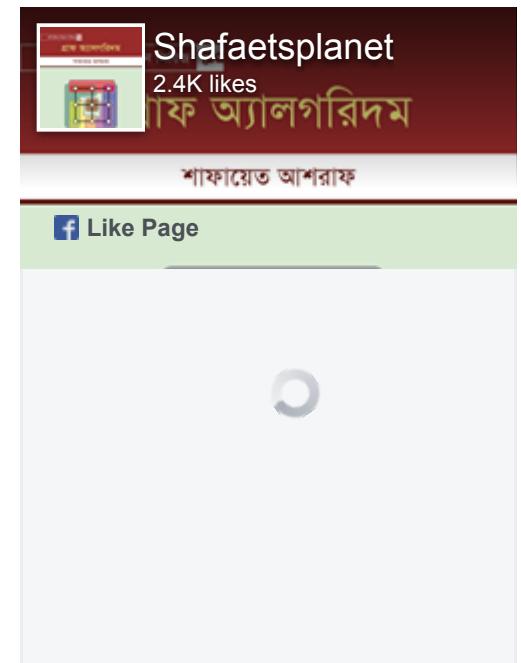
* কাউটিং সর্ট বা রেডিভ্র সর্ট ব্যবহার করে $O(n)$ এ সর্টিং করা যায় তবে সেগুলো কাজ করে বিশেষ ধরণের কিছু ইনপুটের ক্ষেত্রে। ইনপুট কি ধরণের হবে সেটার উপর কোনো শর্ত না থাকলে $O(\log_2 n)$ এর কমে সর্টিং করা সম্ভব না সেটা গাণিতিকভাবে প্রমাণ করা যায়।

ফেসবুকে মন্তব্য

3 comments

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



বাইনারি সার্চ - ২

আগস্ট ১৭, ২০১৫ by শাফায়েত



আগের লেখায় আমরা বাইনারি সার্চ কিভাবে কাজ করে দেখেছি। এখন একই পদ্ধতি ব্যবহার করে আমরা অন্যরকম কিছু সমস্যা সমাধান করবো।
আমরা এখন যেটা শিখবো সেটা বাইসেকশন মেথড নামেই বেশি পরিচিত।

সহজ একটা সমস্যা সমাধান করতে করতে আমরা বাইসেকশন কিভাবে কাজ করে দেখবো। মনে করো তুমি যে ভাষা ব্যবহার করে প্রোগ্রামিং করছ
সেখানে বর্গমূল বের করার জন্য কোনো ফাংশন নাই, তোমাকে নিজে ফাংশন লিখে নিতে হবে। আমরা $\text{mysqrt}(X)$ নামের একটা ফাংশন লিখবো
যেখানে X সংখ্যাটা পাঠালে সংখ্যাটার বর্গমূল রিটার্ন করবে, X সংখ্যাটা দশমিকযুক্ত হতে পারে, তবে শূণ্যের কম হবে না।

আমরা জটিল কোনো গাণিতিক হিসাবে যাবো না বর্গমূল বের করার জন্য, আমরা বাইনারি সার্চ করেই বর্গমূল বের করে ফেলবো!

একটু মনে করি আমরা একটা ছোট থেকে বড় সাজানো অ্যারেতে কোনো সংখ্যা খুজে বের করার সময় কি করছিলাম। মাঝখানের সংখ্যাটা যদি
বেশি বড় হয় তাহলে ডানের অংশ বাদ দিয়ে দিছিলাম, বেশি ছোটো হলে বামের অংশটা বাদ দিয়ে দিছিলাম।

বর্গমূল বের করার সময় আমরা জানি যে X এর বর্গমূল অবশ্যই 0 থেকে X এর মধ্যের একটা সংখ্যা হবে। ধরো $X=15$ । আমরা প্রথমে ঠিক মাঝের
সংখ্যাটা নিবো। এক্ষেত্রে মাঝের সংখ্যা হলো 7.5 যাকে আবার বর্গ করলে পাওয়া যায় 56.25 যা 15 এর থেকে অনেক বড়। তারমানে 7.5 থেকে
 15 পর্যন্ত বাকি সব সংখ্যার বগই 15 এর থেকে বড়, এই অংশটা আমরা বাদ দিয়ে দিতে পারি।

এখন তাহলে আমরা আবার 0 থেকে 7.5 এর মধ্যে খুজবো। মাঝের সংখ্যাটা হলো 3.75 যাকে বর্গ করলে পাই 14.0625 যা 15 থেকে ছোটো।
তারমানে 0 থেকে 3.75 পর্যন্ত অংশে বর্গমূল পাবার কোনো সম্ভাবনা নেই। এখন আবার 3.75 থেকে 7.5 এর মাঝে খুজবো।

সাবস্ক্রাইব

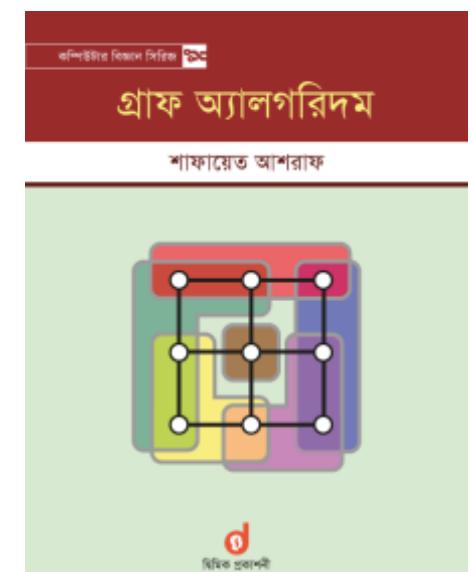


Secured by OneAll Social Login

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

Traveloka Singapore (বিস্তারিত...)



পুরো সিমুলেশনটা হাতেকলমে করছি না, আশা করি তুমি বুঝতে পারছো যে বর্গমূল না পাওয়া পর্যন্ত আমরা এভাবেই খুজতে থাকবো। নিচের কোডটা দেখো:

```
1 def mysqrt(X):
2     low=0.0
3     high=X
4     while high-low>.0001:
5         mid=(low+high)/2
6         print low, high, mid, mid*mid
7         if mid*mid>X:
8             high=mid
9         else:
10            low=mid
11
12    print mid,mid*mid
13    return mid
14
15 mysqrt(15)
```

এই কোডটা চালালে দেখবে বর্গমূল আসছে 3.87296676636 যাকে আবার বর্গ করলে পাওয়া যায় 14.9998715733। বুঝতেই পারছো দশমিক সংখ্যার প্রিসিপেনের সমস্যার কারণে একদম সঠিক উত্তর পাওয়া যায় নি, কাছাকাছি একটা উত্তর পাওয়া গিয়েছে। ৪নষ্ঠর লাইনটা খুব গুরুত্বপূর্ণ, এখানে আমরা ঠিক করছি কতক্ষণ আমরা খোজা চালিয়ে যাবো, যত বেশিক্ষণ খুজবো সঠিক উত্তরের তত কাছাকাছি পৌছাতে পারবো। এখানে আমরা `high` এবং `low` এর পার্থক্য যতক্ষণ না খুব ছোট হয়ে যাচ্ছে ততক্ষণ খুজতেসি। তুমি .0001 এর জায়গায় আরো কোনো ছোটো সংখ্যা বসালে দেখবে আগের থেকে ভালো ফলাফল পাচ্ছো। যেমন .00000001 ব্যবহার করলে বর্গমূল পাবে 3.87298334594 যাকে আবার বর্গ করলে পাওয়া যায় 14.9999999979।

তুমি চাইলে নির্দিষ্ট করে বলে দিতে পারো বাইসেকশন কয়টা ধাপ পর্যন্ত চলবে। তখন কোডটা হবে এরকম:

```
1 def mysqrt(X):
2     low=0.0
3     high=X
4     for step in range(64):
5         mid=(low+high)/2
```



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

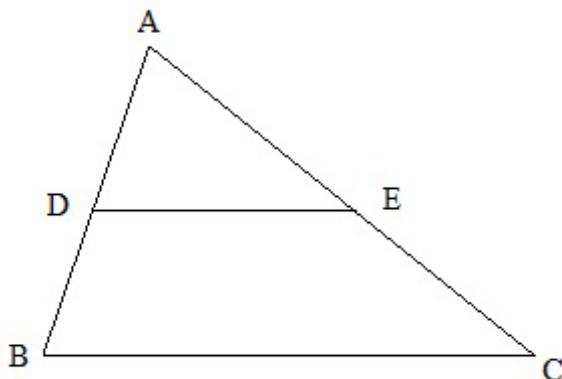
```

6     print low, high, mid, mid*mid
7     if mid*mid>X:
8         high=mid
9     else:
10        low=mid
11
12    print mid,mid*mid
13    return mid
14 mysqrt(15)
15 x

```

এবার আমরা `high`, `low` এর পার্কেয়ের কথা চিন্তা না করেই ৬৪ বার সার্চ করেছি। একটা সংখ্যাকে ৬৪ বার ২দিয়ে ভাগ করা মানে সংখ্যাটাকে প্রচল্লিষ্ণ ছোটো করে ফেলা, তাই তুমি এভাবে সার্থিক উত্তরের খুব কাছে পৌছে যাবে। তুমি চাইলে ৬৪ বারের জায়গায় ১০০ বা ২০০ বারও ভাগ করতে পারো আরো ভালো ফলাফলের জন্য, তবে সেক্ষেত্রে কোডের রানটাইমও বেড়ে যাবে। ধাপসংখ্যা নির্ধারণ করার সময় `high-low` এর মান কত বড় এবং একই সাথে প্রবলেমের টাইমলিমিটের দিকে লক্ষ্য রাখা উচিত।

বাইসেকশন ব্যবহার করে জ্যামিতির অনেক সমস্যা সমাধান করা যায়। ধরো তোমাকে নিচের মত একটা ত্রিভুজ দেয়া আছে:



AB, **AC** আর **BC** এর দৈর্ঘ্য তোমাকে দেয়া আছে, আর বলা আছে যে **DE** আর **BC** সমান্তরাল। এছাড়া **ADE** ত্রিভুজ এবং **BDEC** ট্রাপিজিয়ামের ক্ষেত্রফলের অনুপাত R ও তোমাকে দেয়া আছে। তোমাকে বলতে হবে **AD** এর দৈর্ঘ্য কত?

এটা আমরা সহজেই বাইসেকশন দিয়ে সমাধান করতে পারি। তুমি অনুমান করে নাও যে **AD** এর দৈর্ঘ্য হলো x । এখন **AD** এর দৈর্ঘ্য জানলে তুমি স্কুলে পড়া ত্রিভুজের অনুপাতের নিয়ম ($AD/AB = AE/AC = DE/BC$) দিয়ে সহজেই **AE**, **EC**, **DE** বের করে ফেলতে পারবে। এখন তুমি বাল্ব

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কম্প্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্থায়ার-রুট ডিকম্পেজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

দৈর্ঘ্য দিয়ে ADE এবং BDEC এর ক্ষেত্রফলও সহজে বের করতে পারবে। ক্ষেত্রফল জানার পর অনুপাত বের করে ফেল। যদি দেখ যে তুমি যে অনুপাতটা পেয়েছো সেটা R এর থেকে ছোটো তারমানে তুমি AD এর যে দৈর্ঘ্য x অনুমান করেছিলে সেটা আসল দৈর্ঘ্য থেকে ছোটো। তাহলে তুমি x থেকে high পর্যন্ত রেঞ্জে আবার খুজতে থাকো। আর যদি দেখো যে তোমার পাওয়া অনুপাতটা R এর থেকে বড় তাহলে 0 থেকে x পর্যন্ত রেঞ্জে খুজতে থাকো। high এর মান শুরুতে কত হবে সেটা চিন্তা করে বের করার কাজ তোমার :)।

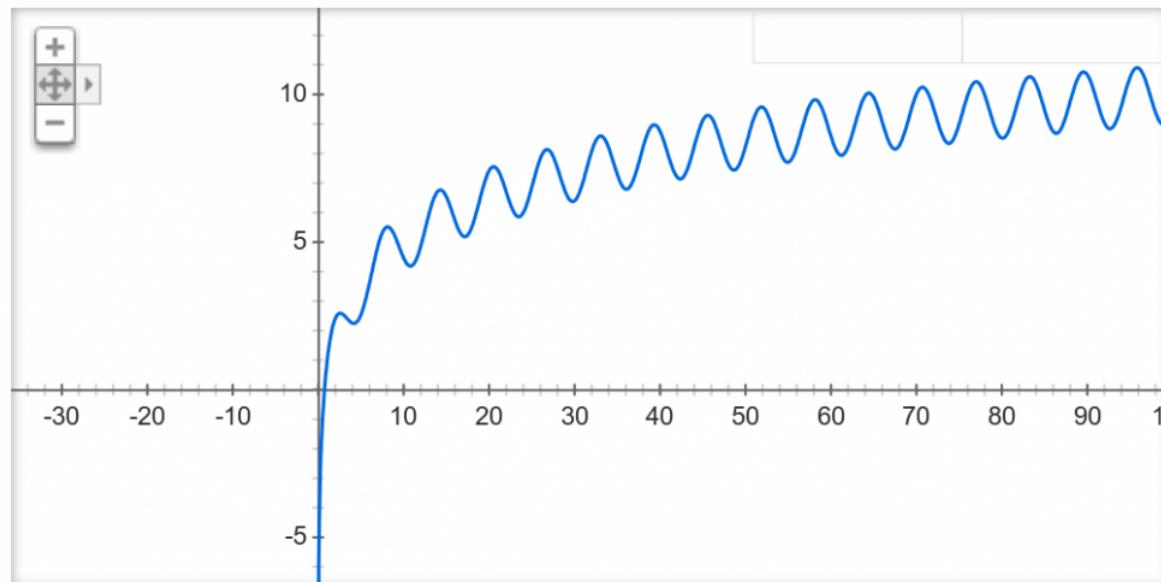
বাইসেকশন কখন কাজ করবে কখন করবে না এটা বুঝতে পারা খুবই গুরুত্বপূর্ণ। বর্গমূল বের করা শেখার পর তুমি হ্যাতো মনে করলে

$\sin(x) + 5 * \log_{10}(x) = 5.27$ ইকুয়েশনটার সমাধান বাইসেকশন দিয়ে করবে। তুমি অনুমান করলে $x=50$, তাহলে

$\sin(50) + 5 * \log_{10}(50) = 9.26$ । এখন কি তুমি নিশ্চিত ভাবে বলতে পারো যে উত্তর 50 এর বামে আছে? পারবে না, কারণ x এর মান কমলে

$\sin(x) + 5 * \log_{10}(x)$ এর মান বাড়তেও পারে, কমতেও পারে। x এর বিভিন্ন মানের জন্য গুগলে গ্রাফটা প্লট করলে তুমি এরকম পাবে:

Graph for $\sin(x) + 5 * \log_{10}(x)$



আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিনেটরিঙ্গ, ডিসিশন প্রবলেম)

এই ইকুয়েশনের ক্ষেত্রে তুমি একটা বেঞ্জের মাঝখানের মান টা নিয়ে কিছু হিসাব-নিকাশ করে বাম বা ডান দিকে অংশ বাতিল করে দিতে পারো না, কারণ তোমার জানার উপায় নাই কোনদিকে তোমার উত্তরটা আছে। বর্গমূল করার সময় ইকুয়েশনটা ছিল x^2 , সেটাকে প্লট করলে পাওয়া যায় এরকম:

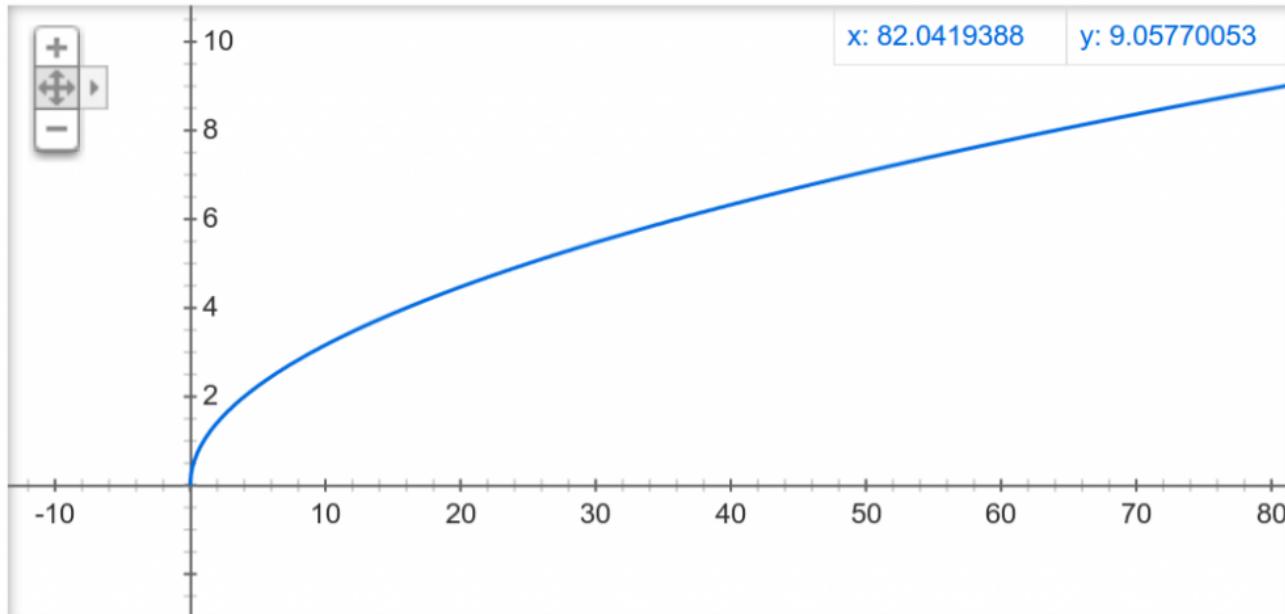
[পুরানো সিরিজ]

বিটমান্স ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

Graph for \sqrt{x}



এক্ষেত্রে বাইসেকশন কাজ করেছে কারণ তুমি সহজেই বাম বা ডানের অংশ ফেলে দিতে পেরেছো।

প্রবলেম সলভ করার সময় প্লট বের করা কোনো দরকার নেই, এটা শুধু মাত্র তোমাকে বুঝাতে দেখিয়েছি। তুমি নিশ্চিত হতে পারো যে মাঝখানের মানটা দেখে কোনো একটা অংশ বাতিল করে দেয়া যাবে শুধুমাত্র তখনই বাইসেকশন কাজ করবে। যেমন $\log(x)+x^2=y$ এই ইকুয়েশন বাইসেকশন দিয়ে সমাধান করতে পারবে কারণ x এর মান বাড়ার সাথে সাথে y সবসময় বাড়বে, কিন্তু $\tan(x)+x^2=y$ এটা বাইসেকশন দিয়ে সমাধান করতে পারবে না।

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

এবার বাইসেকশন দিয়ে একটা **গ্রাফের** প্রবলেম সমাধান করি। তোমাকে একটা গ্রাফ দেয়া আছে এরকম:

প্রতিটি এজ একেকটা বাস্তা যার একটা করে ওয়েট আছে। এখন তুমি A থেকে G তে যেতে চাও এমন ভাবে যেন সেই পথে সর্বোচ্চ ওয়েট এর মান যতটা সম্ভব কম হয়। যেমন A->D->F->G পথে সর্বোচ্চ ওয়েট ১১, আবার A->B->E->G পথে সর্বোচ্চ ওয়েট ৯।

এটা সহজেই বাইসেকশন দিয়ে সমাধান করা সম্ভব। এক্ষেত্রে শুরুতে $high=15$ কারণ কোনো এজের ওয়েট ১৫ এর থেকে বেশি না, $low=0$ আর $mid=7$ । এখন তুমি ৭ এর বড় সবগুলো এজকে গ্রাফ থেকে বাদ দিয়ে দাও আর দেখো যে A থেকে G তে যাওয়ার কোনো বাস্তা আছে নাকি। যদি না থাকে তাহলে ৭ এর বামে উত্তর থাকা সম্ভব না, বামের অংশ বাতিল করে আবার খুজতে থাকো। এক্ষেত্রে বাইসেকশন কাজ করবে কারণ মাঝের মান টা দেখে তুমি বাম বা ডানের অংশ বাতিল করে দিতে পারছো। (এই সমস্যাটা বাইসেকশনের বদলে **ডায়াক্রস্ট্র্যু** অ্যালগরিদমকে কিছুটা পরিবর্তন করেও সমাধান করা সম্ভব।)

অনুশীলনের জন্য সমস্যা:

http://www.lightoj.com/volume_showproblem.php?problem=1043

http://www.lightoj.com/volume_showproblem.php?problem=1076

https://uva.onlinejudge.org/index.php?option=com_onlinejudge&Itemid=35&category=116&page=show_problem&problem=989

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet
2.4K likes
শাফায়েত আশরাফ
Like Page

Be the first of your friends to like this

Shafaetsplanet 59 minutes ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিঙের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হয়।

*i, W) = 1 if W =
i + 1, W - C[i]) -*

SHAFAEETSPALNET.COM
ডাইনামিক প্রোগ্রামিং ৬ / স্যার

ফেসবুকে মন্তব্য

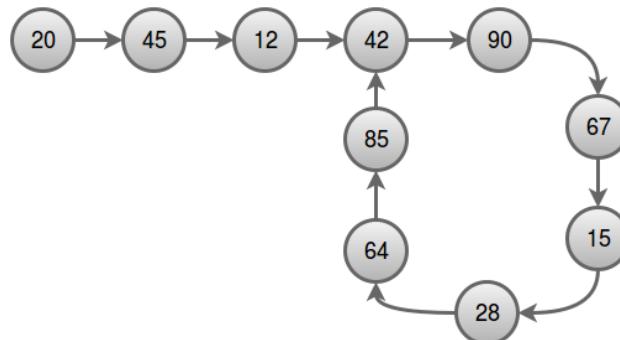
ফুয়েডের সাইকেল ফাইন্ডিং অ্যালগরিদম

আগস্ট ১৩, ২০১৬ by শাফায়েত



তোমাকে একটা [লিংকড লিস্ট](#) দেয়া আছে, বলতে হবে লিংকড লিস্টে কোনো সাইকেল আছে নাকি। এটা খুবই কমন একটা ইন্টারভিউ প্রশ্ন, আমরা ফুয়েডের সাইকেল ফাইন্ডিং অ্যালগরিদম দিয়ে এই সমস্যাটা সমাধান করা শিখবো।

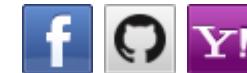
(If you want to read the same article in english, go to my [english blog](#))



ছবির [লিংকড লিস্টে](#) দেখা যাচ্ছে ৭ দৈর্ঘ্যের একটা সাইকেল আছে।

সাইকেল ডিটেক্ট করার সবথেকে সহজ উপায় হলো ডিকশনারি বা হ্যাশম্যাপ ব্যবহার করা। প্রথম নোড থেকে এক এক ঘর আগাতে হবে এবং প্রতিটা নোডকে ডিকশনারিতে সেভ করে রাখতে হবে। যদি কোনো নোডে গিয়ে দেখা যায় নোডটা আগে থেকেই ডিকশনারিতে আছে তাহলে বুঝতে হবে লিংকড লিস্টটা সাইক্লিক। এই অ্যালগরিদমের টাইম কমপ্লেক্সিটি আর মেমরি কমপ্লেক্সিটি দুইটাই $O(n)$ ।

সাবস্ক্রাইব

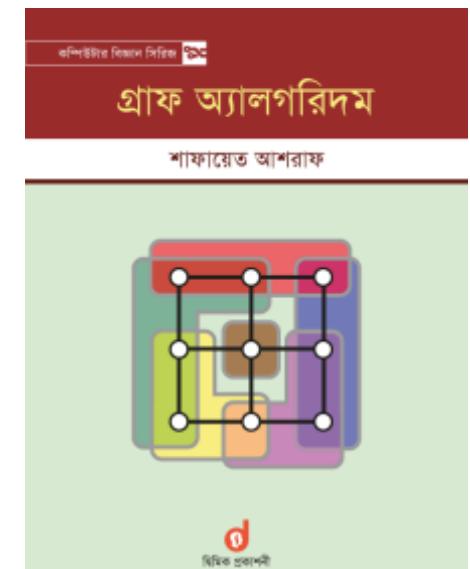


Secured by [OneAll Social Login](#)

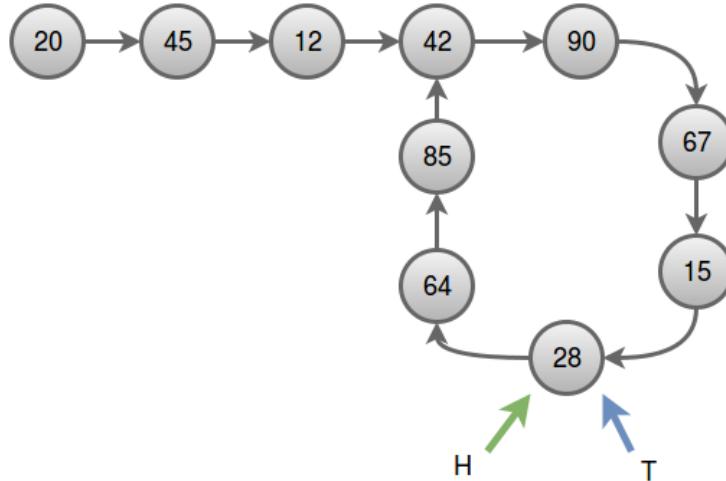
আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

Traveloka Singapore (বিস্তারিত...)



আরো ১ সেকেন্ড পরে T থাকবে 12 নম্বর নোডে, H থাকবে 90 নম্বর নোডে। এভাবে কয়েক ধাপ হাতে-কলমে সিমুলেট করলে দেখবে দুইটি পয়েন্টারই 28 নম্বর নোডে মিলিত হয়েছে।

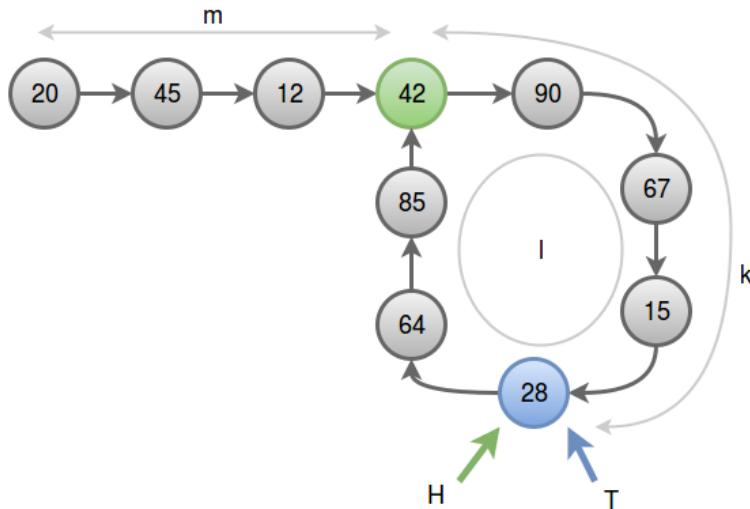


দুটি পয়েন্টার একই নোডে মিলিত হওয়ার মানে হলো লিংকড-লিস্টে অবশ্যই সাইকেল আছে। সাইকেল না থাকলে H পয়েন্টারটি সামনে আগাতে আগাতে লিংকড লিস্টের শেষ মাথায় চলে যেত।

এখন কিভাবে আমরা সাইকেলের প্রথম নোডটা খুজে বের করবো?

স্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)
ডিসজয়েট সেট(ইউনিয়ন ফাইল)
ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)
সেগমেন্ট ট্রি-১
সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)
অ্যারে কমপ্রেশন/ম্যাপিং
লোয়েস্ট কমন অ্যানসেক্টর
বাইনারি ইনডেক্সড ট্রি
স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:
এল-আর-ইউ ক্যাশ(**নতুন**)
ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):
গ্রাফ থিওরিতে হাতেখড়ি
অ্যাডজেসেন্সি ম্যাট্রিক্স
অ্যাডজেসেন্সি লিস্ট
ব্রেথড ফার্স্ট সার্চ (বিএফএস)
মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)
মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)
টপোলজিকাল সর্ট
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট
ডায়াক্রস্ট্রা
ফ্লয়েড ওয়ার্শল
বেলম্যান ফোর্ড



মনে করো,

m = রুট নোড থেকে সাইকেলের প্রথম নোডের দূরত্ব

k = সাইকেলের প্রথম নোড থেকে খরগোশ ও কচ্ছপের মিটিং পয়েন্টের দূরত্ব

l = সাইকেলের দৈর্ঘ্য

এখন যদি কচ্ছপ মোট c_T বার সাইকেলে চক্কর খেয়ে খরগোশের সাথে মিলিত হয় তাহলে কচ্ছপের অতিক্রম করা মোট দূরত্ব হবে:

$$D_T = m + c_T * l + k$$

খরগোশ যদি c_H বার সাইকেলে চক্কর খেয়ে কচ্ছপের সাথে মিলিত হয় তাহলে খরগোশের অতিক্রম করা মোট দূরত্ব হবে:

$$D_H = m + c_H * l + k$$

যেহেতু খরগোশের গতি কচ্ছপের দ্বিগুণ তাই আমরা বলতে পারি:

$$2 * (m + c_T * l + k) = m + c_H * l + k$$

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পানেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

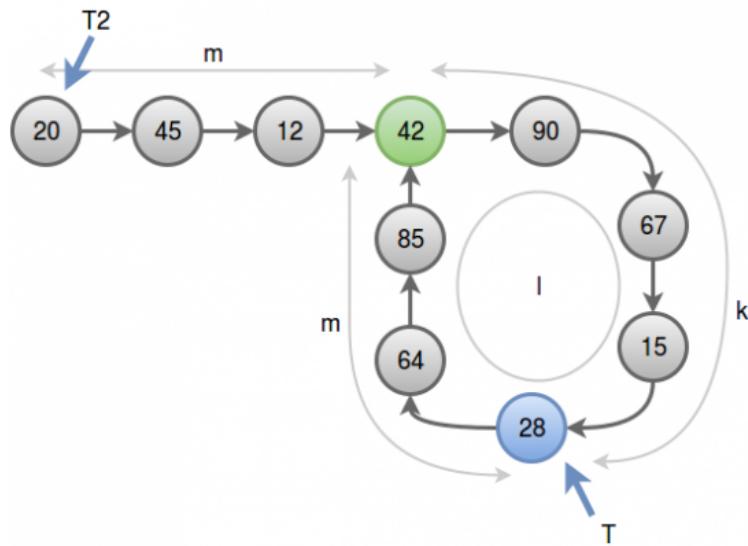
ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিনেটরিঙ, ডিসিশন প্রবলেম)

এটাকে একটু ঘুরিয়ে লেখা যায়:

$$m + k = (c_H - 2 * c_T) * l$$

l হলো সাইকেলের দৈর্ঘ্য, তারমানে $m + k$ হলো সাইকেলের দৈর্ঘ্যের একটা গুণিতক। আর সেটার মানে হলো যদি তুমি সাইকেলের প্রথম নোড থেকে $m + k$ দৈর্ঘ্য অতিক্রম করো তাহলে তুমি আবার প্রথম নোডে ফিরে আসবে। এটা বোঝাই এই অ্যালগোরিদমের সবথেকে গুরুত্বপূর্ণ অংশ।

এখন যদি তুমি মিটিং পয়েন্ট থেকে m ঘর সামনে যাও তাহলেই তুমি সাইকেলের প্রথম নোডে আবার ফিরে আসবে, কারণ প্রথম নোড থেকে মিটিং পয়েন্টের দূরত্ব k । কিন্তু তুমি m বা k কারো মান ই জানো না, তাহলে কিভাবে m ঘর সামনে যাবে? সেটার জন্য খুবই সহজ আর মজার একটা উপায় আছে। তোমার নিচ্ছয়াই মনে আছে যে কুট নোড থেকে সাইকেলের প্রথম নোডের দূরত্বও m ।



মনে করো খরগোশ এখন আব নেই, কিন্তু কুট নোড এ নতুন একটা কচ্ছপ পয়েন্টার $T2$ হাজির হয়েছে, আর T' সেই আগের মিটিং পয়েন্টেই আছে। এখন দুটি পয়েন্টারকেই এক ঘর করে আগাতে থাকলে তারা যেখানে মিলিত হবে সেটাই সাইকেলের প্রথম নোড!

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাকিং(১):

ব্যকট্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

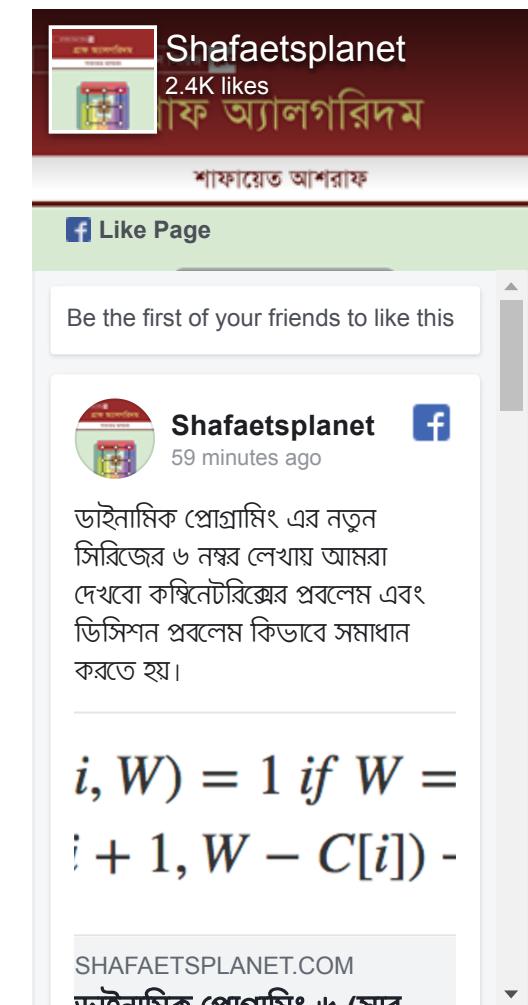
এটাই হলো ফুয়েডের সাইকেল ডিটেকশন অ্যালগোরিদম। টাইম কমপ্লেক্সিটি এখনও $O(n)$ ই আছে ([কেন?](#)) কিন্তু মেমরি কমপ্লেক্সিটি হয়ে গেছে $O(1)$ ।

একটি সি++ কোড দেখি:

```
1 /**
2  * Definition for singly-linked list.
3  * struct ListNode {
4  *     int val;
5  *     ListNode *next;
6  *     ListNode(int x) : val(x), next(NULL) {}
7  * };
8 */
9 ListNode* Solution::detectCycle(ListNode* A) {
10     if(!A->next) return NULL;
11
12     ListNode *tortoise=A;
13     ListNode *hare=A;
14
15     //check if there is a cycle
16     while(hare){
17         if(hare->next and hare->next->next)
18             hare=hare->next->next;
19         else
20             return NULL; //no cycle
21         tortoise=tortoise->next;
22         if(hare==tortoise)break; //cycle exists
23     }
24
25     ListNode* tortoise2 = A;
26     while(tortoise2 != tortoise){
27         tortoise2 = tortoise2->next;
28         tortoise = tortoise->next;
29     }
30     return tortoise;
31 }
```

লিংকড লিস্টে সাইকেল ডিটেক্ট করা ছাড়াও এই অ্যালগোরিদম অনেক কাজে লাগে। যেমন কোনো গাণিতিক ফাংশন বা pseudo-random নাম্বার জেনারেটরের সাইকেল ডিটেক্ট করা।

কোয়ান্টাম কম্পিউটার কী?
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Shafaetsplanet
2.4K likes
শাফায়েত আশরাফ
[Like Page](#)

Be the first of your friends to like this

Shafaetsplanet 59 minutes ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটারের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

$i, W) = 1 \text{ if } W =$
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / মার

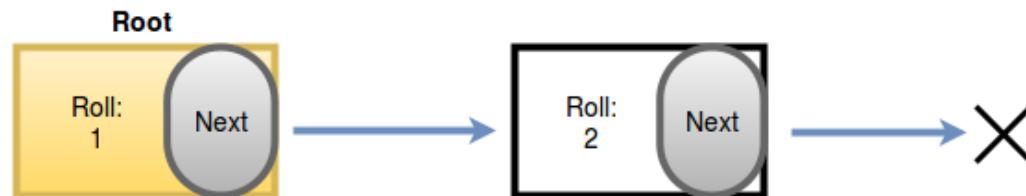
ডাটা স্ট্রাকচার : লিংকড লিস্ট

জানুয়ারি ৩০, ২০১৬ by শাফায়েত



লিংকড লিস্ট বেসিক একটা **ডাটা স্ট্রাকচার**। আমরা সাধারণত তথ্য বাখার জন্য অ্যাবে ব্যবহার করি, তবে অ্যাবের কিছু সীমাবদ্ধতা আছে যে কারণে অনেক সময় লিংকড লিস্ট ব্যবহারের দরকার হয়। লিংকড লিস্ট নিয়ে জানতে হলে অবশ্যই পয়েন্টার সম্পর্কে ধারণা থাকতে হবে।

লিংক লিস্টের প্রতিটা এলিমেন্ট কে বলবো আমরা নোড। প্রতিটা নোডে সাধারণত দুইটা তথ্য থাকে: ১) যে তথ্যটা আমরা সংরক্ষণ করতে চাচ্ছি ২) পরবর্তি তথ্যটা কোথায় আছে তার ঠিকানা।



ছবিতে দেখা যাচ্ছে প্রথম নোড এ একজন ছাত্রের বোল নম্বর লেখা আছে, এবং পরবর্তি ছাত্রের তথ্য কোন নোড এ আছে সেটা দেখিয়ে দিচ্ছে `next` নামের একটা পয়েন্টার। দ্বিতীয় নোডটাই শেষ নোড, তাই এই নোডের নেক্সট পয়েন্টার একটা `null` নোডকে পয়েন্ট করছে। প্রথম নোডকে আমরা বলবো রুট নোড।

সাবস্ক্রাইব



Secured by OneAll Social Login

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

Traveloka Singapore (বিস্তারিত...)



অ্যারের সাথে লিংক লিস্টের একটা বড় পার্থক্য হলো অ্যারের তথ্যগুলো মেমরিতে পরপর সংরক্ষণ করা হয়। যদি অ্যারেটা একটা 4 বাইটের ইন্টজার অ্যারে হয় এবং অ্যারের প্রথম এলিমেন্টটা যদি থাকে x তম মেমরি সেল এ, তাহলে পরের তিনি এলিমেন্ট $x+4, x+8, x+12$ মেমরি সেল এ থাকবে। নিচের কোডটা রান করলেই প্রমাণ পাবে।

```

1 int main(){
2     int a[5];
3     for(int i=0;i<5;i++)
4     {
5         printf("%u\n",&(a[i])); #print address of each element
6     }
7     return 0;
8 }
```

সেজন্য অ্যারের প্রথম এলিমেন্টের অ্যাড্রেস জানলেই এবপর যেকোনো এলিমেন্টের অ্যাড্রেস সহজেই বের করে ফেলা যায়, ইন্টজার অ্যারের p তম এলিমেন্ট থাকে $x + p * 4$ অ্যাড্রেসে যেখানে x হলো শূন্যতম এলিমেন্টের অ্যাড্রেস।

লিংকড লিস্টে তথ্যগুলো থাকে ছড়িয়ে-ছিটিয়ে, তাই প্রতিটা এলিমেন্টকে পরের এলিমেন্টের ঠিকানা সংরক্ষণ করে রাখতে হয়। এই পদ্ধতির কিছু সুবিধাও আছে, অসুবিধাও আছে, সেগুলো আমরা দেখবো।

একটা সি তে লিংকড লিস্ট তৈরির জন্য শুরুতেই একটা স্ট্রাকচার ডিফাইন করতে হবে, যেখানে থাকবে যে তথ্য সংরক্ষণ করতে চাই সেটা এবং পরবর্তী নোডের অ্যাড্রেস।

```

1 nstruct node
2 {
3     int roll;
4     node *next;
5 };
6 node *root=NULL;
7 int main(){
8
9     return 0;
10 }
```

`node *next` হলো একটা পয়েন্টার যেটা একটা `node` এর অ্যাড্রেস সংরক্ষণ করে।

`node *root` হলো একটা পয়েন্টার যেটা সবসময় প্রথম নোডের অ্যাড্রেস সংরক্ষণ করবে। শুরুতে লিস্ট এ কোনো নোড নেই, তাই রুট



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৮):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

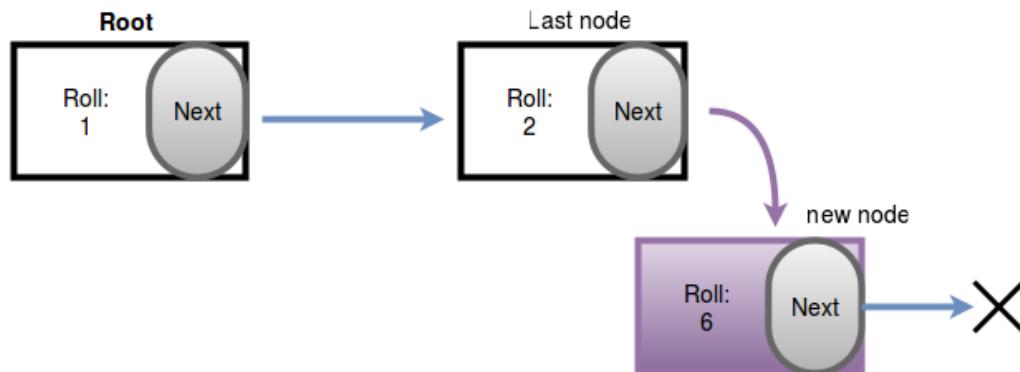
কিউ+সার্কুলার কিউ

পয়েন্টারের মান নাল(NULL)। প্রথম নোডের অ্যাড্রেস ব্যবহার করে আমরা পরবর্তীতে অন্য নোডের তথ্য পড়তে পারবো।

এখন আমাদের একটা ফাংশন দরকার যেটা ব্যবহার করে নতুন একটা নোড লিস্টে শেষে প্রবেশ করাতে পারবো। মনে করো ফাংশনটার নাম append। এই ফাংশনটা লেখার সময় ২টা কেস মাথায় রাখতে হবে। প্রথম কেস হলো যে নোডটা প্রবেশ করাচ্ছি সেটাই লিঙ্ক লিস্টের প্রথম নোড কি না। যদি তাই হয়, তাহলে রুট পয়েন্টার ব্যবহার করে প্রথম নোডটা তৈরি করতে হবে।

```
1 void append(int roll)
2 {
3     if(root==NULL) //If the list is empty
4     {
5         root=new node(); //create new node in root
6         root->roll=roll;
7         root->next=NULL;
8     }
9 }
```

যদি লিঙ্কড লিস্টে আগেই কিছু নোড থাকে তাহলে আমাদেরকে শেষ নোডটা খুজে বের করতে হবে। তারপর শেষ নোডের নেক্সট পয়েন্টার ব্যবহার করে পরবর্তী নোডটা তৈরি করতে হবে।



```
1 void append(int roll)
2 {
3     if(root==NULL) //If the list is empty
4     {
5         root=new node();
6         root->roll=roll;
```

শ্লাইডিং রেজ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

7     root->next=NULL;
8 }
9 else
10 {
11     node *current_node=root; //make a copy of root node
12     while(current_node->next!=NULL) //Find the last node
13     {
14         current_node=current_node->next; //go to next address
15     }
16     node *newnode = new node(); //create a new node
17     newnode->roll=roll;
18     newnode->next=NULL;
19
20     current_node->next=newnode; //link the last node with new node
21 }
22 }
```

আমরা প্রথমে লুপ চালিয়ে শেষ নোডটা বের করছি। শেষ নোড কোনটা বোধ খুব সহজ, যেই নোডের নেক্সট পয়েন্টার নাল সেটাই শেষ নোড।

এরপর নতুন একটা নোড তৈরি করে শেষ নোডের সাথে সেটা লিংক করে দিচ্ছি। আমাদের এই অ্যাপেন্ড ফাংশনের **কমপ্লেক্ষিটি $O(n)$** ।

লক্ষ্য করো, কুট পয়েন্টারকে আমরা সামনে নিচ্ছি না, সেটার একটা কপি তৈরি সেটাকে সামনে নিচ্ছি। কারণ কুট পয়েন্টারকে আমরা সামনে নিলে প্রথম নোডের অ্যাড্রেস হারিয়ে ফেলবো!

সবগুলো ছাত্রের বোল নম্বর প্রিন্ট করতে চাইলেও একইভাবে করতে পারবো। আগের মতই লুপ চালিয়ে শেষ নোড পর্যন্ত যাবো এবং সবগুলো মান প্রিন্ট করবো।

```

1 void print()
2 {
3     node *current_node=root;
4     while(current_node!=NULL) //loop until you reach null
5     {
6         printf("%d\n",current_node->roll);
7         current_node=current_node->next;
8     }
9 }
10 int main(){
11     append(1);
12     append(2);
```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এবং ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

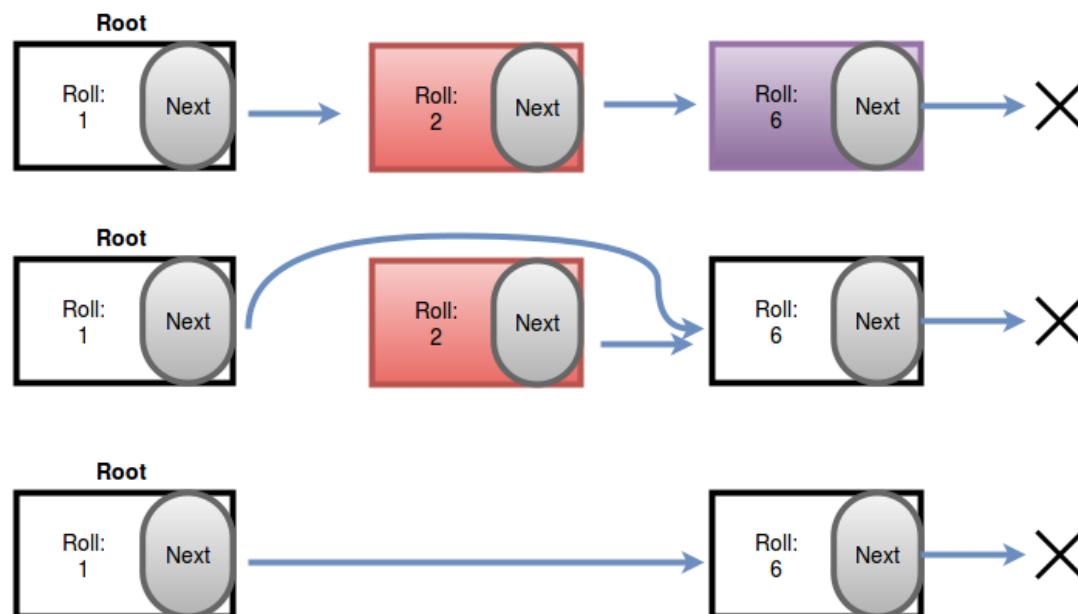
```

14     append(6);
15     print();
16     return 0;
17 }

```

এখন তুমি যদি চাও শুধুমাত্র 10 তম ছাত্রের রোল প্রিন্ট করতে, তাহলে কি করবে? তোমাকে লুপ চালিয়ে 10 নম্বর নোড খুজে বের করে প্রিন্ট করতে হবে। কিন্তু অ্যারেতে আমরা $roll[10]$ লিখেই 10তম ছাত্রের রোল প্রিন্ট করে ফেলতে পারতাম। লিংকড লিস্টে তথ্যগুলো মেমরিতে পরপর সাজান্ত নেই তাই রেভেন্ম এক্সেস করা যায় না। লিংকড লিস্টে কোনো ইনডেক্স খুজে বের করার কমপ্লেক্সিটি তাই $O(n)$, যেখানে অ্যারেতে $O(1)$ । [পুরানো আমলের গানশোনার ফিতার ক্যাসেটগুলোর কথা মনে আছে? সেখানেও কোনো গানে লাফ দিয়ে চলে যাওয়া যেত না, ফিতা ঘুরিয়ে খুজে বের করতে হতো। এখানেও একই ব্যাপার ঘটছে!]

লিংক লিস্ট এর সুবিধা হলো চাইলেও কোনো তথ্য মাঝখান থেকে মুছে ফেলা যায়। অ্যারেতে তুমি চাইলেই মাঝখান থেকে একটা ইনডেক্স মুছে ফেলতে পারবে না, মুছতে হলে ডানের সব এলিমেন্টকে একঘর বামে টেনে এনে ফাকা জায়গা পূরণ করতে হবে, এবং সবার শেষের এলিমেন্টটাকে মুছে ফেলতে হবে। কিন্তু লিংকড লিস্ট তুমি সহজেই মাঝখান থেকে একটা নোড মুছে ফেলতে পারবে।



[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাষ্ঠার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেক্ষেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেক্ষন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

ছবিতে বোল ২ কে কিভাবে মুছে ফেলা যায় দেখানো হয়েছে। বোল ২ এর আগের নোড বোল ১ এর পয়েন্টারকে দিয়ে বোল ২ এর পরের নোড এর অ্যাড্রেস কে পয়েন্ট করানো হয়েছে, এবং মাঝের নোডটা মেমরি থেকে মুছে ফেলা হয়েছে।

লক্ষ্য করো, কট নোডের আগে কোনো নোড নেই। তাই কট নোড মুছে ফেলা আরো সহজ, শুধুমাত্র কট পয়েন্টার এক ঘর এগিয়ে দিতে হবে এবং আগের নোডটা মুছে ফেলতে হবে।

```
1 void delete_node(int roll)
2 {
3     node *current_node=root;
4     node *previous_node=NULL;
5     while(current_node->roll!=roll) //Searching node
6     {
7         previous_node=current_node; //Save the previous node
8         current_node=current_node->next;
9     }
10    if(current_node==root) //Delete root
11    {
12        node *temp=root; //save root in temporary variable
13        root=root->next; //move root forward
14        delete(temp); //free memory
15    }
16    else //delete non-root node
17    {
18        previous_node->next=current_node->next; //previous node points the current node's next node
19        delete(current_node); //free current node
20    }
21 }
22 }
```

উপরের কোডে প্রথমে আমরা খুজে বের করেছি যে বোল নম্বরটা মুছতে হবে সেই নোডটাকে। যদি সেটাই কট নোড হয় তাহলে কটকে একঘর এগিয়ে দিয়েছি, নাহলে উপরের ছবির মত করে মুছেছি।

লক্ষ্য করো, আমি `delete(node)` নামের একটা লাইব্রেরি ফাংশন ব্যবহার করেছি। মোছার সময় পয়েন্টার ঠিকঠাক করার পর অবশ্যই `delete` ফাংশন ব্যবহার করে মেমরি ফ্রি করে দিতে হবে, নাহলে লিংকড লিস্ট থেকে নোড মুছে গেলেও নোডটা মেমরিতে থেকে যাবে, অন্য কোনো প্রোগ্রাম সেটাকে ব্যবহার করতে পারবে না। লিংকড লিস্টের কোড লেখার সময় `delete()` ফাংশন ব্যবহার করতে ভুলে যাওয়া খুবই কমন একটা ভুল।

কোয়ান্টাম কম্পিউটার কী?

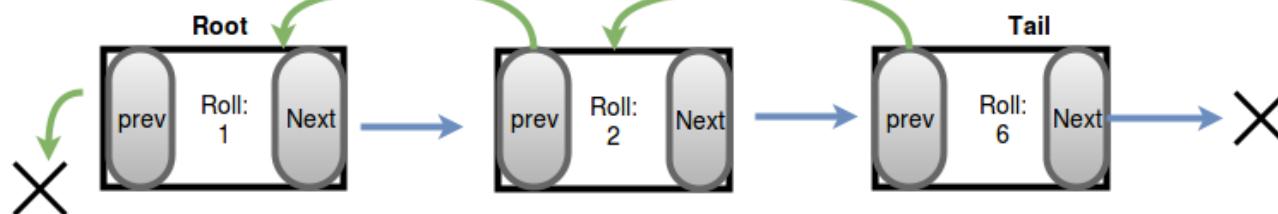
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

The image shows a Facebook post from a page named 'Shafaetsplanet'. The post has 2.4K likes and is titled 'কাফ অ্যালগরিদম' (Kaf Algoritam). It includes a link to 'Like Page'. The post content is: 'Be the first of your friends to like this'. Below the post, there is a comment from the page itself: 'ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটারের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।' A reply to this comment from a user 'SHAFAEETSPALNET.COM' says: 'i, W) = 1 if W = i + 1, W - C[i]) -'

ଲିଂକଡ ଲିସ୍ଟ ଏ ତୁମି ଚାହିଁଲେ ମାଘିଥାନେଓ ନୋଡ ଯୋଗ କରତେ ପାରବେ । ଏହି ପର୍ଯ୍ୟନ୍ତ ବୁଝେ ଥାକଲେ ତୋମାର କାଜ ହବେ ଦୁଇ ନୋଡ ଏବଂ ମାଧ୍ୟମ ନତୁନ ନୋଡ ଯୋଗ କରାବ ଜଣ୍ୟ ଫାଂଶନ ଲେଖା । ଏଟା ଅନେକଟା `delete-node` ଫାଂଶନର ମତ କରେ ଲିଖିତେ ହବେ । ଫାଂଶନର ପ୍ରୟାରମ୍ଭିଟାର ହିସାବେ ନିବେ `roll1`, `roll2`, ତୋମାର ଫାଂଶନର କାଜ ହବେ `roll1` ଯେ ନୋଡେ ଆଛେ ସେଟା ଖୁଜେ ବେର କରେ ସେଟାର ପରେ `roll2` ନୋଡ଼ଟା ଯୋଗ କରା ।

বাইডিরেকশনাল লিংকড লিস্ট

আমাদের আগের কোড এ অ্যাপেন্ড অপারেশন ছিলো $O(n)$, লুপ চালিয়ে বারবার শেষ পর্যন্ত যেতে হচ্ছিলো। এছাড়া লিংকড লিস্টটা উল্টো দিব থেকে ট্রাভার্স করা সম্ভব হচ্ছিলো না।



উপরের ছবির লিংক লিস্টে প্রতি নোডে দুইটা পয়েন্টার ব্যবহার করা হয়েছে। `prev` পয়েন্টারটা প্রতিটা নোডের আগের নোডের অ্যাড্রেসকে পয়েন্ট করে আছে। এছাড়াও এআমরা কটা `tail` পয়েন্টার এর সাহায্যে শেষ নোডটার অ্যাড্রেস মনে রাখছি।

এখন লিংকড লিস্টের শেষে নতুন নোড যোগ করা সম্ভব $O(1)$ কমপ্লেক্সিটিতে।

```
1 struct node  
2 {  
3     int roll;  
4     node *next, *prev;  
5 };  
6 node *root, *tail;
```

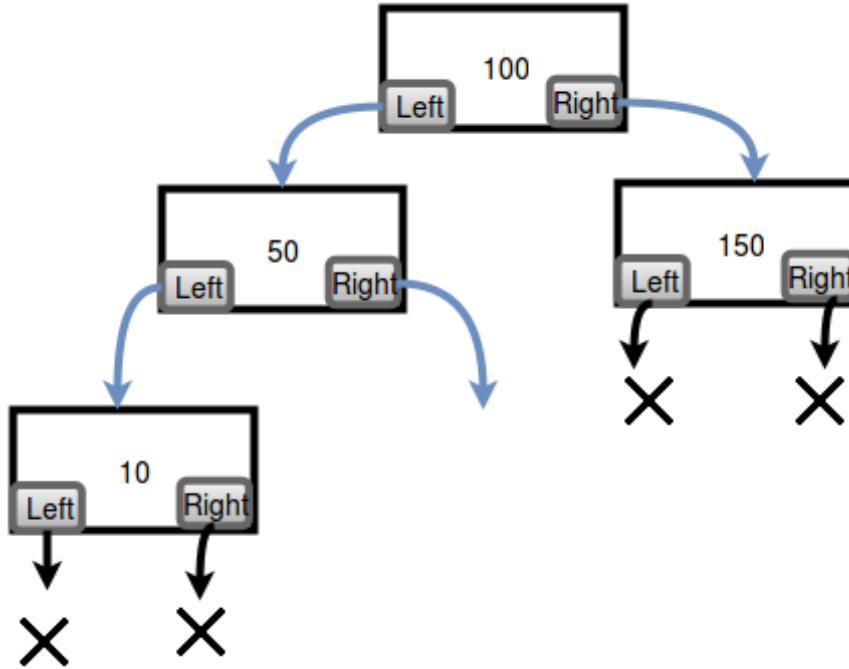
```

7 void append(int roll)
8 {
9     if(root==NULL) //If the list is empty
10    {
11        root=new node();
12        root->roll=roll;
13        root->next=NULL;
14        tail=root;
15    }
16    else
17    {
18        node *newnode=new node();
19        newnode->roll=roll;
20        newnode->next=NULL;
21        tail->next=newnode; //add the new node after tail node
22        tail=tail->next; //move tail pointer forward
23    }
24 }
```

এবার append ফাংশন খুব সহজ হয়ে গিয়েছে। নতুন নোডটা tail এর সাথে যোগ করে টেইল এক ঘর এগিয়ে নেয়া হচ্ছে। নোড ডিলিট করার সময়েও এখন আর previous_node ড্যারিয়েবলটা রাখা দরকার নেই।

বাইনারি সার্চ ট্ৰি

লিংকড লিস্টের আৱেকটা ব্যবহার হলো বিভিন্ন ধৰণের ট্ৰি তৈরি কৰা। যেমন নিচে একটা বাইনারি সার্চ ট্ৰি তৈরি কৰা হয়েছে:



বাইনারি সার্চ ট্রি তে বাম পাশের নোডে সবসময় ছোটো মান, ডানের নোডে সমান বা বড় মান থাকে। আমাদের left এবং right নামের দুইটা পয়েন্টার লাগবে।

```

1 struct node
2 {
3     int roll;
4     node *left, *right;
5     node() //initialize the node using null
6     {
7         left=NULL;
8         right=NULL;
9     }
10 };
11 node *root;
  
```

নতুন নোড ইনসার্ট করার সময় আগের মতই লুপ চালিয়ে শেষ নোডে আসতে হবে। লুপ চালানোর সময় বামে নাকি ডানে যাবে সেটা নোডের মান এবং নতুন মান তুলনা করে বের করতে হবে।

```

1 void insert(int roll)
2 {
3     if(root==NULL) //first node in tree
4     {
5         root=new node();
6         root->roll=roll;
7     }
8     else
9     {
10        node *current=root,*parent;
11        while(current!=NULL)
12        {
13            if(roll<current->roll)
14            {
15                parent=current; //keep track of parent node
16                current=current->left;
17            }
18            else
19            {
20                parent=current;
21                current=current->right;
22            }
23        }
24        node *newnode=new node();
25        newnode->roll=roll;
26        if(newnode->roll<parent->roll) parent->left=newnode;
27        else parent->right=newnode;
28    }
29 }

```

কোড ঠিক আছে নাকি বোঝার জন্য একটা প্রিন্ট ফাংশন লিখি:

```

1 void print_preorder(node *current)
2 {
3     if(current==NULL) return;
4     cout<<current->roll<<endl;
5     print_preorder(current->left);
6     print_preorder(current->right);
7 }
8

```

এই ফাংশনটা রিকার্সিভলি নোডগুলার মান প্রিন্ট করবে।

কোডটা যদি বুঝে থাকো তাহলে বাইনারি সার্চ ট্রি থেকে নোড মুছে ফেলার ফাংশনটা লিখে ফেলো। নোড মোছার সময় বেশ কয়েকটা কেস চিন্তা করতে হয়, সেটা নিয়ে এখানে আলোচনা করবো না। তুমি কোরম্যানের অ্যালগোরিদম বই থেকে বা গুগলে একটু সার্চ করে শিখে নিতে পারো।

লিংকড লিস্টে সাইকেল কিভাবে বের করতে হয় জানতে আমার [এই লেখাটা পড়ো](#)। লিংকড লিস্টের একটা খুবই সুন্দর প্রয়োগ হলো **LRU ক্যাষ**, বিস্তারিত জানতে এই লেখাটা পড়ো।

লিংকড লিস্টের কোড লেখার সময় কিছু কমন ভুল হয় শুরুর দিকে। যেমন পয়েন্টারের মান নাল হয়ে যাবার পরেও মান প্রিন্ট করার চেষ্টা করে বা আরো সামনে আগানোর চেষ্টা করা, সেক্ষেত্রে কোড রান টাইম ইরোর দিবে। এছাড়া মেমরি ফ্রি করতে ভুলে যাওয়াও খুব সাধারণ একটা ভুল। আরেকটা ভুল হলো রুট বা টেইল পয়েন্টারের মান বদলে ফেলা।

লিংকড লিস্ট শেখার পর স্ট্যাক, কিউ, বাইনারি ট্রি, হিপ ইত্যাদি ডাটা স্ট্রাকচারগুলো লিংকড লিস্ট দিয়ে ইমপ্লিমেন্ট করা শিখতে হবে। তুমি যদি কনস্টেন্ট করো আর আরেকটু অ্যাডভান্সড কিছু শিখতে চাও তাহলে এখান থেকে [ট্রাই ডাটা-স্ট্রাকচার](#) কিভাবে লিংকড লিস্ট ব্যবহার করে ইমপ্লিমেন্ট করে শিখে নিতে পারো।

হ্যাপি কোডিং!

ফেসবুকে মন্তব্য

7 comments

ডাটা স্ট্রাকচার : স্ট্যাক

নভেম্বর ১১, ২০১৫ by শাফায়েত



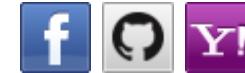
যেকোনো ডাটা স্ট্রাকচার কোর্সে একদম শুরুর দিকে যেসব ডাটা স্ট্রাকচার পড়ানো হয় তার মধ্যে স্ট্যাক অন্যতম। স্ট্যাককে বলা হয় **LIFO** বা লাস্ট-ইন-ফার্স্ট-আউট ডাটা স্ট্রাকচার। তুমি এভাবে চিন্তা করতে পারো, তোমার কাছে অনেকগুলো বই একটার উপর আরেকটা সাজানো আছে, তুমি চাইলে সবার উপরের বইটা সরিয়ে ফেলতে পারো(Pop), অথবা সবার উপরে আরেকটা বই রাখতে পারো(Push)। এটা হলো বইয়ের একটা স্ট্যাক। তুমি এই স্ট্যাকের উপরে ছাড়া কোনো জায়গায় বই ঢুকাতে পারবে না, উপরের বই ছাড়া কোনো বই সরাতে পারবে না, এগুলো করলে সেটা আর স্ট্যাক থাকবে না।

স্ট্যাক হলো কিছু বস্তু(এলিমেন্ট) একটা সংগ্রহ। এখানে দুইরকম অপারেশন করা যায়:

Push(new_element): স্ট্যাকের উপরে নতুন বস্তু রাখো। যদি শুরুতে স্ট্যাকটা খালি হয়, তাহলে প্রথম জায়গায় বস্তু রাখতে হবে।

Pop(): স্ট্যাকের উপরের বস্তু সরিয়ে ফেলো। যদি শুরুতে স্ট্যাকটা খালি হয় তাহলে এই অপারেশনটা করা সম্ভব না।

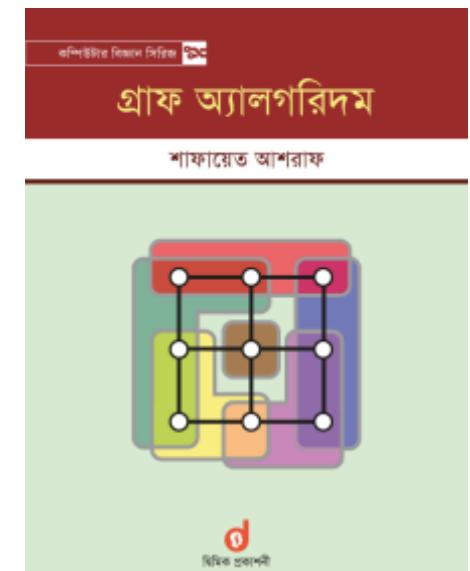
সাবস্ক্রাইব

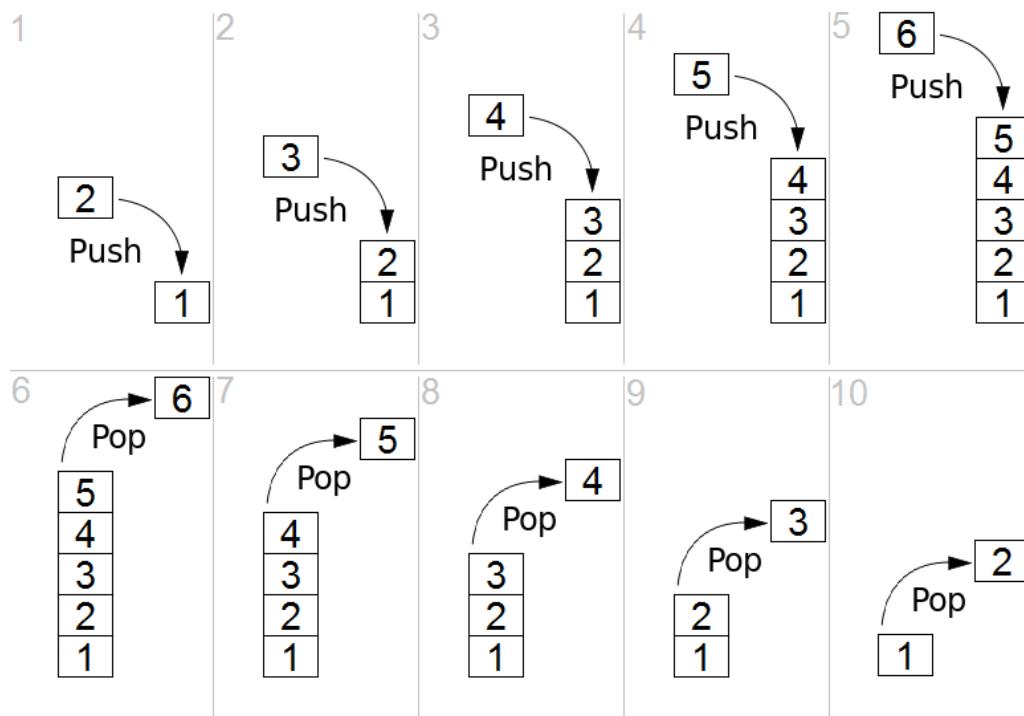


Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)





চিত্র ১: স্ট্যাক এ পুশ এবং পপ অপারেশন (সোর্স: উইকিপিডিয়া)

চিত্র ১ এ দেখা যাচ্ছে কিভাবে স্ট্যাকে পুশ এবং পপ অপারেশন করতে হয়। স্ট্যাকে আরেকটা অপারেশন থাকতে পারে Peek(), যেটা ব্যবহার করে সবার উপরের বন্ধটা কি জানা যাবে বন্ধটা পপ না করেই।

স্ট্যাকের আকার যদি সীমিত হয়, যেমন যদি ১০টার বেশি বন্ধ রাখার সামর্থ কোনো স্ট্যাকের না থাকে তাহলে ১১তম বন্ধ পুশ করলে স্ট্যাকটা “ওভারফ্লো” স্টেট এ চলে যাবে এবং তোমার প্রোগ্রাম ক্র্যাশ করবে। একই ভাবে খালি স্ট্যাক থেকে কিছু পপ করার চেষ্টা করলে স্ট্যাক ‘আন্ডারফ্লো’ স্টেটে চলে যাবে এবং প্রোগ্রাম ক্র্যাশ করবে। স্ট্যাক ব্যবহারের সময় এই দুই ব্যাপারে খুব সতর্ক থাকতে হয়।

স্ট্যাক ইম্প্রিমেন্ট করা খুব সহজ। সাধারণ অ্যারে ব্যবহার করেই নির্দিষ্ট আকারের স্ট্যাক ইম্প্রিমেন্ট করা যায়। সবার উপরের এলিমেন্টটা কোন ইনডেক্সে আছে সেটা একটা অতিরিক্ত ড্যারিয়েবলে সেভ করে রাখতে হবে। নিচে পাইথনে একটা ইম্প্রিমেন্টেশন দেখানো হলো, একই ভাবে সি++



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৮):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

এও তুমি করতে পারবে:

```
1 class Stack:
2     def __init__(self, max_size): #initialize a stack of max_size
3         self.top_pointer = -1 #Keep track of top element using this
4         self.stack = [None for x in range(max_size)] #create a list of max_size
5
6     def push(self, new_element):
7         self.top_pointer = self.top_pointer + 1 #Move the pointer
8         self.stack[self.top_pointer] = new_element #Add the new_element to the top
9
10    def pop(self):
11        last_element = self.stack[self.top_pointer]
12        self.top_pointer = self.top_pointer - 1 #Move the pointer
13        return last_element #Pop the last element
14
15    def peek(self):
16        return self.stack[self.top_pointer]
17
18    def is_empty(self):
19        return top.pointer == -1
```

এই কোডে আমি কোনো ওভারফ্লো বা আন্ডারফ্লো হ্যান্ডেল করি নি। তুমি কোডটার উন্নতি করতে চাইলে এ ব্যাপারগুলো নিয়ে কাজ করতে পারো। যদি তুমি নির্দিষ্ট আকারের স্ট্যাক না চাও তাহলে লিংকড-লিস্ট ব্যবহার করতে হবে।

স্ট্যাকে প্রতিটা অপারেশনের কমপ্লেক্সিটি $O(1)$

স্ট্যাকের ব্যবহার:

আমরা ব্রাউজারে প্রায়ই আগের ওয়েবসাইটে ফিরে যেতে "Back" বোতামে চাপ দেই। এটাও স্ট্যাক ব্যবহার করে তৈরি করা যায়। যখন নতুন ওয়েবসাইটে যাবে তখন আগের ওয়েবসাইটটাকে স্ট্যাকের উপরে রেখে দাও, ফিরে যেতে চাইলে স্ট্যাকের উপরের ওয়েব সাইট ফেরত গিয়ে স্ট্যাক থেকে পপ করে দাও। এভাবে তুমি তোমার সফটওয়্যারে "undo" ফিচার বানাতে পারো।

প্রোগ্রামিং ল্যাংগুয়েজে ফাংশন কল করার সময় স্ট্যাকের খুব গুরুত্বপূর্ণ ব্যবহার আছে। মনে করো তুমি একটা ফাংশন `func1` থেকে `func2` কল করছো, সেখান থেকে আবার `func3` কল করছো:

স্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যাক-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

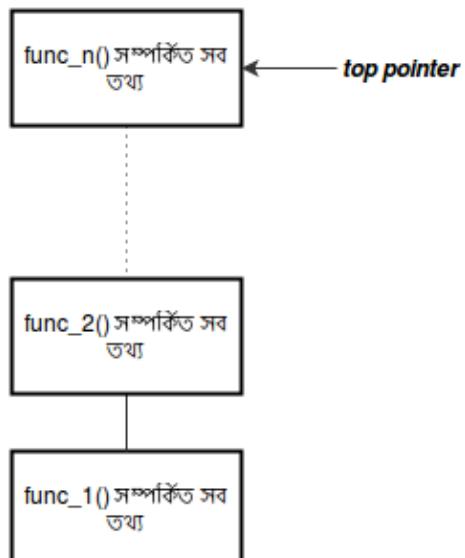
বেলম্যান ফোর্ড

```

1 def func_3():
2     return 42
3
4 def func_2():
5     x=20
6     y=func3()
7     return x+y
8
9 def func_1():
10    x=10
11    y=func2()
12    return x+y
13
14 print func1()

```

`func_1()` এ কিছু লোকাল ভ্যারিয়েবল আছে। তুমি যখন `func_1()` থেকে `func_2()` কে কল করছো তখন `func_1()` এর সব তথ্য একটা স্ট্যাকে ঢুকিয়ে রেখে নতুন ফাংশন `func_2()` কে লোড করা হয়। আবার যখন `func_3()` কে কল করছো তখন `func_2()` কে স্ট্যাকে ঢুকিয়ে রাখে হবে। যদি $n + 1$ টা ফাংশন থাকে তাহলে যখন n তম ফাংশন টা $n + 1$ তম ফাংশনকে কল করবে তখন স্ট্যাকের চেহারাটা হবে এরকম:



আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

`func_3()` এর কাজ শেষ হবার পর আবার `func2()` কে মেমরিতে লোড করা হবে এবং স্ট্যাক থেকে পপ করে দেয়া হবে। এটাকে কল-স্ট্যাক বলা হয়। রিকার্সিভ ফাংশনও এভাবে আগের স্টেটগুলোকে স্ট্যাকে ঢুকিয়ে রাখে।

হ্যাকারো এক ধরণের আক্রমণ মাঝে মাঝে ব্যবহার করে যাকে বলা হয় “স্ট্যাক স্ম্যাশিং”। আগ্রহী হলে এই [আর্টিকেলটা](#) পড়তে পারো।

স্ট্যাকের খুবই কমন একটা ব্যবহার হলো ব্রাকেট এর ব্যালেন্স বা ডারসাম্য ঠিক আছে সেটা পরীক্ষা করা (Parenthesis Balance)। মনে করো তোমাকে একটা ব্রাকেট সিকোয়েন্স S দেয়া হলো এরকম “ $((\{))$ ”। তোমাকে বলতে হবে সিকোয়েন্সটার ব্যালেন্স ঠিক আছে নাকি নেই। ব্যালেন্স ঠিক থাকবে যদি নিচের শর্ত গুলো পূরণ হয়:

যদি S একটা ০ দৈর্ঘ্যের স্ট্রিং হয়।

যদি A আর B দুইটা ব্যালেন্সড সিকোয়েন্স হয় তাহলে AB ও একটা ব্যালেন্সড সিকোয়েন্স। যেমন $GA = "(\{\})"$ এবং $B = "()"$ দুইটা ব্যালেন্সড সিকোয়েন্স হলে $AB = "(\{\}())"$ ও ব্যালেন্সড।

যদি S একটা ব্যালেন্সড সিকোয়েন্স হয় তাহলে (S) অথবা $\{S\}$ ও ব্যালেন্সড। যেমন $S = "\{\}()$ ব্যালেন্সড হলে $(S) = "(\{\}())"$ এটাও একটা ব্যালেন্সড সিকোয়েন্স।

কিছু ডারসাম্যহীন সিকোয়েন্স হলো “ $(($, $"(\{)"$, $()\()$ ” ইত্যাদি।

“ $($ ” আর “ $)$ ” কে আমরা বলবো “ওপেন ব্রাকেট”, আর “ $\}$ ” এবং “ $\}$ ” কে বলবো “ক্লোজিং ব্রাকেট”।

এখন আমরা $S = "((\{))"$ ব্যালেন্সড নাকি পরীক্ষা করবো স্ট্যাক ব্যবহার করে। আমরা বাম থেকে ডানে একটা একটা ক্যারেক্টার নিয়ে কাজ করবো। কোনো “ওপেন ব্রাকেট” পেলেই সেটাকে স্ট্যাকে পুশ করবো।

এই সিকোয়েন্সের প্রথম ক্যারেক্টার “ $($ ”, আমরা এটাকে একটা স্ট্যাকে পুশ করবো।



[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ডারটেক্স কডার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেক্ষেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

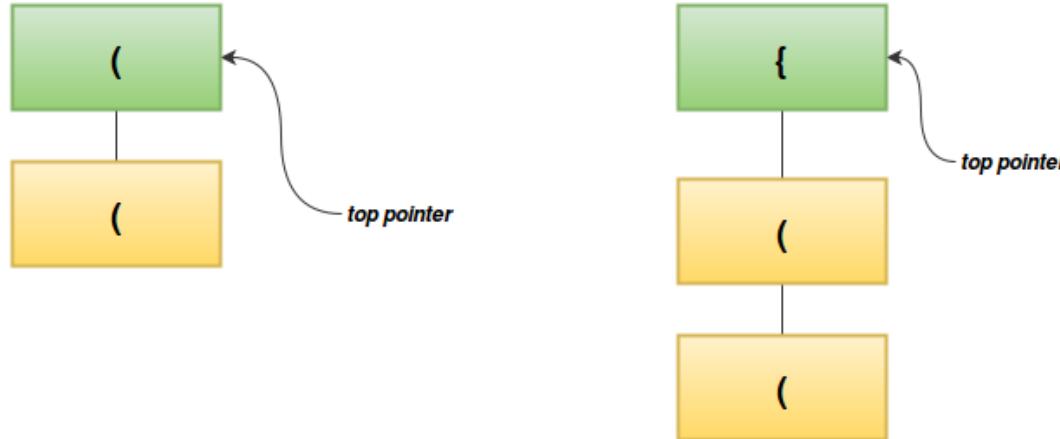
ডিরেক্ষন অ্যারে

মিট ইন দ্য মিডল

চেইল-কল রিকার্সন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

২য় এবং ৩য় ক্যারেক্টার হলো “(” এবং “{”। এদেরকেও পুশ করবো:



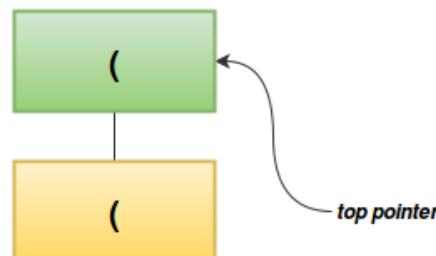
পরের ক্যারেক্টারটা হলো “}”। ক্লোজিং ব্রাকেট যখন পাবো তখন আমরা দেখবো স্ট্যাকের উপরে কি আছে:

যদি স্ট্যাকটা খালি হয় তাহলে সিকোয়েন্টা ব্যালেন্সড না।

যদি বর্তমান ক্যারেক্টারটা “}” হয় তাহলে স্ট্যাকের উপরে অবশ্যই “{” থাকতে হবে।

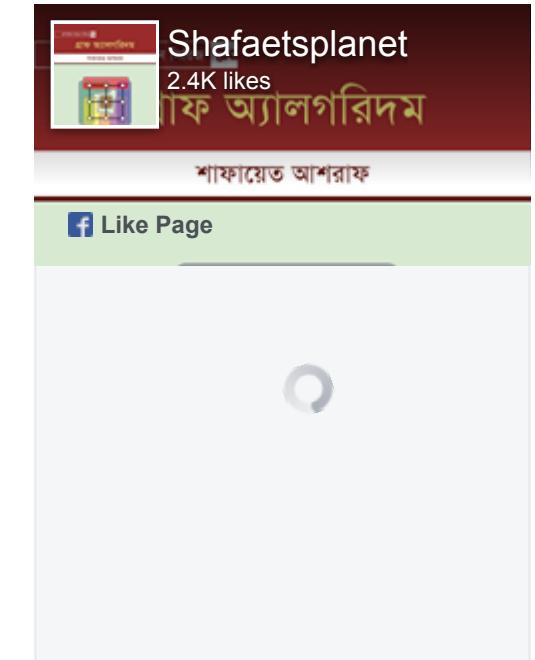
যদি বর্তমান ক্যারেক্টারটা “)” হয় তাহলে স্ট্যাকের উপরে অবশ্যই “(” থাকতে হবে।

আমাদের স্ট্যাকের উপরে “{” আছে যেটা “}” এর সাথে ম্যাচ করে। আমরা “}” নিয়ে কিছু করবো না কিন্তু “{” কে স্ট্যাকের উপর থেকে পপ করে দিবো।

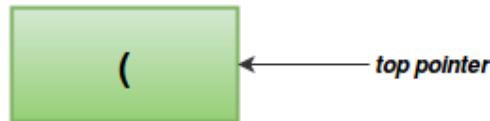


কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



শেষ ক্যারেক্টারটা হলো ")" যেটা "(" এর সাথে ম্যাচ করে, আবার আমরা পপ করবো:



সব কাজ শেষে দেখা যাচ্ছে স্ট্যাকে এখনো একটা ব্রাকেট থেকে গেছে যেটা কারো সাথে মিলানো যাচ্ছে না! তারমানে আমাদের ব্রাকেট সিকোয়েজটা ব্যালেন্সড না। ব্যালেন্সড হলে সব ক্যারেক্টার নিয়ে কাজ করার পর অবশ্যই স্ট্যাক খালি হয়ে যেত।

আমরা স্ট্যাক এর কোড ইতিমধ্যেই লিখেছি, এবার ব্রাকেট ব্যালেন্স করার কোডটা লিখে ফেলি:

```
1 def checkBalance(s):
2     mystack=Stack(len(s))
3     if s=="":
4         return True
5     for c in s:
6         if c=="(" or c=="{":
7             mystack.push(c) #push the opening bracket
8         else:
9             if mystack.is_empty():
10                 return False
11             if c=="{" and mystack.peek()!="}": #the brackets dont match
12                 return False
13             if c=="(" and mystack.peek()!=")": #the brackets dont matchs
14                 return False
15             mystack.pop() #pop matching brackets
16
17     if mystack.is_empty(): #stack must be empty at the end
18         return True
19     return False
20
21
22
23 print checkBalance("(){}")
24 print checkBalance("()(((")
25 print checkBalance("(){}")
```

অ্যালগোরিদমটা বুঝেছে নাকি পরীক্ষা করতে **UVA 674** প্রবলেমটা সমাধান করে ফেলো।

স্ট্যাকের আরেকটা গুরুত্বপূর্ণ ব্যবহার হলো গাণিতিক ইকুয়েশনের মান বের করা, যেমন $(1 + (2 + 5) * 3) + (5 * 7)$ এরকম একটা ইকুয়েশন দেয়া থাকলে মান বের করা খুব সহজ না। কিন্তু [রিভার্স পলিশ নোটেশন](#) লিখলে স্ট্যাক দিয়ে খুব সহজে মান বের করা যায়। এটা এখন তুমি সহজেই উইকিপিডিয়া থেকে শিখে ফেলতে পারবে, আমি বিস্তারিত লিখবো না, আমার কাজ শুধু বেসিক জিনিসগুলো তোমাকে জানানো, কিন্তু ডালো করতে হলে নিজে কষ্ট করে শেখার কোনো বিকল্প নেই 😊

হাপি কোডিং।

ফেসবুকে মন্তব্য

1 comments

Powered by [Facebook Comments](#)

ডাটা স্ট্রাকচার: কিউ এবং সার্কুলার কিউ

অক্টোবর ৬, ২০১৬ by শাফায়েত



কিউ একটা বেসিক ডাটা স্ট্রাকচার। এটাকে তুমি চিন্তা করতে পারো বাসের লাইনের মত, যে সবার সামনে দাঢ়িয়ে আছে সে সবার আগে উঠবে, নতুন কোনো যাত্রী আসলে সে লাইনের পিছনে দাঢ়াবে।

কিউতে দুইরকম অপারেশন থাকে। এনকিউ(Enqueue) মানে হলো কিউতে নতুন এলিমেন্ট যোগ করা এবং ডিকিউ(Dequeue) বা পপ মানে হলো সবথেকে পুরোনো এলিমেন্টটা কিউ থেকে সরিয়ে ফেলা।

অ্যারে ব্যবহার করে আমরা ফিল্ড সাইজের কিউ ইম্প্রিমেন্ট করতে পারি। আমাদেরকে সবসময় দুইটা পয়েন্টার রাখতে হবে, হেড (Head) পয়েন্টার নির্দেশ করবে কিউয়ের সামনের এলিমেন্টের পজিশন এবং টেইল (Tail) পয়েন্টার নির্দেশ করবে পিছনের এলিমেন্টের পজিশন। একদম শুরুতে Head = -1, Tail = -1 রাখতে পারো। প্রতিটি এলিমেন্ট এনকিউ করার সময় টেইলকে এক পজিশন সামনে এগিয়ে নিয়ে সেই পজিশনে নতুন এলিমেন্টকে রাখতে হবে। ডিকিউ করার সময় শুধুমাত্র হেড একধাপ এগিয়ে নিতে হবে। একটা সিমুলেশন দেখলেই ব্যাপারটা পরিষ্কার হবে:

সাবস্ক্রাইব

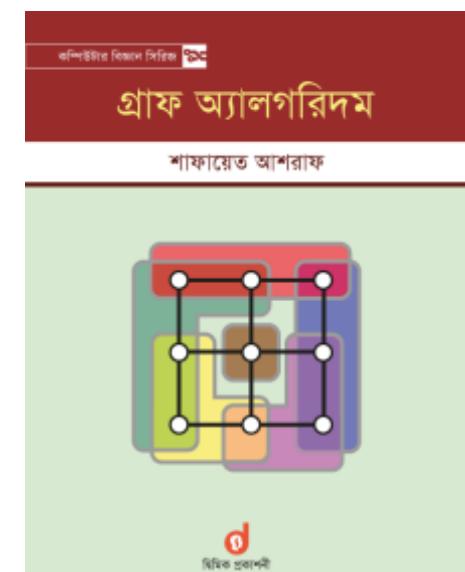


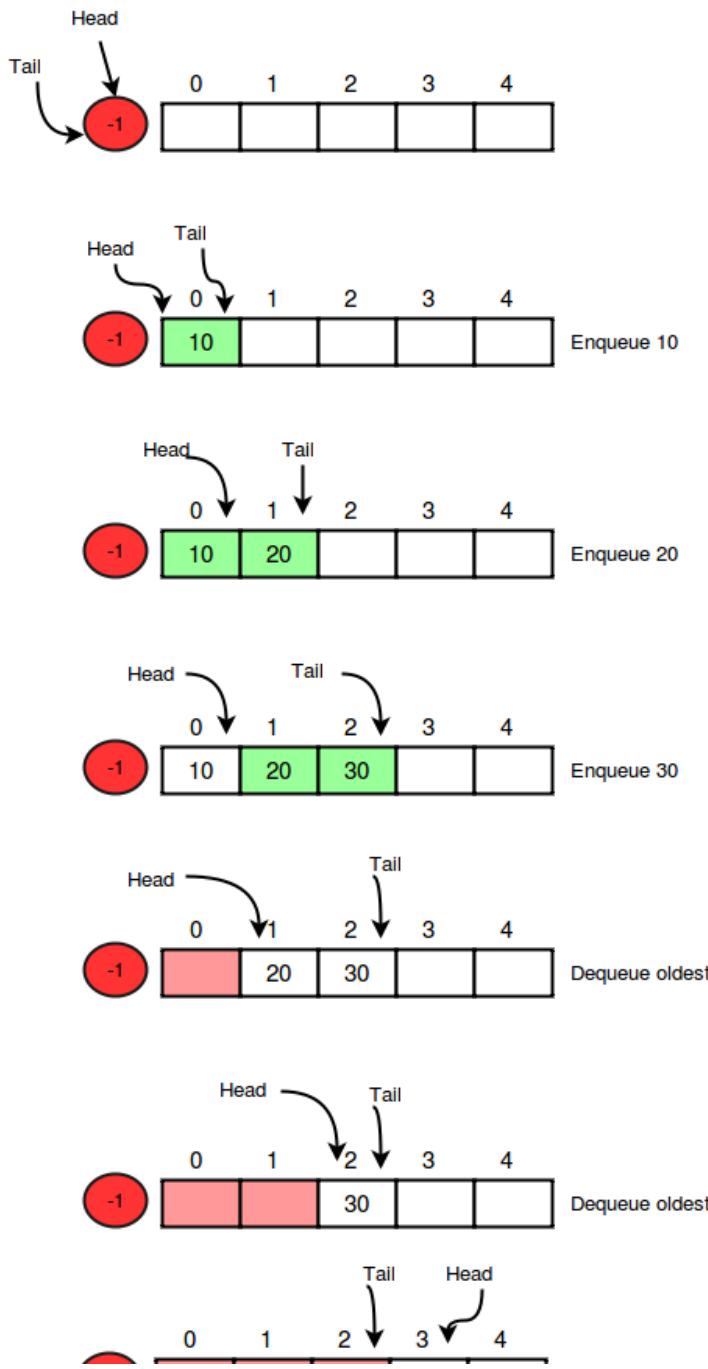
Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

Traveloka Singapore (বিস্তারিত...)





[Like Page](#)

প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাগে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইড্ভিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



ଲକ୍ଷ କର, ଶେଷେ ଏଲିମେଣ୍ଟ୍‌ଟା ଡିକିଉ କରାର ପର ହେତୁ ଟେଇଲେର ଏର ସାମନେ ଚଲେ ଗେଛେ । ଏତେ କୋନୋ ସମସ୍ୟା ନାହିଁ, Head > Tail ବା Head = -1 ମାନେ ହଲୋ କିଉଟା ପୁରୋପୁରି ଖାଲି ।

ଉପରେ କିଉଟେ ସର୍ବୋଚ୍ଚ ୫ଟା ଏଲିମେଣ୍ଟ ରାଖା ଯାବେ । କିନ୍ତୁ ଆରୋ ବଡ଼ ଏକଟା ସମସ୍ୟା ଆଛେ । Head ବା Tail ସବସମୟ ସାମନେ ଆଗାଚେ ଏବଂ ଡିକିଉ କରାର ସମୟ ଅୟାରେ ଯେ ଜାୟଗାଟା ଫାକା ହେଁ ଯାଚେ ସେଟା ଦ୍ଵିତୀୟବାର ବ୍ୟବହାର କରାର କୋନୋ ଉପାୟ ନେଇ ! ସର୍ବଶେଷ ଧାପେ Tail = 2 ହେଁ ଗିଯେଛେ, ଯଦିଓ କିଉଟା ଖାଲି । ତାରମାନେ ପ୍ରଥମ ୩ଟା ପରିଭରନ ଆର ବ୍ୟବହାର କରତେ ପାରବେ ନା । ଏହି କାରଣେ ବାସ୍ତବେ କଥନୋଇ ଏତାବେ କିଉ ଇମାଲିମେଣ୍ଟ କରା ହ୍ୟ ନା, କରଲେ ମେମରିର ସର୍ବୋଚ୍ଚ ବ୍ୟବହାର କରତେ ପାରବେ ନା ।

ଏକଟା ସମାଧାନ ହଲୋ **ଲିଂକଡ ଲିସ୍ଟ** ବ୍ୟବହାର କରା । ଲିଂକଡ ଲିସ୍ଟ ବ୍ୟବହାର କରଲେ ପ୍ରଥମ ସୁବିଧା ହଲୋ କିଉ ଏବଂ ସାଇଜ ଫିଳ୍‌ଡ କରେ ଦେୟା ଦରକାର ନେଇ । ଆବେକଟା ସୁବିଧା ହଲୋ ତୁମି ଯଥନ ଡିକିଉ କରବେ ତଥନ ଟେଇଲ ପଯେଟୋର ଯେ ଏଲିମେଣ୍ଟକେ ପଯେନ୍ଟ କରେ ଆଛେ ତାକେ ମେମରି ଥେକେ ମୁଛେ ଦିତେ ପାରବେ ।
ଲିଂକଡ ଲିସ୍ଟ ବ୍ୟବହାର କରେ ଏକଟା ସିମୁଲେଶନ ଦେଖି ।

ଶାଇଡ଼ିଂ ବେଙ୍ଗ ମିନିମାମ କୁଯେରି (ଡିକିଉ)

ଡିସଜ୍ୟେଟ ସେଟ(ଇଉନିଯନ ଫାଇଲ)

ଟ୍ରାଇ(ପ୍ରିଫିଲ୍ ଟ୍ରି/ବେଡିଲ୍ ଟ୍ରି)

ସେଗମେନ୍ଟ ଟ୍ରି-୧

ସେଗମେନ୍ଟ ଟ୍ରି-୨(ଲେଜି ପ୍ରପାଗେଶନ)

ଆରେ କମପ୍ରେଶନ/ମ୍ୟାପିଂ

ଲୋୟେସ୍ଟ କମନ ଅୟାନ୍‌ସେଟ୍‌ର

ବାଇନାରି ଇନଡେକ୍ୟୁଅଟ ଟ୍ରି

ସ୍଱ଯାର-ରୁଟ ଡିକମ୍ପୋଜିଶନ କ୍ୟାଶିଂ ଅୟାଲଗାରିଦମ:

ଏଲ-ଆର-ଇଉ କ୍ୟାଶ(ନୃତ୍ୟ)

ବ୍ଲୁମ ଫିଲ୍ଟାର(ନୃତ୍ୟ)

ଗ୍ରାଫ ଥିଓରି(୨୦):

ଗ୍ରାଫ ଥିଓରିତେ ହାତେଖଡ଼ି

ଅୟାଡଜେସେନ୍ ମ୍ୟାଟ୍ରିକ୍ୱ

ଅୟାଡଜେସେନ୍ ଲିସ୍ଟ

ଓର୍ଥଡ ଫାର୍ସ୍ ସାର୍ଚ (ବିଏଫ୍‌ୱେସ୍)

ମିନିମାମ ସ୍ପ୍ଯାନିଂ ଟ୍ରି ୧ (ପ୍ରିମ୍‌ସ ଅୟାଲଗୋରିଦମ)

ମିନିମାମ ସ୍ପ୍ଯାନିଂ ଟ୍ରି ୨ (କ୍ରୁସକାଲ ଅୟାଲଗୋରିଦମ)

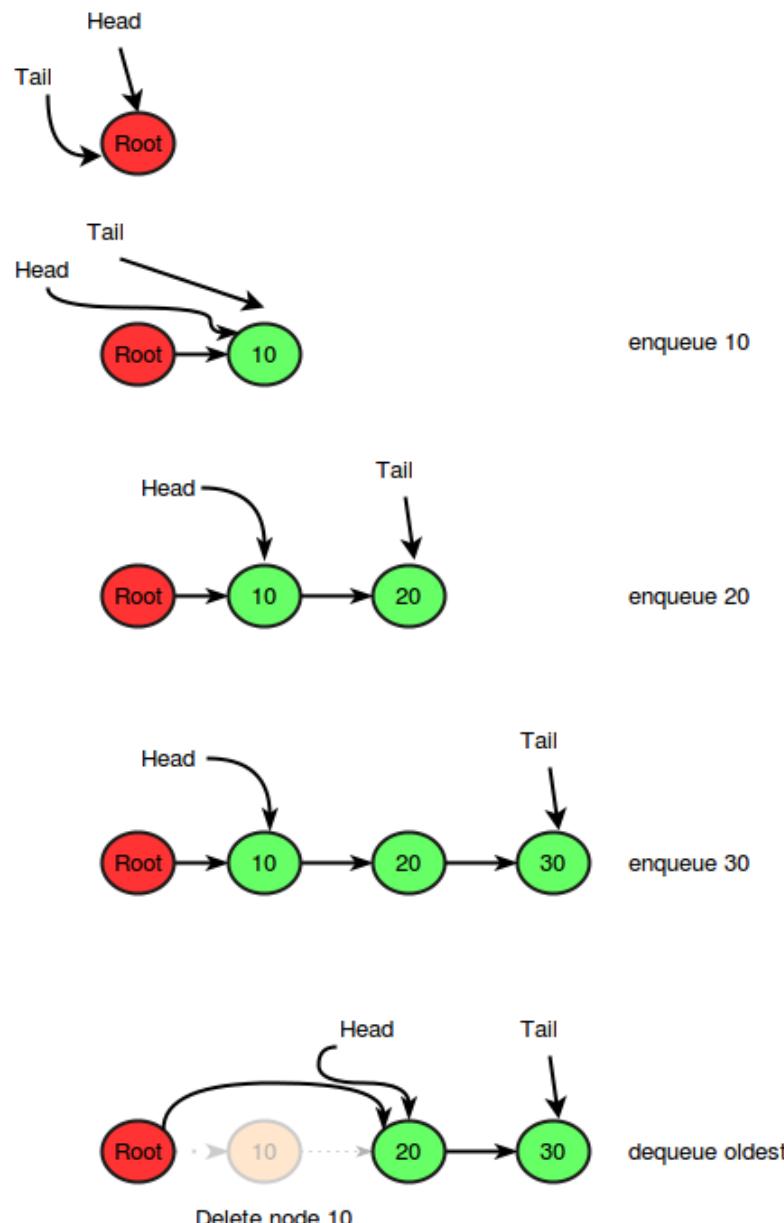
ଟପୋଲାଜିକାଲ ସାର୍ଚ

ଡେପଥ ଫାର୍ସ୍ ସାର୍ଚ ଏବଂ ଆବାରୋ ଟପୋଲାଜିକାଲ ସାର୍ଚ

ଡାୟାକ୍ରିଟ୍ରା

ଫ୍ଲୋଇ ଓ୍ୟାର୍ଶଲ

ବେଲମ୍ୟାନ ଫୋର୍ଡ



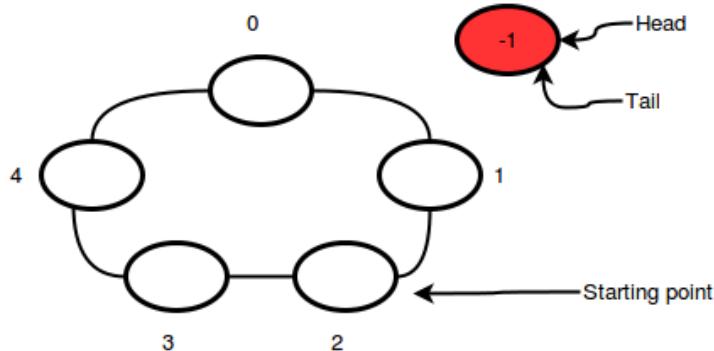
আমরা দেখতে পাচ্ছি এখানে কোন মেমরি অপচয় হবার সুযোগ নেই।

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ
স্ট্রাংলি কানেক্টেড কম্পানেন্ট
ম্যাঞ্চিমাম ফ্লো-১
ম্যাঞ্চিমাম ফ্লো-২
স্টেবল ম্যারেজ প্রবলেম
অয়লার ট্যুর মিনিমাম ডারটেক্স কভার
ট্রি এর ডায়ামিটার নির্ণয়
লংগেস্ট পাথ প্রবলেম

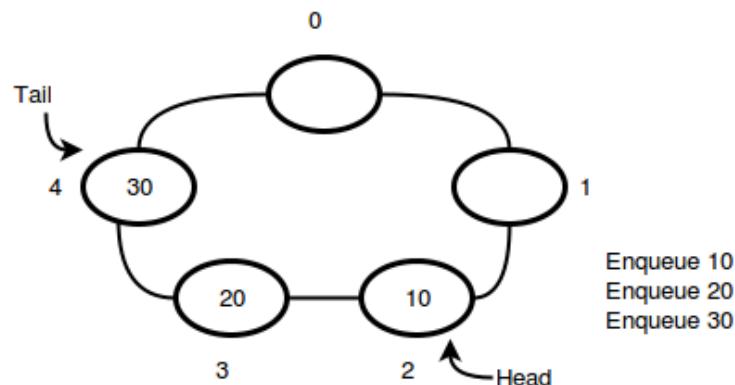
অ্যালগরিদম গেম থিওরি(৩):
গেম থিওরি-১
গেম থিওরি-২
গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):
ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)
ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)
ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন
সার্ভিসিকোয়েস)
ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১
ন্যাপস্যাক)
ডাইনামিক প্রোগ্রামিং ৬ (সার্ভেস্ট সাম,
কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

ফিল্ড সাইজের অ্যারে দিয়েও এমনভাবে কিউ ইমপ্লিমেন্ট করা যাতে মেমরি অপচয় না হয়। একে বলা হয় সার্কুলার কিউ যা দেখতে অনেকটা এরকম:



সার্কুলার কিউতে তুমি শুরুতে যেকোনো পজিশনে এনকিউ করতে পারো, উদাহরণ হিসাবে ছবিতে 2 নম্বর ইনডেক্সকে স্টার্টিং পয়েন্ট ধরেছি।
আগের মতোই কিউ 10, 20 এবং 30 এনকিউ করার পর দেখতে হবে এরকম:



ডিকিউ ও একই ভাবে করবো:

[পুরানো সিরিজ]
বিটমাস্ক ডিপি
মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)
ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

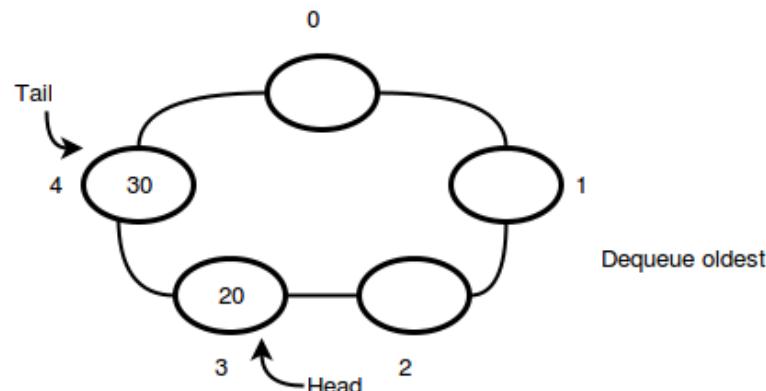
ব্যকট্যাক্স(১):
ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):
মডুলার অ্যারিথমেটিক
প্রাইম জেনারেটর (Sieve of Eratosthenes)
বিটওয়াইজ সিভ
ডিরেঞ্জমেন্ট
প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

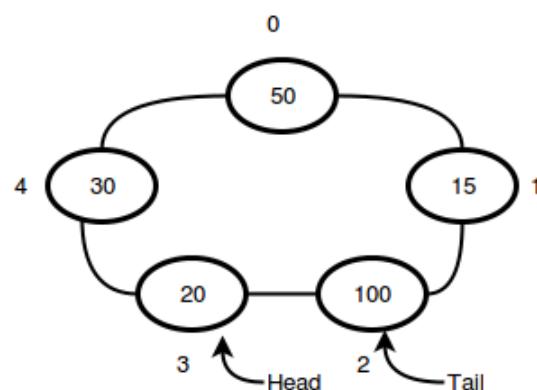
স্ট্রিং ম্যাচিং(২):
রবিন-কার্প
কেএমপি (KMP)

অন্যান্য(৩) :
ডিরেকশন অ্যারে
মিট ইন দ্য মিডল
টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



এখন মনে করো কিউটা সম্পূর্ণভাবে গেছে:



এখন যদি আরো একটা নতুন এলিমেন্ট এনকিউ করতে চাও তাহলে তোমার প্রয়োজনের উপর ডিপেন্ড করে দুইরকম ঘটনা ঘটতে পারে। এনকিউ ফাংশন এরোর দিতে পারে যে জায়গা খালি নেই। অথবা সবথেকে পুরানো এলিমেন্টটা ফেলে দিয়ে সেই জায়গায় নতুন এলিমেন্টটা রাখতে পারো, এটাকে বলা হয় **সার্কুলার বাফার**।

এখন প্রশ্ন হলো কিভাবে বুঝবে যে সার্কুলার কিউ ফুল হয়ে গেছে নাকি? কিউ ফুল হলে অবশ্যই টেইলের পজিশন হেডের এক ধাপ পিছনে হবে।
সার্কুলার কিউ এর একটা পাইথন কোড দেখ:

```
1 | Q = []
2 | head = -1
```

কোয়ান্টাম কম্পিউটার কী?
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Be the first of your friends to like this



ডাইনামিক প্রোগ্রামিং এর নতুন
সিরিজের ৬ নম্বর লেখায় আমরা
দেখবো কম্বিনেটরিক্যাল প্রবলেম এবং
ডিসিপ্লিন প্রবলেম কিভাবে সমাধান
করতে হ্য।

$i, W) = 1 \text{ if } W =$
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ সেমা

```

3 tail = -1
4 capacity = 5
5 starting_point = 2
6 Q = [None for x in range(0, capacity)]
7 def enqueue(value):
8     global head
9     global tail
10    global capacity
11    global Q
12    global starting_point
13
14    if (tail + 1)%capacity == head:
15        print "Q is full"
16        return
17    if head == -1:
18        head = starting_point
19        tail = head
20        Q[tail] = value
21    else:
22        tail = (tail + 1)%capacity
23        Q[tail] = value
24
25 def deque():
26     global head
27     global tail
28     global capacity
29     global Q
30     global starting_point
31     if head == -1:
32         print "Q is already empty"
33         return
34     Q[head] = None
35     if head == tail:
36         head = -1
37         tail = -1
38     else:
39         head = (head +1)%capacity

```

আরেক ধরণের কিউ আছে যার নাম ডাবল এন্ডেড কিউ। ডাবল এন্ডেড কিউ এর দুই পাশেই এলিমেন্ট প্রবেশ করানো যায়, আবার দুইদিক থেকেই পপ করা যায়। ডাবল এন্ডেড কিউ দিয়ে সমাধান করা যায় এমন একটা মজার সমস্যা আলোচনা করেছি **স্লাইডিং রেঞ্জ মিনিমাম কুয়েরি** নিয়ে **লেখায়**।

প্রায়োরিটি কিউ নামের আরও এক ধরণের কিউ আছে। সেখানে প্রতিটা এলিমেন্টের একটা প্রায়োরিটি থাকে, পপ করার সময় যার প্রায়োরিটি বেশি সে আগে পপ হয়। প্রায়োরিটি কিউ ইমপ্লিমেন্ট করতে হলে হিপ ডাটা স্ট্রাকচার সম্পর্কে জানতে হবে, সেটা নিয়ে আরেকদিন আলোচনা করবো।

প্রোগ্রামিং কনটেন্ট কিউ এর সবথেকে কমন ব্যবহার হলো [ব্রেখড ফার্স্ট সার্চ](#)। প্রায়োরিটি কিউ ব্যবহার করে ডায়াক্রস্ট্রাকচার অ্যালগোরিদম ইমপ্লিমেন্ট করা হয়। আবার অপারেটিং সিস্টেম বিভিন্ন রকমের কিউ ব্যবহার করে টাঙ্ক শিডিউলিং এর জন্য।

প্রতিটি বড় প্রোগ্রামিং ল্যাংগুয়েজেই কিউ লাইব্রেরি আছে যা দিয়ে খুব সহজে কিউ ব্যবহার করা যায়। কিন্তু শেখার সময় নিজেকে অবশ্যই ইমপ্লিমেন্ট করতে হবে, নাহলে কিউ কিভাবে কাজ করে বুঝতে পারবে না।

আজ এই পষ্টেই, হ্যাপি কোডিং!

ফেসবুকে মন্তব্য

3 comments

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি

অক্টোবর ১৩, ২০১৫ by শাফায়েত



মনে করো তোমাকে একটা অ্যারে দেয়া হয়েছে যেখানে n টা সংখ্যা আছে। তোমাকে বলা হলো সেই অ্যারের $m=3$ আকারের যতগুলো সাবঅ্যারে আছে সবগুলো থেকে সবথেকে ছোটো সংখ্যাটা বের করতে হবে।

যেমন অ্যারেটা যদি হয় $10, 2, 5, 9, 6, 8$ তাহলে $m=3$ সাইজের সবগুলো সাবঅ্যারে হলো:

“ ১০, ২, ৫, সবনিম্ন সংখ্যা ২

২, ৫, ৯, সবনিম্ন সংখ্যা ২

৫, ৯, ৬, সবনিম্ন সংখ্যা ৫

৯, ৬, ৮, সবনিম্ন সংখ্যা ৮

তাহলে তোমার আউটপুট হবে $[2, 5, 5, 8]$ ।

m এর মান ৩ না হয়ে ১ থেকে n পর্যন্ত যেকোনো সংখ্যা হতে পারে।

n এর মান যদি ছোটো হয় তাহলে আমরা সহজেই প্রতিটা সাবঅ্যারের উপর লুপ চালিয়ে সমস্যাটা সমাধান করতে পারি। নিচের পাইথন কোডটি দেখো:

সাবক্ষাইব

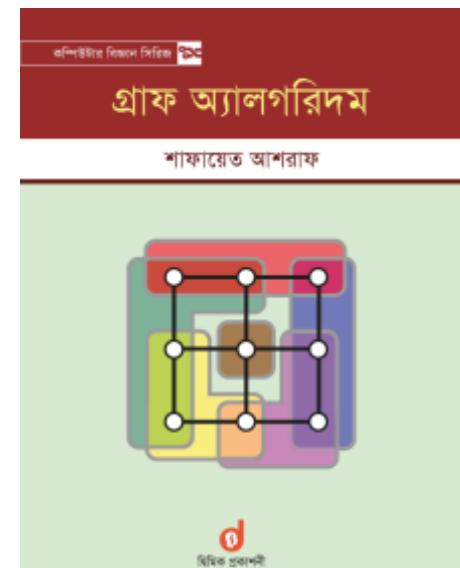


Secured by OneAll Social Login

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

Traveloka Singapore (বিস্তারিত...)



```

1 def brute_rmq(arr,m):
2     res=[]
3     for i in range(0,len(arr)-m+1):
4         subarr=arr[i:i+m] #take subarray of size m, starting from index i
5         res.append(min(subarr)) #append the minimum element in result
6     return res

```

এই কোডের কমপ্লেক্সিটি $O(n^2)$ ।

আমরা $O(n \log n)$ এ সমস্যাটা সমাধান করতে পারি সেগমেন্ট ট্রি ব্যবহার করে।

শ্বাইডিং উইল্ড এবং মনোটোনাস ডিকিউ ব্যবহার করে সমস্যাটা $O(n)$ কমপ্লেক্সিটিতে সমাধান করা সম্ভব, সেটাই আজকে আমরা শিখবো।
মনোটোনাস ডিকিউ বা ডাবল-এন্ডেড-কিউ হলো এমন একটা ডিকিউ যেখানে সংখ্যাগুলো সবসময় সর্টেড থাকে।

মনে করি অ্যারেতে সংখ্যাগুলো হলো $[10, 50, 15, 12, 8]$ এবং $m=3$ ।

আমরা বাম থেকে ডানে একটা একটা সংখ্যা নিয়ে কাজ করতে থাকবো। আমরা সংখ্যাগুলোকে এমনভাবে ডিকিউতে ঢুকাবো যেন সবথেকে ছোটো সংখ্যাটা সবসময় সবার ডানে থাকে। তাম ইনডেক্সে যখন থাকবো তখন $(i - m + 1, i)$ সাবঅ্যারের সর্বনিম্ন সংখ্যাটাকে ডিকিউর সবথেকে ডানে পাওয়া যাবে।

প্রথম সংখ্যাটা হলো 10, এটাকে আমরা ডিকিউ তে বামদিক থেকে ঢুকাবো:

“ $DQ=[10]$ ”

পরের সংখ্যাটা হলো 50, এটাকেও বামদিক থেকে ঢুকাবো:

“ $DQ=[50, 10]$ ”

পরের সংখ্যাটা হলো 15। এখন লক্ষ্য করো, এখন পর্যন্ত যতগুলো সংখ্যা পেয়েছি তাদের মধ্যে যারা 15 এর থেকে বড় তারা কখনোই সর্বনিম্ন সংখ্যা হতে পারবে না, কারণ তারা 15 এর বামে আছে এবং তারা যে সাবঅ্যারেতে আছে সেগুলোতে 15 ও অবশ্যই আছে। এটা বোঝাই



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগরিদম

ডাটা স্টোরেজ(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

অ্যালগোরিদমের সবথেকে গুরুত্বপূর্ণ অংশ। কোনো একটা সংখ্যা ডিকিউতে চুকানোর আগে সেই সংখ্যাটার থেকে যতগুলো বড় সংখ্যা ডিকিউতে আছে সেগুলো বের করে দিতে হবে।

DQ=[১৫,১০]

তাহলে প্রথম ৩ আকারের সাবঅ্যারে [১০,৫০,১৫] এ সর্বনিম্ন সংখ্যা হলো ডিকিউ এর সর্বডানের সংখ্যা ১০।

পরের সংখ্যাটা হলো ১২। তাহলে আমরা ১৫ কে ফেলে দিয়ে ১২ কে চুকাবো।

“ DQ=[১২,১০]

লক্ষ্য করো আমরা এখন $i=3$ নম্বর ইনডেক্সে আছি এবং $i-m+1=3-3+1=1$ নম্বর ইনডেক্সের বামের কোনো সংখ্যা আমাদের দরকার নেই কারণ সেগুলো বেঞ্জের বাইরে। কিউ এর সবার ডানের সংখ্যা ১০ মূল অ্যারের এর ০ তম ইনডেক্সে অবস্থিত, সেটাকে আমরা ফেলে দিতে পারি।

“ DQ=[১২]

তাহলে ২য় ৩ আকারের সাবঅ্যারে [৫০,১৫,১২] এ সর্বনিম্ন সংখ্যা হলো ডিকিউ এর সর্বডানের সংখ্যা ১২।

পরবর্তি সংখ্যাটা হলো ৪। আমরা ১২ ফেলে দিয়ে ৪ চুকাবো:

“ DQ=[৪]

তাহলে ৩য় ৩ আকারের সাবঅ্যারে [১৫,১২,৪] সর্বনিম্ন সংখ্যা হলো ডিকিউ এর সর্বডানের সংখ্যা ৪।

তাহলে $O(n)$ কমপ্লিক্ষিটিতে আমরা সবগুলো বেঞ্জের সর্বনিম্ন সংখ্যাগুলো বের করে ফেললাম।

নিচের পাইথন কোডে উপরের অ্যালগোরিদমটা ইম্প্লিমেন্ট করা হয়েছে। পাইথন না জানলেও বুঝতে সমস্য হবে না:

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

1 def sliding_rmq(arr, m):
2
3     DQ = deque()
4     res = []
5     for i, val in enumerate(arr):
6         while len(DQ) and DQ[0][0] >= val: #DQ[0][0] is the leftmost element of DQ
7             DQ.popleft()
8         DQ.appendleft((val, i)) #pushing a pair containing the value and the index
9
10    while len(DQ) and DQ[-1][1] <= i - m: #DQ[-1][1] is the index of the rightmost element of DQ
11        DQ.pop() #popping the out-of-range elements
12
13    if i >= m - 1: #We got a m size range
14        print(DQ[-1][0]) #print the rightmost element of DQ
15        res.append(DQ[-1][0])
16
17 return res

```

চিন্তা করার জন্য সমস্যা:

১. মনে করো তোমাকে n টা সংখ্যা এবং q টা বেঝি দেয়া হয়েছে, বেঝগুলো হলো $[a_1, b_1], [a_2, b_2], \dots, [a_q, b_q]$ এবং প্রতিটা $i < q$ এর জন্য $a_i \leq a_{i-1}$ এবং $b_i \leq b_{i-1}$ । প্রতিটা বেঝের সর্বনিম্ন সংখ্যা বের করতে হবে। কিভাবে করবে?

২. <http://www.spoj.com/problems/PARSUMS/>

ফেসবুকে মন্তব্য

1 comments

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন
সার্বিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১
ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম,
কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

ডাটা স্ট্রাকচার: ডিসজয়েন্ট সেট(ইউনিয়ন ফাইন্ড)

আগস্ট ২৮, ২০১১ by শাফায়েত



ডাটা স্ট্রাকচার কম্পিউটার সাইনের চমতকার অংশগুলোর একটি। আমরা অসংখ্য উপায়ে কম্পিউটারে ডাটা জমা বাখতে পারি। আমরা বাইনারি ট্রি বানাতে পারি, পরে সে গাছ বেয়ে বেয়ে logN এ ডাটা বের করে আনতে পারি, বাসের লাইনের মত কিউ বানাতে পারি, প্রিফিক্স ট্রি বা trie বানিয়ে খুব দ্রুত স্ট্রিং সার্চ করতে পারি। আজ আমরা দেখবো অসাধারণ একটি ডাটা স্ট্রাকচার যার নাম “ডিসজয়েন্ট সেট”।

kruskal's MST বা **tarjan's offline LCA** ইত্যাদি অ্যালগরিদম ইমপ্লিমেন্ট করতে ডিসজয়েন্ট সেট খুব গুরুত্বপূর্ণ। এটি ইমপ্লিমেন্ট করতে আমাদের একটি অ্যারে ছাড়া কিছু লাগবেনা।

প্রথমে আমরা দেখবো কি ধরণের কাজে আমাদের এই ডাটা স্ট্রাকচারটি দরকার। মনে করি A,B,C,D,E ৫ জন মানুষ। A-B যদি বন্ধু হয় এবং B-C যদি বন্ধু হয় তাহলে আমরা বলতে পারি A-C ও বন্ধু, অর্থাৎ তাদের বন্ধুত্ব transitive। এখন আমি বলে দিলাম যে A-B,B-C,D-E এবা পরম্পরের বন্ধু। এখন তোমাকে প্রশ্ন করলাম A-C কি পরম্পরের বন্ধু? B-D কি পরম্পরের বন্ধু? প্রথম প্রশ্নের উত্তর হবে “হ্যা”, দ্বিতীয় প্রশ্নের উত্তর হবে “না”。 আমাদের তথ্যগুলোকে গ্রাফের মাধ্যমে দেখাতে পারি:

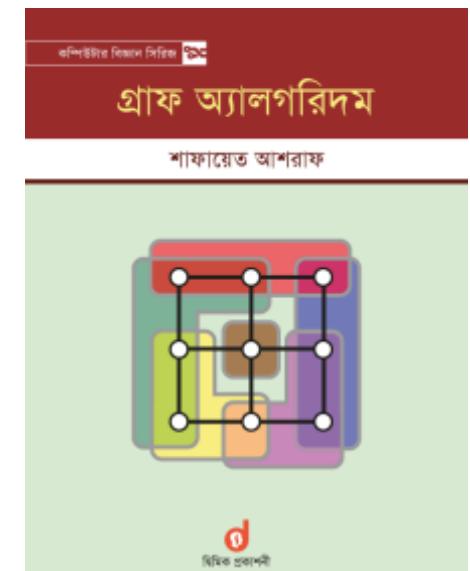
সাবস্ক্রাইব

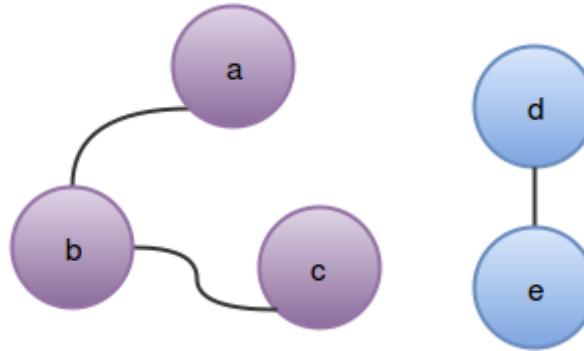


Secured by OneAll Social Login

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)





সহজেই বোধ যাচ্ছে গ্রাফে যেসব নোড একই কম্পনেন্ট বা সাবগ্রাফের মধ্যে আছে তারা পরস্পরের বন্ধু। তাহলে দু-জন ব্যক্তি বন্ধু নাকি সেটা জানতে হল আমাদের দেখতে হবে তারা একই সাবগ্রাফে আছে নাকি। সহজেই বিএফএস বা ডিএফএস চালিয়ে এটা বের করা যায়। কিন্তু এগুলোর থেকেও ভালো উপায় হলো “ডিসজয়েন্ট সেট”, কেনো এটা ভালো সেট কিছুক্ষণ পরেই বুঝতে পারবে।

প্রতিটি মানুষ একটি করে নোড হলে ডিসজয়েন্ট সেট দিয়ে সমাধানের আইডিয়া এরকম:

- “ ১. যারা পরস্পরের বন্ধু তারা সবাই একই সেটের অন্তর্গত
- ২. যতগুলো সাবগ্রাফ থাকবে ততগুলো সেট থাকবে
- ৩. প্রতিটি সেটের একটি *representative* থাকবে।
- ৪. দুটি নোডের *representative* একই হলে তারা একই সেটে আছে। সুতরাং দুজন ব্যক্তি বন্ধু নাকি বুঝতে আমাদের তারা যে সেটে আছে তার *representative* চেক করতে হবে।

যেলাটে লাগছে? সমস্যা নেই, আমরা গ্রাফিকালভাবে এখন ব্যাপারটি বুঝবো, সাথে কোডও দেখবো। শুরুতে কেও কারো বন্ধু নয়। তাহলে সবাই আলাদা আলাদা সেটে আছে। এবং যেহেতু সেটগুলোতে মাত্র একটি করে সদস্য তাই সেই সদস্যটিই হবে সেটের *representative*। ছবিটি দেখো:



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

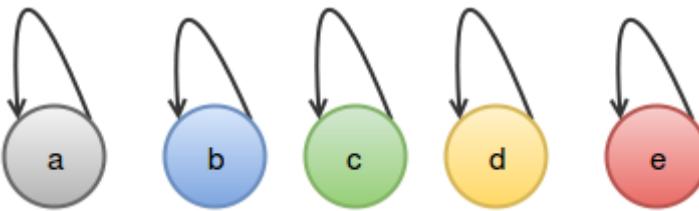
ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগোরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



আমরা সেটের representative কে সবসময় হলুদ রং দিবো।

এই অংশটি ইম্প্লিমেন্ট করবো এভাবে:

```

1 #Python2.7.6
2 #assume a=1,b=2,c=3,d=4,e=5
3 element=5
4 par=[None]*(element+1) #Creating an array with 6 elements
5
6
7 def makeset(n):
8     par[n]=n
9
10 for i in range(1,element+1):
11     makeset(i)

```

par নামক অ্যারেটা দিয়েই আমরা সেট বানাবো। শুরুতে সবার প্যারেন্ট সে নিজেই। প্যারেন্ট বলতে বুঝাচ্ছে নোডটি কার সাথে সংযুক্ত। একটি মাত্র সাধারণ অ্যাবে ব্যবহার করে আমরা পুরো স্ট্রাকচারটি বানাবো! এটা একধরনের ট্ৰি স্ট্রাকচাৰ।

এখন a আৰ b হঠাত ঠিক কৱলো যে তাৰা বন্ধু হবে। তাহলে তাদেৱকে একই সেটের মধ্য আসতে হবে। a ঠিক কৱলো যে b এৰ representative কে নিজেৰ representative বানিয়ে ফেলবে, তাহলেই তাৰা একই সেটে চলে আসবে। তাই a এসে b এৰ সাথে যুক্ত হয়ে গেলো কাৰণ b এৰ representative হলো b নিজেই। ছবি দেখো:

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েৰি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইন্ড)

ট্ৰাই(প্ৰিফিল্ট্ৰ ট্ৰি/বেডিল্ফ ট্ৰি)

সেগমেন্ট ট্ৰি-১

সেগমেন্ট ট্ৰি-২(লেজি প্ৰপাগেশন)

অ্যারে কমপ্ৰেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেষ্টৰ

বাইনাৰি ইনডেক্সড ট্ৰি

স্থায়াৰ-কুট ডিকম্পোজিশন ক্যাশিং অ্যালগোরিদম:

এল-আৱ-ইউ ক্যাশ(নতুন)

বনুম ফিল্টাৰ(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্ৰিক্স

অ্যাডজেসেন্সি লিস্ট

ব্ৰেথড ফার্স্ট সাৰ্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্ৰি ১ (প্ৰিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্ৰি ২ (ক্ৰুসকাল অ্যালগোরিদম)

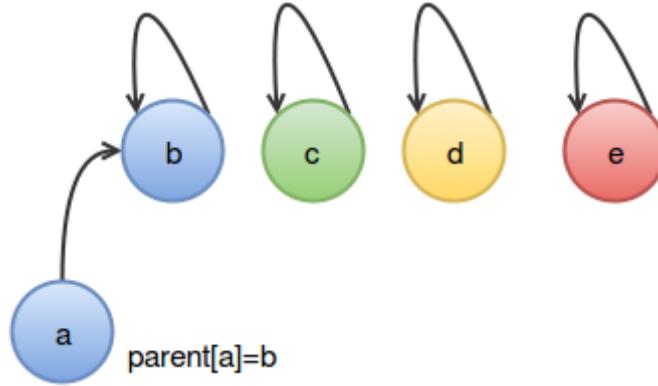
টপোলজিকাল সেট

ডেপথ ফার্স্ট সাৰ্চ এবং আবাৰো টপোলজিকাল
সেট

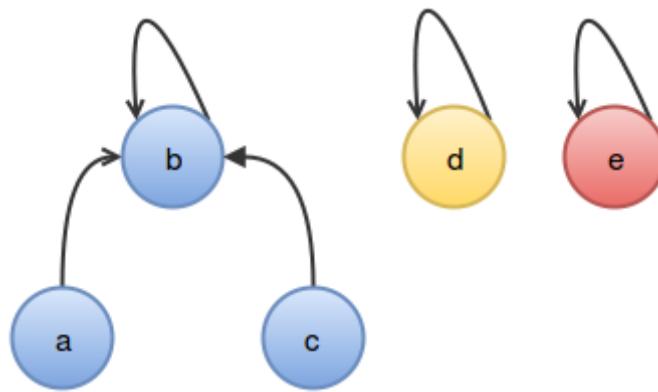
ডায়াক্ৰটা

ফ্ৰয়েড ওয়াশৰ্স্ল

বেলম্যান ফোৰ্ড



এখন c ভাবলো সেও a এর বন্ধু হবে। এখন c কার নিচে যুক্ত হবে? a এর নিচে c যুক্ত না হবেনা, c যুক্ত হবে a এর representative এর নিচে, তারমানে b এর নিচে।



d,e কেও বন্ধু বানিয়ে দেই:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্র্যার মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

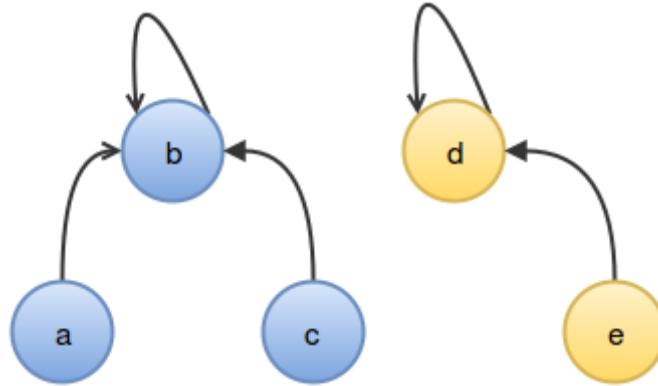
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

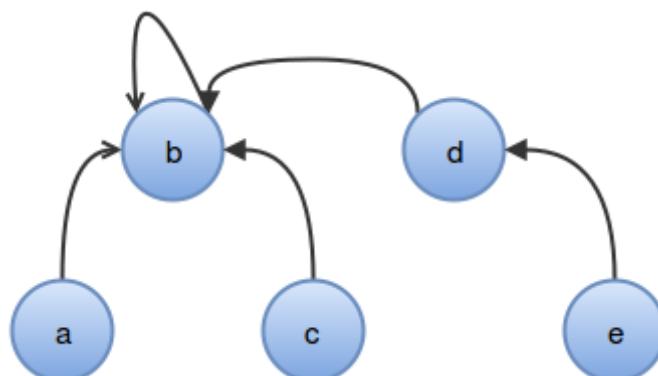
ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সারসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)



`parent[c]=representative(a)=b`

পরের ধাপটি ভালো করে দেখো। এবার আমরা e আর c কে বন্ধ করব। c যেই সেটের representative হলো b। তার সাথে আমরা ঘুর্ণ করবো e এর representative কে। e এর representative হলো d। b কে d এর প্যারেন্ট বানিয়ে দিলাম। তাহলে b এখন e এরও representative হয়ে গেলো।



আমরা আবার একটু কোড দেখি:

```

1 #Python2.7.6
2 def union(a,b):
3     u=find(a)
4     v=find(b)
```

[পূর্বনো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্রিং(১):

ব্যকট্যাক্রিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

চেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

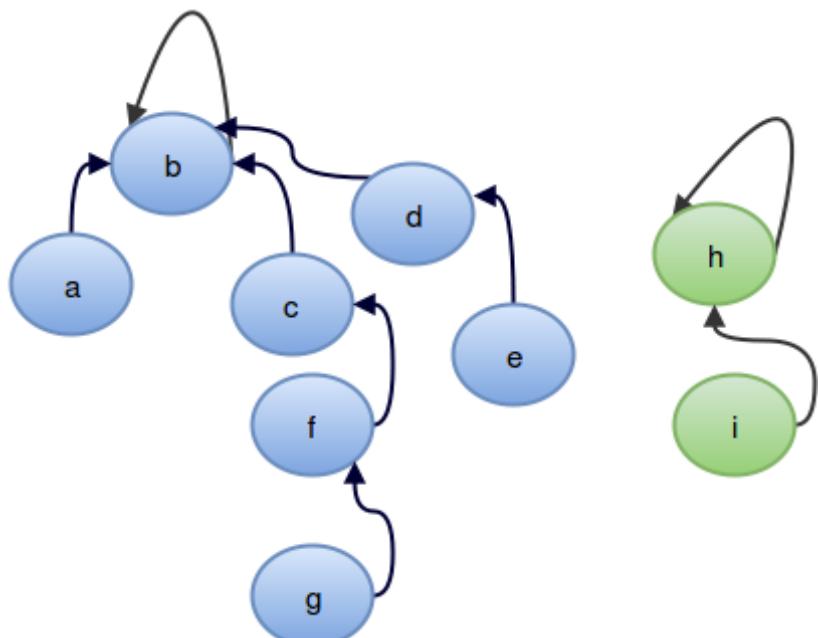
```

5 if u==v:
6     print "They are already friends"
7 else:
8     par[u]=v #Or you can write par[v]=u too

```

আমরা যে দুটি নোডকে বন্ধু বানাবো তাদের representative বের করলাম find() ফাংশন কল করে। (ফাংশনটির ডেফিনিশন পরে দেখবো) এবার একটিকে আরেকটির প্যারেন্ট বানিয়ে দিলাম।

এখন প্রশ্ন হলো representative খুজবে? মনোযোগদিয়ে এই পর্যন্ত পড়ে থাকলে একটা জিনিস চোখে পড়ার কথা, কোনো একটি সেটের **representative element** এর প্যারেন্ট সেই এলিমেন্ট নিজেই, অর্থাৎ এলিমেন্টটি যদি হ্যাঁ r তাহলে $\text{par}[r]=r$ । আমরা আবেক্ষ বড় গ্রাফ দেখি। মনে করি a,b,c,d,e এর বন্ধু হতে আরো কয়েকজন চলে এসেছে:



আগের সেটেই শুধু কিছু নোড যুক্ত করা হয়েছে পরের অংশ বোঝার সুবিধার জন্য। ধরি f এর representative বের করতে হবে। f এর প্যারেন্ট সে নিজে নয়, তাহলে তার representative খুজতে তার প্যারেন্টের প্যারেন্ট চেক করি। f এর প্যারেন্ট c। c ও representative নয় কারণ জে নিজেই নিজের প্যারেন্ট নয়, c এর প্যারেন্ট b এবং b নিজেই নিজের প্যারেন্ট। তাহলে b ই হলো representative। ফাংশনটি লিখে ফেলি:

কোয়ান্টাম কম্পিউটার কী?
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

 Shafaetsplanet
2.4K likes
শাফায়েত আশরাফ
[Like Page](#)

Be the first of your friends to like this

 Shafaetsplanet about an hour ago
ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটারিঙ্গের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্যাঁ।

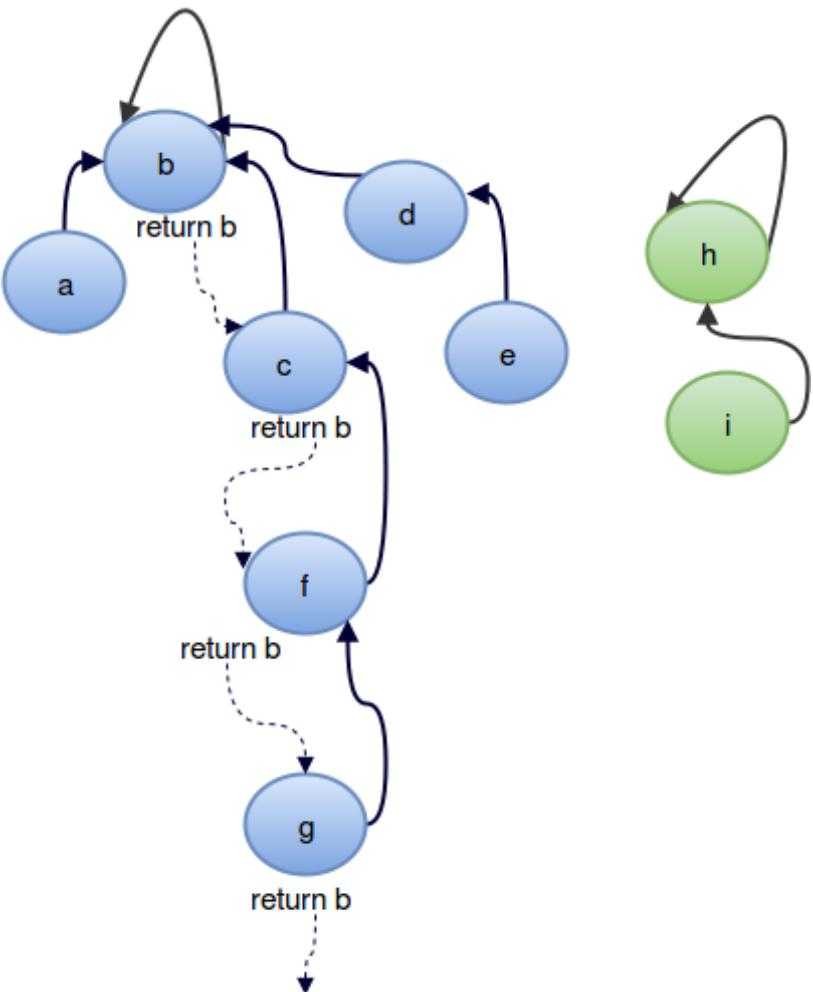
$i, W) = 1 \text{ if } W =$
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / ম্যান

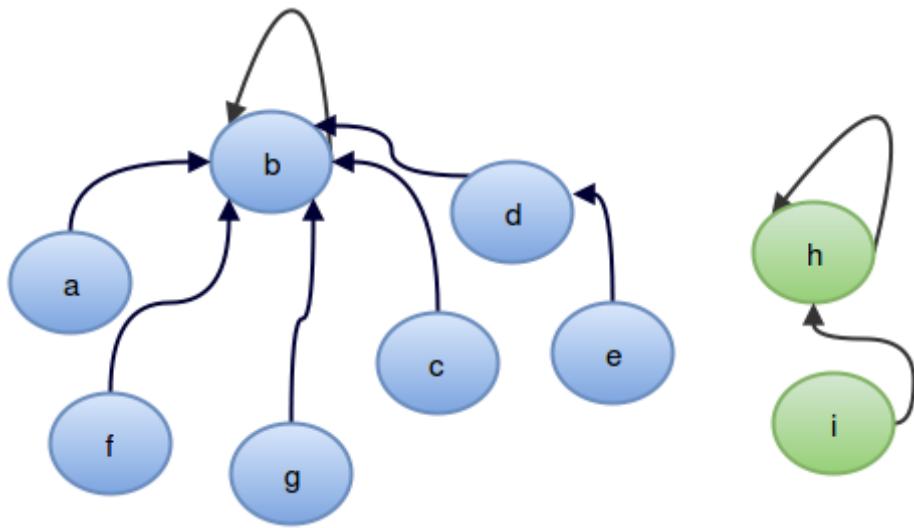
```
1 #python2.7.6
2 def find(r):
3     if par[r]==r: return r
4     return find(par[r])
```

ফাংশনটি গাছ(tree) বেয়ে উপরে উঠলো যতক্ষননা representative কে খুজে পাওয়া যায়। আগের ফাংশনদুটি(union এবং makeset) তাদের আসল কাজ করেছে $O(1)$ এ। find() ফাংশনে এসে complexity'r প্রশ্ন এসে পড়েছে। worst case এ $O(n)$ টাইম লাগতে পারে। উপরের find() ফাংশনের, tree এর depth যত বেশি হবে তত টাইম বেশি লাগবে। ডাস্টির ল্যাবে লেখা কোডের জন্য এটা ঠিক আছে, কিন্তু কনটেন্টে বা অন্য কোথাও যদি নোড অনেক বেশি হয় তাহলে পারফরম্যান্স খারাপ দিবে।

এখনে একটি চমৎকার optimization আছে। আসলে এই optimization টাই ডিসজয়েন্ট সেট স্ট্রাকচারের সৌন্দর্য। প্রতিটি find() ফাংশন কল এর সাথে আমরা গ্রাফটি এমন ভাবে পরিবর্তন করবো যেন depth কমে যায়। g এর representative খুজতে আমরা g এর প্যারেন্ট f কে কল দিয়েছি। f তার প্যারেন্টকে কল দিয়ে representative খুজে বিটার্ন করেছে। বিটার্ন ভ্যালুগুলোকে এভাবে দেখতে পারি:



প্রতিটি নোডের নিচে রিটার্ন ভ্যালু লিখে দেয়া হয়েছে। প্রতিটি নোড তার আগের ফাংশন একই ভ্যালু রিটার্ন করে করে, যেটা সবশেষে আমরা পাই, এই ভ্যালুটাই **representative**। আমরা `return find(par[r])` না লিখে যদি আগে লিখতাম `par[r]=find(par[r])` এবং তারপর লিখতাম `"return par[r]"` তাহলে কি ঘটতো? রিটার্ন করার সময় রিটার্ন ভ্যালুটাকে নিজের প্যারেট বানিয়ে ফেলতো, অর্থাৎ সবাই **representative** কেই বানিয়ে ফেলতো নিজের প্যারেট! ফাংশন কলের সাথে সাথে গ্রাফটি হয়ে যেত এমন:



টি এর depth কমে গিয়েছে, তাই নয় কি? এটাকে বলা হ্য path compression। পরে আমরা যখন f এর representative খুজবো তখন আব মাত্র একটি ডাল(edge) বেয়ে উঠতে হবে। wiki তে সুন্দর করে লেখা আছে:

“ path compression, is a way of flattening the structure of the tree whenever Find is used on it. The idea is that each node visited on the way to a root node may as well be attached directly to the root node; they all share the same representative. To effect this, as Find recursively traverses up the tree, it changes each node's parent reference to point to the root that it found. The resulting tree is much flatter, speeding up future operations not only on these elements but on those referencing them, directly or indirectly.

