ডাইনামিক প্রোগ্রামিং এ হাতেখড়ি

তাসমীম রেজা
মামনুন সিয়াম

ড্রাফ্‌ট ২৩ জুলাই ২০২১

অধ্যায় 1

কিছু ব্রুটফোর্স, ব্যাকট্র্যাকিং এবং বিটমাস্ক ট্রিকস

সরাসরি ডাইনামিক প্রোগ্রামিং শুরু না করে আমরা যেকোনো কিছু ব্রুটফোর্স করে কিভাবে সমাধান করা যায় তা দেখা যাক। যেমন আমাদের কোন সমস্যায় মিনিমাম কস্ট বের করতে বলা হলে আমরা সবধরনের অ্যারেঞ্জমেন্ট ট্রাই করবো আর যেসব অ্যারেঞ্জমেন্ট প্রবলেমে দেওয়া শর্ত পূরণ করে সেগুলোর জন্য মিনিমাম কস্ট বের করে আমাদের ফাইনাল অ্যান্সার আপডেট করবো। এইধরনের চিন্তাধারা আমাদের সবচেয়ে বেশি কাজে লাগবে যেসব কাউন্টিং প্রবলেম ডিপি দিয়ে সল্ভ করতে হয় সেগুলো সল্ভ করার বেলায়। যদি তোমার আগে থেকে জানা না থাকে, তাহলে দেরি না করে এরকম কিছু ব্রুটফর্স টেকনিক দেখে নেয়াও যাক।

1.1 একটুখানি বিট

তোমাদের নিশ্চয়ই জানা আছে কম্পিউটার সবকিছু ০ আর ১ দিয়ে হিসাব করে। যেমন, `int` ডাটা টাইপে ৩২টা বিট স্টোর থাকে। যদিও, যেকোনো ম্যাথম্যাটিক্যাল অপারেটর (যেমন, যোগ, বিয়োগ, গুন, ভাগ ইত্যাদি) গুলোও বিটগুলো নিয়ে কাজ করে, এই অপারেটর গুলো ছাড়াও আরও কয়েকটি অপারেটর আছে যেগুলো ব্যবহার করে আমরা আমাদের ইমপ্লিমেন্টেশনকে অনেক সহজ আর সুন্দর করে ফেলতে পারি। সেগুলো দেখবো আমরা এখন।

1.1.1 কম্পিউটার কিভাবে সংখ্যা স্টোর রাখে?

`int` ডাটা টাইপে 59 নাম্বারটি এইভাবে স্টোর থাকেঃ

$$\texttt{00000000000000000000000000111011}$$

শুরুর দিকে সব ০ থাকার কারণ হচ্ছে, যদিও ৫৯ কে বাইনারিতে প্রকাশ করতে আমাদের ঐ বিটগুলো দরকার হচ্ছে না, তারপরও যেহেতু `int` ডাটা টাইপ ৩২-বিটের, তাই ঐ বিট গুলোতে ০ সেভ রাখা হচ্ছে।

বিটগুলো নাম্বারিং করা হয় ডানপাশ থেকে বামপাশে। যেমন, কোন সংখ্যা b এর i-তম বিটকে যদি আমরা b_i দিয়ে প্রকাশ করি তাহলে সংখ্যাটিকে বাইনারিতে লেখা হবে এইভাবেঃ $\overline{b_{u-1}\dots b_2b_1b_0}$,

যেখানে u হচ্ছে ডাটা টাইপের লেংথ। আর এই বাইনারিকে দশমিকে নিতে হলে আমরা এই ফরমুলা ব্যবহার করতে পারিঃ $b_{u-1}2^{u-1} + \ldots + b_2 2^2 + b_1 2^1 + b_0 2^0$।

ডাটা টাইপ আবার দুইধরনের হতে পারে, Signed এবং Unsigned (যেমন, `int`, `unsigned int`)। Signed ডাটা টাইপে ঋণাত্মক আর অঋণাত্মক সংখ্যা স্টোর রাখা এবং হিসাব নিকাশ করার জন্য 2's complement ব্যবহার করা হয়। একটা u সাইজের signed ডাটা টাইপের ক্ষেত্রে যেকোনো সংখ্যা x এর 2's Complement x' কে এমনভাবে ডিফাইন করা হয় যেন তা নিচের শর্ত পূরণ করেঃ

$$x + x' = 2^u$$

। এই x' কেই কম্পিউটার $-x$ হিসেবে চিনে। এটা করে লাভ কি হলো? খেয়াল করো, $x + (-x)$ করার পরে কিন্তু কম্পিউটার যেটা পাচ্ছে তা হলো 2^u (অর্থাৎ, u-তম বিট অন শুধু, বাকি সব ০)। কিন্তু u সাইজের একটা ডাটা টাইপ তো শুধু $u-1, u-2, \ldots, 2, 1, 0$ বিট গুলো স্টোর রাখতে পারে! তাহলে সে আসলে ঐ u-তম বিটটা ফেলে দিবে আর শেষপর্যন্ত সে যেটা সেভ রাখবে সেটার সব বিট অফ হবে – অর্থাৎ শুন্য। তাই তো হওয়ার কথা! একটা সংখ্যার সাথে তার যোগাত্বক বিপরীত সংখ্যা যোগ করলে তও শুন্যই পাওয়ার কথা। তুমি যদি একটু চিন্তা করে দেখো, তাহলে দেখবে, দুটি সংখ্যা x আর y দিয়ে কম্পিউটারকে যদি বলা হয় $x - y$ হিসাব করতে, তাহলে সে কিন্তু x এর সাথে y' যোগ করে দিয়েই বিয়োগফল বলে দিতে পারবে! আর বাইনারিতে যোগ করা তও সোজা।

1.1.2 বিট অপারেশনসমূহ

And অপারেশন

দুটো সংখ্যা x আর y এর and অপারেশন x & y এমন একটা সংখ্যা বের করবে যেটার বাইনারিতে i-তম বিট অন থাকবে যদি ও কেবল যদি x আর y উভয়ের i-তম বিট অন থাকে। যেমন `207 & 158 = 142`।

$$\begin{array}{ccc} & 11001111 & (207) \\ \& & 10011110 & (158) \\ \hline = & 10001110 & (142) \end{array}$$

Or অপারেশন

দুটো সংখ্যা x আর y এর or অপারেশন x | y এমন একটা সংখ্যা বের করবে যেটার বাইনারিতে i-তম বিট অন থাকবে যদি ও কেবল যদি x এবং y এর অন্তত একটির i-তম বিট অন থাকে। যেমন `79 | 44 = 111`।

$$\begin{array}{ccc} & 01001111 & (79) \\ | & 00101100 & (44) \\ \hline = & 01101111 & (111) \end{array}$$

Xor অপারেশন

দুটো সংখ্যা x আর y এর xor অপারেশন $x\hat{}y$ এমন একটা সংখ্যা বের করবে যেটার বাইনারিতে i-তম বিট অন থাকবে যদি ও কেবল যদি x এবং y এর মধ্যে বরাবর একটিতে i-তম বিট অন থাকে। যেমন `245 ^ 67 = 182`। Xor অপারেটরকে ম্যাথেম্যাটিক্যালি অনেকসময় $\oplus$ দিয়েও লেখা হয়।

$$\begin{array}{ccc} & 11110101 & (245) \\ \hat{} & 01000011 & (67) \\ \hline = & 10110110 & (182) \end{array}$$

Not অপারেশন

কোন সংখ্যা x এর উপর Not অপারেশন $(\sim x)$ অ্যাপ্লাই করলে এমন একটা সংখ্যা পাওয়া যায় যার প্রত্যেকটা বিট x এর উল্টা। যেমন, 16-bit ডাটা টাইপের জন্যঃ

$$\begin{array}{rcrl} \sim x & = & 14977 & 0011101010000001 \\ \sim x & = & -14978 & 1100010101111110 \end{array}$$

চিন্তা করে দেখো এই ফরমুলাটা কেন কাজ করেঃ $-x = \sim x + 1$।

বিট শিফট

```
int ~someShit;
int y = a ^ b;
int fhat = 0;
```

উদাহরণ 1.1. তোমাকে একটি n সাইজের অঋণাত্মক সংখ্যার অ্যারে a $(1 \leq n \leq 20, 0 \leq a_i \leq 10^9)$ দেওয়া হয়েছে, তোমাকে বলতে হবে ঐ অ্যারে এর একটি উপাদান সর্বোচ্চ একবার নিয়ে কোন কোন যোগফল বানানো যায়।

অধ্যায় 2

ম্যাট্রিক্স এক্সপোনেন্সিয়েশন

2.1 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_{ij} দিয়ে প্রকাশ করা হয়। যেমন উপরের ম্যাট্রিক্সের জন্য $A_{11} = 1$, আবার $A_{23} = 7$। 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 $C_{ij} = A_{ij} + B_{ij}$ 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 $m \times l$ ম্যাট্রিক্স গুন করতে পারবে এবং গুণফল হবে একটা $n \times l$ ম্যাট্রিক্স। অর্থাৎ প্রথম ম্যাট্রিক্সের কলাম সংখ্যা আর দ্বিতীয় ম্যাট্রিক্সের সারি সংখ্যা সমান হতে হবে। C যদি A এবং B ম্যাট্রিক্সের গুণফল হয় তাহলে

$$C_{ij} = \sum_{k=1}^{m} A_{ik} B_{kj} \tag{2.1.১}$$

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 হয়ত (2.1.১) সমীকরণ দিয়ে ভালভাবে কল্পনা করা একটু কঠিন। এজন্য আমাদের ভেক্টর-ভেক্টর গুণফল ভালভাবে বুঝতে হবে আগে।

2.2 ভেক্টর-ভেক্টর গুণফল

$n \times 1$ বা $1 \times n$ আকারের ম্যাট্রিক্সগুলোর একটি বিশেষ নাম আছে। এদের কে ভেক্টর বলা হয়। স্বভাবতই, $1 \times n$ ম্যাট্রিক্স রো ভেক্টর (row vector) নামে পরিচিত, কারণ এটি অনেকটা রো এর মতই দেখতে। একই ভাবে $n \times 1$ ম্যাট্রিক্স কলাম ভেক্টর (column vector) নামে পরিচিত, কারণ এটি অনেকটা কলামের মত দেখতে। সাইজ দেখেই বুঝতে পারছ, n সাইজের একটি রো ভেক্টর এর সাথে n সাইজের একটি কলাম ভেক্টর গুন করলে 1×1 ম্যাট্রিক্স পাওয়া যাবে। এই 1×1 ম্যাট্রিক্সকে ম্যাট্রিক্স না বলে একটা সংখ্যা হিসেবেই কল্পনা করা যায়। এই যে আমরা একটা রো ভেক্টর এর সাথে কলাম ভেক্টরের গুন করলাম এটারও একটা বিশেষ নাম আছে কিন্তু। এটাকেই বলা হয় ম্যাট্রিক্সের ডট প্রডাক্ট। এই গুণফলকে সংজ্ঞায়িত করা হয়েছে এভাবে:

$$\begin{pmatrix}a_1 & a_2 & a_3\end{pmatrix}\begin{pmatrix}b_1\\ b_2\\ b_3\end{pmatrix} = a_1b_1 + a_2b_2 + a_3b_3$$

এখানে আমরা 3 সাইজের ভেক্টর এর জন্য দেখলাম, কিন্তু অন্য ভেক্টর এর জন্যও একি ভাবে বের করা যাবে। সোজা কথায় রো ভেক্টরের i তম সংখ্যার সাথে কলাম ভেক্টরের i তম সংখ্যা গুন দিয়ে সবগুলোর যোগফল নিলেই হবে। আমরা একটু আগে যে ম্যাট্রিক্স গুণফল শিখেছিলাম তার চেয়ে কিন্তু এটা ভিজুয়ালাইজ করা বেশ সহজ।

একটা জিনিশ খেয়াল কর। একটি $n \times m$ ম্যাট্রিক্স কিন্তু n টা রো ভেক্টর নিচে নিচে সাজালেই পাওয়া যাবে। একইভাবে একটি $n \times m$ ম্যাট্রিক্সকে m টি কলাম ভেক্টর পাশাপাশি সাজালেই পাওয়া যায়। অর্থাত যেকোনো ম্যাট্রিক্সকেই কিছু রো ভেক্টর বা কিছু কলাম ভেক্টর এর সমাহার হিসেবে চিন্তা করা যায়।

এবার আমরা ম্যাট্রিক্স গুনকে একটু ভিন্ন ভাবে দেখতে পারি। A এর i তম রো এবং B এর j তম কলাম ডট গুন করলেই আমরা AB এর (i, j) অবস্থানের মান বের করতে পারব। নিচের ম্যাট্রিক্সটি দেখ।

$$\begin{pmatrix}1 & 3 & 2\\ 9 & 0 & 7\end{pmatrix}\begin{pmatrix}5 & 6 & 0 & 3\\ 0 & 2 & -1 & 1\\ 1 & 1 & 4 & -1\end{pmatrix} = \begin{pmatrix}7 & 14 & 5 & 4\\ 52 & 61 & 28 & 20\end{pmatrix}$$

ধর আমরা গুণফলের $(2, 3)$ অবস্থানের মান বের করতে চাই। তাহলে বামপাশের ম্যাট্রিক্সের 2 তম রো এবং ডান পাশের ম্যাট্রিক্সের 3 তম কলাম নিব। ছবিতে রো আর কলাম দুটি মার্ক করে দিয়েছি। এবার এই রো ভেক্টর আর কলাম ভেক্টর গুন করলেই কাঙ্ক্ষিত সংখ্যাটি পেয়ে যাব।

$$\begin{pmatrix} 9 & 0 & 7 \end{pmatrix} \begin{pmatrix} 0 \\ -1 \\ 4 \end{pmatrix} = (9 \times 0) + (0 \times -1) + (7 \times 4) = 28$$

এখন চিন্তা করলে দেখ। (2.1.১) এ যে সূত্র লেখেছিলাম সেটা কিন্তু আসলে A এর i তম রো এবং B এর j তম কলামের ডট গুণনই করছে। অর্থাৎ দুটি আসলে একই জিনিশ। কিন্তু ভেক্টর ভেক্টর গুন ভালভাবে বুঝে গেলে ম্যাট্রিক্স গুনের পুরো প্রক্রিয়াটি ভিজুয়ালাইজ করা খুবই সহজ হয়ে যায়।

2.3 অ্যাসোসিয়েটিভিটি

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 BA এর সমান নয়। কোনটিকে আগে কোনটিকে পরে গুন করতে হবে তা লক্ষ্য রাখতে হবে।

2.4 ডাইনামিক প্রোগ্রামিং এর সাথে সম্পর্ক

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

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

তোমার মনে প্রশ্ন আসতে পারে, এই রিকারেন্স থেকে আবার ম্যাট্রিক্স আসলো কী করে? একটু মাথা খাটালে বুঝতে পারবে এরকম রিকারেন্সকে কিন্তু ম্যাট্রিক্স এর সাহায্যে প্রকাশ করা যায়।

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

এটা মনে হয় একটু বেশি সহজ হয়ে গেল। একটু জেনারেল কেইস নিয়ে চিন্তা করি। ধর আমাদের রিকারেন্সটি দেখতে এরকম:

$$f_n = a_1 f_{n-1} + a_2 f_{n-2} + a_3 f_{n-3} + \cdots + a_k f_{n-k} \tag{2.4.১}$$

এখানে $a_1, \ a_2, \ \cdots, \ a_k$ ধ্রুবক (যেমন ফিবোনাচ্চি রিকারেন্সে $a_1 = a_2 = 1$)। এই ধরনের রিকারেন্সের নাম লিনিয়ার রিকারেন্স। এই রিকারেন্সের ডিগ্রি k কারণ এখানে প্রতিটি পদ আগের k টি পদের ওপর নির্ভর করছে। সব ধরনের লিনিয়ার রিকারেন্স ম্যাট্রিক্স গুণফল দিয়ে প্রকাশ করা যায়। যেমন:

$$\begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_k \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \\ f_{n-3} \\ \vdots \\ f_{n-k} \end{pmatrix} = a_1 f_{n-1} + a_2 f_{n-2} + \cdots + a_k f_{n-k} = f_n \qquad (2.4.২)$$

এখন আমাদের কি টারগেট সেটা জানা দরকার। নিচের কলাম ভেক্টর দুটি দেখ। আমাদের টারগেট হল বাম পাশের ভেক্টরের সাথে একটি ম্যাট্রিক্স গুন করে ডান পাশের ভেক্টরটি পাওয়া।

$$\begin{pmatrix} f_{n-1} \\ f_{n-2} \\ f_{n-3} \\ \vdots \\ f_{n-k} \end{pmatrix} \rightarrow \begin{pmatrix} f_n \\ f_{n-1} \\ f_{n-2} \\ \vdots \\ f_{n-k+1} \end{pmatrix}$$

একটা k সাইজের কলাম ভেক্টর থেকে আরেকটা k সাইজের কলাম ভেক্টর পেতে চাইলে আমাদের অবশ্যই একটি $k \times k$ ম্যাট্রিক্স দিয়ে ভেক্টরটিকে বাম দিকে গুন করতে করতে হবে (অন্য আকার সম্ভব নয়। এটা নিজে প্রমাণ করার চেষ্টা কর)। অর্থাৎ সমীকরণটি দেখতে কিছুটা এমন হবে।

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

এখন তোমার এখানে পড়া থামিয়ে দাও। কিছুক্ষণ চিন্তা কর কিভাবে মাত্রিক্সটি বানানো যায়। এটা বেশ সহজই, তাই আমি বলব আগে নিজে কিছুক্ষণ চেষ্টা করতে।

যদি চেষ্টা করার পরে না বুঝতে পারো, তাহলে প্রথমে লক্ষ্য কর। প্রথম রো তে কিন্তু আমরা (২.৩) এর রো ভেক্টরটাই বসিয়ে দিতে পারি। অর্থাৎ ম্যাট্রিক্সটি এখন:

$$\begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_k \\ & & & & \\ & & & & \\ & & & & \\ & & & & \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \\ f_{n-3} \\ \vdots \\ f_{n-k} \end{pmatrix} = \begin{pmatrix} f_n \\ \\ \\ \\ \\ \end{pmatrix}$$

অর্থাৎ $f_{n-1}, f_{n-2}, \cdots, f_{n-k}$ থেকে আমরা f_n বানাতে পারলাম। আসল কাজ কিন্তু হয়ে গেছে। এখন আমাদের ভেক্টরটি থেকে $f_{n-1}, f_{n-2}, \cdots, f_{n-k+1}$ এগুলোর মান বের করতে হবে। কিন্তু এগুলো

ভেক্টরে অলরেডি আছে। যেমন f_{n-1} পেতে পারি এভাবে:

$$\begin{pmatrix} 1 & 0 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \\ f_{n-3} \\ \vdots \\ f_{n-k} \end{pmatrix} = f_{n-1}$$

আবার f_{n-2} পেতে চাইলে

$$\begin{pmatrix} 0 & 1 & 0 & \cdots & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \\ f_{n-3} \\ \vdots \\ f_{n-k} \end{pmatrix} = f_{n-2}$$

এই প্যাটার্ন ধরে আমরা পুরো ম্যাট্রিক্সটিই বানিয়ে ফেলতে পারব

$$\begin{pmatrix} a_1 & a_2 & a_3 & \cdots & a_{k-1} & a_k \\ 1 & 0 & 0 & \cdots & 0 & 0 \\ 0 & 1 & 0 & \cdots & 0 & 0 \\ \vdots & & & \ddots & & \\ 0 & 0 & 0 & \cdots & 1 & 0 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \\ f_{n-3} \\ \vdots \\ f_{n-k} \end{pmatrix} = \begin{pmatrix} f_n \\ f_{n-1} \\ f_{n-2} \\ \vdots \\ f_{n-k+1} \end{pmatrix} \qquad (2.4.৩)$$

ম্যাট্রিক্স এক্সপনেনশিয়েশন এর ম্যাট্রিক্স বানানো শিখে গিয়েছি আমরা!

2.5 ফিবোনাচ্চি ম্যাট্রিক্স

এবার আমরা ফিবোনাচ্চি ম্যাট্রিক্স বানানোর জন্য প্রস্তুত। আগের অংশে আমরা দেখিয়েছি ফিবোনাচ্চি রিকারেন্সটিকে এভাবে লেখা যায়

$$\begin{pmatrix} 1 & 1 \end{pmatrix} \begin{pmatrix} f_{n-1} \\ f_{n-2} \end{pmatrix} = f_n$$

আর আমরা এমন একটি ম্যাট্রিক্স A বানাতে চাই যেন

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

হয়। তাহলে (2.4.৩) অনুযায়ী A ম্যাট্রিক্সটি হবে

$$A = \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}$$

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। আগেই বামদিকের ম্যাট্রিক্স দুটো গুন না করে ডানদিকের ম্যাট্রিক্স আর ভেক্টর আগে গুন করে নিয়েছি। 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 $\mathcal{O}(k^3 \log n)$, যেখানে k হল আমাদের লিনিয়ার রিকারেন্সের ডিগ্রি।

2.6 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

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

মজার ব্যাপার হচ্ছে একটা ম্যাট্রিক্স দিয়েই একাধিক লিনিয়ার রিকারেন্স কে হ্যান্ডল করা যায়। এই ট্রিকটা এমন প্রবলেমগুলোতে লাগে যেখানে একের বেশি লিনিয়ার রিকারেন্স আছে এবং একটি রিকারেন্স আরেকটির ওপর নির্ভরশীল। নিচের উদাহরণ দেখলে বুঝবে।

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

ধরে নাও f_0, f_1, g_0, g_1 এর মান জানা আছে। অর্থাৎ এগুলো আমাদের বেস কেইস। 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 বের করে ফেলা যায়

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

আশা করি ম্যাট্রিক্স বানানো নিয়ে কারো কোন সমস্যা নেই আর।

প্রবলেম 2.1. 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

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

প্রবলেম 2.2. 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।

$$\begin{aligned} h_0 &= 0 \\ h_n &= h_{n-1} + 2g_{n-1} + 1 \end{aligned}$$

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

$$\begin{aligned} p_0 &= 0 \\ p_n &= p_{n-1} + 3h_{n-1} + 3g_{n-1} + 1 \end{aligned}$$

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 $k \le 50$) এবং n যদি অনেক বড় হয় (যেমন $n \le 10^9$) তাহলে এভাবেই আমাদের সমস্যাটি সমাধান করতে হবে।

2.7 গ্রাফ থিওরি এবং ম্যাট্রিক্স

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

প্রবলেম 2.3. 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

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

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

$$\begin{aligned}
&D_1 = A \\
&D_2 = D_1 \times A = A^2 \\
&D_3 = D_2 \times A = A^3 \\
&\vdots \\
&D_k = D_{k-1} \times A = A^k
\end{aligned}$$

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

2.8 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_{ik} এবং B_{kj} 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

$$C_{ij} = \max_{k=1}^{m}\{A_{ik} + B_{kj}\}$$

হবে। এটিও আগের মতই অ্যাসোসিয়েটিভ হবে।

প্রবলেম 2.4. 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

$$D_{k,i,j} = \min_{m=1}^{n}\{D_{k-1,i,m} + A_{m,j}\}$$

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

2.9 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 $A^1, A^2, A^4, \ldots, A^{2^k}$ 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 $A^1, A^2, A^4, \ldots, A^{2^k}$ বের করার পরে $\mathcal{O}(n^2 \log k)$ কমপ্লেক্সিটিতেই তুমি উত্তর বের করতে পারবে।

অনুশীলনী

1. 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 3

ন্যাপস্যাক

3.1 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 $f_{i-1,0}$, $f_{i-1,1}$, $f_{i-1,2}$, . . . , $f_{i-1,W}$ এর ওপর নির্ভর করে তাই $\mathcal{O}(W)$ মেমরি দিয়েও কাজটি করা সম্ভব। (মেমোরি অপটিমাইজেশনের চ্যাপ্টারটা দেখ)

3.2 $0 - K$ 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 সল্ভ 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 সল্ভ 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।

3.3 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 অপ-টিমাইজেশন আছে। তা হল `bitset` ব্যবহার করা। `bitset` ব্যবহার করলে টাইম কমপ্লেক্সিটি দাড়ায় $\mathcal{O}(\frac{nm}{64})$ এবং মেমোরি কমপ্লেক্সিটি দাড়ায় $\mathcal{O}(\frac{m}{64})$।

3.4 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 $\bmod P$ 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)।

3.5 $\mathcal{O}\left(s\sqrt{s}\right)$ 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 4

ব্যারিকেডস ট্রিক

4.1 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।

4.2 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:

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

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।

4.3 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 4.1. 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

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

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

$$= \mathcal{O}\left(\left(\sum_{i=1}^{m}|T_{u_i}|\right)^2\right)$$

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

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।

4.4 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:

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

4.5 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

নিজে করোঃ

অধ্যায় 5

এক্সচেঞ্জ আর্গুমেন্ট

5.1 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 পারবো অনেক সময়। 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 অন্তত এমন একটা অপ্টিমাল সলিউশন আছে, যেটা কিনা তোমার গ্রিডি সলিউশনের চাইতে খারাপ অথবা একই। অন্যভাবে বলতে গেলে, তোমার সলিউশনও একটি অপ্টিমাল সলিউশন। একটা উদাহরণ দেখা যাক।

উদাহরণ 5.1 (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 $A_i B_{i+1} + A_{i+1} B_i - A_i B_i - A_{i+1} B_{i+1}$। সুতরাং আমাদের প্রমাণ করতে

হবে-

$$\begin{aligned} A_iB_{i+1} + A_{i+1}B_i - A_iB_i - A_{i+1}B_{i+1} &\leq 0 \\ A_i(B_{i+1} - B_i) - A_{i+1}(B_{i+1} - B_i) &\leq 0 \\ A_i &\leq A_{i+1} \qquad \text{কারণ, } B_{i+1} - B_i > 0 \end{aligned}$$

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!

