লিংকগুলো
টিউটোরিয়াল
প্রোগ্রামিং রিসোর্সেস

শর্টেস্ট পাথের অ্যালগরিদম

আমরা যখন প্রবলেম সলভ করি কম্পিউটার ব্যবহার করে, বেশিরভাগ প্রবলেম হয় অপটিমাইজেশন প্রবলেম। (অপটিমাইজেশন মানে হচ্ছে যদি অনেকগুলো পথ থাকে কিছু একটা করার, আমরা সবচে' সেরা পথটাতে সেটা করবো) ধরা যাক কেউ ঢাকা থেকে সান ফ্রান্সিসকো যাবে, সে করবে কি একটা ট্র্যাভেল এজেন্ট এর কাছে যাবে সবচে' কম খরচের রুটটা বের করে দিতে। ট্র্যাভেল এজেন্ট করবে কি, একটা কম্পিউটার প্রোগ্রামকে জিজ্ঞেস করবে পসিবল রুটগুলো দেখানোর জন্য, আর কম খরচের রুটগুলো দেখানোর জন্য। তারপর সে সেই অনুযায়ী টিকেট কেটে দেবে। এখানে সবচে' সস্তা রুটটা হচ্ছে আমাদের "অপটিমাল" পথ।

আমরা বুঝলাম যে ঢাকা থেকে সান ফ্রান্সিসকো যাবে সে কি কি করবে। কিন্তু কম্পিউটার প্রোগ্রামটা করবে টা কি?

প্রিরিকুইজিট

- প্রোগ্রামিং - যদি না জানো এখানে যাও - শিখে ফেলো! কি আছে জীবণে?
- STL - কিউ, প্রায়োরিটি কিউ আর ভেক্টর জানা লাগবে - এখানে দেখতে পারো
- স্ট্রাকচার - এখানে উঁকি মারো

শর্টেস্ট পাথ

শর্টেস্ট পাথ এর সংজ্ঞাটা হবে অনেকটা এরকম। আমার যদি অনেকগুলো শহর থাকে, আর শহরগুলোর মধ্যে যদি অনেকগুলো পথ থাকে, (যেই পথগুলোর একেকটার একেকরকমের দৈর্ঘ্য থাকতে পারে - এবং কোন পথে কোন জ্যাম থাকবে না) তাহলে কোন একটা শহর থেকে অন্য আরেকটা শহরে যেই পথ দিয়ে গেলে সবচে' কম দূরত্ব যাওয়া লাগবে সেটা হচ্ছে আমার প্রথম শহরটা থেকে দ্বিতীয় শহরটার শর্টেস্ট পাথ।

যখন আমরা গ্রাফ থিওরীর টার্মিনোলজিতে কথা বলি, তখন আমরা শহরগুলোকে শহর না বলে বলি নোড(node), আর পথগুলো পথ না বলে বলি এজ(edge), আর আমরা এই পুরো ম্যাপটাকে বলে গ্রাফ(graph)। টার্মিনোলজি শেখাটা কাজের, কারণ তাহলে তুমি যেই ছেলেটা চাইনিজ ছাড়া আর কিছুতে কথা বলতে পারে না, কিন্তু গ্রাফ থিওরী জানে, তার সাথেও কিছুক্ষণ গ্রাফ থিওরী নিয়ে পটপট করতে পারবা। আমার প্রথম প্রথম ম্যাপটাকে "গ্রাফ" বলে চিন্তা করতে খুব সমস্যা হতো, কারণ আমরা যেটাকে "গ্রাফ" বলে চিন্তা স্কুল কলেজে, সেটা পুরোপুরি আলাদা জিনিস। কিন্তু একটা সময় তুমি এমনিতেই অভ্যস্ত হয়ে যাবা।

ক্যামনে করবো?

