সূচিপত্র

অধ্যায় ১

ভূমিকা

You can being your chapters with this quote box :D

– Me

গণিত সম্পর্কিত কোনো বিষয়ের কিছু লিখতে গেলে ল্যাটেকের কোনো বিকল্প নেই। তবে ল্যাটেক দিয়ে বাংলায় সরাসরি কিছু লিখতে গেলে তেমন ভালো সাপোর্ট পাওয়া যায় না। সেই সমস্যাকে ট্যাকেল করতে গণিত অলিম্পিয়াডের আদীব হাসানের বানানো ল্যাটেকবাংলা প্যাকেজটি অত্যন্ত গুরুত্বপূর্ণ। পরবর্তীতে যাওয়াদ আহমেদ চৌধুরী ও এম আহসান আল মাহীর সেই প্যাকেজটিকে তাদের বইয়ে ব্যবহারের জন্য আরো কিছু ফিচার যুক্ত করেছেন।

এই টেমপ্লেট এ প্রায় সব environment ডিফাইন করা আছে। সেগুলোর টাইটেল বাংলায় আসবে। যেমন

সমস্যা ১.১: এটি একটি সমস্যা

এছাড়াও আর কিছু environment বানানো আছে, সেগুলো environments.sty ফাইলে পাওয়া যাবে।

অধ্যায় ২

ম্যাট্রিক্স এক্সপোনেন্সিয়েশন

§ ২.১ শুরুর কথা

নামটা শুনতে কঠিন মনে হলেও ম্যাট্রিক্স এক্সপোনেন্সিয়েশন আসলে তেমন কঠিন কিছু না। ম্যাট্রিক্স সম্পর্কে কমবেশি সবারই জানা থাকার কথা। তারপরেও যারা এ সম্পর্কে জানো না তারা ম্যাট্রিক্সকে 2D অ্যারের মত চিন্তা করতে পার। বাইরে থেকে দুটি একইরকমই দেখতে। যদি কোন ম্যাট্রিক্সের n টি সারি আর m টি কলাম থাকে তাহলে ম্যাট্রিক্সটিকে $n \times m$ ম্যাট্রিক্স বলা হয়। যেমন নিচের ম্যাট্রিক্সটি একটি 2×3 ম্যাট্রিক্স।

$$\begin{pmatrix} 1 & 3 & 2 \\ 9 & 0 & 7 \end{pmatrix}$$

ঠিক অ্যারের মতই কোন ম্যাট্রিক্স A এর i তম সারির j তম সংখ্যাকে $A_{i,j}$ দিয়ে প্রকাশ করা হয়। যেমন উপরের ম্যাট্রিক্সের জন্য $A_{1,1} = 1$, আবার $A_{2,3} = 7$। ম্যাট্রিক্সের যোগ, বিয়োগও সম্ভব, তবে তুমি একটি $n \times m$ ম্যাট্রিক্সের সাথে আরেকটি $n \times m$ ম্যাট্রিক্সই যোগ বা বিয়োগ করতে পারবে। এক্ষেত্রে A এবং B যোগ করে C পাওয়া গেলে $C_{i,j} = A_{i,j} + B_{i,j}$ হতে হবে। যেমন

$$\begin{pmatrix} 1 & 3 \\ 9 & 0 \end{pmatrix} + \begin{pmatrix} 2 & -1 \\ 3 & 1 \end{pmatrix} = \begin{pmatrix} 1+2 & 3-1 \\ 9+3 & 0+1 \end{pmatrix}$$

তবে সবচেয়ে অদ্ভুত হচ্ছে ম্যাট্রিক্সের গুন। গুনের ক্ষেত্রে একটি $n \times m$ ম্যাট্রিক্সের সাথে কেবল একটা $m \times k$ ম্যাট্রিক্স গুন করতে পারবে এবং গুণফল

হবে একটা $n \times k$ ম্যাট্রিক্স। অর্থাৎ প্রথম ম্যাট্রিক্সের কলাম সংখ্যা আর দ্বিতীয় ম্যাট্রিক্সের সারি সংখ্যা সমান হতে হবে। C যদি A এবং B ম্যাট্রিক্সের গুণফল হয় তাহলে

$$C_{i,j} = \sum_{x=1}^{m} A_{i,x} \times B_{x,j}$$

যেমন ধর,

$$\begin{pmatrix} 1 & 3 & 2 \\ 9 & 0 & 7 \end{pmatrix} \begin{pmatrix} 5 & 6 & 0 & 3 \\ 0 & 2 & -1 & 1 \\ 1 & 1 & 4 & -1 \end{pmatrix} = \begin{pmatrix} 5 & 6 & 7 & 8 \\ 9 & 10 & 12 & 13 \end{pmatrix}$$

এখানে 2×3 ম্যাট্রিক্সের সাথে 3×4 ম্যাট্রিক্স গুন করে 2×4 ম্যাট্রিক্স পাওয়া গিয়েছে। তবে গুণফলটা আসলে কীভাবে বের হল সেটা বুঝতে একটু ছোট উদাহরণ দেখা যাক। নিচের ২টি 2×2 ম্যাট্রিক্সের গুণ করা যাক

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} p & q \\ r & s \end{pmatrix} = \begin{pmatrix} ap + br & aq + bs \\ cp + dr & cq + ds \end{pmatrix}$$

$C_{2,1}$ এর কথা ধর। প্রথম ম্যাট্রিক্সের ২য় সারির সংখ্যাগুলো হচ্ছে c এবং d, আবার দ্বিতীয় ম্যাট্রিক্সের ১ম কলামের সংখ্যাগুলো হচ্ছে p এবং r। তাই c এর সাথে p গুন করেছি আর d এর সাথে q গুন করেছি, এরপর গুণফল দুটিকে যোগ করে দিয়েছি। এজন্যই $C_{2,1}$ এর মান $cp + dr$। অন্য পদগুলোও এভাবেই বের করা যাবে। (তোমরা হয়ত ভাবছ এমন অদ্ভুত ভাবে ম্যাট্রিক্স গুন করা হয় কেন। এর উত্তর জানতে লিনিয়ার আলজেব্রা পড়তে হবে। চাইলে 3blue1brown এর ভিডিও সিরিজটি দেখতে পারো)।

ম্যাট্রিক্স গুণফলের সবচেয়ে চমদপ্রদক দিক হল অ্যাসোসিয়েটিভিটি। যেমন ধর তুমি তিনটি ম্যাট্রিক্স A, B, C গুন করতে চাও, অর্থাৎ ABC এর মান বের করতে চাও। তাহলে তুমি AB এর সাথে C কে গুন করলে যে ম্যাট্রিক্স পাওয়া যাবে, A এর সাথে BC কে গুন করলে একই ম্যাট্রিক্স পাওয়া যাবে। সহজ ভাষায় $A(BC) = (AB)C$। সোজা কথায় আমরা যেভাবেই ব্রাকেট বসাই

না কেন একই উত্তর আসবে। এই বৈশিষ্ট্য আমাদের পরে কাজে লাগবে। তবে সাবধান! AB কিন্তু কখনই BA এর সমান নয়। কোনটিকে আগে কোনটিকে পরে গুন করতে হবে তা লক্ষ্য রাখতে হবে।

§ ২.২ ডাইনামিক প্রোগ্রামিং এর সাথে সম্পর্ক

আবার ফিবোনাচ্চি সমস্যায় ফেরত যাওয়া যাক। রিকারেন্সটি নিশ্চয় মনে আছে,

$$\begin{aligned} f_0 &= 0 \\ f_1 &= 1 \\ f_n &= f_{n-1} + f_{n-2} \end{aligned}$$

আমরা এমন একটি 2×2 ম্যাট্রিক্স A বের করতে চাই যেন,

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix} \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} f_{n+1} \\ f_n \end{pmatrix}$$

অর্থাৎ f_n ও f_{n-1} এর ভেক্টরের ($n \times 1$ ম্যাট্রিক্স গুলোকে ভেক্টর বলা হয়) সাথে এমন একটি ম্যাট্রিক্স গুন করতে যেন f_{n+1} ও f_n এর ভেক্টর পাওয়া যায়। কাজটা কিন্তু খুব কঠিন না। একটু চেষ্টা করলেই বুঝবে $A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$ ম্যাট্রিক্সটি কাজ করে

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = \begin{pmatrix} 1f_n + 1f_{n-1} \\ 1f_n + 0f_{n-1} \end{pmatrix} = \begin{pmatrix} f_{n+1} \\ f_n \end{pmatrix}$$

এখন লক্ষ্য কর, A ম্যাট্রিক্সটি যদি দুইবার গুন করি তাহলে কিন্তু $\begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix}$ থেকেই $\begin{pmatrix} f_{n+2} \\ f_{n+1} \end{pmatrix}$ পেয়ে যাবো। কারণ

$$A \times A \times \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = A \times \begin{pmatrix} f_{n+1} \\ f_n \end{pmatrix} = \begin{pmatrix} f_{n+2} \\ f_{n+1} \end{pmatrix}$$

লক্ষ্য কর এখানে আমরা ম্যাট্রিক্সের অ্যাসোসিয়েটিভিটি ধর্মটি ব্যবহার করেছি। আবার যদি আমরা দুইবারের বদলে m বার A ম্যাট্রিক্সটি গুন করতাম, তাহলে একইভাবে আমরা পাব

$$A^m \begin{pmatrix} f_n \\ f_{n-1} \end{pmatrix} = A^{m-1} \begin{pmatrix} f_{n+1} \\ f_n \end{pmatrix} = \cdots = \begin{pmatrix} f_{n+m} \\ f_{n+m-1} \end{pmatrix}$$

উপরের সমীকরণে $n = 1$ বসালে আমরা পাব

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^m \begin{pmatrix} f_1 \\ f_0 \end{pmatrix} = \begin{pmatrix} f_{m+1} \\ f_m \end{pmatrix}$$

তোমরা হয়ত ভাবছ, এত কিছু বের করে আসলে কী লাভ হল। আমরা শুরুতে যখন n তম ফিবোনাচ্চি নাম্বার বের করা শিখেছিলাম সেটার কমপ্লেক্সিটি ছিল $\mathcal{O}(n)$। কিন্তু ম্যাট্রিক্স এক্সপনেন্সিয়েশন দিয়ে আমরা কাজটা $\mathcal{O}(\log n)$ এই করে ফেলতে পারি। কারণ দেখ, n তম ফিবনাচ্চি নাম্বার বের করতে আমাদের A^n কে ফাস্ট ক্যালকুলেট করতে হবে। এজন্য কিন্তু আমরা সংখ্যার ক্ষেত্রে a^b যেভাবে বাইনারি এক্সপনেন্সিয়েশন দিয়ে বের করি সেভাবেই কাজটা করে ফেলতে পারি। অর্থাৎ n জোড় হলে প্রথমে $A^{\frac{n}{2}}$ বের করে তাকে বর্গ করে দিলেই হচ্ছে। আবার n বিজোড় হলে প্রথমে A^{n-1} বের করে তার সাথে A গুন করে দিলেই হচ্ছে। এভাবে আমাদের $\mathcal{O}(\log n)$ বার দুটি 2×2 ম্যাট্রিক্স গুন করতে হচ্ছে। দুটি 2×2 ম্যাট্রিক্স গুন করার কমপ্লেক্সিটি আমরা $\mathcal{O}(1)$ ই ধরতে পারি। তাই সবমিলিয়ে কমপ্লেক্সিটি হবে $\mathcal{O}(\log n)$।

