ডাইনামিক প্রোগ্রামিং এ হাতেখড়ি-৫

আশা করি তুমি এখন lis,knapsack,coin-change প্রবলেম সলভ করতে পারো খুব সহজেই, ডিপির সলিউশন প্রিন্ট করতেও তোমার সমস্য হয়না। এখন আমরা একটু অন্যরকম ডিপি দেখবো যেটার নাম বিটমাস্ক ডিপি। নামটা শুনে ভয় লাগলেও জিনিসটা সহজ, অনেক ক্ষেত্রেই বিটমাস্ক ডিপি প্রবলেম পড়ার সাথে সাথে সলিউশন মাথায় চলে আসে। তবে এই পর্বটা পড়ার আগে তোমাকে বিট নিয়ে কাজ করা শিখতে হবে, যেমন কোনো নির্দিষ্ট পজিশনের বিট অন করা/অফ করা ইত্যাদি। এজন্য তুমি এই চমৎকার টিউটোরিয়ালটা দেখতে পারো, পুরোটা খুবই ভালো করে পড়বে কারণ এটা তোমাদের অনেক জায়গায় কাজে লাগবে। আমি এই টিউটোরিয়ালে বিট অপারেশন নিয়ে লিখছিনা কারণ অপ্রাসঙ্গিক হয়ে যাবে।

আমরা শুরুতেই ৩টি ফাংশন ডিফাইন করি।

C++

```
int Set(int N,int pos){return N=N | (1<<pos);}
int reset(int N,int pos){return N= N & ~(1<<pos);}
bool check(int N,int pos){return (bool)(N & (1<<pos));}
```

Set ফাংশনটি N সংখ্যাটির pos তম পজিশনের বিট ১ করে দেয়, reset ফাংশনটি 0 করে দেয় এবং check ফাংশনটি pos তম বিটে কি আছে সেটা রিটার্ন করে। যেকোনো বিটমাস্ক ডিপি প্রবলেমে ফাংশন ৩টি দরকার হবে, বিশেষ করে Set এবং check।

আমরা একটা প্রবলেম দিয়ে শুরু করি। মনে করো তোমাকে nটা দোকান থেকে n টা জিনিস কিনতে হবে। জিনিসগুলো কিনতে তোমার a0,a1,a2,…,a(n-1) টাকা লাগে। তোমার শহরটা খুব অদ্ভুত,তুমি যখন একটা জিনিস কিনে আরেক দোকানে যাও তখন সেই দোকানদার তোমার আগের কেনা জিনিসগুলো দেখে তার দোকানের জিনিসের দাম বাড়িয়ে দেয়!! কত দাম বাড়াবে সেটা নির্ভর করবে তুমি আগে আগে কোন কোন দোকানে গিয়েছো সেটার উপর। ধরো n=2, তাহলে তোমাকে নিচের মতো একটা ম্যাট্রিক্স দেয়া থাকবে:

> *10 10*
> *90 10*

এখন,

> *যদি (i==j) হয় তাহলে matrix[i][j]=matrix[i][i]=i'তম জিনিসটির আসল দাম।*
> *যদি (i!=j) হয় তাহলে matrix[i][j]=j'তম জিনিসটি আগে কিনলে i'তম জিনিসটির সাথে যোগ হওয়া বাড়তি দাম।*

তুমি যদি শুরুতে ০ তম জিনিসটা কিনো তাহলে দাম পড়বে ১০টাকা, এরপর ১নম্বর জিনিসটা কিনলে সেটার দাম হবে ১০+৯০ টাকা, কারণ matrix[1][0]=০ নম্বর জিনিসের আগে ১ নম্বর জিনিস কিনলে যোগ হওয়া বাড়তি দাম=৯০ আর ১ নম্বর জিনিসের আসল দাম=১০, তাহলে মোট খরচ ১০+(১০+৯০)=১১০। কিন্তু তুমি যদি ১নম্বর জিনিসটা আগে কিনো তাহলে মোট খরচ ১০+(১০+১০)=৩০ টাকা।
বুঝতেই পারছো তোমার কাজ হলো খরচ মিনিমাইজ করা। n এর মান সর্বোচ্চ ১৫।

