Dynamic Programming：

1.背包问题：容量W，n个重量，价值分别为a[i]，v[i]的物品，求背包可装的最大价值。

01背包：

for(int i=0;i<n;i++){

for(int j=W;j>=a[i];j--){

dp[j]=max(dp[j],dp[j-a[i]]+v[i]);

}

}

完全背包：

for(int i=0;i<n;i++){

for(int j=a[i];j<=W;j++){

dp[j]=max(dp[j],dp[j-a[i]]+v[i]);

}

}

多重背包：

for(int i=0;i<n;i++){

int num=m[i];

for(int k=1;num>0;k<<=1){

int mul=min(k,num);

for(int j=W;j>=w[i]\*mul;j--){

dp[j]=max(dp[j],dp[j-a[i]\*mul]+v[i]\*num);

}

num-=mul;

}

}

dp[W]表示：所选物品总重量不超过W时的最大价值。

2.多重部分和：n种不同大小的数字a[i],每种m[i]个，判断是否可以用它们组成K。

int dp[N];

memset(dp,-1, sizeof(dp));

dp[0]=0;

for(int i=0;i<n;i++){

for(int j=0;j<=K;j++){

if(dp[j]>=0) dp[j]=m[i];

else if(j<a[i]||dp[j-a[i]]<=0) dp[j]=-1;

else dp[j]=dp[j-a[i]]-1;

}

}

dp[K]<0 代表：不可行，反之可行。

3.多重部分和改版：判断有多少种组成K的方式。

int dp[N];

memset(dp,0, sizeof(dp));

dp[0]=1;

for(int i=0;i<n;i++){

for(int j=K;j>=a[i];j--){

for(int k=1;k<=m[i];k++){

if(j>=k\*a[i]) dp[j]+=dp[j-k\*a[i]];

}

}

}

memset(dp,0, sizeof(dp));//多重背包版

dp[0]=1;

for(int i=0;i<n;i++){

int num=m[i];

for(int k=1;num>0;k<<=1){

int mul=min(k,num);

for(int j=K;j>=a[i]\*mul;j--){

dp[j]=(dp[j]+dp[j-a[i]\*mul])%mod;

}

num-=mul;

}

}

dp[K] 表示组成K的种数。

3.无限个:

int dp[N];

memset(dp,0, sizeof(dp));

dp[0]=1;

for(int i=0;i<n;i++){

for(int j=a[i];j<=K;j++){

dp[j]+=dp[j-a[i]];

}

}

3.最长子列：

LCS：

char s[N],t[N];

int dp[N][N];

for(int i=0;i<n;i++){

for(int j=0;j<m;j++){

if(s[i]==t[j]) dp[i+1][j+1]=dp[i][j]+1;

else dp[i+1][j+1]=max(dp[i][j+1],dp[i+1][j]);

}

}

dp[n][m]即为LCS长度。

LIS：

int a[N],dp[N],ans=0;

for(int i=0;i<n;i++){

dp[i]=1;

for(int j=0;j<i;j++){

if(a[j]<a[i]) dp[i]=max(dp[i],dp[j]+1);

}

ans=max(ans,dp[i]);

}

ans即为LIS长度。

4.计数问题：

划分数：将n个相同的球放进m个相同的箱子中，求方案数。

int dp[N];

memset(dp,0, sizeof(dp));

dp[n][1]=1,dp[n][n]=1;

for(int i=1;i<=n;i++){

for(int j=1;j<=m;j++) {

if (i>=j) dp[i][j]=dp[i-1][j-1]+dp[i-j][j];

}

}

dp[n][m]为:n的m划分数。

多重集组合数：有n种不同的物品每种a[i]个，从中取出m个，求方案数。

int dp[N][N];

for(int i=0;i<=n;i++) {

dp[i][0]=1;

}

for(int i=0;i<n;i++){

for(int j=1;j<=m;j++){

if(j-1-a[i]>=0) dp[i+1][j]=(dp[i+1][j-1]+dp[i][j]-dp[i][j-1-a[i]]+MOD)%MOD;

else dp[i+1][j]=(dp[i+1][j-1]+dp[i][j])%MOD;

}

}

dp[n][m]:所求的方案数。