তবে একটা জিনিশ বলে রাখা দরকার। এখানে ম্যাট্রিক্স এর আকার অনেক ছোট বলে আমরা দুটি ম্যাট্রিক্স গুন করার কমপ্লেক্সিটি $\mathcal{O}(1)$ ধরেছি। কিন্তু অনেক ক্ষেত্রে বেশ বড় ম্যাট্রিক্স লাগতে পারে (যেমন ধর 50×50 ম্যাট্রিক্স)। সেক্ষেত্রে কিন্তু ম্যাট্রিক্স গুন করার কমপ্লেক্সিটি $\mathcal{O}(1)$ ধরলে হবে না। খেয়াল করলে দেখবে দুটি $k \times k$ ম্যাট্রিক্স গুন করতে আমাদের $\mathcal{O}(k^3)$ কমপ্লেক্সিটি প্রয়োজন। সেক্ষেত্রে আমাদের ম্যাট্রিক্স এক্সপনেন্সিয়েশনের কমপ্লেক্সিটি হবে $\mathcal{O}(k^3 \log n)$

§ ২.৩ আরো কিছু উদাহরণ

আরেকটা উদাহরণ দেখা যাক। ধর এবার আমাদের রিকারেন্সটি হল

$$
\begin{aligned}
f_0 &= 0 \\
f_1 &= 2 \\
f_2 &= 1 \\
f_n &= 2f_{n-1} + 3f_{n-2} - 7f_{n-3}
\end{aligned}
$$

যেহেতু f_n আগের তিনটি পদের ওপর নির্ভরশীল, তাই আমাদের এবার একটি 3×3 ম্যাট্রিক্স খুঁজতে হবে। ফিবোনাচ্চির ম্যাট্রিক্স তা যদি বুঝে থাক তাহলে এটা বের করাও তেমন কঠিন না। নিচের ম্যাট্রিক্সটা দেখ

$$
\begin{pmatrix} 2 & 3 & -7 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix}
\begin{pmatrix} f_n \\ f_{n-1} \\ f_{n-2} \end{pmatrix}
=
\begin{pmatrix} 2f_n + 3f_{n-1} - 7f_{n-2} \\ 1f_n + 0f_{n-1} + 0f_{n-2} \\ 0f_n + 1f_{n-1} + 0f_{n-2} \end{pmatrix}
=
\begin{pmatrix} f_{n+1} \\ f_n \\ f_{n-1} \end{pmatrix}
$$

এবার একটু জটিল উদাহরণ চেষ্টা করা যাক। ধর এবার আমাদের কাছে ২ টি রিকারেন্স আছে।

$$
\begin{aligned}
f_n &= 2f_{n-1} + g_{n-2} \\
g_n &= g_{n-1} + 3f_{n-2}
\end{aligned}
$$

ধরে নাও f_0, f_1, g_0, g_1 এর মান জানা আছে। এবার আমাদের ভেক্টরে কিন্তু শুধু f_n, f_{n-1} রাখলে চলবে না, বরং g_n, g_{n-1} এর মানও রাখতে হবে। যদি এটা ধরতে পারো তাহলে আগেরগুলোর মতই এটাও সমাধান করা যায়

$$
\begin{pmatrix} 2 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 3 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}
\begin{pmatrix} f_n \\ f_{n-1} \\ g_n \\ g_{n-1} \end{pmatrix}
=
\begin{pmatrix} 2f_n + g_{n-1} \\ f_n \\ 3f_{n-1} + g_n \\ g_n \end{pmatrix}
=
\begin{pmatrix} f_{n+1} \\ f_n \\ g_{n+1} \\ g_n \end{pmatrix}
$$

সমস্যা ২.১: নিচের রিকারেন্সটির জন্য ম্যাট্রিক্স বের কর।

$$
\begin{aligned}
f_0 &= 0\\
f_1 &= 1\\
f_n &= f_{n-1} + f_{n-2} + n
\end{aligned}
$$

সমাধান: এটা প্রায় ফিবনাচ্চি সমস্যাটির মতোই, কিন্তু ঝামেলা হচ্ছে রিকারেন্সে একটি n যোগ করা হয়েছে। এটা না সরালে ধ্রুবক কোন ম্যাট্রিক্স পাওয়া যাবেনা। এজন্য আমরা আগের সমস্যার মত এমন আরেকটি রিকারেন্স g বের করতে পারি যেন $g_n = n$ হয়। এটা বের করা বেশ সহজ

$$
\begin{aligned}
g_0 &= 0\\
g_n &= g_{n-1} + 1
\end{aligned}
$$

এরপর n এর বদলে g_n বসিয়ে দিলেই আমরা ঠিক আগের উদাহরণের মত ম্যাট্রিক্সটি বের করতে পারব। রিকারেন্স দুটোকে এক করলে পাব

$$
\begin{aligned}
g_n &= g_{n-1} + 1\\
f_n &= f_{n-1} + f_{n-2} + g_n
\end{aligned}
$$

■

সমস্যা ২.২: নিচের ধারাটির জন্য ম্যাট্রিক্স বের কর

$$\sum_{i=1}^{n} i^k = 1^k + 2^k + 3^k + \cdots + n^k$$

সমাধান: যদিও এটা ঠিক ডাইনামিক প্রোগ্রামিং এর সমস্যা না, এরপরেও ম্যাট্রিক্স এক্সপো এর খুব সুন্দর একটা উদাহরণ। যোগফলের জন্য খুব সহজ একটা রিকারেন্স বের করতে পারি

$$
\begin{aligned}
f_0 &= 0\\
f_n &= f_{n-1} + n^k
\end{aligned}
$$

এখানেও n^k পদটা ঝামেলা করছে। যদি $k = 1$ হত তাহলে কিন্তু আমরা আগের মতই $g_n = n$ এর রিকারেন্সটা বসিয়ে দিতে পারতাম। তাহলে আরেকটু কঠিন

কেস চিন্তা করি। $k = 2$ হলে কী করতাম? তখন আমাদের এমন একটি রিকারেন্স h লাগত যেন $h_n = n^2$ হয়। এটা বের করাও কিন্তু বেশ সহজ।

$$h_0 = 0$$
$$h_n = h_{n-1} + 2g_{n-1} + 1$$

এখানে আমরা $n^2 = (n-1)^2 + 2(n-1) + 1$ অভেদটি ব্যবহার করেছি। n^2 এর বদলে h_n, $(n-1)^2$ এর বদলে h_{n-1} এবং $(n-1)$ এর বদলে g_{n-1} বসিয়ে দিলেই রিকারেন্সটি পেয়ে যাব। একইভাবে আমরা n^3 এর রিকারেন্সটিও বের করতে পারি। p_n যদি n^3 এর রিকারেন্স হয়, তাহলে $n^3 = (n-1)^3 + 3(n-1)^2 + 3(n-1) + 1$ থেকে আমরা পাব

$$p_0 = 0$$
$$p_n = p_{n-1} + 3h_{n-1} + 3g_{n-1} + 1$$

প্যাটার্নটি কি বুঝতে পারছ। n^k কে আমরা $(n-1)$ এর বিভিন্ন পাওয়ার দিয়ে লেখছি। দ্বিপদী উপপাদ্য দিয়ে পরের রিকারেন্সগুলো সহজেই বের করে ফেলতে পারি। নিচের অভেদটি ব্যবহার করে $n^1, n^2, n^3, n^4, \ldots, n^k$ সবকিছুর জন্যই রিকারেন্স বের করতে পারব

$$n^m = \sum_{i=0}^{m} \binom{m}{i} (n-1)^i$$

সবমিলিয়ে আমরা $k+1$ টি রিকারেন্স পাব। সুতরাং আমাদের ম্যাট্রিক্সটি হবে একটি $(k+1) \times (k+1)$ ম্যাট্রিক্স। ম্যাট্রিক্স এক্সপনেন্সিয়েশনের দিয়ে আমরা সমস্যাটি $\mathcal{O}(k^3 \log n)$ এ সমাধান করতে পারি। k যদি বেশ ছোট হয় (যেমন $k \leq 50$) এবং n যদি অনেক বড় হয় (যেমন $n \leq 10^9$) তাহলে এভাবেই আমাদের সমস্যাটি সমাধান করতে হবে। ■

§ ২.৪ গ্রাফ থিওরি এবং ম্যাট্রিক্স

গ্রাফকে প্রকাশ করার জন্য অ্যাডজাসেন্সি ম্যাট্রিক্স প্রায় ব্যবহার করি। এই ম্যাট্রিক্স দিয়েও বেশ কিছু কাজ করা যায়। নিচের সমস্যাটি দেখ

সমস্যা ২.৩: ধর তোমার কাছে n টি নোডের একটি গ্রাফ দেওয়া আছে। গ্রাফ 1 নম্বর নোড থেকে n তম নোডে ঠিক k টি এজ ব্যবহার করে কতভাবে যাওয়া যায়?

সমাধান: প্রথমে আমরা ডাইনামিক প্রোগ্রামিং দিয়ে প্রবলেমটি চিন্তা করব। ধর $D_{k,i,j} =$ গ্রাফের নোড i থেকে নোড j তে ঠিক k টি এজ ব্যবহার করে কতভাবে যাওয়া যায়। এটা আমরা নিচের রিকারেন্স দিয়ে বের করতে পারি

$$D_{k,i,j} = \sum_{x=1}^{n} D_{k-1,i,x} \times A_{x,j}$$

যেখানে A হল আমাদের অ্যাডজাসেন্সি ম্যাট্রিক্স। এর ব্যাখ্যা হল প্রথমে আমরা i থেকে কোন একটি নোড x এ $k-1$ টি এজ ব্যবহার করে গিয়েছি। এ কাজটি করা যাবে $D_{k-1,i,x}$ উপায়ে। এরপর x থেকে আমরা j তে গিয়েছি একটিমাত্র এজ ব্যবহার করে। এ কাজটি করা যাবে $A_{x,j}$ উপায়ে, কেননা $A_{x,i} = 1$ হলে x আর j এর মধ্যে এজ বিদ্যমান, সুতরাং একভাবেই যে এজ ব্যবহার করে x থেকে j তে যাওয়া যাবে; আবার $A_{x,j} = 0$ হলে তাদের মধ্যে কোন এজ নাই, তাই শূন্য উপায়ে x থেকে j তে যাওয়া যাবে। দুটি গুন করলেই আমরা সর্বমোট উপায় পাব। আবার x তো কোন নির্দিস্ট নোড না, তাই $x = 1, 2, 3, \ldots, n$ সবার জন্যই $D_{k-1,i,x} \times A_{x,j}$ যোগ করতে হবে।

এটি দেখে কি ম্যাট্রিক্স গুনের কথা মনে পড়ে না? ম্যাট্রিক্স গুন কিন্তু আমরা প্রায় একইভাবে সংজ্ঞায়িত করেছিলাম। ধর $D_{(k)}$ ম্যাট্রিক্সের (i, j) তম এন্ট্রি $D_{k,i,j}$। তাহলে উপরের রিকারেন্সটিকে ম্যাট্রিক্স গুণফল দিয়েই আমরা প্রকাশ করতে পারি

$$D_{(k)} = D_{(k-1)} \times A$$

আবার D_1 এবং অ্যাডজাসেন্সি ম্যাট্রিক্স A কিন্তু একই ম্যাট্রিক্স। তাই

$$\begin{aligned} D_{(1)} &= A \\ D_{(2)} &= D_{(1)} \times A = A^2 \\ D_{(3)} &= D_{(2)} \times A = A^3 \\ &. \end{aligned}$$

$$.$$

$$.$$

$$D_{(k)} = D_{(k-1)} \times A = A^k$$

অর্থাৎ গ্রাফের অ্যাডজাসেন্সি ম্যাট্রিক্স এর k তম পাওয়ার বের করলেই আমরা আমাদের উত্তর পেয়ে যাব!! কমপ্লেক্সিটি হবে $\mathcal{O}(n^3 \log k)$ ■

§ ২.৫ অন্যান্য সাব-রিং

একটা জিনিশ খেয়াল করে দেখেছ? আমরা কিন্তু ম্যাট্রিক্সের অ্যাসোসিয়েটিভিটি ছাড়া আর কোন ধর্মই ব্যবহার করিনি। সাধারণভাবে যেভাবে ম্যাট্রিক্স গুন সংজ্ঞায়িত করা হয় তাকে বলে হয় $(+, \times)$ সাব-রিং। কারণ A ও B এর গুনফল C বের করতে $A_{i,x}$ এবং $B_{x,j}$ গুন করে সেগুলো আমরা যোগ করছি। ম্যাট্রিক্স গুণফল অ্যাসোসিয়েটিভ কারণ যোগ এবং গুন দুটি অ্যাসোসিয়েটিভ অপারেটর। আমরা যদি যোগ, গুনের বদলে অন্য অ্যাসোসিয়েটিভ অপারেটর ব্যবহার করে ম্যাট্রিক্স গুণফল সংজ্ঞায়িত করতাম তাহলেও কিন্তু আমাদের ম্যাট্রিক্স গুণফল অ্যাসোসিয়েটিভই থাকত। একইভাবে আমরা ম্যাট্রিক্সের পাওয়ারও বের করতে পারব। এমন একটি বিশেষ সাব-রিং হচ্ছে $(\max, +)$ সাব-রিং। এই রিং-এ যদি $C = AB$ হয় তাহলে

$$C_{i,j} = \max_{x=1}^{m}\{A_{i,x} + B_{x,j}\}$$

হবে। এটিও আগের মতই অ্যাসোসিয়েটিভ হবে।

সমস্যা ২.৪: ধর তোমার কাছে n টি নোডের একটি ওয়েটেড গ্রাফ (weighted graph) দেওয়া আছে। গ্রাফ 1 নম্বর নোড থেকে n তম নোডে ঠিক k টি এজ ব্যবহার করে এমন শর্টেস্ট পাথের (shortest path) মান কত?

