ডাইনামিক প্রোগ্রামিং এ হাতেখড়ি-২

(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 রিকার্শনের 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 করছিনা।

আরেকটি কাজ দিয়েছিলাম,সেটা হলো nCr এর মান বের করা। এটার সমাধান নিয়ে এখন আলোচনা করবো। কলেজে থাকতে হয়তো পড়েছো:

$$ {}^{n}C_{r} = {}^{(n-1)}C_{r} + {}^{(n-1)}C_{(r-1)} $$

আমরা এই রিলেশনটি ব্যবহার করে nCr এর মান বের করবো, সুত্রের প্রমান লেখার একদম শেষ অংশ দেখে আসতে পারো। 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)*

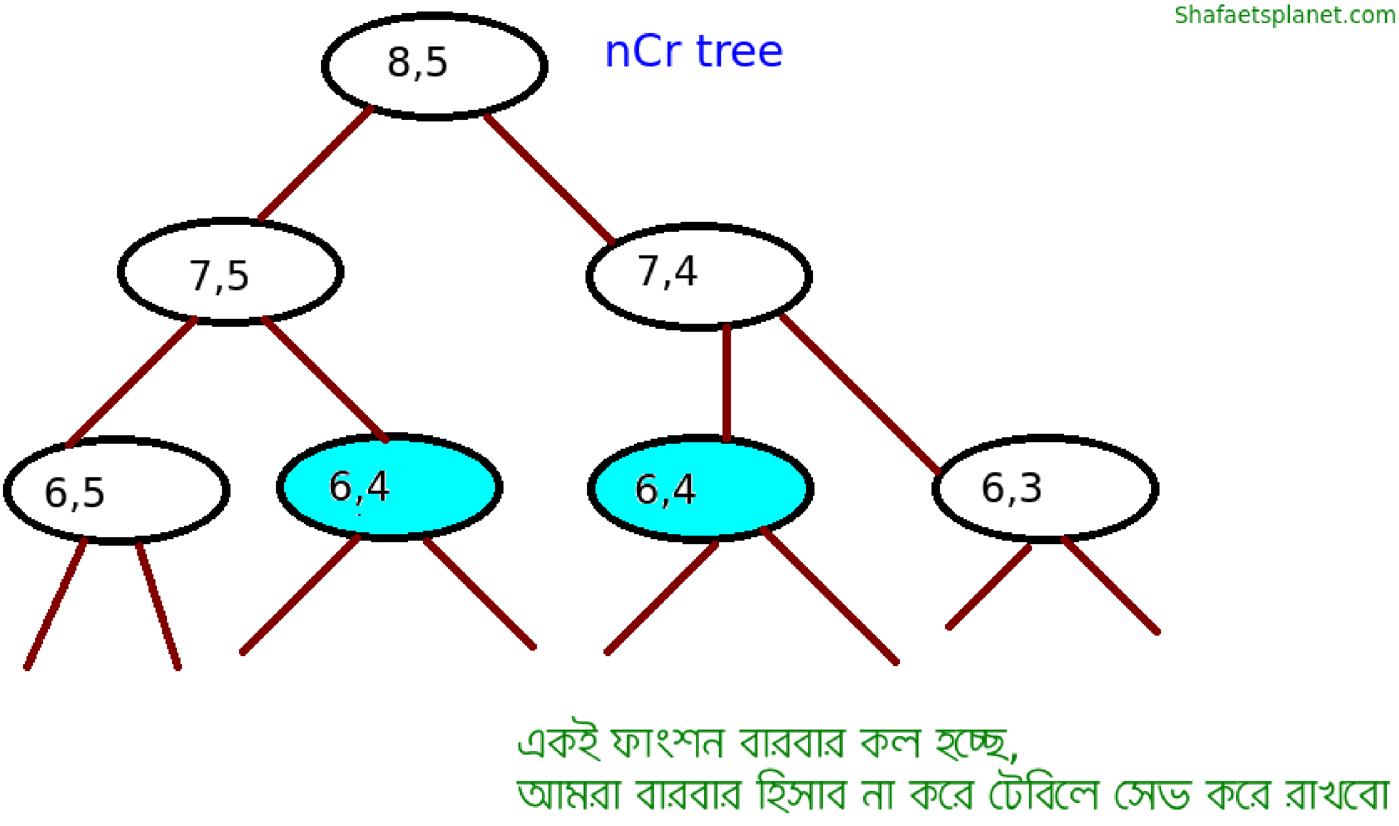
এবার হয়তো জিনিসটা আরো স্পষ্ট হয়েছে তোমার কাছে। কিন্তু রিকার্শনটা শেষ হবে কখন? অর্থাৎ বেস কেস কি? আমরা জানি nCr(n,1)=n এবং nCr(n,n)=1 । এ দুটি শর্ত বেস কেস হিসাবে ব্যবহার করলে কোডটা হবে:

C++

```cpp
#define i64 long long
i64 nCr(int n,int r)
{
    if(r==1) return n;
    if(n==r) return 1;
    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 -১ রেখে নিবো,-১ দিয়ে বুঝাচ্ছে ঘরটি খালি আছে। আগের লেখায় অনেকে initialisation অংশটা বুঝতে পারেনি তাই এবার পুরো কোডটা দিচ্ছি।

```
i64 dp[70][70];
i64 nCr(int n,int r)
{
    if(r==1) return n;
    if(n==r) return 1;
    if(dp[n][r]!=-1) return dp[n][r]; //ভ্যালু টেবিলে থাকলে নতুন করে হিসাব করা দরকার
নেই,ভ্যালুটা রিটার্ন করে দাও
        else{
        dp[n][r]=nCr(n-1,r)+nCr(n-1,r-1);         //ভ্যালু টেবিলে সেভ করে রাখো
        return dp[n][r];
    }


}
int main()
{
 //init dp table with -1
    for(int i=0; i< 70;i++)
        for(int j=0;j < 70;j++)
         dp[i][j]=-1;
          printf("%d\n",nCr(20,2));

}
```

---

view raw gistfile1.txt hosted with ❤ by GitHub

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

C++

```
int/double/long long call(int x1,int x2........,int xn)
{
    CHECK BASE CASES,RETURN IF REACHED ANY OF THE BASE CASES
    CHECK TABLE,RETURN IF VALUE IS ALREADY COMPUTED
    ELSE
    {
      COMPUTE VALUE RECURSIVELY AND SAVE IN A N-Dimensional TABLE/ARRAY
      RETURN VALUE
    }
}

```

nCr এর ক্ষেত্রে প্যারামিটার ছিলো n এবং r। বুঝতেই পারছো 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)। সম্ভবত 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।

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

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

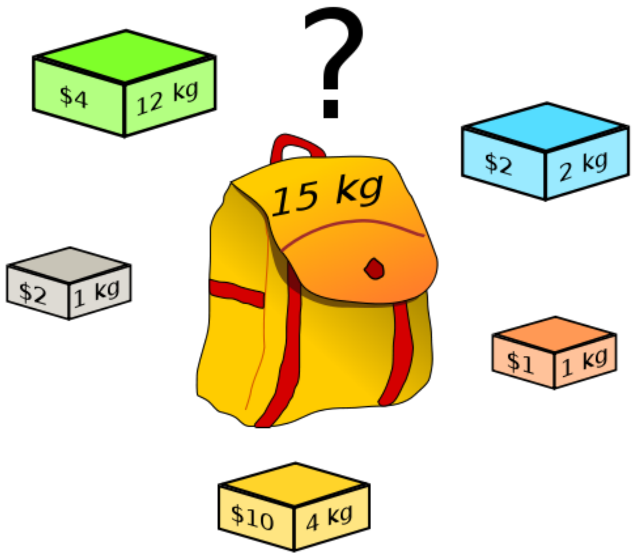
greedy অ্যালগোরিদমে অপটিমাল রেজাল্ট না পাওয়ায় তুমি চিন্তা করলে সবরকমের কম্বিনেশনে চেষ্টা করবে, প্রতিবার একটা করে জিনিস থলেতে ভরবে আর দেখবে আর কত লাভ করা যায়,প্রয়োজনে জিনিসটা থলে থেকে নামিয়ে রেখে আরেকটি নিয়ে চেষ্টা করবে। এবং সুবিধার জন্য তুমি লিস্টের সিরিয়াল অনুযায়ী জিনিস নিয়ে চেষ্টা করবে, ৩ নম্বর জিনিসের পরে ১ নম্বর জিনিস নিতে চেষ্টা করবেনা।

কোনো সময় তোমার অবস্থা বুঝাতে 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:

> *i=current_index*
> *w=weight taken*
> *int 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।

C++

```
//w+weight[i] হলো i তম জিনিস নেবার পর মোট ওজন,এটা কখনোই CAP এর বেশি হতে পারবেনা।
if(w+weight[i]<=CAP)
   profit1=cost[i]+func(i+1,w+weight[i])
else
   profit1=0; //নেয়া গেলোনা,নিলে ব্যাগ ছিড়ে যাবে