n এর মান খুব কম বলে বিটমাস্ক ডিপি দিয়ে প্রবলেমটি সহজেই সলভ করা যাবে। ডিপিতে আমাদের প্রথম কাজ হলো স্টেট নির্ণয় করা। এই কেনাকাটার যেকোনো সময় আমাদের অবস্থা কি কি তথ্য দিয়ে প্রকাশ করা যায়? "এখন পর্যন্ত কোন কোন জিনিস কেনা হয়েছে" এই তথ্যটাই যথেষ্ট, তাইনা? এটা জানলে আমরা পরবর্তি আরেকটি জিনিস কেনার সময় বাড়তি কত খরচ যোগ হবে জানতে পারবো, পরবর্তিতে যেই জিনিসটা কিনলে মোট খরচ কম হবে সেটা আমরা কিনবো। মনে করি এখন কথা হলো স্টেটটা রাখবো কি ভাবে?

একটা উপায় হলো n টি জিনিসের জন্য এভাবে nটা স্টেট রাখা function(a0,a1,a1,….an-1), কিন্তু n এর মান বদলালে তুমি প্যারামিটার সংখ্যা বদলাবে কি ভাবে? আর ১৫টি প্যারামিটার নিয়ে কাজ করলে কোডটা ভয়াবহ জটিল হয়ে যাবে।

২য় উপায় হলো বিটমাস্ক। একটি ইন্টিজারে ৩২টি বিট থাকে। আমরা সেই সুবিধাটাই নিবো। ১ নম্বর বিট 1 হলে আমরা ১ নম্বর জিনিসটা নিয়েছি, ০ হলে নেইনি। ৩ নম্বর বিট 1 হলে আমরা ৩ নম্বর জিনিসটা নিয়েছি, ০ হলে নেইনি। ১ এবং ৩ নম্বর বিট দুইটাই 1 হলে আমরা ২টা জিনিসই নিয়েছি।

শুরুতে আমাদের স্টেট থাকবে 0 বা বাইনারিতে "000000"। তারমানে আমরা কোনো জিনিস এখনও কিনিনি। ০তম এবং ১তম জিনিস কেনা হয়ে

গেলে স্টেট হবে 3 বা “000011″ এবং n=2 এর জন্য এটাই আমাদের base case। n=4 এর জন্য base case হলো 15 বা “0001111″। leading zero নিয়ে চিন্তা করা দরকার নেই, এটা বোঝার সুবিধার্থে দেয়া হয়েছে। একটু চিন্তা করলেই বুঝতে পারবে mask=(2^n)-1 হলে সেটা হবে base case কারণ তখন বাইনারিতে প্রথম n টা বিট ১ থাকবে, আমরা তখন শুন্য রিটার্ণ করে দিবো তখন কারণ আর কোনো জিনিস কেনা বাকি নেই।

C++

```
int dp[(1<<15)+2];
int call(int mask)
{
    if(mask==(1<<n)-1) return 0;
    if(dp[mask]!=-1) return dp[mask];
        //Rest of the calculation
}
```

বেসকেস বুঝলাম, এরপরে আমাদের কাজ হবে যেসব জিনিস কেনা হয়নি সেগুলা নিয়ে চেষ্টা করে দেখা। mask এর i তম পজিশনে যদি ০ থাকে তাহলে i তম জিনিসটি কেনা এখনও বাকি আছে।

C++

```
int dp[(1<<15)+2];
int call(int mask)
{
    if(mask==(1<<n)-1) return 0;
    if(dp[mask]!=-1) return dp[mask];
    //Rest of the calculation
    int ans=1<<28; //Infinite, a large value
    for(int i=0;i<n;i++)
    {
        if(check(mask,i)==0)
        {
             //Rest of the code
        }
    }
    return dp[mask]=ans;
}
```

আমরা n পর্যন্ত লুপ চালিয়ে বের করে নিলাম কোনটা কোনটা নেয়া বাকি আছে। এখন i তম জিনিসটার আসল দাম হলো price=w[i][i]। এই price এর সাথে w[i][j] যোগ হবে যদি i!=j হয় এবং j নম্বর জিনিসটা আগেই কেনা হয়ে থাকে। mask এর j তম বিট চেক করে আমরা বলতে পারি j তম জিনিসটা কেনা হয়েছে নাকি।