সমাধান: এটা কিন্তু প্রায় আগের সমস্যাটির মতই। যদি আমরা অ্যাডজাসেন্সি ম্যাট্রিক্স A এর $A_{i,j} = i$ এবং j এর মধ্যে এজের ওয়েট ধরি (যদি এজ না থাকে তাহলে এর মান ∞ হবে) এবং $D_{k,i,j} =$ গ্রাফের নোড i থেকে নোড j তে ঠিক k টি এজ ব্যবহার করে শর্টেস্ট পাথ ধরি তাহলে আমাদের

রিকারেন্সটি হবে

$$D_{k,i,j} = \max_{i=1}\{D_{k-1,i,x} + A_{x,j}\}$$

এর ব্যাখ্যাও ঠিক আগের সমস্যার মতই। শুধু পার্থক্য হচ্ছে $\sum$ এর বদলে max এবং $\times$ এর বদলে $+$ বসেছে এখানে। তাই এটিকে আমরা $(\max, +)$ সাব-রিং এর ম্যাট্রিক্স গুণফল হিসেবে চিন্তা করতে পারি। এই সাব-রিং এ A^k এর মান বের করলেই আমরা আমাদের উত্তর পেয়ে যাব! ■

§ ২.৬ শেষ কথা

ম্যাট্রিক্স কোড করার জন্য আমি সাধারণত একটা ক্লাস লেখে ফেলি। ক্লাসে তুমি যোগ, গুন এসব অপারেটর ওভারলোড করতে পারবে। আরেকটা ট্রিক হল যদি তোমাকে একই ম্যাট্রিক্স A এর পাওয়ার বারবার বের করতে হয় তাহলে $A^1, A^2, A^4, A^8, \ldots, A^{2^k}$ ম্যাট্রিক্স গুলো আগের বের করতে রাখতে পারো। এরপর পাওয়ারকে বাইনারিতে প্রকাশ করে তুমি বের করা ম্যাট্রিক্সগুলো দিয়েই যেকোনো পাওয়ার বের করতে পারবে। আবার তুমি এই ম্যাট্রিক্সগুলোকে সরাসরি ভেক্টরের সাথে গুন করতে পারো (অ্যাসোসিয়েটিভিটি!!)। দুটো $n \times n$ ম্যাট্রিক্স গুন করতে $\mathcal{O}(n^3)$ কমপ্লেক্সিটি লাগে, কিন্তু একটি $n \times n$ ম্যাট্রিক্সের সাথে একটি $n \times 1$ ভেক্টর গুন করতে $\mathcal{O}(n^2)$ কমপ্লেক্সিটি লাগছে। তাই অনেক সমস্যায় $A^1, A^2, A^4, A^8, \ldots, A^{2^k}$ বের করার পরে $\mathcal{O}(n^2 \log k)$ কমপ্লেক্সিটিতেই তুমি উত্তর বের করতে পারবে।

পড়া থামাও, নিজে চেষ্টা করো

তোমার কাছে একটি $1 \times n$ গ্রিড আছে এবং যথেষ্ট সংখ্যক 1×1 এবং 1×2 ডোমিনো আছে। কত ভাবে তুমি গ্রিডটিতে ডোমিনো গুলো বসাতে পারবে যেন একই ঘরে একাধিক ডোমিনো না থাকে। $(1 \leq n \leq 10^9)$

অধ্যায় ৩

ন্যাপস্যাক

§ ৩.১ $0/1$ ন্যাপস্যাক

ধর তোমার কাছে n টি বস্তু আছে, i তম বস্তুর ওজন w_i এবং দাম v_i। তোমার কাছে একটা ব্যাগ (ন্যাপস্যাক) আছে যা সর্বোচ্চ W ওজনের বস্তু ধারণ করতে পারে। এই ব্যাগে তুমি সর্বোচ্চ কত দামের বস্তু রাখতে পারবে?

একে $0/1$ ন্যাপস্যাক বলা হয়, কারণ এখানে প্রতিটি বস্তু সর্বোচ্চ একবারই নেওয়া যাবে। এটির জন্য আমাদের ডাইনামিক প্রোগ্রামিং এর সাহায্য নিতে হবে। ধরি $f_{i,j} =$ প্রথম i টি বস্তুর মধ্যে সর্বোচ্চ কত দামের বস্তু নেওয়া যায় যাতে বস্তুগুলোর ওজনের যোগফল $\leq j$ হয়। তাহলে আমাদের রিকারেন্সটি

$$f_{i,j} = \max\{f_{i-1,j},\, f_{i-1,j-w_i} + v_i\}$$

অর্থাৎ $f_{n,W}$ এর মানই হবে আমাদের অ্যান্সার। এখানে টাইম ও মেমরি কমপ্লেক্সিটি উভয়ই $\mathcal{O}(nW)$। তবে যেহেতু $f_{i,j}$ এর মান কেবলমাত্র $f_{i-1,0}$, $f_{i-1,1}$, $f_{i-1,2}$, $\dots$, $f_{i-1,W}$ এর ওপর নির্ভর করে তাই $\mathcal{O}(W)$ মেমরি দিয়েও কাজটি করা সম্ভব। (মেমোরি অপটিমাইজেশনের চ্যাপ্টারটা দেখ)

§ ৩.২ 0-K ন্যাপস্যাক

ধর তোমার কাছে n টাইপের বস্তু আছে, i তম টাইপের বস্তু আছে k_i টি এবং এদের প্রত্যেকটির ওজন w_i এবং দাম v_i। তোমার কাছে একটা ব্যাগ

(ন্যাপস্যাক) আছে যা সর্বোচ্চ W ওজনের বস্তু ধারণ করতে পারে। এই ব্যাগে তুমি সর্বোচ্চ কত দামের বস্তু রাখতে পারবে?

আগেরটার সাথে এটার পার্থক্য হচ্ছে এখানে i তম বস্তু সর্বোচ্চ k_i সংখ্যক বার নেওয়া যাবে। এখানেও আগের মতই ডাইনামিক প্রোগ্রামিং ব্যবহার করা যায়, ধরি $f_{i,j} =$ প্রথম i টি বস্তুর মধ্যে সর্বোচ্চ কত দামের বস্তু নেওয়া যায় যাতে বস্তুগুলোর ওজনের যোগফল $\leq j$ হয়। তাহলে,

$$f_{i,j} = \max_{m=0}^{k_i}\{f_{i-1,j-w_i m} + v_i m\}$$

অর্থাৎ i তম বস্তু কতবার নিচ্ছি সেটার সবগুলো অপশন কনসিডার করতে হবে। আগেরটার কোড বুঝে থাকলে এটার কোড নিজেরই পারার কথা। এখানে টাইম কমপ্লেক্সিটি হবে $\mathcal{O}(W \times \sum k_i)$

কিন্তু এইখানে সমস্যা হচ্ছে $\sum k_i$ এর মান অনেক বড় হতে পারে। আশার কথা হল এই প্রবলেমের এইটাই সবচেয়ে অপটিমাল সলিউশন না। $\mathcal{O}(W \times \sum \log k_i)$ কমপ্লেক্সিটিতেও এই প্রবলেমটি সলভ করা সম্ভব।

আইডিয়াটি হচ্ছে প্রত্যেক k_i এর বাইনারি রিপ্রেজেন্টেশনকে ব্যবহার করা। একটি উদাহরণ দেখা যাক, ধর কোন এক টাইপের বস্তুর $(k_i, w_i, v_i) = (27, 13, 5)$। অর্থাৎ ঐ টাইপের বস্তু আছে 27 টি এবং তার ওজন 13 ও দাম 5। এখন 27 কে এইভাবে লেখা যায়:

$$27 = 11011_2 = 1111_2 + 1100_2 = (2^4 + 2^3 + 2^2 + 2^1 + 2^0) + 12$$

অর্থাৎ আমরা যদি $(27, 13, 5)$ বস্তুটির বদলে $(1, 13 \times 2^4, 5 \times 2^4)$, $(1, 13 \times 2^3, 5 \times 2^3)$, $(1, 13 \times 2^2, 5 \times 2^2)$, $(1, 13 \times 2^1, 5 \times 2^1)$, $(1, 13 \times 2^0, 5 \times 2^0)$ এবং $(1, 13 \times 12, 5 \times 12)$ বস্তুগুলোর ওপর ন্যাপস্যাক ডিপি চালাই তাহলে উত্তর চেঞ্জ হবে না, এর কারন হচ্ছে 2^4, 2^3, 2^2, 2^1, 2^0 এবং 12 দিয়ে 0 থেকে 27 পর্যন্ত সব সংখ্যা কে লেখা যায়, তবে 27 এর বড় কোন সংখ্যাকে লেখা যায় না (কিছু কিছু সংখ্যাকে একাধিক উপায়ে লেখা যেতে পারে, কিন্তু সেটা আমাদের জন্য সমস্যা না)। এইভাবে প্রতিটি বস্তুকে তার বাইনারি

রিপ্রেজেন্টেশন অনুযায়ী ভেঙ্গে দিতে হবে। ভেঙ্গে দেওয়ার পর কিন্তু আমাদের আর 0-K ন্যাপস্যাক থাকছে না, 0-1 ন্যাপস্যাক হয়ে যাচ্ছে। কারণ ভেঙ্গে দেওয়ার পর প্রত্যেক বস্তুকে সর্বোচ্চ একবারই নেওয়া সম্ভব ($k_i = 1$)। অর্থাৎ ভেঙ্গে দেওয়ার পর আমাদের মোট বস্তু হবে $\mathcal{O}(\sum \log k_i)$ টি। তাই 0-1 ন্যাপস্যাক এর কমপ্লেক্সিটি হবে $\mathcal{O}(W \times \sum \log k_i)$।

