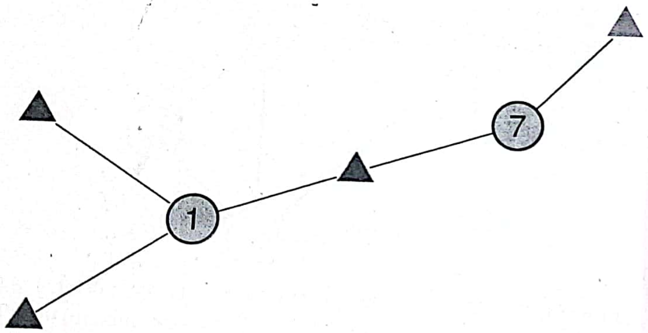
দিলেই আমাদের সব কম্পোনেন্ট বের হয়ে যাবে। আমরা প্রতিটি *unvisited* নোডের জন্য BFS বা DFS চালাব এবং আর্টিকুলেশন ব্রিজ ব্যতীত অন্য সব বাহু দিয়ে আমরা ট্রাভার্স করব।

BCC এর সঙ্গে সম্পর্কিত আরেকটি জিনিস আছে আর তা হলো ব্লক কার ভার্টেক্স ট্রি (Block cut vertex tree). প্রতিটি আনডিরেক্টেড গ্রাফকে যেমন আমরা BCC তে ভাগ করতে পারি ঠিক তেমনই সেই BCC কে আমরা একটি ট্রি আকারে সাজাতে পারি যেখানে ট্রি এর নোডগুলো হলো BCC এর কাট ভার্টেক্স (আর্টিকুলেশন নোড) সমূহ এবং ব্লক (block) বা কম্পোনেন্টগুলো। যদি কোনো একটি কাট ভার্টেক্স একটি ব্লকের অংশ হয় তাহলে তাদের মধ্যে বাহু থাকবে। তাহলে যেকোনো কানেক্টেড আনডিরেক্টেড গ্রাফ (connected undirected graph) এর জন্য আমরা একটি ট্রি পাব। যেমন চিত্র ৮.৩ এর ব্লক কাট ভার্টেক্স ট্রি হবে চিত্র ৮.৪ এর মতো। বেশ কিছু সমস্যায় আমরা দেখব যে এই ট্রি বেশ কাজে লাগে।



নকশা ৮.৪: বাইকানেক্টেড অ্যালগরিদম (Biconnected algorithm)

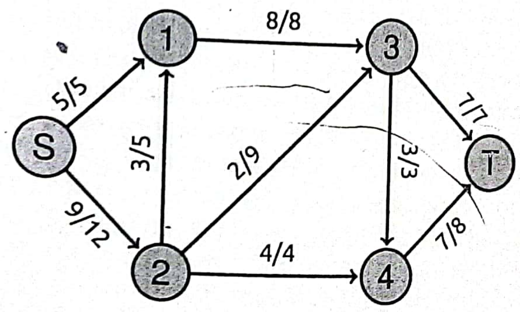
সবই বুঝলাম কিন্তু এর অ্যালগরিদম কী? আগেই বলেছি আমি নিজে এটা কখনও কোড করি নাই। হয়তো পরে কখনও এটা নিয়ে লিখবো। এই সেকশন থেকে তোমরা জেনে রাখলে BCC কী জিনিস এবং কোন সব সমস্যার ক্ষেত্রে এটা ব্যবহার হয়। যদি দরকার হয় তাহলে তোমরা ইন্টারনেট থেকে এর কোনো কোড নিয়ে ব্যবহার করতে পারো।

৮.১৩ ফ্লো (Flow) সম্পর্কিত অ্যালগরিদম

নিঃসন্দেহে ফ্লো (Flow) একটি কঠিন টপিক। এর কোড বেশ সহজ কিন্তু এটি ঠিক মতো বোঝা বা একে রপ্ত করা খুবই কঠিন। দেখা যাক তোমাদের এর মূল ধারনাটি বোঝাতে পারি কিনা।

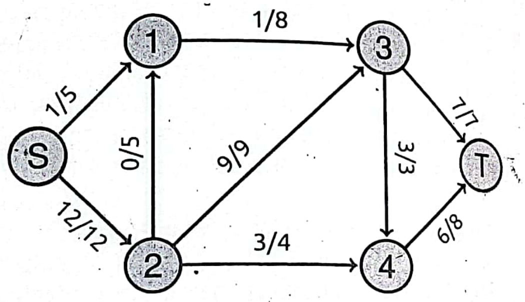
৮.১৩.১ ম্যাক্সিমাম ফ্লো (Maximum flow)

