Dynamic Programming

Knapsack

You are given n objects the ith object has weight Wi and a target weight K. You have to tell whether it is possible to achieve the target weight by taking a subset of objects?

n<=1000, K<=2000

// Brute Force

```
for(int i=0;i<(1 << n);i++)
{
      Int sum=0;
      for(int j=0;j<n;j++)
      {
            if(i&(1 << j))
            sum+=a[j];
      }
      if(sum==K)
      Return true;
}
Return false;
```

Bool Dp[n][k+1];

0000001

4
(000001)|(00010000)
00000010001

2
(000010001)|(0001000100)
00001010101

O(n*k)/32

memset(dp, false, sizeof(dp));
dp[0][0]=true;
Dp[i][j] → true → sum j is possible by taking a subset from first i elements

```
for(int i=1;i<=n;i++)
{
	for(int j=0;j<=k;k++)
	{
		if(dp[i-1][j]==false)
		continue;
		dp[i][j]=true;
		if(j+a[i]<=k)
		dp[i][j+a[i]]=true;
	}
}
```

dp[n][k]
O(n*k)

You are given n objects the ith object has weight Wi and a value Vi and a target weight K.
You have to tell the maximum value possible for any weight <=K

```
Int dp[n][k+1];
for(int i=0;i<=n;i++)
for(int j=0;j<=k;j++)
dp[i][j]=-INF;
dp[0][0]=0;
```

Dp[i][j] → true → maximum value possible for any subset with weight j from first i elements.

```
for(int i=1;i<=n;i++)
{
	for(int j=0;j<=k;k++)
	{
		if(dp[i-1][j]==-INF)
		continue;
		dp[i][j]=max(dp[i][j], dp[i-1][j]);
		if(j+a[i]<=k)
		dp[i][j+a[i]]=max(dp[i][j+a[i]], dp[i-1][j]+v[i]);

	}
}
```

12 23, 34 (x*k)+1
K+y (x*k)+y

```
Int ans=0;
for(int i=0;i<=k;i++)
ans=max(ans, dp[n][i]);
```

We are given n elements each having a maximum capacity and some value. You can take some fraction of this weight subsequently you will also get the same fraction of value. What is the maximum value we can get if the max total weight you can take is K?

n<=1000 k<=2000

You are given n objects the ith object has weight Wi and a value Vi such that the final sum of weights should be a multiple of K?

n<=1000 k<=2000 Wi, Vi<=1e9

n=4, (Wi, Vi), k=2

2, 5
8, 9
7, 11
2, 19
1, 1

0)0, 0, -> 0
  0, 1 -> -INF

1)1, 0 -> 5   (0+2)%2 -> 0 -> (+5)
  1, 1 -> -INF

2)2, 0 -> 14
  2, 1 -> -INF

3)3, 0 -> 14 (0+7)%2 -> 1 (14+11)
  3, 1 -> 25

4)4, 0 -> 33
   4, 1 -> 34

5) 5, 0 -> max(33, 44+1) -> 45
   5, 1 -> max(44, 33+1) -> 44

```
dp[0][0]=0;
for(int i=1;i<=n;i++)
{
        for(int j=0;j<k;k++)
        {
                if(dp[i-1][j]==-INF)
```

```
                continue;
                dp[i][j]=max(dp[i][j], dp[i-1][j]);
                dp[i][(j+a[i])%k]=max(dp[i][(j+a[i])%k], dp[i-1][j]+v[i]);

        }
}
```

You are given n objects the ith object has weight Wi and a value Vi and a target weight K.
You have to tell the maximum value possible for any weight <=K
n<=1e5, k<=1000
Sum of weights <=1e5

Partitions

Ordered Partitions

You are given a number N. Find the number of ordered partitions of N modulo m.

if n=4,
1+1+1+1
1+2+1
2+1+1
3+1
1+1+2
2+2
1+3
4

(i) O(N^2)

States - p[i] = number of partitions of i.
Transitions - p[i] = p[1]+p[2]+p[3]+...+p[i-1]+1 = pre[i-1]+1
Base Case - p[1]=1
Goal - p[N]

N=1	N=2	N=3	N=4
1	1+1 2	(1+1)+1 (2)+1 [1]+2 3	(1+1+1)+1 (1+2)+1 (2+1)+1 (3)+1 {1+1}+2 {2}+2 [1]+3 4

```
int p[N+1];
p[1]=1;
for(int i=2;i<=N;i++)
{
	p[i]=0;
	for(int j=1;j<i;j++)
		p[i]=(p[i]+p[j])%m;
	p[i]=(p[i]+1)%m;
}
cout << p[N];
```

(ii) O(N)

```
int p[N+1];
pre[N+1];
p[1]=1;
pre[1]=1;
for(int i=2;i<=N;i++)
{
	p[i]=(pre[i-1]+1)%m;
	pre[i]=(pre[i-1]+p[i])%m;
}
cout << p[N];
```

Unordered Partitions

You are given a number N. Find the number of unordered partitions of N modulo m.

If N=4,
1+1+1+1
1+1+2
2+2
1+3
4

(i) O(N^3)

States - dp[i][j] = Number of partitions of i such that maximum number used is j.
dp[4][1]=1
dp[4][2]=2
dp[4][3]=1
dp[4][4]=1
Transitions - dp[i][j] = dp[i-j][1]+dp[i-j][2]+dp[i-j][3]+...+dp[i-j][j]
Base Case - dp[i][i]=1, If j>i, dp[i][j]=0

```
int dp[N+1][N+1];
for(int j=2;j<=N;j++)
        dp[1][j]=0;
dp[1][1]=1;
for(int i=2;i<=N;i++)
{
        pre[i][0]=0;
        for(int j=1;j<=N;j++)
        {
                dp[i][j] = 0;
                for(int k=1;k<=j;k++)
                {
                        dp[i][j] = (dp[i][j]+dp[i-j][k])%m;
                }
                if(j==i)
                        dp[i][j]=1;
                if(j>i)
                        dp[i][j]=0;
        }
}
int ans=0;
for(int i=1;i<=N;i++)
        ans = (ans+dp[N][i])%m;
```

(ii) O(N^2)

States - dp[i][j] = Number of partitions of i such that maximum number used is <= j.
Transitions - dp[i][j] = dp[i][j-1]+dp[i-j][j];

1+1+1
1+2
3

1+1+1+1
1+1+2
2+2
1+3
4

1	1	1	1
1	2	2	2
1	2	3	3
1	3	4	5

If we use numbers < j-1 then these are < j also

dp[i][j] += dp[i][j-1]
dp[i][j] += dp[i-j][j]

Base Case - dp[i][0] = 0

```
int dp[N+1][N+1];
for(int i=1;i<=N;i++)
{
        dp[i][0]=0;
        for(int j=1;j<=N;j++)
        {
                dp[i][j]=dp[i][j-1];
                if(i-j>=1)
                {
                        dp[i][j] = (dp[i][j]+dp[i-j][j])%m;
                }
        }
}
cout << dp[n][n];
```


n U's, n R's

k U's, k R's

k+1 U's, k R's

(n-(k+1) U's, (n-k)R's

(n-k)U's (n-(k+1)) R's

(k+1 + (n-k)) U's (k+(n-(k+1))) R's

(n+1) U's (n-1) R's

(N-1, N+1)