মজার ব্যাপার হল এই প্রবলেমের $\mathcal{O}(W \times \sum \log k_i)$ এর চেয়েও ভাল সলিউশন আছে। $\mathcal{O}(nW)$ কমপ্লেক্সিটিতেও 0-K ন্যাপস্যাক সলভ করা সম্ভব। রিকারেন্সটি আবার লক্ষ্য করি:

$$f_{i,j} = \max_{m=0}^{k_i}\{f_{i-1,j-w_i m} + v_i m\} \quad (1)$$

কোনো ফিক্সড i এর জন্য $f_{i,0}\,, f_{i,1}\,, \ldots, f_{i,W}$ এর মান যদি আমরা $\mathcal{O}(W)$ তে বের করতে পারি, তাহলেই $\mathcal{O}(nW)$ কমপ্লেক্সিটি হয়ে যাবে। এখন লক্ষ্য করি, $f_{i,j}$ এর মান $f_{i-1,j}\,, f_{i-1,j-w_i}\,, f_{i-1,j-2w_i}\,, f_{i-1,3w_i}\,, \ldots$ মানগুলোর ওপর নির্ভর করে। অন্যভাবে বলা যায় $f_{i,j}$ এর মান এমন সব $f_{i-1,p}$ এর মানের ওপর নির্ভর করে যাতে $p \equiv j \mod w_i$ হয়। এটাকে কাজে লাগিয়েই $\mathcal{O}(W)$ তে কাজটি করা সম্ভব। আমরা $f_{i,j}$ এর মান $0 \le j \le W$ এর জন্য একসাথে বের না করে w_i এর প্রত্যেক মডুলো ক্লাসের জন্য আলাদা ভাবে বের করতে পারি। বুঝানোর সুবিধার্তে ধরি,

$$g_m(i,j) = f_{i,m+jw_i}$$

যেখানে $0 \le m < w_i$। এখন আমরা একটা ফিক্সড m এর জন্য $g_m(i,j)$ এর সকল মান বের করব, যেখানে $0 \le m + jw_i \le W$। (1) নং রিকারেন্সের সাহায্যে $g_m(i,j)$ কে এইভাবে লেখা যায়:

$$\begin{aligned} g_m(i,j) &= \max_{h=j-k_i}^{j}\{g_m(i-1,h) + (j-h)v_i\} \\ &= \max_{h=j-k_i}^{j}\{g_m(i-1,h) - hv_i\} + jv_i \end{aligned}$$

এখান থেকেই বুঝা যাচ্ছে $g_m(i-1,0), g_m(i-1,1) - v_i, g_m(i-1,2) - 2v_i, \ldots$ এর প্রতিটি $k_i + 1$ দৈর্ঘ্যের সাবঅ্যারের মিনিমাম ভ্যালু বের করতে পারলেই $g_m(i,j)$ এর সকল মান আমরা সহজেই বের করতে পারব। কোনো n দৈর্ঘ্যের অ্যারের প্রতিটি m দৈর্ঘ্যের সাবঅ্যারের মিনিমাম (বা ম্যাক্সিমাম) ভ্যালু $\mathcal{O}(n)$ এই বের করা যায় (স্লাইডিং উইন্ডোর সাহায্যে)। অর্থাৎ প্রত্যেক মডুলো ক্লাসের জন্য আমরা লিনিয়ার টাইমেই g_m এর মান বের করতে পারব। যেহেতু প্রত্যেকটি সংখ্যাই কেবলমাত্র একটি মডুলো ক্লাসের অন্তর্ভুক্ত তাই ওভারঅল কমপ্লক্সিটি হবে $\mathcal{O}(W)$। তাই প্রত্যেকটি i এর জন্য $f_{i,j}$ এর মান বের করতে $\mathcal{O}(nW)$ কমপ্লেক্সিটি প্রয়োজন।

§ ৩.৩ সাবসেট সাম:

এই সেকশনের সব জায়গায় সেট বলতে মাল্টিসেট বুঝান হবে। অর্থাৎ সেটে একই উপাদান একাধিক বার থাকতে পারে।

ন্যাপস্যাকের সবচেয়ে গুরুত্বপূর্ণ ভ্যারিয়েশন এটি। ধর তোমার কাছে n দৈর্ঘ্যের একটা অ্যারে a এবং একটি নাম্বার m দেওয়া আছে। তোমাকে বলতে হবে a এর নাম্বার গুলো ব্যবহার করে যোগফল m বানানো যায় কিনা।

অর্থাৎ $S = \{1, 2, 3, \ldots, n\}$ হলে এমন কোন সাবসেট T পাওয়া সম্ভব কিনা যাতে $T \subseteq S$ এবং $\sum_{i \in T} a_i = m$ হয়।

ধরি,

$$f_{i,j} = \begin{cases} 1, & \text{যদি প্রথম } i \text{ টি সংখ্যা হতে যোগফল } j \text{ বানানো সম্ভব হয়,} \\ 0, & \text{সম্ভব না হয়.} \end{cases}$$

তাহলে,

$$f_{i,j} = f_{i-1,j} \vee f_{i-1,j-a_i}$$

$\vee$ এখানে or অপারেটরটাকে বুঝাচ্ছে। তাহলে এই ডিপিটা ক্যালকুলেট করতে আমাদের $\mathcal{O}(nm)$ টাইম ও $\mathcal{O}(m)$ মেমরি লাগছে। তবে এই সলিউশন কে অপটিমাইজ করার জন্য আরেকটা সস্তা অপটিমাইজেশন আছে। তা হল `bitset` ব্যবহার করা। `bitset` ব্যবহার করলে টাইম কমপ্লেক্সিটি দাড়ায় $\mathcal{O}(\frac{nm}{64})$ এবং মেমোরি কমপ্লেক্সিটি দাড়ায় $\mathcal{O}(\frac{m}{64})$।

§ ৩.৪ ডাইনামিক সাবসেট সাম:

ধর সাবসেট সাম প্রবলেমটায় তোমাকে কিছু আপডেট আর কুয়েরিও দেওয়া হল। অর্থাৎ প্রত্যেক আপডেটে তোমাকে একটি সংখ্যা p দেওয়া হবে এবং তোমাকে সংখ্যাটাকে সেটে অ্যাড করতে হবে অথবা সেট থেকে রিমুভ করতে হবে। প্রত্যেক কুয়েরিতে তোমাকে একটি সংখ্যা r দেওয়া হবে এবং তোমাকে বলতে হবে r সংখ্যাটিকে সেটের সংখ্যাগুলোর যোগফল হিসেবে লেখা যায় কিনা।

ধরা যাক মোট আপডেট ও কুয়েরি Q টি। তাহলে যদি আমরা Q বারই সাবসেট সাম-এর ডিপি টা নতুন করে আপডেট করি তাহলে কমপ্লেক্সিটি $\mathcal{O}(\frac{Qnr_{\max}}{64})$ হয়ে যাচ্ছে। তবে এই প্রবলেমটি $\mathcal{O}(Qr_{\max})$ টাইমেও করা সম্ভব, যেখানে $r_{\max}$ হল r এর ম্যাক্সিমাম ভ্যালু।

এর জন্য আমাদের ডিপি টাকে একটু চেঞ্জ করতে হবে। ধরি, $f_j =$ সেটে যেসব উপাদান আছে তাদের কোনো সাবসেট নিয়ে কতভাবে j সংখ্যাটি বানানো যায়। তাহলে প্রত্যেক কুয়েরিতে $f_r > 0$ কিনা তা চেক করলেই হচ্ছে আমাদের। আর যদি নতুন কোন নাম্বার অ্যাড বা রিমুভ করতে হয় তাহলে নরমাল সাবসেট সাম ডিপির মতই f_j এর মান আপডেট করা যায়। এখন সমস্যা হচ্ছে f_j মান অনেক বড় হয়ে যেতে পারে, এমনকি `long long` এও আটবে না। তাই f_r কে আমরা $\mod P$ ক্যালকুলেট করব যেখানে P র‍্যানডম কোন প্রাইম নাম্বার। এখন যদি $f_r = 0$ হয়, এবং তারপরেও r কে যোগফল হিসেবে লেখা যাবে সেটির সম্ভাবনা নেয় বললেই চলে। (কেউ চাইলে ২-৩ টি `mod` ও ব্যবহার করতে পারে)।

§ ৩.৫ $\mathcal{O}\left(s\sqrt{s}\right)$ সাবসেট সাম:

এখানে s সেটের সবগুলো সংখ্যার যোগফল বুঝাচ্ছে। যদি কোন সংখ্যা t এর থেকে বড় হয়, তাহলে আমরা নরমালি `bitset` দিয়ে ডিপি টা আপডেট করব, এটি করতে $\mathcal{O}\left(\frac{s}{64} \times \frac{s}{t}\right)$ কমপ্লেক্সিটি লাগে (কারন t এর থেকে বড় সংখ্যা সর্বোচ্চ $\frac{s}{t}$ বার পাওয়া যাবে)। আর যদি t এর থেকে ছোট হয় তাহলে আমরা 0-k ন্যাপস্যাক এর মত ডিপি টাকে আপডেট করব। অর্থাৎ t এর থেকে ছোট কোন সংখ্যা কতবার আছে সেটা বের করে তার ওপর 0-k ন্যাপস্যাক প্রয়োগ করব। এ কাজটি করতে সর্বোচ্চ $\mathcal{O}(st)$ কমপ্লেক্সিটি লাগে। $t = \sqrt{\frac{s}{64}}$ হলে টোটাল কমপ্লেক্সিটি দাড়ায়:

$$\mathcal{O}\left(\frac{s}{64} \times \frac{s}{t} + s \times t\right) = \mathcal{O}\left(s\sqrt{\frac{s}{64}}\right)$$

অধ্যায় ৪

ব্যারিকেডস ট্রিক

§ ৪.১ একটি পোলিশ সমস্যা

বাইটল্যান্ড নামের একটি দ্বীপে n টি শহর আছে এবং শহরগুলোর মধ্যে কিছু দ্বিমুখী রাস্তা আছে। এ শহরের ম্যাপ একটি বিশেষ ধরনের, একটি শহর থেকে আরেকটি শহরে কেবলমাত্র একভাবেই যাওয়া যায়। অর্থাৎ গ্রাফ থিওরির ভাষায় বাইটল্যান্ডের মাপটি একটি ট্রি গ্রাফ।

দুঃখজনকভাবে বাইটল্যান্ড দ্বীপটিতে এখন যুদ্ধ চলছে। বাইটল্যান্ডের সেনাবাহিনী নিজেদের প্রতিরক্ষার জন্য একটি যুদ্ধক্ষেত্র তৈরি করতে চায়। তারা যুদ্ধক্ষেত্রটি তৈরি করার জন্য কিছু রাস্তা ব্লক করে দিবে। যুদ্ধক্ষেত্রটি তৈরির জন্য তাদের তিনটি শর্ত মেনে চলতে হবে।

- $\rightarrow$ যুদ্ধক্ষেত্রের অন্তর্গত শহরগুলোর নিজেদের মধ্যে চলাচলের রাস্তা থাকবে। অর্থাৎ যুদ্ধক্ষেত্রের যেকোনো দুটি শহরের মধ্যে কোনো ব্লক করা রাস্তা থাকবে না।
- $\rightarrow$ যুদ্ধক্ষেত্রের ভিতরের কোনো শহর থেকে যুদ্ধক্ষেত্রের বাইরের কোনো শহরে যাওয়ার কোনো রাস্তা থাকবে না।
- $\rightarrow$ যুদ্ধক্ষেত্রের মধ্যে k টি শহর থাকবে।

বেশি সংখ্যক রাস্তা ব্লক করে দিলে শহরের মধ্যে যাতায়াতে সমস্যা হতে হতে পারে। তোমাকে বাইটল্যান্ড দ্বীপটির যুদ্ধক্ষেত্র প্রস্তুত করার দায়িত্ব দেওয়া হয়েছে। তোমাকে বলতে হবে সর্বনিম্ন কয়টি রাস্তা ব্লক করে বাইটল্যান্ড শহরে একটি যুদ্ধক্ষেত্র প্রস্তুত করা সম্ভব।