বেশ তো, কিন্তু প্রথমে চিন্তা করো তুমি কিভাবে, কম্পিউটারে ডাটা হিসেবে শহরগুলোর পথগুলোকে রাখবা। সবচে' সহজভাবে জিনিসটা এভাবে করা যায় - আমি একটা 2D অ্যারে রাখি, distance নামের যেখানে distance[i][j] হবে i তম শহর থেকে j তম শহরে যাওয়ার দূরত্ব। তো এই ম্যাট্রিক্সটাকে সাধারণত বলা হয় অ্যাডজাসেন্সি ম্যাট্রিক্স(adjacency matrix)।

```
#define M 100
int distance[M][M];
```

অ্যাডজাসেন্সি ম্যাট্রিক্স এর প্রথম সমস্যা হচ্ছে এটা জায়গা বেশি খায়। ধরো তোমার ২০ ০০০ টা শহর আছে আর তাদের মধ্য ৫০ ০০০ টা পথ আছে। আমার এখানে সবমিলে ডাটা আসলে ৫০ ০০০। কিন্তু তুমি যখন অ্যারে ডিক্লেয়ার করতে যাচ্ছো, তোমার ২০০০০ x ২০০০০ সাইজের অ্যারে ডিক্লেয়ার করতে হচ্ছে, যার বেশিরভাগই তুমি ব্যবহার করছো না। আর তোমার RAM এ হয়তো এত জায়গাও নেই। দ্বিতীয় সমস্যা হচ্ছে, তুমি ঠিক জানো না i এর সাথে কোন কোন j তে পথ আছে। তো তোমার সবগুলো j খুঁজে খুঁজে দেখতে হবে সেখান থেকে পথ আছে কি না, যেটা আরো সমস্যা। কারণ এমন হতে পারে i নাম্বার শহরটার শুধু একটা শহরের সাথে পথ আছে, আর সেটা আছে সবার শেষ ইন্ডেক্সটাতে। তো এই ক্ষেত্রে আমরা বোকার মত সময় নষ্ট করবো পথ খুঁজতে।

সহজ উপায় হচ্ছে আমরা দুটো ভেক্টরের অ্যারে রাখতে পারি। ধরো প্রথমটার নাম হচ্ছে edge, দ্বিতীয়টার নাম হচ্ছে cost।

```
#define M 100
vector< int > edge[M], cost[M];
```

edge[i] তে থাকবে সবগুলো নোড যাদের i এর সাথে পথ আছে, আর cost[i] তে থাকবে, যথাক্রমে সেই পথগুলোর দূরত্ব। মানে ধরো edge[i] এর প্রথম এলিমেন্টটা যদি 23 হয় তাহলে cost[i] এর প্রথম এলিমেন্টটা হবে i থেকে 23 এর দূরত্ব।

প্রথম অ্যালগরিদম - ব্রেডথ ফার্সট সার্চ (Breadth First Search - BFS)

তোমার যদি অল্পিকটু রিকার্শন জানা থাকে তুমি DFS চালাই দিতে পারো গ্রাফ এর উপর শর্টেস্ট পাথ বের করার জন্য। কিন্তু রিকার্শন একটা ঝামেলার হয়ে যায় আসলে, প্রচুর ফাংশন কল আর ডিপেন্ডেন্সির কারণে জিনিসটা স্লো হয়ে যায়।

শর্টেস্ট পাথ বের করার একটা সহজ আর কাজের পদ্ধতি হচ্ছে BFS। BFS লেখা বেশ সোজা, মানে কন্টেস্টের সময়ে খুব দ্রুত লিখে ফেলা যায়। আর তুমি যদি হাল্কা পাতলা অপটিমাইজ করতে পারো তাহলে BFS ঝড়ের বেগে উড়বে।

BFS এ আমরা যেটা করি তা হচ্ছে। আমরা একটা কিউ(queue) রাখি, আর একটা দূরত্ব রাখার জন্য অ্যারে রাখি। প্রথমে আমরা ধরে নেই সবার দূরত্ব অসীম, আর শুধু শুরু শহরটার দূরত্ব হচ্ছে শূণ্য। তারপর আমরা কিউতে শুরুর শহরটাকে ঢুকাই।

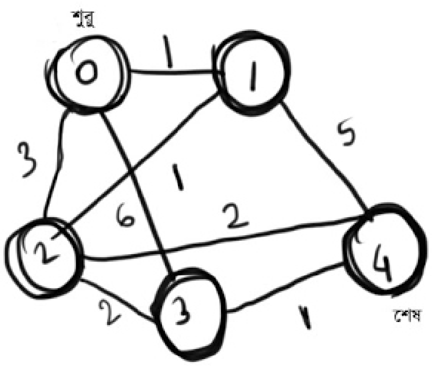
এরপর যতক্ষণ না কিউ খালি হচ্ছে ততক্ষণ আমরা করি কি কিউর প্রথমে যেই শহরটা আছে, সেটাকে বের করে আনি প্রসেসিং এর জন্য। তারপর ওটা থেকে যেই শহরগুলোতে যাবার পথ আছে সেই শহরগুলোতে যদি আমরা এই শহরটা থেকে যাই তাহলে যদি আগের চেয়ে কম সময় লাগে যেতে