এই সমস্যায় কিছু নোড এবং কিছু ডিরেক্টেড বাহু থাকবে। নোডসমূহের মাঝে দুটি বিশেষ নোড থাকবে। একটি হলো উৎস বা সোর্স (source) যা সাধারণত আমরা S দিয়ে প্রকাশ করি আর আরেকটি হলো গন্তব্য বা সিঙ্ক (sink) যা আমরা সাধারণত T দিয়ে প্রকাশ করি। ডিরেক্টেড বাহুসমূহ এর সঙ্গে একটি সংখ্যা থাকবে, একে আমরা ওজন (weight) বলব না, একে আমরা বলব ধারনক্ষমতা বা ক্যাপাসিটি (capacity). এখন মনে কর সোর্সে অপরিসীম তরল পদার্থ আছে, আর সিঙ্কে যেকোনো সময়ে অপরিসীম তরল প্রবেশ করতে পারে। অন্যান্য যেসব বাহুর কথা বললাম তাদের একেকটি পাইপ মনে কর যাদের ক্যাপাসিটি দেওয়া আছে অর্থাৎ কোনো একক সময়ে সর্বোচ্চ কত তরল ওই পাইপ দিয়ে প্রবাহিত হতে পারে তা দেওয়া আছে। তোমাদের বলতে হবে কোনো একক সময়ে কী পরিমাণ তরল সোর্স হতে সিঙ্কে প্রবাহিত হতে পারে? তোমরা ধরে নিতে পার যে, কোনো পাইপ দিয়ে তরল যেতে কোনো সময় লাগে না তবে একক সময়ে তার ক্যাপাসিটির থেকে বেশি তরল প্রবাহিত যেন না হয়। এই সমস্যাটিই হলো ম্যাক্সিমাম ফ্লো (maximum flow) বা সংক্ষেপে ম্যাক্সফ্লো (maxflow). কিছু উদাহরণ দেখা যাক। ধরা যাক, S হতে A তে একটি বাহু আছে যার ক্যাপাসিটি 10, আর A হতে T তে একটি বাহু আছে যার ক্যাপাসিটি 20. তাহলে এই গ্রাফে ম্যাক্সফ্লো হবে 10. যদি আগের গ্রাফে ক্যাপাসিটি ঠিক উল্টো হতো তাহলেও কিন্তু ম্যাক্সফ্লো হতো 10. এবার একটু বড় উদাহরণ দেখা যাক চিত্র ৮.৫ এ। খেয়াল করলে দেখবে যে এখানে বাহুতে f/c আকারে কিছু লেখা আছে। এখানে প্রথম সংখ্যা অর্থাৎ f/c এর f হলো কত ফ্লো হচ্ছে, আর c হলো কত ক্যাপাসিটি। এখন চিত্রের মতো ছাড়া আর অন্য কোনোভাবে ফ্লো দিলেও তুমি 14 এর থেকে বেশি ফ্লো দিতে পারবে না। আশা করি চিত্রের ফ্লো বুঝা যাচ্ছে? যেমন 2 তে দেখো $S-2$ পথে 9 ফ্লো আসে আর $2-3$ পথে 2, $2-1$ পথে 3 আর $2-4$ পথে 4 ফ্লো বের হয়। অর্থাৎ যতখানি ফ্লো ঢুকবে ঠিক ততখানিই বের হয়ে যাবে। এরকম কথা 1, 2, 3, 4 সব নোডের ক্ষেত্রেই প্রযোজ্য। শুধু S আর T ছাড়া। S হতে বের হয় 14 পরিমাণ, আর T তে ঢুকে 14 পরিমাণ। এই পরিমাণই সর্বোচ্চ করাই ম্যাক্সফ্লো এর লক্ষ্য।



নকশা ৮.৫: ম্যাক্সিমাম ফ্লো (Maximum flow)

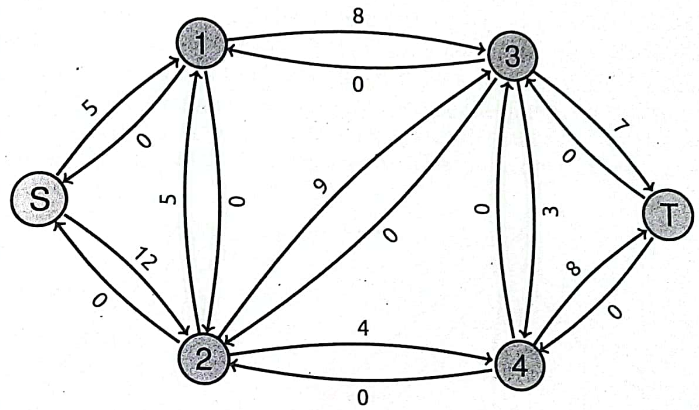
তাহলে আশা করি ম্যাক্সফ্লো কী জিনিস তা বোঝা গেছে? এখন আসো কঠিন অংশে। কীভাবে ম্যাক্সফ্লো সমস্যা সমাধান করা যায়? সহজ হতে শুরু করা যাক। একটি খুব সাধারণ উপায় হলো যতক্ষণ আমরা S হতে T তে এমন একটি পাথ পাব যা দিয়ে কিছু পরিমাণ ফ্লো দেওয়া যায় সেই পাথে ফ্লো দেওয়া। যখন আর এমন কোনো পাথ পাব না তাকে ম্যাক্সফ্লো বলবো। কিন্তু এটা কাজ করবে না। চিত্র ৮.৫ এর গ্রাফে আমরা যদি অন্যভাবে ফ্লো দেই তাহলেই দেখতে পারবে যে মোট ফ্লো 14 না অথচ আর কোনো ফ্লো দিতে পারছ না। মনে কর, $S-2-3-T$ দিয়ে 7, $S-2-$ $3-4-T$ দিয়ে 2, $S-1-3-4-T$ দিয়ে 1 এবং $S-2-4-T$ দিয়ে 3 ফ্লো যদি দাও তাহলে মোট ফ্লো হয় $7+2+1+3=13$ এবং আমাদের গ্রাফ দেখতে চিত্র ৮.৬ এর মত হবে। এখানে দেখো আর কোনো পাথ পাবে না যা দিয়ে তুমি আরও ফ্লো দিতে পারো কিন্তু এর ফ্লো হলো 13. আমরা চিত্র ৮.৫ এ দেখে এসেছি যে অন্তত 14 পাওয়া যায়। সুতরাং আমাদের এ পদ্ধতি কাজ করে না।