5.2 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-

০. ধরো 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।

০. 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।

০. **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।

অনুশীলনী 5.1. 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।

5.3 ডিপির সাথে সম্পর্ক

আমরা যখন কিছু উপাদানের উপর ডিপি করি তখন আমরা কোন কোন উপাদানগুলো বিবেচনা করে ফেলেছি এবং কোনগুলো বাকি আছে তার হিসাব রাখতে হয় এবং বেশিরভাগ ক্ষেত্রেই তা একটি প্রিফিক্স বা সাফিক্স হয়। অর্থাৎ আমাদের $\mathcal{O}(N)$ সাইজের একটা স্টেট রাখতে হয়। কিন্তু মনে করো আমাদের এরকম কিছু করতে বলল-

০. উপাদানগুলোর একটি অপ্টিমাল সাবসেট বাছাই করতে হবে।

০. এরপর চেক করে দেখতে হবে, ঐ সাবসেটটিকে কি এমন কোনো অর্ডারে সাজানো যায় কিনা যাতে সেই অর্ডারিং প্রবলেমে দেওয়া কিছু শর্ত পালন করে।

০. যদি করে, তাহলে সেই সাবসেটটিকে আমরা গ্রহণযোগ্য ধরব।

০. আবার একটি গ্রহণযোগ্য সাবসেটের উপাদান গুলো কিভাবে সাজানো আছে, তার উপর প্রবলেমে দেওয়া কস্ট ফাংশান ডিপেন্ড করে। সুতরাং, একটি সাবসেট বাছাই করে, তার মধ্যে আবার উপাদান গুলো এমন ভাবে সাজাতে হবে যেন কস্ট ফাংশান মিনিমাইজ হয়।

০. সব গ্রহণযোগ্য সাবসেটের মধ্যে মিনিমাম কস্ট বের করতে হবে।

তখন কি করা যায়? এমন প্রবলেম দেখলে মনে হতে পারে কোন গ্রিডি সলিউশন বের করতে পারি কিনা দেখি। হয়তো তুমি পেয়েও যেতে পারো! কিন্তু এরকম সমস্যায় এক্সচেঞ্জ আর্গুমেন্ট এর টেকনিকটিও অ্যাপ্লাই করে দেখা উচিত। এক্সচেঞ্জ আর্গুমেন্ট অ্যাপ্লাই করে আমরা উপাদানগুলোর একটি অর্ডার পেতে পারি যেখানে অন্তত একটি অপ্টিমাল আন্সারে উপাদানগুলো সেই অর্ডার অনুযায়ী সাজানো থাকবে। এতে যেই সুবিধা হয় তা হলো, এরপর আমরা প্রিফিক্সের/সাফিক্সের উপর ডিপি করতে পারবো।

উদাহরণ 5.2 (Code Festival '17 Final D - Zabuton)**.** একটি বালিশ প্রতিযোগিতায় $N \leq 5 \times 10^3$ জন প্রতিযোগী আছে। প্রত্যেক প্রতিযোগীর জন্য ২টি সংখ্যা- তার উচ্চতা $(0 \leq h_i \leq 10^9)$ এবং তার কাছে কয়টি বালিশ আছে $(1 \leq p_i \leq 10^9)$ তা তোমাকে দেওয়া আছে। প্রতিযোগীদের নির্দিষ্ট একটি ক্রমে সাজানোর পর তারা সেই ক্রমে একে একে আসে এবং স্তূপে বালিশের সংখ্যা দেখে (প্রথমে ০ থাকবে)। যদি স্তূপে তার নিজের **উচ্চতার** চেয়ে **বেশি** সংখ্যক বালিশ থাকে তাহলে সে মন খারাপ করে চলে যায়, নতুবা তার কাছে যতটি বালিশ আছে সেগুলো সে স্তূপে রেখে দেয়। তোমাকে বের করতে হবে কিভাবে প্রতিযোগীদের সাজালে সর্বোচ্চ সংখ্যক প্রতিযোগী বালিশ রাখতে পারবে (মন খারাপ করবে না)। তোমাকে শুধু সেই সর্বোচ্চ সংখ্যাটি আউটপুট দিতে হবে।

সমাধান. মনে করো এমন একটি সাজানোর উপায় আছে যাতে সবাই বালিশ রাখতে পারে (আসলে তো না-ই থাকতে পারে, কিন্তু আমরা প্রথমে সিম্পল জিনিস নিয়ে ঘাঁটাঘাঁটি করে দেখি না কি পাই)। ধরো, O হলো এমন একটি সাজানোর উপায়। আমরা এক্সচেঞ্জ আর্গুমেন্ট অ্যাপ্লাই করে বের করার চেষ্টা করবো এদের মধ্যে সম্পর্ক কেমন হতে পারে। O এর কিছু প্রপার্টি লিখে শুরু করা যাক। O তে পাশাপাশি আছে এমন ২টি প্রতিযোগী নাও আর ধরো P হলো i এর আগে আসা প্রতিযোগীদের বালিশের সংখ্যার যোগফল, অর্থাৎ, $P = \sum_{j=1}^{i-1} p_i$। এখন, O একটি ভ্যালিড অর্ডারিং হবে যদি এবং কেবল যদিঃ

$$P \leq h_i \text{ এবং} \tag{5.3.১}$$

$$P + p_i \leq h_{i+1} \tag{5.3.২}$$

হতে হবে। এখন নিচের দুটির মধ্যে যেকোনো একটি হতে পারেঃ

০. i **এবং** $i+1$ **এক্সচেঞ্জ করা যাবে না।** অর্থাৎ, i তম এবং $i+1$ তম প্রতিযোগীর অবস্থান যদি আমরা পরিবর্তন করে দেই তাহলে O একটি ভ্যালিড সিকুয়েন্স থাকবে না। অন্যভাবে বলতে গেলে-

$$h_{i+1} < P \text{ **অথবা**} \tag{5.3.৩}$$

$$h_i < P + p_{i+1} \tag{5.3.৪}$$

হতে হবে। খেয়াল করো, (5.3.২) সত্য হলে (5.3.৩) সত্য হতে পারে না। সুতরাং, (5.3.৪)-কে সত্য হতে হবে। (5.3.১), (5.3.২) এবং (5.3.৪) থেকে হিসাব করে পাই-

$$p_i + h_i < p_{i+1} + h_{i+1} \tag{5.3.৫}$$

-একটি কমপ্লিট অর্ডার! কিন্তু এর মানে কি আসলে? (5.3.৫) আমাদের বলছে, অপ্টিমাল সিকুয়েন্সের পাশাপাশি দুটি উপাদান যদি এক্সচেঞ্জ করা না যায় তাহলে তারা (5.3.৫) শর্ত পূরণ করে। কিন্তু আমাদের তো আরেকটি কেইস বাকি রয়ে গিয়েছে! তখন কি হবে?

০. i **এবং** $i+1$ **এক্সচেঞ্জ করা যাবে।** তাহলে,

$$P \leq h_{i+1} \text{ এবং} \tag{5.3.৬}$$

$$P + p_{i+1} \leq h_i \tag{5.3.৭}$$

হতে হবে। একটু খেয়াল করলে দেখবে (5.3.৭) $\implies$ (5.3.১) এবং (5.3.২) $\implies$ (5.3.৬)। তাই (5.3.১) আর (5.3.৬) আমাদের চিন্তা থেকে বাদ দিয়ে দিতে পারি। আরেকটা খুবই সুন্দর জিনিস হলো, (5.3.৫) এবং (5.3.৭) থেকে আমরা বলতে পারি (5.3.২) সত্য হবে। আবার আমরা আগেই দেখেছি (5.3.৭) সত্য হলে (5.3.১) সত্য হবে। অর্থাৎ, আমরা যদি এক্সচেঞ্জ করতে পারি, তাহলে (5.3.৫) অনুযায়ী সাজালেও O একটি ভ্যালিড সিকুয়েন্স থাকবে!

মোটকথা হলো, যদি অন্তত একটি ভ্যালিড অ্যারেঞ্জমেন্ট থাকে তাহলে প্রতিযোগীদের (5.3.৫) অনুযায়ী সর্ট করলে সেটিও একটি ভ্যালিড সিকুয়েন্স হবে! এখন তাহলে আমাদের কাজ হলো ইনপুটে দেওয়া প্রতিযোগীদের (5.3.৫) দিয়ে সর্ট করার পর (5.3.১) এবং (5.3.২) শর্ত পালন করে এমন ম্যাক্সিমাম লেংথের সাবসিকুয়েন্স বের করা। এই কাজটি আমরা একটি সাধারণ ডিপি দিয়েই করতে পারি।

$dp_{i,P}$ এর মান হলো- প্রথম i টি উপাদান বিবেচনা করলে ম্যাক্সিমাম ভ্যালিড সাবসিকুয়েন্সের লেংথ যাতে সাবসিকুয়েন্সের p_i গুলোর যোগফল P এর সমান হয়। এটার ট্রানজিশন অনেক সোজা। কিন্তু আসল কথা হলো, P এর মান তো অনেক বড় হতে পারে!

ডিপির স্টেট এবং ভ্যালু সোয়াপ করা। আমরা আগেই ডিপির স্টেট-ভ্যালু সোয়াপ করার কিছু উদাহরণ দেখে এসেছি। এখানেও আমাদের সেটি লাগবে। আমাদের নতুন ডিপি $dp_{i,j}$ এর মান হলো কোন একটি j সাইজের ভ্যালিড সাবসিকুয়েন্সের মিনিমাম $\sum p_i$ এর মান। এটার ট্রানজিশনও সোজা, পাঠকের অনুশীলনীর জন্য আর বলে দেওয়া হচ্ছে না।

উদাহরণ 5.3 (JOI Spring Camp '19 - Lamps)**.** তোমাকে দুটি N সাইজের বাইনারি অ্যারে A আর B দেওয়া আছে। তুমি প্রতি ধাপে নিচের যেকোনো একটি অপারেশন A অ্যারের উপর প্রয়োগ করতে পারবা-

০. **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।

০. **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।

০. **টগল 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. 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 অপারেশন আরেকটা অপারেশনের ওপর কিভাবে প্রভাব ফেলছে। দুটো অপারেশন o_i আর o_j নাও $(i < j)$। এখন দেখো, যদি $j > i + 1$ হয় তাহলে ঐ দুটি অপারেশনের মাঝে আরও অনেক অপারেশন এসে আমাদের ঝামেলায় ফেলে দিচ্ছে। তাই আমরা আপাতত $j = i + 1$ ধরি, অর্থাৎ o_i আর o_{i+1} নিয়ে চিন্তা করবো আমরা 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। আর রেঞ্জগুলো যদি ওভারল্যাপ করে তাহলে কিন্তু আমরা উভয় রেঞ্জ থেকে তাদের সাধারণ অংশ বাদ দিয়ে দিতে পারি।

- $\star_i = \oplus, \star_{i+1} = 1$। 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।

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

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

এখন আমরা ডিপি স্টেটে রাখতে পারি- i আর U। অর্থাৎ, যদি আমরা ধরে নেই $[i+1, N]$ এর মধ্যে U সেটের সব অপারেশন আগে থেকেই একটি করে ওপেন করা আছে, তাহলে $\text{dp}_{i,U}$ হলো $A[1 \ldots i]$ অ্যারেকে $B[1 \ldots i]$ অ্যারেতে রূপান্তর করতে মিনিমাম কয়টি অপারেশনের ইন্টারভাল ওপেন অথবা ক্লোজ করতে হবে (আমরা ওপেন ও ক্লোজ করার সময় আলাদা ভাবে +১ করবো এবং শেষে ডিপি ভ্যালুকে ২ দিয়ে ভাগ করলেই আমাদের আসল অ্যান্সার পেয়ে যাবো)। কোন কোন অপারেশনের ইন্টারভাল ওপেন আছে তা রাখার জন্য আমরা একটা বিটমাস্ক রাখবো। বিটমাস্কের i^{th} বিট অন থাকা মানে i^{th} অপারেশনের একটি ইন্টারভাল ওপেন আছে (যেখানে $i \in [0, 2]$ এবং প্রথম অপারেশন সেট, দ্বিতীয় অপারেশন রিসেট, তৃতীয় অপারেশন টগল)। আমাদের সুবিধার জন্য আমরা একটা ফাংশন $f(b, S)$ ডিফাইন করতে পারি যেটা একটা বিট b আর একটা অপারেশনের সেট S ইনপুট নিবে এবং রিটার্ন করবে b বিটটির ওপর S এর অপারেশন গুলো পর্যায়ক্রমে অ্যাপ্লাই করলে শেষে b এর মান কত হবে। $\text{dp}_{i,U}$ ক্যাল্কুলেট করার সময় আমরা ঠিক করবো i এর উপর দিয়ে কোন কোন অপারেশনের ইন্টারভাল যাবে (ধরে নিলাম সেই অপারেশনের সেটটি হলো V)। V সেটটি ফিক্স করার পর (এমন 2^3টি সেট আছে) আমরা A_i এর ওপর V এর অপারেশনগুলো অ্যাপ্লাই করে যদি দেখি তা B_i এর সমান হয়েছে, তাহলে সেটি একটি ভ্যালিড ট্রানজিশন হবে। সেই ট্রানজিশনের কস্ট হবে $|U \oplus V|$[২] কারণ যেসব অপারেশন U তে আছে কিন্তু V তে নেই সেগুলো $i+1$তম ইনডেক্সে ক্লোজ করছি আর যেসব অপারেশন V তে আছে কিন্তু U তে নেই সেগুলো iতম ইনডেক্সে ওপেন করছি। সুতরাং, $i-1$ সাইজের প্রিফিক্সের জন্য ওপেন অপারেশনের সেটটি হবে V। তাহলে $i \geq 1$ এর জন্য আমাদের ডিপি রিকারেন্স হবে অনেকটা এরকমঃ

$$\text{dp}_{i,U} = \min_{V \subseteq \{0,1,\oplus\},\, f(A_i,V)=B_i} \left\{\text{dp}_{i-1,V} + |U \oplus V|\right\}$$

আর বেস কেইস $i = 0$ এর জন্য হবে- $\text{dp}_{0,U} = |U|$। ফাইনাল আন্সার হবে- $\text{dp}_{N,\varnothing}$।

[২] $\oplus$ অপারেটরটি হলো দুটি সেট এর Symmetric Difference, অর্থাৎ, এমন আরেকটি সেট যেখানে শুধু U অথবা V তে আছে কিন্তু তাদের ইন্টারসেকশনে নেই এমন উপাদানগুলো আছে। বিটমাস্কের ভাষায় বললে Exclusive Or বা XOR বলতে পারো। আর $|S|$ এর মানে হলো S সেট এর সাইজ।

ডিপি স্টেট আছে NK টা এবং একটা স্টেট থেকে ট্রানজিশন করা যায় K ভাবে, যেখানে K হলো U অথবা V এর জন্য ভ্যালিড সেটের সংখ্যা। সুতরাং, ওভারঅল কমপ্লেক্সিটি হবে $\mathcal{O}(NK^2)$। যেহেতু $U, V \subseteq \{0, 1, \oplus\}$, তাই $K = 2^3$। কিন্তু একটু চিন্তা করলেই দেখা যাবে সেট আর রিসেট অপারেশন একসাথে থাকার কোন মানেই হয় না। এমন সাবসেটগুলো বাদ দিলে $K = 6$ হয়।

উদাহরণ 5.4 (Codeforces Gym 100971I - Deadline)**.** তোমাকে একটি ডিরেক্টেড গ্রাফ দেওয়া হয়েছে ($N, M \leq 2 \times 10^5$)। প্রতিটি নোড দিয়ে একটি কাজ আর ডিরেক্টেড এজ দিয়ে কোন কাজের আগে কোন কাজগুলো শেষ করতে হবে তা বুঝানো হয়েছে ($u \to v$ এজ থাকা মানে হলো u এর আগে v কাজটি শেষ করতে হবে)। প্রতিটি কাজের জন্য দুটি ভ্যালু দিয়ে দিবে তোমাকে- কাজটি করতে কতক্ষণ লাগবে (c_i) আর কাজটি কত সময়ের ভিতরে শেষ করতে হবে (d_i)। সব কাজ গুলো শেষ করার একটা উপায় বের করতে হবে তোমাকে, অথবা বলবে হবে কোনভাবেই সবগুলো কাজ শেষ করা যাবে না।

সমাধান. ডিপেন্ডেন্সি বিবেচনায় না এনে শুধু c এবং d অ্যারেগুলোর ওপর এক্সচেঞ্জ আর্গুমেন্ট অ্যাপ্লাই করে আমরা পাই, কাজগুলো d_i দিয়ে সর্ট করা থাকতে হবে। তাহলে, আমাদের স্ট্র্যাটেজিটা হবে অনেকটা এরকম- প্রতি ধাপে আমরা যেই নোডগুলো প্রসেস করা হয়নি তাদের মধ্যে সবচেয়ে ছোট d_i এর নোডটা নিবো (ধরো, u) এবং এই নোডটাকে আমরা যত শুরুর দিকে সম্ভব বসানোর চেষ্টা করবো। কত শুরুতে বসাতে পারি আমরা একে? আমাদেরকে অবশ্যই u এর ডিপেন্ডেন্সিগুলো সল্ভ করতেই হবে। তাই, যেসব নোড এখনো প্রসেস করা হয়নি তাদের মধ্যে যেসব নোডে u থেকে পৌঁছানো যায় তাদের সেট S (u নিজেও থাকবে কিন্তু) নিবো আমরা। ধরি, H হলো S দ্বারা ইন্ডিউসড সাবগ্রাফ (Induced Subgraph[৩])। এবার H এর এজ গুলো রিভার্স করে দাও এবং তারপর BFS এর মাধ্যমে H এর টপোলজিক্যাল সর্ট বের করতে হবে। কিন্তু BFS-এ FIFO (First In First Out) কিউ ব্যবহার না করে আমাদের মিনিমাম প্রায়োরিটি কিউ ব্যবহার করতে হবে। এই টপোলজিক্যাল অর্ডার আমাদের ফাইনাল সিকুয়েন্সের শেষে অ্যাড করে দিবো। ধাপগুলো চলাকালীন কোন সাইকেল পাওয়া গেলে বা আমরা শেষে যে সিকুয়েন্স পাবো তা অনুযায়ী যদি কাজগুলো করে কোনটির ডেডলাইন পার হয়ে যায় তাহলে কাজগুলো শেষ করার কোন উপায় নেই।

এটি যদিও একটি ডিপি সমস্যা না, তবুও এই আইডিয়াটা ডিপিতে কাজে লেগে যেতে পারে। কারণ অনেক সময় আমাদের জানা থাকে না কোন অর্ডারে ডিপি ভ্যালুগুলো ক্যালকুলেট করতে হবে, যেমন আমরা হয়ত ডিরেক্টেড অ্যাসাইক্লিক গ্রাফ থেকে নোডগুলোর একটি Partial Order পেতে পারি কিন্তু যেই পেয়ারগুলোর মধ্যে কোন অর্ডার নেই সেগুলো কোন অর্ডারে প্রসেস করতে হতে পারে সেটাও প্রবলেমের একটা অংশ হতে পারে। সেজন্য আমাদের কাছে এই উদাহরণটি এখানে দেখানো ভালো আইডিয়া মনে হয়েছে।

উদাহরণ 5.5 (Pieces of Parentheses)**.** N টি ব্র্যাকেট সিকুয়েন্স দেওয়া আছে (Balanced[৪] নাও হতে পারে)। তুমি প্রথমে সেগুলো একটি ক্রমে সাজাবা, তারপর সেই ক্রমে তাদের জোড়া লাগেতে হবে এবং জোড়া লাগানো সেই স্ট্রিং থেকে কিছু ক্যারেক্টার বাদ দিতে হবে। কাজ গুলো তুমি এমনভাবে করবা যেন তোমাকে মিনিমাম সংখ্যক ক্যারেক্টার বাদ দিতে হয় কিন্তু ফাইনাল স্ট্রিংটা একটা ভ্যালিড ব্র্যাকেট সিকুয়েন্স থাকে।

[৩]একটি ভার্টেক্স সেট S এর Induced Subgraph হলো এমন একটি গ্রাফ, যেখানে S এর সব নোড থাকবে এবং মূল গ্রাফের যেসব এজ শুধু S থেকে S এই গিয়েছে শুধু সেগুলো থাকবে।

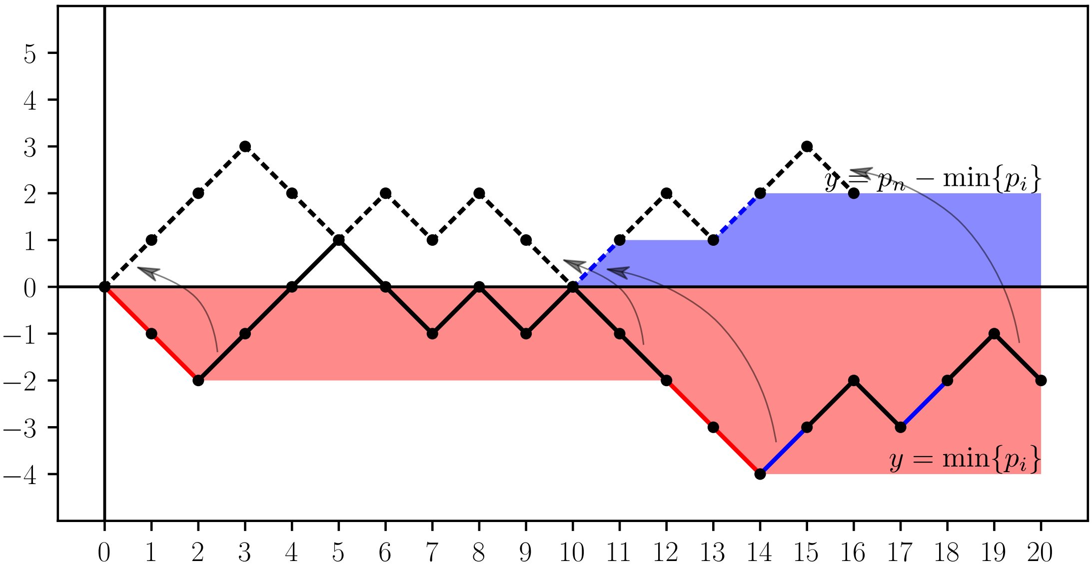
[৪]Balanced Bracket Sequence হলো যেই ব্র্যাকেট সিকুয়েন্স যেটার মাঝখানে মাঝখানে কিছু সংখ্যা আর গাণিতিক অপারেটর বসালে একটি শুদ্ধ গাণিতিক রাশি পাওয়া যায়

সমাধান. আগে একটি স্ট্রিং এর জন্য অ্যান্সার কি হবে চিন্তা করি। আমরা সাধারণ ব্র্যাকেট ম্যাচিং অ্যালগরিদমকে এভাবে মডিফাই করতে পারি-

১ম ধাপ বাম থেকে ডানে স্ট্রিংটা ইটারেট করতে থাকবো, যদি Opening ব্র্যাকেট পাই, তাহলে সেই ক্যারেক্টারটা স্ট্যাকে পুশ করে দিবো। আর, Closing ব্র্যাকেট পেলে দেখবো স্ট্যাকে কোন ক্যারেক্টার আছে কিনা, যদি থাকে তাহলে স্ট্যাক থেকে সেই টপ ক্যারেক্টারটা পপ করে নিবো। কিন্তু যদি স্ট্যাক খালি থাকা অবস্থায় আমরা একটি Closing ব্র্যাকেট পাই তাহলে সেই ক্যারেক্টারটি ডিলিট করতেই হবে, নাহলে আমরা স্ট্রিংটাকে কখনই ব্যালেন্সড ব্র্যাকেট সিকুয়েন্স বানাতে পারবো না।

