

# ডাইনামিক প্রোগ্রামিং এ হাতেখড়ি-৪

shafaetsplanet.com/planetcoding

০-১ ন্যাপস্যাক, কয়েন চেঞ্জ প্রবলেমে আশা করি তোমার এখন দক্ষতা এসে গিয়েছে। এই পর্বে আমরা দেখবো LIS, একই সাথে দেখবো কিভাবে ডিপিতে সলিউশন প্রিন্ট করতে হয়, এটা নিয়ে তোমাদের অনেকেরই সমস্যা হয়েছে বলে জানিয়েছো। এছাড়া আগের পর্বে light oj 1231 প্রবলেমটি সলভ করতে বলেছিলাম, আমার সলিউশন পাবে লেখার একদম শেষে।

প্রথমেই শুরু করি LIS এবং এটার সলিউশন প্রিন্ট করা দিয়ে। LIS হলো Longest increasing subsequence। মনে করো তোমাকে একটি অ্যারে বা sequence দেয়া আছে:

এই অ্যারে থেকে কিছু সংখ্যা মুছে দিয়ে এবং অর্ডার ঠিক রেখে আমরা বিভিন্ন subsequence পেতে পারি। যেমন:

1	2	3	4	5	6	7
5	0	9	2	7	3	4

Shafaetsplanet.com/blog

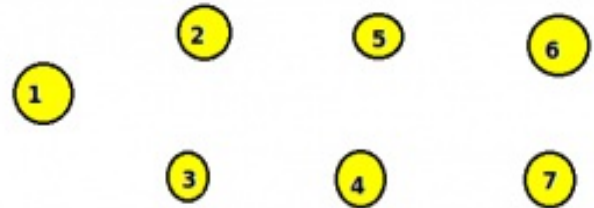
5 7  
5 9 2  
0 3 4  
0 2 3 4  
7 3  
5 0 9 2 7 3 4 (শুনারি সংখ্যা বাদ দেয়া হয়েছে)  
....

একটা sequence এ n টি সংখ্যা থাকলে subsequence থাকতে পারে মোট  $2^n$  টি (কেন??)। increasing subsequence (IS) এ প্রতিটা সংখ্যা তার আগের সংখ্যাটির থেকে বড় হবে। উপরে ১ম, ৩য়, ৪র্থ subsequence গুলো increasing। যতগুলো IS আছে তারমধ্যে সবথেকে বড়টা খুঁজে বের করাই আমাদের লক্ষ্য। উপরে ৪নম্বরটির length ৪ এবং সবগুলো IS এর মধ্যে এটাই সব থেকে বড়।

প্রবলেমটাকে গ্রাফ দিয়ে মডেলিং করলে খুব সহজে সলভ করা যায়। গ্রাফ খিওরি না জানলেও কোনো সমস্যা হবে না, গ্রাফ দেখাচ্ছি খালি বোঝার সুবিধার জন্য। প্রথমেই মনে করি অ্যারের প্রতিটি ইনডেক্স হলো একটি করে নোড:

এখন দুটি শর্ত খেয়াল করো। u নম্বর নোড থেকে v নম্বর নোডে যাওয়া যাবে যদি:

১.  $v > u$  হয়। কারণ আমাদের মূল sequence এর অর্ডার ঠিক রাখতে হবে।
২.  $value[v] > value[u]$  হয়, কারণ sequence টা increasing হতে হবে।



1	2	3	4	5	6	7
5	0	9	2	7	3	4

Shafaetsplanet.com/blog

তাহলে ১ থেকে ২ এ যেতে পারবোনা কারণ  $value[1]$

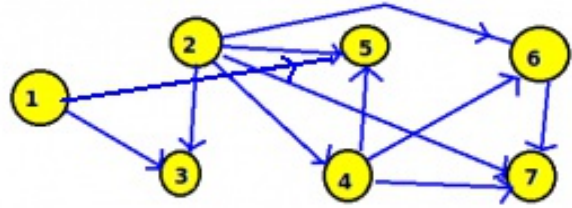
দেখতে হিজিবিজি হলেও ভয় পাবার কিছু নাই, উপরের শর্ত

দুটি মনে গ্রাফটি আকা হয়েছে। গ্রাফের যেকোনো পাথ অনুসরণ করলে আমরা একটা ইনক্রিসিং সিকোয়েন্স পাবো। যেমন গ্রাফে ২-৬-৭ পাথে গেলে ০-৩-৪ সাবসিকোয়েন্সটা পাবো। তাহলে বুঝতেই পারছো সবথেকে লম্বা পথটাই আমার সলিউশন, যতদূর যেতে পারবো সিকোয়েন্স তত বড় হবে, এক্ষেত্রে একটি path এ যে কয়টি নোড আছে সেটাই পাথের দৈর্ঘ্য।