নকশা ৮.৬: ম্যাক্সিমাম ফ্লো (Maximum flow): লোকাল ম্যাক্সিমা (local maxima) কিন্তু ম্যাক্সিমাম (maximum) নয়

তাহলে কীভাবে আমরা ম্যাক্সফ্লো সমস্যা সমাধান করব? এই সমাধান বুঝতে হলে আমাদের গ্রাফে কিছু পরিবর্তন করতে হবে। প্রথম পরিবর্তন হবে গ্রাফের উপস্থাপনে। এতক্ষণ কোনো বাহুতে f/c বলতে আমরা ফ্লো ও ক্যাপাসিটি বুঝিয়েছি। এখন থেকে আমরা দুটি সংখ্যা f/c ব্যবহার না করে মাত্র একটি সংখ্যা cf ব্যবহার করব। cf হলো রেসিডিউয়াল ক্যাপাসিটি (residual capacity) বা গাণিতিকভাবে $c-f$. রেসিডিউয়াল ক্যাপাসিটি মানে হলো আরও কত ফ্লো যেতে পারে। যেমন চিত্র ৮.৫ এ 2 হতে 3 নোডের মাঝের বাহুতে আছে 2/9 এটি নতুন উপস্থাপনে হবে $9-2=7$ আবার S হতে 1 এর মাঝের উপস্থাপন 5/5 হতে পরিবর্তন হয়ে 0 হবে। দ্বিতীয় পরিবর্তন হলো গ্রাফের প্রতিটি বাহুর জন্য আমাদের আরও একটি বাহু থাকবে যার দিক বা ডিরেকশন (direction) হবে সম্পূর্ণ উল্টো। অর্থাৎ আমাদের গ্রাফে যদি 1 হতে 3 এ কোনো বাহু থাকে তাহলে আমাদের 3 হতে 1 এর দিকে একটি নতুন বাহু যোগ করতে হবে। এখন প্রশ্ন হলো এই বাহুর প্রাথমিক cf কী হবে? তার আগে সব মূল বাহুর প্রাথমিক cf কী হবে তা পরিষ্কার করি। এটি হবে c, কারণ প্রথমে কোনো ফ্লো থাকবে না তাই $f=0$ এবং $cf=c-f=c-0=c$.

এখন এর বিপরীত বাহুর কথা ভাবা যাক, এর প্রাথমিক লেবেল হবে 0. কারণ আমাদের এর ভেতর দিয়েও কোনো প্রাথমিক ফ্লো যাবে না এবং এর ক্যাপাসিটিও 0. তাহলে এখন আমরা চিত্র ৮.৭ এ আমাদের গ্রাফের প্রাথমিক রূপ (মানে যখন কোনো ফ্লো যায় নাই সেই অবস্থা) পরিবর্তিত উপস্থাপন (cf এর মাধ্যমে) এ দেখি। আশা করি বুঝতে পারছ যে লাল বাহুগুলো হলো উল্টো বাহু।



নকশা ৮.৭: ম্যাক্সিমাম ফ্লো (Maximum flow): পরিবর্তিত উপস্থাপনে প্রাথমিক রূপ

এখন যা করতে হবে তা হলো এমন একটি পাথ খুঁজে বের করতে হবে যার প্রতিটি বাহুতে কিছু পরিমাণ রেসিডিউয়াল ক্যাপাসিটি থাকে। অর্থাৎ তোমরা S হতে শুরু করবে এবং একটি BFS বা DFS করে T পর্যন্ত যাওয়ার চেষ্টা করবে সেসব বাহু ব্যবহার করে যাদের $cf>0$. এই পাথকে বলা হয় অগমেন্টিং পাথ (augmenting path). এখন যা করতে হবে তা হলো এই পাথের সব বাহু এর cf এর মধ্যে সর্বনিম্ন cf বের করতে হবে। আমরা এই পুরো পাথ দিয়ে এই পরিমাণ ফ্লো করাব। মনে কর এই সর্বনিম্ন cf হলো x. তাহলে যা করতে হবে তা হলো এই পাথের সব বাহুর cf কে x পরিমাণ কমাতে হবে। কারণ cf বলে কী পরিমাণ আরও ফ্লো করানো যাবে আর যেহেতু আমরা x পরিমাণ ফ্লো করছি সেহেতু রেসিডিউয়াল ক্যাপাসিটি x পরিমাণ কমাতে হবে। এটুকু তো বোঝা গেল কিন্তু এর পর যা বলব সেটিই হলো মূল ঝামেলা। সেটি হলো, আমরা যেই যেই বাহুর cf কে x কমিয়েছি তার উল্টো বাহুর cf কে x বাড়াতে হবে। অর্থাৎ যদি অগমেন্টিং পাথ কোনো বাহু দিয়ে যায় তাহলে তার উল্টো বাহুর cf বাড়াতে হবে। এই পুরো প্রসেসকে অগমেন্ট (augment) করা বলে। এখন কথা হলো আমরা কেন উল্টো দিকের বাহুগুলোর cf বাড়াব? এটি আসলে বলে অতটা ভালো করে বোঝানো সম্ভব না, এটি কিছুটা অনুভবের বিষয়। তবুও বলি, যদি আমরা a হতে b এর দিকে ফ্লো দেই এর মানে হলো S হতে কোনোভাবে আমরা a তে এসেছি এর পর $a-b$ বাহু দিয়ে আমরা b তে এসেছি, এর পর কোনোভাবে b হতে T তে গিয়েছি। ধরা যাক

