一.子序列问题

1.最长公共子序列(LCS)

O(mn)

```
#define N 1005
ll a[N]={0},b[N]={0},dp[N][N]={0};
void solve()
{
    ll m,n;cin >> n >> m;
    FORLL(i,1,n) cin >> a[i];
    FORLL(i,1,m) cin >> b[i];
    FORLL(i,1,n)
        FORLL(j,1,m){
            if(a[i]==b[j]) dp[i][j]=dp[i-1][j-1]+1;
            else dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        }
    cout << dp[n][m] << endl;
}
```

2.最长上升子序列

2.1.DP

O(n^2) 洛谷B3637

```
/*----------Code Area----------*/
#define N 10005
ll a[N]={0},dp[N]={0};
void solve()
{
    ll n;cin >> n;
    FORLL(i,1,n) cin >> a[i];
    dp[1]=1;ll ans=0;
    FORLL(i,2,n){
        ll mx=0;
        FORLL(j,1,i-1)
            if(a[i]>a[j]) mx=max(mx,dp[j]);
        dp[i]=mx+1;
        ans=max(ans,dp[i]);
    }
    cout << ans << endl;
}
```

2.2.贪心

O(nlogn) 洛谷B3637 贪心：维护当前子序列d，替换序列中不小于a[i]的第一个元素

```
#define N 10005
ll a[N]={0};
vector<ll> d;
void solve()
{
    ll n;cin >> n;
    FORLL(i,1,n) cin >> a[i];
    d.emplace_back(a[1]);
    FORLL(i,2,n){
        if(d.back()<a[i]) d.emplace_back(a[i]);
        else *lower_bound(ALL(d),a[i])=a[i];
    }//最长不降子序列改成upper_bound
    cout << d.size() << endl;
}
```

二.背包DP

1.01背包

1.1.DFS记忆化搜索

O(mn) 洛谷P1048

```
#define N 1005
#define W 105
ll v[N]={0},w[N]={0};
ll mem[W][N]={0};
ll maxw,n;
ll dfs(ll i,ll curw){
    if(mem[i][curw]) return mem[i][curw];
    if(i>n) return mem[i][curw]=0;
    if(curw>=w[i])
        return mem[i][curw]=max(dfs(i+1,curw-w[i])+v[i],dfs(i+1,curw));
    else
        return mem[i][curw]=dfs(i+1,curw);
}
void solve()
{
    cin >> maxw >> n;
    FORLL(i,1,n) cin >> w[i] >> v[i];
    cout << dfs(1,maxw) << endl;
}
```

1.2.二维数组

O(wn) M(wn) 洛谷P1048

```
/*----------Code Area----------*/
#define N 1005
#define W 105
ll v[N]={0},w[N]={0};
ll dp[N][W]={0};
ll n,maxw;
void solve()
{
    cin >> maxw >> n;
    FORLL(i,1,n) cin >> w[i] >> v[i];
    FORLL(i,1,n)
        FORLL(j,0,maxw){
            if(j>=w[i]) dp[i][j]=max(dp[i-1][j-w[i]]+v[i],dp[i-1][j]);
            else dp[i][j]=dp[i-1][j];
        }
    cout << dp[n][maxw] << endl;
}
```

1.3.一维滚动数组

O(wn) M(w) 洛谷P2871

```
#define N 5005
#define W 20005
ll v[N]={0},w[N]={0};
ll dp[W]={0};
ll n,maxw;
void solve()
{
    cin >> n >> maxw;
    FORLL(i,1,n) cin >> w[i] >> v[i];
    FORLL(i,1,n)
        FORLL_rev(j,maxw,w[i]){//01背包从大到小遍历，每个物品只能取一次
            dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
        }
    cout << dp[maxw] << endl;
}
```

2.完全背包

2.1.一维滚动数组

O(wn) M(w) 洛谷P1616 和01背包唯一区别在剩余容量从小到大遍历，每个物品能取多次

```
#define N 10005
#define W 10000005
ll v[N]={0},w[N]={0};
ll dp[W]={0};
ll n,maxw;
void solve()
{
    cin >> maxw >> n;
    FORLL(i,1,n) cin >> w[i] >> v[i];
    FORLL(i,1,n)
        FORLL(j,w[i],maxw){//和01背包唯一区别在从小到大遍历，每个物品能取多次
            dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
        }
    cout << dp[maxw] << endl;
}
```

