

ডাইনামিক প্রোগ্রামিং এ হাতেখড়ি-২

shafaetsplanet.com/

শাফায়েত

মে ২৩, ২০১২

(১ম পর্ব) (সবগুলো পর্ব)

১ম পর্বে আমরা জেনেছি ডাইনামিক প্রোগ্রামিং কাকে বলে, প্রবলেমে কি রকমের বৈশিষ্ট্য থাকলে সেটা ডাইনামিক প্রোগ্রামিং এর সাহায্যে সমাধান করা যায়। আমরা দেখেছি ডিপি দিয়ে কিভাবে ফিবোনাচ্চি সংখ্যার রিকার্সনের রানটাইম অনেক কমিয়ে আনা যায়। তবে ডিপি এমন একটা জিনিস যে এতকিছু জেনেও তুমি কিছুই সমাধান করতে পারবেনা যদিনা খুব ভালো করে প্র্যাকটিস করো আর চিন্তা করো। তবে এটা শুনে ভয়ের একদমই কিছু নেই, প্র্যাকটিস করতে থাকলে কিছুদিন পর দেখবে অনেক সহজেই রিকার্সিভ ফাংশন বের করে ডিপি প্রবলেম সলভ করে ফেলতে পারছো, আমার কাজ হলো তোমাকে শুরু করিয়ে দেয়া। সব শেষে hexabonacci নামের একটি প্রবলেম সলভ করতে দিয়েছিলাম, আশা করি সবাই প্রবলেমটি খুব সহজেই সমাধান করে ফেলেছে। সমাধান করতে না পারলে আমি বলবো আগের পর্বটা আরেকবার খুব ভালোভাবে পড়ে ফেলতে আর কোনো জায়গায় না বুঝলে মন্তব্য অংশে জানাতে। প্রবলেমটির সাথে ফিবোনাচ্চি প্রবলেমটার খুব একটা পার্থক্য নেই বলে আমি সমাধান নিয়ে আলোচনা করছি।

n টা বস্তুর মধ্য থেকে r টা বস্তু কত ভাবে নেয়া যায়?

এখন আমরা nC_r এর মান কিভাবে ডাইনামিক প্রোগ্রামিং এর সাহায্যে বের করা যায় সেটা নিয়ে আলোচনা করবো। n টা জিনিসের মধ্য থেকে r টা জিনিস কতভাবে নেয়া যায় সেটাই nC_r বা $nCr(n,r)$ । এখন যেকোনো একটি জিনিসের কথা চিন্তা কর। তুমি যদি জিনিসটা না নাও তাহলে বাকি $n-1$ টা জিনিসের মধ্য থেকে আরো r টা জিনিস নিতে হবে, সেটা করা যায় $nCr(n-1,r)$ ভাবে। আর যদি তুমি জিনিসটা নাও তাহলে বাকি $n-1$ টা জিনিসের মধ্য থেকে আরো $r-1$ টা জিনিস নিতে হবে, সেটা করা যায় $nCr(n-1,r-1)$ ভাবে। এই দুইটার যোগফলই হলো মোট উপায়।

$${}^nC_r = {}^{n-1}C_r + {}^{n-1}C_{r-1}$$

আমরা যদি মনে করি $nCr(n,r)$ হলো একটি ফাংশন যা n আর r এর মানকে প্যারামিটার হিসাবে গ্রহণ করে তাহলে আমরা উপরের রিলেশন তাকে এভাবে লিখতে পারি:

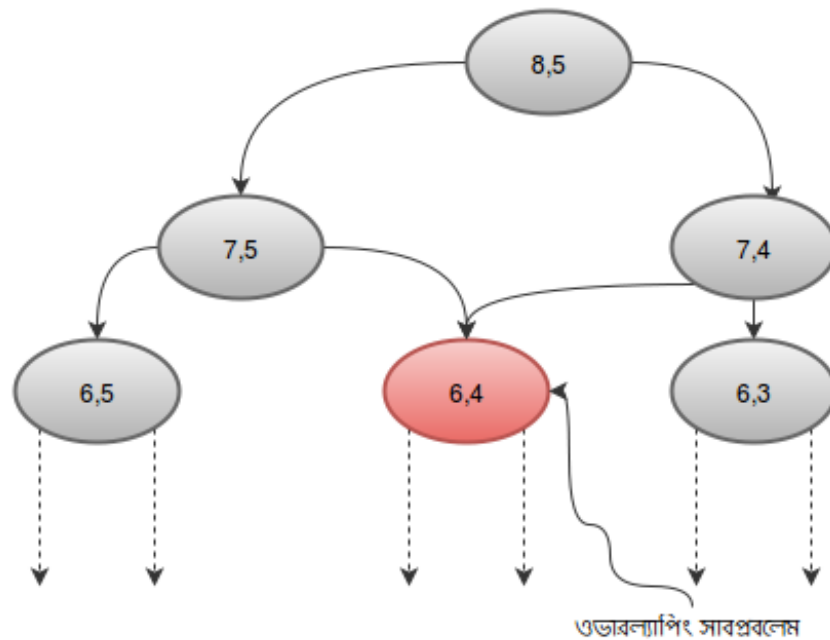
$$nCr(n,r) = nCr(n-1,r) + nCr(n-1,r-1)$$

এবার হয়তো জিনিসটা আরো স্পষ্ট হয়েছে তোমার কাছে। কিন্তু রিকার্সনটা শেষ হবে কখন? অর্থাৎ বেস কেস কি? আমরা জানি $nCr(n,1)=n$ এবং $nCr(n,n)=1$ । এ দুটি শর্ত বেস কেস হিসাবে ব্যবহার করলে কোডটা হবে:

Python

```
1 def nCr(n, r):
2     if r==1: return n
3     if n==r: return 1
4     return nCr(n-1, r) + nCr(n-1, r-1)
```

ফাংশন কল গুলো লক্ষ্য করলে দেখতে পাবে ফিবোনাচ্চির মতো এখানেও একটি ফাংশন বার বার কল হচ্ছে:



আমরা এই অতিরিক্ত ফাংশন কল সহজেই এড়াতে পারি আগের মতো একটা টেবিল রেখে। পার্থক্য হলো এবার টেবিলটা হবে ২-ডি, আর কোনো পার্থক্য নেই। আগে টেবিলটার সব ঘরে -1 রেখে নিবো, -1 দিয়ে বুঝাচ্ছে ঘরটি খালি আছে।

Python

```
1 table = [[-1 for i in range (0,50)] for i in range(0, 50)] //Create a 50*50 table with -1 in each cell
2 def nCr(n, r):
3     if r==1: return n
4     if n==r: return 1
5     if table[n][r] != -1:
6         return table[n][r]
7     table[n][r] = nCr(n-1, r) + nCr(n-1, r-1)
8     return table[n][r]
```

আমাদের সবগুলো ডিপি কোডই মোটামুটি একটা ফরমেট মেনে চলবে, সেটা এরকম:

```
1 def function(state1, state2, ..., staten)
2     Return if reached any base case
3     Check table and return if the value is already computed
4     Calculate the value recursive for this state
5     Save the value in the table and return
6
```

nCr এর ক্ষেত্রে প্যারামিটার ছিলো n এবং r । বুঝতেই পারছেন যতগুলো প্যারামিটার থাকবে মান টেবিলে সেভ করার জন্য তত ডাইমেনশনের টেবিল লাগবে। স্টেট বেশি লাগলে মেমরিও বেশি লাগে। অ্যারের প্রতিটি ডাইমেনশনের সাইজ প্রয়োজনমতো প্যারামিটার অনুসারে কমবেশি হবে। প্যারামিটারকে আমরা সাধারণত বলে থাকি স্টেট(State)।

ডিপিতে সবথেকে গুরুত্বপূর্ণ অংশ হলো স্টেট বুঝতে পারা। ধরো তোমাকে ৫০মিনিটের মধ্যে একটা ক্লাস ধরতে হবে। এখন রাস্তায় তুমি কোন অবস্থায় আছো সেটা আমি জানতে পারবো যদি তুমি আমাকে দুটি তথ্য দাও: তুমি এই মুহূর্তে কোন জায়গায় আছো আর তুমি বাসা থেকে বের হবার পর কয় মিনিট পার হয়েছে। যেমন হয়তো তুমি ফার্মগেটে আছো আর বাসা থেকে বের হবার পর ২০ মিনিট পার হয়েছে। তুমি কি রঙের জামা পড়েছো বা তুমি কোন জুতা পড়েছো এটা কিন্তু এখানে গুরুত্বপূর্ণ না তাই এটা "স্টেট" এর মধ্যে পড়েনা।