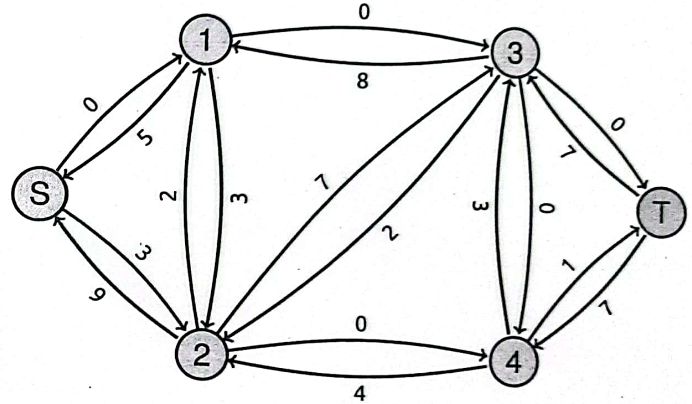
এই পাথের নাম P. এখন কথা হলো আমরা তো চাইলে $a-b$ না গিয়ে $a-c$ ও যেতে পারতাম তাই না? এবং হয়তো ওইভাবে গেলে আমরা অপটিমাল সমাধান পেতাম। কিন্তু আমরা তো $a-b$ দিয়ে চলে এসেছি। কী করা যায়? আচ্ছা মনে কর পরের ধাপে আমরা S হতে কোনোভাবে b তে এসেছি। এখন যদি আমরা a হতে T পর্যন্ত কোনো পাথ পাই তাহলে এই নতুন পাথ আর P মিলে একটি নতুন ফ্লো দিতে পারি। কীভাবে? আমরা নতুন পাথে S হতে b পর্যন্ত আসব, এর পর P যে পাথে b হতে T তে গিয়েছে সেই পাথে এই নতুন অগমেন্টিং পাথ যাবে, আর আগের পাথ S হতে a তে এসে $a-b$ এর মধ্য দিয়ে না গিয়ে a হতে T পর্যন্ত যাওয়ার যেই নতুন পাথ আমরা পেয়েছি সেই পাথে যাবে। অর্থাৎ আগে আমরা a হতে b তে যেই ফ্লো দিয়েছিলাম সেটি বলতে পার আমরা বাতিল করছি। বা এভাবেও ভাবতে পার যে S হতে b হয়ে a তে গিয়ে এর পর T তে ফ্লো দিলাম। এটি সম্ভব হবে যদি উল্টো দিকের cf বাড়ানো হয়। তাহলে এটিই আমাদের অ্যালগরিদম। তবে একটি জিনিস তুমি যদি অগমেন্টিংয়ের জন্য DFS ব্যবহার কর তাহলে আসলে অনেক সময় লেগে যাবে। এটি ম্যাক্সফ্লো এর ফোর্ড ফালকারসন অ্যালগরিদম (Ford Fulkerson Algorithm) নামে পরিচিত। এতে তুমি চাইলে এমন একটি গ্রাফ বানাতে পার যেন তোমার মোট ফ্লো এর সমান সংখ্যকবার অগমেন্ট করতে হতে পারে। ফলে যদি নোড সংখ্যা কমও হয় কিন্তু ক্যাপাসিটির উপর নির্ভর করবে এর রানটাইম। এর টাইম কমপ্লেক্সিটি $O(E \times answer)$. তবে তুমি যদি BFS ব্যবহার কর অগমেন্টিং পাথ বের করার জন্য তাহলে এর রানটাইম হবে $O(VE^2)$ যেখানে V হলো ভার্টেক্সের সংখ্যা আর E হলো বাহুর সংখ্যা। আর একে এডমন্ডস কার্প অ্যালগরিদম (Edmonds Karp Algorithm) বলা হয়।

একটি ছোট কাজ করলেই আমরা এর রানটাইম প্রায় $O(V^2E)$ করতে পারি। আমাদের যা করতে হবে তা হলো, প্রথমে রেসিডিউয়াল ক্যাপাসিটির উপর একটি পূর্ণ BFS চালাতে হবে[১]। পূর্ণ BFS চালানোর সময় BFS এর ডেপথ বা গভীরতার তথ্য রাখতে হবে। অর্থাৎ BFS কে তো একটি ট্রি হিসেবে কল্পনা করা যায়। কোন নোড কত গভীরতায় আছে সেই তথ্য রাখতে হবে। এর পর একটি ব্যাকট্র্যাক (backtrack) বা DFS এর মতো কাজ করতে হবে। আমরা S হতে শুরু করব এবং প্রতিবার পরের গভীরতার কোনো নোডে যাওয়ার চেষ্টা করব (যদি রেসিডিউয়াল ক্যাপাসিটি ধনাত্মক হয়)। যদি আমরা T তে পৌঁছাই তাহলে এই পাথটি অগমেন্ট করব। এভাবে চলতে থাকবে। একে ডিনিকের অ্যালগরিদম (Dinic's algorithm) বা ডিনিকের ব্লকিং অ্যালগরিদম (Dinic's blocking algorithm) বলা হয়। এর থেকেও ভালো অ্যালগরিদম সাধারণত দরকার হয় না।