2.2.贪心优化

O(wn) M(w) 洛谷P1616 贪心思想：对于两件物品 i,j ，如果 $w_i \le w_j \ \& \ v_i \ge v_j$ ，则只需保留 i

```
#define N 10005
#define W 10000005
vector<pll> objs,tv;
ll dp[W]={0};
ll n,maxw,w,v;
void solve()
{
    cin >> maxw >> n;
    tv.resize(n);//物品：<weight,value>
    for(auto &x:tv) cin >> x.first >> x.second;
    SORT(tv);ll maxv=-1;//物品序列按主w次v排序
    for(auto x:tv)
        if(x.second>maxv)//O(n)筛去多余物品
            {objs.emplace_back(x);maxv=x.second;}
    for(auto x:objs){
        w=x.first;v=x.second;
        FORLL(j,w,maxw){//和01背包唯一区别在从小到大遍历，每个物品能取多次
            dp[j]=max(dp[j-w]+v,dp[j]);
        }
    }
    cout << dp[maxw] << endl;
}
```

3.多重背包

3.1.朴素方法

$O(w\sum cnt_i)$ 朴素方法：按有 cnt_i 个的物品 i ，进行01背包

```cpp
#define N 100005
#define W 100005
ll v[N]={0},w[N]={0};
ll dp[W]={0};
ll n=0,maxw,tn;
void solve()
{
    cin >> tn >> maxw;
    ll tv,tw,cnt,n=0;
    FORLL(i,1,tn){
        cin >> tv >> tw >> cnt;
        FORLL(j,1,cnt){
            v[++n]=tv;
            w[n]=tw;
        }
    }
    FORLL(i,1,n)
        FORLL_rev(j,maxw,w[i])
            dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
    cout << dp[maxw] << endl;
}
```

3.2.二进制分组

$O(w\sum \lg cnt_i)$ 二进制分组优化：对于每个物品，将其按二进制分组，捆绑一个物品

```cpp
#define N 100005
#define W 100005
ll v[N]={0},w[N]={0};
ll dp[W]={0};
ll n=0,maxw,tn;
void solve()
{
    cin >> tn >> maxw;
    ll tv,tw,cnt,n=0;
    FORLL(i,1,tn){
        cin >> tv >> tw >> cnt;
        ll b=1;
        while(cnt>b){
            v[++n]=tv*b;
            w[n]=tw*b;
            cnt-=b;b*=2;
        }
        if(cnt) {v[++n]=tv*cnt;w[n]=tw*cnt;}
    }
    FORLL(i,1,n)
        FORLL_rev(j,maxw,w[i])
            dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
    cout << dp[maxw] << endl;
}
```

4.混合背包

$O(wn)$ 洛谷P1833 01背包、多重背包和完全背包的缝合怪//

```
#define N 100005
#define W 100005
ll v[N]={0},w[N]={0};
int cnt[N]={0};
ll dp[W]={0};
ll n=0,maxw,tn;
void solve()
{
    ll h1,m1,h2,m2;char tc;
    cin >> h1 >> tc >> m1 >> h2 >> tc >> m2 >> tn;
    maxw=abs((h2*60+m2)-(h1*60+m1));
    ll tv,tw,tcnt,n=0;
    FORLL(i,1,tn){
        cin >> tw >> tv >> tcnt;
        if(tcnt){
            ll b=1;
            while(tcnt>b){
                v[++n]=tv*b;
                w[n]=tw*b;
                cnt[n]=b;
                tcnt-=b;b*=2;
            }//多重背包二进制分组
            if(tcnt) {v[++n]=tv*tcnt;w[n]=tw*tcnt;cnt[n]=1;}
        }else{
            v[++n]=tv;
            w[n]=tw;
            cnt[n]=0;
        }
    }
    FORLL(i,1,n)
        if(cnt[i]){//01背包
            FORLL_rev(j,maxw,w[i])
                dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
        }else{//完全背包
            FORLL(j,w[i],maxw)
                dp[j]=max(dp[j-w[i]]+v[i],dp[j]);
        }
    cout << dp[maxw] << endl;
}
```