মোটামুটি এই হলো আমাদের disjoint set। দুটি নোড একই সেটে আছে নাকি চেক করতে আমরা তাদের representative বের করে তারা সমান নাকি সেট চেক করবো। দুটি নোড কে একই সেটে নিতে হল তাদের representative বের করে একটিকে আরেকটির প্যারেন্ট বানিয়ে দিবো। **kruskal** এ দুটি নোড একই সাব-ট্রিতে আছে নাকি সেট চেক করতে disjoint set ব্যবহার করা হয়। একই tree তে না থাকলে union ফাংশনটি কল করে এক করা হয়।

এখন যদি সত্যিই কিছু শিখতে চাও তাহলে ষটপট কিছু প্রবলেম সলভ করে ফেলো:

[Graph connectivity](#)

[Nature](#)

[Virtual Friend\(*\)](#)

আরো জানতে চাইলে:

[টপকোডার টিউটোরিয়াল](#)

[Wikipedia](#)

[সি++ কোডগুলো বদলে পাইথনে লিখে দেয়া হলো, তবে সেজন্য বুঝতে সমস্যা হবার কথা না। সমস্যা হলে যোগাযোগ কর]

[ফেসবুকে মন্তব্য](#)

2 comments

ডাটা স্টুকচার: ট্রাই (প্রিফিক্স ট্রি/রেডিক্স ট্রি)

ডিসেম্বর ২৭, ২০১৩ by [শাফায়েত](#)



ট্রাই বা প্রিফিক্স ট্রি ব্যবহার করে ডাটাবেস থেকে খুব সহজে স্ট্রিং খুজে বের করা যায়। ধরো একটা ফোনবুকে একটা শহরের সব মানুষের ফোন নম্বর রাখা আছে। শহরে মানুষ আছে হয়তো কয়েক লক্ষ্য, কিন্তু প্রতিটা মানুষের নাম সর্বোচ্চ ২০টা অক্ষর ব্যবহার করে লেখা যায়। আমরা এমন একটা ডাটা স্টুকচার ব্যবহার করে নাম খুজবো যে নির্ভর করে শুধু মাত্র নামটিতে কয়টি অক্ষর আছে তার উপর। যেমন "Alice" নামটি খুজতে মাত্র ৫টি অপারেশন করা লাগবে ডাটাবেস যত বড়ই হোক না কেন।

এই লেখা পড়ার আগে [লিংকড লিস্ট](#) এবং [রিকার্সন সম্পর্কে ধারণা](#) থাকতে হবে।

ধরো তোমাকে একটা ডিকশনারী দেয়া হলো যেখানে নিচের শব্দগুলো আছে:

" algo
algea
also
tom
to

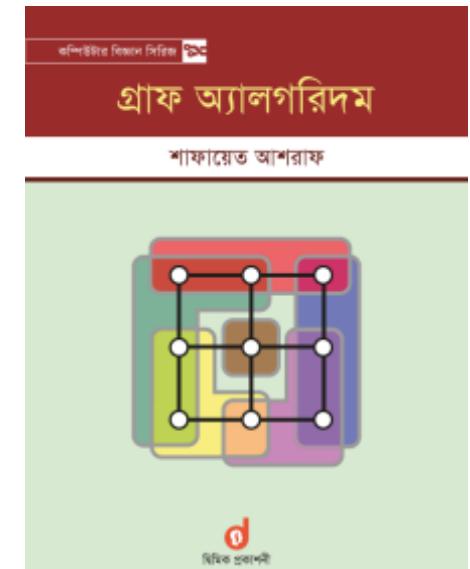
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)

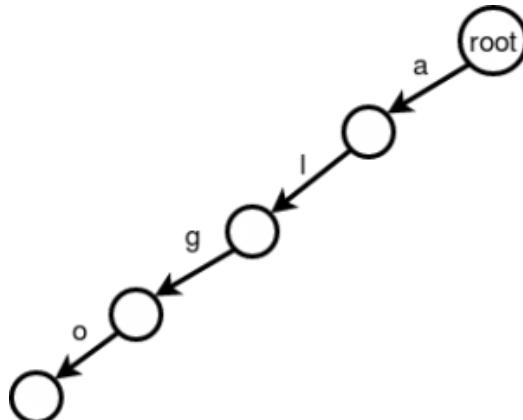


এখন আমরা এই ডিকশনারিটাকে এমনভাবে মেমরিতে রাখতে চেষ্টা করবো যেন খুব সহজে কোনো একটা শব্দ খুজে পাওয়া যায়। একটা উপায় হলো শব্দগুলোকে সঢ় করে রাখা যেটা কাগজের ডিকশনারি গুলোতে রাখা হয়, তাহলে বাইনারি সার্চ করেই আমরা কোনো একটা শব্দ খুজে বের করতে পারবো। আরেকটা উপায় হলো প্রিফিক্স ট্রি বা সংক্ষেপে ট্রাই(trie) ব্যবহার করা। trie শব্দটা এসেছে “*retrieval*” শব্দটা থেকে। সেই হিসাবে এটার উচ্চারণ “ট্রি” হওয়ার কথা কিন্তু গ্রাফ থিওরীতে ট্রি এর আরো ব্যপক ব্যবহার আছে তাই এটাকে বলা হ্য “ট্রাই”। প্রিফিক্স মানে হলো একটা স্ট্রিং এর শুরু থেকে কয়েকটা ক্যারেক্টার নিয়ে নতুন স্ট্রিং তৈরি করা। যেমন blog এর প্রিফিক্স হলো b,bl,blo এবং blog।

শুরুতে আমাদের একটা রুট নোড ছাড়া কিছুই নেই।



এখন আমরা algo শব্দটাকে যোগ করবো। নিচের ছবিতে দেখো কিভাবে শব্দটা যোগ করা হয়েছে। রুট নোড থেকে আমরা একটা এজ দিবো যেই এজ এর নাম হবে “a”। তারপর নতুন তৈরি হওয়া নোড থেকে “l” নামের একটা এজ তৈরি করবো। এভাবে “g” আর “o” এজ দুইটাও তৈরি করবো। লক্ষ্য করো নোডে আমরা কোনো তথ্য রাখছিনা, খালি নোড থেকে এজ বের করছি।



এখন আমরা algea শব্দটা যোগ করতে চাই। রুট থেকে “a” নামের এজ দরকার, সেটা অলবেড়ি আছে, নতুন করে যোগ করা দরকার নাই। ঠিক সেরকম a থেকে। এবং l থেকে g তেও এজ আছে। তারমানে “algt” অলবেড়ি ট্রাই তে আছে, আমরা শুধু e আর a যোগ করবো।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

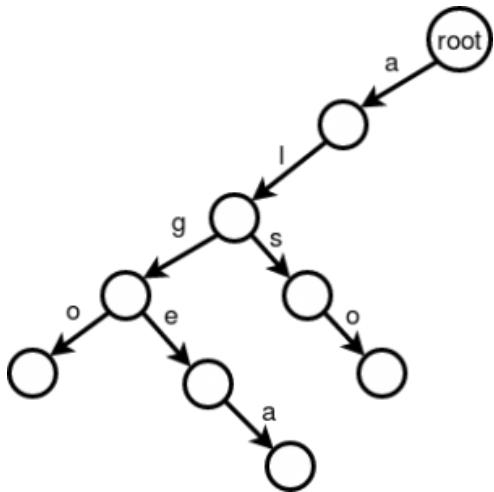
ডাটা স্টোরেজ(১৪):

লিংকড লিস্ট

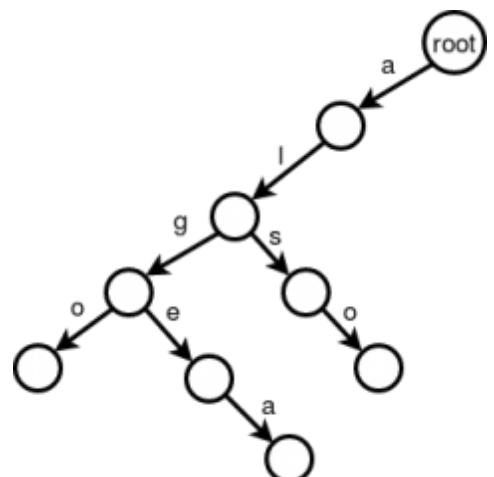
স্ট্যাক

কিউ+সার্কুলার কিউ

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)
ডিসজয়েট সেট(ইউনিয়ন ফাইল)
ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)
সেগমেন্ট ট্রি-১
সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)
অ্যারে কমপ্রেশন/ম্যাপিং
লোয়েস্ট কমন অ্যানসেক্টর
বাইনারি ইনডেক্সড ট্রি
স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:
এল-আর-ইউ ক্যাশ(**নতুন**)
ব্লুম ফিল্টার(**নতুন**)

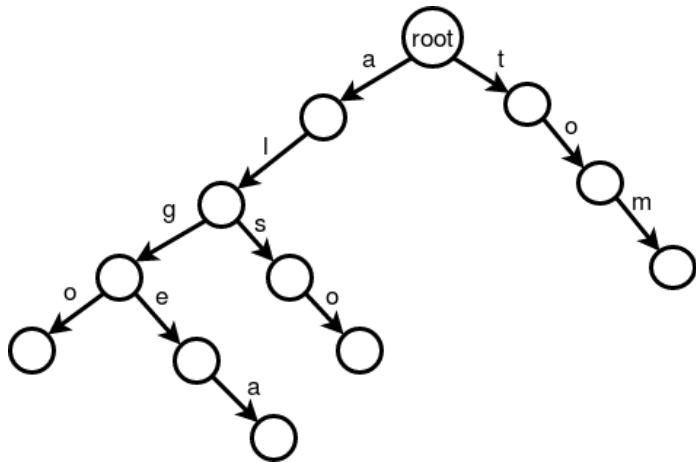


also শব্দটাকে যোগ করবো এবাব। রুট থেকে "al" প্রিফিক্স এরইমধ্যে আছে, তুধু "so" যোগ করতে হবে।



গ্রাফ থিওরি(২০):
গ্রাফ থিওরিতে হাতেখড়ি
অ্যাডজেসেন্সি ম্যাট্রিক্স
অ্যাডজেসেন্সি লিস্ট
ব্রেথড ফার্স্ট সার্চ (বিএফএস)
মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)
মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)
টপোলজিকাল সর্ট
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল
সর্ট
ডায়াক্রস্ট্রা
ফ্লয়েড ওয়ার্শল
বেলম্যান ফোর্ড

এবার “tom” যোগ করি। এবার রুট থেকে নতুন এজ তৈরি করতে হবে কারণ tom এর কোনো প্রিফিক্স আগে যোগ করা হ্যনি।



এখন “to” শব্দটা যোগ করবো কিভাবে? “to” পুরোপুরি tom এর প্রিফিক্স তাই নতুন কোনো এজ যোগ করা দরকার নাই। আমরা যে কাজটা করতে পারি সেটা বলে নোডগুলোতে কিছু এন্ড-মার্ক বসানো। যেসব নোডে এসে অন্তত একটা শব্দ কমপ্লিট হয়েছে সেসব নোডে আমরা এন্ডমার্ক বসিয়ে দেই, ছবিতে ধূসর রঙ দিয়ে এন্ডমার্ক বোঝানো হয়েছে। আগের সব শব্দের জন্য এবং সেই সাথে নতুন শব্দ “to” এর জন্য এন্ডমার্ক বসালে ট্রাইটা এরকম দেখাবে:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

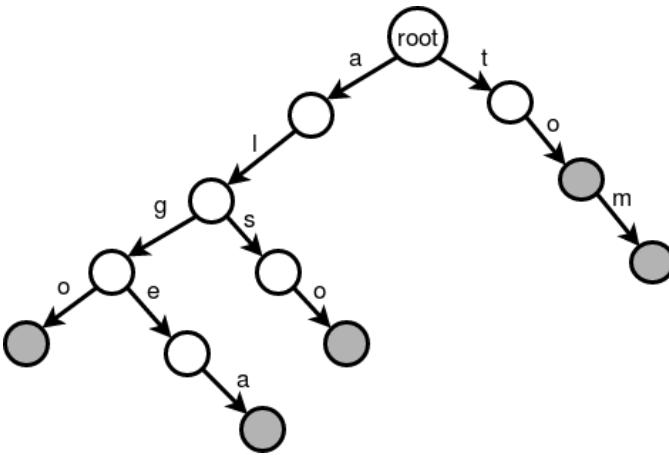
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)



নিচয়ই বুঝতে পারছো এন্ডমার্কগুলো কেন বসিয়েছি। মার্ক দেখে সহজেই বোঝা যাচ্ছে কোন কোন শব্দ ট্রাইতে আছে। কোন ক্যারেকটার নিচ্ছি সেই তথ্য থাকবে এজ এ, আর এন্ডমার্কগুলো থাকবে নোড এ।

এভাবে শব্দগুলো রাখার সুবিধা কি? ধরো তোমাকে বলা হলো “alice” শব্দটা ডিকশনারিতে আছে কিনা বলতে। তুমি শুরু থেকে ট্রাই ধরে আগামে থাকো। প্রথমে দেখো রুট থেকে a নামের এজ আছে নাকি, তারপর চেক করো a থেকে। নামের এজ আছে নাকি। এরপরে। থেকে i নামের এজ খুজে পাওয়া যাচ্ছেনা, তাই বলতে পারো alice শব্দটা নেই।

“alg” শব্দটা খুজতে দিলে তুমি root->a, a->। এবং l->g এজগুলো সবই খুজে পাবে, কিন্তু শেষ পর্যন্ত কোনো ধূসর নোডে যেতে পারবেনা, তারমানে alg ও ডিকশনারিতে নেই। “tom” খুজতে গেলে তুমি একটা ধূসর নোডে গিয়ে শেষ করবে তাই শব্দটা ডিকশনারিতে আছে।

ট্রাই ইমপ্লিমেন্ট করার সহজ একটা উপায় হলো লিংকড লিস্ট ব্যবহার করা। লিংকড লিস্ট, পয়েন্টার এসব গুনে ভয়ের কিছুই নেই, তুমি যদি লিংকলিস্ট ব্যবহার করতে অভ্যন্ত নাও হও আশা করি এই ইমপ্লিমেন্টেশনটা দেখে শিখে ফেলতে পারবে। আমাদের প্রতিটা নোডে ২টি জিনিস থাকবে:

১. এন্ড-মার্ক রাখার জন্য একটা ভ্যারিয়েবল।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্স(১):

ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

২. প্রতিটা নোড থেকে a,b,c,.....,x,y,z ইত্যাদি নামের এজ তৈরি হতে পারে। আমরা প্রতিটা ক্যারেটারের জন্য একটা পয়েন্টার রাখবো। পয়েন্টারের সাহায্যে একটা নোড আরেকটাৰ সাথে যোগ হবে। a নামের পয়েন্টার দিয়ে যোগ হলে বুঝতে হবে কারেট নোড থেকে a নামের একটা এজ আছে। শুরুতে সবগুলো পয়েন্টার “নাল” থাকবে।

আমরা প্রথমেই নোডটা তৈরি করে ফেলি:

```

1 struct node {
2     bool endmark;
3     node* next[26 + 1];
4     node()
5     {
6         endmark = false;
7         for (int i = 0; i < 26; i++)
8             next[i] = NULL;
9     }
10 } * root;
11 int main()
12 {
13     root = new node();
14     return 0;
15 }
```

`next[]` অ্যারের প্রতিটা এলিমেন্ট আরেকটা নোডকে পয়েন্ট করে। `next[0]` দিয়ে নতুন নোডকে পয়েন্ট করা হলে সেই এজ এর নাম “a”, `next[1]` এর জন্য এজ এর নাম “b”, `next[25]` এর জন্য “z”। শুরুতে সবগুলো পয়েন্টার নাল। লক্ষ্য করো নোডের ভিতর একটা কনস্ট্রাক্টর “node()” তৈরি করেছি। যখনই নতুন নোড তৈরির জন্য `new node()` কল করবো তখনই ড্যারিয়েবলগুলোকে শূন্য বা নাল বানিয়ে দিবে। এটা না দিলে গার্বেজ ড্যালু থাকতো। `root` ড্যারিয়েবলটা হলো আমাদের রুট নোড, উপরের ছবিগুলোতে লাল রঙ এর নোড। আসলে রুট একটা পয়েন্টার, যখন `root = new node();` লাইনটা এক্সিকিউট হবে তখনই একটা নতুন নোড তৈরি করে `root` যে মেমরি অ্যাড্রেসকে পয়েন্ট করে সেখানে অ্যাসাইন করে দেয়া হবে। এটাকে একটু গালভো ভাষায় বলে `instance` তৈরি করা। এবার আমাদের একটা ফাংশন লাগবে নতুন শব্দ ট্রাইতে যোগ করার জন্য:

```

1 void insert(char* str, int len)
2 {
3     node* curr = root;
4     for (int i = 0; i < len; i++) {
5         int id = str[i] - 'a';
6     }
7 }
```

কোয়ান্টাম কম্পিউটার কী?
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet
2.4K likes
শাফায়েত আশরাফ
[Like Page](#)

Be the first of your friends to like this

Shafaetsplanet about an hour ago
ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিঙের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

*i, W) = 1 if W =
i + 1, W - C[i]) -*

SHAFAEETPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / মার

```

6     if (curr->next[id] == NULL)
7         curr->next[id] = new node();
8     curr = curr->next[id];
9 }
10 curr->endmark = 1;
11 }

```

ରୁଟ ଡ୍ୟାରିସେଲ୍ଟା ଆମାଦେର ସବସମୟ ଦରକାର ହବେ ତାଇ “curr” ଏର ମଧ୍ୟେ ସେଟୋର କପି ତୈରି କରେ କାଜ କରି । ଯେହେତୁ ପ୍ୟେନ୍ଟାର ନିୟେ କାଜ କରଛି
ତାଇ ରୁଟ ଥିଲେ ନତୁନ ଏଜ ତୈରି କରା ଆର “curr” ଥିଲେ ନତୁନ ଏଜ ତୈରି କରା ଏକଇ କଥା । ଆମରା ଏଖନ ଶୁଣୁ a-z ନିୟେ କାଜ କରଛି, ତାଇ ଅୟାସକି
ଡ୍ୟାଲୁଗୁଲୋକେ 0-25 ଏ କନଭାର୍ଟ କରେ ନିବୋ ‘a’ ଏର ଅୟାସକି ଡ୍ୟାଲୁ ବିଯୋଗ କରେ । insert କରା ଖୁବ ସହଜ, ଆମରା ଶୁଣୁ ଚେକ କରବୋ କାରେନ୍ଟ ନୋଡ
(curr) ଥିଲେ ବର୍ତ୍ତମାନେ ଯେ କ୍ୟାରେକଟାରେ ଆଛି ସେଇ ନାମେର କୋନୋ ଏଜ ଆଛେ ନାକି, ନା ଥାକଲେ ନତୁନ ନୋଡ ତୈରି କରତେ ହବେ । ଏରପରେ ସେଇ ଏଜ
ଧରେ ଆମରା ପରେର ନୋଡେ ଯାବୋ । ସବାର ଶେଷ ନୋଡ଼ଟାଯ ଏନ୍-ମାର୍କ true କରେ ଦିବୋ ।

ଇନସାର୍ଟ କରାର ପର ଏଖନ ସାର୍ଟ କରବୋ । ଏଟା ଆସଲେ ଠିକ ଇନସାର୍ଟ ଏର ମତୋଇ । ପାର୍ଥକ୍ୟ ହଲେ ଯେ ଏଜଟା ଦରକାର ସେଟୋ ନା ଥାକଲେ ତୈରି କରେ ନିଛିଲାମ,
ଏଖନ ଏଜ ନା ଥାକଲେ false ରିଟାର୍ଣ୍ଣ କରେ ଦିବୋ ।

```

1 bool search(char* str, int len)
2 {
3     node* curr = root;
4     for (int i = 0; i < len; i++) {
5         int id = str[i] - 'a';
6         if (curr->next[id] == NULL)
7             return false;
8         curr = curr->next[id];
9     }
10    return curr->endmark;
11 }

```

ଲକ୍ଷ୍ୟ କରୋ ସବକାଜ ଇନସାର୍ଟ ଏର ମତୋଇ କରେଛି । ସବାର ଶେଷେ ଲାସ୍ଟ ନୋଡ଼ଟାର ଏନ୍-ମାର୍କ ରିଟାର୍ଣ୍ଣ କରେ ଦିଯେଛି । ଏନ୍-ମାର୍କ true ହଲେ ଶୁଦ୍ଧ ଆଛେ,
false ହଲେ ନାହିଁ ।

ଆମାଦେର ମୂଳ କୋଡ ଶେଷ । ଆମରା ଏଖନ ଯେକୋନୋ ଶବ୍ଦ ଟ୍ରାଈତେ ଯୋଗ କରତେ ପାରିବୋ, ଆବାର ଟ୍ରାଈ ଥିଲେ କୋନୋ ଶବ୍ଦ ଖୁଜିତେ ପାରିବୋ । ଅନେକ ସମୟ
ପ୍ରତିଟା ଟ୍ୟୁନ୍‌କେସ ଏର ଜନ୍ୟ ଟ୍ରାଈତୈରି କରତେ ଗେଲେ ମେମରି ଲିମିଟ ନିୟେ ସମସ୍ୟା ହୁଏ । ତାଇ ନିରାପଦ ଉପାୟ ହଲେ ପ୍ରତି କେସ ଏର ପର ବ୍ୟବହାର ମେମରି-

সেল গুলোকে ডিলিট করে দেয়া। শুধু root ডিলিট করলে হবেনা, প্রতিটা নোড করতে হবে। আমরা সে জন্য একটা রিকার্সিভ ফাংশন লিখতে পারি:

```
1 void del(node* cur)
2 {
3     for (int i = 0; i < 26; i++)
4         if (cur->next[i])
5             del(cur->next[i]);
6     delete (cur);
7 }
```

এই ফাংশনটা প্রতিটা নোডে গিয়ে আগে চাইল্ড গুলোকে ডিলিট করে এসে তারপর নোডটাকে ডিলিট করে দিবে।

সম্পূর্ণ কোডটা এরকম:

```
1 struct node {
2     bool endmark;
3     node* next[26 + 1];
4     node()
5     {
6         endmark = false;
7         for (int i = 0; i < 26; i++)
8             next[i] = NULL;
9     }
10 } * root;
11 void insert(char* str, int len)
12 {
13     node* curr = root;
14     for (int i = 0; i < len; i++) {
15         int id = str[i] - 'a';
16         if (curr->next[id] == NULL)
17             curr->next[id] = new node();
18         curr = curr->next[id];
19     }
20     curr->endmark = true;
21 }
22 bool search(char* str, int len)
23 {
24     node* curr = root;
25     for (int i = 0; i < len; i++) {
26         int id = str[i] - 'a';
27     }
28 }
```

```

27     if (curr->next[id] == NULL)
28         return false;
29     curr = curr->next[id];
30 }
31 return curr->endmark;
32 }
33 void del(node* cur)
34 {
35     for (int i = 0; i < 26; i++)
36         if (cur->next[i])
37             del(cur->next[i]);
38
39     delete (cur);
40 }
41 int main()
42 {
43
44     puts("ENTER NUMBER OF WORDS");
45     root = new node();
46     int num_word;
47     cin >> num_word;
48     for (int i = 1; i <= num_word; i++) {
49         char str[50];
50         scanf("%s", str);
51         insert(str, strlen(str));
52     }
53     puts("ENTER NUMBER OF QUERY");
54     int query;
55     cin >> query;
56     for (int i = 1; i <= query; i++) {
57         char str[50];
58         scanf("%s", str);
59         if (search(str, strlen(str)))
60             puts("FOUND");
61         else
62             puts("NOT FOUND");
63     }
64     del(root); //টাইটা খুঁস করে দিলাম
65     return 0;
66 }

```

কমপ্লেক্সিটি: প্রতিটা শব্দ খুজতে লুপ চালাতে হচ্ছে শব্দটার লেংথ পর্যন্ত, সার্চিং এর কমপ্লেক্সিটি $O(\text{length})$ । প্রতিটা শব্দ ইনসার্ট করার

কমপ্লেক্সিটি ও একই। মেমরি কতখানি লাগবে সেটা ডিপেন্ড করে ইমপ্লিমেন্টেশন এবং শব্দগুলোর প্রিফিক্স কতখানি ম্যাচ করে তার উপর। উপরের

ইমপ্রিমেন্টেশন দিয়ে প্রায় 10^8 টা ক্যারেকটার ট্রাইতে ইনসার্ট করা যাবে (10^8 টা ওয়ার্ড নয়, ক্যারেকটার বা লেটার)।

তুমি চাইলে ট্রাই লিংকলিস্ট ছাড়া সাধারণ অ্যারে ব্যবহার করে ইমপ্রিমেন্ট করতে পারো, নিজে চেষ্টা করো!

ট্রাই এর কিছু ব্যবহার:

১. একটা ডিকশনারিতে অনেকগুলো শব্দ আছে, কোনো একটা শব্দ আছে নাকি নাই খুজে বের করতে হবে। এই প্রবলেমটা আমরা উপরের কোডেই সলভ করেছি।

২. ধরো তোমার ৩ বন্ধুর টেলিফোন নম্বর হলো "৫৬৭৮", "৮৩২২", "৫৬৭"। তুমি যখন প্রথম বন্ধুকে ডায়াল করবে তখন ৫৬৭ চাপার সাথে সাথে ৩য় বন্ধুর কাছে ফোন চলে যাবে কাবণ ৩য় বন্ধুর নাম্বার প্রথম জনের প্রিফিক্স! অনেকগুলো ফোন নম্বর দেয়া আছে, বলতে হবে এরকম কোনো নম্বর আছে নাকি যেটা অন্য নম্বরের প্রিফিক্স। ([UVA 11362](#))।

৩. একটা ডিকশনারিতে অনেকগুলো শব্দ আছে। এখন কোনো একটা শব্দ কয়বার "prefix" হিসাবে এসেছে সেটা বের করতে হবে। যেমন "al" শব্দটা উপরের ডিকশনারিতে ৩বার প্রিফিক্স হিসাবে এসেছে (algo, algea, also এই সবগুলো শব্দের প্রিফিক্স "al")। এটা বের করার জন্য প্রতিটা নোডে একটা কাউন্টার ভ্যারিয়েবল রাখতে হবে, কোনো নোডে যতবার যাবে ততবার কাউন্টারের মান বাড়িয়ে দিবে। সার্চ করার সময় প্রিফিক্সটা খুজে বের করে কাউন্টারের মান দেখবে।

৪. মোবাইলের ফোনবন্ধুকে সার্চ করার সময় তুমি যখন কয়েকটা লেটার লিখে তখন সেই প্রিফিক্স দিয়ে কি কি নাম শুরু হয়েছে সেগুলো সার্জেশন বক্সে দেখায়। এটা তুমি ট্রাই দিয়ে ইমপ্রিমেন্ট করতে পারবে?

৫. দুটি স্ট্রিং এর "longest common substring" বের করতে হবে। (subsequence হলে ডিপি দিয়ে সহজে করা যায়, এখনে substring চেয়েছি।)

(হিন্টস: একটা স্ট্রিং এর শেষ থেকে এক বা একাধিক ক্যারেকটার নেয়া হলে সেটাকে স্ট্রিংটার সাফিক্স বলে, যেমন blog এর সাফিক্স g,og,log,blog। আর প্রতিটা substring ই কিন্তু কোনো না কোনো সাফিক্স এর প্রিফিক্স!! তাই সবগুলো সাফিক্সকে ট্রাইতে ইনসার্ট করলে কাজটা সহজ হয়ে যায়!)

৫. (অ্যাডভান্সড) সম্বত ২০১১তে ডেফোডিল ইউনিভার্সিটির ন্যাশনাল কনক্ষে এসেছিলো প্রবলেমটা। একটা ডিকশনারি ইনপুট দেয়া থাকবে।
প্রতিবার ডিকশনারির ২টা শব্দ কুয়েরি দিবে, বলতে হবে তাদের মধ্যে common prefix এর দৈর্ঘ্য কত। যেমন algo আর algea এর কমন
প্রিফিক্স alg, দৈর্ঘ্য ৩। ট্রাইতে ডিকশনারিতে ইনসার্চ করে প্রতি কুয়েরিতে শব্দদুটি এন্ড-মার্ক থেকে LCA(lowest common ancestor) বের
করে প্রবলেমটা সলভ করা যায়।

কিছু প্রবলেম:

[UVA 10226](#)

[\(UVA 11362 Phonebook\)](#)

[UVA 11488 Hyper prefix sets](#)

[POJ 2001 Shortest Prefix](#)

[POJ 1056](#)

হ্যাপি কোডিং!

ফেসবুকে মন্তব্য

0 comments

ডাটা স্ট্রাকচার: সেগমেন্ট ট্রি-১

জুন ১১, ২০১৩ by শাফায়েত



তুমি হ্যতো এরকম প্রবলেম কনটেন্ট দেখেছ, একটি ইন্টিজার অ্যাবে দেয়া আছে আর অনেকগুলো কুয়েরি দেয়া আছে। প্রতিটা কুয়েরিতে বলেছে একটা রেঞ্জের মধ্যে সবগুলো সংখ্যার যোগফল বলতে। অ্যাবের সাইজ $10^8 \times 10^8$, কুয়েরির সংখ্যাও $10^8 \times 10^8$! বুঝতেই পারছো প্রতি কুয়েরিতে লুপ চালিয়ে যোগফল বের করতে পারবেনা। কিভাবে প্রবলেমটি সলভ করবে?

এটা সলিউশন খুব সহজ, তোমাকে কিউমূলেটিভ সাম রাখতে হবে। ধরো একটি অ্যাবে আছে $sum[MAX]$, তাহলে $sum[i]$ তে রাখবে ১ থেকে i নম্বর ইনডেক্স পর্যন্ত সবগুলো সংখ্যার যোগফল। i থেকে j পর্যন্ত যোগফল বের করতে দিলে($i \leq j$) $sum[j] - sum[i-1]$ হবে তোমার উত্তর। বুঝতে না পারলে নিচের উদাহরণটা দেখো:

“ ইনপুট:

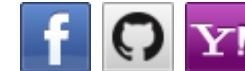
$arr[] = \{4, -9, 3, 7, 1, 0, 2\}$

cumulative sum বের করবো:

$sum[0] = 0;$

(for $i=1; i \leq n; i++$) $sum[i] = sum[i-1] + arr[i];$

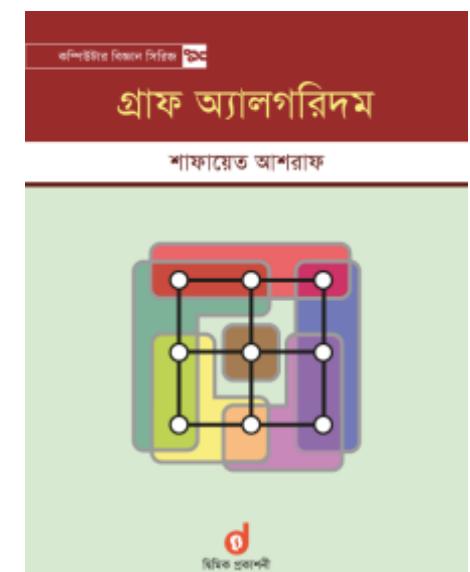
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



তাহলে cumulative sum হবে:

$sum[] = \{4, -5, -2, 5, 6, 6, 8\}$

এটা একদম বাচ্চাদের কাজ, তুমি ৫মিনিটে কোড করে ফেলতে পারবে। কিন্তু প্রবলেমসেটার তোমাকে বিপদে ফেলতে বললো কুয়েরির করার মাঝে মাঝে অ্যারেটি বদলে দেয়া হবে!! মাঝে মাঝে তোমাকে বলবে i-তম ইনডেক্সের সংখ্যাটিকে x বানিয়ে দিতে, আবার আগের মতো যোগফলও বলতে বলবে। এখন কি করবে?



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



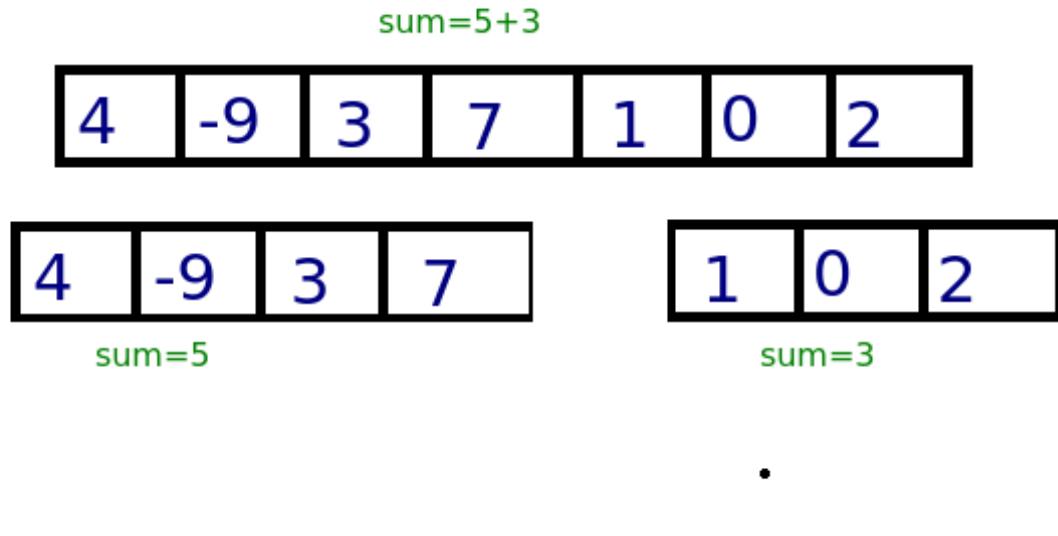
কুয়েরি: i থেকে j ইনডেক্স এর মধ্যে সবগুলো সংখ্যার যোগফল কত?

আপডেট: i তম ইনডেক্সের সংখ্যাটিকে বদলিয়ে x বানিয়ে দাও

এবার আর কিউমুলেটিভ সাম দিয়ে কাজ হবেনা, তোমার দরকার হবে সেগমেন্ট ট্রি নামের একটা ডাটা স্ট্রাকচার। ইউনিভার্সিটিতে ডাটা স্ট্রাকচার কোর্সে তোমাকে এটা পড়াবেনা, কিন্তু এটা ব্যবহার করে অনেক কাজ করা যায়।

পরের অংশে যাবার আগে তোমার কিছু জিনিস সম্পর্কে ধারণা পরিষ্কার থাকতে হবে। রিকার্শন সম্পর্কে কোনো রকম অস্পষ্টতা থাকলে আপাতত সামনে না আগানোই ভালো। এছাড়া তুমি যদি মার্জ সর্ট সম্পর্কে জানো তাহলে সেগমেন্ট বোধ তোমার জন্য খুব সহজ হয়ে যাবে। এছাড়া ট্রি সম্পর্কে যদি কিছুই না জানো তাহলে সেগমেন্ট ট্রি এখনই শেখা কি ঠিক হবে? ঠিক মার্জ সর্টের মতো সেগমেন্ট ট্রি ও “ডিভাইড এন্ড কনকোয়ার” পদ্ধতিতে কাজ করে।

ডিভাইড এন্ড কনকোয়ার পদ্ধতির মূল কথা হলো একটা প্রবলেমকে ভেঙে ছোটো ছোটো অংশ বানাও, আগে সেই ছোট অংশ সলভ করো এবং ছোটো অংশের সলিউশন থেকে বড় অংশের সলিউশন বের করো। আমরা তাই অ্যারেটাকে ২টা অংশে ভাগ করে ফেলবো এবং দুইটা ভাগের যোগফল আলাদা করে বের করবো।



তুমি যদি বাম আর ডান পাশের ভাগের যোগফল আলাদা করে বের করতে পারো তাহলে খুব সহজেই বড় অংশটার যোগফল বের করতে পারবে। আমি বলার আগেই বুঝতে পারছো এরপরে কি করবো। ছোটো অ্যারেগুলোকে আরো টুকরা করবো যতক্ষণনা ১ সাইজের টুকরা পাই। ১ সাইজের টুকরোর যোগফল আমরা জানি, সেখান থেকে বড়গুলোর যোগফল বের করে ফেলবো।

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

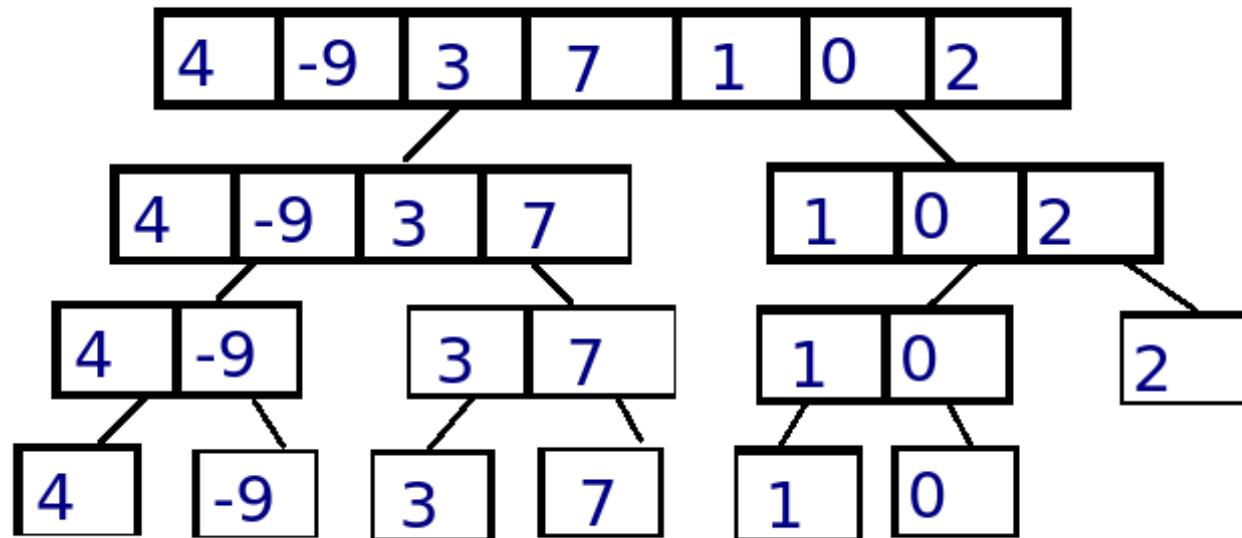
টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



Sum of a segment = sum of (Left segment + Right segment)

Sum of a leaf = value of that leaf

ছবিটা দেখতে বিদ্যুটে হলেও জিনিসটা খুবই সহজ। আমরা অ্যারেটকে ভাঁতে ভাঁতে ছোট করে ফেলেছি, সবথেকে ছোট অংশের(লিফ নোড) যোগফল আমরা জানি, সেখান থেকে বড় গুলো সহজেই বের করতে পারবো বাম এবং ডানের অংশ যোগ করে।

ছবিটায় প্রতিটা সেগমেন্টকে যদি একটা নোড ধরি তাহলে একটা ট্রি'তের হয়ে গিয়েছে, প্রতিটা নোডে আছে একটা অংশ বা বেঞ্জের যোগফল। এটার নামই সেগমেন্ট ট্রি। এখন তোমার মনে হতে পাবে এই জিনিস দিয়ে কিভাবে i থেকে j অংশের যোগফল বের করবে কারণ আমরাতো ভাঁত সম্পূর্ণ অ্যারেটা আর সবশেষে পাছিঃ সবচুকুর যোগফল। কিছুক্ষণের মধ্যে এটার উত্তর পাবে। আমরা ট্রি টাকে একটু অন্যভাবে দেখি:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

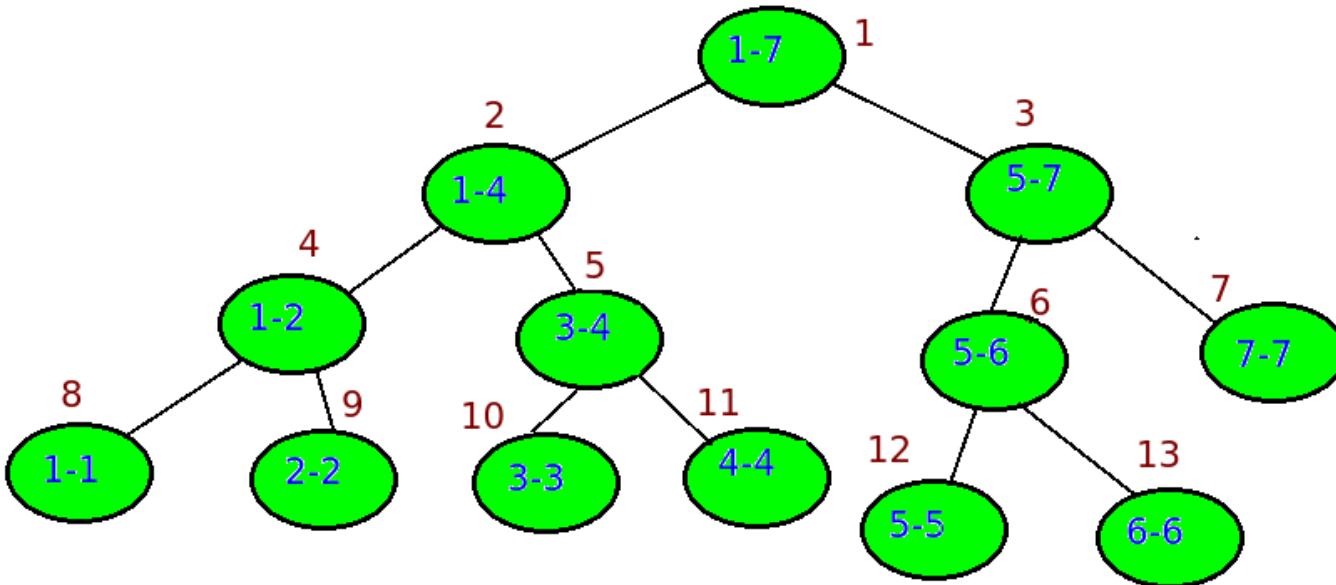
ডাইনামিক প্রোগ্রামিং ২ (শ্টেস্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন
সার্বিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১
ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্বসেট সাম,
কম্বিনেটরিক্স, ডিসিশন প্রবলেম)



খেয়াল করে দেখো আগের ট্রি টাই একেছি কিন্তু এবাব সেগমেন্টগুলো পুরোটা না দেখিয়ে শুধু রেঞ্জটা লিখেছি। যেমন ৩ নম্বর নোডে আছে ৫ থেকে ৭ ইনডেক্সের সবগুলোর যোগফল। নোডের নাম্বারিং টা গুরুত্বপূর্ণ। রুট নোড হবে ১, তার বামের নোড হবে $1*2=2$, এবং ডানের নোড হবে $(1*2+1)=3$ । অর্থাৎ x হলে বামেরটা হবে $2x$ এবং ডানেরটা $2x+1$ । বাইনারি ট্রি অ্যারেতে স্টোর করার জন্য সুবিধাজনক পদ্ধতি এটা।

এখন আগে দেখি কিভাবে এই স্টোকচারটা তৈরি করবো। নিচের কোডটি দেখো, ব্যাখ্যা আছে কোডের নিচে:

```

1 #define mx 100001
2 int arr[mx];
3 int tree[mx * 3];
4 void init(int node, int b, int e)
5 {
6     if (b == e) {
7         tree[node] = arr[b];
8         return;
9     }
10    int Left = node * 2;
11    int Right = node * 2 + 1;
12    int mid = (b + e) / 2;

```

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাম্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

```

13     init(Left, b, mid);
14     init(Right, mid + 1, e);
15     tree[node] = tree[Left] + tree[Right];
16 }
17 int main()
18 {
19 //READ("in");
20 int n;
21 cin >> n
22         repl(i, n) cin
23         >> arr[i];
24     init(1, 1, n);
25
26     return 0;
27 }

```

`tree[]` অ্যারেতে আমরা ট্রি টাকে স্টোর করবো। ট্রি অ্যারের সাইজ হবে ইনপুট অ্যারের ৩গুণ(কেন??)। `init` ফাংশনটি `arr` অ্যারে থেকে ট্রি তৈরি করে দিবে। `init` এর প্যারামিটার হলো `node,b,e`, এখানে `node`=বর্তমানে কোন নোডে আছি এবং `b,e` হলো বর্তমানে কোন রেঞ্জে আছি। শুরুতে আমরা নোড ১ এ থাকবো এবং ১-৭ রেঞ্জে থাকবো(ট্রি এর ছবিটা দেখো)।

যদি (`b==e`) হয়ে যায় তাহলে আমরা শেষ নোডে পৌছে গেছি, এখানে যোগফল হবে অ্যারেতে যে ড্যালু আছে সেটাই, সেটাকে ট্রিতে স্টোর করে রিটার্ন করে দিলাম। যদি (`b==e`) না হয় তাহলে অ্যারেটা কে দুই ভাগে ভাগ করতে হবে। বাম পাশের নোডের ইনডেক্স হবে `node*2` এবং ডান পাশেরটা `node*2+1`। এবং অ্যারেটাকে ডাঙবো ঠিক মাঝখানে। এবার রিকার্সিভলি দুই পাশে `init` কল করলে বাম এবং ডান পাশের ছোটো অংশের যোগফল বের হয়ে যাবে। দুইপাশের কাজ শেষ হয়ে গেলে বর্তমান নোডের যোগফল হবে বাম এবং ডানের নোডের যোগফল। বুঝতে সমস্যা হলে কোডটা হাতে-কলমে একবার সিমুলেট করো, তাহলেই পরিষ্কার হয়ে যাবে।

এইবার আমাদের একটা কুয়েরি ফাংশন দরকার যেটা i থেকে j এর মধ্যে সবগুলো সংখ্যার যোগফল বলে দিবে। ধরো $i=2$ এবং $j=6$ । তাহলে লক্ষ্য করো নিচের হলুদ রঞ্জের নোডগুলোর যোগফলই তোমার উত্তর:

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা


Shafaetsplanet
 2.4K likes
শাফায়েত আশরাফ
[Like Page](#)

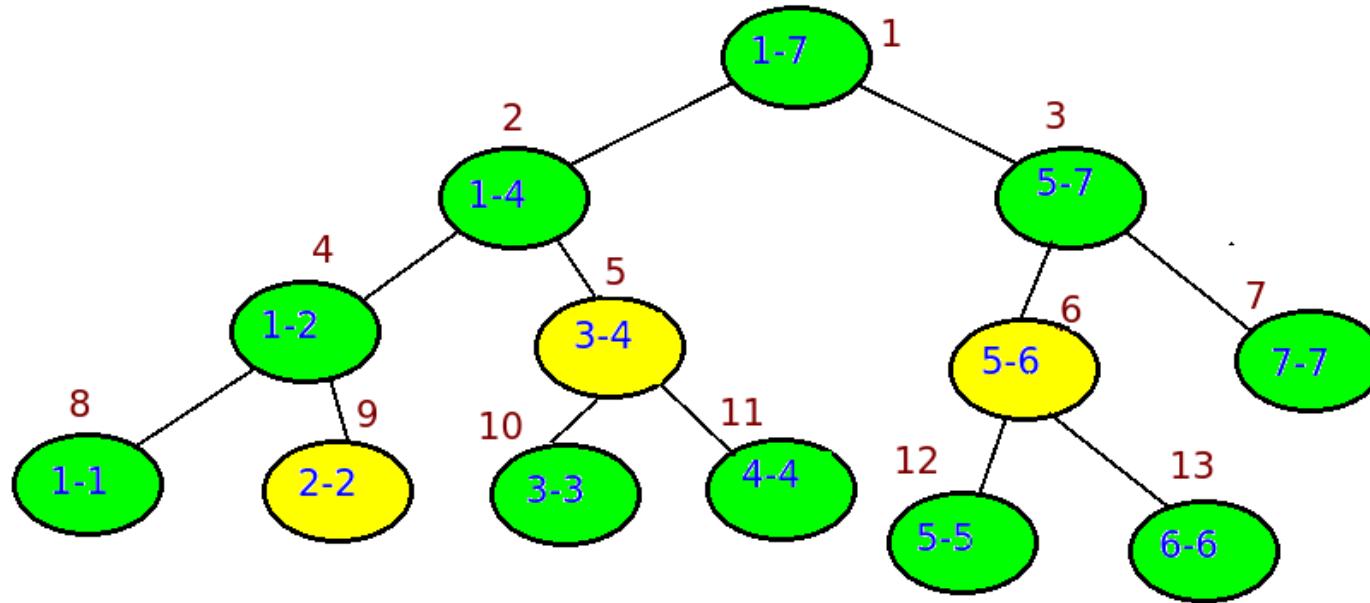
Be the first of your friends to like this


Shafaetsplanet
about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্বিনেটরিক্সের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

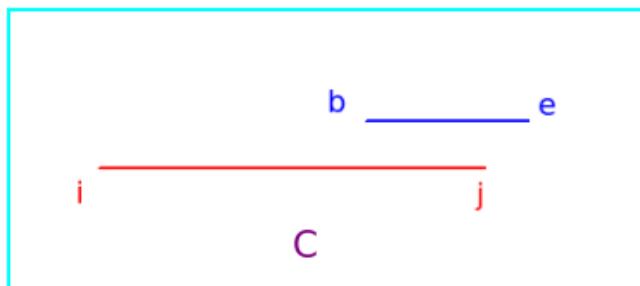
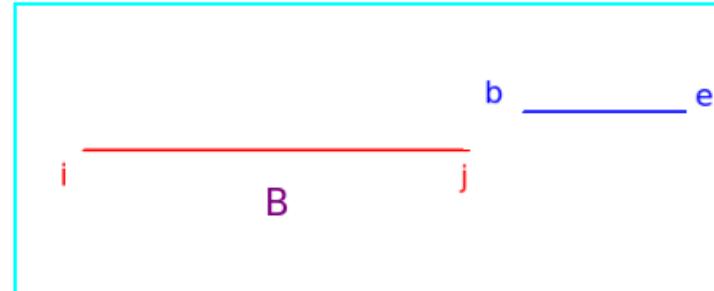
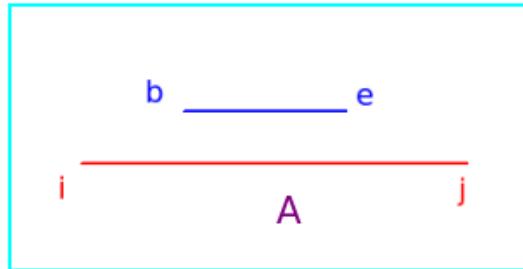
$i, W) = 1 \text{ if } W =$
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM
দাইনামিক প্রোগ্রামিং ৬ / স্যার



২ থেকে ৬ ইনডেক্সের যোগফল বের করতে হলুদ নোডগুলোর যোগফল বের করাই যথেষ্ট

2-6 রেঞ্জের জন্য হলুদ নোডগুলো আমাদের রিলেভেন্ট নোড, বাকিগুলো এক্সট্রা। আমাদের কুয়েরি ফাংশনের কাজ হবে শুধু রিলেভেন্ট নোডগুলোর যোগফল বের করা। কোডটা init ফাংশনের মতোই হবে তবে কিছু কন্ডিশন অ্যাড করতে হবে। ধরো তুমি এমন একটা নোডে আছে যেখানে b-e রেঞ্জের যোগফল আছে। তুমি এই নোডটা রিলেভেন্ট কিনা সেটা কিভাবে বুবে? এখানে ৩ধরণের ঘটনা ঘটতে পারে:



কেস A: ($b \geq i \& e \leq j$) এরকম হলে কারেন্ট সেগমেন্টটা পুরোটাই $i-j$ এর ভিতরে আছে, সেগমেন্টটা রিলেডেন্ট।

কেস B: ($i > e \mid\mid j < b$) এরকম হলে কারেন্ট সেগমেন্টটা পুরোটাই $i-j$ এর বাইরে আছে, এই সেগমেন্টটা নেয়ার দরকার নাই।

কেস C: কেস A,B সত্য না হলে এই সেগমেন্টের কিছু অংশ $i-j$ এর মধ্যে, সেগমেন্টটাকে আরো ডেঙে নিচে গিয়ে রিলেডেন্ট অংশটা নিতে হবে।

তাহলে আমরা কুয়েরি ফাংশনে প্রতি নোডে গিয়ে দেখবো সেগমেন্টটা রিলেডেন্ট নাকি। যদি রিলেডেন্ট হয় তাহলে সেই নোডের যোগফল রিটার্ন করবো, যদি বাইরে চলে যায় তাহলে ০ রিটার্ন করে দিবো, অন্যথায় আমরা সেগমেন্টটা আরো ডেঙে রিলেডেন্ট অংশ খুজবো।

```

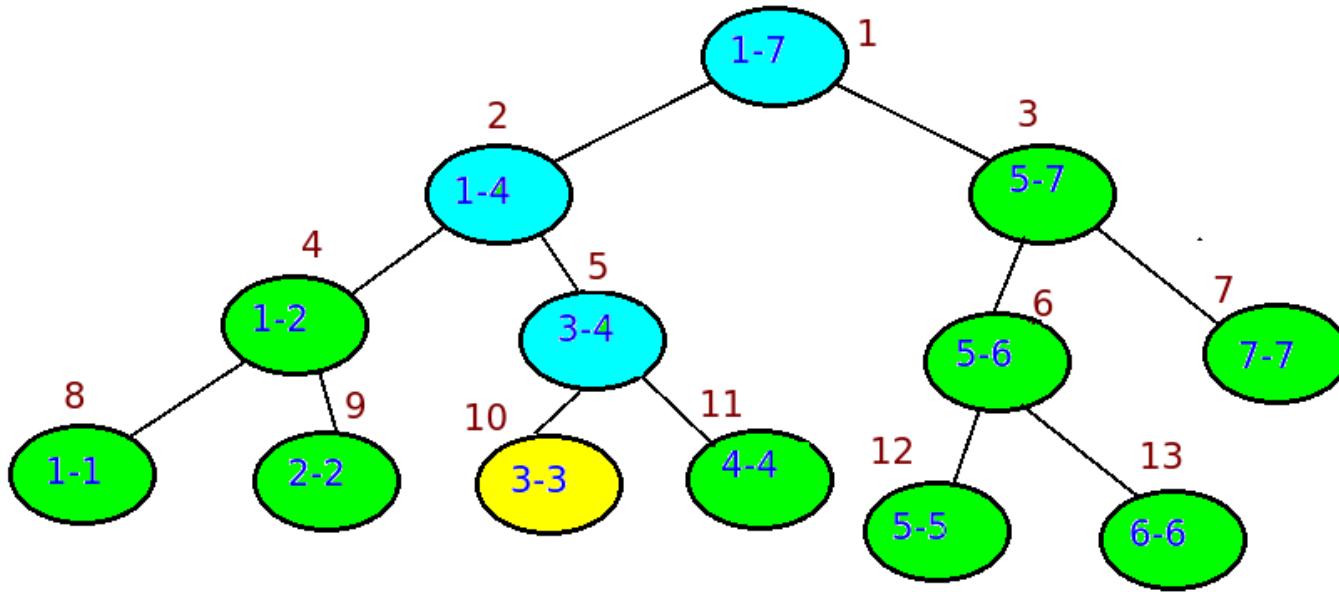
1 int query(int node, int b, int e, int i, int j)
2 {
3     if (i > e || j < b)
4         return 0; //বাইরে চলে গিয়েছে
5     if (b >= i && e <= j)
6         return tree[node]; //রিলেডেন্ট সেগমেন্ট
7     int Left = node * 2; //আরো ডাঙতে হবে

```

```
8     int Right = node * 2 + 1;
9     int mid = (b + e) / 2;
10    int p1 = query(Left, b, mid, i, j);
11    int p2 = query(Right, mid + 1, e, i, j);
12    return p1 + p2; //বাম এবং ডান পাশের যোগফল
13 }
```

init ফাংশনের মতোই কাজ করে কুয়েরি ফাংশনটা। i,j হলো যে রেঞ্জের যোগফল বের করতে হবে সেটা আর b,e হলো কারেন্ট নোডে যে রেঞ্জের যোগফল আছে সেটা।

সবশেষে আপডেট করা, যার জন্য কিউমুলিটিভ সাম ব্যবহার না করে ট্রি বানিয়েছি। তোমাকে বললো i=3 নম্বর ইনডেক্সের ড্যালু x=10 করে দিতে। তারমানে ট্রি তে যেই নোডে 3-3 রেঞ্জের যোগফল আছে সেটা আপডেট করে দিবো(নিচের ছবির হলুদ নোড)। নোডটির ড্যালু আপডেট হলে পথে যেসব নোড ছিলো(নীল নোড) সবগুলোর যোগফল বদলে যাবে, বাকি নোডগুলোর কোনো পরিবর্তন হবেনা কারণ 3 নম্বর নোড সেগুলো রেঞ্জের বাইরে।



যে নোডটি আপডেট করবো সেই নোডে পৌছানোর পথের সবগুলো নোড আপডেট হয়ে যাবে

আপডেটের কোডেও খুব বেশি পার্থক্য নেই:

```

1 void update(int node, int b, int e, int i, int newvalue)
2 {
3     if (i > e || i < b)
4         return; //বাইরে ছলে গিয়েছে
5     if (b >= i && e <= i) { //বিলেভেট সেগমেন্ট
6         tree[node] = newvalue; //নতুন মান বসিয়ে দিলাম
7         return;
8     }
9     int Left = node * 2; //আরো ডাঙ্গতে হবে
10    int Right = node * 2 + 1;
11    int mid = (b + e) / 2;
12    update(Left, b, mid, i, newvalue);
13    update(Right, mid + 1, e, i, newvalue);
14    tree[node] = tree[Left] + tree[Right];
15 }
```

i নম্বর ইনডেক্সে আপডেট করবো, এক্সট্রা সেগমেন্টগুলো শুরুতেই বাদ দিয়ে দিয়েছি। রিলেভেন্ট সেগমেন্টে গেলে নতুন মান বসিয়ে দিয়েছি, এইখানে কন্ডিশনটা `if(b==e)` লিখলেও চলতো কারণ সবসময় লিফ নোডে আপডেট করছি আমরা।

সেগমেন্ট ট্রি তো মোটামুটি এই টটা ফাংশন সবসময় থাকে `init,query,update`। অনেক সময় `init` এর কাজটা আপডেট দিয়ে করে ফেলা যায়। যেমন এখানে তুমি `init` কল করে ট্রি না বানিয়ে প্রতিটা নোড আলাদা করে আপডেট করে ট্রি বানাতে পারতে। টটা ফাংশন মিলিয়ে কোডটা হবে:

```
1 #define mx 100001
2 int arr[mx];
3 int tree[mx * 3];
4 void init(int node, int b, int e)
5 {
6     if (b == e) {
7         tree[node] = arr[b];
8         return;
9     }
10    int Left = node * 2;
11    int Right = node * 2 + 1;
12    int mid = (b + e) / 2;
13    init(Left, b, mid);
14    init(Right, mid + 1, e);
15    tree[node] = tree[Left] + tree[Right];
16 }
17 int query(int node, int b, int e, int i, int j)
18 {
19     if (i > e || j < b)
20         return 0; //বাইরে ছলে গিয়েছে
21     if (b >= i && e <= j)
22         return tree[node]; //রিলেভেন্ট সেগমেন্ট
23     int Left = node * 2; //আবো ভাঁতে হব
24     int Right = node * 2 + 1;
25     int mid = (b + e) / 2;
26     int p1 = query(Left, b, mid, i, j);
27     int p2 = query(Right, mid + 1, e, i, j);
28     return p1 + p2; //বাম এবং ডান পাশের যোগফল
29 }
30 void update(int node, int b, int e, int i, int newvalue)
31 {
32     if (i > e || i < b)
33         return; //বাইরে ছলে গিয়েছে
```

```

34     if (b >= i && e <= i) { //বিলডেট সেগমেন্ট
35         tree[node] = newvalue;
36         return;
37     }
38     int Left = node * 2; //আবো ভাখতে হবে
39     int Right = node * 2 + 1;
40     int mid = (b + e) / 2;
41     update(Left, b, mid, i, newvalue);
42     update(Right, mid + 1, e, i, newvalue);
43     tree[node] = tree[Left] + tree[Right];
44 }
45 int main()
46 {
47     READ("in");
48     int n;
49     cin >> n;
50     repl(i, n)
51         cin
52         >> arr[i];
53     init(1, 1, n);
54
55     update(1, 1, n, 2, 0);
56     cout << query(1, 1, n, 1, 3) << endl;
57     update(1, 1, n, 2, 2);
58     cout << query(1, 1, n, 2, 2) << endl;
59     return 0;
60 }
```

সেগমেন্ট ট্রি অ্যারেকে বারবার ২ভাগে ভাগ করে, ট্রি এর গভীরতা হবে সর্বোচ্চ $\log(n)$ তাই প্রতিটা কুয়েরি আব আপডেটের কমপ্লেক্সিটি $O(\log n)$ । `init` ফাংশনে ট্রি এর প্রতিটা নোডেই একবার যেতে হয়েছে তাই সেক্ষেত্রে কমপ্লেক্সিটি হবে প্রায় $O(n \log n)$ ।

সেগমেন্ট ট্রি তুমি তখনই ব্যবহার করতে পারবে যখন দুইটা ছোটো সেগমেন্টকে একসাথে করে বড় সেগমেন্টের ফলাফল বের করা যায়। যোগফল ছাড়াও একটা রেঞ্জের মধ্যে সর্বোচ্চ বা সবনিম্ন মান তুমি বের করতে পারবে, বামপাশের সর্বোচ্চ মান এবং ডানপাশের সর্বোচ্চ মান জানলে রুট নোডেরটাও বের করা যায় খুবই সহজে।

এখনে একটা গুরুত্বপূর্ণ জিনিস বাদ পড়েছে। ধরো তোমাকে একটা ইনডেক্সে আপডেট করতে না বলে। থেকে j ইনডেক্সে আপডেট করতে বললো, তাহলে কি করবে? প্রতিটা লিফ নোডে আলাদা করে আপডেট করলে O(nlogn) হয়ে যাবে কমপ্লেক্সিটি যেটা TLE দিবে। এটার জন্য খুবই এলিগেন্ট একটা টেকনিক আছে যার নামে অলস(লেজি) প্রপারেশন! সে সম্পর্কে পরের পরে জানবো, এখন তুমি যতটুকু শিখেছো সেটা দিয়ে array queries(lightoj), Multiple of 3 (SPOJ), Frequent Values (UVA), Curious Robin Hood(lighoj) প্রবলেমটা সলভ করো।

পরের পর্ব- লেজি প্রপারেশন

ফেসবুকে মন্তব্য

3 comments

3 Comments

Sort by Oldest



Add a comment...



Dipto Barcelonist

The problem link of LightOJ should be changed

Like · Reply · 1 · 1y



Abdur Rahim

YES, the array queries

Like · Reply · 49w · Edited



Abdur Rahim

I took 5 minuite to understand that !!!

if(b>=i && e<=i) is same as if(b==e && e==i)

ডাটা স্টুকচার: সেগমেন্ট ট্রি-২ (লেজি প্রপাগেশন)

জুলাই ১৮, ২০১৩ by শাফায়েত



সেগমেন্ট ট্রির সবথেকে এলিগেট অংশ হলো লেজি প্রপাগেশন টেকনিক। আমরা **আগের পরে** যে সেগমেন্ট ট্রি যোভাবে আপডেট করেছি তাতে একটা বড় সমস্যা ছিলো। আমরা একটা নির্দিষ্ট ইনডেক্স আপডেট করতে পেরেছি, কিন্তু একটা বেঞ্জের মধ্যে সবগুলো ইনডেক্স আপডেট করতে গেলেই বিপদে পরে যাবো। সে কারণেই আমাদের লেজি প্রপাগেশন শিখতে হবে, প্রায় সব সেগমেন্ট ট্রি প্রবলেমেই এই টেকনিকটা কাজে লাগবে। এই পর্বটা পড়ার আগে অবশ্যই তোমাকে সেগমেন্ট ট্রির একদুটি প্রবলেম সলভ করে আসতে হবে, এছাড়া তোমার বুঝতে সমস্যা হবে। **আগের পরে** লেজি প্রপাগেশন দরকার হয়না এমন কয়েকটি প্রবলেম দিয়েছি, আগে সেগুলো সলভ করতে হবে।

তোমাকে একটি অ্যারে দেয়া আছে n সাইজের এবং তোমাকে কুয়েরি করা হচ্ছে i থেকে j ইনডেক্সের মধ্যে সবগুলো এলিমেন্টের যোগফল বলতে হবে। আর আপডেট অপারেশনে তোমাকে বলা হলো i থেকে j ইনডেক্সের মধ্যে সবগুলো সংখ্যার সাথে একটি নির্দিষ্ট সংখ্যা x যোগ করতে।

যেমন যদি অ্যারেটা শুরুতে হ্যাতো ছিলো এরকম:

“

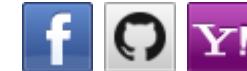
4 1 2 3 9 8 7

তাহলে $i=3, j=5$ ইনডেক্সের মধ্যে সবগুলো সংখ্যার সাথে ২ যোগ করলে অ্যারেটা হবে:

4 1 2+2 3+2 9+2 8 7

আবার $i=3, j=4$ ইনডেক্সের মধ্যে সবগুলো সংখ্যার যোগফল হবে $2+2+3+2=9$ ।

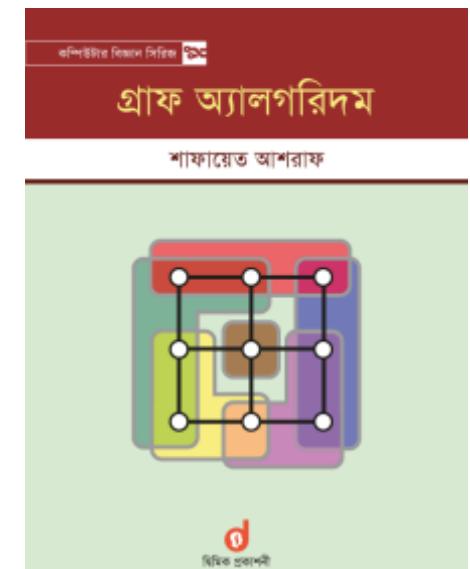
সাবস্ক্রাইব



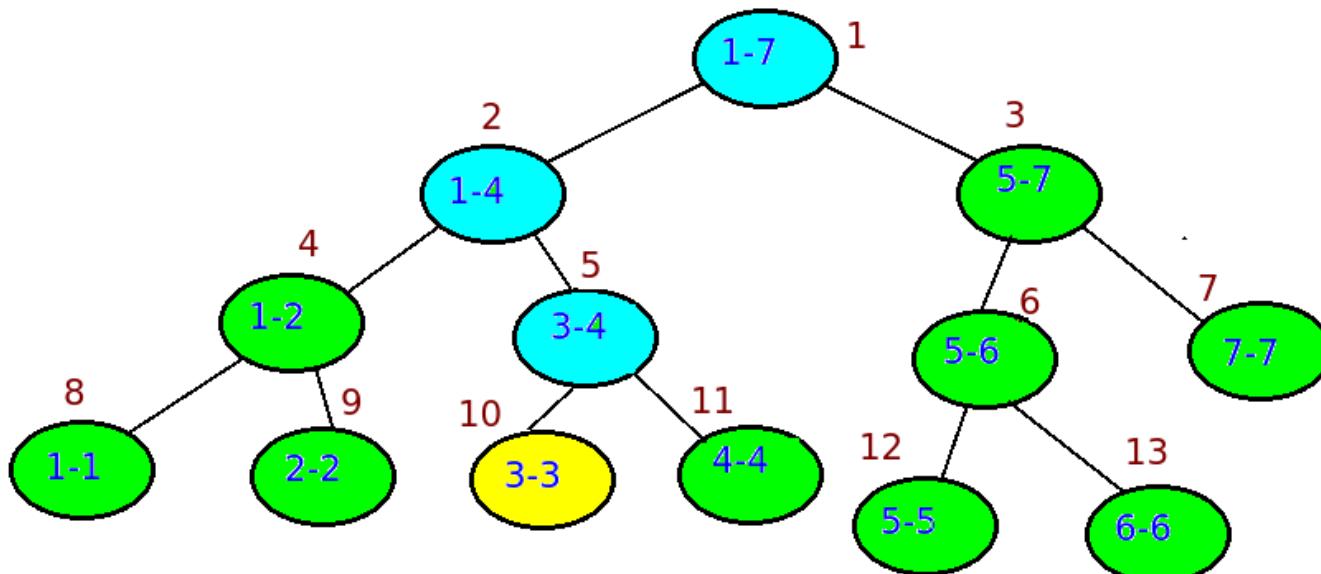
Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



আগের প্রবলেমে আমরা খালি একটা ইনডেক্স আপডেট করেছিলাম। তখন আমি শুধু ৩ নম্বর ইনডেক্সে আপডেট দেখানোর জন্য এই ছবিটা একেছিলাম:



যে নোডটি আপডেট করবো সেই নোডে পৌছানোর পথের সবগুলো নোড আপডেট হয়ে যাবে

আমরা সেগমেন্ট ট্রি এর একদম নিচে গিয়ে ৩ যেখানে আছে সেই নোডটা আপডেট করেছি এবং সেই পথের বাকিনোডগুলোকেও আপডেট করে দিয়েছি উপরে ওঠার সময়।

এখন তোমাকে $i=1$ এবং $j=8$ এই রেঞ্জের সবার সাথে x যোগ করতে বলা হলো। একটা সলিউশন হতে পারে তুমি আগের মতো করেই ১, ২, ৩, ৪ ইত্যাদি ইনডেক্স আলাদা করে আপডেট করলে। তাহলে প্রতি আপডেটে কমপ্লেক্সিটি $\log n$ এবং সর্বোচ্চ n টি আপডেটের জন্য $n \log n$ সময় লাগবে। এটা খুব একটা ভালো সলিউশন না, আমরা এখানে $\log n$ এই আপডেট করতে পারি।

কিছু ডেফিনিশন:

যেকোনো ট্রি স্ট্রাকচারে লিফ নোড হলো সবথেকে নিচের নোডগুলো। লিফ ছাড়া বাকি সবনোড হলো ইন্টারনাল নোড।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

সেগমেন্ট ট্রি তে লিফ নোড গুলোতে কি থাকে? সেখানে থাকে কোনো একটি ইনডেক্সের অরিজিনাল ভ্যালু।

সেগমেন্ট ট্রি তে ইন্টারনাল(লিফ ছাড়া বাকি সব) নোডগুলোতে কি থাকে? সেখানে থাকে নিচে যতগুলো লিফ নোড আছে সবগুলোর মার্জ করা ফলাফল।

সেগমেন্ট ট্রি তে একটা নোডের বেঞ্জ হলো সেই নোডে যেসব ইনডেক্সের মার্জ করা বেজাল্ট আছে। যেমন ছবিতে ৩ নম্বর নোডের বেঞ্জ ৫ থেকে ৭।

যেমন উপরের ছবিতে ১০ নম্বর নোডে আছে ৩ নম্বর ইনডেক্সের ভ্যালু আর ২ নম্বর নোডে আছে ১, ২, ৩, ৪ নম্বর নোডের যোগফল। কি দরকার এতগুলো লিফ নোডে কষ্ট করে নিয়ে আপডেট করে আসা? তার থেকে আমরা কি ২ নম্বর নোডে এসে সেখানে এই ইনফরমেশনটা রেখে দিতে পারিনা “কারেন্ট নোডের নিচের সবগুলো ইনডেক্সের সাথে X যোগ হবে”? তার মানে যখন দেখছি একটা নোডের বেঞ্জ যখন পুরোপুরি কুয়েরির ডিতরে থাকে তখন সেই নোডের নিচে আর না নিয়ে সেখানে প্রপাগেশন ভ্যালুটা সেভ করে রাখতে পারি। একটা নোডের প্রপাগেশন ভ্যালু হলো সেই নোডের বেঞ্জের মধ্যে সব ইনডেক্সের মধ্যে যে ভ্যালুটা যোগ হবে সেটা।

নিচের ছবিতে দেখো কিভাবে এই এক্সট্রা ইনফরমেশনটা সেভ করা হচ্ছে:

স্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

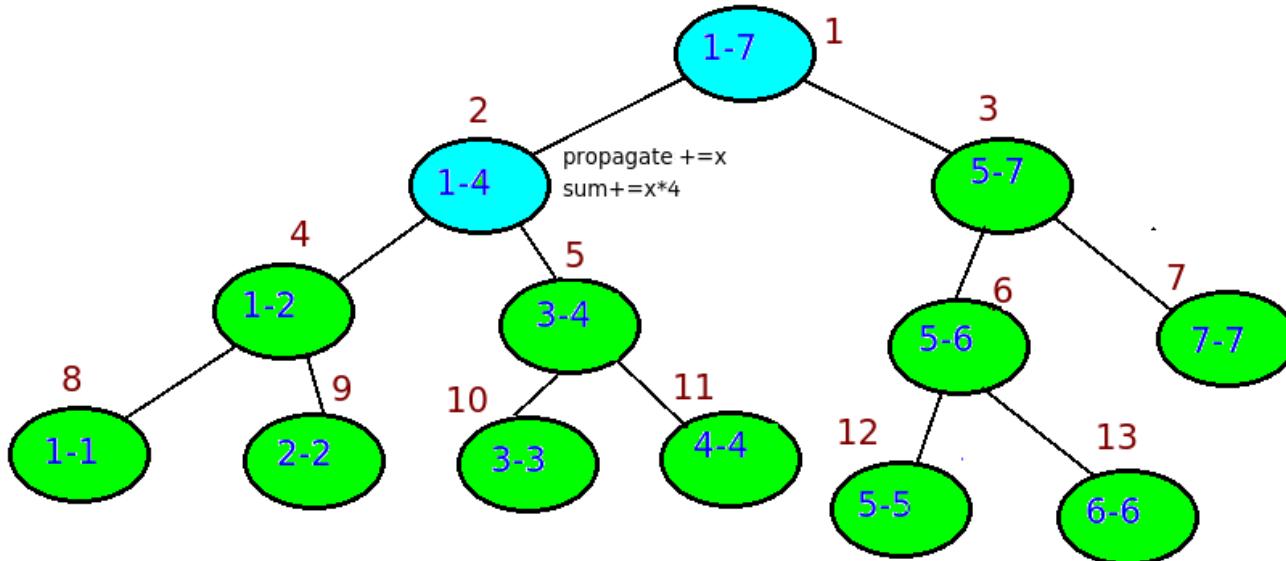
টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল
সর্ট

ডায়াক্স্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



- ১ থেকে ৮ ইনডেক্সের সবার সাথে x যোগ করতে চাই
- ২ নম্বর নোডে গিয়ে বলে দিলাম নিচের সবার সাথে x যোগ হবে
- ২ নম্বর নোডের নিচে ৪ টি লিফ নোড আছে, সবার সাথে x যোগ করলে ২ নম্বর নোডের সাম এর সাথে $4x$ যোগ হবে।

প্রতিটি নোডে যোগফল ছাড়াও আরেকটা ভ্যারিয়েবল রাখতে হবে যেটার নাম দিয়েছি propagate। এই ভ্যারিয়েবলটার কাজ হলো তার নিচের লিফ নোডগুলোর সাথে কত যোগ করতে হবে তার হিসাব রাখা। propagate এর মান শুরুতে থাকে শূন্য। এরপর যে রেঞ্জটা আপডেট করতে বলবে সেই রেঞ্জের “রিলেভেন্ট নোড” গুলোতে গিয়ে propagate এর সাথে x যোগ করে আসবো। (আগের পরেই জেনেছো “রিলেভেন্ট নোড” হলো যেসব নোডের রেঞ্জ পরোপুরি কুয়েরির ডিটাৰে আছে)

আরেকটা উদাহরণ, ২ থেকে ৬ নম্বর নোড যদি আপডেট করতে হয় তাহলে হলুদ নোডগুলোতে গিয়ে বলে দিবো নিচের নোডগুলোর সাথে x যোগ করতে:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্রিমাম ফ্লো-১

ম্যাক্রিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেষ্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

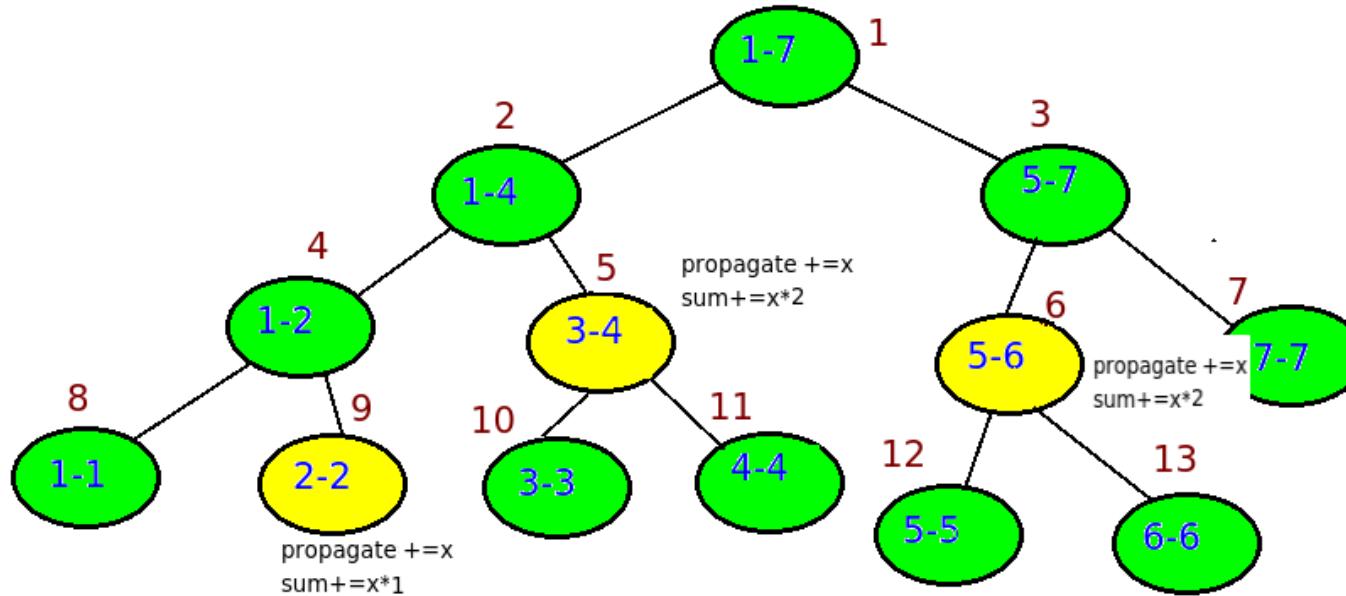
ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেষ্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিনেটরিঙ্গ, ডিসিশন প্রবলেম)



আগের আপডেট ফাংশনের মতোই কোনো একটা নোড আপডেট করার পর সেই পথের সবগুলো নোড আপডেট করে উঠতে হবে। আমরা কোডটা দেখি:

```

1 struct info {
2     i64 prop, sum;
3 } tree[mx * 3]; //sum ছাড়াও নিচে অতিরিক্ত কত যোগ হচ্ছে সেটা বাখে prop এ
4 void update(int node, int b, int e, int i, int j, i64 x)
{
    if (i > e || j < b)
        return;
    if (b >= i && e <= j) //নোডের বেঁজে আপডেটের বেঁজের ভিতরে
    {
        tree[node].sum += ((e - b + 1) * x); //নিচে নোড আছে e-b+1 টি, তাই e-b+1 বার x যোগ হবে এই বেঁজে
        tree[node].prop += x; //নিচের নোডগুলোর সাথে x যোগ হবে
        return;
    }
    int Left = node * 2;

```

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

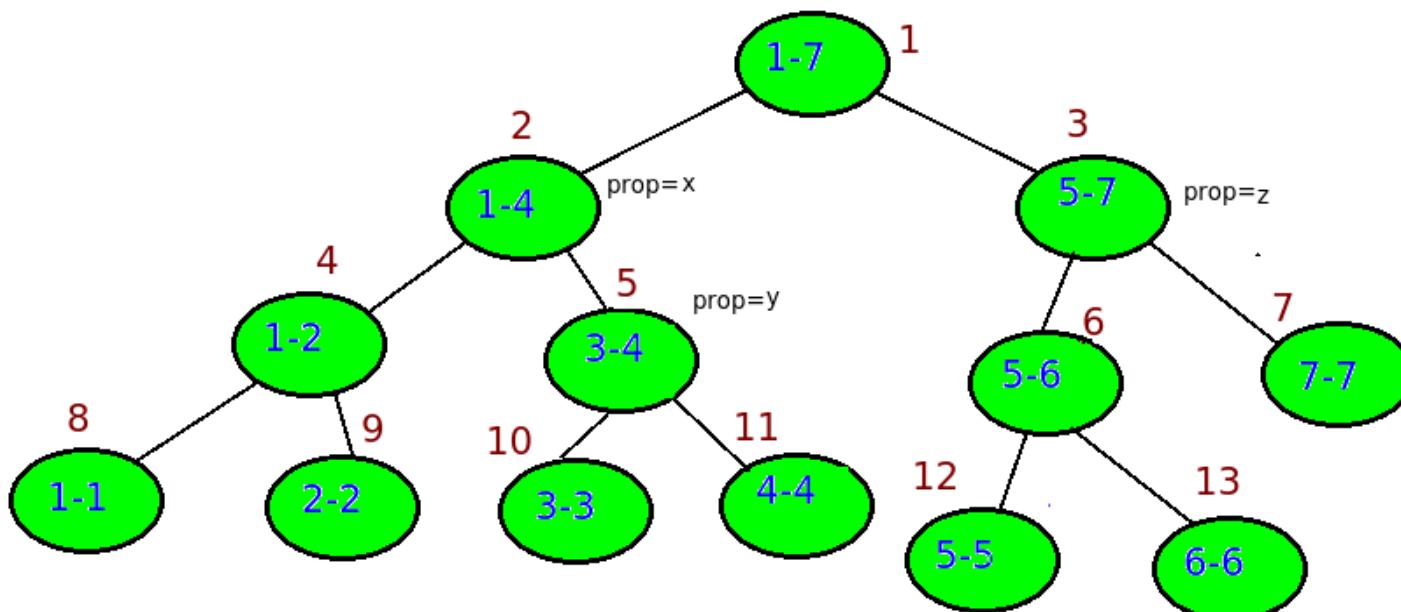
```

15 int Right = (node * 2) + 1;
16 int mid = (b + e) / 2;
17 update(Left, b, mid, i, j, x);
18 update(Right, mid + 1, e, i, j, x);
19 tree[node].sum = tree[Left].sum + tree[Right].sum + (e - b + 1) * tree[node].prop;
20 //উপরে উঠার সময় পথের নোডগুলো আপডেট হবে
21 //বাম আর ডান পাশের সাথে ছাড়াও যোগ হবে নিচে অতিরিক্ত যোগ হওয়া মান
22 }

```

আগের আপডেট ফাংশনের সাথে পার্থক্য হলো এখন একটা বেঞ্জে আপডেট করছি এবং কোনো নোডের বেঞ্জে আপডেট বেঞ্জের ভিতরে হলে আমরা নিচে না গিয়ে বলে দিচ্ছি যে নিচের ইনডেক্সগুলোতে x যোগ হবে।

এখন মনে করো আমরা বেশ কয়েকবার আপডেট ফাংশন কল করেছি। নোডগুলোর প্রপগেটেড ড্যালগুলা আপডেট হয়ে নিচের মতো হয়েছে:



কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Shafaetsplanet

2.4K likes

ফাফ অ্যালগরিদম

শাফায়েত আশরাফ

[Like Page](#)

Be the first of your friends to like this



Shafaetsplanet



about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্বিনেটরিক্যাল প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

$i, W) = 1 \text{ if } W =$
 $i + 1, W - C[i]) -$

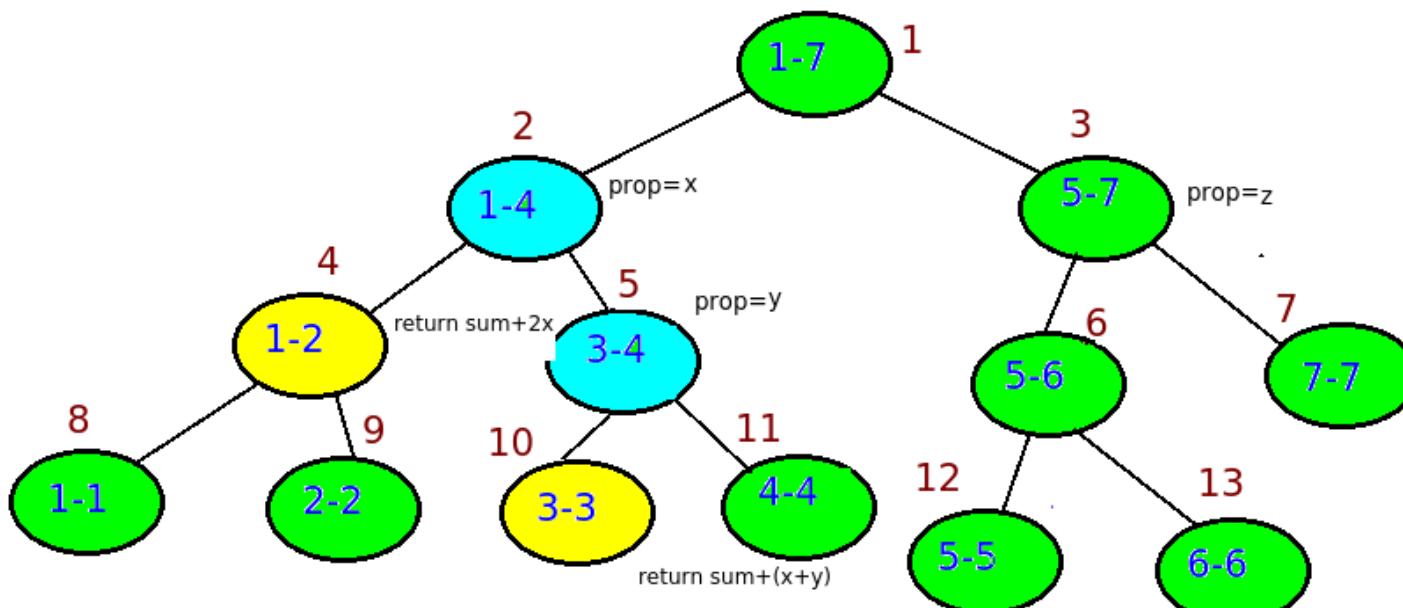
SHAFAEETPLANET.COM

দাইনামিক প্রোগ্রামিং ৬ / মোবার

যেসব নোডের ভ্যালু লিখিনি সেগুলোতে শুণ্য আছে মনে করো। তাহলে উপরের ছবির মানে হলো ১-৪ ইনডেক্সের সাথে x যোগ হবে, ৫-৭ এর সাথে z যোগ হবে ইত্যাদি।

এবাব প্রশ্ন হলো কুয়েরি করবো কিভাবে?

ধরো আমাদের কুয়েরির রেঞ্জ হলো ১-৩। সাধারণভাবে আমরা আমাদের রিলেডেন্ট নোড ৪ আর ১০ থেকে ভ্যালু নিয়ে যোগ করে দিতাম। কিন্তু আমরা জানি ২ নম্বর নোডের নিচের সবাব সাথে x যোগ হয়েছে, তাই ৪ নম্বর নোডের নিচের সবাব সাথেও x যোগ হয়েছে! তাই আমরা ৪ নম্বর রেঞ্জে রাখা ভ্যালুর সাথে যোগ করে দিবো $2*x$, কাৰণ ৪ নম্বর নোডের রেঞ্জে ২টি ইনডেক্স আছে এবং তাদের সবাব সাথে x যোগ হয়েছে। ঠিক একই ভাবে যেহেতু ২ এবং ৫ নম্বর নোডের নিচে আছে ১০, তাই ১০ নম্বর নোডের রেঞ্জে ১টি ইনডেক্স আছে এবং তাৰ সাথে যোগ হবে $(x+y)$ ।



তাৰমানে এটা পৰিষ্কাৰ যে কুয়েরি কৰা সময় কোনো নোডে যাবাৰ সময় উপৰেৰ প্ৰপাগেটেড ভ্যালু গুলোৰ যোগফল সাথে কৰে নিয়ে যেতে হবে।

তাহলে কুয়েরিৰ ফাংশনে carry নামেৰ একটা প্যারামিটাৰ যোগ কৰে দেই যাৰ কাজ হবে ভ্যালুটা বয়ে নিয়ে যাওয়া:

```
1 int query(int node, int b, int e, int i, int j, int carry = 0)
```

```

2 {
3     if (i > e || j < b)
4         return 0;
5
6     if (b >= i and e <= j)
7         return tree[node].sum + carry * (e - b + 1); //সাম এর সাথে যোগ হবে সেই রেঞ্জের সাথে অতিরিক্ত যত যোগ কর
8
9     int Left = node << 1;
10    int Right = (node << 1) + 1;
11    int mid = (b + e) >> 1;
12
13    int p1 = query(Left, b, mid, i, j, carry + tree[node].prop); //প্রপাগেট ভ্যালু বয়ে নিয়ে যাচ্ছে carry ভ্যারিয়েব
14    int p2 = query(Right, mid + 1, e, i, j, carry + tree[node].prop);
15
16    return p1 + p2;
17 }

```

আগের কোডের সাথে ডিফারেন্স হচ্ছে carry প্যারামিটারে এবং রিটার্নভ্যালুতে। শুরুতে হিসাব করে রাখা sum এর সাথে যোগ হচ্ছে অতিরিক্ত যে ভ্যালু যোগ হয়েছে সেটা।

মোটামুটি এই হলো লেজি প্রপাগেশনের কাহিনী। সামারী করলে দাঢ়ায়:

- “ ১. রেঞ্জের আপডেট $O(\log n)$ এ করতে চাইলে লেজি প্রপাগেশন ব্যবহার করতেই হবে।
- ২. লেজি প্রপাগেশনের কাজ হলো লিফ নোডে আপডেট না করে আগেই কোনো নোড আপডেট রেঞ্জের ডিতরে পড়লে সেখানে বলে দেয়া নিচের ইনডেক্সগুলো কিভাবে আপডেট হবে।
- ৩. কুয়েরি করার সময় উপরের নোডগুলোতে সেড করা প্রপাগেশন ভ্যালুগুলো রিলেডেট নোড ক্যারি করে নিয়ে আসতে হবে এবং সেই অনুযায়ী ভ্যালু রিটার্ন করতে হবে।

অনেক সময় প্রবলেমে বলতে পারে একটা রেঞ্জের মধ্যে সবগুলো সংখ্যাকে x দিয়ে বদলে দিতে (x যোগ নয়), সেক্ষেত্রে প্রপাগেশন ভ্যালু হিসাবে রাখতে হবে নিচের সবনোডকে কোন ভ্যালু দিয়ে বদলে দিতে হবে সেটা। কুয়েরি করার সময় carry ভ্যালুতেও সামান্য পরিবর্তন আসবে, কিরকম

পরিবর্তন আসবে সেটা বের করার দায়িত্ব তোমার উপরে ছেড়ে দিলাম।

সেগমেন্ট ট্রির মূল টিউটোরিয়াল এখানেই শেষ। পরবর্তী কোনো পর্বে ২ডি সেগমেন্ট ট্রি এবং কিছু প্রবলেম নিয়ে আলোচনা করবো।

টিউটোরিয়ালের কোনো অংশ বুঝতে সমস্যা হলে ইমেইলে যোগাযোগ করতে পারো বা আরো ডালো হয় মন্তব্য অংশে সেটা জানালে।

প্র্যাকটিস প্রবলেম:

HORRIBLE

LITE

[Lightoj Segment Tree Section](#)

[আগের পর](#)

ফেসবুকে মন্তব্য

2 comments

অ্যারে কম্প্রেশন

ডিসেম্বর ০১, ২০১২ by শাফায়েত



খুবই সহজ কিন্তু দরকারি একটা টেকনিক নিয়ে আলোচনা করবো আজকে। STL এর ম্যাপ ব্যবহার করে আমরা অ্যারে কম্প্রেশন করবো। মনে করো কোনো একটা প্রবলেমে তোমাকে বলা হলো একটি অ্যারেতে ১ লাখটা সংখ্যা দেয়া থাকবে যাদের মান হবে ০ থেকে সর্বোচ্চ ১০০০। এখন যেকোনো আমি একটি সংখ্যা বললে তোমাকে বলতে হবে অ্যারের কোন কোন পজিশনে সংখ্যাটি আছে। যেমন মনে করো ইনপুট অ্যারেটা হলো:

“ ১০০২৫২১০৮৫১২

খুবই সহজ সলিউশন হলো একটি ডেস্টের নেয়া।। তম পজিশনে x সংখ্যাটি পেলে ডেস্টেরের x তম পজিশনে পুশ করে রাখবে।। তাহলে ইনপুট
নেবার পর ডেস্টেরগুলোর চেহারা হবে:

“ [0]->1 2 7

[1]->0 6 10

[2]->3 5 11

[3]->empty

[4]->8

[5]->4 9

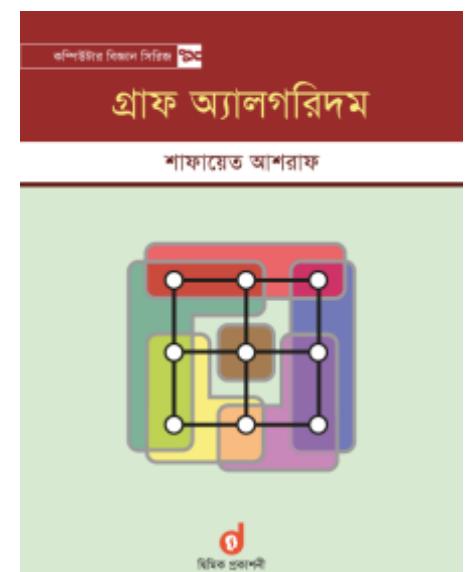
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



কোন সংখ্যা কোন কোন পজিশনে আছে তুমি পেয়ে গেলে। এখন যদি বলে 2 কোথায় কোথায় আছে তাহলে তুমি ২ নম্বর ইনডেক্সে লুপ চালিয়ে 3,5,11 প্রিন্ট করে দাও। 1001 সাইজের ২-ডি ডেস্টের দিয়েই কাজ হয়ে যাবে।

এখন তোমাকে বলা হলো সংখ্যাগুলো নেগেটিভও হতে পারে এবং সর্বোচ্চ 2^{30} পর্যন্ত হতে পারে। এবার কি এই পদ্ধতি কাজ করবে? একটি সংখ্যা -100 বা 2^{30} হলে তুমি সেটার পজিশনগুলা কোন ইনডেক্সে রাখবে? অবশ্যই তুমি এতো বড় ডেস্টের ডিক্রিয়ার করতে পারবেনা অথবা নেগেটিভ ইনডেক্স ব্যবহার করতে পারবেনা। ধরো ইনপুট এবার এরকম:

“
input[]=
{-102,1,134565589,134565589,-102,66666668,134565589,66666668,-102,1,-2}

একটা ব্যাপার লক্ষ্য করো, সংখ্যা দেয়া হবে সর্বোচ্চ 10^5 বা ১ লাখটা। তাহলে অ্যারেতে মোট ডিন্ন সংখ্যা হতে পারে কয়টা? অবশ্যই সর্বোচ্চ ১ লাখটা। আমরা প্রতিটি ডিন্ন সংখ্যাকে ছোটো কিছু সংখ্যার সাথে one-to-one ম্যাপিং করে ফেলবো। উপরের অ্যারেতে ডিন্ন ডিন্ন সংখ্যা গুলো হলো:

“
-102,1,134565589,66666668,-2



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ন্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

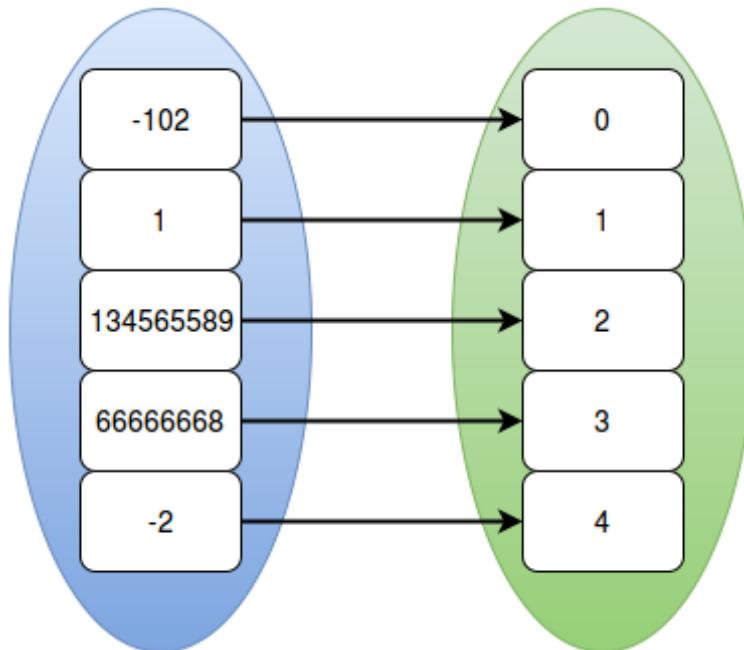
ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



দুইটা উপায় আছে ম্যাপিং করার। একটা হলো STL এর map ব্যবহার করে। map এর কাজ হলো যেকোনো একটি সংখ্যা, স্ট্রিং বা অবজেক্টকে অন্য একটি মান অ্যাসাইন করা যেটা আমরা উপরের ছবিতে করেছি। আমাদের এই প্রবলেমে কোন সংখ্যা কার থেকে বড় বা ছোটো সেট দরকার নাই তাই সেট না করেই ম্যাপিং করে দিতে পারি। নিচের কোডটা দেখো:

```

1 void compress() {
2     map < int, int > mymap;
3     int input[] = {
4         -102,
5         1,
6         134565589,
7         134565589,
8         -102,
9         66666668,
10        134565589,
11        66666668,
12        -102,
    }
  
```

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

13     1,
14     -2
15 };
16 int assign = 0, compressed[100], c = 0, n = sizeof(input) / sizeof(int); //array size;
17 for (int i = 0; i < n; i++) {
18     int x = input[i];
19     if (mymap.find(x) == mymap.end()) { //x not yet compressed
20         mymap[x] = assign;
21         printf("Mapping %d with %d\n", x, assign);
22         assign++;
23     }
24     x = mymap[x];
25     compressed[c++] = x;
26 }
27 printf("Compressed array: ");
28 for (int i = 0; i < n; i++) printf("%d ", compressed[i]);
29 puts("");
30 }

```

কোডের আউটপুট হবে:

“ Mapping -102 with 0
 Mapping 1 with 1
 Mapping 134565589 with 2
 Mapping 66666668 with 3
 Mapping -2 with 4
 Compressed array: 0 1 2 2 0 3 2 3 0 1 4

আমরা ইনপুট অ্যারেতে যখনই একটু নতুন সংখ্যা পাচ্ছি সেটাকে একটা ড্যালু অ্যাসাইন করে দিচ্ছি এবং আসল ড্যালুকে ম্যাপের ড্যালু দিয়ে
 রিপ্লেস করে আরেকটি অ্যারেতে রেখে দিয়েছি। এরপরে আমরা নতুন অ্যারেটা ব্যবহার করে প্রবলেমটা সলভ করতে পারবো। কুয়েরিতে যদি বলে
 -2 কোথায় কোথায় আছে বের করো তাহলে তুমি আসলে 4 কোথায় কোথায় আছে বের করবে কারণ mymap[-2]=4।

আর্টিকুলেশন পয়েন্ট এবং বিজ

স্ট্রিংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্সিমাম ফ্লো-১

ম্যাক্সিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন

সার্সিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১

ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্সেট সাম,
 কষ্টিনেটোরিক্স, ডিসিশন প্রবলেম)

অনেক সময় কম্প্রেস করার পরেও কোন সংখ্যাটি কার থেকে বড় এটা দরকার হয়। যেমন তোমাকে বলতে পারে যে একটি ভ্যালু নেয়ার পর ছেটো কোনো ভ্যালু আর নিতে পারবেনা। এই তথ্যটা কিভাবে রাখবে উপরের পদ্ধতিতে ১ এর থেকে -২ এ অ্যাসাইন করা ভ্যালু বড় হয়ে গিয়েছে। যদি তুমি সেটা না চাও তাহলে আগে ডিম্ব ভ্যালুগুলো আরেকটা অ্যারেতে সর্ট করে নাও।

“ sorted[] ={-102,-2,1,66666668,134565589}

এইবার sorted অ্যারেটা ব্যবহার করে ম্যাপিং করে ফেলো। এ ক্ষেত্রে আসলে যে সংখ্যাটি sorted অ্যারেতে যে পজিশনে আছে সেটাই হবে তার ম্যাপ করা ভ্যালু।

এটা যদি বুঝতে পারো তাহলে নিচয়ই ম্যাপ ছাড়াই বাইনারী সার্চ করে তুমি কম্প্রেস করে ফেলতে পারবে অ্যারেটাকে। ইনপুট অ্যারের উপর লুপ চালাও, প্রতিটি x এর জন্য দেখো sorted অ্যারেতে x এর অবস্থান কোথায়। সেই মানটা তুমি আরেকটা অ্যারেতে রেখে দাও:

“ input[]=
{-102,1,134565589,134565589,-102,66666668,134565589,66666668,-102,1,-2}
compressed_input[]={0,2,4,4,0,3,4,3,0,2,1}

কুয়েরির সময়ও বাইনারী সার্চ করে কম্প্রেস করার মানটা বের করে কাজ করতে হবে।

তুমি ম্যাপ বা বাইনারী সার্চ যেভাবে ইচ্ছা কম্প্রেস করতে পারো। প্রতিবার map এক্সেস করতে logn কম্প্রেক্ষিটি লাগে যেটা বাইনারী সার্চের সমান। stl এ বিভিন্ন class, ডাইনামিক মেমরির ব্যবহারের জন্য map কিছুটা শ্লে, তবে টাইম লিমিট খুব tight না হলে খুব একটা সমস্য হবেনা। map ব্যবহার করলে কোডিং সহজ হয়।

কোনো কোনো গ্রাফ প্রবলেমে নোডগুলা আর এজগুলো কে string হিসাবে ইনপুট দেয়। যেমন ধরো গ্রাফের ঢটা এজ হলো:

“ 3
BAN AUS

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেক্ষেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ভ্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

AUS SRI

SRI BAN

map এ string কেও integer দিয়ে ম্যাপিং করা যায়। এই সুবিধাটা ব্যবহার করে প্রতিটি নোডকে একটি ভ্যালু অ্যাসাইন করবো:

```
1 map < string, int > mymap;
2 int edge, assign = 0;
3 cin >> edge;
4 for (int i = 0; i < edge; i++) {
5     char s1[100], s2[100];
6     cin >> s1 >> s2;
7     if (mymap.find(s1) == mymap.end()) {
8         printf("Mapping %s with %d\n", s1, assign);
9         mymap[s1] = assign++;
10    }
11    if (mymap.find(s2) == mymap.end()) {
12        printf("Mapping %s with %d\n", s2, assign);
13        mymap[s2] = assign++;
14    }
15    int u = mymap[s1];
16    int v = mymap[s2];
17    cout << "Edge: " << u << " " << v << endl;
18 }
```

অনেক সময় বলা হতে পারে নোডগুলোর মান হবে -2^{30} থেকে 2^{30} পর্যন্ত কিন্তু সর্বোচ্চ নোড হবে ১০০০০ টা, তখন আমরা নোডগুলোকে একই ভাবে ছোটো ভ্যালু দিয়ে ম্যাপিং করে ফেলবো।

2-d grid ও কমপ্রেস করে ফেলা যায় অনেকটা এভাবে। সেটা নিয়ে আরেকদিন আলোচনা করতে চেষ্টা করবো, তবে নিজে একটু চিন্তা করলেই বের করতে পারবে।

প্রবলেম:

Drunk

Babel

A Node Too Far

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Shafaetsplanet
2.4K likes
শাফায়েত আশরাফ
[Like Page](#)

Be the first of your friends to like this

Shafaetsplanet about an hour ago
ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটারিঙ্গের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

$i, W) = 1 \text{ if } W =$
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / ম্যার

লোয়েস্ট কমন অ্যানসেস্টর

মার্চ ১৩, ২০১৮ by শাফায়েত



লোয়েস্ট কমন অ্যানসেস্টর জিনিসটা শুনতে একটু কঠিন মনে হলেও জিনিসটা সহজ আৰ খুবই কাজেৰ। বেশ কিছু ধৰণেৰ প্ৰবলেম সলভ কৰে ফেলা যায় এই অ্যালগোরিদম দিয়ে। আমৰা প্ৰথমে লোয়েস্ট কমন অ্যানসেস্টৰ বেৱ কৰাৰ ক্রটফোৰ্স অ্যালগোরিদম দেখবো, তাৰপৰ স্পাৰ্স টেবিল নামেৰ নতুন একটা ডাটা স্টোকচাৰ পিখে কমপ্লেক্সিটি লগ এ নামিয়ে আনবো।

প্ৰথমেই আমৰা জেনে নেই লোয়েস্ট কমন অ্যানসেস্টৰ বা এল.সি.এ(LCA) কি সেট। নিচেৰ ছবিটা দেখ:

সাৰক্ষাইৰ



Secured by [OneAll Social Login](#)

আমাৰ সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়াৰ @
Traveloka Singapore (বিস্তাৰিত...)



শাফায়েত আশুরাফ





প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

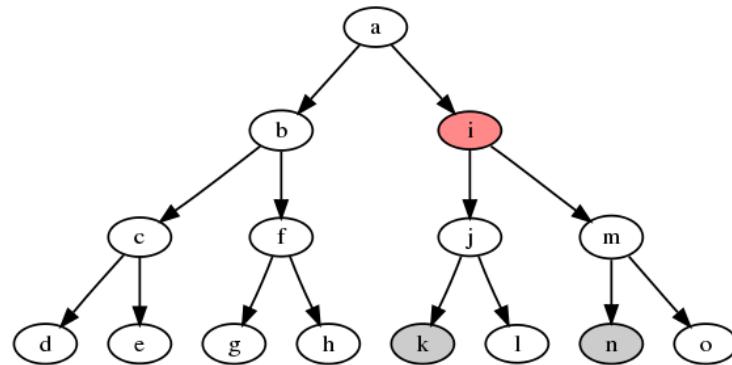
কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম



ছবিতে k আৰু n নোডেৰ প্যারেন্ট ধৰে পিছে যেতে থাকলে তাৰা i নোডে এসে মিলবো। i হলো k,n এৱেলো লোয়েস্ট কমন অ্যানসেস্টুৱ। 'a' ও দুইজনেৰ কমন অ্যানসেস্টুৱ কিন্তু i হলো 'লোয়েস্ট' বা সবথেকে কাছাকাছি।

মনে কৰো ট্ৰি টা দিয়ে বুঝানো হচ্ছে একটা অফিসে কে কাৰ সুপাৰডাইজ বা বস। a হলো অফিসেৰ হেড, b এবং i এৱেলো বস হলো a, আবাৰ c এবং f এৱেলো বস হলো b ইত্যাদি। তাহলে লোয়েস্ট কমন অ্যানসেস্টুৱ অ্যালগোরিদম দিয়ে দুইজন এমপ্লায়িৰ লোয়েস্ট/সব থেকে কাছেৰ কমন বস খুজে বেৱে কৰে ফেলতে পাৰি সহজেই।

প্ৰথমে আমৰা একদম নেইভ একটা উপায় দেখি যেটা খুব বড় ইনপুটেৰ জন্য কাজ কৰবেনা। মনে কৰি আমৰা k,n এৱেল.সি.এ বেৱে কৰতে চাই, প্ৰথমেই k এৱেলো প্যারেন্ট ধৰে উপৰে উঠে সবগুলো অ্যানসেস্টুৱৰ লিস্ট কৰে ফেলি:

k এৱেল অ্যানসেস্টুৱ:

j	i	a
---	---	---

ডাটা স্টোকচাৰ(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলাৰ কিউ

n এর অ্যানসেস্টর:

m	i	a
---	---	---

দুইটা লিস্টেই বাম থেকে ডানে যেতে থাকলে প্রথম যে দুইটা নোড কমন পাবো সেটাই হবে এল.সি.এ! এই উদাহরণে প্রথমে কমন পাবো i, তাই সেটাই হবে এল.সি.এ। 'a' ও একটা কমন এনসেস্টর কিন্তু সেটা "লোয়েস্ট" না। একটা ডিজিটেড অ্যারে বা ফ্ল্যাগ ব্যবহার করে খুব সহজেই আমরা এই অ্যালগরিদম ইমপ্লিমেন্ট করতে পারি। প্রথমে k এর লিস্টে যারা আছে সেগুলোর ইনডেক্সের ফ্ল্যাগ অন করে দিবো, তারপর n এর লিস্টে লুপ চালিয়ে দেখবো ফ্ল্যাগ অন আছে কোন নোডের।

প্রতিটা লিস্ট বানাতে আমাদের সর্বোচ্চ n টা নোড ডিজিট করা লাগতে পারে, তাই কমপ্লেক্সিটি O(n)।

O(n) কমপ্লেক্সিটি খারাপ না যদি আমাদের মাত্র এক জোড়া নোডের এল.সি.এ বের করতে হয়। যদি আমাদের m জোড়া নোড দিয়ে বলে প্রতিটা পেয়ারের এল.সি.এ বের করতে তাহলে কমপ্লেক্সিটি হয়ে যাবে O(m*n)। এটাকে আমরা চাইলে O(m*log(n)) এ সলভ করতে পারি, অর্থাৎ প্রতি কুয়েরি কাজ করবে O(logn) এ। তবে তার আগে কিছু প্রি-প্রসেসিং করে নিতে হবে।

প্রতিটা নোডের জন্য আমরা সেভ করে রাখবো নোডটার:

“ প্রথম প্যারেন্ট বা 2^0 তম প্যারেন্ট

২য় প্যারেন্ট বা 2^1 তম প্যারেন্ট

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই/প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেস্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যাব-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

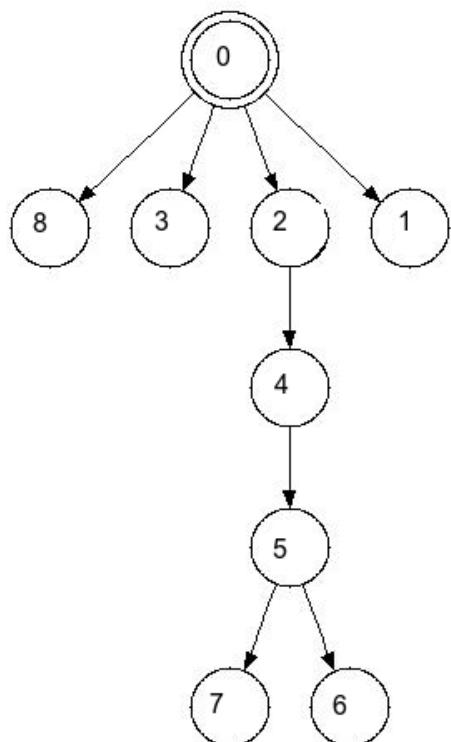
বেলম্যান ফোর্ড

৪৭ প্যারেন্ট বা 2^k তম প্যারেন্ট

“

“ ৮ম প্যারেন্ট বা 2^{k-1} তম প্যারেন্ট

যদি K -তম প্যারেন্ট বলতে কিছু না থাকে তাহলে আমরা -1 রেখে দিবো। যেমন রুট নোডের প্রথম প্যারেন্ট বলতে কিছু নেই, তাই প্রথম প্যারেন্ট হবে -1।



আর্টিকুলেশন পয়েন্ট এবং বিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

তাহলে উপরের ট্রি এর জন্য নিচের মতো একটা ২-ডিমেনশনাল টেবিল তৈরি হবে:

নোড	১৮০ তম প্যারেন্ট	১৮১ তম প্যারেন্ট	১৮২ তম প্যারেন্ট
০	-১	-১	-১
১	০	-১	-১
২	০	-১	-১
৩	০	-১	-১
৪	২	০	-১
৫	৮	২	-১
৬	৫	৮	০
৭	৫	৮	০

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

তাহলে উপরের ট্রি এর জন্য নিচের মতো একটা ২-ডিমেনশনাল টেবিল তৈরি হবে:

এই টেবিলটাকে বলে **স্পার্স টেবিল**। এই টেবিলটা তৈরি করে কি লাভ হলো? এখন তুমি কোনো নোডের k তম প্যারেন্ট খুজেই খুজে বের করতে পারবে। যেমন কারো 2^5 তম প্যারেন্ট দরকার হলে প্রথমে 2^5 এর নিচে 2 এর সবথেকে বড় পাওয়ার $2^{4+1} = 16$ তম প্যারেন্ট খুজে বের করবে। তারপর 16 তম প্যারেন্টের $(2^5 - 16) = 9$ তম প্যারেন্ট খুজে বের করবে একই পদ্ধতিতে! প্রতিটা সংখ্যাকে 2 এর পাওয়ারের যোগফল হিসাবে প্রকাশ করা যায় সেটারই সুবিধা নিছি আমরা। এভাবে করে $O(\log n)$ কমপ্লেক্সিটিতে আমরা কারো k তম প্যারেন্ট বের করতে পারবো যেটা পরবর্তীতে LCA বের করতে কাজে লাগবে।

এখন কথা হলো উপরের টেবিলটা বানাবো কিভাবে? 2^{10} তম প্যারেন্ট বের করা সহজ, ট্রি এর উপর একটা ডেপথ-ফার্স্ট-সার্চ বা ব্রেথড-ফার্স্ট-সার্চ চালিয়ে একটা অ্যারেতে সেভ করে রাখবো কার প্যারেন্ট কে। মনে করি অ্যারেটা হলো T । তাহলে উপরের ট্রি এর জন্য পাবো $T[0] = -1$, $T[1] = 0$, $T[6] = 5$ ইত্যাদি।

এবার আমরা কলাম-বাই-কলাম টেবিলটা ভরাট করবো। প্রথমেই প্রথম কলাম ভরাট করবো যেটা আসলে T অ্যারের সমান।

টেবিলটা নাম P হলে প্রথম কলাম ভরাট করবো এভাবে,

```
1 for (i = 0; i < N; i++)
2     P[i][0] = T[i];
```

এখন চিন্তা করে দেখো একটা নোডের $2^{j-1} = 16$ তম প্যারেন্ট হলো নোডটার $2^{j-1} = 8$ ম প্যারেন্টের $2^{j-1} = 8$ ম প্যারেন্ট। তাহলে কোন নোডের 2^j তম প্যারেন্ট হবে নোডটার $2^{(j-1)}$ তম প্যারেন্টের $2^{(j-1)}$ তম প্যারেন্ট! তাহলে যেকোন নোড i এর জন্য:

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet
2.4K likes
শাফায়েত আশরাফ

Like Page

Be the first of your friends to like this

 **Shafaetsplanet** about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটারের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হয়।

*i, W) = 1 if W =
i + 1, W - C[i]) -*

SHAFAEETSPPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / ম্যাপ

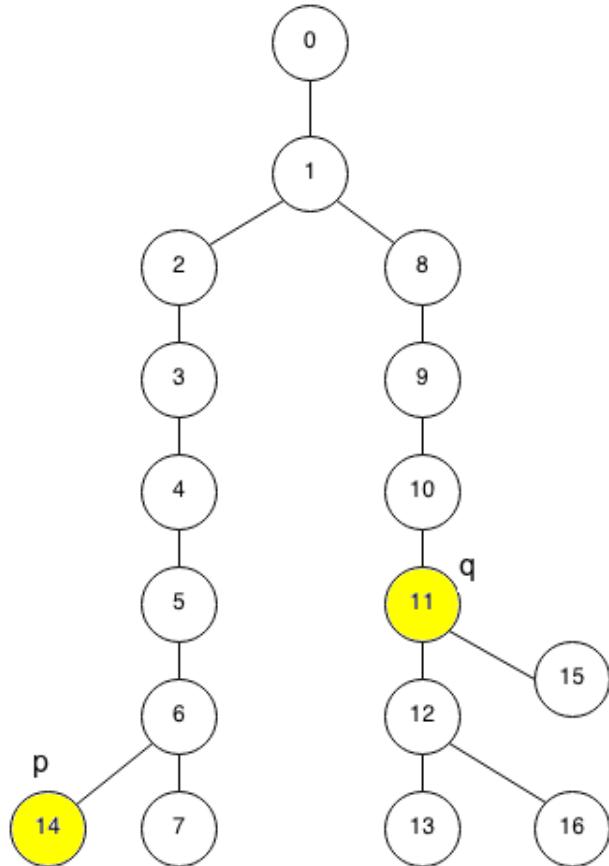
$P[i][j] = P[P[i][j - 1]][j - 1]$ যেখানে $P[i][j - 1]$ হলো $j - 1$ তম প্যারেন্ট।

তাহলে পুরো টেবিলটা ভোট করে ফেলতে পারি এভাবে:

```
1 for (j = 1; (1 << j) < N; j++)
2     for (i = 0; i < N; i++)
3         if (P[i][j - 1] != -1)
4             P[i][j] = P[P[i][j - 1]][j - 1];
```

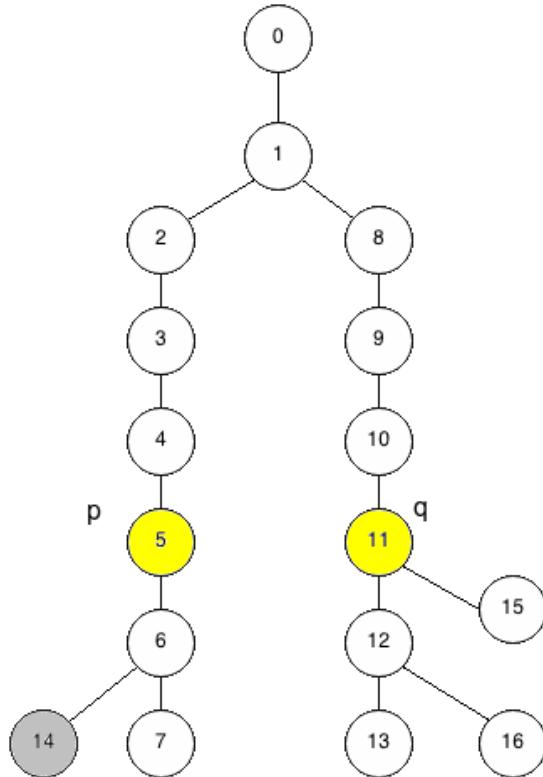
2^j এর মান যদি N এর ছোট না হয় তাহলে নতুন কোনো প্যারেন্ট পাবার সম্ভাবনা নেই, তাই লুপ চলবে ($1 << j$) বা 2^j যতক্ষণ N এর ছোট হবে।

এখন আমরা দুটি নোডের এল.সি.এ বের করবো। আমাদের আরেকটা অতিরিক্ত অ্যারে লাগবে $L[]$ যেখানে থাকবে প্রতিটা নোডের ডেপথ বা লেডেল। রুট নোডের লেডেল হবে ০।



এখন আমরা হলুদ নোড দুইটার এল.সি.এ বের করতে চাই:

প্রথম কাজ হবে এদেরকে একই লেভেলে নিয়ে আসা। p এর লেভেল থেকে ২ এর পাওয়ার বিয়োগ করতে থাকবো এবং একই সাথে স্পার্স টেবিল
ব্যবহার করে উপরে উঠাতে থাকবো যতক্ষণনা q এর লেভেলের সমান হয়।



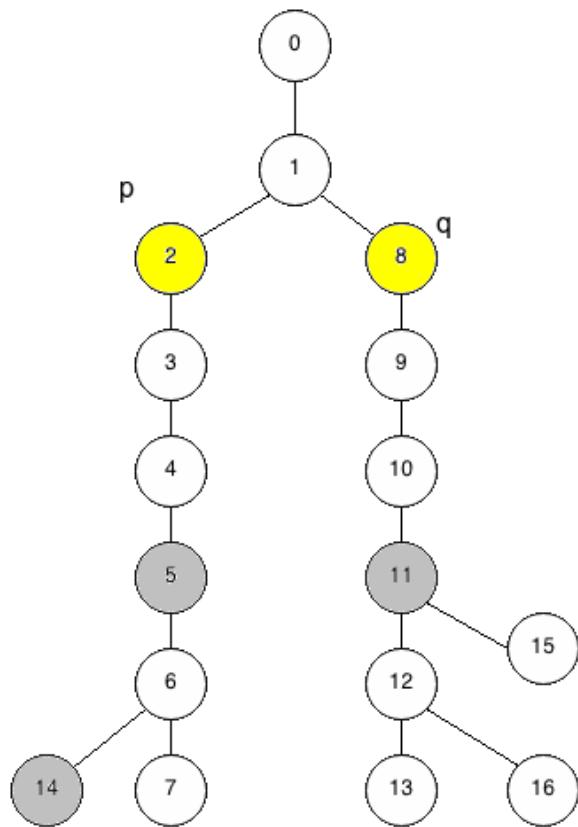
```

1 for (i = log; i >= 0; i--)
2     if (L[p] - (1 << i) >= L[q]) //2^i তম প্যারেন্ট যাও
3     p = P[p][i];

```

এখন দুইজন সমান লেভেলে আসলো। এবাব কাজ হবে দুইটা নোডকেও টেনে একসাথে উপরে উঠানো যতক্ষন নোড দুটির প্যারেন্ট একসাথে

এসে মিলে:



কোডটা হবে এরকম:

```

1  for (i = log; i >= 0; i--)
2      if (P[p][i] != -1 && P[p][i] != P[q][i])
3          p = P[p][i], q = P[q][i];

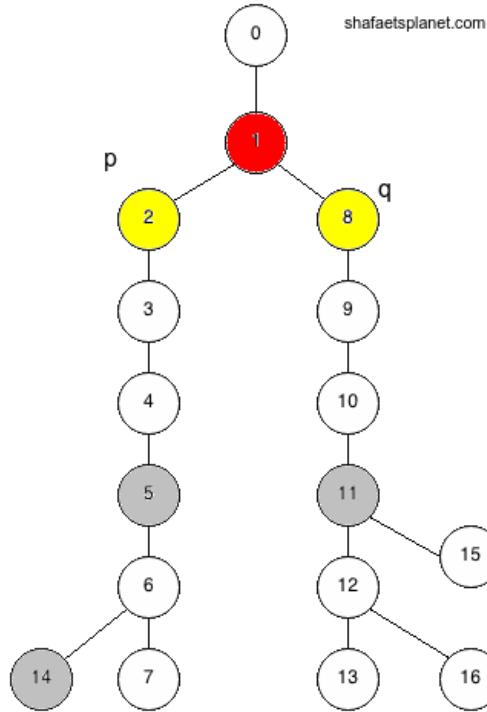
```

আমরা এমন জায়গায় p, q কে আনছি যেখানে তাদের দুইজনেরই প্যারেন্ট একই। তাহলে এখন p বা q এর প্যারেন্টই হবে এল.সি.এ।

```

1  return T[p];

```



আমরা যদি লুপের মধ্যেই ২ টার ৮ নাস্তার নোডকে p, q তে না এনে ডিবেষ্ট ১ নস্তর নোডে আনার চেষ্টা করতাম তাহলে কি ঘটতে পারতো? $P[p][i] \neq P[q][i]$ এই শর্টটা না থাকলে কি হত? এটা চিন্তা করা তোমার কাজ 😊।

কমপ্লেক্সিটি:

স্পার্স টেবিলে রো থাকবে n টা, প্রতিটা রো তে $\log n$ টা কলাম থাকবে, প্রি-প্রোসেসিং কমপ্লেক্সিটি $O(n \log n)$ ।

প্রতি কুয়েরিতে কমপ্লেক্সিটি $O(\log n)$

রিলেটেড প্রবলেম:

১. একটা ওয়েটেড ট্ৰি দেয়া আছে। দুটি নোড দিয়ে বলা হলো তাদের মধ্যের দূৰত্ব বেৰ কৰতে হবে। ([Spoj QTree](#))
২. একটা ওয়েটেড ট্ৰি দেয়া আছে। দুটি নোড দিয়ে বলা হলো তাদের মধ্যকাৰ পাথেৰ সৰ্বোচ্চ ওয়েটেৰ এজ এৰ মান বলতে হবে। ([LOJ A Secret Mission](#))

সলিউশন ১: শুৰুতে ডিএফস চালিয়ে রুট থেকে প্রতিটা নোডেৰ দূৰত্ব সেভ কৰে রাখো। এখন নোড দুইটা a, b এবং তাদেৰ কমন অ্যানসেস্টুৱ L হলে সলিউশন হবে $\text{dist}(\text{root}, a) + \text{dist}(\text{root}, b) - 2 * \text{dist}(\text{root}, L)$ ।

সলিউশন ২: স্পার্স টেবিলে অতিৰিক্ত একটা ইনফোর্মেশন বাখতে হবে। প্রতিটা নোড থেকে $2^0, 2^1, 2^2$ ইত্যাদি তম প্যারেণ্টেৰ জন্য, প্যারেণ্ট থেকে ওই নোডেৰ মধ্যকাৰ পাথেৰ সৰ্বোচ্চ ওয়েটেৰ এজেৰ মান সেভ কৰে রাখতে হবে। বিস্তারিত লিখলাম, এটা সলভ কৰা তোমার কাজ।

আৰো কিছু প্রবলেম:

[TJU Closest Common Ancestors](#)

[UVA Flea Circus](#)

[LightOJ LCA Category](#)

রিসোৰ্স:

এল.সি.এ বেৰ কৰাৰ আৰো কিছু পদ্ধতি আছে যেগুলো পাৰে [এখানে](#)। এগুলো ছাড়াও টাৰজানেৰ একটা [অফলাইন অ্যালগোরিদম](#) আছে যেটা আগে সব কুয়েৰি ইনপুট নিয়ে একটি ডিএফস চালিয়ে সবগুলো পেয়াৰেৰ এল.সি.এ বেৰ কৰে দেয়।

সম্পূৰ্ণ কোড:

ডেষ্টের g তে ট্রি সেভ করতে হবে। প্রথমে dfs ফাংশন কল করে লেডেল, প্যারেন্ট ফিল্ড করে তারপর lca_init কল করে পিপ্রসেসিং করতে হবে।

```
1 //LCA using sparse table
2 //Complexity: O(NlgN,lgN)
3 #define mx 100002
4 int L[mx]; //লেডেল
5 int P[mx][22]; //স্পার্স টেবিল
6 int T[mx]; //প্যারেন্ট
7 vector<int>g[mx];
8 void dfs(int from,int u,int dep)
{
9
10    T[u]=from;
11    L[u]=dep;
12    for(int i=0;i<(int)g[u].size();i++)
13    {
14        int v=g[u][i];
15        if(v==from) continue;
16        dfs(u,v,dep+1);
17    }
18 }
19
20 int lca_query(int N, int p, int q) //N=নোড সংখ্যা
21 {
22     int tmp, log, i;
23
24     if (L[p] < L[q])
25         tmp = p, p = q, q = tmp;
```

```

27     log=1;
28
29     while(1) {
30         int next=log+1;
31         if((1<<next)>L[p])break;
32         log++;
33
34     }
35
36     for (i = log; i >= 0; i--)
37         if (L[p] - (1 << i) >= L[q])
38             p = P[p][i];
39
40     if (p == q)
41         return p;
42
43     for (i = log; i >= 0; i--)
44         if (P[p][i] != -1 && P[p][i] != P[q][i])
45             p = P[p][i], q = P[q][i];
46
47     return T[p];
48
49 void lca_init(int N)
50 {
51     memset (P,-1,sizeof(P)); //গুরুতে সবগুলো ঘরে -1 থাকবে
52     int i, j;
53     for (i = 0; i < N; i++)
54         P[i][0] = T[i];
55

```

```
56     for (j = 1; 1 << j < N; j++)
57         for (i = 0; i < N; i++)
58             if (P[i][j - 1] != -1)
59                 P[i][j] = P[P[i][j - 1]][j - 1];
60
61
62 int main(void) {
63     g[0].pb(1);
64     g[0].pb(2);
65     g[2].pb(3);
66     g[2].pb(4);
67     dfs(0, 0, 0);
68     lca_init(5);
69     printf( "%d\n", lca_query(5,3,4) );
70
71 }
```

LCA hosted with ❤ by GitHub

[view raw](#)

হ্যাপি কোডিং!

ফেসবুকে মন্তব্য

3 comments

ডাটা স্ট্রাকচার: বাইনারি ইনডেক্সড ট্রি

আগস্ট ১৯, ২০১৮ by শাফায়েত



বাইনারি ইনডেক্স ট্রি খুবই চমৎকার একটা ডাটা স্ট্রাকচার যার সবথেকে ভালো দিক হলো কয়েক মিনিটেই কোড লিখে ফেলা যায়। আর খারাপ দিক হলো ভিতরে কি হচ্ছে বুঝতে একটু কষ্ট হয়, তবে তুমি লেখাটা মনযোগ দিয়ে পড়লে আমি নিশ্চিত কোন সমস্যা হবে না। বাইনারি ইনডেক্সড ট্রি বা BIT এর আরেক নাম হলো ফেনউইক(fenwick) ট্রি। এই লেখাটা পড়ার আগে তোমাকে বিটওয়াইজ অপারেশনগুলো(AND,OR ইত্যাদি) সম্পর্কে ভালো জানতে হবে।

আমরা একটা সহজ সমস্যা সমাধান করতে করতে বাইনারি ইনডেক্স ট্রি সম্পর্কে জানব। এই লেখায় আমরা ধরে নিব অ্যারের ইনডেক্স ১ থেকে শুরু, BIT কিভাবে কাজ করে জানার পরে বুঝতে পারবে এটা কেন গুরুত্বপূর্ণ। তোমাকে একটা অ্যারে দেয়া হলো যার প্রতিটি পজিশনে ০ আছে। এখন তোমাকে অনেকগুলো অপারেশন দেয়া হবে। প্রতিটা অপারেশন হতে পারে নিচের যেকোন একটা:

“ * i তম ইনডেক্সে x সংখ্যাটা যোগ কর।

* ১ থেকে i তম ইনডেক্সের সাবঅ্যারের মোট যোগফল বল।

একদম ‘নেইভ(Naive)’ উপায়ে করলে আমরা $O(1)$ এ প্রতি পজিশনে x যোগ করব আর লুপ চালিয়ে $O(n)$ এ যোগফল বের করব। BIT দিয়ে যোগফল $O(\log n)$ এ বের করা সম্ভব, তবে সেক্ষেত্রে আপডেট অপারেশনও $O(\log \times n)$ এ করতে হবে।

আমরা জানি প্রতিটা সংখ্যাকেই কিছু 2 এর পাওয়ারের যোগফল হিসাবে লেখা যায়। যেমন ১৩ এর বাইনারি হলো “১১০১”, বাইনারিতে ১ আছে ০তম, ২য় এবং ৩য় অবস্থানে। তাহলে ১৩ কে লেখা যায় $2^0+2^2+2^3$ বা $1+4+8$ । বাইনারি ইনডেক্সড ট্রি তে আমরা এই প্রোপার্টি ব্যবহার

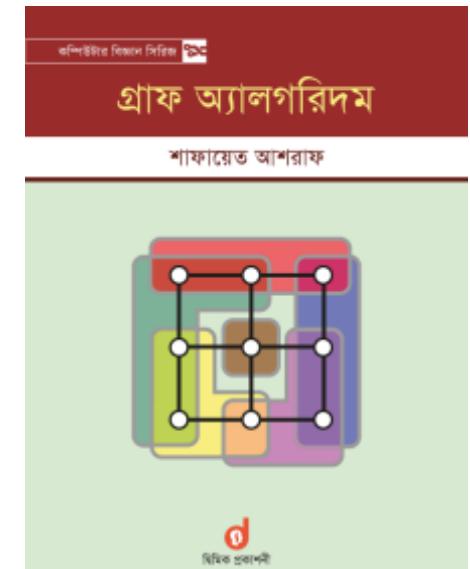
সাবক্ষাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)



করে কিছু কাজ করব। যে সংখ্যাটা অ্যাবেতে যোগ করছি সেটাকে নয় বরং আমরা অ্যাবের ইনডেক্সগুলোকে ২ এর পাওয়ারে যোগফল হিসাবে লিখে কিছু কাজ করব।

১ থেকে i পর্যন্ত অ্যাবের সামকে আমরা অ্যাবের কিছু সেগমেন্টের যোগফল হিসাবে লিখতে পারি। মনে কর আমাদের একটা $sum(i, j)$ ফাংশন আছে যার কাজ হলো i থেকে j ইনডেক্সের মধ্যে অ্যাবের সবগুলো এলিমেন্টের যোগফল বের করা। তাহলে ১৩ তম ইনডেক্সের এর জন্য যোগফল হতে পারে $sum(1, 5) + sum(6, 10) + sum(11, 12)$ । এখানে ১৩কে আমরা ৩টি সেগমেন্টে ভাগ করেছি। বাইনারি ইনডেক্স ট্রি তে আমরা প্রতিটি সংখ্যাকে কিছু সেগমেন্টে ভাগ করব। তবে উপরে যেভাবে করেছি সেভাবে ইচ্ছামত করলে হবে না, এটা আমরা একটু বুদ্ধিমানের মত করব। যেহেতু “বাইনারি ইনডেক্স ট্রি” ব্যবহার করব, তাই বুঝতে পারছ হ্যত যে অ্যাবের ইনডেক্সগুলোর বাইনারি রিপ্রেজেন্টেশনকে ব্যবহার করে কোন কাজ করতে হবে।

কোন সংখ্যাকে যেহেতু ২এর পাওয়ারের যোগফল হিসাবে লেখা যায়, তারমানে সংখ্যাটা থেকে ২এর পাওয়ার বিয়োগ দিতে দিতে আমরা ০ বানিয়ে ফেলতে পারি। ১৩ থেকে একে একে ১, ৪ এবং ৮ বাদ দিলে আমরা পাবো:

$$“ 13 - 1 = 12$$

$$12 - 8 = 4$$

$$4 - 4 = 0$$

তাহলে ১৩ এর জন্য সেগমেন্ট হবে $sum(1, 8) + sum(9, 12) + sum(13, 13)$ । আরেকটা উদাহরণ দেখলে পরিষ্কার হবে, ৬(বাইনারিতে ১১০)=২৮১+২৮২ এর জন্য একই ভাবে লিখতে পারি:

$$“ 6 - 2 = 4$$

$$4 - 4 = 0$$

এবাব তাহলে ৬ এর জন্য সেগমেন্ট হবে $sum(1, 4) + sum(5, 6)$ ।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

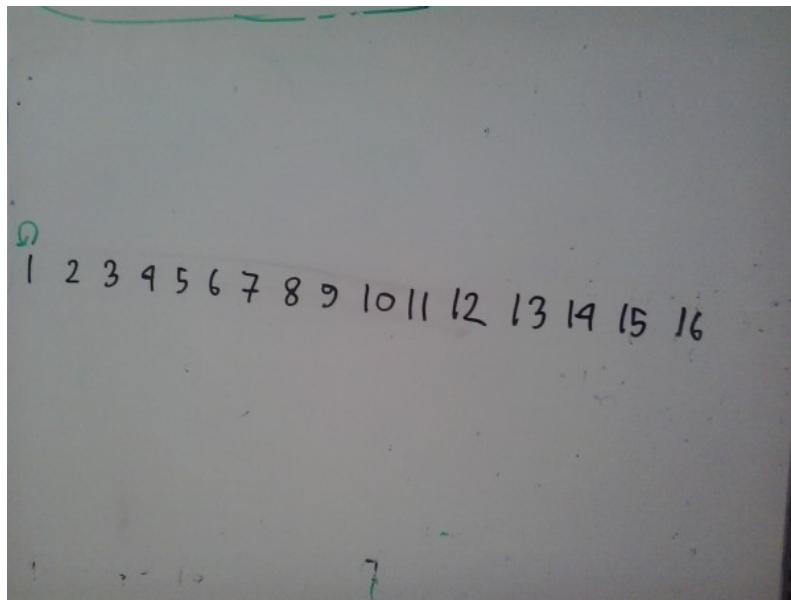
ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

কোড কিভাবে করব সেটা এখন চিন্তা করা দরকার নেই, তুমি শুধু চিন্তা কর গাণিতিকভাবে ভাগটা কিভাবে করা হচ্ছে সেটা তুমি বুঝতে পারছ নাকি। এখন আমরা সেগমেন্টগুলো ছবিতে দেখব:



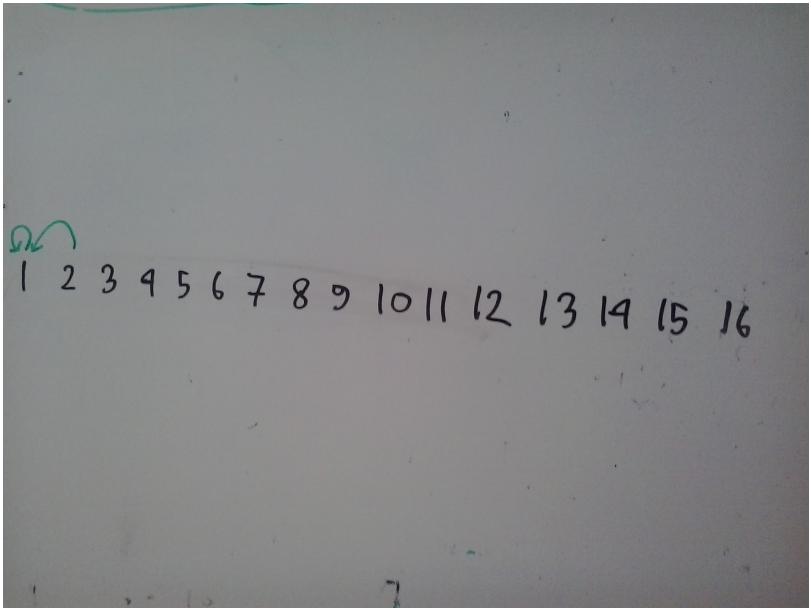
ছবিতে ১ এর জন্য সেগমেন্টটা চিহ্নিত করা হয়েছে। এরপরে আমরা ২এর জন্য করব। ২ এর জন্য উপরের মত করে লিখতে পারি:

$$" 2-2=0$$

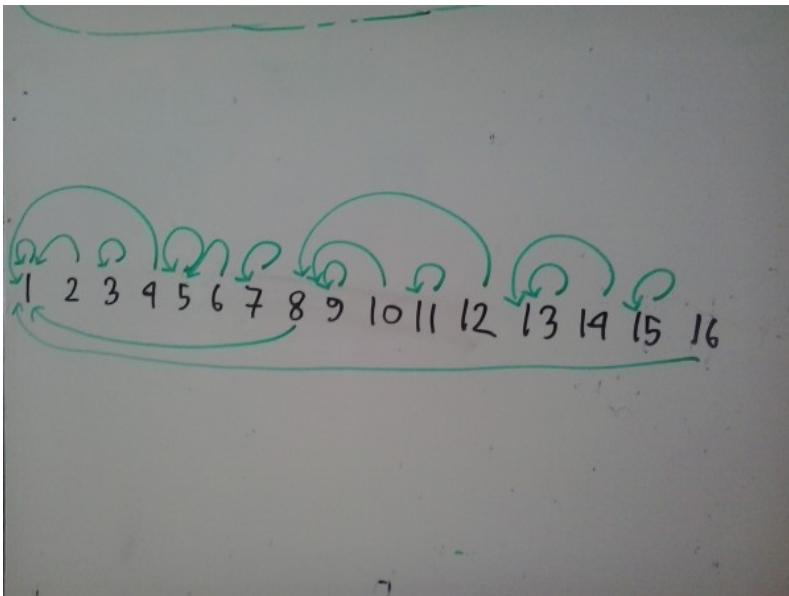
তাই সেগমেন্টটা হবে ১থেকে ২পর্যন্ত।

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)
ডিসজয়েট সেট(ইউনিয়ন ফাইল)
ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)
সেগমেন্ট ট্রি-১
সেগমেন্ট ট্রি-২(লেজি প্রপারেশন)
অ্যারে কমপ্রেশন/ম্যাপিং
লোয়েস্ট কমন অ্যানসেক্টর
বাইনারি ইনডেক্সড ট্রি
স্থার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:
এল-আর-ইউ ক্যাশ(নতুন)
ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):
গ্রাফ থিওরিতে হাতেখড়ি
অ্যাডজেসেন্সি ম্যাট্রিক্স
অ্যাডজেসেন্সি লিস্ট
ব্রেথড ফার্স্ট সার্চ (বিএফএস)
মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)
মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)
টপোলজিকাল সর্ট
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল
সর্ট
ডায়াক্রস্টা
ফ্লয়েড ওয়ার্শল
বেলম্যান ফোর্ড



এখন তোমার কাজ ১৬ পর্যন্ত সবগুলো সংখ্যার জন্য এভাবে সেগমেন্ট একে ফেলা। কিভাবে সেগমেন্ট তৈরি হচ্ছে বোঝাটা বাইনারি ইনডেক্সড ট্রি এর সবথেকে দরকারি অংশ, তাই না বুঝলে আবার ভাল করে পড়। তোমার আকার সাথে নিচের ছবিটি মিলিয়ে দেখ:



আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন
সার্বিসিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১
ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম,
কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

এই ছবি দেখেই আমরা প্রতিটা সংখ্যার জন্য সেগমেন্টগুলো বলে দিতে পারি। যেমন ১১ এর জন্য সেগমেন্টগুলো হবে $(11, 11), (10, 9), (8, 1)$ । তুমি যদি নিজে বুঝে উপরের ছবিটা একে থাকো তাহলে তুমি বুঝে গেছ কিভাবে ছবি দেখে সেগমেন্ট বের করে ফেললাম।

আমরা আগেই $sum(i, j)$ ফাংশন ডিফাইন করেছি। কিন্তু সেই ফাংশনটা ব্যবহার করলে আমাদেরকে i থেকে j তে লুপ চালিয়ে যোগফল বের করতে হবে যেটা আমরা চাইনা। আমরা আরেকটা অ্যারে ডিফাইন করব $Tree[]$]। তুমি ছবিতে দেখতে পাচ্ছ অ্যারের প্রতিটি ইনডেক্স একটা সাবঅ্যারেকে কভার করে। যেমন ১২ নম্বর ইনডেক্স $(8, 12)$ সাবঅ্যারেকে কভার করে। আমরা এমন কিছু করার চেষ্টা করব যেন $Tree[i]$ তে i তম ইনডেক্স যতটুকু সাবঅ্যারে কভার করে সেটার যোগফল থাকে। তাহলে $Tree[12]$ তে সেভ করা থাকবে $sum(8, 12)$, আবার $Tree[14]$ তে সেভ করা থাকবে $sum(13, 14)$ ।

এই কাজটা যদি করতে পারি তাহলে ১১ নম্বর ইনডেক্সের জন্য $sum(11, 11) + sum(10, 9) + sum(8, 1)$ বের না করে আমরা $tree[11] + tree[10] + tree[8]$ বের করলেই যোগফল পেয়ে যাবো।

এখন ধরি আমাকে বলা হল ৫ নম্বর ইনডেক্সে x যোগ করতে চাই। আমাকে এখন বুদ্ধিমানের মত Tree অ্যারেতে কিছু ইনডেক্সে x সংখ্যাটা যোগ করে দিতে হবে। আমরা প্রথমে $Tree[5]$ এ x যোগ করব, এরপরে সবথেকে ছোট সেই ইনডেক্সে করব যেটা উপরের ছবিতে ৫কে কভার করে। সেই ইনডেক্সটা হলো ৬, তাহলে $tree[6]$ এ x যোগ করব। এরপরে ৬কে কভার করে ৮, তাই $tree[8]$ এ x যোগ করব। এরপরে একই ভাবে $tree[16]$ তে x যোগ করব।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেক্ষেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

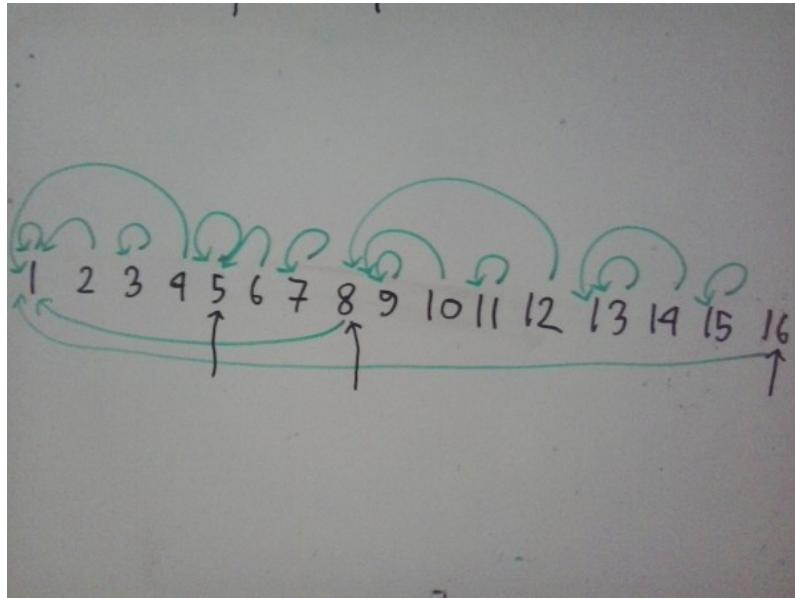
অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

চেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



আমরা tree অ্যারেতে ছবিতে তীব্র চিহ্ন দেয়া ইনডেক্সগুলোতে x যোগ করব। এখানে লক্ষ্য করার বিষয় হলো উপরে লেখা পদ্ধতিতে যোগফল বের করার সময় তুমি এই টোটা ইনডেক্সের সর্বোচ্চ যেকোন এক্সেস করতে পারবে। যেমন ১৬ তে এক্সেস করলে সরাসরি ৫, ৮ ইত্যাদিকে স্কিপ করে সরাসরি ১ এ চলে যেতে হবে। এই প্রোপার্টি থাকার কারণে একই আপডেট দুইবার গুণে ফেলার সম্ভাবনা নেই। আপডেট অপারেশন ঠিকমত বুঝেছো নাকি পরীক্ষা করতে ন এবং জন্য উপরের মত করে তীরচিহ্ন বসায় এবং তারপর নিচের সমাধানের সাথে মিলিয়ে দেখ:

কোয়ান্টাম কম্পিউটার কী?
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

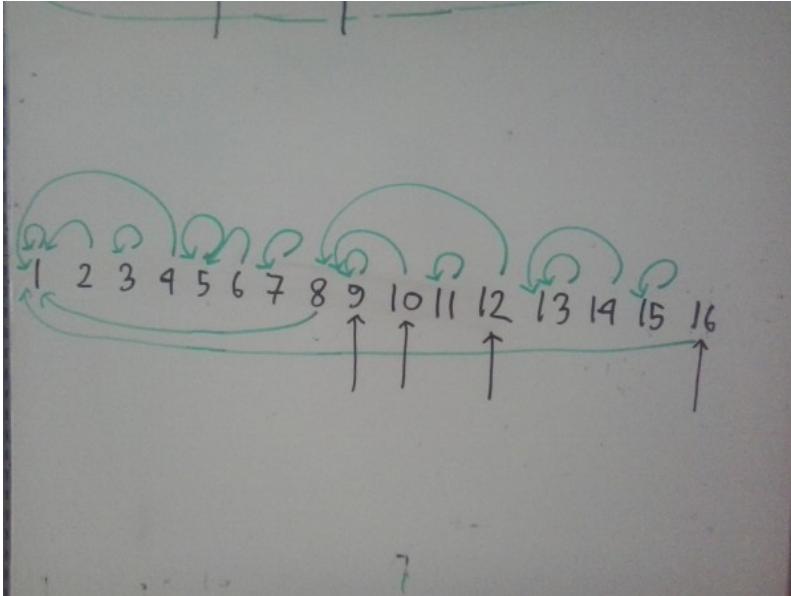


Be the first of your friends to like this

 **Shafaetsplanet** 
about an hour ago
ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটারের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

$i, W) = 1 \text{ if } W =$
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / ম্যান



ছবিতে ৯থেকে শুরু করে প্রতিবার এমন ইনডেক্সে গিয়েছি যেটা আগের ইনডেক্সটাকে পুরোটা কভার করে। যোগফল বের করার সময় কোনভাবেই এই ৮টা ইনডেক্সের একটার বেশি এক্সেস করা সম্ভব না!

এবার আমরা কোডের দিকে যাবো। প্রথমেই কুয়েরি অপারেশনটা ইম্প্লিমেন্ট করব। কোন সংখ্যা ২ এর পাওয়ার হলে তার বাইনারি রিপ্রেজেন্টেশন হ্য ১, ১০, ১০০, ১০০০, ১০০..০০ এরকম, অর্থাৎ ১ এর পর কিছু শূন্য। এখন ১১ নাম্বার ইনডেক্সের জন্য কুয়েরি করতে চাই। ১১ কে বাইনারিতে লিখতে পারি ১০১১। আমরা প্রথমেই সবথেকে ডানের ১ খুজে বের করব এবং সেই পজিশন থেকে বাকি অংশটা মূল সংখ্যা থেকে বিয়োগ দিব। তাহলে শুরুতেই বিয়োগ হবে ১।

“ ১০১১-১=১০১০ (অথবা দ্রষ্টব্য পদ্ধতিতে ১১-১=১০)

আবারো সব থেকে ডানের ১ থেকে শুরু করে বাকিটা বিয়োগ দিলে পাব:

১০১০-১০=১০০০ (অথবা দ্রষ্টব্য পদ্ধতিতে ১০-২=৮)

এভাবে বিয়োগ করতে থাকবো যতক্ষণ শূন্য না পাই।

“

“ $1000 - 1000 = 0$ (অথবা দশমিক পদ্ধতিতে $8 - 8 = 0$)

লক্ষ্য কর ১১ এর জন্য দরকারি টটা ইনডেক্স ১১,১০,৮ আমরা পেয়ে গিয়েছি ২এর পাওয়ার বিয়োগ করে করে! ১৩ বা বাইনারিতে ১১০১ এর জন্য
নিজে কর এবং তারপর নিচের সমাধান দেখ:

“ $1101 - 1 = 1100$ (অথবা দশমিক পদ্ধতিতে $13 - 1 = 12$)

$1100 - 100 = 1000$ (অথবা দশমিক পদ্ধতিতে $12 - 8 = 4$)

$1000 - 1000 = 0$ (অথবা দশমিক পদ্ধতিতে $8 - 8 = 0$)

১৩ এর জন্য দরকারি ইনডেক্স ১৩,১২,৮ পেয়ে গিয়েছি!

এখন পশ্চ হল এই কাজটা কোডে করব কিভাবে? লুপ চালিয়ে একটা একটা বিট পরীক্ষা করে করা যেতে পারে কিন্তু কাজ $O(1)$ এই করা যায়। idx
নম্বর ইনডেক্সের পরের ইনডেক্সটা হবে:

“ $idx = idx - (idx \& -idx)$

এটা কিভাবে কাজ করে বোঝার জন্য খাতাকলমে বিয়োগটা করে দেখ আমার ব্যাখ্যা পড়ার আগেই! আমি একটা উদাহরণ দিচ্ছি, ধরি $idx = 10$ বা
বাইনারিতে ১০১০। তাহলে 2's complement পদ্ধতিতে $-idx$ বা -10 হবে $0101 + 1 = 0110$ । (যারা ভুলে গেছ তাদের জন্য, 2's কমপ্লিমেন্ট বের
করতে বিটগুলোকে উল্টে দিয়ে ১ যোগ করতে হয়।) এবার যদি বিটওয়াইজ AND করি তাহলে পাবো:

1010

"
0110
—
—

0010

সবথেকে ডানের ১থেকে বাকি অংশ পেয়ে গেলাম! এবার বিয়োগ করে দিলেই কাজ শেষ। এটা কেন কাজ করছে? আমরা বাইনারি সংখ্যাটাকে তটা অংশে ভাগ করতে পারি $x1z$ । 1 হলো সবথেকে ডানের ১, x হলো তার আগের যেকোন বিট, z হলো ১ এর পরের থাকা ০ বিট(x বা z এ শৃঙ্খলা, একটা বা একাধিক বিট থাকতে পারে। এখন বিটগুলো উল্টেদিলে পাবো: $x'0z'$ । x বা z অংশের বিটগুলোকে উল্টে দিয়ে পেয়েছি x' আর z' , তারমানে z' হলো কিছু ১ বিট। 2's কমপ্লিমেন্ট বের করতে ১ যোগ করলে পাবো: $x'1z$ । এখন আগের সংখ্যা $x1z$ আর $x'1z$ কে AND করলে থাকে $1z$ । x এর ভাগটা পুরোটা শূণ্য হয়ে যাবে তাই সেটা বাদ। তাহলে আমরা ডানের ১থেকে বাকি অংশ পেয়ে যাবো!

কোডটা তাহলে খুব সহজ:

```
1 int query(int idx){  
2     int sum=0;  
3     while(idx>0){  
4         sum+=tree[idx];  
5         idx -= idx & (-idx);  
6     }  
7     return sum;  
8 }
```

আপডেট অপারেশন কিভাবে করব? ১০১০ বা ১০ এর পরের কোন ইনডেক্সটা ১০ কে কভার করবে? এবার আমরা সবথেকে ডানের ১ কে বামপাশে শিফট করে দিব। ১০১০ এর সাথে ১০ যোগ করলে পাই ১১০০ বা ১২ যেটা ১০কে পুরোটা কভার করবে। ঠিক সেরকম ১২ বা ১১০০ এর সাথে ১০০ যোগ করলে পাই ১০০০০ যেটা ১২কে পুরোটা কভার করবে। এটা কিভাবে কাজ করছে সেটা চিন্তা করে বের করে ফেল!

আগের মত করেই শুধু বিয়োগের জায়গায় যোগ করলেই সবথেকে ডানের ১টা বামে সরে যাবে। তারমানে যে সংখ্যাটা যোগ করছি, নতুন ইনডেক্স থেকে কুয়েরির সময় তারথেকেও বড় একটা সংখ্যা বিয়োগ করা হবে তাই নতুন ইনডেক্সটা আগেরটাকে কভার করতে বাধ্য! কোডটা হবে এরকম:

```

1 void update(int idx, int x, int n) //n is the size of the array, x is the number to add
2 {
3     while(idx<=n)
4     {
5         tree[idx] +=x;
6         idx += idx & (-idx);
7     }
8 }
```

এটাই হলো বাইনারি ইনডেক্সেড ট্ৰি। এখন আমরা সহজেই কোন একটা ইনডেক্স আপডেট কৰতে পাৰব এবং 1 থেকে কোন ইনডেক্স পৰ্যন্ত যোগফল বেৰ কৰতে পাৰব।

কমপ্লেক্সিটি:

আপডেট এবং কুয়েরি সব ক্ষেত্ৰেই 2 এৰ পাওয়াৰ যোগ বা বিয়োগ কৰছি, প্ৰথমে ছোট পাওয়াৰ থেকে শুৰু কৰে বড় পাওয়াৰ বিয়োগ কৰছি। তাই টাইম কমপ্লেক্সিটি হবে $O(\log n)$ । শুধুমাৰি Tree অ্যাৰেটাতেই সব অপারেশন হচ্ছে, স্পেস কমপ্লেক্সিটি $O(n)$ ।

লক্ষ কৰ যে এই প্ৰবলেমটা সেগমেন্ট ট্ৰি দিয়ে কৰা গেলেও এখানে মেমৰি লাগছে অনেক কম, কোডও খুব ছোট। তবে **সেগমেন্ট ট্ৰি**তে বিভিন্ন সব বেঞ্জেৰ ভিতৰ আপডেট কৰা যায়, এখানে কৰা যায় 1 নম্বৰ ইনডেক্স সহ বেঞ্জে।

প্ৰবলেম ১: কুয়েরি কৰাৰ সময় i থেকে j ইনডেক্সেৰ মধ্যেৰ সাৰাংশ্যাৰেৰ যোগফল বেৰ কৰতে বললে কি কৰবে?

প্ৰবলেম ২: কুয়েরিতে যদি বলা হয় অ্যাৱেৰ সৰ্বনিম্ন কোন ইনডেক্সে গেলে যোগফল X এৰ বেশি পাৰো তাহলে ইনডেক্সটা কিভাৱে বেৰ কৰবে?

হিটস: $O(\log n * \log n)$

অনলাইন জাজ থেকে কিছু প্ৰবলেম:

[Curious Robin Hood](#)

[Inversion Count](#)

স্কয়ার-রুট ডিকম্পোজিশন

সেপ্টেম্বর ২৮, ২০১৯ by [শাফায়েত](#)



স্কয়ার-রুট ডিকম্পোজিশন [টেকনিক](#) ব্যবহার করে বেশ কিছু প্রবলেমের টাইম কমপ্লেক্সিটি $O(Sqrt(n))$ এ নামিয়ে আনা যায়। একটা উদাহরণ দিয়ে শুরু করি। ধরা যাক তোমাকে একটা ইন্টিজার অ্যারে দেয়া আছে এবং কিছু অপারেশন দেয়া আছে। অপারেশন দুই ধরণের হতে পারে, একটা হলো $[l, r]$ ইনডেক্সের ভিত্তির সবগুলো সংখ্যার যোগ ফল বের করতে হবে, অন্যটা হলো। তম ইনডেক্সের মান আপডেট করতে হবে। অনেকে হয়তো [সেগমেন্ট ট্রি](#) বা [বাইনারি ইনডেক্সেড ট্রি](#) ব্যবহার করে এটা সমাধান করতে পারবে। আজকে আমরা এটা সমাধান করার আরেকটা নতুন পদ্ধতিতে শিখবো।

1	4	2	2	1	1	2	1	3	5	1	2	7
---	---	---	---	---	---	---	---	---	---	---	---	---

একেবাবেই সাধারণ পদ্ধতিতে আমরা কি করবো? প্রতিবার যোগফল বের করতে বললে $[l, r]$ রেঞ্জে একটা লুপ চালিয়ে যোগফল বের করবো এবং আপডেট অপারেশন শুধু i তম ইনডেক্সটা আপডেট করে দিবো। সেক্ষেত্রে যোগফল বের করার কমপ্লেক্সিটি হচ্ছে $O(n)$ ।

এখন আমরা চেষ্টা করবো য্যারেটাকে কিছু সেগমেন্টে ভাগ করে ফেলতে। এরপর যোগফল বের করার সময় প্রতিটা সেগমেন্টের যোগফল বের করলেই হবে। নিচের ছবিটা দেখো:

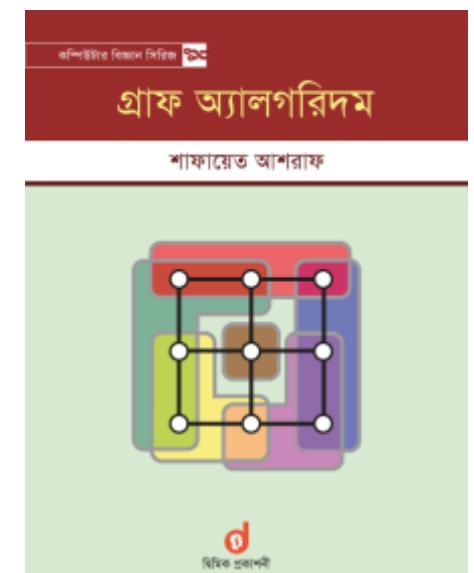
সাবক্ষাইব

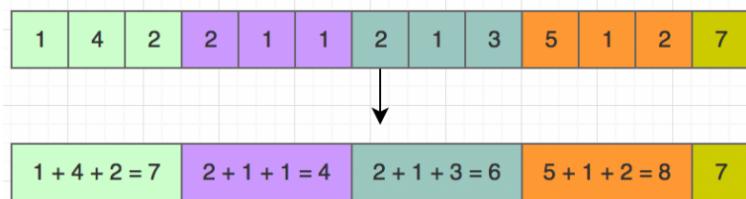


Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @
Traveloka Singapore (বিস্তারিত...)





আমাদের অ্যারের সাইজ ছিলো 13। আমরা এমন ভাবে ভাগ করত চাই যেন প্রতিটা সেগমেন্টের সাইজ প্রায় এক হয় এবং সেগমেন্টের সাইজ এবং সেগমেন্ট সংখ্যা প্রায় একই হয়। 13 এর স্ক্যার রুট হলো 3.613 । আমরা যদি অ্যারেটাকে 3 সাইজের সেগমেন্টে ভাগ করি তাহলে আমরা $13/3 = 4$ টা সেগমেন্ট পাবো, শেষের সেগমেন্ট ছাড়া প্রতিটার আকার 3।

অ্যারের সাইজ যদি নিজে একটা স্ক্যার নাওয়ার না হয় তাহলে আমরা একটা সেগমেন্ট বেশি পাবো এবং শেষের সেগমেন্টের সাইজ একটু কম হবে, সেটা তেমন কোনো সমস্যা না। কেন এভাবে ভাগ করলে সুবিধা সেটা আমরা একটু পরেই বুঝবো।

এবার আমাদেরকে যদি $[l, r]$ রেঞ্জে কুয়েরি করতে বলে তাহলে আমরা শুধু দেখবো কোন কোন সেগমেন্ট আমাদের রেঞ্জের ভিতর পড়ছে এবং সেই রেঞ্জ গুলোর যোগফল বের করবো।

শুরুতে আমাদেরকে সেগমেন্ট গুলো প্রি-প্রসেস করে নিতে হবে। নিচের কোডটি দেখো:

```

1 int segment[10000];
2 int preprocess(int input[], int n) {
3     int current_segment = -1;
4     int segment_size = sqrt(n);
5
6     for (int i=0; i<n; i++) {
7         if (i % segment_size == 0) {
8             current_segment++; //new segment
9         }
10        segment[current_segment] += input[i];
11    }
12
13    return segment_size;
14 }
```

আমরা ইনপুট অ্যারেকে ট্র্যাভার্স করছি এবং যতবার i হচ্ছে ততবার পরের সেগমেন্টে চলে যাচ্ছি।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ন্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগরিদম

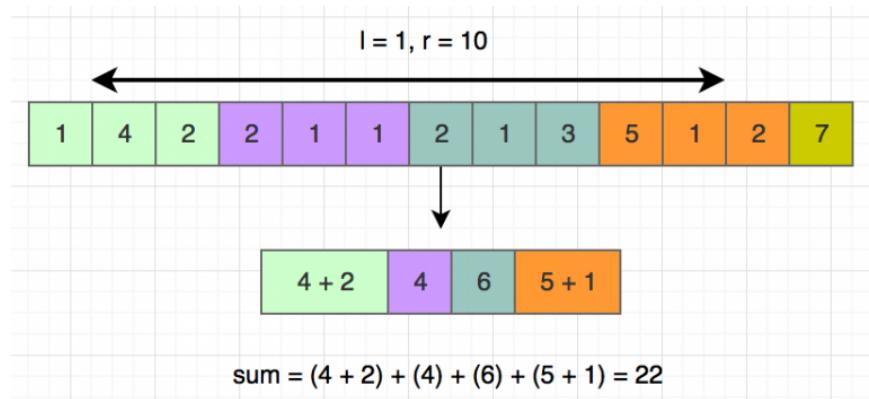
ডাটা স্টোকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

[1, 10] ৰেঞ্জের জন্য কোন কোন সেগমেন্টের যোগফল বের করতে হবে সেটা পৰের ছবিতে দেখানো হয়েছে।



ঘামেলা হলো সবগুলো সেগমেন্ট পুরোপুরি আমাদের ৰেঞ্জের ভিতৰ পড়েনি। প্ৰথম এবং শেষ সেগমেন্ট যদি পুরোপুরি ৰেঞ্জ টাকে কভাৰ না কৰে তাহলে সেই দুটি সেগমেন্টের জন্য আমাদের লুপ চালিয়ে যোগফল বের কৰতে হবে। আমাদের সেগমেন্ট আছে \sqrt{n} টা, আবাৰ প্ৰতিটা সেগমেন্টের আকাৰও \sqrt{n} , সৰ্বোচ্চ দুটি সেগমেন্টে লুপ চালাতে হবে, তাই কুয়েৰিৰ মোট কমপ্লেক্সিটি থাকছে \sqrt{n} । এইজন্য আমৰা সেগমেন্ট সংখ্যা এবং সেগমেন্টের আকাৰ যতটা সম্ভব কাছাকাছি রেখেছি।

এখন ছোট একটি প্ৰশ্ন। কোনো একটা ইনডেক্স | এৰ জন্য কিভাৱে বুৰুবে | কোন সেগমেন্টে অবস্থিত? খুব সহজ, $i / segment_size$ এৰ মান দেখেই বোৰা যাবে।

নিচেৰ কোডে দেখিয়েছি কিভাৱে কুয়েৰি কৰতে হবে:

```
1 int query(int input[], int segment_size, int l, int r) {  
2     int sum = 0;  
3  
4     //loop the first segment  
5     //until we reach r or a starting index
```

স্লাইডিং ৰেঞ্জ মিনিমাম কুয়েৰি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইন্ড)

ট্ৰাই(প্ৰিফিল ট্ৰি/বেডিল ট্ৰি)

সেগমেন্ট ট্ৰি-১

সেগমেন্ট ট্ৰি-২(লেজি প্ৰপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টৰ

বাইনারি ইনডেক্সেড ট্ৰি

স্থায়াৰ-কুট ডিকম্পোজিশন ক্যাশিং অ্যালগোরিদম:

এল-আৱ-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টাৰ(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্ৰিক্স

অ্যাডজেসেন্সি লিস্ট

ব্ৰেথড ফার্স্ট সাৰ্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্ৰি ১ (প্ৰিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্ৰি ২ (ক্ৰুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সাৰ্চ এবং আবাৰো টপোলজিকাল
সেট

ডায়াক্ৰটা

ফ্ৰয়েড ওয়াৰ্শল

বেলম্যান ফোৰ্ড

```

6
7     while (l < r && l % segment_size != 0) {
8         sum += input[l];
9         l++;
10    }
11
12    //Loop until we reach
13    //segment that contains r
14    while (l + segment_size <= r) {
15        sum += segment[l / segment_size];
16        l += segment_size;
17    }
18
19    //loop until r
20    while (l<=r) {
21        sum += input[l];
22        l++;
23    }
24
25    return sum;
26 }

```

কুয়েরি কে আমরা ৩ ভাগে ডাগ করেছি যেটার কথা একটু আগেই আলোচনা করেছি।

সবশেষে আপডেট। আপডেট খুবই সহজ, i তম ইনডেক্স আপডেট করতে হলে i কোন সেগমেন্টে আছে সেটা বের করে সেটার যোগফল আপডেট করে দিলেই হচ্ছে।

```

1 void update(int input[], int segment_size, int i, int val) {
2     int segment_no = i / segment_size;
3
4     segment[segment_no] -= input[i];
5     segment[segment_no] += val;
6     input[i] = val;
7 }

```

আপডেট করার কমপ্লেক্সিটি $O(1)$ ।

এটাতো গেলো খুবই সহজ একটা প্রবলেম। এই টেকনিক ব্যবহার করে ট্রি এর LCA ও রেব করা যায়, সেটা শিখতে হলে [এই লিংকটা](#) দেখতে পারো। আর যারা একটু অ্যাডভান্সড লেভেলে তারা এই [পিডিএফ](#) টা দেখতে পারো।

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্সিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

ক্যাপিং অ্যালগরিদম: এল-আর-ইউ ক্যাপ

মার্চ ২৫, ২০২০ by শাফায়েত



আজকে আমরা জানবো ক্যাপিং কি এবং এল-আর-ইউ ক্যাপ কিভাবে কাজ করে। তুমি যদি ক্যাপিং সম্পর্কে আগে থেকেই জানো তাহলে প্রথম অংশটা দ্রুত পড়ে পরের অংশে চলে যেতে পারো।

ধৰা যাক তুমি একটি অ্যান্ড্রয়েড অ্যাপ তৈরি করছো। অ্যাপের ইউজারদের তুমি ৩টা বকমের লেভেল অ্যাসাইন করেছো, মাটিনাম, গোল্ড আর সিলভার। যখনই কোনো ইউজার অ্যাপ চালু করে তখনই তোমার ডটাবেসে ইউজারের লেভেল চেক করতে হয়। তোমার অ্যাপ এ ইউজার আছে এক লাখ এবং প্রতি ঘন্টায় কয়েক হাজার ইউজার অ্যাপটি ব্যবহার করে। একসময় তুমি দেখলে ডটাবেজের রিকুয়েস্ট খুব বেশি বেড়ে যাচ্ছে এবং পেজ লোড শো হয়ে যাচ্ছে। এই সমস্যার সমাধান হলো কিছু ইউজারের লেভেল মেমরিতে সেভ করে রাখা। যখনই ইউজার অ্যাপ চালু করবে তখন আগে তুমি দেখবে সেই ইউজারের লেভেল মেমরিতে সেভ করা আছে নাকি, যদি থাকে তাহলে তোমার আর ডটাবেসে রিকুয়েস্ট পাঠানো লাগবে না। এই প্রক্রিয়াটাকেই বলা হয় ক্যাপিং।

সাবস্ক্রাইব



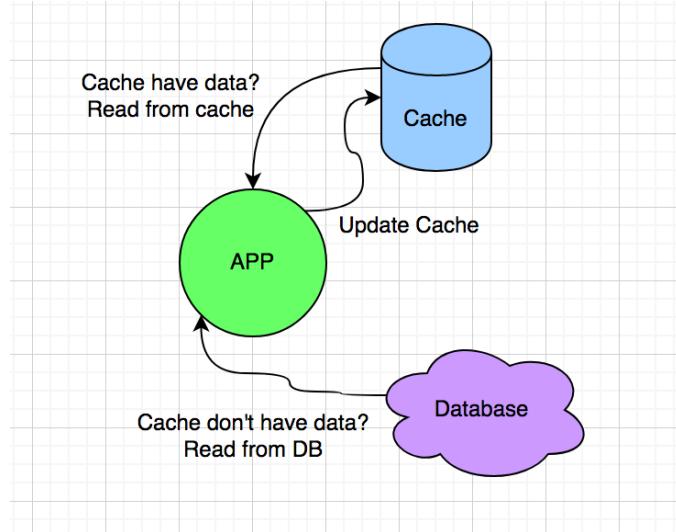
Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

Traveloka Singapore (বিস্তারিত...)





ক্যাষিং যে সবসময় ডাটাবেস থেকে পড়ার জন্য করা হবে সেটা না। কম্পিউটারে হার্ডডিশ থেকে ডাটা নিয়ে ব্যামে রাখা হ্যাদুত এক্সেস করার জন্য, এটাও এক ধরনের ক্যাষিং। আবার প্রতিটা সিপিউ কোর কিছু ডাটা নিজের কাছে ক্যাষ করে রাখে যাতে ব্যাম থেকেও বেশি না পড়তে হ্যাদু।
বুরতেই পারছো ক্যাষিং অনেক লেয়ারে হতে পারে।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

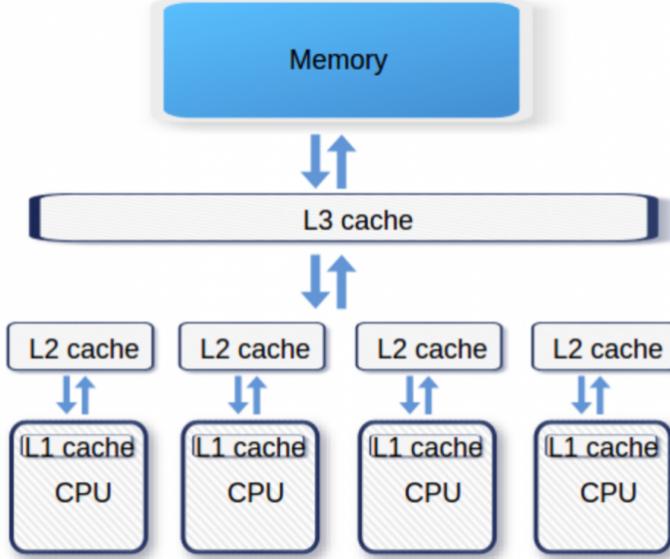
ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



শাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)
ডিসজয়েট সেট(ইউনিয়ন ফাইল)
ট্রাঈ(প্রিফিল ট্রি/বেডিল ট্রি)
সেগমেন্ট ট্রি-১
সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)
অ্যারে কমপ্রেশন/ম্যাপিং
লোয়েস্ট কমন অ্যানসেক্টর
বাইনারি ইনডেক্সেড ট্রি
স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:
এল-আর-ইউ ক্যাশ(**নতুন**)
ব্লুম ফিল্টার(**নতুন**)

এখন ক্যাশিং করার সময় একটা সমস্যার মুখোমুখি সবসময় হতে হয়, সেটা হলো ক্যাশ মেমরির আকার ডাটাবেস, হার্ডডিস্ক থেকে অনেক ছোট হয়। তোমার হার্ডডিস্কে হ্যাতো ৫ টেরাবাইট জায়গা আছে, কিন্তু তোমার ব্যামের সাইজ খুব বেশি হলে ৩২ গিগাবাইট। সমস্যা যেটা হয়, সব ডাটা একসাথে ক্যাশ মেমরিতে লোড করা যায় না। নতুন ডাটা লোড করার সময় যদি ক্যাশ ফুল হয়ে যায় তাহলে পুরোনো কিছু ডাটা ফেলে দিতে হয়। এই প্রসেসটাকে বলা হয় Cache Eviction বা ক্যাশ এভিকশন পলিসি। ক্যাশ এভিকশনের জন্য অনেক ধরনের অ্যালগরিদম আছে, যেমন: FIFO (First in first out), FILO (First in last out), LRU (least recently used) ইত্যাদি।

LRU ক্যাশ

আমরা আজকে দেখবো LRU ক্যাশ কিভাবে কাজ করে এবং কিভাবে ইমপ্লিমেন্ট করতে হয়। LRU ক্যাশ প্রতিবার ডাটা এক্সেস করার সময় মনে রাখে শেষ কখন সেই ডাটা এক্সেস করা হয়েছিলো। নতুন ডাটা ক্যাশে সেভ করার সময় যদি ক্যাশ ফুল হয়ে যায় তাহলে সে যে ডাটা পুরোনো হয়ে গেছে সেটাকে ফেলে দেয়। এক্ষেত্রে যে ডাটা যত আগে এক্সেস করা হয়েছিলো সেটা তত পুরোনো (least recent)।

একটা ছোট সিমুলেশন করে ফেলি। মনে করো তোমার ক্যাশ মেমরি সাইজ মাত্র ৩, অর্থাৎ মাত্র ৩জন ইউজারের ডাটা আমরা ক্যাশে রাখতে পারবো। শুরুতেই মনে করো Alice লগইন করলো এবং তুমি ক্যাশে কুয়েরি করে দেখলে যে Alice সেখানে নেই। তুমি ডাটাবেসে কুয়েরি করে

গ্রাফ থিওরি(২০):
গ্রাফ থিওরিতে হাতেখড়ি
অ্যাডজেসেন্স ম্যাট্রিক্স
অ্যাডজেসেন্স লিস্ট
ব্রেথ ফার্স্ট সার্চ (বিএফএস)
মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)
মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)
টপোলজিকাল সেট
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল
সেট
ডায়াক্রস্ট্রো
ফ্লয়েড ওয়ার্শল
বেলম্যান ফোর্ড

দখলে Alice একজন প্লাটিনাম ইউজার, তুমি সেই ডাটা ক্যাশে রেখে দিলে। নিচের ছবি দেখো:

Key	Value
Alice	Platinum
~~~~~	~~~~~
~~~~~	~~~~~

এখন আরেকজন ইউজার Bob আসলো যে একজন গোল্ড ইউজার, তাকেও ক্যাশে যোগ করি:

Key	Value
Bob	Gold
Alice	Platinum
~~~~~	~~~~~

এরপরে আসলো সিলভার ইউজার John:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পানেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন  
সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১  
ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভেস্ট সাম,  
কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

Key	Value
John	Silver
Bob	Gold
Alice	Platinum

Most Recent

Least Recent

এদিকে Bob করলো কি, অ্যাপে অন্য একটা পেজ ওপেন করলো যেই পেজে আবার ক্যাশে কুয়েরি করলো। এখন Bob এর স্ট্যাটাসে ক্যাশেই আছে, ডাটাবেস কুয়েরি করা দরকার হবে না। এদিকে Bob এর স্ট্যাটাস যেহেতু সবশেষে এক্সেস করা হয়েছে, সে হয়ে যাবে 'most recent', বাকিদের অর্ডার একই থাকবে:

Key	Value
Bob	Gold
John	Silver
Alice	Platinum

Most Recent

Least Recent

এখন যদি চতুর্থ কোনো ইউজার Amy আসে, তাহলে কি হবে? যে Key তে সবার পুরোনো সে বাদ যাবে, এক্ষেত্রে সে হলো Alice।

[পুরোনো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্সিং(১):

ব্যকট্যাক্সিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেন্সিভ ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

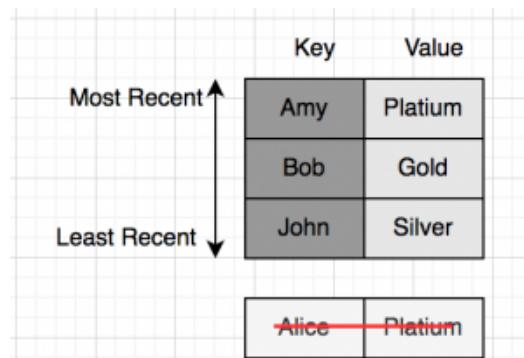
অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



ক্যাশ ইমপ্রিমেন্ট করার সময় আমাদের মূলত ২টা বেসিক ফাংশন ইমপ্রিমেন্ট করতে হয়। একটা হলো get(key) যেটা ইনপুট হিসাবে একটা key নেয় এবং ক্যাশ থেকে সেই key টা খুজে বের করে, আরেকটি হলো put(key, value) যেটা একটা key-value পেয়ারকে ক্যাশ এ সেভ করে। তো আমরা এখন দেখবো কিভাবে LRU ক্যাশের জন্য আমি এই দুটি ফাংশন ইমপ্রিমেন্ট করতে পারি।

**অ্যালগরিদম ১:** আমরা একটা লিস্ট/ডেস্টের ব্যবহার করতে পারি ক্যাশ হিসাবে। লিস্টের প্রতিটা পজিশনে (key, value) এর সাথে সাথে শেষ কথন এক্সেস করা হয়েছিলো সেটা রেখে দিতে হবে। put অপারেশনের জন্য একটা লুপ চালিয়ে সবার পুরানো ইনডেক্সটাকে রিপ্লেস করে দিবো। get অপারেশনের জন্য লুপ চালিয়ে key টাকে খুজে বের করবো। তাহলে দুটি অপারেশনেরই টাইম কমপ্লেক্সিটি  $O(n)$ , আমাদের আরেকটু ভালো কিছু দরকার।

**অ্যালগরিদম ২:** লিস্টের বদলে আমরা একটা হ্যাশম্যাপ ব্যবহার করতে পারি। আগের মতোই put অপারেশনের জন্য key গুলোর উপর লুপ চালিয়ে পুরানোটা রিপ্লেস করে দিবো  $O(n)$  কমপ্লেক্সিটিতে। তবে get অপারেশন এখন খুব দ্রুত কাজ করবে, হ্যাশম্যাপে এভাবেজ কেস এ  $O(1)$  এ key খুজে বের করা যায়। কিছুটা উন্নতি হয়েছে।

**অ্যালগরিদম ৩:** আগের অ্যালগরিদমের বটলনেক ছিলো সবথেকে পুরানো key টা খুজে বের করা। কোন ডাটা স্ট্রাকচার ব্যবহার করলে key গুলো সর্টেড থাকবে? আমরা একটা ব্যালেন্সড বাইনারি সার্চ ট্রি ব্যবহার করতে পারে। সেক্ষেত্রে put এবং get দুটোই  $O(\log n)$  এ কাজ করবে।

**অ্যালগরিদম ৪:** ডাবলি (Doubly) লিংকড-লিস্ট ব্যবহার করে দুটি অপারেশনই  $O(1)$  এ করা সম্ভব। আজকের লেখায় এই অ্যালগরিদমটাই বিস্তারিত বর্ণনা করবো।

## ডাবলি (Doubly) লিংকড-লিস্ট ব্যবহার করে LRU ক্যাশ

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Be the first of your friends to like this

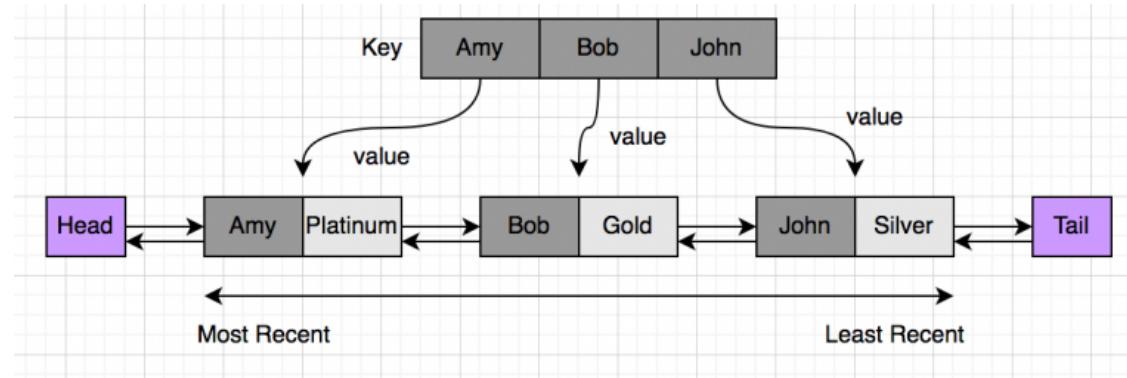


ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিনের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্যাত।

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / স্যাব

উপরে অ্যালগরিদম ২ তে হ্যাশম্যাপ ব্যবহার করে একটা সলিউশনের কথা বলেছিলাম। হ্যাশম্যাপে key-value পেয়ারগুলো রেখে দিলে আমরা খুব দ্রুত key খুজে পাচ্ছি। কিন্তু সমস্যা হচ্ছিলো সবথেকে পুরানো key টা খুজে বের করতে। সেটার সমাধান করতে আমরা ডাবলি (Doubly) লিংকড-লিস্ট ব্যবহার করবো। ডাবলি লিংকড লিস্টে প্রতিটা নোড তার সামনের এবং পিছের নোডকে পয়েন্ট করে। নিচের ছবিটা দেখো, তাহলে ব্যাখ্যা করতে সুবিধা হবে:



আমি এবার আমার ডাটাগুলোকে একটা লিংকড লিস্টে রেখে দিয়েছি। লিস্টের প্রতিটা নোড একটা করে key-value পেয়ার। লিস্টের বামে হেড এর কাছে থাকবে সবথেকে নতুন ডাটা, ডানে থাকবে সবথেকে পুরানো ডাটা। সেই সাথে আমার আগের মতোই একটা হ্যাশম্যাপ থাকবে তবে এবার হ্যাশম্যাপে শুধুমাত্র ভ্যালুর বদলে থাকবে লিংকড লিস্টের নোডের অ্যাড্রেস।

এবার যখন একটা ডাটা আসবে তখন খুজে দেখতে হবে যে ডাটাটা আগেই হ্যাশম্যাপে আছে নাকি। যদি থেকে থাকে তাহলে লিংকড লিস্ট থেকে সেই নোডটা আগে ডিলিট করে দিতে হবে। এরপর নতুন একটা নোড লিস্টের বামে বসিয়ে দিতে হবে। কোনো ডাটা যখন কুয়েরি করা হবে তখনও একই কাজ করতে হবে কারণ যে ডাটা কুয়েরি করা হচ্ছে সবথেকে নতুন ডাটা।

ডাবলি লিংকড লিস্টে নোড ডিলিট করা খুব সহজ, পিছের নোডটাকে সামনের নোডের সাথে পয়েন্ট করিয়ে দিলেই হয়ে যায়। আর হ্যাশম্যাপে নোডের অ্যাড্রেস রেখে দেয়ার কারণে  $O(1)$  এভাবেজেই নোডটা খুজে পাওয়া যাবে।

তাহলে এখন আর Put অপারেশনের জন্য সবথেকে পুরানো ডাটা লুপ চালিয়ে খুজে বের করা লাগবে না। যদি ক্যাশের ক্যাপাসিটি পূর্ণ হয়ে যায় তাহলে সবথেকে বামের নোডটাকে ডিলিট করে দিলেই হয়ে যাচ্ছে যেটা  $O(1)$  এই করা সম্ভব।

তাহলে আমরা এমন একটা অ্যালগরিদম পেয়ে গেলাম যেটা  $O(1)$  এ LRU ক্যাশ থেকে ডাটা রিড-রাইট করতে পারে।

আমরা সি++ এ কোডটা ইমপ্লিমেন্ট করতে পারি। শুরুতেই আমি লিংকড লিস্টের নোড ক্লাস ডিফাইন করবো:

```
1 struct Node {
2     string key;
3     string value;
4     Node* prev;
5     Node *next;
6
7     Node() {
8         prev = NULL;
9         next = NULL;
10    }
11
12    Node(string _key, string _value) {
13        key = _key;
14        value = _value;
15        Node();
16    }
17};
```

লিংকড লিস্টটা ইমপ্লিমেন্ট করাই এই কোডের কঠিন কাজ। মেইন লজিক সহজ হয়ে যাবে যদি আমরা লিংকড লিস্টের একটা ক্লাস তৈরি করে ফেলি। আমাদের লিংকড লিস্টের বামে নোড চুকানো আর লিস্ট থেকে নোড ডিলিট করার মেথড লাগবে।

```
1 struct DoublyList {
2     Node* head;
3     Node* tail;
4
5     DoublyList() {
6         head = new Node();
7         tail = new Node();
8
9         head->next = tail;
10        tail->prev = head;
11    }
12
13    void erase(Node *curr) {
14        curr->prev->next = curr->next;
15        curr->next->prev = curr->prev;
16    }
}
```

```

17
18     void insertHead(Node* curr) {
19         Node* temp = head->next;
20         head->next = curr;
21         curr->next = temp;
22         temp->prev = curr;
23         curr->prev = head;
24     }
25
26     string popTail() {
27         Node* last = tail->prev;
28         string key = last->key;
29         last->prev->next = tail;
30         tail->prev = last->prev;
31         return key;
32     }
33 };

```

আমাদের বেসিক স্টোকচার বেডি, এবাব ক্যাপ্শন্টি ইমপ্লিমেন্ট করি:

```

1 class LRUCache {
2
3     private:
4         int capacity;
5         map<string, Node*> cache;
6         DoublyList *list;
7
8     public:
9
10    LRUCache(int _capacity) {
11        capacity = _capacity;
12        list = new DoublyList();
13    }
14
15    string get(string key) {
16        if (cache.find(key) == cache.end()) {
17            return "NOT FOUND";
18        }
19
20        Node *curr = cache[key];
21        list->erase(curr);
22        list->insertHead(curr);
23
24        return curr->value;

```

```

25 }
26
27 void put(string key, string value) {
28     Node* newNode = new Node(key, value);
29
30     if (cache.find(key) != cache.end()) { //Found in hashmap
31         Node *curr = cache[key];
32         curr->value = value;
33
34         list->erase(curr);
35         list->insertHead(curr);
36     } else {
37         cache[key] = newNode;
38         list->insertHead(newNode);
39
40         if (cache.size() > capacity) {
41             Node* lastNode = list->popLast();
42             cache.erase(lastNode->key);
43         }
44     }
45 }
46 };

```

কোডটা স্টেপ বাই স্টেপ ফলো করলেই বুঝে যাবে কিভাবে এটা কাজ করে। তুমি যদি পর্যাকটিস করতে চাও তাহলে [লিটকোডে নিজের কোড](#) সাবমিট করতে পারো। LRU ক্যাশ খুবই কমন একটা ইন্টারভিউ কোশেন।

বাস্তবে যত্তে তোমার কখনোই এটা ইমপ্লিমেন্ট করতে হবে না, বেশিভাগ ক্যাশিং মেকানিজমই LRU মেথড সাপোর্ট করে, যেমন [বেডিস](#)। ক্যাশের ডাটার আরো কিছু প্রোপার্টি থাকে, যেমন TTL বা Time to live, এটা দিয়ে ডিফাইন করা হয় কতক্ষণ পর একটা ডাটা একাই এক্সপায়ার হয়ে যাবে। তুমি চিন্তা করে বের করার চেষ্টা করতে পারো সেটা কিভাবে ইমপ্লিমেন্ট করতে হয়!

হ্যাপি কোডিং!

## ফেসবুকে মন্তব্য

# প্রোবাবিলিস্টিক ডাটা স্ট্রাকচার: খুম ফিল্টার

মার্চ ১৪, ২০২০ by শাফায়েত



যদি তোমার একটি নিজস্ব ওয়েবসাইট থাকে তাহলে অবশ্যই সেখানে একধর্মী ইউজার থাকবে। আর প্রতিটি ইউজারের একটি নিজস্ব অ্যাকাউন্ট থাকবে যেখানে তারা নিজেদের ইউজারনেম এবং পাসওয়ার্ড দিয়ে লগইন করতে পারবে। এখন নতুন কোনো ইউজার যদি অ্যাকাউন্ট খুলতে চায় তখন তোমাকে খেয়াল বাধতে হবে যে, সে যেই ইউজারনেমটি ব্যবহার করছে সেই একই ইউজারনেম ব্যবহার করে ইতোমধ্যেই অন্য কেও অ্যাকাউন্ট তৈরি করেছে কিনা।



ধৰা যাক, সবগুলো ইউজারনেম তুমি আগেই একটি ডাটাবেসে সেভ করে রেখেছো। প্রতিবার যখন নতুন ইউজার রেজিস্টার করার চেষ্টা করে তখন কিভাবে তুমি চেক করবে যে ইউজারনেমটি ডুপ্লিকেট নাকি? সহজ উপায় হলো প্রতিবার ইউজারনেমটিকে সার্ভারে পাঠিয়ে ডাটাবেসে চেক করা।

এখন মনে করো তোমার ওয়েবসাইট খুবই বিখ্যাত, ইউজার আছে ১০ লাখ এবং প্রতি মিনিটেই কয়েক হাজার নতুন ইউজার নতুন অ্যাকাউন্ট তৈরি। সেক্ষেত্রে ডুপ্লিকেট চেক করার জন্য সার্ভারে অনেক রিকুয়েস্ট পাঠাতে হবে, ডাটাবেসও এক্সেস করতে হবে অনেকবার। এক্ষেত্রে তুমি একটা ক্যাশ ব্যবহার করে ডাটাবেস এক্সেস কমিয়ে আনতে পারো কিন্তু সবথেকে ভালো হয় যদি সার্ভারে রিকুয়েস্ট না পাঠিয়েই আমরা ডুপ্লিকেট চেক করতে পারি। একটা উপায় হলো ১০লাখ ইউজারের লিস্ট ক্লায়েন্টের ব্রাউজারে পাঠিয়ে দেয়া এবং সেখানে ডুপ্লিকেশন চেক করা কিন্তু তাহলে প্রচুর ডাটা ব্রাউজারে পাঠাতে হবে, এবং ইউজারনেমের তালিকাও ফাঁস হয়ে যাবে সবার কাছে।

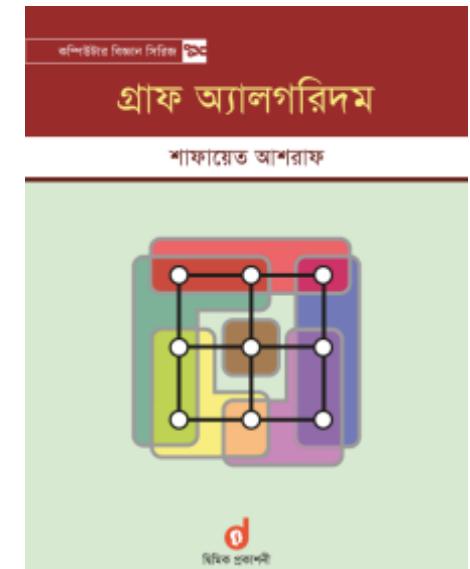
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



এই ধরণের সমস্যা সমাধানের জন্য ব্যবহার করা হয় বুম ফিল্টার নামক একটা প্রোগ্রামিস্টিক ডাটা স্টোকচার। এটা খুবই সিম্পল একটা ডাটা স্টোকচার, তোমার যদি হ্যাশিং নিয়ে ধারণা থাকে তাহলে বুঝতে কোনো সমস্যা হবে না।

বুম ফিল্টারের বৈশিষ্ট হলো তুমি যদি এখানে একটা ইউজারনেম ইনপুট দাও তাহলে সে দুইরকম আউটপুট দিবে, একটি হলো, “আমি নিশ্চিত ইউজার নেমটি ডাটাবেজে নাই”; আরেকটি হলো, “আমি প্রায় নিশ্চিত যে ইউজারনেমটি ডাটাবেসে আছে”। এখানে লক্ষ্য করার বিষয় হলো বুম ফিল্টার যদি বলে যে ইউজার নেমটি ডাটাবেসে নাই তাহলে ১০০% নিশ্চিত যে নামটি নেই; কিন্তু যদি বলে ডাটাবেসে আছে তাহলে সেটা থাকতেও পারে, নাও থাকতে পারে এবং নিশ্চিত হতে হলে তোমাকে ডাটাবেসে খুজে দেখতে হবে।

বুম ফিল্টার হলো  $m$  বিটের একটি ডেষ্টের। একদম শুরুতে সবগুলো বিটের মান শূন্য।

0	0	0	0	0	0	0	0	0	0
0	1	2	3	4	5	6	7	8	9

আমাদের এই উদাহরণে ধরে নিবো  $m = 9$ । এখন আমাদেরকে  $k$  টা হ্যাশ ফাংশন ডিফাইন করতে হবে। প্রতিটা হ্যাশ ফাংশনের কাজ হলো একটি ইউজারনেম ইনপুট হিসাবে নিয়ে সেটাকে  $m$  সাইজের বিট ডেষ্টের একটি ধরে অ্যাসাইন করা। উদাহরণ হিসাবে ধরে নিলাম  $k = 3$ । এখন মনে করো প্রথম ইউজারের নাম হলো “BOB”। “BOB” কে 3 টা হ্যাশ ফাংশন দিয়ে হ্যাশ করে আমরা ধরো পেয়েছি 1, 4 এবং 7। আমরা বুম ফিল্টারে সেই বিটগুলোকে অন করে দিবো।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

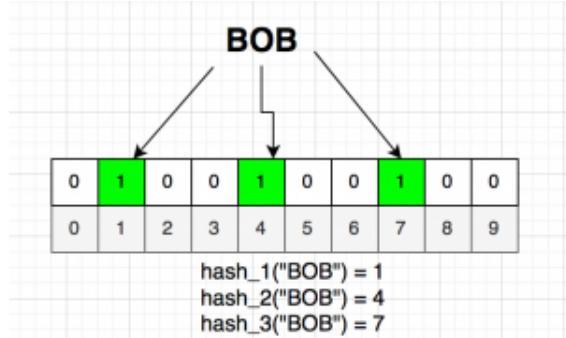
ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

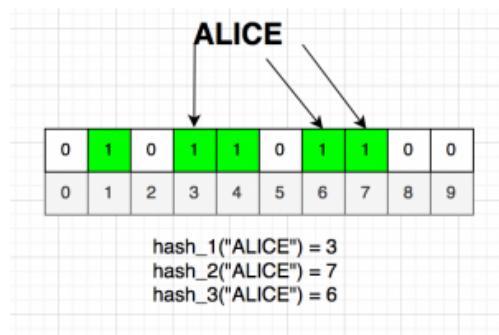
লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



এবং ইউজারের নাম ALICE এবং ALICE কে হ্যাশ করে পেয়েছি 3, 7 এবং 6। আমরা বিটগুলোকে অন করে দেই।



এখন আমাদের ব্লুম ফিল্টারে ALICE এবং BOB নামদুটি সেভ করা আছে।

এখন নতুন একটা শব্দ ব্লুম ফিল্টারে আছে নাকি নেই সেটা বুঝতে হলে তোমাকে আগে সেই নামটাকে  $k$  বার হ্যাশ করতে হবে এবং  $k$  টি পজিশনে গিয়ে দেখতে হবে বিটটি অন আছে নাকি। যদি তুমি অস্তত একটি বিট অফ পাও তার মানে শব্দটি নিশ্চিতভাবেই ব্লুম ফিল্টারে নেই।

ব্লাইডিং রেজ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

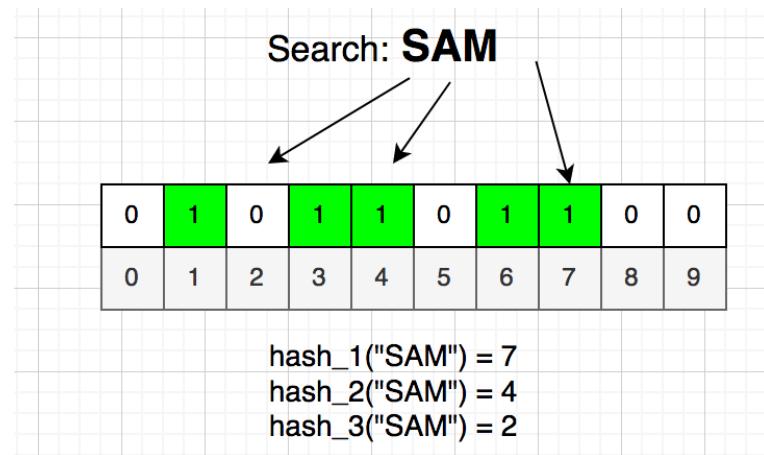
টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
সেট

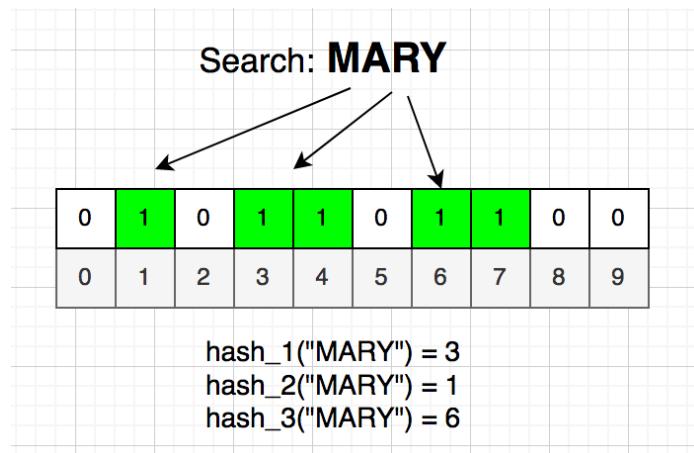
ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



কিন্তু তুমি যদি  $k$  টি পজিশনেই বিট অন পাও তাহলে কিন্তু নিশ্চিতভাবে বলতে পারবে না শব্দটি আছে নাকি নেই। যেমন ধরা যাক "MARY" নামটি হ্যাশ করার পর আমরা পেয়েছি 3, 1 এবং 6। এর আগে ALICE থেকে থেকে পেয়েছিলাম {3, 7, 6} আর BOB থেকে {1, 4, 7}। যখন আমরা MARY খুজবো তখন সবগুলো বিটই অন পাবো, কিন্তু শব্দটা আমরা আগে সেভ করিনি।



এজন্যই আগেই বলেছি ক্লুম ফিল্টার একটা প্রোবাইলিস্টিক ডাটা স্টুকচার। এটা মাঝে মাঝে False Positive দেয়, অর্থাৎ শব্দটি ডেটাবেজে না থাকলেও বলে যে আছে।

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কডার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

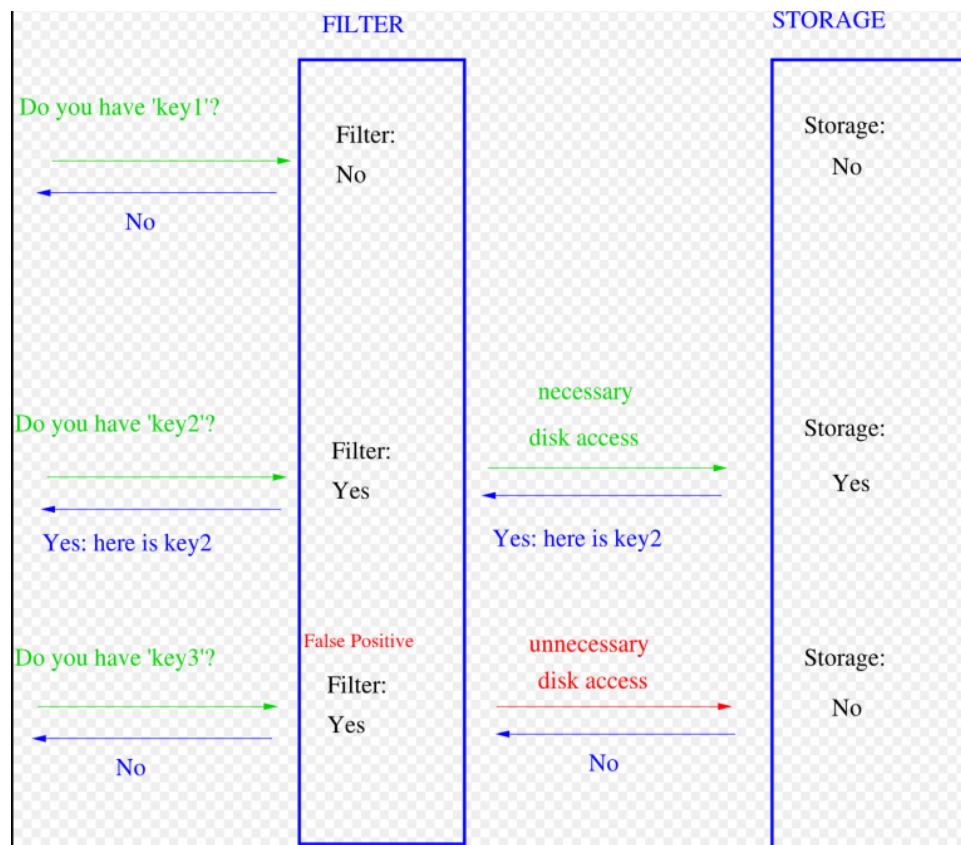
ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

এখন আমরা সার্ভারে প্রতিটা ইউজারনেম হ্যাশ করে ব্লুম ফিল্টার তৈরি করে বাখবো। প্রতিবার ইউজার বেজিস্টার ফর্ম ওপেন করলে আমরা বিট ডেক্সট্রি ইউজারের ব্রাউজারে পাঠিয়ে দিবো। ইউজার যখন কোনো নাম টাইপ করবে তখন আমরা সাথে সাথে সেই ফিল্টার চেক করে বলে দিতে পারবো যে ওই নামের কোনো ইউজার ডাটাবেসে আছে নাকি। বেশিরভাগ সময়ই ব্লুম ফিল্টার নেগেটিভ আউটপুট দিবে কারণ নতুন ইউজারনেম সাধারণত অন্য নামের সাথে মিলবে না কিন্তু যদি মিলে যায় তাহলে আমাদের সার্ভারে বিকুয়েস্ট পাঠিয়ে ডাবল-চেক করতে হবে।

নিচের ছবিটি উইকিপিডিয়া থেকে নেয়া:



এভাবে ব্লুম ফিল্টার ব্যবহার করে তুমি সার্ভার বা ডাটাবেস এক্সেস অনেক কমিয়ে আনতে পারবে। ব্লুম ফিল্টারের পারফরমেন্স বাড়াতে হলে আমাদেরকে ফলস পজিটিভ কমাতে হবে। ব্লুম ফিল্টারের এরোর রেট মূলত নির্ভর করে হ্যাশ ফাংশনের সংখ্যা ( $k$ ) এবং ফিল্টারের সাইজের ( $m$ )

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্স(১):

ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

উপর। নিচের ফর্মুলা ব্যবহার করে এরোর বেট বের করা যায়:

$$p \approx (1 - e^{-\frac{kn}{m}})^k$$

এখানে  $k$  = হ্যাশ ফাংশনের সংখ্যা,  $m$  = বিট ভেস্টের সাইজ,  $n$  = ইনপুটের সাইজ। ফর্মুলা কিভাবে ডিরাইড করা হয়েছে সেটা আমি এখানে বর্ণনা করবো না, তুমি চাইলে অন্য কোথাও থেকে পড়ে নিতে পারো।

হ্যাশ ফাংশনের সংখ্যা বাড়লে অথবা  $m$  এর সাইজ বাড়লে এরোর বেট কমে আসবে। কিন্তু তুমি যদি বেশিবার হ্যাশ করবে তত বেশি সময় লাগবে ক্যালকুলেশন করতে, আবার  $m$  এর মান বাড়লে মেমরি বেশি লাগবে। আজকাল **অনেক টুলস আছে** যেগুলো ব্যবহার করে তুমি এরোর বেট ক্যালকুলেট করতে পারো। যেমন **এই টুলটি** ব্যবহার করে আমি নিচের ডাটা পেয়েছি:

$$\begin{aligned}n &= 1,000,000 \\p &= 0.1 (1 \text{ in } 10) \\m &= 5,015,835 (612.28\text{KiB}) \\k &= 5\end{aligned}$$

এর মানে আমার যদি ১০ লাখ স্ট্রিং থাকে এবং আমি যদি চাই যে এরোর বেট হবে ০.১% এবং আমি ৫টি হ্যাশ ফাংশন ব্যবহার করবো, তাহলে আমার 612 কিলোবাইট সাইজের ব্লুম ফিল্টার ব্যবহার করতে হবে। দেখতেই পাচ্ছা ব্লুম ফিল্টার খুবই অল্প মেমরি ব্যবহার করে।

ব্লুম ফিল্টারের আরো কিছু ব্যবহার হলো:

তুমি তোমার সার্ভারে দুর্বল পাসওয়ার্ডের একটা তালিকা রেখে দিতে পারো (যেমন: 123456, password, abcdef)। ইউজার যখন রেজিস্টার করবে এবং নতুন পাসওয়ার্ড দিবে তখন ব্লুম ফিল্টার দিয়ে চেক করতে পারো যে পাসওয়ার্ডটি দুর্বল পাসওয়ার্ডের তালিকায় আছে নাকি।

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet  
2.4K likes  
শাফায়েত আশরাফ  
Like Page  
Be the first of your friends to like this

Shafaetsplanet about an hour ago  
ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিঙের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

i, W) = 1 if W =  
i + 1, W - C[i]) -

SHAFAEETPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / ম্যান

ବୁମ ଫିଲ୍ଟାର ଦିଯେ ସ୍ପେଲ-ଚେକାର ତୈରି କରା ଯେତେ ପାରେ । ଇଉଜାରେର ଟାଇପ କରା ଶବ୍ଦ ଡିକଷନାରିତେ ନା ଥାକଲେ ବୁମ ଫିଲ୍ଟାର ସେଟା ବଲେ ଦିବେ ।

ବୁମ ଫିଲ୍ଟାରେର ମତୋ ଆରେକଟା ଡାଟା-ସ୍ଟ୍ରାକ୍ଚାର ଆଛେ ଯେଟାର ନାମ “କାଉନ୍ଟ-ମିନ ସ୍କେଚ” ଯେଟା ଅନେକଟା ଏକଇ ଭାବେ କାଜ କରେ କିନ୍ତୁ ପ୍ରୟୋଗ ଭିନ୍ନ, ସେଟାଓଞ୍ଜ ଏଥିନ ଶିଖ ଫେଲେତ ପାରୋ ।

ଆଜ ଏହି ପର୍ଣ୍ଣତ୍ବ, ହ୍ୟାପି କୋଡ଼ିଂ ।

# গ্রাফ থিওরিতে হাতেখড়ি – ১

ডিসেম্বর ২১, ২০১০ by শাফায়েত



(গ্রাফ অ্যালগরিদম বইটি অনলাইনে পাওয়া যাচ্ছে [রকমারি.কম](#) এ। এছাড়াও নীলক্ষ্মতে হক, মানিক এবং রানা লাইব্রেরিতে বইটি পাওয়া যাচ্ছে)

তুমি কি জানো পৃথিবীর প্রায় সব রকমের প্রবলেমকে কিছু রাস্তা আর শহরের প্রবলেম বানিয়ে সলভ করে ফেলা যায়? গ্রাফ থিওরির যখন জন্ম হয় তখন তোমার আমার কাবোই জন্ম হ্যানি, এমনকি পৃথিবীতে কোন কম্পিউটারও ছিলোনি! সেই সতেরো দশকের শেষের দিকে লিওনার্ড অয়লার গ্রাফ থিওরি আবিষ্কার করেন কনিসবার্গের সাতটি ব্রিজের সমস্যার সমাধান করতে। তখনই সবার কাছে পরিষ্কার হয়ে যায় যে যেকোন প্রবলেমকে শহর-রাস্তার প্রবলেম দিয়ে মডেলিং করে চেনা একটা প্রবলেম বানিয়ে ফেলতে গ্রাফ থিওরির জুড়ি নেই। আর যখনই তুমি একটা প্রবলেমকে চেনা কোন প্রবলেমে কনভার্ট করে ফেলতে পারবে তখন সেটা সমাধান করা অনেক সহজ হয়ে যাবে।

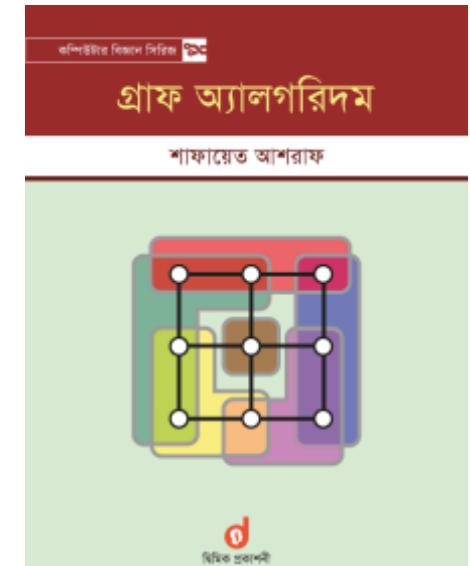
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)

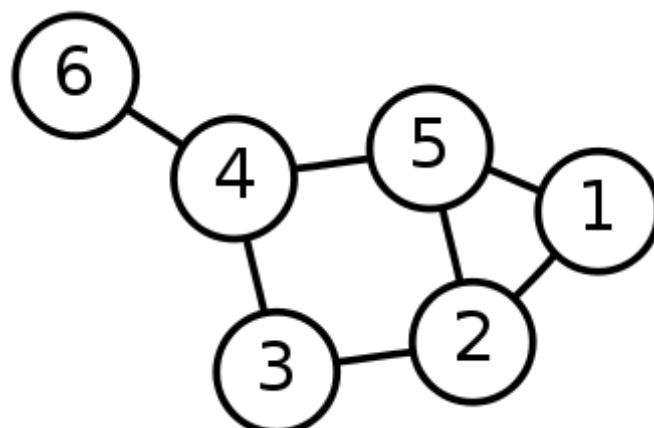


গ্রাফ থিওরির অনেক অ্যাপ্লিকেশন আছে। সবথেকে কমন হলো এক শহর থেকে আরেক শহরে যাবার দ্রুততম পথ খুজে বের করা। তুমি হয়তো জানো সার্ভার থেকে একটা ওয়েবপেজ তোমার পিসিতে পৌছাতে অনেকগুলো রাউটার পার করে আসতে হয়, গ্রাফ থিওরি দিয়ে এক রাউটার থেকে আরেকটাতে যাবার পথ খুজে বের করা হয়। যদ্দের সময় একটা দেশের কোন কোন রাষ্ট্র বোমা দিয়ে উড়িয়ে দিলে দেশের বাজধানী সব শহর থেকে বিচ্ছিন্ন হয়ে যাবে সেটাও বের করে ফেলা যায় গ্রাফ থিওরি দিয়ে। আমরা গ্রাফ অ্যালগোরিদমগুলো শেখার সময় আরো অনেক অ্যাপ্লিকেশন দেখবো।

আমাদের এই হাতেখড়ির লক্ষ্য হবে প্রথমেই গ্রাফ থিওরির একদম বেসিক কিছু সংজ্ঞা জানা, তারপর কিভাবে গ্রাফ মেমরিতে স্টোর করতে হয় সেটা জানা, এরপরে গ্রাফের কিছু বেসিক অ্যালগোরিদম জানা এবং সাথে সাথে মজার কিছু প্রবলেম দেখা। সব সংজ্ঞা একসাথে দেখলে মাথায় থাকবেনা, তাই একদম না জানলেই নয় সেরকম কিছু সংজ্ঞা প্রথমে দেখবো, এরপর প্রয়োজনমতো আরো কিছু সংজ্ঞা জেনে নিবো।

## গ্রাফ কি?

ধৰা যাক ৬টি শহরকে আমরা ১,২,৩,৪,৫,৬ দিয়ে চিহ্নিত করলাম। এবার যে শহর থেকে যে শহরে সরাসরি রাষ্ট্র আছে তাদের মধ্যে লাইন টেনে দিলাম:



শহরগুলোর নাম ১,২ ইত্যাদি দিয়ে দিতে হবে এমন কোন কথা নেই, তুমি চাইলে ঢাকা, চট্টগ্রাম ইত্যাদি দিতে পারো। এটা খুবই সাধারণ একটা গ্রাফ যেখানে কিছু শহরের মধ্যের রাষ্ট্রগুলো দেখানো হয়েছে। গ্রাফ থিওরির ভাষায় শহরগুলোকে বলা হয় **নোড(Node)** বা ভারটেক্স(Vertex) আর রাষ্ট্রগুলোকে বলা হয় **এজ(Edge)**। গ্রাফ হলো কিছু নোড আর কিছু এজ এর একটা কালেকশন।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগোরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

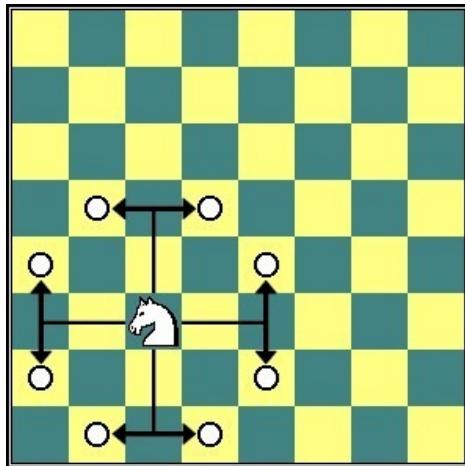
লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

গ্রাফে নোড দিয়ে অনেককিছু বুঝতে পারে, কোন গ্রাফে হয়তো নোড দিয়ে শহর বুঝায়, কোন গ্রাফে এয়ারপোর্ট, কোন গ্রাফে আবার হয়তো দাবার বোর্ডের একটা ঘর বুঝাতে পারে! আর এজ দিয়ে বুঝায় নোডগুলোর মধ্যের সম্পর্ক। কোন গ্রাফে এজ দিয়ে দুটি শহরের দূরত্ব বুঝাতে পারে, কোন গ্রাফে এক এয়ারপোর্ট থেকে আরেক এয়ারপোর্টে যাবার সময় বুঝাতে পারে, আবার দাবার বোর্ডে একটা ঘরে ঘোড়া থাকলে সেই ঘর থেকে কোন ঘরে যাওয়া যায় সেটাও বুঝাতে পারে।

নিচের ছবিতে দাবার বোর্ডটাও একটা গ্রাফ। প্রতিটা ঘর একটা করে নোড। যে ঘরে ঘোড়া আছে সেখান থেকে এজগুলো দেখানো হয়েছে:



এককথায় নোডের কাজ কোন একধরণের অবজেক্টকে রিপ্রেজেন্ট করা আর এজ এর কাজ হলো দুটি অবজেক্টের মধ্যে সম্পর্কটা দেখানো।

### অ্যাডজেস্ট নোড:

A নোড থেকে B নোডে একটা এজ থাকলে B কে বলা হয় A এর অ্যাডজেস্ট নোড। সোজা কথায় অ্যাডজেস্ট নোড হলো এজ দিয়ে সর্বাসবি কানেক্টেড নোড। একটা নোডের অনেকগুলো অ্যাডজেস্ট নোড থাকতে পারে।

### ডিরেক্টেড গ্রাফ আৰ আনডিরেক্টেড গ্রাফ:

ডিরেক্টেড গ্রাফে এজগুলোতে তীব্রচিহ্ন থাকে, তাৰমানে এজগুলো হলো একমুখি(Unidirectional), আনডিরেক্টেড গ্রাফে এজগুলো দ্বিমুখী(Bidirectional)। নিচের ছবি দেখলৈ পৰিষ্কাৰ হবে:

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েৰি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্ৰাই(প্ৰিফ্ৰিঙ্ক ট্ৰি/বেডিঙ্ক ট্ৰি)

সেগমেন্ট ট্ৰি-১

সেগমেন্ট ট্ৰি-২(লেজি প্ৰপাগেশন)

অ্যারে কমপ্ৰেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টৱ

বাইনারি ইনডেক্সড ট্ৰি

স্থায়াৱ-কুট ডিকম্পোজিশন ক্যাশিং অ্যালগোরিদম:

এল-আৱ-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টাৰ(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেস্ট ম্যাট্ৰিক্স

অ্যাডজেস্ট লিস্ট

ব্ৰেথড ফার্স্ট সাৰ্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্ৰি ১ (প্ৰিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্ৰি ২ (ক্ৰুসকাল অ্যালগোরিদম)

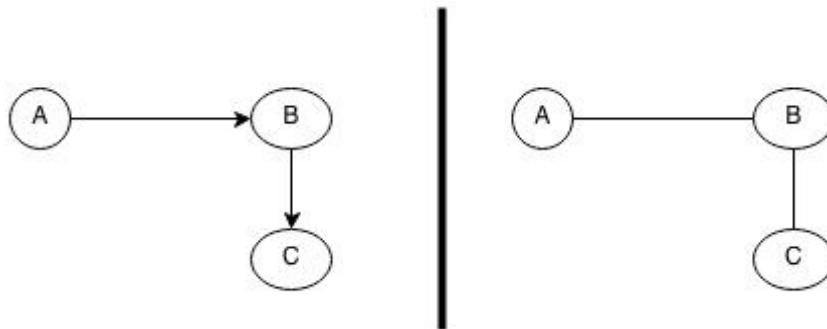
টপোলজিকাল সেট

ডেপথ ফার্স্ট সাৰ্চ এবং আবাৰো টপোলজিকাল  
সেট

ডায়াক্রস্টা

ফ্ৰয়েড ওয়াৰ্শল

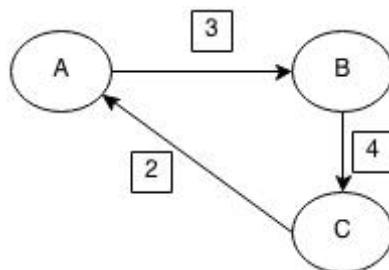
বেলম্যান ফোৰ্ড



বামের ছবির গ্রাফ ডিরেক্টড, ডানেরটা আনডিরেক্টড।

### ওয়েটেড আর আনওয়েটেড গ্রাফ:

অনেক সময় গ্রাফে এজগুলোর পাশে ছোট করে ওয়েট(Weight) বা কস্ট(Cost) লেখা থাকতে পারে:



এই ওয়েট বা কস্ট দিয়ে অনেককিছু বুঝাতে পারে, যেমন দুটি শহরের দূরত্ব কত কিলোমিটার, অথবা বাস্তাটি দিয়ে যেতে কত সময় লাগে, অথবা রাস্তা দিয়ে কয়টা গাড়ি একসাথে যেতে পারে ইত্যাদি। আগের গ্রাফগুলো ছিলো আনওয়েটেড, সেক্ষেত্রে আমরা ধরে নেই সবগুলো এজের ওয়েটের মান এক(1)। সবগুলো ওয়েট 1 হলে আলাদা করে লেখা দরকার হ্যন।

### পাথ:

পাথ(Path) হলো যে এজগুলো ধরে একটা নোড থেকে আরেকটা নোডে যাওয়া যায়। অর্থাৎ এজের একটা সিকোয়েন্স।

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পানেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেষ্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

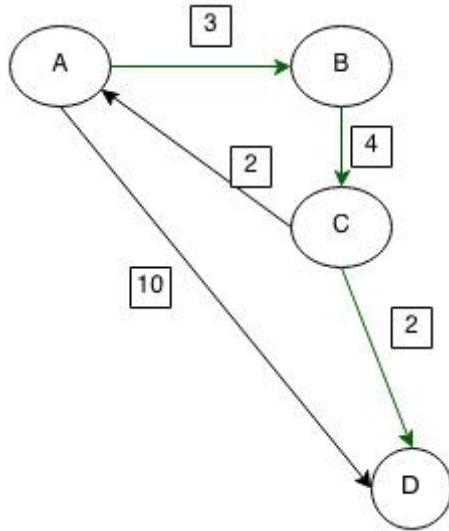
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেষ্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)



এক নোড থেকে আরেক নোডে যাবার অনেকগুলো পাথ থাকতে পারে। ছবিতে A থেকে D তে যাবার দুইটা পাথ আছে। A->B, B->C, C->D হলো একটা পাথ, এই পাথের মোট ওয়েট হলো  $3+4+2=9$ । আবার A->D ও একটা পাথ হতে পারে যেই পাথের মোট কস্ট 10। যে পাথের কস্ট সবথেকে কম সেটাকে শর্টেস্ট পাথ বলে।

### ডিগ্রী:

ডিরেক্টেড গ্রাফে একটা নোডে কয়টা এজ প্রবেশ করেছে তাকে **ইনডিগ্রী**, আর কোন নোড থেকে কয়টা এজ বের হয়েছে তাকে **আউটডিগ্রী** বলে। ছবিতে প্রতিটা নোডের ইনডিগ্রী আর আউটডিগ্রী দেখানো হয়েছে:

[পূর্বনো সিরিজ]

বিটমান্স ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্রিং(১):

ব্যকট্যাক্রিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

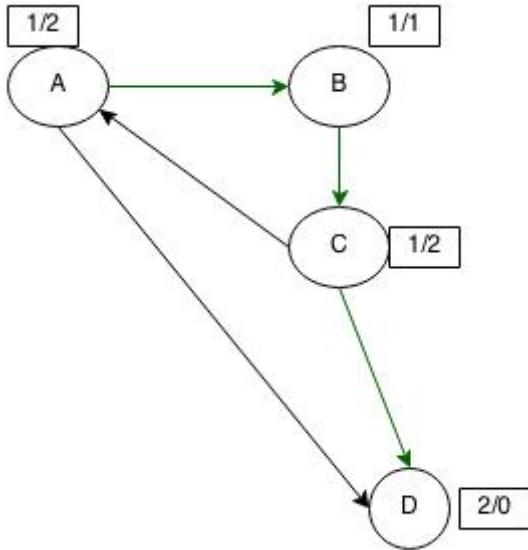
অন্যান্য(৩) :

ডিরেকশন অ্যারে

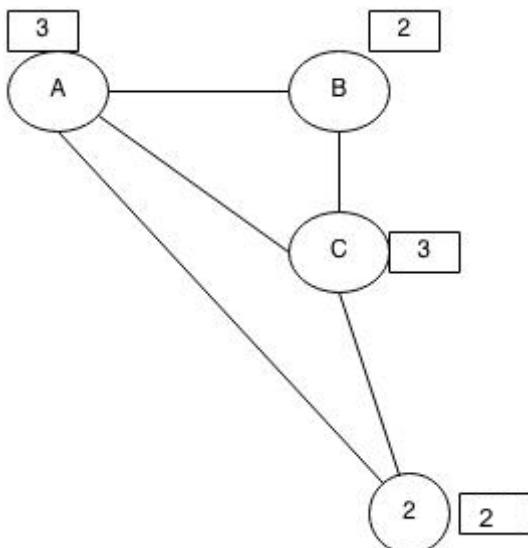
মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



আনডিরেটেড গ্রাফে ইন বা আউটডিগ্রী আলাদা করা হয়ন। একটা নোডের যতগুলো অ্যাডজেসেন্ট নোড আছে সেই সংখ্যাটাই নোডটার ডিগ্রী।



কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Be the first of your friends to like this



$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / মার

হ্যান্ডশেকিং লেমা একটা জিনিস আছে যেটা বলে একটা বিজোড় ডিগ্রীর নোডের সংখ্যা সবসময় জোড় হয়। উপরের গ্রাফে A আর C এর ডিগ্রী 3, এবং বিজোড় ডিগ্রীর নোড। তাহলে বিজোড় ডিগ্রীর নোড আছে ২টা, ২ হলো একটা জোড় সংখ্যা। হ্যান্ডশেক করতে সবসময় ২টা হাত লাগে, ঠিক সেরকম একটা এজ সবসময় ২টা নোডকে যোগ করে। তুমি একটু চিন্তা করে দেখো:

“ ২টা জোড় ডিগ্রীর নোডকে এজ দিয়ে যোগ করলে ২টা নতুন বিজোড় ডিগ্রীর নোড তৈরি হয়।

২টা বিজোড় ডিগ্রীর নোডকে এজ দিয়ে যোগ করলে ২টা বিজোড় ডিগ্রীর নোড কমে যায়।

১টা জোড় আর একটা বিজোড় ডিগ্রীর নোড যোগ করলে মোট বিজোড় ডিগ্রীর নোড সমান থাকে(এক পাশে কমে, আবেক পাশে বাড়ে)।

তাহলে দেখা যাচ্ছে হ্য ২টা করে বাড়তেসে বা ২টা করে কমতেসে বা সমান থাকছে, তাই বিজোর ডিগ্রীর নোডের সংখ্যা সবসময় জোড়।

একইভাবে এটাও দেখানো যায় একটা গ্রাফের ডিগ্রীগুলোর যোগফল হবে এজসংখ্যার দ্বিগুণ। উপরের গ্রাফে ডিগ্রীগুলোর যোগফল ১০, আর এজসংখ্যা ৫।

এগুলো গেল একেবারেই প্রাথমিক কথাবার্তা। পরবর্তি লেখায় তুমি জানতে পারবে কিভাবে **ড্যারিয়েলে গ্রাফ স্টোর** করতে হয়। আশা করি গ্রাফ খিওরীতে তোমার যাত্রাটা দারুণ আনন্দের হবে, অনেক কিছু শিখতে পারবে।

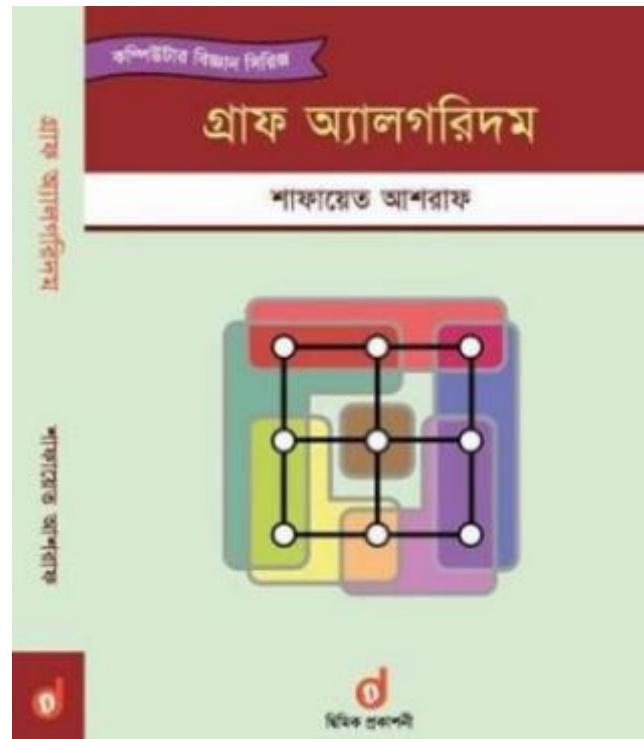
অন্যান্য পর্ব

ফেসবুকে মন্তব্য

2 comments

# গ্রাফ থিওরিতে হাতেখড়ি – ২ (ড্যারিয়েবলে গ্রাফ স্টোর-১)

ডিসেম্বর ২৬, ২০১০ by শাফায়েত



আগের পোস্টে আমরা দেখেছি গ্রাফ থিওরি কি কাজে লাগে, আব এলিমেন্টারি কিছু টার্ম শিখেছি। এখন আমরা আবেকষ্ট ভিতরে প্রবেশ করবো।  
প্রথমেই আমাদের জানা দরকার একটা গ্রাফ কিডাবে ইনপুট নিয়ে স্টোর করে রাখা যায়। অনেকগুলো পক্ষতির মধ্যে দুটি খুব কমন:

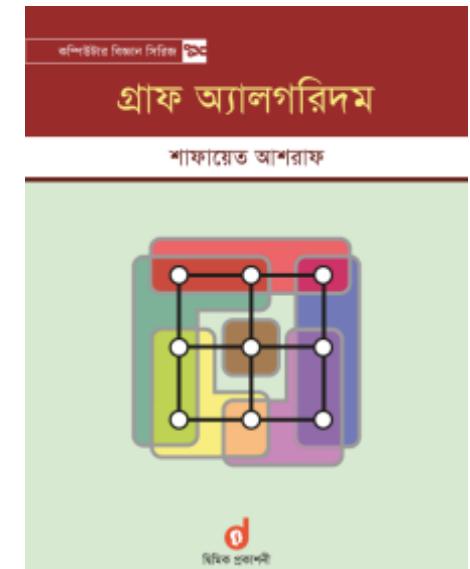
সাবস্ক্রাইব



Secured by OneAll Social Login

আমার সম্পর্কে

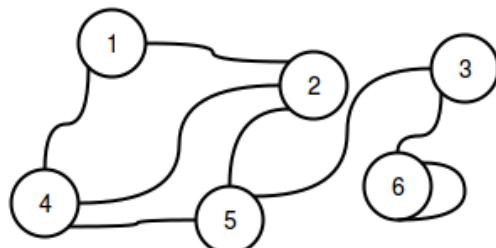
শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



## ১. অ্যাডজেসেন্সি ম্যাট্রিক্স(adjacency matrix)

## ২. অ্যাডজেসেন্সি লিস্ট(adjacency list)

অ্যাডজেসেন্ট(adjacent) শব্দটার অর্থ “কোন কিছুর পাশে”। যেমন তোমার পাশের বাড়ির প্রতিবেশিকা তোমার অ্যাডজেসেন্ট। গ্রাফের ভাষায় এক নোডের সাথে আরেকটা নোডে যাওয়া গেলে ২য় নোডটি প্রথমটির অ্যাডজেসেন্ট। এই পোস্টে আমরা ম্যাট্রিক্সের সাহায্যে কোন নোড কার অ্যাডজেসেন্ট অর্থাৎ কোন কোন নোডের মাঝে এজ আছে সেটা কিভাবে স্টোর করা যায় দেখবো। ম্যাট্রিক্স বলতে এখনে শুধুমাত্র ২-ডি অ্যারে বুঝানো হয়েছে, তাই ঘাবড়ে যাবার কিছু নেই!



Nodes	1	2	3	4	5	6
1	0	1	0	1	0	0
2	1	0	0	1	1	0
3	0	0	0	0	1	1
4	1	1	0	0	1	0
5	0	1	1	1	0	0
6	0	0	1	0	0	1

গ্রাফের পাশে একটি টেবিল দেখতে পাচ্ছ। এটাই আমাদের অ্যাডজেসেন্সি ম্যাট্রিক্স। ম্যাট্রিক্সের  $[i][j]$  ঘরে 1 থাকে যদি  $i$  থেকে  $j$  তে কোনো এজ থাকে, না থাকলে 0 বসিয়ে দেই।

এজগুলা ওয়েটেড হতে পারে, যেমন ঢাকা থেকে চট্টগ্রামে একটা এজ দিয়ে বলে দিতে পারে শহর দূরত্ব ৩০০ কিলোমিটার। তাহলে তোমাকে ম্যাট্রিক্সে ওয়েটও বসাতে হবে।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

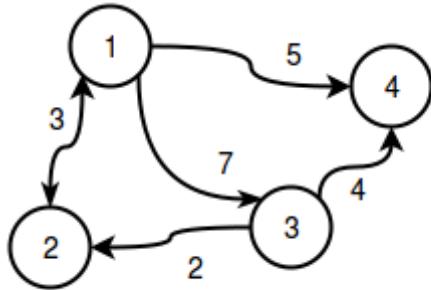
ডাটা স্টোরেজ(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

উপরের গ্রাফটি বাইডিরেকশনাল বা আনডিরেস্টেড, অর্থাৎ ১ থেকে ২ এ যাওয়া গেলে ২ থেকে ১ এও যাওয়া যাবে। যদি গ্রাফটি ডিরেক্টেড হতো তাহলে এজগুলোর মধ্যে তীব্রচিহ্ন থাকতো। তখনো আমরা আগের মতো করেই ম্যাট্রিক্সে স্টোর করতে পারবো। নিচের ছবিতে ডিরেক্টেড ওয়েটেড গ্রাফের অ্যাডজেসেন্সি ম্যাট্রিক্সের উদাহরণ দেখানো হয়েছে।



Nodes	1	2	3	4
1	inf	3	7	5
2	3	inf	inf	inf
3	inf	2	inf	4
4	inf	inf	inf	inf

যেসব নোড এর ভিতর কোনো এজ নাই তাদেরকে এখানে ইনফিনিটি বা অনেক বড় একটা সংখ্যা দিয়ে দেখানো হয়েছে।

একটা ব্যাপার লক্ষ করো, গ্রাফ আনডিরেক্টেড হলে ম্যাট্রিক্সটি সিমেট্রিক হয়ে যায়, অর্থাৎ  $\text{mat}[i][j] = \text{mat}[j][i]$  হয়ে যায়।

### ছোট একটা এক্সারসাইজ:

কল্পনা কর একটি গ্রাফ যার ৩টি নোড আছে edge সংখ্যা ৩, এবং সবগুলো edge bidirectional। edge গুলো হলো ১-২(cost ৫), ২-৩(cost ৮), ১-৩(cost ৩)। এটার adjacency matrix টা কেমন হবে?

চট করে নিজেই খাতায় একে ফেলতে চেষ্টা কর এবং নিচের উত্তরের সাথে মিলিয়ে দেখো:

“ 0 5 3  
5 0 8  
3 8 0

ব্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েন্ট সেট(ইউনিয়ন ফাইন্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

আশা করি বুঝতে পারছ কিভাবে ম্যাট্রিক্সটি আকলাম। না বুঝলে উপরের অংশটা আরেকবার পড়ে ফেল।

## গ্রাফ ইনপুট যেভাবে দেয়া হবে:

ঠিক উপরের ম্যাট্রিক্সটা প্রোগ্রামিং প্রবলেমে ইনপুট হিসাবে দিয়ে দেয়া হতে পারে, শুরুতে শুধু নোড সংখ্যা বলে দিবে। লক্ষ্য কর এই ম্যাট্রিক্সটা ইনপুট নিতে আমাদের এজ সংখ্যা জানা জরুরী না। আমাদের একটি ড্যারিয়েবল লাগবে নোড সংখ্যা ইনপুট নিতে, আরেকটি ২-ডি অ্যারে লাগবে ম্যাট্রিক্স ইনপুট নিতে।

```
1 int N;
2 int matrix[100][100]; //এই সর্বোচ্চ 100 নোডের গ্রাফ স্টোর করা যাবে।
3
4 //ডিঙ্গেয়ার করার পরে ইনপুট নেবার পালা। খুব সহজ কাজ:
5 scanf("%d",&N);
6 for(int i=1;i<=N;i++)
7 for(int j=1;j<=N;j++)
8 scanf("%d" ,&matrix[i][j]);
```

সরাসরি ম্যাট্রিক্স না দিয়ে নোড সংখ্যা, edge সংখ্যা বলে দিয়ে edge গুলো কি কি বলে দিতে পারে, এভাবে:

```
" 3 3 //৩ টা নোড এবং ৩টা এজ
 1 2 5 //node1-node2-cost
 2 3 8
 1 3 3
```

এটা ইনপুট নিব এভাবে:

```
1 int Node,Edge;
2 int matrix[100][100];
3 scanf("%d%d",&Node,&Edge);
4 for(i=0;i<Edge;i++)
5 {
```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাট্রিমাম ফ্লো-১

ম্যাট্রিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন  
সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১  
ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্বসেট সাম,  
কম্পিনেটরিঙ্গ, ডিসিশন প্রবলেম)

```

6 int n1,n2,cost;
7 scanf("%d%d%d",&n1,&n2,&cost);
8 matrix[n1][n2]=cost;
9 matrix[n2][n1]=cost;
10 }

```

আবো অনেক উপায়ে প্রবলেমে গ্রাফ ইনপুট দিতে পারে। নোডের নম্বর এলোমেলো হতে পারে, যেমন ৩টি নোডকে ১, ২, ৩ দিয়ে চিহ্নিত না করে ১০০, ১০০০০, ৮০০ নামে চিহ্নিত করা হতে পারে। সেক্ষেত্রে আমাদের ম্যাপিং করতে হবে। অর্থাৎ ১০০ কে আমরা ম্যাপ করব ১ দিয়ে, মানে ১০০ বলতে বুঝব ১, ১০০০০ বলতে বুঝব ২। index নামক একটি array রেখে  $index[100]=1; index[100000]=2;$  এভাবে চিহ্নিত করে দিলেই চলবে। পরে নোড নম্বর ইনপুট দিলে আমার ইনডেক্স থেকে আমাদের দেয়া নম্বর বের করে আনব। ব্যাপারটাকে বলা হয় অ্যারে কম্প্রেশন, তুমি বিশ্বাসিত জানতে চাইলে পরে কোনো সময় **আমার এই লেখাটা দেখতে পারো।**

### অ্যাডজেসেন্সি ম্যাট্রিক্স ব্যবহার করার সমস্যা:

মেমরি একটা বিশাল প্রবলেম, এজ যতগুলোই থাকুকনা কেন তোমার লাগছে  $N * N$  সাইজের ম্যাট্রিক্স যেখানে  $N$  হলো নোড সংখ্যা। 10000 টা নোড হলো  $N * N$  ম্যাট্রিক্সের সাইজ দাঢ়াবে  $4 * 1000 * 1000$  বাইট বা প্রায় 381 মেগাবাইট! এজ কম হলে এটা মেমরির বিশাল অপচয়।

কোনো একটা নোড  $u$  থেকে অন্য কোন কোন নোডে যাওয়া যায় বের করতে হলে আমাদের  $N$  টা নোডের সবগুলো চেক করে দেখতে হবে, টাইমের বিশাল অপচয়!

### অ্যাডজেসেন্সি ম্যাট্রিক্স ব্যবহার করার সুবিধা:

$u - v$  নোডের মধ্যে কানেকশন আছে নাকি বা cost কত সেটা খুব সহজেই  $mat[u][v]$  চেক করে জেনে যেতে পারি।

এই সমস্যাগুলা দূর করে দিবে অ্যাডজেসেন্সি লিস্ট, সাথে নতুন কিছু সমস্যাও হাজির করবে! তোমরা পরের পর্বে সেটা শিখবে। তার আগে তোমাকে একটা জিনিস শিখতে হবে, সেটা হলো C++ এর স্ট্যান্ডার্ড টেমপ্লেট লাইব্রেরি(STL)। আমরা STL এর ডেষ্টের ব্যবহার করে কাজ করবো কারণ এটা ব্যবহার করা খুব সহজ। তুমি নিচের দুটি লিংকের সাহায্যে খুবই সহজে শিখতে পারবে:

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাম্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

১: <http://sites.google.com/site/smilitude/stl> এটি ফাহিম ভাইয়ের ব্লগের লিংক, তার টিউটোরিয়াল গুলো অবিতীয়।

২: <http://www.cplusplus.com/reference/stl/> STL এর বিভিন্ন ফাংশনের কাজ শেখাব জন্য সেরা সাইট।

ডেষ্টের ব্যবহার শেখা হয়ে গেলে পড়া শুরু করো পরের পর: **adjacency list**

## ফেসবুকে মন্তব্য

0 comments

0 Comments

Sort by Oldest



Add a comment...

Facebook Comments plugin

Powered by [Facebook Comments](#)



Posted in অ্যালগরিদম/প্রবলেম সলভিং, প্রোগ্রামিং, সি/সি++ ? Tagged গ্রাফ থিওরি, সি++

69,213 বার পড়া হয়েছে

◀ গ্রাফ থিওরিতে হাতেখড়ি - ১

গ্রাফ থিওরিতে হাতেখড়ি - ৩ (ভ্যারিয়েবলে গ্রাফ স্টোর-২) ▶

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet  
2.4K likes  
**গ্রাফ অ্যালগরিদম**  
শাফায়েত আশরাফ

Like Page

Be the first of your friends to like this



Shafaetsplanet  
about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন  
সিরিজের ৬ নম্বর লেখায় আমরা  
দেখবো কম্বিনেটরিক্সের প্রবলেম এবং  
ডিসিশন প্রবলেম কিভাবে সমাধান  
করতে হ্য।

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM  
দাইনামিক প্রোগ্রামিং ৬ / মার

# গ্রাফ থিওরিতে হাতেখড়ি – ৩ (ভ্যারিয়েবলে গ্রাফ স্টোর-২)

ডিসেম্বর ২৭, ২০১০ by শাফায়েত

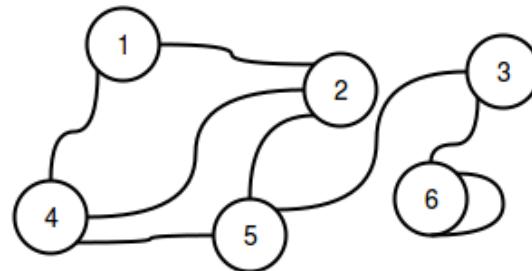


আগের পর

সবগুলো পর্ব

এই পর্বে গ্রাফ স্টোর করার ২য় পদ্ধতি অ্যাডজেসেন্সি লিস্ট নিয়ে লিখিব। এ পদ্ধতিতে গ্রাফ স্টোর করে কম মেমরি ব্যবহার করে আরো efficient কোড লেখা যায়। এ ক্ষেত্রে আমরা ডায়নামিক্যালি মেমরি অ্যালোকেট করব, ভয়ের কিছু নেই সি++ এর standard template library(STL) ব্যবহার করে খুব সহজে কাজটা করা যায়। আগের লেখার শেষের দিকে STL এর উপর কয়েকটি টিউটোরিয়ালের লিংক দিয়েছি, আশা করছি ডেস্টের কিভাবে কাজ করে এখন তুমি জানো।

অ্যাডজেসেন্সি লিস্ট শুনতে ঘটটা ভয়ংকর শুনায়, ব্যাপারটি আসলে তেমনই সহজ। আমরা আবার আগের পোস্টের ছবিটিতে ফিরে যাই:



Nodes	1	2	3	4	5	6
1	0	1	0	1	0	0
2	1	0	0	1	1	0
3	0	0	0	0	1	1
4	1	1	0	0	1	0
5	0	1	1	1	0	0
6	0	0	1	0	0	1

সাবস্ক্রাইব

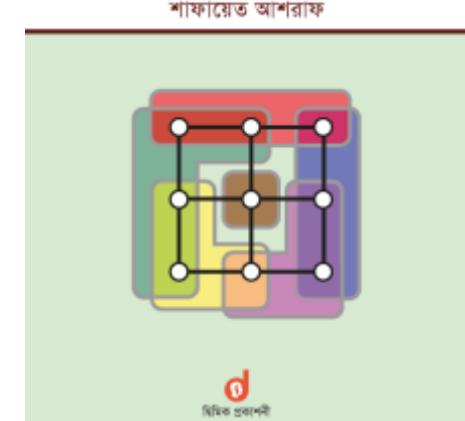


Secured by OneAll Social Login

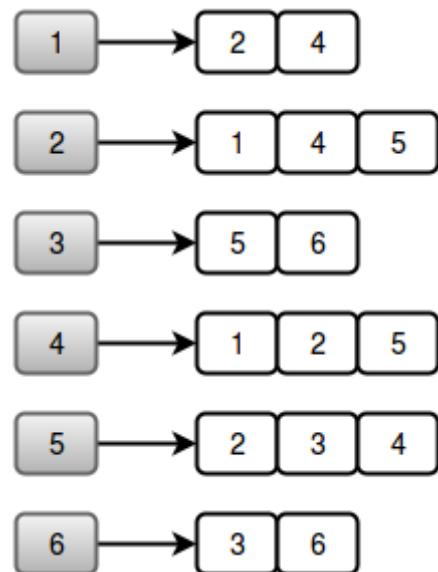
আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

Traveloka Singapore (বিস্তারিত...)



এবার বাজাৰ কৰাৰ লিস্টেৰ মত একটি লিস্ট বানাই:



এটাই অ্যাডজেসেন্সি লিস্ট, কোন নোডের সাথে কোন নোড যুক্ত আছে সেটাৱ একটা তালিকা। কিন্তু কোড কৰাৰ সময় কিভাবে এই লিস্টটা স্টোৱ কৰবো?

### প্ৰথম উপায়(অ্যারে):

সাধাৰণ ২ড়ি অ্যারে ব্যবহাৰ কৰে লিস্টটি স্টোৱ কৰা যায়। যেমন:

```
arr[1][1]=2, arr[1][2]=4;
```

```
arr[2][1]=1; arr[2][2]=4, arr[2][3]=5;
```

কিন্তু এভাৱে স্টোৱ কৰলে কিছু সমস্যা আছে:



প্ৰোগ্ৰামিং কনটেন্ট এবং অ্যালগোৰিদম

অনুপ্ৰৱণ(৪):

কেন আমি প্ৰোগ্ৰামিং শিখবো?

কম্পিউটাৱ বিজ্ঞান কেন পড়বো?