আরও কিছু বলার আগে ছোটখাটো দুই একটি কথা জেনে রাখা ভালো। প্রথমত আমি এখানে বোঝানোর সুবিধার জন্য ইচ্ছা করে ডিরেক্টেড গ্রাফ নিয়েছিলাম। কিন্তু তুমি যদি বই পড় বা সমস্যা দেখ তাহলে দেখবে বেশির ভাগ সময় আনডিরেক্টেড বাহুতে ক্যাপাসিটি দেওয়া থাকে। এসময় যা করতে হয় তা হলো দুই দিকের জন্য দুটি ডিরেক্টেড বাহু তোমাকে লাগাতে হবে। তুমি চাইলে এই দুটি দিয়েই কাজ সারতে পার বা এই দুটির উল্টো দিকে আরও দুটি বাহু লাগাতে পার। আরেকটি বিষয় হলো তুমি কীভাবে এই গ্রাফের বাহু বা cf রাখবে? তুমি চাইলে অ্যাডজাসেন্সি ম্যাট্রিক্স রাখতে পার। তাতে সমস্যা হলো BFS এর সময় আর তোমার টাইম কমপ্লেক্সিটি $O(()E)$ থাকবে না

[১]পূর্ণ বলার কারণ হলো এর আগেরবার আমরা S হতে BFS শুরু করতাম আর T তে পৌঁছে গেলেই আমরা শেষ করে দিতে পারতাম চাইলে। কিন্তু এবার যতক্ষণ না সব ভার্টেক্সই $visited$ হচ্ছে ততক্ষণ আমাদের BFS চালাতে হবে।

$O(V^2)$ হয়ে যাবে। আবার তুমি যদি অ্যাডজাসেন্সি লিস্ট কর তাহলে উল্টো দিকের cf পরিবর্তন করা আবার আরেক ঝামেলা। তুমি চাইলে একই সঙ্গে অ্যাডজাসেন্সি লিস্ট ও ম্যাট্রিক্স রাখতে পার। তবে আমি ভেক্টর (vector) দিয়ে অ্যাডজাসেন্সি লিস্ট বানিয়ে এই কোড করে থাকি। মনে কর তোমাকে x হতে y এর দিকে একটি বাহু দেওয়া হলো। প্রথমে x আর y এর অ্যাডজাসেন্সি লিস্টে কতগুলো উপাদান আছে তা দেখে নাও। ধরা যাক এই দুটি সংখ্যা যথাক্রমে $size_x$ এবং $size_y$. এর মানে তুমি x হতে y এই বাহুটি যখন আমাদের ডেটা স্ট্রাকচারে রাখবে একটি বাহু থাকবে x এর লিস্টে sz_x তম স্থানে আর তার উল্টো বাহু থাকবে y এর sz_y স্থানে। তোমাকে যা করতে হবে তা হলো একটি বাহু যখন ইনসার্ট করবে তখন 3 টি তথ্য রাখবে: অপর প্রান্ত, রেসিডিউয়াল ক্যাপাসিটি এবং রিভার্স (reverse) বা উল্টো বাহুর ইনডেক্স (index). শেষ। এখন তুমি কোনো বাহুর cf কমানোর সময় খুব সহজেই তার উল্টো দিকের বাহুতে গিয়ে তার cf এর মান পরিবর্তন করে ফেলতে পার। তাহলে চিত্র ৮.৭ এর ম্যাক্সফ্লো রূপটি চিত্র ৮.৮ এ দেখানো হলো।



নকশা ৮.৮: ম্যাক্সিমাম ফ্লো (Maximum flow): ম্যাক্সফ্লো (maxflow) তে গ্রাফের ছবি

৮.১৩.২ মিনিমাম কাট (Minimum cut)

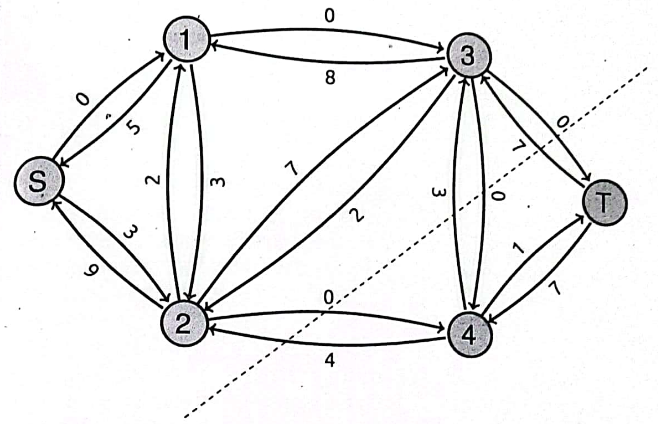
যেকোনো অপটিমাইজেশন সমস্যার একটি দ্বিত্ব বা dual সমস্যা থাকে। অর্থাৎ যদি একটি সমস্যা থাকে যেখানে আমাদের কোনো কিছুর মান বাড়াতে বা ম্যাক্সিমাইজেশন (maximization) করতে হবে তাহলে আমরা সেই সমস্যাকে অন্যভাবে দেখতে পারি যেখানে কোনো কিছুর মানকে কমাতে বা মিনিমাইজেশন (minimization) করতে হয়। প্রধান সমস্যা বা primal এর যা উত্তর হয় dual এরও একই উত্তর হয়। আমরা কিছুক্ষণ আগে ম্যাক্সফ্লো সমস্যা দেখলাম। সেখানে