২য় ধাপ প্রথম ধাপটি শেষ হলে স্ট্যাকে যেই ক্যারেক্টারগুলো বাকি থাকবে তাদের বাদ দিবো।

এই দুটি ধাপে কোন কোন ক্যারেক্টার ডিলিট করছি আমরা, তা যদি ব্র্যাকেট সিকুয়েন্সের প্রিফিক্স সাম গ্রাফে[৫] দেখো তাহলে বুঝতে পারবে আমাদের বরাবর $p_n - 2\min\{p_i\}$ টি ক্যারেক্টার ডিলিট করতে হচ্ছে। খেয়াল করো, কিছু স্ট্রিং রিঅ্যারেঞ্জ করে জোড়া লাগানোর পর সেই বড় স্ট্রিং এর p_n আর $\min\{p_i\}$ এর



চিত্র 5.1:))((())()())))(()(() এর জন্য গ্রাফ। প্রথম ধাপে লাল ক্যারেক্টারগুলো ডিলিট হবে। এর পর আমরা লাল অংশগুলো বাদ দিয়ে বাকি অংশগুলোতে একটার পর আরেকটা জোড়া দিলে ডোরাকাটা গ্রাফটি পাবো। সেই গ্রাফের নীল ক্যারেক্টারগুলো ডিলিট করা হবে দ্বিতীয় ধাপে। মোট লাল আর নীল ছায়া করা অংশ দুটি ডিলিট হবে, যা হলো $p_n - 2\min\{p_i\}$।

মান আমরা কিন্তু শুধু ছোট স্ট্রিং গুলোর p_n এবং $\min\{p_i\}$ এর মান দেখেই বলে দিতে পারি। আবার ছোট স্ট্রিং গুলো যেভাবেই সাজাও না কেনো বড় স্ট্রিং এর p_n এর মান কিন্তু একই থাকবে (সবগুলো p_n এর যোগফল)। তাহলে আমাদের উত্তর শুধু বড় স্ট্রিং এর $\min\{p_i\}$ এর মানের উপর নির্ভর করছে আর আমাদের উদ্দেশ্য হলো এর মান যতটা সম্ভব বড় করা।

[৫]ধরো, x_i এর মান -1 হবে যদি ব্র্যাকেট সিকুয়েন্সের i-তম ক্যারেক্টারটি Closing ব্র্যাকেট হয়, আর নাহলে 1। এখন $p_0 = 0$ এবং $p_i = p_{i-1} + x_i$ এর জন্য যদি আমরা (i, p_i) পয়েন্ট গুলো গ্রাফে প্লট করি তাহলে তাকে সেই ব্র্যাকেট সিকুয়েন্সের প্রিফিক্স সাম গ্রাফ বলছি আমরা।

আগের মতই আমরা যেকোনো একটি অপ্টিমাল সলিউশন O নিয়ে ঘাঁটাঘাঁটি করবো। O এর i-তম ছোট স্ট্রিংটার p_n কে আমরা s_i আর $\min\{p_i\}$[৬] কে m_i দিয়ে সূচিত করবো। ধরো সব অপ্টিমাল সলিউশনের $\min\{p_i\}$ এর মান M আর যেসব সলিউশনের $\min\{p_i\} \geq M$ তাদের আমরা ভ্যালিড সলিউশন বলবো। তাহলে O যদি একটি অপ্টিমাল সলিউশন হয় তাহলে সব i এর জন্য নিচের শর্তটি পূরণ হবেঃ

$$S + m_i \geq M \text{ এবং} \tag{5.3.৮}$$

$$S + s_i + m_{i+1} \geq M \tag{5.3.৯}$$

, যেখানে $S = \sum_{j=1}^{i-1} s_j$। এখন, আমরা যদি i আর $i+1$ সোয়াপ করতে না পারি তাহলে-

$$S + m_{i+1} < M \text{ অথবা} \tag{5.3.১০}$$

$$S + s_{i+1} + m_i < M \tag{5.3.১১}$$

আগের প্রবলেমের মতো এখানে (5.3.৯) থেকে কিন্তু আমরা বলতে পারি না (5.3.১০) মিথ্যা, কারণ s_i ধনাত্মক, ঋণাত্মক বা শুন্য যেকোনোটিই হতে পারে। তাহলে কি করা যায়? দুটি কেইস আলাদাভাবে চিন্তা করতে পারি আমরা- s_i ধনাত্মক হলে অবশ্যই সেটাকে একটি ঋণাত্মক বা শুন্য s_{i+1} এর আগে রাখা উচিত, কারণ আগে রাখলে আমাদের কোনো লস হচ্ছে না বরং পরের কোনো স্ট্রিং-এ এই লাভটা কাজে লেগে যেতে পারে। এটাকে আরেক্টু গুছিয়ে বললে হয়- যদি একটি অপ্টিমাল সলিউশনের $s_i \leq 0$ এবং $s_{i+1} > 0$ হয় তাহলে আমরা এই দুটি উপাদান এক্সচেঞ্জ করলে যে সলিউশন পাবো সেটিও একটি ভ্যালিড সলিউশন হবে[৭]। এবার প্রশ্ন হলো s_i আর s_{i+1} দুটোই ধনাত্মক হলে কি হবে? চিন্তা করে দেখো, যদি অপ্টিমাল সলিউশনে $m_i < m_{i+1}$ হয় তাহলে এখানেও এদের সোয়াপ করলে সলিউশনটি ভ্যালিড থাকবে। কিন্তু আমরা এখানে কোনো কারণ ছাড়া এক্সচেঞ্জ করছি কেন মনে হতে পারে। আগের এক্সচেঞ্জ এর উদ্দেশ্য ছিল সলিউশন ভ্যালিড রেখে আমাদের কিছু প্রফিট আদায় করা, যেগুলো আমরা পরে খরচ করতে পারবো। এখানেও একই। আমরা যদি $m_i > m_{i+1}$ অর্ডারে রাখি তাহলে পরে বৃহত্তর লস (m ভ্যালু গুলোকে লস মনে করো) এর জন্য আমরা আগে থেকে বাঁচিয়ে রাখা কিছু প্রফিট (s_i) ব্যবহার করতে পারবো।

এবার আসা যাক s_i আর s_{i+1} উভয়ই ঋণাত্মক বা শুন্য হলে কি হবে। খেয়াল করো, এখন কিন্তু আমরা (5.3.৯) থেকে বলতে পারি (5.3.১০) মিথ্যা! এখন 5.2 প্রবলেম এর মতো করে আমরা এরকম একটা অসমতা পাবোঃ

$$s_i - m_i > s_{i+1} - m_{i+1} \tag{5.3.১২}$$

তাহলে শেষ পর্যন্ত এই দাঁড়ালো- প্রথমে সব ধনাত্মক s_i কে আগে রাখতে হবে আর বাকি গুলো পরে। তারপর ধনাত্মক s_i গুলোর মধ্যে আমরা $i < j$ এর জন্য $m_i > m_j$ অর্ডারে সাজাবো আর ধনাত্মক বা শুন্য s_i গুলোর ক্ষেত্রে আমরা $i < j$ এর জন্য $s_i - m_i > s_j - m_j$ অনুযায়ী সাজাবো। এই Comparator টা কিন্তু একটা Complete Order[৮]!

এখন কিন্তু আরেকটি কাজ বাকি রয়ে গিয়েছে! আমরা ধরে নিয়েছিলাম আমরা সোয়াপ করতে পারবো না অর্থাৎ সোয়াপ করলে সলিউশনটা ইনভ্যালিড হয়ে যাবে। কিন্তু সোয়াপ করলে পারলে (5.3.১২) অনুযায়ী সাজালেও যে সেটি একটি ভ্যালিড সলিউশন থাকবে তা প্রমাণ করতে হবে। এটা 5.2 প্রবলেমটির মতো করে প্রভ করার চেষ্টা করো।

---

[৬]এখানে i-তম স্ট্রিং এর জন্য খালি p_i বিবেচনা করছি- এরকম না। আসলে সব জায়গায় $\min\{p_i\}$ দিয়ে ঐ স্ট্রিং এর প্রিফিক্স সাম গুলোর মিনিমাম ভ্যালু বুঝানো হচ্ছে।

[৭]প্রমাণ করে দেখো।

[৮]ভেরিফাই করে দেখো।

5.4 অনুশীলনী

অনুশীলনী 5.2 (Codeforces 1354F - Summoning Minions)**.** তোমার কাছে $n\,(1 \leq n \leq 75)$-টা মিনিয়ন আছে এবং তুমি তাদের হাজির করতে পারো। i-তম মিনিয়নের প্রথমিক পাওয়ার লেভেল হলো $a_i\,(1 \leq a_i \leq 10^5)$, এবং যখন তুমি এই মিনিয়নটিনে হাজির করবে, তখন আগের মিনিয়ন সবগুলোর পাওয়ার $b_i\,(0 \leq b_i \leq 10^5)$ করে বেড়ে যাবে। মিনিয়নগুলোকে তুমি যেকোনো ক্রমে হাজির করতে পারবা। কিন্তু একটা শর্ত আছে, তুমি যেকোনো সময় $k\,(1 \leq k \leq n)$ টার বেশি মিনিয়ন হাজির করে রাখতে পারবে না। তুমি যেকোনো সময় হাজির করা মিনিয়নকে ধ্বংস করে দিতে পারবা – অন্যভাবে বলতে গেলে, প্রতিটা মিনিয়নকে তুমি সর্বোচ্চ একবার হাজির (বা ধ্বংস) করতে পারবা। তোমার লক্ষ্য হলো সবচাইতে শক্তিশালী মিনিয়নের আর্মি হাজির করা, অর্থাৎ, শেষ পর্যন্ত থাকা (যেগুলো হাজির করেছ, কিন্তু ধ্বংস করোনি) মিনিয়নগুলোর পাওয়ার লেভেলের যোগফল ম্যাক্সিমাইজ করা। প্রতিটা ইনপুট ফাইলে $T \leq 75$ টা টেস্ট কেইস থাকতে পারে।

অনুশীলনী 5.3 (Codeforces 1107F - Vasya and Endless Credits)**.** তুমি একটা গাড়ি কিন্তে চাও, কিন্তু তোমার কাছে বর্তমাকে ০ টাকা আছে। সেজন্য ব্যাংক তোমাকে n-টা অফার দিয়েছে। i-তম অফারটি তুমি যেই মাসে নিবে, সেই মাসের শুরুতে তোমাকে ব্যাংক a_i টাকা দিবে। আবার, যেই মাস থেকে শুরু করেছ, সেই মাস থেকে ঠিক টানা k_i মাস ধরে প্রতি মাসের শেষ দিনে তোমার ব্যাংককে b_i টাকা দিতে হবে (যেই মাসে কিনেছ, সেই মাসের শেষেও b_i দিতে হবে)। অফারগুলো তুমি যেকোনো অর্ডারে, যেকোনো মাসে কিনতে পারো, কিন্তু কোনো এক মাসে তুমি একটার বেশি অর্ডার কিন্তে পারবা না। তবে, একসাথে একাধিক অফার চালু থাকতে পারবে, যার মানে হলো প্রতি মাসের শেষে, যেই যেই অফারগুলো চালু আছে, সেগুলোর b_i এর যোগফলের সমান টাকা মাসের শেষে ব্যাংককে দিতে হবে। এখন, তুমি কোন এক মাসের মাঝখানে গাড়িটা কিন্তে চাও, আর সেই সময় তোমার কাছে যত টাকা আছে সব নিয়ে গাড়ি কিনে পালিয়ে যাবা – এটা তোমার প্ল্যান (তারপর আর ব্যাংককে পাওনা টাকা ফেরত দিতে হবে না)। তোমাকে বের করতে হবে সর্বোচ্চ কত দামের গাড়ি তুমি কিনতে পারবা। $1 \leq n \leq 500$, $1 \leq a_i, b_i, k_i \leq 10^9$।

অধ্যায় 6

পলিনমিয়াল ইন্টারপোলেশন

6.1 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 n ডিগ্রি পলিনমিয়াল, এবং দেওয়া আছে

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

অর্থাৎ $n+1$ টা $P(x) = y$ আকারের শর্ত দেওয়া আছে। 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 বলা 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

- একটি n ডিগ্রি পলিনমিয়ালের সাথে আরেকটা m ডিগ্রি পলিনমিয়াল যোগ করলে যোগফলের ডিগ্রি হবে $\max(n,m)$।
- একটি n ডিগ্রি পলিনমিয়ালের সাথে আরেকটা m ডিগ্রি পলিনমিয়াল বিয়োগ করলে বিয়োগফলের সর্বোচ্চ ডিগ্রি হবে $\max(n,m)$। তবে এর চেয়ে কমও হতে পারে।
- একটি n ডিগ্রি পলিনমিয়ালের সাথে আরেকটা m ডিগ্রি পলিনমিয়াল গুন করলে গুনফলের ডিগ্রি হবে $n+m$।

6.2 কীভাবে পলিনমিয়াল ইন্টারপোলেশন কাজ করে

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

উদাহরণ 6.1. এমন দ্বিঘাতী পলিনমিয়াল বের কর যেন

$$\begin{aligned} P(1) &= -3 \\ P(4) &= 0 \\ P(5) &= 0 \end{aligned}$$

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

$$\begin{aligned} P(4) &= 0 \\ P(5) &= 0 \end{aligned}$$

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

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

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 6.2. 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।

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

$P_1,\ P_2,\ P_3$ 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।

6.3 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।

6.4 ডাইনামিক প্রোগ্রামিং-এর সাথে সম্পর্ক

আপাতদৃষ্টিতে পলিনমিয়াল ইন্টারপোলেশনের সাথে ডাইনামিক প্রোগ্রামিং এর তেমন কোন সম্পর্ক বুঝা যাচ্ছে না। সত্য কথা বলতে কিছু উদাহরণ না দেখালে এ সম্পর্ক পুরোপুরি বুঝতে পারবে না। তবে মুল আইডিয়াটা হল এমন:

ধর তোমার ডিপির কোন এক স্টেট বিশাল বড় হয়ে গেছে (10^9 ধরতে পার)। এমন কিছু প্রব্লেমে ডিপিটাকে ওই স্টেটটির একটি পলিনমিয়াল হিসেবে চিন্তা করা যায়। অর্থাৎ ডিপি থেকে তুমি সেই স্টেটটি পুরোপুরি সরিয়ে ফেলতে পার। উদাহরণ হিসেবে ধর আমাদের একটি ডাইনামিক প্রোগ্রামিং এর জন্য $f_{i,j}$

বের করতে হবে। এখানে j এর মান বিশাল বড় হতে পারে। তুমি কোনোভাবেই $f_{i,j}$ এর সব j এর জন্য মান বের করতে পারবে না। তাহলে কী করা যায়? এক্ষেত্রে সমাধান হল $f_{i,j}$ কে j এর একটি পলিনমিয়াল ধরতে পার। অর্থাৎ

$$f_i(j) = \sum_{k=0}^{n} a_k j^k$$

যদি এমন একটা পলিনমিয়াল সত্যিই থেকে থাকে, তাহলে কিন্তু আমাদের সব j এর জন্য $f_{i,j}$ এর মান বের করতে হচ্ছে না। শুধু $a_0, a_1, a_2, \ldots, a_n$ এর মান গুলো জানা থাকলেই আমরা যেকোনো j এর জন্য সহজেই $f_{i,j}$ এর মান বের করতে পারব।

এখন কথা হচ্ছে সব রিকারেন্সের জন্যই এমন একটি পলিনমিয়াল পাওয়া সম্ভব? অবশ্যই না। অনেক ক্ষেত্রেই এমন পাওয়া সম্ভব, আবার অনেক সময় পাওয়া সম্ভব না। কখন এটা খাটবে সেটা তোমাকেই প্রমাণ করে নিতে হবে। আসল কন্টেস্টের সময় অনেক ক্ষেত্রে অনুমান করাও যথেষ্ট (অভিজ্ঞ প্রোগ্রামাররা কিছু ক্ষেত্রে তাই করে)। তবে এটা বুঝার একটি উপায় হল যদি তোমার একটি স্টেট বেশ বড় হয় এবং রিকারেন্সের মধ্যে সব বীজগাণিতিক অপারেটর ব্যবহার করা হয় (যেমন যোগ, বিয়োগ, গুন; $\max$, $\min$, xor এসব কিন্তু বীজগাণিতিক অপারেটর নয়) তাহলে অনেক ক্ষেত্রেই পলিনমিয়াল ইন্টারপোলেশন খাটে।

6.4.1 কিছু উদাহরণ

এবার কিছু উদাহরণ দেখা যাক।

প্রবলেম 6.1. (Luogu P4463) তোমার কাছে দুটি সংখ্যা n এবং k দেওয়া আছে ($1 \le n \le 500$, $1 \le k \le 10^9$)। কোন একটা n দৈর্ঘ্যের সিকুয়েন্স a কে **ভালো** বলা হবে যদি a এর সংখ্যাগুলো 1 থেকে k এর মধ্যে থাকে এবং সবগুলো সংখ্যা ভিন্ন ভিন্ন হয়। a এর সংখ্যাগুলোর গুণফলকে বলা হয় a সিকুয়েন্সটির ভ্যালু। তোমাকে যতগুলো সম্ভাব্য **ভালো** সিকুয়েন্স আছে সবগুলোর ভ্যালুর যোগফল বলতে হবে।

সমাধান. k এর বিশাল লিমিট দেখে ভয় পেয়ে যেয়ো না। প্রথমে আমরা ডিপির স্টেট আর রিকারেন্সটা বের করি। বোঝাই যাচ্ছে স্টেটে আমাদের n এবং k দুটোই রাখা লাগবে। প্রব্লেমের সুবিধার্থে ধর আমাদের ভালো সিকুয়েন্সটার সংখ্যাগুলো ছোট থেকে বড় ক্রমানুসারে সাজানো থাকবে। এটা ধরে সমাধান করার পর $n!$ দিয়ে গুন করলেই উত্তর পেয়ে যাব। এখান থেকে আমরা রিকারেন্সটি লেখতে পারি এভাবে

$$f_{n,k} = f_{n,k-1} + k \times f_{n-1,k-1}$$

যদি সিকুয়েন্সটির শেষ সংখ্যাটি k এর চেয়ে ছোট হয় তাহলে প্রতিটি সংখ্যা 1 থেকে $k-1$ এর মধ্যে থাকবে। এই n টি দৈর্ঘ্যের ভালো সিকুয়েন্সগুলোর ভ্যালুর যোগফল হবে $f_{n,k-1}$। আবার যদি শেষ সংখ্যাটি ঠিক k এর সমান হয় তাহলে বাকি $n-1$ টি সংখ্যা 1 থেকে $k-1$ এর মধ্যে থাকবে। এই $n-1$ দৈর্ঘ্যের সিকুয়েন্সগুলর ভ্যালুর যোগফল হবে $f_{n-1,k-1}$। তবে এর সাথে k গুন দিতে হবে, কারণ n তম সংখ্যাকে আমরা k ধরেছি। তাই $n-1$ দৈর্ঘ্যের সিকুয়েন্সগুলোর ভ্যালুগুলো k দিয়ে গুন হবে। এ পর্যন্ত আমরা যা যা বের করলাম তা বেশ সহজ-ই। আগের চ্যাপ্টারেগুলোতে আমরা এর চেয়েও কঠিন ডিপি বের করেছিলাম। কিন্তু আমাদের সমস্যা এখনো মোটেই সমাধান হয়নি। এই ডিপি ক্যাল্কুলেট করতে আমাদের $\mathcal{O}(nk)$ কমপ্লেক্সিটি প্রয়োজন, যেটা আমাদের সাধ্যের বাইরে।

এর সমাধান হল মনে মনে চিন্তা কর $f_{i,j}$ আসলে j এর একটি পলিনমিয়াল। যেহেতু j এর পলিনমিয়াল তাই $f_{i,j}$ এর পরিবর্তে আমরা $f_i(j)$ লেখব। কিন্তু কত ডিগ্রি পলিনমিয়াল সেটা বুঝব কি করে? আগের রিকারেন্সটাতে ফেরত যাই। রিকারেন্সটা একটু গুছিয়ে এভাবে লেখা যায়

$$f_i(j) - f_i(j-1) = j \times f_{i-1}(j-1)$$

$f_i(j)$ এর পলিনমিয়ালের ডিগ্রি $g(i)$ হলে বামপক্ষের ডিগ্রি হবে $g(i)-1$, কারণ যেকোনো পলিনমিয়াল P এর জন্য $P(x) - P(x-1)$ এর ডিগ্রি হয় $\deg P - 1$ (এটা নিজে প্রমাণ করার চেষ্টা কর)। আবার ডান পক্ষের ডিগ্রি হবে $g(i-1)+1$। দুটি সমান হতে হলে $g(i)-1 = g(i-1)+1$ হতে হবে। এটি সমাধান করলে দেখবে $g(i) = 2i$। অর্থাৎ $f_n(x)$ পলিমনিয়ালের ডিগ্রি $2n$।

আমাদের কাজ অনেক সহজ হয়ে গেল এখন। আগে আমাদের $f_n(1), f_n(2), \ldots, f_n(k)$ সবগুলো মান বের করতে হচ্ছিল। কিন্তু এখন আমাদের জন্য শুধু $f_n(1), f_n(2), \ldots, f_n(2n+1)$ এর মানগুলো বের করাই যথেষ্ট। এরপর এই মান গুলো দিয়ে পলিমিয়াল ইন্টারপোলেশন করলেই আমরা f_n এর পলিনমিয়াল পেয়ে যাব। লক্ষ্য কর, পলিনমিয়ালের ডিগ্রি $2n$ হওয়াতে আমাদের $2n+1$ টা পয়েন্টে ডিপির মান বের করতে হয়েছে।

আমাদের সমাধানের মধ্যে কিন্তু একটা ঘাপলা থেকে গিয়েছে। আমরা শুরুতেই ধরে নিয়েছিলাম $f_{i,j}$ আসলে j এর একটি পলিনমিয়াল হবে। কিন্তু আসলেই যে পলিনমিয়াল হবে সেটা প্রমাণ করা হয় নি। সত্য কথা বলতে গেলে প্রমাণের অনেকখানি কাজ আমরা ইতোমধ্যে করে ফেলেছি। ডিগ্রির শর্তগুলো যখন বের করছিলাম তখন এর সাথে গাণিতিক আরোহ জুড়ে দিলেই প্রমাণ হয়ে যেত। এ কাজটি তোমাদের জন্য রেখে দিলাম।

প্রবলেম 6.2. (Codeforces Round 492 Div1 F) n টি নোডের একটি রুটেড ট্রি (rooted tree) দেওয়া থাকবে, যেখানে ১ নম্বর নোডটি হল রুট। ট্রি এর প্রত্যেক নোডে 1 থেকে D এর মধ্যে একটি সংখ্যা বসাতে হবে যেন রুট ব্যতীত যেকোনো নোডে বসানো সংখ্যা তার প্যারেন্টের সংখ্যার চেয়ে ছোট হয়। কতভাবে সংখ্যাগুলো বসানো যাবে। $(1 \leq n \leq 3000,\ 1 \leq D \leq 10^9)$

সমাধান. এটা অনেকটা আগের সমস্যাটার মতই। এর ডিপিটাও আগের সমস্যার ডিপির মত অনেকটা, তাই পড়া থামিয়ে নিজে বের করার চেষ্টা কর আগে।

আমরা ডিপিটাকে সংজ্ঞায়িত করব এভাবে: $f_u(j) =$ নোড u এর সাবট্রিতে 1 থেকে j এর মধ্যে সংখ্যাগুলো কতভাবে বসানো যায় যেন প্রত্যেক কোন চাইল্ডে প্যারেন্টের চেয়ে বড় সংখ্যা না থাকে। তাহলে রিকারেন্স হবে

$$f_u(j) = f_u(j-1) + \prod_{v \in \text{child(u)}} f_v(j-1)$$

এর ব্যাখ্যাও প্রায় আগের সমস্যার মতই। u নোডে যদি j না বসাই তাহলে সাবট্রির প্রত্যেক নোডে 1 থেকে $j-1$ এর মধ্যে কোন একটি সংখ্যা বসাতে হবে, যেটি করা যায় $f_u(j-1)$ উপায়ে। আর যদি u নোডে j বসাই তাহলে u এর চাইল্ডগুলোতে 1 থেকে $j-1$ এর মধ্যে সংখ্যাগুলো বসাতে হবে, যেটি করা যায় $\prod_{v \in \text{child(u)}} f_v(j-1)$ উপায়ে।