এটি আসলে পোল্যান্ডের ইনফরমাটিক্স অলিম্পিয়াডের ব্যারিকেডস নামের প্রবলেম। এই প্রবলেম থেকেই মূলত এই অধ্যায়ের আইডিয়াটা জনপ্রিয় হয়েছিল, তাই এখন এই ট্রিক এখন ব্যারিকেডস ট্রিক নামেই প্রোগ্রামিং মহলে অধিক পরিচিত।

§ ৪.২ সমাধান

সমস্যাটি দেখে অনেকেই আন্দাজ করতে পারছ এইখানে ট্রি গ্রাফটির ওপরেই ডাইনামিক প্রোগ্রামিং করতে হবে। এ ধরনের সমস্যা সমাধানের জন্য একটি বিশেষ ধরনের ডাইনামিক প্রোগ্রামিং ব্যবহার করা হয় যাকে সিবলিং ডিপি নামে অনেকে চিনে। প্রথমে দেখা যাক আমাদের ডিপি স্টেট কি হতে পারে।

প্রথমে আমরা যেকোনো একটি নোডকে ট্রি-এর রুট ধরে নিব। ধরা যাক ১ নম্বর নোডটিকে আমরা রুট হিসেবে ধরেছি। v নোডটির সাবট্রিকে আমরা T_v দ্বারা প্রকাশ করব এবং সাবট্রি-এর মধ্যে নোড সংখ্যাকে $|T_v|$ দ্বারা প্রকাশ করব। অর্থাৎ T_1 দিয়ে সম্পূর্ণ ট্রি টাকেই বুঝানো হচ্ছে। যারা ট্রি ডিপির সাথে মোটামুটি পরিচিত তারা ইতোমধ্যে বুঝে গিয়েছ আমাদের স্টেট কি হতে পারে। ধরা যাক $f_{v,x}$ এর মান হল সর্বনিম্ন কতটি এজ মুছে দিলে v এর সাবট্রি-এর মধ্যে x টি নোডের একটি কানেক্টেড সাবগ্রাফ পাওয়া যাবে যাতে v নোডটি নিজেও সেই সাবগ্রাফের অংশ হয়। আমরা যদি প্রতিটি নোড v জন্য $f_{v,x}$ এর মানগুলো বের করে নিতে পারি তাহলে খুব সহজেই প্রতিটি কুয়েরি $\mathcal{O}(n)$ কমপ্লেক্সিটিতে বের করে ফেলতে পারব।

এখন দেখা যাক কিভাবে আমরা $f_{v,x}$ এর মানগুলো ক্যালকুলেট করতে পারি। ধরা যাক নোড v এর জন্য আমরা $f_{v,x}$ এর মান বের করছি। v এর সাবট্রিতে $|T_v| - 1$ টি এজ আছে, তাই $|T_v| - 1$ টির বেশি এজ মুছে ফেলা সম্ভব না, এজন্য $1 \leq x < |T_v|$ এর জন্য $f_{v,x}$ এর মান বের করাই আমাদের জন্য যথেষ্ট। ধর নোড v এর চাইল্ডগুলো হল $u_1, u_2, \ldots, u_m$। প্রতিটি চাইল্ডের জন্য যদি আমাদের $f_{u_i,*}$ এর মানগুলো ক্যালকুলেট করা থাকে তাহলে $f_{v,x}$ এর মান আমরা কিভাবে বের করতে পারি সেটি একটু চিন্তা করে দেখ।

যেকোনো একটি চাইল্ড u_i এর কথা চিন্তা কর। আমাদের হাতে দুটি অপশন

আছে: হয় আমরা u_i এর সাবট্রি থেকে আমরা q_i টি নোডের এমন একটি সাবগ্রাফ নিব যাতে u_i নোডটিও তার অন্তর্ভুক্ত থাকে, অথবা (v, u_i) এজটিই আমরা মুছে দিব; সেক্ষেত্রে আমরা $q_i = 0$ ধরতে পারি। প্রথম ক্ষেত্রে আমাদের f_{u_i,q_i} টি এজ মুছে ফেলতে হবে, আর দ্বিতীয় ক্ষেত্রে আমাদের ১ টি এজ মুছে ফেলতে হবে। আর আমাদের $f_{v,x}$ এর মান বের করার জন্য এমন ভাবে q_i সিলেক্ট করতে হবে যেন $q_1 + q_2 + \cdots + q_m = x - 1$ হয়।

ডিপি স্টেট-এ শুধুমাত্র v আর x এর মান রেখে আমরা আর আগাতে পারছি না, কারন আমরা যদি প্রতিটি চাইল্ড থেকে সম্ভাব্য সকল ধরনের q_i এর মান নিয়ে চেক করি তাহলে আমাদের কমপ্লেক্সিটি এক্সপোনেনশিয়াল হয়ে যাবে। তাই আমাদের $f_{v,x}$ এর মান বের করার জন্য আরেকটি ডিপির সাহায্য নিতে হবে।

ধরি $g_{i,x}$ এর মান হল v এর প্রথম i টি চাইল্ড থেকে সর্বনিম্ন যে কয়টি এজ মুছে দিলে x টি নোডের একটি সাবগ্রাফ পাওয়া যাবে যেন v নোডটিও সেই সাবগ্রাফের অংশ হয়। অর্থাৎ প্রথম i টি চাইল্ড থেকে $q_1, q_2, \ldots, q_i$ এমনভাবে সিলেক্ট করতে হবে যেন $q_1 + q_2 + \cdots + q_i = x - 1$ হয়। এখন $g_{i,x}$ এর মান আমরা $g_{i-1,*}$ মানগুলো থেকে খুব সহজেই বের করে নিতে পারি নিচের রিকারেন্সটির মাধ্যমে:

$$g_{i,x} = \min\{g_{i-1,x} + 1, \min_{1 \le a \le x} g_{i-1,x-a} + f_{u_i,a}\}$$

উপরের লাইনে দুটি অপশনই বিবেচনা করা হয়েছে। যদি i তম চাইল্ডের সাথে v এর এজটি মুছে ফেলা হয় তাহলে i তম চাইল্ডের আগের চাইল্ডগুলো থেকে x টি নোডের সাবগ্রাফ পেতে কমপক্ষে $g_{i-1,x}$ টি এজ মুছে ফেলতে হবে এবং (v, u_i) এজটি সহ মোট $g_{i-1,x} + 1$ টি এজ মুছতে হবে। আর যদি i তম চাইল্ড u_i এর সাবট্রি থেকে a টি নোডের সাবগ্রাফ নেওয়া হয় যাতে u_i তাতে অন্তর্ভুক্ত থাকে তাহলে u_i এর সাবট্রি থেকে কমপক্ষে $f_{u_i,a}$ টি এজ মুছে ফেলতে হবে এবং $u_1, u_2, \ldots, u_{i-1}$ চাইল্ডগুলো থেকে মোট $g_{i-1,x-a}$ টি এজ মুছে ফেলতে হবে। অর্থাৎ মোট $g_{i-1,x-a} + f_{u_i,a}$ টি এজ মুছে ফেলতে হবে। সবশেষে $g_{m,x}$ এর যে মান ক্যালকুলেট করা হবে সেটিই হবে $f_{v,x}$ এর মান। এভাবে প্রতিটি নোডের জন্য আমরা আরেকটি ডিপির মাধ্যমে $f_{v,x}$ এর

মানগুলো নির্নয় করতে পারব।

§ ৪.৩ কমপ্লেক্সিটি অ্যানালাইসিস

নির্দিষ্ট কোনো একটি নোড v এর জন্য $f_{v,*}$ এর মানগুলো বের করতে কয়টি অপারেশন লাগবে সেটি হিসেব করার চেষ্টা করব আমরা। প্রথমত কোনো নোড v এর সাবট্রিতে $|T_v| - 1$ সংখ্যক এজ আছে, সুতরাং $x = 1, 2, 3, \ldots, (|T_v| - 1)$ এর জন্য $f_{v,x}$ এর মানগুলো বের করলেই হবে আমাদের। আবার $g_{i-1,*}$ থেকে $g_{i,*}$ এর মানগুলো বের করতে আমাদের $\mathcal{O}\left(|T_v|.|T_{u_i}|\right)$ কমপ্লেক্সিটি প্রয়োজন। সুতরাং নোড v এর জন্য $f_{v,*}$ এর মানগুলো বের করতে আমাদের সর্বমোট কমপ্লেক্সিটি $\mathcal{O}\left(|T_v| \times \sum_{i=1}^{m} |T_{u_i}|\right)$। যেহেতু $|T_v| = 1 + \sum_{i=1}^{m} |T_{u_i}|$ তাই আমরা একে লেখতে পারি: $\mathcal{O}\left(|T_v|.|T_v|\right) = \mathcal{O}\left(|T_v|^2\right)$ হিসেবে। আর সব নোডের জন্য এই মান যোগ করলে আমাদের কমপ্লেক্সিটি হবে $\mathcal{O}\left(\sum_{i=1}^{n} |T_i|^2\right) = \mathcal{O}\left(n^3\right)$

মজার ব্যাপার হল আমরা আমাদের অ্যালগোরিদমকে তেমন কোনো পরিবর্তন না করেই $\mathcal{O}(n^2)$ বানিয়ে দিতে পারি। এজন্য আমাদের একটু ভিন্নভাবে অ্যানালাইসিস করতে হবে।

লেমা

T_v এর সকল নোডের জন্য $f_{*,*}$ এর মানগুলো $\mathcal{O}\left(|T_v|^2\right)$ কমপ্লেক্সিটিতে বের করা সম্ভব।

প্রমাণ: প্রমাণের জন্য গানিতিক আরোহের সাহায্য নিব। এখানে আমরা $|T_v|$ এর ওপর গাণিতিক আরোহ প্রয়োগ করব। ধর, যদি কোন নোড h এর জন্য $|T_h| < |T_v|$ হয় তাহলে T_h এর সকল নোডের জন্য $f_{*,*}$ এর মানগুলো $\mathcal{O}(|T_h|^2)$ কমপ্লেক্সিটিতে বের করা সম্ভব। আমরা প্রমাণ করব তাহলে T_v এর সকল নোডের জন্যও $f_{*,*}$ এর মানগুলো $\mathcal{O}(|T_v|^2)$ কমপ্লেক্সিটিতে বের করা সম্ভব। বেস কেস $|T_v| = 1$ এর জন্য নিঃসন্দেহে $\mathcal{O}(1^2) = \mathcal{O}(1)$ কমপ্লেক্সিটিতে $f_{*,*}$ এর মানগুলো বের করা সম্ভব।

ধর v এর চাইল্ডগুলো হল $u_1, u_2, \ldots, u_m$। যেহেতু $|T_{u_i}| < |T_v|$ তাই $u_1, u_2, \ldots, u_m$ চাইল্ডগুলোর সাবট্রির সকল নোডের জন্য $f_{*,*}$ এর মানগুলো বের করতে আমাদের যথাক্রমে $\mathcal{O}(|T_{u_1}|^2), \mathcal{O}(|T_{u_2}|^2), \ldots, \mathcal{O}(|T_{u_m}|^2)$ কমপ্লেক্সিটি প্রয়োজন। সুতরাং চাইল্ডগুলোর সাবট্রির সকল নোডের জন্য $f_{*,*}$ এর মানগুলো বের করতে $\mathcal{O}\left(\sum_{i=1}^{m} |T_{u_i}|^2\right)$ কমপ্লেক্সিটি লাগবে।

এখন আমাদের শুধুমাত্র $f_{v,*}$ এর মানগুলো বের করা বাকি। লক্ষ্য কর, v এর প্রথম i টি চাইল্ড থেকে সর্বোচ্চ $\sum_{j=1}^{i} |T_{u_j}|$ টি এজ মুছে ফেলা সম্ভব। তাই $g_{i,x}$ এর মান বের করার সময় আমাদের x এর মান সর্বোচ্চ $\sum_{j=1}^{i} |T_{u_j}|$ পর্যন্ত বিবেচনা করলেই হচ্ছে। $g_{i,x}$ এর রিকারেন্সটি আবার লক্ষ্য কর:

$$g_{i,x} = \min\{g_{i-1,x} + 1, \min_{1 \le a \le x} g_{i-1,x-a} + f_{u_i,a}\}$$