০-১ ন্যাপস্যাক

কোনো এক রাতে তুমি চুরি করতে বের হলে!! বন্ধুর বাসায় জানালা দিয়ে ঢুকে দেখলে প্রচুর জিনিসপত্র,কিন্তু তোমার চুরি করার খলেতে জায়গা আছে মাত্র ১০ইউনিট,এর বেশি নিলে খলে ছিড়ে যাবে। প্রতিটা জিনিসের ওজন আছে আর একেক জিনিসের মূল্যও একেকরকম।

১. মানিব্যাগ: ১ পাউন্ড, ১২০টাকা
২. কোরম্যানের-বই: ৭ পাউন্ড, ৪০০টাকা
৩. ডিভিডি-কালেকশন: ৪ পাউন্ড, ২৮০ টাকা
৪. ফেলুদা-সমগ্র: ৩ পাউন্ড, ১৫০টাকা
৫. ফুটবল: ভর: ৪ পাউন্ড, ২০০টাকা

কোনো জিনিস নিলে পুরোটাই নিতে হবে, ৪টি ডিভিডির ২টি তুমি নিতে পারবেনা,ফেলুদা সমগ্রের অর্ধেক ছিড়ে আনতে পারবেনা। প্রথমদিন চুরি করতে গিয়েছো এই জন্য কিছু না ভেবেই তুমি ফটাফট দামি জিনিসগুলো ভরতে থাকলে। প্রথমেই তুমি ৪০০টাকার কোরম্যানের বই নিয়ে নিলে,তারপর ১৫০টাকার ফেলুদা সমগ্র নিয়ে বাসায় ফিরে আসলে,তোমার লাভ হলো ৫৫০ টাকা। বাসায় এসে হিসাব করে দেখলে তুমি যদি লোভীর মতো (greedy) দামী জিনিসগুলো আগে না নিয়ে একটু ভেবে-চিন্তে নিতে তাহলে ২৮০টাকার ডিভিডি,২০০টাকার ফুটবল আর ১২০টাকার মানিব্যাগ নিয়ে ফিরতে পারতে, তোমার লাভ হতো ৬০০টাকা। greedy অ্যালগোরিদমে অপটিমাল রেজাল্ট না পাওয়ায় তুমি চিন্তা করলে সবরকমের কন্সনেশনে চেষ্টা করবে, প্রতিবার একটা করে জিনিস খলেতে ভরবে আর দেখবে আর কত লাভ করা যায়,প্রয়োজনে জিনিসটা খলে থেকে নামিয়ে রেখে আরেকটি নিয়ে চেষ্টা করবে। এবং সুবিধার জন্য তুমি লিস্টের সিরিয়াল অনুযায়ী জিনিস নিয়ে চেষ্টা করবে, ৩ নম্বর জিনিসের পরে ১ নম্বর জিনিস নিতে চেষ্টা করবেনা। কোনো সময় তোমার অবস্থা বুঝতে ২টা তথ্যই যথেষ্ট।

১. তুমি এখন কত নম্বর জিনিস ট্রাই করছো।
২. এখন পর্যন্ত তুমি কত ইউনিট জিনিস নিয়েছো।

তুমি একটা ফাংশন লিখে ফেললে যেটা জিনিসপত্রের লিস্ট থেকে অপটিমাল রেজাল্ট বের করে দিতে পারে। ফাংশনটির রিটার্ন টাইপ আর প্যারামিটার হবে এরকম:

```
1 func(int i,int w)
```

মনে করি i নম্বর বস্তুটির ভর হলো `weight[i]` আর মূল্য `cost[i]`। আর ব্যাগের `capacity=CAP`। প্রতিবার তুমি i নম্বর জিনিসটি নিতে পারো যদি ব্যাগে জায়গা থাকে, অথবা তুমি i নম্বর জিনিসটি না নিয়ে i+1 তম জিনিস ট্রাই করতে পারো। i নম্বর জিনিসটি যদি তুমি নাও তাহলে লাভ হবে **`cost[i] + পরবর্তি স্টেটে লাভ`**, তাহলে আমরা `cost[i]` যোগ করে পরবর্তি স্টেটে চলে গিয়ে কত লাভ হয় সেটা হিসাব করবো।

```
1 if(w+weight[i]<=CAP)
2   profit1 = cost[i] + func(i + 1,w + weight[i])
3 else
4   profit1=0;
```

i নম্বর জিনিসটি যদি তুমি না নাও তাহলেও লাভ বেশি হতে হবে তাই সেটাও আমাদের হিসাব করতে হবে:

```
1 profit2=func(i+1,w) //নতুন ওজন যোগ হচ্ছেনা,i+1 তম বস্তু নিয়ে ট্রাই করছি।
```