আমরা

1. সেই শহরটার নতুন দূরত্ব আপডেট করি
2. নতুন দূরত্ব পাওয়া শহরটাকে প্রসেস করার জন্য কিউতে পুশ করি।

তারপর যতক্ষণ না পর্যন্ত সব শহরগুলোকে প্রসেস হচ্ছে ততক্ষণ ধরে আমরা আপডেট করতে থাকবো। যদি কোন একটা সময় আপডেট আর করা না যায় (মানে, আমরা কোন শহরের জন্যই এমন কোন নতুন পথ পাচ্ছি না যেটা দিয়ে গেলে ওই শহরটাতে বর্তমান পথটার চেয়ে তাড়াতাড়ি যাওয়া যাবে)

ধরো এই গ্রাফটার জন্য



প্রথমে আমরা 0 কে কিউতে পুশ করবো। এখন 0 এর দূরত্ব হচ্ছে 0 আর বাকি সবার দূরত্ব হচ্ছে অসীম।

শহর	দূরত্ব
0	0
1	অসীম
2	অসীম
3	অসীম
4	অসীম

0 থেকে যদি আমি 1 এ যাই, তাহলে 1 এর নতুন দূরত্ব হবে 1। 1 কে প্রসেসিং এর জন্য কিউতে ঢুকাই।
0 থেকে যদি আমি 2 এ যাই, তাহলে 2 এর নতুন দূরত্ব হবে 3। 2 কে প্রসেসিং এর জন্য কিউতে ঢুকাই।
0 থেকে যদি আমি 3 এ যাই, তাহলে 3 এর নতুন দূরত্ব হবে 6। 3 কে প্রসেসিং এর জন্য কিউতে ঢুকাই।

এখন আমাদের কিউ এর অবস্থা এই রকম 1,2,3

শহর	দূরত্ব
0	0
1	1
2	3
3	6
4	অসীম

আমরা কিউ এর ডগা থেকে 1 কে বের করে আনবো প্রসেসিং এর জন্য। 1 এর দূরত্ব হচ্ছে 1।

1 থেকে যদি আমি 0 এ যাই, তাহলে 0 এর নতুন দূরত্ব হবে 1+1 = 2। কোন দরকার নাই আপডেটের, 0 এর দূরত্ব 0।
1 থেকে যদি আমি 2 এ যাই, তাহলে 2 এর নতুন দূরত্ব হবে 1+1 = 2। 2 কে প্রসেসিং এর জন্য আবার কিউতে ঢুকাই।
1 থেকে যদি আমি 4 এ যাই, তাহলে 4 এর নতুন দূরত্ব হবে 1+5 = 6। 4 কে প্রসেসিং এর জন্য আবার কিউতে ঢুকাই।

এখন আমাদের কিউ এর অবস্থা এই রকম 2,3,2,4

শহর	দূরত্ব
0	0
1	1
2	2
3	6
4	6

আমরা কিউ এর ডগা থেকে 2 কে বের করে আনবো প্রসেসিং এর জন্য। 2 এর দূরত্ব হচ্ছে 2।

2 থেকে যদি আমি 0 এ যাই, তাহলে 0 এর নতুন দূরত্ব হবে 2+3= 5 । কোন দরকার নাই আপডেটের, 0 এর দূরত্ব 0।
2 থেকে যদি আমি 1 এ যাই, তাহলে 1 এর নতুন দূরত্ব হবে 2+1= 3 । কোন দরকার নাই আপডেটের, 1 এর দূরত্ব 1।
2 থেকে যদি আমি 3 এ যাই, তাহলে 3 এর নতুন দূরত্ব হবে 2+2= 4 । 3 কে প্রসেসিং এর জন্য আবার কিউতে ঢুকাই।
2 থেকে যদি আমি 4 এ যাই, তাহলে 4 এর নতুন দূরত্ব হবে 2+2 = 4। 4 কে প্রসেসিং এর জন্য আবার কিউতে ঢুকাই।