5.二维费用背包

$O(nw_1w_2)$ 洛谷P1855 具有两种费用属性的背包问题，以01背包为例

```
#define N 105
#define W 205
ll v[N]={0},w1[N]={0},w2[N]={0};
ll dp[W][W]={0};//二维滚动数组
ll n,maxw1,maxw2;
void solve()
{

    cin >> n >> maxw1 >> maxw2;
    FORLL(i,1,n){
        cin >> w1[i] >> w2[i];
        v[i]=1;//本题中价值均为1
    }
    FORLL(i,1,n)
        FORLL_rev(j1,maxw1,w1[i])
        FORLL_rev(j2,maxw2,w2[i])
            dp[j1][j2]=max(dp[j1-w1[i]][j2-w2[i]]+v[i],dp[j1][j2]);
    cout << dp[maxw1][maxw2] << endl;
}
```

6.分组背包

$O(nw)$ 洛谷P1757 01背包的进化体，每个组中最多能取1个物品

```
#define N 65536
ll maxw,n;
ll v[105][N]={0},w[105][N]={0},dp[1000*N]={0};
ll cnt[105]={0},mxid=0;
void solve()
{
    cin >> maxw >> n;
    ll tv,tw,id;
    FORLL(i,1,n){
        cin >> tw >> tv >> id;
        cnt[id]++;
        v[id][cnt[id]]=tv;
        w[id][cnt[id]]=tw;
        mxid=max(mxid,id);
    }
    FORLL(k,1,mxid)//对于每一组物品
        FORLL_rev(j,maxw,0)//对于每一种容量（从后往前遍历，保证同组两种物品不同时取到）
            FORLL(i,1,cnt[k]) if(j>=w[k][i])//放入该组每种物品时可达到的最大值
                dp[j]=max(dp[j-w[k][i]]+v[k][i],dp[j]);
    cout << dp[maxw] << endl;
}
```

7.有依赖的背包

$O(nw)$ 洛谷P1064 分组背包的进化体，将所有主副件组合方案作为一组进行分组背包

```
#define W 32005
#define N 61
ll maxw,n;
ll w[N][4]={0},v[N][4]={0},dp[W]={0},cnt[N]={0};
vector<ll> id_list;
void solve()
{
    cin >> maxw >> n;
    ll tw,tp,id;
    FORLL(i,1,n){
        cin >> tw >> tp >> id;
        if(id==0){//0：纯主件
            id=i;w[id][0]=tw;v[id][0]=tw*tp;
            id_list.emplace_back(i);
        }else if(cnt[id]==0){//1：主件+配件1
            cnt[id]++;
            w[id][1]=tw+w[id][0];
            v[id][1]=tw*tp+v[id][0];
        }else if(cnt[id]==1){//2：主件+配件2//3：主件+配件1+配件2
            cnt[id]++;
            w[id][2]=tw+w[id][0];
            v[id][2]=tw*tp+v[id][0];
            w[id][3]=tw+w[id][1];
            v[id][3]=tw*tp+v[id][1];
        }
    }
    for(auto k:id_list){//对于每一组物品
        ll t=0; if(cnt[k]==1) t=1; if(cnt[k]==2) t=3;
        FORLL_rev(j,maxw,0)//对于每一种容量（从后往前遍历，保证同组两种方案不同时取到）
            FORLL(i,0,t) if(j>=w[k][i])//采取该组每种方案时可达到的最大值
                dp[j]=max(dp[j-w[k][i]]+v[k][i],dp[j]);
    }
    cout << dp[maxw] << endl;
}
```

8.进阶问题

1.求具体方案

$O(wn)$ 以完全背包为例，在转移时记录容量j下选择的物品编号

```
#define N 10005
#define W 10000005
ll v[N]={0},w[N]={0},cnt[N]={0};
ll dp[W]={0},g[W]={0};
ll n,maxw;
void solve()
{
    cin >> maxw >> n;
    FORLL(i,1,n) cin >> w[i] >> v[i];
    FORLL(i,1,n)
        FORLL(j,w[i],maxw){//和01背包唯一区别在从小到大遍历，每个物品能取多次
            if(dp[j-w[i]]+v[i]>dp[j]){
                dp[j]=dp[j-w[i]]+v[i];
                g[j]=i;
            }
        }
    cout << dp[maxw] << endl;
    ll curw=maxw;
    while(g[curw]){
        cnt[g[curw]]++;
        curw-=w[g[curw]];
    }
    FORLL(i,1,n) ;//输出方案
}
```