প্ৰোগ্ৰামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্ৰোগ্ৰামাৰ

অ্যালগোৰিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্ৰবলেম

বাইনাৰি সাৰ্চ - ১

বাইনাৰি সাৰ্চ - ২(বাইসেকশন)

ফ্ৰয়েড সাইকেল ফাইভিং অ্যালগোৰিদম

ডাটা স্টোকচাৰ(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলাৰ কিউ

সমস্যা ১. আমাদের ৬টি নোড আছে। প্রতি নোডের সাথে সর্বোক্ষ ৬টি নোড যুক্ত থাকতে পারে(ধরে নিচ্ছি দুটি নোডের মধ্যে ১টির বেশি সংযোগ থাকবেনা)। এ ক্ষেত্রে আমাদের লাগবে [6][6]আকারের ইন্টিজার অ্যারে। যদি ১ নম্বর নোডের সাথে মাত্র ২টি নোড যুক্ত থাকে তাহলে বাকি array[1][0],array[1][1] কাজে লাগবে,array[1][2] থেকে array[1][6] পর্যন্ত জায়গা কোনো কাজেই লাগবেনা। মনে হতে পারে এ আর এমন কি সমস্যা। কিন্তু চিন্তা কর ১০০০০ টি নোড আছে এমন একটি গ্রাফের কথা। [10000][10000] integer অ্যারে তুমি ব্যবহার করতে পারবেনা,memory limit অতিক্রম করে যাবে,এছাড়া এভাবে মেমরি অপচয় করা ভালো প্রোগ্রামারের লক্ষণ নয় :)। অ্যাডজেসেন্সি ম্যাট্রিক্স ব্যবহার করার সময় যেমন মেমরির সমস্যা হয়েছিলো, এখনও সেই সমস্যা রয়ে যাবে।

সমস্যা ২. অ্যারের কোন ইনডেক্সে কয়টি এলিমেন্ট আছে তার হিসাব রাখতে প্রতি ইনডেক্সের জন্য আরেকটি ভ্যারিয়েবল মেইনটেইন করতে হবে।

### ঞিতীয় উপায়(ডেষ্টের):

সমস্যাগুলো দূর করতে আমরা STL vector বা list ব্যবহার করে গ্রাফ স্টোর করব। ডেষ্টের/লিস্টে তোমাকে লিস্টের সাইজ বলে দিতে হবেনা, খালি সর্বোক্ষ নোড সংখ্যা বলে দিলেই হবে। এই টিউটোরিয়ালে আমি ডেষ্টের ব্যবহার করব কারণ list এ বেশ কিছু সমস্যা আছে।

১০০০০০ নোডের গ্রাফ ইনপুট দেয়ার সময় কখনো ম্যাট্রিক্স হিসাবে দিবেনা, তাহলে ইনপুটের আকারই মাত্রাতিরিক্ত বিশাল হয়ে যাবে। আগের পোস্টে ২য় উদাহরণে যেভাবে দেখিয়েছি সেরকম ইনপুট দিতে পারে, অর্থাৎ প্রথমে নোড আর এজ সংখ্যা বলে দিয়ে তারপর কোন নোডের সাথে কে যুক্ত আছে বলে দিবে। উপরের গ্রাফের জন্য ইনপুট:

```
“ 6 8 //node-edge
      1 2 //node1-node2
      1 4
      2 4
      2 5
      4 5
```

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)
ডিসজয়েট সেট(ইউনিয়ন ফাইন্ড)
ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)
সেগমেন্ট ট্রি-১
সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)
অ্যারে কমপ্রেশন/ম্যাপিং
লোয়েস্ট কমন অ্যানসেক্টর
বাইনারি ইনডেক্সড ট্রি
স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:
এল-আর-ইউ ক্যাশ(**নতুন**)
বুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):
গ্রাফ থিওরিতে হাতেখড়ি
অ্যাডজেসেন্সি ম্যাট্রিক্স
অ্যাডজেসেন্সি লিস্ট
ব্রেথড ফার্স্ট সার্চ (বিএফএস)
মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)
মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)
টপোলজিকাল সর্ট
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল
সর্ট
ডায়াক্স্ট্রা
ফ্লয়েড ওয়ার্শল
বেলম্যান ফোর্ড

5 3

3 6

6 6

এটি ডেস্টের দিয়ে ইনপুট নির এভাবে:

```
1 #include <cstdio>
2 #include <vector>
3 using namespace std;
4
5 #define MAX 100000 //maximum node
6 vector <int> edges[MAX];
7 vector <int> cost[MAX]; //parallel vector to store costs;
8
9 int main() {
10     int numNodes, numEdges;
11
12     scanf("%d%d", &numNodes, &numEdges);
13     for (int i = 1; i <= numEdges; i++) {
14         int x, y;
15         scanf("%d%d", &x, &y);
16         edges[x].push_back(y);
17         edges[y].push_back(x);
18         cost[x].push_back(1);
19         cost[y].push_back(1);
20     }
21     return 0;
22 }
```

cost নামক ডেস্টেরটি এ গ্রাফের ক্ষেত্রে দরকার ছিলনা, তবে ওয়েটেড গ্রাফে অবশ্যই দরকার হবে। নিচ্যই বুঝতে পারছ edge ও cost ডেস্টের দুটি সমান্তরাল ভাবে কাজ করবে, অর্থাৎ edge ডেস্টের যে পজিশনে তুমি দুটি নির্দিষ্ট নোডের কানেকশন পাবে cost ডেস্টের সেই পজিশনেই তুমি cost পাবে।

যদি 1000 বা তার কম মোড থাকে তাহলে ম্যাট্রিক্স বা লিস্ট কোনো ক্ষেত্রেই মেমরি সমস্যা হবেনা। তাও আমরা ডেস্টের দিয়েই গ্রাফ স্টোর করব।  
কারণ, চিত্ত কর তোমাকে 100টা নোডের ম্যাট্রিক্সে 1 এর সাথে কি কি সংযুক্ত আছে বের করতে matrix[1][0],matrix[1][1].....matrix[1]

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন

সার্সিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১  
ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সারসেট সাম,  
কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

[99] এভাবে ১০০টি পজিশন চেক করে কোনটায় কোনটায় ০ নেই বের করতে হবে, ১ নম্বর নোডের সাথে যতটি নোডই সংযুক্ত থাকুকনা কেন। তাই এখনেও অ্যারে আমাদের বাড়তি সুবিধা দিতে পারছেন।

একটা নোডের সাথে কোন কোন নোড যুক্ত আছে বের করা:

ধরো তুমি ১ নম্বর নোডের সাথে যুক্ত সবগুলো নোডের নম্বর চাও, তুমি তাহলে edges[1] এর সাইজ পর্যন্ত লুপ চালাবে এভাবে:

```
1 size=edges[1].size();
2 for(int i=0; i < size ; i++)
3 printf("%d ",edges[1][i]);
```

১>> ০ ১ ০ ০ ০ ১ ১ ০ পুরোটা ঘুরে আসার থেকে ১>> ২, ৬, ৭ ঘুরে আসতে কম সময় লাগবে, তাই নয়কি? 😊।

অ্যাডজেসেন্সি ম্যাট্রিক্স কখন লিস্ট অপেক্ষা সুবিধাজনক?

যদি কোনো প্রবলেমে তোমার  $u, v$  নোডের এর মধ্যে কোনো এজ আছে নাকি চেক করতে বলে তখন লিস্ট ব্যবহার করলে তোমাকে লুপ চালিয়ে চেক করতে হবে, কিন্তু ম্যাট্রিক্সে জাস্ট matrix[u][v] ইনডেক্স চেক করেই বলে দিতে পারবে তাদের মধ্যে কানেকশন আছে নাকি।

এক্সারসাইজ:

এ পর্যন্ত বুঝে থাকলে তুমি মোটামুটি bfs,dfs এর মত বেসিক অ্যালগোরিদম গুলো শেখাব জন্য প্রস্তুত। পরবর্তি লেখাটি পড়ার আগে একটি ছোট exercise করে ফেল। এমন একটি কোড লিখ যেটায় উপরের মত করে ইনপুট দিলে নিচের কাজগুলো করে:

১. একটি adjacency list তৈরি করে। (গ্রাফটিকে directed ধরে নিবে, bidirectional নয়)
২. কোন নোডের সাথে কয়টা নোড যুক্ত আছে, নোডগুলো কি কি সেগুলো প্রিন্ট করে।
৩. indegree হলো একটি নোডে কয়টি নোড প্রবেশ করেছে, outdegree হলো ঠিক তার উল্টোটা। প্রতিটি নোডের outdegree ও indegree প্রিন্ট কর।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেক্ষেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

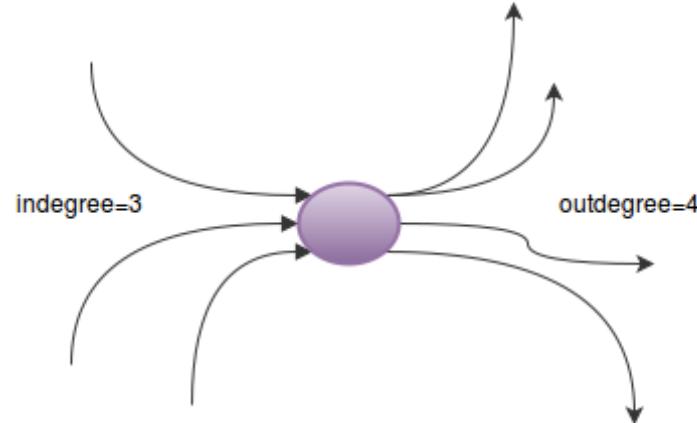
অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



পর-৮, বিএফএস: <http://www.shafaetsplanet.com/planetcoding/?p=604>

## ফেসবুকে মন্তব্য

0 comments

**0 Comments**

Sort by **Oldest**



Add a comment...

Facebook Comments plugin

Powered by [Facebook Comments](#)



কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

**Shafaetsplanet**  
2.4K likes  
**শাফায়েত আশরাফ**

[Like Page](#)

Be the first of your friends to like this



ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্বিনেটরিক্সের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / ম্যাচ

# গ্রাফ থিওরিতে হাতেখড়ি-৪(ব্রেডথ ফার্স্ট সার্চ)

ফেব্রুয়ারি ২২, ২০১৮ by শাফায়েত



আগের [পর্বগুলোতে](#) আমরা দেখেছি কিভাবে ম্যাট্রিক্স বা লিস্ট ব্যবহার করে গ্রাফ স্টোর করতে হয়। এবার আমরা প্রথম অ্যালগরিদম দেখবো এব দিকে যাবো। শুরুতেই আমরা যে অ্যালগরিদমটা শিখব তার নাম ব্রেডথ ফার্স্ট সার্চ(breadth first search,bfs)।

বিএফএস এব কাজ হলো গ্রাফে একটা নোড থেকে আরেকটা নোড যাবার শর্টেস্ট পাথ বের করা। বিএফএস কাজ করবে শুধুমাত্র আন-ওয়েটেড গ্রাফের ক্ষেত্রে, তারমানে সবগুলো এজের কষ্ট হবে ১।

বিএফএস অ্যালগরিদমটা কাজ করে নিচের ধারণাগুলোর উপর ভিত্তি করে:

- “ ১. কোনো নোডে ১ বারের বেশি যাওয়া যাবেনা
- ২. সোর্স নোড অর্থাৎ যে নোড থেকে শুরু করছি সেটা ০ নম্বর লেভেলে অবস্থিত।
- ৩. সোর্স বা ‘লেভেল ০’ নোড থেকে সরাসরি যেসব নোডে যাওয়া যায় তারা সবাই ‘লেভেল ১’ নোড।
- ৪. ‘লেভেল ১’ নোডগুলো থেকে সরাসরি যেসব নোডে যাওয়া যায় তারা সবাই ‘লেভেল ২’ নোড।  
এভাবে লেভেল এক এক করে বাড়তে থাকবে।
- ৫. যে নোড যত নম্বর লেভেলে, সোর্স থেকে তার শর্টেস্ট পথের দৈর্ঘ্য তত।

## সাবস্ক্রাইব

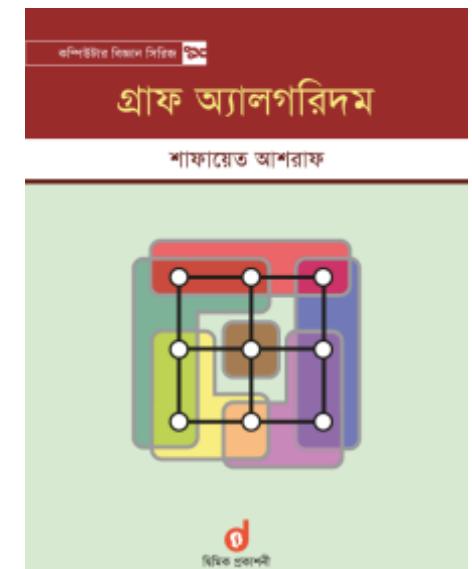


Secured by [OneAll Social Login](#)

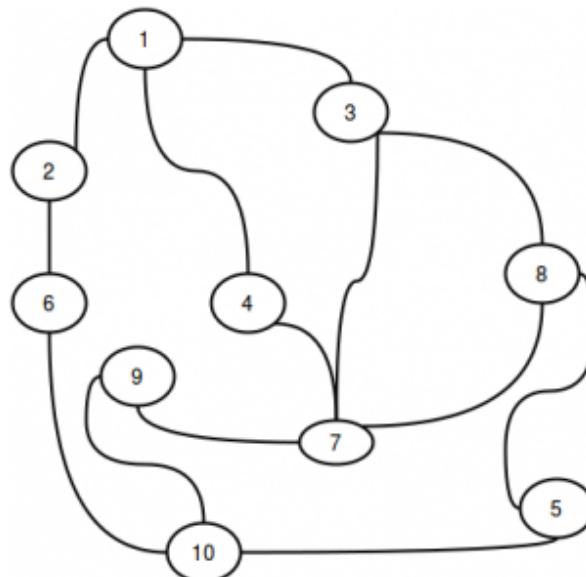
## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

**Traveloka Singapore** (বিস্তারিত...)



উপরে লেখাগুলো পুরোপুরি না বুঝলে আমরা একটা উদাহরণ দেখে বাকিটা পরিষ্কার করব।



ধর তুমি ১ নম্বর শহর থেকে ১০ নম্বর শহরে যেতে চাও। প্রথমে আমরা সোর্স ধরলাম ১ নম্বর নোডকে। ১ তাহলে একটা 'লেভেল ০' নোড। ১ কে ডিজিটেড চিহ্নিত করি।

১ থেকে সরাসরি যাওয়া যায় ২,৩,৪ নম্বর নোডে। তাহলে ২,৩,৪ হলো 'লেভেল ১' নোড। এবার সেগুলোকে আমরা ডিজিটেড চিহ্নিত করি এবং সেগুলো নিয়ে কাজ করি। নিচের ছবি দেখ:



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ন্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

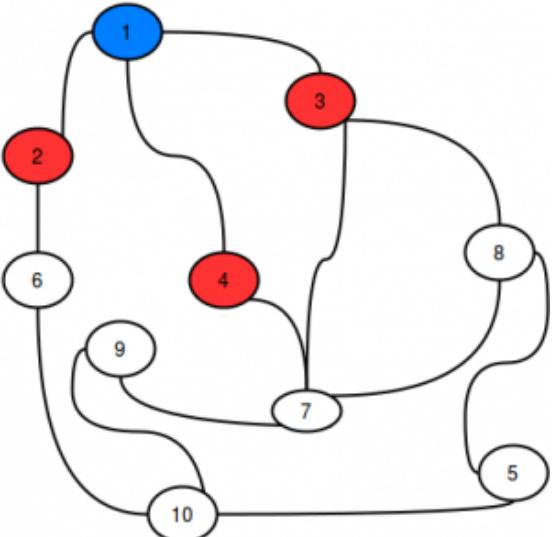
ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

ডাটা স্ট্রাকচাৰ(১৪):

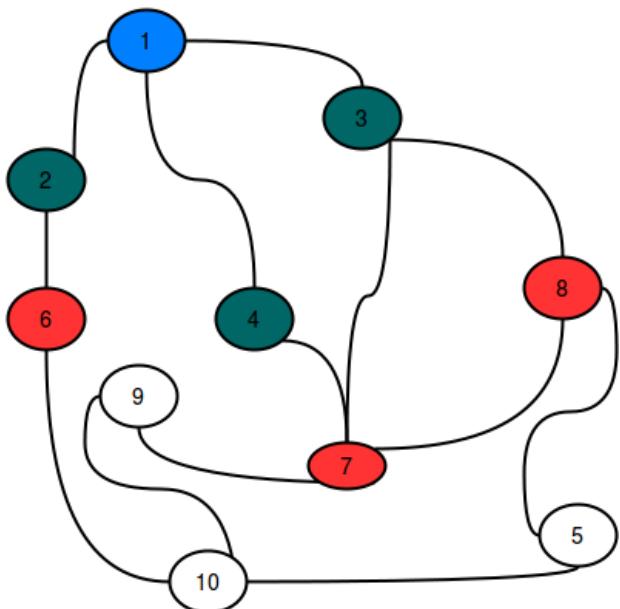
লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলাৰ কিউ



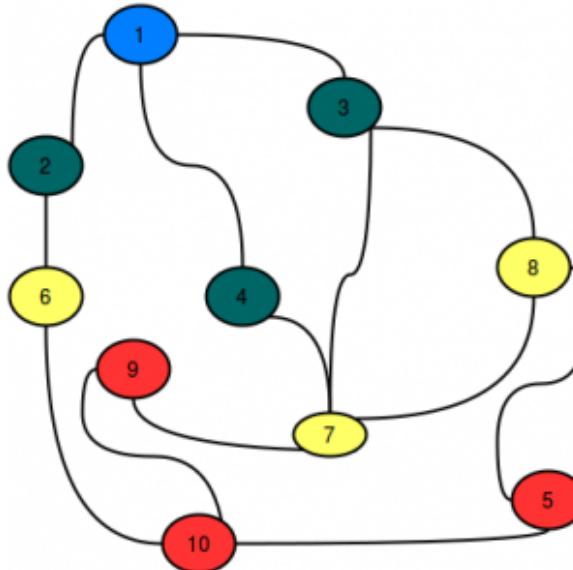
লাল নোডগুলো নিয়ে আমরা এখন কাজ করবো। রঙিন সবগুলো নোড ডিজিটেড, এক নোডে ২বার কখনো যাবোনা। ২,৩,৪ থেকে শর্টেস্ট পথে যাওয়া যায় ৬,৭,৮ এ। সেগুলো ডিজিটেড চিহ্নিত করি:



শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)  
 ডিসজয়েট সেট(ইউনিয়ন ফাইল)  
 ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)  
 সেগমেন্ট ট্রি-১  
 সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)  
 অ্যারে কমপ্রেশন/ম্যাপিং  
 লোয়েস্ট কমন অ্যানসেস্টর  
 বাইনারি ইনডেক্সড ট্রি  
 স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:  
 এল-আর-ইউ ক্যাশ(**নতুন**)  
 ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):  
 গ্রাফ থিওরিতে হাতেখড়ি  
 অ্যাডজেসেন্সি ম্যাট্রিক্স  
 অ্যাডজেসেন্সি লিস্ট  
 ব্রেথড ফার্স্ট সার্চ (বিএফএস)  
 মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)  
 মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)  
 টপোলজিকাল সর্ট  
 ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
 সর্ট  
 ডায়াক্রস্ট্রা  
 ফ্লয়েড ওয়ার্শল  
 বেলম্যান ফোর্ড

লক্ষ কর যে নোডকে যত নম্বর লেভেলে পাঞ্চিসোর্স থেকে তার শর্টেস্ট পথের দৈর্ঘ্য ঠিক তত। যেমন ২নম্বর লেভেলে ৮কে পেয়েছি তাই ৮ এর দুরত্ব ২। ছবিগুলোকে একেকটা লেভেলের একেক রং দিয়ে হয়েছে। আর লাল নোড দিয়ে বুঝানো হয়েছে আমরা এখন ওগুলো নিয়ে কাজ করছি। আমরা ১০ এ পৌছাইনি তাই পরের নোডগুলো ডিজিট করে ফেলি:



আমরা দেখতে পাচ্ছি ২টি লেভেল পার হয়ে ৩ নম্বর লেভেলে আমরা ১০ কে পাঞ্চি। তাহলে ১০ এর শর্টেস্ট পথ ৩। লেভেল বাই লেভেল গ্রাফটাকে সার্চ করে আমরা শর্টেস্ট পথ বের করলাম। যেসব এজ গুলো আমরা ব্যবহার করিনি সেগুলোকে বাদ দিয়ে ছবিটিকে নিচের মত করে আকতে পাবি:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

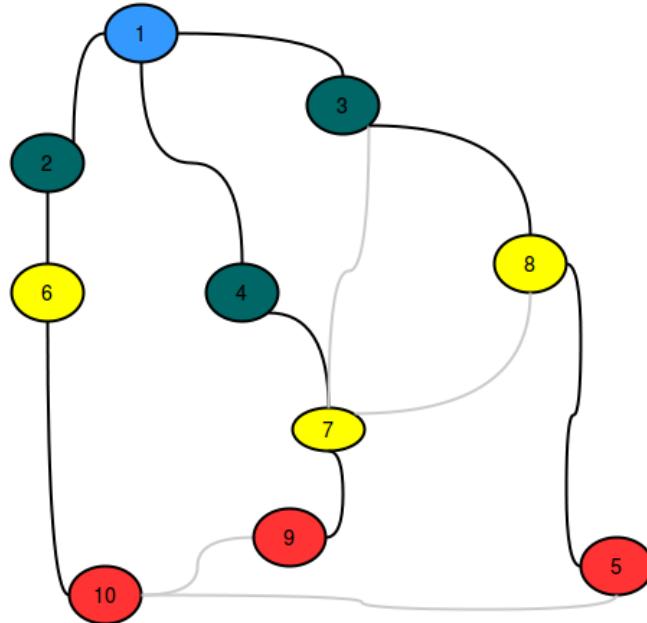
ডাইনামিক প্রোগ্রামিং ২ (শর্টেস্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)



যেসব এজ ব্যবহার করিনি সেগুলো হালকা করে দিয়েছি, এই এজ গুলো বাদ দিলে গ্রাফটি একটি ট্রি হয়ে যায়। এই ট্রি টাকে বলা হয় বিএফএস ট্রি।

তারমানে আমাদের কাজ গুলো সোর্স থেকে লেভেল ১ নোডগুলোতে যাওয়া, তারপর লেভেল ১ এর নোডগুলো থেকে লেভেল ২ নোডগুলো খুজে বের করা, এভাবে যতক্ষণ না গন্তব্যে পৌছে যাচ্ছি অথবা সব নোড ডিজিট করা শেষ হয়ে গিয়েছে ততক্ষণ কাজ চলতে থাকবে।

কিউ ডাটা স্ট্রাকচারটার সাথে আশা করি সবাই পরিচিত। কিউ হলো হ্বৃহ বাসের লাইনের মতো ডাটা স্ট্রাকচার। যখন একটা সংখ্যা কিউতে যোগ করা হয় তখন সেটা আগের সবগুলো সংখ্যার পিছে গিয়ে দাঢ়ায়, যখন কোন একটা সংখ্যা বের করে ফেলা হয় তখন সবার প্রথমের সংখ্যাটা নেয়া হয়। একে বলা ফার্স্ট ইন ফার্স্ট আউট। আমরা বিএফএস এ কিউ কাজে লাগাতে পারি। লেভেল ১ থেকে যখন কয়েকটা নতুন লেভেল ২ নোড পাবো সেগুলোকে কিউতে বা লাইনে অপেক্ষা করিয়ে রাখবো, আব সবসময় প্রথম নোডটা নিয়ে কাজ করবো। তাহলে বড় লেভেলের নোডগুলো সবসময় পিছের দিকে থাকবে, আমরা ছোট লেভেলগুলো নিয়ে কাজ করতে করতে আগবো। উপরের গ্রাফের জন্য এটা আমরা সিমুলেট করে দেখি:

প্রথমে কিউতে সোর্স পুশ করবো:

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্স(১):

ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

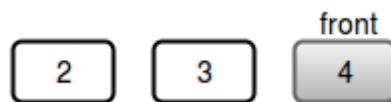
কোয়ান্টাম কম্পিউটার(২)



১ এর লেডেল হবে ০ বা লেডেল[১] = ০। এবার বিএফএস শুরু করবো।

প্রথমে কিউ এর সবার সামনের নোডটাকে নিয়ে কাজ করবো। সবার সামনে আছে ১, সেখান থেকে যাওয়া যায় ৪, ৩, ২ এ। ৪, ৩, ২ এ এসেছি ১ থেকে, তাহলে লেডেল[৪] = লেডেল[১] + ১ = ১, লেডেল[৩] = লেডেল[১] + ১ = ১, লেডেল[২] = লেডেল[১] + ১ = ১।

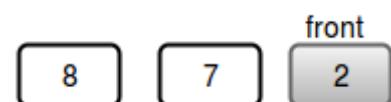
১ কে ফেল দিয়ে এদেরকে কিউতে পুশ করে রাখি:



এবার ৪ নিয়ে কাজ করি। ৪ থেকে যাওয়া যায় ৭ এ। তাহলে আমরা বলতে পারি লেডেল[৭]=লেডেল[৪]+১=২। ৪ কে ফেলে দিয়ে ৭ কে কিউতে পুশ করি:



৩ থেকে ৭,৮ এ যাওয়া যায়। ৭ কে এরই মধ্যে নিয়েছি, শুধু ৮ পুশ করতে হবে। লেডেল[৮]=লেডেল[৩]+১=২।



এভাবে যতক্ষণনা কিউ খালি হচ্ছে ততক্ষণ কাজ চলতে থাকবে। লেডেল[] অ্যারের মধ্যে আমরা পেয়ে যাবো সোর্স থেকে সবগুলো নোডের দূরস্থ!

সুড়োকোড়:

```
1 procedure BFS(G,source):
```

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet
  
2.4K likes

## ফাফ অ্যালগরিদম

শাফায়েত আশরাফ

[Like Page](#)

Be the first of your friends to like this

---

**Shafaetsplanet**
about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্বিনেটরিক্সের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

---

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

[SHAFAEETPLANET.COM](#)  
[ডাইনামিক প্রোগ্রামিং ৬ সেমার](#)

```

2 Q=queue(), level[] =infinity
3 Q.enqueue(source)
4 level[source]=0
5 while Q is not empty
6     u ← Q.pop()
7     for all edges from u to v in G.adjacentEdges(v) do
8         if level[v] = infinity:
9             level[v] = level[u] + 1;
10            Q.enqueue(v)
11        end if
12    end for
13 end while
14. Return distance;

```

ঠিক যেভাবে সিমুলেট করেছি সেভাবেই কোডটা লিখেছি, আপা কবি বুঝতে সমস্যা হচ্ছেন।

শুধু পাথের দৈর্ঘ্য যথেষ্ট না, পাথটাও দরকার হতে পারে। লক্ষ্য করো আমরা  $u$  থেকে  $v$  তে যাবার সময়  $parent[v] = u$  করে দিচ্ছি। আমরা প্রতিটা নোডের জন্য জানি কোন নোড থেকে সেই নোডে এসেছি। তাহলে আমরা যে নোডের জন্য পাথ বের করতে চাই সেই নোড থেকে তার প্যারেন্ট নোডে যেতে থাকবো যতক্ষণনা সোর্সে পৌছে যাই। খুবই সহজ কাজ, পাথ বের করার কোড করা তোমার উপর ছেড়ে দিলাম।

### কমপ্লেক্সিটি:

প্রতিটা নোডে একবার করে গিয়েছি, প্রতিটা এজ এ একবার গিয়েছি। তাহলে **কমপ্লেক্সিটি** হবে  $O(V + E)$  যেখানে  $V$  হলো নোড সংখ্যা এবং  $E$  হলো এজ সংখ্যা।

কখনো কখনো ২-ডি গ্রিডে বিএফএস চালানো লাগতে পারে। যেমন একটা দাবার বোর্ডে একটি যোড়া আব একটা রাজা আছে। যোড়টা মিনিমাম কয়টা মুড়ে রাজার ঘরে পৌছাতে পারবে? অথবা একটা ২-ডি অ্যারেতে কিছু সেল ব্রক করে দেয়া হয়েছে, এখন কোনো সেল থেকে আরেকটি সেলে মিনিমাম মুড়ে পৌছাতে হবে, প্রতি মুড়ে শুধুমাত্র সামনে-পিছে-বামে-ডানে যাওয়া যায়। আগে নোডকে আমরা প্রকাশ করছিলাম একটা মাত্র সংখ্যা দিয়ে, এখন নোডকে প্রকাশ করতে হবে দুটি সংখ্যা দিয়ে, রো(row) নাম্বার, এবং কলাম নাম্বার। তাহলে আমরা নোড রিপ্রেজেন্ট করার জন্য সি তে একটা স্ট্রাকচার বানিয়ে নিতে পারি এরকম:

```
“ struct node{int r,c;};
```

অথবা আমরা সি++ এর “পেয়ার” ব্যবহার করতে পারি।

pair<int,int>

“

এ ক্ষেত্রে ডিজিটেড, প্যারেণ্ট, লেভেল অ্যারেগুলো হবে ২ ডিমেনশনের, যেমন  $visited[10][10]$  ইত্যাদি। কিউতে নোডের বদলে স্ট্রাকচার পুশ করবো। আর কোন একটা ঘর থেকে অন্য ঘরে যাবার সময় চেক করতে হবে বোর্ডের বাইরে চলে যাচ্ছে কিনা। একটা স্যাম্পল সি++ কোড দেখি:

```
1 #define pii pair<int,int>
2 int fx[]={1,-1,0,0}; //ডিরেকশন অ্যারে
3 int fy[]={0,0,1,-1};
4 int cell[100][100]; //cell[x][y] যদি -> হ্য তাহলে সেলটা ঋক
5 int d[100][100],vis[100][100]; //d means destination from source.
6 int row,col;
7 void bfs(int sx,int sy) //Source node is in [sx][sy] cell.
8 {
9     memset(vis,0,sizeof vis);
10    vis[sx][sy]=1;
11    queue<pii>q; //A queue containing STL pairs
12    q.push(pii(sx,sy));
13    while(!q.empty())
14    {
15        pii top=q.front(); q.pop();
16        for(int k=0;k<4;k++)
17        {
18            int tx=top.uu+fx[k];
19            int ty=top.vv+fy[k]; //Neighbor cell [tx][ty]
20            if(tx>=0 and tx<row and ty>=0 and ty<col and cell[tx][ty]!=-1 and vis[tx][ty]==0) //Check
21            {
22                vis[tx][ty]=1;
23                d[tx][ty]=d[top.uu][top.vv]+1;
24                q.push(pii(tx,ty)); //Pushing a new pair in the queue
25            }
26        }
27    }
28 }
```

তুমি যদি ডিরেকশন অ্যারের ব্যাপারটা না বুঝো তাহলে [এই লেখাটা](#) পড়লে আরো কিছু ডিটেইলস জানতে পারবে।

বিএফএস শুধুমাত্র আন-ওয়েটেড গ্রাফে কাজ করে, ওয়েটেড গ্রাফে শর্টেস্ট পাথ বের করতে **ডায়ালগ্রাম** অ্যালগোরিদম ব্যবহার করতে পারো। গ্রাফে  
নেগেটিভ সাইকেল থাকলে **বেলম্যান ফোর্ড** ব্যবহার করতে হবে।

প্র্যাকচিসের জন্য প্রবলেম:

**Bicoloring**(Bipartite checking)

**A Node Too Far**(Shortest path)

**Risk**(Shortest path)

**Bombs! NO they are Mines!!**(bfs in 2d grid)

**Knight Moves**(bfs in 2d grid)

**We Ship Cheap**(Printing path)

**Word Transformation**(strings)

পরের পর

ফেসবুকে মন্তব্য

4 comments

# গ্রাফ থিওরিতে হাতেখড়ি ৫: মিনিমাম স্প্যানিং ট্ৰি(প্ৰিম অ্যালগোৱিদম)

আগস্ট ৪, ২০১১ by শাফায়েত



(সিরিজের অন্যান্য পোস্ট)

একটি গ্রাফ থেকে কয়েকটি নোড আৰ এজ নিয়ে নতুন একটি গ্রাফ তৈৰি কৰা হলে সেটাকে বলা হয় সাবগ্রাফ। স্প্যানিং ট্ৰি হলো এমন একটি সাবগ্রাফ যেটায়:

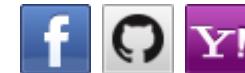
- “ * মূল গ্রাফের সবগুলো নোড আছে।  
* সাবগ্রাফটি একটি ট্ৰি। ট্ৰিটে কখনো সাইকেল থাকেনা, এজ থাকে  $n - 1$  টি যেখানে  $n$  হলো নোড  
সংখ্যা।

একটি গ্রাফের অনেকগুলো স্প্যানিং ট্ৰি থাকতে পাৰে, যে ট্ৰি এৰ এজ গুলোৱ কস্ট/ওয়েট এৰ যোগফল সব থেকে কম সেটাই মিনিমাম স্প্যানিং ট্ৰি। আমৰা এই লেখায় প্ৰিম অ্যালগোৱিদমেৰ সাহায্যে মিনিমাম স্প্যানিং ট্ৰি বেৰ কৰা শিখবো।

মনে কৰি নিচেৰ গ্রাফেৰ প্ৰতিটি নোড হলো একটি কৰে বাড়ি। আমাদেৱ বাড়িগুলোৱ মধ্যে টেলিফোন লাইন বসাতে হবে। আমৰা চাই সবথেকে কম খৰচে লাইন বসাতে। এজ গুলোৱ ওয়েট লাইন বসানোৱ খৰচ নিৰ্দেশ কৰে:



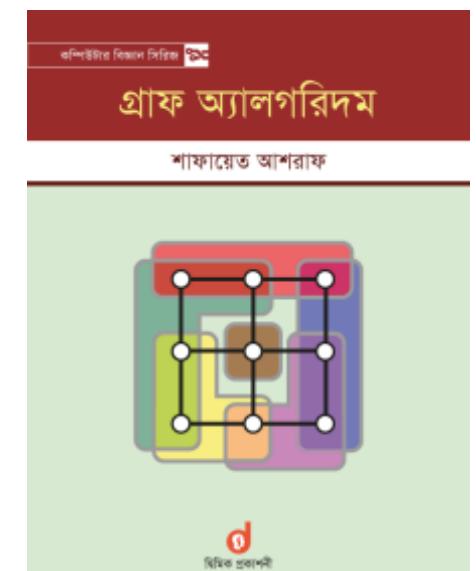
সাৰক্ষাইৰ



Secured by OneAll Social Login

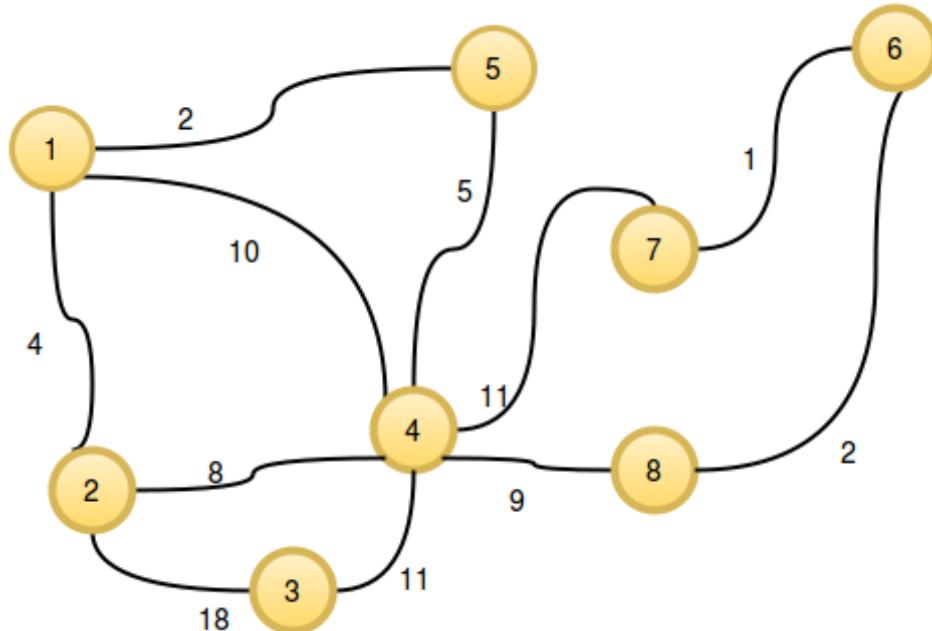
আমাৰ সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়াৰ @  
**Traveloka Singapore** (বিস্তাৰিত...)



আমরা অনেক ভাবে লাইন বসাতে পারতাম। ছবিতে লাল এজ দিয়ে টেলিফোন লাইন বসানোর একটি উপায় দেখানো হয়েছে। টেলিফোন লাইনগুলো একটি সাবগ্রাফ তৈরি করেছে যেটায় অবশ্যই  $n - 1$  টি এজ আছে, কোনো সাইকেল নেই কারণ অতিরিক্ত এজ বসালে আমাদের খবর বাড়বে, কোনো লাভ হবেনা। মিনিমাম স্প্যানিং ট্রি বের করার সময় আমরা এমন ভাবে এজগুলো নিবো যেন তাদের এজ এর যোগফল মিনিমাইজ হয়।

এখন নিচের গ্রাফ থেকে কিভাবে আমরা মিনিমাম স্প্যানিং ট্ৰি বেঁকুৱা কৈবল্য পাবো?



গ্রিডি(greedy) অ্যাপ্রোচে খুব সহজে মিনিমাম স্প্যানিং ট্রি বের করা যায়। আমরা এখন প্রিমস অ্যালগোরিদম কিভাবে কাজ করে দেখব। তুমি যদি আগে ক্রসকাল শিখতে চাও তাহলেও সমস্যা নেই, **সৱাসৱি পরের পরে** চলে যেতে পারো।

আমরা প্রথমে যেকোনো একটি সোর্স নোড নিব। ধরি সোর্স হলো ১। ১ থেকে যতগুলো এজ আছে সেগুলোর মিনিমাম টিকে আমরা সাবগ্রাফে যোগ করব। নিচের ছবিতে তীল এজ দিয়ে বুঝানো হচ্ছে এজটি সাবগ্রাফে যুক্ত করা হয়েছে:



## প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

### অনুপ্রেরণা(৪):

## কেন আমি প্রোগ্রামিং শিখবো?

## কম্পিউটার বিজ্ঞান কেন পড়বো?

## প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

ଶତେଖା

কনফিউজড প্রোগ্রামার

### অ্যালগরিদম বেসিক(৬):

## বিগ "Q" নোটেশন

কমপ্লেক্সিটি ন্যাস(P-NP, টেরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্ট -

বাইনারি সার্ট - ১

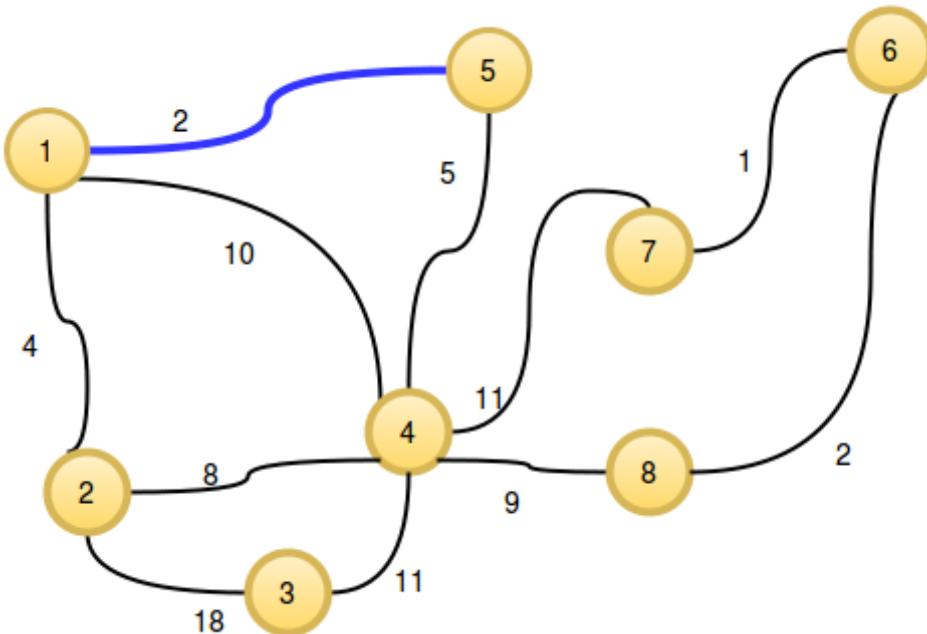
ফর্যেড সাইকেল ফাইলিং আয়োজন

### ডাটা স্টুকচার(১৪):

## ଲିଂକଡ ଲିସ୍ଟ

স্ট্যাক

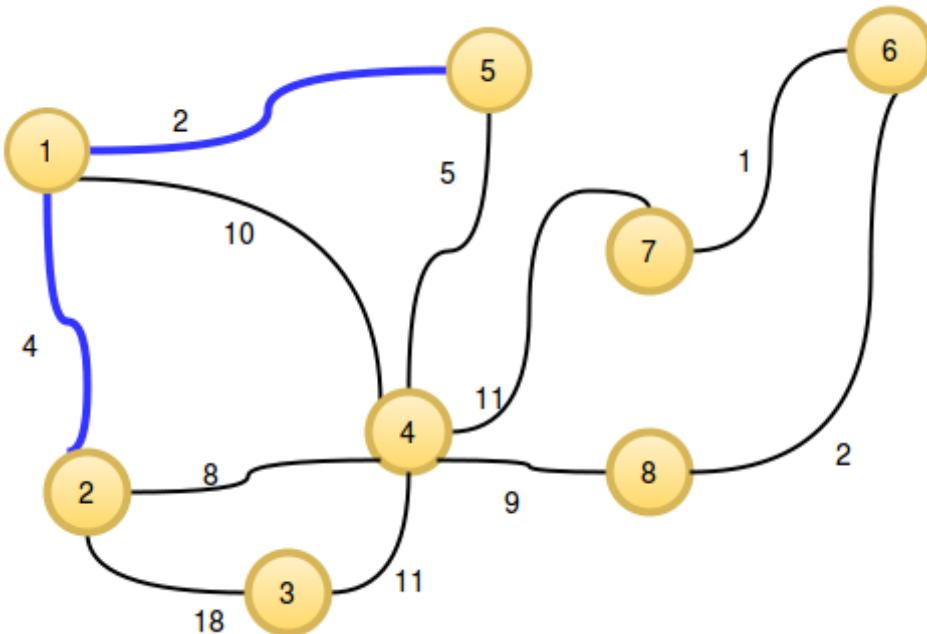
କିଉ+ସାର୍କୁଲାର କିଉ



এবাব সোর্স ১ এবং ৫ নম্বর নোড থেকে মোট যত এজ আছে(আগের এজগুলো সহ) তাদের মধ্যে মিনিমাম টি নির:

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)  
 ডিসজয়েট সেট(ইউনিয়ন ফাইল)  
 ট্রাই(প্রিফিল ট্রি/বেডিল ট্রি)  
 সেগমেন্ট ট্রি-১  
 সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)  
 অ্যারে কমপ্রেশন/ম্যাপিং  
 লোয়েস্ট কমন অ্যানসেস্টর  
 বাইনারি ইনডেক্সড ট্রি  
 স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:  
 এল-আর-ইউ ক্যাশ(**নতুন**)  
 ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):  
 গ্রাফ থিওরিতে হাতেখড়ি  
 অ্যাডজেসেন্সি ম্যাট্রিক্স  
 অ্যাডজেসেন্সি লিস্ট  
 ব্রেথড ফার্স্ট সার্চ (বিএফএস)  
 মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)  
 মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)  
 টপোলজিকাল সর্ট  
 ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
 সর্ট  
 ডায়াক্রস্ট্রা  
 ফ্লয়েড ওয়ার্শল  
 বেলম্যান ফোর্ড



এবাব নিব ১,২ এবং ৫ নম্বর নোড থেকে মোট যত এজ আছে(আগের এজগুলো সহ) তাদের মধ্যে মিনিমাম:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রিংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

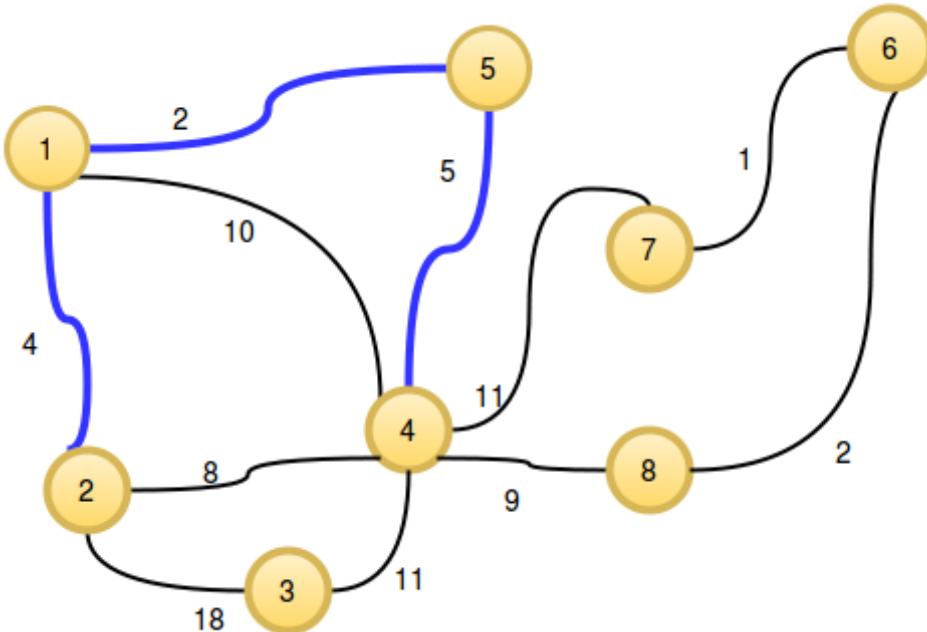
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)



পরের ধাপটি গুরুত্বপূর্ণ। ১,২,৫,৮ থেকে যত এজ আছে তাদের মধ্য মিনিমাম হলো ২-৪, কিন্তু ২ নম্বর নোড এবং ৪ নম্বর নোড দুইটাই অলরেডি সাবগ্রাফের অংশ, তারা আগে থেকেই কানেক্টেড, এদের যোগ করলে সাবগ্রাফে সাইকেল তৈরি হবে, তাই ২-৪ এজটি নিয়ে আমাদের কোনো লাভ হবেনা। আমরা এমন প্রতিবার এজ নিব যেন নতুন আরেকটি নোড সাবগ্রাফে যুক্ত হয়। তাহলে ৪-৮ হবে আমাদের পরের চয়েস।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্রিং(১):

ব্যকট্যাক্রিং বেসিক এবং পারমুটেশন জেনারেটর

নাম্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

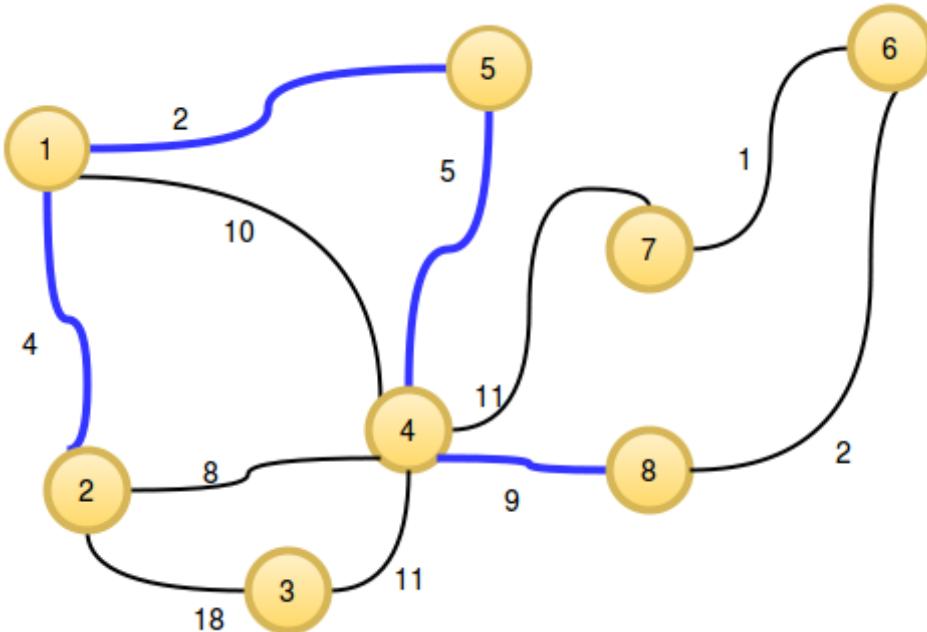
অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



এবপৰ ৮-৬ যোগ কৰবো:

কোয়ান্টাম কম্পিউটাৰ কী?  
কোয়ান্টাম কম্পিউটাৰেৰ শক্তি এবং সীমাবদ্ধতা

 **Shafaetsplanet**  
2.4K likes  
শাফায়েত আশরাফ

[Like Page](#)

Be the first of your friends to like this

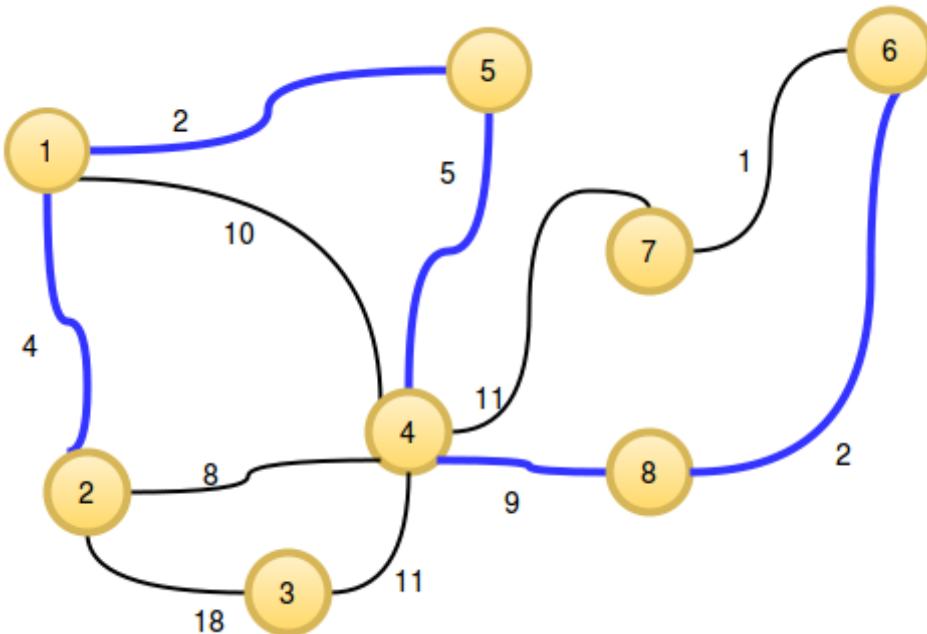
 **Shafaetsplanet**   
about an hour ago

ডাইনামিক প্ৰোগ্ৰামিং এৰ নতুন  
সিৱিজেৰ ৬ নম্বৰ লেখায় আমৰা  
দেখবো কম্পিউটাৰেৰ প্ৰবলেম এবং  
ডিসিশন প্ৰবলেম কিভাৱে সমাধান  
কৰতে হ্য।

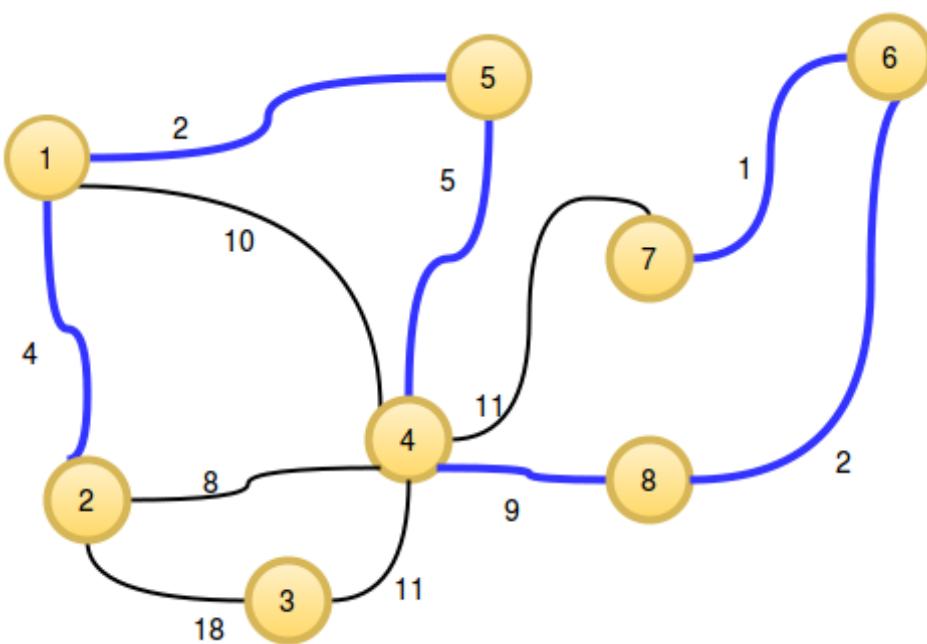
---

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

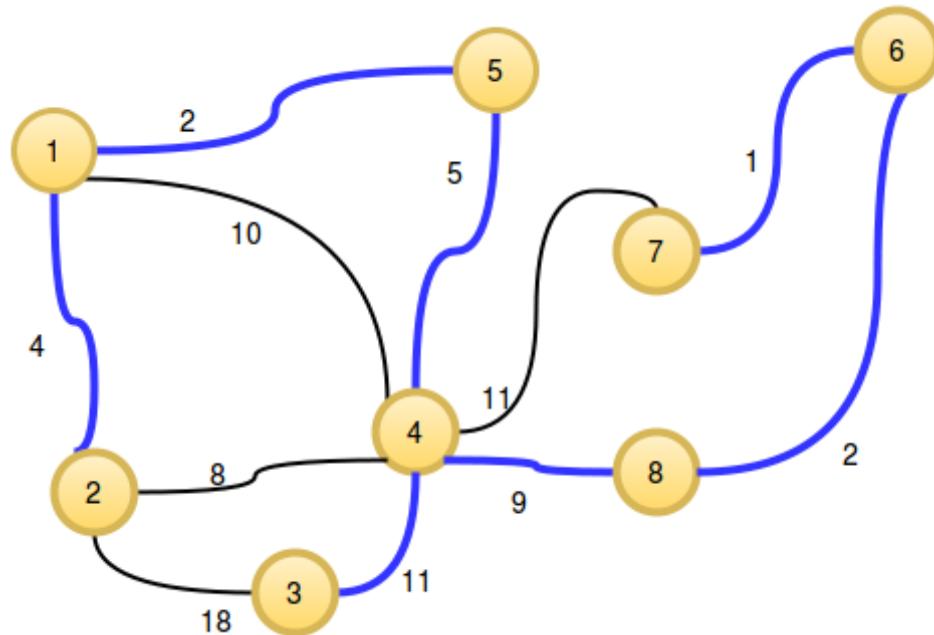
SHAFAEETPLANET.COM  
ডাইনামিক প্ৰোগ্ৰামিং ৬ / চোৱা



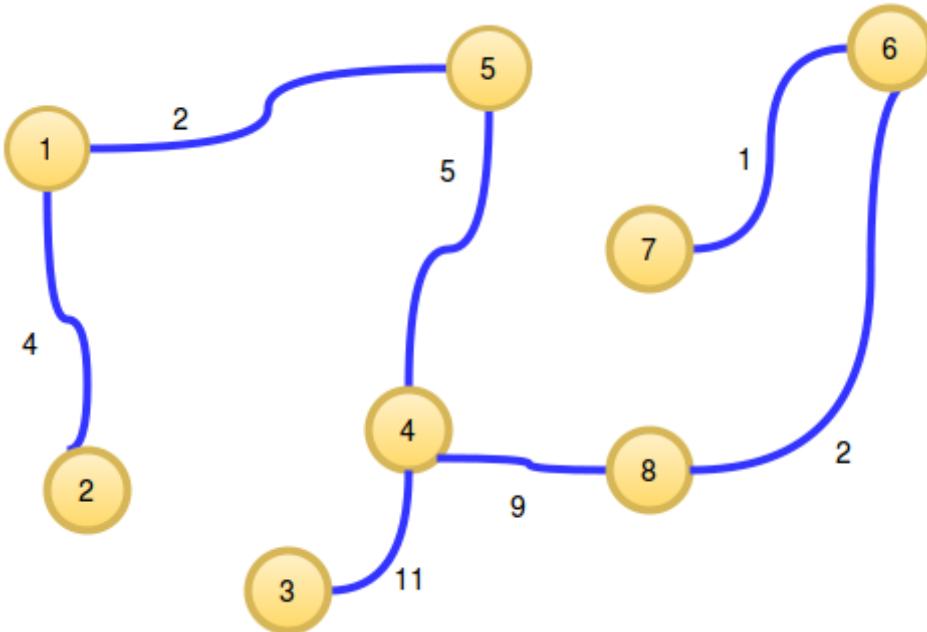
এবপৰ ৬-৭:



সবশেষ ৪-৩ যোগ করবো:



নীলৰং এব এই সাবগ্রাফটাই আমাদের মিনিমাম স্প্যানিং ট্ৰি। বাকি এজণ্ডলো মুছে দিলে থাকে:



তাহলে টেলিফোন লাইন বসাতো মোট খরচ:  $8 + 2 + 5 + 11 + 9 + 2 + 1 = 38$ । একটি গ্রাফে এক বা একাধিক মিনিমাম স্প্যানিং ট্ৰি থাকতে পারে।

আমাদের সুড়োকোড হবে এরকম:

```

1 * Input: A non-empty connected weighted graph with vertices V and edges E (the weights can be negative)
2 * Initialize: Vnew = {x}, where x is an arbitrary node (starting point) from V, Enew = {}
3 * Repeat until Vnew = V:
4 o Choose an edge (u, v) with minimal weight such that u is in Vnew and v is not
5 (if there are multiple edges with the same weight, any of them may be picked)
6 o Add v to Vnew, and (u, v) to Enew
7 * Output: Vnew and Enew describe a minimal spanning tree
  
```

এখন মাথায পশ্চ আসতে পারে কি ভাবে প্ৰিমস অ্যালগোৱিদম ইম্পৰ্মেন্ট কৰব? বাৰবাৰ লুপ চালিয়ে নেইড অ্যোপোচে কোড লিখলে তোমাৰ কোড টাইম লিমিটেৰ মধ্যে রান না কৰাৰ সম্ভাৱনাই বেশি।

রান্টাইম কমাতে প্রায়োরিটি কিউ ব্যবহার করতে পারো। যখন নতুন একটা নোড  $V_{\text{new}}$  তে যোগ করছো তখন সেই নোডের অ্যাডজেসেন্ট সবগুলো এজ প্রায়োরিটি কিউতে ঢুকিয়ে রাখতে হবে। এখন প্রায়োরিটি কিউ থেকে সবথেকে মিনিমাম ওয়েটের এজটা লগারিদম কমপ্লেক্সিটিতে খুজতে পারবে। মোট কমপ্লেক্সিটি হবে  $O(E \log E)$ । তবে এজের বদলে কিউতে নোড পুশ করে  $O(E \log V)$  তে কমপ্লেক্সিটি নামিয়ে আনা যায়, সেটা কিভাবে করা যায় চিন্তা করে বের করো।

অ্যালগোরিদমটা ইমপ্রিমেন্ট করার পর অবশ্যই নিচের সমস্যা গুলো সমাধানের চেষ্টা করবে।

<http://uva.onlinejudge.org/external/5/544.html>(Straight forward)

<http://uva.onlinejudge.org/external/9/908.html>

<http://uva.onlinejudge.org/external/100/10034.html>(Straight forward)

<http://uva.onlinejudge.org/external/112/11228.html>

<http://uva.onlinejudge.org/external/104/10462.html>(2nd best mst)

spoj:

<http://www.spoj.pl/problems/MST/>(Straight forward)

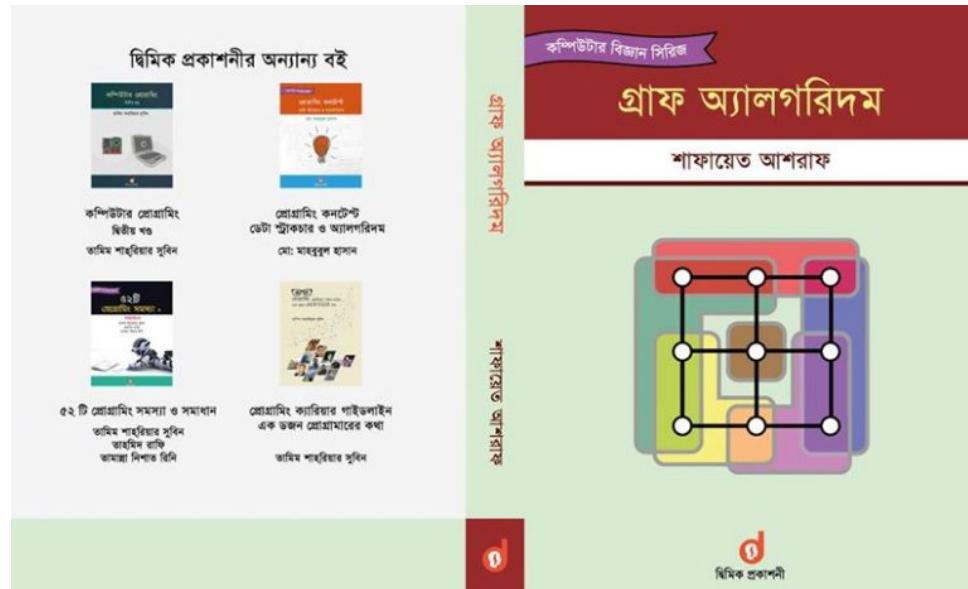
মিনিমাম স্প্যানিং ট্ৰি বের কৰার জন্য আৱেকটি অ্যালগোরিদম আছে যা ক্রুসকাল অ্যালগোরিদম নামে পৰিচিত। পৰের পৰে আমৰা সেটা শিখবো।

## ফেসবুকে মন্তব্য

2 comments

# গ্রাফ থিওরিতে হাতেখড়ি ৬: মিনিমাম স্প্যানিং ট্রি(ক্রুসকাল অ্যালগরিদম)

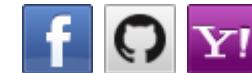
সেপ্টেম্বর ২৯, ২০১১ by শাফায়েত



(অন্যান্য পোস্ট)

আগের পোস্টে আমরা প্রিমস অ্যালগরিদম ব্যবহার করে **mst** নির্ণয় করা দেখেছি। mst কাকে বলে সেটাও আগের পোস্টে বলা হয়েছে। এ পোস্টে আমরা দেখবো mst বের করার আরেকটি অ্যালগরিদম যা ক্রুসকালের অ্যালগরিদম নামে পরিচিত। এটি mst বের করার সবথেকে সহজ অ্যালগরিদম। তবে তোমাকে অবশ্যই ডিসজয়েন্ট সেট ডাটা স্ট্রাকচার সম্পর্কে জানতে হবে, না জানলে [এই পোস্টটি](#) অবশ্যই দেখে আসো।

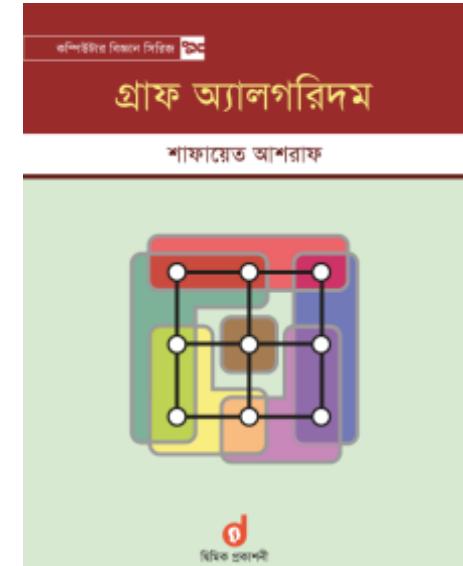
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

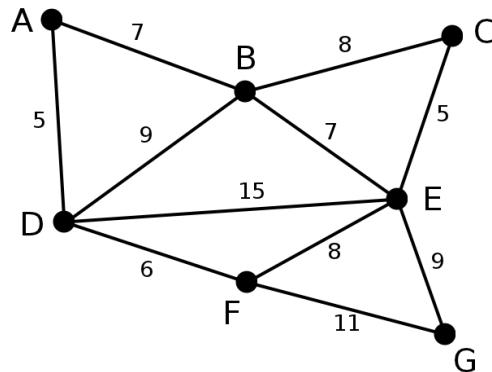
আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



এই পোস্টে নিজের আকা ছবি ব্যবহার করবোনা। উইকিতে ক্রসকাল নিয়ে খুব সুন্দর করে লেখা আছে, আমি ওখানকার ছবিগুলোই ব্যবহার করে সংক্ষেপে অ্যালগোরিদমটা বুঝানোর চেষ্টা করবো।

নিচের গ্রাফটি দেখো:



প্রথমে আমাদের ট্রিতে একটি এজও নেই। আমরা মূল গ্রাফের এজগুলোকে cost অনুযায়ী সর্ট করে ফেলবো। সব থেকে কম cost এর এজ আগে নিবো, বেশি cost এর এজ পরে নিবো। দুটি এজের cost সমান হলে যেকোনো একটি আগে নিতে পারি। তারপর একটি করে এজ নিবো আর দেখবো এজের দু প্রত্তের নোডগুলোর মধ্যে ইতোমধ্যে কোনো পথ আছে নাকি, যদি থাকে তাহলে এজটি নিলে সাইকেল তৈরি হবে, তাই এজটা আমরা নিবোনা। বুঝতেই পারছো প্রিমসের মত এটিও একটি 'গ্রিডি' অ্যালগোরিদম।

উপরে AD আর CE হলো সবথেকে কম cost এর এজ। আমরা AD কে সাবগ্রাফের অন্তর্ভুক্ত করলাম।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগোরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

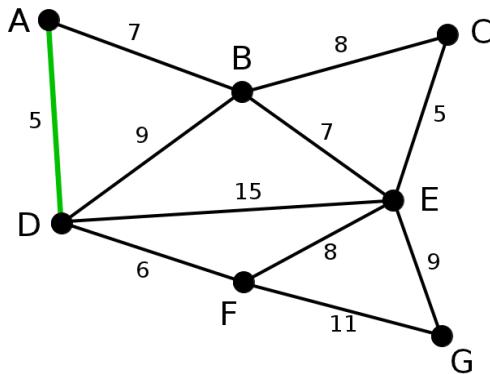
ফ্লয়েড সাইকেল ফাইভিং অ্যালগোরিদম

ডাটা স্ট্রাকচার(১৪):

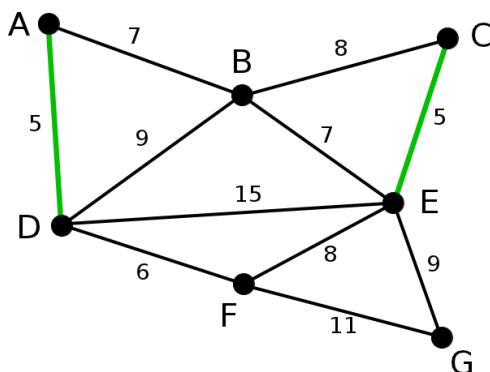
লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



একইভাবে এরপে CE তারপর DF, AB এবং BE কে যোগ করবো:



শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজেনেট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

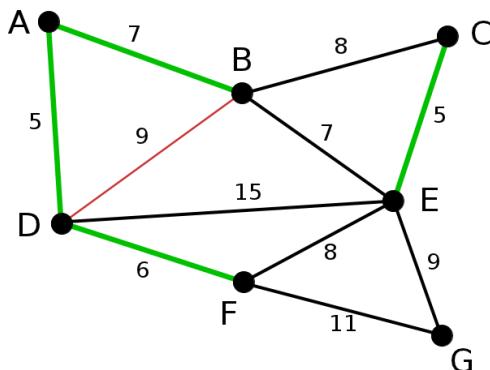
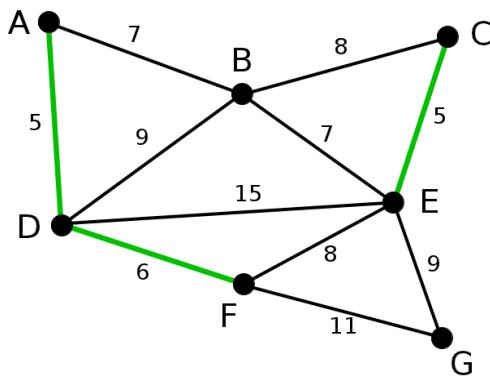
টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

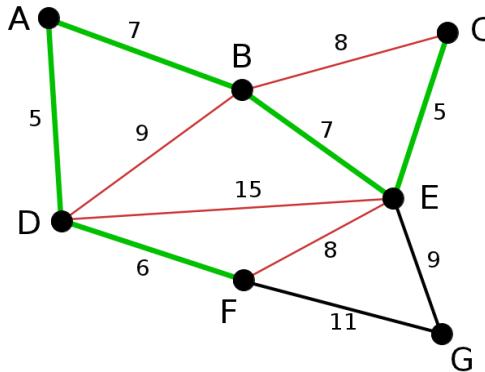
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

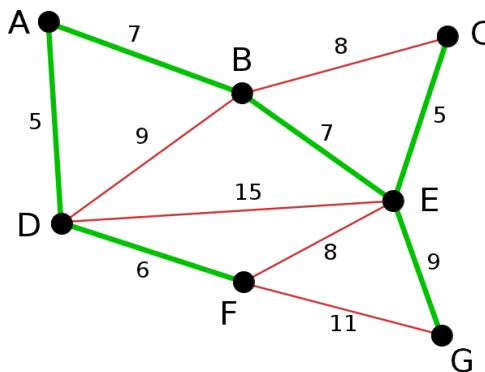
ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)



এরপর সবথেকে ছেট এজ হলো EF, এটাকে আমরা নিতে পারবোনা কারণ EF নিলে একটি সাইকেল তৈরি হয়ে যাবে, E থেকে F তে যাবার রাস্তা আগে থেকেই আছে, তাই এজটি নেয়ার কোনো দরকার নেই। এভাবে BC, DB সহ লাল রঙের এজগুলো বাদ পড়বে কারণ এরা সাইকেল তৈরি করে।

সবশেষে EG যোগ করলে আমরা mst পেয়ে যাবো।



এখন আমরা ইম্প্রিমেন্টেশনে আসি। আমাদের প্রথম কাজ হলো সর্ট করা। পরের কাজ হলো একটি একটি এজ নিয়ে চেক করা যে দু প্রান্তের নোড দুটির মধ্য পথ আছে নাকি, অর্থাৎ তারা একই কম্পোনেন্টের ভিতর আছে নাকি। এটা চেক করতে লাগবে ডিসজয়েন্ট সেট। ডিসজয়েন্ট সেট নিয়ে

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

চিউটোরিয়ালে দেখিয়েছিলাম কিভাবে দুটি নোড একই সাবগ্রাফে আছে নাকি বের করতে হয়। তুমি সেই কাজটাই এখানে করবে। তারপর একই সাবগ্রাফে না থাকলে আগের মত Union ফাংশন কল দিয়ে তাদের একসাথে নিয়ে আসবে আর এজটি একটি ডেষ্টের বা অ্যাবেতে সেভ করে রাখবে।

নিচে একটা ইম্প্রিমেন্টেশন দিলাম, আশা করি এটা কপি না করে নিজে বুঝে লিখবে:

```
1 struct edge {
2     int u, v, w;
3     bool operator<(const edge& p) const
4     {
5         return w < p.w;
6     }
7 };
8 int pr[MAXN];
9 vector<edge> e;
10 int find(int r)
11 {
12     return (pr[r] == r) ? r : find(pr[r]);
13 }
14 int mst(int n)
15 {
16     sort(e.begin(), e.end());
17     for (int i = 1; i <= n; i++)
18         pr[i] = i;
19
20     int count = 0, s = 0;
21     for (int i = 0; i < (int)e.size(); i++) {
22         int u = find(e[i].u);
23         int v = find(e[i].v);
24         if (u != v) {
25             pr[u] = v;
26             count++;
27             s += e[i].w;
28             if (count == n - 1)
29                 break;
30         }
31     }
32     return s;
33 }
34 int main()
```

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

The screenshot shows a Facebook page for 'Shafaetsplanet' with 2.4K likes. A post from about an hour ago discusses dynamic programming, specifically mentioning the knapsack problem and how it relates to solving problems like the one described in the text above. The post includes a link to 'SHAFAEETSPPLANET.COM' and a note about dynamic programming being a 6-month topic.

Be the first of your friends to like this

Shafaetsplanet about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন  
সিরিজের ৬ নম্বর লেখায় আমরা  
দেখবো কম্পিউটারের প্রবলেম এবং  
ডিসিপ্লিন প্রবলেম কিভাবে সমাধান  
করতে হয়।

*i, W) = 1 if W =  
i + 1, W - C[i]) -*

SHAFAEETSPPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ মাস

```

36 {
37     // READ("in");
38     int n, m;
39     cin >> n >> m;
40     for (int i = 1; i <= m; i++) {
41         int u, v, w;
42         cin >> u >> v >> w;
43         edge get;
44         get.u = u;
45         get.v = v;
46         get.w = w;
47         e.push_back(get);
48     }
49     cout << mst(n) << endl;
50     return 0;
51 }

```

### কমপ্লেক্স অ্যানালাইসিস:

মনে করি  $E$  হলো এজ সংখ্যা। এজগুলোকে সর্ট করতে হবে, সেটার কমপ্লেক্সি  $O(E \log E)$ , এবপরে শুধু এজগুলোর উপর লিনিয়ার লুপ চালাতে হবে। তাহলে মোট কমপ্লেক্সি  $O(E \log E)$ ।

**mst** সম্পর্কিত অনেকগুলো সহজ প্রবলেম দিয়েছি প্রিমস এর টিউটোরিয়ালে, ওগুলো সলভ করে প্র্যাকটিস করতে পারো। আরেকটু ভালো প্রবলেম করতে চাইলে দেখো:

### ২য় সেরা স্প্যানিং ট্রি?

অনেক সময় প্রবলেমে বলা হয় সেকেন্ড বেস্ট MST বের করতে। এটা আমরা কুট ফোর্স দিয়ে বের করতে পারি। MST বের করা পর যে এজগুলো পারো সেগুলার প্রত্যেকটা একবার করে বাদ দিয়ে নতুন করে MST বের করতে হবে, এভাবে করে যে MST টা মিনিমাম হবে সেটাই সেকেন্ড বেস্ট MST।

<http://uva.onlinejudge.org/external/103/10369.html>

<http://uva.onlinejudge.org/external/117/11733.html>

# গ্রাফ থিওরিতে হাতেখড়ি ৭:টপোলজিকাল সর্ট

অক্টোবর ৬, ২০১১ by শাফায়েত



## (অন্যান্য পর্য)

মনে কর তোমার হাতে কিছু কাজের একটা তালিকা আছে, কাজগুলো অবশ্যই শেষ করতে হবে। কাজগুলো হলো অফিসে যাওয়া, সকালে নাস্তা করা, টিভিতে খেলা দেখা, কিছু ই-মেইলের উত্তর দেয়া, বন্ধুদের সাথে ডিনার করা ইত্যাদি। কাজগুলো কিন্তু আপনি যেকোনো অর্ডারে করতে পারবেনা, কিছু শর্ত মানতে হবে। যেমন অফিসে যাবার আগে নাস্তা করতে হবে, খেলা দেখার আগে অফিসে যেতে হবে, ডিনারে বসার আগে ইমেইলের উত্তর দিতে হবে।

তুমি শর্তগুলোর তালিকা করে ফেললে:

১. সকালের নাস্তা —> অফিস (ক—> খ এর মানে হলো 'খ' কাজটি করার আগে 'ক' কাজটি করতে হবে)
২. সুট-টাই পড়া —> অফিস
৩. অফিস —> ইমেইল
৪. অফিস —> ডিনার
৫. অফিস —> খেলা
৬. ইমেইল —> ডিনার
৭. ইমেইল —> খেলা
৮. ডিনার —> খেলা

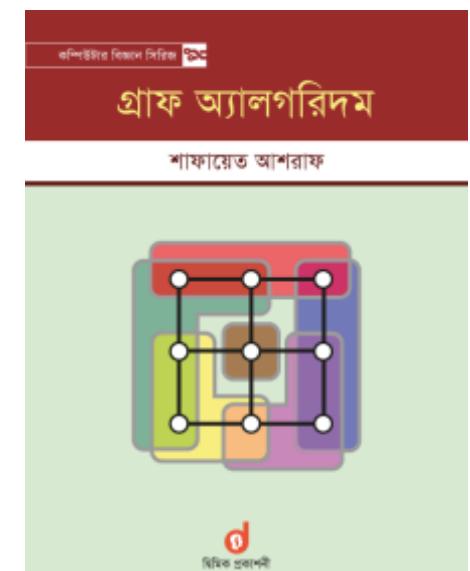
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

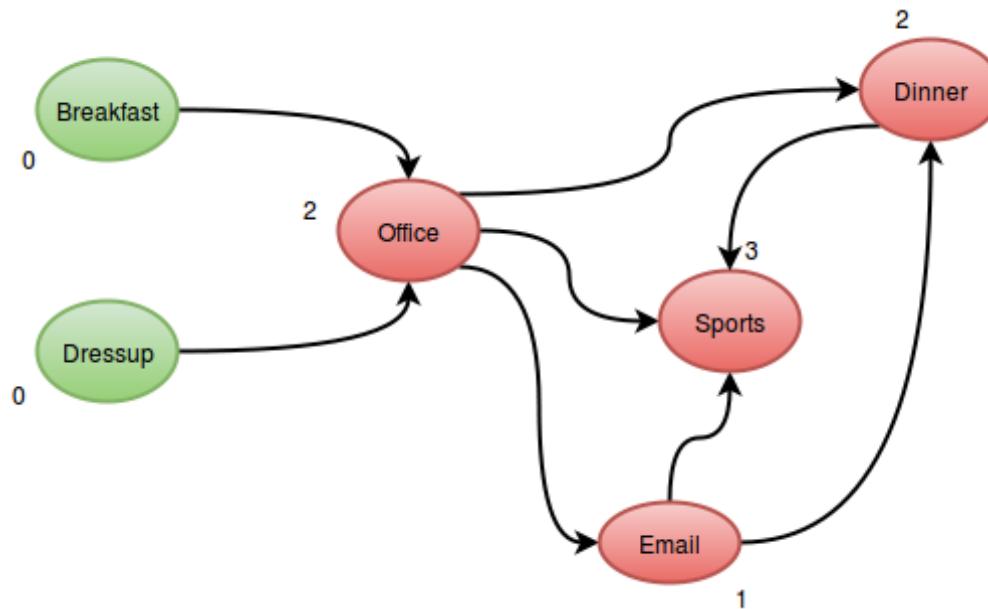
শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



তুমি এখন কোন কাজ কখন করবে? উল্টাপাল্টা অর্ডারে করলে তোমার কাজ ডঙ্গল হয়ে যাবে, ইমেইল না করে খেলা দেখতে বসলে তুমি ক্লায়েন্ট হ্যাবে, তাই অর্ডারিং খুব জরুরি।

এটা একটি "টাস্ক শিডিউলিং" প্রবলেম। কোন কাজের পর কোন কাজ করতে হবে সেটা আমাদের বের করতে হবে। অর্থাৎ এটা এক ধরণের সার্টিং যাকে টপোলোজিকাল সার্টিং বলে। আমরা এ টিউটোরিয়ালে এজ সরিয়ে বা ইনডিগ্রী কমিয়ে টপস্ট বের করবো।

আমরা প্রথমেই সমস্যাটাকে নিচের গ্রাফ দিয়ে মডেলিং করবো:



উপরের ছবিতে প্রতিটা কাজ একটি করে নোট দিয়ে দেখানো হয়েছে। ব্রেকফাস্ট থেকে অফিসের দিকে তীব্রচিহ্ন দিয়ে বুঝানো হচ্ছে যে অফিসে আসার আগে ব্রেকফাস্ট করতে হবে। উপরের ৮টি শর্ত ছবিতে ৮টি ডিরেক্টেড এজ দিয়ে দেখানো হয়েছে।

প্রতিটি নোডের পাশে ছোট করে কিছু সংখ্যা দেখতে পাচ্ছো। যেমন অফিসের সাথে ০, ডিনারের সাথে ২ ইত্যাদি। এগুলো দিয়ে বুঝাচ্ছে একটি কাজ অন্য কয়টি কাজের উপর নির্ভরশীল। যেমন ডিনারের আগে তোমাকে অফিস, ইমেইল এই ২টা কাজ করতে হবে, ডিনার নোভাটিতে ২টি



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

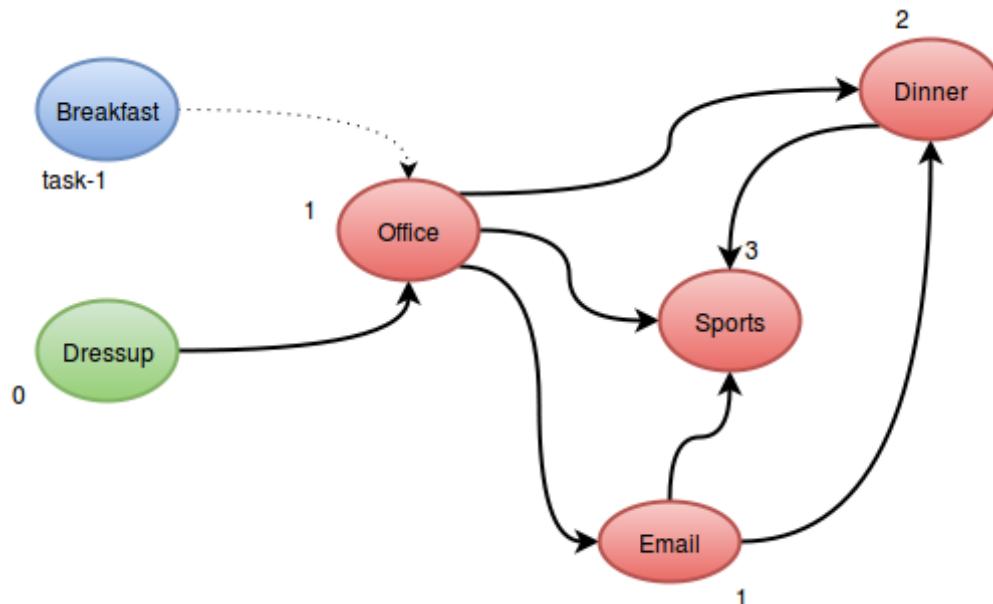
লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

তীব্রচিহ্ন প্রবেশ করেছে, আর আমরা পাশে লিখে দিয়েছি “২”। ঠিক এভাবে ইমেইলের পাশে লেখা হয়েছে ১। এ সংখ্যাগুলোকে indegree বলা হয়।

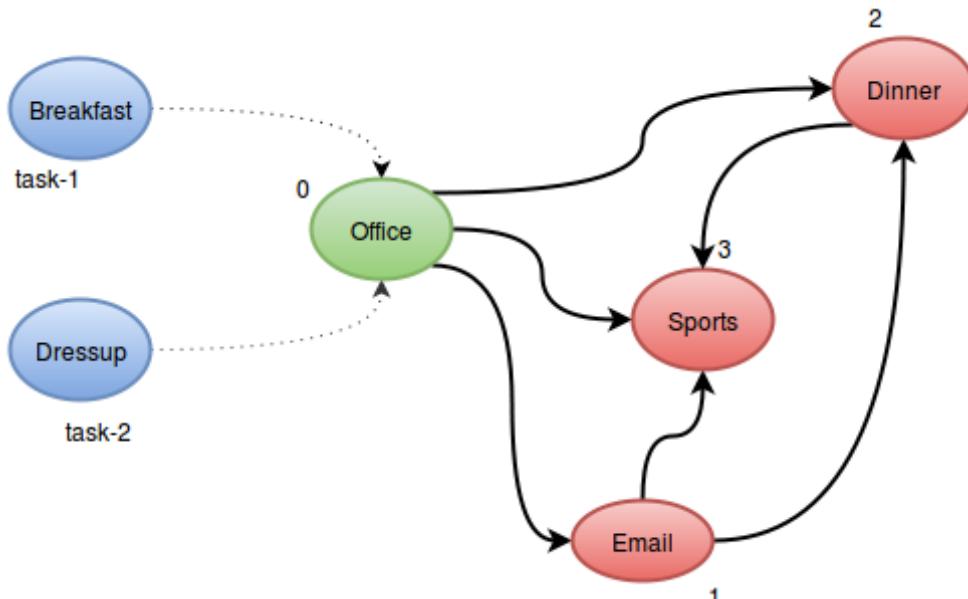
লক্ষ্য কর ব্রেকফাস্ট এবং ড্রেসআপ কোনো কাজের উপর নির্ভরশীল নয়, তাই তাদের পাশে ০ লেখা হয়েছে। তারমানে আমরা এ দুটি কাজের যেকোনোটা দিয়ে দিন শুরু করতে পারি। মনে করি তুমি নাস্তা আগে খেতে চাও। নাস্তা খেয়ে নেবার পর যেসব কাজ ব্রেকফাস্টের উপর নির্ভরশীল ছিল তারা আর সেটার উপর নির্ভরশীল থাকলোনা, গ্রাফটা হয়ে গেল এরকম:



ব্রেকফাস্ট থেকে অফিসের তীব্রচিহ্ন সরিয়ে দিয়েছি। এখন অফিস আর মাত্র ১টি কাজের উপর নির্ভরশীল(আগে ছিল ২টির উপর)। এবার তোমাকে ড্রেসআপ করতে হবে, কাবণ এখন একমাত্র এই কাজটিই কাবো উপর নির্ভরশীল নয়:

স্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)  
 ডিসজয়েট সেট(ইউনিয়ন ফাইল)  
 ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)  
 সেগমেন্ট ট্রি-১  
 সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)  
 অ্যারে কমপ্রেশন/ম্যাপিং  
 লোয়েস্ট কমন অ্যানসেক্টর  
 বাইনারি ইনডেক্সড ট্রি  
 স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:  
 এল-আর-ইউ ক্যাশ(নতুন)  
 ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):  
 গ্রাফ থিওরিতে হাতেখড়ি  
 অ্যাডজেসেন্সি ম্যাট্রিক্স  
 অ্যাডজেসেন্সি লিস্ট  
 ব্রেথড ফার্স্ট সার্চ (বিএফএস)  
 মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)  
 মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)  
 টপোলজিকাল সর্ট  
 ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
 সর্ট  
 ডায়াক্রস্টা  
 ফ্লয়েড ওয়ার্শল  
 বেলম্যান ফোর্ড



ড্রেসআপ থেকে অফিসের তীরচিহ্ন সরিয়ে দেয়া হয়েছে, আর কোনো কাজ নেই, এবাব তুমি অফিসে যাবার জন্য প্রস্তুত। অফিসে যাবার উপর যারা নির্ভরশীল তাদের তীরচিহ্নগুলো এখন সরিয়ে দিতে পারি:

আর্টিফিশিয়াল ইণ্ডিপেন্ডেন্স

স্ট্রাংলিং কানেক্টেড কম্পানেন্ট

ম্যাক্সিমাম ফ্লো-১

ম্যাক্সিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটাৰ নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

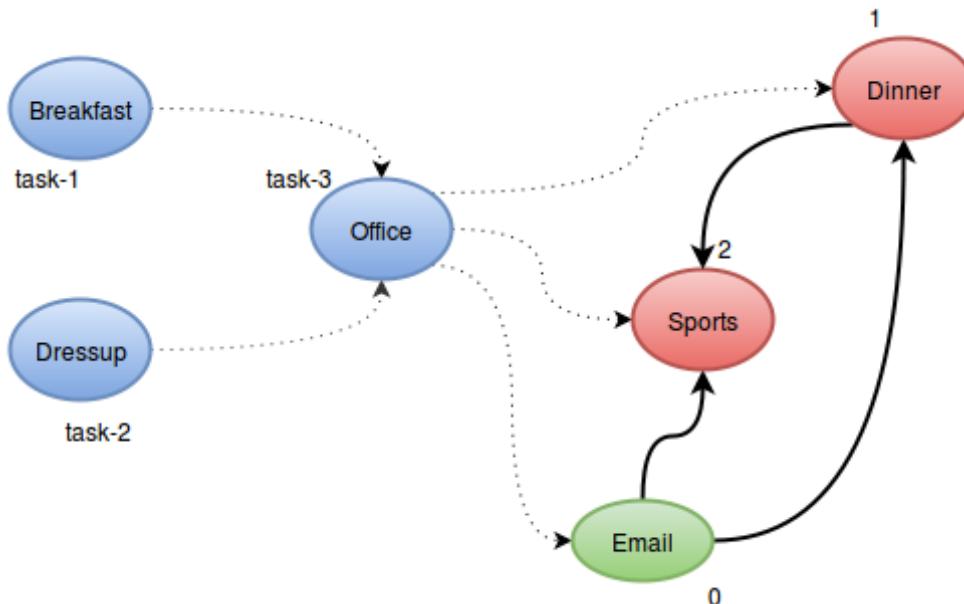
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

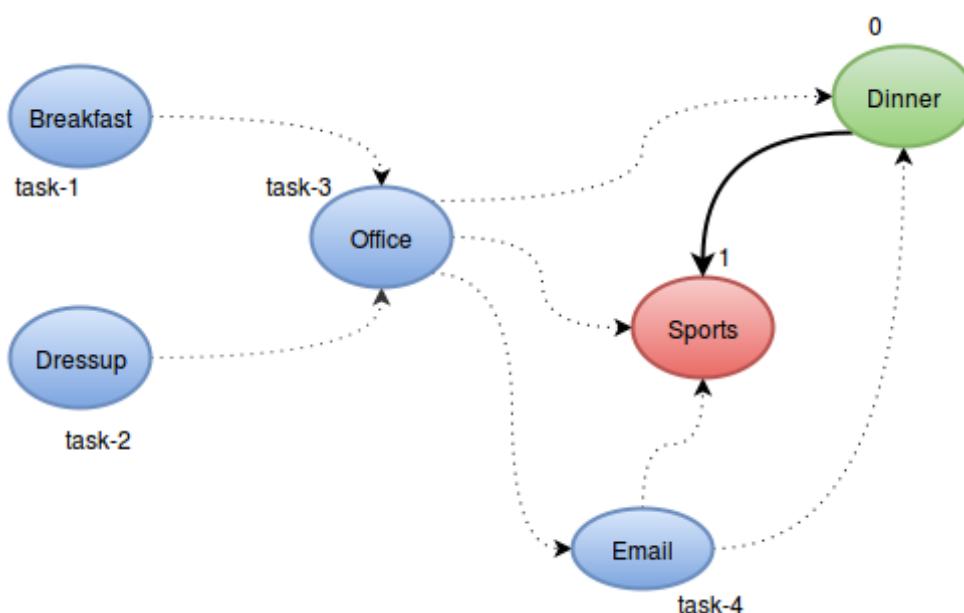
ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্বিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/O-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)



এখন ইমেইল “নোড” এর নির্ভরশীলতা ০ হয়ে গিয়েছে:



[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

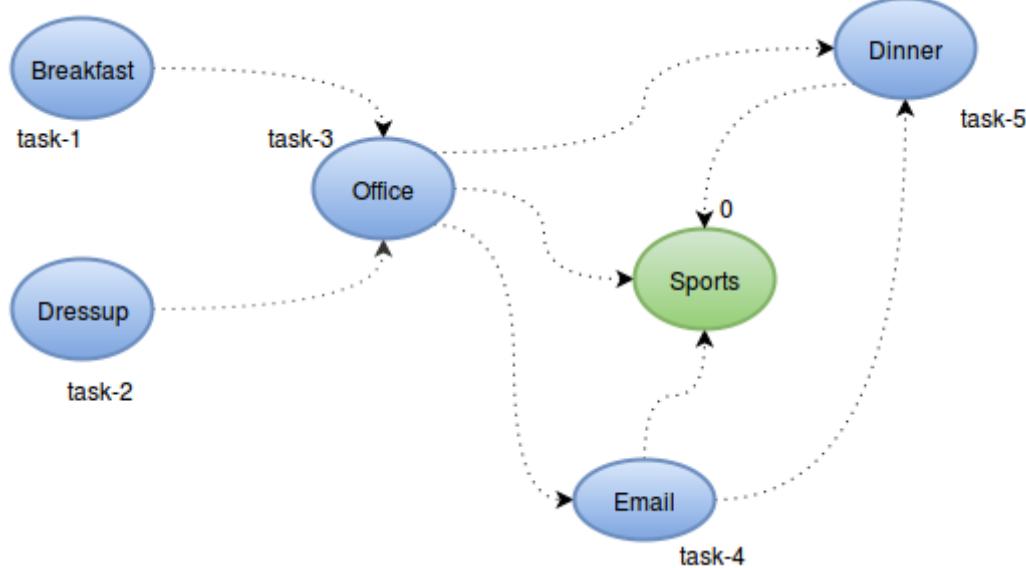
ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

এরপর ডিনার:



সবশেষে খেলা দেখতে বসা:

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

Shafaetsplanet  
2.4K likes  
ফাফ অ্যালগরিদম  
শাফায়েত আশরাফ

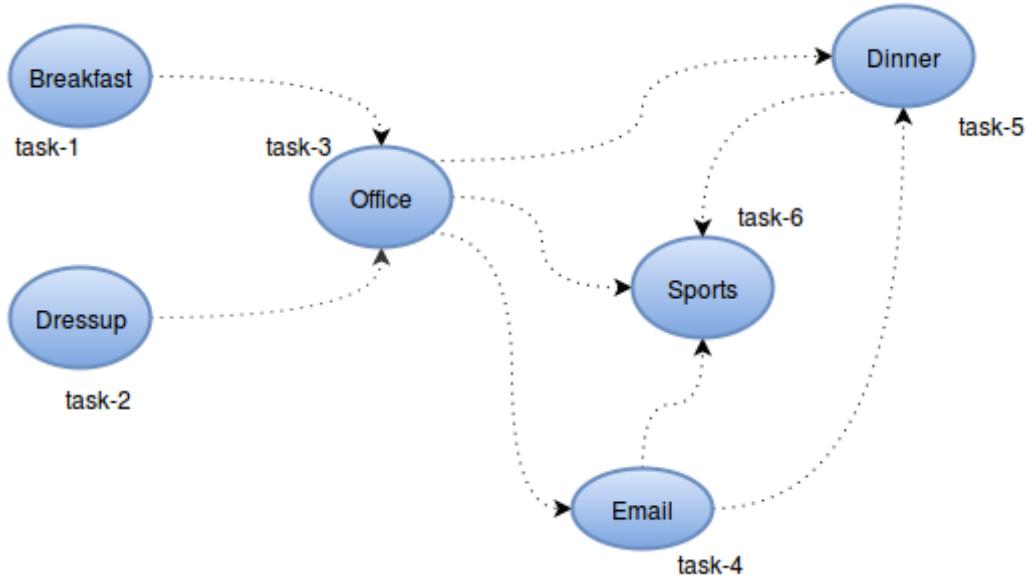
**Like Page**

Be the first of your friends to like this

Shafaetsplanet about an hour ago  
ডায়ানামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিঙের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETSPPLANET.COM  
ডায়ানামিক প্রোগ্রামিং ৬ / মার



এখন তুমি কাজের অর্ডারিং পেয়ে গিয়েছো, নাস্তা করা, অফিসের পোষাক পড়া, অফিসে যাওয়া, ইমেইল করা, ডিনার করা, খেলা দেখা।

এটাকেই বলা হ্য টপোলোজিক্যাল সর্ট (topological sort) বা টপসর্ট। মূলত কাজের অর্ডারিং খুজে বের করতে এই অ্যালগোরিদমটি ব্যবহার করা হয়। কম্পিউটার তার অভ্যন্তরে বিভিন্ন কাজের অর্ডারিং ঠিক করতে টপসর্ট ব্যবহার করে। অনেক রিয়েল লাইফ প্রয়োগ থাকায় টপসর্ট কম্পিউটার সায়েন্সে খুবই গুরুত্বপূর্ণ একটি টপিক। টপসর্ট বের করার আরেকটি পদ্ধতি আছে যার নাম “ডেপথ ফার্স্ট সার্চ”, এটা নিয়ে [আলোচনা করেছি এই লেখায়](#)।

উপরের অ্যালগোরিদমটি  $O(n^2)$  এ কাজ করে। একটি গ্রাফের অনেকগুলো সর্টেড অর্ডার থাকতে পারে, যেমন উপরের সমস্যায় তুমি নাস্তা খাবার আগে জামা-কাপড় পরতে পারতে। তোমাকে লেক্সিকোগ্রাফিকালি ছোটটা প্রিন্ট করতে বলতে পারে অথবা যেটা ইনপুটে আগে আছে সেটাকে আগে প্রিন্ট করতে বলতে পারে, আশা করি এসব ক্ষিণিত সহজে হ্যান্ডল করতে পারবে।

ইমপ্লিমেন্টেশন নিয়ে তেমন কিছু বলার নেই। সবগুলো নোডের indegree বের করবে। তারপর যার indegree শূন্য তার সাথে যাদের এজ আছে তাদের indegree 1 কমিয়ে দিবে, তারপর আবার খুজবে কার indegree এখন শূন্য। এক নোডকে কখনো ২বার নিবেন। ছবিতে এজ উঠিয়ে দেখিয়েছি বুধানোর জন্য, তোমার ম্যাট্রিক্স থেকে এজ উঠানোর দরকার নেই, indegree কমালেই চলবে।

অনেক সময় বলতে পারে যতৰকম ভাবে topsort করা যায় সবগুলো বের করতে। তখন তোমাকে **backtracking** এর সাহায্য নিতে হবে।

নিচের সমস্যাগুলো সমাধান করে ফেলো:

<http://uva.onlinejudge.org/external/103/10305.html>(Easy,straight-forward,special judge)

<http://uva.onlinejudge.org/external/110/11060.html>(Easy)

<http://uva.onlinejudge.org/external/1/124.html>(Medium,All possible topsort)

<http://uva.onlinejudge.org/external/4/452.html>(Medium)

(অন্যান্য পোস্ট)

## ফেসবুকে মন্তব্য

0 comments

**0 Comments**

Sort by **Oldest**



Add a comment...

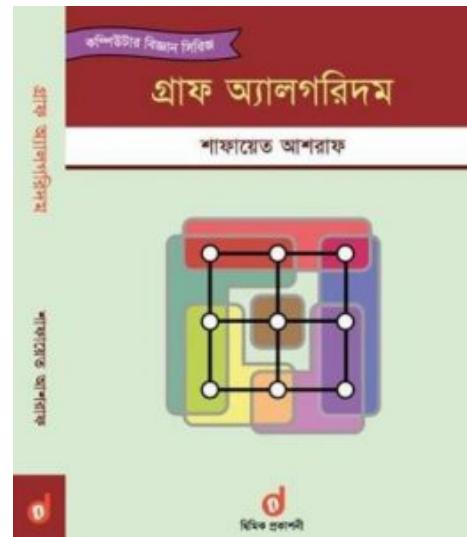
Facebook Comments plugin

Powered by [Facebook Comments](#)



# গ্রাফ থিওরিতে হাতেখড়ি ৮: ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

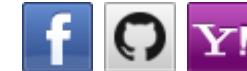
মার্চ ৮, ২০১২ by শাফায়েত



আগের পরগুলো পড়ে থাকলে হ্যতো ডেপথ ফার্স্ট সার্চ বা ডিএফএস এতদিনে নিজেই শিখে ফেলেছো। তাবপরেও এই টিউটোরিয়ালটি পড়া দরকার কিছু কনসেপ্ট জানতে।

ডিএফএস এ আমরা গ্রাফটাকে লেভেল বাই লেভেল সার্চ করেছিলাম, নিচের ছবির মতো করে:

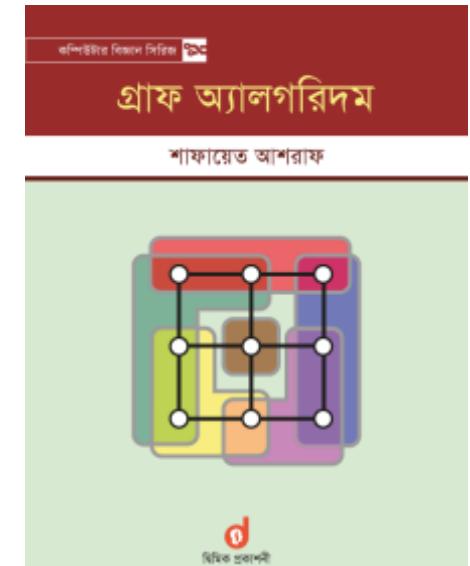
সাবস্ক্রাইব

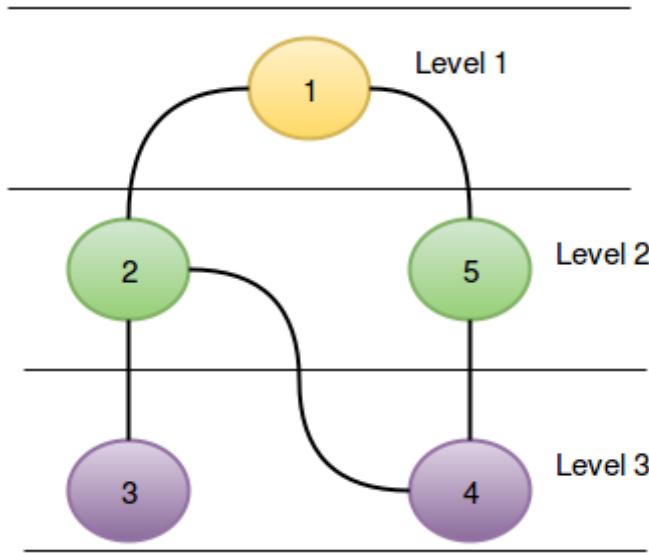


Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)





এবাব আমরা কোনো নোড পেলে সাথে সাথে সে নোড থেকে আরো গভীরে চলে যেতে থাকবো, যখন আর গভীরে যাওয়া যাবেনা তখন আবাব  
আগের নোডে ফিরে এসে অন্য আরেক দিকে যেত চেষ্টা করবো, এক নোড কখনো ২বার ডিজিট করবোনা। আমরা নোডের ৩টি রং(কালার) দিবো:

**“ সাদা নোড= যে নোড এখনো খুজে পাইনি/ডিজিট করিনি।**

**গ্রে বা ধূসর নোড= যে নোড ডিজিট করেছি কিন্তু নোডটি থেকে যেসব চাইল্ড নোডে যাওয়া যায়  
সেগুলো এখনো ডিজিট করে শেষ করিনি, অর্থাৎ নোডটিকে নিয়ে কাজ চলছে।**

**কালো নোড= যে নোডের কাজ সম্পূর্ণশেষ।**

এবাব আমরা ধাপগুলো দেখতে পাৰি:



প্ৰোগ্ৰামিং কনটেন্ট এবং অ্যালগোৰিদম

অনুপ্ৰৱণা(৪):

কেন আমি প্ৰোগ্ৰামিং শিখবো?

কম্পিউটাৰ বিজ্ঞান কেন পড়বো?

প্ৰোগ্ৰামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্ৰোগ্ৰামাৰ

অ্যালগোৰিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্ৰবলেম

বাইনাৰি সাৰ্চ - ১

বাইনাৰি সাৰ্চ - ২(বাইসেকশন)

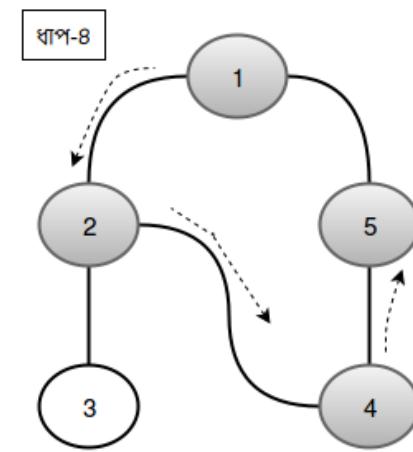
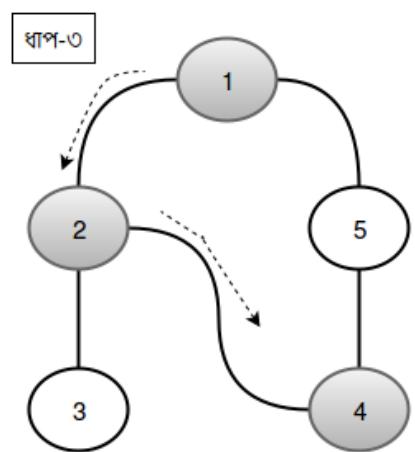
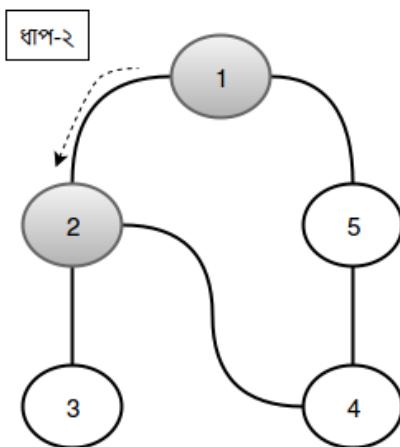
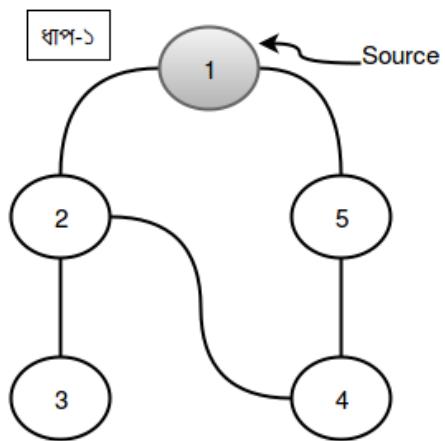
ফুয়েড সাইকেল ফাইডিং অ্যালগোৰিদম

ডাটা স্টোকচাৰ(১৪):

লিংকড লিস্ট

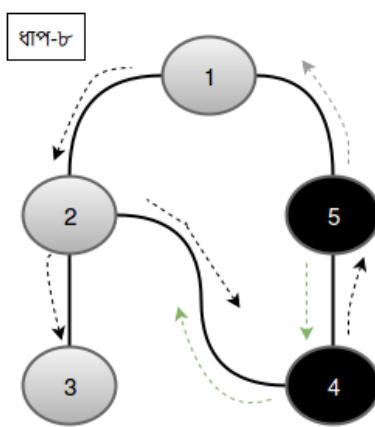
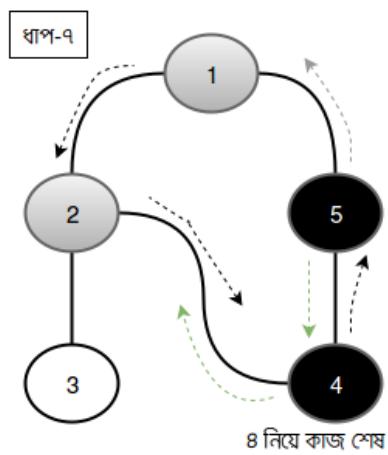
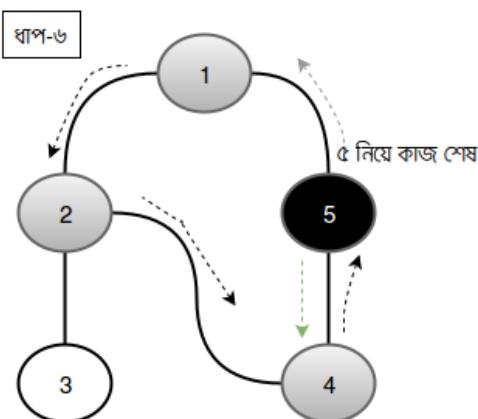
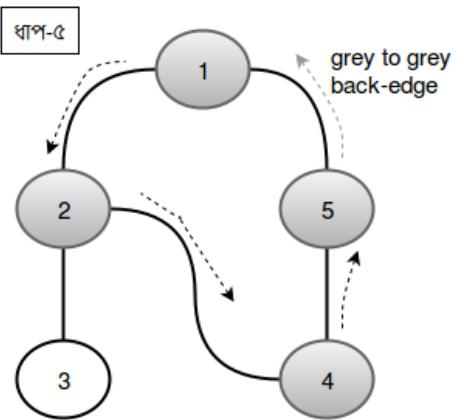
স্ট্যাক

কিউ+সার্কুলাৰ কিউ



শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)  
 ডিসজয়েট সেট(ইউনিয়ন ফাইল)  
 ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)  
 সেগমেন্ট ট্রি-১  
 সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)  
 অ্যারে কমপ্রেশন/ম্যাপিং  
 লোয়েস্ট কমন অ্যানসেক্টর  
 বাইনারি ইনডেক্সড ট্রি  
 স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:  
 এল-আর-ইউ ক্যাশ(**নতুন**)  
 ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):  
 গ্রাফ থিওরিতে হাতেখড়ি  
 অ্যাডজেসেন্সি ম্যাট্রিক্স  
 অ্যাডজেসেন্সি লিস্ট  
 ব্রেথড ফার্স্ট সার্চ (বিএফএস)  
 মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)  
 মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)  
 টপোলজিকাল সর্ট  
 ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
 সর্ট  
 ডায়াক্রস্ট্রা  
 ফ্লয়েড ওয়ার্শল  
 বেলম্যান ফোর্ড



আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

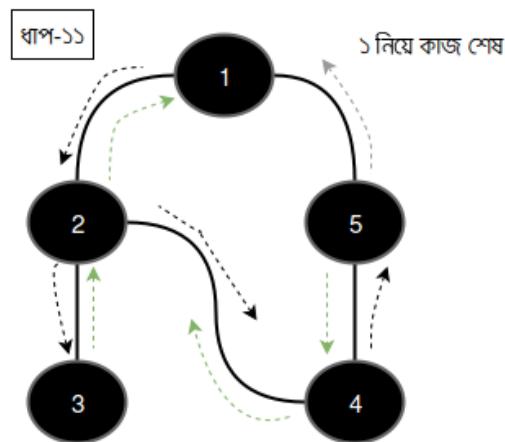
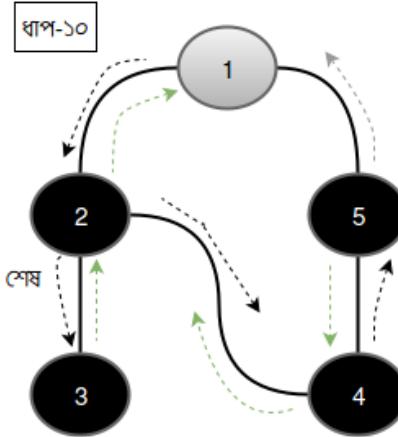
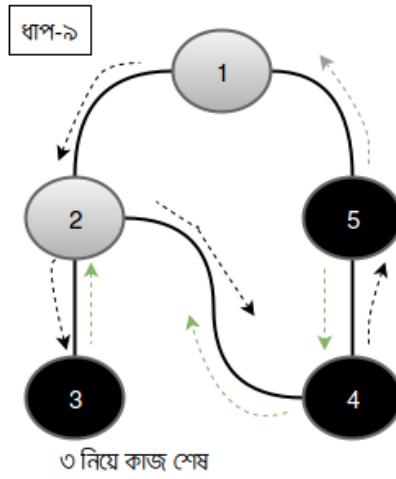
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)



আশা করি ডিএফএস কিভাবে কাজ করে এটা পরিষ্কার, খুব সহজ জিনিস এটা। এবার আমরা একটা খুব গুরুত্বপূর্ণ টার্ম শিখবো, সেটা হলো ব্যাকএজ(backedge)। লক্ষ করো ৫-১ কে ব্যাকএজ বলা হয়েছে। এর কারণ হলো তখনও ১ এর কাজ চলছে, ৫ থেকে ১ এ যাওয়া মানে এমন একটা নোড ফিরে যাওয়া যাকে নিয়ে কাজ এখনো শেষ হ্যানি, তারমানে অবশ্যই গ্রাফে একটি সাইকেল আছে। এ ধরনের এজকে ব্যাকএজ বলে, dfs এ যদি কোনো সময় একটি গ্রে নোড থেকে আবেকটি গ্রে নোডে যেতে চেষ্টা করে তাহলে সে এজটি ব্যাকএজ এবং গ্রাফে অবশ্যই

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

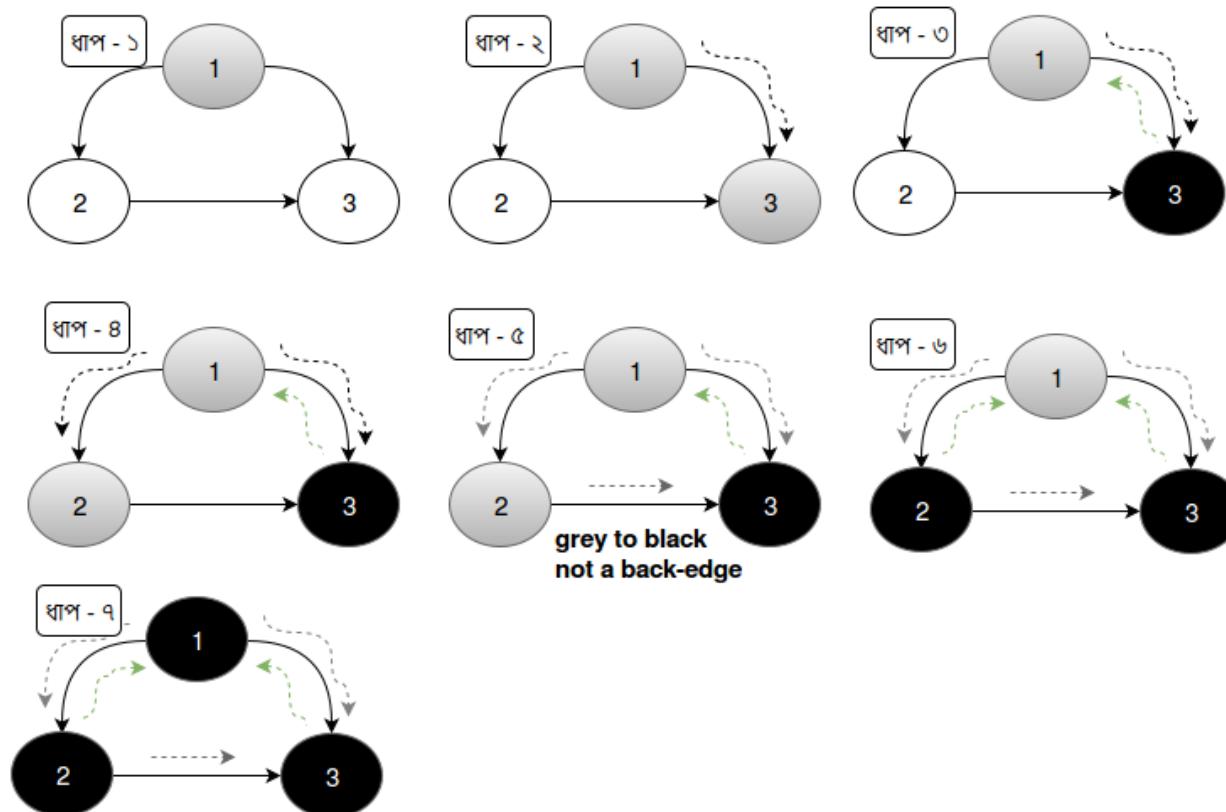
টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

সাইকেল আছে। dfs এর সোর্স নোড এবং নোড ডিজিট করার অর্ডার এর উপর নির্ভর করে সাইকেলে যে কোনো এজকে ব্যাকএজ হিসাবে পাওয়া যেতে পারে, যেমন ১ থেকে আগে ২ এ না গিয়ে ৫ এ গেলে পরে ২-১ কে ব্যাকএজ হিসাবে পাওয়া যেতো।

আর যখন আমরা স্বাভাবিক ভাবে গ্রে থেকে সাদা নোডে যাচ্ছি তখন সে এজগুলোকে বলা হয় ট্রি এজ। শুধুমাত্র ট্রি এজ গুলো রেখে বাকি এজগুলো মুছে দিলে যে গ্রাফটা থাকে তাকে বলা হয় ডিএফএস ট্রি।

আনডিরেস্টেড গ্রাফের ক্ষেত্রে আগে ডিজিট করা কোনো নোডে ফিরে গেলেই সেটা ব্যকএজ, কালার চেক না করলেও হ্য। তবে ডিরেস্টেড গ্রাফের ক্ষেত্রে অবশ্যই করতে হবে। পরের ছবিটা দেখো:



কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

 Shafaetsplanet  
2.4K likes  
**শাফায়েত আশরাফ**

**Like Page**

Be the first of your friends to like this

 Shafaetsplanet  
about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্বিনেটরিক্যাল প্রবলেম এবং ডিসিশন প্রবলেম কিভাবে সমাধান করতে হ্য।

*i, W) = 1 if W =  
i + 1, W - C[i]) -*

SHAFAEETPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / ম্যাপ

২-৩এর এজটাকে ব্যাকএজ বলা যাচ্ছনা, কারণ ৩ এর কাজ আগেই শেষ হয়ে গেছে।

প্রতিটা নোড আব এজ নিয়ে একবার করে করছি, dfs এর কমপ্লেক্সিটি  $O(V+E)$ ।

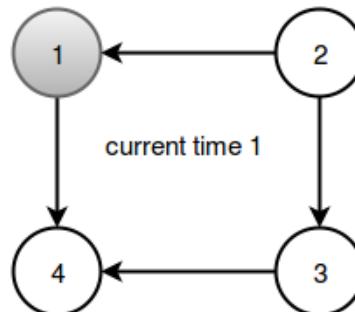
আমরা টপোলজিকাল সর্টের সমস্যা সমাধান করেছিলাম বারবার indegree উঠিয়ে। এবার আমরা খুব সহজে dfs দিয়ে এটা করবো।

টপোলজিকাল কি সেটা না জানলে আগে [এই পোস্টটা পড়ো](#), তারপর আগাও।

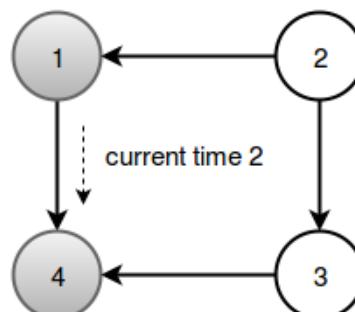
মনে করি আমাদের এজগুলো হলো: ২-১, ২-৩, ৩-৪, ১-৪। অর্থাৎ ১ নম্বর কাজ করার আগে ২ নম্বরটি করতে হবে ইত্যাদি। এবার আমরা dfs চালানোর সময় একটি স্টপওয়াচ চালু করে দিবো। আব কোনো নোড নিয়ে কাজ শুরু করলে ঘড়ি দেখে নোডটি starting time/discovery time লিখে রাখবো, কাজ শেষ হলো নোডটির finishing time লিখে রাখবো।

$d[]$ =discovery time  
 $f[]$ =finishing time

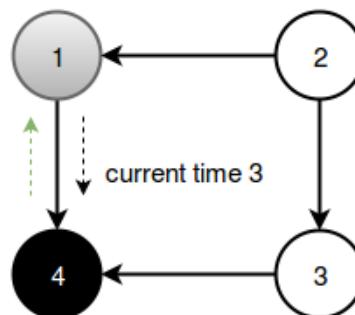
node	$d[]$	$f[]$
1	1	null
2	null	null
3	null	null
4	null	null



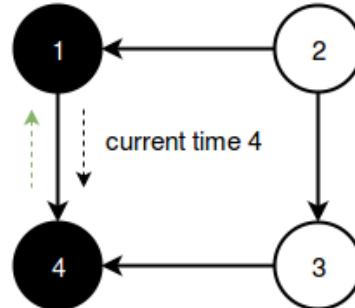
node	$d[]$	$f[]$
1	1	null
2	null	null
3	null	null
4	2	null



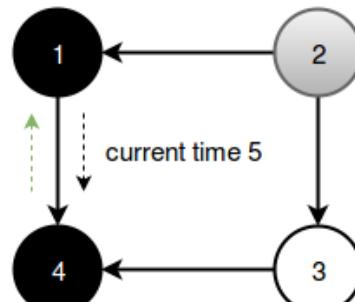
node	$d[]$	$f[]$
1	1	null
2	null	null
3	null	null
4	2	3



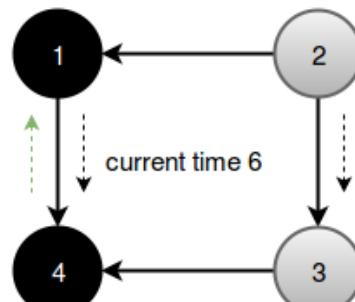
node	d[]	f[]
1	1	4
2	null	null
3	null	null
4	2	3



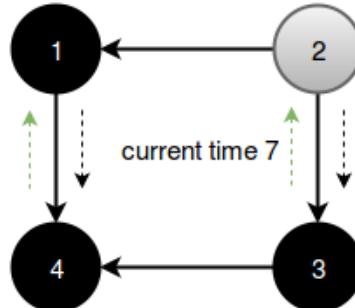
node	d[]	f[]
1	1	4
2	5	null
3	null	null
4	2	3



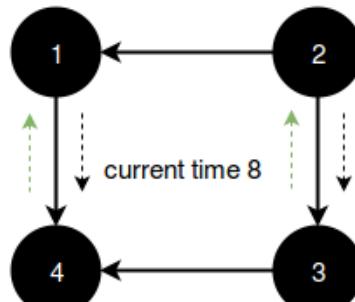
node	d[]	f[]
1	1	4
2	5	null
3	6	null
4	2	3



node	d[]	f[]
1	1	4
2	5	null
3	6	7
4	2	3



node	d[]	f[]
1	1	4
2	5	8
3	6	7
4	2	3



ফিনিশিং টাইম অনুযায়ী সর্ট করে পাই: ২,৩,১,৮

**finishing time** দেখে আমরা সহজেই টপস্ট করতে পারি। যে নোডটি সবার আগে আসবে তার finishing time অবশ্যই সবথেকে বেশি হবে, কারণ প্রথম নোডের উপর নির্ভরশীল সব নোড ঘুরে আসার পরে সে নোডের finishing time assign করা হয়। [uva 11504](#)-  
dominos প্রবলেমে আগে নোডগুলোকে finishing time দিয়ে সর্ট করে তারপর আবার dfs চালাতে হয়, প্রবলেমটা ছেষ করো।

ডিএফএস দিয়ে আমরা যেসব কাজ করি সেগুলোর অনেকগুলোই bfs দিয়ে করতে পারি। bfs এ সাধারণত টাইম কমপ্লেক্সিটি কম হয় তবে dfs কোডিং করতে খুব কম সময় লাগে। একটা সিম্পল dfs এর সুড়োকোড এরকম:

```

1 procedure DFS(G, source):
2   U ← source
3   time ← time+1

```

```
4   d[u] ← time
5   color[u] ← GREY
6   for all edges from u to v in G.adjacentEdges(v) do
7     if color[v] = WHITE
8       DFS(G,v)
9     end if
10  end for
11  color[u] ← BLACK
12  time ← time+1
13  f[u] ← time
14  return
```

নিচের প্রবলেমগুলো সলভ করতে চেষ্টা করো:

<http://uva.onlinejudge.org/external/2/280.html>

<http://uva.onlinejudge.org/external/115/11518.html>

<http://uva.onlinejudge.org/external/104/10452.html>

এবপরে [এই আর্টিকেলটা](#) পড়ে ফেলো বিস্তারিত জানার জন্য,আমার লেখা পড়ে তুমি বেসিকটা শিখতে পারবে,বিস্তারিত জানতে এবং কঠিন প্রবলেম সলভ করতে আরো অনেক কিছু জানতে হবে।

## ফেসবুকে মন্তব্য

1 comments

# গ্রাফ থিওরিতে হাতেখড়ি-৯ (ডায়াক্রট্রা)

এপ্রিল ৯, ২০১৩ by শাফায়েত



আমরা শুক্রতেই শিখেছি কিভাবে শর্টেস্ট পাথে এক জায়গা থেকে আবেক জায়গায় যেতে হয়। সেজন্য আমরা শিখেছি **ওডথ ফার্স্ট সার্চ** নামের একটি সার্চিং অ্যালগরিদম। অ্যালগোরিদমটি চমৎকার কিন্তু সমস্যা হলো সে ধরে নেয় প্রতিটি রাস্তা দিয়ে যেতে সমান সময় লাগে, মানে সব এজ এব কস্ট সমান। প্র্যাকটিকাল লাইফে বেশিভাগ ক্ষেত্রেই এটা অচল হয়ে পড়ে, তখন আমাদের দরকার পরে ডায়াক্রট্রা। প্রথমে নাম শুনে আমার ধারণা হয়েছিলো ডায়াক্রট্রা খুবই ভয়ংকর কোনো জিনিস কিন্তু আসলে বিএফএস লেখার মতোই সহজ ডায়াক্রট্রা লেখা, আমি তোমাদের দেখানোর ছেঁটা করবো কিভাবে বিএফএসকে কিছুটা পরিবর্তন করে একটা প্রায়োরিটি কিউ যোগ করে সেটাকে ডায়াক্রট্রা বানিয়ে ফেলা যায়।

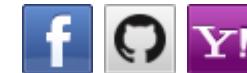
ডায়াক্রট্রা শুরু করার আগে আমরা পাথ রিল্যাক্সেশন(relax) নামের একটা ছোট জিনিসের সাথে পরিচিত হই। ধরো সোর্স থেকে প্রতিটা নোডের ডিস্টেন্স রাখা হয়েছে  $d[]$  অ্যাবেতে। যেমন  $d[3]$  মানে হলো সোর্স থেকে বিভিন্ন এজ পার হয়ে ৩ এ আসতে মোট  $d[3]$  ডিস্টেন্স লেগেছে। যদি ডিস্টেন্স জানা না থাকে তাহলে ইনফিনিটি অর্থাৎ অনেক বড় একটা মান রেখে দিবো। আর  $cost[u][v]$  তে রাখা আছে  $u-v$  এজ এর cost।

ধরো তুমি বিভিন্ন জায়গা ঘুরে ফার্মগেট থেকে টিএসসি তে গেলে ১০ মিনিটে, আবার ফার্মগেট থেকে কার্জন হলে গেলে ২৫ মিনিটে। তাহলে তোমার কাছে ইনফরমেশন আছে:

“  $d[\text{টিএসসি}] = 10, d[\text{কার্জনহল}] = 25$

এখন তুমি দেখলে টিএসসি থেকে ৭ মিনিটে কার্জনে চলে যাওয়া যায়,

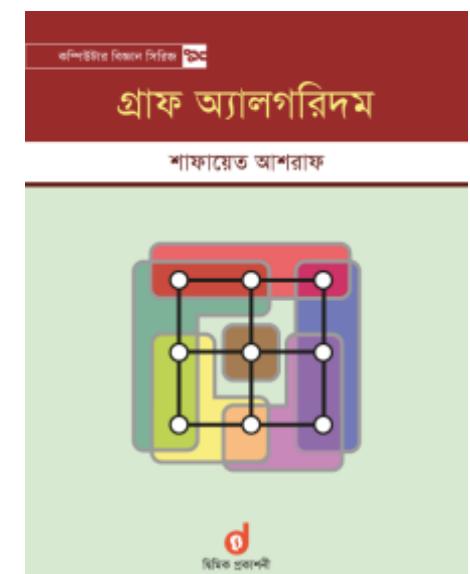
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



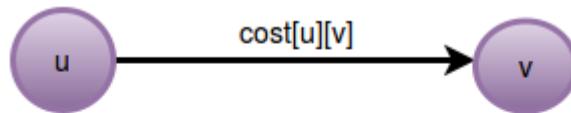
$cost[\text{টিএসসি}][\text{কার্জন}] = 9$

“

তাহলে তুমি ২৫ মিনিটের জায়গায় মাত্র  $10 + 7 = 17$  মিনিটে কার্জনহলে যেতে পারবে। যেহেতু তুমি দেখেছো:

“  $d[\text{টিএসসি}] + cost[\text{টিএসসি}][\text{কার্জন}] < d[\text{কার্জনহল}]$

তাই তুমি এই নতুন রাস্তা দিয়ে কার্জন হলে গিয়ে  $d[\text{কার্জনহল}] = d[\text{টিএসসি}] + cost[\text{টিএসসি}][\text{কার্জন}]$  বানিয়ে দিতেই পারো!!



if  $d[u] + cost[u][v] < d[v]$   
 $d[v] = d[u] + cost[u][v]$

উপরের ছবিটা সেটাই বলছে। আমরা  $u$  থেকে  $v$  তে যাবো যদি  $d[u] + cost[u][v] < d[v]$  হয়। আর  $d[v]$  কে আপডেট করে  $d[v] = d[u] + cost[u][v]$  বানিয়ে দিবো। ভবিষ্যতে যদি কার্জনহলে অন্য রাস্তা দিয়ে আরো কম সময়ে যেতে পারি তখন সেই রাস্তা এভাবে কম্পেয়ার করে আপডেট করি দিবো। ব্যাপারটা অনেকটা এককম:

```
1 if  $d[u] + cost[u][v] < d[v]$ :  
2    $d[v] = d[u] + cost[u][v]$ 
```

উপরের অংশটা যদি বুঝে থাকো তাহলে ডায়াক্লিউট্রা বোধার ৬০% কাজ হয়ে গেছে। না বুঝে থাকলে আবার পড়ো।

বিএফএস নিষ্যষ্ট তুমি ভালো করে বুঝো। **বিএফএস** এ আমাদের একটা নোডে কখনো দুইবার যাওয়া দরকার হ্যনি, আমরা প্রতিবার দেখেছি একটা নোড ভিজিটেড কিনা, যদি ভিজিটেড না হয় তাহলে সেই নোডকে কিউতে পুশ করে দিয়েছি এবং ডিস্টেন্স ১ বাড়িয়ে দিয়েছি। ডায়াক্লিউট্রাতে আমরা একই ভাবে কিউ তে নোড রাখবো তবে ভিজিটেড দিয়ে আপডেট না করে নতুন এজকে “রিল্যাক্স” বা আপডেট করবো উপরের পদ্ধতিতে। নিচের গ্রাফটা দেখো:



প্রোগ্রামিং কনটেন্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

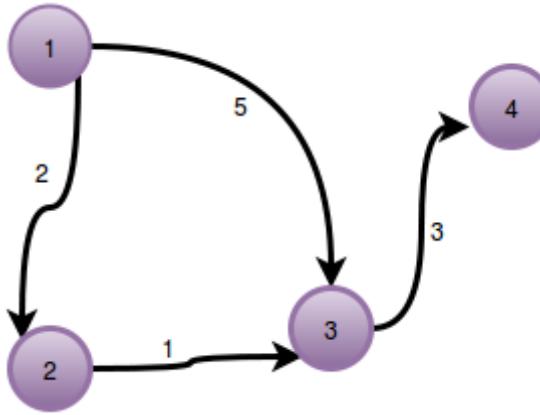
ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



ধরে নেই সোর্স হলো ১ নম্বর নোড। তাহলে

*“  $d[1] = 0, d[2] = d[3] = d[4] = \text{infinity}$ (a large value)*

ইনফিনিটি কারণ ২,৩,৪ এর দূরত্ব আমরা এখনো জানিনা, আর সোর্সের দূরত্ব অবশ্য শূন্য। এখন তুমি আগের বিএফএস এর মতোই সোর্স থেকে যতগুলো নোডে যাওয়া যায় সেগুলা আপডেট করার চেষ্টা করো, আপডেট করতে পারলে কিউতে পুশ করো। যেমন ১ – ২ এজটা ধরে আমরা আগাবো কারণ  $d[1] + 2 < d[2]$  এই শর্তটা পূরণ হচ্ছে। তখন  $d[2]$  হয়ে যাবে ২, একই ভাবে ১ থেকে ৩ এ গেলে  $d[3]$  হয়ে যাবে ৫।

কিন্তু ৫ তো ৩ নম্বরনোডে যাওয়ার শর্টেস্ট ডিসটেন্স না! আমরা বিএফএস এ দেখেছি একটা নোড একবারের বেশি আপডেট হয়না, সেই প্রোপার্টি এখনে কাজ করছেনা। ২ নম্বর নোড থেকে ২-৩ এজ ধরে এগিয়ে আবার আপডেট করলে তখন  $d[3]$  তে  $d[2] + 1 = 3$  পাবো। তাহলে আমরা দেখলাম এক্ষেত্রে একটা নোড অনেকবার আপডেট হতে পাবে। (প্রশ্ন: সর্বোচ্চ কত বার?)

