# 背包九讲（二）

# 完全背包模型



有 $N$ 种物品和一个容量为 $V$ 的背包，每种物品都有无限件可用。

放入第 $i$ 种物品 的费用是 $C_i$，价值是 $W_i$。

求解：将哪些物品装入背包，可使这些物品的耗费的费用总 和不超过背包容量，且价值总和最大。

# 分析问题

回忆01背包

一个直接的想法就是把每个物品多复制几份

时间复杂度较大

考虑空间优化后的01背包问题

为什么要倒着循环?

```
for(int i=1;i<=n;i++){
    for(int j=V;j>=w[i];j--){//倒着转移
        dp[j]=max(dp[j],dp[j-w[i]]+c[i]);//选与不选
    }
}
cout<<dp[V]<<endl;
```

**为了避免某个物品被多次选择**

# 分析问题



- 01背包倒着循环是为了不重复选择某个物品
- 如果某个物品可以选择多次呢?
- 只需要将倒着循环的代码改为正着循环

```
for(int i=1;i<=n;i++){
    for(int j=w[i];j<=V;j++){//正着转移
        dp[j]=max(dp[j],dp[j-w[i]]+c[i]);//选与不选
    }
}
cout<<dp[V]<<endl;
```

# 多重背包问题



有 $N$ 种物品和一个容量为 $V$ 的背包，每种物品都有$M_i$件可用。

放入第 $i$ 种物品 的费用是 $C_i$，价值是 $W_i$。

求解：将哪些物品装入背包，可使这些物品的耗费的费用总 和不超过背包容量，且价值总和最大。

# 分析问题



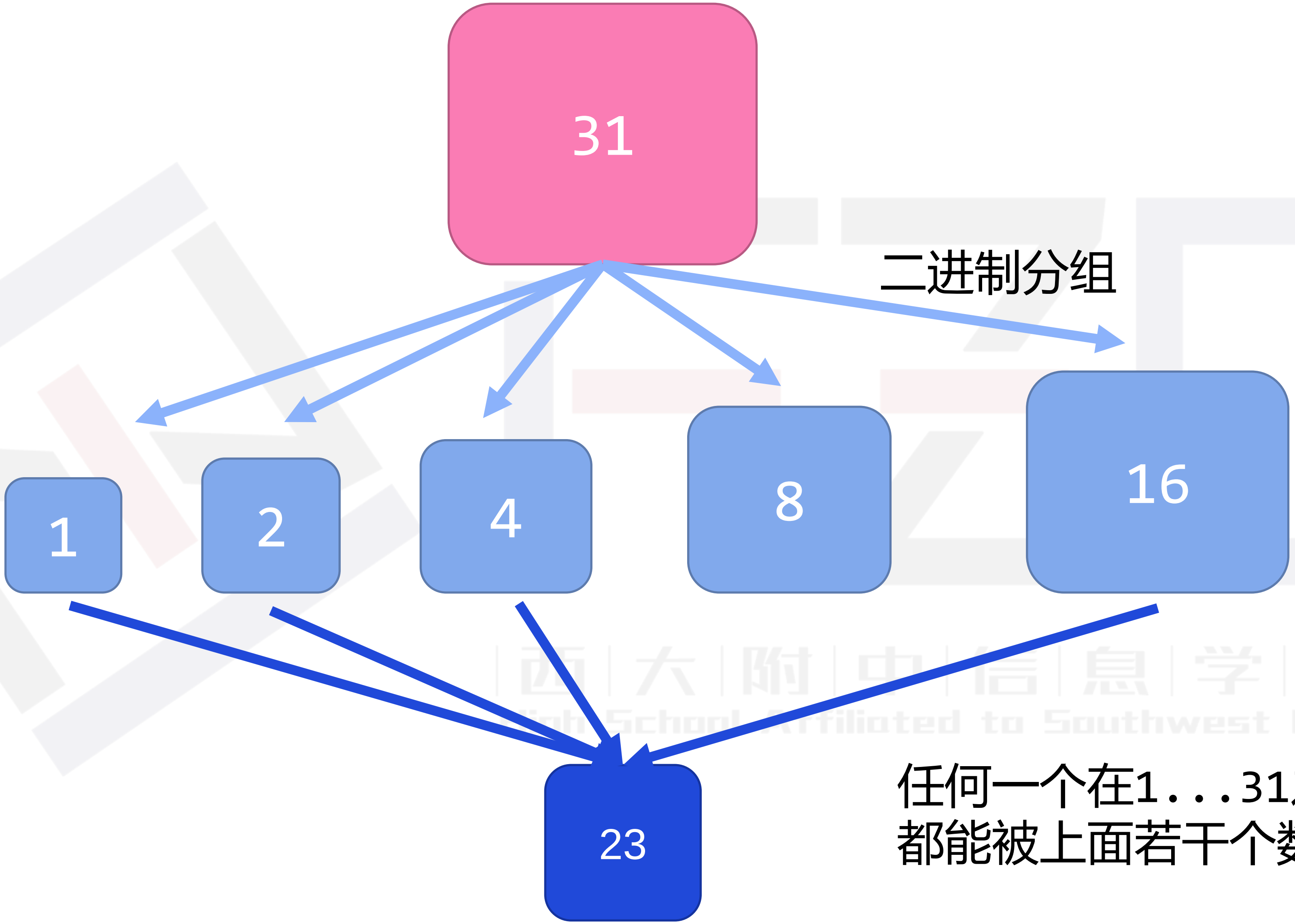
- 回忆01背包问题
- 此处每种物品有$M_i$个
- 如果把每种物品拆散，就相当于$\sum M_i$个物品的01背包
- 时间复杂度有点大：$O(V\sum M_i)$

```
for(int i=1;i<=n;i++){
    for(int k=1;k<=M[i];k++)//第i物品有M[i]个
        for(int j=V;j>=w[i];j--){//倒着转移
            dp[j]=max(dp[j],dp[j-w[i]]+c[i]);//选与不选
        }
}
cout<<dp[V]<<endl;
```

