গ্রাফ থিওরিতে হাতেখড়ি – ১

shafaetsplanet.com/

শাফায়েত

December 21, 2010

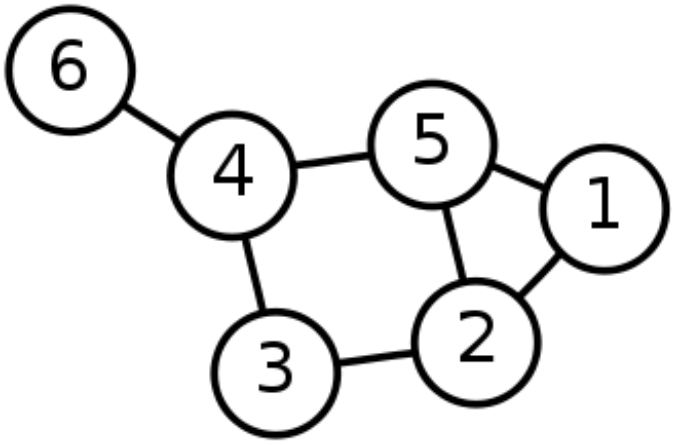
তুমি কি জানো পৃথিবীর প্রায় সব রকমের প্রবলেমকে কিছু রাস্তা আর শহরের প্রবলেম বানিয়ে সলভ করে ফেলা যায়? গ্রাফ থিওরির যখন জন্ম হয় তখন তোমার আমার কারোই জন্ম হয়নি, এমনকি পৃথিবীতে কোন কম্পিউটারও ছিলোনা! সেই সতেরো দশকের শেষের দিকে লিওনার্দ অয়লার গ্রাফ থিওরি আবিষ্কার করেন কনিসবার্গের সাতটি ব্রিজের সমস্যার সমাধান করতে। তখনই সবার কাছে পরিস্কার হয়ে যায় যে যেকোন প্রবলেমকে শহর-রাস্তার প্রবলেম দিয়ে মডেলিং করে চেনা একটা প্রবলেম বানিয়ে ফেলতে গ্রাফ থিওরির জুড়ি নেই। আর যখনই তুমি একটা প্রবলেমকে চেনা কোন প্রবলেমে কনভার্ট করে ফেলতে পারবে তখন সেটা সমাধান করা অনেক সহজ হয়ে যাবে।

গ্রাফ থিওরির অনেক অ্যাপ্লিকেশন আছে। সবথেকে কমন হলো এক শহর থেকে আরেক শহরে যাবার দ্রুততম পথ খুজে বের করা। তুমি হয়তো জানো সার্ভার থেকে একটা ওয়েবপেজ তোমার পিসিতে পৌছাতে অনেকগুলো রাউটার পার করে আসতে হয়, গ্রাফ থিওরি দিয়ে এক রাউটার থেকে আরেকটাতে যাবার পথ খুজে বের করা হয়। যুদ্ধের সময় একটা দেশের কোন কোন রাস্তা বোমা দিয়ে উড়িয়ে দিলে দেশের রাজধানী সব শহর থেকে বিচ্ছিন্ন হয়ে যাবে সেটাও বের করে ফেলা যায় গ্রাফ থিওরি দিয়ে। আমরা গ্রাফ অ্যালগোরিদমগুলো শেখার সময় আরো অনেক অ্যাপ্লিকেশন দেখবো।

আমাদের এই হাতেখড়ির লক্ষ্য হবে প্রথমেই গ্রাফ থিওরির একদম বেসিক কিছু সংজ্ঞা জানা, তারপর কিভাবে গ্রাফ মেমরিতো স্টোর করতে হয় সেটা জানা, এরপরে গ্রাফের কিছু বেসিক অ্যালগোরিদম জানা এবং সাথে সাথে মজার কিছু প্রবলেম দেখা। সব সংজ্ঞা একসাথে দেখলে মাথায় থাকবেনা, তাই একদম না জানলেই নয় সেরকম কিছু সংজ্ঞা প্রথমে দেখবো, এরপর প্রয়োজনমতো আরো কিছু সংজ্ঞা জেনে নিবো।

গ্রাফ কি?

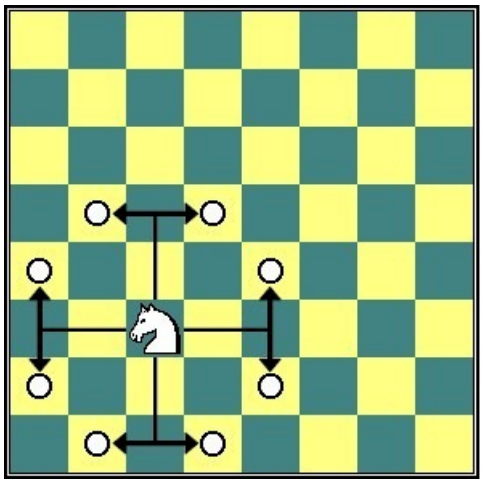
ধরা যাক ৬টি শহরকে আমরা ১,২,৩,৪,৫,৬ দিয়ে চিহ্নিত করলাম। এবার যে শহর থেকে যে শহরে সরাসরি রাস্তা আছে তাদের মধ্যে লাইন টেনে দিলাম:



শহরগুলোর নাম ১,২ ইত্যাদি দিয়ে দিতে হবে এমন কোন কথা নেই, তুমি চাইলে ঢাকা, চট্টগ্রাম ইত্যাদি দিতে পারো। এটা খুবই সাধারণ একটা গ্রাফ যেখানে কিছু শহরের মধ্যের রাস্তাগুলো দেখানো হয়েছে। গ্রাফ থিওরির ভাষায় শহরগুলোকে বলা হয় **নোড(Node)** বা ভারটেক্স(Vertex) আর রাস্তাগুলোকে বলা হয় **এজ(Edge)**। গ্রাফ হলো কিছু নোড আর কিছু এজ এর একটা কালেকশন।

গ্রাফে নোড দিয়ে অনেককিছু বুঝাতে পারে, কোন গ্রাফে হয়তো নোড দিয়ে শহর বুঝায়, কোন গ্রাফে এয়ারপোর্ট, কোন গ্রাফে আবার হয়তো দাবার বোর্ডের একটা ঘর বুঝাতে পারে! আর এজ দিয়ে বুঝায় নোডগুলোর মধ্যের সম্পর্ক। কোন গ্রাফে এজ দিয়ে দুটি শহরের দূরত্ব বুঝাতে পারে, কোন গ্রাফে এক এয়ারপোর্ট থেকে আরেক এয়ারপোর্টে যাবার সময় বুঝাতে পারে, আবার দাবার বোর্ডে একটা ঘরে ঘোড়া থাকলে সেই ঘর থেকে কোন ঘরে যাওয়া যায় সেটাও বুঝাতে পারে।

নিচের ছবিতে দাবার বোর্ডটাও একটা গ্রাফ। প্রতিটা ঘর একটা করে নোড। যে ঘরে ঘোড়া আছে সেখান থেকে এজগুলো দেখানো হয়েছে:

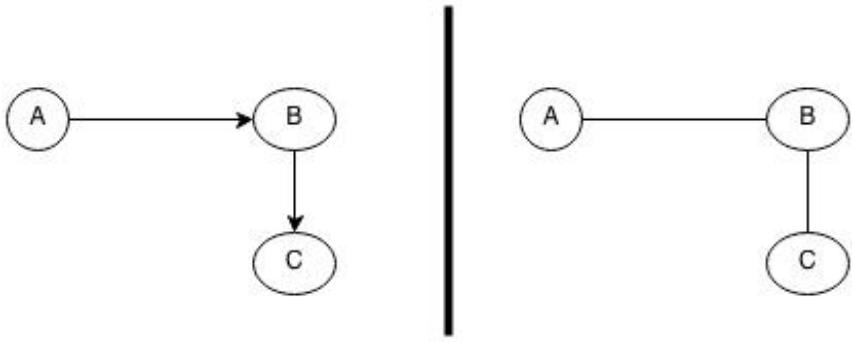


এককথায় নোডের কাজ কোন একধরণের অবজেক্টকে রিপ্রেজেন্ট করা আর এজ এর কাজ হলো দুটি অবজেক্টের মধ্যে সম্পর্কটা দেখানো।

অ্যাডজেসেন্ট নোড:
A নোড থেকে B নোডে একটা এজ থাকলে B কে বলা হয় A এর অ্যাডজেসেন্ট নোড। সোজা কথায় অ্যাডজেসেন্ট নোড হলো এজ দিয়ে সরাসরি কানেক্টেড নোড। একটা নোডের অনেকগুলো অ্যাডজেসেন্ট নোড থাকতে পারে।

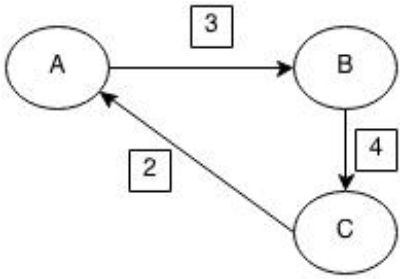
ডিরেক্টেড গ্রাফ আর আনডিরেক্টেড গ্রাফ:
ডিরেক্টেড গ্রাফে এজগুলোতে তীরচিহ্ন থাকে, তারমানে এজগুলো হলো একমুখি(Unidirectional), আনডিরেক্টেড গ্রাফে এজগুলো দ্বিমুখী(Bidirectional)। নিচের

ছবি দেখলেই পরিষ্কার হবে:



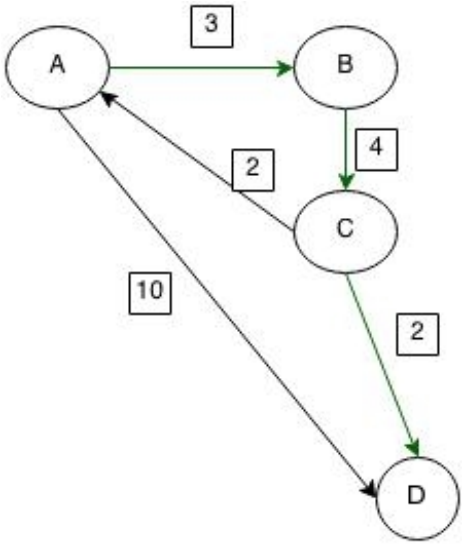
বামের ছবির গ্রাফ ডিরেক্টেড, ডানেরটা আনডিরেক্টেড।

ওয়েটেড আর আনওয়েটেড গ্রাফ:
অনেক সময় গ্রাফে এজগুলোর পাশে ছোট করে ওয়েট(Weight) বা কস্ট(Cost) লেখা থাকতে পারে:



এই ওয়েট বা কস্ট দিয়ে অনেককিছু বুঝাতে পারে, যেমন দুটি শহরের দূরত্ব কত কিলোমিটার, অথবা রাস্তাটি দিয়ে যেতে কত সময় লাগে, অথবা রাস্তা দিয়ে কয়টা গাড়ি একসাথে যেতে পারে ইত্যাদি। আগের গ্রাফগুলো ছিলো আনওয়েটেড, সেক্ষেত্রে আমরা ধরে নেই সবগুলো এজের ওয়েটের মান এক(১)। সবগুলো ওয়েট ১ হলে আলাদা করে লেখা দরকার হয়না।

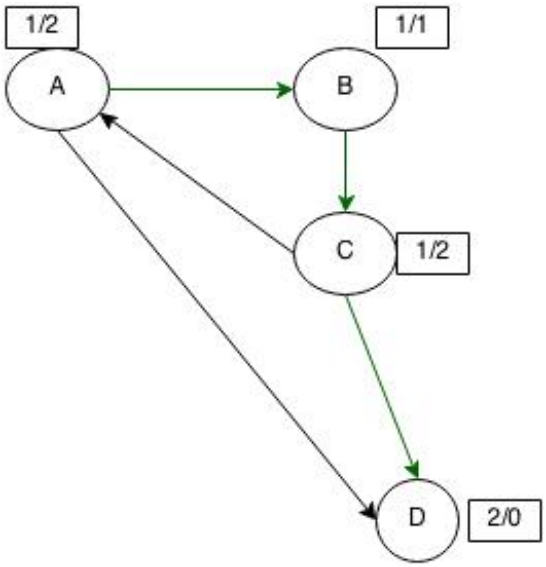
পাথ:
পাথ(Path) হলো যে এজগুলো ধরে একটা নোড থেকে আরেকটা নোডে যাওয়া যায়। অর্থা এজের একটা সিকোয়েন্স।



এক নোড থেকে আরেক নোডে যাবার অনেকগুলো পাথ থাকতে পারে। ছবিতে A থেকে D তে যাবার দুইটা পাথ আছে। A->B,B->C,C->D হলো একটা পাথ, এই পাথের মোট ওয়েট হলো ৩+৪+২=৯। আবার A->D ও একটা পাথ হতে পারে যেই পাথের মোট কস্ট ১০। যে পাথের কস্ট সবথেকে কম সেটাকে **শর্টেস্ট পাথ** বলে।

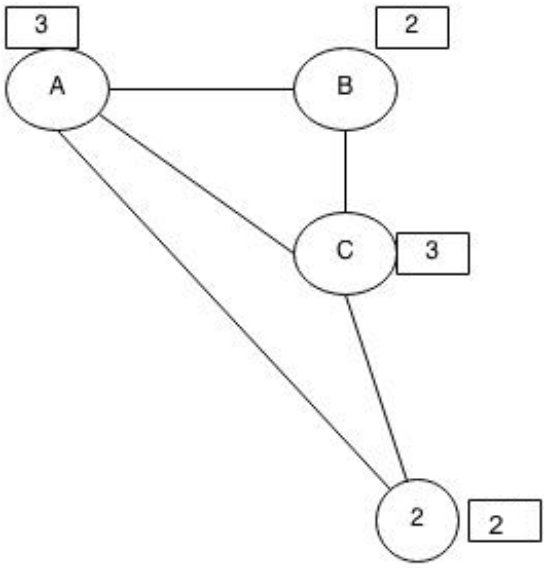
ডিগ্রী:

ডিরেক্টেড গ্রাফে একটা নোডে কয়টা এজ প্রবেশ করেছে তাকে **ইনডিগ্রী**, আর কোন নোড থেকে কয়টা এজ বের হয়েছে তাকে **আউটডিগ্রী** বলে। ছবিতে প্রতিটা নোডের ইনডিগ্রী আর আউটডিগ্রী দেখানো হয়েছে:



আনডিরেক্টেড গ্রাফে ইন বা আউটডিগ্রী আলাদা করা হয়না। একটা নোডের যতগুলো অ্যাডজেসেন্ট নোড আছে সেই সংখ্যাটাই নোডটার ডিগ্রী।

হ্যান্ডশেকিং লেম্মা একটা জিনিস আছে যেটা বলে একটা বিজোড় ডিগ্রীর নোডের সংখ্যা সবসময় জোড় হয়। উপরের গ্রাফে A আর C এর ডিগ্রী ৩, এরা বিজোড় ডিগ্রীর নোড। তাহলে বিজোড় ডিগ্রীর নোড আছে ২টা, ২ হলো একটা জোড় সংখ্যা। হ্যান্ডশেক করতে সবসময় ২টা হাত লাগে, ঠিক সেরকম একটা এজ সবসময় ২টা নোডকে যোগ করে। তুমি একটু চিন্তা করে দেখো:



> ২টা জোড় ডিগ্রীর নোডকে এজ দিয়ে যোগ করলে ২টা নতুন বিজোড় ডিগ্রীর নোড তৈরি হয়।
>
> ২টা বিজোড় ডিগ্রীর নোডকে এজ দিয়ে যোগ করলে ২টা বিজোড় ডিগ্রীর নোড কমে যায়।
>
> ১টা জোড় আর একটা বিজোড় ডিগ্রীর নোড যোগ করলে মোট বিজোড় ডিগ্রীর নোড সমান থাকে(এক পাশে কমে, আরেক পাশে বাড়ে)।

তাহলে দেখা যাচ্ছে হয় ২টা করে বাড়তেসে বা ২টা করে কমতেসে বা সমান থাকছে, তাই বিজোর ডিগ্রীর নোডের সংখ্যা সবসময় জোড়।

একইভাবে এটাও দেখানো যায় **একটা গ্রাফের ডিগ্রীগুলোর যোগফল হবে এজসংখ্যার দ্বিগুণ**। উপরের গ্রাফে ডিগ্রীগুলোর যোগফল ১০, আর এজসংখ্যা ৫।

এগুলো গেল একেবারেই প্রাথমিক কথাবার্তা। পরবর্তি লেখায় তুমি জানতে পারবে কিভাবে ভ্যারিয়েবলে গ্রাফ স্টোর করতে হয়। আশা করি গ্রাফ থিওরীতে তোমার যাত্রাটা দারুণ আনন্দের হবে, অনেক কিছু শিখতে পারবে।

অন্যান্য পর্ব

গ্রাফ থিওরিতে হাতেখড়ি – ২ (ভ্যারিয়েবলে গ্রাফ স্টোর-১)

shafaetsplanet.com/

শাফায়েত

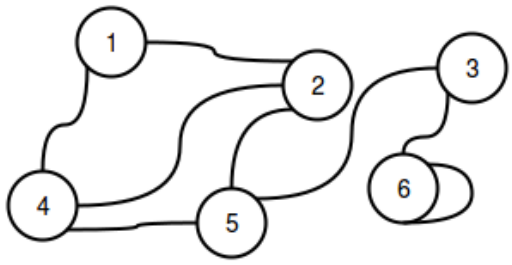
December 26, 2010

আগের পোস্টে আমরা দেখেছি গ্রাফ থিওরি কি কাজে লাগে,আর এলিমেন্টারি কিছু টার্ম শিখেছি। এখন আমরা আরেকটু ভিতরে প্রবেশ করবো। প্রথমেই আমাদের জানা দরকার একটা গ্রাফ কিভাবে ইনপুট নিয়ে স্টোর করে রাখা যায়। অনেকগুলো পদ্ধতির মধ্যে দুটি খুব কমন:

১. অ্যাডজেসেন্সি ম্যাট্রিক্স(adjacency matrix)

২. অ্যাডজেসেন্সি লিস্ট(adjacency list)

অ্যাডজেসেন্ট(adjacent) শব্দটার অর্থ “কোন কিছুর পাশে”। যেমন তোমার পাশের বাড়ির প্রতিবেশিরা তোমার অ্যাডজেসেন্ট। গ্রাফের ভাষায় এক নোডের সাথে আরেকটা নোডে যাওয়া গেলে ২য় নোডটি প্রথমটির অ্যাডজেসেন্ট। এই পোস্টে আমরা ম্যাট্রিক্সের সাহায্যে কোন নোড কার অ্যাডজেসেন্ট অর্থা কোন কোন নোডের মাঝে এজ আছে সেটা কিভাবে স্টোর করা যায় দেখবো। ম্যাট্রিক্স বলতে এখানে শুধুমাত্র ২-ডি অ্যারে বুঝানো হয়েছে, তাই ঘাবড়ে যাবার কিছু নেই!

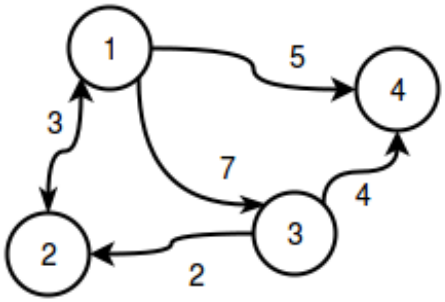


Nodes	1	2	3	4	5	6
1	0	1	0	1	0	0
2	1	0	0	1	1	0
3	0	0	0	0	1	1
4	1	1	0	0	1	0
5	0	1	1	1	0	0
6	0	0	1	0	0	1

গ্রাফের পাশে একটি টেবিল দেখতে পাচ্ছ। এটাই আমাদের অ্যাডজেসেন্সি ম্যাট্রিক্স। ম্যাট্রিক্সের $[i][j]$ ঘরে 1 থাকে যদি i থেকে j তে কোনো এজ থাকে, না থাকলে 0 বসিয়ে দেই।

এজগুলা ওয়েটেড হতে পারে, যেমন ঢাকা থেকে চট্টগ্রামে একটা এজ দিয়ে বলে দিতে পারে শহর দুটির দূরত্ব ৩০০ কিলোমিটার। তাহলে তোমাকে ম্যাট্রিক্সে ওয়েটও বসাতে হবে।

উপরের গ্রাফটি **বাইডিরেকশনাল বা আনডিরেক্টেড**, অর্থা ১ থেকে ২ এ যাওয়া গেলে ২ থেকে ১ এও যাওয়া যাবে। যদি গ্রাফটি **ডিরেক্টেড** হতো তাহলে এজগুলোর মধ্যে তীরচিহ্ন থাকতো। তখনো আমরা আগের মতো করেই ম্যাট্রিক্সে স্টোর করতে পারবো। নিচের ছবিতে ডিরেক্টেড ওয়েটেড গ্রাফের অ্যাডজেসেন্সি ম্যাট্রিক্সের উদাহরণ দেখানো হয়েছে।



Nodes	1	2	3	4
1	inf	3	7	5
2	3	inf	inf	inf
3	inf	2	inf	4
4	inf	inf	inf	inf

যেসব নোড এর ভিতর কোনো এজ নাই তাদেরকে এখানে ইনফিনিটি বা অনেক বড় একটা সংখ্যা দিয়ে দেখানো হয়েছে।

একটা ব্যাপার লক্ষ করো, গ্রাফ আনডিরেক্টেড হলে ম্যাট্রিক্সটি সিমেট্রিক হয়ে যায়, অর্থা mat[i][j]=mat[j][i] হয়ে যায়।

ছোট একটা এক্সারসাইজ:

কল্পনা কর একটি গ্রাফ যার ৩টি নোড আছে edge সংখ্যা ৩,এবং সবগুলো edge bidirectional । edge গুলো হলো ১-২(cost ৫),২-৩(cost ৮),১-৩(cost ৩)। এটার adjacency matrix টা কেমন হবে?

চট করে নিজেই খাতায় একে ফেলতে চেষ্টা কর এবং নিচের উত্তরের সাথে মিলিয়ে দেখো:

```
0 5 3
5 0 8
3 8 0
```

আশা করি বুঝতে পারছ কিভাবে ম্যাট্রিক্সটি আকলাম। না বুঝলে উপরের অংশটা আরেকবার পড়ে ফেল।

গ্রাফ ইনপুট যেভাবে দেয়া হবে:

ঠিক উপরের ম্যাট্রিক্সটা প্রোগ্রামিং প্রবলেমে ইনপুট হিসাবে দিয়ে দেয়া হতে পারে,শুরুতে শুধু নোড সংখ্যা বলে দিবে। লক্ষ্য কর এই ম্যাট্রিক্সটা ইনপুট নিতে আমাদের এজ সংখ্যা জানা জরুরী না। আমাদের একটি ভ্যারিয়েবল লাগবে নোড সংখ্যা ইনপুট নিতে,আরেকটি ২-ডি অ্যারে লাগবে ম্যাট্রিক্স ইনপুট নিতে।

C++

```
int N;
int matrix[100][100]; //এই সর্বোচ্চ ১০০ নোডের গ্রাফ স্টোর করা যাবে।
//ডিক্লেয়ার করার পরে ইনপুট নেবার পালা। খুব সহজ কাজ:
scanf("%d",&N);
for(int i=1;i<=N;i++)
for(int j=1;j<=N;j++)
scanf("%d" ,&matrix[i][j]);

```

সরাসরি ম্যাট্রিক্স না দিয়ে নোড সংখ্যা,edge সংখ্যা বলে দিয়ে edge গুলো কি কি বলে দিতে পারে,এভাবে:

> 3 3 //৩ টা নোড এবং ৩টা এজ
> 1 2 5 //node1-node2-cost
> 2 3 8
> 1 3 3

এটা ইনপুট নিব এভাবে:

C++

```
int Node,Edge;
int matrix[100][100];
scanf("%d%d",&Node,&Edge);
for(i=0;i<Edge;i++)
{
int n1,n2,cost;
scanf("%d%d%d",&n1,&n2,&cost);
matrix[n1][n2]=cost;
matrix[n2][n1]=cost;
}
```

আরো অনেক উপায়ে প্রবলেমে গ্রাফ ইনপুট দিতে পারে। নোডের নম্বর এলোমেলো হতে পারে,যেমন ৩টি নোডকে ১,২,৩ দিয়ে চিহ্নিত না করে ১০০,১০০০০,৪০০ নামে চিহ্নিত করা হতে পারে। সেক্ষেত্রে আমাদের ম্যাপিং করতে হবে। অর্থাত ১০০ কে আমরা ম্যাপ করব ১ দিয়ে,মানে ১০০ বলতে বুঝব ১,১০০০০ বলতে বুঝব ২। index নামক একটি array রেখে index[100]=1;index[100000]=2;এভাবে চিহ্নিত করে দিলেই চলবে। পরে নোড নম্বর ইনপুট দিলে আমার ইনডেক্স থেকে আমাদের দেয়া নম্বর বের করে আনব। ব্যাপারটাকে বলা হয় অ্যারে কম্প্রেশন, তুমি বিস্তারিত জানতে চাইলে পরে কোনো সময় আমার এই লেখাটা দেখতে পারো।

অ্যাডজেসেন্সি ম্যাট্রিক্স ব্যবহার করার সমস্যা:

মেমরি একটা বিশাল প্রবলেম, এজ যতগুলোই থাকুকনা কেন তোমার লাগছে $N*N$ সাইজের ম্যাট্রিক্স যেখানে N হলো নোড সংখ্যা। 10000 টা নোড হলো $N*N $ম্যাট্রিক্সের সাইজ দাড়াবে $4*1000*1000$ বাইট বা প্রায় 381 মেগাবাইট! এজ কম হলে এটা মেমরির বিশাল অপচয়।

কোনো একটা নোড u থেকে অন্য কোন কোন নোডে যাওয়া যায় বের করতে হলে আমাদের N টা নোডের সবগুলো চেক করে দেখতে হবে, টাইমের বিশাল অপচয়!

অ্যাডজেসেন্সি ম্যাট্রিক্স ব্যবহার করার সুবিধা:

$u – v$ নোডের মধ্যে কানেকশন আছে নাকি বা cost কত সেটা খুব সহজেই $mat[u][v]$ চেক করে জেনে যেতে পারি।

এই সমস্যাগুলা দূর করে দিবে অ্যাডজেসেন্সি লিস্ট, সাথে নতুন কিছু সমস্যাও হাজির করবে! তোমরা পরের পর্বে সেটা শিখবে। তার আগে তোমাকে একটা জিনিস শিখতে হবে, সেটা হলো C++ এর স্ট্যান্ডার্ড টেমপ্লেট লাইব্রেরি(STL)। আমরা STL এর ভেক্টর ব্যবহার করে কাজ করবো কারণ এটা ব্যবহার করা খুব সহজ। তুমি নিচের দুটি লিংকের সাহায্যে খুবই সহজে শিখতে পারবে:

১: http://sites.google.com/site/smilitude/stl এটি ফাহিম ভাইয়ের ব্লগের লিংক,তার টিউটোরিয়াল গুলো অদ্বিতীয়।

২: http://www.cplusplus.com/reference/stl/ STL এর বিভিন্ন ফাংশনের কাজ শেখার জন্য সেরা সাইট।

ভেক্টর ব্যবহার শেখা হয়ে গেলে পড়া শুরু করো পরের পর্ব: adjacency list

গ্রাফ থিওরিতে হাতেখড়ি – ৩ (ভ্যারিয়েবলে গ্রাফ স্টোর-২)

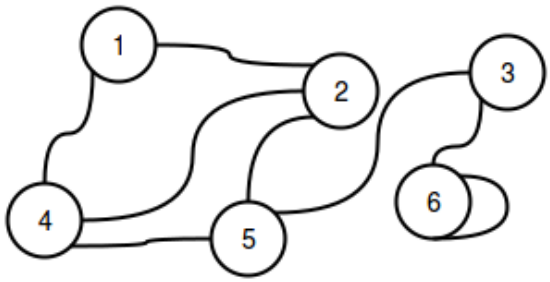
shafaetsplanet.com/

শাফায়েত

December 27, 2010

এই পর্বে গ্রাফ স্টোর করার ২য় পদ্ধতি অ্যাডজেসেন্সি লিস্ট নিয়ে লিখব। এ পদ্ধতিতে গ্রাফ স্টোর করে কম মেমরি ব্যবহার করে আরো efficient কোড লেখা যায়। এ ক্ষেত্রে আমরা ডায়নামিক্যালি মেমরি অ্যালোকেট করব,ভয়ের কিছু নেই সি++ এর standard template library(STL) ব্যবহার করে খুব সহজে কাজটা করা যায়। আগের লেখার শেষের দিকে STL এর উপর কয়েকটি টিউটোরিয়ালের লিংক দিয়েছি, আশা করছি ভেক্টর কিভাবে কাজ করে এখন তুমি জানো।

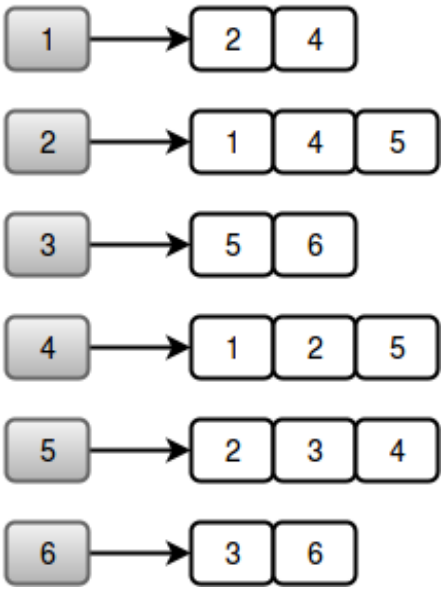
অ্যাডজেসেন্সি লিস্ট শুনতে যতটা ভয়ংকর শুনায়,ব্যাপারটি আসলে তেমনই সহজ। আমরা আবার আগের পোস্টের ছবিটিতে ফিরে যাই:



Nodes	1	2	3	4	5	6
1	0	1	0	1	0	0
2	1	0	0	1	1	0
3	0	0	0	0	1	1
4	1	1	0	0	1	0
5	0	1	1	1	0	0
6	0	0	1	0	0	1

এবার বাজার করার লিস্টের মত একটি লিস্ট বানাই:

এটাই অ্যাডজেসেন্সি লিস্ট, কোন নোডের সাথে কোন নোড যুক্ত আছে সেটার একটা তালিকা। কিন্তু কোড করার সময় কিভাবে এই লিস্টটা স্টোর করবো?



প্রথম উপায়(অ্যারে):

সাধারণ ২ডি অ্যারে ব্যবহার করে লিস্টটি স্টোর করা যায়। যেমন:

arr[1][1]=2, arr[1][2]=4;

arr[2][1]=1; arr[2][2]=4, arr[2][3]=5;

কিন্তু এভাবে স্টোর করলে কিছু সমস্যা আছে:

সমস্যা ১. আমাদের ৬টি নোড আছে। প্রতি নোডের সাথে সর্বোচ্চ ৬টি নোড যুক্ত থাকতে পারে(ধরে নিচ্ছি দুটি নোডের মধ্যে ১টির বেশি সংযোগ থাকবেনা)। এ ক্ষেত্রে আমাদের লাগবে [6][6]আকারের ইন্টিজার অ্যারে। যদি ১ নম্বর নোডের সাথে মাত্র ২টি নোড যুক্ত থাকে তাহলে বাকি array[1][0],array[1][1] কাজে লাগবে,array[1][2] থেকে array[1][6] পর্যন্ত জায়গা কোনো কাজেই লাগবেনা। মনে হতে পারে এ আর এমন কি সমস্যা। কিন্তু চিন্তা কর ১০০০০ টি নোড আছে এমন একটি গ্রাফের কথা। [10000][10000] integer অ্যারে তুমি ব্যবহার করতে পারবেনা,memory limit অতিক্রম করে যাবে,এছাড়া এভাবে মেমরি অপচয় করা ভালো প্রোগ্রামারের লক্ষণ নয় :)। অ্যাডজেসেন্সি ম্যাট্রিক্স ব্যবহার করার সময় যেমন মেমরির সমস্যা হয়েছিলো, এখনও সেই সমস্যা রয়ে যাবে।

সমস্যা ২. অ্যারের কোন ইনডেক্সে কয়টি এলিমেন্ট আছে তার হিসাব রাখতে প্রতি ইনডেক্সের জন্য আরেকটি ভ্যারিয়েবল মেইনটেইন করতে হবে।

দ্বিতীয় উপায়(ভেক্টর):

সমস্যাগুলা দূর করতে আমরা STL vector বা list ব্যবহার করে গ্রাফ স্টোর করব। ভেক্টর/লিস্টে তোমাকে লিস্টের সাইজ বলে দিতে হবেনা,খালি সর্বোচ্চ নোড সংখ্যা বলে দিলেই হবে। এই টিউটোরিয়ালে আমি ভেক্টর ব্যবহার করব কারণ list এ বেশ কিছু সমস্যা আছে।

১০০০০০ নোডের গ্রাফ ইনপুট দেয়ার সময় কখনো ম্যাট্রিক্স হিসাবে দিবেনা,তাহলে ইনপুটের আকারই মাত্রাতিরিক্ত বিশাল হয়ে যাবে। আগের পোস্টে ২য় উদাহরণে যেভাবে দেখিয়েছি সেরকম ইনপুট দিতে পারে, অর্থা প্রথমে নোড আর এজ সংখ্যা বলে দিয়ে তারপর কোন নোডের সাথে কে যুক্ত আছে বলে দিবে। উপরের গ্রাফের জন্য ইনপুট:

```
6 8 //node-edge
1 2 //node1-node2
1 4
2 4
2 5
4 5
5 3
3 6
6 6
```

এটি ভেক্টর দিয়ে ইনপুট নিব এভাবে:

list

```
#include <cstdio>
#include <vector>
using namespace std;
#define MAX 100000 //maximum node
vector <int> edges[MAX];
vector <int> cost[MAX]; //parallel vector to store costs;
int main() {
  int numNodes, numEdges;
  scanf("%d%d", &numNodes, &numEdges);
  for (int i = 1; i <= numEdges; i++) {
    int x, y;
    scanf("%d%d", & x, & y);
    edges[x].push_back(y);
    edges[y].push_back(x);
    cost[x].push_back(1);
    cost[y].push_back(1);
  }
  return 0;
}
```

cost নামক ভেক্টরটি এ গ্রাফের ক্ষেত্রে দরকার ছিলনা,তবে ওয়েটেড গ্রাফে অবশ্যই দরকার হবে। নিশ্চয়ই বুঝতে পারছ edge ও cost ভেক্টর দুটি সমান্তরাল ভাবে কাজ করবে,অর্থাত edge ভেক্টরের যে পজিশনে তুমি দুটি নির্দিষ্ট নোডের কানেকশন পাবে cost ভেক্টরের সেই পজিশনেই তুমি cost পাবে।

যদি ১০০০ বা তার কম নোড থাকে তাহলে ম্যাট্রিক্স বা লিস্ট কোনো ক্ষেত্রেই মেমরি সমস্যা হবেনা। তাও আমরা ভেক্টর দিয়েই গ্রাফ স্টোর করব। কারণ, চিন্তা কর তোমাকে ১০০টা নোডের ম্যাট্রিক্সে ১ এর সাথে কি কি সংযুক্ত আছে বের করতে matix[1][0],matrix[1][1] .....matrix[1][99] এভাবে ১০০টি পজিশন চেক করে কোনটায় কোনটায় ০ নেই বের করতে হবে,১ নম্বর নোডের সাথে যতটি নোডই সংযুক্ত থাকুকনা কেন। তাই এখানেও অ্যারে আমাদের বাড়তি সুবিধা দিতে পারছেনা।

একটা নোডের সাথে কোন কোন নোড যুক্ত আছে বের করা:

ধরো তুমি ১ নম্বর নোডের সাথে যুক্ত সবগুলো নোডের নম্বর চাও, তুমি তাহলে edges[1] এর সাইজ পর্যন্ত লুপ চালাবে এভাবে:

```
C++

1    size=edges[1].size();
2    for(int i=0; i < size ; i++)
3    printf("%d ",edges[1][i]);
```

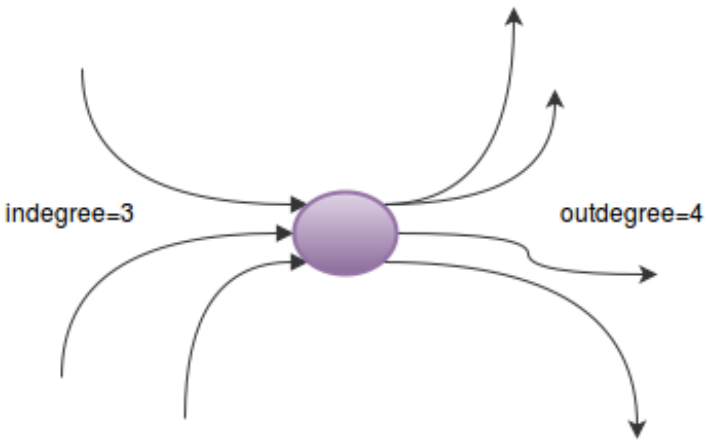
১ >> ০ ১ ০ ০ ০ ১ ১ ০ পুরোটা ঘুরে আসার থেকে ১>>২,৬,৭ ঘুরে আসতে কম সময় লাগবে,তাই নয়কি? ☺ ।

অ্যাডজেসেন্সি ম্যাট্রিক্স কখন লিস্ট অপেক্ষা সুবিধাজনক?