লক্ষ্য করো এক্ষেত্রে কারেন্ট প্রফিট বা ওজনের কোনো পরিবর্তন হচ্ছেনা, কোনো কিছু না নিয়েই পরের স্টেটে গিয়ে দেখছি লাভ কত হবে। তাহলে সহজেই বোঝা যাচ্ছে অপটিমাল রেজাল্টের জন্য আমাদের profit1 আর profit2 এর মধ্যে বড়টা নিতে হবে,সেই মানটা আমরা রিটার্ন করে দিবো।

```
1 return max(profit1,profit2)
```

তাহলে যদি তুমি func(1,0) কল করো তাহলে রিকার্নশন তোমাকে রেজাল্ট বের করে দিবে। func(1,0) কল করার কারণ হলো শুরুতে তুমি ১ নম্বর জিনিসটা নিয়ে ট্রাই করবে এবং শুরুতে ব্যাগে সম্পূর্ণ খালি। যখন সবগুলো জিনিস নিয়ে ট্রাই করা হয়ে যাবে তখন রিকার্নশন শেষ হবে। যদি \$n\$ টি জিনিস থাকে তাহলে বেস কেস হবে:

```
1 if(i==n+1) //Base Case
2 return 0
```

শূন্য রিটার্ন করছি কারণ সবকিছু নেয়া হয়ে গেলে আর প্রফিট করা সম্ভবনা।

এই কোডে উপরের উদাহরণের ইনপুটটা নিচের মতো করে দিলে দিলে ৬০০ আউটপুট আসবে

```
4 10 //৫ টা জিনিস, ১০ ক্যাপাসিটি
```

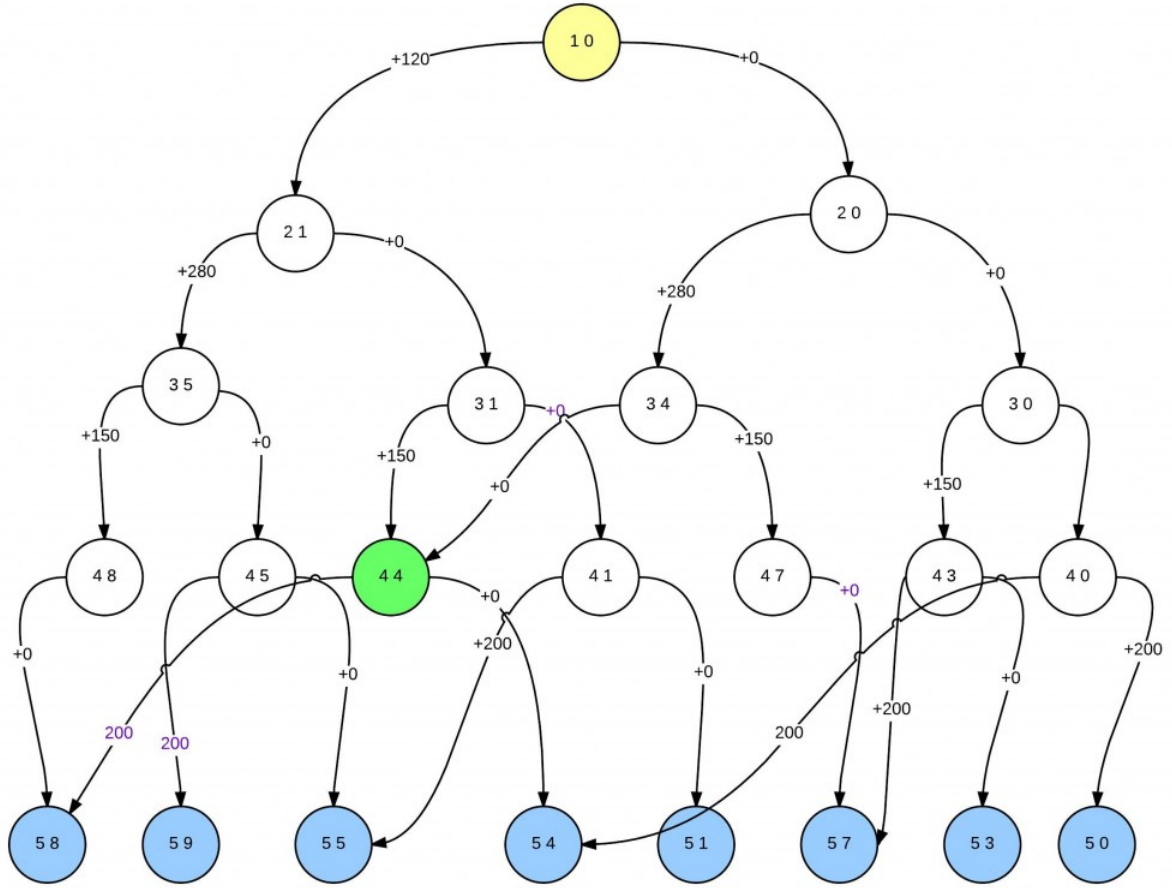
```
1 120 //প্রতিটা জিনিসের ওজন এবং দাম
```

```
4 280
```

```
3 150
```

```
4 200
```