আমরা তাহলে আগের বিএফএস এর সুড়োকোডের আপডেট অংশ একটু পরিবর্তন করি যাতে একটা নোড বার বার আপডেট হতে পাবে:

```

1 procedure BFSmodified(G,source):
2     Q = queue(), distance[] = infinity
3     Q.enqueue(source)
4     distance[source] = 0
5     while Q is not empty
6         u ← Q.pop()
  
```

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইন্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

7     for all edges from u to v in G.adjacentEdges(v) do
8         if distance[u] + cost[u][v] < distance[v]
9             distance[v] = distance[u] + cost[u][v]
10            Q.enqueue(v)
11        end if
12    end for
13 end while
14 Return distance

```

আমরা ঠিক আগের বিএফএস এর কোডেই জাস্ট কস্ট বসায় বারবার আপডেট করছি! এই কোড তোমাকে সোর্স থেকে প্রতিটা নোডের শর্টেস্ট পাথ বের করে দিবে কিন্তু কমপ্লেক্সিটির দিক থেকে এটা খুবই বাজে! এজন্য আমদের লাগবে একটা প্রয়োরিটি কিউ।

বিএফএস এ আমরা যখন ১ নোড থেকে অনেকগুলো নোডে যাচ্ছি তখন সেই নোডগুলো থেকে আবার নতুন করে কাজ করার সময় “আগে আসলে আগে পাবেন” ভিত্তিতে কাজ করছি। যেমন উপরের গ্রাফে ১ থেকে আগে ৩ নম্বর নোডে এবং তারপর ২ নম্বর নোডে এ গেলে আগে ৩ নিয়ে কাজ করছি, এবপর ২ নিয়ে কাজ করছি।

ভালো করে দেখো এখানে কি সমস্যাটা হচ্ছে। ৩ নিয়ে আগে কাজ করলে আমরা ৪ এর ডিস্টেন্সকে আপডেট করে দিচ্ছি ডিস্টেন্স  $5 + 3 = 8$  হিসাবে। পরবর্তীতে যখন ২ দিয়ে ৩ কে আবার আপডেট করা হচ্ছে তখন ৩ এর ডিস্টেন্স হয়ে গিয়েছে ৩, এবার ৪ এর ডিস্টেন্সকে আবার আপডেট করছি  $3 + 3 = 6$  হিসাবে। ৪ কে মোট দুইবার আপডেট করা লাগলো।

বিজ্ঞানী ডায়াক্লট্রা চিত্তা করলেন যদি এই “আগে আসলে আগে পাবেন” ভিত্তিতে কাজ না করে সবথেকে কাছের নোডগুলোকে আগে প্রসেস করি তাহলে অনেক কমবার আপডেট করা লাগে। আমরা যদি ২ কে নিয়ে আগে কাজ করতাম তাহলে ৩ আগেই আপডেট হয়ে যেত এবং ৪ কে একবার আপডেট করেই শর্টেস্ট ডিস্টেন্স পেয়ে যেতাম! একটু হাতে কলমে সিমুলেট করে দেখো। আইডিয়াটা হলো যেকোনো সময় কিউ তে যতগুলো নোড থাকবে তাদের মধ্যে যেটা সোর্স থেকে সবথেকে কাছে সেটা নিয়ে আগে কাজ করবো। এজন্যই আমরা কিউ এর জায়গায় বসিয়ে দিবো একটি প্রয়োরিটি কিউ যে কিউতে নোড পুশ করার সাথে সাথে কাছের নোডটাকে সামনে এনে দিবো। পার্থক্য হলো আগে খালি নোড নাষ্টার পুশ করেছি, এখন বর্তমান ডিস্টেন্স অর্থাত  $d[u]$  এর মানটাও পুশ করতে হবে।

নিচে একটা সম্পূর্ণ ডায়াক্লট্রার সুড়েকোড দিলাম যেটা ১ থেকে  $n$  তম নোডে যাবার শর্টেস্ট পাথ বের করে এবং পাথটাও প্রিন্ট করে:

```

1 procedure dijkstra(G, source):
2     Q = priority_queue(), distance[] = infinity

```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্সিমাম ফ্লো-১

ম্যাক্সিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শর্টেস্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

```

3   Q.enqueue({distance[source], source})
4   distance[source]=0
5   while Q is not empty
6       u = nodes in Q with minimum distance[]
7       remove u from the Q
8       for all edges from u to v in G.adjacentEdges(v) do
9           if distance[u] + cost[u][v] < distance[v]
10          distance[v] = distance[u] + cost[u][v]
11          Q.enqueue({distance[v], v})
12      end if
13  end for
14 end while
15 Return distance

```

সি++ কোড দেখতে [ক্লিক করো এখানে](#)।

এখানে আগের সুড়োকোডের থেকে কয়েক জায়গায় একটু ভিন্নতা আছে। এখানে সাধারণ কিউ এর জায়গায় প্রায়োরিটি কিউ ব্যবহার করা হয়েছে। কিউ থেকে পপ হবার সময় তাই সোর্স থেকে এখন পর্ণ্ণ পাওয়া সরথেকে কাছের নোডটা পপ হচ্ছে এবং সেটা নিয়ে আগে কাজ করছি।

উপরের সুড়োকোডে সোর্স থেকে বাকি সব নোডের দূরত্ব বের করা হয়েছে। তুমি যদি শুধু একটা নোডের দূরত্ব বের করতে চাও তাহলে সেটা যখন কিউ থেকে পপ হবে তখনই রিটার্ন করে দিতে পারো।

নেগেটিভ এজ থাকলে কি ডায়াক্রস্ট্রু অ্যালগোরিদম কাজ করবে? যদি নেগেটিভ সাইকেল থাকে তাহলে ইনফিনিট লুপে পরে যাবে, বারবার আপডেট করে কষ্ট কমাতে থাকবে। যদি নেগেটিভ এজ থাকে কিন্তু সাইকেল না থাকে তাহলেও কাজ করবেনা। তবে তুমি যদি টার্গেট পপ হবার সাথে সাথে রিটার্ন করে না দাও তাহলে কাজ করবে কিন্তু সেটা তখন আর মূল ডায়াক্রস্ট্রু অ্যালগোরিদম থাকবেনা।

### কমপ্লেক্সিটি:

বিএফএস এর [কমপ্লেক্সিটি](#) ছিলো  $O(V + E)$  যেখানে  $V$  হলো নোড সংখ্যা আর  $E$  হলো এজ সংখ্যা। এখানেও আগের মতোই কাজ হবে তবে প্রায়োরিটি কিউ তে প্রতিবার সর্ট করতে  $O(\log V)$  কমপ্লেক্সিটি লাগবে। মোট:  $O(V \log V + E)$ ।

নেগেটিভ সাইকেল নিয়ে কাজ করতে হলে আমাদের জানতে হবে [বেলম্যান ফোর্ড অ্যালগোরিদম](#)। সেখানেও এজ রিল্যাক্স করে আপডেট করা হয়, একটা নোডকে সর্বোচ্চ  $n - 1$  বার আপডেট করা লাগতে পারে সেই প্রোপার্টি কাজে লাগানো হয়।

[পুরানো সিরিজ]

বিটমান্স ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

**ব্যকট্যাক্সিং(১):**

ব্যকট্যাক্সিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

**স্ট্রিং ম্যাচিং(২):**

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

[কোয়ান্টাম কম্পিউটার\(২\)](#)

ডায়াক্সেন্ট ভালো করে শিখতে নিচের প্রবলেমগুলো ঝটপট করে ফেলো:

Dijkstra?

Not the Best

New Traffic System

হ্যাপি কোডিং!

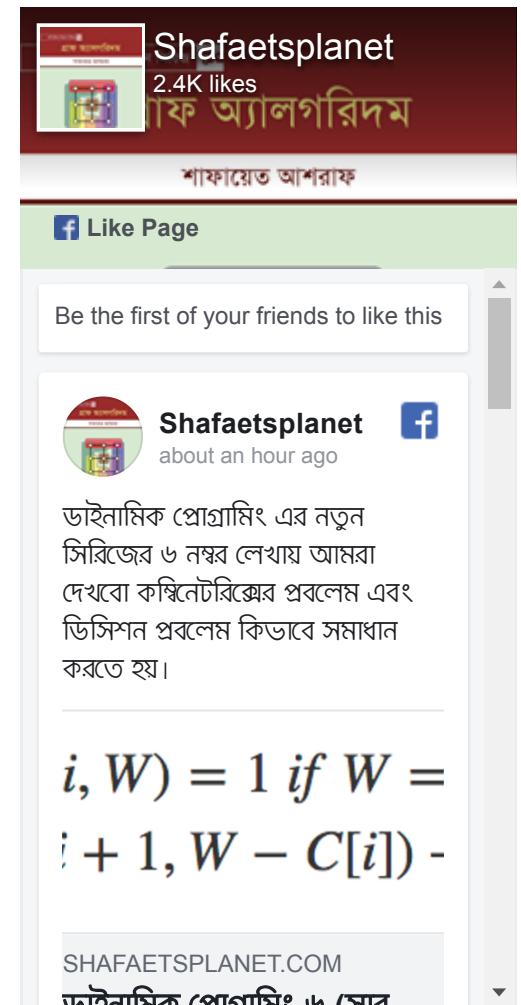
(গ্রাফ থিওরি নিয়ে সবগুলো লেখা)

ফেসবুকে মন্তব্য

2 comments

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



A screenshot of a Facebook post from the page "Shafaetsplanet". The post features a profile picture of a book titled "জ্ঞান পুরণ" (Gyan Puran) and the text "শাফায়েত আশরাফ" (Shafayet Ashraff). It has 2.4K likes. A call-to-action button says "Like Page". Below the post, there's a comment section with a placeholder "Be the first of your friends to like this". The main post content is a status update by Shafaetsplanet about an hour ago, which reads: "ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটারের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।" At the bottom right of the post area, there's a watermark for "SHAFAEETPLANET.COM" and "ডাইনামিক প্রোগ্রামিং ৬ / ম্যান" (Dynamic Programming 6 / Man).

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

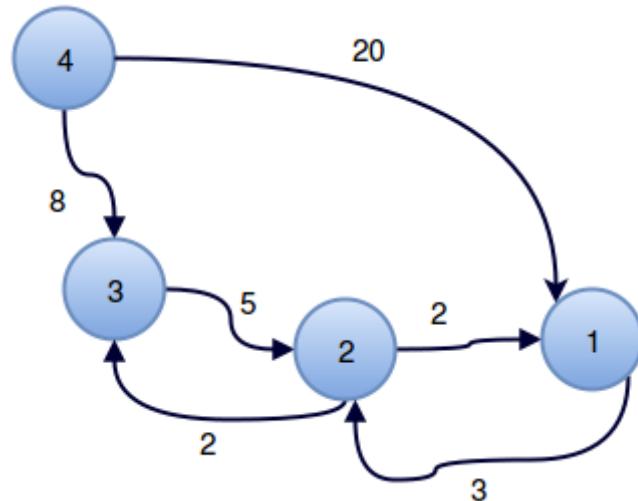
# গ্রাফ থিওরিতে হাতেখড়ি ১০: ফ্লয়েড ওয়ার্শল

জুলাই ২২, ২০১৪ by শাফায়েত



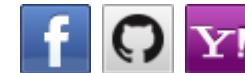
ফ্লয়েড ওয়ার্শল সম্বরত সব থেকে ছোট আকারের গ্রাফ অ্যালগরিদম, মাত্র ওলাইনে এটা লেখা যায়! তবে ও লাইনের এই অ্যালগরিদমেই বোঝার অনেক কিছু আছে। ফ্লয়েড ওয়ার্শলের কাজ হলো গ্রাফের প্রতিটা নোড থেকে অন্য সবগুলো নোডের সংক্ষিপ্তম দূরত্ব বের করা। এ ধরণের অ্যালগরিদমকে বলা হয় “অল-পেয়ার শর্টেস্ট পাথ” অ্যালগরিদম। এই লেখাটা পড়ার আগে [অ্যাডজেসেন্সি ম্যাট্রিক্স](#) সম্পর্কে জানতে হবে।

আমরা একটা গ্রাফের উপর কিছু সিমুলেশন করে সহজেই অ্যালগরিদমটা বুঝতে পারি। নিচের ছবিটা দেখ:



Node	1	2	3	4
1	0	3	inf	inf
2	2	0	2	inf
3	inf	5	0	inf
4	20	inf	8	0

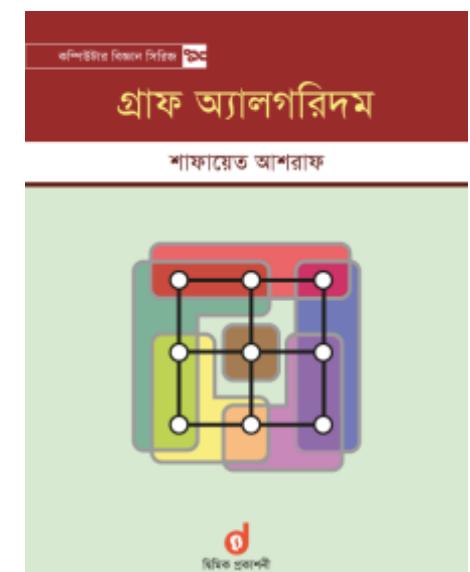
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

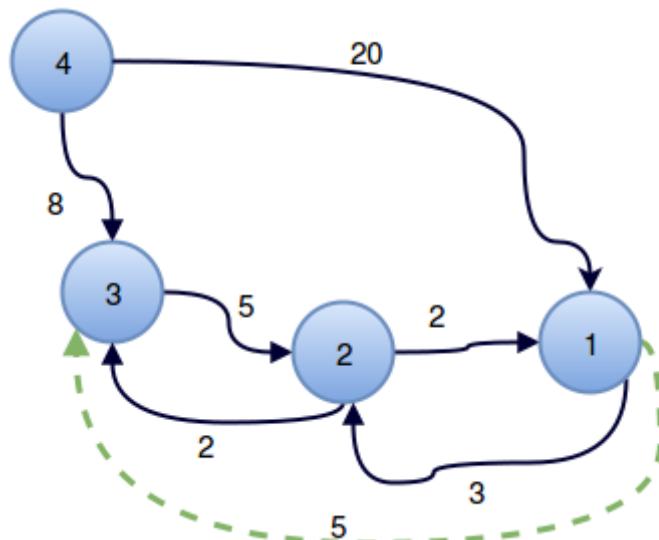
শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



ছবিতে চার নোডের একটা ওয়েটেড গ্রাফ দেখা যাচ্ছে। আর উপরে ডান কোনায় একটা ম্যাট্রিক্স। ম্যাট্রিক্সের  $u,v$  তম ঘরে বসানো হয়েছে  $u-v$  এজ এর ওয়েট বা কস্ট। যাদের মধ্যে সরাসরি এজ নেই সেসব ঘরে অসীম বা ইনফিনিটি বসিয়ে দেয়া হয়েছে। আর কোনাকুনি ঘরগুলোতে মান 0 কাবণ নিজের বাসা থেকে নিজের বাসাতেই যেতে কোন দূরত্ব অতিক্রম করতে হয় না!

এখন মনে করো “২” নম্বর নোডটাকে আমরা “মাঝের নোড” হিসাবে ধরলাম। মাঝের নোডকে আমরা বলবো  $k$ । তারমানে এখন  $k=2$ । (আমরা একে একে সব নোডকেই মাঝের নোড হিসাবে ধরবো, এটা যেকোন অর্ডারে করা যায়)

এখন যেকোন এক জোড়া নোড  $(i,j)$  নাও। ধরি  $i=1, v=3$ । আমরা চাই  $u$  থেকে  $v$  তে যেতে,  $k$  নোডটাকে মাঝে রেখে। তাহলে আমাদের  $i$  থেকে  $k$  তে যেতে হবে, তারপর  $k$  থেকে থেকে  $j$  তে যেতে হবে। কিন্তু লাভ না হলে আমরা এভাবে যাব কেন? আমরা  $k$  কে মাঝে রেখে যাবো কেবল যদি মেট কস্ট(cost) কমে যায়। ১ থেকে ৩ এর বর্তমান দূরত্ব  $\text{matrix}[1][3]=\text{ইনফিনিটি}$ । আর যদি  $k=2$  কে মাঝখানে রাখি তাহলে দূরত্ব দাঢ়াবে  $\text{matrix}[1][2] + \text{matrix}[2][3] = 3 + 2 = 5$ । তারমানে কস্ট কমে যাচ্ছে! আমরা গ্রাফটা আপডেট করে দিতে পারি এভাবে:



Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	inf	5	0	inf
4	20	inf	8	0



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইড্রিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

আমরা ২ কে “মাঝের মোড” হিসাবে ব্যবহার করে ১ থেকে ৩ এ গিয়েছি মোট ৫ কস্ট এ। গ্রাফে তাহলে ১ থেকে ৩ এ সরাসরি একটা এজ দিয়ে দিতে পারি ৫ কস্ট এ।

এখন আবার ধর  $i=2, j=4$ । আর আগের মতই  $k=2$ । এবার  $\text{matrix}[2][4]=\text{ইনফিনিটি}$ । এদিকে  $\text{matrix}[2][2] + \text{matrix}[2][4] = 0 + \text{ইনফিনিটি}$ । এবার কিন্তু দূরুত্ব কমলো না। তাই গ্রাফ আপডেট করার দরকার নেই। নিচয়ই বুঝতে পারছো আপডেটের শর্তটা হবে এরকম:

```
“ if(matrix[i][k] + matrix[k][j] < matrix[i][j])
    matrix[i][j] = matrix[i][k] + matrix[k][j]
```

অথবা আমরা একলাইনে লিখতে পারি:

```
“ matrix[i][j] = min(matrix[i][j] , matrix[i][k] + matrix[k][j])
```

এখন  $k=2$  স্থির রেখে আমরা  $i,j$  এর সবরকমের কমিনেশন নিবো। তুমি নিজেই চিন্তা করলে দেখবে  $k=2$  এর জন্য  $i=3, j=1$  এই কমিনেশনে আমরা আরেকটা নতুন এজ পাবো, বাকি গ্রাফ আগের মতই থাকবে।

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েন্ট সেট(ইউনিয়ন ফাইন্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

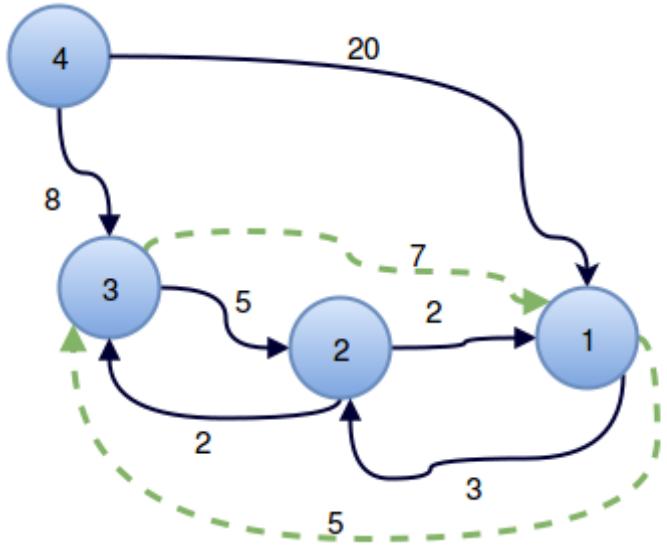
টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
সর্ট

ডায়াক্রস্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	7	5	0	inf
4	20	inf	8	0

এবার আমরা  $k=1$  কে মাঝের নোড হিসাবে চিত্য করি। মাথা খাটাতে চাইলে নিচে দেখার আগে নিজেই খাতায় একে ফেলতে পারো নতুন এজগুলো।

এবার একটা মাত্র এজ যোগ হবে।  $i=4, j=2$  হলে আমরা 8 থেকে 1 হয়ে 2 নম্বর নোডে যেতে পারি 20 কষ্ট এ। তাহলে গ্রাফটা হবে এরকম:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

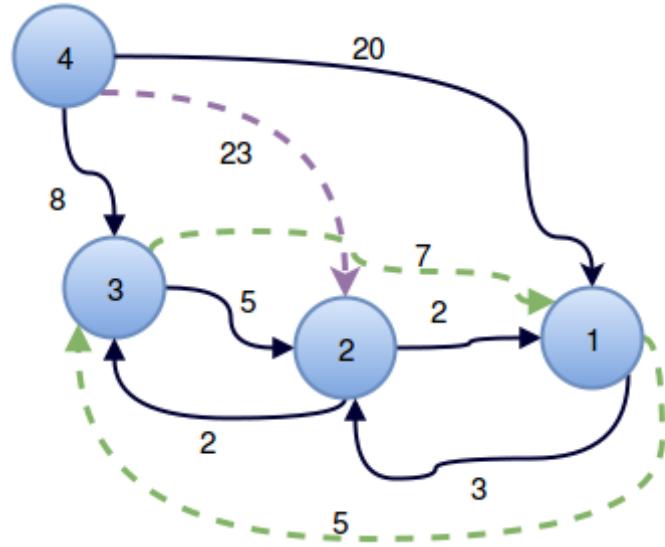
ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)



Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	7	5	0	inf
4	20	23	8	0

এবার ৩ নম্বর নোডকে মাঝের নোড হিসাবে ধরবো। আবারো নিজে চেষ্টা করে তারপর নিচের অংশ দেখ।

এবার কিছু মজার জিনিস ঘটবে। ৪ থেকে ১ এ আগে লাগছিল ২০ কস্ট। এখন ৩ কে মাঝে রেখে ৪ থেকে ১ এ গেলে লাগবে  $8+7=15$  কস্ট।  
লক্ষ্য কর একদম শুরুতে ৩ থেকে ১ এ আমাদের এজ ছিল না। কিন্তু আপডেট করার সময় আমরা এজ বসিয়ে দিয়েছি, এখন ৩ থেকে ১ এ যদিও সরাসরি চলে যাচ্ছি, মূল গ্রাফে আসলে  $3 \rightarrow 2 \rightarrow 1$  পথে যাচ্ছি।  
একই ভাবে ৪ থেকে ২ এর ২০ কস্ট এর পথটা আপডেট হয়ে ১৩ হয়ে যাবে।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

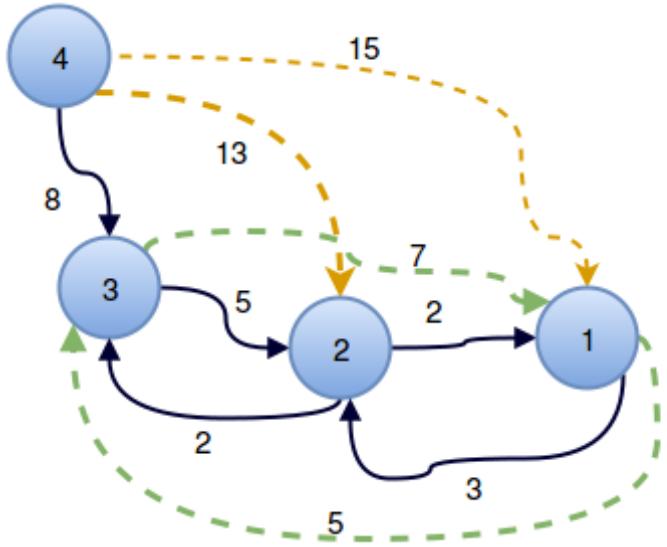
অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	7	5	0	inf
4	15	13	8	0

$k=8$  এর জন্য আবরণ কোন নোড থেকে 8 এ যাওয়া যায় না।

এখন আমরা ম্যাট্রিক্স দেখেই বলে দিতে পারছি কোন নোড থেকে কোন নোডে কত কস্ট এ যাওয়া যায়। ইনফিনিটি থাকা মানে সেই নোডে যাবার পথ নেই।

ইনপুট থেকে অ্যাডজেসেন্সি ম্যাট্রিক্স বানাবার পর তাহলে কাজ খুবই সহজ:

```

1 for k from 1 to |V|
2   for i from 1 to |V| \\\ |V| = number of nodes
3     for j from 1 to |V|
4       if matrix[i][j] > matrix[i][k] + matrix[k][j]
5         matrix[i][j] ← matrix[i][k] + matrix[k][j]
```

কোডে আমরা  $k$  এর লুপটা 1 থেকে চালাচ্ছি যদিও উদাহরণে আগে 2 নিয়েছি। এটা আসলে যেকোন অর্ডারেই করা যায়, তুমি আগে 1 নিলেও দেখবে অ্যালগরিদম কাজ করবে।

এভাবেতো আমরা শুধু পাথের কস্ট পেলাম, পাথটা কিভাবে পাব?

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM  
দাইনামিক প্রোগ্রামিং ৬ / ম্যাচ

ধরো আমাদের একটা ম্যাট্রিক্স আছে  $\text{next}[][]$ । এখন  $\text{next}[i][j]$  দিয়ে আমরা বুঝি  $i$  থেকে  $j$  তে যেতে হলে পরবর্তি যে নোড এ যেতে হবে সেই নোডটা। তাহলে একদম শুরুতে সব  $i, j$  এর জন্য  $\text{next}[i][j] = j$  হবে। কারণ শুরুতে কোন “মাঝের নোড” নেই এবং তখনও শর্টেস্ট পাথ বের করা শেষ হয় নি।

এখন আমরা যখন  $\text{matrix}[i][j]$  আপডেট করবো লুপের সেটার মানে হলো মাঝে একটা নোড  $k$  ব্যবহার করে আমরা যাবো। লক্ষ্য কর আমরা কিন্তু মূল গ্রাফে সরাসরি এজ দিয়ে  $i$  থেকে  $k$  তে নাও যেতে পারি, আমরা শুধু জানি  $i, j$  নোড দুটোর মাঝে একটা নোড  $k$  আছে যেখানে আমাদের যেতে হবে  $j$  তে যাবার আগে।  $i$  থেকে  $k$  তে যাবার পথে পরবর্তি যে নোডে যেতে হবে সেটা রাখা আছে  $\text{next}[i][k]$  তে! তাহলে  $\text{next}[i][j] = \text{next}[i][k]$  হয়ে যাবে।

```
1 for k from 1 to |V|
2     for i from 1 to |V|
3         for j from 1 to |V|
4             if matrix[i][k] + matrix[k][j] < matrix[i][j] then
5                 matrix[i][j] ← matrix[i][k] + matrix[k][j]
6                 next[i][j] ← next[i][k]
7
8
9 findPath(i, j)
10    path = [i]
11    while i ≠ j
12        i ← next[i][j]
13        path.append(i)
14    return path
```

`findpath` ফাংশনে আমরা  $j$  কে ফিল্ড রেখে `next` অ্যাবে ধরে আগাছি যতক্ষণ না  $j$  তে পৌছাচ্ছি। তাহলে আমরা পেয়ে গেলাম পাথ!

## ট্রান্সিটিভ ক্লোজুর(Transitive Closure):

ধরো আমাদের অ্যাডজেন্সি ম্যাট্রিক্সটা এরকম:

`matrix[i][j] = 1` যদি  $i$  থেকে  $j$  তে সরাসরি এজ থাকে

`matrix[i][j] = 0` যদি এজ না থাকে

এখন আমরা এমন একটা ম্যাট্রিক্স তৈরি করতে চাই যেটা দেখে বলে দেয়া যাবে  $i$  থেকে  $j$  তে এক বা একাধিক এজ ব্যবহার করে যাওয়া যায় কিনা।

আমরা চাইলে উপরের মত করে 0 এর জায়গায় ইনফিনিটি দিয়ে শর্টেস্ট পাথ বের করে কাজটা করতে পারতাম। কিন্তু এক্ষেত্রে “OR” আর “AND”

অপারেশন ব্যবহার আরো দ্রুত কাজটা করা যায়। এখন আপডেটের শর্টটা হয়ে যাবে এরকম:

```
matrix[i][j] = matrix[i][j] || (matrix[i][k] && matrix[k][j])
```

এটার মানে  $\text{matrix}[i][j]$  তে তখনই 1 বসবে যখন হ্য “ $\text{matrix}[i][j]$  তে 1 আছে” অথবা “ $\text{matrix}[i][k]$  এবং  $\text{matrix}[k][j]$ ” দুটোতেই 1

আছে। তারমানে হ্য সরাসরি যেতে হবে অথবা মাঝে একটা নোড  $k$  ব্যবহার করে যেতে হবে।

### কমপ্লেক্সিটি:

তটা নেস্টেড লুপ ঘুরছে নোড সংখ্যার উপর, টাইম কমপ্লেক্সিটি  $O(n^3)$ । ২ডি ম্যাট্রিক্স ব্যবহার করায় স্পেস কমপ্লেক্সিটি  $O(n^2)$ ।

### কিছু প্রশ্ন:

১.  $k$  এর লুপটা কি  $i,j$  লুপের ভিতর দিলে অ্যালগোরিদম কাজ করত?

২. গ্রাফে নেগেটিভ কস্ট থাকলে ফ্রয়েড ওয়ার্শল কাজ করবে কি?

### রিলেটেড প্রবলেম:

Page Hopping

05-2 Rendezvous

Minimum Transport Cost

Asterix and Obelix

আরো অনেক প্রবলেম

হ্যাপি কোডিং!

# গ্রাফ থিওরিতে হাতেখড়ি ১১: বেলম্যান ফোর্ড

অক্টোবর ১৭, ২০১৮ by শাফায়েত



বেলম্যান ফোর্ড গ্রাফে শর্টেস্ট পাথ বের করার একটা অ্যালগোরিদম। এই অ্যালগোরিদম একটা নোডকে সোর্স ধরে সেখান থেকে সব নোডের সংক্ষিপ্তম বা শর্টেস্ট পথ বের করতে পারে। আমরা একদম শুরুতে এই কাজ করার জন্য ব্রেডথ ফার্স্ট সার্চ শিখেছি। কিন্তু **বিএফএস(BFS)** যেহেতু ওয়েটেড গ্রাফে কাজ করে না তাই এরপর আমরা শিখেছি ডায়াক্রস্ট্রো অ্যালগোরিদম। এখন বেলম্যান ফোর্ড শিখব কারণ আগের কোনো অ্যালগোরিদমই নেগেটিভ ওয়েট এর এজ আছে এমন গ্রাফে কাজ করে না।

আমরা **ডায়াক্রস্ট্রো** শেখার সময় রিল্যাক্সেশন নামের একটা ব্যাপার শিখেছিলাম। তোমার যদি মনে না থাকে বা ডায়াক্রস্ট্রো না শিখে থাকো তাহলে আমরা প্রথমে একটু ঝালাই করে নেই আরেকবার। মনে থাকলে পরের অংশটা বাদ দিয়ে সরাসরি **এখানে যেতে পার**।

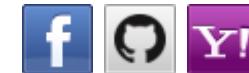
## এজ রিল্যাক্সেশন:

ধর একটা গ্রাফে সোর্স থেকে প্রতিটা নোডের ডিস্টেন্স/কস্ট রাখা হয়েছে  $d[]$  অ্যাবেতে। যেমন  $d[3]$  মানে হলো সোর্স থেকে বিভিন্ন এজ পার হয়ে 3 নম্বর নোড এ আসতে মোট  $d[3]$  ডিস্টেন্স পার করতে হয়েছে। যদি ডিস্টেন্স জানা না থাকে তাহলে ইনফিনিটি অর্থাৎ অনেক বড় একটা মান রেখে দিবো। আর  $cost[u][v]$  তে রাখা আছে  $u - v$  এজ এর cost।

ধর তুমি বিভিন্ন জায়গা ঘুরে ফার্মগেট থেকে টিএসসি তে গেলে 10 মিনিটে, আবার ফার্মগেট থেকে কার্জন হলে গেলে 25 মিনিটে। তাহলে ফার্মগেটকে সোর্স ধরে আমরা বলতে পারি:

$$d[\text{টিএসসি}] = 10, d[\text{কার্জন হল}] = 25$$

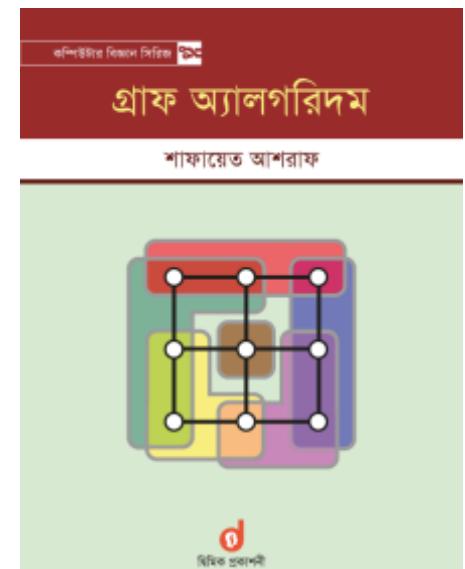
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @ **Traveloka Singapore** (বিস্তারিত...)



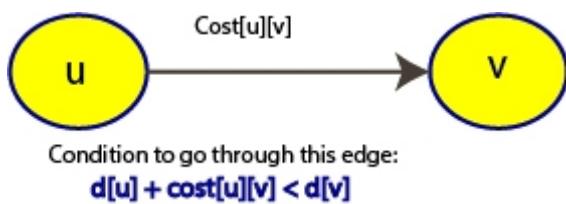
এখন তুমি দেখলে টিএসসি থেকে ৭ মিনিটে কার্জনে চলে যাওয়া যায়,

“  
“ cost[টিএসসি][কার্জন হল] = ৭

তাহলে তুমি ২৫ মিনিটের জায়গায় মাত্র  $10 + 7 = 17$  মিনিটে কার্জ নহলে যেতে পারবে। যেহেতু তুমি দেখেছো:

“  $d[\text{টিএসসি}] + \text{cost}[\text{টিএসসি}][\text{কার্জন হল}] < d[\text{কার্জন হল}]$

তাই তুমি এই নতুন বাস্তা দিয়ে কার্জন হলে গিয়ে  $d[\text{কার্জন হল}] = d[\text{টিএসসি}] + \text{cost}[\text{টিএসসি}][\text{কার্জন হল}]$  বানিয়ে দিতেই পারো!!



উপরের ছবিটা সেটাই বলছে। আমরা  $u$  থেকে  $v$  তে যাবো যদি  $d[u] + \text{cost}[u][v] < d[v]$  হয়। আর  $d[v]$  কে আপডেট করে  $d[v] = d[u] + \text{cost}[u][v]$  বানিয়ে দিবো। তবিষ্যতে যদি কার্জনহলে অন্য বাস্তা দিয়ে আরো কম সময়ে যেতে পারি তখন সেই বাস্তা এভাবে কম্পেয়ার করে আপডেট করি দিবো। ব্যাপারটা অনেকটা এরকম:

```
1 if(d[u]+cost[u][v] < d[v])
2 d[v] = d[u] + cost[u][v];
```

এটাই হলো এজ বিল্যাক্সেশন। এখন আমরা বেলম্যান ফোর্ড শেখার জন্য তৈরি।

বেলম্যান ফোর্ড



প্রোগ্রামিং কনটেন্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

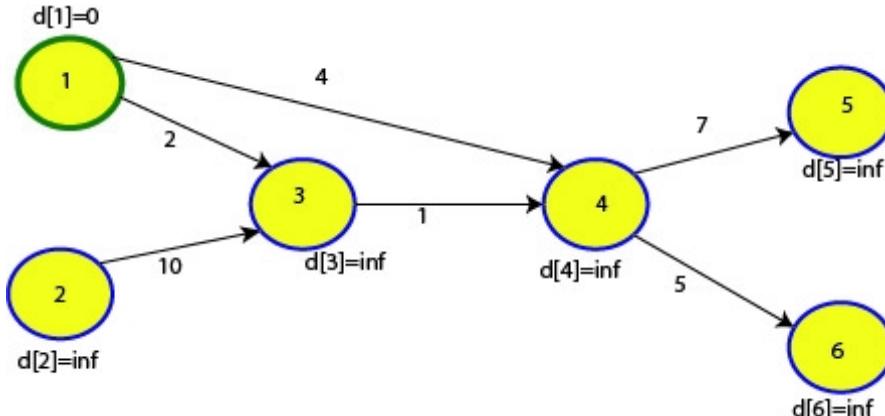
ডাটা স্টোকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

নিচের গ্রাফে আমরা ১ থেকে শুরু করে প্রতিটা নোডে যাবার শর্টেস্ট পাথ বের করতে চাই:



শুরুতে  $d[1] = 0$  কারণ ১ হলো সোর্স। বাকিসবগুলোতে ইনফিনিটি রেখেছি কারণ আমরা এখনও জানিনা শর্টেস্ট পাথের কষ্ট কত।

তুমি এজ রিল্যাক্স কিভাবে করতে হয় এরই মধ্যে শিখে গেছ। এখন কাজ হলো সবগুলো এজকে একবার করে রিল্যাক্স করা, যেকোন অর্ডারে।  
একবার ‘গ্রাফ রিল্যাক্স’ করার মানে হল গ্রাফটার সবগুলো এজকে একবার করে রিল্যাক্স করা। আমি নিচের অর্ডারে রিল্যাক্স করতে চাই,

Serial	1	2	3	4	5	6
Edge	4 -> 5	3 -> 4	1 -> 3	1 -> 4	4 -> 6	2 -> 3

তুমি চাইলে অন্য যেকোনো অর্ডারেও এজগুলো নিতে পারতে। এখন চিন্তা কর এজগুলোকে একবার রিল্যাক্স করলে আমরা  $d[]$  অ্যারেতে কি পাব? সোর্স থেকে শুরু করে সর্বোচ্চ ১টা এজ ব্যবহার করে অন্যান্য নোডে যাবার শর্টেস্ট পাথের কষ্ট আমরা পেয়ে যাব। উপরের ছবিতে রিল্যাক্স করার পর  $d[]$  এর মানগুলো আপডেট করে দাও। করার পর ছবিটা নিচের মত হবার কথা:

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

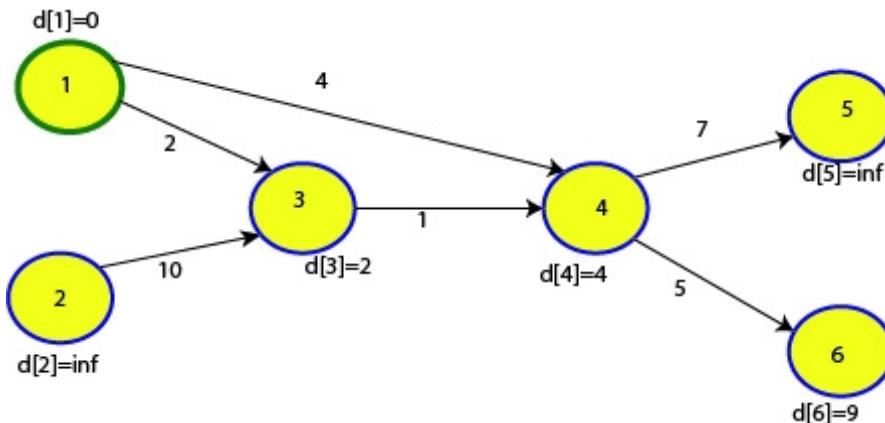
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্স্ট্রা

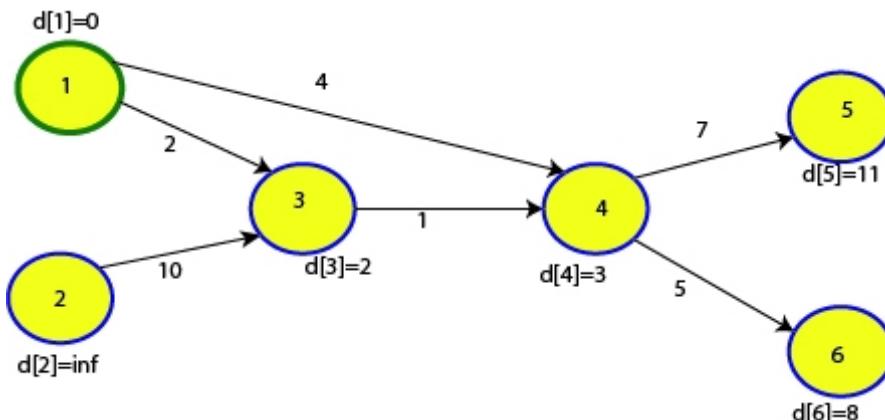
ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



এজ রিল্যাক্স করার সময় কিছু নোড এর কস্ট আপডেট করতে পারি নি কারণ  $d[u] + \text{cost}[u][v] < d[v]$  শর্তটা পূরণ করে নি। বাকি এজগুলো আপডেট করার পর  $d[]$  অ্যারের এর মান উপরের ছবির মত পেয়েছি। ১ নম্বর নোড থেকে শুরু করে সর্বোচ্চ একটি এজ ব্যবহার করে সব নোডে যাবার শর্টেস্ট পাথ এখন আমরা জানি!

এখন সর্বোচ্চ ২টা এজ ব্যবহার করে সব নোডে যাবার শর্টেস্ট পাথের cost বের করতে আরেকবার রিল্যাক্স করে ফেলি! আবারো যেকোন অর্ডারে করা যাবে, তবে প্রথমে যে অর্ডারে করেছি সেভাবেই প্রতিবার করা কোড লেখার সময় সুবিধাজনক।



আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কডার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শর্টেস্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

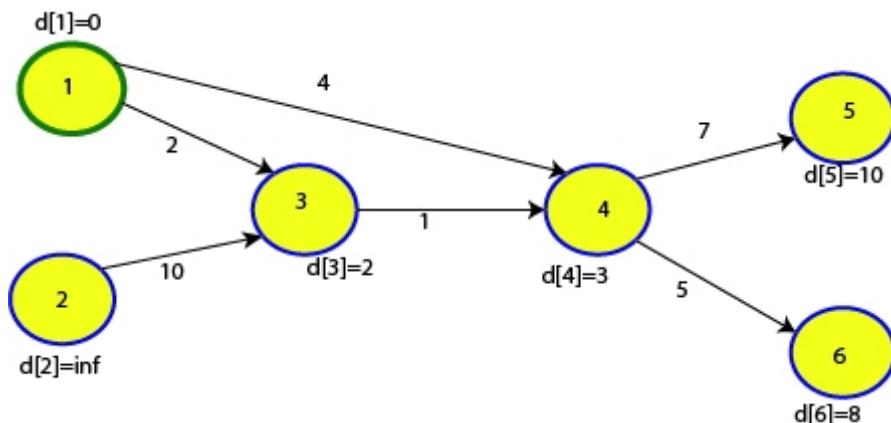
ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিনেটরিঙ্গ, ডিসিশন প্রবলেম)

একটা ব্যাপার লক্ষ্য কর, ১ থেকে ৬ তে যাবার শর্টেস্ট পথে তুমি এজ আছে ( $1 \rightarrow 3$ ,  $3 \rightarrow 8$ ,  $8 \rightarrow 6$ ) এবং পথের দৈর্ঘ্য  $2+1+5=8$ । মাত্র ২বার রিল্যাক্স করলেও আমরা এখনই  $d[6]$  তে ৮ পেয়ে গেছি, অর্থাৎ আমাদের এখন সর্বোচ্চ ২টা এজ ব্যবহার করে শর্টেস্ট পাথের cost পাবার কথা। এটা নির্ভর করে তুমি কোন অর্ডারে এজ রিল্যাক্স করেছ তার উপর। সে কারণে ৫ এ যাবার শর্টেস্ট পাথ ১০ হলেও  $d[5]$  এ এখনো ১০ পাইনি। **X** বার ‘গ্রাফ রিল্যাক্স’ করলে সর্বোচ্চ **X** টা এজ ব্যবহার করে সোর্স প্রতিটা নোডে যাবার শর্টেস্ট পাথ তুমি নিশ্চিত ভাবে পাবে। **X** এর থেকে বেশি এজের ব্যবহার করে প্রতিটা নোডে যাবার শর্টেস্ট পাথ তুমি **X** বার গ্রাফ রিল্যাক্সের পর পেতেও পাব, নাও পেতে পাব, সেটা এজ এর অর্ডারের উপর নির্ভর করে।

এখন ৩য় বারের মত রিল্যাক্স করি:



এবার শুধু মাত্র ৫ নম্বর নোড আপডেট হবে।

এরপরে আমরা আর যতই আপডেট করি,  $d[]$  অ্যারেতে কোনো পরিবর্তন হবে না, আমরা ১ থেকে প্রতিটা নোডে যাবার শর্টেস্ট পাথ পেয়ে গিয়েছি।

এখন স্বাভাবিকভাবেই প্রশ্ন আসবে যে রিল্যাক্স কয়বার করতে হবে? গ্রাফে যদি নোড  $n$  টা থাকে তাহলে এক নোড থেকে অন্য নোডে যেতে সর্বোচ্চ  $n - 1$  টা এজ ব্যবহার করতে হবে। তারমানে কোনো নোড সর্বোচ্চ  $n - 1$  বার আপডেট হতে পারে। তাই রিল্যাক্স করার লুপটাও চালাতে হবে  $n - 1$  বার। তবে আমরা যেরকম উপরের গ্রাফে দেখেছি ৩বারের পরেই আর কোন নোড আপডেট করা যাচ্ছে না, সেরকম হলে আর নতুন করে রিল্যাক্স করার দরকার নাই।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

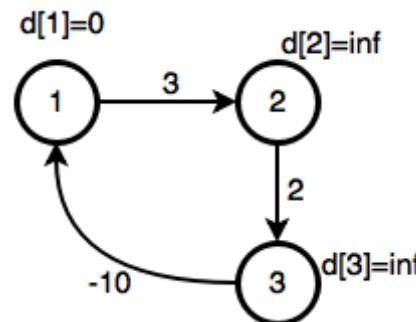
ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

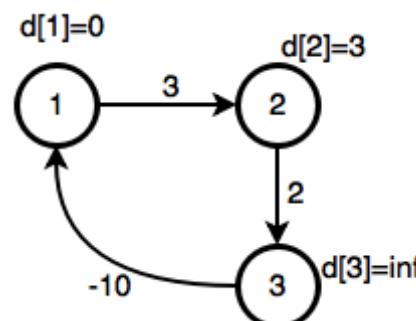
কোয়ান্টাম কম্পিউটার(২)

এখন নিচের মাত্র তিনোভের গ্রাফটায় বেলম্যান ফোর্ড অ্যালগরিদম চালাও, অর্থাৎ যতক্ষণ কোন নোড আপডেট করা যায় ততক্ষণ পুরো গ্রাফটা রিল্যাঞ্চ কর:



Serial	1	2	3
Edge	2 -> 3	1 -> 2	3 -> 1

প্রথমবার রিল্যাঞ্চ করার পর পার:



২যবার করার পর:

কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

**Shafaetsplanet**  
 2.4K likes  
 শাফায়েত আশরাফ

---

Like Page
Be the first of your friends to like this

---

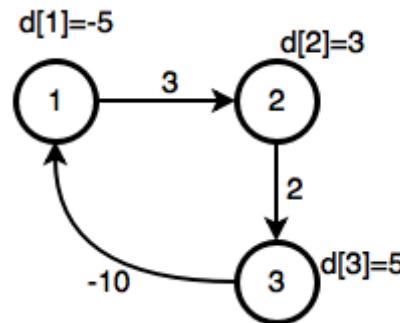
**Shafaetsplanet**  
 about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিনের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

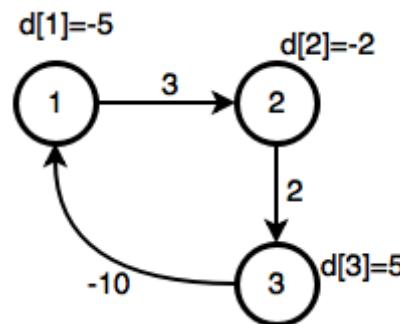
---

i, W) = 1 if W =
i + 1, W - C[i]) -

SHAFAEPLANET.COM
ডাইনামিক প্রোগ্রামিং ৬ / মার



3 নোডের গ্রাফে সোর্স থেকে কোনো নোডে শর্টেস্ট যেতে ২টার বেশি এজ লাগবে না, তয় বার রিল্যাক্স করার চেষ্টা করলে কোনো নোড আপডেট হবার কথা না। কিন্তু এই গ্রাফে আপডেট হচ্ছে:



এটা হচ্ছে কারণ  $1 \rightarrow 2 \rightarrow 3 \rightarrow 1$  সাইকেলটার মোট ওয়েট নেগেটিভ ( $3 + 2 - 10 = -5$ )। তাই তুমি যতবার এই সাইকেলে ঘূরবে শর্টেস্ট পাথ তত ছোট হতে থাকবে। তাই নেগেটিভ সাইকেল থাকলে এবং সোর্স থেকে সেই নেগেটিভ সাইকেলে যাবার পথ থাকলে সোর্সের শর্টেস্ট পাথ আনডিফাইনড বা অসংজ্ঞায়িত। যদি  $n - 1$  বার গ্রাফ রিল্যাক্স করার পর দেখি যে  $n$  তম বারও কোনো নোডের cost আপডেট করা যায় তখন বুঝতে হবে আমরা নেগেটিভ সাইকেলে গিয়ে পরেছি, শর্টেস্ট পাথ বের করা সম্ভব না।

আমাদের সুভোকোড তাহলে হবে এরকম:

```

Input: A non-empty connected weighted graph G with vertices G.V and edges G.E
procedure Bellman_Ford(G,source):
1 Let distance[] ← infinity
2 Let N ← number of nodes
3 distance[source] = 0
4 for step from 1 to N-1
    for all edges from (u,v) in G.E
        if distance[u] + cost[u][v] < distance[v]
            distance[v] = distance[u] + cost[u][v]
        end if
    end for
10 end for
11 for all edges from (u,v) in G.E
    if distance[u] + cost[u][v] < distance[v]
13     return "Negative cycle detected"
14     end if
15 end for
16 return distance

```

পাথ প্রিন্ট করা:

বিএফএস বা ডায়াক্স্ট্রাতে যেভাবে পাথ প্রিন্ট করে ঠিক সেভাবে এখনেও পাথ প্রিন্ট করা যাবে। *previous[]* নামের একটা অ্যারে নাও।

*previous[v] = u* মানে হলো *v* তম নোডে তুমি *u* থেকে এসেছ। শুরুতে অ্যারেতে ইনফিনিটি থাকবে। *u -> v* এজটা রিল্যাক্স করার

সময় *previous[v] = u* করে দাও। এখন তুমি *previous* অ্যারে দেখে সোর্স থেকে যেকোন নোডের পাথ বের করে ফেলতে পারবে।

কমপ্লেক্সিটি:

সর্বোচ্চ  $n - 1$  বার প্রতিটা এজকে রিল্যাক্স করতে হবে, টাইম কমপ্লেক্সিটি  $O(n * e)$ ।

চিন্তা করার জন্য কিছু প্রবলেম:

তোমাকে একটা গ্রাফ দিয়ে বলা হল শর্টেস্ট পাথে সর্বোচ্চ  $x$  টা এজ থাকতে পারে। এবাব কিভাবে শর্টেস্ট পাথ বের করবে? (UVA 11280)

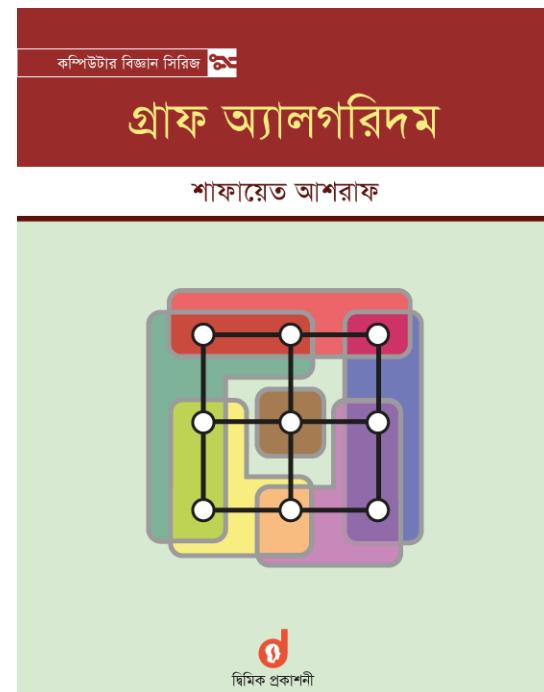
একটি গ্রাফে কোন কোন নোড নেগেটিভ সাইকেলের অংশ কিভাবে বের করবে? (হিটস: স্ট্রংলি কানেক্টেড কম্পোনেন্ট + বেলম্যান ফোর্ড)

রিলেটেড কিছু প্রবলেম:

English Blog

# গ্রাফ থিওরিতে হাতেখড়ি ১৪ – স্টুংলি কানেক্টেড কম্পোনেন্ট

নভেম্বর ২৯, ২০১৫ by শাফায়েত



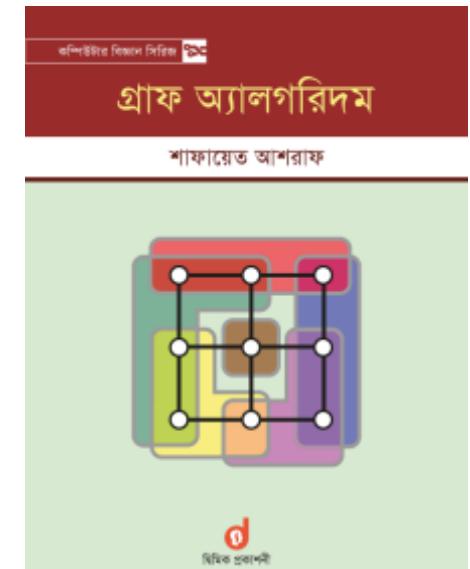
সাবস্ক্রাইব



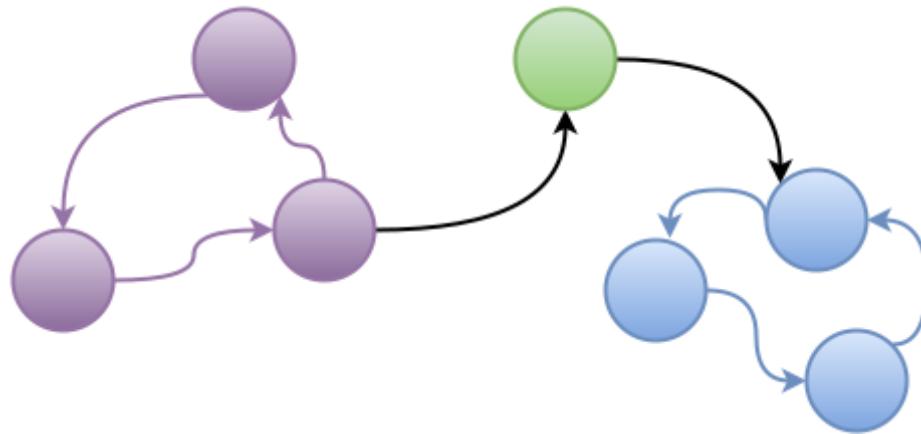
Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)

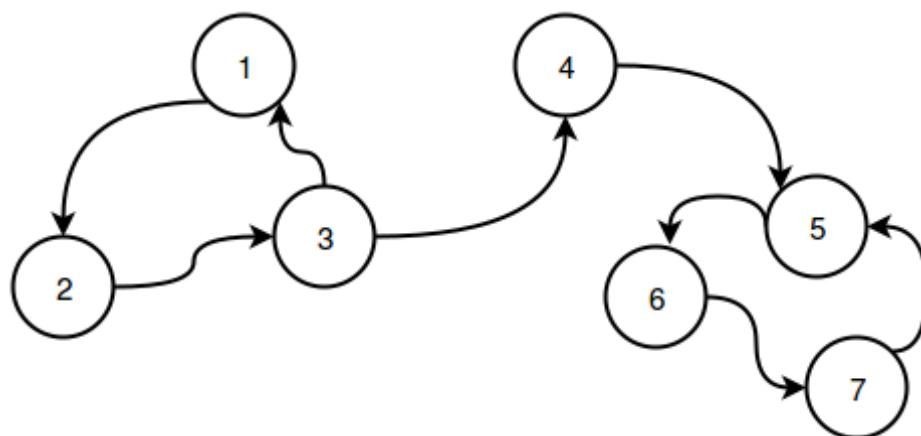


একটা ডিরেক্ট গ্রাফের স্ট্রংলি কানেক্টেড কম্পোনেন্ট বা SCC হলো এমন একটা কম্পোনেন্ট যার প্রতিটা নোড থেকে অন্য নোডে যাবার পথ আছে। নিচের ছবিতে একটা গ্রাফের প্রতিটা স্ট্রংলি কানেক্টেড কম্পোনেন্ট আলাদা রঙ দিয়ে দেখানো হয়েছে।



ডেপথ ফার্স্ট সার্চ এর ফিনিশিং টাইমের ধারণা ব্যবহার করে আমরা  $O(V + E)$  তে একটা গ্রাফের স্ট্রংলি কানেক্টেড কম্পোনেন্ট গুলোকে আলাদা করে ফেলতে পারি। এই লেখাটা পড়ার আগে অবশ্যই টপলোজিকাল সার্টিং আর ডেপথ ফার্স্ট সার্চ এর ডিসকভারি এবং ফিনিশিং টাইম সম্পর্কে ধারণা থাকতে হবে।

নিচের গ্রাফটা দেখ:



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

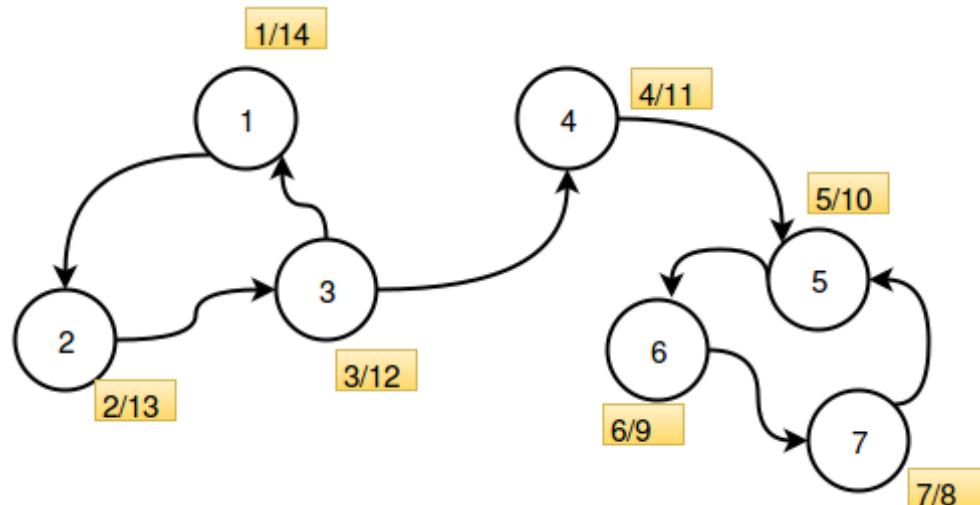
স্ট্যাক

কিউ+সার্কুলার কিউ

প্রথমেই একটা ভুল পদ্ধতিতে অনেকে SCC বের করার চেষ্টা করে। সেটা হলো যেকোনো নোড থেকে ডিএফএস চালিয়ে যেসব নোডে যাওয়া যায় তাদেরকে একটা কম্পোনেন্ট হিসাবে ধরা। কিন্তু খুব সহজেই বোধ যায় এটা কাজ করবে না, উপরের গ্রাফে ১ থেকে ডিএফএস চালালে সবগুলো নোড ভিজিট করা যাবে, কিন্তু ১ থেকে ৪ এ যাওয়া গেলেও ৪ থেকে ১ এ যাবার কোনো পথ নেই, তাই এরা একই কানেক্টেড কম্পোনেন্ট এর অংশ না। এই পদ্ধতিতে সমস্যা হলো ডিএফএস কানেক্টেড কম্পোনেন্ট থেকে বের হয়ে আন্য কম্পোনেন্ট এ চলে যায়। এই সমস্যা সমাধান করতে আমরা একটু বুদ্ধিমানের মত ডিএফএস চালাবো।

দুটি নোড  $u, v$  একই SCC তে থাকবে শুধুমাত্র যদি  $u$  থেকে  $v$  তে যাবার পথ থাকে এবং  $v$  থেকে  $u$  তে যাবারও পথ থাকে।

প্রথমে আমরা ১ থেকে ডিএফএস চালিয়ে সবগুলো নোডের ডিসকভারি টাইম আর ফিনিশিং টাইম লিখে ফেলি। নোডগুলো ১,২,৩,৪,৫,৬,৭ অর্ডারে ভিজিট করলে আমরা নিচের ছবির মত স্টার্টিং/ফিনিশিং টাইম পাবো:



এখন বোঝার সুবিধার জন্য স্ট্রাইলি কানেক্টেড কম্পোনেন্টের সবগুলো নোডকে একটা বড় নোড মনে করি:

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েন্ট সেট(ইউনিয়ন ফাইন্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কম্প্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

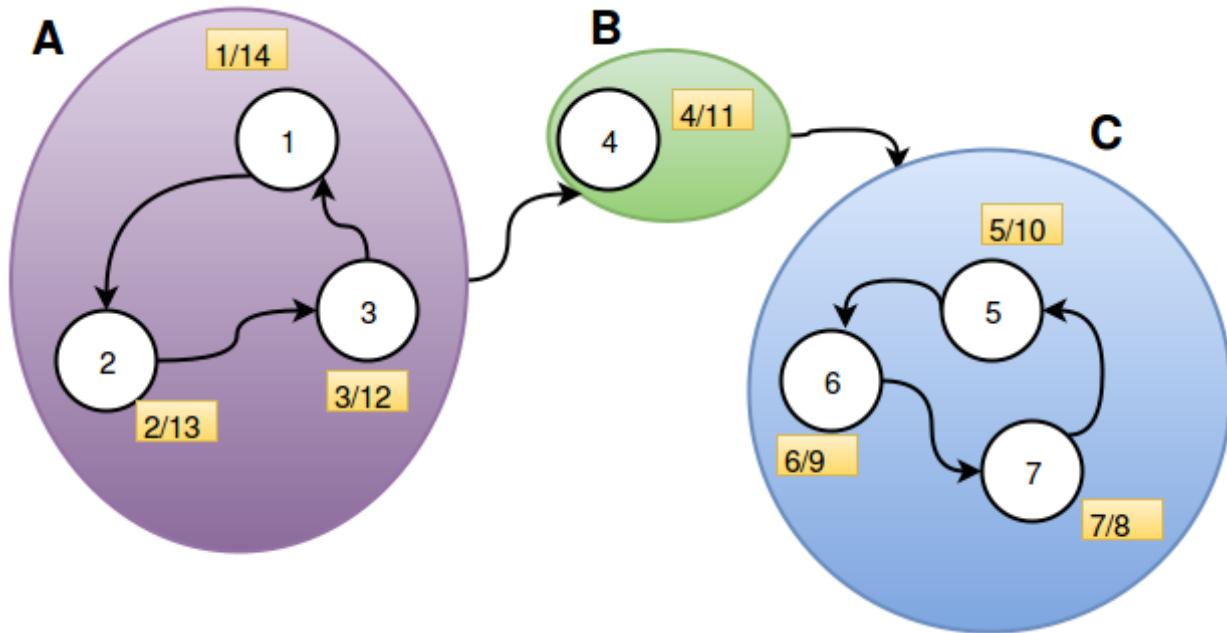
টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



লক্ষ্য করো, গ্রাফটাকে এভাবে ‘ডিকম্পোজ’ করার পর গ্রাফটিতে আর কোনো সাইকেল থাকা সম্ভব না, অর্থাৎ গ্রাফটি একটি ড্যাগ বা ডিবেটেড অ্যাসাইন্স গ্রাফে পরিণত হয়েছে। এখন বড় নোডগুলোকে সহজেই টপোলজিকাল অর্ডারে সাজানো সম্ভব, অর্ডারটা হবে A,B,C।

এখন লক্ষ্য করো ড্যাগে একটা নোড D1 থেকে অন্য নোড D2 এ যাওয়া যায় তাহলে D1 টপোলজিকাল অর্ডারে D2 এর আগে অবশ্যই থাকবে। আবার আমরা আগেই জানি যে টপোলজিকাল অর্ডারে যে আগে থাকে তার ফিনিশিং টাইম বেশি হয় কারণ অন্যান্য সব নোডের কাজ শেষ করে ওই নোডে ফিরে আসতে হয়।

তাহলে D1 যদি টপোলজিকাল অর্ডারে D2 এর আগে থাকে তাহলে যেসব ছোটো ছোটো নোড নিয়ে D2 গঠিত হয়েছে তাদের সবার ফিনিশিং টাইম অবশ্যই D1 এর সব নোডের থেকে কম হবে।

এখন U থেকে V তে যাওয়া গেলেই তারা একই SCC এর অন্তর্ভুক্ত না, V থেকে U তে যাবার পথও থাকতে হবে। অথবা আমরা বলতে পারি ‘উল্টো-গ্রাফ’ এও U থেকে V তে যাবার পথ থাকতে হবে!

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রিংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

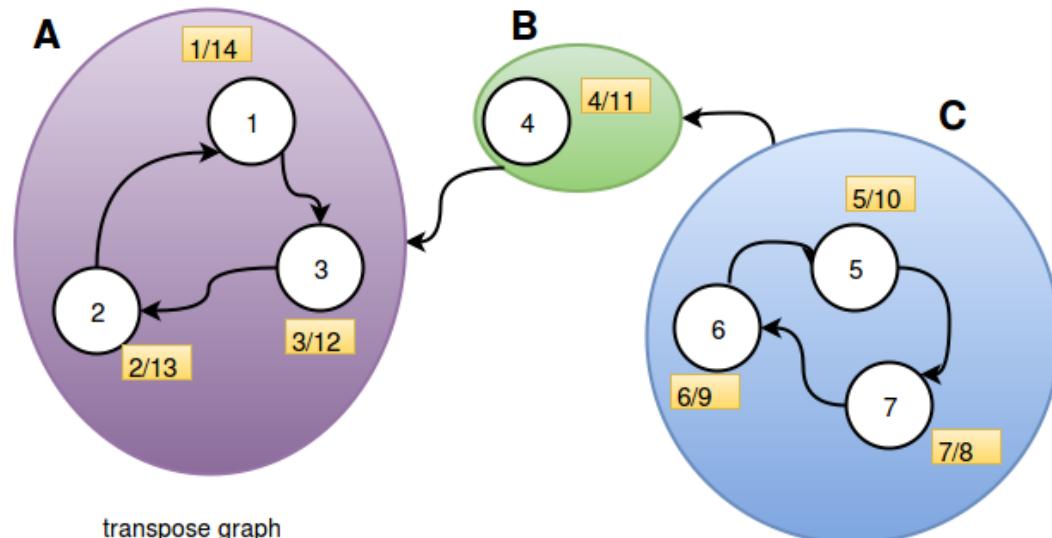
ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন

সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

যদি গ্রাফের এজগুলো উল্টে দেয়া হয়, তাহলেও SCC গুলো একই থাকে। একে বলা হয় ট্রান্সপোজ গ্রাফ(transpose) , ট্রান্সপোজ গ্রাফে সাইকেল গুলোর কোনো পরিবর্তন হয় না। মূল গ্রাফে যদি u-v একই SCC এর মধ্যে থাকে তাহলে তারা অবশ্যই একই সাইকেলের মধ্যে থাকবে। এটাই আমাদের অ্যালগোরিদমের মূল ভিত্তি।



উপরের ছবিতে আগের গ্রাফের এজগুলো উল্টে দেয়া হয়েছে। ডিসকভারি এবং ফিনিশিং টাইম আগেরটাই লেখা আছে।

এখন লক্ষ্য করো তুমি যদি শুরুতে টপলোজিকাল অর্ডারে আগে থাকা নোডগুলো থেকে ডিএফএস চালাও অর্থাৎ যার ফিনিশিং টাইম বড় সেখান থেকে শুরু করো তাহলে তুমি প্রথম প্রথম SCC টা পেয়ে যাবে।

উপরের গ্রাফে 1 এর ফিনিশিং টাইম সরখেকে বেশি (14)। 1 থেকে ডিএফএস চালালে তুমি যেতে পারবে  $\{1,2,3\}$  নোডগুলোতে যারা একই SCC'র অংশ। এবার  $\{1,2,3\}$  নোডগুলো গ্রাফ থেকে মুছে ফেল। এরপর 4 এর ফিনিশিং টাইম বড়। 4 থেকে শুধুমাত্র {4} এ যাওয়া যায়। এরপর 5 থেকে ডিএফএস চালাবো, সেখান থেকে যাওয়া যায়  $\{5,6,7\}$  নোডগুলোতে যারা একটি SCC এর অংশ।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

যেসব নোডগুলো একই কম্পোনেন্ট এর অংশ তাদের কে আমরা আলাদা লিস্টে সেভ করে রাখবো নিজের সুড়োকোড়টা দেখো:

```
1  procedure DFS(G, u):
2      color[u] ← GREY
3      for all edges from u to v in G.adjacentEdges(u) do
4          if color[v]=WHITE
5              DFS(G,v)
6          end if
7      end for
8      stk.add(source)
9      return
10
11
12
13
14  procedure DFS2(R,u, mark)
15      components[mark].add(u) //save the nodes of the new component
16      visited[u] ← true
17      for all edges from u to v in R.adjacentEdges(u) do
18          if visited[v] ← false
19              DFS2(R,v, mark)
20          end if
21      end for
22      return
23
24  procedure findSCC(G):
25      stk ← an empty stack
26      visited[] ← null
27      color[] ← null
28      components[] ← null
29      mark=0
30      for each u in G
31          if color[u]=WHITE
32              DFS(G,u)
33          end if
34      end for
35      R=reverseEdges(G)
36      while stk not empty
37          u=stk.removeTop()
38          if visited[u]=false
39              mark=mark+1 //A new component found, it will be identified by 'mark'
40              DFS2(R,u,mark)
41          end if
42      end for
43      return components
```

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Be the first of your friends to like this



ডাইনামিক প্রোগ্রামিং এর নতুন  
সিরিজের ৬ নম্বর লেখায় আমরা  
দেখবো কম্পিউটারিঙ্গের প্রবলেম এবং  
ডিসিপ্লিন প্রবলেম কিভাবে সমাধান  
করতে হ্য।

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / স্যার

কোডটা একটু বড় মনে হলেও বোধ্য খুব সহজ। প্রথমে একটা ডিএফএস চালিয়ে ফিনিশিং টাইম অনুযায়ী নোডগুলো সর্ট করছি। একটা স্ট্যাক ব্যবহার করে কাজটা করছি। যার ফিনিশিং টাইম কম সে কাজ আগে শেষ করে ১১ নম্বর লাইনে আসবে, তখন সেই নোডটা স্ট্যাকে ঢুকিয়ে রাখবো। সবশেষে স্ট্যাকের উপরে যে নোড থাকবে তার ফিনিশিং টাইম হবে সব থেকে বেশি। এর পর ২য় ডিএফএস চালিয়ে কম্পোনেন্টগুলো আলাদা করে ফেলবো। mark নামের ভ্যারিয়েবল টা ব্যবহার করছি প্রতিটা কম্পোনেন্ট এর আলাদা নাম দেয়ার জন্য, ছবিতে যেভাবে A,B,C নাম দেয়া হয়েছে।

সলভ করার জন্য কিছু প্রবলেম পাবে এখানে।

## ফেসবুকে মন্তব্য

1 comments

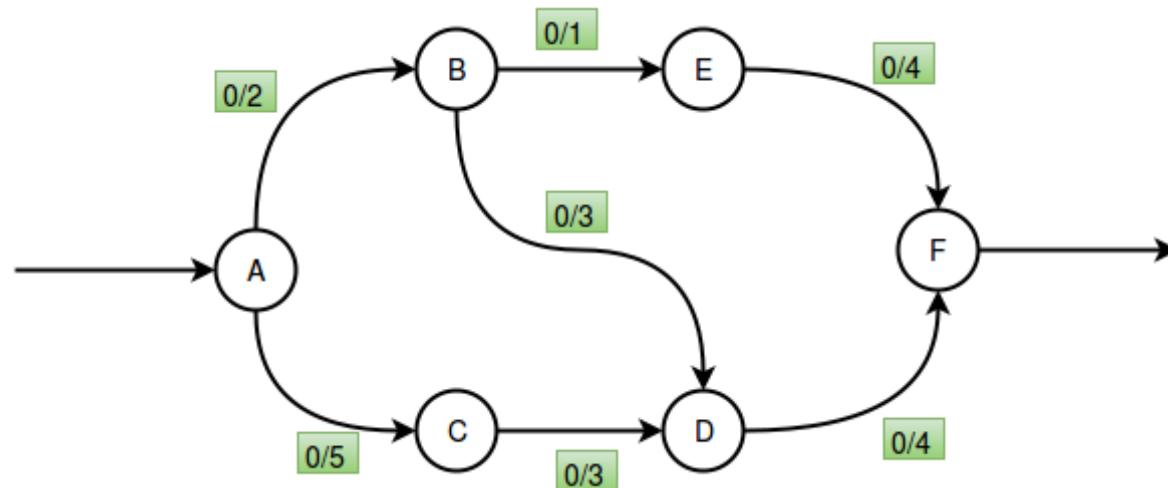
# গ্রাফ থিওরিতে হাতেখড়ি ১২ – ম্যাক্সিমাম ফ্লো (১)

জানুয়ারি ২৩, ২০১৫ by [শাফায়েত](#)



এই লেখায় আমরা গ্রাফে ম্যাক্সিমাম ফ্লো বের করার অ্যালগরিদম শিখবো। ম্যাক্স ফ্লো এর ধারণাটা ব্যবহার করে বেশ কিছু ইটারেস্টিং প্রবলেম সলভ করা যায়, তাই এটা শেখা খুবই গুরুত্বপূর্ণ। এই লেখাটা পড়ার আগে তোমাকে [গ্রাফ থিওরি](#) বেসিক অ্যালগরিদমগুলো, বিশেষ করে শর্টস্টেট পাথ বের করার অ্যালগরিদমগুলো ভালো করে শিখে নিবে।

প্রথমেই আমরা দেখি একটি খুবই সাধারণ ম্যাক্স ফ্লো প্রবলেম:



চিত্র: ১

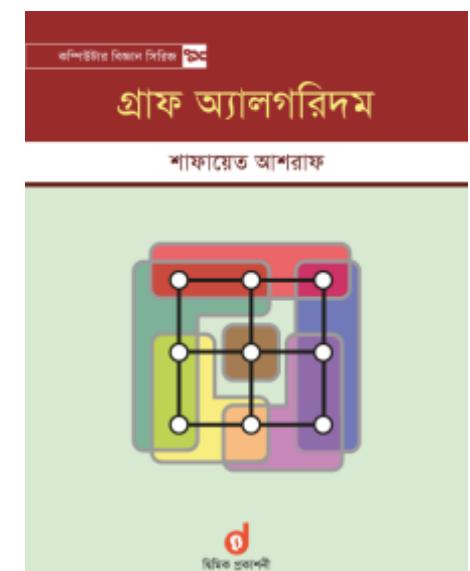
সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



তোমাকে চিত্র-১ এর মত একটা গ্রাফ দেয়া আছে। মনে কর গ্রাফের প্রতিটা এজ একটা করে পানির পাইপ। প্রতিটা পাইপ দিয়ে প্রতি সেকেন্ডে কত লিটার পানি প্রবাহিত হতে পারবে সেটার একটা সীমা আছে যেটাকে বলা হয় পাইপের **ক্যাপাসিটি**। আর কোনো পাইপ দিয়ে সেকেন্ডে যতটুকু পানি যাচ্ছে সেটা হলো পাইপটার **ফ্রো**। প্রতিটা এজের সাথে “ফ্রো/ক্যাপাসিটি” উল্লেখ করে দেয়া হয়েছে। যেমন A->B এজ দিয়ে সেকেন্ডে সর্বোচ্চ ২লিটার পানি যেতে পারে এবং এই মুহূর্তে সেকেন্ডে পানি যাচ্ছে ০ লিটার। শুধুমাত্র A নোড দিয়ে পানি প্রবেশ করানো যায় এবং শুধুমাত্র F নোড দিয়ে পানি বের হয়ে যায়। A কে বলা হয় **সোর্স** এবং F হলো **সিংক**। তোমাকে বলতে হবে A থেকে F এ প্রতি সেকেন্ডে সর্বোচ্চ কত লিটার পানি প্রবাহিত করা যাবে? অর্থাৎ পানির “ফ্রো” সর্বোচ্চ কত হতে পারে?

মূল সমস্যা সমাধানের আগে আমরা ছোট একটা সমস্যা সমাধান করি। A->C->D->F এই পথটা ব্যবহার করে পানি সোর্স থেকে সিংক এ গেলে সর্বোচ্চ কত লিটার পানি প্রবাহিত হতে পারবে? ছবিতে দেখা যাচ্ছে A->C এজের ক্যাপাসিটি ৫, C->D এজের ক্যাপাসিটি ৩, D->F এজের ক্যাপাসিটি ৪। এখনে সর্বনিম্ন ক্যাপাসিটি হলো ৩, তাই আমরা কোনো সময় তলিটারের বেশি পানি এই পথে পাঠাতে পারবোনা।

কোনো পথের সর্বনিম্ন ক্যাপাসিটির এজ সেই পথের ফ্রো নিয়ন্ত্রণ করবে। বোতলের সরু মুখের সাথে তুলনা দিয়ে এই ব্যাপারটাকে বলা হয় **বোটলনেক(bottleneck)**। যেমন ধরো তোমার বাসায় ৫MBps এর ইন্টারনেট কানেকশন আছে, এখন তুমি একটা মুভি ডাউনলোড করতে চাচ্ছ। এখন মুভির সার্ভার থেকে তোমার বাসায় ডাটা প্যাকেট আসার আগে অনেকগুলো নেটওয়ার্ক পার হয়ে আসে, কোনো একটা নেটওয়ার্কে গতি মাত্র ১MBps। এখন নেটওয়ার্কের বাকি অংশের গতি যতই বেশি হোক না কেন তুমি ১MBps এর বেশি গতিতে ডাউনলোড করতে পারবে না, নেটওয়ার্কের সরবেকে ধীরগতির অংশই তোমার ডাউনলোডের গতি নিয়ন্ত্রণ করবে।

এখন দেখি কিভাবে সর্বোচ্চ পানির ফ্রো পাবার সমস্যাটার সমাধান করা যায়। সমাধান খুবই সহজ, আমরা প্রতিবার যেকোনো একটা করে পথ দিয়ে পানি পাঠাতে থাকবো যতক্ষণ পাইপে ক্যাপাসিটি থাকে। কিন্তু এভাবে কি আমরা সর্বোচ্চ ফ্রো বা প্রবাহ পাবো? পাবো তবে সেজন্য একটু বুদ্ধি খাটাতে হবে। প্রথমে আমরা একটা পথে পানির ফ্রো পাঠিয়ে দেখি কি ঘটে। যেকোনো একটা পথ বেছে নিতে পারি, আমি বুঝানোর সুবিধার জন্য A->B->D->F পথটা বেছে নিলাম। এই পথে সর্বনিম্ন ক্যাপাসিটি হলো ২। তাহলে এই পথে ২লিটার পানির ফ্রো পাঠিয়ে দেবার পর গ্রাফটা দেখতে হবে এরকম:



প্রোগ্রামিং কনটেন্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

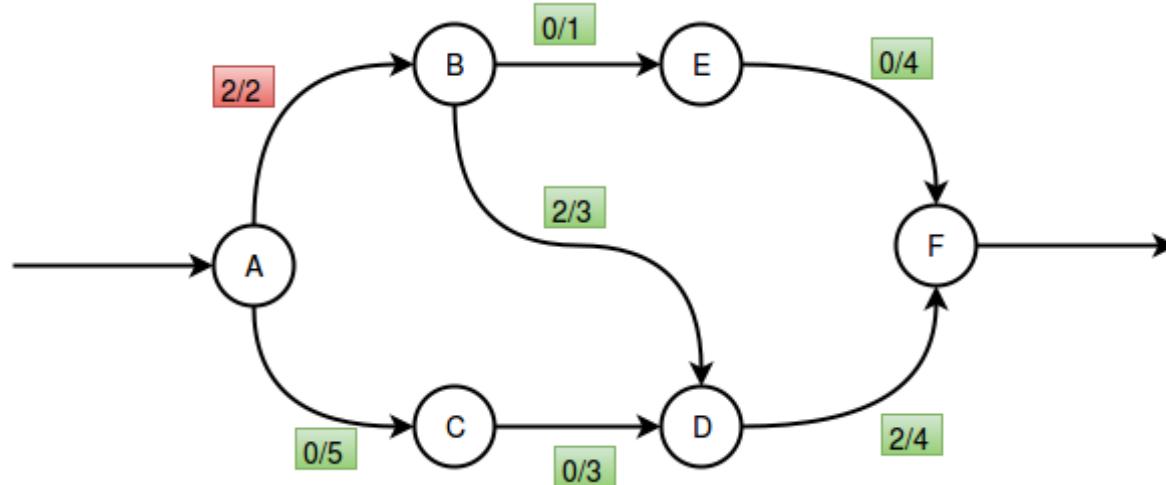
ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



চিত্র-২: A->B->D->F পথে ফ্লো পাঠানোর পর

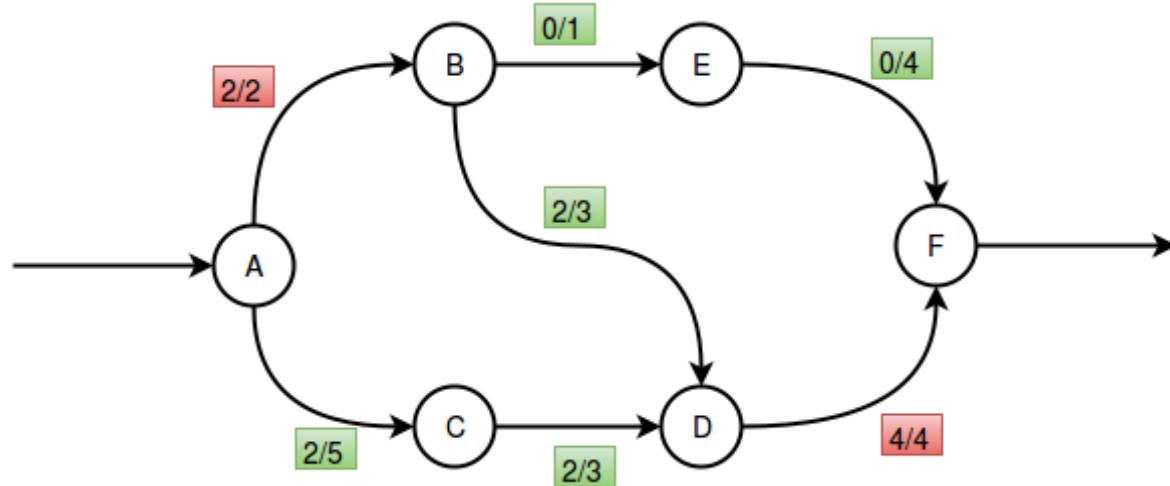
আমরা A->B->D->F পথের প্রতিটি এজের ফ্লো ২বার্ডিয়ে দিয়েছি। লক্ষ্য করো A->B এজের আসল ক্যাপাসিটি ২ এবং ফ্লো পাঠানো হয়েছে ২, তাই এই এজ নিয়ে আর কোনো ফ্লো পাঠানো যাবে না। তেমনি B->D এজের ক্যাপাসিটি ৩ কিন্তু ফ্লো পাঠানো হয়েছে ২, তাই এই এজটা দিয়ে আরো  $3-2=1$  ফ্লো পাঠানো সম্ভব। আমরা বলতে পারি B->D এজের **residual ক্যাপাসিটি=১**, residual শব্দটার অর্থ হলো কোনো কিছু ব্যবহার করে ফেলার পর বেচে যাওয়া অংশ।

**“ residual ক্যাপাসিটি = এজ এর ক্যাপাসিটি - ব্যবহৃত ক্যাপাসিটি বা ফ্লো এর পরিমাণ ।**

এবার একইভাবে A->C->D->F পথটা নির্বাচিত করি। এই পথে D->F এজের আসল ক্যাপাসিটি ৪ হলেও ২ ফ্লো আগেই পাঠানো হয়েছে, তাই বর্তমান ক্যাপাসিটি বা **residual ক্যাপাসিটি**  $4-2=2$ । এই পথটা দিয়ে ২ ফ্লো পাঠানো সম্ভব।

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)  
ডিসজয়েট সেট(ইউনিয়ন ফাইল)  
ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)  
সেগমেন্ট ট্রি-১  
সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)  
অ্যারে কমপ্রেশন/ম্যাপিং  
লোয়েস্ট কমন অ্যানসেক্টর  
বাইনারি ইনডেক্সড ট্রি  
স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:  
এল-আর-ইউ ক্যাশ(**নতুন**)  
ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):  
গ্রাফ থিওরিতে হাতেখড়ি  
অ্যাডজেসেন্সি ম্যাট্রিক্স  
অ্যাডজেসেন্সি লিস্ট  
ব্রেথ্র ফার্স্ট সার্চ (বিএফএস)  
মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)  
মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)  
টপোলজিকাল সর্ট  
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
সর্ট  
ডায়াক্রস্টা  
ফ্লয়েড ওয়ার্শল  
বেলম্যান ফোর্ড



চিত্র-৩: A->C->D->F পথে ফ্লো পাঠানোর পর

আমরা এখন মোট  $2+2=4$  ফ্লো পেলাম, অর্থাৎ পানি প্রবাহের হার এখন সেকেতে 4 লিটার। এই গ্রাফে কি আরো বেশি পানির ফ্লো পাঠানো সম্ভব? A->B এবং D->F এজ এরই মধ্যে সর্বোচ্চ ক্যাপাসিটি পৌছে গেছে, এই ২টা এজের কোনোটাকে না নিয়ে সিংক এ যাবার পথ গ্রাফে নেই। তাই দেখে মনে হতে পারে আর ফ্লো পাঠানো সম্ভব না। কিন্তু আমরা এখন বুদ্ধি খাটিয়ে আরো বেশি ফ্লো পাঠিয়ে দিব!

আমাদের এখন প্রতিটা এজের residual ক্যাপাসিটি ব্যবহার করে **residual গ্রাফ** আকঁতে হবে কিভাবে আরো ফ্লো পাঠাবো সেটা বুঝতে হল। residual গ্রাফ আকার নিয়ম হলো:

- “ ১. গ্রাফের প্রতিটা এজ  $(u,v)$  এর ক্যাপাসিটি হবে এজ টার residual ক্যাপাসিটির সমান।
- ২. প্রতি এজ  $(u,v)$  এর জন্য উল্টা এজ  $(v,u)$  এর residual ক্যাপাসিটি হবে  $(u,v)$  এজ এ ফ্লো এর সমান।

তারমানে কোনো এজ দিয়ে যতটুকু ফ্লো পাঠিয়েছি সেটা হবে উল্টো এজের residual ক্যাপাসিটি। তাহলে residual গ্রাফে মূল এজ এবং উল্টো এজের residual ক্যাপাসিটির যোগফল হবে মূল এজ এজের মোট ক্যাপাসিটির সমান। চিত্র-৪ এর গ্রাফটার residual গ্রাফটা হবে এরকম:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রিংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

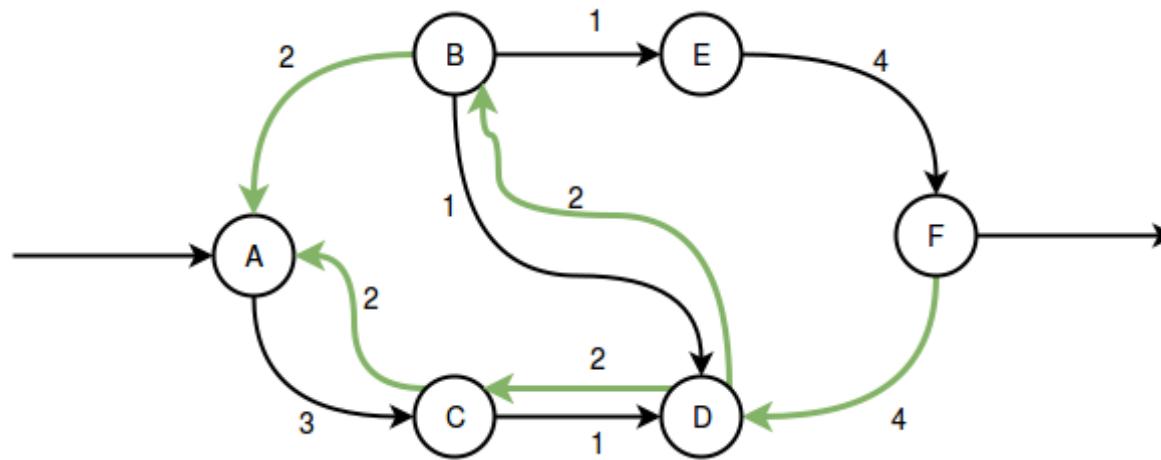
ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সারসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)



চিত্র-৪: ৩নং চিত্রের গ্রাফের residual গ্রাফ

সবুজ রঙ দিয়ে উল্লেখ এজ এবং তাদের residual ক্যাপাসিটি দেখানো হয়েছে। রেড ক্যাপাসিটির এজগুলোকে গ্রাফে একে দেখাইনি।  $A \rightarrow C$  এজ দিয়ে আগে ২ফ্লো পাঠানো হয়েছে বলে উল্লেখ এজ এবং residual ক্যাপাসিটি ২, এবং এজটার আরো ৩ ফ্লো পাঠানোর ক্ষমতা আছে তাই মূল এজ এবং residual ক্যাপাসিটি ৩। ঠিক এভাবে অন্যান্য এজগুলো আঁকা হয়েছে, তুমি নিজে একে ঘাটাই করে নিতে পারো ঠিকমত বুঝেছো নাকি বা আমার ছবিতে ভুল আছে নাকি।

এখন উল্লেখ এজ এ ফ্লো পাঠানোর মানে কি? আমরাতো পাইপের উল্লেখ দিক দিয়ে পানি পাঠাতে পারবো না বা রাশার উল্লেখ দিক দিয়ে গাড়ি চালাতে পারবো না। উল্লেখ দিকে ফ্লো পাঠানোর মানে হলো মূল ফ্লো টাকে বাতিল করে দেয়া! আমরা যদি  $D \rightarrow B$  এজ দিয়ে ২ফ্লো পাঠাই তার মানে  $B \rightarrow D$  এজ দিয়ে আসা ২লিটার পানির প্রবাহকে আমরা বাতিল করে দিচ্ছি। তখন residual গ্রাফে মূল এজের residual ক্যাপাসিটি যাবে বেড়ে, এবং উল্লেখ এজের residual ক্যাপাসিটি যাবে কমে, যোগফল আগের মতই থাকবে!

এখন চিন্তা করে দেখো চিত্র-৪ দিয়ে আরো ফ্লো সিংক এ পাঠানো যায় নাকি। মনে রাখবে চিত্র-৩ আর চিত্র-৪ এ একই গ্রাফকেই ডিন্বভাবে আঁকা হয়েছে।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্স(১):

ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

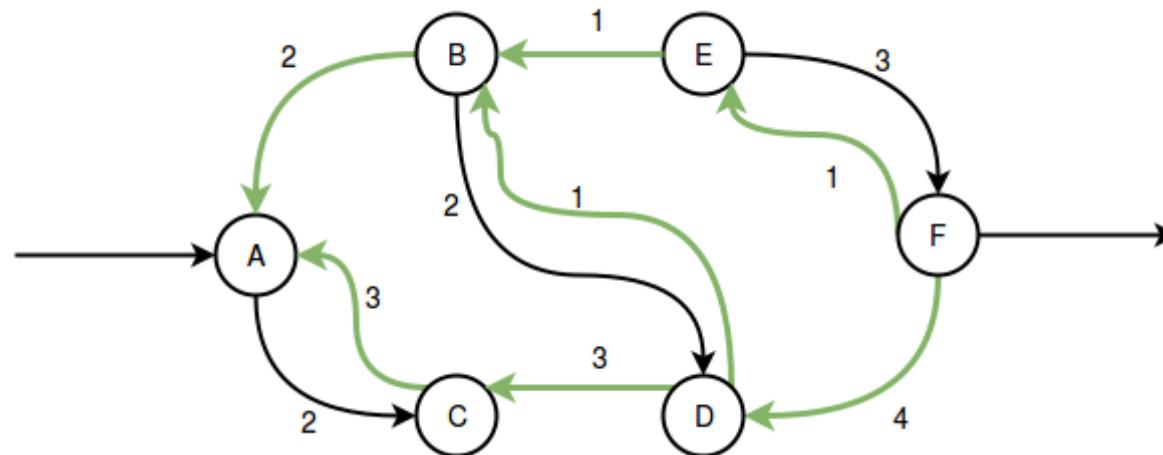
ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

চিত্র ৪ এ দেখা যাচ্ছে A->C->D->B->E->F পথে আরো ১ ফ্লো পাঠানো সম্ভব! এই পথে D->B হলো উল্টো এজ। তারমানে D->B এজ এর ১ ফ্লো বাতিল করে দিয়ে আমরা গ্রাফে মোট ফ্লো ১বাড়িয়ে ফেলতে পারছি। এখন residual গ্রাফ কিরকম হবে দেখি:



চিত্র-৫: A->C->D->B->E->F পথে ফ্লো পাঠানোর পর residual গ্রাফ

তুমি যদি এ পর্যন্ত বুঝে থাকो তাহলে চিত্র-৫ থেকে মূল গ্রাফটাও নিজে একে নিতে পারবে। চিত্র-৪ এ C->D এজ দিয়ে ১ ফ্লো পাঠানো হয়েছে। তাই চিত্র-৫ এ C->D এজের residual ক্যাপাসিটি কমে  $1-1=0$  হয়ে গিয়েছে এবং D->C এজের residual ক্যাপাসিটি বেড়ে  $2+1=3$  হয়ে গিয়েছে। এভাবে সবগুলো এজ আঁকা হয়েছে।

এই গ্রাফে সোর্স থেকে সিংক এ পানি পাঠানোর আর কোনো পথ নেই। আমরা ৩টি পথে মোট  $2+2+1=5$  ফ্লো পেয়েছি, অর্থাৎ পানি প্রবাহের হার প্রতি সেকেন্ডে ৫ লিটার। এটাই গ্রাফটার জন্য ম্যাক্সিমাম ফ্লো।

residual গ্রাফে আমরা যখন একটা পথ বের করি সেই পথের একটা বিশেষ নাম আছে, সেটা হলো **অগমেন্টেড পাথ(Augmented path)**। তাহলে আমদের অ্যালগোরিদম খুব সহজ, যতক্ষণ সম্ভব আমরা residual graph এ একটা অগমেন্টেড পাথ খুজে বের করবো এবং সেই পথে ফ্লো পার্শ্যে দিবো! এটার নাম Ford-Fulkerson অ্যালগোরিদম। এটার সুড়েকোড এরকম:

1 Input: A graph  $G = (V, E)$  with flow capacity cap, source node s and sink node t.

কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

**Shafaetsplanet**  
2.4K likes  
শাফায়েত আশরাফ

---

**Like Page**

Be the first of your friends to like this

Shafaetsplanet  
about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিনের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

---

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / ম্যার

```

2 Output: Calculate maximum flow from s to t.
3 Initialize:
4   1. total_flow=0
5   2. Residual Capacity Cf(u,v)=cap(u,v) for each edge(u,v) in the graph
6 Algorithm:
7   1. While there is a path from s to t such that Cf[u][v]>0 for all edges (u,v) in path:
8     1.1. min_res_cap = minimum residual capacity among all the edges (u,v) in the path.
9     1.2. For each edge (u,v) in the path:
10       Cf(u,v) = Cf(u,v) - min_res_cap
11       Cf(v,u) = Cf(v,u) + min_res_cap
12     1.3. total_flow = total_flow + min_res_cap
13   3. Return total_flow

```

সহজ কথায় একটা ২-ডি অ্যারে  $C_f[u][v]$  তে প্রতিটা এজের residual ক্যাপাসিটি থাকবে, যে এজগুলো গ্রাফে নাই সে এজের residual ক্যাপাসিটি ০। এখন তুমি সোর্স থেকে সিংক এ যাবার এমন একটা পথ বের করবে যে পথের প্রতিটা এজের residual ক্যাপাসিটি ০ থেকে বড়। পথের এজগুলোর মধ্যে মিনিমাম residual ক্যাপাসিটির মান খুজে বের করবে। ফ্লো পাঠানোর পর প্রতিটা এজের residual ক্যাপাসিটি কমে যাবে এবং উল্টা এজের residual ক্যাপাসিটি বেড়ে যাবে। মিনিমাম residual ক্যাপাসিটিটা মোট ফ্লো এর মানের সাথে যোগ হতে থাকবে। সবশেষে চাইলে তুমি আসল ক্যাপাসিটি থেকে residual capacity বিয়োগ করে কোনো এজ এ ফ্লো এর পরিমাণ নির্ণয় করতে পারো।

গ্রাফটা ডিরেক্টেড হতে হবে এমন কোনো কথা নেই। বাইডিরেকশনাল এজও থাকতে পারে। A-B বাইডিরেকশনাল এজের ক্যাপাসিটি ১০ হলে  $C_f[A][B] = C_f[B][A]=10$  হবে। এখন আগের মতই অগমেন্টেড পাথ বের করে ম্যাক্সিমাম ফ্লো বের করতে পারবে। তবে এক্ষেত্রে কোন নোডে ফ্লো কর্ত হচ্ছে সেটা বের করতে হলে তোমাকে আরেকটু বুদ্ধি খাটাতে হবে।

অ্যালগোরিদমে ২নম্বর ধাপে পাথ বের করার জন্য **বিএফএস/ডিএফএস/বেলম্যান ফোর্ড** ইত্যাদি অ্যালগোরিদম ব্যবহার করা যেতে পারে। প্রবলেমের ধরণ অনুযায়ী এই ধাপে একেক অ্যালগোরিদম একেক ধরণের সুবিধা দিবে, সেগুলো তুমি প্রবলেম সলভ করতে করতে জানতে পারবে। আর পাথ বের করার জন্য বিএফএস বা যেটাই ব্যবহার করোনা কেন, প্রতিটা নোডে যাবার সময় নোডটার প্যারেন্ট কে সেটা মনে রাখতে হবে কোনো একটা অ্যারেতে।

তুমি যদি বিএফএস ব্যবহার করে সমাধান করো তাহলে সেটাকে বলা হ্য **এডমন্ড কার্প** অ্যালগোরিদম। ফোর্ড-ফুলকার্সন অ্যালগোরিদমে শুধু পাথ খুজে বের করার কথা বলে হয়েছে, সেটা যেকোনোভাবে বের করা যেতে পারে, আর এডমন্ড-কার্প বিএফএস ব্যবহার করে কাজটা করতে বলা

হয়েছে অর্থাৎ ফোর্ড-ফুলকার্সন ইমপ্লিমেন্ট করার একটা উপায় হলো এডমন্ড কার্প অ্যালগোরিদম যার কমপ্লেক্সিটি  $O(VE^2)$ ।

এখন তোমাদের জন্য চিন্তা করার মত কয়েকটা প্রশ্ন:

প্রশ্ন ১: আমাদের প্রবলেমে সোর্স এবং সিংক ছিলো একটা। কিন্তু গ্রাফে একাধিক নোড দিয়ে পানি প্রবেশ করলে এবং একাধিক নোড দিয়ে পানি বের হয়ে গেলে কিভাবে অ্যালগোরিদমটা পরিবর্তন করবে?

প্রশ্ন ২: যদি প্রতিটা নোডের কিছু ক্যাপাসিটি থাকে, অর্থাৎ একটা নোড দিয়ে কত সর্বোচ্চ ফ্লো পাঠানো যাবে সেটা নির্দিষ্ট করা থাকে তাহলে কিভাবে সমাধান করবে?

প্রশ্ন ৩: দুটি নোডের মধ্যে একাধিক এজ থাকলে কি করবে?

প্রশ্ন ৪: দুই বন্ধু একই নোড থেকে যাত্রা শুরু করে একই গন্তব্যে পৌছাতে চায় কিন্তু দুইজনেই চায় ডি঱ ডিম রাস্তা ব্যবহার করে যেতে, তারমানে একই এজ কখনো ২জন ব্যবহার করতে পারবে না। গ্রাফটি দেয়া হলে তুমি কি বলতে পারবে এরকম ২টি পথ আছে নাকি? এ ধরণের পথকে এজ ডিসজঞ্জেন্ট পাথ বলে।

এই প্রশ্নগুলো নিয়ে আলোচনা করা হয়েছে [পরের পৰ্বে](#)।

প্রোগ্রামিং কনটেস্টে ম্যাক্সিমাম ফ্লো প্রবলেম এ কোডিং এর থেকে অনেক কঠিন হলো গ্রাফটা কিভাবে বানাতে হবে সেটা বের করা, তাই অনেক সমস্যা সমাধান করে প্র্যাকটিস করতে হবে। শুরু করার জন্য কয়েকটা প্রবলেম দিয়ে দিলাম:

[http://lightoj.com/volume_showproblem.php?problem=1153](http://lightoj.com/volume_showproblem.php?problem=1153)

[http://lightoj.com/volume_showproblem.php?problem=1155](http://lightoj.com/volume_showproblem.php?problem=1155)

<http://uva.onlinejudge.org/external/110/11045.html>

হ্যাপি কোডিং!

# গ্রাফ থিওরিতে হাতেখড়ি-১২ – ম্যাট্রিমাম ফ্লো (২)

মার্চ ৩০, ২০১৫ by শাফায়েত



আগের পরে আমরা দেখেছি কিভাবে ফোর্ড-ফুলকারসন পদ্ধতি ব্যবহার করে ম্যাট্রিমাম ফ্লো বের করতে হয়। এই পরে ম্যাট্রিমাম ফ্লো সমস্যার সহজ কিছু ভ্যারিয়েশন দেখবো।

## একাধিক সোর্স/সিংক:

আগের পরে একটা প্রশ্ন করেছিলাম এরকম “আমাদের প্রবলেমে সোর্স এবং সিংক ছিলো একটা। কিন্তু গ্রাফে একাধিক নোড দিয়ে পানি প্রবেশ করলে এবং একাধিক নোড দিয়ে পানি বের হয়ে গেলে কিভাবে অ্যালগোরিদমটা পরিবর্তন করবে?” অর্থাৎ একাধিক সোর্স বা সিংক থাকলে কি করতে হবে সেটা জানতে চাওয়া হয়েছে।

চিত্র-১ এ বাম পাশের নীল নোডগুলো হলো সোর্স এবং ডানের সবুজ নোডগুলো হলো সিংক।

## সাবক্ষাইব

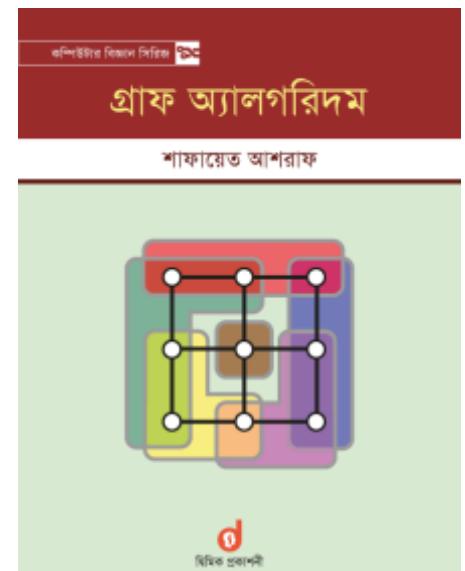


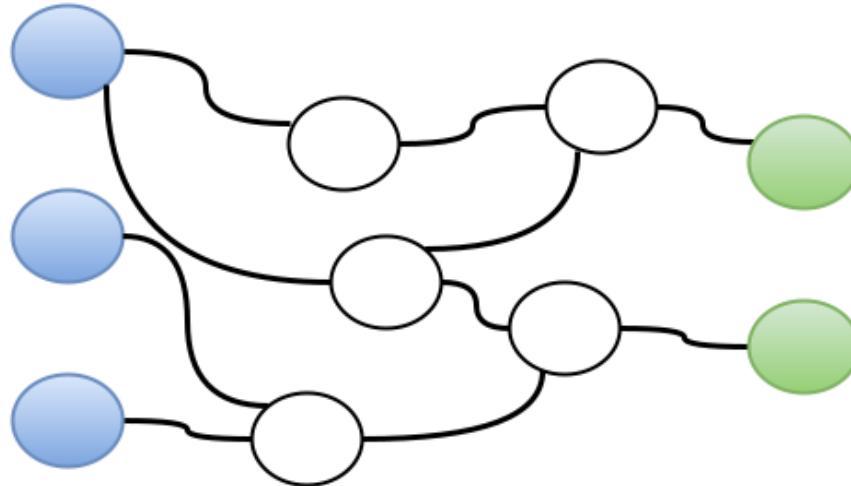
Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

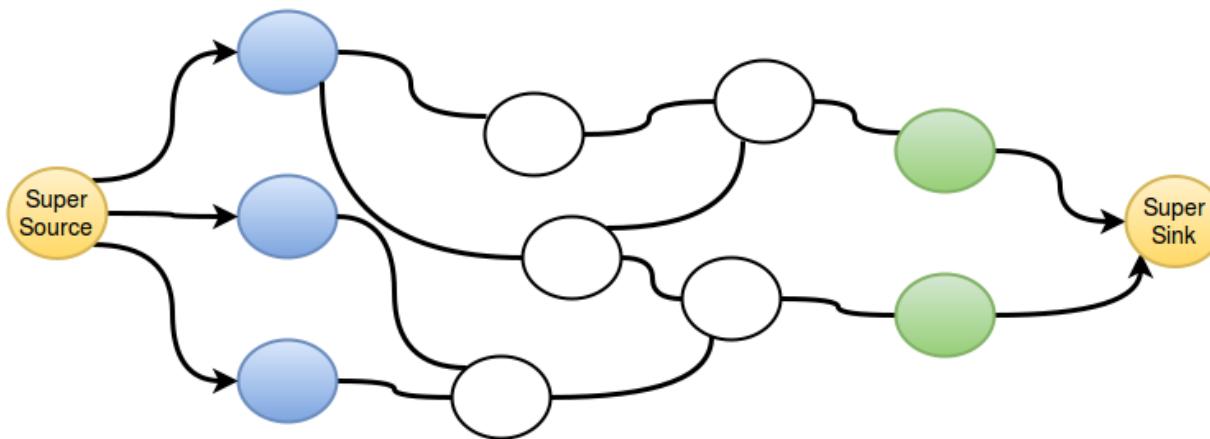
**Traveloka Singapore** (বিস্তারিত...)





চিত্র -1: একাধিক সোর্স এবং সিংক সহ একটি গ্রাফ

এ ধরণের গ্রাফে এডমন্ড কার্প অ্যালগোরিদম প্রয়োগ করার সহজ উপায় হলো সুপার-সোর্স এবং সুপার সিংক বানিয়ে নেয়া। সুপার সোর্স হলো এমন একটা নোড যেটা সবগুলো সোর্সের সাথে ডিরেক্টেড এজ দিয়ে যুক্ত। ঠিক সেভাবে, সুপার সিংক প্রতিটি সিংকের সাথে ডিরেক্টেড এজ দিয়ে সংযুক্ত। এবং এই এজগুলোর ক্যাপাসিটি হবে অসীম বা ইনফিনিটি।



চিত্র-2: সুপার সোর্স এবং সুপার সিংক



প্রোগ্রামিং কনটেন্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগোরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ল্যাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

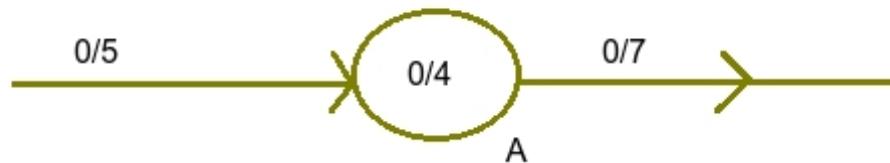
স্ট্যাক

কিউ+সার্কুলার কিউ

চিত্র-২ তে সুপার সোর্স এবং সুপার সিংক দেখানো হয়েছে। ইনফিনিটি হিসাবে বেছে নিতে পারো সবগুলো এজের সম্মিলিত ক্যাপাসিটির থেকে বড় কোনো মানকে। এখন সাধারণ ফ্লো অ্যালগোরিদম ব্যবহার করেই এই গ্রাফে ম্যাঞ্চিমাম ফ্লো বের করতে পারবে।

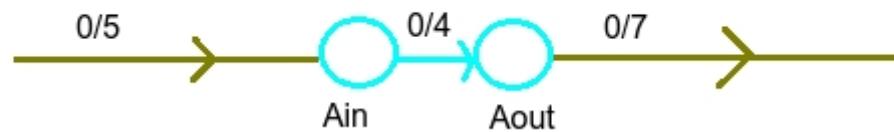
### নোড ক্যাপাসিটি:

এতক্ষণ সবগুলো গ্রাফে এজের নির্দিষ্ট ক্যাপাসিটি ছিলো, নোডের ছিলো না। কিন্তু বাস্তবে অনেক সমস্যায় নোডের ক্যাপাসিটি থাকতে পারে। যেমন ধরো কোনো একটা দেশে প্রতিটা রাস্তার পাশাপাশি প্রতিটা শহরের নির্দিষ্ট গাড়ি ধারণ ক্ষমতা আছে, সেই দেশের গ্রাফ চিত্র-৩ এর মতো হতে পারে:



চিত্র-৩: নোড ক্যাপাসিটি

ছবিতে পুরো গ্রাফটা না একে শুধু একটা নোড আর ২টি এজ একেছি, নোডটাতে ঢোকার এজ এর ক্যাপাসিটি ৫, যে এজটি বাইরে চলে গেছে তার ক্যাপাসিটি ৭, এদিকে নোডের নিজের ক্যাপাসিটি ৪। আগে শেখা অ্যালগোরিদমে আমরা এজের ক্যাপাসিটির হিসাব বাখার জন্য একটা অ্যারে ব্যবহার করেছিলাম, এখনও আমরা সেই অ্যারেটা ব্যবহার করেই কাজ করতে পারবো, বুদ্ধিটা হলো নোডটাকে দুই ভাগে ভাগ করে ফেলা, এবং ভাগ দুটিকে নতুন এজ দিয়ে যোগ করে দেয়া। চিত্র-৪ দেখলেই পরিষ্কার হবে ব্যাপারটা:



চিত্র-৪: A নোডটিকে দুইভাগ করা হয়েছে

স্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

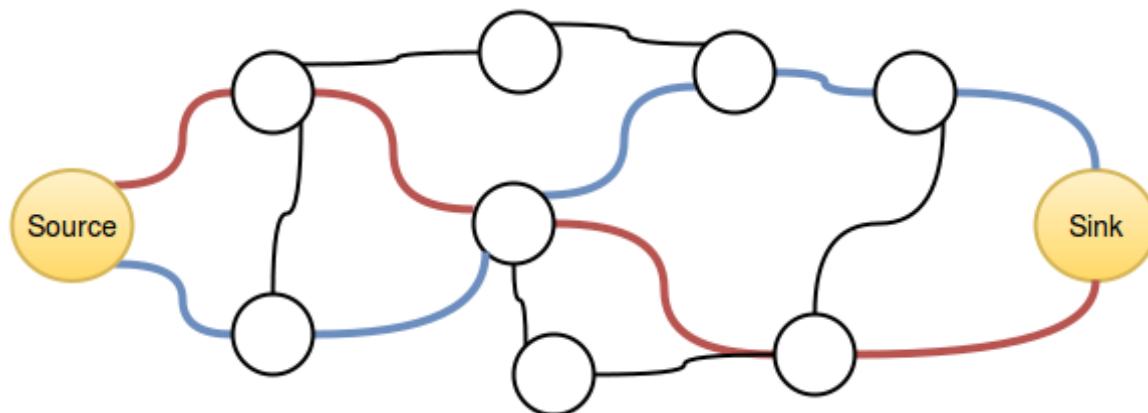
বেলম্যান ফোর্ড

আমরা A নোডটা Ain এবং Aout এই দুটি নোডে ভাগ করেছি। এখন আসল গ্রাফ যতগুলো এজ A তে প্রবেশ করেছে সেগুলো প্রবেশ করবে Ain এ এবং আসল গ্রাফে যতগুলো এজ A থেকে বাইরে গিয়েছে সেগুলো এখন বাইরে যাবে Aout থেকে। Ain থেকে Aout এ একটা এজ প্রবেশ করবে যেটার ক্যাপাসিটি হবে এজ এর ক্যাপাসিটির সমান।

এখন নিশ্চিন্তে তুমি আগের অ্যালগরিদম ব্যবহার করতে পারো। কোড লেখার সময় কিভাবে নোড দুইভাগ করবে, আনডিরেস্টেড গ্রাফের ক্ষেত্রে ব্যাপারটা কিরকম হবে সেগুলো চিন্তা করা তোমার কাজ!

### এজ ডিসজয়েন্ট পাথ:

দুই বন্ধু একই নোড থেকে যাত্রা শুরু করে একই গন্তব্যে পৌছাতে চায় কিন্তু দুইজনেই চায় ভিন্ন ভিন্ন রাস্তা ব্যবহার করে যেতে, তারমানে একই এজ কখনো ২জন ব্যবহার করতে পারবে না। এধরণের পথকে এজ ডিসজয়েন্ট পাথ বলে। তোমাকে বলতে হবে কোনো একটা গ্রাফে দুটি এজ ডিসজয়েন্ট পাথ আছে নাকি। চিত্র-৫ এ একটা উদাহরণ দেখানো হয়েছে:



ছবিতে দুইজনই বামের সোর্স নোডটা থেকে যাত্রা শুরু করে ডানের সিংক নোডে যেতে চায়। লাল এবং নীল রঙ ব্যবহার করে দুটি এজ-ডিসজয়েন্ট পাথ দেখানো হয়েছে।

সাধারণ ম্যাট্রিক্স ব্যবহার করেই এজ ডিসজয়েন্ট পাথ বের করা যায়। শুরুর নোডকে সোর্স এবং গন্তব্য নোডকে সিংক ধরবে। এবার সবগুলো এজ এর ক্যাপাসিটি বানিয়ে দাও ১ এর সমান। এখন যদি তুমি সোর্স থেকে সিংকে দুই ফ্লো পাঠাতো পারো সেটার মানে হলো দুটি ডিসজয়েন্ট পাথ

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাট্রিমাম ফ্লো-১

ম্যাট্রিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

আছে। প্রতিটা এজের ক্যাপাসিটি ১ হওয়াতে ২ ফ্লো যে দুটি পথে গিয়েছে তাদের মধ্যে কমন এজ থাকা সম্ভব না।

ঠিক একই ভাবে তুমি একটা গ্রাফে সর্বোচ্চ কয়টা ডিসজয়েন্ট পাথ থাকা সম্ভব অথবা দুই বন্ধুর জায়গায় K টা বন্ধু থাকলে কি হতো বের করে ফেলতে পারবে।

এখন প্রশ্ন হলো তুমি যদি প্রতিটা রাস্তার নির্দিষ্ট দৈর্ঘ্য থাকে এবং ডিসজয়েন্ট পাথ দুটির মোট দৈর্ঘ্য মিনিমাইজ করতে চাও তাহলে ফ্লো এবং অ্যালগোরিদমটা কিভাবে পরিবর্তন করবে? এটা বের করতে পারলে [uva 10806](#) সমস্যাটা সমাধান করে ফেলো, সমস্যাটার নামের ডিতরেই কিভাবে সমাধান করতে হবে বলা আছে!

আজকের পর্ব এখানেই শেষ। মিন-কাট এবং ম্যাট্রিক্স নিয়ে আলোচনার জন্য আরেকটা পর্ব অপেক্ষা করতে হবে। কনটেন্টে ম্যাট্রিক্স-ফ্লো প্রবলেমের কঠিন অংশ হলো গ্রাফটা কিভাবে তৈরি করবো, এজগুলো কিভাবে যোগ করবো, কোন এজের ক্যাপাসিটি কত এগুলো বের করা, এসব করার পর ফ্লো অ্যালগোরিদম চালিয়ে দেয়া সহজ কাজ। তাই তোমাকে প্রচুর প্র্যাকটিস করে এই জিনিসগুলো আয়স্বরে আনতে হবে।

কিছু প্রবলেম:

[Down Went Titanic](#)

[Clever Naming Pattern](#)

[Diagonal Sum](#)

[হ্যাপি কোডিং!](#)

[ফেসবুকে মন্তব্য](#)

1 comments

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেক্ষেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাট্রিক্স(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

[কোয়ান্টাম কম্পিউটার\(২\)](#)

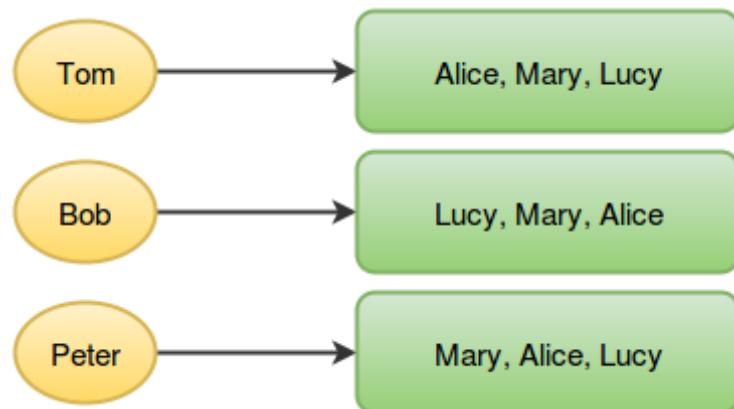
# গ্রাফ থিওরি: স্টেবল ম্যারিজ প্রবলেম

আগস্ট ১৫, ২০১২ by শাফায়েত



বেশ কিছুদিন ডিপি নিয়ে লেখার পর আবার গ্রাফ থিওরিতে ফিরে এলাম। আজকে আমরা একটা সহজ কিন্তু ইন্টারেস্টিং প্রবলেম দেখবো। স্টেবল ম্যারিজ(Stable Marriage) প্রবলেম এক ধরনের বাইপারটাইট ম্যাট্রিঃ প্রবলেম, তবে এটা শেখার জন্য অন্য কোনো অ্যালগরিদম জানার প্রয়োজন নেই।

মনে করি  $n$  টা ছেলে আর  $n$  টা মেয়ে আছে। এখন তাদের মধ্যে বিয়ে দিতে হবে এমন ভাবে যেনো বিয়ে “স্টেবল” হয়। প্রত্যেকের সাথেই প্রত্যেকের বিয়ে সম্ভব তবে প্রতিটা ছেলে আর মেয়ের কিছু পছন্দ আছে, প্রত্যেকেই চাহিবে তার পছন্দের মানুষকে বিয়ে করতে। যদি ছেলে ৩জনের নাম Tom,Bob,Peter , আর মেয়ে ৩জনের নাম Alice,Mary,Lucy হ্য তাহলে ছেলেদের পছন্দের তালিকা হতে পারে এরকম:



তালিকাটা বেশি থেকে কম পছন্দের ক্রমে করা হয়েছে। যেমন টম এলিসকে বেশি পছন্দ করে, লুসিকে কম পছন্দ করে।

## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

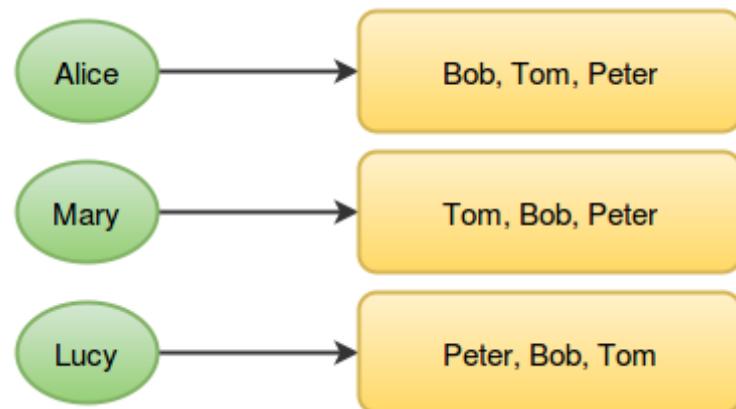
শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



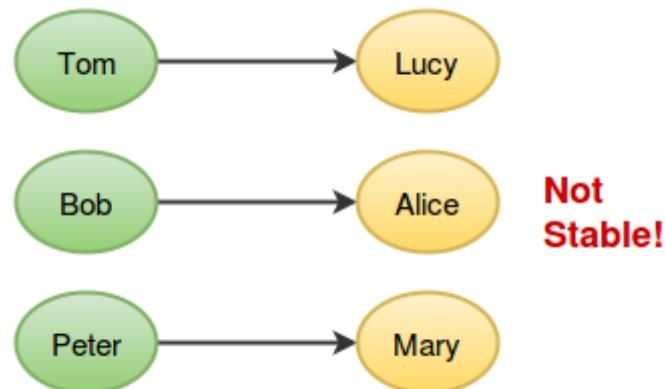
শাফায়েত আশুরাফ



আবার মেয়েদের পছন্দের তালিকাটা হতে পারে এরকম:



এখন কিভাবে বিয়ে দিলে বিয়ে স্টেবল হবে? আগে বুঝা দরকার স্টেবল বলতে কি বুঝাচ্ছি। ধরো নিচের মতো করে বিয়ে দেয়া হলো:



এই ম্যাট্রিক্স/বিয়েটা স্টেবল না, কারণ টম লুসির থেকে মেরিকে বেশি পছন্দ করে, আবার মেরি পিটারের থেকে টমকে বেশি পছন্দ করে। তাই টম আর মেরি বিয়ে ডেও একসাথে চলে আসতে পারে। যদি A,B ছেলে আর C,D মেয়ে হয় আর A-C এবং B-D কে বিয়ে দেয়া হয় তাহলে **বিয়ে স্টেবল হবেনা যদি** নিচের দুটি স্টেমেন্টই সত্য হয়:



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

১. A যদি C এর থেকে D কে বেশি পছন্দ করে।

“ ২. D যদি B এর থেকে A কে বেশি পছন্দ করে।

এটি স্টেটমেন্ট সত্য হলে A আর D বিয়ে ভেঙে চলে আসবে! তবে যেকোনো একটা স্টেটমেন্ট মিথ্যা হল বিয়ে স্টেবল হবে।

১৯৬২ সালে David Gale আর Lloyd Shapley প্রমাণ করেন, সমান সংখ্যক ছেলে আর মেয়ের জন্য সমসময় স্টেবল ম্যারেজ প্রবলেমের একটি সমাধান আছে। তারা খুব সহজে একটা অ্যালগোরিদম আবিষ্কার করেন সমস্যাটি সমাধানে জন্য। অ্যালগোরিদমটি এরকম:

“ ১. প্রথমে প্রতিটি অবিবাহিত ছেলে তার সবথেকে পছন্দের মেয়েটাকে প্রস্তাব পাঠাবে যাকে সে এখনো প্রস্তাব পাঠায়নি, মেয়েটি অল্বেড়ি এনগেজড হলেও সমস্যা নাই একটি মেয়েকে একাধিক ছেলে প্রস্তাব পাঠাতে পাবে। একটি ছেলে কখনো একটি মেয়েকে দুইবার প্রস্তাব পাঠাবেন।

২. এবার প্রতিটা মেয়ে তাকে যারা প্রস্তাব পাঠায়েছে তাদের মধ্যে থেকে যাকে সবথেকে পছন্দ তাকে নির্বাচিত করবে, বাকি সবাইকে বাতিল করে দিবে। মেয়েটি আগেই কাওকে পছন্দ করে থাকলে তাকেও বাতিল করে দিবে।

৩. এখনো কেও অবিবাহিত থাকলে ১ম ধাপের পুনরাবৃত্তি হবে।

অ্যালগোরিদমটি কেনো কাজ করে? ধরি A-C এবং B-D এর বিয়ে দেয়া হয়েছে। তাহলে বিয়ে ভাঙবে A যদি D কে বেশি পছন্দ করে এবং D যদি A কে বেশি পছন্দ করে। কিন্তু উপরের অ্যালগোরিমে সেটা সম্ভবনা। কারণ:

A যদি D কে বেশি পছন্দ করে তাহলে সে D কে আগে প্রস্তাব পাঠাবে, D রাজি না হলে বা ছেড়ে দিলেই একমাত্র C কে প্রস্তাব পাঠাবে।

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

D যদি A কে বেশি পছন্দ করে তাহলে সে অন্য যে কাওকে ছেড়ে দিয়ে A কে বিয়ে করবে।

“ আর D যদি A কে বিয়ে না করে অন্য কাওকে করে তারমানে সে অন্য কাওকেই বেশি পছন্দ করে, এক্ষেত্রে বিয়ে ডাঙুর সভাবনা নেই।

এই অ্যালগোরিমটা স্টেবল ম্যাটিং দিবে ঠিকই তবে অপটিমাল রেজাল্ট নাও দিতে পারে। প্রতিটি ছেলের জন্য রেজাল্ট অপটিমাল হবে, কিন্তু মেয়েদের জন্য অপটিমাল নাও হতে পারে, অর্থাৎ এমন স্টেবল ম্যাটিং থাকতে পারে যেটাও কোনো একটি মেয়ে আরো পছন্দের কাওকে বিয়ে করতে পারতো। অর্থাৎ যে প্রস্তাব পাঠাবে তার জন্য রেজাল্ট অপটিমাল হবে।

অ্যালগোরিদমটি কোডে ইম্প্রিমেন্ট করা খুব সহজ। preference লিস্ট তোমাকে ইনপুট দেয়া থাকবে। কে কাকে প্রস্তাব পাঠিয়েছে, কে কার সাথে এখন এনগেজড এই তথ্যগুলো অ্যাবেতে রেখে সহজেই কোডটা লিখে ফেলতে পারবে। [wikipedia](#) তে দেয়া সুড়োকোড়টা এরকম:

```
1 function stableMatching {
2   Initialize all m ∈ M and w ∈ W to free
3   while ∃ free man m who still has a woman w to propose to {
4     w = m's highest ranked such woman to whom he has not yet proposed
5     if w is free
6       (m, w) become engaged
7     else some pair (m', w) already exists
8     if w prefers m to m'
9       (m, w) become engaged
10    m' becomes free
11  else
12    (m', w) remain engaged
13  }
14 }
```

প্রবলেম:

[Light OJ: Employment](#)

[Codechef: Stable Marriage](#)

[Uva: Chemical Attraction](#)

[Codechef: Blocking](#)

আর্টিকুলেশন পয়েন্ট এবং বিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটাৰ নিৰ্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগোরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিস্টিং)

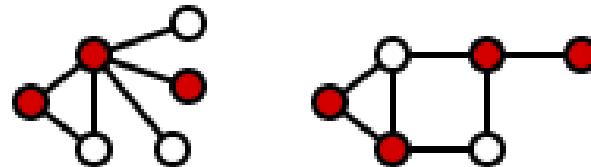
ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

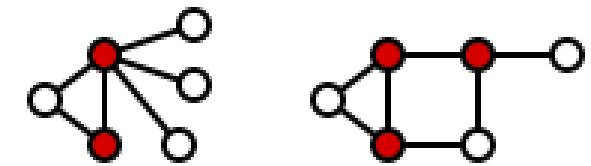
ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেটরিঙ্ক, ডিসিশন প্রবলেম)

# মিনিমাম ভারটেক্স কভার প্রবলেম

অক্টোবর ৪, ২০১২ by শাফায়েত



মিনিমাম ভারটেক্স কভার একটি ক্লাসিক গ্রাফ প্রবলেম। ধৰা যাক একটি শহরে কিছু রাস্তা আছে, এখন প্রতি রাস্তায় মোড়ে আমরা পাহারাদার বসাতে চাই। কোনো নোডে পাহারাদার বসালে সে নোডের সাথে যুক্ত রাস্তাগুলো একাই পাহারা দিতে পারে। উপরের ছবিতে নোডগুলো হলো রাস্তার মোড়। এখন সব কয়টা রাস্তা পাহারা দিতে নৃন্যতম কয়জন পাহারাদার দরকার? ছবিতে লাল নোডগুলোতে পাহারাদার বসানো হয়েছে। এটা অপটিমাল না, নিচের ছবির মত বসালে পাহারাদার কম লাগত:



এটি একটি NP-hard প্রবলেম, অর্থাৎ এই প্রবলেমের কোনো পলিনমিয়াল টাইম সলিউশন নেই। তবে গ্রাফটি যদি **Tree** হয় অর্থাৎ  $n-1$  edge থাকে আব কোনো সাইকেল না থাকে তাহলে **ডাইনামিক প্রোগ্রামিং** বা ম্যাক্স ফ্লো/বাইপারটাইট ম্যাচিং এর সাহায্যে প্রবলেমটি সলভ করা সম্ভব।

## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

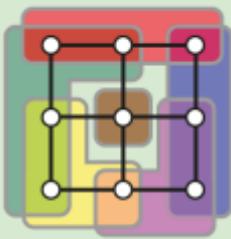
## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)

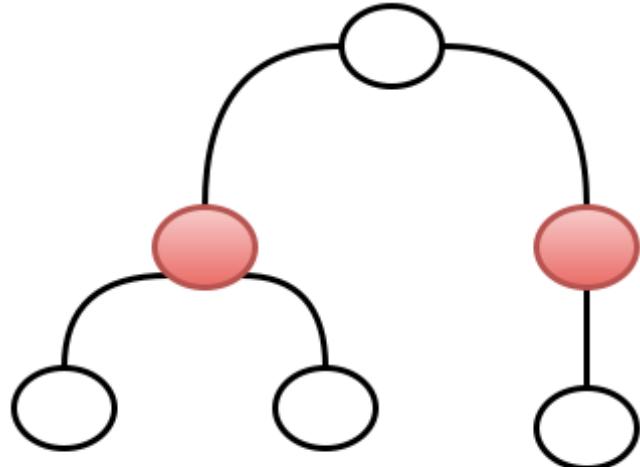
অ্যাপেলের বিজ্ঞান সর্কিন

### গ্রাফ অ্যালগরিদম

শাফায়েত আশরাফ



**d**  
বিজ্ঞান প্রকাশনা



Minimum Vertex Cover in a Tree

ডাইনামিক প্রোগ্রামিং সলিউশনটা আমি বিশ্লেষিত লিখছি, তারপর ম্যান্ড্রো/বাইপারটাইট ম্যাচিং দিয়ে কিভাবে করতে হয় লিখবো।

ডিপি সলিউশনে ২টি কেস আমাদের লক্ষ্য করতে হবে:

- “ ১. কোনো নোডে পাহাদার না বসালে তার সাথে সংযুক্ত সব নোডে অবশ্যই পাহাদার বসাতে হবে, এছাড়া সব বাস্তা কভার হবে না। অর্থাৎ যদি  $v$  আর  $v'$  সংযুক্ত থাকে তাহলে  $v$  তে পাহাদার না বসালে  $v'$  তে অবশ্যই বসাতে হবে।
- ২. কোনো নোডে পাহাদার বসালে সংযুক্ত নোডগুলোতে পাহাদার বাসানো বাধ্যতামূলক না তবে বসালে লাভ হতে পারে। তাই  $v$  তে পাহাদার বসালে  $v'$  তে পাহাদার একবার বসিয়ে এবং একবার না বসিয়ে দেখবো কোনটা লাভজনক



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

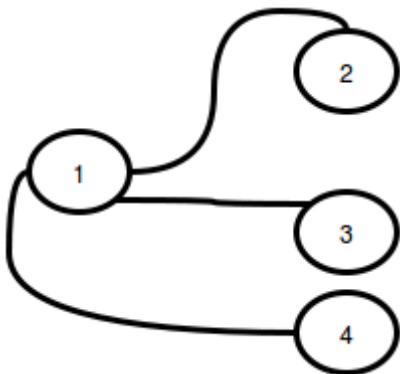
কিউ+সার্কুলার কিউ

সব ডিপির প্রবলেমের মতো এখানেও একটা রিকার্সিভ ফাংশন ডিফাইন করবো। আমাদের স্টেট হবে বর্তমানে কোন নোডে আছি, এবং সেই নোডে কোনো পাহারাদার বসানো হয়েছে নাকি।

“  $F(u,1) = \text{বর্তমানে } u \text{ নম্বর নোডে আছি এবং এই নোডে \text{পাহারাদার আছে।}$   $f(u,1)$  রিটার্ন করবে বাকি নোডগুলোতে মোট পাহারাদার সংখ্যা।

$F(u,0) = \text{বর্তমানে } u \text{ নম্বর নোডে আছি এবং এই নোডে \text{পাহারাদার নাই।}$   $f(u,0)$  রিটার্ন করবে বাকি নোডগুলোতে মোট পাহারাদার সংখ্যা।

ধরি 1 নম্বর নোডের সাথে 2, 3, 4 নম্বর নোড যুক্ত।



বুঝাই যাচ্ছে 1 নম্বর নোডে পাহারা না বসালে অবশ্যই 2, 3, 4 সবগুলোয় পাহারা বসাতে হবে। তাহলে আমরা বলতে পারি:

“  $F(1,0)=F(2,1)+F(3,1)+F(4,1) + 0$ , অর্থাৎ 1 এর সাথে সংযুক্ত সব নোডগুলোতে পাহারা বসালে প্রয়োজনীয় মোট পাহারাদার সংখ্যা।

সবশেষে 0 যোগ করছি কারণ বর্তমান নোডে পাহারাদার বসাইনি।

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাঈ(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

এবাব  $F(1,1)$  এর মান বের কৰি। ১ নম্বৰ নোডে পাহারা বসালে সংযুক্ত নোডগুলোতে পাহারা বসালেও চলে, না বসালেও চলে, তবে যেটা অপটিমাল বেজাট দেয় সেটা আমরা নিব:

$$F(1,1) = 1 + \min(F(2,1), F(2,0)) + \min(F(3,1), F(3,0)) + \min(F(4,1), F(4,0))$$

১ নম্বৰ নোডে পাহারাদার বসাচ্ছি তাই সবশেষে ১ যোগ হচ্ছে, প্রতি নোডে একবাব পাহারা বসিয়ে, আবাব না বসিয়ে দেখছি কোনটা অপটিমাল।

একটা ব্যাপার লক্ষ রাখতে হবে যে প্যারেন্ট নোড নিয়ে কখনো হিসাব করবোনা। উপরের ছবিতে ১ থেকে ২ এ গেলে  $\text{parent}[2]=1$ , তাই ২ থেকে আবাব ১ নম্বৰ নোডে যাবোনা।

এবাব base case এ আসি। কোনো নোড থেকে নতুন কোনো নোডে যাওয়া না গেলে ১ বা ০ রিটার্ন কৰে দিতে হবে, পাহারাদার বসালে ১, না বসালে ০। কোনো ট্রি তে একটি মাত্র নোড থাকলে ১ রিটার্ন কৰতে হবে(কিছু প্রবলেমে ০ ও রিটার্ন কৰতে হতে পাৰে)।

Spoj এর [PT07X\(vertex cover\)](#) প্রবলেমটি straight forward প্রবলেম। এটার জন্য আমাৰ কোডটা এৰকম:

```

1 #define MAXN 100002
2 int dp[MAXN][5];
3 int par[MAXN];
4 vectoredges[MAXN];
5
6 int f(int u, int isGuard)
7 {
8     if (edges[u].size() == 0)
9         return 0;
10    if (dp[u][isGuard] != -1)
11        return dp[u][isGuard];
12    int sum = 0;
13    for (int i = 0; i < (int)edges[u].size(); i++) {
14        int v = edges[u][i];
15        if (v != par[u]) {
16            par[v] = u;
17            if (isGuard == 0)
18                sum += f(v, 1);
19            else
20                sum += min(f(v, 1), f(v, 0));
21    }
22    dp[u][isGuard] = sum;
23    return sum;
24}
```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লাৱ ট্ৰ্যুৱ মিনিমাম ডারটেক্স কভাৱ

ট্ৰি এৰ ডায়ামিটাৱ নিৰ্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগৱিদম গেম থিওৰি(৩):

গেম থিওৰি-১

গেম থিওৰি-২

গেম থিওৰি-৩

ডাইনামিক প্ৰোগ্ৰামিং(৮):

ডাইনামিক প্ৰোগ্ৰামিং ১ (ফিবোনাচি)

ডাইনামিক প্ৰোগ্ৰামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্ৰোগ্ৰামিং ৩ (LIS এবং পাথ প্ৰিস্টিং)

ডাইনামিক প্ৰোগ্ৰামিং ৪ (লংগেস্ট কমন সাবসিকোয়েম)

ডাইনামিক প্ৰোগ্ৰামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্ৰোগ্ৰামিং ৬ (সাবসেট সাম, কম্পিনেটোৱিঙ্গ, ডিসিশন প্রবলেম)

```

21     }
22 }
23     return dp[u][isGuard] = sum + isGuard;
24 }
25
26 int main()
27 {
28     memset(dp, -1, sizeof(dp));
29     int n;
30     scanf("%d", &n);
31     for (int i = 1; i < n; i++) {
32         int u, v;
33         scanf("%d%d", &u, &v);
34         edges[u].push_back(v);
35         edges[v].push_back(u);
36     }
37     int ans = 0;
38     ans = min(f(1, 1), f(1, 0));
39     printf("%d\n", ans);
40     return 0;
41 }

```

আমি ট্রি এর root সবসময় ১ ধরে কোড লিখেছি। ৩৮ নম্বর লাইনে মেইন ফাংশনে root এ পাহারাদার একবার বসিয়ে আর একবার না বসিয়ে অপটিমাল বেজাল্ট টা নিছি।

ফাংশনে u হলো current node, isguard কারেন্ট নোডে পাহারাদার আছে নাকি নাই সেটা নির্দেশ করে।

১০ নম্বর লাইনে ট্রি এর সাইজ ১ হলে ১ রিটার্ন করে দিয়েছি।

১৩ নম্বর লাইনে লুপের ভিতর current নোড থেকে সবগুলো child নোডে যাচ্ছি। কারেন্ট নোডে পাহারাদার না থাকলে পরেরটায় বসাচ্ছি, আর থাকলে ২ভাবেই চেষ্টা করছি। ১৫ নম্বর লাইনের কভিশন দিয়ে প্যারেট নোডে যেতে দিচ্ছিনা।

সবশেষে sum+isGuard রিটার্ন করছি। অর্থাৎ কারেন্ট নোডে পাহারাদার থাকলে 1 যোগ করছি, নাহল 0।

মোটামুটি এই হলো ডিপি সলিউশন। ট্রি তে সাইকেল না থাকায় এটা অবশ্যই বাইপারটাইট গ্রাফ। ১৯৩১ সালে Dénes König প্রমাণ করেন কোনো বাইপারটাইট গ্রাফে maximum matching=minimum vertex cover। এটা গ্রাফ থিওরির অনেক min-max থিওরেমের একটা যেখানে কিছু একটা ম্যাক্সিমাইজ করলে অন্য আরেকটা কিছু মিনিমাইজ হয়। তুমি যদি ম্যাক্সিমাম ম্যাচিং এর অ্যালগোরিদম জানো তাহলে

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাকিং(১):

ব্যকট্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

টি টা বাইকালারিং করে ম্যাচিং বের করলেই ভারটেক্স কভার বের হয়ে যাবে। কোড সহজ হলেও complexity বেড়ে যাবে, তাই নোড বেশি থাকলে কাজ করবেনা। আবার ম্যাক্সিমাম ম্যাচিং যেহেতু ম্যাত্র-ফ্লো এর একটি ড্যারিয়েশন তাই ফ্লো চালিয়েও সমাধান করা সম্ভব।

এরকম আরেকটা প্রবলেম [uva-10243\(fire fire fire\)](#)। আমি প্রথমে এটা সমাধান করে পরে spoj এর টা করেছি।

## ফেসবুকে মন্তব্য

1 comments

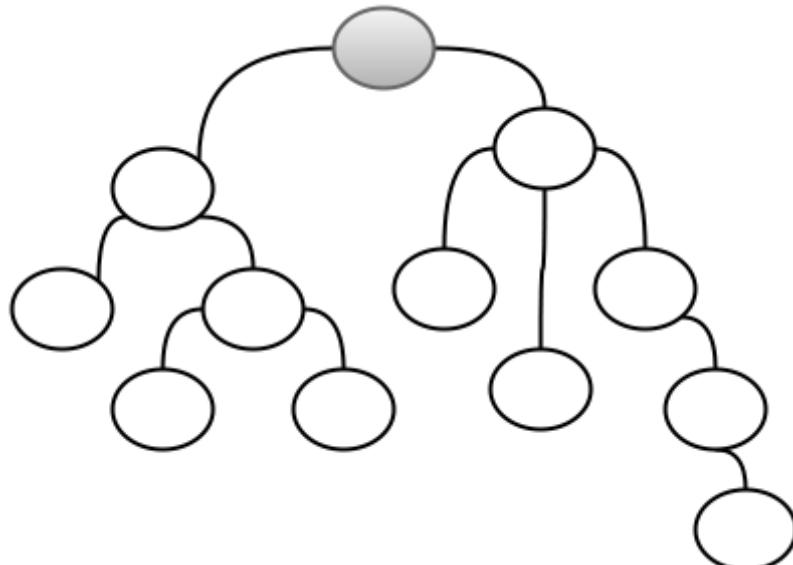
# ট্রি ডায়ামিটার

জানুয়ারি ৮, ২০১৪ by শাফায়েত



ট্রি হলো এমন একটা আনডিরেষ্টেড গ্রাফ যেটার সব নোড থেকে সব নোডে যাওয়া যায় এবং কোনো সাইকেল নেই। এখন আমাদের ট্রি এর সবথেকে দূরের দুটা নোড খুজে বের করতে হবে, একেই বলা হ্য ট্রি এর ডায়ামিটার।

মনে করো বিছু কম্পিউটারের মধ্যে নেটওয়ার্ক কেবল লাগানো হয়েছে নিচের ছবির মতো করে। এখন তুমি জানতে চাইতেই পারো কোন দুটি কম্পিউটার সবথেকে দূরে আছে।



## সাবস্ক্রাইব

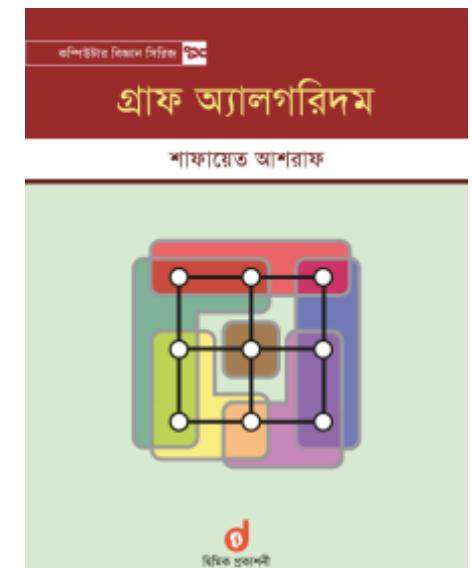


Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

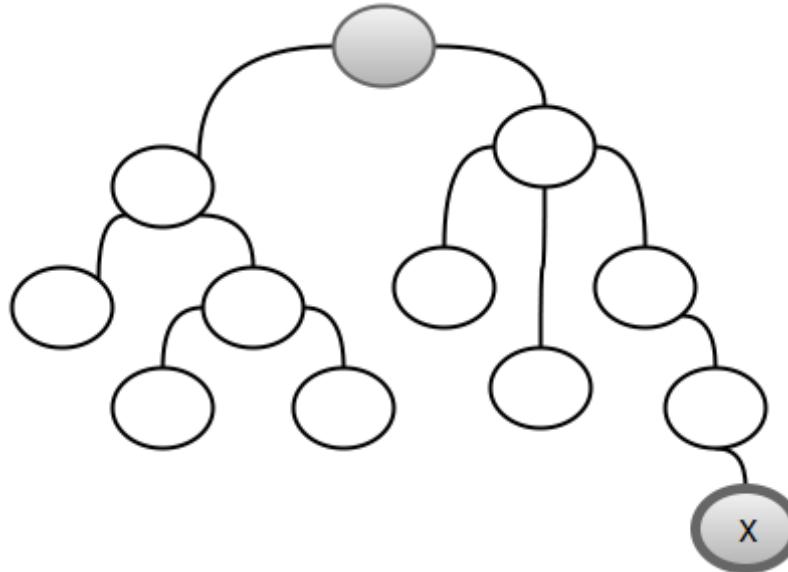
শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

**Traveloka Singapore** (বিস্তারিত...)



এটা বের করা খুব সহজ, এজন্য তোমার জানতে হবে বিএফএস বা ডিএফএস এর যে কোন একটা। আনডিরেটেড ট্রি তে যেকোন নোডকেই রুট ধরা যায়, আমরা মনে করি উপরের ধূসর নোডটা ট্রি এর রুট।

আমাদের প্রথম কাজ কাজ হলো রুট হতে সবথেকে দূরের নোডটা খুজে বের করা। সেই নোডটাকে মনে করি X। একাধিক নোডের দূরস্থ সবথেকে দূরের নোডের দূরস্থের সমান হলেও সমস্যা নেই, যেকোন একটাকে সিলেক্ট করতে হবে। এই কাজটা আমরা ডিএফএস/বিএফএস চালিয়ে বের করতে পারি।



এখন ২য় কাজ হলো X নোড থেকে শুরু আবেকটি ডিএফএস/বিএফএস চালিয়ে X থেকে সবথেকে দূরের নোড খুজে বের করা। মনে করি নোডটা হলো Y।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

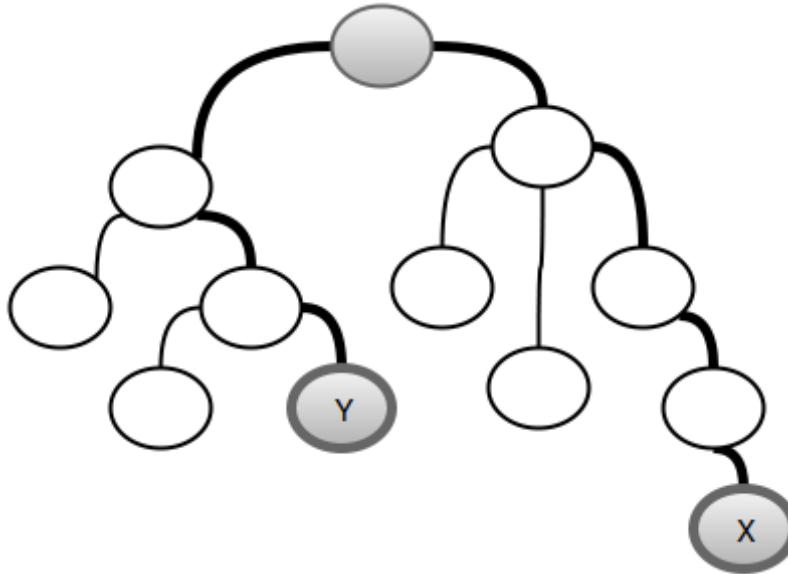
ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



X আর Y এর মধ্যকার দূরবর্তী ট্রি এর ডায়ামিটার! উপরের ছবিতে ডায়ামিটার ৭।

#### প্রমাণ:

আমরা যদি প্রমাণ করতে পারি ১ম ধাপে খুজে পাওয়া X সবসময়ই ডায়ামিটারের একটা প্রান্ত হবে তাহলেই অ্যালগোরিদমটা প্রমাণ হয়ে যায়। কারণ

X যদি নিশ্চিতভাবে ডায়ামিটারের একটা প্রান্ত হয় তাহলে ২য় ধাপটা অবশ্যই সঠিক, X থেকে সবথেকে দূরের নোডই হবে ট্রি এর ডায়ামিটার।

এটা আমরা প্রমাণ করবো “প্রফ বাই কন্ট্রাডিকশন” এর সাহায্যে। এটা বহুল ব্যবহৃত একটা পদ্ধতি। আমরা প্রমাণ করতে চাই X নিশ্চয়ই কোনো ডায়ামিটারের প্রান্ত। কিন্তু তা যদি না হয়, অর্থাৎ X কোনো ডায়ামিটারের প্রান্ত না হলে এমন একটা ডায়ামিটার থাকবে যার দুই প্রান্ত (ধরি) a এবং b, এখনে a এবং b যেকোনো দুটি নোড হতে পারে(X ছাড়া)। a-b ডায়ামিটার ব্যবহার করে আমরা দেখাবো এমন আরেকটা ডায়ামিটার পাওয়া সম্ভব যেটা a-b এর চেয়ে বড় বা সমান এবং যার এক প্রান্তে X আছে। (উল্লেখ্য, X হলো রুট থেকে সবচেয়ে দূরের একটা নোড)।

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

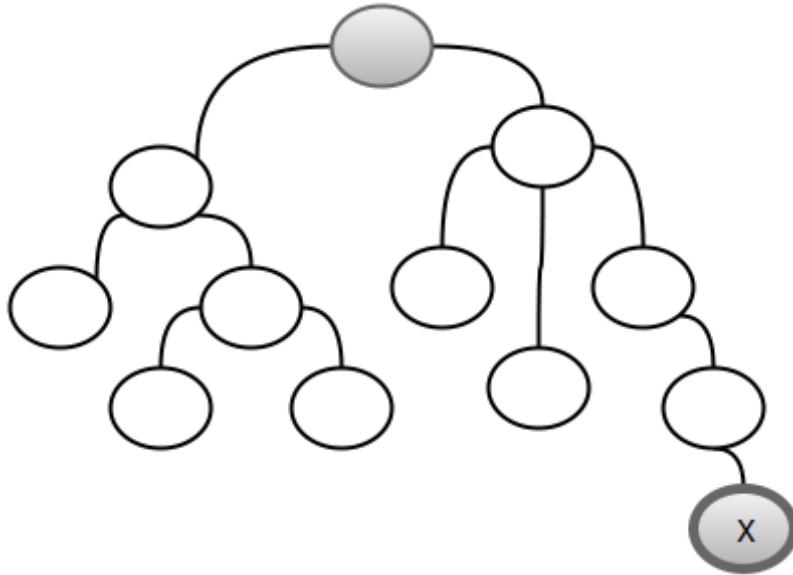
টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



এখন **আর্বিট্রালি(arbitrarily)** দুটি নোড a আর b সিলেক্ট করি। নোড দুটির মধ্যকার পাথ রুটের কাছে যে নোডে এসে মিলবে সেটাৰ নাম দেই h, অর্থাৎ h হলো নোড দুইটাৰ কমন অ্যানসেস্টর। (অনেকেই হয়তো বুঝে গেছো আমোৱা এখনে **lowest common ancestor** এৰ কথা বলছি)।

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লাৱ ট্ৰুৱ মিনিমাম ডারটেক্স কভাৱ

ট্ৰি এৰ ডায়ামিটাৰ নিৰ্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগৱিদিম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্ৰোগ্ৰামিং(৮):

ডাইনামিক প্ৰোগ্ৰামিং ১ (ফিবোনাচি)

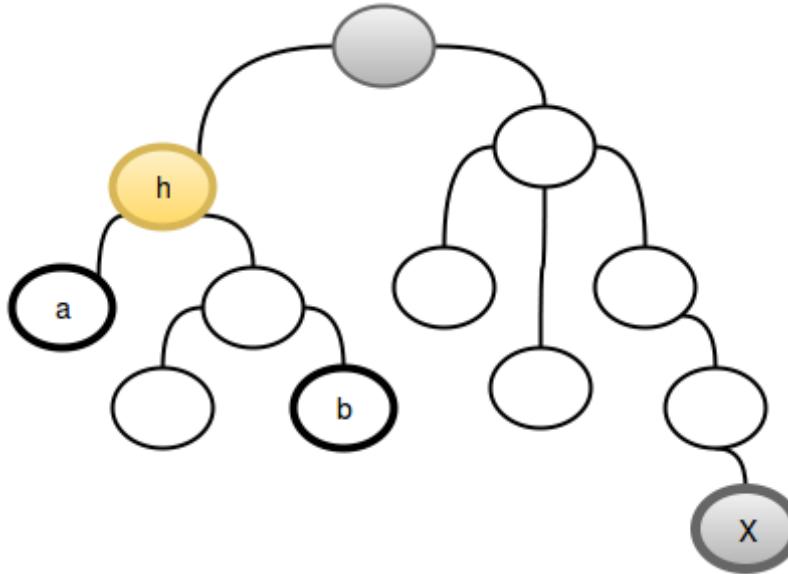
ডাইনামিক প্ৰোগ্ৰামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্ৰোগ্ৰামিং ৩ (LIS এবং পাথ প্ৰিন্টিং)

ডাইনামিক প্ৰোগ্ৰামিং ৪ (লংগেস্ট কমন  
সাৰসিকোয়েম)

ডাইনামিক প্ৰোগ্ৰামিং ৫ (কয়েন চেঞ্জ/০-১  
ন্যাপস্যাক)

ডাইনামিক প্ৰোগ্ৰামিং ৬ (সাৰসেচ সাম,  
কম্বিনেটোৱিঙ্গ, ডিসিশন প্রবলেম)



কিন্তু রুট থেকে b এর দূরত্ব, রুট থেকে X এর দূরত্বের কম বা সমান হতে বাধ্য, বেশি হবেনা কারণ রুট থেকে সবথেকে দূরের নোড হলো X।

অর্থাৎ:

$$\text{“ } \text{distance}(\text{root}, b) \leq \text{distance}(\text{root}, X)$$

আবার এটাও নিশ্চিত যে রুট থেকে b যত দূরে, h থেকে b এর দূরত্ব তার থেকে কম বা সমান। (সমান হবে যদি h আর রুট একই নোড হয় অর্থাৎ a আর b যদি রুটের দুইপাশে থাকে)

$$\text{“ } \text{distance}(h, b) \leq \text{distance}(\text{root}, b)$$

তাহলে,

$$\text{distance}(h, b) \leq \text{distance}(\text{root}, b) \leq \text{distance}(\text{root}, X)$$

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্স(১):

ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

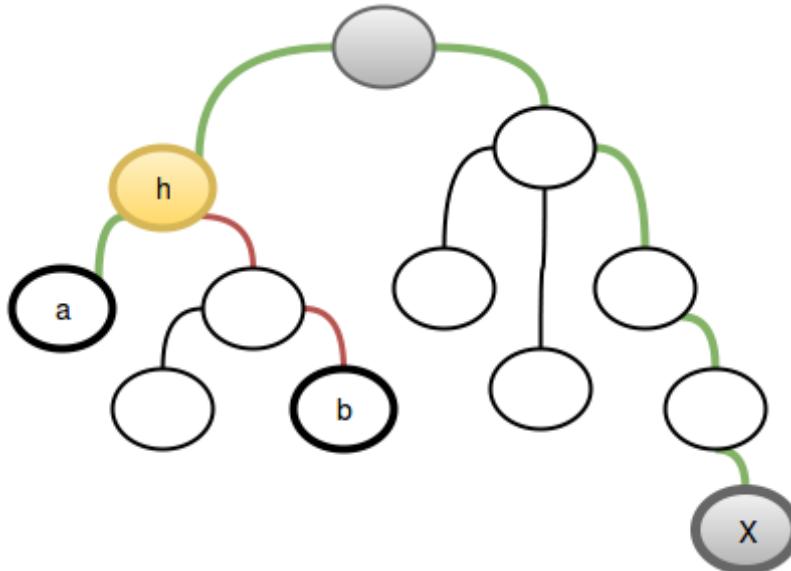
টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

$distance(h,b) \leq distance(root,X)$

“

তাহলে আমরা h-b পাথটা root-X পাথ দিয়ে রিস্পেস করে দিলে এবং h-x এর মধ্যে পাথ বসিয়ে দিলে(যদি h আর x সমান না হয়) অবশ্যই আগের সমান বা আগের থেকে বড় একটা পাথ পাবো!



তারমানে আমরা যে X কে বাদ দিয়ে যেই দুইটা নোডই সিলেক্ট করিনা কেন, X কে নিয়ে তার থেকে বড় বা সমান একটা পাথ পাওয়া যাবে। তারমানে X অবশ্যই ডায়ামিটারের একটা প্রাণ্ট!

#### কমপ্লেক্সিটি:

বিএফএস/ডিএফএস এর কমপ্লেক্সিটির সমান:  $O(V+E)$  যেখানে V হলো নোডসংখ্যা এবং E হলো এজ সংখ্যা।

এই অ্যালগোরিদমটা জেনারেল গ্রাফে কাজ করবেনা, শুধু ট্ৰি এৰ ক্ষেত্ৰে কৰবে। জেনারেল গ্রাফে লংগেস্ট পাথ বেৱ কৰাৰ প্ৰবলেম **এন-পি হার্ড**, আৱ ডায়ামিটাৰ বেৱ কৰাৰ প্ৰবলেম লংগেস্ট পাথ

কোয়ান্টাম কম্পিউটাৰ কী?

কোয়ান্টাম কম্পিউটাৰেৰ শক্তি এবং সীমাবদ্ধতা

 Shafaetsplanet  
2.4K likes  
শাফায়েত আশৰাফ

[Like Page](#)

Be the first of your friends to like this

---

 Shafaetsplanet about an hour ago  
ডাইনামিক প্ৰোগ্ৰামিং এৰ নতুন  
সিৱিজেৰ ৬ নম্বৰ লেখায় আমৰা  
দেখবো কম্বিনেটৱিঙ্গেৰ প্ৰবলেম এবং  
ডিসিশন প্ৰবলেম কিভাৱে সমাধান  
কৰতে হ্য।

---

*i, W) = 1 if W =  
i + 1, W - C[i]) -*

SHAFAEETPLANET.COM  
ডাইনামিক প্ৰোগ্ৰামিং ৬ / চ্যাপ

প্রবলেমেরই স্পেশাল কেস।

“

Farthest Nodes in a Tree

Farthest Nodes in a Tree (II)

হ্যাপি কোডিং!

কৃতজ্ঞতা শীকার:

<http://stackoverflow.com/questions/20010472/proof-of-correctness-algorithm-for-diameter-of-a-tree-in-graph-theory>

<http://apps.topcoder.com/forums/?module=Thread&threadID=668470&start=0&mc=12#1216875>

ফেসবুকে মন্তব্য

0 comments

# লংগেস্ট পাথ প্রবলেম

জুলাই ২১, ২০১৬ by শাফায়েত

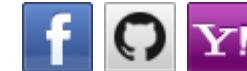


তোমাকে একটা আনওয়েটেড গ্রাফ এবং একটা সোর্স মোড দেয়া আছে। তোমাকে সোর্স মোড থেকে সর্বোচ্চ দৈর্ঘ্যের পাথ বের করতে হবে। এটাই হলো লংগেস্ট পাথ প্রবলেম। প্রশ্ন হলো কিভাবে প্রবলেমটা সলভ করবে?

এটা আমার খুবই প্রিয় একটা ইন্টারভিউ প্রশ্ন। এখন পর্যন্ত প্রায় ৭-৮টা ইন্টারভিউতে আমি এই প্রশ্ন করেছি, মাত্র ২জন সহজেই উত্তর দিতে পেরেছে, ১জন হিটস দেয়ার পর পেরেছে, বাকিরা সবাই ডুল উত্তর দিয়েছে। অ্যালগোরিদম কোর্স এ+ অনেকেই পায়, কিন্তু এধরণের প্রশ্ন করলে বোঝা যায় ক্যান্ডিডেট আসলে কতখানি জানে।

প্রশ্নটা করার পর সবাই প্রথমে যে ডুল করে সেটা হলো জিঞ্জেস করে না সর্বোচ্চ দৈর্ঘ্যের পাথের সংজ্ঞা কি, আমি চাইলে দুটি মোডের মধ্যে অসীম সংখ্যকবার আসা-যাওয়া করে দৈর্ঘ্য অসীম করে ফেলতে পারি। তাই লংগেস্ট পাথের আরো ভালো কোনো সংজ্ঞা দরকার সমাধান করা জন্য। ধরা যাক আমি এমন একটা লংগেস্ট পাথ চাই যেখানে কোনো মোডে একবারের বেশি যাওয়া যাবে না। এখন আর দৈর্ঘ্য অসীম হবার কোনো সুযোগ নেই।

## সাবস্ক্রাইব

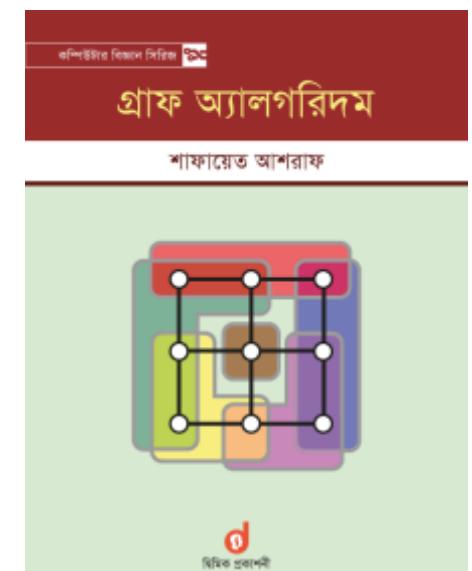


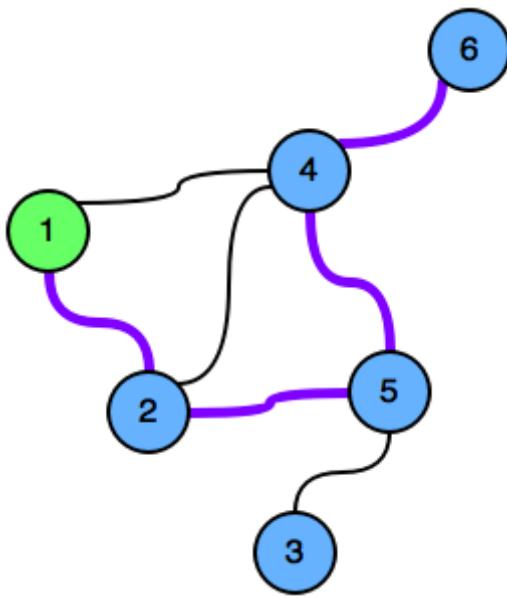
Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

**Traveloka Singapore** (বিস্তারিত...)





ছবিতে ১ নম্বর নোড থেকে সর্বোচ্চ দৈর্ঘ্যের পাথ দেখানো হয়েছে যার দৈর্ঘ্য হলো ৮ (১->২->৫->৮->৬)।

সমাধান নিয়ে চিন্তা করার আগে পরের প্রশ্ন হলো গ্রাফটা কি ডিরেক্টেড নাকি আনডিরেক্টেড। ধরে নাও গ্রাফটা আনডিরেক্টেড এবং সর্বোচ্চ ১০০,০০০ টা নোড থাকতে পারে।

এই প্রবলেমের অনেক ধরণের ভূল সমাধান দিতে দেখেছি ক্যান্ডিডেটদের, যেমন:

কেও কেও গ্রাফের এজগুলোর ওয়েট -১ ধরে শর্টেস্ট পাথ বের করার চেষ্টা করে। এটা কোনোভাবেই কাজ করবে না, ইনফিনিটি লুপে আটকে যাবে।

কেও কেও ডায়নামিক প্রোগ্রামিং ব্যবহার করে সমাধান করতে চেষ্টা করে, এটাও কাজ করবে না এবং লুপে আটকে যাবে।

ম্যাত্র ফ্রো ব্যবহার করতে চায় অনেকে কিন্তু সেটাও কাজ করবে না।

তাহলে সমাধান টা কি? মনে করি গ্রাফটার নোড সংখ্যা  $n$ । যেহেতু  $n$  এর মান অনেক বড় হতে পারে, তাই অবশ্য পলিনোমিয়াল সলিউশন দরকার। এখন এমন একটা গ্রাফের কথা চিন্তা করো যেটায় সোর্স থেকে লংগেস্ট পাথের দৈর্ঘ্য হলো  $n - 1$ । যেমন নিচের গ্রাফটা:



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৮):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

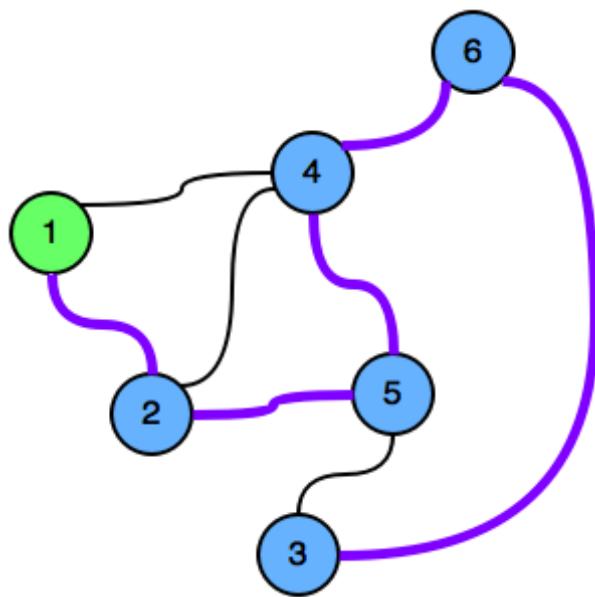
ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



এই গ্রাফটিতে লংগেস্ট পাথের দৈর্ঘ্য হলো  $n - 1$  যখনে  $n = 6$ । তারমানে হলো গ্রাফটাতে এমন একটা সোর্স নোড আছে যেখান থেকে যাত্রা শুরু করে সবগুলো নোড একবার করে ভ্রমণ করা যায় কোনো নোড পুনরাবৃত্তি না করেই। এ ধরণের পাথকে বলা হয় **হ্যামিল্টন পাথ**! তুমি যদি গ্রাফথিওরি নিয়ে কিছুটা পড়ালেখা করে থাকো তাহলে অবশ্য জানার কথা যে হ্যামিল্টন পাথ প্রবলেম একটা **এন-পি কমপ্লিট (NP-complete)** প্রবলেম। এন-পি কমপ্লিট কোনো প্রবলেমকে পলিনমিয়াল সময়ে সমাধান করার কোনো উপায় এখন পর্যন্ত কারো জানা নেই।

দেখতেই পাচ্ছা হ্যামিল্টন পাথ প্রবলেমকে খুব সহজেই লংগেস্ট পাথ প্রবলেমে কনভার্ট করা যায়। একটা গ্রাফে কোনো সোর্স থেকে এমন একটা লংগেস্ট পাথ বের করতে হবে যেন পাথের দৈর্ঘ্য হয়  $n - 1$ , এটাই হ্যামিল্টন পাথ প্রবলেম। লংগেস্ট পাথ প্রবলেমকে পলিনমিয়াল সময়ে সমাধান করা গেলে হ্যামিল্টন পাথ প্রবলেমও সমাধান হয়ে যেত!

তাহলে দেখতেই পাচ্ছি যে লংগেস্ট পাথ প্রবলেমের কোনো পলিনমিয়াল সলিউশন নেই। নোড সংখ্যা যদি খুব অল্প হয় তাহলে **ব্যাকট্যুকিৎ** করে এই সমস্যা সমাধান করা সম্ভব, কিন্তু এটার কমপ্লেক্সিটি এক্সপোনেনশিয়াল(exponential)। লংগেস্ট পাথ হলো একটা এন-পি হার্ড প্রবলেম।

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)  
ডিসজয়েট সেট(ইউনিয়ন ফাইল)  
ট্রাই(প্রিফিল ট্রি/বেডিল ট্রি)  
সেগমেন্ট ট্রি-১  
সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)  
অ্যারে কমপ্রেশন/ম্যাপিং  
লোয়েস্ট কমন অ্যানসেস্টর  
বাইনারি ইনডেক্সড ট্রি  
স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:  
এল-আর-ইউ ক্যাশ(**নতুন**)  
ব্লুম ফিল্টার(**নতুন**)

**গ্রাফ থিওরি(২০):**  
গ্রাফ থিওরিতে হাতেখড়ি  
অ্যাডজেসেন্সি ম্যাট্রিক্স  
অ্যাডজেসেন্সি লিস্ট  
ব্রেথড ফার্স্ট সার্চ (বিএফএস)  
মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)  
মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)  
টপোলজিকাল সর্ট  
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
সর্ট  
ডায়াক্রস্টা  
ফ্লয়েড ওয়ার্শল  
বেলম্যান ফোর্ড

# অ্যালগরিদম গেম থিওরি – ১

অক্টোবর ২২, ২০১৫ by শাফায়েত

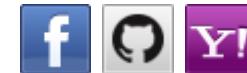


আমরা বাস্তবে যে সব খেলাখুলা করি সেগুলোতে আমরা খেলার শুরুতেই বলে দিতে পারি না কে খেলাতে জিতবে, আমরা এটাও ধরে নিতে পারি না যে সব খেলোয়াড়ই সবসময় সেৱা চাল দিবে। এছাড়া অনেক খেলায় ডাগ্যেরও সহায়তা দরকার হয়। যেমন তাস খেলায় আমরা জানি না প্রতিপক্ষের কাছে কি কি কার্ড আছে, বা লুভু খেলায় আমরা জানি না ছক্ষা বা ডাইস এ কখন কোন সংখ্যাটা আসবে। এই সিরিজে সময় আমরা মূলত মাত্র এমন সব গেম নিয়ে কাজ করবো যাব নিচের বৈশিষ্ট্যগুলো আছে:

১. গেমের বোর্ড, চাল ইত্যাদি সম্পর্কে পূর্ণসং তথ্য আমদের কাছে আছে, প্রতিপক্ষ কি অবস্থায় আছে সেটাও আমরা জানি।
২. খেলায় কোনো ডাগ্যের সহায়তা দরকার হয় না।
৩. খেলা শেষে কেও একজন জিতবে বা হারবে।
৪. খেলায় দুইজন মাত্র খেলোয়াড় থাকবে, প্রথম ও দ্বিতীয় খেলোয়াড় পালাবদল করে চাল দিতে থাকবে।
৫. দুইজন খেলোয়াড়ই খেলার যেকোনো সময় সবথেকে সেৱা চালটি দিবে, কেও কোনো ভুল চাল দিবে না। কোনো খেলোয়াড়ের জয়ের সম্ভাবনা থাকলে সে অবশ্যই জিতবে, সম্ভাবনা না থাকলে প্রতিপক্ষের ভুলে সে কখনো জিততে পারবে না। এককথায়, প্রতিটা খেলোয়াড় সবসময় 'অপটিমাল' চালটি দিবে।

আমরা যে ধরণের গেমগুলো দেখবো সেগুলো উপরের শর্তগুলো মেনে চলে, অ্যালগরিদম গেম আৰো অনেক বুকম হতে পাৰে। এই লেখাটা পড়াৰ আগে **ডাইনামিক প্ৰোগ্ৰামিং** এৰ বেসিক নিয়ে জানা থাকলে বুৰুতে সুবিধা হবে।

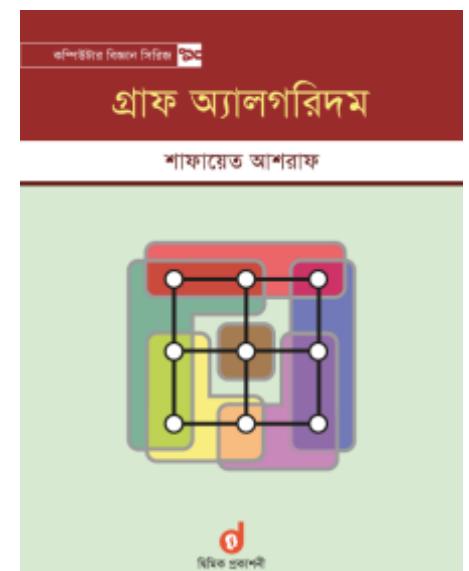
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



এই ধরনের খেলায় একদম শুরুতে বা খেলার মাঝে খেলার নিয়মগুলো দেখে এবং দুই খেলোয়াড়ের বর্তমান অবস্থা দেখেই বলে দেয়া যায় যে দুইজনই অপটিমাল চাল দিলে কে খেলায় জিতবে। বাস্তবে এ ধরনের খেলা বোরিং মনে হলেও গেম থিওরির বাস্তব জীবনে অনেক ব্যবহার আছে, অর্থনীতি, রাজনীতি, জীববিজ্ঞান ইত্যাদিতে গেম থিওরি অনেক ব্যবহার হয়, তবে এই টিউটোরিয়ালে আমরা গেম থিওরির একটা ছোটো সাবসেট ‘অ্যালগরিদম গেম’ নিয়ে জানবো।

ছোটোবেলায় টিভিতে ম্যাগাজিন অনুষ্ঠানে একটা খেলা দেখেছিলাম, টেবিলে কিছু মার্বেল রাখা থাকে, একজন দর্শককে স্টেজে নিয়ে আসা হয়। দর্শকের সাথে উপস্থাপক একটা খেলা খেলে এরকম, প্রথম চালে দর্শক বা ১ম খেলোয়াড় ২টা, ৩টা বা ৫টা মার্বেল তুলে নিতে পারবে, পরের চালে উপস্থাপক বা ২য় খেলোয়াড় ২, ৩ বা ৫টা মার্বেল তুলে নিতে পারবে, এরপর দর্শক আবার একই ভাবে মার্বেল তুলবে। এভাবে চলতে থাকবে যতক্ষণ না কারো পক্ষে আর চাল দেয়া সম্ভব না হয়। শেষবার যে মার্বেলগুলো তুলেছে সে খেলাটা জিতে যাবে।

অনুষ্ঠানে দেখা যেত একের পর এক দর্শক আসছে আর হেবে যাচ্ছে, কোনোভাবেই উপস্থাপককে হারানো সম্ভব হচ্ছে না। আমরা এখন খেলাটাকে বিশ্লেষণ করলেই বুঝে যাবো কেন তাকে হারানো সম্ভব হয় নি সবথেকে বুদ্ধিমান দর্শকের পক্ষেও।

এই খেলায় খেলোয়াররা কোনো সময় কি অবস্থানে আছে সেটা জানার জন্য আমাদের দুইটা তথ্য দরকার, বর্তমানে কয়টা মার্বেল আছে, এবং কার চাল দেয়ার পালা। যদি ৫টা মার্বেল থাকে তাহলে এখন যে খেলোয়াড়ের চাল দেয়ার পালা সে একটা ‘উইনিং পজিশন’ বা ‘বিজয়ী অবস্থা’ তে চলে গিয়েছে, সে কোনো ভুল না করলে তাকে আর হারানো সম্ভব না, ৫টা মার্বেল তুলে নিয়ে সে খেলাটা শেষ করে দিবে। কিন্তু টেবিলে যদি ১টা বা ০টা মার্বেল থাকে তাহলে যে খেলোয়াড়ের চাল দেবার পালা সে ‘লুজিং পজিশন’ বা ‘পরাজিত অবস্থা’ য চলে গিয়েছে। আমরা এখন দেখবো যে টেবিলে মার্বেলের সংখ্যা যতই হোক না কেন, প্রথম খেলোয়ার অর্থাৎ যে এখন চাল দিবে পরাজিত অবস্থা না বিজয়ী অবস্থায় আছে আগেই জেনে যাওয়া সম্ভব। আমরা ধরে নিচ্ছি কোনো খেলোয়াড়ই ভুল চাল দিবে না।

মনে করি টেবিলে মার্বেল আছে  $n$  টা।

তৃটা জিনিস আমাদের মাথায় রাখতে হবে:

১. যেসব অবস্থা থেকে আর কোনো চাল দেয়া সম্ভব না সেগুলো পরাজিত অবস্থা বা লুজিং পজিশন। যেমন এই খেলায়  $n = 1$  এবং  $n = 0$  হলো লুজিং পজিশন। এগুলোকে টার্মিনাল পজিশন বলা হয় কারণ এই অবস্থা থেকে আর কোনো চাল দেয়া সম্ভব না।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগোরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্কেল(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইনেক্সেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

২. যদি আমরা টার্মিনাল পজিশনে না থাকি তাহলে দেখতে হবে কোনো চাল দিয়ে প্রতিপক্ষকে একটা লুজিং পজিশনে ফেলে দেয়া যায় নাকি। যদি যায় তাহলে বর্তমান খেলোয়ার উইনিং পজিশন এ আছে। যেমন  $n = 5$  হলে আমি চাইলে ২, ৩ বা ৫টি মার্বেল তুলে নিয়ে টেবিলে ৫-২=৩টি অথবা  $5 - 3 = 2$ টি অথবা  $5 - 5 = 0$  টি মার্বেল রেখে দিতে পারি। আমরা আগেই জানি  $n = 0$  একটা লুজিং পজিশন যেখান থেকে কোনো খেলোয়াড়ের পক্ষে জেতা সম্ভব না। যেহেতু টেবিলে ৫টি মার্বেল থাকলে প্রতিপক্ষকে লুজিং পজিশনে নিয়ে যাওয়া সম্ভব হচ্ছে,  $n = 5$  একটা উইনিং পজিশন, বর্তমান খেলোয়াড় কোনো ভুল না করলে তাকে এই অবস্থা থেকে কিছুতেই হারানো সম্ভব না।

৩. বর্তমান অবস্থা থেকে কোনো চাল দিয়ে প্রতিপক্ষকে একটা লুজিং পজিশনে নিয়ে যাওয়া না গেলে বর্তমান খেলোয়ার লুজিং পজিশনে আছে, প্রতিপক্ষ ভুল না করলে তার পক্ষে আর কিছুতেই জেতা সম্ভব না।

এখন আমরা সহজেই যে কোনো  $n$  এর জন্য বর্তমান খেলোয়াড় কি অবশ্য আছে বের করে ফেলতে পারি নিচের কোড দিয়ে:

```

1 def can_win(n):
2     if n==0 or n==1: return 0
3     if(can_win(n-2)==0 or can_win(n-3)==0 or can_win(n-5)==0):
4         return 1
5     return 0

```

আমরা প্রতিটা  $n$  এর জন্য নিচের মত আউটপুট পাবো:

n	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
can_win(n)	0	0	1	1	1	1	1	0	0	1	1	1	1	1	0	0

০ মানে হলো লুজিং পজিশন, ১ মানে উইনিং। এখান থেকে স্পষ্ট যে  $n = 15$  এর জন্য বর্তমান খেলোয়াড় কখনো জিততে পারবে না। ১৫টি মার্বেল থেকে ২, ৩ বা ৫টি মার্বেল তুলে আমরা ১৩, ১২ বা ১০টা মার্বেল রেখে দিতে পারি, কিন্তু টেবিল থেকে দেখা যাচ্ছে সবগুলোই উইনিং পজিশন, তাই বর্তমান অবস্থা থেকে জেতা সম্ভব না। টিভির সেই অনুষ্ঠানে যদি উপস্থাপক ১৫টি মার্বেল নিয়ে দর্শককে প্রথম খেলোয়াড় বানিয়ে খেল শুরু করে এবং সে যদি বুদ্ধিমান হয় তাহলে তাকে হারানো অসম্ভব!

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

আমি নিশ্চিত এখন তোমরা বুঝে গিয়েছো কিভাবে এধরণের সমস্যা সমাধান করতে হয়। চিন্তা করার মতো কিছু সমস্যা দিলাম, সবগুলোই উপরের সমস্যাটার মতো:

১. এলিস আর বব একটা খেলা খেলছে, তাদের কাছে একটা সংখ্যা আছে  $p = 1$ । প্রতি চালে একজন খেলোয়ার  $p$  এর সাথে ২ থেকে ৯ এর ভিতর একটা সংখ্যা গুণ করতে পারে। গুণ করার পর সংখ্যাটা  $x$  এর থেকে বড় হয়ে গেলে সে জিতে যাবে।  $x$  এর মান দেয়া আছে, এলিস সবসময় প্রথম চাল দেয়। খেলায় কে জিতবে? (UVA 847)

২. UVA 11489

৩. আবেকটু কঠিন সমস্যা সমাধান করতে চাইলে UVA 10891

পরের পরে অ্যালগরিদম গেম এর সবথেকে গুরুত্বপূর্ণ বিষয় “নিম গেম” নিয়ে আলোচনা করবো।

রেফারেন্স:

<https://www.topcoder.com/community/data-science/data-science-tutorials/algorithm-games/>

ফেসবুকে মন্তব্য

0 comments

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এবং ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)

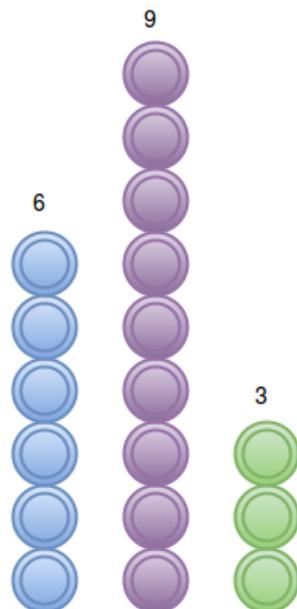
# অ্যালগরিদম গেম থিওরি-২ (নিম গেম)

জানুয়ারি ৩, ২০১৬ by [শাফায়েত](#)



আগের পর্বে গেম থিওরি কিছু বেসিক শিখেছি, এবাবের পর্বে আমরা জানবো নিম-গেম নিয়ে। নিম-গেম খুবই গুরুত্বপূর্ণ কারণ অনেক ধরণের গেমকে নিম গেম এ রূপান্তর করে ফেলা যায়।

নিম-গেম এ দুইজন খেলোয়ার আর কিছু পাথরের স্টপ(pile) থাকে। প্রতি চালে একজন খেলোয়াড় যেকোনো একটা স্টপ থেকে এক বা একাধিক পাথর তুলে নিতে পারে। কেও চাল দিতে ব্যর্থ হলে হেবে যাবে। অর্থাৎ শেষ পাথরটা যে তুলে নিয়েছে সে গেমে জিতবে।



সাবস্ক্রাইব

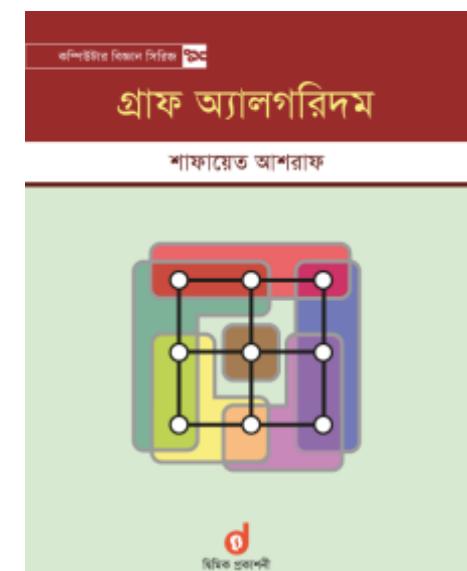


Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

**Traveloka Singapore** (বিস্তারিত...)



উপরের ছবিতে ৩টা পাথরের স্তপ দেখা যাচ্ছে, প্রথমটায় ৬টা, দ্বিতীয়টায় ৮টা এবং তৃতীয়টায় ৩টা পাথর আছে।

আগের গেমগুলার মতো এখনেও প্রত্যেক খেলোয়াড় অপটিমাল পদ্ধতিতে খেলবে, কেও কোনো ভুল চাল দিবে না। তোমাকে কোন স্তপে কয়টি পাথর আছে সেটা দেখে বলতে হবে প্রথম খেলোয়ার জিতবে নাকি হারবে।

এখন মনে করো  $n$  টা স্তপে যথাক্রমে  $a_1, a_2 \dots, a_n$  টা পাথর আছে। খেলায় প্রথম খেলোয়াড় হারবে শুধুমাত্র যদি  $xorsum = a_1 \oplus a_2 \oplus \dots \oplus a_n = 0$  হয়। এখনে  $\oplus$  দিয়ে xor অপারেটর বুঝানো হচ্ছে।

উপরের ছবির উদাহরণে  $xorsum = 6 \oplus 9 \oplus 3 = 12! = 0$ । এখনে xorsum শুন্য না, তাই অপটিমাল পদ্ধতিতে খেললে প্রথম খেলোয়াড়কে হারানো সম্ভব না।

এখন প্রশ্ন হলো এটা কেন আর কিভাবে কাজ করছে?  $xorsum = 0$  যদি লুজিং স্টেট হয়ে থাকে তাহলে বর্তমান খেলোয়াড় যেভাবেই চাল দিক না কেন সে গেমটাকে একটা উইনিং স্টেট এ নিয়ে যাবে।

মনে করো  $n = 4$  এবং স্তপ গুলোতে পাথরের সংখ্যার সেট  $\{9, 7, 11, 5\}$ । এদেরকে বাইনারিতে নিচের মত করে লিখি:

1 0 0 1 (= 9)

0 1 1 1 (= 7)

1 0 1 1 (= 11)

0 1 0 1 (= 5)

আমরা বাইনারি সংখ্যাগুলোকে একটা  $8 \times 8$  গ্রিড হিসাবে চিত্র করতে পারি। লক্ষ্য করো প্রতিটা কলামেই জোড় সংখ্যক 1 আছে, তাই  $xorsum = 0$  হবে, এটা একটা লুজিং স্টেট। এখন বর্তমান খেলোয়াড় একটা স্তপ বেছে নিয়ে কিছু পাথর উঠিয়ে নিলো। সে এই কাজটা যেভাবেই করুক, কোনো একটা সারির অস্তত ১টি 1 কে 0 তে পরিবর্তন করতে হবে। ফলে অস্তত একটা কলামে বিজোড় সংখ্যক 1 থেকে যাবে এবং  $xorsum > 0$  হয়ে যাবে।

তারমানে বর্তমান স্টেট এ  $xorsum = 0$  হলে তুমি যেভাবেই চাল দাও না কেনো  $xorsum > 0$  হয়ে যাবে।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগোরিদম

ডাটা স্টোরেজ(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

এখন বর্তমান স্টেট এ যদি  $xorsum > 0$  হয় তাহলে দেখানো যায় যে বর্তমান খেলোয়াড়ের পক্ষে এমন চাল দেয়া সম্ভব যাতে গেমটা লুজিং স্টেট এ চলে যায়, অর্থাৎ  $xorsum = 0$  হয়ে যায়।

মনে করো পাথরের সংখ্যার সেট  $\{5, 14, 9, 5\}$ ।

0 1 0 1 =(5)

1 1 1 0 =(14)

1 0 0 1 =(9)

0 1 0 1 = (5)

বাম থেকে ২য় এবং ৩য় কলামে বিজোড় সংখ্যক 1 আছে, তাই  $xorsum > 0$  হবে। এখন আমি এমন চাল দিতে চাই যেনো  $xorsum=0$  হয়ে যায়। এজন্য প্রথমেই সবথেকে বামের কলামটা খুজে বের করবো যেটাই বিজোড় সংখ্যা 1 আছে, এক্ষেত্রে সেটা ২য় কলাম। এবার ২য় কলামে 1 আছে এমন একটা রো বেছে নিবো, এক্ষেত্রে সেটা হতে পারে প্রথম, দ্বিতীয় বা চতুর্থ রো। এবার সেই রো এর ২য় কলামের 1 টাকে 0 বানিয়ে দাও এবং অন্যান্য কলামের 0 বা 1 কে এমন ভাবে পরিবর্তন করো যেন প্রতিটা কলামে জোড় সংখ্যক পাথর থাকে। তাহলেই  $xorsum = 0$  হয়ে গেলো!

তারমানে বর্তমান স্টেট এ  $xorsum > 0$  হলে বর্তমান খেলোয়াড় সহজেই এমন চাল দিতে পারবে যেনো  $xorsum = 0$  হয়ে যায়।

তাহলে দেখা যাচ্ছে লুজিং স্টেট ( $xorsum = 0$ ) থেকে যেভাবে চাল দেয়া হোক না কেনো শুধুমাত্র উইনিং স্টেট ( $xorsum > 0$ ) এ যাওয়া যায়, আবার বুদ্ধিমানের মত খেললে উইনিং স্টেট ( $xorsum > 0$ ) থেকে সবসময় লুজিং স্টেট এ যাওয়া যায় ( $xorsum = 0$ )। তাই শুরুতে  $xorsum > 0$  হলে প্রথম খেলোয়াড়কে কখনোই হারানো সম্ভব না সে যদি অপটিমাল পদ্ধতিতে খেলে।

এখন আমরা নিম্ন গেম এর কিছু ড্যারিয়েশন দেখি।

### মিজেরা(Misere) নিম

লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

**Misere** একটা ফ্রেঞ্চ শব্দ। মিজেরা নিমগেম এ যে খেলোয়ার শেষ পাথরটা তুলে নিবে সে হেবে যাবে। মিজেরা নিম এও  $xorsum > 0$  উইনিং পজিশন। তবে প্রথম খেলোয়াড়কে স্ট্রাটেজি কিছুটা পরিবর্তন করতে হবে। শেষ চালে সবগুলো পাথর তুলে না নিয়ে একটা মাত্র পাথর রেখে দিতে হবে, যিন্তীয় খেলোয়াড় তখন শেষ পাথরটা তুলতে বাধ্য হবে। তবে যদি প্রতিটা স্টপেই ঠিক ১টা করে পাথর থাকে তখন আর xorsum দিয়ে কাজ হবে না। তখন দেখতে হবে স্টপের সংখ্যা জোড় নাকি বেজোড়। যদি বেজোড় সংখ্যা স্টপ থাকে এবং প্রতিটাতে ১টা করে পাথর থাকে তাহলে বর্তমান খেলোয়াড়কে হাবানো সম্ভব না।

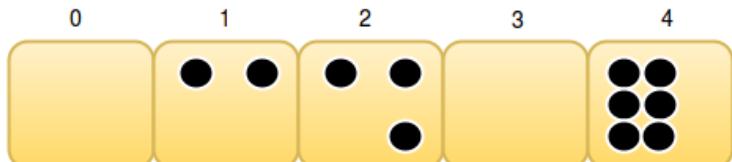
## প্রাইম পাওয়ার গেম

একটা সংখ্যা  $n$  দেয়া আছে। একজন খেলোয়াড় তার চালে  $n$  কে কোনো একটা প্রাইম সংখ্যার পাওয়ার দিয়ে ভাগ করতে পারে। সংখ্যাটা যদি ১ হয় যায় তাহলে বর্তমান খেলোয়াড় জিতে যাবে।

লক্ষ্য করো যেকোনো সংখ্যা  $n$  কে কিছু প্রাইম সংখ্যারণ্যফল হিসাবে লেখা যায়। যেমন  $n = 56700$  হলে আমরা লিখতে পারি  $n = (2 \times 2) \times (3 \times 3 \times 3 \times 3) \times (5 \times 5) \times (7) = 2^2 \times 3^4 \times 5^2 \times 7^1$ । এখন তুমি মনে করে ৪টা পাথরের স্টপ আছে, এবং পাথরের সংখ্যার সেট  $\{2, 4, 2, 1\}$ । এখন এটা নিম গেম এ পরিণত হয়েছে, তুমি যেকোনো একটা স্টপ থেকে এক বা একাধিক পাথর তুলে নিতে পারো!

## নিম্বল(Nimble)

নিম্বল গেম এ  $n$  টা ঘর থাকে যাদেরকে 0 থেকে  $n - 1$  দিয়ে চিহ্নিত করা হয়। প্রতিটা ঘরে এক বা একাধিক কয়েন থাকে। নিচের ছবি দেখো:



প্রতি চালে কোনো একটা ঘর থেকে একটামাত্র কয়েন সরিয়ে বামের কোনো একটা ঘরে রাখা যায়। যে চাল দিতে পারবে না সে হেবে যাবে। সবগুলো পাথর যখন 0-তম ঘরে চলে আসবে তখন আর চাল দেয়া সম্ভব হবে না।

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাইক কানেক্টেড কম্পোনেন্ট

ম্যানিমাম ফ্লো-১

ম্যানিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সারসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

লক্ষ্য করো ২ নম্বর ঘরে ৩টা পাথর আছে এবং প্রতিটা কয়েনকে ঠিক ২বার করে সরানো সম্ভব। আবার ১ নম্বর ঘরের প্রতিটি কয়েনকে ১বার এবং ৮ নম্বর ঘরের প্রতিটি কয়েনকে ৪ বার করে সরানো সম্ভব।

আমরা  $i$ -তম ঘরের প্রতিটা কয়েনকে  $i$  আকারের পাথরের স্তপ হিসাবে চিন্তা করতে পারি কারণ কয়েনটা ঠিক  $i$  বার সরানো সম্ভব। তাহলে উপরের উদাহরণে পাথরের স্তপের সেট হবে এরকম  $\{1, 1, 2, 2, 2, 4, 4, 4, 4, 4, 4\}$ । ৬টা স্তপের আকার ৪ কারণ ৪ নম্বর ঘরে ৬টা কয়েন আছে। এখন গেমটা সাধারণ নিমগেমে পরিণত হয়েছে।

## ম্যাট্রিক্স গেম

একটা  $n \times m$  আকারের গ্রীড দেয়া আছে। প্রতিটি ঘরে কিছু পাথর রাখা আছে। একজন খেলোয়াড় যেকোনো একটা সারির এক বা একাধিক কলাম থেকে কিছু পাথর তুলে নিতে পারে। প্রতি চালে অন্তত একটা পাথর তুলতেই হবে। শেষ পাথরটা যে তুলবে সে জিতে যাবে।

12	4	6	1
6	0	4	2
9	5	1	3

এই গেমটাকেও সহজে সাধারণ নিম গেম এ রূপান্তর করা যায়। প্রতিটি সারির মোট পাথরসংখ্যাকে একটা স্তপ হিসাবে চিন্তা করতে হবে। তারপর xorsum বের করলেই কাজ শেষ।

## পোকার নিম

পোকার নিম হলো একধরণের **বোগাস নিম**। বোগাস নিম এ পাথর শুধু তোলাই যায় না, পাথর যোগও করা যায়। পোকার নিম এর নিয়ম হলো যেকোনো স্তপে এক বা একাধিক পাথর যোগ করা যাবে। কোনো স্তপ এর আকার যদি শুরুতে হয়  $m$  তাহলে পাথর যোগ করার পরেও আকার  $m$  এর বেশি হতে পারবে না। আর গেমটা যাতে অসীম সময় ধরে না চলে সে জন্য কোনো খেলোয়াড়  $k$  বার এর বেশি পাথর যোগ করতে পারবে না।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাম্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

এই গেম এ যদি প্রথম খেলোয়াড় উইনিং স্টেট এ থাকে ( $xorsum > 0$ ) তাহলে ২য় খেলোয়াড় যখন কিছু পাথর যোগ করবে, প্রথম খেলোয়াড় পাথরগুলো সরিয়ে ফেলে গেমটাকে আবার এই স্টেট এ নিয়ে আসবে! তাই সাধারণ নিম এর মতই  $xorsum$  দেখে বলে দেয়া যায় গেম এ কে জিতবে।

**পরের পর্বে** আমরা আরো কিছু ড্যারিয়েশন দেখবো এবং গ্রান্ডি সংখ্যা নিয়ে জানবো। এখন লাইটওজে তে **এই সমস্যাগুলো** সমাধান করার চেষ্টা করো। নিম এর **আরো কিছু ড্যারিয়েশন** পাবে এখানে।

## ফেসবুকে মন্তব্য

0 comments

# অ্যালগরিদম গেম থিওরি ৩ (স্প্র্যাগ-গ্রান্ডি সংখ্যা)

এপ্রিল ১৬, ২০১৬ by শাফায়েত



আমরা এখন **নিম-গেম** কিভাবে সমাধান করতে হয় জানি, এখন আমরা গ্রান্ডি সংখ্যা দিয়ে কিছু কম্পোজিট গেম এর সমাধান করবো।

একটা গেম এর কথা চিন্তা করি যেখানে  $s$  টা পাথরের একটা স্তুপ(pile) আছে, প্রতিবার কোনো খেলোয়াড় ১টা বা ২টা পাথর তুলে নিতে পারে, শেষ পাথরটা যে তুলে নিবে সে জিতবে। পাথরের সংখ্যা দেখে তোমাকে বলতে হবে অপটিমাল পদ্ধতিতে খেললে প্রথম খেলোয়াড় জিততে পারবে কিনা। প্রথম পর্ব পড়ে থাকলে এটা সমাধান করা তোমার জন্য খুবই সহজ। আমাদের সুড়োকোড়টা হবে এরকম:

```

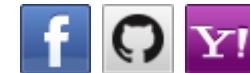
1 def can_win(s):
2     if s==0: return 0
3     if s==1: return 1
4     if(can_win(s-1)==0 or can_win(s-2)==0):
5         return 1
6     return 0

```

এখন ধরো পাথরের স্তুপের সংখ্যা একটার বদলে যদি  $n$  টা এবং প্রতিটা স্তুপে  $s_1, s_2 \dots \dots s_n$  টা পাথর আছে। কোনো খেলোয়াড় যেকোনো একটা স্তুপ থেকে ১ বা ২টা পাথর তুলে নিতে পারে, শেষ পাথরটা যে তুলবে সে জিতবে। এবার কিভাবে সমাধান করবে? এধরণের গেমকে কম্পোজিট গেম বলা হয়।

আমরা প্রতিটা স্তুপের জন্য একটা গ্রান্ডি সংখ্যা নির্ধারণ করবো। একটা স্তুপের গ্রান্ডি সংখ্যা  $g$  হলে আমরা ধরে নিতে পারি সেই স্তুপটা নিমগেম এ  $g$  টা পাথরের একটা স্তুপের সমতুল্য।  $n$  টা স্তুপের জন্য আমরা  $n$  টা গ্রান্ডি সংখ্যা  $g[s_1], g[s_2] \dots \dots g[s_n]$  পাবো, এবং এরপর নিমগেমের মতো  $g[s_1] \oplus g[s_2] \oplus \dots \dots g_n$  বের করলেই আমরা বুঝে যাবো গেমটায় কে জিতবে।

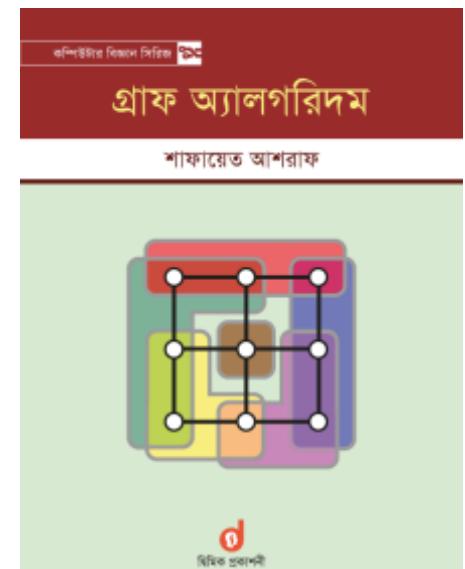
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



প্রতিটা সেট এর জন্য গ্রান্ডি সংখ্যা বের করার নিয়ম হলো, ওই স্টেপ থেকে অন্য যেসব সেট এ যাওয়া যায় সেগুলার গ্রান্ডি সংখ্যা প্রথম বের করতে হবে। মনে করো তাদের সেট হলো  $X = \{x_1, x_2, \dots\}$ । এখন সবনিম্ন যে সংখ্যাটা এই সেট এ নাই সেটাই হবে বর্তমান সেট এর গ্রান্ডি সংখ্যা। একটা উদাহরণ দেখলে ব্যাপারটা পরিষ্কার হবে।

মনে করো কোনো একটা স্ট্রপে ০টা পাথর আছে। আমাদের সেট হলো  $s = 0$ । আমরা জানি এটা একটা লুজিং পজিশন, আমরা ধরে নিতে পারি  $s = 0$  এর জন্য গ্রান্ডি সংখ্যা  $g[0] = 0$ ।

যদি স্ট্রপে ১টা পাথর থাকে তাহলে  $s = 1$ । এখান থেকে ১টা পাথর তুলে  $s = 0$  সেট এ যাওয়া যায়। সেট  $X$  এ তাহলে আছে  $X = \{g[0]\} = \{0\}$ । ১ হলো সবনিম্ন সংখ্যা যা  $X$  এ নেই, তাহলে আমরা ধরে নিবো  $g[1] = 1$ ।

যদি স্ট্রপে ২টা পাথর থাকে তাহলে  $s = 2$ । এখান থেকে ১টা বা ২টা পাথর তুলে  $s = 1$  বা  $s = 0$  সেট এ যাওয়া যায়। সেট  $X$  এ তাহলে আছে  $X = \{g[1], g[0]\} = \{0, 1\}$ । ২ হলো সবনিম্ন সংখ্যা যা  $X$  এ নেই, তাহলে আমরা ধরে নিবো  $g[2] = 2$ ।

যদি স্ট্রপে ৩টা পাথর থাকে তাহলে  $s = 3$ । এখান থেকে ১টা বা ২টা পাথর তুলে  $s = 2$  বা  $s = 1$  সেট এ যাওয়া যায়। সেট  $X$  এ তাহলে আছে  $X = \{g[2], g[1]\} = \{2, 1\}$ । ০ হলো সবনিম্ন সংখ্যা যা  $X$  এ নেই, তাহলে আমরা ধরে নিবো  $g[3] = 0$ ।

একই ভাবে হিসাব করলে তুমি পাবে  $g[4] = 1, g[5] = 2$  ইত্যাদি।

এখন যদি ৩টা স্ট্রপে  $\{5, 2, 3\}$  টা পাথর থাকে তাহলে তুমি বলতে পারো গেমটা  $\{g[5], g[2], g[3]\} = \{2, 2, 0\}$  টা পাথর নিয়ে সাধারণ নিম গেম খেলার সমতুল্য। এখন তুমি *xorsum* দেখে বলতে পারো খেলাটায় কে জিতবে।

গ্রান্ডি সংখ্যা বের করার জন্য সাধারণ সুড়োকোড হতে পারে এরকম:

```

1 def grundyNumber (current_state):
2     moves[] = possible positions to which I can move from current_state
3     set X;
4     for (all new_state in moves)
5         X.insert(grundyNumber(new_state));
6
7     int ret=0;
```



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

```

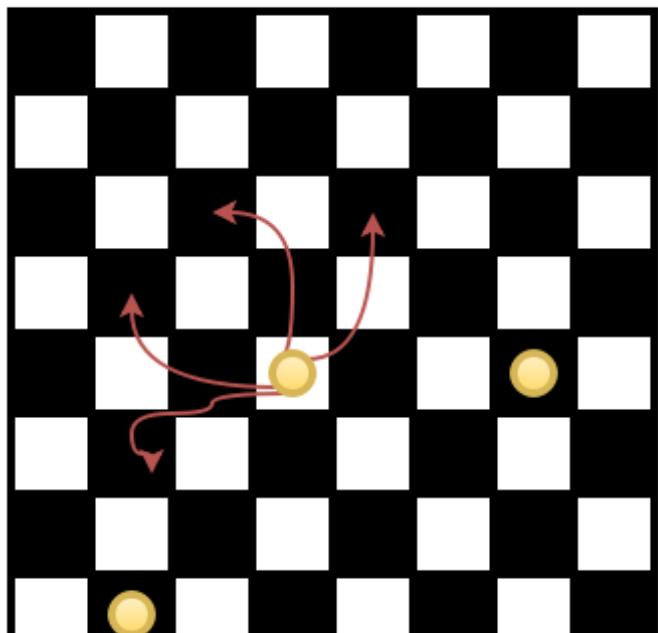
8     while (X.contains(ret)) ret++;
9     #ret is the smallest non-negative integer not in the set s;
10    return ret;

```

এ পর্যন্ত যদি কুরে থাকো তাহলে নিচের সমস্যাটা সমাধান করো:

একটা দাবার বোর্ডে  $k$  টা কয়েন রাখা আছে। কোনো কয়েন  $(x,y)$  ঘরে থাকলে প্রতি চালে কয়েনটিকে  $(x-2, y+1)$  অথবা  $(x-2, y-1)$  অথবা  $(x-1, y-2)$  অথবা  $(x+1, y-2)$  ঘরে সরানো যায়, তবে কয়েনটি বোর্ডের বাইরে সরানো যাবে না।

(1,1)



(8,8)

প্রতি চালে বর্তমান খেলোয়াড় উপরের নিয়ম মেনে একটা কয়েন সরাতে পারবে। যে খেলোয়াড় কয়েন সরাতে পারবে না সে হেবে যাবে। তোমাকে কয়েনগুলোর স্থানাংক দেয়া থাকবে, বলতে হবে খেলায় কে জিতবে।

এই দাবার বোর্ডে প্রতিটা ঘরের জন্য গ্রান্তি সংখ্যা হিসাব করলে তুমি নিচের মতো একটা টেবিল পাবে:

শ্বাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইন্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

0	0	1	1	0	0	1	1
0	0	2	1	0	0	1	1
1	2	2	2	3	2	2	2
1	1	2	1	4	3	2	3
0	0	3	4	0	0	1	1
0	0	2	3	0	0	2	1
1	1	2	2	1	2	2	2
1	1	2	3	1	1	2	0

লক্ষ্য করে দেখো যেসব ঘর থেকে অন্য কোনো ঘরে যাওয়া যায় না তাদের গ্রান্ডি সংখ্যা ০, তারমানে সেসব ঘরগুলো হলো লুজিং পজিশন। বাকি ঘরগুলোর জন্য গ্রান্ডি সংখ্যা হিসাব করতে হলে প্রথমে একটা ঘর থেকে অন্য যেসব ঘরে যাওয়া যায় তাদের গ্রান্ডি সংখ্যা হিসাব করতে হবে। মনে করো সেই সংখ্যাগুলোর সেট হলো  $S$ । সবনিম্ন যে নন-নেগেটিভ সংখ্যা  $S$  এ নেই সেটাই হবে বর্তমান ঘরের গ্রান্ডি সংখ্যা।

আজকে এই পষ্টেই। গ্রান্ডি সংখ্যা কিভাবে নির্ণয় করতে হয় বুঝে থাকলে [এই সমস্যাগুলো সমাধান](#) করো। এই পদ্ধতিটা কেন কাজ করে সেটাও চিন্তা করে বের করার চেষ্টা করো।

হ্যাপি কোডিং!

## ফেসবুকে মন্তব্য

2 comments

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)

# ডাইনামিক প্রোগ্রামিং – ১ (ফিবোনাচি)

এপ্রিল ১৮, ২০২০ by শাফায়েত



ডাইনামিক প্রোগ্রামিং নামটা শুনতে একটু কঠিন মনে হলেও এর পিছনে কনসেপ্টটা বেশ সহজ। ডাইনামিক প্রোগ্রামিং কে এক কথায় বর্ণনা করতে গেলে বলা যায় – একটা সমস্যাকে ছোট ছোট ভাগ করে সাব-প্রবলেমগুলো সমাধান করবো তবে একই সাবপ্রবলেম একবারের বেশি সমাধান করবো না। মোটা দাগে বলতে গেলে এটাই ডাইনামিক প্রোগ্রামিং। তবে কখন একটা প্রবলেমকে ডাইনামিক প্রোগ্রামিং দিয়ে সমাধান করা যাবে সেটা বুঝতে একটু অভিজ্ঞতা প্রয়োজন। তো এই সিরিজে আমরা দেখবো ডাইনামিক প্রোগ্রামিং কি এবং কিছু কমন সমস্যা যেগুলো ডাইনামিক প্রোগ্রামিং দিয়ে সমাধান করা যায়।

ডাইনামিক প্রোগ্রামিং বা ডিপি টার্মটা একটু কনফিউজিং কারণ 'ডাইনামিক' শব্দটা এখানে খুব একটা অর্থপূর্ণ উপায়ে ব্যবহার করা হয় নি। কিছু বুরোক্রেটিক কারণে ডষ্টের রিচার্ড বেলম্যান (বেলম্যান অ্যালগরিদমের জনক) এই নামটি ব্যবহার করেন, সে গল্পটি তুমি উইকিপিডিয়ায় সার্চ করলে পাবে। একথাটা শুরুতে বলে নেয়ার কারণ হলো আমি জানি ডায়নামিক প্রোগ্রামিং শেখার পর তুমি চিন্তা করবে এই টেকনিকের কোন অংশটা 'ডাইনামিক'! এটা নিয়ে চিন্তা করার দরকার নেই, রিসার্চ গব্যাট বাচাতে বেলম্যান এই অঙ্গুত নামটা ব্যবহার করেছেন।

ডাইনামিক প্রোগ্রামিং শেখার প্রথম পূর্বশর্ত হলো রিকার্শন বোঝা। রিকার্শন নিয়ে ভালো ধারণা না থাকলে [এই লেখাটা পড়তে পারো](#)। এছাড়াও আশা করবো তোমার টাইম এবং স্পেস কমপ্লেক্সিটি নিয়ে ধারণা আছে।

এই সিরিজের মূল রেফারেন্স হিসাবে ব্যবহার করেছি MIT'র লেকচার সিরিজ, তুমি চাইলে সরাসরি [সেখান থেকেই](#) ডিপি শিখে ফেলতে পারো।

## ডাইনামিক প্রোগ্রামিং

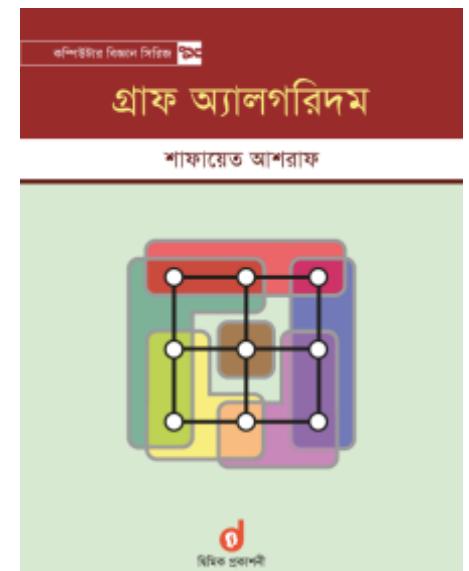
### সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

### আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



ডাইনামিক প্রোগ্রামিং একটা নির্দিষ্ট কোন সমস্যা সমাধানের অ্যালগরিদম না, বরং এটা একটা প্রবলেম সলভিং টেকনিক যেটা দিয়ে অনেক ধরণের অপটিমাইজেশন প্রবলেম বা কাউন্টিং প্রবলেম সলভ করা যায়, যেমন ফিবোনাচি, কয়েন চেঙ্গ, ম্যাট্রিক্স চেইন মাণ্টিপ্লিকেশন। ডাইনামিক প্রোগ্রামিং কে এক ধরণের ক্রুট ফোর্স অ্যালগরিদম বললে ভুল হবে না। এটা একটা প্রবলেমকে ছোট ছোট ভাগে ভাগ করে সব রকম সম্ভাব্য সাবপ্রবলেম থেকে সঠিক সমাধান খুজে নিয়ে আসে। তবে এটা একটু বৃদ্ধিমান ক্রুট ফোর্স যা পলিনমিয়াল কমপ্লেক্সিটিতেই কাজ করে। অনেক প্রবলেম আছে যেগুলা সলভ করার একমাত্র পলিনমিয়াল অ্যালগরিদম ডাইনামিক প্রোগ্রামিং।

## ফিবোনাচি সিকুয়েন্স

সবথেকে সহজ উদাহরণ দিয়ে আমরা শুরু করবো। ইটালিয়ান গণিতবিদ Leonardo Pisano Bigollo যাকে আমরা ফিবোনাচি নামে চিনি খরগোশের বংশবৃক্ষ পর্যবেক্ষণ করতে গিয়ে একটা নাম্বার সিরিজ আবিষ্কার করে বসলেন। সিরিজটি এরকম:

“ 0, 1, 1, 2, 3, 5, 8, 13, 21, 34 ....

লক্ষ্য করো ১ম দুটি সংখ্যা ছাড়া প্রতিটি সংখ্যা হলো আগের দুটি সংখ্যার যোগফল। আমরা একটি ফাংশন কল্পনা করি \$ f(n) \$ যা \$ f(n) \$ তম ফিবোনাচি সংখ্যা রিটার্ন করে। ফিবোনাচি সংখ্যার ফর্মুলা বিকার্সিভ ফাংশনের মাধ্যমে প্রকাশ করলে আমরা পাবো:

$$\begin{aligned}f(0) &= 0 \\f(1) &= 1 \\f(n) &= f(n - 1) + f(n - 2) \text{ if } n > 1\end{aligned}$$

সি++ এ কোড লিখলে সেটা হবে এরকম:

```
1 int f(int n) {
2     if (n == 0) return 0;
3     if (n == 1) return 1;
4
5     return f(n - 1) + f(n - 2);
6 }
```



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগোরিদম

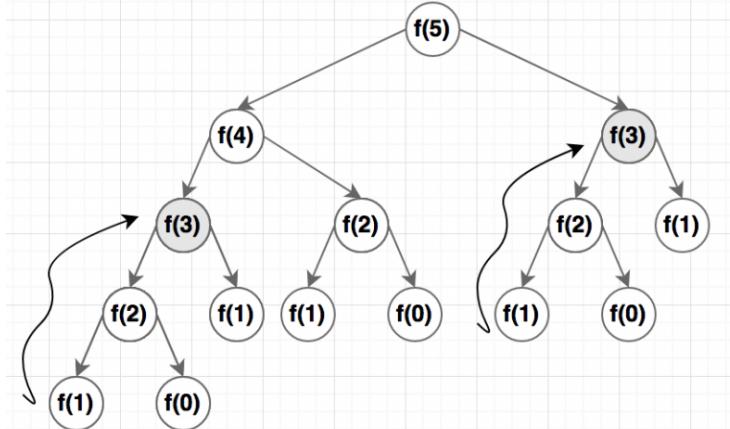
ডাচা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

আমরা  $f(n)$  প্রবলেমটাকে ছোট দুটো সাবপ্রবলেম ভাগ করে ফেলতে পেরেছি। এই কোডের কমপ্লেক্সিটি কত সেটাকি তুমি জানো? আমরা যদি ফাংশনটাকে  $n = 5$  দিয়ে কল করি তাহলে ফিবোনাচি খুজে পেতে কোন কোন ফাংশন কল হবে সেটা নিচের ছবির ট্রি-তে দেখানো হয়েছে:



0 থেকে  $n$  পর্যন্ত সবগুলো মানের জন্য দুটো করে ফাংশন কল হচ্ছে (বেস কেস ছাড়া)। টাইম কমপ্লেক্সিটি গিয়ে দাঢ়াচ্ছে এ। এটা একাপোনেশিয়াল কমপ্লেক্সিটি যে খুব ধীরগতিতে কাজ করবে। আমাদের আরেকটু ভালো কিছু দরকার।

হয়তো তুমি এরই মধ্যে লক্ষ্য করেছো আমি  $f(3)$  কে দুই জায়গায় আলাদা করে চিহ্নিত করে দিয়েছি।  $f(5)$  ক্যালকুলেট করতে গিয়ে  $f(3)$  কল করা হচ্ছে দুইবার এবং দুইবারই আমরা  $f(3)$  এর নিচের সবগুলো সবগুলো সাবপ্রবলেম নতুন করে সলভ করছি।

আমি শুক্রতেই ডাইনামিক প্রোগ্রামিং নিয়ে বলেছি "একটা সমস্যাকে ছোট ছোট ভাগ করে সাব-প্রবলেমগুলো সমাধান করবো তবে একই সাবপ্রবলেম একবারের বেশি সমাধান করবো না"। এখানে আমরা সাবপ্রবলেমে ঠিকই ভাগ করেছি কিন্তু একই সাবপ্রবলেম বারবার সলভ করছি। তো সেটা না করে সাবপ্রবলেমের রেজাল্টগুলো একটা টেবিলে সেভ করে রাখলেই কিন্তু কাজ হয়ে যায়। যদি দেখি যে কোনো সাবপ্রবলেমের রেজাল্ট আমরা জানি তাহলে সেটা আবার সমাধান করার দরকার নেই।

```

1 #define MAX_N 20
2 #define EMPTY_VALUE -1
3
4 int memo[MAX_N];
5
6 int f(int n) {

```

শ্লাইডিং রেজ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্লেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

7 if (n == 0) return 0;
8 if (n == 1) return 1;
9
10 if (memo[n] != -1) {
11     return memo[n];
12 }
13
14 memo[n] = f(n - 1) + f(n - 2);
15 return memo[n];
16 }
17
18 void init() {
19     for (int i = 0; i < MAX_N; i++) {
20         memo[i] = EMPTY_VALUE;
21     }
22 }

```

আমরা `memo` নামের একটা অ্যারে ব্যবহার করেছি সাবপ্রবলেমের রেজাল্টগুলো সেভ করে রাখতে। একদম শুরুতে অ্যারেতে এমন ভ্যালু রাখতে হবে যেটা কখনোই উত্তর হওয়া সম্ভব না, এক্ষেত্রে আমরা  $-1$  ব্যবহার করছি। প্রতিবার রিকার্সিভ ফাংশন কল করার আগে দেখছি যে সাবপ্রবলেমটার সমাধান অ্যারেতে এরই মধ্যে রাখা আছে নাকি, থাকলে সেটা রিটার্ন করে দিচ্ছি। আর না থাকলে সাবপ্রবলেমের রেজাল্ট ক্যালকুলেট করে আগে অ্যারেতে সেভ করছি এবং তারপর রিটার্ন করছি।

এবার আমাদের কোডের টাইম কমপ্লেক্সিটি হয়ে যাচ্ছে  $O(n)$  কারণ প্রতিটা সাবপ্রবলেম আছে  $n$  টা এবং প্রতিটা সাবপ্রবলেম আমরা মাত্র ১বার সমাধান করছি। রিকার্সিভ ফাংশন কল করা ছাড়া আর কোনো কাজ আমরা এই ফাংশনে করিনি, করলে সেটার টাইম কমপ্লেক্সিটিও গুণ হতো, সেটা আমরা পরের প্রবলেমেই দেখবো।

এভাবে সাবপ্রবলেমের রেজাল্ট সেভ করে রাখার একটা নাম আছে, সেটা হলো **Memoization**। ল্যাটিন ভাষার শব্দ **memorandum** থেকে এই শব্দটাকে করেন Donald Mitchie নামক একজন এ.আই.রিসার্চার, ইংরেজি **memorization** শব্দটির সাথে এর তেমন একটা পার্থক্য নেই।

ডাইনামিক প্রোগ্রামিং এর কোড রিকার্সিভ হতে হবে এমন কোন কথা নেই। ইটারেটিভ কোড লিখতে সাবপ্রবলেম গুলোকে মনে মনে টপোলজিকাল অর্ডারে সাজিয়ে নিতে হবে এবং যে সাবপ্রবলেমের উত্তর আমাদের জানা আছে (**base case**) সেখান থেকে রেজাল্ট বিন্দুত্ত্ব করতে হবে। টপোলজিকাল অর্ডার মানে হলো কতগুলো ঘটনাকে কোনটার উপর কোনটা নির্ভরশীল সেই অর্ডারে সাজানো।

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সাবসিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

ফিবোনাচির ক্ষেত্রে আমরা  $f(0)$ ,  $f(1)$  এর রেজাল্ট আগে থেকেই জানি। আমরা এই দুটো ভ্যালুকে টেবিলে সেভ করে  $f(2)$ ,  $f(3)$  . . . .  $f(n)$  এই অর্ডারে রেজাল্ট বিন্দআপ করতে পারি।

```
1 #define MAX_N 20
2
3 int memo[MAX_N];
4
5 int f(int n) {
6     memo[0] = 0;
7     memo[1] = 1;
8     for(int i = 0;i < n;i++) {
9         memo[n] = memo[n - 1] + memo[n - 2];
10    }
11
12    return memo[n];
13 }
```

ডাইনামিক প্রোগ্রামিং সমাধান করার সময় তোমাকে শুরুতে মনে মনে বা খাতায় রিকার্সিভ সমীকরণ বের করে নিতে হবে। এরপর তুমি চাইলে কোড রিকার্সিভ বা ইটারেটিভ ভাবে লিখতে পারে। রিকার্সিভ ভাবে কোড লেখা খুবই সহজ, সমীকরণটাকেই কোড হিসাবে লিখে দিলে ম্যাজিকের মত সমাধান বের করে আনবে। ইটারেটিভ ভাবে লিখতে হলে তোমাকে সাবপ্রবলেমের অর্ডারিং নিয়ে চিন্তা করতে হবে যেটা একটু কঠিন হয়ে যায় যদি সাবপ্রবলেমের একাধিক প্যারামিটার থাকে। তবে ইটারেটিভ পদ্ধতিও আমাদের জন্য থাকতে হবে কারণ ইটারেটিভ পদ্ধতিতে অনেক সময়ই স্পেস অপটিমাইজ করা যায়। যেমন উপরের কোডেই আমাদের অ্যারের ঠিক আগের দুটি ভ্যালু ছাড়া বাকিগুলো কোনো কাজে লাগছে না, তাই  $n$  সাইজের অ্যারের কোনো দরকার নাই।

ডাইনামিক প্রোগ্রামিং প্রবলেম অ্যাটাক করার কিছু নিয়ম আছে, আমরা সেই নিয়ম মনে চিন্তা করতে শিখলেই কোন প্রবলেম ডাইনামিক প্রোগ্রামিং দিয়ে সমাধান করা যাবে এবং কিভাবে সমাধানটা বের করা যাবে সেটা বুঝে যাবো। ফিবোনাচি খুব সহজ উদাহরণ হলেও খুব ভালো উদাহরণ না কারণ এখানে কোন কিছু অপটিমাইজ করা হয় না। **পরের পরে** আমরা একটা অপটিমাইজেশন প্রবলেম দেখবো যেটা আমাদের ডিপির ধারণা আরো পরিস্কার করে দিবে।

## ফেসবুকে মন্তব্য

2 comments

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ভ্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

# ডাইনামিক প্রোগ্রামিং-২ (শর্টেস্ট পাথ)

ঃ এপ্রিল ১৮, ২০২০ by শাফায়েত



আগের পর্বে আমরা ফিবোনাচি নাম্বার নিয়ে আলোচনা করেছি। আমরা দেখেছি কিভাবে ডাইনামিক প্রোগ্রামিং ব্যবহার করে রিকার্সিভলি এবং ইটারেটিভলি ফিবোনাচি সংখ্যা জেনারেট করা যায়। এই পর্বে আমরা কথা বলবো শর্টেস্ট পাথ প্রবলেম নিয়ে এবং সেটা নিয়ে আলোচনা করার সময় ডিপির কিছু নতুন প্রোপার্টি নিয়ে জানবো।

যদিও শর্টেস্ট পাথ গ্রাফ থিওরির একটা প্রবলেম, এই লেখা বুঝতে গ্রাফ নিয়ে না জানলেও চলবে।

মনে করো আমাদেরকে এক শহর থেকে অন্য শহরে যাবার শর্টেস্ট পাথ খুজে বের করতে হবে। শহর আছে মোট  $n$  টি। 0 হলো প্রথম নোড, এবং  $n-1$  হলো শেষ নোড। কোন শহর থেকে কোন শহরে সরাসরি যাওয়া যায় এবং শহরগুলোর মধ্যে দুরুত্ব কত সেটা অ্যারো দিয়ে দেখিয়ে দেয়া আছে নিচের ছবির মতো:

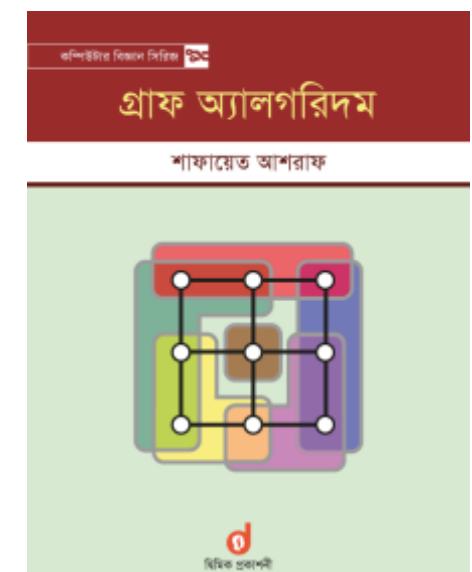
সাবক্ষাইব

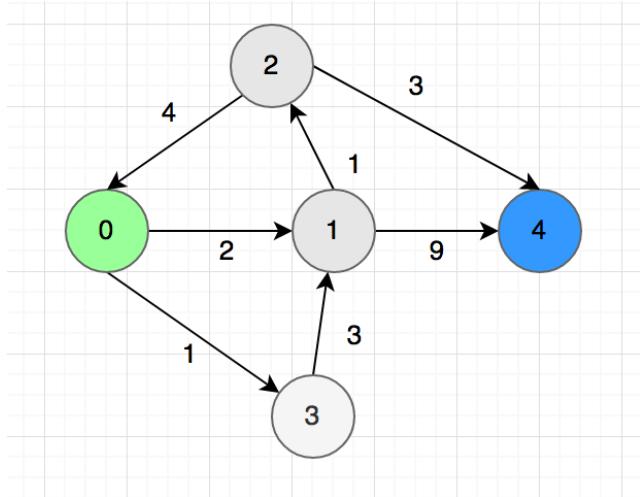


Secured by OneAll Social Login

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)





এখন আমাদের প্রবলেমটা হলো  $0$  থেকে  $n - 1$  এ যাবার শ্টেপ্ট পাথের দৈর্ঘ্য কত? এখন আমরা কয়েকটা ধাপে সমস্যাটা ফর্মুলেট করবো।

#### প্রবলেমের প্যারামিটার বা স্টেট নির্ধারণ:

প্রথমেই আমাদের বের করতে হবে প্রবলেমটাকে কি কি প্যারামিটার বা স্টেট দিয়ে প্রকাশ করা যায়। স্টেট শব্দটা মাথায় রেখো, সামনে অনেক বার এটা ব্যবহার হবে। আমাদের স্টেট হবে এক্ষেত্রে তুমি বর্তমানে কোন শহরে আছো সেটা। ধরে নিলাম তুমি বর্তমানে  $u$  তম শহরে আছো এবং আমাদের প্রবলেমটাকে আমরা  $f(u)$  ফাংশন দিয়ে প্রকাশ করবো। ফিবোনাচির মতো আমাদেরকে এখন একটা রিকার্সিভ রিলেশন বের করতে হবে।

#### স্টেট ট্রানজিশন এবং রিকার্শন:

আমরা জানিনা শহর  $u$  থেকে কোন শহরে গেল দ্রুততম উপায়ে গত্তব্য পৌছাতে পারবো। এই ধাপে এসে আমরা সেটা অনুমান করবো। প্রতিটা অনুমান হবে একটা করে সাব-প্রবলেম। ছবিতে শহর  $0$  থেকে  $1$  এবং  $3$  এ যাওয়া যায়। তাহলে  $f(0)$  থেকে আমরা নিচের শ্টেপ্ট পাথ পাবো এই ফর্মুলা দিয়ে:

$$f(0) = \min(f(1) + 2, f(3) + 1)$$



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে  
হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগোরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

আইডিয়াটা হলো প্রতিটা শহর থেকে আমরা অন্য সব শহরে যাবো এবং সেখান থেকে শর্টেস্ট পাথ খুজে বের করার চেষ্টা করবো। সবগুলো অনুমানের মধ্যে যেটা সবথেকে ছোট সেটাই হবে উত্তর। জেনারেলাইজড করে লিখলে:

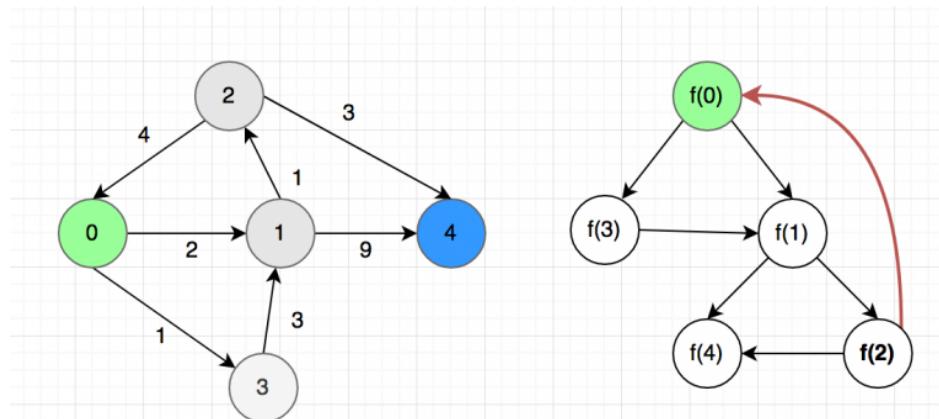
$$f(n-1) = 0$$

$$f(u) = \min(f(u, v) + w(u, v)) \text{ where } (u, v) \in E \text{ for } u \neq n-1$$

এখানে  $w(u, v)$  হলে  $u$  থেকে  $v$  তে যাবার দূরত্ব।

### সাবপ্রবলেম/স্টেট অর্ডারিং:

এখন একটু চিন্তা করো এই রিকার্শন চালিয়ে দিলে কিভাবে ফাঁশন কল হবে।



এবার আমরা একটু সমস্যা পড়ে গিয়েছি, সাবপ্রবলেমগুলোর মধ্যে সাইকেল তৈরি হয়ে গিয়েছে।  $f(0)$  এর মান জানতে  $f(2)$  জানতে হবে কিন্তু  $f(2)$  আবার  $f(0)$  এর উপর নির্ভরশীল। এই সলিউশন ইমপ্রিমেন্ট করলে আমাদের কোড ইনফাইনাইট লুপে আটকে যাবে।

এত কষ্ট করে একটা ভুল সমাধান বের করার উদ্দেশ্য আসলে তোমাদের **DAG** বা **ডিরেক্টেড অ্যাসাইন্ক গ্রাফ** কনসেপ্টটার সাথে পরিচয় করিয়ে দেয়। DAG মানে হলো একটা ডিরেক্টেড গ্রাফ যেখানে কোনো সাইকেল নেই। ডাইনামিক প্রোগ্রামিং সলিউশন কাজ করবে শুধুমাত্র তখনই যখন সাবপ্রবলেমগুলো একটা DAG তৈরি করবে। সাইকেল থাকলেই সাবপ্রবলেমগুলা ইনফিনিট লুপে আটকে যাবে।

স্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইন্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

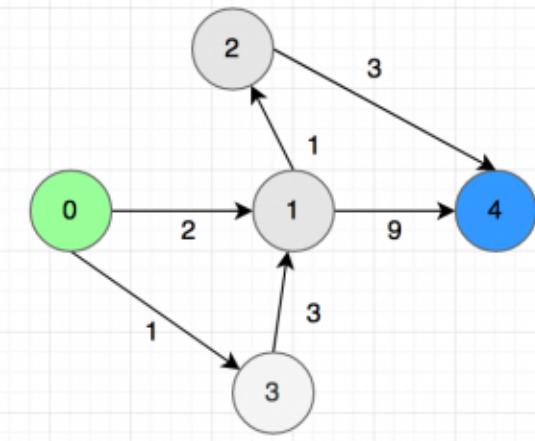
সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

DAG এ শর্টেস্ট পাথ প্রবলেম আমাদের বের করা ফর্মুলা দিয়েই সমাধান করা যাবে। উপরের গ্রাফে  $2 \rightarrow 0$  অ্যারোটা মুছে দিলেই গ্রাফটা অ্যাসাইন্সেশন হয়ে যাবে।



অ্যাসাইন্সেশন গ্রাফের জন্য কোডটা হবে এরকম:

```
1 #define MAX_N 20
2 #define INF 99999999
3 #define EMPTY_VALUE -1
4
5 int w[MAX_N][MAX_N];
6 int mem[MAX_N];
7
8 int f(int u, int n) {
9     if (u == n - 1) {
10         return 0;
11     }
12
13     if (mem[u] != EMPTY_VALUE) {
14         return mem[u];
15     }
16
17     int ans = INF;
18     for (int v = 0; v < n; v++) {
```

অ্যাসাইন্সেশন পয়েন্ট এবং বিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শর্টেস্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিনেটরিঙ্গ, ডিসিশন প্রবলেম)

```

19     if (w[u][v] != INF) {
20         ans = min(ans, f(v, n) + w[u][v]);
21     }
22 }
23
24 mem[u] = ans;
25 return mem[u];
26 }

```

ইটারেটিং ব্যপার হলো সাইক্লিক গ্রাফেও ডিপি দিয়ে শর্টেস্ট পাথ বের করা যায় তবে সেক্ষেত্রে একটা অতিরিক্ত প্যারামিটার  $k$  যোগ করতে হয় যেটা দিয়ে বুঝায় 'সর্বোচ্চ  $k$  টা এজ ব্যবহার করে শর্টেস্ট পাথ কত?'। তখন সেটাই হয়ে যাবে **বেলম্যানফোর্ড অ্যালগরিদম!**

### কমপ্লেক্সিটি:

কোডে আমরা দুটো প্যারামিটার  $u$  আর  $n$  পাস করলেও প্রবলেমের স্টেট আসলে শুধু প্রথম প্যারামিটারটাই।  $u$  এর ভ্যালু হতে পারে ০ থেকে  $n - 1$  পর্যন্ত। প্রতিটা সাবপ্রবলেমের জন্য আবার সেই শহরের সাথে কোন শহরের কানেকশন আছে সেটা আমাদের চেক করতে হচ্ছে একটা লুপ চালিয়ে, কানেকশন থাকতে পারে সর্বোচ্চ  $n$  টা। মোট কমপ্লেক্সিটি পেতে আমরা স্টেট সংখ্যা এবং ডিতরের কমপ্লেক্সিটি গুণ করে দিয়ে যাবো  $O(n * n)$ ।

### ইটারেটিভ ভার্সন:

এই প্রবলেমের ইটারেটিভ ভার্সন লেখা একটু কঠিন তবে সম্ভব। আমরা জানি ইটারেটিভ ডিপির জন্য সাবপ্রবলেমগুলোকে **টপোলজিকাল অর্ডারে** সাজিয়ে নিতে হবে। কারণ প্রতিটা স্টেটে তোমাকে নিশ্চিত করতে হবে যে তুমি ডিপেন্ডেন্ট স্টেটগুলো আগেই ক্যালকুলেট করে এসেছো। ফিবোনাচির জন্য কাজটা খুব সহজ ছিল,  $0, 1, 2, 3 \dots$  এভাবে স্টেটগুলো আগাচ্ছিলো, কিন্তু গ্রাফের জন্য ব্যাপারটা একটু কঠিন। গ্রাফে নোডগুলোকে কিভাবে টপোলজিকাল সার্টিং করতে হয় সেটা তুমি **এখান থেকে** শিখে নিতে পারো। উপরের উদাহরণে সর্ট করার পর আমরা পারে  $0, 3, 1, 2, 4$ । এখান এই অ্যারের উন্টা দিক থেকে লুপ চালালে তুমি ডিপির টেবিল বিন্দুআপ করতে হবে।

এই প্রবলেমের ইটারেটিভ সলিউশন তুমি আপাতত না লিখতে পারলেও সমস্যা নেই, ইটারেটিভ সলিউশনটা জটিল এবং তেমন কোনো অ্যাডভান্টেজ দিবে না। ইটারেটিভ ডিপি আমরা আরো ভালো লেখা শিখবো যখন LIS, ন্যাপস্যাক সলভ করবো।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ভ্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

# ডাইনামিক প্রোগ্রামিং-৩ (লংগেস্ট ইনক্রিজিং সাবসিকোয়েন্স, পাথ প্রিন্টিং)

এপ্রিল ১৮, ২০২০ by শাফায়েত



আমরা এই মধ্যে ডাইনামিক প্রোগ্রামিং এর বেসিক শিখে গিয়েছি, আমরা ফিবোনাচি এবং DAG এ শর্টেস্ট পাথ বের করতে পারি। এবার আমরা শিখবো Longest Increasing Subsequence বা LIS বের করা। এতদিন আমরা শুধু অপটিমাল সলিউশনটা বের করতে শিখেছি, কোন পথ ধরে সলিউশনে পৌছাতে হ্য সেটা বের করা শিখিনি। এবার আমরা নেক্সট-পয়েন্টার ব্যবহার করে সেটা বের করাও শিখবো।

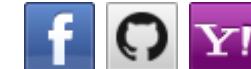
ধৰা যাক আমাদের নিচের ছবির মতো একটা অ্যারে আছে যার নাম  $A$ ।

0	1	2	3	4	5	6
5	0	9	2	7	3	4

একটা অ্যারের সাবসিকোয়েন্স বলতে বুঝায় অ্যারে থেকে কিছু এলিমেন্ট মুছে দিলে বাকি যে সিকোয়েন্সটা থাকে সেটা। এলিমেন্টগুলোর অর্ডারিং পরিবর্তন করা যাবে না।  $n$  সাইজের একটা অ্যারের  $2^n$  টি সাবসিকোয়েন্স থাকতে পারে (প্রতিটা এলিমেন্টের জন্য ২টি চয়েস, রেখে দেয়া বা মুছে দেয়া)। ইনক্রিসিং সাবসিকোয়েন্স হলো এমন একটা সাবসিকোয়েন্স যার প্রতিটি পজিশনের ড্যালু আগের পজিশনের ড্যালুর থেকে বড়।

উপরের উদাহরণে  $\{0, 9\}, \{5, 9\}, \{0, 2, 7\}, \{0, 2, 3, 4\}$  কিছু ইনক্রিসিং সাবসিকোয়েন্স। সবথেকে লম্বা ইনক্রিসিং সাবসিকোয়েন্স বা LIS হলো  $\{0, 2, 3, 4\}$ । প্রথমে আমরা চেষ্টা করবো LIS এর দৈর্ঘ্য বের করতে, এরপরে মূল সাবসিকোয়েন্সটাও টুর্যাক করা শিখবো।

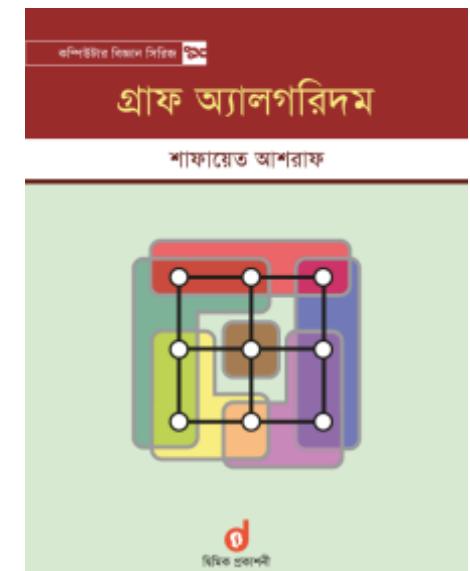
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



## প্রবলেমের প্যারামিটার বা স্টেট নির্ধারণ:

আমরা প্রবলেমটা এভাবে চিত্ত করতে পারি: আমরা বর্তমানে ইনডেক্স  $i$  তে আছি এবং আমাদেরকে সবথেকে লম্বা সার্বিসকোয়েম বের করতে হবে যেটা  $i$  তম ইনডেক্স থেকে শুরু হয়েছে। ধরা যাক  $f(i)$  ফাংশনটা সেই জিনিসটা তোমাকে ক্যালকুলেট করে দিতে পারে। এখন আগের মতো  $f(0)$  বের করলেই কি হবে? হবে না কারণ আমরা জানিনা LIS ঠিক কোন ইনডেক্স থেকে শুরু হয়েছে। আমাদেরকে ফাইনাল রেজাল্ট হবে হবে –  $\max(f(0), f(1) \dots f(n-1))$

## স্টেটট্রানজিশন এবং রিকার্শন:

এখন আমরা জানিনা ইনডেক্স  $i$  থেকে কোন ইনডেক্সে গেলে আমরা সব থেকে লম্বা সার্বিসকোয়েম পাবো। এখন আমাদেরকে অনুমান করতে হবে।  $i$  থেকে যে যে ইনডেক্সে যাওয়া যায় সবগুলোয় গিয়ে গিয়ে আমরা দেখবো সেখান থেকে LIS এর দৈর্ঘ্য কত এবং যেটা সবথেকে লম্বা সেটাকে বেছে নিবো।

ইনডেক্স  $i$  থেকে কোন কোন ইনডেক্সে যাওয়া যায়?  $j$  ইনডেক্সকে দুটো শর্ত পূরণ করতে হবে:

যেহেতু অর্ডার মেইনটেইন করতে হবে, তাই নতুন ইনডেক্স  $j$  অবশ্যই  $i$  এর থেকে বড় হবে ( $i < j < n$ )।

সেই সাথে  $A[j]$  এর মানও  $A[i]$  থেকে বড় হতে হবে ( $A[j] > A[i]$ )।

অর্থাৎ  $(i < j < n) \& (A[j] > A[i])$  হলেই শুধুমাত্র আমরা  $i$  থেকে  $j$  তে যেতে পারবো। রিকার্শনটা তাহলে হবে এরকম

$$f(i) = 1 + \max(f(j)) \text{ where } i < j < n \& A[j] > A[i]$$

কনফিউজড লাগছে? আগের শর্টেস্ট পাথ প্রবলেমের মত মনে করো প্রতিটা ইনডেক্স একটা করে শহর। এবার কোন শহর থেকে কোন শহরে যাওয়া যায় সেটা অ্যাবো টেনে দেখিয়ে দাও উপরের শর্ত মনে।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

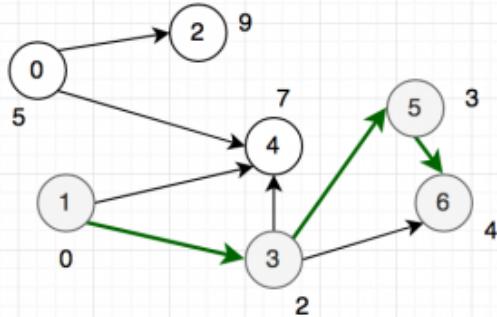
ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

0	1	2	3	4	5	6
5	0	9	2	7	3	4



অ্যারেটাকে আমরা গ্রাফ দিয়ে মডেলিং করলাম ডিজুয়ালাইজেশনের জন্য। আমাদের প্রবলেমটা এখন হয়ে গেলো ডিরেক্টেড অ্যাসাইন্স গ্রাফে  
লংগেস্ট পাথ বের করা!

লক্ষ্য করো আমাদের রিকার্শনে কোনো বেস কেস ডিফাইন করিনি। এই প্রবলেমে সেটা দরকার নেই কারণ এক স্টেট থেকে আরেক স্টেটে এমন  
ভাবে যাচ্ছ যে রিকার্শন এমনিই থেমে যাবে শেষে গিয়ে (আমরা সবসময় অ্যারের সামনে আগাচ্ছ, একসময় আর আগানো যাবে না)। রিকার্শন  
লুপে পড়ারও কোনো সম্ভাবনা নেই।

কোডটা লিখে ফেলি:

```

1 #define MAX_N 20
2 #define EMPTY_VALUE -1
3
4 int mem[MAX_N];
5
6 int f(int i, vector<int> &A) {
7     if (mem[i] != EMPTY_VALUE) {
8         return mem[i];
9     }
10
11     int ans = 0;
12     for (int j = i + 1;j < A.size();j++) {
13         if (A[j] > A[i]) {
  
```

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্স ম্যাট্রিক্স

অ্যাডজেসেন্স লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

14         ans = max(ans, f(j, A));
15     }
16 }
17
18 mem[i] = ans + 1;
19 return mem[i];
20 }
21
22 int findLIS(vector<int> A){
23     int ans = 0;
24
25     for(int i = 0;i<A.size();i++) {
26         mem[i] = EMPTY_VALUE;
27     }
28
29     for(int i = 0;i<A.size();i++) {
30         ans = max(ans, f(i, A));
31     }
32
33     return ans;
34 }
```

**কমপ্লেক্সিটি:** শর্টেস্ট পাথ প্রবলেমের মতোই এটারও কমপ্লেক্সিটি  $O(n * n)$ । ভিন্ন ভিন্ন স্টেট আছে  $n$  টা এবং প্রতিটি স্টেটের জন্য  $n$  সাইজের লুপ চালাতে হচ্ছে।

### ইটারেটিভ ভার্সন:

প্রবলেমটা আমরা শর্টেস্ট পাথের মত ফর্মুলায় ফেললেও এবার ইটারেটিভ ভার্সন লেখা খুব একটা কঠিন না। স্টেটগুলোকে টপলোজিকাল অর্ডারে সাজাতে গেলে দেখবে সবসময় আমরা ছোট থেকে বড় ইনডেক্সে যাচ্ছি।  $n - 1, n - 2 \dots, 1, 0$  এই অর্ডারে ইটারেটিভলি ডিপি টেবিল বিল্ডআপ করা যায় খুব সহজেই।

```

1 int lisIterative(vector<int> A) {
2     for (int i = A.size() - 1;i >= 0;i--) {
3         int ans = 0;
4         for (int j = i + 1;j < A.size();j++) {
5             if (A[j] > A[i]) {
6                 ans = max(ans, mem[j]);
7             }
8         }
}
```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শর্টেস্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/0-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেটরিঙ্ক, ডিসিশন প্রবলেম)

```

9     mem[i] = ans + 1;
10    }
11
12    int final_ans = 0;
13    for(int i = 0;i<A.size();i++) {
14        final_ans = max(final_ans, mem[i]);
15    }
16
17    return final_ans;
18 }

```

খেয়াল করো এবার আমাদেরকে আর mem টেবিলটা – 1 দিয়ে ভরে নিতে হচ্ছে না।  $i$  এর লুপটার ভিতরে বাকি অংশটা অনেকটা রিকার্সনের মতোই।  $i$  এর লুপটা 0 থেকে শুরু করলে কোডটা কাজ করতো না।

## সলিউশনের পাথ প্রিন্টিং

ইটারেটিভ বা রিকার্সিভ দুই পদ্ধতিতেই পাথ প্রিন্টিং এর নিয়ম একই। আমাদেরকে আরেকটা টেবিল মেইনটেইন করতে হবে যার নাম হতে পারে next_pointer। যখনই কোন সাবপ্রবলেম একটু বেটার রেজাল্ট দিবে তখনই টেবিলে সেভ করে বাখতে হবে কোন স্টেট থেকে কোন স্টেটে যাচ্ছি। সবশেষে প্রথম স্টেট থেকে নেক্সট পয়েন্টার ধরে ধরে পাথ খুজে বের করা যাবে।

```

1 #define MAX_N 20
2 #define EMPTY_VALUE -1
3
4 int mem[MAX_N];
5 int next_index[MAX_N];
6
7 int f(int i, vector<int> &A) {
8     if (mem[i] != EMPTY_VALUE) {
9         return mem[i];
10    }
11
12    int ans = 0;
13    for (int j = i + 1;j < A.size();j++) {
14        if (A[j] > A[i]) {
15            int subResult = f(j, A);
16            if (subResult > ans) {
17                ans = subResult;
18                next_index[i] = j;
19            }

```

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাষ্ঠার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

```

20     }
21 }
22     mem[i] = ans + 1;
23 return mem[i];
24 }
25 }

26 vector<int> findLIS(vector<int> A){
27     int ans = 0;
28
29     for(int i = 0;i<A.size();i++) {
30         mem[i] = EMPTY_VALUE;
31         next_index[i] = EMPTY_VALUE;
32     }
33
34     int start_index = -1;
35
36     for(int i = 0;i<A.size();i++) {
37         int result = f(i, A);
38         if (result > ans) {
39             ans = result;
40             start_index = i;
41         }
42     }
43
44     vector<int> lis;
45     while(start_index != -1) {
46         lis.push_back(A[start_index]);
47         start_index = next_index[start_index];
48     }
49 }
50 return lis;
51 }

```

তাহলে এই পর্বে তুমি শিখলে:

ডিপির প্রবলেমকে গ্রাফ দিয়ে ডিজুয়ালাইজেশন করলে কোন স্টেট থেকে কোন স্টেটে যাবে বুঝতে সুবিধা হ্য।

প্রতিটা স্টেটের জন্য নেক্সট পয়েন্টার রেখে রেখে পাথ প্রিন্ট করা যায়।

পর্যাকৃটিস প্রবলেম:

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETPLANET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / মার

# ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সাবসিকোয়েজ)

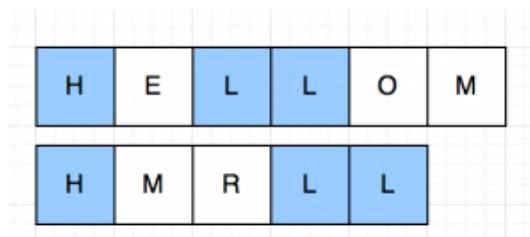
**এপ্রিল ১৯, ২০২০** by [শাফায়েত](#)



**আগের পর** গুলোতে আমরা যেসব প্রবলেম দেখে এসেছি সেগুলোর সাবপ্রবলেমের স্টেট ছিল মাত্র ১টা। এইবাবে আমরা আরেকটু জটিল সমস্যা সমাধান করবো যার নাম লংগেস্ট কমন সাবসিকোয়েজ বা LCS। এটা শেখার পরে আমি এডিট ডিসটেন্স প্রবলেম নিয়ে অন্ন কিছু কথা বলবো এবং তোমার কাজ হবে সেটা নিজে নিজে সমাধান করা।

এই প্রবলেমে তোমাকে দুটি স্ট্রিং দেয়া থাকবে  $S$  এবং  $W$ । তোমাকে তাদের মধ্যে লংগেস্ট কমন সাবসিকোয়েজ এর দৈর্ঘ্য বের করতে হবে।

সাবসিকোয়েজের সংজ্ঞাটা মনে করিয়ে দেই, একটা স্ট্রিং থেকে কিছু ক্যারেক্টার মুছে দিলে যা বাকি থাকে সেটাই স্ট্রিংটা সাবসিকোয়েজ। একটা স্ট্রিং এর  $2^n$  টা সাবসিকোয়েজ থাকতে পারে। নিচের ছবিতে দুটি স্ট্রিং এবং তাদের লংগেস্ট কমন সাবসিকোয়েজ দেখানো হয়েছে।



LCS এর দৈর্ঘ্য এক্ষেত্রে ৩।

আমাদের প্রবলেমটা হলো  $S$  এবং  $W$  এর LCS বের করতে হবে। আমরা  $S$  এর ইনডেক্সগুলো হলো  $i$  এবং  $W$  এর ইনডেক্স  $j$  দিয়ে প্রকাশ করবো।  
এখন নিচ

## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)

অ্যাপ্লিকেশন বিজ্ঞান সিরিজ

### গ্রাফ অ্যালগরিদম

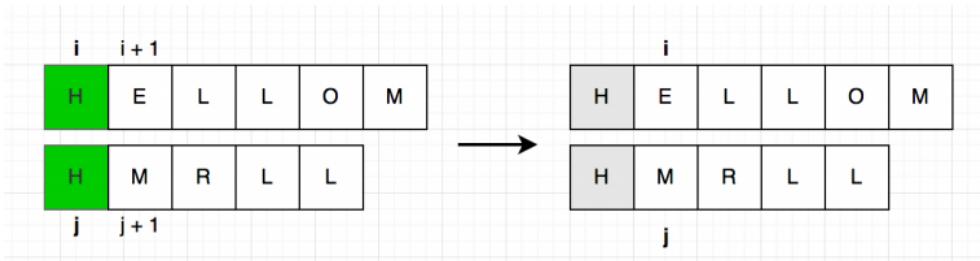
শাফায়েত আশরাফ

**d**  
বিজিক প্রকাশনা

এখন আমরা এভাবে চিন্তা করতে পারি, শুরুতে আমরা দুটি স্ট্রিং এরই প্রথম ইনডেক্স  $i = 0$  এবং  $j = 0$  তে আছি এবং আমাদেরকে  $\text{lcs}(0, 0)$  বের করতে হবে। এখন আমরা যদি যেকোনো  $(i, j)$  এর জন্য  $\text{lcs}(i, j)$  বের করতে পারি তাহলেই কিন্তু আমাদের কাজ হয়ে যায়।  $\text{lcs}(i, j)$  বলতে বুঝাচ্ছে  $i$  এবং  $j$  তম ইনডেক্স থেকে শুরু হওয়া  $S$  এবং  $W$  এর সাফিঙ্গের LCS। যেমন  $\text{lcs}(1, 2)$  হলো “ELLOM” এবং “RLL” এর LCS।

এখন আমরা যখন  $(i, j)$  তে আছি তখন ৩টি ঘটনা ঘটতে পারে:

$S[i] = W[j]$ , অর্থাৎ দুটি স্ট্রিং এই একই ক্যারেক্টার আছে। সেক্ষেত্রে ওই ক্যারেক্টারটাকে LCS এর অংশ ধরে নিয়ে আমরা বাকি সাফিঙ্গটুকুর  
LCS বের করতে পারি। তারমানে এখন আমাদের  $\text{lcs}(i + 1, j + 1)$  প্রবলেমটা সলভ করতে হবে, এখানে অনুমান করার কোনো ব্যাপার নেই।



$S[i] \neq W[j]$  হলে ব্যাপারটা আরেকটু ইন্টারেক্ষন। যেহেতু ক্যারেক্টার দুটি মিলছে না, অন্তত একটা ক্যারেক্টার আমাদের ফেলে দিতে হবে। এখন আমাদের হাতে দুটি চয়েজ,  $i$  তম ইনডেক্সটা ফেলে দিয়ে  $\text{lcs}(i + 1, j)$  ক্যালকুলেট করা, অথবা  $j$  তম ইনডেক্সটাকে ফেলে দিয়ে  $\text{lcs}(i, j + 1)$  ক্যালকুলেট করা। বুঝতেই পারছো আমরা দুটি পথে গিয়ে ম্যাক্সিমাইজ করতে চাই।

**প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম**

অনুপ্রবর্ণ(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

**অ্যালগরিদম বেসিক(৬):**

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

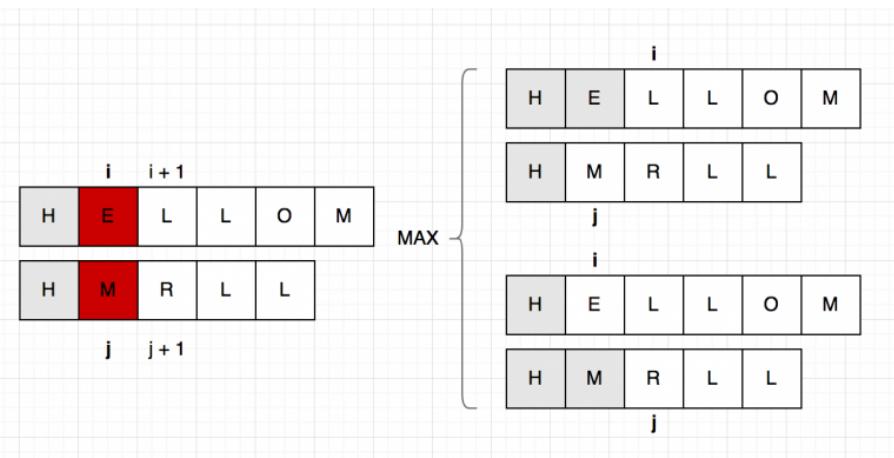
ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

**ডাটা স্ট্রাকচার(১৪):**

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



$i = n$  বা  $j = m$  হয়ে যাওয়া মানে কোন একটা স্ট্রিং এর শেষ মাথায় চলে গেছি, সেক্ষেত্রে 0 রিটার্ন করতে হবে।

আমরা আমাদের রিকার্সিভ ফর্মুলা তাহলে পেয়ে গিয়েছি:

$$\begin{aligned} lcs(i, j) &= 0 \text{ if } i = n \text{ or } j = m \\ lcs(i, j) &= lcs(i + 1, j + 1) \text{ if } S[i] = W[j] \\ lcs(i, j) &= \max(lcs(i + 1, j), lcs(i, j + 1)) \text{ if otherwise} \end{aligned}$$

এখন কোড লিখে ফেলা খুবই সহজ:

```

1 #define MAX_LEN 20
2 #define EMPTY_VALUE -1
3
4 int mem[MAX_LEN][MAX_LEN];
5
6 int lcs(int i, int j, string &S, string &W) {
7     if(i == S.size() || j == W.size()) return 0;
8
9     if(mem[i][j] != EMPTY_VALUE) {

```

শ্লাইডিং রেজ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

10     return mem[i][j];
11 }
12
13 int ans=0;
14 if(S[i] == W[j]) {
15     ans = 1 + lcs(i + 1, j + 1, S, W);
16 }
17 else{
18     int val1 = lcs(i + 1, j, S, W);
19     int val2 = lcs(i, j + 1, S, W);
20
21     ans=max(val1,val2);
22 }
23
24 mem[i][j] = ans;
25 return mem[i][j];
26 }

```

### কমপ্লেক্সিটি:

আমাদের স্টেট হলো  $(i, j)$  যেখানে  $i$  এর মান হতে পারে  $[0, n - 1]$  এর মধ্যে এবং  $j$  হতে পারে  $[0, m - 1]$  এর মধ্যে। তাহলে মোট স্টেট আছে  $n * m$  টা এবং রিকার্সনের ভিত্তির আমরা বাকি যেসব কাজ করেছি সেগুলো কমপ্লেক্সিটি কনস্টেন্ট। তাহলে মোট কমপ্লেক্সিটি হবে  $O(n * m)$ , জেনারেলাইজ করে বলা যায়  $O(n^2)$ ।

### ইটারেটিভ ভাস্তু:

ইটারেটিভ ভাস্তু লেখার জন্য আমাদেরকে বুঝতে হবে স্টেটগুলো কোন অর্ডারে আপডেট হচ্ছে (টপোলজিকাল অর্ডার)। আমরা একটু mem টেবিলটার দিকে তাকাই:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিস্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

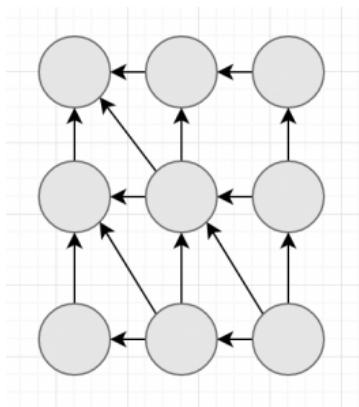
ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

	H	M	R	L	L	NULL	j
H	3	2	2	2	1	0	
E	2	2	2	2	1	0	
L	2	2	2	2	1	0	
L	1	1	1	1	1	0	
O	1	1	0	0	0	0	
M	1	1	0	0	0	0	
NULL	0	0	0	0	0	0	

i

LCS between "LLOM" & "RLL"

রিকার্সিভ ভার্সনে এই টেবিলটা তৈরি হয়েছে  $lcs(i, j)$  কল করার পর (শেষ রো এবং কলাম টেবিলের অংশ না, দেখানো হয়েছে বোধার সুবিধার জন্য)। এই টেবিলের প্রতিটি সেল  $(i, j)$  আপডেট হয়েছে হ্য তার ডান কোনার সেল  $(i + 1, j + 1)$  থেকে অথবা উপর বা নিচের সেল  $(i, + 1, j)$ ,  $(i, j + 1)$  থেকে। তো তুমি প্রতিটা সেলকে গ্রাফের একটা নোড হিসাবে কল্পনা করলে সেটা এরকম দেখাবে



[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্স(১):

ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

বোঝা যাচ্ছে নিচের ডান কোনার সেল  $(n - 1, m - 1)$  থেকে শুরু করে আমাদেরকে টেবিলটা পূরণ করতে হবে। আমরা সেটা চাইলে row by row করতে পারি। এখন আমার সাজেশন হবে উপরের উদাহরণটার জন্য তুমি হাতে-কলমে একবার টেবিলটা তৈরি করো, তাহলেই জিনিসটা মাথায় গেথে যাবে।

আমরা যদি এরকম ভাবে নেস্টেড লুপ চালাই তাহলে টপোলজিকাল অর্ডারে সেলগুলোকে পাবো:

```
1 for (int i = n - 1; i >= 0; i--) {  
2     for (int j = m - 1; j >= 0; j--) {  
3         //TODO: update table  
4     }  
5 }
```

ইটারেটিভ ডিপিতে বেস-কেস হ্যান্ডেল করতে একটু বেশি সতর্ক থাকতে হয়। আমরা যদি বেস-কেস টাকে শুরুতেই টেবিলে রেখে দিয়ে কাজ শুরু করি তাহলে জীবন কিছুটা সহজ হয়ে যায়। পুরো কোডটা হবে এরকম:

```
1 int lcsIterative(string S, string W) {  
2     int n = S.size();  
3     int m = W.size();  
4  
5     for (int i = 0; i < n; i++) mem[i][m] = 0;  
6     for (int j = 0; j < m; j++) mem[n][j] = 0;  
7  
8     for (int i = n - 1; i >= 0; i--) {  
9         for (int j = m - 1; j >= 0; j--) {  
10             if (S[i] == W[j]) {  
11                 mem[i][j] = mem[i + 1][j + 1] + 1;  
12             } else {  
13                 mem[i][j] = max(mem[i + 1][j], mem[i][j + 1]);  
14             }  
15         }  
16     }  
17  
18     return mem[0][0];  
19 }
```

শেষ করার আগে একটা মাথা খাটনোর জন্য প্রশ্ন। কিছু কিছু ক্ষেত্রে রিকার্সিভ ফাংশন ইটারেশনের থেকে দ্রুত কাজ করবে। ইটারেটিভ ডিপিকে সবসময় পুরো টেবিলটাই পূর্ণ করতে হবে কিন্তু রিকার্সিভ ডিপি পুরো টেবিল ফিলআপ না করেই অনেক ক্ষেত্রে বেজান্ট বের করে আনতে পারবে।

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

প্রশ্ন হলো কেন এরকম ঘটে এবং কি ধরনের ইনপুটের জন্য রিকার্সিভ ফাংশন ভালো কাজ করবে?