এবার আগের মতই আবার ধরব $f_u(j)$ একটি পলিনমিয়াল যার ডিগ্রি $g(u)$। রিকারেন্সটি একটু সাজিয়ে লেখলে পাই

$$f_u(j) - f_u(j-1) = \prod_{v \in \text{child(u)}} f_v(j-1)$$

এর দুইপাশে ডিগ্রি সমতা করলে পাব

$$g(u) - 1 = \sum_{v \in \text{child(u)}} g(v)$$

এই রিকারেন্সটিকে চিনতে পেরেছ? সাবট্রি সাইজ বের করার জন্য আমরা ঠিক এরকম একটি রিকারেন্স ব্যবহার করি। এখান থেকে বোঝা যায় যে $g(u)$ এর মান আসলে u এর সাবট্রি তে যতগুলো নোড আছে তার সমান হবে। অর্থাৎ রুট 1 এর জন্য পলিনমিয়ালের ডিগ্রি হবে ঠিক ঠিক n। সুতরাং আমাদের $f_1(1), f_1(2), \ldots, f_1(n)$ এর মান বের করে পলিনমিয়াল ইন্টারপোলেশন করে দিলেই হচ্ছে।

এখানেও আমরা গাণিতিক আরোহ ব্যবহার করে পুরো জিনিশটা ফরমালি প্রমাণ করতে পারি। বেস কেইস হবে লিফ নোডগুলো। লিফ নোডগুলোয় $f_u(j) = j$ হয়, অর্থাৎ এটাকে আমরা 1 ডিগ্রি পলিনমিয়াল হিসেবে চিন্তা করতে পারি। লিফ ছাড়া অন্য নোড u এর জন্য চাইল্ডের জন্য f_v (v, u এর চাইল্ড) পলিনমিয়াল হবে এটা সত্য ধরে নিয়ে f_u এর জন্যও পলিনমিয়াল হবে এটা প্রমাণ করতে পারি।

অধ্যায় 7

ডিজিট ডিপি

কিছু কিছু সমস্যায় তোমাকে কোন একটা রেঞ্জের মধ্যে বিশেষ কোন ধর্ম সিদ্ধ করে এমন পূর্নসংখ্যা নিয়ে কাজ করতে হয়। এমন সমস্যা দেখলে মনে হয় হয়ত গাণিতিক কোন ধর্ম ব্যবহার করে এগুলো সমাধান করতে হবে। এই ধরনের সমস্যাও যে ডাইনামিক প্রোগ্রামিং দিয়ে সমাধান করা যায় তা সহজে আন্দাজ করা যায় না। ডিজিট ডিপি এমনই একটি টেকনিক। আমরা এ পর্যন্ত যেসব প্রবলেম দেখেছি তার চেয়ে এটি বেশ ভিন্ন ধরনের। তবে মুল আইডিয়াটা ধরতে পারলে এটি মোটেও কঠিন কোন ডিপি নয়।

7.1 সংখ্যা নিয়ে কিছু কথা

ডিজিট ডিপি বুঝতে হলে আমরা দুটি পূর্নসংখ্যা কীভাবে তুলনা করি সেটা ভালোভাবে বুঝতে হবে। দুটি সংখ্যা দেওয়া থাকলে কোনটি কোনটি ছোট সেটা হয়ত একটা বাচ্চাও বলতে পারবে। কিন্তু আমরা সংখ্যা তুলনা করার সময় যে অ্যালগরিদম ব্যবহার করলাম (মনের অজান্তে হলেও) সেটা নিয়ে চিন্তা করি না। ডিজিট ডিপি বোঝার জন্য আমাদের এই প্রসেসটার একটু গভীরে যেতে হবে। একটি উদাহরণ দেখা যাক। ধর তোমার কাছে দুটি সংখ্যা $a = 56744$ এবং $b = 56729$ দেওয়া আছে। তোমাকে বলতে হবে কোনটা বড়। এর জন্য আমরা যেটা করি তা হল সংখ্যা দুটির অঙ্কগুলোকে বাম থেকে ডান দিকে এক এক করে তুলনা করতে থাকি। প্রথম যে সংখ্যাতে ছোট ডিজিট পাবো সেটাকেই ছোট সংখ্যা বলে ঘোষণা করে দিতে পারব। নিচের ছবিটা দেখ। a আর b এর ডিজিটগুলোকে নিচে নিচে লেখেছি।

5	6	7	4	4

5	6	7	2	9

আমরা বাম দিকে থেকে ডিজিটগুলো এক এক করে তুলনা করেছি এবং চতুর্থ ডিজিটে প্রথম ভিন্ন ভিন্ন অঙ্ক পেয়েছি (mismatch পেয়েছি)। উপরের সংখ্যার অঙ্কটি বড় তাই উপরেরটিই বড় সংখ্যা। একটা জিনিশ খেয়াল কর। চতুর্থ ডিজিটের পর কোন কোন ডিজিট আসলো তা কিন্তু আমাদের আর দেখারই দরকার নাই। প্রথম যে পজিশনে ভিন্ন ভিন্ন অঙ্ক পাওয়া গেছে সেটা দিয়েই সংখ্যা দুটি তুলনা করা যাবে। এখানে a আর b তে একই সংখ্যক অঙ্ক ছিল বলে আমাদের সুবিধা হয়েছে। কিন্তু দুটিতে একই সংখ্যক অঙ্ক না থাকলেও কিন্তু আমরা আগে কিছু শূন্য বসিয়ে দুটিকে সমান ডিজিট বিশিষ্ট সংখ্যা বানিয়ে নিতে পারতাম। তাই এই অ্যালগরিদম আসলে যেকোনো দুটি সংখ্যা তুলনা করার ক্ষেত্রেই খাটবে। আর এই আইডিয়া ব্যবহার করেই ডিজিট ডিপির সব কাজ করা হয়।

এবার একটু ভিন্ন দিকে আসা যাক। ধর তোমাকে 123456 এর চেয়ে ছোট একটা সংখ্যা বানাতে বলা হল। কিন্তু তোমার ছোট ভাই এসে বাম দিকের কিছু অঙ্ক অলরেডি বসিয়ে দিয়েছে। তোমাকে বাকি অঙ্কগুলো পূরণ করতে হবে। যেমন নিচের সংখ্যাতে তোমার ভাই প্রথম তিনটা সংখ্যা বসিয়ে দিয়েছে

1	2	0			

এখানে তুমি বাকি দুটি ঘরে যে অঙ্কই বসাও না কেন সংখ্যাটি 123456 এর চেয়ে ছোট হবে। কারণ 123456 এর তৃতীয় ডিজিট 3 কিন্তু আমাদের তৈরি করা সংখ্যাতে তৃতীয় ডিজিট 0। তাই বাকি ঘরগুলোতে যেটাই বসাও না কেন 123456 এর চেয়ে বড় সংখ্যা পাওয়া সম্ভব নয়।

কিন্তু যদি তোমার ছোট ভাইয়ের বসানো সংখ্যাগুলো এমন হয়

1	2	5			

তাহলে তুমি বাকি ঘরগুলোতে যাই বসাও না কেন 123456 এর চেয়ে ছোট সংখ্যা বানাতে পারবে না (একই কারণ)। আরেকটা কেইস আছে। সেটা হল যদি বসানো সংখ্যাগুলো এমন হয়

1	2	3	?		

এ ক্ষেত্রে তোমার কিছু বাধ্যবাধকতা আছে। ? চিহ্নিত ঘরটাতে তুমি যেকোনো সংখ্যা বসাতে পারবে না। তোমাকে সেখানে অবশ্যই 4 এর সমান বা ছোট একটি ডিজিট বসাতে হবে, নাহলে সংখ্যাটি বড় হয়ে যাবে।

আমাদের আলোচনার মূল পয়েন্ট হল তুমি যদি বাম থেকে ডান দিকে ডিজিট বসাতে থাক তাহলে কোন পজিশনে ডিজিট বসানোর সময় কেবল এটা জানাই যথেষ্ট যে মূল সংখ্যার ডিজিটগুলোর সাথে আমাদের বানানো সংখ্যার ডিজিটগুলোর কোথাও মিসম্যাচ (mismatch) হয়েছে কিনা, অর্থাৎ মূল সংখ্যা থেকে ভিন্ন কোনো ডিজিট কোনো পজিশনে বসিয়েছি কিনা। যদি বসিয়ে থাকি তাহলে পরবর্তী ফাঁকা ঘরটাতে আমরা যেকোনো ডিজিট বসাতে পারব। আর যদি না বসিয়ে থাকি তাহলে ফাঁকা ঘরটিতে এমন একটি ডিজিট বসাতে হবে যেন তা মূল সংখ্যার ডিজিটের চেয়ে বড় না হয়ে যায়।

অধ্যায় 8

ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন এবং ইন্টারভাল ডিপি

অধ্যায়ের টাইটেল দেখে হয়তো আন্দাজ করতে পারছো এই ধরনের ডিপিতে স্টেট হিসেবে একটা অ্যারের সাব-অ্যারেকে ডিপিতে স্টেট হিসেবে রাখতে হবে। এই ক্যাটাগরির সবচাইতে ক্লাসিক্যাল উদাহরণ দিয়ে শুরু করা যাক।

8.1 একটি ক্লাসিক্যাল সমস্যা

উদাহরণ 8.1 (ম্যাট্রিক্স চেইন মাল্টিপ্লিকেশন)**.** $n(\leq 500)$ টা ম্যাট্রিক্স আছে তোমার কাছে, তোমাকে সবচাইতে কম কস্টে তোমাকে এদের গুণফল বের করতে হবে। ফরমালি বলতে গেলে, তোমাকে n টা ম্যাট্রিক্স $A_1, A_2, \ldots A_n$ এর dimension গুলো, অর্থাৎ, $(N_1, M_1), (N_2, M_2), \ldots, (N_n, M_n)$ গুলো দেওয়া আছে, যেখানে A_i এর সাইজ হলো $n_i \times m_i$ আর, $M_i = N_{i+1} (1 \leq i < n)$। তোমাকে বের করতে হবে সবচাইতে কম কতটি লুপ চালিয়ে তুমি $A_1 A_2 A_3 \ldots A_n$ বের করতে পারবা। $a \times b$ এবং $b \times c$ সাইজের দুটি ম্যাটিক্স গুন করার কস্ট abc।

সমাধান. তুমি যদি ম্যাট্রিক্স এক্সপোনেন্সিয়েশনের চ্যাপ্টারটি পড়ে থাকো তাহলে জানার কথা ম্যাট্রিক্স মাল্টিপ্লিকেশন একটি অ্যাসোসিয়েটিভ অপারেশন। যেমন, $(AB)C$ আর $A(BC)$ একই জিনিস, অর্থাৎ, A আর B এর গুণফল বের করে সেটাকে C দিয়ে গুন দেওয়া যেই কথা, A কে B আর C এর গুণফল দিয়ে গুন দেওয়াও একই কথা। কিন্তু এদের গুনের অর্ডারের উপর ABC বের করতে কত টাইম লাগবে তা নির্ভর করে। যেমন ধরো, A, B আর C এর সাইজ যথাক্রমে 2×1000, 1000×3, 3×4। যদি $(AB)C$ করি তাহলে কস্ট কত হয় দেখা যাক। প্রথমে AB করার জন্য কস্ট হলো $2 \times 1000 \times 3$, এবং এরপর AB ম্যাট্রিক্সটির সাইজ হবে 2×3। এখন (AB) এর সাথে C গুন করার কস্ট হলো $2 \times 3 \times 4$। সুতরাং মোট কস্ট হবে $2 \times 1000 \times 3 + 2 \times 3 \times 4 = 6024$। কিন্তু $A(BC)$ এর ক্ষেত্রে কস্ট হবে $1000 \times 3 \times 4 + 2 \times 1000 \times 4 = 20000$!

একইভাবে ৪টা ম্যাট্রিক্সকে ৫ ভাবে, ৫টা ম্যাট্রিক্সকে ১৪ ভাবে গুন করতে পারবে। n টা ম্যাট্রিক্সকে যতভাবে গুন করতে পারা যায় তাকে C_{n-1} দিয়ে লেখা যায়, যেখানে C_n হলো n-তম Catalan number। আসলে n টা ম্যাট্রিক্স গুন করার প্রতিটা উপায়কেই আমরা একটা n লিফের পারফেক্ট বাইনারি ট্রি[১] দিয়ে প্রকাশ করতে পারি। আর n টা লিফের C_{n-1} টা ভিন্ন ভিন্ন পারফেক্ট বাইনারি ট্রি আছে। n

[১]পারফেক্ট বাইনারি ট্রিঃ যেই রুটেড ট্রি এর লিফ ছাড়া প্রতিটা নোডের ২টা করে চাইল্ড আছে।

তম Catalan number বের করার ফর্মুলা হলো $\frac{1}{n+1}\binom{2n}{n}$। চিত্র 8.1-তে ৫টা ম্যাট্রিক্স গুন করার সব উপায় দেখানো হয়েছে।

ডায়াগ্রামটা যদি একটু ভালোমত দেখো তাহলে খেয়াল করবা আমরা প্রতিটা উপায় জেনারেট করার জন্য প্রথমে $ABCDE$ এর মঝে কোন এক জায়গায় ভাগ করেছি, ধরো B আর C এর মাঝে ভাগ করলাম, তারপর AB এবং CDE কে যতভাবে গুন করা যায় তা রিকারসিভলি হিসাব করেছি। আর এরপর (AB) কে (CDE) এর সাথে গুণ করার জন্য বিবেচনা করেছি।

সুতরাং আমাদের ডিপি দেখতে এরকম হবেঃ $\text{dp}[l, r] = A_l A_{l+1} A_{l+2} \ldots A_r$ বের করার মিনিমাম কস্ট। বেস কেইসের জন্য $\text{dp}[i, i] = 0$, কারণ একটা ম্যাট্রিক্সের গুণফল বের করতে তো কোন অপারেশনই লাগে না। এখন, l থেকে r ম্যাট্রিক্স গুলোর গুণফল বের করার জন্য আমরা মাঝখানে কোথাও, ধরি i আর $i + 1$ তম ম্যাট্রিক্সের মাঝে ভাগ করলাম, তাহলে আমরা প্রথমে $A_l \ldots A_i$ আর $A_{i+1} \ldots A_r$ বের করার অপ্টিমাল কস্ট হিসাব করবো, যেটা আমরা পাচ্ছি $\text{dp}[l, i]$ এবং $\text{dp}[i + 1, r]$ তে। সাথে $(A_l \ldots A_i) \times (A_{i+1} \ldots A_r)$ করার কস্ট হলো $N_l M_i M_r$, কারণ $(A_l \ldots A_i)$ ম্যাট্রিক্সের এর সাইজ হবে $N_l \times M_i$ আর $(A_{i+1} \ldots A_r)$ ম্যাট্রিক্সের সাইজ হবে $N_{i+1} \times M_r$। সুতরাং $l < r$ এর ক্ষেত্রে ডিপির রিকারেন্স হলোঃ

$$\text{dp}[l, r] = \min_{l \le i < r} \text{dp}[l, i] + \text{dp}[i + 1, r] + N_l M_i M_r$$

ফাইনাল অ্যান্সার হবে $\text{dp}[1, n]$।

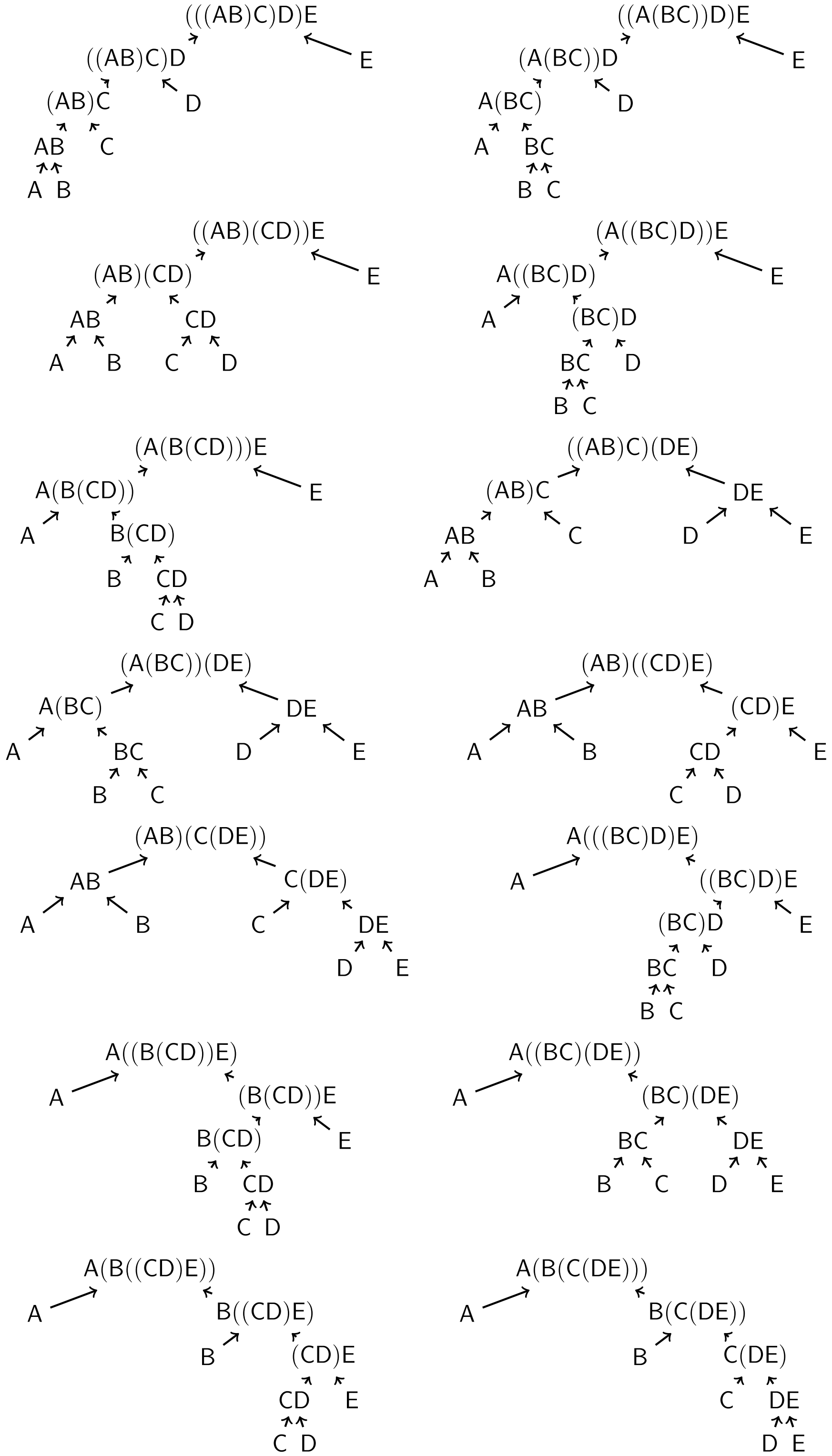
উদাহরণ 8.2 (SPOJ - Mixtures)**.** হ্যারি পটারের সামনে পাশাপাশি একটা সারিতে $n\,(\le 100)$ টা মিশ্রণ সাজানো আছে। প্রত্যেকটা মিশ্রণের ১০০টা রঙের মধ্যে একটা রঙ আছে (০ থেক ৯৯ পর্যন্ত নাম্বারিং করা), i-তম মিশ্রণের রঙ $a_i\,(0 \le a_i \le 99)$। সে সবগুলা মিশ্রণকে একসাথে মিশানোর জন্য $n - 1$ বার এই অপারেশনটি করবেঃ

→ পাশাপাশি ২টা মিশ্রণ নিয়ে তাদের একসাথে মিশিয়ে ২টার মাঝখানে মিশ্রণটা রেখে দিবে, অর্থাৎ, বাকি মিশ্রণ গুলোর ক্রমের কোন পরিবর্তন হবে না। পাশাপাশি নির্বাচন করা মিশ্রণগুলোর রঙ যদি x এবং y হয়, তাহলে তাদের মিশ্রণের রঙ হবে $(x + y) \mod 100$[২]। আর তাদের মিশ্রিত করার সময় xy পরিমাণের ধোঁয়া উৎপন্ন হয়।

তোমাকে বের করতে হবে সবচাইতে কম কতো পরিমাণের ধোঁয়া উৎপন্ন করে মিশ্রণ গুলোকে হ্যারি মিশ্রিত করতে পারবে।

সমাধান. আগের প্রবলেমের মতই এই প্রবলেমেও nটা মিশ্রণকে মিক্স করার যেকোনো উপায়কেই তুমি একটা n লিফের পারফেক্ট বাইনারি ট্রি হিসেবে আঁকতে পারবা। $\text{dp}[l, r]$ হলো l থেকে r এর মধ্যে মিশ্রণ গুলোকে যদি অপ্টিমালি মিশানো হয়, তাহলে সর্বনিম্ন কি পরিমাণের ধোঁয়া উৎপন্ন হবে। এখন তুমি মাঝখানে কোথায় ভাঙ্গবে তার উপর ইটারেট করবা। ধরো, i আর $i + 1$ এর মাঝে ভাঙ্গেছো, তাহলে ২ পাশের কস্ট হলো $\text{dp}[l, i]$ আর $\text{dp}[i+1, r]$। $l \ldots i$ এর মিশ্রণগুলোকে মিক্স করে যেই মিশ্রণ পাবো তার রঙ হবে $\left(\sum_{j=l}^{i} a_j\right) \mod 100$ (কারণ, $(((x+y) \mod m) + z) \mod m = (x+y+z) \mod m$)। একইভাবে $(i + 1) \ldots r$ এর মিশ্রণগুলোকে মিক্স করার পর যেই রঙের মিশ্রণ পাবা তা

[২]এখানে $a \mod m$ দিয়ে a কে m দিয়ে ভাগ করলে যেই ভাগশেষ থাকে তা বুঝানো হচ্ছে।



চিত্র 8.1: ৫টা ম্যাট্রিক্সকে গুন করার সবরকম উপায়

হলো $\left(\sum_{j=i+1}^{r} a_j\right) \mod 100$। এদের মিক্স করতে গেলে আবার $\left(\left(\sum_{j=l}^{i} a_j\right) \mod 100\right) \times \left(\left(\sum_{j=i+1}^{r} a_j\right) \mod 100\right)$ পরিমাণের ধোঁয়া উৎপন্ন হবে। তাহলে রিকারেন্সটা হলোঃ

$$\mathrm{dp}[l, r] = \min_{i=l}^{r-1} \mathrm{dp}[l, i] + \mathrm{dp}[i + 1, r] + L \times R$$

যেখানে, $L = \left(\sum_{j=l}^{i} a_j\right) \mod 100$, $R = \left(\sum_{j=i+1}^{r} a_j\right) \mod 100$, এবং $\mathrm{dp}[i, i] = 0$।

8.2 আরও কিছু উদাহরণ

উদাহরণ 8.3 (USACO - Greedy Pie Eaters)**.** ফার্মার জনের কাছে Mটা গরু আর N টা পাই আছে। গরুগুলো 1 থেকে M এবং পাইগুলো 1 থেকে N পর্যন্ত নাম্বারিং করা। i-তম গরু $[l_i, r_i]$ রেঞ্জের মধ্যে পাইগুলো খেতে পছন্দ করে। তোমাকে আরেকটা জিনিস বলে দেওয়া আছে, তা হলো ২টা গরুর পছন্দের রেঞ্জ একই হবে না কখনো। i-তম গরুর ওজন হলো w_i।

ফার্মার জন একটা সিকুয়েন্স $c_1, c_2, \ldots, c_K$ বাছাই করবে, যেটা দিয়ে গরুগুলো কি অর্ডারে পাই খেতে আসবে তা বুঝাবে, অর্থাৎ, প্রথমে c_1-তম গরুটি আসবে, এরপর c_2-তম গরুটি আসবে...। একটা গরু আসলে সে তার পছন্দের রেঞ্জে বাকি থাকা সব পাই খেয়ে ফেলবে। কিন্তু যদি তার পছন্দের রেঞ্জে কোন পাইই বাকি না থাকে তাহলে সে মন খারাপ করে বসবে! ফার্মার জন এমন একটা সিকুয়েন্স বাছাই করতে চায় যাতে $(w_{c_1} + w_{c_2} + \ldots + w_{c_K})$ এর মান ম্যাক্সিমাইজ হয়, এবং বাছাই করা K টা গরুর মধ্যে কেও মন খারাপ না করে। $1 \leq N \leq 300, 1 \leq M \leq \frac{N(N+1)}{2}, 1 \leq w_i \leq 10^6$।

সমাধান. প্রবলেমটা সল্ভ করার জন্য প্রসেসটাতে কি হচ্ছে তা উলটা দিক থেকে চিন্তা করবো – ধরো তুমি একটা সিকুয়েন্স $c_1, c_2, \ldots, c_K$ ঠিক করেছ। এখন এই সিকুন্সটা ভ্যালিড হতে হলে প্রথমত c_K-তম গরুকে অন্তত একটি পাই পেতে হবে। এরপর, c_{K-1}-তম গরুটিকে c_K-তম গরুটি যেই পাইটি খেয়েছে, সেটা বাদে অন্য আরেকটি পাই খেতে হবে, আবার c_{K-1}-তম গরুটি এমন একটি গরু হতে হবে যাতে তার রেঞ্জে c_K-তম গরুটি যেই পাই খেয়েছে সেটি না থাকে। এভাবে যদি যেতে থাকো তাহলে বুঝতে পারবে আমরা আমাদের প্রবলেমটাকে একটু অন্যভাবে প্রকাশ করতে পারিঃ