এখন আমাদের কিউ এর অবস্থা এই রকম 3,2,4,3,4

শহর	দূরত্ব
0	0
1	1
2	2
3	4
4	4

আমরা কিউ এর ডগা থেকে 3 কে বের করে আনবো প্রসেসিং এর জন্য। 3 এর দূরত্ব হচ্ছে 4।

3 থেকে যদি আমি 3 এ যাই, তাহলে 0 এর নতুন দূরত্ব হবে 4+6= 10 । কোন দরকার নাই আপডেটের, 0 এর দূরত্ব 0।
3 থেকে যদি আমি 2 এ যাই, তাহলে 2 এর নতুন দূরত্ব হবে 4+2= 6। কোন দরকার নাই আপডেটের, 2 এর দূরত্ব 2।
3 থেকে যদি আমি 4 এ যাই, তাহলে 4 এর নতুন দূরত্ব হবে 4+1= 5। কোন দরকার নাই আপডেটের, 4 এর দূরত্ব 4।

এখন আমাদের কিউ এর অবস্থা এই রকম 2,4,3,4

আর দূরত্বগুলো হচ্ছে

শহর	দূরত্ব
0	0
1	1
2	2
3	4
4	4

এরপর কিউ খালি না হওয়া পর্যন্ত লুপ চলতে থাকবে। কিন্তু এরপর আর কোন আপডেট হবে না। কারণ এটাই আসলে 0 থেকে বাকি সব শহরগুলোতে যাবার সবচে' জন্য কম দূরত্ব।

সিপিপিতে এই অ্যালগরিদমটার ইম্প্লিমেন্টেশন হবে এরকম

```
vector<int> edge[100], cost[100];
const int infinity = 1000000000;

edge[i][j] = jth node connected with i
cost[i][j] = cost of that edge

int bfs(int source, int destination) {

    int d[100];
    for(int i=0; i<100; i++) d[i] = infinity;

    queue<int> q;
    q.push( source );
    d[ source ] = 0;

    while( !q.empty() ) {
        int u = q.front(); q.pop();
        int ucost = d[ u ];

        for(int i=0; i<edge[u].size(); i++) {
            int v = edge[u][i], vcost = cost[ u ][i] + ucost;

            // updating - this part is also called relaxing
            if( d[v] > vcost ) {
                d[v] = vcost;
                q.push( v );
            }
        }
    }

    return d[ destination ];
}
```

দ্বিতীয় অ্যালগরিদম - ডায়াক্সট্রা (Dijkstra)

BFS খুবই ভালো জিনিস। সত্যি কথা আমার ভয়াবহ পছন্দের একটা জিনিস BFS কারণ ধাই ধাই করে BFS কোড লিখে ফেলা যায়, যদি তুমি প্রবলেমটাকে গ্রাফের প্রবলেমে পাল্টাই ফেলতে পারো। অল্প কিছু চালাকি দিয়ে BFS কে ফাস্ট করে ফেলা যায়। একটা হয়তো তুমি এখনই দেখতে পাচ্ছো, একটা শহরকে যদি এরিমধ্যে কিউতে থাকে, ওটাকে আবার কিউতে ঢোকানোর কোন মানে নেই। এ ধরণের চালাকিকে আমরা প্রোগ্রামাররা বলি "প্রুনিঙ" (pruning)। প্রুনিং এর খাস বাংলা হচ্ছে ছাটাই করা - মানে ধরে ছাটাই করে ছোট করে দেয়া।

BFS এর একটা পিছুটান আছে। (পিছুটান বলতে আমি আসলে Draw-back বুঝাচ্ছি, ফেলে আসা স্মৃতির কথা বলছি না) সেটা হচ্ছে, যদি এমন হয়, যে আমি শহর 5 কে যখন প্রসেস করলাম তার দূরত্ব পেলাম 100। তো আমি ওর আশে পাশের সব শহরকে আপডেট করলাম। তারপর আরেকটু পর আবার 5 কে পেলাম কিউতে তখন তার দূরত্ব হচ্ছে 50। আমি আবার আশেপাশের সবাইকে আপডেট করলাম। তারপর আবার কিউতে 5 কে পেলাম। 5 মুখটা পাঁচ করে ভ্যাবাচ্যাকা খেয়ে দাঁড়িয়ে আছে 40 দূরত্ব নিয়ে।