# 二进制分组

- 对于每种物品，我们直接将其完全拆散，拆成$M_i$个
- 这样不管是从中选取1个、2个、3个...都能满足
- 上述方法的缺点是，每种物品被拆成了$M_i$份，时间复杂度增加
- 考虑优化
- 是否有某种分组的方式，使得分出的组数变少
- 并且也能够保证不管从中选取1个、2个、3个...都能满足
- 二进制分组

# 二进制分组

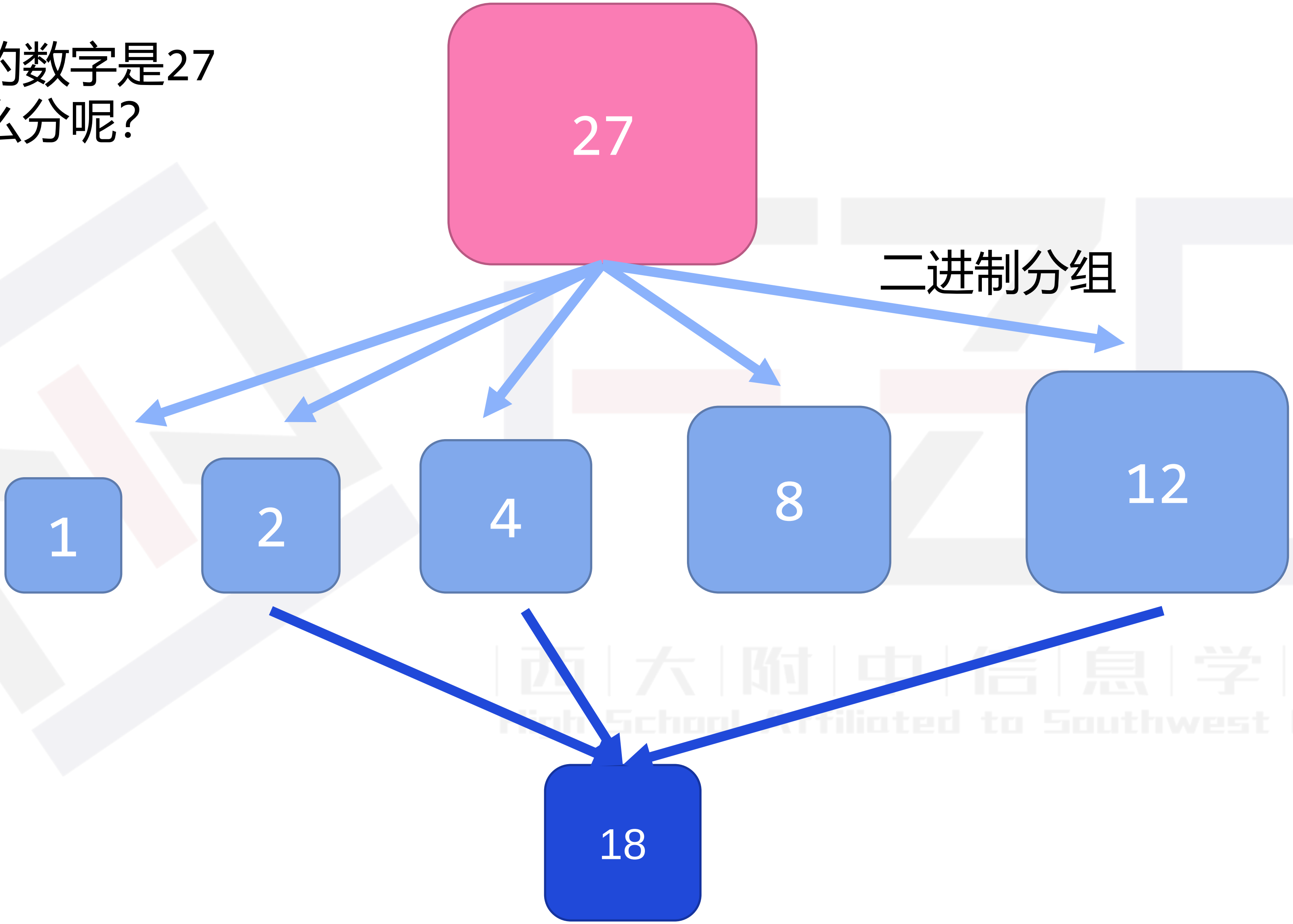



任何一个在1...31之间的数字
都能被上面若干个数字拼凑出来

# 二进制分组



如果待的数字是27
应该怎么分呢?



结合二进制分组
如何代码实现?

```
for(int i=1;i<=n;i++){
    for(int k=1;k<=M[i];k*=2)//第i物品有M[i]个
        for(int j=V;j>=k*w[i];j--){//倒着转移
            dp[j]=max(dp[j],dp[j-k*w[i]]+k*c[i]);//选与不选
        }
}
cout<<dp[V]<<endl;
```

对吗?

# 多重背包问题

西南大学附属中学
High School Affiliated to Southwest University

```
for(int i=1;i<=n;i++){
    int tmp=M[i];
    int k=1;
    while(tmp>0){//二进制分组
        k=min(k,tmp);
        tmp-=k;
        for(int j=V;j>=k*w[i];j--){//倒着转移
            dp[j]=max(dp[j],dp[j-k*w[i]]+k*c[i]);//选与不选
        }
        k*=2;
    }
}
```

有 $N$ 种物品和一个容量为 $V$ 的背包

有些物品有$M_i$件可用

有些物品有无穷多件可用

有些物品只有1件可用

放入第 $i$ 种物品 的费用是 $C_i$，价值是 $W_i$。