মনে করি আমাদের একটা ফাংশন আছে  $longest(u)$  যেটা u নম্বর নোড থেকে longest path রিটার্ন করে। যেমন  $longest(6)=2$  কারণ ৬ নম্বর নোড থেকে খালি ৬-৭ এই পাথে যাওয়া যায়। এখন লক্ষ্য করো:

কোনো নোড থেকে longest path হবে সেই নোড থেকে যেসব নোডে যাওয়া যায় সেগুলো থেকে longest path এর maximum + ১।

আমরা সহজেই একটা রিকার্সিভ রিলেশন বের করে ফেলতে পারি:



1	2	3	4	5	6	7
5	0	9	2	7	3	4

Shafaetsplanet.com/blog

$longest(2) = 1 + \max(longest(3), longest(4), longest(5), longest(6), longest(7))$

$longest(1) = 1 + \max(longest(3), longest(5))$

$longest(3) = 1$  (এটা একটা বেসকেস কারণ ৩ থেকে কোথাও যাওয়া যায়না)

.....

অর্থাৎ  $u$  থেকে  $v_1, v_2, v_3, \dots, v_k$  নোডে যাওয়া গেলে:

$longest(u) = 1 + \max(longest(v_1), longest(v_2), \dots, longest(v_k))$

এখন আমরা সহজেই কোড করে প্রতিটা নোড থেকে longest path বের করে ফেলতে পারি:

C++

```

1  #define mx 1000
2  int n=7;
3  int value[]={-100000,5,0,9,2,7,3,4};
4  int dp[mx],dir[mx];
5  int longest(int u)
6  {
7      if(dp[u]!=-1) return dp[u];
8      int maxi=0;
9      for(int v=u+1;v<=n;v++) //১ম শর্ত,v>u
10     {
11         if(value[v]>value[u]) //২য় শর্ত, value[v]>value[u]
12         {
13             if(longest(v)>maxi) //সর্বোচ্চ মানটা নিবো
14             {
15                 maxi=longest(v);
16                 dir[u]=v;
17             }
18         }
19     }
20     dp[u]=1+maxi; //১ যোগ হবে কারণ u নম্বর নোডটাও পথের মধ্যে আছে
21     return dp[u];
22 }
23
24 int main()
25 {
26     READ("in");
27     memset(dp,-1,sizeof dp);
28     memset(dir,-1,sizeof dir);
29     int LIS=0,start;
30     for(int i=1;i<=n;i++)
31     {
32         printf("longest path from: %d\n",longest(i));
33         if(longest(i)>LIS)
34         {
35             LIS=longest(i);
36             start=i;
37         }
38     }
39     printf("LIS = %d Starting point %d\n",LIS,start);
40
41     return 0;
42 }

```

---

longest(u) প্রতিটা নোড u থেকে longest path রিটার্ন করে,এদের মধ্যে আবার যেটা ম্যাক্সিমাম সেটাই আমাদের LIS। এই সলিউশনটা কমপ্লেক্সিটি  $O(n^2)$  (আরো ভালো একটা সলিউশন আছে [বাইনারী সার্চ](#) ব্যবহার করে)। এখন সলিউশন প্রিন্ট করার পালা। আসলে মূল কাজটা আমরা উপরের কোডেই করে ফেলছি। dir[] নামের একটি অ্যারে রেখেছি যে প্রতিটা নোডের জন্য যে ডিরেকশনে সর্বোচ্চ মান পাওয়া যায় সেটা সেভ করে রাখে। কোনো নোড থেকে অন্য কোথাও যাওয়া না গেলে ডিরেকশন হবে -১। এখন সলিউশন ফাংশন হবে এরকম:

C++

```

1 void solution(int start)
2 {
3     while(dir[start]!=-1)
4     {
5         printf("index %d value %d\n",start,value[start]);
6         start=dir[start];
7     }
8 }

```

---

অর্থাৎ ডিরেকশন অ্যারে দেখে আমরা সামনে যাবো যতক্ষণ যাওয়া যায়।

এখানে লক্ষ্য করার কিছু বিষয় হলো উপরের গ্রাফটিতে কোনো সাইকেল নেই দেখে আমরা ডিপি করতে পেরেছি,এধরণের গ্রাফকে বলা হয় **directed acyclic graph** বা ড্যাগ। কোনো প্রবলেমকে তুমি যদি ড্যাগ এ কনভার্ট করতে পারো তাহলে খুব ভালো সম্ভাবনা আছে যে প্রবলেমটি ডিপি চালিয়ে সলভ করা যাবে। গ্রাফে সাইকেল থাকলে **longest path** একটি **np-complete** প্রবলেম,অর্থাৎ পলিনমিয়াল সলিউশন জানা নেই। এখন একটি প্রশ্ন: একটা **sequence** এর কয়টি **LIS** আছে সেটা বের করতে বলা হলে কি করবে? চিন্তা করে জানাও :-)