যদি কোনো প্রবলেমে তোমার u,v নোডের এর মধ্যে কোনো এজ আছে নাকি চেক করতে বলে তখন লিস্ট ব্যবহার করলে তোমাকে লুপ চালিয়ে চেক করতে হবে, কিন্তু ম্যাট্রিক্সে জাস্ট matrix[u][v] ইনডেক্স চেক করেই বলে দিতে পারবে তাদের মধ্যে কানেকশন আছে নাকি।

এক্সারসাইজ:

এ পর্যন্ত বুঝে থাকলে তুমি মোটামুটি bfs,dfs এর মত বেসিক অ্যালগোরিদম গুলো শেখার জন্য প্রস্তুত। পরবর্তি লেখাটি পড়ার আগে একটি ছোট exercise করে ফেল। এমন একটি কোড লিখ যেটায় উপরের মত করে ইনপুট দিলে নিচের কাজগুলো করে:
১. একটি adjacency list তৈরি করে। (গ্রাফটিকে directed ধরে নিবে,bidirectional নয়)
২. কোন নোডের সাথে কয়টা নোড যুক্ত আছে,নোডগুলো কি কি সেগুলো প্রিন্ট করে।
৩. indegree হলো একটি নোডে কয়টি নোড প্রবেশ করেছে,outdegree হলো ঠিক তার উল্টোটা। প্রতিটি নোডের outdegree ও indegree প্রিন্ট কর।



পর্ব-৪,বিএফএস:http://www.shafaetsplanet.com/planetcoding/?p=604

গ্রাফ থিওরিতে হাতেখড়ি-৪(ব্রেডথ ফার্স্ট সার্চ)

shafaetsplanet.com/planetcoding/

শাফায়েত

February 22, 2014

আগের পর্বগুলোতে আমরা দেখেছি কিভাবে ম্যাট্রিক্স বা লিস্ট ব্যবহার করে গ্রাফ স্টোর করতে হয়। এবার আমরা প্রথম অ্যালগোরিদম দেখবো এর দিকে যাবো। শুরুতেই আমরা যে অ্যালগোরিদমটা শিখব তার নাম ব্রেডথ ফার্স্ট সার্চ(breadth first search,bfs)।

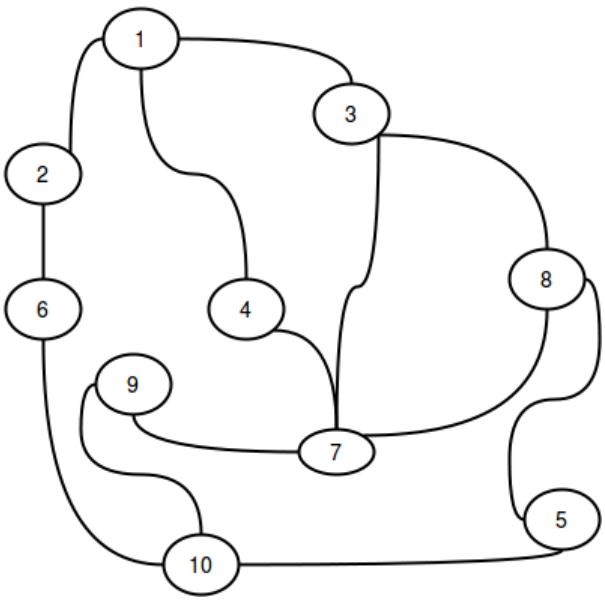
বিএফএস এর কাজ হলো গ্রাফে একটা নোড থেকে আরেকটা নোডে যাবার শর্টেস্ট পাথ বের করা। বিএফএস কাজ করবে শুধুমাত্র আন-ওয়েটেড গ্রাফের ক্ষেত্রে, তারমানে সবগুলো এজের কস্ট হবে ১।

বিএফএস অ্যালগোরিদমটা কাজ করে নিচের ধারণারগুলোর উপর ভিত্তি করে:

> ১. কোনো নোডে ১ বারের বেশি যাওয়া যাবেনা
> ২. সোর্স নোড অর্থা যে নোড থেকে শুরু করছি সেটা ০ নম্বর লেভেলে অবস্থিত।
> ৩. সোর্স বা ‘লেভেল ০’ নোড থেকে সরাসরি যেসব নোডে যাওয়া যায় তারা সবাই ‘লেভেল ১’ নোড।
> ৪. ‘লেভেল ১’ নোডগুলো থেকে সরাসরি যেসব নোডে যাওয়া যায় তারা সবাই ‘লেভেল ২’ নোড। এভাবে লেভেল এক এক করে বাড়তে থাকবে।
> ৫. যে নোড যত নম্বর লেভেলে,সোর্স থেকে তার শর্টেস্ট পথের দৈর্ঘ্য তত।

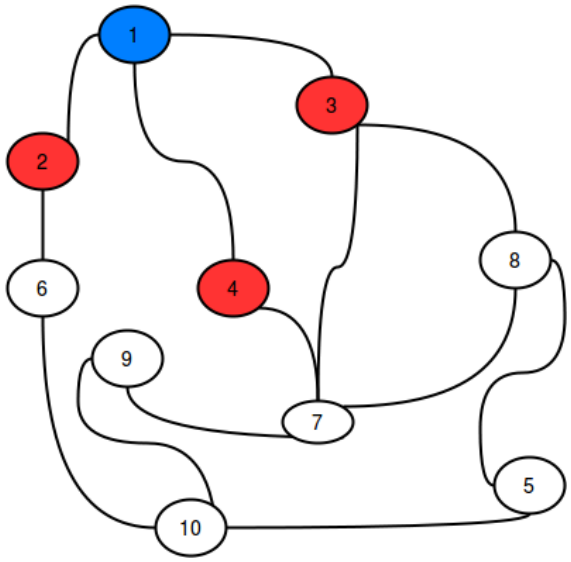
উপরে লেখাগুলো পুরোপুরি না বুঝলে আমরা একটা উদাহরণ দেখে বাকিটা পরিস্কার করব।

ধর তুমি ১ নম্বর শহর থেকে ১০ নম্বর শহরে যেতে চাও। প্রথমে আমরা সোর্স ধরলাম ১ নম্বর নোডকে। ১ তাহলে একটা ‘লেভেল ০’ নোড। ১ কে ভিজিটেড চিহ্নিত করি।

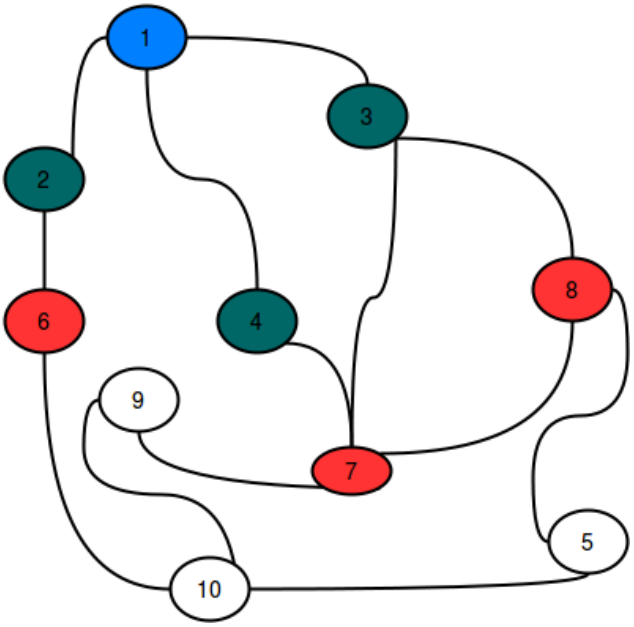


১ থেকে সরাসরি যাওয়া যায় ২,৩,৪ নম্বর নোডে। তাহলে ২,৩,৪ হলো ‘লেভেল ১’ নোড। এবার সেগুলোকে আমরা ভিজিটেড চিহ্নিত করি এবং সেগুলো নিয়ে কাজ করি। নিচের ছবি দেখ:

লাল নোডগুলো নিয়ে আমরা এখন কাজ করবো। রঙিন সবগুলো নোড ভিজিটেড, **এক নোডে ২বার কখনো যাবোনা**। ২,৩,৪ থেকে শর্টেস্ট পথে যাওয়া যায় ৬,৭,৮ এ। সেগুলো ভিজিটেড চিহ্নিত করি:

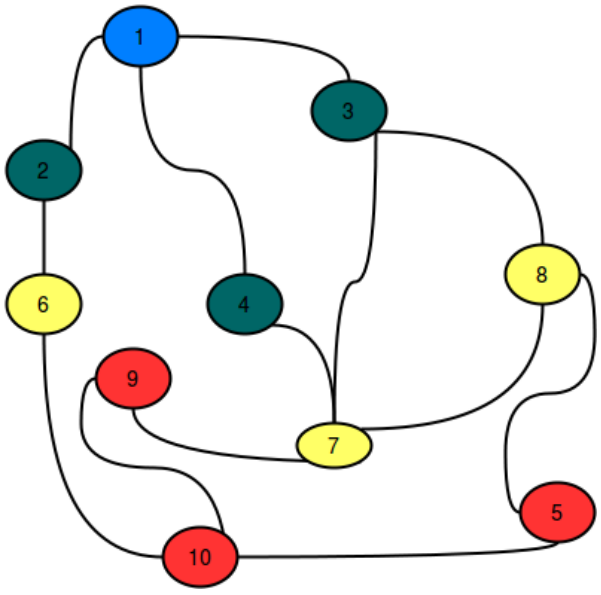


লক্ষ কর যে নোডকে যত নম্বর লেভেলে পাচ্ছি,সোর্স থেকে তার শর্টেস্ট পথের দৈর্ঘ্য ঠিক তত। যেমন ২নম্বর লেভেলে ৮কে পেয়েছি তাই ৮ এর দুরত্ব ২। ছবিগুলোকে একেকটা লেভেলের একেক রং দেয়া হয়েছে। আর লাল নোড দিয়ে বুঝানো হয়েছে আমরা এখন ওগুলো নিয়ে কাজ করছি। আমরা ১০ এ পৌছাইনি তাই পরের নোডগুলো ভিজিট করে ফেলি:



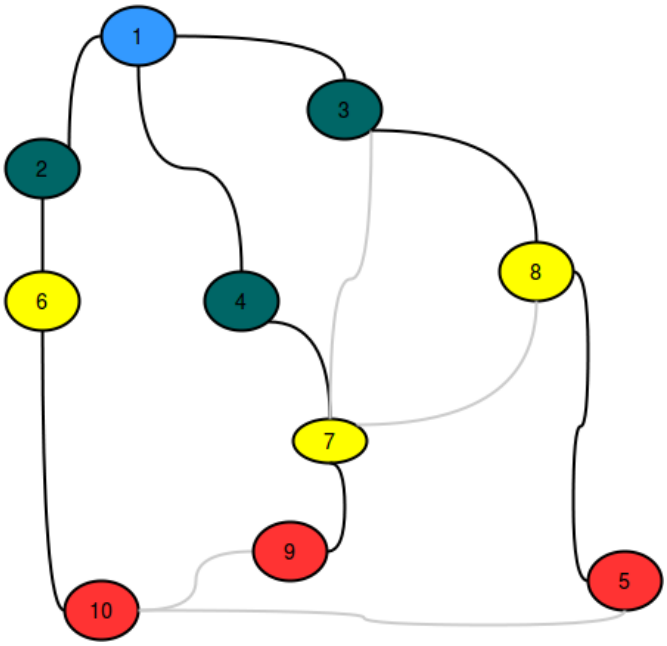
আমরা দেখতে পাচ্ছে ২টি লেভেল পার হয়ে ৩ নম্বর লেভেলে আমরা ১০ কে পাচ্ছি। তাহলে ১০ এর শর্টেস্ট পথ ৩। লেভেল বাই লেভেল গ্রাফটাকে সার্চ করে আমরা শর্টেস্ট পথ বের করলাম। যেসব এজ গুলো আমরা ব্যবহার করিনি সেগুলোকে বাদ দিয়ে ছবিটিকে নিচের মত করে আকতে পারি:

যেসব এজ ব্যবহার করিনি সেগুলো হালকা করে দিয়েছি,এই এজ গুলো বাদ দিলে গ্রাফটি একটি ট্রি হয়ে যায়। এই ট্রি টাকে বলা হয় বিএফএস ট্রি।



তারমানে আমাদের কাজ গুলো সোর্স থেকে লেভেল ১ নোডগুলোতে যাওয়া, তারপর লেভেল ১ এর নোডগুলো থেকে লেভেল ২ নোডগুলো খুজে বের করা, এভাবে যতক্ষন না গন্তব্যে পৌছে যাচ্ছি অথবা সব নোড ভিজিট করা শেষ হয়ে গিয়েছে ততক্ষণ কাজ চলতে থাকবে।

কিউ ডাটা স্ট্রাকচারটার সাথে আশা করি সবাই পরিচিত। কিউ হলো হুবুহু বাসের লাইনের মতো ডাটা স্ট্রাকচার। যখন একটা সংখ্যা কিউতে যোগ করা হয় তখন সেটা আগের সবগুলো সংখ্যার পিছে গিয়ে দাড়ায়, যখন কোন একটা সংখ্যা বের করে ফেলা হয় তখন সবার প্রথমের সংখ্যাটা নেয়া হয়। একে বলা ফার্স্ট ইন ফার্স্ট আউট। আমরা বিএফএস এ কিউ কাজে লাগাতে পারি। লেভেল ১ থেকে যখন কয়েকটা নতুন লেভেল ২ নোড পাবো সেগুলোকে কিউতে বা লাইনে অপেক্ষা করিয়ে রাখবো, আর সবসময় প্রথম নোডটা নিয়ে কাজ করবো। তাহলে বড় লেভেলের নোডগুলো সবসময় পিছের দিকে থাকবে, আমরা ছোট লেভেলগুলো নিয়ে কাজ করতে করতে আগাবো। উপরের গ্রাফের জন্য এটা আমরা সিমুলেট করে দেখি:



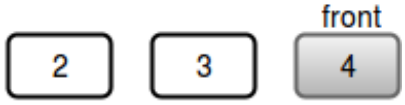
প্রথমে কিউতে সোর্স পুশ করবো:

১ এর লেভেল হবে ০ বা লেভেল[১] = ০। এবার বিএফএস শুরু করবো।

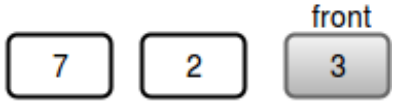
প্রথমে কিউ এর সবার সামনের নোডটাকে নিয়ে কাজ করবো। সবার সামনে আছে ১, সেখান থেকে যাওয়া যায় ৪, ৩, ২ এ। ৪, ৩, ২ এ এসেছি ১ থেকে, তাহলে লেভেল[৪] = লেভেল[১] + ১ = ১, লেভেল[৩] = লেভেল[১] + ১ = ১, লেভেল[২] = লেভেল[১] + ১ = ১।

১ কে ফেল দিয়ে এদেরকে কিউতে পুশ করে রাখি:

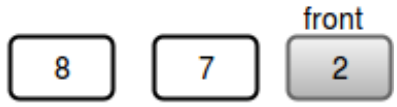
এবার ৪ নিয়ে কাজ করি। ৪ থেকে যাওয়া যায় ৭ এ। তাহলে আমরা বলতে পারি লেভেল[৭]=লেভেল[৪]+১=২। ৪ কে ফেলে দিয়ে ৭ কে কিউতে পুশ করি:



৩ থেকে ৭,৮ এ যাওয়া যায়। ৭ কে এরই মধ্যে নিয়েছি, শুধু ৮ পুশ করতে হবে। লেভেল[৮]=লেভেল[৩]+১=২।



এভাবে যতক্ষণনা কিউ খালি হচ্ছে ততক্ষণ কাজ চলতে থাকবে। লেভেল[] অ্যারের মধ্যে আমরা পেয়ে যাবো সোর্স থেকে সবগুলো নোডের দূরত্ব!



সুডোকোড:

```
1     1  procedure BFS(G,source):
2     2     Q=queue(), level[]=infinity
3     3     Q.enqueue(source)
4     4     level[source]=0
5     5     while Q is not empty
6     6        u ← Q.pop()
7     7        for all edges from u to v in G.adjacentEdges(v) do
8     8           if level[v] = infinity:
9     9                 level[v] = level[u] + 1;
10    10                 Q.enqueue(v)
11    11          end if
12    12       end for
13    13      end while
14    14.   Return distance;
```

ঠিক যেভাবে সিমুলেট করেছি সেভাবেই কোডটা লিখেছি, আশা করি বুঝতে সমস্যা হচ্ছেনা।

শুধু পাথের দৈর্ঘ্য যথেষ্ট না, পাথটাও দরকার হতে পারে। লক্ষ্য করো আমরা u থেকে v তে যাবার সময় $parent[v]=u$ করে দিচ্ছি। আমরা প্রতিটা নোডের জন্য জানি কোন নোড থেকে সেই নোডে এসেছি। তাহলে আমরা যে নোডের জন্য পাথ বের করতে চাই সেই নোড থেকে তার প্যারেন্ট নোডে যেতে থাকবো যতক্ষণনা সোর্সে পৌছে যাই। খুবই সহজ কাজ, পাথ বের করার কোড করা তোমার উপর ছেড়ে দিলাম।

কমপ্লেক্সিটি:
প্রতিটা নোডে একবার করে গিয়েছি, প্রতিটা এজ এ একবার গিয়েছি। তাহলে কমপ্লেক্সিটি হবে $O(V+E)$ যেখানে V হলো নোড সংখ্যা এবং E হলো এজ সংখ্যা।

কখনো কখনো ২-ডি গ্রিডে বিএফএস চালানো লাগতে পারে। যেমন একটা দাবার বোর্ডে একটি ঘোড়া আর একটা রাজা আছে। ঘোড়াটা মিনিমাম কয়টা মুভে রাজার ঘরে পৌছাতে পারবে? অথবা একটা ২-ডি অ্যারেতে কিছু সেল ব্লক করে দেয়া হয়েছে, এখন কোনো সেল থেকে আরেকটি সেলে মিনিমাম মুভে পৌছাতে হবে, প্রতি মুভে শুধুমাত্র সামনে-পিছে-বামে-ডানে যাওয়া যায়। আগে নোডকে আমরা প্রকাশ করছিলাম একটা মাত্র সংখ্যা দিয়ে, এখন নোডকে প্রকাশ করতে হবে দুটি সংখ্যা দিয়ে, রো(row) নাম্বার, এবং কলাম নাম্বার। তাহলে আমরা নোড রিপ্রেজেন্ট করার জন্য সি তে একটা স্ট্রাকচার বানিয়ে নিতে পারি এরকম:

```
struct node{int r,c;};
```

অথবা আমরা সি++ এর “পেয়ার” ব্যবহার করতে পারি।

> pair<int,int>

এ ক্ষেত্রে ভিজিটেড, প্যারেন্ট, লেভেল অ্যারেগুলো হবে ২ ডিমেনশনের, যেমন $visited[10][10]$ ইত্যাদি। কিউতে নোডের বদলে স্ট্রাকচার পুশ করবো। আর কোন একটা ঘর থেকে অন্য ঘরে যাবার সময় চেক করতে হবে বোর্ডের বাইরে চলে যাচ্ছে কিনা। একটা স্যাম্পল সি++ কোড দেখি:

C++

```
#define pii pair<int,int>
int fx[]={1,-1,0,0}; //ডিরেকশন অ্যারে
int fy[]={0,0,1,-1};
int cell[100][100]; //cell[x][y] যদি -১ হয় তাহলে সেলটা ব্লক
int d[100][100],vis[100][100]; //d means destination from source.
int row,col;
void bfs(int sx,int sy) //Source node is in [sx][sy] cell.
{
memset(vis,0,sizeof vis);
vis[sx][sy]=1;
queue<pii>q; //A queue containing STL pairs
q.push(pii(sx,sy));
while(!q.empty())
{
pii top=q.front(); q.pop();
for(int k=0;k<4;k++)
{
int tx=top.uu+fx[k];
int ty=top.vv+fy[k]; //Neighbor cell [tx][ty]
if(tx>=0 and tx<row and ty>=0 and ty<col and cell[tx][ty]!=-1 and vis[tx][ty]==0) //Check
if the neighbor is valid and not visited before.
{
vis[tx][ty]=1;
d[tx][ty]=d[top.uu][top.vv]+1;
q.push(pii(tx,ty)); //Pushing a new pair in the queue
}
}
}
}
```

তুমি যদি ডিরেকশন অ্যারের ব্যাপারটা না বুঝো তাহলে এই লেখাটা পড়লে আরো কিছু ডিটেইলস জানতে পারবে।

বিএফএস শুধুমাত্র আন-ওয়েটেড গ্রাফে কাজ করে, ওয়েটেড গ্রাফে শর্টেস্ট পাথ বের করতে ডায়াক্সট্রা অ্যালগোরিদম ব্যবহার করতে পারো। গ্রাফে নেগেটিভ সাইকেল থাকলে বেলম্যান ফোর্ড ব্যবহার করতে হবে।

প্র্যাকটিসের জন্য প্রবলেম:
Bicoloring(Bipartite checking)
A Node Too Far(Shortest path)
Risk(Shortest path)
Bombs! NO they are Mines!!(bfs in 2d grid)
Knight Moves(bfs in 2d grid)
We Ship Cheap(Printing path)
Word Transformation(strings)

পরের পর্ব

গ্রাফ থিওরিতে হাতেখড়ি ৫: মিনিমাম স্প্যানিং ট্রি(প্রিম অ্যালগোরিদম)

shafaetsplanet.com/

শাফায়েত

August 4, 2011

একটি গ্রাফ থেকে কয়েকটি নোড আর এজ নিয়ে নতুন একটি গ্রাফ তৈরি করা হলে সেটাকে বলা হয় **সাবগ্রাফ**। স্প্যানিং ট্রি হলো এমন একটি সাবগ্রাফ যেটায়:

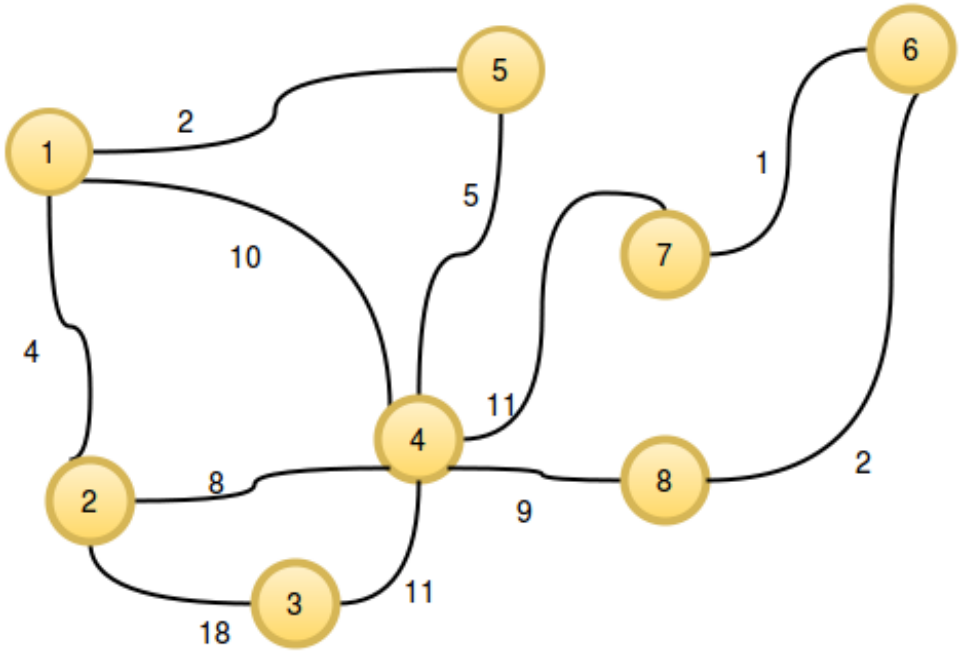
> * মূল গ্রাফের সবগুলো নোড আছে।
> * সাবগ্রাফটি একটি ট্রি। ট্রিতে কখনো সাইকেল থাকেনা,এজ থাকে $n-1$ টি যেখানে n হলো নোড সংখ্যা।

একটি গ্রাফের অনেকগুলো স্প্যানিং ট্রি থাকতে পারে,যে ট্রি এর এজ গুলোর কস্ট/ওয়েট এর যোগফল সব থেকে কম সেটাই মিনিমাম স্প্যানিং ট্রি। আমরা এই লেখায় প্রিম অ্যালগোরিদমের সাহায্যে মিনিমাম স্প্যানিং ট্রি বের করা শিখবো।

মনে করি নিচের গ্রাফের প্রতিটি নোড হলো একটি করে বাড়ি। আমাদের বাড়িগুলোর মধ্যে টেলিফোন লাইন বসাতে হবে। আমরা চাই সবথেকে কম খরচে লাইন বসাতে। এজ গুলোর ওয়েট লাইন বসানোর খরচ নির্দেশ করে:

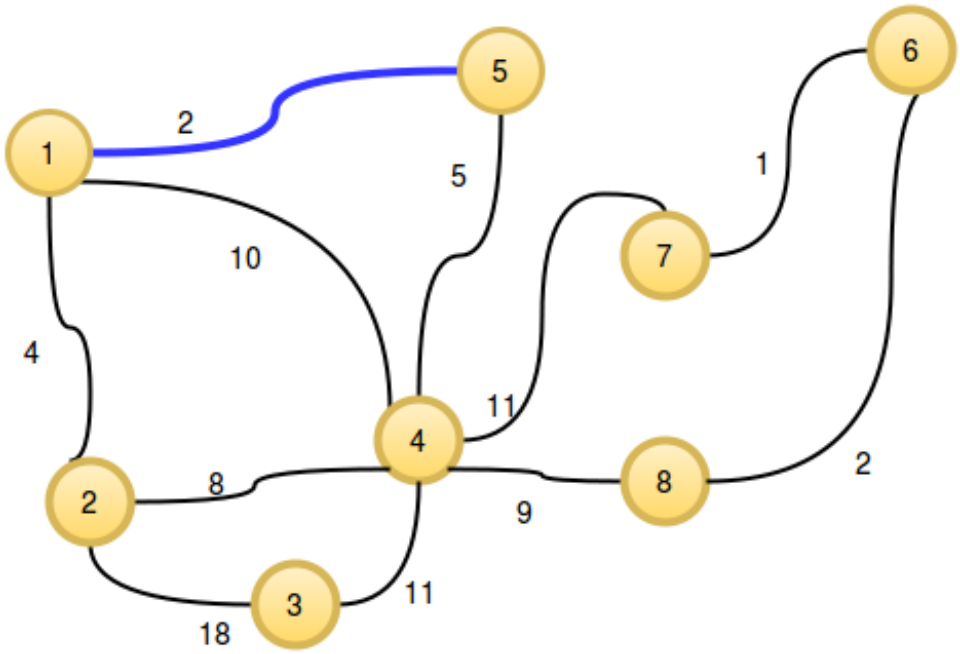
আমরা অনেক ভাবে লাইন বসাতে পারতাম। ছবিতে লাল এজ দিয়ে টেলিফোন লাইন বসানোর একটি উপায় দেখানো হয়েছে। টেলিফোন লাইনগুলো একটি সাবগ্রাফ তৈরি করেছে যেটায় অবশ্যই $n-1$ টি এজ আছে,কোনো সাইকেল নেই কারণ অতিরিক্ত এজ বসালে আমাদের খরচ বাড়বে,কোনো লাভ হবেনা। মিনিমাম স্প্যানিং ট্রি বের করার সময় আমরা এমন ভাবে এজগুলো নিবো যেন তাদের এজ এর যোগফল মিনিমাইজ হয়।

এখন নিচের গ্রাফ থেকে কিভাবে আমরা মিনিমাম স্প্যানিং ট্রি বের করব?

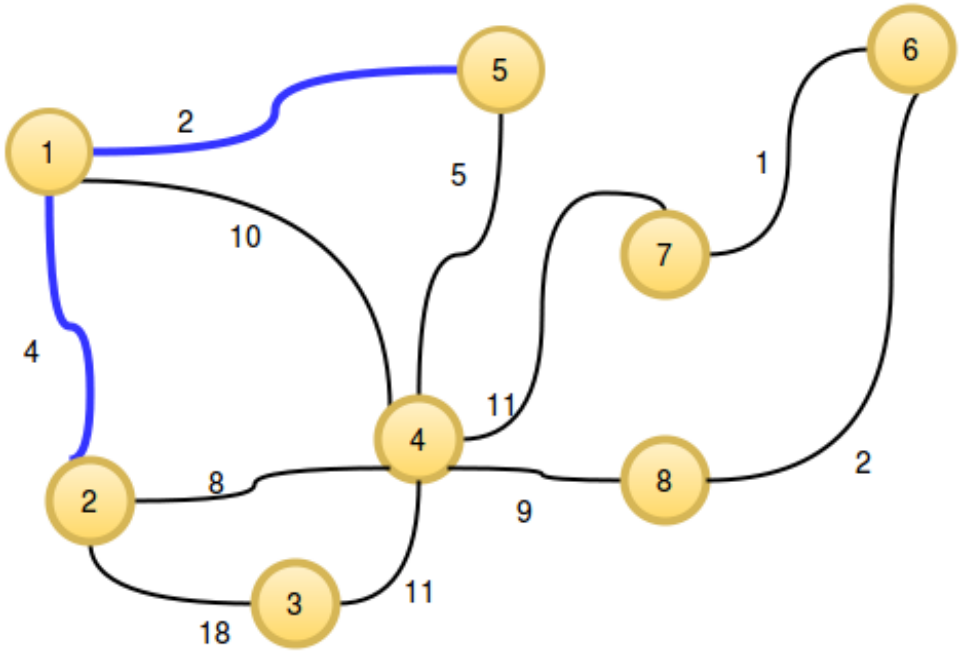


গ্রিডি(greedy) অ্যাপ্রোচে খুব সহজে মিনিমাম স্প্যানিং ট্রি বের করা যায়। আমরা এখন প্রিমস অ্যালগোরিদম কিভাবে কাজ করে দেখব। তুমি যদি আগে ক্রুসকাল শিখতে চাও তাহলেও সমস্যা নেই, সরাসরি পরের পর্বে চলে যেতে পারো।

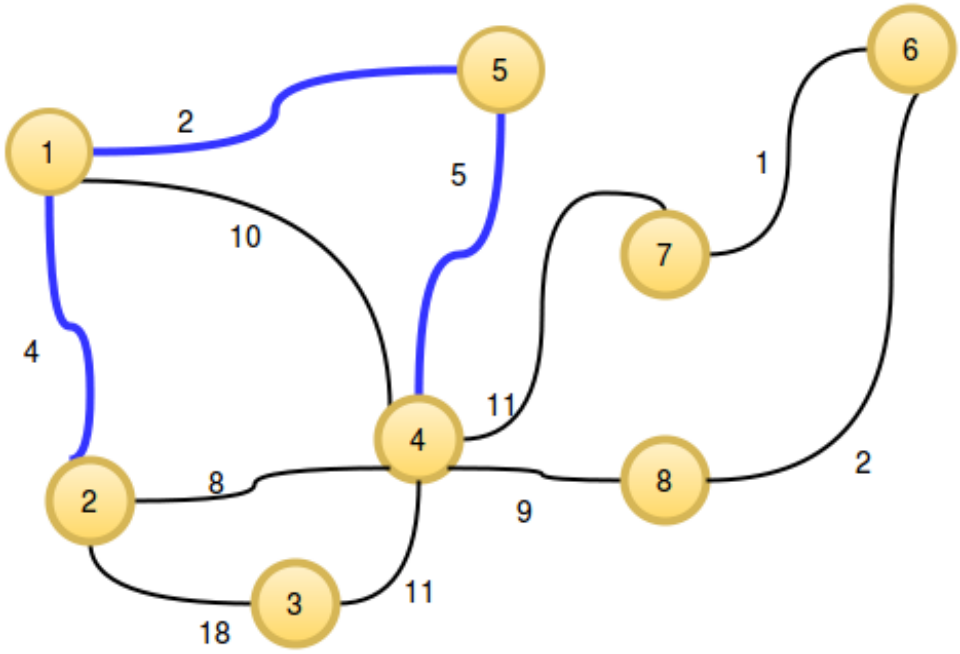
আমরা প্রথমে যেকোনো একটি সোর্স নোড নিব। ধরি সোর্স হলো ১। ১ থেকে যতগুলো এজ আছে সেগুলোর মিনিমাম টিকে আমরা সাবগ্রাফে যোগ করব। নিচের ছবিতে নীল এজ দিয়ে বুঝানো হচ্ছে এজটি সাবগ্রাফে যুক্ত করা হয়েছে:



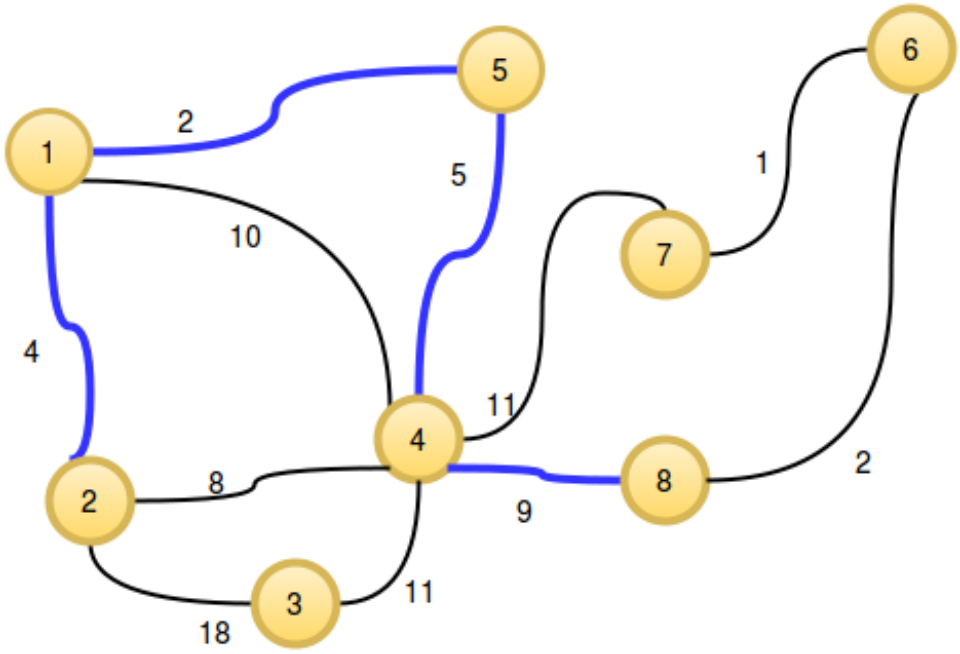
এবার সোর্স ১ এবং ৫ নম্বর নোড থেকে মোট যত এজ আছে(আগের এজগুলো সহ) তাদের মধ্যে মিনিমাম টি নিব:



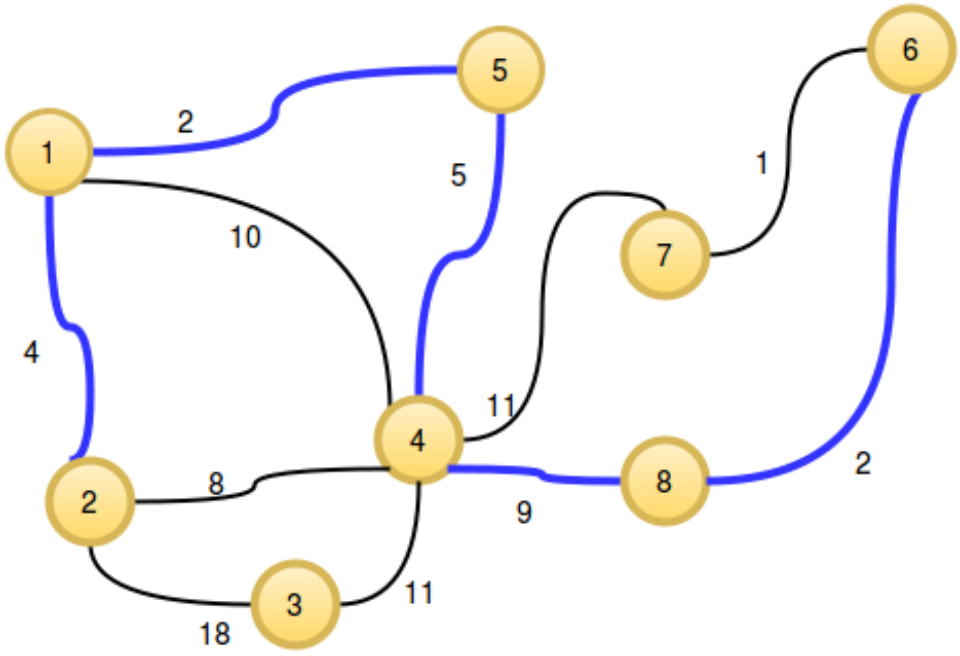
এবার নিব ১,২ এবং ৫ নম্বর নোড থেকে মোট যত এজ আছে(আগের এজগুলো সহ) তাদের মধ্যে মিনিমাম:



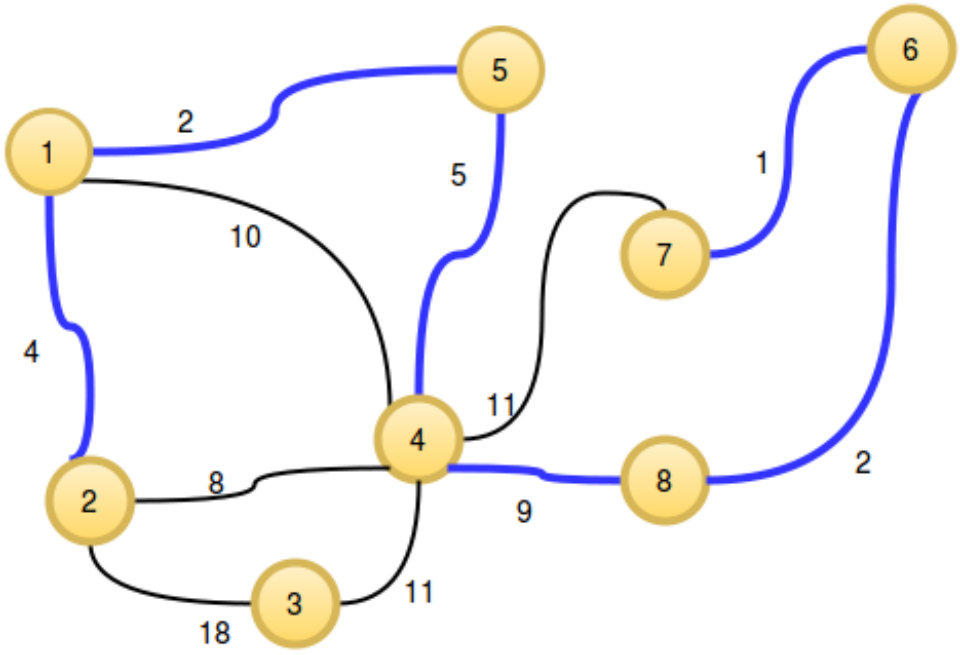
পরের ধাপটি গুরুত্বপূর্ণ। ১,২,৫,৪ থেকে যত এজ আছে তাদের মধ্য মিনিমাম হলো ২-৪, কিন্তু ২ নম্বর নোড এবং ৪ নম্বর নোড দুইটাই অলরেডি সাবগ্রাফের অংশ,তারা আগে থেকেই কানেক্টেড,এদের যোগ করলে সাবগ্রাফে সাইকেল তৈরি হবে,তাই ২-৪ এজটি নিয়ে আমাদের কোনো লাভ হবেনা। **আমরা এমন প্রতিবার এজ নিব যেন নতুন আরেকটি নোড সাবগ্রাফে যুক্ত হয়।** তাহলে ৪-৮ হবে আমাদের পরের চয়েস।



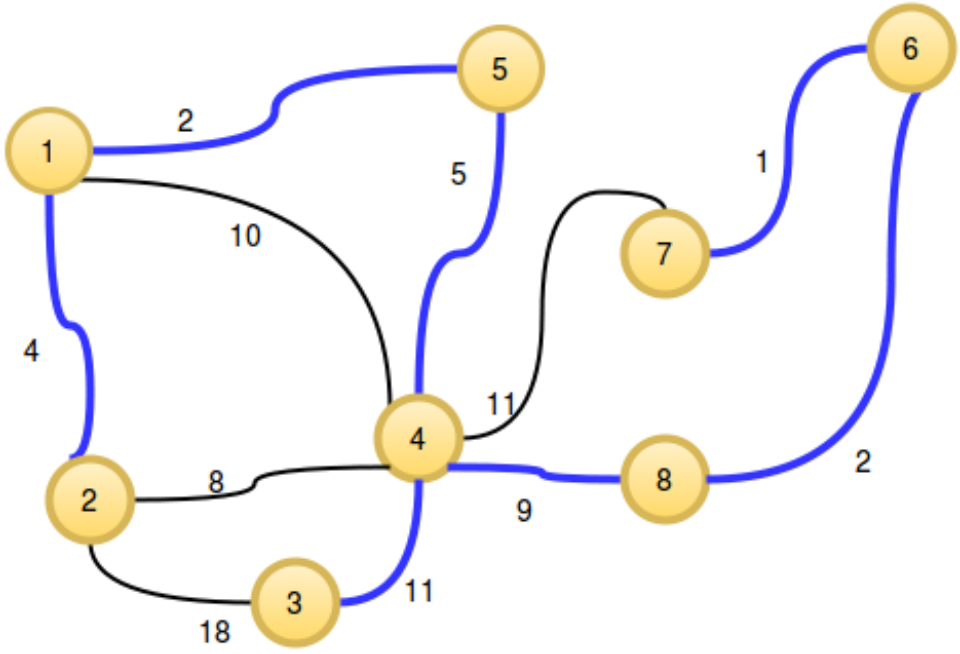
এরপর ৮-৬ যোগ করবো:



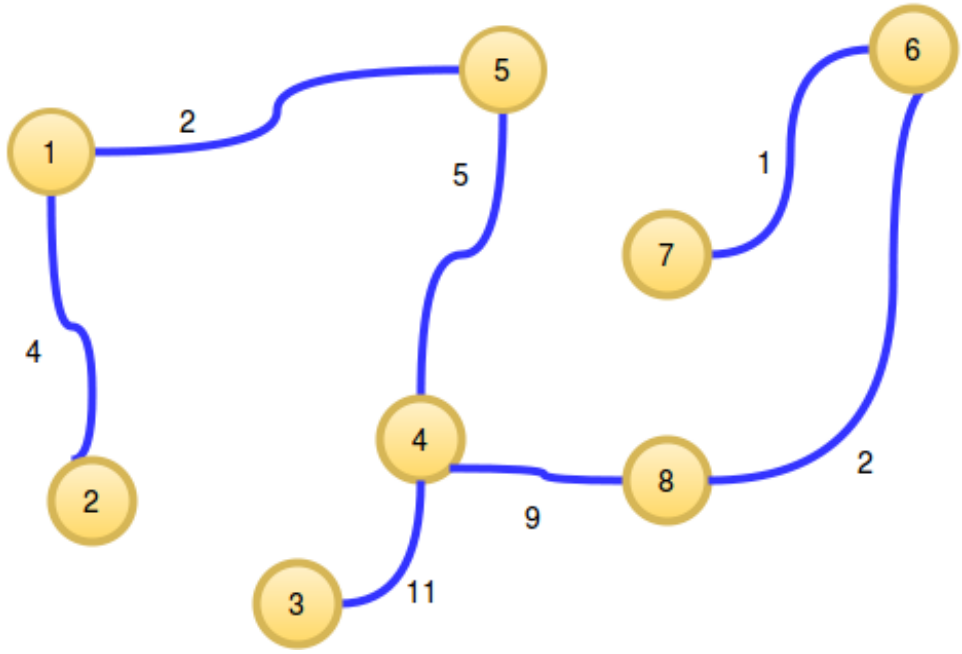
এরপর ৬-৭:



সবশেষে ৪-৩ যোগ করবো:



নীলরং এর এই সাবগ্রাফটাই আমাদের মিনিমাম স্প্যানিং ট্রি। বাকি এজগুলো মুছে দিলে থাকে:



তাহলে টেলিফোন লাইন বসাতো মোট খরচ: ৪ + ২ + ৫ + ১১ + ৯ + ২ + ১ = ৩৪। একটি গ্রাফে এক বা একাধিক মিনিমাম স্প্যানিং ট্রি থাকতে পারে।

আমাদের সুডোকোড হবে এরকম:

```
1  * Input: A non-empty connected weighted graph with vertices V and edges E (the
2  weights can be negative).
3  * Initialize: Vnew = {x}, where x is an arbitrary node (starting point) from V, Enew = {}
4  * Repeat until Vnew = V:
5  o Choose an edge (u, v) with minimal weight such that u is in Vnew and v is not
6  (if there are multiple edges with the same weight, any of them may be picked)
7  o Add v to Vnew, and (u, v) to Enew
   * Output: Vnew and Enew describe a minimal spanning tree
```

এখন মাথায় প্রশ্ন আসতে পারে কি ভাবে প্রিমস অ্যালগোরিদম ইম্প্লিমেন্ট করব? বারবার লুপ চালিয়ে নেইভ অ্যাপ্রোচে কোড লিখলে তোমার কোড টাইম লিমিটের মধ্যে রান না করার সম্ভাবনাই বেশি।

রানটাইম কমাতে প্রায়োরিটি কিউ ব্যবহার করতে পারো। যখন নতুন একটা নোড Vnew তে যোগ করছো তখন সেই নোডের অ্যাডজেসেন্ট সবগুলো এজ প্রায়োরিটি কিউতে ঢুকিয়ে রাখতে হবে। এখন প্রায়োরিটি কিউ থেকে সবথেকে মিনিমাম ওয়েটের এজটা লগারিদম কমপ্লেক্সিটিতে খুজতে পারবে। মোট কমপ্লেক্সিটি হবে $O(ElogE)$। তবে এজের বদলে কিউতে নোড পুশ করে $O(ElogV)$ তে কমপ্লেক্সিটি নামিয়ে আনা যায়, সেটা কিভাবে করা যায় চিন্তা করে বের করো।

অ্যালগোরিদমটা ইমপ্লিমেন্ট করার পর অবশ্যই নিচের সমস্যা গুলো সমাধানের চেষ্টা করবে।

http://uva.onlinejudge.org/external/5/544.html(Straight forward)
http://uva.onlinejudge.org/external/9/908.html
http://uva.onlinejudge.org/external/100/10034.html(Straight forward)
http://uva.onlinejudge.org/external/112/11228.html
http://uva.onlinejudge.org/external/104/10462.html(2nd best mst)
spoj:
http://www.spoj.pl/problems/MST/(Straight forward)

মিনিমাম স্প্যানিং ট্রি বের করার জন্য আরেকটি অ্যালগোরিদম আছে যা ক্রুসকাল অ্যালগোরিদম নামে পরিচিত। পরের পর্বে আমরা সেটা শিখবো।