2.装满方案计数

$O(wn)$ 对于给定的一个背包容量、物品费用、其他关系等的问题，求装到一定容量的方案总数。以01背包为例

```
#define N 5005
#define W 20005
ll v[N]={0},w[N]={0};
ll dp[W]={1,0};//dp[0]即不选，方案数为1
ll n,maxw;
void solve()
{
    cin >> n >> maxw;
    FORLL(i,1,n) cin >> w[i] >> v[i];
    FORLL(i,1,n)
        FORLL_rev(j,maxw,w[i]){//01背包从大到小遍历，每个物品只能取一次
            dp[j]=dp[j]+dp[j-w[i]];
        }
    cout << dp[maxw] << endl;
}
```

3.最优方案计数

$O(wn)$ 求最优背包方案数，以01背包为例

```
#define N 5005
#define W 20005
ll v[N]={0},w[N]={0};
ll dp[W]={0},g[W]={1,0};//g[0]即不选，方案数为1
ll n,maxw;
void solve()
{
    cin >> n >> maxw;
    FORLL(i,1,n) cin >> w[i] >> v[i];
    FORLL(i,1,n)
        FORLL_rev(j,maxw,w[i])//01背包从大到小遍历，每个物品只能取一次
            if(dp[j-w[i]]+v[i]>dp[j]){//取 最优
                dp[j]=dp[j-w[i]]+v[i];
                g[j]=g[j-w[i]];
            }else if(dp[j-w[i]]+v[i]==dp[j]) g[j]+=g[j-w[i]]; //取或不取都最优
    cout << dp[maxw] << ' ' << g[maxw] << endl;
}
```

4.求第k优解

$O(wnk)$ HDU2639 求背包的第k优解，以01背包为例

```
#define N 105
#define W 1005
void solve()
{
    ll v[N]={0},w[N]={0};
    ll dp[W][35]={0};//dp[j][k]表示容量j下第k优解
    ll a[35]={0},b[35]={0};
    ll n,maxw,tark;
    ll x,y,z;
    cin >> n >> maxw >> tark;
    a[tark+1]=b[tark+1]=-1;//哨兵
    FORLL(i,1,n) cin >> v[i];
    FORLL(i,1,n) cin >> w[i];
    FORLL(i,1,n) //考虑前i个物品时
        FORLL_rev(j,maxw,w[i]){ //容量为j下
            FORLL(k,1,tark){
                a[k]=dp[j-w[i]][k]+v[i];//第k优解，选
                b[k]=dp[j][k];//第k优解，不选
            }//dp数组不升保证a,b数组不升
            x=y=z=1;
            while(z<=tark&&!(a[x]==-1&&b[y]==-1)){//循环直到找到全部前k解，或两指针都到末尾
                if(a[x]>b[y]) dp[j][z]=a[x++];
                else dp[j][z]=b[y++];
                if(dp[j][z]!=dp[j][z-1]) z++;//非严格比较则删去if条件
            }
        }
    cout << dp[maxw][tark] << endl;
}
```

三.区间DP

四.DP优化

1. 单调队列优化DP

O(mn) M(m) CF372C 利用单调队列将每次区间DP均摊复杂度降至O(1)

```
#define N 305
#define M 150005
ll a[N]={0},b[N]={0},t[N]={0};
ll dp[2][M]={0};
ll n,m,d;int fl=1;
ll que[M]={0};
void solve()
{
    cin >> n >> m >> d;
    FORLL(i,1,m) cin >> a[i] >> b[i] >> t[i];
    fl=1;
    FORLL(i,1,m){
        ll l=1,r=0,k=1;
        FORLL(j,1,n){
            for(;k<=min(n,j+(t[i]-t[i-1])*d);k++){
                while(r>=l&&dp[fl^1][que[r]]<dp[fl^1][k]) r--;
                que[++r]=k;//单调队列优化DP：维护上一状态的有效区间内的最大值的下标
            }
            while(r>=l&&que[l]<max(1ll,j-(t[i]-t[i-1])*d)) l++;
            dp[fl][j]=dp[fl^1][que[l]]+b[i]-abs(a[i]-j);
        }fl^=1;//状态转换
    }fl^=1;//回到最终状态
    ll ans=-INF;
    FORLL(i,1,n) ans=max(ans,dp[fl][i]);
    cout << ans << endl;
}
```

Reference：OI-Wiki