এমন দুটি সিকুয়েন্স $c_1, c_2, \ldots, c_K$ এবং $p_1, p_2, \ldots p_K$ বের করো যাতে $(w_{c_1} + w_{c_2} + \ldots + w_{c_K})$ ম্যাক্সিমাইজ হয় এবং সিকুয়েন্সদুটি নিচের শর্তগুলো পালন করেঃ

- p_i দিয়ে বুঝানো হচ্ছে c_i-তম গরুর জন্য কোন পাইটি বরাদ্দ করা হয়েছে। সুতরাং, $p_i \in [l_{d_i}, r_{d_i}]$।
- আমরা d এবং p সিকুয়েন্স দুটি $(c_K, p_K), (c_{K-1}, p_{K-1}), \ldots$ এই অর্ডারে ঠিক করবো, অর্থাৎ আমরা শেষ থেকে গরু ঠিক করছি আর তাদের জন্য একটি একটি করে পাই বরাদ্দ করে যাচ্ছি । কিন্তু প্রশ্ন আসতে পারে, প্রতিটা গরুতো যেই পাই গুলো বাকি থাকবে তা সব খেয়ে ফেলবে, তাহলে আমরা একটা একটা করে বরাদ্দ করছি কেন? আসলে আমরা তো শেষ থেকে দেখছি প্রসেসটাকে – শেষ গরুর জন্য একটা পাই বরাদ্দ করেই আমরা বাকি পাই গুলো নিয়ে কাজ করছি, সেই পাই বাদে আর যেটাই তার আগের গরুগুলো নিয়ে যাক না কেন আমাদের কোন সমস্যা নেই, অন্তত একটা পেলেই হল।

০. কোন গরুর জন্য বরাদ্দকৃত পাইটি তার আগে আসা গরুগুলোর রেঞ্জের মধ্যে থাকতে পারবে না। অর্থাৎ, প্রত্যেক $j < i$ এর জন্য, $p_i \notin [l_{c_j}, r_{c_j}]$ হতে হবে।

ধরো আমরা প্রথমে c_K এবং p_K ঠিক করে ফেলেছি। তাহলে এরপর $c_{K-1}, c_{K-2}, \ldots$ এই গরুগুলো যখন আমরা নির্বাচন করবো, তখন তাদের প্রত্যেকের রেঞ্জই হয় p_k এর পুরাপুরি বামে হবে নাহয় পুরাপুরি ডানে হবে। অর্থাৎ, প্রত্যেক $j < K$ এর জন্য হয় $r_{c_j} < p_K$ হতে হবে নাহয় $l_{c_j} > p_K$ হতে হবে। এর ফলে আমরা একটা খুবই চমৎকার ফলাফল পাবো; বাম পাশ থেকে যেই রেঞ্জ (গরু) গুলো নির্বাচন করবো, তার কোনোটিই ডান পাশ থেকে কোন রেঞ্জগুলো নির্বাচন করতে পারবো তার উপর কোন প্রভাব ফেলবে না – তারা ইন্ডিপেন্ডেন্ট দুটি সাব-প্রবলেম! এরপর আমরা রিকার্সিভলি বলতে পারি $[1, p_K)$ থেকে কিছু গরু নির্বাচন করো, এবং $(p_K, N]$ থেকে কিছু গরু নির্বাচন করো। চিত্র 8.2-এ এভাবে রেঞ্জগুলো বাছাই করার একটি উদাহরণ দেখানো হয়েছে। এরপর মনে করো $[1, p_K)$-তে রেঞ্জ নির্বাচন করবো। খেয়াল করো, এখন কিন্তু যেই গরুগুলো নির্বাচন করবো সেগুলো কিন্তু সম্পুর্নভাবে $[1, p_K)$ রেঞ্জের ভিতর থাকতে হবে। এবার মনে করো d_{K-1} এবং p_{K-1} নির্বাচন করলা $[1, p_K)$ রেঞ্জ থেকে। এর জন্য যেই ২টি সাব-প্রবলেম পাবো আমরা তা হলো $[1, p_{K-1})$ এবং $(p_{k-1}, p_K]$। এভাবে যেতে থাকলে বুঝতে পারবা আমাদের ডিপি স্টেট বা সাব-প্রবলেমকে আমরা ২টি ভ্যালূ দিয়ে প্রকাশ করতে পারি– l এবং r, এবং $\mathrm{dp}[l, r]$ দিয়ে বুঝানো হবে $l \ldots r$ এই পাই গুলো এবং পুরাপুরি $[l, r]$ এর ভিতরে থাকা গরুগুলোকে বিবেচনা করে একটি c সিকুয়েন্স বাছাই করার ম্যাক্সিমাম প্রফিট কতো হতে পারে। আর ডিপি রিকারেন্স হবেঃ

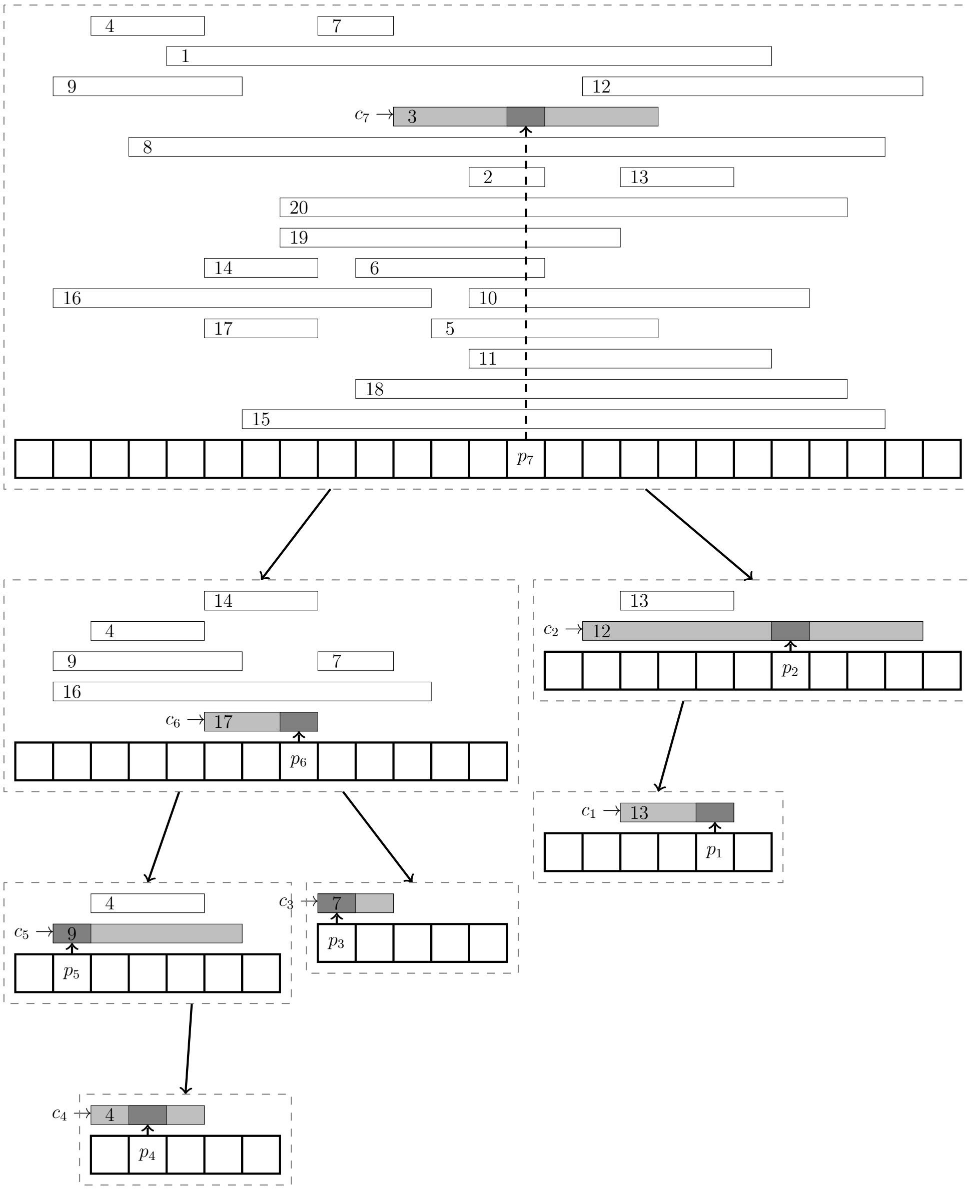
$$\mathrm{dp}[l, r] = \max_{c,\, l \le l_c \le r_c \le r} \max_{p \in [l_c, r_c]} \mathrm{dp}[l, p-1] + \mathrm{dp}[p+1, r] + w_c$$

এর রিকারেন্সের কমপ্লেক্সিটি হলো $O(n^3 m)$। অবশ্য এটাকে একটু অন্যভাবে দেখলে খেয়াল করবা আমাদের আগে গরু ঠিক করে তারপর তার জন্য বরাদ্দ করার জন্য ইনডেক্স ঠিক না করে বরং আগে বরাদ্দকৃত ইনডেক্স ফিক্স করে তারপর ওই ইনডেক্সের উপরে দিয়ে যায় এমন রেঞ্জগুলোর মধ্যে ম্যাক্সিমাম w যার, তাকে নির্বাচন করলেই হয়। সুতরাং আমাদের নতুন রিকারেন্স হবেঃ

$$\mathrm{dp}[l, r] = \max_{p \in [l, r]} \mathrm{dp}[l, p-1] + \mathrm{dp}[p+1, r] + f[l, r, p]$$

যেখানে $f[l, r, p]$ হলো যেসব রেঞ্জ পুরাপুরি $[l, r]$ রেঞ্জের ভিতরে আছে এবং p এর উপরে দিয়ে যায়, তাদের মধ্যে ম্যাক্সিমাম w এর মান। শুরুতে আমাদের f টেবিলটা প্রি-ক্যাল্কুলেট করে নিতে হবে $O(n^3)$ টাইমের মধ্যে। এটাও একটা ছোটোখাটো ডিপি প্রবলেম বলতে পারো, রিকারেন্স এর সাহায্যে ক্যাল্কুলেট করে নিতে পারবা। পাঠকের অনুশীলনের জন্য রেখে দেওয়া হলো।

উদাহরণ 8.4 (Codeforces 1146G - Zoning Restrictions)**.** তুমি n টা বিল্ডিং বানাবে, এবং বিল্ডিং বানানোর স্পটগুলো 1 থেকে n পর্যন্ত নাম্বারিং করা। প্রতিটা বিল্ডিঙের উচ্চতা 0 থেকে h এর মধ্যে যেকোনো একটি পূর্ণসংখ্যা হতে পারে। কোন স্পটে যদি a উচ্চতার বিল্ডিং বানাও তাহলে তুমি a^2 টাকা পাবা। তুমি যাতে ইচ্ছা মতো উচ্চতার বিল্ডিং নির্মাণ করতে না পারো তাই m টা শর্ত দেওয়া আছে – i তম শর্তে তোমাকে বলা আছে l_i থেকে r_i স্পটের বিল্ডিং গুলোর উচ্চতা সর্বোচ্চ v_i হতে পারবে। যদি এদের মধ্যে কোনটার উচ্চতা v_i এর বেশি হয় তাহলে তোমাকে c_i টাকা পেনাল্টি দিতে হবে। খেয়াল করো, l_i থেকে r_i এর মধ্যে একাধিক বিল্ডিং-এর উচ্চতা v_i এর চাইতে বেশি হলেও কিন্তু i-তম শর্ত ভঙ্গের জন্য একবারই c_i টাকা পেনাল্টি দিবে। অপ্টিমালভাবে বিল্ডিং-এর উচ্চতা নির্বাচন করে ম্যাক্সিমাম কতো প্রফিট পেতে পারো তা হিসাব করো। $1 \le n, m, h \le 50, 1 \le l_i \le r_i \le n, 0 \le v_i \le h, 1 \le c_i \le 5\,000$।



চিত্র 8.2: রেঞ্জগুলো রিকার্সিভলি নির্বাচন করার একটি উপায়।

সমাধান. এই প্রব্লেমটা কিছুটা আগের প্রব্লেমের মতো। ধরো তুমি প্রথমে একটা বিল্ডিং i নির্মাণ করবা ঠিক করেছো। এই i নাম্বার বিনল্ডিংটি কিছু কিছু শর্ত ভাঙ্গবে, সেগুলোর পেনাল্টি ধরো তুমি দিয়ে দিলে। এখন পরের বিল্ডিংগুলো যখন নির্মাণ করতে যাবে, তখন i বিল্ডিঙের জন্য কোন কোন শর্ত ভাঙ্গা হয়েছিলো আর কোনগুলোর পেনাল্টি তুমি হিসাবে নিয়ে ফেলেছ এগুলো ট্র্যাক রাখা একটু জটিল হয়ে যাবে। এর জন্য আমরা যেটা করবো তা হলো, প্রথম বিল্ডিংটি এমনভাবে নির্মাণ করবো যেন সেটা সব বিল্ডিংগুলোর মধ্যে ম্যাক্সিমাম উচ্চতার বিল্ডিংগুলোর একটি হয়। এটা করে কি লাভ আমাদের? এটা করার ফলে, এই বিল্ডিংটি নির্মাণ করতে গিয়ে যেইসব শর্তের রেঞ্জ গুলো i নাম্বার বিল্ডিংটির উপর দিয়ে গিয়েছে তাদের মধ্যে যেগুলো মানা হয়নি তাদের পেনাল্টি আমরা এখনি হিসাবে নিয়ে নিবো, আর যেগুলো ভাঙ্গা হয়নি সেগুলো সামনেও কখনো ভাঙ্গা হবে না কারণ ভবিষ্যতের বিল্ডিংগুলোর উচ্চতা সর্বোচ্চ i নাম্বার বিল্ডিঙের উচ্চতার সমান হবে। সেগুলো পরের অন্য বিল্ডিং গুলো নির্মাণের সময় বিবেচনায় রাখারই দরকার নেই। মোটকথা, i নাম্বার বিল্ডিং-এর উপর দিয়ে যেই রেঞ্জ গুলো গিয়েছে তাদের আমরা বিবেচনা করে ফেলেছি, আর ভবিষ্যতে তাদের বিবেচনা করতে হবে না। বাকি যেই রেঞ্জ গুলো আছে, সেগুলো আগের প্রব্লেমের মতই হয় পুরোপুরি i এর বামে হবে, নাহয় পুরোপুরি ডানে হবে, এবং একইভাবে ২টা ইন্ডিপেন্ডেন্ট সাব-প্রবলেম পাবো।

এখন কথা হচ্ছে আমরা i নাম্বার বিল্ডিংটি যাতে সর্বোচ্চ বিল্ডিং গুলোর একটি হয় সেটা নিশ্চিত করবো কিভাবে? এটার জন্য আমরা ডিপিতে আরেকটি স্টেট রাখবোঃ নির্মানকৃত বিল্ডিং গুলোর উচ্চতা সর্বোচ্চ কত হতে পারবে, তাহলে আমরা অনেকটা এরকম ইনফরমেশন পাস করতে পারবোঃ "এই ২টি সাব-প্রবলেমের মধ্যে বিল্ডিং এর সর্বোচ্চ উচ্চতা i তম বিল্ডিং এর সমান হতে পারবে"।

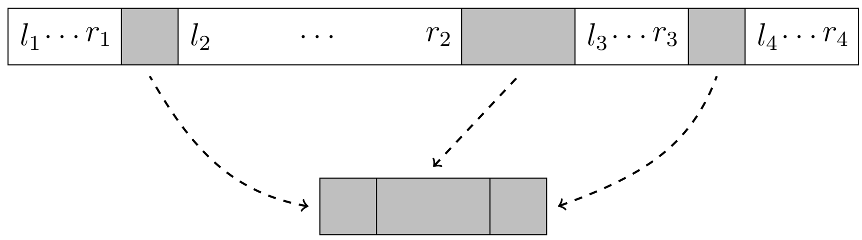
সুতরাং আমাদের ডিপির স্টেট ৩টা হবেঃ l, r এবং x, আর $\mathrm{dp}[l, r, x]$ এর মানে হলো শুধুমাত্র সেইসব শর্তগুলো, যেগুলোর রেঞ্জ সম্পূর্ণ $[l, r]$ এর ভিতরে, সেগুলো বিবেচনা করে $l \dots r$ বিল্ডিংগুলো নির্মাণ করে ম্যাক্সিমাম কত প্রফিট পাওয়া যায় যাতে কোন বিল্ডিং-এর উচ্চতা x এর বেশি না হয়। $\mathrm{dp}[l, r, x]$ ক্যাল্কুলেট করার জন্য আমরা সর্বোচ্চ বিল্ডিং কোনটা সেটার উপরে, এবং সেই বিল্ডিং-এর উচ্চতা কতোটুক হবে – এই ২টির উপর ইটারেট করবো। তাহলে রিকারেন্সটা হবেঃ

$$\mathrm{dp}[l, r, x] = \max_{l \le i \le r, 0 \le y \le x} \mathrm{dp}[l, i-1, y] + \mathrm{dp}[i+1, r, y] - f[l, r, i, y] + y^2$$

, যেখানে $f[l, r, i, y]$ হলো এমন সব শর্ত j, যেগুলো সম্পূর্ণভাবে $[l, r]$ এর ভিতরে, i এর উপর দিয়ে গিয়েছে, এবং $v_j < y$, তাদের c_j এর যোগফল।

উদাহরণ 8.5 (Codeforces 1107E - Vasya and Binary String)**.** তোমার কাছে n লেংথের একটি বাইনারি স্ট্রিং s, এবং n সাইজের একটি অ্যারে s আছে। স্ট্রিংটা খালি না হয়ে যাওয়া পর্যন্ত তুমি এই অপারেশনটি অ্যাপ্লাই করবাঃ s এর মধ্যে একটা কন্সেকিউটিভ সাবস্ট্রিং বাছাই করবা যাতে সেই সাবস্ট্রিং এর সব ক্যারেক্টার একই হয়, এবং সেই সাবস্ট্রিংটা ডিলিট করে এরপর ২ পাশের বাকি থাকা সাবস্ট্রিংগুলো (এগুলোর কোনটা ফাকা হলে সমস্যা নেই) জোড়া লাগিয়ে দিবা। যেমন $1111101 \to 1101$। x লেংথের সাবস্ট্রিং ডিলিট করলে a_x পয়েন্ট পাবে। ম্যাক্সিমাম কতো পয়েন্ট পেতে পারো তুমি? $1 \le n \le 100, 1 \le a_i \le 10^9$।

সমাধান. এখানেও আমরা উল্টা দিক থেকে চিন্তা করবো। একদম শেষ অপারেশনটার কথা চিন্তা করো – সেখানে সবগুলো ক্যারেক্টার একই হবে। কিন্তু সেগুলা একসাথে জড়ো হওয়ার আছে নিশ্চয়ই প্রাথমিক অ্যারেতে কোনো সাব-সিকুয়েন্স হিসেবে ছিল! চিত্রে এমন একটা উদাহরণ দেখানো হয়েছে। নিচের অ্যারেটি পেতে হলে আগে তোমাকে $l_1 \dots r_1$, $l_2 \dots r_2$, $l_3 \dots r_3$ এবং $l_4 \dots r_4$ রেঞ্জগুলো ডিলিট করতে হবে। আর এই রেঞ্জগুলো এককেটা সাব-প্রবলেম, যারা পুরোপুরি ইন্ডিপেন্ডেন্ট!



তাহলে বুঝতে পারছো আমাদের ডিপিতে যেই ২টা স্টেট থাকতেই হবে সেগুলো হলো l এবং r – অ্যারের একটি রেঞ্জ। এরপর ট্রানজিশন করার সময় আমরা এই রেঞ্জের একটা সাব -সিকুয়েন্স বাছাই করতে পারি যার সব ক্যারেক্টার একই। ধরো সাব-সিকুয়েন্সটির লেংথ x, তাহলে আমরা a_x অ্যাড করবো, আর সাব-সিকুয়েন্স-এর মাঝে মাঝে যেই সাব-অ্যারে গুলো পাবো সেগুলোর ডিপি ভ্যালু গুলো যোগ করবো। কিন্তু এভাবে করলে আমাদের ট্রানজিশন এক্সপোনেনশিয়াল সংখ্যক হয়ে যাচ্ছে।

এর জন্য আমরা যেটা করবো সেটা হলো বাম থেকে ডানে সাব-সিকুয়েন্স একটা একটা ইনডেক্স করে বানাবো। খেয়াল করো, সাব-সিকুয়েন্সে কোন কোন ইনডেক্স আছে তা কিন্তু আমাদের ট্র্যাক রাখতে হচ্ছে না। আমরা শুধু ট্র্যাক রাখবো ইতোমধ্যে কয়টা ইনডেক্স নিয়ে ফেলেছি সাব-সিকুয়েন্সে।

এই সবগুলো আইডিয়া মিলিয়ে আমরা যেই ডিপি পাচ্ছি তা হলো, $\mathrm{dp}[l, r, x]$, যার মানে হলো আমাদেরকে $s[l \ldots r]$ পুরোটা ডিলিট করতে হবে, সাথে বলা আছেঃ $s[1 \ldots (l-1)]$ এর কিছু কিছু সাব-অ্যারে ডিলিট করে আমাদের কাছে $s[1 \ldots (l-1)]$ এর যেই সাব-সিকুয়েন্স বাকি আছে, তার লেংথ x এবং এদের সবার ক্যারেক্টার s_l এর সমান।

ট্রানজিশন গুলো হবেঃ

- আমরা শুরুতে যেই সাব-সিকুয়েন্স বাছাই করা শুরু করেছিলাম, সেটার শেষ ইন্ডেক্সটাই হলো l, তাহলে আগের x টা ইনডেক্সসহ মিলে মোট $x+1$ লেংথ এর একটা সাব-সিকুয়েন্স পাচ্ছি, এর পয়েন্ট হলো a_{x+1}। এরপর আবার আমাদেরকে $s[l+1 \ldots r]$ এই সাব অ্যারেটি ডিলিট করতে হবে, সেটার ম্যাক্সিমাম পয়েন্ট হলো $\mathrm{dp}[l+1, r, 0]$।

- আমরা জানি s_l হলো সাব-সিকুয়েন্সে বাছাই করা উপাদান গুলোর মধ্যে একটি। এখন আমরা $(l+1) \ldots r$ রেঞ্জে ইটারেট করবো কোন ইন্ডেক্সটা সেই সাব-সিকুয়েন্সের পরবর্তী ইনডেক্স হবে। ধরো সেটা হলো i (এখানে $s_i = s_l$ হতে হবে কিন্তু!)। তাহলে আমাদের আগে $s[(l+1) \ldots (i-1)]$ কে ডিলিট করতে হবে। যার ম্যাক্সিমাম পয়েন্ট হলো $\mathrm{dp}[l+1, i-1, 0]$। এরপর আমাদের আগের মতো সাব-সিকুয়েন্স বাছাই করে করে আগাতে হবে, যার ম্যাক্সিমাম পয়েন্ট হলো $\mathrm{dp}[i, r, x+1]$।

টাইম কমপ্লেক্সিটি হলো $O(n^3)$।

উদাহরণ 8.6 (XXI Open Cup, GP of Suwon - Generate The Array)**.** ধরো তোমাকে একটা N লেংথের অ্যারে A দেওয়া আছে, এবং তুমি এতে কিছু কুয়েরি করবাঃ অ্যারের একটা সেগমেন্ট $[i, j]$ এর জন্য সেই সেগমেন্টের ম্যাক্সিমাম বের করবা, ধরো $[i, j]$ সেগমেন্টের ম্যাক্সিমাম $R_{i,j}$। $[i, j]$ সেগমেন্টটি $Q_{i,j}$ বার করা হবে।
কিন্তু অ্যারেটা তোমাকে দেওয়া নাই, তুমি বানাবে সেটা। 1 থেকে N এর মধ্যে প্রতিটা i এর জন্য A_i এর মান হিসেবে তুমি K_i টা আলাদা আলাদা মান $V_{i,1}, V_{i,2}, \ldots, V_{i,K_i}$ থেকে একটি বাছাই করতে পারবা। A_i এর জন্য $V_{i,j}$ বাছাই করার কস্ট হলো $C_{i,j}$। ধরো, i-তম ইনডেক্সের জন্য P_i-তম ভ্যালুটি বাছাই করেছ, অর্থাৎ $A_i = V_{i,P_i}$।