求解：将哪些物品装入背包，可使这些物品的耗费的费用总 和不超过背包容量，且价值总和最大。

# 混合背包问题



还是先枚举每种物品

如果当前物品只有1件可用，选择01背包的方式进行转移

如果当前物品有$M_i$件可用，使用多重背包进行转移

如果当前物品有无穷多可用，选择完全背包的方式进行转移

(其实01背包可以看作多重背包的一个特例)

```
for(int i=1;i<=n;i++){
    if(){//只有一个物品
        //01背包转移
    }
    else if(){//有m[i]个物品
        //多重背包转移
    }
    else{//有无穷多个物品
        //完全背包转移
    }
}
```

有 $N$ 组物品和一个容量为 $V$ 的背包，每组物品中有$M_i$种物品。

每组物品中，最多只能选择其中一个物品放入背包

放入第 $i$ 组第 $j$ 个物品 的费用是 $C_{ij}$，价值是 $W_{ij}$。

求解：将哪些物品装入背包，可使这些物品的耗费的费用总和不超过背包容量，且价值总和最大。

# 分析问题

- 回顾01背包问题
- 设计状态，dp[i][j]表示考虑前i组物品，背包容量为j的情况下，最大价值
- 假如第i组只有一个物品
  - 按照01背包的方式转移，通过dp[i-1][0...j]转移得到dp[i][j]
- 假如第i组有两个物品
  - 对第一个物品按照01背包方式转移得到dp1[i][j]
  - 对第二个物品也重新按照01背包方式转移得到dp2[i][j]
  - 于是dp[i][j]=max(dp1[i][j],dp2[i][j])
- 依次类推

# 参考代码



```
for(int i=1;i<=n;i++){//n组
    for(int j=V;j>=0;j--){//背包容量枚举
        for(int k=1;k<=M[i];k++){//第i组的第k个物品
            if(j>=c[i][k])
                dp[j]=max(dp[j],dp[j-c[i][k]]+w[i][k]);
        }
    }
}
```