এখানে $x - a$ এর মান সর্বোচ্চ $\sum_{j=1}^{i-1} |T_{u_j}|$ হবে এবং a এর মান সর্বোচ্চ $|T_{u_i}|$ হবে। তাই $g_{i,*}$ এর মান বের করতে আমাদের আসলে $\mathcal{O}\left(|T_{u_i}| \times \sum_{j=1}^{i-1} |T_{u_j}|\right)$ কমপ্লেক্সিটি লাগবে। $x - a \le \sum_{j=1}^{i-1} |T_{u_j}|$ এবং $a \le |T_{u_i}|$ কে একত্র করলে আমরা পাব $x - \sum_{j=1}^{i-1} |T_{u_j}| \le a \le |T_{u_i}|$ অর্থাৎ, রিকারেন্সটিতে a এর রেঞ্জ $1 \le a \le x$ কে পরিবর্তন করে $x - \sum_{j=1}^{i-1} |T_{u_j}| \le a \le |T_{u_i}|$ করে দিলেই হবে। এভাবে সবগুলো চাইল্ডের জন্য ক্যালকুলেট করতে $\mathcal{O}\left(\sum_{i=1}^{m} \sum_{j=1}^{i-1} |T_{u_i}|.|T_{u_j}|\right)$ কমপ্লেক্সিটি লাগবে। সুতরাং মোট কমপ্লেক্সিটি হবে

$$\begin{aligned}
&\mathcal{O}\left(\sum_{i=1}^{m} \sum_{j=1}^{i-1} |T_{u_i}|.|T_{u_j}| + \sum_{i=1}^{m} |T_{u_i}|^2\right) \\
&\le \mathcal{O}\left(2\sum_{i=1}^{m} \sum_{j=1}^{i-1} |T_{u_i}|.|T_{u_j}| + \sum_{i=1}^{m} |T_{u_i}|^2\right) \\
&= \mathcal{O}\left(\left(\sum_{i=1}^{m} |T_{u_i}|\right)^2\right)
\end{aligned}$$

$$= \mathcal{O}\left(|T_v|^2\right)$$

এখন T_1 এর উপর এই এই উপপাদ্যটি প্রয়োগ করলেই প্রমাণ হয়ে যাবে সকল $f_{*,*}$ এর মান $\mathcal{O}(n^2)$ কমপ্লেক্সিটিতে বের করা সম্ভব।

§ ৪.৪ কম্বিনেটরিয়াল প্রমাণ

একটি ভিন্ন সমস্যা নিয়ে চিন্তা করা যাক। ধর আমাদের বের করতে এমন কয়টি ক্রমজোড় (x, y) আছে যেন নোড x এবং নোড y এর লোয়েস্ট কমন অ্যানসেসটর (lowest common ancestor) নোড v হয় এবং x ও y এর কোনটিই v এর সমান না হয়। একে আমরা F_v দ্বারা প্রকাশ করব। x আর y লোয়েস্ট কমন অ্যানসেসটর v হলে x এবং y অবশ্যই v এর দুটি ভিন্ন ভিন্ন চাইল্ডের সাবট্রিতে অবস্থিত। ধরা যাক x নোডটি T_{u_i} এবং y নোডটি T_{u_j} তে অবস্থিত। সুতরাং (x, y) ক্রমজোড়টিকে মোট $|T_{u_i}| \times |T_{u_j}|$ ভাবে বাছাই করা যেতে পারে। যদি আমরা সকল সম্ভাব্য চাইল্ডের ক্রমজোড় (u_i, u_j) (যাতে $u_i \neq u_j$ হয়) এর জন্য $|T_{u_i}| \times |T_{u_j}|$ এর যোগফল নির্নয় করি তাহলেই আমরা কাঙ্ক্ষিত উত্তর পেয়ে যাব। অর্থাৎ এমন ক্রমজোড় সংখ্যা হবে

$$F_v = \sum |T_{u_i}|.|T_{u_j}| = 2 \sum_{i=1}^{m} \sum_{j=1}^{i-1} |T_{u_i}| \times |T_{u_j}|$$

যেহেতু যেকোনো ক্রমজোড় (x, y) এর জন্য একটি অনন্য লোয়েস্ট কমন অ্যানসেসটর আছে এবং সর্বমোট $2\binom{n}{2}$ টি (x, y) ক্রমজোড় গঠন করা সম্ভব তাই আমরা লিখতে পারি

$$\sum_{i=1}^{n} F_i \leq 2\binom{n}{2}$$

কিন্তু আমরা জানি $\sum_{i=1}^{m} \sum_{j=1}^{i-1} |T_{u_i}| \times |T_{u_j}|$ কমপ্লেক্সিটিতে আমরা কোনো নোড v এর জন্য $f_{*,*}$ এর মানগুলো বের করতে পারি। অর্থাৎ $f_{*,*}$ এর

মানগুলো বের করতে আমাদের $\mathcal{O}(F_v)$ কমপ্লেক্সিটি প্রয়োজন। সুতরাং সকল নোডের জন্য $f_{*,*}$ এর মান বের করলে আমাদের কমপ্লেক্সিটি হবে:

$$\mathcal{O}\left(\sum_{i=1}^{n} F_i\right) = \mathcal{O}\left(2\binom{n}{2}\right) = \mathcal{O}\left(n^2\right)$$

§ ৪.৫ অন্যান্য সমস্যা

এই আইডিয়াটার সবচেয়ে ভালো দিক হচ্ছে এটি অন্যান্য অনেক ট্রি ডিপি সমস্যাতেই প্রয়োগ করা যায়। বিশেষত যদি ডিপি স্টেট-এ নোড ছাড়াও আরও একটি স্টেট থাকে তাহলে বেশির ভাগ ক্ষেত্রেই ব্যারিকেডস ট্রিক অ্যাপ্লিকেবল। নিজের করার জন্য কিছু অনুশীলন দেওয়া হল

পড়া থামাও, নিজে চেষ্টা করো

অধ্যায় ৫

এক্সচেঞ্জ আর্গুমেন্ট

§ ৫.১ প্রমাণ দাও

সাধারণত গ্রিডি অ্যালগরিদম গুলো অনেকটা এরকম হয়ঃ যতক্ষণ পর্যন্ত সম্ভব প্রদত্ত শর্তগুলো ঠিক রেখে তুমি প্রতিবার একটি করে ইলিমেন্ট সিলেক্ট করে তোমার সলিউশনে অ্যাড করবা যেটায় তোমার সবচেয়ে বেশি লাভ হয়। আমরা এক্সচেঞ্জ আর্গুমেন্ট ব্যবহার করে যেমন আমাদের এই গ্রিডি অ্যালগরিদমের শুদ্ধতা প্রমাণ করতে পারি, তেমনি এক্সচেঞ্জ আর্গুমেন্ট এর ধাপ গুলো নিয়ে চিন্তা করতে গিয়ে আমাদের গ্রিডি সলিউশনও দাঁড় করিয়ে ফেলতে পারি। এক্সচেঞ্জ আর্গুমেন্ট প্রুফ গুলোর মেইন আইডিয়া হলো, তুমি যেকোনো একটি অপ্টিমাল সলিউশন নিবে, তারপর সেটিকে ধাপে ধাপে এমনভাবে তোমার গ্রিডি সলিউশনে পরিবর্তন করবে যেন প্রতি ধাপে তোমার কোন লস না হয়। তাহলে তুমি বলতে পারবে তোমার গ্রিডি সলিউশন অন্তত কোন একটি অপ্টিমাল সলিউশনের চাইতে খারাপ না। অন্যভাবে বলতে গেলে, তোমার সলিউশনও একটি অপ্টিমাল সলিউশন। একটা উদাহরণ দেখা যাক।

সমস্যা ৫.১ (ডট প্রডাক্ট মিনিমাইজেশন): তোমাকে দুটি অ্যারে দেওয়া আছে। তোমাকে এমনভাবে অ্যারে দুটিকে রিঅ্যারেঞ্জ করতে হবে যেন তাদের ডট গুণফল অর্থাৎ, $\sum_{i=1}^{N} A_i B_i$ এর মান মিনিমাম হয়।

সমাধান: আমরা চাই না দুটি বড় বড় সংখ্যা একসাথে থাকুক কারণ তাদের গুণফল অবশ্যই বড় হয়ে যাবে। অন্যদিকে, দুটি ছোট ছোট সংখ্যা একসাথে থাকলে লাভ হতে পারে বলে মনে হতে পারে। কিন্তু এরকম করলে বড় বড় সংখ্যা গুলো একসাথে হয়ে যাবে। তাহলে এরকম একটা কিছু করা যায়- একটি ছোট আর একটি বড় সংখ্যা একসাথে পেয়ারআপ করা। এই আইডিয়াটাকে

গুছিয়ে বললে হবে- প্রথম অ্যারেটিকে নন-ডিক্রিজিং অর্ডারে সর্ট করা এবং দ্বিতীয় অ্যারেটিকে নন-ইনক্রিজিং অর্ডারে সর্ট করা। এখন আমাদের প্রমাণ করতে হবে, এটি একটি অপ্টিমাল সলিউশন। আমরা ধরে নিতে পারি প্রথম অ্যারেটি নন-ডিক্রিজিং অর্ডারে সর্ট করা আছে। এখন ধরো এমন একটা অপ্টিমাল সলিউশন আছে যেখানে B ডিক্রিজিং অর্ডারে সর্ট করা নেই, অর্থাৎ, এমন একটা i আছে যেন, $B_i < B_{i+1}$। এখন আমরা এদেরকে সোয়াপ করে আমাদের গ্রিডি সলিউশনের দিকে যেতে চাই। যদি সোয়াপ করি, তাহলে আমদের গুণফলে যেই অতিরিক্ত কস্ট অ্যাড হবে তা হলোঃ $A_iB_{i+1} + A_{i+1}B_i - A_iB_i - A_{i+1}B_{i+1}$। সুতরাং আমাদের প্রমাণ করতে হবে-

$$A_iB_{i+1} + A_{i+1}B_i - A_iB_i - A_{i+1}B_{i+1} \leq 0$$
$$A_i(B_{i+1} - B_i) - A_{i+1}(B_{i+1} - B_i) \leq 0$$
$$A_i \leq A_{i+1} \quad \text{কারণ, } B_{i+1} - B_i > 0$$

আসলেই তাই! (ইমপ্লিকেশন গুলো উল্টা অর্ডারে লিখতে হবে আরকি ফর্মাল প্রুফে...) তাহলে আমরা প্রুফ করে ফেললাম- এভাবে সোয়াপ করতে থাকলে আমরা কোন লস ছাড়াই অপ্টিমাল সলিউশন থেকে গ্রিডি সলিউশনে পৌছাতে পারবো (খেয়াল করো, শুধুমাত্র দুটো পাশাপাশি উপাদান সোয়াপ করে করেই কিন্তু একটি সিকুয়েন্সের যেকোনো পারমুটেশনে পৌছনো যায়)। অর্থাৎ, আমাদের গ্রিডি সলিউশনও একটি অপ্টিমাল সলিউশন! ■

§ ৫.২ মুল টেকনিক

গ্রিডি অ্যালগরিদম বের করার পরে তা এক্সচেঞ্জ আর্গুমেন্ট দিয়ে প্রমাণ করার জন্য আমরা যা করি তাকে মূলত নিচের ৩টা স্টেপে ভাগ করা যায়-

১. ধরলাম আমাদের গ্রিডি অ্যালগরিদম ব্যবহার করে আমরা একটা সলিউশন $G = \{g_1, g_2, \ldots, g_n\}$ পেয়েছি, আর $O = \{o_1, o_2, \ldots, o_m\}$ একটি অপ্টিমাল সলিউশন। এখানে কিন্তু আমরা ধরে নিচ্ছি G আর O দুটোই সবরকমের শর্ত মেনেই বানানো হয়েছে।