সবগুলো কুয়েরির শেষে তোমার স্কোর হবেঃ (সব ইন্টারভাল কুয়েরির রেজাল্টের যোগফল) $-$ (A_i ভ্যালু গুলা বাছাই করার কস্টের যোগফল)। অর্থাৎ,

$$\text{স্কোর} = \sum_{1 \le i \le j \le N} Q_{i,j} \cdot R_{i,j} - \sum_{i=1}^{N} C_{i,P_i}$$

ম্যাক্সিমাম কতো স্কোর পেতে পারো তা বের করো। তোমাকে N, Q, C, V, K দিয়ে দেওয়া হবে। $1 \le N \le 300$, $0 \le Q_{i,j} \le 999$, $0 \le V_{i,j} \le 10^8$, $0 \le C_{i,j} \le 10^{13}$, $\sum K_i \le 3 \cdot 10^5$।

সমাধান. আগের প্রবলেম গুলোর সলিউশনের আলোচনা থেকে আশা করি বুঝতে পারছো আমাদের এখানে একটা রেঞ্জের উপর সাব-প্রবলেম সল্ভ করতে হবে। যেহেতু আমাদেরকে রেঞ্জের ম্যাক্সিমাম নিয়ে কাজ করতে হচ্ছে, সেজন্য আমরা আমাদের বর্তমান সাব-প্রবলেমের রেঞ্জে (ধরো, $l \dots r$) কোন ইন্ডেক্সটা ম্যাক্সিমাম ভ্যালু হবে তা ঠিক করবো। ধরো $i \in [l, r]$ ইনডেক্সটি হলো ম্যাক্সিমাম ভ্যালুর ইনডেক্স। এখন আমরা i-তম ইনডেক্সে কোন ভ্যালু বসাবো তা ঠিক করবো এবং $[l, i)$ আর $(i, r]$ ইন্টারভালে রিকার্স করবো – সাথে বলে দিবো এই ২টি ইন্টারভালের ভ্যালু গুলো i-তম ইনডেক্সের ভ্যালুর চেয়ে ছোট হবে।

সবকিছু গুছালে আমাদের নাইভ সলিউশনটি এমন দাঁড়াবেঃ ডিপির স্টেট ৩টি – l, r এবং x, আর $\mathrm{dp}[l, r, x]$ এর মানে হলো, $A[l \dots r]$ এই অ্যারেটা বানানো ম্যাক্সিমাম স্কোর, যাতে সব $i \in [l, r]$ এর জন্য $A_i \le x$ হয়। এর রিকারেন্স হবেঃ

$$\mathrm{dp}[l, r, x] = \max_{i=l}^{r} \max_{j=1}^{K_i} \mathrm{dp}[l, i-1, V_{i,j}] + \mathrm{dp}[i+1, r, V_{i,j}] + T[l, r, i] \cdot V_{i,j} - C_{i,j}$$

, যেখানে $T[l, r, i]$ হলো $[l, r]$ রেঞ্জের ভিতর আছে এবং i এর উপর দিয়ে যায়, এমন কয়টি কুয়েরি আছে, অর্থাৎ $T[l, r, i] = \sum_{l \le l' \le i \le r' \le r} Q_{i,j}$। এটা $O(N^3)$ টাইমের মধ্যে প্রি-ক্যাল্কুলেট করে নেওয়া যাবে। এই ডিপির স্টেট আছে $N \times N \times \left(\sum_{i=1}^{N} K_i\right)$ টি, আর প্রতি স্টেটে মোটামুটি $O(N) \times O(K_i)$ সাইজের লুপ চলছে। আর যাই হোক, আমাদেরকে যেই কন্সট্রেইন্ট গুলো দেওয়া হয়েছে, তাতে এই ডিপি কাজ করবে না।

এখন আমাদের নতুন কিছু অবজারভেশন লাগবে ডিপিটিকে অপ্টিমাইজ করতে।

অপটিমাইজেশন 8.6.1. $l \dots r$ এর মধ্যে i কে ম্যাক্সিমাম ধরার পর $l \dots (i-1)$ এবং $(i+1) \dots r$ এর ভ্যালু গুলো যাতে সর্বোচ্চ i এর সমান হয়, এটা নিশ্চিত করার দরকার নেই – ওই ভ্যালু গুলো যদি i এর চাইতে বড়-ও হয়ে যায়, তাহলে সেটা কখনো অপ্টিমাল হবে না, এবং আমাদের ফাইনাল অ্যান্সারে প্রভাব ফেলবে না। এর ফলে আমাদের যা লাভ হবে তা হলো আমরা ডিপির তৃতীয় স্টেটটা (x) বাদ দিয়ে দিতে পারি।

এই অবজারভেশনটা বুঝার জন্য এভাবে চিন্তা করোঃ ধরো তোমার কাছে A এর ভ্যালু গুলা ফিক্স করা আছে। তাহলে একটা জিনিশ খেয়াল করো, এর ফলে কিন্তু আমাদের স্কোর ফাংশনের $-\sum_{i=1}^{N} C_{i,P_i}$ এই অংশটাও ফিক্স হয়ে গিয়েছে।

$$\text{স্কোর} = \sum_{1 \le i \le j \le N} Q_{i,j} \cdot R_{i,j} \boxed{- \sum_{i=1}^{N} C_{i,P_i}}$$

আর বাকি থাকছে শুধু $\sum_{1 \le i \le j \le N} Q_{i,j} \cdot R_{i,j}$ অংশটা। আমাদের টার্গেট হলো এটাকে ম্যাক্সিমাইজ করা। চিন্তা করে দেখো, আমরা যদি ডিপির স্টেটে x কে না রেখে অ্যারের বিভিন্ন সাব-অ্যারের ম্যাক্সিমামকে

ভুলভাবে চিহ্নিত করি, তাহলে কিন্তু সেটা থেকে যেই $\sum_{1\le i\le j\le N} Q_{i,j} \cdot R_{i,j}$ এর মান পাবো, সেটা সবসময়ই অ্যারের ম্যাক্সিমাম গুলোকে ঠিক মতো আন্দাজ করতে পারলে যেই মান পাবো তার চেয়ে ছোট অথবা সমান হবে। আর আমাদের ডিপিতে আমরা যেহেতু ডিপিতে সব উপায়ে আন্দাজ করে করে দেখছি, সেহেতু একটা না একটাতে আমরা সঠিকভাবে আন্দাজ করে ফেলবো, এবং আমাদের ফাইনাল অ্যান্সার সবসময় ঠিক হবে। তাহলে আমাদের নতুন ডিপিটা হবেঃ

$$\mathrm{dp}[l,r] = \max_{i=l}^{r} \max_{j=1}^{K_i} \mathrm{dp}[l,i-1] + \mathrm{dp}[i+1,r] + T[l,r,i] \cdot V_{i,j} - C_{i,j}$$

অপটিমাইজেশন 8.6.2. কনভেক্স হাল ট্রিক।

আগের ডিপি ফরমুলাটাকে আমরা একটু অন্যভাবে লিখতে পারিঃ

$$\mathrm{dp}[l,r] = \max_{i=l}^{r} \mathrm{dp}[l,i-1] + \mathrm{dp}[i+1,r] + \boxed{\max_{j=1}^{K_i} T[l,r,i] \cdot V_{i,j} - C_{i,j}}$$

আমরা প্রতি (l,r,i) টুপলের জন্য $\max_{j=1}^{K_i} T[l,r,i] \cdot V_{i,j} - C_{i,j}$ এর মান আগে থেকে $O(N^3 + \sum K_i)$ টাইমে বের করে ফেলতে পারবো। কিভাবে? যদি তুমি i ফিক্স করো, তাহলে K_i টা $mx+b$ আকারের লাইন ইকুয়েশন পাবা, যেখানে $m = V_{i,j}$ এবং $b = -C_{i,j}$। এবার প্রতি (l,r) পেয়ারের উপর ইটারেট করে $x = T[l,r,i]$-তে মিনিমাম ভ্যালু কতো তা বের করতে হবে, আর এই প্রবলেমটাই কনভেক্স হাল ট্রিক দিয়ে সল্ভ করা হয়। বিস্তারিত পড়তে চাইলে কনভেক্স হাল ট্রিক চ্যাপ্টারটি পড়ো।

8.3 ইমপ্লিমেন্টেশন ট্রিক

এইধরনের প্রবলেমে সাধারণত $\mathrm{dp}[l,r]$ ক্যাল্কুলেট করার জন্য $\mathrm{dp}[l,i]$ এবং $\mathrm{dp}[i,r]$ এসব ডিপি ভ্যালুর মান জানা থাকতে হয়। যদি তুমি রিকার্সিভ মেময়িজেশন দিয়ে ইমপ্লিমেন্ট করে থাকো, তাহলে তো সহজই, আর কিছু চিন্তা করতে হবে না। কিন্তু অনেক সময় সলিউশনের কন্সটেন্ট ফ্যাক্টর কমানোর বা রান টাইন কমানোর জন্য আমাদের বটম-আপ ইমপ্লিমেন্টেশনের দরকার হয়। এইধরনের ডিপির বটম-আপ ইমপ্লিমেন্টেশনের ট্রিক হলো, ডিপি ভ্যালু গুলো $r-l$ এর উর্ধক্রমে ক্যাল্কুলেট করা। নিচের সুডোকোডটা দেখো

Algorithm 1: ইন্টারভাল ডিপি ক্যাল্কুলেট করার একটি বটম-আপ পদ্ধতি।

Result: Calculates dp table.
for $d \leftarrow 0$ **to** $n-1$ **do**
    for $l \leftarrow 1$ **to** n **do**
        $r \leftarrow l + d$;
        for $i \leftarrow l$ **to** r **do**
            relax $\mathrm{dp}[l,r]$ using $\mathrm{dp}[l,i]$ and $\mathrm{dp}[i,r]$;

8.4 অনুশীলনী

অনুশীলনী 8.1 (AtCoder - Visibility Sequence)**.** আগের বিল্ডিং বানানোর ঠিকাদারিতে তুমি ব্যাপক পরিমাণের লাভ করেছ (ডাইনামিক প্রোগ্রামিংকে ধন্যবাদ না দিলেই নয়), তাই তুমি আবারো

পরিকল্পনা করেছ N টা বিল্ডিং বানাবে। এইবারের শর্তগুলো হলো, প্রতিটা $i\,(1 \leq i \leq N)$ এর জন্য তোমাকে একটা X_i দেয়াও আছে, যার মানে হলো i তম বিল্ডিংয়ের উচ্চতা 1 থেকে X_i এর মধ্যে যেকোনো একটি পুর্ণসংখ্যা হতে পারবে। ধরো তুমি i তম বিল্ডিং বানিয়েছ H_i উচ্চতার। এখন প্রতি $i\,(1 \leq i \leq N)$ এর জন্য আমরা P_i কে এভাবে ডিফাইন করবোঃ যদি এমন কোন পর্ণসংখ্যা $j\,(1 \leq j < i)$ থাকে যাতে $H_j > H_i$ হয়, তাহলে P_i হবে এমন ম্যাক্সিমাম j, আর নাহলে $P_i = -1$। এবার H সিকুয়েন্সটির সবরকম কম্বিনেশনের কথা চিন্তা করো, তারা প্রত্যেকেই একটি করে P জেনারেট করবে। দুটি ভিন্ন H এর জন্য তাদের জেনারেট করা P একই হয়ে যেতে পারে আবার ভিন্নও হতে পারে। তোমাকে বের করতে হবে, কয়টা ভিন্ন ভিন্ন P জেনারেট হবে। $1 \leq N \leq 100, 1 \leq X_i \leq 10^5$।

অধ্যায় 9

Zeta Transform, Möbius Inversion এবং Subset Convolution

9.1 Zeta Transform

ধরো তোমার কাছে একটি ফাংশন f আছে, যেটা $N = \{0, 1, 2, \ldots n-1\}$ এর একটি সাবসেট ইনপুট নেয় এবং একটি ইন্টিজার রিটার্ন করে। অর্থাৎ, f এর ডোমেইন হলো $\mathcal{F}$, যেটা $\{0, 1, 2, \ldots n-1\}$ এর সব সাবসেটের ফ্যামিলি, আর কোডমেইন হলো পূর্ণসংখ্যার সেট (অন্য কিছুও হতে পারে, খালি ২টি উপাদানের কম্পোজিশন সংজ্ঞায়িত হলেই হবে)। $\mathcal{F}$ এর প্রতিটি উপাদানকে এমরা $\{0,1\}^n$ এর একটা উপাদান, অর্থাৎ একটি n-টুপল বা n লেংথের বাইনারি সিকুয়েন্স/স্ট্রিং/নাম্বার দিয়ে প্রকাশ করতে পারি। $\{0,1\}^n$ বলতে $\underbrace{\{0,1\} \times \{0,1\} \times \cdots \times \{0,1\}}_{n \text{ সংখ্যক}}$ বুঝানো হচ্ছে, যেখানে $A \times B$ মানে হলো A এবং B সেট দুটির কার্তেসিয় গুণন। মূলত, $\{0,1\}^n$ এর প্রতিটা উপাদান হলো এককেকটি n-টুপল। যেমন, $n = 4$ হলে এমন একটি টুপল হলো $(0, 1, 1, 0)$। এই টুপল না বাইনারি নাম্বারের i-তম বিট 0 হয়, তার মানে হলো সাবসেটটিতে i নেই, আর যদি 1 হয় তাহলে i আছে।

আমাদেরকে যেই প্রবলেমটা সল্ভ করতে হবে তা হলোঃ যদি আমাদের f দিয়ে দেওয়া হয়, তাহলে আরেকটি একই প্রকৃতির ফাংশন $\hat{f}$ (অর্থাৎ, $\hat{f}$ এর ডোমেইন এবং কোডোমেইন যথাক্রমে f এর ডোমেইন এবং কোডোমেইনের সমান)ক্যাল্কুলেট করতে হবে যেটার সংজ্ঞা হলোঃ

$$\hat{f}[x] = \sum_{y \subseteq x} f[y]$$

অন্যভাবে বললে, প্রতি $x \in \mathcal{F}$-এর জন্য x এর যত সাবসেট y আছে, তাদের $f[y]$ এর যোগফল বের করা। আমরা যদি বিটমাস্কের ভাষায় বলি তাহলে দাঁড়ায় x এর সব সাবমাস্ক y এর জন্য $f[y]$ এর সাম বের করা। খেয়াল করো, আমরা কিন্তু প্রতিটা সেটকেই সেটার বাইনারি সিকুয়েন্সকে ইন্টিজারে রূপান্তর করে একটা ইন্টিজার দিয়ে প্রকাশ করতে পারি। f থেকে $\hat{f}$ এর এই ট্রানফর্মেশনকে Zeta transform বলা হয়। যেহেতু সব সাবসেটের সাম নেওয়া হচ্ছে তাই একে অনেকে সাম ওভার সাবসেটও (Sum Over Subset, বা SOS) বলে। f এর Zeta transform-কে আমরা $\zeta(f)$ দিয়ে লিখবো। অর্থাৎ, $\hat{f} = \zeta(f)$।

এখানে অবশ্য n এর মান এমন হবে যাতে 2^n এর মান ছোট হয়। কারণ f কে ডিফাইন করতেই তো $O(2^n)$ সাইজের ইনপুট প্রয়োজন হবে!

এই চ্যাপ্টারের আলোচনায় আমরা সাবসেটকে বিটমাস্ক লিখবো অনেক সময়, আবার অনেক সময় বিটমাস্ককে সাবসেট লিখবো। যখনই কোন সাবসেটের বিটমাস্ক উল্লেখ করা হবে, তখন বুঝে নিতে হবে এমন একটি বিটমাস্ক নিয়ে কথা বলা হচ্ছে যেটার i-তম বিট অন থাকবে যদি ও কেবল যদি সেটটির মধ্যে i উপাদানটি বিদ্যমান থাকে। আরেকটা জিনিস হলো আমরা $f, \hat{f}$ এগুলোকে ফাংশন বলেও স্কয়ার ব্র্যাকেট ব্যবহার করছি। আসলে এটা তেমন আহামরি কিছু না, এগুলোও যেহেতু আমাদের জন্য একেকটা ডিপি টেবিল, তাই প্যারেন্থেসিস (parenthesis) এর বদলে খালি স্কয়ার ব্র্যাকেট ব্যবহার করা হয়েছে। এছাড়াও, এই চ্যাপ্টার জুড়ে ফাংশনের কয়েকটি নোটেশন দেখতে পাবে – $f(x)$, $f[x]$, f_x সব একই; x দ্বারা ফাংশনের ইনপুট/প্যারামিটার/আর্গুমেন্ট/ডিপি টেবিলের ইনডেক্স বুঝানো হবে।

$\hat{f}$ কিভাবে ইফিশিয়েন্টলি ক্যাল্কুলেট করা যায় তা শিখার আগে একটা ছোট্ট অ্যাপ্লিকেশন দেখে নেই।

উদাহরণ 9.1. একটি $N(\leq 10^5)$ সাইজের পুর্ণসংখ্যার অ্যারে a দেওয়া আছে, যেখানে প্রতিটি উপাদান $a_i < 2^{20}$ হবে। তোমাকে প্রতিটি $i \in [1, N]$ এর জন্য ক্যাল্কুলেট করতে হবেঃ

- প্রথম সমস্যা: এমন কয়টা $j \in [1, N]$ আছে, যাতে $a_i \,\&\, a_j = a_j$ হয়, যেখানে & হলো বিটওয়াইজ অ্যান্ড অপারেটর।
- দ্বিতীয় সমস্যা: এমন কয়টা $j \in [1, N]$ আছে, যাতে $a_i \mid a_j = a_j$ হয়, যেখানে | হলো বিটওয়াইজ অর অপারেটর।
- তৃতীয় সমস্যা: এমন কয়টা $j \in [1, N]$ আছে, যাতে $a_i \,\&\, a_j = 0$ হয়।

সমাধান. $a_i \,\&\, a_j = a_j$ হবে যদি এবং কেবল যদি a_j এবং a_i কে বাইনারিতে লিখলে a_j, a_i এর সাবমাস্ক হয়। কারণ, যদি a_j তে এমন কোন অতিরিক্ত বিট অন থাকে যেটা a_i তে অফ আছে, সেই অতিরিক্ত বিটগুলো $a_i \,\&\, a_j$-তে অফ হয়ে যাবে। আবার $a_i \,\&\, a_j = a_j$ যদি হয় তাহলে বলা যায় a_j-তে যেই বিটগুলো আছে, সেগুলোর সবগুলোই a_i তেও আছে, সুতরাং $a_j \subseteq a_i$।

এখন আমরা f কে সংজ্ঞায়িত করবো এভাবেঃ $f[x]$ হলো অ্যারেটিতে এমন কয়টা উপাদান আছে যাদেরকে বাইনারিতে লিখলে সেই বিটমাস্কটা x এর সমান হয়। এবার যদি আমরা f এর সাম ওভার সাবসেট নিয়ে g পাই, তাহলে i এর জন্য অ্যান্সার হবে $g[a_i]$। উল্লেখ্য যে, এই প্রবলেমে সব বিটমাস্কের সাইজ হবে $n = 20$ কারণ সব $a_i \leq 2^n$।

দ্বিতীয় প্রবলেমের ক্ষেত্রে $a_i \mid a_j = a_j$ হবে যদি ও কেবল যদি a_j, a_i এর সুপারমাস্ক[১] হয়। সাম ওভার সাবসেটের সলিউশন শিখার পর সেটা একটু এডিট করেই সাবমাস্কের পরিবর্তে সুপারমাস্কের যোগফল ক্যাল্কুলেট করতে পারবা। কিন্তু সেটা ছাড়াও আমরা শুধুমাত্র সাম ওভার সাবসেটের কোড ব্যবহার করেই সাম ওভার সুপারমাস্ক ক্যাল্কুলেট করতে পারি। নিচের বৈশিষ্ট্যটি খেয়াল করো:

$$x \subseteq y \Leftrightarrow \overline{x} \supseteq \overline{y}$$

যদি আমরা আরেকটি ফাংশন f'-কে এমনভাবে ডিফাইন করি যাতে $f'[x] = f[\overline{x}]$ হয়, তাহলে i এর জন্য অ্যান্সার হবে $\zeta(f')[a_i]$ হবে।

তৃতীয় প্রবলেমের জন্য সমাধান হলো $\zeta(f)[\overline{a_i}]$।

[১] x-এ y এর সুপারমাস্ক বলা হয় যদি x-এ y-এর সব বিটগুলোই থাকে। অনেকটা সাবমাস্কের উল্টা সংজ্ঞা।

9.2 $O(3^n)$ কমপ্লেক্সিটির ব্রুটফোর্স সলিউশন

একদম সাদামাটা ব্রুটফোর্সটা হলোঃ

Algorithm 2: 4^n কমপ্লেক্সিটিতে সাবসেট সাম বের করার সুডোকোড।

Result: f দেওয়া থাকলে আরেকটি ফাংশন $\hat{f}$ ক্যাল্কুলেট করবে।
initialize an array $\hat{f}$ of size 2^n with 0s;
for $x \in \{0, 1, \ldots, n-1\}$ **do**
  for $y \in \{0, 1, \ldots, n-1\}$ **do**
    if $y \subseteq x$ **then**
      $\hat{f}[x] \leftarrow \hat{f}[x] + f[y]$;

একে C++-এ লিখলে হবেঃ

```
vector<int> f(1 << n);
// take input of f
vector<int> fhat(1 << n, 0);
for(int x = 0; x < (1 << n); ++x) {
  for(int y = 0; y < (1 << y); ++y) {
    if((x & y) == y) {
      fhat[x] += f[y];
    }
  }
}
```

এই কোডের দ্বিতীয় লুপটায় অনেক ইটারেশন অপচয় হচ্ছে। আমরা কোনোভাবে যদি শুধুমাত্র x এর সাবসেটগুলোতে অর্থাৎ, এমনসব y তে ইটারেট করতে পারতাম যাতে `(x & y) == y` শর্তটা পূরণ হয়, তাহলে আরেকটু ইফিশিয়েন্ট করতে পারতাম।

1111 এর সাবমাস্ক গুলো যদি আমরা বড় থেকে ছোট অর্ডারে লিখি তাহলে পাবোঃ

1111
1110
1101
1100
1011
1010
1001
1000
0111
0110
0101
0100

```
0011
0010
0001
0000
```

এগুলো পাওয়ার জন্য আমরা $15, 14, 13, \ldots, 0$ এর উপর লুপ চালাতে পারিঃ

```
for(int i = 15; i >= 0; --i) {
  // binary representation of i is a submask of 1111
  cout << bitset<4>(i) << '\n';
}
```

একইভাবে আমরা 10110 এর সাবসেটের উপরেও এই অর্ডারে লুপ চালাবোঃ

```
10110
10100
10010
10000
00110
00100
00010
00000
```

আগের মতো এখানেও যদি আমরা এক বিয়োগ করে করে যেতে থাকি তাহলে হবে না। কারণ 10110 এর পর 10101-এ যাবে। কিন্তু খেয়াল করো, আমরা বিয়োগ করার পর 10101 এর বিট গুলোকে 10110 দিয়ে ফিল্টার করে নিতে পারি, অর্থাৎ 10110 দিয়ে অ্যান্ড করে নিবো।

$$\begin{aligned}\texttt{mask} &= \texttt{11010010001111100000}\\ \texttt{submask-1} &= \texttt{1101001000}\underbrace{\texttt{XXXXX}}_{\text{শুধু এই অংশটি দেখলে মনে হবে এটি 11111 এর সকল সাবসেটের উপরে ইটারেট করছে}}\texttt{11111}\end{aligned}$$

নিচে C++-এ একটি বিটমাস্ক mask এর সব সাবমাস্কের উপর ইটারেট করে $\hat{f}$ ক্যাল্কুলেট করার কোড দেওয়া হলোঃ

```
vector<int> fhat(1 << n, 0);
for(int mask = 0; mask < (1 << n); ++mask) {
  for(int submask = mask; submask > 0; submask = (submask-1) & mask) {
    fhat[mask] += f[submask];
  }
  // we have to consider the empty set separately
  fhat[mask] += f[0];
}
```

এর কমপ্লেক্সিটি কতো? যদি $T(n)$ দ্বারা এমন কয়টা (x, y) পেয়ার আছে যাতে $y \subseteq x$ হয় তার সংখ্যাকে বুঝায়, তাহলে কমপ্লেক্সিটি হবে $O(T(n))$। এমন কয়টা পেয়ার আছে তা হিসাব করার জন্য আমরা x আর y এর একটা একটা করে বিট বসানোর চেষ্টা করবোঃ

x_{n-1}	x_{n-2}	$\dots$	x_i	$\dots$	x_1	x_0
y_{n-1}	y_{n-2}	$\dots$	y_i	$\dots$	y_1	y_0

