算法设计与分析 (5.20 作业)

智科三班　严中圣　222020335220177

2022 年 6 月 8 日

Algorithm 1 $0-1$ *Knapsack problem —— Brute Force*

Require: $n, c, V[n], W[n]$

Ensure: $X[n]$ 最终存储状态;$MaxValue$ 最大价值

$Initialize\ X[n], S[n]$ //最终存储状态，当前存储状态

$Initialize\ CurrentValue, CurrentVolume$ //当前价值，当前消耗容量

$MaxValue \leftarrow Force(1)$

return $X[n], MaxValue$

Algorithm 2 $Force(i)$

if $i > n$ **then**

    if $MaxValue < CurrentValue \quad and \quad CurrentVolumn \leq c$ **then**

        for $k \leftarrow 1$ **to** n **do**

            $X[k] \leftarrow S[k]$ //存储最优路径

        end for

        $MaxValue \leftarrow CurrentValue$

    end if

    return $MaxValue$

end if

$CurrentVolume \leftarrow CurrentVolume + V[i]$

$CurrentValue \leftarrow CurrentValue + W[i]$

$S[i] = 1$

$Force(i+1)$

$CurrentVolume \leftarrow CurrentVolume - V[i]$

$CurrentValue \leftarrow CurrentValue - W[i]$

$S[i] \leftarrow 0$

$Force(i+1)$

return $MaxValue$

Algorithm 3 $0-1$ *Knapsack problem —— Dynamic processing*

Require: $n, c, V[n], W[n]$

Ensure: $X[n]$ 最终存储状态;$MaxValue$ 最大价值

$Initialize\ X[n]$ //最终存储状态，当前存储状态

$Initialize\ V[n+1][c]$ //dp 数组，初始化数组为 0

for $k \leftarrow 1$ **to** c **do**

  if $k > V[1]$ **then**

    $V[1][k] \leftarrow W[1]$

  end if

end for

for $i \leftarrow 2$ **to** n **do**

  for $j \leftarrow 1$ **to** c **do**

    if $j < V[i-1]$ **then**

      $V[i][j] \leftarrow V[i-1][j]$

    else

      $V[i][j] \leftarrow \max(V[i-1][j], V[i-1][j-V[i-1]] + W[i-1])$

    end if

  end for

end for

$j \leftarrow c$

for $i \leftarrow n$ **to** 1 **do**

  if $i == 1$ **then**

    if $V[i][j] > 0$ **then**

      $X[1] \leftarrow 1$

    else

      $X[1] \leftarrow 0$

    end if

  end if

  if $V[i][j] > V[i-1][j]$ **then**

    $X[i] \leftarrow 1$

    $j \leftarrow j - V[i]$

  else

    $X[i] \leftarrow 0$

  end if

end for

return $X[n], V[n][c]$

Algorithm 4 $0-1$ $Knapsack\ \ problem$ —— $BackTracking\ Method$

Require: $n, c, V[n], W[n]$

Ensure: $X[n]$ 最终存储状态;$MaxValue$ 最大价值

$Initialize\ X[n], S[n]$ //最终存储状态，当前存储状态
$Initialize\ CurrentValue, CurrentVolume$ //当前价值，当前消耗容量
$MaxValue \leftarrow BackTrack(1)$
return $X[n], MaxValue$

Algorithm 5 $BackTrack(i)$

if $i > n$ **then**
  if $MaxValue < CurrentValue$ **then**
    for $k \leftarrow 1$ **to** n **do**
      $X[k] \leftarrow S[k]$ //存储最优路径
    end for
    $MaxValue \leftarrow CurrentValue$
  end if
  return $MaxValue$
end if
if $CurrentVolume + V[i] \leq c$ **then**
  $CurrentVolume \leftarrow CurrentVolume + V[i]$
  $CurrentValue \leftarrow CurrentValue + W[i]$
  $S[i] \leftarrow 1$
  $BackTrack(i + 1)$
  $CurrentVolume \leftarrow CurrentVolume - V[i]$
  $CurrentValue \leftarrow CurrentValue - W[i]$
end if
$S[i] \leftarrow 0$
$BackTrack(i + 1)$
return $MaxValue$

Algorithm 6 $0-1$ *Knapsack problem —— Branch and Bound*

Require: $n, c, V[n], W[n]$

Ensure: $X[n]$ 最终存储状态;$MaxValue$ 最大价值

$Initialize\ X[n], S[n]$ //最终存储状态，当前存储状态
$Initialize\ CurrentValue, CurrentVolume$ //当前价值，当前消耗容量
$MaxValue \leftarrow BackTrack(1)$
return $X[n], MaxValue$

Algorithm 7 $BackTrack(i)$

if $i > n$ **then**
  if $MaxValue < CurrentValue$ **then**
    for $k \leftarrow 1$ **to** n **do**
      $X[k] \leftarrow S[k]$ //存储最优路径
    end for
    $MaxValue \leftarrow CurrentValue$
  end if
  return
end if
if $CurrentVolume + V[i] \leq c$ **then**
  $CurrentVolume \leftarrow CurrentVolume + V[i]$
  $CurrentValue \leftarrow CurrentValue + W[i]$
  $BackTrack(i+1)$
  $CurrentVolume \leftarrow CurrentVolume - V[i]$
  $CurrentValue \leftarrow CurrentValue - W[i]$
end if
if $Bound(i+1) > MaxValue$ **then**
  $S[i] \leftarrow 0$
  $BackTrack(i+1)$
end if

Algorithm 8 $Bound(i)$

$Initialize\ RemainValue$ //剩余最大价值
while $k \leq n$ **do**
  $RemainValue \leftarrow RemainValue + W[k]$
  $k \leftarrow k + 1$
end while
return $RemainValue + CurrentValue$