পর্যাকটিস প্রবলেম: <https://leetcode.com/problems/longest-common-subsequence/>

## রিলেটেড প্রবলেম - এডিট ডিসটেন্স

তোমাকে দুটি স্ট্রিং  $S, W$  দেয়া আছে। তুমি শুধুমাত্র  $S$  স্ট্রিংটার উপর ৩টা অপারেশন করতে পারো, কোন একটা ক্যারেক্টার বদলে দিতে পারো, কোন ক্যারেক্টার মুছে ফেলতে পারো, যেকোন পজিশনে নতুন ক্যারেক্টার ঢুকাতে পারো। তারমানে চেঞ্জ, ডিলিট, ইনসার্ট হলো তোমার ৩টা অপারেশন। এখন তোমার কাজ মিনিমাম অপারেশনে  $S$  স্ট্রিংটা টাকে  $W$  বানানো।

যেমন "blog" কে "bogs" বানাতে তুমি। মুছে ফেলে স্ট্রিং এর শেষে  $S$  ইনসার্ট করতে পারো।

হিন্টস: LCS এর মতোই দুইটা ইনডেক্স  $i, j$  কে স্টেট রাখতে হবে। এখন তুমি চিন্তা করো স্ট্রিং  $S$  থেকে কোন ক্যারেক্টার মুছে ফেললে  $i, j$  এর পরিবর্তন কিরকম হবে। ঠিক সেভাবে বাকি ২টি অপারেশনের জন্য কিভাবে  $i, j$  পরিবর্তন হবে সেটা বের করো।

পর্যাকটিস: <https://leetcode.com/problems/edit-distance/>

পরের পর্বে আমরা ক্লাসিকাল কয়েন চেঞ্জ প্রবলেম শিখবো। হ্যাপি কোডিং!

## ফেসবুকে মন্তব্য

comments

Powered by [Facebook Comments](#)

# ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ, ০-১ ন্যাপস্যাক)

এপ্রিল ২১, ২০২০ by শাফায়েত



(আগের পর্ব) আজকে আরো দুটি ক্লাসিকাল ডাইনামিক প্রোগ্রামিং প্রবলেম শিখবো। প্রথমটা হলো কয়েন চেঞ্জ। প্রবলেমটার নাম শুনেই বোঝা যাচ্ছে এটা টাকা ভাংতি করা নিয়ে প্রবলেম, তোমাকে সবথেকে কম সংখ্যক কয়েন ব্যবহার করে নির্দিষ্ট পরিমাণ টাকা ভাংতি করতে হবে।

মনে করো তোমার কাছে  $n$  টা ডিন ভিন্ন কয়েন আছে, কয়েনগুলোর ভ্যালুকে  $C_0, C_1 \dots C_{n-1}$  দিয়ে প্রকাশ করা যায়। আর তোমাকে একটা অ্যামাউন্ট দেয়া আছে  $W$ । এখন তোমাকে বলতে সবনিম্ন কয়টা কয়েন ব্যবহার করে তুমি  $W$  অ্যামাউন্টটা বানাতে পারবে। প্রতিটা ভ্যালুর কয়েন আছে মাত্র ১টা করে।

একটা উদাহরণ দেখি। ধৰা যাক কয়েনগুলোর ভ্যালু হলো  $C = \{2, 5, 9, 13, 15\}$  টাকা। এখন তুমি এই কয়েনগুলো দিয়ে  $W = 22$  বানাতে চাইলে একটা উপায় হলো  $15 + 5 + 2$ , এক্ষেত্রে কয়েন লাগছে ৩টা। কিন্তু তুমি চাইলে ২টা কয়েন ব্যবহার করেও ২২ বানাতে পারো ( $9 + 13$ )। আমাদের টাগেট কয়েন ব্যবহার মিনিমাইজ করা।

অনেকে শুরুতে এটা গ্রিডী (greedy) পদ্ধতি সমাধানের চেষ্টা করে কিন্তু সেটা কাজ করবে না। তুমি যদি সবথেকে বড় কয়েন থেকে নেয়া শুরু করো তাহলে কয়েন সংখ্যা মিনিমাইজ নাও হতে পারে যেটা উপরের উদাহরণেও দেখেছি।

আমরা আগের মতোই একটা রিকার্সিভ ফর্মুলা তৈরি করবো। প্রথমেই চিন্তা করি সাবপ্রবলেম বা স্টেট কি হবে। কয়েনগুলো কোন অর্ডারে নিতে হবে সেটা নির্দিষ্ট করা নেই, তবে আমরা সুবিধার জন্য ধরে নেই আমরা অ্যারের বাম পাশ থেকে কয়েন নেয়া শুরু করবো। তাহলে স্বাভাবিক ভাবেই মাথায় যে সাবপ্রবলেম আসবে সেটা হলো আমরা বর্তমানে কোন কয়েন নিয়ে কাজ করছি সেটা। তাহলে আমাদের রিকার্সভ ফাংশন হবে  $f(i)$ । আমরা চেষ্টা করবো  $i$  তম কয়েন থেকে  $n-1$  তম কয়েনগুলো দিয়ে  $W$  বানাতে।

সাবক্ষাইব

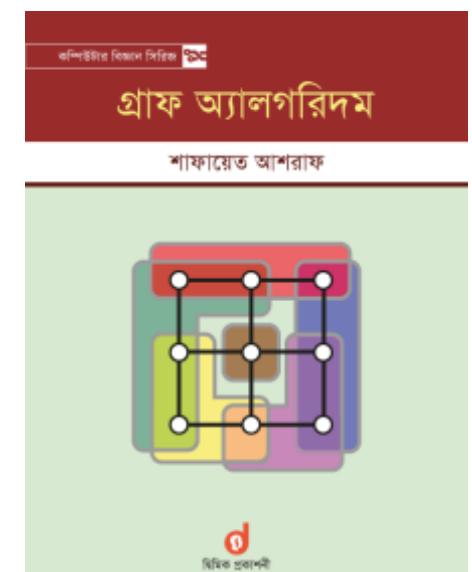


Secured by OneAll Social Login

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

**Traveloka Singapore** (বিস্তারিত...)



এখন  $i$  তম কয়েন হতে নিয়ে আমাদের দুইটা অপশন আছে:

$i$  তম কয়েনটাকে ব্যবহার করা, তাহলে পরবর্তি সাবপ্রবলেম হবে  $f(i + 1)$ ।

$i$  তম কয়েনটা ব্যবহার না করে পরের কয়েন  $(i + 1)$  এ চলে যাওয়া।, এবাবও পরবর্তি সাবপ্রবলেম হবে  $f(i + 1)$ ।

এখন আমরা একটা ঝামেলায় পড়ে গেছি, যখন এক সাবপ্রবলেম থেকে আরেক সাবপ্রবলেমে যাচ্ছি তখন আমরা জানিনা আমাদের টার্গেট অ্যামাউন্ট এখন কত। শুরুতে টার্গেট অ্যামাউন্ট  $W$  থাকলেও প্রতিবার কয়েন নিলে সেটা বদলে যাবে। আমাদেরকে কোনোভাবে সেটা মনে রাখতে হবে। মনে রাখার পদ্ধতি হলো আরেকটা স্টেট ব্যবহার করা। তাহলে আমরা নতুন করে ফাংশন ডিফাইন করি  $f(i, W)$  এবং অপশন দুটো নিয়ে আবাব চিত্ত করি:

$i$  তম কয়েনটাকে ব্যবহার করলে আমাদের টার্গেট বাকি থাকে  $W - C_i$ । তাহলে পরবর্তি সাবপ্রবলেম হবে  $f(i + 1, W - C_i)$ ।

$i$  তম কয়েনটাকে ব্যবহার না করলে আমাদের টার্গেট পরিবর্তন হবে না। তাহলে পরবর্তি সাবপ্রবলেম হবে  $f(i + 1, W)$ ।

আমাদেরকে দুইটা সাবপ্রবলেমের মিনিমাম না নিতে হবে। প্রথম অপশনে যেহেতু কয়েন নিচ্ছি তাই 1 যোগ করতে হবে। এবাব তাহলে ফর্মুলাটা লিখে ফেলি:

$$\begin{aligned} f(i, W) &= 0 \text{ if } W = 0 \\ f(i, W) &= \inf \text{ if } W < 0 \\ f(i) &= \min(1 + f(i + 1, W - C[i]), f(i + 1, W)) \text{ otherwise} \end{aligned}$$

এখন কোড লিখে ফেলা খুবই সহজ:

```
1 #define MAX_N 20
2 #define MAX_W 10000
3
4 #define INF 99999999
5 #define EMPTY_VALUE -1
6
7 int C[MAX_N];
```



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে  
হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

```

8 int mem[MAX_N][MAX_W];
9 int n;
10
11 int f(int i, int W) {
12     if (W < 0) return INF;
13     if (i == n) {
14         if (W == 0) return 0;
15         return INF;
16     }
17
18     if (mem[i][W] != EMPTY_VALUE) {
19         return mem[i][W];
20     }
21
22     int res_1 = 1 + f(i + 1, W - C[i]);
23     int res_2 = f(i + 1, W);
24
25     mem[i][W] = min(res_1, res_2);
26
27     return mem[i][W];
28 }
29 }
```

কোনো অ্যামাউন্ট যদি বানানো না যায় তাহলে এই কোড INF রিটুর্ন করবে।

### কমপ্লেক্সিটি:

আমাদের সাবপ্রবলেম সংখ্যা  $n * W$  এবং প্রতিটা সাবপ্রবলেম অন্য সাবপ্রবলেম কল করা ছাড়া বাকি কাজ constant টাইমে করছি। টাইম কমপ্লেক্সিটি হবে  $O(nW)$ । (এটা কিন্তু পলিনোমিয়াল কমপ্লেক্সিটি না, এটা সুড়োপলিনোমিয়াল, তুমি গুগল করে এ বিষয়ে জেনে নিতে পারো)

### ইটারেটিভ ভার্সন:

আমরা জানি ইটারেটিভ ভার্সনে যে সাবপ্রবলেমটা প্রথমে সলভ হচ্ছে সেখান থেকে টেবিল বিল্ডআপ শুরু করতে হয়। আমাদের ফর্মুলায়  $i$  এর মান 0 থেকে সামনের দিকে যাচ্ছে এবং  $W$  এর মান পিছন দিকে যাচ্ছে। আমরা টেবিল বিল্ডআপ শুরু করবো  $(n-1, 0)$  সেল থেকে। তাহলে প্রতিটা সাবপ্রবলেম টপোলজিকাল অর্ডারে আপডেট হবে। বুরতে সমস্যা হলে আমার সাজেশন হবে  $n * W$  সাইজের একটা টেবিল খাতায় একে ঘরগুলো হাতে-কলমে আপডেট করা।

স্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

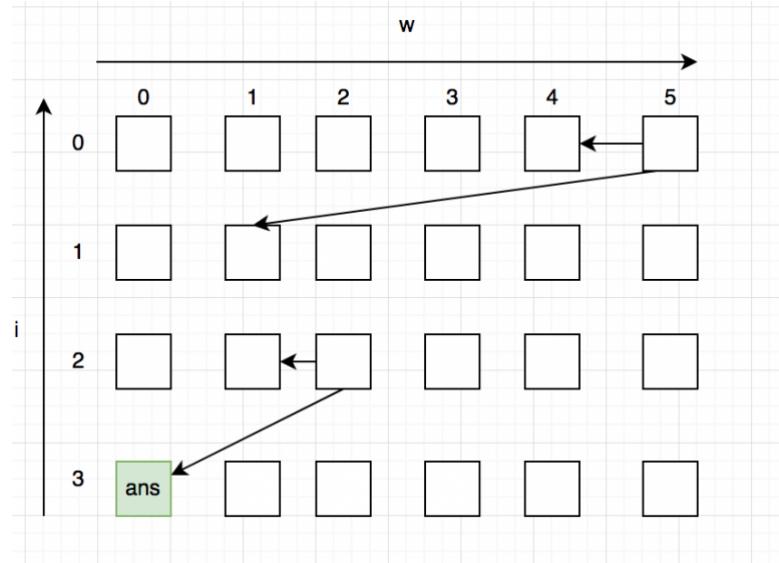
টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



আমি একটা আবিট্রোরি টেবিল একে দেখিয়ে দিলাম উত্তরটা কোন কোনায় থাকবে এবং সেলগুলো কোন ডিরেকশনে আপডেট হবে। তোমার কাজ হবে ঠিকমত আপডেট করা।

ইটারেটিভ ডিপিতে কর্নার কেস নিয়ে একটু সাবধান থাকতে হয় কারণ রিকার্সিভ ফাংশনের জায়গায় সরাসরি টেবিল থেকে আপডেট করছি, উল্টাপাটা ইনডেক্সে এক্সেস করলে বানটাইম এরো দিবে। এখানে আমি যেটা করি, টেবিল এক্সেস করার কোডটুকু একটা নতুন ফাংশনের মাধ্যমে করি।

```

1 int evaluate_table(int i, int W, int n) {
2     if (W < 0) return INF;
3     if (i == n) {
4         if (W == 0) return 0;
5         return INF;
6     }
7
8     return mem[i][W];
9
10}
11
12 int coin_change_iterative(int n, int target) {
13     for (int i = n - 1;i >= 0;i--) {

```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

```

14     for (int w = 0; w <= target; w++) {
15
16         int res_1 = 1 + evaluate_table(i + 1, w - C[i], n);
17         int res_2 = evaluate_table(i + 1, w, n);
18         mem[i][w] = min(res_1, res_2);
19     }
20 }
21
22 return mem[0][target];
23 }
```

এখনে একটা দারুণ মেমরি অপটিমাইজেশনের সুযোগ আছে। খেয়াল করো, ভিতরের লুপে আমাদের খালি  $i + 1$  ৰো এর ড্যালুগুলো লাগছে। তারমানে  $i$  এর মান যখন  $x$  তখন খালি তোমার row  $x + 1$  এর ড্যালুগুলো দরকার হবে, বাকিগুলো টেবিল আর কোনো কাজে লাগবে না। তারমানে বর্তমান ৰো এবং তার আগের ৰো ছাড়া বাকিগুলো কাজে লাগছে না। এভাবে তুমি  $2 * W$  সাইজের টেবিল ব্যবহার করেই প্রবলেমটা সলভ করতে পারো।

### ড্যারিয়েশন:

এবাব আমরা কিছু ড্যারিয়েশন দেখবো কয়েন চেঞ্জেৰ। মনে করো এখনো আমাদের কয়েনের ড্যালুগুলো একই আছে  $C = \{2, 5, 9, 13, 15\}$ , কিন্তু এবাব প্রতিটা কয়েনের সাপ্লাই অসীম। আগে 30 বানাতে কয়েন লাগতো ঠটা ( $15 + 13 + 2$ ) কিন্তু এখন লাগবে ২টা ( $15 + 15$ )।

এটা করা যাবে কোডে মাত্র ১টা ক্যারেটার মুছে দিয়ে। পৱের লাইন পড়াৰ আগে একটু নিজে নিজে চিন্তা করো।

আগে আমরা একটা কয়েন নিলে পৱের স্টেট  $f(i + 1, W - C[i])$  এ চলে যাচ্ছিলাম, তাই একটা কয়েন একবাৰেৰ বেশি নেয়া হচ্ছিলো না। আমরা যদি  $i$  তম কয়েন নেয়াৰ পৱ একই কয়েনে থেকে যাই, খালি  $W$  আপডেট কৰি তাহলেই কিন্তু কাজ হয়ে যাবে। ফৰ্মুলাটা এবাব হবে:

$$f(i) = \min(1 + f(i, C - W[i]), f(i + 1, W))$$

আগেৰটাৰ সাথে পাৰ্থক্য হলো প্ৰথম  $i + 1$  টাকে  $i$  কৰে দিয়েছি।

```

1 #define MAX_N 20
2 #define MAX_W 10000
```

[পুৱানো সিৱিজ]

বিটমাস্ক ডিপি

মিনিমাম ভাৱটেক্স কভাব(গ্ৰাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটোৱ

নাষ্ঠাৰ থিওৰি/গণিত(৫):

মডুলাৰ অ্যারিথমেটিক

প্ৰাইম জেনারেটোৱ (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিৱেজমেন্ট

প্ৰোৰাবিলিটি: এক্সপেন্সিভ ড্যালু

স্ট্ৰিং ম্যাচিং(২):

ৱিন-কাৰ্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিৱেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকাৰ্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটাৱ(২)

```

3 #define INF 99999999
4 #define EMPTY_VALUE -1
5
6
7 int C[MAX_N];
8 int mem[MAX_N][MAX_W];
9 int n;
10
11 int f(int i, int W) {
12     if (W < 0) return INF;
13     if (i == n) {
14         if (W == 0) return 0;
15         return INF;
16     }
17
18     if (mem[i][W] != EMPTY_VALUE) {
19         return mem[i][W];
20     }
21
22     int res_1 = 1 + f(i, W - C[i]); //only this line updated
23     int res_2 = f(i + 1, W);
24
25     mem[i][W] = min(res_1, res_2);
26
27     return mem[i][W];
28 }
```

এখন আমাদের আরেকটু অপটিমাইজেশনের জায়গা তৈরি হয়েছে, আমরা চাইলে স্টেট কমিয়ে স্পেস বাচাতে পারি। এখন যেহেতু আমাদের আনলিমিটেড কয়েন আছে, আমরা কোন কয়েন কয়বার নিছি সেটা নিয়ে আর চিন্তা করতে হচ্ছে না। আমরা চাইলে সাবপ্রবলেমকে খালি  $f(W)$  দিয়ে ডিফাইন করতে পারি। আমরা বের করতে চাই  $W$  বানাতে কয়টা কয়েন লাগবে, কোন কয়েন আমি এখন নিবো সেটা ব্যাপার না। এরপর ফাংশনের ডিতরে একটা লুপ চালিয়ে একটা একটা করে কয়েন নিয়ে চেষ্টা করবো।

$$\begin{aligned}f(W) &= 0 \text{ if } W = 0 \\f(W) &= \text{inf} \text{ if } W < 0 \\f(W) &= \min(f(W - C_i)) \text{ for } i \in [0, n]\end{aligned}$$

কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



Be the first of your friends to like this



ডায়ানামিক প্রোগ্রামিং এর নতুন  
সিরিজের ৬ নম্বর লেখায় আমরা  
দেখবো কম্বিনেটরিক্যার প্রবলেম এবং  
ডিসিপ্লিন প্রবলেম কিভাবে সমাধান  
করতে হ্য।

$i, W) = 1 \text{ if } W =$   
 $i + 1, W - C[i]) -$

SHAFAEETSPPLANET.COM  
ডায়ানামিক প্রোগ্রামিং ৬ / ম্যাচ

এবার টাইম কমপ্লেক্সিটি একই থাকলেও টেবিলের সাইজ অনেক কমে যাবে। স্টেট কমিয়ে দিয়ে লুপ ব্যবহার করে মেমরি কমানো ডাইনামিক প্রোগ্রামিং এর খুবই কমন একটা ট্রিক।

```
1 #define MAX_N 20
2 #define MAX_W 10000
3
4 #define INF 99999999
5 #define EMPTY_VALUE -1
6
7 int C[MAX_N];
8 int mem[MAX_W];
9 int n;
10
11 int f_optimized(int W) {
12     if (W < 0) return INF;
13     if (W == 0) return 0;
14
15     if (mem[W] != EMPTY_VALUE) {
16         return mem[W];
17     }
18
19     int ans = INF;
20     for (int i = 0;i < n;i++) {
21         ans = 1 + f_optimized(W - C[i]);
22     }
23
24     mem[W] = ans;
25     return mem[W];
26 }
```

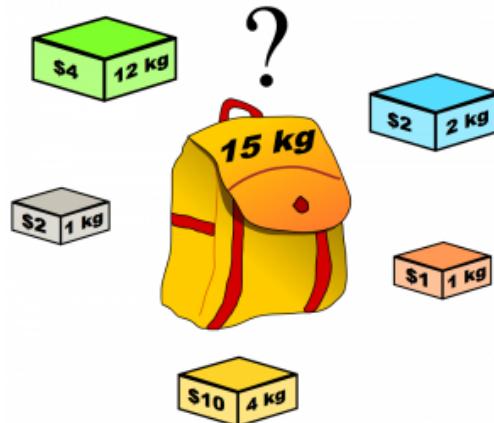
এখন একটা বাড়ির কাজ: যদি বলা হয় প্রতিটা কয়েন সর্বোচ্চ  $k$  বার ব্যবহার করা যাবে তাহলে কি করবে?

## ০-১ ন্যাপস্যাক:

তুমি কয়েন চেঙ্গ বেশ কয়েকভাবে সলভ করতে শিখে গিয়েছো, তাই 0-1 ন্যাপস্যাক আমি সলভ করে দিব না। আমি প্রবলেমটা কি সেটা বলবো এবং কিছু হিটস দিবো।

ন্যাপস্যাক এর বাংলা হলো খলে বা ব্যাগ। তোমার কাছে একটা ব্যাগ আছে যারা নির্দিষ্ট একটা ক্যাপাসিটি আছে, ধরলাম সেই ক্যাপাসিটি হলো  $C$ ।

এখন তোমার সামনে  $n$  টা আইটেম আছে, প্রতিটা আইটেমের নির্দিষ্ট দাম এবং ওজন আছে। নিচের ছবিটা wikipedia থেকে নেয়া:



এখন তোমাকে বলতে হবে তুমি সর্বোচ্চ কত দামের জিনিস ব্যাগে ভরতে পারবে। “ $0-1$ ” ন্যাপস্যাক বলার কারণ হলো কোনো জিনিস নিলে পুরোটাই নিতে হবে, ডেঙে অর্ধেক করে নিতে পারবে না। ডেঙে নেয়ার নিয়ম থাকলে সেই প্রবলেমটাকে ফর্যাকশনাল ন্যাপস্যাক বলে।  
ফর্যাকশনালের ক্ষেত্রে দামি জিনিসগুলো আগে নিলেই অপটিমাল রেজাল্ট পাওয়া যায়,  $0-1$  ন্যাপস্যাক এ সেটা কাজ করবে না।

এখনে ইনপুট হিসাবে দেয়া হবে দুটি অ্যারে  $P$  এবং  $W$ ।  $i$  তম বস্তুর দাম  $P_i$  এবং ওজন  $W_i$ ।

আমাদের সাবপ্রবলেম আগের মতোই হবে  $f(i, C)$  যা দিয়ে বুঝাবে  $i$  থেকে শুরু করে  $n - 1$  তম আইটেম গুলো দিয়ে পাওয়া সর্বোচ্চ প্রফিট। সেখান থেকে আমাদের দুইটা চয়েস

$i$  তম আইটেম ব্যাগে না ভরা, তাহলে পরবর্তী সাবপ্রবলেম হবে  $f(i + 1, C)$

$i$  তম আইটেম নেয়া, পরবর্তি সাবপ্রবলেম হবে  $f(i + 1, C - W_i)$ ।

আমাদেরকে এই দুইটার মধ্যে থেকে বড়টাকে নিতে হবে। ২য় ক্ষেত্রে প্রফিট হবে  $P_i$ , সেটাও যোগ করতে হবে।

$$f(i, C) = \max(f(i + 1, C), f(i + 1, C - W_i) + P_i)$$

তোমার কাজ হবে এই ফর্মুলার রিকাৰ্ড এবং ইটাৰেটিভ ভাৰ্সন লেখা এবং ইটাৰেটিভ ভাৰ্সনেৰ মেমৰি অপটিমাইজ কৰা।

আজ এই পঞ্চাত্তি। তুমি ডাইনামিক প্ৰোগ্ৰামিং এৱ অনেকগুলো ট্ৰিকস শিখে ফেলেছো এৱমধ্যেই, এখন বেশি কৰে প্ৰবলেম সলভ কৱলে জিনিসটা আয়োজন কৰে। তুমি এখন পৰ্যন্ত যেসব প্ৰবলেম দেখেছো সেগুলোতে ৱেজাল্ট ম্যাক্সিমাইজ বা মিনিমাইজ কৱতে হয়, এৱপৰ আমৰা দেখবো কম্পিউটাৰে কিছু প্ৰবলেম।

পৰ্যাকৃটিস প্ৰবলেম:

<https://leetcode.com/problems/coin-change/>

<https://leetcode.com/problems/minimum-cost-for-tickets/>

<https://www.spoj.com/problems/KNAPSACK/>

ফেসবুকে মন্তব্য

0 comments

# ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্পিউটেরিক্স, ডিসিশন প্রবলেম)

এপ্রিল ২২, ২০২০ by শাফায়েত



আগের পর্ণগুলোয় যেসব প্রবলেম দেখেছি তাৰ মধ্যে ফিবোনাচি সবগুলোতেই আমাদেৱকে কিছু না কিছু ম্যাঞ্চিমাইজ বা মিনিমাইজ কৰতে হয়।

এগুলো ছাড়া ডাইনামিক প্রোগ্রামিং এৰ আৰো কিছু ব্যবহাৰ আছে, একটা হলো কোন একটা কাজ কত ভাৱে কৱা যায় সেটা বেৰ কৱা, আৱেকটা হলো ডিসিশন প্রবলেম সলভ কৱা (অৰ্থাৎ কোন একটা কাজ কৱা যাবে কি যাবে না সেটা বেৰ কৱা)।

শুৰুতেই দেখবো সাবসেট সাম প্রবলেম। এই প্রবলেমটা অনেকটাই কয়েন চেঞ্জ প্রবলেমের মত, আশা কৱবো তুমি **কয়েন চেঞ্জ নিয়ে লেখাটা** পড়ে ফেলেছো, কাৰণ এবাৰ আমি আগেৰ মত এত বিস্তারিত বৰ্ণনা কৱবো না।

## সাবসেট সাম

তোমাকে একটা ইন্টিজাৰ অ্যারে  $C$  দেয়া আছে এবং একটা ড্যালু  $W$  দেয়া আছে। তোমাকে বলতে হবে  $C$  এৰ আইটেমগুলো দিয়ে কতভাৱে  $W$  বানানো যায়।

যেমন ধৰা যাক  $C = \{5, 15, 3, 17, 12\}$  এবং  $W = 20$ । আমৰা ৩ ভাৱে ২০ বানাতে পাৰি ->  
 $(5 + 15), (3 + 17), (5 + 3 + 12)$ ।

একটা  $n$  সাইজেৰ অ্যারেৰ  $2^{n-1}$  টা non-empty সাবসেট থাকে। আমৰা ডাইনামিক প্রোগ্রামিং ব্যবহাৰ কৱলে সবগুলো সাবসেট বেৰ কৰতে হবে না।

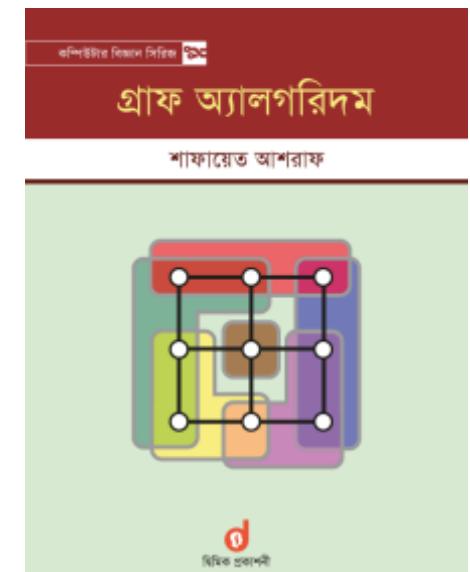
সাবক্ষাইব



Secured by [OneAll Social Login](#)

আমাৰ সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়াৰ @  
**Traveloka Singapore** (বিস্তাৰিত...)



কয়েন চেঙ্গ প্রবলেমের মতোই আমাদের সাবপ্রবলেম হলো  $f(i, W)$ , অর্থাৎ আমরা বের করতে চাই  $i$  থেকে  $n - 1$  তম আইটেমগুলো নিয়ে কর্তব্য  $W$  বানানো যায়। আমাদের আবারো দুইটা চয়েজ:

$i$  তম আইটেমটা ব্যবহার করলে পরের সাবপ্রবলেম হবে  $f(i + 1, W - C[i])$

$i$  তম আইটেমটা ব্যবহার না করলে পরের সাবপ্রবলেম হবে  $f(i + 1, W)$

আগেরবার আমরা দুটো সাবপ্রবলেমের মিনিমাম বা ম্যাক্সিমাম নিয়েছিলাম, এবার জাস্ট দুটোর বেজাল্ট যোগ করে দিতে। বেস কেস হবে  $W = 0$ , সেক্ষেত্রে আমরা টাগেটি পৌছে গেছি, এসময় আমাদেরকে 1 রিটার্ন করতে হবে।

$$f(i, W) = 1 \text{ if } W = 0$$

$$f(i, W) = f(i + 1, W - C[i]) + f(i + 1, W)$$

কোডটা লিখে ফেলি:

```

1 #define MAX_N 20
2 #define MAX_W 10000
3
4 #define EMPTY_VALUE -1
5
6 int C[MAX_N];
7 int mem[MAX_N][MAX_W];
8 int n;
9 int f(int i, int W) {
10    if (W == 0) return 1;
11    if (i == n + 1) return 0;
12
13    if (mem[i][W] != EMPTY_VALUE) {
14        return mem[i][W];
15    }
16
17    int way_1 = f(i + 1, W);
18    int way_2 = f(i + 1, W - C[i]);
19
20    mem[i][W] = way_1 + way_2;

```



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ন্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

```
21     return mem[i][W];  
22 }
```

কমপ্লেক্সিটি কয়েন চেঞ্জের মতোই সুড়োপলিনোমিয়াল  $O(n * W)$ । ইটারেটিভ ভার্সন আমি আর দেখাচ্ছি, এখন তুমি নিজেই সেটা লিখতে পারবে।

## সাবসেট সাম ডিসিশন প্রবলেম

এখন আমাদেরকে যদি বলতো  $W$  কতভাবে বানানো যাবে সেটা বের করা দরকার নেই,  $W$  বানানো যাবে নাকি যাবে না সেটা বের করে দাও। আগের কোডটা ব্যবহার করেই সেটা করা যাবে, উত্তর পজিটিভ মানে  $W$  বানানো যায়। তবে সেটা আরো সহজে এবং কম মেমরি ব্যবহার করেই করা যায়।

এক্ষেত্রে আমরা ইন্টিজার রিটার্ন না করে বুলিয়ান true/false রিটার্ন করবো। যদি দুটি সাবপ্রবলেমের অন্তর্গত একটা true রিটার্ন করে তাহলে রেজাল্ট হবে true।

$$\begin{aligned}f(i, W) &= \text{true if } W = 0 \\f(i, W) &= f(i + 1, W - C[i]) \text{ || } f(i + 1, W) \text{ otherwise}\end{aligned}$$

এখনে || হলো লজিকাল OR অপারেটর।

## কষ্টনেশন

তোমার কাছে  $n$  টা আইটেম আছে, সেখান থেকে  $r$  টা আইটেম তুলে নিতে হবে। কতভাবে করা যায়? এটা ক্লাসিক  nC_r  প্রবলেম যেটা হাইস্কুলেই সবাই পড়ে এসেছে। তখন আমরা এটার একটা ফর্মুলা মুখ্যস্থ করে ফেলতাম:  $\frac{n!}{(n - 1)!r!}$ । এখন দেখি এটাকে বিকার্সিভলি কিভাবে লেখা যায়।

আমরা একটা ফাংশন ডিফাইন করি  $f(n, r)$ । এখন হাতে দুইটা চয়েজ:

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কমপ্লেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

একটা আইটেম তুলে নেয়া, তাহলে যাকবে মোট  $n - 1$  টা আইটেম এবং তুলতে হবে আরো  $r - 1$  টা আইটেম, অর্থাৎ

$$f(n - 1, r - 1)$$

একটা আইটেম ফেলে দেয়া, তাহলে যাকবে মোট  $n - 1$  টা আইটেম এবং তুলতে হবে আরো  $r$  টা আইটেম, অর্থাৎ  $f(n - 1, r)$

তাহলে আমরা উভয় পেয়ে যাবো  $f(n - 1, r - 1)$  আর  $f(n - 1, r)$  যোগ করে দিলেই। বেসকেস সহ ফর্মুলা হবে:

$$f(n, 0) = f(n, n) = 1$$

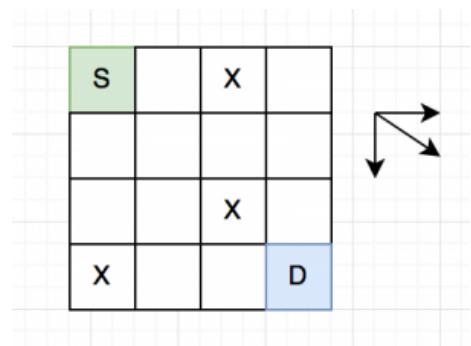
$$f(n, r) = 0 \text{ if } r > n$$

$$f(n, r) = f(n - 1, r - 1) + f(n - 1, r) \text{ otherwise}$$

এটার কোড আর দিব না, তুমি নিজেই লিখতে পারবে।

## কতগুলো পথ?

তোমাকে একটা  $n * m$  সাইজের ২ডি গ্রিড দেয়া আছে। তুমি বর্তমানে আছো গ্রিডের উপরের বাম কোণয়  $(0, 0)$ , তোমাকে সেখান থেকে  $(n - 1, m - 1)$  ঘরে যেতে হবে। বলতে হবে কতগুলো ভিন্ন ভিন্ন উপায়ে যাওয়া যায়। তুমি শুধু মাত্র ৩ ডি঱েকশনে যেতে পারো, ডানে, নিচে বা কোনাকুনি। গ্রিডের কিছু কিছু সেল রক করা আছে, সেগুলোতে যাওয়া যাবে না।



৩ ডি঱েকশনের শর্তটা দেয়া হয়েছে যাতে তুমি ঘুরেফিরে একই সেলে ফিরে আসতে না পারো, সেক্ষেত্রে সাইকেল তৈরি হয়ে যাবে, DAG যাকবে না।

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

এই প্রবলেমটা খুবই সহজ। তোমার সাবপ্রবলেম হবে  $f(i, j)$  এবং এখন থেকে তোমার টটা চয়েজ আছে। কি কি চয়েজ নিশ্চয়ই বুঝতে পারছে, সেগুলো যোগ করে দিলেই উত্তর বের হয়ে আসবে।

$$f(n - 1, m - 1) = 1$$

$f(i, j) = 0$  if  $grid_{i,j} = X$  or  $i, j$  is outside the grid

$f(i, j) = f(i + 1, j) + f(i, j + 1) + f(i + 1, j + 1)$  otherwise

এখন আমরা যদি এটাকে ডিসিশন প্রবলেম হিসাবে চিন্তা করি, বের করতে চাই গন্তব্যে পৌছানো যায় কি যায় না তাহলে কি করবে? উত্তর গুলো যোগ না করে লজিকাল OR করে দিবে।

পর্যাকটিস প্রবলেম:

<https://leetcode.com/problems/unique-paths/>

<https://leetcode.com/problems/pascals-triangle/>

<https://leetcode.com/problems/partition-equal-subset-sum/>

ফেসবুকে মন্তব্য

0 comments

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেক্ষেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

চেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

# বিটমাস্ক ডাইনামিক প্রোগ্রামিং

ডিসেম্বর ১৭, ২০১২ by শাফায়েত



**[নোটিস (১৮ এপ্রিল ২০২০): ডাইনামিক প্রোগ্রামিং নিয়ে পুরানো লেখাগুলো আমি ধীরে ধীরে সরিয়ে ফেলছি এবং **নতুন সিরিজ শুরু করছি। এই আর্টিকেলটি এখনো আপডেট করা হয় নি]****

আশা করি তুমি এখন [lis, knapsack, coin-change](#) প্রবলেম সলভ করতে পারো খুব সহজেই, ডিপির সলিউশন প্রিন্ট করতেও তোমার সমস্য হ্যন। এখন আমরা একটু অন্যরকম ডিপি দেখবো যেটার নাম বিটমাস্ক ডিপি। নামটা শুনে ভয় লাগলেও জিনিসটা সহজ, অনেক ক্ষেত্রেই বিটমাস্ক ডিপি প্রবলেম পড়ার সাথে সাথে সলিউশন মাথায় চলে আসে। তবে এই পর্টা পড়ার আগে তোমাকে বিট নিয়ে কাজ করা শিখতে হবে, যেমন কোনো নির্দিষ্ট পজিশনের বিট অন করা/অফ করা ইত্যাদি। এজন্য তুমি এই চমৎকার [টিউটোরিয়ালটা](#) দেখতে পারো, পুরোটা খুবই ভালো করে পড়বে কারণ এটা তোমাদের অনেক জায়গায় কাজে লাগবে। আমি এই টিউটোরিয়ালে বিট অপারেশন নিয়ে লিখছিনা কারণ অপ্রাসঙ্গিক হয়ে যাবে।

আমরা শুরুতেই ৩টি ফাংশন ডিফাইন করি।

```

1 int Set(int N,int pos){return N=N | (1<<pos);}
2 int reset(int N,int pos){return N= N & ~(1<<pos);}
3 bool check(int N,int pos){return (bool)(N & (1<<pos));}

```

Set ফাংশনটি N সংখ্যাটির pos তম পজিশনের বিট 1 করে দেয়, reset ফাংশনটি 0 করে দেয় এবং check ফাংশনটি pos তম বিটে কি আছে সেটা রিটার্ন করে। যেকোনো বিটমাস্ক ডিপি প্রবলেমে ফাংশন ৩টি দরকার হবে, বিশেষ করে Set এবং check।

## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

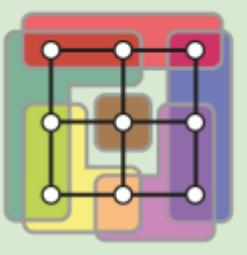
শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

**Traveloka Singapore** (বিস্তারিত...)

অ্যাপ্লিকেশন বিজ্ঞান সিরিজ

## গ্রাফ অ্যালগরিদম

শাফায়েত আশরাফ



d
  
 বিজিক প্রকাশনা

আমরা একটা প্রবলেম দিয়ে শুরু করি। মনে করো তোমাকে  $n$ টা দোকান থেকে  $n$  টা জিনিস কিনতে হবে। জিনিসগুলো কিনতে তোমার  $a_0, a_1, a_2, \dots, a_{(n-1)}$  টাকা লাগে। তোমার শহরটা খুব অঙ্গুত, তুমি যখন একটা জিনিস কিনে আরেক দোকানে যাও তখন সেই দোকানদার তোমার আগের কেনা জিনিসগুলো দেখে তার দোকানের জিনিসের দাম বাড়িয়ে দেয়!! কত দাম বাড়াবে সেটা নির্ভর করবে তুমি আগে আগে কোন কোন দোকানে গিয়েছো সেটার উপর। ধরো  $n=2$ , তাহলে তোমাকে নিচের মতো একটা ম্যাট্রিক্স দেয়া থাকবে:

“ 10 10  
90 10

এখন,

“ যদি ( $i==j$ ) হয় তাহলে  $\text{matrix}[i][j]=\text{matrix}[i][i]=i$ ’তম জিনিসটির আসল দাম।  
যদি ( $i!=j$ ) হয় তাহলে  $\text{matrix}[i][j]=j$ ’তম জিনিসটি আগে কিনলে  $i$ ’তম জিনিসটির সাথে যোগ হওয়া  
বাড়তি দাম।

তুমি যদি শুরুতে 0 তম জিনিসটা কিনো তাহলে দাম পড়বে 10টাকা, এবপর 1নম্বর জিনিসটা কিনলে সেটার দাম হবে  $10+90$  টাকা, কারণ  $\text{matrix}[1][0]=0$  নম্বর জিনিসের আগে 1 নম্বর জিনিস কিনলে যোগ হওয়া বাড়তি দাম= $90$  আর 1 নম্বর জিনিসের আসল দাম= $10$ , তাহলে মোট খরচ  $10+(10+90)=110$ । কিন্তু তুমি যদি 1নম্বর জিনিসটা আগে কিনো তাহলে মোট খরচ  $10+(10+10)=30$  টাকা।  
বুঝতেই পারছো তোমার কাজ হলো খরচ মিনিমাইজ করা।  $n$  এর মান সর্বোচ্চ ১৫।

$n$  এর মান খুব কম বলে বিটমাস্ক ডিপি দিয়ে প্রবলেমটি সহজেই সলভ করা যাবে। ডিপিতে আমাদের প্রথম কাজ হলো স্টেট নির্ণয় করা। এই কেনাকাটার যেকোনো সময় আমাদের অবস্থা কি কি তথ্য দিয়ে প্রকাশ করা যায়? “এখন পর্ষ্ণ কোন কোন জিনিস কেনা হয়েছে” এই তথ্যটাই যথেষ্ট, তাইনা? এটা জানলে আমরা পরবর্তি আরেকটি জিনিস কেনার সময় বাড়তি কর খরচ যোগ হবে জানতে পারবো, পরবর্তিতে যেই জিনিসটা কিনলে মোট খরচ কম হবে সেটা আমরা কিনবো। মনে করি এখন কথা হলো স্টেটটা বাখবো কি ভাবে?



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইনেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

একটা উপায় হলো  $n$  টি জিনিসের জন্য এভাবে  $n$ টা স্টেট রাখা function( $a_0, a_1, a_2, \dots, a_{n-1}$ ), কিন্তু  $n$  এর মান বদলালে তুমি প্যারামিটার সংখ্যা বদলাবে কি ভাবে? আর ১৫টি প্যারামিটার নিয়ে কাজ করলে কোডটা ডয়াবহ জটিল হয়ে যাবে।

২য় উপায় হলো বিটমাশ্ক। একটি ইটিজাবে ৩২টি বিট থাকে। আমরা সেই সুবিধাটাই নিবো। ১ নম্বর বিট ১ হলে আমরা ১ নম্বর জিনিসটা নিয়েছি, ০ হলে নেইনি। ৩ নম্বর বিট ১ হলে আমরা ৩ নম্বর জিনিসটা নিয়েছি, ০ হলে নেইনি। ১ এবং ৩ নম্বর বিট দুইটাই ১ হলে আমরা ২টা জিনিসই নিয়েছি।

শুরুতে আমাদের স্টেট থাকবে ০ বা বাইনারিতে “000000”。 তারমানে আমরা কোনো জিনিস এখনও কিনিনি। ০তম এবং ১তম জিনিস কেনা হয়ে গেলে স্টেট হবে ৩ বা “000011” এবং  $n = 2$  এর জন্য এটাই আমাদের base case।  $n=4$  এর জন্য base case হলো 15 বা “0001111”。 leading zero নিয়ে চিন্তা করা দরকার নেই, এটা বোঝার সুবিধার্থে দেয়া হয়েছে। একটু চিন্তা করলেই বুঝতে পারবে  $mask = (2^n) - 1$  হলে সেটা হবে base case কারণ তখন বাইনারিতে প্রথম  $n$  টা বিট ১ থাকবে, আমরা তখন শুন্য রিটোর্ন করে দিবো তখন কারণ আর কোনো জিনিস কেনা বাকি নেই।

```
1 int dp[(1 << 15) + 2];
2 int call(int mask)
3 {
4     if (mask == (1 << n) - 1)
5         return 0;
6     if (dp[mask] != -1)
7         return dp[mask];
8     //Rest of the calculation
9 }
```

বেসকেস বুঝলাম, এবপরে আমাদের কাজ হবে যেসব জিনিস কেনা হ্যানি সেগুলা নিয়ে চেষ্টা করে দেখা।  $mask$  এর  $i$  তম পজিশনে যদি 0 থাকে তাহলে  $i$  তম জিনিসটি কেনা এখনও বাকি আছে।

```
1 int dp[(1 << 15) + 2];
2 int call(int mask)
3 {
4     if (mask == (1 << n) - 1)
5         return 0;
6     if (dp[mask] != -1)
7         return dp[mask];
8     //Rest of the calculation
9     int ans = 1 << 28; //Infinite, a large value
10    for (int i = 0; i < n; i++) {
```

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্স ম্যাট্রিক্স

অ্যাডজেসেন্স লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

11     if (check(mask, i) == 0) {
12         //Rest of the code
13     }
14 }
15 return dp[mask] = ans;
16 }
```

আমরা  $n$  পর্যন্ত লুপ চালিয়ে বের করে নিলাম কোনটা কোনটা নেয়া বাকি আছে। এখন  $i$  তম জিনিসটার আসল দাম হলো  $price = w[i][i]$ । এই  $price$  এর সাথে  $w[i][j]$  যোগ হবে যদি  $i! = j$  হয় এবং  $j$  নম্বর জিনিসটা আগেই কেনা হয়ে থাকে।  $mask$  এর  $j$  তম বিট চেক করে আমরা বলতে পারি  $j$  তম জিনিসটা কেনা হয়েছে নাকি।

```

1 int dp[(1 << 14) + 2];
2 int call(int mask)
3 {
4     if (mask == (1 << n) - 1)
5         return 0;
6     if (dp[mask] != -1)
7         return dp[mask];
8     int ans = 1 << 28;
9     for (int i = 0; i < n; i++) {
10         if (check(mask, i) == 0) {
11             int price = w[i][i];
12             for (int j = 0; j < n; j++)
13                 if (i != j and check(mask, j) != 0)
14                     price += w[i][j];
15             int ret = price + call(Set(mask, i));
16             ans = min(ans, ret);
17         }
18     }
19     return dp[mask] = ans;
20 }
```

$j$  এর লুপটা দিয়ে আমরা মোট দাম বের করে নিলাম। এখন  $i$  তম জিনিসটি কিনলে পরবর্তি স্টেট কি হবে? শুধু  $mask$  এর  $i$  তম বিটটি 1 করে দিতে হবে। আমরা  $call(Set(mask,i))$  এভাবে  $i$  তম জিনিস কিনে পরবর্তি স্টেটে চলে গেলাম। এভাবে প্রতিটি জিনিস কিনে যেটায় দাম মিনিমাম হয় সেটা বিটার্ন করে দিলাম। কাজ শেষ! সম্পূর্ণ কোড:

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্রো-১

ম্যাঞ্চিমাম ফ্রো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভেস্ট সাম, কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

```

1 int w[20][20];
2 int n;
3 int dp[(1 << 15) + 2];
4 int call(int mask)
5 {
6     if (mask == (1 << n) - 1)
7         return 0;
8     if (dp[mask] != -1)
9         return dp[mask];
10    int mn = 1 << 28;
11    for (int i = 0; i < n; i++) {
12        if (check(mask, i) == 0) {
13            int price = w[i][i];
14            for (int j = 0; j < n; j++) {
15                if (i != j and check(mask, j) != 0) {
16                    price += w[i][j];
17                }
18            }
19            int ret = price + call(Set(mask, i));
20            mn = min(mn, ret);
21        }
22    }
23    return dp[mask] = mn;
24 }
25
26 int main()
27 {
28     mem(dp, -1);
29     cin >> n;
30     for (int i = 0; i < n; i++) {
31         for (int j = 0; j < n; j++) {
32             scanf("%d", &w[i][j]);
33         }
34     }
35
36     int ret = call(0);
37     printf("%d\n", ret);
38
39     return 0;
40 }

```

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাষ্ঠার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

আমাদের call ফাংশনটি ক্যাটি ডিন স্টেটে থাকতে পারে?  $n$  টি বিটের প্রতিটি হয় ০ হবে নাহয় ১ হবে, তাহলে স্টেট থাকতে পারে  $2^1 5$  টি। আর ভিতরে একটা  $n^2$  লুপ চলছে তাই মোট complexity  $(2^n) * (n^2)$ ।

বিটমাস্ক ডিপি চেনার সবথেকে সহজ উপায়  $n$  এর মান দেখা।  $n$  এর মান ১৬ বা তার কম হলে খুব ভালো সম্ভাবনা আছে যে প্রবলেমটিকে বিটমাস্ক ডিপি দিয়ে সলভ করা যাবে।

**বিটমাস্ক ডিপি কখন ব্যবহার করবো?** বিটমাস্ক লাগবে আমাদের তখনই যখন আগের স্টেটে কোন কোন জিনিস/ডিজিট/নোড ইত্যাদি ব্যবহার করা হয়েছে সে তথ্যটি আমার বর্তমান স্টেটে লাগবে। সেই তথ্য অনুযায়ী আমরা বর্তমান স্টেট থেকে নতুন ডিজিট/নোড ইত্যাদি নিবো এবং সেই বিটটি অন করে দিয়ে সামনের স্টেটে যাবো। যখন  $n$  টি বিট অন হয়ে যাবে তখন বেসকেস রিটার্ন করে দিবো।

আমার সলভ করা প্রথম বিটমাস্ক ডিপি হলো [uva 10651](#)। আশা করি প্রবলেমটি এখন সহজেই করতে পারবে। প্রবলেমটায় একটি মাস্কের সাহায্যে কোনো সময় বোর্ডের কি অবস্থা সেই তথ্যটা রাখবে, এবং সে অবস্থায় যতগুলো চাল দেয়া সম্ভব সবগুলো দিয়ে মিনিমামটা রিটার্ন করে দিবে।

আপাতত এগুলোই ছিলো বিটমাস্ক ডিপির বেসিক। সামনের পর্বগুলোতে আরো বিস্তারিত আলোচনার চেষ্টা করবো। এখন নিচের প্রবলেমগুলো সলভ করার চেষ্টা করো, ১ম প্রবলেমটি নিয়ে এই পর্বে আলোচনা করেছি:

[Pimp My Ride](#)

[False Mirror](#)

[Agent 47](#)

[Painful Bases](#)

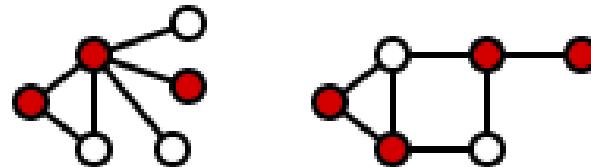
[সবগুলো পর্ব](#)

[ফেসবুকে মন্তব্য](#)

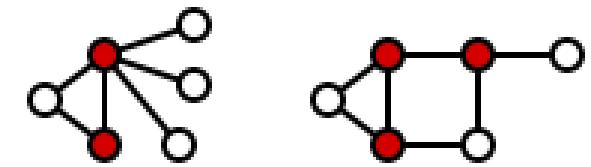
0 comments

# মিনিমাম ভারটেক্স কভার প্রবলেম

অক্টোবর ৪, ২০১২ by শাফায়েত



মিনিমাম ভারটেক্স কভার একটি ক্লাসিক গ্রাফ প্রবলেম। ধৰা যাক একটি শহরে কিছু রাস্তা আছে, এখন প্রতি রাস্তায় মোড়ে আমরা পাহারাদার বসাতে চাই। কোনো নোডে পাহারাদার বসালে সে নোডের সাথে যুক্ত রাস্তাগুলো একাই পাহারা দিতে পারে। উপরের ছবিতে নোডগুলো হলো রাস্তার মোড়। এখন সব কয়টা রাস্তা পাহারা দিতে নৃন্যতম কয়জন পাহারাদার দরকার? ছবিতে লাল নোডগুলোতে পাহারাদার বসানো হয়েছে। এটা অপটিমাল না, নিচের ছবির মত বসালে পাহারাদার কম লাগত:



এটি একটি NP-hard প্রবলেম, অর্থাৎ এই প্রবলেমের কোনো পলিনমিয়াল টাইম সলিউশন নেই। তবে গ্রাফটি যদি **Tree** হয় অর্থাৎ  $n-1$  edge থাকে আব কোনো সাইকেল না থাকে তাহলে **ডাইনামিক প্রোগ্রামিং** বা ম্যাক্স ফ্লো/বাইপারটাইট ম্যাচিং এর সাহায্যে প্রবলেমটি সলভ করা সম্ভব।

## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

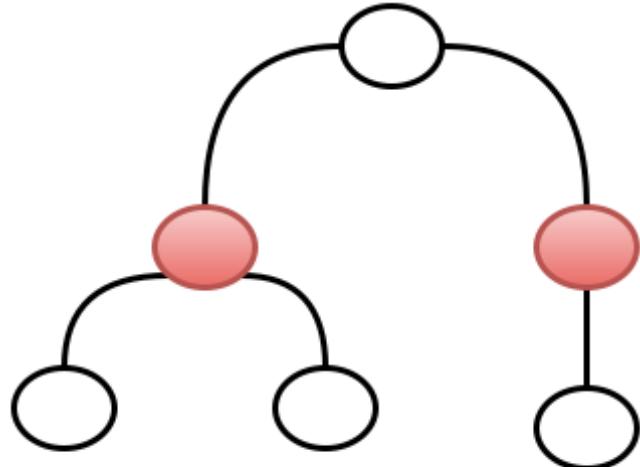
শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)

অ্যাপেলের বিজ্ঞান সিরিজ

গ্রাফ অ্যালগরিদম

শাফায়েত আশুরাফ

d
বিজ্ঞান প্রকাশনা



Minimum Vertex Cover in a Tree

ডাইনামিক প্রোগ্রামিং সলিউশনটা আমি বিশ্লেষিত লিখছি, তারপর ম্যান্ড্রো/বাইপারটাইট ম্যাচিং দিয়ে কিভাবে করতে হয় লিখবো।

ডিপি সলিউশনে ২টি কেস আমাদের লক্ষ্য করতে হবে:

- “ ১. কোনো নোডে পাহাদার না বসালে তার সাথে সংযুক্ত সব নোডে অবশ্যই পাহাদার বসাতে হবে, এছাড়া সব বাস্তা কভার হবে না। অর্থাৎ যদি  $v$  আর  $v'$  সংযুক্ত থাকে তাহলে  $v$  তে পাহাদার না বসালে  $v'$  তে অবশ্যই বসাতে হবে।
- ২. কোনো নোডে পাহাদার বসালে সংযুক্ত নোডগুলোতে পাহাদার বাসানো বাধ্যতামূলক না তবে বসালে লাভ হতে পারে। তাই  $v$  তে পাহাদার বসালে  $v'$  তে পাহাদার একবার বসিয়ে এবং একবার না বসিয়ে দেখবো কোনটা লাভজনক



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

ডাটা স্ট্রাকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

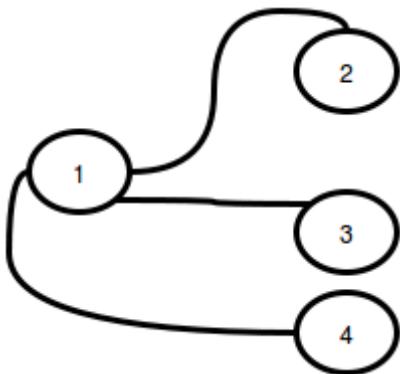
কিউ+সার্কুলার কিউ

সব ডিপির প্রবলেমের মতো এখানেও একটা রিকার্সিভ ফাংশন ডিফাইন করবো। আমাদের স্টেট হবে বর্তমানে কোন নোডে আছি, এবং সেই নোডে কোনো পাহারাদার বসানো হয়েছে নাকি।

“  $F(u,1) = \text{বর্তমানে } u \text{ নম্বর নোডে আছি এবং এই নোডে \text{পাহারাদার আছে।}$   $f(u,1)$  রিটার্ন করবে বাকি নোডগুলোতে মোট পাহারাদার সংখ্যা।

$F(u,0) = \text{বর্তমানে } u \text{ নম্বর নোডে আছি এবং এই নোডে \text{পাহারাদার নাই।}$   $f(u,0)$  রিটার্ন করবে বাকি নোডগুলোতে মোট পাহারাদার সংখ্যা।

ধরি 1 নম্বর নোডের সাথে 2, 3, 4 নম্বর নোড যুক্ত।



বুঝাই যাচ্ছে 1 নম্বর নোডে পাহারা না বসালে অবশ্যই 2, 3, 4 সবগুলোয় পাহারা বসাতে হবে। তাহলে আমরা বলতে পারি:

“  $F(1,0)=F(2,1)+F(3,1)+F(4,1) + 0$ , অর্থাৎ 1 এর সাথে সংযুক্ত সব নোডগুলোতে পাহারা বসালে প্রয়োজনীয় মোট পাহারাদার সংখ্যা।

সবশেষে 0 যোগ করছি কারণ বর্তমান নোডে পাহারাদার বসাইনি।

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

এবাব  $F(1,1)$  এর মান বের কৰি। ১ নম্বৰ নোডে পাহারা বসালে সংযুক্ত নোডগুলোতে পাহারা বসালেও চলে, না বসালেও চলে, তবে যেটা অপটিমাল রেজাল্ট দেয় সেটা আমরা নিব:

$$F(1,1) = 1 + \min(F(2,1), F(2,0)) + \min(F(3,1), F(3,0)) + \min(F(4,1), F(4,0))$$

১ নম্বৰ নোডে পাহারাদার বসাচ্ছি তাই সবশেষে ১ যোগ হচ্ছে, প্রতি নোডে একবাব পাহারা বসিয়ে, আবাব না বসিয়ে দেখছি কোনটা অপটিমাল।

একটা ব্যাপার লক্ষ রাখতে হবে যে প্যারেন্ট নোড নিয়ে কখনো হিসাব করবোনা। উপরের ছবিতে ১ থেকে ২ এ গেলে  $\text{parent}[2]=1$ , তাই ২ থেকে আবাব ১ নম্বৰ নোডে যাবোনা।

এবাব base case এ আসি। কোনো নোড থেকে নতুন কোনো নোডে যাওয়া না গেলে ১ বা ০ রিটার্ন কৰে দিতে হবে, পাহারাদার বসালে ১, না বসালে ০। কোনো ট্রি তে একটি মাত্র নোড থাকলে ১ রিটার্ন কৰতে হবে(কিছু প্রবলেমে ০ ও রিটার্ন কৰতে হতে পাৰে)।

Spoj এর [PT07X\(vertex cover\)](#) প্রবলেমটি straight forward প্রবলেম। এটার জন্য আমাৰ কোডটা এৰকম:

```

1 #define MAXN 100002
2 int dp[MAXN][5];
3 int par[MAXN];
4 vectoredges[MAXN];
5
6 int f(int u, int isGuard)
7 {
8     if (edges[u].size() == 0)
9         return 0;
10    if (dp[u][isGuard] != -1)
11        return dp[u][isGuard];
12    int sum = 0;
13    for (int i = 0; i < (int)edges[u].size(); i++) {
14        int v = edges[u][i];
15        if (v != par[u]) {
16            par[v] = u;
17            if (isGuard == 0)
18                sum += f(v, 1);
19            else
20                sum += min(f(v, 1), f(v, 0));
21    }
22    dp[u][isGuard] = sum;
23    return sum;
24}
```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লাৱ ট্ৰ্যুৱ মিনিমাম ডারটেক্স কভাৱ

ট্ৰি এৰ ডায়ামিটাৱ নিৰ্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগৱিদম গেম থিওৰি(৩):

গেম থিওৰি-১

গেম থিওৰি-২

গেম থিওৰি-৩

ডাইনামিক প্ৰোগ্ৰামিং(৮):

ডাইনামিক প্ৰোগ্ৰামিং ১ (ফিবোনাচি)

ডাইনামিক প্ৰোগ্ৰামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্ৰোগ্ৰামিং ৩ (LIS এবং পাথ প্ৰিস্টিং)

ডাইনামিক প্ৰোগ্ৰামিং ৪ (লংগেস্ট কমন সাবসিকোয়েম)

ডাইনামিক প্ৰোগ্ৰামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্ৰোগ্ৰামিং ৬ (সাবসেট সাম, কম্পিনেটোৱিঙ্গ, ডিসিশন প্রবলেম)

```

21     }
22 }
23     return dp[u][isGuard] = sum + isGuard;
24 }
25
26 int main()
27 {
28     memset(dp, -1, sizeof(dp));
29     int n;
30     scanf("%d", &n);
31     for (int i = 1; i < n; i++) {
32         int u, v;
33         scanf("%d%d", &u, &v);
34         edges[u].push_back(v);
35         edges[v].push_back(u);
36     }
37     int ans = 0;
38     ans = min(f(1, 1), f(1, 0));
39     printf("%d\n", ans);
40     return 0;
41 }

```

আমি ট্রি এর root সবসময় ১ ধরে কোড লিখেছি। ৩৮ নম্বর লাইনে মেইন ফাংশনে root এ পাহারাদার একবার বসিয়ে আর একবার না বসিয়ে অপটিমাল বেজাল্ট টা নিছি।

ফাংশনে u হলো current node, isguard কারেন্ট নোডে পাহারাদার আছে নাকি নাই সেটা নির্দেশ করে।

১০ নম্বর লাইনে ট্রি এর সাইজ ১ হলে ১ রিটার্ন করে দিয়েছি।

১৩ নম্বর লাইনে লুপের ভিতর current নোড থেকে সবগুলো child নোডে যাচ্ছি। কারেন্ট নোডে পাহারাদার না থাকলে পরেরটায় বসাচ্ছি, আর থাকলে ২ভাবেই চেষ্টা করছি। ১৫ নম্বর লাইনের কভিশন দিয়ে প্যারেট নোডে যেতে দিচ্ছিনা।

সবশেষে sum+isGuard রিটার্ন করছি। অর্থাৎ কারেন্ট নোডে পাহারাদার থাকলে 1 যোগ করছি, নাহল 0।

মোটামুটি এই হলো ডিপি সলিউশন। ট্রি তে সাইকেল না থাকায় এটা অবশ্যই বাইপারটাইট গ্রাফ। ১৯৩১ সালে Dénes König প্রমাণ করেন কোনো বাইপারটাইট গ্রাফে maximum matching=minimum vertex cover। এটা গ্রাফ থিওরির অনেক min-max থিওরেমের একটা যেখানে কিছু একটা ম্যাক্সিমাইজ করলে অন্য আরেকটা কিছু মিনিমাইজ হয়। তুমি যদি ম্যাক্সিমাম ম্যাচিং এর অ্যালগোরিদম জানো তাহলে

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাকিং(১):

ব্যকট্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

টি টা বাইকালারিং করে ম্যাচিং বের করলেই ভারটেক্স কভার বের হয়ে যাবে। কোড সহজ হলেও complexity বেড়ে যাবে, তাই নোড বেশি থাকলে কাজ করবেনা। আবার ম্যাক্সিমাম ম্যাচিং যেহেতু ম্যাক্স-ফ্লো এর একটি ড্যারিয়েশন তাই ফ্লো চালিয়েও সমাধান করা সম্ভব।

এরকম আরেকটা প্রবলেম [uva-10243\(fire fire fire\)](#)। আমি প্রথমে এটা সমাধান করে পরে spoj এর টা করেছি।

## ফেসবুকে মন্তব্য

1 comments

# ব্যাকট্র্যাকিং: পারমুটেশন জেনারেটর

সেপ্টেম্বর ২০, ২০১৬ by [শাফায়েত](#)



ব্যাকট্র্যাকিং একধরণের ক্রটফোর্স টেকনিক। ক্রটফোর্সের মতই এটা সম্ভাব্য সবধরণের বিন্যাস-সমাবেশ থেকে ফলাফল খুজে নিয়ে আসে। যেমন ধর তোমাকে ঢাকা থেকে চট্টগ্রামে যাবার সবথেকে ছোটো পথ খুজে বের করতে বলা হলো। তুমি ডায়াল্ট্রার দেয়া অ্যালগরিদম ব্যবহার না করে যাবার যত পথ আছে সবগুলো খুজে বের করলে এবং তারপর তারমধ্যে থেকে সবথেকে ছোট কোনটা সেটা বের করলে, এটা হলো ক্রটফোর্স বা কমপ্লিট সার্চ।

এই লেখাটা পড়ার আগে অবশ্যই রিকার্সন সম্পর্কে ডালো ধারণা থাকতে হবে।

সার্চস্পেসের আকার ছোটো হলে এটা খুবই কার্যকর একটা পদ্ধতি। সার্চস্পেস হলো কতটুকু অংশজুড়ে তোমার সলিউশন থাকতে পারে সেইটুকু। যেমন তোমাকে যদি ১০সাইজের দুটি সেট দিয়ে বের করতে বলে ২য় সেট ১মটির সাবসেট কিনা তাহলে খুব সহজে তুমি ব্যাকট্র্যাক করে  $2^{10} = 1024$  টি সেট বের করে সবগুলোর সাথে মিলিয়ে দেখতে পারো, কিন্তু সেটের আকার ১০০ হলে তোমার এভাবে সলিউশন বের করার জন্য এই জীবনকাল যথেষ্ট নয়!

যেসব প্রবলেমের পলিনমিয়াল কোনো সলিউশন আমরা এখনো জানিনা অর্থাৎ **NP বা NP-hard ক্যাটাগরির** প্রবলেম সেগুলোকে ব্যাকট্র্যাক করেই সমাধান করতে হয়। ব্যাকট্র্যাকিং কোড লেখার আগে **কমপ্লেক্সিটির** ব্যাপারে খুব সতর্ক থাকতে হবে।

এখন আমরা দেখবো কিভাবে ব্যাকট্র্যাক করে ০ থেকে  $n - 1$  পর্যন্ত সংখ্যাগুলোর প্রতিটি পারমুটেশন বের করা যায়। যদি  $n = 2$  হয় তাহলে পারমুটেশনগুলো হবে:

## সাবক্ষাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)

অ্যাপ্লিকেশন বিজ্ঞান সিরিজ

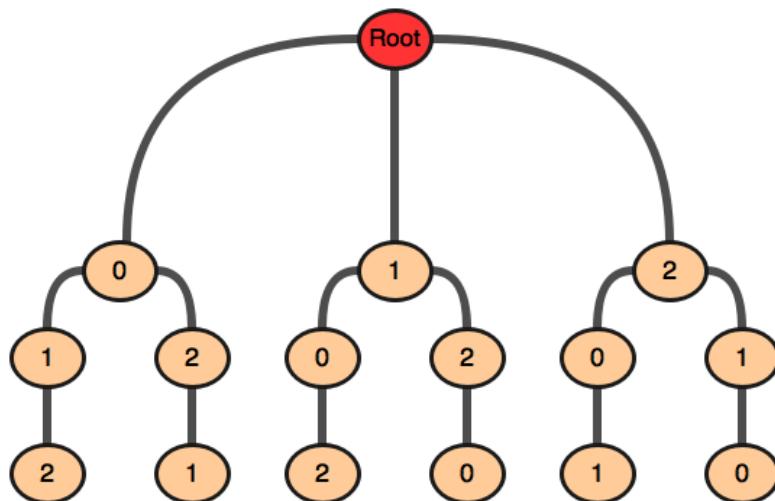
গ্রাফ অ্যালগরিদম

শাফায়েত আশরাফ

d  
বিশিষ্ট প্রকাশনা

Position	0	1	2
Permutations	0	1	2
	0	2	1
	1	0	2
	1	2	0
	2	0	1
	2	1	0

সহজে বোঝার জন্য আমরা পারমুটেশনগুলোকে একটা ট্রি আকারে দেখি:



এই ট্রি তে রুট থেকে যেকোনো পথে আগতে থাকলে একটা করে পারমুটেশন পাওয়া যাবে। আমরা একটা রিকার্সিভ ফাংশন লিখবো যেটা এই ট্রি এর সবগুলো পথে একবার করে ঘুরে আসবে। আমদেরকে সত্যি সত্যি অ্যাডজেসেন্সি ম্যাট্রিক্স দিয়ে ট্রি বানিয়ে ফেলা দরকার নেই, ছবিটা দেয়া হয়েছে সহজে বোঝার জন্য।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে  
হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

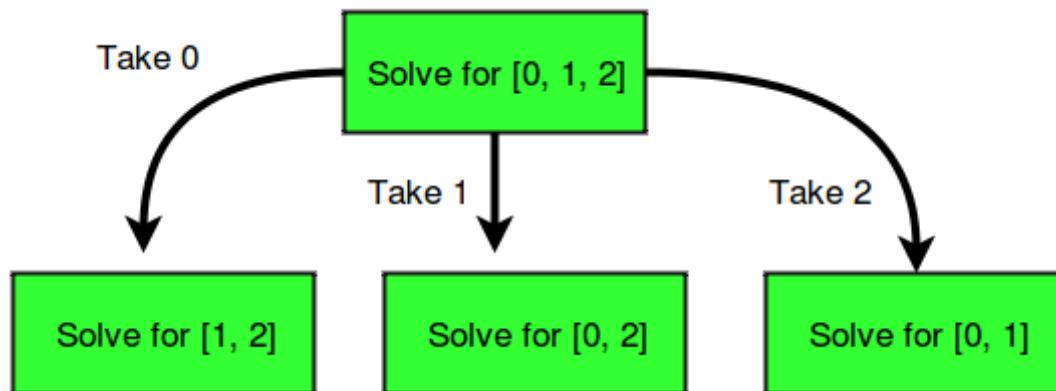
কিউ+সার্কুলার কিউ

মনে করো আমাদের ফাংশনের নাম হলো *generate(idx)*। *idx* মানে হলো এখন আমরা *idx* তম পজিশনে একটা সংখ্যা বসাতে চাই।  
নিচের সুড়োকোড়টি দেখো, এরপর আমি ব্যাখ্যা করছি:

```
1 procedure generate(idx)
2     if idx == n
3         print position
4         return
5     for i from 0 to n
6         if taken[i] = False
7             taken[i] <- true
8             position[idx] = i
9             generate(idx+1)
10            taken[i] <- false
```

৫ নম্বর লাইনে আমি ০ থেকে *n* পর্যন্ত একটি লুপ চালিয়েছি। ৬ নম্বর লাইনে দেখছি যে *i* সংখ্যাটি এবই মধ্যে নেয়া হয়েছে নাকি। যদি না নেয়া হয়ে  
থাকে তাহলে *idx* তম পজিশনে *i* বসিয়ে রিকার্সিভলি *idx + 1* বাকি পজিশনগুলোর জন্য প্রবলেমটা সলভ করছি।

ইটারেস্টিং ব্যাপার হলো রিকার্সিভলি *generate(idx + 1)* এর জন্য সলভ করার পর আমরা *taken[i] <- false* করে দিচ্ছি। কারণ *i*  
তম সংখ্যাটা *idx* পজিশনে বসানোর পর আবার সেই পজিশন থেকে *i* কে ফেলে দিয়ে *i + 1* কে একই পজিশনে বসিয়ে রিকার্সিভলি সলভ  
করবো।



ব্যাকট্র্যাক্টিং করার জেনারেল আইডিয়াটা হলো:

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

- “
১. প্রতিটি ফাংশন কলে সম্ভাব্য অপশনগুলোর একটি বাছাই করো।
  ২. বাকি অপশনগুলো থেকে রিকার্সিভলি সমাধান বের করার চেষ্টা করো।
  ৩. বাছাই করা অপশনটি ফেলে দিয়ে অন্য আরেকটি নিয়ে আবার চেষ্টা করো।

এখন তোমার কাজ হবে সুড়েকোডটাকে আসল কোডে রূপান্তর করা। যদি এটা পারো তাহলে তুমি আরো কিছু একইরকম সমস্যা সহজেই সমাধান করতে পারবে। যেমন তোমাকে যদি একটা স্ট্রিং "abcd" দিয়ে বলা হয় সবরকম পারমুটেশন জেনারেট করতে তাহলে একইভাবে সহজেই করতে পারবে। কিন্তু স্ট্রিংটায় যদি এই ক্যারেক্টার একাধিকবার থাকে (যেমন "abbcdd") তাহলে একটু ঝামেলায় পড়বে, দেখবে একই পারমুটেশন বারবার জেনারেট হচ্ছে। এটা এড়াতে তোমাকে একটা ফ্ল্যাগ রেখে একই পজিশনে একই ক্যারেক্টার একাধিকবার বসিয়ে রিকার্সিভলি সলভ করা বন্ধ করতে হবে।

প্র্যাকটিসের জন্য নিচের সমস্যাগুলো সমাধান করো:

[Determine The combination](#)

[Prime Ring problem](#)

[House of santa clause](#)

[All Walks of length n](#)

[Following orders](#)

আপাতত এখানেই শেষ, পরের পর্বে ব্যাকট্র্যাকিং দিয়ে সমাধান করা যায় এমন কিছু সমস্যা দেখবো।

[ফেসবুকে মন্তব্য](#)

0 comments

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রিংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

[অ্যালগরিদম গেম থিওরি\(৩\):](#)

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

[ডাইনামিক প্রোগ্রামিং\(৮\):](#)

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভেস্ট সাম, কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

# মডুলার অ্যারিথমেটিক

জানুয়ারি ২০, ২০১২ by শাফায়েত



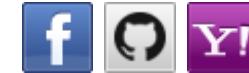
— ১৭ কে ৫ দিয়ে ভাগ করলে ভাগশেষ কত হয়?  $2^{1000}$  কে ১৭ দিয়ে ভাগ করলে ভাগশেষ কত হয় সেটা কি তুমি ওভারফ্লো এডিয়ে নির্ণয় করতে পারবে?  $O(n)$  এ পারলে  $O(\log_2 n)$  কমপ্লেক্সিটিতে পারবে? যদি কোনো একটি উত্তর “না” হয় তাহলে এই পোস্ট তোমার জন্য। তবে তুমি যদি মডুলার ইনভার্স বা এডভাঞ্চড কিছু শিখতে পোষ্টটি খুলো তাহলে তোমাকে আপাতত হতাশ করতে হচ্ছে।

সি/জাভা সহ বেশিভাগ প্রোগ্রামিং ল্যাংগুয়েজে এ % কে ভাগশেষ অপারেটর ধরা হয়।  $x$  কে  $m$  দিয়ে ভাগকরে ভাগশেষ বের করার অর্থ  $x \% m$  এর মান বের করা অথবা আমরা বলতে পারি  $x$  কে  $m$  দিয়ে mod করা। “determine answer modulo 1000” এ কথাটির অর্থ হলো উত্তরকে 1000 দিয়ে mod করে তারপর আউটপুট দিতে হবে।

একটি সমস্যা দিয়ে শুরু করি। তোমার 100টি বই আছে, তুমি কয়ভাবে বইগুলো সাজাতে পারবে? খুব সহজ, 100! (100 ফ্যাক্টরিয়াল) ডাবে সাজাতে পারবে। 100! ১৫৮ ডিজিটের বিশাল একটি সংখ্যা। তাই আমি তোমাকে প্রবলেমটা সহজ করে দিলাম, ধরো তুমি  $x$  উপায়ে বইগুলো সাজাতে পারবে, তাহলে তোমাকে  $x$  কে সেটা বলতে হবে। অর্থাৎ 100! বের করে ৯৭ দিয়ে ভাগ করে ভাগশেষটা বের করাই তোমার সমস্যা। (Determine 100 factorial modulo 97)

এটা কিভাবে করবে? 100! এর মান তুমি বের করতে পারবেনা ৬৪বিট আনসাইনড ইন্টিজার দিয়েও, এবা  $2^{64}$  — ১ পর্যন্ত সংখ্যা নিয়ে কাজ করতে পারে, তাই ওভারফ্লো হবে। কিন্তু আমরা জানি আমাদের উত্তর কখনোই 97 এর বড় হবেনা কারণ কোনো সংখ্যাকে  $m$  দিয়ে mod করা হলে সংখ্যাটি  $m$  এর থেকে বড় হতে পারবেনা।

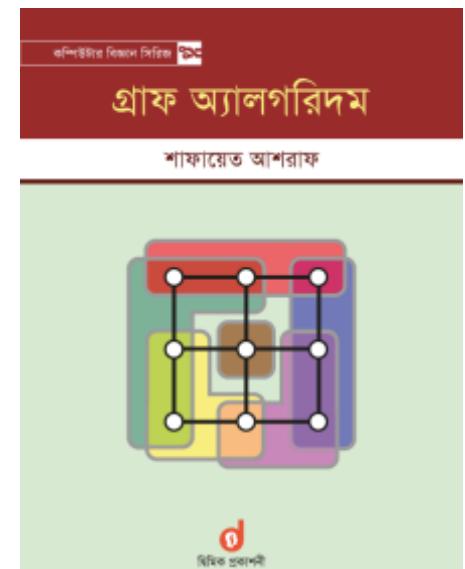
## সাবস্ক্রাইব



Secured by OneAll Social Login

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



আমরা এ ধরণের সমস্যা সমাধান করতে সাহায্য নিবো দুটি সুত্রে:

$$(a + b)\%m = ((a\%m) + (b\%m))\%m$$
$$(a * b)\%m = ((a\%m) * (b\%m))\%m$$

$n$  সংখ্যক নম্বর  $a_1, a_2 \dots a_n$  এর জন্য সুত্র দুটি ব্যবহার করতে পারবে।

উপরের সমস্যাটিতে ২য় সুত্রটি লাগবে। তোমার বের করা দরকার  $100! \bmod 97$  অর্থাত:

$$(100 * 99 * 98 * \dots * 1) \bmod 97$$

তুমি যেটা করবে সেটা হলো গুণ করার সময় ২য় সুত্রের মত করে mod করতে থাকবে, তাহলে কোনো সময়ই overflow ঘটবেনা কারণ mod করলে প্রতি স্টেপে সংখ্যাটি ছোটো হয়ে যাচ্ছে। এটার কোড হতে পারে এরকম:

```
1 int fact=1;
2     for(int i=1;i<=100;i++)
3     {
4         fact=((fact%97)*(i%97))%97;
5     }
6 }
```

printf("%d\n",fact);

এটার আউটপুট আসবে ০। অর্থাত  $100! \bmod 97 = 0$ । একটু খেয়াল করলেই বুঝবে এখানে আমরা ২য় সুত্রটি প্রয়োগ করেছি ২টি করে সংখ্যা নিয়ে।

সুত্র দুটি কেনো কাজ করে সেটা জানা দরকার। আমি ১ম সুত্রটির প্রমাণ দেখাচ্ছি, ২য়টিও একইভাবে করা যায়। প্রমানটি আমার নিজের মত করে করা।

ধরি  $(x+y)\%5$  এর মান আমাদের বের করতে হবে। এখন যদি  $x\%5 = c_1$  আর  $y\%5 = c_2$  হয়, তাহলে  $x$  কে আমরা লিখতে পারি  $5n_1 + c_1$  এবং  $y$  কে লিখতে পারি  $5n_2 + c_2$  যেখানে  $n_1$  আর  $n_2$  দুটি ইন্টিজার। এটা একদম বেসিক রুল, আশা করে বুঝতে সমস্যা হচ্ছেনা। এখন:



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে  
হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইনেক্সেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

$$\begin{aligned}
 & (x + y)\%5 \\
 &= (5n_1 + c_1 + 5n_2 + c_2)\%5 \\
 &= (5n_1 + 5n_2 + c_1 + c_2)\%5 \quad \text{--(১)}
 \end{aligned}$$

এখনে  $5n_1 + 5n_2$  অবশ্যই 5 এর মাল্টিপল, তাই আমরা লিখতে পারি

$$5n_1 + 5n_2 = 5N \text{ যখন } N = n_1 + n_2$$

$$\text{এবং } c_1 + c_2 = C$$

তাহলে (১) থেকে পাচ্ছি:

$$(5N + C)\%5$$

এখন পরিষ্কার বোধ যাচ্ছে যে উত্তর হলো  $C\%5$ ।  $C$  কে আবার mod করতে হলো কারণ  $c_1 + c_2$  এর মান 5 এর থেকে বড় হতেই পারে।

এখন

$$\begin{aligned}
 & ((x\%5) + (y\%5))\%5 \quad \text{--(২)} \\
 &= ((5n_1 + c_1)\%5) + ((5n_2 + c_2)\%5)\%5 \\
 (5n_1 + c_1)\%5 &= c_1 \\
 (5n_2 + c_2)\%5 &= c_2
 \end{aligned}$$

তাহলে ২ কে লিখতে পারি:

$$(c_1 + c_2)\%5 = C\%5$$

তাহলে ১ম সুত্রটি প্রমাণিত হলো। তারমান যোগ করে mod করা আবার আগে mod করে তারপর যোগ করে আবার mod করা একই কথা। সুবিধা হলে সংখ্যাটি কোনো স্টেপেই বেশি বড় হতে পারেনা। গুণের ক্ষেত্রেই একই সুত্র প্রযোজ্য।

স্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

নেগেটিভ সংখ্যার mod নিয়ে একটু আলাদা ভাবে কাজ করতে হয়। সি তে  $-17\%5$  এর মান দেখায় -2। কিন্তু সচরাচর আমরা ভাগশেষের যে সংজ্ঞা ব্যবহার করি তাতে  $x\%m = p$  হলে গাণিতিকভাবে

“  $m$  এর সবথেকে বড় থেকে বড় মাল্টিপল যেটা  $x$  এর থেকে ছোট সেই সংখ্যাটিকে  $x$  থেকে বিয়োগ করলে যে সংখ্যাটি পাওয়া যায় সেটাই  $p$ ।

যেমন  $23\%5$  এর ক্ষেত্রে  $5 \times 8 = 20$  হলো 5 এর সবথেকে বড় মাল্টিপল যেটা 20 এর থেকে ছোট, তাই

$23\%5 = 23 - (5 \times 4) = 3$ ।  $-17\%5$  এর ক্ষেত্র খেয়াল করো  $-20$  হলো 5 এর সবথেকে বড় মাল্টিপল যেটা  $-17$  থেকে ছোট, তাই উত্তর হবে 3।

এই কেসটা handle করা একটি উপায় হলো নেগেটিভ সংখ্যাটিকে একটি 5 এর মাল্টিপল এর সাথে যোগ করা যেন সংখ্যাটি 0 থেকে বড় হয়ে যায়, তারপরে mod করা। যেমন:

$$\begin{aligned} & -17\%5 \\ & = (-17 + 100)\%5 \\ & = 83\%5 \\ & = 3 \end{aligned}$$

এটা উপরের সুত্রের প্রমাণের মত করেই কাজ করে, একটু গুতালেই প্রমাণ করতে পারবে। **নেগেটিভ সংখ্যার mod নিয়ে কনটেন্ট সবসময় সতর্ক থাকবে, এটা wrong answer খাওয়ার একটা বড় কারণ হতে পারে।**

এবার আসি সুপরিচিত big mod সমস্যায়। সমস্যাটি হলো তোমাকে  $(a^b)\%m$  এর মান বের করতে হবে,  $a, b, m$  তোমাকে বলে দেয়া হবে, সবগুলোর range  $2^{31}$  পর্যন্ত হতে পারে।  $100! \bmod 97$  বের করার মত করে সহজেই তুমি overflow না খেয়ে মানটি বের করতে পারবে, সমস্যা হলো তুমি লুপ চালিয়ে একটি একটি গুণ করে  $2^{2000000000}$  বের করতে চাইলে উত্তর পেতে পেতে সম্ভবত নাশ্তা শেষ করে

আর্টিকুলেশন পয়েন্ট এবং বিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্সিমাম ফ্লো-1

ম্যাক্সিমাম ফ্লো-2

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/0-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্সেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

আসতে পারবে। আমরা চাইলে  $O(\log_2 n)$  এ এটা করতে পারি।

লক্ষ করো

$$“ 2^{100}$$

$$=(2^{50})^2$$

এবং

$$(2^{50})$$

$$=(2^{25})^2$$

এখন বলো  $2^{50}$  বের করতে কি  $2^{26}, 2^{27}$  ইত্যদি বের করার দরকার আছে নাকি  $2^{25}$  পর্যন্ত বের করে square করে দিলেই হচ্ছ? আবার  $2^{25}$  পর্যন্ত আসতে  $(2^{12})^2$  পর্যন্ত বের করে square করে সাথে ২ গুণ করে দিলেই যথেষ্ট, অতিরিক্ত ২ গুণ করছি সংখ্যাটি বিজোড় সে কারণে। প্রতি স্টেপে গুণ করার সময় mod করতে থাকবে যাতে overflow না হয়। recursion ব্যবহার করে কোডটি লেখা জলের মত সোজা:

```
1 #define i64 long long
2 i64 M;
3 i64 F(i64 N,i64 P)
4 {
5     if(P==0) return 1;
6     if(P%2==0)
7     {
8         i64 ret=F(N,P/2);
9         return ((ret%M)*(ret%M))%M;
10    }
11    else return ((N%M)*(F(N,P-1)%M))%M;
12
13 }
```

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

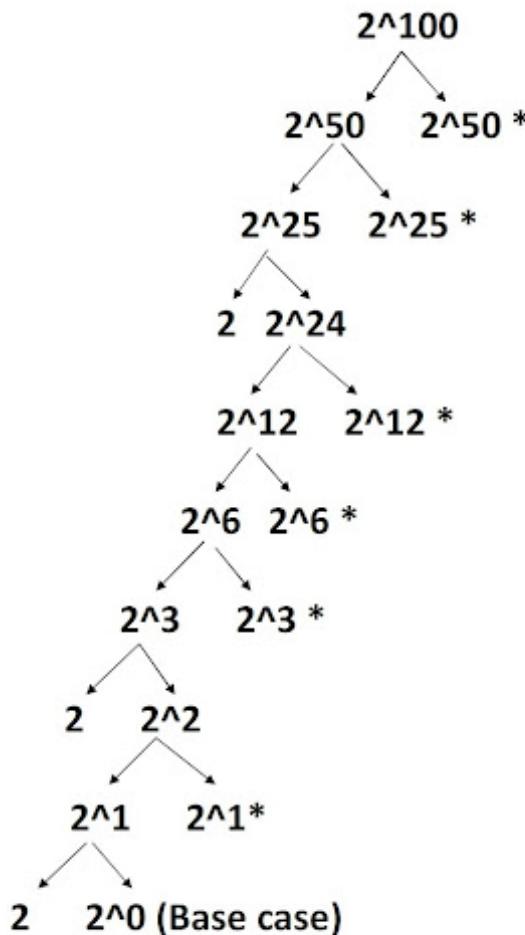
ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

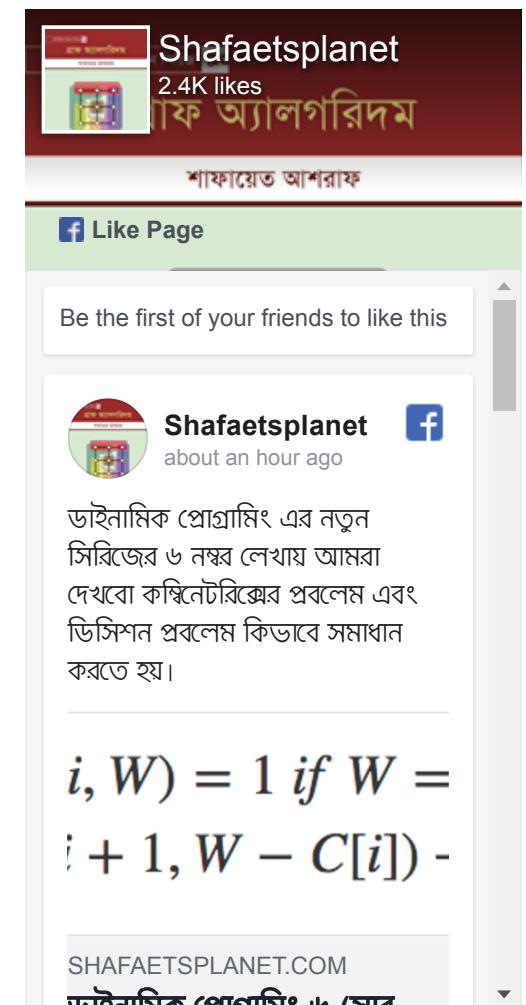
মন্তব্য অংশে “হাসান” একটি বিগ মডের সুন্দর রিকার্চন-ট্রি এর ছবির লিংক দিয়েছে, ছবিটা এরকম:



মডুলার অ্যারিথমেটিক ব্যবহার করে বিশাল আকারের ফলাফল কে আমরা ছেট করে আনতে পারি ফলাফলে বিভিন্ন প্রোপার্টিকে নষ্ট না করে, তাই এটা গণিতে খুব গুরুত্বপূর্ণ। প্রোগ্রামিং কনটেন্টে প্রায়ই বিভিন্ন প্রবলেমে মডুলার অ্যারিথমেটিক প্রয়োজন পড়বে, বিশেষ করে counting আর combinatorics এ যেখানে ফলাফল অনেক বড় হতে পারে, ফ্যাটেরিয়াল নিয়ে কাজ করতে হতে পারে।

কোয়ান্টাম কম্পিউটার কী?

কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা



ভাগ করার সময় গুণ, আর যোগের মত সূত্র দুটি কাজ করেনা, এটার জন্য তোমাকে extended euclid আর modular inverse জানতে হবে।

সিপিউর জন্য mod খুব costly একটা অপারেশন। যোগ, গুণের থেকে mod করতে অনেক বেশি সময় লাগে। অপ্রয়োজনে mod ব্যবহার করলে কোড time limit exceed করতে পারে, তাই overflow হবার আশংকা না থাকলে সব জায়গায় mod করা দরকার নেই। আমার একটি কোড তিসেকেন্ডে time limit exceed হবার পর খালি কিছু mod সরিয়ে ১.৩ সেকেন্ড নামিয়ে এনেছি।

এখন চিন্তা করার জন্য একটি প্রবলেম। ধরো তোমাকে একটি অনেক বড় সংখ্যা(bigint) দিয়ে সেটাকে  $2^{31}$  এর ছোট একটি সংখ্যা দিয়ে mod করতে বলা হলো।  $O(\text{length_of_bigint})$  কমপ্লেক্সিটিতে কিভাবে করবে?

সাহায্য:

$$23 = (0 * 10 + 2) * 10 + 3$$

$$1239 = (((0 * 10 + 1) * 10 + 2) * 10 + 3) * 10 + 9$$

প্র্যাকটিসের জন্য প্রবলেম:

<http://uva.onlinejudge.org/external/3/374.html>

<http://uva.onlinejudge.org/external/101/10127.html>

ফেসবুকে মন্তব্য

3 comments

# প্রাইম জেনারেটর (Sieve of Eratosthenes)

 জুলাই ১৭, ২০১১ by শাফায়েত



প্রাচীনকাল থেকেই গণিতবিদরা মাথা ঘামাচ্ছেন প্রাইম নাস্বার বা মৌলিক সংখ্যা নিয়ে। প্রাইম নাস্বারগুলো মধ্যে লুকিয়ে আছে বিষয়কর কিছু সৌন্দর্য। যেকোনো কম্পিউটার বা যৌগিক সংখ্যাকে একাধিক প্রাইমের গুণফল হিসাবে মাত্র একভাবে লেখা যায়, ঠিক যেমন সব যৌগিক পদার্থ একাধিক মৌলিক পদার্থের সমন্বয়ে তৈরি। প্রাচীনকাল থেকেই মানুষ প্রাইম নিয়ে গবেষণা করছে, চলছে এখনো। গাউস, ফার্মা, ইউলারের মত কিংবদন্তি গণিতবিদরা কাজ করেছেন প্রাইম নিয়ে।



দ্রুত গতিতে প্রাইম সংখ্যা বের করার একটি পদ্ধতি আবিষ্কার করেন Eratosthenes, ২০০ খ্রিস্টপূর্বের একজন গ্রীক গণিতবিদ, বিজ্ঞানি ও কবি। ২২০০ বছরেরও পুরানো সেই পদ্ধতি ব্যবহার করে আমরা আধুনিক কম্পিউটারে প্রাইম জেনারেট করি, খুব কম সময়ে বের করা যায় ১০কোটির নিচে সব প্রাইম সংখ্যা। এই অ্যালগরিদমটি sieve of Eratosthenes নামে পরিচিত, প্রোগ্রামিং এর জগতে সুন্দরতম অ্যালগরিদমগুলোর মধ্যে এটি একটি।

sieve এর শাব্দিক অর্থ হলো ছাকনি যা অপ্রয়োজনীয় অংশ ছেটে ফেলে (A sieve, or sifter, separates wanted elements from unwanted material using a woven screen such as a mesh or net)। Eratosthenes এর ছাকনি যৌগিক সংখ্যাগুলোকে ছেটে ফেলে দেয়।

*Sift the Twos and sift the Threes,  
The Sieve of Eratosthenes.*

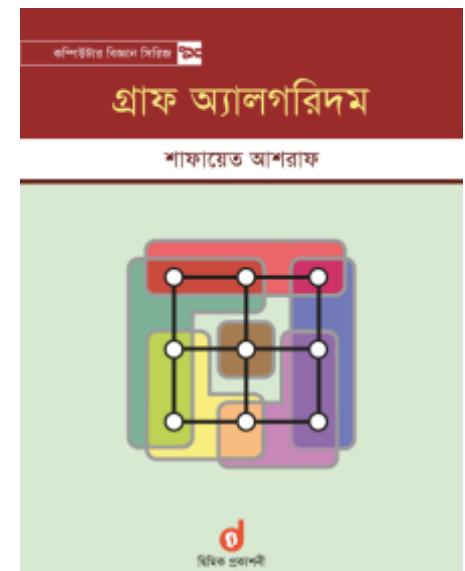
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



When the multiples sublime,  
“  
The numbers that remain are Prime  
(Traditional, collected from wikipedia)

আমরা জানি প্রাইম সংখ্যা হলো সেসব সংখ্যা যাদের ১ এবং সেই সংখ্যটি ব্যতিত কোনো সংখ্যা দিয়ে ভাগ করা যায়না, যেমন ২, ৩, ৫, ৭, ২৯ ইত্যাদি। অন্যভাবে বলা যায় **সেসব সংখ্যাই প্রাইম যাদেরকে সংখ্যাটির বর্গমূলের সমান বা ছেটো কোনো প্রাইম দিয়ে ভাগ করা যায় না।** এই সংজ্ঞাটাই অ্যালগোরিদমের মূল অংশ, তাই আগে আমরা এটা বুঝতে চেষ্টা করব। ফর্মালভাবে প্রমাণ না করে ব্যপারটি বুঝানোর চেষ্টা করি। যেকোনো সংখ্যাকে আমরা কয়েকটি প্রাইমের গুণফল হিসাবে লিখতে পারি যাদের প্রাইম ফ্যাক্টর বলা হয়:

“  $n=p_1*p_2*p_3....*p_k$

$n$  যদি নিজেই প্রাইম হয় তাহলে  $n=p_1 (=n)$ । অন্যথায় অবশ্যই একাধিক প্রাইম ফ্যাক্টর থাকতে হবে। এবার চিন্তা করো কোনো সংখ্যা  $c$  কে দুটি সংখ্যার গুণফল  $c=a*b$  হিসাবে লিখলে  $a$  আর  $b$  এর একটি অবশ্যই সংখ্যাটির বর্গমূলের থেকে ছোট, অন্যটি বড়।  $a, b$  দুটো সংখ্যাই  $c$  এর বর্গমূলের থেকে বড় হলে গুণফল  $c$  থেকে বড় হতো (ঠিক যেমন  $c=a+b$  হলে  $a$  বা  $b$  এর একটি  $c$  এর অর্ধেকের থেকে ছোট অন্যটি বড়)।

এবার  $n=p_1*p_2*p_3....*p_k$  তে ফিরে আসি।  $p_1, p_2, p_3$  ইত্যাদির মধ্যে যে কোনো ২টি যদি  $n$  এর বর্গমূল থেকে বড় হয় তাহলে তাদের গুণফল  $n$  কে ছাড়িয়ে যাবে, তাই নয় কি? সর্বোচ্চ একটি প্রাইম ফ্যাক্টর বর্গমূলের বাইরে যেতে পারে, বাকি গুলো কে অবশ্যই ডিতরে থাকতে হবে।

তাহলে আমরা নিশ্চিত যে যৌগিক সংখ্যা কে তার বর্গমূলের থেকে ছোট কোনো প্রাইম দিয়ে ভাগ করা যাবে। ২য় সংজ্ঞাটি এখন আমাদের কাছে পরিষ্কার: “সেসব সংখ্যাই প্রাইম যাদেরকে সংখ্যাটির বর্গমূলের সমান বা ছেটো কোনো প্রাইম দিয়ে ভাগ করা যায় না”। বুঝতে না পারলে আরেকবার ভালো করে চিন্তা করে নিচের অংশ পড়ো।

এবার আমরা আমাদের ছাকনি চালু করি এবং প্রাইম বের করি। ২৫ এর নিচের সব প্রাইম আমরা বের করব। ২৫ এর বর্গমূল ৫, তাই ২৫ বা তার থেকে ছোট কোন সংখ্যাকে অবশ্যই ৫ বা তার থেকে ছোট কোনো প্রাইম দিয়ে ভাগ করা যাবে।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগোরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

২ একটি প্রাইম কারণ ২কে তার বর্গমূলের নিচে কোনো সংখ্যা দিয়ে ভাগ করা যায়না। তাহলে ২ এর মাল্টিপলগুলো কেও প্রাইম নয় কারণ তাদের ২ দিয়ে ভাগ করা যায়, সেগুলোকে আমরা কেটে দেই:

“ ২, ৪, ৬, ৮, ১০, ১২, ১৪, ১৬, ১৮, ২০, ২২, ২৪

২ এর পরের সংখ্যা ৩। ৩ যদি প্রাইম না হতো তাহলে ৩ এর বর্গমূলের নিচের কোনো প্রাইম ৩ কে বাদ দিয়ে দিত, যেহেতু ৩ বাদ পড়েনি তাই সংজ্ঞানতে ৩ প্রাইম। ৩ এর মাল্টিপল গুলো কে বাদ দেই:

“ ৩, ৬, ৯, ১২, ১৫, ১৮, ২১, ২৪

পরের সংখ্যা ৪। ৪ বাদ পড়ে গিয়েছে আগেই। তারপর আছে ৫। ৫ যদি প্রাইম না হতো তাহলে আগেই ছাকনিতে কাটা পড়ত, ৫ এর মাল্টিপল গুলোকে কেটে দেই:

“ ৫, ১০, ১৫, ২০, ২৫

আমাদের আর কাটাকাটি প্রয়োজন নেই। ২৫ এর বর্গমূল ৫, তাই ২৫এর নিচের সব সংখ্যার বর্গমূল ৫ থেকে ছোট। সুতরাং ২৫ এর নিচের সকল যৌগিক সংখ্যা ৫ বা তার নিচের কোনো প্রাইম দিয়ে বিভাজ্য। যেহেতু আমরা ২, ৩, ৫ এর সব মাল্টিপল কেটে দিয়েছি, বাকি সংখ্যগুলো অবশ্যই প্রাইম। ছাকনির উপর থেকে সেগুলো সংগ্রহ করে নেই:

“ ২, ৩, ৫, ৭, ১১, ১৩, ১৭, ১৯, ২৩

আমরা সিডের একটা কোড দেখি:

```
1 bool status[1100002];
2 void siv()
3 {
4     int N=1000000;
5     int sq=sqrt(N);
```

স্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

```

6   for(int i=4;i<=N;i+=2) status[i]=1;
7   for(int i=3;i<=sq;i+=2){
8     if(status[i]==0)
9     {
10       for(int j=i*i;j<=N;j+=i) status[j]=1;
11     }
12   }
13   status[1]=1;
14
15 }

```

status অ্যারেটা দিয়ে নির্দেশ করে একটি সংখ্যা প্রাইম নাকি কম্পোজিট। status[i]=0 হলে i একটি প্রাইম। শুরুতে সব ইনডেক্সে 0 আছে, আমরা উপরের অ্যালগরিদম অনুযায়ী নন প্রাইম সংখ্যা গুলোকে কেটে দিবো, অর্থাৎ j যদি নন-প্রাইম হয় status[j]=1 করে দিবো। ৮ নম্বর লাইনে শুরুতেই ২ এর সব মাল্টিপল কেটে দিলাম। এবপরের পরের লুপটা ৩ থেকে শুরু করে ২ করে বাড়াবো কারণ জোড় সংখ্যা নিয়ে আর চিন্তা করা দরকার নেই। ১০ নম্বর লাইনে এসে যদি status[i]=0 পাই তাহলে অ্যালগরিদম অনুযায়ী i অবশ্যই প্রাইম কারণ। এখনও কাটা পড়েনি, এবার i এর সবগুলো মাল্টিপল কেটে দিবো, এজন্য j এর লুপ শুরু করবো  $2*i$  থেকে এবং বাড়াবো i পরিমাণ। আমাদের কাজ শেষ, নন-প্রাইম সংখ্যাগুলো সব কেটে দিবে ভিতরের লুপটি, এখন status[i] এর মান দেখে আমরা i প্রাইম কিনা বের করতে পারবো।

সিভ দিয়ে প্রাইম জেনারেট করে খুব সহজে কোন সংখ্যার প্রাইম ফ্যাক্টর বা উৎপাদকে বিশ্বেষণ করা যায়। এই কাজটা তোমার হাতেই থাকলো :)।

সিভে প্রতিটি সংখ্যা প্রাইম নাকি নন-প্রাইম সেট আমরা একটি বুলিয়ান অ্যারে দিয়ে চেক করি। যত পর্যন্ত প্রাইম জেনারেট করব তত সাইজের অ্যারে লাগবে। ১০^৮ আকারের অ্যারে অনেক মেমোরি দখল করবে। মেমোরি অপটিমাইজ করার জন্য অসাধারণ একটি পদ্ধতি হলো বিট ব্যবহার করা, একে bitwise সিভ বলা হয়। একটি ইন্টিজারে ৩২টি বিট থাকে যার প্রতিটিকে আমরা ফ্ল্যাগ হিসাবে ব্যবহার করতে পারি, সেট নিয়ে বিস্তারিত আলোচনা পাবে [এখানে](#)।

## ফেসবুকে মন্তব্য

2 comments

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

টি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেরিঙ্গ, ডিসিশন প্রবলেম)

# বিটওয়াইজ সিভ(Bitwise sieve)

নভেম্বর ৫, ২০১১ by শাফায়েত



বিটওয়াইজ সিভ প্রাইম সংখ্যা বের করার জন্য প্রচলিত অ্যালগরিদম **Sieve of Eratosthenes** এ মেমরির ব্যবহার অনেক কমিয়ে আনা যায়! সাধারণ সিভে  $N$  পর্যন্ত প্রাইম জেনারেট করলে  $N$  সাইজের একটি অ্যারে ডিক্রেয়ার করতে হয়। অ্যারের প্রতিটি এলিমেন্ট একটি করে ফ্ল্যাগ হিসাবে কাজ করে যেটা দেখে আমরা বুঝি একটি সংখ্যা প্রাইম নাকি কম্পোজিট। বিটওয়াইজ সিভে আমরা ফ্ল্যাগ হিসাবে ইন্টিজার বা বুলিয়ান এর বদলে সবাসবি বিট ব্যবহার করবো।

এ চিউটোরিয়াল পড়ার আগে দুটি বিষয় তোমাকে জেনে আসতে হবে

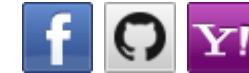
১. Sieve of Eratosthenes এর সাধারণ ভার্সন, তুমি এটা আমার [এই পোস্টটি](#) পড়ে শিখতে পারবে সহজেই।

২. সি/সি++ এ বিটওয়াইজ অপারেটরের ব্যবহার। এটাও খুব সহজে শিখতে পারবে [এখান থেকে](#)। চিউটোরিয়ালটির ১ম ২টি অংশ খুব ভালো করে পড়ে ফেলো, বিটমাস্ক ডিপি, বিএফএস যখন শিখবে তখন অনেক কাজে লাগবে।

আশা করি এখন তুমি বিটওয়াইজ অপারেটর সম্পর্কে অনেক কিছু জানো, সাধারণ সিভ লিখতে কোনো সমস্যা হ্যনা তোমার। এবার আমরা শিখবো বিটওয়াইজ সিভ।

সাধারণ সিভে status বা flag অ্যারেটার কাজ কি? এই অ্যারের ইনডেক্সের মান দেখে আমরা বলতে পারি একটি সংখ্যা প্রাইম কিনা। ধরলাম তোমার status অ্যারেটা ইন্টিজার টাইপের। প্রতিটি ইন্টিজারের মধ্য আছে ৩২টি বিট। আমরা কেনো এতগুলো বিট ব্যবহার করবো খালি ০ বা ১ নির্দেশ করতে? আমরা অ্যারের যেকোনো ইনডেক্সের প্রতিটি বিট দিয়ে একটি সংখ্যা নির্দেশ করতে পারি।

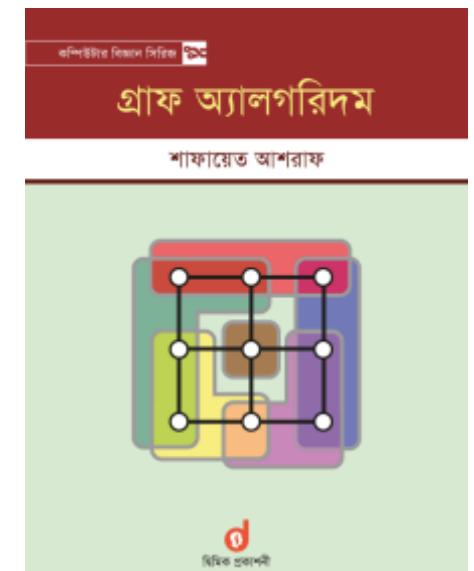
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



তুমি যখন ইন্টিজার অ্যারেতে ১-৭ পর্যন্ত প্রাইম জেনারেট কর, তোমার অ্যারের অবস্থা বাইনারিতে থাকে এরকম:

```
1 status[১]=০০০.....০০(৩২টি শূন্য)
2 status[২]=০০০.....০১(৩১টি শূন্য, ১টি ১)
3 status[৩]=০০০.....০১(৩১টি শূন্য, ১টি ১)
4 status[৪]=০০০.....০০(৩২টি শূন্য)
5 status[৫]=০০০.....০১(৩১টি শূন্য, ১টি ১)
6 status[৬]=০০০.....০০(৩২টি শূন্য)
7 status[৭]=০০০.....০১(৩১টি শূন্য, ১টি ১)
```

প্রতিটি ইনডেক্সে ৩১টি বিট কোনো কাজে লাগছেনা আর্থ এই বিশাল সংখ্যক অব্যবহৃত বিট আমরা সহজেই কাজে লাগাতে পারি। আমরা ধরে নিবেশ:

```
1 status[০] এর
2     >>> শূন্যতম বিট ০ এর প্রাইমালিটি নির্দেশ করে (সবথেকে ডানের বিট==০ তম বিট)
3     >>> ১ম বিট ১ এর প্রাইমালিটি নির্দেশ করে (সবথেকে ডানের বিট==০ তম বিট)
4     >>> ২য় বিট ২ এর প্রাইমালিটি নির্দেশ করে
5     >>> ৩য় বিট ৩ এর প্রাইমালিটি নির্দেশ করে
6     .....
7     >>> ৩১তম বিট ৩১ এর প্রাইমালিটি নির্দেশ করে
8
9 status[১] এর
10    >>> শূন্যতম বিট ৩২ এর প্রাইমালিটি নির্দেশ করে
11    >>> ১ম বিট ৩৩ এর প্রাইমালিটি নির্দেশ করে
12    .....
13
14 status[২] এর
15    >>> শূন্যতম বিট ৬৪ এর প্রাইমালিটি নির্দেশ করে
16    >>> ১ম বিট ৬৫ এর প্রাইমালিটি নির্দেশ করে
```

তাহলে ১-৭ পর্যন্ত প্রাইম জেনারেট করলে তোমার অ্যারের অবস্থা দাঢ়াবে:

“ status[১]= ০০০০.....১০১০১১০০(মোট ৩২টি বিট) (সবথেকে ডানের বিট==০ তম বিট)

ষটি সংখ্যার কাজ একটি ইনডেক্সেই শেষ!!। শুধু ৭টি নয়, আসলে ৩১টি সংখ্যার কাজ শেষ হবে ১টি ইনডেক্স(কারণ প্রতি ইনডেক্সের ৩১টি বিট ব্যবহার করছি আমরা)। এখন প্রশ্ন হলো কোনো সংখ্যার প্রাইমালিটি কত নম্বর ইনডেক্সের কত নম্বর বিট দিয়ে নির্দেশ করা হবে? খুব সহজ, সংখ্যাটি



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্টোরেজ(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

i হলে i/32 নম্বর ইনডেক্সের i%32 নম্বর বিট আমাদের চেক করতে হবে। তাহলে i=1 হলে চেক করবো 0 নম্বর ইনডেক্সের 1 নম্বর বিট, i=33 হলে চেক করবো 1 নম্বর ইনডেক্সের 1 নম্বর বিট ইত্যাদি। (শুন্য বেসড ইনডেক্সিং)

কোড়িং অংশ একদম সহজ। তুমি যেহেতু বিটওয়াজ অপারেটরের ব্যবহার জানো, কোনো সংখ্যার pos তম বিটে 1 বা 0 আছে নাকি সহজেই চেক করতে পারবে। pos তম বিটে নিজের ইচ্ছামত 1 বা 0 বসাতেই পারবে। আমাদের এখানে 0 বসানো দরকার নেই, 1 বসাতে পারলেই চলবে। দুটি ফাংশন লিখে ফেলি:

```
1 bool Check(int N,int pos){return (bool)(N & (1<<pos));}
2 int Set(int N,int pos){ return N=N | (1<<pos);}
```

এবার সিভ লিখে ফেলি:

```
1 int N =100,prime[100];
2 int status[100/32];
3 void sieve()
4 {
5     int i, j, sqrtN;
6     sqrtN = int( sqrt( N ) );
7     for( i = 3; i <= sqrtN; i += 2 )
8     {
9         if( check(status[i/32],i%32)==0)
10        {
11            for( j = i*i; j <= N; j += 2*i )
12            {
13                status[j/32]=Set(status[j/32],j % 32) ;
14            }
15        }
16    }
17    puts("2");
18    for(i=3;i<=N;i+=2)
19        if( check(status[i/32],i%32)==0)
20            printf("%d\n",i);
21
22 }
```

লক্ষ্য করো, আমরা সাধারণ সিভের মত করেই সব লিখেছি, তবে status[i] এর মান চেক করার বদলে status[i/32] এর i%32 নম্বর বিটের মান চেক করেছি।

শ্লাইডিং রেজ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(নতুন)

ব্লুম ফিল্টার(নতুন)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

বিটওয়াইজ সিভ ব্যবহার করে  $10^{18}$  পর্যন্ত প্রাইম তুমি জেনারেট করে ফেলতে পারবে। সাধারণ সিডের থেকে সময় + মেমরি কম লাগবে। সাধারণ গুণ, ভাগ অপারেশনের থেকে বিটের অপারেশনগুলো দ্রুত কাজ করে। আমরা আরো কিছু অপটিমাইজেশন করতে পারি। যেমন তুমি উপরে দেয়া চিউটোরিয়াল পড়ে থাকলে জানো যে কাওকে ২ দিয়ে গুণ করা আর সংখ্যাটির বাইনারিকে ১ ঘর বামে শিফট করা একই কথা। আবার ২ দিয়ে ভাগ করা আর ১ ঘর ডানে শিফট করা একই কথা, ৩২ দিয়ে mod করা আর ৩১ দিয়ে AND করা একই কথা। তাহলে আমরা নিচের মত করে কোডটি লিখতে পারি:

```

1 int status[(mx/32)+2];
2 void sieve()
3 {
4     int i, j, sqrtN;
5     sqrtN = int( sqrt( N ) );
6     for( i = 3; i <= sqrtN; i += 2 )
7     {
8         if( Check(status[i>>5],i&31)==0)
9         {
10            for( j = i*i; j <= N; j += (i<<1) )
11            {
12                status[j>>5]=Set(status[j>>5],j & 31) ;
13            }
14        }
15    }
16    puts("2");
17    for(i=3;i<=N;i+=2)
18        if( Check(status[i>>5],i&31)==0)
19            printf("%d\n",i);
20
21 }
```

ফাংশন ব্যবহার না করে ম্যাক্রো ব্যবহার করলে আরো কম সময় লাগবে। **প্রোগ্রামিং কনটেস্ট** খুব কমই বিটওয়াইজ সিভ ব্যবহার করা দরকার হয়, সাধারণ সিডেই কাজ চলে। তাবপরেও এটা শিখলে বিটের কনসেপ্ট গুলো কিছুটা পরিষ্কার হবে, অন্য কোনো প্রবলেমে হ্যাতো মেমরি কমিয়ে ফেলতে পারবে। ডাইনামিক প্রোগ্রামিং এ আমরা প্রায়ই বিটমাস্কের ব্যবহার করি মূলত মেমরি কমানোর জন্য।

তোমার ইম্প্রেমেটেশন সঠিক নাকি চেক করতে নিচের সমস্যাটি সমাধান করে ফেলো:

<http://www.spoj.pl/problems/TDPRIMES/>

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন  
সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১  
ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম,  
কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

(নোট: অনেকের ভূল ধারণা আছে যে bool টাইপের ড্যারিয়েবলের আকার ১বিট। আসলে bool এর আকার ৮বিট বা এক বাইট,char এর সমান। এর কারণ হলো কম্পিউটার ১ বাইটের ছোটো মেমরি সেগমেন্টকে অ্যাড্রেস করতে পারেনা, তাই ড্যারিয়েবলের নৃন্যতম আকার ১ বাইট)

## ফেসবুকে মন্তব্য

2 comments

# কম্পিনেটোরিক্স: অ্যারেঞ্জমেন্ট এবং ডি-রেঞ্জমেন্ট গণনা

মে ৮, ২০১৩ by শাফায়েত



কনটেস্ট প্রোগ্রামিং এর একটা দারুণ ব্যাপার হলো কনটেস্টেন্টদের শুধু ভালো প্রোগ্রামিং জানলেই হয়না, সাথে ভালো গণিতও জানা দরকার হয়।  
বিশেষ করে কম্পিনেটোরিক্স আর প্রোবাবিলিটিতে ভালো ধারণা থাকলে অনেক ধরণের প্রবলেম সলভ করে ফেলা যায়।

৪টি টুপি পাশাপাশি সাজানো আছে, টুপিগুলোকে যথাক্রমে ১,২,৩,৪ সংখ্যাগুলো দিয়ে চিহ্ন দেয়া হয়েছে। এখন টুপিগুলোকে এলোমেলো করে  
কতভাবে সাজানো যাবে? আমরা কয়েকভাবে সাজিয়ে চেষ্টা করি:

“ ১,২,৩,৪

১, ৩, ২, ৪

১, ৪, ২, ৩

১, ৩, ৪, ২

.....

.....

৪, ৩, ২, ১

সাবস্ক্রাইব

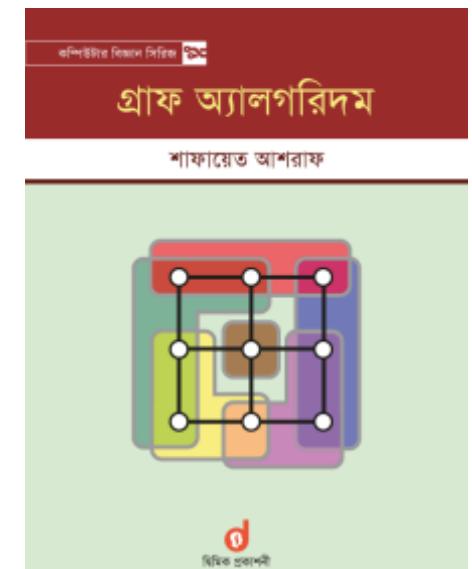


Secured by [OneAll Social Login](#)

আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

**Traveloka Singapore** (বিস্তারিত...)



মোট কতভাবে সাজানো যাবে? কলেজে করে আসা অংক থেকে তুমি সহজেই বলতে পারবে  $factorial(8) = 28$  ভাবে সাজানো যায়। এটাকে আমরা একটু প্রোগ্রামারের দৃষ্টিভঙ্গী থেকে দেখি। ৪টা জায়গা বা স্লট আছে, প্রতিটি স্লটে ১টি করে টুপি বসানো যায়। এখন প্রথম স্লটে ১,২,৩ বা ৪ এর যেকোনো একটা বসালে:

“  
      ১,  
      ২,  
      ৩,  
      ৪

প্রথম স্লটে টুপি কত ভাবে বসানো যায়? অবশ্যই ৪ ভাবে। এখন ২য় স্লটে কয়ভাবে বসানো যায়? একটা টুপি আমরা বসিয়ে ফেলেছি আগেরটায়, তাই ২য় স্লটে বসাতে পারবো  $4-1=3$  ভাবে। ঠিক এভাবে ৩য় স্লটে ২ভাবে এবং ২য় স্লটে ১ ভাবে। তাহলে মোট উপায়  $4 \times 3 \times 2 \times 1 = 28$  টা। ৪টার জায়গায়  $n$  টা টুপি থাকলে কি করতে? আমরা প্রোগ্রামার তাই বারবার কষ্ট করে হিসাব না করে ধূম করে একটা ফাঁশন লিখে ফেলি। মনে করো ফাঁশনটা হলো  $permutation(n)$ ।  $n = 0$  হলে সাজানো যায় ১ ভাবে, তাহলে:

“  $permutation(0) = 0$

$n > 0$  হলে প্রথম স্লটে বসানো যায়  $n$  ভাবে, এরপরে সমস্যাটা ছেটো হয়ে দাঢ়ায় “ $n - 1$  টা টুপি  $n - 1$  টা স্লটে কতভাবে বসানো যায়?” অর্থাৎ সমস্যাটা  $permutation(n - 1)$  হয়ে যায়। সাথে গুণ হবে  $n$  কারণ কারেন্ট স্লটে  $n$  ভাবে বসিয়েছি। তাহলে লিখতে পারি:

“  $permutation(n) = n \times permutation(n - 1)$

আশা করি ব্যাপারটা পরিষ্কার। সহজ ব্যাপারটা নিয়ে এত কথা বললাম যাতে রিকার্ণেন্ট পরিষ্কার হয় যেটা কাজে লাগবে ডিরেঞ্জমেন্ট গোণার জন্য।

এখন ধরো ১,২,৩,৪ এই ৪টা টুপির মালিক হলো যথাক্রমে সাকিব, নাসির, তামিম, রহিম। তারা খুবই ভালো বন্ধু বলে ঠিক করলো একজন আরেকজনের টুপি পড়ে ক্রিকেট খেলতে যাবে। কেও তার নিজের টুপি পড়তে পারবেনা, তাহলে বন্ধুস্ব থাকবেনা! এখন কতভাবে তারা টুপি পড়তে পারবে?

গণিতের ভাষায় এর নাম ডিরেঞ্জমেন্ট, এমন কয়টি পারমুটেশন আছে যেখানে কেও তার নিজের জায়গায় নেই।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রাম

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

১,৩,২,৪ ডি-বেঞ্জমেন্ট নয় কারণ সাকিব আর বহিম তাদের নিজ নিজ টুপি পড়ে আছে(১ ও ৪ নম্বর) ! ২,১,৪,৩ একটি ডি-বেঞ্জমেন্ট, সবাই তার বন্ধুর টুপি পড়েছে।

আমরা একটা ফাংশন বানাবো  $d(n)$  যেটা  $n$  টা টুপি কতভাবে সাজানো যায় যাতে কেও তার নিজের টুপি না পায় সেটা বের করে দেয়।

প্রথম মানুষ সাকিবের কাছে ৪-১=৩টা চয়েস আছে, সে ১ নম্বর বাদে যেকোনো টুপি নিতে পারে। মনে করলাম সে তামিমের টুপি নিলো। এখন ২টা ঘটনা ঘটতে পারে:

“ ১. পরের বার তামিম নিলো সাকিবের টুপি। এখন ৪-২=২ জন মানুষ বাকি, টুপিও বাকি ঠিক ৪-২=২ টা।

“ ২. পরের বার তামিম সাকিব ছাড়া অন্য কারো টুপি নিলো। এখন মানুষ বাকি ৪-১=৩ জন। তামিম যেহেতু সাকিবের টুপি নিচ্ছনা তাই ওটাকেই তার নিষিদ্ধ টুপি ধরতে হবে, আর বাকি সবার কাছে নিষিদ্ধ টুপি হলো তার নিজের টুপিটা। তাহলে এখন ৪-১=৩ জন মানুষের জন্য ৪-১=৩ টা করে চয়েস আছে। লক্ষ্য করো

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েন্ট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



Sakib

Nasir

Tamim

Rahim



Sakib

Nasir

Tamim

Rahim

সাকিব আৰ তামিম একজন আবেকজনের টুপি নিলো, এখন  $8-2=2$  জন মানুষের জন্য সমস্যাটি সমাধান কৰতে হবে।



Sakib

Nasir

Tamim

Rahim

সাকিব তামিমের টুপি নিয়েছে। কিন্তু তামিম সাকিবেটা নিবেনা, তাহলে  $8-1=7$  টা মানুষের জন্য সমস্যাটি সমাধান কৰতে হবে।

দুই ক্ষেত্ৰেই মানুষ আৰ টুপিৰ সংখ্যা সমান থাকছে। 8 এৰ জায়গায় n ধৰে ২টা কভিশন মিলিয়ে সহজেই রিকাৰ্ড রিলেশনটা লিখতে পাৰি:

$$d(n) = (n - 1) * (d(n - 1) + d(n - 2))$$

বেস কেস:  $d(1) = 0, d(2) = 1$

আর্টিকুলেশন পয়েন্ট এৰং ব্ৰিজ

স্ট্ৰাংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্ৰলেম

অয়লাৱ ট্ৰ্যুৱ মিনিমাম ভাৰটেক্স কভাৱ

ট্ৰি এৰ ডায়ামিটাৱ নিৰ্ণয়

লংগেস্ট পাথ প্ৰলেম

অ্যালগৱিদিম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্ৰোগ্ৰামিং(৮):

ডাইনামিক প্ৰোগ্ৰামিং ১ (ফিবোনাচি)

ডাইনামিক প্ৰোগ্ৰামিং ২ (শ্টেচ্ট পাথ)

ডাইনামিক প্ৰোগ্ৰামিং ৩ (LIS এৰং পাথ প্ৰিন্টিং)

ডাইনামিক প্ৰোগ্ৰামিং ৪ (লংগেস্ট কমন  
সাবসিকোয়েম)

ডাইনামিক প্ৰোগ্ৰামিং ৫ (কয়েন চেঞ্জ/০-১  
ন্যাপস্যাক)

ডাইনামিক প্ৰোগ্ৰামিং ৬ (সাবসেচ সাম,  
কম্পিউটেটোৱিঙ্গ, ডিসিশন প্ৰলেম)

এই রিকার্সনটা কোড করার সময় মাথায় রাখতে হবে যে একই ফাংশন অনেকবার কল হচ্ছে, তাই ডিপি টেবিলে মানগুলো সেভ করে রাখতে হবে। তুমি ডাইনামিক প্রোগ্রামিং নিয়ে পড়ালেখা করতে পারো এ সম্পর্কে জানতে।

এবাব আরেকটা মজার উপায়ে প্রবলেমটা সলভ করি।  ${}^n C_r$  বা  $\binom{n}{r}$  এর সাথে তোমরা পরিচিত,  $n$  টা জিনিস থেকে  $r$  টি জিনিস করত্বাবে নেয়া যায় সেটাই প্রকাশ করে  $\binom{n}{r}$ ।  $n$  টা টুপিকে মোট সাজানো যায়  $n!$  উপায়। এর মধ্যে যেসব পারমুটেশনে অন্তত একটি টুপি নিজের জায়গায় আছে তাদের বাদ দিলে ডিরেঞ্জমেন্ট পাওয়া যায়।  $n$  টি টুপি থেকে ১ টি টুপি নেয়া যায়  $\binom{n}{1}$  উপায়ে, ১টি টুপিকে নিজের জায়গায় রেখে বাকি  $n - 1$  টা টুপিকে সাজানো যায়  $(n - 1)!$  উপায়ে। তাহলে  $n! - \binom{n}{1} \times (n - 1)!$  বের করলেই ডিরেঞ্জমেন্ট বের হয়ে যাচ্ছেনা? কারণ আমরা মোট উপায় থেকে যেসব পারমুটেশনকে অন্তত ১ জন নিজের জায়গায় আছে তাদের বাদ দিচ্ছি।  $\binom{n}{1}$  দিয়ে গুণ দিচ্ছি কারণ প্রতিবার ১জন কে ফিল্ড করে  $n - 1$  জনকে পারমুটেশন করতেসি।

কিন্তু এখনে একটা বড় সমস্যা আছে। ধরো তুমি তামিমের টুপিকে তামিমের কাছেই রেখে বাকি টুপিগুলো কয়ত্বাবে সাজানো যায় বের করলে। আবাব নতুন করে সাকিবেরটা সাকিবের কাছে রেখে বাকিগুলো কয়ত্বাবে সাজানো যায় বের করলে। ভালোমত চিন্তা করে দেখ যেসব পারমুটেশনে সাকিবেরটা সাকিবের কাছে আছে আব তামিমেরটা তামিমের কাছে আছে সেগুলো কি ২বাব গণনা করা হয়ে গেলো না?  $\binom{n}{1} \times (n - 1)!$  এ এই কারণে কিছু পারমুটেশন একাধিক বাব ক্যালকুলেট করা হয়ে যাবে। সেগুলো আমরা কিভাবে বাদ দিবো? আমরা ১টা সংখ্যা ফিল্ড করে যখন গুনেছি তখন যেসব পারমুটেশনে ২টা সংখ্যা ফিল্ড সেগুলো একাধিক বাব গুণে ফেলেছি, সেগুলো আমরা বাদ দিয়ে দেই।  $\binom{n}{1} \times (n - 1)!$  থেকে বাদ দিয়ে দিবো  $\binom{n}{2} \times (n - 2)!$ । একটু চিন্তা করলে বুঝতে পারবে এখানেও সমস্যা আছে, যেখানে ৩টা ফিল্ড সেগুলোকেও আমরা বাদ দিয়ে দিচ্ছি!! তাহলে সেটা আবাব যোগ করে দাও। মাথা গুলিয়ে গেলে ভ্যান ডায়াগ্রামের কথা চিন্তা করো:

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্সন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)



ভ্যান ডায়াগ্রামে ঠটা অংশের কমন এরিয়া বের করতে আমরা সবগুলো অংশ যোগ করি, তারপর যেসব অংশ দুটি বৃত্তে আছে সেগুলো বাদ দেই, যেগুলো ঠটি বৃত্তে আছে সেগুলো আবার যোগ করে দেই, বৃত্ত আরো বেশি থাকলে এভাবে যোগ বিয়োগ চলতেই থাকে। দুটি সেট  $A, B$  হলে  $|A \cup B| = |A| + |B| - |A \cap B|$  ঠিক এই কাজটি করবো এখানে। আমাদের ফর্মুলা হবে:

$$n! - \binom{n}{1} \times (n-1)! + \binom{n}{2} \times (n-2)! - \dots + (-1)^k \times \binom{n}{k} \times (n-k)! + \dots + (-1)n$$

কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

**Shafaetsplanet**  
 2.4K likes  
 শাফায়েত আশরাফ

---

Like Page
Be the first of your friends to like this

**Shafaetsplanet**  
 about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটারের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।

আমরা একবার যোগ করছি, একবার বিয়োগ করছি, এভাবে অপ্রযোজনীয় অংশ বাদ দিয়ে ফলাফল পেয়ে যাচ্ছি। এ জিনিসটাই রাশভারী নাম  
হলো ইন্সুশন-এক্সুশন প্রিমিপাল।

আজ এ পর্টেই। সলভ করার জন্য প্রবলেম:

[www.topcoder.com/stat?c=problem_statement&pm=2013](http://www.topcoder.com/stat?c=problem_statement&pm=2013)

<http://uva.onlinejudge.org/external/112/11282.html>

[http://www.lightoj.com/volume_showproblem.php?problem=1095](http://www.lightoj.com/volume_showproblem.php?problem=1095)

## ফেসবুকে মন্তব্য

0 comments

# প্রোবাবিলিটি: এক্সপেন্সেড ভ্যালু

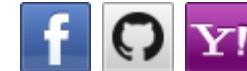
ফেব্রুয়ারি ২৫, ২০১৭ by শাফায়েত



মনে করো তুমি লুডু খেলতে গিয়ে একটা ডাইস বা ছক্কা নিয়ে বোল করছো। এখন তোমার যেকোনো একটা সংখ্যা পাবার প্রোবাবিলিটি কত? বেসিক প্রোবাবিলিটি যদি জেনে থাকো তাহলে তুমি সহজেই বলতে পারবে যে উত্তরটা হলো  $\frac{1}{6}$ । প্রোবালিটি  $\frac{1}{6}$  এর মানেটা কী এখানে? এর মানে হলো, তুমি যদি ছক্কাটা নিয়ে অসীম সংখ্যক বার খেলতে থাকো তাহলে ছয় ভাগের এক ভাগ বার তুমি 1 পাবে, অন্য আরেকভাগ বার তুমি 2 পাবে, অন্য আরেকভাগ বার তুমি 3 পাবে ইত্যাদি। যেমন তুমি 600 বার খেললে, 1 থেকে 6 পর্যন্ত প্রতিটা সংখ্যা 100 বার করে পাবে। এটাতো গেলে গাণিতিক হিসাব, বাস্তবে 600 বার খেলতে গেলে দেখা যাবে প্রতিটা 100 বার না পেয়ে হয়তো কিছুটা কমবেশি হবে। কিন্তু তুমি যত বেশিবার খেলবে তত সংখ্যাগুলো 6 ভাগের 1 ভাগের খুব কাছাকাছি চলে আসবে, অসীমবার খেললে আমরা ধরে নিতে পারি যে প্রতিটা সংখ্যা সমান সংখ্যকবার করে পাবে কারণ প্রতিটা সংখ্যা আসার প্রোবাবিলিটি একই।

এখন প্রশ্ন হলো, ছক্কা দিয়ে যদি আমরা অসীমসংখ্যক বার খেলি তাহলে যে সংখ্যাগুলো পাবো তাৰ গড় মান কত? এটাকেই বলে এক্সপেন্সেড ভ্যালু। কোনো একটা এক্সপেরিমেন্ট অসীম সংখ্যক বার কৰা হলে গড়ে যে ফলাফলটা পাওয়া যায় সেটাৰ নামই এক্সপেন্সেড ভ্যালু। একটা ছক্কার সবগুলো সংখ্যার যোগফল হলো  $1 + 2 + 3 + 4 + 5 + 6 = 21$  এবং যেকোনো সংখ্যা পাবার প্রোবাবিলিটি  $\frac{1}{6}$ । এক্সপেন্সেড ভ্যালু বেৱ কৰতে হলে সবাসবি সব সংখ্যা যোগ না কৰে প্রতিটা সংখ্যা সাথে সেই সংখ্যা পাবার প্রোবাবিলিটি গুণ কৰে দিতে হবে। এই ক্ষেত্ৰে এক্সপেন্সেড ভ্যালু হবে  $1 \cdot \frac{1}{6} + 2 \cdot \frac{1}{6} + 3 \cdot \frac{1}{6} + 4 \cdot \frac{1}{6} + 5 \cdot \frac{1}{6} + 6 \cdot \frac{1}{6} = \frac{1}{6}(1 + 2 + 3 + 4 + 5 + 6) = 3.5$ । তাৰমানে তুমি অসীম সংখ্যকবার খেললে গড়ে প্রতিবাৰ তুমি 3.5 পাবে। এখানে লক্ষ্য কৰাৰ মতো ব্যাপার হলো, আমাদেৱ ছক্কায় 3.5 কোথায় লেখা নেই, এটা হলো অসীম সংখ্যক বার এক্সপেরিমেন্ট কৰে প্রাপ্ত গড় মান।

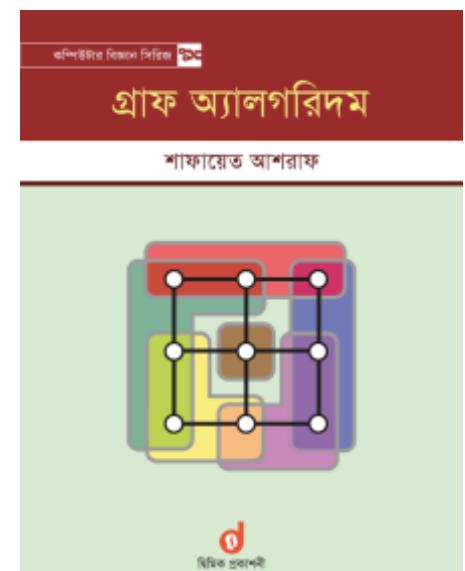
## সাবস্ক্রাইব



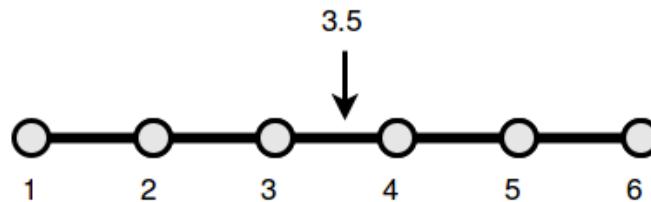
Secured by OneAll Social Login

## আমাৰ সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়াৰ @  
**Traveloka Singapore** (বিস্তারিত...)



তুমি যদি গণিত বাদ দিয়ে নিজের মত করে ভাবো তাহলেও ব্যাপারটা সহজেই বুঝতে পারবে, 3.5 হলো 1 থেকে 6 এর ঠিক মাঝের সংখ্যা, যেকোনো সংখ্যা পাবার প্রোবাবিলিটি যখন সমান, অসীমবার খেললে গড়ে মাঝখনের সংখ্যাটা পাওয়াইতো স্বাভাবিক।



এখন মনে করো কোনো কারণে ছস্কার কিছু পাশ ভারী, কিছু পাশ হালকা, তাই প্রতিটা সংখ্যা পাবার প্রোবাবিলিটি একই না। প্রতিবার ছুড়ে মাঝলে  $\times$  পাবার প্রোবাবিলিটি হলো  $p(x)$  এবং অবশ্যই  $p(1) + p(2) + \dots + p(6) = 1$ । তাহলে এক্সপেক্টেড ভ্যালু কত হবে? খুবই সহজ, আগের বাব প্রতিটা সংখ্যার সাথে  $\frac{1}{6}$  গুণ করেছো, এবাব  $\times$  এর সাথে গুণ করবে  $p(x)$ । এক্সপেক্টেড ভ্যালু তাহলে হবে

$$E = p(1) * 1 + p(2) * 2 + \dots + p(6) * 6$$

আমরা একটা সাধারণ ফর্মুলা বের করাব চেষ্টা করছি যা সবসময় কাজে লাগবে। যদি ছস্কায় 6টি পাশ না থেকে  $n$  টা পাশ থাকে তাহলে কি হবে? তাহলে ছস্কা শব্দটা ব্যবহাব করা যাবে না, ডাইস বলতে হবে! তবে এক্সপেক্টেড ভ্যালুর ফর্মুলায় তেমন পরিবর্তন আসবে না, এখন আমরা যোগ করবো  $n$  টা টার্ম।  $E = p(1) * 1 + p(2) * 2 + \dots + p(n) * n = \sum_{i=1}^n p(i) \cdot i$

আমরা আরো জেনারেলাইজেশন করতে পারি, আমরা এতক্ষণ ধরেছি  $n$  টা পাশে 1 থেকে  $n$  পর্যন্ত প্রতিটা সংখ্যা একবাব করে আছে। এখন আমরা ধৰো ডাইসের  $i$  তম সাইডে যে সংখ্যা লেখা আছে সেটা হলো  $x(i)$  এবং ডাইস ছুড়ে মাঝলে  $i$  তম সাইডটা পাবার প্রোবাবিলিটি আগের মতোই  $p(i)$ । এখন ফর্মুলাটা হবে  $E = p(1) * x(1) + p(2) * x(2) + \dots + p(n) * x(1) = \sum_{i=1}^n p(i) \cdot x(i)$ । সহজভাবে বলতে গেলে প্রতিটা  $i$  এর জন্য আমরা  $i$  তম সাইডে যে সংখ্যাটা লেখা আছে সেটাকে  $i$  তম সাইড পাবার প্রোবাবিলিটি দিয়ে গুণ করছি এবং সবগুলো গুণফল যোগ করে দিচ্ছি।

তুমি যখন গণিতের উপর কোনো একাডেমিক বই পড়বে তখন দেখবে এক্সপেক্টেড ভ্যালুর সংজ্ঞায় উপরের ফর্মুলাটাই দেয়া আছে, সাথে Random Variable নিয়ে কিছু কথ্যবার্তা আছে। আমাদের উদাহরণে random variable হলো ডাইসের গায়ে লেখা সংখ্যা যেটার মান হবে পাবে  $x(1), x(2), \dots, x(n)$ । শুধু ডাইস না, যেকোনো এক্সপেরিমেন্টের ক্ষেত্ৰেই তুমি উপরের ফর্মুলা কাজে লাগাতে পারবে।



প্রোগ্রামিং কনটেন্ট এবং অ্যালগরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিঙ্কড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

আমরা এতক্ষণ যা শিখলাম সেটা দিয়ে কয়েকটি সমস্যা সমাধান করে ফেলি।

## সমস্যা ১:

ধরো তোমার কাছে একটা কয়েন আছে, এই কয়েনেরও এক পাশ একটু ভারী, কয়েনটা একবার টস করলে হেড পাবার প্রোবাবিলিটি  $0.4$  এবং টেইল পাবার প্রোবাবিলিটি  $1 - 0.4 = 0.6$ । তুমি কয়েনটা টস করছো যতক্ষণ না পর্যন্ত হেড পাও। হেড পাবার জন্য গড়ে (এক্সপেক্টেড) তোমার কয়বার কয়েন টস করা লাগবে?

মনে করো যে গড়ে  $E$  বার কয়েন টস করলে তুমি হেড পাবে। এখানে মনে রাখতে হবে, প্রতিটা কয়েন টস একটা স্বাধীন বা ইন্ডিপেন্ডেন্ট ইভেন্ট, তারমানে একবার টস করলে যে ফলাফল পাবে তারউপর পরবর্তি টসের ফলাফলের কোনো সম্পর্ক নেই। এখন এক্সপেরিমেন্টের ফলাফল আমাদের দুইরকম হতে পারে:

তুমি একটা টেইল পেয়েছ, একটা টস নষ্ট হলো, তোমাকে এখনো গড়ে  $E$  বার টস করতে হবে। এক্সপেক্টেড ভ্যালু  $0.4(1 + E)$ ।

রিকার্সনের ধারণাটা এখানে কাজে লাগছে।

তুমি একটা হেড পেয়েছ, তোমার আর টস করা দরকার নেই। এক্সপেক্টেড ভ্যালু  $(0.6) \cdot (1)$ ।

সবমিলিয়ে গড়ে তোমাকে  $E = 0.4 \cdot (1 + E) + (0.6) \cdot (1)$  বার কয়েন টস করতে হবে। এটাকে ঘূরিয়ে লিখলে আমরা পাবো  $1.6667$ । এরমানে অসীম সংখ্যকবার এক্সপেরিমেন্ট করলে তুমি গড়ে  $1.6667$  টা টসের পরেই হেড পাবে।

## সমস্যা ২:

তোমার কাছে একটা কয়েন আছে যেটা টস করলে হেড কিংবা টেইল পাবার প্রোবাবিলিটি হলো  $p(h)$  এবং  $p(t)$ । পরপর দুটি হেড পেতে হল গড়ে (এক্সপেক্টেড) কয়বার টস করতে হবে?

মনে করো  $E$  বার টস করলে তুমি পরপর দুইটি হেড পাবে। এখন তুমি যদি প্রথমবার টেইল পাও তাহলে একটা টস নষ্ট হবে এবং তোমাকে গড়ে আরো  $p(t) \cdot (1 + E)$  বার টস করতে হবে। কিন্তু তুমি যদি প্রথমবার হেড পাও তাহলে দুটি ঘটনা ঘটতে পারে, পরের বার তুমি টেইল পাবে এবং আরো  $p(t) \cdot (1 + E)$  বার টস করতে হবে, অথবা পরেরবার তুমি আরেকটা হেড পাবে এবং আর টস করা দরকার নেই। সবগুলো ঘটনা

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ট ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্থ্যার-রুট ডিকম্পেজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

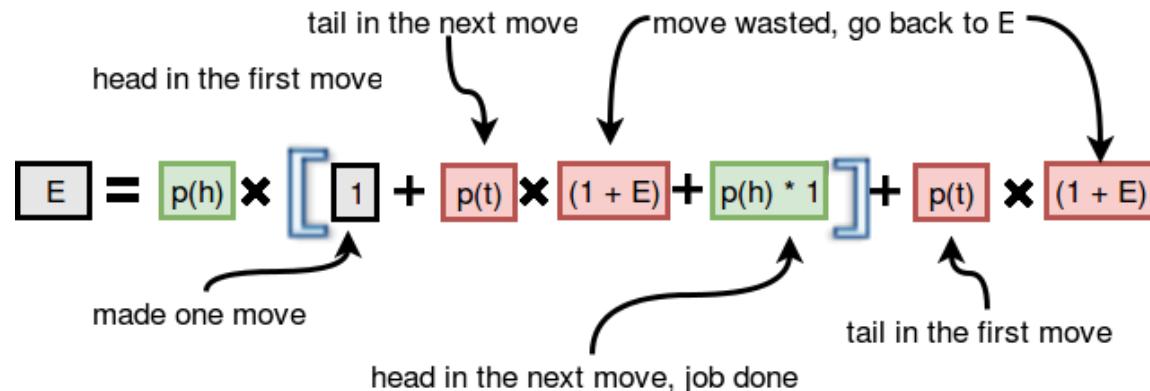
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

একসাথে করলে এক্সপেন্টেড ভ্যালু হবে  $E = p(h) \cdot (1 + p(t) \cdot (1 + E) + p(h) \cdot 1) + p(t) \cdot (1 + E)$ । নিচের ছবিতে আবারো ব্যাখ্যা করা আছে সূত্রটা:



যদি কয়েনটা ফেয়ার কয়েন হয়, অর্থাৎ  $p(h) = p(t) = 0.5$  হ্য তাহলে  $E$  এর মান হবে 6, তুমি যাচাই করে দেখতে পারো।

এখন প্রশ্ন হলো দুইটির বদলে পরপর  $n$  টা হেড পেতে চাইলে কয়ার টস করতে হবে? এটা তুমি চিন্তা করে বের করো, না পারলে উত্তর আছে [এই সাইটে](#)।

### সমস্যা ৩:

একটা কয়েন  $n$  বার টস করা হলে তুমি এক্সপেন্টেড কয়টি হেড পাবে?

প্রথমে চিন্তা করে যদি কয়েন ১বার টস করা হ্য তাহলে তুমি গড়ে (এক্সপেন্টেড) কয়টি হেড পাবে? 0.5 প্রোবাবিলিটি যে তুমি ১টি হেড পাবে, বাকি 0.5 প্রোবাবিলিটিতে তুমি একটাও হেড পাবে না। তাহলে এক্সপেন্টেড ভ্যালু হলো  $0.5 \cdot 1 + 0.5 \cdot 0 = 0.5$ । এর মানে হলো তুমি যদি অসীমসংখ্যক বার এক্সপেরিমেন্ট করো এবং প্রতি এক্সপেরিমেন্টে ১বার করে কয়েন টস করো তাহলে গড়ে তুমি প্রতিবার 0.5 টি হেড পাবে।

$n$  বার টস করলে তোমাকে এই ভ্যালুটাই  $n$  বার যোগ করতে হবে:

$(0.5 \cdot 1 + 0.5 \cdot 0) + (0.5 \cdot 1 + 0.5 \cdot 0) + \dots (n \text{ times}) = n \cdot 0.5$ । এর মানে হলো তুমি যদি অসীমসংখ্যক বার

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংলি কানেক্টেড কম্পানেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটারিঙ্গ, ডিসিশন প্রবলেম)

এক্সপেরিমেন্ট করো এবং প্রতি এক্সপেরিমেন্টে  $n$  বার করে কয়েন টস করো তাহলে গড়ে তুমি প্রতিবার  $n \times 0.5$  টি হেড পাবে।

একইরকম আরেকটা সমস্যা হলো,  $n$  টি শিক্ষার্থী আছে, প্রত্যেককে বলা হলো 1 থেকে 100 এর মধ্যে একটা সংখ্যা লিখতে। অসীম সংখ্যকবার এক্সপেরিমেন্টটা করা হলে গড়ে কতজন শিক্ষার্থী ১ থেকে  $n$  এর মধ্যে কোনো একটা সংখ্যা লিখবে? ধরে নাও প্রতিটি সংখ্যা লেখার প্রোবাবিলিটি সমান (যদিও বাস্তবে এক্সপেরিমেন্টটা করা হলে সেটা সত্য হবে না, মানুষ কিছু কিছু সংখ্যাকে বেশি পছন্দ করে!)।

## সমস্যা ৪:

$n$  টা হেড পেতে হলে তোমাকে এক্সপেন্সেড ক্যবার কয়েন টস করতে হবে?

এই সমস্যাটাকে আমরা রিকার্সিভলি সলভ করবো। মনে করো  $n$  টা হেড পেতে হল  $E(n)$  বার কয়েন টস করতে হবে। এখন যদি একটা হেড পাই তাহলে আমার আরো  $n - 1$  টা কয়েন লাগবে যার জন্য আমাকে আরো  $E(n - 1)$  বার টস করতে হবে। কিন্তু যদি একটা টেইল পাই তাহলে আমাকে আরো  $E(n)$  বার টস করতে হবে।

তাহলে মোট কয়েন টস করতে হবে  $E(n) = 0.5 \cdot (1 + E(n - 1)) + 0.5 \cdot (1 + E(n))$  বার। এটাকে সরল করলে পাবে  $E(n) = E(n - 1) + 2$ । এখন আমাদের রিকার্সন থামানোর জন্য একটা বেস কেস দরকার। যদি আমাদের 0 টা হেড লাগে তাহলে আর টস করা দরকার নেই,  $E(0) = 0$ ।

## সমস্যা ৫:

এই সমস্যাটা ২০১৭'র NCPC কন্টেন্সের প্রিলিমিনারিতে আমি সেট করেছিলাম। তোমার কাছে  $n$  টা বাল্ব আছে, শুরুতে সবগুলো বাল্ব অফ। প্রতিটা মুড এ তুমি random একটা বাল্ব সিলেক্ট করতে পারো। এখন বাল্বটা যদি অফ থাকে তাহলে তুমি একটা কয়েন টস করবে, যদি হেড পাও তাহলে বাল্বটা অন করবে, যদি টেইল পাও তাহলে কিছুই করবে না। আর বাল্বটা যদি আগেই অন থাকে তাহলে সেই মুডে তোমার কিছুই করা দরকার নেই। এক্সপেন্সেড ক্যটা মুডে তুমি সবগুলো বাল্ব অন করতে পারবে? কয়েনটা ফেয়ার কয়েন না, প্রতিবার টেইল পাবার প্রোবাবিলিটি  $p$ ।

এই প্রবলেমও রিকার্সিভলি সলভ করতে হবে। তোমার মুলত জানা দরকার বর্তমানে কয়টা বাল্ব অন আছে। ধরো বর্তমানে  $x$  টা বাল্ব অন আছে এবং এক্সপেন্সেড মুড লাগবে  $e(x)$  টি। তাহলে অলবেডি অন আছে এমন বাল্ব সিলেক্ট করার প্রোবাবিলিটি  $\frac{x}{n}$ , সেক্ষেত্রে এক্সপেন্সেড মুড লাগবে

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাকিং(১):

ব্যকট্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেন্সেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকার্সন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

আরো  $\frac{xn}{n-x}(1 + e(x))$  টি। অলবেড়ি অফ আছে এমন বাল্ব সিলেষ্ট কৰাৰ প্ৰোবাবিলিটি  $\frac{n-x}{n}$ । সেক্ষেত্ৰে আবাৰ ২টি ঘটনা ঘটতে পাৰে,  $p$  প্ৰোবাবিলিটিতে তুমি টেইল পাৰে এবং আৱো  $e(x)$  টি মুভ লাগবে, অথবা  $1 - p$  প্ৰোবাবিলিটিতে হেড পাৰে এবং আৱো  $e(x+1)$  টি মুভ লাগবে।

$$\text{সবমিলিয়ে ইকুয়েশনটা হবে } e(x) = \frac{x}{n} \cdot (1 + e(x)) + \frac{n-x}{n} \cdot (p \cdot (1 + e(x)) + (1-p) \cdot (1 + e(x+1)))$$

এক্সপেক্টেড ভ্যালু নিয়ে আৱো জানতে কোডশেফেৱ [এই আর্টিকেলটি](#) পড়তে পাৰো। প্ৰ্যাকটিস কৰাৰ জন্য [lightoj](#)'ৰ [প্ৰোবাবিলিটি সেকশনটা](#) দেখো।

হ্যাপি কোডিং!

## ফেসবুকে মন্তব্য

0 comments

কোয়ান্টাম কম্পিউটাৰ কী?

কোয়ান্টাম কম্পিউটাৰেৰ শক্তি এবং সীমাবদ্ধতা

# রবিন-কার্প স্ট্রিং ম্যাচিং

ডিসেম্বর ১, ২০১৬ by শাফায়েত

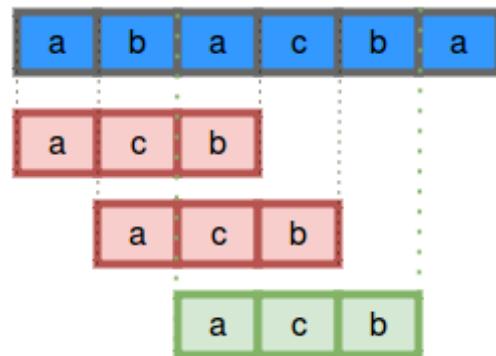


রবিন-কার্প (Rabin-Carp) একটি স্ট্রিং ম্যাচিং অ্যালগরিদম। দুটি স্ট্রিং দেয়া থাকলে এই অ্যালগরিদমটি বলে দিতে পারে যে ২য় স্ট্রিংটি প্রথম স্ট্রিং এর সাবস্ট্রিং কিনা। রবিন-কার্প বোলিং হ্যাশ টেকনিক ব্যবহার করে স্ট্রিং ম্যাচিং করে। যদিও স্ট্রিং ম্যাচিং এর জন্য কেএমপি অ্যালগরিদম ব্যবহার করাই ভালো, কিন্তু রবিন-কার্প শেখা গুরুত্বপূর্ণ মূলত বোলিং হ্যাশ কিভাবে কাজ করে সেটা শেখার জন্য।

এই লেখাটা পড়ার আগে [মডুলার অ্যারিথমেটিক সম্পর্কে](#) জেনে আসতে হবে।

স্ট্রিং ম্যাচিং করার সময় প্রথম স্ট্রিং টাকে আমরা বলবো টেক্সট (Text) এবং দ্বিতীয়টিকে প্যাটার্ন (Pattern)। আমাদের কাজ হলো টেক্সট এর মধ্যে প্যাটার্ন খুজে বের করা।

প্রথমে আমরা একটা ক্রটফোর্স অ্যালগরিদমের কথা ডাবি। আমরা টেক্সট এবং প্রতিটা সাবস্ট্রিং বের করে প্যাটার্নের সাথে মিলিয়ে দেখতে পারি:



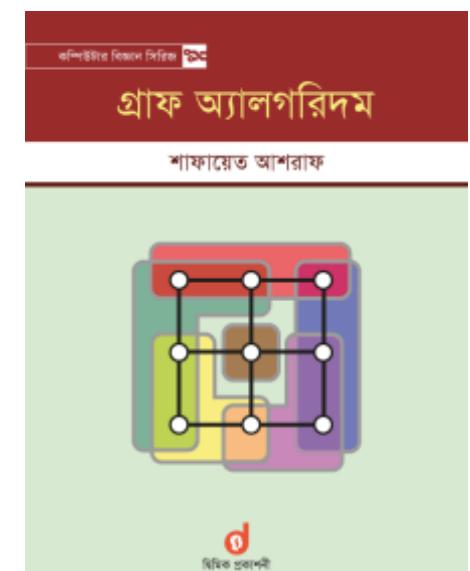
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



ছবিতে abacba টেক্সট এর ভিতর acb প্যাটার্ন খোজা হচ্ছে।

```
1 function naive_matching(text, pattern){  
2     n = text.size()  
3     m = pattern.size()  
4     for(i = 0; i < n; i++) {  
5         for(j = 0; j < m && i + j < n; j++) {  
6             if(text[i + j] != pattern[j]) {  
7                 break; // mismatch found, break the inner loop  
8             }  
9         }  
10        if(j == m) {  
11            // match found  
12        }  
13    }  
14 }
```

নেইড স্ট্রিং অ্যালগোরিদমের কমপ্লেক্সিটি  $O(n * m)$ , যেখানে  $n$  হলো টেক্সট এর দৈর্ঘ্য এবং  $m$  হলো প্যাটার্ন এর দৈর্ঘ্য।

আমদের যদি দুটি স্ট্রিং তুলনা করার সময় একটা একটা ক্যারেক্টার না দেখে ইন্টিজারের মতো  $O(1)$  এ তুলনা করতে পারতাম তাহলে আমরা খুব দ্রুত স্ট্রিং ম্যাচিং করতে পারতাম। হ্যাশিং টেকনিক ব্যবহার করে আমরা স্ট্রিং কে ইন্টিজারে পরিণত করতে পারি। রবিন-কার্প অ্যালগোরিদম এ সেটারই সুবিধা নেয়া হয়েছে।

আমরা যেকোন স্ট্রিংকে একটা Base-B সংখ্যা হিসাবে কল্পনা করতে পারি যেখানে  $B$  এর মান অ্যাসকিতে যতগুলো ক্যারেক্টার আছে তার সমান বা বড়। তাহলে আমরা একটা ফাংশন লিখতে পারি যেটা  $s$  কে Base-B সংখ্যা থেকে Base-10 সংখ্যায় রূপান্বর করবে। এটাই হতে পারে আমাদের হ্যাশ ফাংশন:

$$Hash(s) = s_0 \cdot B^{m-1} + s_1 \cdot B^{m-2} + \dots + s_{m-1} \cdot B^1 + s_{m-1} \cdot B^0$$

এই ফাংশনে প্যারামিটার হিসাবে একটা স্ট্রিং পাঠানো হয়েছে।  $s_i$  দিয়ে বুঝানো হয়েছে  $i$  তম ক্যারেক্টারের অ্যাসকি ড্যালু। এই হ্যাশ ফাংশন দিয়ে যেকোনো স্ট্রিং এর জন্য ডি঱ ডি঱ হ্যাশড্যালু পাওয়া যাবে। কিন্তু সমস্যা হলো ওভারফ্লো, হ্যাশড্যালুর মান সহজেই 64-বিট এর বড় হয়ে যাবে। এই জন্য আমাদেরকে হ্যাশ ড্যালুটাকে  $M$  দিয়ে ভাগ করে ভাগশেষ (modulo) নিতে হবে। তাহলেই সংখ্যাটা  $M$  এর থেকে ছোটো হয়ে যাবে:



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগোরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি ন্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

$$Hash(s) = (s_0 \cdot B^{m-1} + s_1 \cdot B^{m-2} + \dots + s_{m-1} \cdot B^1 + s_{m-1} \cdot B^0) \bmod M$$

কিন্তু এইস্কেত্রে সমস্যা হলো একাধিক স্ট্রিং এর হ্যাশভ্যালু একই হয়ে যেতে পারে। এই সমস্যাটাকে বলা হয় হ্যাশ কলিশন (hash collision)। তবে  $B$  এবং  $M$  যদি প্রাইম সংখ্যা হয় এবং  $M$  এর মান অনেক বড় হয় তাহলে কলিশন করার সম্ভাবনা খুব কমে যায়। (তবে সম্ভাবনা কমে গেলেও একদম শূণ্য হয়ে যায় না, যে জন্য বিবিন্ন কার্পেরও worse case complexity  $O(n * m)$ , সে কথায় পরে আসছি)

এখন প্রথমেই আমাদের কাজ হবে প্যাটার্নের হ্যাশ ভ্যালু বের করা। এরপর টেক্সট এর প্রতিটা  $m$  দৈর্ঘ্যের সাবস্ট্রিং এর জন্য হ্যাশভ্যালু বের করে  $Hash(pattern)$  এর সাথে মিলিয়ে দেখতে হবে। এখন প্রশ্ন হলো প্রতিটা  $m$  দৈর্ঘ্যের সাবস্ট্রিং এর জন্য হ্যাশভ্যালু কিভাবে বের করবো? যদি প্রতিটা সাবস্ট্রিং কে তুমি উপরের হ্যাশ ফাংশনে পাঠাও তাহলে কমপ্লেক্সিটি হয়ে যাবে  $O(n * m)$ । আমাদেরকে একটা পদ্ধতি বের করতে হবে যেন স্ট্রিং এর উপর শুধু একটা লুপ চালিয়েই  $O(n)$  এ প্রতিটা  $m$  দৈর্ঘ্যের সাবস্ট্রিং এর জন্য হ্যাশভ্যালু বের করা যায়। এখানেই বোলিং হ্যাশ পদ্ধতি কাজে লাগবে।

মনে করো  $H_i$  হলো  $s$  এর  $i$  তম ইনডেক্সের শুরু হয়েছে এমন  $m$  দৈর্ঘ্যের স্ট্রিং এর হ্যাশ ভ্যালু। তাহলে আমরা লিখতে পারি:

$$H_i = s_i \cdot B^{m-1} + s_{i+1} \cdot B^{m-2} + \dots + s_{i+m-1} \cdot B^0$$

এখন যদি  $m = 3$  হয় তাহলে  $H_0$  এবং  $H_1$  কে লিখতে পারি:

$$H_0 = s_0 \cdot B^2 + s_1 \cdot B^1 + s_2$$

$$H_1 = s_1 \cdot B^2 + s_2 \cdot B^1 + s_3$$

এখন দেখো  $H_1$  কে কিভাবে  $H_0$  এর মাধ্যমে প্রকাশ করা যায়:

$$H_1 = ((s_0 \cdot B^2 + s_1 \cdot B^1 + s_2) - (s_0 \cdot B^2)) \times B + s_3$$

$$H_1 = (H_0 - S_o \cdot B^2) \cdot B + s_3$$

শ্লাইডিং রেজ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
সর্ট

ডায়াক্রস্ট্রা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

সাধারণভাবে বলা যায়:

$$H_i = (H_{i-1} - S_{i-1} \cdot B^{m-1}) \cdot B + s_{i+m-1}$$

এখন এই সূত্র ব্যবহার করে খুব সহজেই  $O(n)$  এ প্রতিটা  $m$  দৈর্ঘ্যের সাবস্ট্রিং এর হ্যাশ ভ্যালু বের করা যাবে। শুরুতে প্রথম  $m$  ক্যারেক্টারের জন্য হ্যাশভ্যালু বের করে নিয়ে এরপর বোলিং হ্যাশ পদ্ধতিতে বাকিগুলো বের যাবে।

রবিন-কার্পের একটা [সি++ কোড](#) দেখি:

```
1 //Implementation of Rabin Carp String Matching Algorithm
2 //https://github.com/Shafaet/Programming-Contest-Algorithms/blob/master/Useful%20C%2B%2B%20Libraries
3 #include <bits/stdc++.h>
4 using namespace std;
5
6 typedef long long i64;
7
8 //Returns the index of the first match
9 //Complexity O(n+m), this is unsafe because it doesn't check for collisions
10
11 i64 Hash(const string &s, int m, i64 B, i64 M){
12     i64 h = 0, power = 1;
13     for(int i = m-1;i>=0;i--){
14         h = h + (s[i] * power) % M;
15         h = h % M;
16         power = (power * B)%M;
17     }
18     return h;
19 }
20 int match(const string &text, const string &pattern) {
21     int n = text.size();
22     int m = pattern.size();
23     if(n < m) return -1;
24     if(m == 0 or n == 0)
25         return -1;
26
27     i64 B = 347, M = 1000000000+7;
28
29     //Calculate B^(m-1)
30     i64 power = 1;
```

আর্টিকুলেশন পয়েন্ট এবং বিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

```

31     for(int i=1;i<=m-1;i++){
32         power = (power * B) % M;
33
34     //Find hash value of first m characters of text
35     //Find hash value of pattern
36     i64 hash_text = Hash(text, m, B, M);
37     i64 hash_pattern = Hash(pattern, m, B, M);
38
39     if(hash_text == hash_pattern){ //returns the index of the match
40         return 0;
41         //We should've checked the substrings character by character here as hash collision might have occurred
42     }
43
44     for(int i=m;i<n;i++){
45
46         //Update Rolling Hash
47         hash_text = (hash_text - (power * text[i-m]) % M) % M;
48         hash_text = (hash_text + M) % M; //take care of M of negative value
49         hash_text = (hash_text * B) % M;
50         hash_text = (hash_text + text[i]) % M;
51
52         if(hash_text==hash_pattern){
53             return i - m + 1; //returns the index of the match
54             //We should've checked the substrings character by character here as hash collision might have occurred
55         }
56     }
57     return -1;
58 }
59
60
61 int main() {
62     cout<<match("HelloWorld", "ello")<<endl;
63     return 0;
64 }
```

এই কোডের কমপ্লেক্সিটি  $O(n + m)$ । কিন্তু এখানে একটা বড় সমস্যা আছে। যখন `hash_text` আর `hash_pattern` মিলে যাচ্ছে তখন আমরা ধরে নিচ্ছি যে স্ট্রিং দুটি মিলে গিয়েছে। কিন্তু আগে বলেছিলাম যে ভাগশেষ(mod) নেয়ার কারণে একাধিক স্ট্রিং এর একই হ্যাশড্যালু আসতে পারে (কলিশন হতে পারে)। সে ক্ষেত্রে এই কোড ভুল আউটপুট দিবে।

কলিশনের কারণে ভুল এড়ানোর একটা উপায় হলো, যখনই `hash_text = hash_pattern` হবে তখন আবার লুপ চালিয়ে ক্যারেক্টার বাই ক্যারেক্টার মিলিয়ে দেখা। কিন্তু সেক্ষেত্রে worse case কমপ্লেক্সিটি  $O(n * m)$  হয়ে যাচ্ছে যেটা ক্রট ফোর্সের কমপ্লেক্সিটির সমান।

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যকট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্যা মিডল

টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

আরেকটা উপায় হলো ডাবল হ্যাশিং করা। ডাবল হ্যাশিং মানে হলো ভিন্ন ভিন্ন B এবং M ব্যবহার করে প্রতিটা স্ট্রিং এর জন্য দুটি করে হ্যাশ ভ্যালু বের করবো। সেক্ষেত্রে দুই জায়গাতেই কলিশনের সম্ভাবনা খুবই কমে যাবে। চাইলে ২বারের বেশিও হ্যাশিং করা যায়। ডাবল হ্যাশিং করলে প্রোগ্রামিং কনটেন্টে বেশিভাগ সময় পার পেয়ে যাওয়া যাবে, কিন্তু বাস্তব ক্ষেত্রে এটা খুব একটা ভালো উপায় না, কারণ কলিশনের সম্ভাবনা কমে গেলেও একদম শূন্য হয়ে যাচ্ছে না।

এসব কারণে রবিন-কার্প স্ট্রিং ম্যাচিং এর জন্য তেমন একটা ব্যবহার করা হয় না, এর থেকে [কেএমপি](#) ব্যবহার করা সুবিধাজনক। কিন্তু রোলিং হ্যাশ টেকনিক অনেক ধরণের সমস্যা সমাধান করতে কাজে লাগে যে কারণে আমরা রবিন-কার্প শিখি।

### চিন্তা করার জন্য সমস্যা:

তোমাকে দুটি স্ট্রিং  $s_1, s_2$  দেয়া আছে। তোমাকে এদের লংগেস্ট কমন সাবস্ট্রিং এর দৈর্ঘ্য বের করতে হবে। অর্থাৎ সবথেকে বড় স্ট্রিং বের করতে হবে যেটা দুটি স্ট্রিং এবই সাবস্ট্রিং।

### সমাধান:

প্রথমে চিন্তা করো যেকোনো ইন্টিজার  $k$  এর জন্য কি তুমি বের করতে পারবে যে  $k$  দৈর্ঘ্যের কোনো লংগেস্ট কমন সাবস্ট্রিং আছে নাকি। এটা সহজেই  $O(n)$  কমপ্লেক্সিটিতে বের করা যাবে রোলিং হ্যাশ ব্যবহার করে।  $s_1$  এর প্রতিটা  $k$  দৈর্ঘ্যের সাবস্ট্রিং এর হ্যাশভ্যালু বের করো, এরপর  $s_2$  এর প্রতিটা  $k$  দৈর্ঘ্যের সাবস্ট্রিং এর হ্যাশভ্যালু বের করো। যদি কোনো হ্যাশভ্যালু কমন থাকে তারমানে  $k$  দৈর্ঘ্যের কোনো লংগেস্ট কমন সাবস্ট্রিং আছে।

এখন লংগেস্ট কমন সাবস্ট্রিং এর সর্বোচ্চ দৈর্ঘ্য হতে পারে  $\max_len = \min(s_1.length, s_2.length)$ । এখন তুমি 0 থেকে  $\max_len$  এর উপর বাইনারি সার্চ চালিয়ে সহজেই সমস্যাটা সমাধান করতে পারবে। কমপ্লেক্সিটি হবে  $O(n \cdot \log n)$ ।

হ্যাপি কোডিং!

রেফারেন্স: [টপকোডার](#)

কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

The screenshot shows a Facebook page for 'Shafaetsplanet' with 2.4K likes. The post, made about an hour ago, discusses quantum computing, mentioning Shafayet Al-Ashrafi and a link to SHAFAETSPLANET.COM. The post text is in Bengali.

Be the first of your friends to like this

Shafaetsplanet about an hour ago

ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিনের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্যাঁ।

i, W) = 1 if W =  
i + 1, W - C[i]) -

SHAFAEETSPALNET.COM  
ডাইনামিক প্রোগ্রামিং ৬ / ম্যান

# স্ট্রিং ম্যাচিং: নুথ-মরিসন-প্র্যাট (কেএমপি) অ্যালগরিদম

জুন ১১, ২০১৮ by শাফায়েত



আজকে আমরা শিখবো কেএমপি (KMP) অ্যালগরিদম ব্যবহার করে স্ট্রিং ম্যাচিং করা। কেএমপি শব্দটি এসেছে ৩জন কম্পিউটার বিজ্ঞানী

Donald Knuth, James H. Morris এবং Vaughan Pratt এর নাম থেকে।

আমাদের প্রবলেম হলো একটা দুটি স্ট্রিং text এবং pattern দেয়া আছে, আমাদের বলতে হবে text এর ভিতর pattern স্ট্রিংটি সাবস্ট্রিং হিসাবে আছি কিনা। যেমন ধরো টেক্সটটি হলো “MOD”, এই স্ট্রিংটার ৬টা সাবস্ট্রিং আছে “M”, “O”, “D”, “MO”, “OD” এবং “MOD”, এখন যদি pattern = “MO” খুজতে বলে আমরা true রিটোর্ন করবো।

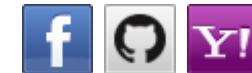
ক্রটফোর্স অ্যালগরিদম ব্যবহার করে স্ট্রিং ম্যাচিং করার টাইম কমপ্লেক্সিটি  $O(n * m)$ , যেখানে  $n$  এবং  $m$  হলো টেক্সট ও প্যাটার্নের দৈর্ঘ্য। কিন্তু কেএমপি ব্যবহার করে  $O(n + m)$  কমপ্লেক্সিটিতে প্যাটার্ন খুজে বের করা যায়।

আমরা প্রথমে ক্রটফোর্স অ্যালগরিদমটা শিখবো এবং শিখতে গিয়ে দেখবো যে আমরা কিছু কাজ বারবার করছি যেটা একটু বুদ্ধিমানের মত চিন্তা করলে করার দরকার নেই। সেখান থেকে আমরা কেএমপি অ্যালগরিদম শিখবো। খাতা-কলম বা হোয়াইট-বোর্ড ছাড়া কেএমপি অ্যালগরিদম ব্যাখ্যা করা আমার জন্য একটু কষ্টকর, তাও আমি চেষ্টা করছি, আশা করি লেখাটা দ্রুত পড়ে ফেলার চেষ্টা না করে ধীরে ধীরে মনযোগ দিয়ে পড়বে।

মনে করো আমাদের টেক্সট হলো “abababacd” এবং প্যাটার্ন হলো “ababac”。 ক্রটফোর্স অ্যালগরিদমে আমরা টেক্সটের প্রতিটা ইনডেক্সে গিয়ে সেখান থেকে লুপ চালিয়ে প্যাটার্ন খুজে বের করার চেষ্টা করি।

```
1 bool naive_matching(string text, string pattern) {
2     int n = text.size();
```

## সাবস্ক্রাইব

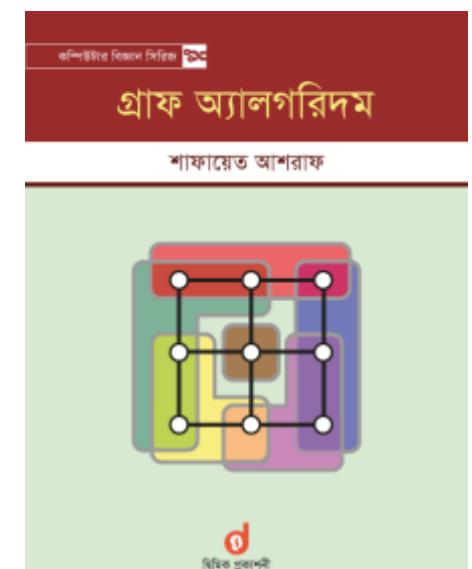


Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

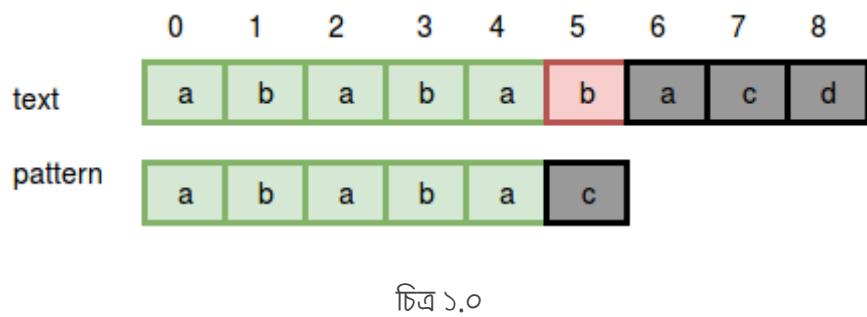
**Traveloka Singapore** (বিস্তারিত...)



```

3 int m = pattern.size();
4 for(int i = 0; i < n; i++) {
5     //for each position i in text, we will try
6     //to match text[i, i+1, ..., i+m-1] with pattern[0, 1, ... m-1]
7     int j = 0;
8     for(j = 0; j < m && i + j < n; j++) {
9         if(text[i + j] != pattern[j]) {
10            break; // mismatch found, break the inner loop
11        }
12    }
13    if(j == m) {
14        return true;
15    }
16 }
17 return false;
18 }
```

আমরা এই কোডটাকে সিমুলেট করার চেষ্টা করি। শুরুতে  $i = 0$ , আমরা টেক্সটের প্রথম ক্যারেক্টার থেকে শুরু করে প্যাটার্ন এবং টেক্সটের একটি একটি করে ক্যারেক্টার মিলাতে থাকবো। যদি সবগুলো ক্যারেক্টার মিলে যায় তাহলে আমরা প্যাটার্ন পেয়ে গেছি। যদি কোনো এক জায়গায় গিয়ে না মিলে তাহলে লুপ ব্রেক করে দিবো। নিচের ছবি দেখ:



5 নম্বর ক্যারেক্টারে গিয়ে আমরা একটি মিসম্যাচ পেয়েছি। ক্রট ফোর্স অ্যালগরিদমের ভিতরের লুপটা 6 নম্বর লাইনে ব্রেক করবে, এবপর ইনডেক্স  $i = 1$  এ গিয়ে আবার খুজতে থাকবে।



প্রোগ্রামিং কনটেন্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৮):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেন্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইভিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

	0	1	2	3	4	5	6	7	8
text	a	b	a	b	a	b	a	c	d
pattern	a	b	a	b	a	c			

চিত্র ১.২

এভাবে text এর প্রতিটি ইনডেক্সে গিয়ে লুপ চালিয়ে প্যাটার্ন খুজতে হবে, এজন্য এই অ্যাপ্রোচের টাইম কমপ্লেক্সিটি  $O(n * m)$ । কোনোভাবে কি আমরা প্রতিটা ইনডেক্সে গিয়ে লুপ চালানো এড়তে পারি? তার আগে আমাদের জানতে হবে সাফিক্স এবং প্রিফিক্স কি।

**প্রিফিক্স:** একটা স্ট্রিং থেকে শূন্য বা তার বেশি সংখ্যক ক্যারেক্টার ফেলে দিলে যা বাকি থাকে সেটাই একটা স্ট্রিং এর প্রিফিক্স। যেমন "ABC" স্ট্রিং টার প্রিফিক্স হলো "A", "AB" এবং "ABC"। এর মধ্যে "A" এবং "AB" হলো **প্রোপার (Proper)** প্রিফিক্স কারণ এগুলো মূল স্ট্রিংটার সমান না।

**সাফিক্স:** একটা স্ট্রিং শুরু থেকে শূন্য বা তার বেশি সংখ্যক ক্যারেক্টার ফেলে দিলে যা বাকি থাকে সেটাই একটা স্ট্রিং এর সাফিক্স। যেমন "ABC" স্ট্রিং টার সাফিক্স হলো "C", "BC" এবং "ABC"। এর মধ্যে "C" এবং "BC" হলো **প্রোপার (Proper)** সাফিক্স কারণ এগুলো মূল স্ট্রিংটার সমান না।

মনে করো আমরা যে প্যাটার্নটা খুজছি সেটা হলো abxyabcd। এবার নিচের (চিত্র ১.৩) ছবিটা দেখ:

????	A	B	X	Y	A	B	?	?	?	????
A	B	X	Y	A	B	C	D			

চিত্র ১.৩

ছবিতে (চিত্র ১.৩) আমরা টেক্সটের সাথে প্যাটার্ন মিলাতে এক জায়গায় মিসম্যাচ পেয়েছি। কোন মিসম্যাচ হওয়া ক্যারেক্টারটা কি অথবা তার পরের ক্যারেক্টারগুলো কি সেটা নিয়ে আপাতত চিন্তা করা দরকার নাই। এখন আমরা যদি ক্রটফোর্সের মতো প্যাটার্নকে একঘর বামে শিফট করে

শ্লাইডিং রেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপার্শন)

অ্যারে কমপ্লেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

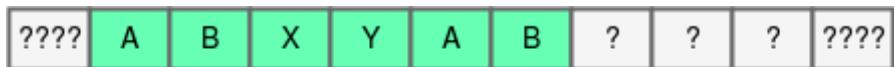
ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

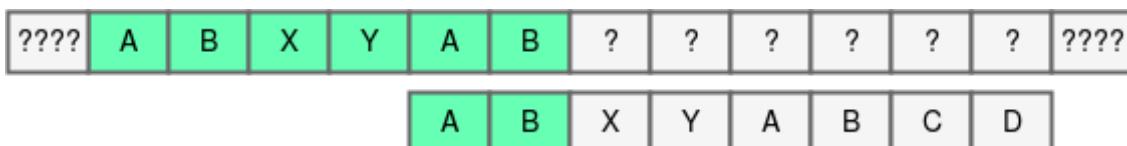
বেলম্যান ফোর্ড

মিলাতে চেষ্টা করি তাহলে আদৌ কি কোন লাভ আছে?



চিত্র ১.৪

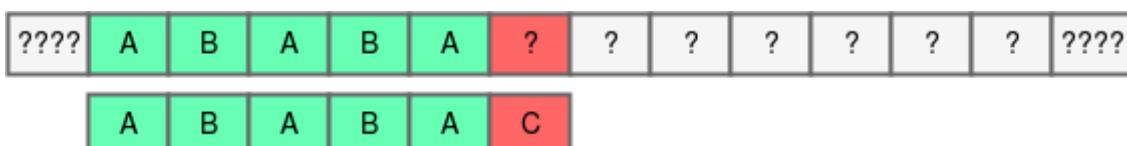
১ ঘর যদি শিফট করি তাহলে প্রশ্নবোধক চিহ্নের জায়গাগুলোয় যাই থাকুক না কেন কোনো লাভ নাই। কত ঘর শিফট করলে লাভ হতেও পারে সেটা কিসের উপর নির্ভর করে? সেটা নির্ভর করে প্যাটার্নের যতটুকু প্রিফিক্স টেক্সটের সাথে ম্যাচ করেছে সেটার উপর, এক্ষেত্রে সেই প্রিফিক্সটা হলো “ABXYAB”। আমরা যদি নিচের মত করে শিফট করি তাহলে ম্যাচ পেলেও পেতে পারি:



চিত্র ১.৫

তারমানে আমাদেরকে প্যাটার্নটা এমনভাবে শিফট করতে হবে যাতে প্যাটার্নের প্রিফিক্সের সাথে প্যাটার্নেরই সাফিক্সের ‘আংশিক’ (partial) ম্যাচিং পাই। তাহলে যেটা ঘটবে, আমরা প্যাটার্নের প্রিফিক্সের সাথে ইনপুট টেক্সটের partial ম্যাচ পাবো এবং এরপর আমরা আবার সামনে গিয়ে ক্যারেক্টার বাই ক্যারেক্টার মিলিয়ে দেখবো পুরো টেক্সটা ম্যাচ করে নাকি।

আরেকটা উদারহণ দেখলে পরিষ্কার হবে। মনে করো এবার প্যাটার্নটা হলো “ABABAC” এবং আমরা নিচের মতো আংশিক ম্যাচিং পেয়েছি:



চিত্র ১.৬

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রাংগুলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুব মিনিমাম ডারটেক্স কভার

ট্রি এবং ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

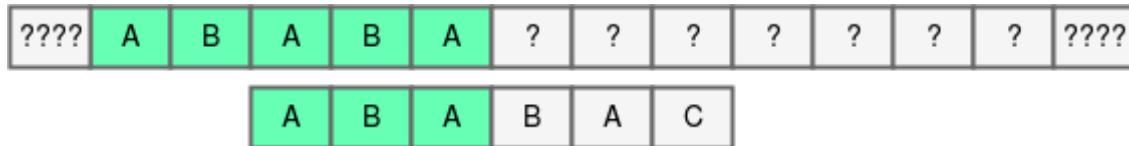
ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঙ্গ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সাবসেট সাম, কম্বিনেটরিক্স, ডিসিশন প্রবলেম)

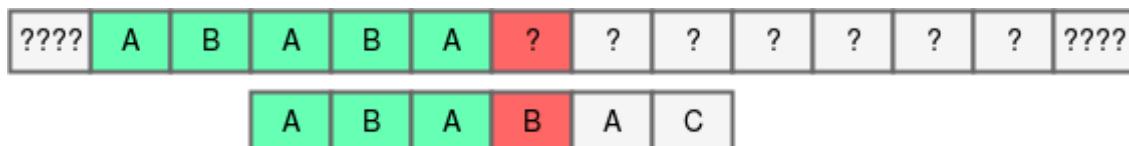
এবার আমরা কটুকু শিফট করবো সেটা নির্ভর করবে ম্যাচ করা প্রিফিক্স "ABABA" এর উপর। নিচের ছবিটা দেখো:



চিত্র ১.৭

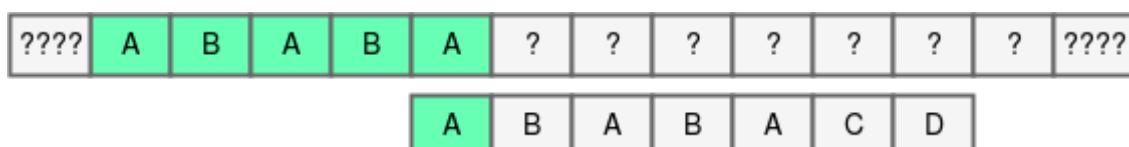
চিত্র ১.৭ এ প্যাটার্নটা ডানে ২ ঘর শিফট করেছি। এতে করে আমরা প্যাটার্নের প্রিফিক্সের সাথে প্যাটার্নেই সাফিক্সের আংশিক ম্যাচ পাবো। এক্ষেত্রে প্যাটার্নের প্রথম ৩ ক্যারেক্টারের সাথে শেষ ৩ ক্যারেক্টার ম্যাচ করছে। তারমানে টেক্স্টের সাথেও প্যাটার্নের প্রথম ৩ ক্যারেক্টার ম্যাচ করবে। এরপর আমরা আবার সামনে গিয়ে বাকি ক্যারেক্টারগুলো মিলিয়ে দেখবো।

এখন ধরো দুর্ভাগ্যক্রমে আমরা আবার মিসম্যাচ পেলাম:



চিত্র ১.৮

এখন কতখানি শিফট করবো? সেটা নির্ভর করে "ABA" এর উপর। আমাদেরকে এমনভাবে ABA কে এমনভাবে ডানে শিফট করতে হবে যেন শিফট করার পর ABA এর প্রিফিক্সের সাথে ABA এর সাফিক্স আংশিক ম্যাচ করে। এক্ষেত্রে চিত্র ১.৯ এর মত করে শিফট করতে হবে:



চিত্র ১.৯

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভারটেক্স কডার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাক্স(১):

ব্যকট্যাক্স বেসিক এবং পারমুটেশন জেনারেটর

নাস্বার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

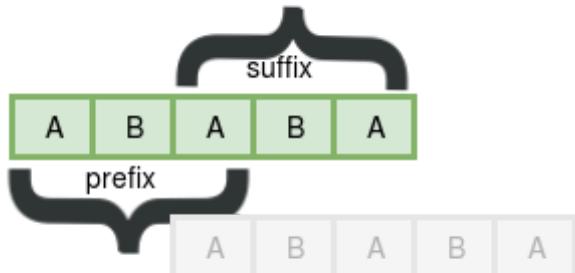
টেইল-কল রিকার্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটার(২)

এতগুলো উদাহরণ দেয়ার উদ্দেশ্য একটা জিনিস পরিষ্কার করা, প্যাটার্নের কতখানি প্রিফিক্স টেক্সটের সাথে ম্যাচ করেছে তার উপর নির্ভর করবে প্যাটার্ন কয়েক শিফট করবো তার উপর নির্ভর করবে প্যাটার্ন কয়েক শিফট করবো।

মনে করো যতখানি প্রিফিক্স ম্যাচ করেছে সেই স্ট্রিংটুকুর নাম  $P$ ।  $P$  কতব্যের শিফট করলে আমরা “যে টেক্সটটুকু অলবেড়ি ম্যাচ করেছে তার সাফিক্সের সাথে” আংশিক ম্যাচ পাবো?

সেটা জানতে আমাদের বের করতে হবে  $P$  স্ট্রিংটার সবথেকে বড় প্রিফিক্স যেটা একই সাথে  $P$  এর একটা সাফিক্স ও। এই লাইনটা শুনলে আমার নিজেরই তালগোল পাকিয়ে যায়, তাই আসো আরেকটা ছবি দেখি:



### Longest prefix that is also a suffix

চিত্র ১.১০

আশা করি ছবি দেখে পরিষ্কার হয়েছে লাইনটির মানে। সর্বোচ্চ কতখানি সাফিক্স প্রিফিক্সের সাথে মিলে যায় সেটা বের করে শিফট করতে হবে। "ABABAC" স্ট্রিং এর প্রিফিক্স আছে ৭টা:

কোয়ান্টাম কম্পিউটার কী?  
কোয়ান্টাম কম্পিউটারের শক্তি এবং সীমাবদ্ধতা

A screenshot of a Facebook post from the page 'Shafaetsplanet'. The post has 2.4K likes. It features a profile picture of a book titled 'জ্ঞান পুরণ' (Gyan Puran) and the text 'শাফায়েত আশরাফ' (Shafayet Ashraff). Below the post is a button to 'Like Page'. The main content of the post is: 'Be the first of your friends to like this'. The post itself reads: 'ডাইনামিক প্রোগ্রামিং এর নতুন সিরিজের ৬ নম্বর লেখায় আমরা দেখবো কম্পিউটেরিঙের প্রবলেম এবং ডিসিপ্লিন প্রবলেম কিভাবে সমাধান করতে হ্য।' At the bottom of the post, there is a link to 'SHAFAEETPLANET.COM' and the text 'ডাইনামিক প্রোগ্রামিং ৬ / ম্যাচ'.

length	prefix
0	""
1	A
2	AB
3	ABA
4	ABAB
5	ABABA
6	ABABAC

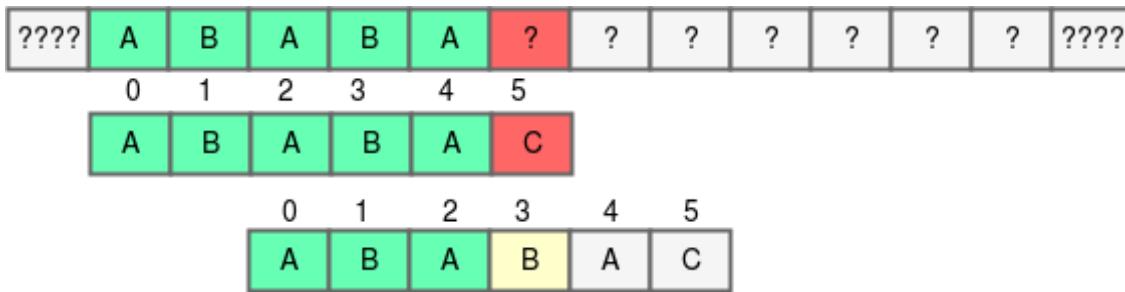
চিত্র ১.১১

এবাব সবগুলো স্ট্রিং এর জন্য সবথেকে বড় প্রিফিক্সের দৈর্ঘ্য বের করবো যেটা একই সাথে একটা সাফিক্স। এক্ষেত্রে যেহেতু আমাদের আংশিক ম্যাচ দরকার, আমরা শুধুমাত্র প্রোপার সাফিক্স ও প্রিফিক্স নিয়ে চিন্তা করবো (অর্থাৎ সাফিক্স/প্রিফিক্সের দৈর্ঘ্য হবে স্ট্রিংটার থেকে কম)।

length	prefix	longest prefix length is also a suffix
0	""	0
1	A	0
2	AB	0
3	ABA	1
4	ABAB	2
5	ABABA	3
6	ABABAC	0

চিত্র ১.১২

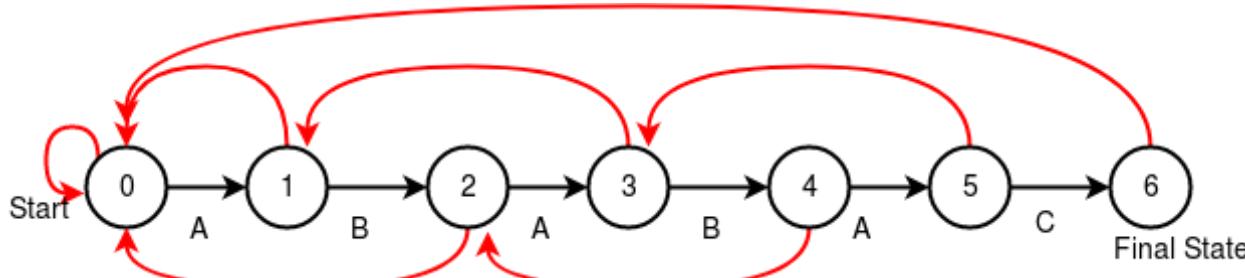
এই টেবিলটির একটা নাম আছে, এটাকে বলা হয় Failure table (ফেইলর টেবিল)। এই টেবিল দেখে আমরা বলে দিতে পারি প্যাটার্নের কতখানি প্রিফিক্স ম্যাচ করার পর মিসম্যাচ পাওয়া গেলে আবার কোথা থেকে ম্যাচিং শুরু করতে হবে।



চিত্র ১.১৩

যেমন উপরের ABABA পর্যন্ত ম্যাচ করার পর একটা ফেইলড ম্যাচ পেয়েছি। চিত্র ১.১২ এর টেবিলের ৬নংর বো থেকে পাই যে প্যাটার্নের প্রথম ৩ ক্যারেক্টার নিয়ে যে প্রিফিক্স হয় সেটা প্যাটার্নের শেষ ৩ক্যারেক্টার নিয়ে যে সাফিক্স হয় তার সমান। তারমানে প্যাটার্নের প্রথম ৩ ক্যারেক্টার "যে টেক্সটকু অলবেডি ম্যাচ করেছে" তার সাফিক্সের সমান। তাহলে আমরা প্যাটার্নের প্রথম ৩ক্যারেক্টার বাদ দিয়ে পরের ক্যারেক্টার থেকে আবার ম্যাচিং শুরু করবো।

তুমি যদি থিওরি অফ কম্পিউটেশনের কোনো কোর্স করে থাকो তাহলে বুঝতে পারছো ফেইলর টেবিলটা আসলে এক ধরনের "ফাইনাইট সেট অটোমাটা" (না পড়ে থাকলেও চিত্রের কিছু নেই)। অটোমেশন হলো একটা "সেট অফ সেট (set of states)" এবং এক সেট থেকে অন্য সেট এ কিভাবে যেতে হবে সেরকম কিছু নিয়ম। আমাদের ক্ষেত্রে সেট হলো প্যাটার্নের কতখানি ম্যাচ করেছে সেট। আর এক সেট থেকে অন্য সেটে কিভাবে যাবো সেট নির্ভর করে টেক্সট এবং প্যাটার্নের পরবর্তী ক্যারেক্টার ম্যাচ করেছে নাকি তার উপর। যদি ম্যাচ করে তাহলেতো সহজ, পরের ক্যারেক্টারে গিয়ে আবার ম্যাচ করার চেষ্টা করবো। আর যদি ম্যাচ না করে তাহলে কতখানি সাফিক্স প্রিফিক্সের সাথে মিলে গিয়েছে (চিত্র ১.১০) সেট বের করবো ফেইলর টেবিল দেখে। সহজে বোধার জন্য আমরা ফেইলর টেবিলটাকে নিচের মত করে আকতে পারি:



চিত্র ১.১৪ এ আমরা ABABAC স্ট্রিং এর জন্য অটোমেশনটাকে দেখতে পাচ্ছি। Start স্টেট হলো যখন কোনো ক্যারেক্টার ম্যাচ করেনি। প্রতিবার একটা করে ক্যারেক্টার ম্যাচ করলে আমরা কালো এজ দিয়ে পরের স্টেটে যাবো, ফাইনাল স্টেটে চলে গেলে কাজ শেষ। লাল তীব্র চিহ্ন দিয়ে দেখানো হয়েছে মিসম্যাচ পেলে কোন স্টেট এ যাবো। চিত্র ১.১২ এর টেবিলের সাথে মিলালে বুঝবে ফেইলর টেবিল আর উপরের গ্রাফটি আসলে একই জিনিস বুঝাচ্ছে।

এখন আমাদের হাতে ২টি সমস্যা, প্রথমটা হলো ফেইলর টেবিলটা তৈরি করা, ২য়টি হলো সেটা ব্যবহার করে ম্যাচিং করা।

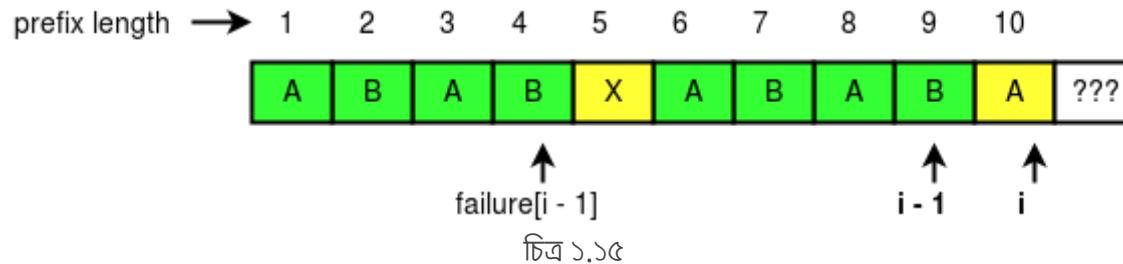
প্রথমে আমরা ফেইলর টেবিলটা তৈরি করি। আমরা এমন একটি ফাংশন লিখবো যেটা *failure[]* নামের m সাইজের একটা অ্যারে তৈরি করবে, "ABABAC" স্ট্রিং এর জন্য অ্যারেটা হবে এরকম:

```
failure[0] = 0
failure[1] = 0
failure[2] = 0
failure[3] = 1
failure[4] = 2
failure[5] = 3
failure[6] = 0
```

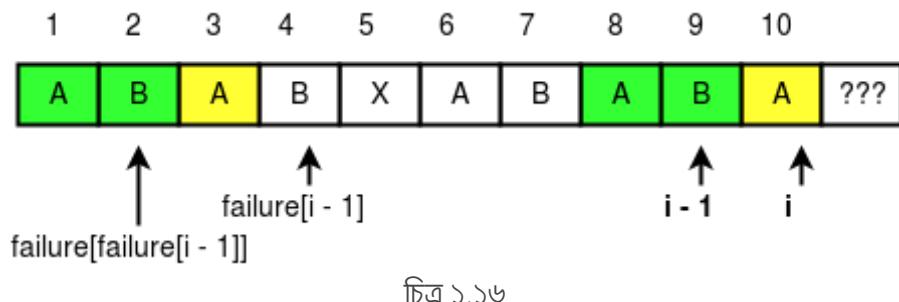
এখানে ইনডেক্সিং নিয়ে একটু সাবধানে থাকতে হবে। *failure[i]* দিয়ে আমরা স্ট্রিং এর i নম্বর ইনডেক্সের কথা বুঝাচ্ছি না, বরং i দৈর্ঘ্যের প্রিফিক্সের কথা বুঝাচ্ছি।

এখন মনে করো i দৈর্ঘ্যের প্রিফিক্সের জন্য *failure[i]* এর মান আমি জানি না কিন্তু  $0 \leq i \leq length$  পর্যন্ত সব দৈর্ঘ্যের জন্য আমি *failure[length]* এর মান আগেই কোনোভাবে বের করে ফেলেছি। এখন আমি দেখবো  $i - 1$  দৈর্ঘ্যের স্ট্রিং এর জন্য কতখানি ম্যাচ পেয়েছি এবং বর্তমানে যে ক্যারেক্টারে আছি সেটা ব্যবহার করে ম্যাচটাকে লম্বা করা যায় নাকি।

চিত্র ১.১৫ এ একটা উদাহরণ দেখানো হয়েছে:



চিত্র ১.১৫ এ আমরা  $failure[10]$  এর মান বের করার চেষ্টা করছি। আমরা আগের ক্যারেটারের গিয়ে  $failure[i - 1]$  এর মান দেখে বুঝবো কতটুকু প্রিফিক্স-সাফিক্স অলরেডি ম্যাচ করেছে। এরপর চেষ্টা করবো পরের ক্যারেটারের সাথে বর্তমান ক্যারেটারকে মিলানোর (চিত্র ১.১৫ এ হলুদ রঙ এর ঘর)। যদি মিলে যেত তাহলে  $failure[10]$  এর মান হতো  $failure[9] + 1 = 4 + 1 = 5$ । কিন্তু যেহেতু মিলেনি, আমরা পরবর্তি সেবা ম্যাটিং এ চলে যাবো এবং সেটা পাবো  $failure[failure[i - 1]]$  এ:



এবাব হলুদ ঘরের ক্যারেটার দুটো মিলে গেছে, তারমানে  $i = 10$  এর জন্য “ABA” সাফিক্স এবং প্রিফিক্স ম্যাচ করে যাব দৈর্ঘ্য 3, তাই  $failure[10]$  এর মান হবে 3।

আমাদের বেস কেস হবে,  $failure[0] = failure[1] = 0$ । বাকি  $2 <= i <= length$  এর জন্য  $failure[i]$  মান আমরা আগেরগুলো দেখে দেখে বের করতে পারবো। নিচের কোডটা দেখো:

1 //<https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-string-search/>

```

2 #define MAX 100000
3 int failure[MAX];
4
5 void build_failure_function(string pattern, int m) {
6     failure[0] = 0
7     failure[1] = 0; //base case
8
9     for(int i = 2; i <= m; i++) { //i is length of the prefix we are dealing with
10        // j is the index of the largest next partial match
11        // (the largest suffix/prefix) of the string under index i - 1
12        int j = failure[i - 1];
13        while(true) {
14            // check if the last character of prefix of length i "expands" the current "candidate"
15            if(pattern[j] == pattern[i - 1]) {
16                failure[i] = j + 1;
17                break;
18            }
19            // if we cannot "expand" even the empty string
20            if(j == 0) {
21                failure[i] = 0;
22                break;
23            }
24            // else go to the next best "candidate" partial match
25            j = failure[j];
26        }
27    }
28 }

```

ফেইলর টেবিল জেনারেট করা হয়ে গেলে আমরা স্ট্রিং ম্যাচিং শুরু করতে পারি। কোড লেখার আগে তুমি একবার অটোমেশন গ্রাফটা একে হাতে কলমে সিমুলেট করো। আমাদের দুইটা পয়েন্টার থাকবে  $i$  এবং  $j$ ।  $i$  দিয়ে বুঝাবো আমরা অটোমেশনের কোন স্টেট এ আছি, অর্থাৎ কতখানি প্রিফিক্স অলবেড়ি ম্যাচ করেছে এবং  $j$  দিয়ে বুঝাবো টেক্সটের কোন ক্যারেক্টারটার সাথে ম্যাচ করছি। যদি  $text[j] == pattern[i]$  হ্য তাহলে আমরা  $i$  এবং  $j$  এর মান ইনক্রিমেন্ট করে আবার ম্যাচ করার চেষ্টা করবো, অর্থাৎ অটোমেশনের কালো তীব্রচিহ্ন ধরে আগবো (চিত্র ১.১৪)। কিন্তু যদি  $text[j]! = pattern[i]$  হ্য তাহলে আমরা ফেইলর টেবিল ব্যবহার করে যতটুকু সাফিক্স-প্রিফিক্স ম্যাচ করেছে সেখান থেকে আবার ম্যাচিং শুরু করবো, অর্থাৎ  $i = failure[i]$  হ্যে যাবে। আর যদি দেখি  $i = 0$  হ্যে গেছে কিন্তু আমরা এখনোও ম্যাচ করতে পারছি না তাহলে  $j$  এর মান ইনক্রিমেন্ট করে টেক্সটের পরের ক্যারেক্টার থেকে আবার ম্যাচিং শুরু করতে হবে।

```

1 bool kmp(string text, string pattern)
2 {
3     int n = text.size();
4     int m = pattern.size();
5     build_failure_function(pattern, m);
6
7     int i = 0; // the initial state of the automaton is
8         // the empty string
9
10    int j = 0; // the first character of the text
11
12    while(true) {
13        if(j == n) {
14            return false; //reached the end of the text
15        }
16
17        // character matched
18        if(text[j] == pattern[i]) {
19            i++; // change the state of the automaton
20            j++; // get the next character from the text
21            if(i == m) {
22                return true;
23            }
24        } else {
25            if (i == 0) {
26                // if we reached the empty string and failed to
27                // "expand" even it; we go to the next
28                // character from the text, the state of the
29                // automaton remains zero
30                j++;
31            }
32            else {
33                //we try to "expand" the next best (largest) match
34                i = failure[i];
35            }
36        }
37    }
38    return false;
39 }

```

কেএমপি অ্যালগরিদমের টাইম কমপ্লেক্সিটি  $O(n + m)$ । লিনিয়ার টাইম স্ট্রিং ম্যাচিং করার আরেকটা অ্যালগরিদম **রবিন-কাপ্র** নিয়ে আগে লিখেছিলাম। কিন্তু হ্যাপ কলিশনের জন্য রবিন-কাপ্রের পারফরমেন্স অনেক ক্ষেত্রেই খারাপ হয়ে যেতে পারে, বেশিভাগ ক্ষেত্রেই কেএমপি ব্যবহার করে ম্যাচিং করা সুবিধাজনক।

প্রোগ্রামিং কনটেক্টে সর্বাসরি কেএমপি ব্যবহার করে সমাধান করতে হয় এমন সমস্যা খুব বেশি পাবে না, কিন্তু ফেইলের ফাংশনের প্রোপার্টি ব্যবহার করে সমাধান করতে হয় এমন সমস্যা প্রায়ই পাওয়া যায়। পরবর্তি কোনো একটা লেখায় সেরকম কিছু সমস্যা নিয়ে আলোচনা করবো। আপাতত এই পর্যন্তই, হ্যাপি কোডিং!

প্র্যাকটিস প্রবলেম:

[http://www.lightoj.com/volume_showproblem.php?problem=1255](http://www.lightoj.com/volume_showproblem.php?problem=1255)

[http://www.lightoj.com/volume_showproblem.php?problem=1258](http://www.lightoj.com/volume_showproblem.php?problem=1258)

রিসোর্স:

<https://www.topcoder.com/community/data-science/data-science-tutorials/introduction-to-string-searching-algorithms/>

ফেসবুকে মন্তব্য

3 comments

# ডিরেকশন অ্যারে

জানুয়ারি ২৫, ২০১৩ by [শাফায়েত](#)



অনেক সময়ই এমন প্রবলেম থাকে যেখানে বলা হয় তুমি একটা ২-ডি অ্যারের কোনো এক পজিশনে আছো, সেখান থেকে তুমি উপরে-নিচে-বামে-ডানে যেতে পারবে। অথবা দাবার বোর্ডে একটা ঘোড়া আছে, তাকে ৮টা দিকে মুভ করানো যায়, এখন কোনো একটা পজিশনে শর্টেস্ট পাথে যেতে হবে। বিগিনার কোডারো এ ধরণের প্রবলেমে ছোট একটা ট্রিকস না জানার কারণে কোডের সাইজ বিশাল বানিয়ে ফেলে।

ডিরেকশন অ্যারের ট্রিকসটা যারা যানেনা এ ধরণের প্রবলেমে তাদের কোড হয় অনেকটা এরকম:

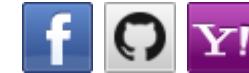
```

1 int x=5,y=3,row=5,col=5,nx,ny;
2 nx=x+1; ny=y;
3 if(nx>=1 && nx<=row && ny>=1 && ny<=col)
4 {
5     DO SOMETHING
6 }
7 nx=x-1; ny=y;
8 if(nx>=1 && nx<=row && ny>=1 && ny<=col)
9 {
10    DO SOMETHING
11 }
12 nx=x; ny=y+1;
13 .....
14 .....
15 .....

```

এভাবে বারবার একই লাইন লিখতে কোডের চেহারা ভয়াবহ হয়ে যায়, আর কাওকে কোড দেখতে দিলে তারও পাগল হবার অবস্থা হয়! ৮ ডিরেকশনে মুভ করা গেলেতো কথাই নেই। সবথেকে বড় সমস্যা হলো এক জায়গায় চেঙ্গ করলে সবজায়গায় চেঙ্গ করতে হয়।

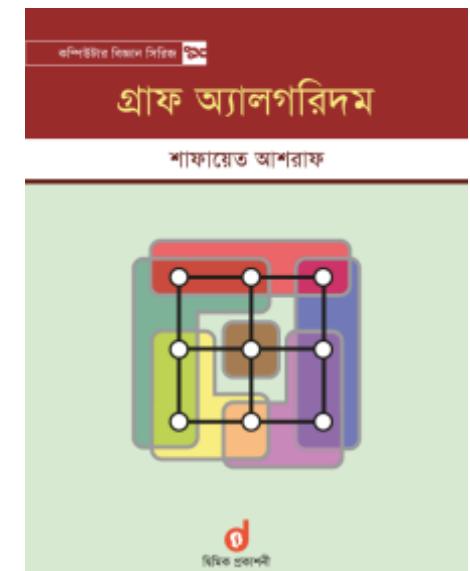
## সাবস্ক্রাইব



Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



একটা improvement হতে পারে if(nx>=1 && nx<=row && ny>=1 && ny<=col) এই লাইনের কমিশনটা একটা ম্যাক্রো বানিয়ে ফেলা। যেমন:

```
1 #define valid(nx,ny) nx>=1 && nx<=row && ny>=1 && ny<=col
```

এখন if(valid) লিখলেই হচ্ছে। এবপরে DO SOMETHING অংশের কাজগুলোও একটা ফাংশন বানিয়ে ফেললে ঘামেলা কিছুটা কমে, এখন খালি ম্যাক্রো বা ফাংশনে চেঞ্চ করলে সব জায়গায় চেঞ্চ হয়ে যাবে। তারপরেও বার বার কমিশন চেক বা ফাংশন কল করতে হচ্ছে আমাদের। এজন্য আমরা ব্যবহার করবো ডিরেকশন অ্যারে। 8 দিকে মুড করা যায় এটার অর্থ হলো:

- “
- ১. current row এর সাথে ১ যোগ এবং current col এর সাথে ০ যোগ
  - ২. current row এর সাথে -১ যোগ এবং current col এর সাথে ০ যোগ
  - ৩. current row এর সাথে ০ যোগ এবং current col এর সাথে ১ যোগ
  - ৪. current row এর সাথে ০ যোগ এবং current col এর সাথে -১ যোগ

তারমানে x,y পজিশন থেকে উপরে যেতে হলে (x,y)+(1,0) করতে হবে, নিচে যেতে (x,y)+(-1,0) করতে হবে, একই ভাবে ডানে-বামে যেতে যোগ করতে হবে শুধু y এর সাথে।



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে  
হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

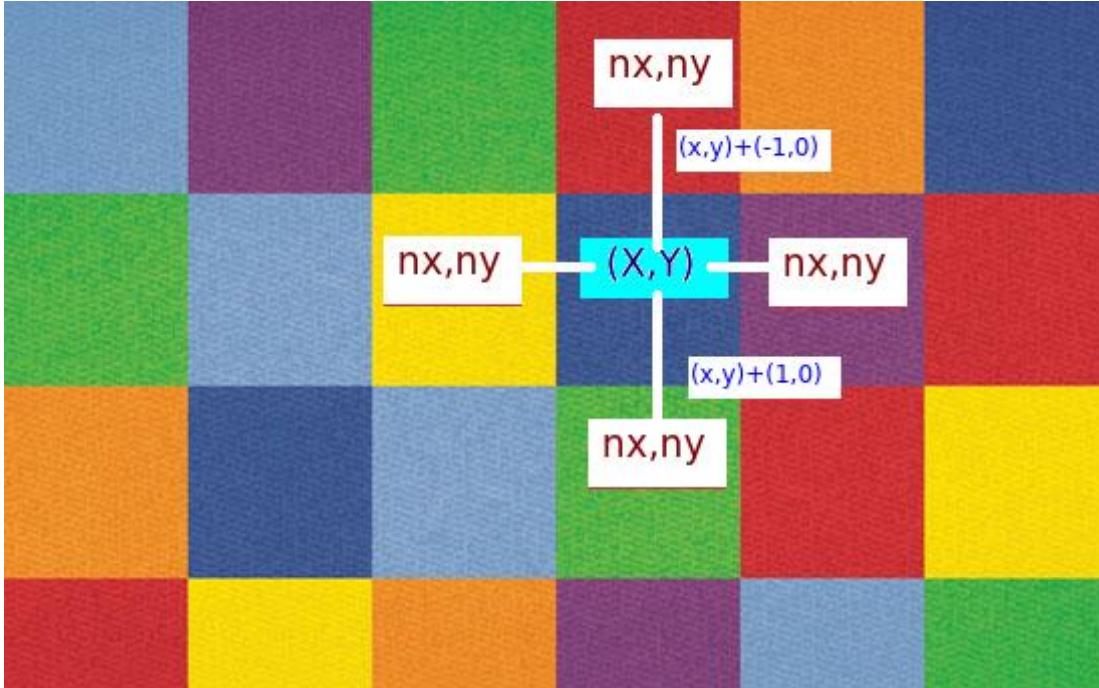
ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ



আমরা দুটি অ্যারে ডিক্লেয়ার করি এভাবে:

```
1 int fx[]={+1,-1,+0,+0};
2 int fy[]={+0,+0,+1,-1};
```

`fx[]` দিয়ে বুঝাচ্ছি `row` এর সাথে কত যোগ করবো এবং `fy` দিয়ে বুঝাচ্ছি `y` এর সাথে কত যোগ করবো। এবার কাজ খুব সহজ হয়ে গেলো:

```
1 #define valid(nx,ny) nx>=1 && nx<=row && ny>=1 && ny<=col
2 int x=5,y=3,row=5,col=5,nx,ny;
3 for(int k=0;k<4;k++)
4 {
5     int nx=x+fx[k]; //Add fx[k] with current row
6     int ny=y+fy[k]; //Add fy[k] with current col
7     if(valid(nx,ny)
8     {
9         DO SOMETHING;
10    }
11 }
```

শ্লাইডিং রেজে মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল ট্রি/বেডিল ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সেড ট্রি

স্থায়ার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথ ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল সর্ট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

তুমি 8 দিকে যেতে চাইলে অ্যারেটা হবে এরকম:

```
1 int fx[]={+0,+0,+1,-1,-1,+1,-1,+1};  
2 int fy[]={-1,+1,+0,+1,+1,-1,-1};
```

একটু চিন্তা করলেই তুমি দাবার ঘোড়ার মুড়ের জন্যেও ডিরেকশন অ্যারে লিখতে পারবে। 3-ডি তেও এটা কাজ করবে, তখন `fx[]` নামের আরেকটা অ্যারে লাগবে।

তুমি যদি সম্পূর্ণ কোড চাও তাহলে এই বিএফএস এর কোডটা দেখতে পাবো:

```
1 //Problem link:http://acm.timus.ru/problem.aspx?space=1&num=1145  
2 #define rep(i,n) for(int i=0; i<(int)n; i++)  
3 #define pb(x) push_back(x)  
4 #define mem(x,y) memset(x,y,sizeof(x));  
5 #define pii pair<int,int>  
6 #define pmp make_pair  
7 #define uu first  
8 #define vv second  
9 using namespace std;  
10 #define READ(f) freopen(f, "r", stdin)  
11 #define WRITE(f) freopen(f, "w", stdout)  
12  
13 int fx[]={1,-1,0,0};  
14 int fy[]={0,0,1,-1};  
15 int mr,mc,mx=0;  
16 char w[1000][1000];  
17 int d[1000][1000];  
18 int r,c;  
19 void bfs(int x,int y,int dep)  
20 {  
21     mem(d,63);  
22     d[x][y]=0;  
23     queue<pii>q;  
24     q.push(pii(x,y));  
25     while(!q.empty())  
26     {  
27         pii top=q.front(); q.pop();  
28         if(d[top.uu][top.vv]>mx)  
29         {  
30             mx=d[top.uu][top.vv];  
31         }
```

আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

স্ট্রংলি কানেক্টেড কম্পোনেন্ট

ম্যাঞ্চিমাম ফ্লো-১

ম্যাঞ্চিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটাৰ নিৰ্ণয়

লংগেষ্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্রেষ্ঠ পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিস্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেষ্ট কমন সার্ভিসিকোয়েস)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১ ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সার্ভিসেট সাম, কম্পিউটেটরিঙ্গ, ডিসিশন প্রবলেম)

```

32         mx=d[top.uu][top.vv];
33         mr=top.uu;
34         mc=top.vv;
35     }
36     rep(k,4)
37     {
38         int tx=top.uu+fx[k];
39         int ty=top.vv+fy[k];
40         if(tx>=0 and tx<r and ty>=0 and ty<c and w[tx][ty]=='..' and d[top.uu][top.vv]+1<d[tx][ty])
41         {
42             d[tx][ty]=d[top.uu][top.vv]+1;
43             q.push(pii(tx,ty));
44         }
45     }
46 }
47 }
48 int main(){
49 // READ("in");
50     int sx,sy,cc=0;
51     cin>>r>>c;
52     swap(r,c);
53     rep(i,r)
54     cin>>w[i];
55     rep(i,r)
56         rep(j,c)
57         if(w[i][j]=='..') {sx=i;sy=j;cc++;}
58     bfs(sx,sy,0);
59
60     mx=0;
61     bfs(mr,mc,0);
62     if(cc==1) mx=0;
63     cout<<mx<<endl;
64     return 0;
65 }

```

এই কোডটা শুধু ডিরেকশন অ্যারের ব্যবহার দেখানোর জন্য, প্রবলেমটার সলিউশন বের করার কাজ তোমার, **তু এর ডায়ামিটাৰ** বের করতে বলা হয়েছে প্রবলেমটায়।

হ্যাপি কোডিং!

ফেসবুকে মন্তব্য

[পুরানো সিরিজ]

বিটমাস্ক ডিপি

মিনিমাম ভাৰটেক্স কভার(গ্রাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যাকট্র্যাকিং(১):

ব্যক্ট্র্যাকিং বেসিক এবং পারমুটেশন জেনারেটর

নাষ্ঠার থিওরি/গণিত(৫):

মডুলার অ্যারিথমেটিক

প্রাইম জেনারেটর (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিরেঞ্জমেন্ট

প্রোবাবিলিটি: এক্সপেক্টেড ড্যালু

স্ট্রিং ম্যাচিং(২):

রবিন-কার্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিরেকশন অ্যারে

মিট ইন দ্য মিডল

টেইল-কল রিকাৰ্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটাৰ(২)

# মিট ইন দ্য মিডল টেকনিক

ফেব্রুয়ারি ২, ২০১৪ by শাফায়েত



মিট ইন দ্য মিডল খুবই এলিগেন্ট একটা প্রবলেম সলভিং টেকনিক। এটার কাজ হলো প্রবলেমটাকে ঠিক দুইভাগে ভাগ করে ফেলে তারপর সেই দুইভাগকে কোনোভাবে মার্জ করে প্রবলেমটা সলভ করা। তবে ডিভাইড এন্ড কনকোয়ারের সাথে এটার পার্থক্য হলো ডিভাইড এন্ড কনকোয়ারে দুই ভাগে ভাগ করার পর ছোট ভাগগুলোকে বারবার ভাগ করা হয়, মিট ইন দ্য মিডলে আমরা শুধু একবার ভাগ করবো।

আমরা কিছু প্রবলেম দেখবো যেগুলোকে মিট ইন দ্য মিডলের সাহায্যে সলভ করা সম্ভব।

## প্রবলেম ১: সাম অফ ফোর

(দুরকারি নলেজ: বাইনারি সার্চ)

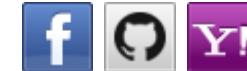
তোমাকে  $8$ টা  $n$  সাইজের অ্যারে  $A, B, C, D$  দেয়া আছে। প্রতিটা অ্যারে থেকে এক্স্যালি একটা করে ভ্যালু সিলেক্ট করতে হবে যেন তাদের যোগফল  $0$  হয়।

যেমন  $n = 3$  এবং  $A=\{1,2,3\}$ ,  $B=\{-1,-4,-5\}$ ,  $C=\{1,5,8\}$ ,  $D=\{9,8,5\}$  হলে  $1 + (-4) + 8 - 5 = 0$  হতে পারে একটা সলিউশন।

## সলিউশন-১

প্রথমেই মাথায় যে সলিউশন আসে সেটা হলো  $8$ টা নেস্টেড লুপ চালিয়ে সব কম্বিনেশনে সংখ্যাগুলোকে যোগ করে দেখা। এটার কমপ্লেক্সিটি হবে  $O(n^4)$ । যদি  $n = 1000$  হয় তাহলে প্রোগ্রামটা শেষ হতে কয়েক ঘণ্টা লেগে যাবে।

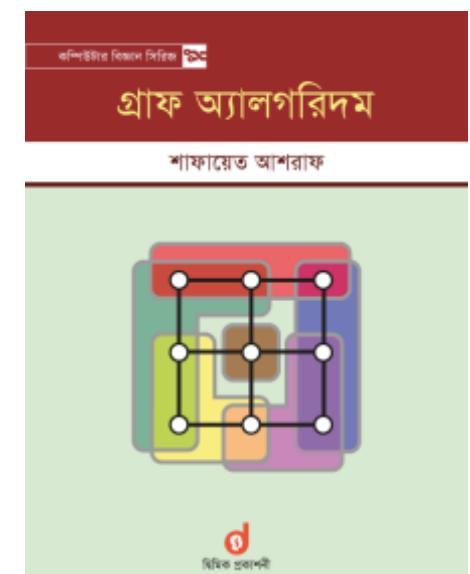
## সাবস্ক্রাইব



Secured by OneAll Social Login

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @  
**Traveloka Singapore** (বিস্তারিত...)



আমাদের ৪টা মান  $a, b, c, d$  দরকার যাতে  $a + b + c + d = 0$  হয়। এটাকে লেখা যায়,  $(a + b) = -(c + d)$ ।  $a$  ভ্যালুটা পাবো A অ্যাবে থেকে,  $b$  পাবো B অ্যাবে,  $c$  পাবো C অ্যাবে থেকে এবং  $d$  পাবো D অ্যাবে থেকে।

এখন দুটি নেক্সেড লুপ চালিয়ে A এবং B অ্যাবে দিয়ে যতগুলো কম্পিউটেশন বানানো যায় সবগুলো বের করে ফেলি। তাহলে আমরা সবগুলো  $(a+b)$  পেয়ে গেলাম। সবগুলো ভ্যালুকে সর্ট করে রাখো বা ম্যাপে ইনসার্ট করে রাখো।

আবার দুটি নেক্সেড লুপ চালিয়ে C এবং D অ্যাবে দিয়ে যতগুলো কম্পিউটেশন বানানো যায় সবগুলো বের করি। এখন পেলাম সবগুলো  $c+d$ । প্রতিটা  $c+d$  এর জন্য  $a+b$  খুজে বের করো আগের সর্টেড ভ্যালুতে বাইনারি সার্চ চালিয়ে বা ম্যাপের মধ্যে খুজে।

এখন আমাদের কমপ্লেক্সিটি হলো  $O(n^2 * \log n^2)$ । যদি ভ্যালুগুলো ছোটো হয় তাহলে ম্যাপের জায়গায় সাধারণ বুলিয়ান ফ্ল্যাগ ব্যবহার করে  $O(n^2)$  এ সলভ করা সম্ভব।

## প্রবলেম ২: কয়েন চেঙ্গ

(দরকারি নলেজ: বাইনারি সার্চ, ব্যাকট্র্যাকিং অথবা বিটমাস্ক ব্যবহার করে সবগুলো সাবসেট জেনারেশন)

ধৰো তোমার কাছে কিছু কয়েন আছে। বলতে হবে এগুলো থেকে কিছু কয়েন ব্যবহার করে নির্দিষ্ট একটা ভ্যালু তৈরি করা যায় নাকি। কোনো কয়েন একবাবের বেশি ব্যবহার করা যাবেনা। যেমন কয়েনগুলো যদি হয় ১, ৩, ৬, ১০ তাহলে তুমি ১৩ বা ১১ বানাতে পারবে কিন্তু ৫০ বা ২ কিছুতেই বানাতে পারবেনা। কয়েনের সংখ্যা সর্বোচ্চ ৩০টা।

## সলিউশন

মনে করি কয়েনগুলার মান হতে পারে ১ থেকে ১০০ পর্যন্ত। তুমি যদি ডাইনামিক প্রোগ্রামিং জানো তাহলে নিশ্চয়ই ডাবছো এটাতে খুব সহজ প্রবলেম, সবগুলো কয়েনের ভ্যালুর যোগফল সর্বোচ্চ হতে পারে  $30 * 100$ , তাহলে  $30 * (30 * 100)$  কমপ্লেক্সিটিতে খুব সহজে প্রবলেমটা সলভ করা যাবে। এবার আমি প্রবলেমটাকে কঠিন করে দেই, ধৰি কয়েনগুলোর ভ্যালু হতে পারে ১ থেকে  $10^{18}$ । এখন কমপ্লেক্সিটি হয়ে গেলো  $30 * (30 * 10^{18})$ । এটা ডাইনামিক প্রোগ্রামিং দিয়ে সলভ করা সম্ভবনা, মেমরি বা টাইম কোনোটাতেই কুলিয়ে উঠবেনা। এখন কি করা যেতে পারে? এখানে লক্ষ্য করার বিষয় হলো কয়েনের সংখ্যা অনেক কম!



প্রোগ্রামিং কনটেক্ট এবং অ্যালগরিদম

অনুপ্রেরণা(৪):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামার

অ্যালগরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইল্ডিং অ্যালগরিদম

ডাটা স্ট্রাকচার(১৪):

লিংকড লিস্ট

স্ট্যাক

কিউ+সার্কুলার কিউ

কয়েনের সংখ্যা মাত্র ৩০ হলেও ৩০টা কয়েনের একটা সেটের সাবসেট সংখ্যা ২৪৩০। তারমানে সর্বোচ্চ ২৪৩০টা ভিন্ন ভ্যালু তৈরি করা যাবে কয়েনগুলো দিয়ে। তাই ব্যাকট্র্যাকিং করে সবগুলো ভ্যালু জেনারেট করা পসিবল না। কিন্তু কয়েন যদি ১৫টা হতো তাহলে  $2^{15} = 32768$  টা সাবসেট থাকতো এবং সবগুলো ভ্যালু আমরা জেনারেট করতে পারতাম ব্যাকট্র্যাক করো! এই পর্যন্ত পড়ার পর একটু খেমে কিছুক্ষণ ছিন্ন করো কিভাবে প্রবলেমটা সলভ করা সম্ভব। এরপরে নিচের অংশ পড়ো।

আমরা কয়েনগুলোকে দুই ভাগে ভাগ করে ফেলি। প্রতিটা ভাগে আছে ১৫টা করে কয়েন। এখন প্রথম ভাগের ১৫টা কয়েন নিয়ে সবগুলো ভ্যালু জেনারেট করে একটা অ্যারেতে রেখে দাও, মনে করি অ্যারেটার নাম A। একই ভাবে ২য় ১৫টা কয়েন নিয়ে সবগুলো ভ্যালু জেনারেট করে আবেকটা অ্যারেতে রেখে দাও, মনে করি অ্যারেটার নাম B। B অ্যারেটাকে সর্ট করে ফেলো, A কে সর্ট করার দরকার নেই।

এখন বাকি কাজ সহজ। মনে করো তোমাকে X ভ্যালুটা বানাতে হবে। A অ্যারের উপর লুপ চালিয়ে সবগুলো ভ্যালু চেক করো। তাম ভ্যালু যদি হয় A[i] তাহলে তুমি চেক করো B অ্যারেতে X-A[i] ভ্যালুটা আছে নাকি। যদি থাকে তাহলে তুমি X বানাতে পারবে!! B তুমি সর্ট করে রেখেছো, তাহলে বাইনারি সার্চ করেই X-A[i] আছে নাকি চেক করতে পারবে। একটু ভাবলেই বুঝতে পারবে A অ্যারেতে প্রথম ১৫টাৰ সবৰকম কম্বিনেশন আছে, তাই X-A[i] কে A তে খোজার কোনো দরকার নাই।

n টা কয়েনের জন্য তাহলে কমপ্লেক্সিটি হবে  $O(2^{(n/2)} * \log 2^{(n/2)})$ , কারণ আমরা  $2^{(n/2)}$  টা ভ্যালুর জন্য বাকি অর্ধেকের উপর বাইনারি সার্চ করছি।

### প্রবলেম ৩: বাইডি঱েকশনাল সার্চ

(দরকারি নলেজ: গ্রাফ থিওরি)

একটা গ্রাফে দুটি নোড দেয়া আছে, নোড দুটির মধ্যে শর্টেস্ট পাথ বের করতে হবে।

বাইডি঱েকশনাল সার্চ (ডিকিটু)

ডিসজয়েট সেট(ইউনিয়ন ফাইল্ড)

ট্রাই(প্রিফিক্স ট্রি/বেডিক্স ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

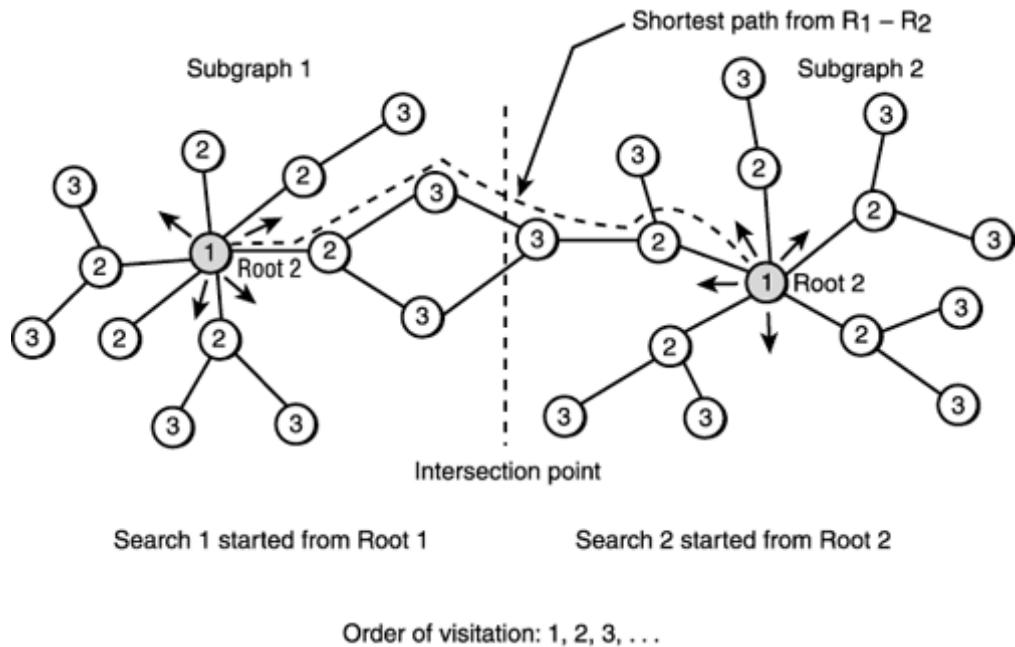
টপোলজিকাল সর্ট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল  
সর্ট

ডায়াক্রস্ট্রো

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড



## সলিউশন:

মোটামুটি নিশ্চিত গ্রাফ থিওরি যারা পারো তাৰা মাথা চুলকিয়ে ভাবছো, এটা কোনো প্রবলেম হলো? বিএফএস দিয়েইতো সলভ কৱা যায়। কিন্তু এখনে আমৰা চিন্তা কৰছি বিশাল গ্রাফের কথা যেখানে বিলিয়ন বিলিয়ন নোড আৰ এজ থাকতে পাৰে।

বিএফএস এ প্রতিটা নোড থেকে তাৰ অ্যাডজেসেন্ট নোডগুলোতে যেতে হয়, সেগুলো থেকে আবাৰ তাৰ অ্যাডজেসেন্ট নোডগুলোতে যেতে হয়। প্রতিটা নোডের অ্যাডজেসেন্ট নোড সংখ্যাকে বলা হয় নোডটা ডিগ্রী। এখন যদি প্রতিটা নোডের এভাৱেজ ডিগ্রী হয়  $p$  আৰ শট্টেস্ট পাথ যদি হয়  $k$  তাহলে তোমাকে মোটামুটি  $O(p^k)$  টা নোড এক্সপ্লোৱ কৰতে হবে।

যদি তুমি সোৰ্স আৰ ডেস্টিনেশন দুই পাশ থেকে সাৰ্চ শুৰু কৱো যতক্ষণনা তাৰা একসাথে মিলছে তাহলে কমপ্লেক্সিটি হয়ে যাবে  $O(p^{k/2})$ । আগেৱ  
কয়েন চেঞ্জ প্রবলেম সলভ কৰেই বুঝতে পাৱছো এই অধৰ্মী হওয়াটা এক্সপোনেনশিয়াল কমপ্লেক্সিটিৰ ক্ষেত্ৰে এটা খুবই সিগনিফিকেন্ট উন্নতি।

গ্রাফ রিলেটেড আৰো দুটি ইন্টাৰেস্টিং প্রবলেম হলো:

১. ফেসবুকেৰ গ্রাফে দুজন ইউজাৱেৰ মধ্যে মিউচুয়াল ফ্ৰেন্ড আছে নাকি বেৰ কৰতে হবে

আর্টিকুলেশন পয়েন্ট এৰং ব্ৰিজ

স্ট্ৰাংলি কানেক্টেড কম্পোনেন্ট

ম্যাক্রিমাম ফ্লো-১

ম্যাক্রিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লাৱ ট্ৰ্যুৱ মিনিমাম ডারটেক্স কডাব

ট্ৰি এৰ ডায়ামিটাৰ নিৰ্ণয়

লংগেষ্ট পাথ প্রবলেম

অ্যালগৱিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্ৰোগ্ৰামিং(৮):

ডাইনামিক প্ৰোগ্ৰামিং ১ (ফিবোনাচি)

ডাইনামিক প্ৰোগ্ৰামিং ২ (শট্টেস্ট পাথ)

ডাইনামিক প্ৰোগ্ৰামিং ৩ (LIS এৰং পাথ প্ৰিন্টিং)

ডাইনামিক প্ৰোগ্ৰামিং ৪ (লংগেষ্ট কমন  
সাৰসিকোয়েম)

ডাইনামিক প্ৰোগ্ৰামিং ৫ (কয়েন চেঞ্জ/০-১  
ন্যাপস্যাক)

ডাইনামিক প্ৰোগ্ৰামিং ৬ (সাৰসেষ্ট সাম,  
কম্পিনেটোৱিঙ্গ, ডিসিশন প্রবলেম)

২. ধারণা করা হয় সোস্যাল নেটওয়ার্কে যেকোনো দুজন মানুষের দূরস্থ সর্বোচ্চ ৬টি নোড। দুটি ইউজারনেম ইনপুট দিলে কিভাবে এটা ডেরিফাঈ করবে?

**রেফারেন্স:** [ইলফো এরিনা](#)

**কৃতজ্ঞতা:** মাহবুবুল হাসান শাস্ত্র ভাইকে ধন্যবাদ সাম অফ ফোর প্রবলেমের সলিউশনে ডুল ধরিয়ে দেয়ার জন্য।

সলভ করার জন্য প্রবলেম:

[Coin Change\(IV\)](#)

[Sum of Four](#)

[Funny Knapsack](#)

কোন প্রশ্ন থাকলে কমন্টের ঘরে জানাও বা যোগাযোগ করো: shafaet[dot]csedu@gmail[dot]com। অনুরোধ করছি যথেষ্ট চিন্তাভাবনা আর চেষ্টা না করে কোনো কোড ডিবাগ করতে পাঠাবেন।

হ্যাপি কোডিং!

ফেসবুকে মন্তব্য

0 comments

# টেইল-কল রিকার্শন অপটিমাইজেশন

এপ্রিল ২৮, ২০১৯ by শাফায়েত



রিকার্শন ব্যবহার করতে গেলে অনেক সময় দেখা যায় রিকার্শনের গভীরতা অনেক বেশি হয় গেলে মেমরি শেষ হয়ে যায় এবং কোড কৃত্য করবে।

আমরা আজকে একটি অপটিমাইজেশন টেকনিক শিখবো যেটা ব্যবহার করে কোনো কোনো সময় রিকার্শনের মেমরি ব্যবহার অনেক কমিয়ে আনা সম্ভব।

প্রথমে জানা দরকার টেইল-কল (Tail call) জিনিসটা কি। সহজভাবে বলতে গেলে, টেইল-কল হলো একটা ফাংশনের শেষ লাইন। অর্থাৎ যে কাজটি করার পর ফাংশনের আর কোনো কাজ থাকে না সেটাই টেইল-কল। যেমন নিজের ফাংশনটি দেখ:

```

1 int mod = 1000007;
2
3 int fact(int n) {
4
5     int ans = 1;
6     for (int i = 1; i <= n; i++) {
7         ans = (ans * i) % mod;
8     }
9
10    return ans;
11 }
```

এটা ফ্যাক্টরিয়েল বের করার একটি ফাংশন। এই ফাংশনে `return ans` এর পর ফাংশনটির আর কোনো কাজ নেই, তাই এটাই হলো টেইল-কল।

**mod কেন ব্যবহার করছি?** কারণ আমরা অনেক বড়  $n$  ব্যবহার করবো কিন্তু আমরা চাইনা ইন্টিজার ওভারফ্লো হোক।

## সাবস্ক্রাইব

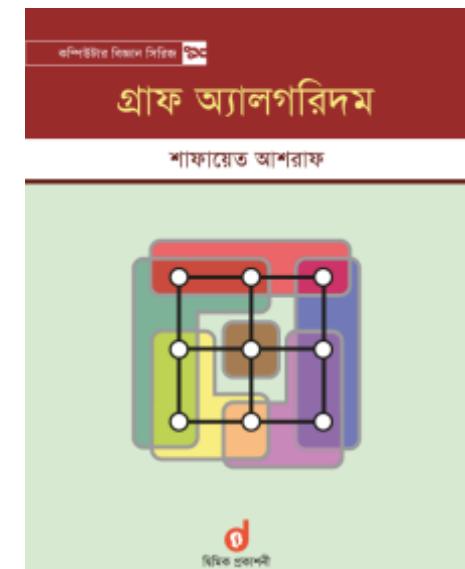


Secured by [OneAll Social Login](#)

## আমার সম্পর্কে

শাফায়েত, সফটওয়্যার ইঞ্জিনিয়ার @

**Traveloka Singapore** (বিস্তারিত...)



এখন এই ফাংশনটাকেই আমরা রিকার্সন ব্যবহার করে লিখবো:

```
1 int mod = 1000007;
2
3 int fact(int n) {
4     if (n == 0) {
5         return 1;
6     }
7
8     int ret = (n * fact(n - 1) % mod);
9 }
```

একটা রিকার্সিভ ফাংশনকে টেইল-রিকার্সন বলা হয় যদি রিকার্সিভ কল করা টাই হয় এই ফাংশনের শেষ কাজ। উপরের ফাংশনটির শেষ কাজ হলো  $n * fact(n - 1)$  এর মান রিটার্ন করা। তারমানে শেষ কাজ হলো রিকার্সিভলি  $fact(n - 1)$  কে কল করা? তুমি যদি রিকার্সন কিভাবে কাজ করে সেটা বুঝে থাকো তাহলে এতক্ষণে বুঝে গেছো যে এটা শেষ কাজ না! ফাংশনটি প্রথমে  $fact(n - 1)$  এর মান রিকার্সিভলি বের করে আনবে, এরপর সেটাকে  $n$  দিয়ে গুণ করবে। আমরা চাইলে নিচের মতো করে লিখতে পারি:

```
1 int mod = 1000007;
2
3 int fact(int n) {
4     if (n == 0) {
5         return 1;
6     }
7
8     int ret = fact(n - 1);
9     return (n * ret) % mod;
10 }
```

একই ফাংশন, শুধু শেষ লাইনটা একটু ভেঙে লিখেছি। এখান থেকে বোঝা যাচ্ছে এই ফাংশনটা টেইল-রিকার্সন না, শেষ লাইনে আমরা রিকার্সিভ ফাংশন কল করছি না, বরং আগের লাইনে রিকার্সিভ ফাংশন কল করে শেষ লাইনে সেটা ব্যবহার করে ক্যালকুলেশন করছি।

এখন তুমি এই ফাংশনটা ব্যবহার করে  $fact(1000000)$  এর মান প্রিন্ট করা চেষ্টা করো (আমি ধরে নিচ্ছি তুমি কোনো সেটিং পরিবর্তন করে মেমরি বাড়িয়ে নিচ্ছে না)। আমি যখন  $fact(1000000)$  রান করলাম, সাথে সাথে প্রোগ্রাম কর্য্যাপ করলো আর দেখালো segmentation fault। তারমানে প্রোগ্রামটি অনেক বেশি মেমরি ব্যবহার করেছে এবং কর্য্যাপ করেছে। তুমি জিঞ্জেস করতে পারো আমিতো কোনো অ্যারে ব্যবহার করিনি, কিভাবে এই কোড এত বেশি মেমরি খরচ করে ফেললো?



প্রোগ্রামিং কনটেক্ট এবং অ্যালগোরিদম

অনুপ্রেরণা(8):

কেন আমি প্রোগ্রামিং শিখবো?

কম্পিউটার বিজ্ঞান কেন পড়বো?

প্রোগ্রামিং কনটেক্ট এবং অনলাইন জাজে

হাতেখড়ি

কনফিউজড প্রোগ্রামাৰ

অ্যালগোরিদম বেসিক(৬):

বিগ "O" নোটেশন

কমপ্লেক্সিটি স্লাস(P-NP, টুরিং মেশিন ইত্যাদি)

হাল্টিং প্রবলেম

বাইনারি সার্চ - ১

বাইনারি সার্চ - ২(বাইসেকশন)

ফ্লয়েড সাইকেল ফাইন্ডিং অ্যালগোরিদম

ডাটা স্টোকচার(১৪):

লিংকড লিস্ট

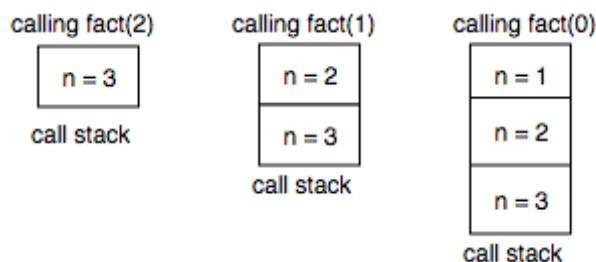
স্ট্যাক

কিউ+সার্কুলার কিউ

প্রতিটি ফাংশন একটা মেমরি স্পেসে রান করে। তুমি যখন রিকার্সিভলি একই ফাংশন কল করো তখন একই মেমরি স্পেসে নতুন প্যারামিটার ব্যবহার করে ফাংশনটা আবার রান করে এবং ফিরে আসার পর পুরানো প্যারামিটার ব্যবহার করে বাকি কাজ শেষ করে।

ধরো তুমি *fact(3)* বের করতে চাও। লাইন ৮ এ গিয়ে কোডটা একই ফাংশন কল করবে  $n = 2$  ব্যবহার করে। তারমানে একই মেমরি স্পেসে নতুন প্যারামিটার  $n = 2$  ব্যবহার করে ফাংশনটা এখন রান করবে। এরপর একসময় ফিরে এসে লাইন ৯ এ বাকি ক্যালকুলেশন করবে পুরানো প্যারামিটার ব্যবহার করে। যেহেতু একই মেমরি স্পেস ব্যবহার হচ্ছে, ফিরে আসার পর প্রসেসটা কিভাবে জানবে যে  $n$  এর মান আগে কত ছিলো?

এটা জানতে ইন্টার্নালি একটা স্ট্যাক ব্যবহার করা হয়। যতবার তুমি রিকার্সিভ কল করো, স্ট্যাকে পুরানো প্যারামিটার গুলো সেভ করে রাখা হয়।  
নিচের ছবি দেখো:



যখন *fact(3)* থেকে *fact(2)* কে কল করা হচ্ছে তখন  $n = 3$  কে স্ট্যাকে সেভ করে রাখা হচ্ছে। ঠিক সেভাবে  $n = 1$  কল করার সময়  $n = 2$  কে স্ট্যাকে সেভ করে রাখা হচ্ছে। এভাবে সেভ করে রাখার কারণে রিকার্সিভ কল শেষ করে ফিরে আসার সময় স্ট্যাকের উপরে আগের ফাংশন কলের প্যারামিটার গুলো খুজে পাওয়া যায় সহজেই। (বিস্তারিত জানতে স্ট্যাক নিয়ে আমার [এই লেখাটা](#) পড়তে পারো।)

তুমি যখন *fact(n)* কল করছো তখন স্ট্যাকে এভাবে করে  $n$  টি এন্ট্রি সেভ করে রাখতে হচ্ছে। সি++ এ রিকার্সন কর মেমরি ব্যবহার করতে পারবে তার একটি লিমিট আছে।  $n$  এর মান যখন অনেক বড় তখন মেমরির ব্যবহার লিমিটের বাইরে চলে যাচ্ছে আর কোড কর্যাপ করছে।

আমরা যদি এই মেমরি ব্যবহার কোনোভাবে কমাতে পারি তাহলে কোড কর্যাপ করবে না। বর্তমানের কম্পাইলারগুলো খুবই বৃদ্ধিমান, সে যদি দেখে যে অতিরিক্ত মেমরি ব্যবহার দরকার নেই তাহলে সে এমনভাবে অপটিমাইজড মেশিন কোড জেনারেট করে যাতে মেমরি ব্যবহার কম হয়। আমরা সেই বৃদ্ধিটাই কাজে লাগাবো।

শ্লাইডিং বেঞ্জ মিনিমাম কুয়েরি (ডিকিউ)

ডিসজয়েট সেট(ইউনিয়ন ফাইল)

ট্রাই(প্রিফিল্ট ট্রি/বেডিল্ফ ট্রি)

সেগমেন্ট ট্রি-১

সেগমেন্ট ট্রি-২(লেজি প্রপাগেশন)

অ্যারে কমপ্রেশন/ম্যাপিং

লোয়েস্ট কমন অ্যানসেক্টর

বাইনারি ইনডেক্সড ট্রি

স্ট্যার-রুট ডিকম্পোজিশন ক্যাশিং অ্যালগরিদম:

এল-আর-ইউ ক্যাশ(**নতুন**)

ব্লুম ফিল্টার(**নতুন**)

গ্রাফ থিওরি(২০):

গ্রাফ থিওরিতে হাতেখড়ি

অ্যাডজেসেন্সি ম্যাট্রিক্স

অ্যাডজেসেন্সি লিস্ট

ব্রেথড ফার্স্ট সার্চ (বিএফএস)

মিনিমাম স্প্যানিং ট্রি ১ (প্রিমস অ্যালগোরিদম)

মিনিমাম স্প্যানিং ট্রি ২ (ক্রুসকাল অ্যালগোরিদম)

টপোলজিকাল সেট

ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলজিকাল

সেট

ডায়াক্রস্টা

ফ্লয়েড ওয়ার্শল

বেলম্যান ফোর্ড

আগের কোডে আমাদেরকে পুরানো প্যারামিটারগুলো স্ট্যাকে সেভ করতে হয়েছে কারণ ফিরে আসার পর সেগুলো কাজে লাগবে। আমরা কি এমনভাবে কোডটা লিখতে পারি যাতে ফিরে আসার পর পুরানো প্যারামিটার আর কোনো দরকার নেই? এখানেই টেইল-কল এর আইডিয়া কাজে লাগবে। আমরা এমন ভাবে কোডটা লিখবো যাতে রিকার্সিভ ফাংশন কল করার পর আর কোনো কাজ করা না লাগে, তাহলে স্ট্যাকে  $n$  এর মানগুলো সেভ করে রাখা দরকার হবে না।

```
1 int mod = 100007;
2
3 int fact_tail(int n, int result) {
4     if (n == 0) {
5         return result;
6     }
7
8     return fact_tail(n - 1, (n * result) % mod);
9 }
10
11 int fact_optimized(int n) {
12     return fact_tail(n, 1);
13 }
```

এবার আমরা প্রতিবার রিকার্সিভ কল শেষ হবার পর বেজাল্ট ক্যালকুলেশন না করে আমরা প্রতিবার আংশিক একটা বেজাল্ট ক্যালকুলেট করে সেটাকে প্যারামিটার হিসাবে পাঠিয়ে দিচ্ছি। একদম শেষে গিয়ে ওই প্যারামিটারেই আমরা ফলাফল পেয়ে যাবো।

```
1 fact(4, 1)
2 = fact(3, 4 * 1)
3 = fact(2, 4 * 3)
4 = fact(1, 4 * 3 * 2)
5 = fact(0, 4 * 3 * 2 * 1)
6 = fact(0, 24)
```

এইটা একটা টেইল রিকার্শন, কারণ শেষ লাইনে রিকার্সিভ ফাংশন কল করার পরে আর কোনো কাজ করা হ্য নি। এই কোড যখন কম্পাইল করা হবে তখন কম্পাইলার দেখবে স্ট্যাক ব্যবহার করে পুরানো মান সেভ করে রাখার কোনো দরকার নেই। তাই এই কোডে ততটুকুই মেমরি লাগবে যতখানি লাগতো সাধারণ লুপ ব্যবহার করলে!

আধুনিক যেকোন ল্যাংগুয়েজের কম্পাইলার এই অপটিমাইজেশনটা করতে পাবে। তুমি যদি সি/সি++ ব্যবহার করো তাহলে প্রোগ্রাম বান করার আগে তুমি দেখো -O2 কম্পাইলার অপটিমাইজেশন চালু করা আছে নাকি, না থাকলে আগে চালু করে দাও, তুমি যে কোড এডিটর ব্যবহার

আর্টিকুলেশন পয়েন্ট এবং বিজ

স্ট্রাংলি কানেক্টেড কম্পানেন্ট

ম্যাক্সিমাম ফ্লো-১

ম্যাক্সিমাম ফ্লো-২

স্টেবল ম্যারেজ প্রবলেম

অয়লার ট্যুর মিনিমাম ডারটেক্স কভার

ট্রি এর ডায়ামিটার নির্ণয়

লংগেস্ট পাথ প্রবলেম

অ্যালগরিদম গেম থিওরি(৩):

গেম থিওরি-১

গেম থিওরি-২

গেম থিওরি-৩

ডাইনামিক প্রোগ্রামিং(৮):

ডাইনামিক প্রোগ্রামিং ১ (ফিবোনাচি)

ডাইনামিক প্রোগ্রামিং ২ (শ্টেচ্স্ট পাথ)

ডাইনামিক প্রোগ্রামিং ৩ (LIS এবং পাথ প্রিন্টিং)

ডাইনামিক প্রোগ্রামিং ৪ (লংগেস্ট কমন  
সার্সিকোয়েম)

ডাইনামিক প্রোগ্রামিং ৫ (কয়েন চেঞ্জ/০-১  
ন্যাপস্যাক)

ডাইনামিক প্রোগ্রামিং ৬ (সারসেট সাম,  
কম্পিউটেরিক্স, ডিসিশন প্রবলেম)

করছো সেখানে চালু কৰাৰ অপশন থাকবে। অথবা তুমি সৱাসিৰ টাৰ্মিনাল থেকেও কম্পাইল কৰতে পাৰো -O2 অপশন ব্যবহাৰ কৰে। কিভাবে কৰতে হয় গুগলে সাৰ্চ দিয়ে দেখে নাও। (প্ৰোগ্ৰামিং কনটেক্টে জাজ মেশিনে সাধাৰণত এই অপটিমাইজেশন চালু কৰা থাকে)

এবাৰ fact_optimized(1000000) বান কৰে দেখো সুন্দৰ আউটপুট দিচ্ছে।

এটাই হলো রিকাৰ্শনেৰ টেইল-কল অপটিমাইজেশন।

## এক্সাৰসাইজ ১

নিচেৰ কোডটা একট অ্যাৰেৰ সবগুলো সংখ্যাৰ যোগফল বেৰ কৰে:

```
1 int find_sum(int * numbers, int index) {
2     if (index == 0) {
3         return numbers[0];
4     }
5
6     return numbers[index] + find_sum(numbers, index - 1);
7 }
8
9 int main() {
10    int numbers[] = {5, 10, 15, 20};
11    int n = 4;
12    cout<<find_sum(numbers, n - 1)<<endl;
13 }
```

এই কোডটাকে রিকাৰ্শিভলি আগেৰ সংখ্যাগুলোৰ যোগফল বেৰ কৰে বৰ্তমান সংখ্যাৰ সাথে যোগ কৰা হচ্ছে। তোমাৰ কাজ হবে টেইল-কল রিকাৰ্শন ব্যবহাৰ কৰা যাতে কৰে রিকাৰ্শিভ ফাংশন কল কৰাৰ পৰ আৱ কোনো ক্যালকুলেশন কৰা না লাগে।

## এক্সাৰসাইজ ২

ফিবোনাচি সিৱিজ হলো এমন একটি সিৱিজ যাৰ প্ৰতিটি সংখ্যা সিৱিজেৰ আগেৰ দুটি সংখ্যাৰ যোগফল। সিৱিজেৰ প্ৰথম দুটি সংখ্যা হলো 0 আৱ 1 এবং সিৱিজেৰ প্ৰথম কয়েকটি সংখ্যা হলো 0, 1, 1, 2, 3, 5, 8, 13, 21 . . . .। একে নিচেৰ ফাংশন দিয়ে প্ৰকাশ কৰা যায়:

[পুৱানো সিৱিজ]

বিটমাস্ক ডিপি

মিনিমাম ভাৱটেক্স কভাৰ(গ্ৰাফ+ডিপি)

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন

ব্যকট্যাকিং(১):

ব্যকট্যাকিং বেসিক এবং পাৰমুটেশন জেনাৰেটোৱ

নাষ্ঠাৰ থিওৰি/গণিত(৫):

মডুলাৰ অ্যাৰিথমেটিক

প্ৰাইম জেনাৰেটোৱ (Sieve of Eratosthenes)

বিটওয়াইজ সিভ

ডিৱেজমেন্ট

প্ৰোবাবিলিটি: এক্সপেন্সিভ ড্যালু

স্ট্ৰিং ম্যাচিং(২):

ৱিন-কাৰ্প

কেএমপি (KMP)

অন্যান্য(৩) :

ডিৱেকশন অ্যাৰে

মিট ইন দ্যা মিডল

টেইল-কল রিকাৰ্শন অপটিমাইজেশন(নতুন)

কোয়ান্টাম কম্পিউটাৱ(২)

```
1 fib(0) = 0
2 fib(1) = 1
3 fib(n) = fib(n - 1) + fib(n - 2) where n > 1
```

সাধাৰণ ভাৱে বিকার্সিত ফাংশন লিখলে সেটা টেইল-কল রিকাৰ্শন হবে না কাৰণ সেটা পথমে  $fib(n-1)$  এৰ মান বেৰ কৰে ফিৰে এসে আবাৰ  $fib(n-2)$  বেৰ কৰবে। তোমাৰ কাজ  $n$  তম ফিবোনাচি বেৰ কৰাৰ জন্য টেইল-কল রিকাৰ্শন লেখা।

## ফেসবুকে মন্তব্য

0 comments