প্রতিটা i এর জন্য x এর i-তম বিট x_i এবং y এর i-তম বিট y_i হলে, যদি $y \subseteq x$ হতে হয়, তাহলে (x_i, y_i) এর জন্য ঠিক ৩টি অপশন আছে – $(0,0), (1,0), (1,1)$। যেহেতু প্রতিটা i এর জন্য ৩টি অপশন, আর এমন সিদ্ধান্ত আমাদের n বার নিতে হবে তাই আমরা বলতে পারি $T(n) = 3^n$।

9.3 $O(n2^n)$ ডিপি সলিউশন

আমরা চাইলে একটু অন্যভাবে রিকার্সিভ উপায়ে একটা মাস্ক mask-এর সব সাবমাস্কের জেনারেট করতে পারি। আমরা সাবমাস্কের বিটগুলো একে একে ঠিক করবো (ধরো বাম থেকে ডানে), এর জন্য আমাদের ব্যাক্ট্র্যাকিং ফাংশনে ২টি জিনিস থাকতে হবে একটা হলো ইনডেক্স i, যার মানে আমরা $(n, i]$[২] বিটগুলো ফিক্স করে ফেলেছি, এখন $i - 1$ তম বিটটি বাছাই করবো। আরেকটা আর্গুমেন্ট হবে একটা বিটমাস্ক যেটার $(n, i]$ বিটগুলো হবে বাছাইকৃত বিটগুলোর সমান, আর $(i, 0]$ বিটগুলো হবে mask বিট গুলোর সমান। যখন আমরা submask-এর $(i-1)$-তম বিট $\texttt{submask}_{\texttt{i-1}}$ কি হবে তা ঠিক করতে যাবো তখন আমাদের ২টা কেইস থাকবে (mask এর t-তম বিটকে $\texttt{mask}_{\texttt{t}}$ দিয়ে প্রকাশ করছি আমরা):

$\texttt{mask}_{\texttt{i-1}} = 0$ এক্ষেত্রে আমাদের আর কোন অপশন নেই, $\texttt{submask}_{\texttt{i-1}}$-ও 0 হতে হবে।
$\texttt{mask}_{\texttt{i-1}} = 1$ এক্ষেত্রে আমাদের ২টি অপশন আছে – $\texttt{submask}_{\texttt{i-1}}$ 0 বা 1 ২টিই হতে পারে।

$i = 0$ হয়ে গেলে বুঝবো submask এর সব বিট ফিক্স করা হয়ে গিয়েছে। নিচে এই ব্যাক্ট্র্যাকিং-এর C++ কোড দেওয়া হলোঃ

```
int n;
vector<int> submasks;
void backtrack(int i, int mask) {
  if(i == 0) {
    // everything is fixed, mask is a submask of the initial mask
    submasks.push_back(mask);
  } else {
    if(mask >> (i+1) & 1) { // i-th bit of mask is on
      backtrack(i+1, mask); // i-th bit of submask is also on
      backtrace(i+1, mask ^ (1 << (i-1))); // i-th bit of submask if off
    } else {
      backtrack(i+1, mask); // nothing to do
```

[২]এই আলোচনায় আমরা ইন্টার্ভাল নোটেশনকে একটু ভিন্নভাবে (রিভার্স ইন্টারভাল বলা যায়) ব্যবহার করছি... যেমন, $(n, i]$ বলতে $n-1, n-2, \dots, i+1, i$ এই ইন্টিজার গুলোকে বুঝানো হচ্ছে। আসলে বিটগুলো বাম থেকে ডানে বড় থেকে ছোট অর্ডারে নাম্বারিং করা বলে এভাবে লিখলে সুবিধা।

```
    }
  }
}
...
submasks.clear();
backtrack(n, some_mask);
// submasks will contain all the submasks of some_mask
```

এখান থেকে আশা করি বুঝতে পারছো একটা ডিপি সলিউশন বানানো সম্ভব। ব্যাক্ট্র্যাকিং-এর ফাংশনে যেই আর্গুমেন্টগুলো ব্যবহার করেছি সেগুলোই হবে আমাদের ডিপি স্টেট। $\mathrm{dp}[i, \mathtt{mask}]$ এর সংজ্ঞা হলো, এমন সব মাস্কের যোগফল, যেগুলোর $(n, i]$ বিটগুলো ফিক্স করা হয়ে গিয়েছে, অর্থাৎ হুবুহু mask এর $(n, i]$ তম বিট গুলোর সমান, এবং $(i, 0]$ বিটগুলো mask এর $(i, 0]$ তম বিট গুলোর সাবমাস্ক। $i > 0$ এর ক্ষেত্রে ডিপির ফর্মুলা হবেঃ

$$\mathrm{dp}[i, \mathtt{mask}] = \begin{cases} \mathrm{dp}[i-1, \mathtt{mask}] & \text{if } \mathtt{mask_{i-1}} = 1 \\ \mathrm{dp}[i-1, \mathtt{mask}] + \mathrm{dp}[i-1, \mathtt{mask} - 2^{i-1}] & \text{if } \mathtt{mask_{i-1}} = 0 \end{cases}$$

, আর বেইস কেইস হবে $i = 0$ হলেঃ

$$\mathrm{dp}[0, \mathtt{mask}] = f[\mathtt{mask}]$$

সবার শেষে $\hat{f}[\mathtt{mask}] = \mathrm{dp}[n, \mathtt{mask}]$ হবে। নিচে C++-এ এর রিকার্সিভ ইমপ্লিমেন্টেশন দেওয়া হলোঃ

```
int mem[n+1][1 << n];
int dp(int i, int mask) {
  int& ret = mem[i][mask];
  if(ret != -1) return ret;
  if(i == 0) return ret = f[mask];
  if(mask >> (i-1) & 1) {
    ret = dp(i-1, mask) + dp(i-1, mask - (1 << (i-1)));
  } else {
    ret = dp(i-1, mask);
  }
  return ret;
}
...
// initiallize mem[][] with -1
for(int mask = 0; mask < (1 << n); ++mask) {
  fhat[mask] = dp(n, mask);
}
```

বটম আপ ইমপ্লিমেন্টেশনকে অপটিমাইজ করে $O(2^n)$ মেমোরিতেই $\hat{f}$ ক্যাল্কুলেট করা সম্ভব।

```
for(int mask = 0; mask < (1 << n); ++mask) {
```

```
  dp[0][mask] = f[mask];
}
for(int i = 1; i <= n; ++i) {
  for(int mask = 0; mask < (1 << n); ++mask) {
    if(mask >> (i-1) & 1) {
      dp[i][mask] = dp[i-1][mask] + dp[i-1][mask - (1 << (i-1))];
    } else {
      dp[i][mask] = dp[i-1][mask];
    }
  }
}
// fhat = dp[n]
```

যেহেতু `dp[i][...]` ক্যাল্কুলেট করার জন্য শুধু `dp[i-1][...]` প্রয়োজন হচ্ছে, তাই আমরা `dp[2][1 << n]` 2D অ্যারে ব্যবহার করেই ইমপ্লিমেন্ট করতে পারিঃ

```
for(int mask = 0; mask < (1 << n); ++mask) {
  dp[0][mask] = f[mask];
}
for(int i = 1; i <= n; ++i) {
  for(int mask = 0; mask < (1 << n); ++mask) {
    if(mask >> (i-1) & 1) {
      dp[i & 1][mask] = dp[~i & 1][mask] + dp[~i & 1][mask - (1 << (i-1))];
    } else {
      dp[i & 1][mask] = dp[~i & 1][mask];
    }
  }
}
// fhat = dp[n & 1]
```

9.4 হাইপারকিউব এবং প্রিফিক্স সাম

ধরো আমাদেরকে একটা অ্যারে A (0-indexed) দেওয়া আছে, এবং বলা হলো A এর প্রিফিক্স সাম অ্যারে ক্যাল্কুলেট করো। অর্থাৎ এমন একটি অ্যারে P ক্যাল্কুলেট করো যাতে $P[i] = \sum_{j=0}^{i} A_j$ হয়। কিভাবে করি আমরা? $P[0] = A[0]$ সেট করে বাকি $P[i]$ গুলো ক্যাল্কুলেট করার জন্য $P[i] = P[i-1]+A[i]$ এই রিকার্শনটি ব্যবহার করি।

```
P[0] = A[0];
for(int i = 0; i < n; ++i)
  P[i] = P[i-1] + A[i];
```

আলাদা একটি অ্যারে P না বানিয়ে A-তেই যদি আমরা প্রিফিক্স সাম স্টোর করতে চাই তাহলে কোডটা

হবে এমন:

```
for(int i = 0; i < n; ++i) {
  if(i != 0) A[i] += A[i-1];
}
```

এবার ধরো তোমাকে একটি $n \times m$ সাইজের গ্রিড G দেওয়া আছে (আবারও 0-indexed, অর্থাৎ $G[0 \ldots (n-1), 0 \ldots (m-1)]$), আর বলা হলো G এর প্রিফিক্স সাম অ্যারে P ক্যাল্কুলেট করো, যেখানে $P[x,y] = \sum_{i=0}^{x} \sum_{j=0}^{y} G[i,j]$ হবে। এই ক্ষেত্রে এমরা সাধারণত প্রিন্সিপাল অফ ইনক্লুশন-এক্সক্লুশন দিয়ে P ক্যাল্কুলেট করে থাকি, যেমন P এর প্রথম রো এবং প্রথম কলামের ভ্যালুগুলো 1D প্রিফিক্স সাম ব্যবহার করে ক্যাল্কুলেট করার পর $P[x > 0, y > 0]$ এর ভ্যালুগুলো ক্যাল্কুলেট করতে আমরা এই রিকার্শনটি ব্যবহার করা হয়:

$$P[x,y] = P[x-1,y] + P[x,y-1] - P[x-1,y-1] + G[x,y]$$

এটা ছাড়াও আমরা আরেকটি উপায় P বের করতে পারি। এর জন্য আমাদের আরও কয়েকটি 2D অ্যারে ডিফাইন করতে হবে:

→ G এর রো গুলোর প্রিফিক্স সামের গ্রিড R, অর্থাৎ, $R[x,y] = \sum_{i=0}^{y} G[x,i]$।

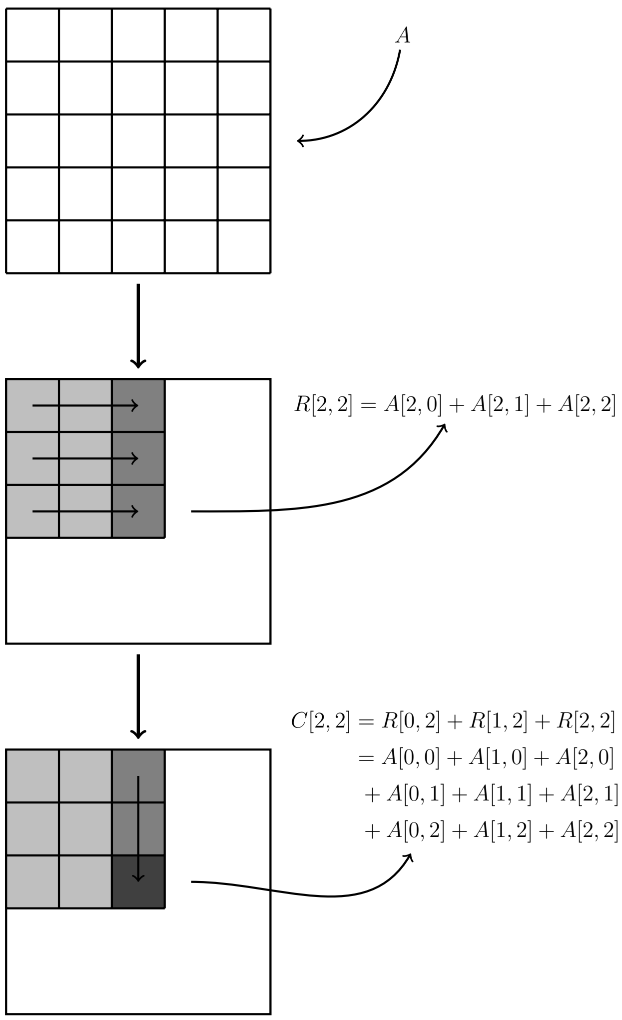
→ R এর কলামগুলোর প্রিফিক্স সামের গ্রিড C, অর্থাৎ, $C[x,y] = \sum_{i=0}^{x} R[i,y]$।

একটু খেয়াল করলে বুঝবে C গ্রিডটিই হলো G এর প্রিফিক্স সাম গ্রিড, অর্থাৎ, $C = P$। C++ ইমপ্লিমেন্টেশন:

```
// i = row, j = column
for(int i = 0; i < n; ++i) {
  R[i][0] = A[i][0];
  for(int j = 1; j < m; ++j) {
    R[i][j] = R[i][j-1] + A[i][j];
  }
}
for(int j = 0; j < m; ++j) {
  C[0][j] = R[0][j];
  for(int i = 1; i < n; ++i) {
    C[i][j] = C[i-1][j] + R[i][j];
  }
}
```

দ্বিতীয় 2D for-লুপ ২টিকে আমরা সোয়াপ করে দিতে পারি:

```
for(int j = 0; j < m; ++j)
  C[0][j] = R[0][j];
for(int i = 1; i < n; ++i) {
  for(int j = 0; j < m; ++j) {
```



চিত্র 9.1: $G \to R \to C$

```
    C[i][j] = C[i-1][j] + R[i][j];
  }
}
```

এমনকি আলাদা আলাদা অ্যারে R, এবং C ব্যবহার না করেই শুধু A এর উপর অপারেশনগুলো অ্যাপ্লাই করেই A তেই প্রিফিক্স সাম স্টোর করা সম্ভব:

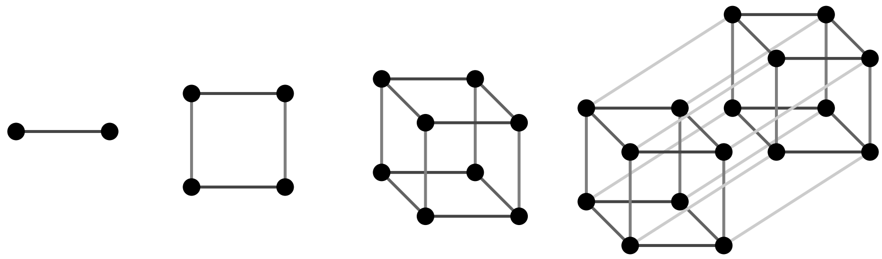
```
// first operation: A --> R
for(int i = 0; i < n; ++i) {
  for(int j = 0; j < m; ++j) {
    if(j != 0) A[i][j] += A[i][j-1];
  }
}
// second operation: R --> C
for(int i = 0; i < n; ++i) {
  for(int j = 0; j < m; ++j) {
    if(i != 0) A[i][j] += A[i-1][j];
  }
}
```

একটি প্যাটার্ন কি দেখতে পাচ্ছো? আমরা কিন্তু 3D অ্যারের জন্যও একইভাবে প্রিফিক্স সাম ক্যাল্কুলেট করতে পারবো! যেমন:

```
for(int i = 0; i < lim_x; ++i) {
  for(int j = 0; j < lim_y; ++j) {
    for(int k = 0; k < lim_z; ++k) {
      if(i != 0) a[i][j][k] += a[i-1][j][k];
    }
  }
}
for(int i = 0; i < lim_x; ++i) {
  for(int j = 0; j < lim_y; ++j) {
    for(int k = 0; k < lim_z; ++k) {
      if(j != 0) a[i][j][k] += a[i][j-1][k];
    }
  }
}
for(int i = 0; i < lim_x; ++i) {
  for(int j = 0; j < lim_y; ++j) {
    for(int k = 0; k < lim_z; ++k) {
      if(k != 0) a[i][j][k] += a[i][j][k-1];
    }
  }
}
```

9.4.1 প্রিফিক্স সামের সাথে সাম ওভার সাবসেটের সম্পর্ক

হাইপারকিউব হলো n-ডাইমেনশনের একটি কিউব যার 2^n-টি শীর্ষ (vertex) $\{0,1\}^n$ বিন্দুগুলোতে অবস্থিত। যেমন নিচের চিত্রে $n = 1, 2, 3, 4$ এর উদাহরণ দেখানো হলো:



যেমন, আমাদের আগের আলোচনায় 1D অ্যারের ক্ষেত্রে $n = 2$ কিংবা 2D অ্যারের ক্ষেত্রে $(n, m) = (2, 2)$, এমনকি 3D অ্যারের ক্ষেত্রে $(\mathtt{lim_x}, \mathtt{lim_y}, \mathtt{lim_z}) = (2, 2, 2)$ হলে সেগুলো একেকটি হাইপারকিউব হয়ে যেত।

প্রতিটা n লেংথের বিটমাস্ককে আমরা n-ডাইমেনশনের হাইপারকিউবের একটি ''সেল'' বলতে পারি। আর এভাবে যদি একটি বিটমাস্ককে একটি সেল হিসেবে ডিফাইন করি, তাহলে খেয়াল করবে, সেই বিটমাস্কের প্রতিটি সাবসেট হলো হাইপারকিউবের মধ্যে $(0, 0, \ldots, 0)$ পয়েন্ট থেকে ওই বিটমাস্কের পয়েন্ট পর্যন্ত সব সেল বা vertex এর একটি। অর্থাৎ, একটি হাইপারকিউব f এর প্রিফিক্স সামের হাইপারকিউব হলো $\hat{f} = \zeta(f)$। 2D বা 3D এর মতো একইভাবে n-ডাইমেনশনের জন্যও আমরা প্রিফিক্স সাম ক্যাল্কুলেট করতে পারি। নিচে কমেন্ট সহ এর C++ কোড দেওয়া হলো:

```
for(int i = 0; i < n; ++i) { // iterate on the dimensions
  for(int mask = 0; mask < (1 << n); ++mask) { // iterate over all the
      points
    if(mask >> i & 1) {
      /* similar to g[i][j][k] += g[i-1][j][k], g[i][j][k] += g[i][j-1][k],
       * and g[i][j][k] += g[[i][j][k-1] */
      f[mask] += f[mask - (1 << i)];
    }
  }
}
// we have applied zeta transformation on f; now fhat[x] = f[x]
```

9.5 Möbius Inversion

এককথায়, $\zeta^{-1}(\hat{f})$ বা $\mu(\hat{f}) = f$, অর্থাৎ, আমাদেরকে $\hat{f}$ দেওয়া থাকলে এমন একটা f ক্যাল্কুলেট করতে হবে যেন $\zeta(f) = g$ হয়। $\hat{f} \mapsto f$ এই ট্রান্সফরমেশনকেই Möbious inversion বলা হয়।

প্রথম প্রশ্ন হলো আমরা কি আসলেই এমন ফাংশন বের করতে পারবো কিনা, বা পারলেও সবসময়ই পারবো কিনা। এটা বুঝার জন্য একটা উপায় হলো লিনিয়ার অ্যালজেব্রার ভাষায় চিন্তা করা। ফাংশন f কে আমরা একটি $2^n \times 1$ সাইজের কলাম ভেক্টর $\vec{f}$ দিয়ে প্রকাশ করতে পারি, যেখানে

$$\vec{f} = \begin{pmatrix} f[0] \\ f[1] \\ \vdots \\ f[2^n - 1] \end{pmatrix}$$

একইভাবে $\hat{f}$ কে কলাম ভেক্টর $\vec{\hat{f}}$ দিয়ে প্রকাশ করা যাবে। এবার খেয়াল করলে দেখবে, $\vec{f} \mapsto \vec{\hat{f}}$ একটি লিনিয়ার ট্রান্সফরমেশন। এর ট্রান্সফরমেশন ম্যাট্রিক্স ζ কে আমরা এভাবে সংজ্ঞায়িত করতে পারি: $\zeta[x, y] = 1$ যদি $y \subseteq x$ হয়, নাহলে $\zeta[x, y] = 0$। তাহলে,

$$\vec{\hat{f}} = \zeta \vec{f}$$

এই ট্রান্সফরমেশন ম্যাট্রিক্স ζ এর ইনভার্স ম্যাট্রিক্স $\zeta^{-1} = \mu$ বিদ্যমান (invertible), কারণ খেয়াল করলে দেখবে ζ একটি lower triangular ম্যাট্রিক্স যার diagonal-এ সব 1। সুতরাং আমরা বলতে পারি, প্রত্যেক ফাংশন $\hat{f}$ এরই Möbius inversion আছে।

$$\vec{f} = \zeta^{-1} \vec{\hat{f}} = \mu \vec{\hat{f}}$$

এটা ছাড়াও অন্য একভাবে zeta transform এর ইনভার্স কেন আছে তা ব্যাখ্যা করতে পারি। আগের সেকশনে আমরা zeta transform এর কোড দেখেছি, যেটা f-এর উপর কিছু ম্যাথম্যাটিক্যাল অপারেশন অ্যাপ্লাই করে সেটাকে $\hat{f}$ তে রূপান্তর করেছে। n-বিটের বিটমাস্কের ফাংশন f এর zeta transform কে আমরা $n \cdot 2^{n-1}$ টা বিটমাস্কের পেয়ার দিয়ে প্রকাশ করতে পারি:

$$\begin{gathered} (x_1, y_1) \\ (x_2, y_2) \\ \vdots \\ (x_{n2^{n-1}}, y_{n2^{n-1}}) \end{gathered}$$

যার মানে হলো নিচের অপারেশন গুলো ক্রমান্বয়ে f অ্যারের উপর অ্যাপ্লাই করা:

$$\begin{gathered} \mathtt{f}[\mathtt{x}_1] := \mathtt{f}[\mathtt{x}_1] + \mathtt{f}[\mathtt{y}_1] \\ \mathtt{f}[\mathtt{x}_2] := \mathtt{f}[\mathtt{x}_2] + \mathtt{f}[\mathtt{y}_2] \\ \vdots \\ \mathtt{f}[\mathtt{x}_{\mathtt{n2^{n-1}}}] := \mathtt{f}[\mathtt{x}_{\mathtt{n2^{n-1}}}] + \mathtt{f}[\mathtt{y}_{\mathtt{n2^{n-1}}}] \end{gathered}$$

যেহেতু যোগ $(+)$ অপারেটরের "ইনভার্স" অপারেটর (-) আছে, এবং এই লিস্টে সব i এর জন্যই $x_i \neq y_i$, তাই আমরা এই লিস্টের ইনভার্স লিস্ট লিখতে পারবো:

$$\begin{gathered}
\mathtt{f}[\mathtt{x}_{\mathtt{n}2^{\mathtt{n}-1}}] := \mathtt{f}[\mathtt{x}_{\mathtt{n}2^{\mathtt{n}-1}}] - \mathtt{f}[\mathtt{y}_{\mathtt{n}2^{\mathtt{n}-1}}] \\
\mathtt{f}[\mathtt{x}_{\mathtt{n}2^{\mathtt{n}-1}-1}] := \mathtt{f}[\mathtt{x}_{\mathtt{n}2^{\mathtt{n}-1}-1}] - \mathtt{f}[\mathtt{y}_{\mathtt{n}2^{\mathtt{n}-1}-1}] \\
\vdots \\
\mathtt{f}[\mathtt{x}_1] := \mathtt{f}[\mathtt{x}_1] - \mathtt{f}[\mathtt{y}_1]
\end{gathered}$$

এখান থেকে আশা করি বুঝতে পারছো zeta transforma এর কোডের লুপ গুলোকে উল্টা অর্ডারে লিখলেই আমরা আবার f পেয়ে যাবো!

```
for(int i = n-1; i >= 0; --i) {
  for(int mask = (1 << n) - 1; mask >= 0; --mask) {
    if(mask >> i & 1) {
      fhat[mask] -= fhat[mask - (1 << i)];
    }
  }
}
// we've inverted fhat; now f[x] = fhat[x]
```

9.6 সেট দিয়ে Zeta Transform-এর ব্যাখ্যা এবং Möbius Inversion এর ফর্মুলা

Zeta transform-এর সাবসেটকে পরিবর্তন করে সুপারসেট করে দিয়ে আমরা $\zeta_{\supseteq}$-transform ডিফাইন করলাম ধরো। অর্থাৎ, $\hat{f} = \zeta_{\supseteq}(f)$, যেখানে

$$\hat{f}[x] = \sum_{y \supseteq x} f[y]$$

আমরা ζ-transform এবং ζ-inversion (Möbius inversion) নিয়ে কথা না বলে $\zeta_{\supseteq}$-transform এবং তার সংশ্লিষ্ট ইনভার্শন নিয়ে আলোচনা করবো এই সেকশনে, কারণ ২টা জিনিসই একটা থেকে আরেকটায় কনভার্ট করা যায়।