আমরা উপরের ইনপুটের জন্য কিভাবে ফাংশন কলে হচ্ছে তার একটা সিমুলেশন দেখি:



ছবিতে প্রতিটা নোড প্রকাশ করছে আমরা এখন কোন জিনিসটা নিয়ে কাজ করছি এবং ব্যাগে কত ওজনের জিনিস আছে। তারপর জিনিসটা একবার ব্যাগে ভরে(যদি জায়গা থাকে) নতুন নোডে যাচ্ছি, আরেকবার না ভরে নতুন জায়গায় যাচ্ছি। নীল রঙ এর নোডগুলো হলো বেস কেস, কারণ আর কোনো জিনিস নেয়ার মত নেই। লক্ষ্য কর সবুজ রঙ এর নোডটায় একাধিক পথে আসা যায়, তাই আমরা প্রতিটি নোডের রিটার্ন ভ্যালু ডিপি অ্যারেতে সেভ করে রাখবো।

কমপ্লেক্সিটি:

আমাদের কোডে ভিন্ন ভিন্ন স্টেট সংখ্যা $n \times \text{cap}$ টা যেখানে n হলো জিনিস সংখ্যা আর cap হলো ব্যাগের ক্যাপাসিটি। ফাংশনের ভিতর কোনো লুপ নেই। তাই মোট $O(n \times \text{cap})$ টাইম কমপ্লেক্সিটিতে কোডটা চলবে। ডিপি অ্যারের আকার হবে $n \times \text{cap}$, তাই স্পেস কমপ্লেক্সিটিও $O(n \times \text{cap})$ ।

এখন ছোট্ট একটা প্রশ্ন, ডিপি প্রবলেমে টাইম আর স্পেস কমপ্লেক্সিটি ভিন্ন হবে কখন?

এটাই হলো ক্লাসিকাল 0-1 knapsack প্রবলেম, ন্যাপস্যাক শব্দটির অর্থ হলো থলে। ০-১ নাম দেয়ার কারণ হলো তুমি কোনো জিনিস অর্ধেক নিতে পারবে না, হয় পুরোটা নিবে অথবা নিবে না। fractional-knapsack বলে আরেক ধরনের প্রবলেম আছে, সেটার আলোচনা অন্য কোনোদিন হবে। ন্যাপস্যাকে সলিউশন প্রিন্ট করা শিখতে সিরিজের চতুর্থ পর্বটা দেখো।

এখন তোমার এখন কাজ হলো এটার সম্পূর্ণ কোড ইমপ্লিমেন্ট করা। uva 10130 প্রবলেমটি সলভ করতে চেষ্টা করো।

ডাইনামিক প্রোগ্রামিং এ ভালো করার একমাত্র উপায় হলো প্রচুর চিন্তা করা। তোমাকে অনুরোধ করবো পরবর্তী পর্ব পড়ার আগে অবশ্যই আগের পর্বের সমস্যাগুলো নিয়ে খুব ভালো করে অন্তত ১দিন চিন্তা করবে।

(সবগুলো পর্ব)

পরের পর্ব-কয়েন চেঞ্জ,রক ক্লাইম্বিং