গ্রাফ থিওরিতে হাতেখড়ি ৬: মিনিমাম স্প্যানিং ট্রি(ক্রুসকাল অ্যালগোরিদম)

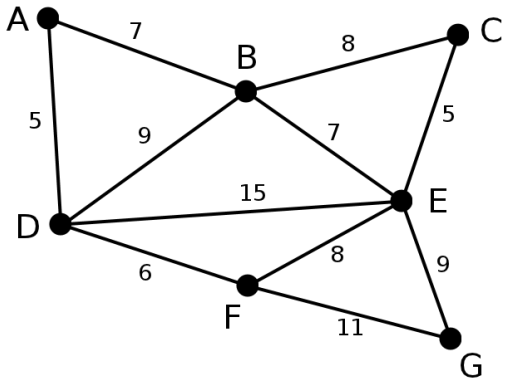
shafaetsplanet.com/

শাফায়েত সেপ্টেম্বর ২৯, ২০১১

আগের পোস্টে আমরা প্রিমস অ্যালগোরিদম ব্যবহার করে mst নির্ণয় করা দেখেছি। mst কাকে বলে সেটাও আগের পোস্টে বলা হয়েছে। এ পোস্টে আমরা দেখবো mst বের করার আরেকটি অ্যালগোরিদম যা ক্রুসকালের অ্যালগোরিদম নামে পরিচিত। এটি mst বের করার সবথেকে সহজ অ্যালগোরিদম। তবে তোমাকে অবশ্যই ডিসজয়েন্ট সেট ডাটা স্ট্রাকচার সম্পর্কে জানতে হবে,না জানলে এই পোস্টটি অবশ্যই দেখে আসো।

এই পোস্টে নিজের আকা ছবি ব্যবহার করবোনা। উইকিতে ক্রুসকাল নিয়ে খুব সুন্দর করে লেখা আছে,আমি ওখানকার ছবিগুলোই ব্যবহার করে সংক্ষেপে অ্যালগোরিদমটা বুঝানোর চেষ্টা করবো।

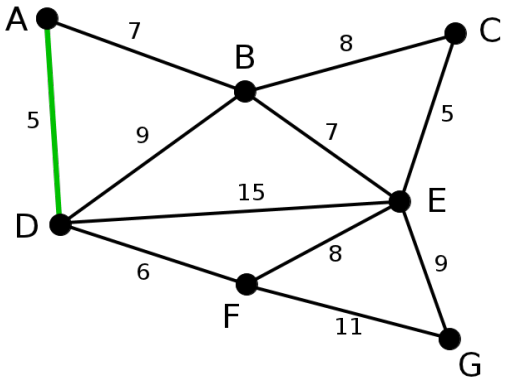
নিচের গ্রাফটি দেখো:



প্রথমে আমাদের ট্রিতে একটি এজও নেই। আমরা মুল গ্রাফের এজগুলোকে cost অনুযায়ী সর্ট করে ফেলবো। সব থেকে কম cost এর এজ আগে নিবো,বেশি cost এর এজ পরে নিবো। দুটি এজের cost সমান হলে যেকোনো একটি আগে নিতে পারি। তারপর একটি করে এজ নিবো আর দেখবো এজের দু প্রান্তের নোডগুলোর মধ্যে ইতোমধ্যে কোনো পথ আছে নাকি,যদি থাকে তাহলে এজটি নিলে সাইকেল তৈরি হবে,তাই এজটা আমরা নিবোনা। বুঝতেই পারছো প্রিমসের মত এটিও একটি ‘গ্রিডি’ অ্যালগোরিদম।

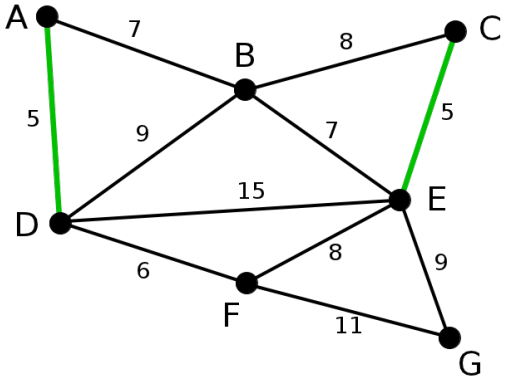
উপরে AD আর CE হলো সবথেকে কম cost এর এজ। আমরা AD কে সাবগ্রাফের অন্তর্ভূক্ত করলাম।

একই ভাবে এরপ CE তারপর DF,AB এবং BE কে যোগ করবো:

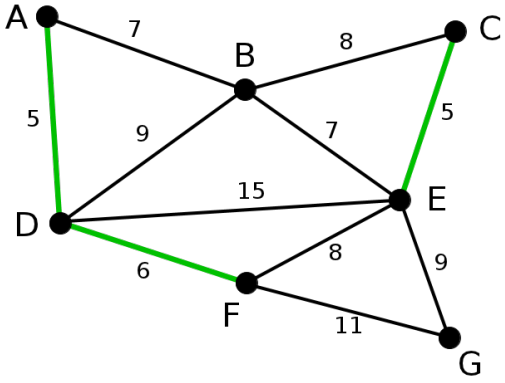


এরপর সবথেকে ছোটে এজ হলো EF, এটাকে আমরা নিতে পারবোনা কারণ EF নিলে একটি সাইকেল তৈরি হয়ে যাবে,E থেকে F তে যাবার রাস্তা আগে থেকেই আছে,তাই এজটি নেয়ার কোনো দরকার নেই। এভাবে BC,DB সহ লাল রঙের এজগুলো বাদ পড়বে কারণ এরা সাইকেল তৈরি করে।

সবশেষে EG যোগ করলে আমরা mst পেয়ে যাবো।



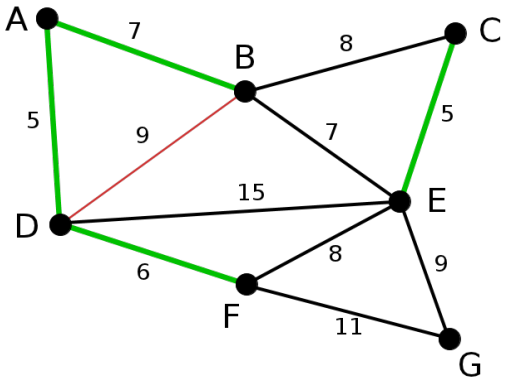
এখন আমরা ইম্প্লিমেনটেশনে আসি। আমাদের প্রথম কাজ হলো সর্ট করা। পরের কাজ হলো একটি একটি এজ নিয়ে চেক করা যে দু প্রান্তের নোড দুটির মধ্য পথ আছে নাকি,অর্থাত তারা একই কম্পোনেন্টের ভিতর আছে নাকি। এটা চেক করতে লাগবে ডিসজয়েন্ট সেট। ডিসজয়েন্ট সেট নিয়ে টিউটোরিয়ালে দেখিয়েছিলাম কিভাবে দুটি নোড একই সাবগ্রাফে আছে নাকি বের করতে হয়। তুমি সেই কাজটিই এখানে করবে। তারপর একই সাবগ্রাফে না থাকলে আগের মত Union ফাংশন কল দিয়ে তাদের একসাথে নিয়ে আসবে আর এজটি একটি ভেক্টর বা অ্যারেতে সেভ করে রাখবে।



নিচে একটা ইমপ্লিমেন্টেশন দিলাম, আশা করি এটা কপি না করে নিজে বুঝে লিখবে:

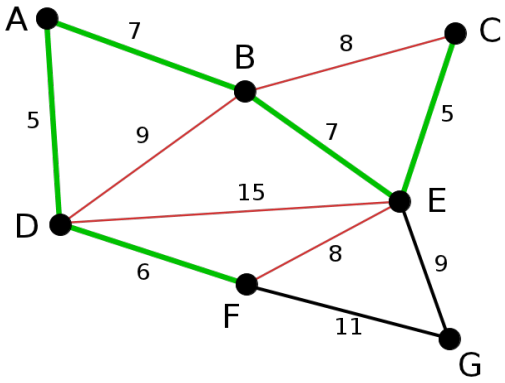
kruskal

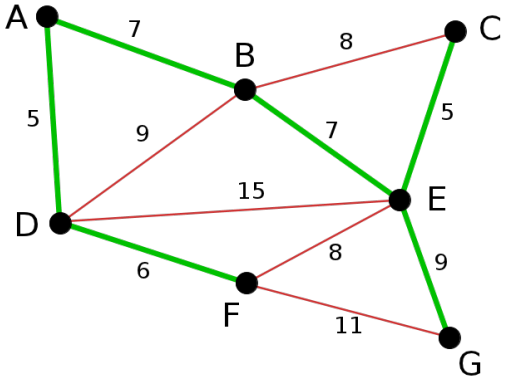
C++




```
1    struct edge {
2        int u, v, w;
3        bool operator<(const edge& p) const
4        {
5            return w < p.w;
6        }
7    };
8    int pr[MAXN];
9    vector<edge> e;
```

```
int find(int r)
{
    return (pr[r] == r) ? r : find(pr[r]);
}
int mst(int n)
{
    sort(e.begin(), e.end());
    for (int i = 1; i <= n; i++)
        pr[i] = i;
    int count = 0, s = 0;
    for (int i = 0; i < (int)e.size(); i++) {
        int u = find(e[i].u);
        int v = find(e[i].v);
        if (u != v) {
            pr[u] = v;
            count++;
            s += e[i].w;
            if (count == n - 1)
                break;
        }
    }
    return s;
}
int main()
{
    // READ("in");
    int n, m;
    cin >> n >> m;
    for (int i = 1; i <= m; i++) {
        int u, v, w;
        cin >> u >> v >> w;
        edge get;
        get.u = u;
        get.v = v;
        get.w = w;
        e.push_back(get);
    }
    cout << mst(n) << endl;
    return 0;
}
```





কমপ্লেক্সিটি অ্যানালাইসিস:

মনে করি E হলো এজ সংখ্যা। এজগুলোকে সর্ট করতে হবে, সেটার কমপ্লেক্সিটি $O(ElogE)$, এরপরে শুধু এজগুলোর উপর লিনিয়ার লুপ চালাতে হবে। তাহলে মোট কমপ্লেক্সিটি $O(ElogE)$।

mst সম্পর্কিত অনেকগুলো সহজ প্রবলেম দিয়েছি প্রিমস এর টিউটোরিয়ালে,ওগুলো সলভ করে প্র্যাকটিস করতে পারো। আরেকটু ভালো প্রবলেম করতে চাইলে দেখো:

২য় সেরা স্প্যানিং ট্রি?
অনেক সময় প্রবলেমে বলা হয় সেকেন্ড বেস্ট MST বের করতে। এটা আমরা ব্রুট ফোর্স দিয়ে বের করতে পারি। MST বের করা পর যে এজগুলো পাবো সেগুলার প্রত্যেকটা একবার করে বাদ দিয়ে নতুন করে MST বের করতে হবে, এভাবে করে যে MST টা মিনিমাম হবে সেটাই সেকেন্ড বেস্ট MST।

http://uva.onlinejudge.org/external/103/10369.html
http://uva.onlinejudge.org/external/117/11733.html

গ্রাফ থিওরিতে হাতেখড়ি ৭:টপোলোজিকাল সর্ট

shafaetsplanet.com/

শাফায়েত অক্টোবর ৬, ২০১১

(অন্যান্য পর্ব)

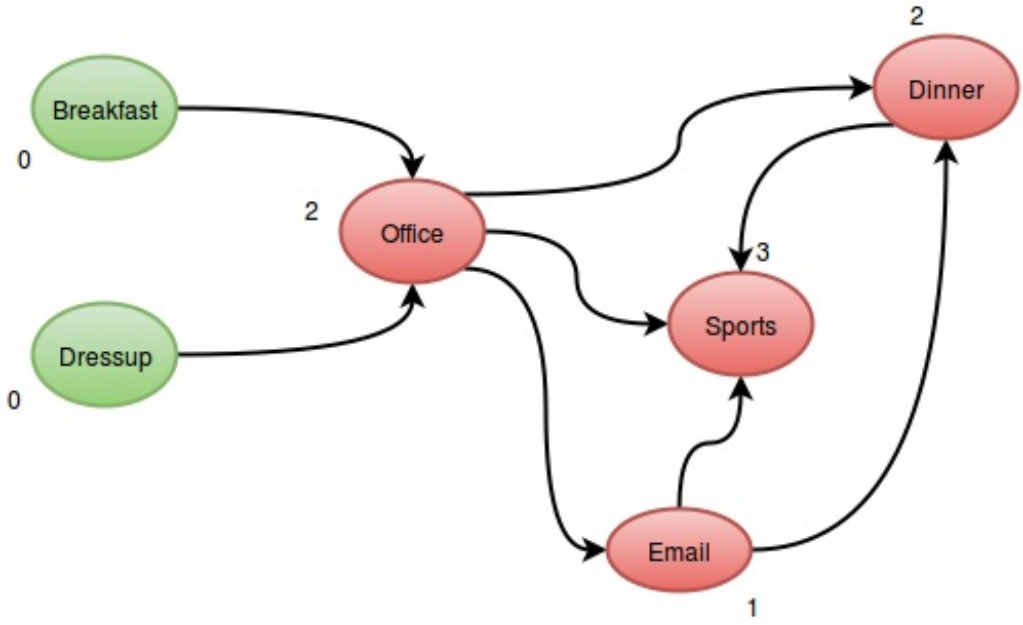
মনে কর তোমার হাতে কিছু কাজের একটা তালিকা আছে,কাজগুলো অবশ্যই শেষ করতে হবে। কাজগুলো হলো অফিসে যাওয়া,সকালে নাস্তা করা,টিভিতে খেলা দেখা,কিছু ই-মেইলের উত্তর দেয়া ,বন্ধুদের সাথে ডিনার করা ইত্যাদি। কাজগুলো কিন্তু আপনি যেকোনো অর্ডারে করতে পারবেনা,কিছু শর্ত মানতে হবে। যেমন অফিসে যাবার আগে নাস্তা করতে হবে,খেলা দেখার আগে অফিসে যেতে হবে,ডিনারে বসার আগে ইমেইলের উত্তর দিতে হবে।
তুমি শর্তগুলোর তালিকা করে ফেললে:

১. সকালের নাস্তা —> অফিস (ক—>খ এর মানে হলো ‘খ’ কাজটি করার আগে ‘ক’ কাজটি করতে হবে)
২. সুট-টাই পড়া —-> অফিস
৩. অফিস —-> ইমেইল
৪. অফিস —-> ডিনার
৫. অফিস —-> খেলা
৬. ইমেইল —> ডিনার
৭. ইমেইল —> খেলা
৮. ডিনার —> খেলা

তুমি এখন কোন কাজ কখন করবে? উল্টাপাল্টা অর্ডারে করলে তোমার কাজ ভন্ডুল হয়ে যাবে,ইমেইল না করে খেলা দেখতে বসলে তুমি ক্লায়েন্ট হারাবে,তাই অর্ডারিং খুব জরুরি।

এটা একটি “টাস্ক শিডিউলিং” প্রবলেম। কোন কাজের পর কোন কাজ করতে হবে সেটা আমাদের বের করতে হবে। অর্থা এটা এক ধরণের সর্টিং যাকে টপোলোজিকাল সর্টিং বলে। আমরা এ টিউটোরিয়ালে এজ সরিয়ে বা ইনডিগ্রী কমিয়ে টপসর্ট বের করবো।

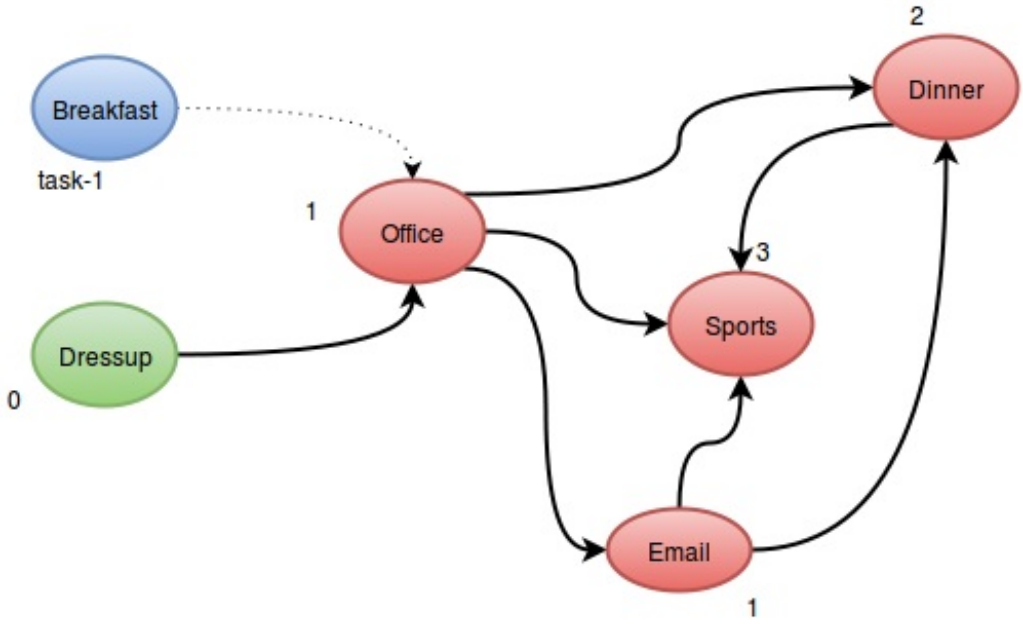
আমরা প্রথমেই সমস্যাটাকে নিচের গ্রাফ দিয়ে মডেলিং করবো:



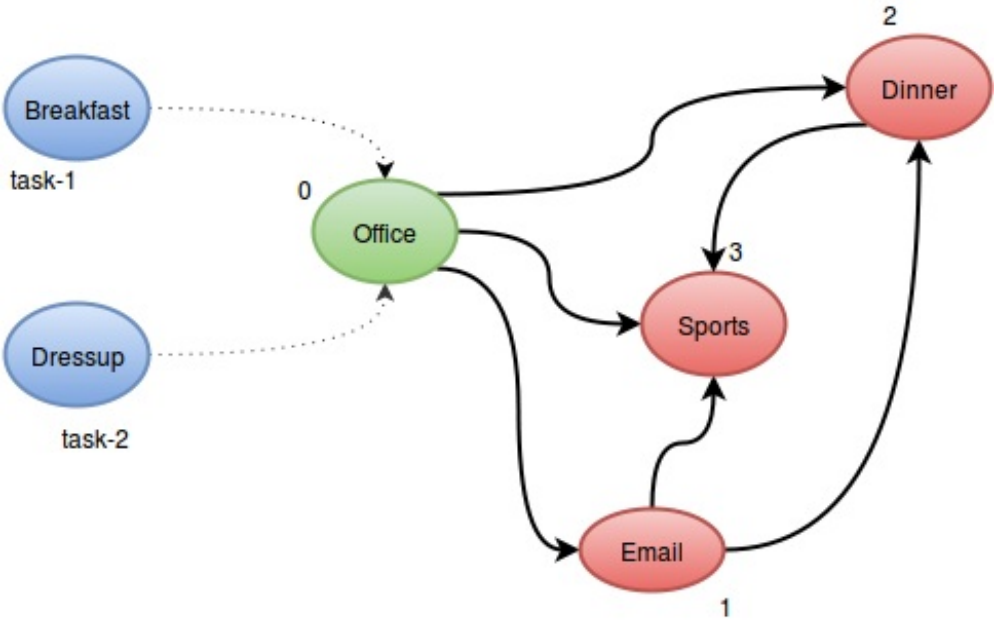
উপরের ছবিতে প্রতিটা কাজ একটি করে নোড দিয়ে দেখানো হয়েছে। ব্রেকফাস্ট থেকে অফিসের দিকে তীরচিহ্ন দিয়ে বুঝানো হচ্ছে যে অফিসে আসার আগে ব্রেকফাস্ট করতে হবে। উপরের ৮টি শর্ত ছবিতে ৮টি ডিরেক্টেড এজ দিয়ে দেখানো হয়েছে।

প্রতিটি নোডের পাশে ছোট করে কিছু সংখ্যা দেখতে পাচ্ছো। যেমন অফিসের সাথে ০,ডিনারের সাথে ২ ইত্যাদি। এগুলো দিয়ে বুঝাচ্ছে একটি কাজ অন্য কয়টি কাজের উপর নির্ভরশীল। যেমন ডিনারের আগে তোমাকে অফিস,ইমেইল এই ২টা কাজ করতে হবে,ডিনার নোডটিতে ২টি তীরচিহ্ন প্রবেশ করেছে,আর আমরা পাশে লিখে দিয়েছি “২”। ঠিক এভাবে ইমেইলের পাশে লেখা হয়েছে ১। এ সংখ্যাগুলোকে indegree বলা হয়।

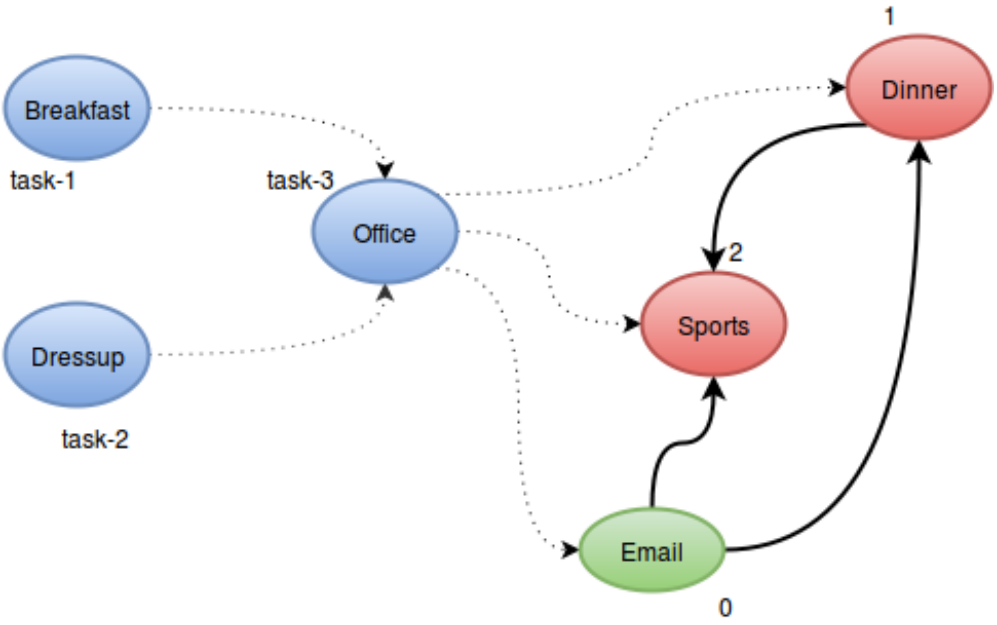
লক্ষ্য কর ব্রেকফাস্ট এবং ড্রেসআপ কোনো কাজের উপর নির্ভরশীল নয়,তাই তাদের পাশে ০ লেখা হয়েছে। তারমানে আমরা এ দুটি কাজের **যেকোনোটা** দিয়ে দিন শুরু করতে পারি। মনে করি তুমি নাস্তা আগে খেতে চাও। নাস্তা খেয়ে নেবার পর যেসব কাজ ব্রেকফাস্টের উপর নির্ভরশীল ছিল তারা আর সেটার উপর নির্ভরশীল থাকলোনা,গ্রাফটা হয়ে গেল এরকম:



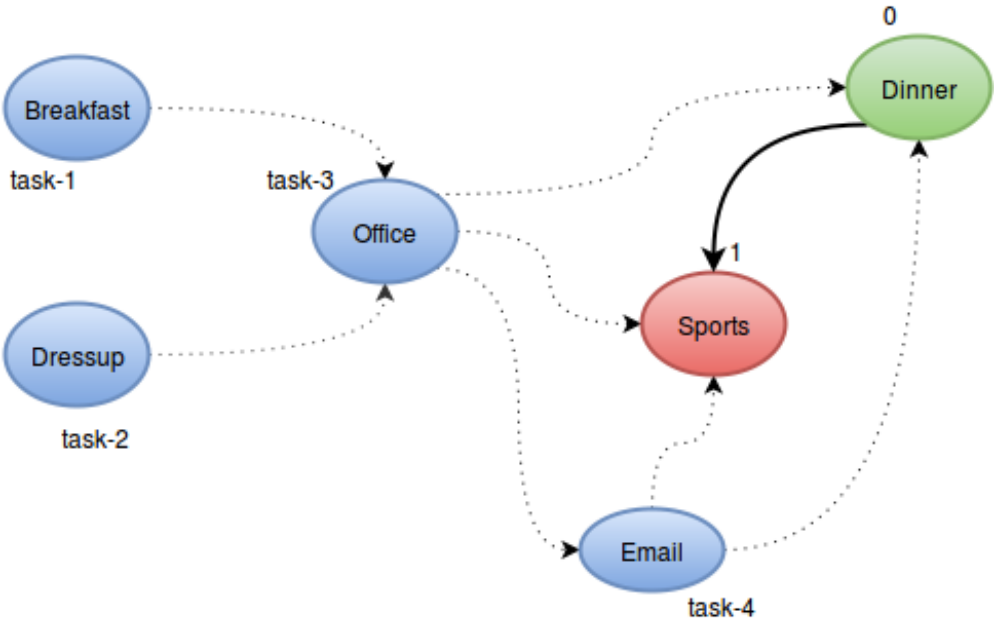
ব্রেকফাস্ট থেকে অফিসের তীরচিহ্ন সরিয়ে দিয়েছি। এখন অফিস আর মাত্র ১টি কাজের উপর নির্ভরশীল(আগে ছিল ২টির উপর)। এবার তোমাকে ড্রেসআপ করতে হবে,কারণ এখন একমাত্র এই কাজটিই কারো উপর নির্ভরশীল নয়:



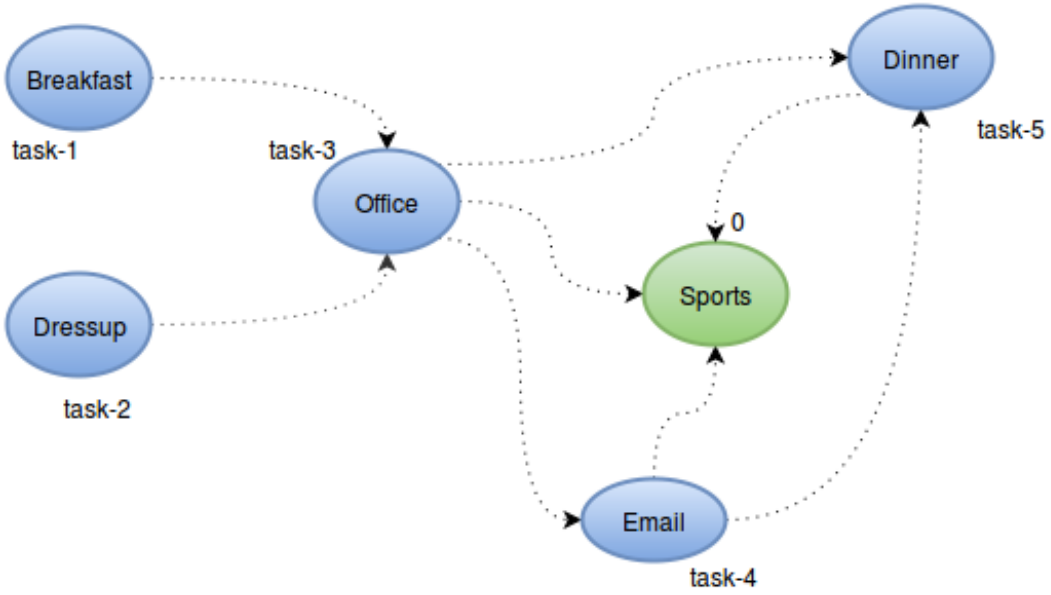
ড্রেসআপ থেকে অফিসের তীরচিহ্ন সরিয়ে দেয়া হয়েছে,আর কোনো কাজ নেই,এবার তুমি অফিসে যাবার জন্য প্রস্তুত। অফিসে যাবার উপর যারা নির্ভরশীল তাদের তীরচিহ্নগুলো এখন সরিয়ে দিতে পারি:



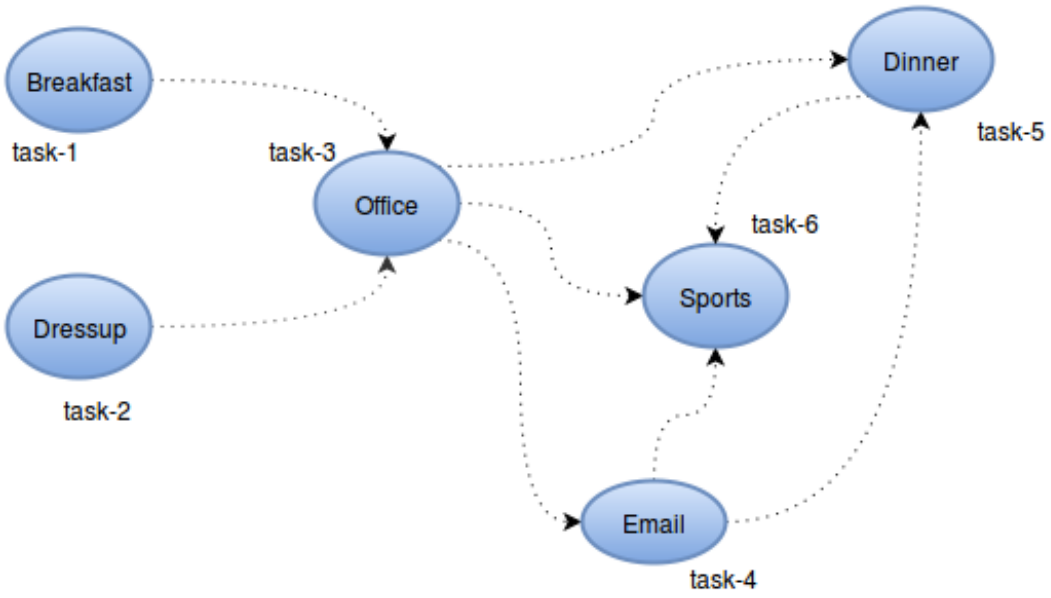
এখন ইমেইল “নোড” এর নির্ভরশীলতা ০ হয়ে গিয়েছে:



এরপর ডিনার:



সবশেষে খেলা দেখতে বসা:



এখন তুমি কাজের অর্ডারিং পেয়ে গিয়েছো,নাস্তা করা,অফিসের পোষাক পড়া,অফিসে যাওয়া,ইমেইল করা,ডিনার করা,খেলা দেখা।

এটাকেই বলা হয় টপোলোজিক্যাল সর্ট(topological sort) বা টপসর্ট। মূলত কাজের অর্ডারিং খুজে বের করতে এই অ্যালগোরিদমটি ব্যবহার করা হয়। কম্পিউটার তার অভ্যন্তরে বিভিন্ন কাজের অর্ডারিং ঠিক করতে টপসর্ট ব্যবহার করে। অনেক রিয়েল লাইফ প্রয়োগ থাকায় টপসর্ট কম্পিউটার সায়েন্সে খুবই গুরুত্বপূর্ণ একটি টপিক। টপসর্ট বের করার আরেকটি পদ্ধতি আছে যার নাম “ডেপথ ফার্স্ট সার্চ”,এটা নিয়ে আলোচনা করেছি এই লেখায়।

উপরের অ্যালগোরিদমটি O(n^2) এ কাজ করে। একটি গ্রাফের অনেকগুলো সর্টেড অর্ডার থাকতে পারে,যেমন উপরের সমস্যায় তুমি নাস্তা খাবার আগে জামা-কাপড় পরতে পারতে। তোমাকে লেক্সিকোগ্রাফিকালি ছোটটা প্রিন্ট করতে বলতে পারে অথবা যেটা ইনপুটে আগে আছে সেটাকে আগে প্রিন্ট করতে বলতে পারে,আশা করি এসব কন্ডিশন সহজে হ্যান্ডল করতে পারবে।

ইমপ্লিমেন্টেশন নিয়ে তেমন কিছু বলার নেই। সবগুলো নোডের indegree বের করবে। তারপর যার indegree শূন্য তার সাথে যাদের এজ আছে তাদের indegree ১ কমিয়ে দিবে,তারপর আবার খুজবে কার indegree এখন শূন্য। এক নোডকে কখনো ২বার নিবেনা। ছবিতে এজ উঠিয়ে দেখিয়েছি বুঝানোর জন্য,তোমার ম্যাট্রিক্স থেকে এজ উঠানোর দরকার নেই,indegree কমালেই চলবে।

অনেক সময় বলতে পারে যতরকম ভাবে topsort করা যায় সবগুলো বের করতে। তখন তোমাকে backtracking এর সাহায্য নিতে হবে।

নিচের সমস্যাগুলো সমাধান করে ফেলো:
http://uva.onlinejudge.org/external/103/10305.html(Easy,straight-forward,special judge)
http://uva.onlinejudge.org/external/110/11060.html(Easy)
http://uva.onlinejudge.org/external/1/124.html(Medium,All possible topsort)
http://uva.onlinejudge.org/external/4/452.html(Medium)

(অন্যান্য পোস্ট)

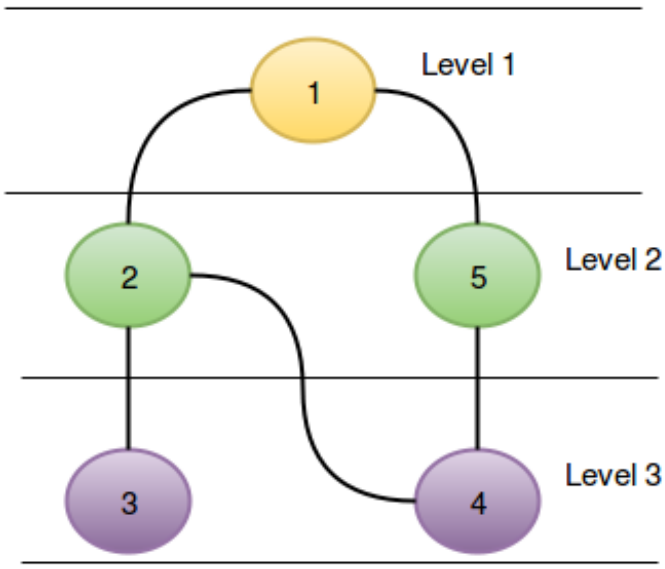
গ্রাফ থিওরিতে হাতেখড়ি ৮: ডেপথ ফার্স্ট সার্চ এবং আবারো টপোলোজিকাল সর্ট

shafaetsplanet.com/

শাফায়েত মার্চ ৮, ২০১২

আগের পর্বগুলো পড়ে থাকলে হয়তো ডেপথ ফার্স্ট সার্চ বা ডিএফএস এতদিনে নিজেই শিখে ফেলেছো। তারপরেও এই টিউটোরিয়ালটি পড়া দরকার কিছু কনসেপ্ট জানতে।

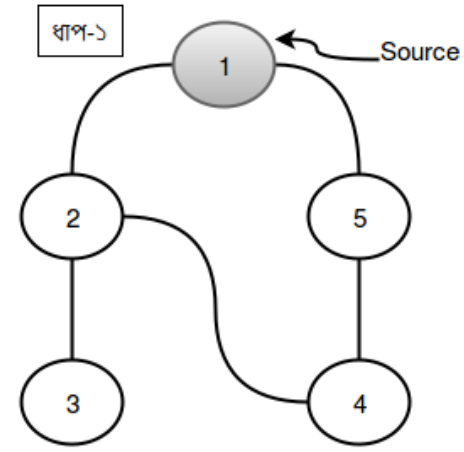
বিএফএস এ আমরা গ্রাফটাকে লেভেল বাই লেভেল সার্চ করেছিলাম,নিচের ছবির মতো করে:

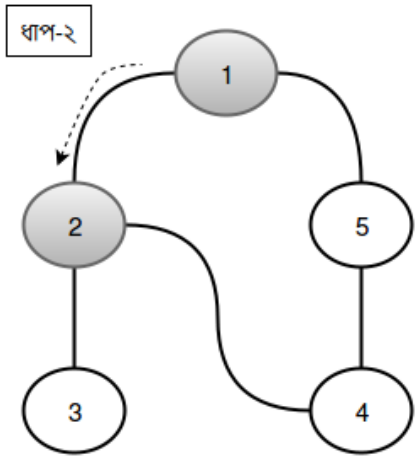


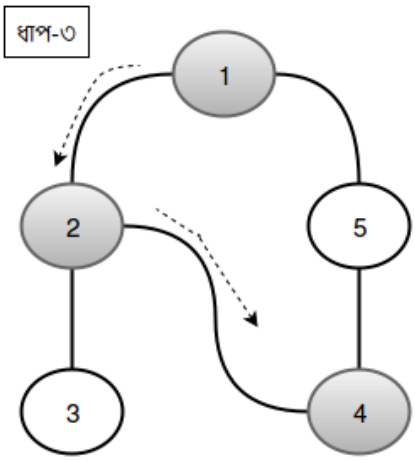
এবার আমরা কোনো নোড পেলে সাথে সাথে সে নোড থেকে আরো গভীরে চলে যেতে থাকবো,যখন আর গভীরে যাওয়া যাবেনা তখন আবার আগের নোডে ফিরে এসে অন্য আরেক দিকে যেত চেষ্টা করবো,এক নোড কখনো ২বার ভিজিট করবোনা। আমরা নোডের ৩টি রং(কালার) দিবো:

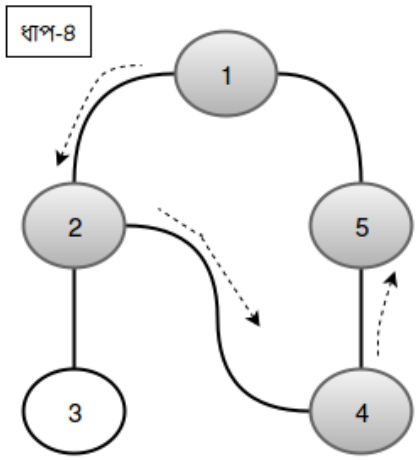
> সাদা নোড= যে নোড এখনো খুজে পাইনি/ভিজিট করিনি।
> গ্রে বা ধুসর নোড= যে নোড ভিজিট করেছি কিন্তু নোডটি থেকে যেসব চাইল্ড নোডে যাওয়া যায় সেগুলো এখনো ভিজিট করে শেষ করিনি,অর্থাত নোডটিকে নিয়ে কাজ চলছে।
> কালো নোড= যে নোডের কাজ সম্পূর্ণ শেষ।

এবার আমরা ধাপগুলো দেখতে পারি:

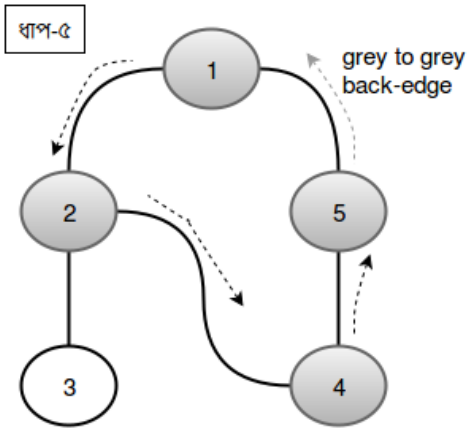

ধাপ-১
1
Source
2
5
3
4


ধাপ-২
1
2
5
3
4

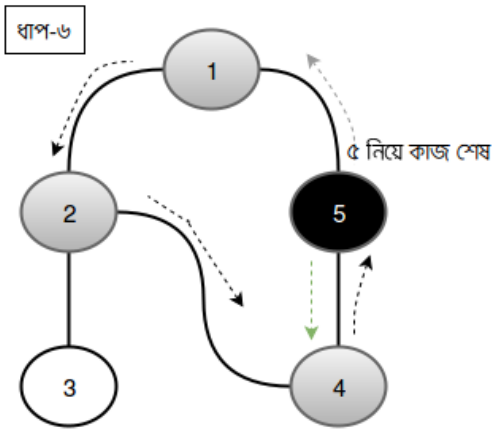

ধাপ-৩
1
2
5
3
4


ধাপ-৪
1
2
5
3
4

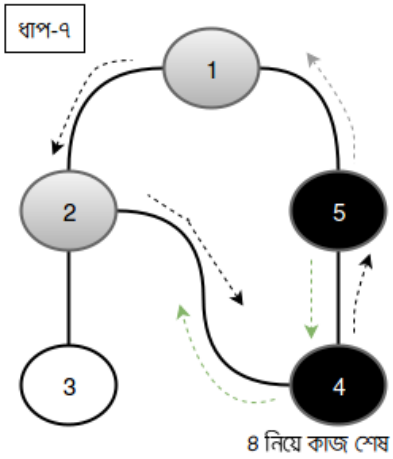
ধাপ-৫



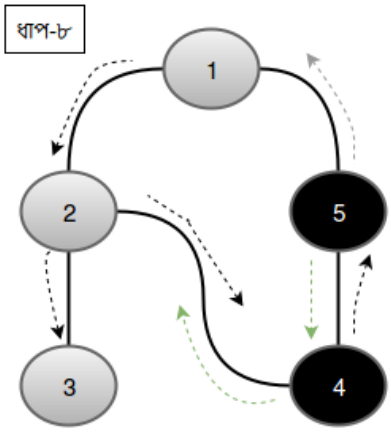
ধাপ-৬

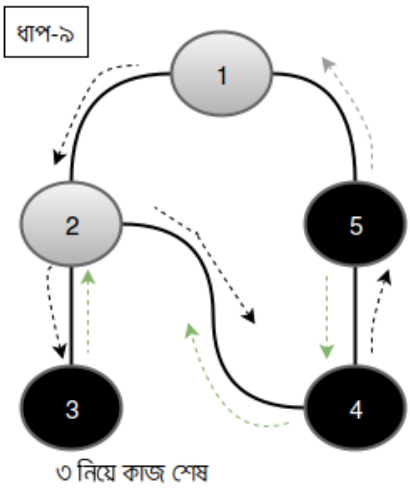


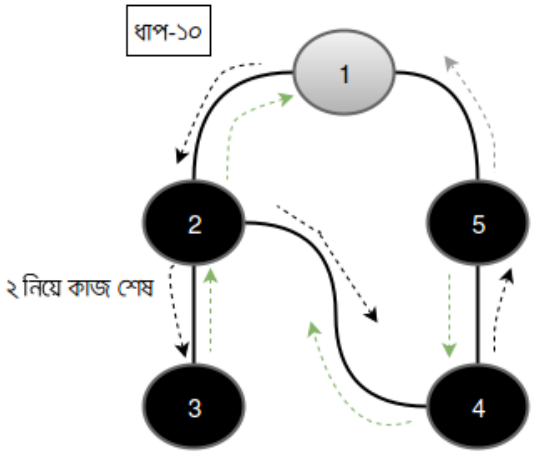
ধাপ-৭

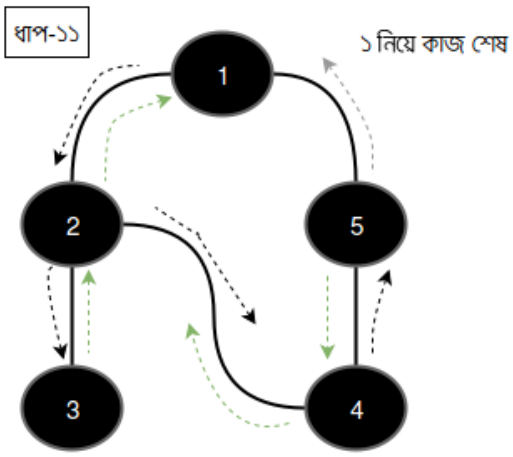


ধাপ-৮





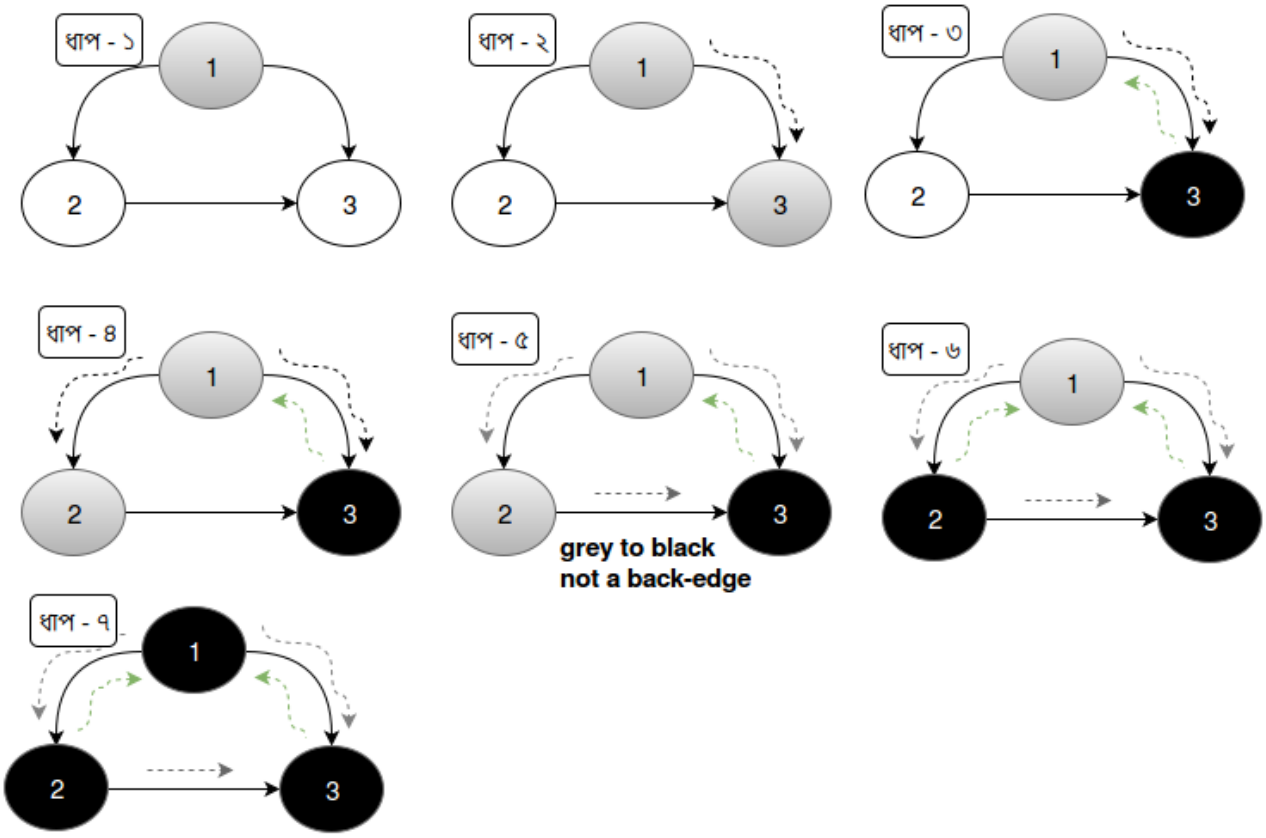




আশা করি ডিএফএস কিভাবে কাজ করে এটা পরিস্কার,খুব সহজ জিনিস এটা। এবার আমরা একটা খুব গুরুত্বপূর্ণ টার্ম শিখবো,সেটা হলো ব্যাকএজ(backedge)। লক্ষ করো ৫-১ কে ব্যাকএজ বলা হয়েছে। এর কারণ হলো তখনও ১ এর কাজ চলছে,৫ থেকে ১ এ যাওয়া মানে এমন একটা নোড ফিরে যাওয়া যাকে নিয়ে কাজ এখনো শেষ হয়নি,তারমানে অবশ্যই গ্রাফে একটি সাইকেল আছে। এ ধরনের এজকে ব্যাকএজ বলে,dfs এ **যদি কোনো সময় একটি গ্রে নোড থেকে আরেকটি গ্রে নোডে যেতে চেষ্টা করে তাহলে সে এজটি ব্যাকএজ এবং গ্রাফে অবশ্যই সাইকেল আছে**। dfs এর সোর্স নোড এবং নোড ভিজিট করার অর্ডার এর উপর নির্ভর করে সাইকেলে যে কোনো এজকে ব্যাকএজ হিসাবে পাওয়া যেতে পারে,যেমন ১ থেকে আগে ২ এ না গিয়ে ৫ এ গেলে পরে ২-১ কে ব্যাকএজ হিসাবে পাওয়া যেতো।

আর যখন আমরা স্বাভাবিক ভাবে গ্রে থেকে সাদা নোডে যাচ্ছি তখন সে এজগুলোকে বলা হয় **ট্রি এজ**। শুধুমাত্র ট্রি এজ গুলো রেখে বাকি এজগুলো মুছে দিলে যে গ্রাফটা থাকে তাকে বলা হয় **ডিএফএস ট্রি**।

আনডিরেক্টেড গ্রাফের ক্ষেত্রে আগে ভিজিট করা কোনো নোডে ফিরে গেলেই সেটা ব্যাকএজ,কালার চেক না করলেও হয়। তবে ডিরেক্টেড গ্রাফের ক্ষেত্রে অবশ্যই করতে হবে। পরের ছবিটা দেখো:



২-৩এর এজটাকে ব্যাকএজ বলা যাচ্ছেনা,কারণ ৩ এর কাজ আগেই শেষ হয়ে গেছে।

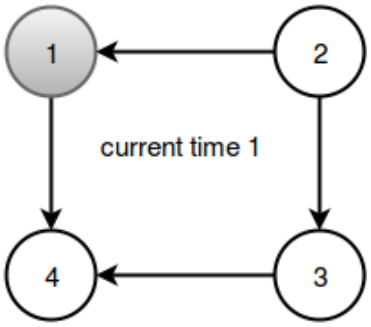
প্রতিটা নোড আর এজ নিয়ে একবার করে করছি, dfs এর কমপ্লেক্সিটি O(V+E)।

আমরা টপোলজিকাল সর্টের সমস্যা সমাধান করেছিলাম বারবার indegree উঠিয়ে। এবার আমরা খুব সহজে dfs দিয়ে এটা করবো। টপোলজিকাল কি সেটা না জানলে আগে এই পোস্টটা পড়ো,তারপর আগাও।
মনে করি আমাদের এজগুলো হলো: ২-১,২-৩,৩-৪,১-৪। অর্থাত ১ নম্বর কাজ করার আগে ২ নম্বরটি করতে হবে ইত্যাদি। এবার আমরা dfs চালানোর সময় একটি স্টপওয়াচ চালু করে দিবো। আর কোনো নোড নিয়ে কাজ শুরু করলে ঘড়ি দেখে নোডটি starting time/discovery time লিখে রাখবো,কাজ শেষ হলো নোডটির finishing time লিখে রাখবো।

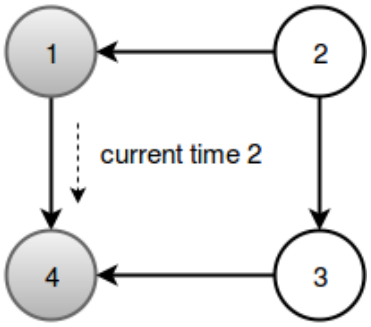
finishing time দেখে আমরা সহজেই টপসর্ট করতে পারি। যে নোডটি সবার আগে আসবে তার finishing time অবশ্যই সবথেকে বেশি হবে,কারণ প্রথম নোডের উপর নির্ভরশীল সব নোড ঘুরে আসার পরে সে নোডের finishing time assign করা হয়। uva 11504-dominos প্রবলেমে আগে নোডগুলোকে finishing time দিয়ে সর্ট করে তারপর আবার dfs চালাতে হয়,প্রবলেমটা চেষ্টা করো।

d[]=discovery time
f[]=finishing time

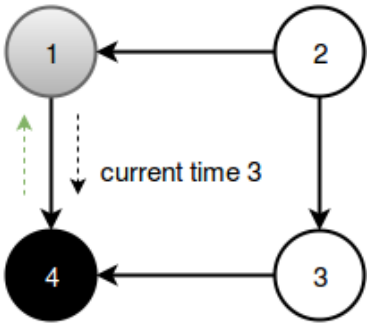
node	d[]	f[]
1	1	null
2	null	null
3	null	null
4	null	null



node	d[]	f[]
1	1	null
2	null	null
3	null	null
4	2	null



node	d[]	f[]
1	1	null
2	null	null
3	null	null
4	2	3



ডিএফএস দিয়ে আমরা যেসব কাজ করি সেগুলোর অনেকগুলোই bfs দিয়ে করতে পারি। bfs এ সাধারণত টাইম কমপ্লেক্সিটি কম হয় তবে dfs কোডিং করতে খুব কম সময় লাগে। একটা সিম্পল dfs এর সুডোকোড এরকম:

```
1     procedure DFS(G, source):
2       U ← source
3       time  ← time+1
4       d[u]  ← time
5       color[u]  ← GREY
6       for all edges from u to v in
7     G.adjacentEdges(v) do
8         if color[v] = WHITE
9          DFS(G,v)
10       end if
11     end for
12     color[u] ← BLACK
13     time ← time+1
14     f[u] ← time
       return
```

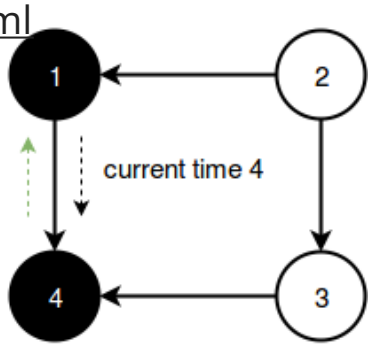
নিচের প্রবলেমগুলো সলভ করতে চেষ্টা করো:

http://uva.onlinejudge.org/external/2/280.html
http://uva.onlinejudge.org/external/115/11518.html

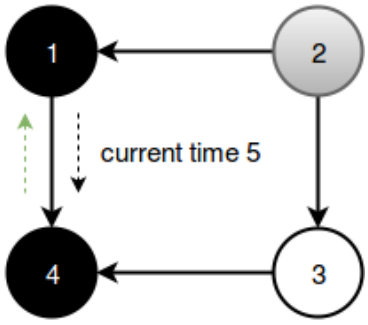
http://uva.onlinejudge.org/external/104/10452.html

এরপরে এই আর্টিকেলটা পড়ে ফেলো বিস্তারিত জানার জন্য,আমার লেখা পড়ে তুমি বেসিকটা শিখতে পারবে,বিস্তারিত জানতে এবং কঠিন প্রবলেম সলভ করতে আরো অনেক কিছু জানতে হবে।

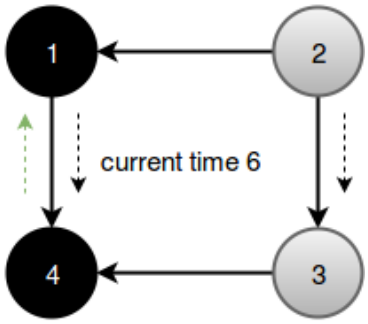
node	d[]	f[]
1	1	4
2	null	null
3	null	null
4	2	3



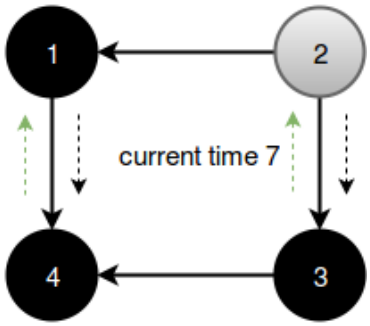
node	d[]	f[]
1	1	4
2	5	null
3	null	null
4	2	3



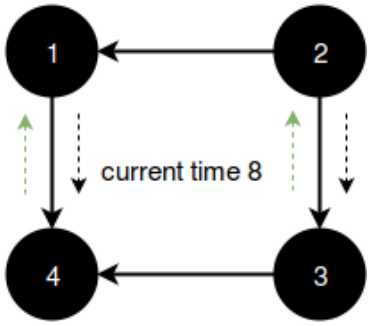
node	d[]	f[]
1	1	4
2	5	null
3	6	null
4	2	3



node	d[]	f[]
1	1	4
2	5	null
3	6	7
4	2	3



node	d[]	f[]
1	1	4
2	5	8
3	6	7
4	2	3



ফিনিশিং টাইম অনুযায়ি সর্ট করে পাই: ২,৩,১,৪

গ্রাফ থিওরিতে হাতেখড়ি-৯ (ডায়াক্সট্রা)

shafaetsplanet.com/

শাফায়েত

April 8, 2013

আমরা শুরুতেই শিখেছি কিভাবে শর্টেস্ট পাথে এক জায়গা থেকে আরেক জায়গায় যেতে হয়। সেজন্য আমরা শিখেছি ব্রেডথ ফার্স্ট সার্চ নামের একটি সার্চিং অ্যালগোরিদম। অ্যালগোরিদমটি চম□কার কিন্তু সমস্যা হলো সে ধরে নেয় প্রতিটি রাস্তা দিয়ে যেতে সমান সময় লাগে, মানে সব এজ এর কস্ট সমান। প্র্যাকটিকাল লাইফে বেশিভাগ ক্ষেত্রেই এটা অচল হয়ে পড়ে, তখন আমাদের দরকার পরে ডায়াক্সট্রা। প্রথমে নাম শুনে আমার ধারণা হয়েছিলো ডায়াক্সট্রা খুবই ভয়ংকর কোনো জিনিস কিন্তু আসলে বিএফএস লেখার মতোই সহজ ডায়াক্সট্রা লেখা, আমি তোমাদের দেখানোর চেষ্টা করবো কিভাবে বিএফএসকে কিছুটা পরিবর্তন করে একটা প্রায়োরিটি কিউ যোগ করে সেটাকে ডায়াক্সট্রা বানিয়ে ফেলা যায়।

ডায়াক্সট্রা শুরু করার আগে আমরা পাথ রিল্যাক্সেশন(relax) নামের একটা ছোট্ট জিনিসের সাথে পরিচিত হই। ধরো সোর্স থেকে প্রতিটা নোডের ডিসটেন্স রাখা হয়েছে $d[]$ অ্যারেতে। যেমন $d[3]$ মানে হলো সোর্স থেকে বিভিন্ন এজ পার হয়ে ৩ এ আসতে মোট $d[3]$ ডিসটেন্স লেগেছে। যদি ডিসটেন্স জানা না থাকে তাহলে ইনফিনিটি অর্থা□ অনেক বড় একটা মান রেখে দিবো। আর $cost[u][v]$ তে রাখা আছে $u – v$ এজ এর cost।

ধরো তুমি বিভিন্ন জায়গা ঘুরে ফার্মগেট থেকে টিএসসি তে গেলে ১০ মিনিটে, আবার ফার্মগেট থেকে কার্জন হলে গেলে ২৫ মিনিটে। তাহলে তোমার কাছে ইনফরমেশন আছে:

> d[টিএসসি] = ১০, d[কার্জনহল] = ২৫

এখন তুমি দেখলে টিএসসি থেকে ৭ মিনিটে কার্জনে চলে যাওয়া যায়,

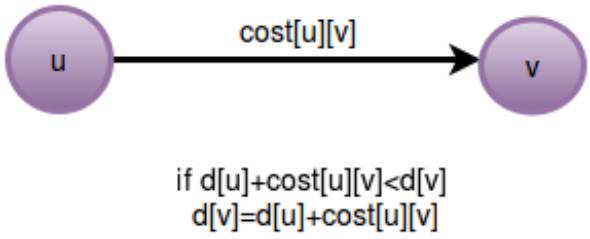
> cost[টিএসসি][কার্জন] = ৭

তাহলে তুমি ২৫ মিনিটের জায়গায় মাত্র ১০ + ৭ = ১৭ মিনিটে কার্জনহলে যেতে পারবে। যেহেতু তুমি দেখেছো:

> d[টিএসসি] + cost[টিএসসি][কার্জন] < d[কার্জনহল]

তাই তুমি এই নতুন রাস্তা দিয়ে কার্জন হলে গিয়ে d[কার্জনহল] = d[টিএসসি] + cost[টিএসসি][কার্জন] বানিয়ে দিতেই পারো!!

উপরের ছবিটা সেটাই বলছে। আমরা u থেকে v তে যাবো যদি **d[u] + cost[u][v] < d[v]** হয়। আর $d[v]$ কে আপডেট করে $d[v] = d[u] + cost[u][v]$ বানিয়ে দিবো।



if d[u]+cost[u][v]<d[v]
d[v]=d[u]+cost[u][v]

ভবিষ্যতে যদি কার্জনহলে অন্য রাস্তা দিয়ে আরো কম সময়ে যেতে পারি তখন সেই রাস্তা এভাবে কম্পেয়ার করে আপডেট করি দিবো। ব্যাপারটা অনেকটা এরকম:

C++

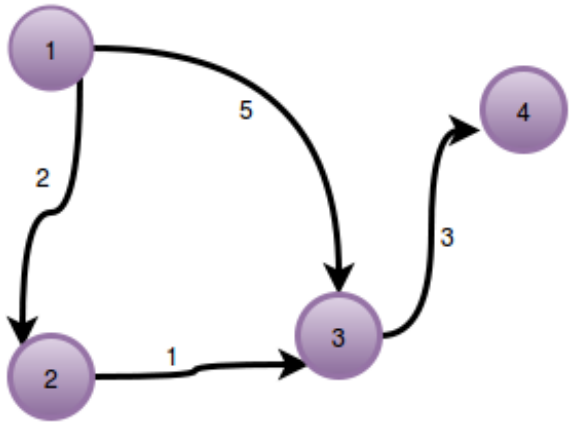
```
1    if d[u] + cost[u][v] < d[v]:
2    d[v] = d[u] + cost[u][v]
```

উপরের অংশটা যদি বুঝে থাকো তাহলে ডায়াক্সট্রা বোঝার ৬০% কাজ হয়ে গেছে। না বুঝে থাকলে আবার পড়ো।

বিএফএস নিশ্চয়ই তুমি ভালো করে বুঝো। বিএফএস এ আমাদের একটা নোডে কখনো দুইবার যাওয়া দরকার হয়নি, আমরা প্রতিবার দেখেছি একটা নোড ভিজিটেড কিনা, যদি ভিজিটেড না হয় তাহলে সেই নোডকে কিউতে পুশ করে দিয়েছি এবং ডিসটেন্স ১ বাড়িয়ে দিয়েছি।
ডায়াক্সট্রাতে আমরা একই ভাবে কিউ তে নোড রাখবো তবে ভিজিটেড দিয়ে আপডেট না করে নতুন এজকে "রিল্যাক্স" বা আপডেট করবো উপরের পদ্ধতিতে। নিচের গ্রাফটা দেখো:

ধরে নেই সোর্স হলো ১ নম্বর নোড। তাহলে

> d[1] = 0, d[2] = d[3] = d[4] = infinity(a large value)



ইনফিনিটি কারণ ২,৩,৪ এর দূরত্ব আমরা এখনো জানিনা, আর সোর্সের দূরত্ব অবশ্য শূন্য। এখন তুমি আগের বিএফএস এর মতোই সোর্স থেকে যতগুলো নোডে যাওয়া যায় সেগুলা আপডেট করার চেষ্টা করো, আপডেট করতে পারলে কিউতে পুশ করো। যেমন ১ – ২ এজটা ধরে আমরা আগাবো কারণ $d[1]+2 < d[2]$ এই শর্তটা পূরণ হচ্ছে। তখন $d[2]$ হয়ে যাবে ২, একই ভাবে ১ থেকে ৩ এ গেলে $d[3]$ হয়ে যাবে ৫।

কিন্তু ৫ তো ৩ নম্বরনোডে যাওয়ার শর্টেস্ট ডিসটেন্স না! আমরা বিএফএস এ দেখেছি একটা নোড একবারের বেশি আপডেট হয়না, সেই প্রোপার্টি এখানে কাজ করছেনা। ২ নম্বর নোড থেকে ২-৩ এজ ধরে এগিয়ে আবার আপডেট করলে তখন $d[3]$ তে $d[2] + 1 = 3$ পাবো। তাহলে আমরা দেখলাম এক্ষেত্রে একটা নোড অনেকবার আপডেট হতে পারে। (প্রশ্ন: সর্বোচ্চ কত বার?)

আমরা তাহলে আগের বিএফএস এর সুডোকোডের আপডেট অংশ একটু পরিবর্তন করি যাতে একটা নোড বার বার আপডেট হতে পারে:

```
1     1  procedure BFSmodified(G,source):
2     2     Q = queue(), distance[] = infinity
3     3     Q.enqueue(source)
4     4     distance[source] = 0
5     5     while Q is not empty
6     6        u ← Q.pop()
7     7        for all edges from u to v in G.adjacentEdges(v) do
8     8           if distance[u] + cost[u][v] < distance[v]
9     9                 distance[v] = distance[u] + cost[u][v]
10    10                 Q.enqueue(v)
11    11          end if
12    12       end for
13    13    end while
14    14    Return distance
```

আমরা ঠিক আগের বিএফএস এর কোডেই জাস্ট কস্ট বসায় বারবার আপডেট করছি! এই কোড তোমাকে সোর্স থেকে প্রতিটা নোডের শর্টেস্ট পাথ বের করে দিবে কিন্তু কমপ্লেক্সিটির দিক থেকে এটা খুবই বাজে! এজন্য আমাদের লাগবে একটা প্রায়োরিটি কিউ।

বিএফএস এ আমরা যখন ১ নোড থেকে অনেকগুলো নোডে যাচ্ছি তখন সেই নোডগুলো থেকে আবার নতুন করে কাজ করার সময় “আগে আসলে আগে পাবেন” ভিত্তিতে কাজ করছি। যেমন উপরের গ্রাফে ১ থেকে আগে ৩ নম্বর নোডে এবং তারপর ২ নম্বর নোডে এ গেলে আগে ৩ নিয়ে কাজ করছি, এরপর ২ নিয়ে কাজ করছি।

ভালো করে দেখো এখানে কি সমস্যাটা হচ্ছে। ৩ নিয়ে আগে কাজ করলে আমরা ৪ এর ডিসটেন্সকে আপডেট করে দিচ্ছি ডিসটেন্স ৫ + ৩ = ৮ হিসাবে। পরবর্তীতে যখন ২ দিয়ে ৩ কে আবার আপডেট করা হচ্ছে তখন ৩ এর ডিসটেন্স হয়ে গিয়েছে ৩, এবার ৪ এর ডিসটেন্সকে আবার আপডেট করছি ৩ + ৩ = ৫ হিসাবে। ৪ কে মোট দুইবার আপডেট করা লাগলো।

বিজ্ঞানী ডায়াক্সট্রা চিন্তা করলেন যদি এই “আগে আসলে আগে পাবেন” ভিত্তিতে কাজ না করে সবথেকে কাছের নোডগুলোকে আগে প্রসেস করি তাহলে অনেক কমবার আপডেট করা লাগে। আমরা যদি ২ কে নিয়ে আগে কাজ করতাম তাহলে ৩ আগেই আপডেট হয়ে যেত এবং ৪ কে

একবার আপডেট করেই শর্টেস্ট ডিসটেন্স পেয়ে যেতাম! একটু হাতে কলমে সিমুলেট করে দেখো। আইডিয়াটা হলো যেকোনো সময় কিউ তে যতগুলো নোড থাকবে তাদের মধ্যে যেটা সোর্স থেকে সবথেকে কাছে সেটা নিয়ে আগে কাজ করবো। এজন্যই আমরা কিউ এর জায়গায় বসিয়ে দিবো একটি প্রায়োরিটি কিউ যে কিউতে নোড পুশ করার সাথে সাথে কাছের নোডটাকে সামনে এনে দিবে। পার্থক্য হলো আগে খালি নোড নাম্বার পুশ করেছি, এখন বর্তমান ডিসটেন্স অর্থাত $d[u]$ এর মানটাও পুশ করতে হবে।

নিচে একটা সম্পূর্ণ ডায়াক্সট্রার সুডোকোড দিলাম যেটা ১ থেকে n তম নোডে যাবার শর্টেস্ট পাথ বের করে এবং পাথটাও প্রিন্ট করে:

C++

```
1   1  procedure dijkstra(G, source):
2   2    Q = priority_queue(), distance[] = infinity
3   3    Q.enqueue({distance[source], source})
4   4    distance[source]=0
5   5    while Q is not empty
6   6      u = nodes in Q with minimum distance[]
7   7      remove u from the Q
8   8      for all edges from u to v in G.adjacentEdges(v) do
9   9        if distance[u] + cost[u][v] < distance[v]
10  10               distance[v] = distance[u] + cost[u][v]
11  11               Q.enqueue({distance[v], v})
12  12       end if
13  13     end for
14  14   end while
15  15   Return distance
```

সি++ কোড দেখতে ক্লিক করো এখানে।

এখানে আগের সুডোকোডের থেকে কয়েক জায়গায় একটু ভিন্নতা আছে। এখানে সাধারণ কিউ এর জায়গায় প্রায়োরিটি কিউ ব্যবহার করা হয়েছে। কিউ থেকে পপ হবার সময় তাই সোর্স থেকে এখন পর্যন্ত পাওয়া সবথেকে কাছের নোডটা পপ হচ্ছে এবং সেটা নিয়ে আগে কাজ করছি।

উপরের সুডোকোডে সোর্স থেকে বাকি সব নোডের দূরত্ব বের করা হয়েছে। তুমি যদি শুধু একটা নোডের দুরত্ব বের করতে চাও তাহলে সেটা যখন কিউ থেকে পপ হবে তখনই রিটার্ন করে দিতে পারো।

নেগেটিভ এজ থাকলে কি ডায়াক্সট্রা অ্যালগোরিদম কাজ করবে? যদি নেগেটিভ সাইকেল থাকে তাহলে ইনফিনিট লুপে পরে যাবে, বারবার আপডেট করে কস্ট কমাতে থাকবে। যদি নেগেটিভ এজ থাকে কিন্তু সাইকেল না থাকে তাহলেও কাজ করবেনা। তবে তুমি যদি টার্গেট পপ হবার

সাথে সাথে রিটার্ণ করে না দাও তাহলে কাজ করবে কিন্তু সেটা তখন আর মূল ডায়াক্সট্রা অ্যালগোরিদম থাকবেনা।

কমপ্লেক্সিটি:
বিএফএস এর কমপ্লেক্সিটি ছিলো $O(V+E)$ যেখানে V হলো নোড সংখ্যা আর E হলো এজ সংখ্যা। এখানেও আগের মতোই কাজ হবে তবে প্রায়োরিটি কিউ তে প্রতিবার সর্ট করতে $O(logV)$ কমপ্লেক্সিটি লাগবে। মোট: $O(VlogV+E)$ ।

নেগেটিভ সাইকেল নিয়ে কাজ করতে হলে আমাদের জানতে হবে বেলম্যান ফোর্ড অ্যালগোরিদম। সেখানেও এজ রিল্যাক্স করে আপডেট করা হয়, একটা নোডকে সর্বোচ্চ $n – 1$ বার আপডেট করা লাগতে পারে সেই প্রোপার্টি কাজে লাগানো হয়।

ডায়াক্সট্রা ভালো করে শিখতে নিচের প্রবলেমগুলো ঝটপট করে ফেলো:

Dijkstra?
Not the Best
New Traffic System

হ্যাপি কোডিং!

(গ্রাফ থিওরি নিয়ে সবগুলো লেখা)

গ্রাফ থিওরিতে হাতেখড়ি ১০: ফ্লয়েড ওয়ার্শল

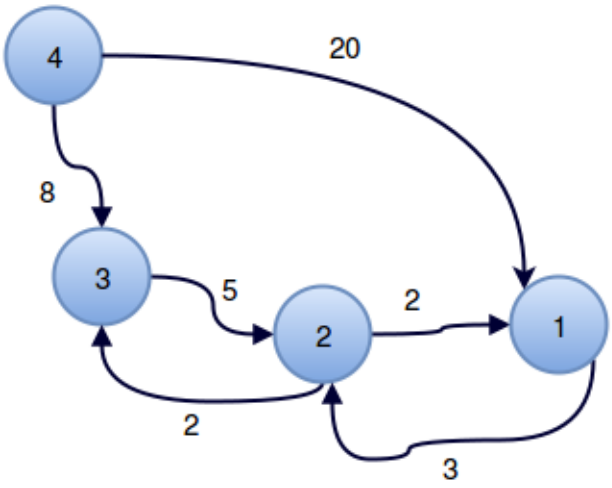
shafaetsplanet.com/

শাফায়েত

July 22, 2014

ফ্লয়েড ওয়ার্শল সম্ভবত সব থেকে ছোট আকারের গ্রাফ অ:্যালগোরিদম, মাত্র ৩লাইনে এটা লেখা যায়! তবে ৩ লাইনের এই অ:্যালগোরিদমেই বোঝার অনেক কিছু আছে। ফ্লয়েড ওয়ার্শলের কাজ হলো গ্রাফের প্রতিটা নোড থেকে অন:্য সবগুলো নোডের সংক্ষিপ্ততম দুরত্ব বের করা। এ ধরণের অ:্যালগোরিদমকে বলা হয় "অল-পেয়ার শর্টেস্ট পাথ" অ:্যালগোরিদম। এই লেখাটা পড়ার আগে অ:্যাডজেসেন্সি ম:্যাট্রিক্স সম্পর্কে জানতে হবে।

আমরা একটা গ্রাফের উপর কিছু সিমুলেশন করে সহজেই অ:্যালগোরিদমটা বুঝতে পারি। নিচের ছবিটা দেখ:



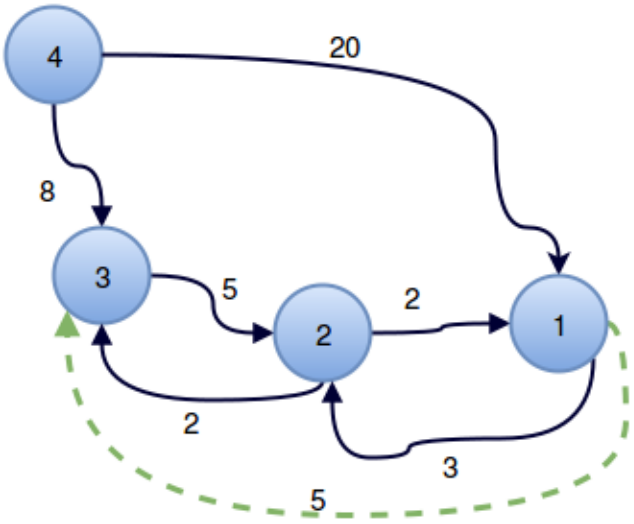
Node	1	2	3	4
1	0	3	inf	inf
2	2	0	2	inf
3	inf	5	0	inf
4	20	inf	8	0

ছবিতে চার নোডের একটা ওয়েটেড ডিরেক্টেড গ্রাফ দেখা যাচ্ছে। আর উপরে ডান কোনায় একটা ম:্যাট্রিক্স। ম:্যাট্রিক্সের u,v তম ঘরে বসানো হয়েছে u-v এজ এর ওয়েট বা কস্ট। যাদের মধ:্যে সরাসরি এজ নেই সেসব ঘরে অসীম বা ইনফিনিটি বসিয়ে দেয়া হয়েছে। আর কোনাকুনি ঘরগুলোতে মান ০ কারণ নিজের বাসা থেকে নিজের বাসাতেই যেতে কোন দূরত্ব অতিক্রম করতে হয় না!

এখন মনে করো "২" নম্বর নোডটাকে আমরা "মাঝের নোড" হিসাবে ধরলাম। মাঝের নোডকে আমরা বলবো k। তারমানে এখন k=2। (আমরা একে একে সব নোডকেই মাঝের নোড হিসাবে ধরবো, **এটা যেকোন অর্ডারে করা যায়**)

এখন যেকোন এক জোড়া নোড (i,j) নাও। ধরি i=1, v=3। আমরা চাই **u থেকে v তে যেতে, k নোডটাকে মাঝে রেখে**। তাহলে আমাদের i থেকে k তে যেতে হবে, তারপর k থেকে থেকে j তে যেতে হবে। কিন্তু লাভ না হলে আমরা এভাবে যাব কেন? আমরা k কে মাঝে রেখে যাবো কেবল

যদি মোট কস্ট(cost) কমে যায়। ১ থেকে ৩ এর বর্তমান দূরত্ব matrix[1][3]=ইনফিনিটি। আর যদি k=২ কে মাঝখানে রাখি তাহলে দূরত্ব দাড়াবে matrix[1][2] + matrix[2][3] = 3 +2 = 5। তারমানে কস্ট কমে যাচ্ছে! আমরা গ্রাফটা আপডেট করে দিতে পারি এভাবে:



Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	inf	5	0	inf
4	20	inf	8	0

আমরা ২ কে "মাঝের নোড" হিসাবে ব:্যবহার করে ১ থেকে ৩ এ গিয়েছি মোট ৫ কস্ট এ। গ্রাফে তাহলে ১ থেকে ৩ এ সরাসরি একটা এজ দিয়ে দিতে পারি ৫ কস্ট এ।

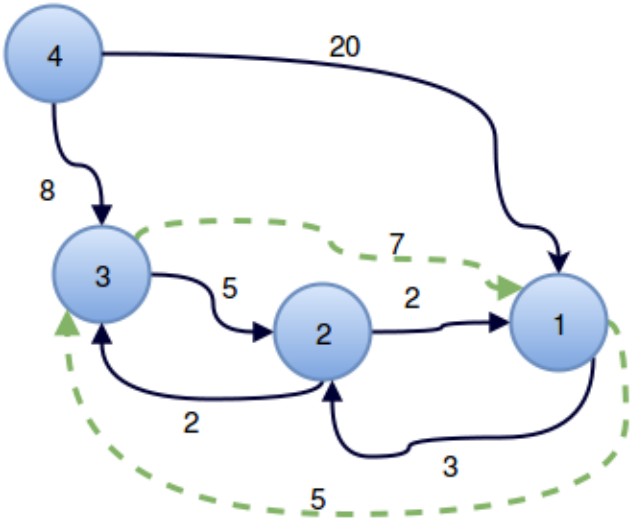
এখন আবার ধর i=2, j=4। আর আগের মতই k=2। এবার matrix[2][4]=ইনফিনিটি। এদিকে matrix[2][2] + matrix[2][4] = ০ + ইনফিনিটি। এবার কিন্তু দূরত্ব কমলো না। তাই গ্রাফ আপডেট করার দরকার নেই। নিশ্চয়ই বুঝতে পারছো আপডেটের শর্তটা হবে এরকম:

```
if(matrix[i][k] + matrix[k][j] < matrix[i][j])
matrix[i][j] = matrix[i][k] + matrix[k][j]
```

অথবা আমরা একলাইনে লিখতে পারি:

```
matrix[i][j] = min(matrix[i][j] , matrix[i][k] + matrix[k][j])
```

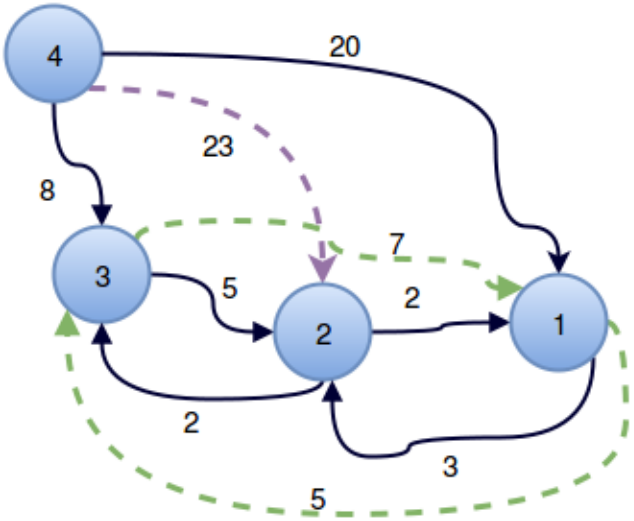
এখন k=2 স্থির রেখে আমরা i,j এর সবরকমের কম্বিনেশন নিবো। তুমি নিজেই চিন্তা করলে দেখবে k=2 এর জন:্য i=3, j=1 এই কম্বিনেশনে আমরা আরেকটা নতুন এজ পাবো, বাকি গ্রাফ আগের মতই থাকবে।



Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	7	5	0	inf
4	20	inf	8	0

এবার আমরা k=1 কে মাঝের নোড হিসাবে চিন্তা করি। মাথা খাটাতে চাইলে নিচে দেখার আগে নিজেই খাতায় একে ফেলতে পারো নতুন এজগুলো।

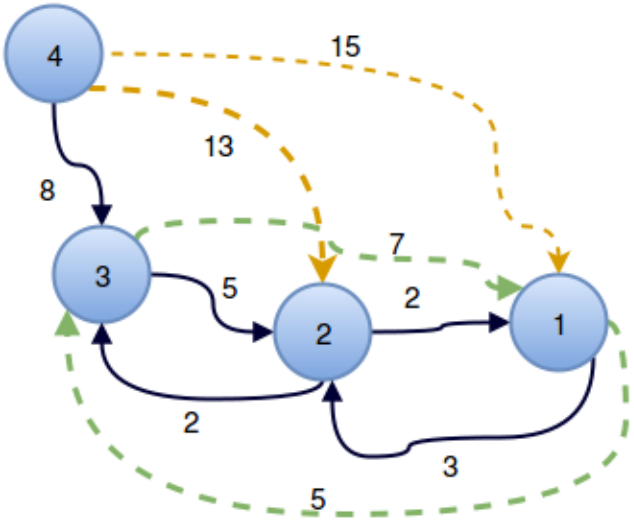
এবার একটা মাত্র এজ যোগ হবে। i=4, j=2 হলে আমরা ৪ থেকে ১ হয়ে ২ নম্বর নোডে যেতে পারি ২৩ কস্ট এ। তাহলে গ্রাফটা হবে এরকম:



Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	7	5	0	inf
4	20	23	8	0

এবার ৩ নম্বর নোডকে মাঝের নোড হিসাবে ধরবো। আবারো নিজে চেষ্টা করে তারপর নিচের অংশ দেখ।

এবার কিছু মজার জিনিস ঘটবে। ৪ থেকে ১ এ আগে লাগছিল ২০ কস্ট। এখন ৩ কে মাঝে রেখে ৪ থেকে ১ এ গেলে লাগবে ৮+৭=১৫ কস্ট। লক্ষ্য কর একদম শুরুতে ৩ থেকে ১ এ আমাদের এজ ছিল না। কিন্তু আপডেট করার সময় আমরা এজ বসিয়ে দিয়েছি, এখন ৩ থেকে ১ এ যদিও সরাসরি চলে যাচ্ছি, মূল গ্রাফে আসলে ৩->২->১ পথে যাচ্ছি।
একই ভাবে ৪ থেকে ২ এর ২৩ কস্ট এর পথটা আপডেট হয়ে ১৩ হয়ে যাবে।



Node	1	2	3	4
1	0	3	5	inf
2	2	0	2	inf
3	7	5	0	inf
4	15	13	8	0

k=৪ এর জন্য আর কোন আপডেট হবে না কারণ কোনো নোড থেকে ৪ এ যাওয়া যায় না।

এখন আমরা ম্যাট্রিক্স দেখেই বলে দিতে পারছি কোন নোড থেকে কোন নোডে কত কস্ট এ যাওয়া যায়। ইনফিনিটি থাকা মানে সেই নোডে যাবার পথ নেই।

ইনপুট থেকে অ্যাডজেসেন্সি ম্যাট্রিক্স বানাবার পর তাহলে কাজ খুবই সহজ:

C++

```
1   for k from 1 to |V|
2      for i from 1 to |V| \\ |V| = number of nodes
3         for j from 1 to |V|
4            if matrix[i][j] > matrix[i][k] + matrix[k][j]
5               matrix[i][j] ← matrix[i][k] + matrix[k][j]
```

কোডে আমরা k এর লুপটা ১ থেকে চালাচ্ছি যদিও উদাহরণে আগে ২ নিয়েছি। **এটা আসলে যেকোন অর্ডারেই করা যায়,** তুমি আগে ১ নিলেও দেখবে অ্যালগোরিদম কাজ করবে।

এভাবেতো আমরা শুধু পাথের কস্ট পেলাম, পাথটা কিভাবে পাব?