২. ধরে নাও $G \neq O$ আর তাদের মধ্যে পার্থক্য করো, যেমন, ধর G তে এমন একটি উপাদান পেলে যেটি O তে নেই (অথবা, O তে এমন একটি উপাদান পেলে যেটি G তে নেই) অথবা এমন দুটি উপাদান আছে যারা G তে যেই অর্ডারে আছে, O তে তার বিপরীত অর্ডারে আছে।

৩. **এক্সচেঞ্জ।** যেমন, প্রথম কেইস এর জন্য O থেকে একটি উপাদান বের করে আরেকটি উপাদান ঢুকালা, অথবা দ্বিতীয় কেইস এর জন্য অর্ডারটা সোয়াপ করে দিলে (বেশিরভাগ সময় খালি পাশাপাশি ২টা উপাদান নিয়েই কাজ করা হয়)। এখন কারণ দেখাও, এক্সচেঞ্জ করার পর তোমার নতুন সলিউশনটা আগেরটার তুলোনায় খারাপ না এবং এরপর দেখাবে তুমি যদি এইরকম এক্সচেঞ্জ করতে থাকো তাহলে একসময় O কে G এর সমান বানাতে পারবে। সুতরাং তোমার গ্রিডি সলিউশন যেকোনো অপ্টিমাল সলিউশনের (বা যেকোনো নন-অপ্টিমাল সলিউশনের) চাইতে ভাল বা সমান, যার মানে দাঁড়ালো তোমার সলিউশনও একটি অপ্টিমাল সলিউশন।

অনেক ভারী ভারী আলোচনা হয়ে গেলো! আসলে প্রথমেই যে বলেছিলাম এক্সচেঞ্জ আর্গুমেন্ট দিয়ে প্রুফ করতে গিয়ে আমরা অনেকসময় গ্রিডি সলিউশনও দাঁড় করিয়ে ফেলতে পারি- এভাবে চিন্তা করলে আমরা কিছু কন্ডিশন পাই (যেমন পাশাপাশি ২টা উপাদানের মধ্যে কিরকম সম্পর্ক হতে পারে) এবং সেগুলো থেকে আমরা উপাদান গুলোর একটি অর্ডারিং পেতে পারি যেটা আমাদের কাজকে অনেক সহজ করে দেয়। আশা করি পরের অংশের উদাহরণগুলো দেখলে বিষয়টা পরিষ্কার হবে।

পড়া থামাও, নিজে চেষ্টা করো

দুটি অ্যারে দেওয়া আছে (একই উপাদান বার বার থাকতে পারে)। অ্যারে দুটির উপাদানের মাল্টিসেট গুলো সমান, অর্থাৎ, এদেরকে সর্ট করলে অ্যারে দুটি একই হবে। তুমি প্রতি ধাপে প্রথম অ্যারেটির দুটি পাশাপাশি উপাদান সোয়াপ করতে পারবা। মিনিমাম কয়টি মুভে প্রথম অ্যারেটিকে তুমি দ্বিতীয় অ্যারের সমান করতে পারবে তা বের করতে হবে।

§ ৫.৩ ডিপির সাথে সম্পর্ক

সমস্যা ৫.২: তোমাকে দুটি N সাইজের বাইনারি অ্যারে A আর B দেওয়া আছে। তুমি প্রতি ধাপে নিচের যেকোনো একটি অপারেশন A অ্যারের উপর প্রয়োগ করতে পারবা-

১. **সেট অপারেশনঃ** একটি রেঞ্জ $[l, r]$ যেখানে $1 \leq l \leq r \leq N$ বাছাই করে $A[l \ldots r]$ এর সব মান 0 করে দিবে।

২. **রিসেট অপারেশনঃ** একটি রেঞ্জ $[l, r]$ যেখানে $1 \leq l \leq r \leq N$ বাছাই করে $A[l \ldots r]$ এর সব মান 1 করে দিবে।

৩. **ফ্লিপ অপারেশনঃ** একটি রেঞ্জ $[l, r]$ যেখানে $1 \leq l \leq r \leq N$ বাছাই করে $A[l \ldots r]$ এর সব মান পরিবর্তন করে দিবে (০ থাকলে ১ আর ১ থাকলে ০ করতে হবে)।

তোমাকে বের করতে হবে মিনিমাম কয়টি অপারেশনে তুমি A অ্যারেকে B এর সমান করতে পারবে।

সমাধান: যদিও বেশিরভাগ অপটিমাইজেশন প্রবলেমই হয় গ্রিডি না হয় ডিপি হয়, তাও কেও যদি এইধরনের প্রবলেম আগে কখনো না দেখে থাকে তাহলে এটা যে আদৌ ডিপি প্রবলেম, তা আন্দাজ করারও উপায় আছে বলে আমি মনে করি না। প্রবলেমটা সম্পর্কে কিছু আইডিয়া পাওয়ার জন্য আমরা একটি মিনিমাম অপারেশনের সিকুয়েন্স কেমন হতে পারে তা চিন্তা করতে পারি। ধরো এমন একটা সিকুয়েন্স হলো $o_1, o_2, \ldots, o_k$ (তাহলে k হলো আমাদের উত্তর, আর, একটা অপারেশনকে আমরা একটা টুপল $o_i = (l_i, r_i, \star_i)$ দিয়ে বর্ণনা করবো)। এখন আমরা একটু খতিয়ে দেখবো, একটা অপারেশনের ওপর আরেকটা অপারেশনের প্রভাব কি হতে পারে। ২টা অপারেশন o_i আর o_j নাও $(i < j)$। এখন দেখো, যদি $j > i+1$ হয় তাহলে ঐ ২টি অপারেশনের মাঝে আরও অনেক অপারেশন এসে যাচ্ছে, যেগুলো আমাদের চিন্তাকে জটিল করে ফেলছে। তাই, আমরা আপাতত $j = i+1$ ধরি অর্থাৎ o_i আর o_{i+1} নিয়ে চিন্তা করবো এখন। আমরা এবার এই অপারেশন দুটো কোনোভাবে কম্বাইন করে একটি অপারেশন বানানোর চেষ্টা করবো যাতে আমাদের অপারেশনের সংখ্যা

কমে যায়। কিন্তু আমরা তো একটা মিনিমাম সাইজের সিকুয়েন্স নিয়েছিলাম! হ্যাঁ, আমরা যদি ঐ ২টা অপারেশন কম্বাইন করতে পারি, তাহলে এমন বৈশিষ্ট্যের ২টি অপারেশন আমরা কোন অপ্টিমাল সিকুয়েন্সে পাশাপাশি পাবো না। এভাবে আমরা কিরকম বৈশিষ্ট্য একটি অপ্টিমাল সিকুয়েন্সে থাকবে আর কিরকম বৈশিষ্ট্য থাকবে না তা সম্পর্কে ধারনা পেতে পারি। কয়েকটা কেইস আছে-

- $\star_i = \oplus, \star_{i+1} = \oplus$[১]। প্রথমেই সবচেয়ে সহজটা দেখা যাক। দুটি রেঞ্জের জন্য সবরকমের অপশন এঁকে দেখতে পারো, যেমন- এমটা রেঞ্জের ভিতর আরেকটা অথবা একটার ভিতর আরেকটা সম্পূর্ণ না থেকে ওভারল্যাপ করছে ইত্যাদি। যদি রেঞ্জ দুটি একে-অপরকে ছেদই না করে তাহলে তো আমাদের আর তেমন কিছু করার নেই। কিন্তু সবকিছু সাজিয়ে রাখার জন্য আমরা যেটা করতে পারি তা হলো- যদি $l_i > l_{i+1}$ হয় তাহলে তাদের সোয়াপ করে দিতে পারি। আমরা এখন থেকে যখনই পারি, l এর এরকম Non-decreasing অর্ডার ঠিক রাখার চেষ্টা করবো।

- $\star_i = \oplus, \star_{i+1} = 1$। রেঞ্জগুলো যদি ওভারল্যাপ না করে তাহলে আগের মতই তেমন কিছু করতে হবে না। কিন্তু আমাদের সুবিধার জন্য আমরা সেট অপারেশনটাকে আগে নিয়ে আসতে পারি আর টগল অপারেশনটাকে পরে নিয়ে যেতে পারি। খেয়াল করো, আমাদের এই ট্রান্সফর্মেশনের পরেও কিন্তু ফাইনাল অ্যারে একই থাকছে। আর টগল অপারেশনটাকে পরে নেওয়ার কারণ হলো সেট বা রিসেট অপারেশনের চাইতে টগল অপারেশনে আমরা এক দিক দিয়ে বেশি অপশন পাই। এখন, রেঞ্জগুলো যদি ওভারল্যাপ করে তাহলে কি হবে? চিন্তা করে দেখো, আমরা কিন্তু প্রথমে o_i এর রেঞ্জে রিসেট অপারেশন অ্যাপ্লাই করে তারপর $[l_i, r_i] \cup [l_{i+1}, r_{i+1}]$ রেঞ্জে টগল অপারেশন অ্যাপ্লাই করতে পারি; ফাইনাল অ্যারে একই থাকবে।

- $\star_i = \oplus, \star_{i+1} = 0$। আগের কেইসের মত এখানেও প্রথম অপারেশনটিকে সেট এবং পরের অপারেশনটিকে টগল বানানো যায়।

---

[১] $\oplus$ দিয়ে টগল, 1 দিয়ে সেট এবং 0 দিয়ে রিসেট অপারেশন বুঝানো হয়েছে

- বাকি কেইস গুলাতে আসলে সব রেঞ্জগুলো আলাদা আলাদা (disjoint) করে ফেলা যায়। এরপর না হয় আগে সেট অপারেশন এবং পরে রিসেট অপারেশন- এইরকম অর্ডার ঠিক রাখলাম।

উপরের কেইসগুলোতে প্রথমে সেট বা রিসেট অপারেশন রেখে এবং পরে টগল অপারেশন রেখে বিবেচনা করা হয়নি কারণ আমরা এমনিতেই চাচ্ছি টগল অপারেশনকে পরে পাঠাতে।
উপরের ঘাঁটাঘাঁটি থেকে আমরা এই অবজারভেশন পাই- অন্তত একটি এমন অপ্টিমাল সলিউশন আছে যেটাতে সব সেট অপারেশন আগে, তারপর সব রিসেট অপারেশন এবং শেষে সব টগল অপারেশন থাকবে। যদিও আমাদের কাছে কোনো গ্রিডি সলিউশন বা তেমন কিছু জানা ছিল না, তারপরও আমরা সেই এক্সচেঞ্জ আর্গুমেন্ট এর ধাপ গুলোই প্রয়োগ করার চেষ্টা করেই এমন গুরুত্বপূর্ণ অবজারভেশন পেয়ে গেলাম। এখন আমাদের বাকি এই অবজারভেশনের সাথে ইন্টারভাল ডিপি এবং বিটমাস্ক ডিপির সমন্বয় করে একটা ডিপি সলিউশন দাঁড় করানো। এখানে একটি খেয়াল করার বিষয় হলো, আমরা এই অবজারভেশন বের করতে দিয়ে আরও কিছু অপ্রয়োজনীয় কাজ করেছি, যেমন- প্রথম কেইসে l দ্বারা অর্ডারিং করা। আসলে আমরা অনেকসময়ই এরকম করে থাকি (যেমন আমাদের একটি অ্যারে দেওয়া থাকলে আর অ্যারের উপাদানগুলো যদি যেকোনো ক্রমে নিয়ে কাজ করা যায় তাহলে আমরা ধরে নেই অ্যারেটা সর্টেড আছে) কারণ সবকিছু সাজানো গুছানো থাকলে চিন্তা করতে সুবিধা হয়। এটা একটা সাধারণ প্রবলেম সলিভং স্ট্রেটেজি।