আমরা ফ্লো কে ম্যাক্সিমাইজেশন করেছিলাম। এখন কথা হলো এর dual সমস্যা কী? এটিই হলো মিনিমাম কাট (minimum cut) বা সংক্ষেপে মিনকাট (mincut). মিনকাট সমস্যাটি হলো, আগের মতোই সোর্স বা সিঙ্ক থাকবে এবং বাহুসমূহে ক্যাপাসিটি থাকবে। তোমাকে কিছু বাহু ডিলিট করে সোর্স এবং সিঙ্ককে ডিসকানেক্টেড করতে হবে যেন সোর্সের কম্পোনেন্ট থেকে সিঙ্কের কম্পোনেন্টে যাওয়া বাহুসমূহের ক্যাপাসিটির যোগফল বিয়োগ সিঙ্কের কম্পোনেন্ট থেকে সোর্সের কম্পোনেন্টে যাওয়া বাহুসমূহের ক্যাপাসিটির যোগফল সর্বনিম্ন হয়। খেয়াল কর আমি বলেছি ক্যাপাসিটির যোগফল রেসিডিউয়াল ক্যাপাসিটির না কিন্তু! যেমন চিত্র ৮.৭ এ যদি আমরা $(S, 1, 2)$ কে একটি কম্পোনেন্ট আর $(3, 4, T)$ কে আরেকটি কম্পোনেন্ট ধরি তাহলে এই কাট (cut) এর মূল্য হবে $8 + 4 + 9 = 21$. যদি $(S, 1, 2, 3, 4)$ এক কম্পোনেন্ট আর বাকি (T) এক কম্পোনেন্ট হয় তাহলে মূল্য হবে $7 + 8 = 15$. যদি $(S, 1)$ একটি আর $(2, 3, 4, T)$ আরেকটি হয় তাহলে মূল্য হয় $8 - 5 + 12 = 15$. অপটিমাল কাট হবে $(S, 1, 2, 3)$ এবং $(4, T)$ এর ক্ষেত্রে। কারণ এর মূল্য $7+3+4 = 14$ যা ম্যাক্সফ্লো এর সমান। তুমি কাগজে কলমে ম্যাক্সফ্লো বা মিনকাট বের করার সময় নিশ্চিত হওয়ার জন্য এই দুটিই বের করে দেখতে পার, যদি তারা সমান হয় তার মানে তোমার উত্তর ঠিক আছে।

প্রশ্ন হলো মিনকাট কীভাবে বের করা যায়? মানে মিনকাট এর উত্তর তো তোমরা ম্যাক্সফ্লো এর উত্তর থেকেই পেতে পার কিন্তু কীভাবে দুটি কম্পোনেন্টে ভাগ করলে তুমি মিনকাট পাবে তা কীভাবে বের করা যায়? সহজ, ম্যাক্সফ্লো শেষে তোমরা সোর্স হতে শুরু করে একটি BFS চালাবে, BFS এর সময় তুমি সেইসব বাহু দিয়ে যাবে যাদের এখনও কিছু রেসিডিউয়াল ক্যাপাসিটি বাকি আছে। ব্যাস তাহলে *visited* নোডগুলো একটি কম্পোনেন্ট আর *unvisited* নোডগুলো অপর কম্পোনেন্ট তৈরি করবে। যেহেতু ম্যাক্সফ্লো শেষে আমরা এই কাজ করছি সেজন্য অবশ্যই আমরা T কে *visit* করতে পারব না (করা গেলে তো আবার ফ্লো করা যেত)। চিত্র ৮.৮ এ এই BFS চালালে *visited* হওয়া নোডসমূহকে চিত্র ৮.৯ এ দেখানো হলো। এটি আশা করি বোঝা যাচ্ছে যে নীল নোডগুলো একদিক এবং সবুজ নোডগুলো আরেক দিক। যদি তুমি এই দুই সাইডের মধ্যে ক্যাপাসিটি যোগ বিয়োগ কর তাহলে পাবে $7+3+4 = 14$. যদিও এই চিত্রে ক্যাপাসিটি দেখানো নেই কিন্তু তুমি আগের চিত্র ৮.৫ এর সঙ্গে তুলনা করে বাহুগুলোর ক্যাপাসিটি দেখে নিতে পার।

৮.১৩.৩ মিনিমাম কস্ট ম্যাক্সিমাম ফ্লো (Minimum cost maximum flow)

অনেক সময় বাহুসমূহের মূল্য বা কস্ট (cost) ও দেওয়া থাকে। অর্থাৎ এই বাহু দিয়ে যদি তুমি 1 ফ্লো চালাও তাহলে তোমাকে এতো মূল্য দিতে হবে। আর আমাদের কাছে জানতে চাওয়া হয় যে সবচেয়ে কম খরচে সবচেয়ে বেশি সংখ্যক ফ্লো কীভাবে দেওয়া যায়। এর সমাধান ম্যাক্সফ্লো এর মতোই। শুধু পার্থক্য হলো এখানে আমরা S হতে T তে যাওয়ার পথ, BFS বা DFS করে বের করব না। বরং বেলম্যান ফোর্ড অ্যালগরিদম বা repeated ডায়াকস্ট্রা'র অ্যালগরিদম করে বের করব। আর প্রতিবার ফ্লো এর মূল্যগুলো যোগ করব। তাহলেই মিনিমাম কস্ট ম্যাক্সিমাম ফ্লো (Minimum cost maximum flow) সমাধান হয়ে যাবে।