ধরো আমাদের একটা ম্যাট্রিক্স আছে next[][]। এখন next[i][j] দিয়ে আমরা বুঝি **i থেকে j তে যেতে হলে পরবর্তি যে নোড এ যেতে হবে** সেই নোডটা। তাহলে একদম শুরুতে সব i,j এর জন্য next[i][j] = j হবে। কারণ শুরুতে কোন "মাঝের নোড" নেই এবং তখনও শর্টেস্ট পাথ বের করা শেষ হয় নি।

এখন আমরা যখন matrix[i][j] আপডেট করবো লুপের সেটার মানে হলো মাঝে একটা নোড k ব্যবহার করে আমরা যাবো। লক্ষ্য কর আমরা কিন্তু মূল গ্রাফে সরাসরি এজ দিয়ে i থেকে k তে নাও যেতে পারি, আমরা শুধু জানি i,j নোড দুটোর মাঝে একটা নোড k আছে যেখানে আমাদের যেতে হবে j তে যাবার আগে। i থেকে k তে যাবার পথে পরবর্তি যে নোডে যেতে হবে সেটা রাখা আছে next[i][k] তে! তাহলে next[i][j] = next[i][k] হয়ে যাবে।

```
1    for k from 1 to |V|
2        for i from 1 to |V|
3          for j from 1 to |V|
4            if matrix[i][k] + matrix[k][j] < matrix[i][j] then
5              matrix[i][j] ← matrix[i][k] + matrix[k][j]
6              next[i][j] ← next[i][k]
7    findPath(i, j)
8      path = [i]
9      while i ≠ j
10       i ← next[i][j]
11       path.append(i)
12     return path
13
14
```

findpath ফাংশনে আমরা j কে ফিক্সড রেখে next অ্যারে ধরে আগাচ্ছি যতক্ষণ না j তে পৌছাচ্ছি। তাহলে আমরা পেয়ে গেলাম পাথ!

ট্রান্সিটিভ ক্লোজার(Transitive Closure):

ধরো আমাদের অ্যাডজেন্সি ম্যাট্রিক্সটা এরকম:

```
1    matrix[i][j] = 1 যদি i থেকে j তে সরাসরি এজ থাকে
2    matrix[i][j] = 0 যদি এজ না থাকে
```

এখন আমরা এমন একটা ম্যাট্রিক্স তৈরি করতে চাই যেটা দেখে বলে দেয়া যাবে i থেকে j তে এক বা একাধিক এজ ব্যবহার করে যাওয়া যায় কিনা। আমরা চাইলে উপরের মত করে o এর জায়গায় ইনফিনিটি দিয়ে শর্টেস্ট পাথ বের করে কাজটা করতে পারতাম। কিন্তু এক্ষেত্রে “OR” আর “AND” অপারেশন ব্যবহার আরো দ্রুত কাজটা করা যায়। এখন আপডেটের শর্তটা হয়ে যাবে এরকম:

```
1    matrix[i][j] = matrix[i][j] || (matrix[i][k] && matrix[k][j])
```

এটার মানে matrix[i][j] তে তখনই ১ বসবে যখন হয় “matrix[i][j] তে ১ আছে” অথবা “matrix[i][k] এবং matrix[k][j]” দুটোতেই ১ আছে। তারমানে হয় সরাসরি যেতে হবে অথবা মাঝে একটা নোড k ব্যবহার করে যেতে হবে।

কমপ্লেক্সিটি:

৩টা নেস্টেড লুপ ঘুরছে নোড সংখ্যার উপর, টাইম কমপ্লেক্সিটি O(n^3)। ২ডি ম্যাট্রিক্স ব্যবহার করায় স্পেস কমপ্লেক্সিটি O(n^2)।

কিছু প্রশ্ন:

১. k এর লুপটা কি i,j লুপের ভিতর দিলে অ্যালগোরিদম কাজ করত?
২. গ্রাফে নেগেটিভ কস্ট থাকলে ফ্লয়েড ওয়ার্শল কাজ করবে কি?
রিলেটেড প্রবলেম:

Page Hopping
05-2 Rendezvous
Minimum Transport Cost
Asterix and Obelix
আরো অনেক প্রবলেম

হ্যাপি কোডিং!

গ্রাফ থিওরিতে হাতেখড়ি ১১: বেলম:্যান ফোর্ড

shafaetsplanet.com/

শাফায়েত

October 17, 2014

বেলম:্যান ফোর্ড গ্রাফে শর্টেস্ট পাথ বের করার একটা অ:্যালগোরিদম। এই অ:্যালগোরিদম একটা নোডকে সোর্স ধরে সেখান থেকে সব নোডের সংক্ষিপ্ততম বা শর্টেস্ট পথ বের করতে পারে। আমরা একদম শুরুতে এই কাজ করার জন:্য ব্রেডথ ফার্স্ট সার্চ শিখেছি। কিন্তু বিএফএস(BFS) যেহেতু ওয়েটেড গ্রাফে কাজ করে না তাই এরপর আমরা শিখেছি ডায়াক্সট্রা অ:্যালগোরিদম। এখন বেলম:্যান ফোর্ড শিখব কারন আগের কোনো অ:্যালগোরিদমই নেগেটিভ ওয়েট এর এজ আছে এমন গ্রাফে কাজ করে না।

আমরা ডায়াক্সট্রা শেখার সময় রিল:্যাক্সেশন নামের একটা ব:্যাপার শিখেছিলাম। তোমার যদি মনে না থাকে বা ডায়াক্সট্রা না শিখে থাকো তাহলে আমরা প্রথমে একটু ঝালাই করে নেই আরেকবার। মনে থাকলে পরের অংশটা বাদ দিয়ে সরাসরি এখানে যেতে পার।

এজ রিল:্যাক্সেশন:

ধর একটা গ্রাফে সোর্স থেকে প্রতিটা নোডের ডিসটেন্স/কস্ট রাখা হয়েছে $d[]$ অ্যারেতে। যেমন $d[3]$ মানে হলো সোর্স থেকে বিভিন্ন এজ পার হয়ে ৩ নম্বর নোড এ আসতে মোট $d[3]$ ডিসটেন্স পার করতে হয়েছে। যদি ডিসটেন্স জানা না থাকে তাহলে ইনফিনিটি অর্থা অনেক বড় একটা মান রেখে দিবো। আর $cost[u][v]$ তে রাখা আছে $u-v$ এজ এর cost।

ধর তুমি বিভিন্ন জায়গা ঘুরে ফার্মগেট থেকে টিএসসি তে গেলে ১০ মিনিটে, আবার ফার্মগেট থেকে কার্জন হলে গেলে ২৫ মিনিটে। তাহলে ফার্মগেটকে সোর্স ধরে আমরা বলতে পারি:

> d[টিএসসি] = ১০, d[কার্জন হল] = ২৫

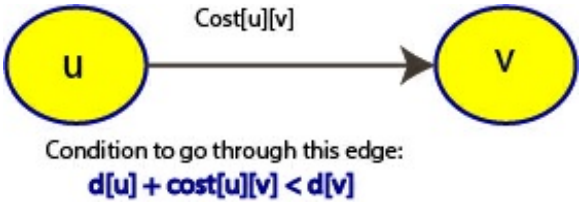
এখন তুমি দেখলে টিএসসি থেকে ৭ মিনিটে কার্জনে চলে যাওয়া যায়,

> cost[টিএসসি][কার্জন হল] = ৭

তাহলে তুমি ২৫ মিনিটের জায়গায় মাত্র ১০ + ৭ = ১৭ মিনিটে কার্জ নহলে যেতে পারবে। যেহেতু তুমি দেখেছো:

> d[টিএসসি]+cost[টিএসসি][কার্জন] < d[কার্জন হল]

তাই তুমি এই নতুন রাস্তা দিয়ে কার্জন হলে গিয়ে d[কার্জন হল] = d[টিএসসি] + cost[টিএসসি][কার্জন হল] বানিয়ে দিতেই পারো!!



উপরের ছবিটা সেটাই বলছে। আমরা u থেকে v তে যাবো যদি **d[u]+cost[u][v] < d[v]** হয়। আর $d[v]$ কে আপডেট করে $d[v] = d[u] + cost[u][v]$ বানিয়ে দিবো। ভবিষ্যতে যদি কার্জনহলে অন্য রাস্তা দিয়ে আরো কম সময়ে যেতে পারি তখন সেই রাস্তা এভাবে কম্পেয়ার করে আপডেট করি দিবো। ব্যাপারটা অনেকটা এরকম:

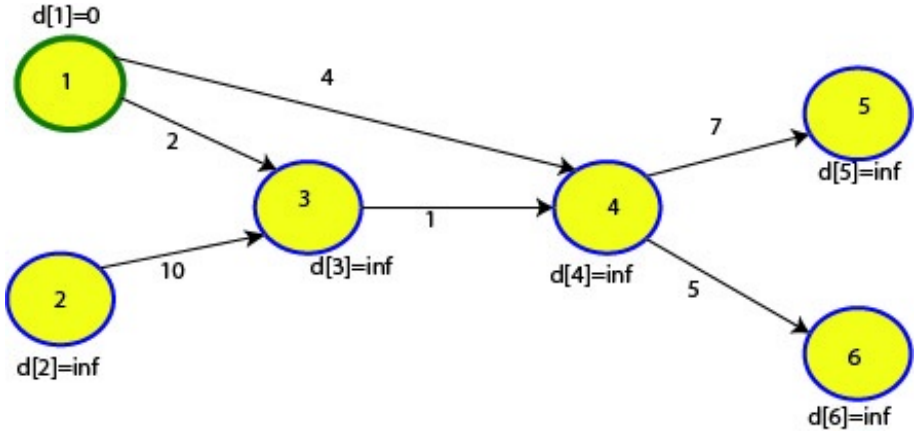
C++

```
1  if(d[u]+cost[u][v] < d[v])
2  d[v] = d[u] + cost[u][v];
```

এটাই হলো এজ রিল◌্যাক্সেশন। এখন আমরা বেলম◌্যান ফোর্ড শেখার জন◌্য তৈরি।

বেলম◌্যান ফোর্ড

নিচের গ্রাফে আমরা ১ থেকে শুরু করে প্রতিটা নোডে যাবার শর্টেস্ট পাথ বের করতে চাই:

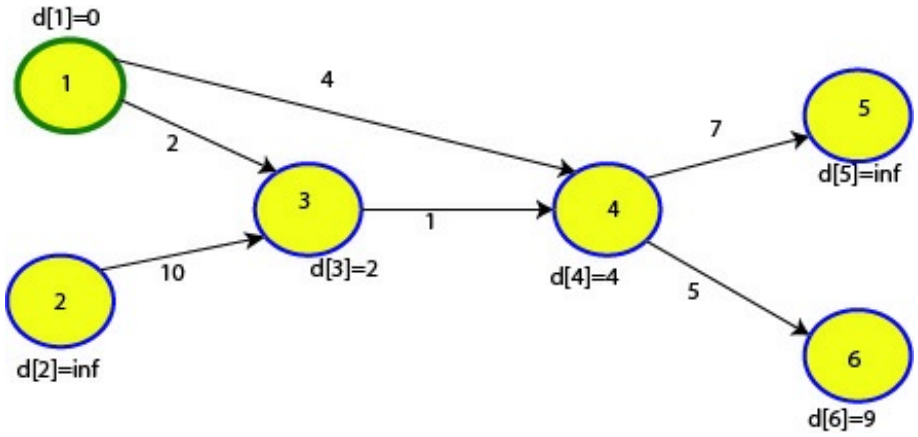


শুরুতে $d[1]=0$ কারণ ১ হলো সোর্স। বাকিসবগুলোতে ইনফিনিটি রেখেছি কারণ আমরা এখনও জানিনা শর্টেস্ট পাথের কস্ট কত।

তুমি এজ রিল◌্যাক্স কিভাবে করতে হয় এরই মধ◌্যে শিখে গেছ। এখন কাজ হলো সবগুলো এজকে একবার করে রিল◌্যাক্স করা, **যেকোন অর্ডারে। একবার 'গ্রাফ রিল◌্যাক্স' করার মানে হল গ্রাফটার সবগুলো এজকে একবার করে রিল◌্যাক্স করা।** আমি নিচের অর্ডারে রিল◌্যাক্স করতে চাই,

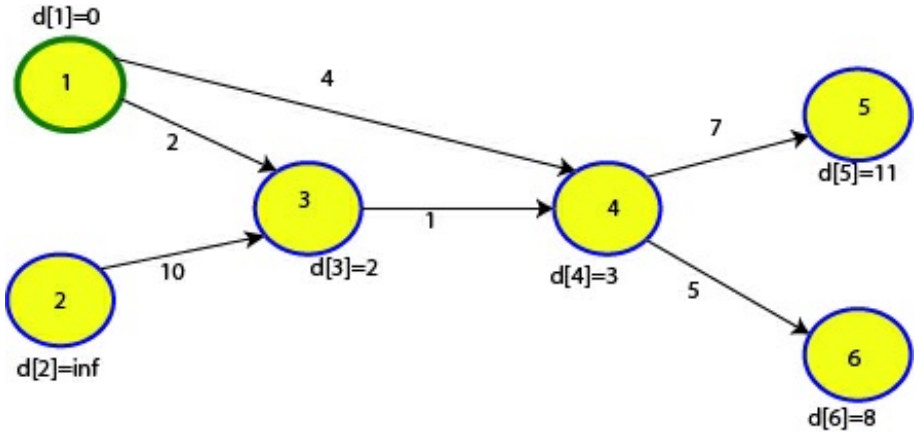
Serial	1	2	3	4	5	6
Edge	4 -> 5	3 -> 4	1 -> 3	1 -> 4	4 -> 6	2 -> 3

তুমি চাইলে অন্য যেকোনো অর্ডারেও এজগুলো নিতে পারতে। এখন চিন্তা কর এজগুলোকে একবার রিল্যাক্স করলে আমরা d[] অ্যারেতে কি পাব? **সোর্স থেকে শুরু করে সর্বোচ্চ ১টা এজ ব্যবহার করে অন্যান্য নোডে যাবার শর্টেস্ট পাথের কস্ট আমরা পেয়ে যাব।** উপরের ছবিতে রিল্যাক্স করার পর d[] এর মানগুলো আপডেট করে দাও। করার পর ছবিটা নিচের মত হবার কথা:



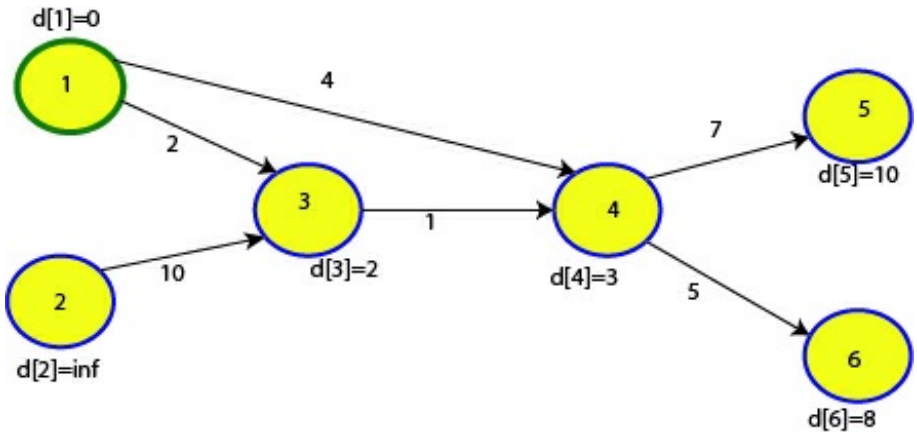
এজ রিল্যাক্স করার সময় কিছু নোড এর কস্ট আপডেট করতে পারি নি কারণ **d[u]+cost[u][v] < d[v]** শর্তটা পূরণ করে নি। বাকি এজগুলো আপডেট করার পর $d[]$ অ্যারের এর মান উপরের ছবির মত পেয়েছি। ১ নম্বর নোড থেকে শুরু করে সর্বোচ্চ একটি এজ ব্যবহার করে সব নোডে যাবার শর্টেস্ট পাথ এখন আমরা জানি!

এখন সর্বোচ্চ ২টা এজ ব্যবহার করে সব নোডে যাবার শর্টেস্ট পাথের cost বের করতে আরেকবার রিল্যাক্স করে ফেলি! আবারো যেকোন অর্ডারে করা যাবে, তবে প্রথমে যে অর্ডারে করেছি সেভাবেই প্রতিবার করা কোড লেখার সময় সুবিধাজনক।



একটা ব্যাপার লক্ষ্য কর, ১ থেকে ৬ তে যাবার শর্টেস্ট পথে ৩ টা এজ আছে (১->৩, ৩->৪, ৪->৬) এবং পথের দৈর্ঘ্য ২+১+৫=৮। মাত্র ২বার রিল্যাক্স করলেও আমরা এখনই d[6] তে ৮ পেয়ে গেছি, অথচ আমাদের এখন সর্বোচ্চ ২টা এজ ব্যবহার করে শর্টেস্ট পাথের cost পাবার কথা। এটা নির্ভর করে তুমি কোন অর্ডারে এজ রিল্যাক্স করেছ তার উপর। সে কারণে ৫ এ যাবার শর্টেস্ট পাথ ১০ হলেও d[5] এ এখনো ১০ পাইনি। **X বার 'গ্রাফ রিল্যাক্স' করলে সর্বোচ্চ X টা এজ ব্যবহার করে সোর্স প্রতিটা নোডে যাবার শর্টেস্ট পাথ তুমি নিশ্চিত ভাবে পাবে। X এর থেকে বেশি এজের ব্যাবহার করে প্রতিটা নোডে যাবার শর্টেস্ট পাথ তুমি X বার গ্রাফ রিল্যাক্সের পর পেতেও পার, নাও পেতে পার, সেটা এজ এর অর্ডারের উপর নির্ভর করে।**

এখন ৩য় বারের মত রিল্যাক্স করি:



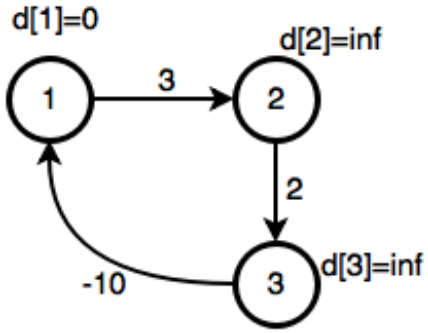
এবার শুধু মাত্র ৫ নম্বর নোড আপডেট হবে।

এরপরে আমরা আর যতই আপডেট করি, $d[]$ অ্যারেতে কোনো পরিবর্তন হবে না, আমরা ১ থেকে প্রতিটা নোডে যাবার শর্টেস্ট পাথ পেয়ে গিয়েছি।

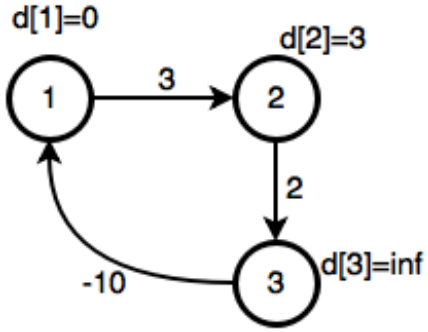
এখন স্বাভাবিকভাবেই প্রশ্ন আসবে যে রিল্যাক্স কয়বার করতে হবে? গ্রাফে যদি নোড n টা থাকে তাহলে এক নোড থেকে অন্য নোডে যেতে সর্বোচ্চ $n-1$ টা এজ ব্যবহার করতে হবে। তারমানে কোনো নোড সর্বোচ্চ $n-1$ বার আপডেট হতে পারে। তাই রিল্যাক্স করার লুপটাও চালাতে হবে $n-1$ বার। তবে আমরা যেরকম উপরের গ্রাফে দেখেছি ৩বারের পরেই আর কোন নোড আপডেট করা যাচ্ছে না, সেরকম হলে আর নতুন করে রিল্যাক্স করার দরকার নাই।

এখন নিচের মাত্র ৩নোডের গ্রাফটায় বেলম্যান ফোর্ড অ্যালগোরিদম চালাও, অর্থা যতক্ষণ কোন নোড আপডেট করা যায় ততক্ষণ পুরো গ্রাফটা রিল্যাক্স কর:

Serial	1	2	3
Edge	2 -> 3	1 -> 2	3 -> 1

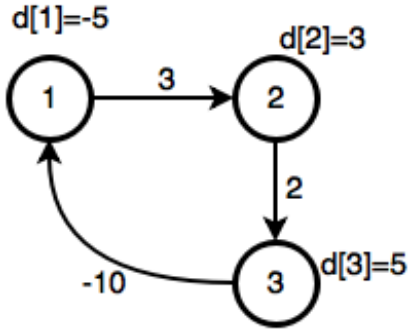


প্রথমবার রিলাক্স করার পর পাব:

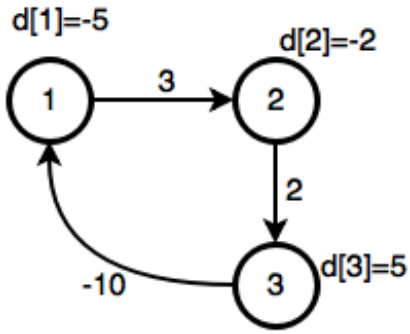


২য়বার করার পর:

৩ নোডের গ্রাফে সোর্স থেকে কোনো নোডে শর্টেস্ট যেতে ২টার বেশি এজ লাগবে না, ৩য় বার রিলাক্স করার চেষ্টা করলে কোনো নোড আপডেট হবার কথা না। কিন্তু এই গ্রাফে আপডেট হচ্ছে:



এটা হচ্ছে কারণ $1->2->3->1$ সাইকেলটার মোট ওয়েট নেগেটিভ ($3 + 2 – 10 = -5$)। তাই তুমি যতবার এই সাইকেলে ঘুরবে শর্টেস্ট পাথ তত ছোট হতে থাকবে। **তাই নেগেটিভ সাইকেল থাকলে এবং সোর্স থেকে সেই নেগেটিভ সাইকেলে যাবার পথ থাকলে সোর্সের শর্টেস্ট পাথ আনডিফাইনড বা অসংজ্ঞায়িত।** যদি $n-1$ বার গ্রাফ রিলাক্স করার পর দেখি যে n তম বারও কোনো নোডের cost আপডেট করা যায় তখন বুঝতে হবে আমরা নেগেটিভ সাইকেলে গিয়ে পরেছি, শর্টেস্ট পাথ বের করা সম্ভব না।



আমাদের সুডোকোড তাহলে হবে এরকম:

bellman

```
Input: A non-empty connected weighted graph G with vertices G.V and edges G.E
procedure Bellman Ford(G,source):
1  Let distance[] ← infinity
2  Let N ← number of nodes
3  distance[source] = 0
4  for step from 1 to N-1
5          for all edges from (u,v) in G.E
6             if distance[u] + cost[u][v] < distance[v]
7                     distance[v] = distance[u] + cost[u][v]
8                end if
9          end for
10 end for
11 for all edges from (u,v) in G.E
12         if distance[u] + cost[u][v] < distance[v]
13      return “Negative cycle detected
14         end if
15 end for
16 return distance
```

পাথ প্রিন্ট করা:

বিএফএস বা ডায়াক্সট্রাতে যেভাবে পাথ প্রিন্ট করে ঠিক সেভাবে এখানেও পাথ প্রিন্ট করা যাবে। $previous[]$ নামের একটা অ্যারে নাও। $previous[v] = u$ মানে হলো v তম নোডে তুমি u থেকে এসেছ। শুরুতে অ্যারেতে ইনফিনিটি থাকবে। $u->v$ এজটা রিল্যাক্স করার সময় $previous[v] = u$ করে দাও। এখন তুমি $previous$ অ্যারে দেখে সোর্স থেকে যেকোন নোডের পাথ বের করে ফেলতে পারবে।

কমপ্লেক্সিটি:

সর্বোচ্চ $n-1$ বার প্রতিটা এজকে রিল্যাক্স করতে হবে, টাইম কমপ্লেক্সিটি $O(n*e)$।

চিন্তা করার জন্য কিছু প্রবলেম:

- তোমাকে একটা গ্রাফ দিয়ে বলা হল শর্টেস্ট পাথে সর্বোচ্চ x টা এজ থাকতে পারে। এবার কিভাবে শর্টেস্ট পাথ বের করবে? (UVA 11280)
- একটি গ্রাফে কোন কোন নোড নেগেটিভ সাইকেলের অংশ কিভাবে বের করবে? (হিন্টস: স্ট্রংলি কানেক্টেড কম্পোনেন্ট + বেলম্যান ফোর্ড)

রিলেটেড কিছু প্রবলেম:

UVA 558(সহজ)
LOJ 1108
UVA 10449

হংয্যাপি কোডিং!

গ্রাফ থিওরিতে হাতেখড়ি ১২ – ম্যাক্সিমাম ফ্লো (১)

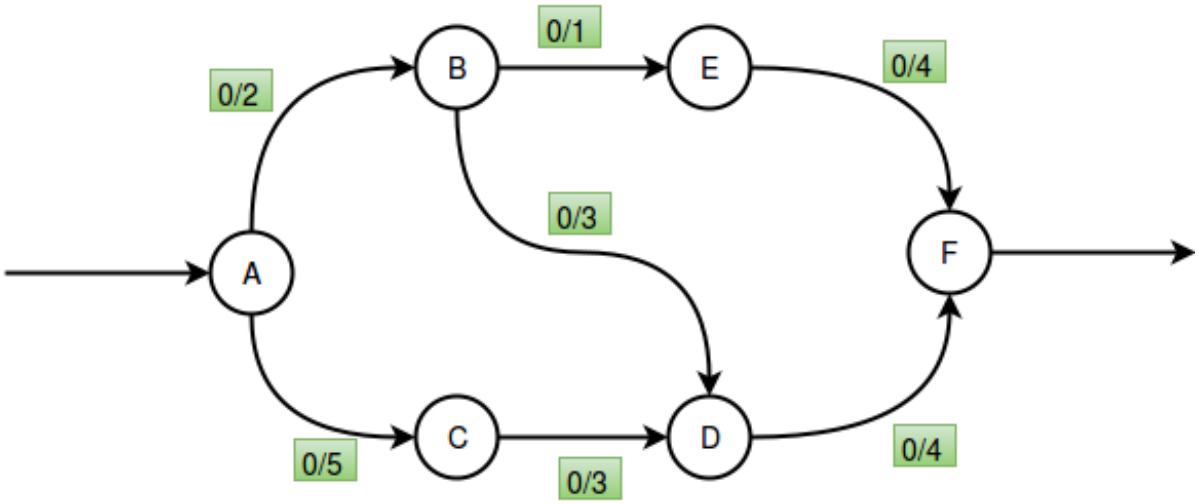
shafaetsplanet.com/

শাফায়েত

January 23, 2015

এই লেখায় আমরা গ্রাফে ম্যাক্সিমাম ফ্লো বের করার অ্যালগোরিদম শিখবো। ম্যাক্স ফ্লো এর ধারণাটা ব্যবহার করে বেশ কিছু ইন্টারেস্টিং প্রবলেম সলভ করা যায়, তাই এটা শেখা খুবই গুরুত্বপূর্ণ। এই লেখাটা পড়ার আগে তোমাকে গ্রাফ থিওরির বেসিক অ্যালগোরিদমগুলো, বিশেষ করে শর্টেস্ট পাথ বের করার অ্যালগোরিদমগুলো ভালো করে শিখে নিবে।

প্রথমেই আমরা দেখি একটি খুবই সাধারণ ম্যাক্স ফ্লো প্রবলেম:



চিত্র: ১

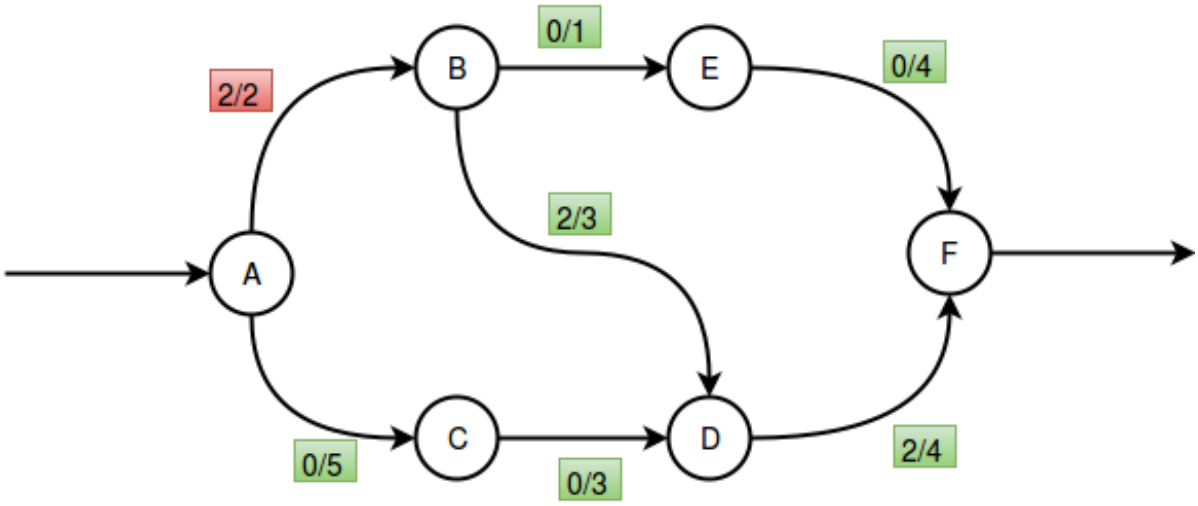
তোমাকে চিত্র-১ এর মত একটা গ্রাফ দেয়া আছে। মনে কর গ্রাফের প্রতিটা এজ একটা করে পানির পাইপ। প্রতিটা পাইপ দিয়ে প্রতি সেকেন্ডে কত লিটার পানি প্রবাহিত হতে পারবে সেটার একটা সীমা আছে যেটাকে বলা হয় পাইপের **ক্যাপাসিটি**। আর কোনো পাইপ দিয়ে সেকেন্ডে যতটুকু পানি যাচ্ছে সেটা হলো পাইপটার **ফ্লো**। প্রতিটা এজের সাথে "ফ্লো/ক্যাপাসিটি" উল্লেখ করে দেয়া হয়েছে। যেমন A->B এজ দিয়ে সেকেন্ডে সর্বোচ্চ ২লিটার পানি যেতে পারে এবং এই মুহূর্তে সেকেন্ডে পানি যাচ্ছে ০ লিটার। শুধুমাত্র A নোড দিয়ে পানি প্রবেশ করানো যায় এবং শুধুমাত্র F নোড দিয়ে পানি বের হয়ে যায়। A কে বলা হয় **সোর্স** এবং F হলো **সিংক**। তোমাকে বলতে হবে A থেকে F এ প্রতি সেকেন্ডে সর্বোচ্চ কত লিটার পানি প্রবাহিত করা যাবে? অর্থা পানির "ফ্লো" সর্বোচ্চ কত হতে পারে?

মূল সমস্যা সমাধানের আগে আমরা ছোট একটা সমস্যা সমাধান করি। A->C->D->F এই পথটা ব্যবহার করে পানি সোর্স থেকে সিংক এ গেলে সর্বোচ্চ কত লিটার পানি প্রবাহিত হতে পারবে? ছবিতে দেখা যাচ্ছে A->C এজের ক্যাপাসিটি ৫, C->D এজের ক্যাপাসিটি 3, D->F

এজের ক্যাপাসিটি ৪। এখানে সর্বনিম্ন ক্যাপাসিটি হলো ৩, তাই আমরা কোনো সময় ৩লিটারের বেশি পানি এই পথে পাঠাতে পারবোনা।

কোনো পথের সর্বনিম্ন ক্যাপাসিটির এজ সেই পথের ফ্লো নিয়ন্ত্রণ করে। বোতলের সরু মুখের সাথে তুলনা দিয়ে এই ব্যাপারটাকে বলা হয় বোটলনেক(bottleneck)। যেমন ধরো তোমার বাসায় ৫MBps এর ইন্টারনেট কানেকশন আছে, এখন তুমি একটা মুভি ডাউনলোড করতে চাচ্ছ। এখন মুভির সার্ভার থেকে তোমার বাসায় ডাটা প্যাকেট আসার আগে অনেকগুলো নেটওয়ার্ক পার হয়ে আসে, কোনো একটা নেটওয়ার্কে গতি মাত্র 1MBps। এখন নেটওয়ার্কের বাকি অংশের গতি যতই বেশি হোক না কেন তুমি 1MBps এর বেশি গতিতে ডাউনলোড করতে পারবে না, নেটওয়ার্কের সবথেকে ধীরগতির অংশই তোমার ডাউনলোডের গতি নিয়ন্ত্রণ করবে।

এখন দেখি কিভাবে সর্বোচ্চ পানির ফ্লো পাবার সমস্যাটার সমাধান করা যায়। সমাধান খুবই সহজ, আমরা প্রতিবার যেকোনো একটা করে পথ দিয়ে পানি পাঠাতে থাকবো যতক্ষণ পাইপে ক্যাপাসিটি থাকে। কিন্তু এভাবে কি আমরা সর্বোচ্চ ফ্লো বা প্রবাহ পাবো? পাবো তবে সেজন্য একটু বুদ্ধি খাটাতে হবে। প্রথমে আমরা একটা পথে পানির ফ্লো পাঠিয়ে দেখি কি ঘটে। যেকোনো একটা পথ বেছে নিতে পারি, আমি বুঝানোর সুবিধার জন্য A->B->D->F পথটা বেছে নিলাম। এই পথে সর্বনিম্ন ক্যাপাসিটি হলো ২। তাহলে এই পথে ২লিটার পানির ফ্লো পাঠিয়ে দেবার পর গ্রাফটা দেখতে হবে এরকম:

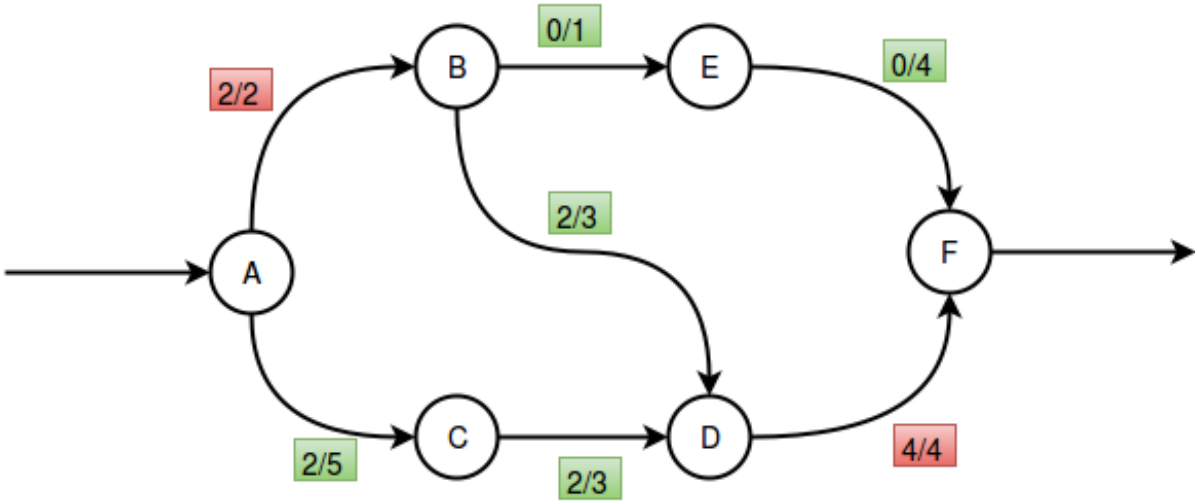


চিত্র-২: A->B->D->F পথে ফ্লো পাঠানোর পর

আমরা A->B->D->F পথের প্রতিটি এজের ফ্লো ২বাড়িয়ে দিয়েছি। লক্ষ্য করো A->B এজের আসল ক্যাপাসিটি ২ এবং ফ্লো পাঠানো হয়েছে ২, তাই এই এজ নিয়ে আর কোনো ফ্লো পাঠানো যাবে না। তেমনি B->D এজের ক্যাপাসিটি ৩ কিন্তু ফ্লো পাঠানো হয়েছে ২, তাই এই এজটা দিয়ে আরো ৩-২=১ ফ্লো পাঠানো সম্ভব। আমরা বলতে পারি B->D এজের **residual ক্যাপাসিটি**=১, residual শব্দটার অর্থ হলো কোনো কিছু ব্যবহার করে ফেলার পর বেচে যাওয়া অংশ।

> residual ক্যাপাসিটি = এজ এর ক্যাপাসিটি – ব্যবহৃত ক্যাপাসিটি বা ফ্লো এর পরিমাণ ।

এবার একইভাবে A->C->D->F পথটা নির্বাচিত করি। এই পথে D->F এজের আসল ক্যাপাসিটি ৪ হলেও ২ ফ্লো আগেই পাঠানো হয়েছে, তাই বর্তমান ক্যাপাসিটি বা **residual ক্যাপাসিটি** ৪-২=২। এই পথটা দিয়ে ২ ফ্লো পাঠানো সম্ভব।



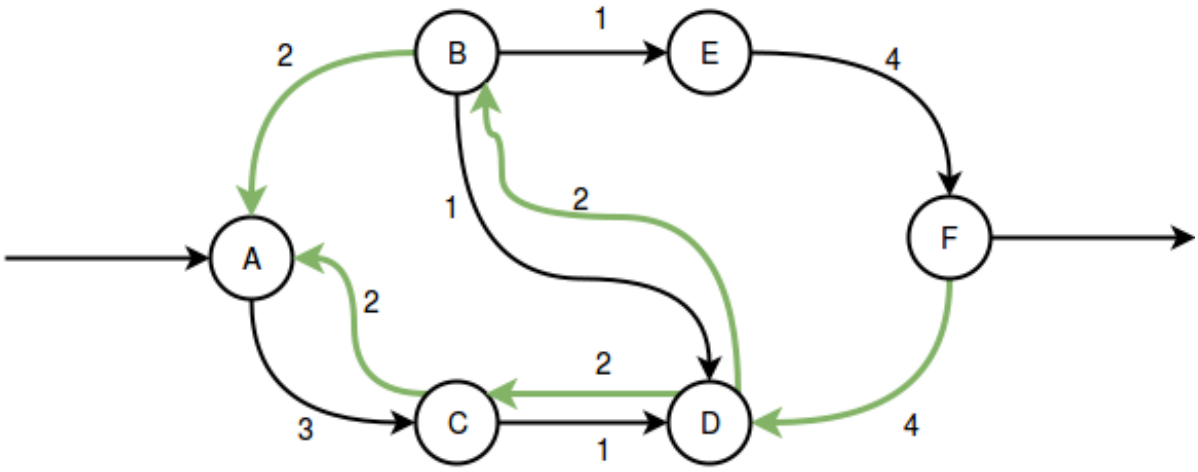
চিত্র-৩: A->C->D->F পথে ফ্লো পাঠানোর পর

আমরা এখন মোট ২+২=৪ ফ্লো পেলাম, অর্থা পানি প্রবাহের হার এখন সেকেন্ডে ৪ লিটার। এই গ্রাফে কি আরো বেশি পানির ফ্লো পাঠানো সম্ভব? A->B এবং D->F এজ এরই মধ্যে সর্বোচ্চ ক্যাপাসিটিতে পৌছে গেছে, এই ২টা এজের কোনোটাকে না নিয়ে সিংক এ যাবার পথ গ্রাফে নেই। তাই দেখে মনে হতে পারে আর ফ্লো পাঠানো সম্ভব না। কিন্তু আমরা এখন বুদ্ধি খাটিয়ে আরো বেশি ফ্লো পাঠিয়ে দিব!

আমাদের এখন প্রতিটা এজের residual ক্যাপাসিটি ব্যবহার করে **residual গ্রাফ** আকঁতে হবে কিভাবে আরো ফ্লো পাঠাবো সেটা বুঝতে হলে।residual গ্রাফ আকার নিয়ম হলো:

> ১. গ্রাফের প্রতিটা এজ (u,v) এর ক্যাপাসিটি হবে এজ টার residual ক্যাপাসিটির সমান।
> ২. প্রতি এজ (u,v) এর জন্য উল্টা এজ (v,u) এর residual ক্যাপাসিটি হবে (u,v) এজ এ ফ্লো এর সমান।

তারমানে কোনো এজ দিয়ে যতটুকু ফ্লো পাঠিয়েছি সেটা হবে উল্টো এজের residual ক্যাপাসিটি। তাহলে residual গ্রাফে মূল এজ এবং উল্টো এজের residual ক্যাপাসিটির যোগফল হবে মূল এজ এজের মোট ক্যাপাসিটির সমান। চিত্র-৪ এর গ্রাফটার residual গ্রাফটা হবে এরকম:



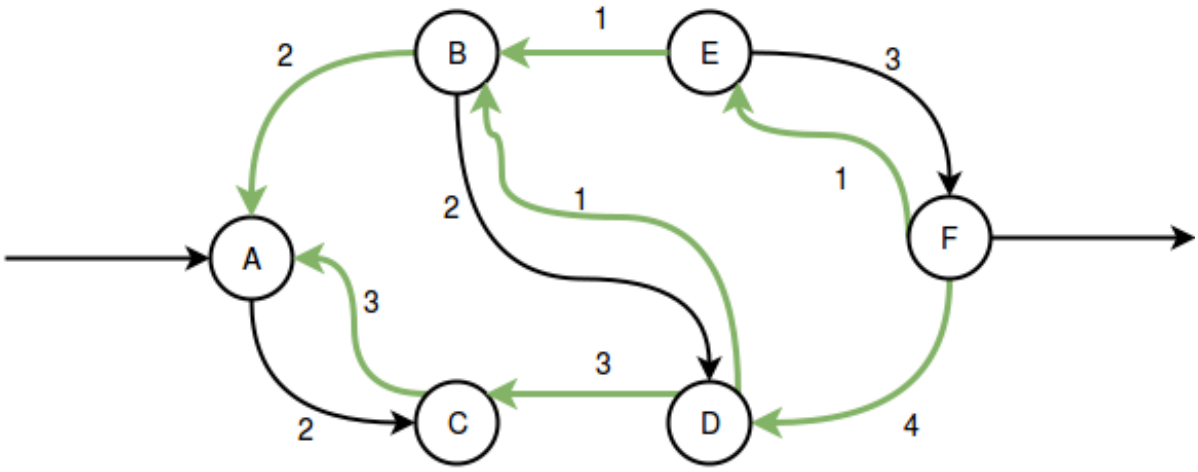
চিত্র-৪: ৩নং চিত্রের গ্রাফের residual গ্রাফ

সবুজ রঙ দিয়ে উল্টো এজ এবং তাদের residual ক্যাপাসিটি দেখানো হয়েছে। ০ ক্যাপাসিটির এজগুলোকে গ্রাফে একে দেখাইনি। A->C এজ দিয়ে আগে ২ফ্লো পাঠানো হয়েছে বলে উল্টো এজ এর residual ক্যাপাসিটি ২, এবং এজটার আরো ৩ ফ্লো পাঠানোর ক্ষমতা আছে তাই মূল এজ এর residual ক্যাপাসিটি ৩। ঠিক এভাবে অন্যান্য এজগুলো আঁকা হয়েছে, তুমি নিজে একে যাচাই করে নিতে পারো ঠিকমত বুঝেছো নাকি বা আমার ছবিতে ভুল আছে নাকি।

এখন উল্টা এজ এ ফ্লো পাঠানোর মানে কি? আমরাতো পাইপের উল্টো দিক দিয়ে পানি পাঠাতে পারবো না বা রাস্তার উল্টো দিক দিয়ে গাড়ি চালাতে পারবো না। উল্টো দিকে ফ্লো পাঠানোর মানে হলো মূল ফ্লো টাকে বাতিল করে দেয়া! আমরা যদি D->B এজ দিয়ে ২ফ্লো পাঠাই তার মানে B->D এজ দিয়ে আসা ২লিটার পানির প্রবাহকে আমরা বাতিল করে দিচ্ছি। তখন residual গ্রাফে মূল এজের residual ক্যাপাসিটি যাবে বেড়ে, এবং উল্টা এজের residual ক্যাপাসিটি যাবে কমে, যোগফল আগের মতই থাকবে!