তো তুমি দেখতে পাচ্ছো, আমাদের আপডেটে একটা সমস্যা আছে - আমরা কিউতে যাকে আগে পাচ্ছি তাকে প্রসেস করছি। তো এভাবে প্রসেস করলে এরকম একটা পরিস্থিতি সম্ভব যেখানে আমরা একই শহরকে বহুবার ভুল দূরত্ব পেয়েও আপডেট করতে থাকবো। ওই শহরের উপর যেসব শহর নির্ভরশীল তারাও বহুবার আপডেট হবে। তাদের উপর যারা নির্ভরশীল তারাও বহুবার আপডেট হবে। সব মিলে একটা মহা ক্যাচাল লেগে যাচ্ছে আপডেটের।

Edsger W. Dijkstra ১৯৫৯ সালে এই ক্যাচাল বন্ধ করার একটা পথ খুঁজে পেলো। সে বলল কি, আমরা আরেকটা চালাকি করে দেখতে পারি, কিউ থেকে যখন শহর বের করে আনবো প্রসেস করার জন্য, আমরা সবচে' কাছের শহরটাকে বের করে আনতে পারি। তাহলে সেই শহরটাকে আর কখনো আরেকবার প্রসেস করা লাগবে না।

এটা হচ্ছে উইকিপিডিয়া থেকে নেয়া একটা অ্যানিমেশন ডায়াক্স্ট্রার অ্যালগরিদম এর।

একটা মজার জিনিস হচ্ছে, ডায়াক্স্ট্রা যখন বিয়ে করতে গেলো, বিয়ে রেজিস্ট্রার করার জন্য ওর পেশাতে লিখলো - "কম্পিউটার প্রোগ্রামার", তারপর কাজি অফিসের লোকগুলা বলল মিথ্যা কথা কম বলতে, "কম্পিউটার প্রোগ্রামার" কোন পেশার নামই হতে পারে না। তো সে আরেকটা পেশার নাম লিখলো, তারপর লোকগুলো ওকে বিয়ে করতে দিলো। কে জানতো পন্চাশ বছর পর লাখ লাখ মানুষ "কম্পিউটার প্রোগ্রামিং" করে পয়সা কামাবে?

তো আমাদের কাজ হচ্ছে শুধু কিউটা তুলে দিয়ে সেখানে একটা প্রায়োরিটি কিউ বসানোর, সে নোডটা সবচে' কম দূরত্বে আছে তাকে আমরা সবার আগে প্রসেস করবো - এটা হচ্ছে আমাদের প্রায়োরিটি।

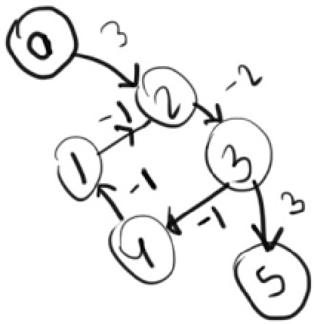
সিপিপি তে একটা সহজ ইম্প্লিমেন্টেশন হবে এরকম -

```
vector<int> edge[100], cost[100];
const int infinity = 1000000000;

edge[i][j] = jth node connected with i
cost[i][j] = cost of that edge

struct data {
    int city, dist;
    bool operator < ( const data& p ) const {
        return dist > p.dist;
    }
};

int dijkstra(int source, int destination) {

    int d[100];
    for(int i=0; i<100; i++) d[i] = infinity;

    priority_queue<data> q;
    data u, v;
    u.city = source, u.dist = 0;
    q.push( u );
    d[ source ] = 0;

    while( !q.empty() ) {
        u = q.top(); q.pop();
        int ucost = d[ u.city ];

        for(int i=0; i<edge[u.city].size(); i++) {
            v.city = edge[u.city][i], v.dist = cost[u.city][i] + ucost;
            // relaxing :)
            if( d[v.city] > v.dist ) {
                d[v.city] = v.dist;
                q.push( v );
            }
        }
    }

    return d[ destination ];
}
```

আমি প্রথম দিকে কেন জানি ডায়াক্সট্রা লিখতে খুব ভয় পেতাম। আমি এখন ঠিক বুঝতে পারি না কেন। আমি কেন ডায়াক্সট্রা লিখতে ভয় পেতাম, সেটা আমার কাছে বিশাল রহস্য। বিশ্বাস করো আর নাই করো - তুমি যদি তোমার ভয়ের সুইচটা কোনভাবে বন্ধ করে দিতে পারো, তোমার ভাঙা গাড়ি রকেটের মতো চলতে শুরু করবে।

তৃতীয় অ্যালগরিদম - বেলম্যান ফোর্ড (Bellman Ford)

গ্রাফটা যদি এরকম হয় তাহলে কি হবে?