C++

```
int dp[(1<<14)+2];
int call(int mask)
{
    if(mask==(1<<n)-1) return 0;
    if(dp[mask]!=-1) return dp[mask];
    int ans=1<<28;
    for(int i=0;i<n;i++)
    {
        if(check(mask,i)==0)
        {
            int price=w[i][i];
            for(int j=0;j<n;j++)
              if(i!=j and check(mask,j)!=0)     price+=w[i][j];
            int ret=price+call(Set(mask,i));
            ans=min(ans,ret);

        }
    }
    return dp[mask]=ans;

}
```

---

j এর লুপটা দিয়ে আমরা মোট দাম বের করে নিলাম। এখন i তম জিনিসটি কিনলে পরবর্তি স্টেট কি হবে? শুধু mask এর i তম বিটটি 1 করে দিতে হবে। আমরা call(Set(mask,i)) এভাবে i তম জিনিস কিনে পরবর্তি স্টেটে চলে গেলাম। এভাবে প্রতিটি জিনিস কিনে যেটায় দাম মিনিমাম হয় সেটা রিটার্ন করে দিলাম। কাজ শেষ! সম্পূর্ণ কোড:

C++

```
int w[20][20];
int n;
int dp[(1<<15)+2];
int call(int mask)
{
    if(mask==(1<<n)-1) return 0;
    if(dp[mask]!=-1) return dp[mask];
    int mn=1<<28;
    for(int i=0;i<n;i++)
    {
        if(check(mask,i)==0)
        {
            int price=w[i][i];
            for(int j=0;j<n;j++)
            {
                if(i!=j and check(mask,j)!=0)
                {
                    price+=w[i][j];
                }
            }
            int ret=price+call(Set(mask,i));
            mn=min(mn,ret);

        }
    }
    return dp[mask]=mn;
}
int main()
{
        mem(dp,-1);
        cin>>n;
        for(int i=0;i<n;i++)
        {
            for(int j=0;j<n;j++)
            {
                scanf("%d",&w[i][j]);

            }
        }

        int ret=call(0);
        printf("%d\n",ret);


    return 0;
}
```

আমাদের call ফাংশনটি কয়টি ভিন্ন স্টেটে থাকতে পারে? n টি বিটের প্রতিটি হয় ০ হবে নাহয় ১ হবে, তাহলে স্টেট থাকতে পারে 2^15 টি। আর ভিতরে একটা n^2 লুপ চলছে তাই মোট complexity (2^n)*(n^2)।

বিটমাস্ক ডিপি চেনার সবথেকে সহজ উপায় n এর মান দেখা। n এর মান ১৬ বা তার কম হলে খুব ভালো সম্ভাবনা আছে যে প্রবলেমটিকে বিটমাস্ক ডিপি দিয়ে সলভ করা যাবে।

বিটমাস্ক ডিপি কখন ব্যবহার করবো? বিটমাস্ক লাগবে আমাদের তখনই যখন আগের স্টেটে কোন কোন জিনিস/ডিজিট/নোড ইত্যাদি ব্যবহার করা হয়েছে সে তথ্যটি আমার বর্তমান স্টেটে লাগবে। সেই তথ্য অনুযায়ী আমরা বর্তমান স্টেট থেকে নতুন ডিজিট/নোড ইত্যাদি নিবো এবং সেই বিটটি অন করে দিয়ে সামনের স্টেটে যাবো। যখন n টি বিট অন হয়ে যাবে তখন বেসকেস রিটার্ণ করে দিবো।

আমার সলভ করা প্রথম বিটমাস্ক ডিপি হলো uva 10651। আশা করি প্রবলেমটি এখন সহজেই করতে পারবে। প্রবলেমটায় একটি মাস্কের সাহায্যে কোনো সময় বোর্ডের কি অবস্থা সেই তথ্যটা রাখবে, এবং সে অবস্থায় যতগুলো চাল দেয়া সম্ভব সবগুলো দিয়ে মিনিমামটা রিটার্ণ করে দিবে

আপাতত এগুলোই ছিলো বিটমাস্ক ডিপির বেসিক। সামনের পর্বগুলোতে আরো বিস্তারিত আলোচনার চেষ্টা করবো। এখন নিচের প্রবলেমগুলো

সলভ করার চেষ্টা করো, ১ম প্রবলেমটি নিয়ে এই পর্বে আলোচনা করেছি:

Pimp My Ride
False Mirror
Agent 47
Painful Bases