

*“Heaven’s Light is Our Guide”*

# Department of Computer Science & Engineering

**RAJSHAHI UNIVERSITY OF ENGINEERING & TECHNOLOGY**

**Lab Report**

**Course No:** CSE 2202

**Course Name:** Sessional Based on CSE 2201

# Submitted to:

**Biprodip Pal**

Assistant Professor

Department of Computer Science & Engineering

# Submitted by:

**Md. Al Siam**

Department of Computer Science & Engineering Session: 2016-2017

Section: A

**Roll No.: 1603008**

|  |
| --- |
| **Problem** |

Given some weights with a value for each. Find the maximum summation of value considering a maximum summation of the picked weights. (0/1 Knapsac Problem)

Make a table considering solution for each stage dynamically. From this, derive the final solution.

|  |
| --- |
| **Solution in Table Making Approach** |

#include <bits/stdc++.h>

using namespace std;

int wt[] = {1, 3, 4, 5};

int val[] = {1, 4, 5, 7};

int dp[1000][1000];

int main(){

int n = sizeof(wt) / sizeof(wt[0]);

int max\_wt = 7;

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

for(int j = 0; j <= max\_wt; j++){

if(i == 0 || j == 0) dp[i][j] = 0;

else if(j < wt[i-1]) dp[i][j] = dp[i-1][j];

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

}

}

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

for(int j = 0; j <= max\_wt; j++)

printf((j == max\_wt) ? "%d\n" : "%d ", dp[i][j]);

printf("Maximum value can be obtained: %d\n", dp[n][max\_wt]);

}

|  |
| --- |
| **Output** |

0 0 0 0 0 0 0 0

0 1 1 1 1 1 1 1

0 1 1 4 5 5 5 5

0 1 1 4 5 6 6 9

0 1 1 4 5 7 8 9

Maximum value can be obtained: 9

|  |
| --- |
| **Solution in Recursive Approach** |

#include <bits/stdc++.h>

using namespace std;

int wt[] = {1, 3, 4, 5};

int val[] = {1, 4, 5, 7};

int n = sizeof(wt) / sizeof(wt[0]);

int max\_wt = 7;

int dp[1000][1000];

int call(int index, int curr\_total){

if(dp[index][curr\_total] != -1) return dp[index][curr\_total];

if(index == n) return 0;

if(curr\_total == 0) return 0;

int ret1;

int ret2;

ret1 = ret2 = 0;

if(curr\_total-wt[index] >= 0){

ret1 = val[index] + call(index+1, curr\_total-wt[index]);

}

ret2 = call(index+1, curr\_total);

return dp[index][curr\_total] = max(ret1, ret2);

}

int main(){

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

int ans = call(0, max\_wt);

printf("Maximum value can be obtained: %d\n", ans);

}

|  |
| --- |
| **Output** |

Maximum value can be obtained: 9

|  |
| --- |
| **Problem** |

Given a set of numbers. Find if it is possible make a number summing up any subset of the given set. (Subset Sum Problem)

|  |
| --- |
| **Solution in Table Making Approach** |

#include <bits/stdc++.h>

using namespace std;

int num[] = {9, 6, 3, 7};

bool dp[1000][1000];

int main(){

int n = sizeof(num) / sizeof(num[0]);

int target;

while(scanf("%d", &target) == 1){

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

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

if(i == 0 && j == 0) dp[i][j] = true;

else if(i == 0) dp[i][j] = false;

else if(j == 0) dp[i][j] = true;

else if(j < num[i-1]) dp[i][j] = dp[i-1][j];

else dp[i][j] = dp[i-1][j-num[i-1]] | dp[i-1][j];

}

}

if(dp[n][target])

printf("Possible to make %d\n", target);

else printf("Not possible to make %d\n", target);

}

}

|  |
| --- |
| **Output** |

10

Possible to make 10

11

Not possible to make 11

15

Possible to make 15

16

Possible to make 16

17

Not possible to make 17

|  |
| --- |
| **Solution in Recursive Approach** |

#include <bits/stdc++.h>

using namespace std;

int num[] = {9, 6, 3, 7};

int n = sizeof(num) / sizeof(num[0]);

int target;

int dp[1000][1000];

bool call(int index, int curr\_sum){

if(dp[index][curr\_sum] != -1) return dp[index][curr\_sum];

if(curr\_sum == target) return true;

if(index == n) return false;

bool ret1;

bool ret2;

ret2 = ret1 = false;

ret1 = call(index+1, curr\_sum+num[index]);

ret2 = call(index+1, curr\_sum);

return dp[index][curr\_sum] = ret1 | ret2;

}

int main(){

while(scanf("%d", &target) == 1){

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

if(call(0, 0))

printf("Possible to make %d\n", target);

else printf("Not possible to make %d\n", target);

}

}

|  |
| --- |
| **Output** |

10

Possible to make 10

11

Not possible to make 11

15

Possible to make 15

16

Possible to make 16

17

Not possible to make 17

|  |
| --- |
| **Comment** |

Both 0/1 knapsac problem and subset problem is basically activity selection problem. The idea is to take an element from the set or avoid the element from the set to make subset that fulfills the required condition. There can be 2n subset of a set containing n elements, that requires a huge amount of time to compute. By optimizing the solution step by step storing the solution of the subproblem in a table, we can reduce computation time. The table can also be used to avoid computing overlapping subproblems.