আমরা যতবার ইচ্ছা 2-3-4-1 এ পাক খেতে পারি। যতবার পাক খাবো তত আমাদের 0 থেকে 5 এর দূরত্ব কমবে, তাই না? তো এই লুপ অনন্তবার চলতে থাকবে তো চলতে থাকবে। আমাদের BFS তো ফেইল খাবেই খাবে, ডায়াক্সট্রাও ফেইল খাবে।

গ্রাফে যদি নেগেটিভ সাইকেল থাকে - শুধু সেক্ষেত্রেই এই গ্যানজামটা লাগতেসে।
তো এই ক্ষেত্রে আমরা বেলম্যান ফোর্ড ব্যবহার করি। খেয়াল করো, যে একটা কানেক্টেড গ্রাফে যদি একটা নেগেটিভ সাইকেল আমরা পাই, তাহলে গ্রাফটাতে শর্টেস্ট পাথ খোঁজার আর কোন মানে হয় না। বেলম্যান ফোর্ড করে কি যতক্ষণ রিল্যাক্স করা যায়, ততক্ষণ রিল্যাক্স করে। (রিলাক্স মানে আপডেট করা, আরাম করা না কিন্তু - আর সে n-1 এর বেশি আপডেট করে না) তারপর রিলাক্স করা শেষে সে দেখে এখনো কোন রিলাক্স করার মতো edge আছে কিনা। n-1 বার প্রতিটা নোড নিয়ে আপডেট করলে, ততক্ষণে সবগুলো আপডেট করার মতো নোড আপডেট হয়ে যাবার কথা। যদি তা না হয়, তার মানে অবশ্যই নেগেটিভ একটা সাইকেল আছে কোথাও।

এখানে রিল্যাক্স করার মানে হচ্ছে যদি এজটা হয় u থেকে v তে আর দৈর্ঘ্য হয় cost[u][v]

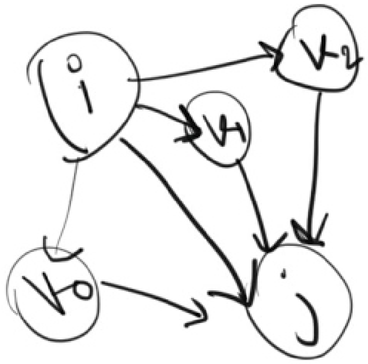
```
if(  d[v] > d[u] + cost[u][v] )
     d[v] = d[u] + cost[u][v];
```

তুমি যাতে এখানে এসে একটা ধাক্কা না খাও, সেজন্য আগের প্রত্যেকবার রিল্যাক্স করার সময় কমেন্টে আমি আলাদা করে বলে নিয়েছি যে আমি রিল্যাক্স করছি।

বেলম্যান ফোর্ডের একটা সহজ সুডোকোড উইকিপিডিয়াতে আছে। যেহেতু তুমি এখন শর্টেস্ট পাথ নিয়ে অনেক কিছু জানো, তোমার একটুও কষ্ট হবে না এটা বুঝতে।

চতুর্থ অ্যালগরিদম - ফ্লয়েড ওয়ার্শাল (Floyd-Warshall)

ফ্লয়েড ওয়ার্শাল এর মূল কন্সেপ্টটা খুব সহজ। তুমি যদি কন্সেপ্টটা বোঝো, তুমি যেকোন ফ্লয়েড ওয়ার্শাল প্রবলেম সলভ করতে পারবে। কিন্তু অ্যালগরিদমটা কেন কাজ করে সেটা বোঝার জন্য তুমি এখানে একটা ঢু মারতে পারো। আমার এই সাধাসিধা টিউটোরিয়ালটা দিয়ে আমি আপাতত কাওকে ভয় পাওয়াতে চাই না কঠিন কিছু লিখে।