■

অধ্যায় ৬

পলিনমিয়াল ইন্টারপোলেশন

§ ৬.১ পলিনমিয়াল নিয়ে কিছু কথা

তোমরা বহুপদী বা পলিনমিয়াল নিয়ে আগে হয়ত কাজ করেছ। সবচেয়ে বহুল প্রচলিত উদাহরণ হচ্ছে দ্বিঘাতী সমীকরণগুলো। যেমন ধর

$$2x^2 + 5x - 15$$

এটি একটি দ্বিঘাতী পলিনমিয়াল (second degree)। আবার নিচের পলিনমিয়ালটি একটি ত্রিঘাতী পলিনমিয়াল (third degree)

$$x^3 - 5x^2 + 2x + 3$$

সাধারণভাবে বলতে গেলে

$$P(x) = \sum_{i=0}^{n} a_i x^i = a_n x^n + a_{n-1} x^{n-1} + \cdots + a_1 x + a_0$$

একটি n ঘাতী পলিনমিয়াল (n th degree)। পাঠ্যবইয়ের ভাষায় বলতে গেলে একটি n ঘাতী পলিনমিয়াল হল এমন একটি এক চলক বিশিষ্ট ফাংশন যার ঘাতগুলো অঋণাত্মক পুর্ণসংখ্যা এবং সর্বোচ্চ ঘাত n।

পলিনমিয়াল কী তা হয়ত সবাই বুঝতে পেরেছ। কিন্তু পলিনমিয়াল ইন্টারপোলেশন বলতে আসলে কি বুঝাচ্ছে। আমরা জানি পলিনমিয়ালগুলো বিশেষ ধরনের ফাংশন। ধর আমাদের একটা অজানা পলিনমিয়াল $P(x)$ বের করতে হবে। শুধু জানা আছে $P(x)$ একটি তৃতীয় ঘাতী পলিনমিয়াল, এবং

$$P(1) = 21$$

$$P(3) = 4$$
$$P(10) = -5$$
$$P(15) = -8$$

শুধু এটুকু জানলেই কী $P(x)$ কে বের করে ফেলা সম্ভব? উত্তর হচ্ছে হ্যাঁ। সাধারণভাবে বলা যায়, যদি আমরা পলিনমিয়ালের ডিগ্রি বা ঘাত সম্পর্কে জানি (ধর এই ডিগ্রি n), এবং $n+1$ টি **ভিন্ন ভিন্ন** x এর জন্য $P(x)$ এর মান জানি, তাহলে আমরা পলিনমিয়ালটিকে বের করে ফেলতে পারব (শুধু তাই নয়, সব শর্ত মেনে চলে এমন পলিনমিয়াল একটাই পাওয়া যাবে)। এই যে $n+1$ টি $P(x)$ এর মান থেকে আমরা n ডিগ্রি পলিনমিয়ালটিকে বের করে ফেললাম এই প্রসেসটাকেই বাল হয় পলিনমিয়াল ইন্টারপোলেশন।

পরবর্তী সেকশনে যাওয়ার আগে পলিনমিয়ালের ডিগ্রির ব্যাপারে কিছু কথা বলে নেওয়া দরকার। যদিও এগুলো সবারই জানার কথা, তবুও পরবর্তীতে এটা অনেক জায়গায় কাজে লাগবে বলে আবার বলছি

১. একটি n ডিগ্রি পলিনমিয়ালের সাথে আরেকটা m ডিগ্রি পলিনমিয়াল যোগ করলে যোগফলের ডিগ্রি হবে $\max(n, m)$।

২. একটি n ডিগ্রি পলিনমিয়ালের সাথে আরেকটা m ডিগ্রি পলিনমিয়াল বিয়োগ করলে বিয়োগফলের সর্বোচ্চ ডিগ্রি হবে $\max(n, m)$। তবে এর চেয়ে কমও হতে পারে।

৩. একটি n ডিগ্রি পলিনমিয়ালের সাথে আরেকটা m ডিগ্রি পলিনমিয়াল গুন করলে গুনফলের ডিগ্রি হবে $n+m$।

§ ৬.২ কীভাবে পলিনমিয়াল ইন্টারপোলেশন কাজ করে

কীভাবে পলিনমিয়ালটাকে বের করতে পারব সেটা বুঝার জন্য শুরুতেই একটা সহজ উদাহরণ দেখা যাক।

উদাহরণ ৬.১

এমন দ্বিঘাতী পলিনমিয়াল বের কর যেন

$$P(1) = -3$$
$$P(4) = 0$$
$$P(5) = 0$$

সমাধান: তোমরা এটা নিশ্চয় জানো যদি কোন বহুপদী বা পলিনমিয়াল $f(x)$ এর জন্য $f(a) = 0$ হয় তাহলে $(x - a)$ পলিনমিয়ালটির একটি উৎপাদক। আমরা এ জিনিশটিই এখানে ব্যবহার করব। প্রশ্ন অনুযায়ী

$$P(4) = 0$$
$$P(5) = 0$$

তার মানে $(x-4)$ এবং $(x-5)$ উভয়েই $P(x)$ এর উৎপাদক। তাই আমরা $P(x)$ কে এভাবে লিখতে পারি

$$P(x) = (x - 4)(x - 5)Q$$

এখানে Q কিন্তু একটি ধ্রুবক হবে। কারণ হল $(x - 4)$ এবং $(x - 5)$ এর গুণফল নিজেই একটি দ্বিঘাতী পলিনমিয়াল। তাই Q এর ঘাত শূন্য হতে হবে (উভয় পাশে ডিগ্রি বা ঘাত সমান রাখার জন্য), অর্থাৎ Q কে একটি ধ্রুবকই হতে হবে। উপরের সমীকরণে আমরা $x = 1$ বসালেই কিন্তু Q এর মান বের করে ফেলতে পারব

$$\begin{aligned} P(1) = (1 - 4)(1 - 5)Q &= -3 \\ \Rightarrow Q &= \frac{-3}{12} \end{aligned}$$

সুতরাং $P(x)$ এর মান হচ্ছে

$$P(x) = \frac{-3}{12}(x-4)(x-5)$$

এখন একে বিস্তার (expand) করে দিলেই $P(x)$ এর সব সহগগুলো বের করে ফেলতে পারব। ■

এবার আরেকটু কঠিন উদাহরণ দেখা যাক

উদাহরণ ৬.২

এমন দ্বিঘাতী পলিনমিয়াল বের কর যেন

$$P(1) = -1$$
$$P(2) = -5$$
$$P(3) = 3$$

সমাধান: আগের উদাহরণটি আমাদের জন্য সহজ হয়ে গিয়েছিল কেন বল তো? কারণ ছিল একটি বাদে বাকি $P(x)$ গুলোর মান 0 ছিল। তাই আমরা $P(x)$ এর সব উৎপাদক বের করে ফেলতে পেরেছিলাম। কিন্তু এখানে কোন $P(x) = 0$ নেই। তাহলে কী করা যায়?

আমরা কিছুটা আগের উদাহরণের মতই চেষ্টা করব। ধরে নাও, শুধু $P(x) = -1$ বাকি $P(x)$ গুলোর মান 0 (অর্থাৎ $P(2) = P(3) = 0$)। তাহলে আমরা আগের উদাহরণের মত একটি পলিনমিয়াম বের করতে পারব। এই পলিনমিয়ালের নাম দিলাম P_1।

একইভাবে এবার ধর শুধু $P(2) = -5$, বাকি $P(x)$ গুলোর মান 0 (অর্থাৎ $P(1) = P(3) = 0$)। এবারও আরেকটি পলিনমিয়াল P_2 বের হবে।

শেষমেষ তৃতীয় পলিনমিয়াল P_3 বের করার জন্য $P(3) = 3$ এবং $P(1) = P(2) = 0$ ধরে নিয়ে সমধান করতে হবে। এভাবে আমরা তিনটি পলিনমিয়াল P_1, P_2, P_3 পেলাম।

আমাদের কাজ কিন্তু প্রায় শেষ। এখন পলিনমিয়াল তিনটিকে যোগ করে দিলেই

কাঙ্ক্ষিত পলিনমিয়ালটি পেয়ে যাব। অর্থাৎ

$$P = P_1 + P_2 + P_3$$

এর কারণও খুব সহজ।

$$\begin{aligned}
P(1) = P_1(1) + P_2(1) + P_3(1) = (-1) + 0 + 0 = &\quad -1 \\
P(2) = P_1(2) + P_2(2) + P_3(2) = 0 + (-5) + 0 = &\quad -5 \\
P(3) = P_1(3) + P_2(3) + P_3(3) = 0 + 0 + (+3) = &\quad 3
\end{aligned}$$

P_1, P_2, P_3 সবগুলোই 2 ডিগ্রি পলিনমিয়াল হওয়ায় P ও 2 ডিগ্রি পলিনমিয়াল হবে। অর্থাৎ যেহেতু P সব শর্ত সিদ্ধ করে করে, তাই এটিই নির্ণেয় উত্তর। ■

এখানে আমরা দ্বিঘাতী পলিনমিয়ালের জন্য ইন্টারপোলেশন করেছি। কিন্তু একই নিয়মে উপরের ঘাতের পলিনমিয়ালগুলোর জন্যও ইন্টারপোলেশন করা যাবে।

§ ৬.৩ ল্যাগ্রাঞ্জ ইন্টারপোলেশন

আমরা কিন্তু ল্যাগ্রাঞ্জ ইন্টারপলেশন ইতোমধ্যে শিখে ফেলেছি। আগের উদাহরণগুলোয় আমরা যেভাবে পলিনমিয়ালটা বের করেছি সেটার প্রচলিত নাম হচ্ছে ল্যাগ্রাঞ্জ ইন্টারপোলেশন। n ডিগ্রি পলিনমিয়ালের জন্য আমাদের ইন্টারপোলেশন করতে হবে এভাবে: যদি

$$\begin{aligned}
&P(x_0) = y_0 \\
&P(x_1) = y_1 \\
&P(x_2) = y_2 \\
&\vdots \\
&P(x_n) = y_n
\end{aligned}$$

হয়, তাহলে n ডিগ্রি পলিনমিয়াল $P(x)$ বের করার জন্য

→ প্রথমে প্রত্যেক i এর জন্য $P(x_i) = y_i$ এবং $P(x_j) = 0$ (যেখানে $i \neq j$) ধরে নিয়ে একটি পলিনমিয়াল বের করতে হবে। অর্থাৎ আমরা এভাবে $n + 1$ টি n ডিগ্রি পলিনমিয়াল পাব। আগের উদাহরণটির মত যদি সমাধান কর তাইলে দেখবে i তম পলিনমিয়াল P_i হবে

$$P_i(x) = y_i \times \prod_{\substack{j=0 \\ i \neq j}}^{n} \frac{x - x_j}{x_i - x_j}$$

→ এরপর প্রত্যেক পলিনমিয়ালকে বিস্তার করে দাও (এ কাজটি ফাস্ট ফুরিয়ার ট্রান্সফর্ম দিয়ে করা যায়; তবে আমাদের বইয়ের আলোচনার জন্য এটি দরকার নেই, $\mathcal{O}(n^2)$ কম্পেক্সিটিতে বিস্তার করাই যথেষ্ট)।

→ শেষ ধাপে আমাদের $n + 1$ টি পলিনমিয়াল যোগ করে দিতে হবে। যোগফলই হবে আমাদের কাঙ্ক্ষিত পলিনমিয়াল। অর্থাৎ

$$P(x) = \sum_{i=0}^{n} P_i(x)$$

এটাই ল্যাগ্রাঞ্জ ইন্টারপলেশনের অ্যালগরিদম।

§ ৬.৪ ডাইনামিক প্রোগ্রামিং-এর সাথে সম্পর্ক