এখন চিন্তা করে দেখো চিত্র-৪ দিয়ে আরো ফ্লো সিংক এ পাঠানো যায় নাকি। মনে রাখবে চিত্র-৩ আর চিত্র-৪ এ একই গ্রাফকেই ভিন্নভাবে আঁকা হয়েছে।

চিত্র ৪ এ দেখা যাচ্ছে A->C->D->B->E->F পথে আরো ১ ফ্লো পাঠানো সম্ভব! এই পথে D->B হলো উল্টো এজ। তারমানে D->B এজ এর ১ ফ্লো বাতিল করে দিয়ে আমরা গ্রাফে মোট ফ্লো ১বাড়িয়ে ফেলতে পারছি। এখন residual গ্রাফ কিরকম হবে দেখি:



চিত্র-৫: A->C->D->B->E->F পথে ফ্লো পাঠানোর পর residual গ্রাফ

তুমি যদি এ পর্যন্ত বুঝে থাকো তাহলে চিত্র-৫ থেকে মূল গ্রাফটাও নিজে একে নিতে পারবে। চিত্র-৪ এ C->D এজ দিয়ে ১ ফ্লো পাঠানো হয়েছে। তাই চিত্র-৫ এ C->D এজের residual ক্যাপাসিটি কমে ১-১=০ হয়ে গিয়েছে এবং D->C এজের residual ক্যাপাসিটি বেড়ে ২+১=৩ হয়ে গিয়েছে। এভাবে সবগুলো এজ আঁকা হয়েছে।

এই গ্রাফে সোর্স থেকে সিংক এ পানি পাঠানোর আর কোনো পথ নেই। আমরা ৩টি পথে মোট ২+২+১=৫ ফ্লো পেয়েছি, অর্থা পানি প্রবাহের হার প্রতি সেকেন্ডে ৫ লিটার। এটাই গ্রাফটার জন্য ম্যাক্সিমাম ফ্লো।

residual গ্রাফে আমরা যখন একটা পথ বের করি সেই পথের একটা বিশেষ নাম আছে, সেটা হলো **অগমেন্টেড পাথ(Augmented path)**। তাহলে আমাদের অ্যালগোরিদম খুব সহজ, যতক্ষণ সম্ভব আমরা residual graph এ একটা অগমেন্টেড পাথ খুজে বের করবো এবং সেই পথে ফ্লো পাঠিয়ে দিবো! এটার নাম Ford-Fulkerson অ্যালগোরিদম। এটার সুডোকোড এরকম:

Max flow pseudo-code

```
Input: A graph G = (V,E) with flow capacity cap, source node s and sink node t.
Output: Calculate maximum flow from s to t.
Initialize:
   1. total_flow=0
   2. Residual Capacity Cf(u,v)=cap(u,v) for each edge(u,v) in the graph
Algorithm:
   1. While there is a path from s to t such that Cf[u][v]>0 for all edges (u,v) in path:
      1.1. min_res_cap = minimum residual capacity among all the edges (u,v) in the
path.
      1.2. For each edge (u,v) in the path:
         Cf(u,v) = Cf(u,v) - min_res_cap
         Cf(v,u) = Cf(v,u) + min_res_cap
      1.3. total_flow = total_flow + min_res_cap
   3. Return total_flow
```

সহজ কথায় একটা ২-ডি অ্যারে $C_f[u][v]$ তে প্রতিটা এজের residual ক্যাপাসিটি থাকবে, যে এজগুলো গ্রাফে নাই সে এজের residual ক্যাপাসিটি ০। এখন তুমি সোর্স থেকে সিংক এ যাবার এমন একটা পথ বের করবে যে পথের প্রতিটি এজের residual ক্যাপাসিটি ০ থেকে বড়। পথের এজগুলোর মধ্যে মিনিমাম residual ক্যাপাসিটির মান খুজে বের করবে। ফ্লো পাঠানোর পর প্রতিটা এজের residual ক্যাপাসিটি কমে যাবে এবং উল্টা এজের residual ক্যাপাসিটি বেড়ে যাবে। মিনিমাম residual ক্যাপাসিটিটা মোট ফ্লো এর মানের সাথে যোগ হতে থাকবে। সবশেষে চাইলে তুমি আসল ক্যাপাসিটি থেকে residual capacity বিয়োগ করে কোনো এজ এ ফ্লো এর পরিমাণ নির্ণয় করতে পারো।

গ্রাফটা ডিরেক্টেড হতে হবে এমন কোনো কথা নেই। বাইডিরেকশনাল এজও থাকতে পারে। A-B বাইডিরেকশনাল এজের ক্যাপাসিটি ১০ হলে $C_f[A][B] = C_f[B][A]$=১০ হবে। এখন আগের মতই অগমেন্টেড পাথ বের করে ম্যাক্সিমাম ফ্লো বের করতে পারবে। তবে এক্ষেত্রে কোন নোডে ফ্লো কত হচ্ছে সেটা বের করতে হলে তোমাকে আরেকটু বুদ্ধি খাটাতে হবে।

অ্যালগোরিদমে ২নম্বর ধাপে পাথ বের করার জন্য বিএফএস/ডিএফএস/বেলম্যান ফোর্ড ইত্যাদি অ্যালগোরিদম ব্যবহার করা যেতে পারে। প্রবলেমের ধরণ অনুযায়ী এই ধাপে একেক অ্যালগোরিদম একেক ধরণের সুবিধা দিবে, সেগুলো তুমি প্রবলেম সলভ করতে করতে জানতে পারবে। আর পাথ বের করার জন্য বিএফএস বা যেটাই ব্যবহার করোনা কেন, প্রতিটা নোডে যাবার সময় নোডটার প্যারেন্ট কে সেটা মনে রাখতে হবে কোনো একটা অ্যারেতে।

তুমি যদি বিএফএস ব্যবহার করে সমাধান করো তাহলে সেটাকে বলা হয় **এডমন্ড কার্প** অ্যালগোরিদম। ফোর্ড-ফুলকার্সন অ্যালগোরিদমে শুধু পাথ খুজে বের করার কথা বলে হয়েছে, সেটা যেকোনোভাবে বের করা যেতে পারে, আর এডমন্ড-কার্প বিএফএস ব্যবহার করে কাজটা করতে বলা হয়েছে অর্থা ফোর্ড-ফুলকার্সন ইমপ্লিমেন্ট করার একটা উপায় হলো এডমন্ড কার্প অ্যালগোরিদম যার কমপ্লেক্সিটি ***O(VE^2)***।

এখন তোমাদের জন্য চিন্তা করার মত কয়েকটা প্রশ্ন:

প্রশ্ন ১: আমাদের প্রবলেমে সোর্স এবং সিংক ছিলো একটা। কিন্তু গ্রাফে একাধিক নোড দিয়ে পানি প্রবেশ করলে এবং একাধিক নোড দিয়ে পানি বের হয়ে গেলে কিভাবে অ্যালগোরিদমটা পরিবর্তন করবে?

প্রশ্ন ২: যদি প্রতিটা নোডের কিছু ক্যাপাসিটি থাকে, অর্থা একটা নোড দিয়ে কত সর্বোচ্চ ফ্লো পাঠানো যাবে সেটা নির্দিষ্ট করা থাকে তাহলে কিভাবে সমাধান করবে?

প্রশ্ন ৩: দুটি নোডের মধ্যে একাধিক এজ থাকলে কি করবে?

প্রশ্ন ৪: দুই বন্ধু একই নোড থেকে যাত্রা শুরু করে একই গন্তব্যে পৌছাতে চায় কিন্তু দুইজনেই চায় ভিন্ন ভিন্ন রাস্তা ব্যবহার করে যেতে, তারমানে একই এজ কখনো ২জন ব্যবহার করতে পারবে না। গ্রাফটি দেয়া হলে তুমি কি বলতে পারবে এরকম ২টি পথ আছে নাকি? এ ধরণের পথকে **এজ ডিসজয়েন্ট পাথ** বলে।

এই প্রশ্নগুলো নিয়ে আলোচনা করা হয়েছে পরের পর্বে।

প্রোগ্রামিং কনটেস্টে ম্যাক্সিমাম ফ্লো প্রবলেম এ কোডিং এর থেকে অনেক কঠিন হলো গ্রাফটা কিভাবে বানাতে হবে সেটা বের করা, তাই অনেক সমস্যা সমাধান করে প্র্যাকটিস করতে হবে। শুরু করার জন্য কয়েকটা প্রবলেম দিয়ে দিলাম:

http://lightoj.com/volume_showproblem.php?problem=1153

http://lightoj.com/volume_showproblem.php?problem=1155

http://uva.onlinejudge.org/external/110/11045.html

হ্যাপি কোডিং!

(এখন থেকে কোনো লেখায় সি++/জাভায় সোর্স-কোড দেয়া হবে না এবং আগের লেখাগুলোর সি++ সোর্স-কোডগুলোও ধীরে ধীরে সরিয়ে নেয়া হবে, তুমি যদি অ্যালগোরিদমটা বুঝে থাকো তাহলে নিজেই কোড লিখতে পারবে। আর বুঝতে সমস্যা হলে বা বিশেষ কোনো কারণে কোড চাইলে সরাসরি আমার সাথে যোগাযোগ কর)

গ্রাফ থিওরিতে হাতেখড়ি-১২ – ম্যাক্সিমাম ফ্লো (২)

shafaetsplanet.com/

শাফায়েত

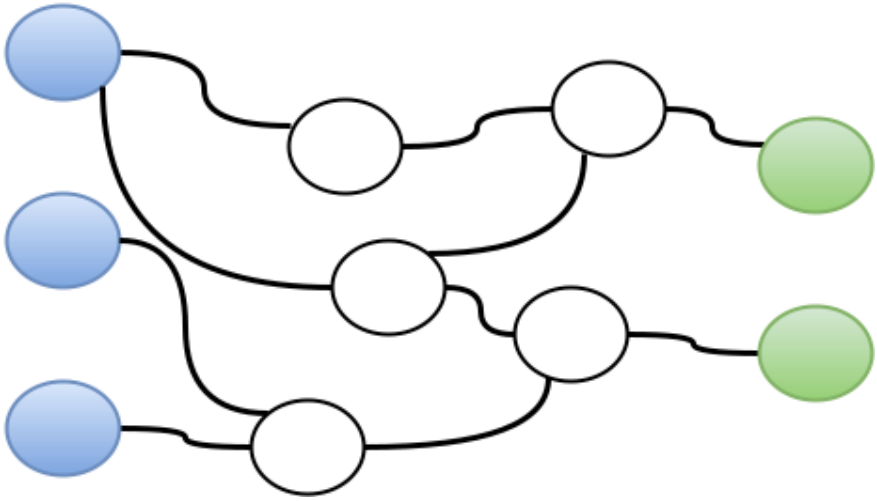
March 30, 2015

আগের পর্বে আমরা দেখেছি কিভাবে ফোর্ড-ফুলকারসন পদ্ধতি ব্যবহার করে ম্যাক্সিমাম ফ্লো বের করতে হয়। এই পর্বে ম্যাক্সিমাম ফ্লো সমস্যার সহজ কিছু ভ্যারিয়েশন দেখবো।

একাধিক সোর্স/সিংক:

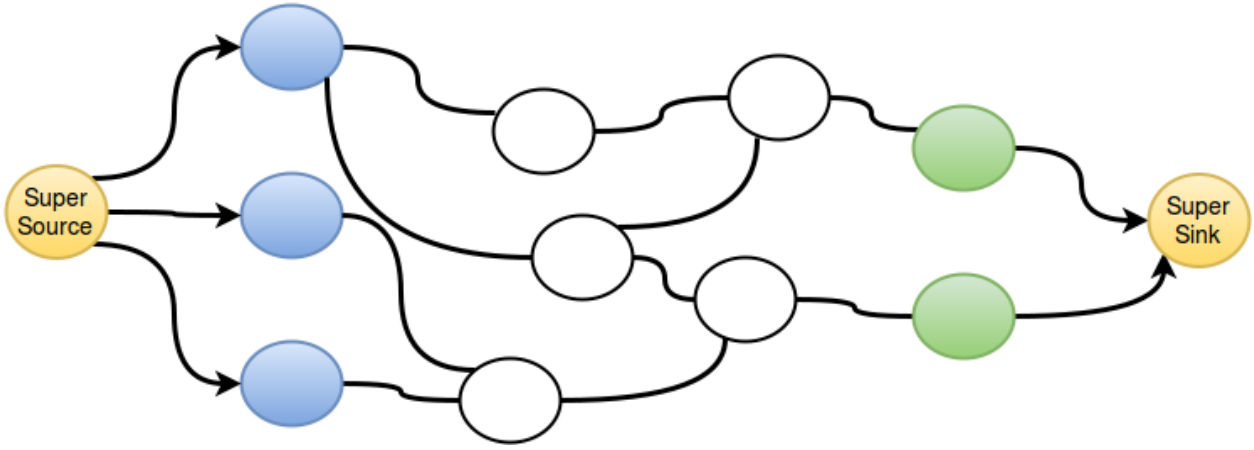
আগের পর্বে একটা প্রশ্ন করেছিলাম এরকম “আমাদের প্রবলেমে সোর্স এবং সিংক ছিলো একটা। কিন্তু গ্রাফে একাধিক নোড দিয়ে পানি প্রবেশ করলে এবং একাধিক নোড দিয়ে পানি বের হয়ে গেলে কিভাবে অ্যালগোরিদমটা পরিবর্তন করবে?” অর্থা একাধিক সোর্স বা সিংক থাকলে কি করতে হবে সেটা জানতে চাওয়া হয়েছে।

চিত্র-১ এ বাম পাশের নীল নোডগুলো হলো সোর্স এবং ডানের সবুজ নোডগুলো হলো সিংক।



চিত্র -১: একাধিক সোর্স এবং সিংক সহ একটি গ্রাফ

এ ধরণের গ্রাফে এডমন্ড কার্প অ্যালগোরিদম প্রয়োগ করার সহজ উপায় হলো সুপার-সোর্স এবং সুপার সিংক বানিয়ে নেয়া। সুপার সোর্স হলো এমন একটা নোড যেটা সবগুলো সোর্সের সাথে ডিরেক্টেড এজ দিয়ে যুক্ত। ঠিক সেভাবে, সুপার সিংক প্রতিটি সিংকের সাথে ডিরেক্টেড এজ নিয়ে সংযুক্ত। এবং এই এজগুলোর ক্যাপাসিটি হবে অসীম বা ইনফিনিটি।



চিত্র-২: সুপার সোর্স এবং সুপার সিংক

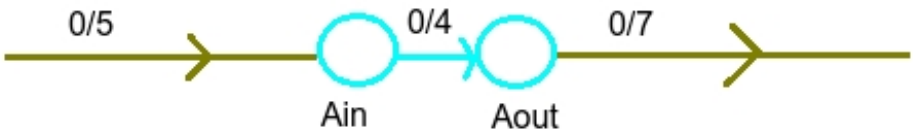
চিত্র-২ তে সুপার সোর্স এবং সুপার সিংক দেখানো হয়েছে। ইনফিনিটি হিসাবে বেছে নিতে পারো সবগুলো এজের সম্মিলিত ক্যাপাসিটির থেকে বড় কোনো মানকে। এখন সাধারণ ফ্লো অ্যালগোরিদম ব্যবহার করেই এই গ্রাফে ম্যাক্সিমাম ফ্লো বের করতে পারবে।

নোড ক্যাপাসিটি:

এতক্ষণ সবগুলো গ্রাফে এজের নির্দিষ্ট ক্যাপাসিটি ছিলো, নোডের ছিলো না। কিন্তু বাস্তবে অনেক সমস্যায় নোডের ক্যাপাসিটি থাকতে পারে। যেমন ধরো কোনো একটা দেশে প্রতিটা রাস্তার পাশাপাশি প্রতিটা শহরের নির্দিষ্ট গাড়ি ধারণ ক্ষমতা আছে, সেই দেশের গ্রাফ চিত্র-৩ এর মতো হতে পারে:

চিত্র-৩: নোড ক্যাপাসিটি

ছবিতে পুরো গ্রাফটা না একে শুধু একটা নোড আর ২টি এজ একেছি, নোডটাতে ঢোকার এজ এর ক্যাপাসিটি ৫, যে এজটি বাইরে চলে গেছে তার ক্যাপাসিটি ৭, এদিকে নোডের নিজের ক্যাপাসিটি ৪। আগে শেখা অ্যালগোরিদমে আমরা এজের ক্যাপাসিটির হিসাব রাখার জন্য একটা অ্যারে ব্যবহার করেছিলাম, এখনও আমরা সেই অ্যারেটা ব্যবহার করেই কাজ করতে পারবো, বুদ্ধিটা হলো নোডটাকে দুই ভাগে ভাগ করে ফেলা, এবং ভাগ দুটিকে নতুন এজ দিয়ে যোগ করে দেয়া। চিত্র-৪ দেখলেই পরিস্কার হবে ব্যাপারটা:



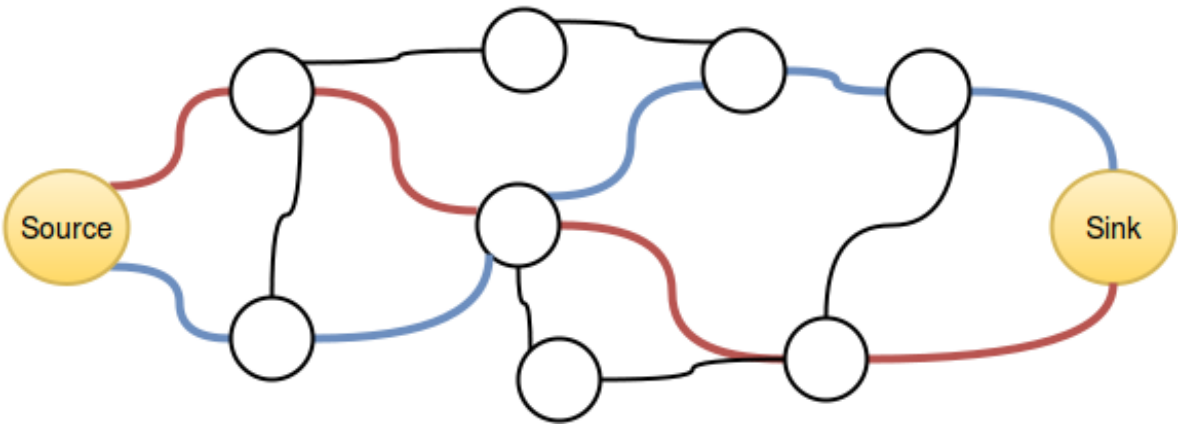
চিত্র-৪: A নোডটিকে দুইভাগ করা হয়েছে

আমরা A নোডটা Ain এবং Aout এই দুটি নোডে ভাগ করেছি। এখন আসল গ্রাফ যতগুলো এজ A তে প্রবেশ করেছে সেগুলো প্রবেশ করবে Ain এ এবং আসল গ্রাফে যতগুলো এজ A থেকে বাইরে গিয়েছে সেগুলো এখন বাইরে যাবে Aout থেকে। Ain থেকে Aout এ একটা এজ প্রবেশ করবে যেটার ক্যাপাসিটি হবে এজ এর ক্যাপাসিটির সমান।

এখন নিশ্চিন্তে তুমি আগের অ্যালগোরিদম ব্যবহার করতে পারো। কোড লেখার সময় কিভাবে নোড দুইভাগ করবে, আনডিরেক্টেড গ্রাফের ক্ষেত্রে ব্যাপারটা কিরকম হবে সেগুলো চিন্তা করা তোমার কাজ!

এজ ডিসজয়েন্ট পাথ:

দুই বন্ধু একই নোড থেকে যাত্রা শুরু করে একই গন্তব্যে পৌছাতে চায় কিন্তু দুইজনেই চায় ভিন্ন ভিন্ন রাস্তা ব্যবহার করে যেতে, তারমানে একই এজ কখনো ২জন ব্যবহার করতে পারবে না। এধরণের পথকে এজ ডিসজয়েন্ট পাথ বলে। তোমাকে বলতে হবে কোনো একটা গ্রাফে দুটি এজ ডিসজয়েন্ট পাথ আছে নাকি। চিত্র-৫ এ একটা উদাহরণ দেখানো হয়েছে:



ছবিতে দুইজনই বামের সোর্স নোডটা থেকে যাত্রা শুরু করে ডানের সিংক নোডে যেতে চায়। লাল এবং নীল রঙ ব্যবহার করে দুটি এজ-ডিসজয়েন্ট পাথ দেখানো হয়েছে।

সাধারণ ম্যাক্স-ফ্লো ব্যবহার করেই এজ ডিসজয়েন্ট পাথ বের করা যায়। শুরুর নোডকে সোর্স এবং গন্তব্য নোডকে সিংক ধরবে। এবার সবগুলো এজ এর ক্যাপাসিটি বানিয়ে দাও ১ এর সমান। এখন যদি তুমি সোর্স থেকে সিংকে দুই ফ্লো পাঠাতো পারো সেটার মানে হলো দুটি

ডিসজয়েন্ট পাথ আছে। প্রতিটা এজের ক্যাপাসিটি ১ হওয়াতে ২ ফ্লো যে দুটি পথে গিয়েছে তাদের মধ্যে কমন এজ থাকা সম্ভব না।

ঠিক একই ভাবে তুমি একটা গ্রাফে সর্বোচ্চ কয়টা ডিসজয়েন্ট পাথ থাকা সম্ভব অথবা দুই বন্ধুর জায়গায় K টা বন্ধু থাকলে কি হতো বের করে ফেলতে পারবে।

এখন প্রশ্ন হলো তুমি যদি প্রতিটা রাস্তার নির্দিষ্ট দৈর্ঘ্য থাকে এবং ডিসজয়েন্ট পাথ দুটির মোট দৈর্ঘ্য মিনিমাইজ করতে চাও তাহলে ফ্লো এর অ্যালগোরিদমটা কিভাবে পরিবর্তন করবে? এটা বের করতে পারলে uva 10806 সমস্যাটা সমাধান করে ফেলো, সমস্যাটার নামের ভিতরেই কিভাবে সমাধান করতে হবে বলা আছে!

আজকের পর্ব এখানেই শেষ। মিন-কাট এবং ম্যাচিং নিয়ে আলোচনার জন্য আরেকটা পর্ব অপেক্ষা করতে হবে। কনটেস্টে ম্যাক্স-ফ্লো প্রবলেমের কঠিন অংশ হলো গ্রাফটা কিভাবে তৈরি করবো, এজগুলো কিভাবে যোগ করবো, কোন এজের ক্যাপাসিটি কত এগুলো বের করা, এসব করার পর ফ্লো অ্যালগোরিদম চালিয়ে দেয়া সহজ কাজ। তাই তোমাকে প্রচুর প্র্যাকটিস করে এই জিনিসগুলো আয়ত্বে আনতে হবে।

কিছু প্রবলেম:

Down Went Titanic
Clever Naming Pattern
Diagonal Sum

হ্যাপি কোডিং!

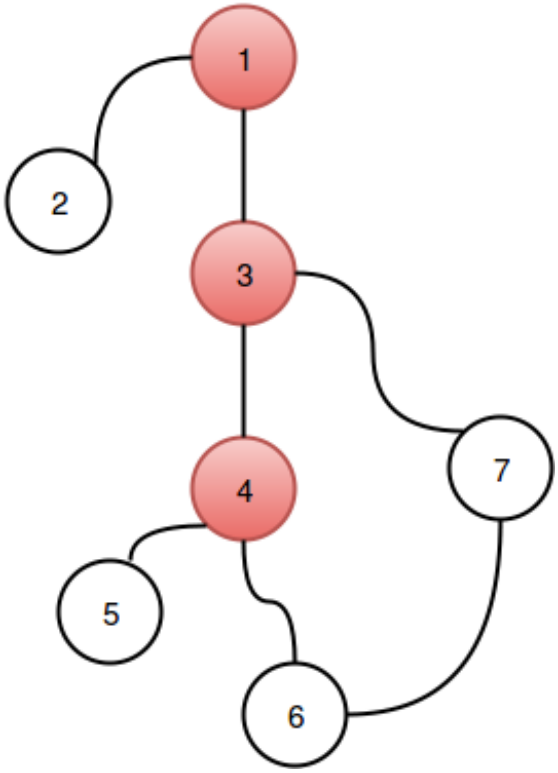
গ্রাফ থিওরিতে হাতেখড়ি ১৩: আর্টিকুলেশন পয়েন্ট এবং ব্রিজ

shafaetsplanet.com/

শাফায়েত নভেম্বর ২৮, ২০১৫

আর্টিকুলেশন পয়েন্ট হলো আনডিরেক্টেড গ্রাফের এমন একটা নোড যেটা গ্রাফ থেকে মুছে ফেললে বাকি গ্রাফটুকু একাধিক কম্পোনেন্ট এ ভাগ হয়ে যায়।

উপরের ছবিতে ১, ৩ অথবা ৪ নম্বর নোড এবং সেই নোডের অ্যাডজেসেন্ট এজগুলোকে মুছে দিলে গ্রাফটা একাধিক ভাগ হয়ে যাবে, তাই ১, ৩ ও ৪ হলো এই গ্রাফের আর্টিকুলেশন পয়েন্ট। আর্টিকুলেশন পয়েন্টকে অনেকে কাট-নোড(cut node) , আর্টিকুলেশন নোড বা ক্রিটিকাল পয়েন্ট (critical point) ও বলে।



আর্টিকুলেশন পয়েন্ট বের করার একটা খুব সহজ উপায় হলো, ১টা করে নোড গ্রাফ থেকে মুছে দিয়ে দেখা যে গ্রাফটি একাধিক কম্পোনেন্ট এ বিভক্ত হয়ে গিয়েছে নাকি।

```
1    1  procedure articulationPointNaive(G):
2    2      articulation_points=[]
3    3      for all nodes u in G
4    4          G.removeNode(u)
5    5          if
6    get_number_of_component(G)>1
7    6                  articulation_points.add(u)
8    7          end if
9    8          G.addBackNode(u)
10   9      end for
     10     return articulation_points
```

কম্পোনেন্ট সংখ্যা ডিএফএস বা বিএফএস দিয়ে খুব সহজে বের করা যায়। এই পদ্ধতিতে V বার ডিএফএস চালাতে হবে যেখানে V হলো নোড সংখ্যা, মোট কমপ্লেক্সিটি $O(V \times (V+E))$ বা $O(V^3)$ কারণ সর্বোচ্চ এজ সংখ্যা V^2।

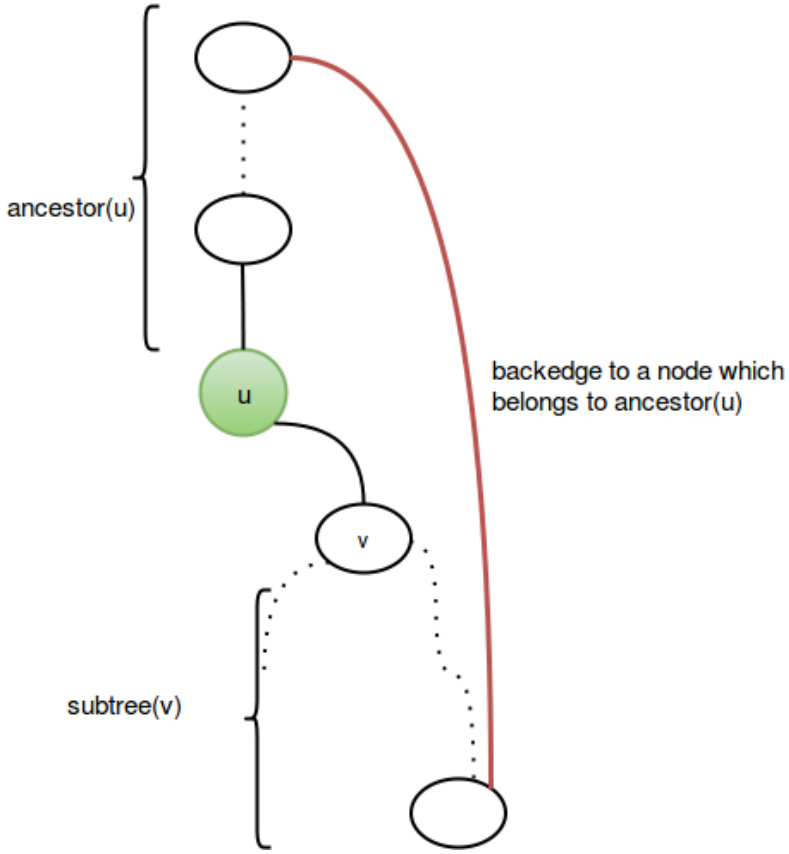
এখন আমরা একবার মাত্র ডিএফএস চালিয়ে আর্টিকুলেশন পয়েন্ট বের করবো। এই অ্যালগোরিদম শেখার জন্য ডিএফএস এর ডিসকভারি/ফিনিশিং টাইম এবং ট্রি এজ ও ব্যাক এজ নিয়ে ধারণা থাকতে হবে।

একটা গ্রাফে ডিএফএস চালালে যেসব ট্রি এজ পাওয়া যায় সেগুলো নিয়ে তৈরি হয়ে ডিএফএস ট্রি।

দুটি কেস থাকতে পারে। যদি একটা নোড ট্রি এর রুট হয় তাহলে একভাবে কাজ করবো, রুট না হলে আরেকভাবে কাজ করবো।

একটা নোড u যদি ট্রি এর রুট হয় এবং ডিএফএস ট্রি তে নোডটার একাধিক চাইল্ড নোড থাকে তাহলে নোডটা আর্টিকুলেশন পয়েন্ট।

রুট ছাড়া বাকি নোডের জন্য কাজটা একটু জটিল।

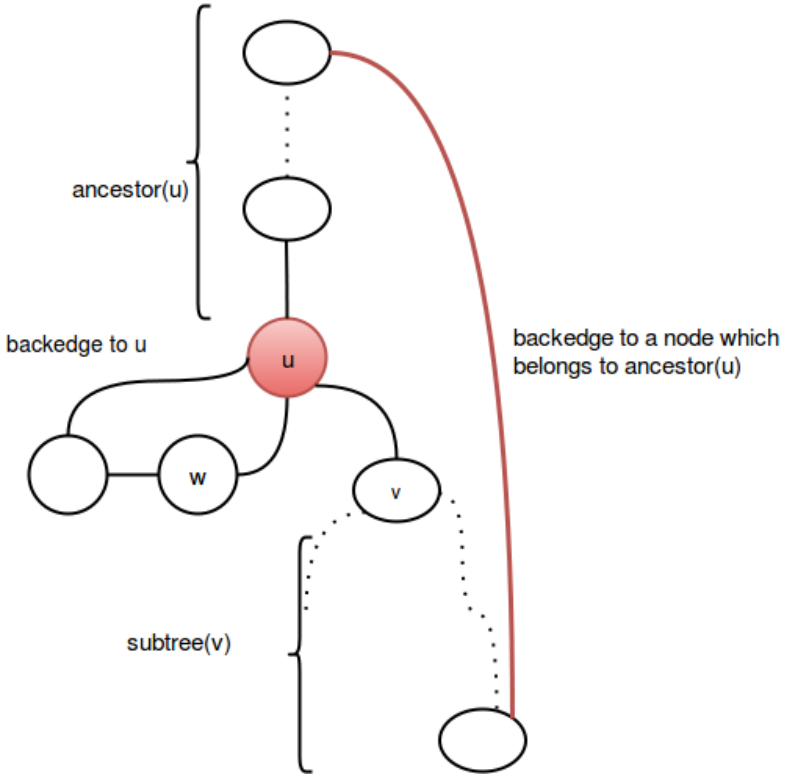


ডিএফএস ট্রি এর একটা এজ $u-v$ এর কথা চিন্তা করো। রুট থেকে u তে আসার পথে যেসব নোড ভিজিট করেছো তাদের আমরা বলবো $ancestor(u)$। এখন v যে সাবট্রি এর রুট সেই সাবট্রির সবগুলো নোডের সেটকে আমরা বলবো $subtree(v)$।

এখন u একটা আর্টিকুলেশন পয়েন্ট হবে যদি মূল গ্রাফে u কে মুছে দিলে $subtree(v)$ এর নোডগুলো একটা আলাদা কম্পোনেন্ট এ পরিণত হয়। $subtree(v)$ আলাদা কম্পোনেন্ট এ পরিণত হবে যদি না মূল গ্রাফে সাবট্রি $subtree(v)$ এর কোনো নোড থেকে ancestor(u) তে একটা ব্যাকএজ থাকে। যদি ব্যাকএজ থাকে তাহলে নোড u এবং অ্যাডজেসেন্ট এজগুলো মুছে গেলেও $ancestor(u)$ থেকে ব্যাকএজ দিয়ে $subtree(v)$ তে পৌছানো যাচ্ছে, নতুন কম্পোনেন্ট তৈরি হচ্ছে না।

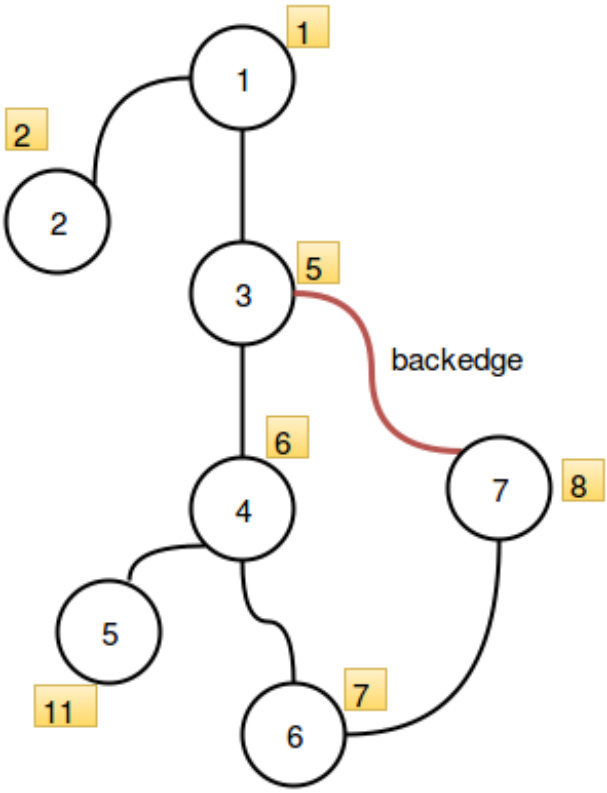
u এর যেকোনো একটা চাইল্ড নোড **v** এর জন্য যদি **$subtree(v)$** থেকে **$ancestor(u)$** তে পৌছানো না যায় , তাহলে **u** আর্টিকুলেশন পয়েন্ট, **u** কে মুছে দিলে সেইসব **$subtree(v)$** নতুন কম্পোনেন্ট এ পরিণত হবে যাদের সাথে **$ancestor(u)$** এর কোনো ব্যাকএজ সংযোগ নেই।

নিচের ছবিতে $subtree(v)$ যদিও ব্যাকএজ দিয়ে $ancestor(u)$ এর সাথে সংযুক্ত, $subtree(w)$ থেকে $ancestor(u)$ তে ব্যাকএজ নেই। তাই u একটা আর্টিকুলেশন পয়েন্ট।



এবার প্রথম গ্রাফটায় ফিরে আসি। গ্রাফের নোডগুলো ১,২,৩,৪,৬,৭,৫ এই অর্ডারে ভিজিট করলে আমরা প্রতিটা নোডের যা ডিসকভারি টাইম পাবো সেটা পাশে ছোটো করে লেখা হয়েছে:

ডিসকভারি টাইম কিভাবে বের করতে হয় না বুঝলে ডিএফএস নিয়ে টিউটোরিয়ালটা দেখো। d[] দিয়ে আমরা ডিসকভারি টাইম বুঝাবো।



গ্রাফের ব্যাকএজ টা লাল এজ দিয়ে দেখানো হয়েছে। বাকি কালো এজগুলো ডিএফএস ট্রি এর অংশ। 1 হলো রুট নোড।

ডিএফএস ট্রি তে রুট নোড 1 এর চাইল্ড সংখ্যা এখানে ২টা (২ এবং ৩)। তাই 1 একটা আর্টিকুলেশন পয়েন্ট।

লক্ষ্য করো নোড $ancestor(u)$ এর যেকোনো নোডের ডিসকভারি টাইম $d[u]$ এর থেকে ছোটো। আবার u এর অ্যাডজেসেন্ট যেকোনো এজ $u-v$ এর জন্য $subtree(v)$ এর সব নোডের ডিসকভারি টাইম $d[u]$ এর থেকে বড়। এখন $subtree(v)$ এর কোনো নোড থেকে যদি এমন একটা ব্যাকএজ $v-w$ থাকে যেন $d[w] < d[u]$ হয় তাহলে বুঝতে হবে তুমি $u-v$ এজ পার হয়ে $subtree(v)$ দিয়ে $ancestor(u)$ তে পৌছে গেছো এবং $w \in ancestor(u)$ । তারমানে u মুছে দিলেও $subtree(v)$ থেকে w তে পৌছানো যাবে।

যেমন 4 নম্বর নোডের কথা চিন্তা করো। 4 এর ডিসকভারি টাইম $d[4]=6$ এবং $ancestor(4)=\{1,2,3\}$। এখন 4-6 এজটার কথা ভাবি। $subtree(6)$ এ একটা ব্যাকএজ $7-3$ আছে, এবং $d[3]=5$ যা $d[4]$ এর থেকে ছোটো। তারমানে $3 \in ancestor(4)$। তাহলে তুমি 4 নোডটা মুছে দিলেও $subtree(6)$ ব্যাকএজের মাধ্যমে $ancestor(4)$ এর সাথে সংযুক্ত থাকবে।

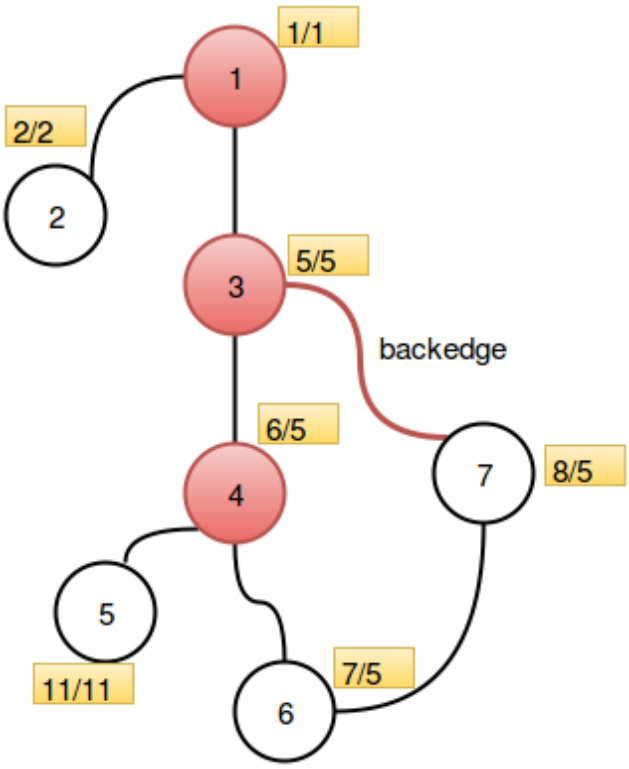
এবার আমরা আরেকটা ভ্যারিয়েবল ডিফাইন করবো $low[u]$। মনে করো $subtree(u)$ এবং $subtree(u)$ এর সাথে ব্যাকএজ দিয়ে সংযুক্ত সবগুলো নোডের একটা সেট বানানো হলো, সেটা টা হলো $\{x1,x2...xm\}$। তাহলে $low[u]$ হবে $min(d[x_1], d[x_2] ...,d[x_m])$।

যেমন 4 নম্বর নোডের জন্য $subtree(u)=\{5,6,7\}$ এবং $subtree(u)$ এর সাথে ব্যাকএজ দিয়ে যুক্ত আছে নোড 3। তাহলে $low[u]=min(d[5], d[6], d[7] , d[3]) = 5$।

এখন চিন্তা করো কোনো একটা এজ $u-v$ এর জন্য d[u]>low[v] হবার অর্থ কি? d[u] এর থেকে ডিসকভারি টাইম ছোটো একমাত্র $ancestor(u)$ সেটের নোডগুলোর। subtree(v) এর কোনো নোড ব্যাকএজ দিয়ে ancestor(u) এর সাথে যুক্ত, সেজন্য low[v] এর মান d[u] এর থেকে কমে গিয়েছে। যদি $d[u] <=low[v]$ হয়, তাহলেই শুধুমাত্র u একটা আর্টিকুলেশন পয়েন্ট হবে।

আগের গ্রাফেই ডিসকভারি টাইমের পাশাপাশি $low[u]$ এর মানগুলোও দেখি:

তাহলে আমরা আর্টিকুলেশন পয়েন্ট বের করার একটা অ্যালগোরিদম পেয়ে গিয়েছি। প্রতিটা নোডের জন্য d[u], low[u] বের করতে পারলেই কাজ শেষ। low[u] বের করা কঠিন কিছু না, সুডোকোড দেখলেই পরিস্কার হবে:



```
1          articulation_point[] ← false
2          visited[] ← false
3          low[]=d[u] ← 0
4          time ← 0
5     1     Procedure FindArticulationPoint(G, u):
6     2     time ←  time+1
7     3     low[u]=d[u] ← time
8     4     visited[u] ← true
9     5     no_of_children ←  0
10    6     for each edge u to v in G.adjacentEdges(u) do
11    7       if(v == parent[u]) continue
12    8       if visited[v] //This is a backedge
13    9         low[u] = min(low[u], d[v])
14    10      end if
15    11      if not visited[v]   //This is a tree edge
16    12        parent[u] = v
17    13        FindArticulationPoint(G, v)
18    14        low[u] = min(low[u], low[v])
19    15        if d[u] <= low[v] and u is not root:
20    16           articulation_point[u]=true
21    17        end if
22    18        no_of_children=no_of_children+1
23    19      end if
24    20      if(no_of_children>1 u is root):
25    21          articulation_point[u]=true
26    22      end if
27    23    end for
```

ব্রিজ জিনিসটা আর্টিকুলেশন পয়েন্টের মতই। গ্রাফ থেকে যে এজ তুলে দিলে গ্রাফটা একাধিক কম্পোনেন্টে ভাগ হয়ে যায় তাকেই বলা হয় ব্রিজ।

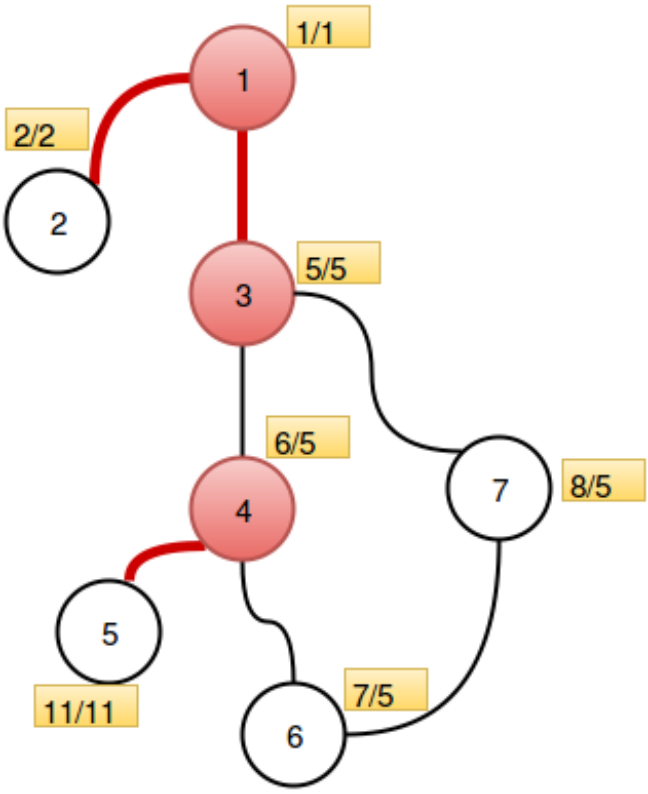
উপরের গ্রাফে $4-5$, $1-2$, আর $1-3$ এই ৩টি এজ হলো ব্রিজ।

ব্রিজ আর আর্টিকুলেশন পয়েন্টের সুডোকোডের পার্থক্য খালি এক জায়গায় ১৫ নম্বর লাইনে d[u]<=low[v] এর জায়গায় d[u]<low[v] লিখতে হবে। এটা কেন কাজ করে তুমি সহজেই বুঝতে পারবে যদি তুমি সুডোকোডটা বুঝে থাকো, তাই আর ব্যাখ্যা করলাম না।

দুটি নোডের মধ্যে একাধিক এজ থাকলে অবশ্য এটা কাজ করবে না। তখন কি করতে হবে সেটা চিন্তা করা তোমার কাজ!