//cost[i] হলো i তম জিনিসটার দাম
//func(i+1,w+weight[i]) = আমরা i+1 তম জিনিস নিয়ে ট্রাই করবো
```

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

C++

```
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।

> *return max(profit1,profit2)*

তাহলে তুমি যদি main থেকে func(1,0) কল করো তাহলে রিকার্শন 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 রিকার্শন শেষ হবে,অর্থাৎ

> *n টা বস্তু থাকলে বেসকেস হবে if(i==n+1) return 0*

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

তোমার বোঝার সুবিধার জন্য সম্পূর্ণ একটা কোড দিয়ে দিলাম:

```
#define MAX_N 100
#define MAX_W 1000
int n;
int dp[MAX_N+1][MAX_W+1];
int weight[MAX_N+1];
int cost[MAX_N+1];
int CAP;
int func(int i,int w)
{
 if(i==n+1) return 0; //সব কিছু নেয়া হয়ে গেছে
 if(dp[i][w]!=-1) return dp[i][w]; //এই স্টেটটা আগেই হিসাব করে এসেছি
 int profit1=0,profit2=0;
 if(w+weight[i]<=CAP)
  profit1=cost[i]+func(i+1,w+weight[i]);//যদি i তম জিনিসটা নেয়া যায় তাহলে লাভের পরিমাণ profit1

 profit2=func(i+1,w); //যদি জিনিসটা না নেই তাহলে লাভ profit2
 dp[i][w]=max(profit1,profit2); //বেশি লাভ যেটায় হবে সেটাই আমরা নিবো
 return dp[i][w];
}
int main()
{

 freopen("in","r",stdin);
 memset(dp,-1,sizeof(dp));
 scanf("%d%d",&n,&CAP);
 for(int i=1;i<=n;i++)
 {
  scanf("%d%d\n",&weight[i],&cost[i]);
 }
 printf("%d\n",func(1,0));


}
```

view raw gistfile1.txt hosted with ❤ by GitHub

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

> *5 10 //৫ টা জিনিস, ১০ ক্যাপাসিটি*
> *1 120 //প্রতিটা জিনিসের ওজন এবং দাম*
> *7 400*
> *4 280*
> *3 150*
> *4 200*

একটু চিন্তা করলেই দেখবে আগের প্রবলেমগুলোর মতো এটাতেও একই ফাংশন বারবার কল হয়,তাই ২-ডি অ্যারেতে মানগুলো সেভ করতে হবে। অ্যারের সাইজ হবে [n][cap] যেখানে n=কয়টা জিনিস আছে, এই উদাহরণে n=5। তবে মেমরির ব্যবহার অনেক কমিয়ে আনা যায় কাজটা রিকার্শনে না করে iterative করলে,সেটা আমরা পরে দেখবো।

এটাই হলো ক্লাসিকাল 0-1 knapsack প্রবলেম,ন্যাপস্যাক শব্দটির অর্থ হলো থলে। ০-১ নাম দেয়ার কারণ হলো তুমি কোনো জিনিস অর্ধেক নিতে পারবেনা,হয় পুরোটা নিবে অথবা নিবেনা। fractional-knapsack বলে আরেক ধরণের 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। তাই আমি বেশিভাগ সলিউশন ১০০% না দিয়ে চিন্তা করার জন্য কিছু অংশ রেখে দিবো। তোমাকে অনুরোধ করবো পরবর্তি পর্ব পড়ার আগে অবশ্যই আগের পর্বের সমস্যাগুলো নিয়ে খুব ভালো করে অন্তত ১দিন চিন্তা করবে।

(সবগুলো পর্ব)
পরের পর্ব-কয়েন চেঞ্জ,রক ক্লাইম্বিং

পরিশিষ্ট:

$$^{n}C_{r} = ^{(n-1)}C_{r} + ^{(n-1)}C_{(r-1)}$$

এই সূত্রটি কেন কাজ করে বোঝা খুব সহজ। n টা জিনিসের মধ্যে r টা জিনিস কতভাবে নেয়া যায় সেটাই nCr বা nCr(n,r)। এখন যেকোন ১টা জিনিসের কথা চিন্তা কর। তুমি যদি জিনিসটা না নাও তাহলে বাকি n-1 টা জিনিসের মধ্যে 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 মধ্যে থেকে আরো r-1 টা জিনিস নিতে হবে, সেটা করা যায় nCr(n-1,r-1) ভাবে। এই দুইটার যোগফলই হলো মোট উপায়।