分组的背包问题将彼此互斥的若干物品称为一个组，这建立了一个很好的模型。

不少背包问题的变形都可以转化为分组的背包问题

# 依赖背包



背包容量为w，有n件物品，每件物品的体积是ci，价值是vi，问能装下 的最大价值是多少？

## 输入

物品种类数n （<=10000），背包容量 W(<=1000)
每行三个数 f w v表示它的主件为编号为f的物品，价值为v，体积为w，输入顺序即为编号
如果f为0，说明本身就是主件，保证附件没有附件；

## 输出

最大价值

## 样例

### 样例输入1

```
6 10
0 2 3
0 2 4
5 1 5
0 3 6
0 5 4
0 4 5
```

### 样例输出1

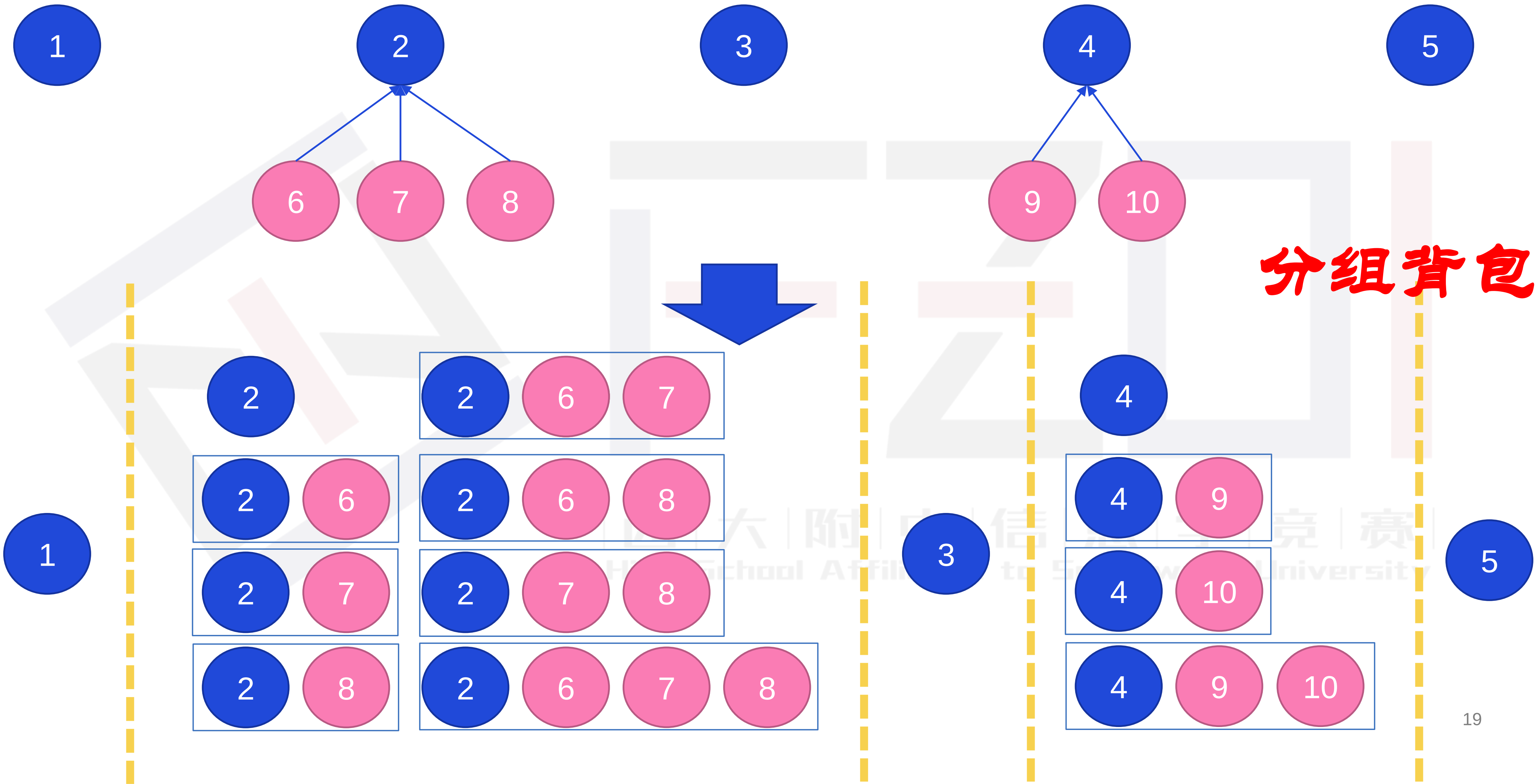
```
16
```

# 分析问题


选了对应的蓝色物品，才能去选择剩下的粉红色物品

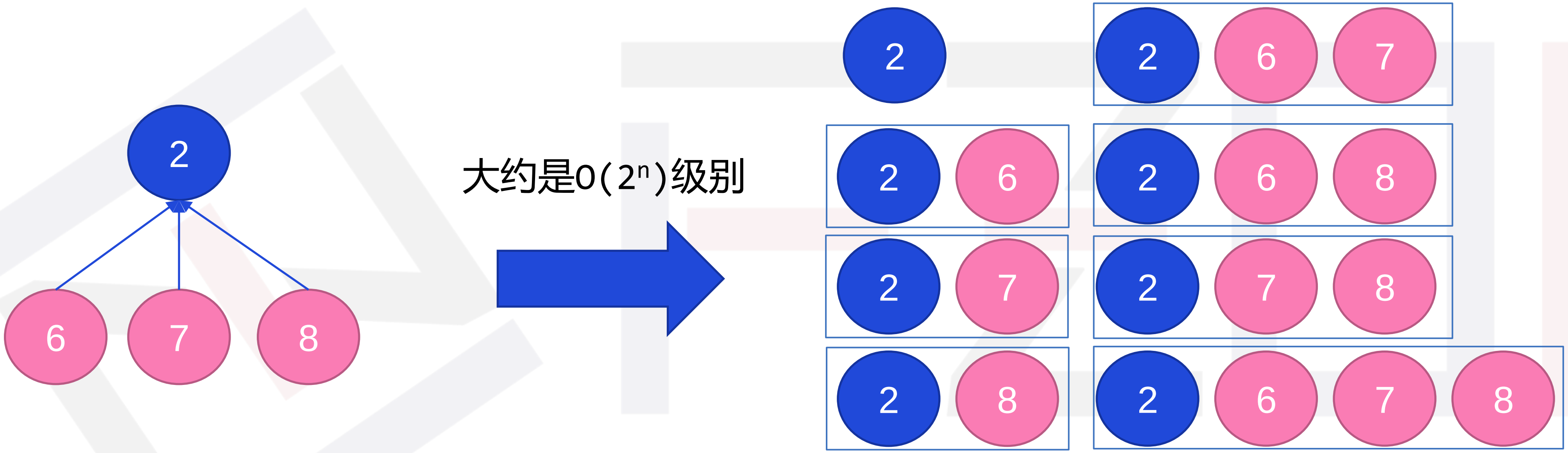
# 分析问题





# 分析问题

每组内部分出来的物品数量太多




考虑优化。

# 分析问题

- 可以发现分解出的$O(2^n)$中有很多是重复的
- 比如重量为x，价值为y的可能有多个
- 其实只需要保留一个
- 进一步还可以发现，重量为x，价值不同的物品中，只需要保留最大的那一个
- 因为背包容量为V，只需要保留重量在0...V之间的物品
- 这不就是想要求
- 把这n个物品看作01背包的n个物品，求出它对应的dp[0...V]数组吗?

# 参考代码

```cpp
//f: 是否是主件
//w:重量 v:价值
struct node{
    int w,v,f;
}bag[10010];
int n,m;
//f:附件的价值
//f1:整体的价值
int s1[10010],s2[10010];
int main(){
    cin>>n>>m;
    for(int i=1;i<=n;i++)
        cin>>bag[i].f>>bag[i].v>>bag[i].w;
    for(int i=1;i<=n;i++){//枚举的是主件
        if(bag[i].f==0){//这是一个主件

            memset(s2,0,sizeof(s2));
            for(int j=1;j<=n;j++){
                if(bag[j].f==i){//如果它的主件是所枚举的
                    //对附件进行一次01背包
                    for(int l=m;l>=bag[j].v;l--){
                        s2[l]=max(s2[l],s2[l-bag[j].v]+bag[j].w);
                        //记录附件的价值
                    }
                }
            }
            //再对整体做一次分组背包
            for(int j=m;j>=bag[i].v;j--){
                for(int l=0;l<=j-bag[i].v;l++){
                    s1[j]=max(s1[j],s1[j-bag[i].v-l]+bag[i].w+s2[l]);
                }
            }
        }
    }
    cout<<s1[m]<<endl;
    return 0;
}
```

FZOI

西大附中信息学竞赛

High School Affilinted to Southwest University