সলভ করার জন্য কিছু প্রবলেম পাবে এখানে।

হ্যাপি কোডিং!

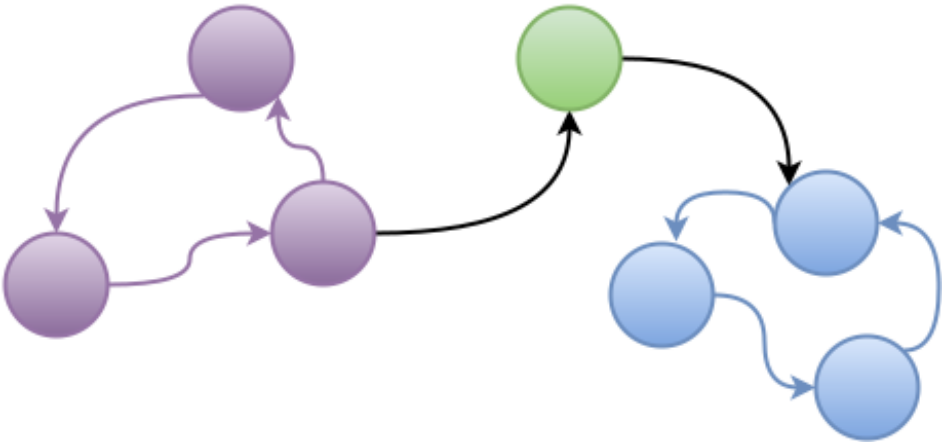


গ্রাফ থিওরিতে হাতেখড়ি ১৪ – স্ট্রংলি কানেক্টেড কম্পোনেন্ট

shafaetsplanet.com/

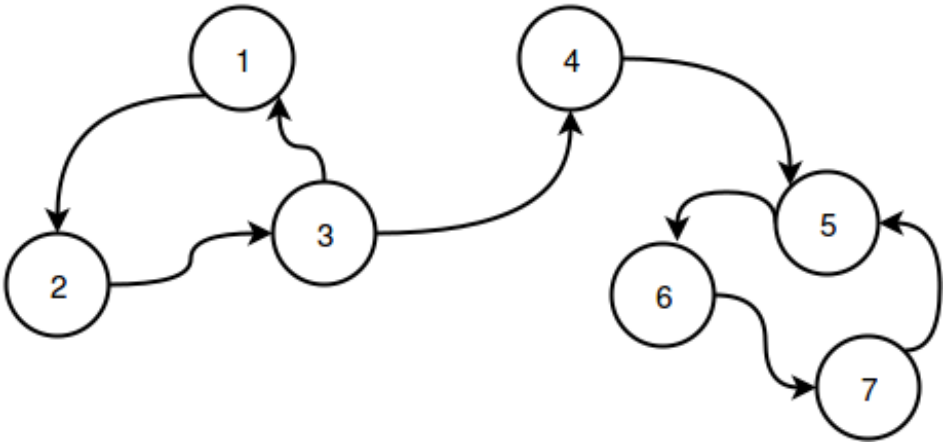
শাফায়েত নভেম্বর ২৯, ২০১৫

একটা ডিরেক্টেট গ্রাফের স্ট্রংলি কানেক্টেড কম্পোনেন্ট বা SCC হলো এমন একটা কম্পোনেন্ট যার প্রতিটা নোড থেকে অন্য নোডে যাবার পথ আছে। নিচের ছবিতে একটা গ্রাফের প্রতিটা স্ট্রংলি কানেক্টেড কম্পোনেন্ট আলাদা রঙ দিয়ে দেখানো হয়েছে।



ডেপথ ফার্স্ট সার্চ এর ফিনিশিং টাইমের ধারণা ব্যবহার করে আমরা $O(V+E)$ তে একটা গ্রাফের স্ট্রংলি কানেক্টেড কম্পোনেন্ট গুলোকে আলাদা করে ফেলতে পারি। এই লেখাটা পড়ার আগে অবশ্যই টপলোজিকাল সর্টিং আর ডেপথ ফার্স্ট সার্চ এর ডিসকভারি এবং ফিনিশিং টাইম সম্পর্কে ধারণা থাকতে হবে।

নিচের গ্রাফটা দেখ:

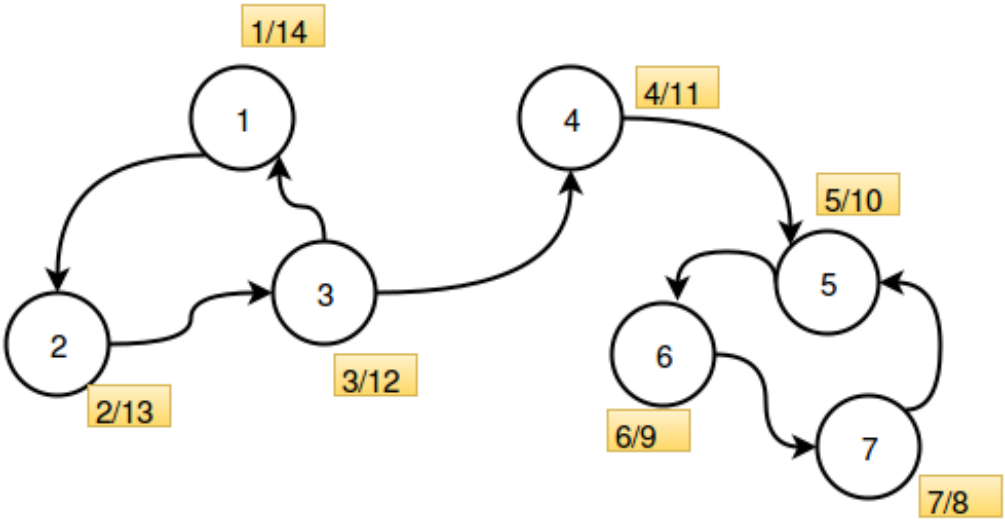


প্রথমেই একটা ভুল পদ্ধতিতে অনেকে SCC বের করার চেষ্টা করে। সেটা হলো যেকোনো নোড থেকে ডিএফএস চালিয়ে যেসব নোডে যাওয়া যায় তাদেরকে একটা কম্পোনেন্ট হিসাবে ধরা। কিন্তু খুব সহজেই বোঝা যায় এটা কাজ করবে না, উপরের গ্রাফে ১ থেকে ডিএফএস চালালে সবগুলো নোড ভিজিট করা যাবে, কিন্তু ১ থেকে ৪ এ যাওয়া গেলেও ৪ থেকে ১ এ যাবার কোনো

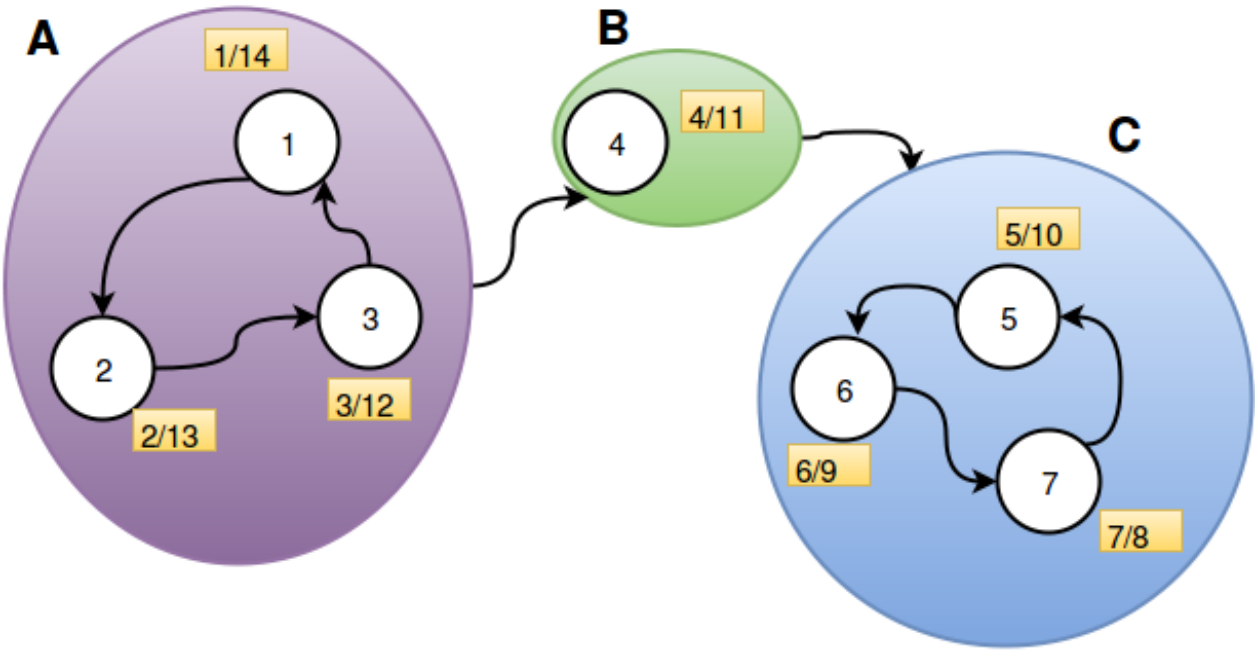
পথ নেই, তাই এরা একই কানেক্টেড কম্পোনেন্ট এর অংশ না। এই পদ্ধতিতে সমস্যা হলো ডিএফএস কানেক্টেড কম্পোনেন্ট থেকে বের হয়ে অন্য কম্পোনেন্ট এ চলে যায়। এই সমস্যা সমাধান করতে আমরা একটু বুদ্ধিমানের মত ডিএফএস চালাবো।

দুটি নোড u,v একই SCC তে থাকবে শুধুমাত্র যদি u থেকে v তে যাবার পথ থাকে এবং v থেকে u তে যাবারও পথ থাকে।

প্রথমে আমরা ১ থেকে ডিএফএস চালিয়ে সবগুলো নোডের ডিসকভারি টাইম আর ফিনিশিং টাইম লিখে ফেলি। নোডগুলো ১,২,৩,৪,৫,৬,৭ অর্ডারে ভিজিট করলে আমরা নিচের ছবির মত স্টার্টিং/ফিনিংশিং টাইম পাবো:



এখন বোঝার সুবিধার জন্য স্ট্রংলি কানেক্টেড কম্পোনেন্টের সবগুলো নোডকে একটা বড় নোড মনে করি:



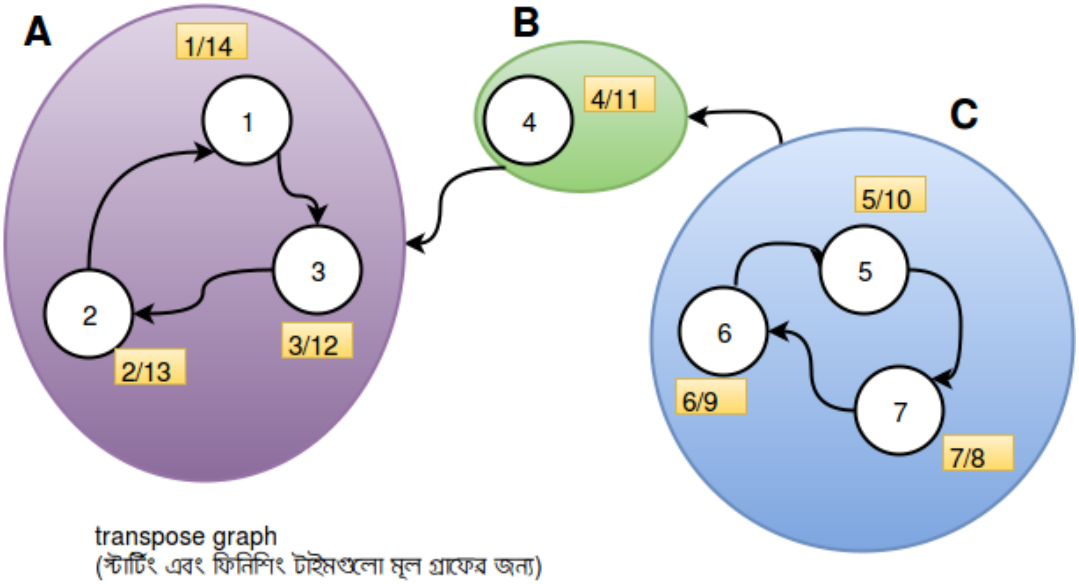
লক্ষ্য করো, গ্রাফটাকে এভাবে ‘ডিকম্পোজ’ করার পর গ্রাফটিতে আর কোনো সাইকেল থাকা সম্ভব না, অর্থা গ্রাফটি একটি ড্যাগ বা ডিরেক্টেড অ্যাসাইক্লিক গ্রাফে পরিণত হয়েছে। এখন বড় নোডগুলোকে সহজেই টপোলজিকাল অর্ডারে সাজানো সম্ভব, অর্ডারটা হবে A,B,C।

এখন লক্ষ্য করো ড্যাগে একটা নোড D1 থেকে অন্য নোড D2 এ যাওয়া যায় তাহলে D1 টপোলজিকাল অর্ডারে D2 এর আগে অবশ্যই থাকবে। আবার আমরা আগেই জানি যে টপোলজিকাল অর্ডারে যে আগে থাকে তার ফিনিশিং টাইম বেশি হয় কারণ অন্যান্য সব নোডের কাজ শেষ করে ওই নোডে ফিরে আসতে হয়।

তাহলে D1 যদি টপোলজিকাল অর্ডারে D2 এর আগে থাকে তাহলে যেসব ছোটো ছোটো নোড নিয়ে D2 গঠিত হয়েছে তাদের সবার ফিনিংশিং টাইম অবশ্যই D1 এর সব নোডের থেকে কম হবে।

এখন u থেকে v তে যাওয়া গেলেই তারা একই SCC এর অন্তর্ভূক্ত না, v থেকে u তে যাবার পথও থাকতে হবে। অথবা আমরা বলতে পারি ‘উল্টো-গ্রাফ’ এও u থেকে v তে যাবার পথ থাকতে হবে!

যদি গ্রাফের এজগুলো উল্টে দেয়া হয়, তাহলেও SCC গুলো একই থাকে । একে বলা হয় ট্রান্সপোজ গ্রাফ(transpose) , ট্রান্সপোজ গ্রাফে সাইকেল গুলোর কোনো পরিবর্তন হয় না। মূল গ্রাফে যদি u-v একই SCC এর মধ্যে থাকে তাহলে তারা অবশ্যই একই সাইকেলের মধ্যে থাকবে। এটাই আমাদের অ্যালগোরিদমের মূল ভিত্তি।



transpose graph
(স্টার্টিং এবং ফিনিশিং টাইমগুলো মূল গ্রাফের জন্য)

উপরের ছবিতে আগের গ্রাফের এজগুলো উল্টে দেয়া হয়েছে। ডিসকভারি এবং ফিনিশিং টাইম আগেরটাই লেখা আছে।

এখন লক্ষ্য করো তুমি যদি শুরুতে টপলোজিকাল অর্ডারে আগে থাকা নোডগুলো থেকে ডিএফএস চালাও অর্থা যার ফিনিশিং টাইম বড় সেখান থেকে শুরু করো তাহলে তুমি প্রথম প্রথম SCC টা পেয়ে যাবে।

উপরের গ্রাফে 1 এর ফিনিশিং টাইম সবথেকে বেশি (14)। 1 থেকে ডিএফএস চালালে তুমি যেতে পারবে {1,2,3} নোডগুলোতে যারা একই SCC'র অংশ। এবার {১,২,৩} নোডগুলো গ্রাফ থেকে মুছে ফেল। এরপর 4 এর ফিনিশিং টাইম বড়। 4 থেকে শুধুমাত্র {4} এ যাওয়া যায়। এরপর 5 থেকে ডিএফএস চালাবো, সেখান থেকে যাওয়া যায় {5,6,7} নোডগুলোতে যারা একটি SCC এর অংশ।

যেসব নোডগুলো একই কম্পোনেন্ট এর অংশ তাদের কে আমরা আলাদা লিস্টে সেভ করে রাখবো নিজের সুডোকোডটা দেখো:

```
1   1    procedure DFS(G, u):
2   5        color[u]  ← GREY
3   6        for all edges from u to v in G.adjacentEdges(u) do
4   7            if color[v]=WHITE
5   8                DFS(G,v)
6   9            end if
7   10       end for
8   11       stk.add(source)
9   13       return
10  14   procedure DFS2(R,u, mark)
11  15       components[mark].add(u) //save the nodes of the new component
12  16       visited[u] ← true
13  17       for all edges from u to v in R.adjacentEdges(u) do
14  18           if visited[v] ← false
15  19               DFS2(R,v, mark)
16  20           end if
17  21       end for
18  22       return
19
20  23    procedure findSCC(G):
21  24       stk ← an empty stack
22  25       visited[] ← null
23  26       color[] ← null
24  27       components[] ← null
25  28       mark=0
26  29       for each u in G
27  30           if color[u]=WHITE
28  31               DFS(G,u)
29  32           end if
30  33       end for
31  34       R=reverseEdges(G)
32  35       while stk not empty
33  36           u=stk.removeTop()
34  37           if visited[u]=false
35  38               mark=mark+1 //A new component found, it will be identified by 'mark'
36  39               DFS2(R,u,mark)
37  40           end if
38  41       end for
39  42       return components
40
```

কোডটা একটু বড় মনে হলেও বোঝা খুব সহজ। প্রথমে একটা ডিএফএস চালিয়ে ফিনিশিং টাইম অনুযায়ী নোডগুলো সর্ট করছি। একটা স্ট্যাক ব্যবহার করে কাজটা করছি। যার ফিনিশিং টাইম কম সে কাজ আগে শেষ করে ১১ নম্বর লাইনে আসবে, তখন সেই নোডটা স্ট্যাকে ঢুকিয়ে রাখবো। সবশেষে স্ট্যাকের উপরে যে নোড থাকবে তার ফিনিশিং টাইম হবে সব থেকে বেশি। এর

পর ২য় ডিএফএস চালিয়ে কম্পোনেন্টগুলো আলাদা করে ফেলবো। mark নামের ভ্যারিয়েবল টা ব্যবহার করছি প্রতিটা কম্পোনেন্ট এর আলাদা নাম দেয়ার জন্য, ছবিতে যেভাবে A,B,C নাম দেয়া হয়েছে।

সলভ করার জন্য কিছু প্রবলেম পাবে এখানে।

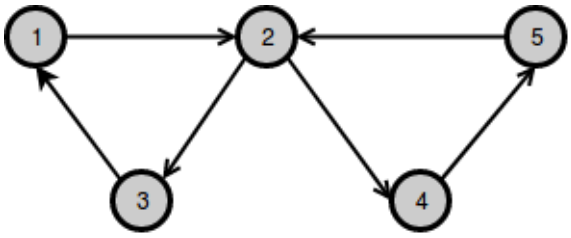
অয়লার ট্যুর (ফ্লিরি এবং হেয়ারহজলার অ্যালগরিদম)

shafaetsplanet.com/

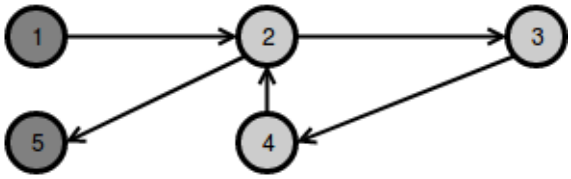
শাফায়েত ফেব্রুয়ারি ২৫, ২০১৮

আমরা অনেকেই জানি অয়লার সার্কিট/পাথ কী এবং গ্রাফে অয়লার সার্কিট/পাথ আছে নাকি সেটা কিভাবে বের করতে হয়, কিন্তু গ্রাফে যদি অয়লার সার্কিট/পাথ থেকে থাকে তাহলে সেটা কিভাবে খুজে বের করা যায় সেটা অনেকেই জানিনা। আজকে আমরা সেটাই শিখবো।

অয়লার সার্কিট এবং পাথ কী সেটা একটু মনে করা নেয়া যাক। অয়লার সার্কিট হলো গ্রাফের এমন একটা পাথ যেটা যে নোডে শুরু হয়েছে সে নোডেই শেষ হয়েছে কিন্তু প্রতিটি এজ ঠিক একবার ব্যবহার করেছে। আর অয়লার পাথ হলো গ্রাফে এমন একটি পাথ যেটা যেকোনো একটি নোডে শুরু হয়ে অন্য একটি নোডে শেষ হয়েছে কিন্তু প্রতিটি এজ ঠিক একবার ব্যবহার করেছে।



Euler circuit = 1->2->4->5->2->3->1



Euler path = 1->2->3->4->2->5

- একটি ডিরেক্টেড গ্রাফে অয়লার সার্কিট থাকবে যদি গ্রাফটি কানেক্টেড হয় এবং প্রতিটি নোডের আউটডিগ্রি এবং ইনডিগ্রি সমান হয়।
- একটি আনডিরেক্টেড গ্রাফে অয়লার সার্কিট থাকবে যদি গ্রাফটি কানেক্টেড হয় এবং প্রতিটি নোডের ডিগ্রী জোড় সংখ্যা হয়।
- একটি ডিরেক্টেড গ্রাফে অয়লার পাথ থাকবে শুধুমাত্র যদি গ্রাফটি কানেক্টেড হয় এবং ২টি মাত্র নোডের আউটডিগ্রি এবং ইনডিগ্রির পার্থক্য 1 হয় এবং বাকি সব নোডের আউটডিগ্রি এবং ইনডিগ্রি সমান হয়। শুরুর নোডের ক্ষেত্রে $indegree = outdegree – 1$ আর শেষের নোডের ক্ষেত্রে $outdegree = indegree – 1$ হবে।
- একটি আনডিরেক্টেড গ্রাফে অয়লার পাথ থাকবে যদি গ্রাফটি কানেক্টেড হয় ২টি মাত্র নোডের ডিগ্রি বিজোড় সংখ্যা হয় (শুরু এবং শেষের নোড) এবং বাকি সব নোডের ডিগ্রি জোড় সংখ্যা হয়।

এখানে ডিরেক্টেড গ্রাফে কানেক্টেড গ্রাফ বলতে বুঝাচ্ছি যদি গ্রাফটিকে আনডিরেক্টেড কল্পনা করো তাহলে প্রতিটি নোড থেকে প্রতিটা নোডে যাবার অন্তত একটি পথ থাকবে।

অয়লার পাথ সমস্যাটি কনিসবার্গ সেতু সমস্যা নামেও পরিচিত, এ ব্যাপারে বিস্তারিত আলোচনা করেছি আমার গ্রাফ অ্যালগরিদম বইয়ে। গ্রাফে অয়লার সার্কিট বা পাথ আছে নাকি সেটা বের করা খুবই সহজ, কিন্তু সার্কিট বা পাথটি খুজে বের করা আরেকটু কঠিন, আজকে সেটা নিয়েই আলোচনা করবো।

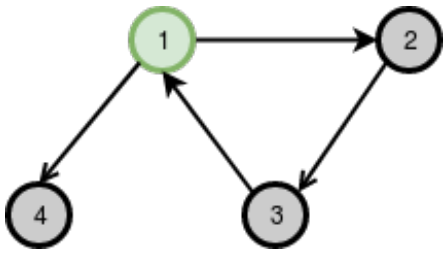
অয়লার সার্কিট/পাথ বের করার জন্য একটি পরিচিত অ্যালগরিদম হলো ফ্লিরি'র(Fleury) অ্যালগরিদম। অ্যালগরদিমটি প্রথম প্রকাশিত হয় ১৮৮৩ সালে, বুঝতেই পারছো গ্রাফ থিওরি বয়স কত পুরানো।

ফ্লিরির অ্যালগরিদমটি বুঝতে হলে আমাদের জানতে হবে গ্রাফে ব্রিজ কাকে বলে। আনডিরেক্টেড ব্রিজ হলো এমন এজ যেটা গ্রাফ থেকে মুছে দিলে গ্রাফটি ডিসকানেক্টেড (disconnected) হয়ে যায়। আমরা যখন ডিরেক্টেড গ্রাফ নিয়ে কাজ করবো তখন ব্রিজ খোজার সময় কল্পনা করে নিব গ্রাফটি আনডিরেক্টেড, অর্থা আমরা 'underlying undirected graph' থেকে ব্রিজ বের করবো।

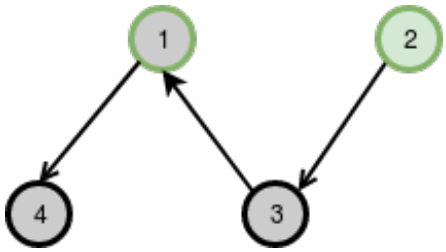
গ্রাফে ফ্লিরি'র অ্যালগরিদম অ্যাপ্লাই করার আগে আগে নোডগুলোর ডিগ্রী দেখে জেনে নিতে হবে গ্রাফে অয়লার সার্কিট আছে কি নেই। যদি অয়লার সার্কিট থেকে থাকে তাহলে আমরা যেকোনো নোড থেকে খোজা শুরু করতে পারি, যদি অয়লার পাথ থাকে তাহলে প্রথম নোডটি থেকে শুরু করতে হবে। কিভাবে বুঝবে কোনটা প্রথম নোড? আনডিরেক্টেড গ্রাফে যেকোন নোড ,যার ডিগ্রী বিজোড় সংখ্যা, কে শুরুর নোড ধরে নিতে পারো। ডিরেক্টেড গ্রাফে যে নোডের আউটডিগ্রী ইনডিগ্রীর থেকে ১ বেশি সেটাই অয়লার পাথের শুরুর নোড।

এখন ধরে নিলাম শুরুর নোডটি হলো u। এবার আমরা u এর অ্যাডজেসেন্ট যেকোনো একটি নোড v নিব এমন ভাবে যেন $u-v$ এজটি মুছে দিলে গ্রাফটি ডিসকানেক্টেড না হয়ে যায় যায়, অর্থা $u-v$ যেন ব্রিজ না হয়। এরপর $u-v$ এজটা মুছে দিয়ে আমরা পরের নোড v নোড থেকে আবার একই কাজ করবো। কিন্তু এমন হতে পারে যে এমন কোনো এজ $u-v$ নেই যেটা একটি ব্রিজ নয়, সেক্ষেত্রে ব্রিজ ধরেই এগিয়ে যাবো। এভাবে যতক্ষণ না সবগুলো এজ মুছে ফেলা হয় ততক্ষণ আগাতে থাকবো। সবগুলো এজ মুছে ফেলা হয়ে গেলে আমরা অয়লার সার্কিট বা পাথও খুজে পাবো।

উদাহরণ হিসাবে উপরের গ্রাফটি দেখ। এই গ্রাফে অয়লার সার্কিট নেই কিন্তু পাথ আছে। এখানে 1 হল শুরুর নোড (ইনডিগ্রি 1, আউটডিগ্রী 2) । শুরুতে আমরা দেখতে পাচ্ছি $1->4$ একটি ব্রিজ, আমরা সেটা বাদ দিয়ে $1->2$ ধরে এগিয়ে যাবো এবং এজটি মুছে দিব।



এখন নোড 2 থেকে একটি মাত্র এজ ধরে যাওয়া যায়, $2->3$ এবং $2->3$ একটি ব্রিজ। যেহেতু আর কোনো অপশন নেই, আমরা $2->3$ ধরেই এগিয়ে যাবো। এভাবে সিমুলেশন করলে দেখা যাবে যে আমাদের পাথ হলো $1->2->3->1->4$। এটাই গ্রাফটির অয়লার পাথ।



অ্যালগরিদমের প্রমাণে যাবো না কিন্তু মূল ধারণাটা হলো একপাশের সবগুলো এজ ঘুরে না আসার আগেই ব্রিজ ভেঙে না দেয়া। তুমি যদি একবার ব্রিজ ভেঙে ফেলো তাহলে গ্রাফটি ডিসকানেক্টেড হয়ে যাবে, তুমি ব্রিজের অন্যপাশের নোডে আর ফিরে আসতে পারবে না। তাই আমরা বুদ্ধি করে আগে সব নন-ব্রিজ ঘুরে আসবো, যখন দেখবো আর সেরকম এজ নাই তখনই শুধু ব্রিজ ভাঙবো।

ফ্লিরি'র অ্যালগরিদম বোঝা সহজ হলেও ইমপ্লিমেন্ট করা কিছুটা ঝামেলার কারণ প্রতিবার এজ মুছে দেয়ার পর ব্রিজ খুজে বের করতে হয়। তুমি যদি প্রতিবার টারজানের ব্রিজ খুজে বের করার অ্যালগরিদম চালাও তাহলে তুমি E বার টারজান অ্যালগরিদম চালাচ্ছো এবং কমপ্লেক্সিটি হয়ে যাচ্ছে $O(E^2)$। তাই আমরা এত ঝামেলায় না গিয়ে অন্য একটি সহজ অ্যালগরিদম ইমপ্লিমেন্ট করবো। তাহলে কেন আমরা ফ্লিরি'র অ্যালগরিদম শিখলাম? কারণ একটা তোমাকে সাইকেল, ব্রিজ, কানেক্টেড কম্পোনেন্ট ইত্যাদি নিয়ে ভাবাবে এবং হয়তো তোমাকে কোনো একদিন অন্য একটা সমস্যা সমাধান করতে সাহায্য করবে।

এখন আমরা শিখবো হেয়ারহলজারের (Heirholzer) অ্যালগরিদম। এই অ্যালগরিদম মূলত অয়লার সার্কিট বের করার জন্য কিন্তু একটু বুদ্ধি খাটিয়ে অয়লার পাথও বের করা যায়, সে কথায় পরে আসছি। একটি গ্রাফে অয়লার সার্কিট থাকলে প্রতিটা নোডের ইনডিগ্রি এবং আউটডিগ্রি সমান হয়। এখন আমাদের পর্যবেক্ষণ শক্তিকে কাজে লাগাতে হবে।

গ্রাফে অয়লার সার্কিট থাকলে প্রতিটা নোড অবশ্যই একটা সাইকেলের অন্তর্গত হতে হবে। প্রতিটা নোড থেকে তুমি যখন বের হবে তখন অবশ্যই একটা সাইকেল ঘুরে সেই নোডে ফিরে আসার পথ থাকবে। এবং সবথেকে গুরুত্বপূর্ণ পর্যবেক্ষণ হলো **অয়লার সার্কিট গ্রাফ থেকে একটি সাইকেলের সবগুলো এজ মুছে দিলে বাকি যে কানেক্টেড গ্রাফটি থাকবে সেটা নতুন আরেকটি অয়লার সার্কিট হবে**, কারণ যখন তুমি সাইকেলের এজগুলো মুছে দিচ্ছ তখন সাইকেলের অন্তর্ভুক্ত সবগুলো নোডের ইনডিগ্রি এবং আউটডিগ্রি ১ কমছে।

আমাদের কাজ হবে একটা করে সাইকেল ডিটেক্ট করা এবং সাইকেলের এজগুলোকে মুছে ফেলা। কাজটি আমরা রিকার্সিভলি করবো যেন একটি নোড যতগুলো সাইকেলের সাথে যুক্ত সবগুলো সাইকেলের এজ মুছে ফেলা যায়।

euler tour

```
1   tour_stack = empty stack
2   find_circuit(u):
3       for all edges  u->v in G.adjacentEdges(v) do:
4           remove u->v
5           find_circuit(v)
6       end for
7       tour_stack.add(u)
8       return
9
```

উপরের সুডোকোডে কি হচ্ছে বোঝার চেষ্টা করো। এখানে আমরা প্রতিটি নোড থেকে একটি করে এজ মুছে ফেলছি এবং রিকার্সিভলি সেই কম্পোনেন্টের সবগুলো এজ মুছে ফেলছি। সবশেষে নোডটি স্ট্যাকে যোগ করছি। সবকাজ শেষ হওয়ার পর স্ট্যাক থেকে নোডগুলো বের করে নিলেই আমরা অয়লার সার্কিট পেয়ে যাবো।

অ্যালগরিদম বুঝতে সমস্যা হলে বোঝার সবথেকে ভালো উপায় হলো একটি গ্রাফে সিমুলেট করে ফেলা। একবার খাতা-কলমে সিমুলেট করে ফেললেই জিনিসটা পরিস্কার হয়ে যাবে। আমি এখানে সিমুলেশনটা দেখাতে পারতাম, কিন্তু তাহলে তোমার চিন্তা করার অভ্যাস তৈরি হবে না। আশা করি তুমি এটা নিজে নিজে করতে পারবে।

আগে বলেছি এই অ্যালগরিদমটা অয়লার সার্কিট বের করার জন্য, তাহলে অয়লার পাথ কিভাবে বের করা যায়? খুব সহজ, প্রথম নোড থেকে শেষ নোডে একটি ডামি এজ যোগ করে সার্কিট খুজে বের করলেই পাথও পেয়ে যাবে। এই অ্যালগরিদমের কমপ্লিক্সিটি $O(E)$, অ্যালগরিদমটি আনডিরেক্টেড গ্রাফেও কাজ করবে।

মনে করো একজন পোস্টম্যান প্রতিদিন সকালে পোস্টঅফিস থেকে সাইকেল নিয়ে বের হয়ে বিভিন্ন রাস্তায় চিঠি দিয়ে আবার পোস্টঅফিসে ফিরে আসে। কোন পথে গেলে তার সবথেকে কম কষ্ট করতে হবে? যদি রাস্তাগুলোকে একটি গ্রাফ চিন্তা করা হয় এবং গ্রাফটিতে অয়লার সার্কিট থাকে তাহলে এক রাস্তায় কখনোই দুইবার যেতে হবে না। কিন্তু বাস্তবে বেশিভাগ ক্ষেত্রেই সেটা সম্ভব না। সেক্ষেত্রে এক রাস্তা একাধিকবার ব্যবহার করলেও চেষ্টা করা হয় মোট দূরত্ব কমিয়ে আনার। এই প্রবলেমের একটি নাম আছে, এটাকে বলে চাইনিজ পোস্টম্যান প্রবলেম। এটা একটু অ্যাডভান্সড লেভেলের প্রবলেম যেটা সলভ করতে ওয়েটেড বাইপারটাইট ম্যাচিং বা মিন-কস্ট-ম্যাক্স-ফ্লো জানা লাগে। তোমার আগ্রহ থাকলে এই নিয়ে ঘাটাঘাটি করতে পারো।

আজ এ পর্যন্তই, হ্যাপি কোডিং!