নকশা ৮.৯: মিনিমাম কাট (Minimum cut) বা মিনকাট (mincut)

৮.১৩.৪ ম্যাক্সিমাম বাইপারটাইট ম্যাচিং (Maximum Bipartite Matching)

প্রথমে আমাদের জানতে হবে বাইপারটাইট গ্রাফ (bipartite graph) কী জিনিস। এটি এমন একটি গ্রাফ যেখানে নোডগুলোকে দুই ভাগে ভাগ করা যায় যেন প্রতিটি ভাগের নোডগুলোর মধ্যে কোনো বাহু না থাকে। যা বাহু আছে তা হবে এই দুই ভাগের মধ্যে। এর বাস্তব উদাহরণ এরকম হতে পারে, মনে কর তোমার কাছে কিছু বল আর কিছু বাক্স আছে। যদি বল আর বাক্সকে তুমি নোড দিয়ে প্রকাশ কর তাহলে তুমি সেই সব বল আর বাক্সের মধ্যে বাহু দিবে যেন সেই বল সেই বাক্সের মধ্যে আঁটে। এই গ্রাফটি অবশ্যই একটি বাইপারটাইট গ্রাফ কারণ এখানে দুটি বল বা দুটি বাক্সের মধ্যে কোনো বাহু নেই। এখন যদি তোমাকে জিজ্ঞাসা করা হয় তুমি সর্বোচ্চ কতগুলো বল কোনো বাক্সতে ভরতে পারবে? মানে অবশ্যই তুমি একটি বলকে দুটি বাক্সে বা কোনো বাক্সে দুটি বল রাখতে পারবে না। এই সমস্যাটিই হলো ম্যাক্সিমাম বাইপারটাইট ম্যাচিং (maximum bipartite matching). অর্থাৎ তুমি সর্বোচ্চ কতগুলো বাহু নির্বাচন করতে পারবে যেন কোনো নোডে একাধিক নির্বাচিত বাহু না থাকে।

ম্যাক্সফ্লো ব্যবহার করে এই সমস্যা খুব সহজেই সমাধান করা যায়। মনে কর নোডগুলো যেই দুই ভাগে বিভক্ত তাদের নাম L ও R (left, right এর সংক্ষিপ্ত রূপ)। এখন তুমি একটি সোর্স S নাও এবং S হতে L এর সব নোডের 1 ক্যাপাসিটির বাহু দাও। একইভাবে একটি সিঙ্ক T নাও

এবং R এর সব নোড হতে T তে 1 ক্যাপাসিটির বাহু দাও। আর L ও R এর মধ্যের বাহুগুলোর 1 ক্যাপাসিটি করে দিতে হবে। তাহলে এই গ্রাফের ম্যাক্সফ্লো-ই হলো ম্যাক্সিমাম বাইপারটাইট ম্যাচিং। কেন এটি সঠিক তা আশা করি বলতে হবে না।

যদিও আমরা ম্যাক্সফ্লো ব্যবহার করে সমাধান করতে পারি কিন্তু আসলে আমরা এসব অতিরিক্ত নোড না লাগিয়েই সমাধান করতে পারি। আমরা এখন যেই সমাধানের কথা বলব সেটি আসলে এডমন্ডস কার্প অ্যালগরিদমের বাইপারটাইট গ্রাফ ভার্সন মনে করতে পার। প্রথমে দুটি অ্যারে নাও $matchL$ এবং $matchR$ এবং এদের -1 দিয়ে ইনিশিয়ালাইজেশন কর। যদি L এ থাকা একটি নোড i হয় তাহলে $matchL[i]$ হলো সেই নোডের সঙ্গে ম্যাচিং (matching) হওয়া R সাইডের নোড। $matchR$ একই রকম জিনিস। এখন একে একে L থেকে একটি করে নোড নাও (ধর নোডটি হলো x) আর একটি কাজ কর। কাজটি চাইলে BFS এর মতো করে করতে পার বা DFS এর মতো করে করতে পার। আমি DFS এর মতো করে করি কারণ এতে কোড বেশ ছোট হয়। আমি এখানে DFS করে কীভাবে করতে হবে তার বর্ণনা দিচ্ছি। যেহেতু আমরা DFS করব তাই তোমাদের একটি $visited$ এর অ্যারে নিতে হবে এবং একে 0 দিয়ে ইনিশিয়ালাইজেশন করতে হবে (আমরা কিন্তু এই ইনিশিয়ালাইজেশন এবং DFS L এর প্রতিটি ভার্টেক্সের জন্য করব)। এখন তুমি x হতে DFS শুরু কর। তুমি যেই নোডের জন্য DFS এ আছ (ধরা যাক y, অর্থাৎ প্রথমে y এর মান হবে x) তার প্রতিবেশী নোডগুলো একে একে দেখ। আমরা এমনভাবেই কাজ করব যেন y সবসময় L এর হয়ে থাকে। সুতরাং এর প্রতিবেশী হবে R এর, ধরা যাক এরকম একটি প্রতিবেশী হলো z. এখন তোমাকে দেখতে হবে $matchR[z]$ কি -1 কিনা। যদি -1 না হয় তাহলে $matchR[z]$ এর DFS কল করতে হবে, তবে কল করার আগে দেখে নিও যে এই নোডটি আগেই $visited$ কিনা। যদি $visited$ হয় তাহলে আর কল করার দরকার নেই। আর যদি -1 হয় এর মানে একটি অগমেন্টিং পাথ পেয়েছ। তখন তোমাকে $matchL[y] = z$ এবং $matchR[z] = y$ করতে হবে এবং 1 রিটার্ন করতে হবে। কিছুক্ষণ আগে কিন্তু তুমি $matchR[z]$ -1 না হলে DFS কল করেছিলে। যদি এই DFS তোমাকে 1 রিটার্ন করে তার মানে তুমি অগমেন্টিং পাথ পেয়েছ এবং আগের মতোই $matchL[y] = z$ এবং $matchR[z] = y$ করবে এবং 1 রিটার্ন করে বুঝাতে হবে যে তুমি অগমেন্টিং পাথ পেয়ে গেছো। আর যদি দেখ y এর জন্য সব z শেষ কিন্তু তাও তুমি অগমেন্টিং পাথ পাওনি তাহলে 0 রিটার্ন কর। এই DFS এর কোডটি দেখতে কোড ৮.৯ এর মতো হবে।