ধরো তোমার কাছে n-টা সেট $A_0, A_1, \ldots, A_{n-1}$ আছে। আমরা বর্ণনার সুবিধার্থে একটি নোটেশন ডিফাইন করে নেওয়া যাক – N এর সব সাবসেট J এর জন্য A_J হলো J-তে যেই ইনডেক্স গুলো আছে, সেই সেট গুলোর ইন্টারসেকশন। অর্থাৎ,

$$A_J = \bigcap_{j \in J} A_j$$

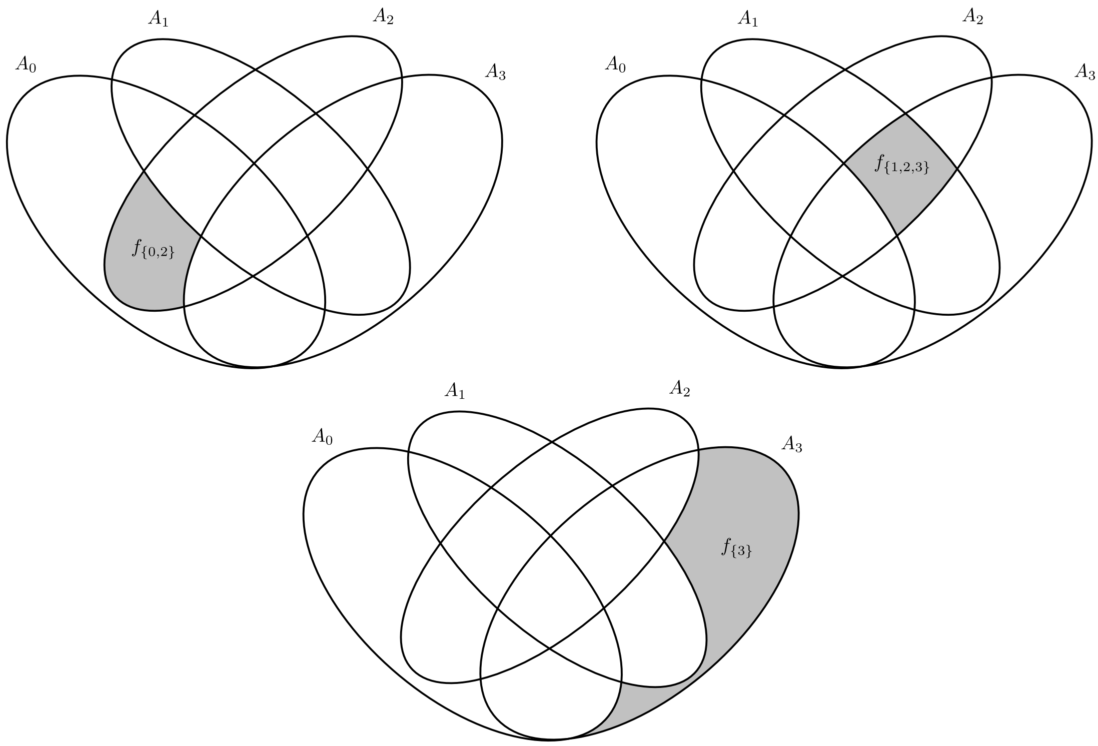
বিশেষ করে $A_\varnothing = \varnothing$। এবার আমরা A_i গুলোকে এমন ভাবে ডিফাইন করবো যেন সেগুলো সব $J \subseteq N$ এর জন্য নিচের শর্তটি পূরণ করে:

ধরো t হলো এমন সব উপাদানের সংখ্যা, যেগুলো এমন সব সেট A_j এর প্রত্যেকটিতেই আছে যেখানে $j \in J$ কিন্তু এমন সব সেট A_j এর একটিতেও নেই যেখানে $j \neq J$। অনেকটা এভাবে বলা যায়: ''যেসব উপাদান বিশেষভাবে শুধুমাত্র A_j সেট গুলোতেই আছে, যেখানে $j \in J$''। সেট থিওরির ভাষায় বললে হয়:

$$t = \left| \left(\bigcap_{j \in J} A_j \right) \setminus \left(\bigcup_{j \neq J} A_j \right) \right|$$

t এর মান $f[J]$ এর সমান হতে হবে।

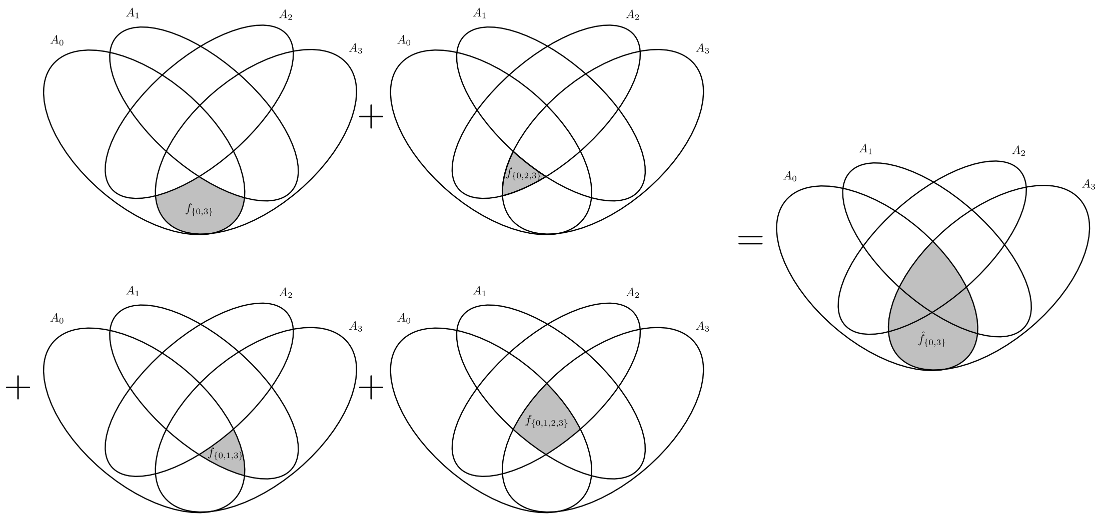
চিত্র 9.2-এ $f_{\{0,2\}}$, $f_{\{1,2,3\}}$ এবং $f_{\{3\}}$ এর এলাকাকে ছায়া দিয়ে লেবেল করে দেখানো হয়েছে। একটি উল্লেখযোগ্য বিষয় হলো সব $f[J]$ এর ক্ষেত্রগুলো কিন্তু বিচ্ছিন্ন।



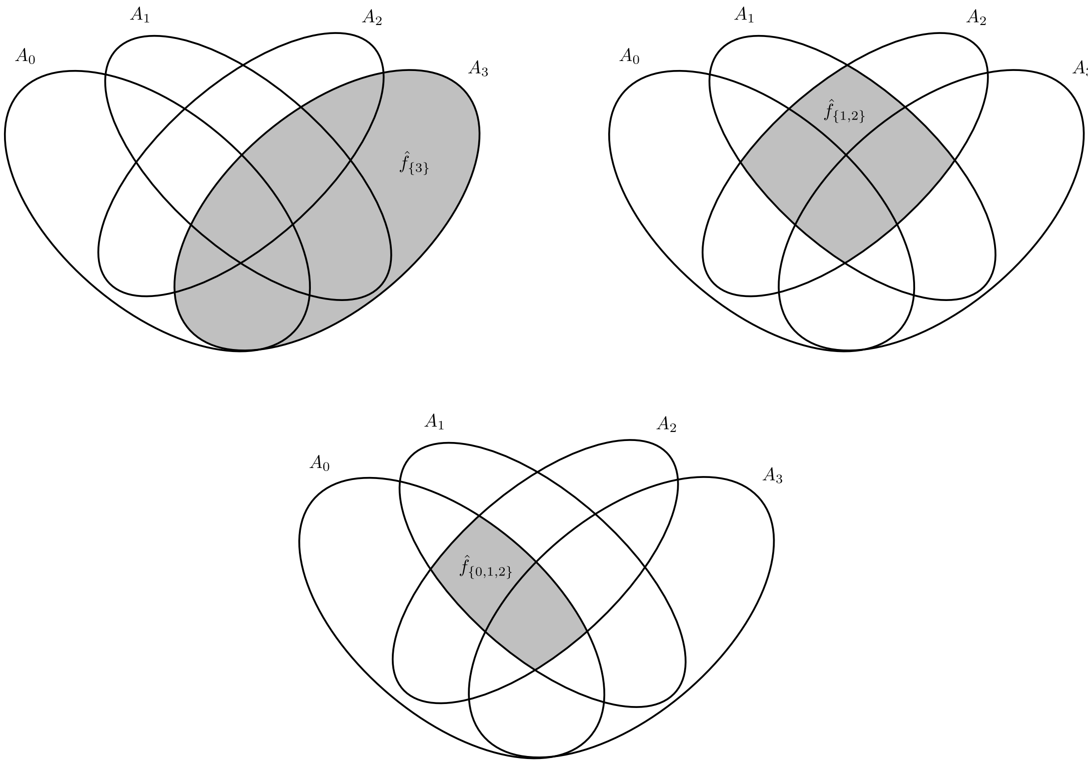
চিত্র 9.2: A_0, A_1, A_2, A_3 এর সংজ্ঞা অনুযায়ী কয়েকটি J এর জন্য $f[J]$ এর উদাহরণ।

9.6.1 Zeta Transform

ধরো আমরা $\hat{f}_{\{0,3\}}$ এর মান বের করতে চাচ্ছি। তাহলে যেটা হবে তা চিত্র 9.3-তে দেখানো হলো: $\hat{f}_{\{0,3\}} = A_0 \cap A_3$ পাওয়া যায়! আর কয়েকটা এভাবে একে দেখলে বুঝতে পারবে সব J এর জন্য $\hat{f}[J] = \left| \bigcap_{j \in J} A_j \right|$ হয়। চিত্র 9.4-এ এমন আর কয়েকটি $\hat{f}[J]$ এর উদাহরণ দেখানো হয়েছে। সুতরাং,



চিত্র 9.3: $\hat{f}_{\{0,3\}} = f_{\{0,3\}} + f_{\{0,1,3\}} + f_{\{0,2,3\}} + f_{\{0,1,2,3\}}$



চিত্র 9.4: কয়েকটি $\hat{f}[J]$ এর উদাহরণ

$\zeta_{\supseteq}$-transform হলো:

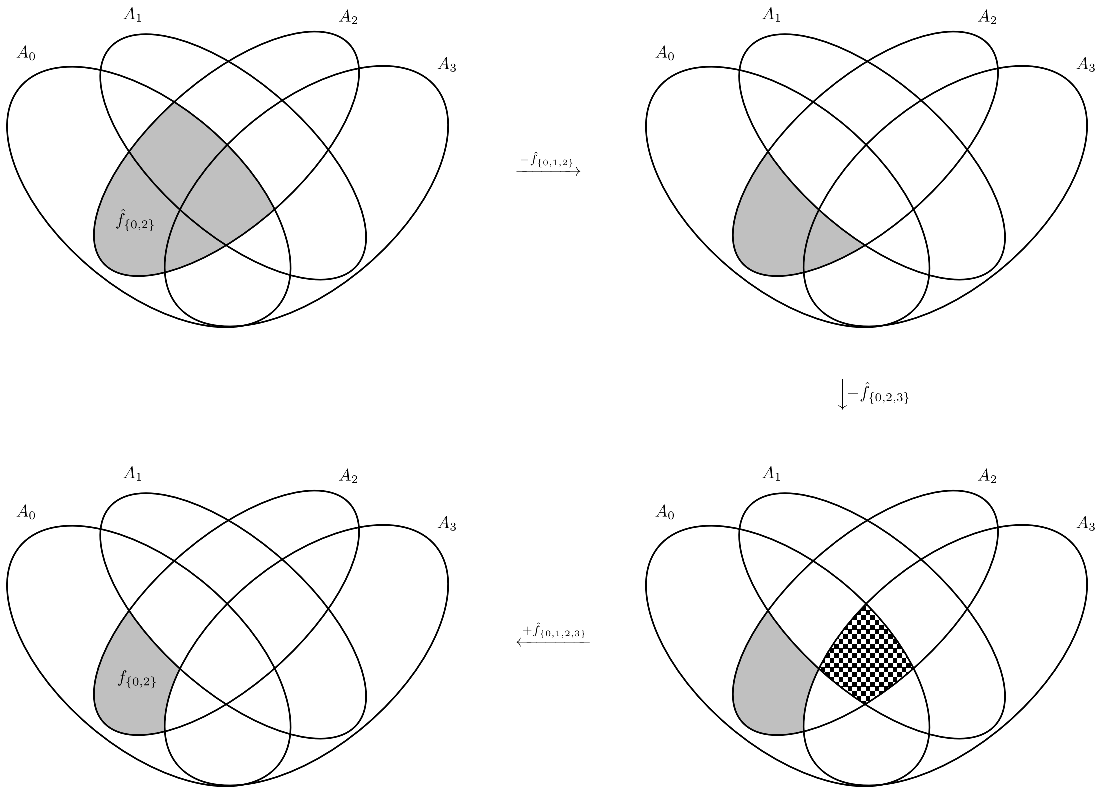
$$f[J] = \left|\left(\bigcap_{j\in J} A_j\right) \setminus \left(\bigcup_{j\neq J} A_j\right)\right|$$
$$\Big\downarrow \zeta_{\supseteq}$$
$$\hat{f}[J] = \left|\bigcap_{j\in J} A_j\right|$$

9.6.2 Möbius Inversion

এখন যেহেতু আমরা বুঝতে পারছি zeta transform এর মাধ্যমে প্রতি J এর জন্য $f[J]$ venn diagram-এর কোন অংশ হতে ট্রান্সফর্ম হয়ে কোন অংশে যাচ্ছে, তাই এটাকে কাজে লাগিয়ে আমরা $\hat{f}$ এর অংশগুলো থেকে f এর অংশ গুলো বের করার চেষ্টা করবো। মূলত এখন সমস্যাটা দাঁড়ালো এই:

> যদি প্রতিটি $J \subseteq N$ এর জন্য $\left|\bigcap_{j\in J} A_j\right|$ দেওয়া থাকে, তাহলে প্রতিটি $J \subseteq N$ এর জন্য $\left|\left(\bigcap_{j\in J} A_j\right) \setminus \left(\bigcup_{j\neq J} A_j\right)\right|$ ক্যাল্কুলেট করতে হবে।

এটা খুব সহজেই প্রিন্সিপাল অফ ইনক্লুশন-এক্সক্লুশন দিয়ে করা যায়। যেমন, যদি $n = 4$ এর ক্ষেত্রে $f_{\{0,2\}}$ ক্যাল্কুলেট করতে চাই তাহলে প্রথমেই $\hat{f}_{\{0,2\}}$ কে আমাদের যোগফলে যোগ করবো। এরপর $\hat{f}_{\{0,2\}}$ থেকে $f_{\{0,2,3\}}$, $f_{\{0,1,2\}}$, $f_{\{0,1,2,3\}}$ গুলো বাদ দেওয়ার জন্য $\hat{f}_{\{0,1,2\}}$ এবং $\hat{f}_{\{0,2,3\}}$ বিয়োগ দিবো। কিন্তু এরপর আবার $f_{\{0,1,2,3\}}$ একবার বেশি বিয়োগ হয়ে গিয়ে থাকবে। সেটা ঠিক করার জন্য আবার $\hat{f}_{\{0,1,2,3\}}$ যোগ করতে হবে। সব মিলিয়ে হবে:



$$f_{\{0,2\}} = \hat{f}_{\{0,2\}} - \hat{f}_{\{0,1,2\}} - \hat{f}_{\{0,2,3\}} + \hat{f}_{\{0,1,2,3\}}$$

সাধারণ ভাবে বললে:

$$f[X] = \sum_{Y \supseteq X} (-1)^{|Y \setminus X|} \hat{f}[Y]$$

এটাই $\mu_{\supseteq}$-inversion ফর্মুলা! অনুরূপভাবে $\mu_{\subseteq}$ বা ক্লাসিক্যাল Möbius inversion ফর্মুলা হবে:

$$f[X] = \sum_{Y \subseteq X} (-1)^{|X \setminus Y|} \hat{f}[Y]$$

[1]

9.7 আরও কিছু উদাহরণ

উদাহরণ 9.2 (Or Plus Max)**.** তোমাকে একটা 2^n লেংথের ইন্টিজার সিকুয়েন্স $A_0, A_1, \ldots, A_{2^n-1}$ দেওয়া আছে। প্রতিটা ইন্টিজার $k\,(1 \le k \le 2^n - 1)$ এর জন্য ক্যাল্কুলেট করতে হবে: $A_i + A_j$ এর ম্যাক্সিমাম ভ্যালু, যেখানে $0 \le i < j \le 2^n - 1$ এবং $(i \mid j) \le k$। এখানে | দিয়ে বিটওয়াইজ অর বুঝানো হয়েছে। $1 \le n \le 18$, $1 \le A_i \le 10^9$।

উদাহরণ 9.3. শুরুতে তোমার কাছে একটি n লেংথের অ্যারে $A = [0, 0, \ldots, 0]$ আছে। তুমি এর উপর কিছু অপারেশন অ্যাপ্লাই করতে পারো। প্রতিটা অপারেশন হলো: প্রথমে A এর একটি সাবঅ্যারে নির্বাচন করবা, তারপর সেই সাবঅ্যারেতে তোমার পছন্দের একটি ধনাত্মক পূর্ণসংখ্যা যোগ করে দিবা। তোমাকে বের করতে হবে, মিনিমাম কয়টি অপারেশন অ্যাপ্লাই করে A অ্যারেটিকে তুমি আরেকটি প্রদত্ত অ্যারে B এর সমান বানাতে পারবে। $n \le 20$, $1 \le B_i \le 10^9$।

উদাহরণ 9.4 (Innopolis Open 2019 - Cake Testing)**.** কারিনা কেক খুব পছন্দ করে। তার শহরে n টি কেকের দোকান আছে। সে মোট $2^n - 1$ দিন ঘর থেকে বের হবে। দিন গুলো $1, 2, \ldots, 2^n - 1$ নিয়ে নাম্বারিং করা, এবং দোকান গুলো $0, 1, \ldots, n-1$ দিয়ে। i তম দিনে বের হলে সে j-তম দোকানে যাবে যদি i-এর বাইনারিতে j-তম বিট অন থাকে। কোন দোকানে গেলে সে দোকানে থাকা সব টাইপের কেক একটি করে খায়। অবশ্য একই টাইপের কেক একাধিক দোকানে থাকতে পারে। দিন শেষে সে নোট করে, সারাদিনে সে কয়টা ভিন্ন ভিন্ন টাইপের কেক খেয়েছে (একই টাইপের কেক একাধিক দোকানে খেয়ে থাকলেও একবারই হিসাবে যোগ হবে)। i-তম দিনের জন্য এই সংখ্যাটি হলো a_i। তোমাকে যাচাই করতে হবে a_i এর ভ্যালু গুলো সামঞ্জস্যপূর্ণ কিনা। অর্থাৎ, আমরা যদি i-তম দোকানে পাওয়া যায় এমন কেকের টাইপগুলোর সেটকে S_i দিয়ে প্রকাশ করি, তাহলে তোমাকে বের করতে হবে এমন কোনো সেটের সিকুয়েন্স $[S_1, S_2, \ldots, S_{2^n-1}]$ আছে কিনা যাতে সেটা $a_1, a_2, \ldots, a_{2^n-1}$ ভ্যালুগুলো মেনে চলে – প্রতি i এর জন্য যাতে $\bigcup_{j,\, i_j=1} S_j = a_i$ হয়। $n \le 19$, $1 \le a_i \le 1000$।

উদাহরণ 9.5 (COCI - Kosare)**.**

উদাহরণ 9.6 (Codechef - Beautiful Sandwich)**.**

উদাহরণ 9.7 (Omkar and Pies)**.**

উদাহরণ 9.8 (USACO 2012 - Skyscraper)**.**

উদাহরণ 9.9 (Pepsi Cola)**.**

উদাহরণ 9.10 (Codeforce - AND Graph)**.**

9.8 অনুশীলনী

অনুশীলনী 9.1 (Codechef - Prefix And)**.**

খন্ড I

বাছাইকৃত কিছু সমস্যার হিন্ট সমূহ

5.3 প্রথম অভজারভেশন হলো, দুটো মাস i এবং j তে যদি তুমি ২টি অফার চালু করো (যেখানে $i < j$), তাহলে i আর j এর মধ্যে এমন কোন মাস k থাকতে পারবে না যেটাতে তুমি কোন অফার চালু করোনি (অর্থাৎ, $i < k < j$ হতে পারবে না)। সুতরাং যেই মাসগুলোতে তুমি চালু করবা সেগুলো একটা consecutive রেঞ্জ হবে। ধরো তুমি একটা সিকুয়েন্স ঠিক করেছ $s_0, s_1, \ldots, s_{m-1}\,(m \leq n)$, যার মানে হলো প্রথম মাসে তুমি s_{m-1}-তম অফারটি চালু করবে, দ্বিতীয় মাসে s_{m-2}-তম... m-তম মাসে s_0-তম অফারটি নিয়েছ, তাহলে এই সিকুয়েন্সের কস্ট হবেঃ

$$\sum_{i=0}^{m-1} a_{s_i} - b_{s_i} \cdot \min(k_{s_i}, i)$$

এইরকম কস্ট ফাংশনে এক্সচেঞ্জ আর্গুমেন্ট অ্যাপ্লাই করতে ঝামেলা হবে, কারণ একটা min চলে এসেছে। সেজন্য আমরা এক কাজ করতে পারি, যেগুলোর জন্য $k_{s_i} < i$ হবে (অর্থাৎ, তুমি যখন গাড়ি নিয়ে পালিয়ে যাবে, তার আগেই এসব অফারের মেয়াদ শেষ হয়ে যাবে), সেগুলো পুরাপুরি আলাদা করে ফেলা। এদেরকে প্রথম টাইপের অফার বলবো এখন থেকে, আর বাকিগুলোকে দ্বিতীয় টাইপের অফার। এখন আরেকটা অভজারভেশন হলো, আমরা যদি প্রথম টাইপের অফার গুলো সব আগেভাগে নিয়ে ফেলি তাহলে আমাদের কোন লস হবে না। আরেকটা ক্রুশাল বিষয় হলো, এখন আমরা ধরে নিতে পারি প্রথম টাইপের অফার গুলা আমাদের মোট যোগফলে $a_i - b_i \cdot k_i$ কন্ট্রিবিউট করবে, আর এই জিনিসটা পুরাপুরি ইন্ডিপেন্ডেন্ট – এই অফার কোন মাসে নেয়া হচ্ছে তার উপর নির্ভর করে না। কেন? এমন কি হতে পারে না যে এই অফারটিকে যখন চালু করেছিলাম তার পরে k_i মাস পার হওয়ার আগেই তুমি গাড়ি নিয়ে পালিয়েছ? সেরকম হলে তো এই অফার আরও বেশি কন্ট্রিবিউট করতে পারতো! হতে পারে, কিন্তু যেটা খেয়াল করার বিষয় তা হলো, আমাদেরকে তো ম্যাক্সিমাম কস্ট বের করতে বলেছে। এমন যদি হয়, আমরা যেই ফিক্সড কন্ট্রিবিউশন ধরে নিয়েছি, তার কারণে আসল কস্টের চাইতে ডিপিতে কম অ্যাড হচ্ছে, তাহলে সেই সলিউশনটা অপ্টিমাল হবে না! চিন্তা করে দেখো এটা।

সুতরাং আমরা বলতে পারিঃ

$$\begin{aligned}
&\max_{s}\left(\sum_{i=0}^{m-1} a_{s_i} - b_{s_i} \cdot \min(k_{s_i}, i)\right) \\
= &\max_{p \cap q = \varnothing}\left(\sum_{i \in p} a_i - b_i \cdot k_i + \sum_{i=0}^{|q|-1} a_{q_i} - b_{q_i} \cdot i\right)
\end{aligned}$$

এখন আমাদের q এর উপাদান গুলা কিভাবে সাজাতে হবে সেটা চিন্তা করতে হবে। এই কস্ট ফাংশনে এক্সচেঞ্জ আর্গুমেন্ট অ্যাপ্লাই করলে দেখবে উপাদান গুলো b_i এর decreasing অর্ডারে সাজালে সবসময় অপ্টিমাল হবে। এরপর খালি একটা ডিপি লেখা বাকি আমাদের। শুরুতে সবকিছুকে b_i দিয়ে বড় থেকে ছোট অর্ডারে সাজানোর পর বাম থেকে ডানে যাবা, একটা উপাদানের জন্য তিনটা অপশানঃ p তে নিবা, q তে নিবা, কোনটাতেই নিবা না। এছাড়াও, q তে ইতোমধ্যে কয়টা নিয়ে ফেলেছ সেটাও স্টেটে রাখতে হবে।

তথ্যবিবরণী

[1] Andreas Björklund, Thore Husfeldt, Petteri Kaski, and Mikko Koivisto. Fourier meets möbius: Fast subset convolution. In *Proceedings of the Thirty-Ninth Annual ACM Symposium on Theory of Computing*, STOC '07, pages 67–74, New York, NY, USA, 2007. Association for Computing Machinery.