প্র্যাকটিস প্রবলেম:
http://www.lightoj.com/volume_showproblem.php?problem=1256

রেফারেন্স:
http://www.algorithmist.com/index.php/Euler_tour

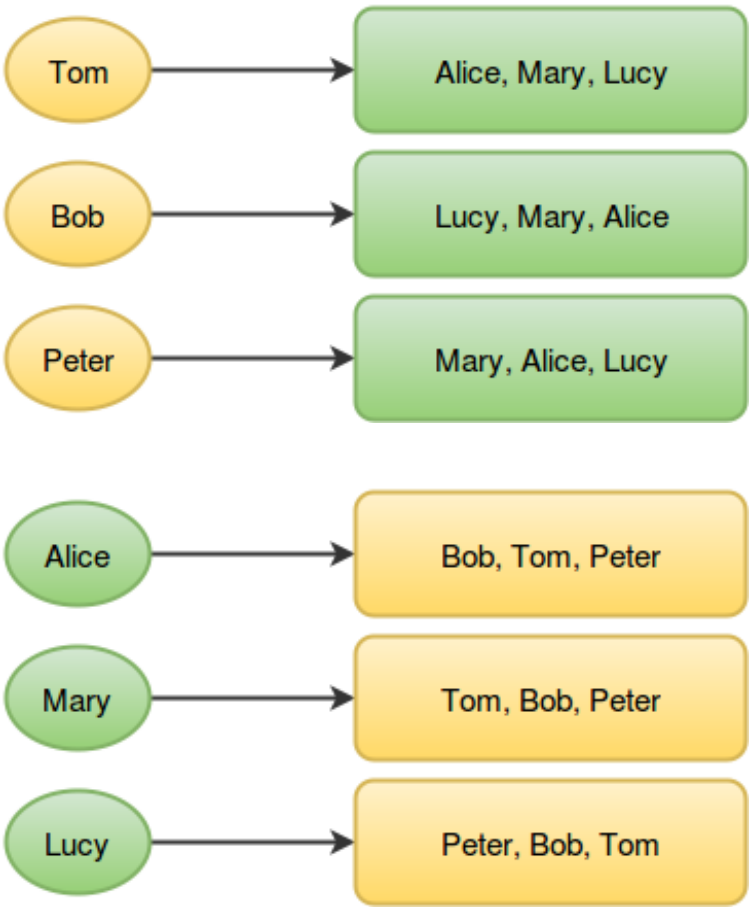
গ্রাফ থিওরি: স্টেবল ম্যারেজ প্রবলেম

shafaetsplanet.com/

শাফায়েত আগস্ট ১৫, ২০১২

বেশ কিছুদিন ডিপি নিয়ে লেখার পর আবার গ্রাফ থিওরিতে ফিরে এলাম। আজকে আমরা একটা সহজ কিন্তু ইন্টারেস্টিং প্রবলেম দেখবো। স্টেবল ম্যারিজ(Stable Marriage) প্রবলেম এক ধরনের বাইপারটাইট ম্যাচিং প্রবলেম,তবে এটা শেখার জন্য অন্য কোনো অ্যালগোরিদম জানার প্রয়োজন নেই।

মনে করি n টা ছেলে আর n টা মেয়ে আছে। এখন তাদের মধ্যে বিয়ে দিতে হবে এমন ভাবে যেনো বিয়ে "স্টেবল" হয়। প্রত্যেকের সাথেই প্রত্যেকের বিয়ে দেয়া সম্ভব তবে প্রতিটা ছেলে আর মেয়ের কিছু পছন্দ আছে,প্রত্যেকেই চাইবে তার পছন্দের মানুষকে বিয়ে করতে। যদি ছেলে ৩জনের নাম Tom,Bob,Peter , আর মেয়ে ৩জনের নাম Alice,Mary,Lucy হয় তাহলে ছেলেদের পছন্দের তালিকা হতে পারে এরকম:



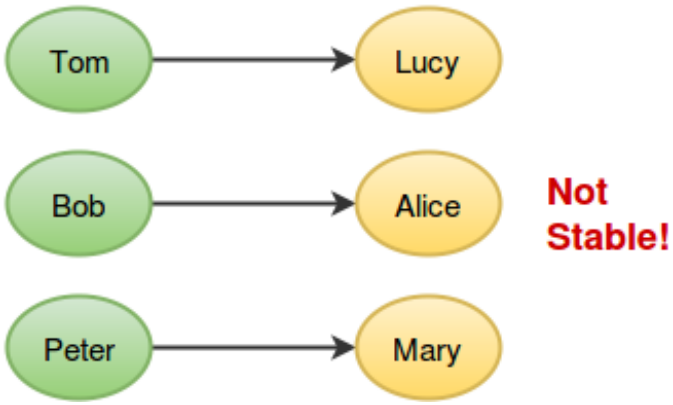
তালিকাটা বেশি থেকে কম পছন্দের ক্রমে করা হয়েছে। যেমন টম এলিসকে বেশি পছন্দ করে, লুসিকে কম পছন্দ করে।

আবার মেয়েদের পছন্দের তালিকাটা হতে পারে এরকম:

এখন কিভাবে বিয়ে দিলে বিয়ে স্টেবল হবে? আগে বুঝা দরকার স্টেবল বলতে কি বুঝাচ্ছি। ধরো নিচের মতো করে বিয়ে দেয়া হলো:

এই ম্যাচিং/বিয়েটা স্টেবল না, কারণ টম লুসির থেকে মেরিকে বেশি পছন্দ করে,আবার মেরি পিটারের থেকে টমকে বেশি পছন্দ করে। তাই টম আর মেরি বিয়ে ভেঙে একসাথে চলে আসতে পারে। যদি A,B ছেলে আর C,D মেয়ে হয় আর A-C এবং B-D কে বিয়ে দেয়া হয় তাহলে **বিয়ে স্টেবল হবেনা যদি নিচের দুটি স্টেটমেন্টই সত্য হয়:**

> ১. A যদি C এর থেকে D কে বেশি পছন্দ করে।
> ২. D যদি B এর থেকে A কে বেশি পছন্দ করে।



২টি স্টেটমেন্ট সত্য হলে A আর D বিয়ে ভেঙে চলে আসবে! তবে **যেকোনো একটা স্টেটমেন্ট মিথ্যা হলে বিয়ে স্টেবল হবে।**

১৯৬২ সালে David Gale আর Lloyd Shapley প্রমাণ করেন, সমান সংখ্যক ছেলে আর মেয়ের জন্য সমসময় স্টেবল ম্যারেজ প্রবলেমের একটি সমাধান আছে। তারা খুব সহজ একটা অ্যালগোরিদম আবিষ্কার করেন সমস্যাটি সমাধানে জন্য। অ্যালগোরিদমটি এরকম:

> ১. প্রথমে প্রতিটি অবিবাহিত ছেলে তার সবথেকে পছন্দের মেয়েটাকে প্রস্তাব পাঠাবে যাকে সে এখনো প্রস্তাব পাঠায়নি, মেয়েটি অলরেডি এনগেজড হলেও সমস্যা নাই,একটি মেয়েকে একাধিক ছেলে প্রস্তাব পাঠাতে পারে। একটি ছেলে কখনো একটি মেয়েকে দুইবার প্রস্তাব পাঠাবেনা।
>
> ২. এবার প্রতিটা মেয়ে তাকে যারা প্রস্তাব পাঠিয়েছে তাদের মধ্যে থেকে যাকে সবথেকে পছন্দ তাকে নির্বাচিত করবে,বাকি সবাইকে বাতিল করে দিবে। মেয়েটি আগেই কাওকে পছন্দ করে থাকলে তাকেও বাতিল করে দিবে।
>
> ৩. এখনো কেও অবিবাহিত থাকলে ১ম ধাপের পুনরাবৃত্তি হবে।

অ্যালগোরিদমটি কেনো কাজ করে? ধরি A-C এবং B-D এর বিয়ে দেয়া হয়ছে। তাহলে বিয়ে ভাঙবে A যদি D কে বেশি পছন্দ করে এবং D যদি A কে বেশি পছন্দ করে। কিন্তু উপরের অ্যালগোরিমে সেটা সম্ভবনা। কারণ:

> A যদি D কে বেশি পছন্দ করে তাহলে সে D কে আগে প্রস্তাব পাঠাবে,D রাজি না হলে বা ছেড়ে দিলেই একমাত্র C কে প্রস্তাব পাঠাবে।
>
> D যদি A কে বেশি পছন্দ করে তাহলে সে অন্য যে কাওকে ছেড়ে দিয়ে A কে বিয়ে করবে। আর D যদি A কে বিয়ে না করে অন্য কাওকে করে তারমানে সে অন্য কাওকেই বেশি পছন্দ করে,এক্ষেত্রে বিয়ে ভাঙার সম্ভাবনা নেই।

এই অ্যালগোরিমটা স্টেবল ম্যাচিং দিবে ঠিকই তবে অপটিমাল রেজাল্ট নাও দিতে পারে। প্রতিটি ছেলের জন্য রেজাল্ট অপটিমাল হবে,কিন্তু মেয়েদের জন্য অপটিমাল নাও হতে পারে,অর্থা এমন স্টেবল ম্যাচিং থাকতে পারে যেটাও কোনো একটি মেয়ে আরো পছন্দের কাওকে বিয়ে

করতে পারতো। অর্থা যে প্রস্তাব পাঠাবে তার জন্য রেজাল্ট অপটিমাল হবে।

অ্যালগোরিদমটি কোডে ইমপ্লিমেন্ট করা খুব সহজ। preference লিস্ট তোমাকে ইনপুট দেয়া থাকবে। কে কাকে প্রস্তাব পাঠিয়েছে,কে কার সাথে এখন এনগেজড এই তথ্যগুলো অ্যারেতে রেখে সহজেই কোডটা লিখে ফেলতে পারবে। wikipedia তে দেয়া সুডোকোডটা এরকম:

Python

```
1   function stableMatching {
2   Initialize all m ∈ M and w ∈ W to free
3   while ∃ free man m who still has a woman w to propose to {
4   w = m's highest ranked such woman to whom he has not yet proposed
5   if w is free
6   (m, w) become engaged
7   else some pair (m', w) already exists
8   if w prefers m to m'
9   (m, w) become engaged
10  m' becomes free
11  else
12  (m', w) remain engaged
13  }
14  }
```

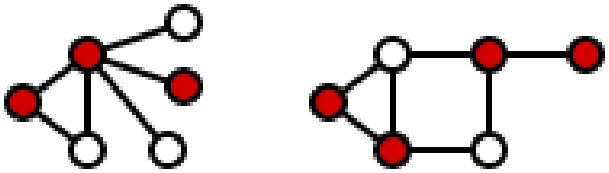
প্রবলেম:

Light OJ: Employment
Codechef: Stable Marriage
Uva: Chemical Attraction
Codechef: Blocking

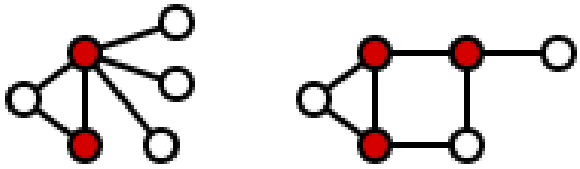
গ্রাফ থিওরি নিয়ে অন্যান্য লেখা

মিনিমাম ভারটেক্স কভার প্রবলেম

শাফায়েত অক্টোবর ৪, ২০১২



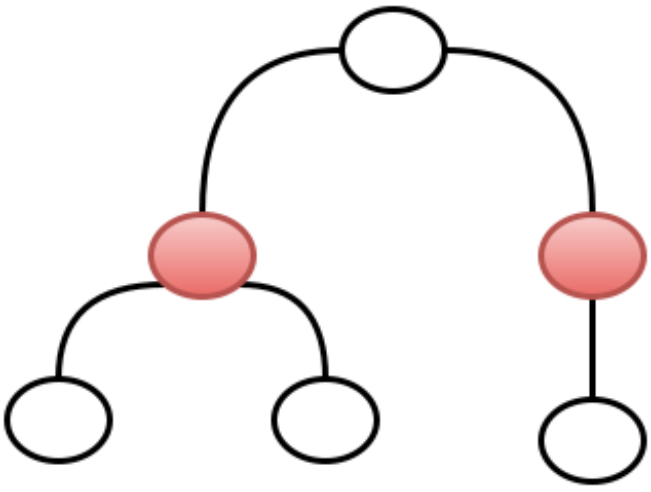
মিনিমাম ভারটেক্স কভার একটি ক্লাসিক গ্রাফ প্রবলেম। ধরা যাক একটি শহরে কিছু রাস্তা আছে,এখন প্রতি রাস্তায় মোড়ে আমরা পাহারাদার বসাতে চাই। কোনো নোডে পাহারাদার বসালে সে নোডের সাথে যুক্ত রাস্তাগুলো একাই পাহারা দিতে পারে। উপরের ছবিতে নোডগুলো হলো রাস্তার মোড়। এখন সব কয়টা রাস্তা পাহারা দিতে নূন্যতম কয়জন পাহারাদার দরকার? ছবিতে লাল নোডগুলোতো পাহারাদার বসানো হয়েছে। এটা অপটিমাল না,নিচের ছবির মত বসালে পাহারাদার কম লাগত:



এটি একটি NP-hard প্রবলেম,অর্থা এই প্রবলেমের কোনো পলিনমিয়াল টাইম সলিউশন নেই। তবে গ্রাফটি যদি **Tree** হয় অর্থাত n-1 টা edge থাকে আর কোনো সাইকেল না থাকে তাহলে ডাইনামিক প্রোগ্রামিং বা ম্যাক্স ফ্লো/বাইপারটাইট ম্যাচিং এর সাহায্যে প্রবলেমটি সলভ করা সম্ভব।

ডাইনামিক প্রোগ্রামিং সলিউশনটা আমি বিস্তারিত লিখছি,তারপর ম্যাক্স ফ্লো/বাইপারটাইট ম্যাচিং দিয়ে কিভাবে করতে হয় লিখবো।

ডিপি সলিউশনে ২টি কেস আমাদের লক্ষ্য করতে হবে:



Minimum Vertex Cover in a Tree

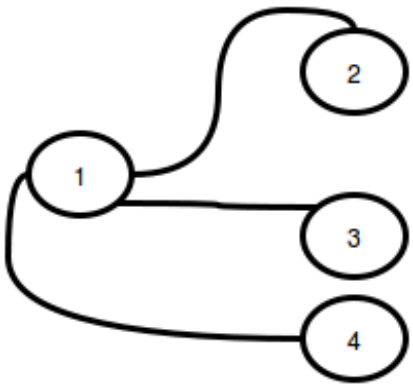
> ১. কোনো নোডে পাহারাদার না বসালে তার সাথে সংযুক্ত সব নোডে অবশ্যই পাহারাদার বসাতে হবে,এছাড়া সব রাস্তা কভার হবে না। অর্থা যদি u আর v সংযুক্ত থাকে তাহলে u তে পাহারাদার না বসালে v তে অবশ্যই বসাতে হবে।
> ২. কোনো নোডে পাহারাদার বসালে সংযুক্ত নোডগুলোতে পাহাদার বাসানো বাধ্যতামূলক না তবে বসালে লাভ হতে পারে। তাই u তে পাহারাদার বসালে v তে পাহারাদার একবার বসিয়ে এবং একবার না বসিয়ে দেখবো কোনটা লাভজনক

সব ডিপির প্রবলেমের মতো এখানেও একটা রিকার্সিভ ফাংশন ডিফাইন করবো। আমাদের স্টেট হবে বর্তমানে কোন নোডে আছি,এবং সেই নোডে কোনো পাহারাদার বসানো হয়েছে নাকি।

> F(u,1) = বর্তমানে u নম্বর নোডে আছি এবং এই নোডে **পাহারাদার আছে**। f(u,1) রিটার্ণ করবে বাকি নোডগুলোতে মোট পাহারাদার সংখ্যা।
> F(u,0) = বর্তমানে u নম্বর নোডে আছি এবং এই নোডে **পাহারাদার নাই**। f(u,0) রিটার্ণ করবে বাকি নোডগুলোতে মোট পাহারাদার সংখ্যা।

ধরি ১ নম্বর নোডের সাথে ২,৩,৪ নম্বর নোড যুক্ত।

বুঝাই যাচ্ছে ১ নম্বর নোডে পাহারা না বসালে অবশ্যই ২,৩,৪ সবগুলোয় পাহারা বসাতে হবে। তাহলে আমরা বলতে পারি:



> F(1,0)=F(2,1)+F(3,1)+F(4,1) + 0 ,অর্থাত ১ এর সাথে সংযুক্ত সব নোডগুলোতে পাহারা বসালে প্রয়োজনীয় মোট পাহারাদার সংখ্যা।

সবশেষে ০ যোগ করছি কারণ বর্তমান নোডে পাহারাদার বসাইনি।

এবার F(1,1) এর মান বের করি। ১ নম্বর নোডে পাহারা বসালে সংযুক্ত নোডগুলোতে পাহারা বসালেও চলে,না বসালেও চলে,তবে যেটা অপটিমাল রেজাল্ট দেয় সেটা আমরা নিব:

> F(1,1)=1+min (F(2,1),F(2,0)) +min (F(3,1),F(3,0)) +min (F(4,1),F(4,0))

১ নম্বর নোডে পাহারাদার বসাচ্ছি তাই সবশেষে ১ যোগ হচ্ছে,প্রতি নোডে একবার পাহারা বসিয়ে,আবার না বসিয়ে দেখছি কোনটা অপটিমাল।

একটা ব্যাপার লক্ষ রাখতে হবে যে প্যারেন্ট নোড নিয়ে কখনো হিসাব করবোনা। উপরের ছবিতে ১ থেকে ২ এ গেলে parent[2]=1,তাই ২ থেকে আবার ১ নম্বর নোডে যাবোনা।

এবার base case এ আসি। কোনো নোড থেকে নতুন কোনো নোডে যাওয়া না গেলে 1 বা 0 রিটার্ন করে দিতে হবে,পাহারাদার বসালে 1,না বসালে 0। কোনো ট্রি তে একটি মাত্র নোড থাকলে ১ রিটার্ণ করতে হবে(কিছু প্রবলেমে ০ ও রিটার্ণ করতে হতে পারে)।

Spoj এর PT07X(vertex cover) প্রবলেমটি straight forward প্রবলেম। এটার জন্য আমার কোডটা এরকম:

```
1   #define MAXN 100002
2   int dp[MAXN][5];
3   int par[MAXN];
4   vectoredges[MAXN];
5   int f(int u, int isGuard)
6   {
7       if (edges[u].size() == 0)
8           return 0;
9       if (dp[u][isGuard] != -1)
10          return dp[u][isGuard];
11      int sum = 0;
12      for (int i = 0; i < (int)edges[u].size(); i++) {
13          int v = edges[u][i];
14          if (v != par[u]) {
15              par[v] = u;
16              if (isGuard == 0)
17                  sum += f(v, 1);
18              else
19                  sum += min(f(v, 1), f(v, 0));
20          }
21      }
22      return dp[u][isGuard] = sum + isGuard;
23  }
24  int main()
25  {
26      memset(dp, -1, sizeof(dp));
27      int n;
28      scanf("%d", &n);
29      for (int i = 1; i < n; i++) {
30          int u, v;
31          scanf("%d%d", &u, &v);
32          edges[u].push_back(v);
33          edges[v].push_back(u);
34      }
35      int ans = 0;
36      ans = min(f(1, 1), f(1, 0));
37      printf("%d\n";, ans);
38      return 0;
39  }
40
41
```

আমি ট্রি এর root সবসময় ১ ধরে কোড লিখেছি। ৩৮ নম্বর লাইনে মেইন ফাংশনে root এ পাহারাদার একবার বসিয়ে আর একবার না বসিয়ে অপটিমাল রেজাল্ট টা নিচ্ছি।
ফাংশনে u হলো current node,isguard কারেন্ট নোডে পাহারাদার আছে নাকি নাই সেটা নির্দেশ করে।
১০ নম্বর লাইনে ট্রি এর সাইজ ১ হলে ১ রিটার্ণ করে দিয়েছি।
১৩ নম্বর লাইনে লুপের ভিতর current নোড থেকে সবগুলো child নোডে যাচ্ছি। কারেন্ট নোডে পাহারাদার না থাকলে পরেরটায় বসাচ্ছি,আর থাকলে ২ভাবেই চেষ্টা করছি। ১৫ নম্বর লাইনের কন্ডিশন দিয়ে প্যারেন্ট নোডে যেতে দিচ্ছিনা।
সবশেষে sum+isGuard রিটার্ণ করছি। অর্থাত কারেন্ট নোডে পাহারাদার থাকলে 1 যোগ করছি,নাহল 0।

মোটামুটি এই হলো ডিপি সলিউশন। ট্রি তে সাইকেল না থাকায় এটা অবশ্যই বাইপারটাইট গ্রাফ। ১৯৩১ সালে Dénes Kőnig প্রমাণ করেন কোনো বাইপারটাইট গ্রাফে maximum matching=minimum vertext cover। এটা গ্রাফ থিওরির অনেক min-max থিওরেমের একটা যেখানে কিছু একটা ম্যাক্সিমাইজ করলে অন্য আরেকটা কিছু মিনিমাইজ হয়। তুমি যদি ম্যাক্সিমাম ম্যাচিং এর অ্যালগোরিদম জানো তাহলে ট্রি টা বাইকালারিং করে ম্যাচিং বের করলেই ভারটেক্স কভার বের হয়ে যাবে। কোড সহজ হলেও complexity বেড়ে যাবে,তাই নোড বেশি থাকলে কাজ করবেনা। আবার ম্যাক্সিমাম ম্যাচিং যেহেতু ম্যাক্স-ফ্লো এর একটি ভ্যারিয়েশন তাই ফ্লো চালিয়েও সমাধান করা সম্ভব।

এরকম আরেকটা প্রবলেম uva-10243(fire fire fire)। আমি প্রথমে এটা সমাধান করে পরে spoj এর টা করেছি।

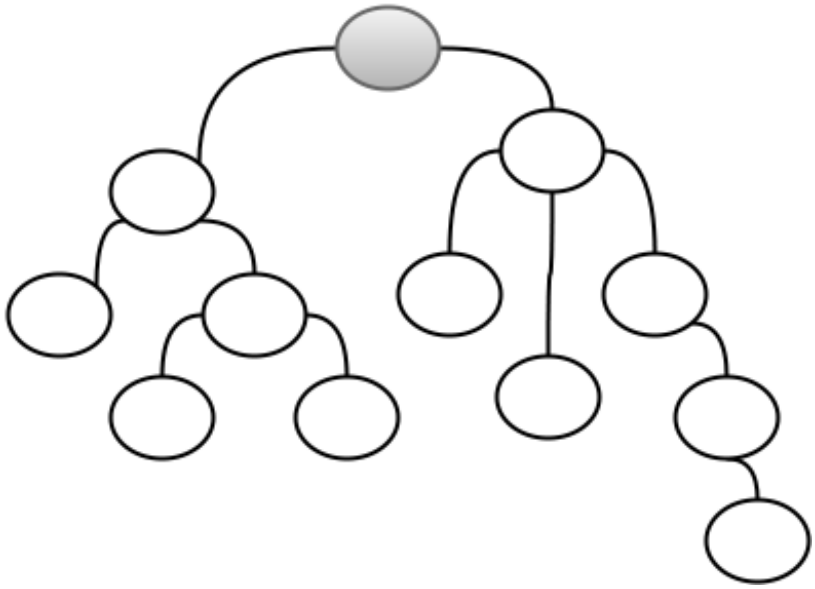
ট্রি ডায়ামিটার

shafaetsplanet.com/

শাফায়েত জানুয়ারি ৮, ২০১৪

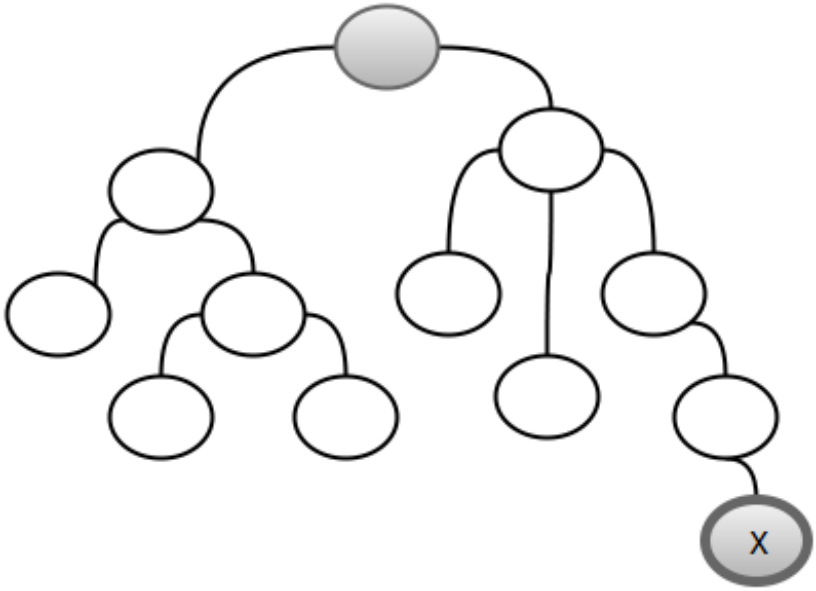
ট্রি হলো এমন একটা আনডিরেক্টেড গ্রাফ যেটার সব নোড থেকে সব নোডে যাওয়া যায় এবং কোনো সাইকেল নেই। এখন আমাদের ট্রি এর সবথেকে দূরের দুটা নোড খুজে বের করতে হবে, একেই বলা হয় ট্রি এর ডায়ামিটার।

মনে করো কিছু কম্পিউটারের মধ্যে নেটওয়ার্ক কেবল লাগানো হয়েছে নিচের ছবির মতো করে। এখন তুমি জানতে চাইতেই পারো কোন দুটি কম্পিউটার সবথেকে দূরে আছে।

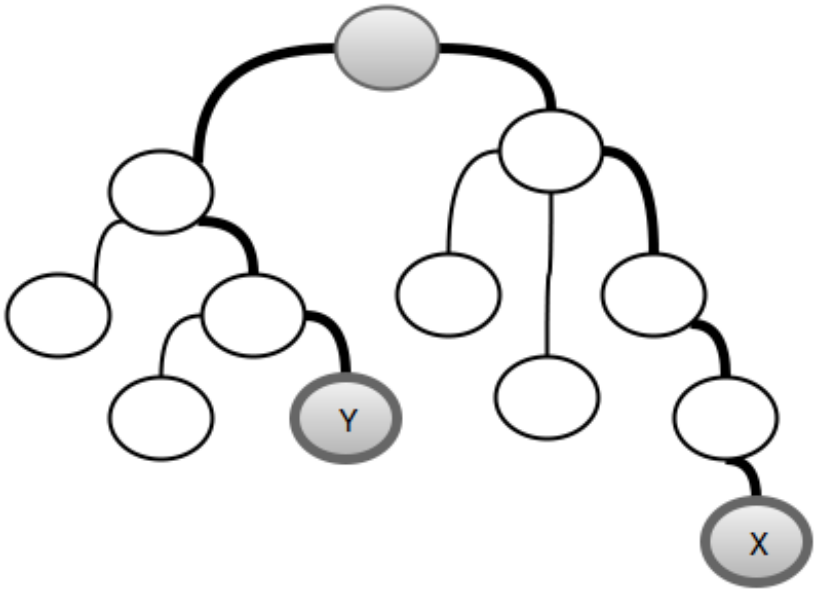


এটা বের করা খুব সহজ, এজন্য তোমার জানতে হবে বিএফএস বা ডিএফএস এর যে কোন একটা। আনডিরেক্টেড ট্রি তে **যেকোন নোড**কেই রুট ধরা যায়, আমরা মনে করি উপরের ধূসর নোডটা ট্রি এর রুট।

আমাদের প্রথম কাজ কাজ হলো রুট হতে সবথেকে দূরের নোডটা খুজে বের করা। সেই নোডটাকে মনে করি X । একাধিক নোডের দূরত্ব সবথেকে দূরের নোডের দূরত্বের সমান হলেও সমস্যা নেই, যেকোন একটাকে সিলেক্ট করতে হবে। এই কাজটা আমরা ডিএফএস/বিএফএস চালিয়ে বের করতে পারি।



এখন ২য় কাজ হলো X নোড থেকে শুরু আরেকটি ডিএফএস/বিএফএস চালিয়ে X থেকে সবথেকে দূরের নোড খুজে বের করা। মনে করি নোডটা হলো Y।



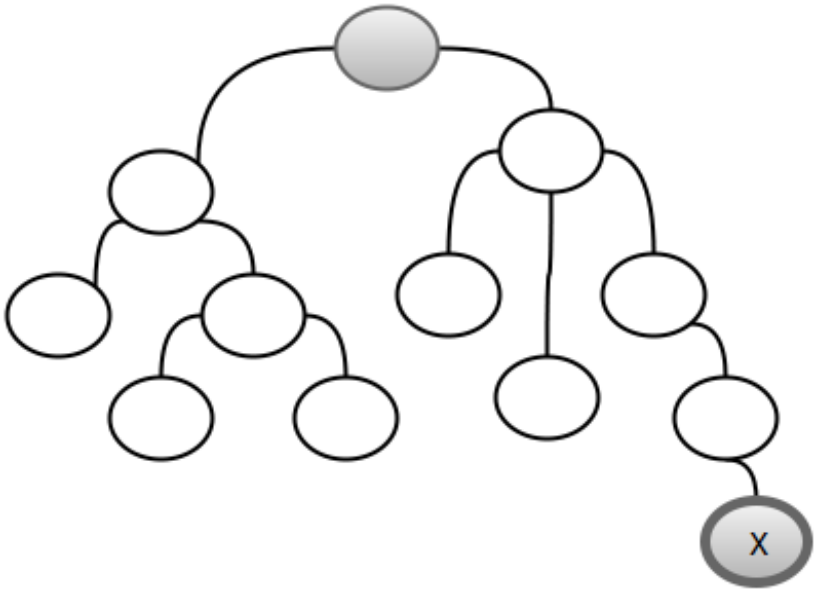
X আর Y এর মধ্যকার দূরত্বই ট্রি এর ডায়ামিটার! উপরের ছবিতে ডায়ামিটার ৭।

প্রমাণ:

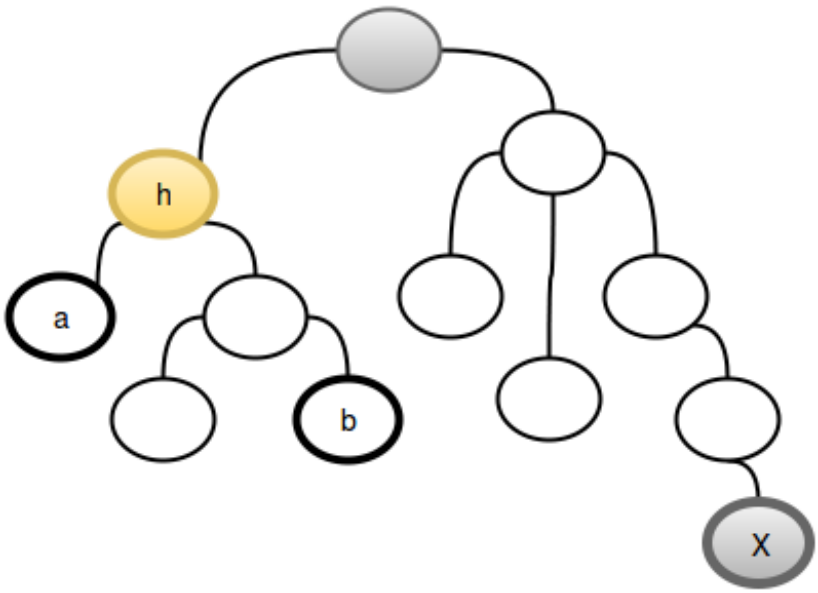
আমরা যদি প্রমাণ করতে পারি ১ম ধাপে খুজে পাওয়া X সবসময়ই ডায়ামিটারের একটা প্রান্ত হবে তাহলেই অ্যালগোরিদমটা প্রমাণ হয়ে যায়। কারণ X যদি নিশ্চিতভাবে ডায়ামিটারের একটা প্রান্ত হয় তাহলে ২য় ধাপটা অবশ্যই সঠিক, X থেকে সবথেকে দূরের নোডই হবে ট্রি এর ডায়ামিটার।

এটা আমরা প্রমাণ করবো “প্রুফ বাই কনট্রাডিকশন” এর সাহায্যে। এটা বহুল ব্যবহৃত একটা পদ্ধতি। আমরা প্রমাণ করতে চাই X নিশ্চয়ই কোনো ডায়ামিটারের প্রান্ত। কিন্তু তা যদি না হয়, অর্থা X কোনো ডায়ামিটারের প্রান্ত না হলে এমন একটা ডায়ামিটার থাকবে যার দুই প্রান্ত (ধরি) a

এবং b, এখানে a এবং b যেকোনো দুটি নোড হতে পারে(X ছাড়া) । a-b ডায়ামিটার ব্যবহার করে আমরা দেখাবো এমন আরেকটা ডায়ামিটার পাওয়া সম্ভব যেটা a-b এর চেয়ে বড় বা সমান এবং যার এক প্রান্তে X আছে। (উল্লেখ্য, X হলো রুট থেকে সবচেয়ে দূরের একটা নোড)।



এখন **আর্বিট্রালি(arbitrarily)** দুটি নোড a আর b সিলেক্ট করি। নোড দুটির মধ্যকার পাথ রুটের কাছে যে নোডে এসে মিলবে সেটার নাম দেই h, অর্থা h হলো নোড দুইটার কমন অ্যানসেস্টর। (অনেকেই হয়তো বুঝে গেছো আমরা এখানে lowest common ancestor এর কথা বলছি)।



কিন্তু রুট থেকে b এর দূরত্ব, রুট থেকে X এর দূরত্বের কম বা সমান হতে বাধ্য, বেশি হবেনা কারণ রুট থেকে সবথেকে দূরের নোড হলো X।

অর্থা:

> distance(root,b)<=distance(root,X)

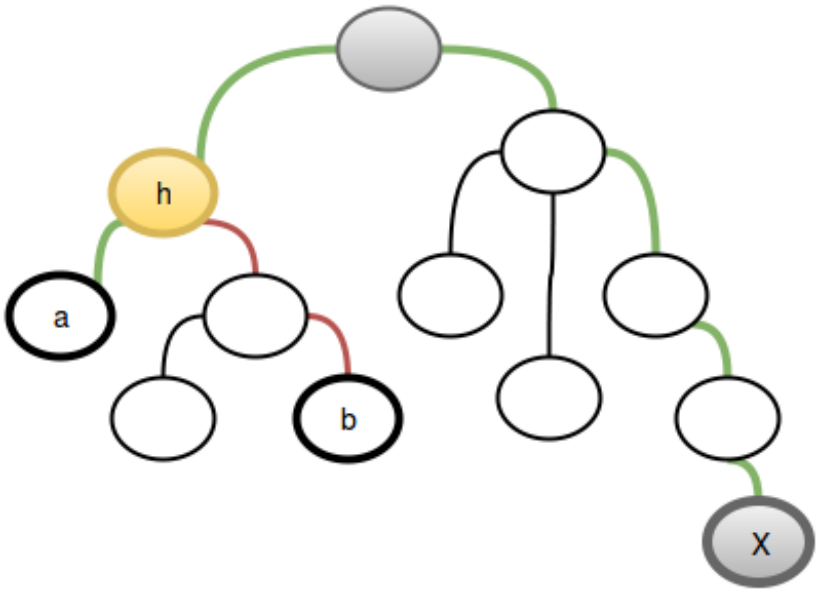
আবার এটাও নিশ্চিত যে রুট থেকে b যত দূরে, h থেকে b এর দূরত্ব তার থেকে কম বা সমান। (সমান হবে যদি h আর রুট একই নোড হয় অর্থা a আর b যদি রুটের দুইপাশে থাকে)

> distance(h,b)<=distance(root,b)

তাহলে,

> distance(h,b)<=distance(root,b)<=distance(root,X)
>
> distance(h,b)<=distance(root,X)

তাহলে আমরা h-b পাথটা root-X পাথ দিয়ে রিপ্লেস করে দিলে এবং h-x এর মধ্যে পাথ বসিয়ে দিলে(যদি h আর x সমান না হয়) অবশ্যই আগের সমান বা আগের থেকে বড় একটা পাথ পাবো!



তারমানে আমরা যে X কে বাদ দিয়ে যেই দুইটা নোডই সিলেক্ট করিনা কেন, X কে নিয়ে তার থেকে বড় বা সমান একটা পাথ পাওয়া যাবে। তারমানে X অবশ্যই ডায়ামিটারের একটা প্রান্ত!

কমপ্লেক্সিটি:

বিএফএস/ডিএফএস এর কমপ্লেক্সিটির সমান: O(V+E) যেখানে V হলো নোডসংখ্যা এবং E হলো এজ সংখ্যা।

> এই অ্যালগোরিদমটা জেনারেল গ্রাফে কাজ করবেনা, শুধু ট্রি এর ক্ষেত্রে করবে। জেনারেল গ্রাফে লংগেস্ট পাথ বের করার প্রবলেম এন-পি হার্ড, আর ডায়ামিটার বের করার প্রবলেম লংগেস্ট পাথ প্রবলেমেরই স্পেশাল কেস।

Farthest Nodes in a Tree

Farthest Nodes in a Tree (II)

হ্যাপি কোডিং!

কৃতজ্ঞতা স্বীকার:

http://stackoverflow.com/questions/20010472/proof-of-correctness-algorithm-for-diameter-of-a-tree-in-graph-theory

http://apps.topcoder.com/forums/?module=Thread&threadID=668470&start=0&mc=12#1216875

লংগেস্ট পাথ প্রবলেম

shafaetsplanet.com/

শাফায়েত

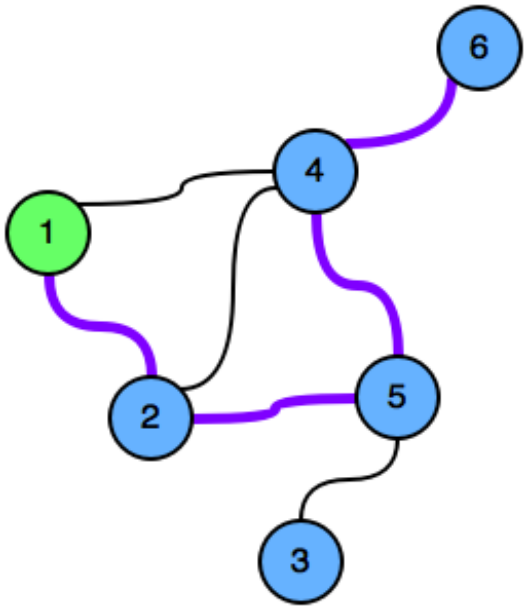
July 21, 2016

তোমাকে একটা আনওয়েটেড গ্রাফ এবং একটা সোর্স নোড দেয়া আছে। তোমাকে সোর্স নোড থেকে সর্বোচ্চ দৈর্ঘ্যের পাথ বের করতে হবে। এটাই হলো লংগেস্ট পাথ প্রবলেম। প্রশ্ন হলো কিভাবে প্রবলেমটা সলভ করবে?

এটা আমার খুবই প্রিয় একটা ইন্টারভিউ প্রশ্ন। এখন পর্যন্ত প্রায় ৭-৮টা ইন্টারভিউতে আমি এই প্রশ্ন করেছি, মাত্র ২জন সহজেই উত্তর দিতে পেরেছে, ১জন হিন্টস দেয়ার পর পেরেছে, বাকিরা সবাই ভুল উত্তর দিয়েছে। অ্যালগোরিদম কোর্সে এ+ অনেকেই পায়, কিন্তু এধরণের প্রশ্ন করলে বোঝা যায় ক্যান্ডিডেট আসলে কতখানি জানে।

প্রশ্নটা করার পর সবাই প্রথমে যে ভুল করে সেটা হলো জিজ্ঞেস করে না সর্বোচ্চ দৈর্ঘ্যের পাথের সংজ্ঞা কি, আমি চাইলে দুটি নোডের মধ্যে অসীম সংখ্যকবার আসা-যাওয়া করে দৈর্ঘ্য অসীম করে ফেলতে পারি। তাই লংগেস্ট পাথের আরো ভালো কোনো সংজ্ঞা দরকার সমাধান করা জন্য। ধরা যাক আমি এমন একটা লংগেস্ট পাথ চাই যেখানে কোনো নোডে একবারের বেশি যাওয়া যাবে না। এখন আর দৈর্ঘ্য অসীম হবার কোনো সুযোগ নেই।

ছবিতে ১ নম্বর নোড থেকে সর্বোচ্চ দৈর্ঘ্যের পাথ দেখানো হয়েছে যার দৈর্ঘ্য হলো ৪ (১->২->৫->৪->৬)।

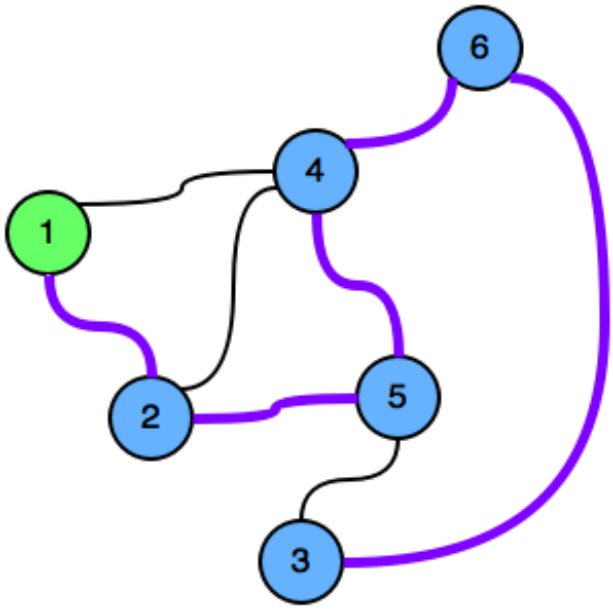


সমাধান নিয়ে চিন্তা করার আগে পরের প্রশ্ন হলো গ্রাফটা কি ডিরেক্টেড নাকি আনডিরেক্টেড। ধরে নাও গ্রাফটা আনডিরেক্টেড এবং সর্বোচ্চ ১০০,০০০ টা নোড থাকতে পারে।

এই প্রবলেমের অনেক ধরণের ভুল সমাধান দিতে দেখেছি ক্যান্ডিডেটদের, যেমন:

- কেও কেও গ্রাফের এজগুলোর ওয়েট -১ ধরে শর্টেস্ট পাথ বের করার চেষ্টা করে। এটা কোনোভাবেই কাজ করবে না, ইনফিনিটি লুপে আটকে যাবে।
- কেও কেও ডায়নামিক প্রোগ্রামিং ব্যবহার করে সমাধান করতে চেষ্টা করে, এটাও কাজ করবে না এবং লুপে আটকে যাবে।
- ম্যাক্স ফ্লো ব্যবহার করতে চায় অনেকে কিন্তু সেটাও কাজ করবে না।

তাহলে সমাধান টা কি? মনে করি গ্রাফটার নোড সংখ্যা n। যেহেতু n এর মান অনেক বড় হতে পারে, তাই অবশ্য পলিনোমিয়াল সলিউশন দরকার। এখন এমন একটা গ্রাফের কথা চিন্তা করো যেটায় সোর্স থেকে লংগেস্ট পাথের দৈর্ঘ্য হলো $n-1$। যেমন নিচের গ্রাফটা:



এই গ্রাফটিতে লংগেস্ট পাথের দৈর্ঘ্য হলো $n-1$ যেখানে $n=6$। তারমানে হলো গ্রাফটাতে এমন একটা সোর্স নোড আছে যেখান থেকে যাত্রা শুরু করে সবগুলো নোড একবার করে ভ্রমণ করা যায় কোনো নোড পুনরাবৃত্তি না করেই। এ ধরণের পাথকে বলা হয় হ্যামিল্টন পাথ! তুমি যদি গ্রাফথিওরি নিয়ে কিছুটা পড়ালেখা করে থাকো তাহলে অবশ্য জানার কথা যে হ্যামিল্টন পাথ প্রবলেম একটা এন-পি কমপ্লিট (NP-complete) প্রবলেম। এন-পি কমপ্লিট কোনো প্রবলেমকে পলিনমিয়াল সময়ে সমাধান করার কোনো উপায় এখন পর্যন্ত কারো জানা নেই।

দেখতেই পাচ্ছো হ্যামিল্টন পাথ প্রবলেমকে খুব সহজেই লংগেস্ট পাথ প্রবলেমে কনভার্ট করা যায়। একটা গ্রাফে কোনো সোর্স থেকে এমন একটা লংগেস্ট পাথ বের করতে হবে যেন পাথের দৈর্ঘ্য হয় $n-1$, এটাই হ্যামিল্টন পাথ প্রবলেম। লংগেস্ট পাথ প্রবলেমকে পলিনমিয়াল সময়ে সমাধান করা গেলে হ্যামিল্টন পাথ প্রবলেমও সমাধান হয়ে যেত!

তাহলে দেখতেই পাচ্ছি যে লংগেস্ট পাথ প্রবলেমের কোনো পলিনমিয়াল সলিউশন নেই। নোড সংখ্যা যদি খুব অল্প হয় তাহলে ব্যাকট্র্যাকিং করে এই সমস্যা সমাধান করা সম্ভব, কিন্তু এটার কমপ্লেক্সিটি এক্সপোনেনশিয়াল(exponential)। লংগেস্ট পাথ হলো একটা এন-পি হার্ড প্রবলেম।

এখন প্রশ্ন হলো কেন হ্যামিল্টন পাথ প্রবলেম এন-পি কমপ্লিট কিন্তু লংগেস্ট পাথ এন-পি হার্ড? যদি এটার উত্তর না জেনে থাকো তাহলে আমার এই লেখাটা পড়ে ফেলো।

হ্যাপি কোডিং