কোড ৮.৯: bpm dfs.cpp

```
int dfs(int y) {
    visited[y] = 1;
    for (int i = 0; i < V[y].size(); i++) {
        int z = V[y][i];
        if (matchR[z] == -1 ||
            (!visited[matchR[z]] && dfs(matchR[z]))) {
```



```
            matchL[y] = z;
            matchR[z] = y;
            return 1;
        }
    }
    return 0;
}
```

একদম শুরুতে তো x এর জন্য DFS কল করেছিলে। যদি এই কল 1 রিটার্ন করে এর মানে তোমার ম্যাচিং 1 বেড়েছে বা তুমি চাইলে $matchL$ এ দেখতে পার কতগুলোর জন্য -1 নেই এভাবে তুমি ম্যাক্সিমাম ম্যাচিং কয়টি তা বের করতে পার আর $matchL$ দেখে এও বলতে পার যে কার সঙ্গে কে ম্যাচিং করেছে। এর টাইম কমপ্লেক্সিটি হলো $O(VE)$ কারণ তুমি V সংখ্যকবার DFS চালাচ্ছ।

তবে তুমি যদি এই অ্যালগরিদমকে একটু পরিবর্তন কর তাহলে $O(E\sqrt{V})$ টাইম কমপ্লেক্সিটির অ্যালগরিদম পেয়ে যাবে যার নাম হপক্রফট কার্প অ্যালগরিদম (Hopcroft Karp Algorithm). যা করতে হবে তা হলো একটি একটি করে নোড নিয়ে DFS না করে, L এর সব আনম্যাচড (unmatched) নোড নিয়ে প্রথমে BFS কর এবং BFS ট্রি তে L এর সব নোডের গভীরতা লিখে রাখো। এর পর একটি DFS চালিয়ে শুধুমাত্র পরের গভীরতার নোডে যেতে চেষ্টা করো এবং দেখ একটি অগমেন্টিং পাথ পাও কিনা। কিছুটা ডিনিকের অ্যালগরিদমের মতো। এটিই তোমাকে $O(E\sqrt{V})$ টাইম কমপ্লেক্সিটি দিবে। আমি মূল ধারনাটি সহজভাবে বললাম। ইন্টারনেটে আরও বিস্তারিত পড়তে পার।

৮.১৩.৫ ভার্টেক্স কাভার (Vertex cover) ও ইনডিপেন্ডেন্ট সেট (Independent set)

ম্যাক্সিমাম বাইপারটাইট ম্যাচিং সমস্যার সঙ্গে জড়িত দুটি সমস্যা হলো ভার্টেক্স কাভার (vertex cover) এবং ইনডিপেন্ডেন্ট সেট (independent set). ভার্টেক্স কাভার বলে এমন কিছু নোড নির্বাচন করতে যেন সব বাহুর কোনো একটি মাথা যেন অবশ্যই নির্বাচিত ভার্টেক্সগুলোর মধ্যে একটি হয়। যেহেতু আমরা চাইলেই সব নোড নির্বাচন করতে পারি তাই আমাদের লক্ষ্য হলো সবচেয়ে কম সংখ্যক নোড নির্বাচন করা। এ জন্য এই সমস্যাকে বলা হয় মিনিমাম ভার্টেক্স কাভার (minimum vertex cover). ইনডিপেন্ডেন্ট সেট ঠিক এর উল্টো সমস্যা। এই সমস্যায় বলে এমন কিছু নোড নির্বাচন করতে যেন তাদের মধ্যে কোনো বাহু না থাকে। যেহেতু আমরা চাইলেই মাত্র একটি নোড নির্বাচন করতে পারি সেহেতু আমাদের লক্ষ্য হলো সবচেয়ে বেশি সংখ্যক নির্বাচন করা, সেজন্য একে বলা হয় ম্যাক্সিমাম ইনডিপেন্ডেন্ট সেট (maximum independent set). কথা হলো সাধারণ গ্রাফে এই দুটি সমস্যা NP অর্থাৎ আমরা জানি না এদের ভালো কোনো সমাধান আছে কিনা। তবে বাইপারটাইট গ্রাফে এদের খুব সুন্দর সমাধান আছে। সেজন্য আমরা এখানে