আমি যদি i থেকে j তে যেতে চাই তাহলে , হয় আমি সোজা i থেকে j তে যাবো, অথবা কোন একটা k হয়ে j তে যাবো। যখন আমি k হয়ে যাবো তখন আমার i থেকে j এর সবচে' ছোট দূরত্ব dist[i][j] হবে dist[i][k] + dist[k][j] । এখানে dist[a][b] মানে a থেকে b যাওয়ার সবচে' ছোট পথটার দূরত্ব। আমি যদি সবগুলো k এর জন্য ট্রাই করি একবার করে, তাহলে আপডেট করতে থাকলে আপডেট করার শেষে অবশ্যই dist[a][b] হবে a থেকে b তে যাওয়ার সবচে' ছোট পথ।

তাহলে অ্যালগরিদমটা দাড়াচ্ছে এরকম

```
int d[100][100]; // d[i][j] = distance from i to j

for(int k=0; k<n; k++)
for(int i=0; i<n; i++)
for(int j=0; j<n; j++) {
    if( d[i][j] > d[i][k] + d[k][j] )
        d[i][j] = d[i][k] + d[k][j];
}
```

ফ্লয়েড ওয়ার্শালের মজা হচ্ছে এটা লেখা খুব সহজ। সত্যি কথা আমি যখন লিখি কন্টেস্টের সময় ম্যাকরো লাগিয়ে তখন সেটা দেখতে এরকম হয়

```
REP(k,n) REP(i,n) REP(j,n) d[i][j] = min( d[i][j], d[i][k] + d[k][j] );
```

আর আরেকটা সুবিধা হচ্ছে, বাকি সব অ্যালগরিদম একটা শহরকে ধরে নেয় শুরুর শহর (source), তারপর বাকি সবগুলো শহরের দূরত্ব বের করে ওই শহরটা থেকে। এটাকে আমরা বলি Single Source Shortest Path (SSSP)। ফ্লয়েড ওয়ার্শাল করে কি সবগুলো শহর থেকে সবগুলো শহরের শর্টেস্ট পাথ বের করে ফেলে। তোমার অ্যলগরিদম শেষে তুমি যদি জানতে চাও a থেকে b এর দূরত্ব কত সেটা তুমি dist[a][b] থেকেই পেয়ে যাবে।

গল্প, যদি শুনতে চাও

খুব চাছাছোলাভাবে,

- BFS এর কম্প্লেক্সিটি হচ্ছে O(n^2)
- ডায়াক্সট্রার কম্প্লেক্সিটি হচ্ছে O(n log n)
- বেলম্যান ফোর্ড O(n^3)
- ফ্লয়েড ওয়ার্শাল O(n^3)

যখন ইনপুটের সাইজ দেখবা, যেটা লিখতে সবচে' সোজা আর যেই অ্যালগরিদমটা কাজ করবে, সেটা লিখবা। যেমন ধরো n যদি 100 এর ছোট হয়, কোন দরকার নাই BFS লিখে মারা যাবার। একটা ছোট্ট ফ্লয়েড লিখে ফেলো। আবার হিসেব করে যদি দেখো যে BFS এ পোষাচ্ছে ভুলেও ডায়াক্সট্রা লিখতে যেও না। আর খেয়াল রেখো গ্রাফটাতে নেগেটিভ সাইকেল আছে কিনা। যদি থাকে সাবধানে বেলম্যান ফোর্ড লিখে ফেইলো।

আজকে আমার একদম এনার্জি শেষ। :(
এর পরের টিউটোরিয়ালটা হবে শর্টেস্ট পাথ এর প্রবলেমগুলা নিয়ে। কিভাবে একটা বান্দর আর লাঠি ওয়ালা প্রবলেমকে শহর আর পথের প্রবলেম বানিয়ে ফেলা যায় সেটা দেখাবো (এটাকে বলে মডেলিং - সত্যি সত্যি!) আর প্র্যাকটিস করার জন্য কিছু প্রবলেমের লিস্ট দিয়ে দিবো। আপাতত যদি প্র্যাকটিস করতে চাও এই ফাইলটায় উঁকি মারতে পারো।

অনেক অনেক শুভ কামনা! :)

Comments