আমরা জানি ০-১ ন্যাপস্যাকে স্টেট থাকে ২টা,সেক্ষেত্রে ডিরেকশন অ্যারের স্টেটও হবে ২টা। নিচের কোডটা দেখো:

C++

```

1 int dir[][]={{-1}};
2 int dp[][]={{-1}};
3 int func(int i,int w) //i নম্বর আইটেম নিয়ে চেষ্টা করা হচ্ছে,w ওজনের জিনিস নেয়া হয়েছে
4 {
5     .....
6     //BASE CASE
7     .....
8
9     if(w+weight[i]<=CAP) //i নম্বর জিনিসটি নিবো
10         profit1=cost[i]+func(i+1,w+weight[i])
11     else
12         profit1=0;
13     profit2=func(i+1,w) // i নম্বর জিনিসটি নিবো না
14     if(profit1>profit2){dir[i][w]=1; return dp[i][w]=profit1;}
15     else {dir[i][w]=2; return dp[i][w]=profit2;}
16 }

```

---

ন্যাপস্যাকের কোডটাকে সামান্য পরিবর্তন করে ডিরেকশন রাখা হয়েছে। আমাদের **starting state** হলো (i=1,w=0),কারণ শুরুতে আমরা ১ নম্বর জিনিসটা নিয়ে ট্রাই করবো এবং এখন পর্যন্ত নেয়া জিনিসের ওজন ০। ডিপি শেষ হবার পর যদি:

$dir[i][w] = 1$  হয় তাহলে  $i$  নম্বর জিনিসটি নিবো,  $i$  প্রিন্ট করে আমরা চলে যাবো  $(i+1,w+weight[i])$  স্টেটে।

$dir[i][w] = 2$  হয় তাহলে  $i$  নম্বর জিনিসটি নিবো না, $i$  প্রিন্ট না করেই আমরা চলে যাবো  $(i+1,w)$  স্টেটে।

$dir[i][w] = -1$  হলে আমরা থেমে যাবো।

এটাই হলো ডিপির সলিউশন প্রিন্টের মূল কথা। ডিপি অ্যারের পাশাপাশি ডিরেকশন অ্যারেতে সেভ করে রাখবো কোন স্টেট থেকে কোন স্টেটে যাচ্ছি সেটা। এরপর **starting state** থেকে ডিরেকশন অনুযায়ী আগাতে থাকবো আর প্রিন্ট করবো।

এক স্টেট থেকে একাধিক স্টেটে গেলে, আমরা সেভ করবো যে স্টেটে গেলে  
ম্যাক্সিমাইজ বা মিনিমাইজ বা যে স্টেট থেকে ড্যালিড আউটপুট পাওয়া যায় শুধু সেটা

shafaetsplanet.com/blog

```
struct next
{
    int a,b;
    next(int _a,int _b){a=_a,b=_b;}
    next(){}
}direction[100][100];
int dp[100][100];
int call(int a,int b)
{
    .....
    .....
    int ret1=call(a+1,b);
    int ret2=call(a,b+1);
    if(ret1>ret2) {
        dp[a][b]=ret1;
        direction[a][b]=next(a+1,b);
    }
    else {
        dp[a][b]=ret2;
        direction[a][b]=next(a,b+1);
    }
    return dp[a][b];
}
```

ডিপি অ্যারেতে রিটার্ন ভ্যালু সেভ করছি,  
আর ডিরেকশন অ্যারেতে কোন দিকে যাচ্ছি সেটা সেভ করছি!

প্রিন্ট করার সময় আমরা ডিরেকশন অ্যারের দেখানো  
পথে শুধু সামনে এগিয়ে যাবো!

```
void go(int a,int b)
{
    //Dont forget to return from function when base case reached
    //otherwise you'll fall into infinite loop
    int next_a=dir[a][b].a;
    int next_b=dir[a][b].b;
    if(next_a!=a) puts("Took a");
    else puts("Took b");
    go(next_a,next_b);
}
```

আজকের পর্ব এখানেই শেষ। light oj 1231 এর সলিউশন: <http://pastebin.com/vmpWp0g1>, mod করার অংশটা বুঝতে না পারলে এই লেখাটা দেখো। সলভ করার জন্য প্রবলেম:

Diving for gold  
Sense of Beauty  
Bicolored Horses  
Testing the CATCHER  
How Many Dependencies?  
Longest Path

পরের পর্ব-বিটমাস্ক ডিপি  
সবগুলো পর্ব